გოჩა ჩოგოვაძე, არჩილ ფრანგიშვილი,
გია სურგულაძე

მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი

„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

გოჩა ჩოგოვაძე, არჩილ ფრანგიშვილი,
გია სურგულაძე

მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი

დამტკიცებულია მონოგრაფიად
საქართველოს ტექნიკური უნივერსიტეტის
სარედაქციო-საგამომცემლო საბჭოს
მიერ. 27.12.2016, ოქმი №3

თბილისი
2017

უაკ 004.5

მონოგრაფიაში განხილულია მართვის საინფორმაციო სისტემების პროგრამული უზრუნველყოფის ობიექტორიენტირებული ანალიზის, დაპროექტების, დეველოპინგის, ტესტირების, დანერგვისა და რეინჟინერინგის საკითხები, UML/Agile/ITIL მეთოდოლოგიების, ბიზნესპროცესების IT-სერვისების უსაფრთხოების BSI/COBIT სტანდარტების, მონაცემთა რელაციური და NoSQL/NewSQL ტიპის ბაზების მართვის სისტემების გამოყენების საფუძველზე. გადმოცემულია აგრეთვე დაპროექტების ავტომატიზაციის, კლიენტსერვერული და სერვისორიენტირებული სისტემების იმიტაციური მოდელების აგების და ანალიზის საკითხები პეტრის ფერადი ქსელებით (CPN). ექსპერიმენტული ნაწილი შესრულებულია უნივერსიტეტის, საფინანსო ბანკის, მულტიმოდალური გადაზიდვების სისტემის, ელექტრონული არჩევნების, შავი ზღვის ეკოლოგიის მონიტორინგის სისტემის და საწარმოო ფირმის მარკეტინგის საპრობლემო სფეროებისათვის. პრაქტიკული ამოცანები რეალიზებულია დაპროგრამების ჰიბრიდული ტექნოლოგიების (WPF,WF,WCF) გამოყენებით MsVisual Studio.NET Framework 4.5 ინტეგრირებულ პლატფორმაზე. წიგნი გამიზნულია პროგრამული ინჟინერიისა და მართვის საინფორმაციო სისტემების სპეციალობათა ბაკალავრების, მაგისტრანტ-დოქტორანტებისა და პროგრამული უზრუნველყოფის და მონაცემთა მენეჯმენტის საკითხებით დაინტერესებული მკითხველისათვის.

რეცენზენტები: საქართველოს მეცნიერებათა ეროვნული აკადემიის
წევრ-კორესპონდენტი, ტექნიკის მეცნიერებათა დოქტორი,
ინფორმატიკისა და მართვის სისტემების ფაკულტეტის
პროფესორი ***გიორგი გოგიჩაიშვილი***,

ფიზიკა-მათემატიკის მეცნიერებათა დოქტორი,
პროფესორი ***ჰამლეტ მელაძე***

ტექნიკის მეცნიერებათა დოქტორი, ინფორმატიკისა და მართვის
სისტემების ფაკულტეტის პროფესორი ***რომან სამხარაძე***

© საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი", 2017
ISBN 978-9941-20-790-7

http://www.gtu.ge

ყველა უფლება დაცულია, ამ წიგნის არც ერთი ნაწილი (იქნება ეს ტექსტი, ფოტო, ილუსტრაცია თუ სხვ.) არანაირი ფორმით და საშუალებით (ელექტრონული თუ მექანიკური), არ შეიძლება გამოყენებულ იქნას გამომცემლის წერილობითი ნებართვის გარეშე. საავტორო უფლებების დარღვევა ისჯება კანონით.

Georgian Technical University

Gocha Chogovadze, Archil Prangishvili,
Gia Surguladze

HYBRID SOFTWARE TECHNOLOGIES AND DATA ENGINEERING FOR MANAGEMENT INFORMATION SYSTEMS

The present book discusses topics of object-oriented analysis, design, development, testing, implementation and reengineering for software of management information systems based on using UML/Agile/ITIL methodologies, business process IT Service security standards BSI/COBIT, data relation and NoSQL/NewSQL type database management systems. In the present book, topics such as automation of design, designing simulation models for client-server and service oriented systems using Petri Colored Networks (CPN). Experimental part has been carried out for problem areas of a University, commercial bank, multimodal transportation system, electronic elections, Black Sea Ecology monitoring system and a manufacturing business. Practical problems have been carried out using hybrid technologies of programming (WPF, WF, WCF) under platform of MsVisual Studio.NET Framework 4.5. This book is intended for bachelor, master, doctoral students of Management Information Systems as well as readers interested in software engineering and data management topics.

© Publication House "Technical University", Tbilisi, 2017

ISBN 978-9941-20-790-7

http://www.gtu.ge

All rights reserved. No part of this publication may be reproduced (will this be a text, photo, illustration or otherwise) in any form or by any means (electronic or mechanical) without the prior written permission of publisher. Piracy is punished according to the law.

ეძღვნება:

საქართველოს ტექნიკური უნივერსიტეტის ***„მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერიის)“*** კათედრის დაარსების
45-ე წლისთავს
(1971-2016)

Dedicated to:

To the 45 Anniversary of the Department „MANAGEMENT INFORMATION SYSTEMS (SOFTWARE ENGINEERING)” of Georgian Technical University

(1971-2016)

სარჩევი

შესავალი 15
I ნაწილი. ბიზნესპროცესების მართვის საინფორმაციო სისტემები და
პროგრამული ინჟინერია 17
I თავი. დაპროგრამების ტექნოლოგიები, პლატფორმები და ენები 18
1.1. პროგრამული ტექნოლოგიების განვითარების ისტორია........................... 18
1.2. დაპროგრამების თანამედროვე პლატფორმები და ენები 23
1.3. NET პლატფორმის ტიპების ზოგადი სისტემა 28
1.4. დაპროგრამების მეთოდები, სტილი, მოდელი და ენა 30
1.5. დაპროგრამების პარადიგმები 34
II თავი. Windows- და Web-დაპროგრამების ჰიბრიდული
ტექნოლოგიები 37
2.1. Windows Presentation Foundation ტექნოლოგია 38
2.2. Workflow Foundation ტექნოლოგია 39
2.3. Windows Communication Foundation ტექნოლოგია 40
2.4. XAML ენის საფუძვლები 45
III თავი. Ms Visual Studio.NET Framework ინტეგრირებული გარემო,
აპლიკაციების სტრუქტურა და პროცესების მართვა 52
3.1. .NET პლატფორმის კომპონენტები და მისი აპლიკაციების სტრუქტურა.... 52
3.2. .NET პლატფორმის აპლიკაციები 54
3.3. ობიექტორიენტირებული დაპროგრამების მეთოდი: კლასები და
ობიექტები 56
IV თავი. Visual C#.NET ენა - აპლიკაციების აგების ვიზუალური
ინსტრუმენტული საშუალება 61
4.1. შესავალი: Visual C# 2015-ის სამუშაო გარემო და ენის ელემენტები 61
4.2. აპლიკაციების აგება დაპროგრამების რამდენიმე ენის საფუძველზე .dll
ფაილების შექმნით 64
4.3. კლასების და ობიექტების დაპროგრამების ამოცანა 76
4.4. რეფაქტორინგი: კოდის დამუშავება და რეორგანიზაცია 88
4.5. პროგრამული აპლიკაციების ტესტირება 92
V თავი. გამოყენებითი პროგრამული სისტემების მენეჯმენტი 101
5.1 პროექტების მენეჯმენტი 101
5.2. პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის მოდელები 104
5.3. განტერის მოდელი „ფაზები და ფუნქციები“ 111
5.4. ობიექტორიენტირებული დაპროგრამების პრინციპები 115

II ნაწილი. ბიზნესის მართვის საინფორმაციო სისტემების უსაფრთხოების საერთაშორისო სტანდარტები და აგების ტექნოლოგიები 125
VI თავი. BSI და ინფორმაციული უსაფრთხოების ISO სტანდარტები 126
6.1. BSI და ინფორმაციული უსაფრთხოება 128
6.2. რა არის ინფორმაციული უსაფრთხოება 135
6.3. ISO სტანდარტები ინფორმაციული უსაფრთხოებისათვის 130
6.4. BSI სტანდარტები ინფორმაციული უსაფრთხოებისათვის 132
6.5. სხვა სტანდარტები: ITIL და COBIT 133
6.6. ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემის კომპონენტები... 134
6.7. სასიცოცხლო ციკლი ინფორმაციულ უსაფრთხოებაში 136
6.8. ინფორმაციული უსაფრთხოების პროცესის აღწერა 137
6.9. რესურსები ინფორმაციული უსაფრთხოებისათვის 139
6.10. უსაფრთხოების პოლიტიკა და რისკები 140
6.11. უსაფრთხოების კონცეფციის დანერგვა 144
VII თავი. ინფორმაციული ტექნოლოგიების ინფრასტრუქტურის ბიბლიოთეკა (ITIL) 146
7.1. შესავალი ITIL-ში: ძირითადი ცნებები და ტერმინები 146
7.2. IT-სერვისის სასიცოცხლო ციკლი 152
7.3. სერვისების სტრატეგიის აგება 156
7.4. სერვისების მიმწოდებელთა ტიპები 159
7.5. ოთხი „P“ სტრატეგიის ასაგებად 164
7.6. ფინანსების მართვა 168
7.7. სერვისების დაპროექტება, როგორც სერვისების სასიცოცხლო ციკლის ეტაპი 175
7.8. სერვისების დაპროექტების ასპექტები 183
7.9. ტექნოლოგიების არქიტექტურის დაპროექტება 184
7.10. პროცესების დაპროექტება 186
7.11. სერვისების უწყვეტობის მართვა და რისკების შეფასება 188
7.12. ინფორმაციული უსაფრთხოების მართვა 200
7.13. სერვისების დანერგვა 208
VIII თავი. COBIT სტანდარტები 212
8.1. COBIT-ტექნოლოგია და ძირითადი ტერმინები 212
8.2. COBIT-ის მიზნები და ფუნქციები 213
8.3. COBIT-ის პროცესები 214
IX თავი. პროგრამული აპლიკაციების ინტეგრაციის თანამედროვე ტექნოლოგიები 217

9.1. მართვის საინფორმაციო სისტემებში ინტეგრაციისა და მონაცემთა დამუშავება-გადაცემის ტექნოლოგიები 217
9.2. სერვისორიენტირებული არქიტექტურის რეალიზაციის საშუალება Ms BizTalk Server 223
9.3. კორპორაციათა ბიზნესპროცესების ინტეგრაციის ამოცანა სერვისორიენ-ტირებული სისტემების ასაგებად 229
9.4. ბიზნესპროცესების რეალიზაციის ინსტრუმენტული საშუალებანი: JavaNetBeans, BPEL 230
III ნაწილი. პროგრამული ინჟინერია UML და Agile ტექნოლოგიებით 233
X თავი. უნიფიცირებული მოდელირების ენის კონცეფცია და განვითარება... 234
10.1. პროგრამული ინჟინერია UML ტექნოლოგიის ბაზაზე 234
10.2. UML დიაგრამები 240
10.2.1. Use Case დიაგრამა 240
10.2.2. Acrivity Case დიაგრამა 241
10.2.3. ინტერაქტიული (Sequence და Collaboration) დიაგრამები........ 243
10.2.4. კლასების (Class) დიაგრამა 244
10.2.5. Clas-Assotiation დიაგრამები 246
10.2.6. მდგომარეობათა (Statechaert) დიაგრამა 248
10.2.7. კომპონენტების (Component) დიაგრამა 249
10.2.8. განთავსების (Deployment) დიაგრამა 249
10.3. კლასების დიაგრამიდან პროგრამული კოდის გენერაცია 250
XI თავი. სისტემის მოთხოვნილების განსაზღვრა და ობიექტორიენტირებული ანალიზი 257
11.1. ინფორმაციული რესურსები და ბიზნესპროცესები 257
11.2. სისტემის ბიზნესმოთხოვნილების განსაზღვრა 260
11.3. ობიექტორიენტირებული ანალიზის ეტაპი და ინტერაქტიული დიაგრამების აგება 265
11.3.1. მიმდევრობითობის (Sequence) დიაგრამები 265
11.3.2. თანამოქმედების (Collaboration) დიაგრამები 271
XII თავი. ობიექტორიენტირებული დაპროექტება 273
12.1. კლასთა ასოციაციის დიაგრამა 273
12.2. მდგომარეობათა (Statechart) დიაგრამები 284
12.2.1. მულტიმოდალური გადაზიდვის სტანდარტული Statechart მოდელი... 287
12.2.2. არასტანდარტული Statechart მოდელი 290
XIII თავი. მართვის საინფორმაციო სისტემების აგება Agile ტექნოლოგიით.... 294
13.1. ექსტრემალური დაპროგრამების პრინციპები და ინსტრუმენტული საშუალებანი 294

13.2. მოქნილი (Agile) მოდელირება და მისი ფასეულობანი 296
13.3. Scrum - მოქნილი მეთოდის მაგალითი 300
13.4. UML/Agile მეთოდოლოგიების კომპრომისული გამოყენება....................... 304

IV ნაწილი. მართვის საინფორმაციო სისტემების დაპროგრამების ახალი ტექნოლოგია: Windows Presentation Foundation 315

XIV თავი. WPF-ტექნოლოგია - ძირითადი ცნებები და თეორიული ასპექტები 316

14.1. Windows Presentation Foundation ტექნოლოგია 316
14.1.1. WPF ტექნოლოგიის შესაძლებლობები 318
14.1.2. WPF -ის შესაძლებლობები დიზაინერებისთვის 318
14.1.3. WPF -ის შესაძლებლობები C# - დეველოპერებისთვის 319
14.2. WPF-ის არქიტექტურა და კლასები 320
14.3. მომხმარებლის ინტერფეისის დაკომპლექტება 323
14.4. მართვის ელემენტები და შიგთავსები 325
14.5. ტექსტური მართვის ელემენტები 330
14.6. სიების მართვის ელემენტები 330
14.7. სპეციალიზებული მართვის ელემენტები 331
14.8. ბრძანებები 331
14.9. რესურსები 333
14.10. სტილები 336
14.11. შაბლონები 338
14.11.1. მართვის ელემენტთა შაბლონები 339
14.11.2. მონაცემთა შაბლონები 342
14.12. XAML ენის საფუძვლები 344
14.13. WPF- აპლიკაციების შექმნა 350
14.14. WPF აპლიკაცია მონაცემთა ბაზებით 351
14.15. მართვის ელემენტების მიბმა მონაცემთა წყაროსთან 362
14.16. მარშრუტიზებადი მოვლენები 365

XV თავი. WPF-ტექნოლოგიის ვიზუალური ელემენტები და მათი გამოყენება აპლიკაციების ასაგებად 370

15.1. Windows Presentation Foundation ტექნოლოგის სამუშაო გარემო და ინსტრუმენტების პანელი 370
15.2. ღილაკის დაპროგრამება ანიმაციის ელემენტებით 373
15.3. WPF-ის ფანჯრების მართვის ელემენტები 381
15.4. მრავალფანჯრიანი პროექტი WPF-ში 387
15.5. WPF-ის მომხმარებელთა მართვის ელემენტები 394
15.6. WPF-ში Web-გვერდების აპლიკაციების შექმნა 398

15.7. WPF-ში ფუნჯის სახეები და ფერების შერჩევა 409
15.8. WPF-ის მართვის ელემენტები: Menu, ToolBar, TabControl და ToolTip 426
15.9. WPF-ის მართვის ელემენტები: Thumb, Border და Popup 433
15.10. WPF-ის მართვის ელემენტები: ScrollViewer, Viewbox და StackPanel 440
15.11. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: Canvas.. 445
15.12. ინტერფეისული ელემენტების განლაგების მენეჯერი: StackPanel და DockPanel 450
15.13. ინტერფეისული ელემენტების განლაგების მენეჯერი: WrapPanel и UniformGrid 455
15.14. ინტერფეისული ელემენტების განლაგების მენეჯერი: Grid პანელი 463
15.15. WPF-ის საკუთარი განლაგების მენეჯერის დამუშავება: MyPanel 472

V ნაწილი. Workflow ტექნოლოგია 480

XVI თავი. ბიზნესპროცესების (Workflow) ვიზუალური დაპროგრამების ელემენტები 481

16.1. მარტივი ბიზნესპროცესის აგება 481
16.2. პროცედურული ელემენტები 484
16.3. დიზაინერის მართვა და XAML კოდი 494
16.4. კოდირებული სამუშაო პროცესები 501
16.5. ბიზნესპროცესის დიაგრამა 511

XVII თავი. ბიზნესპროცესების (Workflow) დაპროექტება ………………….......…. 521

17.1. რთული ბიზნესპროცესის არგუმენტები 521
17.2. ბიზნესპროცესის გამოსახულებათა ქმედებები 530
17.3. განმეორებადი ქმედებები 534
17.4. გამონაკლისების დამუშავება 540
17.5. Built-In ქმედების გაფართოება 548

XVIII თავი. კორპორაციის პროგრამული სისტემის დაპროექტება UML/2 და Workflow ტექნოლოგიებით ………………………….....…............................…… 559

18.1. საპრობლემო სფეროს დაპროექტება WF ტექნოლოგიით .NET პლატფორმაზე 559
18.2. საპრობლემო სფეროს სისტემის პროგრამული რეალიზაცია WF ტექნოლოგიით 564

VI ნაწილი. WCF ტექნოლოგია 566

XIX თავი. კომუნიკაცია აპლიკაციებს შორის WCF-ის ბაზაზე ……….............. 567

19.1. ინფორმაციის „გადაცემის“ და „მიღების“ ქმედებები 567
19.1.1. ახალი პროექტის შექმნა 567
19.1.2. შეტყობინებათა განსაზღვრა 568

19.1.3. კონტრაქტის შეტყობინება 572
19.1.4. სერვისის კონტრაქტი …..…………………….. 573
19.1.5. აპლიკაციის კონფიგურაცია 574
19.1.6. ბიზნესპროცესების განსაზღვრა 575
19.1.7. კლიენტი – მოთხოვნების გაგზავნა 575
19.1.8. გაგზავნის ქმედება 577
19.2. მომხმარებლის ქმედება 577
19.2.1. პასუხის მიღების ქმედება 581
19.2.2. სერვერი – მოთხოვნების დამუშავება 581
19.2.3. მიღების ქმედება 582
19.2.4. მომხმარებლის ქმედება 582
19.2.5. SendReply ქმედება 585
19.3. აპლიკაციის რეალიზაცია 590
19.3.1. ServiceHost კლასი 591
19.3.2. სერვისი - WorkflowService კლასი 592
19.3.3. ბოლო წერტილი 592
19.3.4. სამუშაო პროცესის გამომძახებელი 592
19.3.5. აპლიკაციის ამუშავება 595
19.3.6. ბიბლიოთეკის ფილიალის კონფიგურირება 595
19.3.7. მოსალოდნელი შედეგები 597
19.3.8. პორტებთან მიმართვის შესაძლებლობა 599
XX თავი. კომუნიკაცია ჰოსტაპლიკაციასთან 600
20.1. WPF პროექტის შექმნა 600
20.1.1. Class-ების გამოყენება 1-ელი თავიდან 601
20.1.2. Window Form-ის განსაზღვრა 601
20.1.3. სანიშნეები 615
20.2. SendRequest სამუშაო პროცესის რეალიზაცია 618
20.3. აპლიკაციის რეალიზაცია 625
20.3.1. მხარდაჭერა სამუშაო პროცესების ეგზემპლარებისათვის 625
20.3.2. მოვლენათა დამმუშავებელი (Event Handlers) 626
20.4. ApplicationInterface მეთოდები 628
20.5. აპლიკაციის ამუშავება 634
XXI თავი. Web - სერვისები 637
21.1. Workflow პროცესის სერვისის შექმნა 637
21.2. სერვისის კონტრაქტის განსაზღვრა 639
21.3. Receive და SendReply კონფიგურირება 644
21.4. PerformLookup აქტიურობის შექმნა 649

21.5. სერვისის ტესტირება 651
21.6. პარამეტრების გამოყენება 654
21.7. მეორე სერვისის შექმნა 654
21.8. მოდიფიცირებული PerformLookup ქმედების შექმნა 656
21.9. სერვისის ხელახალი ტესტირება 658
21.10. კლიენტის Workflow პროცესის შექმნა 659
21.11. Workflow პროცესის განსაზღვრა 661
21.12. ჰოსტ აპლიკაციის რეალიზაცია 662
21.13. აპლიკაციის ამუშავება 663
21.14. Pick-ის გამოყენება ………………………………………….................................. 664

VII ნაწილი. მონაცემთა მენეჯმენტი 667
XXII თავი. კორპორაციათა მართვის სისტემების მონაცემთა საცავები 668
22.1. მონაცემთა მოდელები მულტიმედიური სისტემებისათვის 668
22.1.1. მონაცემთა ახალი ტიპები 670
22.1.2. კლასთა იერარქიის აგება განზოგადების საფუძველზე 674
22.1.3. SQL/MM: მულტიმედიური მონაცემები და სტანდარტები ……............... 676
22.2. ობიექტრელაციური მულტიმედიური მონაცემთა ბაზის სისტემები 683
22.3. მოთხოვნების დამუშავება ობიექტრელაციურ მონაცემთა ბაზებში 689
22.4. ობიექტორიენტირებული მულტიმედიური მონაცემთა ბაზის
სისტემები 692
22.5. მულტიმედიური რელაციური ბაზის ოპტიმალური სტრუქტურის
დაპროექტება 701
22.6. მონაცემთა ბაზების ახალი ტექნოლოგიები ....……..... 708
22.6.1. დოკუმენტორიენტირებული მონაცემთა ბაზა 708
22.6.2. გრაფულ-ორიენტირებული მონაცემთა ბაზა 709
22.6.3. NewSQL მონაცემთა ბაზები 711
22.6.4. NoSQL მონაცემთა ბაზა MySQL ბაზასთან ერთად 713
22.6.5. MariaDB ბაზა MySQL-ის Open Source ალტერნატივა 715
22.6.6. MongoDB ბაზა და მისი ინსტალაცია 718
22.6.7. Hadoop – „დიდ მონაცემთა" ახალი ტექნოლოგია 718
22.7. მონაცემთა არარელაციური ბაზების განვითარების ძირითადი
მოთხოვნები და ტენდენციები ………………………...................................…..... 720
22.7.1. ვერტიკალური და ჰორიზონტალური სკალირება 721
22.7.2. მონაცემთა შენახვის ტიპები 723
22.7.3. განაწილებული გარემო: replication და sharding 725
22.7.4. MySQL Mongo DB ბაზების შედარება 730

22.7.5. მონაცემთა ბაზის აგება Mongo DB სისტემაში 733
22.7.5.1. RoboMongo პლატფორმა 734
22.7.5.2. ბრძანებების და ოპერაციების მაგალითები MongoDB ბაზაში 740
XXIII თავი. მონაცემთა ბაზების მართვის სისტემები 747
23.1. მონაცემთა ბაზების მართვის სისტემა SQL Server 2012/14 747
23.1.1. მონაცემთა ძირითადი ტიპები 747
23.1.2. მონაცემთა ბაზის ობიექტების შექმნა 749
23.1.3. დეკლარაციული მთლიანობის შეზღუდვები და დომენები 753
23.2. მონაცემთა ბაზების უსაფრთხოების სისტემა 755
23.2.1. აუტენთიფიკაცია 756
23.2.2. მონაცემთა შიფრაცია 756
XXIV თავი. სივრცითი მონაცემთა ტიპები SQL Server მონაცემთა ბაზაში......... 758
24.1. მონაცემთა ტიპი GEOMETRY 758
24.2. მონაცემთა ტიპი GEOGRAPHY 760
24.3. სივრცით მონაცემებთან მუშაობა 761
24.3.1. GEOMETRY ტიპის მონაცემებთან მუშაობა 762
24.3.2. GEOGRAPHY ტიპის მონაცემებთან მუშაობა 765
24.4. სივრცით ინდექსებთან მუშაობა 766
24.5. ინფორმაციის ასახვა სივრცითი მონაცემების შესახებ 768
24.6. სიახლეები სივრცით მონაცემებთან სამუშაოდ 769
24.7. ახალი სისტემური შენახვადი პროცედურები სივრცითი მონაცემებისთვის 771
VIII ნაწილი. ბიზნესაპლიკაციების რეალიზაცია დაპროგრამების ჰიბრიდული ტექნოლოგიებით 773
XXV თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „უნივერსიტეტი“.. 774
25.1. სისტემის ობიექტროლური მოდელის დაპროექტება 775
25.2. არსთა დამოკიდებულების ER მოდელის დაპროექტება 779
25.3. მონაცემთა ბაზის სერვერზე განთავსება 780
25.4. ბიზნესპროცესის სერვისის შექმნა 782
25.5. სერვისის კონტრაქტის განსაზღვრა 784
25.6. Receive და SendReply კონფიგურირება 787
25.7. PerformLookUp ქმედების აგება (ძებნის შესრულება) 788
25.8. სერვისის ტესტირება 789
XXVI თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „საფინანსო ბანკი“ 793
26.1. ინფორმაციის გაცვლის პროგრამული რეალიზაციის ამოცანა SOA-ისთვის 793

26.2. Windows Form-ის განსაზღვრა 793
26.3. სერვისის პროგრამული რეალიზაცია 797
26.4, ServiceHost -ის რეალიზაცია 798
XXVII თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „სატრანსპორტო გადაზიდვები“ 801
27.1. მულტიმოდალური გადაზიდვების მართვის საინფორმაციო სისტემის აგების ამოცანა 801
27.2. მონაცემთა ბაზის დაპროექტება და რეალიზაცია 802
27.3. მონაცემთა ბაზასთან მუშაობის ინტერფეისის აგება 808
XXVIII თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „ელექტრონული არჩევნები“ 817
28.1. ელექტრონული არჩევნების მართვის საინფრომაციო სისტემის აგების ამოცანა 817
28.2. ელექტრონული არჩევნების სისტემის კონცეპტუალური მოდელის აგება ORM/ERM ტექნოლოგიით 818
28.3. მონაცემთა ბაზის აგების პროგრამული რეალიზაციის ავტომატიზებული პროცედურა 825
28.4. სისტემის პროგრამული რეალიზაცია VS .NET Framework გარემოში 828
28.5. მომხმარებლის ინტერფეისის დამუშავება 829
28.6. კომუნიკაცია: WCF ტექნოლოგია 834
28.7. ბიზნესპროცესის რეალიზაცია 849
28.8. სერვისის კონტრაქტის რეალიზაცია 851
28.9. ServiceHost-ის რეალიზაცია 852
28.10. SendRequest ბიზნესპროცესის რეალიზაცია 853
28.11. ProcessRequest ბიზნესპროცესის რეალიზაცია 856
28.12. აპლიკაციის რეალიზაცია 859
28.13. ბიზნესპროცესების ეგზემპლარების მხარდაჭერა 859
28.14. მოვლენათა დამმუშავებელი (Event Handlers) 860
28.15. ApplicationInterface მეთოდები 862
28.16. აპლიკაციის ამუშავება 867
28.17. დასკვნა 870
XXIX თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „შავი ზღვის ეკოლოგია“ 872
29.1. შავი ზღვის საქართველოს მდინარეების მონაცემთა ბაზის განახლების ინტერფეისის აგების ამოცანა 872
29.2. მონაცემთა ბაზის განახლება ADO.NET დრაივერის და DataGridView კლასის გამოყენებით 881

29.3. შავის ზღვის კონცეპტუალური ORM-მოდელის აგება ეკოლოგიური პარამეტრებისთვის 893
29.4. შავი ზღის ეკოპარამეტრების Ms SQL Server ბაზა 897
XXX თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „მარკეტინგი“ 900
30.1. მარკეტინგის მენეჯმენტის ბიზნესპროცესები და მათი მოდელირების ნოტაცია 900
30.2. საწარმოო რესურსების მართვის (ERP) სისტემა და მისი დანერგვის პროცესები 903
30.3. ბიზნესპროცესის დოკუმენტირება და სიმულაცია 918
30.4. მარკეტინგული პროცესების მოდელირება პეტრის ფერადი ქსელებით... 937
30.5. CPN ქსელის მდგომარეობათა სივრცის ანალიზი პროდუქციის რეალიზაციის პროცესისათვის 943
30.6. იმიტაციური მოდელირების პროცესის ლისტინგი პროდუქციის მიწოდების CPN ქსელისათვის 950
30.7. მარკეტინგული პროცესების პროგრამული რეალიზაცია კლიენტ-სერვერული არქიტექტურით 952
30.8. დასკვნა 958
ლიტერატურა 959
დანართი_1: Web-ის უსაფრთხოების და სერვისის მომხმარებელთა ინტერფეისების რეალიზაცია 970
დანართი_2: მონაცემთა ბაზის აგების ექსპერიმენტული მაგალითი MongoDB სისტემაში 977
დანართი_3: Web-სერვისის რეალიზაცია შავი ზღვის მდინარეთა ესტუარების მონიტორინგის სისტემისათვის 991
Index / ინდექსი 996

შესავალი

მართვის საინფორმაციო სისტემების პროგრამული უზრუნველყოფის ობიექტორიენტირებული ანალიზის, დაპროექტების, დეველოპინგის, ტესტირების, დანერგვისა და რეინჟინერინგის საკითხების ინტენსიური კვლევა და სწავლება განსაკუთრებით მნიშვნელოვანი და აქტუალურია თანამედროვე კომპიუტერული ინდუსტრიისა და ინფორმაციული ტექნოლოგიების უახლესი მიღწევების ფონზე [1,2]. მონოგრაფიაში შემოთავაზებულია გამოყენებითი პროგრამული სისტემების დაპროექტებისა და რეალიზაციის საკითხები მაიკროსოფთის კორპორაციის ახალი ტექნოლოგიების ბაზაზე; აგრეთვე მონაცემთა მენეჯმენტის აქტუალური საკითხები, როგორიცაა კორპორაციათა მართვის საინფორმაციო სისტემების ინფორმაციული რესურსების დაცვისა და ეფექტიანი გამოყენების ამოცანები.

წიგნი რვა ნაწილისა და 30 თავისაგან შედგება. მასში ასახულია პროგრამული ინჟინერიისა და მონაცეთა მენეჯმენტის თანამედროვე კონცეფციები და ტექნოლოგიები. განსაკუთრებით ყურადღება გამახვილებულია მაიკროსოფთის .NET პლატფორმასა და უნივერსალური პროგრამული ენების არსენალზე [3-5].

პირველ ნაწილში გაანალიზებულია დაპროგრამების მეთოდები, სტილი, მოდელები და ენები - დაპროგრამების პარადიგმების ფონზე. წარმოდგენილია პროგრამული უზრუნველყოფის აგების ჰიბრიდული ტექნოლოგიები: Windows Presentation Foundation (WPF), Workflow Foundation (WF) და Windows Communication Foundation (WCF), რომლებიც დღეისათვის ნამდვილად მძლავრ და მოქნილ ინსტრუმენტულ საშუალებებს მიეკუთვნება [10-18]. მათი გამოყენებით შექმნილია და ფუნქციონირებს არა ერთი დიდი პროექტი სახელმწიფო და კერძო სტრუქტურების ობიექტების მართვის საინფორმაციო სისტემების სახით.

წიგნის მეორე ნაწილი ეხება უსაფრთხო პროგრამული უზრუნველყოფის შექმნისა და გამოყენების სტანდარტებსა და მეთოდოლოგიებს, რომლებიც აღიარებულია ამერიკის, ევროპისა და აღმოსავლეთის მოწინავე ქვეყნების საერთაშორისო ორგანიზაციების მიერ. ესაა BSI, ITIL და COBIT სისტემები [6,19-24,49,50].

მესამე ნაწილში ასახულია გამოყენებითი სისტემების პროგრამული უზრუნველყოფის შექმნის უნიფიცირებული (UML) და მოქნილი (Agile) მეთოდოლოგიები, სასიცოცხლო ციკლის გუნდური მენეჯმენტის ამოცანები [25-33].

ნაშრომის მეოთხე, მეხუთე და მეექვსე ნაწილები ეხება WPF, WF და WCF დაპროგრამების ჰიბრიდული ტექნოლოგიების დეტალურ განხილვას, მრავალი საილუსტრაციო საინჟინრო ამოცანით [16-18]. წარმოდგენილია კლიენტ-სერვერული და Web-სერვისების აგების საკითხები.

მეშვიდე ნაწილში ასახულია მონაცემთა მენეჯმენტის თანამედროვე პრობლემები, ამოცანები და გადაწყვეტის ტექნოლოგიები. განიხილება კორპორაციათა მონაცემთა

საცავების, ბაზებისა და დაცვის საკითხები, მონაცემთა ახალი ტიპებისა და სივრცითი ბაზების აქტუალური საკითხები [34-38].

წიგნის მერვე ნაწილი წარმოადგენს ავტორთა მიერ წლების განმავლობაში შესრულებული საპროექტო სამუშაოების შედეგების აღწერას დაპროგრამების ჰიბრიდული ტექნოლოგიებისა და მონაცემთა მენეჯმენტის სფეროში.

ნაშრომი განკუთვნილია პროგრამული ინჟინერიისა და მართვის საინფორმაციო სისტემების სფეროს სპეციალისტებისათვის, სტუდენტების, მაგისტრანტებისა და დოქტორანტებისათვის, აგრეთვე პროგრამული უზრუნველყოფის მენეჯმენტის საკითხით დაინტერესებულ მკითხველთა ფართო წრისათვის.

I ნაწილი:

ბიზნესპროცესების მართვის საინფორმაციო სისტემები და პროგრამული ინჟინერია

I თავი. დაპროგრამების ტექნოლოგიები, პლატფორმები და ენები 18

II თავი. Windows- და Web-დაპროგრამების ჰიბრიდული ტექნოლოგიები ………... 38

III თავი. Ms Visual Studio.NET Framework ინტეგრირებული გარემო,
აპლიკაციების სტრუქტურა და პროცესების მართვა ……………………........... 53

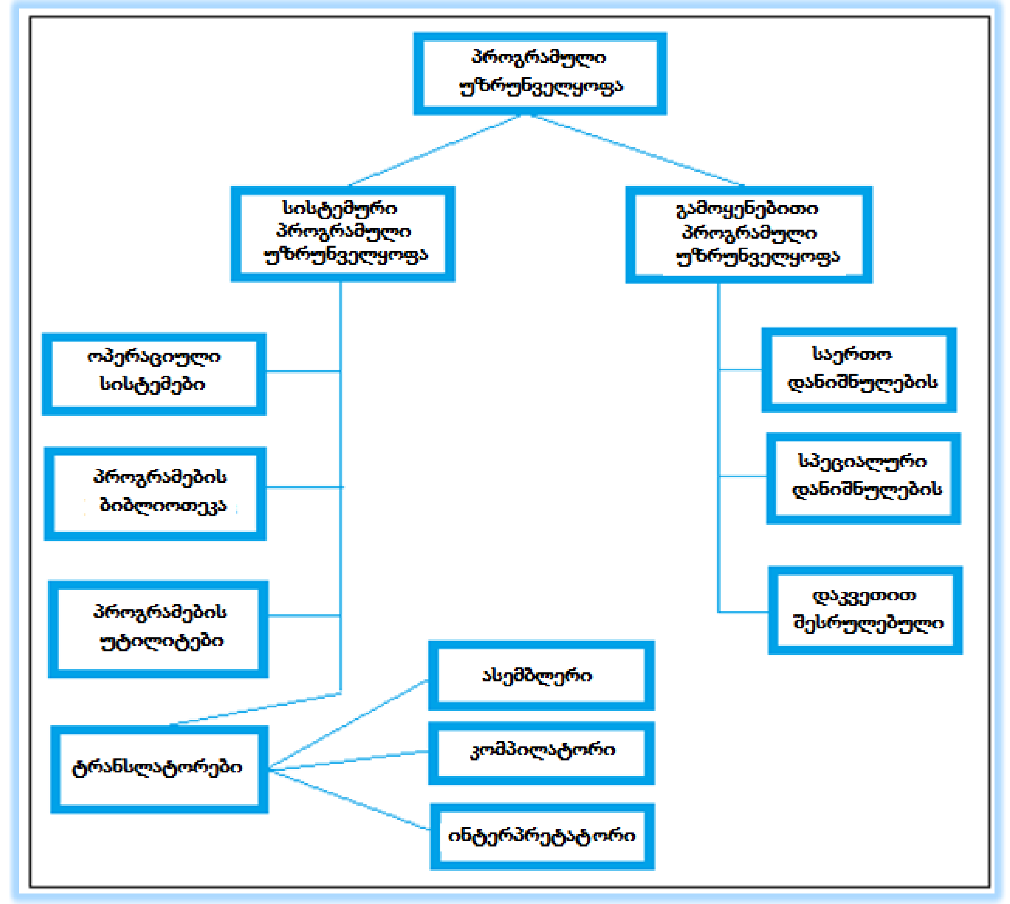
IV თავი. Visual C#.NET ენა - აპლიკაციების აგების ვიზუალური
ინსტრუმენტული საშუალება 62

V თავი. გამოყენებითი პროგრამული სისტემების მენეჯმენტი 104

I თავი
დაპროგრამების ტექნოლოგიები, პლატფორმები და ენები

1.1. პროგრამული ტექნოლოგიების განვითარების ისტორია

პროგრამული უზრუნველყოფის (Software) დამუშავება (Software Engineering, Development) განსაკუთრებულად რთული სისტემების კლასს მიეკუთვნება. მრავალფეროვანია დღეისათვის არსებული პროგრამული სისტემების სიმრავლე და მათი სახეები. თუ მათ კლასიფიკაციას შევეხებით, ძირითადად შეიძლება ორი დიდი მიმართულება გამოვყოთ: სისტემური პროგრამები (System Software) და გამოყენებითი პროგრამები (Applied Software), რომლებიც თავის მხრივ სპეციალური სახის პროგრამულ ქვესიმრავლეებად იყოფა (ნახ.1.1) [10].



ნახ.1.2. პროგრამული უზრუნველყოფის კლასიფიკაცია

ჩვენი წიგნის კვლევის ობიექტის, როგორც ზემოთ აღვნიშნეთ, გამოყენებითი პროგრამული პაკეტების მენეჯმენტის საკითხებია. კერძოდ კი კორპორაციული ორგანიზაციების ბიზნესპროცესების ავტომატიზაციის მიზნით შესაბამისი ინფორმაციული, მოდელური და პროგრამული უზრუნველყოფების შემუშავება ახალი ტექნოლოგიების ინტეგრაციის საფუძველზე, ინფორმაციული უსაფრთხოების დაცვით [11].

პროგრამული სისტემების მენეჯმენტის თანამედროვე მეთოდოლოგიები, პროგრამული პარადიგმების განვითარების ფონზე, მნიშვნელოვნად განსხვავდება ერთმანეთისაგან, რაც, ერთგვარად, მრავალ ობიექტურ და სუბიექტურ ფაქტორზეა დამოკიდებული. ამ მიმართულებათა მიმდევარი მეცნიერები და პრაქტიკოსი პროგრამისტები ცდილობენ დაასაბუთონ თავიანთ მოსაზრებათა ჭეშმარიტება და იდეალურად წარმოადგინონ ესა თუ ის კონცეფცია. მაგალითად, უნიფიცირებული მოდელირების ენა (UML - გ. ბუჩი, ი. ჯაკობსონი, ჯ. რამბო და სხვ.) თუ მოქნილი (Agile) დაპროგრამება, ექსტრემალური პროგრამირების მაგალითზე (ზ. კენტი, დ. მარტინი და სხვ.) [12,13]. თუმცა ისეთებიც მოინახება, რომლებიც კომპრომისული მიდგომით ჰიბრიდულ ვარიანტსაც გვთავაზობენ (ს. ამბლერი, ბ. რუმპე და სხვ.) [14,15].

საქართველოში ამ მიმართულებას ავითარებენ სტუ-ს პროფესორები გ. გოგიჩაიშვილი, გ. სურგულაძე, ე. თურქია, თ. სუხიაშვილი [1,2,5,9, 15-18].

ახლა მოკლედ განვიხილოთ გამოყენებითი პროგრამული სისტემების დაპროგრამების ტექნოლოგიების განვითარების ისტორია, რაც ლიტერატურულ წყაროებში ვრცლადაა წარმოდგენილი [19-21].

თვდაპირველად საჭიროა გაირკვეს, თუ რა პრობლემების გადაჭრის მიზნით წარმოიშვა მოქნილი მეთოდოლოგიები. მოკლედ რომ ვთქვათ, პრობლემის არსი იმაში მდგომარეობს, რომ დაწყებული პირველი პროგრამული პროექტებიდან დღემდე, პროგრამული უზრუნველყოფის დამუშავება იყო და რჩება ნაკლებ პროგნოზირებად და ხშირად წარუმატებელ საქმედ [19].

პროგრამული უზრუნველყოფის შექმნის პროექტების დიდი რაოდენობა კვლავაც მთავრდება ბიუჯეტისა და ვადების გადამეტებით, ხოლო შედეგად შექმნილი პროგრამები ხშირად ბოლომდე ვერ პასუხობს მომხმარებელთა მოთხოვნებს, ან მოაქვთ მცირე რეალური სარგებლობა ბიზნესისთვის.

აღნიშნული პრობლემები არის პროგრამული უზრუნველყოფის კრიზისის ძირითადი გამოვლინება. მიუხედავად მნიშვნელოვანი ინტელექტუალური ძალისხმევისა კრიზისის დასაძლევად, დღემდე ვერ მოინახა რაიმე უნივერსალური გადაწყვეტა (როგორც ხშირად ამბობენ „ვერცხლის ტყვია" - მეტაფორა ფრედ ბუკსის 1986 „No Silver Bullet": IT-ში ახლი ეფექტური ტექნოლოგიური გადაწყვეტა („ვამპირების წინააღმდეგ"), მაგალითად, პროექტების მართვის სტრუქტურული მიდგომა) [20,21].

პროექტების სტრუქტურული მართვის არსი მდგომარეობს 10-ეტაპიანი სამუშაოს შესრულებაში:

1. მიზნის ვიზუალური წარმოდგენა; ყურადღების გამახვილება პრიზისათვის;
2. ამოცანათა სიის შემუშავება, რომლებიც უნდა შესრულდეს;
3. უნდა არსებობდეს მხოლოდ ერთი ლიდერი;
4. ადამიანთა მიმაგრება ამოცანებზე;
5. მოსალოდნელი შედეგების მართვა, რეზერვების გათვლა მოსალოდნელი შეცდომებისათვის, სათადარიგო პოზიციების გამომუშავება (რისკების მართვა);
6. ხელმძღვანელობის შესაბამისი სტილის გამოყენება;
7. ცოდნა იმისა, რაც ხდება;
8. შემსრულებელთა ინფორმირება იმაზე, რაც ხდება;
9. გამეორდეს 1-8 ეტაპები მე-10 ეტაპამდე;
10. პრიზი (ჯილდო).

ამ ეტაპების შესრულება გარანტიაა პროექტის წარმატების დასასრულებლად. გამოიყენება ტერმინები: „ჩანჩქერის“ მეთოდი, იტერაცია. დაგეგმვა. რისკები.

მოქნილი მეთოდები. ესაა პროგრამული ინდუსტრიის თანამედროვე პასუხი, თუ როგორ უნდა შესრულდეს პროექტები, რათა ისინი წარმატებით დასრულდეს მაღალი ალბათობით და მოუტანოს კმაყოფილება ყველა დაინტერესებულ მხარეს - და, პირველ რიგში, დამკვეთს და პროექტის შემსრულებელ გუნდს [19,22].

მოკლედ თვალი გადავავლოთ პროგრამული უზრუნველყოფის განვითარების ისტორიას. დაპრგრამების ენების განვითარება და დიდი პროგრამული სისტემების შექმნა დაიწყო 1950-იან წლებიდან. დაპროგრამების თავდაპირველი მიდგომები იყო ნაკლებად ფორმალიზებული და წარმოადგენდა პროცესს სახელწოდებით Code-and-Fix (კოდირება და გასწორება).

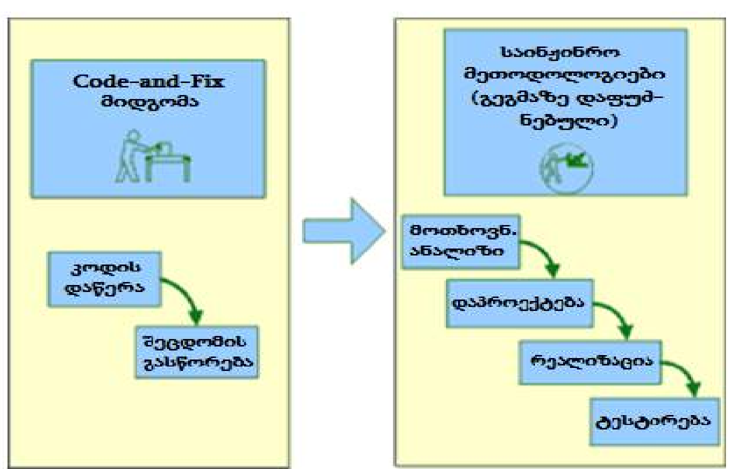
ამ მიდგომის დროს პროგრამული უზრუნველყოფის დამუშავება იწყებოდა უშუალოდ კოდირებით (ყოველგვარი წინასწარი დაგეგმვის, მოთხოვნილებათა ანალიზისა და დაპროექტების გარეშე). ამის შემდეგ კოდში ნაპოვნი პრობლემები (დეფექტები, მოთხოვნილებებთან შეუთავსებლობა და ა.შ.) სწორდებოდა კოდში ცვლილებების უშუალოდ ხელით შეტანით.

დროთა განმავლობაში გასაგები გახდა, რომ დიდი სტაბილური პროგრამული სისტემების ასაგებად საჭირო იყო უფრო გააზრებული და ფორმალური მიდგომები. პრობლემების გადასაწყვეტად ყურადღება მიექცა იმ დროისათვის უკვე კარგად განვითარებულ ისეთ ადამიანური შემოქმედების სფეროს, რომელიც კავშირშია რთულ საწარმოო პროცესებთან. მაგალითად, როგორიცაა სისტემტექნიკა (System Engineering), დაპროექტება და სხვა საინჟინრო დისციპლინები.

ასე გაჩნდა ახალი მიმართულება - პროგრამული ინჟინერია (Software Engineering), ხოლო ფაქტობრივი სტანდარტის სახით წლების განმავლობაში დამკვიდრებულ იქნა ე.წ.

პროგრამული უზრუნველყოფის დამუშავების საინჟინრო მეთოდოლოგიები. ამ მეთოდოლოგიებს ხშირად უწოდებენ გეგმაზე დაფუძნებულს (Plan-driven), რადგანაც მის საფუძველში დევს მოსაზრება, რომ პროგრამული უზრუნველყოფის დამუშავების პროცესი არის დეტერმინირებული ინჟინრული პროცესი, რომელიც შეიძლება დაიგეგმოს თავიდან ბოლომდე და შემდეგ შესრულდეს გეგმის მიხედვით ფორმალური ინჟინრული მიდგომების გამოყენებით.

პროგრამის სასიცოცხლო ციკლის აგების ძირითად ვარიანტად გამოიყენებოდა „ჩანჩქერის“ მოდელი, რომელიც ითვალისწინებს მოთხოვნილებათა ანალიზის, დაპროექტების, კოდირების, ტესტირებისა და სხვა ფაზების ერთჯერადად შესრულებას (ნახ.1.2).



ნახ.1.2. საინჟინრო მეთოდოლოგიებზე გადასვლა

გარდა აქ განხილული ორი მიდგომისა, არსებობდა აგრეთვე სასიცოცხლო ციკლის სხვა ვარიანტებიც, როგორც გარკვეული მოდიფიკაციები ჩანჩქერული და იტერაციული მოდელებისა. ჩვენ აქ ვიხილავთ მხოლოდ ძირითადს, რომლებიც პრაქტიკაში მასობრივად გამოყენებოდა.

- **საინჟინრო მეთოდოლოგიების პრობლემები:**

გადასვლა საინჟინრო მეთოდლოგიებზე და ჩანჩქერულ მოდელზე ნამდვილად იყო წინგადადგმული ნაბიჯი. მან შემოიტანა განსაზღვრული მოწესრიგება და ორგანიზებულობა დამუშავების პროცესში, ასევე აღმოფხვრა Code-and-Fix მიდგომის მრავალი პრობლემა [19]. მაგრამ საინჟინრო მეთოდოლოგიებმა ვერ შეძლეს პროგრამული პროექტების ყველა პრობლემის გადაწყვეტა, უპირველეს ყოვლისა, მასში ჩადებული შინაგანი კონფლიქტის გამო. საქმე ისაა, რომ ისინი თავდაპირველად იქმნებოდა სხვა საინჟინრო დისციპლინების ნიმუშის მიხედვით და სრულად არ ითვალისწინებდა თავისებურებებს, რომლებიც პროგრამულ უზრუნველყოფას ახასიათებს, როგორც

უნიკალური სახის პროდუქციას (მაგალითად, Java-პროგრამული პროდუქტის შექმნა და მაცივრის წარმოება).

პროგრამული უზრუნველყოფის უნიკალურობის კონფლიქტს საინჟინრო მეთოდოლოგიების მიდგომებთან აქვს მინიმუმ ორი ძირითადი გამოვლენა:

1. **ადამიანური ფაქტორის როლის არასაკმარისი შეფასება.**

საინჟინრო მეთოდოლოგიები განიხილავდა პროგრამული უზრუნველყოფის დამუშავების პროცესს როგორც აბსტრაქტული შემსრულებლების მიერ ჩატარებულ ბიჯების თანამიმდევრობას, ანუ ძირითადი ყურადღება მახვილდებოდა მხოლოდ იმაზე, თუ რა უნდა გაკეთებულიყო და არა იმაზე, თუ ვის უნდა გაეკეთებინა. სინამდვილეში, პროგრამული პროექტების უმეტეს პრობლემას აქვს სოციალური და არა ტექნოლოგიური ხასიათი. სწორედ პროექტის მონაწილეები და მათ შორის ურთიერთობები არის პროექტის წარმატების განმსაზღვრელი ძირითადი ფაქტორი. როგორც გამოკვლევები ამ სფეროში გვიჩვენებს, კომპანიის საუკეთესო დეველოპერების მწარმოებლურობა 10-ჯერ მაღალია, ვიდრე დაბალი დონისა, და 2,5-ჯერ მეტი, ვიდრე საშუალოსი. ამგვარად, მწარმოებლურობაზე მოქმედი ძირითადი ფაქტორებია არა დამუშავების საშუალებები და მეთოდოლოგიები, არამედ სამუშაო ადგილის ხარისხი, ფსიქოლოგიური გარემო გუნდში, ეფექტური კომუნიკაციები და სხვა სოციალური ფაქტორები;

2. **ჩანჩქერული სასიცოცხლო ციკლის შეუსაბამობა პროგრამული უზრუნველყოფის ბუნებასთან.**

აქ არ განიხილება ჩანჩქერული მოდელის დეტალები (ისინი მრავლადაა ლიტერატურულ წყაროებში). საჭიროა აღინიშნოს მხოლოდ პრინციპული მომენტი, რომ ყოველი პროგრამული პროდუქტი პრაქტიკულად არის უნიკალური და ახალი. მისი დამუშავება მიმდინარეობს მაღალი განუსაზღვრელობის პირობებში, და მცდელობა იმისა, რომ წინასწარ იქნას გათვალისწინებული ყველა ფაქტორი პროექტის დასაწყისში, (დეტალური მოთხოვნების შედგენა, სისტემის დიზაინის სრული დამუშავება და სხვა) წინასწარაა განწირული წარუმატებლობისათვის. ჩანჩქერული მოდელის ნაკლოვანებები არა მხოლოდ ართულებს პროგრამული სისტემის დამუშავებას, არამედ იგი ავიწროვებს ადამიანურ ურთიერთობებს. მაგალითად, პროექტის დასაწყისში შედგენილი და დამტკიცებული დეტალური მოთხოვნები ხშირად ხდება დამკვეთის და მიმწოდებლის უთანხმოების მიზეზი რეალიზაციის სტადიაზე, როცა საჭიროა აუცილებელი ცვლილებების განხორციელება. რა თქმა უნდა, არსებობს პროექტებიც, რომლისთვისაც ჩანჩქერული სასიცოცხლო ციკლი მისაღებია, მაგრამ უმრალესი პროექტისათვის ზემოაღწერილი პრობლემები აქტუალურია და მათ არგათვალისწინებას პროექტების ჩავარდნამდე მივყავართ.

დასკვნის სახით შეიძლება ითქვას, რომ საინჟინრო მეთოდოლოგიებმა ვერ შეძლო პროგრამული უზრუნველყოფის კრიზისის გადაწყვეტა. პროექტების წარუმატებლობის მრავალმა შემთხვევამ, რომლების საინჟინრო მეთოდოლოგიების საფუძველზე

ხორციელდებოდა, დღის წესრიგში დააყენა ალტერნატიული მიდგომების ძიების საკითხი, რაც შემდგომ განხორციელდა.

- **მსუბუქი მეთოდოლოგიები:**

ამ მიზნით პროგრამული უზრუნველყოფის დამუშავების საინჟინრო მეთოდოლოგიების პრობლემების შეცნობამ გამოიწვია ახალი ალტერნატიული მიდგომების შექმნა. განსაკუთრებით სწრაფად განვითარდა ეს პროცესი 90-ანი წლების მეორე ნახევარში, როცა დიდი პოპულარობა მოიპოვა მსუბუქმა (ან მსუბუქი წონის) მეთოდებმა, როგორებიცაა Scrum, DSDM, Crystal და სხვ. [19, 23].

მიუხედავად ზოგიერთი განსხვავებისა, პრაქტიკული გამოყენების თვალსაზრისით, ეს მიდგომები მსგავსია იმით, რომ ალტერნატივის სახით მძიმე და ზედმეტად ფორმალიზებული საინჟინრო მეთოდოლოგიებისაგან განსხვავებით, ისინი იძლევა ადაპტურ, იტერაციულ და ადამიანზე ორიენტირებულ მიდგომებს.

მსუბუქი მიდგომების ავტორები, მ. ფოულერი, კ. ბეკი, ა. კოუბერნი და სხვ. [13,24] ერთობლივი შეხვედრებისა და კონსულტაციების საფუძველზე (მაგალითად, 2001 წ. სნოუბორდში, იუტას შტატი) მივიდნენ იმ გადაწყვეტილებამდე, რომ ამ მიმართულებისათვის დაერქმიათ "Agile" (მოქნილი, სწრაფი). შეიქმნა დოკუმენტი „პროგრამული სისტემების მოქნილი დამუშავების მანიფესტი" (4 პუნქტით), მას მოგვიანებით დაემატა პრინციპების სია (12 პუნქტით), ის დეტალურად ხსნიდა მანიფესტს [24].

1.2. დაპროგრამების თანამედროვე პლატფორმები და ენები

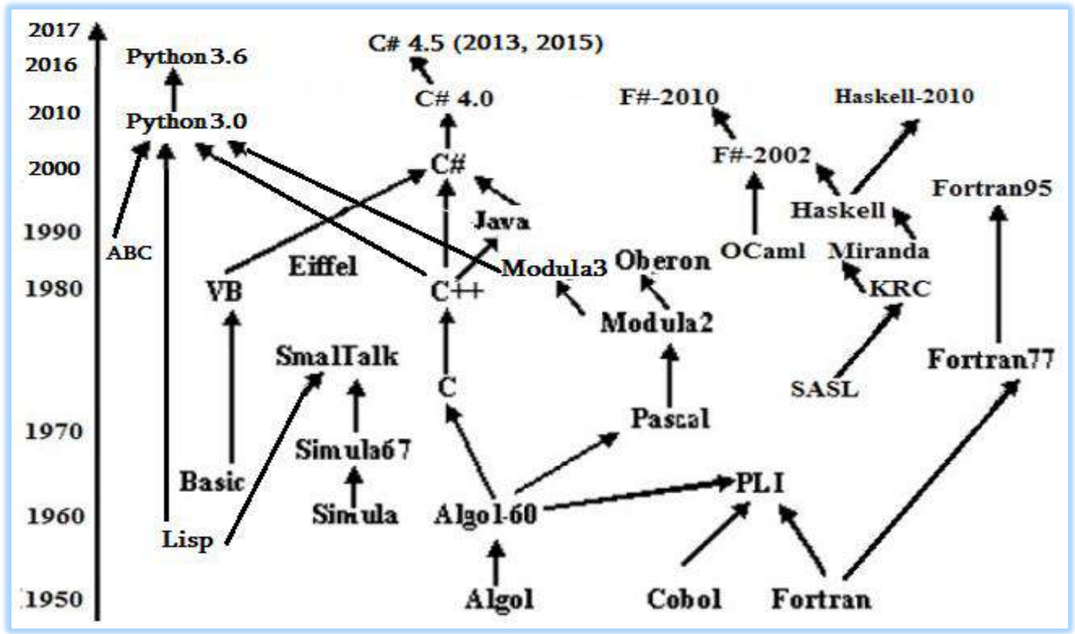
დაპროგრამების პლატფორმა და ენა სხვადასხვა ცნებებია. პლატფორმა მიეკუთვნება კომპიუტერის აპარატულ-პროგრამულ ტექნოლოგიას.

კომპიუტერული პლატფორმა განისაზღვრება მისი პროცესორის და შინების აპარატურული გადაწყვეტით, რომელიც პირდაპირ კავშირშია მის ოპერაციულ სისტემასთან (მაგალითად, Unix, MsWindows, Linux). ოპერაციულ სისტემებს აქვს აპლიკაციის დაპროგრამების ინტერფეისები (API), რომელთა თავსებადობითაც განისაზღვრება ოპერაციულ სისტემათა ოჯახი. ამრიგად, *პროგრამული პლატფორმა* ოპერაციული სისტემაა, რომლის მეშვეობითაც რეალიზდება გამოყენებითი პროგრამული პაკეტები, ანუ მომხმარებელთა პროგრამული აპლიკაციები.

დაპროგრამების ენა ის ინსტრუმენტული საშუალებაა, რომელზეც იწერება როგორც პროგრამული პლატფორმები (ოპერაციული სისტემები), ასევე მომხმარებელთა პროგრამული აპლიკაციები. მაგალითად, C-ენაზე 1972 წელს დაიწერა პირველი ქსელური ოპერაციული სისტემა Unix და შეიქმნა ახალი „მაღალი საიმედოობით"

ცნობილი პლატფორმა, რომელსაც დღესაც ერთ-ერთი წამყვანი ადგილი უჭირავს პროგრამული ინჟინერიის ბაზარზე.

გამოთვლით მანქანებზე რეალიზებული დაპროგრამების ენების განვითარების ისტორია 1950 წლიდან იწყება (ნახ.1.3).



ნახ.1.3. დაპროგრამების ენების განვითარების ისტორია

დღემდე შექმნილია 2000-ზე მეტი ენა, მათგან კი მხოლოდ „ძლიერებმა" მოაღწიეს ჩვენს საუკუნემდე და დღესაც ვითარდებიან. დაპროგრამების ენების საკონკურენციო ბაზარზე C-ენის „შთამომავლები" (C++, Java, C#) ბატონობენ. ამას ხელი შეუწყო თვით Unix ოპერაციული სისტემის უნივერსალობამ და ანტივირუსულმა საიმედოობამ (MsWindows-თან შედარებით) [2].

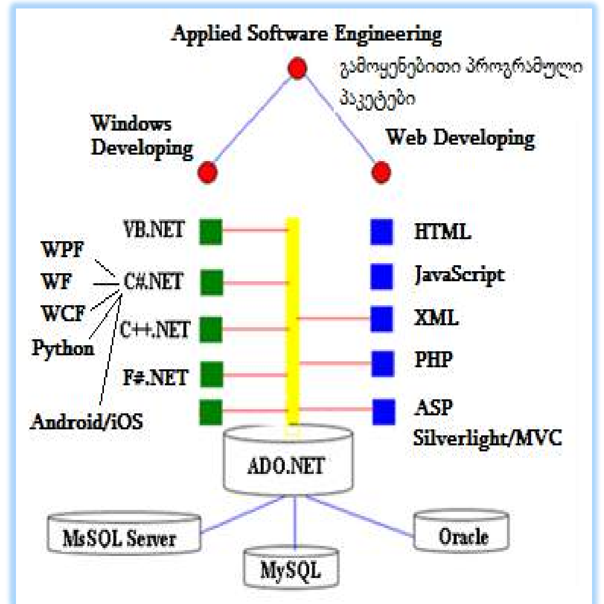
ნახაზიდან ჩანს დაპროგრამების ენების (აქ ყველა არაა წარმოდგენილი ინფორმაციის დიდი მოცულობის გამო) ევოლუცია „პროგრამირების პარადიგმის" თვალსაზრისით. პროგრამირების პარადიგმა არის ცნებათა და კონცეფციათა ერთობლიობა, რომლებიც პროგრამების დაწერის **სტილს** განსაზღვრავს. მას ენაში შეაქვს თვისობრივი ცვლილებები.

დღეისათვის ცნობილია დაპროგრამების რამდენიმე სტილი: ოპერატორული, ფუნქციური, იმპერატიული, ლოგიკური, სტრუქტურული, ობიექტ-ორიენტირებული, ვიზუალური, კომპონენტური, სუბიექტური და ა.შ.

მაგალითად C-ენა სტრუქტურული დაპროგრამების სტილითაა ცნობილი, C#, C++, Visual Basic, Java ენები ობიექტორიენტირებული სტილისაა და ა.შ. Haskell და F#

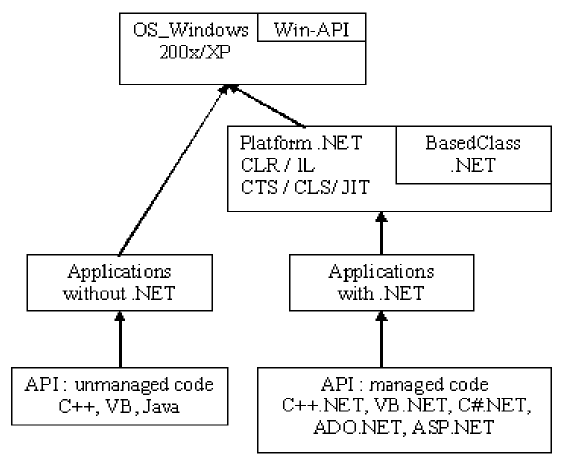
მეტაფუნქციური ენებია, რომლებიც ობიექტორიენტირებულ სტილსაც ფლობენ, ამიტომ აქტუალურია მათი გამოყენება ამერიკისა და ევროპის უნივერსიტეტებში.

მაიკროსოფთმა Ms Visual Studio.NET 2010 ვერსიაში პირველად წარმოადგინა F# ენა, რომლის გამოყენებასაც წარმატებული მომავალი ექნება მათემატიკური და საინჟინრო ამოცანების გადაწყვეტაში (ნახ.1.4).



ნახ.1.4

Ms Visual Studio.NET Framework 4.0/4.5 პროგრამული პლატფორმაა, რომელიც Windows ოპერაციულ სისტემასა და გამოყენებით პროგრამულ აპლიკაციას შორისაა მოთავსებული (ნახ.1.5). ამ ინტეგრირებული პროგრამული პაკეტების გამოყენების მიზანია რთული სისტემების მოდელირება, კონსტრუირება და რეალიზაცია უნიფიცირებული პროგრამირების კონცეფციის გამოყენებით.

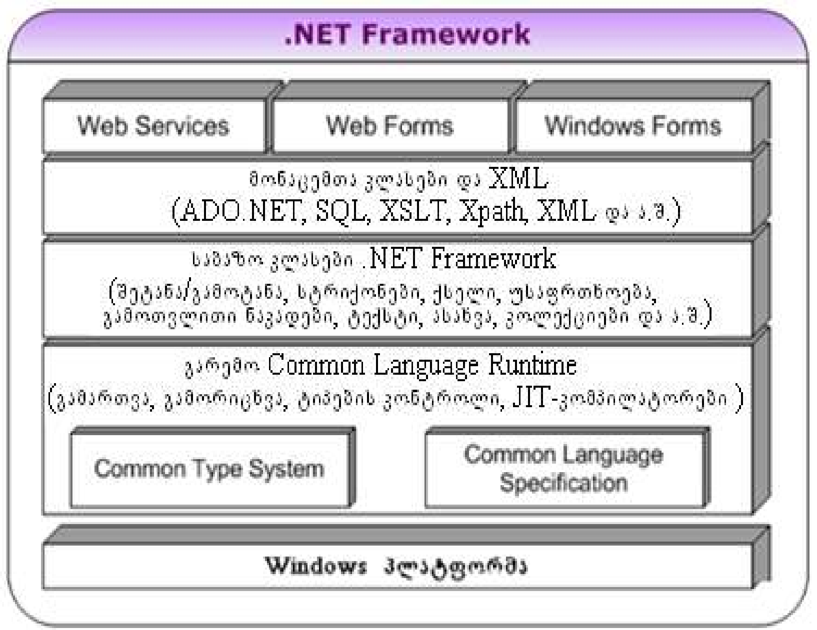


ნახ.1.5. .NET პლატფორმის კონცეფცია

Windows-სისტემა უშუალოდ მუშაობს C++, VB, Java და სხვა ენებზე დაწერილ პროგრამულ API-დანართებთან (Application Programming Interface), რომლებიც რეალიზებულია როგორც უმართავი კოდები (unmanaged code). ამასთანავე იგი მუშაობს C#.NET, C++.NET, VB.NET და ა.შ., ზოგადად .NET-პლატფორმის მიერ მართვად (managed code) პროგრამულ დანართებთან.

მართვაში იგულისხმება ის, რომ ეს კოდები ამუშავდება უშუალოდ .NET-ის მიერ, იმართება მათი პროცესები და მონაცემთა ნაკადები, მათ მიეწოდება შესასრულებლად საჭირო დამხმარე რესურსები და ა.შ.

პრინციპში, .NET-პლატფორმა ასრულებს “ოპერაციული სისტემის” გარკვეულ ფუნქციებს და მოქნილად ფუნქციონირებს Windows-თან. ამავე ნახაზზე საყურადღებოა თვით NET -პლატფორმის ბლოკი. რომელშიც ძირითადი ქვებლოკი Based Class.NET არის ამ პლატფორმის საბაზო კლასების ბიბლიოთეკა (უმრავლესობა დაწერილია C#-ენაზე). იგი სრულად ობიექტ-ორიენტირებულია, შედგება ობიექტთა ერთობლიობისაგან, რომელთაგანაც თითოეულში რეალიზებულია განსაზღვრულ მეთოდთა ჯგუფები. მაგალითად, ფანჯრებისა და ფორმების ასახვა (Windows GUI), მონაცემთა ფაილებთან ურთიერთობა (ADO.NET), ვებგვერდების ორგანიზება და ინტერნეტთან კავშირი (ASP.NET) და სხვ. 1.6 ნახაზზე ნაჩვენებია .NET Framework-ის არქიტექტურა.



ნახ.1.6. NET Framework-ის არქიტექტურა

ამავე ბლოკში ნაჩვენებია .NET-runtime – პლატფორმის სამუშაო გარემო (რომელშიც სრულდება პროგრამა), ანუ CLR(Common Language Runtime) და მას შესრულების საერთო გარემოსაც უწოდებენ. ესაა პროგრამული უზრუნველყოფა მომხმარებელთა გამოყენებითი პროგრამების შესასრულებლად.

CTS საერთო ტიპების სისტემაა (Common Type System), რომლის საფუძველზეც NET-პლატფორმა უზრუნველყოფს დაპროგრამების სხვადასხვა ენის თავსებადობას. ამასთანავე CTS აღწერს მომხმარებელთა კლასების განსაზღვრის წესებსაც.

IL შუალედური გარდაქმნის ენაა (Intermediate Language). პროგრამები, რომელთა საწყისი კოდები დაწერილია, მაგალითად, C#, C++ ან VB ენებზე .NET-ში, კომპილატორი ამ მართვად კოდებს გადაიყვანს შუალედურ IL-ენაზე, რომელთაც შემდეგ CTS სწრაფად აკომპილირებს მანქანურ კოდში. ამგვარად, ობიექტური კოდები IL-ენის საშუალებით ისე მიიღება, რომ მათში არაა დაფიქსირებული, თუ რომელ ენაზეა დაწერილი საწყისი კოდი.

CLS ენის საერთო სპეციფიკაციაა (Common Language Specification), ანუ იმ სტანდარტების მინიმალური ერთობლიობა, რომელიც უზრუნველყოფს კოდებთან მიმართვას .NET-ის ნებისმიერი ენიდან. ამ ენების ყველა კომპილატორს აქვს CLS მხარდაჭერა.

JIT (Just-In-Time) ესაა შუალედური კოდის კომპილაციის ფაზა მანქანურ კოდში. სახელწოდება მიუთითებს იმაზე, რომ კოდის მხოლოდ იმ ცალკეული ნაწილების კომპილაცია ხდება, რომლებიც საჭიროა პროგრამის შესასრულებლად დროის მოცემულ მომენტში.

.NET Framework -ში გამოიყენება 2-ეტაპიანი კომპილაცია:

1. ეტაპი: კომპილაცია MSIL-ენაში;
2. ეტაპი: “just-in-time” კომპილაცია უშუალოდ შესრულების პროცესში.

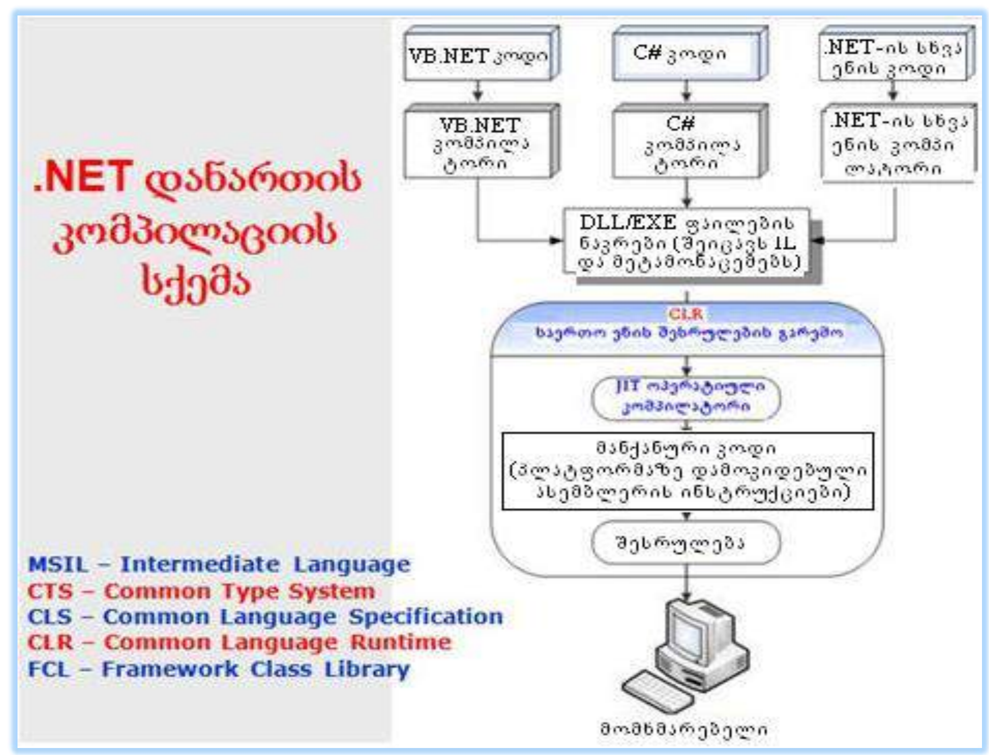
MSIL – ასემბლერული ენაა, რომელიც არაა დამოკიდებული მანქანაზე. ის სრულდება ყველგან, სადაც დაყენებულია CLR.

HTML-ისგან aspx-გვერდი განსხვავდება მასში სერვერული მართვის ელემენტების არსებობით, რომლებიც აღიწერება სპეც-ტეგებით. 1.6 ნახაზზე ნაჩვენებია სამომხმარებლო აპლიკაციის კომპილაციის სქემა .NET პლატფორმაზე.

C# („სი შარფ”) ენა ობიექტორიენტირებული ენების ერთ-ერთი ახალი და მძლავრი წარმომადგენელია, რომელიც შეიქმნა სპეციალურად NET-პლატფორმისათვის და თავსებადია Windows-ის თანამედროვე ვერსიებთან და ინტერნეტთან. ამ ენაზეა რეალიზებული NET-პლატფორმის უმრავლესი საბაზო კლასები.

როგორც ცნობილია, C++ ენა კომპილირდება ასემბლერულ კოდში, C# ენა კი - შუალედურ IL-ენაში.

IL-ენის დანიშნულებაა პლატფორმული და ენობრივი დამოუკიდებლობის განხორციელება ობიექტორიენტირებულ გარემოში. Java-ენაც უზრუნველყოფს პლატფორმულ (Windows, Unix, Linux) დამოუკიდებლობას, მაგრამ მისი ბაიტ-კოდის შესრულების ეტაპზე იგი ინტერპრეტირდება (IL - კი კომპილირდება).



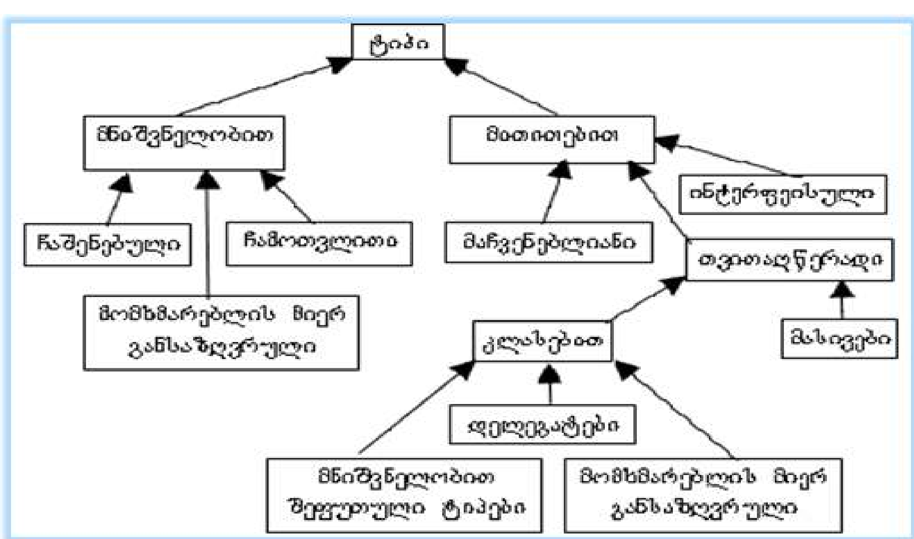
ნახ.1.6. პროგრამის კომპილაცია .NET პლატფორმაზე

1.3. .NET პლატფორმის ტიპების ზოგადი სისტემა

NET-პლატფორმისათვის ენობრივი თავსებადობა ხორციელ-დება IL ენაში არსებული ტიპების დიდი რაოდენობით, რომლებიც ორგანიზებულია ტიპთა იერარქიის ობიექტ-ორიენტირებული პრინციპებით. 1.7 ნახაზზე ნაჩვენებია ტიპთა ასეთი იერარქია მემკვიდრეობითობის კავშირის გამოყენებით.

მოვიტანოთ ზოგიერთი კომენტარი, რომელიც ახსნის ნახაზს:

- **ტიპი** არის საბაზო კლასი, რომელიც ზოგადად ასახავს ნებისმიერ ტიპს;
- **ტიპი მნიშვნელობით** საბაზო კლასია, რომელიც ზოგადად ასახავს ნებისმიერ ტიპს მნიშვნელობით;
- **ჩაშენებული ტიპები მნიშვნელობით** სტანდარტული საბაზო ტიპებია, რომლებიც აღწერს რიცხვებს, სიმბოლოებსა და ლოგიკურ მნიშვნელობებს;
- **ჩამოთვლით ტიპი** ჩამონათვალთა ერთობლიობაა, რომელშიც თითოეულ მნიშვნელობას შეესაბამება რიცხვითი მნიშვნელობა (0,1,... და ა. შ.) მისი მდებარეობის მიხედვით;



ნახ.1.7. ტიპების ზოგადი სისტემა

- **მომხმარებლის მიერ განსაზღვრული ტიპი** არის საწყის კოდში (მომხმარებლის პროგრამაში) აღწერილი ტიპები, რომლებიც ინახება მნიშვნელობებით (ესაა მაგალითად, სტრუქტურები);
- **ტიპი მითითებით** მონაცემთა ნებისმიერი ტიპია, რომელთანაც მიმართვა ხორციელდება მიმთითებლებით და ინახება ნაკადში;
- **თვითაღწერადი ტიპები** არის ტიპი, რომელიც ასახავს ინფორმაციას თავის შესახებ;
- **მასივები** არის ნებისმიერი ტიპი, რომელიც შეიცავს ობიექტების მასივს;
- **ტიპები კლასებით** თვითაღწერადი ტიპებია, რომლებიც არაა მასივები;
- **დელეგატები** ტიპებია, რომლებიც დამუშავდა მიმთითებელთა შესანახად კლასის მეთოდებისათვის;
- **მნიშვნელობით შეფუთული ტიპები** არის ტიპები მნიშვნელობით, რომლებიც დროებით დაიყვანება ტიპებამდე მიმთითებლით, რათა შენახულ იქნას ნაკადში;
- **მომხმარებლის მიერ განსაზღვრული ტიპები მითითებით** არის ტიპები, განსაზღვრული საწყის კოდში, როგორც ტიპები მითითებით. პროგრამაში ესაა მაგალითად, ნებისმიერი კლასი.

.NET პლატფორმის კომპონენტებს, მისი აპლიკაციის სტრუქტურას და პროცესებს მე-3 თავში დავუბრუნდებით.

1.4. დაპროგრამების მეთოდები, სტილი, მოდელი და ენა

„მეთოდი“ ბერძნული სიტყვაა (methodos) და ნიშნავს გზას, კვლევის გზას, თეორიას. ესაა მიზნის მიღწევის ხერხი კონკრეტული ამოცანის გადაწყვეტისას, გარკვეული წესების სიმრავლის სისტემატური გამოყენების საფუძველზე.

„პროგრამა“ ასევე ბერძნულია (program, routine) და გამოხატავს ოპერაციების შესრულების მიმდევრობის აღწერას წინასწარ ცნობილი ალგორითმის საფუძველზე.

დაპროგრამება არის პროგრამის შექმნის პროცესი. ის მოიცავს პროგრამის დაპროექტების, კოდირებისა და ტესტირების ეტაპებს. ამგვარად, დაპროგრამების მეთოდი მართვის პროცესია, რომლის დროსაც იქმნება პროგრამული პროდუქტი (პროგრამული პაკეტი).

პროგრამის ხარისხი და მუშაობის ეფექტურობა დამოკიდებულია როგორც დაპროგრამების მეთოდზე (ობიექტური ფაქტორი), ისე დამპროგრამებლის ცოდნაზე, გამოცდილებასა და ტემპერამენტზე (სუბიექტური ფაქტორი). პროგრამა შეიძლება ასრულებდეს თავის მოვალეობას, ხსნიდეს ამა თუ იმ ამოცანას, მაგრამ იგი იყოს ცუდად ან კარგად შედგენილი. „ცუდად“ ან „კარგად“ ნიშნავს იმას, თუ რამდენად გასაგებადაა პროგრამა დაწერილი (შინაარსობრივად), როგორია მისი გამოსახვის სტრუქტურა ანუ ფორმა.

დაპროგრამების პროცესის განხილვისას გამოყოფენ ოთხ ძირითად ცნებას: დაპროგრამების მეთოდს, დაპროგრამების სტილს, დამუშავების ალგორითმს (მოდელს) და დაპროგრამების ენას.

დაპროგრამების მეთოდი ასახავს პროგრამის შესრულების ხერხს, ტექნოლოგიას. მაგალითად, სტრუქტურული დაპროგრამება ნიშნავს პროგრამული პაკეტების შექმნას დაღმავალი დაპროექტებისა და პროგრამების მოდულური პრინციპების გამოყენებით.

მაგალითად, არსებობს სახლის აშენების ტრადიციული მეთოდი, დაწყებული საძირკვლიდან, პირველი, მეორე და ა.შ. სართულები და ბოლოს, სახურავი. არსებობს მეორე მეთოდიც, საძირკველი, შემდეგ სახურავი, ზედა სართული, ქვედა სართული და ბოლოს პირველი სართული.

რა თქმა უნდა, შენობის აგების ამ ორ შესრულებას განსხვავებული ტექნოლოგია სჭირდება. საბოლოო პროდუქტი შენობაა, რომელსაც არ ემჩნევა რა მეთოდით აშენდა, ე.ი. მეთოდი არ ემჩნევა პროდუქტს. ასევეა პროგრამაში. მისი წაკითხვით ძნელი დასადგენი იქნება, თუ რომელი მეთოდია გამოყენებული (მაგალითად, დაღმავალი თუ აღმავალი დაპროექტების).

ის, რაც შეიძლება შევამჩნიოთ პროგრამას ან შენობას, არის ფორმა, ანუ გამოსახვის, აღწერის სტილი. „სტილი“ ბერძნული სიტყვაა (stylus) და ჩხირს ნიშნავს, რომლითაც ძველად სანთლის დაფებზე წერდნენ (ჩხაპნიდნენ).

მშენებლობის სტილი: მაგალითად, სვეტიცხოველი (მე-10 საუკუნე) მართლმადიდებლური ეკლესიების სტილითაა აგებული, პარიზის ღვთისმშობლის ტაძარი (მე-12 საუკუნე) – გოთური სტილია (ბაზილიკა), რითაც კათოლიკური ეკლესიები გამოირჩევა. პროტესტანტული ეკლესია (მე-16 საუკუნე) სტილით არ ჰგავს თავის წინამორბედებს, არა აქვს გუმბათები, წმინდა ხატები და ქანდაკებები.

სტილის საშუალებით ხშირად განსაზღვრავენ მხატვრული ნაწარმოების ან სპორტული კლუბის ხელწერას. მაგალითად, პიკასოს სტილი (კუბიზმი, ნეოკლასიკური მიმდევრობა, მე-20 საუკუნე), ფეხბურთში – ბრაზილიური სტილი და ა.შ. ამგვარად, სტილია ის, რისი შემჩნევაცაა შესაძლებელი, სხვა ობიექტებთან შედარების გზით შესაძლებელია მისი გამორჩევა.

დაპროგრამების სტილი – ამბობენ, რომ დამპროგრამებელს აქვს ცუდი სტილი, თუ ის ხშირად გამოიყენებს GOTO ოპერატორს, ან მონაცემთა აღწერისათვის იყენებს ისეთ სიმბოლოებს და სახელებს, რომელთაც არა აქვს კავშირი შინაარსთან, დანიშნულებასთან.

პროგრამა შეიძლება აგებული იქნეს ოპერატორების გამოყენებით, მაშინ იგი *პროცედურული* სტილისაა (მაგალითად, C, Pascal და სხვ.). თუ პროგრამაში იყენებენ მხოლოდ ფუნქციებს, მაშინ საქმე გვაქვს *ფუნქციურ* სტილთან (მაგალითად, Lisp). შედეგების მისაღებად პროგრამა შეიძლება აგებული იქნეს ლოგიკური პროგრამების (წესების) სიმრავლით და ამ წესების დამუშავებისათვის ლოგიკური გამოყვანის ალგორითმებით. მაშინ ვლაპარაკობთ დაპროგრამების *ლოგიკურ* სტილზე (მაგალითად, Prolog). თუ დაპროგრამების დროს გამოიყენება ობიექტების ასახვის აბსტრაქტული ხერხი, რომლის საფუძველზეც ისინი შეიძლება გაერთიანებულ იქნას ერთ კლასში და აქვთ შესაძლებლობა გადასცეს თავიანთი თვისებები ახლად შექმნილ ობიექტებს, მაშინ საქმე გვაქვს ობიექტებზე ორიენტირებულ ანუ დაპროგრამების *ობიექტ-ორიენტირებულ* სტილთან (მაგალითად, C++, C#, Java, Python, Delphi და სხვ.).

ერთ პროგრამაში შესაძლებელია გამოყენებული იქნეს რამდენიმე სტილი, მაშინ ასეთ ენებს ჰიბრიდულს უწოდებენ. ენა მით უფრო მძლავრია, რაც მეტი სტილია მასში რეალიზებული. აღნიშნულ საკითხებს მომდევნო პარაგრაფებში უფრო დეტალურად შევეხებით. დავსვათ კითხვა: რომელი სტილია უკეთესი, რომელი სტილი შევარჩიოთ ამოცანის გადასაწყვეტად? სწორ პასუხს თვით კონკრეტული ამოცანის ანალიზი მოგვცემს. საჭიროა დადგინდეს ამოცანის მიზანი, შინაარსი, ძირითადი ელემენტების ნაირსახეობა და ბოლოს, რაც მთავარია, – ამ ამოცანის გადამუშავების მოდელი (ალგორითმი).

ამგვარად, თუ დაპროგრამების სტილი პროგრამის ფორმას გვაძლევს, გადამუშავების მოდელი ამ ფორმას შინაარსით ავსებს. სტილის მსგავსად, გადამუშავების სხვადასხვა მოდელი არსებობს. განვიხილოთ ზოგიერთი მათგანი.

– **გადამუშავების ტრადიციული მოდელი** ფონ-ნეიმანის (ამერიკელი მათემატიკოსი, 1903-1957) მანქანის სახელითაა ცნობილი. ამ მანქანის პრინციპით

მუშაობს დღემდე არსებული თითქმის ყველა კომპიუტერი. ესაა მიმდევრობითი, ბიჯური შესრულება ყოველი ოპერატორისა, რომელიც ცვლის მანქანის მდგომარეობას. იგი ფლობს მეხსიერებას და მონაცემთა დამუშავების ისეთ ოპერაციებს, როგორიცაა შედარება, გადაცემა, შეერთება, გამოყვანა (წარმოება), წაშლა და ა.შ.

– გადამუშავების ფუნქციური მოდელი, იგულისხმება მათემატიკური ფუნქციის ცნება, რომელიც გვაძლევს განსაზღვრის არის ცალსახა ასახვას მნიშვნელობათა არეზე. განსაზღვრისა და მნიშვნელობათა არეები არსებობს აბსტრაქტულად (სიმრავლის სახით), იმისგან დამოუკიდებლად, თუ რა მიმდევრობით აირჩევა წყვილები, რომლებიც ფუნქციურ დამოკიდებულებებს ასრულებს. მანქანას მიეწოდება ფუნქციის სახელი და არგუმენტები, რომლის საფუძველზე ის იპოვის, გადაამუშავებს და გამოსცემს მონაცემთა შესაბამის მნიშვნელობებს.

– გადამუშავების რელაციური მოდელი, იგულისხმება მათემატიკური დამოკიდებულებების, მიმართების ცნება. რელაციები განისაზღვრება როგორც ქვესიმრავლეები მონაცემთა სხვადასხვა დომენების დეკარტული ნამრავლიდან. ესაა ცხრილი სტრიქონებით (კორტეჟებით), რომელთა სვეტები მოთავსებულია ატრიბუტთა (ველების) სახელებით, ხოლო ცხრილის სტრიქონისა და სვეტის გადაკვეთაზე მოთავსებულია მონაცემის კონკრეტული მნიშვნელობა შესაბამისი დომენიდან (ერთგვაროვან მონაცემთა სიმრავლე). მანქანას მიეწოდება რელაციის სახელი და არგუმენტები, მაგალითად, მოსაძებნ ატრიბუტთა სახელები, რომლებიც წინასწარ განსაზღვრულ პრედიკატს, ლოგიკურ პირობას უნდა აკმაყოფილებდეს. იგი ამუშავებს ამ პრედიკატს და ეძებს რელაციაში შესაბამის მონაცემთა სტრიქონებს.

– მტკიცებათა მოდელი. იგი ეფუძნება დედუქციას და იყენებს ლოგიკურ ფორმულებს, რომლებიც ქმნის სიმრავლეს აქსიომათა სახით. ამათგან მტკიცდება, გამოიყვანება ახალი ფორმულები თეორემების სახით. მტკიცებათა თეორიის ფუძემდებელია გერმანელი მათემატიკოსი დავით გილბერტი (1862-1943), რომელმაც პირველმა შემოიტანა ეს ცნება და გამოიყენა მათემატიკური თეორიის, კერძოდ, აქსიომათა სისტემის არაწინააღმდეგობრიობის დასამტკიცებლად. მტკიცებათა მოდელის გამოსაყენებლად მანქანას უნდა შეეძლოს პირველი რიგის ლოგიკური პრედიკატების დამუშავების მექანიზმის რეალიზება.

– პრობლემათა გადამწყვეტი. იგი ამუშავებს პრობლემების სპეციფიკაციებს. პრობლემა აღიწერება განსაზღვრული ფორმით, ხოლო პრობლემათა გადამწყვეტი გამოიყვანს მისი ამოხსნის გზას. ხელოვნური ინტელექტის სისტემებში დღემდე გამოყენებულ ასეთი მექანიზმები ორიენტირებულია იმ მარტივი პრობლემების აღსაწერად, რომლებიც მოცემულია საწყისი სიტუაციების მიზნობრივი პრედიკატებისა და დასაშვებ ოპერატორთა ერთობლიობის ბაზაზე.

როგორც ზემოაღწერილ, ისე სხვა სახის დამუშავების მოდელების ანალიზი გვიჩვენებს, რომ უმრავლესი მათგანი მათემატიკური მოდელებია, მათ შორის

გარკვეული წილი ლოგიკურ მოდელებზე მოდის. ამ საკითხების შესწავლა და შემდგომი განვითარება დღესაც აქტუალურ მეცნიერულ მიმართულებად ითვლება, განსაკუთრებით კი მათემატიკური მოდელების სემანტიკური პრობლემების გადაწყვეტა.

ჩვენ გავეცანით დაპროგრამების მეთოდის, სტილისა და მოდელის ცნებებს. დავსვით ამოცანა (ზოგადად), განვსაზღვრეთ მისი გადაწყვეტის გზა. შევარჩიეთ დაპროგრამების მეთოდი და სტილი, მოვახდინეთ ამოცანის მოდელისა და მისი შესრულების ალგორითმის ფორმალიზება. დაგვრჩა ამ ალგორითმის კომპიუტერში გადატანა და შედეგების მიღება. ეს საკითხები წყდება დაპროგრამების ენებით.

დაპროგრამების ენები ფორმალური სისტემებია, რომელთა საშუალებით აღიწერება ამოცანის გადაწყვეტის ალგორითმები ისე, რომ შესაძლებელი ხდება მათი შემდგომი მანქანური გადამუშავება. ამგვარად, დაპროგრამების ენა ის ინსტრუმენტული საშუალებაა, რომლითაც ალგორითმი მანქანურ კოდებში ჩაიწერება.

ყოველ ენას და, მათ შორის, მანქანურსაც, აქვს სიმბოლოების ალფაბეტი, რომლის ბაზაზე აიწყობა ენის კონსტრუქციული ელემენტები: ოპერატორები (დაჯავშნულ სიტყვათა ერთობლიობა), ფუნქციები (სტანდარტული და არასტანდარტული), გამოსახულებები და ოპერაციები (არითმეტიკული და ლოგიკური), რელაციები, წესები და ა.შ. ყოველ ენაში რეალიზებულია ამ კონსტრუქციული ელემენტების მანიპულირების გრამატიკული წესები, მონაცემთა აბსტრაქტული სტრუქტურების აღწერის საშუალებანი, ოპერაციების, ფუნქციებისა და ქვეპროგრამების შესრულების მიმდევრობის მართვის საშუალებანი (განშტოებები, ციკლები, გადამრთველები, რეკურსიები) და ა.შ.

დაპროგრამების ყოველ ენას აქვს თავისი სტილი. შეიძლება ერთ ენაში დაპროგრამების რამდენიმე სტილი იყოს რეალიზებული, ამით იზრდება ენის სიმძლავრე და მოქნილობა. ენა ხშირად ორიენტირებულია გარკვეული კლასის ამოცანების გადაწყვეტაზე, მათ აბსტრაქციაზე. მაგალითად, Prolog ენა ლოგიკური დაპროგრამების ამოცანებს წყვეტს, Lisp – ფუნქციურ, C და Pascal სტრუქტურული დაპროგრამების კარგ ინსტრუმენტებად ითვლება, C++, C# და Python პროცედურული და დესკრიფციული ტიპის მონაცემთა მანიპულირების ისეთი ენები, როგორიცაა ალგებრული ენა ISBL (Information System Base Language), შეკითხვების ენა ეკრანული რედაქტორით QBE (Query By Example), SQL ან Sequel ენები, რომლებიც ალგებრულ და აღრიცხვის ენებს შორის მდგომი ენებია და ა.შ.

ბოლო ათწლეულში განსაკუთრებით განვითარება და ფართო გამოყენება პოვა ობიექტორიენტირებული დაპროგრამების ენებმა (C++, C#, Python), ფუნქციური (Clos) და ლოგიკური დაპროგრამების (Prolog++) კონცეფციებით.

ობიექტორიენტირებული ენები ორ ჯგუფად იყოფა:

1. დაპროგრამების ენების გაფართოებით მიღებული ჰიბრიდული ობიექტ-ორიენტირებული ენები, მათ მიეკუთვნება Simula-67, C++, Prolog++ და სხვ.;

2. ახლად შექმნილი, ე.წ. სუფთა ობიექტორიენტირებული ენები. მაგალითად, Smaltalk-80, Eiffel და ა.შ.

ორივე ჯგუფს თავისი დადებითი და უარყოფითი მომენტი აქვს გამოყენების თვალსაზრისით. მაგალითად, სუფთა ობიექტორიენტირებული ენები კომპაქტური და ადვილად ასათვისებელია, მაგრამ აკლია დაპროგრამების უნივერსალური ენების ბევრი თვისება (საშუალებები), გამოყენების არე ასევე შეზღუდულია. თავის მხრივ, ჰიბრიდული ენები მოცულობით დიდია, მაგრამ მათში შერწყმულია როგორც ძირითადი ენის (მაგალითად C) დადებითი მხარეები, ისე ახალი ობიექტ-ორიენტირებული ენების თვისებები. ეს კი განაპირობებს იმას, რომ C++/C# შეუძლია ფართო კლასის ამოცანების გადაჭრა. იგი ყოველგვარი გადაპროგრამების გარეშე თითქმის მთლიანად ამუშავებს იმ არსებულ სისტემებსაც, რომლებიც C-ზეა დაწერილი მის შექმნამდე.

1.5. დაპროგრამების პარადიგმები

დაპროგრამენის ენების ევოლუციური განვითარების საფუძველს დაპროგრამების პარადიგმებით ხსნიან [100]. ჩვენ წინა პარაგრაფში განვიხილეთ დაპროგრამების მეთოდები, სტილი, მოდელი და ენა. ეს საკითხები იკვეთება შინაარსობრივად პარადიგმის ცნებასთან.

არსებობს „დაპროგრამების პარადიგმების" ცნების განსაზღვრების სხვადასხვა ვერსია. მაგალითად:

„პარადიგმა არის დაპროგრამების სტილი პროგრამისტის განზრახვის აღსაწერად" (დ. ბობოროვი) [101];

„პარადიგმა არის მოდელი ან მიდგომა პრობლემის გადასაწყვეტად" (ბ. შრაივერი) [102*];

„პარადიგმები დაპროგრამების ენების კლასიფიკაციის წესებია გარკვეული პირობების შესაბამისად, რომელთა შემოწმება შესაძლებელია" (პ. ვეგნერი) [103];

„ესაა კონცეპტუალიზაციის ხერხი იმისა, თუ რას ნიშნავს 'გაანგარიშების წარმოება' და როგორ უნდა იქნას კომპიუტერზე გადასაწყვეტი ამოცანა სტრუქტურირებული და ორგანიზებული" (ტ. ბადი) [104] და ა.შ.

ამგვარად, თუ განვაზოგადებთ პარადიგმის სხვადასხვა განსზაღვრებას, შეიძლება ვთქვათ, რომ დაპროგრამების პარადიგმა, როგორც დასმული პრობლემის და მისი გადაწყვეტის საწყისი კონცეპტუალური სქემა, არის გრამატიკული ინსტრუმენტი ფაქტების, მოვლენებისა და პროცესების აღსაწერად.

დაპროგრამების ძირითადი პარადიგმები შემდეგია:

- ***იმპერატიული დაპროგრამება*** (imperative programming): განსაზღვრავს გამოთვლებს მიმდევრობითი ბრძანებების (ინსტრუქციების) სახით, რომლითაც იცვლება პროგრამის მდგომარეობა. ბრანებით მიღებული მონაცემები ინახება მეხსიერებაში და ხელმისაწვდომია. ასეთი ენებია: Assembler, Algol, Fortran, C, C++ და სხვ.;

- *პროცედურული დაპროგრამება* (procedural programming): განსაზღვრავს ბიჯებს, რომლებიც პროგრამამ უნდა შეასრულოს რათა მიაღწიოს სასურველ მდგომარეობას. მიმდევრობით შესასრულებელი ოპერატორები შესაძლებელია მოთავსდეს ქვეპროგრამებში. ეს პარადიგმა ტრადიციული კომპიუტერის არქიტექტურას შეესაბამება, რომელიც ფონ ნეიმანმა შემოგვთავაზა. ასეთი ენებია: Basic, Fortran, PL/1, Pascal, C და სხვ.;
- *სტრუქტურული დაპროგრამება* (structured programming): პროგრამა წარმოდგენილია იერარქიული სტრუქტურის ბლოკების (მოდულების) სახით. იგი მუშავდება დაღმავალი ტექნოლოგიით. არ გამოიყენება goto ოპერატორი. მმართველი სტრუქტურებია: მიმდევრობითობა, განშტოება და ციკლი. სტრუქტურული დაპროგრამების მიზანია პროგრამისტების შრომის ნაყოფიერების ამაღლება დიდი და რთული პროგრამული სისტემების შექმნის, გამართვის და მოდიფიკაციის დროს. ასეთი ენებია: C, Pascal, C++ (არ ვიყენებთ goto ოპერატორს) და სხვა ობიექტ-ორიენტირებული ენები;
- *დეკლარაციული დაპროგრამება* (declarative programming): განსაზღვრავს გამოთვლების ლოგიკას მისი მართვის ნაკადების განსაზღვრის გარეშე. ანუ, მოიცემა დასმული ამოცანის გადაწყვეტის სპეციფიკაცია (მოთხოვნებისა და პარამეტრების ერთობლიობა) თუ რა უნდა იყოს შედეგი. აქ არ ჩანს როგორ უნდა იქნეს ეს შედეგი მოღებული (იმპერატიულ დაპროგრამებაში ეს ჩანს). ასეთი ენის მაგალითებია SQL და HTML და სხვ.
- *ფუნქციონალური დაპროგრამება* (functional programming): განიხილავს გამოთვლებს როგორც მათემატიკური ფუნქციების შესრულების შედეგებს საწყისი მონაცემების ან სხვა ფუნქციების საფუძველზე. იგი არ ითვალისწინებს პროგრამის მდგომარეობის შენახვას (როგორც ეს იმპერატიულში იყო). ასეთი ენებია: Lisp, APL, Haskel, C++, F#.NET და სხვ.
- *ობიექტორიენტირებული დაპროგრამება* (object-oriented programming): მეთოდოლოგიაა, რომელიც პროგრამას განიხილავს როგორც ობიექტების ერთობლიობას, სადაც თითოეული მათგანი გარკვეული კლასის ეგზემპლარია. კლასი ინკაფსულირებულია, აქვს სახელი, თვისებები (კლასის წევრები), მეთოდები (კლასის ფუნქციები), რომელთა ინიცირება ხდება გარედან შემოსული შეტყობინების საფუძველზე [105]. კლასები ქმნის მემკვიდრეობით იერარქიას და აქვს პოლიმორფიზმის თვისება. ობიექტორიენტირებული ენა არაა ალგორითმული ენა ! ასეთი ენებია: C++, Java, C#, Python, Smalltalk, Ruby, Eiffel, PHP და სხვ. ჩვენ წიგნში ოო-დაპროგრამების ენებს დეტალურად განვიხილავთ;
- *მოვლენებზე ორიენტირებული დაპროგრამება* (event-driven programming): ახორციელებს პროგრამის შესრულებას მოვლენების საფუძველზე. ეს შეიძლება იყოს მომხმარებლის მოქმედება (კლავიატურა, მაუსი), სხვა პროგრამებიდან ან ოპერაციული სისტემიდან მიღებული შეტყობინება და ა.შ. პროგრამის ამოცანაა, გააანალიზოს მოვლენა

და აირჩიოს შესაბამისი დამმუშავებელი (event handler). ასეთი ენებია: Javascript, ActionScript, Visual Basic, Elm და სხვ.;

- ***ლოგიკური დაპროგრამება*** (Logic programming): ეყრდნობა თეორემების ავტომატურ მტკიცებას და დისკრეტული მათემატიკის იმ განყოფილებას, რომელიც შეისწავლის ინფორმაციის ლოგიკური გამოყვანის პრინციპებს მოცემული ფაქტებისა და გამოყვანის წესების საფუძველზე. მისი მაგალითებია პირველი ენა Planner და მისგან წარმოებული Popler, Conniver და Qlisp, ასევე Prolog ენა და მისი შვილები Mercury, Visual Prolog და Oz;
- ***ავტომატებზე ბაზირებული დაპროგრამება*** (Automata-based programming): არის დაპროგრამების პარადიგმა, რომლის გამოყენებისას პროგრამა ან მისი ფრაგმენტი მოიაზრება როგორც რაიმე ფორმალური ავტომატის მოდელი. დასმული ამოცანისგან დამოკიდებულებით, ავტომატბაზირებული დაპროგრამებისას შეიძლება გამოყენებულ იქნას სასრული ავტომატები, პეტრის ქსელები ან სხვა გრაფული მოდელები. შეიძლება აქ ვახსენოთ UML ენის Statechart Diagram, რომელიც სწორედ მდგომარეობათა დიაგრამის სახელითაა ცნობილი [25] და ა.შ.

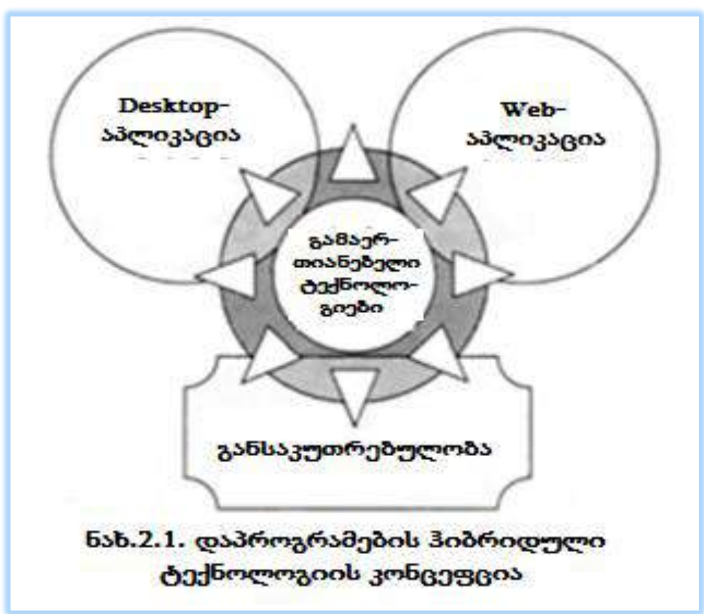
დაპროგრამების პარადიგმთა ეს ჩამონათვალი შეიძლება გაგრძელდეს და უფრო დაკონკრეტდეს, თითოეულის გამოყენება შესაბამის კლასთა ამოცანების გადაწყვეტას ემსახურება. უნდა აღინიშნოს, რომ დაპროგრამების ზოგიერთი თანამედროვე ენა **მულტიპარადიგმული** თვისების მატარებელია, ანუ მასში გამოყენებულია ერთდროულად დაპროგრამების რამდენიმე სტილი. მაგალითად, C++, C#, Python - ობიექტორიენტირებული, სტრუქტურული, ფუნქციური, იმპერატიული და სხვა პარადიგმთა მატარებელია. შეიძლება სტრუქტურული დაპროგრამების C ენაზეც დაიწეროს ობოექტორიენტირებული პროგრამები, მაგრამ ეს საგრძნობლად გაართულებს პროგრამულ კოდს.

II თავი
Windows- და Web-დაპროგრამების ტექნოლოგიები

აპლიკაციების (დანართების) ორი ნაირსახეობაა ცნობილი: ვინდოუს სისტემები, რომელთაც ასევე სამაგიდო აპლიკაციებს უწოდებენ და ვებ-აპლიკაციები, რომელთა გამოყენებაც ინტერნეტ ბრაუზერებიდანაა შესაძლებელი [16-18].

ეს დანართები იქმნება .NET Framework -ის ორი სხვადასხვა პაკეტით. პირველი - Windows Forms კომპონენტებით და მეორე ASP.NET -ის საშუალებით. ორივეს აქვს თავისი უპირატესობა და ნაკლოვანება. კერძოდ, სამაგიდო დანართები ძალზე მოქნილი და რეაქციულია, ხოლო Web-დანართები ინტერნეტის საშუალებით იძლევა დისტანციური წვდომის საშუალებას ერთდროულად მრავალი მომხმარებლისთვის. მაგრამ თანამედროვე კომპიუტერული ტექნოლოგიების სამყაროში ამ ორი სახის აპლიკაციებს შორის საზღვრები თანდათან იშლება [37].

Web-სამსახურების და WCF (Windows Communication Foundation) სერვის-ორიენტირებული არქიტექტურის აგების საშუალებების გაჩენამ განაპირობა სამაგიდო- და ვებ-დანართების ფუნქციონირების შესაძლებლობა ერთიან განაწილებულ გარემოში, სადაც მონაცემთა გაცვლა ხორციელდება როგორც ლოკალურ, ისე გლობალურ ქსელებში. 2.1 ნახაზზე ნაჩვენებია ეს გამაერთიანებელი პროცესი.



ნახ.2.1. დაპროგრამების ჰიბრიდული ტექნოლოგიის კონცეფცია

WPF – Windows Presentation Foundation ერთ-ერთი ასეთი გამაერთიანებელი ტექნოლოგიაა და აგებს ისეთ დანართს, რომელშიც გამორიცხულია დაპირისპირება

სამაგიდო აპლიკაციას და ინტერნეტს შორის. WPF-დანართს შეუძლია ფუნქციონირება როგორც სამაგიდო აპლიკაციის, ისე ვებ-აპლიკაციას ბრაუზერის შიგნით. არსებობს WPF-ის შეზღუდული ვერსია, სახელით Silverlight, რომლითაც შესაძლებელია ვებ-დანართში დინამიკური მდგენელის დამატება [106].

WF (Workflow Foundation) ტექნოლოგია .NET-ში არის სრულიად ახალი პარადიგმა სამუშაო (ბიზნეს) პროცესებზე (workflow) ბაზირებული აპლიკაციების ასაგებად. იგი ფუნდამენტურად ახლად გააზრებული ტექნოლოგიაა.

წიგნში ჩვენ გავეცნობით ორგანიზაციული მართვის სისტემების პროგრამული აპლიკაციების აგებისა და ექსპლუატაციის ამოცანების გადაწყვეტას მონაცემთა განაწილებული რელაციური ბაზის MsSQL_Server2012/14-ის საფუძველზე.

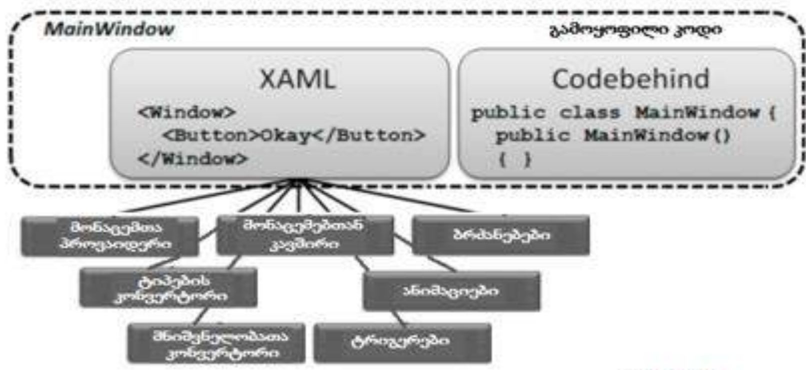
2.1. Windows Presentation Foundation ტექნოლოგია

პრაქტიკული სამუშაოების შესასრულებლად Windows Presentation Foundation (WPF) სისტემაში საჭიროა შემდეგი რესურსები:

➢ **ოპერაციული სისტემა** .Net Framework-ის 3.0 (და SP1) ვერსიას აქვს Windows XP, Windows Vista და Windows Server 2003-ის მხარდაჭერა. .Net Framework 4.0/4.5-ის გამოჩენის შემდეგ იგი ბევრად სრულყოფილი გახდა. ახალი ფუნქციონალობის გამოსაყენებლად სასურველია კონფიგურაციის გაუმჯობესება:

- Windows 7 ან Windows 10 (ან Windows Server 2008 R2);
- გრაფიკული კარტა DirectX-ის მე-9 ვერსიისთვის;

➢ **ინტეგრირებული სამუშაო გარემო:** .Net Framework 4.0/4.5. 2.2 ნახაზზე ნაჩვენებია ორი ძირითადი პროგრამული ტანდემი (XAML-თავისი მეთოდებით და კოდი).



ნახ.2.2

WPF (ადრე ცნობილი იყო სახელით Avalon) არის ტექნოლოგია, რომელიც საშუალებას იძლევა დაიწეროს პლატფორმაზე დამოუკიდებელი აპლიკაციები, დიზაინისა და ფუნქციონალური შესაძლებლობების ცხადი დაყოფით. იგი ეფუძნება ადრე არსებულ ისეთ ტექნოლოგიათა გაფართოებულ ცნებებს და კლასებს, როგორიცაა

Windows Forms, ASP.NET, XML, GDI+ და ა.შ. ასეთი მსგავსება კარგად ჩანს ვებდანართის აწყობისას .NET Framework გარემოში [1,2].

გარდა ამისა, WPF-ში მრავლადაა ახალი ფუნქციური საშუალებები პროგრამისტებისა და მომხმარებლებისთვის, რაც მისი, როგორც ახალი ტექნოლოგიის განხილვის უფლებას იძლევა. მაგალითად, Silverlight ტექნოლოგიას შეუძლია პროგრამები პირდაპირ ბრაუზერში შეასრულოს. ამ პლაგინის ინსტალაცია აუცილებელია. არაა დიდი მოცულობის, კომპაქტური. ისე როგორც WPF-ში, აქაც გამოიყენება XAML და C#. მაგრამ WPF პროგრამები უშუალოდ ბრაუზერში ვერ გაიშვება.

WPF მუშაობს ვექტორულ გრაფიკაზე ბაზირებულად და არა პიქსელებზე. მართვის ელემენტები, გრაფიკები, აგრეთვე ნახაზები არ იხაზება პიქსელებით, არამედ აღიწერება მრუდებითა და წრფეებით. ამის გამო აღარაა მონიტორებისგან ძლიერი დამოკიდებულება.

დეტალურად ეს ტექნოლოგია განხილული იქნება XV-XVI თავებში, კონკრეტული ამოცანებით და მათი გადაწყვეტის მეთოდებითა და ინსტრუმენტული საშუალებებით.

2.2. Workflow Foundation ტექნოლოგია

Windows Workflow Foundation (WF) ტექნოლოგია ვიზუალური დაპროგრამების მაღალი კლასის ინსტრუმენტული საშუალებაა [17,63]. იგი გამოიყენება ბიზნესპროცესების (Wowkflow) განსაზღვრებისა და მართვისათვის. ეს ტექნოლოგია შედის .NET Framework-ს შემადგენლობაში და იყენებს დაპროგრამების დეკლარაციულ მოდელს.

WF ინსტრუმენტული საშუალება ფლობს პროცესების დიზაინერს (process designer) და ვიზუალურ გამმართველს (visual debugger).

WF ტექნოლოგიით შესაძლებელია სამი ტიპის პროცესების აღწერა:

- მიმდევრობითი პროცესი (Sequential Workflow) – ერთი ბიჯიდან მეორეზე გადასვლა უკან დაბრუნების გარეშე;
- წესებით მართვადი პროცესი (Rules-driven Workflow) – ესაა მიმდევრობითი პროცესის კერძო შემთხვევა, რომელშიც გადასვლა შემდგომ ბიჯზე განისაზღვრება წესების ერთობლიობით;
- სასრული ავტომატი (State-Machine Workflow) – გადასვლა ერთი მდგომარეობიდან სხვა მდგომარეობაზე, ასევე შესაძლებელია ნებისმიერი უკან დაბრუნება წინა მდგომარეობებში.

დეტალურად ეს ტექნოლოგია განხილული იქნება XVII- XVIII თავებში, კონკრეტული ამოცანებით და მათი გადაწყვეტის მეთოდებითა და ინსტრუმენტული საშუალებებით.

2.3. Windows Communication Foundation ტექნოლოგია

Windows Communication Foundation (WCF) არის მაიკროსოფთის კომპანიის სერვისორიენტირებული საკომუნიკაციო პლატფორმა, რომელიც გამოიყენება განაწილებული სისტემების დასაპროგრამებლად. იგი აერთიანებს მრავალ ქსელურ ფუნქციას და სთავაზობს პროგრამისტებს ასეთი აპლიკაციების სტანდარტიზაციის საშუალებებს.

WCF-ის საშუალებით მოხერხდა საკომუნიკაციო ტექნოლოგიების DCOM, Enterprise Services, MSMQ, WSE და Web-Services გაერთიანება დაპროგრამების ერთიან ინტერფეისში [18,63,107]:

– DCOM (Distributed Component Object Model) გამოიყენება ქსელში სხვადასხვა კომპიუტერის ობიექტებს შორის კავშირის მხარდასაჭერად [108];

– Enterprise Services (საწარმოო სერვისები) გამოიყენება კორპორაციაში ბიზნეს-პროცესების ავტომატიზაციის მიზნით. იგი SAP-ის (System Analysis and Program Development – გერმანია) ტერმინია. აქ შეიძლება მოვიხსენიოთ ERP (Enterprise Resource Planing – ბრიტანეთი, აშშ და სხვ.) სისტემები [43], აგრეთვე ITIL მეთოდოლოგია, რომელიც IT-სერვისების მართვის საუკეთესო პრაქტიკაა [5];

– MSMQ (Microsoft Message Queuing) შეტყობინებათა რიგების მხარდამჭერი პროტოკოლია, რომელიც უზრუნველყოფს სერვერებს შორის ინფორმაციის (შეტყობინებების) უმტყუნო და უსაფრთხო გადაცემის რეალიზაციას;

– WSE (Web Services Enhancements) არის Ms.NET Framework-ის ვებსერვისების გაფართოება (დამატება). იგი შეიცავს კლასების ერთობლიობას, რომლითაც რეალიზებულია ვებსერვისების დამატებითი სპეციფიკაციები, ძირითადად ისეთ სფეროებში, როგორიცაა უსაფრთხოება, შეტყობინებათა საიმედო მიმოცვლა და დანართების (attachments) გადაცემა. WS-ები ბიზნესლოგიკის კომპონენტებია, რომლებიც გვთავაზობს ფუნქციურ შესაძლებლობებს ინტერნეტის საშუალებით. იგი იყენებს სტანდარტულ პროტოკოლებს, როგორიცაა, მაგალითად, HTTP.

კომუნიკაციისათვის ვებსერვისები იყენებს SOAP (Simple Object Access Protocol) პროტოკლს (სტრუქტურირებული შეტყობინებებისთვის ქსელში) ან REST (Representational state transfer – რეპრეზენტაციული მდგომარეობის გადაცემა) არქიტექტურულ სტილს ქსელში განაწილებული აპლიკაციის კომპონენტების ურთიერთქმედებისათვის [108,109].

.NET Framework -ის ბოლო ვერსიებში მოხდა ვებსერვისების გაერთიანება სხვა ტექნოლოგიებთან, კერძოდ Remoting-თან (დისტანციური ურთიერთმოქმედება). იგი იძლევა ობიექტთა ეგზემპლარების შექმნის საშუალებას ერთ და მის გამოყენებას სხვა პროცესებში.

ამგვარად, შეიქმნა .NET Framework-ის შემადგენელი ნაწილი WCF ტექნოლოგია, რომელიც საფუძვლიანად გახდა დანართებს (აპლიკაციებს) შორის მონაცემთა გაცვლის ზოგადი ინფრასტრუქტურა.

WCF-ის გამოყენებით შესაძლებელია გადასვლა მარტივი დანართებიდან ისეთ აპლიკაციებზე, რომლებიც იყენებს სერვისორიენტირებულ არქიტექტურებს (SOA service-oriented architecture). ეს პლატფორმა ნიშნავს, რომ შესაძლებელია დამუშავების პროცესების დეცენტრალიზაცია და მონაცემთა განაწილებული დამუშავება სერვისებთან მიერთების გზით ლოკალური ქსელებიდან ან ინტერნეტიდან.

WCF-ის კონცეფციიდან გამომდინარე შეიძლება განვიხილოთ მისი შემდეგი ასპექტები:

- WCF-ის კავშირის პროტოკოლები;
- მისამართები, საბოლოო წერტილები და მიმაგრებები;
- კონტრაქტები;
- შეტყობინებათა ნიმუშები;
- ყოფაქცევა;
- ჰოსტინგი.

WCF-ის სერვისებთან წვდომა შესაძლებელია სხვადასხვა სატრანსპორტო პროტოკოლით. .NET Framework 4.5-ში განსაზღვრულია ასეთი 5 საკომუნიკაციო პროტოკოლი:

– **HTTP** უზრუნველყოფს WCF-სერვისებთან კავშირს ინტერნეტით ნებისმიერი გეოგრაფიული წერტილიდან. HTTP კავშირებით შესაძლებელია WCF-ის ვებ-სერვისების შექმნა;

– **TCP** (Transmission Control Protocol) უზრუნველყოფს WCF-სერვისებთან კავშირს ლოკალური ქსელით ან ინტერნეტით, თუ ბრანდმაუერი აწყობილია შეაბამისად. TCP არის უფრო ეფექტური ვიდრე HTTP და აქვს მეტი შესაძლებლობები, მაგრამ აწყობის თვალსაზრისით რთულია. როცა ორ ჰოსტს შორის კავშირი დამყარებულია, მაშინ შესაძლებელია მონაცემთა გადაცემა ორივე მიმართულებით. საიმედოა, რადგან ხდება გადაცემის შემდეგ მიღების დადასტურება. თუ ეს მოხდა, ხელმეორედ გადაიცემა მონაცემები. გადაცემულ შეტყობინებათა მიმდევრობის მოწესრიგება დაცულია;

– **UDP** (User Datagram Protocol) არის TCP-ის მარტივი ალტერნატივა, არ მოითხოვს ჰოსტებს შორის კავშირის წინასწარ დამყარებას. კავშირი მყარდება ინფორმაციის გადაცემით ერთი მიმართულებით, კავშირის არსებობის წინასწარი შემოწმების გარეშე. შეიძლება სერვისმა ერთდროულად გადასცეს ინფორმაცია რამდენიმე კლიენტს. არ მიიღება დასტური გაგზავნილი შეტყობინების მიღებაზე, ის შეიძლება დაიკარგოს კიდევაც (ამგვარად, არასაიმედოა). ამასთანავე არ ხდება გადაცემულ შეტყობინებათა მიმდევრობის მოწესრიგება;

– **Named pipe** (სახელმინიჭებული არხი) უზრუნვეყოფს ერთ კომპიუტერში მონაცემთა გაცვლას სხვადასხვა პროცესს შორის. ჩვეულებრივ, ტრადიციული არხი „უსახელოა“ და არსებობს ანონიმურად მაშინ, როცა პროცესი სრულდება. (სახელმინიჭებული არხი კი არსებობს სისტემაში პროცესის დასრულების შემდეგაც. ამიტომ სასურველია მისი „გაუქმება“ როცა იგი აღარ გამოიყენება;

– **MSMQ** (Microsoft Message Queuing) არის რიგების ტექნოლოგია, რომელიც უზრუნველყოფს შეტყობინებათა გადაცემას რიგის შესაბამისად. ესაა შეტყობინებათა გაცვლის საიმედო ტექნოლოგია, რომელიც უზრუნველყოფს რიგში არსებული გადასაცემი ინფორმაციის ადრესატამდე მისვლას. იგი ასინქრონულია, ამიტომაც მომდევნო შეტყობინება დამუშავდება მას შემდეგ, რაც მისი წინა შეტყობინება რიგში უკვე დამუშავდა.

აღნიშნული პროტოკოლები ხშირად უზრუნველყოფს დაცული კავშირების დამყარებას.

WCF-ის სერვისებთან დასაკავშირებლად საჭიროა ცოდნა მისი ადგილმდება-რეობის შესახებ, ანუ საბოლოო წერტილის მისამართი.

მისამართის ტიპი სერვისისათვის დამოკიდებულია გამოყენებულ პროტოკოლზე. სერვისების მისამართები ფორმატირებულია სამი პროტოკოლისათვის. მაგალითად:

– HTTP პროტოკოლისათვის ესაა URL-მისამართები, როგორც ვიცით: http://<server> :<port>/<service>.

– TCP პროტოკოლისათვის მისამართები ასეთი ფორმისაა:

net.tcp://<server>:<port>/<service>.

– UDP პროტოკოლისათვის მისამართები შემდეგი სახისაა:

soap.udp://<server>:<port>/<service>.

ხშირად ასეთი <server> მნიშვნელობა საჭიროა შეტყობინების მრავალმისამართიანი გადაგზავნისათვის.

– Named pipe მისამართები დასაკავშირებლად ანალოგიურია წინა შემთხვევებისა, ოღონდაც მათ არ გააჩნია პორტის ნომერი. ფორმა ასეთია:

net.pipe://<server>/<service>.

მისამართი სერვისისათვის არის საბაზო მისამართი, რომელიც შეიძლება გამოყენებულ იქნას მისამართების შესაქმნელად **საბოლოო წერტილებისათვის**, რომლებიც წარმოადგენს ოპერაციებს. მაგალითად, შესაძლებელია ასეთი ოპერაციის არსებობა ქსელში:

.tcp://<server>:<port>/<service>/operation1.

დავუშვათ, საჭიროა WCF სერვისის შექმნა ერთი ოპერაციით, რომელსაც აქვს მიმაგრებები სამივე პროტოკოლთან. აქ შესაძლებელია შემდეგი საბაზო მისამართების გამოყენება:

http://www.mydomain.com/services/amazingservices/mygreatservice.svc
net.tcp://myhugeserver:8080/mygreatservice net.pipe://localhost/mygreatservice

შემდგომ შეიძლება ოპერაციებისთვის შემდეგი მისამართების გამოყენება:
http://www.mydomain.com/services/amazingservices/mygreatservice.svc/greatop
net.tcp://myhugeserver:8080/mygreatservice/greatop
net.pipe://localhost/mygreatservice/greatop

კონტრაქტები განსაზღვრავს, თუ როგორ შეიძლება იქნას გამოყენებული WCF სერვისები. კონტრაქტის რამდენიმე ტიპია:

– *მომსახურების კონტრაქტი* (Service contract) შეიცავს ზოგად ინფორმაციას მომსახურებასა და ოპერაციებზე, რომელსაც სერვისის სამსახური გასცემს. მაგალითად, ეს შეიძლება იყოს სერვისის მიერ გამოყენებადი სახელსივრცეები. სერვისებს აქვს უნიკალური სახელსივრცეები, რომლებიც გამოიყენება SOAP-შეტყობინებათა სქემის განსაზღვრისათვის, იმისათვის, რომ გამოირიცხოს შესაძლო კონფლიქტები სხვა სამსახურებთან;

– *ოპერაციის კონტრაქტი* (Operation contract) განსაზღვრავს, თუ როგორ გამოიყენება ოპერაცია. იგი შეიცავს პარამეტრებს და ოპერაციათა მეთოდების დასაბრუნებელ ტიპებს, ასევე დამატებით ინფორმაციას, მაგალითად, თუ იქნება საპასუხო შეტყობინება;

– *შეტყობინების კონტრაქტი* (Message contract) უზრუნველყოფს SOAP-შეტყობინებაში ინფორმაციის ფორმატირებას. მაგალითად, მონაცემები უნდა იყოს მოთავსებული SOAP-სათაურში თუ ინფორმაციის SOAP-ტანში. ეს მნიშვნელოვანია WCF სამსახურის შექმნის დროს, როდესაც იგი უნდა ინტეგრირდეს უკვე არსებულ სისტემებთან;

– *მცდარი კონტრაქტი* (Fault contract) განსაზღვრავს შეცდომებს, რომლებიც შეიძლება დააბრუნოს ოპერაციამ. .NET კლიენტების გამოყენებისას შეცდომები წარმოშობს გამონაკლის სიტუაციებს, რომლებიც ფიქსირდება და უნდა აღმოიფხვრას ჩვეულებრივი გზით;

– *მონაცემების კონტრაქტი* (Data contract). თუ გამოიყენება რთული ტიპები, მომხმარებელთა მიერ განსაზღრული სტრუქტურები და ობიექტები, პარამეტრები და დასაბრუნებელ ოპერაციათა ტიპები და ა.შ., ყველაფერი უნდა აისახოს მონაცემთა კონტრაქტში.

როგორც წესი, ამათ ამატებენ კიდევ სერვისთა კლასების და მეთოდების კონტრაქტს ატრიბუტების დახმარებით.

შეტყობინებათა ნიმუშები (Message Patterns) განიხილება სამი ტიპის:

– მოთხოვნა/პასუხის შეტყობინება (Request/response messaging) კლასიკური ხერხია ინფორმაციის გაცვლისათვის. ეს არ ნიშნავს, რომ მოთხოვნის გამგზავნი კლიენტი აუცილებლად ელოდება პასუხს (ასინქრონული ხერხი);

– ცალმხრივი ანუ სიმპლექსშეტყობინებათა გაცვლა (One-way, or simplex, messaging). შეტყობინებები გადაიგზავნება კლიენტიდან WCF ოპერაციისკენ, მაგრამ არავითარი პასუხი არ იგზავნება;

– ორმხრივი ანუ დუპლექსშეტყობინებათა გაცვლა (Two-way, or duplex, messaging) კლიენტსა და სეერვერს შორის.

ყოფაქცევა (Behaviors) არის დამატებითი კონფიგურაციის გამოყენების საშუალება, რომელიც არ ექვემდებარება უშუალო ზემოქმედებას კლიენტის სერვისებსა და ოპერაციებზე. როდესაც სერვისს ემატება ქცევა, მაშინ შესაძლებელია კონტროლირება იმისა, თუ როგორ კონკრეტიზდება (იქმნება ეგზემპლარი) და გამოიყენება მისი ჰოსტინგ პროცესით, როგორ მონაწილეობს იგი ტრანზაქციებში, როგორ განიხილება მრავალნაკადური საკითხები სერვისებში და ა.შ.

ჰოსტინგი (Hosting). როგორც აღინიშნა, WCF სერვისები შეიძლება განთავსებულ იყოს რამდენიმე სხვადასხვა პროცესში. ასეთი შესაძლებლობებია:

- Web server – ინტერნეტის საინფორმაცო სერვისები (IIS - Internet Information Server|Services). WCF სერვისები შეიძლება გაფართოვდეს ან ინტეგრირდეს IIS-ის უსაფრთხოების და სხვა ფუნქციური შესაძლებლობებით;
- Executable – შესაძლებებლია WCF სერვისის განთავსება ნებისმიერი ტიპის აპიკაციაში, როგორიცაა, მაგალითად, Console-, Windows Forms და WPF applications;
- Windows service – შესაძლებებლია WCF სერვისის განთავსება Windows სერვისში, რა ნიშნავს, რომ ხელმისაწვდომი იქნება ვინდოუსის ფუნქციების გამოყენება. მაგალითად, ავტომატური გაშვება (startup) და აღდგენა შეფერხებების შემდეგ (fault recovery);

Windows Activation Service (WAS) – ვინდოუსის აქტივაციის სერვისი დამუშავდა სპეციალურად WCF სერვისის განსათავსებლად, როგორც საბაზო მარტივი IIS. გამოიყენება იქ, სადაც მთლიანი IIS მიუწვდომელია.

დეტალურად ეს ტექნოლოგია განხილული იქნება XIX-XXI თავებში, კონკრეტული ამოცანებით და მათი გადაწყვეტის მეთოდებით, კლიენტ-სერვერული, ვებსერვისული და ინსტრუმენტული საშუალებებით.

2.4. XAML ენის საფუძვლები

მომხმარებელთა ინტერფეისების აგება WPF- და Silverlight-დანართებისთვის (აპლიკაციებისთვის) ხორციელდება XAML (Extensible Application Markup Language – აპლიკაციების გაფართოებადი ფორმატირების ენა) ენის გამოყენებით. XAML-დოკუმენტი შეიცავს ფორმატს, რომელიც აღწერს დანართის ფანჯრის (ან გვერდის) გარეგან სახეს და ქცევას, ხოლო მასთან კავშირში მყოფი C# კოდის ფაილები კი – დანართის ლოგიკას. XAML-ენა უზრუნველყოფს დანართის დიზაინის პროცესის (გრაფიკული ნაწილი) გამოყოფას ბიზნეს-ლოგიკის (პროგრამული კოდი) დამუშავების პროცესისგან, დიზაინერებსა და დეველოპერებს შორის [4,5].

WPF-ის XAML არის XML-ენის ქვესიმრავლე, გაფართოებული დამატებითი ფუნქციებით. იგი უზრუნველყოფს WPF-ის შიგთავსის აღწერას ისეთი ელემენტებით, როგორიცაა ვექტორული გრაფიკა, მართვის ელემენტები და დოკუმენტები.

XAML-ის საფუძველია XML და მისი სინტაქსი განისაზღვრება შემდეგი წესებით:

- XAML-დოკუმენტის ყოველი ელემენტი აისახება .NET კლასის რომელიღაც ეგზემპლარში. ასეთი ელემენტის სახელი ზუსტად შეესაბამება კლასის სახელს. მაგალითად, <Button> ელემენტი ემსახურება WPF-ინსტრუქციას Button-კლასის ობიექტის აგების მიზნით;
- XAML-ის ელემენტები შეიძლება ერთმანეთში ჩალაგდეს. ელემენტების ჩალაგების ფორმატი ასახავს ინტერფეისის ელემენტების ჩალაგებას;
- კლასის თვისებები განისაზღვრება ატრიბუტებით ან ჩალაგებული დესკრიპტორების დახმარებით, სპეცსინტაქსით.

XAML-ენა ხასიათდება თვითაღწერადობით. XAML-დოკუმენტში ყოველი ელემენტი არის ტიპის სახელი (მაგალითად, Button, Window ან Page) მოცემული სახელსივრცის ჩარჩოებში. ელემენტთა ატრიბუტები გამოიყენება შესაბამისი ობიექტების თვისებების (მაგალითად, Name, Height, Width და ა.შ.) და მოვლენების (Click, Load და ა.შ) მოსაცემად.

WPF-დანართის MyFirstWpfProject შექმნის დროს VisualStudio აგენერირებს შემდეგ XAML-დოკუმენტს:

```
<Window x:Class="MyFirstWpfProject.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="350" Width="525">
  <Grid>
  . . .
  </Grid>
</Window>
```

WPF-დანართის XAML-დოკუმენტი MyFirstWpfProject იწყება დესკრიპტორით < Window...>.

XAML-დოკუმენტის ყველა დესკრიპტორი იწყება „**<**" - სიმბოლოთი და მთავრდება „**>**" - სიმბოლოთი. ნებისმიერი XAML-დოკუმენტი შედგება XAML-ელემენტებისაგან. ყოველი XAML-დოკუმენტი (XAML-ელემენტი) იწყება გახსნის დესკრიპტორით (მაგალითად, < Window >), რომელსაც მოჰყვება დოკუმენტის შიგთავსი (მაგალითად, ტექსტური სტრიქონი ან სხვა XAML-ელემენტები). გახსნის დესკრიპტორში შეიძლება იყოს მოთავსებული ატრიბუტების აღწერა (მაგალითად, Class, xmlns, Title, Height, Width და სხვ.). XAML-დოკუმენტი (XAML-ელემენტი) უნდა დასრულდეს დახურვის დესკრიპტორით (მაგალითად, „/>" ან „</Window>").

XAML-დოკუმენტის ტექსტი უნდა შეიცავდეს ერთ ფესვურ ელემენტს - ჩალაგების უმაღლესი დონის ელემენტი. WPF-დანართის MyFirstWpfProject XAML-დოკუმენტში ასეთი ელემენტია <Window>. ფესვურ ელემენტში შეიძლება დაემატოს XAML-ის სხვა ელემენტებიც. მაგალითში ასეთი ელემენტია <Grid>.

WPF-დანართის XAML-დოკუმენტის კომპილაციის პროცესში სინტაქსურ ანალიზატორს გადაჰყავს XAML ფაილები აპლიკაციის ორობითი ფორმატირების ფაილებში BAML (Binary Application Markup Language), რომლებიც შემდეგ ჩაშენდება პროექტის ნაკრებში რესურსების სახით. WPF-დანართის კლასების ასაგებად სინტაქსური ანალიზატორი გამოიყენებს სახელსივრცეს, რომელიც განსაზღვრულია XAML-დოკუმენტის ფესვურ დესკრიპტორში.

XAML-დოკუმენტში სახელსივრცე მოიცემა xmlns ატრიბუტის საშუალებით. ზემოაღწერილ დოკუმენტში გამოცხადებულია ორი საბაზო სახელსივრცე:

- xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation – ესაა WPF-ის საბაზო სახელსივრცე, რომელიც მოიცავს WPF-ის ყველა კლასს, მართვის ელემენტების ჩათვლით. ისინი გამოიყენება მომხმარებლის ინტერფეისის ასაგებად. ვინაიდან სახელსივრცე ცხადდება პრეფიქსის გარეშე, იგი ვრცელდება მთელი XAML-დოკუმენტისათვის;
- xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" – ესაა XAML-ის სახელ-სივრცე. იგი შეიცავს XAML უტილიტების სხვადასხვა თვისებას, რომლებიც გავლენას ახდენს იმაზე, თუ XAML-დოკუმენტი როგორ ინტერპრეტირდება. მოცემული სახელსივრცე აისახება x პრეფიქსზე. ეს პრეფიქსი შეიძლება მოთავსდეს ელემენტის სახელის წინ (მაგალითად, x: ელემენტის-სახელი).

მეორე სახელსივრცე გამოიყენება XAML-ის სპეციფიური ლექსემების („საკვანძო სიტყვები") ჩასასმელად. 2.1 ცხრილში მოცემულია შედარებით ასეთი ხშირად გამოყენებადი სიტყვები.

XAML-ის საკვანძო სიტყვები		ცხრ.2.1
N	საკვანძო სიტყვა	დანიშნულება
1.	x:Array	.NET-ის მასივის ტიპი XAML-ზე
2.	x:Class	XAML-ფაილის კლასის სახელი
3.	x:ClassModifier	უზრუნველყოფს კლასის ტიპის ხილვადობის (internal ან public) განსაზღვრას, რომელიც Class საკვანძო სიტყვითაა აღნიშნული
4.	x:Code	პროგრამული კოდი შეიძლება უშუალოდ ჩაიდოს XAML-კოდში
5.	x:FieldModifier	უზრუნველყოფს ტიპის წევრის ხილვადობის (internal, public, private ან protected) განსაზღვრას ფესვის ნებისმიერი სახელმინიჭებული ელემენტისათვის (საკვანძო სიტყვით Name)
6.	x:Key	უზრუნველყოფს გასაღების მნიშვნელობის დაყენებას XAML ელემენტისათვის, რომელიც უნდა მოთავსდეს ლექსიკონის ელემენტში
7.	x:Name	უზრუნველყოფს C#-ით გენერირებული სახელის მითითებას მოცემული XAML ელემენტისათვის
8.	x:Null	წარმოადგენს null-მიმთითებელს
9.	x:Shared	შესაძლებელია მხოლოდ იმ რესურსებისთვის, რომლებიც ერთხელ იძლევა ეგზემპლარს. ჩვეულებრივად, ყოველი წვდომისას იწარმოება რესურსის ახალი ეგზემპლარი
10.	x:Static	უზრუნველყოფს ტიპის სტატიკურ წევრზე მიმართვას
11.	x:Subclass	ეს კონსტრუქცია ქმნის წარმოებულ კლასს და გამოდგება დაპროგრამების მხოლოდ იმ ენებისათვის, რომელთაც არ აქვს დანაწევრებული (partielle) კლასების მხარდაჭერა
12.	x:Type	XAML-ეკვივალენტია C#-ია typeof ოპერაციის (იძახებს System.Type მითითებული სახელის საფუძველზე)
13.	x:TypeArgument	უზრუნველყოფს ელემენტის დაყენებას, როგორც განზოგადებული ტიპისას განსაზღვრული პარამეტრებით
14.	x:Uid	x:Name –ს პარალელურად შეუძლია მართვის ელემენტს ამ ატრიბუტით უნიკალური სახელი მიიღოს, რომელიც შემდგომში თარგმანისთვის იქნება გამოყენებული

15	x:XData	ქმნის „მონაცემთა კუნძულს“ XAML-ის შიგნით, შეუძლია მარტივი XML-მონაცემთა კონსტრუქციით წარმოება
შენიშვნა: 2, 4, 9, 11, 14, 15		VS-2012 ვერსიაში დამატებული საკვანძო სიტყვები

WPF-დანართებში საბაზო სახელსივრცეთა გარდა ასევე იყენებენ შემდეგ სპეციალურ სახელსივრცეებს (რომლებიც არააუცილებელია):

- http://schemas.openxmlformats.org/markup-compatibility/2006 – XAML-ის სახელსივრცეა, დაკავშირებული ფორმატირების თავსებადობის პრობლემასთან სამუშაო გარემოსთან. ეს სახელსივრცე გამოიყენება XAML-ის სინტაქსური ანალიზატორის ინფორმირებისათვის იმის შესახებ, თუ რომელი ინფორმაციაა დასამუშავებელი და რომელი საიგნორირო;
- http://schemas.microsoft.com/expression/blend/2008 – XAML-ის სახელსივრცეა, რომელსაც აქვს მხარდაჭერა Expression Blend და Visual Studio პროგრამებიდან. გამოიყენება გვერდის გრაფიკული პანელის ზომების დასაყენებლად.

Window ობიექტის ატრიბუტებში შეიძლება დაემატოს შემდეგი XAML-აღწერები:

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="600"
```

მოცემული XAML-აღწერა აცხადებს არასავალდებულო სახელსივრცეებს პრეფიქსებით mc და d. თვისებები DesignHeight და DesignWidth იმყოფება სახელსივრცეში, რომელსაც აქვს პრეფიქსი d. ეს თვისებები განსაზღვრავს, რომ დანართის პროექტის დამუშავებისას Visual Studio დიზაინერში ფანჯარას უნდა ჰქონდეს ზომები 300x600. თვისება Ignorable მდებარეობს სახელსივრცეში, რომელიც აღნიშნულია პრეფიქსით mc და ის სინტაქსურ ანალიზატორს აინფორმირებს, რომ მან იგნორირება მოახდინოს XAML-დოკუმენტის ნაწილზე, რომელიც აღნიშნულია d პრეფიქსით.

WPF-დანართის XAML-დოკუმენტში ხშირად საჭიროა განხორციელდეს წვდომა პროექტის სხვა რომელიმე სახელსივრცესთან. ამ დროს აუცილებელია ახალი პრეფიქსის განსაზღვრა და მიეცეს სახელსივრცე. თუ პროექტში არის სახელსივრცე MyFirstWpfProject.Commands, მაშინ მის მიერთებას WPF -დანართის XAML-დოკუმენტთან ექნება შემდეგი სახე (command – გამოიყენება პრეფიქსის სახით).

```
xmlns:command="clr-namespace: MyFirstWpfProject.Commands"
```

პრეფიქსი (command) გამოიყენება მიმართვისთვის სახელსივრცეზე XAML-დოკუმენტში. clr-namespace ლექსემს ენიჭება სახელსივრცის დასახელება .NET ნაკრებში.

XAML-დოკუმენტში კლასის აღსაწერად გამოიყენება ატრიბუტი Class. XAML-დოკუმენტის სტრიქონი

```
<Window x:Class="MyFirstWpfProject.MainWindow" ...>
```

ითვალისწინებს MyFirstWpfProject.MainWindow კლასის შექმნას Window კლასის ბაზაზე. Class ატრიბუტის x პრეფიქსი განსაზღვრავს იმას, რომ ეს ატრიბუტი თავსდება XAML-ის სახელსივრცეში.

MainWindow კლასი გენერირდება ავტომატურად კომპილაციის დროს. კლასის ნაწილისთვის ავტომატურად გენერირდება კოდი (ნაწილობრივი (partial) კლასი):

```
namespace MyFirstWpfProject
{
  // <summary>
  // ურთიერთქმედების ლოგიკა MainWindow.xaml - ისთვის
  // </summary>
  public partial class MainWindow : Window
   {
    public MainWindow()
     {
       InitializeComponent();
      }
   }
}
```

როდესაც სრულდება დანართის კომპილაცია, XAML-ფაილი, რომელიც განსაზღვრავს მომხმარებლის ინტერფეისს (MainWindow.xaml), ტრანსლირდება CLR ტიპის გამოცხადებაში, რომელიც ერთიანდება დანართის ლოგიკასთან გამოყოფილი კოდის კლასის ფაილიდან (MainWindow.xaml.cs).

InitializeComponent() მეთოდი გენერირდება დანართის კომპილაციის დროს და საწყის კოდში არ თავსდება.

XAML-დოკუმენტში აღწერილი მართვის ელემენტების პროგრამულად სამართავად, აუცილებელია მართვის ელემენტს მიეცეს XAML ატრიბუტი Name. მაგალითად, Grid ელემენტისათვის ასე ჩაიწერება:

```
<Grid Name="grid">


</Grid>
```

მარტივი თვისებები XAML-დოკუმენტში მოიცემა შემდეგი სინტაქსის შესაბამისად:

თვისების_სახელი = "მნიშვნელობა"

მაგალითად, **Name = "grid1"**

თვისების მისაცემად, რომელიც არის სრულფასოვანი ობიექტი, გამოიყენება *რთული თვისებები* „თვისება-ელემენტი“ სინტაქსის შესაბამისად:

მშობელი.თვისების_სახელი

მაგალითად, StackPanel კონტეინერისთვის აუცილებელია მიეცეს გრადიენტული ფუნჯი პანელის შესავსებად, რაც განისაზღვრება Background ატრიბუტით. იგი რეალიზდება დესკრიპტორებით:

<StackPanel.Background> . . . </StackPanel.Background>.

თვისების მნიშვნელობის მისაცემად გამოყოფილი კლასიდან გამოიყენება ფორმატირების გაფართოება, რომელიც უზრუნველყოფს XAML გრამატიკის გაფართოებას ახალი ფუნქციონალობით. ფორმატირების გაფართოება შეიძლება გამოყენებულ იქნას ჩალაგებულ დესკრიპტორებში ან XAML-ატრიბუტებში. როცა იყენებენ ატრიბუტებს, მაშინ აუცილებელია ფიგურული ფრჩხილების {...} გამოყენება.

ფორმატირების გაფართოებები იყენებს შემდეგ სინტაქსს:

{ფორმატირების_გაფართოების_კლასი არგუმენტი}

ფორმატირების გაფართოებები რეალიზდება კლასებით, რომლებიც შვილობილია System.Windows.Markup.MarkupExtention კლასის. MarkupExtention საბაზო კლასს აქვს ProvideValue() მეთოდი, რომელიც იძლევა ატრიბუტისთვის საჭირო მნიშვნელობას. მაგალითად, იმისათვის, რომ Button-ობიექტის Foreground ატრიბუტს მიეცეს სტატიკური თვისება, რომელიც სხვა კლასშია განსაზღვრული, აუცილებელია შემდეგი XAML-აღწერის შექმნა:

```
<Button Foreground="{x:Static SystemColors.ActiveCaptionBrush}"
                              />
```

კომპილაციის დროს სინტაქსური ანალიზატორი შექმნის Static Extention კლასის ეგზემპლარს, შემდეგ გამოიძახებს ProvideValue() მეთოდს, რომელიც ამოიღებს საჭირო მნიშვნელობას და დააყენებს მას Foreground თვისებისთვის.

ფორმატირების გაფართოებები შეიძლება გამოყენებულ იქნას როგორც ჩალაგებული თვისებები.

მიერთებული თვისებები აღწერს თვისებებს, რომელთა გამოყენება შეიძლება რამდენიმე მართვის ელემენტთან, ოღონდ რომლებიც განსაზღვრულია სხვა კლასში. WPF-დანართებში მიერთებული თვისებები ხშირად გამოიყენება ინტერფეისის ელემენტების დაკომპლექტების სამართავად. მიერთებული თვისებების სინტაქსი შემდეგია:

განმსაზღვრელი_ტიპი.თვისების_სახელი

მაგალითად, თუ საჭიროა ღილაკის მოთავსება ბადის 0-ოვან სტრიქონში, მაშინ აუცილებელია შემდეგი XAML აღწერის შექმნა:

```
<Button ... Grid.Row="0" >
....
</Button>
```

აქ მიერთებული თვისებაა Grid.Row, ანუ Grid-ელემენტის Row-თვისება, რომელიც არაა Button ობიექტის თვისება. თვისება Row მიუერთდება Button ობიექტის თვისებებს, ვინაიდან ეს ობიექტი განთავსებულია Grid კონტეინერში.

ობიექტის ატრიბუტები შეიძლება გამოყენებულ იქნას მოვლენათა დამმუშავებლების მისაერთებლად, შემდეგი სინტაქსის გამოყენებით:

მოვლენის_სახელი = "მოვლენის_დამმუშავებლის_მეთოდის_სახელი"

მაგ., ღილაკის Click მოვლენისთვის (მისი დაჭერისას), შეიძლება დაყენდეს მოვლენის დამმუშავებელი Exit_Click.

```
<Button Name="Exit" Content="გამოსვლა" Click="Exit_Click" />
```

XAML-აღწერაში Exit_Click დამმუშავებლის განსაზღვრისას, აუცილებელია კლასის კოდში გვქონდეს მეთოდი კორექტული სიგნატურით. ქვემოთ მოყვანილია კოდი, რომელიც გენერირდება ავტომატურად მოვლენის დამმუშავებლის აღწერის შექმნისას XAML-დოკუმენტში.

```
private void Exit_Click(object sender,RoutedEventArgs e)
    {  . . .  }
```

III თავი
Ms Visual Studio.NET Framework ინტეგრირებული გარემო, აპლიკაციების სტრუქტურა და პროცესების მართვა

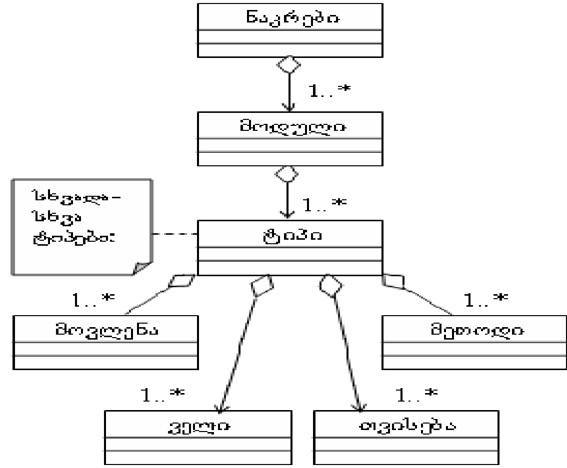
3.1. .NET პლატფორმის კომპონენეტები და მისი აპლიკაციების სტრუქტურა

როგორც 1-ელ თავში აღვნიშნეთ, .NET პლატფორმის საბაზო კლასების დიდი ნაწილი დაწერილია C# ენის გამოყენებით, ამიტომაც საილუსტრაციო მაგალითებს ამ ენაზე დავურთავთ კომენტარს.

ნაკრები (Assembly) .NET პლატფორმის კომპონენტებიდან ერთ-ერთი მთავარი ბლოკია (ანაწყობია), რომელიც ლოგიკურად აერთიანებს კოდს, რესურსებს და მეტამონაცემებს. იგი ლოგიკური და არა ფიზიკური ერთეულია, რადგან შეუძლია მოთავსდეს რამდენიმე ფაილში. ასეთ შემთხვევაში არსებობს ერთი მთავარი ფაილი, რომელშიც ინახება ინფორმაცია დანარჩენებზე.

3.1 ნახაზზე ნაჩვენებია პროგრამული დანართის (Application) შესაბამისი ნაკრების ზოგადი იერარქიული სტრუქტურა აგრეგაციის კავშირის გამოყენებით.

ნახაზიდან ჩანს, რომ ნაკრები ერთი ან რამდენიმე (1..*) მოდულისაგან (მოდულე) შედგება. სწორედ მოდულში ინახება დანართის ან ბიბლიოთეკის კოდი, მისი მეტამონაცემებით. მოდულები შეიცავს ტიპებს. ესაა კოდის შაბლონები (კლასები), რომლებშიც ინკაფსულირებულია გარკვეული მონაცემები და მეთოდები. როგორც წინა პარაგრაფში ვახსენეთ, ტიპები ორი სახისაა: მიმთითებლებით (ანუ კლასები) და მნიშვნელობებით (ანუ სტრუქტურები).



ნახ.3.1. ნაკრების (Assembly) ზოგადი სტრუქტურა

ტიპებს აქვს ველები, თვისებები და მეთოდები. ველი გამოყოფს მეხსიერების ადგილს შესაბამის მონაცემთა ტიპისათვის. თვისებები ველების მსგავსია, ოღონდაც მათი დანიშნულებაა შესაბამის მონაცემთა საწყისი მნიშვნელობების განსაზღვრა და კონტროლი.

მეთოდები განსაზღვრავს მონაცემთა დასამუშავებლად კლასის ქცევას, ანუ რეაქციას გარედან შემოსულ შეტყობინებაზე (მოთხოვნაზე). შეტყობინება ინაფორმაციაა, რომელიც ამა თუ იმ მოვლენის შედეგად ფორმირდება.

პროგრამული ნაკრები შეიძლება იყოს ორი ტიპის: კერძო და საერთო გამოყენების. პირველ შემთხვევაში ნაკრები ინსტალირდება კერძო მომხმარებლს კატალოგში და მასთან სხვა მიმართვა გამორიცხულია.

საერთო გამოყენების ნაკრები შეიცავს პროგრამულ ბიბლიოთეკებს, რომელთაც იყენებს სხვადასხვა დანართი. აქ საჭიროა სპეციალური დაცვის მექანიზმების გამოყენება (სახელების კოლიზიისა და ნაკრებთა ვერსიების კონტროლის თვალსაზრისით).

კლასებს შორის სახელთა კოლიზიის აღმოფხვრის მიზნით .NET პლატფორმა იყენებს "სახელთა სივრცეს".

სახელსივრცე (namespace): ესაა მონაცემთა ტიპების უბრალო დაჯგუფება. ყველა მონაცემთა ტიპის სახელს მოცემულ სახელთა სივრცეში ავტომატურად ემატება პრეფიქსი, რომელიც შედგენილია სახელსივრცის დასახელებისაგან. ასევე შესაძლებელია ჩადგმული სახელსივრცეების შექმნა.

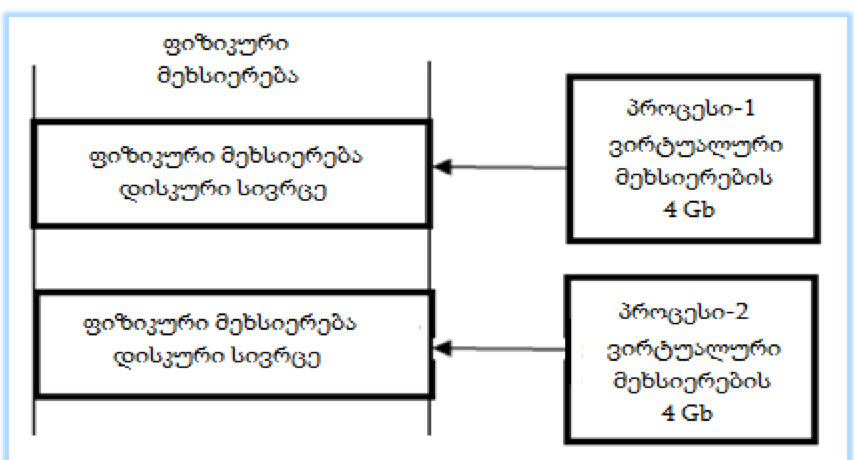
მაგალითად, საბაზო კლასების უმრავლესობისათვის, რომლებიც ზოგადი გამოყენებისთვისაა დანიშნული, მოთავსებულია სახელთა სივრცეში System, ვებგვერდებისათვის - System.Web და ა.შ.

C#-ის პროგრამის ტექსტის მაგალითზე შეიძლება შემდეგი კომენტარის გაკეთება:

```
namespace Magazia.Web // აქ სახელია Magazia.Web
{
   public class Checkout : PageBase
   {
      // და ა.შ.
```

დანართთა არეები (application area) არის .NET პლატფორმის მნიშვნელოვანი ელემენტი. მათი დანიშულებაა ერთდროულად და ერთმანეთთან მომუშავე დანართების იზოლაცია, რათა არ მოხდეს მონაცემთა არასასურველი დამუშავება.

პროგრამული დანართების იზოლაციისათვის Windows გამოიყენებს „**პროცესის**" ცნებას, რომელიც მისამართების სივრცეს ეხება. ყოველ პროცესს გამოეყოფა 4 გიგაბაიტი ვირტუალური მეხსიერება. ისინი დისკოზე სხვადასხვა ფიზიკური მისამართებითაა და არ გადაიკვეთება (ნახ.3.2).



ნახ.3.2. აპლიკაციათა არეების იზოლაცია

პროცესებს აქვს მინიჭებული განსაზღვრული პრივილეგიები და ოპერაციული სისტემა აკონტროლებს მათ, თუ რომელ ოპერაციას რომელი პროცესის გამოყენება შეუძლია.

დანართთა არეების გამოყენების იდეა მდგომარეობს იმაში, რომ პროცესებს შორის მოხერხდეს მონაცემთა გაცვლა. ამიტომაც პროცესი იყოფა რამდენიმე დანართის არედ. თითოეულ დანართის არეში თავსდება ერთი დანართის კოდი.

.NET პლატფორმის მნიშვნელოვანი საშუალებაა JIT (Just-In-Time) კომპილატორი. იგი ახორციელებს პროგრამული კოდის ცალკეული ნაწილის დროულად კომპილირებას (საჭიროებისამებრ).

Visual Studio.NET არის პროგრამული სისტემების დამუშავების ინტეგრირებული გარემო, რომელშიც შესაძლებელია კოდების დაწერა, კომპილირება და გამართვა VB.NET, C++.NET, C#.NET, ASP.NET, ADO.NET და სხვა ტექნოლოგიით.

3.2. .NET პლატფორმის აპლიკაციები

მაიკროსოფთის .NET Framework პლატფორმაზე შესაძლებელია სხვადასხვა ტიპის დანართის (აპლიკაციის) დამუშავება. ასეთი პლატფორმის სიმძლავრე და მოქნილობა იმაშიც გამოიხატება, რომ განსხვავებული ტიპის აპლიკაციებისათვის გამოყენებადია სისტემის საბაზო კლასების ერთიანი ბიბლიოთეკა (სტრიქონებთან, გრაფიკასთან და მათემატიკურ ფუნქციებთან სამუშაოდ, მონაცემებთან წვდომისათვის, ფაილებთან სამუშაოდ, კრიპტოგრაფიული ოპერაციების შესასრულებლად, მონაცემთა სინქრო-ნიზაციისათვის და ა.შ.).

განასხვავებენ .NET პლატფორმის შემდეგი ტიპის დანართებს:

- სამაგიდო აპლიკაციები (Desktop Applications) – მუშაობს მომხმარებლის ლოკალურ კომპიუტერზე;
- ვებდანართები (Web Appliocations) – მუშაობს ვებ-სერვერის ფარგლებში და წვდომადია მომხმარებლისთვის ბრაუზერის საშუალებით;
- ვებდანართები მომხმარებლის მდიდარი ინტერფეისით (Rich Internet Applications, RIA) – მიეწოდება მომხმარებელს HTTP/HTTPS პროტოკოლით ბრაუზერის ფარგლებში და სრულდება კლიენტის მხარეს;
- ვებსერვისები (Web Services) – პროგრამული კოდები, რომლებიც სრულდება სერვერის მხარეს და შეიძლება გამოძახებულ იქნას კლიენტისაგან რაიმე მონაცემების მისაღებად ან ოპერაციის შესასრულებლად;
- მობილური დანართები (Mobile Applications) – სრულდება მობილურ მოწყობილობებზე.

სამაგიდო აპლიკაცია ყველაზე გავრცელებული ტიპის დანართია. მას შეუძლია მიმართვა მომხმარებლის კომპიუტერის რესურსებთან (ვინჩესტერი, მულტიმედიური მოწყობილობები და სხვ.). ეს აპლიკაციები იყოფა სამი სახის დანართებად: ფანჯრული (Windows Forms Application, Windows Presentation Foundation), კონსოლური (Console Aoolication) და სერვისული (Windows Sevice). პირველს აქვს გრაფიკული ინტერფეისი, დანარჩენ ორს – არა.

ვებდანართები განსხვავდება სამაგიდო აპლიკაციებისაგან იმით, რომ ისინი მუშობს ვებსერვერებისგან დაცილებით. მომხმარებელი იყენებს ბრაუზერს და HTTP/HTTPS პროტოკოლს. უპირატესობა არის ის, რომ აპლიკაცია ყენდება სერვერზე, რომლის გამოყენებაც შეუძლიათ ქსელში ჩართულ კლიენტ მომხმარებლებს. ნაკლია ის, რომ მომხმარებლის ინტერფეისი შეზღუდულია (HTML, CSS და JavaScript-ის ფორმატების შეზღუდულობის გამო). მაიკროსოფთის .NET Framework პლატფორმაზე ვებდანართების აგება ხდება ASP.NET-ის (ASP.NET Web Application) და მის საფუძველზე განვითარებული ტექნოლოგიებით (Silverlight, MVC, MVVM და სხვ., მათ მოგვიანებით განვიხილავთ დეტალურად).

მომხმარებლის ინტერფეისის სრულყოფის მიზნით შეიქმნა ახალი ტიპის დანართი RIA (Rich Internet Applications). იდეა ისაა, რომ ბრაუზერში ინტეგრირებულია სპეციალური პლაგინი, რომელსაც შეუძლია ასახოს შინაარსის (შიგთავსის) დამატებითი ტიპი. როდესაც მომხმარებელი ხსნის ბრაუზერის გვერდს, კლიენტის მხარეს გადაეცემა პროგრამული კოდი, რომელიც ამ პლაგინით მუშაობს. ამგვარად, კლიენტს მიეწოდება მდიდარი შესაძლებლობები თავისი ინტერფეისის ასაგებად. ასეთი ტექნოლოგებია, მაგალითად, Adobe Flach, Ms Silverlight და ა.შ.

ვებსერვისული აპლიკაცია არის პროგრამული კოდი, განთავსებული სერვერზე და გამოიძახება კლიენტის მოთხოვნის საშუალებით. მაგალითად, ასეთი სერვისების ერთობლიობა შეიძლება ემსახურებოდეს სერვერზე მოთავსებულ მონაცემთა ბაზასთან

მუშაობას. სამაგიდო- და ვებ-აპლიკაციები ხშირად მიმართავს სერვისებს რაიმე ოპერაციის შესრულების მიზნით ან სერვერიდან მონაცემების მისაღებად.

სერვისების შესაქმნელად .NET Framework პლატფორმაზე არსებობს რიგი ტექნოლოგიებისა. მათ შორისაა ASP.NET Web Services, რომელიც ქმნის მარტივ ვებ-სერვისებს და მუშაობს HTTP/HTTPS პროტოკოლით. შედარებით ახალი და მოქნილი ტექნოლოგიაა Windows Communication Foundation (WCF), რომელიც იყენებს განსხვავებული ტიპის არხებს (HTTP, TCP, სახელდებული არხები და სხვ.). ეს საგრძნობლად აფართოვებს დეველოპერის შესაძლებლობებს სერვისების შესაქმნელად.

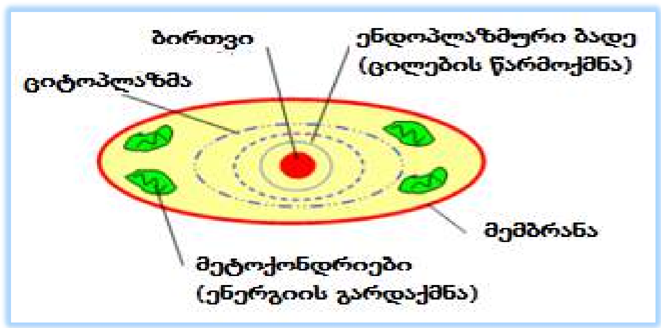
მობილური დანართები ფუნქციონირებს მობილურ მოწყობილობებზე Windows Mobile ოპერაციული სისტემის საფუძველზე. .NET Framework პლატფორმას აქვს აგრეთვე .NET Compact Framework შესაძლებლობათა ქვესიმრავლე, ოპერაციული სისტემებით: Windows, Android და iOS.

3.3. ობიექტორიენტირებული დაპროგრამების მეთოდი: კლასები და ობიექტები

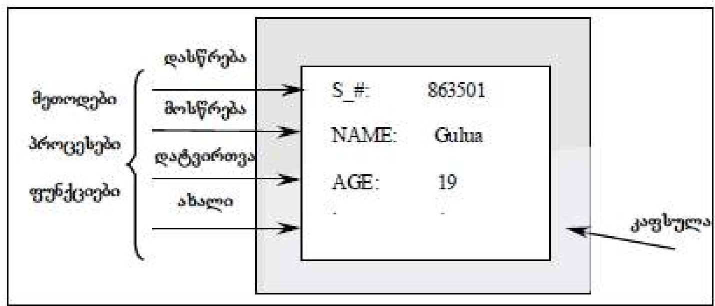
.NET გარემოში დაპროგრამება დაფუძნებულია ობიექტებზე. ობიექტი (object) – არის პროგრამული კონსტრუქცია, რომელშიც ინკაფსულირებულია ლოგიკურად დაკავშირებული მონაცემებისა და მეთოდების ერთობლიობა.

ობიექტი (Object) განიხილება, როგორც გარკვეული არსი (Entity), რომელიც ხასიათდება მდგომარეობით (მონაცემთა ერთობლიობა) და ქცევით (ფუნქციური პროგრამები). ობიექტის ქცევა ანუ რეაქცია, რომლის დროსაც მისი ახალი მდგომარეობა განისაზღვრება, დამოკიდებულია გარედან მოსულ ინფორმაციაზე, შეტყობინებებზე. ვინაიდან ობიექტი უმთავრესი ცნებაა, იგი საწყისია ობიექტორიენტირებული დაპროგრამებისა, ამიტომ დიდი მნიშვნელობა აქვს მის სწორად გაგებას. განვიხილოთ კლასისა და ობიექტის არსი ბიოლოგიური და ინფორმაციული უჯრედების მოდელების შედარების საფუძველზე (ნახ.3.3, 3.4).

ბიოლოგიური უჯრედი კაფსულირებულია მემბრანის (გარსის) საშუალებით, რომლითაც ის გამოიყოფა სხვა უჯრედებისა და გარემოსაგან. მას უნარი აქვს გარემოსგან მიიღოს ქიმიური სახის ინფორმაცია და უჯრედს შიგნით გადასცეს. შუაგულში მოთავსებულია ბირთვი, რომელიც უჯრედის ძირითადი ინფორმაციის მატარებელია. იგი შედგება ქრომოსომებისგან, რომლებიც გარკვეული გენეტიკის მქონეა. უჯრედის დაყოფის (გამრავლების) დროს ხდება მემკვიდრული თვისებების გადაცემა. ბირთვის გარშემო დაჯგუფებულია სხვადასხვა ფუნქციის ელემენტები. მაგალითად, ენდოპლაზმური ბადე – ცილების წარმოქმნის ფუნქცია, მიტოქონდრები – ენერგიის გარდაქმნის ფუნქცია, ციტოპლაზმა (თხევადი მოძრავი გარემო) – ტრანსპორტირების ფუნქცია და ა.შ.



ნახ.3.3. ბიოლოგიური უჯრედი



ნახ.3.4. ინფორმაციული უჯრედი

როგორია „ინფორმაციული უჯრედის" აგებულება და რა ანალოგია აქვს ბიოლოგიურთან? 3.4 ნახაზზე განიხილება ობიექტი-სტუდენტი, რომელიც რეალური სამყაროს ნაწილია. იგი კაფსულირებულია, რომლის შიგნითაც მოთავსებულია ბირთვი ობიექტის თვისებების აღმწერ მონაცემთა ელემენტები S =(studentis nomeri), NAME (gvari, saxeli), AGE (asaki) და ა.შ.

ობიექტები ავტონომიური ელემენტებია, ისინი იქმნება შაბლონით (template), რომელსაც ობიექტორიენტირებულ დაპროგრამების თეორიაში კლასს (class) უწოდებენ.

.NET-ის საბაზო კლასების ბიბლიოთეკა არის კლასთა ერთობლიობა, რომელიც გამოიყენება ობიექტების შესაქმნელად მომხმარებლის კერძო აპლიკაციაში. ამასთანავე პროგრამული პაკეტი Visual Studio საშუალებას იძლევა შეიქმნას საკუთარი კლასები კერძო პროგრამებისთვის.

- **ობიექტები, წევრები და აბსტრაჰირება:**

ობიექტი – პროგრამული კონსტრუქციაა, რომელიც ასახავს განსაზღვრულ არსს (Entity). ესაა რეალურ სამყაროში არსებული ობიექტები, მაგალითად, ადამიანები, მანქანები, ფირმები, ცხოველები, ფრინველები, მცენარეები, ინსტიტუტები,

კომპიუტერები და ა.შ. ყოველ ობიექტს აქვს განსაზღვრული, მისთვის დამახასიათებელი ფუნქციურობა და თვისებები.

აპლიკაციაში ობიექტი შეიძლება იყოს ფორმა, მართვის ელემენტი (ღილაკი, კომბობოქსი, ბაზასთან შეერთება და სხვ.).

ამგვარად, ობიექტი დასრულებული ფუნქციური ერთეულია, რომელიც შეიცავს ყველა მონაცემს და ყველა ფუნქციას, რომლებიც აუცილებელია იმ ამოცანის გადასაწყვეტად, რომლისთვისაც ეს ობიექტი გამოიყენება.

რეალური სამყაროს ობიექტების წარმოდგენას (ასახვას) პროგრამული ობიექტებით უწოდებენ აბსტრაჰირებას (abstraction).

- **კლასები, როგორც ობიექტთა შაბლონები:**

კლასი არის ერთგვაროვან ობიექტთა ერთობლიობა, რომელიც ამ ერთობლიობის სტრუქტურას ასახავს. კლასი განსაზღვრავს ობიექტთა წევრებს და მათ ყოფაქცევას, აგრეთვე საწყის მონაცემთა მნიშვნელობებს, საჭიროებისამებრ.

კლასის ეგზემპლარის შექმნისას კომპიუტერის მეხსიერებაში შეიქმნება ამ კლასის ასლი. ასეთი სახით შექმნილ კლასის ეგზემპლარს უწოდებენ ობიექტს. დაპროგრამების ენაში მისი შექმნა ხდება ოპერატორით new. მაგალითად:

```
  // gamocxaddes cvladi MyDataForm  tipiT DataForm
DataForm  MyDataForm;
  // Seiqmnas obieqtis egzemplari DataForm da
  // Caiweros igi cvladSi MyDataForm
MyDataForm = new DataForm( );
```

- **ობიექტები და წევრები:**

ობიექტები შედგება წევრებისაგან, რომელთაც ეკუთვნის თვისებები, ველები, მეთოდები და მოვლენები. ისინი განსაზღვრავს ობიექტის მონაცემებს და ფუნქციურობას.

ველები და თვისებები შეიცავს ობიექტის მონაცემებს, რომლებიც მის მდგომარეობას ასახავს. მეთოდები განსაზღვრავს მოქმედებებს, რომელთა შესრულება შეუძლია ობიექტს. მოვლენები კი წარმოადგენს შეტყობინებებს, რომელთაც მიიღებს ობიექტი ან გადასცემს სხვა ობიექტებს. მოვლენის დანიშნულებაა, გაააქტიუროს ობიექტის ესა თუ ის მეთოდი, რომელიც გადაამუშავებს მის მონაცემებს.

მაგალითად, ობიექტისთვის „ავტომობილი" თვისებები და ველებია: მოდელი, ფერი, ასაკი, საწვავის-ხარჯი და ა.შ. ეს მონაცემები ასახავს ობიექტის მდგომარეობას. მეთოდის მაგალითებია: სიჩქარის-გადართვა, დამუხრუჭება, საჭის-მობრუნება და სხვ., ისინი ობიექტის ქცევას განსაზღვრავს. ავტომობილისთვის მოვლენები მიიღება შეტყობინების სახით გარედან (მაგალითად, ინფორმაცია გზის მოსახვევის შესახებ, ან სიჩქარის შეზღუდვის შესახებ) ან მანქანის მდგომარეობის ამსახველი ნათურებიდან (წყლის გადახურება, საწვავის დამთავრება და ა.შ.).

- **ობიექტური მოდელები:**

მარტივ ობიექტებს აქვს რამდენიმე თვისება და მეთოდი, ასევე ერთი-ორი მოვლენა. რთულ ობიექტებს გააჩნია მეტი რაოდენობა თვისებების, მეთოდებისა და მოვლენების, აგრეთვე მათ შეიძლება ჰქონდეს "შვილი" ობიექტებიც, რომლებზეც შეუძლია პირდაპირი მიმართვის განხორციელება. მაგალითად, მართვის ელემენტს TextBox, აქვს თვისება Font, რომელიც არის Font-ტიპის ობიექტი. ამ თვალსაზრისით, ნებისმიერი Form კლასის ეგზემპლარი მოიცავს მასზე განთავსებულ მართვის ელემენტებს (Controls ერთობლიობა).

„მშობელი-შვილი" ჩადგმული ობიექტების იერარქია, რომელიც ქმნის ობიექტის სტრუქტურას, არის ობიექტური მოდელი (object model).

მაგალითად, ობიექტი „ავტომობილი" შედგება რამდენიმე „შვილი" ობიექტისგან: ძრავი, ბორბლები, ძარა და ა.შ. „ავტომობილი" ობიექტის ყოფაქცევა, რომელსაც „შვილი" ობიექტისთვის (ძრავი) თვისებაში აქვს ცილინდრების რაოდენობა 4 და 8, იქნება განსხვავებული.

„შვილი" ობიექტი შეიძლება შედგებოდეს თავის საკუთარი „შვილი" ობიექტებისგან და ა.შ. მაგალითად: **ავტომობილი -> ძრავი -> სანთლები.**

- **ინკაფსულაცია:**

ობიექტ-ორიენტირებული დაპროგრამების (ოოდ) ერთ-ერთი საბაზო პრინციპია ინკაფსულაცია, რომლის არსი მდგომარეობს ობიექტის რეალიზაციის გამოყოფაში მისი ინტერფეისისაგან. ანუ პროგრამული დანართი (აპლიკაცია) ურთიერთქმედებს ობიექტთან მისი ინტერფეისის საშუალებით, რომელიც შედგება ღია თვისებებისა და მეთოდებისაგან.

ობიექტები ერთმანეთთან ურთიერთმოქმედებს თავიანთი ღია თვისებებით და მეთოდებით, ამიტომაც ობიექტს უნდა ჰქონდეს ყველა აუცილებელი მონაცემი და მეთოდების ერთობლიობა, ამ მონაცემთა დასამუშავებლად.

ინტერფეისმა არ უნდა მისცეს „სხვას" ობიექტის შიგა მონაცემებთან წვდომის უფლება, რაც გამორიცხავს ამ შემთხვევაში ობიექტის შიგა მონაცემებისთვის public-მოდიფიკატორის გამოყენებას (გამოიყენება private).

- **პოლიმორფიზმი, ინტერფეისი და მემკვიდრეობითობა**

პოლიმორფიზმი (polymorphism) მრავალფორმიანობას ნიშნავს. მას ობიექტ-ორიენტირებულ დაპროგრამებაში განსაკუთრებული როლი აქვს. პოლიმორფიზმის საშუალებით შესაძლებელია ერთიდაიმავე გახსნილი ინტერფეისის სხვადასხვაგვარი რეალიზაცია სხვადასხვა კლასში, ანუ პოლიმორფიზმის საშუალებით შესაძლებელია ობიექტის მეთოდებისა და თვისებების გამოძახება მათი რეალიზაციის მიუხედავად.

მაგალითად, ობიექტი *მძღოლი* ურთიერთქმედებს ობიექტთან *ავტომობილი* იმავე სახელის მქონე ღია ინტერფეისის საშუალებით. თუ სხვა ობიექტი, მაგალითად,

სატვირთო ან *სპორტული-მანქანა* ფლობს ამ ღია ინტერფეისის მხარდაჭერას, მაშინ ობიექტს *მძღოლი* შეუძლია მათთანაც ურთიერთქმედება, მიუხედავად ინტერფეისის განსხვავებული რეალიზაციისა.

პოლიმორფიზმის რეალიზაციის ორი ძირითადი მიდგომა არსებობს: ინტერფეისებისა და მემკვიდრეობითობის გამოყენებით. განვიხილოთ თითოეული.

- **ინტერფეისი** (interface) – ესაა შეთანხმება, რომელიც ობიექტის ყოფაქცევას განსაზღვრავს. იგი ადგენს კლასის წევრთა სიას, მაგრამ არაფერს ამბობს მათი რეალიზაციის შესახებ. ობიექტში დასაშვებია რამდენიმე ინტერფეისის რეალიზაცია, ხოლო ერთიდაიგივე ინტერფეისი შეიძლება რეალიზებულ იქნას სხვადასხვა კლასში.

ნებისმიერ ობიექტებს, რომლებშიც რეალიზებულია რომელიმე ინტერფეისი, შეუძლია ერთმანეთთან ურთიერთმოქმედება მისი საშუალებით. მაგალითად, ობიექტში *ავტომობილი,* რომელზეც ჩვენ ვსაუბრობთ, შეიძლება *IDrivable*-ინტერფეისის რეალიზაცია (ინტერფეისთა სახელები მიღებულია დაიწყოს "I" ასოთი), მეთოდებით: *მოძრაობა-წინ, მოძრაობა-უკან, გაჩერება.* იგივე ინტერფეისი შეიძლება რეალიზდეს სხვა კლასებშიც, როგორიცაა, მაგალითად, *სატვირთო-მანქანა* ან *კატერი.* შედეგად, ამ ობიექტებს შეუძლია ურთიერთმოქმედება ობიექტთან *მძღოლი.* ეს უკანასკნელი მთლიანად უხილავია ინტერფეისის რეალიზაციისათვის, რომელთანაც იგი მოქმედებს, მისთვის ცნობილია მხოლოდ ინტერფეისი.

- **მემკვიდრეობითობა** (inheritance) იძლევა ახალი კლასების შექმნის საშუალებას არსებულის ბაზაზე. ამასთანავე ახალ კლასებში ნებადართულია ძველი კლასების სრული ფუნქციონალობის ჩართვა და, საჭიროებისამებრ, შესაძლებელია მათი წევრების მოდიფიცირებაც.

კლასი, რომელიც გამოცხადებულია სხვა კლასის საფუძველზე, უწოდებენ წარმოებულ კლასს (derived class). ყოველ კლასს შეიძლება ჰქონდეს მხოლოდ ერთი პირდაპირი წინაპარი – მისი საბაზო კლასი (base class). წარმოებულ კლასს აქვს საბაზო კლასის წევრების ერთობლიობა. ასევე შესაძლებელია ახალი წევრების დამატება და ძველი წევრების (მემკვიდრეობით მიღებული) რეალიზაციის ცვლილებაც. წარმოებული კლასები ინარჩუნებს თავისი საბაზო კლასის ყველა მახასიათებელს და უნარი აქვს სხვა ობიექტებთან ისეთი ურთიერთმოქმედებისათვის, როგორც საბაზო კლასის ეგზემპლარებს.

მაგალითად, საბაზო კლასიდან *ავტომობილი* შეიძლება წსრმოებული კლასის *სპორტული-ავტომობილი* გამოცხადება, რომელიც თავის მხრივ, როგორც საბაზო კლასი, შეძლებს ახალი წარმოებული კლასის *სპორტული-კაბრიოლეტი* გამოცხადებას. ყოველ წარმოებულ კლასში შესაძლებელია ახალი წევრების (თვისებები, მეთოდები, მოვლენები) შემოტანა, ამსთანავე ისინი ინარჩუნებს საწყისი საბაზო კლასის *ავტომობილი* ფუნქციურობას უცვლელი ფორმით.

IV თავი
Visual C#.NET ენა - აპლიკაციების აგების ვიზუალური ინსტრუმენტული საშუალება

4.1. შესავალი: Visual C# 2015-ის სამუშაო გარემო და ენის ელემენტები

Visual C# 2015 არის ვიზუალური, ობიექტორიენტირებული ენა, რომელიც მაიკროსოფთის ფირმამ .NET Framework 4.5 ინტეგრირებულ გარემოსთან ერთად, ბოლო ვერსიაში წარმოადგინა, Visual Basic.NET, Visual C++.NET და F#.NET ენებთან ერთად.

Visual C# 2015 (შემდგომში C#) ენა მთლიანად მოიცავს მის წინამორბედს - Visual C# 2010-13 და ახალ გაფართოებებს.

შესავალში აღწერილია ვიზუალური C# ენის საწყისები და პირველი პროგრამული კოდის აგების ელემენტები ვინდოუსის აპლიკაციისათვის.

პრაქტიკული ექსპერიმენტებისათვის შესაძლებელია Visual C# 2015 Express Edition პაკეტის გამოყენება, რომელიც უფასოა და ინტერნეტიდან მისაწვდომი:

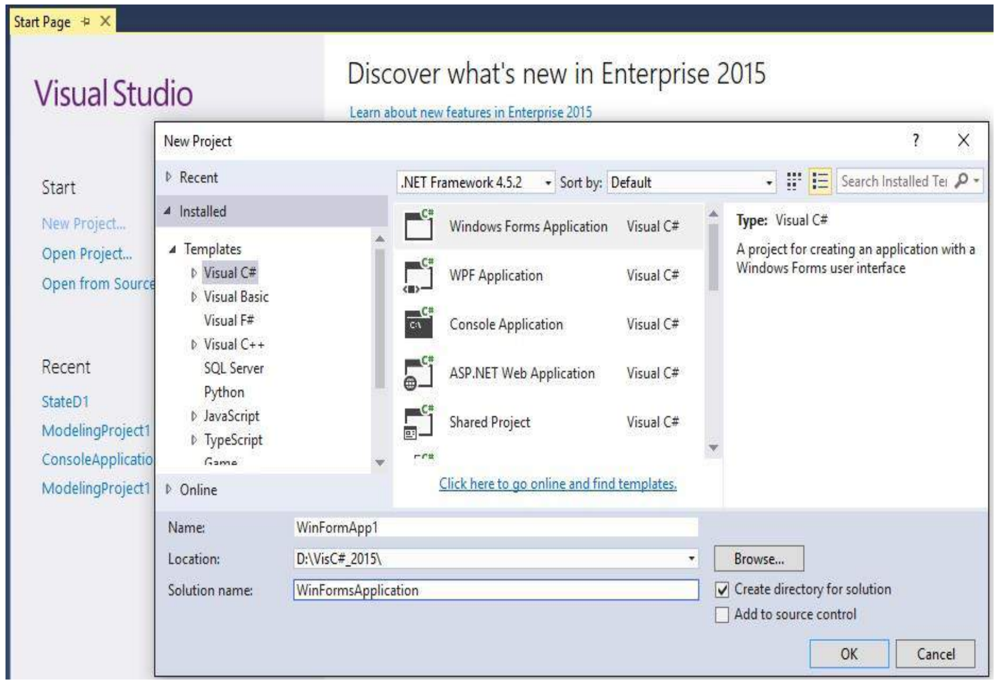
http://www.microsoft.com/visualstudio/en-us/products/2015-editions/ visual-csharp-express

იგი მოიცავს ტექსტურ რედაქტორს პროგრამული კოდის ასაგებად, კომპილატორს მის გასამართად და დებაგერს - შეცდომების აღმოსაჩენად.

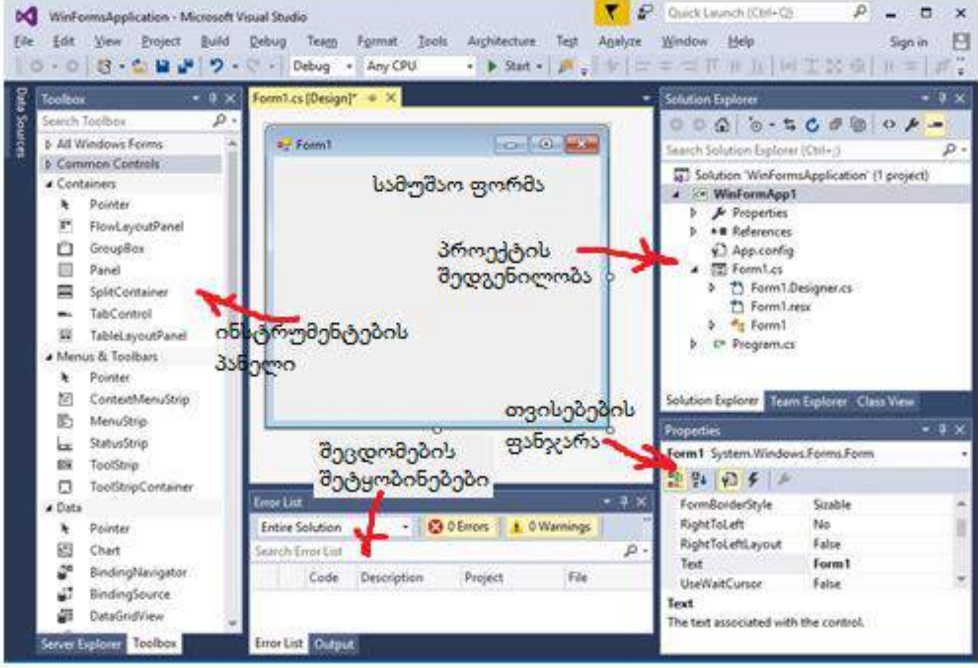
ნახ.4.1

სასტარო ინტერფეისიდან აირჩევა New Project (ახალი პროგრამის შექმნა), განისაზღვრება პროექტის დასახელება და შენახვის კატალოგი (ნახ.4.2).

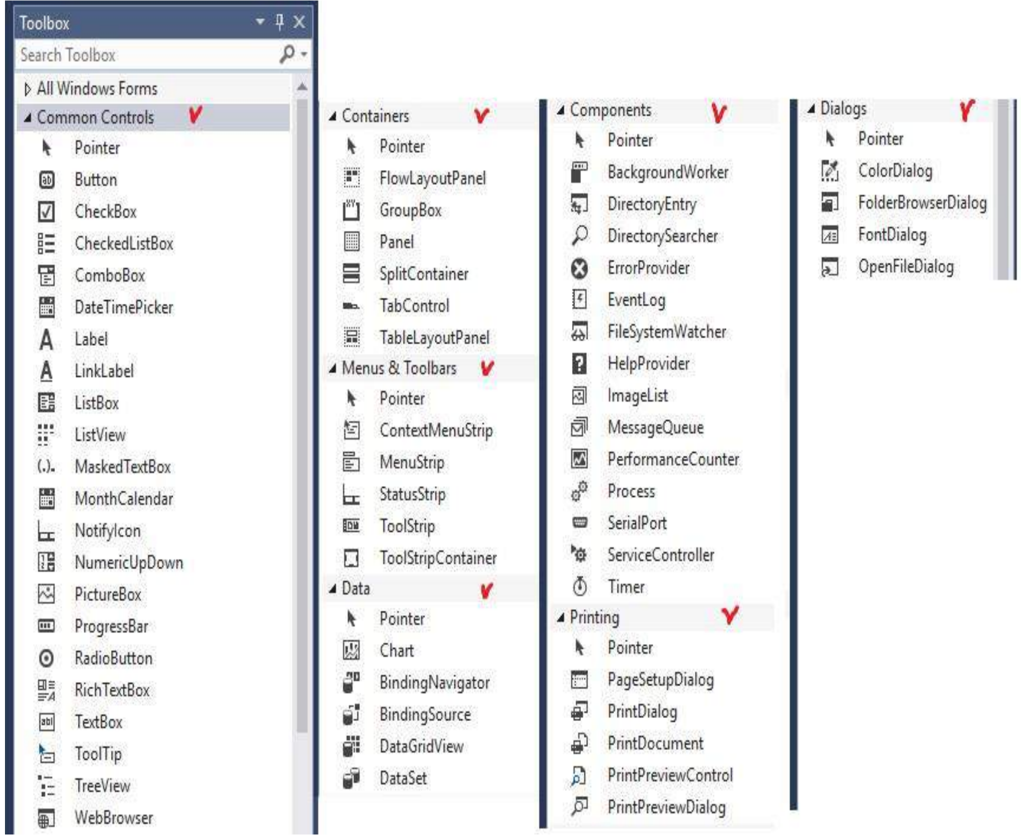
შედეგეად მიიღება მომხმარებლის სამუშაო ფორმა-Form1 (ნახ.4.3) და ToolBox-ინსტრუმენტების პანელი (ნახ.4.4), საიდანაც ხდება პროგრამისთვის საჭირო მართვის ელემენტის გადატანა ფორმაზე. 4.3 ნახაზის ქვედა მარჯვენა ნაწილში მოთავსებულია ფორმის ელემენტების თვისებათა (Properties) ფანჯარა. იგი ცალსახად აღწერს ფორმის ან მისი მონიშნილი ელემენტ(ებ)ის თვისებებს.



ნახ.4.2. ახალი პროექტის შექმნა

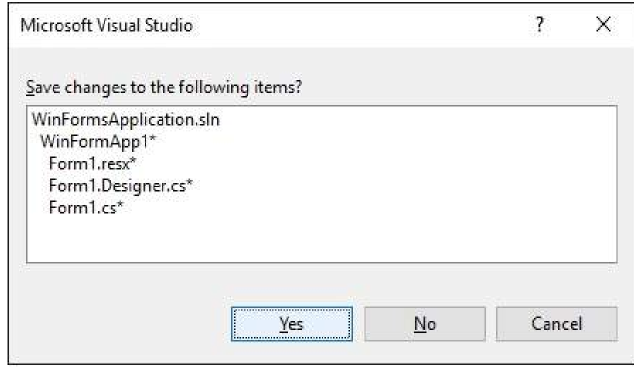


ნახ.4.3. სამუშაო გარემო



ნახ.4.4. Toolbox-ის ელემენტები

4.5 ნახაზზე მოცემულია აპლიკაციის ფორმების და კოდების შენახვის პროცედურა.



ნახ.4.5. Yes - დასტური კოდის შესანახად

4.2. აპლიკაციის აგება დაპროგრამების რამდენიმე ენის საფუძველზე .dll ფაილების შექმნით

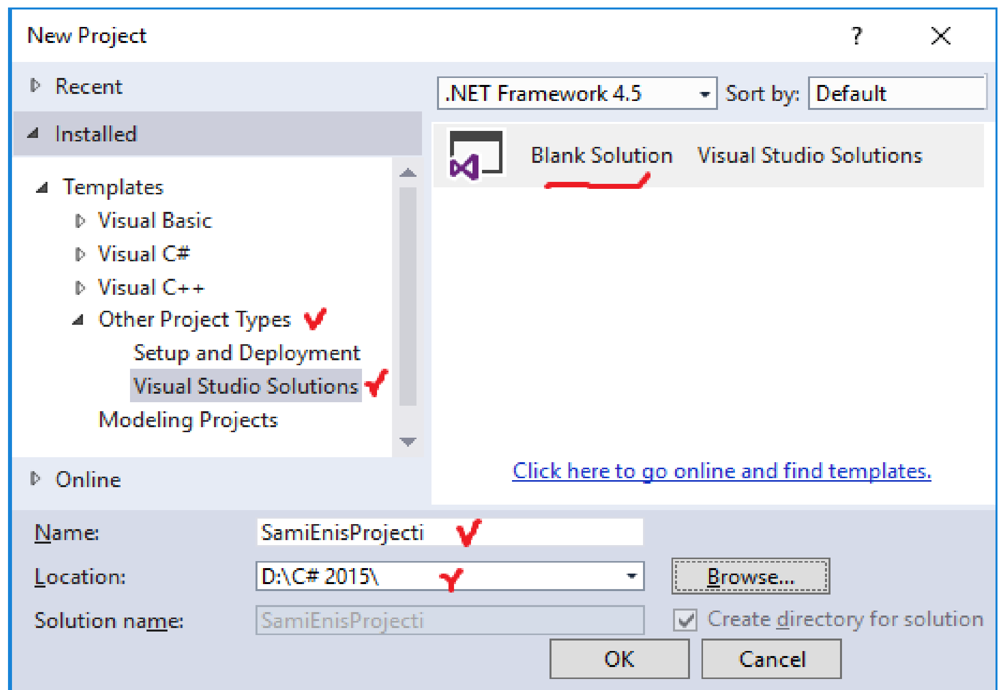
ილუსტრირებულია .NET ტექნოლოგიის ძირითადი კონცეფცია და პრინციპები, რომ პროგრამული აპლიკაციის დამუშავება შესაძლებელია გუნდში სხვადასხვა პროგრამული ენის მცოდნე დეველოპერის მიერ, კერძოდ, C++, Visual Basic და C# ენების საფუძველზე. პროექტის აგებისას გათვალისწინებულ უნდა იქნას საერთო მოთხოვნები ღია ცვლადებზე, მეთოდებსა და თვისებებზე.

ამოცანის გადაწყვეტის გეგმა:

- შეიქმნას C++ -ის საბაზო კლასი;
- შეიქმნას Visual Basic.NET კლასი, რომელიც იქნება C++ - კლასის მემკვიდრე;
- შეიქმნას C# კლასი, რომელიც კონსოლის აპლიკაციაში შექმნის Visual Basic.NET კლასის ეგზემპლარს და გამოიძახებს მის მეთოდს.

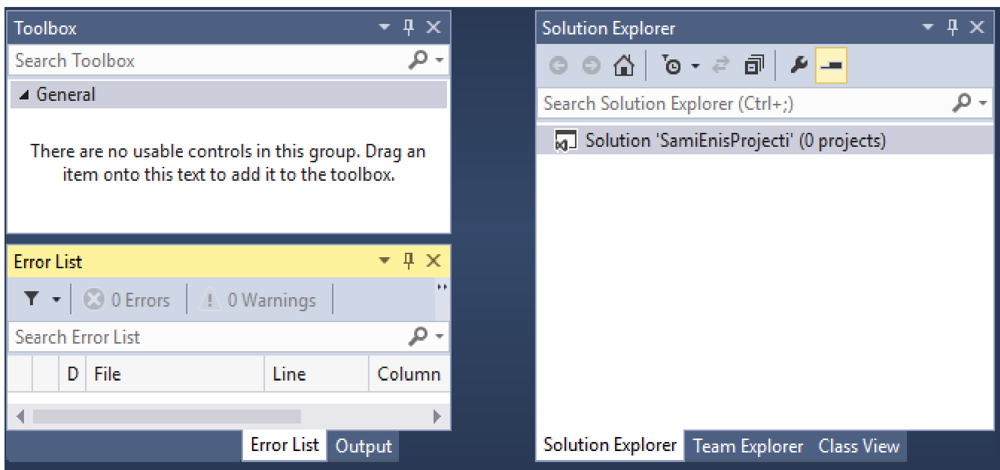
განვახორციელოთ პროექტის პროგრამული რეალიზაცია Visual Studio .NET Framework 4.5 სამუშაო გარემოში.

- შევარჩიოთ ახალი პროექტის განთავსების ადგილი;
- მენიუდან გამოვიძახოთ ახალი ცარიელი პროექტის შექმნის პროგრამა (ნახ.4.6): New->Project და Visual Studio Solution-> Blank Solution.



ნახ.4.6. ცარიელი პროექტის შექმნა

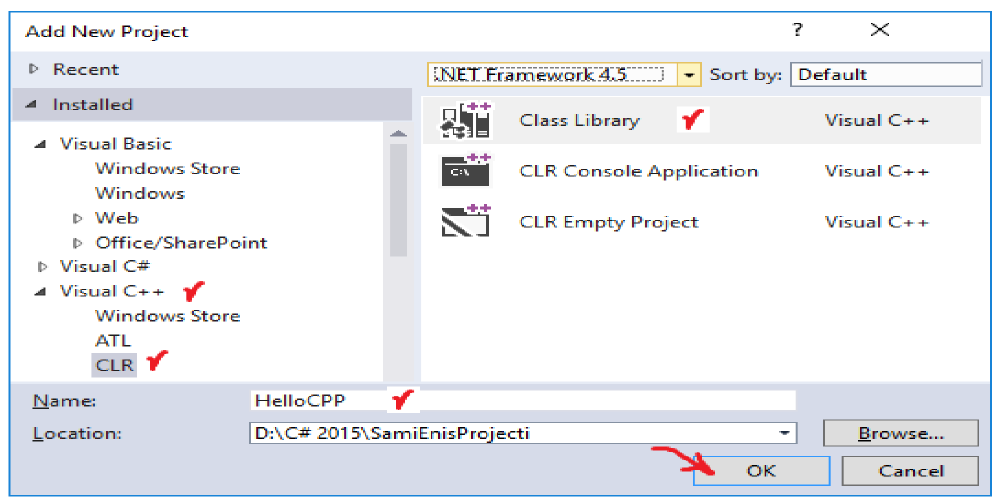
პროექტის სახელი (Name: SamiEnisProjecti) და მისი შენახვის ადგილი (Location) ჩვენი სურვილით მივუთითოთ. OK-ის შემდეგ Blank Solution შაბლონის დახმარებით შეიქმნება აპლიკაცია, რომელიც *ნეიტრალური* იქნება დაპროგრამების ენებისადმი. შედეგად მივიღებთ 4.7 ნახაზზე ნაჩვენებ ფანჯარას.



ნახ.4.7. შეიქმნა ენებისადმი ნეიტრალური პროექტი

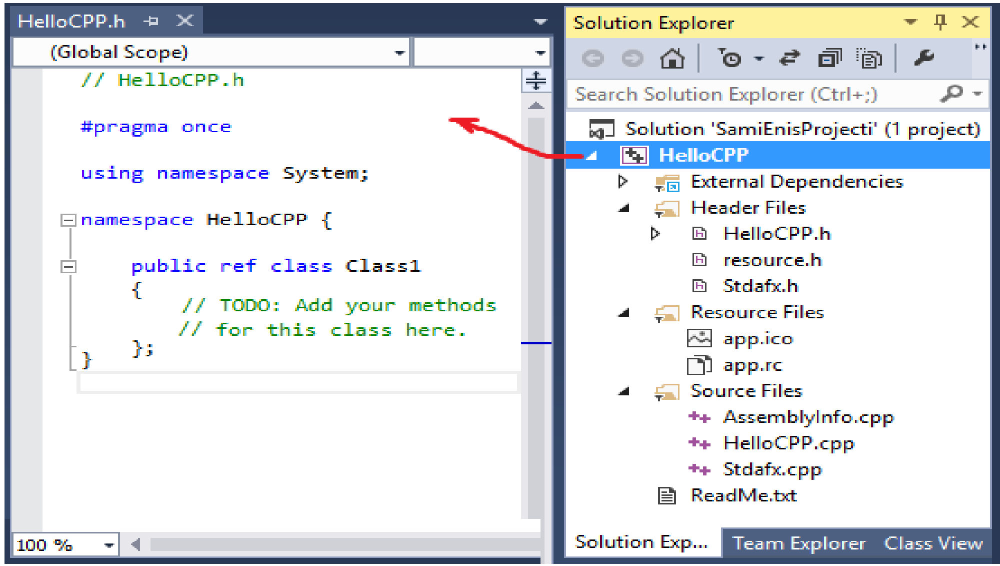
- **კლასის შექმნა ახალ პროექტში C++.NET ენაზე**

ცარიელ პროექტს (გადაწყვეტა - Solution „SamiEnisProjecti“) დავამატოთ ახალი (ქვე)პროექტი მასზე მარჯვენა ღილაკის დაწკაპუნებით და შემდეგ Add -> New Project არჩევით. 4.8 ნახაზზე ნაჩვენებია, თუ როგორ მიეთითება VisualC++ ენა, Class Library და ქვეპროექტის სახელი Name: HelloCPP, შემდეგ OK.



ნახ.4.8. C++ ქვეპროექტის დამატება HelloCPP სახელით

Solution Explorer-ში გამოჩნდება შედეგი (ნახ.4.9). გავხსნათ Hello.CPP.h ფაილი რედაქტორებისათვის. ახლადშექმნილი კლასის Class1 მაგივრად ჩავწეროთ სახელი HelloCPP. HelloCPP.h კოდი მოცემულია 4.1-ელ ლისტინგში.



ნახ.4.9. შედეგი Solution Explorer-ში

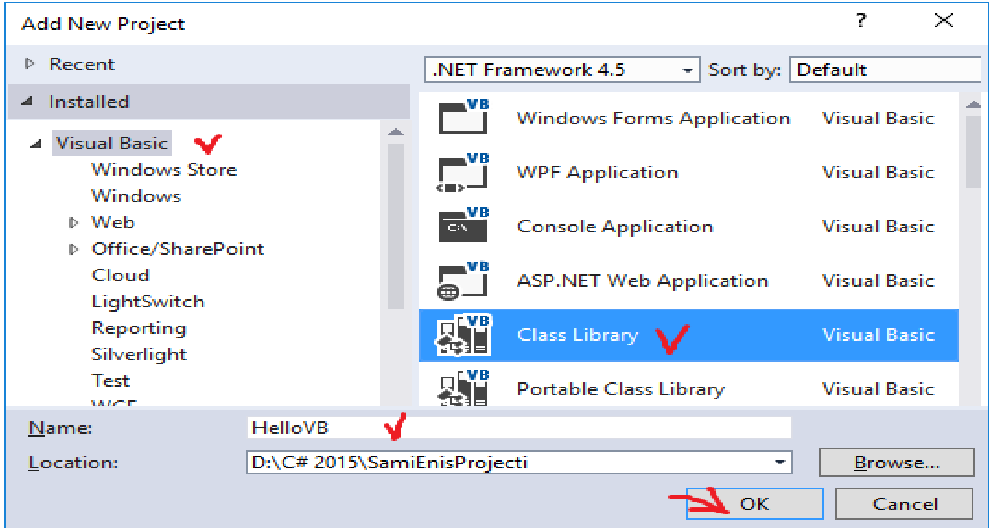
```
// ---- ლისტინგი_4.1 -- შეცვლილი HelloCPP.h ----
#pragma once
using namespace System;
namespace HelloCPP
{
  public ref class HelloCPP
   {
     // --- აქ ჩაემატება კლასის მეთოდი ----
     public:
     virtual void Hello()
      {
        Console::WriteLine("Mogesalmebit C++ enis garemodan da !\n
                           viZaxeb Visual Basics !\n\n“);
      }
   };
}
```

HelloCPP::Hello() მეთოდი იყენებს .NET Framework კლასების ბიბლიოთეკიდან System::Console::WriteLine() ფუნქციას, რათა კონსოლზე გამოიტანოს მისალმება C++ კოდიდან.

პროექტი გავუშვათ სინტაქსური შეცდომების გასასწორებლად, თუ ასეთი იქნება. იგი ჯერ არ უნდა ავამუშავოთ შესრულებაზე.

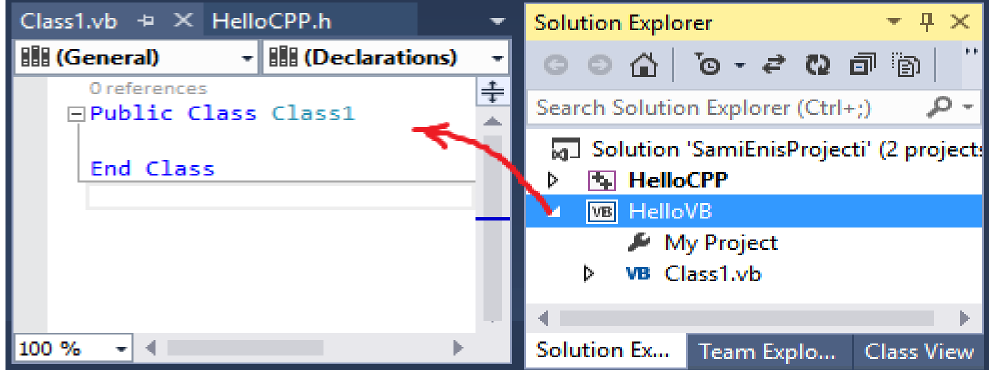
- **კლასის შექმნა ახალ პროექტში Visual Basic .NET ენაზე**

დავამატოთ Solution-ში ახალი პროექტი და მივუთითოთ, რომ იგი შეიქმნას Visual Basic .NET -ში სახელით Name: HelloVB (ნახ.4.10).



ნახ.4.10. VisualBasic ქვეპროექტის დამატება HelloVB სახელით

მივიღებთ ასეთ შედეგს (ნახ.4.11). ახალი პროექტი HelloVB დაემატება Solution Explorer პანელზე თავისი შემადგენელი ფაილებით.



ნახ.4.11. შედეგი Solution Explorer-ში

ახლა უკვე გვაქვს HelloCPP და HelloVB ორი პროექტი.

მეორეში ავირჩიოთ Class1.vb ფაილი და შევუცვალოთ სახელი შინაარსის გათვალისწინებით, მაგალითად, HelloVB.vb, შემდეგ გავხსნათ იგი და ჩავწეროთ ჩვენთვის საჭირო ტექსტი (ლისტინგი_4.2).

```
' --ლისტინგი_4.2 ---  HelloVB.vb ------------
Public Class HelloVB
  Inherits HelloCPP.HelloCPP
  Public Overrides Sub Hello()
     MyBase.Hello()
     System.Console.WriteLine("Mec mogesalmebiT Visual Basic.NET-idan !!")
   End Sub
End Class
```

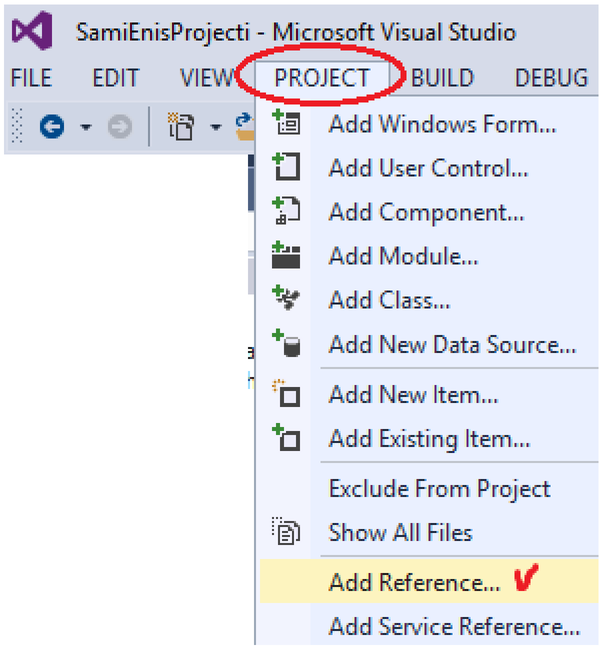
ეს კოდი განსაზღვრავს HelloVB კლასს, რომელიც მემკვიდრეობით იქნა მიღებული C++ ის მმართველი კლასიდან HelloCPP. ამგვარად, HelloVB კლასი ჩაანაცვლებს (Overrides) მშობელი HelloCPP კლასის ვირტუალურ Hello() მეთოდს, ოღონდ ჯერ გამოიძახებს Hello() მეთოდის ვერსიას მშობელი კლასიდან HelloCPP, შემდეკ კი გამოყავს ეკრანზე თავისი მისალმება.

საყურაღებოა, რომ კოდის რეადქტორში საბაზო კლასის სახელი (HelloCPP.HelloCPP) და ჩასანაცვლებელი მეთოდის სახელი Hello() ტალღისებური ხაზით. ეს ნიშნავს, რომ კოდის რედაქტორი ამ სახელებს ვერ ხედავს და წინასწარ იძლევა სინტაქსურ შეცდომას. თუ კურსორს დავაყენებთ ამ ფრაგმეტზე, იგი მოგვცემს შეცდომის მნიშვნელობას (ნახ.4.12).

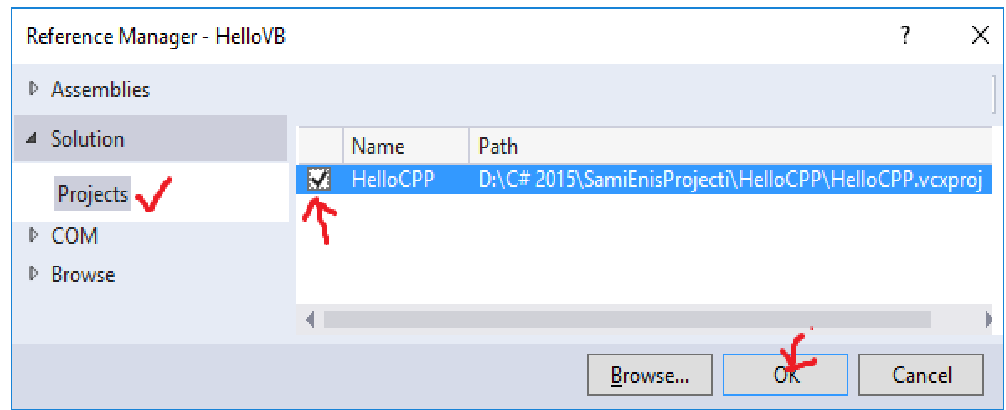
```
HelloVB.vb*   HelloCPP.h
(General)   (Declarations)
0 references
Public Class HelloVB
    Inherits HelloCPP.HelloCPP
    0 references
    Public Ov   Type 'HelloCPP.HelloCPP' is not defined.
        MyBase.Hello()
        System.Console.WriteLine("Mec mogesalmebiT Visual Basic.NET-idan !!")
    End Sub
End Class
```

ნახ.4.12. შეცდომის იდენტიფიკაცია

აუცილებელია Visual Basic-ის კომპილატორს მიეთითოს, თუ სად იმყოფება ეს ნაკრები (assembly), რომელიც განსაზღვრავს მოცემულ ტიპს. ამისათვის მთავარი მენიუს Project-> Add Reference->Projects-დან (ნახ.4.13) ავირჩევთ HelloCPP-ს და OK. აუცილებელია Visual Basic-ის კომპილატორს მიეთითოს თუ სად იმყოფება ეს ნაკრები (assembly), რომელიც განსაზღვრავს მოცემულ ტიპს. ამისათვის მთავარი მენიუს Project-> Add Reference->Projects-დან (ნახ.4.13) ავირჩევთ HelloCPP-ს და OK (ნახ.4.14).

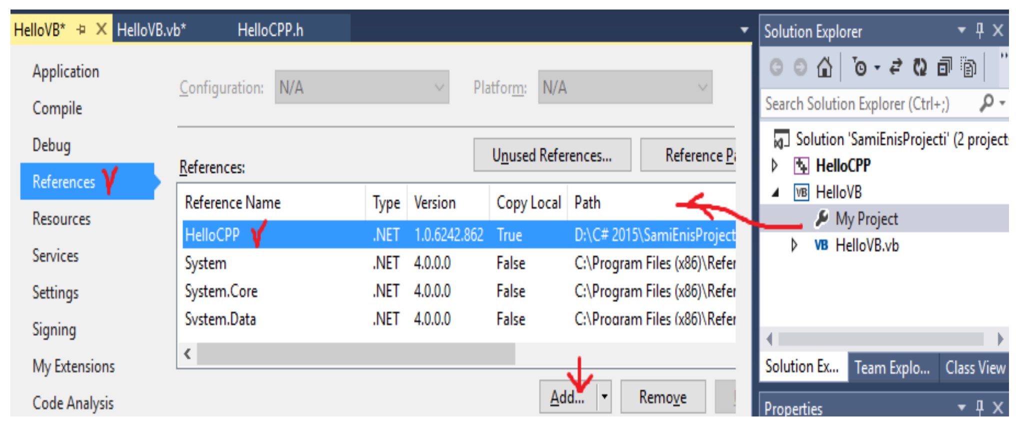


ნახ.4.13, Project-> Add Reference არჩევა



ნახ.4.14. HelloCPP-ის მონიშვნა

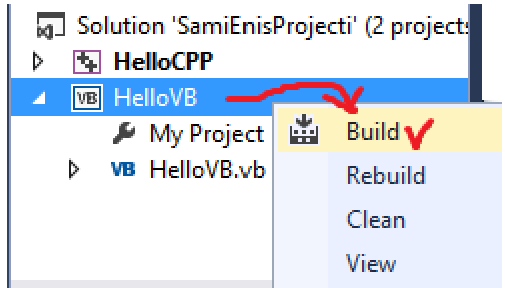
Solution-ში HelloVB-ს MyProject-ის ამოქმედებით გაიხსნება 4.15 ნახაზის მსგავსი ფანჯარა, რომელშიც References-ზე გადართვით გაჩნდება შიგთავსი. ვირჩევთ HelloCPP და Add-ით ვადასტურებთ.



ნახ.4.15. References-ზე გადართვა და Add

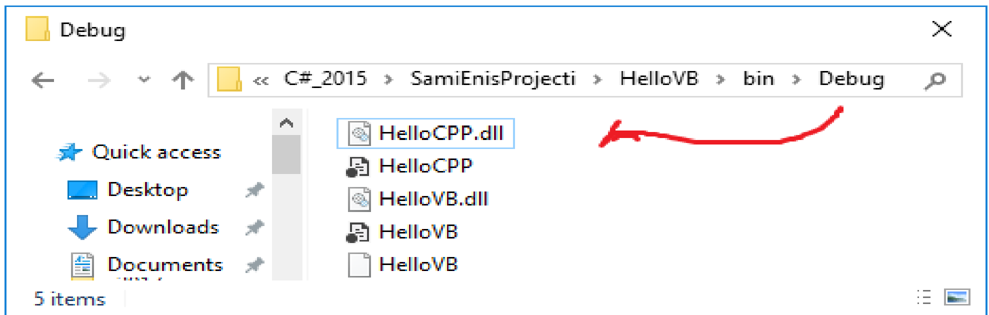
ამის შემდეგ გამოჩნდება Add Reference და OK ღილაკით ვასრულებთ CPP-კლასთან დაკავშირების პროცესს.

ბოლოს საჭიროა HelloVB პროექტზე მარჯვენა ღილაკით გამოვიტანოთ 4.16 კონტექსტური მენიუ და ავირჩიოთ Build (ეს პროექტი ტრანსლატორით ამუშავდება და სინტაქსური გამართვის შედეგს მოგვცემს).



ნახ.4.16

როგორც წესი, თუ შეცდომები არაა HelloVB პროექტში, მაშინ მის შესაბამის ფოლდერის bin->Debug -ში უნდა გამოჩნდეს დაკავშირებული Hello.CPP.dll კლასის ფაილი (ნახ.4.17).



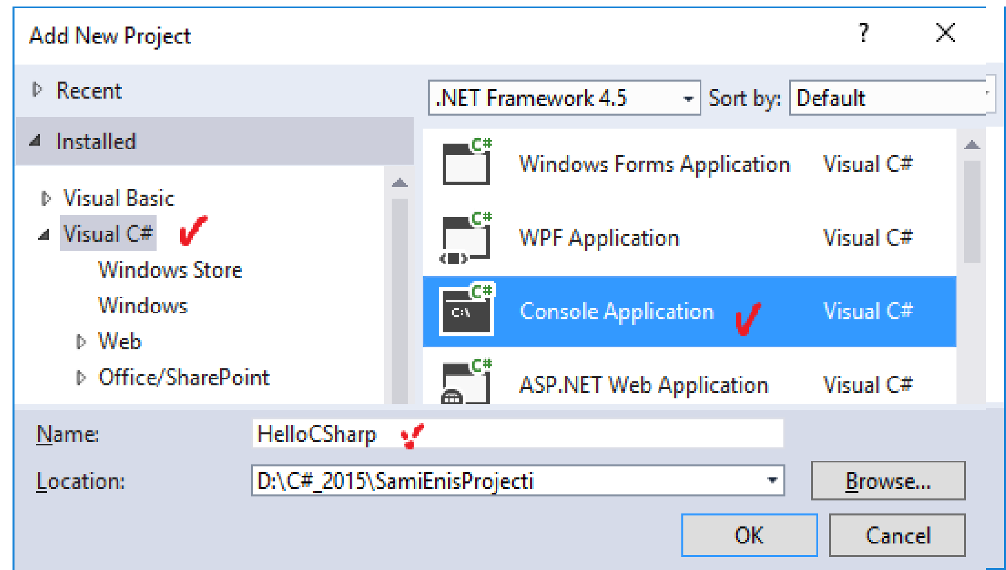
ნახ.4.17

ამგვარად, Hello.CPP პროექტი მიუერთდება გადაწყვეტას (Solution-ში) და გამოჩნდება მომავალში HelloVB-ში, როგორც ზემოთ იქნა ნაჩვენები. ამასთანავე, 4.13 ნახაზზე ნაჩვენები „ქვეშგახაზული" შეცდომების აღნიშვნები HelloVB.vb პროგრამის ტექსტიდან გაქრა ! პროგრამის საბოლოო ტექსტი მოცემულია ლისტინგში:

```
' ----ლისტინგი_4.3 --  HelloVB.vb ---------------
Public Class HelloVB
    Inherits HelloCPP.HelloCPP
    Public Overrides Sub Hello()
    MyBase.Hello()
    System.Console.WriteLine("Mec mogesalmebiT Visual Basic.NET-idan!")
    End Sub
End Class
```

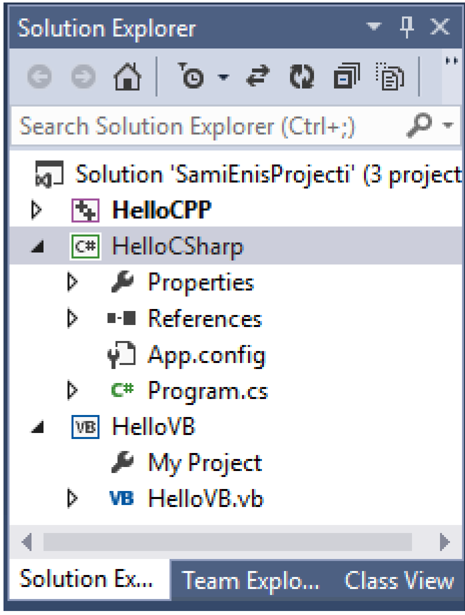
- სასტარტო პროექტის დამატება Solution-ში **C#.NET** ენაზე

დასკვნით ფაზაში ჩვენი გადაწყვეტისათვის (Solution-ში) SamiEnisProjecti უნდა შევქმნათ მესამე, მთავარი სასტარტო პროექტი, რომელიც აგებული იქნება C#.NET ენაზე Console Application შაბლონის სახით. დავარქვათ მას HelloCSharp. ამგვარად, ვამატებთ პროექტს (ნახ.4.18) `HelloCSharp` სახელით.



ნახ.4.18

OK ღილაკის ამოქმედების შემდეგ მივიღებთ Solution-ში ახალ, HelloCSharp პროექტს თავისი შემადგენელი კომპონენტებით (ნახ.4.19).



ნახ.4.19

გადავარქვათ სახელი Program.cs ფაილს ჩვენი პროექტის შინაარსის შესაბამისად: HelloCSharp.cs. გამოვიტანოთ ეს ფაილი რედაქტირების ფანჯარაში და ჩავამატოთ Main() -ში შესაბამისი სტრიქონები. ასევე შეიძლება Main() -ის არგუმენტების წაშლა, აქ არ გვჭირდება. რედაქტირებული ტექსტი მოცემულია 4.20. ნახაზზე.

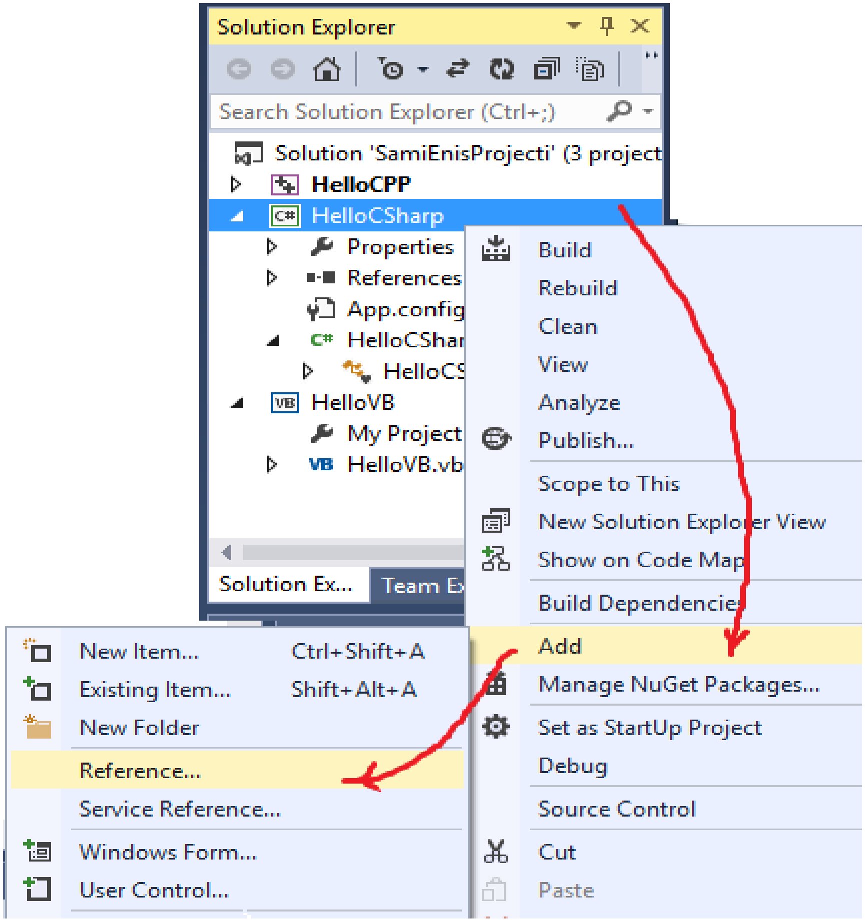
```
HelloCSharp.cs    HelloVB    HelloVB.vb    HelloCPP.h
HelloCSharp.HelloCSharp                      Main()
using System;
namespace HelloCSharp
{
    0 references
    class HelloCSharp
    {
        0 references
        static void Main()
        {
            HelloVB.HelloVB hello = new HelloVB.HelloVB();
            hello.Hello();
            Console.WriteLine("Salami Visual C#.NetPipeStyleUriParser-idan !" );
            Console.ReadLine();
        }
    }
}
```

ნახ.4.20

პროგრამაში Main() სტატიკური მეთოდი არის აპლიკაციაში შესვლის წერტილი. ეს მეთოდი ქმნის HelloVB კლასის ახალ ობიექტს hello და მისთვის იძახებს Hello() მეთოდს, რომელშიც ინახება ორი შეტყობინება. ერთი CPP.NET-იდან, მეორე VB.BET-დან,

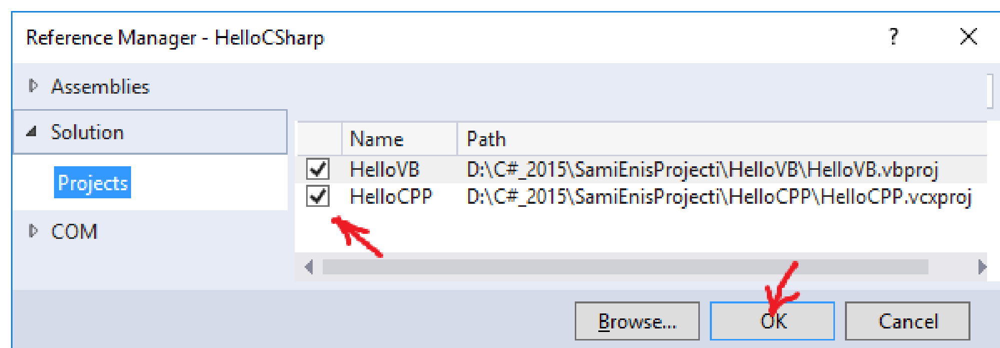
რომლებიც ადრე მოვამზადეთ. მესამე შეტყობინებას თვით C#.NET გამოიტანს, დამშვიდობებასთან ერთად.

ახლა საჭიროა HelloCSharp პროექტში ჩავამატოთ მიმთითებლები (კავშირები) HelloCPP და HelloVB პროექტებზე. მაშინ 4.20 ნახაზზე ნაჩვენები შეცდომები (ქვეშხაზგასმული HelloVB) გასწორდება. ამისათვის ვირჩევთ Add Reference…-ს (ნახ.4.21).

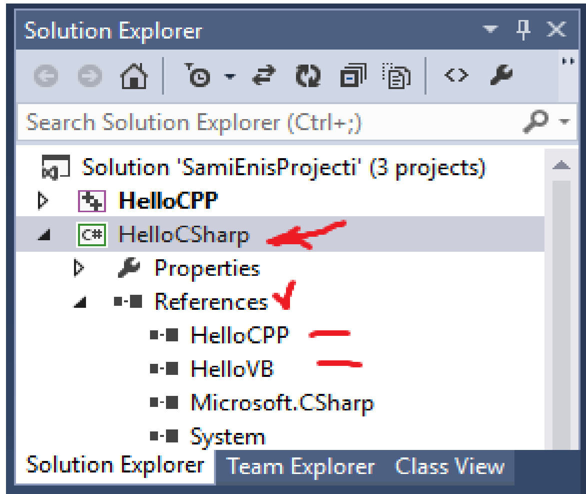


ნახ.4.21

Add Reference ფანჯარაში გადავრთოთ Project-ზე, მოვნიშნოთ ორივე, HelloCPP და HelloVB პროექტები და ღილაკი OK (ნახ.4.22). შედეგად გაქრება შეცდომები, ანუ 4.20 ნახაზზე „ხაზები“, რაც მიუთითებს იმაზე, რომ კავშირი პროექტებს შორის კარგად შესრულდა. ეს შედეგი აისახება Solution-ფანჯარაში HelloCSharp პროექტის References –ში (ნახ.4.23).

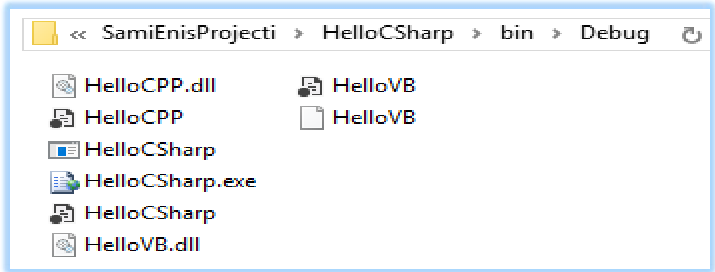


ნახ.4.22



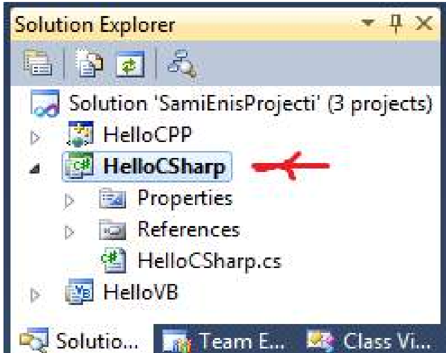
ნახ.4.23

ავამუშავოთ HelloCSharp პროექტი მარჯვენა ღილაკით და build-ით, რათა სინტაქსურად დამუშავდეს იგი. შეცდომების არარსებობის შემთხვევაში მოხდება HelloCPP.dll და HelloVB.dll ფაილების განთავსება HelloCSharp პროექტის bin->Debug ფოლდერში (ნახ.4.24). აქვეა HelloCSharp.exe აპლიკაციის ფაილიც.

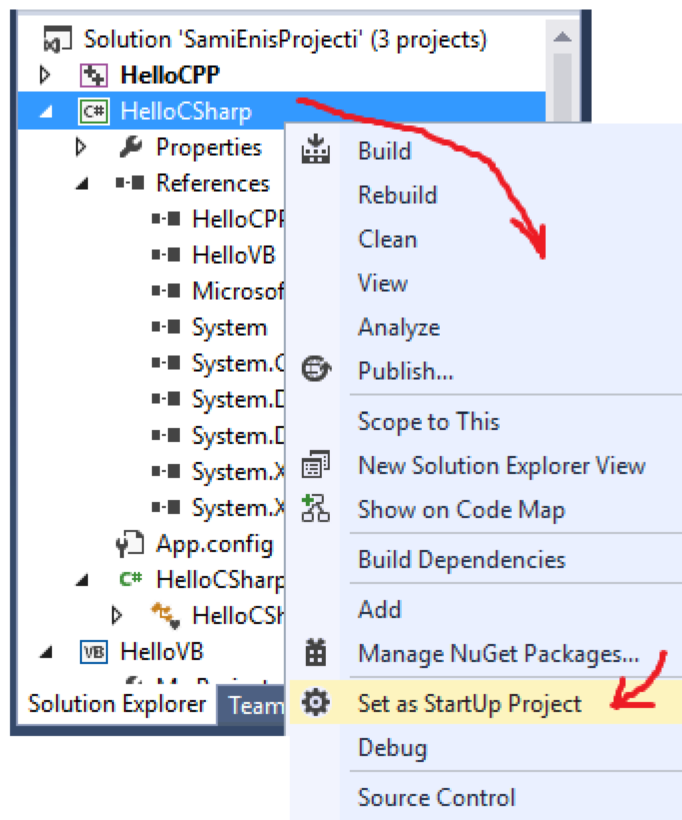


ნახ.4.24

Solution-ში HelloCSharp პროექტი გადავაკეთოთ სასტარტო ფაილად (ნახ.4.25-ა,ბ).



ნახ.1.25-ა



ნახ.4.25-ბ

ბოლოს, ავამუშავოთ აპლიკაცია და მივიღებთ შედეგს (ნახ.4.26).

```
file:///D:/C#_2015/SamiEnisProjecti/HelloCSharp/bin/Debug/HelloCSharp.EXE
Mogesalmebit C++ enis garemodan da !
viZaxeb Visual Basics !

Mec mogesalmebiT Visual Basic.NET-idan !!
Salami Visual C#.NetPipeStyleUriParser-idan !
```

ნახ.4.26. პროგრამის მუშაობის შედეგი

რეზიუმე: აპლიკაციის სხვადასხვა პროექტი დამუშავდა დაპროგარმების სხვადასხვა ენაზე, რომლებიც გამოიყენება .NET გარემოში. მრავალჯერადი გამოყენების ფაილები შემუშავდა კლასების დახმარებით და რეალიზებულია დინამიკური .dll - ფაილების სახით.

4.3. კლასებისა და ობიექტების დაპროგრამება მართვის ამოცანებში

განვიხილოთ ობიექტორიენტირებული დაპროგრამების ისეთი ძირითადი საკითხები, როგორიცაა კლასებისა და ობიექტების უნიფიცირებული მოდელირება და დეველოპმენტი კონკრეტული მართვის ამოცანის მაგალითზე, მათი შემდგომი განზოგადების მიზნით.

ამოცანა_4.1: ავაგოთ ობიექტორიენტირებული პროგრამის კოდი (კლასების და მეთოდების გამოყენებით) მაღლივ შენობაში ლიფტის გადაადგილების პროცესის მოდელირებისათვის [80].

მაგალითად, შენობა N სართულიანია. მას აქვს ლიფტი. პერსონა, რომელიც შედის ლიფტში (ხდება მგზავრი), ირჩევს მისთვის საჭირო სართულს და მიემგზავრება (ზევით ან ქვევით). საჭიროა ამ პროცესის მოდელირება და დაპროგრამება ისე, რომ დაფიქსირდეს ლიფტის საწყისი მდგომარეობა, ამუშავების შემდეგ მისი საბოლოო მდგომარეობა, გავლილი სართულების რაოდენობა (ერთი ამუშავებისას). საბოლოოდ გამოიცეს რეპორტი, თუ სულ რამდენი სართული გაიარა ლიფტმა ერთი სეანსის (გარკვეული პერიოდის) განმავლობაში.

1) კლასთა მოდელი და მათი აგების პირობები:

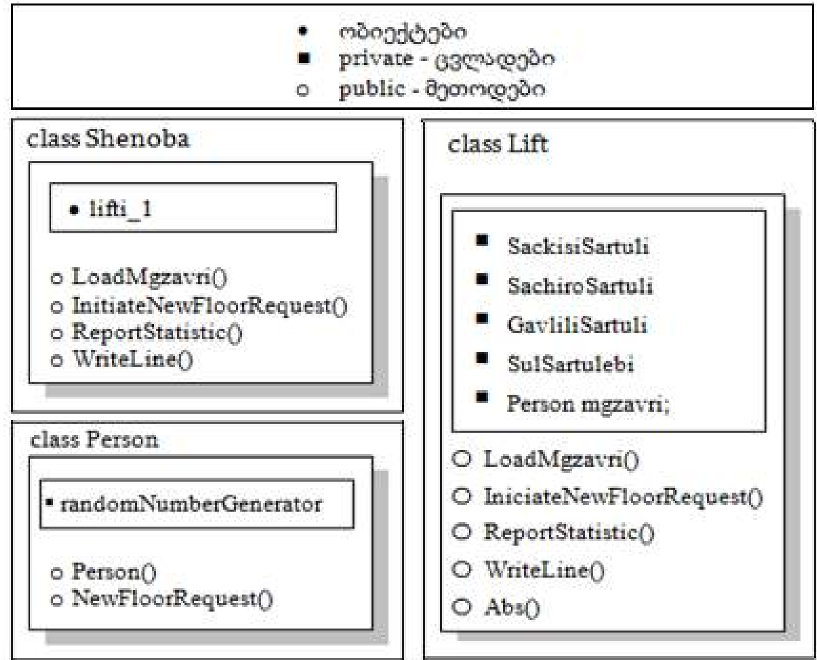
ინკაფსულაციის სქემა მოცემულია 4.27 ნახაზზე.

- პროგრამაში სამი კლასია: Shenoba, Lift და Person;

- Shenoba კლასს აქვს Lift კლასის ერთი ობიექტი, სახელით lifti_1 ;

- Person კლასის ერთი ობიექტი იმყოფება mgzavri - ცვლადის შიგნით, რომელიც იყენებს lifti_1;

- Lift კლასის ობიექტს შეუძლია გადაადგილება ნებისმიერ სართულზე, ინტერვალში [1,N=60].



ნახ.4.27. კლასები და ობიექტები

- პროგრამაში სართულის არჩევა ხდება მგზავრის მიერ (გამოიყენება „შემთხვევით რიცხვთა გენერატორი“ Random-ფუნქციით);

- lifti_1 ლიფტი მოძრაობს საჭირო სართულისაკენ, რაც აისახება კონსოლზე;

- მგზავრ(ებ)ის ლიფტით მოძრაობის სეანსი შედგება რამდენიმე ეტაპისაგან, რომლზეც შეირჩევა ახალი მიზნობრივი სართულები;

- სეანსის ბოლოს გამოიცემა ანგარიშის ტექსტი (რეპორტი), თუ ჯამში რამდენი სართული გაიარა ლიფტმა;

- შუალედური და საბოლოო შედეგები გამოიტანება ეკრანზე.

2) კლასთა კავშირების აღწერა UML ტექნოლოგიით:

მომხმარებლის სამი კლასი: Shenoba, Lift, Person და ერთი სისტემური კლასი System.Random ამოდელირებს ლიფტის მუშაობის პროცესს:

- Shenoba-კლასი შეიცავს Lift-ობიექტს და იძახებს მის მეთოდებს;

- Lift-ობიექტი იყენებს Person-ობიექტს, რათა მართოს ლიფტის მოძრაობა;

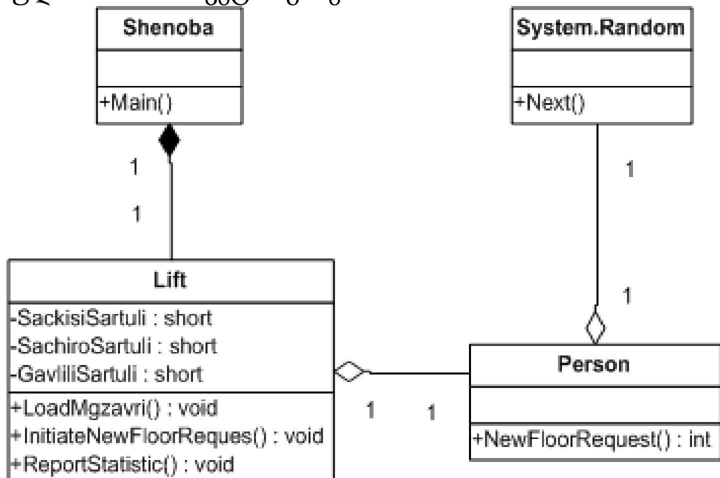
- Person-ობიექტი კი იყენებს System.Random-ობიექტს, რათა აირჩიოს საჭირო სართული შემდგომი გადაადგილებისთვის.

ობიექტორიენტირებული პროგრამული სისტემების დაპროექტებისას კლასებმა მსგავსი ფუნქციები უნდა შეასრულოს. თითოეულს უნდა ჰქონდეს თავისი უნიკალური ფუნქციურობა, და ყველა ერთად უნდა უზრუნველყოფდეს მთლიანი პროგრამის მუშაობას.

ნებისმიერი მაღლივი შენობა, როგორც „მთელი“ შედგება სხვადასხვა ნაწილისაგან: იატაკები, კედლები, ფანჯრები, ლიფტები და ა.შ. ამგვარად, შენობასა და ლიფტს შორის არსებობს დამოკიდებულება „მთელი-ნაწილი“ (აგრეგატული კავშირი UML ენაზე). ამიტომაცაა, რომ Lift-კლასის ობიექტის ცვლადი გამოცხადებულია Shenoba-კლასის შიგნით (ნახ.4.27, lifti_1). პროგრამულად იგი რეალიზებულია სტატიკური ცვლადის სახით (47-ე სტრიქონი, ლისტინგი_4.1): private static Lift lifti_1;

იგი ინახავს Lift კლასის ობიექტს და შეუძლია მისი მეთოდების გამოძახება. გარდა ამისა, Shenoba-კლასს შეიძლება ჰქონდეს აგრეთვე სხვა ცვლადებიც, რომლებიც აგრეგატულად დაქვემდებარებული კლასების ობიექტების ცვლადები იქნება.

4.28 ნახაზზე მოცემულია ჩვენი მაგალითის კლასებს შორის კავშირების UML დიაგრამა, აგებული Ms Visio პაკეტის გარემოში.



ნახ.4.28. კლასების დიაგრამა

Shenoba და Lift კლასებს შორის აგრეგაციას უწოდებენ კომპოზიციას და იგი შავი რომბით გამოისახება. „1“-ები ნიშნავს, რომ 1 შენობაში არის 1 ლიფტი (ჩვენს შემთხვევაში). Lift და Person, აგრეთვე Person და System.Random კლასებს შორის აგრეგატული მიმართებაა, მაგრამ არაკომპოზიციური. ის თეთრი რომბითაა ნაჩვენები. განსხვავება ისაა, რომ შენობას ლიფტი ყოველთვის აქვს. ლიფტში კი პერსონა შეიძლება იყოს, ან არ იყოს, ლიფტი ისედაც მუშაობს.

3) პროგრამული რეალიზაცია:

ცხრილში მოცემულია განხილული კლასების პროგრამული რეალიზაციის ლისტინგი.

- ■ private - ცვლადები
- o public - მეთოდები

class Lift

- ■ SackisiSartuli
- ■ SachiroSartuli
- ■ GavliliSartuli
- ■ SulSartulebi
- ■ Person mgzavri;

- O LoadMgzavri()
- O IniciateNewFloor Request()
- O ReportStatistic()
- O WriteLine()
- O Abs()

class Person

- ▪ randomNumber Generator;

- o Person()
- o NewFloorRequest()

```
1      using System;
2      namespace ConsoleApp_Lift
3      {
4       class Lift
5       {
6          private int SackisiSartuli = 1;
7          private int SachiroSartuli = 0;
8          private int GavliliSartuli = 0;
9          private int SulSartulebi = 0;
10         public  int i = 1;
11         private Person mgzavri;

12      public void LoadMgzavri()
13-15      {   mgzavri = new Person();  }
16     public void InitiateNewFloorRequest()
17      {
18        SachiroSartuli = mgzavri.NewFloorRequest();
19        GavliliSartuli = Math.Abs(SackisiSartuli -
                                         SachiroSartuli);
20        Console.WriteLine("{0,2}.| Sackisi: {1,2}|
              Sachiro: {2,2}| Gavlili: {3,2} ",  i.ToString(),
              SackisiSartuli.ToString(),
              SachiroSartuli.ToString(),
              GavliliSartuli.ToString());
21      // Math.Abs(SackisiSartuli - SachiroSartuli);
22        SulSartulebi += GavliliSartuli;
23         SackisiSartuli = SachiroSartuli;
24         i++;
25       }
26      public void ReportStatistic()
27      {   Console.WriteLine("\n===>>> Sul gavlili
28                          sartulebi: " + SulSartulebi);
29         }
30     }

31     class Person
32      {
33        private System.Random
                           randomNumberGenerator;
34       public Person()   // კონსტრუქტორი
35        {
36          randomNumberGenerator = new
                                 System.Random();
37          }
```

```
38
39      public int NewFloorRequest()
40        {
41          // აბრუნებს არჩეულ შემთხვევით
            //  რიცხვს 1-60 დიპაზონში
42          return randomNumberGenerator.Next(1,60);
43        }
44    }

45
46   class Shenoba   // კლასი შენობა
47    {
48     private static Lift lifti_1;
       private static void Main()
49      {
50        lifti_1 = new Lift();
51        lifti_1.LoadMgzavri();
         Console.WriteLine("      samgzavro sartulebi \n
52        ============================\n");
53        lifti_1.InitiateNewFloorRequest();
54        lifti_1.InitiateNewFloorRequest();
55        lifti_1.InitiateNewFloorRequest();
56        lifti_1.InitiateNewFloorRequest();
57        lifti_1.InitiateNewFloorRequest();
58        lifti_1.ReportStatistic();
59-60   }
   }}
```

- ობიექტი

class Shenoba

- lifti_1
  - LoadMgzavri()
  - InitiateNewFloor Request()
  - ReportStatistic()
  - WriteLine()

ლისტინგი_2.1: კონსოლის რეჟიმი

4) პროგრამის ლისტინგის ანალიზი

N	დანიშნულება
4	Lift - კლასის განსაზღვრების დასაწყისი
6	SackisiSartuli - ცვლადის გამოცხადება **int**-ტიპით, private-წვდომის სპეციფიკატორით და საწყისი მნიშვნელობით=1
11	Mgzavri - ცვლადის გამოცხადება, რომელიც შეიცავს Person- კლასის ობიექტს. Person- კლასი ასრულებს მგზავრის როლს Lift-კლასთან მიმართებაით
12	LoadMgzavri() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
14	Person-კლასის ახალი (**new**) ობიექტის შექმნა. ეს ობიექტი მიენიჭება ცვლადს - mgzavri

16	InitiateNewFloorRequest() - მეთოდის განსაზღვრების დასაწყისი. ის არის Lift-კლასის ინტერფეისის ნაწილი და გამოცხადებულია როგორც public
18	mgzavri-ობიექტისთვის NewFloorRequest() მეთოდის გამოძახება. ამ მეთოდით დაბრუნებული მნიშვნელობა მიენიჭება ცვლადს SachiroSartuli
19	ერთი მგზავრობის შემდეგ იანგარიშება გავლილი სართულების რაოდენობა
20	ეკრანზე გამოიტანება: საწყისი-სართული, საჭირო-სართული, გავლილი_სართულების_რაოდენობა
21	იანგარიშება სულ გავლილი სართულების რაოდენობა ლიფტის მუშაობის მთელი სეანსის დროს
22	ლიფტის საწყისი სართულის მნიშვნელობას ენიჭება მისი ბოლო გაჩერების სართულის მნიშვნელობა
23	ლიფტის ამუშავების მომდევნო ბიჯის ინკრემენტი
26	ReportStatistic() მეთოდის გამოძახებით ეკრანზე გამოიტანება სტატისტიკა SulSartulebi ცვლადით
31	Person კლასის განსაზღვრების დასაწყისი
33	randomNumberGenerator ცვლადის გამოცხადება System.Random კლასის ობიექტის შესანახად
34	სპეციალური მეთოდის (კონსტრუქტორის !) განსაზღვრების დასაწყისი, რომელიც გამოიძახება ავტომატურად Person კლასის ობიექტის შექმნის დროს
36	System.Random-კლასის ახალი ობიექტის შექმნა და მისი მინიჭება randomNumberGenerator - ცვლადზე
39	int ტიპის NewFloorRequest() მეთოდის განსაზღვრა. ის Person-კლასის ინტერფეისის ნაწილია
42	პერსონა (ვირტუალური მგზავრი) ლიფტში ირჩევს საჭირო სართულს (შემთხვევით რიცხვთა გენერატორი ასრულებს ამ ფუნქციას) დიაპაზონში [1-60]
45	Shenoba კლასის აღწერა
47	Shenoba კლასში გამოცხადებულია Lifti ტიპის ცვლადი Lifti_1. Shenoba კლასი კომპოზიციურ კავშირშია Lift კლასთან (ნახ.3.2)
48	Main( ) მეთოდის აღწერის დასაწყისი
49	Lifti კლასის ახალი ობიექტის შექმნა და მისი მინიჭება lifti_1 ცვლადზე

50	lifti_1 ობიექტისათვის LoadMgzavri( ) მეთოდის გამოძახება
52-56	lifti_1 ობიექტისათვის .IniciateNewFloorRequest( ) მეთოდის გამოძახება 5-ჯერ
57	lifti_1 ობიექტისთვის . ReportStatistic() მეთოდის გამოძახება (შედეგების გამოსაცემად ეკრანზე)

5) პროგრამის მუშაობის შედეგები:

4.29 ნახაზზე ნაჩვენებია განხილული პროგრამის შესრულების შედეგები კონსოლის რეჟიმში.

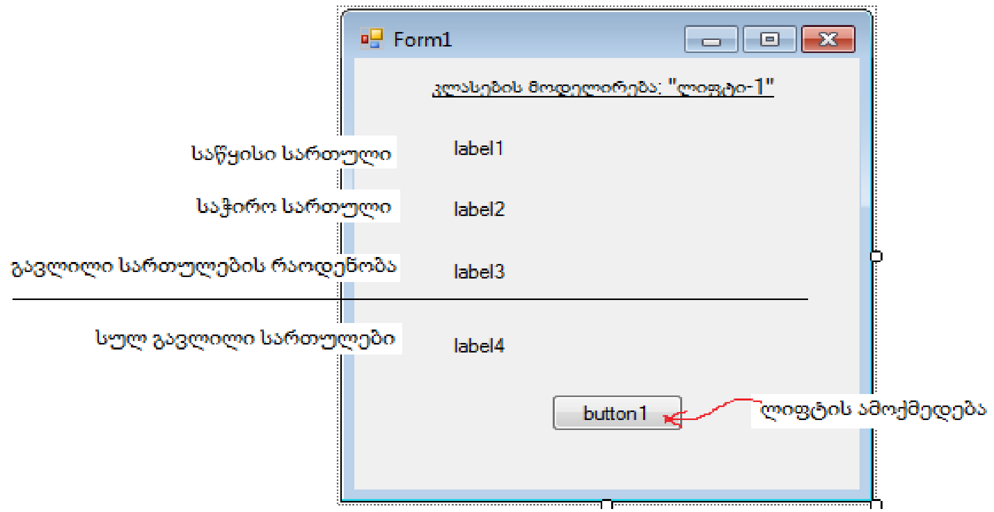
```
file:///C:/C#2010/ConsoleLift/ConsLift1/Con...
     samgzavro sartulebi
========================================

 1.| Sackisi:  1| Sachiro: 32| Gavlili: 31
 2.| Sackisi: 32| Sachiro: 58| Gavlili: 26
 3.| Sackisi: 58| Sachiro: 11| Gavlili: 47
 4.| Sackisi: 11| Sachiro: 37| Gavlili: 26
 5.| Sackisi: 37| Sachiro: 23| Gavlili: 14

===>>>        Sul gavlili sartulebi: 144
```

ნახ.4.29. შედეგები

ამით დავასრულეთ ჩვენი ამოცანის პირველი ნაწილი, კონსოლის რეჟიმში კლასების დაპროგრამების საილუსტრაციო მაგალითის განხილვა. აქ კარგად ჩანს პროგრამის მთლიანი კოდი და მისი შესაბამისობა UML კლასების დიაგრამასთან.

ახლა გადავიდეთ იგივე ამოცანის მეორე ნაწილზე, რომელშიც გამოყენებულ უნდა იქნას დაპროგრამების ვიზუალური საშუალებანი Windows Forms რეჟიმის საფუძველზე.

ამოცანა_4.2: განხილული ამოცანისათვის ავაგოთ პროგრამული კოდი კლასების საფუძველზე ვინდოუსის ფორმის რეჟიმში. 4.30 ნახაზზე ნაჩვენებია სამუშაო ფანჯარა, რომელიც Form კლასის Form1 ობიექტია. მასზე განთავსებულია ოთხი label (1,2,3,4) და ერთი button1, რომლითაც მოდელირდება ლიფტის ამოქმედება (ლიფტის გამოძახება, როცა პერსონა გარეთაა ან სართულის არჩევა ლიფტის ღილაკით, როცა პერსონა ლიფტშია, ანუ მგზავრია).

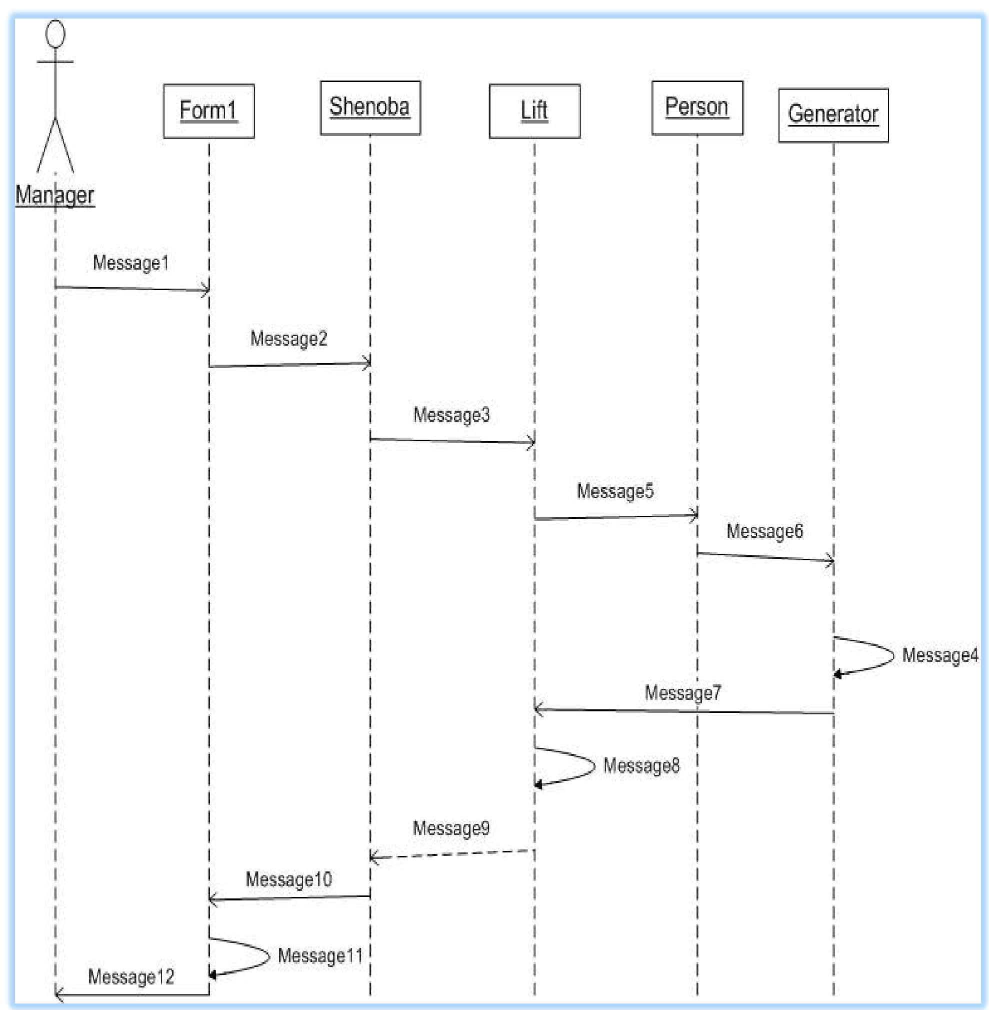


ნახ.4.30. ინტერფეისის ფორმა

```
// ლისტინგი_4.2--Form1  ელემენტებით და  button1-ის კოდი ---
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WinFormLift
{
   public partial class Form1 : Form
   {
      Shenoba shenoba_1;   // კლასი Shenoba  უნდა შეიქმნას
      public Form1() { InitializeComponent();  }
      private void button1_Click(object sender, EventArgs e)
      {  // shenoba_1  ობიექტის შექმნა
         shenoba_1 = new Shenoba(label1, label2, label3, label4);
      }
   }
}
```

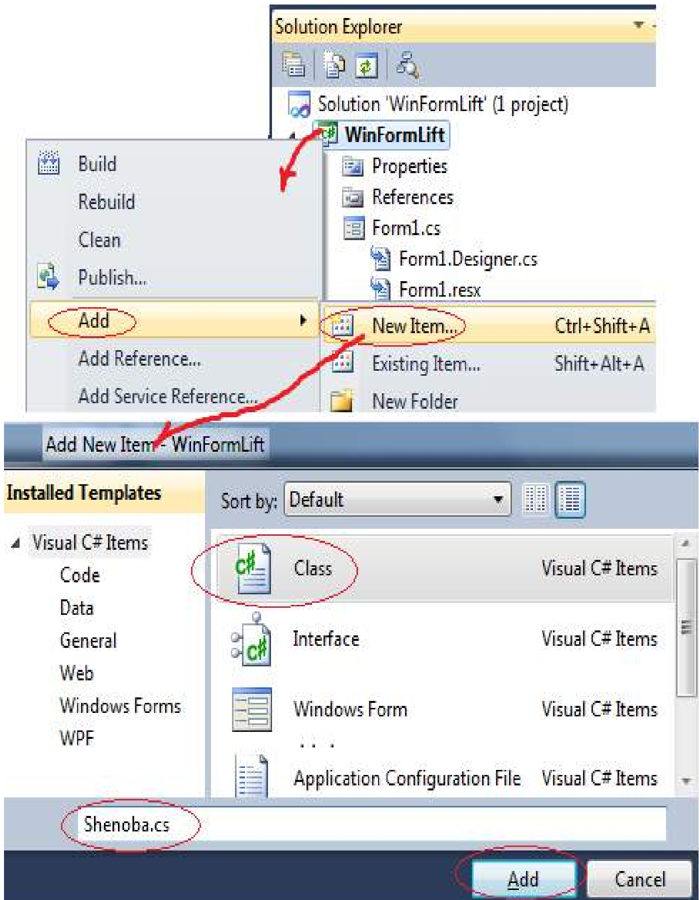
გარდა Form კლასისა (Form1 ობიექტით), რომელიც ასრულებს პროგრამის მომხმარებლის ინტერფეისის ფუნქციას, ლიფტის მუშაობის პროცესის ობიექტ-ორიენტირებული მოდელის ასაგებად საჭიროა კლასები: Shenoba, Lift და Person.

ამ კლასების თვისებები და ფუნქციურობა ჩვენ ზემოთ აღვწერეთ. ახლა საჭიროა ავაგოთ სცენარი „ლიფტის მუშაობა“, რომლისთვისაც გამოვიყენებთ UML-ის მიმდევრობითობის (Sequence) დიაგრამას [8,9] (ნახ.4.31).



ნახ.4.31. მიმდევრობითობის დიაგრამა

ახლა შევქმნათ დანარჩენი კლასები და აღვწეროთ მათი ფუნქციურობები. Solution Explorer-ში დავამატოთ (Add new Items) ახალი კლასები, როგორც ეს 4.32 ნახაზზეა ნაჩვენები.



ნახ.4.32. Shenoba-კლასის დამატება პროექტში

Shenoba კლასის კოდი მოცემულია 4.3_ ლისტინგში.

```
// ლისტინგი_4.3 --- კლასი Shenoba --------
using System;
using System.Windows.Forms;
namespace WinFormLift
{
  public class Shenoba
  {
   static Lift lift_1 = new Lift();

  public Shenoba(Label label1, Label label2, Label label3, Label label4)
```

```
  {
    lift_1.LoadMgzavri();
    lift_1.InitiateNewFloorRequest(label1,label2, label3);
    lift_1.ReportStatistic(label4);
  }
 }
}
```

Lift კლასის პროგრამული კოდი მოცემულია 4.4_ლისტინგში.

```
// ლისტინგი_4.4 --- კლასი Lift  --------
using System;
using System.Windows.Forms;
namespace WinFormLift
{
 public class Lift
  {  private int SackisiSartuli = 1;
    private int SachiroSartuli = 0;
    private int GavlilSartulebi = 0;
    private int SulSartulebi = 0;
    public int i;
    private Person mgzavri;
 public void LoadMgzavri()   { mgzavri = new Person();   }
 public void InitiateNewFloorRequest(Label label1, Label label2, Label label3)
   {
    SachiroSartuli = mgzavri.NewFloorRequest();
    GavlilSartulebi = Math.Abs(SackisiSartuli - SachiroSartuli);
    label1.Text = "საწყისი სართული: " + SackisiSartuli.ToString();
    label2.Text = "საჭირო სართული: " + SachiroSartuli.ToString();
    label3.Text = "გავლილი სართულები: " + GavlilSartulebi.ToString();
    SulSartulebi += GavlilSartulebi;
    SackisiSartuli = SachiroSartuli;
    i++;
   }
 public void ReportStatistic(Label label4)
  {
   label4.Text = "სულ გავლილი სართულები: " + SulSartulebi;
  }
 } }
```

Person კლასის პროგრამა მოცემულია 4.5_ლისტინგში.

```
// ლისტინგი_4.5 --- კლასი Person  --------
using System;
using System.Windows.Forms;
namespace WinFormLift
{   public class Person
    {  private System.Random randomNumberGenerator;
       public Person()    // კონსტრუქტორი
         {  randomNumberGenerator = new System.Random();  }
       public int NewFloorRequest()
         {   return randomNumberGenerator.Next(1, 60);  }
    }
 }
```

პროგრამის ამუშავების შემდეგ მიიღება შედეგები, რომლებიც ასახულია 4.33 ნახაზზე.

ნახ.4.33-ა. 1-ელი ბიჯი

ნახ.4.33-ბ. მე-2 ბიჯი

ნახ.4.33-გ. მე-3 ბიჯი

ნახ.4.33-დ. მე-10 ბიჯი,

და ა.შ.

4.4. რეფაქტორინგი: კოდის დამუშავება და რეორგანიზაცია

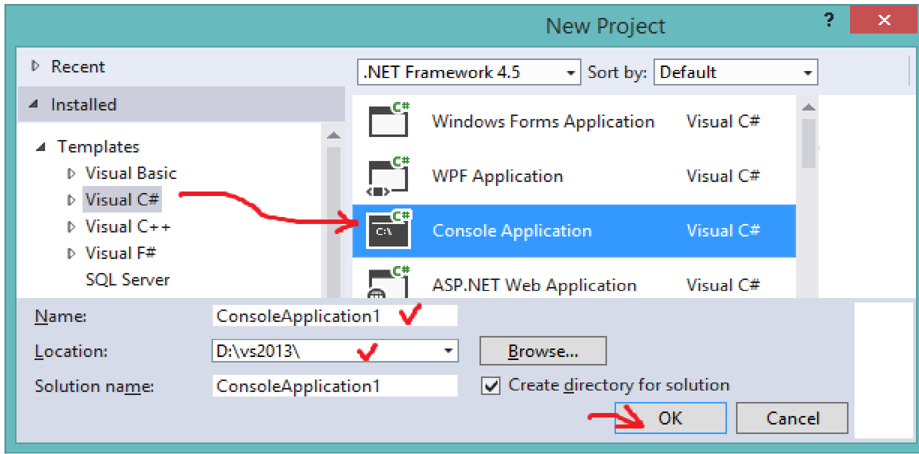
C# პროგრამული კოდის დამუშავებისა და მოდიფიკაციის პროცედურებს განვიხილავთ „რეფაქტორინგის“ ცნების გამოყენებით, არსებული კოდის ვერსიის განახლების მიზნით. პროგრამული კოდის დამუშავება ხდება არჩეული პროექტის ტიპის და კოდის შაბლონის მიხედვით, რომელსაც გვთავაზობს Visual Studio.NET 2013/15 სამუშაო გარემო. ამავე დროს ხორციელდება საწყისი კოდის სინტაქსის სისწორის შემოწმება, კოდის გენერირების ავტომატური დასრულება, აგრეთვე კოდში ნავიგაციის პროცესის გამოყენება (მაგალითად, კონტექსტური მენიუდან Go to definition -ით).

ყოველივე ეს მნიშვნელოვნად ზრდის პროგრამისტ-დეველოპერების მუშაობის მწარმოებლურობას!

„რეფაქტორინგი“ არის არსებული პროგრამული კოდის სისტემატური მოდიფიკაცია და სრულყოფა, მისი ფუნქციონირების სემანტიკის ძირფესვიანი ცვლილების გარეშე. კოდში ცვლილებები ხორციელდება ავტომატიზებული გარდაქმნებით, რომლებსაც .NET სამუშაო გარემო გვთავაზობს.

ამის ტიპური მაგალითია ***მეთოდის სახელის შეცვლა***. ამოცანა მდგომარეობს რომელიმე კონკრეტული მეთოდისთვის *დასახელების* და მისი *განსაზღვრების* ცვლილების განხორიელებაში, ამასთანავე *მისი გამოყენების ყველა ადგილზე*!

თუ პროექტი საკმაოდ დიდია, მაშინ **ხელით** ასეთი ცვლილებების ჩატარების ამოცანის გადაწყვეტა საკმაოდ შრომატევადი და მოუხერხებელია. არაა გამორიცხული შემთხვევა, რომ რომელიმე მოდულში დაგვავიწყდეს ამ ცვლილების განხორციელება, რაც შემდგომში, პროგრამის ამუშავებისას, აუცილებლად გამოჩნდება შეცდომის სახით და მოგვიწევს პროექტის ხელახალი შესწორება. განვიხილოთ რეფაქტორინგის ამოცანა მარტივი კონსოლური აპლიკაციისათვის (ნახ.4.34).



ნახ.4.34. კონსოლური აპლიკაციის შექმნა

მიღებული პროგრამის საწყის ტექსტს ჩავამატეთ რამდენიმე სტრიქონი, რომელსაც აქვს 4.35 ნახაზზე ნაჩვენები სახე.

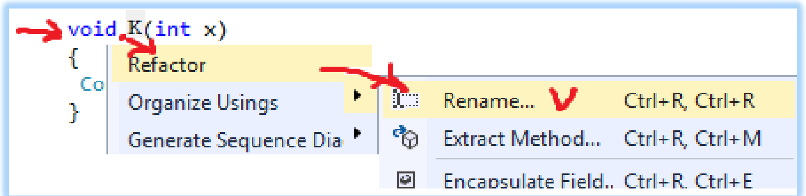
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Program p = new Program();
            Console.WriteLine("Hello Friend form VS 2013 !");
            p.K(1);
            Console.ReadLine();
        }

        void K(int x)
        {
         Console.WriteLine("x=", x);
        }
    }
}
```

ნახ.4.35. პროგრამის ტექსტი

პროგრამაში განსაზღვრულია Program კლასი, სტატიკური მეთოდი Main და K-ეგზემპლარის მეთოდი **x** არგუმენტით.

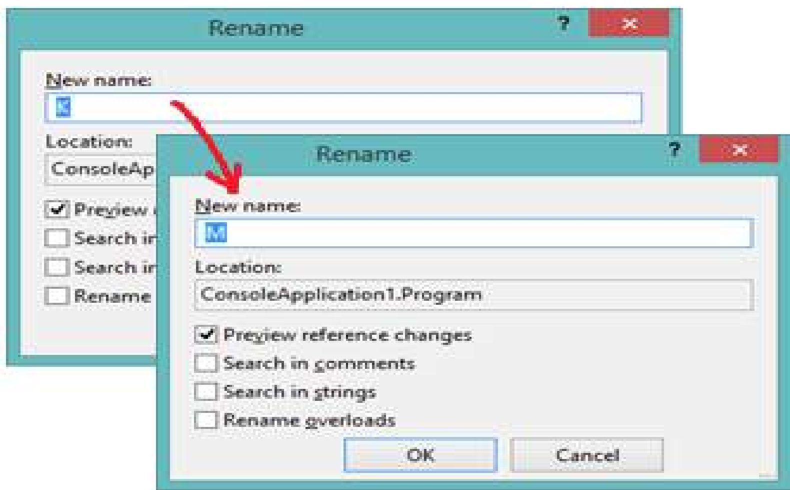
დავუშვათ, რომ საჭიროა შეიცვლაოს K მეთოდის სახელი M-ით. ცვლილება უნდა განხორციელდეს არა მხოლოდ ამ მეთოდის განსაზღვრებაში, არამედ მისი გამოყენების ადგილებზეც !

კოდის რედაქტირების გარემოში მაუსის კურსორს ვაყენებთ K მეთოდის პირველ სტრიქონზე, მოვნიშნავთ შესაცვლელ სიტყვას და კონტექსტურ მენიუდან ვირჩევთ პუნქტს Refactor / Rename (ნახ.4.36).



ნახ.4.36. K მეთოდისათვის რეფაქტორინგის ქმედების არჩევა

4.37 ნახაზზე ნაჩვენებია შეცვლის პროცედურა New name ველში.



ნახ.4.37. K-იცვლება M-ით

მიიღება 4.38 სურათი.

თუ ყველაფერი წესრიგშია, მაშინ ვირჩევთ Apply-ს და ვიღებთ რეფაქტორინგის შედეგს (ნახ.4.39).

VS 2013 სამუშაო გარემოს აქვს რეფაქტორინგის შემდეგი შესაძლებლობები:

- Rename - სახელის შეცვლა;
- Extract Method - მეთოდის ამოღება: კოდის მონიშნული ფრაგმენტის მოდიფიკაცია მეთოდში მითითებული სახელით;
- Encapsulate Field - ველის ინკაფსულირება, მისი private-დ გარდაქმნა, ოღონდ public-თვისების დამატებით, წვდომის მიზნით;
- Extract Interface - ინტერფეისის ამღება: კლასის ტექსტის მონიშვნა მისთვის ავტომატურად შესაბამისი ინტერფეისის ფორმირება (თუ ეს შესაძლებელია);
- Remove parameters - მეთოდის პარამეტრების ნაწილის წაშლა;
- Reorder parameters - მეთოდის პარამეტრების რიგითობის შეცვლა.

```
namespace ConsoleApplic
{
    2 references
    class Program
    {
        0 references
        static void Mai
        {
            Program p =
            Console.Wri
            p.K(1);
            Console.Rea
        }

        1 reference
        void K(int x)
        {
         Console.WriteL
        }
    }
}
```

Preview Changes - Rename

Rename 'K' to 'M':

ConsoleApplication1.Program.K(int x)
Program.cs
p.K(1);

Preview Code Changes:

```
            Console.WriteLine("Hello Friend form VS 2013 !");
            p.M(1);
            Console.ReadLine();
        }

        void M(int x)
        {
         Console.WriteLine("x=", x);
        }
    }
}
```

Apply Cancel

ნახ.4.38. დაგეგმილი ცვლილებების წინასწარ ნახვა (Preview Changes)

Program.cs

ConsoleApplication1.Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    2 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Program p = new Program();
            Console.WriteLine("Hello Friend form VS 2013 !");
            p.M(1);
            Console.ReadLine();
        }

        1 reference
        void M(int x)
        {
         Console.WriteLine("x=", x);
        }
    }
}
```

ნახ.4.39. რეფაქტორინგის შედეგი

4.5. პროგრამული აპლიკაციების ტესტირება

განვიხილოთ C# პროგრამული კოდების ტესტირების პროცესები Visual Studio.NET ინტეგრირებულ გარემოში.

Unit testing და Coded UI არის Microsoft-ის ტესტირების ინსტრუმენტები, რომლებიც სრულდება Visual Studio.NET-გარემოში.

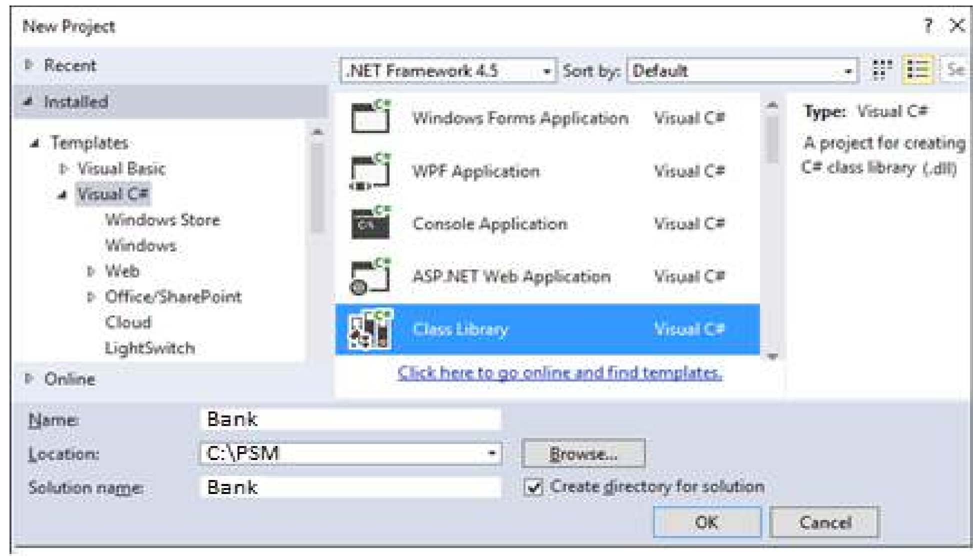
Unit testing ანუ მოდულური ტესტირება დაპროგრამების პროცესია, რომლის საშუალებითაც მოწმდება საწყისი კოდის ცალკეული მოდულების კორექტულობა. ასეთი ტესტირების იდეა მდგომარეობს იმაში, რომ ყოველი არატრივიალური ფუნქციის ან მეთოდისათვის დაიწეროს ტესტი. ეს უზრუნველყოფს კოდის სწრაფად შემოწმებას, ხომ არ მიიყვანა კოდის ბოლო ცვლილებამ პროგრამა რეგრესიამდე, ანუ შეცდომების გაჩენამდე პროგრამის უკვე ტესტირებულ ნაწილებში [10].

Coded UI ტესტი კი ავტომატურად იწერს, შესრულებაზე უშვებს და ამოწმებს ტესტქეისებს.

ასეთი ტესტების წერა შესაძლებელია C# ან Visual Basic-ზე Visual Studio გარემოში. Unit testing ტესტირების ტექნოლოგია განვიხილოთ ვირტუალური ობიექტის, მაგალითად, ფინანსური ობიექტის, ბანკის მაგალითზე.

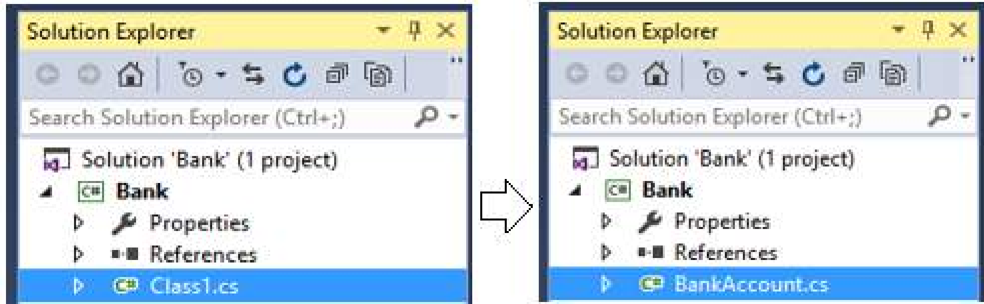
- დასატესტი პროგრამის პროექტის შექმნა: Visual Studio-ში საჭიროა ავირჩიოთ: **File => New => Project.**

შედეგად გამოჩნდება დიალოგური ფანჯარა (ნახ.4.40). სადაც ავირჩევთ:
Visual C# => ClassLibrary და პროექტი სახელით Bank.



ნახ.4.40. დასატესტი კლასის პროექტის შექმნა

მიიღება 4.41 ნახაზზე ნაჩვენები Solution E[plorer ფანჯარა. აქ Class1.cs სახელი შევცვალოთ BankAccount.cs -ით.



ნახ.4.41. Class1 - > BankAccount

შემდეგ BankAccount.cs -ის ტექსტი რედაქტორის არეში შევცვალოთ ჩვენი დასატესტი პროგრამის კოდით.

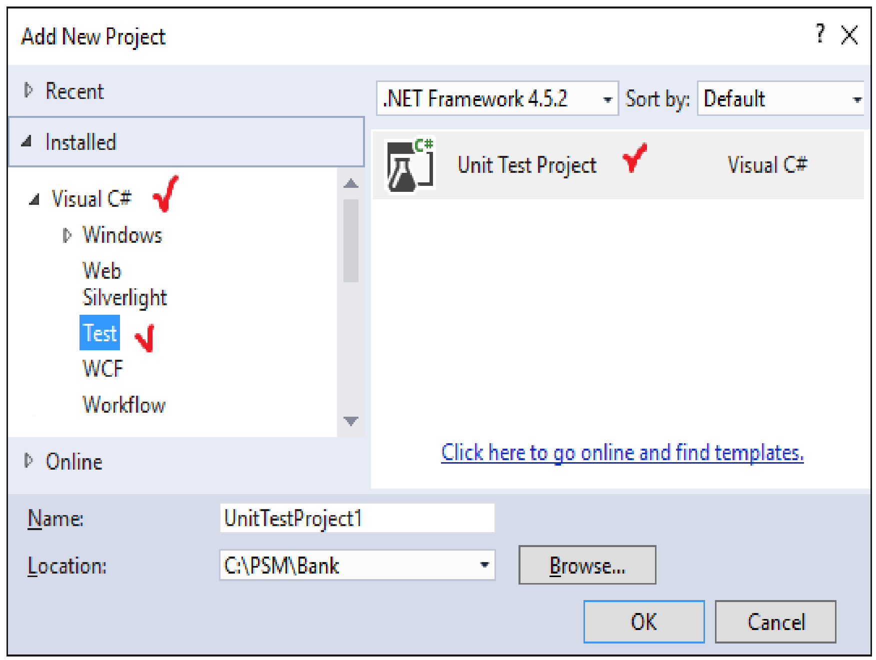
ეს საწყისი ტექსტი, მაგალითად, მოცემულია 4.4 ლისტინგში.

```
//-- ლისტინგი_4.4 --- BankAccount.cs --------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace BankAccountNS
{
 public class BankAccount
  {
    private string m_customerName;
    private double m_balance;
    private bool m_frozen = false;
   private BankAccount() {   }
   public BankAccount(string customerName, double balance)
    {
      m_customerName = customerName;
      m_balance = balance;
    }
   public string CustomerName
    {
      get { return m_customerName; }
```

```
 }
public double Balance
 {
   get { return m_balance; }
 }
public void Debit(double amount)
 {
   if (m_frozen)
    {
      throw new Exception("Account frozen");
    }
   if (amount > m_balance)
    {
       throw new ArgumentOutOfRangeException("amount");
     }
    if (amount < 0)
     {
       throw new ArgumentOutOfRangeException("amount");
     }
     m_balance += amount; // განზრახ არასწორი კოდი
     // m_balance -= amount;   // გასწორებული
  }
 public void Credit(double amount)
  {
    if (m_frozen)
      {
       throw new Exception("Account frozen");
      }
    if (amount < 0)
     {
       throw new ArgumentOutOfRangeException("amount");
     }
     m_balance += amount;
  }
 private void FreezeAccount()
  {
     m_frozen = true;
```

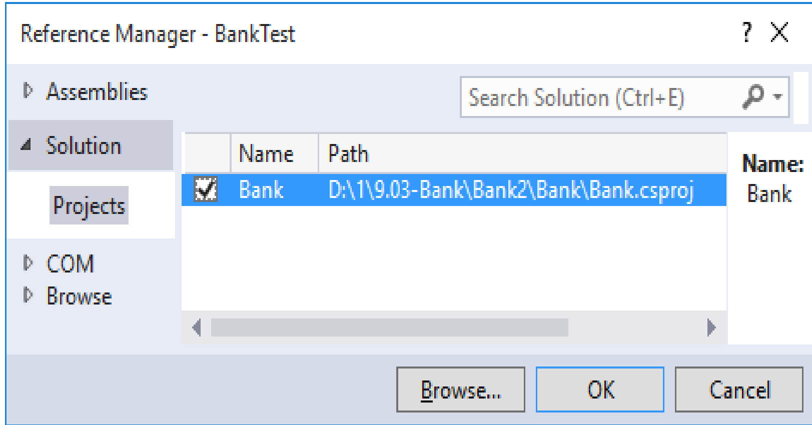
```
    }
    private void UnfreezeAccount()
    {
        m_frozen = false;
    }
    public static void Main()
    {
     BankAccount ba = new BankAccount("Mr.Bryan Walton",
                                        9.99);
     ba.Credit(5.77); ba.Debit(9.22);
     Console.WriteLine("Current balance is ${0}",
                                   ba.Balance);
    }
  }
}
```

- ტესტ-ფაილის პროექტის აგება (Unit Test Project):



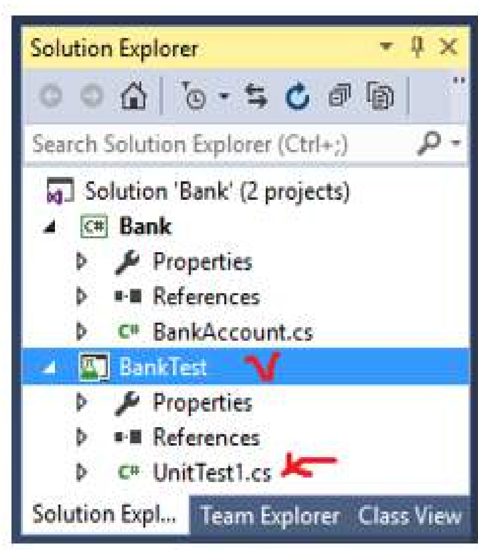
ნახ.4.42. Unit Test პროექტის შექმნა

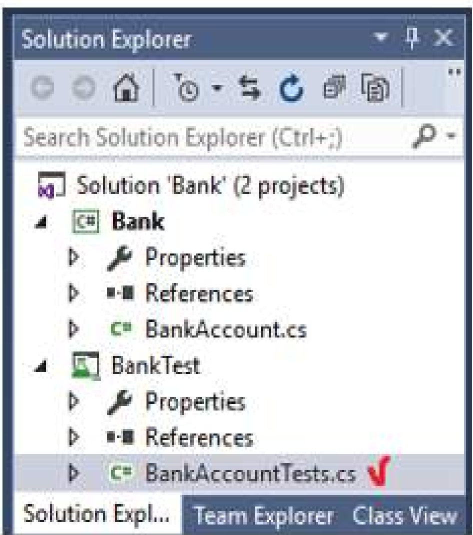
მივიღებთ 4.43 ნახაზზე ნაჩვენებ სურათს BankTest პროექტით. მარჯვენა სურათზე შეცვლილია UnitTest კლასის სახელი BankAccountTests-ით.



ნახ.4.43

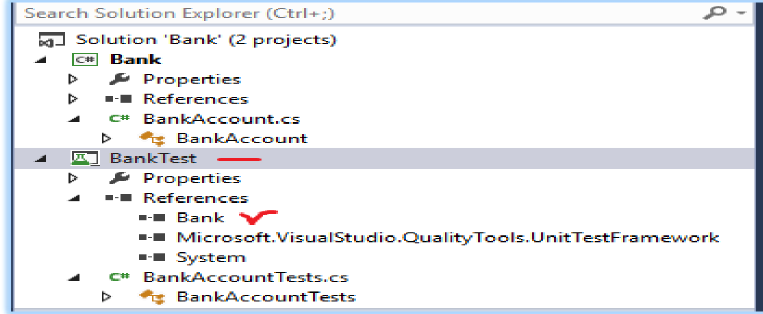
BankTests პროექტში დავამატოთ reference **Bank** solution-იდან. ამისათვის **BankTests**-ზე მაუსის მარჯვენა ღილაკით ავირჩიოთ **Add Reference** და მივიღებთ 4.44 ნახაზზე ნაჩვენებ ფანჯარას. აქ Solution სტრიქონში ვირჩევთ Projects და Bank-ის ჩევბოქსს მოვნიშნავთ.





ნახ.4.44

მივიღებთ 4.45 ნახაზზე ნაჩვენებ შედეგს.



ნახ.4.45

ჩავამატოთ BankAccountTests პროგრამაში სახელსივრცე Bank-ის პროექტიდან:

```
using BankAccountNS;
```

ამგვარად, BankAccountTests.cs ფაილს ექნება 4.5 ლისტინგზე ნაჩვენები სახე.

//-- ლისტინგი_4.5 ----- BankAccountTests.cs -----

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;

namespace UnitTestProject1
{
    [TestClass]
    public class BankAccountTests
    {   [TestMethod]
        public void TestMethod1() {    }
    }
}
```

ახლა შევქმნათ პირველი ტესტმეთოდი. ამ პროცედურაში, დაიწერება Unit-ტესტის მეთოდები BankAccount class-ის Debit მეთოდის ქცევის ვერიფიკაციისათვის. ეს მეთოდები ზემოთაა ჩამოთვლილი.

დასატესტი მეთოდების ანალიზის გზით გაირკვა, რომ საჭიროა მინიმუმ სამი ქცევის შემოწმება:

1. მეთოდი ქმნის ArgumentOutOfRangeException-გამონაკლისს, თუ კრედიტის ჯამი გადააჭარბებს ბალანსს;
2. იგი ქმნის ArgumentOutOfRangeException-გამონაკლისს მაშინაც, როცა კრედიტის ზომა უარყოფითია;
3. თუ 1 და 2 პუნქტები წარმატებით დასრულდა, მაშინ მეთოდი ჯამს ბალანსის ანგარიშიდან ითვლის.

პირველ ტესტში შევამოწმოთ, რომ კრედიტის დასაშვები მნიშვნელობისათვის (როცა დადებითი მნიშვნელობისაა და ბალანსის ანგარიშზე ნაკლებია) ანგარიშიდან მოიხსნება საჭირო თანხა.

1. დავამატოთ BankAccountTests კლასს შემდეგი მეთოდი:

```
// unit test code -----
[TestMethod]
public void Debit_WithValidAmount_UpdatesBalance()
{
  // arrange
   double beginningBalance = 9.99;
   double debitAmount = 4.55;
   double expected = 7.44;
   BankAccount account = new BankAccount("Mr. Dito", beginningBalance);
   // act
   account.Debit(debitAmount);
   // assert
   double actual = account.Balance;
   Assert.AreEqual(expected, actual, 0.001, "Account not debited correctly");
}
```

მეთოდი საკმაოდ მარტივია. ჩვენ ვქმნით ახალ BankAccount ობიექტს საწყისი ბალანსით და შემდეგ ვაკლებთ სწორ ოდენობას. ჩვენ ვიყენებთ Microsoft-ის unit-ტესტის ფრეიმვორკს მართვადი კოდის AreEqual მეთოდისათვის, რათა მოხდეს საბოლოო ბალანსის ვერიფიკაცია - არის ის, რასაც ჩვენ ველით.

ტესტმეთოდის მოთხოვნები ასეთია:

1. მეთოდი მონიშნული უნდა იყოს [TestMethod] ატრიბუტით;
2. მეთოდმა უნდა დააბრუნოს void;
3. მეთოდს არ შეიძლება ჰქონდეს პარამეტრები.

ტესტის აგება და ამუშავება:

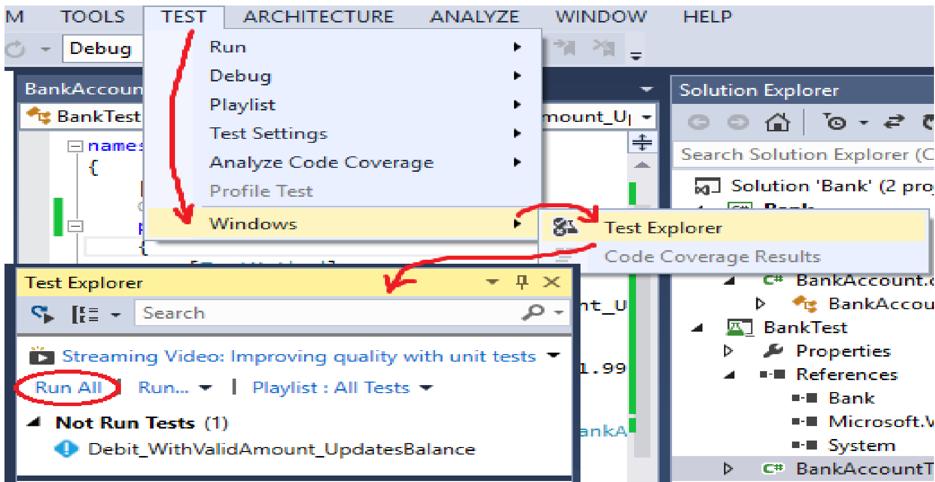
მთლიანი ტესტის კოდი მოცემულია 4.6 ლისტინგში.

```
//-- ლისტინგი_4.6 ----- BankAccountyTests.cs ----
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using BankAccountNS;
namespace UnitTestProject1
{
```

```
  [TestClass]
  public class BankAccountTests
  {
     [TestMethod]
     public void Debit_WithValidAmount_UpdatesBalance()
     { // arrange
       double beginningBalance = 9.99;
       double debitAmount = 4.55;
       double expected = 7.44;
       BankAccount account = new BankAccount("Mr. Dito",
                     beginningBalance);
       // act
       account.Debit(debitAmount);
       // assert
       double actual = account.Balance;
       Assert.AreEqual(expected, actual, 0.001, "Account
               not debited correctly");
     }
  }
}
```

- BUILD მენიუდან ვირჩევთ Build Solution;

- TEST მენიუდან ვირჩევთ Windows და Test Explorer პუნქტებს. იხსნება Test Explorer ფანჯარა (ნახ.4.45).



ნახ.4.45. Test Explorer ფანჯარა

აქ ვირჩევთ Run All - ს და ვიღებთ შედეგს (ნახ.14.8).

Test Explorer
Search
Streaming Video: Improving quality with unit tests
Run All | Run... | Playlist : All Tests
Failed Tests (1)
Debit_WithValidAmount_UpdatesBalance 74 ms
Summary
Last Test Run Failed (Total Run Time 0:00:01)
1 Test Failed

```
namespace UnitTestProject1
{
    [TestClass]
    0 references
    public class BankAccountTests
    {
        [TestMethod]
        0 references
        public void Debit_WithValidAmount_UpdatesBalance()
        {
            // arrange
            double beginningBalance = 11.99;
            double debitAmount = 4.55;
            double expected = 7.44;
            BankAccount account = new BankAccount("Mr. Dito",
            . . .
```

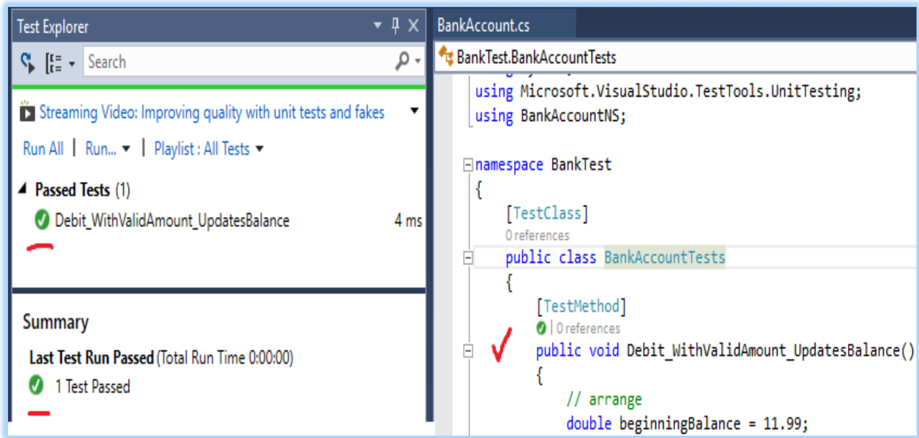
ნახ.4.46. ტესტირების შედეგები

ნახაზზე მითითებული „x“-სიმბოლოები წითელ წრეშია მოთავსებული, ე.ი. ტესტირებამ აღმოაჩინა შეცდომები და კოდი ვერ შესრულდა წარმატებით.

თუ მეთოდი წარმატებით ჩაივლიდა, მაშინ მივიღებდით მწვანე ფერის x-სიმბოლოებს. შემდეგი ეტაპი კოდის გასწორება და ხელახალი ტესტირებაა. დასატესტ პროგრამაში შევცვალოთ სტრიქონში „ + „ ნიშანი „ – „ -ით.

```
// m_balance += amount; // intentionally incorrect code
m_balance -= amount; // intentionally correct code
```

ტესტის თავიდან ამუშავებით ვიღებთ წარმატებულ შედეგს, ანუ მიიღება მწვანე ფერის სიმბოლოები (ნახ.4.47).

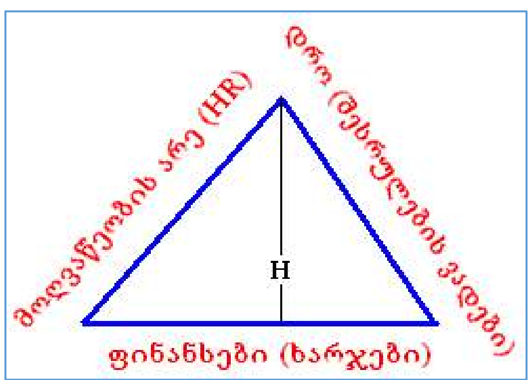


ნახ.4.47. სწორი შედეგი

V თავი
გამოყენებითი პროგრამული სისტემების მენეჯმენტი

5.1. პროექტების მენეჯმენტი

მენეჯერის საქმიანობა პროექტის შესრულების მიზნის მიღწევაა, რომელიც განიხილება პროექტის ისეთი პარამეტრების შეთანხმებით, როგორიცაა: *მოღვაწეობის არეალი, ვადები* და *ფინანსები* [55,110]. ამ პარამეტრების ურთიერთკავშირის ანალიზური და გრაფიკული ინტერპრეტაციის სხვადასხვა ხერხი არსებობს. ერთ-ერთი მარტივი სქემა 5.1 ნახაზზეა მოცემული, ე.წ. „პროექტების მენეჯმენტის სამკუთხედი"-ის სახით.



ნახ.5.1. პროექტების მენეჯმენტის სამკუთხედი

პროგრამული სისტემის მენეჯერის ამოცანა შეიძლება ასე ავხსნათ: საჭიროა **მოღვაწეობის არეალის** (კადრების მწარმოებლურობა გარკვეული სამუშაოს შესასრულებლად), **დროის** (სამუშაოს შესრულების ვადების) და **ფინანსების** (რესურსების ხარჯვის) ისეთი შეთანხმება, რომელიც დააკმაყოფილებს **ხარისხის** მოთხოვნებს. ასეთი ბალანსის დამყარება, შესასრულებელი პროცესების სირთულისა და არაერთგვაროვნობის გამო, საკმაოდ რთულია. ამიტომ მენეჯერი მიზნობრივად ირჩევს ერთ ან ორ პარამეტრს, დანარჩენს კი მათზე დამოკიდებულად. ამგვარად, სამკუთხედი იღებს ასეთ ინტერპრეტაციას: „კარგად-სწრაფად-იაფად: აირჩიეთ ორი მათ შორის".

5.1 ნახაზზე სამკუთხედის გვერდების სიგრძეების (ცვლადების) მნიშვნელობათა ვარირებით მიიღება სხვადასხვა შედეგი. ამ პარამეტრთა ურთიერთდამოკიდებულების ასახვის ინვარიანტულ მაჩვენებლად შემოაქვთ სამკუთხედის ფართობი

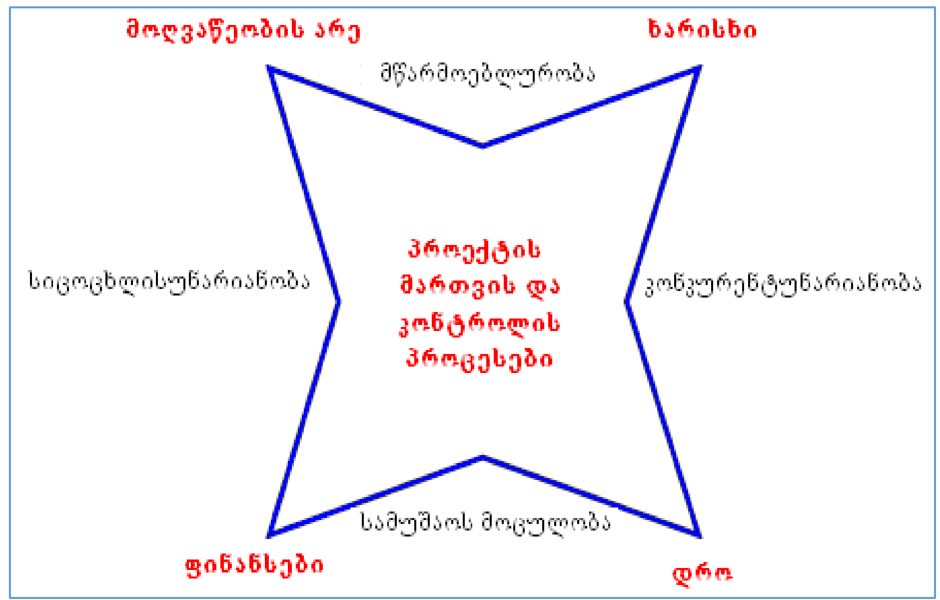
(S = a * H/2),

რომელიც კონკრეტული პროექტისა და კონკრეტული შემსრულებელი გუნდისათვის უცვლელია (კონსტანტაა). აქედან გამომდინარე, თუ სამკუთხედში საწყისად შერჩეულ

იქნება ორი გვერდი, მაშინ მესამის გამოთვლა ყოველთვის შესაძლებელია მისი S-ფართობის და H-სიმაღლის გამოყენებით.

პროექტების მართვის ინსტიტუტის (*PMBOK*) მეცნიერის, რ. ვიდემანის მიერ შემოტანილ იქნა მე-4 პარამეტრი - *ხარისხი* (ნახ.5.2) [110].

მიღებულია ოთხქიმიანი ვარსკვლავის სქემა, რომელშიც, მიუხედავად შინაარსობრივი აღწერის უკეთ შესაძლებლობისა, დაიკარგა სამკუთხედის სიმარტივის პრინციპი და მნიშვნელოვანი ინვარიანტული პარამეტრი - ფართობი.

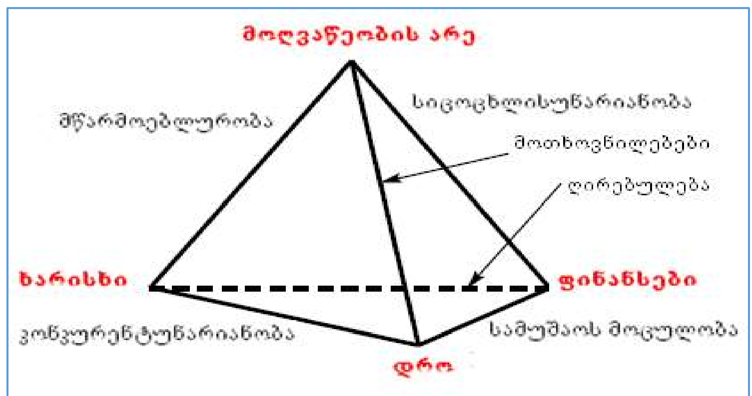


ნახ.5.2

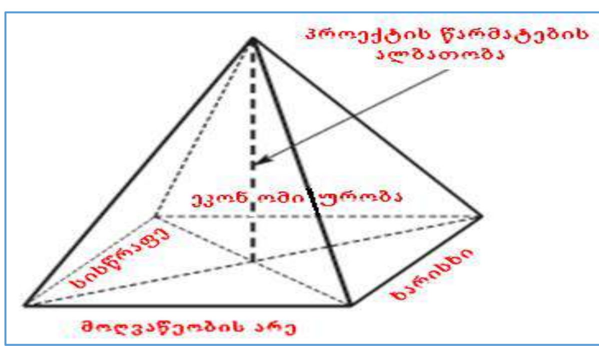
აღნიშნული პრობლემის გადაჭრა განხორციელდა სამგანზომილებიან სივრცეში გადასვლით, სადაც შესაძლებელი გახდა ინვარიანტული ცვლადის გამოყენება, ამჯერად ტეტრაედრის მოცულობის სახით (ნახ.5.3) [110].

მესამე განზომილების შემოტანის საფუძველზე ჯ. მარასკომ მოახერხა „საპროექტო პირამიდის" აგება (ნახ.5.4), რომელშიც კიდევ ერთი ახალი პარამეტრია დამატებული, როგორიცაა პროექტის წარმატების ალბათობა (Probability of Success).

პირამიდის ფუძე ოთხკუთხედია, რომლის გვერდებია: სისწრაფე (Speed), ეკონომიურობა (Frugality), ხარისხი (Quality) და მოღვაწეობის არე (Scope). პირამიდის სიმაღლეს შეესაბამება პროექტის წარმატების ალბათობა. ეს ხუთი პარამეტრი განიხილება როგორც ცვლადები, რომელთა მნიშვნელობები დაკავშირებულია პროექტის განვითარების პირობების ინვარიანტთან მისი შემსრულებელი გუნდის ძალებით. ინვარიანტად მიიღება პირამიდის მოცულობა.



ნახ.5.3



ნახ.5.4.

აღნიშნული მოდელით უფრო რეალურადაა შესაძლებელი პროექტის მენეჯერის საქმიანობაზე დაკვირვება პროექტის შესრულებისა და კორექტირების პროცესებში. ვინაიდან პროექტის წარმატების ალბათობა დროში ცვალებადია, ხოლო პირამიდის მოცულობა მოცემული პროექტისათვის მუდმივია, მაშინ პირამიდის სიმაღლის ცვლილება აუცილებლად გამოიწვევს შედეგად პირამიდის ფუძის ცვლადების შეცვლას. ასევე მართებულია უკუ-გამონათქვამი, რომ პროექტის წარმატების ალბათობა დამოკიდებულია პირამიდის ფუძის ფართობისა.

ამგვარად, პროექტების მართვის ე.წ. „მენეჯმენტის სამკუთხედი“, „მენეჯმენტის პირამიდა“ და ა.შ. წარმოადგენს საილუსტრაციო მოდელებს, რომელთა საშუალებითაც შესაძლებელია პროგრამული სისტემების პროექტირების პროცესების უფრო ღრმად გამოკვლევა.

მენეჯერის მეთოდური მუშაობის ზოგადი სქემა მდგომარეობს შემდეგში: თავიდან განისაზღვრება საპროექტო საქმიანობა, შემდეგ ირკვევა, თუ რა კორექტირებებია შესასრულებელი და რა საშუალებები არსებობს ამისათვის. მენეჯერის საქმიანობის შედეგები ძალზე მრავალფეროვანია, მაგრამ ისინი თავსდება გარკვეულ ჩარჩოებში ბალანსის დამყარების თვალსაზრისით პროექტის მიზნის მისაღწევად.

5.2. პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის მოდელები

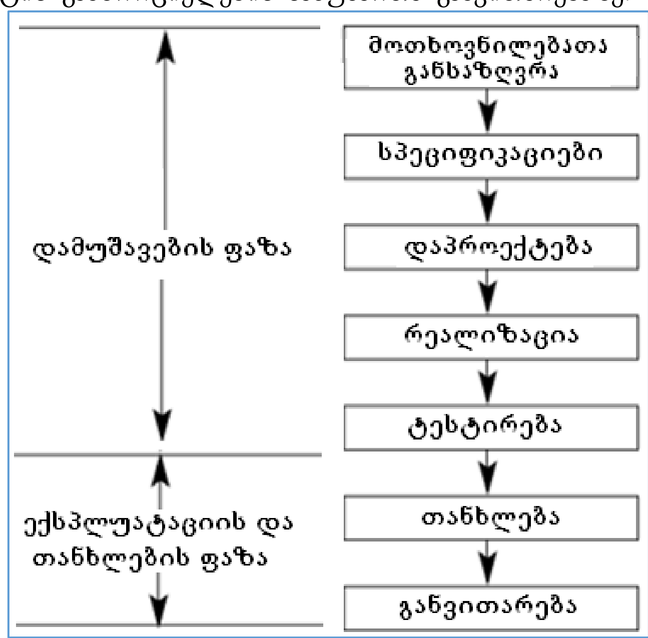
- **პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის საბაზო მოდელი**

  - პროგრამული უზრუნველყოფის სასიცოცხლო ციკლი არის პროგრამული უზრუნველყოფის კონსტრუირების პროცესი. პროგრამულ პროექტზე მოთხოვნილებანი განსაზღვრავს ამ პროცესის მიზანს და რეგლამენტს.

5.5 ნახაზზე ნაჩვენებია პროგრამული სისტემის სასიცოცხლო ციკლის ტრადიციულად მიღებული მოდელი. იგი 2 ფაზისა და 7 ეტაპისაგან შედგება [55].

თითოეული ეტაპი პროექტის მენეჯერისა და შემსრულებელთა გარკვეული როლით ხასიათდება. თუ ჩვენ განვიხილავთ IBM-კომპანიის ცენტრის ობიექტ-ორიენტირებულ ტექნოლოგიას, მაშინ შეიძლება გამოვყოთ საკმაოდ სრული სია ტიპური როლებისა, რომლებიც პროგრამული პროექტების შესრულებასა და განვითარებას ემსახურება.

ესაა პროექტში მონაწილეთა როლები პროგრამული უზრუნველყოფის ფირმებიდან და დამკვეთი კორპორაციებიდან, რომლებიც გავლენას ახდენენ როგორც პროექტის ამოცანების ჩამოყალიბებაზე, ისე შესაბამისი რესურსების გამოყოფასა და საერთოდ პროექტის განხორციელების სამუშაოთა განვითარებაზე.



ნახ.5.5. სასიცოცხლო ციკლის საბაზო მოდელი

როლების დახასიათება მოიცავს შესასრულებელი ორგანიზაციული და საწარმოო ფუნქციების ერთობლიობას.

- დამკვეთი (Customer) – პროექტის რეალურად არსებული ინიციატორი, რომელსაც ემორჩილება საპროექტო გუნდი, ან ორგანიზაციის სხვა წარმომადგენელი, რომელსაც დავალებული აქვს პროექტის ხელშეწყობა, შესრულების კონტროლი და შედეგების მიღება;
- რესურსების მგეგმავი (Planner) – წარმოადგენს და აკოორდინირებს ორგანიზაციაში პროექტისადმი მოთხოვნებს, სადაც მუშავდება პროექტი. აგრეთვე ავითარებს და წარმართავს პროექტის შესრულების გეგმას, ორგანიზაციის თვალსაზრისით;
- პროექტის მენეჯერი (Project Manager) პასუხისმგებელია მთლიანად პროექტის შესრულებაზე, დავალებებისა და რესურსების განაწილებაზე, სამუშაოთა შედეგების მიღებაზე გრაფიკის მიხედვით, შედეგების შესაბამისობაზე მოთხოვნებთან. იგი აქტიურად თანამშრომლობს დამკვეთთან და რესურსების დამგეგმავთან;
- გუნდის ხელმძღვანელი (Team Leader) ახორციელებს გუნდის ტექნიკურ ხელმძღვანელობას პროექტის შესრულების პროცესში. დიდ პროექტებში შეიძლება რამდენიმე ხელმძღვანელის არსებობა ქვეგუნდების მიხედვით, რომლებიც ცალკეულ ამოცანებზე იქნება პასუხისმგებელი;
- არქიტექტორი (Architect) პასუხისმგებელია სისტემის არქიტექტურის დაპროექტებაზე, ათანხმებს პროექტთან დაკავშირებულ სამუშაოთა განვითარებას;
- ქვესისტემის დამპროექტებელი (Designer) პასუხს აგებს ქვესისტემის ან კლასთა კატეგორიების დაპროექტებაზე, განსაზღვრავს რეალიზაციისა და ინტერფეისების საკითხებს სხვა ქვესისტემებთან;
- საგნობრივი სფეროს ექსპერტი (Domain Expert) პასუხისმგებელია საკვლევი სფეროს დანართის ბიზნეს-პროცესების შესწავლასა და მოდელირებაზე, ახორციელებს პროექტის მხარდაჭერას ამ სფეროს ამოცანების გადასაწყვეტად;
- დამმუშავებელი (Developer) ახორციელებს საპროექტო კომპონენტების რეალიზაციას, ფლობს და ქმნის სპეციფიკურ კლასებსა და მეთოდებს, აგებს კოდებს და ატესტირებს მათ, ქმნის პროგრამულ პროდუქტს;
- ინფორმაციული მხარდაჭერის დამმუშავებელი (Information Developer) – ქმნის პროგრამული პაკეტის თანმხლებ დოკუმენტაციას. მას თან ურთავს სისტემის საინსტალიაციო მასალებს, მომხმარებლის ინსტრუქციებს;
- სამომხმარებლო ინტერფეისის სპეციალისტი (Human Factors Engineer) პასუხისმგებელია სისტემის ადვილად და მოხერხებულად გამოყენებაზე. მუშაობს დამკვეთთან, რათა დარწმუნდეს, რომ მომხმარებლის ინტერფეისი აკმაყოფილებს მოთხოვნებს;
- ტესტირების სპეციალისტი (Tester) ამოწმებს პროდუქტის ფუნქციურობას, ხარისხს და ეფექტურობას. აგებს და იყენებს საკონტროლო ტესტებს პროექტის ყოველი ფაზისათვის;

- ბიბლიოთეკარი (Librarian) პასუხს აგებს პროექტის საერთო ბიბლიოთეკის შექმნასა და მოვლაზე, რომელიც მოიცავს ყველა მოქმედი პროექტის პროდუქტს, აგრეთვე ამ პროდუქტების შესაბამისობაზე სტანდარტებთან.

კლასიკური იტერაციული მოდელი

როგორც აღვნიშნეთ, პროგრამული უზრუნველყოფის სასიცოცხლო ციკლი პროგრამული უზრუნველყოფის კონსტრუირების პროცესია, რომლის მიზანს და რეგლამენტს განსაზღვრავს პროგრამული სისტემის ფუნქციური და არაფუნქციური მოთხოვნილებანი [55].

ჩვენ განვიხილეთ პროგრამული სისტემის სასიცოცხლო ციკლის ტრადიციულად მიღებული 7-ეტაპიანი მოდელი (ნახ.5.5).

ისმის კითხვა: გამოდგება კი ასეთი მოდელი ზოგადად ყველანაირი პროგრამული სისტემის აგების პროცესის მენეჯმენტისათვის: დაგეგმვის, აღრიცხვისა და კონტროლის ორგანიზებისათვის ?

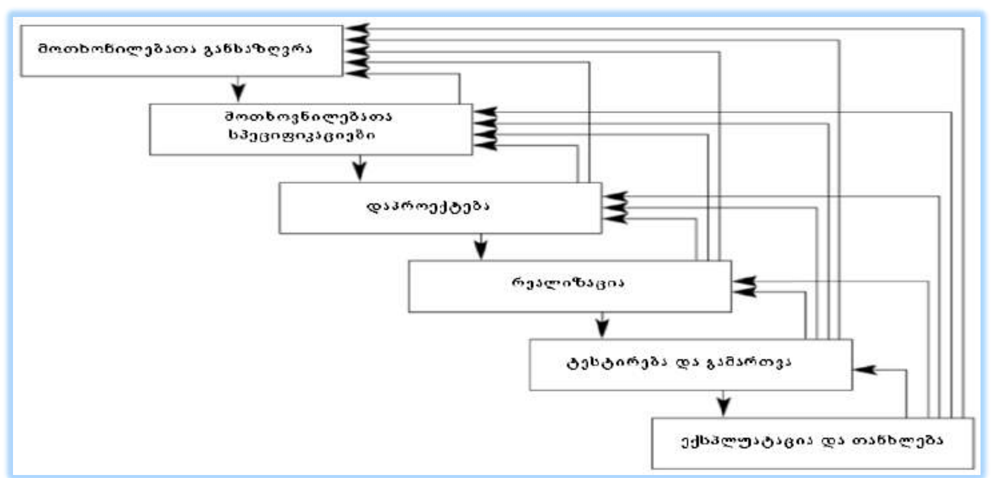
აღნიშნული მოდელის შეზღუდულობის გამო (მაგალითად, აქ თითოეულ ეტაპზე არ ჩანს ცხადად პროექტის შემსრულებელთა-როლების ფუნქციები) იგი არ შეიძლება ჩაითვალოს სრულყოფილად, თუმცა როგორც საწყისი (პირველადი) მოდელი მისაღებია. მისი გამოყენება უპრობლემოდ შეიძლება მარტივი პროექტების შესრულებისას, სადაც არაა საჭირო იტერაციული პროცესები.

იტერაცია – არის პროგრამული სისტემის შესრულების პროცესში წინა ბიჯებზე (ეტაპებზე) დაბრუნება, გარკვეული დასაკორექტირებელი პროცედურების ჩასატარებლად.

რთული პროექტები აუცილებლად მოითხოვს მრავალჯერადი იტერაციული პროცესების ჩატარებას პროგრამული სისტემის სასიცოცხლო ციკლის ყველა ეტაპზე, როგორც წინა ეტაპების შეცდომების გასწორების, განუსაზღვრელობის გათვალისწინების მიზნით, ისე სისეტემის ექსპლუატაციის პირობების მოთხოვნილებების ცვლილებების გამო. კლასიკური იტერაციული მოდელის სასიცოცხლო ციკლის ზოგადი სქემა მოცემულია 5.6 ნახაზზე.

ისრებით (ზემოთ) მითითებულია იტერაციული პროცესები წინა ეტაპებზე, რაც გამოწვეულია გარკვეული შეცდომებისა და გაურკვევლობების გასწორების მიზნით. ტრადიციული მეთოდები ცდილობს ასეთი პროცესების მინიმიზაციას, ანუ მკაცრად მოითხოვს, წინასწარ იყოს ყველაფერი ზუსტად განსაზღვრული და გამორიცხავს უკუიტერაციებს, რაც მათ ნაკლად შეიძლება ჩაითვალოს.

პროგრამული პროექტების იტერაციული განვითარების მეთოდები აქტუალურია, რადგან ისინი უარს ამბობს ეტაპების სრულფასოვნებაზე და არ გამორიცხავს მათ იტერაციულ განვითარებას, ფუნქციურობის და ინტერფეისული შესაძლებლობების გაფართოების თვალსაზრისით.



ნახ.5.6.კლასიკური იტერაციული მოდელი

ამგვარად, პროგრამული სისტემის სასიცოცხლო ციკლის კლასიკური მოდელი მისაღებია, ოღონდ ერთი იტერაციის ფარგლებში და მნიშვნელოვანი შესწორებით: რომ ყველაფერი, რაც სასარგებლო იყო ადრე, შეინახება.

ძველი კოდის შენახვის მხარდაჭერის იდეა, მისი ეფექტიანობის დაკარგვის გარეშე, დაკავშირებულია მთლიანად ობიექტორიენტირებულ დაპროგრამებასა და პროექტირებასთან, CASE-ინსტრუმენტების გამოყენებასთან (ამ საკითხს მომავალში დავუბრუნდებით).

იტერაციული გაფართოების მიზანია სისტემის მოქნილობის ამაღლება, მისი ადაპტაციის შესაძლებლობის უზრუნველყოფა პროგრამული სისტემისა და პროექტის პირობების ცვლილებების დროს. იტერაციული ბიჯების შემოტანა არის სწორედ პროექტის ადაპტურობის ამაღლების საშუალება, რადგანაც ასეთი მიდგომის გამოყენებისას პროექტი უფრო მომარჯვებულია ცვლილებებისადმი.

ასეთი კონცეფცია ეწინააღმდეგება პროგრამული სისტემების აგების ტრადიციულ მეთოდოლოგიებს, თუ პროექტების დამუშავების პროცესში და სისტემის არქიტექტურაში არ იქნება გათვალისწინებული ადაპტაციის მექანიზმები.

ობიეტორიენტირებულ მიდგომაში ეს მექანიზმები უფრო განვითარებულია, ამიტომ ის შეიძლება მიღებულ იქნას პროგრამული პროექტების განვითარების მთლიანი იდეოლოგიის საფუძვლად.

პროგრამული სისტემების ახალი მეთოდოლოგიები მთლიანობაში კი რჩება ობიექტ-ორიენტირებული, მაგრამ ცდილობენ ამ მიდგომის მკაცრი მოთხოვნებიდან გათავისუფლებას. მაგალითად, ექსტრემალური პროგრამირების მეთოდი დასაშვებად მიიჩნევს სისტემის პირველადი დეკომპოზიციის შედეგების (სისტემის დიზაინის) ცვლილებას, პროექტის შესრულების პროცესში მომხმარებელთა მოთხოვნილებების დაზუსტების შედეგად.

ასეთი თვალსაზრისი მოითხოვს პროგრამული პროექტების მენეჯმენტის ახალი, სპეციალური ვარიანტის შექმნას.

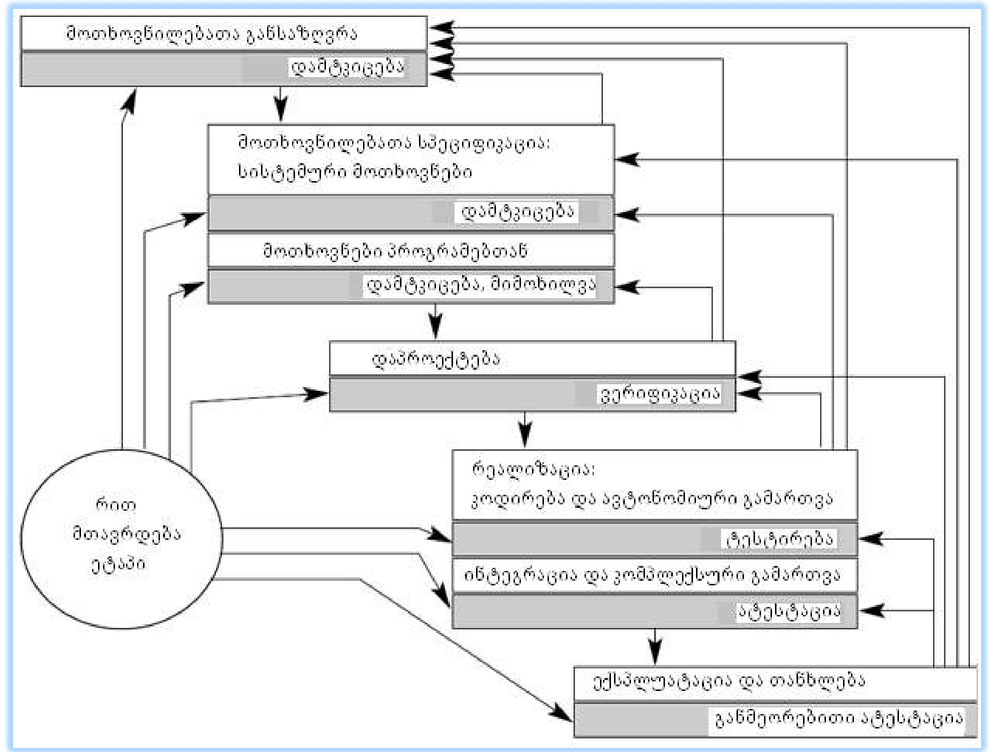
- **კასკადური მოდელი**

კლასიკური იტერაციული მოდელის შედარებით მკაცრი ნაირსახეობაა **კასკადური მოდელი,** რომელიც საილუსტრაციო მაგალითია იმისა, თუ როგორ შეიძლება უკანდაბრუნებების (იტერაციების) მინიმიზაცია.

კასკადური მოდელისათვის დამახასიათებელია:

- ყოველი ეტაპის დასრულება (როგორც კლასიკურ მოდელში) მიღებული შედეგების შემოწმებით, რათა აღმოფხვრილ იქნას პროგრამის დამუშავების რაც შეიძლება მეტი პრობლემა;
- გავლილი ეტაპების ციკლური გამეორება (როგორც კლასიკურ მოდელში).

კასკადური მოდელის მოტივაცია დაკავშირებულია პროგრამული უზრუნველყოფის ხარისხის მართვასთან. ამასთან დაკავშირებით ზუსტდება ეტაპების არსი, ზოგი მათგანი სტრუქტურირდება (მოთხოვნილებათა სპეციფიკაცია და რეალიზაცია). 5.7 ნახაზზე მოცემულია კასკადური მოდელის სქემა, რომელიც აგებულია როგორც კლასიკური იტერაციული მოდელის მოდიფიკაცია.



ნახ.5.7. კასკადური მოდელი

თითოეულ ბლოკში (ეტაპში) მითითებულია ქმედება, რომლითაც თავდება ეტაპი. აქ ტესტირება არაა გამოყოფილი ცალკე ეტაპად, იგი ითვლება ზღურბლად, რომელიც გადალახულ უნდა იქნას, რათა დასრულდეს ეტაპი, ისევე როგორც სხვა მსგავსი ქმედებები (მაგალითად, *მიმოხილვები* – დოკუმენტებია, რომლებშიც აღიწერება სისტემური მოთხოვნები და ისინი უნდა შეთანხმდეს და დამტკიცდეს დამკვეთის მიერ).

დაპროექტების შედეგი ვერიფიცირდება (მოწმდება), უზრუნველყოფს თუ არა სისტემის მიღებული სტრუქტურა და სარეალიზაციო მექანიზმები სპეციფიკური ფუნქციების შესრულებას. რეალიზაცია კონტროლდება კომპონენტების ტესტირების გზით, ხოლო კომპონენტების ინტეგრაციის შემდეგ სისტემაში ხორციელდება ატესტაცია კომპლექსური გამართვისათვის. ანუ ხდება სისტემის რეალიზებული ფუნქციების შემოწმება-ფიქსაცია, რეალიზაციის შეზღუდვების აღწერა და ა.შ.

კასკადურ მოდელში ვერიფიკაცია და ატესტაცია მიწერილია სხვადასხვა ეტაპთან:

- ვერიფიკაცია პასუხობს კითხვას – სწორადაა თუ არა აგებული პროგრამული სისტემა (ანუ იგი ამოწმებს სპეციფიკაციასთან შესაბამისობას და მას ატარებენ დამპროექტებლები და პროგრამისტები);
- ატესტაცია კი – თუ მუშაობს სწორად პროგრამული სისტემა (ის განიხილება სისტემის ექსპლუატაციის დროს და ვარგისიანობაზე დასკვნას ამზადებენ დამკვეთი სპეციალისტები-მომხმარებლები).

ექსპლუატაციის და თანხლების პროცესში დგინდება, თუ რამდენად კარგად შეესაბამება სისტემა მომხმარებელთა მოთხოვნებს, ანუ ხდება განმეორებითი ატესტაცია.

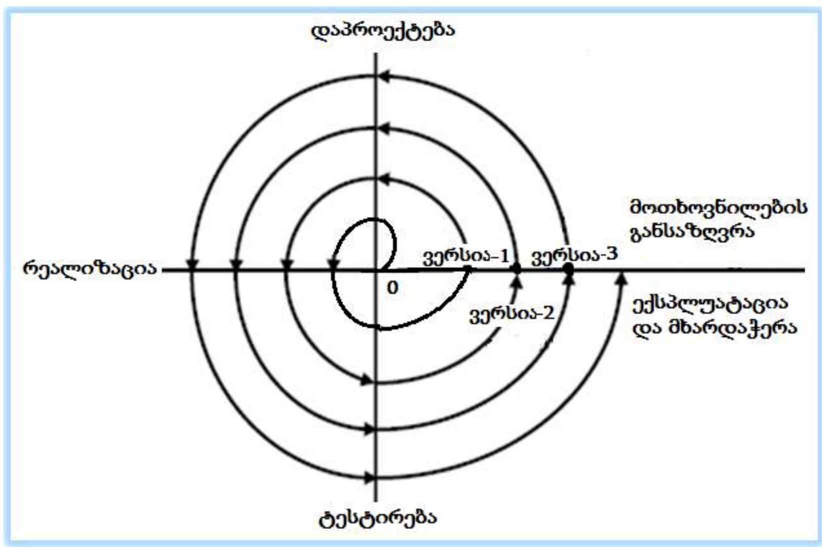
ყოველი შემოწმების შემდეგ შესაძლებელია დამპროექტებლების დაბრუნება ნებისმიერ გავლილ ეტაპზე, რაც უკუისრებითაა ნაჩვენები. უკან-დაბრუნების ბიჯების მინიმიზაციის მიზნით კასკადურ მოდელში დამპროექტებლები იყენებენ გამკაცრებულ შემოწმებას. (ლიტერატურაში ცნობილია ეს მკაცრი კასკადური მოდელების სახელწოდებით – აქ ჩვენ მას დეტალურად არ განვიხილავთ). აღვნიშნოთ მისი სასიცოცხლო ციკლის დამახასიათებელი მომენტები:

- სამუშაოების, დავალებებისა და პასუხისმგებლობათა ზუსტი განაწილება ეტაპების დამმუშავებელთა და მათ შემმოწმებელთა შორის, რომლებიც მომდევნო ეტაპზე გადასვლის თანხმობას იძლევიან;
- მცირე ციკლების არსებობა მეზობელ ეტაპებს შორის, რომელთა შედეგად მიიღწევა კომპრომისული დავალება.

პროგრამული სისტემების სასიცოცხლო ციკლების განხილული მეთოდების გამოყენება არაა უნივერსალური და დამოკიდებულია კონკრეტულ ობიექტზეც და ბევრ სხვა ფაქტორზეც.

- სპირალური მოდელი

პროგრამული სისტემების დამუშავების სასიცოცხლო ციკლის სპირალური მოდელის ძირითადი არსი მდგომარეობს მისი ეტაპების (ანალიზი, დაპროექტება, რეალიზაცია, ტესტირება და დანერგვა-ექსპლუატაცია) ციკლურ ევოლუციურ განვითარებაში (ნახ.5.8).



ნახ.5.8. სპირალური მოდელი

პროგრამული სისტემის ვერსიები (1,2,3,..) იქმნება პროტოტიპების სახით, იგი ვითარდება სპირალურად და ყოველი ახალი ვერსია სულ უფრო ახლოა დამკვეთი-მომხმარებლების მოთხოვნილებებთან.

ტესტირებისა და საექსპლუატაციო დანერგვის პროცესში მონაწილეობენ სისტემური ანალიტიკოსები და დამკვეთები, რაც საშუალებას იძლევა ოპერატიულად განისაზღვროს საჭირო ცვლილებები შემდეგი ვერსიისათვის.

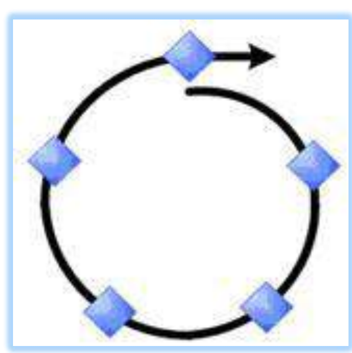
- MSF მოდელი

პროგრამული სისტემების აგების Microsoft Solution Framework (MSF) მეთოდოლოგია (ნახ.5.9) აერთიანებს კასკადური და სპირალური მოდელების მოწესრიგებადობისა და მოქნილობის თვისებებს [6].

საბაზო პრინციპები შემდეგია:

1. პროექტის ერთიანი ხედვა. თავიდან დამკვეთსაც და საპროექტო გუნდსაც ექნებათ თავიანთი საკუთარი ხედვა პროექტის მიზნებზე და ამოცანებზე. აგრეთვე რა უნდა იქნას მიღწეული პროექტზე მუშაობის პროცესში;

პროექტის წარმატება შეუძლებელია დამკვეთისა და საპროექტო გუნდის ერთიანი ხედვის გარეშე, რადგანაც გაურკვევლობის გამო მიზანი მიუღწეველი რჩება.



ნახ.5.9

MSF მეთოდოლოგია ამისათვის ითვალისწინებს ცალკე ეტაპის (ფაზის) გამოყოფას - „კონცეფციის შემუშავება“.

2. მოქნილობის გამოვლენა - მზადყოფნა ცვლილებებისადმი. კასკადური მოდელისაგან განსხვავებით MSF მიდგომა ეფუძნება პროექტის პირობების უწყვეტი ცვლილების პრინციპს.

3. კონცენტრირება ბიზნესპრიორიტეტებზე. დასამუშავებელმა პროდუქტმა უნდა მოიტანოს განსაზღვრული სარგებელი ან უკუგება საბოლოო მომხმარებლებისთვის. ვინაიდან პროგრამულ პროდუქტს შეუძლია თავისი სარგებლობის მოტანა მხოლოდ დანერგვის შემდეგ ორგანიზაციაში, ამიტომ MSF მიდგომა სასიცოცხლო ციკლში ითვალისწინებს დანერგვის ფაზას.

4. თავისუფალი ურთიერთობის წახალისება. MSF-ის პროცესების მოდელი გვთავაზობს ინფორმაციის ღია და თავისუფალ გაცვლას როგორც პროექტის შემსრულებლებს, ისე პროექტით დაინტერესებულ პირებს (stakeholders) შორის. ამან ხელი უნდა შეუწყოს მათ შორის გაუგებრობის, გაურკვევლობისა და უსაფუძვლო ხარჯების შემცირებას. ამიტომაც MSF ითვალისწინებს მუშაობის პროცესის სისტემატურ ანალიზს დადგენილი დროის პუნქტებში, რომლებშიც მონაწილეობენ დამკვეთებიც და შემსრულებლებიც.

5.3. განტერის მოდელი: „ფაზები და ფუნქციები“

პროგრამული პროდუქტების სასიცოცხლო ციკლის მოდელებში აისახება საწარმოო ფუნქციები, რომლებსაც ასრულებენ დამპროექტებლები. ეს ფუნქციები უნდა იყოს კავშირში პროექტების მართვის საკონტროლო ელემენტებთან, ანუ სასიცოცხლო ციკლის ეტაპებთან. ისინი სრულდება პროექტის განვითარების მთელი პერიოდის მანძილზე სხვადასხვა ინტენსივობით. განტერის მოდელი „ფაზები და ფუნქციები“ საფუძველია განვითარებული სასიცოცხლო ციკლის ასაგებად, სადაც ასახულია

ორგანიზაციული და ტექნიკური საწარმოო ფუნქციები [55,75,111]. ამასთანავე განტერის მოდელი ითვალისწინებს იტერაციებსაც.

მოდელი უნდა იყოს პროექტის დამმუშავებლებს შორის ურთიერთდამოკიდებულების ორგანიზაციის საფუძველი. ამგვარად, მისი ერთ-ერთი მიზანი მენეჯერის ფუნქციების მხარდაჭერაა. ეს კი აუცილებლად მოითხოვს პროექტზე საკონტროლო წერტილების დალაგებას, რომელიც ასახავს პროექტის ორგანიზაციულ-დროით ჩარჩოებს, ორგანიზაციულ-ტექნიკურ საწარმოო ფუნქციებს, რომლებიც სრულდება პროექტის განვითარების დროს.

განტერის მოდელს აქვს ორი განზომილება:

- ფაზური, რომელიც ასახავს შესრულების ეტაპებს და თანმხლებ მოვლენებს;
- ფუნქციური, სადაც ჩანს, თუ რომელი საწარმოო ფუნქციები სრულდება პროექტის განვითარების პროცესში და როგორია მათი ინტენსივობა თითოეულ ეტაპზე.

5.10 ნახაზზე მოცემულია განტერის მოდელის სქემა. აბსცისთა ღერძი პროექტის განვითარებას ასახავს, იგი დროის ღერძია. მასზე დასმულია საკონტროლო წერტილები და მოვლენათა დასახელებები.

მოდელში სასიცოცხლო ციკლი მოიცავს შემდეგ ფაზებს (ეტაპებს):

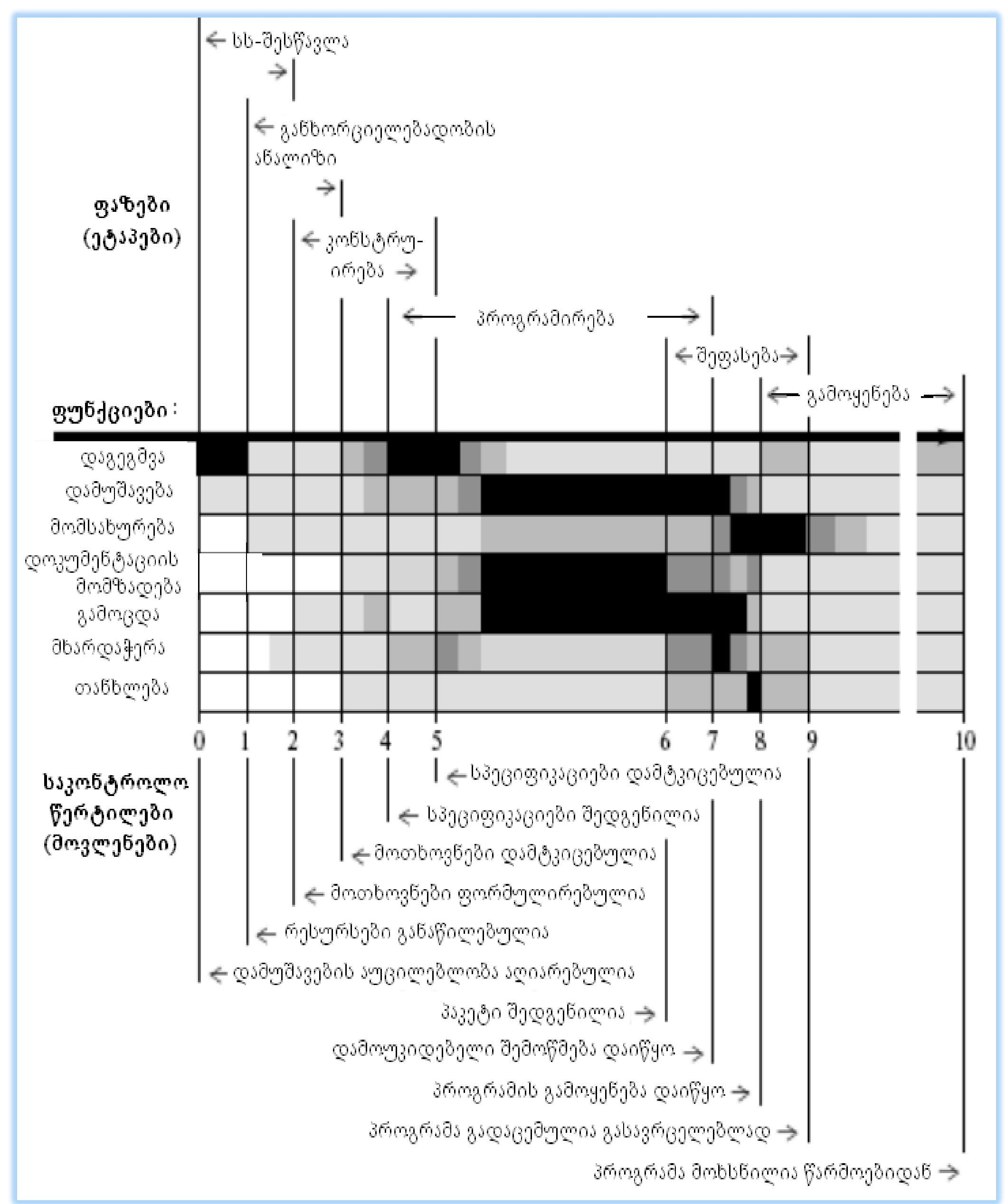
- გამოკვლევის ეტაპი;
- განხორციელებადობის ანალიზი;
- კონსტრუირება;
- დაპროგარმება;
- შეფასება;
- გამოყენება.

სასიცოცხლო ციკლის მოდელის საწარმოო-ორგანიზაციული ფუნქციები შემდეგია:

- დაგეგმვა;
- დამუშავება;
- მომსახურება;
- დოკუმენტაციის მომზადება;
- გამოცდა (შეფასება);
- მხარდაჭერა;
- თანხლება.

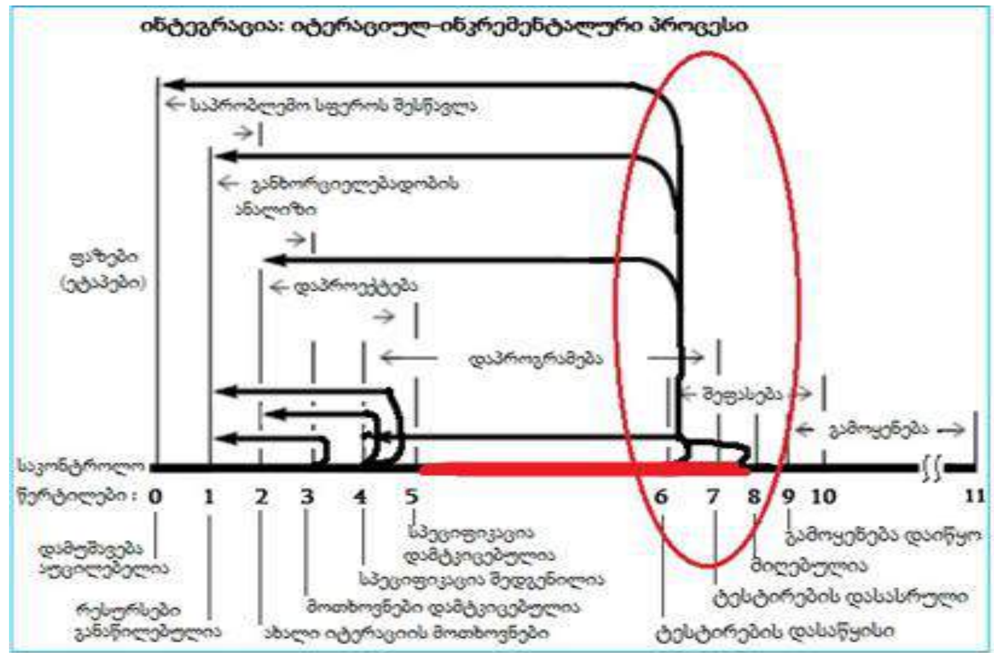
ფუნქციების გამოყენების ინტენსივობა ფაზების მიხედვით მოცემულია მუქი ფერით.

განტერის მოდელი არ ასახავს იტერაციულობას, დაბრუნებას წინა ეტაპებზე, რაც ერთგვარად არასახარბიელოდ ითვლება, თუმცა ტრადიციულ და კასკადურ მოდელებში ეტაპების გარკვეული გადაფარვა ხორციელდება.



ნახ.5.10. განტერის მოდელი „ფაზები-ფუნქციები“

იმისათვის, რომ იტერაციულობა იყოს გათვალისწინებული, საჭიროა მცირე მოდიფიკაცია, რაც 5.11 ნახაზზეა ნაჩვები. აქ სასიცოცხლო ციკლის საკონტროლო წერტილებში (მაგალითად, 3,4,6) ნაჩვენებია იტერაციული ციკლების შესაძლებლობები.



ნახ.5.11. იტერაციულობა განტერის მოდელში
„ფაზები-ფუნქციები“

განიხილება პროექტების შესრულების ორი შემთხვევა:

- რომელიმე საკონტროლო წერტილში მოხდება ძირითადი პროცესის შეჩერება, დაბრუნება რომელიმე წინა ეტაპზე გარკვეული დროის განმავლობაში. შემდეგ ისევ გადმოსვლა ძირითად პროცესზე და შედეგების შერწყმა ცვლილებების გათვალისწინებით;
- ძირითადი პროცესი არ წყდება იტერაციული ციკლის შესრულების მინით. იგი გრძელდება ჩვეულებრივად, ხოლო იტერაცია განიხილება როგორც დამატებით პროცესი, რომელიც უნდა მიუერთდეს სასიცოცხლო ციკლს. ასეთი დამატებითი პროცესი ორგანიზებულად უნდა წარიმართოს, ანუ შესაძლებელია მისი წინასწარი დაგეგმვა.

იტერაციულობა გარდაუვალია რთული პროგრამული სისტემების დასაპროექტებლად. ამიტომაც მიზანშეწონილია ასეთი პროცესების დაგეგმვა. ისინი უნდა იქნას აღქმული არა როგორც „შეცდომების“ გასწორების იტერაციული პროცედურები, არამედ როგორც სისტემის გაფართოების აუცილებელი იტერაციები.

ეს საკითხი განსაკუთრებით საყურადღებოა ობიექტ-ორიენტირებული დაპროგრამების პრინციპების გამოყენებისას.

5.4. ობიექტორიენტირებული დაპროგრამების პრინციპები

პროგრამული პროექტების შერულების პრინციპული მომენტები, რომლითაც ობიექტორიენტირებული მიდგომა განსხვავდება ტრადიციული მიმდევრობითი მეთოდოლოგიებისაგან, შემდეგია [55]:

1. განვითარების იტერაციულობა:

დაწყებული ანალიზის ფაზიდან, დამთავრებული რეალიზაციით, ობიექტ-ორიენტირებული პროექტირების პროცესი, მიმდევრობითი განვითარების მიდგომისგან განსხვავებით, აიგება როგორც იტერაციათა სერია, რომელსაც შესაძლოა წინ უძღოდეს საგნობრივი სფეროს და საპროექტო ამოცანათა მიმდევრობითი შესწავლის პერიოდი (მოთხოვნილებათა განსაზღვრისა და საწყისი დაგეგმვის ეტაპები).

2. ფუნქციონალობის ცვლილებები:

პროექტის შესრულების პროცესში მისი მოთხოვნები პრაქტიკულად ყოველთვის გადაისინჯება. ამასთან მოითხოვება, რომ შეცვლილ იყოს ადრე მიღებული მოთხოვნები. ამგვარად, ფუნქციონალობა ისე უნდა იყოს რეალიზებული, რომ მისი გადასინჯვა მოხდეს მინიმალური დანახარჯებით.

3. პროექტის ცნებათა სისტემის ფორმირება:

პროექტის ცნებათა სისტემა ვითარდება მოთხოვნილებათა გადასინჯვის პროცესში და ფუნქციონალობის ცვლილებისას, აგრეთვე ცვლილებათა გავრცელებისას. სისტემის მთლიანობის უზრუნველსაყოფად საჭიროა შესაბამისი ღონისძიებები, სპეციალური ინსტრუმენტების გამოყენება.

ეს ამოცანა წყდება პროექტის ლექსიკონის (გლოსარიის) დახმარებით. ესაა სპეციალური ცოდნის ბაზა ცნებების, მათი კავშირებისა და ცვლილებათა ისტორიისა პროექტის იტერაციული განვითარების პროცესში. თუ ლექსიკონი ცხადად არაა გათვალისწინებული, მაშინ პროექტის ცნებათა ლექსიკონი სტიქიურად იქმნება, რაც უარყოფითად აისახება პროექტის განვითარებაზე.

კერძოდ, იზრდება კონცეფციის ცვლილების რისკი პროგრამული კოდის ცვლილებების შედეგად, რასაც მივყავართ კონცეპტუალურად შეუთავსებელი პროგრამული პროდუქტის აგებამდე.

4. ფუნქციონალობის გაფართოება სცენარების შესაბამისად:

გამოყენებით შემთხვევათა (Use Case) დიაგრამის აგება იმის დასადგენად, თუ როგორ კავშირშია მომხმარებელი სისტემასთან, არის პირველი დონე იმ მოდელებისა, რომელთაც ობიექტ-ორიენტირებული დაპროექტება გვთავაზობს.

ეს მოდელები აფიქსირებს სიტუაციებს, რომლებიც მომხამარებლისა და სისტემის ურთიერთქმედებისას აღმოცენდება. სისტემის შინაარსობრივი ქმედებები

მომხმარებლებთან აღიწერება სცენარების სახით. გამოყენებითი შემთხვევები და სცენარები საფუძველია სისტემის მოთხოვნილებათა იმ სახით წარმოსადგენად, რომელსაც გამოიყენებს დამპროექტებელი სისტემის არქიტექტურის ასაგებად. ფუნქციების გაფართოება ნიშნავს მისი სცენარების შემდგომ განვითარებას დაპროექტების ეტაპზე. სრული ფუნქციონალობა შედგება ყველა სცენარის ფუნქციონალობისგან. ამასთანავე, იტერაციული გაფართოება მოითხოვს, რომ იტერაციის ყოველ ბიჯზე პროგრამულ სისტემას ჰქონდეს სრულიად მზა ფუნქციონალობა, ხოლო მომდევნო ბიჯებზე ემატებოდეს მას სხვა, ახალი ფუნქციონალობა.

5. არაფერი ერთჯერადად არ კეთდება:

პროგრამირების კლასიკური მიმდევრობითი მიდგომა ითვალისწინებს ანალიზის, შემდეგ კონსტრუირების და, ბოლოს პროგრამირების ეტაპებს. განტერის მოდელში ეს ეტაპები იკვეთება, მაგრამ პრინციპულად სიტუაციას ვერ ცვლის.

ობიექტორიენტირებულ პროექტებში ანალიზი არასდროს თავდება სისტემის განვითარების მთელი პერიოდის მანძილზე, ხოლო კონსტრუირების პროცესი თანმხლებია პროგრამული პაკეტის მთელი სასიცოცხლო ციკლისა.

6. ოპერირება გამრავლების ფაზებზე მსგავსია:

როგორც პროექტირების დასაწყისში, ისე მომდევნო იტერაციებზე ანალიზი წინ უსწრებს კონსტრუირებას, რომელსაც მოსდევს დაპროგრამება, ტესტირება და პროგრამული უზრუნველყოფის დამუშავებისა და გამოყენების ტრადიციული სასიცოცხლო ციკლის სხვა სახის სამუშაოები.

ობიექტორიენტირებული დაპროექტების დროს იტერაციული გაფართოების პროცესში, ჩვეულებისამებრ, სრულდება ტრადიციული ეტაპები:

- *მოთხოვნილებათა განსაზღვრა,* ან იტერაციათა დაგეგმვა ფიქსირდება, თუ რა უნდა იყოს შესრულებული მოცემულ იტერაციაზე სფეროს აღწერის სახით, რომლისთვისაც იგეგმება ფუნქციონალობის დამუშავება, და რა არის ამისთვის საჭირო. ამ ეტაპზე შეირჩევა ის სცენარები, რომლებიც უნდა იყოს რეალიზებული მოცემულ იტერაციაზე;
- *ანალიზი* – ხდება დაგეგმილ მოთხოვნილებათა შესრულების პირობების გამოკვლევა, მოწმდება შერჩეული სცენარების სისრულე მოთხოვნილი ფუნქციონალობის რეალიზაციის თვალსაზრისით;
- *მოდელირება მომხმარებლის ინტერფეისის* – რადგანაც იტერაცია უნდა უზრუნველყოფდეს ფუნქციონალურად დასრულებულ რეალიზაციას, საჭიროა განისაზღვროს ურთიერთმოქმედების წესები, რომლებიც აუცილებელია წარმოდგენილი ფუნქციების გააქტიურებისათვის. ინტერფეისის მოდელი ასახავს მომხმარებლის წარმოდგენას მოცემული იტერაციის ობიექტების ყოფაქცევის შესახებ;

- *კონსტრუირება* პროექტის დეკომპოზიციაა, განხორციელებული ობიექტ-ორიენტირებულ სტილში. იგი მოიცავს კლასთა სისტემის იერარქიის აგებას ან გაფართოებას, მოვლენათა აღწერას და მათზე რეაქციის განსაზღვრას. კონსტრუირების დროს განისაზღვრება ობიექტები, რომლებიც რეალიზდება ან ფართოვდება მოცემულ იტერაციაზე, ასევე ფუნქციათა ერთობლიობა (ობიექტთა მეთოდები), რომელიც უზრუნველყოფს მოცემული იტერაციის ამოცანის გადაწყვეტას;
- *რეალიზაცია (დაპროგრამება)* – პროგრამული განხორციელება გადაწყვეტილებისა, რომელიც მიღებულ იქნა მოცემულ იტერაციაზე. რეალიზაციის აუცილებელ კომპონენტად აქ მოიაზრება შემადგენელი მოდულების თავსებადობის ავტონომიური შემოწმება მათ სპეციფიკაციებთან (კერძოდ, უზრუნველყოფილ უნდა იქნეს ობიექტების საჭირო ყოფაქცევა);
- *ტესტირება* კომპლექსური შემოწმებაა შედეგებისა, რომლებიც მიიღება მოცემულ იტერაციაზე. ზოგჯერ, როგორც ტრადიციულ სქემებში, ტესტირების ეტაპს აერთიანებენ სასიცოცხლო ციკლის შემდეგ ეტაპთან;
- იტერაციის შედეგების შეფასება ხდება მიღებული შედეგების ანალიზი მთლიანი პროექტის ჭრილში. კერძოდ გაირკვევა, თუ პროექტის რომელი ამოცანების გადაწყვეტაა შესაძლებელი მოცემული იტერაციის შედეგებით, რომელ ადრე დასმულ კითხვებზე იქნა პასუხი გაცემული, რა სახის ახალი კითხვები გაჩნდა ახალ პირობებში.

- **განტერის „ფაზა-ფუნქციები“-მოდელის მოდიფიკაცია**

ობიექტორიენტირებული დაპროექტების დროს სასიცოცხლო ციკლის ფაზური განზომილება თითქმის არ იცვლება, მხოლოდ ემატება ერთი ეტაპი –მომხმარებლის ინტერფეისის მოდელირება. ეს საკითხი ძველ მოდელებში განიხილებოდა ანალიზისა და კონსტრუირების ეტაპებზე.

საჭიროა აღინიშნოს, რომ ეს მეტად მნიშვნელოვანი დამატებაა ობიექტ-ორიენტირებული მოდგომით სასიცოცხლო ციკლის მოდელირებისათვის. შემდეგი ორი პრინციპი დაკავშირებულია ამასთან.

7. რელიზებულ მოთხოვნილებათა განაწილება იტერაციების მიხედვით:

სცენარების ერთობლიობა, რეალიზებული წინა და მომდევნო იტერაციებზე, ყოველთვის ქმნის დასრულებულ, მაგრამ არასრული ვერსიის სისტემას, რომელიც გადაეცემა მომხმარებლებს. სხვადასხვა მიზეზის გამო, მათ შორის ორაზროვნების გამორიცხვის მიზნით, საჭიროა სარეალიზაციოდ მიღებული დასაგეგმი საშუალებების წარმოდგენა მოდელების სახით, რომლებშიც შეთანხმებულია მომხმარებელთა (დამკვეთის) შეხედულებანი სისტემაზე, დამპროექტებელთა თვალსაზრისით. ეს მოდელები ჩნდება ანალიზის ეტაპზე, ამიტომაც მათ უწოდებენ *ანალიზის დონის მოდელებს*;

8. სისტემის შესაძლებლობათა გაფართოების განსაკუთრებული სტილი და მისი განვითარება:

ობიექტორიენტირებული მიდგომის დროს სისტემა წარმოიდგინება როგორც სხვადასხვა დამოკიდებულებებით დაკავშირებულ კლასთა ერთობლიობა, რაც პროექტის დეკომპოზიციის საფუძველია. ყოველი ახალი იტერაცია აფართოებს ამ ერთობლიობას ახალი კლასების დამატებით, რომლებიც განსაზღვრულ დამოკიდებულებებში შედის არსებულ კლასებთან. ასეთი გაფართოების კორექტული შესრულება, არსებულისა და პერსპექტივის დეტალების აბსტრაჰირების გათვალისწინების გარეშე, შეუძლებელია. ანუ, აუცილებელია *კონსტრუირების დონის მოდელების* აგება, რომლებიც ასახავს საპროექტო სისტემის სარეალიზაციო სახეს.

გარდა ზემოაღნიშნული ანალიზისა და კონსტრუირების დონეთა მოდელირებისა, არსებობს მესამე ასპექტი მოდელირებისა, რომელიც დაკავშირებულია მომხმარებელთან პროგრამული სისტემის ყოველი ვერსიის წარდგენასთან. თუ განტერის მოდელის სტილს გამოვიყენებთ სასიცოცხლო ციკლის აღსაწერად, მაშინ სწორი იქნება არა მოდელირების ეტაპის გამოყოფა (როგორც ტრადიციულადაა მიღებული), არამედ ორგანიზაციულ-ტექნიკური (საწარმოო) ფუნქციის მოდელირებისა, რომელიც გამჭოლად განიხილავს პროექტის დამუშავების მთელ პროცესს.

ნახაზებზე, შესაბამისად, მოცემულია განტერის სასიცოცხლო ციკლის მოდელი (5.10) და მისი მოდიფიკაცია ობიექტორიენტირებული მიდგომისათვის (5.12).

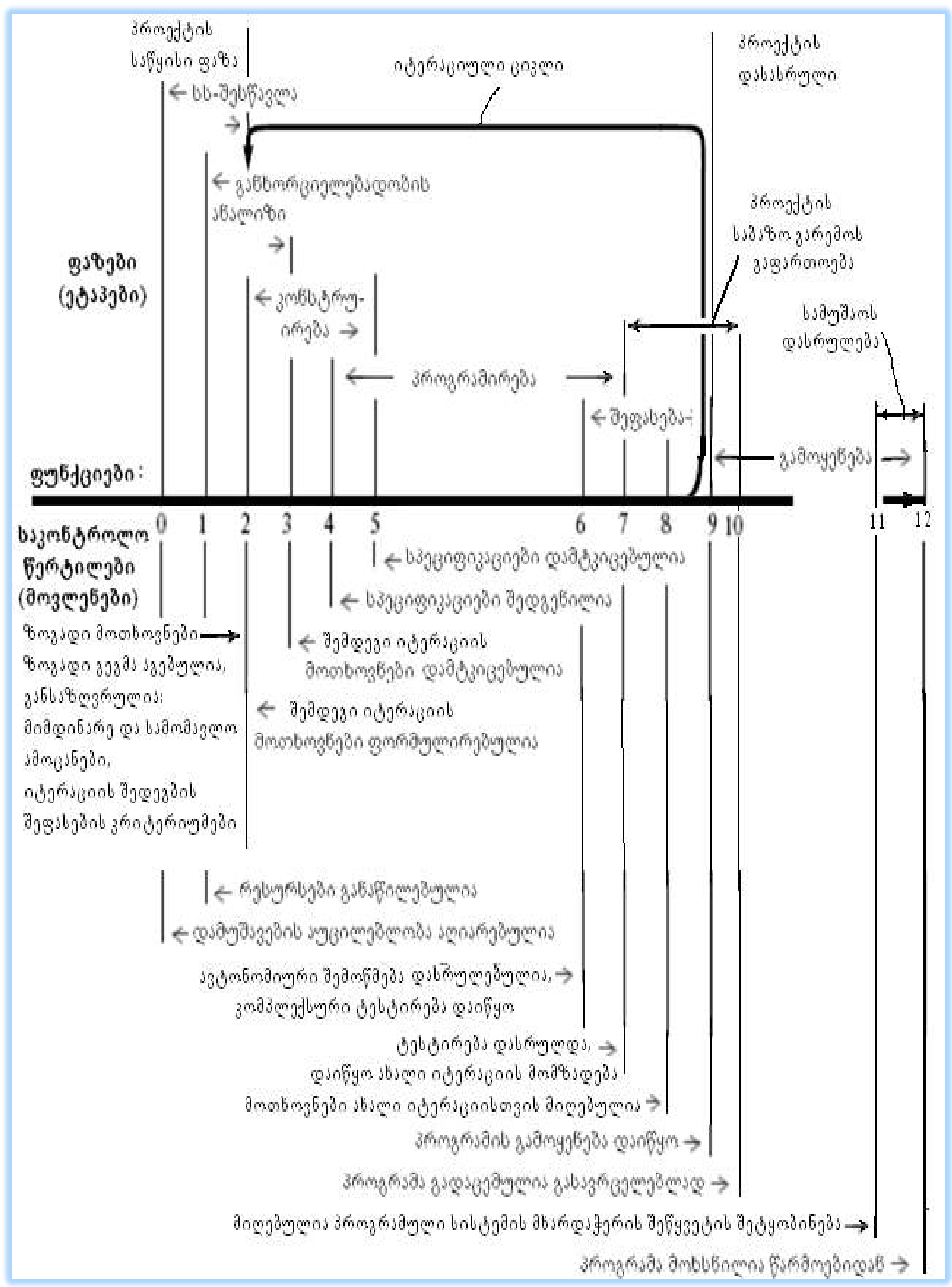
ოო-მიდგომის დროს იტერაცია გულისხმობს არა წინა ბიჯების შეცდომების გასწორებას, არამედ დაგეგმილ აქტს – შესაძლებლობების გაფართოებას.

აქვე შეფასების ეტაპზე გამოყოფილია სპეციალური ჩადგმული ეტაპი – პროექტის საბაზო გარემოს გაფართოება, ანუ იგულისხმება პროგრამული უზრუნველყოფის ხელმეორე გამოყენების დაგეგმვა და რეალიზაცია (პროტოტიპების გამოყენება). სასურველია შეფასების ეტაპზე ხდებოდეს ასეთი სასარგებლო პროექტების შენახვა (საცავებში).

განტერის მოდელი არ ასახავს იტერაციულობას, დაბრუნებას წინა ეტაპებზე, რაც ერთგვარად არასახარბიელოდ ითვლება, თუმცა ტრადიციულ და კასკადურ მოდელებში ეტაპების გარკვეული გადაფარვა ხორციელდება.

იმისათვის, რომ იტერაციულობა იყოს გათვალისწინებული, საჭიროა მცირე მოდიფიკაცია, რაც 5.10 ნახაზზეა ნაჩვები. აქ სასიცოცხლო ციკლის საკონტროლო წერტილებში (მაგალითად, 3,4,6) ნაჩვენებია იტერაციული ციკლების შესაძლებლობები.

- რომელიმე საკონტროლო წერტილში მოხდება ძირითადი პროცესის შეჩერება, დაბრუნება რომელიმე წინა ეტაპზე გარკვეული დროის განმავლობაში. შემდეგ ისევ გადმოსვლა ძირითად პროცესზე და შედეგების შერწყმა ცვლილებების გათვალისწინებით;



ნახ.5.12. განტერის მოდელი იტერაციით ობიექტორიენტირებული
პროგრამული პროექტისთვის

- იტერაციული ციკლის შესრულების მიზნით ძირითადი პროცესი არ წყდება. იგი გრძელდება ჩვეულებრივად, ხოლო იტერაცია განიხილება როგორც დამატებით პროცესი, რომელიც უნდა მიუერთდეს სასიცოცხლო ციკლს. ასეთი დამატებითი პროცესი ორგანიზებულად უნდა წარიმართოს, ანუ შესაძლებელია მისი წინასწარი დაგეგმვა.

განიხილება პროექტების შესრულების ორი შემთხვევა:

იტერაციულობა გარდაუვალია რთული პროგრამული სისტემების დასაპროექ-ტებლად. ამიტომაც მიზანშეწონილია ასეთი პროცესების დაგეგმვა. ისინი უნდა იქნას აღქმული არა როგორც „შეცდომების" გასწორების იტერაციული პროცედურები, არამედ როგორც სისტემის გაფართოების აუცილებელი იტერაციები. ეს საკითხი განსაკუთრებით საყურადღებოა ობიექტორიენტირებული დაპროგრამების პრინციპების გამოყენებისას.

ახლა განვიხილოთ 5.12 ნახაზზე მოცემული *12 საკონტროლო წერტილის* (სწ) არსი [55].

0. დამუშავების აუცილებლობა აღიარებულია: ეს საკონტროლო წერტილი (სწ-0) მიუთითებს პროექტის სამუშაოების დაწყებაზე, ოღონდ თვით პროექტი ჯერ ოფიციალურად არაა დამტკიცებული. იგი ხდება ოფიციალური მას მერე, როცა დამტკიცდება მისი განხორციელებადობა, განისაზღვრება პროექტის მთლიანი ამოცანები, მათ შორის პირველ რიგში გადასაწყვეტ ამოცანათა ერთობლიობა (სწ-2). ამ ეტაპზე მენეჯერი ახორციელებს წინასაპროექტო საქმიანობას (შესაძლოა გუნდთან ერთად). მას გამოაქვს გადაწყვეტილება პროექტის განხორციელების მიზანშეწონილობის შესახებ. ამ ეტაპზე მნიშვნელოვანი საკითხია საპროექტო გუნდის ფორმირების სტრატეგია და ტაქტიკა.

1. რესურსები განაწილებულია: საპროექტო სამუშაოების დასაწყებად აუცილებელია იმის ცოდნა, თუ რა რესურსები გვექნება. მენეჯერი იღებს საჭირო ინფორმაციას დამკვეთი ორგანიზაციის მგეგმავისა და ინვესტორისაგან. ამის საფუძველზე იგი განსაზღვრავს პროექტის განვითარების კონცეფციას, რომელიც მოტივირებულია დამკვეთის მოთხოვნილებებით.

2. მოხოვნილებანი შემდეგი იტერაციისათვის განსაზღვრულია, ზოგადი მოთხოვნები და ზოგადი გეგმა შედგენილია, უახლესი ამოცანა, შედეგების შეფასების კრიტერიუმები და პერსპექტიული ამოცანები განსაზღვრულია: ეს სწ-2 პროექტის განვითარების სტაციონალრი ციკლის დასაწყისია, რომელზეც იგეგმება მიმდინარე იტერაციის მსვლელობა არსებული ინფორმაციის საფუძველზე. პირველ და მომდევნო იტერაციებზე აქ უნდა განისაზღვროს:

- **ზოგადი მოთხოვნილებანი** ანუ რა მოითხოვება პროექტიდან მთლიანად ამ მომენტში;
- **საერთო გეგმა** – როგორ მიიღწევა დასმული მიზნები;

- **უახლესი ამოცანა** – მიმდინარე იტერაციის ამოცანა (ცხადად ჩამოყალიბებული).

იტერაციის შედეგების შეფასება ხდება *კრიტერიუმებით.* მათი შედგენილობა დამოკიდებულია როგორც პროექტის სპეციფიკაზე, ისე მისი შესრულების მდგომარეობაზე. ტიპური კრიტერიუმებია:

– *აქტუალობა მომხმარებლისთვის*: ბიზნესის თალსაზრისით მნიშვნელოვანია, რომ პროგრამულ სისტემაში აისახოს ის ფუნქციურობა, რომელიც შეძლებს ავტომატიზებული საქმიანობის „ვიწრო ადგილების" აღმოფხვრას;

– *შემოთავაზებული საშუალებების სისრულე და ფუნქციური ჩაკეტილობა*: იგულისხმება მომხმარებელთა საქმიანობის რომელიმე სახის სრული ავტომატიზაცია, რომელიც არ მოითხოვს დამატებით რაიმე სხვა საშუალებებს;

– *სისტემური მნიშვნელობა*: პირველ რიგში მიზანშეწონილია იმ ფუნქციურობის რეალიზაცია, რომელიც სასარგებლოა პროექტის შემდგომი განვითარებისთვის, ანუ იმ კომპონენტებისა და საშუალებების დამუშავება, რომლებიც გამოსადეგი იქნება პროექტის მომდევნო იტერაციებზე;

– სადემონსტრაციო მნიშვნელობა: ამ კრიტერიუმის თვალსაზრისით, პირველ რიგში, გამოიყოფა ის საშუალებები, რომელთაც შეუძლია სისტემის შრომის-უნარიანობისა და მისი საუკეთესო თვისებების დემონსტრირება, რაც სასარგებლოა სისტემის ხარისხის საჩვენებლად. ამ კრიტერიუმს დიდი მნიშვნელობა აქვს პროექტის საწყისი ეტაპის პერიოდში, როცა მიღებული შედეგების საფუძველზე დამკვეთიც და შემსრულებლებიც დარწმუნდებიან პროექტის შესრულების სტრატეგიისა და მეთოდების სისწორეში. ეს კრიტერიუმი შეიძლება იყოს საწინააღმდეგო სხვა კრიტერიუმებთან – მაგალითად, აქტუალობის, ფუნქციურობის, სისტემის მნიშვნელობის;

– *რეალიზაციის სისწრაფე:* გამოიყოფა ის სამუშაოები, რომლებიც შეიძლება შესრულდეს ყველაზე მოკლე ვადებში. თუ იტერაციის დრო ფიქსირებულია, მაშინ გამოიყოფა სამუშაოთა ის მოცულობა, რომელსაც გუნდი განახორციელებს ამ დროში. დრო, ამ თვალსაზრისით განიხილება არა როგორც კრიტერიუმი, არამედ როგორც შეზღუდვა.

– *სისრულის კრიტერიუმი.* განასხვავებენ სისრულის სამ ასპექტს, როლებიც ასახავს პროგრამული სისტემის ხარისხის სამ შემადგენელს:

- ფუნქციური სისრულე – შეთავაზებული საშუალებების შესაბამისობა მომხმარებელთა ფუნქციების ერთობლიობასთან, რომლებიც შეიძლება შესრულდეს ავტომატიზებულ სისტემაში;
- რეალიზაციის სისრულე – შესაძლებლობა ფუნქციური სისრულის უზრუნველსაყოფად, სისტემის საბაზო საშუალებების კომბინაციის გზით, სარეალიზაციო მექანიზმებითა და სამუშაო გარემოს საშუალებებით;

- საინტერფეისო სისრულე – სისტემის ყოფაქცევის მართვისათვის ისეთი ენის შერჩევა, რომელიც მომხმარებელს ხელს შეუწყობს სისტემის ადეკვატურად აღსაქმელად.

შეიძლება აღინიშნოს, რომ რეალიზაციისა და საინტერფეისო სისრულეები წარმოებულია ფუნქციური სისრულიდან.

ზოგჯერ მოცემული იტერაციისთვის განიხილავენ იმ მინიმალური რეალიზაციის შესაძლებლობას, რაც უზრუნველყოფს სისრულეს და ფუნქციურ ჩაკეტილობას. ეს განსაკუთრებით ხაზგასმულია **ექსტრემალური დაპროგრამების** მიდგომის დროს.

პირველი იტერაციის ამოცანის გადაწყვეტისას კრიტერიუმები მოწესრიგდება შემდეგი მიმდევრობით: სადემონსტრაციო მნიშვნელობა, სამომხმარებლო აქტუალობა, რეალიზაციის სისწრაფე, სისტემური მნიშვნელობა და ფუნქციონალური სისრულე. **მენეჯერის** ხელოვნება მდგომარეობს იმაში, რომ არსებული შეზღუდვების პირობებში დაიცვას უპირატეს კრიტერიუმთა ბალანსი და უზრუნველყოს მეთოდების, მეტრიკების, სტრატეგიებისა და პროგნოზების შემოწმებისა და კორექტირების შესაძლებლობა.

პირველი რიგის ამოცანისათვის უნდა განისაზღვროს:

- რეალიზაციის გეგმები – დაყოფა ეტაპებად, შესრულების ვადები და რესურსული მოთხოვნები;
- შედეგების შეფასების კრიტერიუმები – მეთოდიკები, რომლებითაც დადგინდება გადაწყვეტილი ამოცანის მოთხოვნილებების შეფასება ვადებისა და ხარისხის მიხედვით;
- პერსპექტიული ამოცანები – მათი დაფიქსირება აუცილებელია მკაცრი ანგარიშგების პირობებში, ის ყოველთვის მოწმდება მე-2 საკონტროლო წერტილში. პერსპექტიული ამოცანების მოთხოვნილებათა შემუშავების ხარისხი შემდგომ იტერაციებზე შეიძლება იყოს განსხვავებული.

ბოლო პუნქტი, მაგალითად, არ სრულდება ექსტრემალური დაპროგრამების დროს, თუმცა ამ დროსაც სასარგებლოა პერსპექტიული განვითარების ჰიპოტეზის შემუშავება, სამომავლო სამუშაოებისათვის.

3. შემდგომი იტერაციების მოთხოვნები დამტკიცებულია: პროექტის შესახებ ყველა ცნობები, რომლებიც მე-2 საკონტროლო წერტილშია წარმოდგენილი, მე-3 საკონტროლო წერტილში მისვლის მომენტში უნდა იყოს შეთანხმებული დასამტკიცებლად. დიდი და რთული პროექტებისთვის ეს საკითხი ფორმალურად უნდა აისახოს, ანუ გაფორმდეს საანგარიშო რეპორტის სახით, რომელიც დამტკიცდება. მარტივ შემთხვევებში ეს არაა საჭირო. მას პროექტის მენეჯერი აკონტროლებს.

4. სარეალიზაციო სცენარების სპეციფიკაციები შედგენილია: კონსტრუირების ეტაპის დასაწყისი უკავშირდება პროექტის (იტერაციის) ამოცანის დეკომპოზიციას და სისტემის არქიტექტურის აგებას. როდესაც განსაზღვრული იქნება არქიტექტურა (თუნდაც ზოგადად), შესაძლებელი იქნება ქვესისტემების დამმუშავებლების სამუშაოების დაწყება. გუნდის ხელმძღვანელებს ექნებათ საპასუხისმგებლო სფეროები მოცემული იტერაციისათვის. პროექტის მენეჯერის მოვალეობა ამ წერტილში არის

გუნდების მიერ მომზადებული სარეალიზაციო სცენარების დასამტკიცებლად გაფორმება.

5. სპეციფიკაციები დამტკიცებულია: საკონტროლო წერტილი აღნიშნავს საკონსტრუქტორო ეტაპის დასასრულს. შემდეგი იტერაციის არქიტექტურა დამტკიცებული და დაფიქსირებულია დავალებების სახით ქვესისტემების დამმუშავებლებისა და გუნდის ხელმძღვანელებისათვის, რომელთაც ევალებათ პროექტის კლასების შექმნა ან მოდელირება.

6. პროგრამული სისტემის ავტონომიური შემოწმება დასრულდა და კომპლექსური ტესტირება დაიწყო: დაპროგრამების ეტაპის დასრულებისას საჭიროა სისტემის მუშაობისუნარიანობის კომპლექსური შემოწმება. ესაა შეფასების ეტაპის დაწყება, ვინაიდან ამ მომენტიდან არის შესაძლებელი პროექტის (ან იტერაციის) შესახებ მოსაზრებების სისწორის პრაქტიკულად შემოწმება. სწრაფი დამუშავების (ექსტრემალური პროგრამირება) პროექტებში არ თვლიან აუცილებლად კონსტრუირების ეტაპის ცალკე გამოყოფას (სწ-ები 3,4,5 შერწყმულია). ასევე ამ მეთოდის გამოყენებისას არ იყოფა შედეგების შემოწმება ავტონომიურად და კომპლექსურად, რადგანაც სისტემის ყოველი ახალი შესაძლებლობა თავიდანვე ინტეგრირდება უკვე არსებულ შესაძლებლობებთან. აქ უნდა გადაწყდეს, თუ როგორ ჩაშენდეს ახალი შესაძლებლობა არსებულ არქიტექტურულ კარკასში.

7. ტესტირება დასრულდა, დაიწყო მზადება ახალი იტერაციისათვის: ეს საკონტროლო წერტილი ასახავს პროგრამული სამუშაოების დასასრულს მოცემულ იტერაციაზე. პროგრამული სამუშაოები გრძელდება, რომლებიც დაკავშირებულია პროექტის საბაზო გარემოს გაფართოებასთან. იგი განიხილება როგორც შედარებით დამოუკიდებელი ეტაპი, რომელიც ჩადგმულია შეფასების ეტაპში, და რომელიც თავდება მასთან ერთად (სწ-10). ეს სამუშაოები ორი სახისაა: ა) მხოლოდ ამ პროექტისათვის მრავალჯერადი გამოყენების ზოგადი კომპონენტების გამოყოფა (სწ-9-ში); ბ) ზოგადი (არა მხოლოდ ამ პროექტისთვის) მრავალჯერადი გამოყენების კომპონენტების გამოყოფა (ამ სამუშაოების დაწყება მიზანშეწონილია მაშინ, როცა იტერაციის პროგრამული პროდუქტი განიხილება როგორც მზა აპლიკაცია (სწ-9) და დასრულდეს სისტემის გასავრცელებლად გადაცემის მომენტში (სწ.10)). პროექტის იტერაციული განვითარების უზრუნველყოფა ხდება ცნობების შეკრებით (ცოდნის მიღებით) ახალი იტერაციისათვის ამ საკონტროლო წერტილში.

8. ახალი იტერაციისთვის მოთხოვნილებები მიღებულია: ახალი იტერაციის ცნობების დამუშავების დროს განისაზღვრება მისაღები მოთხოვნილებანი, რომლებიც ფართოვდება სისტემის გამოყენების პირველ პერიოდში დამმუშავებლების მიერ (მას *ალფა-ტესტირების* პერიოდს უწოდებენ). მისი დასრულების შემდეგ ხდება სასიცოცხლო ციკლის გახლეჩა სისტემის ექსპლუატაციაში გადაცემისა და ახალი იტერაციული სამუშაოების ორგანიზების პროცესების პარალელურად შესრულების მიზნით. გახლეჩა

იწვევს სისტემის ორი ვერსიის არსებობას: ერთი გამოყენების პროცესშია, ხოლო მეორე ჩაისახება ახალი მოთხოვნილებების სახით. ესაა დრო, როდესაც უნდა შემოწმდეს პროექტის აპრიორული ჰიპოტეზები, ჩატარდეს პროექტის მაჩვენებელთა და ნორმატივების კორექტირება,

9. დაიწყო სისტემის გამოყენება: ეს საკონტროლო წერტილი ასახავს ე.წ. *ბეტა-ტესტირების* დაწყებას, ანუ სისტემის გადაცემას ექსპლუატაციაში, მისი გარე შეფასების მიზნით. შესაბამისი სამუშაოების ჩატარების შემდეგ (ნაპოვნი დეფექტების გასწორება) შესაძლებელია იტერაციის შედეგების გავრცელება (სწ-10). სწ-9 მიუთითებს *პროექტის (იტერაციის) ფაზის დასარულს.* ეს ფაზა მსგავსია ექსპლუატაციის და თანხლების ტრადიციული ფაზებისა, მაგრამ არის განსხვავებაც. ვინაიდან ოო-პროექტს საქმე აქვს სისტემის ვერსიების იერარქიასთან, რომლებშიც ასახულია შესაძლებლობათა გაფართოება. ეს ფაზა იკვეთება შეფასების ეტაპთან. პროექტის (იერაციის) ფაზის დასასრული მოიცავს:

- სისტემის პაკეტირება ან მიწოდება მომხმარებელზე (სწ-9);
- პროგრამული პროდუქტის თანხლება;
- სამუშაოს დასრულების ეტაპი (სწ.: 11,12)

10. პროგრამული სისტემა ან მისი ვერსია გადაცემულია გასავრცელებლად: ახალ ვერსიას უჩნდება ახალი მომხმარებლები, რომელთაც ესაჭიროებათ მომსახურება.

11. შეტყობინება სისტემის (ვერსიის) მხარდაჭერის შეწყვეტის შესახებ: აუცილებელია მომხმარებელთა ინფორმირება ასეთი გადაწყვეტილების მიღების შემთხვევაში და დასაბუთება. მომხმარებელმა უნდა შეძლოს ახალ რესურსებზე გადაწყობა ან დაასაბუთოს ძველის თანხლების აუცილებლობა.

12. პროგრამული სისტემა მოხსნილია წარმოებიდან: სისტემაზე (ან მის ვერსიაზე) თანხლება შეწყვეტილია. ზოგიერთ მომხმარებელს დასჭირდება თანხლების გაგრძელება გარკვეული ვადით. კლიენტები რომ არ დაკარგონ, პროგრამისტები ეთანხმებიან დროებით თანხლებაზე, ან სისტემის განახლებაზე. დაფინანსება შესაბამისად ხორციელდება კლიენტის მიერ.

II ნაწილი

ბიზნესის მართვის საინფორმაციო სისტემების უსაფრთხოების საერთაშორისო სტანდარტები და აგების ტექნოლოგიები

VI თავი. BSI და ინფორმაციული უსაფრთხოების ISO სტანდარტები 126

VII თავი. ინფორმაციული ტექნოლოგიების ინფრასტრუქტურის ბიბლიოთეკა (ITIL) 146

VIII თავი. COBIT სტანდარტები 212

IX თავი. პროგრამული აპლიკაციების ინტეგრაციის თანამედროვე ტექნოლოგიები 217

VI თავი

BSI და ინფორმაციული უსაფრთხოების ISO სტანდარტები

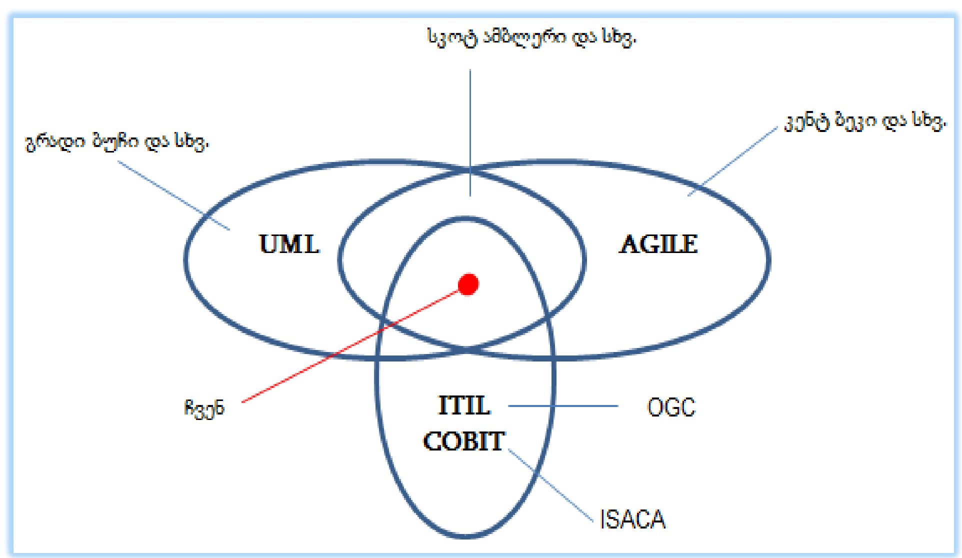
კორპორაციული ბიზნესპროცესების მენეჯმენტის მექანიზმების მუდმივი სრულყოფით შესაძლებელი ხდება მთლიანი სისტემის სასიცოცხლო ციკლის გახანგრძლივება, რაც უდავოდ აქტუალური საკითხია ინტეგრაციის და რეინჟინერინგის ამოცანების გადასაწყვეტად [1,2].

კორპორაციული ობიექტების მართვის საინფორმაციო სისტემის შექმნა მოიცავს ბიზნესპროცესების მოთხოვნილებათა განსაზღვრის, დიაგნოსტიკური ანალიზის, ბიზნესპროგრამების დაგეგმვის და პროექტირების, IT-სერვისების დადგენის, მათი განხორციელების ორგანიზების, ფაქტ-შედეგების აღრიცხვის, რისკების ანალიზის და შეფასების, ინფორმაციული უსაფრთხოების უზრუნველყოფის, ობიექტზე ეფექტური ზემოქმედების მმართველი გადაწყვეტილების მიღების პროცესების ხელშემწყობი მექანიზმების შემუშავებას და მათ კომპიუტერულ რეალიზაციას, მენეჯმენტის საინფორმაციო სისტემების აგების თანამედროვე კონცეფციების საფუძველზე, როგორიცაა ITIL და COBIT [3,4].

სრულყოფილი და საიმედო, მოქნილი პროგრამული უზრუნველყოფის (Software Engineering) სრულყოფილად და სწრაფად დაპროექტება, რეალიზაცია, დანერგვა და შემდგომი თანხლება სისტემის დამკვეთ ორგანიზაციაში მეტად მნიშვნელოვანი ამოცანაა და მისი ეფექტურად გადაწყვეტა ბევრადაა დამოკიდებული როგორც საპროექტო-დეველოპმენტის გუნდის შემადგენლობასა და გამოცდილებაზე, ასევე IT-ინფრასტრუქტურასა და CASE-ინსტრუმენტებზე, კერძოდ, UML და Agile მეთოდოლოგიების გამოყენებაზე [5-9, 16, 28].

6.1 ნახაზზე მოცემულია პროგრამული სისტემების აგების პროცესში თანამედროვე ინფორმაციული ტექნოლოგიების უნივერსალური, მოქნილი და უსაფრთხო სტანდარტების კომპლექსური გამოყენების კონცეფცია, რომლის საფუძველზე შესაძლებელია ეფექტური და საიმედო მენეჯმენტის განხორციელება.

სახელმწიფო ორგანიზაციების და კერძო ბიზნესის სფეროს მართვის საინფორმაციო სისტემების აგების ასეთი მიდგომა, ინფორმაციული უსაფრთხოების სისტემების თვალსაზრისით, ინტენსიურად განიხილება ამერიკის, ევროპის, რუსეთის და სხვა ქვეყნების საუნივერსიტეტო-სამეცნიერო და სამრეწველო-პრაქტიკული დანიშნულების ლიტერატურაში.



ნახ. 6.1. პროგრამული სისტემების და IT-სერვისების მენეჯმენტის მეთოდოლოგიათა კვეთა
- OGC (Office of Government Commerce) - სახელმწიფო სავაჭრო პალატა (დიდი ბრიტანეთი),
- ISACA (Information Systems Audit and Control Association) - საინფორმაციო სისტემების აუდიტის და კონტროლის ასოციაცია (აშშ)

წინამდებარე წიგნში შეტანილია ინგლისურ, გერმანულ და რუსულენოვანი ოფიციალური გამოცემების თარგმნის, გაანალიზებისა და ქართული ორგანიზაციული მართვის სისტემების მენეეჯმენტის თვალსაზრისით ადაპტირებული მასალა, რომელიც ახალია ჩვენი მკითხველისასთვის და უდავოდ სასარგებლო იქნება.

წიგნის ამ ნაწილში გადმოცემულია ინფორმაციული ტექნოლოგიების უსაფრთხო გამოყენების სტანდარტების არსი, მნივნელობა, დანიშნულება და საერთაშორისო მდგომარეობა. დეტალურად განიხილება BSI სტანდარტების სტრუქტურა და კომპონენტები, სტრატეგია და პრინციპები, რესურსები და პროცესები რისკების ანალიზის, შეფასების და მონიტორინგის, რეალიზაციისა და დანერგვის კონცეფციებია. ნაჩვენებია BSI სტანდარტში ITIL და COBIT სისტემების ადგილი, როგორც პროგრამული უზრუნველყოფის მენეჯერის აუცილებელი მეთოდოლოგიური ინსტრუმენტი.

6.1. BSI და ინფორმაციული უსაფრთხოება

BSI - British Standards Institution (ბრიტანეთის სტანდარტების ინსტიტუტი) შეიქმნა 1901 წელს როგორც ინჟინრების კომიტეტი სტანდარტების განსაზღვრის საკითხებზე [34]. ამჟამად იგი სტანდარტების საერთაშორისო კომიტეტის (ISO) წევრია. მისი ჯგუფის ფუნქციებია:

- მენეჯმენტის სისტემებზე სერვისები და გადაწყვეტები;
- სერვისები შეფასებებზე და სერტიფიკაციაზე;
- პროდუქციის სერტიფიკაცია;
- მენეჯმენტის სისტემების სწავლება;
- სტანდარტები და გამოცემები;
- და სხვა.

წინამდებარე წიგნში ჩვენ ვიხილავთ გერმანიის საინფორმაციო ტექნოლოგიების უსაფრთხოების ფედერალურ ბიუროს (BSI - Bundesamt für Sicherheit in der Informationstechnik) მიერ შემუშავებულ სტანდარტებს [29-33]. მათ სორის BSI-Standard 100-1: Managementsysteme für Informations-sicherheit (ISMS) სტანდარტი აღწერს, თუ როგორ შეიძლება აიგოს ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემა (ISMS).

მენეჯმენტის სისტემა მოიცავს ყველა წესს, რომლებიც ითვალისწინებს კონტროლს და მართვას ორგანიზაციის მიზნების მისაღწევად. ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემა, ამგვარად განსაზღვრავს თუ რომელ მეთოდებს და ინსტრუმენტებს მიმართავს გასაგებად ორგანიზაციის ხელმძღვანელობის დონე ინფორმაციულ უსაფრთხოებაზე ორიენტირებული ამოცანებისა და ქმედებებისათვის.

ეს BSI-სტანდარტი პასუხობს შემდეგ კითხვებს:

- რა არის მართვის საინფორმაციო უსაფრთხოების წარმატების ფაქტორები
- როგორ შეიძლება უსაფრთხოების პროცესის მართვა და მონიტორინგი საპასუხისმგებლო მენეჯმენტით ?
- როგორ ხდება უსაფრთხოების მიზნებისა და შესაბამისი უსაფრთხოების სტრატეგიის განვითარება ?
- როგორ შეირჩევა უსაფრთხოების ზომები და როგორ იქმნება უსაფრთხოების კონცეფცია (პოლიტიკა)
- როგორ შეიძლება უსაფრთხოების ერთხელ მიღწეული დონის შენარჩუნება და სრულყოფა

მენეჯმენტის ეს სტანდარტი ასახავს მოკლედ და თვალნათლივ ინფორმაციული უსაფრთხოების მენეჯმენტის უმნიშვნელოვანეს ამოცანებს. ამ რეკომენდაციების რეალიზაციის დროს BSI გვეხმარება IT-საბაზო დაცვის მეთოდიკით. IT-საბაზო დაცვა იძლევა ეტაპობრივ ინსტრუქციებს ინფორმაციული უსაფრთხოების მენეჯმენტის

დასამუშავებლად პარაქტიკაში და კონკრეტულ ზომებს ინფორმაციული უსაფრთხოების ყველა ასპექტით.

IT-საბაზო დაცვის მეთოდიკა აღწერილია BSI-Standard 100-2–ში და და ხელს უწყობს სათანადო დონის ინფორმაციული უსაფრთხოების დონის მიღწევას შესაბამისი ეკონომიკური ეფექტით [30]. ამასთანავე რეკომენდებულია IT-საბაზო დაცვის სტანდარ-ტული უსაფრთხოების ზომების კატალოგები უსაფრთხოების შესაბამისი დონის პრაქტიკული რეალიზაციისთვის.

6.2. რა არის ინფორმაციული უსაფრთხოება

ინფორმაციული უსაფრთხოების მიზანია ყველა სახისა და წარმომავლობის ინფორმაციის დაცვა. ეს ინფორმაცია შეიძლება ინახებოდეს როგორც ქაღალდზე, ისე კომპიუტერულ სისტემებში, ან თუნდაც მომხმარებელთა გონებაში. IT-უსაფრთხოება დაკავებულია, პირველ რიგში, ელექტრონულად შენახული ინფორმაციის უსაფრთხოებაზე და მის დამუშავებაზე.

ინფორმაციული უსაფრთხოების კლასიკური საბაზო ფასეულობებია კონფიდენციალობა, მთლიანობა და წვდომა. ბევრი მომხმარებელი თავიანთ წამოდგენაში განიხილავს სხვა ფასეულობებსაც. ეს შეიძლება სასარგებლოც იყოს ინდივიდუალური აპლიკაციების თვალსაზრისით.

ინფორმაციული უსაფრთხოების სხვა გენერირებული ზოგადი ტერმინებია, მაგალითად, აუთენთიციტეტი, პასუხისმგებლობა, საიმედოობა და უმტყუნობა.

ინფორმაციის უსაფრთხოებას ემუქრება არა მხოლოდ განზრახ ქმედებები (მაგალითად, კომპიუტერული ვირუსები, ინფორმაციის წართმევა/მოსმენა, კომპიუტერის ქურდობა). შემდეგი მაგალითები იძლევა ამის ილუსტრაციას:

- დაუძლეველი ძალის მიერ (როგორიცაა ცეცხლი, წყალი, ქარიშხალი, მიწისძვრა) მედია–მატარებლები და IT-სისტემები დაზარალებულია ან ჩაშლილია ხელმისაწვდომობა მონაცემთა ცენტრში. დოკუმენტები, IT-სისტემები ან სამსახურები აღარაა სურვილისამებრ ხელმისაწვდომები;
- მას შემდეგ, რაც მოხდა წარუმატებელი პროგრამული განახლება, აპლიკაციები აღარ ფუნქციონირებს ან მონაცემები შეუმჩნევლად შეიცვალა;
- მნიშვნელოვანი ბიზნესპროცესი ჭიანურდება, რადგან ერთადერთი ადამიანი, ვინც იცნობს პროგრამებს, ავადაა;
- კონფიდენციალური ინფორმაცია შემთხვევით გადაეცა არასანქცირებული პირს, რადგან დოკუმენტები ან ფაილი არ იყო მონიშნული, როგორც „საიდუმლო".

6.3. ISO სტანდარტები ინფორმაციული უსაფრთხოებისათვის

ინფორმაციული უსაფრთხოების სფეროში სხვადასხვა სტანდარტი შემუშავდა, რომლებშიც ნაწილობრივ სხვადასხა მიზნობრივი ჯგუფი ან თემატური სფერო არის წინა პლანზე წამოწეული. უსაფრთხოების სტანდარტების გამოყენება ბიზნესში ან ხელისუფლებაში არა მხოლოდ აუმჯობესებს უსაფრთხოების დონეს, ის ასევე ხელს უწყობს სხვადასხვა დაწესებულებებს შორის კოორდინაციას, რომლებშიც უსაფრთხოების ზომები უნდა განხორციელდეს ნებისმიერი ფორმით. ქვემოთ, შემდეგი მიმოხილვა გვიჩვენებს ყველაზე მნიშვნელოვანი სტანდარტების მიმართულებებს.

ISO და IEC საერთაშორისო ნორმების ორგანიზაციებში გადწყდა, რომ ინფორმაციული უსაფრთხოების სტანდარტები გაერთიანებულიყო 2700x სერიაში, რომელიც მუდმივად იზრდება. მნიშვნელოვანი სტანდარტებია:

- ISO 2700x

ეს სტანდარტი იძლევა ზოგად მიმოხილვას ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემების (ISMS) და მათი ურთიერთდამოკიდებულების შესახებ ISO 2700 x –ოჯახის სხვადასხვა სტანდარტებს შორის. აქვე გადმოცემულია ISMS–ის ძირითადი პრინციპები, კონცეფციები, ტერმინები და განსაზღვრებანი.

- ISO 27001

საინფორმაციო ტექნოლოგიების სირთულისა და მოწმობებზე მოთხოვნების გამო მრავალი ინსტრუქცია, სტანდარტი და ინფორმაციული უსაფრთხოების ეროვნული სტანდარტები ბოლო წლებში წარმოიშვა. სტანდარტი ISO 27001 – "ინფორმაციული ტექნოლოგია - უსაფრთხოების ტექნიკა - ინფორმაციული უსაფრთხოების მართვის სისტემის მოთხოვნების სპეციფიკაცია" არის **პირველი საერთაშორისო სტანდარტი** ინფორმაციული უსაფრთხოების მართვაში, რომელიც ასევე სერტიფიცირების საშუალებას იძლევა.

ISO 27001 იძლევა 10 - გვერდიან ზოგად რეკომენდაციებს, მათ შორის შესავლის (დანერგვის), ექსპლუატაციიას და დოკუმენტირებული ინფორმაციის უსაფრთხოების მართვის სისტემის სრულყოფისათვის, ასევე რისკების გათვალისწინებით. ნორმატიულ დანართში მოხსენიებულია კონტროლი ISO / IEC 27002 დან. თუმცა, მკითხველი არ იღებს დახმარებას პრაქტიკული განხორციელებისთვის.

- ISO 27002

ISO 27002–ის (ყოფილი ISO 17799:2005) "ინფორმაციული ტექნოლოგიები - ინფორმაციული უსაფრთხოების მენეჯმენტის საპროცესო კოდექსი" მიზანია ინფორმაციული უსაფრთხოების მენეჯმენტის ჩარჩოს განსაზღვრა. ამიტომაც ISO 27002 პირველ რიგში ეხება აუცილებელ ბიჯებს (ეტაპებს), ფუნქციონირებადი უსაფრთხოების მენეჯმენტის ასაგებად და ორგანიზაციაში მიმაგრებას.

აუცილებელი უსაფრთხოების ზომები მოკლედაა აღწერილი ISO-სტანდარტის ISO/IEC 27002–ში, დაახლოებით 100 გვერდზე. რეკომენდაციები, რომელიც განკუთვნილია მართვის დონისთვის და, შესაბამისად, შეიცავს მცირე კონკრეტულ ტექნიკურ შენიშვნებს. უსაფრთხოების ISO 27002–ის რეკომენდაციების რეალიზაცია არის ერთ–ერთი გზა მრავალი შესაძლებლობიდან, რომლებიც აკმაყოფილებს ISO სტანდარტის 27001–ის მოთხოვნებს.

შენიშვნა: ISO 17799 სტანდარტი 2007 წლის დასაწყისში გადაეცა არსებითი ცვლილებების გარეშე ISO 27002–ს, იმისათვის, რომ ხაზი გაესვათ მის მიკუთვნებაზე ISO 2700x სერიისათვის.

- ISO 27005

ეს ISO–სტანდარტი „ინფორმაციული უსაფრთხოების რისკების მენეჯმენტი“ შეიცავს ძრითად რეკომენდაციებს რისკების მართვის შესახებ ინფორმაციული უსაფრთხოებისათვის. მათ შორის იგი მხარს უჭერს ISO/IEC 27001 სტანდარტის მოთხოვნების რეალიზაციას, ოღონდ აქ არავითარი მეთოდი რისკების მართვისათვის არაა მოცემული.

ISO/IEC 27005 ცვლის ISO 13335-2 სტანდარტს. ეს სტანდარტი ISO 13335-2 „უსაფრთხოების ინფორმაციულ–კომუნიკაციური ტექნოლოგიები, ნაწ.2: რისკების მენეჯმენტის მეთოდები ინფორმაციულ უსაფრთხოებაში“, იძლეოდა ინსტრუქციებს ინფორმაციული უსაფრთხოების მენეჯმენტისათვის.

- ISO 27006

ISO-სტანდარტი 27006 „ინფორმაციული ტექნოლოგია – უსაფრთხოების უზრუნველყოფის მეთოდები – მოთხოვნები სერტიფიცირების აკრედიტაციული ორგანოების მიმართ ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემებში“, განსაზღვრავს აკრედიტაციის მოთხოვნებს სერტიფიცირების ორგანოებისთვის ISMS–ში და განიხილება ამ სერტიფიცირების პროცესების თავისებურებანი.

- ISO-2700x- რიგის სხვა სტანდარტები

ISO 2700x სტანდარტული სერია სავარაუდოდ გრძელვადიან პერსპექტივაში ISO სტანდარტების 27000-27019 27030-27044 სახით დაკომპლექტდება. ამ სერიის ყველა სტანდარტი მოიცავს უსაფრთხოების მართვის სხვადასხვა ასპექტებს და ეფუძნება ISO 27001–მოთხოვნებს. სხვა სტანდარტების მიზანია გააუმჯობესოს გაგება და პრაქტიკული გამოყენების ISO 27001–ის. ეს შეთანხმებაა, მაგალითად, ISO 27001–ის პრაქტიკული განხორციელებისთვის, ანუ რისკების შეფასება ან რისკების მართვის მეთოდებია.

6.4. BSI სტანდარტები ინფორმაციული უსაფრთხოებისთვის

- 100-1 ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემები (ISMS)

ეს სტანდარტი განსაზღვრავს ISMS–ის ზოგად მოთხოვნებს. ეს არის სრულად თავსებადი ISO სტანდარტის 27001 და კვლავაც გაითვალისწინებს რეკომენდაციებს ISO სტანდარტების 27000 და 27002. იგი სთავაზობს მკითხველს ადვილად გასაგებ და სისტემატურ ინსტრუქციებს, მიუხედავად იმისა, თუ რომელი მეთოდის საშუალებით სურთ მათ მოთხოვნების განახორციელება.

BSI წარმოადგენს ISO სტანდარტის შინაარსს საკუთარ BSI სტანდარტში, გარკვეული საკითხების უფრო დეტალურად აღსაწერად, და ამით შინაარსის დიდაქტიკური წარმოდგენის საშუალება მიეცეს. გარდა ამისა, სტრუქტურა იყო ისე დამუშავებული, რომ იგი თავსებადია IT-საბაზო დაცვის მეთოდებთან. უნიფიცირებული სათაურების საშუალებით აღნიშნულ დოკუმენტებში მკითხველისთვის ძალიან მარტივია ორიენტაირება.

- 100-2 IT-საბაზო დაცვა–მეთოდკა

IT–საბაზო–დაცვა–მეთოდიკა აღწერს ეტაპობრივად, ნაბიჯ–ნაბიჯ, თუ როგორ უნდა აიგოს ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემა პრაქტიკულად და როგორ მოხდეს მისი ექსპლუატაცია. ინფორმაციული უსაფრთხოების მენეჯმენტის ამოცანები და ორგანიზაციული სტრუქტურის აგება ინფორმაციული უსაფრთხოებისთვის ძალზე მნიშვნელოვანი თემებია. IT–საბაზო–დაცვა–მეთოდიკა დეტალურად განიხილავს იმას, თუ როგორ შეიძლება უსაფრთხოების კონცეფციის (პოლიტიკის) დამუშავება პრაქტიკაში, როგორ აირჩევა შესაბამისი უსაფრთხოების ზომები და რა შეიძლება ჩაითვალოს უსაფრთხოების კონცეფციის რეალიზაციად. ასევე საკითხი, როგორ ხდება ინფორმაციული უსაფრთხოების მხარდაჭერა და სრულყოფა ექსპლუატაციის პირობებში, არის ამომწურავად პასუხგაცემული.

IT–საბაზო–დაცვა BSI-Standard 100-2 სტანდარტთან კავშირში, ახდენს აქამდე დასახელებული 27000, 27001 და 27002 ISO-სტანდარტების ძალზე ზოგადად მიღებული მოთხოვნების ინტერპრეტირებას და ეხმარება მომხმარებლებს პრაქტიკაში რეალიზაციის დროს, მრავალი შენიშვნით, საცნობარო ინფორმაციითა და მაგალითით.

IT–საბაზო–დაცვის–კატალოგები ხსნიან არა მხოლოდ იმას, თუ ***რა*** უნდა გაკეთდეს, არამედ იძლევა ძალიან კონკრეტულ შენიშვნას, თუ ***როგორ*** უნდა გამოიყურებოდეს რეალიზაცია (ასევე ტექნიკურ დონეზე). პროცესი IT–საბაზო–დაცვის მიხედვით არის აპრობირებული და ეფექტური შესაძლებლობა, ზემოჩამოთვლილ ISO-სტანდარტის ყველა მოთხოვნის შესასრულებლად.

- 100-3 რისკების ანალიზი IT–საბაზო–დაცვის საფუძველზე

BSI–მ დაამუშავა რისკების ანალიზის მეთოდიკა IT–საბაზო–დაცვის საფუძველზე. მის გამოყენებას აზრი აქვს მაშინ, როცა კომპანიები ან სახელმწიფო დაწესებულებები

მუშაობენ წარმატებით IT–საბაზო–დაცვასთან და უჩნდებათ სურვილი დამატებითი უსაფრთხოების ანალიზის ჩასატარებლად,

- 100-4 საგანგებო სიტუაციათა მენეჯმენტი

BSI Standard 100-4 სტანდარტში ახსნილია სახელმწიფო დაწესებულებათა ან კომპანიების მასშტაბებში საგანგებო სიტუაციათა მენეჯმენტის აგებისა და ექსპლუატაციის მეთოდიკა. აქ აღწერილი მეთოდიკა ეფუძნება BSI-Standard 100-2 სტანდარტის IT–საბაზო–დაცვის–მეთოდიკას და აფართოვებს მას სასარგებლოდ.

- ISO 27001 სერტიფიცირება IT–საბაზო–დაცვის საფუძველზე

BSI ახდენს საინფორმაციო ქსელების სერტიფიცირებას, ანუ ინფრასტრუქტურული, ორგანიზაციული, პერსონალური და ტექნიკური კომპონენტების ურთიერთმოქმედებას, რომლებიც გამოიყენება ბიზნესპროცესების და ტექნიკური დავალებების სარეალიზაციოდ. BSI სერტიფიცირება მოიცავს როგორც გამოცდას ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემებში, ისე გამოცდას კონკრეტულ უსაფრთხოებიზ ზომებში IT–საბაზო დაცვის საფუძველზე.

BSI სერტიფიცირება ამასთანავე ყოველთვის მოიცავს ოფიციალურ ISO–სერტიფიცირებას ISO 27001–ის მიხედვით, მაგრამ ბევრად მნიშვნელოვანია, ვიდრე უბრალოდ ISO–სერტიფიცირება, დამატებითი კვლევით-ტექნიკური ასპექტების გამო. უსაფრთხოების მენეჯმენტის გამოცდის ძირითადი მოთხოვნები აუდიტის ჩარჩოებში (კონტექსტში) აღმოცენდება უსაფრთხოების მენეჯმენტის საბაზო დაცვის ბლოკის (B 1.0) ღონისძიებებიდან. ამ ბლოკის ეს ზომები ისეა დაწერილი, რომ ISMS-ის BSI–სტანდარტის მნიშვნელოვანი მოთხოვნები სწრაფად იყოს იდენტიფიცირებული (განსაზღრული). 1.1 ნახაზი იძლევა BSI–დოკუმენტების ზოგადი სტრუქტურის ილუსტრაციას.

ISO 27001 სტანდარტთან ადაპტირებისათვის ჩატარდა კორექტირებები სერტიფიცირების სქემაში ინფორმაციული ქსელებისა და სერტიფიცირების სქემაში აუდიტორებისათვის [35, 36].

6.5. სხვა სტანდარტები: ITIL და COBIT

- **ITIL** (IT- Infrastructure Library – IT ინფრასტრუქტურის ბიბლიოთეკა) არის IT სერვის მენეჯმენტის რამდენიმე წიგნის კოლექცია [3]. იგი შემუშავებულ იქნა გაერთიანებული სამეფოს სახელმწიფო კომერციის მთავრობის მიერ (OGC). ITIL განიხილავს IT–სერვისების მენეჯმენტს IT-მომსახურების თალსაზრისით. IT–მომსახურება შეიძლება იყოს როგორც შიგა IT–დეპარტამენტის ან გარე სერვისის პროვაიდერის. საერთო მიზანი არის ოპტიმიზაცია და ხარისხის გაუმჯობესების IT მომსახურების და ხარჯების ეფექტურობის.
- **COBIT** (Control Objectives for Information and related Technology – კონტროლის მიზნები ინფორმაციული და დაკავშირებული ტექნოლოგიებისათვის) აღწერს რისკების კონტროლის მეთოდს, რომელიც ხორციელდება IT–დანერგვის საშუალებით

კრიტიკული ბიზნესპროცესების შესრულების მხარდასაჭერად [4]. COBIT–დოკუმენტები გაიცემა საინფორმაციო სისტემების აუდიტისა და კონტროლის ასოციაციის (ISACA–Information Systems Audit and Control Association) IT მართვის ინსტიტუტის (ITGI – IT Governance Institute) მიერ. COBIT–ის დამუშავების დროს ავტორები ორიენტირებული იყვნენ უსაფრთხოების მენეჯმენტის არსებულ სტანდარტებზე, როგორიცაა ISO 27002.

ITIL ბიბლიოთეკა და და COBIT სტანდარტი განხილულ იქნება წიგნის მომდვნო თავებში.

6.6. ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემის კომპონენტები

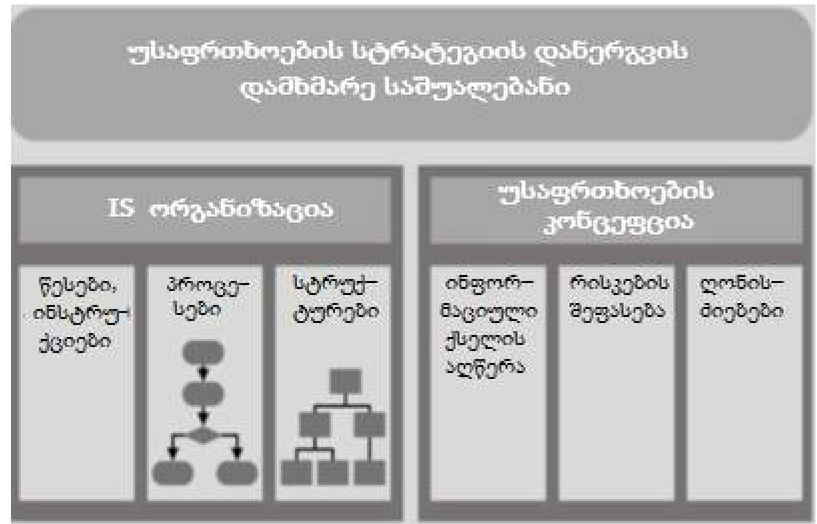
ინფორმაციული უსაფრთხოების მენეჯმენტის სისტემა (ISMS – Information Security Management System) სჭირდება სახელმწიფო და კერძო სტრუქტურების ყველა ორგანიზაციას. ეს მნიშვნელოვანი რგოლი დაკავებიულია ინფორმაციული უსაფრთხოების საკითხებით. ISMS ამტკიცებს, თუ რომელი ინსტრუმენტებით და მეთოდებით მართავს (გეგმავს, ნერგავს, ასრულებს, აკონტროლებს და სრულყოფს) მენეჯმენტი ინფორმაციულ უსაფრთხოებაზე მიმართულ ამოცანებს და ქმედებებს მიზანმიმართულად. ISMS -ს მიეკუთვნება შემდეგი ძირი¬თადი კომპონენ¬ტები (ნახ.6.2):

- მენეჯმენტის პრინციპები;
- რესურსები;
- თანამშრომლები;
- უსაფრთხოების პროცესი;
- ხელმძღვანელობა ინფორმაციული უსაფრთხოებისათვის, რომელშიც მისი მიზნები და სტრატეგია რეალიზაციისათვის დოკუმენტირებულია;
- უსაფრთხოების კონცეფცია (პოლიტიკა);
- ინფორმაციული უსაფრთხოების ორგანიზაცია.

ნახ.6.2. ISMS–ის შედგენილობა

ინფორმაციული უსაფრთხოების ორგანიზაცია და უსაფრთხოების კონცეფცია არის მენეჯმენტის ინსტრუმენტი მისი უსაფრთხოების სტრატეგიის დასანერგად. 6.3 და 6.4 ნახაზები ამ დამოკიდებულებას ნათელს ჰფენს.

ნახ.6.3. ინფორმაციული უასაფრთხოების სტრატეგია,
ISMS–ის მთავარი კომპონენტი



ნახ.6.4. უსაფრთხოების სტრატეგიის დანერგვა

უსაფრთხოების სტრატეგიის ძირითადი პუნქტები დოკუმენტირებულ იქნება სახელმძღვანელო პრინციპებში ინფორმაციის უსაფრთხოებისათვის. უსაფრთხოების პოლიტიკას ცენტრალური მნიშვნელობა აქვს, რადგან იგი შეიცავს ხელმძღვანელობის ხილულ აღიარებას მათი სტრატეგიის შესახებ.

6.7. სასიცოცხლო ციკლი ინფორმაციულ უსაფრთხოებაში

უსაფრთხოება არ არის უცვლელი მდგომარეობა, რომელიც მიიღწევა ერთხელ და არასდროს იცვლება შემდეგ. ყოველი დაწესებულება ექვემდებარება მუდმივ დინამიკურ ცვლილებებს.

ბევრი ცვლილება უკავშირდება ბიზნესპროცესების, სპეციალიზებული ამოცანების, ინფრასტრუქტურის, ორგანიზაციული სტრუქტურების IT–ის და საინფორმაციო უსაფრთხოების ცვლილებებს.

შესამჩნევ ცვლილებებთან ერთად დაწესებულების ფარგლებში ასევე იცვლება გარე პირობები, როგორიცაა სამართლებრივი ან ხელშეკრულებით გათვალისწინებული მოთხოვნები, აგრეთვე არსებული ინფორმაციის ან საკომუნიკაციო ტექნოლოგიებიც შეიძლება შეიცვალოს რადიკალურად. აქედან გამომდინარე, აუცილებელია უსაფრთხოების აქტიური მართვა, რათა შენარჩუნებულ იქნას უსაფრთხოების მიღწეული დონე.

არ არის საკმარისი, მაგალითად, რომ ბიზნესპროცესების დაგეგმვა ან ახალი IT-სისტემის დანერგვა და მიღებული უსაფრთხოების ზომები განხორციელდს მხოლოდ ერთხელ. უსაფრთხოების ზომების განხორციელების შემდეგ ისინი რეგულარულად უნდა იყოს გამოკვლეული ეფექტურობასა და მიზანშეწონილობაზე, ასევე მათ გამოყენებადობასა და ფაქტობრივ გამოყენებაზე. უნდა მოიძებნოს სუსტი წერტილები ან გაუმჯობესების შესაძლებლობები, უნდა მოხდეს ღონისძიებათა ადაპტირება და გაუმჯობესება.

ეს ადაპტაციის საჭიროებით მოითხოვნილი ცვლილებები უნდა იყოს თავიდან დაგეგმილი და განხორციელებული. თუ ბიზნესპროცესები მთავრდება ან კომპონენტები ან IT–სისტემები იცვლება, ან ამოიღება მომსახურებიდან, მაშინ არსებული უსაფრთხოების ასპექტები უნდა გადაიხედოს (მაგალითად, პრივილეგიების ამოღება ან მყარი დისკების უსაფრთხო წაშალა).

IT-საბაზო_დაცვის კატალოგებში უსაფრთხოების ზომები გადანაწილდება მკითხველის უკეთესი სიცხადისათვის შემდეგ ფაზებში: დაგეგმვა და კონცეფცია; შესყიდვა (საჭიროებების შემთხვევაში); დანერგვა; ექსპლუატაცია (ზომები ექსპლუატაციაში ინფორმაციის უსაფრთხოების მხარდასაჭერად მოიცავს მონიტორინგს და შედეგების კონტროლს); გამოყოფა (საჭიროებების შემთხვევაში) და საგანგებო სიტუაციებისადმი მზადყოფნა.

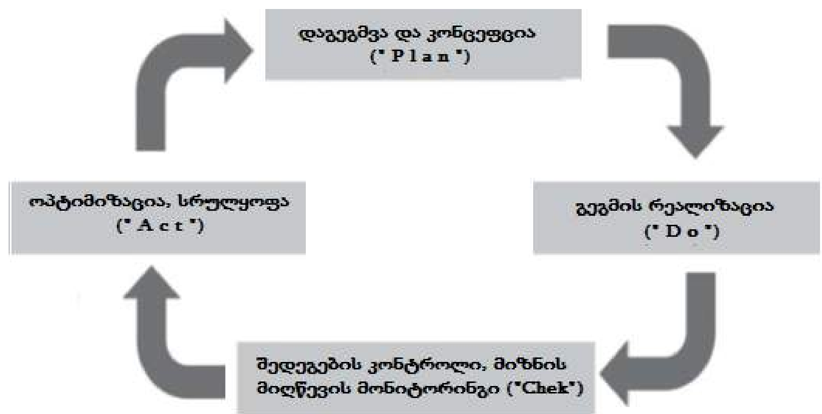
6.8. ინფორმაციული უსაფრთხოების პროცესების აღწერა

არა მხოლოდ ბიზნესპროცესებს და IT-სისტემებს აქვს „სასიცოცხლო ციკლები“. ის აქვს ასევე უსაფრთხოების კონცეფციას, ინფორმაციული უსაფრთხოების ორგანიზაციას და, ბოლოს, თვით მთლიან უსაფრთხოების პროცესს. უსაფრთხოების პროცესის დინამიკის მარტივად აღწერის მიზნით, იგი ლიტერატურაში ხშირად წარმოდგენილია შემდეგი ფაზებით (ნახ.6.5) [106-109]:

1. დაგეგმვა;
2. დანერგვა დაგეგმილის ან პროექტის რეალიზაცია;
3. შედეგების კონტროლი ან მიზნის მიღწევის მონიტორინგი;
4. გამოვლენილი დეფექტებისა და ნაკლოვანებების აღმოფხვრა ან ოპტიმიზაცია და სრულყოფა.

მე-4 ფაზა აღწერს მცირე დეფექტების დაუყოვნებლივ აღმოფხვრას. საფუძვლიანი ან მრავალი ცვლილებების მიზნით უეჭველად ისევ დაგეგმვის ფაზიდან უნდა დაიწყოს პროცესი.

6.5 ნახაზზე მოცემულია მოდელი, ცალკეული ფაზების ინგლისური ქვეაღნიშვნებით („Plan", „Do", „Check", „Act"), ან აღნიშნულია როგორც PDCA–მოდელითა.



ნახ.6.5. სასიცოცხლო ციკლი Deming-ის მიხედვით
(PDCA-მოდელი)

PDCA–მოდელი არის ასევე ISO-Standard 27001 სტანდარტშიც. მისი გამოყენება პრინციპულად შესაძლებელია უსაფრთხოების პროცესის ყველა ამოცანისათვის.

ასევე უსაფრთხოების კონცეფციისა და ინფორმაციული უსაფრთხოების ორგანიზაციის სასიცოცხლო ციკლები შეიძლება ძალზე გასაგებად აღიწეროს. ამ დოკუმენტის ეს თავი ეხება სწორედ სასიცოცხლო ციკლის მოდელის ოთხ ფაზას.

ინფორმაციული უსაფრთხოების პროცესის დაგეგმვის ფაზაზე ანალიზირდება ჩარჩოს (საბაზო) პირობები, რომლებიც უსაფრთხოების მიზნებს განსაზღვრავს და უსაფრთხოების სტრატეგიას ამუშავებს. ისინი შეიცავს საფუძვლიან მტკიცებებს, თუ როგორ უნდა იქნას მიღწეული მიზანი.

უსაფრთხოების სტარატეგია ხორციელდება უსაფრთხოების კონცეფციისა და შესაფერისი სტრუქტურის დახმარებით ინფორმაციიული უსაფრთხოების ორგანიზაციისთვის. უსაფრთხოების კონცეფცია და ინფორმაციიული უსაფრთხოების ორგანიზაცია უნდა იქნას თავიდან დაგეგმილი და რეალიზებული როგორც ამას შედეგების კონტროლი მოითხოვს.

ზედა დონის ინფორმაციიული უსაფრთხოების შედეგების კონტროლის დროს რეგულარულად მოწმდება, არის თუ არა ჩარჩოს პირობები (მაგალითად, კანონები ან ბიზნესმიზნები) შეცვლილი და თუ აღმოჩნდა უსაფრთხოების კონცეფცია და მისი ინფორმაციიული ორგანიზაცია ეფექტური და მოქმედი.

ვინაიდან სხვადასხვა დაწესებულებას აქვს სხვადასხვა საწყისი პირობები, უსაფრთხოების მოთხოვნები და ფინანსური საშუალებები, ეს მეთოდი ნამდვილად გვთავაზობს კარგ ორიენტაციას, მაგრამ უნდა იყოს ადაპტირებული ყოველ კომპანიასა და დაწესებულებაზე თავის საკუთარ მოთხოვნილებებზე. ყოველი დაწესებულება ინდივიდუალურად განსაზღვრავს ან აკონკრეტებს, თუ რომელი ფორმის სასიცოცხლო ციკლის მოდელია მისთვის მისაღები.

მცირე დაწესებულებები და კომპანიები არ უნდა შეშინდეს, რადგან უსაფრთხოების პროცესის ხარჯები როგორც წესი ორგანიზაციის ზომებზეა დამოკიდებული. ამგვარად, ძალიან დიდ კომპანიაში, მრავალი განყოფილებით და თანამშრომლით, ალბათ მოითხოვს უფრო ფორმალურ პროცესს, რომელიც ზუსტად ამტკიცებს, თუ რომელი შიგა და გარე აუდიტებია საჭირო, ვინ ვისთანაა ანგარიშმგებელი, ვინ ქმნის გადაწყვეტილებათა დოკუმენტებს, როდის იძლევა ხელმღვანელობა უსაფრთხოების პროცესზე კონსულტაციას.

მცირე ბიზნესში ესაა ყოვლეწლიური შეხვედრა კომპანიის ხელმძღვანელსა და მის IT–მიმწოდებელს შორის, რომლის დროსაც განიხილება წინა წლის პრობლემები, ხარჯები, ახალი ტექნიკური გადაწყვეტები და სხვა ფაქტორები, რომლებიც უკვე იყო ადაპტირებულ, რათა უსაფრთხოების პროცესის წარმატება კრიტიკულად განიხილოს.

ამ საკითხების უფრო დეტალურად გაცნობა შეიძლება არსებულ ლიტერატურულ წყაროში [10].

6.9. რესურსები ინფორმაციული უსაფრთხოებისათვის

უსაფრთხოების განსაზღვრული დონის მხარდაჭერა ყოველთვის მოითხოვს ფინანსურ, ადამიანურ და დროით რესურსებს, რომლებიც ხელმძღვანელობის მიერ უნდა იყოს საკმარისად უზრუნველყოფილი. თუ მიზნები ვერ მიიღწევა არასაკმარისი რესურსების გამო, აქ პასუხს აგებენ არა პროცესში დაკავებული თანამშრომლები, არამედ ხელმძღვანელები, რომლებმაც არარეალური მიზნები დასახეს ან აუცილებელი რესურსებით ვერ უზრუნველყვეს პროცესი.

იმისთვის, რომ დასმული მიზნების მიღწევის შანსი არ დაიკარგოს, მნიშვნელოვანია, რომ უკვე მიზნების ფორმირებისას განხორციელდეს ხარჯებისა და სარგებლის საწყისი შეფასება. უსაფრთხოების პროცესის მსვლელობისას ეს ასპექტი კვლავ უნდა ასრულებდეს გადამწყვეტ როლს, ერთის მხრივ, რომ არ მოხდეს რესურსების ხარჯვა და, მეორე მხრივ, რომ საჭირო ინვესტიციებით უზრუნველყოფილ იქნას უსაფრთხოების შესაბამისი დონის მიღწევა.

ხშირად IT-უსაფრთხოებასთან ასოცირდება განსაკუთრე-ბული ტექნიკური გადაწყვეტები. ესაა კიდევ ერთი მიზეზი, რომ IT–უსაფრთხოების ნაცვლად ტერმინი ინფორმაციული უსაფრთხოება უკეთ იქნეს გამოყენებული.

უპირველეს ყოვლისა, მნიშვნელოვანია აღინიშნოს, რომ ინვესტიციები ადამიანურ რესურსებში ხშირად უფრო ეფექტურია, ვიდრე ინვესტიციები უსაფრთხოების ტექნიკაში. ტექნიკა დამოუკიდებლად ვერ წყვეტს პრობლემებს, ის ყოველთვის უნდა იყოს მიბმული ორგანიზაციულ გარემოზე. აგრეთვე უსაფრთხოების ზომების ეფექტურობისა და ვარგისიანობის გადამოწმება უზრუნვეყოფილი უნდა იქნას აუცილებელი რესურსით.

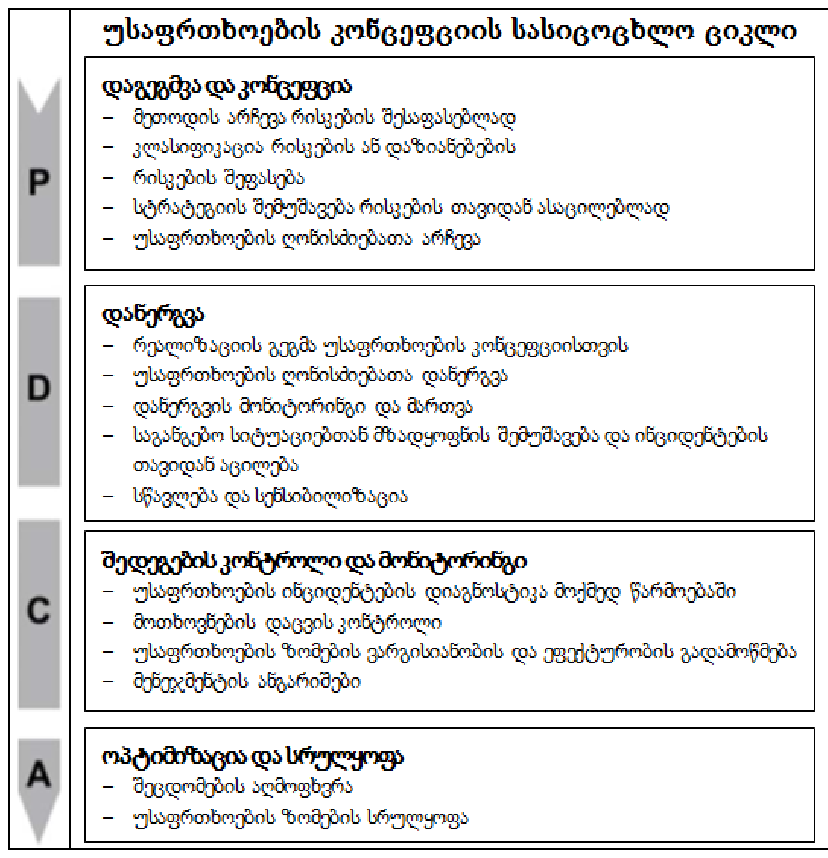
პრაქტიკაში ხშირად უსაფრთხოების შინაგან ექსპერტებს არ ჰყოფნით დრო, რათა გააანალიზონ უსაფრთხოებასთან დაკავშირებული ყველა ფაქტორი და მდგომარეობა (მაგალითად, იურიდიული მოთხოვნები ან ტექნიკური საკითხები). ზოგიერთ შემთხევაში მათ არ გააჩნიათ შესაბამისი ბაზაც. ამიტომ უფრო მისაღებია გარე ექსპერტების გამოყენება, როცა საკითხებისა და პრობლემების გადაწყვეტა ვერ ხერხდება საკუთარი საშუალებებით. ეს უნდა იყოს დოკუმენტალურად დამოწმებული შიგა ექსპერტების მიერ, რათა ხელმძღვანელობის დონემ უზრუნველყოს აუცილებელი რესურსები.

წინაპირობა IT–ის უსაფრთხო მუშაობისათვის კარგად ფუნქციონირებადი IT–ექსპლუატაციაა. ამისათვის კი საკმარისი რესურსებია აუცილებელი. IT–ექსპლუატაციის ტიპური პრობლემები (მცირე რესურსები, გადატვირთული ადმინისტრატორები ან არასტრუქტურირებული და ცუდ მდგომარეობაში მყოფი IT-გარემო), როგორც წესი, უნდა გადაწყდეს, რითაც უსაფრთხოების ზომების განხორციელება ეფექტურად და შედეგიანად იქნება შესაძლებელი.

6.10. უსაფრთხოების პოლიტიკა და რისკები

ინფორმაციული უსაფრთხოების მიზნების დასაკმაყო-ფილებლად და უსაფრთხოების სასურველი დონის მისაღწევად, ჯერ გაგებულ უნდა იქნას, როგორაა ამოცანებისა და ბიზნესპროცესების შესრულება დამოკიდებული ინფორმაციის კონფიდენციალურობაზე, მთლიანობასა და ხელმისაწვდომობაზე.

ამასთანავე განხილულ უნდა იქნას, თუ დაზიანების რომელი მიზეზით რა სიდიდის ძალადობა, ორგანიზაციული ნაკლოვანება, ადამიანური უმოქმედობა ან ასევე IT-რისკები ემუქრება ბიზნესპროცესებს. შემდეგ შეიძლება გადაწყდეს, თუ როგორ იქნას ეს რისკები თავიდან აცილებული. კერძოდ, საჭიროა შემდეგი ქვეეტაპების განხილვა (ნახ.6.6).



ნახ.6.6. უსაფრთხოების კონცეფციის სასიცოცხლო ციკლის მიმოხილვა

მეთოდების შერჩევა გადამწყვეტად მოქმედებს შრომის დანახარჯებზე უსაფრთხოების კონცეფციის (პოლიტიკის) შესაქმნელად.

რისკების შეფასების განსხვავებული სახეები აღწერილია ISO/IEC 27005 ნორმაში. BSI–იმ აქედან ნაწარმოები რამდენიმე მეთოდი დაამუშავა და პრაქტიკაში გამოსცადა. IT-საბაზო–დაცვის–პროცესებში აღწერილია ასევე რისკების შეფასების ძალზე პრაქტიკული მეთოდები, რომლებიც IT-საბაზო–დაცვის–კატალოგის დახმარებით შეიძლება იქნას დანერგილი. ეს მიდგომა ფართოვდება BSI-100–3 სტანდარტით „რისკების ანალიზი, ბაზირებული IT-საბაზო–დაცვაზე “.

IT-საბაზო–დაცვის ან სხვა საუკეთესო–პრაქტიკული–საშუალებების გამოყენებას აქვს ის უპირატესობა, რომ შრომის დანახარჯები საგრძნობლად მცირდება, რადგან ავტორებმა უკვე აღწერეს კონკრეტული მეთოდები და შესაფერისი უსაფრთხოების ზომები შემოგვთავაზეს.

- **რისკების კლასიფიკაცია**

ინფორმაციული უსაფრთხოების მენეჯმენტმა რისკების შეფასების არჩეული მეთოდისაგან დამოკიდებულებით უნდა განსაზღვროს, როგორ კლასიფიცირდება და შეფასდება საფრთხეები, დაზიანებათა პოტენციალი, აღმოცენების ალბათობები და აქედან გამომდინარე – რისკები.

მაგრამ საკმაოდ ძნელი, ხარჯიანია და ამიტომ, შეცდომებითაა, საფრთხეებისა და აღმოცენების ალბათობების ინდივიდუალური მნიშვნელობების დადგენა. რეკომენდებულია, არ დაიხარჯოს დიდი დრო შრომატევად (და შეცდომების შემცველ) რისკების აღმოცენების ალბათობებისა და შესაძლო საფრთხეების ზუსტ განსაზღვრებაზე. ხშირ შემთხვევებში პრაქტიკულია, როგორც რისკების აღმოცენების ალბათობებთან, ისე პოტენციალურ დაზიანებათა სიდიდეებთან მუშაობა კატეგორიებით. აქ გამოყენებულია 3–5 კატეგორია, მაგალითად,

- აღმოცენების ალბათობები: ***იშვიათად, ხშირად, ძალიან ხშირად;***
- პოტენციალური დაზიანების დონე: ***საშუალო, მაღალი, ძალიან მაღალი.***

შემდეგ დაწესებულებისათვის, შესაფერისი გზით განისაზღვრება, ასეთი კატეგორიების გამოყენებით შეიძლება თუ არა რისკების ხარისხობრივი დამუშავება.

- **რისკების შეფასება**

რისკების ყოველი შეფასება მოიცავს შემდეგ ბიჯებს:

- დასაცავი ინფორმაცია და ბიზნესპროცესები უნდა იყოს იდენტიფიცირებული;
- დასაცავი ინფორმაციისა და ბიზნესპროცესების შესაბამისი ყველა საფრთხე უნდა იყოს იდენტიფიცირებული;
- სუსტი ადგილები, რომლებშიც შეუძლია ზემოქმედება საფრთხეებს, უნდა იყოს იდენტიფიცირებული;
- შესაძლო დაზიანებები, გამოწვეული კონფიდენციალობის, მთლიანობია და წვდომის დაკარგვის გამო, უნდა იყოს იდენტიფიცირებული;
- სავარაუდო ზეგავლენები ბიზნესზე ან შესასრულებელ ამოცანებზე, გამოწვეული უსაფრთხოების ინციდენტებით, უნდა იქნას გაანალიზებული;

- რისკი, რომელიც უსაფრთხოების ინციდენტებით იწვევს დაზიანებას, უნდა შეფასდეს.

აქ გამოყენებული ტერმინები „საფრთხე“, „სუსტი ადგილი“ და „რისკი“ განსაზღვრულ იქნება IT–საბაზო–დაცვის კატალოგის ლექსიკონში.

- **სტრატეგიის დამუშავება რისკების დასამუშავებლად**

ხელმძღვანელობის ზედა დონემ უნდა განსაზღვროს, თუ როგორ მოუაროს აღმოჩენილ რისკებს. ინფორმაციული უსაფრთხოების მენეჯმენტის მიერ უნდა იყოს ეს საკითხი მომზადებული. ამისათვის არსებობს შემდეგი ოფციები:

- რისკები შეიძლება შემცირდეს, უსაფრთხოების ადეკვატური ზომების გამოყენებით;
- რისკები შეიძლება შემცირდეს, მაგალითად, ბიზნესპროცესების ან სპეცამოცანების რესტრუქტურირებით ან ამოგდებით;
- რისკები შეიძლება გადაცემულ იქნას, მაგალითად, ოუთსორსინგით ან დაზღვევით;
- რისკები შეიძლება აქცეპტირდეს.

რისკების თავიდან აცილების სახეები უნდა იყოს დოკუმენტირებული და ხელმძღვანელობის ზედა დონის მიერ დამტკიცებული. აუცილებელი რესურსები სტრატეგიის დანერგვისთვის უნდა დაიგეგმოს და უზრუნველყოფილ იქნას.

სტრატეგიის დამუშავებისას ხარჯებისაგან დამატებით, გადაწყვეტილების მნიშვნელოვან კრიტერიუმად განიხილავენ ნარჩენ რისკს, რომელიც გათვალისწინებულ უნდა იქნას ხელმძღვანელობის ზედა დონეზე. ნარჩენი რისკი უნდა შეფასდეს და დოკუმენტირებულ იქნას.

- **სტრუქტურული ანალიზი: დაცვის ობიექტების იდენტიფიკაცია**

სტუქტურული ანალიზის ფარგლებში განისაზღვრება განსახილველი ინფორმაციული ქსელისთვის, როგორიცაა მოქმედების სფერო ან ბიზნესპროცესი, შესაბამისი დაცვის ობიექტები, როგორიცაა ინფორმაციები, აპლიკაციები, IT-სისტემები, ქსელები, ფართები და შენობები, აგრეთვე კომპეტენტური თანამშრომლები.

სტრუქტურული ანალიზის დროს დმატებით წარმოდგენილ უნდა იქნას ურთიერთობები და დამოკიდებულებები ცალკეულ დაცვის ობიექტებს შორის. გამოვლენილი დამოკიდებულებები ძირითადად გამოიყენება უსაფრთხოების ინციდენტების გავლენის დასადგენად ბიზნესპროცესებზე, რათა შემდგომ მოახდინოს სათანადო რეაგირება. მაგალითად: თუ „S-სერვერი" მოხვდება უსაფრთხოების ინციდენტის გავლენის ქვეშ, სასწრაფოდ უნდა გაირკვეს, თუ რომელი აპლიკაციები ან ბიზნესპროცესები იქნება ამით დაზარალებული.

- **დაცვის მოთხოვნების დადგენა: უსაფრთხოების ინციდენტების გავლენის ანალიზი განსახილველ ბიზნესპროცესებზე**

სტრუქტურული ანალიზით დადგენილი ყოველი მნიშვნელობისთვის უნდა განისაზღვროს დაცვის აუცილებლობის ღონისძიება.

მაგალითად, IT-სისტემის ამოვარდნამ შეიძლება გამოიწვიოს მაღალი ზარალი, ამიტომ დადგენილი მნიშვნელობა იქნება მაღალი, რადგან IT-სისტემას აქვს შესაბამისად დაცვის მაღალი დონე. ამგვარად, პირველ რიგში, უნდა განისაზღვროს დაცვის მოთხოვნები ბიზნესპროცესებისთვის. შემდეგ, აქედან გამომდინარე, დადგინდება დაცვის მოთხოვნები აპლიკაციებისათვის, რომლებიც სტრუქტურული ანალიზით გამოვლინდა. ამ დროს გათვალიწინებულ უნდა იქნას, რომელი ინფორმაციები მუშავდება ამ აპლიკაციებით.

უმეტეს დაწესებულებებში ამ პოზიციაზე მოიაზრება საკმარისად მცირე საინფორმაციო ჯგუფები. მაგალითად, თუ ეს მოიცავს კლიენტთა მონაცემებს, სართო წვდომის ინფორმაციას (მაგალითად, მისამართები, ღია სამუშაო საათები) ან სტრატეგიულ მონაცემებს ბიზნესის მართვისათვის. შემდეგ განიხილება რომელი ინფორმაცია სად და რომელი IT-სისტემით მუშავდება, რათა შესაძლებელი იყოს ბიზნესპროცესების შესრულება.

აპლიკაციათა დაცვის მოთხოვნილება გადაიტანება IT-სისტემებზე, რომლებიც შესაბამის დანართებს უჭერენ მხარს. ფართების დაცვის მოთხოვნილება გამომდინარეობს აქ განთავსებული აპლიკაციებისა და IT-სისტემების დაცვის მოთხოვნილებიდან.

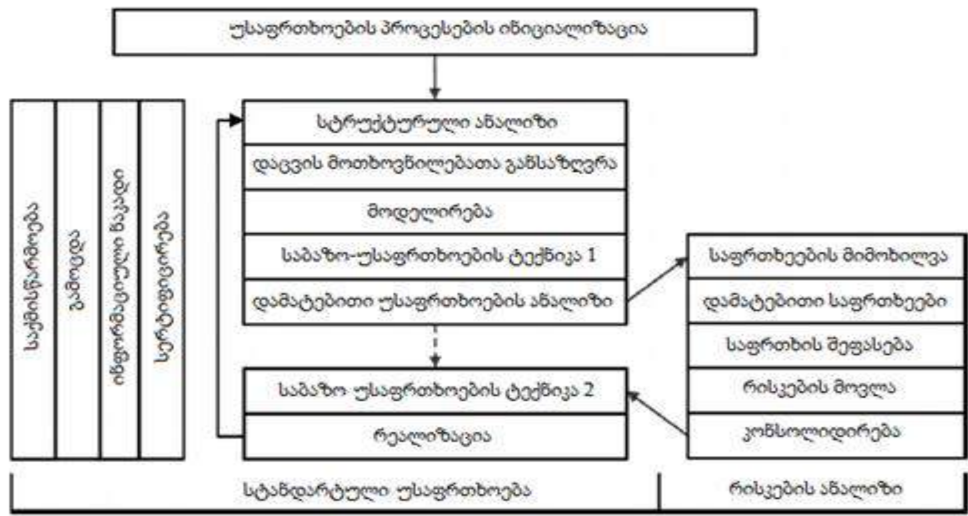
მაგალითად, კლიენტთა მონაცემების დამუშავების ბიზნესპროცესს აქვს დიდი მნიშვნელობა საწარმოო ოპერაციების მხარდასაჭერად. ეს ბიზნესპროცესი მუშაობს S-სერვერზე, რომელსაც აქვს მაღალი დაცვის მოთხოვნილება. სათავსოს, რომელშიც სერვერია განთავსებული, ექნება მინიმუმ მაღალი დაცვის მოთხოვნილება.

- **უსაფრთხოების დამატებითი ანალიზი**

IT-საბაზო დაცვის მეთოდების გამოყენება იძლევა შესაძლებლობას უსაფრთხოების ისეთი დონე იყოს უზრუნველყოფილი, რომელიც იქნება საკმარისი და მისაღები ნორმალური დაცვის მოთხოვნებისთვის. თუ დაცვის მოთხოვნილება განსაზღვრული სფეროსთვის (მაგალითად, აპლიკაციისთვის ან IT-სისტემისათვის) უფრო მაღალია, ან ამ სფეროსთვის არ არსებობს IT-დაცვის ზომები, მაშინ აუცილებელია IT-საბაზო დაცვის დანერგვით დამატებითი უსაფრთხოების ანალიზის ჩატარება.

BSI-ს აქვს დამუშავებული საკუთარი მეთოდი რისკების ანალიზისთვის, რომელიც ეფუძნება IT-საბაზო დაცვის დანერგვას. იგი აღიწერება BSI-Standard 100-3 წყაროში [31]. მეთოდის სახით შესაძლებელია ასევე კლასიკური რაოდენობრივი რისკების ანალიზის არჩევა განსახილველი სფეროსთვის. თუ განიხილება ინფორმაციის გადამუშავების მხოლოდ მცირე სფერო, მაშინ დანახაჯები დამატებითი რისკების ანალიზისათვის არაა მაღალი.

სტანდარტული უსაფრთხოების ღონისძიებებისა და რისკების ანალიზის კომბინაცია სფეროებისათვის, რომელთა დაცვის მოთხოვნილებები ნორმალურზე მაღალია, იგი უფრო ეფექტურია, ვიდრე მხოლოდ რაოდენობრივი რისკების ანალიზის გამოყენება (ნახ.6.7).



ნახ.6.7. რისკების ანალიზის ინტეგრაცია უსაფრთხოების პროცესში

6.11. უსაფრთხოების კონცეფციის დანერგვა

IT-საბაზო დაცვის კატალოგები შეიცავს ტიპურ სტრუქტურული ელემენტების (ბლოკების), საფრთხეებისა და ღონისძიებების კატალოგებს. სტრუქტურულ ბლოკებში აღწერილია ინფორმაციის უსაფრთხოების მენეჯმენტის ტიპური ამოცანები და საფრთხეებისა და სტანდარტული უსაფრთხოების ღონისძიებების IT-გამოყენების სფეროები. ამავდროულად განიხილება ინფორმაციული უსაფრთხოების ორგანიზაციული, პერსონალური, ინფრასტრუქტურული და ტექნიკური ასპექტები.

IT-საბაზო დაცვი კატალოგები შეიცავს ბლოკებს შემდეგი სფეროებიდან:

- ინფორმაციული უსაფრთხოების ძირითადი ასპექტები (მაგალითად, ორგანიზაციის, პერსონალის, საგანგებო მზადყოფნის);
- ინფრასტრუქტურის უსაფრთხოება (შენობები, გამოთვლითი ცენტრი და სხვ.);
- IT-სისტემის უსაფრთხოება (სერვერი, კლიენტი, ქსელის კომპონენტები);
- ქსელის უსაფრთხოება (ქსელის და სისტემის მენეჯმენტი);
- აპლიკაციათა უსაფრთხოება (მაგალითად, ი-მაილები).

სტრუქტურული ანალიზის შემდეგ შესაძლებელია ბიზნესპროცესების მოდელირება ამ სტრუქტურული ბლოკების დახმარებით. აქ განხილული განსაზღვრის სფეროსთვის განსაზღვრულ იქნება IT-საბაზო დაცვის შესაბამისი ბლოკების ნაკრები (საინფორმაციო ქსელი). აქედან გამომდინარეობს რეკომენდებულ ღონისძიებათა ნაკრები, რომელიც შეიძლება მოიაზრებოდეს როგორც უსაფრთხოების კონცეფციის საფუძველი.

IT-საბაზო დაცვის კატალოგებში შემავალი ღონისძიებების სახით განიხილება კონკრეტული რეალიზაციის დახმარებები გენერირებული მოთხოვნილებებისთვის როგორც ISO 27001 ან ISO 27002 სტანდარტებიდან, ისე მრავალრიცხოვანი ტექნიკური ღონისძიებებიდან საიმედო წარმოებისთვის ტიპური IT-სისტემებისა და აპლიკაციებისათვის.

დეტალური სახელმძღვანელო სტრუქტურული ბლოკების შესარჩევად (მოდელირება საბაზო დაცვის მიხედვით) გვეხმარება უსაფრთხოებასთან დაკავშირებული ასპექტების გათვალისწინებაში. ასეთი დახმარებით სახელმწიფო ორგანიზაციებს ან კერძო ბიზნესს შეუძლია სასურველი მიზნების მიღწევა უსაფრთხოების სფეროში, გარე კონსულტანტების გარეშე ან მათი მცირე დახმარებით.

VII თავი
ინფორმაციული ტექნოლოგიების ინფრასტრუქტურის ბიბლიოთეკა (ITIL)

7.1. შესავალი ITIL-ში: ძირითადი ცნებები და ტერმინები

ITIL - Information Technology Infrastructure Library არის ინფორმაციული ტექნოლოგიების ინფრასტრუქტურის ბიბლიოთეკა. იგი დღეისათვის ძალზე აქტუალური და საყოველთაოდ ცნობილი ცოდნის ბაზაა სერვისების მართვის სფეროში მთელი მსოფლიოს მასშტაბით [3,11,38]. ის ასახავს IT–სფეროს მსოფლიოს წამყვანი პრაქტიკოსების ფუნდამენტურ საფუძვლებს. ევროპაში არსებობს ITIL სერტიფიცირების ორი ცენტრი: EXIN – ჰოლანდიის საგამოცდო ინსტიტუტი და ISEB (Information Systems Examination Board)-ბრიტანეთის კომპიუტერული საზოგადოების განყოფილება [39,40].

ITIL განიხილავს სერვისების მართვას ურთიერმოქმედების კონტექსტში: „სერვისების მიმწოდებელი– სერვისების დამკვეთი“.

დამკვეთი (Customer) - ესაა საქონლის ან მომსახურების მყიდველი. IT–სერვისების მიმწოდებლისთვის დამკვეთი არის ადამიანი (ან ჯგუფი), რომელიც აფორმებს შეთანხმებას მიმწოდებელთან IT–მომსახურების მისაღებად და პასუხს აგებს მიღებული მომსახურების ასანაზღაურებლად.

მიმწოდებელი (Service provider) - ესაა ორგანიზაცია, რომელიც აწვდის სერვისს ერთ ან რამდენიმე შიგა ან გარე დამკვეთს.

მომხმარებელი – ესაა IT –სერვისების გამომყენებელი თანამშრომელი დამკვეთ ორგანიზაციაში.

IT-მომსახურება (სერვისი) – დამკვეთებისთვის ფასეულობის მიწოდების ხერხი, რომელთა საშუალებითაც ისინი იღებენ გამოსასვლელზე საჭირო შედეგებს მათთვის სპეციფური დანახარჯებისა და რისკების გარეშე.

შეიძლება განვიხილოთ სხაგვარი განსაზღვრებაც. IT–მომსახურება – ესაა ერთი ან მეტი ტექნიკური ან პროფესიონალური შესაძლებლობა, რომელიც ხელს უწყობს ბიზნესპროცესს. ტერმინები „სერვისი“ და „მომსახურება“ ეკვივალენტურია. მათ აქვს შემდეგი მახასიათებლები:

- აკმაყოფილებს დამკვეთის ერთ ან მეტ მოთხოვნას;
- მხარს უჭერს დამკვეთის ბიზნესმიზნებს;
- დამკვეთისგან აღიქმება როგორც ერთი მთლიანი პროდუქტი, რომელიც მზადაა გამოსაყენებლად.

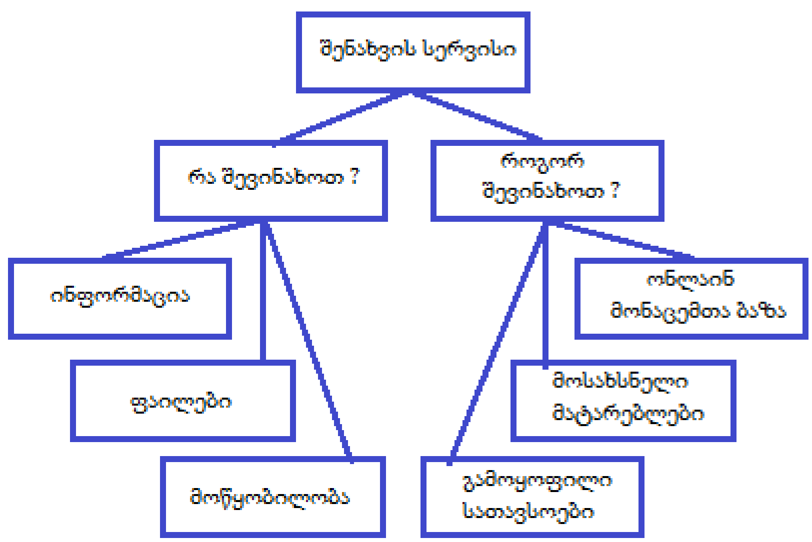
განვიხილოთ დეტალურად ძირითადი ცნებები სერვისის განსაზღვრებაში.

შედეგები გამოსასვლელზე (outcomes) – ის რასაც იღებს დამკვეთი საბოლოო ჯამში. ცხადია, რომ იგი განსხვავდება დამკვეთის საწყისი მოთხოვნებისაგან, გარკვეული

შემზღუდველი ფაქტორების არსებობის გამო. სერვისის დანიშნულებაა ამ ფაქტორების შემცირების და მწარმოებლურობის ამაღლების გზით გამოსასვლელი შედეგების გაუმჯობესება. სერვისების გამოყენების შედეგია გამოსასვლელზე სასურველი შედეგების მიღების ალბათობის გაზრდა.

მომსახურების მოდელები, რომელთაც ITIL გვთავაზობს, გვეხმარება IT–სფეროს სირთულეების, ხარჯების, მოქნილობისა და მრავალსახეობის მართვაში. ყოველ მოდელს აქვს გამოყენების ვარიანტების სიმრავლე კონკრეტული შემთხვევისგან დამოკიდებულებაში, რაც მისი გამოყენების იდეას ხდის უნივერსალურს, მოქნილს და ეფექტურს.

IT–სერვისის მოდელი შეიძლება განვიხილოთ ინფორმაციის შენახვის სისტემის მაგალითზე. სისტემა დანიშნულია ინფორმაციის შენახვის, მოწესრიგებისa და დაცვის განსახორციელებლად რაიმე სამუშაოს ან მოქმედების კონტექსტში. თუ მიმწოდებელი აძლევს დამკვეთს არა მხოლოდ დამხსომებელ მოწყობილობას, არამედ ინფორმაციის შენახვის სერვისსაც, მაშინ უნდა გაეცეს პასუხი კითხვებს „რა შევინახოთ“ და „როგორ შევინახოთ“ (ნახ.7.1). ამასთანავე პრინციპულად მნიშვნელოვანია მოვალეობებისა და პასუხისმგებლობების განაწილება მიმწოდებელსა და დამკვეთს შორის.



ნახ.7.1. ინფორმაციის შენახვის სისტემის სქემა

დამკვეთებს სურთ სასურველი შედეგების მიღება, მაგრამ სხვადასხვა მიზეზთა გამო, არ სურთ თანმხლები პასუხისმგებლობა, ხარჯები და რისკები. მაგალითად, ორგანიზაციას უნდა დაცული ინფორმაციის შენახვის სისტემის შექმნა რამდენიმე ტერაბაიტით ონლაინ-ვაჭრობის მხარდასაჭერად.

ასეთი სისტემის შესაქმნელად „ნულიდან“ ამ ორგანიზაციამ უნდა განვლოს გრძელი გზა, დაწყებული იმის გაგებით, თუ როგორ გააკეთოს, დამთავრებული ძვირადღირებული ტექნიკის შესყიდვით და კვალიფიციური პერსონალის დაქირავებით. ეს კი მეტად ძვირადღირებული სიამოვნება და დროის დიდი დანახარჯია.

შედარებით მარტივია ამ შემთხვევაში მიმწოდებლის სერვისების გამოყენება, რომელიც უკვე ფლობს ინფორმაციის შენახვის დიდ სისტემას და აქვს შესაბამისი გამოცდილება და შესაძლებლობები. ეს იქნება ინფორმაციის დაცული შენახვის სერვისის შეთავაზება.

სერვისის ფასი (value) – იგი იზომება ორი ცნების კონტექსტში:

- სერვისის სარგებლობა (Service Utility) – არის ის, რასაც იღებს დამკვეთი სერვისის გამოყენებით;
- სერვისის ხარისხის გარანტია (Service Warranty) – არის ის, თუ როგორ აძლევს მიმწოდებელი დამკვეთს სერვისს – წვდომის, მწარმოებლურობისა და უსაფრთხოების ტერმინებში.

➢ **ITILv3 –ის ლექსიკონის განსაზღვრებანი:**

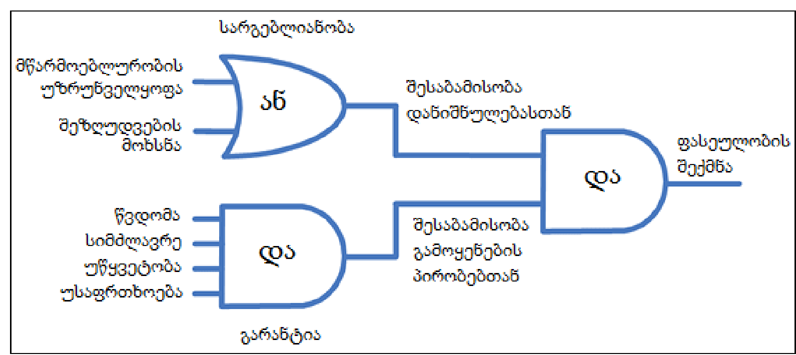
სარგებლიანობა – ფუნქციურობა, რომელსაც იძლევა პროდუქტი ან სერვისი განსაზღვრულ მოთხოვნილებათა უზრუნველსაყოფად. ხშირად განისაზღვრება როგორც „რას აკეთებს პროდუქტი/სერვისი“.

სერვისის სარგებლიანობა – IT-მომსახურების ფუნქციურობა დამკვეთის თვალსაზრისით.

გარანტია – დაპირება იმისა, რომ პროდუქტი ან სერვისი დააკმაყოფილებს მოთხოვნებს.

მომსახურების ხარისხის გარანტია – დარწმუნება იმაში, რომ IT–სერვისი შესაბამისი იქნება შეთანხმებულ მოთხოვნებთან. შესაძლებელია ფორმალური შეთანხმების არსებობა, ხელშეკრულება, ან როგორც მარკეტინგული შეტყობინება.

ამგვარად, სარგებლიანობა – ისაა, რასაც დამკვეთი იღებს, ხარისხის გარანტია – ის, თუ როგორ იღებს (ნახ.7.2).



ნახ.7.2. სერვისის ფასეულობის ფორმირების სქემა

როდესაც დამკვეთი, შეიძენს რა სერვისს, უნდა შედეგის მიღება მისივე გამოყენებით, ანუ ფასეულობის ამოღება.

სარგებლიანობა მიიღწევა ერთ–ერთი ხერხით:

1. დამკვეთის მიერ მოთხოვნილი მწარმოებლურობის უზრუნველყოფით;
2. არსებული შეზღუდვების მოცილებით ან შემცირებით.

მწარმოებლურობა (Performance) - შეფასებაა იმის, რაც იქნა მიღწეული ან შემუშავებული სისტემის, ადამიანის, გუნდის, პროცესის ან IT-სერვისის მიერ.

მწარმოებლურობის ქვეშ იგულისხმება დამკვეთის შესაძლებლობა გააკეთოს მეტი ნაკლებ დროში, ნაკლები დანახარჯებით, ანუ ნაკლები რესურსების გამოყენებით. სხვა სიტყვებით, ესაა გარკვეული ოპტიმიზაცია, რომელიც უზრუნველყოფს დამკვეთს გადაწყვიტოს ამოცანა ნაკლები დროის და ფულის გამოყენებით [11].

შეზღუდვა არის აკრძალვა ან რაღაც ქმედებათა შესრულების შეუძლებლობა. გარანტია შედგება ოთხი ძირითადი ასპექტისაგან:

- წვდომა;
- სიმძლავრე;
- უსაფრთხოება;
- უწყვეტობა.

სერვისის ხარისხის გარანტიის შეფასება უფრო მარტივია, ვიდრე მისი სარგებლიანობისა ბიზნესისათვის. როცა ადამიანი აჭერს ღილაკს, ის ელოდება, რომ აინთება სინათლე. სამწუხაროდ, IT-სერვისების დროს არც ასე მარტივადაა საქმე. IT-სერვისის გამოყენების შედეგი დამოკიდებულია არა მხოლოდ სერვისის თვისებებზე, არამედ ამ სერვისის მართვაზეც. სწორედ აქ ჩნდება ტერმინი service management.

IT სერვისების მართვა სპეციალიზებული ორგანიზაციული შესაძლებლობების ერთობლიობაა დამკვეთისათვის ფასეულობის მისაწოდებლად სერვისის ფორმაში [11]. „სპეციალიზებული შესაძლებლობათა" ქვეშ იგულისხმება პროცესები, მეთოდები, ფუნქციები და როლები, რომელთა გამოყენება შეუძლია მიმწოდებელს, დამკვეთიასთვის სერვისის მიწოდების მიზნით. გამოიყენება ასევე აღნიშვნა **ITSM** (IT Service Management), რომელიც ეკვივალენტურია „სერვისების მართვის".

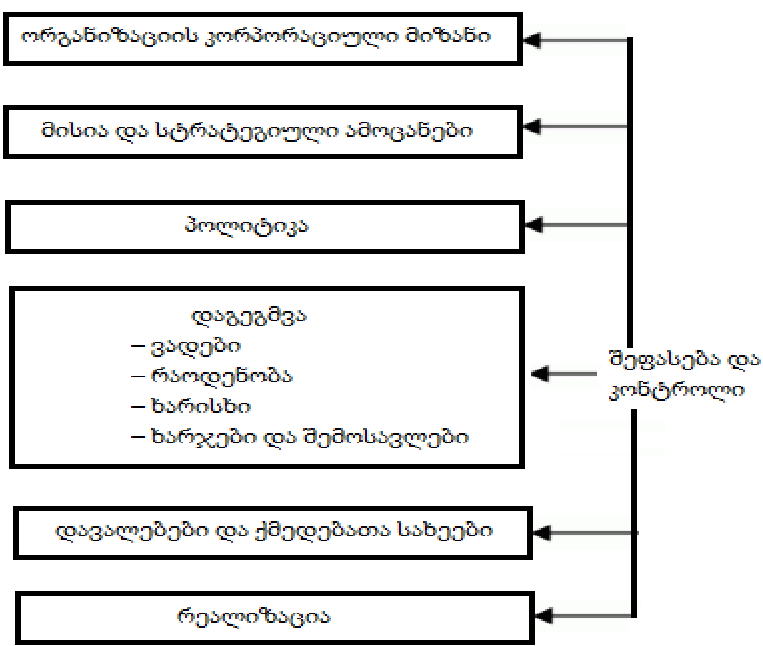
ხარისხი – ობიექტის მახასიათებელთა ერთობლიობაა, რომელიც მიეკუთვნება მის შესაძლებლობას, რათა დააკმაყოფილოს დადგენილი და შემოთავაზებული მოთხოვნები.

ორგანიზაციას შეუძლია ძალზე ძვირი IT–სერვისის ყიდვა, მაგრამ თუ მიმწოდებელს არ შეუძლია ხარისხიანი და საპასუხისმგებლო მართვის უზრუნველყოფა – მაშინ ეს შესყიდვა იქნება უაზრო. დამკვეთის დაკმაყოფილება ბევრადაა დამოკიდებული იმაზე, თუ რამდენად სწორად იქნა შეთანხმებული სერვისის პარამეტრები წინასწარ სერვისის მიმწოდებელთან.

ამგვარად, სერვის მენეჯმენტის ძირითადი მიზანი ITIL კონტექსტში არის დამკვეთებისადმი საიმედო, სტაბილური IT–სერვისების მიწოდება, რომლებიც სრულად დააკმაყოფილებს მათ მოთხოვნებს მოცემულ სფეროში. ერთ–ერთი საკვანძო ტერმინი ITIL-ში არის „ორგანიზაცია". IT–სერვისის დამკვეთი და IT-სევრისის მიმწოდებელი განიხილება როგოორც ორგანიზაციები.

ორგანიზაცია – ესაა ადამიანთა თანამშრომლობის განსაზღვრული ფორმა. და ისმის კითხვა, რაში მდგომარეობს მიზანი ორგანიზაციად გაერთიანებისა ასეთი კორპორატიული მიზნი (vision) შეიძლება იყოს, მაგალითად, ფულის გამომუშავების სურვილი, პერსონალური კომპიუტერების გაყიდვით, ან სერვისის შეთავაზება ინტერნეტში ჩასართავად. იმისათვის, რომ ორგანიზაცია იყოს მიმზიდველი დამკვეთების, ინვესტორების და კომპანიის თანამშრომლებიასთვის, საჭიროა ინფორმაციის მიწოდება, თუ რა უპირატესობა ექნება მათ ამ ორგანიზაციასთან თანამშრომლობისას.

კორპორატიული მიზნის გასაცნობად კომპანიას შეუძლია მისი წარმოდგენა თეზისის სახით თავის მისიაზე (mission) (ნახ.7.3).



ნახ.17.3. ორგანიზაციის კორპორაციული მიზნის ფორმირება

მისია – ესაა ამოცანების მოკლე და ცხადად აღწერა, რომელიც დგას ორგანიზაციის წინაშე და ის იდეალები, რომლებისაც მას (ორგანიზაციას) სწამს.

სტრატეგიული ამოცანები (objectives) – ესაა სრული აღწერა იმისა, რასაც უნდა მიაღწიოს ორგანიზაციამ გრძელვადიანი პერსპექტივით. კარგად ფორმულირებული სტრატეგიული ამოცანები უნდა ფლობდეს ხუთ ძირითად თვისებას (შეესაბამებოდეს SMART პრინციპს):

- იყოს კონკრეტული (Specific);
- ექვემდებარებოდეს შეფასებას (Measurable);
- იყოს სიტუაციისადმი შესაფერისი და შესაბამისი (Appropriate);
- იყოს რეალისტური (Realistic);
- ჰქონდეს მკაფიო დროითი საზღვრები (Time-bound).

ორგანიზაციის პოლიტიკა (policy) – ესაა გადაწყვეტილებების და ზომების ერთობლიობა, მიღებული ორგანიზაციის მიერ სტრატეგიული ამოცანების დასასმელად და მათ გადასაწყვეტად.

ორგანიზაცია, თავისი პოლიტიკის შემუშავების დროს, განსაზღვრავს პრიორიტეტებს, რომლებიც მის წინაშეა სტრატეგიული ამოცანების და მათი გადაწყვეტის მიზნით. პრიორიტეტები შეიძლება შეიცვალოს დროის შესაბამისად. ზუსტად ჩამოყალიბებული კომპანიის პოლიტიკა (წესები) ხელს უწყობს ორგანიზაციის სტრუქტურის მოქნილობას, რადგან კომპანიის ყველა დონეზე შესაძლებელია სიტუაციის ცვლილებებზე სწრაფი რეაგირება [11].

პოლიტიკის რეალიზაცია კონკრეტული სახის ქმედებებისთვის მოითხოვს სტრატეგიის შემუშავებას. სტრატეგია მუშავდება განსაზღვრული პერიოდებისათვის და შედგება რამდენიმე ეტაპისაგან. მნიშვნელოვანია აქ კონტროლის შესაძლებლობა სამუშაოთა შესრულებისას.

არსებობს სხვადასხვა მეთოდი. მაგალითად, ბიზნესში ცნობილია **ბალანსირებულ შეფასებათა რუქა (Balanced Score Card - BSC)**. ამ მეთოდის შესაბამისად, ორგანიზაციის სტრატეგიული მიზნების ან პროცესების მიზნების საფუძველზე განისაზღვრება წარმატების კრიტიკული ფაქტორები (Critical Success Factor - CSF).

წარმატების კრიტიკული ფაქტორები (Critical Success Factor - CSF) – ესაა ფაქტორები, რომლებიც აუცილებლად უნდა განხორციელდეს პროექტის, პროცესის, გეგმის ან სერვისის წარმატებისათვის. ასეთი ფაქტორები ფორმულირდება კომპანიის ინტერესების რამდენიმე უმნიშვნელოვანესი სფეროსათვის, რომელთაც უწოდებენ ორგანიზაციის პერსპექტივებს (პროექციებს): დამკვეთები / ბაზარი, ბიზნეს–პროცესები, პერსონალი / ინოვაციები და ფინანსები. რამდენად წარმატებით რეალიზდება CSFs, გამოიყენებენ KPI–ს.

მწარმოებლურობის გასაღებური მაჩვენებელი (Key Performance Indicator ან KPI) – ესაა მეტრიკა, რომელიც გამოიყენება პროცესების, სერვისის ან ქმედებების სამართავად [11]. შესაძლებელია ეფექტურობის მრავალი მაჩვენებლის შეფასება, მაგრამ განსაკუთრებით მნიშვნელოვანია მხოლოდ KPI.

მაგალითად, ფაქტორი „სერვისის დაცვა ცვლილებების რეალიზაციისას“ შეიძლება გაიზომოს ისეთი KPI–ით, როგორიცაა „არაწარმატებული ცვლილებების რაოდენობის შემცირება %–ში“, „პროცენტული შემცირება, ცვლილებათა რაოდენობის, რომელთაც მივყავართ ინციდენტების აღმოცენებამდე“ და ა.შ.

სხვადასხვა გარემოებათა ზემოქმედებისა და ეფექტურობის შეფასების შედეგებისგან ზემოქმედებით საკონტროლო წერტილებში, სტრატეგიული ამოცანები, მისიები და კორპორაციული მიზნები შეიძლება მნიშვნელოვნად შეიცვალოს.

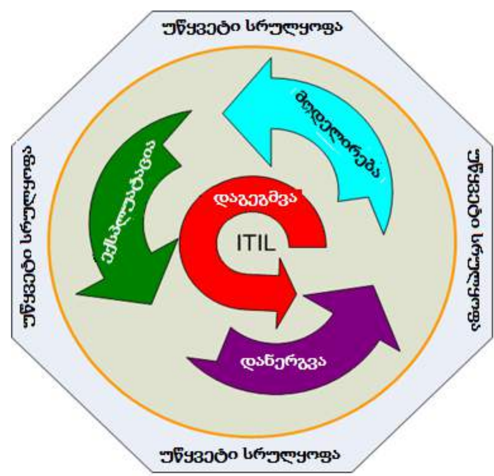
ამასთანავე IT-განყოფილების ან სერვისის მიმწოდებლების სტრატეგიული ამოცანები აგრეთვე უნდა შეიცვალოს ბიზნესის მიზნების მოთხოვნების შესაბამისად.

7.2. IT-სერვისის სასიცოცხლო ციკლი

ITILv3 საფუძველს შეადგენს შემდეგი ექვსი პუბლიკაცია (ანუ ბირთვი) [11]:

1. შესავალი ITIL – ში
2. სერვისის დაგეგმვა (Service Strategy)
3. სერვისის პროექტირება (Service Design)
4. სერვისის დანერგვა (Service Transition)
5. სერვისის ექსპლუატაცია (Service Operation)
6. სერვისის უწყვეტი სრულყოფა (Continual Service Improvement).

ხუთი წიგნი შეესაბამება სერვისების სასიცოცხლო ციკლის ეტაპებს (შესავლის გარდა): ბიზნესის მოთხოვნების პირველადი ანალიზიდან დაწყებული, სტრატეგიის აგებისა და პროექტირების ეტაპებზე, და დამთავრებული სერვისების სრულყოფით ექსპლუატაციის პროცესში. სერვისის სასიცოცხლო ციკლი მოცემულია 7.4 ნახაზზე.



ნახ.7.4. სერვისის სასიცოცხლო ციკლი

სერვისის დაგეგმვა (ან სტრატეგიის აგება) – ესაა სერვისის სასიცოცხლო ციკლის საფუძველი. მისი შესაბამისი პუბლიკაცია აღნიშნავს სერვის მენეჯმენტის ცნების ფუნდამენტურობას სერვისის სასიცოცხლო ციკლის კონტექსტში. წიგნში განიხილება შემდეგი საკითხები: IT-სერვისის ბაზრის განვითარება, სერვისების მიმწოდებელთა მახასიათებლები და ტიპები, სერვისის ძირითადი ხარისხები და რეალიზაციის სტრატეგია სასიცოცხლო პროცესის ციკლში. საკვანძო თემებია ასევე ფინანსური მართვა, მოთხოვნების მართვა, ორგანიზაციული განვითარება და სტრატეგიული რისკები.

მიმწოდებელმა უნდა გამოიყენოს სერვისის დაგეგმვის ეტაპი მიზნების დასასმელად, მომხმარებელთა და გასაღების ბაზრის მოლოდინის (სურვილების) გასარკვევად. სტრატეგიის აგების დანიშნულება, უპირველეს ყოვლისა, არის ის, რომ სერვისების მიმწოდებელმა შეაფასოს საკუთარი შესაძლებლობები და გადაწყვიტოს, შეძლებს თუ არა იგი განახორციელოს სერვისული პორტფელის მოთხოვნები ყველა ხარჯის და რისკის გათვალისწინებით.

სერვისების პორტფელი (ან პორტფოლიო) – ესაა სერვისების სრული ერთობლიობა, რომელიც წარმოდგინდება სერვისების მიმწოდებლის მიერ. პორტფელი გამოიყენება ყველა სერვისის მართვისათვის სასიცოცხლო ციკლის მთელ მანძილზე. იგი შეიცავს სამ კატეგორიას:

1) სერვისები დამუშავებაშია (Service Pipeline) – სერვისები , რომლებიც დამუშავების სტადიაშია:
2) სერვისების კატალოგი – უკვე გამოყენებადი ან შეთავაზებული სერვისების;
3) სერვისები, რომლებიც ამოღებულია ექსპლუატაციიდან (retired Services).

სერვისის დაპროექტება. ყოველი IT-სერვისისთვის ყველაზე მნიშვნელოვანია ბიზნესს წარუდგინოს გარკვეული სარგებელი ან ფასეულობა. ამიტომ მიმწოდებელმა უნდა გაითვალისწინოს ბიზნესის მიზნები.

პუბლიკაცია „სერვისების დაპროექტება" არის სახელმძღვანელო სერვისების მოდელირებისა და სრულყოფისთვის, ასევე რეკომენდაციებისათვის მათ სამართავად პრაქტიკაში. ამ ეტაპზე აღიწერება ძირითადი პრინციპები და მოდელირების მეთოდები სტრატეგიული მიზნების გარდაქმნისათვის განსაზღვრული ხარისხის კონკრეტული სერვისების ერთობლიობაში. იგი მოიცავს ასევე ახალი სერვისების შექმნის, არსებულის ცვლილებისა და სრულყოფის საკითხებს სასიცოცხლო ციკლის ფარგლებში, რაც აუცილებელია მის ფასეულობათა ასამაღლებლად მომხმარებელთა თვალსაზრისით.

წიგნის საკვანძო თემებია ასევე სერვისების კატალოგი, სარგებლიანობა, მწარმოებლურობა და სერვისის უწყვეტობა, სერვისების მართვის დონე, რომლებიც განიხილება შემდგომ.

სერვისის დანერგვა. Transition - გადაადგილება, გადასვლა ან ერთი მდგომარეობის შეცვლა მეორით (პოზიციის, პერიოდის, სტადიის, თემის და სხვა).. მისი შესაბამისი პუბლიკაცია ITIL ბიბლიოთეკაში არის სახელმძღვანელო იმაზე, თუ

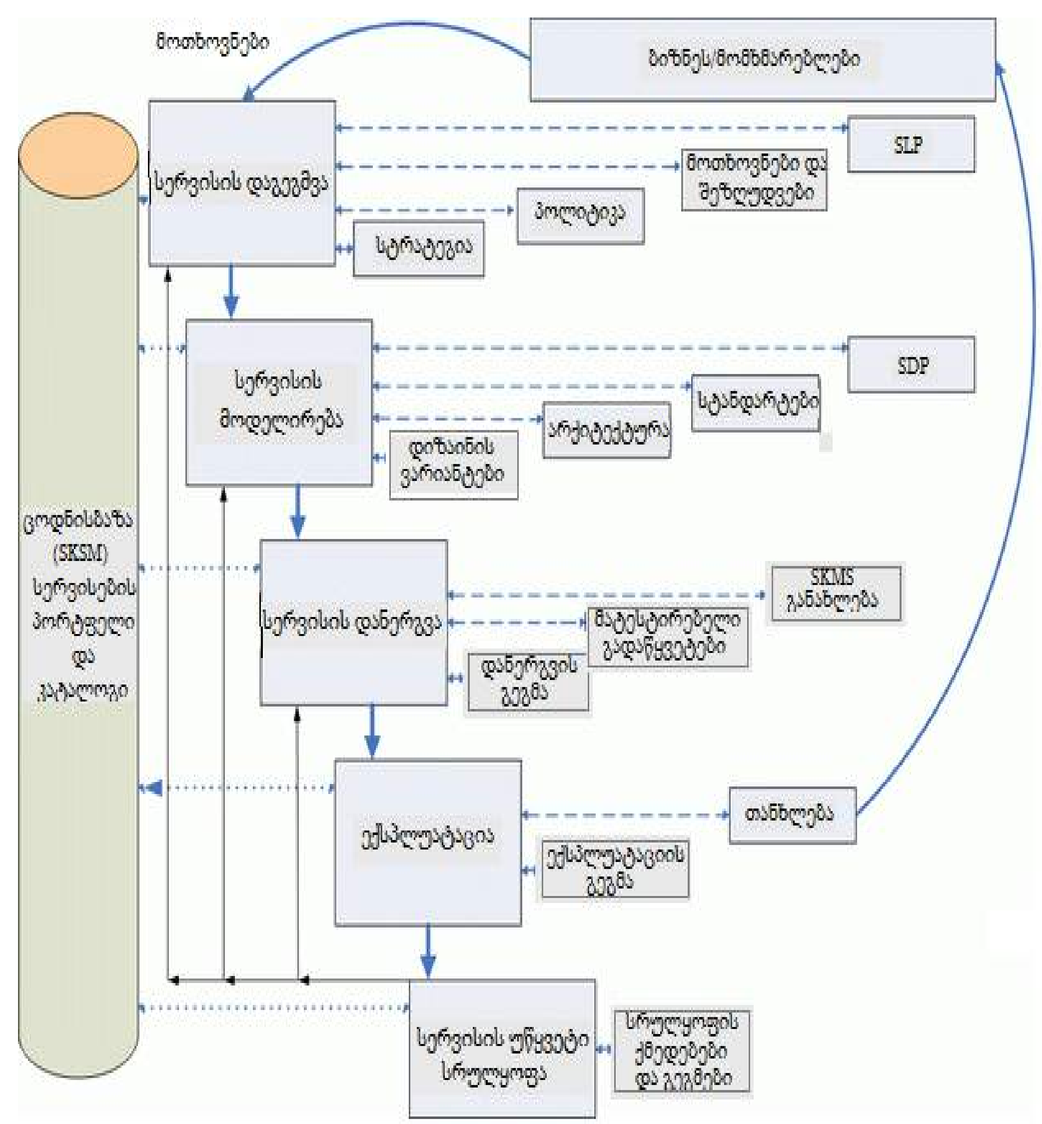
ეფექტურად როგორ რეალიზდეს მოთხოვნები, რომლებიც ფორმულირებულ იქნა პროექტირები,ს და სტრატეგიის აგების სტადიებზე, ექსპლუატაციის ეტაპზე რისკების, მტყუნებებისა და გაუმართაობების კონტროლით. განიხილება რისკების მართვის საკითხებიც.

სერვისის ექსპლუატაცია ახორციელებს სერვისის ბიზნეს–მნიშვნელობის „მიტანის" ეტაპს მიმწოდებლიდან დამკვეთამდე. აქ მნიშვნელოვანია სერვისის მიწოდების ეფექტურობა და მისი ხარისხიანი თანხლება. წიგნი აღწერს როგორ შეიძლება განხორციელდეს სერვისის სტაბილური ექსპლუატაცია, ცვლილების განხორციელების შესაძლებლობასთან ერთად დიზაინში, მასშტაბში, საზღვრებში და ა.შ. ორგანიზაციებს მიეცემათ ინსტრუქციები, მეთოდები და ინსტრუმენტები კონტროლის ორი მეთოდის სარეალიზაციოდ – პრევენციული (პროფილაქტიკური) და პროაქტიური.

წიგნში მოცემული ინფორმაცია სასარგებლო იქნება გადაწყვეტილების მისაღებად სერვისის წვდომის მართვის საკითხებში, სერვისზე მოთხოვნილების კონტროლისათვის, დატვირთვის ოპტიმიზაციის და მიმდინარე პრობლემების გადასაწყვეტად. აღწერილი ყველა ხერხი ითვალისწინებს ახალი მოდელების და არქიტექტურის შესაძლებლობებს, როგორიცაა განაწილებული სერვისები, გაანგარიშებები სქემით „კომუნალური სერვისი" (utility computing), ვებსერვისი და ელკომერცია. სიტყვა utility computing აღწერს ახალ შემოტანილ ბიზნესმოდელს, როცა სერვისების მიმწოდებელი იღებს ფულს სერვისის გამოყენების ფაქტზე, მაგალითად, მისი გამოყენების დროის მიხედვით. ტრადიციულ ბიზნესმოდელში კი მომხმარებელი იხდის სისტემის (სერვისის) ფლობისათვის. ასეთი სერვისების პროვაიდერს შეუძლია თავისი რესურსების გამოყენების ოპტიმიზაცია, მომხმარებელთა განსხვავებული საჭიროების გათვალისწინებით.

სერვისის უწყვეტი სრულყოფა მდგომარეობს სერვისის ფასეულობის ამაღლების მეთოდებისა და საშუალებების აღწერაში სასიცოცხლო ციკლის სხვადასხვა ეტაპზე სრულყოფის რეალიზაციის გზით. ეს ეტაპი აერთიანებს თავის თავში ხარისხის, ცვლილებებისა და მწარმოებლურობის სრულყოფის მართვის პრინციპებს, პრაქტიკებს და მეთოდებს. წიგნიდან ორგანიზაციებმა შეიძლება მიიღოს რეკომენდაციები იმის შესახებ. თუ ეტაპობრივად როგორ განახორციელონ მსხვილმასშტაბური სრულყოფები სერვისების ხარისხში, ექსპლუატაციის ეფექტურობასა და სერვისების მიწოდების უწყვეტობაში. სახელმძღვანელო დანიშნულია სრულყოფის შედეგების უკუკავშირის უზრუნველსაყოფად დაგეგმვის, მოდელირებისა და გარდაქმნების ეტაპებთან. ნახაზი 7.5 გვიჩვენებს თუ როგორაა დამოკიდებული სერვისის სასიცოცხლო ციკლის ეტაპები ბიზნესის მოთხოვნილებათა ცვლილებებზე.

მოთხოვნილებები იქმნება სერვისის დაგეგმვის ეტაპზე **სერვისების დონეების პაკეტი**ს ჩარჩოში **(Service Level Package ან SLP).** ესაა სარგებლიანობის განსაზღვრული დონე და გარანტიები ცალკეული სერვისების პაკეტისთვის. ყოველი SLP მუშავდება ცალკეული პროფილის ბიზნესქმედების მოთხოვნილებათა სარეალიზაციოდ.



ნახ.7.5. სერვისის სასიცოცხლო ციკლის ეტაპების ძირითადი კავშირები, შესასვლელები და გამოსასვლელები

ეს პროცესი გადადის სერვისის დაპროექტებაში, სადაც გადაწყვეტილებები, მიღებული პირველ ეტაპზე, გროვდება ერთად და რეალიზდება **სერვისის საპროექტო დოკუმენტაციის** სახით (Service Design Package ან SDP). ესაა დოკუმენტები, რომლებიც განსაზღვრავს სერვისის ყველა ასპექტს და მისდამი მოთხოვნებს სასიცოცხლო ციკლის ყოველ ეტაპზე [11].

ფაქტობრივად ესაა საპროექტო დოკუმენტაცია, რომელიც მუშავდება ახალი სერვისისთვის, მნიშვნელოვანი ცვლილებების შესატანად ან სერვისის ექსპლუატაციიდან მოხსნის დროს.

SDP გადადის დანერგვის ეტაპზე, რომელზეც ხდება სერვისის ტესტირება, გადის შეფასებას და ვალიდაციას. შედეგად განახლდება სერვისების ცოდნის მართვის სისტემა და სერვისი გადადის ექსპლუატაციის სტადიაზე.

სერვისის ცოდნის ბაზის მართვის სისტემა (Service Knowledge Management System ან SKMS) - ესაა ინსტრუმენტების და მონაცემთა ბაზების ერთობლიობა, რომლებიც გამოიყენება ცოდნის და სერვისების შესახებ ინფორმაციის სისტემატიზაციისთვის. იგი ინახავს, მართავს, განაახლებს და წარმოადგენს მთელ ინფორმაციას, რომელიც საჭიროა მიმწოდებლისთვის სერვისის მართვისათვის სასიცოცხლო ციკლის ყველა ეტაპზე.

ბუნებრივია, რომ სასიცოცხლო ციკლის მთელ მანძილზე სერვისი უნდა გაუმჯობესდეს, ამის აუცილებლობის შემთხვევაში და შესაბამისი შესაძლებლობების დროს.

7.3. სერვისების სტრატეგიის აგება

სერვისების თანამედროვე მსხვილი მიმწოდებლები მსგავსი მახასიათებლებით და შესაძლებლობებით ხასიათდებიან. მათ შორის მთავარი განმასხვავებელი თავისებურება სტრატეგიაა, რომელსაც კონკრეტული მიმწოდებელი იყენებს სერვისებისთვის.

სტრატეგიის აგების დროს სერვისის მიმწოდებელი ორიენტირებულ უნდა იყოს, უპირველეს ყოვლისა, თავისი პოტენციალური დამკვეთის მიზნებზე. ამიტომ ცხადად უნდა ესმოდეს რა როლი უნდა შეასრულოს მიწოდებულმა IT-სერვისმა დამკვეთის ბიზნესში.

IT-სფეროს სწრაფი განვითარება უკვე დღეს მოითხოვს მიმწოდებლებისასგან არა მხოლოდ დამკვეთების მოთხოვნებზე ოპერატიულ რეაგირებას, არამედ იმის ცოდნასაც, თუ მომავალში რა დასჭირდება დამკვეთს. ამიტომაც სტრატეგიის აგება არის ფუნდამენტური ეტაპი სერვისის სასიცოცხლო ციკლში. მიმწოდებელს უნდა ესმოდეს, რომ დამკვეთი მისგან ყიდულობს არა კონკრეტულ პროდუქტს, არამედ საშუალებებს თავიანთი ბიზნესმოთხოვნების დასაკმაყოფილებლად.

სტრატეგიის ასაგებად მიმწოდებელმა უნდა გაითვალისწინოს ფაქტორების სიმრავლე, რომელთაგან ძირითადია:

1. ყველაფერი, რაც IT-სერვისების ირგვლივაა, რთულია: ეს ეხება არა მხოლოდ კონკრეტულ სერვისების ინდივიდუალურ თავისებურებებს, არამედ იმ სირთულეებს, რომლებიც აღმოცენდება IT-სფეროში ცვალებადი და ურთიერთდამოუკიდებული ფაქტორების სიმრავლის შედეგად. საჭიროა განვასხვავოთ მოკლევადიანი და გრძელვადიანი დაგეგმვა, რადგან ბაზრის, მომხმარებელთა და თვით IT-სფეროს ქცევები განსხვავებულია განსახილველი პერიოდისაგან დამოკიდებულებით. პირველი რიგის

ამოცანად განიხილება მეთოდების შემუშავება, რომელიც დაეხმარება ორგანიზაციებს გადაწყვეტილების მისაღებად და შემდგომი ქმედებების სტრატეგიის განსაზღვრაში;

2. დამკვეთების მოთხოვნები ყოველთვის არაა ცხადი, გასაგები და კორექტულიც კი. მრავალი მათგანი იკარგება საპროექტო დოკუმენტაციიდან სერვისის რეალიზაციაზე გადასვლის პროცესში. სტრატეგიული აზროვნების ყველაზე მნიშვნელოვანი ასპექტია იმის გაცნობიერება, თუ რა უნდა იქნას მიღებული შედეგად. ის, რასაც დამკვეთი იღებს თავისი სერვისის ტექნიკური მოთხოვნების სანაცვლოდ, არის საფუძველი მისი დაგეგმვის. დამკვეთთა მოთხოვნების და მიზნების გაგება გვთავაზობს არა მხოლოდ ცოდნას თუ როდის და რატომ წარმოიშვა კონკრეტული მოთხოვნები, არამედ ცხადად გაცნობიერებასაც თუ ვინაა IT–სერვისის საბოლოო მომხმარებელი;

3. კონტექსტისაგან დამოუკიდებლად, რომელშიც მუშაობს მიმწოდებელი, სტრატეგიის აგების დროს მან უნდა გაითვალისწინოს კონკურენციის არსებობა. სახელმწიფო და კერძო IT–ორგანიზაციები მონაწილეობენ კონკურენციაში. სერვისების მიმწოდებლისთვის აუცილებელია ცოდნა რა მდგომარეობა უჭირავს მას ამ ბაზარზე, და მისი სერვისი რით განსხვავდება კონკურენტების ანალოგური სერვისებისაგან.

სერვისების დაგეგმვა, როგორც სასიცოცხლო ციკლის ეტაპი, საშუალებას აძლევს მიმწოდებელს გაერკვეს შემდეგ საკითხებში:

1. რომელი სერვისების შეთავაზება ღირს?
2. ვის უნდა შევთავაზოთ სერვისები?
3. რა სარგებელს (შედეგს) მიიღებენ მომხმარებლები სერვისისგამოყენებით?
4. რა სარგებელს (შედეგს) მიიღებენ ინვესტორები სერვისის გამოყენებით?
5. როგორ განვითარდეს შიგა და გარე გასაღების ბაზრები?
6. როგორ განისაზღვროს სერვისის ხარისხი?
7. როგორ იღებენ გადაწყვეტილებას დამკვეთები სერვისების მიმწოდებლების ამორჩევისას კონკურენციის პირობებში?
8. როგორ გაკონტროლდეს სერვისის ფასეულობის შექმნა ფინანსური მართვის ტერმინებში?
9. როგორ განაწილდეს არსებული რესურსები დასახული მიზნების უფრო ეფექტურად მისაღწევად?

მომხმარებლები აფასებენ IT–სერვისის გამოყენების შედეგებს ყველაზე ხშირად ეკონომიკური ტერმინებით. IT–ორგანიზაციისთვის აუცილებელია ფიქრი როგორც ინვესტიციებზე სერვისების განვითარების მიზნით, ისე ბიზნესზე მათი დანერგვის გზით.

სერვისისთვის მნიშვნელოვანია საბაზრო ადეკვატური ფასი, მწარმოებლურობა, სტაბილურობა (დამკვეთი ითვალისწინებს ამათ). სერვისმენეჯმენტის წარმატება დამოკიდებულია უპირველეს ყოვლისა დამკვეთის და მწარმოებლის ურთიერგაგებაზე.

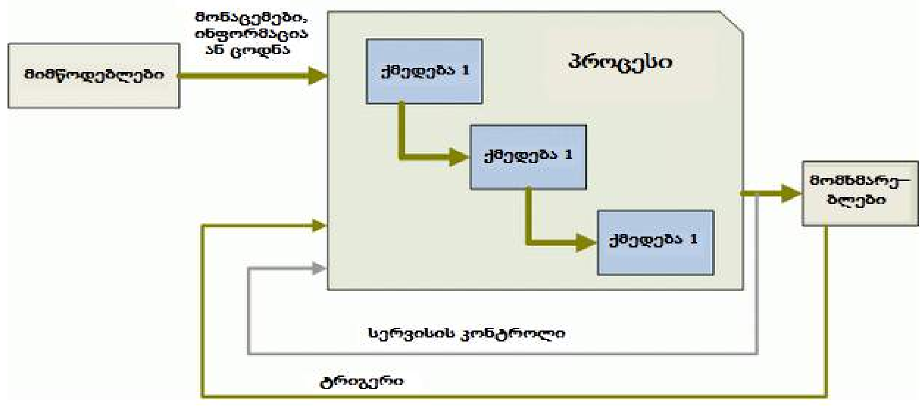
ამიტომაც, წარმატების მიღწევის მიზნით (სერვისების ასაგებად), არის შემოთავაზებული ITIL პუბლიკაციები.

პუბლიკაციაში "ITILv3. სერვისების სტრატეგიის აგება" განისაზღვრება და ფართოდ გამოიყენება ფუნქციებისა და პროცესების ცნებები სერვისის სასიცოცხლო ციკლში.

ფუნქციები (Functions) – ორგანიზაციის ნაწილია, სპეციალიზებული იმისათვის, რომ შესრულდეს განსაზღვრული სახის სამუშაოები და პასუხი გაეცეს შესაბამისი შედეგების ფორმირებას. ფუნქციებს აქვს სამუშაოების შესასრულებლად ყველა აუცილებელი შესაძლებლობა და რესურსი. შესაძლებლობები მოიცავს სამუშაოს საკუთარ მეთოდებს და დაგროვებულ გამოცდილებას. ფუნქციები უზრუნველყოფს ორგანიზაციის სტრუქტურირებას და სტაბილობას [11].

ფუნქციები განსაზღვრავს პასუხისმგებლობას, უფლებებს და როლებს დასმული მიზნების მისაღწევად. ფუნქციათა კოორდინაცია ზოგადი პროცესების საშუალებით არის ნებისმიერი ორგანიზაციის აგების განუყოფელი ნაწილი. საყურადღებოა, რომ ფუნქციები – ეს არაა ყოველთვის განყოფილებები, ანუ „ერთი ფუნქცია – ერთი განყოფილება“ არაა ჭეშმარიტი. მაგალითად, ITILv3–ში გაჩნდა ისეთი ფუნქციები, როგორიცაა Technical Management, Applications Management, რაც მიუთითებს პროფესიონალურ კომპეტენციაზე (ინჟინრები და ადმინისტრატორები), და არ შეიძლება იყოს განყოფილების დასახელება.

პროცესი – ქმედებათა სახეების სტრუქტურირებული ერთობლიობაა, რომელიც დაპროექტებულია განსაზღვრული მიზნის მისაღწევად. პროცესი შეიძლება შეიცავდეს როლს, პასუხისმგებლობას, ინსტრუმენტებს და კონტროლის მეთოდებს, რომლებიც აუცილებელია შედეგების ფორმირებისათვის. პროცესს შეუძლია განსაზღვროს პოლიტიკები, სტანდარტები, ხელმძღვანელობა, ქმედებათა სახეები და სამუშაო ინსტრუქციები (ნახ.7.6).



ნახ.7.6. საბაზო პროცესის სქემა

პროცესების ძირითადი მახასიათებლებია აქტიურობა, მიმდევრობითობა, ერთი მეორეზე დამოკიდებულება. ტერმინი „ატიურობა" ფართოდ გამოიყენება ITIL–ში. **აქტიურობა** – ესაა ქმედებათა ერთობლიობა, დაპროექტებული განსაზღვრული შედეგის მიღების მიზნით. პროცესებს აქვს შემდეგი მახასიათებლები:

1. პროცესები გაზომვადია, ანუ შესაძლებელია პროცესის გაზომვა (შეფასება) რომელიმე შესატყვისი მეთოდით. მენეჯერები ცდილობენ თავდაპირველად გაზომონ ფასი და ხარისხი, ხოლო პრაქტიკოსი მომხმარებლები – პროცესის ხანგრძლივობა და პროდუქტიულობა;

2. პროცესები ემსახურება კონკრეტული შედეგების მიღწევას. პროცესის არსებობის მიზეზია კონკრეტული შედეგის წარმოდგენა, რომელიც შეიძლება იდენტიფიცირდეს და დათვლილ იქნას;

3. პროცესებს ჰყავს მომხმარებლები – ყოველი პროცესი აწვდის თავის შედეგებს მომხმარებლებს ან ინვესტორებს. ისინი შეიძლება იყოს ორგანიზაციის შიგნით ან გარეთ, ოღონდ პროცესები ყველა შემთხვევაში უნდა აკმაყოფილებდეს მოსალოდნელ შედეგებს;

4. პროცესები პასუხს აგებს განსაზღვრულ შედეგებზე;

5. პროცესები რეაგირებას უნდა ახდენდეს განსაზღვრულ მოვლენებზე. სანამ მიმდინარეობს პროცესი, იგი კავშირში უნდა იყოს სპეციალურ საინიციალიზაციო ტრიგერთან.

ფუნქციისა და პროცესის ცნებებს ხშირად ურევენ. შეცდომის მიზეზი ხშირად არის აზრი, რომ თუ შედეგის დათვლა შეიძლება, მაშინ ის პროცესია.

მაგალითად, არებობს მცდარი აზრი, რომ დატვირთვის მართვა არის პროცესი ITSM. ჯერ ერთი, დატვირთვის მართვა – ესაა ორგანიზაციის შესაძლებლობა თავისი შიგა პროცესებით და მეთოდებით.

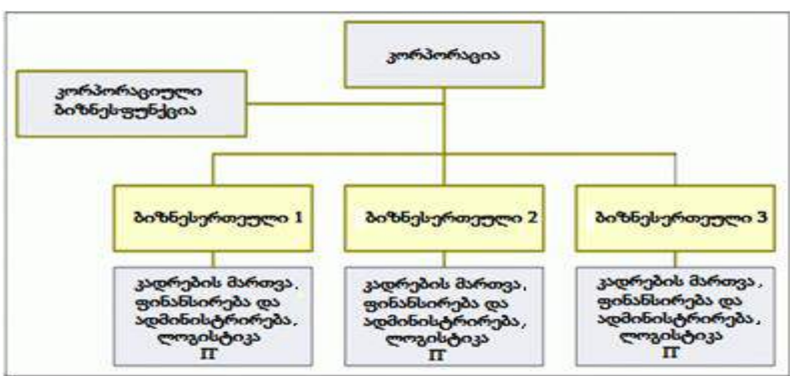
7.4. სერვისების მიმწოდებელთა ტიპები

სერვისების მიმწოდებლები ორგანიზაციასთან მიმართებით შეიძლება იყოს შიგა ან გარე. ITILv3–ში განიხილება სამი ტიპის მიმწოდებელი [11]:

- **ტიპი 1**

ITILv3–ში ფართოდ გამოიყენება ცნება **ბიზნესერთეული (business unit ან BU).** იგი ბიზნესის სეგმენტია, რომელსაც აქვს თავისი საკუთარი მეტრიკები, გეგმები, შემოსავლები და ხარჯები. თითოეული ბიზნეს–ერთეული ფლობს და მართავს აქტივებს, რომლებიც გამოიყენება საქონლის და სერვისების შესაქმნელად განსაზღვრული ფასეულობით.

ბიზნესერთეული არის ორგანიზაციული ერთეული და შეიძლება იყოს კორპორაციის ნაწილი ან სხვა ორგანიზაცია. პირველი ტიპის მიმწოდებლები (ნახ.7.7) მიმაგრებულია იმ ბიზნეს–ერთეულებზე, რომლებსაც ისინი ემსახურებიან და ფინანსირდებიან მისი ბიუჯეტიდან.



ნახ.7.7. სერვისების 1–ელი ტიპის მიმწოდებლები

ისინი პირდაპირ ექვემდებარება ბიზნესს, ხოლო საკვანძო გადაწყვეტილებებს (სერვისების პორტფელის განსაზღვრა, შედეგების შეფასების კრიტერიუმები, ინვესტიციის მოცულობები) იღებენ ორგანიზაციის ტოპმენეჯერები.

სერვისების პირველი ტიპის მიმწოდებელთა ძირითადი მიზანია ფუნქციური მთლიანობის და ეფექტიანობის უზრუნველყოფა ბიზნესერთეულისთვის, რომელთანაც ისინი არიან მიმაგრებული. ანუ, ისინი აწვდიან მას IT–სერვისებს ბიზნესის ვიწრო წრის მოთხოვნილებათა დასაკმაყოფილებლად. ამ ტიპის მიმწოდებელთა წარმატება არ იზომება ეკონომიკურ ტერმინებში, ვინაიდან მათი ძირითადი მიზანი არაა მოგების მიღება, ესაა მხოლოდ აუცილებელი სერვისების მიწოდება კონკრეტულ ბიზნეს-ერთეულებისათვის.

ამ მოდელს აქვს ღირსებებიც და ნაკლოვანებანიც. ძირითადი ნაკლოვანებაა ის, რომ ფაქტობრივად, სერვისების მიმწოდებლის განვითარება შეზღუდულია ბიზნესერთეულის შესაძლებელი განვითარებით, რომელსაც იგი ემსახურება.

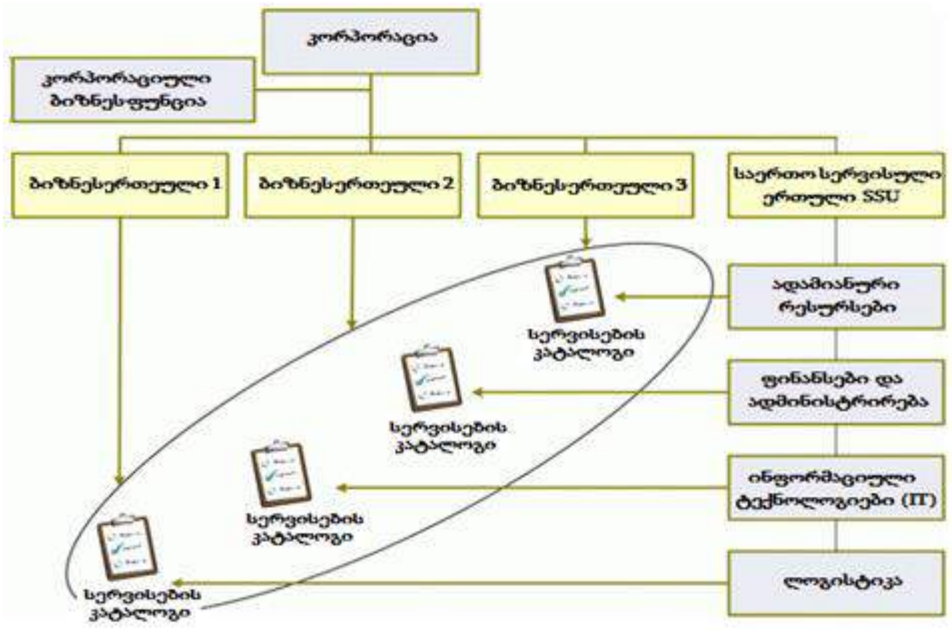
ის, რომ გადაწყვეტილებას იღებს ორგანიზაციის ხელმძღვანელი, ესეც ერთგვარი ნაკლოვანებაა, რადგან ის მთლიანად ვერ ერკვევა IT–სფეროს ტექნიკურ ნიუანსებში. დადებითი მომენტია ის, რომ ბიზნესი არ ეჯახება პრობლემებს, რომლებიც აღმოცენდება სერვისების გარე მიმწოდებლაბთან ურთიერთობისას. ასევე სერვისების პირველი ტიპის მიმწოდებელი არ ეჯახება თავისუფალი ბაზრის სირთულეებს.

ზოგადად, სერვისების მიმწოდებლები, რომლებიც ემსახურებიან ერთზე მეტ დამკვეთს, ეჯახებიან ბევრ რისკს. მათი ახლო თანამშრომლობა დამკვეთთან რისკების აცილების საწინდარია. ამავდროულად, სერვისების გარე მიმწოდებლებს აქვთ მოქმედების და განვითარების მეტი თავისუფლება, ავტონომიურობა და მასშტაბურობა.. აღნიშნული თავისებურებების გამო, პირველი ტიპის მიმწოდებლები უფრო მიესადაგება ისეთ ბიზნესს, სადაც IT ჩადებულია კონკურენტული უპირატესობის საფუძველში, და აქედან გამომდინარე, მოითხოვს საგულდაგულო კონტროლს უშუალოდ ორგანიზაციის ხელმძღვანელობიდან.

- **ტიპი 2**

ისეთი საქმიანი ფუნქციები, როგორიცაა ფინანსური მენეჯმენტი, კადრების მართვა და ლოგისტიკა, ყოველთვის არაა კონკურენტული უპირატესობის საფუძველი. აქედან გამომდინარე, ორგანიზაციის ხელმძღვანელის და ტოპ–მენეჯერებისთვის არაა აუცილებელი აკონტროლონ და მართონ ეს სფეროები. ამის ნაცვლად ასეთ ფუნქციათა სერვისები ერთიანდება ცალკე სერვისულ ერთეულში – **სერვისების საერთო მიმწოდებელი** (**Service Shared Unit** ან **SSU).**

SSU (ნახ.7.8), როგორც სერვისების მიმწოდებელი, ფლობს მეტ თავისუფლებას, ვიდრე პირველი ტიპის მიმწოდებლები. მას შეუძლია შექმნას, განავითაროს და მხარი დაუჭიროს თავისი სერვისების გასაღების შიგა ბაზარს, ანალოგიურად თავისუფალ ბაზარზე მომუშავე მიმწოდებლებისა. ამავდროულად, SSU–ს შეუძლია გამოიყენოს კორპორაციის შესაძლებლობები 1–ელი ტიპის მიმწოდებლების ანალოგიურად. ე.ი. SSU იმყოფება 1-ელი და მე-3 ტიპის მიმწოდებლების გადაკვეთაზე. მეორე ტიპის მიმწოდებლები, ფაქტობრივად, ადუბლირებენ (ემულირება) გარეშე მიმწოდებელთა საქმიანობას, იყენებენ რა მათ მუშა მოდელებს, ბიზნესპრაქტიკას და სტრატეგიებს. აქედან გამომდინარეობს ის, რომ სერვისების გარე მიმწოდებლები მათი ძირითადი კონკურენტებია.



ნახ.7.8. მეორე ტიპის პროვაიდერის სქემა

SSU სერვისების საბოლოო მომხმარებლებია ბიზნესერთეულები, ინვესტორები და მთლიანად კორპორაცია. ამასთანავე, მე–2 ტიპის მიმწოდებლებმა შიძლება უკეთესი ფასები შესთავაზონ, ვიდრე გარე მიმწოდებლებმა, კორპორაციაში მათი მუშაობის, შიგა ხელშეკრულებების და ბიუჯეტიდან ფინანსირების საფუძველზე. მე–2 ტიპის მწარმოებლები, როგორც 1–ელი ტიპისა, იღებენ უპირატესობას შედარებით ჩაკეტილი ბაზრიდან. მაგრამ ამავედროს სერვისების მომხმარებლები ადარებენ მათ გარე მიმწოდებლებს. ცუდი მეორე ტიპის მიმწოდებლები იქნება ჩანაცვლებული გარე მიმწოდებლებით. ეს აიძულებს ხელმძღვანელობას გამოიყენოს უკეთესი პრაქტიკები, აითვისოს ახალი საბაზრო სივრცეები, განსაზღვროს სტრატეგიები და განავითაროს თავისი სერვისების განსხვავებული მახასიათებლები.

3. ტიპი 3 – სერვისების გარე მიმწოდებლები

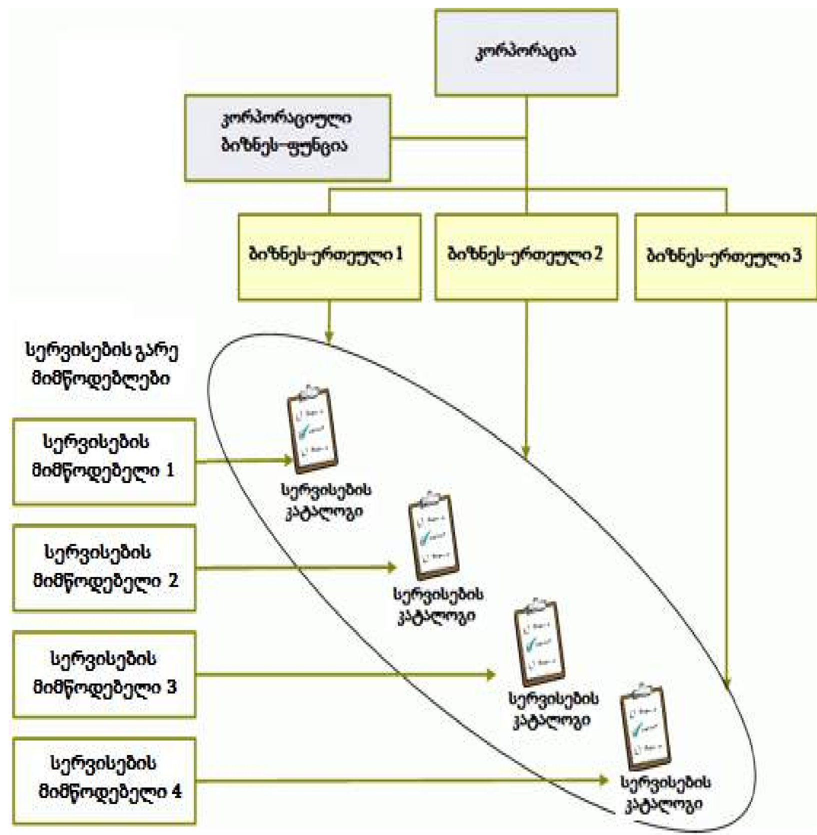
სერვისის გარე მიმწოდებლები თავიანთი დამკვეთი ორგანიზაციის გარეთ არსებობენ, განსხვავებით წინა ორი განხილული ტიპისაგან. ისინი მოქმედებენ ღია ბაზარზე და აქედან გამომდინარე, ეჯახებიან რიგ სირთულესა და რისკს. თუ 1 და 2 ტიპის მიმწოდებლებს ყოველთვის ჰყავთ დამკვეთები, მე–3 ტიპისას უხდება მუდმივად მათი ძებნა, ყურადღების მიქცევა, ამიტომაც უნდა იყვნენ კონკურენტუნარიანი. ეს სირთულეები კომპენსირდება მოქნილობით, მასშტაბურობით და ქმედებებში და გადაწყვეტილებებში თავისუფლებით.

მე–3 ტიპის მიმწოდებლებს აქვს დიდი პრაქტიკული გამოცდილება, ვინაიდან ისინი ემსახურებიან მრავალი სფეროს სხვადასხვა დამკვეთებს (განსხვავებით 1 და 2 ტიპისა, რომლებიც შეზღუდული გამოცდილებით არიან ერთ კორპორაციაში ან ბაზრის ვიწრო ფრაგმენტზე). IT-სფეროსთვის ძალიან მნიშვნელოვანია, რათა სერვისების მიმწოდებელს ჰქონდეს გამოცდილება IT-სერვისების წარმოდგენისათვის, ამიტომაც ეს კრიტერიუმი ხშირად გადამწყვეტია მიმწოდებლის არჩევისას.

მე–3 ტიპის მიმწოდებლების არჩევის მოტივაციისათვის აგრეთვე გაითვალის-წინება გამოცდილება, ცოდნა, რესურსები და ფართო შესაძლებლობები სერვისების მასშტაბირებისათვის. ამასთანავე, ბიზნესი ყოველთვის მიისწრაფვის ხარჯების შემცირებისაკენ, გარე მიმწოდებლებს კი შეუძლიათ კონკურენტუნარიანი ფასების შეთავაზება, ხარჯების შემცირებისა და მოთხოვნებზე სწრაფი რეაგირების გზით. ამიტომაც, ხშირად ორგანიზაციებისათის უფრო ხელსაყრელია გარე მიმწოდებლების გამოყენება, ვიდრე შიგა მიმწოდებლების შენახვა და მთელი აქტივების მართვა, რაც სერვისის დამოუკიდებელი რეალიზაციისათვისაა საჭირო.

შეძლება ითქვას, რომ მე–3 ტიპის პროვაიდერები იმყოფებიან საერთო სერვისული მოდელის მართვის ქვეშ (ნახ.7.9). ეს გამოიხატება იმაში, რომ მათი რესურსები და შესაძლებლობები განაწილებულია კლიენტებს შორის, რომელთა შორის ზოგი მათივე კონკურენტია. აქედან გამომდინარე, კონკურენტები იღებენ ერთმანეთის ფასებზე

წვდომის უფლებას, და ამცირებენ მათ მნიშვნელობებს. ასეთ დროს მნიშვნელოვანია უსაფრთხოების უზრუნველყოფა, ეს განსაკუთრებული ასპექტია IT–სერვისის გამოყენებისას. ხოლო როდესაც გარემო არის საერთო კონკურენტებისათვის, მაშინ ის იღებს განსაკუთრებულ მნიშვნელობას.



ნახ.7.9. სერვისების მესამე ტიპის მიმწოდებლები

ყოველ მიმწოდებელს აქვს ღირსებები და ნაკლოვანებებიც. დამკვეთის მიერ მიმწოდებლის ამორჩევა დამოკიდებულია ბევრ ფაქტორზე: საოპერაციო ხარჯებზე, ინდუსტრიის თავისებურებებზე, მის კომპეტენციებზე და რისკებზე. ორივე მხარისთვის სასარგებლოა საოპერაციო ხარჯების წარმოშობის პროცესის გაგება: დამკვეთს – რათა აირჩიოს მიმწოდებელი, ხოლო მიმწოდებელს – იმის გასაგებად, თუ როგორ შეირჩიოს დამკვეთი. ოპერაციული ხარჯები – ესაა ყველა ხარჯი, რასაც გაიღებს ბიზნესი, სერვისების მიმწოდებელთან მუშაობისას.

გარდა თვით სერვისების ღირებულებისა, ესაა ხარჯები კვალიფიციური მიმწოდებლის მოსაძებნად, მოთხოვნების განსაზღვრის მიზნით სერვისების

პორტფელისათვის, მოლაპარაკებების წარმართვაზე, მწარმოებლურობის შეფასებაზე, დავების გადაწყვეტაზე, ცვლილებების შესატანად და სრულყოფის მიზნით.

ენდობა თუ არა დამკვეთი გარე ან შიგა მიმწოდებლების განსაზღვრულ საქმიან აქტივობას, დამოკიდებულია შემდეგი კითხვების პასუხებზე:

1. სჭირდება თუ არა საქმიან აქტიურობას სპეციფიური აქტივები?
2. რამდენად ხშირად გამოიყენება საქმიანი აქტიურობა ბიზნესციკლში?
3. რამდენად ძნელია საქმიანი აქტიურობა?
4. ძნელია მაღალი დონის მწარმოებლურობის განსაზღვრა?
5. ძნელია მწარმოებლურობის დონის განსაზღვრა?
6. რამდენად მჭიდროდაა იგი დაკავშირებული ბიზნესის სხვა აქტიურობებთან და აქტივებთან მისი გამოყოფა გამოიწვევს პრობლემებს და გაზრდის ბიზნესპრცესების სირთულეს?

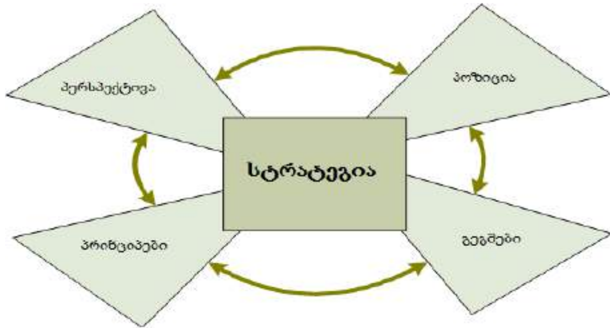
მაგალითად, თუ აქტიურობა გამოიყენება იშვიათად ან ერთეულ შემთხვევაში, ან თუ ის მარტივი, რუტინულია და არ იცვლება დროში, ანუ სტაბილურია, მაშინ ის უკეთესია მიეცეს გარე მიმწოდებელს.

თუ საქმიანი აქტიურობის მწარმოებლურობა რთული გასაზომი, შესაფასებელი და გასაკონტროლირებელია, მაშინ ის უკეთესია მიეცეს შიგა მიმწოდებელს. თუ აქტიურობა მჭიდროდაა კავშირში ბიზნესთან, ხოლო მისი გამოყოფა გამოიწვევს სირთულეს, მაშინ უკეთესია მისი დატოვება ორგანიზაციის შიგნით.

უნდა აღინიშნოს, რომ პასუხები დასმულ შეკითხვებზე შეიძლება შეიცვალოს დროის მიხედვით, მდგომარეობების შეცვლის, ახალი ტექნოლოგიების ან მოთხოვნების გაჩენის გამო.

7.5. ოთხი „P“ სტრატეგიის ასაგებად

წიგნში "ITILv3.Service Strategy" აღიწერება სტრატეგიის აგების ოთხი შესასვლელი წერტილი, ე.წ. „Four Ps of Strategy“ – Perspective (პერსპექტივა), Positions (პოზიცია), Plans (გეგმები) и Patterns (პრინციპები). სწორედ ეს განსაზღვრავს სტრატეგიის ფორმას (ნახ.7.10).



ნახ.7.10. სტრატეგიის ოთხი "P"

პერსპექტივა – განსაზღვრავს სერვისების მიმწოდებლის განვითარების მიმართულებას, მის ფასეულობებს და საერთო მიზანს. სტრატეგიული პერსპექტივა აფორმირებს ურთიერთობის ფილოსოფიას დამკვეთთან და სერვისების წარმოდგენის მეთოდებს. მაგალითად, სერვისების მეორე ტიპის მიმწოდებელს საერთაშორისო იურიდიული კომპანიისთვის შეუძლია დააფორმიროს ეს შემდეგნაირად: „ჩვენ ვიქნებით საუკეთესო პროვაიდერი ჩვენ კლასში ჩვენი იურიდიული ფირმისათვის“.

სერვისების მესამე ტიპის მიმწოდებლისათვის შესაფერისი იქნება „ფოკუსირება მომხმარებელზე, ხოლო სხვა დანარჩენი დაერთვება მას“ ან „ჩვენი მიზანია მომხმარებელთა ცხოვრების გაუმჯობესება“. პერსპექტივა, განსხვავებით გეგმებისა და პოზიციებისგან, შედარებით მუდმივი და მდგრადია ცვლილებებისადმი.

წიგნში "ITILv3.Service Strategy" მოტანილია მაგალითი შვეიცარული საათების ინდუსტრიაზე. 1970–იანი წლების დასაწყისში დაიწყეს საათებში კვარცის გამოყენება რხევითი სისტემის საშუალებად. ამან გამოიწვია წარმოების 10–ჯერ გაიაფება, ხარისხის შენარჩუმნებით მაღალ დონეზე. მიუხედავად ამისა, შვეიცარელმა მწარმოებლებმა გაითვალეს, რომ ამ ტექნოლოგიის გამოყენება ეწინააღმდეგება საათების წარმოების პროფესიონალურ ოსტატობას. იაპონელი მწარმოებლები კი პირიქით, აქტიურად იყენებდნენ კვარცს და აგდებდნენ ბაზრიდან შვეიცარულ საათებს. ეს ხდებოდა მანამდე, სანამ შვეიცარელებმა არ შეცვალეს თავისი მარკეტინგული კამპანია, შეცვალეს ორიენტაცია მდიდარი კლიენტებისკენ, ე.წ. luxury – ბაზრის სეგმენტი. დღეისათვის შვეიცარული საათები თავისებური ნიმუშია ხარისხის, სტილის და სიმდიდრის მოწმეა თავისი მფლობელის.

პოზიცია. პოზიციონირება გულისხმობს პასუხების გაცემას რიგ შეკითხვებზე, მაგალითად:

– უნდა ამაღლდეს სერვისების ფასები თუ შემცირდეს ხარჯები?
– ყურადღება უნდა გამახვილდეს ხარისხის გარანტიაზე თუ სარგებლიანობაზე?

პირველი ტიპის პროვაიდერს შეუძლია პოზიციის აგება ლოზუნგით „ ვიცი, რაც უნდა ვაწარმოო“ ან „ვგრძნობ მომხმარებელს“. პოზიციონირება ხშირად ეფუძნება ბიზნესის მიმდინარე მოთხოვნებს და გამოისახება იმაში, თუ რითი განსხვავდება ეს მიმწოდებელი სხვებისგან მომხმარებლის თვალსაზრისით. გამოყოფენ სამი ტიპის ყველაზე მეტად გავრცელებულ პოზიციას:

პოზიციონირება სერვისების სახის საფუძველზე (variety-based positioning) გულისხმობს, რომ მიმწოდებელი სპეციალიზდება დამკვეთების მოთხოვნილებათა განსაზღვრულ სახეზე (ნახ.7.11). ეს მიდგომა გულისხმობს სერვისების სპექტრის შემცირებას, მაგრამ მათი შესაძლებლობების გაზრდას კონკრეტული სახის მოთხოვნილე-ბების მაქსიმალურად დასაკმაყოფილებლად. განვითარება შესაძლებელია უპირატესად ახალი შესაძლებლობებით სერვისების არსებულ კატალოგში, და არა ახალი სერვისების

შემოტანით. ანუ სერვისების მიმწოდებელს შეუძლია თავიდან მოემსახუროს ერთ ბიზნეს–ერთეულს, შემდეგ რამდენიმე ბიზნესერთეულს კომპანიის ჩარჩოში, ან რამდენიმე კომპანიას რეგიონის ჩარჩოში.

დამკვეთთა სეგმენტები

დამკვეთთა მოთხოვნილებები

	A	B	C	D	E
1					
2					
3					
4					
5					

ნახ.7.11. პოზიციონირება სერვისების სახის საფუძველზე

- პოზიციონირება მოთხოვნილებათა საფუძველზე (needs-based positioning) გულისხმობს, რომ სერვისების მიმწოდებელი ცდილობს დააკმაყოფილოს განსაზღვრული ტიპის დამკვეთის ყველა ან თითქმის ყველა მოთხოვნილება (ნახ.7.12). ეს მოითხოვს სერვისების კატალოგის გაფართოებას, რადგან მიმწოდებელმა უნდა დააკმაყოფილოს სხვადასხვა სახის მოთხოვნილება. განვითარება შესაძლებელია უპირატესად ახალი სერვისების დამატებით კატალოგში.

დამკვეთთა სეგმენტები

დამკვეთთა მოთხოვნილებები

	A	B	C	D	E
1					
2					
3					
4					
5					

ნახ.7.12. პოზიციონირება მოთხოვნილებათა საფუძველზე

• პოზიციონირება წვდომის საფუძველზე (access-based positioning) გულისხმობს, რომ სერვისების მიმწოდებელი თავის განსაკუთრებულ თავისებურებად სთავაზობს მზადყოფნას სერვისების მისაწოდებლად დამკვეთის ადგილმდებარეობის, მასშტაბის და სტრუქტურის გათვალისწინებით (ნახ.7.13).

ნახ.7.13. პოზიციონირება წვდომიოს საფუძველზე

მიმწოდებლები განსხვავდებიან სამუშაოს ზომით, სტრუქტურითა და საზღვრებით. ზოგიერთი კორპორაციის თანამშრომელი მობილურია, მაგრამ მიუხედავად ამისა, უნდათ წვდომის მიღება ყველა კომუნიკაციასთან. სხვა ორგანიზაციის თანამშრომლები მუშაობენ სტაციონარში, მაგრამ პლანეტის შორეულ კუთხეებში. პოზიციონირების ეს სახე გულისხმობს ბიზნესის მოთხოვნების დაკმაყოფილებას, ყველა თანმხლები თავისებურებების გათვალისწინებით. ამ შემთხვევაში ბუნებრივია ვიწრო სპეციალიზაცია. მოცემული ფირმის სტრატეგია ძალზე საშიშია, რადგან იგი ძალზე დაუცველია: მოულოდნელი ცვლილება ბიზნესში ან ბაზრის სეგმენტზე იწვევს მოთხოვნილების მკვეთრ დაქვეითებას და შესაბამისად, სერვისების მიმწოდებლის კრახს.

გეგმა – აღწერს გადაწყვეტილებათა და ქმედებათა მიმდევრობას საწყისი მდგომარეობიდან გადასასვლელად სტრატეგიულ მიზნობრივში. განსაკუთრებით განიხილება ბიუჯეტის, სერვისების პორტფელის, ახალი სერვისების განვითარების, ინვესტიციებისა და სრულყოფის საკითხებზე. გეგმა შეიძლება დეტალიზებულ იქნას, მაგალითად, „როგორ შეგვიძლია ფასიანი ან იაფი სერვისების მიწოდება“.

პრინციპი – აღწერს ორგანიზაციის ფუნდამენტურ გზას. პრინციპი ამ შემთხვევაში არის ქმედებებისა და გადაწყვეტილებების მიმდევრობა, რომლებიც დროში შედარებით მუდმივია. პრინციპები ფორმირდება საუკეთესო შედეგების საფუძველზე, თუ რამემ როდისმე მოიტანა წარმატება, ის შეიძლება კიდევ იყოს გამოყენებული განმეორებით.

სერვისების მიმწოდებელი, რომელიც იძლევა სპეციალიზებულ სერვისებს, მოითხოვს მაღალ კვალიფიკაციას, გამოიყენებს ე.წ. „მაღალი კლასის“ სტრატეგიას. ის, ვინც იძლევა საიმედო სერვისებს, იყენებს „ხარისხის მაღალი გარანტიის“ სტრატეგიას.

მოთხოვნები და პირობები დინამიკურია, ამიტომ სერვისების მიმწოდებელს შეუძლია დაიწყოს ერთი ფორმის სტრატეგიით, და დაამთავროს სხვა სტრატეგიით. მაგალითად, მიმწოდებელმა შეიძლება დაიწყოს პერსპექტივის აგებით, ანუ ორგანიზაციის მიზნის და მიმართულების განსაზღვრით. შემდეგ მას შეუძლია პოზიციონირების გამოყენების გადაწყვეტა, დაფუძნებული ორგანიზაციის შესაძლებლობებზე, რესურსებსა და პოლიტიკებზე. ეს შეიძლება მიღწეულ იქნას გულდასმით მოფიქრებული გეგმით. მიაღწევს რა ერთხელ სასურველ შედეგებს, სერვისების მიმწოდებელს შეუძლია მართოს თავისი პოზიცია კარგად გაგებული გადაწყვეტებით და ქმედებებით – პრინციპებით.

7.6. ფინანსების მართვა

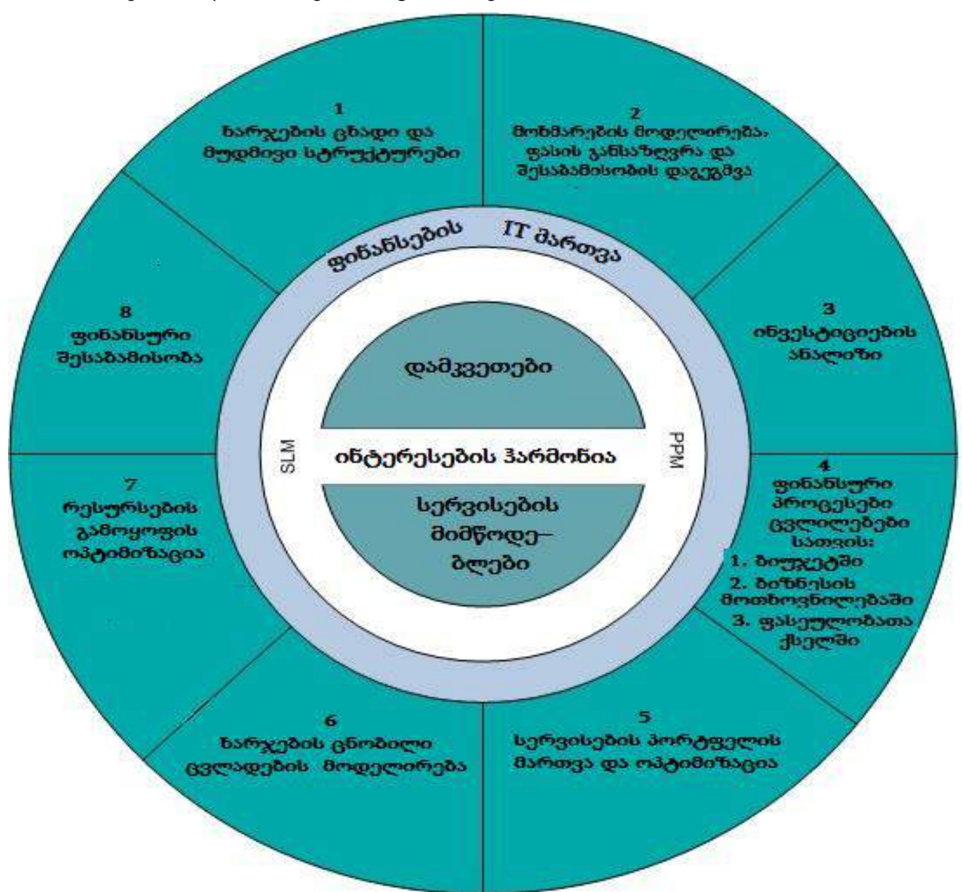
ფინანსების მართვა (Financial Management) - ესაა ფუნქცია და პროცესები, პასუხისმგებლები ბიუჯეტის მართვაზე, სერვისების მიმწოდებლის ხარჯების აღრიცხვის და მათი ანაზღაურების მიზნით. ფინანსების მართვა სტრატეგიული ინსტრუმენტია ყველა ტიპის საერვისების მიმწოდებლისთვის. შიგა მიმწოდებლებიც კი ვალდებულნი არიან იმოქმედონ ფინანსური გამჭვირვალობის დონეების და ბიზნესერთეულების აღრიცხვის შესაბამისად, რომელთაც ის ემსახურება.

ფინანსების მართვა აძლევს ბიზნესს და IT-ს რაოდენობრივ ფინანსურ შეფასებას სერვისების ფასეულობაზე, აქტივების ღირებულებაზე, რომლებიც საფუძვლად უდევს ამ სერვისების გამოყენებას, ასევე მეთოდები და ინსტრუმენტები ოპერატიული პროგნოზირებისათვის. ფინანსების მართვა არის საშუალება ისეთი რთული საკითხის გადასაწყვეტად, როგორიცაა ბიზნესის მიერ IT–სფეროს აღქმა. IT–ორგანიზაციები სულ უფრო ხშირად იყენებს ფინანსების მართვას ისეთ პროცესებში, როგორიცაა:

- გადაწყვეტილების მიღება;
- ცვლილებების დაჩქარება;
- სერვისების პორტფელის მართვა (SPM);
- ფინანსური კონტროლი;
- ოპერატიული მართვა;
- ფასეულობის შექმნა და ფიქსირება[6].

ITIL–ში ორგანიზაციების ქვეშ, რომლებიც აწარმოებენ ბიზნესს, ყველაზე ხშირად იგულისხმება დამკვეთები, ხოლო სერვისის მიმწოდებლები გამოდიან მხოლოდ როგორც ბიზნესის ხელშემწყობები. არსებითად, IT–ორგანიზაციები, აწვდიან რა სერვისებს, ასევე აწარმოებენ ბიზნესს, ხოლო ნებისმიერი ბიზნესისთვის ფუნდამენტურად მნიშვნელოვანია ფინანსების სწორი მართვა.

სერვისების მიმწოდებელი თვალყურს უნდა ადევნებდეს ბალანსს მოთხოვნილებასა და შეთავაზებას შორის, უნდა ამინიმიზირებდეს ხარჯებს და ზრდიდეს სერვისების ფასეულობას. 7.14 ნახაზზე მოცემულია ის მომენტები, რომლებიც საერთოა ბიზნესისა და IT ორგანიზაციისათვის.



ნახ.7.14. საერთო ბიზნესსა და IT-ს შორის

ფინანსების მართვა ინფორმაციის წყაროა, რომელიც ხელს უწყობს IT–ორგანიზაციას პასუხი გასცეს შემდეგ კითხვებს:

1. რომელი სტრატეგიაა ყველაზე ეფექტიანი: უფრო მაღალი მოგების მიღება, ხარჯების შემცირება თუ სერვისების ფართო არჩევის უზრუნველყოფა?

2. რომელ სერვისებზეა ხარჯები ყველაზე მეტი და რატომ ?

3. რომელი ტიპის სერვისები რა მოცულობით არის ყველაზე მოთხოვნადი როგორი ფინანსური დაბანდებებია საჭირო მათ მხარდასაჭერად?

4. რამდენად ეფექტურია წარმოდგენილი სერვისების გამოყენებული მოდელები კონკურენტების ანალოგიურ მოდელებთან?

5. მიიყვანა თუ არა სერვისების დაპროექტების სტრატეგიულმა მიდგომამ კონკურენტუნარიან ფასამდე ამ სერვისებზე რაზეა ორიენტირება უკეთესი რისკების შემცირებაზე თუ ხარისხის ამაღლებაზე?

6. რა ძირითადი ნაკლოვანებანი აქვს ჩვენ სერვისებს?

7. სერვისების უწყვეტი სრულყოფის სტრატეგიის აგებისას რომელ ფუნქციურ სფეროებზეა საჭირო კონცენტრირება?

ინფორმაციის გარეშე, რომელსაც ფინანსების მართვა იძლევა, შეუძლებელია ამ კითხვებზე კორექტული პასუხი. ფინანსების სწორი მართვის არარსებობა ანეიტრალებს სტრატეგიის აგების, დიზაინის და სხვა ნებისმიერი ტექნიკური გადაწყვეტის არსს. ფინანსების მართვა უზრუნველყოფს ხარჯების გამჭვირვალობას და მოზანშეწონილობას ამ სერვისებზე, როგორც ბიზნესისათვის, ისე თვით მიმწოდებლისთვისაც. ფინანსების მართვა მოიცავს შემდეგ ძირითად ამოცანებს:

- სერვისების ფასეულობის შეფასება;
- მოთხოვნის მოდელირება;
- სერვისების პორტფელის მართვა;
- სერვისების უზრუნველყოფის ოპტიმიზაცია;
- შესაბამისობის დაგეგმვა;
- ინვესტიციების ანალიზი სერვისებში;
- საბუღალტრო ანგარიშგების ფორმირება;
- შესაბამისობა;
- ხარჯების ცვლადების მოდელირება.

ახლა უფრო დეტალურად დავახასიათოთ ეს ამოცანები.

1. **სერვისების ფასეულობის შეფასება (Service Valuation)** - ესაა სრული ხარჯების შეფასება მიმწოდებლისთვის მის მიერ წარმოდგენილ სერვისზე და ამ სერვისის სრული ფასეულობა ბიზნესისათვის. სერვისის ფასეულობის შეფასება გამოიყენება იმისთვის, რომ დახმარება გაეწიოს ბიზნესს და მიმწოდებელს, რათა მოხდეს შეთანხმება სერვისის ფასეულობაზე. ამ პროცესის ძირითადი მიზანი – სერვისის ფასის განსაზღვრაა, რომელსაც დამკვეთი ჩათვლის სამართლიანად, და მიმწოდებელს მისცემს მოგებას და სერვისის მხარდაჭერას.

როგორც უკვე აღინიშნა, სერვისის ფასეულობა შედგება ორი ძირითადი პარამეტრისგან – სარგებლიანობა და ხარისხის გარანტია. ეს პარამეტრები მოითხოვს ფინანსურ გამოსახვას. აქედან სერვისების ფასეულობის შეფასება იყენებს ორ საკვანძო კონცეფციას:

1.1. **უზრუნველყოფის ფასი (Provisioning Value)** - ესაა ფაქტობრივი ფასი სერვისის უზრუნველსაყოფად მიმწოდებლისათვის. იგი შეიცავს ხარჯებს რესურსებზე, რომლებიც აუცილებელია მის ასამოქმედებლად. ძირითადი მათგანი მოცემულია ქვემოთ:

- ლიცენზიების ფასი პროგრამულ უზრუნველყოფაზე;

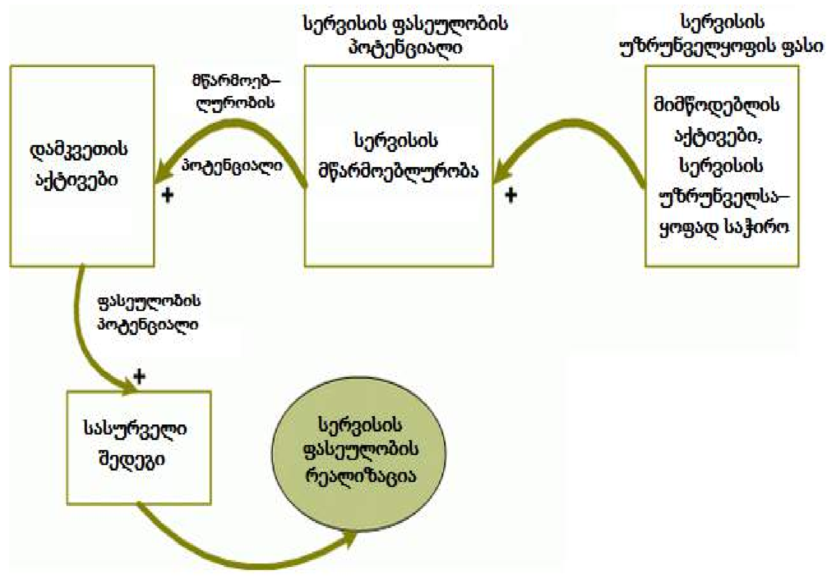
- მოწყობილობის შესყიდვა ან არენდა;
- ადამიანური რესურსები;
- კომუნალური მომსახურება, ქსელის მხარდაჭერა, ინფორმაციული ცენტრის და სხვა ხარჯები მომსახურების საშუალებებზე;
- გადასახადები, ამორტიზაცია, %–ები სესხების მიხედვით.

ამ ხარჯთა ჯამი წარმოადგენს მინიმალურ ფასს სერვისზე – ეს იგივე ფინანსური ზღუდეა, რომლის ქვემოთაც მიმწოდებელი ვერ გადავა კომერციული წინადადების ფორმირებისას.

1.2. **სერვისის ფასეულობის პოტენციელი (Service Value Potential)** - ესაა შეფასება, დაფუძნებული სერვისის ფასეულობაზე დამკვეთის თვალსაზრისით ან წარმოდგენილი სერვისის სარგებლიანობის და გარანტიის ზღვრული მნიშვნელობები, დამკვეთის საკუთარი აქტივების გამოყენებასთან შედარებით.

თავიდან საფუძვლის სახით დგება სერვისის ელემენტები, რომელთაც შეუძლია დამკვეთისთვის ფასეულობის მოტანა. შემდეგ ყოველი ელემენტი ფასდება ცალცალკე მათთვის მინიჭებული ფასეულობების შესაბამისად. ბოლოს, ყველა ელემენტის ჯამი იკრიბება დანახარჯებთან ერთად, რომელიც მის წარმოდგენისთვისაა საჭირო, რათა განისაზღვროს სერვისის საბოლოოს ფასი.

ორი კონცეფციის ურთიერთკავშირი მოცემულია 7.15 ნახაზზე.



ნახ.7.15. მომხმარებლის აქტივები, როგორც სერვისის ფასეულობის ფორმირების საფუძველი

2. **მოთხოვნის მოდელირება**

მოთხოვნის არასაკმარისად ცოდნამ და მისმა გავლენამ ყველა პროცესზე შეიძლება გამოიწვიოს დიდი ხარჯი და რისკი. კერძოდ, მოთხოვნა მჭიდროდაა დაკავშირებული სერვისების რაოდენობაზე, რომლებსაც დამკვეთი „აწარმოებს".

ეს მოითხვს ფინანსების მართვისგან ბიუჯეტის შესაძლო რხევების პროგნოზირების და გაზომვის უნარს, მოთხოვნის ყოველი ცვლილებისას.

სერვისებზე მოთხოვნის შესაფასებლად, გადაწყვეტილების მისაღებად და კონტროლისთვის საკვანძოდ ითვლება ინფორმაცია სერვისების კატალოგიდან და სიმძლავრეების მართვიდან.

სიმძლავრეების მართვა (Capacity Management) - პროცესია, რომელიც პასუხისმგებელია სერვისების სიმძლავრეების და IT-ინფრასტრუქტურის დროულ და ხარჯებით ეფექტიან შესაბამისობაზე მოთხოვნილებებთან, რომლებიც შეთანხმებულია სერვისის დონის მიზნობრივ მაჩვენებლებთან. სიმძლავრეების მართვა ითვალისწინებს ყველა რესურსს, რომელიც აუცილებელია სერვისის უზრუნველსაყოფად, აგრეთვე აწარმოებს ბიზნესის მოთხოვნილებების მოკლევადიან, საშუალო ვადიან და გრძელვადიან დაგეგმვას [11].

მნიშვნელოვან როლს მოთხოვნის მოდელირებისას თამაშობს საერთო ღირებულების დამკვეთის მიერ სერვისის გამოყენების განსაზღვრა. **საერთო ღირებულების გამოყენება (Total Cost of Utilization ან TCU)** - ესაა დამკვეთის სრული დანახარჯები სერვისის გამოყენებაზე მისი მთლიანი სასიცოცხლო ციკლის მანძილზე.

მოთხოვნის მოდელირება ემსახურება ბიზნესის მიერ სერვისის მოსალოდნელი გამოყენების შეფასებას და ამ დროს სერვისების მიმწოდებლის აუცილებელი რესურსების შეფასებას. სერვისების კატალოგი გავლენას ახდენს მოთხოვნის მოდელი-რებაზე, მაგრამ ყველა IT–ორგანიზაციისათვის უნდა არსებობდეს უკუკავშირიც – მოთხოვნის მოდელირება უნდა ახდენდეს გავლენას სერვისების კატალოგზე.

3. **სერვისების პორტფელის მართვა**

უზრუნველყოფის სრული ღირებულების ფინანსური შეფასება ეხმარება მიმწოდებელს თავისი სერვისების შესადარებლად კონკურენტთა ანალოგებთან. ეს შედარება აუცილებელია საკვანძო გადაწყვეტილების მისაღებად – სასარგებლოა თუ არა მიმწოდებლისთვის ამა თუ იმ სერვისის შეთავაზება.

4. **სერვისების უზრუნველყოფის ოპტიმიზაცია (Service Provisioning Optimization ან SPO)** - ესაა სერვისის ფინანსებისა და შეზღუდვების ანალიზი გადაწყვეტილების მისაღებად, იმ შემთხვევაში, როცა სერვისის უზრუნველყოფის ალტერნატიული მიდგომა იძლევა ხარჯების შემცირების ან ხარისხის გაუმჯობესების შესაძლებლობას. ფინანსების მართვა არის საკვანძო SPO–თვის, რომლის ძირითადი კანდიდატებია სერვისები, რომლებიც აღნიშნულია კატალოგში წასაშლელად.

სერვისის უზრუნველყოფა შეიძლება გახდეს არასარგებლიანი მომწოდებლისთვის, თუ კონკურენტებს შეუძლიათ უკეთესი ხარისხის ან სარგებლიანობის ან დაბალი ფასის შეთავაზება. სერვისის წაშლა შეიძლება იყოს სხვა ფაქტორების შედეგიც. მაგალითად, ბანალური დაბერება. ფინანსების მართვა უზრუნველყოფს IT-ორგანიზაციას ინფორმაციით არსებული ხარჯების შესახებ სერვისზე, ალტერნატიული მეთოდების არსებობაზე, მათი გამოყენების შესაძლებლობებზე სხვა სერვისებთან კომბინაციაში, ფინანსურ სტრუქტურებში და ა.შ. ეს ინფორმაცია მეტად მნიშვნელოვანია სერვისების პორტფელის ფორმირებისათვის.

5. **მიმნდობი დაგეგმვა**

ფინანსების მართვის ერთ-ერთი მიზანია სერვისების სათანადო დაფინანსების და თანხლების უზრუნველყოფა. დაგეგმვა ასრულებს სერვისებზე მოთხოვნის რაოდენობრივ შეფასებას მომავლისათვის. შემავალი მონაცემები შეკრებილ უნდა იყოს IT-ორგანიზაციის და ბიზნესის საქმიანობის ყველა სფეროდან და უნდა ასახავდეს მთლიან სურათს.

„მიმნდობი“ აქ ნიშნავს გარკვეული დამაჯერებლობის არსებობას, რომ ფინანსურ მართვაში გამოყენებულ მოთხოვნისა და შეთავაზების მონაცემებს და მოდელებს აქვს ჭეშმარიტობის მაღალი დონე. ინფორმაციის შესაბამისობა მნიშვნელოვანია ორი ძირითადი მიზეზით:

- მონაცემები თამაშობს კრიტიკულ როლს ფინანსების მართვის მიერ დასმული მიზნების მისაღწევად;
- არაკორექტული მონაცემების არსებობა არყევს მიღებული გადაწყვეტილების მნიშვნელობას.

რადგან ფინანსური მართვა იძლევა ინფორმაციას მრავალი გადაწყვეტი-ლებისათვის სერვისმენეჯმენტში, ამიტომ მისი საიმედოობის (ჭეშმარიტების) დონე უნდა იყოს მაღალი. ნებისმიერი უნდობლობა (ეჭვი) ამ ინფორმაციის სიზუსტეზე, გამოიწვევს მთლიანად ფინანსების მართვის ფასეულობის უნდობლობას.

6. **ინვესტიციების ანალიზი სერვისებში**

ინვესტიციების ანალიზის მიზანია ღირებულებითი მაჩვენებლების მოპოვება სერვისის მთელი სასიცოცხლო ციკლის მანძილზე. ღირებულებითი მაჩვენებლები ეფუძნება სერვისთა ფასეულობების და მათი მთლიან სასიცოცხლო ციკლზე ხარჯების მოპოვებას.

7. **ხარჯების აღრიცხვა**

ხარჯების აღრიცხვა სერვის–მენეჯმენტის სფეროში მოითხოვს ტრადიციული საბუღალტრო აღრიცხვისგან განსხვავებულ მეთოდებს და საშუალებებს.

ხარჯების აღრიცხვა (Accounting) - პროცესია, რომელიც პასუხს აგებს ფაქტობრივი ხარჯების იდენტიფიკაციის შესახებ სერვისების უზრუნველყოფაზე, მათ შედარებაზე გეგმიურ ხარჯებთან, და ბიუჯეტის გადახრების სამართავად. ფინანსების მართვა

ასრულებს დამაკავშირებელ როლს კორპორაციულ საფინანსო სისტემასა და სერვისმენეჯმენტს შორის.

ხარჯების აღრიცხვის ფუნქციის შედეგები შესავალი მონაცემებია დაგეგმვისათვის და ხელს უწყობს მომარაგებისა და მოხმარების პროცესების კარგად გაგებას და დეტალიზაციას.

ხარჯების კლასიფიკაციისათვის განიხილავენ შემდეგ ხერხებს:

- კაპიტალური / საექსპლუატაციო ხარჯები - კლასიფიკაცია ასახავს საბუღალტრო აღრიცხვის განსხვავებულ მეთოდოლოგიებს, რომლებსაც ითხოვს ბიზნესი და რეგულატორები;
- პირდაპირი / ირიბი ხარჯები
  - პირდაპირი ხარჯები ეხება კონკრეტულ სერვისს, რომელიც მათი ერთადერთი მომხმარებელია;
  - ირიბი ხარჯები ან „განაწილებული" ხარჯები - ესაა ხარჯები, რომლებიც განაწილებულია მრავალ სერვისს შორის ისე, რომ თითოეული სერვისი იყენებს საერთო თანხის რაღაც ნაწილს.
- მუდმივი / ცვლადი ხარჯები - ეს კლასიფიკაცია ეყრდნობა შეთანხმებულ ვალდებულებებს დროის ან ფასის მიხედვით. ასეთი კლასიფიკაციის სტრატეგიული არსი იმაშია, რომ ბიზნესი უნდა მიისწრაფოდეს მუდმივი ხარჯების ოპტიმიზაციისაკენ და ცვლადი ხარჯების მინიმიცაზიისაკენ, მაქსიმალური პროგნოზირებისა და სტაბილურობის უზრუნველსაყოფად;
- ხარჯების ერთეულები - ესაა ადვილად გასათვლელი (მაგ., თანამშრომელთა რაოდენობა, ლიცენზიების რაოდენობა პროგრამებზე) ან გაზომვადი ობიექტები (მაგ., ცენტრალური პროცესორის დატვირთა, ელენერგიის გამოყენება). ხარჯების ერთეული აიდენტიფიცირებს მოხმარების ერთეულს, გათვლილს კონკრეტული სერვისისათვის.

8. **შესაბამისობა (compliance)** - დამაჯერებლობის უზრუნველყოფა სტანდარტების ან სახელმძღვანელო დოკუმენტაციის ერთობლიობის დაცვაში, რაღაცის სისრულეში და მთლიანობაში, განსაზღვრული დადგენილი წესების გამოყენებაში.

ფინანსების მართვის კონტექსტში შესაბამისობა ნიშნავს მეთოდების და პრაქტიკების გამოყენებას, სათანადო სიზუსტით და ხანგრძლიობით. ეს ეხება ფინანსური აქტივების, კაპიტალიზაციის შეფასებას, შემოსავლის განსაზღვრას, წვდომისა და უსაფრთხოების კონტროლს და ა.შ. შესაბამისობა ადვილად მისაღწევია, თუ გამოყენებული მეთოდები და პრაქტიკები დოკუმენტირებულია.

სერვისების მიმწოდებლისათვის მეტად აუცილებელია შეთავაზებული სერვისების შესაბამისობის უზრუნველყოფის ფასის ცოდნა. სერვისები, რომელთა წარმოდგენა შესაძლებელია მოცემული ფასით ერთ სფეროში, შესაძლოა ვერ იქნას იმავე ფასით შეთავაზებული მეორე სფეროში, სწორედ სტანდარტებთან შესაბამისობის პრობლემების, კანონების, დადგენილი ნორმების გამო.

9. **ცვლადი ხარჯების მოდელირება**

ცვლადი ხარჯების მოდელირება (Variable Cost Dynamics ან VCD) - ესაა ტექნიკა, რომელიც გამოიყენება იმის გასაგებად, თუ როგორ ხდება სრულ ხარჯებზე კომპლექსური ცვლადი ელემენტების (ცვლადების) სიმრავლის ზემოქმედება, რომელთაგან ყველას თავისი წვლილი შეაქვს სერვისების უზრუნველყოფაში.

ქვემოთ მოყვანილია მოკლე ჩამონათვალი ხარჯების შესაძლო ცვლადებისა, რომლებიც შეიძლება განხილულ იქნას ანალიზისთვის:

- მომხმარებელთა რაოდენობა და ტიპები;
- ლიცენზიების რაოდენობა პროგრამებზე;
- მიწოდების მექანიზმები;
- მონაცემთა საცავის თანხლების ღირებულება;
- რესურსების რაოდენობა და ტიპები;
- ერთი ახალი შენახვის მოწყობილობის დამატების ღირებულება;
- ერთი ახალი მომხმარებლის დამატების ღირებულება.

ხარჯების ცვლადების რაოდენობა დამოკიდებულია გასაანალიზებელი სერვისის ტიპზე. ამის გამო VCD შეიცავს სცენარების დიდ რაოდენობას და ვარაუდებს, რომელთაგან თითოეული იყენებს თავის ინსტრუმენტების ერთობლიობას, ხარჯების ცვლადების გასათვლელად.

7.7. სერვისების დაპროექტება, როგორც სერვისების სასიცოცხლო ციკლის ეტაპი

სერვისების სასიცოცხლო ციკლში სტრატეგიის აგების ეტაპის შემდეგ ხორციელდება სერვისების დაპროექტება. ამ ეტაპის ძირითადი მიზანია ახალი სერვისების დაპროექტება ან ცვლილებების შეტანა არსებულ სერვისებში. ძირითადი ამოცანები სერვისების დაპროექტების ეტაპზე შემდეგია:

1. სერვისების დაპროექტება, რომელთაც ძალუძს ბიზნესის დახმარება დაგეგმილი შედეგების მისაღწევად;
2. პროცესების დაპროექტება, რომლებიც მხარს უჭერს სერვისების სასიცოცხლო ციკლს;
3. რისკების იდენტიფიკაცია და მათი მართვა;
4. უსაფრთხოებისა და მდგრადობის დაპროექტება IT -ინფრა-სტრუქტურის, მოწყობილობის, აპლიკაციის, ინფორმაციული რესურსების;
5. მეთოდების და მეტრიკების დაპროექტება აზომვებისათვის;
6. გეგმების, პროცესების, პოლიტიკის, სტანდარტების, არქიტექტურის და დოკუმენტების შექმნა, რომლებიც ხელს შეუწყობს ხარისხიანი IT–გადაწყვეტის დაპროექტებას და მათ მართვას;
7. სხვადასხვა შესაძლებლობებისა და ჩვევების განვითარება IT–სფეროში;

8. სერვისების ხარისხის სრულყოფის ხელშეწყობა.

ახალი სერვისებისათვის მოთხოვნები ფორმირდება, წესისამებრ, სერვისების პორტფელის მონაცემების საფუძველზე და ბიზნესის მოთხოვნილებებით. სერვისების დაპროექტება იწყება ბიზნესის მოთხოვნების ერთობლიობის აგებით და სრულდება გადაწყვეტილების შემუშავებით, რომელიც შეძლებს ამ მოთხოვნების დაკმაყოფილებას და დაეხმარება ბიზნესს დაგეგმილი შედეგების მიღწევაში. ნაპოვნი გადაწყვეტა საპროექტო დოკუმენტაციასთან ერთად გადადის დანერგვის ეტაპზე, ახალი/შეცვლილი სერვისის გაშვების, ტესტირების ან განვითარებისათვის.

სერვისის საპროექტო დოკუმენტაცია (Service Design Package ან SDP) - ესაა დოკუმენტები, რომლებიც განსაზღვრავს სერვისის ყველა ასპექტს და მოთხოვნებს მასთან სასიცოცხლო ციკლის ყველა ეტაპზე. მოთხოვნის რეალიზაციამდე საპროექტო დოკუმენტაციაში, იგი ხელმძღვანელობის მიერ უნდა იქნას გაანალიზებული, ფორმალიზებული და მხარდაჭერილი.

ყველა ცვლილება არ ითხოვს სერვისების სასიცოცხლო ციკლში დაპროექტების ეტაპის ქმედებათა ჩართვას. დაპროექტება საჭიროა, როცა აუცილებელია „მნიშვნელოვანი" ცვლილებები. ორგანიზაციამ უნდა განსაზღვროს თავისი „მნიშვნელოვანი ცვლილებების" ერთობლიობა, რათა ორგანიზაციის ყოველმა თანამშრომელმა გაიგოს, თუ როდისაა საჭირო პროექტირება. ანუ, აბსოლუტურად ყველა ცვლილება უნდა იქნას შეფასებული „მნიშვნელობის" მხრივ დაპროექტების კონტექსტში. ასეთი შეფასება არის ცვლილებების მართვის პროცესის ნაწილი.

დაპროექტების ეტაპზე დამუშავებული გადაწყვეტა უნდა შეესაბამებოდეს კორპორაციის და IT-ის პოლიტიკას. ამიტომაც დაპროექტებისას აუცილებელია სტრატეგიისა და შეზღუდვების გათვალისწინება, რომლებიც სტრატეგიის აგების ეტაპზეა ფორმირებული.

საინტერესოა, რომ ITIL-ში გამოფილია ოთხი „P" სერვისების დაპროექტების ეტაპისათვის, ისევე როგორც სტრატეგიის აგების ეტაპისათვის:

- პერსონალი – ადამიანები, ჩვევები და კვალიფიკაცია, საჭირო სერვისების უზრუნველსაყოფად;
- პროდუქტები – ტექნოლოგიები და მართვის სისტემები, გამოყენებული სერვისების უზრუნველსაყოფად;
- პროცესები – პროცესები, როლები და ქმედებები, ჩართული სერვისების უზრუნველსაყოფად;
- პარტნიორები – ვენდორები, მიმწოდებლები და მწარმოებლები, რომლებიც მხარს უჭერენ და ეხმარებიან სერვისებით უზრუნველყოფას.

სერვისების დაპროექტება გლობალური გაგებით არის ბიზნესის ცვლილების საერთო პროცესის ნაწილი. ბიზნესის ცვლილების პროცესი და IT–ის როლი მასში მოცემულია 7.16 ნახაზზე.

ნახ.7.16. ბიზნესის ცვლილების პროცესი

დაპროექტების ძირითადი როლი ბიზნესის ცვლილების პროცესის კონტექსტში მდგომარეობს ინოვაციური სერვისების დამუშავებაში (მათ შორის არქიტექტურები, პროცესები, პოლიტიკები და დოკუმენტაცია), რომლებიც შეძლებენ დააკმაყოფილონ ბიზნესის დღევანდელი და სამომავლო მოთხოვნილებანი. ამ დროს ITSM–ის საკვანძო პროცესები უნდა იქნას გამოყენებული ახალი სერვისების დამუშავების ან არსებულში ცვლილებების შეტანის დასაწყისშივე. ქვემოთ მოყვანილია ქმედებათა ერთობლიობა, რომელთა განხორციელება აუცილებელია პროექტირების ეტაპზე იმისთვის, რომ დამუშავებულმა გადაწყვეტამ ეფექტურად დააკმაყოფილოს ბიზნესის მოთხოვნილებანი:

1. ახალი გადაწყვეტა უნდა იქნას დამატებული სერვისების პორტფელში უკვე კონცეფციის ფორმირების სტადიაზე. სერვისების პორტფელი რეგულარულად უნდა განახლდეს, რათა იგი ასახავდეს ნებისმიერი, თუნდაც უმნიშვნელო ცვლილების აქტუალურ სტატუსს ინკრემენტალური და იტერაციული განვითარების ჩარჩოებში.

2. სერვისის / სისტემის საწყისი ანალიზის ჩარჩოებში აუცილებელია მოთხოვნების გაგება სერვისების დონის მიმართ. მოთხოვნები სერვისების დონის მიმართ **(Service Level Requirements** ან **SLR)** - ერსაა დამკვეთის მოთხოვნა IT-სერვისზე. SLR–ები ბაზირდება ბიზნესმიზნებზე და გამოიყენება მოლაპარაკებებისას და სერვისების დონის მიზნობრივი მაჩვენებლების შეთანხმებისათვის [1].

3. იყენებს რა SLR–ს, სიმძლავრეების მართვის გუნდს შეუძლია ახალი სერვისის მოდელირება არსებული ინფრასტრუქტურის გამოყენებით და იმის გაგება, შეძლებს თუ არა იგი ამ სერვისის მხარდაჭერას მომავალში. თუ დრო საშუალებას იძლევა, მოდელირების შედეგები აისახება სიმძლავრეების უზრუნველყოფის გეგმაში. **სიმძლავრეების უზრუნველყოფის გეგმა (Capacity Plan)** გამოიყენება რესურსების მართვისა, რომლებიც აუცილებელია IT–სერვისის უზრუნველსაყოფად. ეს გეგმა შეიცავს სცენარებს მოთხოვნილების პროგნოზირებისათვის ბიზნესის მხრიდან, და ხარჯების შეფასებას, რომლებიც აუცილებელია სერვისის დონის შეთანხმებული მიზნობრივი მაჩვენებლების უზრუნველსაყოფად.

4. თუ ახალი სერვისის უზრუნველსაყოფად ან არსებული სერვისის გაფართოების მხარდასაჭერად საჭიროა ახალი ინფრასტრუქტურები, მაშინ აუცილებელია ფინანსების მართვის პროცესის მონაწილეობა.

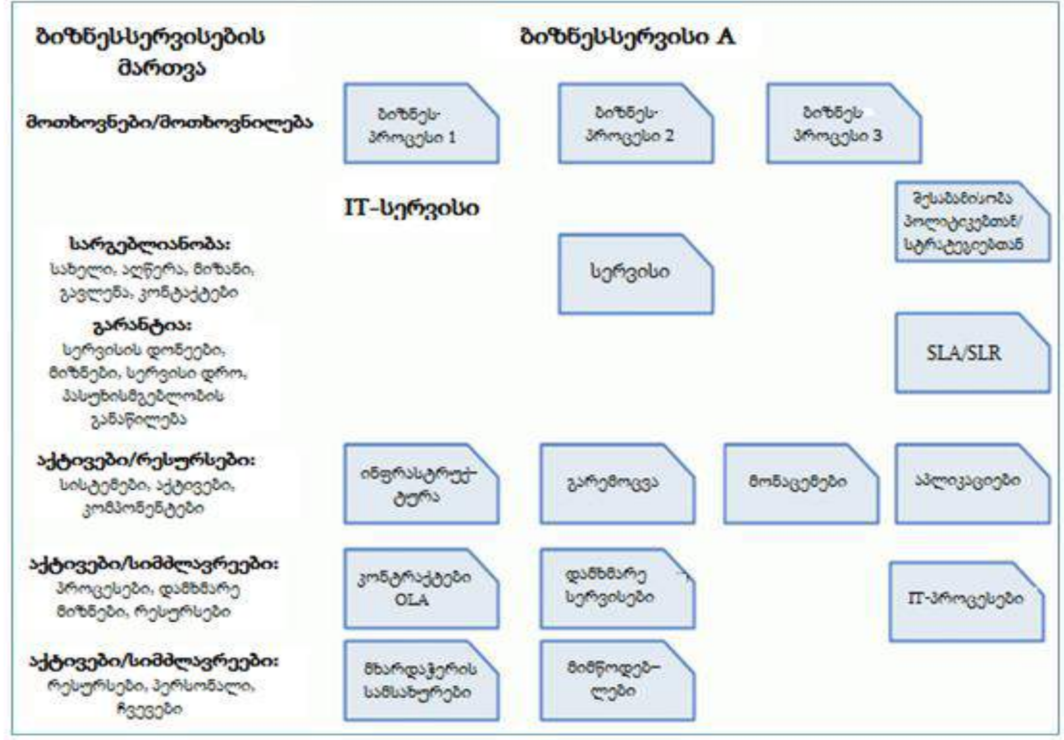
5. ბიზნესზე გავლენის ანალიზი და რისკების შეფასება სერვისთან მიმართებით უნდა ჩატარდეს ადრე, სიმძლავრეების დაგეგმვის, წვდომის დაპროექტებისა და უწყვეტობის სტრატეგიის ფორმირების ეტაპების წინ.

6. სამაგიდო–სერვისი წინასწარ უნდა ემზადებოდეს ახალი სერვისების დასანერგად, კერძოდ, შეასწავლოს თავის პერსონალი.

7. დანერგვის ეტაპი შეიძლება დაიწყოს რეალიზაციის დაგეგმვით და ცვლილებათა ცხრილის აგებით.

8. თუ ახალ სერვისს სჭირდება დამატებითი მომარაგება, საჭიროა მიმწოდებლების მართვის პროცესის ჩართვა.

მიმწოდებლების მართვა (Supplier Management) - ესაა პროცესი, რომელიც პასუხისმგებელია იმის უზრუნველყოფაზე, რომ ხელშეკრულებები მიმწოდებლებთან შეესაბამება ბიზნესის მოთხოვნებს, და ყველა მიმწოდებელი ასრულებს თავის საკონტაქტო ვალდებულებას. სერვისი და მისი კომპონენტები მოცემულია 7.17 ნახაზზე.



ნახ.7.17. სერვისი და მისი კომპონენტები

გადაწყვეტის მოსაძებნად და შესაქმნელად, რომელიც შეძლებენ ბიზნესის ახალი და არსებული მოთხოვნილებების დაკმაყოფილებას, სერვისის დაპროექტებამ უნდა გაითვალისწინოს შემდეგი ასპექტები:

1. ბიზნესპროცესი ფუნქციური მოთხოვნილებების განსაზღვრა, რომლებისათვისაც წარედგინება სერვისი. მაგალითად, ტელეგაყიდა ან ანგარიშ–ფაქტურის შედგენა;

2. სერვისი – თვითონ სერვისი, რომელიც წარედგინება ბიზნესს და მოენახვა;

3. SLA/SLR: დოკუმენტები, შეთანხმებული დამკვეთთან, რომლებიც განსაზღვრავს სერვისის დონეს, არეალს და ხარისხს;

4. ინფრასტრუქტურა – ყველა მოწყობილობა, რომლებიც აუცილებელია მომხმარებლის უზრუნველსაყოფად სერვისით, მათ შორის სერვერები, მარშრუტიზატორები, კონცენტრატორები, ტელეფონები , კომპიუტერები და სხვ.;

5. გარემო – გარემო, რომელიც აუცილებელია ინფრასტრუქტურის უსაფრთხო ექსპლუატაციისათვის: ჰაერის კონდენცირება, ელექტრობა და ა.შ.

6. მონაცემები – მონაცემები, რომლებიც საჭიროა სერვისის მხარდასაჭერად, აგრეთვე ბიზნესპროცესების უზრუნველსაყოფად აუცილებელი ინფორმაციით. მაგალითად, კლიენტთა სია, საბუღალტრო რეგისტრი;

7. აპლიკაცია – ყველა პროგრამული დანართი, რომლებიც აუცილებელია მონაცემთა მართვისათვის და ბიზნესპროცესების ფუნქციური მითხოვნების დასაკმაყოფილებლად;

8. მხარდამჭერი სერვისები: ნებისმიერი დამხმარე სერვისები, რომლებიც აუცილებელია სერვისების უზრუნველსაყოფად;

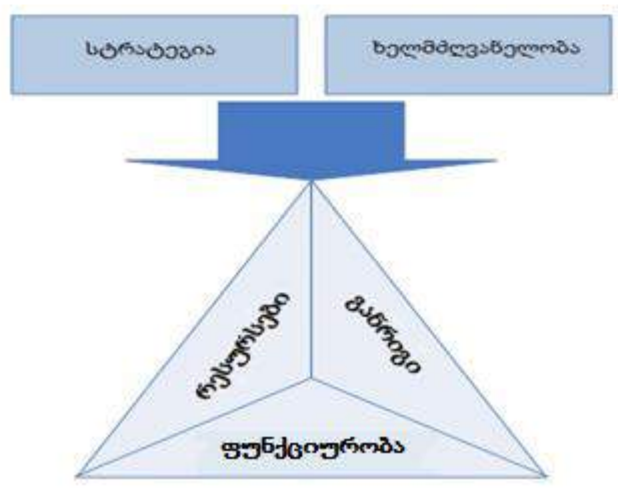
9. ოპერაციული დონის შეთანხმება და კონტრაქტები - ნებისმიერი შეთანხმება, რომელიც აუცილებელია ხარისხიანი სერვისის უზრუნველსაყოფად, რომელიც შეთანხმებულია SLA–ზე;

10. მხარდამჭერი სამსახურები – ნებისმიერი შიგა გუნდი, რომელიც უზრუნველ-ყოფს კომპონენტების მხარდაჭერის პირველ და მეორე ხაზებს, რაც აუცილებელია სერვისის უზრუნველსაყოფად, მაგალითად, Unix ან ქსელები;

11. მიმწოდებლები – პროცესის ნებისმიერი გარე მონაწილეები, რომლებიც უზრუნველყოფენ კომპონენტების მხარდაჭერის მესამე და მეოთხე ხაზებს, რაც აუცილებელია სერვისის უზრუნველსაყოფად – ქსელი, აპარატული და პროგრამული უზრუნველყოფა.

დაპროექტებამ უნდა განიხილოს ზემოჩამოთვლილი თითოეული ასპექტი კომპლექსურად, და არა იზოლირებულად. იმისათვის, რომ შედეგად მიიღონ კონკურენტუნარიანი გადაწყვეტა, ბიზნესის მოთხოვნებისის დამაკმაყოფილებელი, აუცილებელია მითითებული კომპონენტების ურთიერთკავშირებისა და ურთიერთდამოკიდებულებების გათვალისწინება.

სერვისების დაპროექტებისას ბიზნესის ახალი მოთხოვნებით გათვალისწინებული უნდა იქნას ამ მოთხოვნების არამხოლოდ ფუნქციონალური მდგენელი. შემოთავაზებული გადაწყვეტა ამ ეტაპზე უნდა უზრუნველყოფდეს ბიზნესისთვის გეგმურ მწარმოებლურობას. ყველაფრის გაკეთება აუცილებელია არსებული რესურსების გათვალისწინებით, ხარჯებისა და დროის დადგენილ საზღვრებში. ამგვარად, მენეჯერები მუშაობენ სამი მდგენელით (ნახ.7.18):



ნახ.7.18. დაპროექტების სამი მდგენელი

- ფუნქციურობა: სერვისი, მისი ფუნქციური შესაძლებლობები, უნარები და ხარისხი;
- რესურსები: ხელმისაწვდომი ადამიანები, ტექნოლოგიები და ფული;
- განრიგი: დროითი საზღვრები.

დაპროექტების დანიშნულებაა ბალანსის დაცვა ამ სამ მდგენელს შორის, ბიზნესის მოთხოვნილებათა მაქსიმალურად დაკმაყოფილების მიზნით, რომლებიც მუდმივად იცვლება. სამიდან ერთ-ერთის შეცვლა მოქმედებს დარჩენილიდან ერთ-ერთზე მაინც აუცილებლად, ან ორივეზე.

ეფექტური გადაწყვეტების დასამუშავებლად სერვისების მიმწოდებლებისათვის მეტად მნიშვნელოვანია ბიზნესის მამოძრავებელი ფაქტორების გაგება და მათი მოთხოვნილებანი. დაპროექტება ხშირად აღიქმება მხოლოდ როგორც სტადია, რომელიც წინ უსწრებს ექსპლუატაციას. ITIL–ში მიდგომა სხვაგვარადაა, დაპროექტებამ არა მხოლოდ უნდა შემოგვთავაზოს ახალი გადაწყვეტები, არამედ უნდა უზრუნველყოს კიდეც ამ გადაწყვეტების ეფექტური მართვის შესაძლებლობა მთელი სასიცოცხლო ციკლის მანძილზე.

თუ გავაერთიანებთ ყველაფერს ზემოთქმულს, მაშინ დაპროექტებისადმი ერთიანი და სწორი მიდგომა უნდა ითვალისწინებდეს სერვისების დამუშავებას მართვის და სრულყოფის მექანიზმებითა და ფუნქციებით სასიცოცხლო ციკლს ყველა ეტაპზე.

ადამიანები, რომლებიც პასუხისმგებლებია დაპროექტების მართვაზე, უნდა იყვნენ დარწმუნებულნი, რომ უზრუნველყოფილია შემდეგი:

1. კარგი ურთიერთკავშირის არსებობა სხვადასხვა ქმედებებს შორის დაპროექტების ჩარჩოებში და სხვა ნაწილებთან, მათ შორის IT და ბიზნესის გეგმებთან და სტრატეგიებთან;
2. ბიზნესის ბოლო გეგმისა და სტრატეგიის ვერსიათა წვდომა მათთვის, ვინც მონაწილეობს დაპროექტებაში;
3. საპროექტო დოკუმენტაციის შესაბამისობა ბიზნესის და IT-ის გეგმებთან და სტრატეგიებთან;
4. არქიტექტურები და დიზაინი არის:
   - მოქნილი, ამგვარად, შეუძლიათ სწრაფი რეაგირება ბიზნესის ახალ მოთხოვნილებებზე;
   - ინტეგრირებული ბიზნესის და IT-ის ყველა სტრატეგიასა და პოლიტიკასთან;
   - სერვისის სასიცოცხლო ციკლის სხვა სტადიების მოთხოვნილებათა მხარდამჭერი;
   - თანამოქმედებენ ახალი სერვისების წინსვლისთვის ან არსებულის ცვლილებებისათვის, ბიზნესის შესაბამისი მოთხოვნილოებებისათვის.

დაპროექტების ერთ-ერთი ქვეეტაპია ბიზნესისა და მისი დრაივერების მოთხოვნილებათა განსაზღვრა და შემდგომი დოკუმენტირება. დრაივერების ქვეშ იგულისხმება რომელიღაც მოძრავი ბიზნესფაქტორი: ადამიანები, ინფორმაცია და ამოცანები, რომლებიც უზრუნველყოფენ დასმული მიზნების მიღწევას. დაპროექტების პროცესების რეგულირებისთვის ინფორმაცია იყოფა ორ კატეგორიად:

1. ინფორმაცია მოთხოვნების შესახებ არსებული სერვისებისათვის – რა ცვლილებებია საჭირო არსებულ სერვისებში, გათვალისწინებით:
   - ახალი ფუნქციური შესაძლებლობებისა და მოთხოვნების;
   - ცვლილებები ბიზნესპროცესებში, დამოკიდებულებებში, პრიორიტეტებში, გავლენასა და კრიტიკულობაში;
   - ცვლილებები სერვისის ტრანზაქციების მოცულობაში. **ტრანზაქცია (Transaction)** - ესაა დისკრეტული ფუნქცია, შესრულებადი IT-სერვისის მიერ. მაგალითად, ფულის გადარიცხვა ერთი საბანკო ანგარიშიდან მეორეზე. ერთი ტრანზაქცია შეიძლება შეიცავდეს მონაცემთა მრავალ დამატებას, წაშლას და ცვლილებას. ამ დროს ყველა უნდა დასრულდეს წარმატებით, საწინააღმდეგო შემთხვევაში მათგან არც ერთი არ იქნება შესრულებული (ანუ მთლიანი ტრანზაქცია იქნება გაუქმებული);

- სერვისის დონეებისა და მისი მიზნობრივი მაჩვენებლების ამაღლებისას ბიზნესის ახალ დრაივერთან დაკავშირებით ან შემცირება ძველი სერვისაებისათვის, რომლებიც მალე იქნება ჩანაცვლებული;
- მოთხოვნილებების – სერვისების მართვის პროცესების დამატებითი ინფორმაციის შესახებ.

2. ინფორმაცია მოთხოვნების შესახებ ახალი სერვისებისათვის:

- მოთხოვნილი ფუნქციურ;
- მენეჯმენტის ინფორმაცია და სხვა მოთხოვნილებანი;
- მხარდაჭერილი ბიზნეს–პროცესი, დამოკიდებულებები, პრიორიტეტები, გავლენა და კრიტიკულობა;
- სერვისების დონის მოთხოვნები და მიზნობრივი მაჩვენებლები;
- ბიზნესის ტრანზაქციის დონეები, სერვისების ტრანზაქციის დონეები, მომხმარებელთა რაოდენობა და მისი სავარაუდო ზრდა, მომხმარებელთა ტიპები;
- ფინანსური და სტრატეგიული დასაბუთება ბიზნესისათვის;
- ვარაუდი სამომავალო ცვლილებების, ანუ ბიზნესის ცნობილი მომავალი მოთხოვნების შესახებ ან ზრდის ტემპის ამაღლების შესახებ;
- სიმძლავრის დონე ბიზნესისათვის, რომელიც უნდა იყოს უზრუნველყოფილი.

ესაა ინფორმაციის მინიმალური ნაკრები დაპროექტების ეტაპის დაწყებისათვის. მისი სიზუსტე და აკურატულობა პირველხარისხოვანია. თუ გამოყენებული პროექტირების ეტაპზე არაკორექტული ან არასწორი ინფორმაცია იქნება, მაშინ საბოლოოდ ბიზნესის მოთხოვნილებებს დამუშავებული სერვისი ვერ დააკმაყოფილებს.

მოთხოვნები სერვისებთან უნდა იყოს დოკუმენტირებული. დრო, დახარჯული ამისთვის, იქნება კომპენსირებული მომავალში კამათის, დისკუსიებისა და უთანხმოების არარსებობით სერვისის მიმწოდებელსა და დამკვეთს შორის. ბიზნესის მოთხოვნების განსაზღვრის სტადია მდგომარეობს შემდეგში:

1. პროექტის მენეჯერის დანიშვნა, საპროექტო გუნდის შექმნა და ხელმძღვანელობის დამტკიცება ფორმალური და სტრუქტურირებული მეთოდოლოგიის გამოყენებით;
2. იდენტიფიკაცია ყველა დაინტერესებული პირის, შესაბამისი დოკუმენტაციის შედგენა მათი მოთხოვნებით და სარგებლიანობით, რომლებსაც ისინი მიიღებენ პროექტის რეალიზაციით;
3. ანალიზი, დოკუმენტირება, პრიორიტეტების განლაგება და მოთხოვნების შეთანხმება;
4. ბიზნესის ბიუჯეტის/სარგებლის გათვლა და დამტკიცება;
5. პოტენციალური კონფლიქტების გადაწყვეტა ბიზნეს–ერთეულებს შორის და კორპორატიული მოთხოვნების შეთანხმება;

6. პროცესების განსაზღვრა მოთხოვნების დასამტკიცებლად და დამტკიცებულის შესაცვლელად;

7. ურთიერთმოქმედების გეგმის განვითარება დამკვეთთან, ძირითად დამოკიდებულებათა ხაზგასმა, ხარჯები მიმდინარე მომსახურებაზე და IT-ს შორის, და ის, თუ როგორ მოხდება ამ დამოკიდებულებებისა და აუცილებელი კავშირების დაინტერესებულ მხარეებს შორის მართვა.

მას შემდეგ, რაც მოთხოვნები შეთანხმებული და დამტკიცებულია, მათ გამოუჩნდებათ „შემფასებლები", ანუ შესაძლებელია კონკრეტული პროექტის ღირებულების გათვლა. საჭიროა ბალანსის დაცვა, რაც ორგანიზაციას შეუძლია თავის თავზე აიღოს, და რაც მას უნდა. ზოგიერთი მოთხოვნის რეალიზაცია ძალზე ძვირია, ამიტომ ისინი უნდა ამოშალის უკვე პაროექტირების ეტაპზე. ეს საკითხები უნდა დოკუმენტირდეს და შეუთანხმდეს ბიზნესის წარმომადგენელს. ჩვეულებისამებრ, სირთულეები წარმოიშობა ბიზნესის სურვილსა და გამოყოფილ ბიუჯეტს შორის, რომელშიც არაა ასახული სერვისის სრული ღირებულება.

მაგალითად, პროექტირებისას გამოყენებული არქიტექტურები და დიზაინები უნდა იყოს ცხადი, ლაკონიური, მარტივი და დასაბუთებული. სამწუხაროდ, ისინი ხშირად ძალზე რთულია და აქვთ თეორიული ხასიათი, და ამგვარდ ცუდად გამოყენებადია პრაქტიკაში.

7.8. სერვისების დაპროექტების ასპექტები

სერვისების დასაპროექტებლად გამოიყოფა ხუთი ძირითადი ასპექტი:

1. გადაწყვეტათა დაპროექტება, მათ შორის ყველა საჭირო და შეთანხმებული ფუნქციონალური მოთხოვნების, რესურსებისა და შესაძლებლობების;

2. მხარდამჭრი მმართველი სისტემებისა და ინსტრუმენტების დაპროექტება, კერძო, სერვისების პორტფელის სერვისების მართვისა და კონტროლისათვის მათი სასიცოცხლო ციკლის ჩარჩოებში;

3. ტექნოლოგიების, მართვის სისტემებისა და ინსტრუმენტების დაპროექტება, რომლებიც აუცილებელია სერვისების უზრუნველსაყოფად;

4. პროცესების დაპროექტება, რომლებიც აუცილებელია სერვისების დიზაინის ასაგებად, დასანერგად, ექსპლუატაციისა და სრულყოფისათვის;

5. მეთოდებისა და მეტრიკების დაპროექტება სერვისების ხარისხის, ეფექტურობისა და მწარმოებლურობის გასაზომად, არქიტექტურისა და პროცესების.

რა თქმა უნდა, დაპროექტების საკვანძო ასპექტია გადაწყვეტათა დამუშავება, რომლებიც დააკმაყოფილებს ბიზნესის მოთხოვნილებებს. ყოველთვის, ახალი სერვისის ფორმირებისას, ის უნდა შემოწმდეს ყველა ზემოაღწერილ პუნქტში. ესაა გარანტია იმისა, რომ იგი კარგად იმუშავებს სხვა სერვისებთან ერთად.

დეტალური განხილვა შესაძლებელია [10] .

7.9. ტექნოლოგიების არქიტექტურის დაპროექტება

ტერმინს „არქიტექტურა“ აქვს განსხვავებული ინტერპრეტაცია კონტექსტისაგან დამოკიდებულებით. აქ **არქიტექტურა** – სისტემის ფუნდამენტური სტრუქტურაა, რომელიც ასახავს მის კომპონენტებს, მათ ურთიერთქმედაბას ერთმანეთთან და სისტემის ექსპლუატაციის პირობებს, აგრეთვე პრინციპებს, რომლებიც საფუძველია სისტემის დაპროექტებისა და განვითარებისა.

„სისტემის“ ქვეშ აქ იგულისხმება არა მხოლოდ სისტემა IT კონტექსტში. **სისტემა** – კომპონენეტების ერთობლიობაა, რომელიც ორგანიზებულია სპეციფიური ფუნქციის ან ფუნქციათა ერთობლიობის უზრუნველსაყოფად.

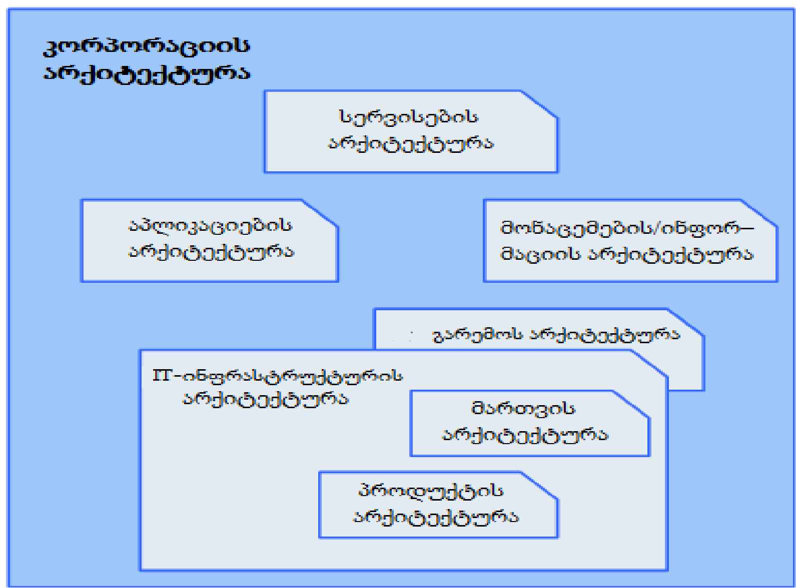
სისტემის სახით მოცემულ კონტექსტში შეიძლება განხილულ იქნას ორგანიზაცია მთლიანად, ბიზნესფუნქცია, ინფორმაციული სისტემა და ა.შ. არქიტექტურის დაპროექტების არსი მდგომარეობს პოლიტიკების, სტრატეგიების, არქიტექტურების, დიზაინების, დოკუმენტების, გეგმების და IT- პროცესების განვითარებასა და მხარდაჭერაში, ორგანიზაციისათვის შესაფერისი სერვისებისა და გადაწყვეტების დანერგვისა და შემდგომი ექსპლუატაციის მიზნით.

არქიტექტურის დაპროექტების შემავალი მონაცემებია ბიზნესისა და სტრატეგიის აგების ეტაპის გეგმები, სტრატეგიები და პოლიტიკები. დამპროექტებლების ამოცანაა დიზაინების, გეგმების, პოლიტიკების და არქიტექტურების სრულყოფა და განვითარება. ეს პროცესი განიხილავს აგრეთვე პასუხისმგებლობათა და როლების განაწილებას, სერვისებს, ტექნოლოგიებს, არქიტექტურებს, პროცესებს და პროცედურებს, პარტნიორებს და მიმწოდებლებს, მართვის მეთოდებს.

არქიტექტურის დაპროექტება მოიცავს ასევე ყველა საკითხს ტექნოლოგიებთან დაკავშირებით, მათ შორის ინფრასტრუქტურას, გარემოს, აპლიკაციებსა და მონაცემებს.

როგორც უკვე აღინიშნა, სისტემად შეიძლება მთლიანი ორანიზაციის განხილვა. იგი რთული სისტემაა კომპონენტების სიმრავლით: პერსონალი, ბიზნესფუნქციები, პროცესები, ორგანიზაციული სტრუქტურა, ინფორმაციული რესურსები, ფინანსური რესურსები, სტრატეგიები, მართვის სისტემები და ა.შ.

კორპორაციის არქიტექტურა უნდა უჩვენებდეს, თუ როგორ ურთიერთქმედებენ ერთმანეთთან ეს კომპონენტები საერთო კორპორაციული მიზნის მისაღწევად. ITIL განიხილავს კორპორაციის არქიტექტურას ბიზნესის, რომელსაც ის ეწევა, და გამოყენებული საინფორმაციო სისტემების კონტექსტში (ნახ.7.19).



ნახ.7.19. კორპორაციის არქიტექტურა

კორპორაციის არქიტექტურა უნდა შედგებოდეს შემდეგი ძირითადი არქიტექტურებისაგან:

1. **სერვისების არქიტექტურა** – გადაჰყავს აპლიკაციები, ინფრასტრუქტურები, ქმედებათა ორგანიზაცია და მხარდაჭერა სერვისების ერთობლიობაში. სერვისების არქიტექტურა არის დამოუკიდებელი, ბიზნესში ინტეგრირებული მიდგომა ბიზნესისთვის სერვისების მისაწოდებლად. იგი იძლევა მოდელს დაყოფისათვის სერვისების არქიტექტურას, აპლიკაციების არქიტექტურას, ინფრასტრუქტურის არქიტექტურას და მონაცემთა არქიტექტურას შორის. სერვისების არქიტექტურის ჩარჩოებში ასევე განიხილება საკითხები მტყუნებებისადმი სტაბილურობის უზრუნველყოფის, შემდგომი კორექტირებისა და უსაფრთხოების უზრუნველყოფის შესახებ;

2. **აპლიკაციების არქიტექტურა** – იძლევა დეტალურ გეგმას ინდივიდუალური აპლიკაციების განვითარებისა და მიწოდების შესახებ, ასახავს ბიზნესის ფუნქციურ მოთხოვნებს აპლიკაციასთან და უჩვენებს ურთიერთდამოკიდებულებას აპლიკაციებს შორის. კომპონენტებზე დაფუძნებული მიდგომა მაქსიმალურს ხდის მათ ხელმეორედ გამოყენებას და ეხმარება აპლიკაციებს მოქნილობაში მომარაგების ცვლადი პოლიტიკების პირობებში.

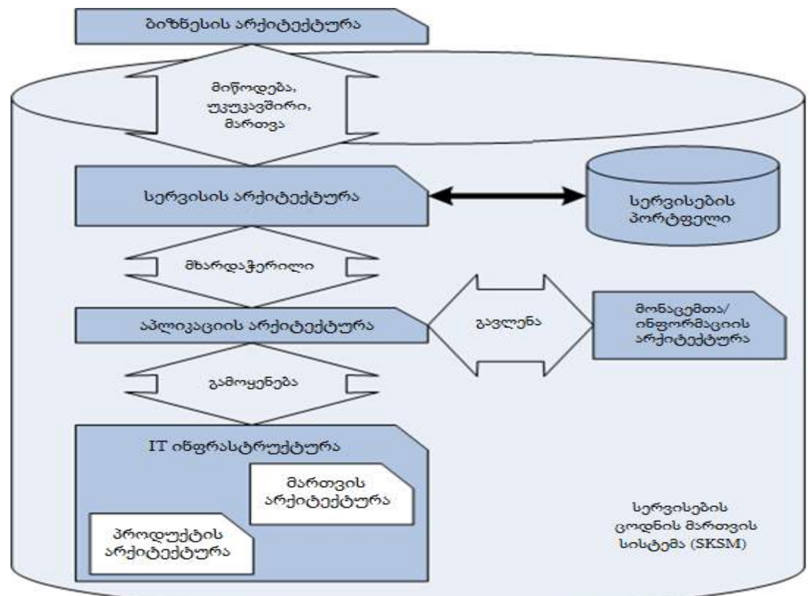
მონაცემთა / ინფორმაციის არქიტექტურა – აღწერს ორგანიზაციის ლოგიკურ და ფიზიკურ ინფორმაციულ აქტივებს და მათი მართვის რესურსებს. იგი უჩვენებს, თუ როგორაა განაწილებული ინფორმაციული რესურსები და მათი მართვა კორპორაციული მიზნის მისაღწევად;

3. **ინფრასტრუქტურის არქიტექტურა** – აღწერს სტრუქტურას, ფუნქციურობას და პროგრამულ/აპარატურული უზრუნველყოფის გეოგრაფიულ განაწილებას, კომუნიკაციის კომპონენეტებს, ასევე მათთან დაკავშირებულ სტანდარტებს;

4. **გარემოს არქიტექტურა** – აღწერს გარემოს ასპექტებს, დონეებს და კონტროლის ტიპებს, აგრეთვე მათი მართვის საკითხებს.

აღწერილი არქიტექტურების ურთიერთკავშირი მოცემულია 7.20 ნახაზზე.

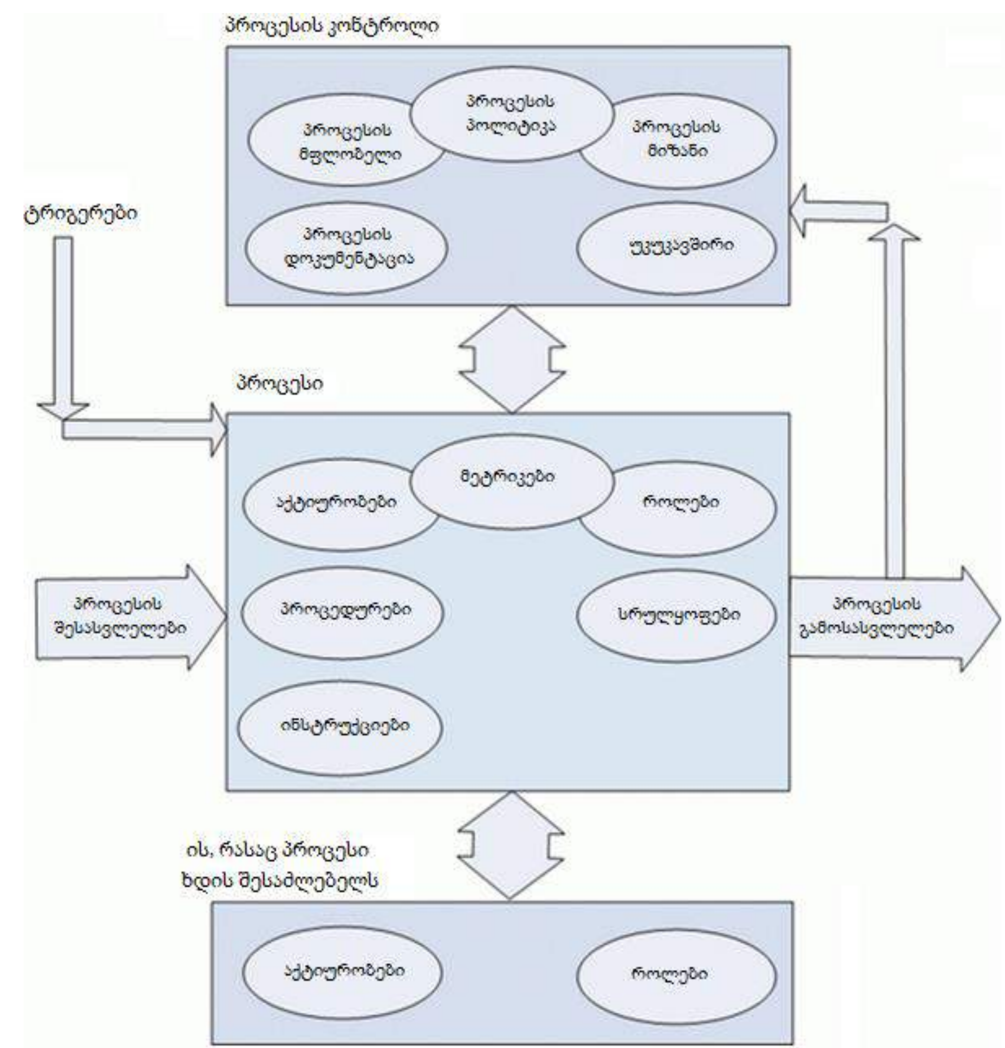
პროცესი – ქმედებათა სტრუქტურირებული ერთობლიობაა, დაპროექტებული სპეციფიკური მიზნის მისაღწევად. პროცესი გარდაქმნის ერთ ან რამდენიმე შესასვლელს განსაზღვრულ გამოსასვლელებში. პროცესის განსაზღვრება მოიცავს ყველა როლს, პასუხისმგებლობის განაწილებას, ინსტრუმენტებს და კონტროლს, აუცილებელს მოლაპარაკებული შედეგების საიმედო მიწოდებაზე.



ნახ.7.20. არქიტექტურათა ურთიერთკავშირი

7.10. პროცესების დაპროექტება

ერთხელ განსაზღვრული პროცესები უნდა კონტროლირდებოდეს და იყოს მართვადი. პროცესების კონტროლი – ესაა ქმედება, დაკავშირებული პროცესის დაგეგმვასა და რეგულირებასთან, პროცესის წარმოდგენის მიზნით ეფექტური, რაციონალური და სტაბილური სახით. მხოლოდ კონტროლის დონეების განსაზღვრის შემდეგ შეიძლება განისაზღვროს კონტროლის ეფეტურობის გამზომი სისტემა შესაბამისი მეტრიკებით (ნახ.7.21).



ნახ.7.21. პროცესის ელემენტები

პროცესი ყოველთვის იქმნება განსაზღვრული მიზნების მისაღწევად. პროცესის გამოსასვლელები უშუალოდ უნდა იყოს დამოკიდებული ამ მიზნებისაგან.

დაპროექტებისას მნიშვნელოვანია შეფასებისა და მეტრიკის სისტემის დამუშავება პროცესის გამოსასვლელების, ანგარიშგების და სრულყოფისათვის.

ყველა პროცესს ჰყავს მფლობელი, პასუხისმგებელი მასზე და მის სრულყოფაზე, და რომ პროცესი უზრუნველყოფს თავის მიზნების მიღწევას.

მიზნები უნდა იყოს გაზომვადი და აღიწერებოდეს ბიზნესისათვის სასარგებლო ტერმინებით, მიღებული პოლიტიკების და სტრატეგიების გათვალისწინებით. დაპროექტების ეტაპზე ყველა პროცესს დაენიშნება მფლობელი.

პროცესების გამოსასვლელები უნდა შეესაბამებოდეს რაღაც ოპერაციული ნორმების ერთობლიობას, რომლის წყარო იქნება ბიზნესის მიზნები. თუ პროცესის შედეგები შეესაბამება ნორმებს, მას შეიძლება ვუწოდოთ ეფექტური (იმიტომ რომ იგი შეიძლება გამეორდეს, რადგან გაზომვადი და მართვადია).

პროცესი ასევე ეფექტურად მოიხსენება, თუ იგი იყენებს რესურსების მინიმალურ ერთობლიობას. პროცესის გაზომვის და ანალიზის შედეგები შესაბამისის მეტრიკებით უნდა აისახოს მმართველ ანგარიშებში და უნდა მიეწოდოს უწყვეტი სრულყოფის პროცესის შესასვლელს.

პროცესი არის ITIL–ის საფუძველი. აუცილებელი შესასვლელების და გამოსასვლელების განსაზღვრა პროცესებისათვის ორგანიზაციის შიგნით იძლევა შესაძლებლობას უფრო ეფექტურად და რაციონალურად იმართოს იგი.

ნორმების დადგენა პროცესებისთვის იძლევა მისი მუშაობის ხარისხის გაზომვის შესაძლებლობას. ნორმები განსაზღვრავს კონკრეტულ პირობებს, რომელთაც უნდა შეესაბამებოდეს პროცესის შედეგები. ნებისმიერი პროცესის დაპროექტების დაწყების წინ მნიშვნელოვანია წარმოდგენა, თუ მისი გამოსასვლელები როგორ გამოიყურება. ყოველი ორგანიზაცია უნდა იყენებდეს ფორმალიზებულ მიდგომას სერვისმენეჯმენტის პროცესების დაპროექტებისა და რეალიზაციისთვის.

არაა საჭირო „იდეალური პროცესებისა" შექმნისაკენ სწრაფვა. მნიშვნელოვანია ორგანიზაციისათვის პრაქტიკული და გამოყენებადი პროცესების დაპროექტება, შემდგომი სრულყოფის შესაძლებლობებით. ერთ–ერთი მიმართულება პროცესორული განვითარების მიდგომისა არის ინსტრუმენტებისა და სტანდარტების შექმნა, რომლებიც საშუალებას მოგვცემს პროცესების ინტეგრაციისათვის, რომელიც სხვადასხვა ორგანიზაციებს ეკუთვნის.

ამის მაგალითია ღია სტანდარტი DMTF, რომელიც ეფუძნება ITIL კონცეფციას. იგი აფორმალიზებს ინფორმაციის გაცვლას ინციდენტების, პრობლემებისა და ცვლილებების შესახებ პროცესებს შორის [102].

7.11. სერვისების უწყვეტობის მართვა და რისკების შეფასება

სერვისების უწყვეტობის მართვა (IT Service Continuity Management ან ITSCM) - ესაა პროცესი, პასუხისმგებელი *რისკების მართვაზე*, რომლებიც გავლენას ახდენს სერვისებზე. ITSCM უზრუნველყოფს შესაძლებლობას, რომ სერვისების მიმწოდებელს მუდმივად მიეცეს სერვისების მინიმალურად შეთანხმებული დონე, რისკების შემცირების გზით მისაღებ დონემდე, აგრეთვე სერვისების აღდგენის დაგეგმვის შესაძლებლობა [11].

სერვისების უწყვეტობის მართვის ძირითადი მიზანია ბიზნესის უწყვეტობის მართვის პროცესის მხარდაჭერა. **ბიზნესის უწყვეტობის მართვა (Business Continuity**

Management ან BCM) - ესაა ბიზნესპროცესი, პასუხისმგებელი რისკების მართვაზე, რომელთაც შეუძლია სერიოზული გავლენა მოახდინოს ბიზნესზე.

BCM იცავს საკვანძო დაინტერესებულ მხარეებს, რეპუტაციას, ბრენდს და ქმედებას ფასეულობის შექმნის მიზნით. *BCM* პროცესი მოიცავს რისკების შემცირებას მისაღებ დონემდე და ბიზნეს–პროცესების აღდგენის ხერხების დაგეგმვას ბიზნესის დარღვევის შემთხვევაში. *BCM* ადგენს მიზნებს, საზღვრებს და მოთხოვნებს IT-სერვისის უწყვეტობის მართვასთან მიმართებით.

სინამდვილეში ტექნოლოგია არის ძირითადი კომპონენტი მრავალი ბიზნეს-პროცესის, ამიტომაც მათი უწყვეტობის და წვდომის უზრუნველყოფა აუცილებელია ბიზნესის არსებობისათვის მთლიანად. ITSCM მართავს სერვისებისა და მათი კომპონენტების აღდგენისათვის. ITSCM –ის შუალედური მიზნებია:

1. მართვა – სერვისების უწყვეტობის უზრუნველყოფისა და სერვისების აღდგენის გეგმების ერთობლიობისა, რომლებიც ბიზნესის უწყვეტობის უზრუნველყოფის გეგმების ნაწილია. **სერვისების უწყვეტობის უზრუნველყოფის გეგმა (IT Service Continuity Plan)** – ესაა გეგმა, რომელიც განსაზღვრავს ბიჯებს ერთი ან რამდენიმე სერვისის აღსადგენად. გეგმამ ასევე უნდა განსაზღვროს მოვლენები, რომლებიც არის საფუძველი: მისი ინიციაციისათვის, ადამიანებისათვის, რომლებიც უნდა ამოქმედდნენ, კომუნიკაციის საშუალებებისათვის და ა.შ.

ბიზნესის უწყვეტობის უზრუნველყოფის გეგმა (Business Continuity Plan ან BCP) - ესაა გეგმა ბიჯების განმსაზღვრელი, რომლებიც აუცილებელია ბიზნესპროცესების აღსადგენად მათი ფუნქციის დარღვევის შემთხვევაში. გეგმა ასევე უნდა შეიცავდეს ინფორმაციას მოვლენების შესახებ, რომლებიც არის საფუძველი: მისი ინიციაციისათვის, ადამიანებისათვის, რომლებიც უნდა ამოქმედდნენ, კომუნიკაციის საშუალებებისათვის და ა.შ.

2. ბიზნესზე გავლენის ანალიზისა დამთავრება უწყვეტობის უზრუნველყოფის გეგმის მართვის გარანტიის ნაწილში ცვალებადი მოთხოვნებისა და ბიზნესის საჭიროების შესაბამისად;

3. რისკების ანალიზის და მენეჯმენტის თანხლება, კერძოთ ბიზნესის ურთიერთქმედებისას წვდომის და უსააფრთხოების მართვის პროცესებთან, რომლებიც მართავენ სერვისებს შესაბამისად სერვისების შეთანხმებული დონის მიხედვით;

4. რეკომენდაციებისა და სახელმძღვანელოების წარმოდგენა IT–ის სხვა სფეროებისთვის საკითხეზში, რომლებიც დაკავშირებულია სერვისების უწყვეტობასა და აღდგენასთან;

5. უწყვეტობისა და აღდგენის მექანიზმების უზრუნველყოფა, რომლებიც ეხმარება მას ბიზნესის მიერ დადგენილი მიზნობრივი მაჩვენებლების მიღწევაში;

6. ცვლილებების გავლენის შეფასება სერვისების უწყვეტობის უზრუნველყოფის გეგმებზე და სერვისების აღდგენის გეგმებზე;

7. პროაქტიური სრულყოფა სერვისების უწყვეტობისა იქ, სადაც ეს ეკონომიკურად ეფექტურია;

8. მოლაპარაკებების წარმართვა და კონტრაქტების დადება მიმწოდებლებთან აღდგენის აუცილებელი შესაძლებლობის უზრუნველყოფის შესახებ, უწყვეტობის მხარდაჭერის მიზნით (მიმწოდებლების მართვის პროცესის მონაწილეობით).

უწყვეტობის მართვა ფოკუსირდება მნიშვნელოვან ნეგატიურ მოვლენებზე, რომლებსაც *ITIL* მოიხსენიებს ბიზნესის „კატასტროფების" ტერმინით. უფრო ნაკლებმნიშვნელოვანი მოვლენები განიხილება ინციდენტების მართვის პროცესის ფარგლებში. არის თუ არა რომელიმე კონკრეტული მოვლენა კატასტროფა, დამოკიდებულია ოპგანიზაციაზე, რომელშიც ის მოხდა.

ზომა და მნიშვნელობა მოვლენის ნეგატიური გავლენისა ბიზნესზე, მაგალითად, ფინანსური დანაკარგი ან რეპუტაციის დაკარგვა, იზომება ბიზნესზე გავლენის ანალიზის ფარგლებში. ბიზნესზე გავლენის ანალიზი განსაზღვრავს მინიმალურ მოთხოვნებს კრიტიკულობასთან, კონკრეტული მოთხოვნები ტექნოლოგიებთან და სერვისებთან განისაზღვრება უწყვეტობის მართვის ფარგლებში.

ITSCM უმთავრესად განიხილავს IT აქტივებს და კონფიგურაციებს, რომლებიც ბიზნესპროცესების მხარდამჭერებია. კატასტროფის შემთხვევაში ბიზნესი უნდა გადაეწყოს ალტერნატიულ სამუშაო ლოკაციაზე. ამ დროს აუცილებელია ისეთი ელემენტების წარმოდგენა, როგორიცაა ოფისის კომფორტი თანამშრომელთათვის, კრიტიკული ქაღალდის ანგარიშების დუბლიკატები, კურიერების სერვისი და სატელეფონი კავშირები კლიენტებთან და პარტნიორებთან. ამ მხრივ უწყვეტობის მართვამ უნდა გაითვალიწინოს ორგანიზაციის ოფისების რაოდენობა და ადგილმდებარეობა, ასევე სერვისები თითოეულში.

უწყვეტობის მართვის ფარგლებში უნდა შესრულდეს შემდეგი ქმედებები:

1. ITSCM–ის და გამოყენებული პოლიტიკების საზღვრების შეთანხმება;

2. ბიზნესზე გავლენის ანალიზი – სერვისის დანაკარგების ბიზნესზე გავლენის რაოდენობრივი შეფასებისათვის;

3. რისკების ანალიზი - იდენტიფიკაცია და შეფასება რისკებისა უწყვეტობის პოტენციალური საფრთხეების განსაზღვრის და მათი განხორციელების ალბათობის შეფასების მიზნით. აქ შედის ასევე საფრთხეების მართვის მექანიზმების გამოყენება იქ, სადაც ეს ეკონომიკურად ეფექტური იქნება;

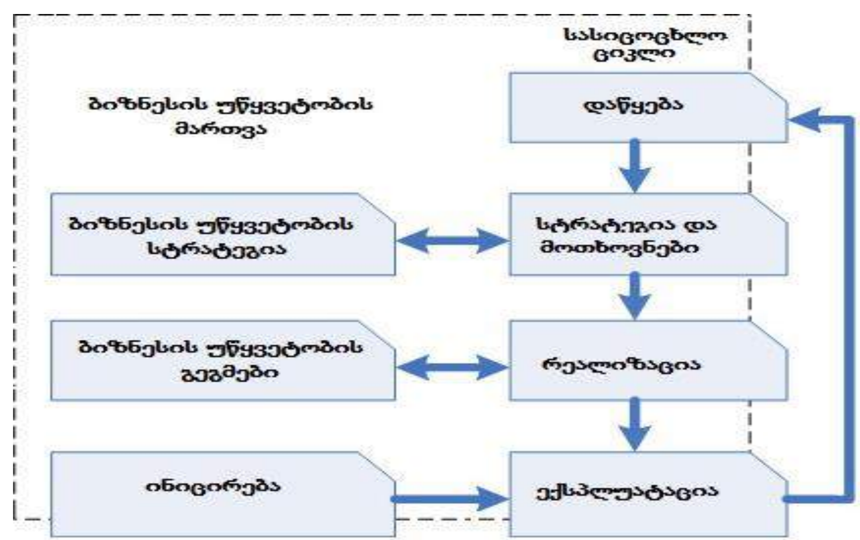
4. ITSCM სტრატეგიის ფორმირება, ინტეგრირებული *BCM* სტრატეგიაში.

5. უწყვეტობის უზრუნველყოფის გეგმების ფორმირება, ინტეგრირებული *BCM* გეგმებში.

6. უწყვეტობის უზრუნველყოფის გეგმების ტესტირება;

7. გეგმების უწყვეტი შესრულება და მათი მართვა.

7.22 ნახაზზე მოცემულია ITSCM-ის სასიცოცხლო ციკლი. ITSCM ციკლურად მეორდება სერვისის მთელ სასიცოცხლო ციკლზე და ის იძლევა გარანტიას, რომ ერთხელ შემუშავებული გეგმები სერვისების აღდგენისა და უწყვეტობის შესახებ შესაბამისობაში იყოს მომავალში ბიზნესის პრიორიტეტებთან და ბიზნესის უწყვეტობის უზრუნველყოფის გეგმებთან. ნახაზზე ნაჩვენებია *BCM* –ის როლი ITSCM–ში.



ნახ.7.22. ITSCM –ის სასიცოცხლო ციკლი

ინიციალიზაციისა და მოთხოვნილებათა ფორმირების სტადიები ეკუთვნის *BCM* -ს. აქ ITSCM მხოლოდ უნდა მონაწილეობდეს ამ სტადიებში, რათა მხარი დაუჭიროს *BCM* -ს, გაიგოს კავშირები ბიზნესპროცესებს შორის და სერვისების დანაკარგების გავლენას მათზე. ამ საწყისი სტადიების შედეგად *BCM* აფორმირებს ბიზნესის უწყვეტობის უზრუნველყოფის სტრატეგიას. ITSCM–თვის პირველი რიგის სერიოზული ამოცანაა თავისი სტრატეგიის ფორმირება, რომელიც შესაძლებელს გახდის და მხარს დაუჭერს ბიზნესის უწყვეტობის სტრატეგიას. განვიხილოთ ITSCM–ის სასიცოცხლო ციკლის სტადიები.

- **სტადია 1 – დაწყება**

ITSCM-ის ეს სტადია მოიცავს შემდეგ ქმედებებს:

- უწყვეტობის უზრუნველყოფის პოლიტიკის ფორმირება – უნდა განხორციელდეს რაც შეიძლება სწრაფად. პოლიტიკამ, მინიმუმ უნდა განსაზღვროს მიზნები, მომენტები და საკითხები, რომლებსაც მენეჯმენტმა უნდა მიაქციოს ყურადღება;
- საზღვრების და კომპეტენციების ტერმინების განსაზღვრა – TSCM–ის საზღვრების დადგენა და პასუხისმგებლობათა განაწილება მთელ პერსონალზე ორგანიზაციაში;

- რესურსების განაწილება – გარემოს ფორმირება ბიზნესის უწყვეტობის უზრუნველსაყოფად, რომელიც მოითხოვს მნიშვნელოვან ფინანსურ და ადამიანურ რესურსს;
- პროექტის განსაზღვრა ITSCM პროცესის ორგანიზების და მისი კონტროლის სტრუქტურისა – ITSCM და *BCM* რთული პროცესებია, რომლებიც თხოულობს ფრთხილ ორგანიზებას და კონტროლს;
- პროექტის და ხარისხის გეგმების შეთანხმება - გეგმები უზრუნველყოფს პროექტის კონტროლს და მის გამ ოყენებას განსხვავებულ სიტუაციაში.

➢ **სტადია 2 – მოთხოვნილებანი და სტრატეგია**

ბიზნესის მოთხოვნების დადგენა სერვისების უწყვეტობაზე კრიტიკულად მნიშვნელოვანია, რადგან სწორედ ამ ეტაპზეა დამოკიდებული ორგანიზაციის მდგრადობა კატასტროფებისადმი და შესაბამისი დანახარჯები. თუ მოთხოვნები არაკორექტულია ან გაიპარა რამე მნიშვნელოვანი ინფორმაცია, მაშინ ITSCM–ის ყველა მექანიზმი იქნება არაეფექტური. ეს სტადია იყოფა ორ ქვესტადიად:

- მოთხოვნები – ბიზნესზე გავლენის ანალიზი და რისკების შეფასება;
- სტრატეგია – სტრატეგია აყალიბებს რისკის შემცირების ზომებს და აღდგენის ოფციებს.

ბიზნესზე გავლენის ანალიზი (Business Impact Analysis ან BIA) – ესაა ქმედება ბიზნესის უწყვეტობის მართვის პროცესის ფარგლებში, რომელიც განსაზღვრავს კრიტიკულ ბიზნესფუნქციებს და მათ დამოკიდებულებას გარემოს ფაქტორებიდან. ეს ფაქტორები შეიძლება იყოს მიმწოდებლები, ადამიანები, სხვა ბიზნესპროცესები, სერვისები და ა.შ. *BIA* განსაზღვრავს სერვისების დანაკარგების შედეგებს ბიზნესზე. დანაკარგები შეიძლება იყოს მნიშვნელოვანი, მაგალითად, დიდი ფინანსური დანაკარგები, „რბილი“ – მორალური დანაკარგები, რეპუტაციის დანაკარგები, კონკურენტული უპირატესობის დანაკარგები და ა.შ.

ბიზნესზე გავლენის ანალიზი განსაზღვრავს:

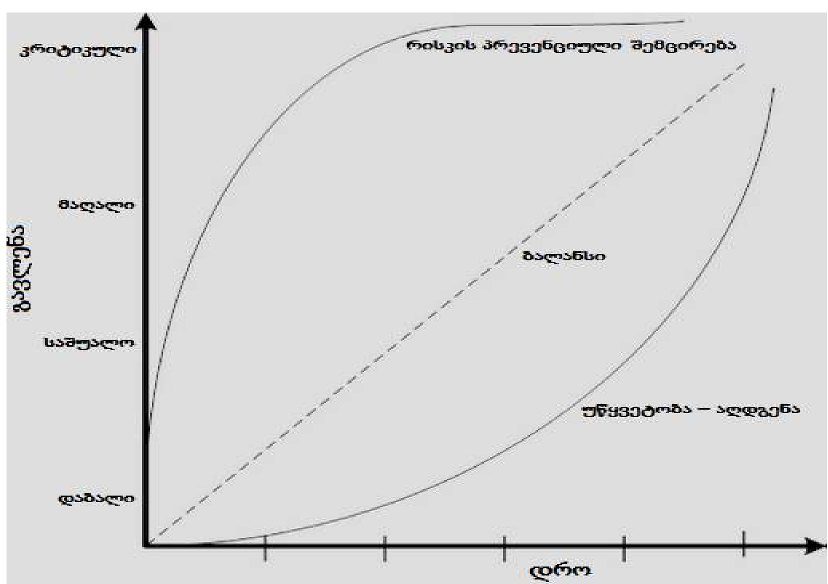
- ფორმას, რომელსაც მიიღებს განადგურება ან დანაკარგები, მაგალითად:
  - დაკარგული შემოსავალი;
  - დამატებითი ხარჯები;
  - რეპუტაციის შელახვა;
  - კეთილგანწყობილი კლიენტების დაკარგვა;
  - კონკურენტული უპირატესობის დანაკარგი;
  - ჯანმრთელობის, კანონიერებისა და უსაფრთხოების დაზიანება და დარღვევა;
  - პერსონალის უსაფრთხოების რისკი;
  - გასაღების ბაზრის დანაკარგი მოკლევადიან და გრძელვადიან პერიოდებში;
  - ოპერაციული შესაძლებლობების დანაკარგი, მაგალითად, კონტროლის.

- როგორ გაიზრდება ნეგატიური შედეგები განადგურების ან დანაკარგების არასასურველი მოვლენის შემდეგ, ასევე დრო დღე–ღამის, კვირის, თვის, როდესაც ისინი იქნება ყველაზე მნიშვნელოვანი;
- საკადრო უზრუნველყოფა, უნარები, აპარატურა და სერვისები, რომლებიც აუცილებელია კრიტიკული ბიზნეს–პროცესების უწყვეტობის მინიმალური დონეების მხარდასაჭერად;
- დროის ჩარჩოები, რომლის საზღვრებში აუცილებელია საკადრო უზრუნველყოფის, აპარატურის, სერვისების და სხვა შესაძლებლობათა აღდგენის მინიმალური დონის უზრუნველყოფა;
- დროის ჩარჩოები, რომლის საზღვრებში აუცილებელია მთლიანად აღდგეს კრიტიკული ბიზნესპროცესები და მათი მხარდამჭერი საკადრო უზრუნველყოფა, აპარატურა, სერვისები და სხვა შესაძლებლობები;
- აღდგენის პრიორიტეტები სერვისებისთვის.

BIA-ს ერთ-ერთი ძირითადი გამოსასვლელია სერვისის ან ბიზნესპროცესის დანაკარგების გავლენის შეფასების დიაგრამის აგება მთლიანად ბიზნესზე (ნახ.7.23).

ბიზნესზე გავლენის *ანალიზი* არის ბაზა ITSCM–ის განსახორციელებლად. ანალიზის საფუძველზე ფორმირდება სერვისების, აპლიკაციებისა და სხვა კომპონენტების სია, რომლებიც ხდება ITSCM-ის განხილვის საგნები.

მეორე ეტაპი ITSCM-ის მოთხოვნების განსაზღვრისა მდგომარეობს არასასურველი მოვლენების აღმოცენების ალბათობის შეფასებაში.



ნახ.7.23. ბიზნესზე გავლენის გრაფიკული წარმოდგენა

რისკების შეფასება (Risk Assessment) – ესაა რისკების მართვის საწყისი ბიჯები. ანალიზდება აქტივების ფასეულობები ბიზნესისათვის, იდენტიფიცირდება საფრთხეები ამ აქტივებთან მიმართებით, და ფასდება აქტივების დაუცველობა ამ საფრთხეებთან მიმართებით [11].

რისკების შესაფასებლად და მათ სამართავად გამოიყენება სტანდარტული მეთოდოლოგია M_o_R (*Management* of Risks), რომელიც შედგება შემდეგისგან:

- **M_o_R პრინციპები** - ბაზირებულია ორგანიზაციის მართვის პრინციპებზე და აუცილებელია რისკების ეფექტური მართვისათვის;
- **M_o_R მიდგომა** - ორგანიზაციის მიდგომა ზემოაღნიშნულ პრინციპებისადმი უნდა აისახოს რიგ დოკუმენტებში, კერძოთ რისკების მართვის პოლიტიკაში;
- **M_o_R პროცესები.** გამოყოფენ ოთხ პროცესს M_o_R–ის ფარგლებში:
  - განსაზღვრება - საფრთხეთა დეფინიცია ქმედებისათვის, რომლებსაც შეუძლია გავლენა იქონიოს გამიზნული შედეგის მიღწევაზე;
  - შეფასება - ყველა განსაზღვრული საფრთხის ჯამური გავლენის შეფასება;
  - დაგეგმვა - მმართველი ქმედებების განსაზღვრა, რომლების ამცირებს რისკებს;
  - რეალიზაცია - დაგეგმილი მმართველი ქმედებების განხორციელება, მათი კონტროლი, ეფექტურობის განსაზღვრა და კორექტირება აუცილებლობის შემთხვევაში.
- **M_o_R–ის გადასინჯვა და დანერგვა** – M_o_R–ის პროცესების, პოლიტიკის და მიდგომის დანერვა ისე, რომ ისინი უწყვეტად კონტროლდებოდეს და რჩებოდეს ეფექტურად;
- **ურთიერთმოქმედება** – ყველა ქმედების ურთიერთ-მოქმედების უზრუნველყოფა M_o_R–ის ფარგლებში ინფორმაციის აქტუალურობის მხარდასაჭერად საფრთხეების, შესაძლებლობების და რისკების მართვის სხვა ასპექტების შესახებ.

ქმედებები ITSCM-ის ფარგლებში უნდა იყოს მიმართული რისკების გავლენისა და მათი წარმოქმნის ალბათობის შემცირებაზე.

ბიზნესზე გავლენის ანალიზის შედეგები და რისკების შეფასება არის სერვისების უწყვეტობის სტრატეგიის საფუძველი ბიზნესის მოთხოვნილებების შესაბამისად. უმეტესი ორგანიზაცია უნდა იცავდეს ბალანსს რისკების შემცირებასა და აღდგენის მექანიზმების ფორმირებას შორის.

რაგინდ კარგად არ ტარდებოდეს ქმედებები რისკების შესამცირებლად, შეუძლებელია მათი მთლიანად აღმოფხვრა. ამიტომაც ყოველთვის აუცილებელია აღდგენის მექანიზმების დანერგვა ინტეგრაციაში წვდომის მართვის პროცესთან, რადგანაც სწორედ სერვისების წვდომა დაზარალდება პირველ რიგში, ბიზნესისთვის არასასიამოვნო მოვლენების აღმოცენების შემთხვევაში.

ტიპური ღონისძიებები რისკების შესამცირებლად შემდეგია:

- UPS-ის და სარეზერვო კვების ინსტალაცია კომპიუტერისათვის;

- სისტემების მტყუნებამდგრადობის უზრუნველყოფა კრიტიკული აპლიკაციებით, რომლებისთვისაც მიუღებელია ნებისმიერი მოცდენა (მაგალითად, საბანკო სისტემა);
- RAID-ის და სარკისებური დისკოების გამოყენება სერვერებისთვის, ინფორმაციის დაკარგვის თავიდან ასაცილებლად და მუშაობის უწყვეტობის უზრუნველსაყოფად;
- სათადარიგო კომპონენტების/მოწყობილობათა არსებობა, რომლებიც გამოყენებულ ინება ძირითადის მტყუნების შემთხვევაში. მაგალითად, სათადარიგო სერვერი მინიმალური აუცილებელი კონფიგურაციით, რომელიც ამუშავდება ძირითადის გამორთვისას;
- SPOF-ების გამორიცხვა, მაგალითად, ქსელში წვდომის ერთიანი წერტილი ან ელექტროკვების ერთიანი წერტილი;.
- საიმედო IT-სისტემების და ქსელების გამოყენება;
- სერვისების აუთსორსინგი რამდენიმე მიმწოდებლისათვის;
- უსაფრთხოებაზე კონტროლის გაზრდა;
- სერვისების მუშაობისა დარღვევების აღმოჩენის კონტროლის გაზრდა;
- ყოვლისმომცველი სტრატეგია აღდგენის და სარეზერვო დუბლირების, რომელიც მოიცავს გარე შენახვასაც. გარე შენახვა გულისხმობს კრიტიკული ინფორმაციის რეგულარულ დუბლირებას (ყოველდღიური) გარე საცავში.

ზემოჩამოთვლილი ზომები ვერ წყვეტს ITSCM-ის ყველა საკითხს, მაგრამ მათი გამოყენება საშუალებას იძლევა მნიშვნელოვნად შემცირდეს დანაკარების რისკი ბიზნესისათვის გაუთვალისწინებელ მდგომარეობათა აღმოცენების შემთხვევაში.

აღდგენის ოფციები ITSCM-ის ფარგლებში, რომლებიც უნდა იქნას გათვალისწინებული სგტრატეგიის ფორმირებისას, შემდეგია [11]:

- **გადასვლა ხელით მუშაობაზე** სერვისთა ზოგიერთი ტიპისთვის შეიძლება გახდეს კარგი ალტერნატივა მოკლე პერიოდში, სანამ აღადგენენ სერვისს. მაგალითად, სერვისდესკს შეუძლია მუშაობა გარკვეული დრო ქაღალდის განაცხადებთან და ჟურნალებთან;
- **ურთიერთშეთანხმება** არის აღდგენის კიდევ ერთი ოფცია. უნდა მოხდეს შეთანხმებების დადება ორგანიზაციებს შორის, რომლებიც იყენებენ მსგავს ტექნოლოგიებს. დღეისათვის არაა მისაღები უმეტესი IT-სისტემებისათვის, მაგრამ შეიძლება მათი ცალკეულ შემთხვევებში გამოყენება, მაგალითად, გარე სარეზერვო დუბლირებისთის ან პრინტერების გამოსაყენებლად.
- **თანდათანობითი აღდგენა (Gradual Recovery - RG)** - აღდგენის ხერხი, ცნობილია როგორც „ცივი რეზერვირება“. გაითვალისწინება სერვისის აღდგენა 72 საათზე მეტ დროში. თანდათანობითი აღდგენის დროს ამოქმედებულია მობილური ან სტაციონარული სარეზერვო ცენტრი, აღჭურვილი სიცოცხლის უზრუნველმყოფელი ელემენტებით და ქსელური გაყვანილობით, კომპიუტერული სისტემების გარეშე. აღდგენის ეს ოფცია რეკომენდებულია არაკრიტიკული სერვისებისათვის, რომელთა

უზრუნველყოფა შეიძლება შეჩერდეს დღეებით და კვირეებით, ბიზნესზე არამნიშვნელოვანი გავლენით;

- **შუალედური აღდგენა (Intermediate Recovery)** - აღდგენის ხერხი, ცნობილი სახელით „თბილი რეზერვირება“. გაითვალისწინება სერვისის აღდგენა 24-72 საათის განმავლობაში. შუალედური აღდგენისას, ჩვეულებისამებრ, გამოიყენება საერთო მობილური ან სტაციონარული სარეზერვო ცენტრი, აღჭურვილი კომპიუტერული სისტემებით და ქსელური კომპონენტებით. აპარატული და პროგრამული კონფიგურირება, ასევე მონაცემთა აღდგენა სრულდება სერვისების უწყვეტობის უზრუნველყოფის გეგმის ფარგლებში. აღდგენის ამ ოფციას ჩვეულებრივად სთავაზობენ მესამე მხარის ორგანიზაციები, რომლებსაც აქვთ ამ მიზნით ყველა აუცილებელი მოწყობილობა და კვალიფიციური პერსონალი. ამ ოფციის ღირებულება დამოკიდებულია მესამე მხარის რესურსებზე, რომლებიც უნდა ამოქმედდეს აღსადგენად, ასევე დროზე, რომლის განმავლობაშიც უნდა აღდგეს სერვისი. ამ მეთოდის უპირატესობაა გამჭვირვალეობა მომხმარებელთათვის. ნაკლოვანებაა ის, რომ ინფორმაცია (კონფიდენციალურიც) იქნება შენახული გარე ორგანიზაციაში. ეს უკანასკნელი კი აღდგენის ამ ხერხს არ იყენებს ბევრი ორგანიზაციისათვის;
- **სწრაფი აღდგენა (Fast Recovery)** - აღდგენის ხერხი. გაითვალისწინება სერვისის აღდგენა დროის მოლკლე შუალედში, 24 საათზე ნაკლებ დროში. სწრაფი ღდგენისას იყენებენ გამოყოფილ სტაციონარულ სარეზერვო ცენტრს კომპიუტერული სისტემებით და პროგრამული უზრუნველყოფებით, კონფიგურირებულს სერვისების მუშაობისთვის. დაუყოვნებლივი აღდგენა მოიცავს 24 საათს, თუ საჭიროა მონაცემთა აღდგენა სარეზერვო კოპირებით.
- **დაუყოვნებლივ აღდგენა (Immediate recovery)** - აღდგენის ხერხი, ცნობილი სახელით „ცხელი რეზერვირება“. გაითვალისწინება სერვისის აღდგენა სერვისის შეწყვეტის გარეშე. დაუყოვნებლივი აღდგენა, ჩეულებისამებრ, იყენებს ტექნოლოგიებს: სარკირება (რეზერვირება), დატვირთვის ბალანსირება და დანადგარების დაყენების ფართობის დაყოფა. ეს ხერხი ყველაზე ხშირად ითვალისწინებს სისტემის კომპონენტების „ორმაგ ლოკაციას“, ანუ სრულ დუბლირებას. ის ყველაზე ძვირადღირებულია და გამოიყენება მხოლოდ კრიტიკული ბიზნეს–პროცესებისთვის, რომელთა მოცდენამ შეიძლება დიდი ზარალი გამოიწვიოს. დუბლები უნდა ინახებოდეს რაც შეიძლება მოცილებით ორიგინალებისაგან, რათა ისიც არ დააზიანოს დამანგრეველმა მოვლენამ.

უწყვეტობის უზრუნველყოფის სტრატეგია უნდა მოიცავდეს აღდგენის ყველა ზემოგანხილულ ხერხს. განსხვავებული სერვისები, ორგანიზაციის მიერ გამოყენებული, ითხოვს აღდგენის გასხვავებულ მიდგომებს და მტყუნებათა რისკების შემცირებას. რომელი ოფციაც არ უნდა იყოს არჩეული, ის უნდა იყოს ეკონომიკურად ეფექტური.

მთავარი წესი – რაც უფრო დიდხანს გაძლებს ბიზნესი სერვისის გარეშე, მით უფრო იაფი უნდა იყოს გადაწყვეტა მისი უწყვეტობის უზრუნველსაყოფად.

➢ **სტადია 3** – **რეალიზაცია**

მას შემდეგ, რაც უწყვეტობის უზრუნველყოფის სტრატეგია განსაზღვრულია, აუცილებელია შემუშავდეს სერვისების უწყვეტობის უზრუნველყოფის გეგმები ბიზნესის უწყვეტობის უზრუნველყოფის გეგმების შესაბამისად. ITSCM გეგმები იხილავს ყველა ქმედებას, რომლებიც აუცილებელია საჭირო სერვისების, შესაძლებლობებისა და რესურსების უზრუნველსაყოფად, უწყვეტობის შესაბამისი დონეებით. ეს ნიშნავს არა მხოლოდ საკითხების განხილვას, დაკავშირებულს სერვისების და შესაძლებლობების აღდგენასთან, არამედ ასევე მათ შორის დამოკიდებულებათა ცოდნასაც, ტესტირებას, მთლიანობისა და მონაცემთა თანამიმდევრულობის შემოწმებას,

ITSCM გეგმები უნდა მოიცავდეს დოკუმენტაციას საიმედოობის უზრუნველყოფის საშუალებების აღდგენის ზომების შესახებ, დასაბუთებას კონკრეტული ზომების გამოყენების შესახებ კონკრეტული სიტუაციისაგან დამოკიდებულებით. გეგმების ფორმირების დროს აუცილებელია იმაში დარწმუნება, რომ მათში დეტალურადაა განხილული და დოკუმენტირებული ყველა ქმედება აღდგენისათვის, მტყუნების შემთხვევაში. ITSCM გეგმები უნდა შეიცავდეს ასევე ისეთ ძირითად მომენტებს, როგორიცაა მონაცემთა აღდგენის წერტილი, დამოკიდებული სისტემების სია, ამ დამოკიდებულების ბუნება, მოთხოვნები პროგრამულ და აპარატულ უზრუნველყოფაზე, კონფიგურაციის დეტალები და სხვა მნიშვნელოვანი ინფორმაცია სისტემებისა და სერვისების შესახებ.

ინფორმაციის ერთ-ერთი ყველაზე მნიშვნელოვანი წყაროებიდან გეგმების ფორმირებისათვის არის ბიზნესზე გავლენის ანალიზი. სხვა სფეროებიც უნდა იყოს გაანალიზებული: *SLA*, უსაფრთხოების მოთხოვნები, ექსპლუატაციის ინსტრუქციები, პროცედურები, გარე კონტრაქტები.

გარდა უწყვეტობის უზრუნველყოფის გეგმების დამუშავებისა, იმისათვის, რომ დაცულ იქნას მიღებული უწყვეტობის უზრუნველყოფის სტრატეგია, აუცილებელია შემდეგი ქმედებები:

1. ორგანიზაციული სტრუქტურის დაგეგმვა

კატასტროფის აღმოცენების შემთხვევაში, ორგანიზაციის სტრუქტურა ნაღდი ალბათობით განიცდის ცვლილებას და იქნება დაფუძნებული, უპირველეს ყოვლისა, შემდეგზე:

- *ხელმძღვანელობა* – ტოპმენეჯერი და ორგანიზაციის მმართველობა, რომლებიც ფლობენ ძალაუფლებას და კონტროლის საშუალებებს ორგანიზაციაზე. სწორედ ხელმძღვანელობაა პასუხისმგებელი მართვის შესახებ კრიზისულ სიტუაციაში;
- *კოორდინაცია* – დონე, პასუხისმგებელი აღდგენის პროცესის შიგნით;

○ *აღდგენა* – ბიზნესის და IT ჯგუფების ერთობლიობა, რომლებიც წარმოადგენენ კრიტიკულ ბიზნესფუნქციებს და სერვისებს, მათ მხარდასაჭრად. თითოეული ჯგუფი პასუხისმგებელია თავისი სფეროს აღდგენის გეგმების შესრულებაზე პერსონალთან, მომხმარებლებთან და მესამე მხარესთან ურთიერთმოქმედებით.

2. ტესტირება

აღდგენის გეგმებმა უნდა გაიაროს ტესტირება. ტესტირება მნიშვნელოვანი ნაწილია. სწორედ ის იძლევა გარანტიას, რომ მიღებული სტრატეგია, შეთანხმებები, გეგმები და პროცედურები ნამდვილად იმუშავებს პრაქტიკაში.

სერვისების მიმწოდებელი პასუხისმგებელია იმაზე, რომ კატასტროფის შემთხვევაში სერვისები შეიძლება აღდგენილ იქნას მოცემულ დროით ინტერვალში მოთხოვნილი ფუნქციურ და მწარმოებლურობით. ტესტები უნდა ჩატარდეს მაქსიმალურად რეალისტური სცენარებით. ამასთანავე, საჭიროა გაგება, რომ ყველაზე დეტალური ტესტირებაც კი ვერ გაითვალისწინებს ყველა ნიუანსს, რომლებიც რეალურად. შეიძლება აღმოცენდეს.

➢ **სტადია 4 - უწყვეტი ექსპლუატაცია**

ეს სტადია შედგება შემდეგისგან:

1. სწავლება, მზადყოფნა, ტრენინგი – პერსონალი უნდა იყოს მზად გაუთვალის-წინებელი მდგომარეობების აღმოცენებასთან და იცოდეს, თუ რა უნდა გააკეთოს ამ შემთხვევაში;
2. გადასინჯვა – ITSCM–ის პროცესის ყველა გამოსასვლელი რეგულარულად უნდა გადამოწმდეს აქტუალურობაზე და აუცილებლობის შემთხვევაში, კორექტირდეს;
3. ტესტირება – გარდა საწყისი ტესტირებისა, საჭიროა რეგულარული ტესტირების გათვალისწინება სტრატეგის, გეგმების და ITSCM–ს სხვა გამოსასვლელების. სარეზერვი დუბლები და აღდგენის მექანიზმები აგრეთვე უნდა დატესტირდეს;
4. ცვლილებების მართვა - პროცესი, პასუხისმგებელი ცვლილების შეფასბაზე, მათი გავლენის თვალსაზრისით ITSCM–ს გეგმებზე.

ინიცირება არის დასკვნითი ტესტი ბიზნესისა და სერვისების უწყვეტობის უზრუნველყოფის გეგმებისთვის. ამ პროცესმა უნდა განიხილოს აღდგენის გეგმების ამოქმედების პროცედურა გაუთვალისწინებელი მდგომარეობების შემთხვევაში.

აუცილებელია დახსომება, რომ გადაწყვეტა გეგმების ინიციალიზაციაზე კარგად უნდა იყოს აწონილი, განსაკუთრებით მესამე მხარის აღდგენის სერვისების გამოყენებაზე.

მტყუნება შეიძლება მოხდეს ნებისმიერ მომენტში, ამიტომ უნდა არსებობდეს აღდგენის გეგმების დაუყოვნებლივ ინიცირების შესაძლებლობა.

ITSCM–ის შესასვლელბია:

1. ინფორმაცია ბიზნესიდან – სტრატეგია, გეგმები და ორგანიზაციის ბიუჯეტი, მიმდინარე და სამომავლო მოთხოვნები;
2. ინფორმაცია IT-დან – სტრატეგია, გეგმები და IT–ს ბიუჯეტი;
3. ბიზნესის უწყვეტობის უზრუნველყოფის სტრატეგია და გეგმები;
4. ინფორმაცია სერვისების შესახებ - ინფორმაცია *SLM–დან*, კერძოთ სერვისების პორტფელიდან, სერვისების კატალოგიდან, SLA/*SLR –დან*;
5. ფინანსური ინფორმაცია - ინფორმაცია ფინანსების მართვის პროცესიდან სერვისების, რესურსებისა და კომპონენტების უზრუნველყოფის ღირებულებათა შესახებ;
6. ინფორმაცია ცვლილებების შესახებ - ინფორმაცია ცვლილებების მართვის პროცესიდან, კერძოთ ცვლილებების განრიგი და მათი გავლენა უწყვეტობის უზრუნველყოფის გეგმებზე;
7. ინფორმაცია ბიზნესისა და სერვისების, დამხმარე სერვისებისა და ტექნოლოგიების ურთიერთმიმართების შესახებ;
8. ბიზნესის უწყვეტობის უზრუნველყოფის მართვის და წვდომის მართვის განრიგები;
9. სერვისების უწყვეტობის უზრუნველყოფის გეგმები და პარტნიორთა, მიმწოდებელთა ტესტირების რეპორტები;

ITSCM–ის გამოსასვლელებია:

1. ITSCM–ის პოლიტიკა და სტრატეგია;
2. გეგმების ერთობლიობა, მათ შორის ანტიკრიზისული მართვის, სასწრაფო საპასუხო ქმედებების, აღდგენებისა კატასტროფების შემდეგ. აგრეთვე დამხმარე გეგმების ერთობლიობა და კონტრაქტები სერვისების აღდგენის მიმწოდებლებთან.

ანტიკრიზისული მართვა (Crisis Management) - პროცესი, პასუხისმგებელი ბიზნესის უწყვეტობის მართვის შესახებ ფართო გაგებით. ანტიკრიზისული მართვის გუნდი პასუხს აგებს სტრატეგიულ საკითხებზე, როგორიცაა მართვა ურთიერთ-დამოკიდებულებისა მასობრივ ინფორმაციის საშუალებებთან, აქციონერთა ნდობის, ასევე იღებს გადაწყვეტილებას ბიზნესის უწყვეტობის უზრუნველყოფის გეგმების ინიციალიზაციის შესახებ.

3. ბიზნესზე გავლენის ანალიზი და შესაბამისი რეპორტები;
4. რისკების ანალიზი მმართველობითი მიმოხილვები და რეპორტები;
5. ITSCM–ის ტესტირების განრიგი;
6. სცენარები ტესტირების ჩასატარებლად;
7. მიმოხილვები და რეპორტები ITSCM–ის ტესტირების შესახებ.

საკვანძო მაჩვენებლი ITSCM–ის მწარმოებლურობის არის ის, რომ წარმოდგენილი სერვისები შეიძლება იქნას აღდგენილი ბიზნესის მხარდაჭერის მიზნით დასახული მიზნების მისაღწევად:

1. ტარდება რეგულარული აუდიტი ITSCM-ის გეგმების იმის შემოწმების მიზნით, რომ ბიზნესის მოთხოვნები აღდგენისათვის შეიძლება იყოს დაკმაყოფილებული;

2. სერვისების აღდგენის ყველა მიზნობრივი მაჩვენებელი დოკუმენტირებულია, შეთანხმებულია SLA-ში, და შეიძლება იქნას მიღწეული ITSCM–ის გეგმების დახმარებით;

3. ტარდება რეგულარული და ყოვლისმომცველი ტესტირება ITSCM–ის გეგმების;

4. დადებულია ITSCM-ის ყველა აუცილებელი კონტრაქტი მესამე მხარესთან;

5. უზრუნველიყოფა რისკების შემცირება და სერვისების მტყუნების ნეგატიური გავლენა.

ეფექტურობის მაჩვენებლის სახით შეიძლება ასევე განხილულ იქნას ორგანიზაციის მზადყოფნა ქმედებებისადმი, ITSCM-ის გეგმების შესაბამისად.

ITSCM-ის ძირითადი რისკებია ინფორმაციის უკმარისობა და არაკორექტულობა, მოსული ბიზნესიდან, IT-დან და სხვა პროცესებიდან, აგრეთვე რესურსების დეფიციტი უწყვეტობის უზრუნველსაყოფად.

7.12. ინფორმაციული უსაფრთხოების მართვა

ინფორმაციული უსაფრთხოების მართვა (Information Security Management ან ISM) – არის პროცესი, რომელიც უზრუნველყოფს კონფიდენციალობას, მთლიანობას და წვდომას ორგანიზაციის აქტივების, ინფორმაციის, მონაცემების და სერვისებისას.

ინფორმაციული უსაფრთხოების მართვა არის ორგანიზაციული მიდგომის ნაწილი ინფორმაციული უსაფრთხოებისადმი, რომელსაც აქვს უფრო ფართო არეალი, ვიდრე სერვისების მიმწოდებელს, და მოიცავს საქაღალდო დოკუმენტების დამუშავებას, შენობაში წვდომას, სატელეფონო ზარებს და ა.შ. მთელი ორგანიზაციისთვის [11].

ISM –ის ძირითადი მიზანია ინფორმაციული უსაფრთხოების მართვის ეფექტური უზრუნველყოფა ყველა სერვისის და ქმედების სერვისების მართვის ფარგლებში. ინფორმაციული უსაფრთხოება დანიშნულია ინფორმაციის კონფიდენციალურობის, წვდომისა და მთლიანობის დარღვევის, ასევე ინფორმაციული სისტემების და კომუნიკაციების დაცვისათვის.

1. **კონფიდენციალურობა** – ინფორმაციის მდგომარეობა, რომლის დროსაც მასზე წვდომას ახორციელებს მხოლოდ ამის უფლების მქონე სუბიექტები;–

2. **მთლიანობა** – ინფორმაციის მდგომარეობა, რომლის დროსაც გამორიცხულია მისი ნებისმიერი ცვლილება, ან ცვლილებას ახორციელებს მხოლოდ ამის უფლების მქონე სუბიექტები;

3. **წვდომა** – ინფორმაციის მდგომარეობა, რომლის დროსაც წვდომის უფლების მქონე სუბიექტებს შეუძლიათ ამის რეალიზაცია შეუფერხებლად.

ინფორმაციული უსაფრთხოების მართვის მიზანი მიღწეულია, თუ:

1. ინფორმაცია მიღწევადია მაშინ, როცა ეს საჭიროა, ხოლო საინფორმაციო სისტემები მდგრადია შეტევებისაგან, შეუძლიათ მათი თავიდან აცილება ან სწრაფი აღდგენა;
2. ინფორმაცია მისაწვდომია მხოლოდ მათთვის, ვისაც ამის უფლება აქვს;
3. ინფორმაცია კორექტულია, სრული და დაცულია არაავტორიზებული ცვლილებებისაგან;
4. ინფორმაციის გაცვლა პარტნიორებთან და სხვა ორგანიზაციებთან საიმედოდ დაცულია.

ბიზნესი განსაზღვრავს, რა და როგორ უნდა იქნეს დაცული. ამ დროს ინფორმაციული უსაფრთხოების უზრუნველყოფის ეფექტურობისა და მთლიანობისთვის აუცილებელია ბიზნეს პროცესების განხილვა თავიდან ბოლომდე, რადგან სუსტმა ადგილმა შეიძლება მთელი სისტემა დააზარალოს.

ISM პროცესი უნდა მოიცავდეს:

- ინფორმაციული უსაფრთხოების პოლიტიკის ფორმირება, მართვა, გავრცელება და დაცვა, ასევე სხვა დამხმარე პოლიტიკებისა, რომლებიც კავშირშია ინფორმაციულ უსაფრთხოებასთან.

ინფორმაციული უსაფრთხოების პოლიტიკა (Security Policy) – ესაა პოლიტიკა, რომელიც განსაზღვრავს ორგანიზაციის მიდგომას ინფორმაციული უსაფრთხოების მართვასთან;

- უსაფრთხოებისადმი ბიზნესის შეთანხმებული მიმდინარე და სამომავლო მოთხოვნების გაგება;
- უსაფრთხოების კონტროლის გამოყენება ინფორმაციული უსაფრთხოებისა და რისკების მართვის პოლიტიკის შესასრულებლად, რომლებიც დაკავშირებულია ინფორმაციის, სისტემების და სერვისების წვდომასთან. ტერმინი „უსაფრთხოების კონტროლი“ ნასესხებია ინგლისურიდან და მოცემულ კონტექსტში ნიშნავს კონტრზომებისა და გამაფრთხილებელი ზომების ერთობლიობას, რომლებიც გამოიყენება რისკების შემცირების, ანულირების და მათთადმი წინააღმდეგობისათვის. ანუ უსაფრთხოების კონტროლი შედგება პროაქტიური და რეაქტიური ქმედებებისაგან;
- უსაფრთხოების კონტროლის სიის დოკუმენტირება, ქმედებები მათი ექსპლუატაციის და მართვისთვის, ასევე მასთან დაკავშირებულ რისკებთან;
- მიმწოდებლებისა და კონტრაქტების მართვა, რომლებიც მოითხოვს წვდომას სისტემებსა და სერვისებზე. ხორციელდება მიმწოდებლების მართვის პროცესთან ურთიერთქმედებით;
- უსაფრთხოების და ინციდენტების ყველა ხვრელის კონტროლი, რომლებიც კავშირშია სისტემებთან და სერვისებთან;
- უსაფრთხოების კონტროლის პროაქტიური სრულყოფა და ინფორმაციული უსაფრთხოების დარღვევის რისკების შემცირება;

- ინფორმაციული უსაფრთხოების ასპექტების ინტეგრაცია სერვისების მართვის ყველა პროცესში.

ინფორმაციული უსაფრთხოების პოლიტიკა უნდა მოიცავდეს შემდეგს:

- ინფორმაციული უსაფრთხოების პოლიტიკის ასპექტების რეალიზაცია;
- ინფორმაციული უსაფრთხოების პოლიტიკის ასპექტების შესაძლო ბოროტად გამოყენება;
- წვდომის კონტროლის პოლიტიკა;
- პაროლების გამოყენების პოლიტიკა;
- ელექტრონული ფოსტის პოლიტიკა;
- ინტერნეტის პოლიტიკა;
- აქტიური დაცვის პოლიტიკა;
- ინფორმაციის კლასიფიკაციის პოლიტიკა;
- დოკუმენტების კლასიფიკაციის პოლიტიკა;
- დაშორებული წვდომის პოლიტიკა;
- მიმწოდებელთა წვდომის პოლიტიკა სერვისებთან, ინფორმაციასთან და კომპონენტებთან;
- აქტივების განლაგების პოლიტიკა.

ჩამოთვლილი პოლიტიკები მისაწვდომი უნდა იყოს მომხმარებლებისა და დამკვეთებისათვის, რომლებიც, თავის მხრივ ვალდებულნი არიან თანხმობა დაამოწმონ წერილობით.

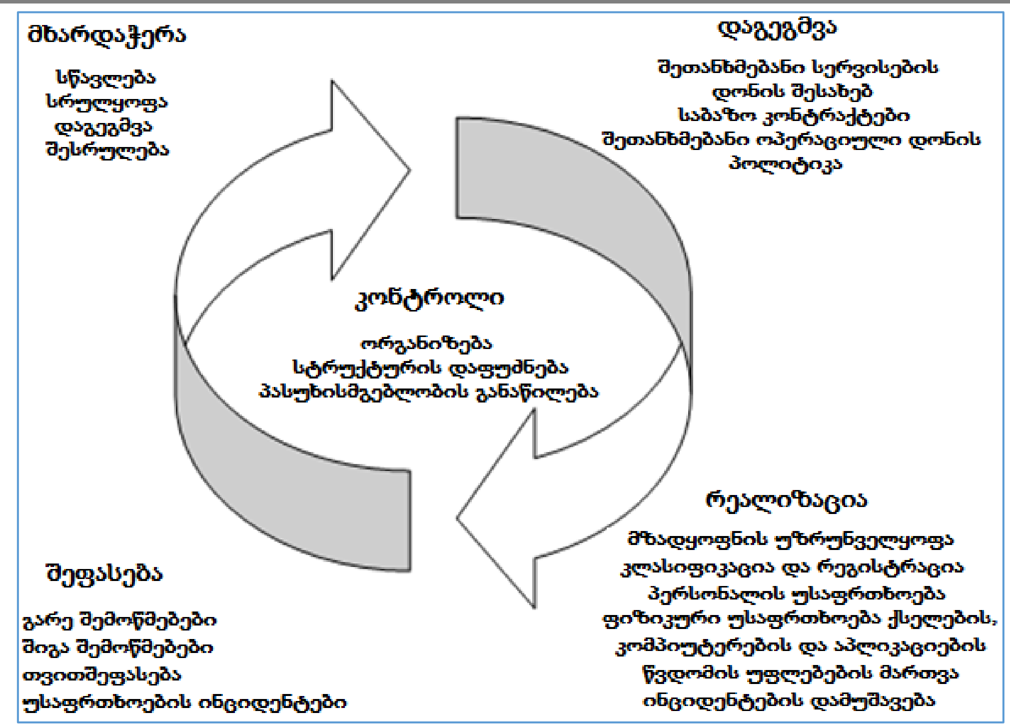
პოლიტიკები მტკიცდება ბიზნესის და IT-ის ხელმძღვანელობის მიერ გადაიხედება მდგომარეობათა მიხედვით. რათა განხორციელდეს ინფორმაციული უსაფრთხოება და მისი მართვა, აუცილებელია ინფორმაციული უსაფრთხოების მართვის სისტემის მხარდაჭერა. **ინფორმაციული უსაფრთხოების მართვის სისტემა (Information Security Management System ან ISMS)** - ესაა სისტემა პოლიტიკების, პროცესების, სტანდარტების, სახელმძღვანელო დოკუმენტებისა და საშუალებების, რომლებიც უზრუნველყოფენ ორგანიზაციებს ინფორმაციული უსაფრთხოების მართვის მიზნების მისაღწევად.

7.24 ნახაზზე ნაჩენებია ISMS-ის სტრუქტურა, რომლებიც უფრო ფართოდ გამოიყენება ორგანიზაციების მიერ.

ნახაზზე განიხილება ISMS-ის სტრუქტურის ხუთი ელემენტი:

1. **კონტროლი**. მიზანია:

   - ინფორმაციული უსაფრთხოების მართვის სისტემის ფორმირება ორგანიზაციის ფარგლებში;
   - ორგანიზაციული სტრუქტურის ფორმირება ინფორმაციული უსაფრთხოების პოლიტიკის რეალიზაციის მომზადების, დამტკიცებისა და რეალიზაციისათვის;
   - პასუხიმგებლობათა განაწილება;
   - კონტროლის დოკუმენტაციის ფორმირება.



ნახ.7.24. **ISMS**

2. დაგეგმვა. მიზანია – ინფორმაციული უსაფრთხოების შესაფერისი მეტრიკებისა და გაზომვის ხერხების დამუშავება და რეკომენდაცია, პირველ რიგში, დაგეგმვა უნდა ითვალისწინებდეს კონკრეტული ორგანიზაციის მოთხოვნებს და თავისებურებებს. ინფორმაციის წყაროებად ინფორმაციული უსაფრთხოების მოთხოვნების დასადგენად განიხილავენ ბიზნესს, რისკებს, გეგმებს, სტრატეგიას, შეთანხმებებს (პირველ რიგში, OLA - operational-level agreement და SLA). ამ დროს მნიშვნელოვანია გათვალისწინებულ იქნას მორალური, სამართლებრივი და ეთიკის.

3. რეალიზაცია. მიზანია - შესაბამისი პროცედურების, ინსტრუმენტებისა და უსაფრთხოების კონტროლების უზრუნველყოფა ინფორმაციული უსაფრთხოების პოლიტიკის მხარდასაჭერად.

რეალიზაციის ფარგლებში ხორციელდება შემდეგი ღონისძიებები:

- *აქტივების იდენტიფიკაცია* - კონფიგურაციების მართვასთან ერთად;
- ინფორმაციის კლასიფიკაცია - ინფორმაცია და ინფორმაციული საცავები უნდა იყოს კლასიფიცირებული შესაბამისად მათი მგრძნობიარობისა და მნიშვნელობისა ინფორმაციული უსაფრთხოების სამ ასპექტთან მიმართებით (კონფიდენციალურობა, მთლიანობა, წვდომა).

4. შეფასება. მიზანი ISMS–ის ფარგლებში:

○ ინფორმაციული უსაფრთხოების პოლიტიკის შესაბამისობის შემოწმება ინფორმაციული უსაფრთხოების მოთხოვნებთან SLA და OLA –დან;

○ ინფორმაციული უსაფრთხოების ტექნიკური მდგენელის რეგულარული შემოწმებების ჩატარება IT–სისტემისათვის;

○ ინფორმაციის მიწოდება რეგულატორებზე და გარე აუდიტებზე აუცილებლობის შემთხვევაში;

5. **მხარდაჭერა.** მიზნები ISMS–ის მხარდასაჭერად:

○ შეთანხმებათა სრულყოფა ინფორმაციულ უსაფრთხოებასთან მიმართებით, მაგალითად, SLA და OLA;

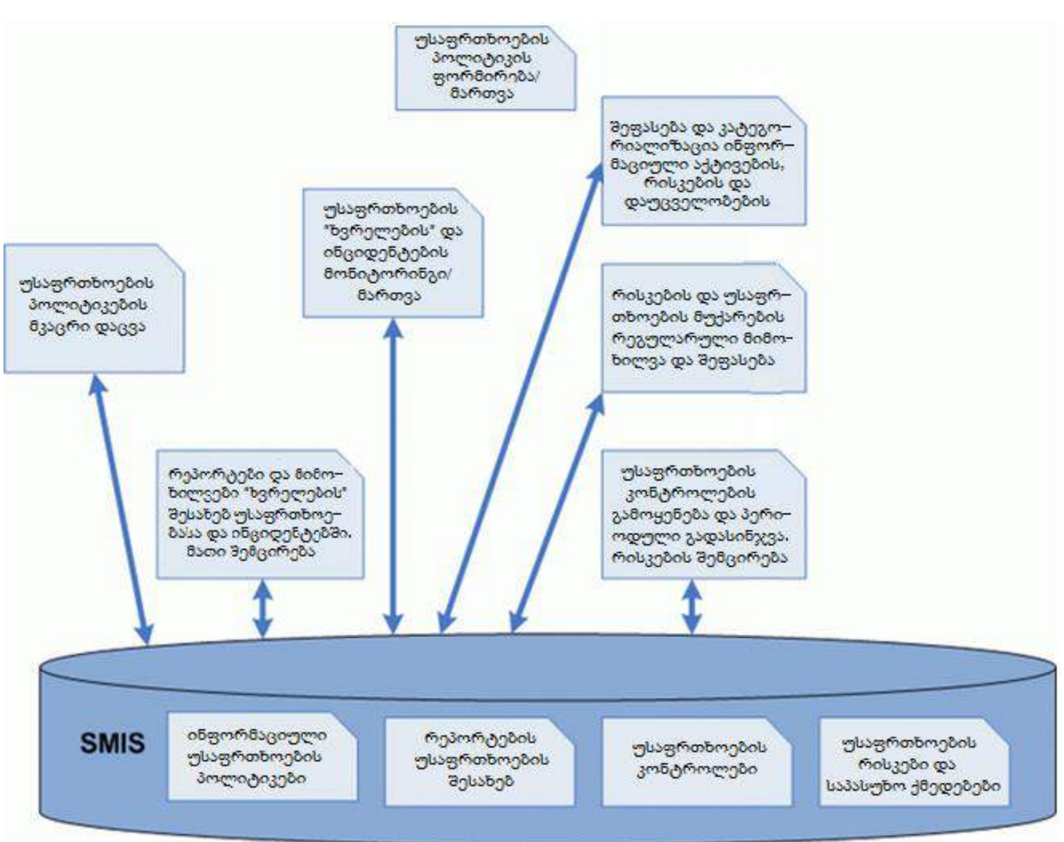
○ ინფორმაციული უსაფრთხოების საშუალებებისა და კონტროლების სრულყოფა.

საკვანძო ქმედებები *ISM –ის* ფარგლებში:

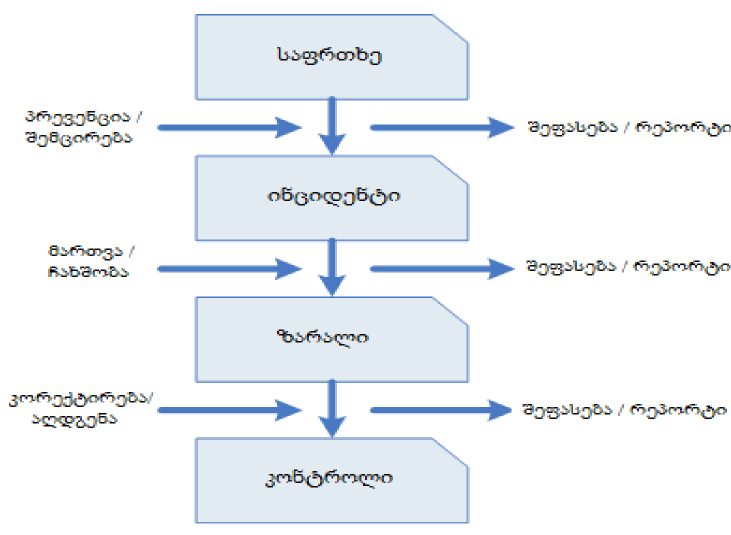
1. ინფორმაციული უსაფრთხოების პოლიტიკსა და მისი მხარდამჭერი დამხმარე პოლიტიკების ერთობლიობის ფორმირება, გადასინჯვა და კორექტირება;
2. ინფორმაციული უსაფრთხოების პოლიტიკის რეალიზაცია და დაცვა, ასევე მათ შორის ურთიერთქმედების უზრუნველყოფა;
3. ინფორმაციული აქტივებისა და დოკუმენტების შეფასება და კლასიფიკაცია;
4. უსაფრთხოების კონტროლების ერთობლიობის, რისკების შეფასების ზომების, საპასუხო ქმედებების;
5. უსაფრთხოების „ხვრელების“ და ინციდენტების მონიტორინგი და მართვა;
6. ანალიზი, რეპორტების წარმოება და უსაფრთხოებაზე „ხვრელების“ გავლენისა და ინციდენტების შემცირება;
7. განრიგის შედგენა და აუდიტების, ტესტირებისა და მიმოხილვების ჩატარება.

აღწერილ ქმედებათა ურთიერთმოქმედება მოცემულია 7.25 ნახაზზე.

ინფორმაციული უსაფრთხოების პოლიტიკის უზრუნველსაყოფად და მხარდასჭერად აუცილებელია უსაფრთხოების კონტროლების ერთობლიობის ფორმირება და გამოყენება. ინციდენტების თავიდან ასაცილებლად და სწორი რეაგირებისათვის მათი აღმოცენების შემთხვევაში იყენებენ უსაფრთხოების ზომებს რომლებიც 7.26 ნახაზზეა ასახული.



ნახ.7.25. საკვანძო ქმედებები ISM–ის ფარგლებში



ნახ.7.26. უსაფრთხოების კონტროლები

ნახაზზე გამოყოფილია ოთხი სტადია.

პირველი – საფრთხის წარმოქმნა. **საფრთხე** არის ყველაფერი, რაც უარყოფითად აისახება ბიზნესპროცესზე ან შეუძლია მისი შეწყვეტა. **ინციდენტი** არის რეალიზებული საფრთხე.

ინციდენტი ამოსავალი წერტილია უსაფრთხოების კონტროლების გამოსაყენებლად. ინციდენტის შედეგად ჩნდება ზარალი. რისკების მართვის ან აღმოფხვრისათვის ასევე იყენებენ უსაფრთხოების კონტროლებს. ყოველი სტადიისათვის საჭიროა ინფორმაციული უსაფრთხოების შესაფერისი ზომების შერჩევა:

1. **პრევენციული** – უსაფრთხოების ზომები, რომლებიც გამორიცხავს წინასწარ ინფორმაციული უსაფრთხოების ინციდენტის გამოვლენას. მაგალითად, წვდომის ნებართვების განაწილება;

2. **აღდგენა** – უსაფრთხოების ზომები, მიმართული პოტენციალური ზარალის შესამცირებლად ინციდენტის შემთხვევაში. მაგალითად, სარეზერვო დუბლირება;

3. **აღმომჩენი** – უსაფრთხოების ზომები, მიმართული ინციდენტების აღმოსაჩენად. მაგალითად, ანტივირუსული დაცვა ან შემოჭრის აღმოჩენის სისტემა;

4. **ჩამხშობი (აღმკვეთი)** – უსაფრთხოების ზომები, რომლებიც ეწინააღმდეგება საფრთხის რეალიზაციის მცდელობას, ანუ ინციდენტებს. მაგალითად, ბანკომატი ართმევს კლიენტს ელ–კარტას მის მიერ რამდენჯერმე PIN-კოდის არასწორად შეტანის შემთხვევაში;

5. **მაკორექტირებელი** – უსაფრთხოების ზომები, მიმართული აღდგენისათვის ინციდენტის შემდეგ. მაგალითად, სარეზერვო დუბლების აღდგენა, წინა სამუშაო მდგომარეობაში დაბრუნება და ა.შ.

ISM პროცესის შესასვლელებია:

1. ინფორმაცია ბიზნესიდან – სტრატეგიები, გეგმები, ბიზნესის ბიუჯეტი, ასევე მისი მიმდინარე და სამომავლო მოთხოვნები;

2. ბიზნესის უსაფრთხოების პოლიტიკები, უსაფრთხოების გეგმები, რისკების ანალიზი;

3. ინფორმაცია IT-დან – სტრატეგია, გეგმები და ბიუჯეტი IT–ის;

4. ინფორმაცია სერვისების შესახებ - *SLM–დან*, კერძოდ, სერვისების პორტფელიდან და კატალოგიდან, SLA/*SLR*;

5. რეპორტები: პროცესებისა და რისკების ანალიზის *ISM–დან*, სერვისების წვდომის მართვისა და უწყვეტობის მართვისა;

6. დეტალური ინფორმაცია ინფორმაციული უსაფრთხოების ყველა ინციდენტზე და „ხვრელებზე" .

7. ინფორმაცია ცვლილებების შესახებ – ინფორმაცია ცვლილებების მართვის პროცესიდან, კერძოდ, ცვლილებათა განრიგი და მათი გავლენა გეგმებზე, პოლიტიკებზე და ინფორმაციული უსაფრთხოების კონტროლებზე;

8. ინფორმაცია ბიზნესის ურთიერთმიმართებაზე სერვისებთან, დამხმარე სერვისებთან და ტექნოლოგიებთან;

9. ინფორმაცია სერვისებთან და სისტემებთან პარტნიორების და მიმწოდებლების წვდომის შესახებ, წარმოდგენილი მიმწოდებელთა მართვის და წვდომის მართვისა პროცესების მიერ.

ISM –ის გამოსასვლელებია:

1. ყოვლისმომცველი ინფორმაციული უსაფრთხოების პოლიტიკა და სხვა დამხმარე პოლიტიკები, რომელთაც აქვს კავშირი ინფორმაციულ უსაფრთხოებასთან;

2. ინფორმაციული უსაფრთხოების მართვის სისტემა (ISMS), რომელიც შეიცავს მთელ ინფორმაციას, აუცილებელს *ISM–ის* უზრუნველსაყოფად;

3. რისკების გადაფასების შედეგები და რევიზიის რეპორტები;

4. უსაფრთხოების კონტროლების ერთობლიობა, მათი ექსპლუატაციისა და მართვის აღწერა, ასევე მათთან დაკავშირებული ყველა რისკისა;

5. ინფორმაციული უსაფრთხოების აუდიტი და რეპორტები;

6. ინფორმაციული უსაფრთხოების გეგმების ტესტირების განრიგი;

7. ინფორმაციული აქტივების კლასიფიკაცია;

8. რეპორტები ინფორმაციულ უსაფრთხოებასა და ინციდენტებში არსებული „ხვრელების" შესახებ;

9. პოლიტიკები, პროცესები და პროცედურები მიმწოდებელთა და პარტნიორთა წვდომის მართვისათვის სერვისებთან და სისტემებთან.

ინფორმაციულ უსაფრთხოების მწარმოებლურობის პროცესის საკვანძო მაჩვენებლებად შეიძლება გამოყენებულ იქნას მეტრიკების სიმრავლე, მაგალითად:

1. ბიზნესის დაცულობა ინფორმაციულ უსაფრთხოების დარღვევისგან.

- შეტყობინებათა %-ული შემცირება „ხვრელების" შესახებ სერვის–დესკზე;
- ბიზნესზე ნეგატიური გავლენის პროცენტული შემცირება „ხვრელების" და ინციდენტების მხრიდან;
- პროცენტული გაზრდა პუნქტების, რომელთაც ეხება ინფორმაციული უსაფრთხოება, SLA–ში.

2. ინფორმაციული უსაფრთხოების ცხადი და შეთანხმებული პოლიტიკის ფორმირება, ბიზნესის მოთხოვნილებათა გათვალისწინებით, ანუ არადამთხვევათა რაოდენობის შემცირება *ISM–ის* პროცესებსა და ბიზნესის ინფორმაციული უსაფრთხოების პროცესებსა და პოლიტიკებს შორის;

3. უსაფრთხოების უზრუნველყოფის პროცედურები, რომლებიც გამართლებული, შეთანხმებული და დამტკიცებულია ორგანიზაციის ხელმძღვანელობის მიერ:

- უსაფრთხოების უზრუნველყოფის პროცედურების შეთანხმებულობისა და სარგებლიანობის გაზრდა;

○ მხარდაჭერის გაზრდა ხელმძღვანელობის მხრიდან;

4. სრულყოფის მექანიზმები:

○ შეთავაზებულ სრულყოფათა რაოდენობა კონტროლების და პროცედურების მიმართებით;

○ არადამთხვევათა რაოდენობის შემცირება, ტესტირებისა და აუდიტის დროს აღმოჩენის პროცესში.

5. ინფორმაციული უსაფრთხოება არის განუყოფელი ნაწილი *ITSM–ის* სერვისებისა და პროცესების, ანუ შესაძლებელია სერვისებისა და პროცესების რაოდენობის ზრდა, რომლებშიც გათვალისწინებულია უსაფრთხოების ზომები.

ISM ეჯახება სიძნელეებსა და რისკებს ინფორმაციული უსაფრთხოების უზრუნველყოფის გზაზე. სამწუხაროდ, პრაქტიკაში ძალზე ხშირად ბიზნესი თვლის, რომ ინფორმაციული უსაფრთხოების საკითხებზე უნდა იმუშაოს მხოლოდ IT–მ. კიდევ უარესი, როცა ბიზნესს არ ესმის, თუ საერთოდ რისთვისაა საჭირო ყურადღების მიქცევა ინფორმაციულ უსაფრთხოებაზე. ინფორმაციის დაცვის ეფექტური სისტემის შექმნა მოიცავს დიდ ხარჯებს, რომლებიც უნდა ესმოდეს ხელმძღვანელობას, რადგან ისინი წყვეტენ ფინანსირების საკითხებს. ამ დროს მნიშვნელოვანია ბალანსის დაცვა – ინფორმაციული უსაფრთხოების უზრუნველყოფა არ უნდა ღირდეს თვით დასაცავ ინფორმაციაზე ძვირი.

7.13. სერვისების დანერგვა

ბიზნესის მართვის პროცესის სრულყოფა ხორციელდება პროექტების განხორციელების საშუალებით, რომლებშიც მონაწილეობს IT-დეპარტამენტიც. ნებისმიერი უმნიშვნელო ოპერაციული სრულყოფა ან გლობალური მოვლენა, რომელიც გარდაქმნის მთლიან ბიზნესს, საბოლოოოდ ჯამში იწვევს ცვლილებას. კერძოდ, ახალი ან შეცვლილი სერვისის გამოყენებაც არის ცვლილება ბიზნესისათვის.

დანერგვის ეტაპი იძლევა გარანტიას, რომ სასიცოცხლო ციკლის წინა სტადიებზე დაგეგმილი და დაპროექტებული სერვისები შეძლებს ბიზნესისა და IT-ისთვის მოსალოდნელი შედეგების პრაქტიკულ მიღებას. ამგვარად, დანერგვა არის სერვისის ერთგვარი შემოწმების პროცესი, მისი უშუალოდ ექსპლუატაციაში გადაცემის წინ.

განვიხილოთ დანერგვასთან დაკავშირებული ტერმინები:

- **გარდაქმნა** (Transition) – მდგომარეობის ცვლილება, რომელიც შეესაბამება სერვისის ან კონფიგურაციული ერთეულის გადაადგილებას სასიცოცხლო ციკლის ერთი სტადიიდან მეორეზე.
- **რელიზი** (Release) – ერთობლიობა: აპარატული და პროგრამული უზრუნ-ველყოფების, დოკუმენტაციის, პროცესების ან სხვა კომპონენტებისა, რომლებიც აუცილებელია სერვისში ერთი ან რამდენიმე შეთანხმებული ცვლილების დასანერგად.

ყოველი რელიზის შედგენილობა იმართება, ტესტირდება, განთავსდება როგორც ცალკე არსი (ობიექტი).

- **მოთხოვნა ცვლილებაზე** (Request for Change ან RFC) - ფორმალური წინადადება ცვლილების სარეალიზაციოდ. RFC შეიცავს შემოთავაზებული ცვლილების დეტალურ აღწერას და შეიძლება ჩაიწეროს ქაღალდზე ან ელექტრონულ ფორმატში.
- **ტესტირება** (Test) – ქმედება, რომელიც ამოწმებს (ადასტურებს), რომ სერვისი, პროცესი ან კონფიგურაციული ერთეული შეესაბამება სპეციფიკაციებს ან შეთანხმებულ მოთხოვნებს.
- **აწყობა** (Build) – ქმედება ერთი ან რამდენიმე კონფიგურაციული ერთეულის ასაწყობად სერვისის ნაწილის ფორმირებისათვის. მაგალითად, სერვერის ან ნოუთბუკის აწყობა.
- **განთავსება** (Deployment) – ქმედება, რომელიც პასუხისმგებელია ახალი ან შეცვლილი დანადგარის, პროგრამის, დოკუმენტაციის, პროცესის გადაადგილებაზე სამრეწველო ექსპლუატაციის გარემოში.
- **მხარდაჭერა ექსპლუატაციის დასაწყისში** (Early Life Support) - მხარდაჭერა, რომელიც სჭირდება ახალ ან შეცვლილ სერვისს სათანადო პერიოდის განმავლობაში, მისი ექსპლუატაციაში შესვლის შემდეგ. მხარდაჭერის პერიოდში სერვისის მიმწოდებელს შეუძლია გადახედოს KPI-ს, სერვისის დონეებს და საკონტროლო ზღვრულ მნიშვნელობებს. აგრეთვე შეუძლია დამატებითი რესურსების ამოქმედება ინციდენტებისა და პრობლემების სამართავად.
- **გარემო** (Environment) – IT-ინფრასტრუქტურის ქვესიმრავლე, რომელიც გამოიყენება სხვადასხვა მიზნებისთვის. რთული გარემოსთვის არის შესაძლებლობა კონფიგურაციული ერთეულების ერთობლივად გამოსაყენებლად. მაგალითად, ტესტირების გარემოს და სამრეწველო ექსპლუატაციის გარემოს შეუძლია გამოიყენოს სხვადასხვა განყოფილებები ერთ მაინფრეიმზე.
- **სამრეწველო ექსპლუატაციის გარემო** (Live Environment) – მართვადი გარემო, შეიცავს კონფიგურაციულ ერთეულებს სამრეწველო ექსპლუატაციის რეჟიმში, რომელიც გამოიყენება სერვისის მისაწოდებლად.
- **ტესტირების გარემო** (Test Environment) – საკონტროლებელი გარემო, რომელიც გამოიყენება სერვისების, კონფიგურაციული ერთეულების, პროცესების, ანაწყობების ტესტირებისათვის.
- **ანაწყობის გარემო** (Build Environment) – საკონტროლებელი გარემო, რომელშიც თავს იყრის აპლიკაციები, სერვისები და სხვა ანაწყობები მანამ, სანამ ისინი იქნება გადაცემული ტესტირების ან სამრეწველო ექსპლუატაციის გარემოში.
- **მიღება** (Acceptance) – ფორმალური შეთანხმება, რომელიც განსაზღვრავს, რომ სერვისი, პროცესი, გეგმა ან სხვა შედეგი დასრულებულია, არის სწორი, საიმედო და

პასუხობს დადგენილ მოთხოვნებს. მიღებას წინ უსწრებს შეფასება ან ტესტირება. მიღება ხშირად სავალდებულოა პროექტის ან პროცესის მომდევნო ეტაპზე გადასასვლელად.

➢ **დანერგვის ეტაპის ძირითადი მიზნებია:**

- დაგეგმვა / მართვა: სიმძლავრეებისა და რესურსების, რათა განხორციელდეს სერვისების დაკომპლექტება, აწყობა, ტესტირება და სამრწეველო ექსპლუატაციაში გაშვება, აგრეთვე სერვისების ფუნქციონირების უზრუნველყოფა ინვესტორების, და დამკვეთების მოთხოვნის შესაბამისად.
- სერვისების სიმძლავრის ზუსტი და თანამიმდევრული შეფასების სისტემის აგება და რისკების სიის ფორმირება მანამ, სანამ ახალი ან შეცვლილი სერვისი იქნება გაშვებული სამრეწველო ექსპლუატაციაში.
- დანერგვის ეტაპზე გამოსაყენებელი სერვისის აქტივების ნაკრების და კონფიგურაციების ფორმირება და მათი მართვა.
- ინფორმაციის მიწოდება, რომელიც აუცილებელია გადაწყვეტილების მისაღებად სერვისის შეტანის შესახებ საწარმოო ექსპლუატაციაში ტესტირების შემდეგ.
- ეფექტური და განმეორებადი მექანიზმების წარმოდგენა ნაკრებისა და ინსტალირებისათვის, რომლებიც შეიძლება გამოყენებულ იქნას რელიზების განთავსებისთვის საწარმოო ექსპლუატაციის და ტესტირების გარემოში.
- სერვისების მართვის, მხარდაჭერისა და კორექტული ექსპლუატაციის უზრუნველყოფა დაპროექტების ეტაპზე განსაზღვრული მოთხოვნების შესაბამისად.

➢ **დანერგვის ეტაპის ამოცანებია:**

- დამკვეთთა მოლოდინის განსაზღვრა, თუ როგორ დაეხმარება ბიზნესს ახალი ან შეცვლილი სერვისი;
- ახალი ან შეცვლილი სერვისის ინტეგრაციის მიზნით დახმარება დამკვეთთა ბიზნესპროცესში;
- განსხვავების შემცირება პროგნოზირებულ და რეალურ მწარმოებლურობას შორის;
- ცნობილი შეცდომებისა და რისკების რაოდენობის შემცირება ახალი ან შეცვლილი სერვისის გაშვებისას საწარმოო ექსპლუატაციაში;
- სერვისის გამოყენების უზრუნველყოფა მისთვის დადგენილი მოთხოვნებისა და შეზღუდვების გათვალისწინებით.

➢ **ბიზნესისათვის დანერგვის ეტაპს აქვს შემდეგი ღირებულება:**

- აუმჯობესებს ბაზარზე ადაპტირების უნარს ახალი მოთხოვნების ან გარემოებებისადმი;
- აუმჯობესებს მართვას დანერგვის დონეზე კომპანიების შთანთქმის, დაყოფის ფარგლებში, სერვისების შესყიდვის ან გადაადგილებისას;

- ამაღლებს ბიზნესისათვის წარმატებული ცვლილებების და რელიზების რაოდენობას;
- აუმჯობესებს პროგნოზირების სიზუსტეს ახალი ან შეცვლილი სერვისის დონის და ხარისხის შესაბამისად;
- აუმჯობესებს შეთანხმებულობას ბიზნესისა და ხელმძღვანელობის მოთხოვ-ნებთან;
- ამცირებს განსხვავების რაოდენობას დამტკიცებულ ბიუჯეტის გეგმასა და რეალობას შორის;
- ამაღლებს პერსონალის პროდუქტიულობას დაგეგმვის სრულყოფის, ახალი ან შეცვლილი სერვისების გამოყენების შედეგად;
- ამცირებს კონტრაქტების დროებითი შეჩერების ან ცვლილებების შემთხვევებს პროგრამულ და აპარატურულ უზრუნველყოფაზე, კომპონენტების გაერთიანების ან დაყოფის შედეგად;
- აუმჯობესებს რისკის დონის გაგებას ცვლილების დროს და მის შემდეგ.

დანერგვის ეტაპი იმყოფება დაპროექტებისა და ექსპლუატაციის ეტაპებს შორის სასიცოცხლო ციკლში. სწორედ ამ ეტაპებთანაა მისი უფრო მჭიდრო და უწყვეტი კავშირები. დანერგვის ეტაპს კავშირები აქვს ასევე ციკლის სხვა ეტაპებთანაც.

VIII თავი
COBIT სტანდარტები

8.1. COBIT – ტექნოლოგია და ძირითადი ტერმინები

ინფორმაციული ტექნოლოგიების მენეჯმენტის საკითხებზე დღეისათვის არსებობს სტანდარტებისა და მეთოდოლოგიების სათანადო სიმრავლე. მისი ერთ-ერთი წარმომადგენელია COBIT (Control Objectives for Information and Related Technology საკონტროლო ობიექტები საინფორმაციო და მასთანდაკავშირებული ტექნილოგიებისთვის), რომელიც შეიქმნა ISACA (Information Systems Audit and Control Association – საინფორმაციო სისტემების აუდიტის და კონტროლის ასოციაცია) ორგანიზაციის მიერ ამერიკის შეერთებულ შტატებში 1969 წელს, საფინანსო აუდიტებისთვის ინფორმაციული ტექნოლოგიების კონტროლის მიზნით. ამჟამად ამ ორგანიზაციას აქვს მსოფლიოში ერთ-ერთი ლიდერის როლი ინფორმაციული ტექნოლოგიების აუდიტის სტანდარტების შემუშავების სფეროში [44,45].

COBIT არის ღია დოკუმენტების ერთობლიობა, 40-მდე საერთაშორისო სტანდარტი და სახელმძღვანელო IT-მართვის, აუდიტის და ინფორმაციული უსაფრთხოების სფეროებში. ესაა ავტორიტეტული, თანამედროვე, საერთაშორისო აღიარებული მეთოდოლოგიის კვლევა, დამუშავება, პუბლიკაცია კორპორაციული მენეჯმენტისათვის IT-სფეროში. მისი დანიშნულებაა ორგანიზაციებში ამ სტანდარტების დანერგვა და ყოველდღიური გამოყენება IT-სფეროს ბიზნეს-მენეჯერების და აუდიტორთა მიერ [4, 46].

COBIT-ის ძირითადი მიზანია ინფორმაციული ტექნოლოგიების მენეჯმენტი. იმავდროულად, ინფორმაციული ტექნოლოგიების მენეჯმენტი თავის მხრივ არის კორპორაციული მენეჯმენტის განუყოფელი ნაწილი. კორპორაციული მენეჯმენტი – მმართველობითი გადაწყვეტილებისა და მეთოდების კომპლექსია, რომელიც გამოიყენება უმაღლესი ხელმძღვანელობის მიერ შემდეგი მიზნებისთვის:

- სტრატეგიული მიმართულების განსაზღვრისთვის;
- მიზნების მიღწევის უზრუნველსაყოფად;
- რისკების ადეკვატურად სამართავად;
- კორპორაციული რესურსების ეფექტურად გამოსაყენებლად.

კორპორაციული მენეჯმენტი და IT-მენეჯმენტი მოითხოვს მიზნებს შორის ბალანსს, რაც დაკავშირებულია ზემდგომი ხელმძღვანელობის მიერ დადგენილი მოთხოვნილებების შესაბამისობის აუცილებლობასა და ეფექტიანობის ამაღლებასთან.

COBIT-ში გამოიყენება ტერმინი „დაინტერესებული მხარეები“ (Stakeholders), რომლებსაც მიეკუთვნება:

- დირექტორთა საბჭო და უმაღლესი ხელმძღვანელობა: IT-ის განვითარების მიმართულების განსაზღვრა, შედეგების შეფასება და ნაკლოვანებათა აღმოფხვრის მოთხოვნების დადგენა;

- ბიზნეს-განყოფილებების ხელმძღვანელები: ბიზნეს-მოთხოვნების განსაზღვრა IT-ის მიმართ, სარგებლიანობის მიღწევის უზრუნველყოფა IT-დან და რისკების მართვა;
- IT-სამსახურის ხელმძღვანელობა: IT-სერვისებით უზრუნველყოფა და მათი სრულყოფა ბიზნესის მოთხოვნილებათა შესაბამისად;
- შიგა აუდიტი / შიგა კონტროლის სამსახური / IT-აუდიტი: დამოუკიდებელი შეფასების უზრუნველყოფა, რომ IT იძლევა საჭირო სერვისებს;
- რისკების მართვა და შესაბამისობის დაცვა: ნორმატიულ დოკუმენტებთან შესაბამისობის შეფასება რისკების გათვალისწინებით.

8.2. COBIT-ის მიზნები და პრინციპები

COBIT-ის საკვანძო ცნებაა სერვისი ან მომსახურება (service). მაგალითად, ინტერნეტში წვდომის ან დაცულ მონაცემთა საცავთან მიმართვის უზრუნველყოფა მიეკუთვნება მომსახურების სახეებს. ჩვენ სერვისის განსაზღვრა შემოვიტანეთ ITIL მეთოდოლოგიის განხილვისას, რომელიც ასევე სერვისების მენეჯმენტს ეხება. სერვისი არის გარკვეული ფასეულობის მიწოდების ხერხი დამკვეთზე, რომელიც მას ხელს უწყობს სასურველი შედეგების მისაღებად თავისი სისტემის გამოსასვლელზე, ყოველგვარი სპეციფიური ხარჯების და რისკების გარეშე. სერვისების მიწოდება რთული და არატრივიალური ამოცანაა, რომელიც პირველ რიგში მოითხოვს შიგა კონტროლის სისტემას.

COBIT-ის ძირითადი პრინციპებია:

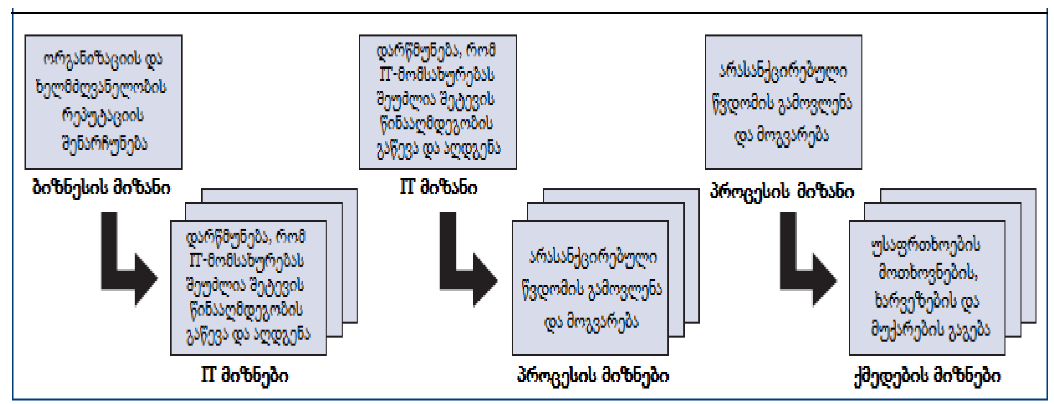
- IT მიზნები უნდა შეესაბამებოდეს ბიზნესის მიზნებს;
- პროცესული მიდგომის გამოყენება;
- IT კონტროლის სისტემა უნდა იყოს შერჩევითი, ანუ განსაზღვროს IT ძირითადი რესურსები და იმუშაოს მასთან;
- კონტროლის მიზნები უნდა იყოს მკაფიოდ განსაზღვრული.

სერვისების მართვის თანამედროვე მიდგომა ყურადღებას ამახვილებს ბიზნესის და IT-ს ურთიერთქმედებაზე.

მიზნები განსაზღვრულია დაღმავალად (top-down ზემოდან-ქვევით) ისე, რომ ბიზნეს-მიზანმა უნდა განსაზღვროს IT-მიზნები თავის მხარდასაჭერად. IT-მიზანი მიიღწევა ერთი პროცესის ან რამდენიმე პროცესის ურთიერთმოქმედებით. ამგვარად, IT-ის მიზანია განსაზღვროს განსხვავებული პროცესების მიზნები.

თავის მხრივ, თითოეული პროცესის მიზანი მოითხოვს აქტიურობათა (ქმედებთა) გარკვეულ რაოდენობას, ასევე მათი მიზნების დადგენას.

8.1 ნახაზზე მოცემულია ბიზნესის, IT-ის, პროცესებისა და ქმედებათა მიზნების დამოკიდებულების მაგალითები.



ნახ.8.1

განვიხილოთ დეტალურად COBIT-ის ძირითადი პრინციპები:

1. **ბიზნესის და IT-ის მიზნები უნდა იყოს ურთიერთდაკავშირებული, მაგრამ განმსაზღვრელი ამ ურთიერთობაში არის ბიზნესის მიზნები.** კომპანიის მოგება პირდაპირ დამოკიდებულებაშია IT-ის ეფექტურ გამოყენებასთან. ამიტომ ხელმძღვანელობამ მეტი ყურადღება უნდა გაამახვილოს მის სარგებლიანობაზე, ინვესტირებაზე, შედეგების მონიტორინგსა და შეფასებაზე;

2. **პროცესული მიდგომის გამოყენება**. პროცესი არის საქმიანობათა სახეების სტრუქტურირებული ერთობლიობა, რომელიც დაპროექტებულია განსაზღვრული მიზნის მისაღწევად. ანუ პროცესი, ზოგადად, პროცედურების ერთობლიობაა, რომელზეც გავლენას ახდენს ორგანიზაციის პოლიტიკა და სხვა წყაროების პროცესები. ბიზნესი განაპირობებს პროცესის წარმოქმნის მიზეზს, მის პასუხისმგებელ მფლობელს, თანამდებობრივ მოვალეობებს, რომლებიც დაკავშირებულია პროცესის შევსების, შესრულების და ეფექტიანობის გაზომვის საშუალებებთან.

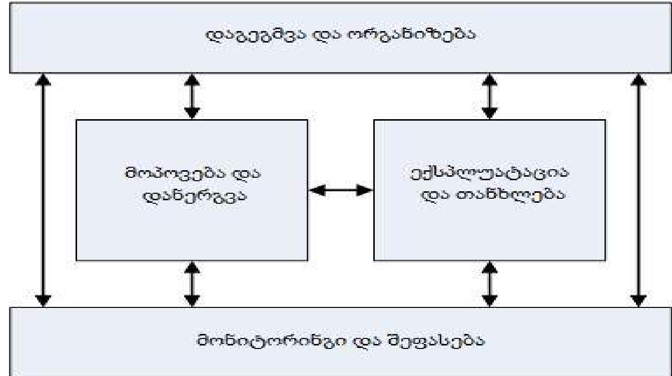
8.3. COBIT-ის პროცესები

პროცესებს აქვს შემდეგი მახასიათებლები:

- **პროცესები გაზომვადია,** ანუ ისინი შეიძლება შეფასდეს რომელიმე შესაბამისი მეთოდით. მაგალითად, მენეჯერები იყენებენ პროცესების ღირებულებას და ხარისხს, მომხმარებლები კი - პროცესების ხანგრძლივობას და პროდუქტიულობას;
- **პროცესები ემსახურება კონკრეტული შედეგების მიღწევას.** პროცესის არსებობის მიზეზი არის კონკრეტული შედეგის მიღება, რომელიც შეიძლება იდენტიფიცირდეს (გამოვლინდეს) და რაოდენობრივად შეფასდეს (დათვლილ იქნეს);
- **პროცესებს ჰყავს მომხმარებლები.** ყოველი პროცესი თავის შედეგებს აწვდის მომხმარებლებს ან სხვა პროცესებს, ორგანიზაციის შიგნით ან გარეთ;

- **პროცესები შედგება ქმედებებისგან.** ქმედება (Activity) არის საქმიანობის ძირითადი სახეები პროცესის ფარგლებში.

COBIT განიხილავს 34 IT-პროცესს, რომლებიც გაერთიანებულია 4 დომენში (Domain – საკონტროლო მიზნების დაჯგუფება ლოგიკურ ეტაპებში IT-ინვესტიციის სასიცოცხლო ციკლის შიგნით). 8.2 ნახაზზე მოცემულია დომენების ურთიერთკავშირის სქემა.



ნახ.8.2

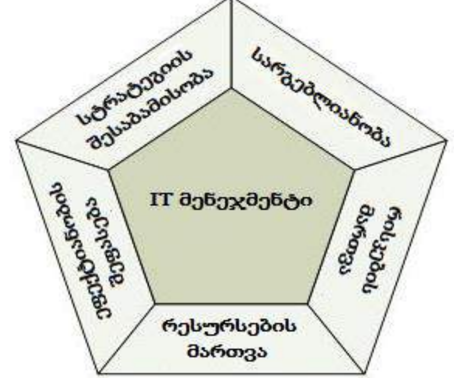
- **დაგეგმვა და ორგანიზება (PO – Plan and Organise):** განსაზღვრავს მიმართულებებს გადაწყვეტილებათა დანერგვის (AI) და სერვისების მიწოდების (DS - Delivery Services) თვალსაზრისით;
- **მოპოვება და დანერგვა (AI – Acquire and Implement)**: უზრუნველყოფს გადაწყვეტილებათა დანერგვას და სერვისებს მათ საფუძველზე;
- **ექსპლუატაცია და თანხლება (DS – Deliver and Support)**. უზრუნველყოფს გადაწყვეტილებათა შესრულებას და საბოლოო მომხმარებლებისთვის მათი გამოყენების მხარდაჭერას;
- **მონიტორინგი და შეფასება (ME – Monitor and Evaluate).** ახორციელებს პროცესების მონიტორინგს (კონტროლს) და შეფასებას.

პროცესების ასეთი სტრუქტურა იძლევა სფეროების სისტემატიზაციის და ინფორმაციის ორგანიზების უზრუნველყოფის საშუალებას, რომლებიც აუცილებელია მათი ბიზნეს-მიზნების მისაღწევად.

3. **რესურსების რანჟირების პრინციპი.** არაა აუცილებელი ყველა რესურსის თვალყური, მხოლოდ იმათზე უნდა გამახვილდეს ყურადღება, რომლებიც გავლენას ახდენს ბიზნეს-პროცესებზე და მათ შედეგებზე.

4. **მიზნების განსაზღვრა** – ერთ-ერთი ყველაზე რთული და მნიშვნელოვანი ამოცანაა. გლობალური გაგებით ხელმძღვანელობა და მენეჯერები ორ მიზანს ითვალისწინებენ – დასმული ბიზნეს-მიზნების მიღწევას და არასასურველი მოვლენების თავიდან აცილება (ან მათი შედეგების გამოსწორება). მაგალითად, სარეზერვო

კოპირების ამოცანას არ მოაქვს პირდაპირი მოგება ბიზნესისათვის, მაგრამ სისტემის მწყობრიდან გამოსვლისას მისი საშუალებით შესაძლებელია ინფორმაციის სწრაფი აღდგენა, რაც მეტად მნიშვნელოვანია ბიზნესის ნორმალური ფუნქციონირებისათვის. ასევე მნიშვნელოვანია საკითხები ხელმძღვანელობისა-თვის, მაგალითად, სადაა საჭირო პროცესების სრულყოფა, რამდენი ინვესტიციაა საჭირო და როგორ გაიზომოს შედეგი. COBIT იძლევა მეთოდლოგიას IT-ხელმძღვანელობისთვის დას-მულ საკითხებზე. COBIT გამოყოფს IT-მენეჯმენტის შემდეგ საკვანძო სფეროებს (ნახ.8.3):



ნახ.8.3. IT-მენეჯმენტის საკვანძო სფეროები

სტრატეგიის შესაბამისობა. უზრუნველ-ყოფს ერთმანეთთან ბიზნესისა და IT-ის თავსებადობას;

- **სარგებლიანობა** პასუხს აგებს: იმის რეალიზაციაზე, რასაც შეუძლია ბიზნესისათვის ფასეულობის მოტანა; კონტროლზე, რათა IT-იმ უზრუნველყოს სტრატეგიით განსაზღვრული უპირატესობები; ხარჯების ოპტიმიზაციასა და ჭეშმარიტი ღირებულების დადასტურებაზე;
- **რესურსების მართვა** პასუხისმგებელია კრიტიკული IT-რესურსების მენეჯმენტზე, ინვესტიციების ოპტიმიზაციაზე და აპლიკაციების, ინფორმაციის, ინფრასტრუქტურის და პერსონალის სათანადო ხელმძღვანელობაზე;
- **რისკების მართვა** მოითხოვს ზემდგომი ხელმძღვანელობის ინფორმირებას რისკების სფეროში; კორპორაციული მიდგომის ნათლად წარმოდგენას მათთან მიმართებით; გამჭვირვალობის მოთხოვნების შესაბამისობას არსებულ რისკებთან დამოკიდებულებით; ორგანიზაციის პრაქტიკაში რისკების მართვის ფუნქციების დანერგვას;
- **ეფექტიანობის შეფასება** პასუხს აგებს სტრატეგიის, გეგმების, რესურსების გამოყენებისა და პროცესების ეფექტიანობის რეალიზაციის კონტროლზე.

IX თავი

პროგრამული აპლიკაციების ინტეგრაციის თანამედროვე ტექნოლოგიები

9.1. მართვის საინფორმაციო სისტემებში ინტეგრაციისა და მონაცემთა მენეჯმენტის ტექნოლოგიები

თანამედროვე საინფორმაციო სისტემებში პროგრამული უზრუნველყოფების უმრავლესობა დაკავშირებულია ერთმანეთთან. მათ შორის წარმოებს ინფორმაციის მიმოცვლა [1,2]. ეს აპლიკაციები შესაძლებელია განთავსებული იყოს როგორც კორპორაციის შიგა ქსელში, ისე გლობალურ ქსელშიც. ორგანიზაციები ფლობს სხვადასხვა პროგრამას, რომლებიც სხვადასხვა მიზანს ემსახურება. ასევე კომპანიებს შორის ხდება ელექტრონული სახით მონაცემების გაცვლა (Electronic Data Interchange), სახელმწიფო სტრუქტურებთან, ასევე ტერიტორიულად დაშორებულ განყოფილებებთან [9]. ყველა ამ პროგრამას ესაჭიროება სათანადო სახის კავშირი სხვა სისტემებთან, რაშიც იგულისხმება მონაცემების გაცვლა-დამუშავება, ამათუიმ ბიზნესლოგიკის შესაბამისად.

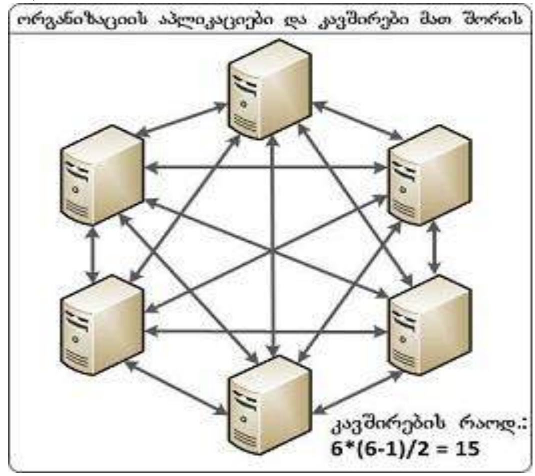
სხვადასხვა ტიპის აპლიკაცია, როგორიცაა პროდუქციის მიწოდების (სასაწყობო მარაგების მართვისა და მიწოდების ორგანიზება), კლიენტებთან ურთიერთქმედების სისტემები (არსებულ და პოტენციურ კლიენტებთან), ბუღალტრული აღრიცხვის, საწარმოთა რესურსების მართვის, Business Intelligence-ს სისტემებთან (OLAP), ადამიანური რესურსების მართვის, ჯანდაცვისა და სხვა აპლიკაციებს, როგორც წესი, საჭიროებს ერთმანეთთან ურთიერთობას, მონაცემების გაცვლის და სხვადასხვა ბიზნესპროცესების მართვის კუთხით [10-13]. ეს სისტემები მუშაობს სხვადასხვა ოპერაციული სისტემის სერვერზე, სხვადასხვა სახის მონაცემთა ბაზაზე, შესრულებულია დაპროგრამების განსხვავებული ენების, განსხვავებული ტექნოლოგიების გამოყენებით ან მოძველებულ სისტემებზე, რომელთა განვითარება და მხარდაჭერაც აღარ ხდება. რის გამოც რთულია მათ შორის პროცესების მართვა, ავტომატიზაცია. ასევე ხდება მონაცემების დუბლირება, მათი სხვადასხვა სისტემებაში განთავსების გამო.

სისტემების ინტეგრაციის (Enterprise Application Integration) ამოცანაა დააკავშიროს და გაამარტივოს ამ კომპანიაში არსებული სისტემების, აპლიკაციების ურთიერთკავშირი და უზრუნველყოს ბიზნესპროცესების ავტომატიზაცია [14].

ამ კავშირების უზრუნველყოფისთვის იქმნება სხვადასხვა პროგრამა, რომლებიც ასრულებს დამაკავშირებელ, შუალედურ რგოლს ამ სისტემებს შორის. მსგავსი ბროკერული აპლიკაციების რაოდენობა იზრდება, რაც უფრო იზრდება ორგანიზაციაში სისტემების და ასევე გარე ორგანიზაციებთან არსებული კავშირები, საჭირო ხდება ამ ბროკერული აპლიკაციების შექმნა, იზრდება მათი ადმინისტრირების, მონიტორინგის,

პროგრამული უზრუნველყოფის ცვლილებების მართვის სამუშაოები, რაც იწვევს ინფორმაციული ტექნოლოგიების რესურსების დიდი ოდენობით მოხმარებას. საჭირო ხდება როგორც ადამიანური რესურსის, ისე ინფრასტრუქტურული, პროგრამული უზრუნველყოფის მუდმივი განახლება. თითოეული ბროკერული აპლიკაციის შექმნა არის ხანგრძლივი და შრომატევადი პროცესი.

თუ ორგანიზაციას სჭირდება N რაოდენობის აპლიკაციების ერთმანეთთან დაკავშირება ამ აპლიკაციებს შორის ბროკერების რაოდენობამ შეიძლება მიაღწიოს N(N-1)/2 რიცხვს, რაც, მაგალითად 10 აპლიკაციის შემთხვევაში არის 45, რაც საკმაოდ დიდი რიცხვია და ზემოთქმულიდან გამომდინარე რესურსების დიდ ხარჯებთან არის დაკავშირებული (ნახ.9.1).



ნახ.9.1

სხვადასხვა სისტემა განსხვავებული ფორმატის მონაცემებს იძლევა გარე სისტემებთან დასაკავშირებლად. შესაძლებელია ეს იყოს ტექსტური ან ორობითი ფაილი, მონაცემები იყოს XML ფორმატში, ვებსერვისების საშუალებით იძლეოდეს მონაცემებზე წვდომის საშუალებას, ან ბაზის სხვადასხვა პროცედურების და ფუნქციების საშუალებით, ამასთანავე მონაცემთა გადაცემა ხდებოდეს TCP, HTTP პროტოკოლების გამოყენებით, თუ მონაცემთა გადაცემის სხვა სახის საშუალებებითაც. საჭიროა სხვა სახის ინტეგრაციული არქიტექტურის შექმნა, რათა კომპანიამ უფრო მარტივად უზრუნველყოს ინტეგრაციის ამოცანები ნაკლები რესურსების გამოყენებით.

ამოცანა მდგომარეობს ისეთი სისტემის შექმნაში, რომელიც უზრუნველყოფს არსებული სისტემების ინტეგრაციას, როგორც სხვადასხვა ორაგნიზაციას შორის, ისე ორგანიზაციის შიგნით, რომელიც უზრუნველყოფს სხვადასხვა ფორმატის მონაცემის მიღება-გადაცემა-დამუშავებას, მაღალ წარმადობას და საიმედოობას [18].

აპლიკაციების ინტეგრაციას ძირითადად აქვს სამი დანიშნულება:

- მონაცემთა ინტეგრაცია, რაც უზრუნველყოფს მონაცემების იდენტურობას სისტემებს შორის;
- აპლიკაციათა მიმწოდებლებზე დამოუკიდებლობა. უზრუნველყოფს, რომ აპლიკაციის ცვლილების შემთხვევაში, ბიზნესპროცესი და ბიზნესწესების ხელახალი შექმნა არ იყოს საჭირო;
- ფასადის/ინტერფეისის შექმნა, რაც უზრუნველყოფს აპლიკაციებთან ერთიერთობის ერთიანი ინტერფესის შექმნას, რომელიც საშუალებას იძლევა აპლიკაციებთან კომუნიკაცია შესრულდეს მათი შიგა სტრუქტურების შესწავლის გარეშე.

ამ ამოცანის რეალიზაცისათვის საჭიროა გამოყენებული იყოს ინტეგრაციის აპრობირებული მეთოდები, პროგრამული უზრუნველყოფის არქიტექტურის მოდელები, სტანდარტები, რომლებიც უზრუნველყოფს მოქნილობას, მასშტაბირებას, არაერთგვაროვანი და კომპლექსური სისტემების ურთიერთკავშირს.

- **ორგანიზაციის აპლიკაციების ინტეგრაცია** (**Enterprise Application Integration**).

ორგანიზაციის აპლიკაციების ინტეგრაცია (EAI) არის ტექნოლოგია, რომლის დანიშნულებაცაა საწარმოს აპლიკაციების ერთმანეთთან დაკავშირება, მათი ერთიან ბიზნესპროცესში ჩართვა, ასევე მათ შორის გადაცემული მონაცემების ფორმატის ტრანსფორმაცია [14,15,22].

EAI მეთოდის გამოყენება ხდება როდესაც საჭიროა [34]:

- აპლიკაციის აპლიკაციასთან დაკავშირება (Application to Application);
- ადამიანის სისტემასთან კავშირი (Person to System);
- ბიზნესის ბიზნესთან კავშირი (Business to Business).

EAI მეთოდის საშუალებით ხდება აპლიკაციებს შორის პირდაპირი კავშირების დამყარების აუცილებლობის გამორიცხვა [32]. მისი რეალიზაციისას ხდება სხვადასხვა მეთოდების, სქემების და სხვა პროგრამული უზრუნველყოფების გამოყენებით სხვადასხვა აპლიკაცბის პროცესორიენტირებული ინტეგრაცია.

განსხვავებული ფორმატის მონაცემების მიღება და ტრანსფორმაცია სპეციალური ადაპტერების საშუალებით სრულდება [36]. EAI საშუალებას იძლევა ცენტრალიზებულად განისაზღვროს ბიზნესპროცესების ლოგიკა, რაც საშუალებას იძლევა სათანადო ბიზნესლოგიკის ცვლილებისას. აპლიკაციების გადაკეთების გარეშე განხორციელდეს ახალი წესების იმპლემენტაცია. ინტეგრაციის არხი ზრუნავს მონაცემები დამუშავდეს წინასწარ განსაზღვრული წესების შესაბამისად და შედეგები გადამისამართდეს კონკრეტული ბიზნესპროცესის შესრულებისათვის [33].

- **სერვის ორიენტირებული არქიტექტურა (Service Oriented Architecture).**

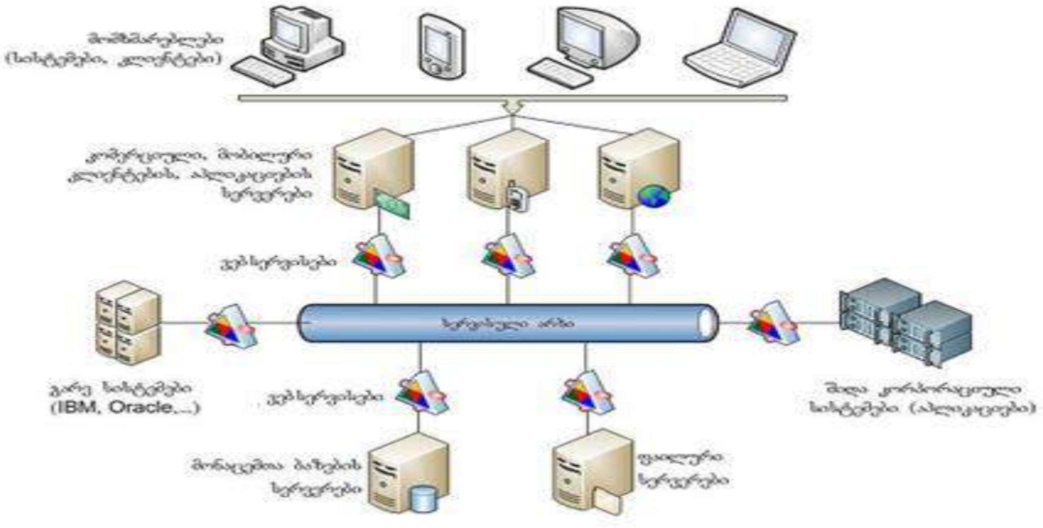
სერვის ორიენტირებული არქიტექტურა ერთმანეთთან ურთიერთმოქმედი სისტემების პროექტირების მეთოდოლოგიაა [16]. EAI-ის შესაძლებლობები ფართოვდება სერვისორიენტირებული არქიტექტურის გამოყენებით, რომელიც სხვადასხვა

აპლიკაციისა და სხვადასხვა ორგანიზაციას შორის სისტემა ინტეგრაციას და ბიზნესამოცანის შესრულებას უზრუნველყოფს [32]. SOA-ს შემთხვევაში სხვადასხვა კომპონენტი განიხილება, როგორც სათანადო მომსახურების მომწოდებელი ან მომხმარებელი სერვისი. მისი საშუალებით ხდება განაწილებულ მონაცემთა ბაზების, ვებსერვისების სხვა სერვისის სტრუქტურირება სათანადო ბიზნესპროცესის შესასრულებლად.

მაგალითად, ბანკში კლიენტისათვის სესხის გაცემა შედგება სხვადასხვა პროცესისგან: ბანკის კლიენტის შექმნა, ანგარიშის გახსნა, სასესხო ხელშეკრულების შექმნა და სხვა. ეს პროცესები შესაძლებელია სრულდებოდეს სხვადასხვა აპლიკაციის მიერ, SOA-ს საშუალებით კი ეს პროცესი შესრულდება სათანადო მიმდევრობისა და ბიზნესპირობების დაცვის საშუალებით, რასაც ორკესტრაცია უზრუნველყოფს [19,22]. ერთი და იგივე სერვისი შესაძლებელია გამოყენებულ იქნას სხვადასხვა ამოცანებისათვის [9]. საბოლოოდ, ასეთი მოდგომა საშუალებას იძლევა დაიზოგოს რესურსები პროგრამულ უზრუნველყოფის შექმნაზე ახალი ამოცანებისთვის უკვე არსებული სერვისების გამოყენების საშუალებით.

- **ორგანიზაციის სერვისული არხი (Enterprise Service Bus).**

ESB წარმოადგენს განაწილებულ სისტემების არქიტექტურულ მოდელს [15]. სხვადასხვა აპლიკაციას შორის კავშირის დასამყარებლად ამ მოდელში გამოიყენება შუალედური პროგრამული საშუალება, რომელიც უზრუნველყოფს ტრანზაქციული ოპერაციის შესრულებას, შეტყობინებათა მარშრუტიზაციას, ტრანსფორმაციას, მონაცემების დაცვას [27]. მისი საშუალებით შესაძლებელია სერვისორიენტირებული არქიტექტურის რეალიზაცია (ნახ.9.2).



ნახ.9.2. სერვისორიენტირებული არქიტექტურა

იგი შედგება დამოუკიდებელი კომპონენტებისაგან, რომლებიც განთავსებულია სერვისულ არხში და საჭიროებისამებრ ურთიერთქმედებს ერთმანეთთან. სხვადასხვა სისტემა შეტყობინებებს გადასცემს ორგანიზაციის სერვისულ არხს, რომელიც ახდენს მათ მიწოდების დაყოვნებას, პრიორიტეტიზაციას, გადავადებას ბუფერში მიმღები სერვისის განთავისუფლებამდე, მონაცემების სისწორის შემოწმებას, მონაცემების ტრანსფორმაციას, პროცესების კონტროლსა და შეცდომებზე რეაგირებას [30].

- **მონაცემების ელექტრონული გაცვლა (Electronic Data Interchange).**

მონაცემების ელექტრონული გაცვლა (EDI) აღნიშნავს მონაცემების ელექტრონულ დამუშავებას ელექტორნული ტრანსპორტირების მეთოდების გამოყენებით. მასში მონაწილეობს სხვადასხვა ორგანიზაციის გამოყენებითი სისტემები. EDI სტანდარტები განსაზღვრავენ მონაცემთა გაცვლის მეთოდებს და რეგულაციას ორგანიზაციებს ან მათ ფილიალებს შორის. მისი დანიშნულებაა ციფრული ინფორმაციის გადაცემის სტანდარტიზება, სხვადასხვა დანიშნულების კომპიუტერული სისტემების პროგრამული ურთიერთქმედების უზრუნველყოფა.

EDI გულისხმობს რამდენიმე ორგანიზაციას შორის შეთანხმებას გარკვეული მნიშვნელობის, ფორმატის, კავშირების საშუალებით მონაცემთა გაცვლაზე [17]. EDI-ის გამოყენების ძირითადი სარგებლობა არის მონაცემების ელექტრონული გადაცემის მაღალი სიჩქარე, ადამიანის ჩარევის გარეშე ბიზნესის მონაცემების გადაცემა, ადამიანური შეცდომების გამორიცხვა.

არსებობს მონაცემთა გადაცემის საერთაშორისო სტანდარტები, მაგალითად SWIFT-საბანკო გადარიცხვებისათვის, GAEB-მშენებლობაში, VDA-საავტომობილო ინდუსტრიაში და სხვ. [17]. მონაცემების გადაცემა სრულდება ასევე გავრცელებული პროტოკოლებით: SMTP, HTTP, FTP და სხვ. [18].

- **ბიზნესპროცესების მენეჯმენტი (Business Process Management).**

ბიზნესპროცესების მენეჯმენტი (BPM) მოიცავს საწარმოო პროცესების იდენტიფიკაციას, დოკუმენტირებას, დაგეგმვას, მართვას და გაუმჯობესებას [20]. იგი მიმართულია არამხოლოდ ტექნიკურ, არამედ ორგანიზაციული საკითხების მოსაგვარებლადაც, სტრატეგიული მიმართულებების, ორგანიზაციული კულტურის, მონაწილე მხარეების ინტეგრაციისა და მართვის საკითხების დასახვეწად.

ბიზნესპროცესების ავტომატიზაციის პრობლემები შესაძლებელია სამ ჯგუფად დავყოთ:

- კომპანიის შიგნით პროგრამული უზრუნველყოფების dakavSireba Enterprise Application Integration (EAI);
- სხვადასხვა კომპანიას პროგრამული უზრუნველყოფების შორის კავშირის დამყარება Business-to-Business B2B ინტეგრაცია;

- ბიზნესპროცესების ავტომატიზაციის განზოგადებული მიდგომის უზრუნველყოფა, რომელიც განსაზღვრულია ბიზნეს პროცესების მენეჯმენტის (BPM) ის მიხედვით.

ბიზნესპროცესების მენეჯმენტის მიზანია, ორგანიზაციაში არსებული პროცესების გაუმჯობესება, უკეთესი მომსახურების, კლიენტების კმაყოფილების, პროდუქციის ხარისხის გაუმჯობესება, წარმოების ეფექტურობის ამაღლება და ორგანიზაციის მიზნების მიღწევა [19,21].

- **აპლიკაციების ინტეგრაციის სისტემები.**

სისტემების დაკავშირებისათვის ცალკეული ბროკერული აპლიკაციების შექმნა და მათი მართვის არაეფექტურობის გამო, უმჯობესია გარკვეული სისტემის გამოყენება, რომელიც უზრუნველყოფდა სხვადასხვა ფორმატისა და არხით მიღებული მონაცემების დამუშავებას, ტრანსლიაციასა და მის გადაცემას.

ამ სისტემის დანიშნულებაა სხვადასხვა ტექნოლოგიაზე, პლატფორმაზე შექმნილი განსხვავებული ბიზნესაპლიკაციის დაკავშირება, ბიზნეს პროცესების ავტომატიზაცია და მისი მონიტორინგი [17,29].

ინტეგრაციის ამოცანების გადაწყვეტა შესაძლებელია სხვადასხვა სისტემის გამოყენებით. ეს სისტემები უზრუნველყოფს ორგანიზაციის სერვისული არხის (ESB), სერვის ორიენტირებულ არქიტექტურის (SOA), მონაცემთა ელექტრონული გაცვლის (EDI), ორკესტრაციის, პროცესების მართვის ინსტრუმენტების საშუალებით სერვისების ინტეგრაციას, ბიზნესპროცესების დამუშავებას და ავტომატიზაციას (BPM), სერვისების, მონაცემების, პროგრამული უზრუნველყოფების დაკავშირებას. ასეთ სისტემებს შორის შესაძლებელია გამოიყოს მსოფლიოში წამყვანი პროგრამული უზრუნველყოფის მწარმოებელი კომპანიების პროდუქტები, როგორიცაა:

- JBoss Enterprise SOA Platform – RedHat-ისგან;
- Oracle Enterprise Service Bus – Oracle-ისგან;
- BizTalk Server – Microsoft-ისგან;
- WebSphere Process Server – IBM-ისგან;
- SAP Exchange Infrastructure – SAP AG-სგან;
- webMethods - Software-სგან AG.

ეს პროდუქტები ძირითადად განსხვავდება ოპერაციული სისტემების, მონაცემთა ბაზების ტიპების მხარდაჭერის შესაძლებლობებით. ამ სისტემებში ინტეგრირებულია სხვადასხვა მზა ადაპტერი, რომლებიც სხვადასხვა აპლიკაციის დაკავშირების საშუალებას იძლევა, რომელთაც მონაცემთა სხვადასხვა ფორმატი აქვს ასევე არის ბიზნესპროცესების სქემების აგების, ავტომატიზაციის შესაძლებლობა. აგრეთვე პროცესებზე მონიტორინგის, სხვადასხვა პარამეტრის გაზომვის ინსტრუმენტი (Business Activity Monitoring); მონაცემების გადაცემის, ტრანსფორმაციის დროს წარმოშობილი შეცდომების დიაგნოსტიკა. ასევე კონფიგურაციების საშუალებით სისტემის

ფუნქციონირების, ბიზნეს წესების პარამეტრიზიცია. მათ ფუნქციებს მიეკუთვნება ასევე მონაცემების დაცვის უზრუნველყოფა, უსაფრთხოება, მაღალი წარმადობა, საიმედოობა.

ამ სისტემების გამოყენების უპირატესობებია:

- სისტემის ელემენტების შექმნისა და ბიზნესლოგიკის დამუშავების სიმარტივე;
- მონაცემებისა და პროცესების მონიტორინგის შესაძლებლობა, ასეცე მონაცემების ანალიზი;
- ბიზნე მომხმარებლის მიერ ბიზნესლოგიკის ცვლილების შესაძლებლობა;
- მონაცემთა უსაფრთხო მიღება-გადაცემა;
- სერვერების ჯგუფების შექმნა დატვირთვების გასანაწილებლად და საიმედოობის ასამაღლებლად და სხვ.

შეიძლება დავასკვნათ, რომ მზარდი მოთხოვნა ინტეგრაციულ სისტემებზე, მონაცემთა დამუშავების ავტომატიზაციის ამოცანების, სქემების, მიდგომების, პრინციპების მუდმივი ცვლილება, ასევე ქსელური და აპარატურული ტექნოლოგიების შესაძლებლობების ზრდა განაპირობებს ამ კუთხით წარმოებული პროდუქტების ფუნქციურ განვითარებას.

ამგვარად, რთული კორპორაციული და კორპორაციათაშორის სისტემებში აუცილებელია თანამედროვე ინტეგრაციული საშუალებების გამოყენება, რომლებიც უზრუნველყოფს სერვისორიენტირებული არქიტეტურის რეალიზაციას. იგი დაფუძნებული იქნება ორგანიზაციის ინტეგრაციის არხზე. მსგავსი მიდგომა თავის მხრივ განაპირობებს ელექტრონული ბიზნესპროცესების მსვლელობას სისწრაფის ზრდას, ეფექტურობას, საიმედოობას, უსაფრთხოობას.

BizTalk ორგანიზაციის სერვისული არხის იმპლემენტაციის პლატფორმაა, რომელიც გამოიყენება სხვადასხვა აპლიკაციას შორის კავშირის დასამყარებლად. გრაფიკული ინტერფეისის საშუალებებით შესაძლებელია ბიზნესპროცესების ორკესტრაცია. რეალიზებულია პროცესებზე მონიტორინგის ბიბლიოთეკები, როგორც პროცესების რაოდენობრივი შეფასებისთვის, ისე სათანადო მოვლენებზე რეაგირებისათვის. BizTalk-ის დახმარებით შესაძლებელია პროცესების ავტომატიზაციის და ინდუსტრიული სტანდარტების უზრუნველყოფა, რაც საშუალებას იძლევა შემცირდეს დანახარჯები და კომპლექსურობა B2B კავშირების დასამყარებლად.

9.2. სერვისორიენტირებული არქიტექტურის რეალიზაციის საშუალება Ms BizTalk Server

კომპანიების საინფორმაციო ტექნოლოგიების განვითარება სულ უფრო მიმდინარეობს სერვისორიენტირებული არქიტექტურის (SOA) მიმართულებით. BizTalk_Server სისტემის დანიშნულებაა სხვადასხვა ტექნოლოგიაზე, პლატფორმაზე შექმნილი განსხვავებული ბიზნესაპლიკაციის დაკავშირება, ბიზნესპროცესების

ავტომატიზაცია და მათი მონიტორინგი [1,21]. ინტერკორპორაციულ სისტემებში BizTalk-ის გამოყენების უპირატესობები შემდეგია:

- სისტემის ელემენტების შექმნისა და ბიზნესლოგიკის დამუშავების სიმარტივე;
- მონაცემთა და პროცესების ანალიზისა და მონიტორინგის შესაძლებლობა;
- ხარვეზების დიაგნოსტიკის სიმარტივე;
- ბიზნესმომხმარებლის მიერ ბიზნესლოგიკის ცვლილების შესაძლებლობა;
- მონაცემთა უსაფრთხო მიღება-გადაცემა;
- სერვერების ჯგუფების შექმნა დატვირთვების გასანაწილებლად და საიმედოობის გასაზრდელად.

BizTalk შედგება სხვადასხვა ქვესისტემისგან:

- **მონაცემთა მიღებისა და გადაცემის ქვესისტემა.** მისი დანიშნულებაა მონაცემების მიღება სხვადასხვა წყაროდან და მათი გადამისამართება გამავალ წყაროებში. ამ სისტემის კონფიგურაცია შესაძლებელია ადმინისტრირების საშუალებით BizTalk Administration. მიმღებ პორტზე ადაპტერის საშუალებით მონაცემების მიღება რაიმე ფიზიკური წყაროდან (ფაილი, ვებ-სერვისი, Sharepoint, ელ-ფოსტა). ადაპტერი მიღებულ ინფორმაციას გარდაქმნის XML ფორმატში. XML-ად გარდაქმნის ეტაპზე შესრულდება მონაცემების შიფრაცია, დეკოდირება, შესაბამისობაზე შემოწმება და სხვა მოქმედებები.

შემდეგ ეტაპზე ხდება იმისგან, დამოუკიდებლად XML დოკუმენტის გარდაქმნა საჭირო ფორმატში. ეს საჭიროა თუ რა სახის ინფორმაცია იყო წარმოდგენილი. რადგანაც ბიზნესლოგიკას დასამუშავებლად მონაცემები გადაეცემა XML ფორმატში.

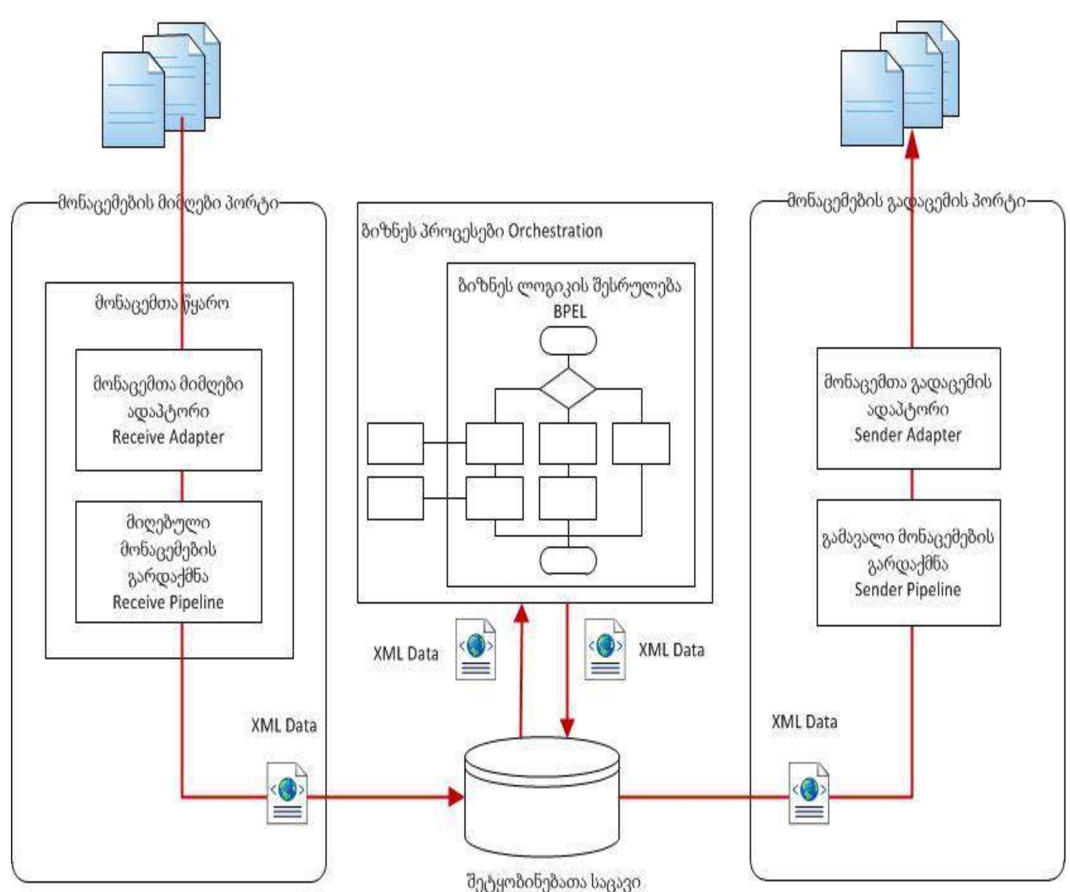
შემდგომ შეტყობინება ხვდება შეტყობინებათა ბაზაში, რის შემდეგაც განისაზღვრება დამუშავდეს ბიზნესლოგიკის შესაბამისად, თუ გადაიგზავნოს სხვაგან.

შეტყობინებათა გაგზავნის ეტაპები შეტყობინების მიღების ანალოგიურია.

სისტემის აღწერისას უნდა აღინიშნოს, რომ მისი ელემენტების შექმნა შესაძლებელია როგორც სტანდარტული ბლოკების საშუალებით, ისე ელემენტების მოთხოვნილებათა მიხედვით (ადაპტერები, რომლებიც არასტანდარტული პროტოკოლებით იღებს შეტყობინებას, ასევე Pipeline, რომელიც შეასრულებს, მაგალითად, მონაცემების არქივაციას მათ გაგზავნამდე. შეტყობინებების დამუშავება ნიშნავს შემავალი ნაკადების გარდაქმნას რამდენჯერმე და ახალი ნაკადების ფორმირებას.

9.3 ნახაზზე მოცემულია BizTalk სერვერის არქიტექტურა და შეტყობინებათა დამუშავების პროცესის სქემა.

ორკესტრაცია. შეტყობინების მიღების შემდეგ შეტყობინებათა ბაზაში იწყება მისი დამუშავება ბიზნესლოგიკის მიხედვით. იგი შეიძლება შედგებოდეს სხვადასხვა საფეხურისგან, მაგალითად, შეტყობინებათა ტრანსფორმაცია და ახალი შეტყობინების ფორმირება.



ნახ.9.3. BizTalk Server-ის არქიტექტურა და შეტყობინებათა
დამუშავების სქემა

ოპერაციები შესრულდება როგორც მიმდევრობით, ისე პარალელურად, განტოტვილი ლოგიკითა და ციკლების საშუალებით, ტრანზაქციების შესაძლებლობით ჩაშენებული ბიზნესლოგიკის შესასრულებლად. ბიზნესლოგიკის განსახორციელებლად შესაძლებელია შეიქმნას ერთი ან რამდენიმე შეტყობინება, რომლებიც ჩაიწერება შეტყობინებათა ბაზაში და შემდგომ მოხდება გადაცემა გამავალ პორტებზე ან სხვა ბიზნესლოგიკაზე.

წესების მექანიზმი. ეს მექანიზმი სშუალებას იძლევა განისაზღვროს წესების კრებული, რომლებიც გამოიყენება გამავალი შეტყობინების მოდერნიზაციისა თუ სხვა მონაცემების შექმნისთვის. წესების ჩაშენება შესაძლებელია ბიზნესლოგიკაში, და შემდგომ შესაძლებელია მათი ცვლილება, პარამეტრიზაციის საშუალებით.

ხარვეზების დიაგნოსტიკა. ნებისმიერი სერვისის გაუმართავი მუშაობის დიაგნოსტორება, შეცდომების გამოვლენა და კორექტირება არის საკმაოდ შრომატევადი ამოცანა, რადგან აპლიკაციების უდიდესი ნაწილი პროგრამული კოდის წერის გარეშე იქმნება, შესაბამისად Visual Studio-ს საშუალებით მათი გამართვა შეუძლებელია.

ამ პრობლემის გადასაჭრელად BizTalk-ში შედის ინსტრუმენტები, რომელთა საშუალებითაც შეიძლება დადგენა, თუ როგორ მუშაობს BizTalk, რა შეტყობინება შემოვიდა, როგორ დამუშავდა, რომელი კომპონენტები იქნა ამოქმედებული.

BizTalk Administration უტილიტის საშუალებით შესაძლებელია დადგინდეს შეტყობინება და ელემენტთა ეგზემპლარები, რომელთა დამუშავებისას წარმოიქმნა შეცდომა. ასეთ შეტყობინებათა დამუშავება შეჩერდება და წყდება საერთოდ. ასევე შესაძლებელია ამ უტილიტის საშუალებით მიეთითოს, თუ რა ეტაპებზე მოხდეს შეტყობინებების მონიტორინგი და რამდენად დეტალურად. ამ უტილიტის საშუალებით შესაძლებელია მოინახოს რა შეტყობინება საიდან იქნა მიღებული, რომელი კომპონენტის მიერ იყო დამუშვებული და რა შეტყობინებები შეიქმნა, ასევე შესაძლებელია შეტყობინების შიგთავსის ნახვა.

ბიზნესაქტიურობის მონიტორინგი (Business Activity Monitoring - BAM). ბიზნესმომ-ხმარებლებს სჭირდებათ ეფექტურობის კოეფიციენტის განსაზღვრა (Key Performance Indicators (KPIs), რომლითაც იზომება ბიზნესმიზნების შესრულების დონე, და სხვა ბიზნესპროცესების მაჩვენებლები. ეს ქვესისტემა საშუალებას იძლევა შეაგროვოს სხვადასხვა ინფორმაცია ბიზნესპროცესზე და შექმნას მონაცემთა კუბები ბიზნესმომხმარებლის მხრიდან გასაანალიზებლად. მონაცემების ფორმირება შესაძლებელია როგორც რეალურ დროში, ისე განსაზღვრული მომენტისთვის. Analysis Service–ს შეუძლია წარმოადგინოს მონაცემები სხვადასხვა სახით, გაფილტროს, დააჯგუფოს როგორც საჭირო იქნება პროცესების შეფასებისათვის, შესაბამის გადაწყვეტილებათა მისაღებად.

BAM შედგება 3 კომპონენტისაგან: აგრეგაციის, აქტიურობის ძებნისა და შეტყობინებების სერვისისაგან. აგრეგაციის საშუალებით შესაძლებელია ბიზნესპროცესების პარამეტრების გაზომვა (მაგალითად, თვის განმავლობაში ფილიალების მიხედვით გაყიდვების რაოდენობა) და მომხმარებლებისთვის სხვადასხვა კრიტერიუმების შესაბამისად გრაფიკული ასახვა. აქტიურობათა ძებნის საშუალებით შესაძლებელია დადგინდეს, აღიძრა თუ არა სისტემაში გარკვეული ტიპის პროცესი (მაგალითად, განსაზღვრულმა კლიენტმა მოახდინა თუ არა შეკვეთა, მოხდა თუ არა განსაზღვრული ოდენობის საქონლის შეკვეთა და სხვ.). შეტყობინებათა სერვისით კი შესაძლებელია მომხმარებელს მიუვიდეს შეტყობინება, სრულდება თუ არა ის პირობა, რომლის მონიტორინგიც ხდება, აქტიურობის ძებნის თუ აგრეგაციების საშუალებით.

ამგვარად, Ms BizTalk Server არის სტრატეგიული ინსტრუმენტი, რომელიც გამოყენებადს ხდის იმ ინფრაქსტრუქტურას, რომელიც მომარაგებულია ინტერნეტის

მიერ ელექტრონული ბიზნესაპლიკაციებისათვის ადვილი ქსელურობისათვის, მომხმარებლებისათვის გაზრდილი მომსახურებისათვის, მიწოდების ჯაჭვის მართვისათვის, პროდუქტიული კომუნიკაციისათვის და სხვ.

საინფორმაციო ტექნოლოგიები აუმჯობესებს სწრაფქმედებას, არსებულ ინფორმაციაზე წვდომას, მაგრამ იმისათვის, რომ შეიმნას ახალი ინფორმაცია და გამოვიყენოთ არსებული ინფორმაცია უფრო ახალი და უფრო მეტად ინოვაციური გზით, საჭიროა შესაბამისი ინსტრუმენტი, რომელსაც შეეძლება ინფორმაციის შეფასება აქამდე გამოუკვლეველი გზებით. BizTalk Server არის ერთ-ერთი ასეთი ინსტრუმენტი. შეცდომა იქნება, თუ მას მარტივ ტექნოლოგიად განვიხილავთ.

ქვემოთ ახსნილია, თუ როგორ ითავსებს ეს ახალი პროგრამული უზრუნველყოფა იმ უპირატესობებს, რომელსაც მცირე და საშუალო საწარმოები (SME - Small and Medium-sized Enterprise) აღწევს მისი ეფექტური გამოყენებით. ბიზნესმენების, ვებსერვისების დეველოპერების, ბიზნესანალიტიკოსების, ტექნოლოგიის მგეგმავების, პროექტების მენეჯერების, აპლიკაციათა დიზაინერებისა და შემსრულებლებისათვის იგი ძალზე გამოყენებადი პროგრამული უზრუნველყოფაა. იგი საშუალებას იძლევა მისი ინფრასტრუქტურით აიგოს წარმატებული ელექტრონული საწარმოების კავშირი.

ეს ახალი თაობის პროგრამული უზრუნველყოფა მთლიანად იყენებს გაფართოებადი მარკირების ენის (XML) შესაძლებლობებს. ინტერნეტის საშუალებით ინფორმაციისა და მონაცემების გადაცემა შესაძლებელია ძალიან გართულდეს, განსაკუთრებით მაშინ, როდესაც სხვადასხვა კომპანიას აპლიკაციები დაწერილია სპეციალურ უნიკალურ ფორმატში, რომელთა გადაცემაც სირთულეს წარმოადგენს. ბიზნესმონაცემებისა და პროცესების აღსაწერად ერთი საერთო „ტექნიკური ლექსიკონის" არქონა, აუცილებელს ხდის ისეთი ენის გამოყენებას, როგორიცაა XML, რომელიც ზუსტად განსაზღვრავს ჩვეულებრივ კოდს/მნიშვნელობას ყველა იმ მონაცემისა, რაც ფართო ქსელში გადაიცემა.

ორგანიზაციები, აპლიკაციები და ინდივიდუალური პირები შესაძლებელია დაუკავშირდეს ერთმანეთს გაცილების უფრო ეფექტურად, თუ ისინი შეთანხმდებიან ინფორმაციის მნიშვნელობის სტრუქტურაზე. XML სპეციალურად შექმნილია იმისათვის, რომ გააადვილოს ასეთი კომუნიკაცია და BizTalk Server კი უზრუნველყოფს მტკიცე საფუძველს, რომელსაც აქვს XML, როგორც სტრატეგიული ხელსაწყო. შეიძლება აღინიშნოს, რომ BizTalk Server-ის დანერგვისათვის XML-ის ცოდნა არაა საჭირო. საჭიროა მცირეოდენი ცოდნა HTML-ის და რელაციური ბაზების, ასევე მცირე მონდომება იმისათვის, რომ გასაგები იყოს პროდუქტის შესაძლებლობები მონაცემთა მარტივი ინტეგრაციისათვის, რესურსებისა და ცოდნის მართვისათვის.

კომპანია მაიკროსოფთმა BizTalk Server შექმნა როგორც მათი საწარმოო აპლიკაცია, რომელიც ინტეგრირებულია .NET სერვერ ოჯახში (www.microsoft.com), რომელიც ქმნის

ხელსაწყოთა და სერვისების კომპლექტს, როგორიცაა მართვა, ადმინისტრაცია, დაგეგმვა და მონიტორინგი. იგი მოიცავს შემდეგს:

- BizTalk Server Orchestration Designer: შექმნის ხელსაწყო რომელიც საშუალებას აძლევს ბიზნესანალიტიკოსებს, დეველოპერებსა და IT პროფესიონალებს, რომ იმოქმედონ ჩვეულებრივ პლატფორმაზე;
- BizTalk Editor: ქმნის და ცვლილებები შეაქვს XML დოკუმენტებზე;
- BizTalk Mapper: გარდაქმნის ერთი XML დოკუმენტი სხვა XML სქემაში. (სქემები არის განსხვავებული გზები, რომლითაც ხდება XML დოკუმენტების სტრუქტურიზაცია და განსაზღვრა იმისათვის, რომ გაადვილდეს მონაცემთა გადაცემა).
- BizTalk Messaging Manger: მინიმალური პროგრამირების საშუალებით შესაძლებელს ხდის ინფრომაციისა და აპლიკაციების გადაცემას.
- BizTalk Framework: მისი საშუალებით ხდება მიღებული და გაცემული მონაცემების მარშრუტიზაცია, მონიტორინგი და ანალიზი.
- BizTalk Administration Tool: მისი საშუალებით ხდება მომხმარებლის სპეციფიკური ინფორმაციის ენდ-ტო-ენდ ინტეგრაცია და ადვილი ონლაინ ანალიტიკური პროცესინგი.

Pipeline Editor: ახდენს დოკუმენტის პროცესებზე დაკვირვებიას დასაწყისიდან მის დასრულებამდე.

BizTalk Server-ს აქვს სტანდარტული ინტერნეტ ტექნოლოგიების მხარდაჭერა, მაგალითად EDI (ელექტრონულ მონაცემთა ურთიერთმოქმედება), მრავალი გადაცემები და პროტოკოლები (HTTP, SMTP), ჩვეულებრივი და კერძო ფაილების ფორმატები. BizTalk Server-ის გამოყენებით შესაძლებელია:

მონაცემთა გაცვლის გამარტივება: (XML-ის არცოდნის მიუხედავად) შესაძლებელია შთამბეჭდავი დოკუმენტების შექმნა და მათი მოდიფიკაცია ნებისმიერი ტექსტური რედაქტორის გამოყენებით;

- დოკუმენტების გადათარგმნა, გაფილტვრა და მარშრუტიზაცია მისი შემცველობის გათვალისწინებით;
- ონლაინ ტრანზაქციების ადვილად გამოყენება, ხმოვანი და ვიდეო ნაკადების გადაცემის საშუალებით;
- მბეჭდავი მომხმარებლების ინტერფეისის ინტეგრაცია ვებ ბრაუზერებში და მონაცემთა ბაზების საშუალებით შესყიდვებისა და შეკვეთის პროცესების სისტემების განვითარება;
- აპლიკაციების შექმნა, რომელთაც შეუძლია ორ ან მეტ ჰეტეროგენულ მონაცემთა ბაზასთან შუამავლობა და ინფორმაციის ჭკვიანურად გამოყენება, რომელიც არის გამიზნული ინდივიდუალური მომხმარებლისათვის;
- წერტილოვანი დაკვირვება მონაცემებზე, კლიენტებს შორის სანდოობის ჩამოყალიბებისათვის;

• დაცული კომუნიკაციის აგება, რომელიც უზრუნველყოფს საზოგადო გასაღების დაშიფვრას, ციფრულ ხელმოწერებს და დაშიფვრას მრავალგამიზნული ინტერნეტ შეტყობინების გაფართოებისათვის (S/MIME), გაუმჯობესებულ ბმულებს კლიენტებთან, მიმწოდებლებთან და პარტნიორებთან.

შესაჯამებლად შეგვიძლია ვთქვათ რომ, BizTalk Server არის ტექნოლოგიური გადაწყვეტა, რომელიც აუმჯობესებს ეფექტურობას და ახდენს ბიზნესიდეებისა და სტრატეგიების განახლებას. იგი აბლიერებს ელექტრონული საწარმოების თვისებებს, ოპერაციული სისტემისა და პროგრამირების ენის დამოუკიდებლობას.

9.3. კორპორაციათა ბიზნესპროცესების ინტეგრაციის ამოცანა სერვისორიენტირებული სისტემების ასაგებად

ამოცანა მდგომარეობს ისეთი სისტემის შექმნაში, რომელიც უზრუნველყოფს არსებული სისტემების ინტეგრაციას, როგორც სხვადასხვა ორაგნიზაციას შორის, ასევე ორგანიზაციის შიგნით, რომელიც უზრუნველყოფს სხვადასხვა ფორმატის მონაცემის მიღება-გადაცემა-დამუშავებას, მაღალ წარმადობას და საიმედოობას. აპლიკაციების ინტეგრაციას ძირითადად აქვს სამი დანიშნულება:

• მონაცემთა ინტეგრაცია, რაც უზრუნველყოფს მონაცემების იდენტურობას სისტემებს შორის;

• აპლიკაციათა მიმწოდებლებზე დამოუკიდებლობა. უზრუნველყოფს, რომ აპლიკაციის ცვლილების შემთხვევაში, ბიზნესპროცესი და ბიზნესწესების ხელახალი შექმნა არ იყოს საჭირო;

• ფასადის/ინტერფეისის შექმნა, რაც უზრუნველყოფს აპლიკაციებთან ერთიერთობის ერთიანი ინტერფესის შექმნას, რომელიც საშუალებას იძლევა აპლიკაციებთან კომუნიკაცია შესრულდეს მათი შიგა სტრუქტურების შესწავლის გარეშე.

წიგნის ამ ნაწილის მიზანია კორპორაციული და ინტერკორპორაციული ბიზნესპროცესების მართვის ვებაპლიკაციების დაპროექტება და რეალიზაცია ელექტრონული სისტემების ინტეგრაციის შესაძლებლობით და სერვისორიენტირებული არქიტექტურით.

მიზნის მისაღწევად ნაშრომში განიხილება შემდეგი ძირითადი ამოცანები:

– არსებული თანამედროვე ინტეგრაციის სისტემების ანალიზი და შესაბამისი ინფორმაციული ტექნოლოგიების კლასიფიკაცია, ობიექტ-, პროცეს- და სერვის-ორიენტირებული დაპროექტების პრინციპებით;

– შიგაკორპორაციული და კორპორაციათაშორისი ბიზნესპროცესების ტრადიციული და სერვისორიენტირებული მოდელების აგება სისტემური ანალიზის საფუძველზე, BPMN და UML ტექნოლოგიების ბაზაზე. მათი შედარებითი ანალიზი;

– ვებაპლიკაციების დასაპროექტებლად ორგანიზაციის სერვისული ბიზნეს-პროცესების ობიექტორიენტირებული მოდელირება კლასების, ობიექტებისა და

ვებმეთოდების ფორმალიზაციის საფუძველზე, პოლიმორფიზმის, მემკვიდრეობითობისა და ინტერფეისული თვისებების გათვალისწინებით;

– სერვისორიენტირებული კორპორაციული ვებაპლიკაციების მონაცემთა განაწილებული ბაზების სტრუქტურების დასაპროექტებლად ობიექტროლური მოდელების (ORM) აგება და კვლევა რევერსიული CASE ტექნოლოგიების გამოყენებით;

– სერვისორიენტირებული არქიტექტურის კორპორაციული სისტემის ბიზნეს-პროცესების უნიფიცირებული მოდელების ასახვის (BPMN -> Activity-D -> PetNet) ალგორითმების შემუშავება სტოქასტური, დროითი პეტრის ქსელის გრაფებში, მათი პროცესების შესრულების ეფექტურობის შეფასებისა და სერვისული უზრუნველყოფის შემდგომი სრულყოფის გადაწყვეტილების მიღების მიზნით;

– სერვისორიენტირებული არქიტექტურის ინტერკორპორაციული სისტემის ინფორმაციული ნაკადების გაცვლის სერვისული ბიზნესპროცესების იმიტაციური მოდელის აგება და მისი ფუნქციონირების დროითი მახასიათებლების კვლევა ფერადი პეტრის ქსელების (CPN) გამოყენებით;

– პროექტის შედეგების საფუძველზე ესპერიმენტული პროგრამული სისტემის რეალიზაცია Ms Visual Studio .NET პლატფორმაზე, ASP.NET, ADO.NET, MsSQL_Server, C#.NET, Natural ORM Architect და BizTalk Server პროგრამული პაკეტების გამოყენებით.

9.4. ბიზნესპროცესების რეალიზაციის ინსტრუმენტული საშუალებანი: JavaNetBeans, BPEL

ვებდანართების სარეალიზაციოდ, დღესდღეობით აქტიურ გამოყენებაშია Java (Java NetBeans) და Microsoft .Net პლატფორმები.

Java NetBeans სისტემა Sun Microsoft Systems კორპორაციისა და NetBeans გაერთიანების მიერ შექმნილი Java ტექნოლოგიის ავტომატიზებული სისტემების დამუშავების ინტეგრირებული გარემოა (IDE), რომელიც წარმოადგენს მრავალფუნქციური დანართების ერთობილიობას და უზრუნველყოფს Java Platform Standard Edition (Java SE), Java Platform Enterprise Edition (Java EE) და Java Platform Micro Edition (Java ME) პლატფორმების კომპლექსურ მხარდაჭერას [15,33,34].

Java NetBeans შეიცავს პროგრამული ინსტრუმენტების ფართო სპექტრს, მათ შორის აღსანიშნავია ვიზუალური პროგრამირების, სერვისორიენტირებული არქიტექტურის დანართების (XML, BPEL), პირდაპირი და რევერსიული დაპროექტებისთვის (BPD, BPMN, UML) მოდელური არქიტექტურის (Model-driven architecture MDA) ინსტრუმენტული საშუალებები.

პროგრამული ტექნოლოგიების მწარმოებელი თანამედროვე, წამყვანი კომპანიები აქტიურად უჭერენ მხარს სერვის-ორიენტირებული არქიტექურის, ვებსერვისული ინტერფეისებისა BPMN სტანდარტისა და BPEL ენის გამოყენებას. ამ თვალსაზრისით, Java NetBeans სისტემა ერთ-ერთი მოქნილი ინსტრუმენტია.

ბიზნესპროცესების შესრულების ენა, BPMN სტანდარტის მიხედვით აგებული მოდელების პროგრამული კოდში გენერაციის საშუალებას იძლევა. იგი გამოიყენება, როგორც ვებსერვისული ფორმით რეალიზებული სხვადახვა ბიზნესპროცესის გამოძახების თანამიმდევრობის აღწერისათვის, ისე სისტემის ტექნოლოგიური პროცესების ნაკადებისა (WorkFlow) და მონაცემთა ნაკადების (DataFlow) ლოგიკური სინთეზისა და კოორდინაციისათვის [34].

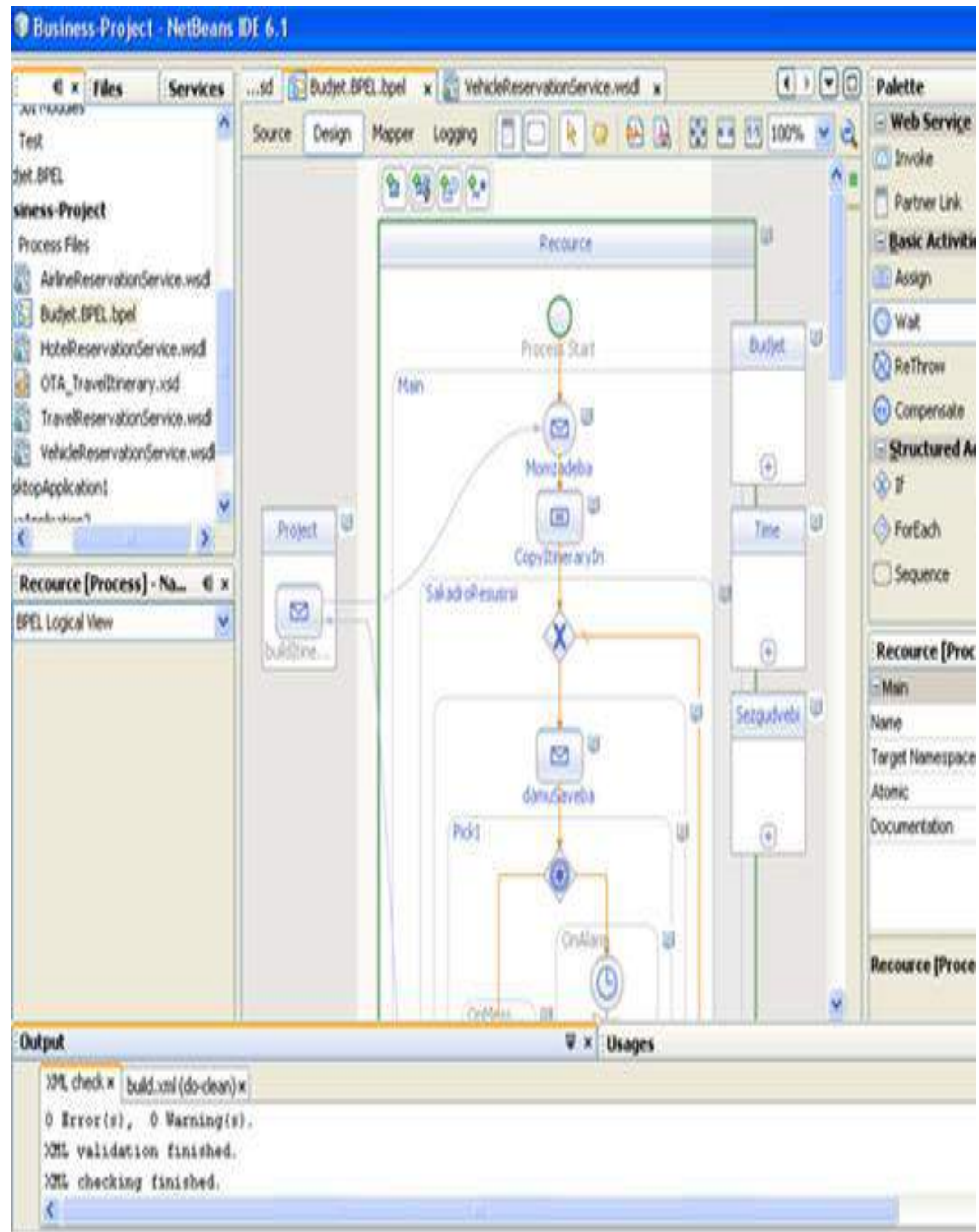
ტექნიკური გამოყენების თვალსაზრისით, იგი განსაზღვრავს როგორ მოხდეს XML შეტყობინების გაგზავნა მოშორებულ სერვისებთან, როგორ განხორციელდეს XML მონაცემთა სტრუქტურის მართვა და მოშორებული სერვისიდან XML შეტყობინებათა ასინქრონულად მიღება (ნახ.9.4).

Microsoft .Net პლატფორმა. ვებსერვისებისა და ვებ-დანართების შექმნისა და განთავსების მოქნილი საშუალებაა Microsoft .NET Framework - პლატფორმა, რომელიც ბაზირებულია მაღალმწარმოებული, სხვადასვა პროგრამული ენის გამოყენების სტანდარტზე. მისი ერთ-ერთი ღირებულებაა არსებული ვებ-დანართებისა და ვებ-სერვისების ინტეგრაციის შესაძლებლობა ახალ პროექტში. .NET Framework შედგება სამი ძირითადი ნაწილისაგან - საერთო ენობრივი გამოყენების გარემო (common language runtime), კლასების უნიფიცირებული ბიბლიოთეკა და ASP კომპონენტური ვერსია - ASP.NET [11,12].

ASP.NET (Active Server Pages - აქტიური სერვერული გვერდები) - .NET ტექნოლოგიის ნაწილია, რომელიც გამოიყენება მასშტაბური კლიენტსერვერული ვებდანართების სარეალიზაციოდ. ინტერაქტიული ვებსაიტის ადვილად შესაქმნელად, იგი შეიცავს მზა მართვის ელემენტების სიმრავლეს და შესაძლებლობას იძლევა შეიქმნას დინამიკური HTML გვერდები.

ASP.NET- ის დამუშავების დროს მისაწვდომია .NET-ის ყველა კლასი, სპეციალური კომპონენტები, შექმნილი C# ან სხვა ენებზე, მონაცემთა ბაზები და ა.შ. ფაქტობრივად, ხელთ გვაქვს ის შესაძლებლობა, რომელსაც იყენებს C# დანართის აგებისას. C#- ის გამოყენება ASP.NET- ში ეფექტურს ხდის დანართის შესრულებას.

ASP.NET ფაილი შეიძლება შეიცავდეს ინსტრუქციების დამუშავებას სერვერისთვის, C#, VB.NET, Jscript.NET ან სხვა პროგრამული ენის კოდებს, რომელთა მხარდაჭერა ხდება .NET პლატფორმით, ნებისმიერი ფორმის შინაარსს, რომელიც გენერირდება რესურსის სახით HTML-ით, ASP.NET - ის ჩადგმული სერვერული მართვის ელემენტებს და ა.შ.



ნახ.9.4. Java NetBeans სისტემის BPEL ვიზუალური რედაქტორი

III ნაწილი

პროგრამული ინჟინერია UML და Agile ტექნოლოგიებით

X თავი. უნიფიცირებული მოდელირების ენის კონცეფცია და განვითარება 234

XI თავი. სისტემის მოთხოვნილების განსაზღვრა და ობიექტორიენტირებუ-
ლი ანალიზი 257

XII თავი. ობიექტორიენტირებული დაპროექტება 273

XIII თავი. მართვის საინფორმაციო სისტემების აგება Agile
ტექნოლოგიებით 294

X თავი
უნიფიცირებული მოდელირების ენის კონცეფცია და განვითარება

10.1. პროგრამული ინჟინერია UML ტექნოლოგიის ბაზაზე

მართვის საინფორმაციო სისტემების სრულყოფილი, საიმედო და მოქნილი პროგრამული უზრუნველყოფის (Software Engineering) სწრაფად დაპროექტება, რეალიზაცია, დანერგვა და შემდგომი თანხლება სისტემის დამკვეთ ორგანიზაციაში მეტად მნიშვნელოვანი ამოცანაა. მისი ეფექტურად გადაწყვეტა ზევრადაა დამოკიდებული როგორც საპროექტო-დეველოპმენტის გუნდის შემადგენლობა-გამოცდილებაზე, ისე IT-ინფრასტრუქტურასა და CASE-ინსტრუმენტებზე.

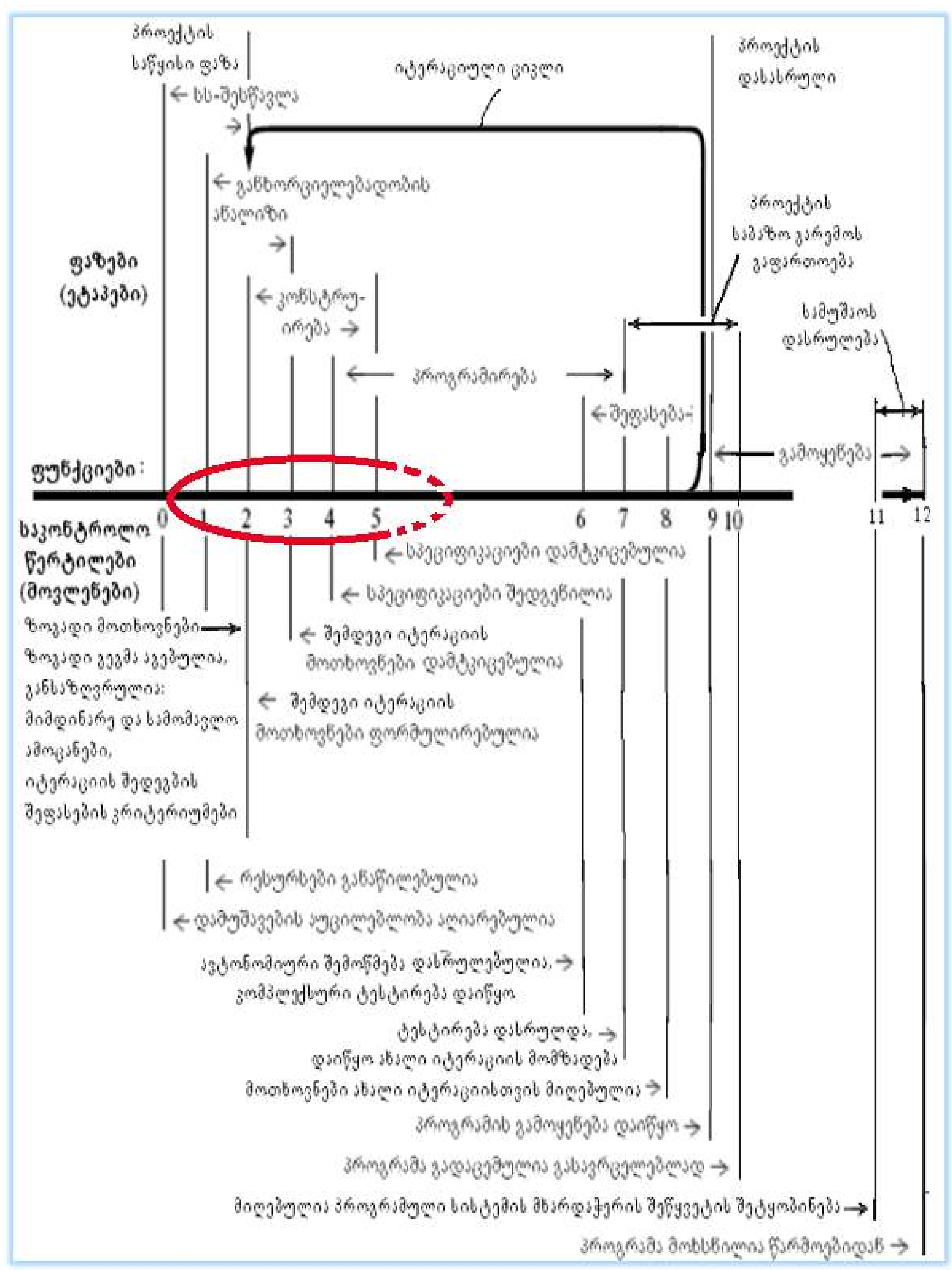
ხშირად შეუძლებელია სრულყოფილი და საიმედო სისტემების აგება „სწრაფად" (მოქნილად – Agile Programing) ისეთი მეთოდებით, როგორიცაა მაგალითად, ექსტრემალური დაპროგრამება [38]. ობიექტორიენტირებული დაპროგრამების მეთოდი, რომელიც უნიფიცირებული მოდელირების ენის (UML) საშუალებით დამკვიდრდა, უნივერსალურია, რომლის გამოყენებით პროგრამის სასიცოცხლო ციკლი მოითხოვს მისი აუცილებელი ეტაპების იტერაციულ განვითარებას [5,39].

10.1 ნახაზზე ნაჩვენებია პროგრამული უზრუნველყოფის დამუშავების სასიცოცხლო ციკლის განტერისეული მოდელის ეტაპები იტერაციული ბიჯებით [40].

პროგრამული სისტემის მენეჯმენტის საკონტროლო (0-12) წერტილებში, ეტაპების მიხედვით ხორციელდება იტერაციული სამუშაოები (დაბრუნება უკანა წერტილებში განმეორებითი პროცედურების ჩასატარებლად), სისტემის ფუნქციონალობის სისრულის დაზუსტების ან გაფართოების მიზნით.

ექსტრემალური პროგრამირების მეთოდის სასიცოცხლო ციკლის მოდელში ძირითადი ყურადღება მახვილდება საპრობლემო ამოცანის სწორად ჩამოყალიბებაში დამკვეთის მიერ ბიზნესანალიტიკოსთან ერთად, ნაკლებად იხარჯება დრო უნივრსალური დიაგრამების აგებასა და საანგარიშო დოკუმენტაციის გაფორმებაზე, და რა თქმა უნდა, ხდება ძირითადი ეტაპების (კონსტრუირება-დაპროგრამება) ფაზათა შერწყმა [38].

აქედან გამომდინარე, პროგრამული სისტემის მენეჯერი, კონკრეტული პროექტის ამოცანებისა და მოთხოვნების შესაბამისად, უნდა განსაზღვრავდეს როგორც პროგრამირების მეთოდის, ეტაპთა ფაზების და იტერაციათა მოთხოვნების შერჩევა-ფორმირებას, ისე სამუშაო გუნდის შემადგენლობას. ამ პროცესში მონაწილე როლებია: დამკვეთი, პროექტის მენეჯერი, ბიზნესპროცესების სპეციალისტი (ბიზნეს-ანალიტიკოსი), სისტემის არქიტექტორი, დეველოპერი-პროგრამისტი, ტესტირების სპეციალისტი და სხვ. პროგრამული სისტემის პროექტის მენეჯერი ახორციელებს ყველა საკონტროლო წერტილის მონიტორინგს.



ნახ.10.1. განტერის მოდელი იტერაციით ობიექტორიენტირებული
პროგრამული პროექტისთვის

დიდი პროექტებისათვის, რომელშიც რესურსები და დროითი ფაქტორები, შედარებით კრიტიკული არაა, ხდება ობიექტ-ორიენტირებული მიდგომის ყველა ეტაპისა და ფაზის გამოყენება შესაბამისი საკონტროლო წერტილების აუცილებელი მონიტორინგით და რეპორტებით. ამ დროს სრული მოცულობით ხორციელდება უნიფიცირებული მოდელირების ენის (UML/2) და შესაბამისი ინსტრუმესნტული საშუალების, მაგალითად, MsVisio ან Enterprise Architect პაკეტის გამოყენება [41,42]. ეს ინსტრუმენტი რეალიზაებულია აგრეთვე Ms Visual Studio.NET -ის ბოლო ვერსიებშიც, რომელიც საშუალებას იძლევა UML დიაგრამების (მაგალითად, Class-D, Sequence_D) გენერირებულ იქნას ავტომატურად C# კოდი [55].

მოდელირების უნიფიცირებული ენა – UML (Unified Modeling Language) შექმნილია, როგორც უნივერსალური მოდელირების ენა ობიექტორიენტირებული პროგრამირების სფეროში და არის სტანდარტული ვიზუალური მოდელირების ენა, რომელიც იძლევა საშუალებას, სისტემა აღიწეროს გრაფიკულად და ტექსტურად.

UML პრაქტიკულად წარმოადგენს ბუჩის მეთოდის (Booch Method), ობიექტის მოდელირების ტექნიკის (Object-modeling technique – OMT) და ობიექტორიენტირებული პროგრამული უზრუნველყოფის ინჟინერიის (Object-oriented software engineering - OOSE) სინთეზსს და გვევლინება როგორც ერთი, საერთო და ფართო გამოყენების მოდელირების ენა [12].

UML არის მსოფლიოში ყველაზე ფართოდ გამოყენებადი უნიფიცირებული მოდელირების ენა. იგი შექმნილია საერთაშორისო ასოციაციის, ობიექტების მართვის ჯგუფის – OMG (Object Management Group) მიერ, რომელიც ქმნის ღია სტანდარტებს ობიექტორიენტირებული აპლიკაციებისათვის ანუ UML ეს არის გრაფიკული ენა, რომელიც გამოიყენება ობიექტ-ორიენტირებული მოდელირების აღწერისათვის პროგრამული უზრუნველყოფის სფეროში და რაც მნიშვნელოვანია, იგი არის „ღია სტანდარტი“, რომელიც ხელმისაწვდომია ყველასათვის [25]

UML აერთიანებს მონაცემთა მოდელირების (entity relationship diagrams) ბიზნესმოდელირების (work flows), ობიექტების და კომპონენტების მოდელირების მეთოდებს. ის გამოიყენება პროგრამული უზრუნველყოფის და მისი ტექნიკური რეალიზაციის მთელი სასიცოცხლო ციკლის განმავლობაში. UML-ის მიზანია იყოს სტანდარტული მოდელირების ენა, რომლის საშუალებითაც შესაძლებელია პარალელური და განაწილებული სისტემის მოდელების შექმნა [26-28].

არსებობს რამდენიმე მიზეზი, თუ რატომ უნდა გამოვიყენოთ ენა მოდელირებისათვის UML:

- უნიფიცირებული ტერმინოლოგიისა და მნიშვნელობათა სტანდარტიზაციის საშუალებით მნიშვნელოვნად გამარტივებული არის ურთიერთობა მოდელირებადი სისტემის სხვადასხვა მხარეს შორის. ეს აადვილებს მოდელების გაცვლას სხვადასხვა

დეპარტამენტსა და კომპანიას შორის, განსაკუთრებით პროექტების გადაგზავნას პროექტზე მომუშავე ჯგუფსა და საპროექტო ჯგუფებს შორის;

– მოდელირებაზე მოთხოვნის გაზრდასთან ერთად ვითარდება UML - ენაც. ვინაიდან UML არის მძლავრი მოდელირების ენა, ჩვენ შეგვიძლია მისი საშუალებით შევქმნათ, როგორც მარტივი სისტემის მოდელები, ასევე კომპლექსური სისტემების დაწვრილებითი მოდელები და თუ UML–ის ფუნქცინური შესაძლებლობები არასაკმარისი იქნება, მაშინ ჩვენ მის გაფართოებას სტერეოტიპების საშუალებით შევძლებთ;

– UML დაფუძნებულია ფართოდ გამოყენებად დიდ მიღწევებზე. იგი მოდელირების არსებული ენების გამოყენებით რეალური პრობლემების გადასაჭრელად შემუშავდა, რაც უზრუნველყოფს მის მოხერხებულობას და პრაქტიკაში გამართულ ფუნქციონირებას. UML დიდ მხარდაჭერას ფლობს.

UML-ის განვითარება. UML პირველად 1997 წელს გამოჩნდა. მას შემდეგ UML1-ის სხვადასხვა ვერსიები გამოდიოდა 2005 წლის ჩათვლით. გაფართოვდა და დაიხვეწა აქტიურობისა და მიმდევრობითობის დიაგრამები. კლასები გაფართოებულია შიგა სტრუქტურებით და პორტებით ე. წ. კომპოზიციური სტრუქტურებით. დაემატა ინფორმაციული ნაკადები და სხვ. მას შემდეგ UML2 - ის სხვადასხვა ვერსიით გამოვიდა: UML2.1-UML2.5, FTF_Beta1 სახელწოდებით, რომელიც ჯერჯერობით ფართოდ არ არის ცნობილი [27].

OMG-მ დღესდღეობით UML2.0 ვერსიის სტანდარტიზაცია დაასრულა. ახალი UML2.0 სპეციფიკაცია UML-ისთვის შეიცავს ისეთ სრულყოფილებებს, სიახლეებს, რომელიც რესტრუქტურიზაციას უკეთებს და ხვეწს ენას, რათა ის უფრო ადვილი გამოსაყენებელი, შესასრულებელი და ასაგები გახადოს. ყველაზე აშკარა ცვლილებები UML1 ვერსიისაგან განსხვავებით, რაც UML2.0 –ს აქვს არის ახალი დიაგრამები: კომპოზიციური სტრუქტურის დიაგრამა (Composite structure diagram), დროითი დიაგრამა (Timing diagram), ურთიერთქმედების მიმოხილვის დიაგრამა (Interactive overview diagram) [27].

რაც შეეხება მთლიანად UML2-ს, UML1-ისგან განსხვავებით აქ შემუშვებულია ახალი დიაგრამები: ობიექტების დიაგრამა (object diagrams), პაკეტების დიაგრამა (package diagrams), სტრუქტურის შემადგენლობის დიაგრამა (composite structure diagrams), ურთიერთქმედების მიმოხილვის დიაგრამა (interaction overview diagrams), დროითი (სონქრონიზაციის) დიაგრამა (timing diagrams), პროფილების დიაგრამა (profile diagrams), ხოლო კოოპერაციის დიაგრამა (collaboration diagrams) გვხვდება კომუნიკაციის დიაგრამის (communication diagrams) სახელით. მოხდა მოღვაწეობის დიაგრამის (activity diagrams) და მიმდევრობის დიაგრამის (sequence diagrams) გაფართოება.

UML2.0-ის ოთხი უმთავრესი დოკუმენტაციაა:

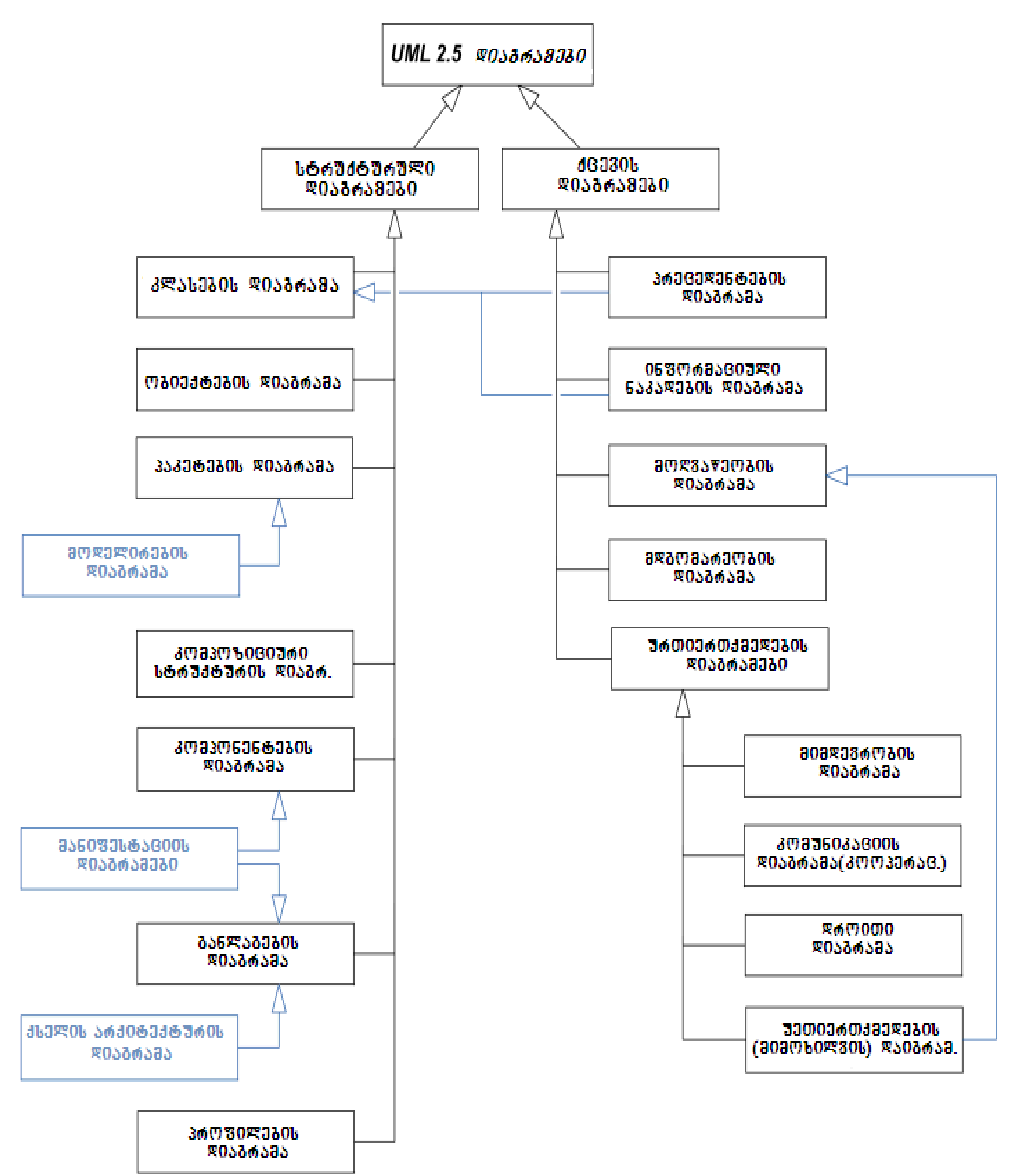
- **სუპერსტრუქტურა.** სუპერსტრუქტურა იყოფა 6 ძირითად დიაგრამად (3 ქცევის და 4 ურთიერთქმედების დიაგრამა) და მათში შემავალი ელემენტებისაგან;
- **ინფრასტრუქტურა.** UML2.0 – ინფრასტრუქტურა განსაზღვრავს საბაზისო კლასებს, რომელიც ქმნის საფუძველს არამარტო UML2.0 – ის სუპერსტრუქტურისათვის, არამედ MOF 2.0 - სთვისაც;
- **დიაგრამების ურთიერთქმედება (ურთიერთჩანაცვლება).** UML2.0-ის ურთიერთქმედება აფართოებს UML-ის მეტამოდელს დამატებითი პაკეტით, გრაფორიენტირებული ინფორმაციით, რომელიც მოდელებს აძლევს საშუალებას ჩაენაცვლონ ერთმანეთს, შენახულ ან აღდგენილ იქნან და წარმოგვიდგნენ საწყის მდგომარეობაში;
- **შეზღუდვების ობიექტების ენა (Object Constraint Language).** UML2.0 - ის შეზღუდვების ობიექტების ენა - ეს არის ენა, რომელიც მოქმედებების და შესრულებადი კოდების დაწერისათვის კი არა, არამედ შეზღუდვებისა და მოთხოვნების განსაზღვრისათვის არის განკუთვნილი.

UML2.0-ის დადებითი მხარეებია: ახალი სტრუქტურა; არქიტექტურული მოდელირების კონსტრუქციები; პორტები, კავშირები და ნაწილები; ახალი UML2.0-დიაგრამები; სტრუქტურის შემადგენლობის დიაგრამა; ქცევის დიაგრამების UML2.0-განახლება; მდგომარეობის დიაგრამის განახლება; ურთიერთქმედების დიაგრამის განახლება; მოღვაწეობის დიაგრამის განახლება; UML - პროფილი [27,28].

ერთ-ერთი ბოლო ვერსიაა UML 2.5. მას აქვს 15 ტიპის დიაგრამა, რომელიც იყოფა 2 კატეგორიად. აქედან 7 დიაგრამა გვაძლევს სტრუქტურულ ინფორმაციას, ხოლო დანარჩენი 8 ქცევის საერთო ტიპებს. მათ შორის 4 სახის დიაგრამა ასახავს ურთიერთქმედების სხვადასხვა ასპექტს. ამ დიაგრამების სტრუქტურა შეიძლება მოცემულია 10.2-ე ნახაზზე.

UML არ აწესებს ელემენტებს რომელიმე ერთი კონკრეტული ტიპის დიაგრამისთვის. ზოგადად, ყველა UML ელემენტი შეიძლება შეგვხვდეს დიაგრამების თითქმის ყველა ტიპში. ეს მოქნილობა ნაწილობრივ შეზღუდულია UML2.0-ში. UML პროფილებმა შეიძლება განსაზღვროს დამატებითი დიაგრამის ტიპები ან გააფართოვოს უკვე არსებული დიაგრამები დამატებითი აღნიშვნებით [27].

საინჟინრო ხაზვის ტრადიციის მიხედვით UML - დიაგრამაში შესაძლებელია გამოყენების შესახებ კომენტარის და შენიშვნის მითითება, ასევე შეზღუდვის ან მიმართულების ჩვენება. დიაგრამების სტრუქტურა ახდენს იმის ხაზგასმას, რაც წამოდგენილი უნდა იყოს სისტემაში, რომლის მოდელირებასაც ვახორციელებთ.



ნახ.10.2. UML2.5-ის დიაგრამების სტრუქტურა

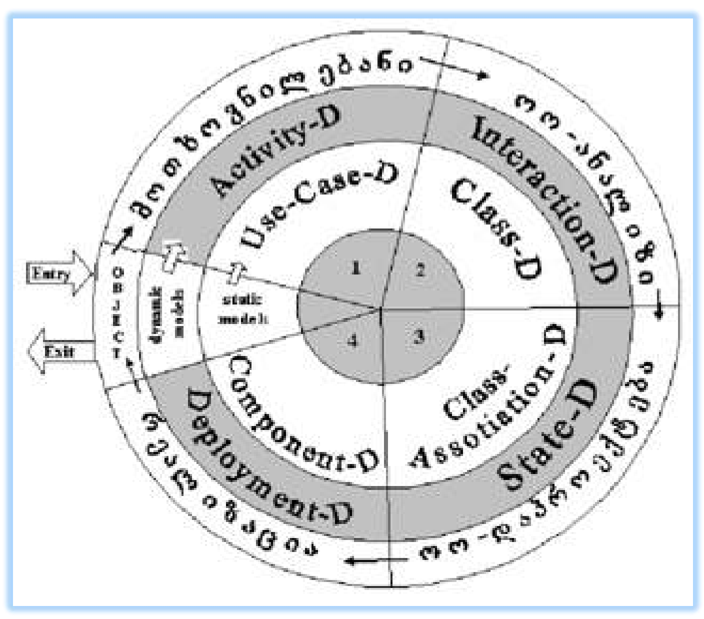
10.2. UML დიაგრამები

UML ტექნოლოგია თეორიულ და პრაქტიკული ინფორმატიკის ბაზაზე ჩამოყალიბდა. იგი განაწილებული მართვის საინფორმაციო სისტემების დაპროექტების მეთოდოლოგიური საფუძველია.

პროგრამული პაკეტების აგების პროცესის სტანდარტიზაცია სამი ძირითადი მიმართულების „გენეტიკური“ მემკვიდრეა: დაპროექტების ავტომატიზაცია, დაპროგრამების ავტომატიზაცია და მონაცემთა ბაზების აგების ავტომატიზაცა [32].

10.3 ნახაზზე ნაჩვეებია UML ტექნოლოგიის კლასიკური მოდელი 4 წყვილი (ეტაპის) დიაგრამით. მათგან 4 სტატიკური მოდელია და 4 - დინამიკური.

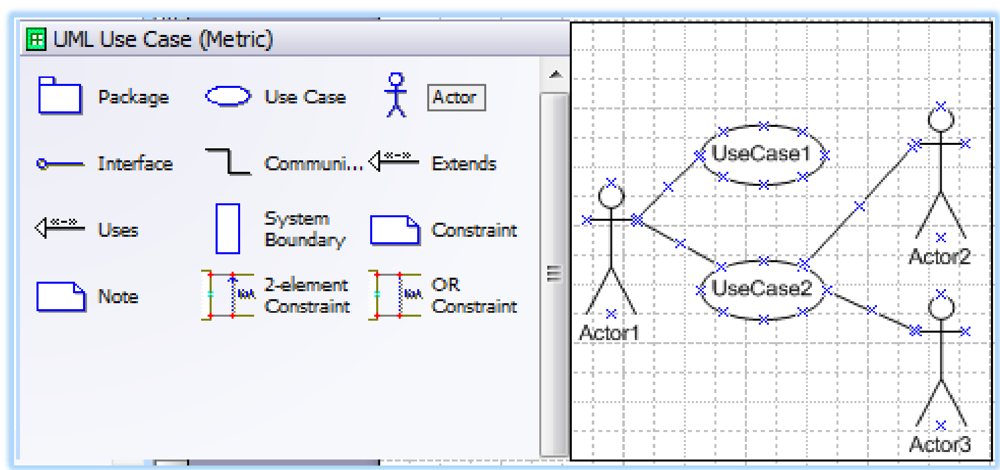
მომდევნო პარაგრაფებში განვიხილავთ თითოეულ მათგანს დეტალურად.



ნახ.10.3.

10.2.1. Use Case დიაგრამა

პირველი დიაგრამა, რომელიც UML ტექნოლოგიით უნდა აიგოს, არის გამოყენებით შემთხვევათა (პრეცედენტების) UseCase დიაგრამა. იგი როლების (Actors) და ფუნქციების (Actions) ურთიერთდაკავშირებული სქემაა (ნახ.10.4). აქ UseCase1 ეკუთვნის მხოლოდ პირველ როლს, ხოლო UseCase2-ის შესასრულებლად ორივე როლი მონაწილეობს.



ნახ.10.4. UseCase დიაგრამის აგების ინტერფეისი

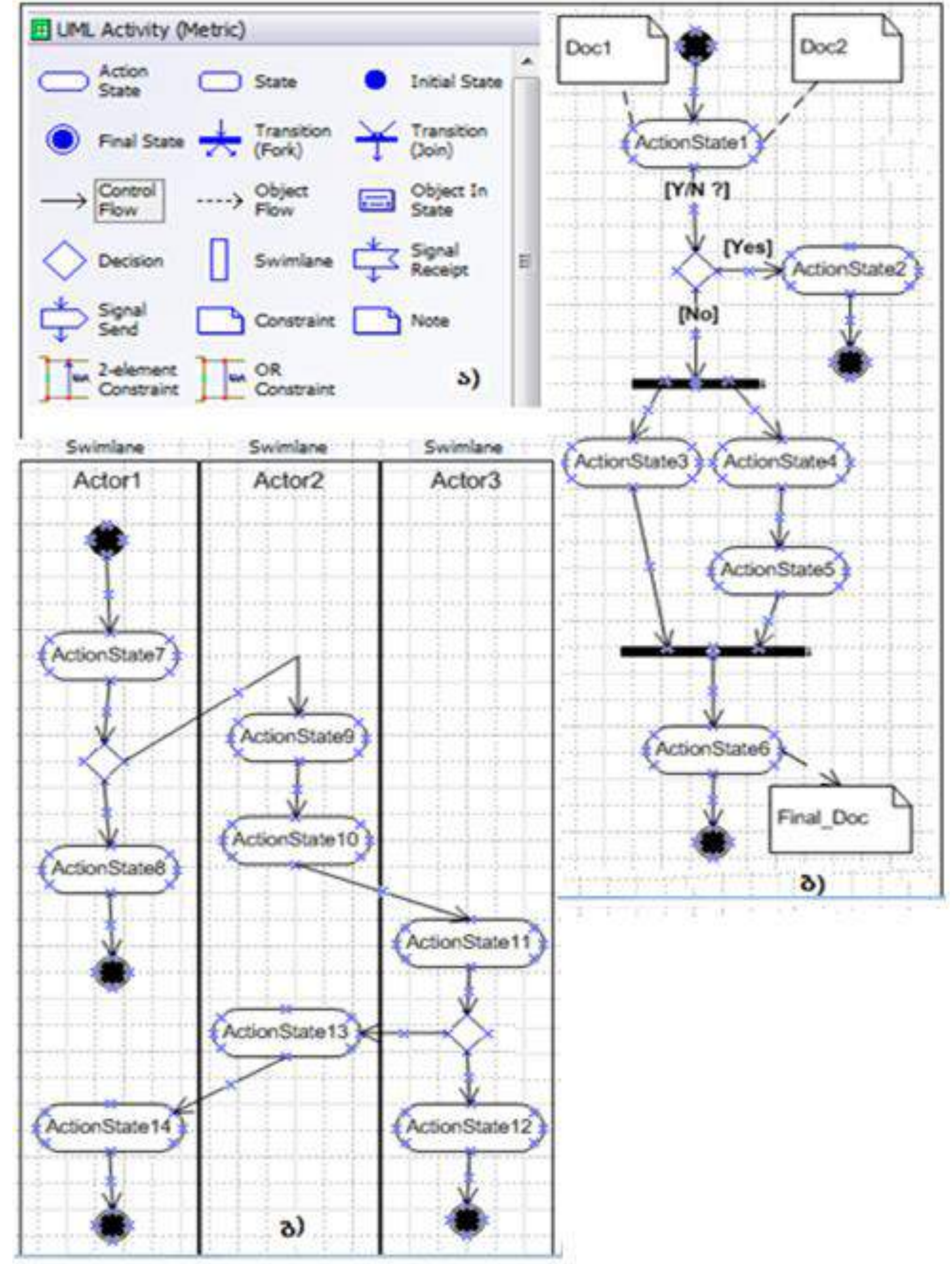
10.2.2. Activity დიაგრამა

ბიზნესპროცესებისა და ბიზნესწესების გაცნობის, ანალიზის და სტრუქტურული ფორმალიზაციის საფუძველზე აიგება აქტიურობის (ქმედებათა) დიაგრამა. იგი კონკრეტული როლის (როლების) კონკრეტული ფუნქციაა, რომელიც შედგება იერარქიულად სივრცესა და დროში დალაგებული მიმდევრობით ან პარალელურად შესასრულებელი სუბქმედებებისგან. აქვს ერთი დასაწყისი და რამდენიმე შესაძლო დასასრული, ბიზნესწესებით განსაზღვრული განშტოების ან შეერთების პროცედურები, საწყისი, შუალედური ან საშედეგო დოკუმენტაცია და ა.შ. საილუსტრაციო მაგალითი მოცემულია 10.5 ნახაზზე.

ქმედებათა დიაგრამა მიეკუთვნება პროცესების აღწერის დინამიკურ მოდელს, იგი ასახავს საკვლევი ობიექტის ქცევას. ასეთი დინამიკური მოდელების პროცესების გამოსაკვლევად გამოიყენება პეტრის ქსელები [19,44].

საბოლოოდ, ამ ორი სახის (UseCase, Activity) დიაგრამათა ერთობლიობის ანალიზის საფუძველზე კეთდება დასკვნები საავტომატიზაციო ობიექტის მართვის სისტემის ფუნქციური და არაფუნქციური მოთხოვნილებების განსაზღვრის შესახებ.

ამავდროულად დგება მომავალი პროგრამული სისტემის შექმნის ტექნიკური დავალება შესაბამის ეკონომიკურ გაანგარიშებებთან ერთად, რომელიც დამკვეთ ორგანიზაციის ხელმძღვანელობასთან კონსულტაციების შემდეგ ორმხრივად მტკიცდება. ამის შემდეგ იწყება ობიექტორიენტირებული ანალიზის ეტაპი, რომელზეც აიგება სისტემის მომხმარებელთა ინტერაქტიული სქემები: მიმდევრობითი და თანამოქმედების დიაგრამათა სახით.



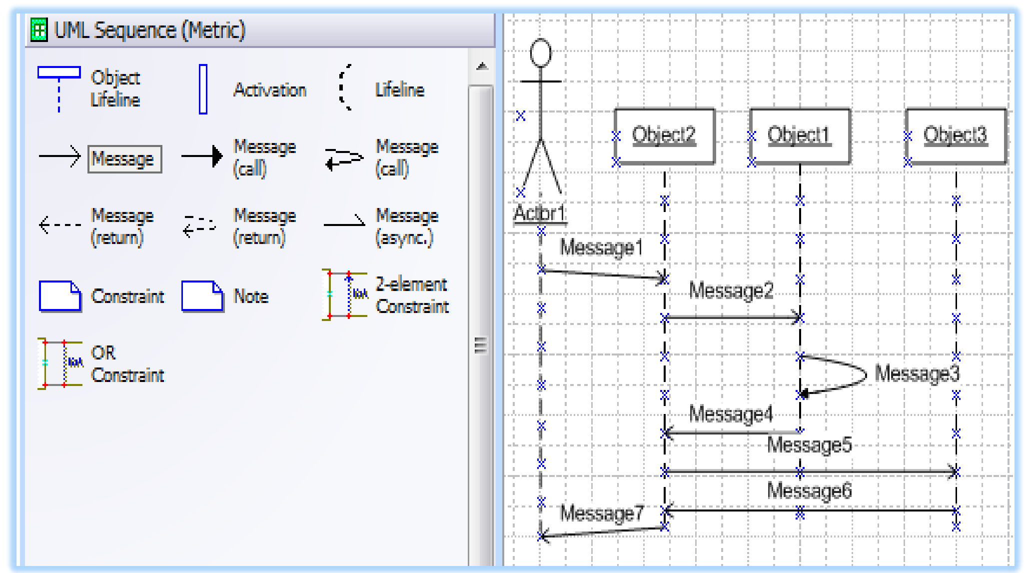
ნახ.10.5. Activity დიაგრამის:

ა) ინსტრუმენტების პანელი; ბ) ქმედებათა სქემა (მარტივი);

გ) ქმედებათა სქემა (მართვის სფეროების ან როლების ბილიკებით)

10.2.3. Sequence და Collaboration დიაგრამები

მიმდევრობითობის დიაგრამა აღწერს საპრობლემო სფეროს კონკრეტული ამოცანის შესრულების სცენარს. აქ ხდება როლის სისტემასთან ურთიერთქმედების ბიზნესპროცესის ქმედებათა და მათი მაინიცირებელ, სინქრონულ ან ასინქრონულ შეტყობინებათა დროში მიმდევრობით განლაგება (ნახ.10.6).

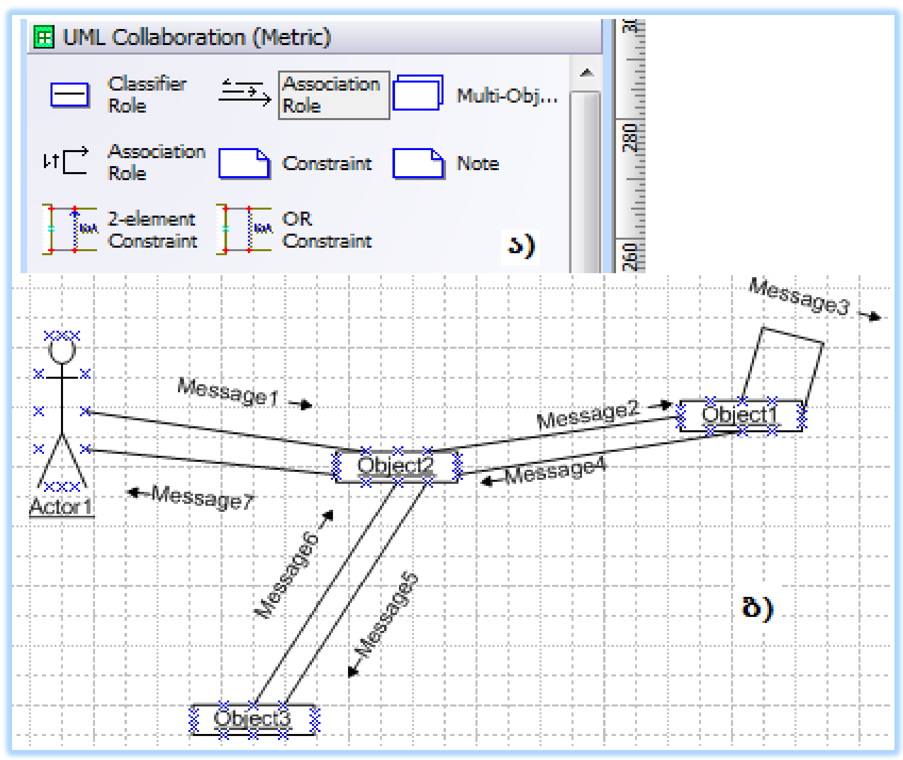


ნახ.10.6. Sequence დიაგრამისაგების ინტერფეისი

10.7 ნახაზზე ნაჩვენებია თანამოქმედების დიაგრამის აგების ინსტრუმენტების პანელი (ა) და თვით დიაგრამა (ბ), რომელიც შესაბამისი მიმდევრობითობის დიაგრამის ტრანსფორმაციით იქნა მიღებული.

აქ შეტყობინებათა და მონაცემთა გაცვლის მიმდევრობა არაა დროში დალაგებული, სამაგიეროდ ჩანს კლასის ობიექტებს შორის კავშირებისა და ინფორმაციული ნაკადების გაცვლის სემანტიკა.

CASE ტექნოლოგიის მრავალ პროდუქტში (მაგალითად, ფირმა Rational Rose) თანამიმდევრობითობის დიაგრამის აგება ხდება ავტომატიზებულ რეჟიმში. ანუ ჯერ აიგება მიმდევრობითობის დიაგრამა და შემდეგ ინსტრუმენტის რომელიმე სწრაფი ღილაკით (მაგალითად, F5) სისტემა თვითონ ააგებს თანამოქმედების დიაგრამას].

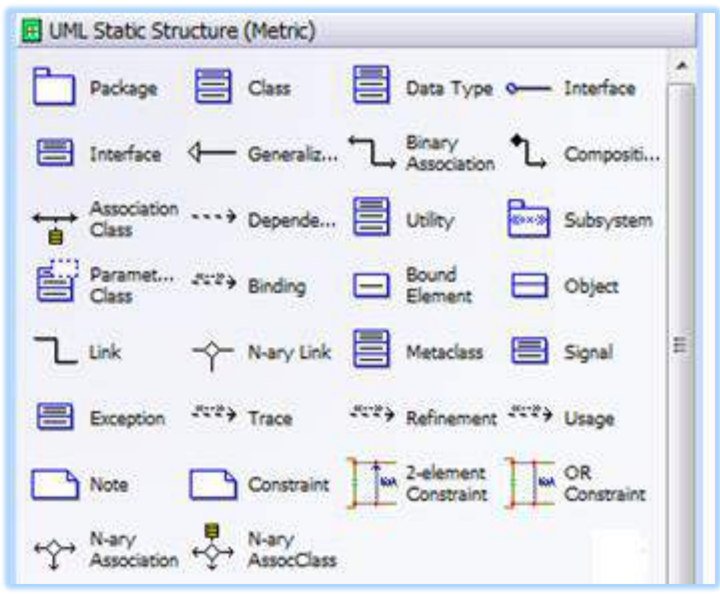


ნახ.10.7. Collaboration დიაგრამის აგების ინტერფეისი (ა)
და სქემის მაგალითი (ბ)

10.2.4. კლასების (Class) დიაგრამა

კლასი ერთგვაროვან ობიექტთა ერთობლიობის სტრუქტურაა. მაგალითად, ყველა მოქალაქე (ზოგადად Person), სტუდენტები, ლექტორები, ავტომანქანები, ცხოველები და ა.შ.

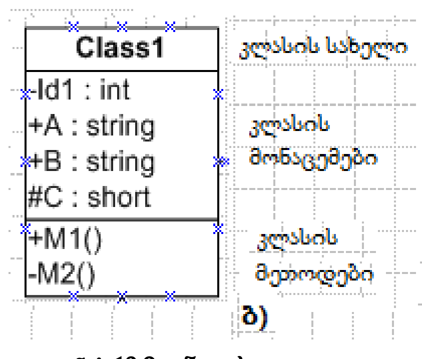
ტერმინი „კლასიფიკაცია“ სწორედ განსახილველი სფეროს ობიექტების სისტემატურ მოწესრიგებას ემსახურება (გენერალიზაცია (იერარქიაში განზოგადება ზევით) და სპეციფიკაცია (იერარქიაში დეტალიზაცია ქვევით). MsVisio-ში კლასებისა და კლასთაშორის კავშირების მოდელირებისათვის გამოიყენება Static Structure ინსტრუმენტების პანელი (ნახ.10.8).



ნახ.10.8

ობიექტორიენტირებული მოდელირებისა და პროგრამირების გაგებით, კლასი არის „დასახელების", „კლასის მონაცემებისა" და „კლასის მეთოდების" ინკაფსულაცია.

მართვის სფეროს შესაბამისი კლასი, ზოგადად ასე უნდა გამოიყურებოდეს (ნახ.10.9). კლასის ატრიბუტებს შეესაბამება მონაცემთა გარკვეული ტიპი (int, float, string ან სხვ.) და „ხილვადობის" ანუ მონაცემთა წვდომის მოდიფიკატორები („-" private, „+" public, „#" protection). ასევეა კლასის მეთოდებისთვისაც.

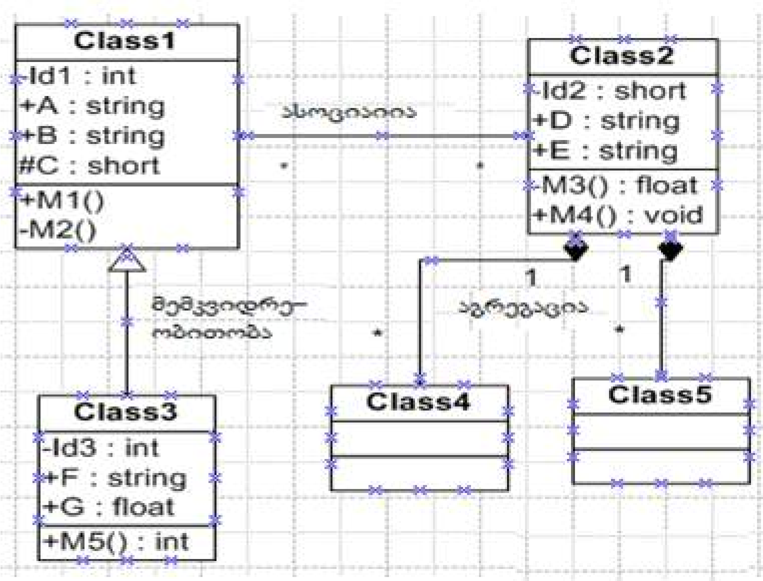


ნახ.10.9. ინკაფსულაცია

კლასის მეთოდები (ან ფუნქციები) ის პროგრამული მოდულებია, რომლებიც ამუშავებს ამ კლასის მონაცემებს. მათი ინიციალიზაცია ხდება გარედან შემოსული შეტყობინების საფუძველზე.

10.2.5. Class-Assotiation დიაგრამა

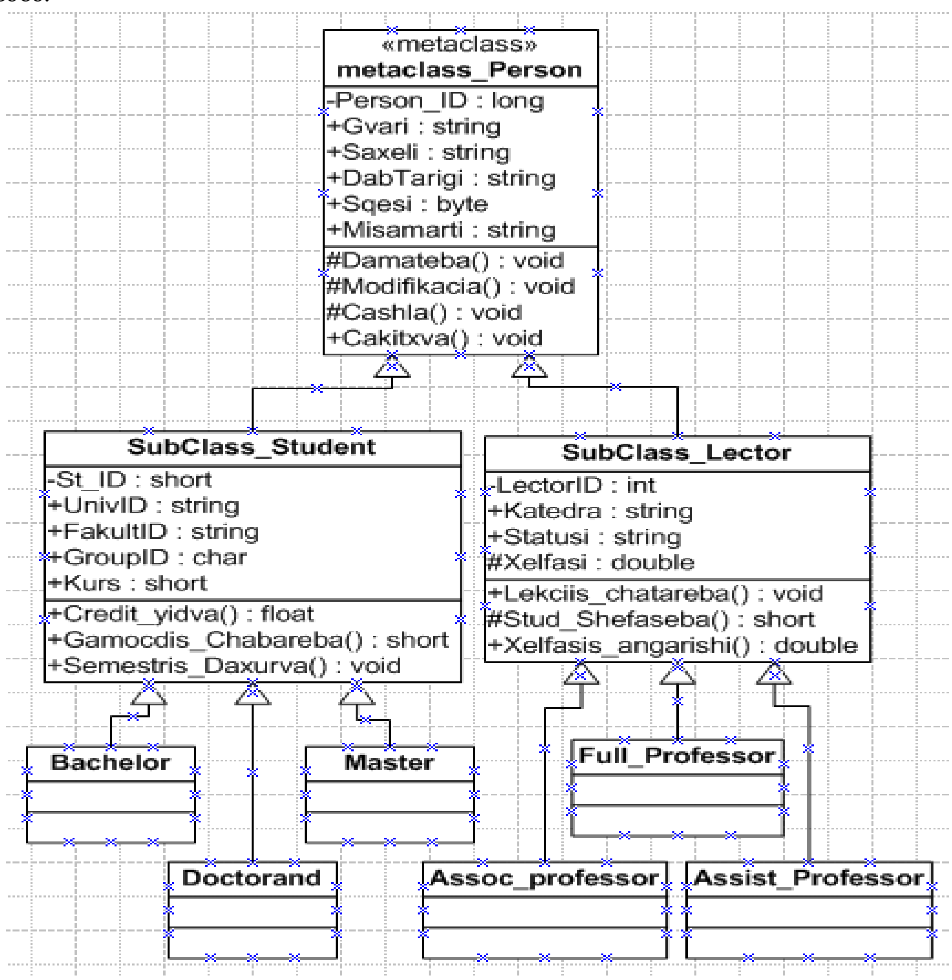
კლასთაშორისი კავშირები შეიძლება იყოს: მემკვიდრეობითი, აგრეგატული, რელაციური და ასოციაციური (ნახ.10.10):



ნახ.10.10. კლასთაშორისი კავშირების დიაგრამა

- მემკვიდრეობითი (Generalization) ასახავს „გენეტიკურ“, განზოგადოებულ კავშირებს კლასებს შორის. ასეთ დროს ერთი კლასი („შვილი“) მთლიანად იღებს მეორე კლასის („მშობელი“) ყველა ატრიბუტს, მეთოდს და კავშირს;
- აგრეგირებული (Aggregation) ნიშნავს კავშირს „მთელი“–„ნაწილი“. მაგალითად, „ავტომობილი“ – „ძარა, ძრავი, საბურავები და სხვ.“;
- ასოციაციური (Assotiation) ნიშნავს სემენტიკურ კავშირს კლასებს შორის. ის შეიძლება გამოისახოს ერთ- ან ორმიმართუ-ლებიანი (იგივეა, რაც უისრო) ხაზით. ისარი გვიჩვენებს შეტყობინების გადაცემის მიმართულებას. ასოციაციური კავშირის რეალიზება ხდება ერთ კლასში დამატებით მეორე კლასის ატრიბურტის ჩასმით. ეს ჰგავს პირველადი (PrimaryKey) და მეორეული (ForeignKey) გასაღებური ატრიბუტების შეერთებას;
- რელაციური (Dependency) ნიშნავს ერთი კლასის დამოკიდებულებას მეორეზე. იგი ერთმიმართულებიანი წყვეტილი ისრით გამოიხატება. მასში დამატებითი დამაკავშირებელი ატრიბუტები არ გამოიყენება.

10.11 ნახაზზე ილუსტრირებულია კლასთა ასოციაციის დიაგრამა მემკვიდრეობითი კავშირების საფუძველზე. ისარი მიმართულია „შვილიდან“ „დედისკენ“, რაც მათ ცალსახა დამოკიდებულებაზე მეტყველებს. „შვილს“ ჰყავს ერთი „დედა“, ხოლო „დედას“ შეიძლება ჰყავდეს რამდენიმე „შვილი“, ამიტომაც ეს არაა ცალსახა.



ნახ.10.11. Class-Assotiation დიაგრამა მემკვიდრეობითი კავშირებით

მშობელი კლასი ლიტარატურაში ზოგჯერ „მეტაკლასად“ (MetaClass) მოიხსენიება, რომელიც შედგება ქვეკლასებისაგან (SubClasses). შეიძლება იერარქიაში ქვეკლასი იყოს მის ქვევით მდგარი კლასისათვის მეტაკლასი. მაგალითად, SubClass_Student არის ქვეკლასი MetaClass_Person კლასისათვის და, ამავდროულად იგი არის მეტაკლასი სამი ქვეკლასისთვის: Bachelor, Master და Doctorand. იგივე შეიძლება ითქვას კლასებისათვის:

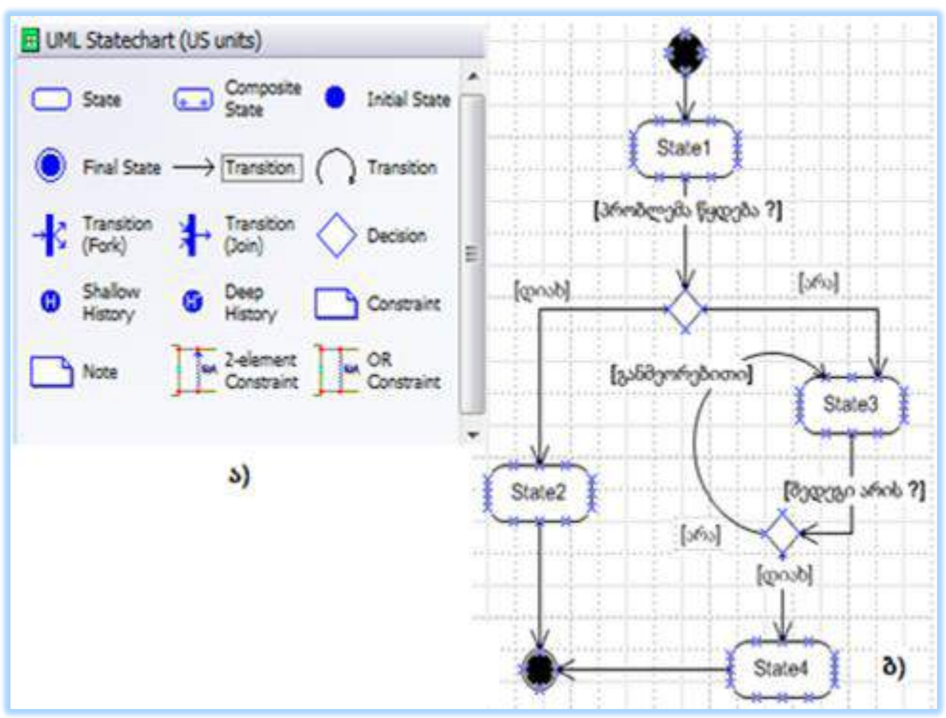
Metaclass_Person <- SubClass_Lector <- {Full_Professor, Assoc_Professor, Assist_Professor},

სადაც როლები ასეა განაწილებული: „მშობელი"-Person, „შვილი"-Lector და „შვილიშვილები" Full_Professor, Assoc_Professor, Assist_Professor.

10.2.6. მდგომარეობათა (Statechart) დიაგრამა

ყოფაქცევის დიაგრამებიდან ადრე განვიხილეთ ქმედებათა (Activity) დიაგრამა და ინტერაქტიული (Sequence, Collaboration) დიაგრამები. არსებობს კიდევ ერთი ასეთი სახის დიაგრამა, *კლასების მდგომარეობათა* დიაგრამა - Statechart-D. იგი აღწერს ქმედებებს, ობიექტთა მდგომარეობებს, მდგომარეობათა გადასვლებს და მოვლენებს.

10.12-ა,ბ ნახაზებზე ნაჩვენებია ინსტრუმენტული პანელისა და მდგომარეობათა დიაგრამის ფრაგმენტი ზოგადი მაგალითისთვის.



ნახ.10.12. Statechart-ის პანელი (ა) და ზოგადი დიაგრამა (ბ)

მისი გამოყენება ყველა კლასისათვის არაა საჭირო. აუცილებელია მხოლოდ მაშინ, როდესაც კლასი შეიძლება იმყოფებოდეს რამდენიმე მდგომარეობაში და თითოეულ მათგანში მისი ქცევა უნდა იყოს სხვადასხვანაირი.

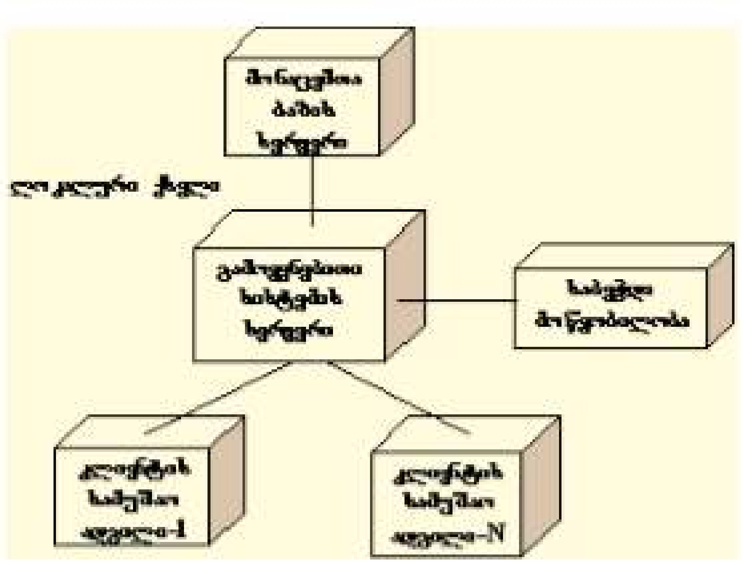
10.2.7. კომპონენტების (Components) დიაგრამა

პროგრამული რეალიზაციის ეტაპზე აიგება კომპონენტების დიაგრამა (Component-D), რომელშიც იგულისხმება პროგრამული კოდების (მაგალითად: .cs, .cpp, .h, .dll, .exe და ა.შ.) დამუშავება (ნახ.10.13).

ნახ.10.13. კომპონენტების დიაგრამა

10.2.8. განთავსების (Deployment) დიაგრამა

განაწილებული სისტემის რეალიზაციის ბოლო ეტაპზე აიგება განთავსების დიაგრამა (Deployment-D), რომელიც აღწერს კომპონენტების განაწილებას „კლიენტ-სერვერის" ქსელში (ნახ.10.14).

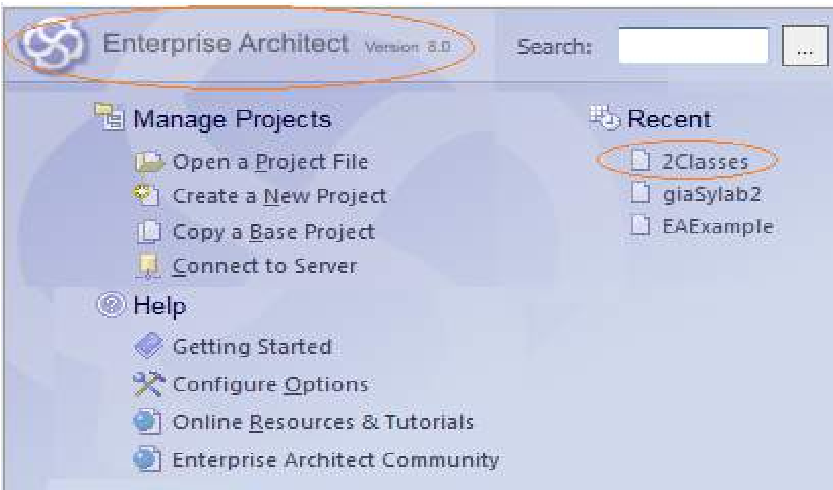


ნახ.10.14. განთავსების დიაგრამა

10.3. კლასების დიაგრამიდან პროგრამული კოდის გენერაცია

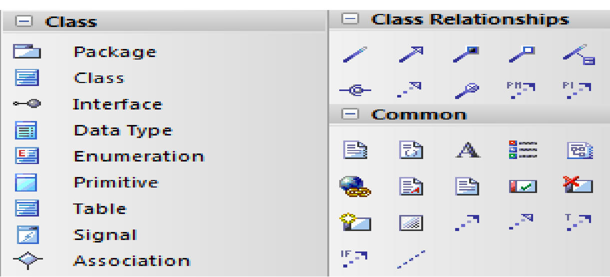
თანამედროვე CASE -ტექნოლოგიები, რომლებიც სისტემების დაპროგრამების ავტომატიზაციაზეა ორიენტირებული, მაგალითად, Rational Rose, Visual Paradigm, Enterprise Architect და მრავალი სხვა [42], ახორციელებს რევერსული დაპროგრამების კონცეფციას. ანუ კლასების დიაგრამიდან შესაძლებელია პროგრამული კოდის გენერაცია და პირიქითაც, კოდიდან ავტომატურად აიგება გრაფიკული დიაგრამა.

MsVisio არ მიეკუთვნება ასეთი სიმძლავრის ინსტრუმენტს. მისი საშუალებით დიალოგურ რეჟიმში იხაზება დიაგრამები (UML-ის სტანდარტულ აღნიშვნათა საერთაშორისო ნორმებით), მაგრამ კოდის გენერაცია არაა შესაძლებელი.
Ms Visual Studio .NET Framework- ისთვის შექმნილია ინსტრუმენტები და მათი ინტეგრაციით .NET გარემოში, შესაძლებელია დიაგრამებიდან კოდის გენერაცია. ასეთი პაკეტები ყოველთვის ფასიანია და ძვირადღირებული. აქ განვიხილავთ SparX ფირმის Enterprise Architect პროდუქტის ამ კონკრეტულ ფუნქციას, კლასების დიაგრამიდან კოდის გენერაციის ამოცანას. 10.15 ნახაზზე ნაჩვენებია პაკეტის ამუშავების საწყისი გვერდი.

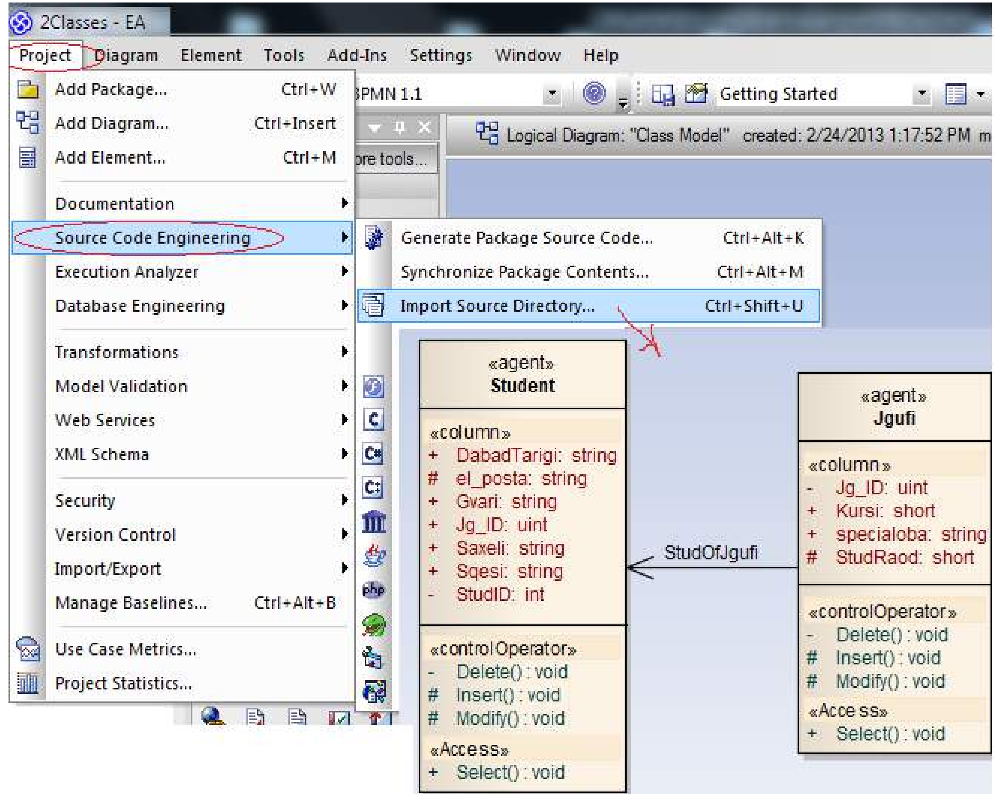


ნახ.10.15. Enterprose Architect საწყისი გვერდი

მაგალითისთვის ვიხილავთ ამ გარემოში ორი კლასის (2Clases მოდელი) აგების და მათი პროგრამულ კოდში გადაყვანის ამოცანას. 10.16 ნახაზზე მოცემულია Enterprise Architect პაკეტის კლასთა დიაგრამის აგების ინსტრუმენტების პანელი. ვირჩევთ Report->Source Code Engineering->Import Source Directory-ის (ნახ.10.17).



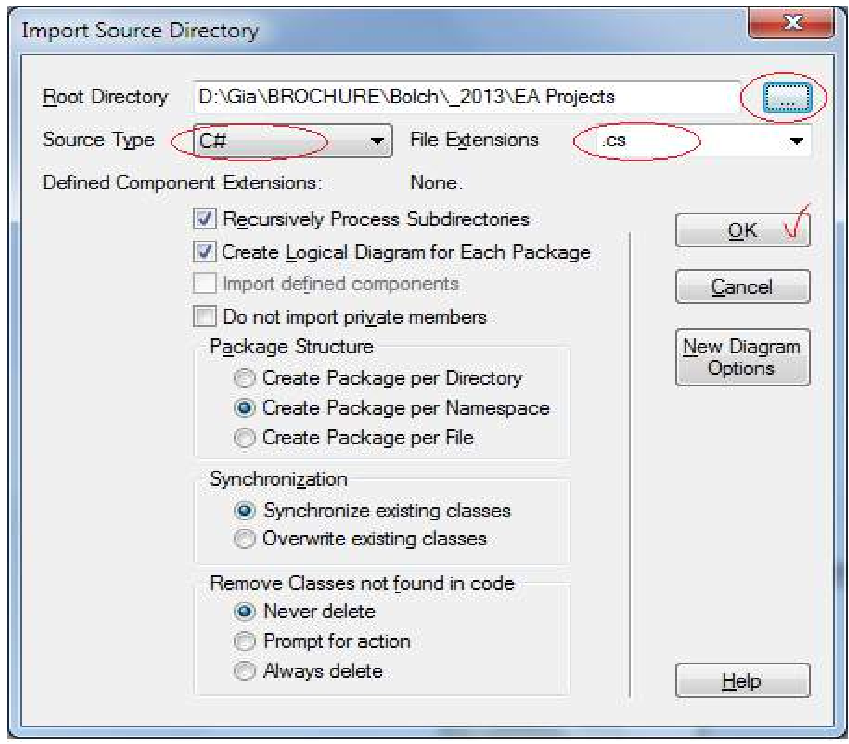
ნახ.10.16



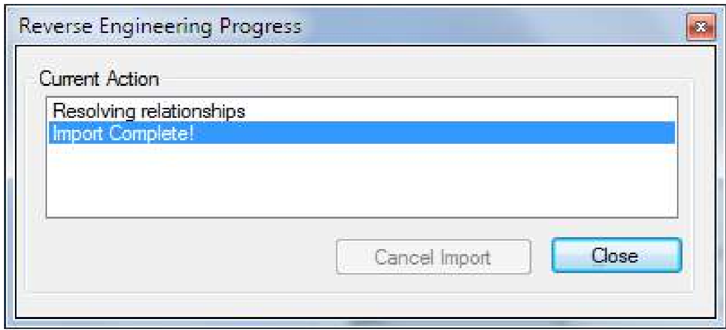
ნახ.10.17. Student და Jgufi კლასების მომზადება „Code Engineering“ პროცესისათვის Enterprise Architect გარემოში

არჩევის შემდეგ გამოვა 10.18 ნახაზზე ნაჩვენები ფანჯარა, რომელშიც უნდა განისაზღვროს ზოგიერთი მნიშვნელოვანი პარამეტრი, მაგალითად, ენა (C#), მომავალი კოდის შესანახი ადგილი (დირექტორია) და ა.შ.

ბოლოს „Ok“ და მივიღებთ 10.19 ნახაზზე მოცემულ შედეგს.

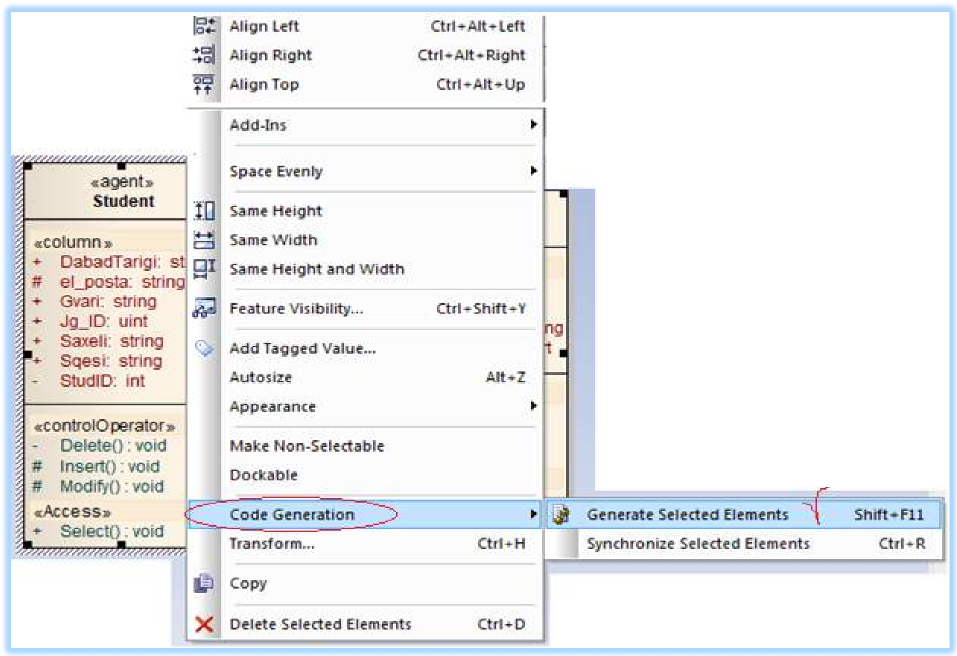


ნახ.10.18. განისაზღვროს C#-კოდის დირექტორია

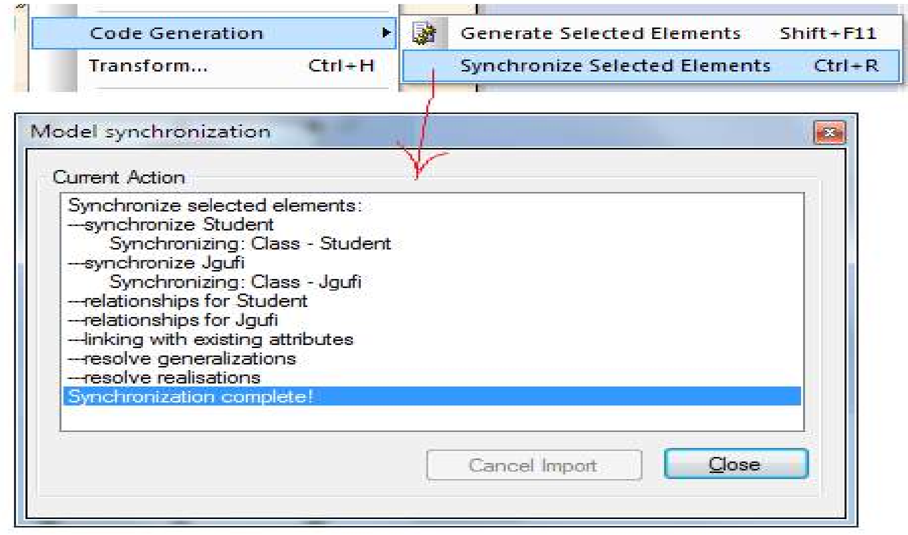


ნახ.10.19. იმპორტი დასრულებულია

ახლა უნდა ჩატარდეს უშუალოდ კოდის გენერაცია წინასწარ მომზადებული (Student-Jgufi) კლასების დიაგრამიდან. ვააქტიურებთ კლასებს და მაუსის მარჯვენა ღილაკით გამოტანილ კონტექსტური მენიუდან ვირჩევთ „Code Generation“-ს (ნახ.10.20-21).

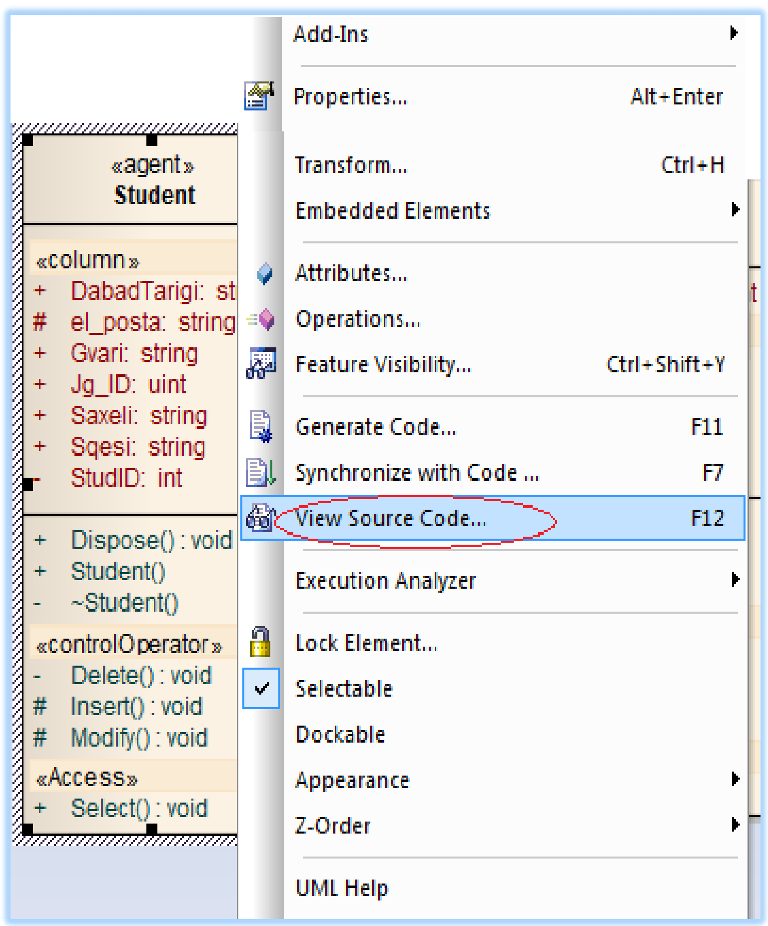


ნახ.10.20. კოდის გენერაციის დაწყება



ნახ.10.21. არჩეული ელემენტის სინქრონიზაციის შედეგი

ბოლო ფაზაზე საჭიროა ეკრანზე გამოვიტანოთ კლასების ბაზაზე გენერირებული კოდის ლისტინგები (ნახ.10.22).



ნახ.10.22. კოდის გამოტანის პუნქტი მენიუში

10.23 ნახაზზე ნაჩვენებია Enterprise Architect გარემოში Student კლასის დიაგრამიდან ავტომატურად გენერირებული C#-კოდის საწყისი ტექსტი.

ფანჯრის მარცხენა ნაწილში მოთავსებულია Student კლასის კაფსულა, თავისი მონაცემებით და მეთოდებით, მათ შორის კონსტრუქტორით და დესტრუქტორით (მეთოდები, რომელთაც აქვს კლასის იდენტური სახელი).

ფანჯრის მარჯვენა ნაწილში სისტემას გამოაქვს პროგრამის ტექსტი, რომელიც შედგება კომენტარული ნაწილის (სტრიქონები 1-7) და კლასის აღწერის ნაწილისაგან (8-31).

2Classes - EA   Student.cs

Student

- Student
  - DabadTarigi
  - Gvari
  - Jg_ID
  - Saxeli
  - Sqesi
  - StudID
  - el_posta
  - Delete()
  - Dispose()
  - Insert()
  - Modify()
  - Select()
  - Student()
  - ~Student()

```
1 ///////////////////////////////////////////////
2 //   Student.cs
3 //   Implementation of the Class Student
4 //   Generated by Enterprise Architect
5 //   Created on:      24-Feb-2013 2:26:10 PM
6 //   Original author: user
7 ///////////////////////////////////////////////
8 public class Student {
9     public string DabadTarigi;
10    protected string el_posta;
11    public string Gvari;
12    public uint Jg_ID;
13    public string Saxeli;
14    public string Sqesi;
15    private int StudID;
16    public Student(){   }  // constructor
17    ~Student(){ }  // destructor
18    public virtual void Dispose(){   }
19    private void Delete(){
20      // . . . code-1
21    }
22    protected void Insert(){
23      // . . . code-2
24    }
25    protected void Modify(){
26     // . . . code-3
27    }
28    public void Select(){
29     // . . . code-4
30    }
31 }//end Student
```

ნახ.10.23. C#-კოდის ლისტინგი Student კლასისათვის

მომდევნო ლისტინგში მოცემულია Jgufi კლასის საწყისი ტექსტი. აქაც, C# კოდის ტექსტი შედგება კომენტარული ნაწილისაგან, რომელშიც ასახულია პროგრამის ზოგადი მახასიათებლები, სახელი, ინსტრუმენტი, შექმნის თარიღი, ავტორი. პროგრამის ტექსტი კლასიკური ფორმატით აღიწერება კლასის მონაცემები ხილვადობის private, public da protection ატრიბუტებით. შემდეგ მოსდევს კონსტრუქტორის public Jgufi(){ } და დესტრუქტორის ~Jgufi(){ } სტრიქონები. Dispose() მეთოდი გამოიყენება პროგრამის შესრულების დამთავრების შემდეგ ოპერაციული სისტემის მიერ გამოყოფილი რესურსების გასათავისუფლებლად.

```
///////////////// ლისტინგი //////////////////////
//  Jgufi.cs
//  Implementation of the Class Jgufi
//  Generated by Enterprise Architect
//  Created on:    24-Feb-2016 2:26:15 PM
//  Original author: user
/////////////////////////////////////////////////////////
public class Jgufi {
            private uint Jg_ID;
            public short Kursi;
            public string specialoba;
            protected short StudRaod;
            public Student m_Student;
            public Jgufi() {   }     // კონსტრუქტორი
            ~Jgufi() {  }    // დესტრუქტორი
            public virtual void Dispose() {  }
            private void Delete(){
            // . . . code-1
            }
            protected void Insert(){
            // . . . code-2
            }
            protected void Modify(){
           // . . . code-3
            }
            public void Select(){
           // . . . code-4
            }
}//end Jgufi
```

ამით ვამთავრებთ უნიფიცირებული მოდელირების ენის და მისი ინსტრუმენ-ტების მოკლე აღწერას.

UML/2 ვერსიაში მოხდა 7 ახალი დიაგრამის დამატება, რაც აიხსნება არსებული 8 კლასიკური დიაგრამის სემანტიკური გაფართოების მიზნით.

ამ დიაგრამებს განვიხილავთ მომდევნო თავებში, სადაც რეალური ობიექტების უნიფიცირებული მოდელირების საკითხებს შევეხებით.

XI თავი
სისტემის მოთხოვნილების განსაზღვრა და ობიექტორიენტირებული ანალიზი

მაიკროსოფთის კორპორაცია თავის Visual Studio.NET პაკეტის ახალ ვერსიებში (2010, 2013, 2015) სულ უფრო აფართოებს პროექტების მოდელირების ფუნქციებს და შესაძლებლობებს, ჩაშენებული აქვს UML-დიაგრამების აგებისა და რევერსიული ინჟინერინგის („კოდი-მოდელი-კოდი“) რეალიზების საშუალებები. ჩვენ წინამდებარე და მომდევნო თავებში სწორედ ამ ტექნოლოგიას გამოვიყენებთ [55, 77].

პროგრამული პაკეტების აგების პროცესის სტანდარტიზაცია სამი ძირითადი მიმართულების „გენეტიკური“ მემკვიდრეა: დაპროექტების ავტომატიზაცია, დაპროგრამების ავტომატიზაცია და მონაცემთა ბაზების აგების ავტომატიზაცია [115].

მართვის კომპიუტერული სისტემების პროგრამული უზრუნველყოფის აგების პროცესების ასეთი სრულფასოვანი ავტომატიზაცია ვიზუალური მოდელირების სახელწოდებით დამკვიდრდა. იგი მოდელების გრაფიკულ წარმოდგენას ეყრდნობა და ფლობს მოქნილ რევერსულ ტექნოლოგიას.

როგორც 10.3 ნახაზიდან ჩანდა, საპრობლემო სფეროს შესწავლა და გამოკვლევა, მისი შესაბამისი კომპიუტერული სისტემის ფუნქციური მოთხოვნილებების დასადგენად, იწყება Use Case და Activity დიაგრამების აგებით. შემდეგ მას მოჰყვება ობიექტორიენტირებული ანალიზის ეტაპი, რომელზეც აიგება ინტერაქტიული მოდელები - Sequence და Collaboration დიაგრამები, რომლებიც აღწერს მომხმარებელთა ინტერფეისებს და მათ მოქმედების სცენარებს მომავალ სისტემაში.

მაგალითებს განვიხილავთ ლოგისტიკის სფეროდან, კერძოდ, მულტიმოდალური გადაზიდვების მართვის სისტემისთვის [81-83].

11.1. ინფორმაციული რესურსები და ბიზნესპროცესები

მულტიმოდალური გადაზიდვის მართვის სისტემის ბიზნესპროცესები დაკავშირებულია ტრანსპორტის ორი ან მეტი სახეობის გამოყენებასთან [81,83]. საერთაშორისო თუ ადგილობრივი მასშტაბების გადაზიდვების განხორციელებაში მონაწილეობს რამდენიმე სახეობის ტრანსპორტი, ამიტომაც მასში აუცილებლად გავლენას იქონიებს თითოეული სახეობის, როგორც დადებითი, ისევე უარყოფითი თვისებები.

მულტიმოდალური გადაზიდვების ეტაპების შესრულებისას ყურადღება ექცევა თითოეული სახის სატრანსპორტო საშუალებებს. მაგალითად, საზღვაო პორტის დინამიკურ ობიექტებს მიეკუთვნება გემები (სამგზავრო, სატვირთო და შერეული), ვაგონები (დახურული, ღია, სპეციალური), ამწეები (საპორტო, ხიდკაბელური, მუხლუხა, საავტომობილო, სარკინიგზო), გადასატვირთი მანქანები (საავტომობილო, ელექტრული და სხვ.), ტვირთები (ნაყარი, საცალო, მშრალი, თხევადი და სხვ.), მუშა ბრიგადები და სხვ.

აღნიშნული ობიექტები ზემოქმედებს გადაზიდვის პროცესზე და შესაბამისად, დაგეგმვისა თუ განხორციელების ეტაპზე საჭიროა მათი გათვალისწინება. ტრანსპორტირების ჯაჭვში ობიექტის პორტი არასწორმა დაგეგმვამ ან განხორციელებამ შესაძლოა გამოიწვიოს ტვირთის შეყოვნება, ჯარიმა, რომელიც პირდაპირ ზიანს აყენებს გადაზიდვის პროცესის ეფექტურობას.

შესაბამისად, მომავალი კომპიუტერული მხარდამჭერი სისტემა უნდა დაეხმაროს ტრანსპორტირების ორგანიზატორს სწორედ ისეთი ამოცანების გადაჭრაში, როგორიცაა:

- გემებისა და სარკინიგზო ვაგონების მოცდენის დროის მაქსიმალური შემცირება;
- დატვირთვა-დაცლის მექანიზმების მაქსიმალური გამოყენება (ამწეები, სპეცმანქანები და სხვ.);
- პორტის სატრანზიტო დროის მაქსიმალურად ეფექტურად მართვა.

ზოგადად კი ტვირთის, რაც შეიძლება სწრაფად და იაფად მიწოდება გადაზიდვის ჯაჭვით გათვალისწინებული მომდევნო სატრანსპორტო საშუალებისათვის. ქვემოთ მოცემული გვაქვს მულტიმოდალური გადაზიდვების პროცესის ინფრასტრუქტურის ძირითადი ობიექტებისა და მათი თვისებების (ატრიბუტების) სემანტიკური აღწერა, რაც მომავალში გამოყენებულ იქნება ავტომატიზებული სისტემის მონაცემთა ბაზების ასაგებად.

კლიენტი – იდენტიფიკატორი, დასახელება/ვინაობა, იურიდიული/ფიზიკური პირი, მისამართი, ტელეფონი, ელ_მისამართი და სხვ.;

ტვირთი – იდენტიფიკატორი, ტიპი, მდგომარეობა, შეფუთვის ტიპი, ერთეულის ზომები (სიგრძე, სიგანე, სიმაღლე), ერთეულის მოცულობა, ჯამური მოცულობა, ერთეულის წონა, ერთეულის რაოდენობა, ჯამური წონა, უსაფრთხოობა, საბაჟო კოდი, გამგზავნი, მიმღები, გადაზიდვის ხელშეკრულების იდენტიფიკატორი და სხვ.;

მიმწოდებელი – იდენტიფიკატორი, დასახელება, იურიდიული/ფიზიკური პირი, მისამართი, ტელეფონი, ელ_მისამართი, ფაქსი, ტრანსპორტის სახე და სხვ.;

გემი - იდენტიფიკატორი, ტიპი, კრანით/უკრანო, მდგომარეობა, სასაწყობო ლიმიტი, ტვირთამწეობა, ტვირთმოცულობა, ადგილმდებარეობა, მიმწოდებლის იდენტიფიკატორი და სხვ.;

თვითმფრინავი – იდენტიფიკატორი, ტიპი, მდგომარეობა, ტვირთმოცულობა, გადასაზიდი ერთეულის დასაშვები ზომები (სიგრძე, სიგანე, სიმაღლე), მიმწოდებლის იდენტიფიკატორი, ადგილმდებარეობა და სხვ.;

ავტოტრანსპორტი – იდენტიფიკატორი, ტიპი, მდგომარეობა, ტვირთმოცულობა, გადასაზიდი ერთეულის დასაშვები ზომები (სიგრძე, სიგანე, სიმაღლე), გადასაზიდი ერთეულის დასაშვები წონა, მაქსიმალური დატვირთვა, ადგილმდებარეობა, მიმწოდებლის იდენტიფიკატორი, და სხვ.;

სარკინიგზო სატვირთო ვაგონი – იდენტიფიკატორი, ტიპი, ტვირთამწეობა, მოცულობა, დასაშვები დატვირთვა, ადგილმდებარეობა, მიმწოდებლის იდენტიფი-კატორი, მდგომარეობა და სხვ.;

საწყობი – იდენტიფიკატორი, სახე, ფართობი, სართული, დაკავებულობის პროცენტი, დასაშვები დატვირთვა, ადგილმდებარეობა, მისამართი, მიკუთვნება რაიონზე და სხვ.;

გადაზიდვის ხელშეკრულება კლიენტთან - იდენტიფიკატორი, საწყისი მდებარეობა, თარიღი_1, საბოლოო მდებარეობა, თარიღი_2, გადაზიდვის ღირებულება, გადახდილი თანხა, გადახდის_თარიღი, მდგომარეობა და სხვ.;

გადაზიდვის ხელშეკრულება ტრანსპორტის მიმწოდებელთან – იდენტიფიკატორი, ტვირთის_იდენტიფიკატორი, მიმწოდებლის_იდენტიფიკატორი, ტვირთის საწყისი მდებარეობა, თარიღი_1, ტვირთის მიტანის მისამართი, თარიღი_2, გადაზიდვის ღირებულება, გადახდილი თანხა, გადახდის_თარიღი, მდგომარეობა და სხვ.;

გადაზიდვის მარშრუტი – იდენტიფიკატორი, საწყისი პოზიცია (ქალაქი/ქვეყანა), გასვლის პუნქტი/ლოკაცია, ტრანზიტული დანიშნულებ(ებ)ის ადგილი, საბოლოო_პოზიცია (ქალაქი/ქვეყანა), შესვლის პუნქტი/ლოკაცია, მანძილი, ტრანზიტის დრო (გეგმიური), ტრანზიტის დრო (ფაქტობრივი) და სხვ.;

გადაზიდვის პირობა – იდენტიფიკატორი, პირობა დატვირთვის ადგილას, პირობა დანიშნულების ადგილას (საერთაშორისო გადაზიდვის პირობები - INCOTERMS) და ა.შ.

შესაძლებელია ობიექტების დამატება და თვისებების გაფართოება კონკრეტული ფუნქციური ამოცანების დამატების შემთხვევაში.

პროდუქციის (ტვირთების) გადაზიდვის ბიზნესპროცესების ორგანიზაციის (ოპერატიული მართვის პროცესების) მოდელირების საკითხი [81]. ბიზნესპროცესის შინაარსი ასეთია:

„პროცესში მონაწილეობს სამი ძირითადი როლი: დამკვეთი, ექსპედიტორი და ტრანსპორტიორი (გადამზიდავი).

დამკვეთი უკავშირდება ექსპედიტორს (ელ-ფოსტა, ტელეფონი, ფაქსი, ვიზიტი) და ათავსებს მოთხოვნას გადაზიდვის პირობების მიწოდებაზე. იმისათვის, რომ ექსპედიტორმა მიღებული მოთხოვნა დაამუშაოს, დამკვეთმა უნდა მიაწოდოს ტვირთის შესახებ ძირითადი დეტალები: ტვირთის სახეობა, დატვირთვის მისამართი, მიტანის მისამართი, შეფუთვის სახეობა, ტვირთის რაოდენობა, შესაბამისი წონები და მოცულობები, სასურველი ვადები და სხვა (საჭიროებისამებრ, გადაზიდვის სპეციფიკიდან გამომდინარე);

ექსპედიტორი ამოწმებს, თუ აქვს უკვე დამუშავებული (არსებული ფასების ბაზაში თუ იძებნება) ყველა საჭირო ფასი. თუ აქვს, მაშინ იგი ამზადებს კოტირების ფაილს

(Quotation) და უგზავნის დამკვეთს. პასუხს ელ-ფოსტით შესაბამისად დამკვეთი განიხილავს წინადადებას და გასცემს დადებით ან უარყოფით პასუხს.თუ ექსპედიტორს არ აქვს მზად ფასები, იგი უკავშირდება ტრანსპორტიორებს და/ან აგენტებს (Carrier/Agent) და აზუსტებს ფასებს.

ტრანსპორტიორი/აგენტი ამზადებს და უგზავნის კოტირებას ექსპედიტორს.

ექსპედიტორი მიღებულ ფასებს ამუშავებს და ელ-ფოსტით კოტირების ფაილს დამკვეთს უგზავნის.

მას შემდეგ, რაც დამკვეთი დადებით პასუხს გასცემს ექსპედიტორს, ხდება შეკვეთის გაფორმება (გადაზიდვის ინიცირების საფუძველი შეიძლება გახდეს უბრალო მიმოწერა ელ-ფოსტაზე, გადაზიდვის დაკვეთის ორდერის გაფორმების გარეშეც).

ექსპედიტორი ათავსებს დაკვეთას და უთანხმდება გადაზიდვის პირობებზე ტრანსპორტიორს/აგენტს, რომელიც თავის მხრივ სატრანსპორტო დოკუმენტაციას ათანხმებს ექსპედიტორთან, რომელსაც შემდგომ უგზავნის (B/L, CMR, AWB, RWB თუ სხვ).

როცა ტვირთი მიუახლოვდება დანიშნულების ადგილს, ტრანსპორტიორი/აგენტი უგზავნის ექსპედიტორს შეტყობინებას ტვირთის ჩამოსვლის თარიღის მითითებით, რათა მან დროულად მოახდინოს დამკვეთის გაფრთხილება.

ექსპედიტორი უგზავნის დამკვეთს ტვირთის სავარაუდო ჩამოსვლის შეტყობინებას, თარიღის მითითებით. ექსპედიტორი ამზადებს და უგზავნის დამკვეთს გადაზიდვის ანგარიშს (ინვოისს).

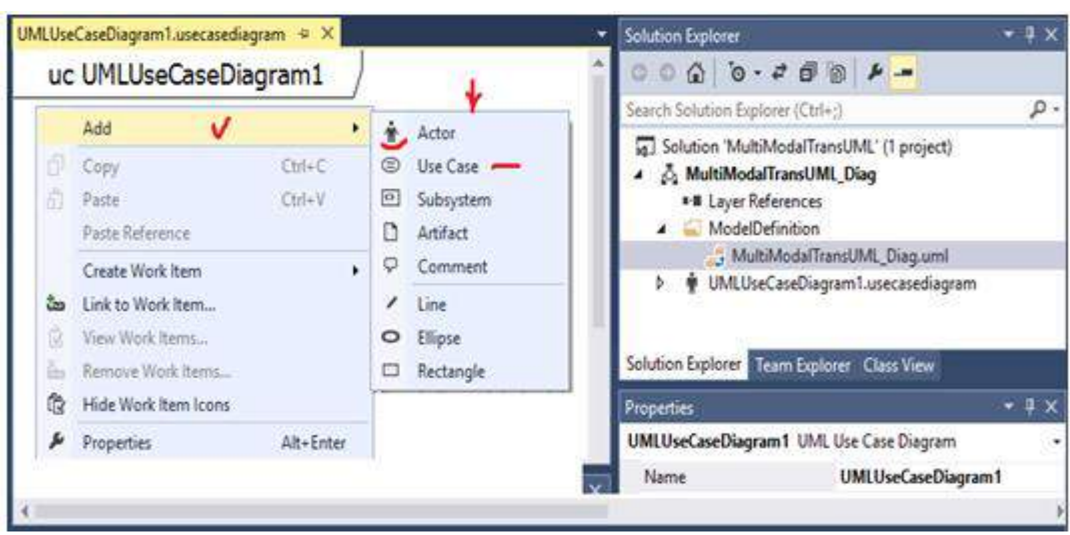
დამკვეთი იღებს შეტყობინებას და უკავშირდება ექსპედიტორს, რათა გამოართვას ტვირთის ორიგინალი დოკუმენტები (კომერციული ინვოისი, სატრანსპორტო დოკუმენტაცია, წარმომავლობის მოწმობა და სხვ).

ექსპედიტორის ფინანსური დეპარტამენტი განიხილავს ინვოისს, ახდენს ანგარიშსწორების პროცესის მიდევნებას და თანხის მიღების შემდეგ ამოწმებს მის სისწორეს. შემდეგ კი ახდენს ტრანსპორტიორებთან და აგენტებთან ანგარიშსწორებას“.

11.2. სისტემის მოთხოვნილების განსაზღვრა

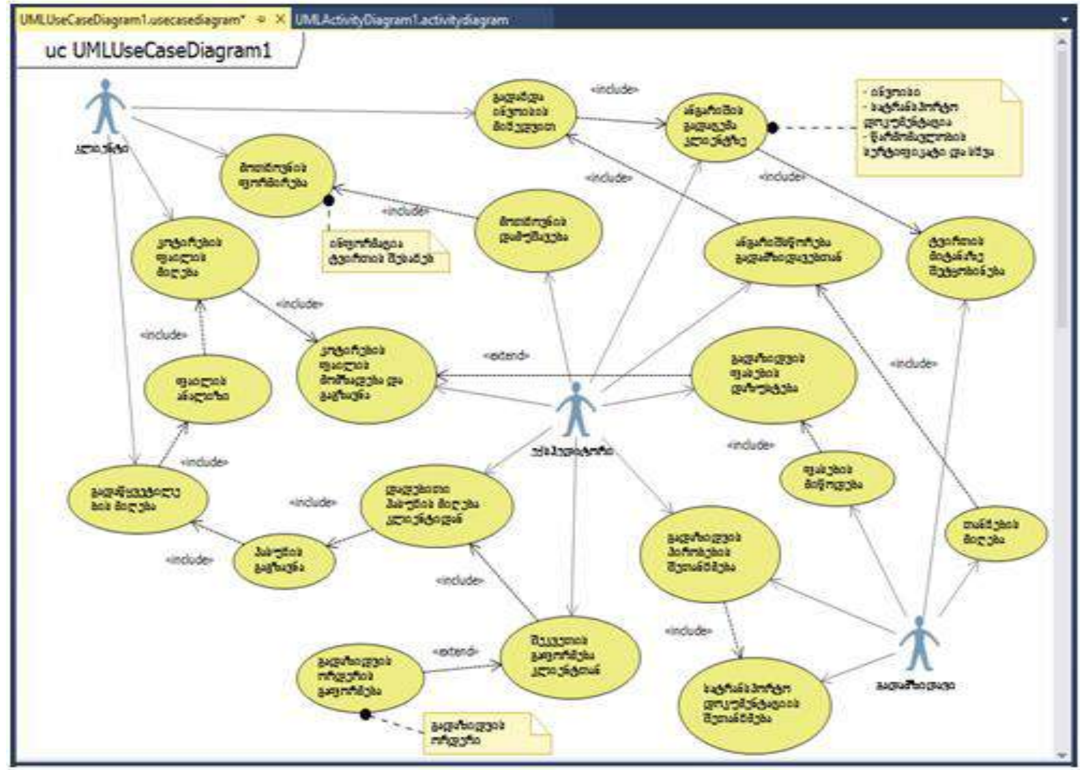
11.1 ნახაზზე ნაჩვენებია Visual Studio.NET 2013/15 სამუშაო გარემოში UseCase დიაგრამის აგების ამოცანა მულტიმოდალური გადაზიდვების საპრობლემო სფეროსათვის.

აქ ჩვენ ავაგებთ UML ტექნოლოგიის 1-ელ და მე-2 ეტაპების დიაგრამებს (მოთხოვნილების განსაზღვრა და სისტემის ოო-ანალიზი).



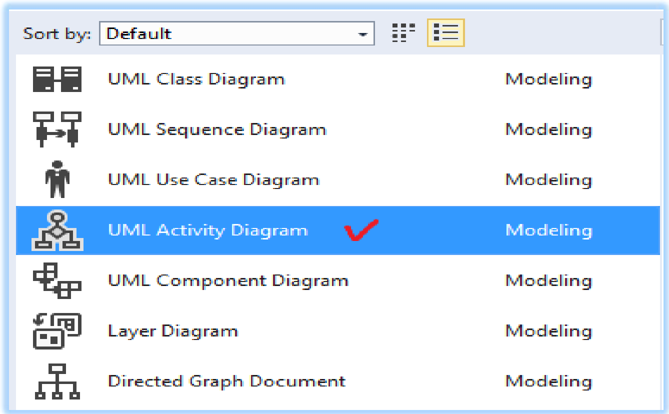
ნახ.11.1. MultiModalTransUML პროექტის აგება

11.2 ნახაზზე ნაჩვენებია Visual Studio.NET-2015 გარემოში UseCase დიაგრამის ფრაგმენტი, „*კლიენტი–ექსპედიტორი–გადამზიდავი*“ როლებით და მათი ფუნქციებით.



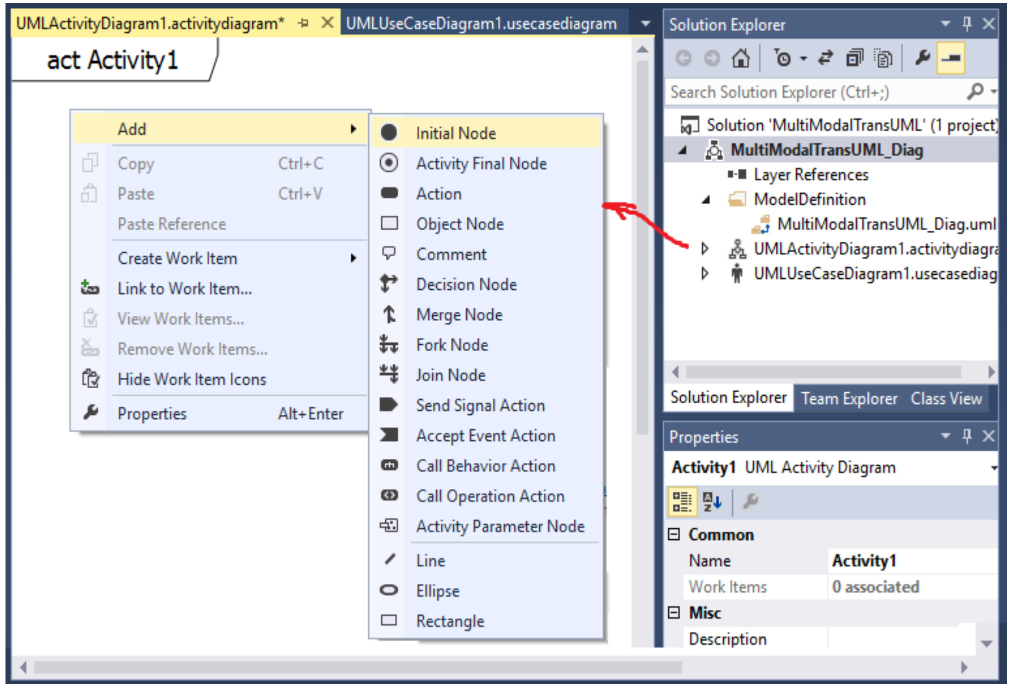
ნახ.11.2. UseCase -დიაგრამა მულტიმოდალური გადაზიდვის სისტემისათვის

შემდეგ ეტაპზე თითოეული ოვალისათვის (როლის ფუნქციისათვის) აგებენ ქმედებათა დიაგრამებს, რომლებსაც აქტიურობათა დიაგრამებს უწოდებენ (Activity-D). 11.3 ნახაზზე ნაჩვენებია Solution Explorer-ის პროექტზე Add new Item ფუნქციით მიღებული ფანჯარა, სადაც უნდა ავირჩიოთ UML Activity Diagram.



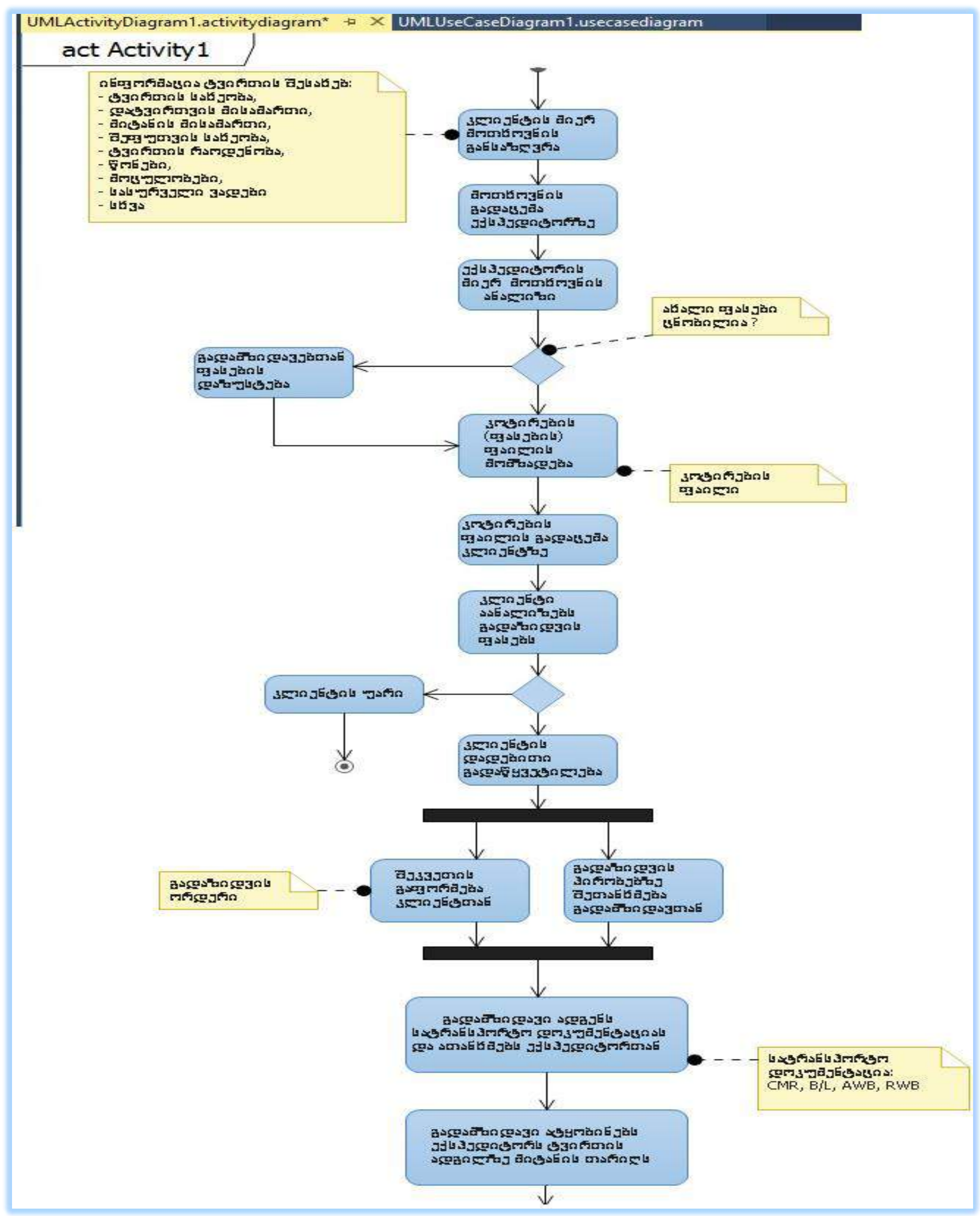
ნახ.11.3. დავალების არჩევა

11.4 ნახაზზე გამოტანილია Visual Studio .NET ახალი სამუშაო გარემო აქტიურობათა დიაგრამის ასაგებად.

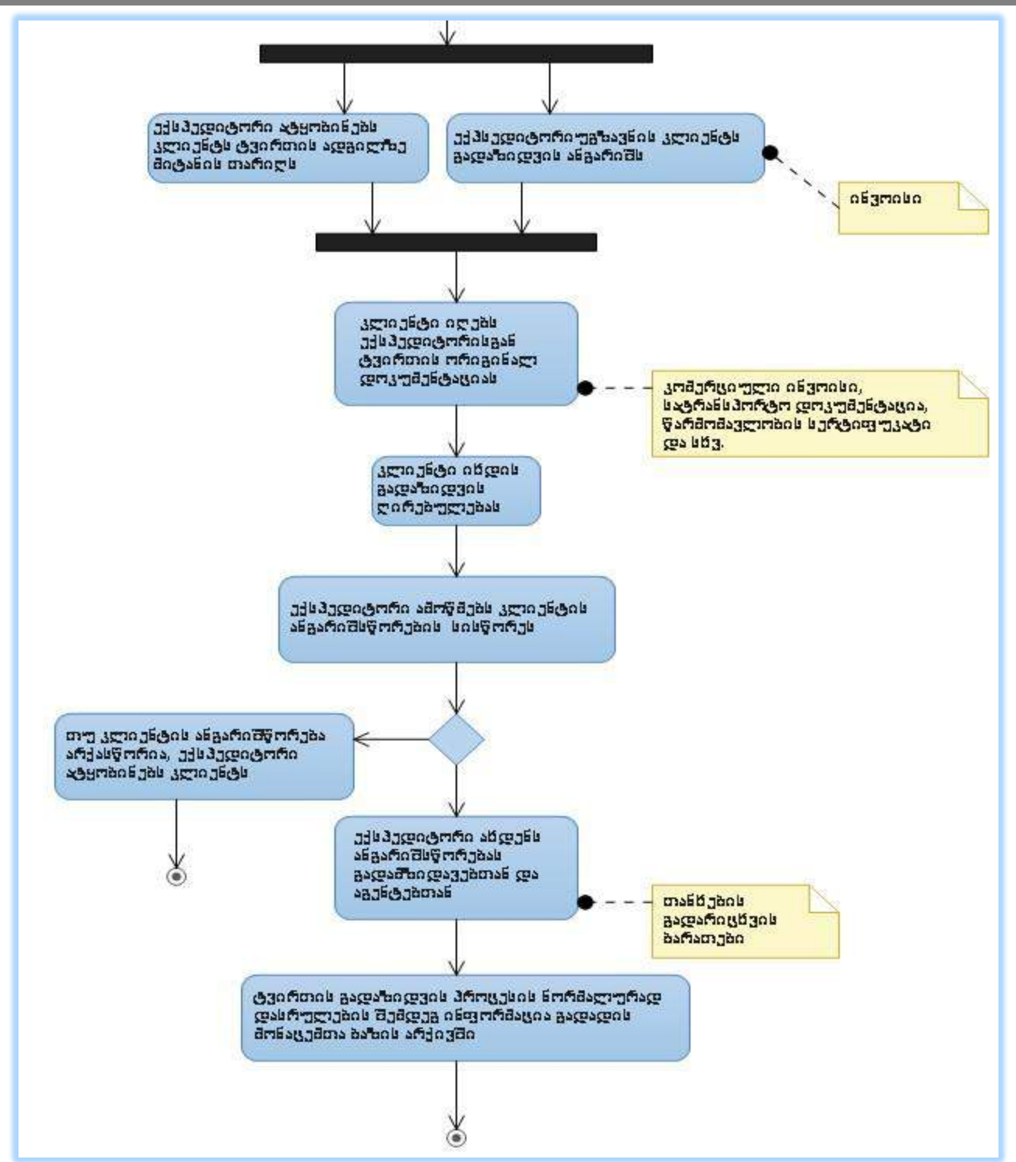


ნახ.11.4. Activity Diagram-ის აგების დასაწყისი

11.5 ნახაზზე წარმოდგენილია მულტიმოდალური გადაზიდვების სისტემის მთლიანი აქტიურობათა დიაგრამა, ბიზნესპროცესებით და ბიზნესწესებით.

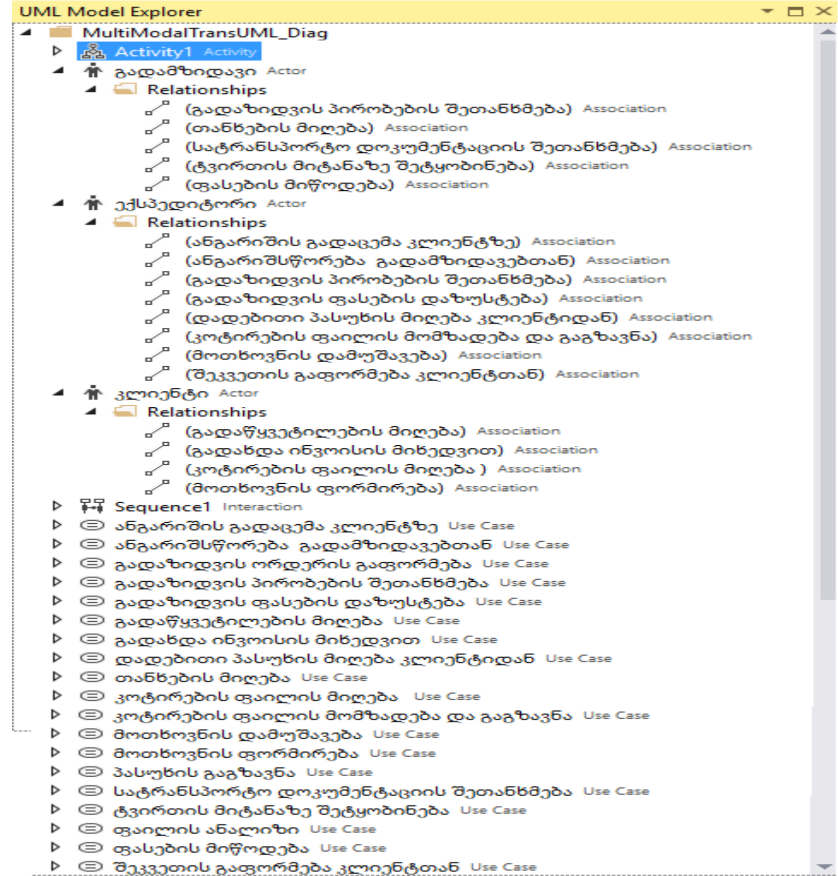


ნახ.11.5. სისტემის აქტიურობათა დიაგრამა (დასაწყისი)



ნახ.11.5. აქტიურობათა დიაგრამა (გაგრძელება)

11.6 ნახაზზე მოცემულია MultiModalTransUML_Diag პროექტის შესაბამისი შიგთავსის იერარქიული ხე, რომელიც მოდელირების პროცესში სისტემამ თვითონ შექმნა ოპერაციების პარალელურად.



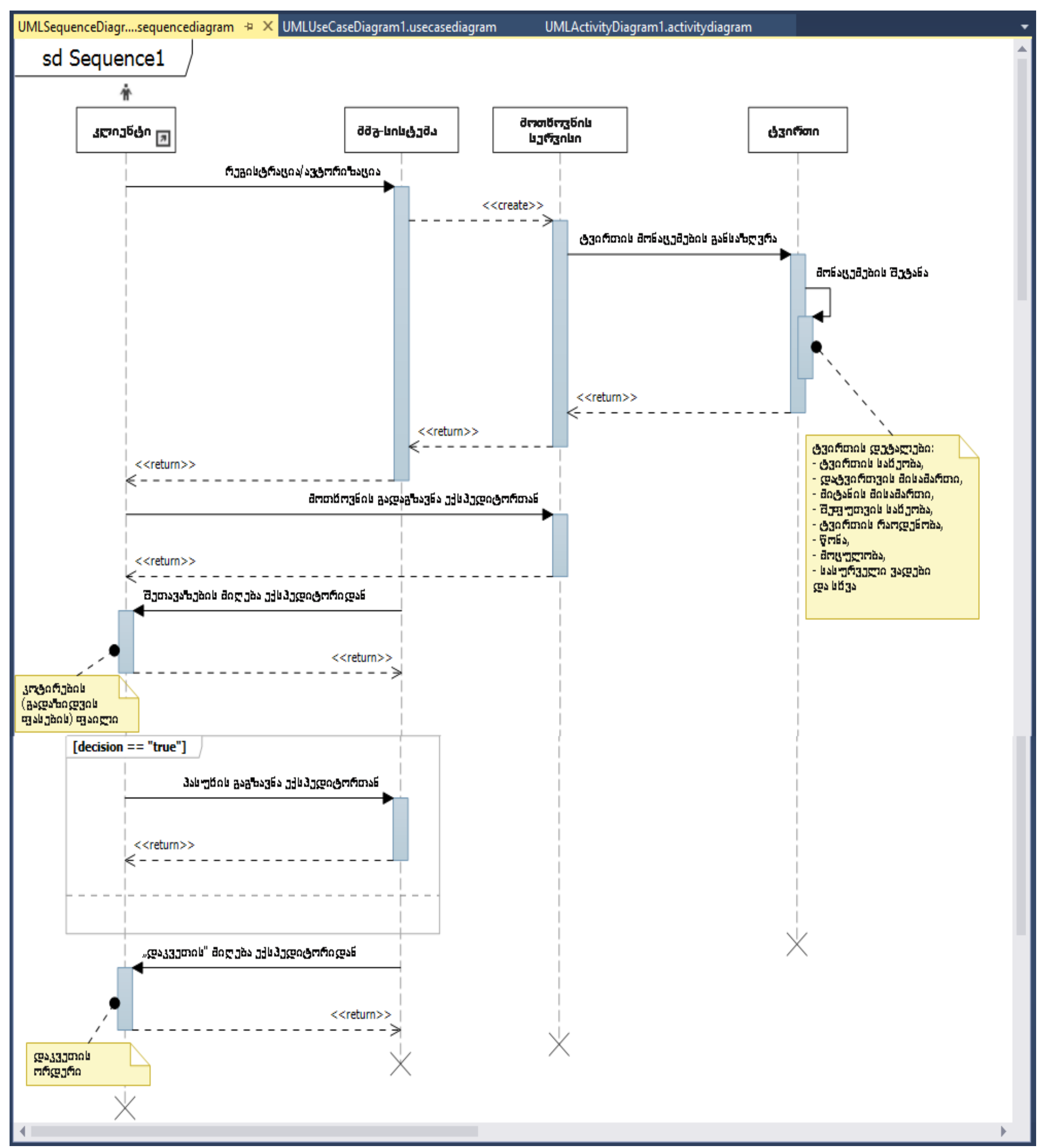
ნახ.11.6. პროექტის შიგთავსის ხე

11.3. ობიექტორიენტირებული ანალიზის ეტაპი და ინტერაქტიული პროცესების აგება

როგორც ცნობილია, ამ ეტაპზე ხორციელდება საპრობლემო სფეროს კლასებისა და ობიექტების ფორმირება, აგრეთვე სისტემაში მონაწილე როლების სცენარების განსაზღვრა. ანუ უნდა აიგოს მულტიმოდალური გადზიდვების პროცესში მონაწილე როლების (კლიენტი, ექსპედიტორი და გადამზიდავი), მიმდევრობითობის და თანამოქმედების დიაგრამები.

11.3.1. Sequence დიაგრამა

11.7 ნახაზზე მოცემულია „კლიენტის“ (ტვირთის მესაკუთრის) მიმდევრობითობის დიაგრამა. მართკუთხედები ასახავს კლასის ობიექტებს, რომელთანაც მას აქვს ურთიერთქმედება შეტყობინებათა გაცვლის დონეზე, რომლის საფუძველზეც უნდა ამოქმედდეს შესაბამისი კლასის მეთოდები (ფუნქციების შესასრულებლად).



ნახ.11.7. Sequence-D როლისთვის „კლიენტი”

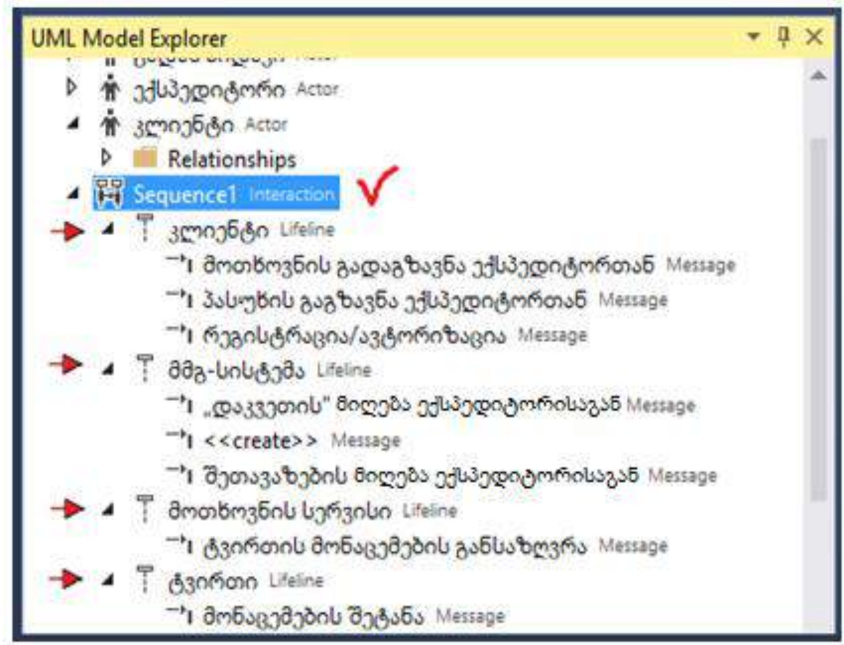
როლის ქმედებები ჩალაგებულია დროში მიმდევრობით ზემოდან ქვემოთ. დასაწყისში კლიენტი შედის „ტვირთების გადაზიდვის“ რომელიმე ფირმის საიტზე და აინტერესებს პირობები თავისი მიზნებისთვის. ფირმა უმეტეს შემთხვევაში სთავაზობს იურიდიულ ან ფიზიკურ პირს რეგისტრაციის გავლას. თუ პირი „ძველია“, ის თავისი სახელით და პაროლით გადის ავტორიზაციას.

შემდეგ პოტენციური კლიენტი ავსებს ფირმის „სერვის-ფორმას“, სადაც მიუთითებს ტვირთის მახასიათებლებს და გეოგრაფიულ მისამართებს (საწყისი, საბოლოო). „სერვის-ფორმა“ ინტერფეისის როლს ასრულებს კლიენტისათვის და იგი განთავსებულია მომსახურეობის ფირმის საიტზე. აქ შეიტანება შემდეგი მონაცემები: **ტვირთის** ტიპი, მდგომარეობა, შეფუთვის ტიპი, ერთეულის ზომები (სიგრძე, სიგანე, სიმაღლე), ერთეულის მოცულობა, ჯამური მოცულობა, ერთეულის წონა, ერთეულის რაოდენობა, ჯამური წონა, უსაფრთხოობა, საბაჟო კოდი, გამგზავნი, მიმღები და სხვ.;

როდესაც კლიენტი, გაეცნობა ექსპედიტორიდან მიღებულ ინფორმაციას გადაზიდვების პირობების შესახებ, იღებს გადაწყვეტილებას (დადებითი ან უარყოფითი) შემდეგი ქმედებების შესახებ. დადებითის დროს ექსპედიტორს უგზავნის შეტყობინებას, რომ მზადაა ტვირთის გადასაცემად.

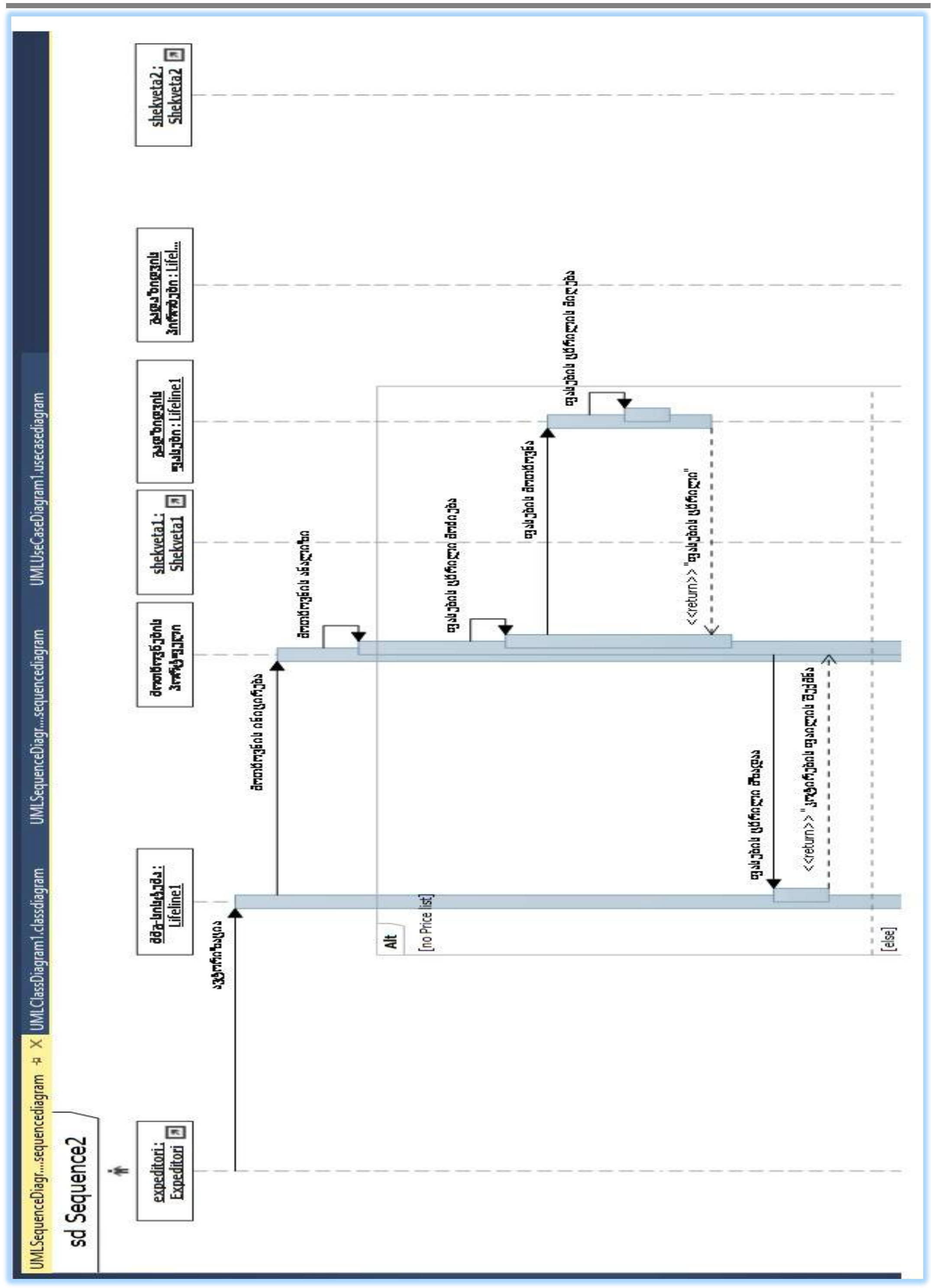
როდესაც მოლაპარაკება დადებითად გადაწყდება, ექსპედიტორი, ანიჭებს ტვირთს იდენტიფიკატორსა და გადაზიდვის ხელშეკრულების იდენტიფიკატორს.

11.8 ნახაზზე ნაჩვენებია პროექტის შიგთავსის ფრაგამენტი Sequence დიაგრამისათვის.

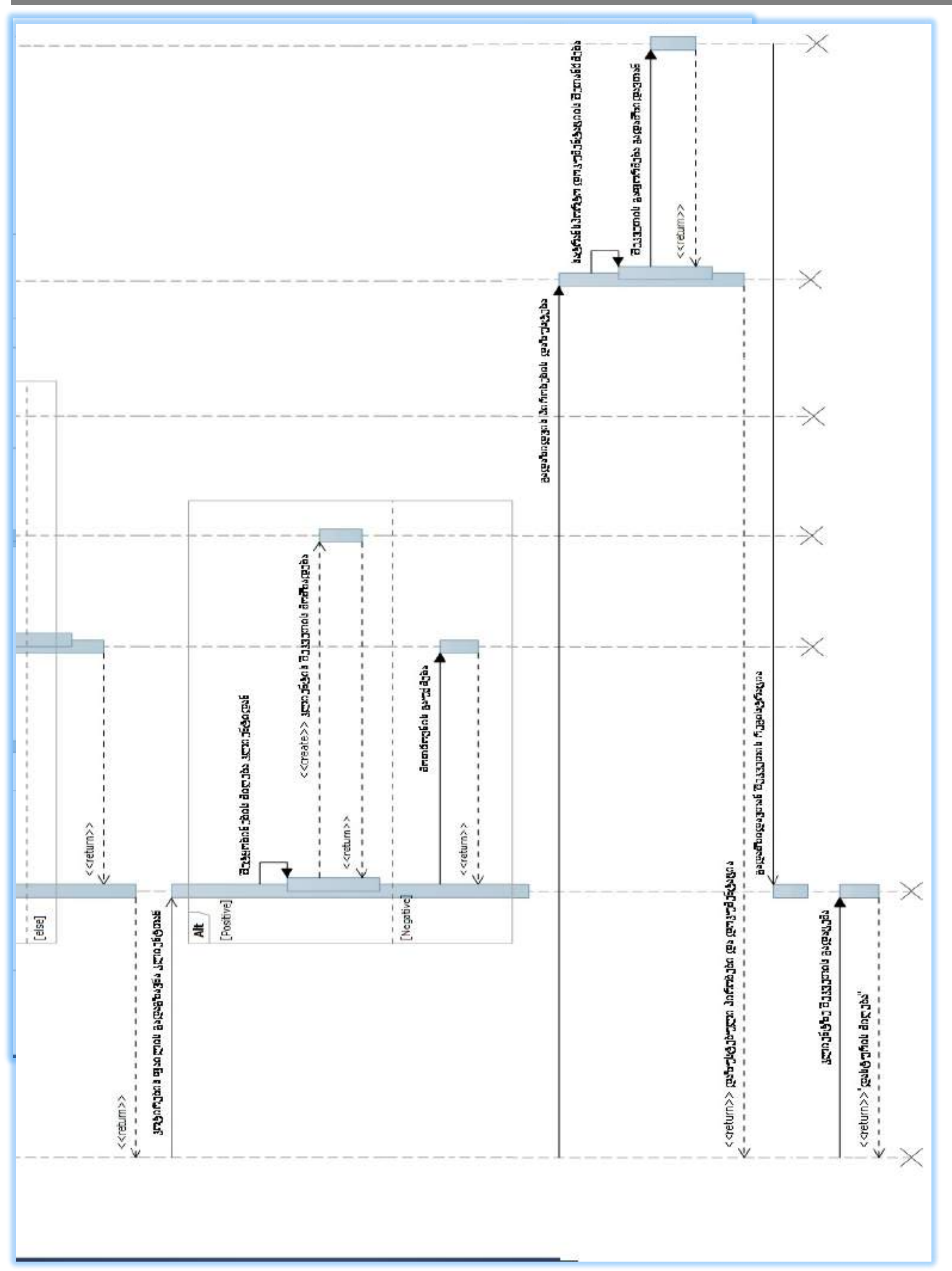


ნახ.11.8. პროექტის Sequence დიაგრამის შიგთავსის ფრაგმენტი

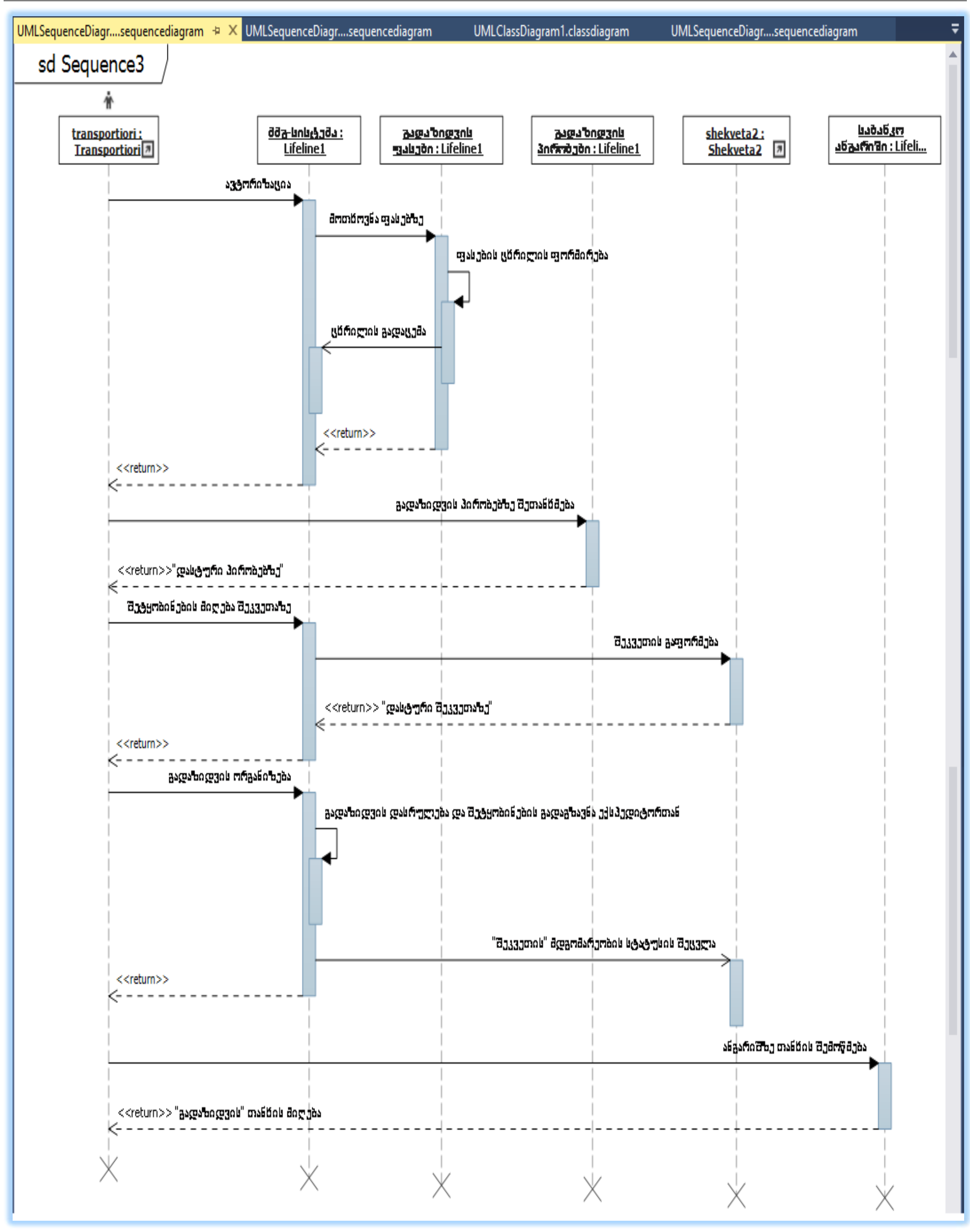
ასეთივე წესით აიგება Sequence დიაგრამები ექსპედიტორისა და გადამზიდავისათვის, რაც ნაჩვენებია 11.9 და 11.10-ა,ბ ნახაზებზე.



ნახ.11.10-ა. Sequence დიაგრამა “ექსპედიტორი” (დასაწყისი)



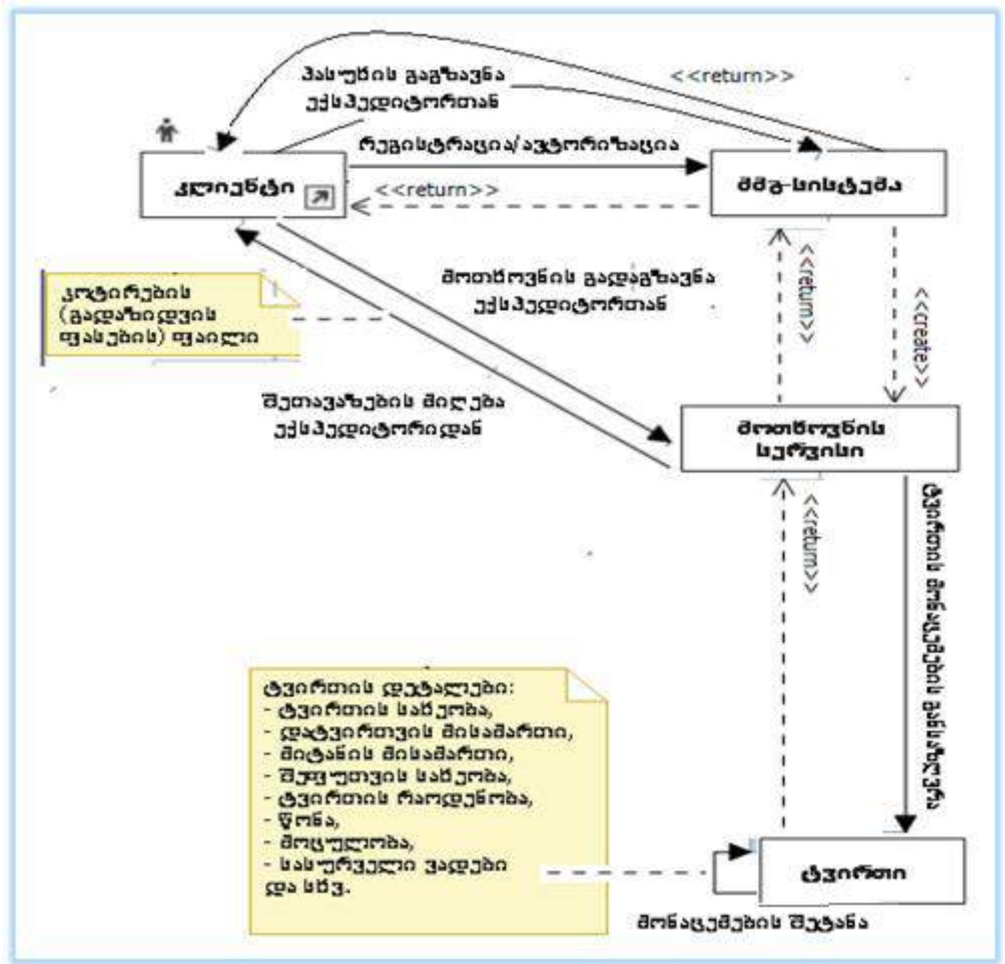
ნახ.11.10-ბ. Sequence დიაგრამა “ექსპედიტორი” (გაგრძელება)



ნახ.11.10. Sequence დიაგრამა „გადამზიდავი"

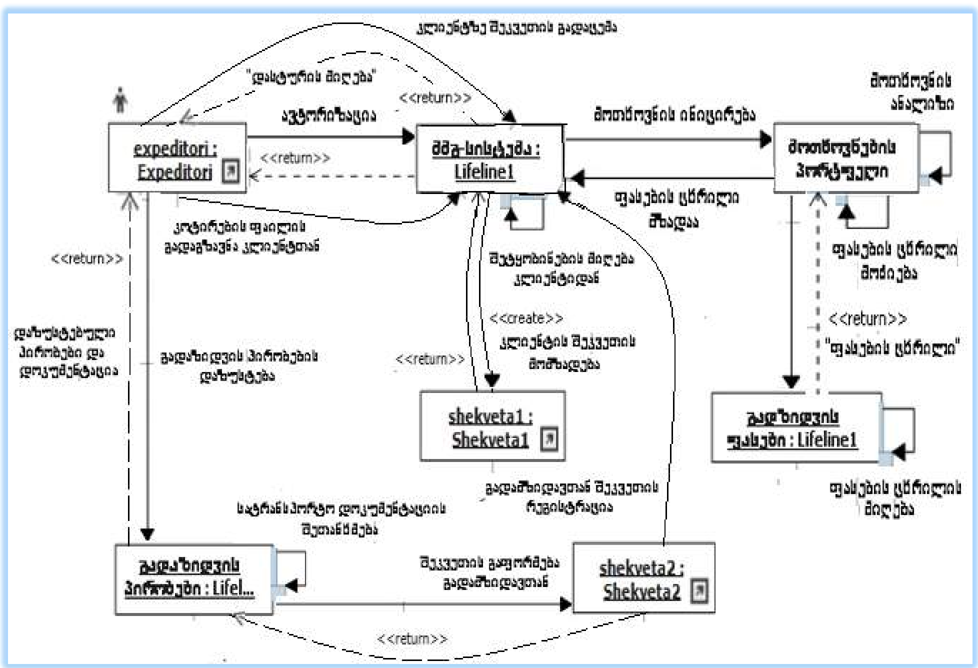
11.3.2. Collaboration დიაგრამა

11.11-13 ნახაზებზე მოცემულია წინა პარაგრაფში აგებული მიმდევრობითობის სქემების შესაბამისი თანამოქმედების დიაგრამები: კლიენტის, ექსპედიტორისა და ტრასპორტიორისათვის.

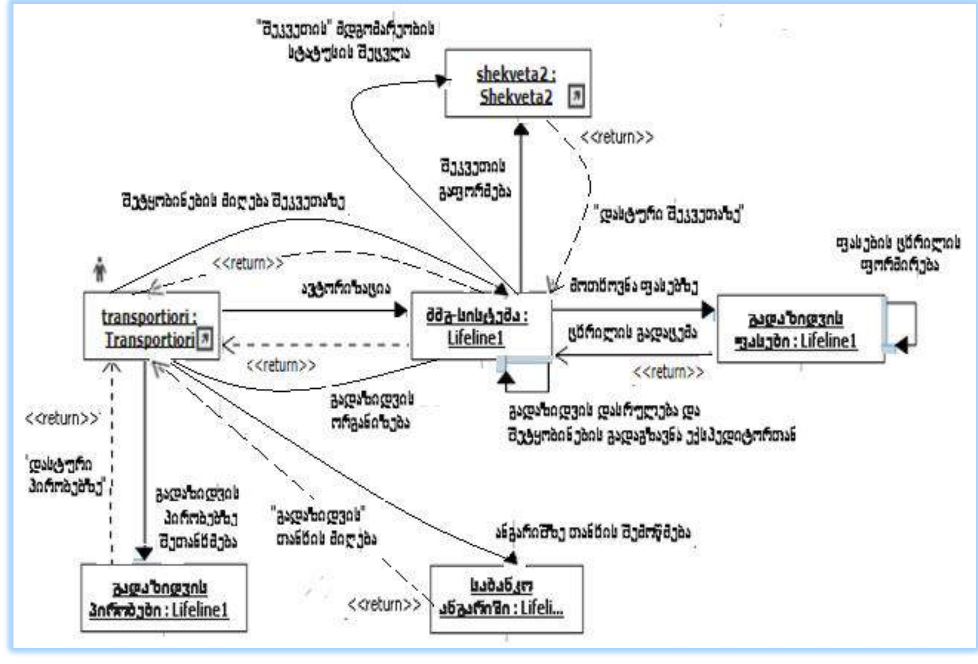


ნახ.11.11. თანამოქმედების დიაგრამა „კლიენტი“

თანამოქმედების დიაგრამებზე აისახება იმავე კლასთა ობიექტები, რაც გვქონდა თანამიმდევრობის დიაგრამებზე. აქ ჩანს ობიექტებს შორის კავშირები შეტყობინებების გაცვლის დონეზე და შესაბამისი მეთოდებისა და ინფორმაციული ნაკადების გაცვლის მდგომარეობები.



ნახ.11.12. თანამოქმედების დიაგრამა „ექსპედიტორი“



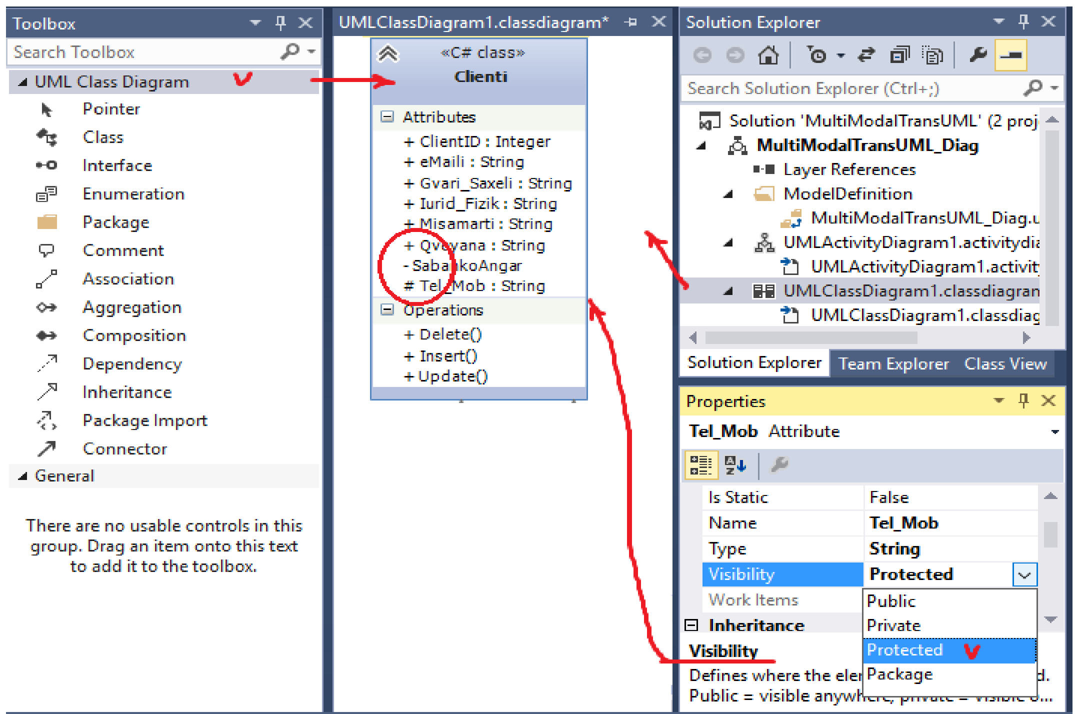
ნახ.11.13. თანამოქმედების დიაგრამა „გადამზიდავი“

XII თავი
ობიექტორიენტირებული დაპროექტება

12.1. კლასთა ასოციაციის დიაგრამა

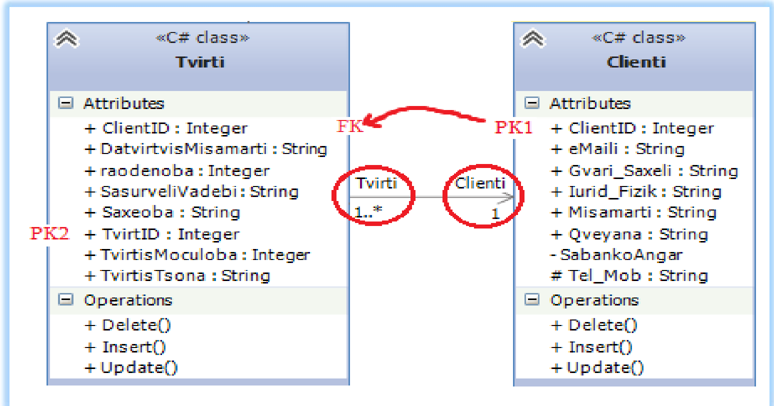
განვაგრძობთ მულტიმოდალური გადაზიდვების საპრობლემო სფეროს განხილვას და ამჯერად შევეხებით კლასების, კლასთაასოციაციისა და მდგომარეობათა დიაგრამების აგების საკითხებს. ეს ამოცანები მიეკუთვნება უნიფიცირებული მოდელირების ტექნოლოგიის ობიექტორიენტირებული დაპროექტების სფეროს.

როგორც ვიცით, კლასი ერთგვაროვან ობიექტთა სიმრავლეა, რომელიც პროგრამული თვალსაზრისით მონაცემთა კომპლექსური ტიპია (სტრუქტურაა, ოღონდ მომხმარებლის კერძო სახის, რომელშიც შესაძლებელია გარკვეული ცვლადების (მონაცემების) და ფუნქციების (მეთოდების) დამალვა. Properties თვისებაში Visibility იქნება Private (-) ან Protection (#). ყველასათვის ხელმისაწვდომი მონაცემები და მეთოდები (+)-ითაა მოცემული, რაც Public-ს შეესაბამება [2]. სისტემის ობიექტების შესაბამისი კლასების აგება Visual Studio. NETგარემოში, UML Class Diagram ინსტრუმენტების პანელით, Client კლასით, Solution Explorer-ით და Properties-ით ნაჩვენებია 12.1 ნახაზზე.



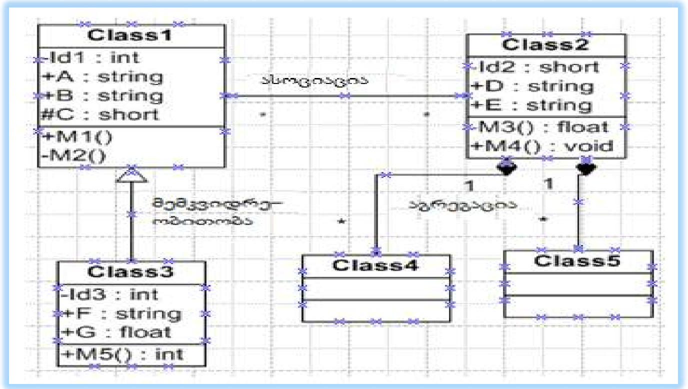
ნახ.12.1. კლასის აგების მაგალითი VisualStudio.NET გარემოში

12.2 ნახაზზე მოცემულია ორი კლასის შემთხვევა (Client და Tvirti). ანუ სქემაზე დაემატა ტვირთის შესაბამისი კლასი (ატრიბუტებით და მეთოდებით). აქვე განისაზღვრა კავშირები ამ ორ კლასს შორის.



ნახ.12.2. კლასის დამატება დიაგრამაზე

კლასთაშორისი კავშირები შეიძლება იყოს: მემკვიდრეობითი, აგრეგატული, რელაციური და ასოციაციური (ნახ.12.3):



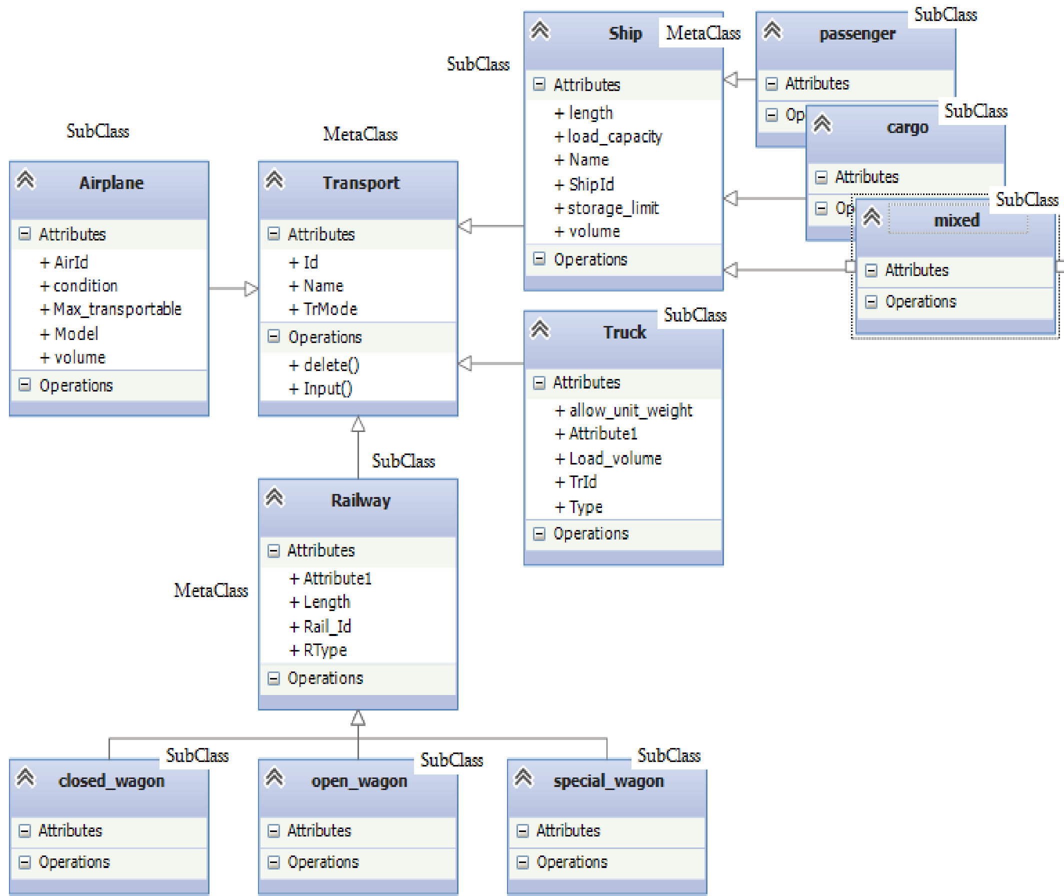
ნახ.12.3. კლასთაშორის კავშირთა სახეები

- მემკვიდრეობითი (Generalization) ასახავს „გენეტიკურ“, განზოგადებულ კავშირებს კლასებს შორის. ასეთ დროს ერთი კლასი („შვილი“) მთლიანად იღებს მეორე კლასის („მშობელი“) ყველა ატრიბუტს, მეთოდას და კავშირს;
- აგრეგირებული (Aggregation) ნიშნავს კავშირს „მთელი-ნაწილი“. მაგალითად, „ავტომობილი“ – „ძარა, ძრავი, საბურავები და ა.შ.“;
- ასოციაციური (Assotiation) ნიშნავს სემენტიკურ კავშირს კლასებს შორის. ის შეიძლება გამოისახოს ერთ- ან ორმიმართულ-ლებიანი (იგივეა, რაც უისრო) ხაზით. ისარი

გვიჩვენებს შეტყობინების გადაცემის მიმართულებას. ასოციაციური კავშირის რეალიზება ხდება ერთ კლასში დამატებით მეორე კლასის ატრიბურტის ჩასმით. ეს ჰგავს პირველადი (Primary - PK) და მეორეული (Foreign - FK) გასაღებური ატრიბუტების შეერთებას;

- რელაციური (Dependency) ნიშნავს ერთი კლასის დამოკიდებულებას მეორეზე. იგი ერთმიმართულებიანი წყვეტილი ისრით გამოიხატება. მასში დამატებითი დამაკავშირებელი ატრიბუტები არ გამოიყენება.

მაგალითად, ჩვენი შემთხვევისათვის შეიძლება კლასი Transport იყოს მეტაკლასი. 12.4 ნახაზზე ილუსტრირებულია კლასთა ასოციაციის დიაგრამა მემკვიდრეობითი კავშირების საფუძველზე. ისარი მიმართულია „შვილიდან“ „დედისაკენ“, რაც მათ ცალსახა დამოკიდებულებაზე მეტყველებს. „შვილს“ ჰყავს ერთი „დედა“, ხოლო „დედას“ შეიძლება ჰყავდეს რამდენიმე „შვილი“, ამიტომაც ეს არაა ცალსახა.



ნახ.12.4. მემკვიდრეობითი (inheritance) კავშირები ტრანსპორტის სახეების კლასებს შორის (კლასიფიკაცია)

მშობელი კლასი ლიტარატურაში ზოგჯერ „მეტაკლასად“ (MetaClass) მოიხსენიება, რომელიც შედგება ქვეკლასებისაგან (SubClasses). შეიძლება იერარქიაში ქვეკლასი იყოს მის ქვევით მდგარი კლასისათვის მეტაკლასი. მაგალითად, კლასი MetaClass_Transport არის მეტაკლასი ოთხი ქვეკლასისთვის: SubClass_ Airplane, SubClass_ Truck, SubClass_ Ship და SubClass_ Railway.

ამასთანავე, კლასი RailWay არის მეტაკლასი closed_wagon, open_wagon და special_wagon ქვეკლასებისათვის, ხოლო მეტაკლასი Ship კი - passenger, cargo და mixed ქვეკლასებისათვის.

ტვირთი – წარმოების პროდუქციაა (ნედლეული, ნახევარფაბრიკატები, მზა პროდუქცია), ტრანსპორტის მიერ გადაზიდვით მიღებული [1-5].

ტვირთის თვისებები ან ატრიბუტები, რომლებიც მონაცემთა ბაზაში უნდა იქნას შენახული, შემდეგია: იდენტიფიკატორი, ტიპი, მდგომარეობა, შეფუთვის ტიპი, ერთეულის ზომები (სიგრძე, სიგანე, სიმაღლე), ერთეულის მოცულობა, ჯამური მოცულობა, ერთეულის წონა, ერთეულის რაოდენობა, ჯამური წონა, უსაფრთხოობა, საბაჟო კოდი, გამგზავნი, მიმღები, გადაზიდვის ხელშეკრულების იდენტიფიკატორი და სხვ. ამგვარად, ტვირთი ხასიათდება შენახვის რეჟიმით, შეფუთვის, გადატვირთვისა და გადაზიდვის ხერხებით, ფიზიკურ-ქიმიური თვისებებით, გადაზიდვაზე წარდგენილი ტვირთების ზომებით, მოცულობით, მასითა და ფორმით [5].

თუ ტვირთი შეფუთულია გადაზიდვების პირობების შესაბამისად, მარკირებულია წესების მიხედვით და იმყოფება საჭირო კონდიცირებულ მდგომარეობაში, იგი შეიძლება დაცულად იქნას გადაზიდული. ასეთ შემთხვევაში ითვლება, რომ იგი იმყოფება ტრანსპორტაბელურ მდგომარეობაში.

ტვირთის მდგომარეობების ერთობლიობა, მისი საწყისი მდგომარეობიდან საბოლოო მდგომარეობამდე, სატრანსპორტო წესებისა და ოპერაციების ჩათვლით, განხილული გვექნება მომდევნო პარაგრაფში მდგომარეობათა დიაგრამების (Statechart diagrams) სახით.

ახლა კი წარმოვადგინოთ ტვირთის, როგორც ობიექტის სახეები მისი კლასიფიკაციის საფუძველზე (ნახ.12.5).

როგორც ნახაზიდან ჩანს, ტვირთების სახეობების დაჯგუფება შესაძლებელია შემდეგ კატეგორიებად:

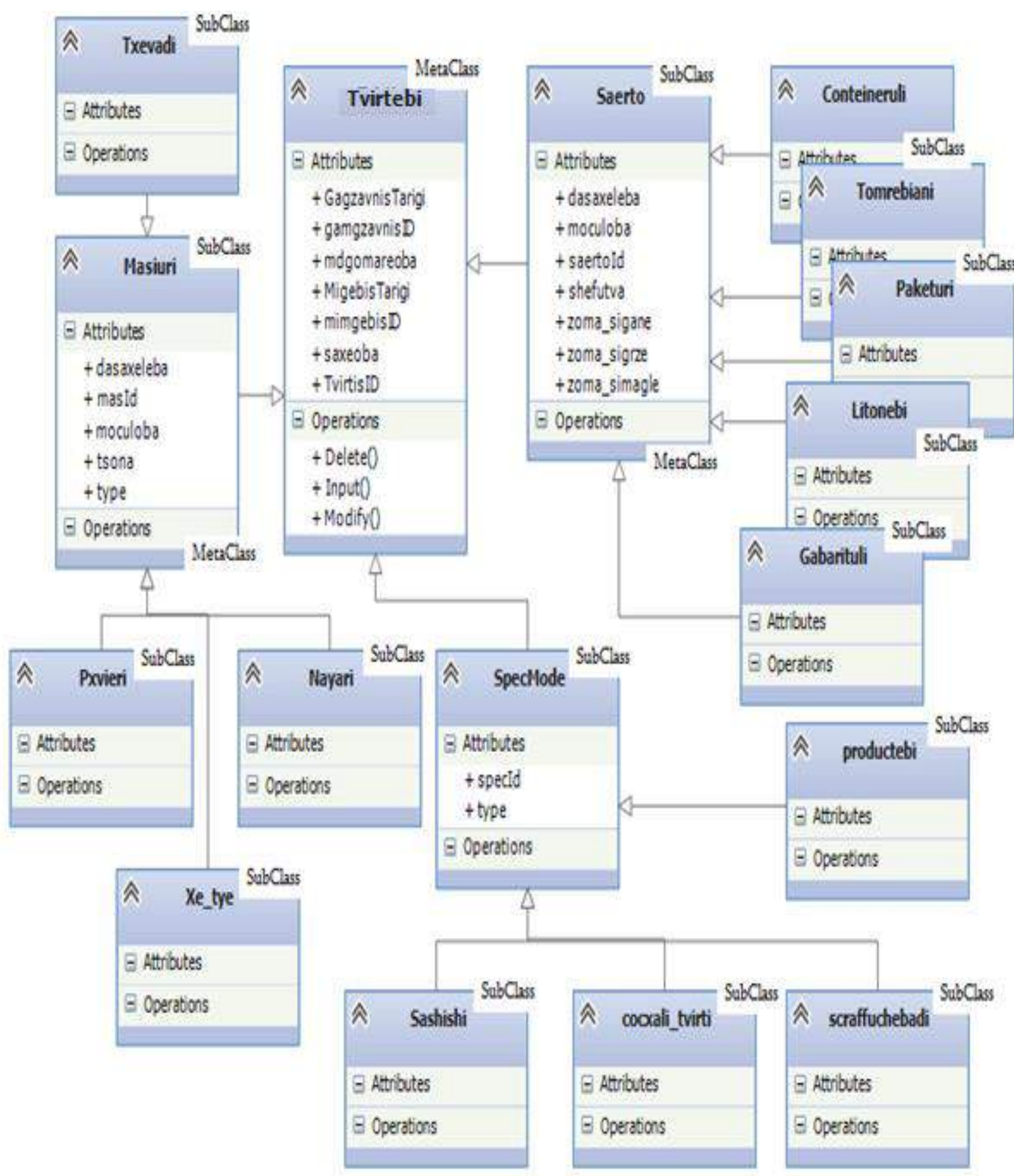
– **ნაყარი და დაშლილი ნაყარი ტვირთები** [5].

ნაყარი ტვირთის კატეგორიებს მიეკუთვნება თხევადი, მშრალი ნაყარი, ნეო-ნაყარი, ბორბლიანი და გაყინული/გაგრილებული ტვირთები.

– **თხევადი:** ნედლი ნავთობი, ნავთობპროდუქტების უმრავლესობა, ღვინო, გათხევადებული ნახშირი;

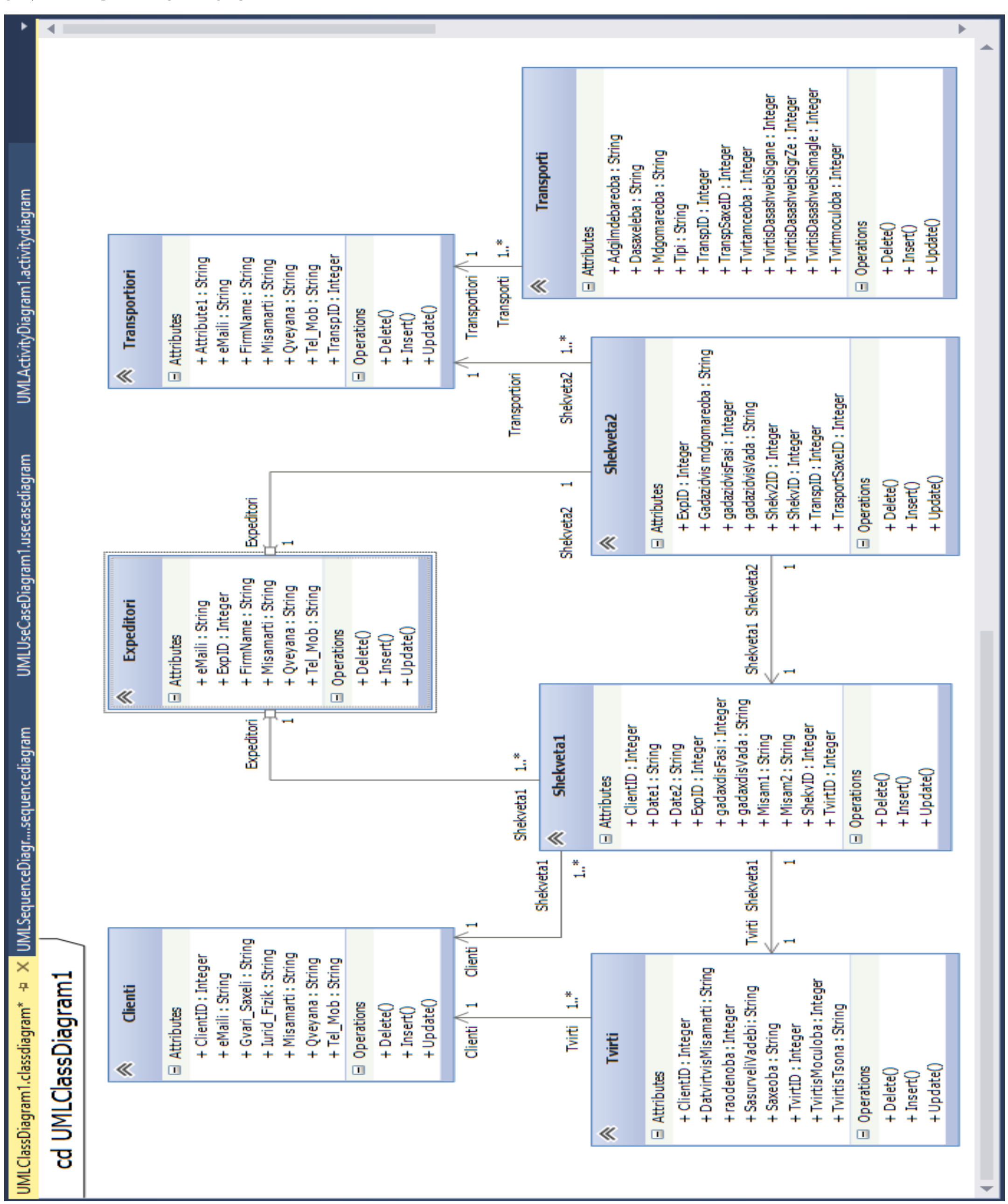
– **მშრალი ნაყარი:** მარცვლეული, შაქარი, ფხვნილები (ალუმინის ჟანგი, თიხამიწა, ცემენტი);

– **ნეო-ნაყარი:** ტყის პროდუქტები, ფოლადის პროდუქტები, ბეილირებული ჯართი;
– **ბორბლიანი:** ავტომანქანები, სატვირთო მანქანები, სარკინიგზო ვაგონები;
– **გაყინული/გაგრილებული:** ხორცი, ხილი, რძის პროდუქტები და ა.შ.



ნახ.12.5. მემკვიდრეობითი (inheritance) კავშირები ტვირთის
სახეების კლასებს შორის (კლასიფიკაცია)

12.6 ნახაზზე მოცემულია ტვირთების მულტიმოდალური გადაზიდვების პროცესის კლასთა ასოციაციის დიაგრამა, რომელზეც განხორციელებულია სხვადასხვა კლასის ურთიერთკავშირი.

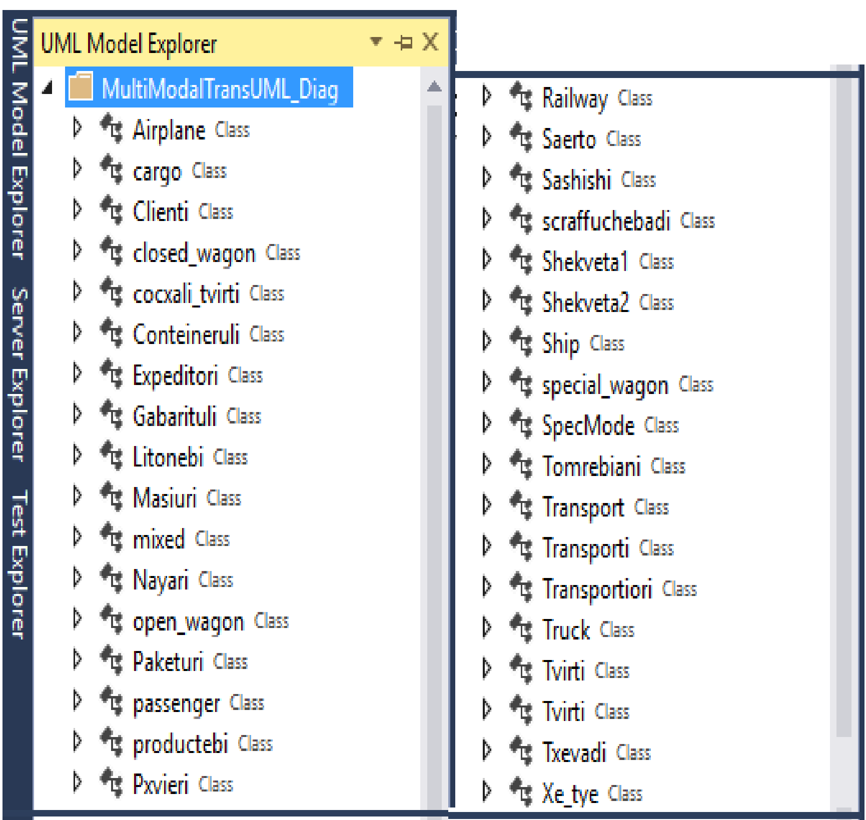


ნახ.12.6. Class-D:MMT-System

ამ ნახაზზე ჩანს აგრეთვე თითოეული კლასის სავარაუდო შედგენილობა (კლასის ცვლადები მონაცემთა ტიპებით და კლასის მეთოდები).

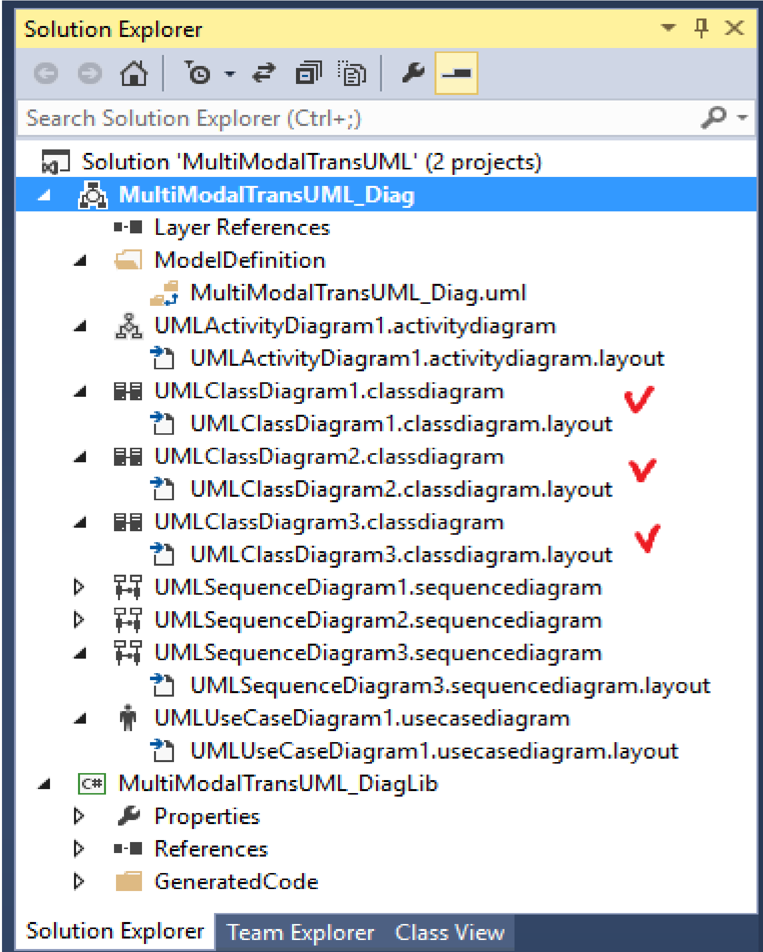
მიღებული დიაგრამის საფუძველზე შესაძლებელია თითოეული კლასის შესაბამისი კოდის გენერაციის პროცედურის ჩატარება და საბოლოოდ C#-პროგრამის ლისტინგების მიღება.

VisualStudio.NET_2013/15 სამუშაო გარემოში დაფიქსირდა ჩვენს მიერ დაპროექტებული კლასების ერთობლიობა (ნახ.12.7).



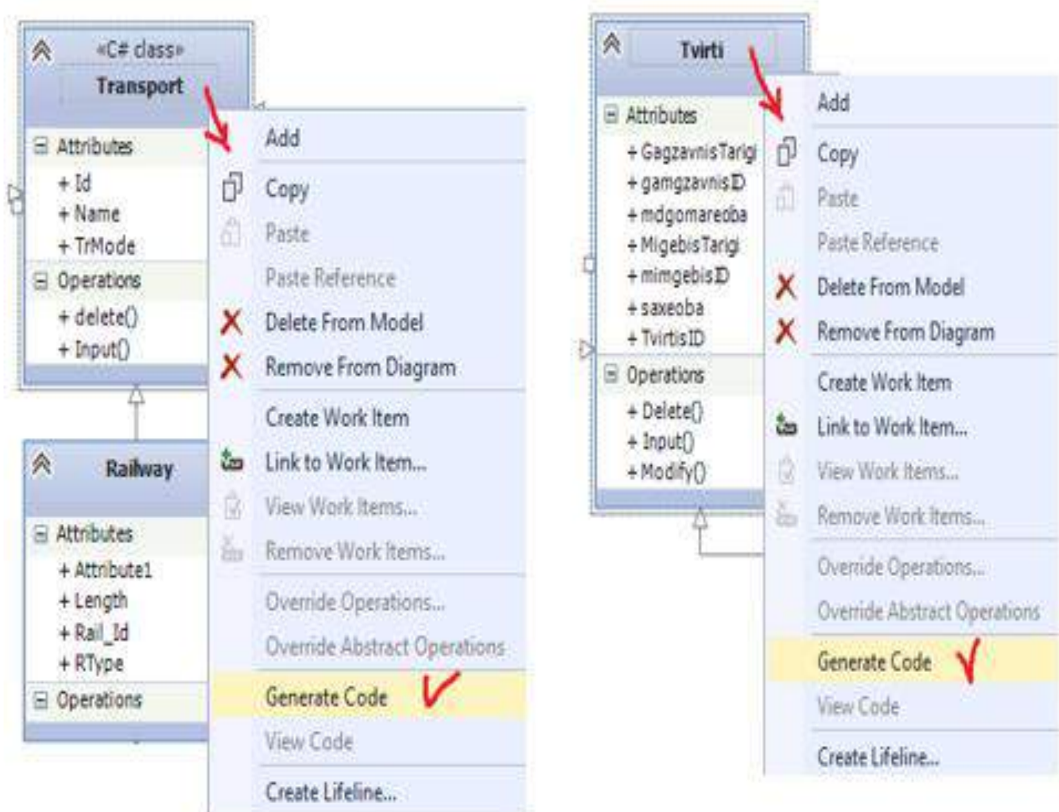
ნახ.12.7. პროგრამული პაკეტით ფორმირებული კლასების სია (VisualStudio.NET_2013/15)

12.8 ნახაზზე მოცემულია MultiModalTransUML_Diag პროექტის შიგთავსი სისტემის Solution Explorer-ში.

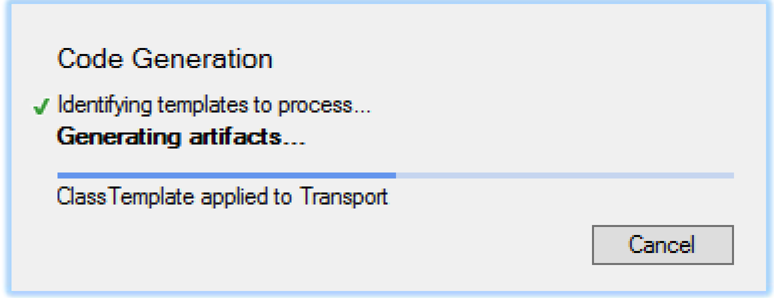


ნახ.12.8. სისტემის Solution Explorer კლასების დიაგრამებით

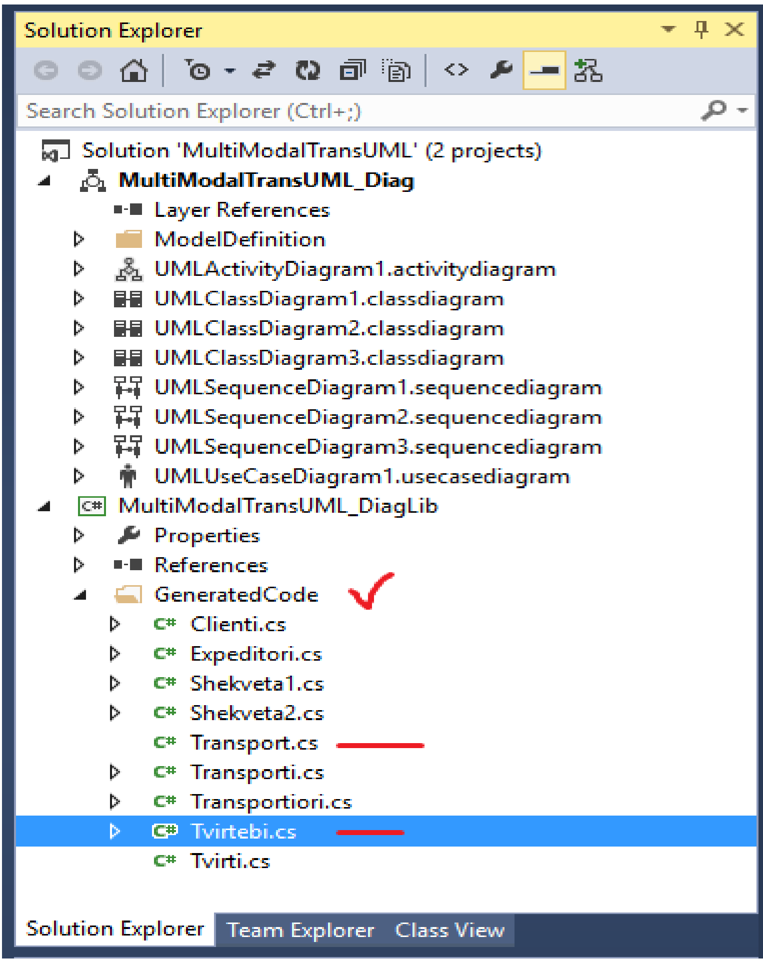
პროგრამული კოდის ავტომატური გენერაციის შესრულება მნიშვნელოვნად ამარტივებს სისტემის პროგრამული უზრუნველყოფის შექმნას (ნახ.12.9-11).



ნახ.12.9. C# კოდების გენერაცია მულტიმოდალური გადაზიდვების Transport და Tvirti კლასების დიაგრამებისთვის



ნახ.12.10. მიმდინარეობს C# კოდის გენერაციის პროცესი



ნახ.12.11. შედეგები: შეიქმნა ახალი C#-კლასის კოდები:
Transport და Tvirtebi

ქვემოთ ლისტინგებში ნაჩვენებია Transport.cs და Tvirtebi.cs კლასების შესაბამისი კოდის ფრაგმენტები, რომლებიც გენერირებულ იქნა ავტომატიზებულ რეჟიმში თვით VisualStudio.NET პროგრამული პაკეტის მიერ, რაც რევერსული ინჟინერინგის კარგი მაგალითია.

```
//----ლისტინგი_1 --------- Transport ----------------------
// <auto-generated>
//     This code was generated by a tool
//     Changes to this file will be lost if the code is
//     regenerated.
```

```
// </auto-generated>
//-----------------------------------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
public class Transport
{
   public virtual object Id {get; set; }
   public virtual object Name {get; set; }
   public virtual object TrMode {get; set; }
   public virtual void Input() {
         throw new System.NotImplementedException();
       }
   public virtual void delete() {
         throw new System.NotImplementedException();
       }
}

//------ ლისტინგი_2---------- Tvirtebi ---------------------
// <auto-generated>
//      This code was generated by a tool
//      Changes to this file will be lost if the code is
//      regenerated.
// </auto-generated>
//-----------------------------------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
public class Tvirtebi
{
   public virtual object TvirtisID { get; set; }
   public virtual object saxeoba { get; set; }
  public virtual object mdgomareoba { get; set; }
  public virtual object gamgzavnisID { get; set; }
  public virtual object mimgebisID { get; set; }
```

```
  public virtual object GagzavnisTarigi {get; set;}
  public virtual object MigebisTarigi {get; set; }
  public virtual void Input(){
          throw new System.NotImplementedException();
        }
  public virtual void Modify() {
          throw new System.NotImplementedException();
        }
  public virtual void Delete(){
          throw new System.NotImplementedException();
        }
}
```

12.2. მდგომარეობათა (Statechart) დიაგრამები

Statechart დიაგრამა არის UML-ის ერთ-ერთი მოდელი, რომელიც გამოიყენება სისტემის დინამიკური ქცევის აღწერის მიზნით. იგი განსაზღვრავს ობიექტის სახვადასხვა მდგომარეობებს მისი არსებობის მთელი პერიოდის მანძილზე [11]. ეს მდგომარეობები იცვლება მოვლენების (events) შესაბამისად.

Statechart-ის გამოყენება სასარგებლოა რეაქციული სისტემებისათვის. ესაა სისტემა, რომელიც რეაგირებს შიგა ან გარე მოვლენებზე.

Statechart დიაგრამა აღწერს მართვის ნაკადს ერთი მდგომარეობიდან სხვა მდგომარეობაში. მდგომარეობა არის ის, რომელშიც იმყოფება ობიექტი და იცვლება მაშინ, როდესაც ამოქმედდება მოვლენა. ამგვარად, Statechart დიაგრამის მნიშვნელოვანი მიზანია ობიექტის სასიცოცხლო დროის მოდელირება მისი შექმნიდან არსებობის დასრულებამდე.

Statechart დიაგრამა გამოიყენება აგრეთვე სისტემების პირდაპირი და რევერსიული პროექტირებისათვის, მაგრამ მისი მთავარი მიზანი მაინც რეაქციული სისტემის მოდელირებაა.

Statechart-ის გამოყენების ძირითადი მიზნებია:

- სისტემის დინამიკური ასპექტების მოდელირება;
- რეაქციული სისტემის სასიცოცხლო დროის მოდელირება;
- ობიექტების სხვადასხვა მდგომარეობების აღწერა მისი მოქმედების პერიოდში;
- მდგომარეობათა მანქანის (სასრული ავტომატის) განსაზღვრა ობიექტის მდგომარეობათა მოდელირებისათვის.

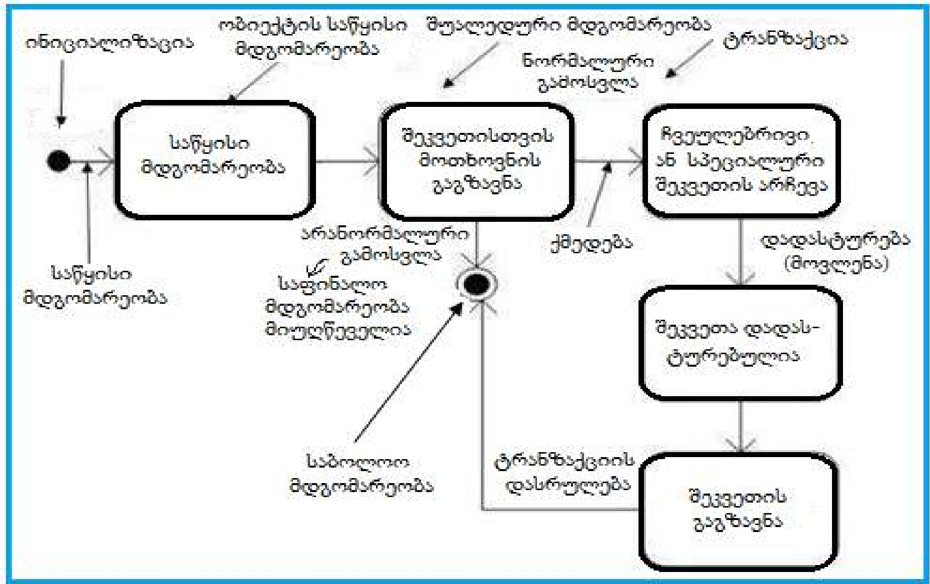
Statechart დიაგრამის გამოყენება ყველა კლასისთვის არაა საჭირო. აუცილებელია მხოლოდ მაშინ, როდესაც კლასი შეიძლება იმყოფებოდეს რამდენიმე მდგომარეობაში და თითოეულ მათგანში მისი ქცევა იყოს სხვადასხვანაირი.

სისტემების ობიექტ-ორიენტირებული ანალიზისა და პროექტირების დროს, სანამ ავაგებთ Statechart დიაგრამას, უნდა დავადგინოთ:

- ძირითადი ობიექტები, რომელთა ანალიზია საჭირო;
- მდგომარეობები;
- მოვლენები.

განვიხილოთ 12.12 ნახაზზე მოცემული Statechart დიაგრამა, რომელშიც ხდება „შეკვეთა“კლასის ობიექტების მდგომარეობათა ანალიზი [12-15].

პირველი მდგომარეობა არის არააქტიური მდგომარეობა (Idle), საიდანაც იწყება პროცესი. შემდეგი მდგომარეობები მიიღება ისეთი მოვლენებით, როგორიცაა: „მოთხოვნის გაგზავნა“ (*send request*), „მოთხოვნაზე დასტური“ (*confirm request*) და „შეკვეთის გაგზავნა“ (*dispatch order)*. ეს მოვლენები აგებს პასუხს „შეკვეთა“-ობიექტის მდგომარეობის ცვლილებაზე.



ნახ.12.12. შეკვეთების მენეჯმენტის სისტემის მდგომარეობათა დიაგრამა

ობიექტი („შეკვეთა“) სასცოცხლო ციკლის მანძილზე გაივლის აღნიშნულ მდგომარეობებს, ასევე შესაძლებელია არანორმალური მდგომარეობებიც, რომლებიც სისტემაში არსებული პრობლემების გამო შეიძლება მივიღოთ. როცა მთელი სასიცოცხლო ციკლი დასრულებულია, ითვლება, რომ ტრანზაქცია დასრულდა. აქვე მოცემულია საწყისი და საბოლოო მდგომარეობებიც (Initial state, Final state).

Statechart სქემა განსაზღვრავს კომპონენტის მდგომარეობებს და ეს ცვლილებები დინამიკურია თავისი ბუნებით. ამგვარად, მისი კონკრეტული მიზანია განსაზღვროს მდგომარეობათა ცვლილებები, რომლებიც აქტიურდება მოვლენებით. მოვლენები შეიძლება იყოს შიგა ან გარე ფაქტორები, რომლებიც მოქმედებს სისტემაზე.

Statechart დიაგრამები გამოიყენება მდგომარეობების და სისტემაში მოქმედი მოვლენების მოდელირებისათვის. სისტემის დანერგვის დროს მეტად მნიშვნელოვანია ობიექტის სხვადასხვა მდგომარეობის გარკვევა ამ ობიექტის არსებობის მანძილზე და Statechart დიაგრამებიც სწორედ ამისთვის გამოიყენება.

თუ დავაკვირდებით Statechart დიაგრამის პრაქტიკულ რეალიზაციას, ჩანს, რომ იგი ძირითადად გამოიყენება ობიექტის მდგომარეობათა ანალიზისთვის მოვლენების ზემოქმედების გათვალისწინებით. ეს ანალიზი სასარგებლოა სისტემის ყოფაქცევის გასაგებად მისი შესრულების დროს.

ამგვარად, Statechart -ის ძირითადი გამოყენება შეიძლება ასე აღიწეროს:

- სისტემის ობიექტების მდგომარეობათა მოდელირება;
- რეაქციული სისტემის მოდელირება, რომელიც შედგება რეაქციული ობიექტებისგან;
- მოვლენების იდენტიფიცირება, რომლებიც პასუხისმგებელია მდგომარეობათა ცვლილებებზე;
- პირდაპირი და რევერსიული ინჟინერინგისათვის.

➢ მულტიმოდალური გადაზიდვების პროცესის მდგომარეობათა დიაგრამების დაპროექტება

როგორც აღვნიშნეთ, Statechart დიაგრამის გამოყენება ყველა კლასისთვის არაა საჭირო. იგი აუცილებელია მხოლოდ მაშინ, როდესაც კლასის ობიექტები შეიძლება იმყოფებოდეს რამდენიმე მდგომარეობაში და თითოეულ მათგანში მისი ქცევა იყოს სხვადასხვანაირი. ქვემოთ განვიხილავთ ჩვენი სისტემის მაგალითისათვის რამდენიმე ასეთ შემთხვევას.

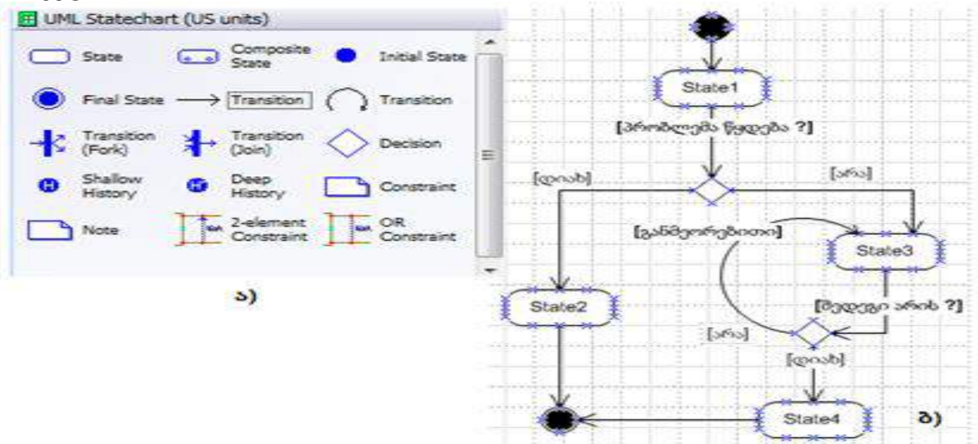
პირველი მათგანი ნორმალური სიტუაციის თანმხლები პროცესებია, როდესაც მულტიგადაზიდვების პროცესები დასაწყისიდან დასასრულამდე ვითარდება სტანდარტულად, ყოველგვარი გამონაკლისის გარეშე. მომდევნო სამი დიაგრამა კი აღწერს მულტიგადაზიდვების პროცესის სხვადასხვა წერტილში შესაძლო მოვლენების წარმოშობას, რომლებიც ცვლის სტანდარტულ პროცესთა თანამიმდევრობას და საჭიროებს დამატებითი სპეციფიკური ოპერაციების განხორციელებას. სასურველია, რომ ასეთი მოვლენები წინასწარ იყოს გამოკვლეული და მისი განეიტრალების მეთოდებიც იქნას შემუშავებული. ასეთი მიდგომა განაპირობებს მოსალოდნელი რისკების შემცირებას და დამატებითი ხარჯების ან საჯარიმო გადასახადების პრევენციულ გადაწყვეტას.

12.2.1. მულტიმოდალური გადაზიდვის სტანდარტული Statechart-მოდელი

ობიექტორიენტირებული მოდელირების მეთოდოლოგიის საფუძველზე მულტიმოდალური გადაზიდვების ბიზნესპროცესების ერთ-ერთი ძირითადი კლასია „**ტვირთი**“ (ნახ.12.5-6). მისი თვისებები ანუ ატრიბუტებია: იდენტიფიკატორი, ტიპი, მდგომარეობა, შეფუთვის ტიპი, ერთეულის ზომები (სიგრძე, სიგანე, სიმაღლე), ერთეულის მოცულობა, ჯამური მოცულობა, ერთეულის წონა, ერთეულის რაოდენობა, ჯამური წონა, უსაფრთხოობა, საბაჟო კოდი, გამგზავნი, მიმღები, საწყისი მდებარეობა, საბოლოო მდებარეობა და სხვ.

როგორც შესავალში აღვნიშნეთ, არსებობს რაღაც ნივთი (საგანი, პროდუქცია ან სხვა), რომელიც ეკუთვნის ფიზიკურ ან იურიდიულ პირს და მას სურს ამ ნივთის გადატანა გეოგრაფიულად ერთი წერტილიდან მეორეში (შეიძლება სხვადასხვა ქვეყნების და კონტინენტების ფარგლებში). ამ პირს ჩვენ ვუწოდეთ კლასი „**კლიენტი**“, რომლის ატრიბუტებია კლ_იდენტიფიკატორი, დასახელება/ვინაობა, იურიდიული/ფიზიკური პირი, მისამართი, ტელეფონი, ელ_მისამართი და სხვ.;

ასევე საჭიროა კლასი „ექსპედიტორი“ და კლასი „ტრანსპორტიორი“ (ანუ გადამზიდავი (ნახ.12.6). აღნიშნული ნივთი (ან ნივთების ერთობლიობა) მიიღებს ტვირთის სტატუსს, როდესაც კლიენტსა და ექსპედიტორს შორის მოლაპარაკების საფუძველზე შედგება „შეკვეთა“ (Order). შემდეგ ექსპედიტორსა და გადამზიდავ ფირმას შორის შეთანხმდება სატრანსპორტო დოკუმენტაცია და, ბოლოს, დაიწყება ტრანსპორ-ტირების პროცესი. იურიდიულად და ორგანიზაციულად ამ პროცესში მონაწილე როლები (კლიენტი, ექსპედიტორი, გადამზიდავი) აგებენ პასუხს თავიანთ ფუნქციებსა და მოვალეობებზე. მდგომარეობათა დიაგრამების ასაგებად ვიყენებთ მაიკროსოფთის Ms Visio პაკეტს (ნახ.12.13).



ნახ.12.13. Statechart-ის პანელი (ა) და მდგომარეობათა დიაგრამის ზოგადი მაგალითი

12.14 ნახაზზე მოცემულია ტვირთების გადაზიდვის პროცესის სტანდარტული მდგომარეობათა დიაგრამა, რომელზეც ასახულია ცალკეული სავალდებულო მიმდევრობითი პროცესების ერთობლიობა. მცირე ზომის რომბის ფიგურებით ვითვალისწინებთ შესაძლო არასტანდარტული მოვლენების (რისკების თვალსაზრისით) აღმოცენების შესაძლებლობას, რომლებსაც ჩვენ მომდევნო პარაგრაფებში დეტალურად განვიხილავთ.

მოცემული მდგომარეობათა დიაგრამა ასახავს ტვირთის - როგორც კლასის ობიექტის მდგომარეობებს, ანუ რა მდგომარეობები უნდა გაიაროს ტვირთმა (ზოგადად) კლიენტის (გამგზავნის) მიერ მოთხოვნის ფორმირებიდან ბოლოს, ტვირთის ჩაბარებამდე მიმღებზე. ეს სტანდარტული ბიზნესპროცესი აგებულია არსებული წესების, მოთხოვნებისა და შეზღუდვების დაცვით, რომლებიც არსებობს საერთაშორისო და კონკრეტული ქვეყნების კანონმდებლობის საფუძველზე.

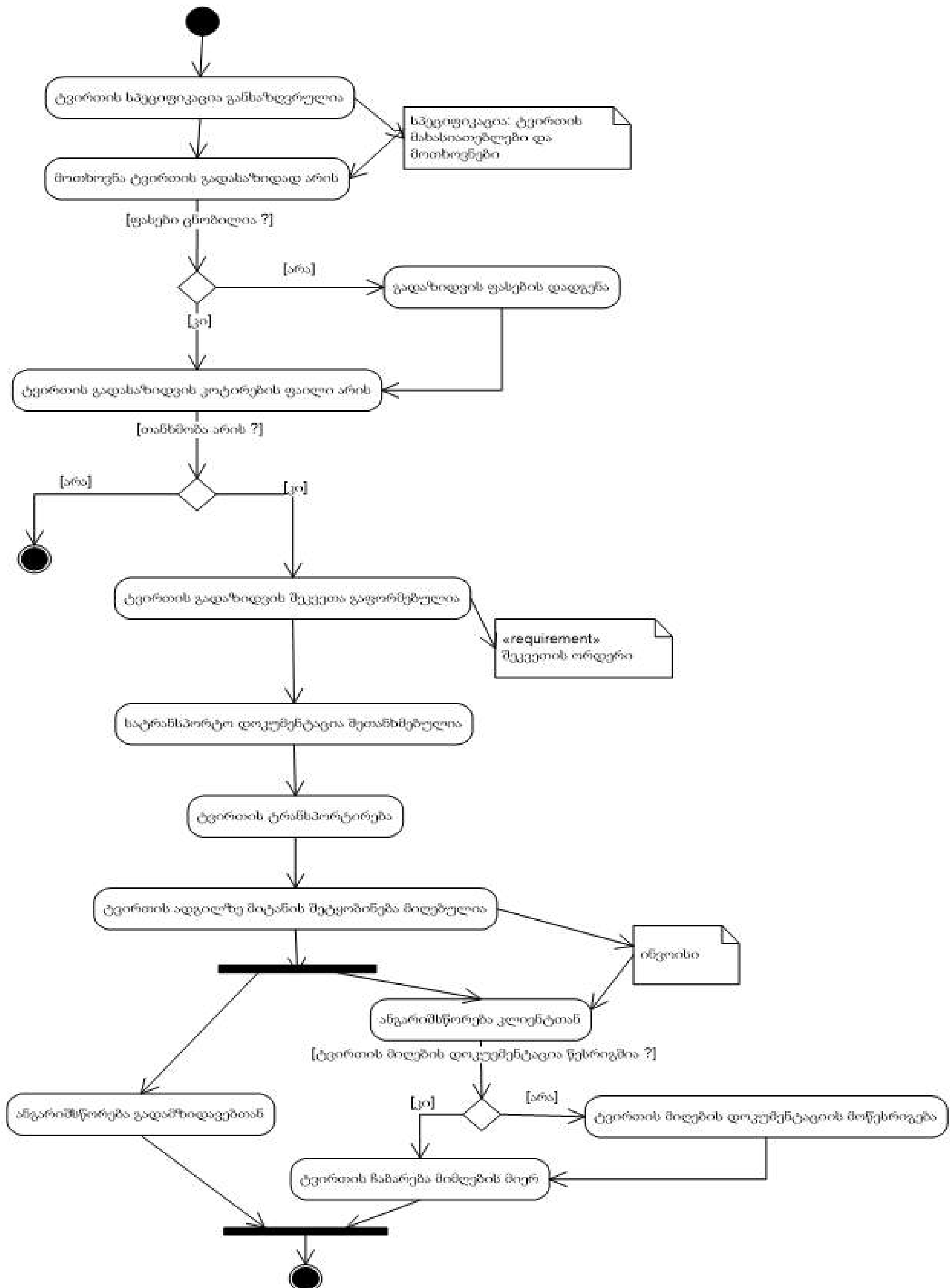
მრგვალკუთხედებში ჩაწერილია კონკრეტული მდგომარეობის ამსახველი ქმედება, მაგალითად: „**ტვირთის სპეციფიკაცია განსაზღვრულია**“, „**ტვირთზე მოთხოვნა მიღებულია**“,. . ., „**ტვირთის გადასაზიდი შეკვეთა გაფორმებულია**“ და ა.შ.

რომბებში აისახება პირობა (ბიზნესწესი), რომლის საფუძველზე მიიღება ერთი ან მეორე გადაწყვეტილება შემდეგი ქმედებისათვის. მაგალითად, თუ გადაზიდვის მარშრუტის ფასები არაა ცნობილი, მაშინ ექსპედიტორი მიმართავს გადამზიდავ ფირმებს ამ ინფორმაციის მოსაპოვებლად და პასუხის მიღების შემდეგ უგზავნის დამკვეთ-კლიენტს „**კოტირების ფაილს**“. თუ კლიენტისთვის მისაღებია ეს ფასები (მეორე რომბი), იგი უგზავნის ექსპედიტორს „**თანხმობას**“, რის საფუძველზეც დგება „**შეკვეთა“**.

შეკვეთის (ორდერის) გაფორმების შემდეგ ფორმდება შეთანხმება ექსპედიტორსა და გადამზიდავ ფირმას შორის ტრანსპორტირების პირობებსა და ვადებზე. ექსპედიტორი ათანხმებს გადამზიდავთან სატრანსპორტო დოკუმენტაციას, რის შემდეგაც გადამზიდავი იბარებს ტვირთს და იწყება ტრანსპორტირების პროცესი.

ტვირთის გადაზიდვის პროცესი შეიძლება იყოს მულტიმოდალური, ანუ საწყის და საბოლოო პუნქტებს შორის მისი გადაზივა მოხდეს რამდენიმე ტრანსპორტის საშუალებით, კერძოდ, ავტოტრანსპორტით, გემით, რკინიგზითა და თვითმფრინავით. ეს ციკლური ოპერაციები, პუნქტიდან პუნქტამდე, უნდა იყო სხელშეკრულებაში დეტალურად აღწერილი, იურიდიული პასუხისმგებლობა ეკისრებათ ამ პროცესში მონაწილეებს. აგრეთვე შესაძლებელია შუალედური საწყობების გამოყენებაც, თუ ამას მოითხოვს გადაზიდვის პროცესი. ყველა შესაძლო ვარიანტი უნდა იქნას წინასწარ გათვალისწინებული, დამატებითი მოსალოდნელი ხარჯები წინასწარ გათვლილი, ვინ უნდა გადაიხადოს ეს ხარჯი ან ჯარიმა და ა.შ. ექსპედიტორის როლი ასეთ შემთხვევბში ძალზე მნიშვნელოვანია.

ახლა განვიხილოთ რამდენიმე არასტანდარტული შემთხვევა, რომელიც შეიძლება აღმოცენდეს ტვირთების გადაზიდვის პროცესში და ისინი უნდა შესრულდეს.



ნახ.12.14. სტანდარტული Statechart დიაგრამა ტვირთის
გადაზიდვის პროცესისათვის

ასეთ შემთხვევებს „მოვლენებს“ (Events) უწოდებენ და ისინი შეიძლება მოხდეს ან არ მოხდეს, საინტერესოა როგორ ვითარდება პოლიტიკური, სტიქიური, ეკოლოგიური ან სხვა პროცესები [116]. გარკვეული რისკების არსებობა არაა გამორიცხული და სასურველია მათი პრევენციის მიზნით გამოყენებული იყოს სხვადასხვა დამცავი მექანიზმი (მაგალითად, ტვირთის დაზღვევა და ა.შ.).

12.2.2. არასტანდარტული Statechart-მოდელი

განვიხილოთ *მოვლენა_1:* **„ფორს-მაჟორული სიტუაცია გადატვირთვის ან დანიშნულების პორტში“** [116].

დავუშვათ, რომ მულტიმოდალური ტრანსპორტირების ამოცანა შემდეგია:

– საზღვაო გადაზიდვა: ჩინეთიდან ფოთის პორტამდე;
– სახმელეთო გადაზიდვა: ფოთი - თბილისი.

კლიენტთან შეთანხმებულია მთლიანი გადაზიდვის ტარიფი ჩინეთი - თბილისი და სატრანზიტო დრო 45 დღე.

ფოთის პორტში ჭარბი შემომავალი ტვირთნაკადის გამო ხდება ტვირთების დაყოვნებით დამუშავება გაურკვეველი დროით (შესაძლებელია არა ტვირთების სიჭარბის, არამედ პორტის თანამშრომლების გაფიცვის გამო).

შესაბამისად დგება ორი რისკი:

1) სატრანზიტო დრო იზრდება ექსპედიტორის საქმიანობისგან დამოუკიდებელი მიზეზით, რაც კლიენტს თბილისში უგვიანებს საქონლის მიღებას;

2) საკონტეინერო ხაზი (გადამზიდავი) არ იღებს პასუხისმგებლობას საკუთარი სატვირთო ტერმინალის ოპერაციულ საქმიანობაზე, თუნდაც ხარვეზიანი იყოს. ის მხოლოდ ითვლის და ანგარიშობს ტერმინალში გაჩერებული კონტეინერების შენახვის საფასურს, რომლის გადახდა შესაძლოა ექსპედიტორს დაეკისროს.

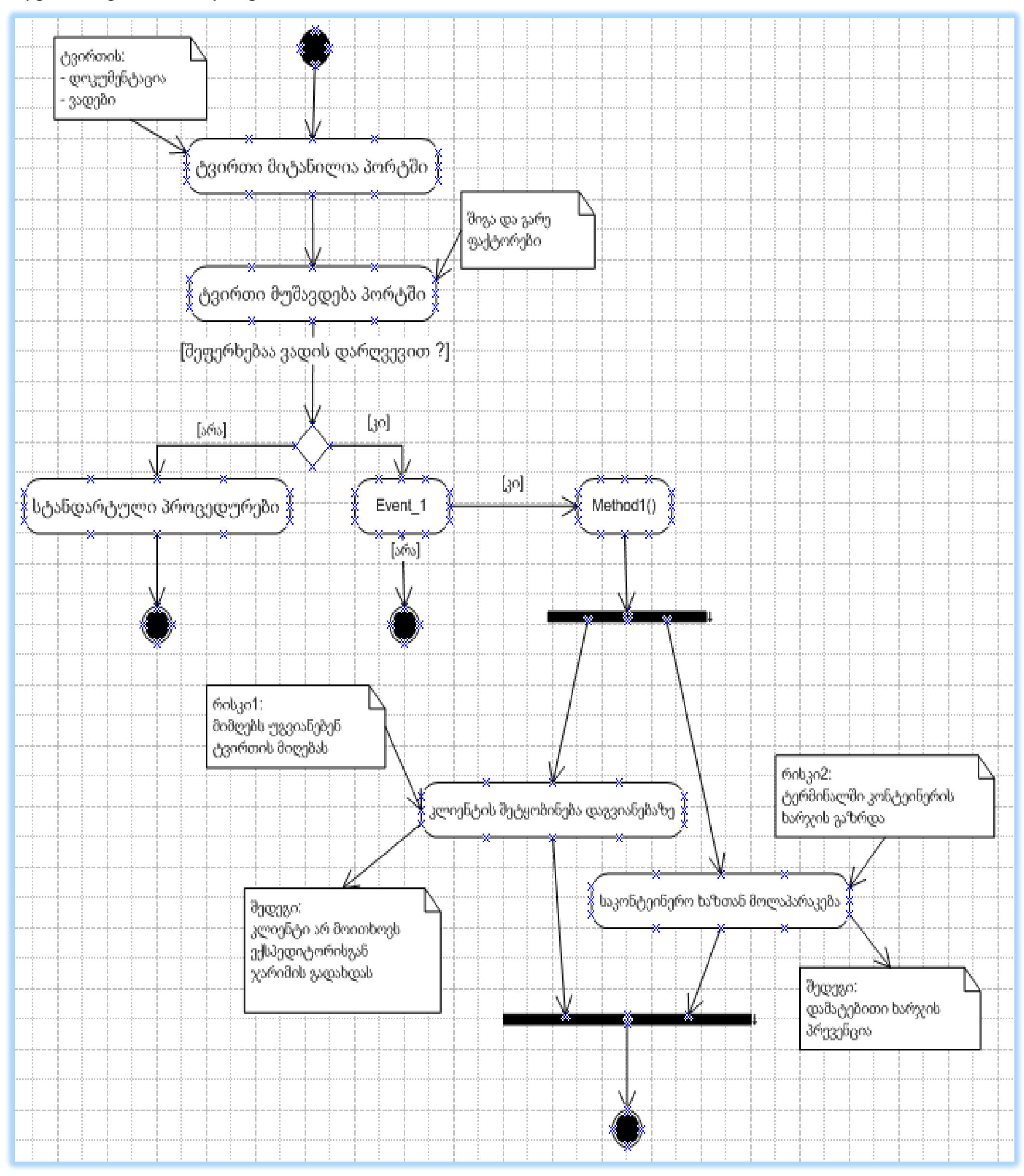
ასეთ შემთხვევებში საჭიროა ორი ქმედების განხორციელება:

1) კლიენტისათვის სასწრაფოდ შეტყობინება დაყოვნების თაობაზე და ექსპედიტორის პასუხისმგებლობის არდაყენების შესახებ მისგან დასტურის მიღება და ბოლოს: 2) საკონტეინერო ხაზთან მოლაპარაკების წარმოება შესაძლო დამატებითი ხარჯის თავიდან აცილებაზე გარანტიის მიღების მიზნით.

<u>შენიშვნა</u>: 1. უნდა გადაიხედოს საკონტეინერო ხაზის შემოთავაზება, მათი გადაზიდვის პირობების ჩათვლით, რომელიც ძალა აქვს საზღვაო გადაზიდვის დასრულებამდე (სანამ ტვირთს ექსპედიტორი გასატანად მიაკითხავს ფოთის პორტში), რათა დავრწმუნდეთ, რომ აღნიშნული პირობები ითვალისწინებს მსგავსი ფინანსური რისკებისაგან ექსპედიტორის დაცვას.

2. გადაიხედოს კლიენტისათვის გამიზნული შეთავაზება (ან ხელშეკრულება), რათა დავრწმუნდეთ, რომ კონკრეტული რისკებისაგან დაცვის მექანიზმი მასში თავიდანვე იყო ჩადებული.

12.15 ნახაზზე მოცემულია განხილული ფორსმაჟორული შემთხვევის მდგომარეობათა დიაგრამა.



ნახ.12.15. პორტში ფორსმაჟორული სიტუაციის მდგომარეობის დიაგრამა

- *მოვლენა_2*: მულტიმოდალური გადაზიდვა გამონაკლისი სიტუაციით

მაგალითად გვაქვს მულტიმოდალური გადაზიდვის შეკვეთა:

საზღვაო + სარკინიგზო გადაზიდვა: ჰამბურგი (გერმანია) - ბიშკეკი (ყირგიზეთი)

კლიენტი ამ შემთხვევაში ყირგიზული კომპანიაა, რომელიც უკვეთავს ქართულ ექსპედიტორულ კომპანიას მთლიან გადაზიდვას.

მოხდა ისე, რომ ბიშკეკში ჩასული სარკინიგზო ტვირთი, რომლის იმპორტულ რეჟიმში მოქცევა ევალება ქართული ექსპედიტორული კომპანიის პარტნიორ კომპანიას ყირგიზეთში, უკავშირდება ექსპედიტორს და ატყობინებს რომ აღმოჩნდა ტვირთმიმღებს (კლიენტს) არ აქვს მზად (ანუ არ აუღია ჯერ) გარკვეული სახის ნებართვა სახელმწიფო უწყებიდან ტვირთის იმპორტირების თაობაზე.

ეს ნიშნავს, რომ ვაგონები დაყოვნდება დანიშნულების სადგურზე. ჩნდება დამატებითი შეკითხვები:

- უნდა მოხდეს თუ არა ტვირთის გადაცლა ვაგონებიდან დროებითი შენახვის საწყობში, სადაც ის გაჩერდება მანამ, სანამ მიმღები არ გადასცემს ნებართვას ყირგიზულ ექსპედიტორულ კომპანიას;

- თუ დაშვებულ იქნას ვაგონების მოცდენა, რაც რკინიგზისათვის დამატებით გადასახადს ნიშნავს.

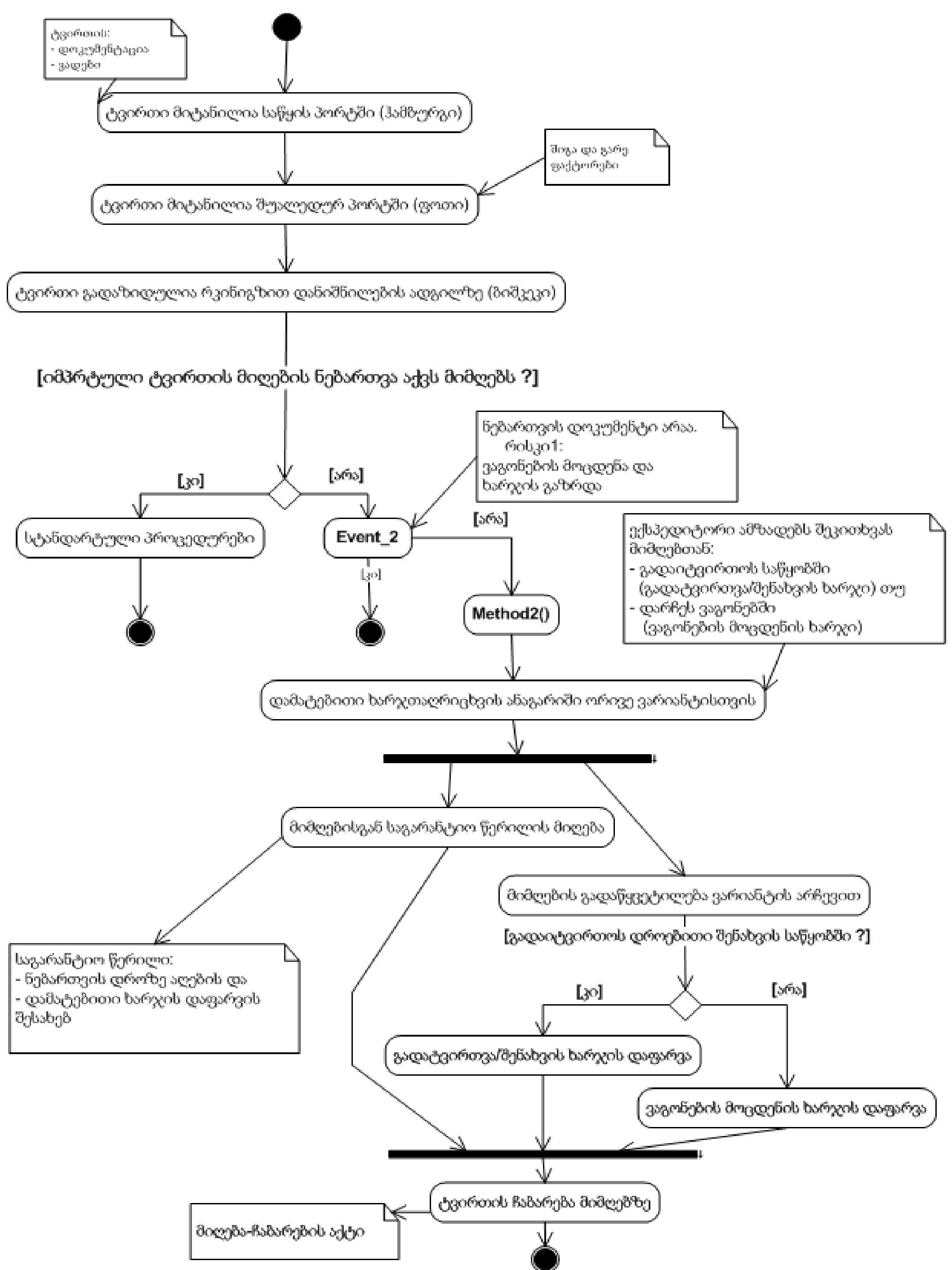
საჭირო ღონისძიებები:

- მიმღებისაგან მიღებული უნდა იქნას წერილობითი გარანტია, რომ დროულად უზრუნველყოფს საბუთების მიწოდებას და რომ ყველა დაკავშირებულ ხარჯს საკუთარ თავზე აიღებს;

- ასევე ტვირთმიმღებმა უნდა გადაწყვიტოს და დაადასტუროს, რომელი ვარიანტი ურჩევნია. გადმოცლა თუ ვაგონების მოცდენა;

- ხარჯთაღრიცხვა მას ქართულმა ექსპედიტორმაუნდა წარუდგინოს, რათა მიმღებმა შეძლოს შედარება, რომელი ვარიანტი უფრო ოპტიმალურია მისთვის.

12.16 ნახაზზე მოცემულია განხილული შემთხვევის მდგომარეობათა დიაგრამა.



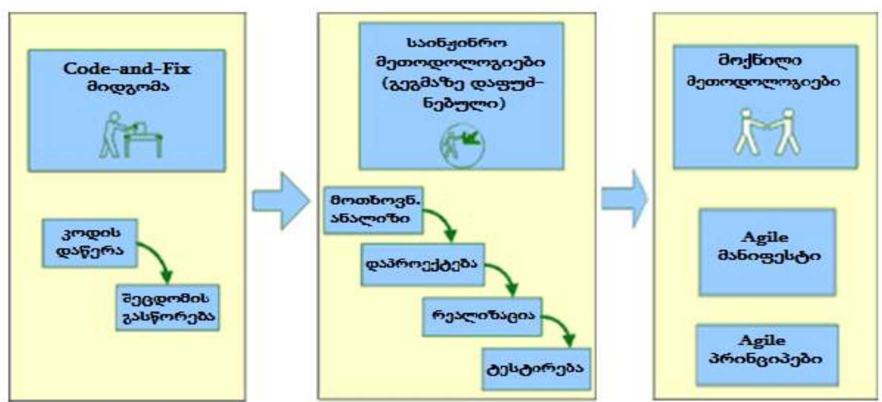
ნახ.16. მდგომარეობათა დიაგრამა გამონაკლისი შემთხვევით.
მოვლენა: „მიმღებს არ აქვს იმპორტული ტვირთის მიღების ნებართვის საბუთი“

XIII თავი
მართვის საინფორმაციო სისტემების აგება Agile ტექნოლოგიით

13.1. ექსტრემალური დაპროგრამების პრინციპები და ინსტრუმენტული საშუალებანი

2001 წლის თებერვლიდან ოფიციალურად იქნა ჩამოყალიბებული პროგრამული სისტემების დამუშავების ახალი, Agile - მეთოდოლოგია [13]. ადრინდელი ქაოსური მიდგომების (code-and-fix) და მკაცრად ფორმალიზებული საინჟინრო მეთოდოლოგიების ნაცვლად განვითარება დაიწყო მოქნილი მეთოდოლოგების ინდუსტრიამ, რომელიც დაფუძნებული იყო პროგრამული უზრუნველყოფის დამუშავების მანიფესტსა და მისი რეალიზაციის პრინციპებზე (ნახ.27) [19, 24].

ფორმალიზაციის ხარისხის ნიხედვით მოქნილი მეთოდოლოგიები იკავებს შუალედურ ადგილს წინა ორ განხილულ მიდგომას შორის, ანუ, ისინი არც ისე მკაცრად ფორმალიზებულია, როგორც საინჟინრო მიდგომები და არც ქაოსური, უსისტემო მიდგომაა, როგორც code-and-fix



ნახ.13.1. Agile მეთოდოლოგიებზე გადასვლა

- **მოქნილი დამუშავების მანიფესტი (Agile Manifesto) და პრინციპები**

მანიფესტი ოთხი პუნქტისაგან შედგება, რომელთაგანაც თითოეული ალტერნატივაა [24]. მის მარცხენა ნაწილში მოთავსებულია ცნებები და ასპექტები, რომლებსაც პროგრამული უზრუნველყოფის დამუშავებისას დიდი ღირებულება აქვს, ვიდრე მარჯვენა ნაწილში. მოქნილი მოდელირების ძირითადი კონცეფციები ასეთია:

- ადამიანები და ურთიერთობები უფრო მნიშვნელოვანია, ვიდრე პროცესები და ინსტრუმენტები;
- სამუშო პროდუქტი უფრო მნიშვნელოვანია, ვიდრე ვრცელი-ამომწურავი დოკუმენტაცია;

- დამკვეთთან თანამშრომლობა უფრო მნიშვნელოვანია, ვიდრე კონტრაქტით შეთანხმებული პირობები;
- მზადყოფნა ცვლილებებისადმი უფრო მნიშვნელოვანია, ვიდრე პირველსაწყისი გეგმის დაცვა.

მოქნილი დამუშავების პრინციპები ბაზირებულია Agile მანიფესტის ფასეულობებზე, დეტალურად ხსნის და განავრცობს მათ მეტი პრაქტიკული თვისებების მქონე ინფორმაციით. ეს პრინციპებია:

1. კლიენტის დაკმაყოფილება ღირებული პროგრამული უზრუნველყოფის ადრეული და უწყვეტი მიწოდების მეშვეობით;
2. მოთხოვნილებათა ცვლილებების მისალმება სამუშაოს ბოლო ეტაპზეც კი (ეს ზრდის საშედეგო პროდუქტის კონკურენტუნარიანობას);
3. სამუშაო პროგრამული უზრუნველყოფის ხშირი მიწოდება დამკვეთზე (ყოველთვიური, კვირეული ან უფრო ხშირი);
4. დამკვეთის ხშირი, ყოველდღიური კონტაქტი მიმწოდებელთან პროექტის შესრულების მთელ მანძილზე;
5. პროექტზე მუშაობენ მოტივირებული პირები, რომლებიც უზრუნველყოფილია მუშაობის საჭირო პირობებით, მხარდაჭერითა და ნდობით;
6. ინფორმაციის გადაცემის რეკომენდებული მეთოდი - პირადი საუბრები (პირისპირ);
7. **მომუშავე პროგრამული უზრუნველყოფა - პროგრესის საუკეთესო საზომია**. პროექტების მიზანია პროგრამული სისტემის შექმნა და არა გეგმებისა და დოკუმენტაციის. აპლიკაციის მუშაობისუნარიანობის შეფასებით შეიძლება პროექტის პროგრესის ობიექტური გაზომვა;
8. სპონსორებს, მიმწოდებლებსა და მომხმარებლებს უნდა ჰქონდეთ გაურკვეველი ვადით მხარდაჭერის შესაძლებლობა მუდმივი ტემპის შესანარჩუნებლად;
9. მუდმივი ყურადღება სჭირდება ტექნიკური ოსტატობის (უნარების) სრულყოფას და მოსახერხებელ დიზაინს;
10. **სიმარტივე - ხელოვნება ზედმეტი სამუშაოს შესრულების გარეშე.** არაა საჭირო რთული უნივერსალური გადაწყვეტების მიღება, თუ ამის ცხადი აუცილებლობა არაა;
11. თვითორგანიზებულ გუნდს საუკეთესო ტექნიკური მოთხოვნები, დიზაინი და არქიტექტურა გამოსდის;
12. **მუდმივი ადაპტაცია ცვალებადი გარემოებებისადმი**. იტერაციული სასიცოცხლო ციკლი ბაზირებულია მართვაზე უკუკავშირით, რომლის მნიშვნელოვანი ელემენტია შედეგების ანალიზი, უკუკავშირის განხორციელება და პროცესის სრულყოფა.

მოქნილი დამუშავების მანიფესტი და პრინციპები მოიცავს მაღალი დონის კონცეფციებს იმის შესახებ თუ როგორ უნდა განხორციელდეს პროგრამული უზრუნველყოფის დამუშავების პროცესი, რათა წარმატებით დასრულდეს პროექტი,

შეიქმნას მუშა გუნდები, რომლებშიც სასიამოვნო და საინტერესო იქნება მუშაობა. ეს დოკუმენტები აღწერს, თუ რა უნდა გაკეთდეს ამისათვის, მაგრამ არაფერს ამბობს, თუ როგორ.

13.2. მოქნილი (სწრაფი) მოდელირება (Agile Modeling)

Agile Modeling (AM) – არის პროგრამული უზრუნველყოფის (Software) შექმნის სპეციალისტების ერთობლივი მუშაობის ეფექტური ორგანიზების ხერხი დამკვეთების მოთხოვნილებათა დასაკმაყოფილებლად. მოქნილი მეთოდებით პროგრამული სისტემების დამუშავების სპეციალისტები ქმნიან ერთ გუნდს დამკვეთთან ერთად, რომლის წარმომადგენლებიც უშუალოდ და აქტიურად მონაწილეობენ სისტემის ანალიზის, დაპროექტებისა და აგების პროცესებში. AM-გუნდის მუშაობის მთავარი მიზანია ეფექტურობა, დამკვეთის მეტი წვლილის ჩადება საბოლოო პროდუქტში, შეძლებისდაგვარად მარტივი მოდელების აგება, სამუშაო სისტემის შექმნა.!

ამგვარად, მოქნილი მოდელირება ესაა პროფესიონალთა გუნდის მუშაობის ეფექტურობის ამაღლების მეთოდოლოგია პროგრამული უზრუნველყოფის შესაქმნელად.

AM ითვალისწინებს აგრეთვე CASE-საშუალებების ზომიერად გამოყენებასაც, თუკი ამით ეფექტურობა მაღლდება. ინსტრუმენტული საშუალებებიდან შეირჩევა უმარტივესი, რომელიც დასმული ამოცანის გადაწყვეტის საშუალებას იძლევა.

AM-ის ფასეულობებია [14]: ურთიერთობა (კომუნიკაცია), სიმარტივე, უკუკავშირი, გამბედაობა და თავმდაბლობა.

აქედან პირველი ოთხი „ექსტრემალური დაპროგრამების" მამას, კენტ ბეკსაც ჰქონდა მოცემული (2000 წ. XP) [13]. ს. ამბლერმა მეხუთე დაამატა [14]. განვიხილოთ თითოეული მათგანი მოქნილი მოდელირების თვალსაზრისით.

- **კომუნიკაცია (ურთიერთობა)**

ტერმინოლოგიური ლექიკონის მიხედვით ესაა „პროცესი, რომლის დროსაც ხდება ინფორმაციის გადაცემა ერთი პირიდან მეორეზე, არსებული სიმბოლოების, ნიშნების ან ქმედებების საერთო სისტემის საშუალებით". პროგრამული პროექტის შესრულების დროს განსაკუთრებული მნიშვნელობა აქვს ურთიერთობას პროექტის მონაწილეებს შორის, კერძოდ, ეფექტურ კომუნიკაციას დამპროექტებლებს, პროგრამისტებსა და დამკვეთებს შორის.

პროექტის პრობლემები ჩნდება იქ, სადაც ურთიერთობა შეწყვეტილია. მაგალითად, როდესაც დეველოპერი არ ეუბნება კოლეგებს, რომ მისი კოდი არ მუშაობს და საჭიროა დახმარება, ან დამკვეთი ვერ ამახვილებს ყურადღებას პროექტის მნიშვნელოვან მახასიათებლებზე და დეველოპერები მეორეხარისხოვანი საკითხებითაა დაკავებული.

ასეთი საკითხები პრობლემების სახით ბოლოს მაინც გამოჩნდება რაც მოითხოვს დამატებით დროს და სხვა რესურს მათ გადასაწყვეტად.

მოდელირების პროცესის სიკეთე ისიცაა რომ იგი აადვილებს კომუნიკაციას პროექტში მონაწილე პიროვნებებს შორის. მაგალითად, როცა დამკვეთი შინაარსობრივად აღწერს რთულ ბიზნესპროცესს, ჩვენ შეგვიძლია იგი ავსახოთ მონაცემთა ნაკადების დიაგრამის სახით, რაც მის ბიზნესლოგიკას შეესაბამება. ამ დროს დამკვეთს უადვილდება პროცესის უკეთ აღქმა და ხშირად ამ პროცესის სრულყოფის საფუძველიც ხდება (მაგალითად, პროცედურების სიჭარბის აღმოფხვრის თვალსაზრისით).

ამგვარად, ურთიერთობისას ხუთ წუთში შეიძლება მეტი ინფორმაციის მიღება, ვიდრე კორპორაციული დოკუმენტების 5 საათიანი კითხვისას.

ასევე შესაძლებელია დეველოპერებს შორის კლასთა სტრუქტურის უკეთ გასაგებად UML დიაგრამების გამოყენება. ეს კი აუცილებელია გუნდის წევრებს შორის ერთიანი კონცეფციის არსებობისა და მეგობრული დამოკიდებულებისათვის.

- სიმარტივე

პროგრამული უზრუნველყოფის წარმოების ინდუსტრიის მთელი არსებობის მანძილზე მიეთითება, რომ სისტემა იყოს შეძლებისდაგვარად მარტივი, მაგრამ, სამწუხაროდ ეს პირობა ხშირად ვერ სრულდება. რაც იწვევს პროგრამის რეალიზაციის, ტესტირებისა და ექსპლუატაციის პროცესების გართულებას. ყველაზე ხშირად პროგრამის გართულების გამომწვევი მოვლენებია:

- **რთული შაბლონების (Pattern) ხშირი გამოყენება**. საერთოდ არსებული შაბლონის გამოყენება ახალი პროგრამული სისტემის ასგაებად ერთ-ერთი მარტივი და შესაძლებელი ხერხია (პროტოტიპების თვალსაზრისით), მაგრამ დასმული ამოცანისთვის ის უნდა იყოს შესაბამისად მისაღები, (შეიძლება, ძნელი ვიდრე ამას ამოცანა მოითხოვს);
- **სისტემის არქიტექტურის გართულება მომავალი შესაძლო გაფართოებების მხარდასაჭერად**. ხშირად ტრადიციული პროგრამული ტექნოლოგიების მიმდევრები, რომელთაც არ სურთ მომავალში ცვლილებების განხორციელება, ცდილობენ წინასწარ გაითვალისწინონ და დააპროექტონ სისტემის ჭარბი, გაფართოებული ვარიანტი (თუმცა, შეიძლება ასეთი მოთხოვნილება მომავალში ნაკლებად ძნელი იყოს, ან საერთოდ არ იყოს საჭირო). მოქნილი მოდელირების მიმდევრები არ ქმნიან ჭარბ პროგრამულ სისტემას, მათ განკარგულებაშია შეძლებისდაგვარად მარტივი პროდუქტი, დღეისათვის აუცილებელი ფუნქციური. თუ საჭირო გახდება სისტემის გაფართოება, მხოლოდ მაშინ დაუმატებენ ისინი არსებულ სისტემას ახალ, ან შეძლებისდაგვარად მარტივ ფუნქციონალობას. გამოიყენება პრინციპი „ხვალის პრობლემა გადაწყდეს ხვალ“. ამ დროს გაფართოების მიზანი ცალსახად იქნება გარკვეული, რაც გამორიცხავს სიჭარბის შექმნას და სისტემის ზედმეტად გართულებას.

- **რთული ინფრასტრუქტურის შემუშავება**. ეს ფართოდ გავრცელებული შეცდომაა, რომელსაც გუნდი უშვებს. კერძოდ, გუნდის საწყისი ძალისხმევა მიმართულია პროექტის ინფრასტრუქტურის შექმნაზე (მაგალითად, კომპონენტები, კლასების ბიბლიოთეკა და სხვ), და არა სისტემის ცალკეული ბლოკების აგებაზე. ნაკლოვანება ისაა, რომ სისტემაში თავიდან ჩაიდება დამკვეთის საკმაო რესურსი და ამავე დროს მათ არ წარედგინებათ არავითარი შედეგი, რომელსაც ისინი გამოიყენებდნენ ოპერატიულად. დამკვეთს უნდა მიიღოს მიმწოდებლისგან კონკრეტული პროდუქტი, რომელიც მას დაეხმარება სამუშაოს შესრულებაში (და არა ცარიელი მონაცემთა ბაზების სისტემა ან შეცდომების დამუშავების ქვესისტემა). ვინაიდან ვერ ხერხდება საჭირო ფუნქციონალობის პროგრამების სწრაფი დამუშავება, ეს წარმოშობს გარკვეულ რისკს პროექტის შესასრულებლად. საუკეთესო მიდგომაა - შემცირდეს ინფრასტრუქტურის დამუშავების მასშტაბები პროექტის შესრულებისას მანამ, სანამ ის არ გახდება აუცილებელი. მაგალითად, შეცდომების გასწორების ქვესისტემა შეიძლება დამუშავდეს მოგვიანებით, როცა ის აუცილებელი გახდება.

მოქნილი მოდელირების დროს ძირითადი დებულება მდგომარეობს იმაში, რომ მოდელები შეძლებისდაგვარად მარტივად იქნას შენარჩუნდეს, დღეს მოხდეს იმის მოდელირება, რაც დღესაა საჭირო, და ხვალისა შესრულდეს ხვალ. მოდელები თამაშობს ძირითად როლს პროგრამებისა და მათი შექმნის პროცესების გასამარტივებლად. განსაკუთრებით მარტივია იდეის შემუშავება და ამოცანის გაგება ერთი-ორი დიაგრამის დახაზვით, ვიდრე კოდის ასობით სტრიქონის დაწერით.

- **უკუკავშირი**

არის თუ არა სამუშაო შესრულებული სწორად ერთადერთი ხერხია მასზე გამოძახილის (შეფასების, რეცენზიის) მიღება. მოდელის სისწორის შეფასებაც უნდა მოხდეს ასეთივე ხერხით, რომელიც განიხილება როგორც უკუკავშირი. მოდელი არის აბსტრაქცია. მაგალითად, UseCase ელემენტების ერთობლიობა - სისტემასთან მუშაობის ხერხების აბსტრაქციაა, ხოლო Component-ების მოდელი - პროგრამული უზრუნველყოფის შინაგანი სტრუქტურის აბსტრაქცია. როგორ უნდა დადგინდეს, არის თუ არა მიღებული აბსტრაქცია მართებული? არსებობს ხერხების სიმრავლე, რომლებიც უზრუნველყოფს მოდელებისათვის უკუკავშირს:

- **მოდელი მუშავდება გუნდში**. აქ გამოძახილი მიიღება ოპერატიულად, სწრაფად;
- **მოდელის განხილვა ხდება მიზნობრივ აუდიტორიასთან.** მოთხოვნილებათა მოდელირება უნდა შესრულდეს დამკვეთის მომხმარებლებთან ერთად, რომლებიც ბოლოს იმუშავებენ ამ სისტემასთან. დაპროექტების დეტალური მოდელები კი უნდა შემუშავდეს დეველოპერ-პროგრამისტებთან ერთად. თუ ეს არ ხერხდება, მაშინ ყოველი შემუშავებული მოდელი სისტემური ანალიტიკოსის მიერ განიხილება პროგრამისტებთან ერთად, ასევე სასურველია დაიწეროს სცენარები თითოეულის გამოყენების მიზნით. შესაძლებელია ასევე არაფორმალური განხილვის (დისკუსიის)

მოწყობა, მაგალითად, სპეციალის-ექსპერტთან, რომლებიც შემდეგ მოგვცემს გამოძახილს.

- **მოდელის რეალიზაცია.** ესაა ყველაზე საიმედო ხერხი გამოძახილის მისაღებად ანუ, მოდელი მუშავდება პროგრამის სახით და მიეწოდება დამკვეთ-მომხმარებელს სამუშაოდ;
- **ტარდება მიღება-ჩაბარების ტესტირება**. მოდელი უნდა ასახავდეს დამკვეთის მოთხოვნებს სისტემასთან. მიღება-ჩაბარების ტესტირების დროს დამკვეთი ამოწმებს თავისი მოთხოვნების სისწორეს.

საინტერესოა განხილულ იქნას დროთი დანახარჯები გამოძახილის მიღების თითოეული ვარიანტისათვის. როცა მოდელი მუშავდება გუნდში, გამოძახილი მიიღება მყისიერად, წამებში ან წუთებში. არაფორმალური განხილვისას შედეგი შეიძლება მივიღოთ რამდენიმე წუთის ან საათის განმავლობაში. ფორმალური განხილვები შეიძლება გადაიდოს რამდენიმე დღით, კვირით ან თვით, მონაწილეებისაგან დამოკიდებულებით. მოდელის რეალიზაციის დროს პროგრამის საშუალებით შედეგები მიიღება სწრაფად, რამდენიმე წუთში, საათში ან დღეში. მიღება-ჩაბარების ტესტირება მოითხოვა რამდენიმე კვირას ან თვეს.

დროითი ხარჯები მნიშვნელოვანია, რადგან რაც უფრო სწრაფად მიიღება გამოძახილი მოდელის შესახებ, მით უფრო ადვილია არსებული მოდელის ცვლილება დამკვეთის მოთხოვნილებათა შესაბამისად.

უპირატესობა უნდა მიენიჭოს მოდელების შემუშავებას გუნდურად და მათ პროგრამულ რეალიზაციას, რადგან ქაღალდზე შეიძლება მოდელი ლამაზად გამოიყურებოდეს, მაგრამ მისი რეალიზაციის შედეგი არ მუშაობდეს.

- **გამბედაობა**

მოქნილი მოდელირება, ან ზოგადად პროგრამული უზრუნველყოფის მოქნილი დამუშავება შედარებით ახალი მეთოდოლოგიაა და იგი უპირისპირდება ტრადიციულ, უკვე კარგად დამკვიდრებულ მეთოდოლოგიებს და მათ მიმდევრებს. ამიტომაც ამ ახალი მიმართულების გამტარებლებს დიდი გამბედაობა და რთულ წინააღმდეგობათა გადალახვა სჭირდებათ. გამბედაობა არის პროგრამების მოქნილი (სწრაფად) დამუშავების უცილებელი კომპონენტი.

უპირველეს ყოვლისა, გამბედაობაა საჭირო, რათა მიღებულ იქნას გადაწყვეტილება მოქნილი მიდგომის გამოყენების შესახებ. შემდეგ კი მისი გამოყენების გაგრძელების შესახებ, თუ საქმე ვერ წავიდა წარმატებით (რაც ხშირად მოსალოდნელია). ორგანიზაციაში მოიძებნება სხვა შეხედულების ადამიანები, რომელთა წინააღმდეგობა დასაძლევია. ეს პოლიტიკაა.

მეორე მხრივ, პროგრამული სისტემის დამუშავებისას საჭიროა გამბედაობა მნიშვნელოვანი გადაწყვეტილების მისაღებად, კერძოდ, რომელი არქიტექტურული გადაწყვეტა ან რომელი დაპროგრამების ენა შეირჩეს. მუშაობის პროცესში საჭიროა

გამბედაობა, რათა თუ საჭიროა, შეცვლილ იქნეს მიმართულება, უარი ითქვას შესრულებულზე ან ჩატარდეს რეფაქტორინგი, თუ ზოგიერთი გადაწყვეტის შედეგი არასწორი აღმოჩნდა.

მესამე მხრივ, საჭიროა გამბედაობა, რათა გაგებულ იქნას, რომ არ ვართ იდეალური და შეგვიძლია შეცდომების დაშვებაც. გამბედაობაა საჭირო, რათა გვწამდეს, რომ ხვალის ამოცანების გადაწყვეტას ხვალვე შევძლებთ.

- **თავმდაბლობა**

პროგრამული უზრუნველყოფის კარგ დეველოპერებს აქვთ საკმარისი თავმდაბლობა, რომ შეიმეცნონ, რომ მათ არაფერი არ იციან. მოქნილი მოდელირების კონცეფციით მომუშავე სპეციალისტებმა კარგად იციან, რომ მათი კოლეგები და დამკვეთებითავიანთ სფეროებში არიან ექსპერტები, აქვთ ცოდნა, ანუ ღირებული ინფორმაცია, რომელიც საჭიროა პროექტისათვის. მაგალითად, არსებობენ დეველოპერები, რომლებიც უკეთესად ქმნიან კოდს, ან ატესტირებენ პროგრამებს, უკეთესად ამოდელირებენ მოთხოვნილებებს ან ქმნიან არქიტექტურებს. მომხმარებლები უკეთესად ერკვევიან თავიანთ ბიზნესპროცესებში. ორგანიზაციის ხელმძღვანელებს უკეთესად ესმით თავიანთი სფეროს განვითარების ტენდენციები, ხოლო თანამშრომლებს კარგად ესმით, რისი გაკეთების უფლება აქვთ და რისი არა საკუთარ პროდუქციასთან. მოქნილი მოდელირების სპეციალისტს უნდა ჰქონდეს საკმარისი თავმდაბლობა თავისი სამუშაოს შესასრულებლად, მას სჭირდება დახმარება და უნდა ითანამშრომლოს ამ ადამიანებთან. თავმდაბლობა ადამიანის ხასიათის თვისებაა (დიპლომატიურობაა), რომლის წყალობითაც ის კომუნიკაბელურია და მეტ ინფორმაციას იღებს ადამიანებთან ურთიერთობისას, რაც ამაღლებს მის მწარმოებლურობას და, ე.ი. პროექტის შესრულების წარმატებას.

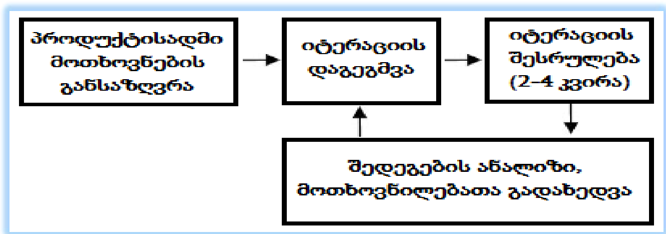
ამგვარად, პროგრამული სისტემის მენეჯერი, კონკრეტული პროექტის ამოცანებისა და მოთხოვნების შესაბამისად, უნდა განსაზღვრავდეს როგორც პროგრამირების მეთოდის, ეტაპთა ფაზების და იტერაციათა მოთხოვნების შერჩევა-ფორმირებას, ისე სამუშაო გუნდის შემადგენლობას.

13.3. Scrum - მოქნილი მეთოდის მაგალითი

პროგრამული ინდუსტრიის სფეროში Agile მეთოდოლოგიის მნიშვნელოვანი წარმომადგენელია Scrum მეთოდი [19,29]. იგი პირველად იაპონელებმა მოიხსენიეს, როგორც ახალი მიდგომა ახალი სერვისების და პროდეუქტების დასამუშავებლად (არა მხოლოდ პროგრამული პროდუქტებისათვის) [30]. მეთოდის ძირითადი არსი მდგომარეობდა მცირე ზომის უნივრსალური გუნდის შეკრულ, თანამიმდევრულ მუშაობაში, რომელიც ამუშავებს პროექტის ყველა ფაზას. სიმბოლურ ანალოგიას ავლებენ „რაგბის" თამაშთან, როდესაც ერთიანი გუნდი მოძრაობს წინ და უკან, ბურთის გადაცემის შესაბამისად.

Scrum - ნიშნავს შეჭიდებას (схватка).

ზოგადი აღწერა. Scrum მეთოდი საშუალებას იძლევა პროექტები დამუშავდეს მოქნილად (სწრაფად) მცირე გუნდის მიერ (5-9 კაცი გუნდში). დამუშავების პროცესი იტერაციულია და დიდ თავისუფლებას აძლევს გუნდს. ამასთანავე, მეთოდი ძალზე მარტივია და შესასწავლად ადვილი, ამიტომაც პრაქტიკაში ადვილად გამოყენებადია (ნახ.13.2).



ნახ.13.2. scrum მეთოდის ბიჯები

თავიდან განისაზღვრება მოთხოვნები მთლიანი პროდუქტისთვის. შემდეგ ამოირჩევა მათგან ყველაზე აქტუალური და შეიქმნება პირველი (მომდევნო) იტერაციის გეგმა. იტერაციის პერიოდში გეგმა არ იცვლება (ეს ხელს უწყობს დამუშავების პროცესის სტაბილობას), მისი ხანგრძლიობა 2-4 კვირაა. იტერაცია სრულდება პროდუქტის სამუშაო ვერსიის შექმნით, რომელიც შეიძლება გადაეცეს დამკვეთს, მოხდეს მისი დემონსტრირება, თუნდაც მინიმალური ფუნქციური შესაძლებლობებით.

ამის შემდეგ ხდება შედეგების განხილვა და პროდუქტისადმი მოთხოვნილებათა კორექტირება. ეს მოსახერხებელია, რადგან არა მხოლოდ ზუსტდება პროდუქტის ფუნქციები, არამედ დამკვეთს შეუძლია მისი გამოყენებაც. შემდეგ იგეგმება ახალი იტერაცია და ყველაფერი მეორდება.

იტერაციის შიგნით მთლიანად მუშაობს გუნდი. Scrum აქ როლებს არ განსაზღვრავს. მენეჯმენტთან და დამკვეთთან სინქრონიზაცია ხდება იტერაციის დასრულების შემდეგ. იტერაცია შეიძლება შეწყდეს მხოლოდ განსაკუთრებულ შემთხვევებში.

როლები. Scrum მეთოდში არის სამი როლი:

- პროდუქტის მფლობელი (Product Owner) - ესაა პროექტის მენეჯერი, რომელიც წარმოადგენს დამკვეთის ინტერესებს. მისი მოვალეობაა პროდუქტის საწყისი მოთხოვნების (Product Backlog) განსაზღვრა, მათი დროულად კორექტირება, პრიორიტეტების, ჩაბარების ვადების დადგენა და სხვ. იგი უშუალოდა რ მონაწილეობს იტერაციის შესრულებაში;
- Scrum-ოსტატი (Scrum Master) - უზრუნველყოფს გუნდის მაქსიმალურ მწარმოე-ბლურობას და პროდუქტიულობას, როგორც Scrum-პროცესის შესასრულებლად, ასევე

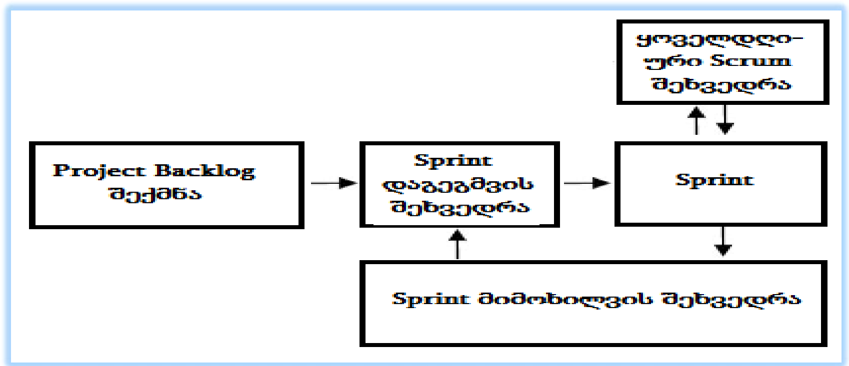
სამეურნეო და ადმინისტრაციული ამოცანების გადასაწყვეტად. კერძოდ, მისი ამოცანაა იტერაციის დროს ყოველგვარი გარე ზემოქმედებიდან გუნდის დაცვა;

- Scrum-გუნდი (Scrum Team) - ჯგუფია, რომელიც შედგება 5-9 დამოუკიდებელი, ინიციატივიანი პროგრამისტების. გუნდის პირველი ამოცანაა იტერაციისათვის რეალურად მიღწევადი და პროექტისთვის პრიორიტეტული დავალებების განსაზღვრა (Project Backlog-ის საფუძველზე და პროდუქტის მფლობელის და Scrum-ოსტატის აქტიური მონაწილეობით). მეორე ამოცანაა ამ დავალებების უეჭველი შესრულება დადგენილ ვადებში და მოთხოვნილი ხარისხით. მნიშვნელოვანია, რომ გუნდი თვითონ მონაწილეობს დავალებათა დასმის პროცესში და თვითონ წყვეტს მათ. აქ შეთავსებულია თავისუფლება და პაუხისმგებლობა, კარგად აისახება მოვალეობათა დისციპლინა.

პრაქტიკები. Scrum-ში განსაზღვრულია შემდეგი პრაქტიკები:

- Sprint Planning Meeting. შეხვედრა Sprint-ის დასაგეგმად (ესაა მოკლე დისტანცია, ანუ იტერაცია). თავიდან პროდუქტის მფლობელი, Scrum-ოსტატი, გუნდი, ასევე დამკვეთის წარმომადგენელი და სხვა დაინტერესებული პირები განსაზღვრავენ, თუ რომელი მოთხოვნებია Project Backlog-დან უფრო პრიორიტეტული და რომელი უნდა განხორციელდეს მოცემული სპრინტის ფარგლებში. ფორმირდება Sprint-Backlog. შემდეგ Scrum-ოსტატი და Scrum-გუნდი განსაზღვრავენ, თუ როგორ უნდა იქნას მიღწეული დასმული მიზნები Sprint-Backlog-დან. მისი ყოველი ელემენტისათვის დადგინდება ამოცანათა სია და შეფასდება მათი შრომატევადობა.
- Daily Scrum Meeting – ყოველდღიური, 15-წუთიანი Scrum თათბირი, რომლის მიზანია იმის გარკვევა, თუ რა მოხდა წინა თათბირის შემდეგ, კორექტირდეს სამუშაო გეგმა დღევანდელი დღის შესაბამისად და განისაზღვროს არსებული პრობლემის გადაწყვეტის გზა. Scrum-გუნდის ყოველი წევრი პასუხობს სამ კითხვას: რა გააკეთა წინა თათბირის შემდეგ, რა პრობლემები აქვს და რა უნდა გააკეთოს მომდევნო შეხვედრამდე. ამ თათბირზე დასწრება შეუძლია ნებისმიერ დაინტერესებულ პირს, მაგრამ გადაწყვეტილების მიღების უფლება აქვთ მხოლოდ Scrum-გუნდის წევრებს. ეს წესი მართებულია, რადგან ისინი იღებენ ვალდებულებას იტერაციის მიზნის მიღსაღწევად. გარე პირის ჩარევა ასეთ დროს ხსნის მათგან შედეგის პასუხისმგებლობას.
- Sprint Review Meeting. Sprint მიმოხილვის შეხვედრა. იმართება ყოველი სპრინტის დამთავრების შემდეგ (ნახ.29). ჯერ Scrum-გუნდი წარმოადგენს პროდუქტის დემონსტრაციას, რომელიც ამ სპრინტის დროს განხორციელდა. აქ მოწვეული იქნება დამკვეთის ყველა დაინტერესებული წარმომადგენელი. პროდუქტის მფლობელი განსაზღვრავს, თუ რომელი მოთხოვნა შესრულდება Sprint Backlog-დან რასაც განიხილავენ გუნდთან და დამკვეთის წარმომადგენლებთან ერთად, თუ როგორ განაწილდეს პრიორიტეტები უკეთესად მომდევნო სპრინტის Sprint Backlog-ში. შეხვედრის მეორე ნაწილი ეხება წინა სპრინტის ანალიზს, რომელსაც წარმართავს Scrum-ოსტატი. Scrum-გუნდი აანალიზებს ბოლო სპრინტის დროს ერთობლივი მუშაობის

დადებით და უარყოფით მომენტებს, იღებს დასკვნებს მნიშვნელოვან გადაწყვეტილებებს შემდგომი მუშაობისათვის. Scrum-გუნდი ასევე ეძებს გზებს მომავალი სამუშაოს ეფექტურობის ასამაღლებლად. შემდეგ ციკლი მეორდება.



ნახ.13.3. Scrum-მეთოდი Sprint-ბიჯებით

13.4. UML/Agile მეთოდოლოგიების კომპრომისული გამოყენება

განიხილება საპრობლემო სფეროს მართვის საინფორმაციო სისტემების დაპროექტებისა და დეველოპმენტის პროცესების დიაგნოსტიკისა და IT-კონსალტინგის ამოცანები ობიექტორიენტირებული მიდგომის საფუძველზე. ყურადღება გამახვი-ლებულია პროგრამული პროექტების მენეჯმენტის საკითხებზე, უნიფიცირებული მოდელირების ენისა და ექსტრემალური პროგრამირების პრინციპების და მეთოდების კომპრომისულ გამოყენებაზე. შემოთავაზებულია პროგრამული სისტემების სასიცოცხლო ციკლის ბიზნესპროცესების მოდელირების აქტიურობის დიაგრამები და შესაბამისი თვალსაზრისით რეალიზებული პრაქტიკული ექსპერიმენტების შედეგები MsVisual_Studio.NET /Enterprise Architect/Ms Visio გარემოში.

საპრობლემო სფეროს სახით განვიხილავთ კორპორაციულ ობიექტებს, რომელთაც მიეკუთვნება სახელმწიფო ან კერძო სტრუქტურათა საწარმოო, ორგანიზაციული, საბანკო-საფინანსო და სხვა ტიპის იურიდიული სუბიექტები. მათი ფუნქციონირების კეთილდღეობა, კონკურენციის მძაფრ პირობებში, ბევრადაა დამოკიდებული მართვის აპარატის მოქნილობასა და საიმედოობაზე. რთული ბიზნესპროცესების კორპორაციული დაგეგმვისა და ოპერატიული მართვის მექანიზმების ეფექტური ორგანიზებითა და მუდმივი სრულყოფით შესაძლებელი ხდება ამ ობიექტების სასიცოცხლო ციკლის გახანგრძლივება, რაც უდავოდ აქტუალურია.

კორპორაციის მართვის საინფორმაციო სისტემის (პროგრამული სისტემის) შექმნა მოიცავს IT-კონსალტინგის: დიაგნოსტიკური ანალიზის, ექსპერტული შეფასებების,

ბიზნეს-პროგრამების დაგეგმვის, მათი განხორციელების ორგანიზების, ფაქტ-შედეგების აღრიცხვის, ეკონომიკური ანალიზისა და შეფასების, ობიექტზე ეფექტური ზემოქმედების მმართველი გადაწყვეტილების მიღების პროცესების ხელშემწყობი მექანიზმების შემუშავებას და მათ კომპიუტერულ რეალიზაციას [1,2].

ასეთი ობიექტების მართვის მექანიზმების მოდელი საკმაოდ რთულია და მიეკუთვნება ძნელად ფორმალიზებად დიდი სისტემების კლასს. მათი აგებისა და ეფექტური გამოყენებისათვის მიზანშეწონილია არაერთგვაროვანი, რაოდენობრივი და ხარისხობრივი მეთოდების კომპლექსური გამოყენებით. კერძოდ, ერთი მხრივ, კოგნიტური მოდელების ასაგებად ექსპერტულ შეფასებათა სხვადასხვა მეთოდის ინტეგრირებული გამოყენება, და მეორეს მხრივ, ბიზნეს-პროგრამების დაგეგმვის პროცესების ოპტიმიზაცია და კვლევა პეტრის ქსელებისა და მასობრივი მომსახურების სისტემების თეორიის საფუძველზე [3,4].

სრულყოფილი და საიმედო, მოქნილი პროგრამული უზრუნველყოფის (Software Engineering) სწრაფად დაპროექტება, რეალიზაცია, დანერგვა და შემდგომი თანხლება სისტემის დამკვეთ ორგანიზაციაში მეტად მნიშვნელოვანი ამოცანაა და მისი ეფექტურად გადაწყვეტა ბევრადაა დამოკიდებული როგორც საპროექტო-დეველოპმენტის გუნდის შემადგენლობასა და გამოცდილებაზე, ისე IT-ინფრასტრუქტურასა და CASE-ინსტრუმენტებზე.

ხშირად შეუძლებელია სრულყოფილი და საიმედო სისტემების აგება „სწრაფად“. ობიექტორიენტირებული დაპროგრამების მეთოდი, რომელიც უნიფიცირებული მოდელირების ენის (UML) საშუალებით დამკვიდრდა, უნივერსალურია, რომლის გამოყენებით პროგრამის სასიცოცხლო ციკლი მოითხოვს მისი აუცილებელი ეტაპების იტერაციულ განვითარებას [5].

მეხუთე თავის 5.12 ნახაზზე ნაჩვენებია ეს ეტაპები განტერის მოდელის საფუძველზე იტერაციული ბიჯებით [6]. შედარების მიზნით, 5.11 ნახაზზე მოცემულია იგივე ეტაპები ტრადიციული მეთოდისათვის. პროგრამული სისტემების აგების ბიზნესპროცესების კვლევა და მოდელირება სწორედ ამ დიაგრამების საფუძველზე ტარდება. ამ ნახაზების შედარებით ცხადი ხდება, რომ პროგრამული სისტემის საკონტროლო წერტილებში (0-12), ეტაპების მიხედვით ხორციელდება იტერაციული სამუშაოები (დაბრუნება უკანა წერტილებში განმეორებითი პროცედურების ჩასატარებლად), სისტემის ფუნქციობის სისრულის დაზუსტების ან გაფართოების მიზნით.

პროგრამული სისტემის რეალიზაციის სისწრაფე (დროითი ფაქტორი - ვადები) და სისრულე (იგულისხმება ფუნქცია, რეალიზაცა და საინტერფეისო, ხარისხობრივი ასპექტები) საყურადღებო შეზღუდვებია, რომელთა გათვალისწინება შესრულების პროცესში მეტად მნიშვნელოვანია და განაპირობებს რეალიზაციის მეთოდებისა და ინსტრუმენტების გამოყენების პირობებსაც.

ამ კონტექსტში იტერაციული დაპროგრამების ტრადიციული და ობიექტ-ორიენტირებული მეთოდების გვერდით საჭიროა განვიხილოთ ექსტრემალური დაპროგრამების მეთოდიც (XP, Agile Programming) [7,8]. მისთვის დამახასიათებელია დროითი და ფუნქციური სისრულის კომპრომისული გადაწყვეტა, რომლის დროსაც მოცემული იტერაციისათვის განიხილავენ იმ მინიმალური რეალიზაციის შესაძლებლობას, რაც უზრუნველყოფს სისრულეს და ფუნქციურ შეკრულობას.

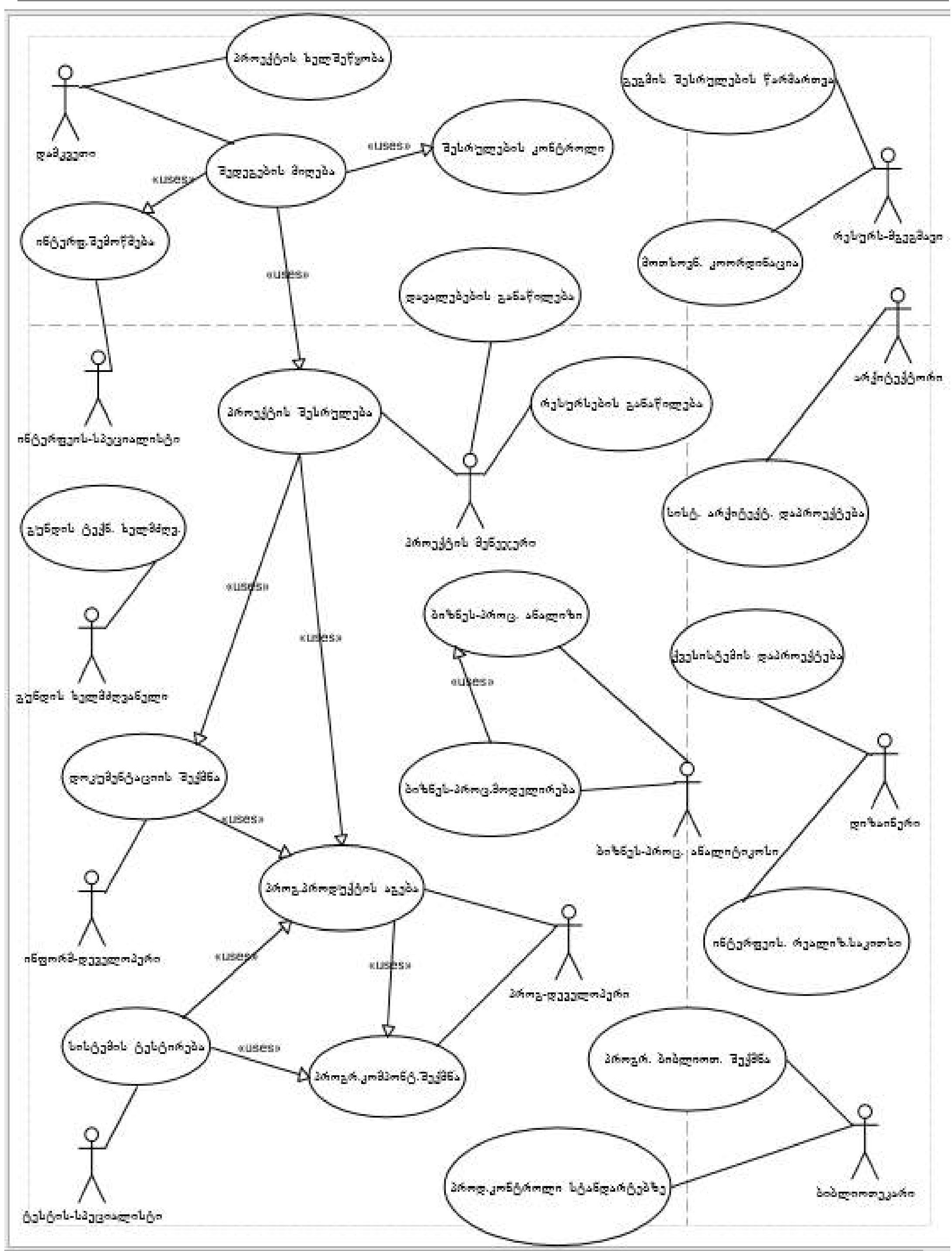
ექსტრემალური პროგრამირების მეთოდის სასიცოცხლო ციკლის მოდელში ძირითადი ყურადღება მახვილდება საპრობლემო ამოცანის სწორად ჩამოყალიბებაში დამკვეთის ბიზნეს-ანალიტიკოსთან ერთად, ნაკლებად იხარჯება დრო უნივერსალური დიაგრამების აგებასა და საანგარიშო დოკუმენტაციის გაფორმებაზე, დ,ა, რა თქმა უნდა, ხდება ძირითადი ეტაპების (კონსტრუირება-დაპროგრამება) ფაზათა შერწყმა.

აქედან გამომდინარე, პროგრამული სისტემის მენეჯერი კონკრეტული პროექტის ამოცანებისა და მოთხოვნების შესაბამისად უნდა განსაზღვრავდეს როგორც პროგრამირების მეთოდის, ეტაპთა ფაზებისა და იტერაციათა მოთხოვნების შერჩევა-ფორმირებას, ასევე გუნდის შემადგენლობას.

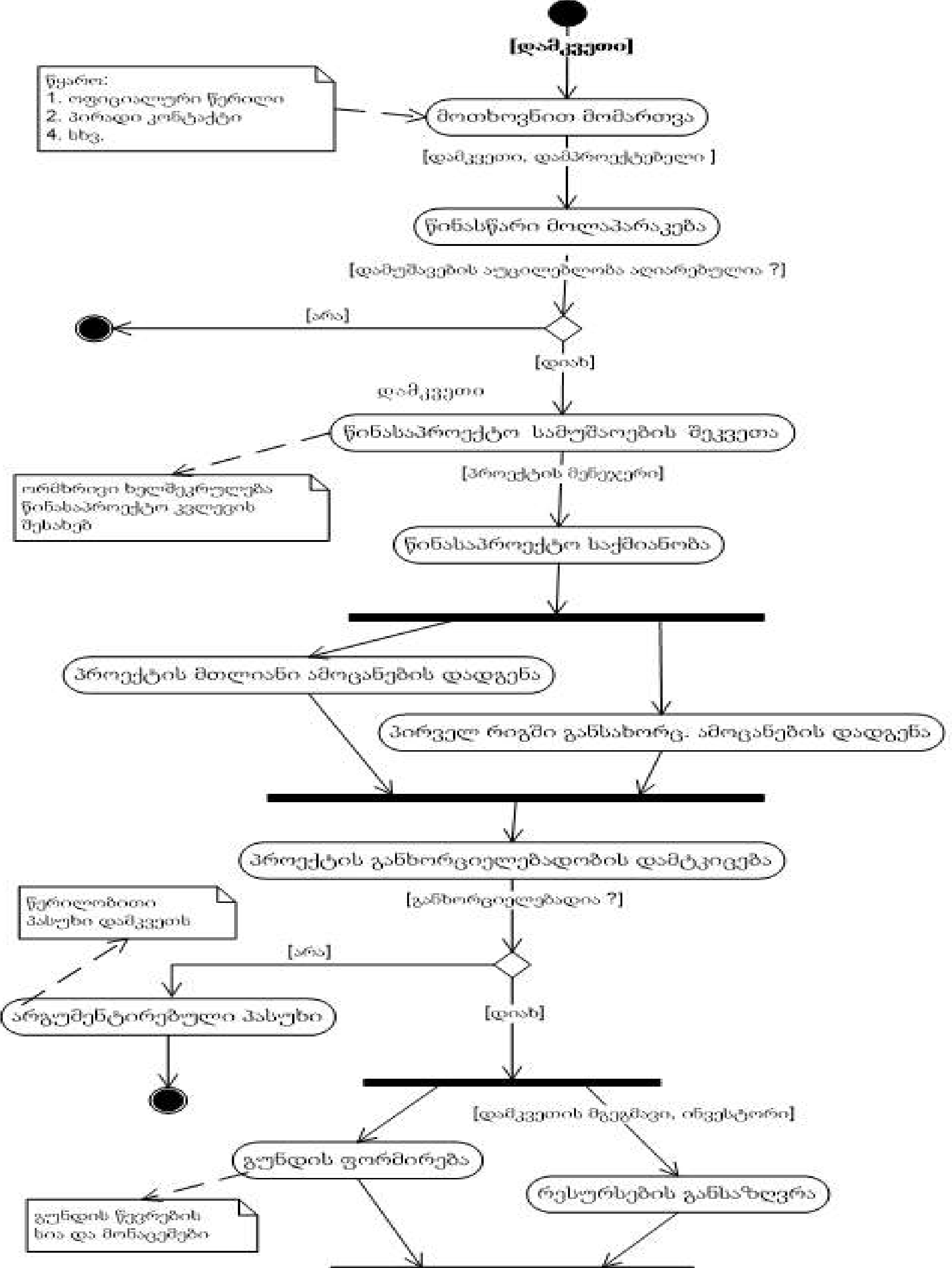
პროგრამული სისტემების პროექტის მენეჯერისთვის მნიშვნელოვანია პროდუქტის სასიცოცხლო ციკლის ბიზნეს-პროცესების დეტალური ანალიზი მის ცალკეულ ეტაპებზე ეფექტური მართვის განსახორციელებლად.

13.4 ნახაზზე მოცემულია პროგრამული პროექტების შესრულების ერთიანი პროცესის ზოგადი მოდელი, აგებული UML–ის UseCase დიაგრამის სახით. მოდელზე ასახულია ამ პროცესში მონაწილე როლები (დამკვეთი, პროექტის მენეჯერი, ბიზნეს-პროცესების სპეციალისტი, სისტემის არქიტექტორი, დეველოპერი-პროგრამისტი, ტესტირების სპეციალისტი და სხვ.) და მათი ფუნქციები.

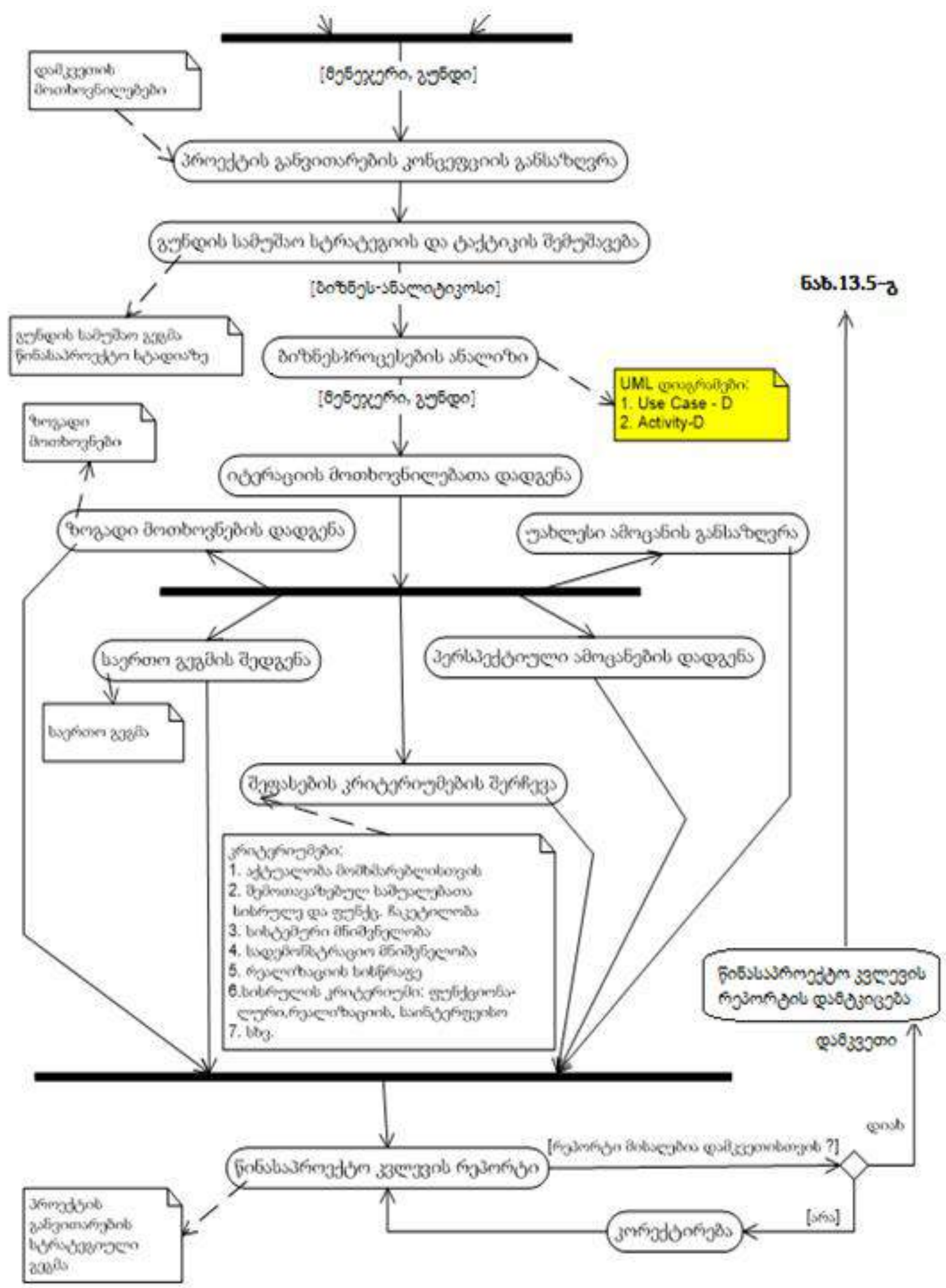
13.5 ნახაზზე ილუსტრირებულია შესაბამისი ბიზნეს-პროცესების აქტიურობათა დიაგრამა. აღწერილი ზოგადი მოდელები მენეჯერის მიერ ადაპტირდება კონკრეტული პროექტისათვის რაც იღებს კერძო სახეს. მთლიანი სისტემის ბიზნესპროცესების ანალიზი საფუძველია სისტემის არქიტექტურის და საერთოდ, IT-ინფრასტრუქტურის დასადგენად.



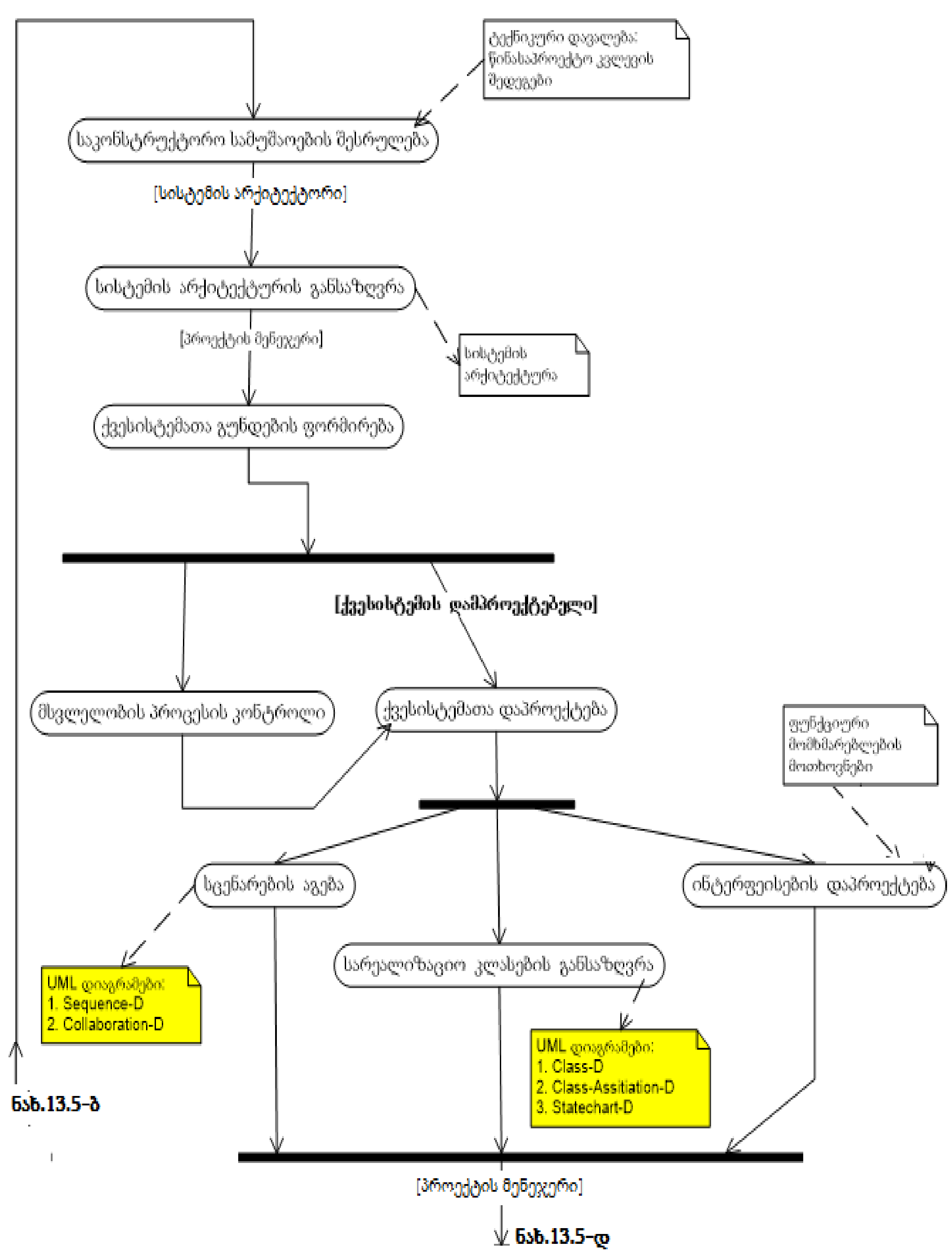
ნახ.13.4. პროგრამული სისტემის რეალიზაციის
UseCase მოდელი



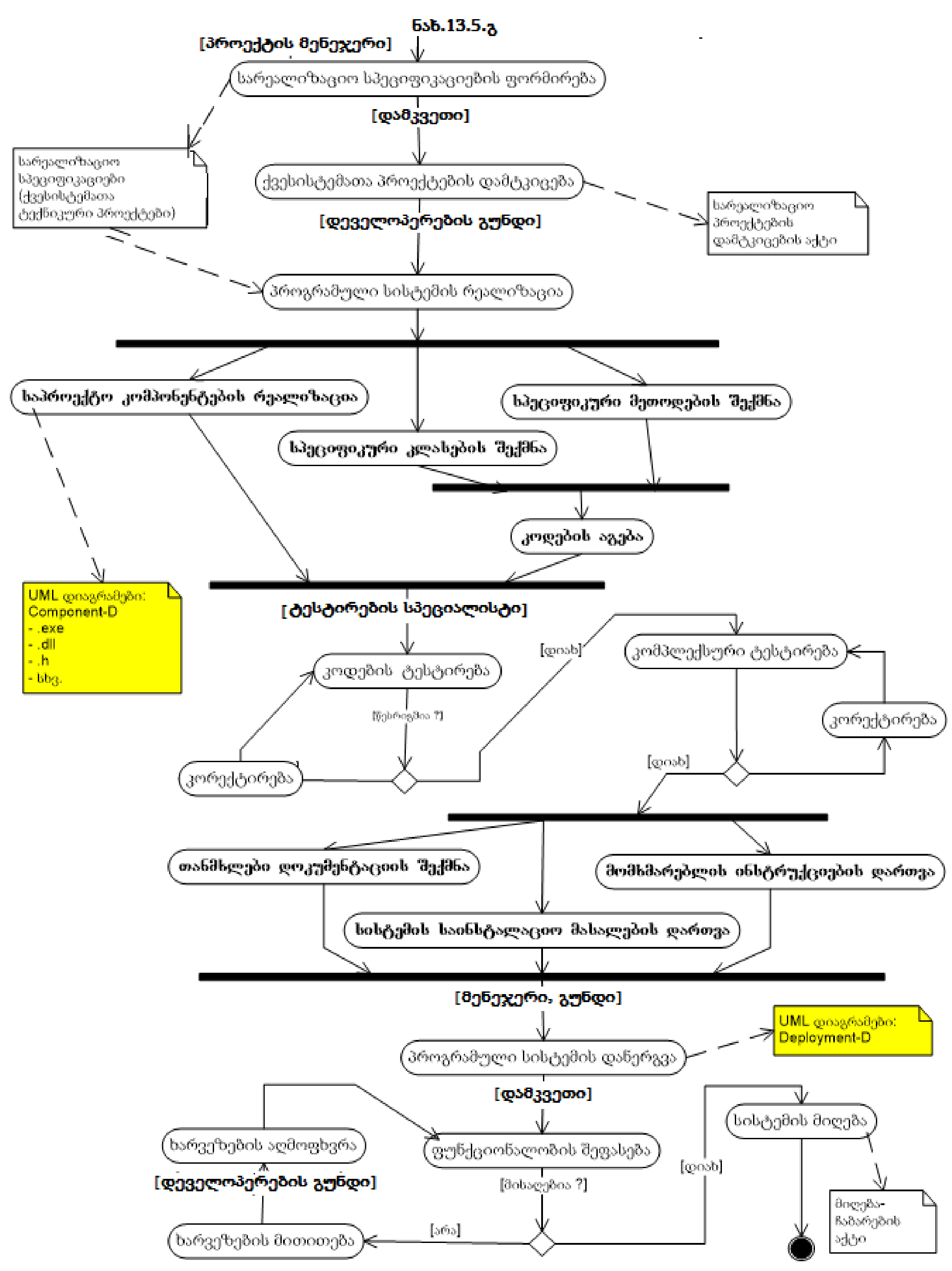
ნახ.13.5-ა. პროგრამული სისტემის რეალიზაციის
Activity დიაგრამა (გაგრძელება: 13.15-ბ,გ,დ)



ნახ.13.5-ზ.



ნახ.13.5-გ.



ნახ.13.5-დ.

პროგრამული სისტემის პროექტის მიზნების, საწყისი პირობებისა და მოთხოვნილებათა გათვალისწინებით მენეჯერის მიერ შეირჩევა კომპრომისული გადაწყვეტილება და, მაგალითად, UML/2-ის ინსტრუმენტში Enterprise Architect ბიზნესპროცესების მოდელირების, კონსტრუირების, დეველოპმენტისა და ტესტირების ეტაპების დასაგეგმად ჩაირთვება შესაბამისი „ჩეკბოქსები" (ნახ.13.6).

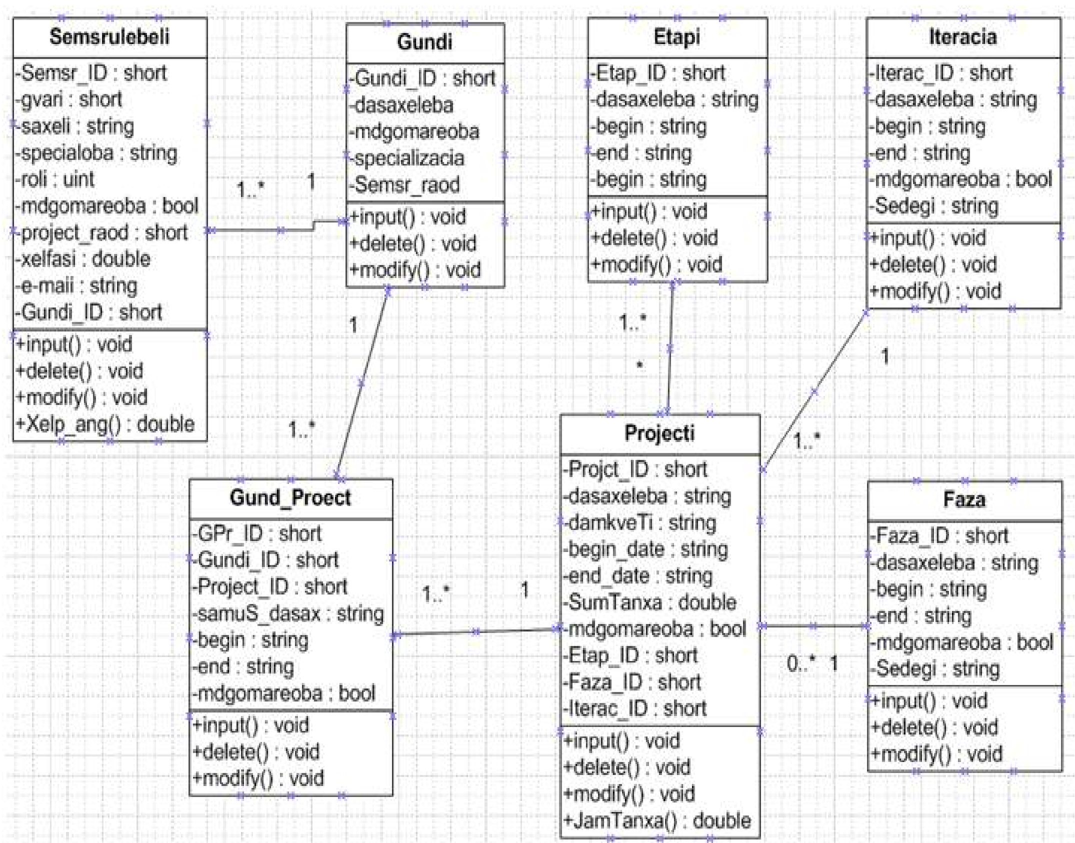
ნახ.13.6.. Enterprise Architect - პროექტის შედგენილობის დაგეგმვა

დიდი პროექტებისათვის, რომელშიც რესურსები და დროითი ფაქტორები, შედარებით კრიტიკული არაა, ხდება ობიექტორიენტირებული მიდგომის ყველა ეტაპისა და ფაზის გამოყენება შესაბამისი საკონტროლო წერტილების აუცილებელი მონიტორინგით და რეპორტებით.

ექსტრემალური მიდგომის მეთოდის გამოყენებისას, მცირე ან საშუალო პროექტებში, სადაც რეალიზაციის დრო კრიტიკულია, ძირითადი ყურადღება ამოცანის სწორად ჩამოყალიბების, დეველოპინგის და ტესტირების ეტაპებზე გადადის.

13.7 ნახაზზე მოცემულია პროგრამული პროექტების მენეჯმენტის სისტემის კლასთაასოციაციის დიაგრამის ფრაგმენტი, რომელშიც ასახულია ინფორმაცია პროექტების, გუნდების, შემსრულებლების, იტერაციების, ეტაპებისა და ფაზების შესახებ, აგრეთვე მათი ურთიერთკავშირები და თითოეულის მდგომარეობა.

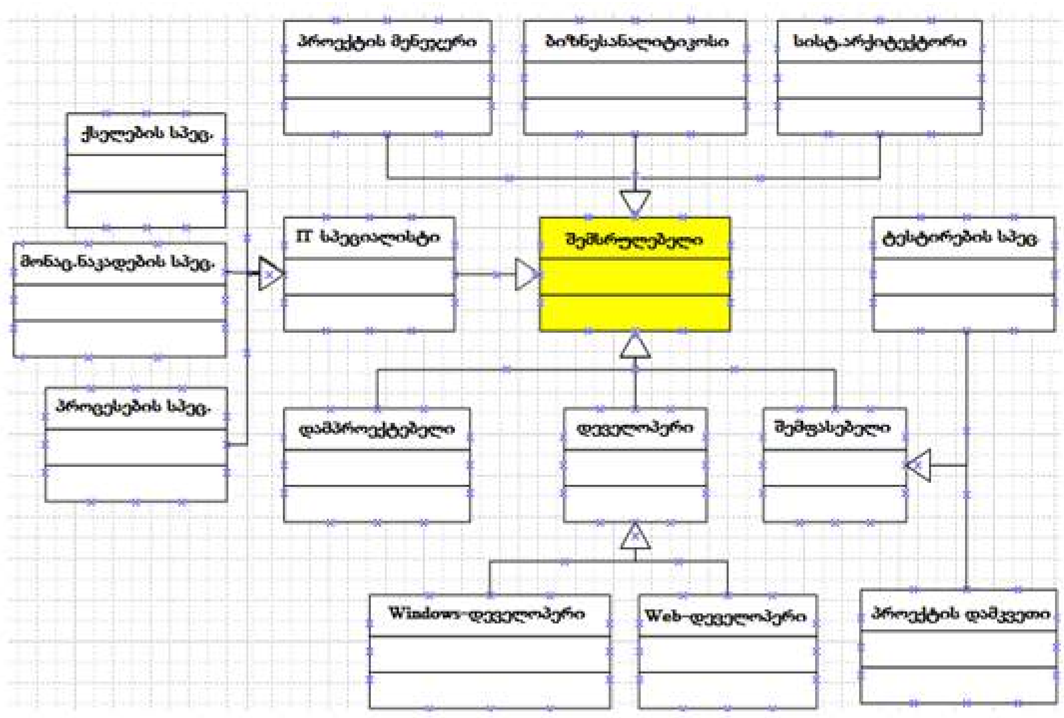
ასეთი ინფორმაცია სასარგებლოა პროექტების მენეჯმენტის განსახორციელებლად, კერძოდ, საპროექტო სამუშაოების გასანაწილებლად შემსრულებლებს შორის მათი დატვირთვისა და კალენდარული ვადების გათვალისწინებით, რომელიც წყდება, მაგალითად, MsProject პაკეტით.



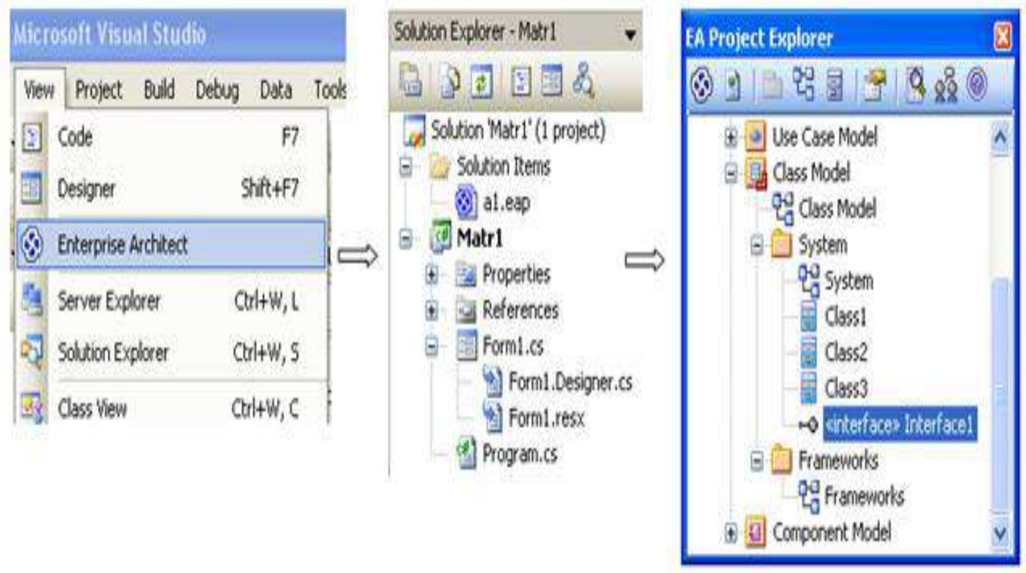
ნახ.13.7. კლასთა ასოციაციის დიაგრამის ფრაგმენტი

13.8 ნახაზზე ილუსტრირებულია პროექტის შემსრულებელთა გუნდის წევრების იერარქიული სქემა მემკვიდრეობითობის (Generalization) კავშირების საფუძველზე. პროექტის მენეჯერი თვითონაც გუნდის წევრია, რომელსაც ეხება გუნდის დაკომპლექტება პროექტის მიზნებიდან, სირთულიდან და განხორციელების ვადებიდან გამომდინარე. იგი აგრეთვე გაითვალისწინებს, საჭიროების შემთხვევაში, ექსტრემალური პროგრამირების პრინციპებსაც [8,13].

UML-დიაგრამების (მაგალითად, კლასების, აქტიურობათა, მიმდევრობითობის) დაპროექტების შემდეგ საჭიროა კოდის გენერაციის პროცედურების შესრულება. ჩვენს შემთხვევაში შესაძლებელია MDG (Model Driven Generation) გამოყენება. MDG Link for Visual Studio .NET საშუალებას იძლევა Enterprise Architect ინსტრუმენტით ვიმუშაოთ .NET გარემოში და განვახორციელოთ პირდაპირი და რევერსიული დაპროექტება (ნახ.13.9).



ნახ.13.8. პროექტის შემსრულებელთა როლების
მემკვიდრეობითი კავშირები



ნახ.13.9. Visual Studio.NET + Enterprise Architect

ამგვარად, პროგრამული სისტემების შექმნის თანამედროვე ინტეგრირებული ტექნოლოგიები საშუალებას იძლევა არა მხოლოდ დავაჩქაროთ ხარისხიანი პროგრამული პროდუქტების წარმოება, არამედ განვახორციელოთ მათი სასიცოცხლო ციკლის მონიტორინგი და ვმართოთ მისი პროცესები, რომლებიც დაკავშირებულია კომპიუტერული და ადამიანური რესურსების ეფექტიან ორგანიზებასა და თვით პროდუქტის გამოყენების პერიოდის გახანგრძლივებასთან.

შემოთავაზებულია ტრადიციული, CASE და eXtreme Programming მეთოდების პრინციპების ანალიზი და მათი ბიზნესპროცესების მოდელები, მენეჯერისა და გუნდის წევრების ფუნქციები და პროექტის შესრულების სტრატეგიული გეგმების ფორმირება კომპრომისული გადაწყვეტილებების საფუძველზე.

IV ნაწილი

მართვის საინფორმაციო სისტემების დაპროგრამების ახალი ტექნოლოგია: Windows Presentation Foundation

XIV თავი. WPF-ტექნოლოგია - ძირითადი ცნებები და თეორიული ასპექტები 316
XV თავი. WPF-ტექნოლოგიის ვიზუალური ელემენტები და მათი გამოყენება
აპლიკაციების ასაგებად 370

განიხილება კორპორაციული მენეჯმენტის პროცესების ავტომატიზაციის საფუძვლები ახალი ინფორმაციული ტქნოლოგიების ბაზაზე. კერძოდ, მოცემულია MsVisual Studio.NET Framework 4.5 ინტეგრირებულ გარემოში ჰიბრიდული კომპიუტე-რული სისტემების (Windows- და Web-აპლიკაციების) დაპროგრამების ინსტრუმენტული საშუალებები:

- WPF (Windows Presentation Foundation);
- WF (Workflow Foundation) და
- WCF (Windows Communication Foundation) .

ეს ტექნოლოგიები ეფუძნება XAML (სისტემის დიზაინის ნაწილი) და დაპროგრამების ენების (მაგალითად, C#, C++, VB) (სიტემის ლოგიკური ნაწილი) კომპლექსურ გამოყენებას.

თეორიულ საკითხებთან ერთად წარმოდგენილია პრაქტიკული ღირებულების ამოცანათა ერთობლიობა, მათი პროგრამული ლისტინგები და გადაწყვეტის მეთოდური ინსტრუქციები.

XIV თავი
WPF-ტექნოლოგია: ძირითადი ცნებები და თეორიული ასპექტები

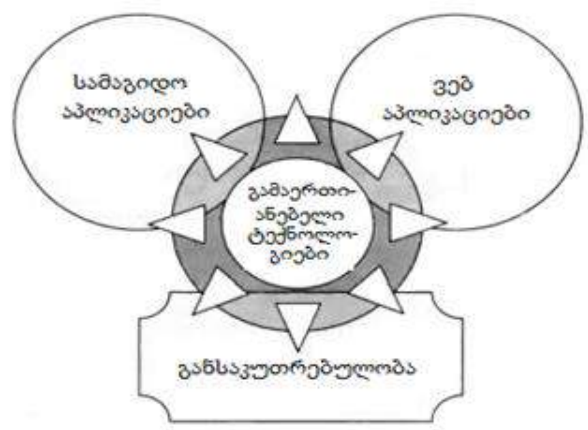
14.1. Windows Presentation Foundation ტექნოლოგია

აპლიკაციების (დანართების) ორი ნაირსახეობაა ცნობილი: ვინდოუს სისტემები, რომელთაც ასევე სამაგიდო (Desktop) აპლიკაციებს უწოდებენ და ვებაპლიკაციები, რომელთა გამოყენებაც ინტერნეტ ბრაუზერებიდანაა შესაძლებელი [16-18].

ეს დანართები იქმნება .NET Framework-ის ორი სხვადასხვა პაკეტით. პირველი - Windows Forms კომპონენტებით და მეორე ASP.NET-ის საშუალებით. ორივეს აქვს თავისი უპირატესობები და ნაკლოვანებანი. კერძოდ, სამაგიდო დანართები ძალზე მოქნილი და რეაქციულია, ხოლო Web-დანართები ინტერნეტის საშუალებით იძლევა დისტანციური წვდომის საშუალებას ერთდროულად მრავალი მომხმარებლისათვის.

მაგრამ თანამედროვე კომპიუტერული ტექნოლოგიების სამყაროში ამ ორი სახის აპლიკაციებს შორის საზღვრები სულ უფრო და უფრო იშლება [4].

Web-სამსახურების და WCF (Windows Communication Foundation) სერვის-ორიენტირებული არქიტექტურის აგების საშუალებების გაჩენამ განაპირობა სამაგიდო- და ვებდანართების ფუნქციონირების შესაძლებლობა ერთიან განაწილებულ გარემოში, სადაც მონაცემთა გაცვლა ხორციელდება როგორც ლოკალურ, ისე გლობალურ ქსელებში. 14.1 ნახაზზე ნაჩვენებია ეს გამაერთიანებელი პროცესი.



ნახ.14.1

WPF (Windows Presentation Foundation) არის ერთ-ერთი ასეთი გამაერთიანებელი ტექნოლოგია. იგი საშუალებას იძლევა აიგოს ისეთი დანართი, რომელშიც გამორიცხულია დაპირისპირება სამაგიდო აპლიკაციასა და ინტერნეტს შორის.

WPF-დანართს, როგორც ეს ნაჩვენები იქნება ქვემოთ, შეუძლია ფუნქციონირება როგორც სამაგიდო აპლიკაციას, ისე როგორც ვებაპლიკაციას ბრაუზერის შიგნით. არსებობს ასევე WPF-ის შეზღუდული ვერსია, სახელით Silverlight, რომლითაც შესაძლებელია ვებდანართში დინამიკური მდგენელის დამატება [39].

წიგნში განიხილება დაპროგრამების საფუძვლები WPF ტექნოლოგიით, WPF-ის საბაზო სტრუქტურა, მისი ძირითადი ცნებები და შედგენილობა, პროექტების აგება Ms Visual Studio .NET Framework 4.0/4.5 გარემოში WPF/C# -ის საფუძველზე [1,5,8].

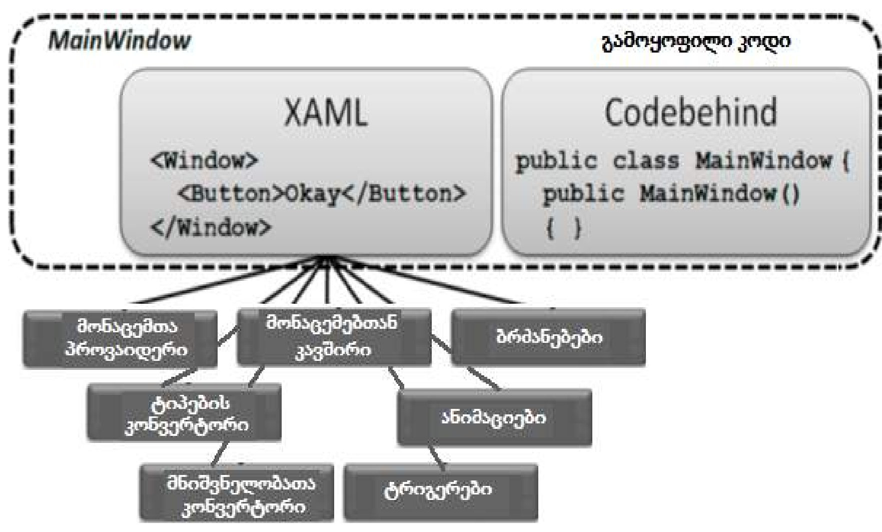
პრაქტიკული სამუშაოების შესასრულებლად Windows Presentation Foundation სისტემაში საჭიროა (მინიმალურად) შემდეგი რესურსები:

➢ **ოპერაციული სისტემა** - .Net Framework-ის 3.0 (და SP1) ვერსიას აქვს Windows XP, Windows Vista და Windows Server 2003–ის მხარდაჭერა. .Net Framework 4.0/4.5-ის გამოჩენის შემდეგ იგი ბევრად სრულყოფილი გახდა. ახალი ფუნქციონალობის გამოსაყენებლად სასურველია კონფიგურაციის გაუმჯობესება:

- Windows 7 ან Windows 8 (ან Windows Server 2008 R2);
- Graphics card (გრაფიკული დაფა) DirectX-ის მე-9 ვერსიისთვის;

➢ **ინტეგრირებული სამუშაო გარემო:** .Net Framework 4.0/4.5.

14.2 ნახაზზე ნაჩვენებია ორი ძირითადი პროგრამული ტანდემი (XAML-თავისი მეთოდებით და კოდი).



ნახ.14.2

14.1.1. WPF ტექნოლოგიის შესაძლებლობები

WPF (ადრე ცნობილი იყო სახელით Avalon) არის ტექნოლოგია, რომელიც საშუალებას იძლევა დაიწეროს პლატფორმაზე დამოუკიდებელი აპლიკაციები, დიზაინისა და ფუნქციური შესაძლებლობების ცხადი დაყოფით. იგი ეფუძნება ადრე არსებულ ისეთ ტექნოლოგიათა გაფართოებულ ცნებებს და კლასებს, როგორიცაა Windows Forms, ASP.NET, XML, GDI+ და ა.შ. ასეთი მსგავსება კარგად ჩანს ვებ-დანართის აწყობისას .NET Framework გარემოში [1,2].

გარდა ამისა WPF-ში მრავლადაა ახალი ფუნქციონალური საშუალებები პროგრამისტებისა და მომხმარებლებისთვის, რაც მისი, როგორც ახალი ტექნოლოგიის განხილვის უფლებას იძლევა. მაგალითად, Silverlight ტექნოლოგიას შეუძლია პროგრამები პირდაპირ ბრაუზერში შეასრულოს. ამ პლაგინის ინსტალაცია აუცილებელია. არაა დიდი მოცულობის, კომპაქტურია. ისე როგორც WPF-ში, აქაც გამოიყენება XAML და C#. მაგრამ WPF პროგრამები უშუალოდ ბრაუზერში ვერ გაიშვება.

WPF მუშაობს ვექტორულ გრაფიკაზე ბაზირებულად და არა პიქსელებზე. მართვის ელემენტები, გრაფიკები, აგრეთვე ნახაზები არ იხაზება პიქსელებით, არამედ აღიწერება მრუდებითა და წრფეებით. ამის გამო აღარაა ძლიერი დამოკიდებულება მონიტორებისაგან.

14.1.2. WPF -ის შესაძლებლობები დიზაინერებისათვის

მომხმარებელთა ინტერფეისების დასაპროექტებლად WPF-ში გამოიყენება დანართების ფორმატირების გაფართოებული ენა XAML (Extensible Application Markup Language) [3]. იგი მსგავსია ASP.NET-ის ფორმატირების ენის, რომელიც იყეებს XML-სინტაქსს და შეუძლია მომხმარებლის ინტერფეისს დაუმატოს მართვის ელემენტები დეკლარაციული, იერარქიული სახით. ამასთანავე მას შეუძლია მერთვის ელემენტების შექმნა, რომლებიც შეიცავს მართვის სხვა ელემენტებს, რაც მნიშვნელოვანია როგორც ინტერფეისის კონსტრუირებისათვის, ისე დანართის ფუნქციური შესაძლებლობებისათვის.

XAML-ენა ბევრად მძლავრია, ვიდრე ASP.NET და არაა შეზღუდული HTML ენის შესაძლებლობებით მონაცემთა ვიზუალიზაციის დროს ინტერფეისებში [4]. ის დამუშავდა სპეციალურად დღეისათვის არსებული მძლავრი გრაფიკული დაფების გათვალისწინებით DirectX-ის საფუძველზე [5]. მაგალითად:

- ვექტორული გრაფიკა და კოორდინატები მცოცავი მძიმით - უზრუნველყოფს ობიექტების მასშტაბირებას, ბრუნვას და ტრანსფორმირებას ხარისხის დაუკარგავად;
- ორ- და სამგანზომილებიანი შესაძლებლობები ვიზუალიზაციის სრულყოფისათვის;
- შრიფტების დამუშავებისა და ვიზუალიზაციის სრულყოფილი საშუალებები;

- გრადიენტული, უწყვეტი და ტექსტური შევსება გამჭვირვალეობის არააუცილებელი ეფექტებით მომხმარებლის ინტერფეისის ობიექტებისათვის;
- ანიმაციის კადრების დაშლის ტექნოლოგია, რომელიც გამოიყენება ყველა სიტუაციაში, მათ შორის მომხმარებლის მიერ გენერირებულ მოვლენებში, მაგალითად, ღილაკის დაჭერის იმიტაცია;
- მრავალჯერადი მოხმარების რესურსები, რომელთა გამოყენება შეიძლება მართვის ელემენტების დინამიკური სტილიზაციისათვის.

დიზაინერებისათვის განსაკუთრებული ინტერესი აქვს მაიკროსიფთის ინსტრუმენტს Expression Blend (EB), რომლითაც შესაძლებელია XAML-ფაილების შექმნა და შემდგომ მათი გადატანა დანართებში. EB-ში შესაძლებელია პროექტების შექმნა და რედაქტირება ისევე, როგორც VisualStudio-ს Solution Explorer-ში. ამგვარად, Expression Blend სასურველი, მაგრამ არააუცილებელი კომპონენტია WPF-დანართების ასაგებად. ის VS სტანდარტულ ვერსიას არ მოჰყვება, მაგრამ შეიძლება მისი დემო-ვერსიის ჩამოტვირთვა:

microsoft.com/expression/products/overview.aspx?key=blend

14.1.3. WPF -ის შესაძლებლობები C# -ის დეველოპერებისათვის

აპლიკაციების დეველოპერები პროექტებს ქმნიან VC (Visual Studio) და VCE (Visual C# Express - არის VC-ს შეზღუდული უფასო ვერსია) გარემოში სამუშაოდ.

WPF-ში გამოიყენება გამოყოფილი კოდის მოდელი, როგორც ASP.NET-ში. მაგალითად, Button მართვის ელემენტისათვის მოვლენის დამმუშავებლის (პროგრამის) მიმაგრება შეიძლება მისი XML ელემენტისთვის Click ატრიბუტის დამატებით. ეს ატრიბუტი მიუთითებს მოვლენის დამმუშავებლის სახელზე შესაბამისი XAML-გვერდის გამოყოფილი კოდის ფაილში, რომელიც შეიძლება დაწერილი იყოს C#-ზე.

WPF დანართებში მართვის ელემენტების მანიპულირება შეიძლება Windows Forms მსგავსად, რომლის დროსაც გამოიყენება დაპროგრამების ხერხები მომხმარებლის ინტერფეისების ასაგებად. გამოყოფილი კოდის საშუალებით შეიძლება შეიქმნას მართვის ელემენტის ეგზემპლარი, დაყენდეს თვისებები, მიებას მოვლენათა დამმუშავებლები და დაემატოს ფორმაზე მართვის ეს ელემენტი. ამ პროცესს შეუძლია მთლიანად გამორიცხოს XAML-კოდი.

ასეთ შემთხვევაში გამოყოფილი კოდის ზომა იქნება გაცილებით დიდი, ვიდრე შესაბამისი დეკლარაციული XAML-ის კოდი და ამასთანავე იკარგება დიზაინსა და ფუნქციურ შესაძლებლობებს შორის ცხადი საზღვარი, რაც არასახარბიელოა. ამიტომ სასურველია მომხმარებლის ინტერფეისზე მართვის ელემენტების განლაგება განხორციელდეს XAML-ით.

14.2. WPF-ის არქიტექტურა და კლასები

Windows Presentation Foundation (WPF) ეფუძნება ვიზუალიზაციის ვექტორულ სისტემას და ორიენტირებულია კლიენტების ვებ-აპლიკაციების (დანართების) დამუშავებაზე Microsoft.NET პლატფორმისათვის. ამ თავში განიხილება WPF-ის არქიტექტურა, ძირითად კლასთა იერარქია, მომხმარებლის ინტერფეისის აგების საკითხები, კონტროლის ელემენტების გამოყენების თავისებურებანი, შედგენილობის მართვის ძირითადი ელემენტები, მათი თვისებები და დეკლარაციული აღწერა. წარმოდგენილ ინფორმაციას - რესურსების, სტილის და შაბლონების შესახებ WPF-ის საკმაოდ ეფექტურ და მრავალფეროვან კონსტრუქციებისთვის აქვს შესავალი ხასიათი.

Windows Presentation Foundation (WPF) - ესაა კლიენტების Windows-დანართების აგების სისტემა Microsoft.NET ტექნოლოგიისათვის ვიზუალური შესაძლებლობებით. WPF-ით შეიძლება აიგოს ფართო სპექტრი როგორც ავტონომიური დანართების, ისე ვებ-ბრაუზერში განთავსებული აპლიკაციებისათვის.

WPF-ის საფუძველია ვიზუალიზაციის ვექტორული სისტემა, რომელიც გათვლილია თანამედროვე გრაფიკულ საშუალებებზე. ვიზუალური ინტერფეისის ასაგებად გამოიყენება XAML ენა, მართვის ელემენტები, მონაცემებთან მიბმა, მაკეტები, 2- და 3-განზომილებიანი გრაფიკა, ანიმაცია, სტილები და შაბლონები, დოკუმენტები და ტექსტები, მულტიმედია და გაფორმებები.

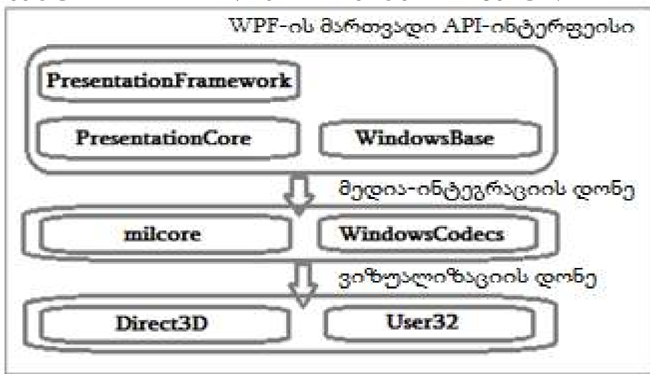
WPF-ის გრაფიკულ ტექნოლოგიას საფუძვლად უდევს DirectX. Windows Forms-ისაგან განსხვავებით, სადაც გამოიყენება GDI/GDI+, WPF-ის მწარმოებლურობა უფრო მაღალია, ვიდრე GDI+-ის, გრაფიკის აპარატურული დამაჩქარებლის გამოყენების გამო DirectX -ში.

WPF უზრუნველყოფს მომხმარებლის მაღალი დონის ინტერფეისს და იძლევა შემდეგ შესაძლებლობებს:

- ვებ-ისმაგვარი მოდელის აწყობა, რომელიც უზრუნველყოფს მართვის ელემენტების განლაგებას და მოწესრიგებას შინაარსის მიხედვით;
- ხატვის მრავალფუნქციურ მოდელს გრაფიკული კომპონენტების საფუძველზე (საბაზო ფორმები, ტექსტური ბლოკები, გრაფიკული ინგრედიენტები);
- მოდელი ფორმატირებული ტექსტით, რომელიც უზრუნველყოფს ფორმატირებული სტილიზებული ტექსტის ასახვას მომხმარებლის ინტერფეისის ნებისმიერ ნაწილში, ტექსტის კომბინირებას სიებთან, ნახატებთან და სხვა ინტერფეისულ ელემენტებთან;
- ანიმაციის მოცემა დეკლარაციული დესკრიპტორებით;
- აუდიო-ვიზუალური გარემოს მხარდაჭერა ნებისმიერი აუდიო- და ვიდეოფაილების შესასრულებლად;

- სტილები და შაბლონები, რომლებიც უზრუნველყოფს ფორმატირების და (მართვის ელემენტების) ვიზუალიზების მართვის სტანდარტიზებას, აგრეთვე ამ შედეგების მრავალჯერ გამოყენება პროექტის სხვადასხვა ადგილას;
- ბრძანებები, რომლებითაც ისინი განისაზღვრება ერთ ადგილას, მაგრამ მრავალგზის უკავშირდება დანართის სხვა ელემენტებს;
- მომხმარებლის დეკლარაციული ინტერფეისი, რომელიც ფანჯრების ან გვერდების შინაარსს აღწერს XAML ენის საშუალებით.

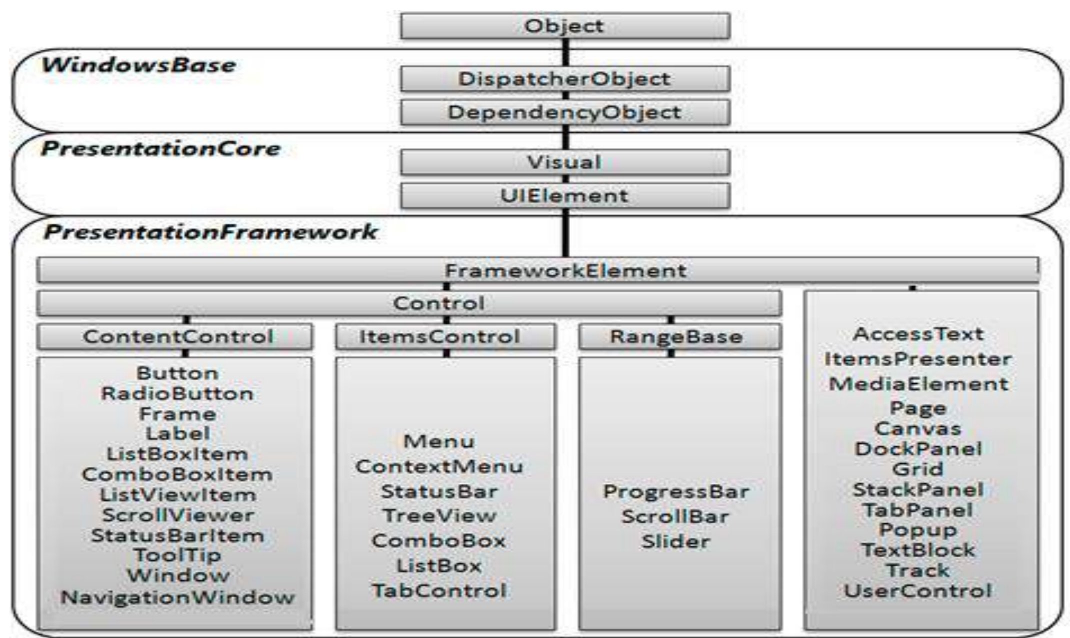
WPF არქიტექტურის ძირითადი კომპონენტები მოცემულია 14.3 ნახაზზე.



ნახ.14.3. WPF –ის არქიტექტურა

- PresentationFramework კომპონენტი შეიცავს WPF–ის ზედა დონის ტიპებს, როგორიცაა ფანჯრების, პანელებისა და სხვა ელემენტების წარმოდგენა;
- PresentationCore შეიცავს საბაზო ტიპებს, როგორიცაა UIElement და Visual, რომელთაგანაც იწარმოება მართვის ყველა ფორმა და ელემენტები;
- WindowsBase შეიცავს განსხვავებულ ტიპებს, რომელთა გამოყენება შეიძლება WPF–ის გარეთ. მაგალითად, DispatchObject და DependencyObject კომპონენტები;
- milcore კომპონენტი არის WPF ვიზუალიზაციის ბირთვი;
- WindowsCodecs ქვედა დონის API-ინტერფეისია გამოსახულებათა აგების მხარდასაჭერად;
- Direct 3D არის ასევე ქვედა დონის API-ინტერფეისი, რომლითაც ხორციელდება WPF-ის გრაფიკის ვიზუალიზაცია;
- User32 გამოიყენება იმის დასადგენად, თუ რომელ პროგრამას აქვს მიცემული ეკრანის რომელი უბანი.

14.4 ნახაზზე ნაჩვენებია WPF-ის მართვის ელემენტები კლასთა იერარქიის მიხედვით [8].



ნახ.14.4

WPF-ში მართვის ელემენტის საფუძველი არის ყოველთვის FrameworkElement კლასი, რომელიც წარმოიდგინება როგორც ყველაზე საფუძვლიანი საბაზი კლასი.

WPF–ის არქიტექტურა განსაზღვრავს ძირითად სახელსივრცეებს კლასთა იერარქიისათვის. მართვის ელემენტების საბაზო ერთობლიობა განსაზღვრავს სისტემის კლასთა საკვანძო იერარქიებს. 14.5 ნახაზზე აბსტრაქტული კლასები ნაჩვენებია ოვალებით, ხოლო კონკრეტული კლასები – მართკუთხედებით.

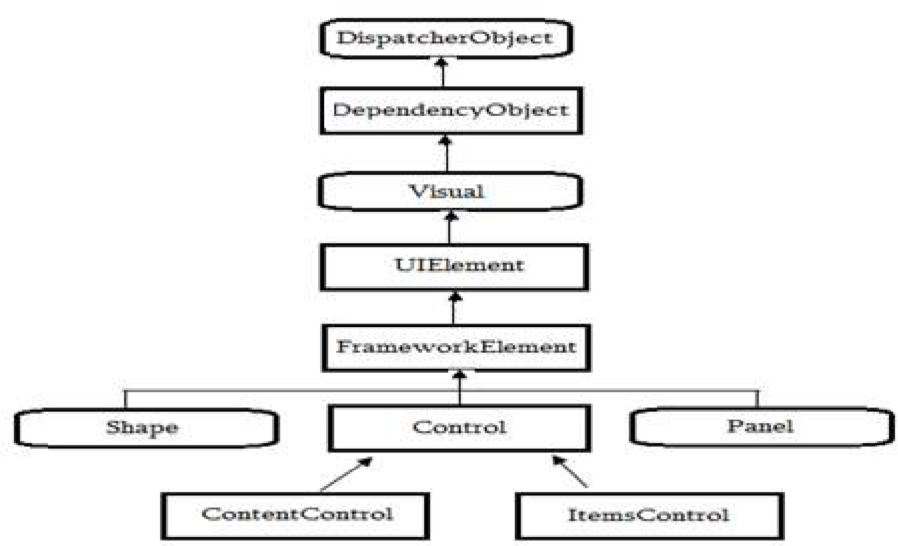
ობიექტთა უმრავლეოსობა WPF–ში იწარმოება DispatcherObject აბსტრაქტული კლასიდან. WPF ეფუძნება შეტყობინებათა გაცვლის სისტემას, რომლებიც მომხმარებლის ინტერფეისისათვის ფორმირდება ერთ ნაკადში, რომელიც იმართება და კონტროლდება დისპეტჩერის მიერ.

DispatcherObject კლასი უზრუნველყოფს დანართის ინტერფეისის ყოველი ელემენტისთვის ნაკადში შესრულების შემოწმებას და წვდომას დისპეტჩერისაკენ.

WPF კლასები იღებს მხარდაჭერას დამოკიდებულების თვისებებისაგან, რომელიც წარმოიშობა DependencyObject კლასისაგან.

Visual კლასი ერთეულოვანი ობიექტია, რომელიც აინკაფსულირებს ხატვის ინსტრუქციებს და დეტალებს, აგრეთვე საბაზო ფუნქციებს. WPF–ის ინტერფეისული ელემენტები წარმოქმნილ უნდა იქნას Visual კლასისაგან.

მომხმარებელთა ყველა მართვის ელემენტი შთამომავალია UIElement ან FrameworkElement კლასების.



ნახ.14.5. WPF ფუნდამენტური კლასები:
აბსტრაქტული (მრგვალკუთხა) და კონკრეტული (მართკუთხა)

UIElement კლასი უზრუნველყოფს ისეთ ფუნქციონალობას, როგორიცაა დაკომპლექტება, შეტანა, ფოკუსი და მოვლენები.

FrameworkElement-კლასი UIElement-ის ფუნქციონალობას უმატებს ველების განსაზღვრას, გასწორებას, მონაცემთა დაკავშირების მხარდაჭერას, ანიმაციას და სტილებს.

Shape კლასი საბაზოა ისეთი გეომეტრიული ფიგურების ასაგებად, როგორიცაა მართკუთხედი, ელიფსი, მრავალკუთხედი, წრფე და გზა.

Control კლასი განსაზღვრავს მართვის ელემენტებს, რომელთაც შეუძლია მომხმარებელთან ურთიერთობა. ესაა ღილაკები, სიები, ტექსტური ელემენტები.

ContentControl და ItemsControl კლასები საბაზოა მართვის ელემენტებისათვის, რომელთაც შეიძლება ჰქონდეს ერთადერთი მნიშვნელობა ან შესაბამისად მნიშვნე-ლობების კოლექცია.

Panel კლასი საბაზოა ყველა კონტეინერული ელემენტთა შემადგენლობისათვის, რომლებიც შეიცავს ერთ ან მეტ შვილობილ ელემენტს.

14.3. ინტერფეისის დაკომპლექტება

აპლიკაციის მომხმარებლის ინტერფეისის დაპროექტების დროს აუცილებელია ფანჯარაში (ფორმაზე) ან გვერდზე საჭირო მართვის ელემენტების ფორმირება და შესაბამისი თვისებების განსაზღვრა, ანუ შინაარსის ორგანიზების ჩატარება. ამ პროცესს უწოდებენ დაკომპლექტებას (შედგენას).

WPF-ში დაკომპლექტება ხორციელდება სხვადასხვა კონტეინერით. ყოველ მათგანს თავისი საკუთარი დაკომპლექტების ლოგიკა აქვს. ზოგი ალაგებს ელემენტებს მიმდევრობით სტრიქონში, ზოგი ალაგებს მათ უხილავი უჯრების ბადეში.

ფანჯარას და გვერდს WPF-ში შეუძლია შეიცავდეს მხოლოდ ერთ ელემენტს - კონტეინერს. კონტეინერში შეიძლება განთავსდეს მომხმარებლის ინტერფეისის განსხვავებული ელემენტები და სხვა კონტეინერები. WPF-ში განლაგება განისაზღვრება გამოყენებული კონტეინერის ტიპით. ეს კონტეინერებია; პანელები, წარმოებული System.Windows.Controls.Panel აბსტრაქტული კლასიდან.

აპლიკაციებში გამოიყენება შემდეგი კლასები:

- Grid და UniformGrid – განალაგებს ელემენტებს უხილავი ცხრილის სტრიქონებსა და სვეტებში, შესაბამისად;
- StackPanel – განალაგებს ელემენტებს ჰორიზონტალურ ან ვერტიკალურ სვეტში.
- WrapPanel – განალაგებს ელემენტებს ხელმისაწვდომ სივრცეში, ერთ სტრიქონად ან სვეტად;
- DockPanel – განალაგებს ელემენტებს ერთ-ერთი სასაზღვრო გვერდის შეფარდებით;
- Frame – ანალოგიურია StackPanel-ის, მაგრამ უფრო მოსახერხებელია გვერდების გადასასვლელების ორგანიზებისათვის.

Grid არის WPF-ის ყველაზე მძლავრი კონტეინერი. ის, რასაც სხვა კონტეინერები ასრულებს ცალკ-ცალკე, შეიძლება Grid-ში შერულდეს. იგი იდეალური ინსტრუმენტია ფანჯრის (გვერდის) დასაყოფად შედარებით მცირე ზომის არეებად, რომელთა მართვა განხორციელდება სხვა პანელებით. Grid ანაწილებს ელემენტებს უხილავი ბადის სტრიქონებსა და სვეტებში. ბადის ერთ უჯრაში მიზანშეწონილია ერთი ელემენტის მოთავსება, რომელიც, საჭიროებისამებრ, თვითონ შეიძლება იყოს სხვა კონტეინერი, რომელშიც განლაგდება საკუთარი მართვის ელემენტთა ჯგუფი.

StackPanel - ერთ-ერთი უმარტივესი კონტეინერია. იგი ალაგებს თავის შვილობილ ელემენტებს ერთ სტრიქონში ან სვეტში.

UniformGrid კონტეინერი, Grid-ისგან განსხვავებით, მოითხოვს მხოლოდ სტრიქონების და სვეტების რაოდენობის მითითებას, და აფორმირებს ერთი ზომის უჯრებს, რომელთაც დაკავებული აქვს ფანჯრის (გვერდის) ან ჩადგმული კონტეინერის ელემენტის მთელი ხელმისაწვდომი არე.

WrapPanel აწესრიგებს ელემენტების განლაგებას პანელზე Orientation თვისების შესაბამისად, ჰორიზონტალურად (Horizontal) ან ვერტიკალურად (Vertical). სტრიქონის ან სვეტის შევსების შემდეგ მომდევნო ელემენტი გადადის ახალ სტრიქონზე ან სვეტზე.

DockPanel უახლოვებს მართვის ელემენტებს მის რომელიმე მხარეს, Dock თვისების შესაბამისად, რომელიც იქნება Left, Right, Top ან Bottom. ელემენტის სიმაღლე განისაზღვრება MaxHeight პარამეტრით.

Frame მართვის ელემენტია შიგთავსისთვის, რომელიც იძლევა შესაძლებლობას შიგთავსზე ან მის ასახვაზე გადასასვლელად. Frame შეიძლება მოთავსდეს სხვა შიგთავსის შიგნით, როგორც სხვა ელემენტები. შიგთავსი შეიძლება იყოს .NET Framework-ის და HTML-ფაილების ნებისმიერი ტიპი. ჩვეულებისამებრ, Frame გამოიყენება შიგთავსის ჩასაწყობად, რომელიც განსაზღვრავს გადასასვლელებს გვერდებზე.

დაკომპლექტების თვისებები განისაზღვრება კონტეინერით, მაგრამ შვილობილი ელემენტებიც ახდენს მასზე გალენას. დაკომპლექტების პანელები მუშაობს შვილობილ ელემენტებთან შეთანხმებით, შემდეგი თვისებების საუძველზე:

- HorizontalAlignment და VerticalAlignment - განსაზღვრავს, თუ როგორ პოზიციონირებს შვილობილი ელემენტი კომპლექტის შიგნით, როცა არსებობს დამატებითი ჰორიზონტალური/ვერტი-კალური სივრცე;
- Margin - ელემენტის ირგვლივ ამატებს ცარიელ სივრცეს;
- MinWidth / MaxWidth აყენებს ელემენტის მაქსიმალურ ზომებს;
- Width и Height – ცხადად აყენებს ელემენტის ზომებს.

WPF-ში არსებობს კონტეინერები, რომლებსაც აქვს მართვის ელემენტები მოცემული კოორდინატების შესაბამისად, რომლებიც ზომებითაა მოცემული. ეს კონტეინერებია Canvas და InkCanvas. ისინი ძირითადად გამოიყენება გრაფიკული პრიმიტივებისა და ფიგურების ვიზუალიზაციისათვის.

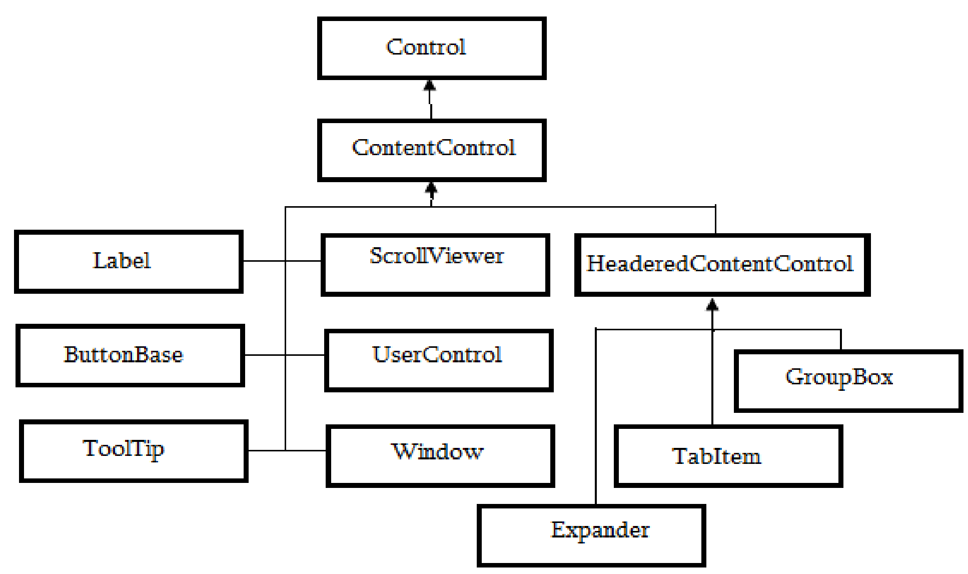
14.4. მართვის ელემენტები და შიგთავსები

მართვის ელემენტების დანიშნულებაა მომხმარებელის ინტერაქტიული კავშირის მხარდაჭერა. მათ შეუძლია კლავიატურიდან ან მაუსიდან ფოკუსის და შემავალი მონაცემების მიღება.

შიგთავსის მართვის ელემენეტები სპეციალიზებული ტიპის მართვის ელემენტებია, რომლებიც ინახავს განსაზღვრულ შიგთავსს - ერთ ან რამდენიმე ელემენტს. შიგთავსის ყველა მართვის ელემენტი მემკვიდრეა ContentControl კლასის (ნახ.14.6).

ContentControl კლასი მემკვიდრეა System.Windows.Control კლასის, რომელიც მას და მის შვილობილ კლასებს ანიჭებს საბაზო მახასიათებლებს, რომლებიც:

- საშუალებას იძლევა განისაზღვროს შიგთავსი მართვის ელემენტის შიგნით;
- საშუალებას იძლევა განისაზღვროს გადასვლების მიმდევრობა Tab-კლავიშით;
- ხელს უწყობს ფონის, წინა პლანის და ჩარჩოს ხატვას;
- ხელს უწყობს ტექსტური შინაარსის ზომის და შრიფტის ფორმატირებას.



ნახ.14.6. შიგთავსის მართვის ელემენტების იერარქია

მართვის ელემენტებს აქვს ფონი (ელემენტის ზედაპირი) და ტექსტი (ფონზე მოთავსებული). მათი ფერები WPF-ში განისაზღვრება Background და Foreground თვისებებით. ისინი იყენებს ფუნჯს - ობიექტი Brush. იგი უზრუნველყოფს ფონისა და ტექსტის შევსებას მთლიანი ფერით (ფუნჯის კლასი SolidColorBrush), ან გრადიენტულით, მაგალითად, LinearGradientBrush ფუნჯის კლასის დახმარებით.

მაგალითად, ღილაკის ფონის მისაცემად Brush ობიექტით აუცილებელია SolidColorBrush ფუნჯის Color თვისებას მიენიჭოს ფერის მნიშვნელობა, მაგალითად, Blue - ლურჯი.

```
<Button> ღილაკი A
  <Button.Background>
    <SolidColorBrush Color="Blue"></SolidColorBrush>
  </Button.Background>
</Button>
```

WPF-ში შესაძლებელია მოკლე ფორმის გამოყენებაც. მაგალითად, იმავეს ექნება ასეთი სახე:

```
<Button Background="Black"
     Foreground="Red">ღილაკი A</Button>
```

ამ დროს WPF-ის სინტაქსური ანალიზატორი ავტომატურად შექმნის SolidColorBrush ობიექტებს მითითებული ფერებით და გამოიყენებს ამ ობიექტებს ფონისა და ტექსტისათვის.

თუ საჭიროა გრადიენტული ტიპის LinearGradientBrush ფუნჯის გამოყენება, მაშინ ფუნჯის ობიექტი უნდა შეიქმნას დამოუკიდებლად:

```
<Button>ღილაკი A
  <Button.Background>
    <LinearGradientBrush>
    <LinearGradientBrush.GradientStops>
      <GradientStop Offset="0.0" Color="Blue"></GradientStop>
      <GradientStop Offset="1.0" Color="Red"></GradientStop>
    </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Button.Background>
</Button>
```

ფუნჯის დახმარებით იხატება ჩარჩო მართვის ელემენტის ირგვლივ. ეს ხორციელდება BorderBrush და BorderThickness თვისებებით. ამ დროს BorderBrush თვისება იღებს არჩეულ ფუნჯს, ხოლო BorderThickness – ჩარჩოს სიგანეს:

```
<Button Width="80" Height="30" Background="Red"
BorderBrush="Blue" BorderThickness="5">ღილაკი A</Button>
```

WPF ტექნოლოგია უზრუნველყოფს გამჭვირვალობას. თუ ფორმაზე ან გვერდზე განლაგებულია რამდენიმე ელემენტი ერთიმეორეზე, და მათთვის განსაზღვრულია სხვადსხვა ხარისხის გამჭვირვალეობა, მაშინ ქვედა ელემენტები (წარწერებით) გამოჩნდება ზედა ელემენტების შიგნით. ეს საშუალებას იძლევა შეიქმნას მრავალშრიანი ანომაციური ობიექტები. გამჭვირვალობა შეიძლება გადმოიცეს ორი ხერხით:

- თვისებით Opacity (არაგამჭვირვალე), რომელიც იღებს მნიშვნელობას საზღვრებში 0-1. აქ 1 არის სრულიად არაგამჭვირვალე, ხოლო 0 - სრულად გამჭვირვალე;
- ნახევრადგამჭვირვალე ფერის გამოყენებით, ალფა-არხის მნიშვნელობის მიწოდებით. თუ ალფა-არხის მნიშვნელობა 255-ზე ნაკლები, მაშინ ის ნახევრადგამჭვირვალეა.

Control კლასი განსაზღვრავს შრიფტების თვისებათა ერთობლიობას:

- FontFamily – შრიფტის სახელი მართვის ელემენტში;
- FontSize – შრიფტის ზომა (1/96 დიუმი);
- FontStyle – ტექსტის დახრის მითითება;
- FontWeight – ტექსტის წონა;
- FontStretch – სიდიდე, რომლის მიხედვითაც ტექსტი გაიწელება ან შეიკუმშება.

მართვის ელემენტისთვის ტექტსის შრიფტის არჩევისას უნდა მიეთითოს შრიფტების ოჯახის სრული სახელი:

```
<Button FontFamily="Sylfaen" FontSize="18">ღილაკი A</Button>
```

ნახევრადსქელი, დახრილი შრიფტის ასარჩევად უნდა მიეცეს შემდეგი თვისებები:

<Button FontFamily="Sylfaen"

FontSize="18" FontStyle="Italic" FontWeight="Bold"> ღილაკი A</Button>

WPF-ის მრავალი მართვის ელემენტი არის შიგთავსის მართვის ელემენტები. ესაა: Label, Button, CheckBox და RadioButton.

Label - ჭდე: არის შიგთავსის უმარტივესი მართვის ელემენტი. ამ ფუნქციისთვის გამოიყენება Target თვისება, რომელსაც ენიჭება მისაბმელი გამოსახულება. აქ უნდა მიეთითოს სხვა მართვის ელემენტი, რომელზეც გადავა ფოკუსი სწრაფი წვდომის ღილაკის დაჭერისას.

<Label Target="{Binding ElementName = txtA}">არჩევა_A</Label>

<TextBox Name="txtA">ტექსტის არჩევა</TextBox>

ჭდის ტექსტში რომელიმე ასოზე ხაზგასმა მიუთითებს სწრაფი წვდომის ღილაკზე. ასეთ მნემონიკურ ბრძანებებში ერთდროულად მუშაობს <Alt+A>, სადაც **A** ხაზგასმული ასოა. ჩვენი მაგალითისათვის ფოკუსი გადაეცემა მართვის ელემენტს TextBox, რომლის სახელია txtA.

WPF-ში განსაზღვრულია სამი კლასი: Button, CheckBox და RadioButton, რომლებიც ButtonBase კლასის მემკვიდრეებია. ButtonBase კლასი განსაზღვრავს Click მოვლენას და უმატებს იმ ბრძანებათა მხარდაჭერას, რომლებიც უზრუნველყოფს ღილაკების მიერთებას დანართის ამოცანებისათვის. აქვეა ClickMode თვისება, რომელიც განსაზღვრავს, თუ როდის აგენერირებს ღილაკი Click-მოვლენას მაუსის მოქმედების საპასუხოდ. გამოუცხადებლად ესაა ClickMode.Release მნიშვნელობა, რომელიც ნიშნავს მოვლენის გენერირებას მაუსის ღილაკის დაჭერის ან აშვების დროს. ClickMode.Press - მხოლოდ დაჭერის დროს, ClickModeHover - როცა მაუსის კურსორი ღილაკზე დადგება და შეჩერდება.

Button კლასი ამატებს ორ თვისებას: IsCancel და IsDefault. როცა IsCancel = true, მაშინ ღილაკი იმუშავებს როგორც ფანჯრის შეცვლა, <Esc>-ის მსგავსად. თუ IsDefault= true, მაშინ ღილაკი ითვლება აქტიურად გამოუცხადებლად.

CheckBox და RadioButton კლასები არის ToggleButton კლასის მემკვიდრეები, რომელიც ასახავს ღილაკს ორი მდგომარეობით: „დაჭერილია" და „აშვებულია". ამ კლასში არის მოვლენები: Checked, Unchecked და Intermediate, რომლებიც გენერირდება ღილაკის ჩართვის, ამორთვის ან განუსაზღვრელ მდგომარეობაში გადასვლისას.

CheckBox ღილაკისთვის ჩართვა ნიშნავს „ალმის" („ v ") დაყენებას. IsChecked თვისებას, რომელიც ToggleButton-იდანაა ნაანდერძევი, შეუძლია სამი მნიშვნელობის მიღება: true (ჩართული), false (გამორთული), null (განუსაზღვრელი, რომელიც ჩანს როგორც ჩამუქებული ფანჯარა, და გამოიყენება როგორც შუალედური მდგომარეობა). XAML-აღწერა CheckBox-ის ამ სამი მდგომარეობისა ნაჩვენებია ქვემოთ:

<CheckBox Height="16" Name="checkBox1"

```
    Width="120" IsChecked="False"
            ClickMode="Release">არჩევა A</CheckBox>
<CheckBox Height="16" Name="checkBox2"
    Width="120" IsChecked="True"
            ClickMode="Press">არჩევა B</CheckBox>
<CheckBox Height="16" Name="checkBox3"
    Width="120" IsChecked="{x:Null}"
            ClickMode="Hover">არჩევა C</CheckBox>
```

RadioButton ღილაკისთვის დამატებულია GroupName თვისება, რომელიც უზრუნველყოფს გადამრთველების განლაგების მართვას ჯგუფში. ჯგუფიდან შეიძლება მხოლოდ ერთი გადამრთველის არჩევა.

```
<GroupBox Header="რადიოღილაკების ჯგუფი" Height="100"
                              Name="groupBox1" Width="200">
  <StackPanel>
    <RadioButton Height="16" Name="radioButton2"
                 Width="120">არჩევა D</RadioButton>
    <RadioButton Height="16" Name="radioButton1"
                 Width="120">არჩევა E</RadioButton>
    <RadioButton Height="16" Name="radioButton3"
                 Width="120">არჩევა F</RadioButton>
  </StackPanel>
</GroupBox>
```

კონტექსტური ფანჯრის (მაუსის კურსორის მიტანისას მართვის ელემენტზე გამოჩნდება პატარა ფანჯარა ტექსტით, ხოლო კურსორის მოცილებისას - გაქრება) გამოტანა შეიძლება ToolTip კლასის ან თვით მართვის ელემენტის ToolTip თვისების გამოყენებით.

```
<Button ToolTip="ღილაკი A კონტექსტური ტექსტით "></Button>
```

ScrollViewer მართვის ელემენტი უზრუნველყოფს შიგთავსის დათვალიერებას ტექსტის ეკრანზე გადახვევით. ამ ელემენტით იქმნება ტექსტების გადახვევის შესაძლებლობის მქონე პანელები.

UserControl ელემენტის დანიშნულებაა მომხმარებლის კონტროლის ელემენტების შექმნა, რომლებშიც ერთიანდება რამდენიმე სხვა ელემენტი.

Windows ელემენტი შიგთავსის მართვის ელემენტია და გამოიყენება დანართის ყველა ფანჯრის შესაქმნელად.

HeaderedContentControl ელემენტი ContentControl კლასის მემკვიდრეა და მშობელია იმ ელემენტებისა, რომელთაც შიგთავსის გარდა აქვს სათაურის არე.

GroupBox ელემენტი არის ფანჯარა სათაურით და გამოიყენება დაკავშირებული ელემენტების დაჯგუფებისათვის, მაგალითად, რადიოღილაკები.

TabItem მართვის ელემენტი ასახავს გვერდს TabControl კლასისათვის. ეს ელემენტი ასახავს იმ ჩადგმულ გვერდს TabControl პანელში, რომელიც აქტიურია ამ წუთში.

Expander მართვის ელემენტი უზრუნველყოფს შიგთავსის განსაზღვრული უბნის გამოჩენას ან დამალვას.

14.5. ტექსტური მართვის ელემენტები

WPF-ში განსაზღვრულია შემდეგი ტექსტური მართვის ელემენტები: TextBlock, TextBox, RichTextBox და PasswordBox.

PasswordBox ელემენტი მემკვიდრეა Control კლასის, ხოლო ელემენტები TextBox და RichTextBox - მემკვიდრეა TextBase კლასის.

TextBlock ელემენტი გამოიყენება მცირე ზომის ტექსტისათვის.

TextBox ელემენტი ინახავს ტექსტის სტრიქონს.

RichTextBox ელემენტს შეუძლია FlowDocument-ის შიგთავსის შენახვა, რომელშიც შეიძლება იყოს ელემენტთა რთული კომბინაცია.

PasswordBox ელემენტი შეიცავს ტექსტის სტრიქონს, იყენებს SecureString ელემენტს დასაცავად ზოგიერთი სახის თავდასხმისაგან.

TextBox ელემენტი, ჩვეულებრივად ინახავს ერთ სტრიქონს: <TextBox Name="txtA">ტექსტის არჩევა</TextBox>.

თუ საჭიროა მრავალსტრიქონიანი წარმოდგენის შექმნა, მაშინ თვისებას TextWrapping მიენიჭება Wrap მნიშვნელობა. ასეთი TextBox-ისთვის შეიძლება მინიმალური და მაქსიმალური სტრიქონების რაოდენობის მითითება, MinLines და MaxLines თვისაებათა გამოყენებით.

14.6. სიების მართვის ელემენტები

ListBox და ComboBox - სიების მართვის ელემენტებია. ისინი ItemsControl კლასის მემკვიდრეებია Selector კლასის გავლით.

ItemsControl კლასი საბაზოა სიების მართვის ყველა ელემენტისათვის და ითვალისწინებს სიის ელემენტების შევსების ორ ხერხს. პირველში ხდება ელემენტის დამატება უშუალოდ Items კოლექციაში, რომლის დროსაც გამოიყენება კოდი ან XAML. მეორე ხერხით ხდება მონაცემთა მიბმა ItemsSource თვისებით (მონაცემთა წყარო).

Selector კლასს აქვს თვისებები, რომლებიც თვალყურს ადევნებს მოცემული დროის მომენტში გამოყოფილი სიის ელემენტს (SelectedItem) ან მის პოზიციას (SelectedIndex).

ელემენტთა დასამატებლად ListBox-ში შეიძლება ჩაიდოს ListBoxItem-ის ელემენტები ListBox-ში, როგორც ეს ქვემოთაა ნაჩვენები ფერთა სიის შესადგენად (მწვანე, ცისფერი, ყვითელი, წითელი):

```
<ListBox>
    <ListBoxItem>მწვანე</ListBoxItem>
    <ListBoxItem>ცისფერი</ListBoxItem>
    <ListBoxItem>ყვითელი</ListBoxItem>
    <ListBoxItem>წითელი</ListBoxItem>
</ListBox>
```

მართვის ელემენტი ListBox ინახავს თავის კოლექციაში ყოველ ჩადგმულ ობიექტს. ამავდროულად, ListBoxItem-ს შეუძლია არა მხოლოდ სტრიქონების შენახვა, არამედ ნებისმიერი თავისუფალი ელემენტის. მართვის ელემენტი ComboBox მსგავსია ListBox-ის, ოღონდ ვიზუალიზაციისათვის გამოიყენებს ჩამოშლად სიას, რომლიდანაც მომხმარებელი ირჩევს მხოლოდ ერთ ელემენტს.

14.7. სპეციალიზებული მართვის ელემენტები

WPF-ში არის მართის ელემენტები, რომლებიც იყენებს მნიშვნელობათა დიაპაზონებს: ScrollBar, ProgressBar და Slider. მართვის ეს ელემენტები მემკვიდრეებია RangeBase კლასის, რომელშიც განსაზღვრულია ისეთი თვისებები, როგორიცაა Value (ელემენტის მიმდინარე მნიშვნელობა), Minimum და Maximum დასაშვები მნიშვნელობები.

ScrollBar მართვის ელემენტი იძლევა გადახვევის ზოლს გადაადგილებადი ელემენტით, რომლის პოზიცია შეესაბამება განსაზღვრულ მნიშვნელობას.

ProgressBar მართვის ელემენტი უჩვენებს ხანგრძლივი ამოცანის შესრულების მსვლელობას.

Slider მართვის ელემენტი გამოიყენება რიცხვითი მნიშვნელობის მოსაცემად კურსორის გადაადგილების შესაბამისად გადახვევის სახაზავზე.

14.8. ბრძანებები

WPF-ის ბრძანებათა მოდელი იძლევა შემდეგ შესაძლებლობებს:

- მოვლენათა დელეგირება შესაფერის ბრძანებებზე;
- მართვის ელემენტის ჩართული მდგომარეობის შენარჩუნება სინქრონიზებული სახით შესაბამისი ბრძანების მდგომარეობის დახმარებით.

ბრძანებათა მოდელი შეიცავს შემდეგ კომპონენტებს:

- ბრძანებები, რომლებიც წარმოადგენს დანართის ამოცანას, მათი წვდომის თვალყურის დევნის მექანიზმით პროგრამის შესრულების დროს;
- ბრძანებათა მიბმა, რომელიც გულისხმობს ბრძანების მიერთებას დანართის ლოგიკასთან. იგი პასუხისმგებელია მომხმარებლის ინტერფეისის გარკვეული უბნის მომსახურებაზე;

- ბრძანების წყარო, რომელსაც იგი აინიცირებს;
- ბრძანების მიზნობრივი ობიექტები - დანართის ელემენტები, რომლებისთვისაც არის გამიზნული ეს ბრძანება.

WPF-ის ბრძანებათა კლასები ICommand ინტერფეისს უნდა უჭერდეს მხარს.

```
public interface ICommand
  {
     void Execute(object par);
     bool CanExecute(object par);
     event EventHandler CanExecuteChanged;
  }
```

მეთოდი Execute() განსაზღვრავს ბრძანების რეალიზაციას დანართში. ეს შეიძლება იყოს დანართის ლოგიკის კოდი ან მოვლენის ამოქმედება, რომელიც მუშავდება დანართის სხვა კლასში. მეთოდი CanExecute() აბრუნებს ინფორმაციას ბრძანების წვდომის მდგომარეობაზე. Execute() და CanExecute() მეთოდები შეიძლება გამოძახებულ იქნას **par** პარამეტრით, რომელიც გამოიყენება დამატებითი ინფორმაციის გადასაცემად.

CanExecuteChanged მოვლენა გამოიძახება ბრძანების წვდომის მდგომარეობის შეცვლისას. ამ შემთხვევისთვის გამოიძახება CanExecute() მეთოდი და მოწმდება ბრძანების მდგომარეობა.

WPF-ში არსებობს საბაზო ბრძანებათა ბიბლიოთეკა, რომელიც ხელმისაწვდომია შემდეგი სტატიკური კლასების სტატიკური თვისებებიდან:

- ApplicationCommands – ზოგადი ბრძანებები, მაგალითად, Create, Open, Colpy, Store და სხვა;
- Navigation Commands – ბრძანებები ნავიგაციისათვის;
- Editing Commands – ბრძანებები დოკუმენტაციის რედაქტირებისათვის;
- Media Commands – მულტიმედიასთან სამუშაო ბრძანებები.

ბრძანების ინიციალიზაციისათვის საჭიროა ბრძანების წყაროს გამოყენება, ხოლო მასზე საპასუხოდ - ბრძანების მიბმა მისი შესრულების გადამისამართებასთან ჩვეულებრივ მოვლენის დამმუშავებელზე.

დავუშვათ, რომ **შექმნა** ბრძანების წყარო არის ღილაკი New. Command თვისების დახმარებით ღილაკს ვაბამთ New ბრძანებას.

```
<Button Name="New" Content="შექმნა" Width="200" Command="New"/>
```

შემდეგ ბიჯზე აუცილებელია ბრძანების მიბმა Window ფანჯარასთან.

```
<Window.CommandBindings>
      <CommandBinding Command="New"
            Executed="CommandBinding_Executed"
            CanExecute="CommandBinding_CanExecute"/>
</Window.CommandBindings>
```

მიბმის დროს CommandBinding ობიექტისათვის, გარდა New ბრძანებისა, საჭიროა დამმუშავებლების მითითება Execute() და CanExecute() მეთოდებისთვის, შესაბამისად Executed და CanExecute თვისებებით. დამმუშავებელი CommandBinding_Executed რეალიზებას გაუკეთებს **შექმნა**-ბრძანების ბიზნესლოგიკას, ხოლო დამმუშავებელი CommandBinding_CanExecute კი - ბრძანების წვდომის შემოწმებას.

მომხმარებლის ბრძანების შექმნისას მიზანშეწონილია RoutedCommand კლასის გამოყენება, რომელიც რეალიზაციას უკეთებს ICommand ინტერფეისს.

საილუსტრაციოდ განვიხილოთ **რედაქტირება**-ბრძანების კონსტრუირება. შევქმნათ DataCommands კლასი.

```
public class DataCommands
{
    private static RoutedCommand edit;
    public static RoutedCommand Edit
   {
      get { return DataCommands.edit; }
   }
  static DataCommands()
   {
    InputGestureCollection inputs = new InputGestureCollection();
    inputs.Add(new KeyGesture(Key.E, ModifierKeys.Control, "Ctrl+E"));
    Edit = new RoutedCommand("Edit", typeof(DataCommands), inputs);
   }
}
```

DataCommands კლასში გამოცხადებულია RoutedCommand კლასის ეგზემპლარის სტატიკური ველი (edit) და თვისება (Edit). სტატიკური კონსტრუქტორი ქმნის **შექმნა**-ბრძანების (Edit) ეგზემპლარს, იყენებს რა InputGestureCollection-კლასის (inputs)-ეგზემპლარს. ეს კლასი არის KeyGesture ობიექტების მოწესრიგებული კოლექცია, რომლებიც განსაზღვრავს „ცხელ" ღილაკს ბრძანების გამოსაძახებლად. მაგალითში ესაა „Ctrl+E". შემდეგ აუცილებელია ბრძანებისათვის მოინახოს წყარო და მიებას ბრძანება, როგორც ეს ადრე იყო განხილული.

14.9. რესურსები

ობიექტური რესურსი - ესაა .NET-ობიექტი, რომელიც განისაზღვრება ერთ ადგილას და გამოიყენება რამდენიმე სხვა ადგილას დანართში. ეს რესურსი განისაზღვრება XAML ფორმატირების ენაში [4,8]. ობიექტურ რესურსებს აქვს რიგი მნიშვნელოვანი მახასიათებელი:

- ეფექტურობა კოდის მრავალჯერადი გამოყენების პრინციპის რეალიზაციის გამო;

- მოხერხებული თანხლება ფორმატირების დაბალი დონის დეტალების გადატანის გამო ერთ ცენტრალურ ადგილას;
- ადაპტაციის უნარი ინფორმაციის დინამიკურად ცვლილების შესაძლებლოიბის გამო, რომელიც კოდისაგან გამოყოფილია.

ყოველი ელემენტი, რომელიც მემკვიდრეა FrameworkElement კლასის (ნახ.1.2) შეიცავს Resources თვისებას, რომელშიც ინახება რესურსების Resourcesdictionary-ლექსიკონური კოლექცია. აქ შეიძლება ინახებოდეს ნებისმიერი ტიპის ობიექტი. ყოველ ელემენტს აქვს წვდომა როგორც საკუთარი რესურების, ისე თავისი მშობლების რესურსების კოლექციებთან. ხშირად რესურსები ინახება ფანჯრის ან გვერდის დონეზე მათი ერთობლივად გამოყენების უზრუნველსაყოფად ყველა მართვის ელემენტის მიერ.

რესურსები განისაზღვრება Resource სექციაში შესაბამისი ელემენტისათვის, მაგალითად, ფანჯრისათვის:

```
<Window x:Class="MyFirstWpfProject.MainWindow" ... >
        <Window.Resources>
                ....
        </Window.Resources>
....
</Window>
```

ყოველი რესურსი ჩაიწერება ასეთი სახით:

```
<TypeResource  x:Key="რესურსის გასაღები" >
        ...
</ TypeResource >
```

(TypeResource)- ობიექტის თვისებების მისაცემად შეიძლება გამოყენებულ იქნას ატრიბუტთა სინტაქსი ან ელემენტთა თვისებათა სინტაქსი. XAML ელემენტები მიმართავს რესურსებს გასაღებით (KeyResource), რომელიც განისაზღვრება ატრიბუტით x:Key. რესურსის გასაღები მოიცემა StaticResource-ს გამოყენებით.

მაგალითად, დანართში შექმნილია გრადიენტული ფუნჯი ღილაკის ფერით შესავსებად, რომელიც გამოყენებულ უნდა იქნას ინტერფეისის სხვა ელემენტებში. მაშინ გრადიენტული ფუნჯი განსაზღვრული უნდა იყოს როგორც რესურსი:

```
<Window.Resources>
   <LinearGradientBrush x:Key="BrushButton"
                              EndPoint="0,1" StartPoint="0,0">
      <GradientStop Color="#FFD2E995"  Offset="0.9"/>
      <GradientStop Color="#FF74EDB3"  Offset="0.55"/>
      <GradientStop Color="#FF72D8A7"  Offset="0.75"/>
      <GradientStop Color="#FF233CC6"  Offset="0.25"/>
   </LinearGradientBrush>
```

```
</Window.Resources>
<StackPanel>
    <Button x:Name="New" Content="შექმნა" Width="200"
            Command="New" Margin="154.5,0"
             Background="{StaticResource BrushButton}"/>
</StackPanel>
```

გრადიენტულ ფუნჯს აქვს LinearGradientBrush ტიპი და მისთვის მოცემულია გასაღები BrushButton. ღილაკის XAML-აღწერაში ატრიბუტი Background განისაზღვრება გაფართოებული ფორმატით {StaticResource BrushButton}.

ჩვენ მაგალითში გამოყენებულია სტატიკური რესურსი. იგი ამოიღება რესურსების კოლექციიდან მხოლოდ ერთხელ. ამასთანავე, ნებისმიერი ცვლილებები რესურსის ობიექტში შეიმჩნევა მყისიერად. თუ დანართის ფუნქციონირების პროცესში შექმნილი რესურსი შეიძლება შეიცვალოს, მაშინ გამოყენებულ უნდა იქნას დინამიკური რესურსი - DynamicResource. იგი განისაზღვრება რესურსების კოლექციაში ყოველთვის, როცა ამის საჭიროება აღმოცენდება.

რესურსები შეიძლება განისაზღვროს ელემენტის, კონტეინერის, ფანჯრის ან გვერდის, დანართის, სისტემის ან ნაკრების დონეზე.

რესურსების ხელმეორედ გამოსაყენებლად იქმნება რესურსების ლექსიკონები. ესაა XAML-ფაილი რესურსების შესანახად. თუ მაგალითად, ჩვენ მიერ შექმნილი გრადიენტული ფუნჯის რესურსს გადავიტანთ ლექსიკონში, მაშინ რესურსების ლექსიკონის MyDictionary.xaml ფაილს ექნება შემდეგი სახე:

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
      <LinearGradientBrush x:Key="BrushButton"
                              EndPoint="0,1" StartPoint="0,0">
        <GradientStop Color="#FFD2E995" Offset="0.9"/>
        <GradientStop Color="#FF74EDB3" Offset="0.55"/>
        <GradientStop Color="#FF72D8A7" Offset="0.75"/>
        <GradientStop Color="#FF233CC6" Offset="0.25"/>
    </LinearGradientBrush>
</ResourceDictionary>
```

რესურსების ლექსიკონის გამოსაყენებლად იგი გაერთიანებულ უნდა იქნას რესურსების კოლექციასთან, მაგალითად ფანჯრის:

```
<Window.Resources>
   <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
```

```
        <ResourceDictionary
                        Source="MyDictionary.xaml" />
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Window.Resources>
```

რესურსების ლექსიკონის გაერთიანება სხვა ობიექტის რესურსეთან წარმოებს ResourceDictionary.MergedDictionaries თვისების მოცემით. MergedDictionaries კოლექცია - ესაა ResourceDictionary ობიექტების კოლექცია, რომლებიც უნდა დაემატოს რესურსების კოლექციას. დასამატებელი რესურსების ლექსიკონები მოიცემა როგორც Source ატრიბუტის მნიშვნელობები. განხილულ მაგალითში ესაა რესურსების ლექსიკონის ფაილი MyDictionary.xaml.

14.10. სტილები

სტილები - ესაა თვისებათა მნიშვნელობების კოლექციები, რომლებიც გამოიყენება ელემენტისთვის. ისინი საშუალებას იძლევა განისაზღვროს ფორმატირების მახასიათებლების ზოგადი ერთობლიობა და გამოყენებულ იქნას მთელი დანართის ფარგლებში შეთანხმებადობის უზრუნველსაყოფად. WPF-ში სტილებს შეუძლია დამოკიდებულებათა ნებისმიერი თვისების დაყენება. მათი გამოყენება შეიძლება რომელიმე ელემენტის ვიზუალური ქცევის სტანდარტიზაციისათვის. WPF-ის სტილებს აქვს ტრიგერები, რომლებიც იძლევა საშუალებას ელემენტის სტილის შესაცვლელად სხვა თვისებების შეცვლის დროს. სტილები იყენებს შაბლონებს მართვის ელემენტების სტანდარტული ვიზუალური წარმოდგენის ხელახლა განსაზღვრისათვის.

სტილი იქმნება Style კლასის ბაზაზე, რომლის თვისებები მოცემულია 14.1 ცხრილში.

Style კლასის თვისებები **ცხრ.14.1**

სახელი	აღწერა
BasedOn	აბრუნებს ან იძლევა განსაზღვრულ სტილს, რომელიც საბაზოა მიმდინარე სტილისათვის
Dispatcher	აბრუნებს Dispatcher ობიექტს, რომელთანაც კავშირშია ეს DispatcherObject ობიექტი
IsSealed	აბრუნებს მნიშვნელობას, რომელიც მიუთითებს, არის თუა არა მხოლოდ წაკითხვის სტილი
Resources	აბრუნებს ან იძლევა რესურსების კოლექციას, რომლებიც გამოყენებული იქნება მოცემული სტილის ხილვადობის არეში
Setters	აბრუნებს Setter და EventSetter ობიექტების კოლექციას
TargetType	აბრუნებს ან იძლევა ტიპს, რომლისთვისაც დანიშნულია ეს სტილი

Triggers	აბრუნებს TriggerBase ობიექტთა კოლექციას, რომლებიც იყენებს თვისებების მნიშვნელობებს მოცემული პირობრბის საფუძველზე

მაგალითად, დანართში საჭიროა მრავალჯერადი გამოყენების ღილაკები, რომელთაც აქვს ღია-ცისფერი ფონი და ცისფერი ჩარჩო.

```
<Style x:Key="ButtonStyle" TargetType="Button">
     <Setter Property="Background"  Value="LightBlue"/>
     <Setter Property="BorderBrush" Value="Blue" />
</Style>
```

სახელი მიეცემა ატრიბუტით x:Key (ButtonStyle). ატრიბუტი TargetType განსაზღვრავს ტიპს, რომლისთვისაც გამოყენებულ იქნება სტილი (Button). ყოველი Setter-ობიექტი აყენებს ელემენტში ერთ თვისებას, რომელიც აუცილებლად უნდა იყოს დამოკიდებულების თვისება.

თვისებების დაყენება წარმოებს Property ატრიბუტის დახმარებით, რომელიც განსაზღვრავს თვისების სახელს და Value - მის მნიშვნელობას.

სტილში შეიძლება EventSetters ობიექტის დამატება, რომელსაც მიებმება მოვლენა გარკვეული დამმუშავებლისთვის.

ტრიგერები იძლევა სტილის შეცვლის უფლებას განსაზღვრული პირობების შესრულებისას.

დავამატოთ ButtonStyle ღილაკის სტილში ტრიგერი, რომელიც დააფორმირებს წითელ ფონს, როცა მაუსის კურსორი დადგება ღილაკზე.

```
<Style x:Key="ButtonStyle" TargetType="Button">
   <Setter Property="Background"
                                 Value="LightBlue"/>
   <Setter Property="BorderBrush" Value="Blue" />
   <Style.Triggers>
     <Trigger Property="IsMouseOver" Value="true">
        <Setter Property="Background" Value="Red" />
     </Trigger>
   </Style.Triggers>
</Style>
```

ტრიგერში მითითებული უნდა იყოს მაიდენტიფიცირებელი თვისება (ჩვენ მაგალითში IsMouseOver), რომელიც უნდა ვაკონტროლოთ, და მნიშვნელობა, რომელსაც უნდა ველოდოთ (მაგალითად, true). როცა გამოჩნდება აუცილებელი მნიშვნელობა, ყენდება თვისება, რომელიც განისაზღვრება Setter ობიექტით.

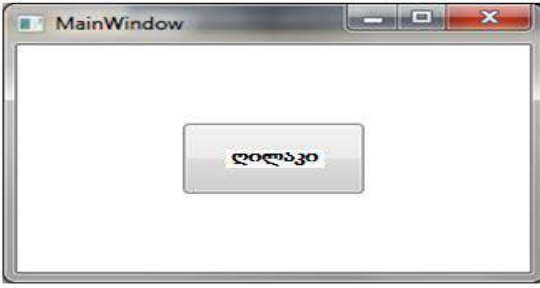
ჩვენ მაგალითში, როცა IsMouseOver = true, ანუ მაუსის კურსორი მიყვანილია ღილაკზე, მაშინ Background თვისებას მიენიჭება მნიშვნელობა Red, ანუ ღილაკის ფონი

ხდება წითელი. როცა კურსორი გამოდის ღილაკის ზონიდან, მაშინ ტრიგერის ამუშავების პირობა ირღვევა და ღილაკის ფონი გადადის საწყის მდგომარეობაში.

თვისების ცვლილების მომლოდინე ტრიგერის გარდა არსებობს მოვლენათა ტრიგერები (EventTrigger), რომლებიც ელოდება განსაზღვრულ მოვლენათა აღმოცენებას.

14.11. შაბლონები

WPF-აპლიკაციების დაპროექტების დროს ფანჯრები ან გვერდები არის კონტეინერები, რომლებშიც განლაგებულია სხვა კონტეინერები და ინტერფეისის სხვადასხვა ელემენტი (ჭდეები, ტექსტური ბლოკები, ღილაკები, სიები და სხვა მართვის ელემენტები). მაგალითად, ფანჯარა ერთი ღილაკით (ნახ.14.6).

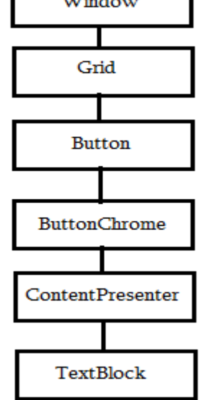


ნახ.14.6

XAML-დოკუმენტი, შექმნილი ფანჯრისათვის შემდეგი სახისაა:

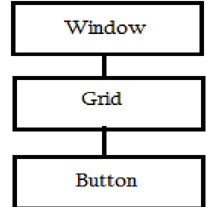
```
<Window x:Class="WpfApplication1.MainWindow"
     xmlns="http://schemas.microsoft.com/winfx/2006/
                                                  xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Title="MainWindow" Height="200" Width="300">
     <Grid>
       <Button Content="ღილაკი" Width="100"  Height="50"/>
     </Grid>
</Window>
```

ფანჯრის XAML-დოკუმენტში მოცემულია სამი ობიექტი: Window, Grid და Button. ეს ელემენტები ქმნის ფანჯრის ლოგიკურ ხეს (ნახ.14.7).



ნახ.14.7. ფანჯრის ლოგიკური ხე

WPF-ში ლოგიკური ხე დეტალიზირდება *ვიზუალირი ხის* დახმარებით, რომელიც ასახავს ლოგიკური ხის ელემენტებს უფრო მცირე ფრაგმენტების სახით (ნახ.14.8).



მაგალითად, ღილაკი, ვიზუალური ხის დონეზე, ინკაპსულირებულია მართკუთხედის სახით, რომელიც შეიცავს საზღვარს (ButtonChrome კლასი), შინაარსს (ContentPresenter კლასი) და ბლოკს ტექსტით (TextBlock კლასი).

ნახ.14.8. ფანჯრის ვიზუალური ხე

საერთოდ, არსებობს არაერთი ხერხი ლოგიკური ხის გაფართოებისა ვიზუალურ ხემდე. მაგალითად, ელემენტი Button, რომელიც შიგთავსის ელემენტია, შეუძლია შეიცავდეს ნებისმიერ სხვა ელემენტს, რაც აისახება მის ვიზუალურ ხეში.

ვიზუალური ხე საშუალებას იძლევა შეიცვალოს მისი ელემენტები სტილების დახმარებით და შეიქმნას ახალი შაბლონები მართვის ელემენტებისათვის.

WPF-ში არსებობს მართვის ელემენტების შაბლონები, მონაცეთა შაბლონები და პანელის სპეციალური შაბლონები.

ControlTemplate - მართვის ელემენტების შაბლონი გამოიყენება ამ ელემენტების გამოსახვის (წარმოდგენის) და მათი ვიზუალური ქცევის მოსაცემად.

DataTemplate - მონაცემთა შაბლონი გამოიყენება მონაცემთა ამოსაღებად ობიექტიდან და მათი ასახვისთვის შიგთავსის მართვის ელემენტში.

Hierarchical DataTemplate - პანელების შაბლონები გამოიყენება ელემენთა კომპლექტების მართვისათვის სიის ტიპის მართვის ელემენტებში.

14.11.1. მართვის ელემენტთა შაბლონები

მართვის ყოველ ელემენტს აქვს Template-თვისება, რომელიც განსაზღვრავს შაბლონს მისი ვიზუალიზაციისთვის. თუ ეს თვისება ცხადად არაა მოცემული, მაშინ გამოიყენება მართვის ელემენტის სტანდარტული შაბლონი, რომელიც WPF-შია განსაზღვრული. მართვის ელემენტის მომხმარებლის შაბლონის შესაქმნელად აუცილებელია განისაზღვროს ControlTemplate ობიექტი.

```
<ControlTemplate x:Key="შაბლონის_გასაღები"
                          TargetType="ელემენტის_ტიპი">
        ...
</ControlTemplate>
```

ატრიბუტი x:Key განსაზღვრავს გასაღებს, რომლითაც მიმართავენ შაბლონს, ხოლო TargetType განსაზღვრავს ელემენტის ტიპს, რომლისთვისაც იქმნება შაბლონი.

ღილაკის უმარტივესი შაბლონის მაგალითი მოცემულია ქვემოთ:.

```
<ControlTemplate x:Key="ButtonTemplate"
                          TargetType="Button">
```

```
    <Border BorderBrush="Blue" BorderThickness="3" CornerRadius="25"
                    Background="Azure" TextBlock.Foreground="Green">
        <ContentPresenter
            HorizontalAlignment="Center" VerticalAlignment="Center"/>
    </Border>
</ControlTemplate>
```

ContentPresenter ელემენტი განსაზღვრავს მართვის ელემენტის შიგთავსის ჩასმის ადგილს კონტეინერში (მაგალითად, კონტეინერი არის ჩარჩო – Border).

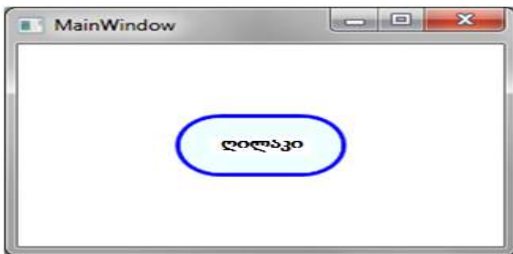
მართვის ელემენტის შაბლონი შეიძლება განთავსდეს ფანჯრის რესურსების კოლექციაში.

```
<Window.Resources>
        <ControlTemplate x:Key="ButtonTemplate" TargetType="Button">
        ...
        </ControlTemplate>
</Window.Resources>
```

შაბლონის მიბმა მართვის ელემენტთან ხორციელდება ამ ელემენტის Template თვისების მიცემით.

```
<Grid>
    <Button Content="ღილაკი" Width="100" Height="50"
                    Template="{StaticResource  ButtonTemplate}"/>
</Grid>
```

Template თვისება განისაზღვრება განლაგების (layout) გაფართოებით სტატიკურ რესურსზე (StaticResource) მიმართვისათვის რესურსის გასაღებით (ButtonTemplate). პროექტის ამუშავებით ღილაკს ექნება ასეთი სახე (ნახ.14.9).



ნახ.14.9. ღილაკის წარმოდგენა შაბლონის გამოყენებით

განხილული შაბლონი განსაზღვრავს ღილაკის სტატიკურ ვიზუალურ წარმოდგენას. დინამიკური ვიზუალური ქცევის დასამატებლად შეიძლება შაბლონთა ტრიგერების გამოყენება. მაგალითად, დავამატოთ შაბლონში ტრიგერი, რომელიც

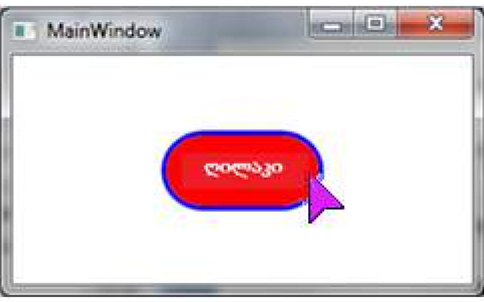
შეცვლის ღილაკის ფონის შევსებას წითელი ფერით, როცა მასზე მაუსის კურსორი დადგება, ამასთანავე შეცვლის წარწერის ფერს თეთრით.

```
<ControlTemplate x:Key="ButtonTemplate" TargetType="Button">
    <Border Name="Border" BorderBrush="Blue"  BorderThickness="3"
        CornerRadius="25" Background="Azure" TextBlock.Foreground="Green">
      <ContentPresenter HorizontalAlignment="Center"
                                  VerticalAlignment ="Center"/>
   </Border>
   <ControlTemplate.Triggers>
     <Trigger Property="IsMouseOver" Value="true">
        <Setter TargetName="Border"  Property="Background" Value="Red" />
        <Setter TargetName="Border" Property="TextBlock.Foreground" Value="White" />
      </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>
```

როცა ამოქმედდება მოვლენა MouseOver თვისება IsMouseOver გახდება true, შედეგად იმუშავებს ტრიგერი და შეიცვლება ღილაკის თვისებები Background და TextBlock.Foreground შესაბამისად Red და White-ით.

Background და TextBlock.Foreground თვისებების დაყენებისას TargetName-ელემენტში მიეთითება Border სახელი, რომელიც მინიჭებული აქვს შაბლონში Border-ელემენტს.

14.10 ნახაზზე ნაჩვენებია ღილაკის წარმოდგენა, როცა მასზე მაუსის კურსორია მოთავსებული.



ნახ.14.10

ტრიგერების გამოყენებით შეიძლება აუცილებელ მოვლენებზე განისაზღვროს მართვის ელემენტების ვიზუალური ქცევის რეაქცია.

14.11.2. მონაცემთა შაბლონები

მონაცეთა შაბლონი - ესაა XAML-ფორმატირების ნაწილი, რომელიც განსაზღვრავს, თუ როგორ უნდა აისახოს მონაცემების მიმაგრებული ობიექტი. მონაცემთა შაბლონს შეიძლება ჰქონდეს ელემენეტების ნებისმიერი კომბინაცია და უნდა შეიცავდეს ერთ ან მეტ მიბმის გამოსახულებას.

მონაცემთა შაბლონებს აქვს შემდეგი ელემენტები:

- შიგთავსის მართვის ელემენტები ContentTemplate თვისებით, რომელიც გამოიყენება ნებისმიერი Content-ში მოთავსებული შიგთავსის ვიზუალიზებისათვის;
- სიისებური მართვის ელემენეტები ItemTemplate თვისებით, კოლექციის ელემენტების ვიზუალიზაციისთვის, რომელიც მითითებულია როგორც მონაცემთა წყარო.

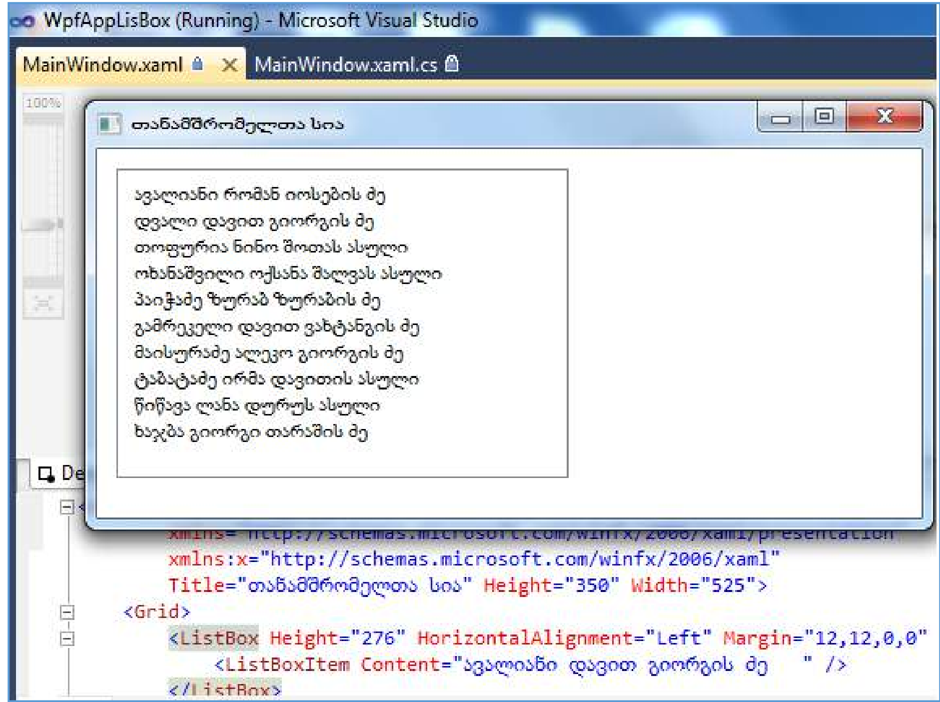
მაგალითად, დავამუშავოთ მონაცემთა შაბლონი ListBox სიისთვის. სიაში უნდა აისახოს მონაცემები თანამშრომლის შესახებ: გვარი, სახელი და მამის სახელი. მონაცემთა წყაროდ listBoxEmployees სიისთვის განიხილება კოლექცია Employees, რომელიც შეიცავს Employee კლასს. Employee კლასის თვისებებია:

- Surname –გვარი;
- Name - სახელი;
- Patronymic – მამის სახელი.

მონაცემთა შაბლონს listBoxEmployees სიისთვის აქვს შემდეგი სახე:

```
<ListBox Name="listBoxEmployees" >
   <ListBox.ItemTemplate>
     <DataTemplate>
       <StackPanel Orientation="Horizontal">
           <TextBlock Text="{Binding Path=Surname}" />
           <TextBlock Text="{Binding Path=Name}" />
           <TextBlock Text="{Binding Path=Patronymic}" />
       </StackPanel>
     </DataTemplate>
   </ListBox.ItemTemplate>
</ListBox>
```

პროგრამის ამუშავებით ფანჯარაში გამოიტანება სია (ნახ.14.11).



ნახ.14.11. სიის წარმოდგენა მომხმარებლური
მონაცემთა შაბლონით

მონაცემთა შაბლონის მრავალჯერადი გამოყენების მიზნით ის უნდა გადმოიცეს ფანჯრის ან დანართის რესურსის სახით, შაბლონის გასაღების მითითებით (ListBoxEmployee).

```
<Application.Resources>
        <DataTemplate x:Key="ListBoxEmployee" >
        ...
        </DataTemplate>
</Application.Resources>
```

ამავე დროს სიის XAML-აღწერაში ItemTemplate თვისებისთვის მოიცემა ფორმატის გაფართოება სტატიკურ რესურსზე.

```
<ListBox Grid.Row="1" Name="listBoxEmployees"
        ItemTemplate="{StaticResource ListBoxEmployee}" />
```

მონაცემთა შაბლონი არის ეფექტური ინსტრუმენტი მართვის ელემენტთა ვიზუალური ასახვის ცვლილებისათვის.

14.12. XAML ენის საფუძვლები

მომხმარებელთა ინტერფეისების აგება WPF- და Silverlight-დანართებისთვის (აპლიკაციებისთვის) ხორციელდება XAML (Extensible Application Markup Language - აპლიკაციების გაფართოებადი ფორმატირების ენა) ენის გამოყენებით. XAML-დოკუმენტი შეიცავს ფორმატს, რომელიც აღწერს დანართის ფანჯრის (ან გვერდის) გარეგან სახეს და ქცევას, ხოლო მასთან კავშირში მყოფი C# კოდის ფაილები კი - დანართის ლოგიკას. XAML-ენა უზრუნველყოფს დანართის დიზაინის პროცესის (გრაფიკული ნაწილი) გამოყოფას ბიზნესლოგიკის (პროგრამული კოდი) დამუშავების პროცესისგან, დიზაინერებსა და დეველოპერებს შორის [4,5].

WPF-ის XAML არის XML-ენის ქვესიმრავლე, გაფართოებული დამატებითი ფუნქციებით. იგი უზრუნველყოფს WPF-შიგთავსის აღწერას ისეთი ელემენტებით, როგორიცაა ვექტორული გრაფიკა, მართვის ელემენტები და დოკუმენტები.

XAML-ის საფუძველია XML და მისი სინტაქსი განისაზღვრება შემდეგი წესებით:

- XAML-დოკუმენტის ყოველი ელემენტი აისახება .NET კლასის რომელიღაც ეგზემპლარში. ასეთი ელემენტის სახელი ზუსტად შეესაბამება კლასის სახელს. მაგალითად, <Button> ელემენტი ემსახურება WPF-ინსტრუქციას Button-კლასის ობიექტის აგების მიზნით;
- XAML-ის ელემენტები შეიძლება ერთმანეთში ჩალაგდეს. ელემენტების ჩალაგების ფორმატი ასახავს ინტერფეისის ელემენტების ჩალაგებას;
- კლასის თვისებები განისაზღვრება ატრიბუტებით ან ჩალაგებული დესკრიპტორების დახმარებით, სპეცსინტაქსით.

XAML-ენა ხასიათდება თვითაღწერადობით. XAML-დოკუმენტში ყოველი ელემენტი არის ტიპის სახელი (მაგალითად, Button, Window ან Page) მოცემული სახელსივრცის ჩარჩოებში.

ელემენტთა ატრიბუტები გამოიყენება შესაბამისი ობიექტების თვისებების (მაგალითად, Name, Height, Width და ა.შ.) და მოვლენების (Click, Load და ა.შ) მოსაცემად.

WPF-დანართის MyFirstWpfProject შექმნის დროს VisualStudio აგენერირებს შემდეგ XAML-დოკუმენტს.

```
<Window x:Class="MyFirstWpfProject.MainWindow"
     xmlns="http://schemas.microsoft.com/winfx/
                                            2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     Title="MainWindow" Height="350" Width="525">
   <Grid>
   . . .
   </Grid>
```

</Window>

WPF-დანართის XAML-დოკუმენტი MyFirstWpfProject იწყება დესკრიპტორით < Window...>.

XAML-დოკუმენტის ყველა დესკრიპტორი იწყება „**<**“ - სიმბოლოთი და მთავრდება „**>**“ - სიმბოლოთი. ნებისმიერი XAML-დოკუმენტი შედგება XAML-ელემენტებისაგან. ყოველი XAML-დოკუმენტი (XAML-ელემენტი) იწყება გახსნის დესკრიპტორით (მაგალითად, < Window >), რომელსაც მოჰყვება დოკუმენტის შიგთავსი (მაგალითად, ტექსტური სტრიქონი ან სხვა XAML-ელემენტი). გახსნის დესკრიპტორში შეიძლება იყოს მოთავსებული ატრიბუტების აღწერა (მაგალითად, Class, xmlns, Title, Height, Width და სხვ.). XAML-დოკუმენტი (XAML-ელემენტი) უნდა დასრულდეს დახურვის დესკრიპტორით (მაგალითად, „/>“ ან „</Window>“).

XAML-დოკუმენტის ტექსტი უნდა შეიცავდეს ერთ ფესვურ ელემენტს - ჩალაგების უმაღლესი დონის ელემენტი. WPF-დანართის MyFirstWpfProject XAML-დოკუმენტში ასეთი ელემენტია <Window>. ფესვურ ელემენტში შეიძლება დაემატოს XAML-ის სხვა ელემენტებიც. მაგალითში ასეთი ელემენტია <Grid>.

WPF-დანართის XAML-დოკუმენტის კომპილაციის პროცესში სინტაქსურ ანალიზატორს გადაჰყავს XAML ფაილები აპლიკაციის ორობითი ფორმატირების ფაილებში BAML (BAML BAML), რომლებიც შემდეგ ჩაშენდება პროექტის ნაკრებში რესურსების სახით. WPF-დანართის კლასების ასაგებად სინტაქსური ანალიზატორი გამოიყენებს სახელსივრცეს, რომელიც განსაზღვრულია XAML-დოკუმენტის ფესვურ დესკრიპტორში.

XAML-დოკუმენტში სახელსივრცე მოიცემა xmlns ატრიბუტის საშუალებით. ზემოაღწერილ დოკუმენტში გამოცხადებულია ორი საბაზო სახელსივრცე:

- xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation - ესაა WPF-ის საბაზო სახელსივრცე, რომელიც მოიცავს WPF-ის ყველა კლასს, მართვის ელემენტების ჩათვლით. ისინი გამოიყენება მომხმარებლის ინტერფეისის ასაგებად. ვინაიდან სახელსივრცე ცხადდება პრეფიქსის გარეშე, იგი ვრცელდება მთელი XAML-დოკუმენტისათვის;
- xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" - ესაა XAML-ის სახელსივრცე. იგი შეიცავს XAML უტილიტების სხვადასხვა თვისებას, რომლებიც გავლენას ახდენს იმაზე, თუ XAML-დოკუმენტი როგორ ინტერპრეტირდება. მოცემული სახელსივრცე აისახება x პრეფიქსზე. ეს პრეფიქსი შეიძლება მოთავსდეს ელემენტის სახელის წინ (მაგ., x:ელემენტის_სახელი).

მეორე სახელსივრცე გამოიყენება XAML-ის სპეციფიური ლექსემების („საკვანძო სიტყვები“) ჩასასმელად. 14.2 ცხრილში მოცემულია შედარებით ხშირად გამოყენებადი ასეთი სიტყვები.

XAML-ის საკვანძო სიტყვები		ცხრ.14.2
N	საკვანძო სიტყვა	დანიშნულება
1	x:Array	წარმოადგენს .NET-ის მასივის ტიპს XAML-ზე
2	x:Class	XAML-ფაილის კლასის სახელი
3	x:ClassModifier	უზრუნველყოფს კლასის ტიპის ხილვადობის (internal ან public) განსაზღვრას, რომელიც Class საკვანძო სიტყვითაა აღნიშნული
4	x:Code	პროგრამული კოდი შეიძლება უშუალოდ ჩაიდოს XAML-კოდში
5	x:FieldModifier	უზრუნველყოფს ტიპის წევრის ხილვადობის (internal, public, private ან protected) განსაზღვრას ფესვის ნებისმიერი სახელმინიჭებული ელემენტისთვის (საკვანძო სიტყვით Name)
6	x:Key	უზრუნველყოფს გასაღების მნიშვნელობის დაყენებას XAML ელემენტისათვის, რომელიც უნდა მოთავსდეს ლექსიკონის ელემენტში
7	x:Name	უზრუნველყოფს C#-ით გენერირებული სახელის მითითებას მოცემული XAML ელემენტისათვის
8	x:Null	წარმოადგენს null-მიმთითებელს
9	x:Shared	შეიძლება მხოლოდ იმ რესურსებისათვის, რომლებიც ერთხელ იძლევა ეგზემპლარს. ჩვეულებრივად, ყოველი წვდომისას იწარმოება რესურსის ახალი ეგზემპლარი
10	x:Static	უზრუნველყოფს ტიპის სტატიკურ წევრზე მიმართვას
11	x:Subclass	ეს კონსტრუქცია ქმნის წარმოებულ კლასს და გამოდგება დაპროგრამების მხოლოდ იმ ენებისათვის, რომელთაც არ აქვს დანაწევრებული (partielle) კლასების მხარდაჭერა
12	x:Type	XAML-ეკვივალენტია C#-ია typeof ოპერაციის (იძახებს System.Type მითითებული სახელის საფუძველზე)
13	x:TypeArgument	უზრუნველყოფს ელემენტის დაყენებას, როგორც განზოგადებული ტიპისას განსაზღვრული პარამეტრებით
14	x:Uid	x:Name-ს პარალელურად შეუძლია მართვის ელემენტს ამ ატრიბუტით მიიღოს ცალსახა სახელი, რომელიც შემდგომი თარგმანისათვის იქნება გამოყენებული
15	x:XData	ქმნის „მონაცემთა კუნძულს“ XAML-ის შიგნით, შეუძლია მარტივი XML-მონაცემთა კონსტრუქციით წარმოება

WPF-დანართებში საბაზო სახელსივრცეთა გარდა იყენებენ ასევე სპეციალურს, რომლებიც არააუცილებელია:

- http://schemas.openxmlformats.org/markup-compatibility/2006 - XAML-ის სახელ-სივრცეა, დაკავშირებული ფორმატირების თავსებადობის პრობლემასთან სამუშაო გარემოსთან. ეს სახელსივრცე გამოიყენება XAML-ის სინტაქსური ანალიზატორის ინფორმირებისათვის იმის შესახებ, თუ რომელი ინფორმაციაა დასამუშავებელი და რომელი საიგნორირო;
- http://schemas.microsoft.com/expression/blend/2008 - XAML-ის სახელსივრცეა, რომელსაც აქვს მხარდაჭერა Expression Blend და Visual Studio პროგრამებიდან. გამოიყენება გვერდის გრაფიკული პანელის ზომების დასაყენებლად.

Window ობიექტის ატრიბუტებში შეიძლება დაემატოს შემდეგი XAML-აღწერები:

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="600"
```

მოცემული XAML-აღწერა აცხადებს არასავალდებულო სახელსივრცეებს პრეფიქსებით mc და d. თვისებები DesignHeight და DesignWidth იმყოფება სახელსივრცეში, რომელსაც აქვს პრეფიქსი d. ეს თვისებები განსაზღვრავს, რომ დანართის პროექტის დამუშავებისას Visual Studio დიზაინერში ფანჯარას უნდა ჰქონდეს ზომები 300x600. თვისება Ignorable მდებარეობს სახელსივრცეში, რომელიც აღნიშნულია პრეფიქსით mc და ის სინტაქსურ ანალიზატორს აინფორმირებს, რომ მან იგნორირება გაუკეთოს XAML-დოკუმენტის ნაწილს, რომელიც აღნიშნულია d პრეფიქსით.

WPF-დანართის XAML-დოკუმენტში ხშირად საჭიროა განხორციელდეს წვდომა პროექტის სხვა რომელიმე სახელსივრცესთან. ამ დროს აუცილებელია ახალი პრეფიქსის განსაზღვრა და მიეცეს სახელსივრცე. თუ პროექტში არის სახელსივრცე MyFirstWpfProject.Commands, მაშინ მის მიერთებას WPF -დანართის XAML-დოკუმენტთან ექნება შემდეგი სახე (command - გამოიყენება პრეფიქსის სახით).

```
xmlns:command="clr-namespace: MyFirstWpfProject.Commands"
```

პრეფიქსი (command) გამოიყენება მიმართვისთვის სახელსივრცეზე XAML-დოკუმენტში. clr-namespace ლექსემს ენიჭება სახელსივრცის დასახელება .NET ნაკრებში.

XAML-დოკუმენტში კლასის აღსაწერად გამოიყენება ატრიბუტი Class. XAML-დოკუმენტის სტრიქონი

```
<Window x:Class="MyFirstWpfProject.MainWindow" ...>
```

ითვალისწინებს MyFirstWpfProject.MainWindow კლასის შექმნას Window კლასის ბაზაზე. Class ატრიბუტის x პრეფიქსი განსაზღვრავს იმას, რომ ეს ატრიბუტი თავსდება XAML-ის სახელსივრცეში.

MainWindow კლასი გენერირდება ავტომატურად კომპილაციის დროს. კლასის ნაწილისთვის ავტომატურად გენერირდება კოდი (ნაწილობრივი (partial) კლასი):

```
namespace MyFirstWpfProject
{
  // <summary>
  // ურთიერთქმედების ლოგიკა MainWindow.xaml - ისთვის
  // </summary>
  public partial class MainWindow : Window
   {
    public MainWindow()
     {
       InitializeComponent();
      }
   }
}
```

როდესაც სრულდება დანართის კომპილაცია, XAML-ფაილი, რომელიც განსაზღვრავს მომხმარებლის ინტერფეისს (MainWindow.xaml), ტრანსლირდება CLR ტიპის გამოცხადებაში, რომელიც ერთიანდება დანართის ლოგიკასთან გამოყოფილი კოდის კლასის ფაილიდან (MainWindow.xaml.cs).

InitializeComponent() მეთოდი გენერირდება დანართის კომპილაციის დროს და საწყის კოდში არ თავსდება.

XAML-დოკუმენტში აღწერილი მართვის ელემენტების პროგრამულად სამართავად, აუცილებელია მართვის ელემენტს მიეცეს XAML ატრიბუტი Name. მაგალითად, Grid ელემენტისათვის ჩაიწერება ასე:

```
<Grid Name="grid">

</Grid>
```

მარტივი თვისებები XAML-დოკუმენტში მოიცემა შემდეგი სინტაქსის შესაბამისად:

თვისების_სახელი = "მნიშვნელობა"

მაგალითად, **Name = "grid1"**

თვისების მისაცემად, რომელიც არის სრულფასოვანი ობიექტი, გამოიყენება *რთული თვისებები* „თვისება-ელემენტი" სინტაქსის შესაბამისად:

მშობელი.თვისების_სახელი

მაგალითად, StackPanel კონტეინერისთვის აუცილებელია მიეცეს გრადიენტული ფუნჯი პანელის შესავსებად, რაც განისაზღვრება Background ატრიბუტით. იგი რეალიზდება დესკრიპტორებით:

<StackPanel.Background> . . . </StackPanel.Background>.

თვისების მნიშვნელობის მისაცემად გამოყოფილი კლასიდან გამოიყენება ფორმატირების გაფართოება, რომელიც უზრუნველყოფს XAML გრამატიკის გაფართოებას ახალი ფუნქციონალობით. ფორმატირების გაფართოება შეიძლება გამოყენებულ იქნას ჩალაგებულ დესკრიპტორებში ან XAML-ატრიბუტებში. როდესაც იყენებენ ატრიბუტებს, მაშინ აუცილებელია ფიგურული ფრჩხილების {...} გამოყენება.

ფორმატირების გაფართოებები იყენებს შემდეგ სინტაქსს:

{ფორმატირების_გაფართოების_კლასი არგუმენტი}

ფორმატირების გაფართოებები რეალიზდება კლასებით, რომლებიც შვილობილია System.Windows.Markup.MarkupExtention კლასის. MarkupExtention საბაზო კლასს აქვს ProvideValue() მეთოდი, რომელიც იძლევა ატრიბუტისათვის საჭირო მნიშვნელობას. მაგალითად, იმისათვის, რომ Button-ობიექტის Foreground ატრიბუტს მიეცეს სტატიკური თვისება, რომელიც სხვა კლასშია განსაზღვრული, აუცილებელია შემდეგი XAML-აღწერის შექმნა:

```
<Button Foreground="{x:Static SystemColors.ActiveCaptionBrush}"
                                          />
```

კომპილაციის დროს სინტაქსური ანალიზატორი შექმნის Static Extention კლასის ეგზემპლარს, შემდეგ გამოიძახებს ProvideValue() მეთოდს, რომელიც ამოიღებს საჭირო მნიშვნელობას და დააყენებს მას Foreground თვისებისათვის.

ფორმატირების გაფართოებები შეიძლება გამოყენებულ იქნას როგორც ჩალაგებული თვისებები.

მიერთებული თვისებები აღწერს თვისებებს, რომელთა გამოყენება შეიძლება რამდენიმე მართვის ელემენტთან, ოღონდ რომლებიც განსაზღვრულია სხვა კლასში. WPF-დანართებში მიერთებული თვისებები ხშირად გამოიყენება ინტერფეისის ელემენტების დაკომპლექტების სამართავად. მიერთებული თვისებების სინტაქსი შემდეგია:

განმსაზღვრელი_ტიპი.თვისების_სახელი

მაგალითად, თუ საჭიროა ღილაკის მოთავსება ბადის 0-ოვან სტრიქონში, მაშინ აუცილებელია შემდეგი XAML აღწერის შექმნა:

```
<Button ... Grid.Row="0" >
....
</Button>
```

აქ მიერთებული თვისებაა Grid.Row, ანუ Grid-ელემენტის Row-თვისება, რომელიც არაა Button ობიექტის თვისება. თვისება Row მიუერთდება Button ობიექტის თვისებებს, ვინაიდან ეს ობიექტი განთავსებულია Grid კონტეინერში.

ობიექტის ატრიბუტები შეიძლება გამოყენებულ იქნას მოვლენათა დამმუშავებლების მისაერთებლად, შემდეგი სინტაქსის გამოყენებით:

მოვლენის_სახელი = "მოვლენის_დამმუშავებლის_მეთოდის_სახელი"

მაგ., ღილაკის Click მოვლენისთვის (მისი დაჭერისას), შეიძლება დაყენდეს მოვლენის დამმუშავებელი Exit_Click.

```
<Button Name="Exit" Content="გამოსვლა" Click="Exit_Click" />
```

XAML-აღწერაში Exit_Click დამმუშავებლის განსაზღვრისას, აუცილებელია კლასის კოდში გვქონდეს მეთოდი კორექტული სიგნატურით. ქვემოთ მოყვანილია კოდი, რომელიც გენერირდება ავტომატურად მოვლენის დამმუშავებლის აღწერის შექმნისას XAML-დოკუმენტში.

```
private void Exit_Click(object sender,RoutedEventArgs e)
        {
         . . .
        }
```

14.13. WPF აპლიკაციის შექმნა

კორპორაციული აპლიკაცია (დანართი) პროგრამაა, რომელიც რეალიზაციას უკეთებს განსაზღვრულ ბიზნესამოცანას (ბიზნესფუნქციას). დანართი უნდა მუშაობდეს მონაცემებთან, რომლებიც ინახება საინფორმაციო სისტემის მონაცემთა ბაზაში.

დანართის არქიტექტურა *მოიცავს წარმოდგენის შრეს, ბიზნესლოგიკის შრეს* და *მონაცემთა შრეს.* დანართის თითოეული შრის ფუნქციონალობა ბევრადაა დამოკიდებული საინფორმაციო სისტემის საგნობრივ სფეროზე, თუმცა არსებობს აგრეთვე ზოგადი, ფუძემდებლური ფუნქციები, რომლებიც ახასიათებს პრაქტიკულად ნებისმიერ კორპორაციულ დანართს.

ამგვარად, აპლიკაციაში უნდა დამუშავდეს წარმოდგენის შრე, რომელიც უზრუნველყოფს მომხმარებლის ინტერფეისს სისტემასთან. ინტერფეისი შეიძლება შეიქმნას Windows-ფანჯრებით და WPF გვერდებით, რომლებიც შეივსება მართვის სხვადსხვა ვიზუალური ელემენტით [5].

მართვის ელემენტები უნდა უზრუნველყოფდეს სისტემის ფუნქციურობის ვიზუალურ წარმოდგენას მომხმარებლისათვის, აწარმოებდეს შესატანი მონაცემების ვერიფიკაციას და ურთიერთქმედებდეს ბიზნესკლასებთან.

ბიზნესლოგიკის შრე უნდა უზრუნველყოფდეს დანართის ძირითად ფუნქციონალობას: დააფორმიროს ბიზნესკლასები, რეალიზება გაუკეთოს მონაცემთა დამუშავების ალგორითმებს, უზრუნველყოს მონაცემებთან მიერთება და მათი ქეშირება. ამ შრის რეალიზაცია შეიძლება მოხდეს კლასების საფუძველზე, რომლებიც

ბიზნესლოგიკას არეალიზებს ინტერფეისული ელემენტების კლასთა მეთოდებით ან მონაცემთა მოდელის კლასთა მეთოდებით.

მონაცემთა შრე უნდა უზრუნველყოფდეს დანართის ურთიერთქმედებას ბაზის მონაცემებთან. კორპორაციულ აპლიკაციებში ამისათვის ყველაზე მიზანშეწონილია გამოყენებულ იქნას პლატფორმა ADO.NET Entity Framework და მოდელი EDM (Entity Data Model). EDM მოდელი აღწერს მონაცემთა სტრუქტურას ფიზიკური შენახვის ფორმისგან დამოუკიდებლად.

კორპორაციული დანართების დაპროექტების საკითხების შესასწავლად განვიხილოთ ძირითადი მიდგომები ინფორმაციული სისტემის ცალკეული ფუნქციების ასაგებად, რომელიც უზრუნველყოფს კომპანიის თანამშრომელთა მონაცემების დამუშავებას.

მაგალითისთვის აქ გამოვიყენებთ მონაცემთა ბაზას TitlePresonal, ცხრილებისა და ველების მცირე რაოდენობით, ხოლო აპლიკაციის ფუნქციურობა ითვალისწინებს კომპანიის თანამშრომელთა მონაცემების შეტანას, კორექტირებას და წაშლას. ასაგები დანართი უნდა უზრუნველყოფდეს შემდეგი მონაცემების შენახვას და გადამუშავებას: *გვარი, სახელი, სქესი, დაბადების_თარიღი, თანამდებობა, ტელეფონი, ელ_ფოსტა.*

აპლიკაციის ფუნქციებია:

- თანამშრომელთა მონაცემების დათვალიერება;
- ახალი თანამშრომლის მონაცემთა შეტანა;
- თანამშრომლის მონაცემთა რედაქტირება;
- თანამშრომლის მონაცემების წაშლა;
- მონაცემთა მოძებნა თანამშრომლის შესახებ.

შევქმნათ .NET Framework 4 გარემოში ახალი დანართის WPF-პროექტი სახელით WpfApplProject.

14.14. WPF აპლიკაცია მონაცემთა ბაზებით

განიხილება „არსთა-დამოკიდებულების“ მოდელის ძირითადი დებულებები, მისი საბაზო კომპონენტები: არსები (ობიექტები), ასოციაციები (კავშირები) და თვისებები (ატრიბუტები). სასწავლო მაგალითზე ნაჩვენებია მოდელის აგების პროცესი არსებული ბაზის გამოყენებით. აგებული PageEmployee დანართის გვერდისა და მონაცემთა მოდელისათვის ხორციელდება მონაცემთა მიბმა ინტერფეისის ელემენტებთან: ტექსტბოქსის, კომბობოქსის, ლისტბოქსის და თარიღის. განიხილება დანართის ურთიერთქმედების ოპერაციების დაპროექტების საკითხები მონაცემთა ბაზასთან: მონაცეთა რედაქტირება, დამატება და წაშლა. აღიწერება მონაცემთა შემოწმების შესაძლებლობა მათი შეტანისას, მომხმარებლელთა შემოწმების წესების გამოყენებით.

Entity Data Model (EDM) - არსთა მონაცემთა მოდელი („არსთა-კავშირების“ მოდელი). **EDM** მოდელი წარმოადგენს ძირითად ცნებათა ერთობლიობას, რომელიც აღწერს მონაცემთა სტრუქტურას მისი კომპიუტერის მეხსიერებაში ფიზიკურად შენახვის ფორმისაგან დამოუკიდებლად. EDM მოდელში აღწერილ მონაცემებს შეიძლება ჰქონდეს განსხვავებული სტრუქტურები: რელაციური, ტექსტური ფაილების, XML-ის, ელექტრონული ცხრილების და რეპორტების. EDM მოდელი აღწერს მონაცემთა სტრუქტურას არსებისა და კავშირების საფუძველზე, რომლებიც დამოუკიდებელია შენახვის სქემებისგან. ასეთი მიდგომის საფუძველზე მონაცემთა ფიზიკური დამახსოვრება განცალკევებულია დანართისაგან და არ მოქმედებს მის დამუშავებაზე. ამის უზრუნველყოფა ხდება იმის გამო, რომ არსები და კავშირები აღწერს მონაცემთა სტრუქტურებს ისე, როგორც ეს სჭირდება დანართს. EDM მოდელი კონცეპტუალური მოდელია, რომელიც აღწერს მონაცემთა სტრუქტურებს არსთა კავშირების სახით.

EDM მოდელი იყენებს სამ ძირითად ცნებას მონაცემთა სტრუქტურის აღსაწერად:

- არსის ტიპი;
- ასოციაციის ტიპი;
- თვისება.

არსის ტიპი გამოიყენება მონაცემთა სტრუქტურის აღსაწერად EDM მოდელის დახმარებით. კონცეპტუალურ მოდელში არსთა ტიპები აიგება თვისებებისგან და აღწერს ზედა დონის ძირითად კონცეპტუალურ ელემენტთა სტრუქტურას, როგორიცაა მაგალითად, თანამშრომლები და როლები. არსის ტიპი არის შაბლონი არსებისთვის. არსი არის განსაზღვრული ობიექტი (მაგალითად, რომელიმე თანამშრომელი და მისი როლი ბიზნეს-პროცესში). ყოველ არსს უნდა ჰქონდეს უნიკალური გასაღები არსთა ერთობლიობის შიგნით. არსთა ერთობლიობა არის განსაზღვრული ტიპის არსის ეგზემპლართა კოლექცია. არსთა ერთობლიობები (და ასოციაციათა ერთობლიობები) ლოგიკურად დაჯგუფებულია არსთა კონტეინერებში.

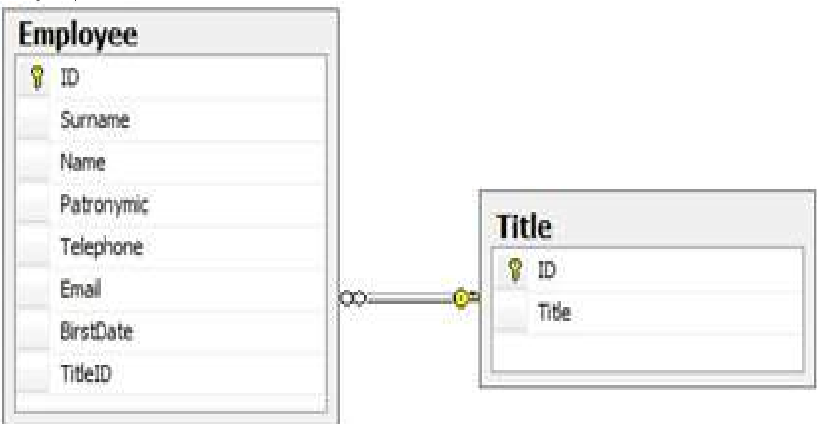
ასოციაციის ტიპი გამოიყენება კავშირთა აღწერის ასაგებად EDM მოდელში. კონცეპტუალურ მოდელში ასოციაცია ასახავს კავშირს ორი ტიპის არსებს შორის. ყოველ ასოციაციას აქვს ორი ბოლო წერტილი, რომლებიც განსაზღვრავს არსთა ტიპებს. ისინი მონაწილეობს ასოციაციაში. ყოველი ბოლო წერტილი განსაზღვრავს მის ჯერადობას, რაც მიუთითებს არსების შესაძლო რაოდენობაზე ასოციაციის ამ ბოლო წერტილში.

არსთა ტიპები შეიცავს თვისებებს, რომლებიც განსაზღვრავს მათ სტრუქტურას და მახასიათებლებს. მაგალითად, არსის ტიპს „თანამშრომელი“ შეიძლება ჰქონდეს თვისებები: ID, გვარი, სახელი, სქესი, დაბ_თარიღი, თანამდებობა, ტელეფონი, ელ_მისამართი და ა.შ.

თვისებები კონცეპტუალურ მოდელში ანალოგიურია პროგრამული აპლიკაციის კლასებისა ან მონაცემთა ბაზების ატრიბუტების. თვისებები შეიძლება შეიცავდეს

პრიმიტიულ მონაცემებს (სტრიქონი, მთელი რიცხვი, თარიღი, ლოგიკური მნიშვნელობა) ან სტრუქტურირებულ მონაცემებს (რთული ტიპი).

კონცეპტუალური მოდელი არის მონაცემთა სტრუქტურის სპეციფიკური ასახვა არსებისა და კავშირების სახით. 14.12 ნახაზზე ნაჩვენებია კონცეპტუალური მოდელის გამოსახვა სქემით, ბაზისათვის TitlePersonal – „თანამშრომელი", ორი ტიპის არსით Employee – თანამშრომელი და Title –როლი/თანამდებობა), და ასოციაციური კავშირით 1:* (ერთი:მრავალთან).



ნახ.14.12. მონაცემთა ბაზის კონცეპტუალური მოდელი

Employee ცხრილი შეიცავს თანამშრომლის მონაცემებს. მისი ატრიბუტებია:

- ID – თანამშრომლის იდენტიფიკატორი;
- Surname – გვარი;
- Name – სახელი;
- Sex –სქესი;
- BirstDate – დაბადების თარიღი;
- Telephone – ტელეფონი;
- Email – ელ_მისამართი;
- TitleID – გარე გასაღები Title ცხრილთან დასაკავშირებლად.

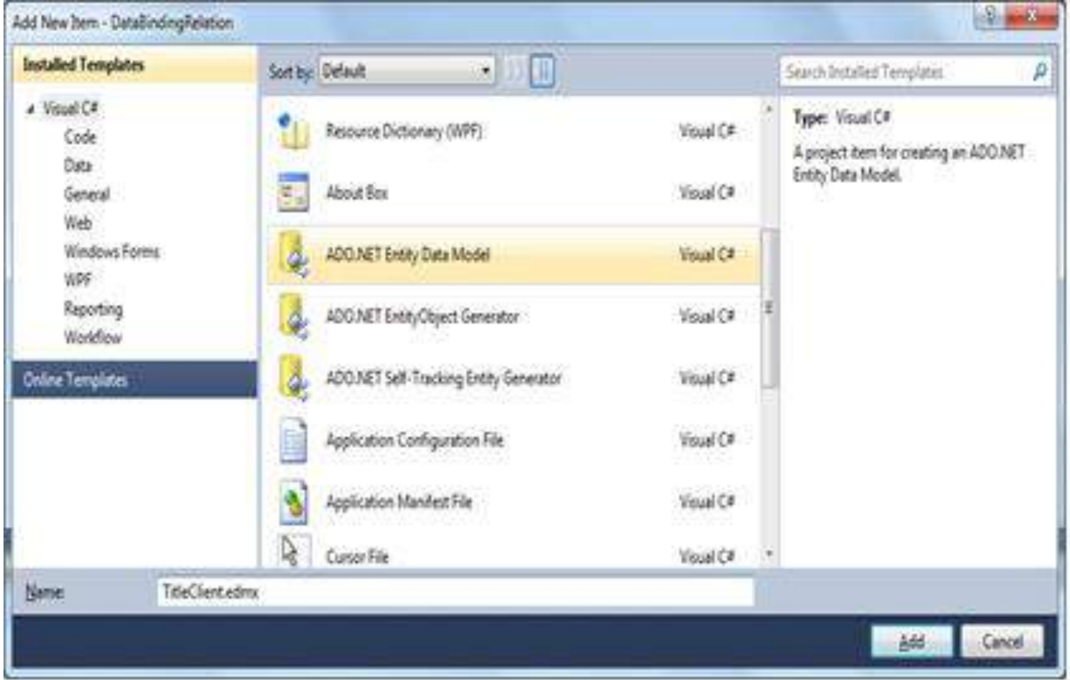
Title ცხრილი თანამდებობათა ცნობარია, რომელიც ამ ორგანიზაციას აქვს. იგი შეიცავს შემდეგ ატრიბუტებს:

- ID – თანამდებობის კოდი;
- Title – თანამდებობის დასახელება.

EDM-მოდელის შესაქმნელად პროექტში Solution Explorer-დან ჩავამატოთ ახალი ელემენტი (ნახ.14.13):

Add -> NewItem -> ADO.NET EDM

და File Name-ში მივცეთ სახელი: TitleClient.



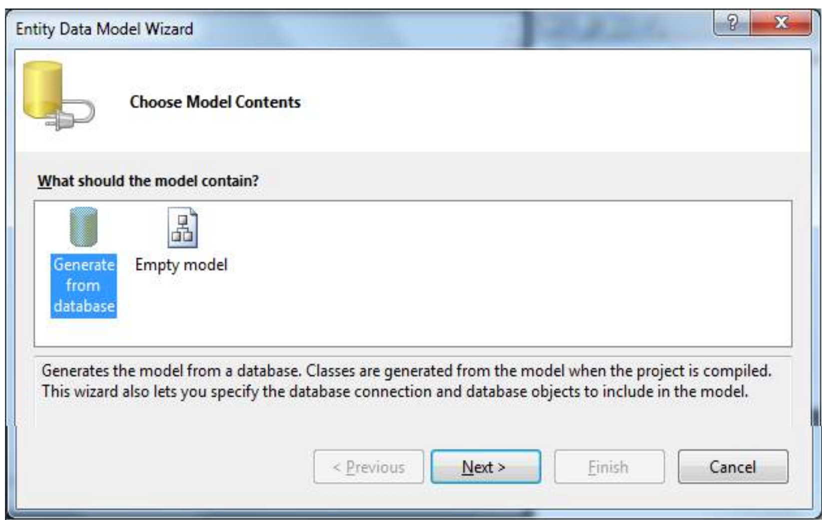
ნახ.14.13. პროექტში EDM მოდელის დამატება

EDM მოდელის შექმნისას 14.14 ნახაზზე ნაჩვენებ Wizard-ში მივუთითოთ „შექმნა მონაცემთა ბაზიდან“, რის შემდეგაც გამოვა 14.15 ნახაზზე ნაჩვენები ფანჯარა. აქ, მონაცემთა ბაზასთან მისაერთებლად, უნდა მივცეთ:

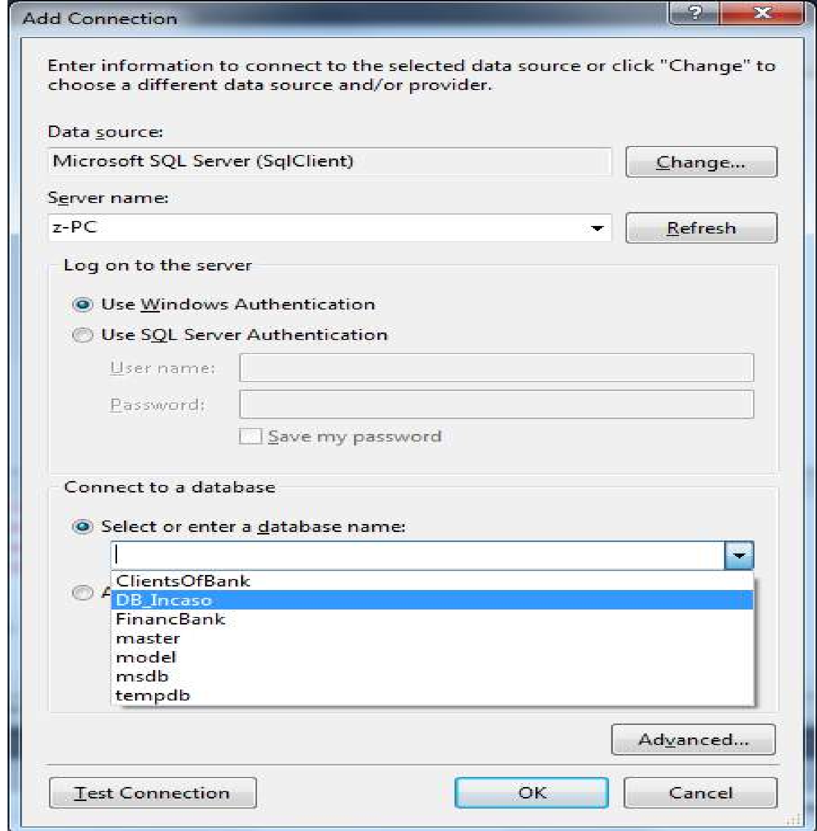
სერვერის_სახელი.მონაცემთა_ბაზის_სახელი

და მონაცემთა მოდელის სახელი - TitlePersonEntities (მიერთების პარამეტრების შენახვა Config-ში).

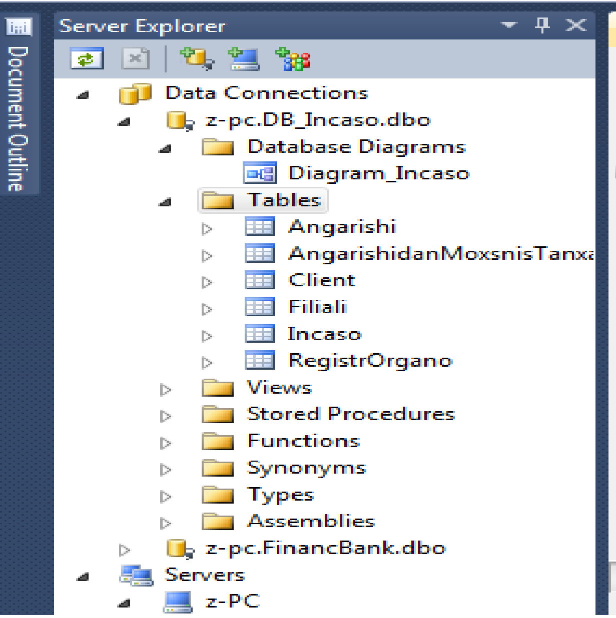
Next ღილაკით გამოიტანება 14.16 ფანჯარა, რომლის „ჩეკბოქსში“ ვირჩევთ მონაცემთა ბაზის ობიექტებს (მაგალითად, Employee და Title), მივუთითებთ ობიექტების ფორმირებას მხოლობით ან მრავლობით რიცხვში.



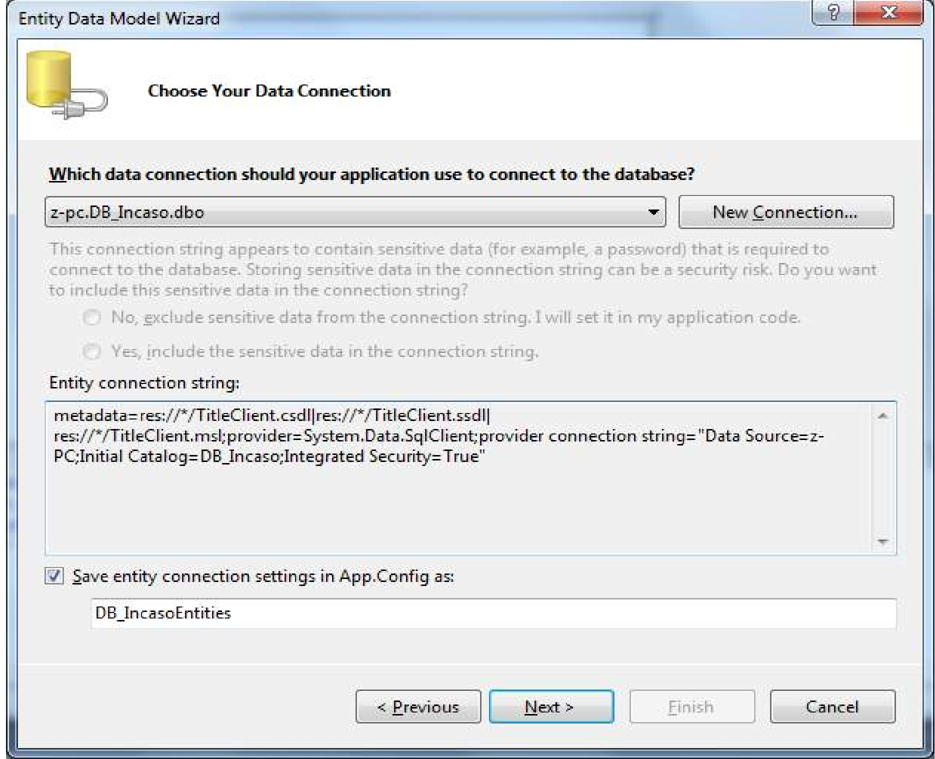
ნახ.14.14. **EDM მოდელის შექმნა მონაცემთა ბაზიდან**



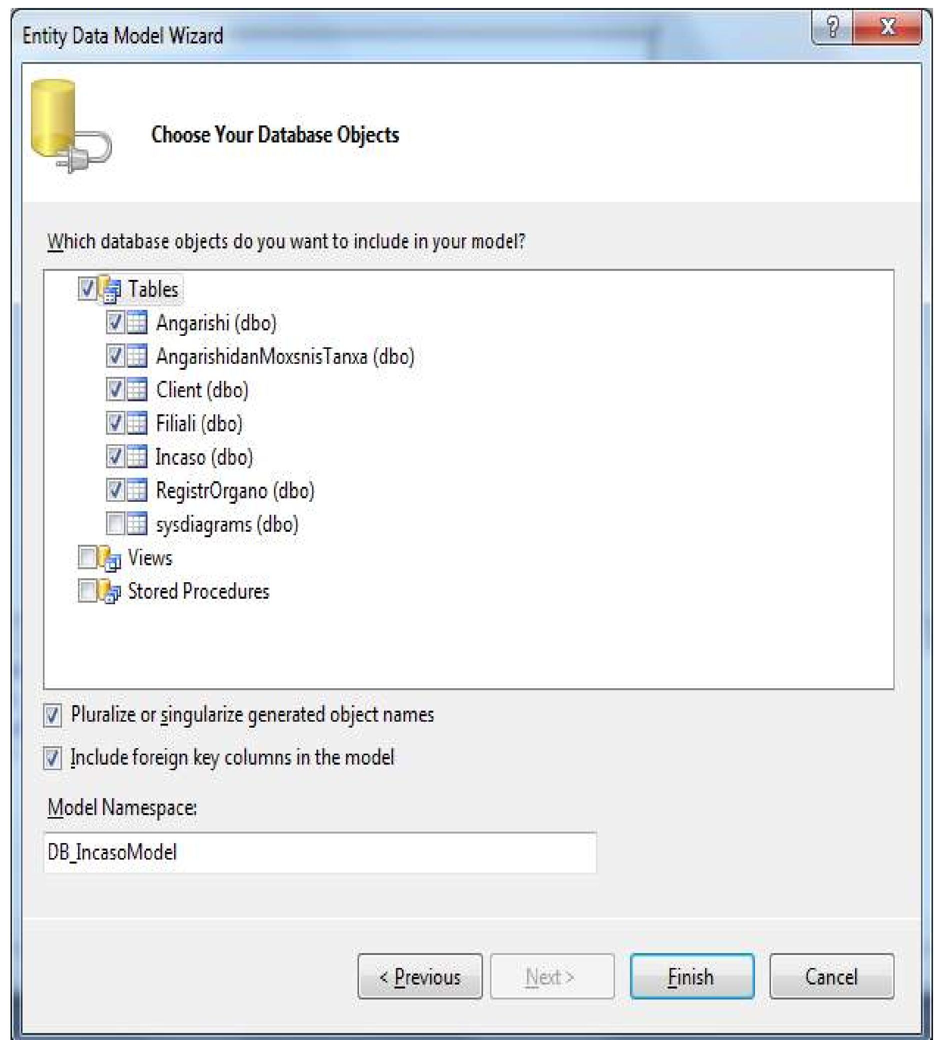
ნახ.14.15-ა. მონაცემთა ბაზასთა მიერთება: ეტაპი_1



ნახ.14.15-ბ. მონაცემთა ბაზასთა მიერთება: ეტაპი_ 2

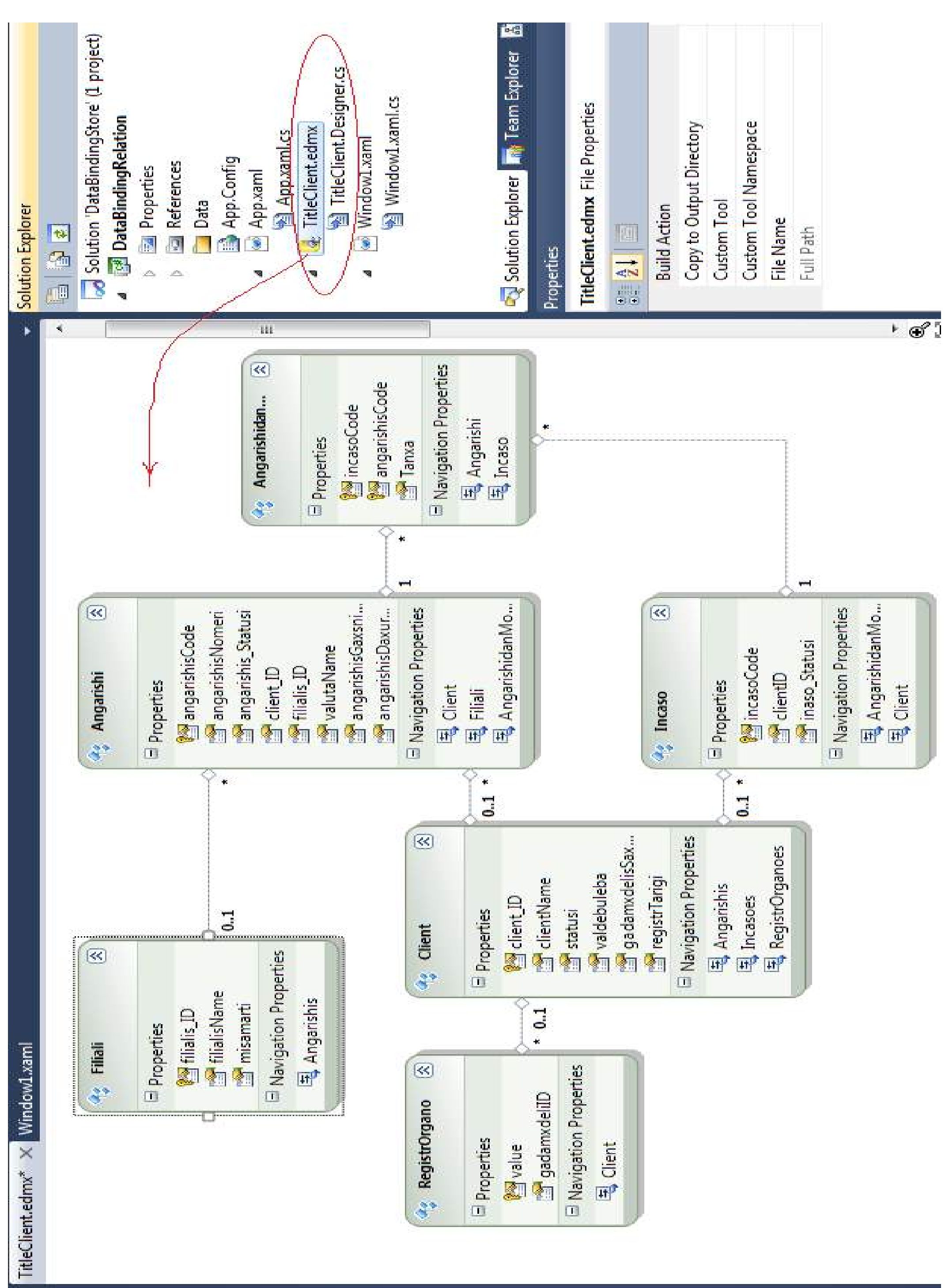


ნახ.14.15-გ. მონაცემთა ბაზასთა მიერთება: ეტაპი_3



ნახ.14.16. მონაცემთა ბაზის ცხრილების არჩევა

EDM მოდელის შექმნის შედეგად პროექტში დამატებული იქნება TitleEmployee.edmx ფაილი (ნახ.14.17). მიმართებები საჭირო ბიბლიოთეკებზე და კონფიგურაციის ფაილი.



ნახ.14.17. პროექტი EDM მოდელით

ავტომატურად გენერირებული კლასი DB_IncasoEntities, რომელიც მემკვიდრეა ObjectContext კლასის, ასახავს TitleClient-მონაცემთა ბაზის არსებს, შეიცავს თვისებებს, რომლებიც ამოდელირებს Tanamshromeli და Tanamdeboba ცხრილებს, კავშირებს მათ შორის.

მონაცემთა მოდელის შექმნისას პროექტში ავტომატურად მოხდა კონფიგურაციის App.Config ფაილის შექმნა, რომელიც შეიცავს მონაცემთა ბაზასთან მიერთების სტრიქონს.

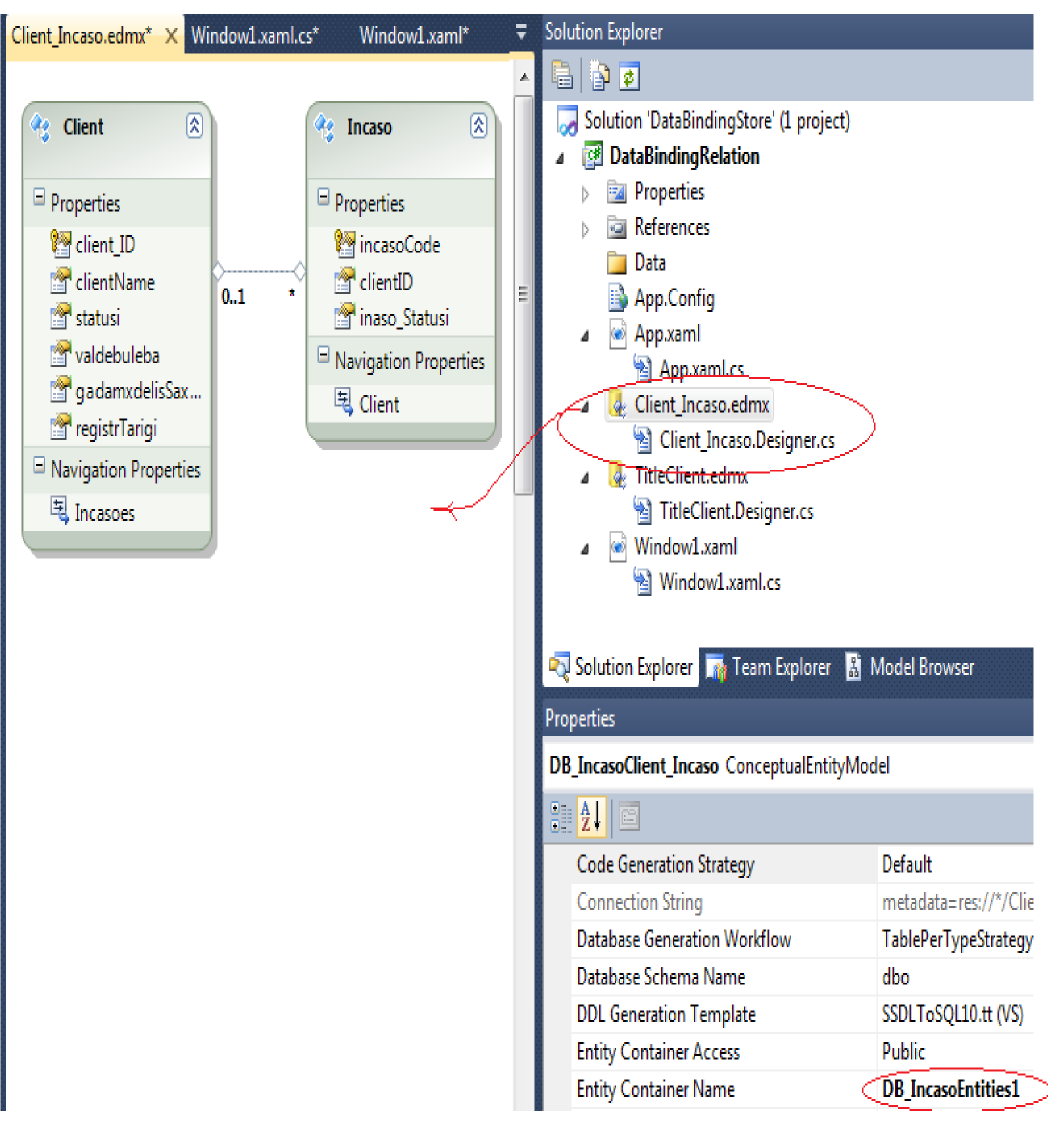
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="DB_IncasoEntities"
      connectionString="metadata=&quot;res://*/
      TitleClient.csdl|&#xD;&#xA;
      res://*/TitleClient.ssdl|res://*/
      TitleClient.msl&quot;
      provider=System.Data.SqlClient; provider connection
      string=&quot;Data Source=z-PC;Initial
      Catalog=DB_Incaso;&#xD;&#xA;
      Integrated
      Security=True;MultipleActiveResultSets=True&quot;"
      providerName="System.Data.EntityClient" />
     <add name="DB_IncasoEntities1"
      connectionString="metadata=res://*/
      Client_Incaso.csdl|res://*/Client_Incaso.ssdl|res://*/
      Client_Incaso.msl;provider=System.Data.SqlClient;
      provider connection string=&quot;
      Data Source=z-PC;Initial Catalog=DB_Incaso;Integrated
      Security=True;MultipleActiveResultSets=True&quot;"
      providerName="System.Data.EntityClient" />
   </connectionStrings>
</configuration>
```

Solution Explorer−ში Edm მოდელის C#-ის TitleClient.Designer.cs -კოდის ფრაგმენტი ასეთი სახისაა:

```
// —— ლისტინგი_EDM მოდელი ————
using System;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
```

```
using System.Data.EntityClient;
using System.ComponentModel;
using System.Xml.Serialization;
using System.Runtime.Serialization;

[assembly: EdmSchemaAttribute()]
#region EDM Relationship Metadata

[assembly: EdmRelationshipAttribute("DB_IncasoModel", "FK_Angarishi_Client",
   "Client", System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
   typeof(DataBindingRelation.Client), "Angarishi",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.Angarishi), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel", "FK_Angarishi_Filiali",
   "Filiali", System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
   typeof(DataBindingRelation.Filiali), "Angarishi",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.Angarishi), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_AngarishidanMoxsnisTanxa_Angarishi",
   "Angarishi", System.Data.Metadata.Edm.RelationshipMultiplicity.One,
   typeof(DataBindingRelation.Angarishi), "AngarishidanMoxsnisTanxa",
System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.AngarishidanMoxsnisTanxa), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_AngarishidanMoxsnisTanxa_Incaso",
   "Incaso", System.Data.Metadata.Edm.RelationshipMultiplicity.One,
   typeof(DataBindingRelation.Incaso), "AngarishidanMoxsnisTanxa",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.AngarishidanMoxsnisTanxa), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel", "FK_Incaso_Client",
   "Client", System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
   typeof(DataBindingRelation.Client), "Incaso",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.Incaso), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel", "FK_RegistrOrgano_Client",
   "Client", System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
```

```
   typeof(DataBindingRelation.Client), "RegistrOrgano",
   System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
   typeof(DataBindingRelation.RegistrOrgano), true)]
#endregion
namespace DataBindingRelation
{
  . . . }
```



ნახ.14.18. EDM მოდელი: Client_Incaso

```
// Client_Incaso.Designer.cs ---------------------
using System;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using System.Data.EntityClient;
using System.ComponentModel;
using System.Xml.Serialization;
using System.Runtime.Serialization;

[assembly: EdmSchemaAttribute()]
#region EDM Relationship Metadata

[assembly: EdmRelationshipAttribute("DB_IncasoClient_Incaso", "FK_Incaso_Client",
    "Client", System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
      typeof(DataBindingRelation.Client),
    "Incaso", System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
      typeof(DataBindingRelation.Incaso), true)]
#endregion

namespace DataBindingRelation
{
. . .
}
```

14.15. მართვის ელემენტების მიბმა მონაცემთა წყაროსთან

აპლიკაციის სამუშაოდ მონაცემთა ბაზასთან აუცილებელია Window1 კლასის კოდში გამოცხადდეს შექმნილი EDM მოდელის DataEntitiesEmployee მონაცემთა კონტექსტის სტატიკური თვისება. დანართის დაპროექტების ამ ეტაპზე ეს თვისება შეიძლება იყოს მხოლოდ ყველასთვის მისაწვდომი, ანუ - public. შემდგომში ამ თვისებას გამოიყენებს სხვა კლასები, ოღონდ PageEmployee კლასის ეგზემპლარის შექმნის გარეშე, ამიტომაც იგი უნდა იყოს სტატიკური.

```
public static TitlePresonalEntities DataEntitiesEmployee
        { get;   set; }
public static DB_IncasoEntities1 DataEntitiesClient
        { get;   set; }
```

ასევე აუცილებელია გამოცხადდეს ListEmployee კოლექცია დანართის სამუშაოდ Employee ობიექტთა კოლექციასთან.

```
public PageEmployee()
{
  InitializeComponent();
DataEntitiesEmployee = new TitlePersonalEntities();
ListEmployee = new ObservableCollection<Employee>();
}
public Window1()
{
     InitializeComponent();
     DataEntitiesClient = new DB_IncasoEntities1();
     ListClient = new ObservableCollection<Client>();
     public static DB_IncasoEntities1 DataEntitiesClient
         { get; set; }
}
```

დანართისთვის მონაცემთა ფორმირება ბაზიდან მოხდება PageEmployee გვერდის ჩატვირთვის დროს. ამისათვის XAML-დოკუმენტში Page დავამატოთ თვისება Loaded:

Loaded="Page_Loaded"

PageEmployee კლასის კოდში ჩავრთოთ დამმუშავებელი Page_Loaded.

```
private void Page_Loaded(object sender, RoutedEventArgs e)
{
  ObjectQuery<Employee> employees =
                              DataEntitiesEmployee.Employees;
  var queryEmployee = from employee in employees
                   orderby employee.Surname
                   select employee;
  foreach (Employee emp in queryEmployee)
    {
     ListEmployee.Add(emp);
    }
  DataGridEmployee.ItemsSource = ListEmployee;
}

private void Page_Loaded(object sender, RoutedEventArgs e)
{
```

```
  ObjectQuery<Client> clients =
                              DataEntitiesClient.Clients;
  var queryClient = from Client in clients
                    orderby client.clientName
                    select client;
  foreach (Client cln in queryClient)
   {
     ListClient.Add(cln);
   }
  DataGridClient.ItemsSource = ListClient;
}
```

clients ველს აქვს ObjectQuery<Client> ტიპი. ObjectQuery<T> კლასი არის მოთხოვნა, რომელიც აბრუნებს ტიპიზებულ არსთა კოლექციას ელემენტთა ნებისმიერი რაოდენობით. ავაგოთ მოთხოვნა Linq-ტექნოლოგიის საშუალებით:

```
var queryClient = from client in clients
          orderby client.clientName
          select client;
```

მოთხოვნის შედეგები მოვაწესრიგოთ „გვარებით“ (orderby client.clientName). შემდეგ ვაფორმირებთ ListClient კლიენტთა კოლექციას და მონაცემთა წყაროს DataGridClient ბადისათვის.

```
foreach (Client cln in queryClient)
   {
     ListClient.Add(cln);
   }
DataGridClient.ItemsSource = ListClient;
```

პროექტირების შედეგად დანართში ფორმირებულია კოლექცია მონაცემებით TitleClient ბაზის Client ცხრილიდან. შემდგომი ამოცანაა DataGridClient ბადის აწყობა მონაცემთა კორექტული ასახვისათვის. თავიდან მოდიფიცირება მოვახდინოთ DataGrid-ის ზოგადი აღწერისა.

```
<DataGrid Name="DataGridClient"
ItemsSource="{Binding}"
AutoGenerateColumns="False"
HorizontalAlignment="Left"
MaxWidth="1000" MaxHeight="295"
RowBackground="#FFE6D3EF"
AlternatingRowBackground="#FC96CFD4"
```

BorderBrush="#FF1F33EB"
BorderThickness="3"
IsReadOnly="True"
RowHeight="25"
Cursor="Hand">

- განვსაზღვროთ მიბმა მონაცემთა წყაროსთვის:
  ItemsSource="{Binding}"
- გავაუქმოთ სვეტების ავტომატური გენერაცია:
  AutoGenerateColumns="False"
- დავადგინოთ მიბმა გვერდის მარცხენა საზღვართან:
  HorizontalAlignment="Left"
- განვსაზღვროთ ბადის მაქსიმალური ზომები:
  MaxWidth="1000" MaxHeight="295"
- დავადგინოთ ბადის შევსების ძირითადი და ალტერნატიული ფერები:
  RowBackground="#FFE6D3EF"
  AlternatingRowBackground="#FC96CFD4"
- დავადგინოთ ბადის ჩარჩოსთვის ფერი და ხაზის სისქე:
  BorderBrush="#FF1F33EB"
  BorderThickness="3"
- დავადგინოთ ბადის სტრიქონთა სიმაღლე:
  RowHeight="25"
- შევცვალოთ კურსორის ფორმა მაუსის ისრის მიტანისას DataGridEmployee ცხრილზე: Cursor="Hand"

14.16. მარშრუტიზებადი მოვლენები

WPF-აპლიკაციები იერერქიული ბუნებისაა. მათ აქვს მართვის ელემენტები, რომლებიც თვითონ შეიცავს სხვა მართვის ელემენტებს, რომელთაც ასევე აქვს სხვა მართვის ელემენტები და ა.შ. [3,5]. მარშრუტიზებადი მოვლენა არის ისეთი მექანიზმი, რომელიც საშუალებას იძლევა, მოვლენა, რომელსაც აქვს გავლენა იერარქიის ერთ ელემენტზე, ჰქონდეს ასევე გავლენა ამავე იერარქიის სხვა ელემენტებზეც, და ამასთანავე არ იყოს საჭირო რთული კოდის დაწერა.

ამის საუკეთესო მაგალითია სიტუაცია, როდესაც მომხმარებლებს ეძლევათ საშუალება დანართში იმუშაონ მაუსის დახმარებით, რაც ძალზე ხშირია. როდესაც მომხმარებელი „კლიკავს" ღილაკს დანართში, ჩვეულებისამებრ საჭიროა, რომ დანართმა როგორღაც იმოქმედოს ამ „დაკლიკვის" მოვლენაზე. ერთ-ერთი ვარიანტი, რომელიც ცნობილია Windows Form და ASP.NET დანართებიდან, არის ამ ღილაკისათვის მოვლენის

დამმუშავებლის კოდის დაწერა (Button_Click(…) {..}), რომელშიც მითითებულია ის ქმედებები, რომლებიც უნდა შესრულდეს ამ ღილაკის „დაკლიკვის“ საპასუხოდ.

ასეთი მიდგომა ერთგვარად ზღუდავს დამმუშავებლის შესაძლებლობებს და ხშირად Windows Form-ში ქმნის საკმაოდ რთულ, გაურკვეველ კოდს. მარტივი მაგალითისათვის დავუშვათ, რომ ფორმაზე ღილაკია. ასეთ დროს რომელმა მართვის ელემენტმა უნდა იმოქმედოს დაკლიკვაზე და შექმნას მოვლენა - ფანჯარამ თუ ღილაკმა, თუ ორივემ ერთად? თუ არსებობს მოთხოვნა, რომ მოვლენა შექმნას ორივემ და ამასთანავე გარკვეული თანამიმდევრობით, მაშინ Windows Form დანართში უნდა დაიწეროს სპეციალური, საკმაოდ რთული კოდი.

WPF-ში მაუსის ღილაკის „დაკლიკვა“ მართვის ელემენტებისთვის (მათ შორის Button და Window) რეალიზდება მარშრუტიზებადი მოვლენის სახით, რაც სრულად გამორიცხავს ზემოხსენებულ პრობლემას. მარშრუტიზებადი მოვლენები გენერირდება ყველა ობიექტის მიერ განსაზღვრული მიმდევრობით იერარქიაში. ამავე დროს დამმუშავებელს ეძლევა სრული კონტროლის საშუალება იმაზე, თუ როგორ იმოქმედოს მათზე.

მაგალითად, განვიხილოთ მართვის ელემენტი Window, რომელიც შეიცავს Grid ელემენტს, რომელიც თავის მხრივ შეიცავს Rectangle ელემენტს (ნახ.14.19) [4].

ნახ.14.19. Grid მართვის ელემენტი Rectangle ელემენტით

თუ შესრულდა „დაკლიკვა“ Rectangle მართვის ელემენტზე, მაშინ ადგილი ექნება შემდეგი მიმდევრობებს:

1. მაუსის ღილაკის „დაკლიკვის“ მოვლენის გენერაცია Window-ში.
2. მაუსის ღილაკის „დაკლიკვის“ მოვლენის გენერაცია Grid-ში.
3. მაუსის ღილაკის „დაკლიკვის“ მოვლენის გენერაცია Rectangle-ში.

ამით პროცესი არ მთავრდება და შემდეგ მოხდება:

4. მაუსის ღილაკის „დაკლიკვის“ კიდევ ერთი მოვლენის გენერაცია Rectangle-ში.
5. მაუსის ღილაკის „დაკლიკვის“ კიდევ ერთი მოვლენის გენერაცია Grid-ში.
6. მაუსის ღილაკის „დაკლიკვის“ კიდევ ერთი მოვლენის გენერაცია Window-ში.

დამმუშავებელს შეუძლია მოვლენაზე რეაგირება აღნიშნული მიმდევრობის ნებისმიერ წერტილში, ანუ უბრალოდ დაამატოს მოვლენის დამუშავების შესაბამისი მეთოდი.

მას შეუძლია ასევე მოვლენის დამუშავების ჩარჩოს ნებისმიერ წერტილში ამ მიმდევრობის შეწყვეტა.

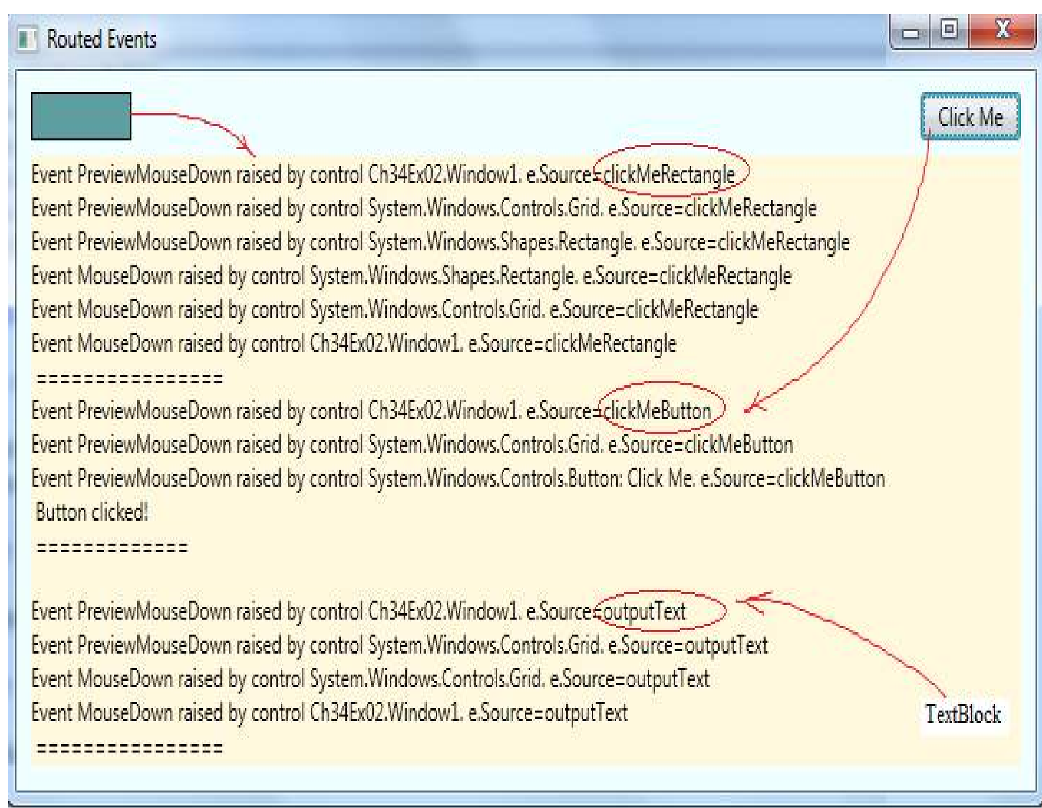
XAML-კოდს აქვს ასეთი სახე:

```
<Window x:Class="Ch34Ex02.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Routed Events" Height="400" Width="800"

        MouseDown="Generic_MouseDown"
        PreviewMouseDown="Generic_MouseDown"
        MouseUp="Window_MouseUp"   >

 <Grid Name="contentGrid" MouseDown="Generic_MouseDown"
    PreviewMouseDown="Generic_MouseDown" Background="Azure">
  <Grid.RowDefinitions>
    <RowDefinition Height="332*" />
    <RowDefinition Height="29*" />
  </Grid.RowDefinitions>
    <Rectangle Name="clickMeRectangle"   Margin="10,10,0,0" Height="23"
            HorizontalAlignment="Left" VerticalAlignment="Top"
                Width="70" Stroke="Black"
            MouseDown= "Generic_MouseDown" PreviewMouseDown=
                        "Generic_MouseDown"
            Fill="CadetBlue" />
  <Button Name="clickMeButton" Margin="0,10,10,0"  Height="23"
           HorizontalAlignment="Right" VerticalAlignment="Top"
               Width="70"
           MouseDown="Generic_MouseDown"
           PreviewMouseDown="Generic_MouseDown"
           Click="clickMeButton_Click">Click Me</Button>
      <TextBlock  Name="outputText" Margin="10,40,10,10"
                   Background="Cornsilk" />
  </Grid>
</Window>
```

3. შევცვალოთ კოდი Window1.xaml.cs ფაილში შემდეგი სახით:

```
//— ლისტინგი_ Window1.xaml.cs —————
using System; using System.Collections.Generic;
using System.Linq; using System.Text;
using System.Windows; using System.Windows.Controls;
using System.Windows.Data; using System.Windows.Documents;
using System.Windows.Input; using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation; using System.Windows.Shapes;
using System.Windows.Media.Animation;   // დაემატა
namespace Ch34Ex02
{
public partial class Window1 : Window
 {
 private void Generic_MouseDown(object sender, MouseButtonEventArgs e)
    {
      outputText.Text = string.Format("{0}Event {1} raised by control {2}. e.Source={3}\n",
         outputText.Text, e.RoutedEvent.Name, sender.ToString(),
((FrameworkElement)e.Source).Name);
     }
 private void Window_MouseUp(object sender,
                                                 MouseButtonEventArgs e)
  {
    outputText.Text = string.Format("{0} =======\n", outputText.Text);
  }
 private void clickMeButton_Click(object sender, RoutedEventArgs e)
   {
   outputText.Text = string.Format("{0} Button clicked!\n==n\n", outputText.Text);
   }
 }
}
```

4. ავამუშავოთ დანართი, შედეგი ნაჩვენებია 14.20 ნახაზზე.

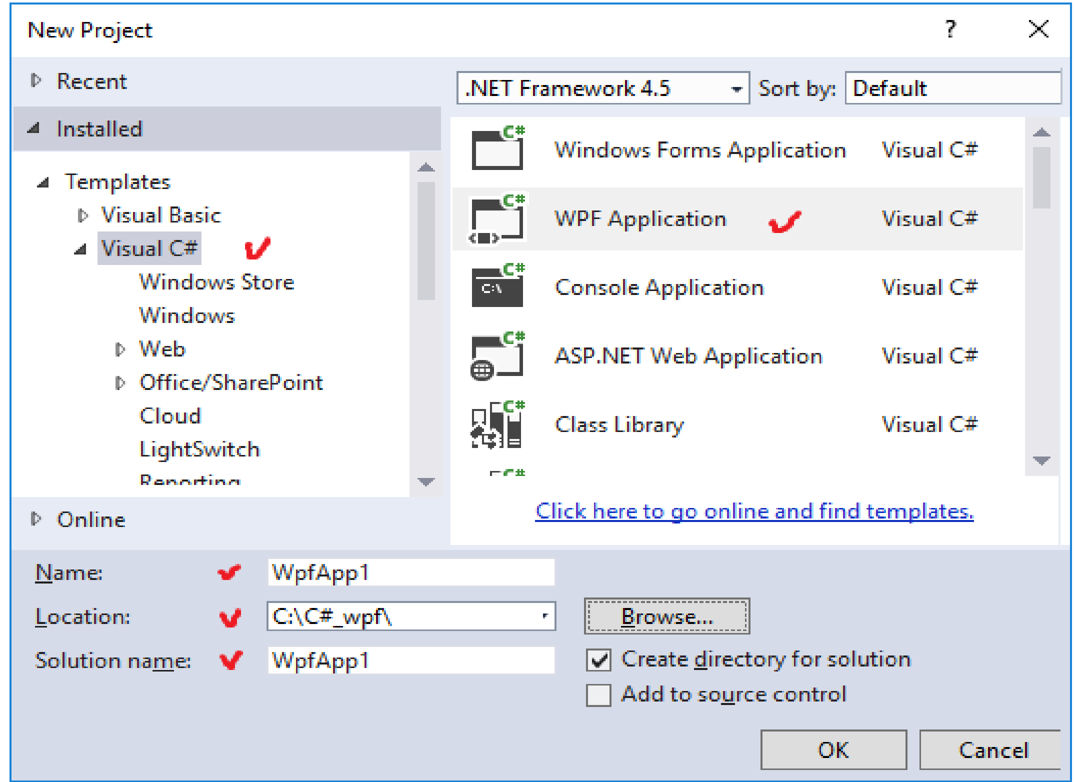


ნახ.14.20

XV თავი
WPF-ტექნოლოგიის ვიზუალური ელემენტები და მათი გამოყენება აპლიკაციების ასაგებად

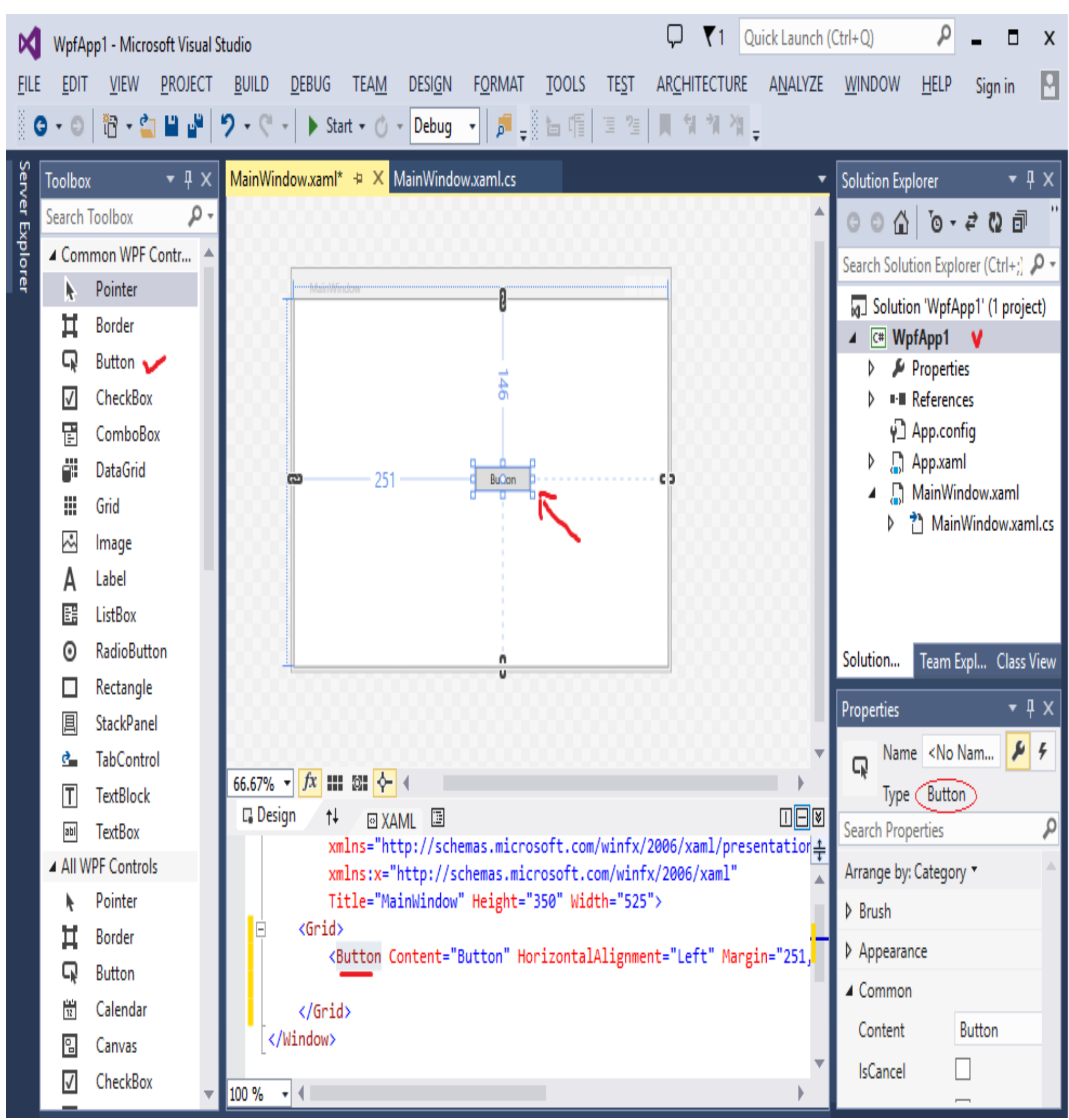
15.1. Windows Presentation Foundation ტექნოლოგის სამუშაო გარემო და ინსტრუმენტების პანელი

ახალი WPF პროექტის შექმნა იწყება 15.1 ნახაზზე მოცემული სურათით, სადაც ჩანს პროექტის სახელის (Name), შენახვის მისამართის (Location) და Solution name-ს მნიშვნელობები.



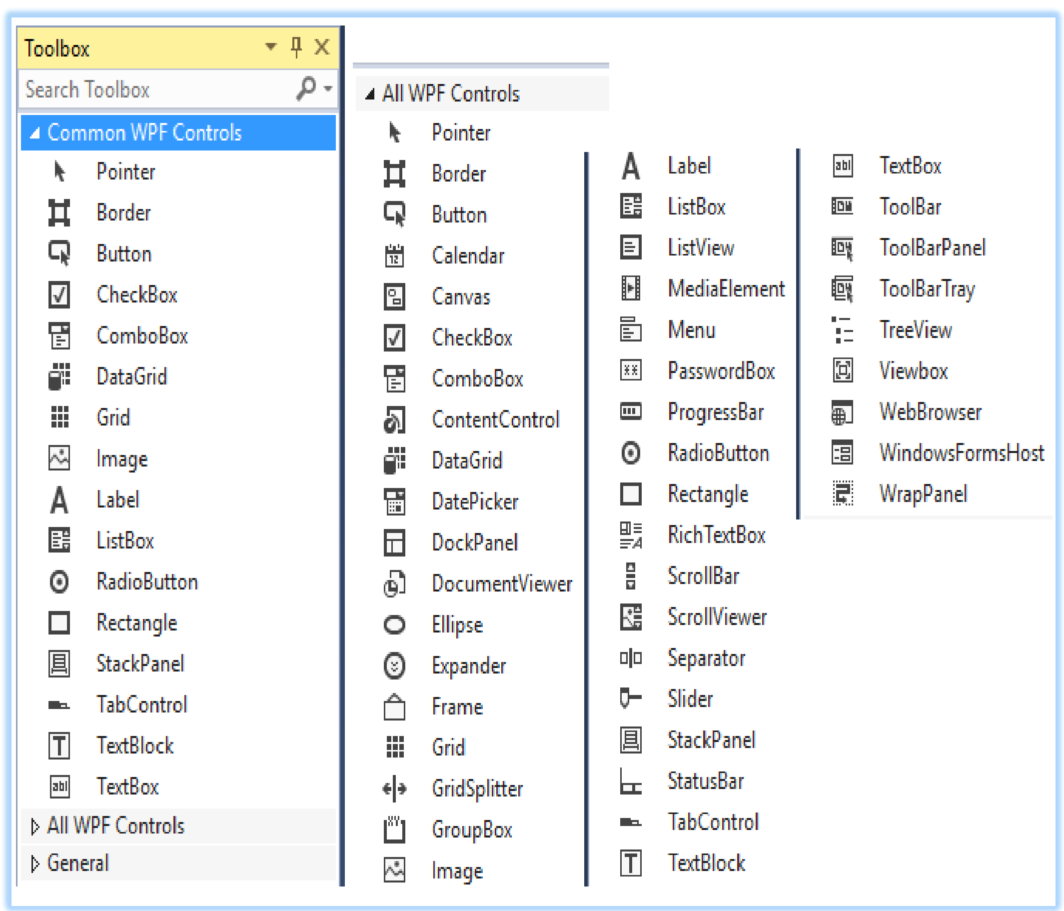
ნახ.15.1. ახალი WpfApp1 პროექტის შექმნა

მიიღება 15.2 ნახაზზე ნაჩვენები ფანჯარა. აქ ჩანს ინსტრუმენტების პანელი - მარცხნივ, Solution Explorer - მარჯვენა ზედა და Properties - მარჯვენა ქვედა ნაწილში. ცენტრის ზედა ნაწილში ხდება ფორმის აწყობა ინსტრუმენტების პანელის სასურველი ელემენტებით (ჩვენ შემთხვევაში Button1). ქვედა ნაწილში მოთავსებულია xaml ფაილის ტექსტი, რომელიც შეესაბამება ფორმის შედგენილობას.



ნახ.15.2. პროექტის სამუშაო გარემო

WPF–ის ინსტრუმენტების პანელი მოცემულია 15.3 ნახაზზე. ჩვენ გამოვიყენებთ პანელის კომპონენტებს და მათი დაპროგრამების საშუალებებს კონკრეტული ამოცანების გადაწყვეტის დროს.



ნახ.15.3. ინსტრუმენტების პანელის კომპონენტები

საინჟინრო, ეკონომიკური, ბიზნესისა და სხვა სფეროების ამოცანებში, რომლებიც მართვის საინფორმაციო სისტემების ასაგებად და გამოსაყენებლად არის გათვალისწინებული, ხშირად საჭიროებს მონაცემების შეტანის, ინფორმაციის გამოტანის, სისტემის მიერ შემოთავაზებული ვარიანტების ამორჩევისა და შესაბამისი ღილაკის (Button) ამოქმედების პროცედურების შესრულებას.

ყოველივე ეს კი მოითხოვს ასეთი მოვლენების დაპროგრამებას, ანუ რა რეაქცია ექნება სისტემას, რა მონაცემებს გამოიტანს იგი მომხმარებლის სურვილის შესაბამისად ფანჯრის ფორმაზე და ა.შ. თავიდან განვიხილოთ ღილაკის აგების ამოცანა.

15.2. ღილაკის დაპროგრამება ანიმაციის ელემენტებით

ავაგოთ მომხმარებლის ინტერფეისი, რომელზეც გამოყენებული იქნება WPF-ის ანიმაციური შესაძლებლობის ღილაკი (Button). მოვახდინოთ მისი მულტიმედიური თვისებების და პროგრამული კოდის დემონსტრირება:

1. გამოვიყენოთ წინა პარაგრაფში შექმნილი პროექტი სახელით: WpfApp1 და Solution-ის სახელით: WpfApp (ნახ.15.1).

შედეგი მოცემული იყო 15.2 ნახაზზე.

2. შევავსოთ XAML კოდი, როგორც ქვემოთაა ნაჩვენები. კომენტარებში მოცემულია პროცედურის შინაარსი.

```
<!----WpfApp1----- Window1.xaml ----- >
<!-- მთლიანად ფანჯრის აწყობა: მომხმარებლის ინტერფეისი -->
<Window x:Class="WpfApp1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="300" Height="300"
        Title="მაგალითი_1"
        x:Name="Window"
        Background="Aqua">

    <!-- აპლიკაციის ფანჯრის რესურსების შესანახი სექცია: -->
    <Window.Resources>
   <!-- იქმნება შაბლონი დასახელებით ClassButton, ღილაკის ტიპის
         ელემენტებისათვის-->
    <ControlTemplate x:Key="ClassButton"
           TargetType="{x:Type Button}">
      <!-- ფანჯრის რესურსების შესანახი სექცია ღილაკისათვის -->
     <ControlTemplate.Resources>
      <!-- სექციაში Storyboard აღიწერება ანიმაციური ეფექტი, მაგ.,
                    ღილაკის დაჭერა-->
   <Storyboard x:Key="Timeline1">
        <!-- მითითებულ დროში, მაგ., 0.3 წამი, ღილაკი ხდება
                                          გაუმჭვირვალე -->
```

```
      <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
        Storyboard.TargetName="glow"
        Storyboard.TargetProperty="(UIElement.Opacity)">
          <!-- ანიმაციის ბოლო წერტილი -->
        <SplineDoubleKeyFrame KeyTime="00:00:0.3" Value="1" />
      </DoubleAnimationUsingKeyFrames>
    </Storyboard>

           <!-- სექცია Storyboard ღილაკის ჩასაქრობად -->
    <Storyboard x:Key="Timeline2">
<!- მითითებულ დროში, მაგ., 0.3 წამში ღილაკი ხდება გამჭვირვალე-->
      <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
        Storyboard.TargetName="glow"
        Storyboard.TargetProperty="(UIElement.Opacity)">
          <!-- ანიაციის ბოლო წერტილი -->
       <SplineDoubleKeyFrame KeyTime="00:00:0.3" Value="0" />
      </DoubleAnimationUsingKeyFrames>
   </Storyboard>
</ControlTemplate.Resources>
       <!-- ღილაკის აღწერის სექცია -->
         <!-- გარე საზღვარი - თეთრი -->
 <Border BorderBrush="#FFFFFFFF" BorderThickness="1,1,1,1"
                                          CornerRadius="4,4,4,4">
         <!-- შიგა საზღვარი - შავი -->
 <Border x:Name="border" Background="#7F000000" BorderBrush="#FF000000"
BorderThickness="1,1,1,1" CornerRadius="4,4,4,4">
      <Grid>
        <Grid.RowDefinitions>
          <!-- ღილაკის ზედა ნახევარი -->
            <RowDefinition Height="0.5*" />
              <!-- ღილაკის ქვედა ნახევარი -->
                <RowDefinition Height="0.5*" />
        </Grid.RowDefinitions>

        <!-- იხატება ღილაკის განათება -->
```

```
<Border Opacity="0" HorizontalAlignment="Stretch"
  x:Name="glow" Width="Auto"
         Grid.RowSpan="2" CornerRadius="4,4,4,4">
    <Border.Background>
    <!-- მიეწოდება რადიალური გრადიენტი წანაცვლებით -->
           <RadialGradientBrush>
             <RadialGradientBrush.RelativeTransform>
                <TransformGroup>
          <ScaleTransform ScaleX="1.702" ScaleY="2.243" />
                  <SkewTransform AngleX="0" AngleY="0" />
                  <RotateTransform Angle="0" />
             <TranslateTransform X="-0.368" Y="-0.152" />
                </TransformGroup>
             </RadialGradientBrush.RelativeTransform>
             <!-- გრადიენტის ფერები ARGB ფორმატში-->
             <GradientStop Color="#B28DBDFF" Offset="0" />
             <GradientStop Color="#008DBDFF" Offset="1" />
             </RadialGradientBrush>
          </Border.Background>
        </Border>
    <!-- ათინათის დახატვა -->
    <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"
Width="Auto" Grid.RowSpan="2" />
  <Border HorizontalAlignment="Stretch" x:Name="shine"
     Width="Auto" CornerRadius="4,4,0,0">
       <Border.Background>
         <LinearGradientBrush StartPoint="0.494,0.028"
            EndPoint="0.494,0.889">
               <GradientStop Color="#99FFFFFF" Offset="0" />
               <GradientStop Color="#33FFFFFF" Offset="1" />
             </LinearGradientBrush>
       </Border.Background>
     </Border>
    </Grid>
  </Border>
 </Border>
<!--ტრიგერული მოვლენების დაყენება, მაუსის დაჭერის რეაქციაზე -->
```

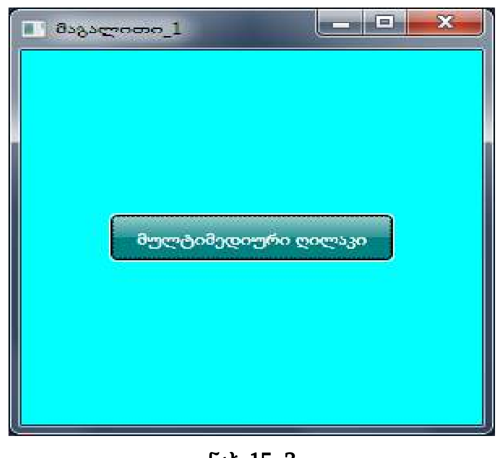
```
    <ControlTemplate.Triggers>
      <!-- მაუსის ღილაკი დაჭერილია -->
    <Trigger Property="IsPressed" Value="True">
 <Setter Property="Opacity" TargetName="shine" Value="0.4" />
  <Setter Property="Background" TargetName="border"
                                    Value="#CC000000" />
  <Setter Property="Visibility" TargetName="glow"
                                    Value="Hidden" />
   </Trigger>
     <!-- მაუსის კურსორი ობიექტზეა -->
       <Trigger Property="IsMouseOver" Value="True">
    <!-- ობიექტზე შესვლა - ანიმაციის გამოძახება Timeline1 -->
         <Trigger.EnterActions>
  <BeginStoryboard Storyboard="{StaticResource Timeline1}" />
         </Trigger.EnterActions>
  <!-- ობიექტიდან გამოსვლა - ანიმაციის გამოძახება Timeline2 -->
         <Trigger.ExitActions>
  <BeginStoryboard Storyboard="{StaticResource Timeline2}" />
         </Trigger.ExitActions>
     </Trigger>
            </ControlTemplate.Triggers>
        </ControlTemplate>
    </Window.Resources>
    <Grid>
        <!-- ფანჯრის შუაში იქმნება ღილაკის ეგზემპლარი -->
        <Button x:Name="Btn1" HorizontalAlignment="Center"
     VerticalAlignment="Center"  Width="176" Height="34"
     Content="მულტიმედიალური ღილაკი" Foreground="#FFFFFFFF"
      Template="{DynamicResource ClassButton}" />
    </Grid>
</Window>
```

3. ავამუშავოთ პროგრამა, მივიღებთ 15.3 ნახაზზე ნაჩვენებ შედეგს, სადაც შაბიამნისფერის ფონია. ღილაკზე დაჭერით იცვლება ფონი იასამნისფერით (ნახ.15.4). შემდგომ ფერების შეცვლა მონაცვლეობით ხდება მაუსის „დაკლიკვით“.

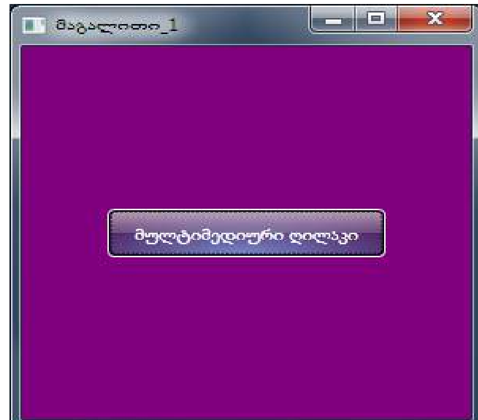
4. დავაპროგრამოთ ლოგიკა C#ენაზე:

```
using System;
using System.Collections.Generic;
```

```
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp1
{
  // Interaction logic for Window1.xaml
  public partial class Window1 : Window
  {
    System.Windows.Media.Brush color;
    bool colorFlag = true;
   public Window1()
    {
       //InitializeComponent();
     Application.LoadComponent(this, new Uri("Window1.xaml",
                      UriKind.Relative));
     Btn1.Click += new RoutedEventHandler(Btn1_Click);
     color = this.Window.Background;
    }
    void Btn1_Click(object sender, RoutedEventArgs e)
       {
         // ფინის ფერის შეცვლა
        if (colorFlag)
           this.Window.Background =
               System.Windows.Media.Brushes.Purple;
         else
           this.Window.Background = color;
       colorFlag = !colorFlag;
      }
  }
}
```



ნახ.15..3



ნახ.15.4

5. Application - ობიექტი

ნებისმიერი დანართი (აპლიკაცია) იყენებს Application კლასს, რომელიც Run() მეთოდის საშუალებით ამ აპლიკაციას ასამუშავებლად მიუერთებს ოპერაციული სისტემის მოვლენების მოდელს.

Application - ობიექტი მართავს დანართის სიცოცხლის დროს, აკვირდება ხილვად ფანჯრებს, ათავისუფლებს რესურსებს და აკონტროლებს აპლიკაციის გლობალურ მდგომარეობას. Run() მეთოდი აამუშავებს შესრულების გარემოს დისპეტჩერს, რომელიც დაიწყებს მოვლენებისა და შეტყობინებების გადაგზავნას დანართის კომპონენტებთან.

დროის მოცემულ მომენტში აქტიური შეიძლება იყოს მხოლოდ ერთი Application - ობიექტი, და ის მუშაობს მანამ, სანამ დანართი არ დასრულდება. შესრულებად Application - ობიექტზე მიმართვა შეიძლება დანართის ნებისმიერი ადგილიდან Application.Current სტატიკური თვისების საშუალებით.

WPF დანართის (და Application ობიექტის) სასიცოცხლო დრო შედგება შემდეგი ეტაპებისაგან:

1. კონსტრუირდება Application ობიექტი;
2. გამოიძახება მისი Run() მეთოდი;
3. ინიცირდება მოვლენა Application.Startup;
4. მომხმარებლის კოდი აკონსტრუირებს ერთ ან რამდენიმე Window (ან Page) ობიექტს და დანართი მუშაობს;
5. გამოიძახება მეთოდი Application.Shutdown();
6. გამოიძახება მეთოდი Application.Exit().

ჩვენი WPF-დანართის აგებისას სისტემამ თვითონ შექმნა Solution Explorer-ში ორი ფაილი Application-ობიექტთან დაკავშირებული ტიპური სახელებით: App.xaml და App.xaml.cs. მათი ნახვა შესაძლებელია. აქ არ ჩანს ცხადად არც Application და Window ობიექტების შექმნა და არც Run() მეთოდის გამოძახება (!)

WPF-პლატფორმის შემქმნელებმა გადაწყვიტეს, რომ, ვინაიდან ეს სტანდარტული ოპერაციები გამოიყენება ყველა დანართისათვის, არაა საჭირო მისი მომხმარებელზე გადაცემა და თვით კომპილატორი (სისტემა) ავტომატურად გამოიყენებს მათ (!), თვითონ შექმნის Application ობიექტს WPF-პროექტის კომპილაციის დროს, შექმნის ასევე Window (ან Page) ობიექტებს და გადასცემს მათ Application.Run() მეთოდს.

სხვა სიტყვებით რომ ვთქვათ, ჩვენ „ვერ ვხედავთ ცხადად" კომპილატორის მიერ ასეთი კოდის შესრულების ტექსტს:

```
// ---- საილუსტრაციო ტექსტი ---------
public partial class App : System.Windows.Application
{
  public App()
    {  System.Windows.Window win = new System.Windows.Window();
       win.Title = "Hello World";
       win.Show();
    }
      // Application entry point
    [System.STAThreadAttribute()]
    [System.Diagnostics.DebuggerNonUserCodeAttribute()]
    public static void Main()
    {  App app = new App();
       app.Run();
    }
}
```

ჩვენი დანართის მაგალითისთვის App.xaml.cs ფაილი ასე გადავაკეთოთ:

```
// ----- App.xaml.cs  -------------------------
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Windows;
namespace WpfApp1
{
  public partial class App : Application
    {
      public App()
        {
        // გაშვებულია Application ობიექტი
```

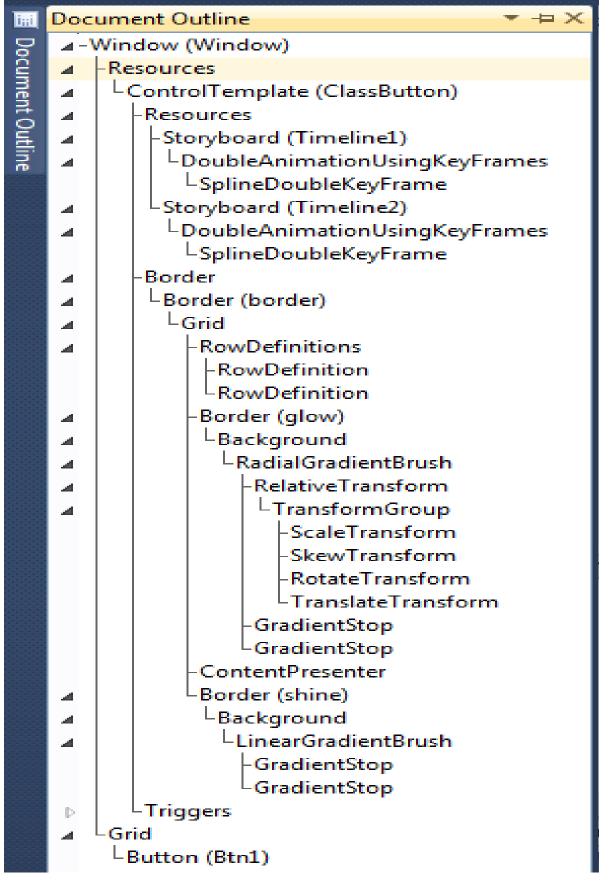
```
        this.Startup += new StartupEventHandler(App_Startup);
         // მოვლენა დაუმუშავებელი გამოსარიცხი სიტუაციისათვის
   this.DispatcherUnhandledException += App_DispatcherUnhandledException;
      }
   void App_DispatcherUnhandledException(object sender,
        System.Windows.Threading.DispatcherUnhandledExceptionEventArgs e)
      {
         e.Handled = true;// შესრულების გაგრძელება
      }
  void App_Startup(object sender, StartupEventArgs e)
      {
         // იქმნება სასტარტო ფანჯრის ობიექტი -----
         Window1 win = new Window1();
        // ფანჯრის ობიექტის აწყობა -----
         win.Title = "ფანჯრის  ახალი სათაური ";
         // ფანჯრის ნახვა ---------
         win.Show();
      }
   }
}
```

ავამუშავოთ დანართი და ვნახოთ შედეგი.

6. XAML-პროგრამის სტრუქტურის სანახავად Document Outline პანელის გამოტანა:

View/Other Windows/Document Outline

მიიღება შემდეგი იერარქიული სურათი (ნახ.15.5):



ნახ.15.5

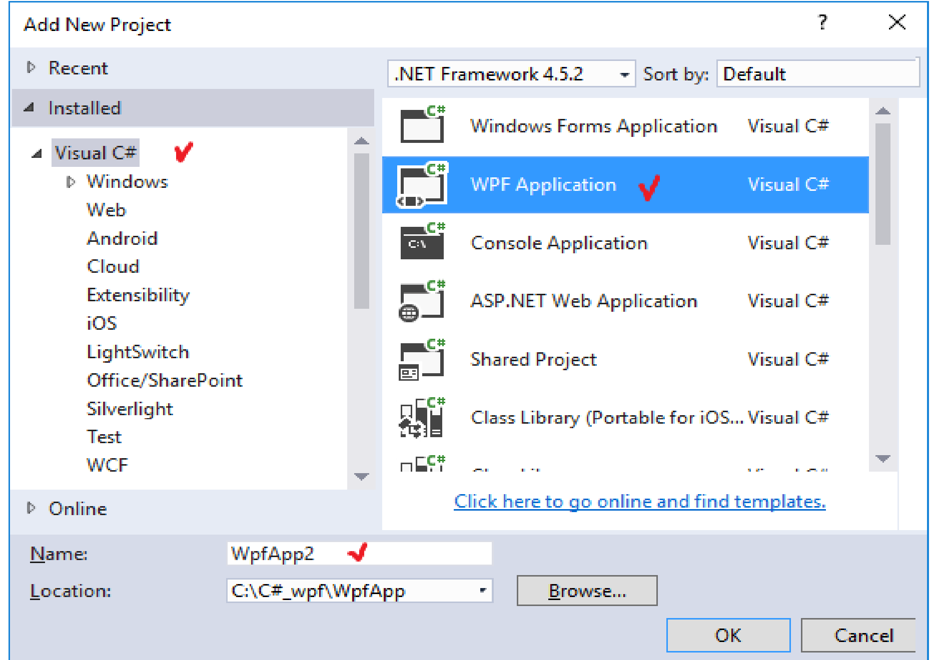
15.3. WPF-ის ფანჯრების მართვის ელემენტები

ფანჯრების შექმნა და მათი გამოყენება მოხერხებული საშუალებაა შესაბამისი ფუნქციურობისა და მონაცემების ინკაფსულაციის სარეალიზაციოდ. ფანჯარა (Window) ყველაზე მაღალი დონეა მომხმარებელთან ურთიერთობისათვის, ის არ შეიძლება იყოს სხვა ფანჯრის ნაწილი.

ფანჯარა შეიძლება იყოს მოდალური და არამოდალური. მოდალური ფანჯარა აბლოკირებს ამ დანართის სხვა ფანჯრებს მის დახურვამდე და თვითონ მუშაობს ოპერაციულ სისტემასთან.

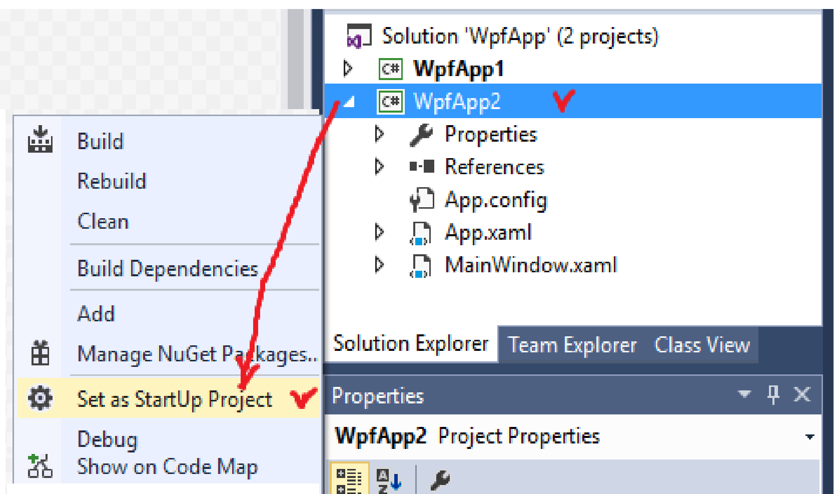
გამოვიყენოთ წინა პროექტის ფაილი. გავხსნათ იგი და:

1. დავამატოთ WpfApp -ს Solution-იდან ახალი პროექტი WpfApp2 სახელით:.



ნახ.5.16

WpfApp2 გავააქტიუროთ Set as StartUp -ით (ნახ.15.17).



ნახ.15.17. WpfApp2 გააქტიურდა (სტრიქონი გამუქდა)

2. შევავსოთ XAML კოდი ასე:

```
<Window x:Class="WpfApp2.MainWindow"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="MainWindow" Height="300" Width="300"
   SizeToContent="WidthAndHeight"
   MinWidth="300"
   MaxWidth="400"
   MinHeight="100"
   WindowStartupLocation="CenterScreen"
  >
   <StackPanel>
     <Button Margin="5" Click="LifeEvents">1) ფანჯრის
              სიცოცხლის დროის მოვლენები</Button>
   </StackPanel>
</Window>
```

3. C# კოდის ნაწილი MainWindow.xaml.cs-თვის:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApp2
{
  public partial class Window0 : Window
  {
     public Window0()
     {
```

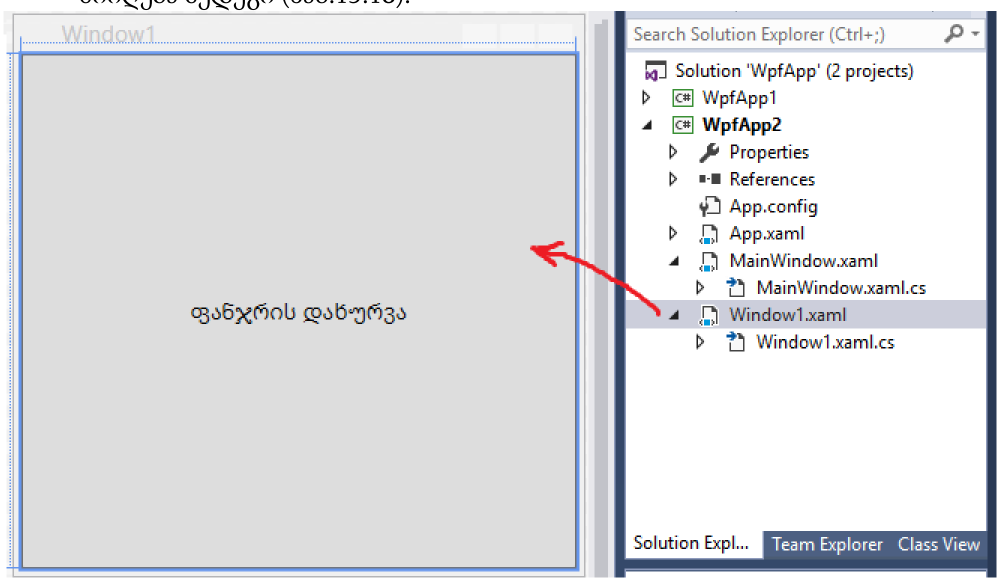
```
        InitializeComponent();
      }

   private void LifeEvents(object sender, RoutedEventArgs e)
    {
     Window1 wnd1 = new Window1();
     wnd1.ShowInTaskbar = false; // არ ჩანს ამოცანების პანელზე
     wnd1.Show();              // გამოჩნდება მოდალურ რეჟიმში
    }
  }
}
```

4. WpfApp2-ში დავამატოთ (Add) ფორმა Window1() : შეიქმნება ახალიWindow1.xaml ფაილი.

ჩავამატოთ ამ ფაილში სტრიქონი `<Grid> . . . </Grid>`: შორის:

```
<Button Click="Button_Click" Content="ფანჯრის დახურვა"> </Button>
```

მიიღება შედეგი (ნახ.15.18).



ნახ.15.18

XAML –კოდში ჩავამატოთ 8 მოვლენის სტრიქონი <Grid>-მდე, რომელთა შესაბამისი მეთოდები ჩაიწერება C# კოდში:

```
<Window x:Class="WpfApp2.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

        Title="Window1" Height="300" Width="300"

        Initialized="Window_Initialized"
        Activated="Window_Activated"
        Deactivated="Window_Deactivated"
        Loaded="Window_Loaded"
        ContentRendered="Window_ContentRendered"
        Closing="Window_Closing"
        Unloaded="Window_Unloaded"
        Closed="Window_Closed"
        >
    <Grid>
      <Button Click="Button_Click" Content="ფანჯრის
                    დახურვა"></Button>
    </Grid>
</Window>
```

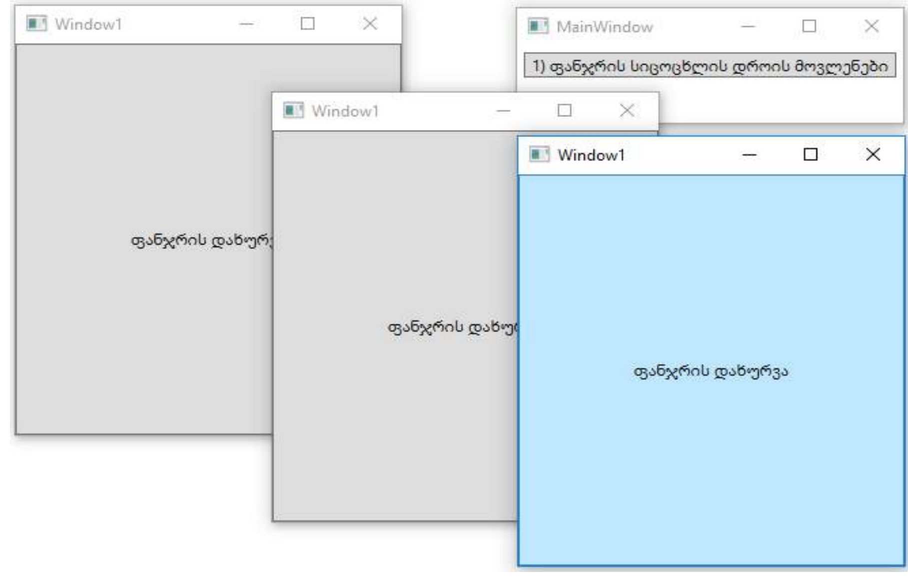
C# - კოდის ლისტინგი 8 მოვლენისათვის:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace WpfApp2
{
   public partial class Window1 : Window
    {
```

```
        public Window1()
         {
         System.Diagnostics.Debug.WriteLine("ამუშავდა ფანჯრის
                                                კონსტრუქტორი");
          InitializeComponent();
        }
   private void Window_Initialized(object sender, EventArgs e)
         {
           System.Diagnostics.Debug.WriteLine("ამოქმედდა
                                  ფანჯრის მოვლენა  Initialized");
         }
   private void Window_Activated(object sender, EventArgs e)
         {
           System.Diagnostics.Debug.WriteLine("ამოქმედდა
                                    ფანჯრის მოვლენა Activated");
         }
   private void Window_Deactivated(object sender, EventArgs e)
         {
           System.Diagnostics.Debug.WriteLine("("ამოქმედდა
                                 ფანჯრის მოვლენა  Deactivated ");
         }
   private void Window_Loaded(object sender, RoutedEventArgs e)
         {
           System.Diagnostics.Debug.WriteLine("("ამოქმედდა
                                     ფანჯრის მოვლენა  Loaded ");
         }
 private void Window_ContentRendered(object sender, EventArgs e)
         {
           System.Diagnostics.Debug.WriteLine("("ამოქმედდა
                              ფანჯრის მოვლენა  ContentRendered");
         }
  private void Window_Closing(object sender,
                      System.ComponentModel.CancelEventArgs e)
         {
           System.Diagnostics.Debug.WriteLine("ამოქმედდა
                                     ფანჯრის მოვლენა  Closing");
```

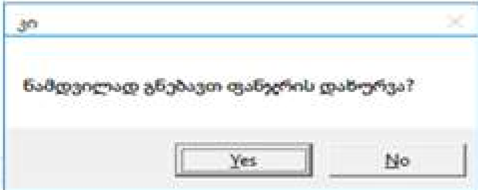
```
    if (MessageBox.Show("ნამდვილად გნებავთ ფანჯრის დახურვა ?",
      "კი", MessageBoxButton.YesNo) == MessageBoxResult.No)
              e.Cancel = true; // არ დაიხუროს
        }
private void Window_Unloaded(object sender, RoutedEventArgs e)
        {
          System.Diagnostics.Debug.WriteLine("ამოქმედდა
                              ფანჯრის მოვლენა Unloaded");
        }
 private void Window_Closed(object sender, EventArgs e)
        {
          System.Diagnostics.Debug.WriteLine("ამოქმედდა
                               ფანჯრის მოვლენა Closed");
        }
 private void Button_Click(object sender, RoutedEventArgs e)
        {
          this.Close();
        }
    }
}
```

5. აპლიკაციის ამუშავებით მიიღება შემდეგი სურათი (ნახ.15.19):



ნახ.15.19

ფანჯარაზე მაუსით „დაკლიკვისას“ გამოვა 15.20 ნახაზზე ნაჩვენები შეტყობინების ფანჯარა.



ნახ.15.20

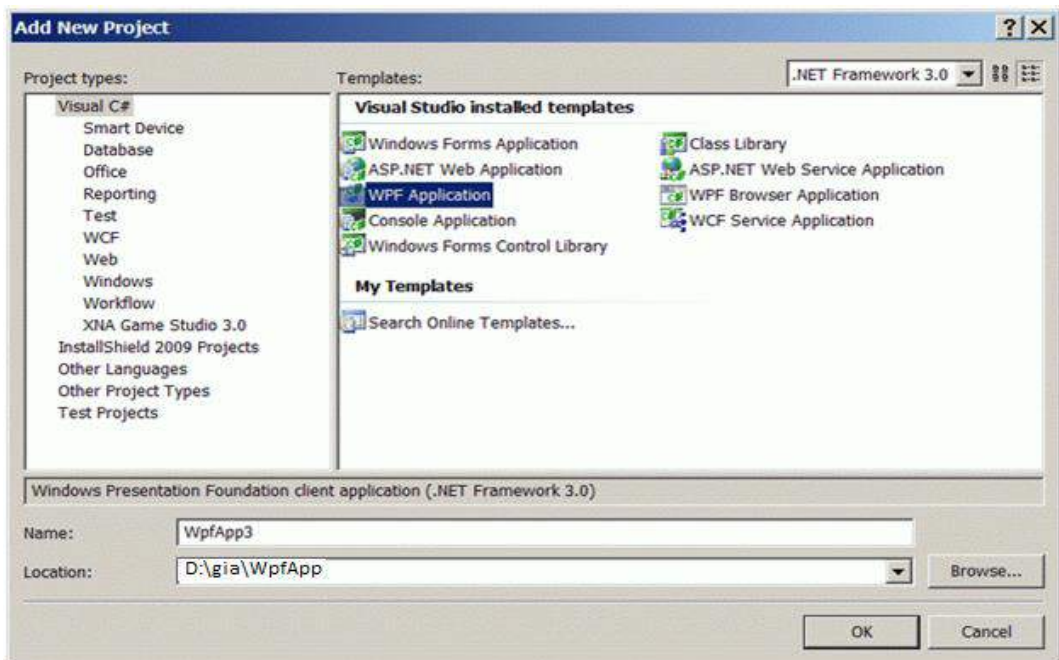
რეალიზაბული პროექტი მუშაობს დიალოგურ რეჟიმში და ახორციელებს ფანჯრების ფუნქციონალობის მართვას.

15.4. მრავალფანჯრიანი პროექტი WPF-ში

ერთ დანართში შესაძლებელია რამდენიმე ფანჯრის შექმნა, რომლებიც შეიძლება ერთდროულად იქნას გახსნილი. მათი დახურვის პროცესი მოითხოვს გარკვეული მართვის განხორციელებას. მაგალითად, თუ დაიხურა ერთი რომელიმე, სხვა რჩება გახსნილი.

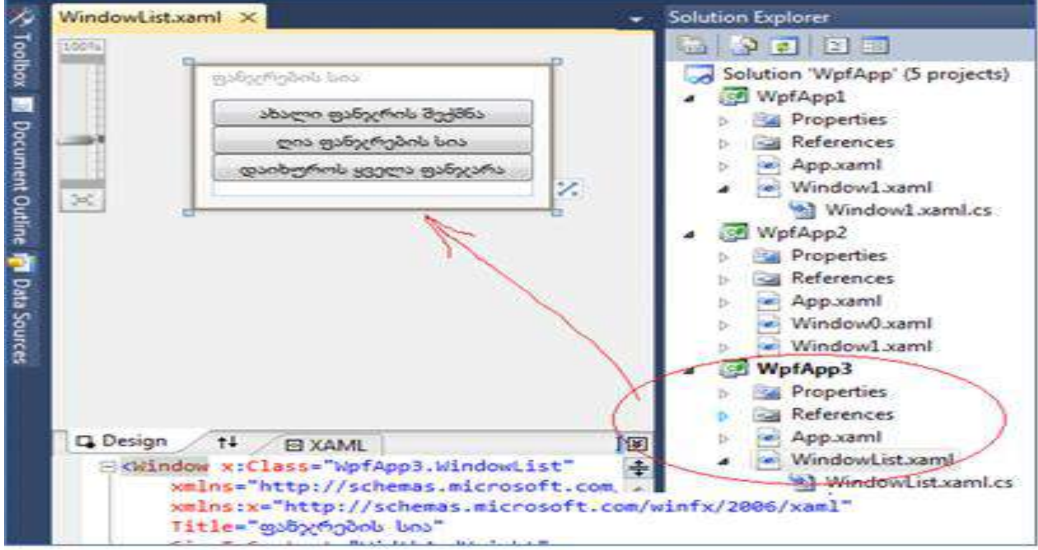
საჭიროა ისეთი პროცესის აგება, როდესაც ყველა ფანჯრის დახურვა იქნება შესაძლებელი ერთდროულად. განვიხილოთ ასეთი პროექტი.

1. დავამატოთ WpfApp -ს Solution-იდან ახალი პროექტი WpfApp3 სახელით:.



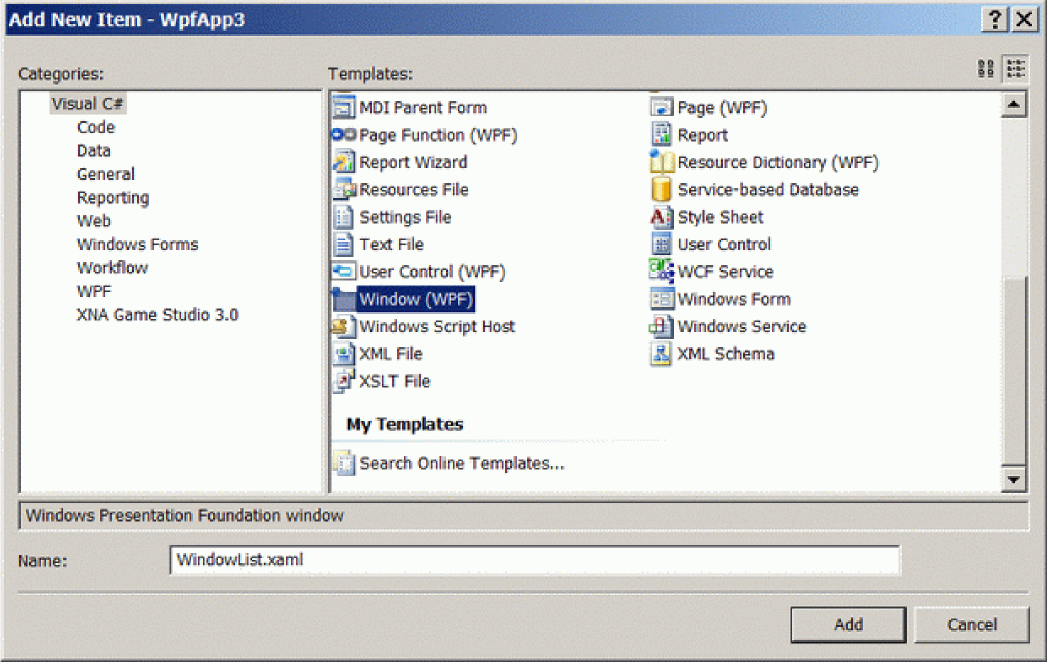
ნახ.15.21

შედეგი:



ნახ.15.22

2. Solution Explorer-ში წავშალოთ Window1.xaml ფაილი და დავამატოთ ახალი ფაილი სახელით WindowList.xaml (ეს შეცვლის ყველგან ამ სახელს).



ნახ.15.23

3. გახსენით App.xaml ფაილი და შეცვალეთ Application-ობიექტის პარამეტრი StartupUri ახალი სასტარტო ფანჯრის ფაილის სახელით.

```
<Application x:Class="WpfApp3.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   StartupUri="WindowList.xaml"
          >
</Application>
```

4. შეავსეთ XAML კოდი WindowList.xaml ფაილისთვის:

```
<Window x:Class="WpfApp3.WindowList"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="ფანჯრების სია"
   SizeToContent="WidthAndHeight"
   MinWidth="200" mc:Ignorable="d"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
d:DesignHeight="116" d:DesignWidth="200">
   <StackPanel>
      <Button Click="NewWindowClicked">ახალი ფანჯრის შექმნა</Button>
      <Button Click="ListOpenWindows">ღია ფანჯრების სია</Button>
      <Button Click="AllCloseWindows">დაიხუროს ყველა ფანჯარა</Button>
   </StackPanel>
</Window>
```

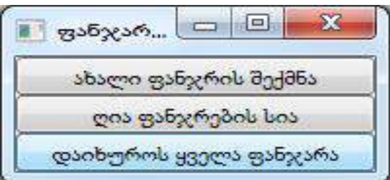
5. შეცვალეთ WindowList.xaml.cs C#-კოდი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
```

```
using System.Windows.Shapes;

namespace WpfApp3
{
 public partial class WindowList : Window
  {
   static int createCount;
   public WindowList()
    {
     InitializeComponent();
    // ფანჯრის სახელის განსაზღვრა და მთავარი ფანჯრის არჩევა
     if (createCount == 3)
      {
       Application.Current.ShutdownMode =
                                    ShutdownMode.OnMainWindowClose;
       Application.Current.MainWindow = this;
       this.Title = "მთავარი ფანჯარა № " +
                                        (createCount++).ToString();
      }
     else
      this.Title = "ფანჯარა № " + (createCount++).ToString();
    }
private void NewWindowClicked(object sender, RoutedEventArgs e)
    {
     new WindowList().Show();
    }
private void ListOpenWindows(object sender, RoutedEventArgs e)
    {
     StringBuilder sb = new StringBuilder();
     foreach (Window openWindow in
                               Application.Current.Windows)
         sb.AppendLine(openWindow.Title);

     MessageBox.Show(sb.ToString(), "დანართის გახსნილი ფანჯრები");
    }
private void AllCloseWindows(object sender, RoutedEventArgs e)
    {
```

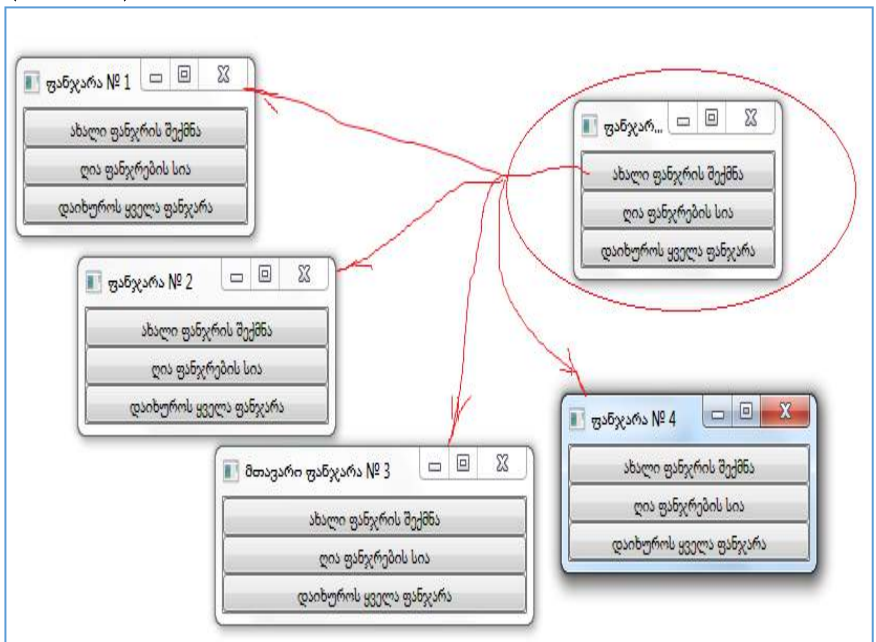
```
    //Application.Current.ShutdownMode =
                                        ShutdownMode.OnExplicitShutdown;
     Application.Current.Shutdown();// ხურავს ფანჯრებს
   }
  }
}
```

6. ავამუშავოთ აპლიკაცია, მივიღებთ ასეთ შედეგს:



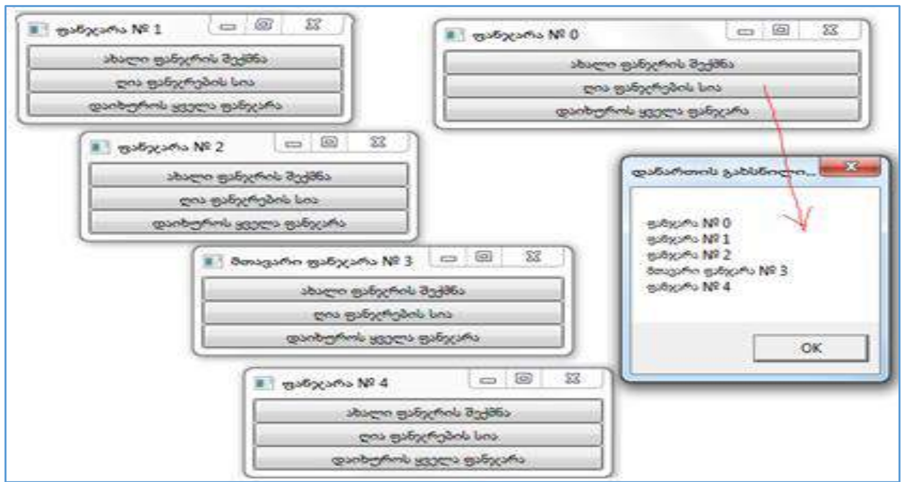
ნახ.15.24

7. პირველ ღილაკზე მაუსით დაკლიკვით იქმნება ახალი ფანჯარა ახალი რიგითი ნომრით (ნახ.15.25).



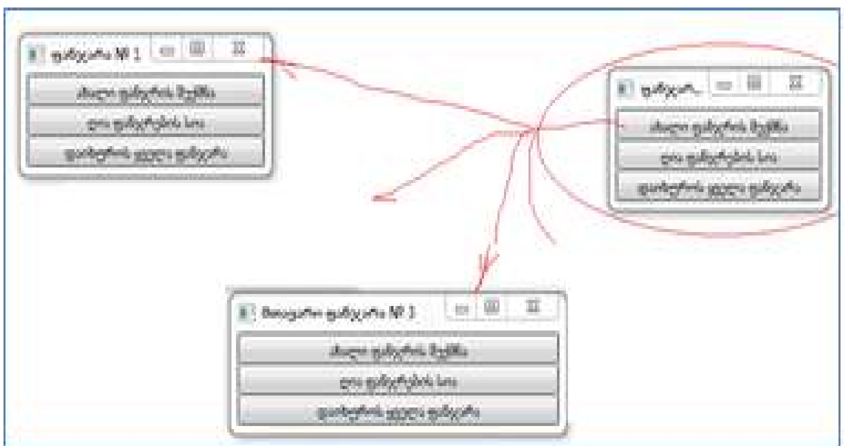
ნახ.15.25

8. მეორე ღილაკზე „**ღია_ფანჯრების_სია**“ მაუსით „დაკლიკვით“ იქმნება ერთი ახალი ფანჯარა „დანართის_გახსნილი_ფანჯრები“ (ნახ.15.26).



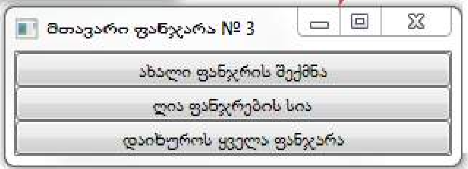
ნახ.15.26

9. მესამე ღილაკზე „**დაიხურის_ყველა_ფანჯარა**“ მაუსით „დაკლიკვით“ ეკრანიდან გაქრება ეს კონკრეტული (მაგალითად, მე–2 და მე–4) ფანჯარა, რომელზეც მაუსით ვიმოქმედეთ (ნახ.15.27).



ნახ.15.27

10. ვინაიდან ჩვენ მე-3 ფანჯარა გვქონდა გამოცხადებული, როგორც მთავარი, ამიტომ მისი დახურვის ღილაკის „დაკლიკვით“ ერთდროულად ყველა ფანჯარა იხურება.

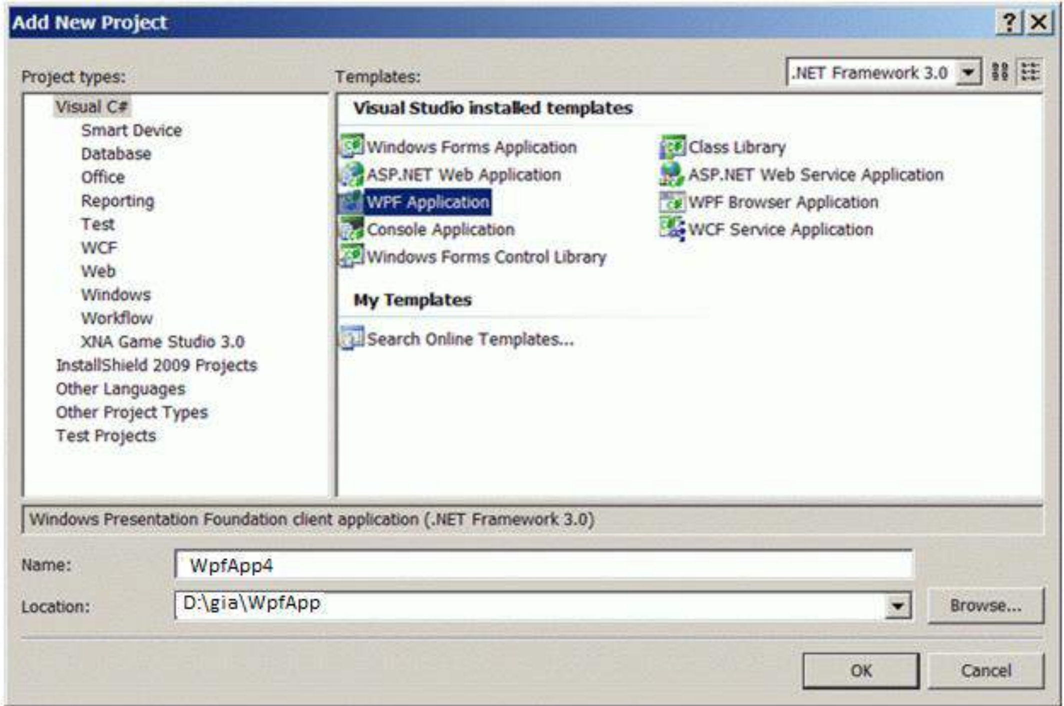


ნახ.2.28

15.5. WPF-ის მომხმარებელთა მართვის ელემენტები

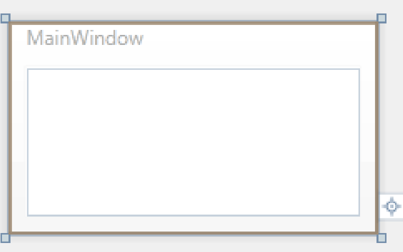
განვიხილოთ WPF-ის აპლიკაციის დამუშავება მომხმარებელთა ინტერფეისების ვიზუალური მართვის ელემენტებით.

1. დავამატოთ WpfApp -ს Solution-იდან ახალი პროექტი WpfApp4 სახელით:.



ნახ.15.29

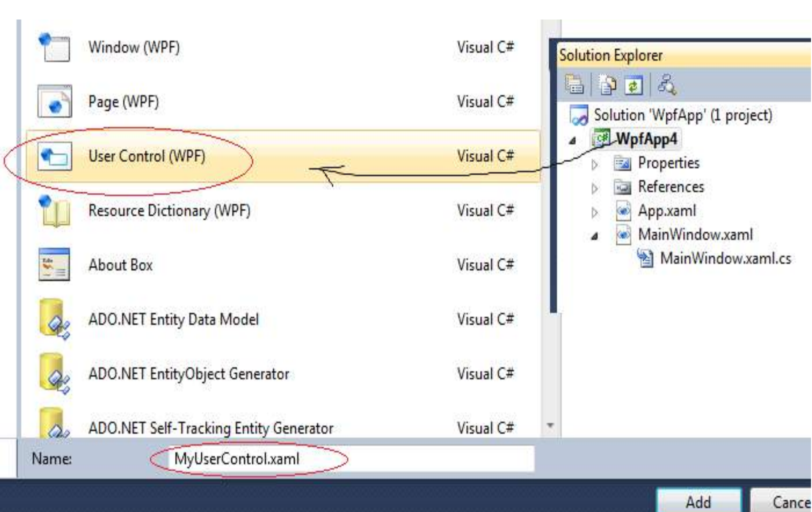
შედეგში ცარიელი ფორმაა:



ნახ.15.30

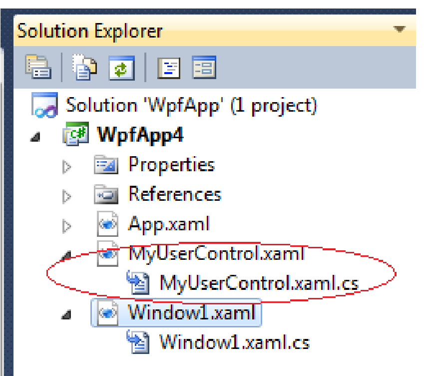
შევცვალოთ სახელი ყველგან Window1-ით.

2. Solution Explorer-ში WpfApp4-s დავამატოთ Add User Control ფაილი და დავამატოთ ახალი ფაილი სახელით MyUserControl.xaml ().



ნახ.15.31

Solution Explorer-ში მიიღება ახალი ფაილი MyUserControl.xaml.



ნახ.15.32

3. გავხსნათ MyUserControl.xaml ფაილი და შევცვალოთ ტექსტი შემდეგი სახით:.

```
<UserControl x:Class="WpfApp4.MyUserControl"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   >
      <Button Click="Button_Click">ჩემი ღილაკი</Button>
</UserControl>
```

4. შევავსოთ C# კოდი MyUserControl.xaml.cs ფაილისათვის:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
```

```
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp4
{
   public partial class MyUserControl : UserControl
   {
      public MyUserControl()
      {
         InitializeComponent();
      }
      private void Button_Click(object sender,
                                                         RoutedEventArgs e)
       {
        MessageBox.Show("მოგესალმებით !", "ელემენტი
                             UserControl");
      }
   }
}
```

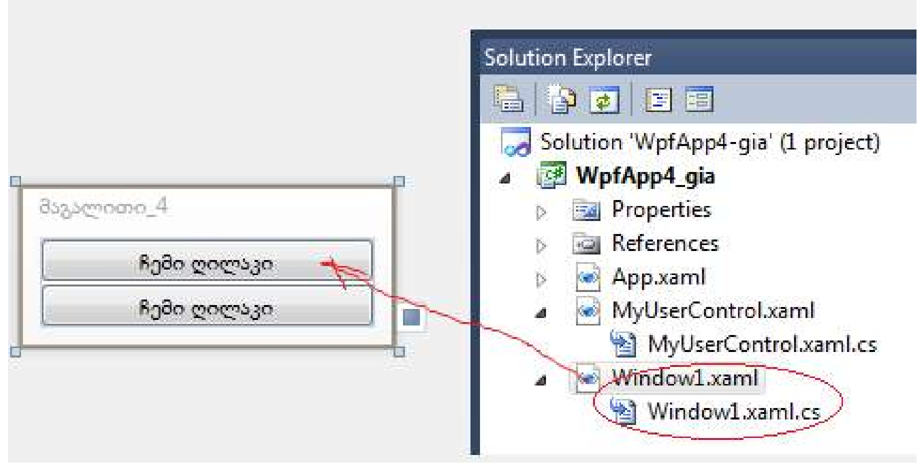
5. გაუშვით პროგრამა კომპილაციაზე, რათა შეიქმნას მომხმარებლის ერთი ღილაკი, რომელსაც შემდგომ გამოიყენებს სხვა ფანჯარაში რამდენჯერმე.

6. შეცვალეთ Window1.xaml კოდი:

```
<Window x:Class="WpfApp4.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:My="clr-namespace:WpfApp4"     (აქ My არის ფსევდონიმი !)
   Title="მაგალითი_4"
   Width="200"
   SizeToContent="Height"
   WindowStartupLocation="CenterScreen"
```

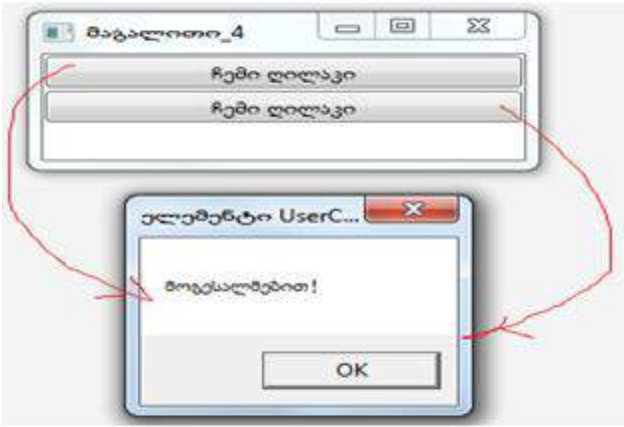
```
    >
  <StackPanel>
    <My:MyUserControl Margin="1"/>
    <My:MyUserControl Margin="1"/>
  </StackPanel>
</Window>
```

მივიღებთ:



ნახ.15.33

7. ავამუშავოთ აპლიკაცია. შედეგი მოცემულია 15.34 ნახაზზე.



ნახ.15.34

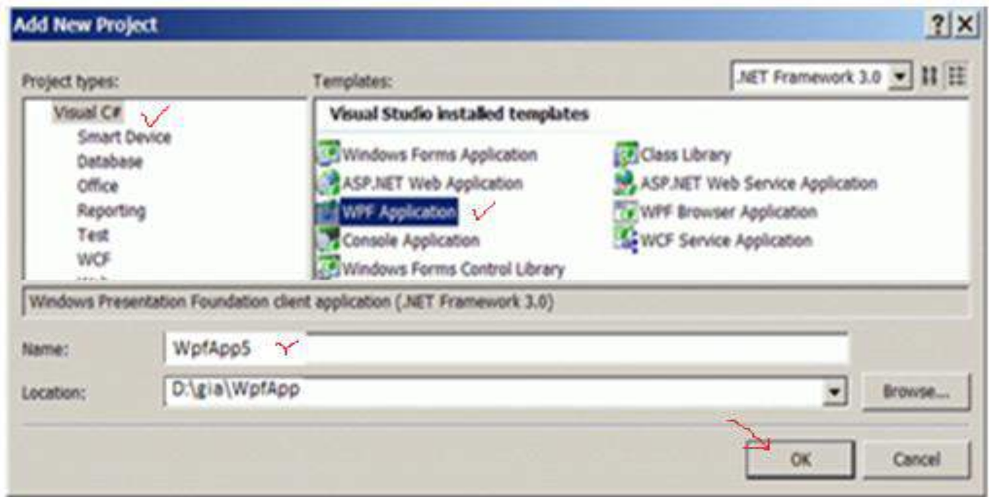
15.6. WPF-ში Web-გვერდების აპლიკაციების შექმნა

WPF-პლატფორმას აქვს ვებგვერდების შექმნის საშუალებები. ასეთი გვერდების გამოყენება ხშირად მომხმარებლისთვის უფრო მოსახერხებელია, ვიდრე ფანჯრული ორგანიზაცია.

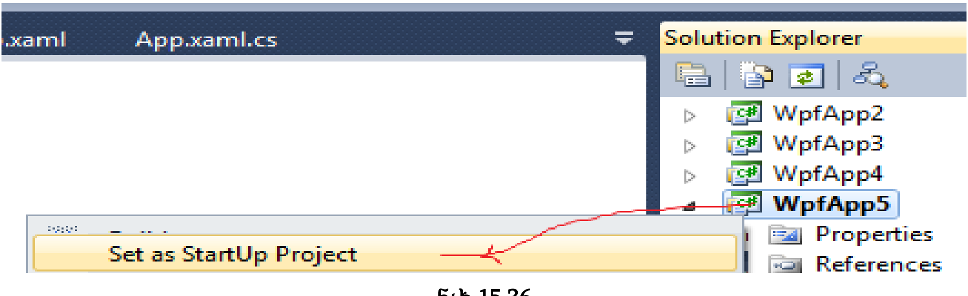
გვერდული დანართის (აპლიკაციის) შიგთავსი ჩაშენდება სპეციალურ ნავიგაციურ კარკასში, რომელსაც აქვს ნავიგაციური კავშირები და ნავიგაციური ჟურნალი.

მისი ძირითადი (ფესვური) კლასია NavigationWindow, რომელიც ამატებს აპლიკაციისთვის **ნავიგაციის სტანდარტულ ინტერფეისს** და მისთვის საჭირო ინფრასტრუქტურას. NavigationWindow კლასი წარმოებულია Window-კლასიდან და აქვს წვდომა დანართის იმავე საშუალებებზე.

1. დავამატოთ WpfApp -ს Solution-იდან ახალი პროექტი WpfApp5 სახელით (ნახ.15.35) და გავხადოთ „სასტარტო" (ნახ.15.36).



ნახ.15.35



ნახ.15.36

2. წავშალოთ ავტომატურად შექმნილი Window1.xaml ფაილი Solution Explorer-ში და ჩავამატოთ მის ნაცვლად Window (WPF)-შაბლონით ახალი ფაილი NavExample.xaml სახელით.

3. გავხსნათ App.xaml ფაილი და შევცვალოთ მასში ატრიბუტი StartupUri="NavExample.xaml"

4. ავამუშავოთ პროექტი WpfApp5. დავრწმუნდეთ, რომ არაა შეცდომები.

5. გავხსნათ ფაილი NavExample.xaml და შევასწოროთ მასში დესკრიპტორული კოდი:

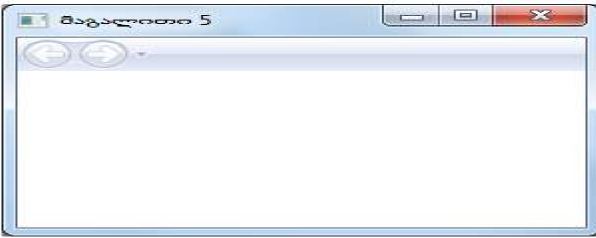
```
<NavigationWindow x:Class="WpfApp5.NavExample"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="მაგალითი 5"
   Height="300"
   Width="300"
   WindowStartupLocation="CenterScreen"
      >
</NavigationWindow>
```

6. გავხსნათ NavExample.xaml.cs ფაილი და შევასწოროთ C# კოდის ტექსტი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

using System.Windows.Navigation;

namespace WpfApp5
{
   public partial class NavExample : NavigationWindow
   {
```

```
        public NavExample()
        {
          InitializeComponent();

          //this.Navigate(new Page1());
        }
      }
    }
```

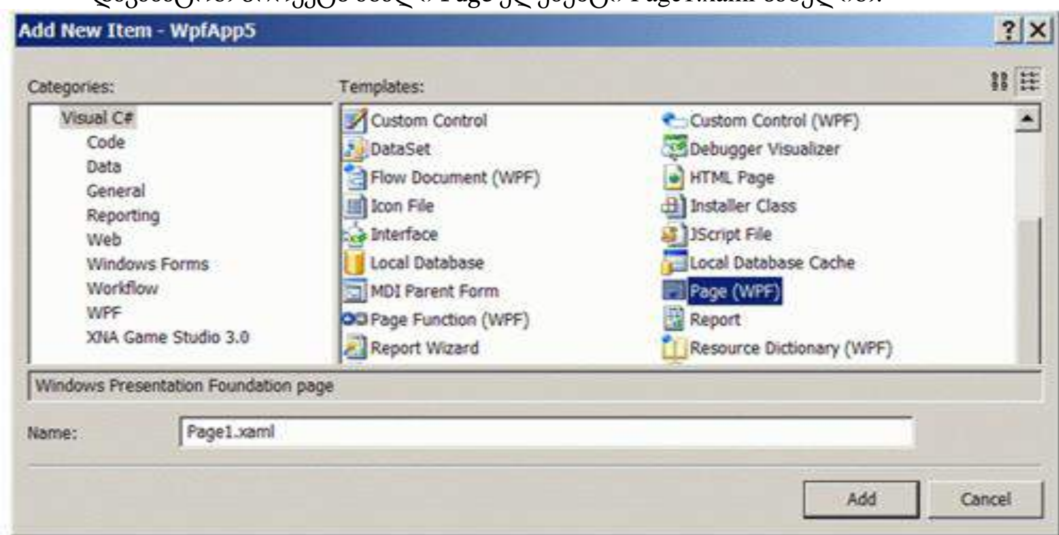
ავამუშავოთ პროექტი, შედეგი იქნება ნახ.15.37, ცარიელი გვერდი ნავიგაციური ელემენტით.



ნახ.15.37

7. ნავიგაციური კვანძის შიგთავსი წარმოდგენილი უნდა იყოს კლასით, რომელიც წარმოებულია ბიბლიოთეკის Page-კლასისგან. შევქმნათ სამი გვერდი და მივაბათ ნავიგაციის კვანძს Navigate() მეთოდის დახმარებით.

- დავამატოთ პროექტს ახალი Page ელემენტი Page1.xaml სახელით.



ნახ.15.38

8. შევასწოროთ Page1.xaml ფაილის ტექსტი:

```
<Page x:Class="WpfApp5.Page1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center" FontSize="24">გვერდი
1</TextBlock>
        <TextBlock></TextBlock>
        <TextBlock>
      <Hyperlink Click="LinkClicked">მე-2 გვერდზე</Hyperlink>
        </TextBlock>
    </StackPanel>
</Page>
```

9. შევასწოროთ Page1.xaml.cs ფაილის კოდი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp5
{
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }
```

```
   private void LinkClicked(object sender, RoutedEventArgs e)
        {
            this.NavigationService.Navigate(page2);
        }
    }
}
```

9. დავამატოთ პროექტს ახალი Page ელემენტი Page2.xaml სახელით. შევასწოროთ xaml-კოდი:

```
<Page x:Class="WpfApp5.Page2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page2"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center" FontSize="24">გვერდი
2</TextBlock>
    </StackPanel>
</Page>
```

10. NavExample.xaml ფაილში დავამატოთ Source-ატრიბუტი, რომელიც მიუერთდება კარკასის გაშვებისას Page1.xaml საწყისი გვერდის შიგთავსს.

```
     <NavigationWindow x:Class="WpfApp5.NavExample"
         ...
         WindowStartupLocation="CenterScreen"
         Source="Page1.xaml"
              >
     </NavigationWindow>
```

11. დავალება: აამუშავეთ პროექტი. შეამოწმეთ ღილაკების მუშაობისუნარიანობა.

დასკვნა:

ამგვარად, ჩვენ შევქმენით კარკასი და ორი ცარიელი გვერდი, რომლებიც არაფერს არ აკეთებს. თითოეული გვერდი ავტონომიურია, შეიძლება მათი შევსება ტულბოქსის ელემენტებით.

გვერდებს შორის გადასვლისას საჭიროა ვიცოდეთ ინფორმაციის გადაცემა ერთი გვერდიდან მეორეში. უნდა არსებობდეს საერთო საფოსტო ყუთი, რომელიც არ იქნება დამოკიდებული გვერდებზე.

WPF-ში მონაცემების გადასაცემად გვერდებს შორის იყენებენ ლექსიკონს (წყვილების მასივი: „გასაღები-მნიშვნელობა“) Application.Current.Properties, ან ინფორმაციის „ჩაკერვას“ უშუალოდ ახალი გვერდის ობიექტში.

(მაგალითის გაგრძელება)

12. Page1-ზე დავამატოთ სახელმინიჭებული ტექსტური ველი შემდეგი სახით:

```
<Page x:Class="WpfApp5.Page1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Page1"
      >
   <StackPanel>
      <TextBlock TextAlignment="Center"
                          FontSize="24">გვერდი 1</TextBlock>
      <TextBlock></TextBlock>
      <Label>შეიტანეთ თქვენი სახელი: </Label>
      <TextBox Name="nameBox" Width="200"></TextBox>
      <TextBlock></TextBlock>
      <TextBlock>
    <Hyperlink Click="LinkClicked">მე-2 გვერდზე</Hyperlink>
      </TextBlock>
   </StackPanel>
</Page>
```

TextBox-ობიექტს მივანიჭეთ სახელი, რათა შეიძლებოდეს მასზე მიმართვა კოდიდან.

13. შევცვალოთ Page1 გვერდის კოდი შემდეგი სახით;

```
using System;
. . .
namespace WpfApp5
{
  public partial class Page1 : Page
   {
   public Page1()
      {
         InitializeComponent();
```

```
        }
 private void LinkClicked(object sender, RoutedEventArgs e)
        {
          Page2 page2 = new Page2();
          page2.Message = nameBox.Text + " !!!"; // ინფორმაციის
                                                            //„ჩაკერვა“ ობიექტში
          this.NavigationService.Navigate(page2);
        }
    }
}
```

14. დავამატოთ Page2 გვერდზე სახელმინიჭებული ტექსტური ჭდე და ჰიპერლინკი Page3-ზე გადასასვლელად. აგრეთვე მომამზადეთ გვერდის მოვლენა Loaded და მოვლენა Click ჰიპერლინკისათვის.

```
<Page x:Class="WpfApp5.Page2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page2"
    Loaded="Page_Loaded"
        >
    <StackPanel>
        <TextBlock TextAlignment="Center"
                              FontSize="24">გვერდი 2</TextBlock>
        <TextBlock></TextBlock>
        <TextBlock>მოგესალმებით </TextBlock>
        <Label Name="nameLabel"></Label>
        <TextBlock Margin="0,10">   <!--Отступ сверху-->
 <Hyperlink Click="LinkClicked">მე-3 გვერდზე</Hyperlink>
        </TextBlock>
    </StackPanel>
</Page>
```

Label ობიექტს მივანიჭეთ სახელი, რათა კოდიდან შეიძლებოდეს მასზე მიმართვა.

15. დავამატოთ Page2-ის კოდს public თვისება, ტექსტურ ჭდეზე გადაცემული ტექსტის მისანიჭებელი კოდი Loaded-მოვლენაში და შემდეგ გვერდზე გადასვლის კოდი.

```
using System;
using System.Collections.Generic;
using System.Text;
```

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfApp5
{
    public partial class Page2 : Page
    {
        public Page2()
        {
            InitializeComponent();
        }

        string message;
        public string Message
        {
            set { message = value; }
        }
   private void Page_Loaded(object sender, RoutedEventArgs e)
        {
            nameLabel.Content = message;
        }
   private void LinkClicked(object sender, RoutedEventArgs e)
        {
            Page3 page3 = new Page3();
            this.NavigationService.Navigate(page3);
        }
    }
}
```

16. აამუშავეთ აპლიკაცია. ინფორმაცია გადაეცემა, ოღონდ ღილაკის ამუშავებით.

(!) ნავიგაციის ღილაკით ახალი ინფორმაცია ტექსტური ველიდან არ გადაიცემა. ინფორმაცია აიღება ისტორიის ჟურნალიდან.

17. Page3 გვერდისათვის შეავსეთ xaml-კოდი:

```
<Page x:Class="WpfApp5.Page3"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Page3">
    <StackPanel>
        <!--გვერდის კონტექსტის სათაური -->
        <TextBlock TextAlignment="Center"
                   FontSize="24">გვერდი 3
            <TextBlock.Margin>0,0,0,10</TextBlock.Margin>
        </TextBlock>
      <!--პირველი ღილაკი-->
        <Button Content="Push Me!" FontSize="22" Width="175"
                            Height="50" Click="Button_Click">
            <Button.Effect>
                <DropShadowEffect />
            </Button.Effect>
        </Button>
      <!--მეორე ღილაკი-->
        <Button FontSize="22" Height="50" Width="175"
                          Margin="0,10" Click="Button_Click">
            "დამკლიკე"
            <Button.Effect>
                <DropShadowEffect />
            </Button.Effect>
            <Button.Foreground>
        <LinearGradientBrush StartPoint="1,0" EndPoint="0,0">
                 <GradientStop Color="Red" Offset="0" />
                 <GradientStop Color="Orange" Offset=".17" />
                 <GradientStop Color="Yellow" Offset=".33" />
                 <GradientStop Color="Green" Offset=".5" />
                 <GradientStop Color="CornflowerBlue"
                                             Offset=".67" />
                <GradientStop Color="Blue" Offset=".84" />
               <GradientStop Color="BlueViolet" Offset="1" />
               </LinearGradientBrush>
            </Button.Foreground>
```

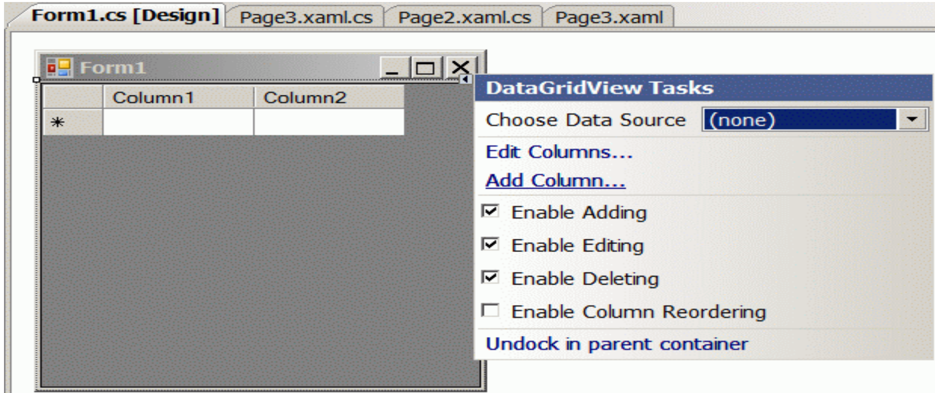
```
        <Button.Background>
    <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
                <GradientStop Color="Red" Offset="0" />
               <GradientStop Color="Orange" Offset=".17" />
               <GradientStop Color="Yellow" Offset=".33" />
                <GradientStop Color="Green" Offset=".5" />
                <GradientStop Color="CornflowerBlue"
                                            Offset=".67" />
                <GradientStop Color="Blue" Offset=".84" />
             <GradientStop Color="BlueViolet" Offset="1" />
              </LinearGradientBrush>
        </Button.Background>
    </Button>
  </StackPanel>
</Page>
```

Click - მოვლენის დამმუშავებელი უნდა ჩაიწეროს ხელით, რათა ის შეიქმნას კოდის ნაწილში.

18. ხანამ შევავსებთ Page3 გვერდის კოდის ნაწილს, საჭიროა პროექტს დაემატოს ახალი ფორმა. ამით შესაძლებელია WPF და Windows Forms ტექნოლოგიების ერთობლივად მუშაობის დემონსტრირება. ღილაკებზე დავდოთ ფორმის ამუშავების კოდი.

19. დავამატოთ WpfApp5-ში ახალი ფორმა Form1.cs

20. Form1 ფორმაზე ტულბოქსიდან დავდოთ DataGridView ელემენტი. დავაყენოთ მისი თვისება Dock=Fill და დავამატოთ ინტელექტუალური დესკრიპტორიდან (SmartTag) ორი სვეტი



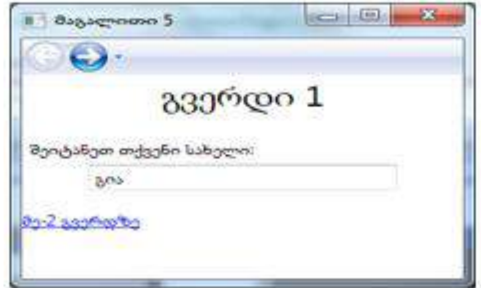
ნახ.15.39

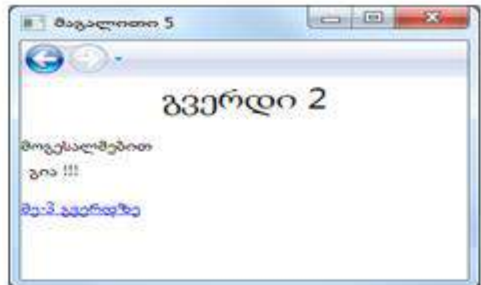
21. ღილაკების საერთო დამმუშავებელი Page3 კოდის ნაწილში შევავსოთ ასე:

```
//--- ლისტინგი -------
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApp5
{
   public partial class Page3 : Page
   {
      public Page3()
      {
         InitializeComponent();
      }
 private void Button_Click(object sender, RoutedEventArgs e)
      {
         Form1 frm = new Form1();
         frm.ShowInTaskbar = false; //  ფორმის ღილაკი არ
                                              //გამოჩნდეს ამოცანების პანელზე
         frm.Show();
      }
   }
}
```

22. ავამუშავოთ პროექტი. დავაკვირდეთ ახალი გვერდის დიზაინს. იხსნება Form1 -იც, რაც ადასტურების WPF და Windows Form ტექნოლოგიების ერთობლივი მუშაობის შესაძლებლობას.

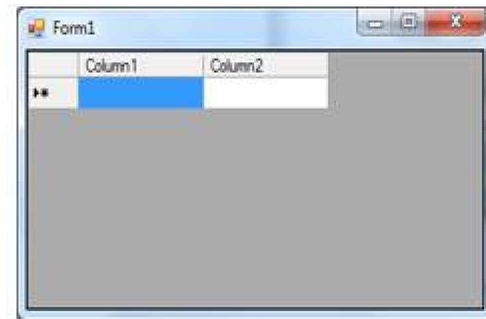
შედეგები (ნახ.15.40–ა,ბ,გ):





ნახ.15.40–ა

ნახ.15.40–ბ



ნახ.15.40–გ

15.7. WPF-ში ფუნჯის სახეები და ფერების შერჩევა

WPF-აპლიკაციების დამუშავება მომხმარებელთა ინტერფეისების გრაფიკული ვიზუალური მართვის ელემენტებით, კერძოდ, ფუნჯებისა და ფერთა პალიტრის გამოყენებით. განხილული იქნება:

- გვერდი_1. SolidColorBrush - ერთგვაროვანი ფუნჯის გამოყენება
- გვერდი_2. Brushes კლასის ფერების გამოყენება
- გვერდი_3. LinearGradientBrush - წრფივი გრადიენტული ფუნის გამოყენება
- გვერდი_4. გრადიენტის მიმართულების პერიოდული ცვლა
- გვერდი_5. საფეხურებრივი შეფერადება გრადიენტებით
- გვერდი_6. RadialGradientBrush - შეფერადება რადიალური გრადიენტით

1. თეორიული ნაწილი: ფუნჯის სახეები WPF-ში

WPF–ში ობიექტების გარეგნული დიზაინის გასაფორმებლად არსებობს როგორც უშუალო ასაწყობი თვისებები, ასევე ფუნჯი–ინსტრუმენტები (Brush).

თვისებები შემდეგი სახისაა: გამჭვირვალეობა (Opacity), ხილვადობა (Visibility), შევსება (Fill), დაჩრდილვა (Stroke) – /შტრიხები/, ფონი (Background), წინა პლანი (Foreground), საზღვარი (BorderBrush), გამჭვირვალეობის ნიღაბი (Opacity masks).

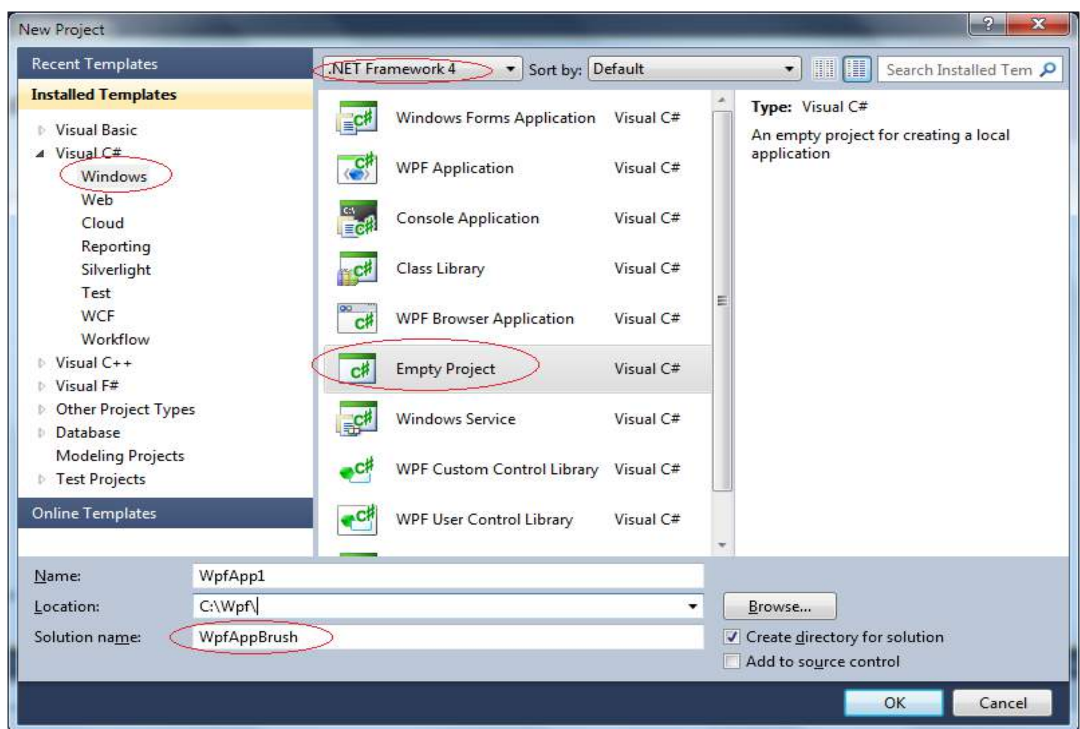
ფუნჯები შემდეგი სახისაა: ერთფეროვანი (Solid color brush), წრფივი გრადიენ-ტული (Linear gradient brush), რადიალური გრადიენტული (Radia l gradient brush), რასტრული გამოსახულების (Image brush), ვექტორული გამოსახულების (Drawing brush), ვიზუალური ეფექტების (Visual brush).

შესაძლებელია ფუნჯების კონვერტირება რესურსებად და შემდგომ მათი მრავალჯერადი გამოყენება სხვადასხვა ობიექტისათვის. WPF–ს აქვს სპეციალური *რასტრული ეფექტების* (Bitmap effects) მიღწევის საშუალებანი:

- არამკაფიო (Blur)
- გარე ნათების (Outer glow)
- ჩრდილი (Drop shadow)
- ზოლი (Bevel) – კანტი
- რელიეფური (Emboss) –ჭედური

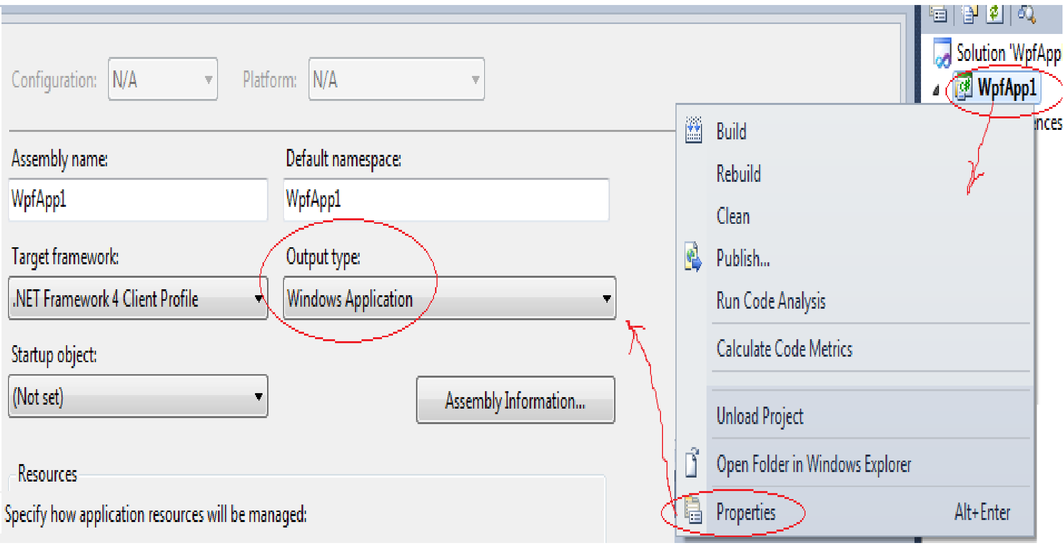
2. პრაქტიკული ნაწილი – აპლიკაციის აგება

1. Visual Studio-ს Solution Explorer-ში WpfAppBrush სახელით შევქმნათ ახალი ცარიელი პროექტი (**Empty Project**) WpfApp1 (ნახ.14.41).



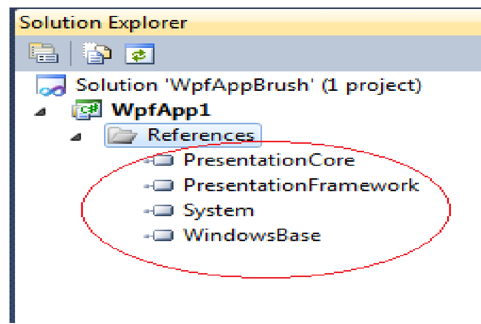
ნახ.15.41

2. WpfApp1-ისთვის გამოვიძახოთ კონტექსტური მენიუ და **Properties**–ის **Application**–ში Output type პარამეტრისთვის ავირჩიოთ მნიშვნელობა **Windows Application.**



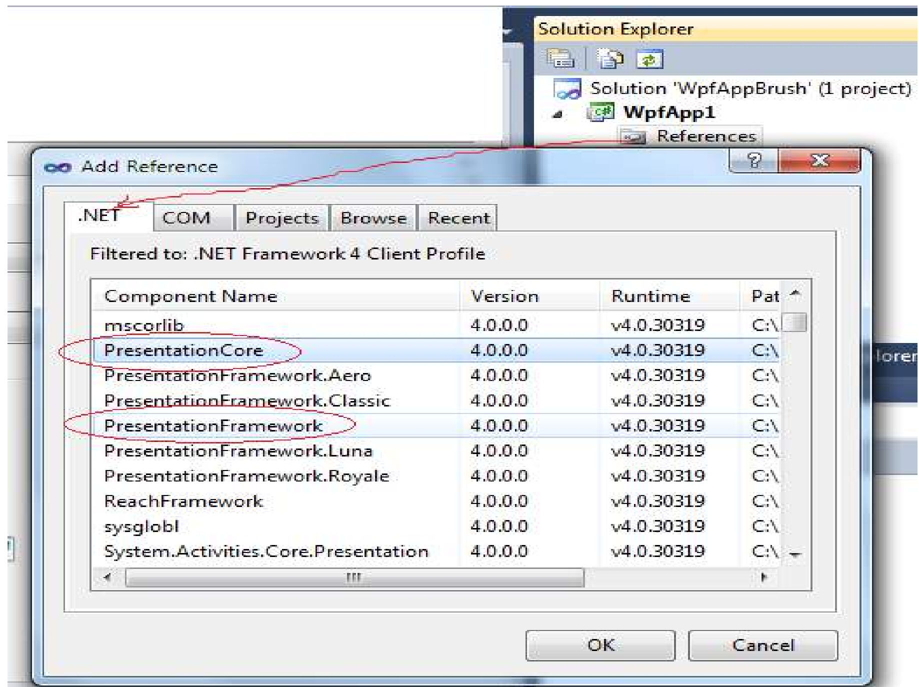
ნახ.15.42

3. პროექტის **References** კვანძს დავუმატოთ **.NET Framework**–ის WPF–პლატფორმის შემდეგი ბიბლიოთეკები: PresentationCore, PresentationFramework, WindowsBase და System (ესაა საერთო ბიბლიოთეკა აპლიკაციის ასამუშავებლად). 2.35–36 ნახაზებზე ნაჩვენებია ეს პროცედურები.

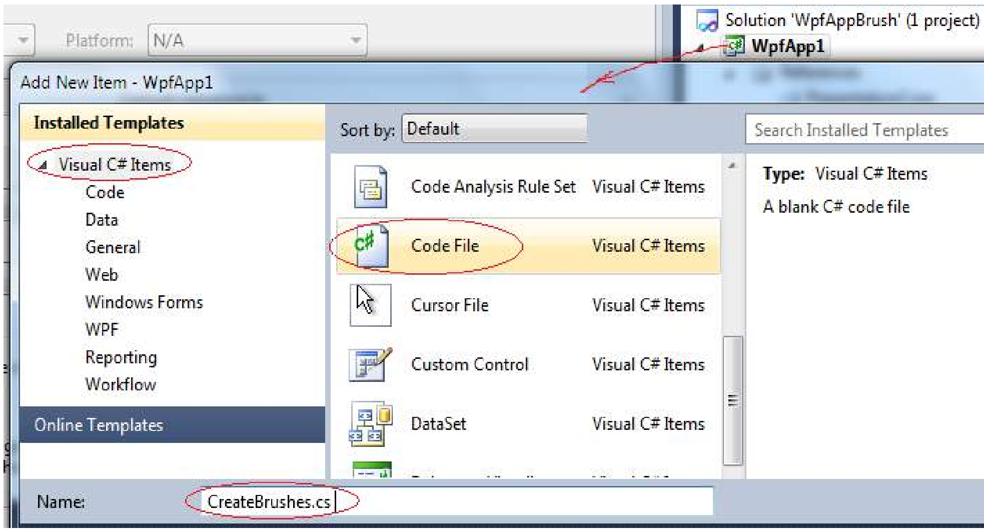


ნახ.15.43

4. პროექტს დავუმატოთ ახალი ფაილი სახელით CreateBrushes.cs (ნახ.15.45).



ნახ.15.44



ნახ.15.45

5. შევავსოთ ფაილი შემდეგი კოდით:

```
// — ლისტინგი —
using System;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Navigation;
using System.Windows.Controls;
using System.Reflection;
namespace WpfApp1
{
  class CreateBrushes : NavigationWindow
  {
    // შესასვლელი წერტილი
    [STAThread] // ატრიბუტი ერთნაკადიანი აპლიკაციისათვის
    public static void Main()
    {
      Application app = new Application();
      app.Run(new CreateBrushes());
    }

    // კონსტრუქტორი
    public CreateBrushes()
    {
      this.Width = 300;
      this.Height = 300;
     this.Title = "1-ელი მაგალითის აპლიკაციის კარკასი";
      //this.ShowsNavigationUI = false; // ნავიგაციური ინტერფეისის დამალვა
    }
  }
}
```

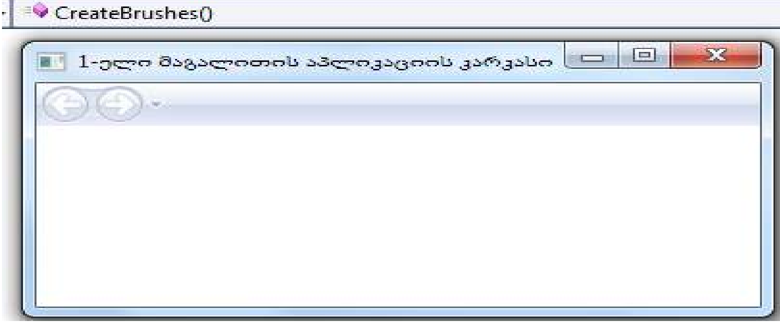
ამოცანა:

ავაგოთ მრავალგვერდიანი აპლიკაცია რამდენიმე მაგალითისათვის (პროექტებისათვის). წინასწარ გვერდებისათვის შევქმენით კარკასი შიგთავსით, რომელშიც ნავიგაციის კვანძია. იგი შეიძლება გაითიშოს ShowsNavigationUI – თვისებით.

კარკასი შექმნილია ნულიდან სუფთა **C#** კოდის გამოყენებით. არ ვიყენებთ XAML–ს. გვერდული კარკასი იქმნება NavigationWindow კლასით, რომელიც წარმოებულია Window კლასიდან. ჩვენ ვაფართოვებთ NavigationWindow ბიბლიოთეკურ კლასს

მომხმარებლის კლასით CreateBrushes. ამგვარად, მისთვის მემკვიდრეობით ხელმისაწვდომი იქნება Window საბაზო კლასის წევრები (მონაცემები და მეთოდები).

6. ავამუშავოთ აპლიკაცია – მივიღებთ „მკვდარ“ კარკასს (გამორთულია ისრები).



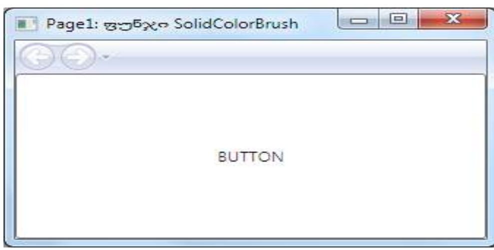
ნახ.15.46

შემდგომში ამ კარკასში უნდა მოვათავსოთ ახალ-ახალი გვერდები, რომლებიც იქმნება Page ბიბლიოთეკური კლასის გაფართოებით და გარკვეული თვისებების მინიჭებით. კარკასის ნავიგაციური მექანიზმი ამოქმედდება NavigationWindow საბაზო კლასის ხელშეწყობით.

➢ გვერდი_1: ერთგვაროვანი ფუნჯი SolidColorBrush

ავაგოთ პირველი გვერდი, რომელზეც კლიენტის მთლიან არეში მოთავსებული იქნება ღილაკი "BUTON". ეს ღილაკი უნდა იყოს ერთგვაროვანი, ნაცრისფერის გრადაციით (მაუსის კურსორის მოცილებისას, და მოცისფერო გრადაციით – მიახლოვებისას).

გვერდი აიგება ხელით, როგორც გაფართოებული კლასი Page ბიბლიოთეკური კლასიდან.



ნახ.15.47

8. ჩავამატოთ CreateBrushes.cs კოდში Page1 გვერდის შექმნის კოდი.

```
namespace WpfApp1
{
 // კლასი - Page1 გვერდის გაფართოებისთვის
  class Page1 : Page
  {
    // ფუნჯის ველის შექმნა და ინიციალიზაცია თეთრი ფერით
   SolidColorBrush brush = new  SolidColorBrush(Colors.White);
   Button btnPage1 = new Button();

   public Page1()
    {
      this.WindowTitle = "Page1: ფუნჯი SolidColorBrush";
        // ღილაკის შექმნა
      btnPage1.Content = "BUTTON";
      btnPage1.Background = brush;
     // დამმუშავებლების რეგისტრაცია
     btnPage1.Click += new outedEventHandler(btnPage1_Click);
          btnPage1.MouseMove += new
                                        MouseEventHandler(btnPage1_MouseMove);
     this.Content = btnPage1; // ღილაკის მოთავსება გვერდზე
    }

     // ღილაკის ფონის ფერის შეცვლა Page1 გვერდზე
    void btnPage1_MouseMove(object sender, MouseEventArgs e)
      {
      // კლიენტის სამუშაო არის სიგანე ფანჯრის ჩარჩოს გარეშე
        double width = btnPage1.ActualWidth;
          // კლიენტის სამუშაო არის სიმაღლე ფანჯრის ჩარჩოს და
                                                                    //  სათაურის გარეშე
       double height = btnPage1.ActualHeight;
                // კურსორის წერტილი ფანჯრის კლიენტის არეზე
       Point ptMouse = e.GetPosition(btnPage1);
                // კლიენტის არის ცენტრი
       Point ptCenter = new Point(width / 2, height / 2);
               // კურსორის გადახრა ცენტრიდან
             Vector vectMouse = ptMouse - ptCenter;
                // კურსორის მდებარეობის კუთხე
```

```
      double angle = Math.Atan2(vectMouse.Y, vectMouse.X);
            // ჩაწერილი ელიფსი
      Vector vectEllipse = new Vector(width / 2 *
                   Math.Cos(angle), height / 2 * Math.Sin(angle));
// იზოკლინა (ღილაკის ცენტრში შავი) მრგვალდება ერთ ბაიტამდე
        Byte byLevel = (byte)(255 * (Math.Min(1,
                          vectMouse.Length / vectEllipse.Length)));
      // ველზე მიბმა
          Color color = brush.Color;
        // ფერები, იზოკლინის პროპორციულები
         color.R = color.G = color.B = byLevel;

        brush.Color = color;// იცვლება ფანჯრის ფონის
                              // ფერი თანაბარი შავი ფერის იზოკლინით
      }

      // გადასვლა მეორე გვერდზე
      Page2 page2;
      void btnPage1_Click(object sender, RoutedEventArgs e)
      {
        if (!this.NavigationService.CanGoForward)
          page2 = new Page2();// იქმნება მხოლოდ ერთხელ
        this.NavigationService.Navigate(page2);
      }
   }
}
```

9. კარკასის კოდის კონსტრუქტორში ჩავამატოთ საწყისი გვერდის ობიექტის შექმნის კოდი:

```
      // კონსტრუქტორი
        public CreateBrushes()
        {
            ...
            // საწყისი გვერდის შექმნა
         Page1 page1 = new Page1();
         this.Content = page1;   // გვერდის მოთავსება კარკასში
        }
```

10. ავამუშავოთ პროგრამა, მივიღებთ 2.39 ნახაზს, კარკასში მოთავსდა გვერდი ღილაკით. ღილაკი ჯერ არ ფუნქციონირებს.

- **გვერდი_2: Brushes კლასის ფერების გამოყენება:**

11. ავაგოთ მეორე გვერდი Page2, რომელზეც გადასვლა განხორციელდება BUTTON ღილაკით. კოდის ტექსტი იქნება ასეთი:

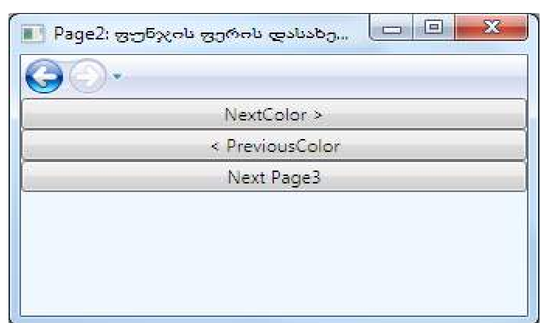
```
namespace WpfApp1
{
  // კლასი - Page2 გვერდის გაფართოება ----
  class Page2 : Page
  {
    int index = 0;// ფერის ნომერი
    PropertyInfo[] props;// თვისებების მასივი

    public Page2()
    {
// რეფლექსიის გამოყენება Brushes კლასის თვისებების წასაკითხად
      props = typeof(Brushes).GetProperties(
        BindingFlags.Public | BindingFlags.Static);

      // პანელის შექმნა
      StackPanel stackPanel = new StackPanel();
      this.Content = stackPanel;// გვერდთან მიერთება

      Button btn = new Button();
      btn.Name = "ButtonNextColor";
      btn.Content = "NextColor >";
      btn.Click += new RoutedEventHandler(btn_Click);
      stackPanel.Children.Add(btn);// პანელზე დამატება

      btn = new Button();
      btn.Content = "< PreviousColor";
      btn.Click += new RoutedEventHandler(btn_Click);
      stackPanel.Children.Add(btn);

      btn = new Button();
```

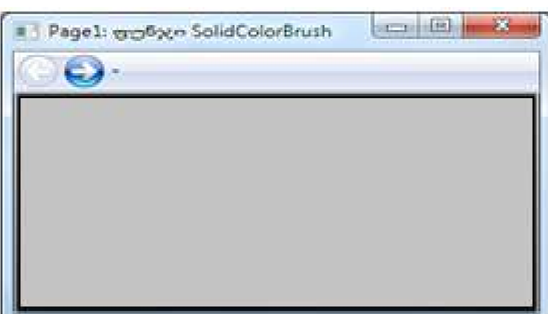
```
        btn.Content = "Next Page3";
      btn.Click += new RoutedEventHandler(btnPage2_Click);
        stackPanel.Children.Add(btn);
           // აქტიურდება გვერდის ყოველი წარმოდგენისას
      this.Loaded += new RoutedEventHandler(Page2_Loaded);
    }
   void Page2_Loaded(object sender, RoutedEventArgs e)
    {
        // ვარიანტი
        //this.NavigationService.LoadCompleted +=
                                    NavigationService_LoadCompleted;
        SetTitleAndBackground();
    }
   void NavigationService_LoadCompleted(object sender,
                                                    NavigationEventArgs e)
    {
        // ვარიანტი
        //SetTitleAndBackground();
    }
 // გვერდის სათაურისა და ფერის შეცვლის ღილაკების დამმუშავებლები
   void btn_Click(object sender, RoutedEventArgs e)
    {
     // ღილაკის ამოცნობა სახელით და ინდექსის კორექტირება
        if (((Button)sender).Name == "ButtonNextColor")
           index += 1;
        else
           index += props.Length - 1;

        index %= props.Length;// მოდულის (%) ოპერაცია
        SetTitleAndBackground();
    }
  // გვერდისათვის სათაურის და ფონის ფერის დაყენება
  void SetTitleAndBackground()
   {
    this.WindowTitle = "Page2: ფუნჯის ფერის დასახელება-"+
                                                     props[index].Name;
    this.Background = (Brush)props[index].GetValue(null, null);
```

```
      }
     // გადასვლა მესამე გვერდზე
       Page3 page3;
      void btnPage2_Click(object sender, RoutedEventArgs e)
       {
          if (!this.NavigationService.CanGoForward)
            page3 = new Page3();// იქმნება მხოლოდ ერთხელ
          this.NavigationService.Navigate(page3);
       }
    }
}
```

Page1–ის კოდის ბოლოში ნაჩვენები იყო Page2–ის გამოძახების ფრაგმენტი, ახალ გვერდზე გადასასვლელად.

12. ავამუშავოთ პროგრამა და კარკასის ნავიგაციის მარჯვენა ღილაკი:





ნახ.15.48

➢ **გვერდი_3: წრფივი გრადიენტული ფუნჯის – LinearGradientBrush გამოყენება**

ერთგვაროვანი ფუნჯის ალტერნატიულია გრადიენტული ფუნჯი, რომლითაც შესაძლებელია ორი ან მეტი ფერის ერთმანეთში თანდათანობითი გადასვლის იმიტაცია.

მარტივ შემთხვევაში წრფივი გრადიენტული ფუნქცის შესაქმნელად საკმარისია ორი ნაპირა წერტილის და მათ შორის ფერების განსაზღვრა. გრადიენტული ფერთა განაწილება მოხდება წერტილებს შორის შემაერთებელი წრფის პერპენდიკულარულად და ზედაპირი მართკუთხედის ფარგლებში შეივსება მითითებული ფერებით.

წრფივი გრადიენტის ფუნქცის ობიექტის კონსტრუქტორს აქვს რამდენიმე გადატვირთვა, რომელთაგან ერთში მიეთითება: - ორი წერტილი და ორი ფერი; - ორი ფერი, გრადიენტის ვექტორის მიბმის საწყისი წერტილი და შეფერადების მიმართულების დახრის კუთხე.

წერტილები იძლევა ფარდობით (გამოუცხადებლად) და აბსოლუტურ კოორდინატებს მართკუთხედის შიგნით, რომელიც განისაზღვრება MappingMode თვისებით და მისი მნიშვნელობით BrushMappingMode ჩამონათვალში.

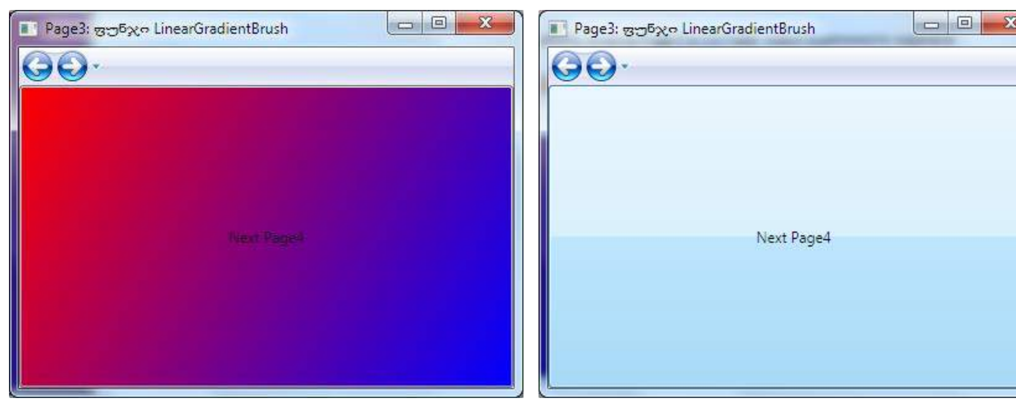
13. შევქმნათ ახალი Page3 გვერდი CreateBrushes.cs ფაილში, რომელზეც ილუსტრირებული იქნება წრფივი გრადიენტული შეფერადება.

```
namespace WpfApp1
{  class Page3 : Page
  {  public Page3()
    {  this.WindowTitle = "Page3: ფუნქი LinearGradientBrush";
      Button btn = new Button();
      btn.Content="Next Page4";
      btn.Click += new RoutedEventHandler(btn3_Click);
      this.Content = btn;
    // გრადიენტის შექმნა და მიერთება
    LinearGradientBrush brush = new LinearGradientBrush(
  Colors.Red, Colors.Blue, new Point(0, 0), new Point(1, 1));
     btn.Background = brush;
  }
   // გადასვლა მეოთხე გვერდზე
   Page4 page4;
   void btn3_Click(object sender, RoutedEventArgs e)
    {
     if (!this.NavigationService.CanGoForward)
       page4 = new Page4();// იქმნება მხოლოდ ერთხელ


     this.NavigationService.Navigate(page4);
    }
  }
}
```

14. ჩავსვათ Page2 კლასის ღილაკში Page3 გვერდის ეგზემპლარის შექმნის კოდი:

```
// გადასვლა მესამე გვერდზე
    Page3 page3;
    void btnPage2_Click(object sender, RoutedEventArgs e)
    {
      if (!this.NavigationService.CanGoForward)
        page3 = new Page3();// იქმნება მხოლოდ ერთხელ

                this.NavigationService.Navigate(page3);
    }
```

15. ავამუშავოთ აპლიკაცია, მივიღებთ (ნახ.15.48):



**ნახ.15.48. მაუსის კურსორის მდებარეობით
იცვლება ფერები**

- **გვერდი_4: გრადიენტის მიმართულების პერიოდული ცვლა**

თუ მონაკვეთი ნაკლებია შემზღუდველ მართკუთხედზე, მაშინ ნაპირა წერტილებს გარეთ ფერის გრადიენტი იცვლის მიმართულებას საწინააღმდეგოზე და ფერის შევსება გრძელდება ისეთივე ინტენსიურობით, როგორც ეს იყო ინტერვალს შიგნით. LinearGradientBrush ფუნჯის ასეთი ქცევა განისაზღვრება SpreatMethod-ის GradientBrush თვისებით და მისი მნიშვნელობით GradientSpreadMethod -ჩამონათლიდან.

16. შევქმნათ ახალი Page4 გვერდი CreateBrushes.cs -ში, რომელშიც ილუსტრირებული იქნება ფერების ტალღისებური გრადიენტული შევსება.

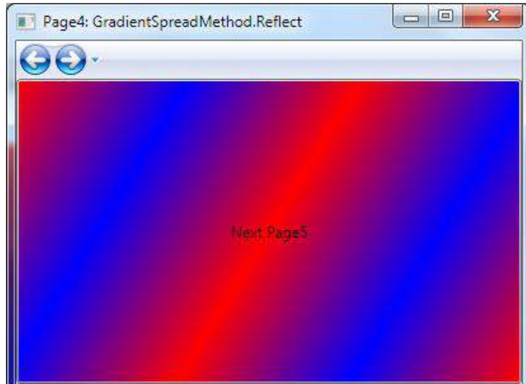
```
namespace WpfApp1
{
  class Page4 : Page
  {
```

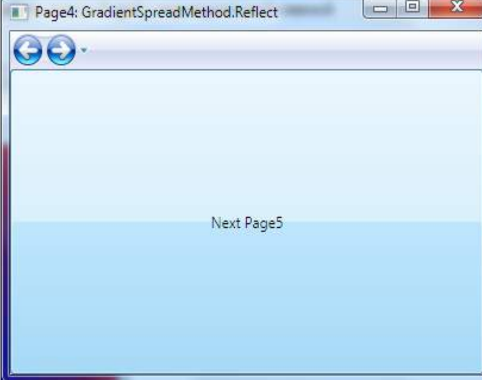
```
  public Page4()
  {
   this.WindowTitle = "Page4: GradientSpreadMethod.Reflect";
   Button btn = new Button();
   btn.Content = "Next Page5";
   btn.Click += new RoutedEventHandler(btn4_Click);
   this.Content = btn;

     // გრადიენტის შექმნა და მიერთება
   LinearGradientBrush brush = new LinearGradientBrush(
           Colors.Red, Colors.Blue, new Point(0, 0), new
                                                    Point(0.25, 0.25));
   brush.SpreadMethod = GradientSpreadMethod.Reflect;
   btn.Background = brush;
  }
   // გადასვლა მეხუთე გვერდზე
     Page5 page5;
    void btn4_Click(object sender, RoutedEventArgs e)
     {
        if (!this.NavigationService.CanGoForward)
          page5 = new Page5();// იქმნება მხოლოდ ერთხელ

               this.NavigationService.Navigate(page5);
     }
   }
}
```

17. Page3 კოდში ჩავსვათ Page4-ის ეგზემპლარის შექმნის კოდი:

```
           // გადასვლა მეოთხე გვერდზე
     Page4 page4;
     void btn3_Click(object sender, RoutedEventArgs e)
     {
        if (!this.NavigationService.CanGoForward)
          page4 = new Page4();// იქმნება მხოლოდ ერთხელ

                 this.NavigationService.Navigate(page4);
     }
```

18. ავამუშავოთ აპლიკაცია და გამოვცადოთ Page4 კლასის მუშაობა წრფივი გრადიენტის მიმართულების პერიოდული ცვლილების დროს (ნახ.15.49).





ნახ.15.49

- **გვერდი_5: საფეხურებრივი შევსება გრადიენტებით**

წრფივ გრადიენტს აქვს GradientStops კოლექცია, რომელიც შეიცავს GradientStop ობიექტებს. თითოეული ობიექტი იძლევა გრადიენტის საწყის ფერს და მარჯვენა საზღვარს ამ ფერით წინა-არსებულის შესავსებად. ამ დროს დამატებით გამოიყენება შევსების სასაზღვრო წერტილები, რომლებიც განისაზღვრება StartPoint და EndPoint თვისებებით. ჩავატაროთ ექსპერიმენტი, შევქმნათ გვერდი, დავდოთ ღილაკი და გავაფერადოთ ეს ღილაკი ვერტიკალური გრადიენტებით.

19. შევქმნათ Page5 გვერდი CreateBrushes.cs ფაილში. დავდოთ ღილაკი. ჩავწეროთ კოდი:

```
namespace WpfApp1
{  class Page4 : Page
  {  public Page4()
    {   this.WindowTitle = "Page4: GradientSpreadMethod.Reflect";
        Button btn = new Button();
        btn.Content = "Next Page5";
        btn.Click += new RoutedEventHandler(btn4_Click);
        this.Content = btn;
      // გრადიენტის შექმნა და მიერთება
       LinearGradientBrush brush = new LinearGradientBrush(
        Colors.Red, Colors.Blue, new Point(0, 0), new Point(0.25, 0.25));
        brush.SpreadMethod = GradientSpreadMethod.Reflect;
     btn.Background = brush;
```

```
  }
     // გადასვლა მეხუთე გვერდზე
     Page5 page5;
     void btn4_Click(object sender, RoutedEventArgs e)
     {
        if (!this.NavigationService.CanGoForward)
              page5 = new Page5();// იქმნება მხოლოდ ერთხელ
         this.NavigationService.Navigate(page5);
     }
   }
}
```

20. Page4 კოდში ჩავწეროთ Page5-ის ეგზემპლარის შექმნის კოდი:

```
// გადასვლა მეოთხე გვერდზე
     Page4 page4;
    void btn3_Click(object sender, RoutedEventArgs e)
     {
        if (!this.NavigationService.CanGoForward)
                page4 = new Page4();// იქმნება მხოლოდ ერთხელ
        this.NavigationService.Navigate(page4);
     }
```

21. ავამუშავოთ აპლიკაცია და Page5 გვერდზე დავაკვირდეთ ცისარტყელურ ფერებს (ნახ.2.43–ა,ბ):

ნახ.15.50-ა

ნახ.15.50–ბ

- **გვერდი_6: RadialGradientBrush რადიალური გრადიენტი**

რადიალური გრადიენტი რეალიზდება RadialGradientBrush კლასით. განვიხილოთ მაგალითი.

22. შევქმნათ ახალი Page6 გვერდი CreateBrushes.cs ფაილში და შევაფერადოთ შემდეგი კოდით:

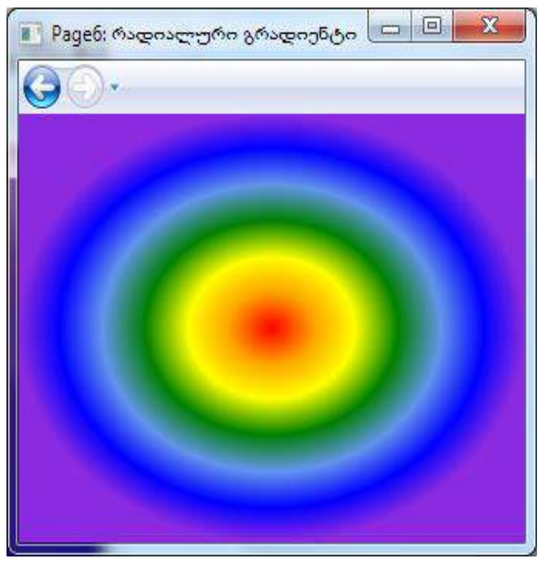
```
namespace WpfApp1
{
 class Page6 : Page
 {
  public Page6()
  {
   this.WindowTitle = "Page6: რადიალური გრადიენტი";

   // გრადიენტის შექმნა და მიერთება
   RadialGradientBrush brush = new RadialGradientBrush();
   this.Background = brush;

   // ცისარტყელას ფერები
brush.GradientStops.Add(new GradientStop(Colors.Red, 0));
brush.GradientStops.Add(new GradientStop(Colors.Orange, .17));
brush.GradientStops.Add(new GradientStop(Colors.Yellow, .33));
brush.GradientStops.Add(new GradientStop(Colors.Green, .5));
brush.GradientStops.Add(new  GradientStop
                                    (Colors.CornflowerBlue, .67));
brush.GradientStops.Add(new GradientStop(Colors.Blue, .84));
brush.GradientStops.Add(new GradientStop(Colors.BlueViolet, 1));
   }
  }
}
```

23. Page5 კოდში ჩავწეროთ Page6-ის ეგზემპლარის შექმნის კოდი:

```
// გადასვლა მეექვსე გვერდზე
    Page6 page6;
    void btn5_Click(object sender, RoutedEventArgs e)
    {
```

```
    if (!this.NavigationService.CanGoForward)
      page6 = new Page6();// იქმნება მხოლოდ ერთხელ
    this.NavigationService.Navigate(page6);
  }
```

24. ავამუშავოთ აპლიკაცია და დავაკვირდეთ რადიალური გრადიენტის ფუნქცის მუშაობას.



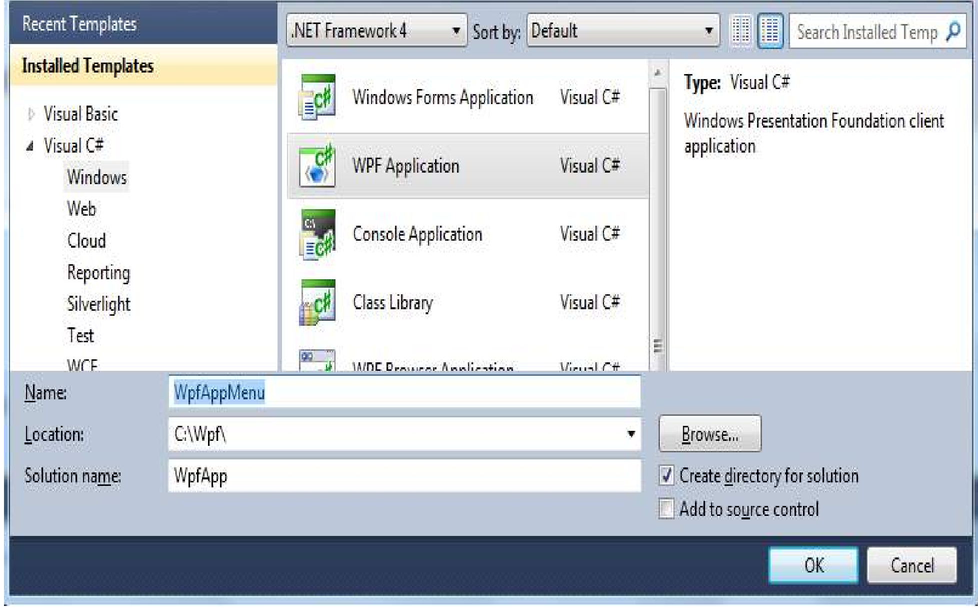
ნახ.15.51

15.8. WPF-ის მართვის ელემენტები: Menu, ToolBar, TabControl და ToolTip

განვიილოთ WPF-აპლიკაციების ასაგებად Menu, ToolBar, TabControl და ToolTip მართვის ელემენტების გამოყენების საკითხი. მომხმარებელთა ინტერფეისების აგება WPF-ის ვიზუალური მართვის ელემენტებით შესაძლებელია მაიკროსოფთის ბიბლიოთეკის ელექტრონული ცნობარიდან:

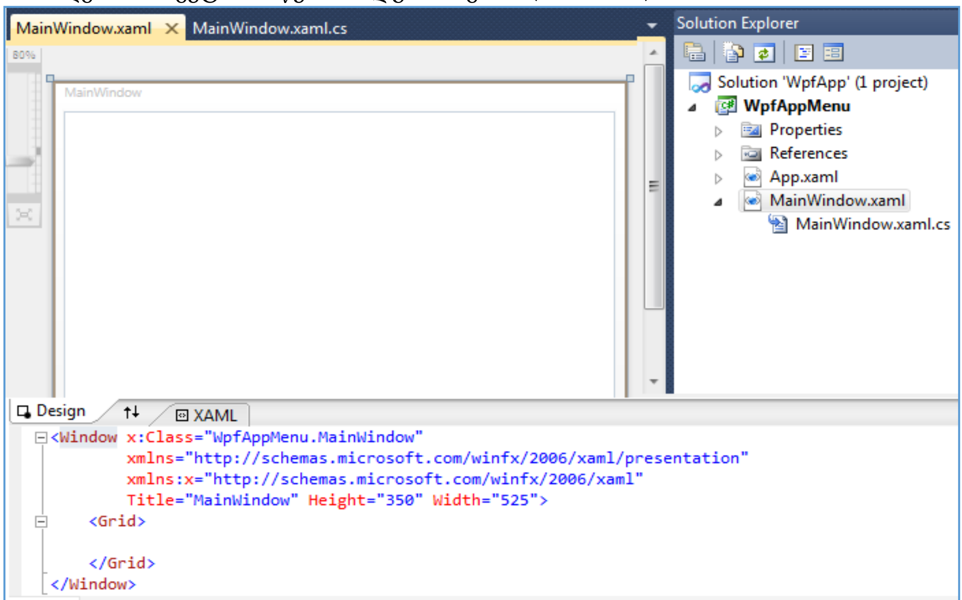
http://msdn.microsoft.com/en-en/library/ms752324.aspx

1. შევქმნათ ახალი პროექტი Visual Studio-ს Solution Explorer-ში WpfAppMenu სახელით (ნახ.15.52) :



ნახ.15.52

მიიღება პროექტის საწყისი მდგომარეობა (ნახ.15.53):



ნახ.15.53

2. მომხმარებლის ინტერფეისის ფანჯრის დიზაინისთვის MainWindow.xaml კოდში ჩავწეროთ შემდეგი ტექსტი:

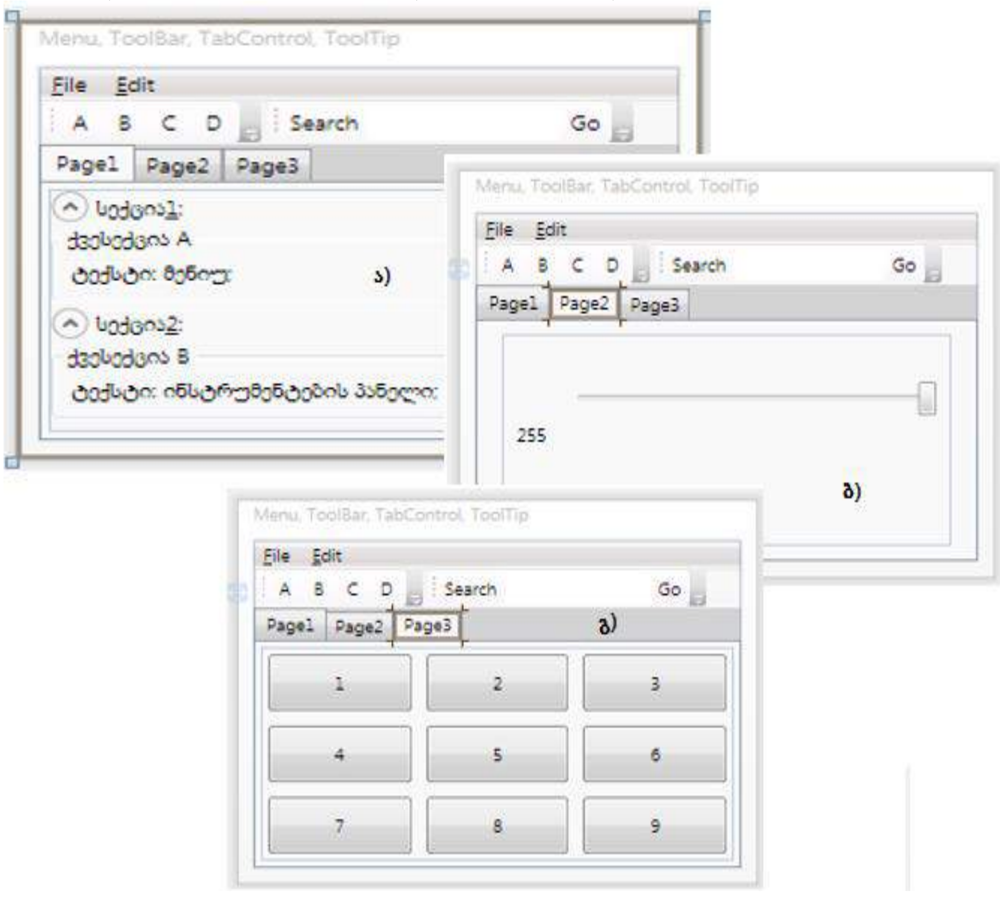
```
<Window x:Class="WpfApp2.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Menu, ToolBar, TabControl, ToolTip"
    SizeToContent="WidthAndHeight"

    ToolTipService.InitialShowDelay="0"
    ToolTipService.ShowDuration="500000" mc:Ignorable="d"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
d:DesignHeight="270">

    <Window.ToolTip>
        <ToolTip x:Name="toolTip"
            Placement="RelativePoint"
            VerticalOffset="10"
            />
    </Window.ToolTip>

    <DockPanel Background="LightGray">
        <Menu DockPanel.Dock="Top">
            <MenuItem Header="_File">
             <MenuItem Header="E_xit" Click="ExitClicked" />
            </MenuItem>
            <MenuItem Header="_Edit">
                <MenuItem Header="_Cut" />
                <MenuItem Header="C_opy" />
                <MenuItem Header="_Paste" />
            </MenuItem>
        </Menu>

        <ToolBarTray DockPanel.Dock="Top">
            <ToolBar>
                <Button Width="23">A</Button>
                <Button Width="23">B</Button>
                <Button Width="23">C</Button>
                <Button Width="23">D</Button>
            </ToolBar>
            <ToolBar Header="Search">
```

```
        <TextBox Width="100" />
        <Button Width="23">Go</Button>
    </ToolBar>
</ToolBarTray>

<TabControl>
    <TabItem Header="Page1">
        <StackPanel>
          <Expander Header="სექცია_1:"
                                                IsExpanded="True">
            <GroupBox Header="ქვესექცია A">
                <Label>ტექსტი: მენიუ; </Label>
            </GroupBox>
           </Expander>
         <Expander Header="სექცია_2:"
                                                IsExpanded="True">
            <GroupBox Header="ქვესექცია B">
      <Label>ტექსტი: ინსტრუმენტების პანელი;</Label>
            </GroupBox>
         </Expander>

        </StackPanel>
    </TabItem>
    <TabItem Header="Page2">
      <StackPanel Orientation="Horizontal"
                                          Margin="5" Width="297">
       <TextBlock Name="value" Margin="10"
                               Text="255" Width="25" Height="23" />
         <Slider Name="slider" Width="241"
             Minimum="0" Maximum="255" Value="255"
    ValueChanged="slider_ValueChanged" Height="77" />
      </StackPanel>
     </TabItem>
    <TabItem Header="Page3">
        <UniformGrid Rows="3" Columns="3">
                <Button ToolTip="ღილაკი 1"  Margin="5">1</Button>
    <Button ToolTip="ღილაკი 2"  Margin="5">2</Button>
```

```
        <Button ToolTip="ღილაკი 3" Margin="5">3</Button>
              <Button ToolTip="ღილაკი 4" Margin="5">4</Button>
        <Button ToolTip="ღილაკი 5"  Margin="5">5</Button>
        <Button ToolTip="ღილაკი 6" Margin="5">6</Button>
         <Button ToolTip="ღილაკი 7" Margin="5">7</Button>
         <Button ToolTip="ღილაკი 8" Margin="5">8</Button>
         <Button ToolTip="ღილაკი 9" Margin="5">9</Button>
       </UniformGrid>
     </TabItem>
   </TabControl>
  </DockPanel>
</Window>
```

შედეგში მიიღება სამი გვერდი Page1, Page2 და Page3 (ნახ15.54).

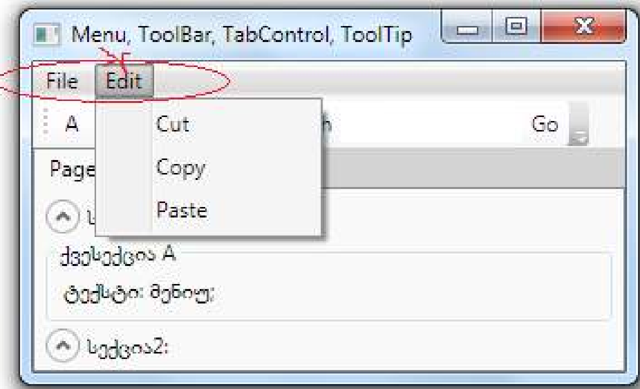


ნახ.15.54. სამი გვერდი: Page1 (ა), Page2 (ბ) და Page3 (გ)

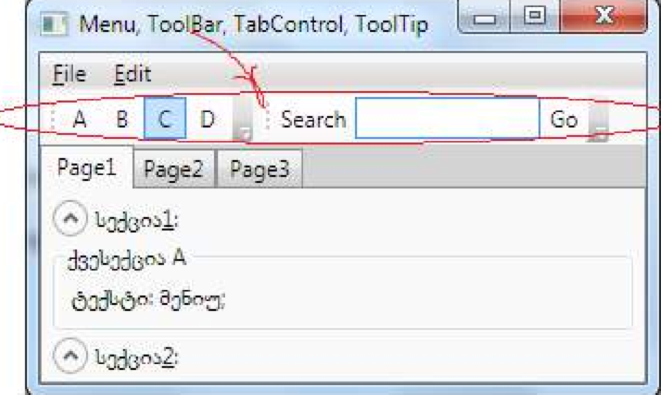
3. MainWindow.xaml.cs კოდისათვის შევიტანოთ შემდეგი ტექსტი:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfAppMenu
{
   public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

   private void ExitClicked(object sender, RoutedEventArgs e)
        {
            this.Close();
        }

   private void slider_ValueChanged(object sender,
                    RoutedPropertyChangedEventArgs<double> e)
        {
            value.Text = ((int)slider.Value).ToString();
        }
    }
}
```

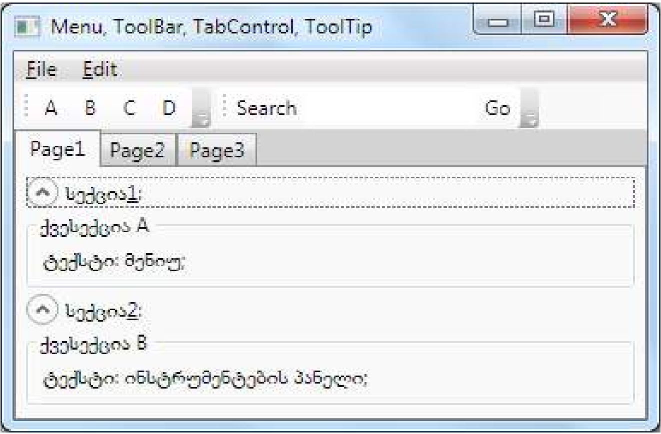
შედეგები ასახულია 15.55 (ა-ე) ნახაზზე.



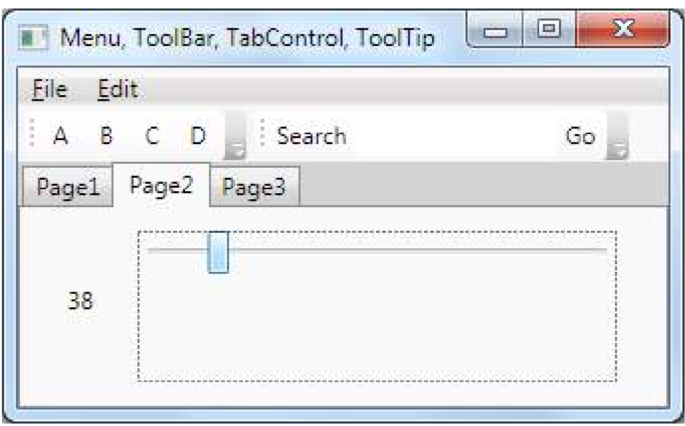
ნახ.15.55–ა



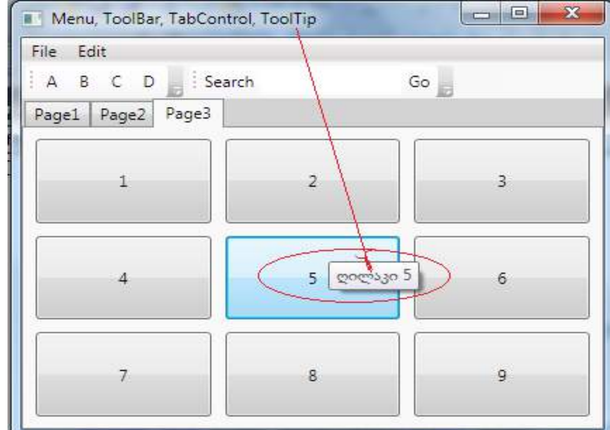
ნახ.15.55–ბ



ნახ.15.55–გ



ნახ.15.55–დ



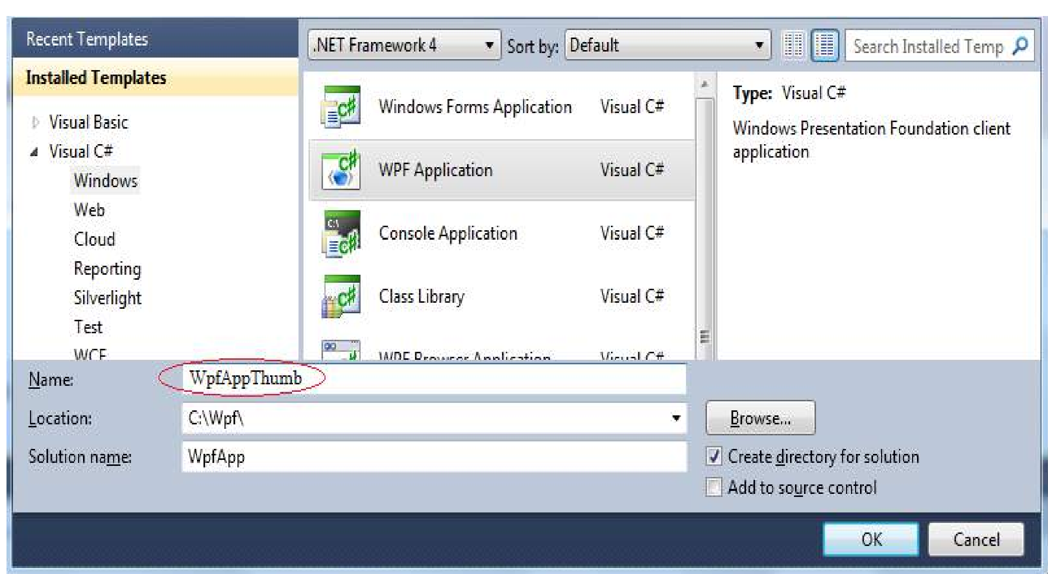
ნახ.15.55–ე

15.9. WPF-ის მართვის ელემენტები: Thumb, Border, Popup

განვიხილოთ WPF-აპლიკაციების ასაგებად Thumb, Border და Popup მართვის ელემენტების გამოყენების საკითხი.

მართვის ელემენტი **Thumb** (მანიპულატორი) აინკაპსულირებს არეს, რომლის გადაადგილებაც შესაძლებელია, იგი ასევე აგენერირებს ყველა აუცილებელ მოვლენას.

1. შევქმნათ ახალი პროექტი WpfAppThumb და მივანიჭოთ სასტარტო მნიშვნელობა.



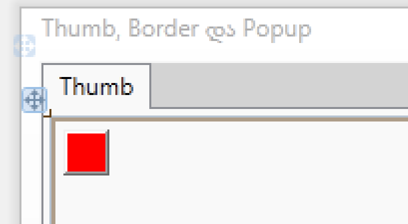
ნახ.15.56

2. Window1.xaml ფაილში ჩავწეროთ შემდეგი ტექსტი:

```
<Window x:Class="WpfAppThumb.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="Thumb, Border და Popup"
       Height="350"
       Width="525"
       Background="LightGray"
    >
    <TabControl>
       <!--ელემენტი, რომელიც აღწერს  გადაადგილების არეს-->
       <TabItem Header="Thumb">
          <Canvas>
             <Thumb
                   Name="thumb1"
                   Background="Red"
                   Canvas.Left="6"
                   Canvas.Top="6"
                   Width="23"
                   Height="23"
                   DragStarted="thumb1_DragStarted"
```

```
                DragDelta="thumb1_DragDelta"
                />
        </Canvas>
    </TabItem>
    <!-- მომავალი ტექსტების ჩასამატებელი ადგილი -------------------
  </TabControl>
</Window>
```

მივიღებთ შემდეგ ფრაგმენტს წითელი კვადრატით, რომლის გადაადგილებაც შეიძლება მაუსით:



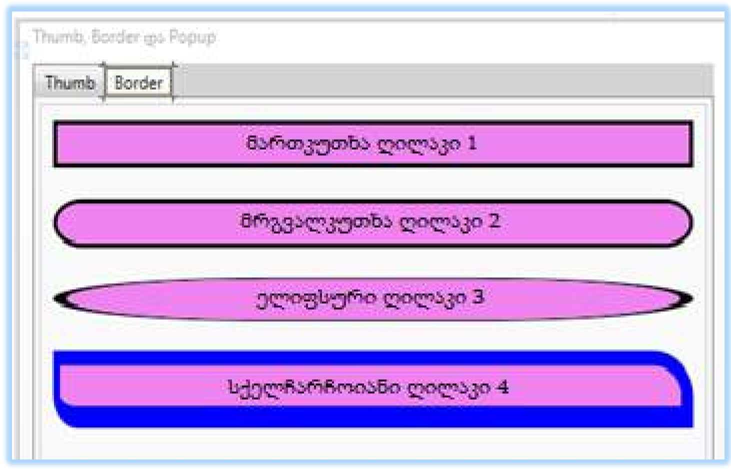
ნახ.15.57

3. Window1.xaml ფაილში ჩავამატოთ ტექსტი:

```
  <!-- Border ჩარჩოების გამოყენება -->
  <TabItem Header="Border">
    <StackPanel>
      <Border
            BorderThickness="3"
            CornerRadius="0"
            BorderBrush="Black"
            Padding="5"
            Margin="10"
            Background="Violet"
            >
        <TextBlock HorizontalAlignment="Center">
            მართკუთხა ღილაკი-1
        </TextBlock>
      </Border>
      <Border
            BorderThickness="3"
            CornerRadius="20"
            BorderBrush="Black"
```

```
                Padding="5"
                Margin="10"
                Background="Violet"
                >
            <TextBlock HorizontalAlignment="Center">
                მრგვალკუთხა ღილაკი-2
            </TextBlock>
        </Border>
        <Border
                BorderThickness="10,1,10,1"
                CornerRadius="150"
                BorderBrush="Black"
                Padding="5"
                Margin="10"
                Background="Violet"
                >
            <TextBlock HorizontalAlignment="Center">
                ელიფსური ღილაკი-3
            </TextBlock>
        </Border>
        <Border
                BorderThickness="5,10,9,15"
                CornerRadius="0,25,0,15"
                BorderBrush="Blue"
                Padding="5"
                Margin="10"
                Background="Violet"
                >
            <TextBlock HorizontalAlignment="Center">
                სქელჩარჩოიანი ღილაკი-4
            </TextBlock>
        </Border>
    </StackPanel>
</TabItem>
<!-- მომავალი ტექსტების ჩასამატებელი ადგილი -----------------
```

მივიღებთ:



ნახ.15.58

4. Window1.xaml ფაილში ჩავამატოთ ახალი ტექსტი Popup ელემენტით.

```
<!-- Popup ელემენტის გამოყენება -->
   <TabItem Header="Popup">
     <StackPanel>
      <Button Name="btn" Click="btn_Click">Toggle</Button>
           <Popup Name="popup"
              PopupAnimation="Fade"
              Placement="Mouse"
             >
        <Button Background="Yellow" Foreground="Red">
                   ღილაკი შექმნილია Popup-ში
        </Button>
      </Popup>
     </StackPanel>
   </TabItem>
```

და ამავდროულად შევასრულოთ პ.5:

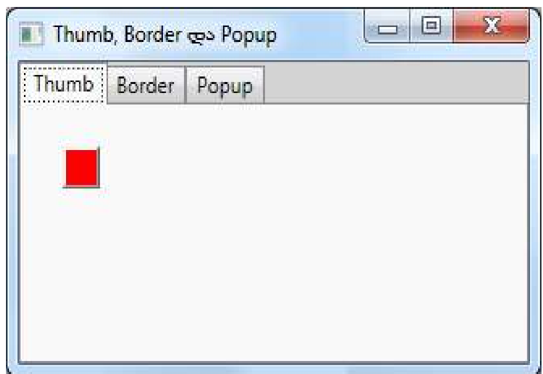
5. ჩავწეროთ Window1.xaml.cs ფაილში შემდეგი C# ტექსტი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
```

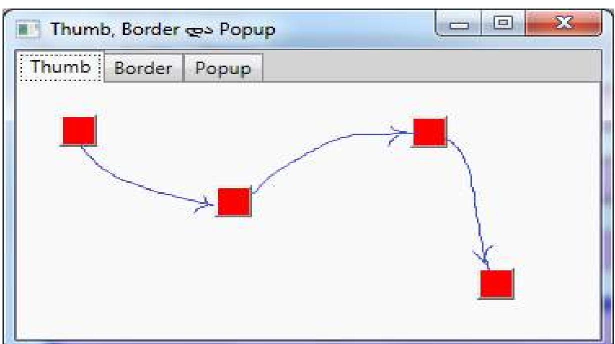
```
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Controls.Primitives;  // ეს დაამატეთ !

namespace WpfAppThumb
{
  public partial class Window1 : Window
   {  public Window1()
      {    InitializeComponent();   }
     protected override void OnMouseDoubleClick(MouseButtonEventArgs e)
      {   base.OnMouseDoubleClick(e);
          // Thumb ბრუნდება საწყის მდგომარეობაში
          Canvas.SetLeft(thumb1, 5);
          Canvas.SetTop(thumb1, 5);
      }
      double originalLeft, originalTop;
    private void thumb1_DragStarted(object sender, DragStartedEventArgs e)
      {
          originalLeft = Canvas.GetLeft(thumb1);
          originalTop = Canvas.GetTop(thumb1);
      }
    private void thumb1_DragDelta(object sender, DragDeltaEventArgs e)
      {   double left = originalLeft + e.HorizontalChange;
          double top = originalTop + e.VerticalChange;
          Canvas.SetLeft(thumb1, left);
          Canvas.SetTop(thumb1, top);
          originalLeft = left;
          originalTop = top;
      }
    private void btn_Click(object sender, RoutedEventArgs e)
      {  popup.IsOpen = !popup.IsOpen;    }
   }
```

}

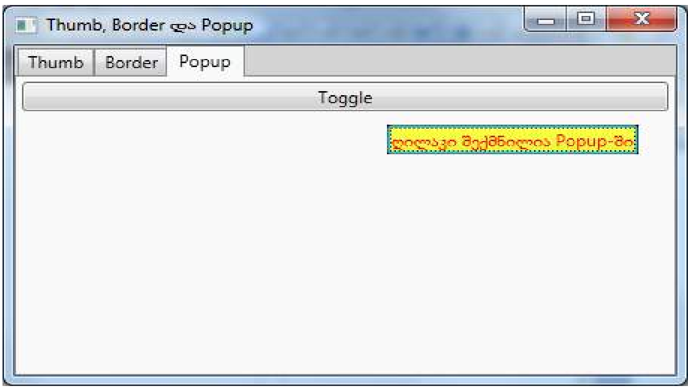
6. ავამუშავოთ აპლიკაცია და დავაკვირდეთ Thumb, Border, Popup ელემენტების მუშაობას:



ნახ.15.59–ა, საწყისი მდებარეობა



ნახ.15.59–ბ. მაუსით გადატანა



ნახ.15.59–გ. ბუტონის დაკლიკვისას ამოტივტივდება დამხმარე ტექსტი

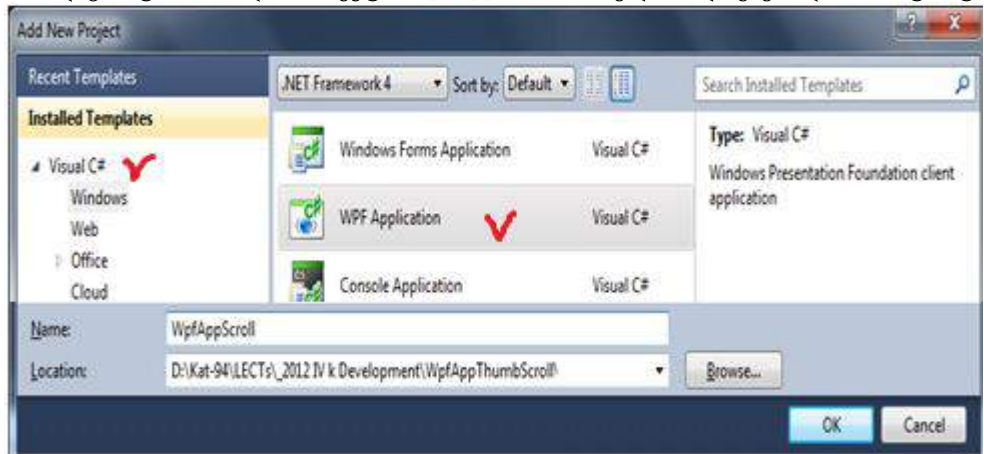
15.10. WPF-ის მართვის ელემენტები: ScrollViewer, Viewbox და StackPanel

განვიხილოთ WPF-აპლიკაციების ასაგებად ScrollViewer, Viewbox და StackPanel მართვის ელემენტების გამოყენების საკითხი.

ScrollViewer ელემენტით ხდება ფანჯრის სივრცის გადახვევა, თუ შვილობილ-ელემენტიანი ინტერფეისი მასში ვერ ეტევა.

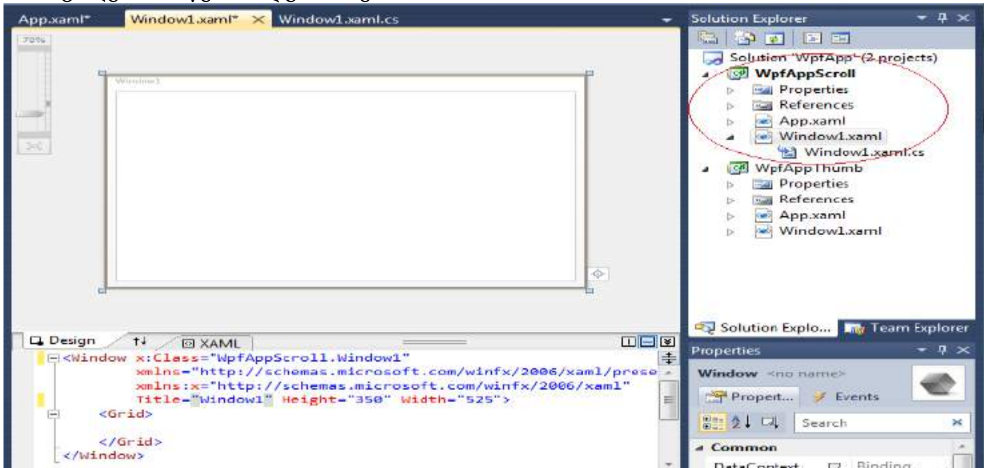
Viewbox ელემენტი იღებს მაგალითად, კონტეინერს და ახდენს მის მასშტაბირებას თავისი ზომების პროპორციულად.

1. დავამატოთ ახალი პროექტი WpfAppScroll სახელით და გავხადოთ სასტარტო.



ნახ.15.60

მივიღებთ საწყის მდგომარეობას:



ნახ.15.61

2. ჩავწეროთ Window1.xaml კოდში შემდეგი ტექსტი:

```
<Window x:Class="WpfAppScroll.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="ScrollViewer, Viewbox და  StackPanel "
        Height="350"
        Width="525"
        Background="LightGray">
    <TabControl>
        <TabItem Header="ScrollViewer">
            <ScrollViewer
                HorizontalScrollBarVisibility="Auto"
                VerticalScrollBarVisibility="Auto"
                >
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto" />
                        <RowDefinition Height="Auto" />
                        <RowDefinition />
                  </Grid.RowDefinitions>
                  <TextBox Name="textBox" Grid.Row="0" Margin="5" />
                     <Button Grid.Row="1" Margin="5"
                                 Click="Add_Click">Add</Button>
                  <ListBox Name="listBox" Grid.Row="2" Margin="5" />
                </Grid>
              </ScrollViewer>
            </TabItem>
             <!-------------- ჩასამატებელი ადგილი --------------- ->
         </TabControl>
</Window>
```

მივიღებთ შედეგს:

ნახ.15.63

3. დავამატოთ ახალი ტექსტი `Viewbox`-ისათვის:

```
<TabItem Header="Viewbox">
        <Viewbox>
          <TextBlock>Текст</TextBlock>
        </Viewbox>
      </TabItem>
   </TabControl>
</Window>
```

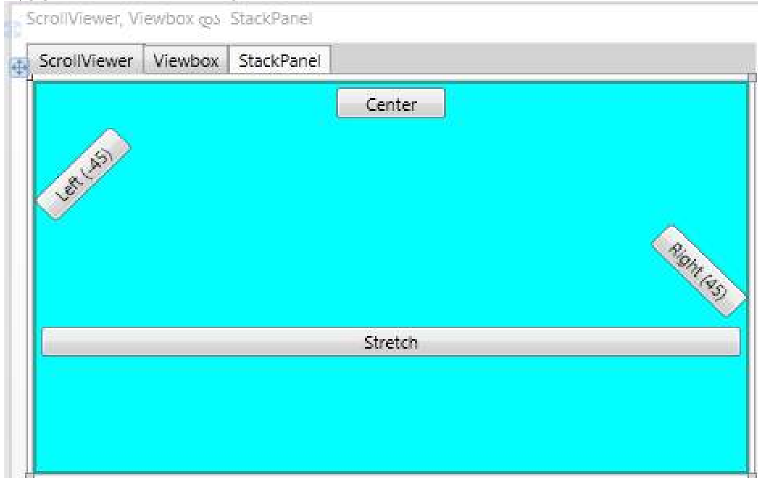
მივიღებთ შედეგს (ნახ.15.62).

ნახ.15.63

4. დავამატოთ ტექსტი StackPanel -ისათვის:

```
  <TabItem Header="StackPanel">
<StackPanel Orientation="Vertical" Background="Aqua">
     <Button HorizontalAlignment="Center" Width="75"
                                              Margin="5"> Center
   </Button>
   <Button HorizontalAlignment="Left" Width="75">
     <Button.LayoutTransform>
       <RotateTransform Angle="-45" />
     </Button.LayoutTransform> Left (-45)
  </Button>
  <Button HorizontalAlignment="Right" Width="75">
      <Button.LayoutTransform>
       <RotateTransform Angle="45" />
     </Button.LayoutTransform> Right (45)
  </Button>
  <Button HorizontalAlignment="Stretch" Margin="5">Stretch</Button>
</StackPanel>
</TabItem>
```

მივიღებთ შედეგს 15.64 ნახაზზე:



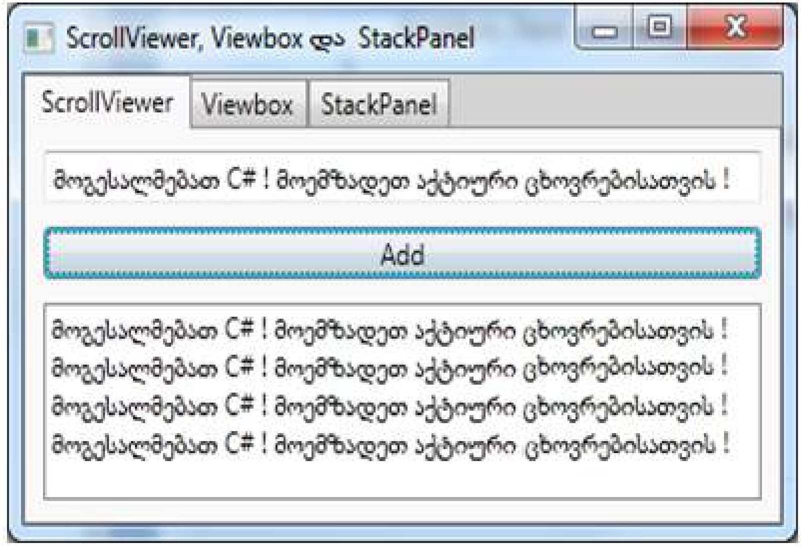
ნახ.15.64

5. შევავსოთ C# კოდი შემდეგი ტექსტით:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfAppScroll
{
   public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
          textBox.Text = "მოგესალმებათ C# ! მოემზადეთ
                  აქტიური ცხოვრებისთვის !";            }
```

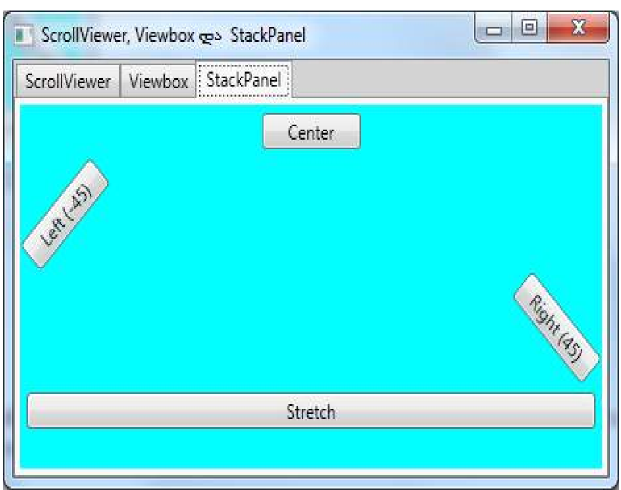
```
    private void Add_Click(object sender, RoutedEventArgs e)
        {
            listBox.Items.Add(textBox.Text);
        }

    }
}
```

ავამუშავოთ აპლიკაცია, მივიღებთ (ნახ.15.65 ა-დ):

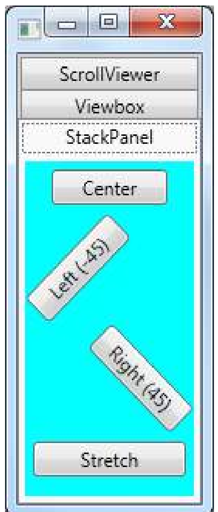


ნახ.15.65-ა

ნახ.15.65-ბ



ნახ.15.65-გ



ნახ.15.65-დ

მასშტაბის შეცვლით ელემენტების განთავსებაც შესაბამისად იცვლება.

15.11. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: Canvas

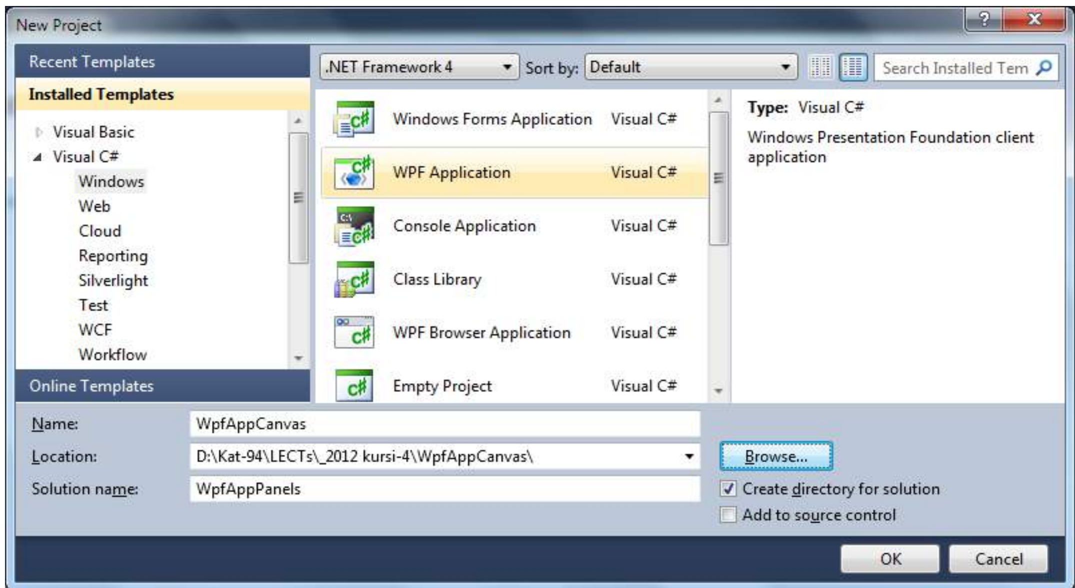
პროგრამულ აპლიკაციებში ინტერფეისული ელემენტების დინამიკური განლაგებისთვის ხშირად გამოიყენება შემდეგი პანელები: Canvas, StackPanel, DockPanel, UniformGrid და Grid.

პანელები უზრუნველყოფს ფანჯარაში ან გვერდზე ელემენტების მოთავსებას სასურველ პოზიციებში. ყველა ზემონახსენები პანელი მემკვიდრეა Panel – აბსტრაქტული კლასის, რომელიც თავის მხრივ იწარმოება საბაზო კლასიდან.

System.Windows.FrameworkElement

თვით პანელები ეკრანზე არ ჩანს, თუ ცხადად არ მიეთითა პანელის ფერადი ფონი. მათი არსებობა იგრძნობა პანელზე მოთავსებული შვილობილი ელემენტების ქცევით.

1. გავხსნათ ახალი პროექტი name: WpfAppCanvas, რომლის Solution name: WpfAppPanels (შემდგომში მას დაემატება სხვა პანელების საილუსტრაციო პროექტები).



ნახ.15.66

2. MainWindow.xaml კოდში ჩავწეროთ შემდეგი ტექსტი:

```
<Window x:Class="WpfAppCanvas.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="განლაგების მენეჯერი – Canvas" Height="300"  Width="300"
                        Background="LightGray">
   <TabControl>
     <TabItem Header="Canvas 1">
      <Canvas Width="200" Height="100" Background="Aqua">
    <Button Canvas.Left="2" Canvas.Top="2">Left, Top </Button>
  <Button Canvas.Right="2" Canvas.Top="2">Right, Top</Button>    <Button Canvas.Left="2"
Canvas.Bottom="2">Left, Bottom</Button>
<Button Canvas.Right="2" Canvas.Bottom="2">Right, Bottom</Button>
      </Canvas>
   </TabItem>
```

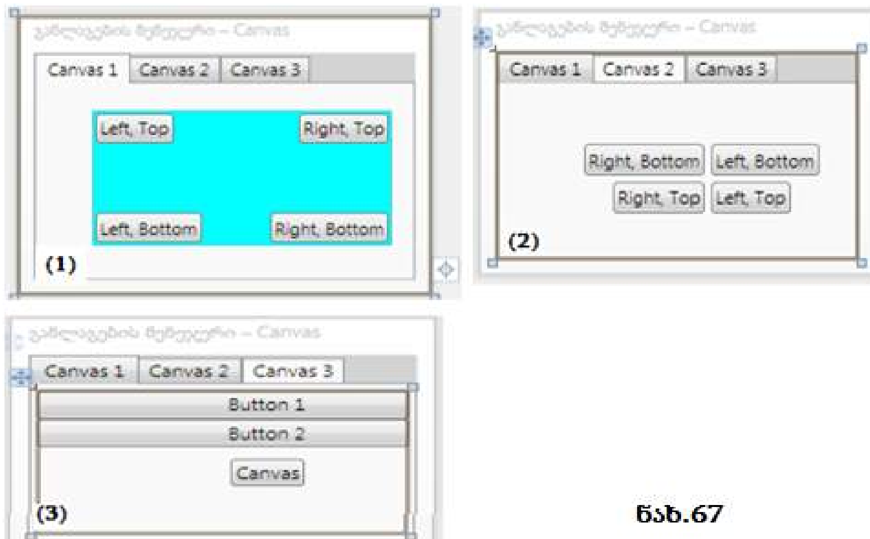
```
    <TabItem Header="Canvas 2">
   <Canvas HorizontalAlignment="Center" VerticalAlignment="Center" Background="Aqua">
 <Button Canvas.Left="2" Canvas.Top="2">Left, Top</Button>
 <Button Canvas.Right="2" Canvas.Top="2">Right, Top</Button>
<Button Canvas.Left="2" Canvas.Bottom="2">Left, Bottom</Button>
<Button Canvas.Right="2" Canvas.Bottom="2">Right, Bottom</Button>
   </Canvas>
  </TabItem>
    <TabItem Header="Canvas 3">
      <StackPanel>
        <Button>Button 1</Button>
        <Button>Button 2</Button>
        <Canvas HorizontalAlignment="Center"  Background="Aqua">
  <Button Canvas.Left="-21" Canvas.Top="8">Canvas</Button>
       </Canvas>
      </StackPanel>
   </TabItem>
  </TabControl>
</Window>
```

3. C# კოდი ავტომატურად იქმნება და მას დამატებითი ცვლილებები ამჯერად არ სჭირდება.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfAppCanvas
{
```

```
  public partial class MainWindow : Window
   {
     public MainWindow()
       {
          InitializeComponent();
       }
   }
}
```

4. შედეგი მიიღება სამი Canvas გვერდით, რომლებშიც ღილაკების განლაგება განსხვავებულია (ნახ.15.67).



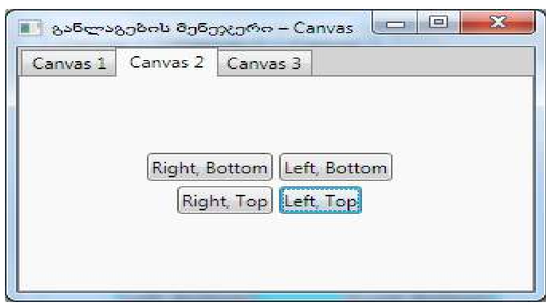
ნახ.67

(1) Canvas–1: აქ განლაგების მენეჯერის სიგანე მოცემულია ცხადად, პანელის ხილვადობისათვის მოცემულია ფონი. ღილაკების განლაგებისათვის პანელის კუთხეებში გამოიყენება სპეციალური თვისებები – ატრიბუტები: Canvas.Left, Canvas.Right, Canvas.Top და Canvas.Bottom.

```
   <TabItem Header="Canvas 1">
     <Canvas Width="200" Height="100" Background="Aqua">
          <Button Canvas.Left="2" Canvas.Top="2">Left, Top</Button>
          <Button Canvas.Right="2" Canvas.Top="2">Right, Top</Button>
          <Button Canvas.Left="2" Canvas.Bottom="2">Left, Bottom</Button>
          <Button Canvas.Right="2" Canvas.Bottom="2">Right, Bottom</Button>
     </Canvas>
   </TabItem>
```

(2) Canvas–2: აქ Canvas პანელის ზომები არაა ცხადად მოცემული, ამიტომაც პანელი შემცირდება მინიმუმამდე და ღილაკები ერთმანეთს მიედება, პანელი არ ჩანს. პანელისთვის აქ მოცემულია თვისებები: HorizontalAlignment და VerticalAlignment, რომელიც ასრულებს პანელის პოზიციონირებას კლიენტის სამუშაო გარემოს მიხედვით. ფანჯრის ზომების შეცვლისას პანელი ღილაკებით ინარჩუნებს ცენტრში ყოფნას.

```
<TabItem Header="Canvas 2">
  <Canvas HorizontalAlignment="Center"
          VerticalAlignment="Center" Background="Aqua">
    <Button Canvas.Left="2" Canvas.Top="2">Left,Top </Button>
    <Button Canvas.Right="2" Canvas.Top="2">Right,Top </Button>
    <Button Canvas.Left="2" Canvas.Bottom="2">Left,Bottom </Button>
    <Button Canvas.Right="2" Canvas.Bottom="2">Right,Bottom </Button>
  </Canvas>
</TabItem>
```



ნახ.15.68

(3) Canvas–3: ამ შემთხვევაში StackPanel პანელი ახორციელებს ღილაკების განლაგებას ვერტიკალში (ნახ.15.69). ბოლო ელემენტია წერტილოვანი Canvas-პანელი, პოზიციონირდება ცენტრში თვისებით HorizontalAlignment. Canvas პანელს მისი ღილაკი მიებმება x ღერძზე უარყოფითი წანაცვლებით.

```
<TabItem Header="Canvas 3">
    <StackPanel>
        <Button>Button 1</Button>
        <Button>Button 2</Button>
          <Canvas HorizontalAlignment="Center" Background="Aqua">
                <Button Canvas.Left="-21"
Canvas.Top="8">Canvas</Button>
          </Canvas>
      </StackPanel>
    </TabItem>
```

ნახ.15.69

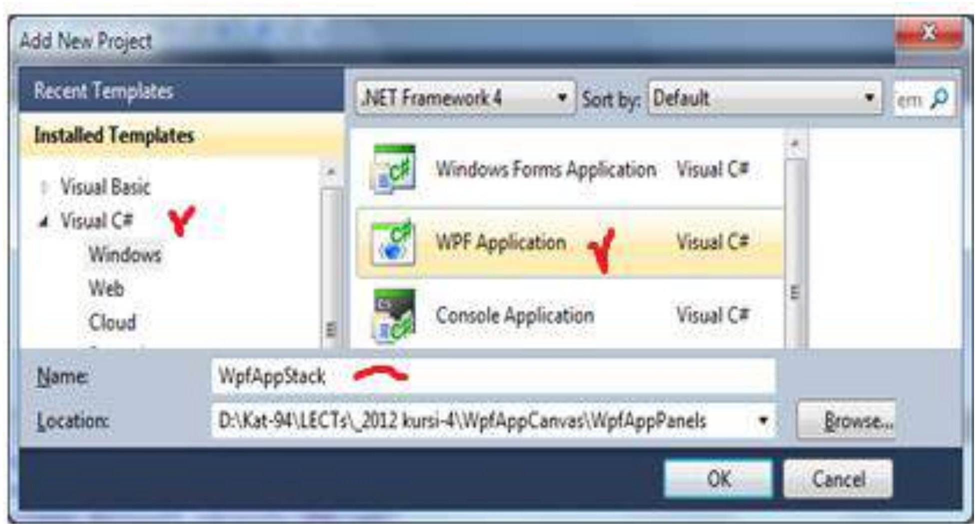
15.12. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: StackPanel და DockPanel

განვიხილოთ WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების მენეჯერის StackPanel და DockPanel ფუნქციურობის გამოყენების საკითხი.

StackPanel განათავსებს ფანჯარაზე მოთავსებულ ელემენტებს ვერტიკალში სტრიქონების სახით. Orientation თვისებით შეიძლება განლაგება ვერტიკალურად ან ჰორიზონტალურად განხორციელდეს. გამოუცხადებლად, StackPanel იკავებს კლიენტის სამუშაო არეს (მაგ., ფანჯარას) მთლიანად. მის ყოველ შვილობილ ელემენტს გამოუყოფს სლოტს (სასიცოცხლო არე), ხოლო თავის ზომას გაითვლის შვილობილებიანი მაქსიმალურის მიხედვით.

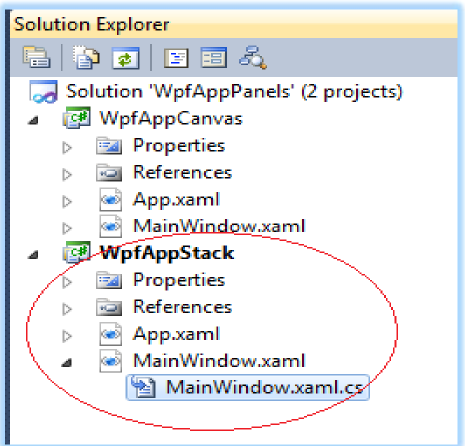
გამოყოფილი სლოტის მიხედვით შვილობილ ელემენტს შეუძლია პოზიციონირება თავისი შეხედულებისამებრ, Margin თვისების გამოყენებით. HorizontalAlignment და VerticalAlignment სლოტის შიგა ადგილი. ელემენტებსშორისი მანძილი მოიცემა Padding თვისებით.

1. დავამატოთ ახალი WpfAppStack პროექტი ძველ Solution WpfAppPanels -ში და გავხადოთ იგი სასტარტო.



ნახ.15.70

მივიღებთ შედეგს (ნახ.15.71).



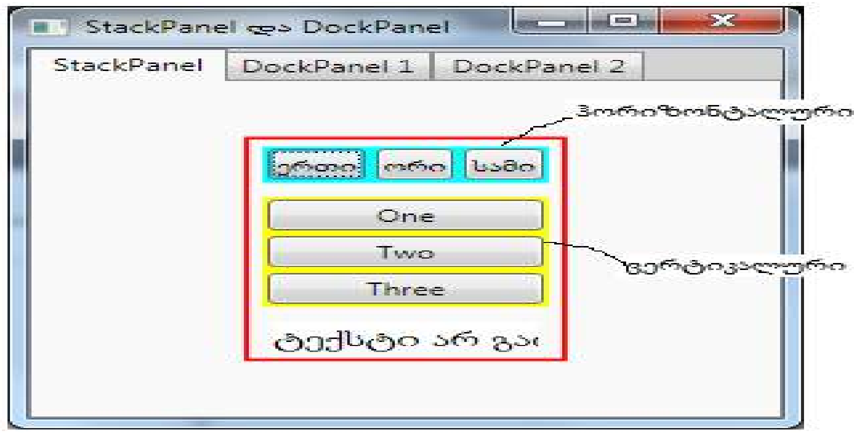
ნახ.15.71

2. დავამატოთ MainWindow.xaml კოდში შემდეგი ტექსტი (ჯერ StackPanel–ის გამოყენებისათვის):

```
<Window x:Class="WpfAppStack.MainWindow"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="StackPanel და DockPanel" Height="300" Width="300"
    Background="LightGray"  >
    <!-- StackPanel–ის გამოყენება -->
    <TabControl>
      <TabItem Header="StackPanel">
        <Border BorderBrush="Red" BorderThickness="2" HorizontalAlignment="Center"
                             VerticalAlignment="Center">
         <StackPanel Orientation="Vertical">
         <StackPanel Orientation="Horizontal" Margin="5"  Background="Aqua">
             <Button Margin="2">ერთი</Button>
             <Button Margin="2">ორი</Button>
             <Button Margin="2">სამი</Button>
         </StackPanel>
         <StackPanel Orientation="Vertical" Margin="5" Background="Yellow">
             <Button Margin="2">One</Button>
             <Button Margin="2">Two</Button>
             <Button Margin="2">Three</Button>
         </StackPanel>
         <StackPanel Orientation="Horizontal" Margin="5" Width="100"
                                                              Background="White">
```

```
            <TextBlock FontSize="12pt"
                  TextWrapping="Wrap"> ტექსტი არ გადაიტანება,
                        რადგან ის ჩატვირთულია StackPanel–ში
            </TextBlock>
          </StackPanel>
         </StackPanel>
        </Border>
      </TabItem>
     </TabControl>
</Window>
```

3. ავამუშავოთ პროგრამა და მივიღებთ შედეგს:



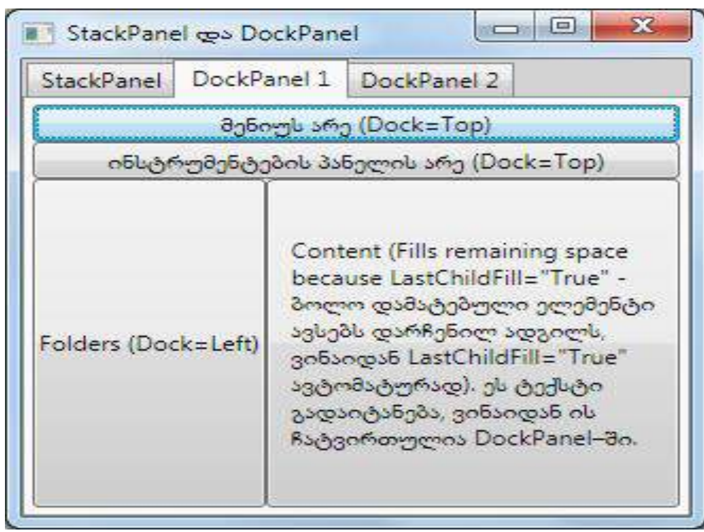
ნახ.15.72

4. ახლა დავამატოთ MainWindow.xaml ფაილში ახალი ტექსტი DockPanel–ის გამოსაყენებლად, რომელიც აისახება „DockPanel 1“ გვერდზე:

```
<!-- Windows Explorer სივრცის იმიტაცია DockPanel–ის გამოყენებით -->
   <TabItem Header="DockPanel 1">
        <DockPanel>
   <Button DockPanel.Dock="Top">მენიუს არე (Dock=Top)</Button>
   <Button DockPanel.Dock="Top">ინსტრუმენტების პანელის არე
              (Dock=Top)</Button>
   <Button DockPanel.Dock="Left">Folders (Dock=Left)</Button>
   <Button>
     <TextBlock TextWrapping="Wrap" Padding="5pt">
```

```
        Content (Fills remaining space because
           LastChildFill="True" - ბოლო დამატებული ელემენტი
           ავსებს დარჩენილ ადგილს, ვინაიდან
                 LastChildFill="True" ავტომატურად).
                 ეს ტექსტი გადაიტანება, ვინაიდან ის
                 ჩატვირთულია DockPanel-ში.
    </TextBlock>
   </Button>
  </DockPanel>
</TabItem>
```

5. პროგრამის ამუშავებით მიიღება ასეთი შედეგი:



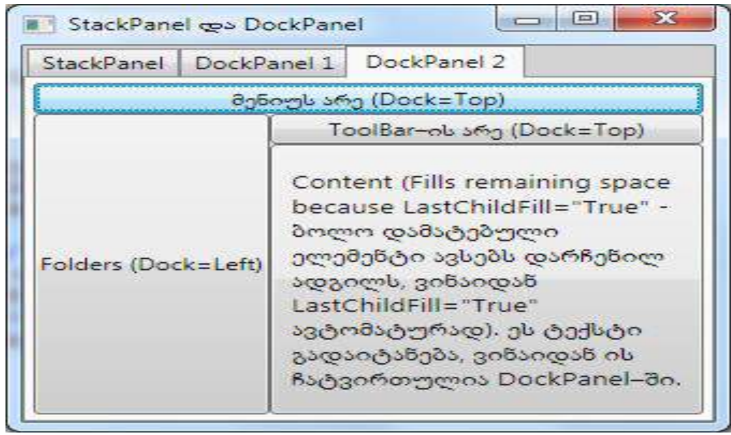
ნახ.15.73

6. MainWindow.xaml ფაილში დავამატოთ ტექსტი DockPanel 2-თვის:

```
<TabItem Header="DockPanel 2">
 <DockPanel LastChildFill="True">
    <Button DockPanel.Dock="Top">მენიუს არე (Dock=Top)</Button>
    <Button DockPanel.Dock="Left">Folders  (Dock=Left)</Button>
    <Button DockPanel.Dock="Top">ToolBar–ის არე (Dock=Top)</Button>
    <Button>
      <TextBlock TextWrapping="Wrap" FontSize="10pt"  Padding="5pt">
         Content (Fills remaining space because
              LastChildFill="True"-ბოლო დამატებული ელემენტი
```

```
            ავსებს დარჩენილ ადგილს, ვინაიდან
            LastChildFill="True" ავტომატურად).
        ეს ტექსტი გადაიტანება, ვინაიდან ის ჩატვირთულია DockPanel–ში.
      </TextBlock>
    </Button>
  </DockPanel>
</TabItem>
```

7. პროგრამის ამუშავების შედეგად მიიღება:



ნახ.15.74

დასკვნა:

1. პირველ გვერდზე იხატება ჩარჩო მოცემული სიგანით და ფერით, რომელშიც თავსდება StackPanel, ჯერ ჰორიზონტალური (ერთი, ორი, სამი), შემდეგ ვერტიკალური ორიენტაციით (One, Two, Three). მათი ფონის ფერები განსხვავებულია. პანელში ქვემოთ მოცემულია ტექსტური ბლოკი, რომელიც სტრიქონს არ გადაიტანს (თუმცა ჩართულია თვისება TextWrapping="Wrap").

2. სტრიქონის გადატანა ტექსტურ ბლოკში შესაძლებელია მაშინ, როცა იგი ჩატვირთულია DockPanel პანელში.

3. DockPanel–ში განლაგების სტრუქტურა დამოკიდებულია ელემენტების დამატების მიმდევრობაზე ამ პანელში. LastChildFill თვისება უფლებას აძლევს ბოლო შვილობილ ელემენტს დაიკავოს მთელი თავისუფალი სივრცე. DockPanel–ის ელემენტებისათვის მოქმედებს მიბმის თვისება, ანუ თავისუფალი ადგილის რომელ მხარეს უნდა მიებას ახალი ელემენტი.

15.13. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: WrapPanel и UniformGrid

განვიხილოთ WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების მენეჯერის WrapPanel და UniformGrid ფუნქციონალობის გამოყენების საკითხი.

პროგრამულ აპლიკაციებში ინტერფეისული ელემენტების დინამიკური განლაგებისთვის ხშირად გამოიყენება UniformGrid პანელი.

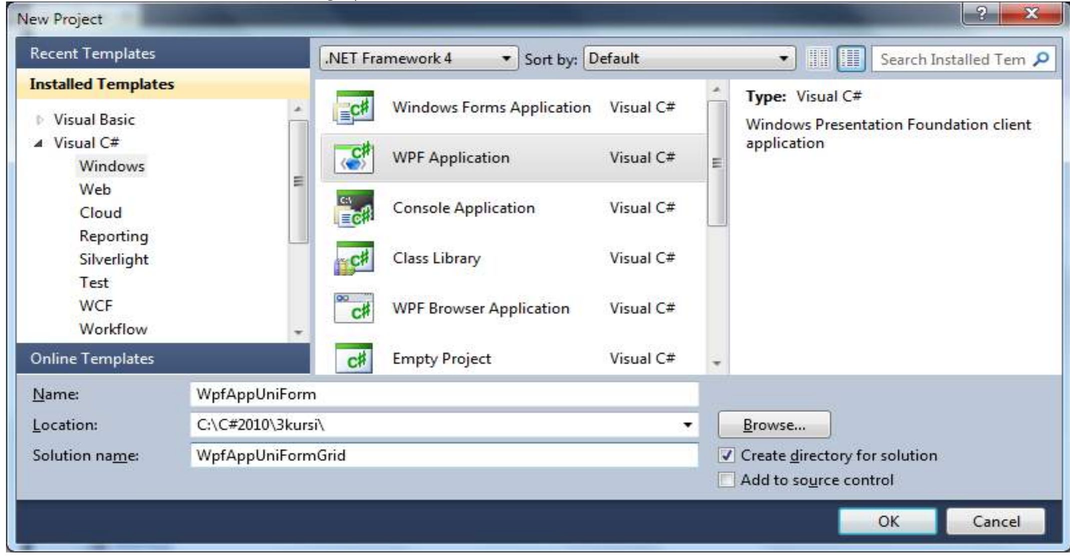
DockPanel კონტეინერში, როგორც ეს წინა ლაბორატორიული სამუშაოდან ვნახეთ, ელემენტები თავსდება (მიეკვრება) ერთ–ერთ თავისუფალ გვერდს მათი ფანჯარაში მოთავსების მიმდევრობის შესაბამისად. WrapPanel კი ელემენტს, რომელიც ვეღარ ეტევა, გადაიტანს ახალ სტრიქონზე.

WrapPanel–ის ვერტიკალური ორიენტაციისას, ელემენტების გადატანა არ ხდება, ოღონდ ელემენტების სიგანე ხდება მათ შორის ყველაზე განიერის. WrapPanel აყენებს თავის „შვილების" ზომებს ავტომატურად, ითვალისწინებს რა მათ შინაარსს. შესაძლებელია ასევე სხვა ზომების მიცემაც ItemWidth და ItemHeight თვისებებით.

UniformGrid პანელი (Uniform–ერთგვაროვანი, ერთნაირი, თანაბარი; Grid – ბადე) ანაწილებს (განათავსებს) „შვილ"-ელემენტებს თანაბარ ბადეში სტრიქონების და სვეტების მითითებული რაოდენობის მიხედვით. შეიძლება მხოლოდ სტრიქონების ან მხოლოდ სვეტების რაოდენობის მითითება, მაშინ მეორე პარამეტრი გაითვლება „შვილების" საერთო რაოდენობიდან.

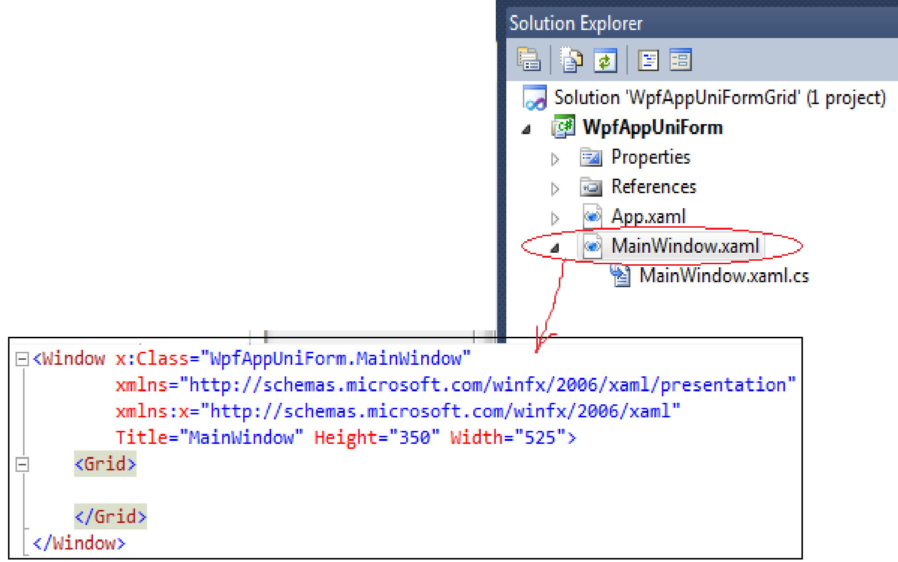
განვიხილოთ პრაქტიკული მაგალითი:

1. შევქმნათ SolutionExplorer-ში name=WpfAppUniFormGrid ახალი პროექტი name=WpfAppUniForm სახელით (ნახ.15.75).



ნახ.15.75

მივიღებთ საწყის მდგომარეობას (ნახ.15.76):



ნახ.15.76

2. MainWindow.xaml ფაილში შევცვალოთ Title:

```
Title="WrapPanel და UniformGrid"
Height="300" Width="300"
Background="LightGray"
```

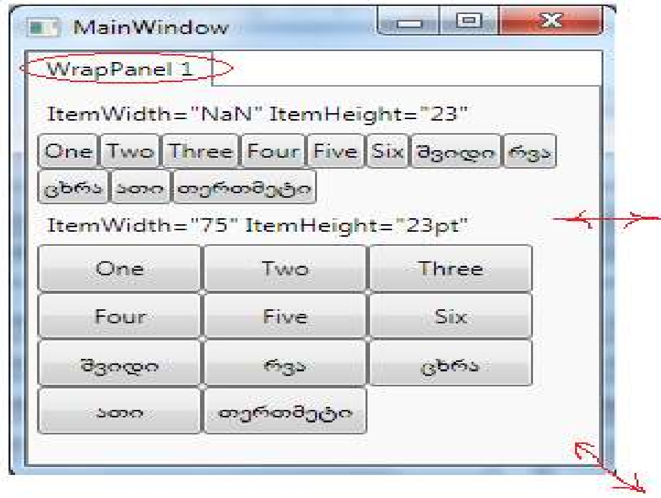
და წავშალოთ <Grid> … </Grid> სტრიქონები.

3. დავამატოთ შემდეგი კოდის ფრაგმენტი:

```
<TabControl>
    <TabItem Header="WrapPanel 1">
      <StackPanel>
        <TextBlock Margin="5">ItemWidth="NaN"
                              ItemHeight="23"</TextBlock>
        <WrapPanel ItemWidth="NaN" ItemHeight="23">
          <Button>One</Button>
          <Button>Two</Button>
          <Button>Three</Button>
          <Button>Four</Button>
          <Button>Five</Button>
          <Button>Six</Button>
          <Button>შვიდი</Button>
          <Button>რვა</Button>
```

```
            <Button>ცხრა</Button>
            <Button>ათი</Button>
            <Button>თერთმეტი</Button>
        </WrapPanel>
        <TextBlock Margin="5">ItemWidth="75"
                                    ItemHeight="23pt"</TextBlock>
        <WrapPanel ItemWidth="75" ItemHeight="23pt">
            <Button>One</Button>
            <Button>Two</Button>
            <Button>Three</Button>
            <Button>Four</Button>
            <Button>Five</Button>
            <Button>Six</Button>
            <Button>შვიდი</Button>
            <Button>რვა</Button>
            <Button>ცხრა</Button>
            <Button>ათი</Button>
            <Button>თერთმეტი</Button>
        </WrapPanel>
      </StackPanel>
    </TabItem>
  </TabControl>
```

მივიღებთ (ნახ.15.77–ა,ბ.):



ნახ.15.77–ა

ნახ.15.77–ბ. ელემენტების განლაგების ცვლილება
ფანჯრის ზომების მიხედვით

4. შევქმნათ WrapPanel 2 გვერდი <TabControl> – ის შიგნით, რომელიც ღილაკებს დაალაგებს ვერტიკალურად. დავამატოთ შემდეგი კოდი:

```
<TabItem Header="WrapPanel 2">
        <StackPanel>
          <TextBlock Margin="5">WrapPanel
                            Orientation="Vertical"</TextBlock>
          <WrapPanel Orientation="Vertical">
            <Button>One</Button>
                    <Button>Two</Button>
            <Button>Three</Button>
            <Button>Four</Button>
            <Button>Five</Button>
            <Button>Six</Button>
            <Button>შვიდი</Button>
            <Button>რვა</Button>
            <Button>ცხრა</Button>
            <Button>ათი</Button>
            <Button>თერთმეტი</Button>
            <Button>თორმეტი</Button>
          </WrapPanel>
        </StackPanel>
      </TabItem>
```

მივიღებთ ვერტიკალში განლაგებულ ბუტონებს (ნახ.15.77-გ):

ნახ.15.77-გ

5. ახლა შევქმნათ UniformGrid გვერდი <TabControl> – ის შიგნით, რომელიც ღილაკებს განალაგებს თანაბრად, სტრიქონების (Rows) ან სვეტების (Columns) მითითებული რაოდენობით. დავამატოთ შემდეგი კოდი:

```
<TabItem Header="UniformGrid">
    <StackPanel>
        <TextBlock Margin="5" Background="White"
                                        Foreground="Red"
          FontWeight="Bold" FontStyle="Italic"
                         TextDecorations="Underline">
          UniformGrid Rows="3" Columns="4"
      </TextBlock>
        <UniformGrid Rows="3" Columns="4">
          <Button>One</Button>
          <Button>Two</Button>
          <Button>Three</Button>
          <Button>Four</Button>
          <Button>Five</Button>
          <Button>Six</Button>
          <Button>შვიდი</Button>
          <Button>რვა</Button>
          <Button>ცხრა</Button>
          <Button>ათი</Button>
          <Button>თერთმეტი</Button>
```

```
        <Button>თორმეტი</Button>
    </UniformGrid>
 <TextBlock Margin="5" Background="Aqua">
        <Bold>
            <Italic>
                <Underline>
                    UniformGrid Rows="2"
                </Underline>
            </Italic>
        </Bold>
 </TextBlock>
    <UniformGrid Rows="2">
        <Button>One</Button>
        <Button>Two</Button>
        <Button>Three</Button>
        <Button>Four</Button>
        <Button>Five</Button>
        <Button>Six</Button>
        <Button>შვიდი</Button>
        <Button>რვა</Button>
        <Button>ცხრა</Button>
        <Button>ათი</Button>
    </UniformGrid>
<TextBlock Margin="5">
    <Run FontWeight="Bold" FontStyle="Italic"
                                            TextDecorations="Underline">
            UniformGrid Columns="4"
        </Run>
</TextBlock>
    <UniformGrid Columns="4">
        <Button>One</Button>
        <Button>Two</Button>
        <Button>Three</Button>
        <Button>Four</Button>
        <Button>Five</Button>
        <Button>Six</Button>
        <Button>შვიდი</Button>
```

```
            <Button>რვა</Button>
            <Button>ცხრა</Button>
            <Button>ათი</Button>
        </UniformGrid>
    </StackPanel>
</TabItem>
```

მივიღებთ, რომ ფანჯრის გაწელით სვეტების რაოდენობა არ იცვლება, როგორც Wrap–ის დროს.

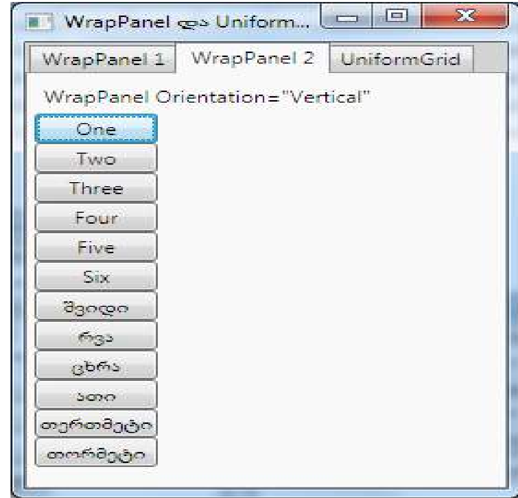
TextBlok–ელემენტი გამოიყენება ტექსტური სათაურების ჩასასმელად.

ნახ.15.78

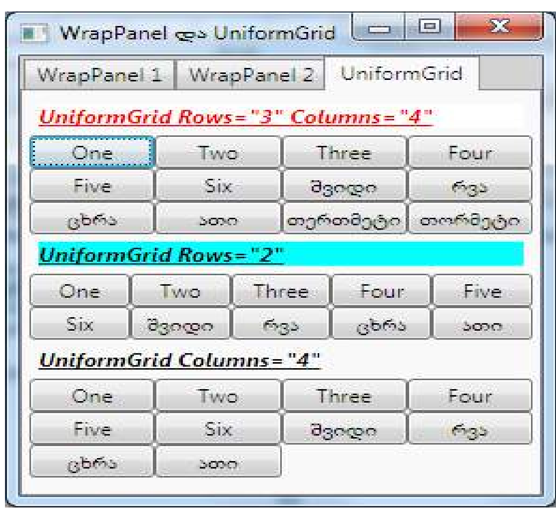
საბოლოო შედეგები მოცემულია 15.79 ა-დ ნახაზებზე:

ნახ.15.79–ა

ნახ.15.79–ბ



ნახ.15.79–გ



ნახ.15.79–დ

15.14. WPF-ის ინტერფეისული ელემენტების განლაგების მენეჯერი: Grid პანელი

განვიხილოთ WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების მენეჯერის, კერძოდ, Grid პანელის დამუშავება და მისი ფუნქციურობის გამოყენების საკითხი.

პროგრამულ აპლიკაციებში ინტერფეისული ელემენტების დინამიკური განლაგებისთვის ხშირად გამოიყენება Grid პანელი.

ჩვენ გავეცანით UniformGrid პანელს (Uniform–ერთგვაროვანი, ერთნაირი, თანაბარი; Grid – ბადე). აქ ყველა უჯრედს ჰქონდა თანაბარი ზომა. ხშირად ეს არაა საკმარისი და მოითხოვს მის გაფართოებას. ამისათვის კი გამოიყენება Grid პანელი. ასეთი პანელის მეზობელ სტრიქონებს შეიძლება ჰქონდეს განსხვავებული სიმაღლე, ხოლო მეზობელ სვეტებს – კი განსხვავებული სიგანე.

Grid–ელემენტი ყველაზე მოქნილი და უნივერსალურია ადრე განხილულ განთავსების მენეჯერებს შორის. იგი ფანჯრებისთვის ავსებს Table ელემენტის HTML ფუნქციონალობას. ამიტომაც, WPF-აპლიკაციის შემქმნელი ოსტატი პროგრამა თავიდანვე ათავსებს მარკირების Grid ელემენტს.

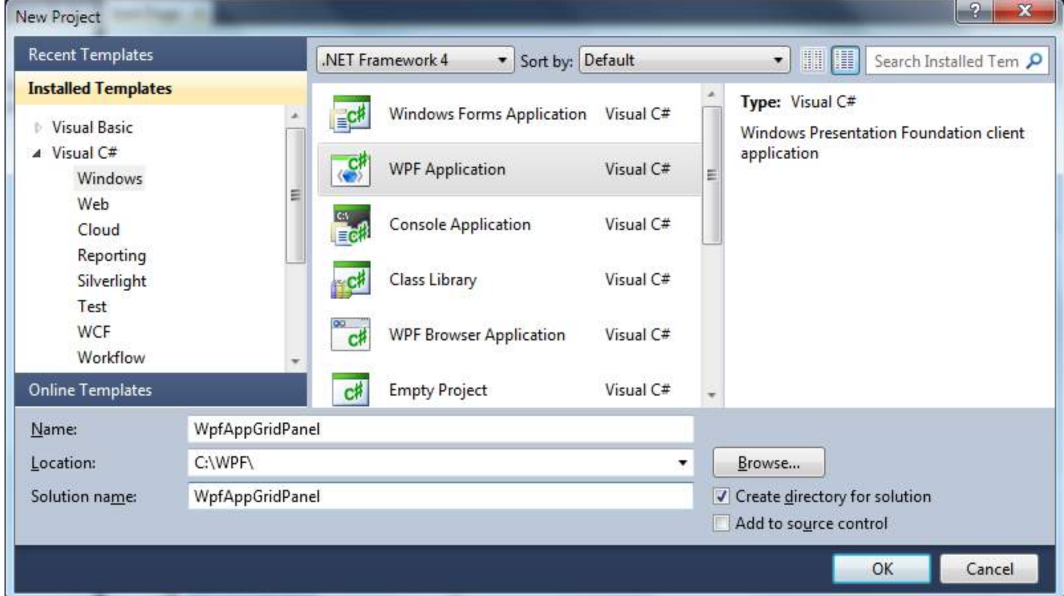
Grid ბადეში განლაგება შედგება ორი ეტაპისგან: ჯერ სრულდება სტრიქონებისა და სვეტების განსაზღვრა, ხოლო შემდეგ იწარმოება შვილობილი ელემენტების მოცემა და მათი განთავსება სლოტების მიხედვით. ყველაზე მარტივი ხერხი Grid–ის გამოსაყენებლად არის RowDefinitions და ColumnDefinitions თვისებების მიცემა. შემდეგ დაემატოს რამდენიმე შვილობილი ელემენტი მათზე მიბმული თვისებებით Grid.Row და Grid.Column , განისაზღროს – რომელი ელემენტი რომელ სლოტში (გრიდის უჯრედში) მოთავსდეს. არსებობს Grid–ის აწყობის უფრო ზუსტი და ფართო შესაძლებლობები, რომელთაც ქვემოთ განვიხილავთ პრაქტიკული ამოცანების დახმარებით.

განვიხილოთ პრაქტიკული მაგალითი:

1. შევქმნათ ახალი WPF აპლიკაცია WpfAppGrid პროექტის სახელით (ნახ.15,80).
2. MainWindow.xaml ფაილისთვის შევიტანოთ შემდეგი კოდის ფრაგმენტი:

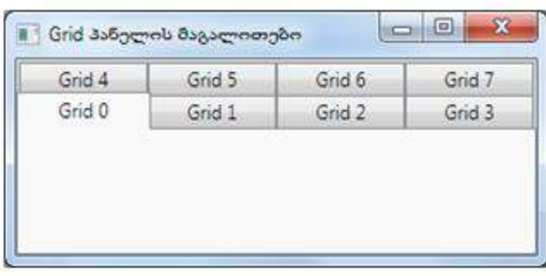
```
<TabControl>
    <TabItem Header="Grid 0">
    </TabItem>
        <TabItem Header="Grid 1">
        </TabItem>
    <TabItem Header="Grid 2">
    </TabItem>
        <TabItem Header="Grid 3">
       </TabItem>
```

```
    <TabItem Header="Grid 4">
    </TabItem>
        <TabItem Header="Grid 5">
        </TabItem>
    <TabItem Header="Grid 6">
    </TabItem>
        <TabItem Header="Grid 7">
        </TabItem>
</TabControl>
```



ნახ.15.80

მიიღება შედეგი:



ნახ.15.81

ჩვენ უნდა განვიხილოთ Grid-პანელის შვიდი შემთხვევა (მაგალითებით).

3. მთლიანი ტექსტი:

```
<Window x:Class="WpfApp8.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Grid პანელის მაგალითები"
   Height="300" Width="300"
   Background="LightGray"
      >
   <TabControl>
      <TabItem Header="Grid 0">
         <Grid>
            <Grid.RowDefinitions>
               <RowDefinition />
               <RowDefinition />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width="Auto" />
               <ColumnDefinition />
            </Grid.ColumnDefinitions>
         <Button Grid.Row="0" Grid.Column="0"
                        Background="LightPink">პირველი</Button>
         <Button Grid.Row="1" Grid.Column="0"
                        Background="Lime">მეორე</Button>
         <Button Grid.Row="1" Grid.Column="1"
                        Background="Aquamarine">მესამე (შიგთავსით)</Button>
         <Button Grid.Row="0" Grid.Column="1"
                        Background="Yellow">მეოთხე</Button>
         </Grid>
      </TabItem>
      <TabItem Header="Grid 1">
         <Grid HorizontalAlignment="Center" VerticalAlignment="Center">
            <Grid.RowDefinitions>
               <RowDefinition Height="Auto" />
               <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width="Auto" />
               <ColumnDefinition Width="Auto" />
```

```
            </Grid.ColumnDefinitions>
        <Button Grid.Row="0" Grid.Column="0"
                        Background="LightPink">პირველი</Button>
        <Button Grid.Row="0" Grid.Column="1"
                        Background="Lime">მეორე</Button>
         <Button Grid.Row="1" Grid.Column="0"
                     Background="Aquamarine">მესამე (შიგთავსით)</Button>
         <Button Grid.Row="1" Grid.Column="1"
                     Background="Yellow">მეოთხე</Button>
      </Grid>
    </TabItem>
    <TabItem Header="Grid 2">
        <Grid>
            <Grid.RowDefinitions>
                 <RowDefinition Height="50" />
                 <RowDefinition Height="*" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                 <ColumnDefinition Width="50" />
                 <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
 <Button Grid.Row="0" Grid.Column="0" MinWidth="0"
                            Background="LightPink">პირველი</Button>
 <Button Grid.Row="0" Grid.Column="1" MinWidth="0"
                            Background="Lime">მეორე</Button>
<Button Grid.Row="1" Grid.Column="0" MinWidth="0"
                            Background="Aquamarine">მესამე</Button>
<Button Grid.Row="1" Grid.Column="1" MinWidth="0"
                            Background="Yellow">მეოთხე</Button>
    </Grid>
  </TabItem>
  <TabItem Header="Grid 3">
        <Grid>
            <Grid.RowDefinitions>
                 <RowDefinition Height="2*" />
                 <RowDefinition Height="1*" />
            </Grid.RowDefinitions>
```

```
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="2*" />
                <ColumnDefinition Width="1*" />
             </Grid.ColumnDefinitions>
<Button Grid.Row="0" Grid.Column="0" MinWidth="0"
                              Background="LightPink">პირველი</Button>
<Button Grid.Row="0" Grid.Column="1" MinWidth="0"
                              Background="Lime">მეორე</Button>
<Button Grid.Row="1" Grid.Column="0" MinWidth="0"
                              Background="Aquamarine">მესამე</Button>
<Button Grid.Row="1" Grid.Column="1" MinWidth="0"
                             Background="Yellow">მეოთხე</Button>
        </Grid>
      </TabItem>
      <TabItem Header="Grid 4">
        <Grid HorizontalAlignment="Center" VerticalAlignment="Stretch">
             <Grid.RowDefinitions>
                <RowDefinition />
                <RowDefinition />
                <RowDefinition />
                <RowDefinition />
             </Grid.RowDefinitions>
             <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
             </Grid.ColumnDefinitions>
 <Button Grid.Row="0" Background="LightPink">პირველი</Button>
 <Button Grid.Row="1" Background="Lime">მეორე</Button>
 <Button Grid.Row="2" Background="Aquamarine">მესამე (ყველაზე განიერი
                             შიგთავსით) </Button>
 <Button Grid.Row="3" Background="Yellow">მეოთხე</Button>
    </Grid>
 </TabItem>
 <TabItem Header="Grid 5">
     <Grid Grid.IsSharedSizeScope="True" HorizontalAlignment="Center">
             <Grid.RowDefinitions>
                <RowDefinition />
                <RowDefinition />
```

```
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" SharedSizeGroup="myKey" />
                <ColumnDefinition Width="102" SharedSizeGroup="myKey" />
            </Grid.ColumnDefinitions>
<Button Grid.Row="0" Grid.Column="0" MinWidth="0"
                     Background="LightPink">პირველი</Button>
<Button Grid.Row="0" Grid.Column="1" MinWidth="0"
                     Background="Lime">მეორე</Button>
<Button Grid.Row="1" Grid.Column="0" MinWidth="0"
                     Background="Aquamarine">
                             ყველაზე განიერი</Button>
<Button Grid.Row="1" Grid.Column="1" MinWidth="0"
                     Background="Yellow">მეოთხე</Button>
        </Grid>
      </TabItem>
    <TabItem Header="Grid 6">
        <StackPanel Grid.IsSharedSizeScope="True">
            <TextBlock HorizontalAlignment="Center">
                ბადეები სვეტების განზოგადებული ზომებით
            </TextBlock>
        <StackPanel Margin="5" />
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition />
                    <RowDefinition />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" SharedSizeGroup="myKey1" />
                    <ColumnDefinition Width="Auto" SharedSizeGroup="myKey2" />
                </Grid.ColumnDefinitions>
    <Button Grid.Row="0" Grid.Column="0"
                         Background="LightPink">პირველი</Button>
    <Button Grid.Row="0" Grid.Column="1"
                         Background="Lime">მეორე</Button>
    <Button Grid.Row="1" Grid.Column="0"
                         Background="Aquamarine">
```

```
                              ყველაზე განიერი 1–ელ სვეტეში  </Button>
 <Button Grid.Row="1" Grid.Column="1"
                              Background="Yellow">მეოთხე</Button>
</Grid>
     <StackPanel Margin="5" />
         <Grid>
            <Grid.RowDefinitions>
               <RowDefinition />
               <RowDefinition />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width="Auto" SharedSizeGroup="myKey1" />
               <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
   <Button Grid.Row="0" Grid.Column="0"
                                Background="LightPink">1</Button>
   <Button Grid.Row="0" Grid.Column="1"
                                Background="Lime">2</Button>
   <Button Grid.Row="1" Grid.Column="0"
                                Background="Aquamarine">3</Button>
   <Button Grid.Row="1" Grid.Column="1"
                                Background="Yellow">4</Button>
 </Grid>
   <StackPanel Margin="5" />
     <Grid>
            <Grid.RowDefinitions>
               <RowDefinition />
               <RowDefinition />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width="Auto" />
               <ColumnDefinition Width="Auto" SharedSizeGroup="myKey2" />
            </Grid.ColumnDefinitions>
     <Button Grid.Row="0" Grid.Column="0"
                                  Background="LightPink">1</Button>
     <Button Grid.Row="0" Grid.Column="1"
                                  Background="Lime">2</Button>
```

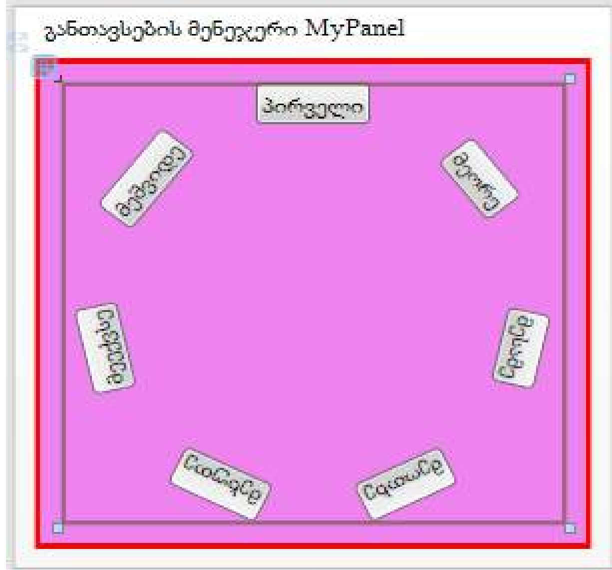
```
      <Button Grid.Row="1" Grid.Column="0"
                                    Background="Aquamarine">3</Button>
      <Button Grid.Row="1" Grid.Column="1"
                                    Background="Yellow">4</Button>
    </Grid>
      <StackPanel Margin="5" />
         <Grid>
            <Grid.RowDefinitions>
               <RowDefinition />
               <RowDefinition />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
               <ColumnDefinition Width="Auto" SharedSizeGroup="myKey1" />
               <ColumnDefinition Width="Auto" SharedSizeGroup="myKey2" />
            </Grid.ColumnDefinitions>
     <Button Grid.Row="0" Grid.Column="0"
                                    Background="LightPink">1</Button>
     <Button Grid.Row="0" Grid.Column="1"
                                    Background="Lime">2</Button>
     <Button Grid.Row="1" Grid.Column="0"
                                    Background="Aquamarine">3</Button>
     <Button Grid.Row="1" Grid.Column="1"
                                    Background="Yellow">განიერი მე–2
                                                         სვეტისთვის</Button>
    </Grid>
  </StackPanel>
</TabItem>
<TabItem Header="Grid 7">
     <Grid>
         <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
         </Grid.RowDefinitions>
         <TextBlock Grid.Row="0" HorizontalAlignment="Center" Margin="5">
            სვეტების გაერთიანება და GridSplitter ელემენტი
         </TextBlock>
         <Grid Grid.Row="1">
```

```
            <Grid.RowDefinitions>
                <RowDefinition Height="*" />
                <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto" MinHeight="50" MaxHeight="150" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*" />
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="Auto" MinWidth="40"
                                                MaxWidth="200" />
            </Grid.ColumnDefinitions>
    <Button Grid.Row="0" Grid.ColumnSpan="3"
                        Background="LightPink">პირველი და
                                            მეორე</Button>
          <GridSplitter
                Grid.Row="1"
                Grid.ColumnSpan="3"
                Height="2"
                ResizeDirection="Rows"
                ResizeBehavior="PreviousAndNext"
                HorizontalAlignment="Stretch" />
  <Button Grid.Row="2" Grid.Column="0"  Background="Aquamarine">მესამე</Button>
            <GridSplitter
                Grid.Row="2"
                Grid.Column="1"
                Width="2"
                ResizeDirection="Columns"
                ResizeBehavior="PreviousAndNext"
                VerticalAlignment="Stretch" />
  <Button Grid.Row="2" Grid.Column="2" Background="Yellow">მეოთხე</Button>
          </Grid>
        </Grid>
    </TabItem>
  </TabControl>
</Window>
```

15.15. WPF-ის საკუთარი განლაგების მენეჯერის დამუშავება: MyPanel

განვიხილოთ WPF-აპლიკაციების ინტერფეისული ელემენტების განლაგების საკუთარი მენეჯერის MyPanel დამუშავებისა და ფუნქციურობის შესწავლა.

ამოცანა: ავაგოთ Wpf-აპლიკაცია, რომლის ინტერფეისის ფანჯარაში წრიულად განთავსდება რამდენიმე ელემენტი (მაგალითად, ღილაკები), თითოეული განსაზღვრული კუთხის მობრუნებით (ნახ.15.82).



ნახ.15.82

ესაა არასტანდარტული ამოცანა და მის გადასაწყვეტად განლაგების ტიპური მენეჯერები არ გამოდგება. საჭიროა მისთვის სპეციალური ალგორითმისა და კოდის შექმნა. განვიხილოთ ასეთი შესაძლებლობანი.

WPF–ის ყველა სტანდარტული პანელი (DockPanel, StackPanel, Canvas, UniformGrid, Grid) არის აბსტრაქტული საბაზო კლასის Panel მემკვიდრე. ჩვენ მიერ ასაგები ახალი ელემენტების განლაგების „კერძო მენეჯერიც“ უნდა იყოს ამ ბიბლიოთეკური კლასის მემკვიდრე.

Panel კლასი, თავის მხრივ მემკვიდრეა UIElement კლასის, ამიტომ ჩვენი კლასი ყველა აუცილებელ გაფართოებას მიიღებს თავის წინაპრებიდან. Panel კლასი არის მემკვიდრეობის ჯაჭვის ისეთი მწვერვალი, რომელიც მოიცავს ყველა ნიმუშს (პატერნს: ყალიბი, „აგური“, „სამშენებლო ბლოკი“, საბაზო ფუნქციური ობიქტი), რომლებიც აუცილებელია განლაგების ნებისმიერი მენეჯერის ასაგებად. ჩვენ შემთხვევაში გაფართოების კლასში დაგვჭირდება რამდენიმე ვირტუალური მეთოდის გადაფარვა.

UIElement ობიექტური მოდელის ის ნაწილი, რომელიც მიეკუთვნება განლაგების მენეჯერს, საკმარისად მარტივია. მეთოდები Measure, MeasureCore, Arrange და ArrangeCore არეალიზებენ განლაგების ორ ეტაპს, ხოლო Visibility თვისება წყვეტს, უნდა აისახოს თუ არა შვილობილი და უნდა განთავსდეს თუ არა იგი.

Visibility თვისება იძლევა შვილობილი ელემენტის განთავსების სამ ხერხს. ავტომატურად ესაა Visible: ელემენტი აისახება და იკავებს ადგილს ეკრანზე. Hidden–ით ის ეკრანზე არ ჩანს, მაგრამ ადგილს იკავებს. Collapsed–ს დროს ელემენტი არც აისახება და არ ადგილს იკავებს.

შვილობილ ელემენტსა და განთავსების მენეჯერს შორის ურთიერთქმედების მექანიზმი შედგება ორი ნაწილისაგან:

- კონტრაქტისაგან, რომელიც აღწერს თუ როგორ იღებს მონაწილეობას განთავსებაში ელემენტი;
- ამ კონტრაქტის რეალიზაციების ერთობლიობისაგან.

არაა არავითარი ჩაშენებული განთავსება, მისი ყველა კონკრეტული ხერხი აგებულია საბაზო კლასების გაფართოების პრინციპზე.

WPF–ის ყველა ელემენტი, მათ შორის განლაგების მენეჯერებიც, ადაპტირდება თავის შიგთავსზე. ეს კონცეფცია გამოიყენება მომხმარებლის ინტერფეისის აგების ყველა დონეზე: ფანჯრებს შეუძლია თავიანთი ზომების დაყვანა მათში არსებულ ელემენტებზე, ღილაკებს – მათზე არსებულ წარწერებზე, ტექსტური ველები – იცვლიან ზომებს ისე, რომ მასზე ყველა სიმბოლო მოთავსდეს.

მენეჯერები ახორციელებს თავის შვილობილი ელემენტების ორეტაპიან განთავსებას: გაზომვა და დაყენება. თავიდან მენეჯერი გამოკითხავს ყველა შვილობილ ელემენტს იმის შესახებ, თუ როგორი ფაქტობრივი ზომაა მისთვის მისაღები, რომ ეკრანზე მთლიანად დაატიოს შიგთავსი. ესაა გაზომვის ეტაპი. შემდეგ მენეჯერი ამ ინფორმაციას მიუსადაგებს თავის ანაწყობათა შესაძლებლობებს, რამდენი შეუძლია დაატიოს, და იწყებს მონტაჟს.

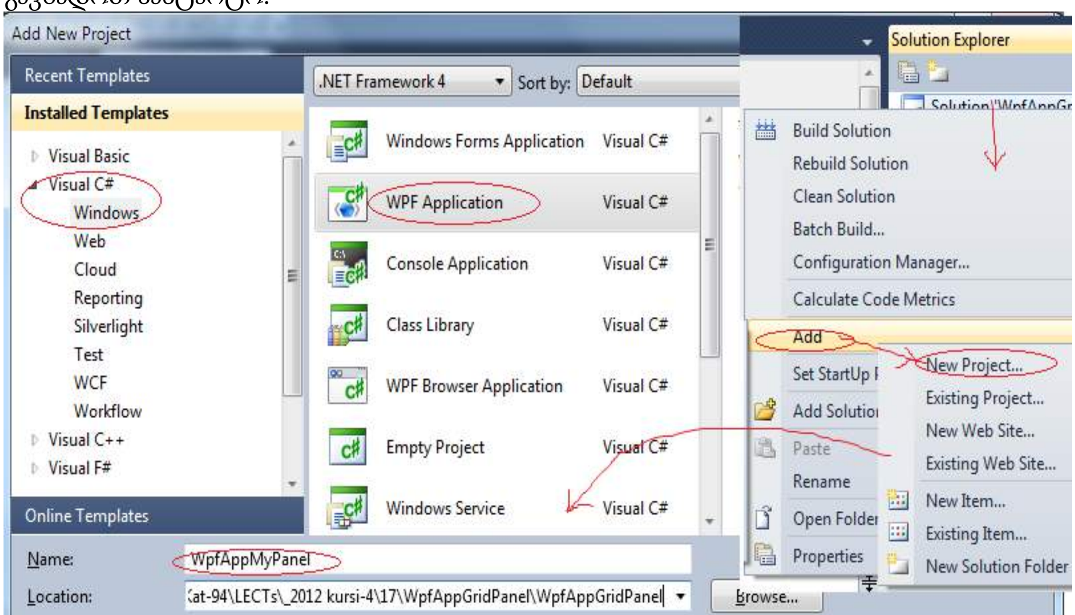
მონტაჟის დროს მშობელი ულაპარაკოდ თხოულობს თავისი თითოეული შვილობილი ელემენტიდან, რომ მან შეარჩიოს განსაზღვრული ზომა და მდებარეობა. ეს მოდელი უზრუნველყოფს, რომ მშობელმა და შვილება მოილაპარაკონ ეკრანზე ადგილის შესახებ (ადგილზე მზის ქვეშ). ამ დროს სამი ზომა ფიგურირებს:

- შესაძლებელი ზომა (available size) – მაქსიმალური არე, რომელიც შეუძლია მშობელს თავიდან მისცეს შვილს;
- სასურველი ზომა (desired size) – ზომა, რომელსაც ისურვებს შვილი დასაყენებლად;
- ფაქტობრივი ზომა (actual size) – საბოლოო ზომა, რომელსაც მშობელი გამოუყოფს შვილს.

ელემენტის ზომებზე შეიძლება შეზღუდვების დაყენება MinWidth, MaxWidth, MinHeight და MaxHeight, რომელთა შიგნითაც მოხდება ადაპტაცია შიგთავსის მიხედვით. თვისებები Width და Height ჩვეულებისამებრ არ მოიცემა, რაც უფლებას იძლევა ზომის ავტომატურად შერჩევისთვის. მაგრამ თუ ისინი მოცემულია ცხადად, მაშინ ადაპტაცია ზომებით გამოირთვება. თვისებები ActualWidth და ActualHeight ხდება განსაზღვრული მხოლოდ ორეტაპიანი ადაპტაციის დასრულების შემდეგ და ნიშნავს ელემენტის ფაქტობრივ ზომას, რომელიც დააყენა განლაგების მენეჯერმა.

განვიხილოთ პრაქტიკული მაგალითი:

1. დავამატოთ Solution Explorer-ში ახალი, WpfAppMyPanel სახელის პროექტი და გავხადოთ სასტარტო.



ნახ.15.83

2. შევიტანოთ C#-ის MainWindow.xaml.cs ფაილში შემდეგი კოდი:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
```

```
using System.Windows.Navigation;
using System.Windows.Shapes;
namespace WpfAppMyPanel
{
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();
      }
   }
}
namespace WpfAppMyPanel
{
   class MyPanel : Panel
   {
    // პირველი ეტაპი: გაზომვა
    // შესასვლელზე: - რამდენი აქვს მშობელს მენეჯერისათვის
    // გამოსასვლელზე:- რამდენს ითხოვს მენეჯერი თავისთვის
  protected override Size MeasureOverride(Size availableSize)
    {
      double maxChildWidth = 0.0;
      double maxChildHeight = 0.0;

               // გაიზომოს ყოველი შვილის მოთხოვნა და
                //  დადგინდეს ყველაზე მაქსიმუმები
      foreach (UIElement child in this.InternalChildren)
       {
       child.Measure(availableSize);
       maxChildWidth =  Math.Max(child.DesiredSize.Width,
                                                        maxChildWidth);
       maxChildHeight = Math.Max(child.DesiredSize.Height,
                                                      maxChildHeight);
      }

     // მიახლოებითი არასრულყოფილი ალგორითმი
     // ყველა ელემენტის განსათავსებლად საჭირო წრის სიგრძე
```

```
   double idealCircumference = maxChildWidth *
                                              this.InternalChildren.Count;
   // წრის საჭირო რადიუსი
double idealRadius = (idealCircumference / (Math.PI * 2) +
                                                          maxChildHeight);
    // წრეზე შემოხაზული კვადრატის აუცილებელი ზომები
Size ideal = new Size(idealRadius * 2, idealRadius * 2);
Size desired = ideal; // მენეჯერს უნდა ამდენი თავისთვის
 // თუ მენეჯერისათვის გამოყოფილი ზომა არაა უსასრულო
        if (!double.IsInfinity(availableSize.Width))
        {
           // თუ გამოყოფილი ზომა მენეჯერს არ ყოფნის
           if (availableSize.Width < desired.Width)
            {
         // მენეჯერის ზომის კორექტირება – სიგანისთვის
              desired.Width = availableSize.Width;
            }
        }
        // იმავე სიმაღლისთვის
        if (!double.IsInfinity(availableSize.Height))
        {
           if (availableSize.Height < desired.Height)
           {
              desired.Height = availableSize.Height;
           }
        }
        return desired;
    }

    // მეორე ეტაპი – მონტაჟი (კონტრაქტის დადება)
    // მენეჯერისათვის გამოყოფილი ფართობის დაყოფა
protected override Size ArrangeOverride(Size finalSize)
    {
      // ვათავსებთ კვადრატულ მენეჯერს მშობლის მიერ
              // გამოყოფილ უბნის ცენტრში
      Rect layoutRect;
      if (finalSize.Width > finalSize.Height)
```

```
        {
         layoutRect = new Rect((finalSize.Width -
                                   finalSize.Height) / 2,0,
                  finalSize.Height, finalSize.Height);
         }
        else
         {
          layoutRect = new Rect(0,(finalSize.Height -
                                   finalSize.Width) / 2,
                  finalSize.Width, finalSize.Width);
         }
      double angleInc = 360.0 / this.InternalChildren.Count;
      double angle = 0;

        // ვათავსებთ წრიულად ყველა შვილ ელემენტს
       foreach (UIElement child in this.InternalChildren)
       {
         // ვათავსებთ შემდეგ ელემენტს წრის ზედა წერტილში
            Point childLocation = new Point(
            layoutRect.Left + ((layoutRect.Width -
                                     child.DesiredSize.Width) / 2),
                                     layoutRect.Top);
         // გადაადგილება საათის ისრის მიხედვით და
                //  შემობრუნება ღერძის გარშემო
           child.RenderTransform = new RotateTransform(angle,
              child.DesiredSize.Width / 2,
              finalSize.Height / 2 - layoutRect.Top);

  // დადგინდა შვილი–ელემენტის საბოლოო ზომა და მდებარეობა
  child.Arrange(new Rect(childLocation, child.DesiredSize));
            angle += angleInc;
       }
      return base.ArrangeOverride(finalSize);
    }
  }
}
```

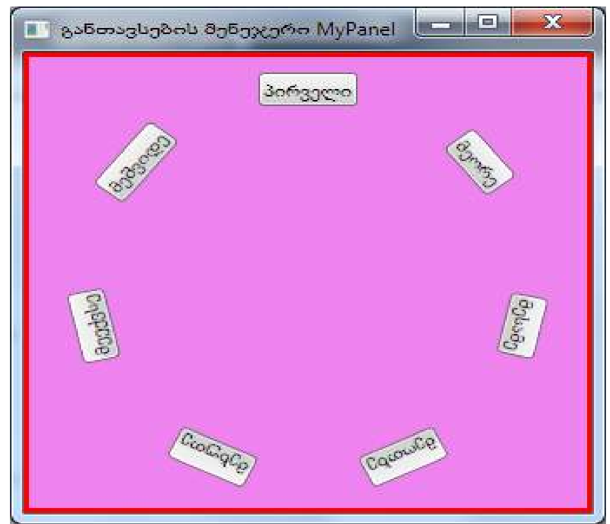
3. Arrange მეთოდის გამოძახება შვილი–ელემენტისთვის ასაბუთებს მასთან კონტრაქტის დადებას, რომლის გარეშეც ელემენტი ვერ გამოჩნდება ეკრანზე.

4. შევავსოთ MainWindow.xaml ფაილი შემდეგი XAML სკრიპტით:

```
<Window x:Class="WpfAppMyPanel.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Height="300" Width="300"
      Title="განთავსების მენეჯერი MyPanel"
      xmlns:MyMy="clr-namespace:WpfAppMyPanel"     >
   <Border
      BorderThickness="3"
      CornerRadius="0"
      BorderBrush="Red"
      Padding="10"
      Background="Violet"       >
      <MyMy:MyPanel>
         <Button>პირველი</Button>
         <Button>მეორე</Button>
         <Button>მესამე</Button>
         <Button>მეოთხე</Button>
         <Button>მეხუთე</Button>
         <Button>მეექვსე</Button>
         <Button>მეშვიდე</Button>
      </MyMy:MyPanel>
      </Border>
</Window>
```

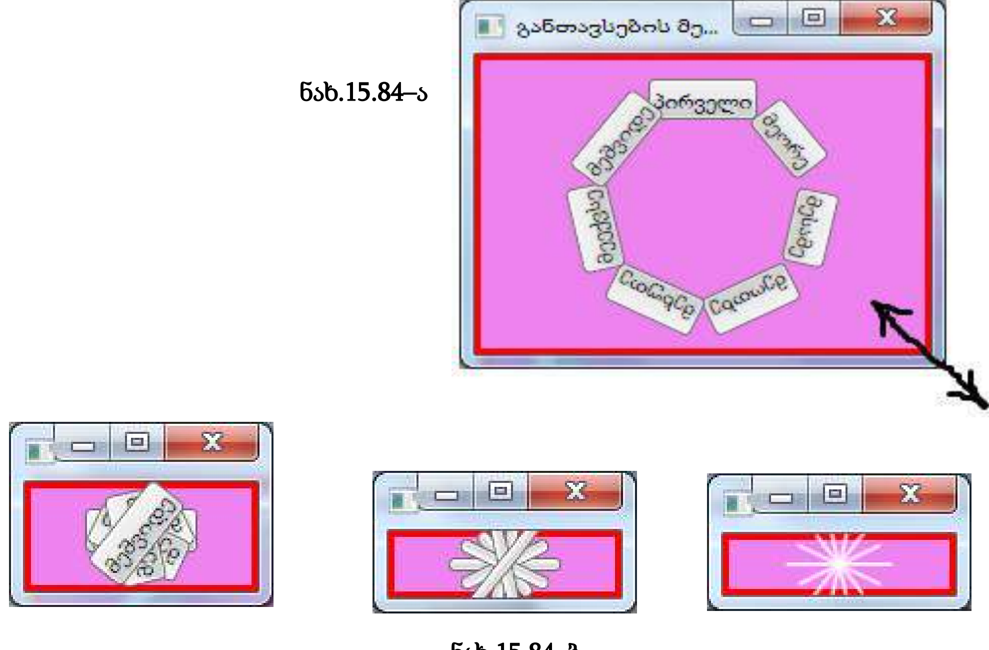
შენიშვნა: MyPanel კლასის ხილვადობისათვის დიზაინის ნაწილში ჩვენ მივუერთეთ მას CLR –შესრულების გარემოს სახელსივრცე, აღნიშნული ნებისმიერი სახელით (MyMy=”clr…).

5. აპლიკაციის ამუშავებით მიიღება შედეგები:



ნახ.15.84

ნახ.15.84–ა



ნახ.15.84–ბ

V ნაწილი

Workflow ტექნოლოგია

XVI თავი. ბიზნესპროცესების (Workflow) ვიზუალური დაპროგრამების ელემენტები 481
XVII თავი. ბიზნესპროცესების (Workflow) დაპროექტება ……………………............... 521
XVIII თავი. კორპორაციის პროგრამული სისტემის დაპროექტება UML/2 და Workflow ტექნოლოგიებით …………………………………………………...… 559

WF (Windows Workflow Foundation) ტექნოლოგია .NET-ში არის სრულიად ახალი პარადიგმა სამუშაო (ბიზნეს) პროცესებზე (workflow) ბაზირებული აპლიკაციების ასაგებად. იგი ფუნდამენტურად ახლად გააზრებული ტექნოლოგიაა [17, 63].

Workflow ტექნოლოგიის საშუალებით შესაძლებელია სამი ტიპის პროცესის აღწერა:

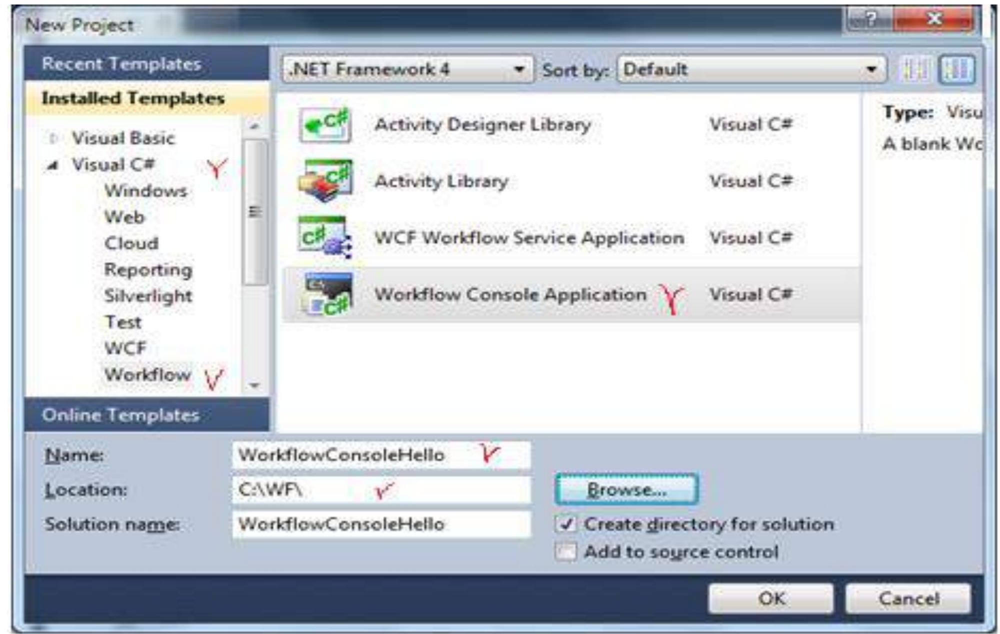
- მიმდევრობითი პროცესი (Sequential Workflow) — ერთი ბიჯიდან მეორეზე გადასვლა უკან დაბრუნების გარეშე;
- წესებით მართვადი პროცესი (Rules-driven Workflow) — ესაა მიმდევრობითი პროცესის კერძო შემთხვევა, რომელშიც გადასვლა შემდგომ ბიჯზე განისაზღვრება წესების ერთობლიობით;
- სასრული ავტომატი (State-Machine Workflow) — გადასვლა ერთი მდგომარეობიდან სხვა მდგომარეობაზე, ასევე შესაძლებელია ნებისმიერი უკან დაბრუნება წინა მდგომარეობებში.

თავიდან ჩვენ გავეცნობით მარტივი Workflow პროცესების აგების საფუძვლებს და ძირითად ცნებებს, შემდეგ კი განვიხილავთ შედარებით რთულ ამოცანებსაც.

XVI თავი
ბიზნესპროცესების (Workflow) ვიზუალური დაპროგრამების ელემენტები

16.1. მარტივი ბიზნესპროცესის (Workflow-ის) აგება

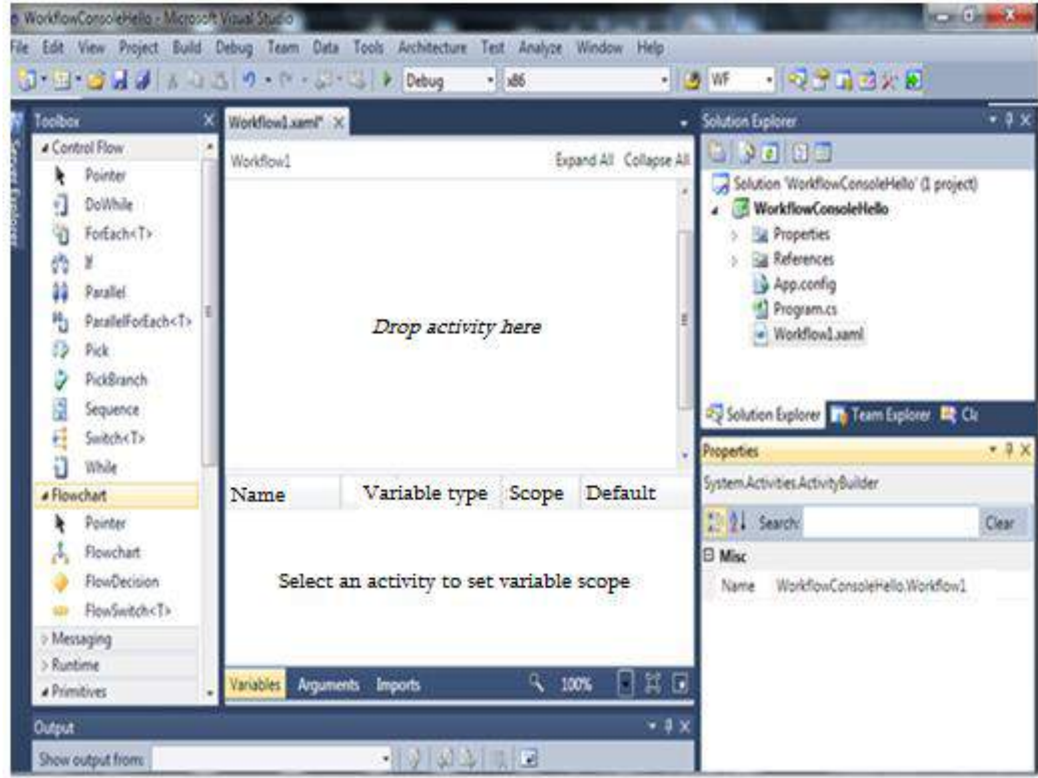
განვიხილოთ მარტივი სამუშაო პროცესის (workflow-ის) შექმნა Visual Studio.NET გარემოში. შევარჩიოთ ახალი პროექტის სახელი (მაგალითად, WorkflowConsoleHello), მისი შენახვის ადგილი (C:\WF\). აგრეთვე ავირჩიოთ Template-ში Visual C# და Workflow, ხოლო შუა ფანჯრიდან Workflow Console Application (ნახ.16.1).



ნახ.16.1. ახალი workflow პროექტის შექმნა

შაბლონი (Template) აგენერირებს Program.cs ფაილს, რომელიც რეალიზებას უკეთებს კონსოლურ აპლიკაციას (დანართს). იგი ასევე აგენერირებს Workflow1.xaml ფაილს, რომელიც განსაზღვრავს ქმედებას (აქტიურობას) სამუშაო პროცესში (workflow-ში). XAML ენა გამოიყენება პროგრამული ელემენტების გამოსაცხადებლად (როგორც WPF აპლიკაციაში). ოღონდაც ლებელის, ტექსტბოქსისა და ბადის ნაცვლად ეს ფაილი შეიცავს ჩვენ მიერ განსაზღვრულ სამუშაო პროცესში წარმოებული ელემენტების აქტიურობებს. Visual Studio იძლევა დიზაინერისათვის ქმედებების (აქტიურობების) გრაფიკულად ნახვისა და რედაქტირების საშუალებას.

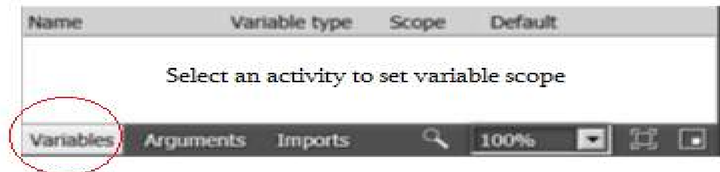
16.2 ნახაზზე ნაჩვენებია Visual Studio–ის ინტეგრირებული დამუშავების გარემო (IDE).



ნახ.16.2. Visual Studio-ს სტანდარტული IT გარემო

მარცხნივ მოთავსებულია ინსტრუმენტების პანელი, რომელიც მოიცავს ჩადგმულ და მომხმარებლის აქტიურობებს, რომელთა გაფართოება შესაძლებელია სხვა ქმედებებითაც. Solution Explorer და თვისებათა ფანჯარა მარჯვნივაა მოთავსებული. ქვემოთ ფანჯარაში გამოიტანება შეცდომების შეტყობინებები, შუალედური შედეგები და სხვა ინფორმაცია.

WF 4.0 დიზაინერი მოთავსებულია შუაში. ქვედა მარჯვენა კუთხეში არის მასშტაბირების კომბობოქსი, სურათის გამადიდებელი და აქტიურობის (ქმედების) მოსაძებნი ღილაკები. ქვედა მარცხენა კუთხეში სამი ელემენტია: ცვლადების, არგუმენტების და იმპორტული ანაწყობის. თუ მუშა პროცესი კლასია, მაშინ ცვლადები იქნება კლასის წევრები. მათი ნახვა შეიძლება გახსნით (ნახ.16.3).



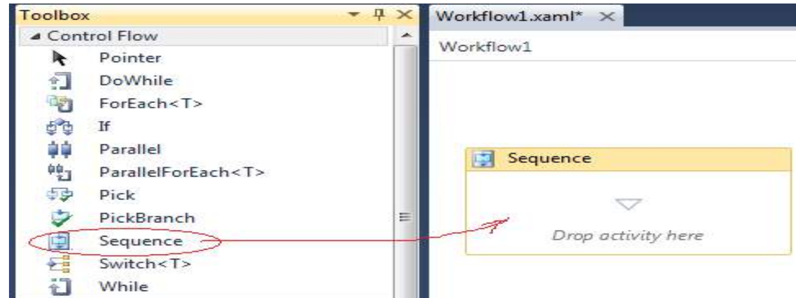
ნახ.16.3. სამუშაო პროცესის ცვლადების ნახვა

- **სამუშაო პროცესის (workflow-ის) დაპროექტება**

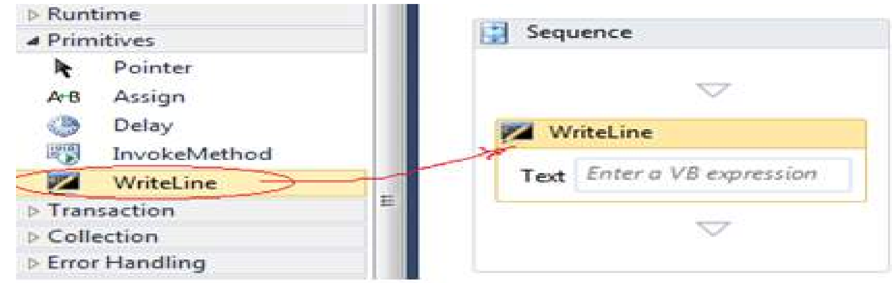
თავიდან სამუშაო ნაკადის კონსტრუქტორი ცარიელია. ინსტრუმენტების პანელიდან საჭირო აქტიურობა გადაიტანება დიზაინერის გარემოში, რითაც განისაზღვრება სამუშაო პროცესის ყოფაქცევა.

ჩვენი პროექტი თავიდან უნდა ასახავდეს მისალმებას „Hello, World !“, შემდეგ კი დაემატება სხვა პროცედურები. გადმოვიტანოთ Sequence ქმედება დიზაინერზე (ნახ.16.4). შემდეგ კი – WriteLine ქმედება ამ Sequence–ზე. სქემას ექნება 16.5 ნახაზზე ნაჩვენები სახე.

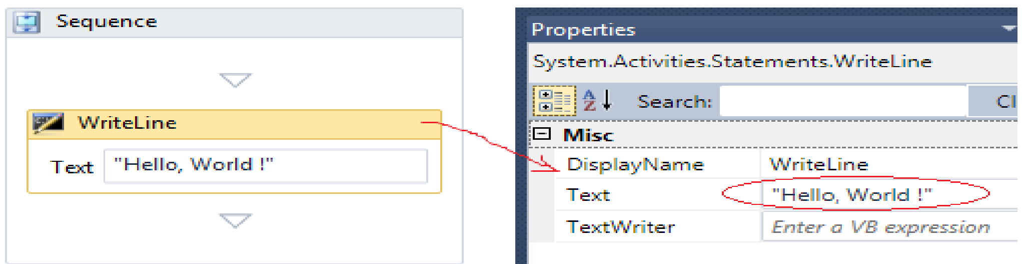
WriteLine-ის Properties-ში ტექსტის შეტანა ნაჩვენებია 16.6 ნახაზზე. სისტემის ამუშავების შემდეგ საბოლოო შედეგი მოცემულია 16.7 ნახაზზე.



ნახ.16.4. Sequence ქმედების გადმოტანა



ნახ.16.5. WriteLine ქმედების დამატება Sequence-ში



ნახ.16.6. Text –ის მნიშვნელობის შეტანა Properties-ში

ნახ.16.7. კონსოლზე გამოსული შედეგი

გავხსნათ პროგრამა Program.cs, რომელიც ამუშავებს კონსოლის აპლიკაციას (ლისტინგი_16.1):

```
// -- ლისტინგი_16.1 ----
using System;
using System.Linq;
using System.Activities;
using System.Activities.Statements;

namespace WorkflowConsoleHello
{
   class Program
   {
      static void Main(string[] args)
      {
         WorkflowInvoker.Invoke(new Workflow1());
        // Console.WriteLine("Press ENTER to exit");
        // Console.ReadLine();
      }
   }
}
```

პროგრამაში ხელითაა ჩამატებული ბოლო ორი სტრიქონი (კომენტარი მოხსენით), რათა შესაძლებელი იყოს შედეგის ნახვა კონსოლზე.

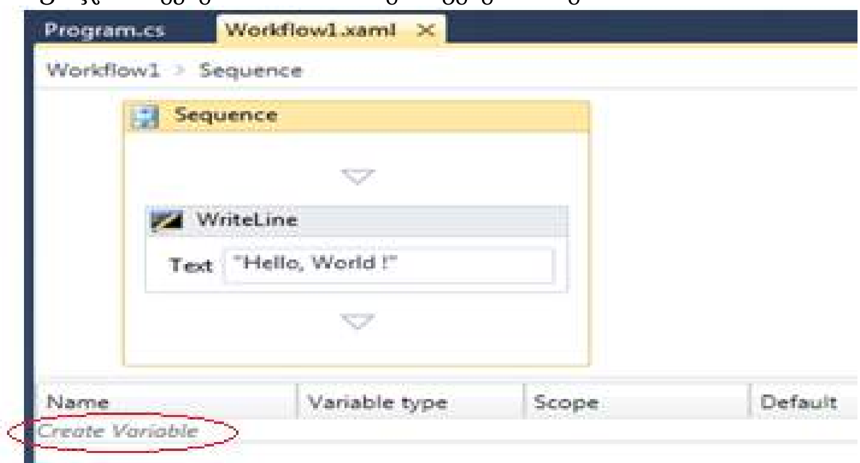
16.2. პროცედურული ელემენტები

განვიხილოთ Workflow-ის ინსტრუმენტების პანელის ძირითადი პროცედურული ელემენტების გამოყენების საკითხი.

WF-ს აქვს პროცედურული ელემენტები, როგორიცაა, მაგალითად, If, While, Assign, Sequence და სხვა. მათი ფუნქციონირების სადემონსტრაციოდ გამოვიყენოთ ზემოთ აგებული მისალმების კოდი. წარმოვადგინოთ ძველებური საათის მექანიზმი, რომელიც ყოველ საათზე გამოსცემს ზარს. გავხსნათ Workflow1.xaml ფაილი.

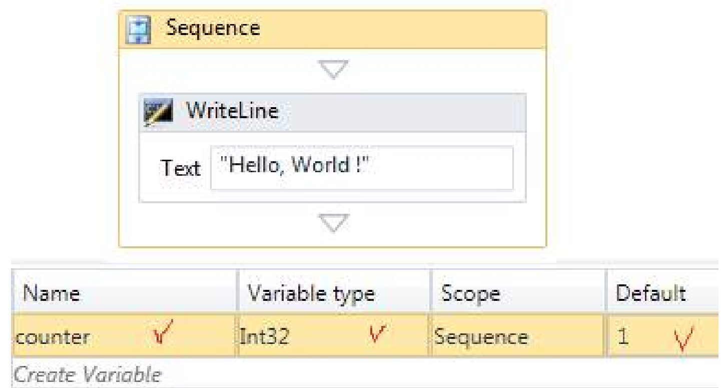
- **ცვლადების გამოყენება**

WF–ში უნდა გამოვაცხადოთ ყველა ცვლადი, რომელიც გამოიყენება მუშა ელემენტებში. ჩვენ შემთხვევაში საჭირო იქნება ორი ცვლადი: ერთი – ზარების რაოდენობისათვის, მეორე – მთვლელისათვის, რომელიც დაითვლის თუ რამდენი ზარი იქნა ამოქმედებული აქამდე. დავაჭიროთ ღილაკს Variables. თუ მასში არაა ელემენტები, ე.ი. არ მომხდარა მათი გამოცხადება. ახლა დავდგეთ მთავარ ქმედებაზე – Sequence, ცვლადების ფანჯარას ექნება 16.8 ნახაზზე ნაჩვენები სახე.



ნახ.16.8

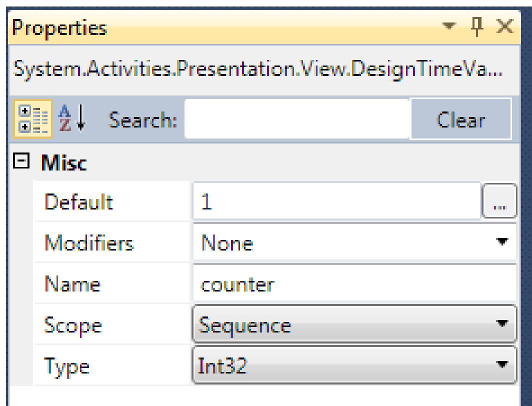
დავკლიკოთ *Create Variable* და შევიტანოთ ცვლადის სახელი, ტიპი და მნიშვნელობა (ნახ.16.9).



ნახ.16.9

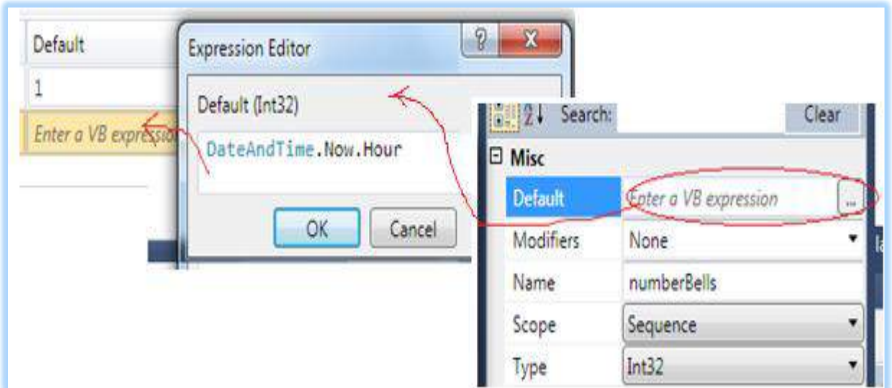
ცვლადი counter ხილვადია როგორც Sequence ქმედების, ისე მისი შვილობილი ელემენტებისათვის (მაგალითად, WriteLine). ცვლადის მნიშვნელობა შევიტანეთ 1.

16.10 ნახაზზე ნაჩვენებია ცვლადების სტრიქონის Properties-ის ფანჯარა. აქაც შესაძლებელია ცვლადების მნიშვნელობათა შეტანა, მაგალითად, Default-ში „1“.



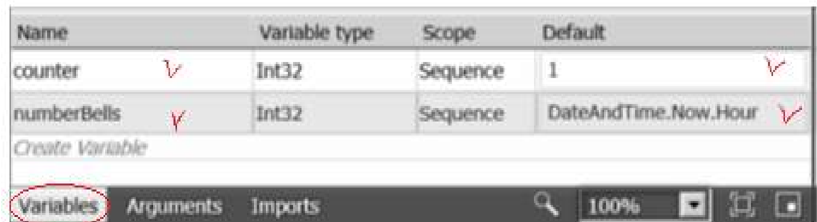
ნახ.16.10

მეორე სტრიქონში, შეიტანება ზარების რაოდენობის ცვლადის მნიშვნელობა, რომელიც გამოიყენებს „*Enter a VB expression*“-ს (ნახ.16.11).



ნახ.16.11

საბოლოო შედეგს ცვლადებისთვის ექნება 16.12 ნახაზზე ნაჩვენები სახე.



Name	Variable type	Scope	Default
counter	Int32	Sequence	1
numberBells	Int32	Sequence	DateAndTime.Now.Hour
Create Variable			

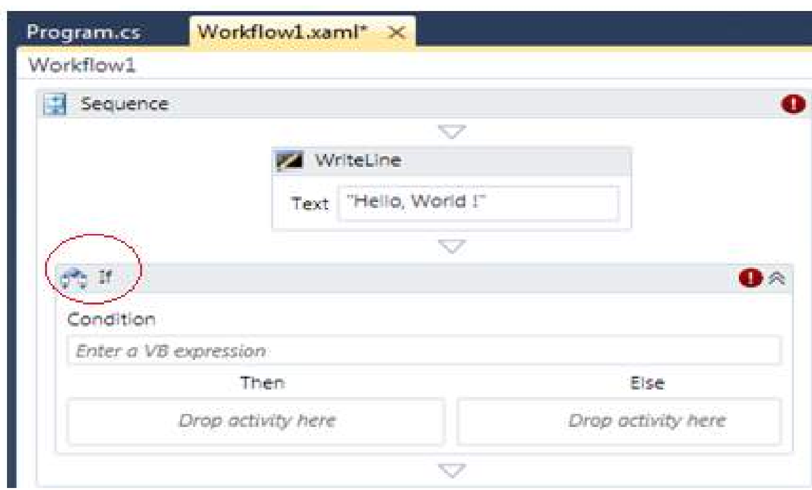
ნახ.16.12

- **If - განშტოების პროცედურა**

DateAndTime კლასის Hour-წევრი აბრუნებს დროის მნიშვნელობას 24–საათიანი ფორმატით. მაგალითად, 2 PM–თვის ის დააბრუნებს 14–ს. ამიტომაც ჩვენ უნდა ავაწყოთ სისტემა ისე, რომ ზარი იყოს 2–ჯერ და არა 14–ჯერ. ამისთვის კოდში უნდა ჩაიწეროს შემდეგი:

```
if (numberBells > 12)
        numberBells -= 12;
```

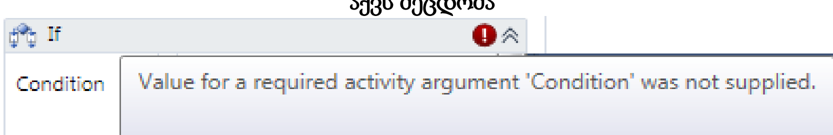
ამ მიზნით გამოიყენება if და Assign ქმედებები. გადმოვიტანოთ ინსტრუმენტების პანელიდან if აქტიურობა Hello აქტიურობის ოდნავ ქვემოთ. დიაგრამა მოცემულია 16.13 ნახაზზე.



ნახ.16.13. if ქმედების დამატება

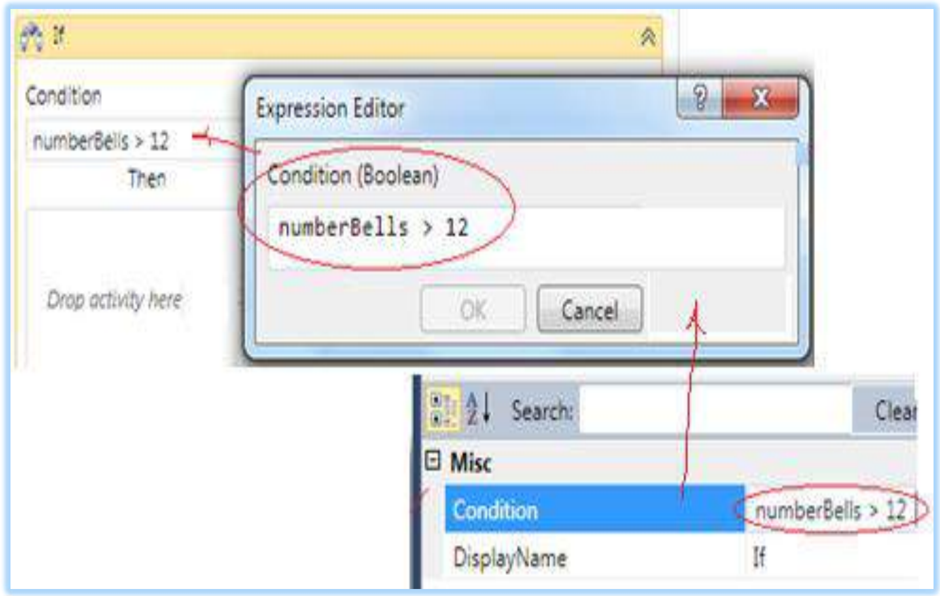
შენიშვნა: ნახაზზე წითელი ძახილის ნიშნით მიეთითება, რომ არის შეცდომა/გაფრთხილება. მაუსის კურსორის მიტანით ჩნდება ტექსტი (ნახ.16.14, 16.15).

ნახ.16.14. გაფრთხილება Sequence-ში, რომ შვილობილს აქვს შეცდომა



ნახ.16.15. გაფრთხილება if-ში, რომ ქმედებას არ აქვს მითითებული პირობა

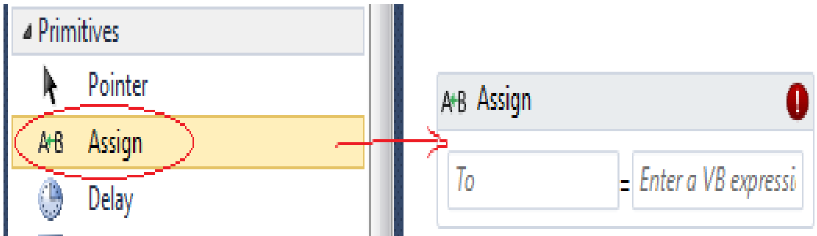
თვისებების ფანჯარაში შევცვალოთ DisplayName–მნიშვნელობა Adjust for PM–ით. if ქმედება შედგება სამი ელემენტისაგან. პირობა Condition განსაზღვრავს ლოგიკას, რომელიც შეფასდება. ეს იქნება ლოგიკური მნიშვნელობა („ჭეშმარიტი" ან „მცდარი"). იგი შეიცავს ქმედებებს, რომლებიც სრულდება, როცა პირობა „ჭეშმარიტია", ან სხვა ქმედებებს, როცა პირობა „მცდარია". მხოლოდ ერთია აუცილებელი. შევიტანოთ პირობა numberBells> 12 (ნახ.16.16).



ნახ.16.16. Condition პირობის შეტანა

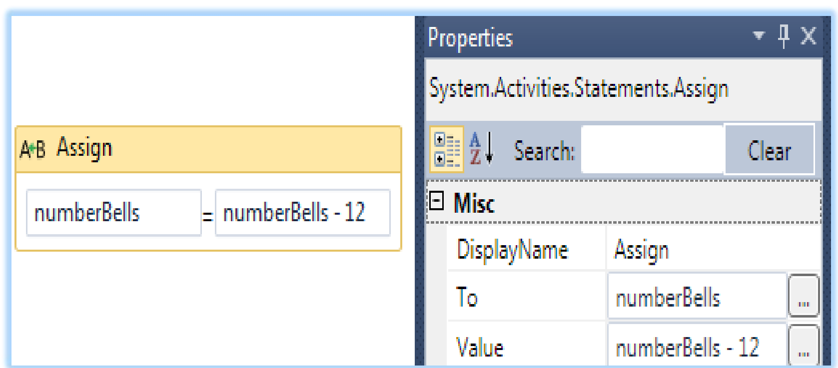
- **Assign - მინიჭების პროცედურა**

გადავიტანოთ ინსტრუმენტების პანელიდან Assign ქმედება. იგი საშუალებას იძლევა ცვლადს ან არგუმენტს მივანიჭოთ მნიშვნელობა. გრაფიკულად მიიღება ასეთი სურათი (ნახ.16.17).



ნახ.16.17. Assign ქმედების დამატება

Assign–ში To და Value, ორივე იღებს მნიშვნელობას. შეტანა შეიძლება უშუალოდ ველში, ან Properties-ის ფანჯარაში (“...“-ით გამოიძახება გამოსახულების შეტანის რედაქტორი). თვისებისთვის (To) შევიტანოთ numberBells. თვისების მნიშვნელობისთვის (Value) კი: numberBells–12. თვისებათა ფანჯარას ასეთი სახე ექნება (ნახ.16.18).

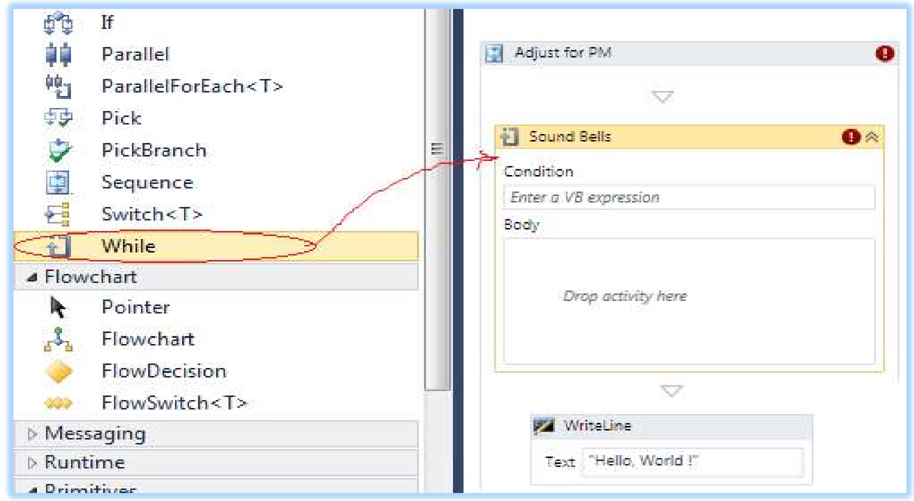


ნახ.16.18. Assign ქმედების თვისებათა ფანჯარა

მრავალი ქმედება არის შედგენილი აქტიურობა (ქმედება), რაც ნიშნავს, რომ იგი შეიძლება შეიცავდეს სხვა სახის ქმედებებს.

- **While - ციკლის პროცედურა**

დავამატოთ While ქმედება ზარის დასარეკად. გადმოვიტანოთ ინსტრუმენტების პანელიდან While აქტიურობა „Adjust for PM"–ის ქვემოთ. შევცვალოთ თვისებებში DisplayName სახელით Sound Bells (ნახ.16.19).



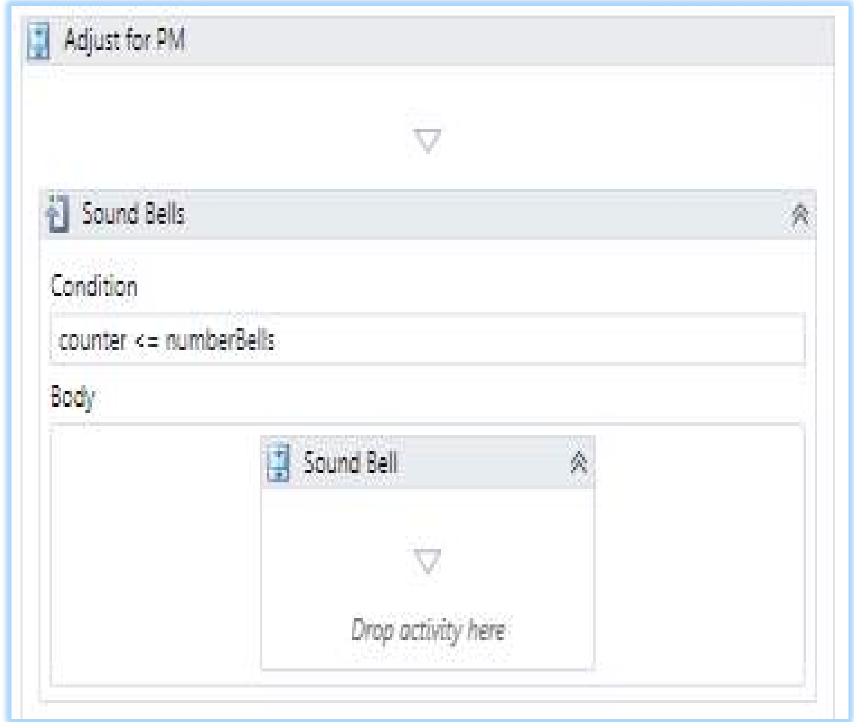
ნახ.16.19. While ქმედების დამატება Sound Bell დასახელებით

While ქმედებაში Body სექციის მოქმედება სრულდება მანამ, სანამ პირობა (Condition) ჭეშმარიტია. თავიდან პირობა მოწმდება და თუ ის არის „true", მაშინ ქმედებები სრულდება. ეს მეორდება მანამ, სანამ პირობა გახდება "false".

შენიშვნა: DoWhile ქმედება იდენტურია While ქმედების, ოღონდაც ჯერ აქტიურობა სრულდება ერთხელ და შემდეგ მოწმდება პირობა.

შევიტანოთ Condition (პირობა) **counter <= numberBells.**

გადმოვიტანოთ ინსტრუმენტების პანელიდან Sequence ქმედება Body-სექციაში. მისი DisplayName შევცვალოთ Sound Bell-ით. მივიღებთ 16.20 ნახაზზე ნაჩვენებ სურათს.



ნახ.16.20. While ქმედება შეიცავს Sequence-ს

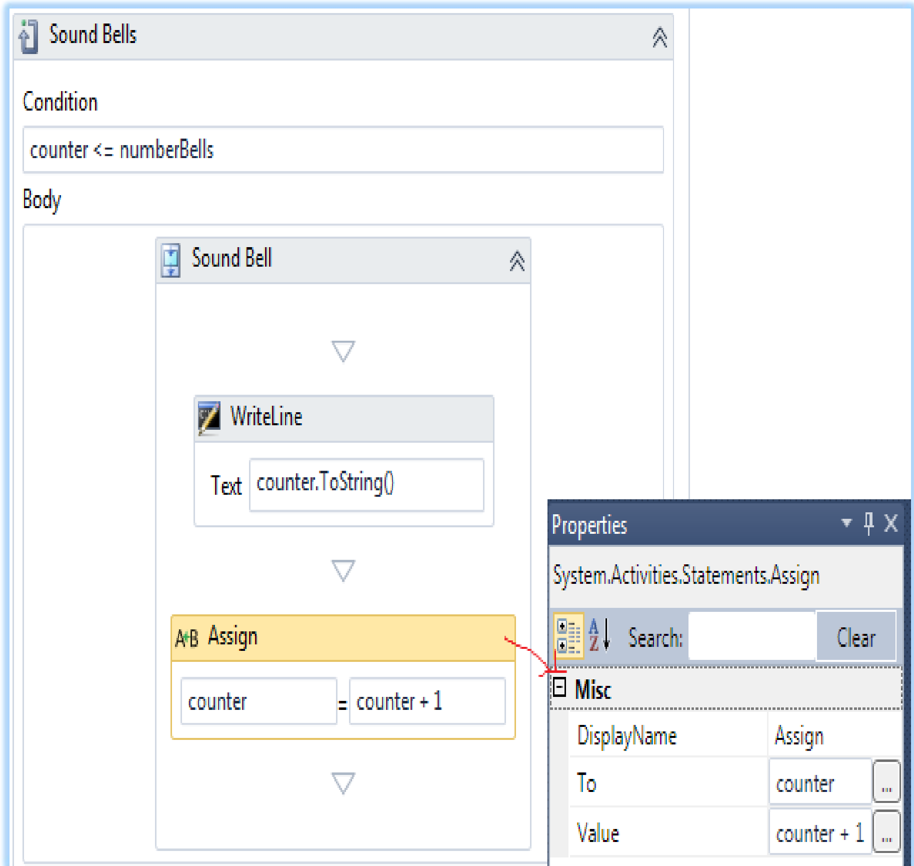
- **Sequence - პროცედურა**

გადმოვიტანოთ სამი სახის ქმედება Sequence–ზე „Sound Bell“. ამ სავარჯიშოში ზარი არ დარეკავს, მაგრამ კონსოლის სტრიქონი გამოიტანს ზარის რეკვის რაოდენობას. გადმოვიტანოთ „Sound Bell“–ში WriteLine ქმედება. თვისებების Text-ში შევიტანოთ ბრძანება:

counter.ToString()

იგი გამოიტანს კონსოლზე მთვლელის მიმდინარე მნიშვნელობას.

გადმოვიტანოთ Assign ქმედება WriteLine ქმედების ქვემოთ. To თვისებისათვის შევიტანოთ counter; Value თვისებისთვის კი counter+1. ესაა მარტივი ინკრემენტის მაგალითი (ნახ.16.21).



ნახ.16.21. საბოლოო Sequence დიაგრამის კომპლექტი

სისტემის ამუშავებისთანავე კონსოლზე გამოჩნდება ასეთი შედეგი (ნახ.16.22).

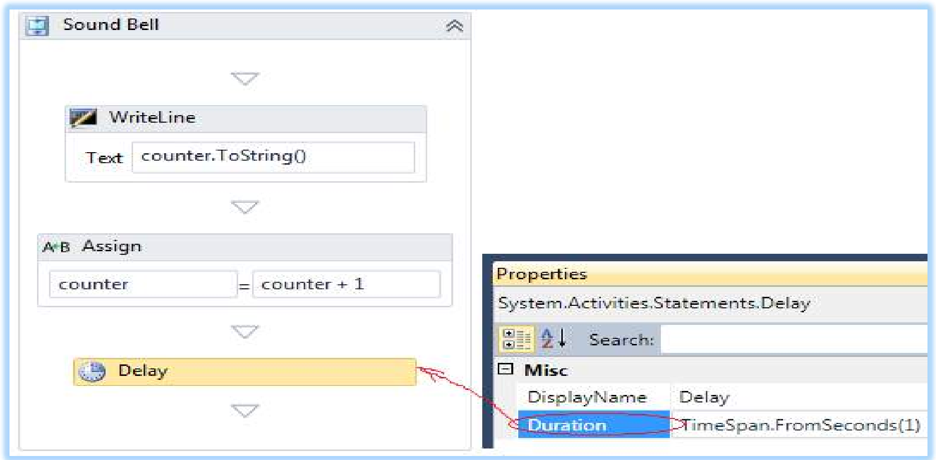
ნახ.16.22. შედეგი

- Delay ქმედება

გადმოვიტანოთ ინსტრუმენტების პანელიდან დაყოვნების Delay ქმედება Assign ქმედების ქვემოთ. დაყოვნების აქტიურობა განსაზღვრავს პაუზის ხანგრძლივობას. იგი მიეთითება როგორც TimeSpan კლასი. შევიტანოთ შემდეგი გამოსახულება: (ნახ.16.23).

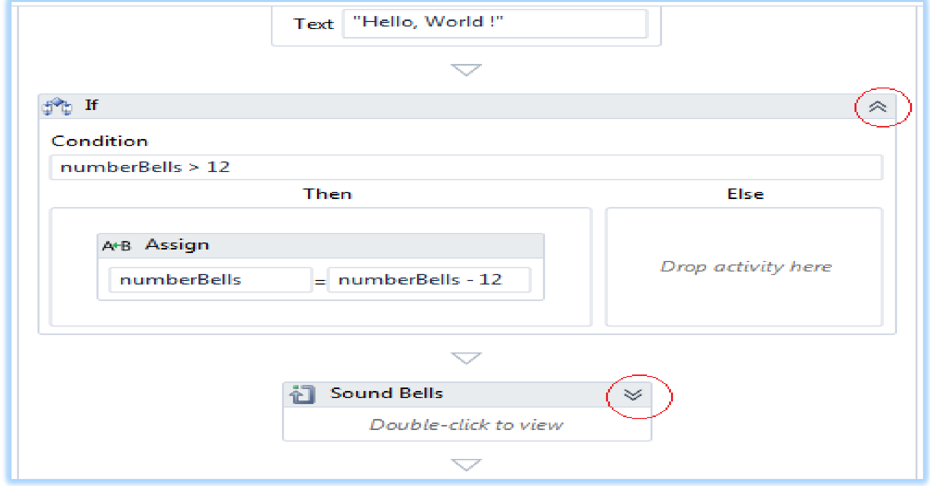
TimeSpan.FromSeconds(1)

მივიღებთ 16.23 ნახაზზე მოცემულ სურათს.



ნახ.16.23. დაყოვნების ქმედების დამატება Sequence-ში

16.24 ნახაზზე რგოლებით შემოხაზულია ჩაკეცვა–გაშლის ისრები. Sound Bell-სთვის (ნახ.16.23) ის ჩაკეცილია და ჩანს მხოლოდ სათაური.



ნახ.16.24

თუ ავამუშავებთ ამ სიტუაციას, მივიღებთ 1.25 სურათს.

რიცხვები 1–11 გამოიტანება მიმდევრობით, ოდნავ დაყოვნებით.

ახლა გადმოვიტანოთ WriteLine ქმედება Sound Bell-ის ქვემოთ. შევცვალოთ DisplayName სახელით Display Time.

ნახ.16.25

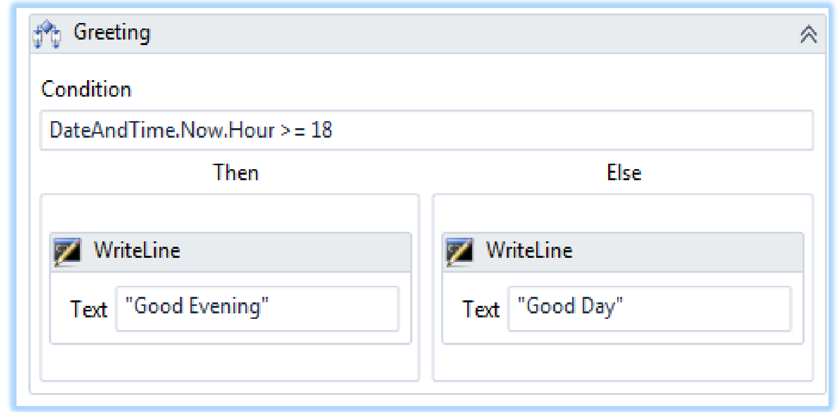
თვისებისთვის Text შევიტანოთ გამოსახულება:

„The time is: " + DateAndTime.Now.ToString()

გადმოვიტანოთ if ქმედება „Display Time”–ს ქვემოთ და DisplayName შევცვალოთ Greeting–ით. Condition პირობისათვის შევიტანოთ გამოსახულება:

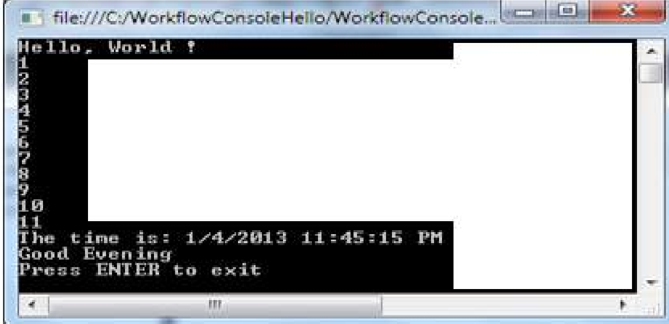
DateAndTime.Now.Hour >= 18

გადმოვიტანოთ WriteLine ქმედება ორივე Then და Else სექციებში. Then სექციისთვის Text-ში შევიტანოთ „Good Evening”; Else სექციისათვის Text-ში შევიტანოთ „Good Day”. ამგვარად, „Greeting“ აქტიურობა მიიღებს ასეთ სახეს (ნახ.16.26).



ნახ.16.26. Greeting ქმედება

ავამუშავოთ აპლიკაცია F5-ით.

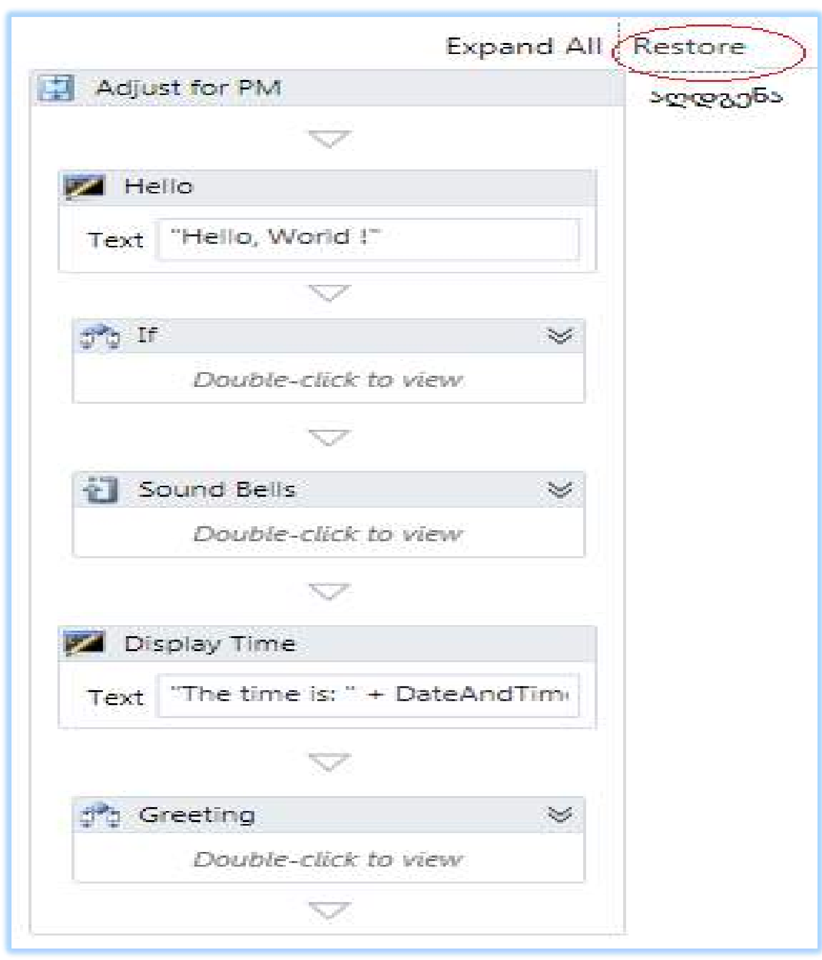


ნახ.16. 27. შედეგი

16.3. დიზაინერის მართვა და XAML კოდი

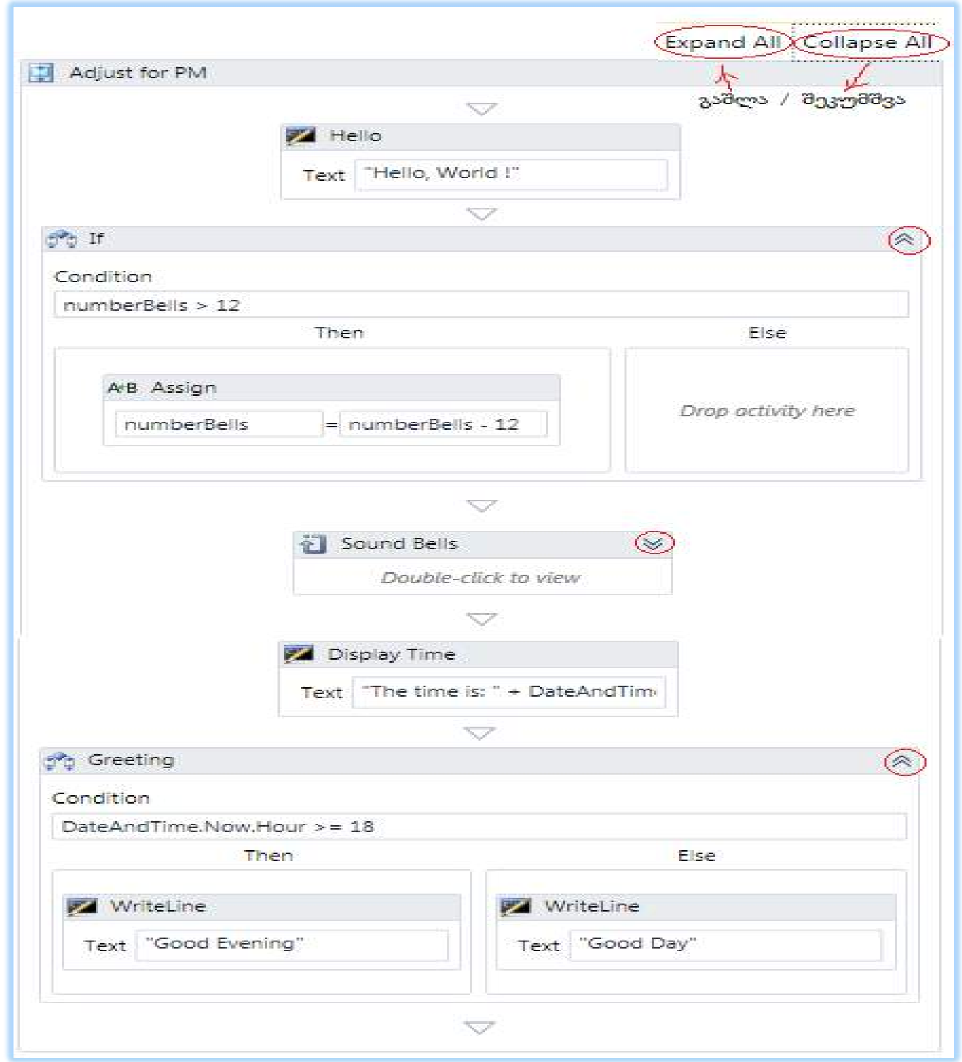
განვიხილოთ Visual Studio.NET პაკეტის WF-ის დიზაინერის სამუშაო გარემოში დიდი პროექტების და სქემების მართვის საშუალებები (Navigating the Designer).

მარტივი მაგალითიდან დავინახეთ, რომ საკმაოდ რთულია სამუშაო პროცესების დიდი სქემების მართვა და მათი მთლიანობაში წარმოდგენა. დიდი პროექტების დროს ის კიდევ უფრო რთულია. ამისათვის დიზაინერს ეძლევა საშუალება, ჩაკეცოს სქემის სექციები, მათი მთლიანი სტრუქტურის დათვალიერებისა და მართვის მიზნით. მაგალითად, ჩვენი პროექტი ჩაკეცვის (Collapse All) შემდეგ ასე გამოიყურება (ნახ.16.28).



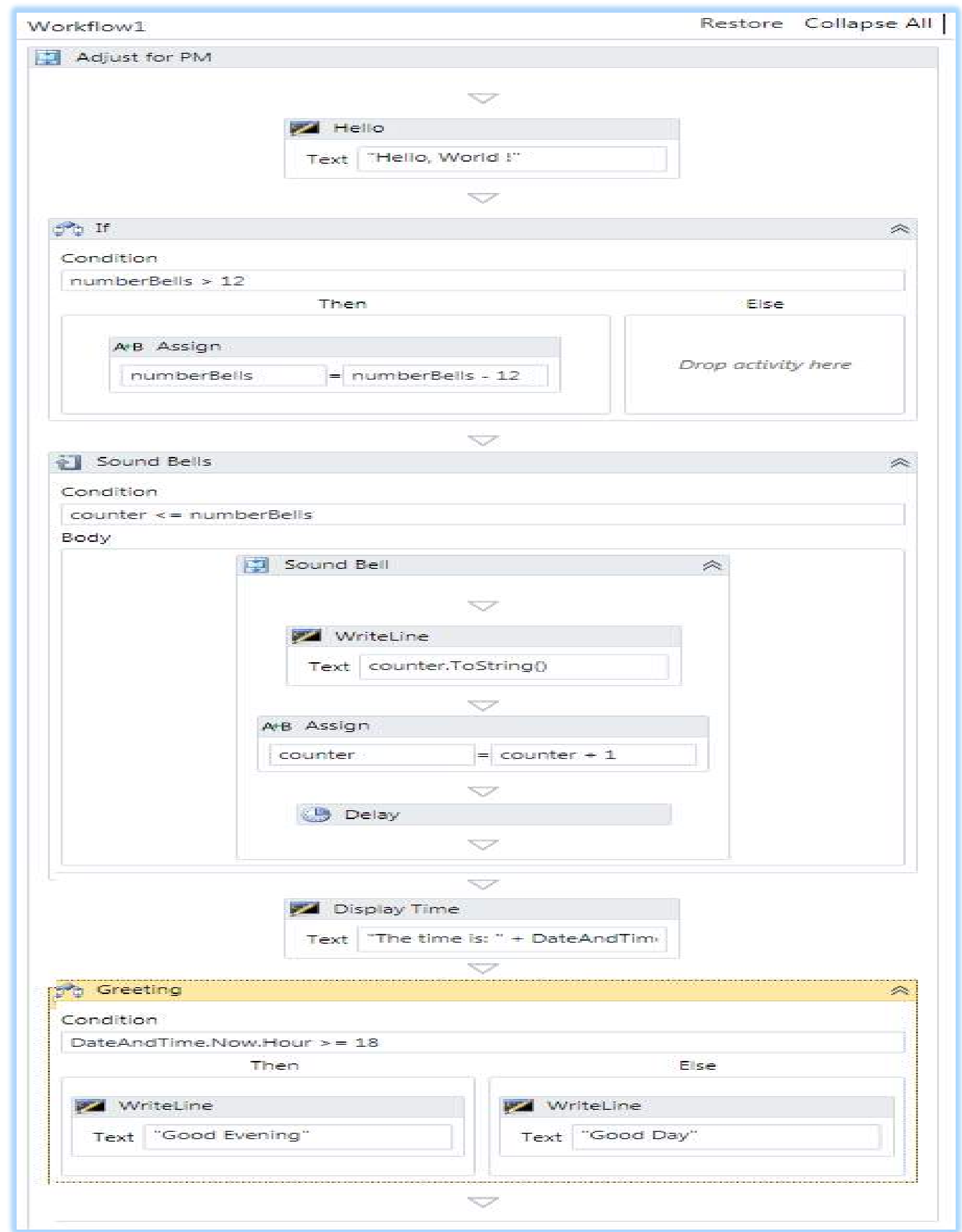
ნახ.16.28. შეკუმშული workflow დიაგრამა

თუ აღდგენის (Restore) ღილაკს გამოვიყენებთ, მაშინ მივიღებთ შეკუმშვამდე სურათს (ნახ.16.29).



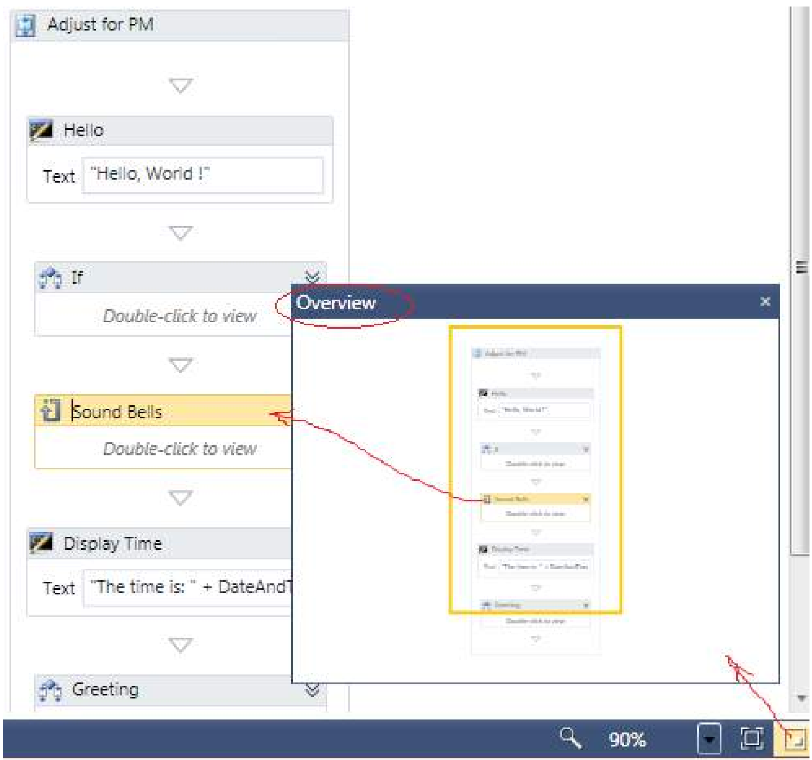
ნახ.16.29.ნაწილობრივ შეკუმშული workflow
დიაგრამა

აქ რამდენიმე ბლოკი გაშლილია მთლიანად, რამდენიმე კი ჩაკეცილია (მაგალითად, Bound Bells). Expand All ღილაკით კი გაიშლება ყველა ჩაკეცილი ბლოკი (ნახ.16.30).



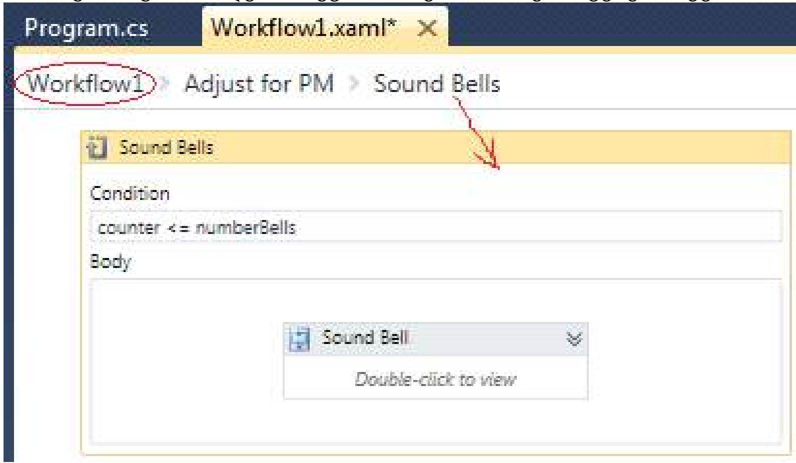
ნახ.16.30. მთლიანად გაშლილი workflow დიაგრამა

დიზაინერის ქვედა მარჯვენა კუთხეში არის Overview–ღილაკი, რომელიც ხსნის პროექტის მონიტორინგის ფანჯარას (ნახ.16.31). მოყვითალო ფერის ზოლი მიუთითებს აქტიურ ხილვად ბლოკზე. აქვეა ეკრანის ზომების ცვლილების (მასშტაბირების) ღილაკებიც.



ნახ.16.31. Overview - ღილაკი

„Sound Bell" ქმედების დაკლიკვით გამოჩნდება მხოლოდ ეს ბლოკი თავისი შვილობილი (ჩადგმული) კომპონენტებით (ნახ.16.32). აქ Window1 გადამრთველის არჩევით დიზაინერში გამოჩნდება ისევ 16.28–ე ნახაზზე ნაჩვენები სქემა.



ნახ.16.32. გამოყოფილი Sound Bell ბლოკი

- დიზაინერის XAML-კოდი

Solution Explorer-ში დავკლიკოთ Workflow1.xaml და დავათვალიეროთ შესაბამისი კოდი (ლისტინგი_16.2):

```
<! ---- ლისტინგი_16.2 -----    >
<Activity mc:Ignorable="" x:Class="WorkflowConsoleHello.Workflow1"
    xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:mv="clr-namespace:Microsoft.VisualBasic;assembly=System"
    xmlns:mva="clr-namespace:Microsoft.VisualBasic.Activities;assembly=
                                                            Sys tem.Activities"
    xmlns:s="clr-namespace:System;assembly=mscorlib"
    xmlns:s1="clr-namespace:System;assembly=System"
    xmlns:s2="clr-namespace:System;assembly=System.Xml"
    xmlns:s3="clr-namespace:System;assembly=System.Core"
    xmlns:s4="clr-namespace:System;assembly=System.ServiceModel"
    xmlns:sa="clr-namespace:System.Activities;assembly=System.Activities"
    xmlns:sad="clr-namespace:System.Activities.Debugger;assembly=
                                                          System.Activities"
xmlns:sap="http://schemas.microsoft.com/netfx/2009/xaml/activities/presentation"
    xmlns:scg="clr-namespace:System.Collections.Generic;assembly=System"
    xmlns:scg1="clr-namespace:System.Collections.Generic;assembly=
                                                        System.ServiceModel"
    xmlns:scg2="clr-namespace:System.Collections.Generic;assembly=System.Core"
    xmlns:scg3="clr-namespace:System.Collections.Generic;assembly=mscorlib"
    xmlns:sd="clr-namespace:System.Data;assembly=System.Data"
    xmlns:sl="clr-namespace:System.Linq;assembly=System.Core"
    xmlns:st="clr-namespace:System.Text;assembly=mscorlib"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <sap:WorkflowViewStateService.ViewState>
    <scg3:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="ShouldCollapseAll">True</x:Boolean>
      <x:Boolean x:Key="ShouldExpandAll">False</x:Boolean>
    </scg3:Dictionary>
  </sap:WorkflowViewStateService.ViewState>

  <Sequence DisplayName="Sequence"
```

```
sad:XamlDebuggerXmlReader.FileName="C:\WorkflowConsoleHello\WorkflowConsoleHello\Workflow1.
xaml"
   sap:VirtualizedContainerService.HintSize="233,564">
  <Sequence.Variables>
    <Variable x:TypeArguments="x:Int32" Default="1" Name="counter" />
    <Variable x:TypeArguments="x:Int32" Default="[DateAndTime.Now.Hour]" Name="numberBells" />
   </Sequence.Variables>
  <sap:WorkflowViewStateService.ViewState>
    <scg3:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="IsExpanded">True</x:Boolean>
    </scg3:Dictionary>
  </sap:WorkflowViewStateService.ViewState>

 <WriteLine DisplayName="Hello" sap:VirtualizedContainerService.HintSize="211,62"
                                                                      Text="Hello,  World !"
/>
   <If Condition="[numberBells &gt; 12]" sap:VirtualizedContainerService.HintSize="211,52">
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">True</x:Boolean>
        <x:Boolean x:Key="IsPinned">False</x:Boolean>
      </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <If.Then>

      <Assign sap:VirtualizedContainerService.HintSize="291,100">
        <Assign.To>
          <OutArgument x:TypeArguments="x:Int32">[numberBells]</OutArgument>
        </Assign.To>
        <Assign.Value>
          <InArgument x:TypeArguments="x:Int32">[numberBells - 12]</InArgument>
        </Assign.Value>
      </Assign>
    </If.Then>
   </If>

   <While DisplayName="Sound Bells" sap:VirtualizedContainerService.HintSize="211,52">
```

```
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">False</x:Boolean>
        <x:Boolean x:Key="IsPinned">False</x:Boolean>
      </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <While.Condition>[counter &lt;= numberBells]</While.Condition>
    <Sequence DisplayName="Sound Bell" sap:VirtualizedContainerService.HintSize="438,100">
      <sap:WorkflowViewStateService.ViewState>
        <scg3:Dictionary x:TypeArguments="x:String, x:Object">
          <x:Boolean x:Key="IsExpanded">False</x:Boolean>
          <x:Boolean x:Key="IsPinned">False</x:Boolean>
        </scg3:Dictionary>
      </sap:WorkflowViewStateService.ViewState>

   <WriteLine sap:VirtualizedContainerService.HintSize="242,62" Text="[counter.ToString()]" />
      <Assign sap:VirtualizedContainerService.HintSize="242,58">
        <Assign.To>
          <OutArgument x:TypeArguments="x:Int32">[counter]</OutArgument>
        </Assign.To>
        <Assign.Value>
          <InArgument x:TypeArguments="x:Int32">[counter + 1]</InArgument>
        </Assign.Value>
      </Assign>

 <Delay Duration="[TimeSpan.FromSeconds(1)]" sap:VirtualizedContainerService.HintSize="242,22" />
    </Sequence>
  </While>

<WriteLine DisplayName="Display Time" sap:VirtualizedContainerService.HintSize="211,62"
        Text="[&quot;The time is: &quot; + DateAndTime.Now.ToString()]" />

 <If Condition="[DateAndTime.Now.Hour &gt;= 18]" DisplayName="Greeting"
        sap:VirtualizedContainerService.HintSize="211,52">
    <If.Then>
      <WriteLine sap:VirtualizedContainerService.HintSize="219,100" Text="Good Evening" />
    </If.Then>
```

```
    <If.Else>
      <WriteLine sap:VirtualizedContainerService.HintSize="220,100" Text="Good Day" />
    </If.Else>
  </If>
 </Sequence>
</Activity>
```

16.4. კოდირებული სამუშაო პროცესები

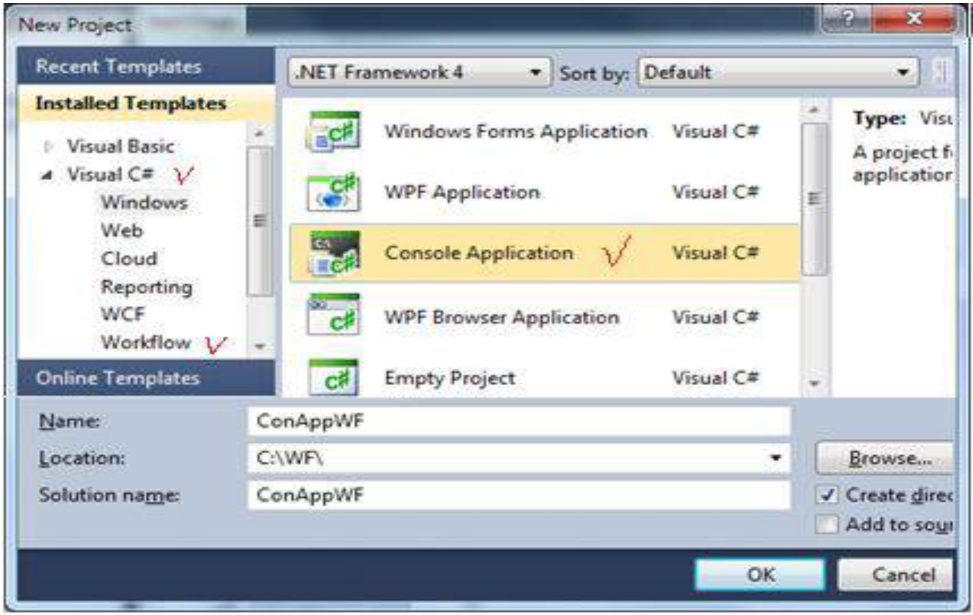
განვიხილოთ Visual Studio.NET პაკეტის WF-ის სამუშაო პროცესის შესრულება კოდის საშუალებით. ზემოთ ჩვენ ავაგეთ მარტივი სამუშაო პროცესი (workflow) დიზაინერის გამოყენებით. ახლა განვიხილოთ იგივე სამუშაო პროცესის შესრულება კოდის გამოყენებით.

ნებისმიერი სამუშაო პროცესი შეიძლება განხორციელდეს **კოდის ან დიზაინერის** გამოყენებით (შემსრულებლის გემოვნების საკითხია). კოდის გამოყენება საშუალებას იძლევა უფრო კარგად გავიგოთ, თუ როგორ ხდება სამუშაო პროცესის (workflow) რეალიზაცია და ფუნქციონირება.

- **კონსოლური აპლიკაციია აგება**

Visual Studio.NET გარემოში შევქმნათ ახალი კონსოლური აპლიკაცია სახელით ConAppWF (ნახ.16.33).

ნახ.16.33. კონსოლის აპლიკაციის შექმნა



დავამატოთ მიმართვა (reference) **System.Activities**. იგი საშუალებას იძლევა აპლიკაციაში გამოყენებულ იქნას სამუშაო პროცესის (workflow) ქმედებები (აქტიურობები).

შევცვალოთ Program.cs ფაილში სახელსივრცეები შემდეგი სახით (ლისტინგი_16.3).

```
// ------- ლისტინგი_16.3 --------
using System;
using System.Activities;
using System.Activities.Statements;
using System.Activities.Expressions;
namespace ConAppWF
{
    class Program
    {
        static void Main(string[] args)
        {
            WorkflowInvoker.Invoke(CreateWorkflow());
            Console.WriteLine("Press ENTER to exit");
            Console.ReadLine();
        }
    }
}
```

Main()–ის კოდი მსგავსია ჩვენ მიერ წინა მასალაში განხილული იგივე კოდისა. არის ერთი განსხვავება, რომ

```
//აქ CreateWorkflow()–ია
WorkflowInvoker.Invoke(CreateWorkflow());

// აქ new Workflow1() –ია
WorkflowInvoker.Invoke(new Workflow1());
```

Workflow1 იყო განსაზღვრული Workflow1.xaml ფაილში, რომელიც შეიქმნა Workflow Designer-ით. CreateWorkflow () კი არის მეთოდი, რომლის რეალიზაცია ახლა უნდა მოვახდინოთ.

- **სამუშაო პროცესის (workflow) განსაზღვრა**

როგორც ზემოთ აღვნიშნეთ, სამუშაო პროცესი არის ჩადგმული კლასების და მათი თვისებების ერთობლიობა. განვიხილოთ ეს საკითხი კოდის გამოყენებით. დავამატოთ Program.cs ფაილში შემდეგი მეთოდი (ლისტინგი_16.4).

```
// ---- ლისტინგი_16.4 ----------------
static Activity CreateWorkflow()
 {
    Variable<int> numberBells = new Variable<int>()
     {
       Name = "numberBells",
       Default = DateTime.Now.Hour
     };
   Variable<int> counter = new Variable<int>()
     {
        Name = "counter",
        Default = 1
      };
  return new Sequence()
    {
    };
}
```

CreateWorkflow() მეთოდი პირველად ქმნის ორ Variable<T> კლასის შაბლონს ტიპით int, სახელით numberBells და counter. ეს ცვლადები გამოიყენება სხვადასხვა ქმედებაში. ეს მეთოდი გამოიყნება იმ აქტიურობაზე დასაბრუნებლად, რომელსაც ელოდება WorkflowInvoker კლასი. იგი ფაქტობრივად აბრუნებს Sequence კლასის ანონიმურ ეგზემპლარზე.

Activity კლასი არის საბაზო, რომლისგანაც იწარმოება სამუშაო პროცესის ყველა ქმედება, ასევე Sequence–ც. ასე რომ კომპილატორი აბრუნებს Sequence ეგზემპლარს და Activity–ის როგორც მის საბაზო კლასს

- **რეალიზაცია 1–ელ დონეზე**

ამგვარად, ჩვენ განვსაზღვრეთ ცარიელი Sequence ქმედება. ეს ეკვივალენტურია ახალი სამუშაო პროცესის შექმნის, რომელსაც აქვს Sequence, ქმედებების გარეშე. ახლა განვსაზღვროთ ქმედებები ამ Sequence–ზე return new Sequence() გამოძახებით და შეცვლით 16.5 ლისტინგზე მოცემული კოდით.

```
// ---- ლისტინგი_16.5 -------------
return new Sequence()
{   DisplayName = "Main Sequence",
    Variables = { numberBells, counter },
    Activities =
       {
```

```
        new WriteLine()
          {
            DisplayName = "Hello",
            Text = "Hello, World!"
           },
        new If()
          {
            DisplayName = "Adjust for PM"
            // Code to be added here in Level 2
          },
        new While()
          {
            DisplayName = "Sound Bells"
            // Code to be added here in Level 2
          },
        new WriteLine()
          {
              DisplayName = "Display Time",
              Text = "The time is: " + DateTime.Now.ToString()
          },
         new If()
           {
              DisplayName = "Greeting"
              // Code to be added here in Level 2
            }
      }
  };
```

ეს კოდი თავიდან განსაზღვრავს DisplayName და ობიექტის დამოკიდებულ ცვლადებს ამ ქმედებასთან. შემდეგ იგი აინიციალიზებს წევრქმედებებს, როგორც ქმედებათა კოლექციას. კერძოდ, იგი ქმნის 16.1 ცხრილში მოცემულ ქმედებებს.

ცხრ.16.1

Type	DisplayName
WriteLine	“Hello”
If	“Adjust for PM”
While	“Sound Bells”
WriteLine	“Display Time”
If	“Greeting”

WriteLine ქმედებებისთვის განსაზღვრულია Text თვისებები. სხვა ქმედებებისთვის რეალიზაცია განსაზღვრულ იქნება შემდეგ დონეზე.

- **რეალიზაცია მე–2 დონეზე**

პირველი If ქმედებისთვის შევიტანოთ შემდეგი კოდი:

```
DisplayName = "Adjust for PM",
    // მე–2 დონეზე დასამატებელი კოდი
    Condition = ExpressionServices.Convert<bool>
        (env => numberBells.Get(env) > 12),
    Then = new Assign<int>()
     {
        DisplayName = "Adjust Bells"
       // მე–3 დონეზე დასამატებელი კოდი
```

ეს კოდი განსაზღვრავს მდგომარეობას (Condition) და Then–ის თვისებებს (აქ არაა Else ნაწილი). Assign ქმედება იქნება რეალიზებული მომდევნო დონეზე. Condition თვისების განსაზღვრა საჭიროებს მცირე კომენტარს.

- **გამოსახულებების შეტანა**

ExpressionServices კლასის სტატიკური Convert<T>() მეთოდი გამოიყენება InArgument <T> კლასის შესაქმნელად, რასაც ელოდება მდგომარეობის (Condition) თვისება. ეს კლასები და მეთოდები იყენებს საერთო (T) ტიპს, ამიტომაც ისინი შეიძლება ნებისმიერი ტიპისათვის იქნას გამოყენებული. ამ შემთხვევაში ჩვენ უნდა გამოვიყენოთ BOOL ტიპი, ვინაიდან If ქმედების პირობის თვისება ელოდება true-ს ან false-ს.

გამოსახულების შეტანა რეალიზდება ლამბდა–გამოსახულებით (LINQ სინტაქსის ანალოგიურად), მონაცემების ამოსაღებად სამუშაო პროცესის გარემოდან (workflow environment). ლამბდა გამოსახულებაში „ => “ –ს უწოდებენ ლამბდა ოპერატორს. მის მარცხნივ თავსდება შემავალი პარამეტრები, ხოლო მარჯვენა მხარეს განისაზღვრება ფაქტობრივი გამოსახულება. ENV–ის მნიშვნელობა მიიღება შესრულების დროს, როცა ის ცდილობს მდგომარეობის (Condition) შეფასებას.

ფაქტობრივად, სამუშაო პროცესი მდგომარეობის გარეშეა, იგი არ ინახავს ელემენტთა მონაცემებს. კლასის ცვლადები განსაზღვრულია მარტივი მონაცემებით. იმისათვის, რომ მივიღოთ ფაქტობრივი მონაცემები კლასის ცვლადებიდან, უნდა გამოვიყენოთ საკუთარი Get () მეთოდი. ის მოითხოვს სორტის მარკერს, რომელიც არის ActivityContext კლასი. ეს საჭიროა, რათა განვასხვავოთ სამუშაო პროცესის კონკრეტული ეგზემპლარი სხვებისაგან, რომლებიც შეიძლება ერთდროულად იქნას გაშვებული. Get (env)–ით დაბრუნებული მნიშვნელობა შეუდარდება, არის თუა არა ის მეტი 12–ზე.

შევიტანოთ შემდეგი კოდი While ქმედებისთვის:

```
DisplayName = "Sound Bells",
// მე–2 დონეზე დასამატებელი კოდი
Condition = ExpressionServices.Convert<bool>
      (env => counter.Get(env) <= numberBells.Get(env)),
Body = new Sequence()
 {
      DisplayName = "Sound Bell"
      // მე–3 დონეზე დასამატებელი კოდი
 }
```

While ქმედების Condition თვისება იდენტურია If ქმედების. ის იყენებს აგრეთვე ExpressionServices კლასს InArgument<T> კლასის შესაქმნელად. აგრეთვე bool ტიპს. შემთხვევაში ის აფასებს, არის თუ არა count <= numberBells. ამ ორივე ცვლადისთვის იგი იყენებს Get(env) მეთოდს ფაქტობრივი მნიშვნელობის მისაღებად.

მეორე If ქმედებისთვის (სახელით “Greeting”) შევიტანოთ შემდეგი კოდი:

```
DisplayName = "Greeting",
// მე–2 დონეზე დასამატებელი კოდი
Condition = ExpressionServices.Convert<bool>
    (env => DateTime.Now.Hour >= 18),
  Then = new WriteLine() { Text = "Good Evening" },
  Else = new WriteLine() { Text = "Good Day" }
```

ამ პირობისათვის (Condition) შემავალი env–პარამეტრი არ გამოიყენება, მაგრამ იგი მაინც უნდა გამოცხადდეს გამოსახულებაში. ლოგიკა იყენებს მიმდინარე დროს, რათა დავინახოთ, არის თუ არა ის 6:00 PM. ორივე Thet და Else თვისებისათვის იქმნება WriteLine ქმედება. ერთი ამბობს „საღამო მშვიდობისა“ (Good Evening), მეორე კი – „გამარჯობათ“ (Good Day).

- **რეალიზაცია მე–3 დონეზე**

პირველი If ქმედებისთვის (სახელით “Adjust for PM”), შექმნილია ცარიელი Assign ქმედება Then თვისებაში. შევიტანოთ შემდეგი ტექსტი ამ რეალიზაციაში:

```
DisplayName = "Adjust Bells",
// მე–3 დონეზე დასამატებელი კოდი
To = new OutArgument<int>(numberBells),
Value = new InArgument<int>(env => numberBells.Get(env) - 12)
```

- **Assign ქმედება (დანიშვნა, მინიჭება)**

Assign კლასი არის უნივერსალური (ზოგადი), ამიტომაც მას შეუძლია მონაცემთა ნებისმიერი ტიპის მხარდაჭერა. ამ შემთხვევაში ის ანიჭებს მთელ მნიშვნელობას, ამიტომ შექმნილია როგორც Assign<int>. თვისებები To და Value აგრეთვე იყენებს შაბლონურ კლასებს და უნდა შეიქმნას იმავე ტიპით (<int>).

To თვისება არის OutArgument კლასის, რომელიც იღებს კლასის ცვლადს კონსტრუქტორიდან. Value თვისება იყენებს InArgument კლასს. ის გამოყენებულ იქნა აქამდე Condition თვისების If და While -ში. მისი კონსტრუქტორისათვის გამოიყენება ლამბდა გამოსახულება, როგორც ეს იყო Condition თვისებისათვის.

- **Sequence ქმედება**

While ქმედებაში Execute თვისებისთვის შევქმენით ცარიელი Sequence. ის განსაზღვრავს შესასრულებელ ქმედებათა მიმდევრობას ციკლის შესრულების ყოველი მომენტისათვის. შევიტანოთ შემდეგი ტექსტი Activities თვისების შესავსებად:

```
DisplayName = "Sound Bell",
// მე–3 დონეზე დასამატებელი კოდი
Activities =
 {
  new WriteLine()
    {
      Text = new InArgument<string>(env => counter.Get(env).ToString())
    },
  new Assign<int>()
    {
      DisplayName = "Increment Counter",
      To = new OutArgument<int>(counter),
      Value = new InArgument<int>(env => counter.Get(env) + 1)
    },
   new Delay()
     {
       Duration = TimeSpan.FromSeconds(1)
     }
 }
```

ეს კოდი ამატებს სამ ქმედებას Sequence-ში:

- WriteLine აქტიურობას counter–ის გამოსატანად ეკრანზე;
- Assign აქტიურობას counter–ის ინკრემენტისათვის;
- Delay აქტიურობას მცირე პაუზისათვის იტერაციებს შორის.

ამ WriteLine ქმედებისთვის Text–ის თვისება არაა სიმბოლური string, როგორც სხვა პირობა იყო. ამ შემთხვევაში ეკრანზე გამოსატანი მნიშვნელობა განსაზღვრულია გამოსახულებაში.
Text–ის თვისება ელოდება string-ს, ამიტომაც იგი ქმნის InArgument<string> კლასს. ჩვენ ამ დროს ვიყენებთ ლამბდა გამოსახულებას. Get(env) მეთოდი Variable კლასისათვის უზრუნველყოფს მიმდინარე ცვლადისთვის მთელ ტიპს (integer). ToString() მეთოდი გარდაქმნის მას სტრიქონად.

Delay ქმედებისთვის ხანგრძლივობის (Duration) თვისება გადაეცემა როგორც TimeSpan კლასი, რომელიც შექმნილია FromSeconds() სტატიკური მეთოდით.

- **Running the Application (ამუშავება)**

F5 ღილაკით ავამუშავოთ აპლიკაცია. დღე-ღამის დროისგან დამოკიდებულებით შედეგები ასეთი იქნება:

```
Hello, World!
1
2
3
4
5
6
7
The time is: 10/5/2009 7:02:41 PM
Good Evening
    Press ENTER to exit
```

რეალიზაციის პროგრამის სრული ტექსტი მოცემულია 16.10 ლისტინგში.

```
// ---- ლისტინგი_16.10 --------------------
using System;
using System.Activities;
using System.Activities.Statements;
using System.Activities.Expressions;
namespace Chapter02
{
  class Program
   {
    static void Main(string[] args)
     {
       WorkflowInvoker.Invoke(CreateWorkflow());
```

```
    Console.WriteLine("Press ENTER to exit");
    Console.ReadLine();
  }
static Activity CreateWorkflow()
 {
  Variable<int> numberBells = new Variable<int>()
    {
      Name = "numberBells",
      Default = DateTime.Now.Hour
    };
  Variable<int> counter = new Variable<int>()
    {
      Name = "counter",
      Default = 1
    };
  return new Sequence()
    {
      DisplayName = "Main Sequence",
      Variables = { numberBells, counter },
      Activities =
         {  new WriteLine()
              {  DisplayName = "Hello",
                 Text = "Hello, World!"
               },
           new If()
             {  DisplayName = "Adjust for PM",
               // Code to be added here in Level 2
               Condition = ExpressionServices.Convert<bool>
               (env => numberBells.Get(env) > 12),
               Then = new Assign<int>()
                 {  DisplayName = "Adjust Bells",
                    // Code to be added here in Level 3
                    To = new OutArgument<int>(numberBells),
                    Value = new InArgument<int>
                    (env => numberBells.Get(env) - 12)
                  }
               },
```

```
        new While()
          {  DisplayName = "Sound Bells",
            // Code to be added here in Level 2
            Condition = ExpressionServices.Convert<bool>
            (env => counter.Get(env) <= numberBells.Get(env)),
            Body = new Sequence()
              { DisplayName = "Sound Bell",
                // Code to be added here in Level 3
                Activities =
                  { new WriteLine()
                      { Text = new InArgument<string>
                        (env => counter.Get(env).ToString())
                      },
        new Assign<int>()
          {
            DisplayName = "Increment Counter",
            To = new OutArgument<int>(counter),
            Value = new InArgument<int>
            (env => counter.Get(env) + 1)
          },
        new Delay()
          {
            Duration = TimeSpan.FromSeconds(1)
          }
        }
      }
    },
new WriteLine()
  {
      DisplayName = "Display Time",
      Text = "The time is: " + DateTime.Now.ToString()
  },
new If()
  {
      DisplayName = "Greeting",
      // Code to be added here in Level 2
      Condition = ExpressionServices.Convert<bool>
```

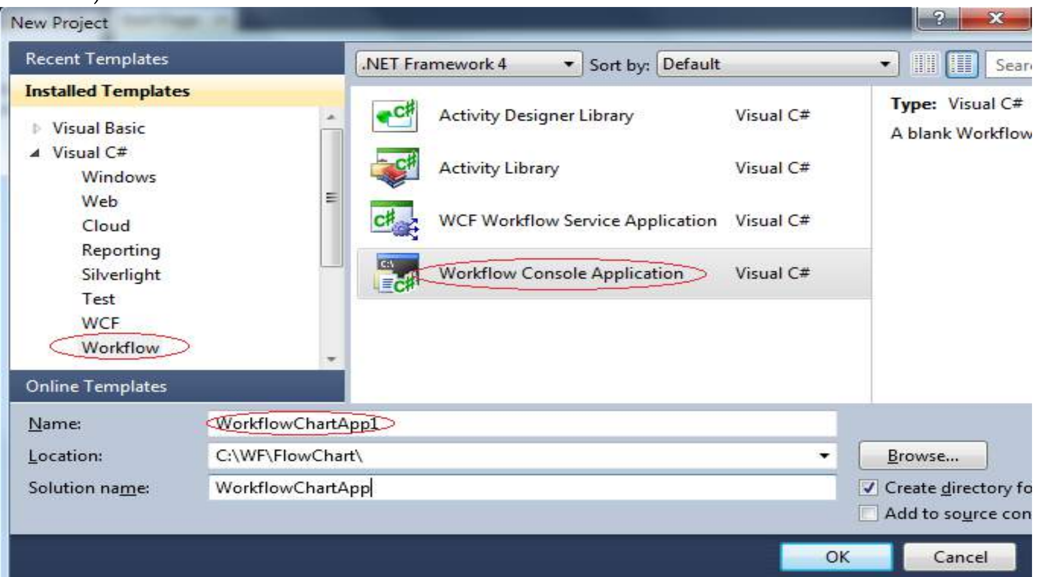
```
            (env => DateTime.Now.Hour >= 16),
            Then = new WriteLine() { Text = "Good Evening" },
            Else = new WriteLine() { Text = "Good Day" }
        }
    }
  };
}}}
```

16.5. ბიზნესპროცესის დიაგრამა (Flowchart Workflow)

ამჯერად უნდა ავაგოთ ბიზნესპროცესი, რომელიც გამოიყენებს აქტიურობათა დიაგრამას. როგორც სათაურიდან ჩანს, ქმედებათა დიაგრამა მუშაობს როგორც ბლოკ–სქემა, ქმედებათა სახეები დაკავშირებულია ერთმანეთთან გადაწყვეტილების ხეებით (decision trees). ქმედებათა მიმდევრობითობის გამოყენებით, შვილი–პროცესები სრულდება მიმდევრობით ზემოდან–ქვევით (top-down). ამისდა მიუხედავად, შვილი–ქმედებები შეიძლება შესრულდეს ნებისმიერი მიმდევრობით, გადაწყვეტილების მიმღები პირის მიერ.

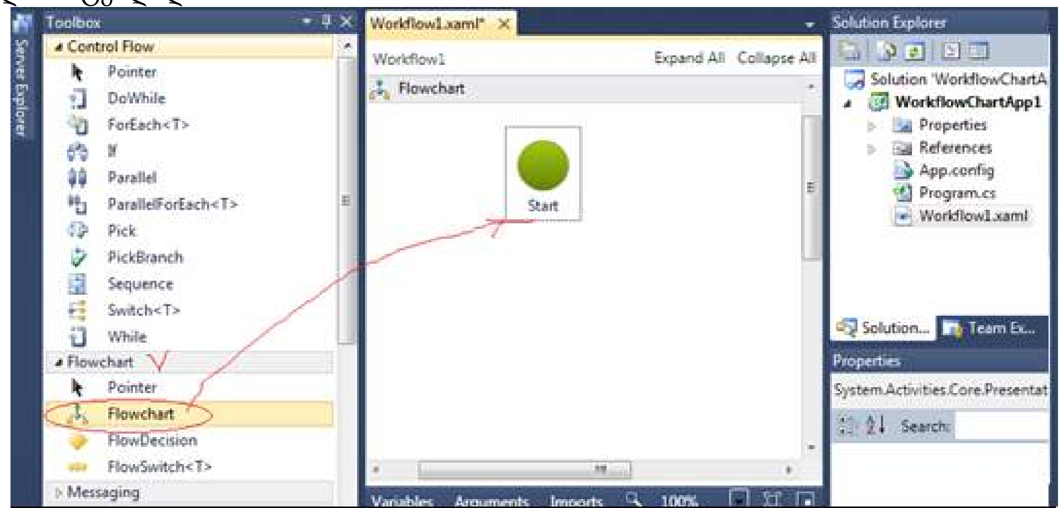
- **ბიზნესპროცესის დიაგრამის შექმნა**

შევქმნათ ახალი პროექტი (solution), ავირჩიოთ workflow და Console Application (ნახ.16.34).



ნახ.16.34

ინსტრუმენტების პანელიდან Flowchart გადმოვიტანოთ ფორმაზე Flowchart-ქმედება. საწყისი სამუშაო პროცესის დიაგრამა სასტარტო მწვანე წრით მოცემულია 16.35 ნახაზზე. ცარიელი სივრცე მის ქვეშ გამოიყენება ასაგები ბიზნესპროცესის ქმედებების დასამატებლად.



ნახ.16.35

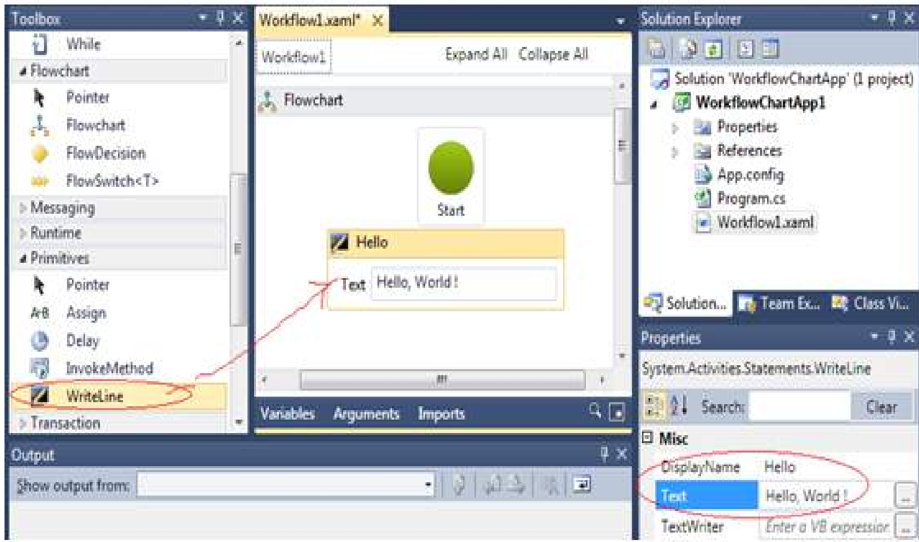
ძირითადი განსხვავება Flowchart ქმედებასა და Sequence ქმედებას შორის ისაა, თუ როგორაა დაკავშირებული შვილი–აქტიურობები. როგორც 1-ელი თავიდან ვიცით, ქმედებების დამატებისას მიმდევრობითობაში ისინი სრულდება დაღმავალი (top-down) მიმდევრობით. შესაძლებელია მისი კონტროლი (მართვა), ქმედებათა გადაადგილებით, ოღონდაც ისინი ყოველთვის გასწორებულია ვერტიკალში და განაწილებულია თანაბრად. ისრები ქმედებებს შორის კი აგებულია ავტომატურად.

Flowchart აქტიურობით შესაძლებელია ქმედებათა განთავსება პალიტრის ნებისმიერ ადგილას. მნიშვნელოვანია ისიც, რომ აქ ისრები ხელით უნდა გაკეთდეს და შეერთება დასაშვებია უკან, წინა ქმედებასთანაც!

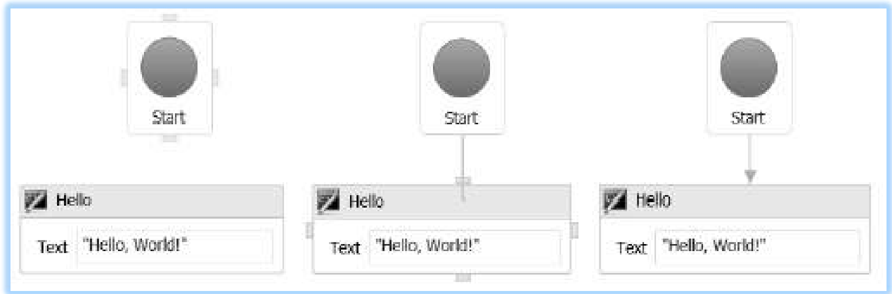
ჩვენ აპლიკაციაში დისპლეიზე უნდა გამოვიტანოთ მისალმებები, შესაბამისად დღე-ღამის დროისა. დავიწყოთ სტანდარტული მისალმების ასახვით „Hello, World". ამისათვის გადმოვიტანოთ WriteLine ქმედება ინსტრუმენტების პანელიდან მწვანე წრის ქვეშ. დავაყენოთ DisplayName-ში "Hello" და Text-ში "Hello, World !" (ნახ.16.36).

- **მიერთების განსაზღვრა**

მაუსით მოვნიშნოთ სასტარტო მწვანე წრე, გავატაროთ კავშირი და ავტომატურად შეიქმნება ისარი ორ ქმედებას შორის (ნახ.16.37).



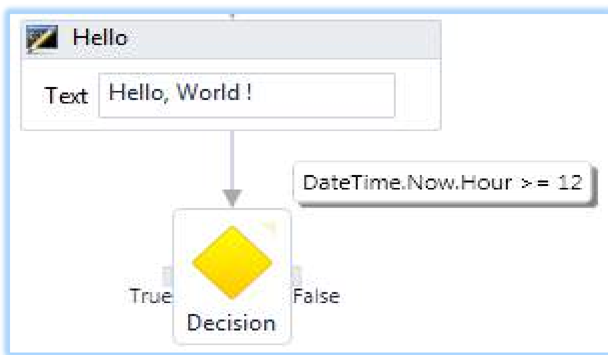
ნახ.16.36



ნახ.16.37. მიერთების საწყისი და ბოლო წერტილები

- გადაწყვეტილების ნაკადი (FlowDecision)

გადმოვიტანოთ სქემაზე FlowDecision ქმედება "Hello" ქმედების შემდეგ. FlowDecision ქმედება გამოიყურება ყვითელი ალმასის სახით, როგორც განშტოების სიმბოლო ნორმალურ ბლოკ–სქემაში, თვისებათა ფანჯარაში შევიტანოთ მდგომარეობა (Condition) DateTime.Now.Hour> = 12. მაუსის კურსორის მიტანით FlowDecision ქმედებაზე გამოჩნდება ასეთი სურათი (ნახ.16.38).



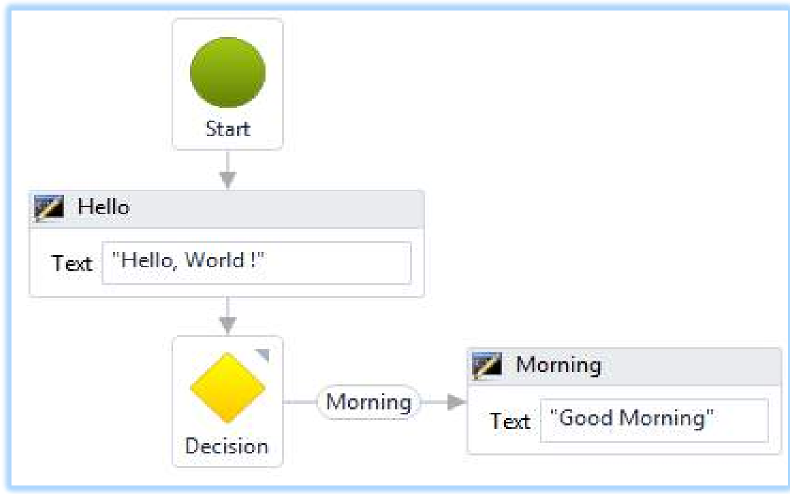
ნახ.16.38

ქმედებას აქვს მარცხნიდან მიერთების წერტილი true შტოსთვის, ხოლო მარჯვნიდან – false შტოსთვის. ზედა მარჯვენა კუთხეში პატარა სამკუთხედით ჩაირთვება პირობის გამოსახულების მუდმივად გამოსაჩენი თვისება (უმაუსოდ).

შეიძლება ტექსტის შეცვლა true/false შტოებისთვის. მაგალითად, FalseLabel–თვის შევიტანოთ Morning, და TrueLabel-თვის Afternoon. მიიღება სურათი:

ნახ.16.39

შევაერთოთ FlowDecision ქმედება მარჯვნიდან WriteLine ქმედებას, რომელშიც ჩაწერილი გვაქვს Morning/”Good Morning” (ნახ.16.40).



ნახ.16.40

FlowDecision ქმედებას არ აქვს DisplayName თვისება, მაგრამ შეუძლია პირობის ჩვენება და true/false შტოების შემოწმება და კორექტირება. ამგვარად ამ აქტიურობის მიზანი და ფუნქციურობა ცალსახად ცხადია.

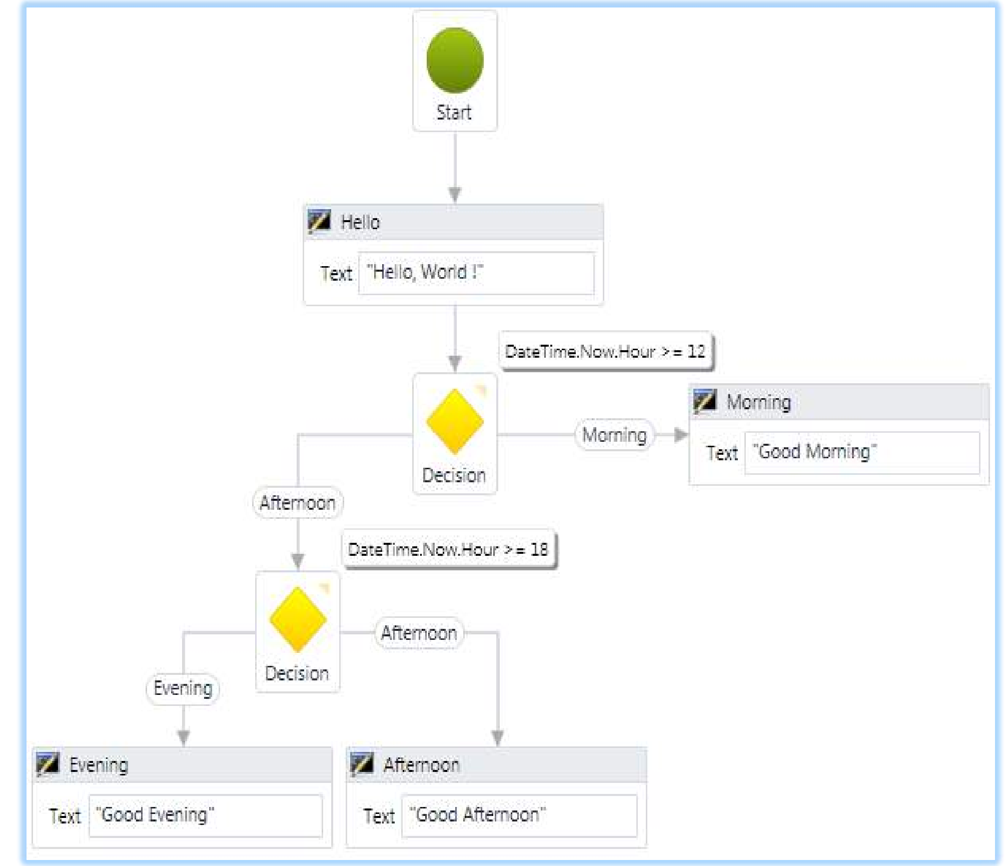
დავამატოთ FlowDecision ქმედების მარცხენა true შტოს ახალი FlowDecision ქმედება, პირობის გამოსახულებით DateTime.Now.Hour >= 18. მისი მარცხენა TrueLabel შტო იყოს Evening, ხოლო მარჯვენა – ისევ Afternoon. ორივეს მივუერთოთ ახალი WriteLine ქმედებები: Evening და Afternoon. 16.41 ნახაზზე მოცემულია ეს სურათი.

სანამ ავამუშავებთ მიღებულ აპლიკაციას, გავხსნათ Program.cs ფაილი. ჩავამატოთ აქ შემდეგი კოდი WorkflowInvoker კლასის გამოძახების შემდეგ. ის დაგვანახვებს შედეგებს პროგრამიდან გამოსვლამდე.

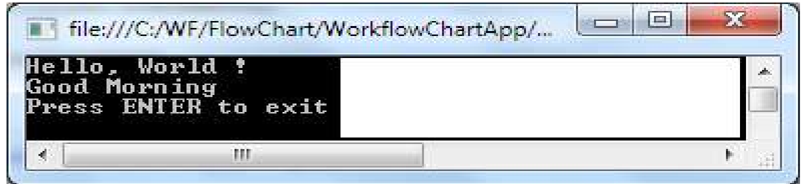
```
Console.WriteLine("Press ENTER to exit");
Console.ReadLine();
```

აპლიკაციის ამუშავება: F5 (ნახ.16.42)

შედეგი დამოკიდებულია იმაზე, დღის რომელ მონაკვეთში ავამუშავებთ აპლიკაციას.



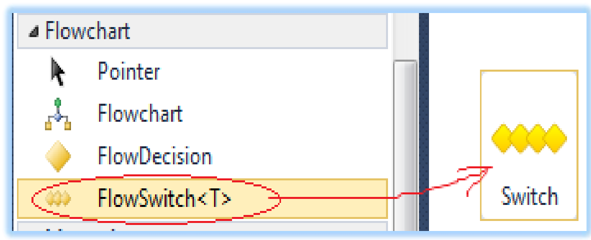
ნახ.16.41



ნახ.16.42. შედეგები

- **პროცესის გადამრთველი (Flow Switch)**

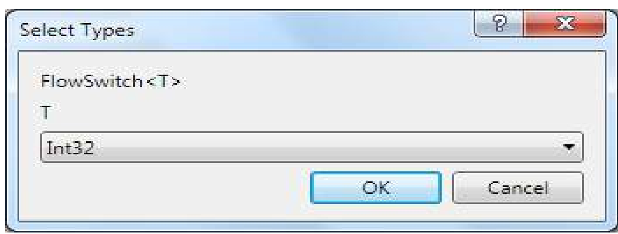
FlowSwitch ქმედება ფუნქციონირებს როგორც FlowDecision იმ გამონაკლისით, რომ არაა შეზღუდვა მხოლდ true/false ორი განშტოებით. შესაძლებელია შტოების ნებისმიერი რაოდენობის განსაზღვრა ისე, როგორც ეს იყო C# ენაში. 16.43 ნახაზზე ნაჩვენებია FlowSwitch აქტიურობის პიქტოგრამა ინსტრუმენტების პანელზე.



ნახ.16.43. პიქტოგრამა

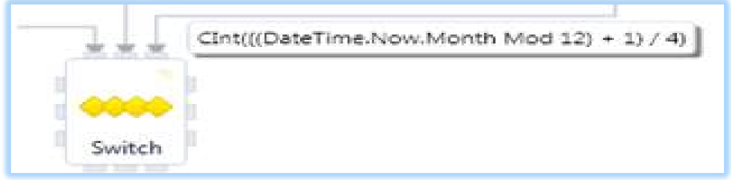
- **FlowSwitch ქმედების დამატება**

გადმოვიტანოთ ინსტრუმენტების პანელიდან FlowSwitch აქტიურობა ჩვენი ბოლო პროექტის სქემის ქვეშ. იგი არის <T> კლასის შაბლონი და უნდა მიეთითოს ტიპი. ჩვენ შემთხვევაში ესაა Int32, რომელიც ავტომატურადაა განსაზღვრული. ავირჩევთ OK-ს (ნახ.16.44).



ნახ.16.44. ტიპის განსაზღვრა

შევაერთოთ სქემის Morning, Evening და Afternoon ქმედებები FlowSwitch ქმედებას, რომელსაც აქვს ერთი თვისება გამოსახულებისათვის და იგი განსაზღვრავს გადართვის შტოს ცვლადის მნიშვნელობას (ნახ.16.45). ჩვენ პროექტში განვახორციელებთ გადამრთველის რეალიზებას მისალმების მიზნით წელიწადის დროის მიხედვით.



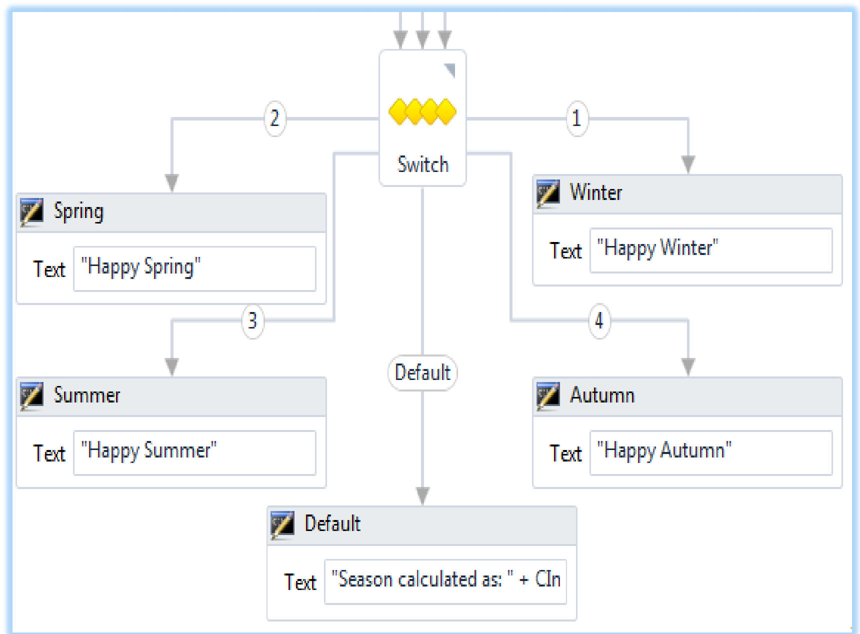
ნახ.16.45

შენიშვნა: გამოსახულების ჩაწერის ფორმა არაა დამოკიდებული გამოყენებული ენის სინტაქსზე. აქ მიღებულია, რომ ეს ფორმა იყოს Visual Basic ენის სინტაქსის მსგავსი.

ჩვენი გამოსახულება განსაზღვრავს წელიწადის დროის (ზამთარი, გაზაფხული, ზაფხული, შემოდგომა) სეზონს. მაგალითად, თუ მიმდინარე (ახლანდელი) თვე არის დეკემბერი, იანვარი ან თებერვალი, მაშინ გამოსახულება გვაძლევს 0–ს. ანალოგიურად მარტი, აპრილი და მაისი მოგვცემს 1–ს და ა.შ. გადამრთველის ოთხ შტოსათვის უნდა განვსაზღვროთ 0,1,2 და 3, ანუ სეზონები.

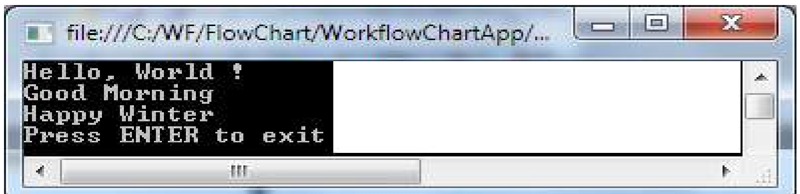
- **FlowStep ქმედების დამატება**

FlowSwitch აქტიურობის ყოველი შტო იძახებს FlowStep ქმედებას. ის Toolbox-ში არაა, ამიტომ მისი ცხადი სახით შექმნა არ შეიძლება. სქემას გადამრთველის გამოსასვლელზე უნდა დაემატოს რამდენიმე ქმედება, მაგალითად, ხუთი WriteLine და მათი შეერთების დროს მოხდება შინაგანად FlowStep–ის მნიშვნელობის განსაზღვრა. WriteLine ქმედებებისთვის ჩავწეროთ DisplayName-ში: Winter, Spring, Summer, Autumn და Default, აგრეთვე შესაბამისი მისალმებები (ნახ.16.46).



ნახ.16.46

პროგრამის ამუშავებით(F5) მივიღებთ შედეგს (ნახ.16.47).



ნახ.16.47. FlowSwitch აპლიკაციის შედეგები

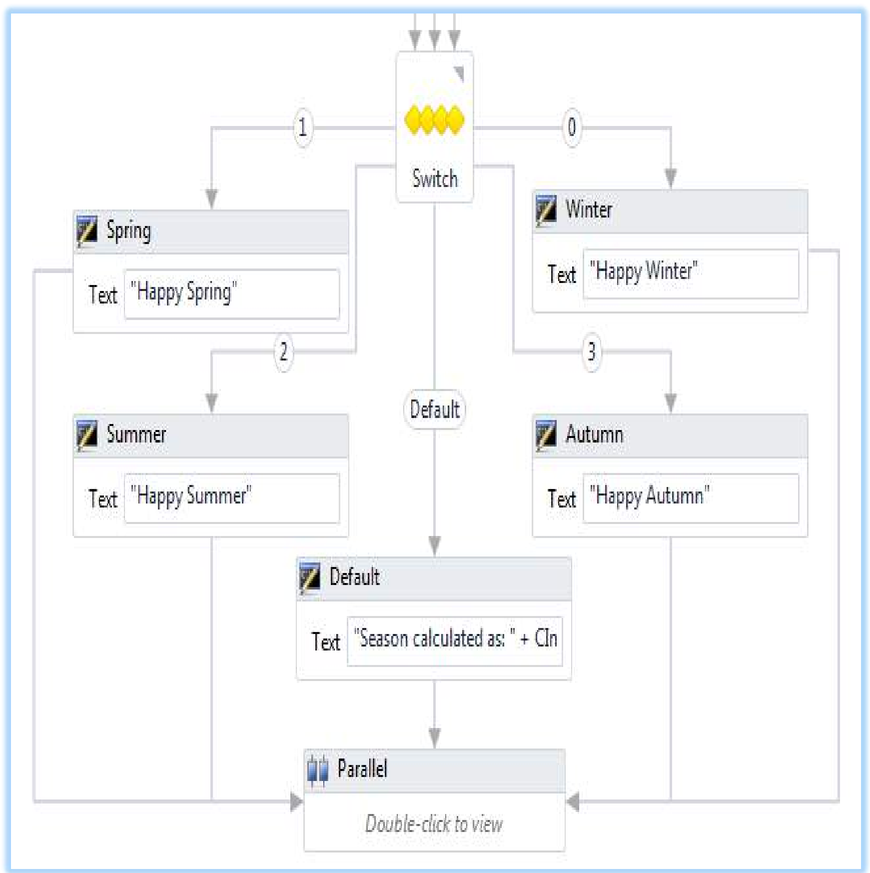
- **პარალელური პროცესები (Parallel)**

ჩვენი პროექტის ფარგლებში განვიხილოთ პარალელური პროცესების საკითხი, Parallel ქმედების მაგალითზე, რომელიც საშუალებას იძლევა განვსაზღვროთ Sequence ქმედებათა რაოდენობა, რომლებიც სრულდება პარალელურად (ერთ-დროულად). ამ პროექტში ყოველი შტო გამოიტანს ინფორმაციის თავის ნაწილს. გამოტანის რიგითობას

არ აქვს არსებითი მნიშვნელობა, მთვარია, რომ ისინი მოთავსებულია ერთ Parallel აქტიურობაში და ყველა სრულდება ერთდროულად.

- **Parallel ქმედების დამატება**

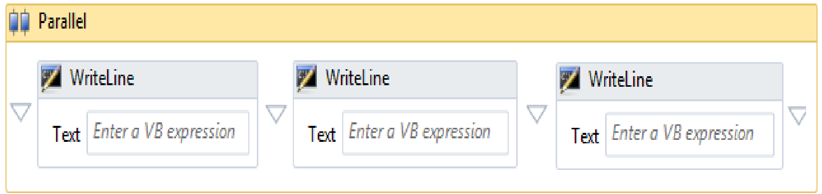
გადმოვიტანოთ Parallel აქტიურობა ჩვენი სამუშაო პროცესის ბოლოში. ხუთივე WriteLine-დან გავავლოთ კავშირი Parallel ქმედებასთან (ნახ.16.48).



ნახ.16.48

- **შტოების დამატება**

ორჯერ დავკლიკოთ Parallel ქმედება და მის ზედაპირზე გადმოვიტანოთ სამი WriteLine ქმედება (ნახ.16.49).



ნახ.16.49

ერთმა უნდა გამოიტანოს მიმდინარე თარიღი, მეორემ – დრო და მესამემ კვირის დღე. მათ Text-ში გამოსახულებებისათვის (expression) შევიტანოთ შემდეგი:

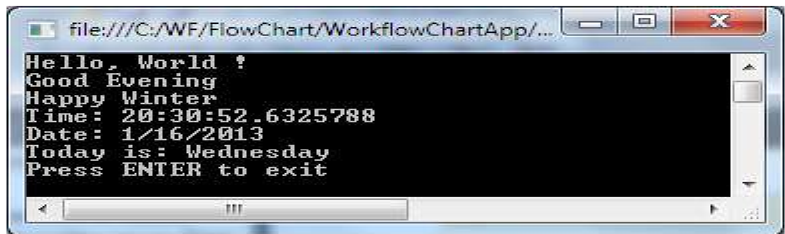
"Date: " + DateTime.Now.Date.ToShortDateString()

"Time: " + DateTime.Now.TimeOfDay.ToString()

"Today is: " + DateTime.Now.ToString("dddd")

შენიშვნა: Parallel ქმედება ნებას იძლევა მხოლოდ ერთი ქმედება განხორციელდეს ერთ შტოში, რაც ჩვენ მაგალითზე კარგად მუშაობს. თუ გვინდა მულტიქმედებების გამოყენება თითოეულ შტოში, მაშინ უნდა ვიხმაროთ Sequence ქმედება. აქ შეიძლება მის შიგნით დავამატოთ ქმედებათა სათანადო რაოდენობა.

პროგრამის ამუშავების შემდეგ მიიღება ასეთი შედეგები:



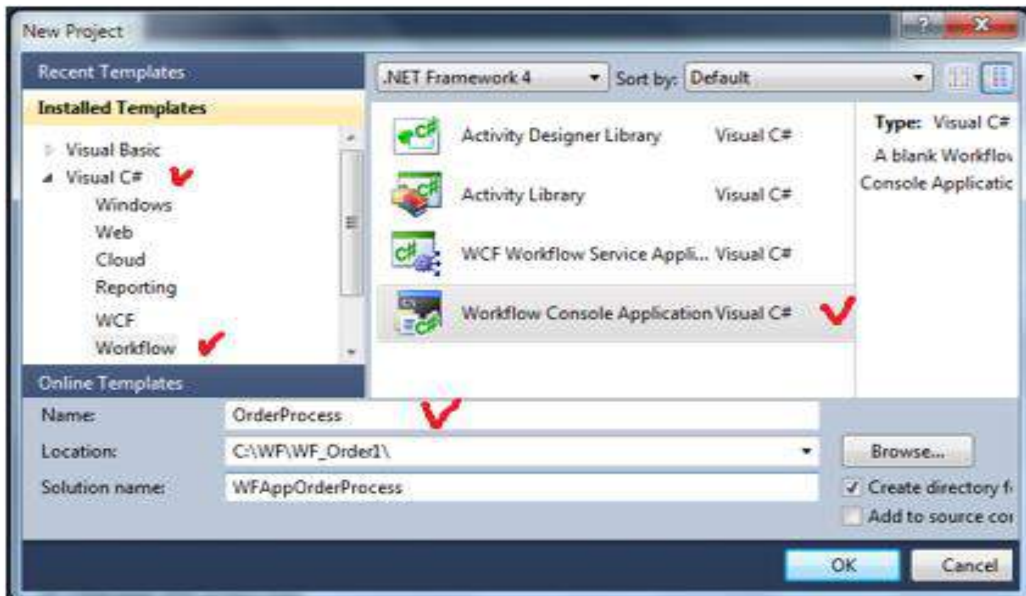
ნახ.16.50

XVII თავი
ბიზნესპროცესების (Workflow) დაპროექტება

17.1. რთული ბიზნესპროცესის არგუმენტები

განვიხილოთ Visual Studio.NET პაკეტის WF-ის სამუშაო გარემოში რთული ბიზნესპროცესის შემავალ-გამომავალი არგუმენტების გადაცემის ხერხები workflow პროცესსა და ჰოსტ-აპლიკაციას შორის.

შევქმნათ ახალი ბიზნესპროცესის კონსოლური აპლიკაცია პროექტის სახელით OrderProcess, და Solution-ის სახელით WFAppOrderProcess (ნახ.17.1).

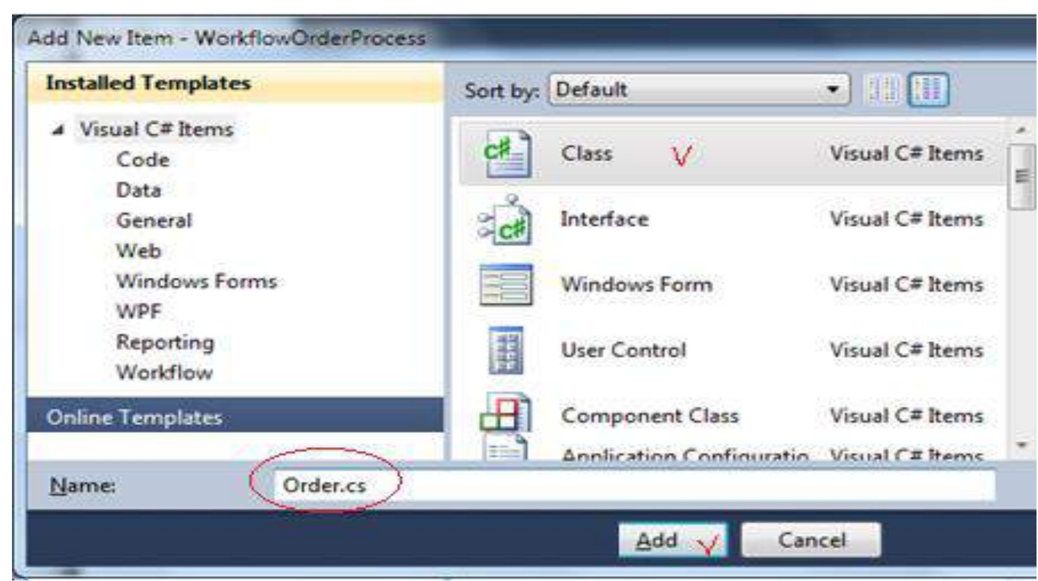


ნახ.17.1. ახალი კონსოლური აპლიკაციის პროექტის შექმნა

პროექტში უნდა განვსაზღვროთ შეკვეთა გარკვეულ პროდუქციაზე და ამ შეკვეთის გადაცემა ბიზნესპროცესში. ეს სამუშაო პროცესი (workflow) გამოითვლის შეკვეთის სრულ ღირებულებას და დააბრუნებს მას აპლიკაციაში.

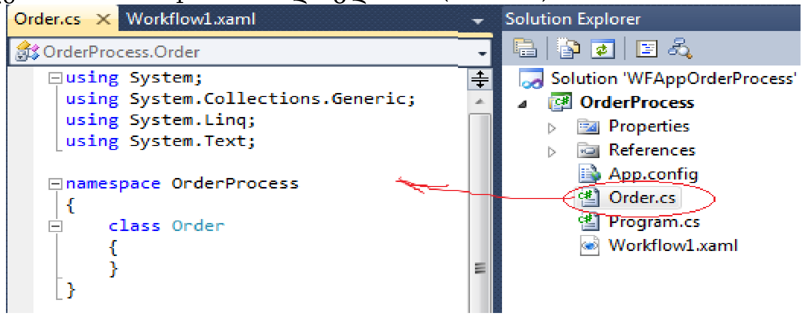
- **შეკვეთის კლასის განსაზღვრა (Defining the Order Class)**

პირველ ბიჯზე უნდა განისაზღვროს მონაცემთა სტრუქტურა, რომელიც უნდა შეიცავდეს შეკვეთის დეტალებს. Solution Explorer–ში დავამატოთ პროექტს ახალი კლასი: Add -> Class, სახელით Order.cs (ნახ.17.2).



ნახ.17.2. Order.cs კლასის შექმნა

მიიღება Solution Explorer ახალი კლასით (ნახ.17.3).



ნახ.17.3

კლასისი განსაზღვრისათვის Order.cs ფაილში შევიტანოთ შემდეგი კოდი (ლისტინგი_17.1).

```
//—— Order.cs ——  ლისტინგი_17.1 ————
using System;
using System.Collections.Generic;
namespace OrderProcess
{
  public class OrderItem
  {
      public int OrderItemID { get; set; }
```

```
        public int Quantity { get; set; }
        public string ItemCode { get; set; }
        public string Description { get; set; }
    }
public class Order
    {
        public Order()
        {
            Items = new List<OrderItem>();
        }
        public int OrderID { get; set; }
        public string Description { get; set; }
        public decimal TotalWeight { get; set; }
        public string ShippingMethod { get; set; }
        public List<OrderItem> Items { get; set; }
    }
}
```

Order კლასი შეიცავს რამდენიმე ღია წევრს (OrderID, Description, TotalWeight და მეთოდს ShippingMethod). აგრეთვე OrderItem კლასის კოლექციას. ეს დეტალები ესაჭიროება ბიზნესპროცესს, რათა გაითვალოს შეკვეთის ღირებულება.

ავაგოთ გადაწყვეტილება (solution) **F6 –ით (!!!)**, ეს შექმნის Order კლასს, რომელიც შემდეგ ბიჯებზე იქნება მისაწვდომი.

- **ბიზნესპროცესის რეალიზაცია (Implementing)**

Solution შაბლონი ქმნის სამუშაო პროცესის ფაილს Workflow1.xaml სახელით. შევცვალოთ ეს სახელი (Rename-თი) OrderWF.xaml სახელით.

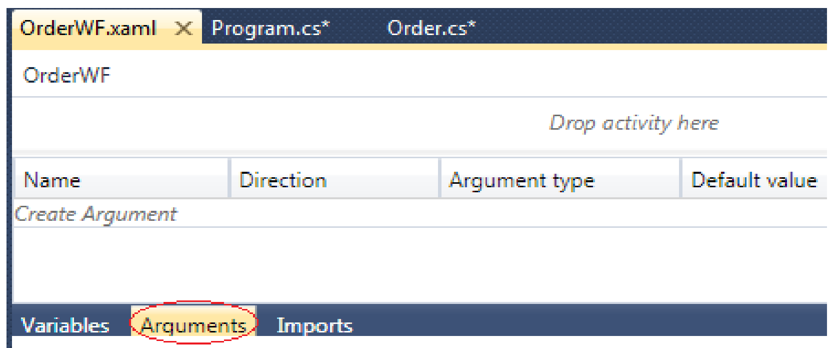
OrderWF.xaml ფაილში შევასწოროთ პირველი სტრიქონი:

x:Class="OrderProcess.OrderWF"

გავხსნათ Program.cs კოდი და `new Workflow1()` შევცვალოთ ახლით - `new` OrderWF()-ით.

- **არგუმენტების განსაზღვრა**

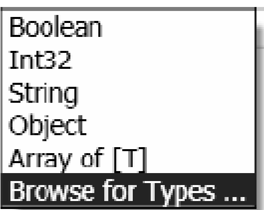
გავხსნათ OrderWF.xaml ფაილი დიზაინის რეჟიმში. ახლა უნდა განვსაზღვროთ (შემავალი და გამომავალი) არგუმენტები ბიზნესპროცესისათვის. დავკლიკოთ Argument ღილაკი ფანჯრის ქვედა მარცხენა ნაწილში. გამოჩნდება არგუმენტების ცარიელი კოლექცია (ნახ.17.4).



ნახ.17.4. ცარიელი არგუმენტების სია

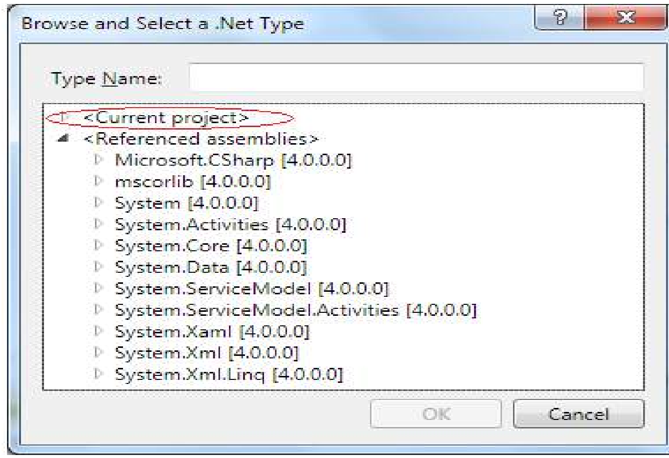
შენიშვნა: ცვლადები განისაზღვრებოდა მთელი პროცესისათვის, ან რომელიმე უბანზე და მათი შვილებისათვის. არგუმენტები ყოველთვის მთელი პროცესისათვისაა.

Create Argument -ით შევიტანოთ Name=OrderInfo, Direction=In და ArgumentType–სთვის ავირჩიოთ Browse for Types:



ნახ.17.5

გამოვა 17.6 ნახაზზე ნაჩვენები შედეგი <Current project>-ით [F6-ით კომპილირება წინა ეტაპზე აუცილებელია ამისთვის !!!].



ნახ.17.6

შევიტანოთ: TypeName=OrderProcess.Order, მივიღებთ ფანჯარას (ნახ.17.7).

ნახ.17.7

შემდეგ გავხსნათ OrderProcess და ავირჩიოთ Order კლასი და ღილაკი OK.

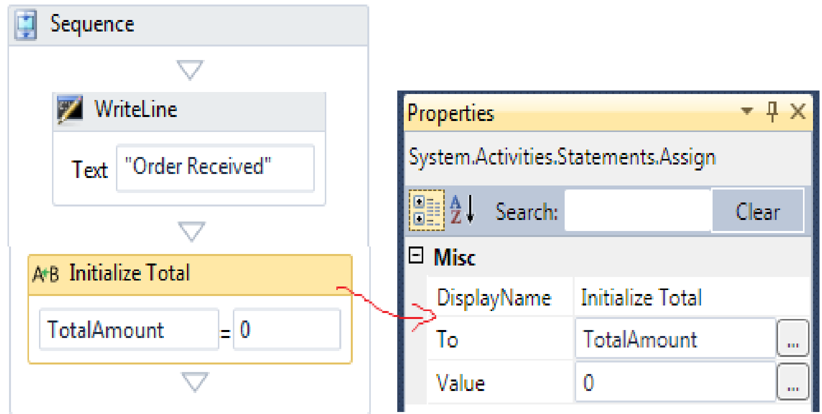
კიდევ ავირჩიოთ არგუმენტის შექმნა და შევიტანოთ სახელი TotalAmount. მდგომარეობა Direction უნდა იყოს **Out**, ასევე ArgumentType უნდა იყოს Decimal. (Type Name–ში შეიძლება System.Decimal მითითება უშუალოდ, ძებნის გარეშე). მიიღება:

Name	Direction	Argument type	Default value
OrderInfo	In	Order	*Enter a VB expression*
TotalAmount	Out	Decimal	*Default value not supported*
Create Argument			

ნახ.17.8

- **ბიზნესპროცესის დაპროექტება (დიზაინი)**

ახლა შეიძლება განსაზღვროს ქმედებები, რომლებიც დაამუშავებენ შეკვეთას, რომელიც გადმოიცა. დავიწყოთ Sequence ქმედების გადმოტანით სამუშაო პროცესის დიაგრამაზე. შემდეგ გადმოვიტანოთ WriteLine ქმედება. Text–სთვის ჩავწეროთ “Order Received”. გადმოვიტანოთ Assign ქმედება WriteLine–ს ქვემოთ. DisplayName დავაყენოთ Initialize Total. თვისებისათვის TotalAmount მნიშვნელობა შევიტანოთ 0.

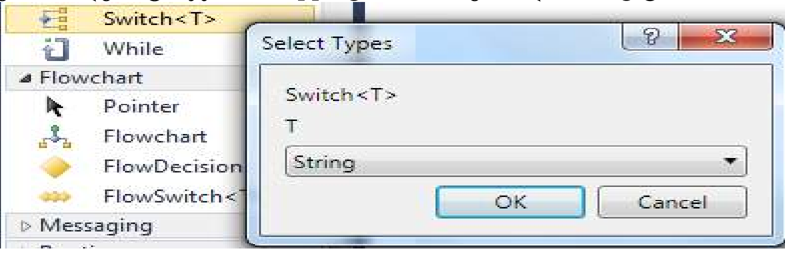


ნახ.17.9. საწყისი ინიციალიზაცია TotalAmount=0

- **Switch ქმედება**

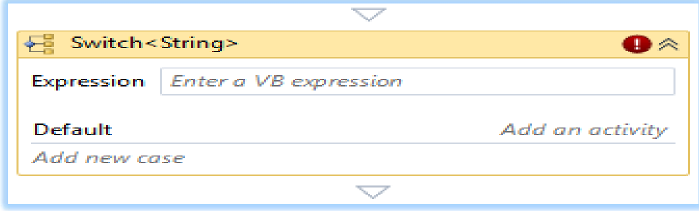
Switch ქმედება მუშაობს ისე, როგორც Switch ოპერატორი C# ენაში. ის საშუალებას იძლევა შესრულდეს sequence ქმედებები გამოსახულებათა ანალიზის საფუძველზე. Switch აქტიურობა გამოიყენება ShippingMethod მეთოდის განსაზღვრისათვის, ზარგის დამუშავების ღირებულების დადგენის მიზნით.

ეს ნიშნავს, რომ იგი განსაზღვრულია როგორც კლასის შაბლონი და მუშაობს მონაცემთა სხადასხვა ტიპებთან. გადმოვიტანოთ ინსტრუმენტების პანელიდან Switch ქმედება Assign ქმედების ქვემოთ, შევარჩიოთ გამოსახულების (Expression) ტიპი. (ნახ.17.10). გამოჩნდება ფანჯარა; ShippingMethod მეთოდი String ტიპისაა.



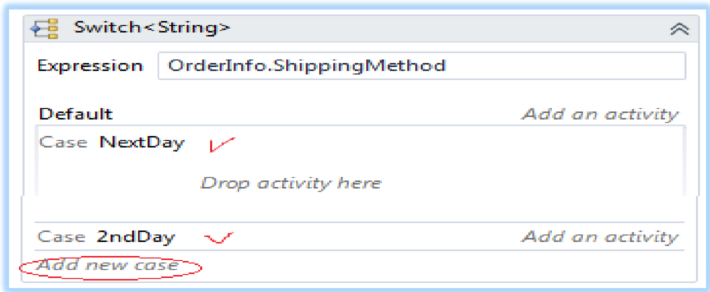
ნახ.17.10.

DisplayName–ში ჩავწეროთ Handling Charges. მივიღებთ:



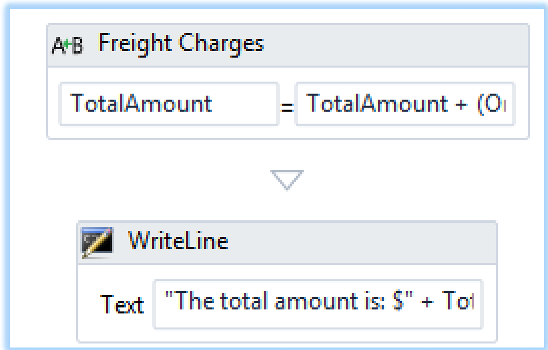
ნახ.17.11

Switch ქმედებას აქვს Expression თვისება, მიმდინარე ბლოკი და რამდენიმე მომხმარებლის მიერ სპეციფირებული case ბლოკი. შევიტანოთ გამოსახულება **OrderInfo.ShippingMethod.** აგრეთვე Ad new case–ში შევიტანოთ ჯერ NextDay, შემდეგ 2ndDay. შედეგები 17.11 ნახაზზეა მოცემული.



ნახ.17.12

დავამატოთ ახალი Assign და WriteLine (ნახ.17.13).



ნახ.17.13

Properties-ში შევიტანოთ გამოსახულებები:
TotalAmount და TotalAmount + (OrderInfo.TotalWeight * 0.5D) – პირველში, "The total amount is: $" + TotalAmount.ToString() – მეორეში.

Program.cs –ის კოდი მოცემულია 172 ლისტინგში.

// --- ლისტინგი-17_2 ------

```
using System;
using System.Activities;
using System.Activities.Statements;
using System.Collections.Generic;
namespace OrderProcess
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            Order myOrder = new Order
            {
                OrderID = 1,
                Description = "Need some stuff",
                ShippingMethod = "2ndDay",
                TotalWeight = 100
            };
  // create dictionary with input arguments for the workflow
           IDictionary<string, object> input =
                        new Dictionary<string, object>
              {
                { "OrderInfo" , myOrder }
              };
            // execute the workflow
           IDictionary<string, object> output =
               WorkflowInvoker.Invoke(new OrderWF(), input);

          // Get the TotalAmount returned by the workflow
            decimal total = (decimal)output["TotalAmount"];
        Console.WriteLine("Workflow returned ${0} for my
                                   order total", total);
            Console.WriteLine("Press ENTER to exit");
            Console.ReadLine();
        }
    }
}
```

OrderWF.xaml ფაილის კოდის ფრაგმენტი <Switch . . . > -თვის:

```
. . .
<Switch x:TypeArguments="x:String" DisplayName="Handling
     Charges" Expression="OrderInfo.ShippingMethod"

   sap:VirtualizedContainerService.HintSize="476,264">
      <Switch.Default>
       <Add x:TypeArguments="x:Decimal, x:Decimal,
           x:Decimal" DisplayName="Add 5"
```

```
        sap:VirtualizedContainerService.HintSize="458,100"
        Left="[TotalAmount]" Result="[TotalAmount]"
             Right="[5.0D]" />
      </Switch.Default>
      <sap:WorkflowViewStateService.ViewState>
       <scg3:Dictionary x:TypeArguments="x:String,
                  x:Object">
          <x:Boolean x:Key="IsExpanded">True</x:Boolean>
          <x:Boolean x:Key="IsPinned">False</x:Boolean>
        </scg3:Dictionary>
      </sap:WorkflowViewStateService.ViewState>
  <Add x:TypeArguments="x:Decimal, x:Decimal, x:Decimal"
       x:Key="NextDay" DisplayName="Add 15"
    sap:VirtualizedContainerService.HintSize="456,100"
    Left="[TotalAmount]" Result="[TotalAmount]"
           Right="[15.0D]" />

  <Add x:TypeArguments="x:Decimal, x:Decimal, x:Decimal"
       x:Key="2ndDay" DisplayName="Add 10"
   sap:VirtualizedContainerService.HintSize="456,100"
   Left="[TotalAmount]" Result="[TotalAmount]"
           Right="[10.0D]" />
</Switch>
      ...
```

მუშაობის შედეგები:

ნახ.17.14

17.2. რთული ბიზნესპროცესის გამოსახულებათა ქმედებები

განვიხილოთ Visual Studio.NET პაკეტის WF-ის სამუშაო გარემოში რთული ბიზნესპროცესის გამოსახულებათა ქმედებების (Expression Activities) გამოყენების საკითხი.

წინა პარაგრაფში ჩვენ განვსაზღვრეთ გადამრთველის ორი case-ბლოკი NextDay და 2ndDay, პლუს ერთი ბლოკი ავტომატურად გამოყენებადი ShippingMethod-ის მიერ, როცა არაა წინა ორი მითითებული. ამჯერად უნდა მიეთითოს ქმედებები (მაგალითად, Sequence), რომლებიც უნდა შესრულდეს თითოეული შემთხვევის დროს. ჩვენი პროექტისათვის გამოვიყენოთ Add ქმედება.

<u>შენიშვნა:</u> System.Activities.Expressions სახელსივრცე შეიცავს ქმედებებს, რომლებიც გამოიყენება ბიზნეს–პროცესებში, მათ შორის Add, Subtract, Multiply და Divide. იგი შეიცავს აგრეთვე ისეთ ჩაშენებულ ქმედებებს, როგორიცაა Equal, GreaterThan, And და Or, რომლებიც გამოიყენება გამოსახულებათა შესაფასებლად. მათი გამოყენება შესაძლებელია უშუალოდ მუშა პროცესებში.

განვიხილოთ პრაქტიკული ამოცანა:

ინსტრუმენტების პანელზე, სამწუხაროდ, არაა Add ქმედება. ჩვენ უნდა შევიდეთ კოდის რედაქტირების არეში. შევინახოთ პროექტი.

Solution Explorer–ში OrderWF.xaml–ზე მარჯვენა ღილაკით ავირჩიოთ Code view. სისტემა მოითხოვს არსებული ფანჯრის დახურვას, „დიახ“.

Switch ქმედება განისაზღვრება შემდეგი კოდით, რომლის ლისტინგი 17.3 მოცემულია ქვემოთ.

```
<!------ ლისტინგი_17.3 ------------>
<Switch x:TypeArguments="x:String" DisplayName="Handling Charges"
          Expression="OrderInfo.ShippingMethod"
          sap:VirtualizedContainerService.HintSize="476,158">
     <sap:WorkflowViewStateService.ViewState>
       <scg3:Dictionary x:TypeArguments="x:String, x:Object">
         <x:Boolean x:Key="IsExpanded">True</x:Boolean>
         <x:Boolean x:Key="IsPinned">False</x:Boolean>
       </scg3:Dictionary>
     </sap:WorkflowViewStateService.ViewState>
     <x:Null x:Key="NextDay" />
     <x:Null x:Key="2ndDay" />
   </Switch>
```

საყურადღებოა, რომ x: Null ატრიბუტები NextDay და 2ndDay ბლოკებში ნიშნავს, რომ ჯერ არაა ქმედებები მათთვის განსაზღვრული. ამიტომ შევცვალოთ ეს სტრიქონები შემდეგით:

```
<Add x:TypeArguments="s:Decimal, s:Decimal, s:Decimal"
        x:Key="NextDay"
     DisplayName="Add 15" Left="[TotalAmount]"
         Result="[TotalAmount]"
                    Right="[15.0D]" />
<Add x:TypeArguments="s:Decimal, s:Decimal, s:Decimal"
        x:Key="2ndDay"
    DisplayName="Add 10" Left="[TotalAmount]"
          Result="[TotalAmount]"
                    Right="[10.0D]" />
```

დავამატოთ შემდეგი კოდი უშუალოდ წინა კოდის წინ, რათა განისაზღვროს ავტომატურად არსებული შემთხვევის ვარიანტი:

```
<Switch.Default>
    <p:Add x:TypeArguments="s:Decimal, s:Decimal, s:Decimal"
                                        DisplayName="Add 5"
     Left="[TotalAmount]" Result="[TotalAmount]" Right="[5.0D]" />
</Switch.Default>
```

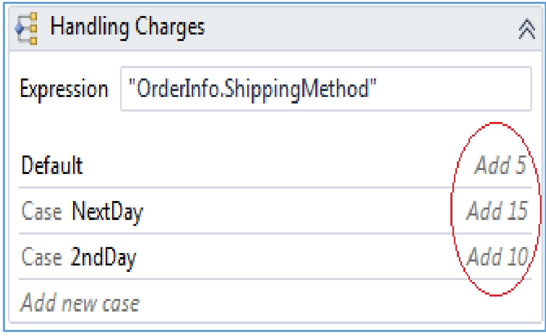
Add ქმედებას აქვს სამი თვისება: Left, Right და Result. ცვლადის მნიშვნელობა Right თვისებიდან ემატება Left–ის მნიშვნელობას, ხოლო ჯამი ინახება Result–ში. Left და Result თვისებებისათვის ყენდება TotalAmount არგუმენტი. Right თვისება განისაზღვრება როგორც სტატიკური ცვლადი, რომელიც განსხვავებულია ყოველი კონკრეტული case-სათვის.

Switch ქმედების საბოლოო განსაზღვრა დაკომპლექტებულია 17.4 ლისტინგში.

```
<!–– ლისტინგი 17.4 –––––––––––––––––––––– ––>
<Switch x:TypeArguments="x:String" DisplayName="Handling Charges"
      Expression="OrderInfo.ShippingMethod"
      sap:VirtualizedContainerService.HintSize="476,158">
    <Switch.Default>
    <Add x:TypeArguments="s:Decimal, s:Decimal, s:Decimal" DisplayName="Add 5"
            Left="[TotalAmount]" Result="[TotalAmount]" Right="[5.0D]" />
```

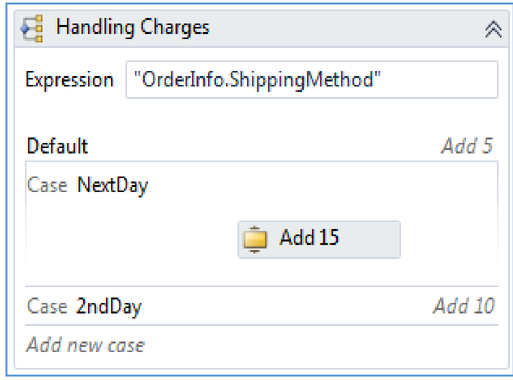
```
  </Switch.Default>
   <Add x:TypeArguments="s:Decimal, s:Decimal, s:Decimal" x:Key="NextDay"
      DisplayName="Add 15" Left="[TotalAmount]" Result="[TotalAmount]"
       Right="[15.0D]" />
   <Add x:TypeArguments="s:Decimal, s:Decimal, s:Decimal" x:Key="2ndDay"
      DisplayName="Add 10" Left="[TotalAmount]" Result="[TotalAmount]"
       Right="[10.0D]" />
</Switch>
```

შევინახოთ პროექტი და Solution Explorer–დან გამოვიტანოთ OrderWF.xaml ფაილი დიზაინერში. Switch ქმედება ასე გამოიყურება (ნახ.17.15):



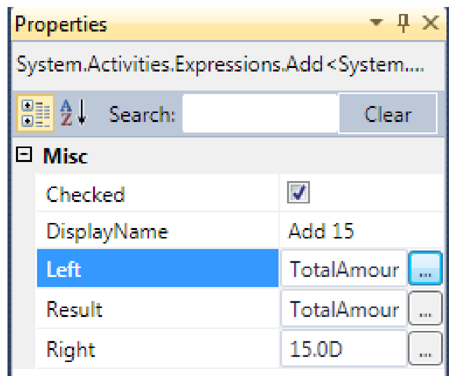
ნახ.17.15

როდესაც ShippingMethod არის NextDay, მაშინ TotalAmount–ს ემატება $ 15, თუ არის 2ndDay, მაშინ ემატება $ 10. მიწოდების ყველა სხვა მეთოდისთვის ავტომატურად ემატება $ 5. დიაგრამა ასახავს ყველა case შემთხვევას და DisplayName–ს შესრულებადი ქმედებისათვის შესაბამის case-ში. თუ დავდგებით Switch ქმედების რომელიმე case-ბლოკზე, მაშინ გააქტიურდება ეს ბლოკი (ნახ.17.16):



ნახ.17.16. Switch ქმედების გაფართოება

Add ქმედებაზე დაკლიკვით გამოვა თვისებების ფანჯარა (ნახ.17.17). აქ შესაძლებელია თვისებების შეცვლა, საჭიროებისამებრ.



ნახ.17.17. Add ქმედების თვისებების ფანჯარა

შენიშნა: ათობითი მნიშვნელობა შეიტანება 15.0D ფორმატით (VB-ენა). C# -ის დროს, მაგალითად, 15.0m მოგვცემს შედომას. უნდა გვახსოვდეს, რომ გამოსახულებები შეიტანება VB–სინტაქსით !!!

გადმოვიტანოთ სქემაზე შემდეგი Assign ქმედება Switch-ის ქვემოთ და დავაყენოთ DisplayName სატრანსპორტო ხარჯი (Freight Charges). To თვისებისათვის შევიტანოთ TotalAmount. ხოლო Value თვისებისთვის შევიტანოთ:

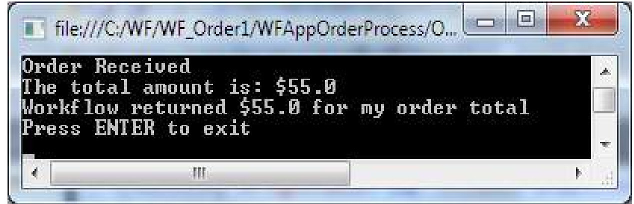
TotalAmount + (OrderInfo.TotalWeight * 0.50D)

ეს ფორმულა მიწოდების ხარჯებისთვის დაამატებს $ 0,50 –ს ყოველ ფუნტზე (წონა).

გადმოვიტანოთ ახალი WriteLine ქმედება „Freight Charges"–ს შემდეგ და Text თვისებაში ჩავწეროთ:

"The total amount is: $" + TotalAmount.ToString()

ეკრანზე გამოიტანება გაანგარიშებული შეკვეთის საერთო ღირებულება. საბოლოო ბიზნეს–პროცესის სქემა ასე გამოიყურება (ნახ.17.18).



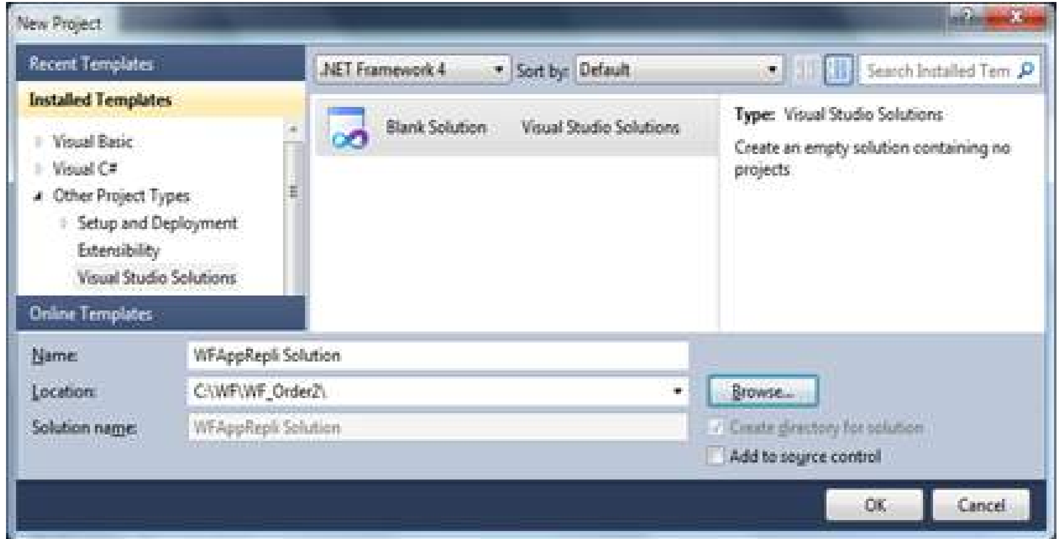
ნახ.17.18

17.3. განმეორებადი ქმედებები

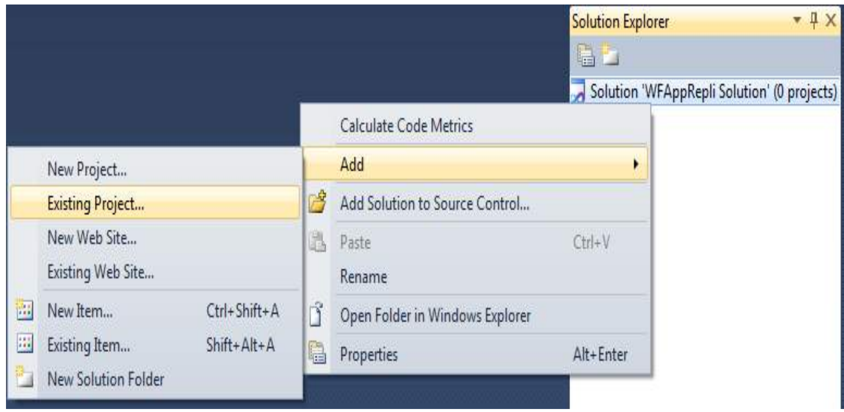
განვიხილოთ Visual Studio.NET პაკეტის WF-ის სამუშაო გარემოში რთული ბიზნესპროცესის შესრულება განმეორებადი ქმედებების გამოყენების საფუძველზე.

წინა პარაგრაფის ამოცანაში შექმნილი ბიზნესპროცესი ითვლიდა შეკვეთის საერთო ღირებულებას და გადასცემდა მას არგუმენტის სახით. ის შედგებოდა მხოლოდ დამუშავებისა და მიწოდების ხარჯებისაგან. ამ თავში დავამატებთ შეკვეთის თითოეული პოზიციის ღირებულებას. ამისათვის კი საჭირო იქნება ქმედების შესრულება ყოველი ელემენტისათვის.

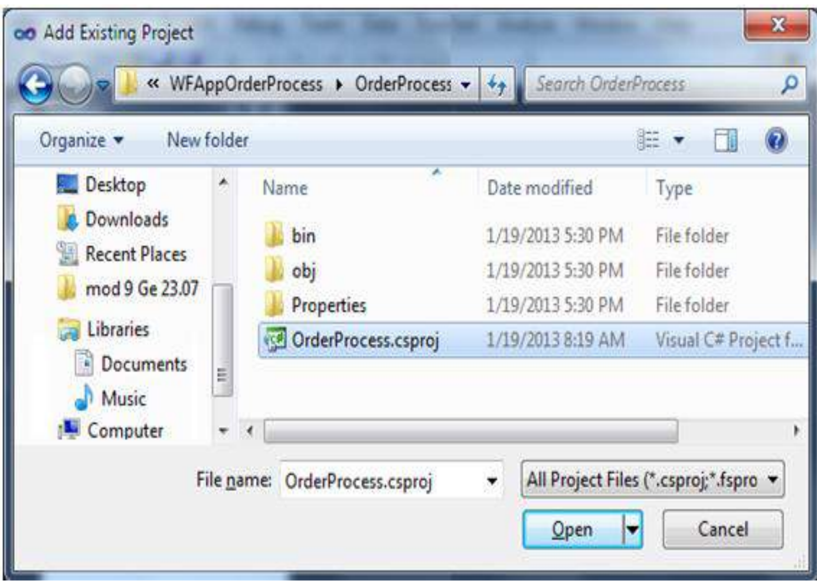
განვიხილოთ წინა პროექტის გამოყენების საკითხი. ავამუშაოთ Visual Studio, შევქმნათ ახალი პროექტი WFAppRepli Solution სახელით (ნახ.17.19).



ნახ.17.19. Blank Solution პროექტის შექმნა



ნახ.17.20. წინა პროექტის გადმოტანა ახალში



ნახ.17.21

ნახ.17.22. შედეგი Solution Explorer-ში

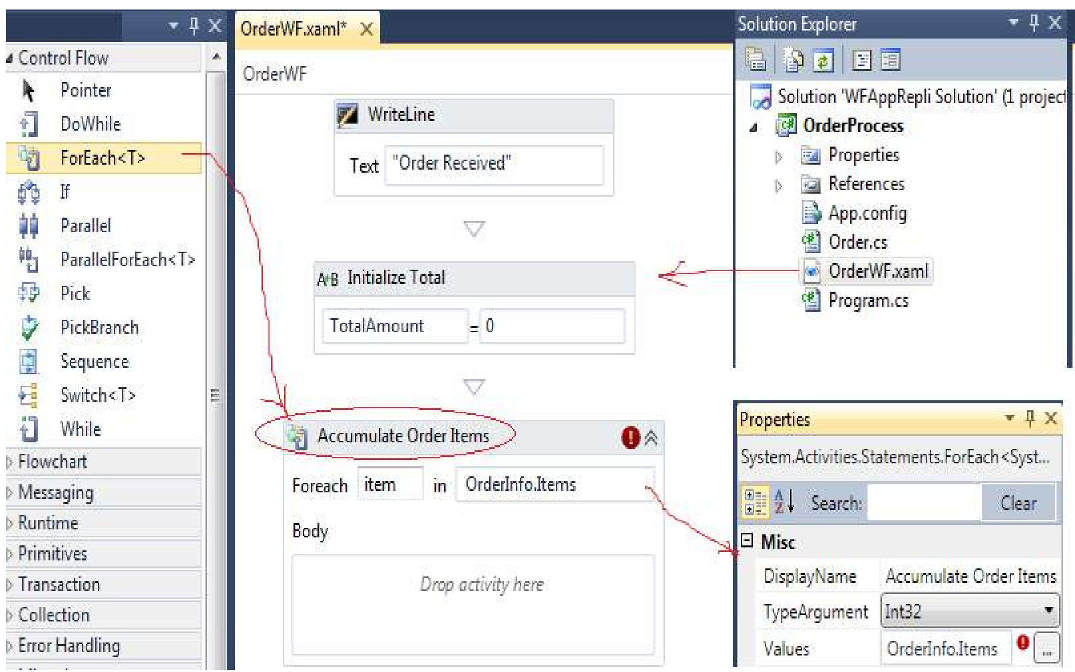
- **შეკვეთის ელემენტების დამუშავება:**

ახლა პროგრამაში უნდა დავამატოთ ბიჯი შეკვეთის ცალკეული კომპონენტის ღირებულების გასაანგარიშებლად.

- **ForEach ქმედება**

აქტიურობა ForEach ასრულებს ქმედებას (ან ქმედებათა მიმდევრობას) ყოველი ელემენტისათვის კოლექციაში. ესაა სწორედ ის ქმედება, რომელიც მოცემულ პროექტში გამოდგება.

გავხსნათ OrderWF.xaml ფაილი დიზაინის რეჟიმში. გადმოვიტანოთ ForEach <T> ქმედება “Initialize Total” ქმედების ქვემოთ. DisplayName შევცვალოთ Accumulate Order Items–ით. ქმედება შეიძლება იყოს შეკუმშული. თუ ასეა, მაშინ გააფართოვეთ (ღილაკი ზედა მარჯვენა კუთხეში). მიიღება (ნახ.17.21).



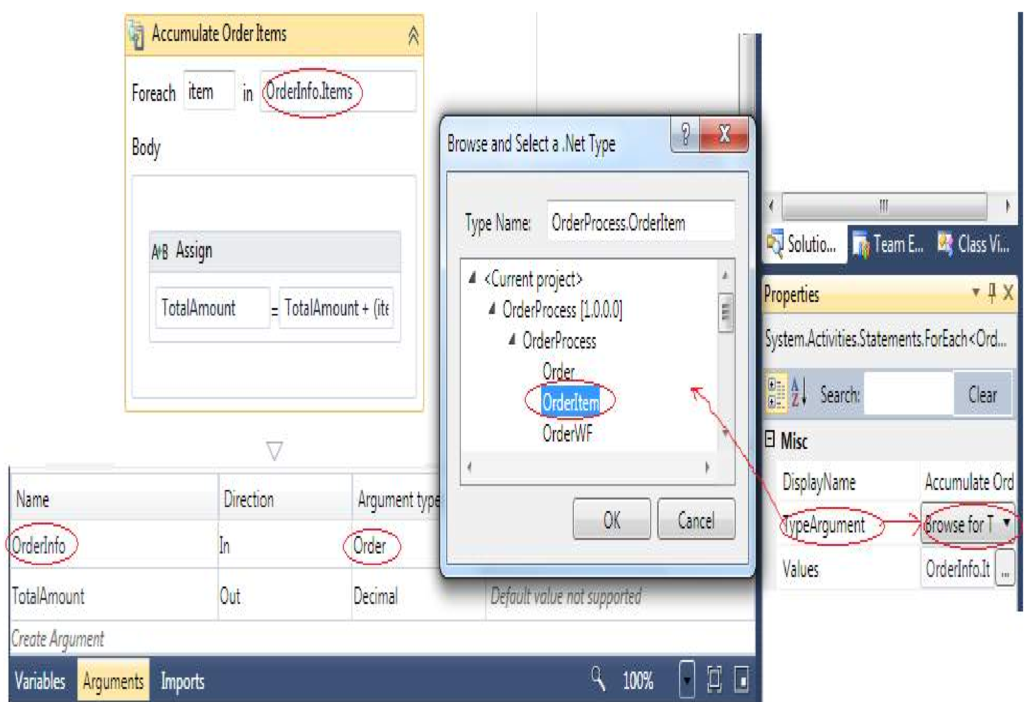
ნახ.17.21. ForEach ქმედების საწყისი მდგომარეობა

Expression ველში ჩავწეროთ OrderInfo.Items.

ქმედებაში <T>–ს არსებობა მიუთითებს იმაზე, რომ იგი არის უნივერსალური (generic) კლასი და საჭიროა ტიპის განსაზღვრა, რომელიც კოლექციაში იქნება. Propertries-ში ავტომატურად მიეთითება Int32 ტიპი. წითელი წრე კუთხეში გვამცნობს, რომ შეცდომაა: შეტანილი გამოსახულება (OrderInfo.Items) არ შეიცავს მთელ რიცხვებს.

OrderInfo.Items არის OrderItem ობიექტების ერთობლიობა Order შეკვეთიდან, რომელიც მიღებულ იქნა სამუშაო პროცესში. Properties–ის TypeArgument–ის კომბობოქსის სიაში ავირჩიოთ ტიპი Browse-თი. OrderProcess–ში ავირჩიოთ OrderItem (ნახ.17.22).

ამ პროექტში ჩვენ უნდა მივიღოთ მარტივად შეკვეთის თითოეული ელემენტის ფიქსირებული ფასი. რეალურ სცენარით ასეთი თვისებები ამოიღება მონაცემთა ბაზიდან. ახლა გადმოვიტანოთ და შევცვალოთ სქემაზე Assign ქმედება ForEach აქტიურობით.



ნახ.17.22. OrderItem კლასის არჩევა

ForEach ქმედებაზე შეიძლება მხოლოდ ერთი ქმედების გადმოტანა. თუ საჭიროა ერთზე მეტი ქმედების გამოყენება, მაშინ უნდა გადმოვიტანოთ ჯერ Sequence აქტიურობა და შემდეგ მასზე დავდოთ რამდენიმე სხვა ქმედება.

Assign ქმედების თვისებებში შევიტანოთ:

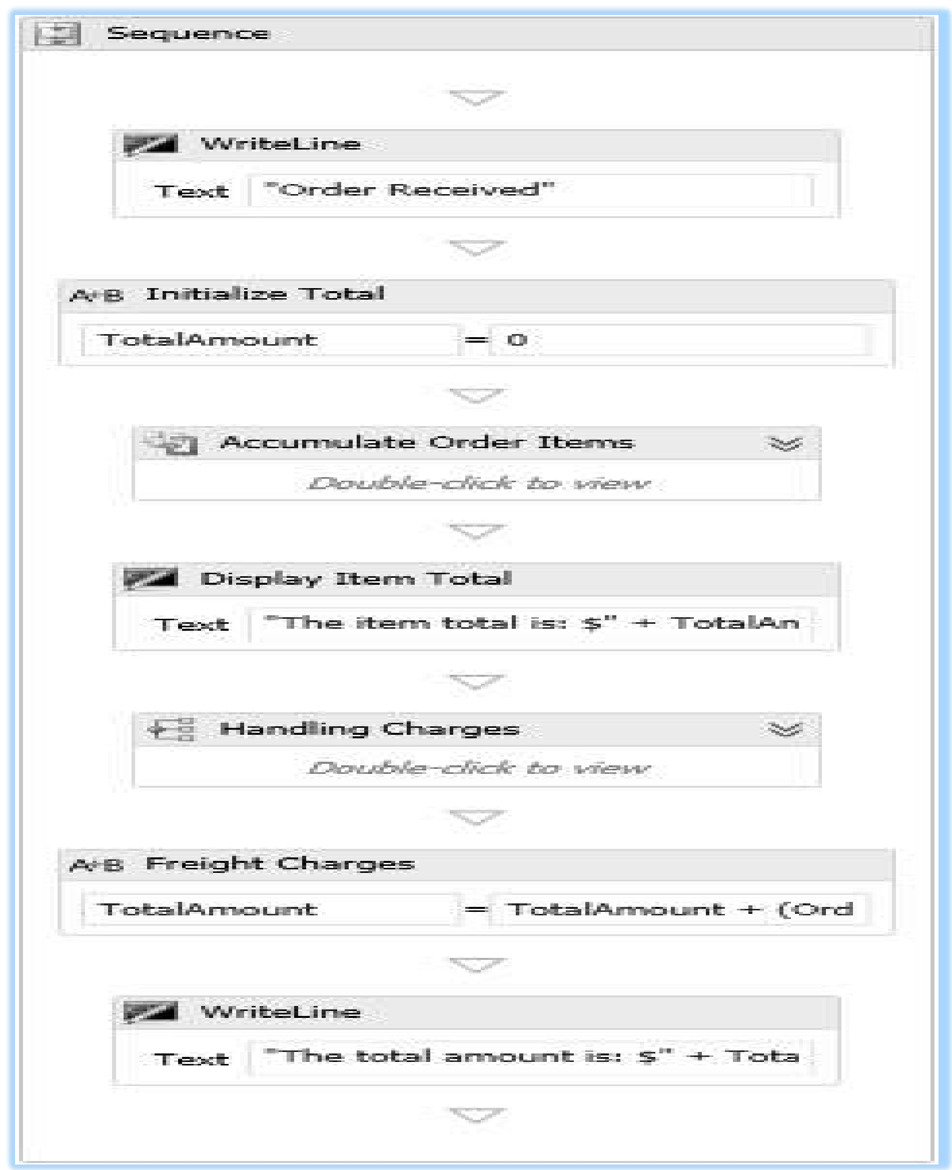
To–თვის TotalAmount და Value–თვის TotalAmount + (item.Quantity * 10.0D). ეს ნიშნავს, რომ $10 ემატება ყოველ ელემენტზე.

Assign ქმედების ქვემოთ გადმოვიტანოთ WriteLine აქტიურობა, რომლისთვისაც ჩავწეროთ სახელი Display Item Total. მისიText თვისებისთვის კი შევიტანოთ გამოსახულება:

```
"The item total is: $" + TotalAmount.ToString()
```

საბოლოოდ, 17.23 ნახაზზე მოცემულა მთლიანი პროცესის workflow სქემა.

აქ შეკუმშული სახითაა მოცემული ქმედებათა რიგი, რომელთა გაშლა შესაძლებელია მარჯვენა ზედა კუთხეში არსებული ისრით.



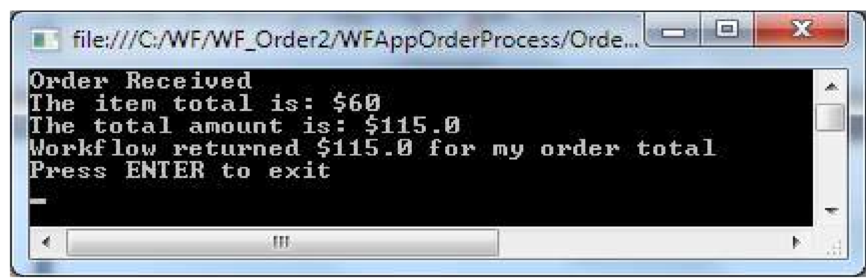
ნახ.17.23

- **შეკვეთის ელემენეტების დამატება**

ForEach ფუნქციის შემოწმებამდე საჭიროა აპლიკაციაში ცვლილების შეტანა, რათა Order კლასში შესაძლებელი იყოს ზოგიერთი OrderItem ობიექტის დამატება. გავხსნათ Program.cs ფაილი და ჩავამატოთ უშუალოდ Order კლასის შემდეგ ასეთი კოდი:

```
// Add some OrderItem objects
            myOrder.Items.Add(new OrderItem
            {
                OrderItemID = 1,
                Quantity = 1,
                ItemCode = "12345",
                Description = "Widget" } );
            myOrder.Items.Add(new OrderItem
            {
                OrderItemID = 2,
                Quantity = 3,
                ItemCode = "12346",
                Description = "Gadget" } );
            myOrder.Items.Add(new OrderItem
            {
                OrderItemID = 3,
                Quantity = 2,
                ItemCode = "12347",
                Description = "Super Widget" } );
```

პროგრამის ამუშავებით მივიღებთ შედეგებს (ნახ.17.24).



ნახ.17.24

- **ParallelForEach ქმედება**

ნაცვლად ForEach ქმედებისა შესაძლებელია ParallelForEach ქმედების გამოყენება. იგი კონფიგურირებულია ზუსტად ForEach-ით. ერთადერთი განსხვავება არის, როგორ სრულდება აქტიურობა. როგორც მისი სახელი ვარაუდობს, ParallelForEach აქტიურობა სრულდება „შვილი პროცესის“ პარალელურად, ხოლო ForEach ქმედება ახორციელებს ამ პროცესებს მიმდევრობით.

ჩვენი პროექტისათვის ეს ეფექტი არ ჩანს, მაგრამ თუ რთული მიმდევრობითი პროცესებია გასაშვები, მაშინ მათი პარალელურად შესრულება უფრო მისაღებია.

მაგალითად, თუ უნდა გაიგზავნოს შეტყობინება და დაველოდოთ პასუხს, შეიძლება პარალელურად გაიშვას სხვა პროცესი, Wait დრო არ არის, დაიკარგება.

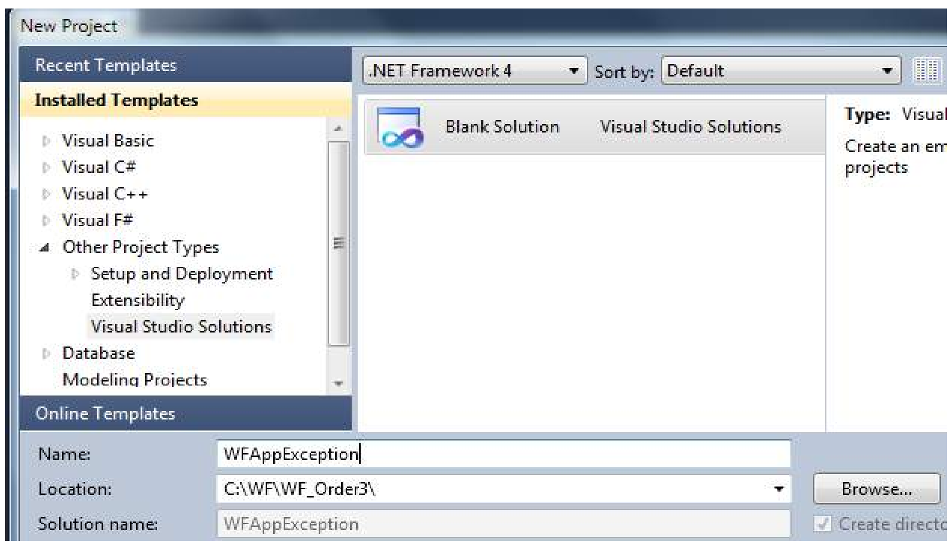
ამის შესამოწმებლად პროექტში შეიძლება წაიშალოს ForEach აქტიურობა და დავდოთ ParallelForEach ქმედება იმავე ადგილზე. კონფიგურირებით და ამუშავებით ვნახავთ, რომ გვაქვს იგივე შედეგი.

17.4. გამონაკლისების დამუშავება

განვიხილოთ გამონაკლისების (გამოსარიცხი პროცესების - Exception Handling) ანალიზის ჩატარებისა და მათი აღმოფხვრის ხერხები და ასევე მათი გამოყენება.

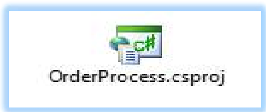
უნდა დავამატოთ ლოგიკა, რათა მოხდეს გადამოწმება, რომ შეკვეთის თითოეული ელემენტი არის საწყობში. ამისათვის, იტარეაციულად შეკვეთის თითოეული ელემენტისათვის, როგორც ეს წინა თავში მოხდა. თუ პროდუქცია (ელემენტი) არაა საწყობში, უნდა გამოვრიცხოთ ეს გამონაკლისი, რომელიც დააფიქსირა workflow პროცესმა.

განვიხილოთ პრაქტიკული მაგალითი: აქაც ახალი პროექტის შესაქმნელად გამოვიყენოთ წინა პროექტი:



ნახ.17.25

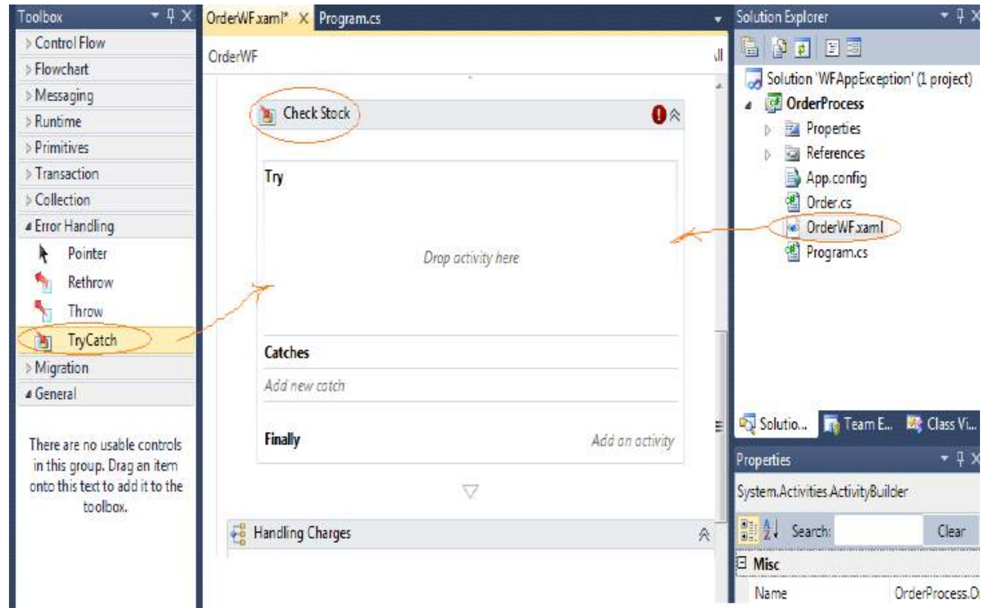
Solution Explorer–დან დავამატოთ Add-> Existing Project ჩვენი წინა პროექტის ფაილი, რომელიც ინახება OrderProcess ფოლდერში.



• მარაგის შემოწმების ქმედება (Check Stock Activity)

დავამატოთ ჩვენ პროექტს ლოგიკა, რომელიც დაადგენს, არის თუ არა ელემენტების საკმარისი მარაგი შეკვეთისათვის.

გავხსნათ OrderWF.xaml ფაილი დიზაინის რეჟიმში. გადმოვიტანოთ TryCatch ქმედება სქემაზე "Handling Charges" აქტიურობის შემდეგ. შევცვალოთ DisplayName სახელი Check Stockით. მიიღება (ნახ.17.26).



ნახ.17.26. TryCatch ქმედება

TryCatch შედგება სამი ნაწილისაგან:

- Try ნაწილში მოთავსდება იმ ქმედებათა მიმდევრობა, რომელთაც პოტენციურად შეუძლია გამონაკლისი სიტუაციების გენერირება;
- Catch ნაწილში განისაზღვრება ერთი ან რამდენიმე Catch–ობიექტი. ყოველი Catch ობიექტი ამუშავებს კონკრეტულ გამონაკლისს, ამიტომ მისი ყოველი ტიპისათვის უნდა დაიწეროს ცალკე დამუშავების პროცედურა;
- Finally ნაწილი, არაა სავალდებულო. აქ უნდა მოთავსდეს იმ ქმედებათა მიმდევრობა, რომელიც სრულდება Try ქმედების შემდეგ;
- გამონაკლისების განსაზღვრა:

ახლა განვსაზღვროთ გამონაკლისი სიტუაცია, როცა შეკვეთისათვის საჭირო ელემენტი არაა საწყობში. გავხსნათ Order.cs ფაილი და ჩავამატოთ შემდეგი კოდი, რომელიც განსაზღვრავს OutOfStockException კლასს. Order კლასის განსაზღვრის შემდეგ OrderProcess სახელსივრცის შიგნით პროგრამის ლისტინგს ასეთი სახე ექნება:

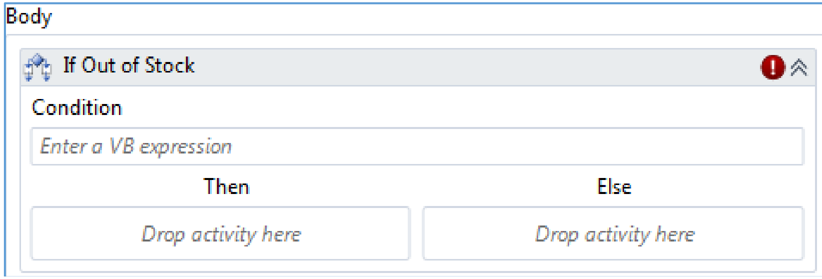
```
// -- ლისტინგი-17.4 --------
using System;
using System.Collections.Generic;
namespace OrderProcess
{
   public class OrderItem
   {
      public int OrderItemID { get; set; }
      public int Quantity { get; set; }
      public string ItemCode { get; set; }
      public string Description { get; set; }    }
   public class Order
   { public Order()
     {
        Items = new List<OrderItem>();
     }
     public int OrderID { get; set; }
     public string Description { get; set; }
     public decimal TotalWeight { get; set; }
     public string ShippingMethod { get; set; }
     public List<OrderItem> Items { get; set; }
   }
   //--------------------------------------------
   // გამონაკლისი სიტუაციის განსაზღვრა, როცა შეკვეთისთვის
   // საჭირო ელემენტი არაა საწყობში
   //--------------------------------------------
   public class OutOfStockException : Exception
   {
     public OutOfStockException() : base()    {    }
     public OutOfStockException(string message) : base(message)  {   }
   }
}
```

- **ForEach ქმედება**

გადმოვიტანოთ ForEach აქტიურობა Try ნაწილში და DisplayName შევცვალოთ სახელით Check Each Item. გავშალოთ ბლოკი და Expression ველში ჩავწეროთ OrderInfo.Items. თვისებებში შევცვალოთ TypeArgument ტიპი (როგორც წინა თავში) და ავირჩიოთ OrderItem კლასი.

- **If ქმედება**

გადმოვიტანოთ If ქმედება Try ნაწილში და DisplayName შევცვალოთ to If Out of Stock სახელით.



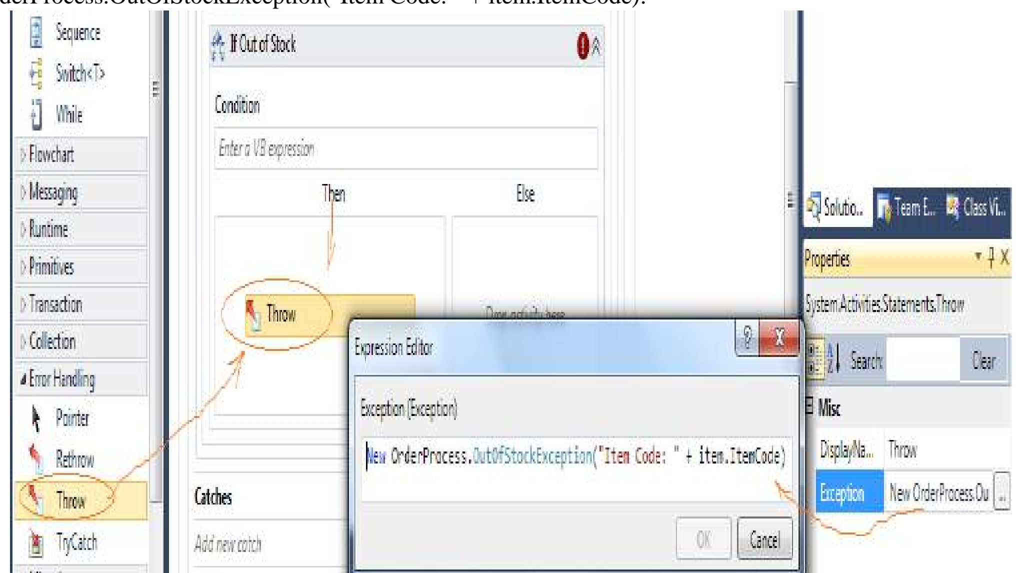
ნახ.17.27

Condition თვისებაში შევიტანოთ გამოსახულება: item.ItemCode = "12346"

<u>შენიშვნა:</u> რეალურ სისტემაში „არსებული ელემენტები“ უნდა შემოწმდეს მონაცემთა ბაზაში. აქ, პროექტის გადსამარტივებლად, გამოყენებულია ItemCode.

- **Throw ქმედება**

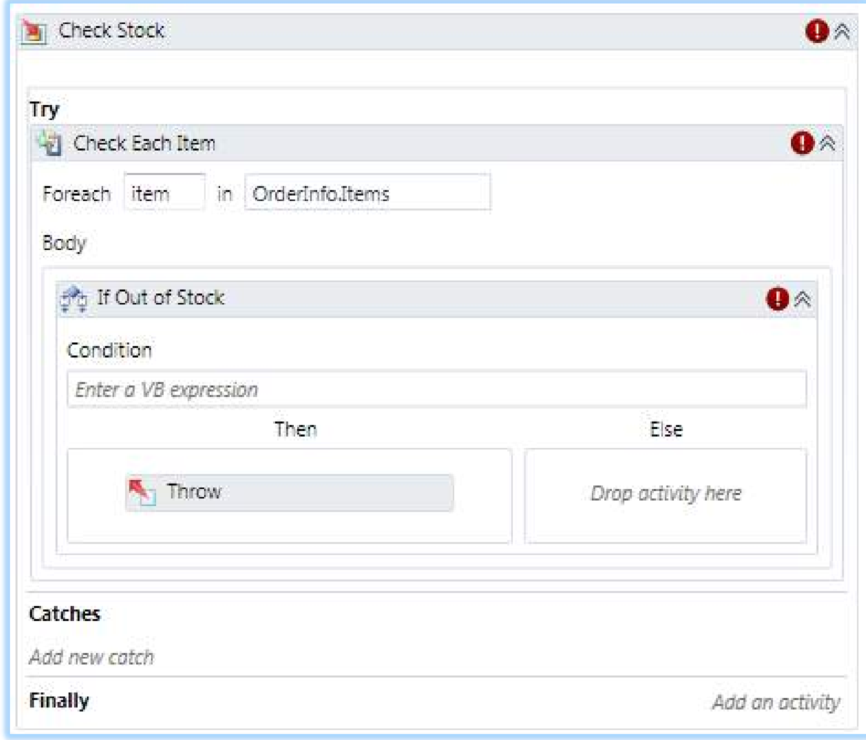
გადმოვიტანოთ Then ბლოკში Throw ქმედება. თვისებაში Exception შევიტანოთ New OrderProcess.OutOfStockException("Item Code: " + item.ItemCode).



ნახ.17.28

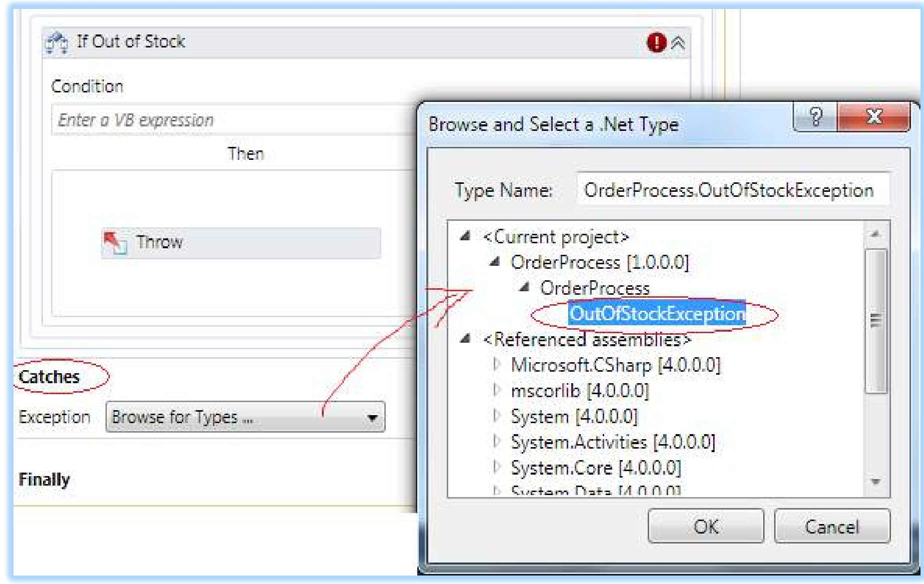
- **Catch ქმედება**

“Check Stock” ქმედება ასე გამოიყურება:

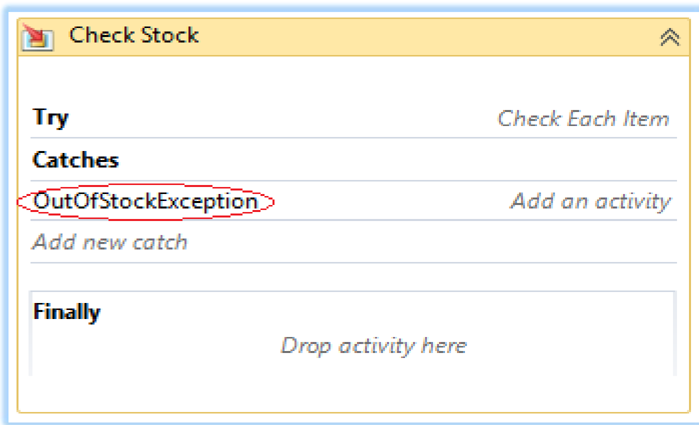


ნახ.17.29. **TryCatch** ქმედების სტრუქტურა

„დავკლიკოთ“ *Add new catch* ღილაკი.



ნახ.17.30. OutOfStockException -ის არჩევა

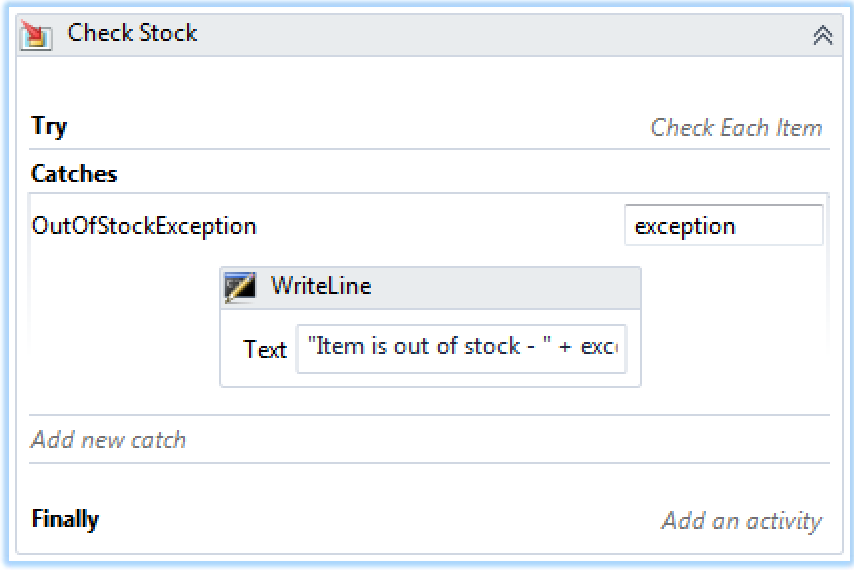


ნახ.17.31. Catch ქმედების ბლოკის დაკომპლექტება

ყოველი გამონაკლისი შემთხვევისათვის უნდა განისაზღვროს შესაბამისი ქმედება და ჩაიდოს ბლოკში. ჩვენი პროექტისათვის უნდა გამოვიტანოთ შეტყობინება, როცა არაა პროდუქცია. გადმოვიტანოთ WriteLine ქმედება Catch ნაწილში და შევიტანოთ Text-ში:

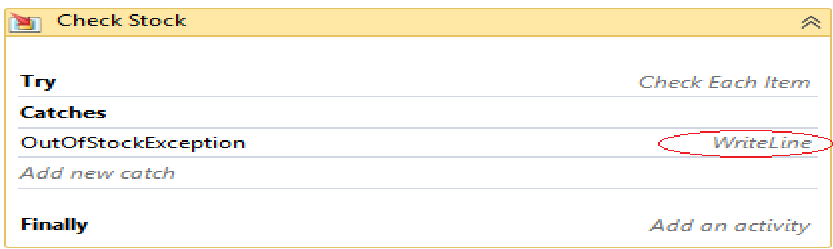
"Item is out of stock - " + exception.Message

საბოლოოდ “Check Stock” ბლოკი ასე გამოიყურება:



ნახ.17.32

Expand-Collapse (გაფართოება/შეკუმშვის) გამოყენების შემდეგ წინა სქემა ასეთ სახეს მიიღებს.

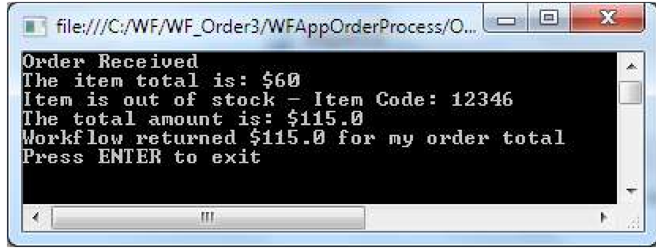


ნახ.17.33

Try, Catches და Finally ნაწილები შეჯამებულია. Try გვიჩვენებს, რომ არსებობს ქმედება სახელით "Check Each Item" (შემოწმდეს თითოეული ელემენტი). Catch –ში ჩამოთვლილია გამონაკლისები, რომლებიც მუშავდება WriteLine ქმედებით (ჩვენთან ასეთი ერთია OutOfStockException). ვინაიდან Finaly–ში არაა ქმედებები, აქ არის Add an Activity ღილაკი, რომლითაც შესაძლებელია დამატებები.

- აპლიკაციის ამუშავება

  პროგრამის მუშაობის შედეგები ასეთია (ნახ.17.34).



ნახ.17.34

- გამონაკლისები (Exceptions)

  გამონაკლისების (ან გამოსარიცხი პროცესების) დამუშავება მნიშვნელოვანი ფრაგმენტია პროგრამული სისტემების მუშაობის დროს გაუთვალისწინებელი მოვლენების გამო პროგრამების ავარიული დამთავრების პრევენციისათვის.

გამონაკლისების დამუშავების პროცედურა TryCatch აქტიურობით რეალიზდება. კერძოდ, მთლიანი სამუშაო პროცესი თავსდება Try ნაწილში.

მაგალითისათვის მოვიყვანოთ შემდეგი კოდის ლისტინგი:

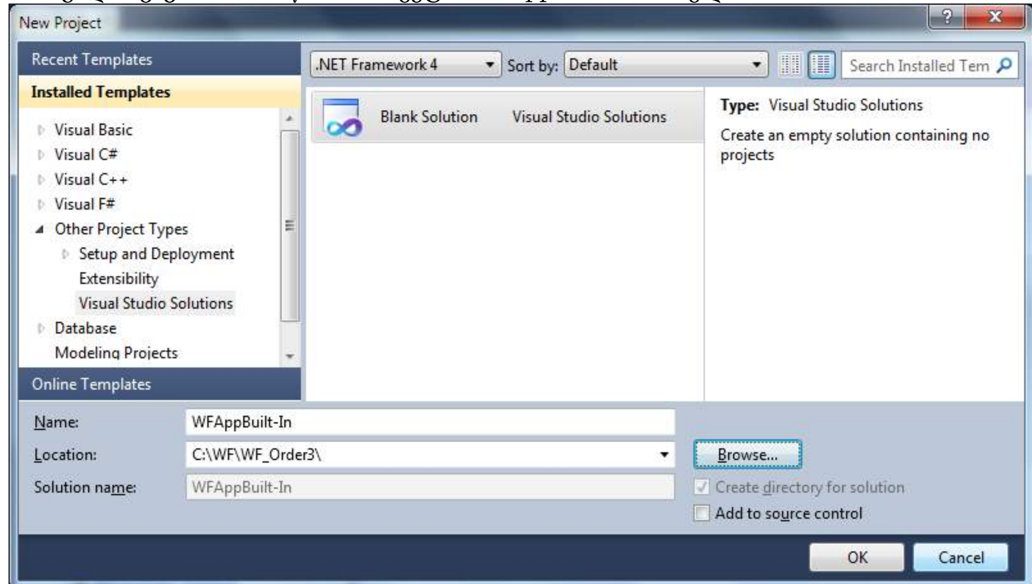
```
// --- ლისტინგი-17.7 ---- TryCatch1---
Sequence
{  CheckStock (Try)
   { Check Each Item (ForEach)
      {  If Out of Stock (If)
          {  Throw Exception (Then)
```

```
            }
        }
    }
    Catch  {
       }
        Remaining Activities
  }

// --- ლისტინგი-17.8 ---- TryCatch2---
Sequence
{ Check Each Item (ForEach)
      {  (Try)
           {    If Out of Stock (If)
                  {
                    Throw Exception (Then)
                   }
           }
          Catch  {
             }
      }
}

// --- ლისტინგი-17.9 ---- TryCatch3---
 (Try)
{  Sequence
     {  Check Each Item
           {
               If Out of Stock
                 {
                   Throw Exception
                  }
            }
         Remaining Activities
      }
  }
 Catch  {
    }
```

17.5. Built-In ქმედების გაფართოება

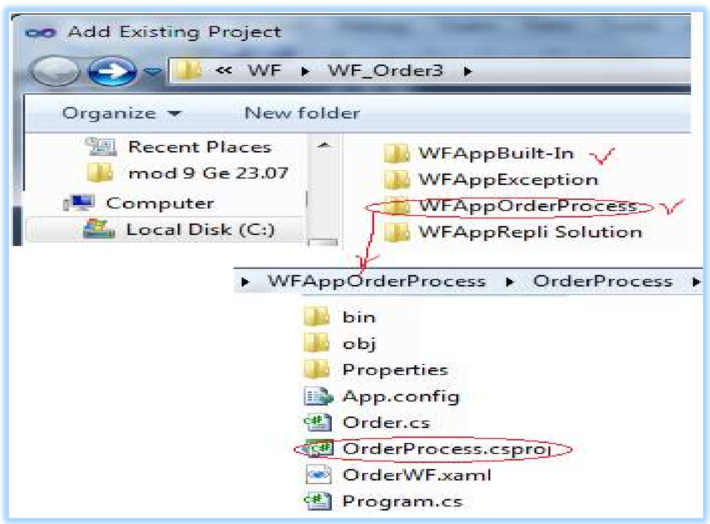
განვიხილოთ წინა პარაგრაფის პროექტისთვის შეკვეთის ფასწარმოქმნის წესების დაზუსტება, გაფართოების ორი ხერხი built-in ქმედებისათვის: მომხმარებლის ქმედებების შექმნა და InvokeMethod ქმედების გამოყენება.

გადმოვაკოპიროთ წინა პროექტი WFAppBuilt-In სახელით:



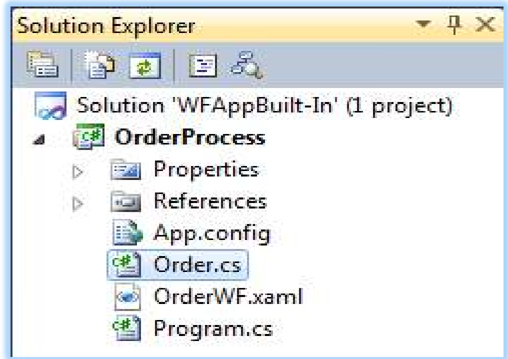
ნახ172.35

WFAppOrderProcess–დან OrderProcess–ში ავირჩიოთ ფაილი OrderProcess.csproj ;



ნახ.17.36

მიიღება Solution Explorer-ის საწყისი შემადგენლობა:



ნახ.17.37

- **მომხმარებლის ქმედებათა გამოყენება (Custom Activities)**

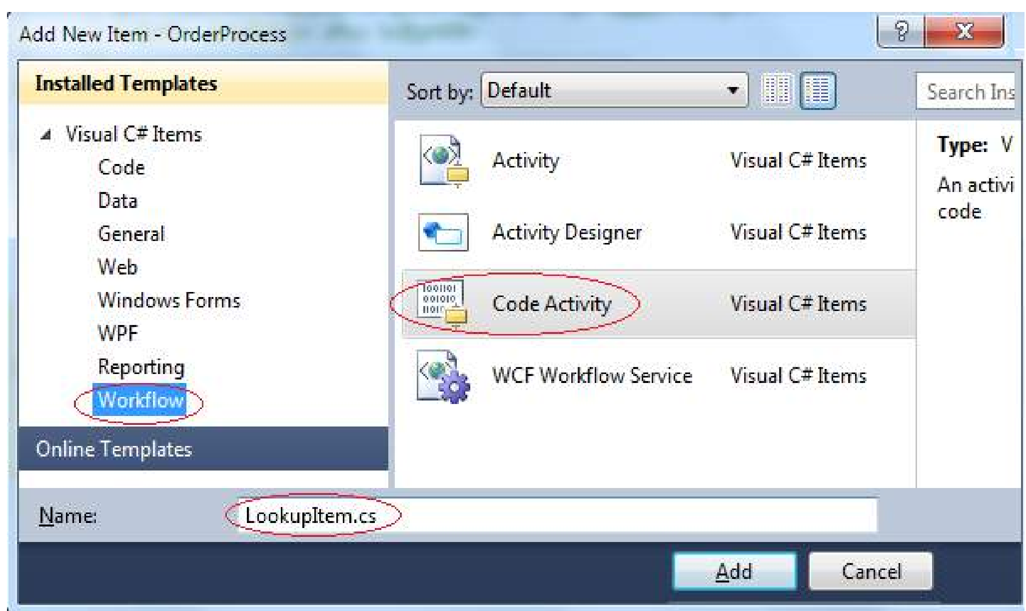
ყველა ელემენტისთვის ჩვენი პროექტი, იყენებს ფიქსირებულ ფასს ($ 10) შეიძლება ფასის დაზუსტება მომხმარებლის ქმედების შექმნით, რომელიც მოიძიებს ყველა ელემენტის ფასს ItemCode თვისებით. გავხსნათ Order.cs ფაილი და ჩავამატოთ შემდეგი კლასის განსაზღვრება:

```
//---------------------------------------------
// სტრუქტურის განსაზღვრა, რომელიც ბრუნდება
// LookupItem მომხმარებლის ქმედებით
//---------------------------------------------
public class ItemInfo
{
  public string ItemCode { get; set; }
  public string Description { get; set; }
  public decimal Price { get; set; }
}
```

შენიშვნა: WF 4.0–დან CodeActivity არის უკვე აბსტრაქტული კლასი, რომელიც გამოიყენება როგორც საბაზო კლასი მრავალი ჩაშენებული built-in ქმედებისათვის (და გამოყენებადი როგორც საბაზო მომხმარებლის ქმედებებისათვის). მაგრამ მისი უშუალო გამოყენება სამუშაო პროცესში არ შეიძლება. WF 4.0–ში მოგვიხდება არაერთი ასეთი, მომხმარებლის ქმედების დაწერა. საბედნიეროთ, ეს მარტივია, რასაც ქვემოთ ვნახავთ.

- **მომხმარებლის ქმედების რეალიზაცია**

Solution Explorer-ში OrderProcess პროექტზე მარჯვენა ღილაკით ავირჩიოთ Add->NewItem. შემდეგ Workflow კატეგორიაში ავირჩით Code Activity შაბლონი. შევიტანოთ კლასის სახელი LookupItem.cs, როგორც ნახაზზეა ნაჩვენები:



ნახ.17.38

ამ კლასის იმპლემენტაცია ნაჩვენებია 17.10 ლისტინგში.

// – – – ლისტინგი 17.10 – – –

```
using System;
using System.Activities;
namespace OrderProcess
{
  public sealed class LookupItem : CodeActivity
   {
    public InArgument<string> ItemCode { get; set; }
    public OutArgument<ItemInfo> Item { get; set; }
    protected override void
    Execute(CodeActivityContext context)
     {
      ItemInfo i = new ItemInfo();
    i.ItemCode = context.GetValue<string>(ItemCode);
         switch (i.ItemCode)
            {
                case "12345":
                    i.Description = "Widget";
                    i.Price = (decimal)10.0;
```

```
                        break;
                    case "12346":
                        i.Description = "Gadget";
                        i.Price = (decimal)15.0;
                        break;
                    case "12347":
                        i.Description = "Super Gadget";
                        i.Price = (decimal)25.0;
                        break;
                }
                context.SetValue(this.Item, i);
            }
        }
    }
```

ისევე, როგორც სამუშაო პროცესებს შეიძლება ჰქონდეს შემავალი და გამომავალი არგუმენტები, ასევე ქმედებასაც, ფაქტობრივად, თვისებები, რომლებიც იქნა დაყენებული built-in კლასისათვის (მაგ., Text თვისება WriteLine ქმედებისთვის), არის ქმედების არგუმენტები. თქვენი მომხმარებლის ქმედება (LookupItem) იღებს ItemCode–ს როგორც შემავალ არგუმენტს და აბრუნებს უკან ItemInfo კლასს, როგორც გამომავალ არგუმენტს.

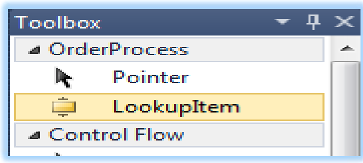
LookupItem კლასი არის წარმოებული CodeActivity საბაზო კლასიდან და უცვლის განსაზღვრებას (გადატვირთავს, overrides) Execute მეთოდს. ეს მეთოდი ქმნის ItemInfo კლასს და ინახავს შიგ ItemCode–ს. საყურადღებოა, რომ იგი უნდა იყენებდეს context.GetValue() მეთოდს, რათა მიიღოს ItemCode მნიშვნელობა, ვინაიდან არგუმენტის მონაცემების მხარდაჭერა ხდება უშუალოდ თვით სამუშაო პროცესის მიერ. CodeActivityContext–ს მიწოდება Execute() მეთოდი, როცა იგი გამოიძახება.

Execute() მეთოდი აკონკრეტებს აღწერისა და ფასის თვისებებს ItemCode–ს ბაზაზე (რეალურ აპლიკაციაში ის უნდა ვეძებოთ მონაცემთა ბაზაზში). და ბოლოს, იგი გამოიძახებს context.SetValue() მეთოდს ItemInfo კლასის შესანახად გამომავალ არგუმენტში.

F5–ით ავამუშავოთ აპლიკაცია.

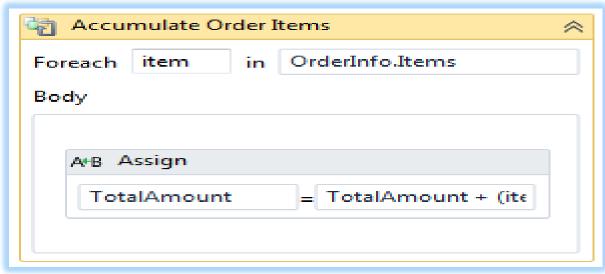
- **LookupItem ქმედების გამოყენება**

გავხსნათ OrderWF.xaml ფაილი დიზაინის რეჟიმში. საყურადღებოა, რომ LookupItem ქმედება დამატებული იყო Toolbox–დან:



ნახ.17.39

workflow სქემაზე გავშალოთ “Accumulate Order Items” ქმედება:

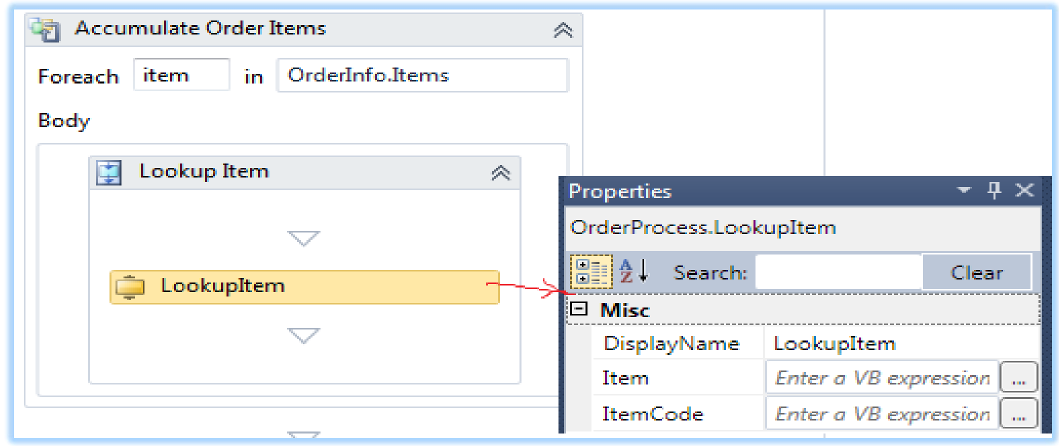


ნახ.17.40

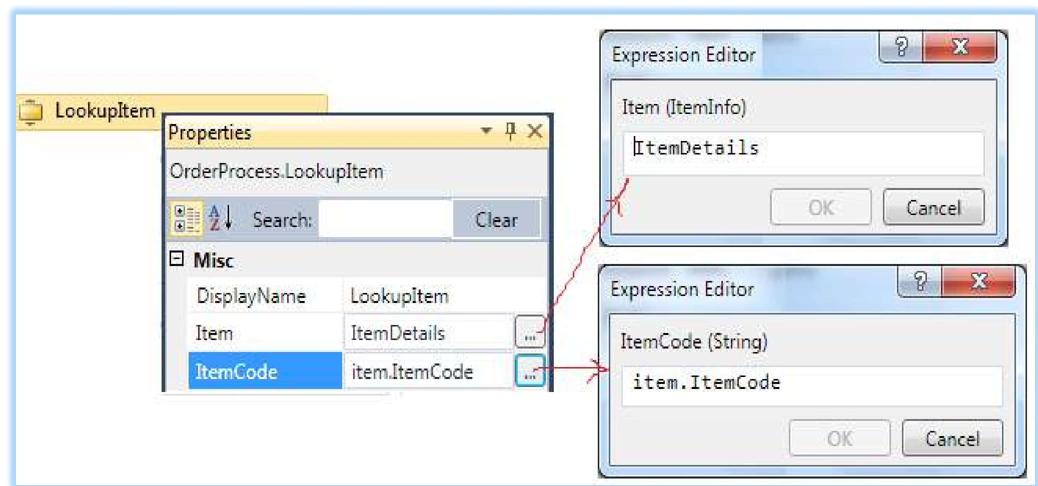
ის შეკვეთის ელემენტებს მარტივად ჩამოთვლის და ასრულებს Assign ქმედებას, რომელიც ამატებს OrderTotal–ში ყოველი ელემენტისათვის $10–ს. ავირჩიოთ Assign აქტიურობა და Delete–თი წავშალოთ იგი სქემიდან. მის ადგილას გადმოვიტანოთ Sequence ქმედება უშუალოდ Body სექციაში. DisplayName თვისებაში ჩავწეროთ Lookup Item და გავშალოთ. გადმოვიტანოთ Lookup Item ქმედება ამ Sequence–ზე (ნახ.17.41).

საყურადღებოა, რომ თვისებათა ფანჯარა შეიცავს არგუმენტების ItemCode–ს და Item–ს, რომლებიც ჩვენ ამ ქმედებისათვის განვსაზღვრეთ. არგუმენტის DisplayName იყო განსაზღვრული CodeActivity საბაზო კლასში.

ItemCode თვისებისათვის შევიტანოთ გამოსახულება: item.ItemCode (ნახ.17.42).

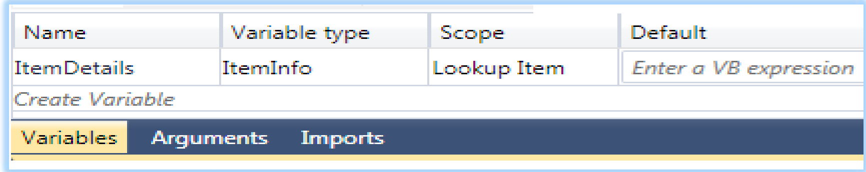


ნახ.17.41

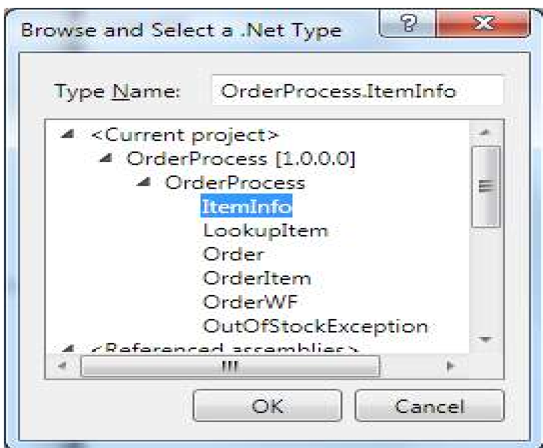


ნახ.17.42

უნდა განვსაზღვროთ ცვლადი (Variables) ItemInfo კლასის შესანახად, რომელიც აბრუნებს უკან მნიშვნელობას. დიზაინერის ქვედა მარცხენა კუთხეში ჩავრთოთ Variables და შევიტანოთ სახელი ItemDetails (ნახ.17.43). ტიპის ასარჩევად გამოვიყენოთ Browse და ავირჩიოთ ItemInfo კლასი (ნახ.17.44). განსაზღვრის არე (Scope) მიმდინარე Sequence ქმედებისათვის უნდა იყოს ("Lookup Item").



ნახ.17.43

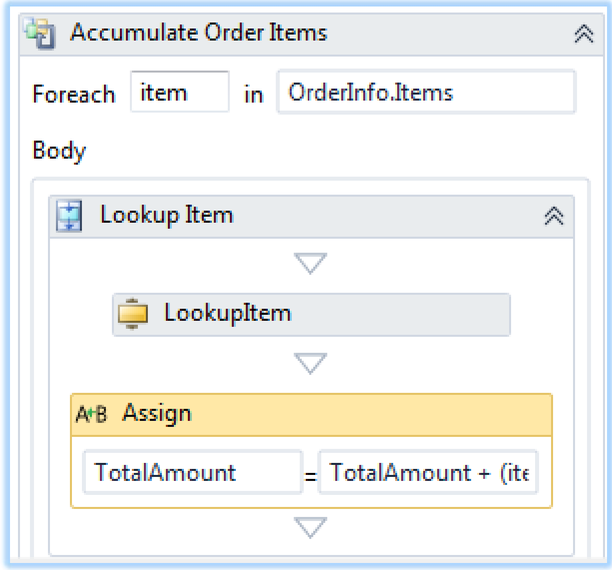


ნახ.17.44

ახლა ავირჩიოთ LookupItem ქმედება და მივუთითოთ Item თვისებაში ItemDetails. იგი აიღებს ItemInfo კლასისათვის მომხმარებლის ქმედებით დაბრუნებულ მნიშვნელობას და შეინახავს ItemDetails ცვლადში. გადმოვიტანოთ Assign ქმედება LookupItem–ის ქვემოთ. To თვისებისთვის შევიტანოთ TotalAmount, ხოლო Value თვისებისათვის გამოსახულება:

TotalAmount + (item.Quantity * ItemDetails.Price).

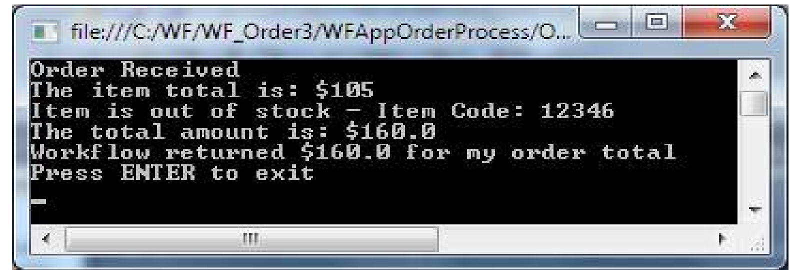
მივიღებთ 17.45 ნახაზზე ნაჩვენე სურათს.



ნახ.17.45

- აპლიკაციის ამუშავება

F5-ით ავამუშავოთ აპლიკაცია და მივიღებთ ასეთ შედეგს (ნახ.2.46).



ნახ.17.46

შედეგების სისწორე ($105) შეიძლება გადავამოწმოთ ხელით გადაანგარი-შებით (ნახ.17.47).

ItemCode	Quantity	Price	Ext. Price
12345	1	$10	$10
12346	3	$15	$45
12347	2	$25	$50
Total			$105

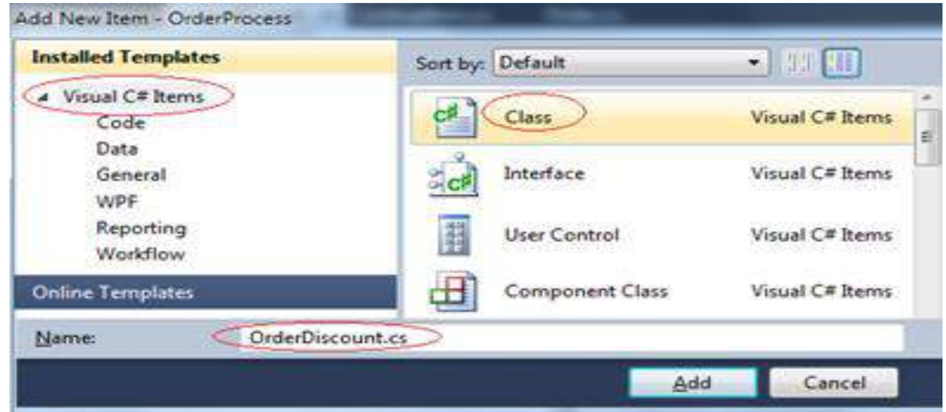
ნახ.17.47

➢ **InvokeMethod ქმედება**

აქტიურობა InvokeMethod არის კიდევ ერთი სასარგებლო ხერხი კოდის რეალიზაციისათვის სტანდარტული built-in ქმედების გარეთ. ამ ქმედების გამოყენება შესაძლებელია კლასის მეთოდის გამოსაძახებლად. კლასი არ უნდა იყოს სამუშაო პროცესის ნაწილი ან იყენებდეს სამუშაო პროცესის საბაზო კლასებს.

ამ პროექტში რეალიზებულია კლასი, რომელიც ანგარიშობს ფასდაკლების ზომას სხვადასხვა წესების საფუძველზე. ეს კლასი გამოიძახება (invoked) სამუშაო პროცესის მიერ ფასდაკლების საანგარიშოდ მითითებული შეკვეთისათვის.

- ფასდაკლების კლსის შექმნა (Discount Class)



ნახ.17.48

OrderDiscount.cs ფაილის ლისტინგი მოცემულია ქვემოთ:

// – – – ლისტინგი 17.11 – – –

```
using System;
using System.Collections.Generic;
namespace OrderProcess
{ public static class OrderDiscount
   { public static decimal ComputeDiscount(Order o, decimal total)
      {
```

```
        // შეკვეთილი ელემენტების რაოდენობის ანგარიში
        int count = 0;
        foreach (OrderItem i in o.Items)
          { count += i.Quantity; }
        // ფასდაკლების პროცენტის განსაზღვრა
        decimal pct = 0;
        if (total > 500)
            pct = (decimal)0.20;
        if (total > 200)
            pct = (decimal)0.15;
        if (total > 100)
            pct = (decimal)0.10;
        // ფასდაკლების ჯამის ანგარიში
        decimal discount = total * pct;
        // დოლარის დაკლება ყოველ შეკვეთილ ელემენტზე
        discount -= (decimal)count;
        // შემოწმება, რომ ის არაა ნულზე ნაკლები
        if (discount < 0)
            discount = 0;
        Console.WriteLine("გაანგარიშებული ფასდაკლება: ${0}",
                                                     discount.ToString());
        return discount;
       }
   }
}
```

ComputeDiscount() მეთოდი იღებს ორ პარამეტრს: Order კლასი და item total (ელემენტების ჯამი). იგი აბრუნებს უკან ფასდაკლების რაოდენობას, რომელიც გამოიყენება მოცემული შეკვეთისათვის. ფასდაკლების ლოგიკა აქ გამოყენებულია რაიმე წესების გარეშე, თავისუფლად. ალგორითმულად ჯერ განისაზღვრება ფასდაკლების პროცენტი შეკვეთის საერთო ჯამიდან. შემდეგ იგი აკლებს 1 $-ს ყოველი ელემენტისათვის.

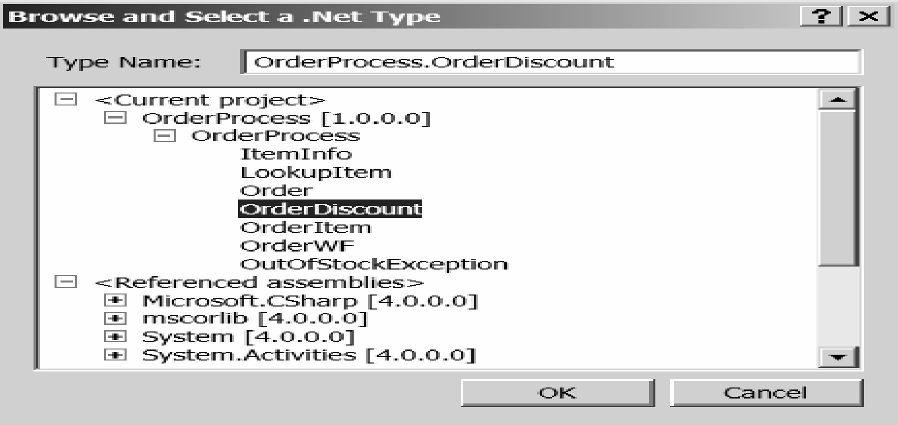
F6-ით დავაკომპილიროთ პროგრამა.

- **InvokeMethod ქმედების გამოყენება**

ჩვენ პროექტში გამოვიყენოთ InvokeMethod ქმედება ComputeDiscount() მეთოდის შესასრულებლად. გადმოვიტანოთ InvokeMethod აქტიურობა სამუშაო პროცესის სქემაზე, უშუალოდ “Check Stock” ქმედების წინ. DisplayName თვისებაში შევიტანოთ Calculate Discount.

- მიზნობრივი ობიექტის განსაზღვრა (Specifying the Target Object)

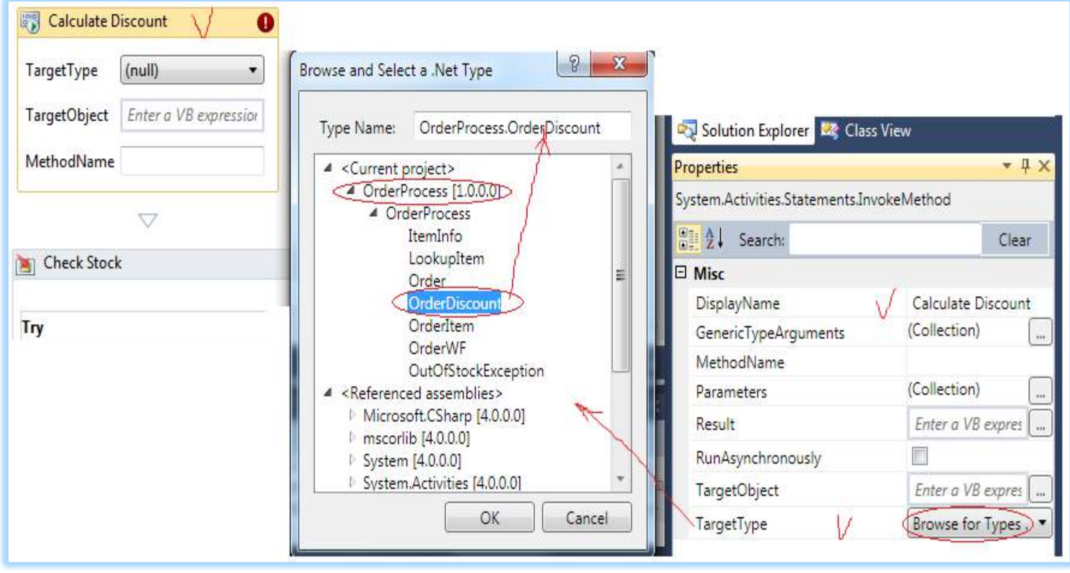
TargetType თვისება განსაზღვრავს კლასს, რომელიც შეიცავს გამოსაძახებელ მეთოდს. ჩამოშლად სიაში ტიპისათვის ავირჩიოთ Browse. დიალოგურ ფანჯარაში გავხსნათ OrderProcess ნაკრები და ავირჩიოთ OrderDiscount კლასი (ნახ.17.49).



ნახ.17.49. OrderDiscount კლასის არჩევა

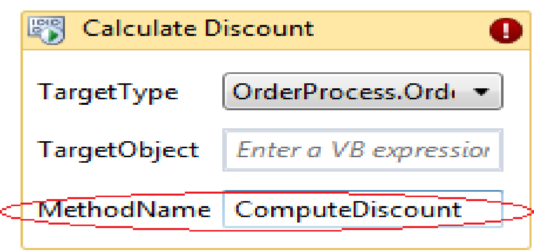
რჩევა: შეიძლება ორი მეთოდის გამოყენება, იმის მისა-თითებლად, რომ ობიექტი შეიცავს მეთოდს, რომელიც იქნება გამოძახებული.

1. თუ მეთოდი სტატიკურ კლასშია, მაშინ შეიძლება მიეთითოს მხოლოდ TargetType თვისება, როგორც ჩვენ მაგალითშია (ნახ.17.50);



ნახ.17.50

2. თუ კლასი არაა სტატიკური, მაშინ უნდა განისაზღვროს ამ ობიექტის კონკრეტული ეგზემპლარი. ამისთვის უკეთესი ხერხია კლასის ტიპის ცვლადის განსაზღვრა. მაშინ TargetObject თვისებაში უნდა მიეთითოს ცვლადის სახელი. თუ არსებობს რამდენიმე ეგზემპლარი და საჭიროა კონტროლი, თუ რომელია გამოყენებაში, მაშინ შეიძლება ცვლადის დაყენება ან Assign ქმედებთ ან მომხმარებლის ქმედებით (custom activity) მანამ, სანამ შესრულდება InvokeMethod ქმედება. თუ მიეთითება TargetObject თვისება, მაშინ აღარაა საჭირო TargetType თვისების მითითება. MethodName თვისებისათვის შევტანოთ ComputeDiscount (ნახ.17.51).



ნახ.17.51

XVIII თავი
კორპორაციის პროგრამული სისტემის დაპროექტება UML/2 და Workflow ტექნოლოგიებით

18.1. საპრობლემო სფეროს სისტემის დაპროექტება WF ტექნოლოგიით .NET პლატფორმაზე

განვიხილოთ საპრობლემო სფეროს ბიზნესპროცესების აქტიურობათა დიაგრამის მოდელირების ამოცანა UML-ენის ბაზაზე და მისი შემდგომი პროგრამული რეალიზაციისთვის ახალი ტექნოლოგიის (WF) გამოყენება.

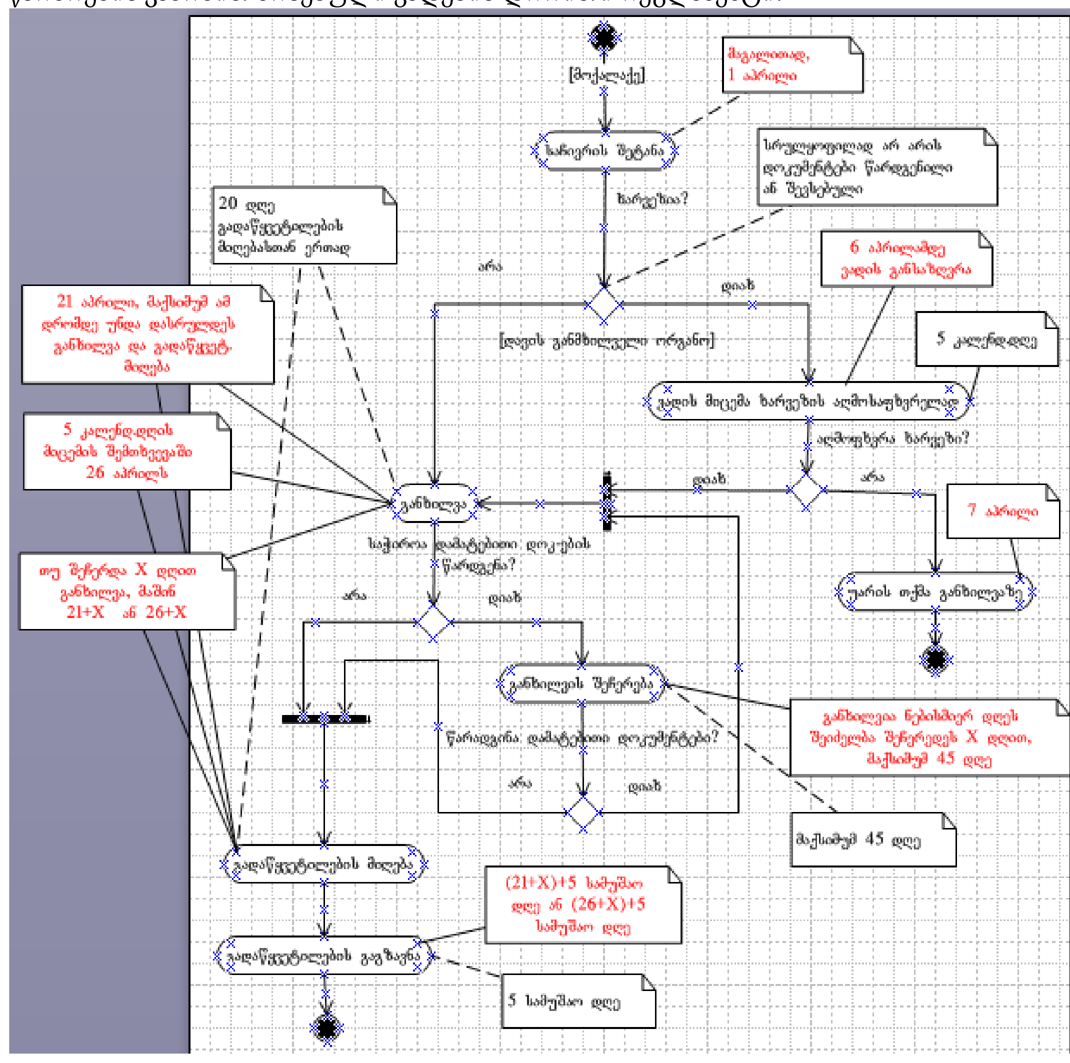
შემოთავაზებულია Workflow Foundation ტექნოლოგიის გამოყენება ბიზნეს-პროცესებისა და ბიზნესწესების ვიზუალური დაპროგრამებისათვის. საპრობლემო სფეროს კონკრეტული მაგალითი განიხილება შემოსავლების სამსახურის, კერძოდ საბაჟო სამართალდარღვევის ბიზნესპროცესების მაგალითზე [8]. შემოსავლების სამსახურის აუდიტისა და საბაჟო დეპარტამენტების ფუნქციებიდან ერთ-ერთი მნიშვნელოვანი საკითხია საგადასახადო სამართალდარღვევების ბიზნესპროცესების მართვა, კერძოდ, მათი გამოვლენა და საგადასახადო დავის წარმოება [10]. იმის მიხედვით, მომჩივანის მიერ საჩივარი დავის განმხილველ რომელ ორგანოში შეიტანება, საგადასახადო სამართალდარღვევის ოქმი და თანდართული მასალები შეიძლება შემოსავლების სამსახურის გარდა ფინანსთა სამინისტროს დავების განხილვის საბჭომ ან სასამართლომ განიხილოს [8]. საგადასახადო სამართალ-დარღვევათა ბიზნესპროცესი, რომელიც მოქალაქეთა საჩივრების სადავო საკითხების განხილვა-წარმოებას ეხება, შემდეგი ბიზნესწესებით იმართება:

- მოქალაქეს შეაქვს საჩივარი დავის განმხილველ ორგანოში (თარიღი ფიქსირდება);
- თუ საჩივარი არ აკმაყოფილებს პროცედურულ მოთხოვნებს, მომჩივანს წერილობით ეცნობება ამის შესახებ და მიეცემა არანაკლებ 5 დღე არსებული ხარვეზის გამოსასწორებლად;
- დავის განმხილვევლ ორგანოს უფლება აქვს მომჩივანის მოტივირებული მოთხოვნით გამოასწოროს ხარვეზი;
- დავის განმხილველი ორგანო საჩივარს განიხილავს 20 დღის ვადაში;
- დამატებითი ინფორმაციის ან/და დოკუმენტაციის მოპოვების საფუძვლით საჩივრის განხილვის შეჩერების ხანგრძლივობა არ უნდა აღემატებოდეს 45 დღეს;
- დავის განმხილველი ორგანოს გადაწყვეტილების დამოწმებული ასლი, მიღებიდან 5 სამუშაო დღეში ეგზავნება მხარეებს.

ნაშრომში განიხილება აღნიშნული ამოცანების მოდელირების, დაპროექტებისა და პროგრამული რეალიზაციის საკითხები უნიფიცირებული მოდელირების ენის (UML) და ახალი Workflow Foundation ტექნოლოგიის გამოყენებით [12,13].

Workflow Foundation ტექნოლოგია .NET Framework 4.0/4.5 -ში არის სრულიად ახალი პარადიგმა სამუშაო პროცესებზე (workflow) ბაზირებული აპლიკაციების ასაგებად. იგი ფუნდამენტურად ახლად გააზრებული ტექნოლოგიაა. აქ ჩვენ განვიხილავთ კონკრეტული ამოცანის, კერძოდ, საბაჟო სამართალდარღვევების სამუშაო პროცესის აქტიურობათა დიაგრამიდან პროგრამული კოდის დაპროექტების პროცესს.

18.1 ნახაზზე ნაჩვენებია სამართალდარღვევის სადავო საკითხების ბიზნესპროსეცებისა და ბიზნესწესების აქტიურობათა დიაგრამა, აგებული MsVisio, ინსტრუმენტის გამოყენებით [13]. ბიზნესპროცესების სქემაზე ცალკეულ ქმედებათა გარდა, ნაჩვენებია ბიზნესწესების კომენტარებიც, რომლებშიც კარგად ჩანს საქმის წარმოების კანონით არსებული ვადების დროითი რეგლამენტი.



ნახ.18.1

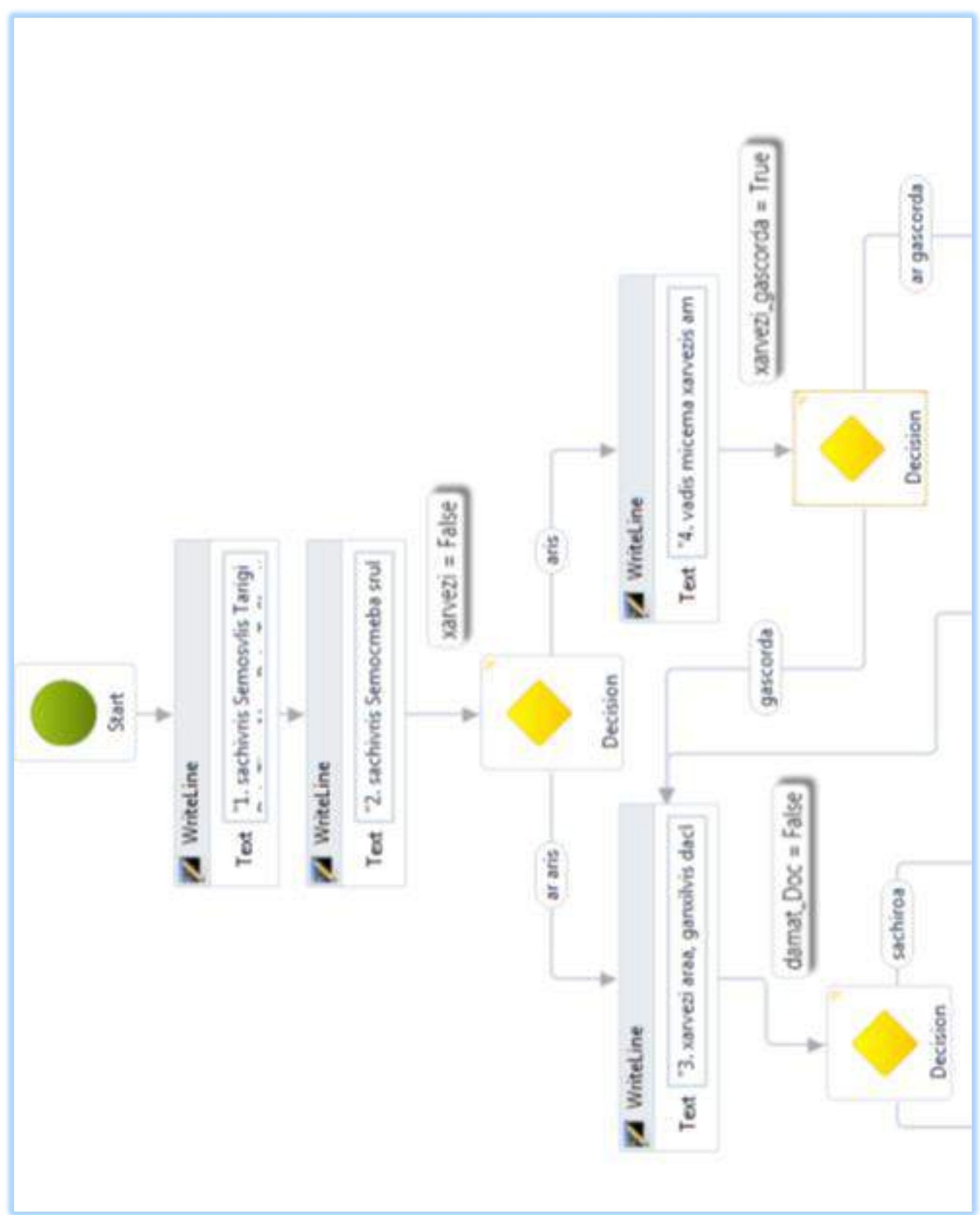
ახლა განვიხილოთ აგებული აქტიურობის დიაგრამის ასახვა Workflow Foundation ტექნოლოგიის სამუშაო გარემოში [5]. ესაა ჰიბრიდული (Windows+Web) სისტემა, რომლის კონსტრუირება (დიზაინი) ხდება შესაბამისი Workflow – ინსტრუმენტის ვიზუალური ელემენტებით (XAML ენაზე), ხოლო პროგრამის მუშაობის ლოგიკა, ჩვენს შემთხვევაში C#.NET კოდით.

18.2 ნახაზზე ნაჩვენებია დასმული ამოცანის გადაწყვეტის ალგორითმის ერთი ვარიანტი Workflow-ინსტრუმენტით. სქემაზე განთავსებულია შემდეგი ქმედებები (აქტიურობები): Flowchart, FlowDecission, If, Switch, Sequence, Assign და სხვ. [13]

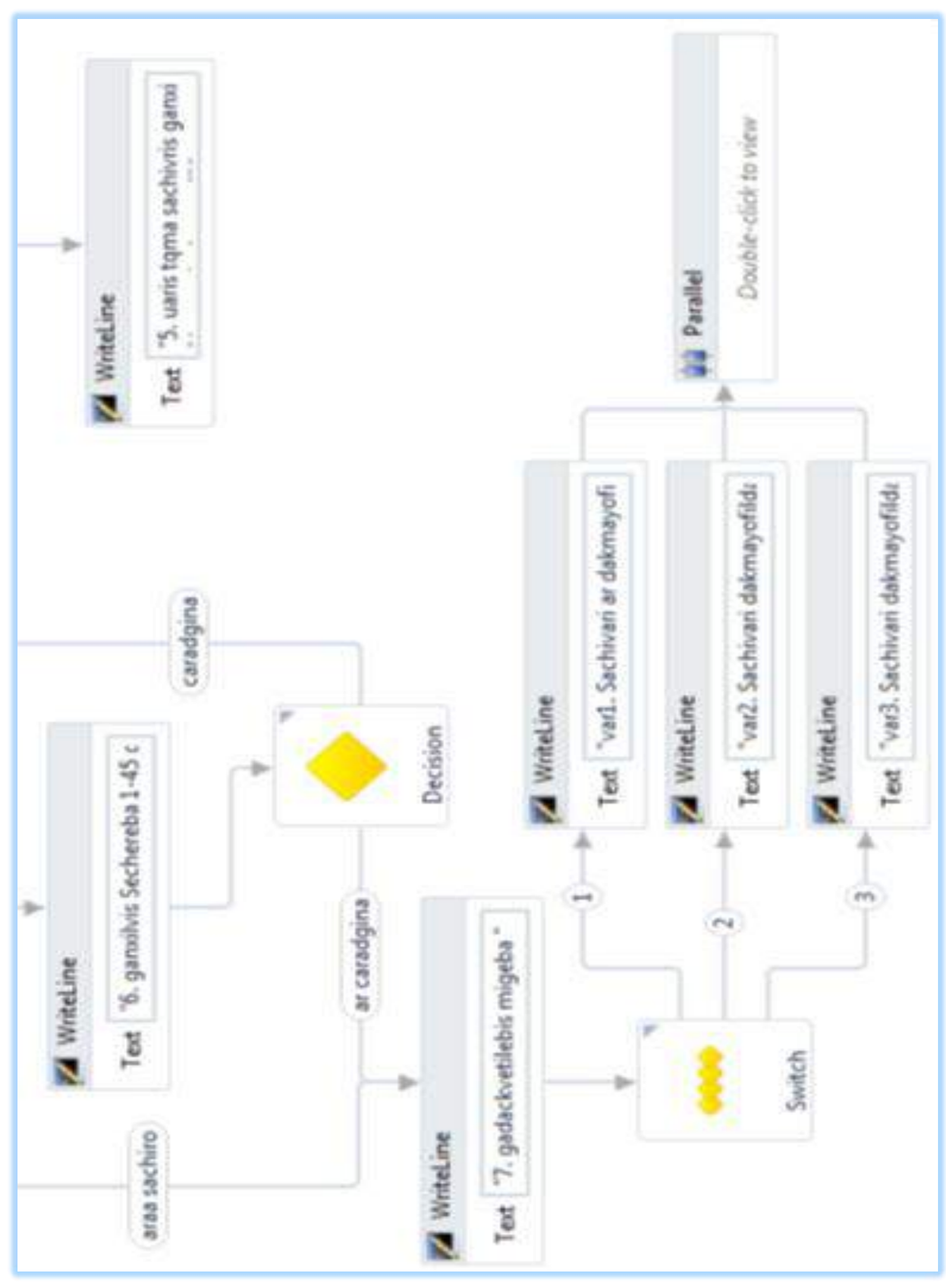
ბიზნესპროცესის დიაგრამა (Flowchart Workflow) გამოიყენებს აქტიურობათა დიაგრამას. ქმედებათა ეს დიაგრამა მუშაობს როგორც ბლოკ–სქემა, ქმედებათა სახეები დაკავშირებულია ერთმანეთთან გადაწყვეტილების ხეებით (decision trees). ქმედებათა მიმდევრობითობის გამოყენებით, შვილი–პროცესები სრულდება მიმდევრობით ზემოდან-ქვევით (top-down). ამის მიუხედავად, შვილი-ქმედებები შეიძლება შესრულდეს ნებისმიერი მიმდევრობით, გადაწყვეტილების მიმღები პირის მიერ. ძირითადი განსხვავება Flowchart ქმედებასა და Sequence ქმედებას შორის ისაა, თუ როგორაა დაკავშირებული შვილი-აქტიურობები. ქმედებათა დამატებისას მიმდევრობითობაში ისინი სრულდება დაღმავალი (top-down) მიმდევრობით.

აქ შესაძლებელია მიმდევრობის კონტროლი (მართვა), ქმედებათა გადაადგილებით, ოღონდაც ისინი ყოველთვის გასწორებულია ვერტიკალში და განაწილებულია თანაბრად. ისრები ქმედებებს შორის კი აგებულია ავტომატურად. Flowchart აქტიურობით შესაძლებელია ქმედებათა განთავსება პალიტრის ნებისმიერ ადგილას. მნიშვნელოვანია ისიც, რომ აქ ისრები ხელით უნდა შესრულდეს, ხოლო შეერთება დასაშვებია უკან, წინა ქმედებასთანაც.

გადაწყვეტილების ნაკადის C ქმედება სქემაზე გამოიყურება „ყვითელი ალმასის“ სახით, როგორც განშტოების სიმბოლო ნორმალურ ბლოკ–სქემაში, Properties-ფანჯარაში შეიტანება მდგომარეობა (Condition), მაგალითად, “ხარვეზი = False”. მაუსის კურსორის მიტანით FlowDecision ქმედებაზე ეს წარწერა გამოჩნდება.



ნახ.18.2. Workflow -დიაგრამის ფრაგმენტი
Visual Studio .NET Framework 4.0/5 გარემოში



ნახ.18.2. გაგრძელება

18.2. საპრობლემო სფეროს სისტემის პროგრამული რეალიზაცია WF ტექნოლოგიით

WF 4.0/5 ვერსიას აქვს რიგი პროცედურული ელემენტების: If, While, Assign, Sequence და სხვა. მაგალითად, ხარვეზების გასწორების პროცესის შედეგი განშტოვდება „გასწორდა“ ან „არ გასწორდა“:

if(xarvezi_gascorda = True) t2=t1+5;
else Console.WriteLine("Sachivari ar ganixileba");

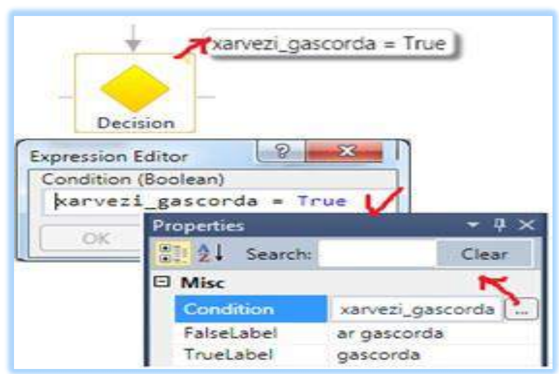
18.3 ნახაზზე ნაჩვენებია ცვლადების ცხრილი ჩვენი ალგორითმისთვის, ხოლო 18.4 ნახაზზე პირობის FlowDecision ქმედებაში მდგომარეობის (Condition) მნიშვნელობის განსაზღვრა.

Name	Variable type	Scope	Default
xarvezi	Boolean	Flowchart	False
xarvezi_gascorda	Boolean	Flowchart	True
damat_Doc	Boolean	Flowchart	False
Varianti	Int32	Flowchart	1
p1	Int32	Flowchart	0
p2	Int32	Flowchart	0
p3	Int32	Flowchart	0
p4	Int32	Flowchart	0
Create Variable			

Variables Arguments Imports

ნახ.18.3. Workflow -ცვლადების აღწერა

Switch ქმედება მუშაობს ისე, როგორც Switch ოპერატორი C# ენაში. ის საშუალებას იძლევა შესრულდეს sequence ქმედებები გამოსახულებათა ანალიზის საფუძველზე. Switch აქტიურობა გამოიყენება გადაწყვეტილების მიღების ვარიანტების განსაზღვრისათვისაც.



ნახ.18.4. FlowDecision -ის პირობის განსაზღვრა

Workflow სქემის აგების ამ ეტაპზე კონსოლის აპლიკაციის ამუშავებით მიიღება 18.5 ნახაზზე მოცემული შედეგები, რომელთა შემდგომი დაზუსტება და გაფართოებაა შესაძლებელი.

Workflow Foundation ტექნოლოგიის გამოყენებით ბიზნეს-პროცესების და ბიზნესწესების დაპროგრამება ხორციელდება ეფექტურად, შესაბამისი ვიზუალური კომპონენტების საფუძველზე. შედეგად საგრძნობლად მცირდება სისტემის დაპროექტების და მისი პროგრამული რეალიზაციის დრო.

```
file:///C:/WF/Sabajo/WF_AppSachivari/WF_AppSachivari/...
1. sachivris Semosvlis Tarigi : 5/6/2013
2. sachivris Semocmeba srulyofilebaze
3. xarvezi araa, ganxilvis dackeba :
7. gadackvetilebis migeba
var1. Sachivari ar dakmayofilda
cerili - Setkobineba
gadackvetilebis teqsti
gadackvetilebis migebis dasturi/xelmocera

Press ENTER to exit
```

ნახ. 18.5. შედეგები

VI ნაწილი

WCF ტექნოლოგია

XIX თავი. კომუნიკაცია აპლიკაციებს შორის WCF-ის ბაზაზე ……………......................... 567

XX თავი. კომუნიკაცია ჰოსტ აპლიკაციასთან 600

XXI თავი. Web - სერვისები 637

ბიზნესპროცესების შესრულებისას ერთ–ერთი მნიშვნელოვანი ასპექტია ურთიერთობა (კომუნიკაცია) აპლიკაციებს შორის, კლიენტებსა და სერვერებს, აგრეთვე სამუშაოპროცესებსა და ჰოსტდანართებს შორის. აქ გავეცნობით, თუ როგორ შეიძლება ბიზნესპროცესების გამოყენებით გამარტივდეს და კოორდინაცია გაეწიოს კომუნიკაციათა სხვადასხვა სცენარს.

აპლიკაციის სახით განიხილება პროექტების აგება უნივერსიტეტის, ბიბლიოთეკის, პროდუქციის წარმოების, ბიზნესის და სხვა სფეროში, რომლებშიც ინფორმაციის („მოთხოვნა-პასუხის“) გადაცემა მათ ფილიალებს ან სტრუქტურულ ერთეულებს შორის ხორციელდება კლიენტ-სერვერული და სერვისორიენტირებული პრინციპებით.

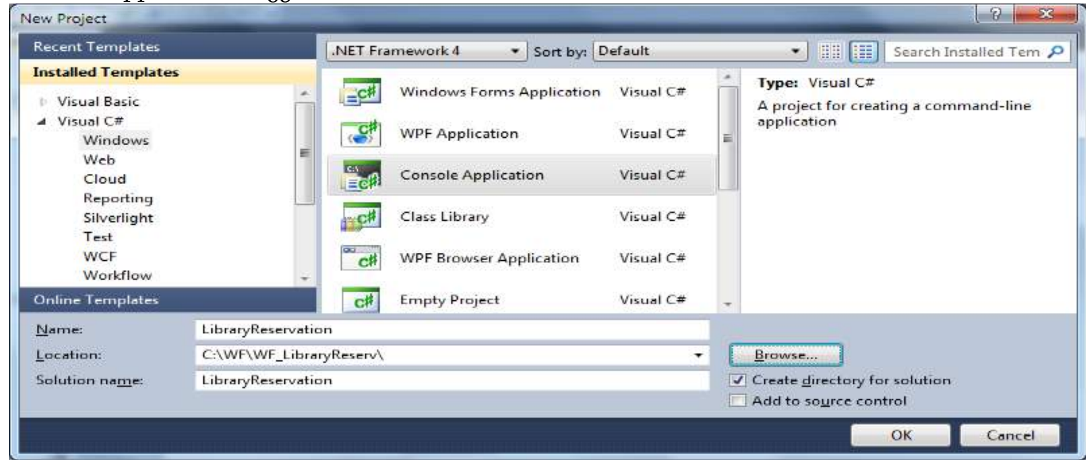
XIX თავი
კომუნიკაცია აპლიკაციებს შორის WCF-ის ბაზაზე
19.1. ინფორმაციის „გადაცემის“ და „მიღების“ ქმედებები

განვიხილოთ Visual Studio.NET პაკეტის WCF-ის სამუშაო გარემო და პირველი აპლიკაციის აგების მაგალითი Send და Receive ქმედებების დაპროგრამება და მათი გამოყენება.

მთავარი ქმედებები, რომლებიც კომუნიკაციისთვის გამოიყენება არის Send (გაგზავნა) და Receive (მიღება). ეს ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply) გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და სამეთვალყურეოდ. ამ თავში ავაგებთ კონსოლის მარტივ აპლიკაციას, რომელიც გამოიყენებს კომუნიკაციას იმავე აპლიკაციის სხვა ბიზნესპროცესთან (ბიბლიოთეკის მაგალითზე) [63].

19.1.1. ახალი პროექტის შექმნა

შევქმნათ ახალი პროექტი LibraryReservation სახელით, ჩვეულებრივ Windows -> Console Application რეჟიმში.

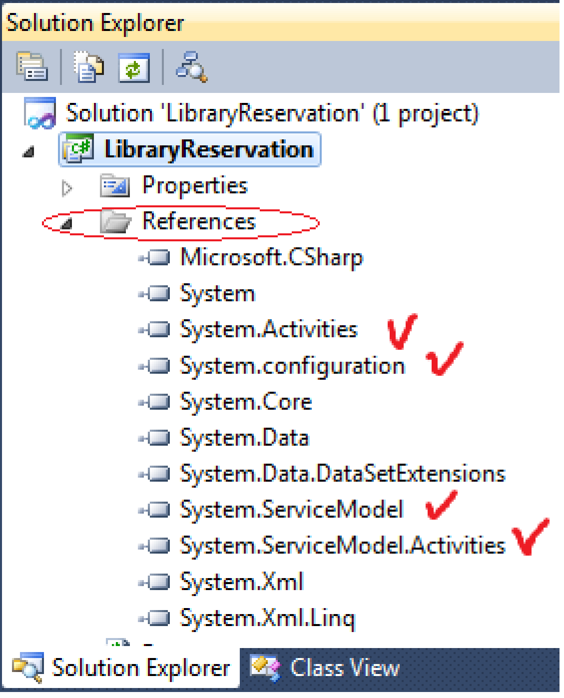


ნახ.19.1

LibraryReservation პროექტის Solution Explorer –ში, მაუსის მარჯვენა ღილაკით ავირჩიოთ Add Reference და .NET tab –დან დავამატოთ შემდეგი სტრიქონები.

- System.Activities
- System.Configuration
- System.ServiceModel
- System.ServiceModel.Activities

მიიღება:



ნახ.19.2

19.1.2. შეტყობინებათა განსაზღვრა

საჭიროა შეიქმნას კლასი, რომელიც განსაზღვრავს შეტყობინებებს აპლიკაციებს შორის. Solution Explorer-იდან Add -> Class და ჩავწეროთ კლასის სახელი Reservation.cs. ამ ფაილში დავამატოთ სახელსივრცეები (namespaces):

```
using System.Runtime.Serialization;
using System.ServiceModel;
```

Reservation.cs ფაილში განვსაზღვროთ სამი კლასი:

- Branch: განსაზღვრავს **მონაცემებს** ბიბლიოთეკის ფილიალის **მდებარეობის შესახებ;**
- ReservationRequest: განსაზღვრავს ფილიალის **მოთხოვნას** სათაურის დაჯავშნით;
- ReservationResponse: განსაზღვრავს პასუხს მოთხოვნის შესაბამის ფილიალისათვის.

Branch კლასის განსაზღვრა მოცემულია 19.1 ლისტინგში. შევიტანოთ იგი LibraryReference namespace–ს შიგნით. წავშალოთ ცარიელი კლასი Reservation, რომელიც წინა ბიჯზე შეიქმნა.

```
/****************ლისტინგი_ 19 .1 **********************************/
public class Branch
{
  public String BranchName { get; set;}
  public String Address { get; set; }
  public Guid BranchID { get; set; }
  #region Constructors
  public Branch()
    {
    }
  public Branch(String name, String address)
    {
      BranchName = name;
      Address = address;
      BranchID = Guid.NewGuid();
    }
 public Branch(String name, String address, Guid id)
   {
     BranchName = name;
     Address = address;
     BranchID = id;
   }
public Branch(String name, String address, String id)
   {
     BranchName = name;
     Address = address;
     BranchID = new Guid(id);
   }
#endregion Constructors
}
```

Branch კლასს აქვს სამი დასამახსოვრებელი წევრი": სახელი, ქსელის მისამართი და უნიკალური იდენტიფიკატორი. ზოგიერთი კონსტრუქტორი შემოტანილია გამოყენების გასამარტივებლად. აქ დამატებულია რეგიონის მარკერები კონსტრუქტორის ირგვლივ, შესაძლებელი რომ იყოს კოდის შეკუმშვა მისი კითხვადობის გასაუმჯობესებლად.

ახლა დავამატოთ ReservationRequest კლასის განსაზღვრება, ლისტინგი 19.2.

```
/********ლისტინგი_19.2   *****************************************/
[MessageContract(IsWrapped = false)]
```

```
public class ReservationRequest
 {
   private String_ISBN;
   private String_Title;
   private String_Author;
   private Guid_RequestID;
   private Branch_Requester;
   private Guid_InstanceID;
  #region Constructors
   public ReservationRequest()
     {
     }
   public ReservationRequest(String title, String author, String isbn,Branch requestor)
     {
      _Title = title;
      _Author = author;
      _ISBN = isbn;
      _Requester = requestor;
      _RequestID = Guid.NewGuid();
      }
    public ReservationRequest(String title, String author, String isbn,Branch requestor, Guid id)
      {
        _Title = title;
        _Author = author;
        _ISBN = isbn;
        _Requester = requestor;
        _RequestID = id;
      }
    #endregion Constructors
    #region Public Properties
   [MessageBodyMember]
    public String Title
     {
      get { return _Title; }
      set { _Title = value; }
     }
   [MessageBodyMember]
```

```
  public String ISBN
    {
      get { return _ISBN; }
      set { _ISBN = value; }
    }
  [MessageBodyMember]
  public String Author
   {
     get { return _Author; }
     set { _Author = value; }
   }
  [MessageBodyMember]
  public Guid RequestID
   {
     get { return _RequestID; }
     set { _RequestID = value; }
   }
  [MessageBodyMember]
  public Branch Requester
   {
    get { return _Requester; }
    set { _Requester = value; }
   }
  [MessageBodyMember]
  public Guid InstanceID
    {
      get { return _InstanceID; }
      set { _InstanceID = value; }
    }
  #endregion Public Properties
}
```

ReservationRequest კლასი შედგება ISBN, Title და Author წევრებისაგან მოთხოვნილი წიგნის აღსაწერად. იგი შეიცავს ასევე Branch კლასს, რომელიც ასახავს მოთხოვნილი წიგნის ფილიალს.

19.1.3. კონტრაქტის შეტყობინება

ვინაიდან ReservationRequest კლასი გამოყენებულ იქნება გამომავალი შეტყობინების განსაზღვრისათვის, MessageContract ატრიბუტი მიუთითებს, რომ ეს კლასი ჩართულ იქნება SOAP ზარათში. SOAP–ის გამოყენების დროს შეტყობინებები გადაიცემა XML–ის მსგავსი ფორმატირებადი ენით. ეს უზრუნველყოფს კლიენტებსა და სერვერს შორის მაღალი ხარისხის ურთიერთქმედების პლატფორმას. SOAP არის სტანდარტული პროტოკოლი, რომლის მხარდაჭერაც აქვს WCF –ს.

არსებობს აგრეთვე MessageBodyMember ატრიბუტი მის ყოველ პუბლიკ–თვისებაზე. ეს აუცილებელია WCF–ფენისთვის რათა სწორად დაფორმატდეს SOAP შეტყობინება.

ახლა შევიტანოთ რეალიზაციის კოდი ReservationResponse კლასისითვის, რომელიც 19.3 ლისტინგზეა მოცემული.

```
/***********ლისტინგი_19.3 ****************************************/
[MessageContract(IsWrapped = false)]
public class ReservationResponse
  {
    private bool _Reserved;
    private Branch _Provider;
    private Guid _RequestID;
   #region Constructors
    public ReservationResponse()
      {
       }
    public ReservationResponse(ReservationRequest request, bool reserved,Branch provider)
      {
         _RequestID = request.RequestID;
         _Reserved = reserved;
         _Provider = provider;
       }
    #endregion Constructors
     #region Public Properties

  [MessageBodyMember]
  public bool Reserved
   {
     get { return _Reserved; }
```

```
    set { _Reserved = value; }
  }
 [MessageBodyMember]
  public Branch Provider
    {
      get { return _Provider; }
      set { _Provider = value; }
    }
  [MessageBodyMember]
   public Guid RequestID
    {
      get { return _RequestID; }
      set { _RequestID = value; }
    }
     #endregion Public Properties
}
```

ReservationResponse კლასი შეიცავს ლოგიკურ ელემენტს (Reserved), რომელიც მიუთითებს, იყო თუ არა წიგნი ხელმისაწვდომი შესაკვეთად. ის შეიცავს ასევე Branch კლასს, რომელიც ასახავს იმ ფილიალს, რომელმაც შესარულა ეს მოთხოვნა.

19.1.4. სერვისის კონტრაქტი

WCF–ის ბოლო წერტილის განსაზღვრისათვის არსებობს ინფორმაციის სამი პორცია, რომლებიც მითითებულ უნდა იქნას: მიერთება (binding), მისამართი და კონტრაქტი.

მიერთება მიუთითებს იმ პროტოკოლს, რომელიც გამოიყენება (მაგ.,HTTP, TCP ან სხვ.).

მისამართი მიუთითებს, სად უნდა ვიპოვოთ ბოლო წერტილი, და მისამართის ტიპს, რომლის გამოყენება დამოკიდებულია მიერთებაზე. მაგალითად, HTTP–მიერთებისას უნდა მიეთითოს URL, ხოლო TCP–თვის მისამართი იქნება სერვერის სახელი ან IP–მისამართი.

კონტრაქტი განისაზღვრება ServiceContract–ით, რომელიც არის ინტერფეისი. იგი განსაზღვრავს მეთოდებს, რომლებიც მიწვდომადია ბოლო წერტილში.

ამგვარად, ჩვენ განვსაზღვრეთ შეტყობინებები, რომლებიც გადაიცემა პარამეტრების სახით სერვის–მეთოდების მიერ.

ახლა დავამატოთ ინტერფეისის განსაზღვრება, რომელიც 19.4 ლისტინგზეა მოცემული იმავე Reservation.cs ფაილში.

```
/***********ლისტინგი_19.4 ***************************************/
// Define the service contract, ILibraryReservation
// which consists of two methods, RequestBook() and RespondToRequest()
/**********************************************************************/
[ServiceContract]
   public interface ILibraryReservation
        {
         [OperationContract]
         void RequestBook(ReservationRequest request);
         [OperationContract]
         void RespondToRequest(ReservationResponse response);
        }
```

RequestBook() მეთოდი გამოძახებულ იქნება კლიენტის მიერ ReservationRequest შეტყობინების გადასაცემად სერვერზე. ამასთანავე RespondToRequest() მეთოდი გააგზავნის ReservationResponse შეტყობინებას უკან – კლიენტისაკენ.

დავაკომპილიროთ პროექტი (F5) და გავმართოთ კოდი.

19.1.5. აპლიკაციის კონფიგურაცია

როგორც ადრე ვთქვით, ჩვენ გვექნება ამ **აპლიკაციის რამდენიმე კოპიო,** რომლებიც ასახავს **სხვადასხვა ფილიალი**ს განთავსებას. ფილიალთა მონაცემები ინახება კონფიგურაციის ფაილებში. აპლიკაციის ყოველ კოპიოს ექნება თავისი კონფიგურაციის ფაილი, რომელიც შეიცავს თავის სპეციფიკურ ატრიბუტებს.

LibraryReservation–პროექტის Solution Explorer–დან ვირჩევთ Add New Item ➤. დიალოგურ ფანჯარაში General ჯგუფში ვირჩევთ Application Configuration File (ნახ.19.3). ფაილის სახელი ავტომატურად არის App.config ().

კონფიგურაციის ფაილში შევიტანოთ საჭირო მონაცემები 19.5 ლისტინგის მსგავსად.

```
<!— ლისტინგი_19.5 ———— –>
<?xml version="1.0" encoding="utf-8" ?>
  <configuration>
    <appSettings>
      <add key="Branch Name" value="Central Library"/>
      <add key="ID" value="{43E6DADD-4751-4056-8BB7-7459B5C361AB}"/>
```

```
        <add key="Address" value="8000"/>
        <add key="Request Address" value="8730"/>
      </appSettings>
    </configuration>
```

AppSettings სექციას აქვს მნიშვნელობები ფილიალის სახელი, ID (უნიკალური იდენტიფიკატორი) და მისამართი (პორტის ნომერი, რომელსაც აპლიკაცია იყენებს). მოთხოვნის მისამართი განსაზღვრავს პორტის ნომერს, საითაც იქნება მოთხოვნები გაგზავნილი. შესაძლებელია თუ ეს საჭიროა სხვა პორტების გამოყენებაც.

19.1.6. ბიზნესპროცესების განსაზღვრა

შემდეგი ბიჯი არის კლიენტისა და სერვერის სამუშაო პროცესების განსაზღვრა (Defining the Workflows). მათ შორის კომუნიკაცია ხორციელდება Send და Receive ქმედებათა გამოყენებით. ამ პროექტში ჩვენ გამოვიყენებთ Workflow–ის კოდირებას, ნაცვლად დიზაინერისა .xaml ფაილის გენერაციისათვის. კოდირების ეს საკითხი მე–2 თავში იყო განხილული.

LibraryReservation–პროექტის Solution Explorer–ში დავამატოთ კლასი: Add->Class. შევიტანოთ სახელი ReservationWF.cs და ჩავამატოთ სახელსივრცეები:

```
using System.Activities;
using System.Activities.Statements;
using System.ServiceModel.Activities;
using System.ServiceModel;
```

19.1.7. „კლიენტი – მოთხოვნების“ გაგზავნა

პირველ რიგში უნდა განისაზღვროს სამუშაო პროცესი (workflow), რომლითაც მოთხოვნა გადაეცემა სხვა ფილიალს. ReservationWF კლასის კოდი შევცვალოთ 19.6 ლისტინგის მიხედვით.

```
public sealed class SendRequest : Activity
  {
    // შესატან და გამოსატან არგუმენტთა განსაზღვრა
    public InArgument<string> Title { get; set; }
    public InArgument<string> Author { get; set; }
    public InArgument<string> ISBN { get; set; }
    public OutArgument<ReservationResponse> Response { get; set; }
    public SendRequest()
    {
```

```
    // ცვლადების განსაზღვრა ამ პროცესისათვის
    Variable<ReservationRequest> request =
                    new Variable<ReservationRequest> { Name = "request" };
    Variable<string> requestAddress =
                    new Variable<string> { Name = "RequestAddress" };
  // გაგზავნის Send ქმედების განსაზღვრა
  Send submitRequest = new Send
   {
     ServiceContractName = "ILibraryReservation",
     EndpointAddress = new InArgument<Uri>
          (env => new Uri("http://localhost:" + requestAddress.Get(env) +
             "/LibraryReservation")),
     Endpoint = new Endpoint
      {
        Binding = new BasicHttpBinding()
      },
     OperationName = "RequestBook", Content = SendContent.Create
                    (new InArgument<ReservationRequest>(request))
    };
    //  SendRequest პროცესის განსაზღვრა
  }
}
```

ამ ბიზნესპროცესს აქვს სამი შემავალი არგუმენტი: Title, Author და ISBN, რომლებიც განსაზღვრავს მოთხოვნილ წიგნს. აგრეთვე აქვს გამომავალი არგუმენტი (Response), რომელიც არის საპასუხო შეტყობინება. იგი ადრე იყო განსაზღვრული ReservationResponse კლასით.

კონსტრუქტორი აგრეთვე განსაზღვრავს ზოგიერთ ლოკალურ ცვლადს. მათი გადაცემა არ ხდება ბიზნესპროცესიდან (ან –ში), ისინი გამოიყენება შიგა სამუშაო პროცესების ქმედებებით. მოთხოვნილი ცვლადი შეიცავს გამომავალ შეტყობინებას, რომელიც არის ReservationRequest კლასი. RequestAddress ცვლადი შეინახავს პორტის ნომერს, რომელიც იმ ფილიალისაა, რომელიც ელოდება მოთხოვნის მიღებას.

შენიშვნა: კოდირებული სამუშაო პროცესისთვის (თავი 2), ყველა ქმედება შეიქმნა როგორც ჩადგმული ანონიმური კლასი. ეს კარგად მუშაობს ხშირ შემთხვევებში. Send ქმედება აქ შეიქმნა როგორც სახელმინიჭებული კლასი, ვინაიდან იგი საჭიროა მიმართვისათვის ReceiveReply ქმედებიდან. იგი არ განსხვავდება არგუმენტებისა და ცვლადებისაგან, რომლებიც შევქმენით. ისინი იქმნება სახელდებული კლასების სახით, რომლებიც მოგვიანებით იქნება გამოყენებული.

19.1.8. გაგზავნის ქმედება

submitRequest (მოთხოვნის გაგზავნის) ეგზემპლარი განისაზღვრება როგორც Send ქმედება. იგი იყენებს WCF–ს, რათა გადასცეს შეტყობინება მითითებულ ბოლო წერტილში. ინფორმაციის სამი ნაწილი გამოიყენება ბოლო წერტილის მისათითებლად:

- ServiceContractName მიეთითება როგორც IlibraryReservation;
- EndpointAddress მიეთითება როგორც URL ცვლადით (პორტის ნომერი);
- Binding მიეთითება BasicHttpBinding კლასით.

ამასთანავე არსებობს კიდევ რამდენიმე თვისება, რომლებიც უნდა განისაზღვროს. OperationName მიუთითებს სერვისის კონტრაქტის სპეციფიკურ მეთოდზე, რომელიც გამოძახებულ უნდა იქნას შეტყობინების მიღების დროს დანიშნულების ადგილას. Content თვისება შეინახავს მიმთითებელს შეტყობინებაზე (ReservationRequest კლასი), რომელიც უნდა გაიგზავნოს.

19.2. მომხმარებლის ქმედება

ახლა უნდა შევქმნათ მომხმარებლის ქმედება, რომელიც ააგებს შეტყობინებას მოთხოვნაზე ბიზნესპროცესით გადმოცემული არგუმენტების საფუძველზე.

LibraryReservation–პროექტის Solution Explorer–ში დავამატოთ კლასი: Add->Class. შევიტანოთ სახელი **CreateRequest.cs**, რომლის რეალიზაცია მოცემულია 19.7 ლისტინგში.

```
//——— ლისტინგი_19.7 ————————————
using System;
using System.Activities;
using System.Configuration;
namespace LibraryReservation
{
  /***************************************************/
  // ეს მომხმარებლის ქმედება ქმნის ReservationRequest კლასს შესატანი
  // პარამეტრებით (Title, Author და ISBN). ეს უზრუნველყოფილია
  // გამომავალი პარამეტრის მოთხოვნაში. იგი ასევე აბრუნებს ქსელის
  მისამართს
  // ფილიალისათვის, რომელსაც  უნდა გაეგზავნოს მოთხოვნა.
  /***************************************************/
  public sealed class CreateRequest : CodeActivity
  {
      public InArgument<string> Title { get; set; }
      public InArgument<string> Author { get; set; }
      public InArgument<string> ISBN { get; set; }
```

```
public OutArgument<ReservationRequest> Request { get; set; }
public OutArgument<string> RequestAddress { get; set; }
protected override void Execute(CodeActivityContext context)
  {
  // config ფაილის გახსნა და Request Address მიღება (get)
  Configuration config = ConfigurationManager
          .OpenExeConfiguration(ConfigurationUserLevel.None);
  AppSettingsSection app =
  (AppSettingsSection)config.GetSection("appSettings");
  //  ReservationRequest კლასის შექმნა და მისი შევსება შესატანი
  არგუმენტებით
  ReservationRequest r = new ReservationRequest
      (
        Title.Get(context),
        Author.Get(context),
        ISBN.Get(context),
        new Branch
          {
            BranchName = app.Settings["Branch Name"].Value,
            BranchID = new Guid(app.Settings["ID"].Value),
            Address = app.Settings["Address"].Value
          }
      );
 // მოთხოვნის მოთავსება OutArgument – ში
Request.Set(context, r);
// address–ის მოთავსება OutArgument –ში
RequestAddress.Set(context, app.Settings["Request Address"].Value);
  }
 }
}
```

Execute() მეთოდი პირველად ხსნის აპლიკაციის კონფიგ–ფაილს ფილიალის დეტალების დასადგენად, რომელსაც ესაჭიროება მოთხოვნის ფორმატირება. შემდეგ იგი ქმნის ReservationRequest კლასს ერთ-ერთი ჩვენ მიერ უზრუნველყოფილი კონსტრუქტორით. Branch კლასი, რომელიც არის კონსტრუქტორის ერთ-ერთი პარამეტრი, შექმნილია როგორც ანონიმური კლასი BranchName, BranchID და Address თვისებების მითითებით. შემდეგ ReservationRequest კლასი შეინახავს მოთხოვნის გამომავალ პარამეტრში.

კონფიგურაციის ფაილში Request Address შეიცავს ფილიალის მისამართს (პორტის ნომერს), რომელმაც უნდა მიიღოს მოთხოვნა. იგი შეინახება RequestAddress გამომავალ არგუმენტში, ვინაიდან ის დასჭირდება სამუშაო პროცესს.

დავუბრუნდეთ ReservationWF.cs ფაილს. დავამატოთ ბიზნესპროცესის განსაზღვრება 19.8 ლისტინგის შესაბამისად. ეს უნდა მოთავსდეს იქ, სადაც მითითებულია (// SendRequest სამუშაო პროცესის განსაზღვრა).

```
// ——— ლისტინგი_19.8 ——
// Define the SendRequest workflow
this.Implementation = () => new Sequence
{
    DisplayName = "SendRequest",
    Variables = { request, requestAddress},
    Activities =
     {
      new CreateRequest
        {
          Title = new InArgument<string>(env => Title.Get(env)),
          Author = new InArgument<string>(env => Author.Get(env)),
          ISBN = new InArgument<string>(env => ISBN.Get(env)),
          Request = new OutArgument<ReservationRequest>
          (env => request.Get(env)),
          RequestAddress = new OutArgument<string>
          (env => requestAddress.Get(env))
        },
      new CorrelationScope
        {
         Body = new Sequence
           {
            Activities =
             {
               submitRequest,
             new WriteLine
                {
                    Text = new InArgument<string>
                    (env => "Request sent; waiting for response"),
                },
          new ReceiveReply
```

```
                {
                    Request = submitRequest,
                    Content = ReceiveContent.Create
                    (new OutArgument<ReservationResponse>
                    (env => Response.Get(env)))
                 }
                }
              }
            },
            new WriteLine
             {
                Text = new InArgument<string>
                (env => "Response received from " +
                Response.Get(env).Provider.BranchName),
             },
            }
        };
```

ქმედების კლასის Implementation თვისება (მითითებით როგორც this. Implementation) შეიცავს შვილ ქმედებას. ამ შემთხვევაში იგი განისაზღვრება როგორც Sequence ქმედება, რომელიც შედგება შვილ ქმედებათა ერთობლიობისაგან. ამ ქმედებათა მიერ გამოყენებული ცვლადები უნდა იყოს გამოცხადებული. ესაა მოთხოვნა და requestAddress ცვლადები, რომლებიც განისაზღვრა კონსტრუქტორში.

პირველი ქმედება არის მომხმარებლის CreateRequest ქმედება, რომელიც ახლახანს ავაგეთ. მივაქციოთ ყურადღება, რომ როგორც ჩვენ მივუთითეთ თვისება, ეს იცის Intellisense–მ (შესატანი და გამოსატანი არგუმენტები, რომლებიც განვსაზღრეთ ჩვენს კლასში). Title, Author და ISBN არის სამუშაო პროცესის შემავალი არგუმენტები. Request და RequestAddress გამომავალი არგუმენტები ინახება სამუშაო პროცესის ცვლადებში.

CorrelationScope ქმედება ამატებს შემდეგს (next), რომელიც შედგება ქმედებათა მიმდევრობისაგან. კერძოდ, ის შეიცავს Send და ReceiveReply ქმედებებს, რომელთა მოთავსებით CorrelationScope ქმედებაში შესაძლებელი ხდება შემოსული მოთხოვნისა და საპასუხო შეტყობინების კორელაციის განსაზღვრა მოცემული ბიზნესპროცესისათვის.

WriteLine ქმედება ემატება Send ქმედების შემდეგ, რათა მიეთითოს, რომ მოთხოვნა იქნა გაგზავნილი.

19.2.1. პასუხის მიღების ქმედება

ReceiveReply ქმედება ასოცირდება გაგზავნის Send ქმედებასთან. იგი ელოდება პასუხს შეტყობინებაზე, რომელიც გაიგზავნა Send ქმედების მიერ. ამის რეალიზაციის მიზნით Request–ის (მოთხოვნის) თვისებას აქვს Send ქმედების სახელმინიჭებული ეგზემპლარის მნიშვნელობა (submitRequest).

თვისების შინაარსი განსაზღვრავს, სად ინახება საპასუხო შეტყობინება (ReservationResponse კლასი). ამით ყენდება სამუშაო ნაკადის Response გამომავალი არგუმენტი. იგი მისაწვდომი იქნება ჰოსტაპლიკაციისთვის, როცა Workflow–პროცესი დასრულდება.

19.2.2. სერვერი – მოთხოვნის დამუშავება

ახლა განვსაზღვროთ სამუშაო პროცესი, რომელიც სრულდება მოთხოვნის დასამუშავებლად სხვა ფილიალიდან. ReservationWF.cs ფაილში დავამატოთ კლასის განსაზღვრა 19.9 ლისტინგის შესაბამისად.

```
// ––– ლისტინგი_19.9 ––––––––––––––––––––––––––––––
public sealed class ProcessRequest : Activity
{
   public ProcessRequest()
    {
      // ცვლადების განსაზღვრა ამ workflow–ისთვის
      Variable<ReservationRequest> request =
                        new Variable<ReservationRequest> { Name = "request" };
      Variable<ReservationResponse> response =
                        new Variable<ReservationResponse> { Name = "response" };
      Variable<bool> reserved = new Variable<bool> { Name = "reserved" };
      Variable<CorrelationHandle> requestHandle =
                    new Variable<CorrelationHandle> { Name = "RequestHandle" };
      // Receive ქმედების შექმნა
      Receive receiveRequest = new Receive
         {
            ServiceContractName = "ILibraryReservation",
            OperationName = "RequestBook",
            CanCreateInstance = true,
            Content = ReceiveContent.Create
                        (new OutArgument<ReservationRequest>(request)),
            CorrelatesWith = requestHandle
         };
```

```
        // ProcessRequest workflow–ის განსაზღვრა
      }
    }
```

ამ სამუშაო პროცესს არ აქვს შემავალი და გამომავალი არგუმენტები. აქვს ოთხი ცვლადი, რომლებიც განსაზღვრულია. მოთხოვნის (request) ცვლადი ინახავს შემავალ შეტყობინებებს (ReservationRequest კლასი), ხოლო პასუხის (response) ცვლადი ინახავს გამომავალ პასუხებს (ReservationResponse კლასი). დარეზერვებული ცვლადი მიუთითებს, შეიძლება თუ არა სათაური (title) იყოს დაჯავშნული. RequestHandle გამოიყენება შემოსულ მოთხოვნისა და პასუხის კორელაციისათვის.

19.2.3. მიღების ქმედება

Receive ქმედების სახელმინიჭებული ეგზემპლარი (receiveRequest – მოთხოვნის მიღება) განსაზღვრულია. არაა საჭირო მიერთების ან მისამართის მითითება WCF შეტყობინების მიღების ბოლოს. მაგრამ უნდა განისაზღვროს სერვისის კონტრაქტი. ServiceContractName მიუთითებს, რომ IlibraryReservation სერვისის კონტრაქტი უნდა იქნას გამოყენებული და OperationName თვისება განსაზღვრავს RequestBook() მეთოდს.

CanCreateInstance დაყენებულია true-ში, ვინაიდან როცა ეს ქმედება სრულდება, იგი შექმნის პროცესის ახალ ეგზემპლარს. იგი მოითხოვს, რომ ეს ქმედება იყოს პირველი სამუშაო პროცესში. Content თვისება უნდა შეიცავდეს შემავალ შეტყობინებას და კონფიგურირდება ისე, რომ შეინახოს იგი მოთხოვნის ცვლადში. CorrelatesWith თვისება იყენებს requestHandle ცვლადს.

19.2.4. მომხმარებლის ქმედება – პასუხის შექმნა

სანამ განვსაზღვრავთ სამუშაო პროცესის ქმედებებს, საჭიროა მომხმარებლის ქმედება ReservationResponse კლასის შექმნისათვის. LibraryReservation პროექტის Solution Explorer–დან დავამატოთ ახალი კლასი სახელით: CreateResponse.cs, რომლის რეალიზაცია მოცემულია 1.10 ლისტინგზე.

```
// —— ლისტინგი_1.10 ——————
using System;
using System.Activities;
using System.Configuration;
namespace LibraryReservation
{
    /****************************************************/
    // ეს custom ქმედება ქმნის ReservationResponse კლასს
```

```
// საწყისი მოთხოვნა უზრუნველყოფილია როგორც InArgument,
// ასევე ლოგიკური (boolean) მითითებით, რომ მოთხოვნა
// დაკმაყოფილდა თუ არა. კლასი უზრუნველყოფილია
// Response–ში OutArgument–ით
public sealed class CreateResponse : CodeActivity
 {
    public InArgument<ReservationRequest> Request { get; set; }
    public InArgument<bool> Reserved { get; set; }
    public OutArgument<ReservationResponse> Response { get; set; }
    protected override void Execute(CodeActivityContext context)
     {
        // config ფაილის გახსნა
        Configuration config = ConfigurationManager
                  .OpenExeConfiguration(ConfigurationUserLevel.None);
        AppSettingsSection app =
                 (AppSettingsSection)config.GetSection("appSettings");
        // ReservationResponse კლასის შექმნა და შევსება
        ReservationResponse r = new ReservationResponse
         (
            Request.Get(context),
            Reserved.Get(context),
            new Branch
             {
                BranchName = app.Settings["Branch Name"].Value,
                BranchID = new Guid(app.Settings["ID"].Value),
                Address = app.Settings["Address"].Value
             }
         );
       // პასუხის შენახვა (Response) OutArgument–ში
  Response.Set(context, r);
 }
}
}
```

CreateResponse ქმედება ძალზე ჰგავს CreateRequest აქტიურობას. ჯერ იგი ხსნის აპლიკაციის კონფიგ–ფაილს, რათა მიიღოს ფილიალის შესახებ დეტალები. შემდეგ ReservationResponse კლასი იქმნება ერთ-ერთი მოცემული კონსტრუქტორით და შეინახება Response გამომავალ არგუმენტში.

დავბრუნდეთ ReservationWF.cs კლასში შევიყვანოთ სამუშაო პროცესის განსაზღვრება, როგორც 19.11 ლისტინგშია. (ეს უნდა ჩაემატოს //სამუშაო პროცესის განსაზღვრა ProcessRequest).

```
// —— ლისტინგი_19.11 ————————
// --- ბიზნესპროცესის ProcessRequest განსაზღვრა ---
this.Implementation = () => new Sequence
{
    DisplayName = "ProcessRequest",
    Variables = { request, response, reserved, requestHandle },
    Activities =
      {
       receiveRequest,
       new WriteLine
         {
           Text = new InArgument<string>(env => "Got request from: " +
                 request.Get(env).Requester.BranchName),
         },
      new WriteLine
        {
          Text = new InArgument<string>(env => "Requesting: " +
           request.Get(env).Title),
        },
        new Assign
        {
          To = new OutArgument<Boolean>(reserved),
          Value = new InArgument<Boolean>(env => true)
        },
        new Delay
         {
                Duration = TimeSpan.FromSeconds(2)
         },
        new CreateResponse
          {
            Request = new InArgument<ReservationRequest>(env =>
            request.Get(env)),
            Response = new OutArgument<ReservationResponse>
                (env => response.Get(env)),
```

```
                Reserved = new InArgument<bool>(env => reserved.Get(env)),
              },
            new WriteLine
              {
                Text = new InArgument<string>(env => "Sending response to: " +
                       request.Get(env).Requester.BranchName),
              },
            new SendReply
              {
                Request = receiveRequest,
                Content = SendContent.Create
                     (new InArgument<ReservationResponse>(response))
              }
        }
    };
```

ეს ოთხი ცვლადი, რომლებიც კონსტრუქტორშია განსაზღვრული, გამოცხადე-ბულია სამუშაო პროცესის კოდის ტანში. Receive ქმედება (receiveRequest) არის პირველი ქმედება სამუშაო პროცესში. იგი, ორ WriteLine ქმედებასთან თანხლებით: პირველს ეკრანზე გამოაქვს ფილიალის სახელი, რომელმაც მოთხოვნა გამოაგზავნა, და მეორე გვიჩვენებს სათაურს, (title) რომელიც მოითხოვება.

Assign ქმედება უბრალოდ აყენებს დარეზერვებულ ცვლადებს true-ში. ამ მაგალითში ჩვენ დავუშვით, რომ სათაური არის მიწვდომადი. Delay ქმედება აყოვნებს სამუშაო პროცესს 2 წამით (დამუშავების პროცესის იმიტაცია). შემდეგ მომხმარებლის CreateResponse ქმედება სრულდება ReservationResponse კლასის შექმნის მიზნით, რომელიც შენახულ იქნება response ცვლადში. ბოლო WriteLine ქმედება მიუთითებს, რომ პასუხი გაიგზავნა.

19.2.5. SendReply ქმედება

SendReply ქმედება ასოცირდება (კავშირშია) Receive ქმედებასთან. ეს ხორციელდება Request თვისების მითითებით როგორც მაჩვენებლისა Receive ქმედებაზე (receiveRequest). თვისების შინაარსი განსაზღვრავს შეტყობინებას, რომელიც იქნება გაგზავნილი უკან საპასუხოდ. ესაა პასუხის ცვლადის მნიშვნელობა.

ჩვენი სამუშაო პროცესი დასრულდა. საფინალო რეალიზაცია ReservationWF.cs ფაილისთვის მოცემულია 19.12 ლისტინგში.

```
// —— ლისტინგი_19.12 ——————
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Activities;
using System.Activities.Statements;
using System.ServiceModel.Activities;
using System.ServiceModel;
namespace LibraryReservation
  {
     /***************************************************/
     // ეს ფაილი შეიცავს ორი workflow–ის განსაზღვრებას.
     // SendRequest აინიციალიზებს ახალ მოთხოვნას
     // ProcessRequest ამუშავებს შემოსულ მოთხოვნას
     //
     /***************************************************/
     public sealed class SendRequest : Activity
     {
       // Define the input and output arguments
       public InArgument<string> Title { get; set; }
       public InArgument<string> Author { get; set; }
       public InArgument<string> ISBN { get; set; }
       public OutArgument<ReservationResponse> Response { get; set; }
       public SendRequest()
         {
           // Define the variables used by this workflow
           Variable<ReservationRequest> request =
           new Variable<ReservationRequest> { Name = "request" };
           Variable<string> requestAddress =
           new Variable<string> { Name = "RequestAddress" };
           // Define the Send activity
           Send submitRequest = new Send
             {
               ServiceContractName = "ILibraryReservation",
               EndpointAddress = new InArgument<Uri>
               (env => new Uri("http://localhost:" + requestAddress.Get(env) +
               "/LibraryReservation")),
               Endpoint = new Endpoint
                {
```

```
            Binding = new BasicHttpBinding()
          },
          OperationName = "RequestBook",
          Content = SendContent.Create
          (new InArgument<ReservationRequest>(request)),
        };
    // Define the SendRequest workflow
    this.Implementation = () => new Sequence
      {
          DisplayName = "SendRequest",
          Variables = { request, requestAddress},
          Activities =
           {
             new CreateRequest
               {
                 Title = new InArgument<string>(env => Title.Get(env)),
                 Author = new InArgument<string>(env => Author.Get(env)),
                 ISBN = new InArgument<string>(env => ISBN.Get(env)),
                 Request = new OutArgument<ReservationRequest>
                 (env => request.Get(env)),
                 RequestAddress = new OutArgument<string>
                 (env => requestAddress.Get(env))
               },
            new CorrelationScope
              {
                  Body = new Sequence
              {
                Activities =
                 {
                 submitRequest,
                 new WriteLine
                    {
                     Text = new InArgument<string>
                     (env => "Request sent; waiting for response"),
                    },
                new ReceiveReply
                    {
```

```
                        Request = submitRequest,
                        Content = ReceiveContent.Create
                        (new OutArgument<ReservationResponse>
                        (env => Response.Get(env)))
                    }
                }
            }
        },
        new WriteLine
        {
            Text = new InArgument<string>
            (env => "Response received from " +
            Response.Get(env).Provider.BranchName),
        },
    }
  };
 }
}
public sealed class ProcessRequest : Activity
 {
    public ProcessRequest()
    {
      // Define the variables used by this workflow
      Variable<ReservationRequest> request =
      new Variable<ReservationRequest> { Name = "request" };
      Variable<ReservationResponse> response =
      new Variable<ReservationResponse> { Name = "response" };
      Variable<bool> reserved = new Variable<bool> { Name = "reserved" };
      Variable<CorrelationHandle> requestHandle =
      new Variable<CorrelationHandle> { Name = "RequestHandle" };
      // Create a Receive activity
      Receive receiveRequest = new Receive
          {
            ServiceContractName = "ILibraryReservation",
            OperationName = "RequestBook",
            CanCreateInstance = true,
            Content = ReceiveContent.Create
```

```
        (new OutArgument<ReservationRequest>(request)),
        CorrelatesWith = requestHandle
    };
  // Define the ProcessRequest workflow
  this.Implementation = () => new Sequence
  {
  DisplayName = "ProcessRequest",
  Variables = { request, response, reserved, requestHandle },
  Activities =
    {
      receiveRequest,
      new WriteLine
        {
         Text = new InArgument<string>(
         env => "Got request from: " +
         request.Get(env).Requester.BranchName),
        },
 new WriteLine
   {
     Text = new InArgument<string>(env => "Requesting: " +
     request.Get(env).Title),
   },
 new Assign
  {
   To = new OutArgument<Boolean>(reserved),
   Value = new InArgument<Boolean>(env => true)
  },
 new Delay
 {
 Duration = TimeSpan.FromSeconds(2)
 },
new CreateResponse
{
Request = new InArgument<ReservationRequest>
(env => request.Get(env)),
Response = new OutArgument<ReservationResponse>
(env => response.Get(env)),
```

```
        Reserved = new InArgument<bool>(env => reserved.Get(env)),
        },
      new WriteLine
        {
        Text = new InArgument<string>
        (env => "Sending response to: " +
        request.Get(env).Requester.BranchName),
        },
       new SendReply
       {
       Request = receiveRequest,
       Content = SendContent.Create
       (new InArgument<ReservationResponse>(response))
       }
     }
   };
  }
 }
}
```

19.3. აპლიკაციის რეალიზაცია

ბოლო ბიჯი პროექტის ასაგებად არის ჰოსტაპლიკაციის რეალიზაცია. უნდა გამოვიყენოთ კონსოლის აპლიკაცია (Program.cs) რომელიც გენერირებულ იქნა შაბლონის (template) სახით. აპლიკაცია ასრულებს მოთხოვნის ინიციალიზებას და დამუშავებას, ამიტომ საჭირო იქნება ორივესთვის ლოგიკის დაწერა. ჯერ უნდა აეწყოს აპლიკაცია მოთხოვნების მისაღებად და დასამუშავებლად, შემდეგ კი უნდა მოხდეს ახალი მოთხოვნის ინიცირება და გაგზავნა სხვა აპლიკაციისათვის.

დავამატოთ Program.cs ფაილში რამდენიმე სახელსივრცე:

```
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
using System.ServiceModel.Description;
using System.Activities;
using System.Xml.Linq;
using System.Configuration;
```

19.3.1. ServiceHost კლასი

შემავალი მოთხოვნების მისაღებად გამოიყენება ServiceHost კლასი, რომელიც WCF ტექნოლოგიაშია. WF 4.0/5 უზრუნველყოფს WorkflowServiceHost კლასს, რომელიც ახდენს ServiceHost –ის რეალიზაციას, ოღონდ აინიცირებს სამუშაო პროცესს, როცა შეტყობინება მიღებულია.

შევიტანოთ 19.13 ლისტინგის კოდი როგორც main() ფუნქციის რეალიზაცია Program კლასისთვის.

```
// --ლისტინგი 19-13. ნაწილობრივი რეალიზაცია  main() ფუნქციის –
//  config ფაილის გახსნა და  ფილიალის სახელის (name) და
// ქსელის მისამართის (adress) მიღება
Configuration config = ConfigurationManager
       .OpenExeConfiguration(ConfigurationUserLevel.None);
AppSettingsSection app = (AppSettingsSection)config.GetSection("appSettings");

string adr = app.Settings["Address"].Value;
Console.WriteLine(app.Settings["Branch Name"].Value);

// სერვისის შექმნა შემოსული მოთხოვნების დასამუშავებლად
WorkflowService service = new WorkflowService
 {
   Name = "LibraryReservation",
   Body = new ProcessRequest(),
   Endpoints =
     {  new Endpoint
        {
            ServiceContractName = "ILibraryReservation",
            AddressUri = new Uri("http://localhost:" + adr +
                "/LibraryReservation"),
            Binding = new BasicHttpBinding(),
        }
   }
};
//  WorkflowServiceHost –ის შექმნა შემოსული მოთხოვნების მისაღებად
System.ServiceModel.Activities.WorkflowServiceHost wsh =
       new System.ServiceModel.Activities.WorkflowServiceHost(service);
wsh.Open();
```

ეს კოდი ჯერ ხსნის აპლიკაციის კონფიგურაციის ფაილს და იღებს მისამართის პარამეტრებს. ის განსაზღვრავს პორტის ნომერს, რომლითაც აპლიკაცია იღებს შემავალ მოთხოვნას. აგრეთვე ფილიალის სახელს, რომელიც ეკრანზე გამოიტანება. ვინაიდან ჩვენ საქმე გვაქვს რამდენიმე აპლიკაციასთან, ეს მექანიზმი საშუალებას მოგვცემს თვალყური ვადევნოთ რომელი რომელია.

19.3.2. სერვისი - WorkflowService კლასი

შემდეგ იგი ქმნის WorkflowService კლასს. Body თვისებისათვის იგი იყენებს ProcessRequest კლასის ახალ ეგზემპლარს, რომელიც განსაზღვრავს პროცესს შემავალი მოთხოვნების დასამუშავებლად.

19.3.3. ბოლო წერტილი

სერვისის კლასი აგრეთვე განსაზღვრავს ბოლო წერტილს, კონტრაქტის IlibraryReservation სერვისის გამოყენებით, URL–ით, რომელიც შეიცავს პორტის ნომრის ცვლადს და BasicHttpBinding კლასს. დასასრულ, WorkflowServiceHost კლასი კონკრეტიზდება განსაზღვრული სერვისკლასის გამოყენებით. შემდეგ ის იხსნება Open () მეთოდის გამოძახებით. ამ მომენტში აპლიკაცია უთვალთვალებს შემავალ შეტყობინებებს. როცა ის მიღებულ იქნება, ProcessRequest სამუშაო პროცესის ეგზემპლარი ამუშავდება მოთხოვნის დასამუშავებლად.

19.3.4. სამუშაო პროცესის გამომძახებელი

ახლა საჭიროა კოდის დამატება მოთხოვნის ინიციალიზებისათვის. შევიტანოთ 19.14 ლისტინგის კოდი ოღონდაც wsh.Open() მეთოდის გამოძახების შემდეგ.

```
// – – – ლისტინგი 19-14.  main() ფუნქციის რეალიზების დარჩენილი ნაწილი – – –
Console.WriteLine("Waiting for requests, press ENTER to send a request.");
Console.ReadLine();
// ლექსიკონის შექმნა შემავალი არგუმენტებით სამუშაო პროცესისთვის
IDictionary<string, object> input = new Dictionary<string, object>
  {
    { "Title" , "Gone with the Wind ქარწაღებული" },
    { "Author", "Margaret Mitchell" },
   { "ISBN", "9781416548898" }
};
//  SendRequest სამუშაო პროცესის გამოძახება
IDictionary<string, object> output =
     WorkflowInvoker.Invoke(new SendRequest(), input);
```

```
ReservationResponse resp = (ReservationResponse)output["Response"];
// პასუხის ეკრანზე გამოტანა
Console.WriteLine("Response received from the {0} branch",
        resp.Provider.BranchName);
Console.WriteLine();
Console.WriteLine("Press ENTER to exit");
Console.ReadLine();
//  WorkflowServiceHost დახურვა
wsh.Close();
```

ეს კოდი იცდის, სანამ მომხმარებელი არ დააჭერს Enter ღილაკს, რომელიც მოგვცემს დროს, რათა მივიღოთ რამდენიმე მუშა კოპიო და თვალყური ვადევნოთ შემოსულ შეტყობინებებს. კოდის დანარჩენი ნაწილი ცნობილია ჩვენთვის, ის ჰგავს 4–7 თავების მასალას. ჯერ იქმნება ლექსიკონი შემავალი არგუმენტისთვის. შემდეგ ის იყენებს WorkflowInvoker კლასის Invoke() მეთოდს, რათა დაწყებულ იქნას SendRequest სამუშაო პროცესის ახალი ეგზემპლარი.

Response პასუხის გამომავალი არგუმენტები ამოიღება ლექსიკონიდან, რომელიც დაბრუნდება უკან, როცა სამუშაო პროცესი დასრულდება. დასასრულ, WorkflowServiceHost დაიხურა სამუშაო პროცესის დასრულებამდე.

ქვემოთ მოცემულია მთლიანი Program.cs პროგრამის ლისტინგი 19.15.

```
// – – – ლისტინგი_19.15 – – – Program.cs  კოდი – – –
     using System;
     using System.Collections.Generic;
     using System.Linq;
     using System.Text;
     using System.ServiceModel;
     using System.ServiceModel.Activities;
     using System.ServiceModel.Activities.Description;
     using System.ServiceModel.Description;
     using System.Activities;
     using System.Xml.Linq;
    using System.Configuration;
//  config ფაილის გახსნა და  ფილიალის სახელის (name) და
// ქსელის მისამართის (adress) მიღება
Configuration config = ConfigurationManager
        .OpenExeConfiguration(ConfigurationUserLevel.None);
AppSettingsSection app = (AppSettingsSection)config.GetSection("appSettings");
```

```
string adr = app.Settings["Address"].Value;
Console.WriteLine(app.Settings["Branch Name"].Value);
// სერვისის შექმნა შემოსული მოთხოვნების დასამუშავებლად
WorkflowService service = new WorkflowService
 {
    Name = "LibraryReservation",
    Body = new ProcessRequest(),
    Endpoints =
      {
        new Endpoint
         {
              ServiceContractName = "ILibraryReservation",
              AddressUri = new Uri("http://localhost:" + adr +
                   "/LibraryReservation"),
              Binding = new BasicHttpBinding(),
         }
    }
};
//  WorkflowServiceHost –ის შექმნა შემოსული მოთხოვნების მისაღებად
System.ServiceModel.Activities.WorkflowServiceHost wsh =
        new System.ServiceModel.Activities.WorkflowServiceHost(service);
wsh.Open();
Console.WriteLine("Waiting for requests, press ENTER to send a request.");
Console.ReadLine();
// ლექსიკონის შექმნა შემავალი არგუმენტებით სამუშაო პროცესისათვის
  IDictionary<string, object> input = new Dictionary<string, object>
   {
      { "Title" , "Gone with the Wind ქარწაღებული" },
      { "Author", "Margaret Mitchell" },
      { "ISBN", "9781416548898" }
  };
   //  SendRequest სამუშაო პროცესის გამოძახება
    IDictionary<string, object> output =
        WorkflowInvoker.Invoke(new SendRequest(), input);
    ReservationResponse resp = (ReservationResponse)output["Response"];
    // პასუხის ეკრანზე გამოტანა
    Console.WriteLine("Response received from the {0} branch",
```

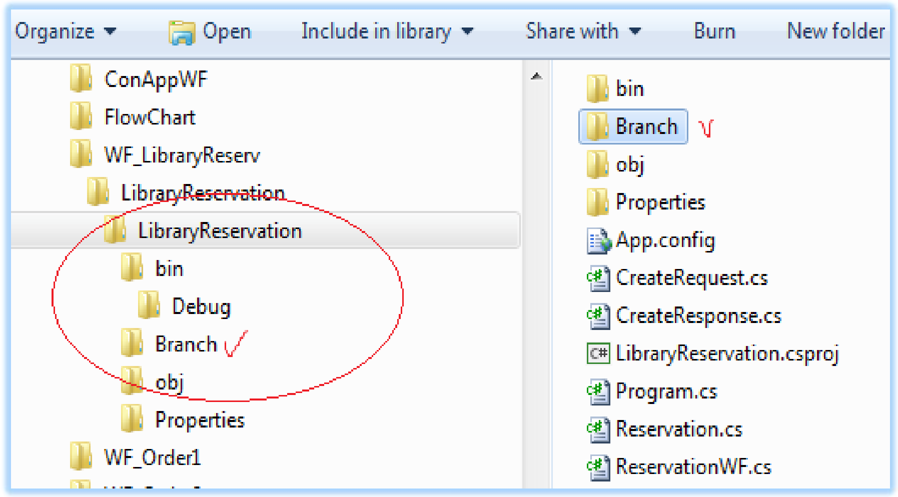
```
      resp.Provider.BranchName);
   Console.WriteLine();
   Console.WriteLine("Press ENTER to exit");
   Console.ReadLine();
   //  WorkflowServiceHost დახურვა
  wsh.Close();
 }
 }
}
```

19.3.5. აპლიკაციის ამუშავება

F5-ით ავამუშავოთ აპლიკაცია და გავმართოთ კოდი. ჩვენ შგვიძლია აპლიკაციის ორი ეგზემპლარის ამუშავება და მათ დასჭირდება განსხვავებული კონფიგურაციის ფაილები. ამიტომაც შეიძლება მათთვის შერჩეულ იქნას სხვადასხვა პორტის ნომერი.

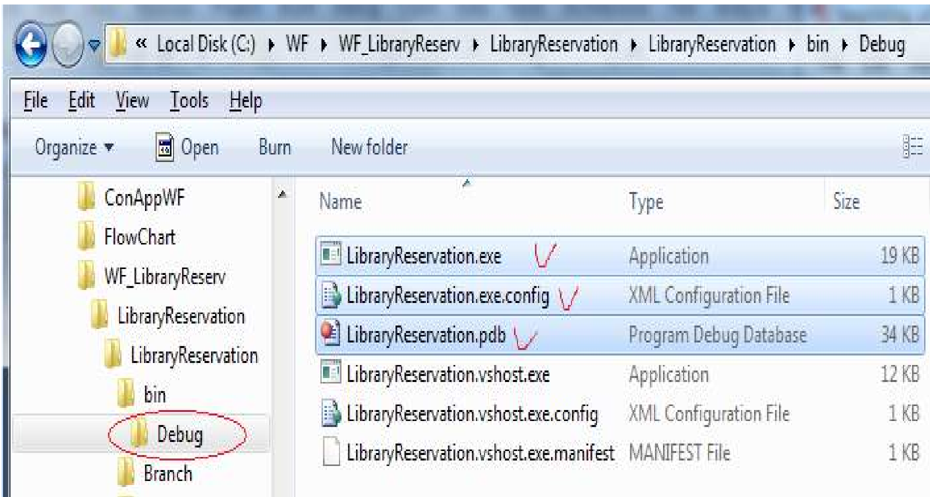
19.3.6. ბიბლიოთეკის ფილიალის კონფიგურირება

გავხსნათ Windows Explorer და LibraryReservation ფოლდერში ჩავამატოთ ახალი ქვეკატალოგი Branch (ფილიალი), როგორც 19.4-ა ნახაზზეა ნაჩვენები.



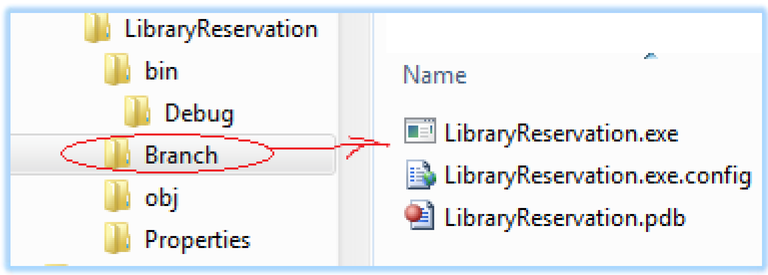
ნახ.19.4-ა

Debag-კატალოგში მოვნიშნოთ სამი ფაილი (ნახ.19.4–ბ) და კოპირებით გადავიტანოთ Branch ფოლდერში.



ნახ.19.4–ბ

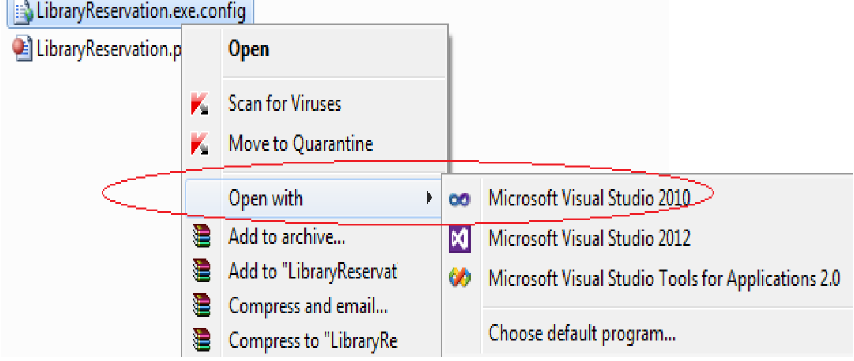
კოპირების შემდეგ Branch–ში მივიღებთ:



ნახ.19.4–გ

გავხსნათ LibraryReservation.exe.config ფაილი რომელიმე ტექსტურ რედაქტორში, მაუსის მარჯვენა ღილაკით Open with… და მაგალითად, Visual Studio – ში (ნახ.19.4–დ).

შევცვალოთ კონფიგურაციის ფაილის ტექსტი შემდეგი კოდის 19.16_ლისტინგით.



ნახ.19.4–დ

```
<!– ლისტინგი_19.16 ——— –>
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <appSettings>
        <add key="Branch Name" value="Southwest Regional"/>
        <add key="ID" value="{CA62F4ED-FACF-4835-8468-16CAAC298F4C}"/>
        <add key="Address" value="8730"/>
        <add key="Request Address" value="8000"/>
   </appSettings>
</configuration>
```

დავაკვირდეთ, რომ პორტის ნომრები მისამართისა და მოთხოვნისთვის წესრიგშია (მთავარის და ფილიალის პორტის ნომრები განსხვავებულია). ავამუშავოთ .exe ფაილი. კონსოლზე მივიღებთ ასეთ ტექსტს:

Southwest Regional
Waiting for requests, press ENTER to send a request.

19.3.7. მოსალოდნელი შედეგები

Visual Studio –დან F5–ით ავამუშავოთ აპლიკაცია. ამ დროს უნდა გაიხსნას მეორე კონსოლის ფანჯარა ასეთი ტექსტით:

Central Library
Waiting for requests, press ENTER to send a request.

შესაძლებელია ფანჯრების გადაადგილება ისე, რომ ორივე ჩანდეს. ავირჩიოთ ერთ-ერთი და დავაჭიროთ Enter-ს. რამდენიმე წამის შემდეგ უნდა გამოჩნდეს ტექსტი:

Southwest Regional
Waiting for requests, press ENTER to send a request.

Request sent; waiting for response
Response received from Central Library
Response received from the Central Library branch

Press ENTER to exit

მეორე ფანჯარაში გამოჩნდება ტექსტი:

Central Library
Waiting for requests, press ENTER to send a request.

Got request from: Southwest Regional
Requesting: Gone with the Wind
Sending response to: Southwest Regional

მეორე ფანჯარაში Enter–ღილაკის დაჭერით მივიღებთ ასეთ ტექსტს:

Central Library
Waiting for requests, press ENTER to send a request.
Got request from: Southwest Regional
Requesting: Gone with the Wind
Sending response to: Southwest Regional

Request sent; waiting for response
Response received from Southwest Regional
Response received from the Southwest Regional branch

Press ENTER to exit

სხვა ფანჯარაში კი გამოჩნდება შემდეგი:

Southwest Regional
Waiting for requests, press ENTER to send a request.

Request sent; waiting for response
Response received from Central Library
Response received from the Central Library branch

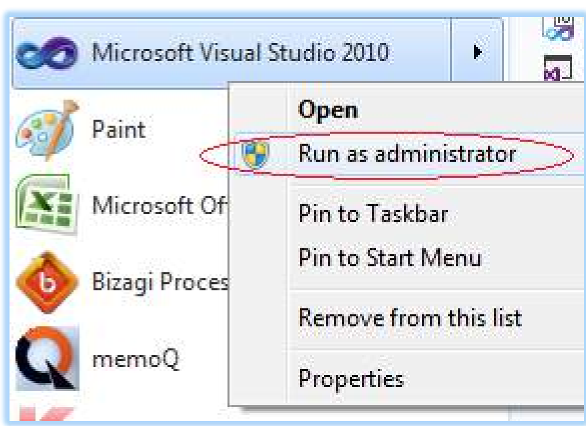
Press ENTER to exit
Got request from: Central Library
Requesting: Gone with the Wind
Sending response to: Central Library

Enter–ღილაკით შეიძლება დაიხუროს აპლიკაციები.

19.3.8. პორტებთან მიმართვის შესაძლებლობა

Windows-ის ოპერაციული სისტემების უსაფრთხოების ამაღლების მიზნით, შესაძლებელია, რომ აპლიკაციამ დააგენერიროს გამონაკლისი სიტუაცია, რის გამოც ვეღარ დავუკავშირდებით მითითებულ პორტს.

თუ ასეთი სიტუაციაა, მაშინ ამ პრობლემის გადაწყვეტის უმარტივესი ხერხია აპლიკაციის ამუშავება „ადმინისტრატორის უფლებებით“. მაგალითად, Visual Studio –ს ამუშავებისას Start-დან ან დესკტოპიდან (ნახ.19.5).



ნახ.19.5. ადმინისტრატორის უფლებით ამუშავება

ფილიალის აპლიკაციის ამუშავებისას აგრეთვე შეიძლება „ადმინისტრატორის უფლებებით“ სარგებლობა.

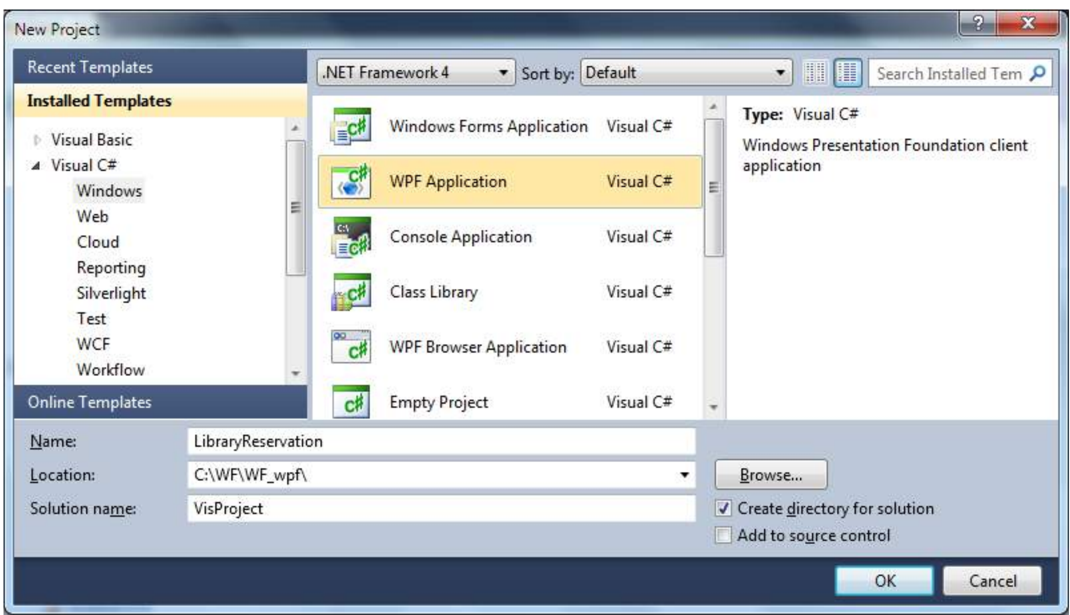
XX თავი
კომუნიკაცია ჰოსტაპლიკაციასთან

ამ თავში ნაჩვენები იქნება წინა თავის პროექტის მსგავსი გადაწყვეტა, ოღონდაც კონსოლის რეჟიმის ნაცვლად გამოყენებულ იქნება Windows Presentation Foundation (WPF) აპლიკაცია.

აქამდე ჩვენ მიერ აგებულ პროექტში ჰოსტი მარტივად იძახებდა სამუშაო პროცესს და დამთავრების შემდეგ ასახავდა შედეგებს. ახალი პროექტისათვის საჭირო იქნება უფრო მეტი კავშირი სამუშაო პროცესსა და ჰოსტაპლიკაციას შორის. საბედნიეროდ, ამ მიზნის მისაღწევად WF 4.5 აქვს კარგი შესაძლებლობები.

20.1. WPF პროექტის შექმნა

1. შევქმნათ ახალი პროექტი WPF აპლიკაციის სახით. პროექტის სახელი იყოს LibraryReservation და Solution-ის VisProject.



ნახ.20.1. WPF აპლიკაციის შექმნა

2. Solution Explorer-ში LibraryReservation პროექტზე მაუსის მარჯვენა ღილაკით ავირჩიოთ Add Reference და .NET ცხრილიდან დავამატოთ შემდეგი კავშირები:

- System.Activities
- System.Configuration
- System.ServiceModel
- System.ServiceModel.Activities

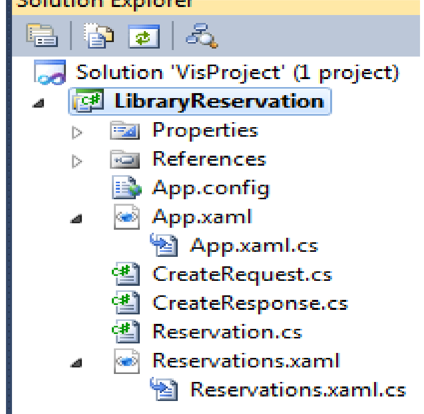
3. Solution Explorer-ში გენერირდება ვინდოუსის ფაილი სახელით Windows1.xaml, რომელიც შევცვალოთ Reservations.xaml - ით.

App.xaml ფაილი განსაზღვრავს ვინდოუსის startup-ს. ესაა ახლა Windows1 ფაილი, რომელიც უნდა შევცვალოთ ასე:

StartupUri="Reservations.xaml"

20.1.1. Class-ების გამოყენება წინა თავიდან

წინა თავის LibraryReservation ფოლდერიდან App.config, CreateRequest.cs, CreateResponse.cs და Reservation.cs ფაილები კოპირებით გადმოვიტანოთ ამ თავის VisProject-ის LibraryReservation ფოლდერში. შემდეგ Solution Explorer-ის LibraryReservation–ზე მაუსის მარჯვენა ღილაკით Add - > Existing Item და პროექტს მივუერთოთ კოპირებული ფაილები. მივიღებთ 20.2 ნახაზზე ნაჩვენებ ფანჯარას.



ნახ.20.2

20.1.2. Window Form –ის განსაზღვრა

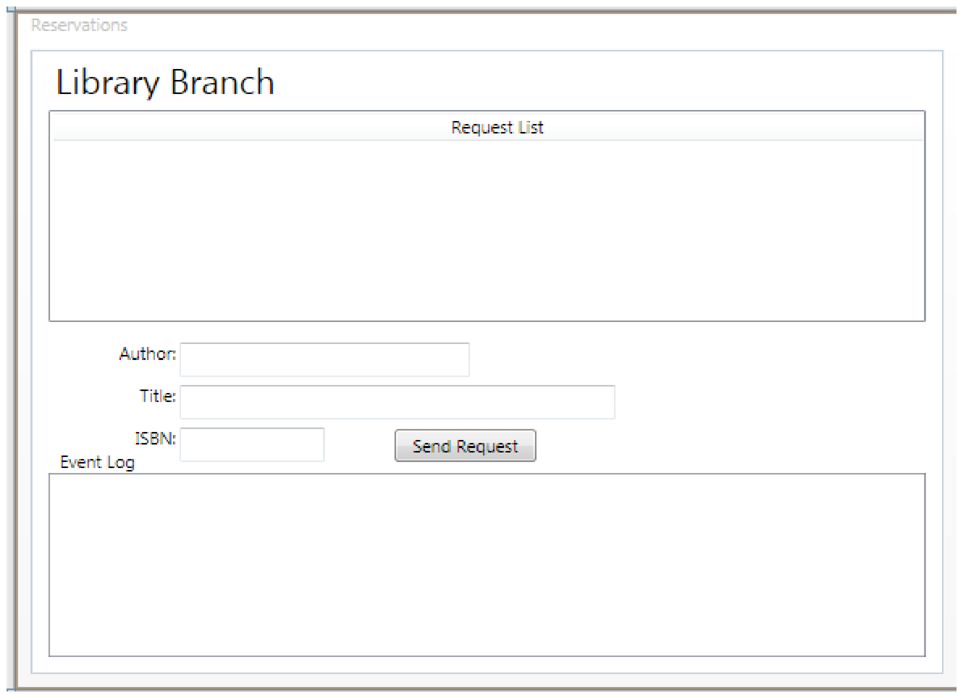
გავხსნათ Reservations.xaml ფაილი და ავირჩიოთ XAML tab. ჩავანაცვლოთ კოდი შემდეგი 20.1 ლისტინგის ტექსტით:

```
<!– – ლისტინგი_20.1 – – – –>
<Window x:Class="LibraryReservation.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Reservations" Height="480" Width="650"
    Loaded="Window_Loaded" Unloaded="Window_Unloaded">
   <Grid>
      <Label Height="40" HorizontalAlignment="Left" Margin="12,0,0,0"
```

```
Name="lblBranch" FontSize="24" VerticalAlignment="Top" Width="276"
    FontStretch="Expanded">Library Branch</Label>
    <ListView x:Name="requestList" Margin="12,42,12,5" Height="150"
     VerticalAlignment="Top" ItemsSource="{Binding}">
      <ListView.View>
      <GridView>
        <GridViewColumn Header="Request List" Width="610">
                <GridViewColumn.CellTemplate>
                    <DataTemplate>
                      <StackPanel Orientation="Horizontal">
                        <TextBlock
                          Text="{Binding Requester.BranchName}"
                                        ing Author}" Width="95"/>
                        <TextBlock Text="{Binding Title}" Width="180"/>
                          <TextBlock Text="{Binding ISBN}" Width="90"/>
                                <Button Content="Reserve"
                                    Tag="{Binding InstanceID}"
                                    Click="Reserve" Width="65"/>
                                <Button Content="Cancel"
                                    Tag="{Binding InstanceID}"
                                    Click="Cancel" Width="60"/>
                          </StackPanel>
                        </DataTemplate>
                      </GridViewColumn.CellTemplate>
                    </GridViewColumn>
                  </GridView>
              </ListView.View>
            </ListView>
              <Label Height="30" Margin="45,25,0,210" Name="label5"
 VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="60"
              HorizontalContentAlignment="Right">Author:</Label>
      <Label Height="30" Margin="45,25,0,180" Name="label2"
 VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="60"
              HorizontalContentAlignment="Right">Title:</Label>
      <Label Height="30" Margin="45,25,0,150" Name="label3"
 VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="60"
              HorizontalContentAlignment="Right">ISBN:</Label>
```

```
        <TextBox Height="25" Margin="102,0,0,210" Name="txtAuthor"
VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="200" />
        <TextBox Height="25" Margin="102,25,0,180" Name="txtTitle"
VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="300" />
        <TextBox Height="25" Margin="102,25,0,150" Name="txtISBN"
VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="100" />
        <Button Height="23" Margin="250,25,12,150" Name="btnRequest"
VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="98"
                Click="btnRequest_Click">Send Request</Button>
    <Label Height="27" HorizontalAlignment="Left" Margin="15,0,0,137"
Name="label4" VerticalAlignment="Bottom" Width="76">Event Log</Label>
    <ListBox Margin="12,0,12,12" Name="lstEvents" Height="130"
VerticalAlignment="Bottom" FontStretch="Condensed" FontSize="10" />
    </Grid>
</Window>
```

შემდეგ ავირჩიოთ Design tab. ფორმას უნდა ჰქონდეს შემდეგი სახე:



ნახ.20.3

ფორმის ზედა ნაწილში Request List ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად სხვა ფილიალში გამოიყენება ფორმის შუაში ველები. აქ მიეთითება ავტორი, დასახელება და ISBN, შემდეგ გაგზავნის ღილაკი.

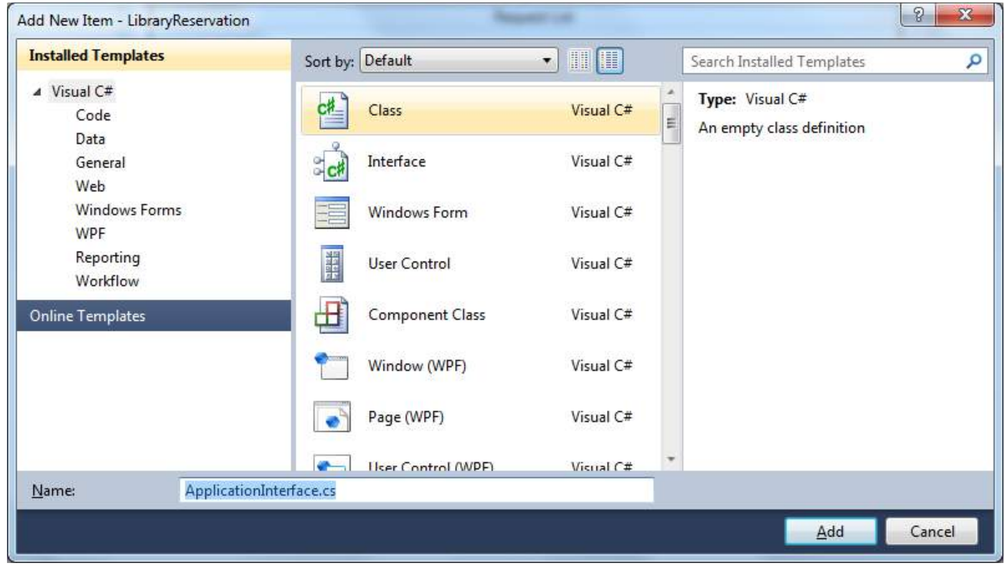
Event Log ქვედა მარცხენა კუთხეში ასახავს სამუშაო პროცესის შეტყობინებას ისე, როგორც კონსოლის რეჟიმშია.

➢ **ტექსტის ჩამწერის რეალიზაცია**

WriteLine ქმედებისთვის, რომელსაც ადრე ვიყენებდით, არ დაგვიყენებია თვისება TextWriter. კონსოლის რეჟიმში ტექსტი ავტომატურად გამოიტანება ეკრანზე. ახლა კი ფორმაზე გამოსატანად უნდა გავეცნოთ TextWriter კლასს.

➢ **აპლიკაციის სტატიკური მიმთითებლის უზრუნველყოფა**

თავიდან უნდა **შეიქმნას სტატიკური კლასი,** რომელიც უზრუნველყოფს აპლიკაციის ფანჯარასთან წვდომას. Solution Explorer-ში LibraryReservation –ზე მაუსის მარჯვენა ღილაკით ვირჩევთ Add -> Class. კლასის სახელია ApplicationInterface.cs, რომლის პროგრამული რეალიზაცია მოცემულია 20.2 ლისტინგში.



ნახ.20.4

```
// – – ლისტინგი_20.2 – – ApplicationInterface.cs  – –
using System;
using System.Windows.Controls;
using System.Activities;
namespace LibraryReservation
```

```
{
  public static class ApplicationInterface
    {
        public static MainWindow _app { get; set; }
        public static void AddEvent(String status)
          {
            if (_app != null)
              {
                new ListBoxTextWriter(_app.GetEventListBox()).WriteLine(status);
              }
          }
    }
}
```

ApplicationInterface კლასს აქვს სტატიკური მიმთითებელი (_app) აპლიკაციის ფანჯარაზე (MainWindow კლასი). სტატიკური AddEvent() მეთოდი ქმნის ListBoxTextWriter კლასის ეგზემპლარს, რომელიც შემდგომში იქნება რეალიზებული და იძახებს მის WriteLine() მეთოდს.

გავხსნათ Reservations.xaml.cs ფაილი და დავამატოთ შემდეგი სახელსივრცეები:

```
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
using System.ServiceModel.Description;
using System.ServiceModel.Channels;
using System.Activities;
using System.Xml.Linq;
using System.Configuration;
```

დავამატოთ კონსტრუქტორის შემდეგი კოდი:

```
ApplicationInterface._app = this;
```

this აინიციალიზებს _app მიმართვას ApplicationInterface კლასში. ვინაიდან იგი სტატიკური კლასია, მასში იქნება მხოლოდ ერთი ეგზემპლარი, რომელსაც ექნება მიმართვა MainWindow კლასზე. დავამატოთ შემდეგი მეთოდები Reservations.xaml.cs ფაილში.

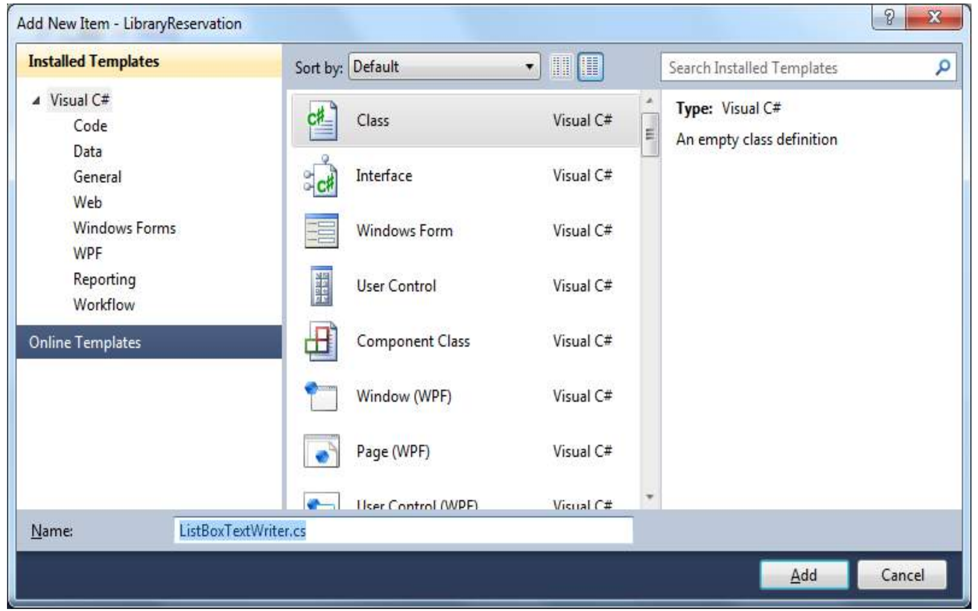
```
public ListBox GetEventListBox()
  {
    return this.lstEvents;
```

```
    }
  private void AddEvent(string szText)
    {
      lstEvents.Items.Add(szText);
    }
```

GetEventListBox() მეთოდი აბრუნებს უკან მიმთითებელს ListBox–ის ფაქტიური კონტროლისათვის, რომელმაც უნდა ასახოს ეს მოვლენები. ამ მეთოდს იყენებს ApplicationInterface კლასი. AddEvent () მეთოდს იყენებს აპლიკაცია მაშინ, როცა მას სჭირდება მოვლენის დამატება.

- **ListBoxTextWriter–ის რეალიზაცია**

დავამატოთ პროექტს კლასი ListBoxTextWriter.cs (ნახ.20.5), რომლის რეალიზაცია 2.3 ლისტინგშია მოცემული.



ნახ.20.5

```
// ---- ლისტინგი_20.3 ------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
```

```
using System.Windows.Controls;

namespace LibraryReservation
 {
   public class ListBoxTextWriter : TextWriter
     {
        const string textClosed = "This TextWriter must be opened before use";
       private Encoding _encoding;
       private bool _isOpen = false;
       private ListBox _listBox;
       public ListBoxTextWriter()
         {
             // Get the static list box
             _listBox = ApplicationInterface._app.GetEventListBox();
             if (_listBox != null)
             _isOpen = true;
         }
      public ListBoxTextWriter(ListBox listBox)
        {
               this._listBox = listBox;
               this._isOpen = true;
        }
      public override Encoding Encoding
         {
           get
            {
               if (_encoding == null)
                  {
                     _encoding = new UnicodeEncoding(false, false);
                   }
               return _encoding;
             }
         }
      public override void Close()
         {
            this.Dispose(true);
         }
```

```
protected override void Dispose(bool disposing)
  {
   this._isOpen = false;
   base.Dispose(disposing);
  }
public override void Write(char value)
  {
      if (!this._isOpen)
           throw new ApplicationException(textClosed) ;  ;

      this._listBox.Dispatcher.BeginInvoke
             (new Action(() => this._listBox.Items.Add(value.ToString())));
  }
public override void Write(string value)
   {
      if (!this._isOpen)
             throw new ApplicationException(textClosed) ;  ;

      if (value != null)
             this._listBox.Dispatcher.BeginInvoke
                  (new Action(() => this._listBox.Items.Add(value)));
   }
public override void Write(char[] buffer, int index, int count)
  {
      String toAdd = "";
      if (!this._isOpen)
               throw new ApplicationException(textClosed) ;  ;

      if (buffer == null || index < 0 || count < 0)
           throw new ArgumentOutOfRangeException("buffer");

      if ((buffer.Length - index) < count)
           throw new ArgumentException("The buffer is too small");

      for (int i = 0; i < count; i++)
           toAdd += buffer[i];
```

```
            this._listBox.Dispatcher.BeginInvoke
                (new Action(() => this._listBox.Items.Add(toAdd)));
        }
      }
  }
```

ListBoxTextWriter კლასი არის მიღებული აბსტრაქტული TextWriter კლასიდან და უზრუნველყოფს Write() მეთოდის განხორციელებას, რომელიც ამატებს სტრიქონს ListBox-ში (თუ თქვენ გსურთ განახორციელოთ Write() მეთოდის სამი გადატვირთვა, იმისთვის რომ იყოს მიღებული, როგორც char ან string ან char [] მასივი.)

არსებული კონსტრუქტორი იყენებს სტატიკურ ApplicationInterface კლასს MainWindow-ის lstEvents კონტროლის მისაღებად. იგი ასევე უზრუნველყოფს კონსტრუქტორს, რომელშიც ListBox შეიძლება შესრულდეს. ეს კონსტრუქტორი გამოიყენება ApplicationInterface კლასის AddEvent () მეთოდით.

ListBox-ის Add() მეთოდი ხორციელდება აპლიკაციის შესრულების ნაკადის (thread) შემდეგაც. იგი აკეთებს ამას Dispatcher-ის BeginInvoke() მეთოდის გამოყენებით, რომელიც ასოცირდება lstEvents მართვის ელემენტთან. ეს საშუალებას აძლევს მეთოდს სხვადასხვა ნაკადებიდან გამოძახების დროსაც იმუშაოს.

იმის გამო, რომ ListBoxTextWriter კლასი არის მიღებული TextWriter-დან, შეიძლება მისი მითითება როგორც TextWriter თვისებისა ნებისმიერი WriteLine ქმედებისათვის. და სტატიკური ApplicationInterface კლასის გამო, ListBoxTextWriter კლასს შეუძლია წვდომა lstEvents ელემენტზე აპლიკაციის გარედანაც კი.

ასე რომ, არსებობს სამი გზა lstEvents მართვის ელემენტში ტექსტის დასამატებლად:

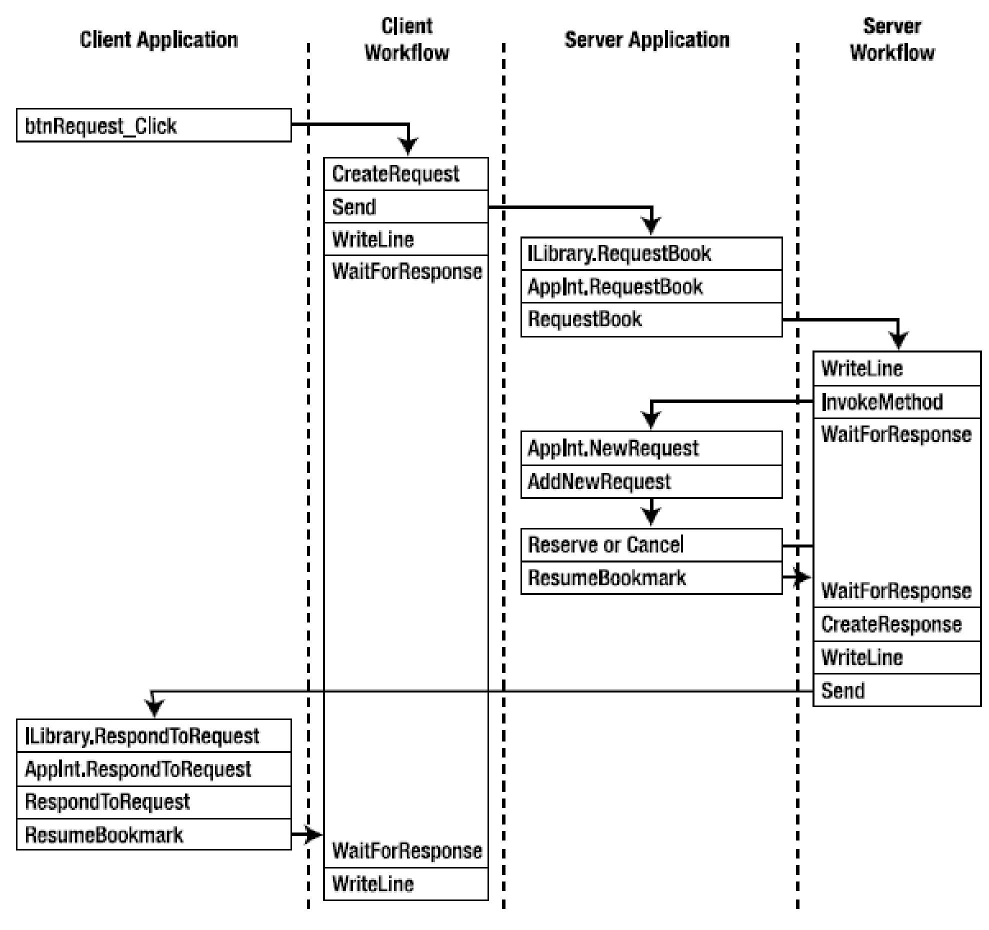
– აპლიკაციის შიგნიდან, გამოიყენება ლოკალური AddEvent () მეთოდი;

– აპლიკაციის გარედან, გამოიყენება ApplicationInterface კლასის AddEvent () მეთოდი;

– WriteLine ქმედებიდან, მიეთითება TextWriter თვისება ListBoxTextWriter-ზე.

- **Workflows პროცესების რეალიზაცია**

20.6 ნახაზზე ნაჩვენებია ზოგადი ლოგიკა და შეტყობინებათა ნაკადი. ამ სქემის ელემენტები მოგვიანებით იქნება ახსნილი, ამჯერად კი განხილულ იქნება ძირითადი კონცეფციები.



ნახ.20.6

ეს მცირეთი განსხვავდება პროცესებისაგან, რომლებიც წინა თავში განვიხილეთ. შესამჩნევი განსხვავება მდგომარეობს იმაში, რომ არ არსებობს არავითარი Receive ქმედება. მის მაგივრად აპლიკაცია მიიღებს შემავალ შეტყობინებებს, შემდეგ კი გამოიძახებს (ან აღადგენს) სამუშაო პროცესს.

- **შეტყობინებათა მიღება (Listening for Messages)**

20.6 ნახაზზე სერვერული აპლიკაცია იღებს შეტყობინებას და მასთან დაკავშირებული ელემენტი დიაგრამეზე არის ლებელი ILibrary.RequestBook. გარდა ამისა, კლიენტის აპლიკაცია იღებს შეტყობინებას და ელემენტი აღნიშნულია ILibrary.RespondToRequest-ით. ესაა მეთოდები სერვისული კონტრაქტის, რომლებიც რეალიზებული იყო მე-8 თავში.

გავხსნათ Reservation.cs ფაილი, რომელშიც გამოჩნდება ინტერფეისის შემდეგი განსაზღვრება:

```
    [ServiceContract]
   public interface ILibraryReservation
   {
      [OperationContract]
      void RequestBook(ReservationRequest request);

      [OperationContract]
      void RespondToRequest(ReservationResponse response);
   }
```

საჭიროა მცირე ცვლილების ჩატარება. კერძოდ, OperationContract -ს უნდა დაემატოს (IsOneWay = true). ქვემოთ ნაჩვენებია ეს:

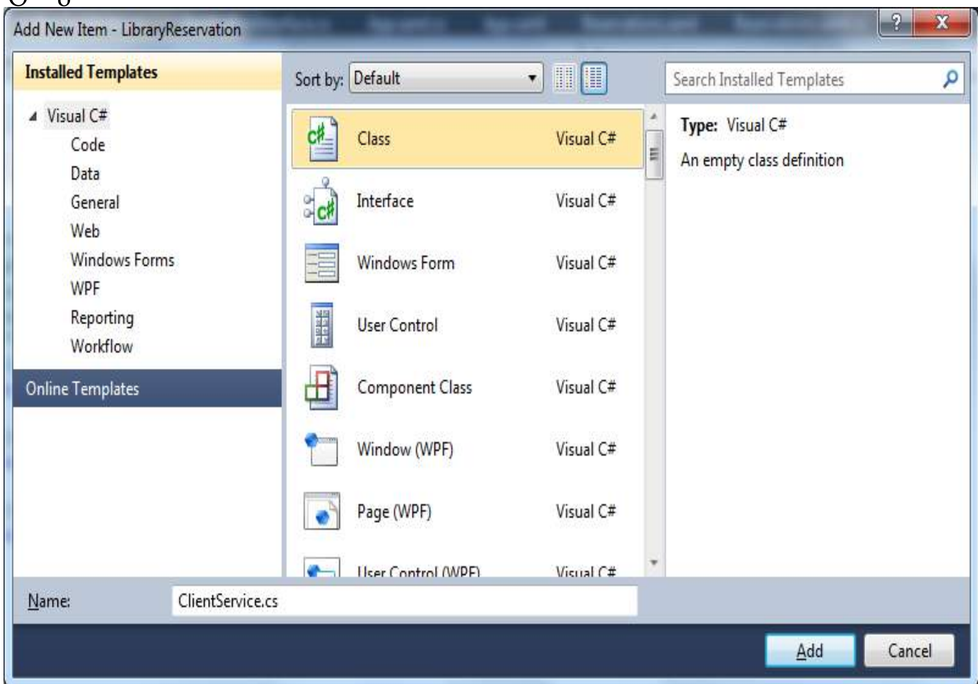
```
    [ServiceContract]
   public interface ILibraryReservation
   {
      [OperationContract(IsOneWay = true)]
      void RequestBook(ReservationRequest request);

      [OperationContract(IsOneWay = true)]
      void RespondToRequest(ReservationResponse response);
   }
```

შეტყობინება იგზავნება სამუშაო პროცესთან ერთად, მაგრამ პასუხი მიიღება ServiceHost-ით აპლიკაციის შიგნით. ასე, რომ ეს არაა ტქნიკური ორმხრივი ლაპარაკი. არსებობს შეტყობინებები ორივე მიმართულებით. რადგან გაგზავნის და მიღების საბოლოო წერტილები სხვადასხვაა, WCF ამას აფიქსირებს ცალკე ერთმიმართულებიანი შეტყობინებები.

- **სერვისის კონტრაქტის რეალიზაცია**

სერვისის კონტრაქტი განსაზღვრავს მხოლოდ ხელმისაწვდომ მეთოდებს, იგი არ უზრუნველყოფს მათ იმპლემენტაციას (რეალიზაციას). წინა თავში სამუშაო პროცესი უზრუნველყოფდა რეალიზაციას.

ჩვენი პროექტისთვის აუცილებელია ამ საკითხის გადაწყვეტა. ამიტომაც, Solution Explorer -ში LibraryReservation-ზე მარჯვენა ღილაკით ვირჩევთ Add Class. მივუთითებთ კლასის სახელს ClientService.cs (ნახ.20.7). მისი კოდის რეალიზაცია ნაჩვენებია 20.4 ლისტინგში.



ნახ.20.7

```
// --------- ლისტინგი 20.4 --------------------
using System;
using System.ServiceModel;
namespace LibraryReservation
{
  public class ClientService : ILibraryReservation
    {
   public void RequestBook(ReservationRequest request)
        {
          ApplicationInterface.RequestBook(request);
        }
   public void RespondToRequest(ReservationResponse response)
  {
```

```
      ApplicationInterface.RespondToRequest(response);
    }
  }
}
```

ეს რეალიზაცია იყენებს ApplicationInterface სტატიკურ კლასს, რომელიც უკვე შექმნილია ჩვენ მიერ. ყოველი მეთოდი უბრალოდ იძახებს ApplicationInterface კლასის შესაბამის მეთოდს. გავხსნათ ApplicationInterface.cs ფაილი და დავამატოთ შემდეგი მეთოდები:

```
public static void RequestBook(ReservationRequest request)
  {
      if (_app != null)
          _app.RequestBook(request);
  }
public static void RespondToRequest(ReservationResponse response)
  {
      if (_app != null)
          _app.RespondToRequest(response);
  }
```

ეს მეთოდები თავის მხრივ იძახებს შესაბამის მეთოდებს აპლიკაციაში სტატიკური მიმთითებლის გამოყენებით. საჭირო ინება ამ მეთოდების რეალიზება Reservation.xaml.cs ფაილში, რასაც მოგვიანებით დავუბრუნდებით.

- **ServiceHost -ის რეალიზაცია**

აპლიკაციისთვის აუცილებელია ServiceHost-ის რეალიზაცია შემავალი შეტყობინებების მისაღებად (მოსასმენად). გავხსნათ Reservation.xaml.cs ფაილი და დავამატოთ შემდეგი კლასის წევრები:

private ServiceHost _sh;

იგი კონსტრუქტორის წინ ასე უნდა მოთავსდეს:

```
public partial class MainWindow : Window
  {
      private ServiceHost _sh;
```

```
        public MainWindow()
        {
            InitializeComponent();
            ApplicationInterface._app = this;
        }
```

იწყება ServiceHost მაშინ, როცა ფანჯარა ჩატვირთულია და იხურება, როცა ფანჯარა ამოტვირთულია. მეთოდების დამატება ნაჩვენებია 20.5 ლისტინგში MainWindow კლასისთვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

//------- ლისტინგი 20.5 --------------- *Event Handlers* -ის ჩატვირთვა/ამოტვირთვა -----

```
private void Window_Loaded(object sender, RoutedEventArgs e)
 { // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და
   // მისი ქსელური მისამართი
 Configuration config = ConfigurationManager.OpenExe
             Configuration(ConfigurationUserLevel.None);
 AppSettingsSection app = (AppSettingsSection)config.
                                GetSection("appSettings");
 string adr = app.Settings["Address"].Value;
   // ფილიალის სახელის გამოტანა ფორმაზე
 lblBranch.Content = app.Settings["Branch Name"].Value;

        // ServiceHost-ის შექმნა
        _sh = new ServiceHost(typeof(ClientService));

        // დასასრულის წერტილის (Endpoint) დამატება
        string szAddress = "http://localhost:" + adr + "/ClientService";
        System.ServiceModel.Channels.Binding bBinding = new
BasicHttpBinding();
       _sh.AddServiceEndpoint(typeof(ILibraryReservation),bBinding,
szAddress);

  // ServiceHost-ის გახსნა შეტყობინებების მისაღებად (listen)
         _sh.Open();

        // ListBoxTextWriter -ის ტესტირება
        //ListBoxTextWriter lbtw = new ListBoxTextWriter();
```

```
        //lbtw.Write("ეს არის ტესტი - This is a test");
    }

        private void Window_Unloaded(object sender, RoutedEventArgs e)
        {
            // service host-ის დატოვება
            _sh.Close();
        }
```

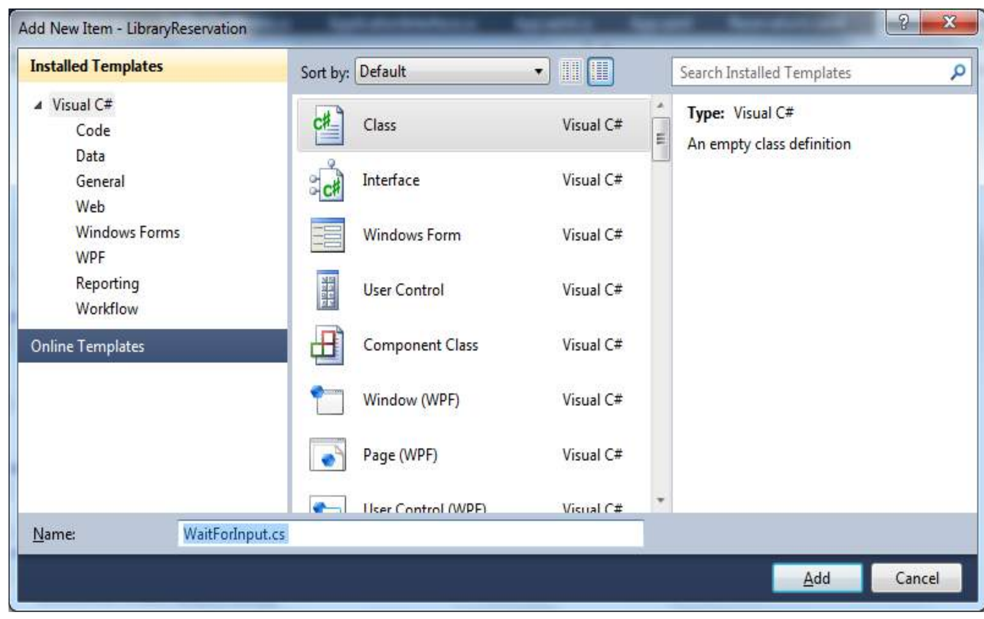
მოვლენის დამმუშავებელი Loaded ხსნის კონფიგურაციის ფაილს და ათავსებს ფილიალის სახელს lblBranch -მართვის ელემენტში, ამიტომაც ფორმა ასახავს ლოკალური ფილიალის სახელს. შემდეგ იქმნება ServiceHost თანამგზავრი (passing) ClientService კლასისა, რომელიც ახლახანს შევქმენით როგორც მისი რეალიზაცია. შემდეგ იგი აკონფიგურირებს დასასრულის წერტილს ServiceHost-თვის, იყენებს რა ცნობილი მისამართის, მიბმისა და კონტრაქტის სამეულს.

Unloaded მოვლენის დამმუშავებელი უბრალოდ ხურავს ServiceHost-ს, ასე რომ მეტი აღარ მოხდება შეტყობინებების მიღება.

20.1.3. სანიშნეები

სანიშნეები (Bookmarks) იძლევა საშუალებას, შეაჩეროს სამუშაო პროცესი და შეინახოს მარკერი ისე, რომ შემდგომ შეიძლებოდეს მისი იმავე წერტილიდან განახლება. ისინი დამუშავებულია მონაცემთა მისაღებად შემდგომი აღდგენის მიზნით. ამ პროექტში, მაგალითად, როცა მოთხოვნა მიღებულია, აპლიკაციას გამოაქვს ეკრანზე ეს მოთხოვნა და ელოდება მომხმარებლისგან პასუხის (კი/არა) შეტანას. შემდეგ სამუშაო პროცესი გრძელდება ამ პასუხის გავლით.

სანიშნეები იქმნება მომხმარებელთა აქტიურობით. აქ ჩვენ შევქმნით ზოგად ქმედებას, რომელიც შემდგომში გამოყენებადი იქნება ყველგან, სადაც სანიშნე მოითხოვს. LibraryReservation-ზე Solution Explorer-ში, მარჯვენა ღილაკით დავამატოთ კლასი WaitForInput.cs, რომლის 20.6 ლისტინგი მოცემულია ქვემოთ.



ნახ.20.8. WaitForInput.cs კლასის შექმნა

// ------------- ლისტინგი 20.6 ---------------

```
using System;
using System.Activities;

namespace LibraryReservation
{
 public sealed class WaitForInput<T> : NativeActivity<T>
    {
        public WaitForInput() : base()
        {
        }

        public string BookmarkName { get; set; }
        public OutArgument<T> Input { get; set; }

   protected override void Execute(NativeActivityContext
             context)
     {
       context.CreateBookmark(BookmarkName,
             new BookmarkCallback(this.Continue));
```

```
      }

   void Continue(NativeActivityContext context, Bookmark
                  bookmark, object obj)
        {
          Input.Set(context, (T)obj);
        }
    protected override bool CanInduceIdle { get { return
           true; } }
    }
}
```

განმარტება: sealed class-ის შესახებ(http://www.techopedia.com/definition/25637/sealed-class-c):

ეს არის **დაცული** კლასი C#-ში, რომელიც არ შეიძლება მიღებულ იქნეს მემკვიდრეობით ყველა კლასის მიერ, მაგრამ მისი შექმნა შესაძლებელია.

დიზაინერული ჩანაფიქრი დაცული კლასის ისაა, რომ იქნას მითითებული, რომ ის სპეციალიზებულია და არაა აუცილებელი მისი გაფართოება, მემკვიდრეობით მისთვის დამატებითი ფუნქციონალობის გადაცემა, მისი ყოფაქცევის გადასატვირთად. დაცული კლასი ხშირად გამოიყენება ლოგიკის ინკაფსულაციისათვის, რომელიც პროგრამამ უნდა გამოიყენოს ყოველგვარი ცვლილებების გარეშე.

დაცული კლასი გამოიყენება ძირითადად უსაფრთხოების მიზნით. დაცულ კლასს არ შეუძლია საბაზო კლასის ფორმირება. დაცული კლასების გამოძახება უფრო სწრაფად ხდება, რადგან ისინი უზრუნველყოფს შესრულების გარკვეულ ოპტიმიზაციას, მაგალითად, ვირტუალური ფუნქცია-წევრების გამოძახება დაცული კლასის შემთხვევაში არავირტუალური გამოძახებისას.

.NET Framework ბიბლიოთეკის ზოგიერთი საკვანძო კლასი შესრულებულია დაცული კლასის სახით, ძირითადად მათი გაფართოების შეზღუდვის მიზნით.

= = = =

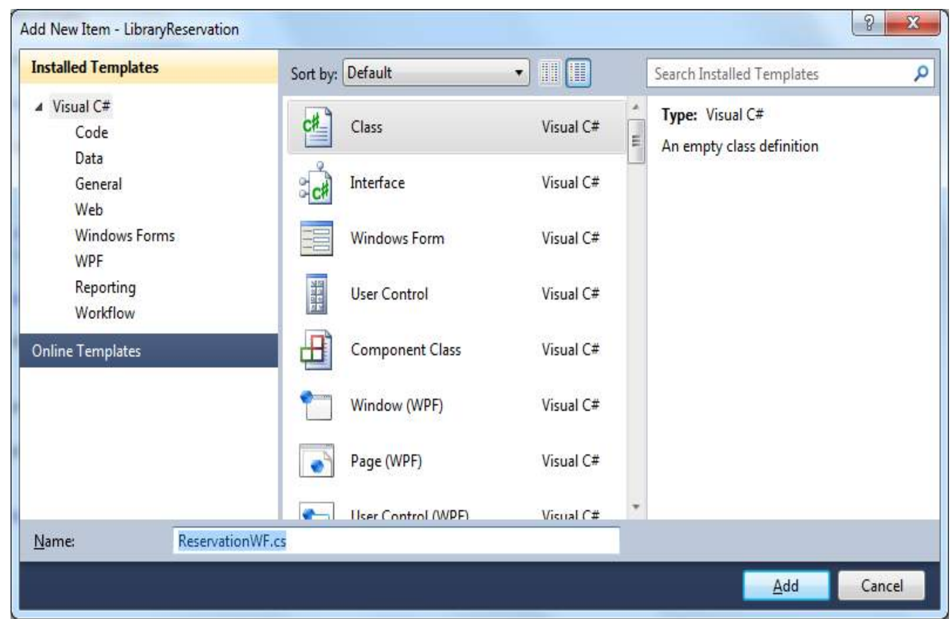
მომხმარებლის ეს ქმედება გამოიყენებს NativeActivity საბაზო კლასს (CodeActivity-ის ნაცვლად), იმიტომ რომ იგი აძლევს მას NativeActivityContext -თან მიმართვის საშუალებას, რომელიც აუცილებელია სანიშნის შესაქმნელად. იგი ასევე იყენებს შაბლონის ვერსიას (კლასში <T> აღნიშვნაა). შემავალი არგუმენტი ასახავს მონაცემებს, რომლებიც გადაეცემა სამუშაო პროცესს, როცა ის განახლდება. შაბლონური ვერსიის დახმარებით ეს ქმედება შესაძლებელია გამოყენებულ იქნას განმეორებით მონაცემთა ნებისმიერ ტიპთან.

Execute() მეთოდი იძახებს NativeActivityContext-ის CreateBookmark() მეთოდს, მიუთითებს რა სანიშნის სახელს და მიმთითებელს უკუ-გამოძახების მეთოდზე, სახელით Continue(). როდესაც სამუშაო პროცესი აღდგენილია, მაშინ ეს უკუგამოძახების მეთოდი სრულდება. ყურადსაღებია, რომ უკუგამოძახების მეთოდი ობიექტი იღებს მესამე პარამეტრის სახით. ესაა მონაცემები, წარმოდგენილი აპლიკაციით. იგი ინახება შემავალ არგუმენტში, რაც ხელმისაწვდომია სამუშაო პროცესისთვის.

აქტიურობები (ქმედებები), რომლებიც იყენებს სანიშნეებს, უნდა იქნას გადატვირთული (override), რომ CanInduceIdle თვისება აბრუნებდეს true მნიშვნელობას. ეს კი უზრუნველყოფს სამუშაო პროცესს, რომ გადავიდეს ლოდინის მდგომარეობაში მანამ, სანამ სანიშნით მოხდება მისი აღდგენა.

20.2. SendRequest სამუშაო პროცესის რეალიზაცია

ახლა შევასრულოთ სამუშაო პროცესების რეალიზაცია. Solution Explorer-ის LibraryReservation-ზე მარჯვენა ღილაკით ავირჩიოთ Add -> Class. სახელი ReservationWF.cs (ნახ.20.9).



ნახ.20.9

კოდის რეალიზაცია მოცემულია 20.7 ლისტინგში.

// ------------ ლისტინგი 20.7 --------------------

```
using System;
using System.Activities;
```

```
using System.Activities.Statements;
using System.ServiceModel.Activities;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Runtime.Serialization;
using System.Xml.Linq;
using System.IO;

namespace LibraryReservation
{
 public sealed class SendRequest : Activity
  {
    // Define the input and output arguments
    public InArgument<string> Title { get; set; }
    public InArgument<string> Author { get; set; }
    public InArgument<string> ISBN { get; set; }
    public InArgument<TextWriter> Writer { get; set;  }
    public OutArgument<ReservationResponse> Response
                {get; set; }
   public SendRequest()
      {
       // Define the variables used by this workflow
       Variable<ReservationRequest> request =
  new Variable<ReservationRequest> { Name = "request" };
       Variable<string> requestAddress =
  new Variable<string> { Name = "RequestAddress" };
       Variable<bool> reserved = new Variable<bool> { Name
             = "Reserved" };
       // Define the SendRequest workflow
      this.Implementation = () => new Sequence
     {
       DisplayName = "SendRequest",
       Variables = { request, requestAddress, reserved },
         Activities =
          {
           new CreateRequest
              {
```

```
            Title = new InArgument<string>(env =>
                    Title.Get(env)),
            Author = new InArgument<string>(env =>
                    Author.Get(env)),
            ISBN = new InArgument<string>(env =>
                    ISBN.Get(env)),
       Request = new OutArgument<ReservationRequest>
            (env => request.Get(env)),
       RequestAddress = new OutArgument<string>
            (env => requestAddress.Get(env))
          },
       new Send
         {
           OperationName = "RequestBook",
        ServiceContractName = "ILibraryReservation",
          Content = SendContent.Create
       (new InArgument<ReservationRequest>(request)),
         EndpointAddress = new InArgument<Uri>
       (env => new Uri("http://localhost:" +
       requestAddress.Get(env) + "/ClientService")),
        Endpoint = new Endpoint
          {
             Binding = new BasicHttpBinding()
          },
        },
      new WriteLine
     {
      Text = new InArgument<string>
        (env => "Request sent; waiting for response"),
      TextWriter = new InArgument<TextWriter>
        (env => Writer.Get(env))
      },
  new WaitForInput<ReservationResponse>
    {
     BookmarkName = "GetResponse",
     Input = new OutArgument<ReservationResponse>
       (env => Response.Get(env))
```

```
        },
      new WriteLine
        {
         Text = new InArgument<string>
          (env => "Response received from " +
         Response.Get(env).Provider.BranchName + " [" +
         Response.Get(env).Reserved.ToString() + "]"),
         TextWriter = new InArgument<TextWriter>
           (env => Writer.Get(env))
        },
       }
      };
     }
    }
    // ProcessRequest ბიზნესპროცესის (workflow) ჩასამატებელი ადგილი
}
```

ამ სამუშაო პროცესის უმეტესი ნაწილი იდენტურია წინა თავში განხილული რეალიზაციის, ამიტომაც აქ იგი დეტალურად აღარ აიხსნება. მოცემულ იქნება მხოლოდ ის, რაშიც განსხვავებაა. უნდა აღვნიშნოთ, რომ ყოველ WriteLine ქმედებას აქვს დამატებითი თვისება:

TextWriter = new ListBoxTextWriter()

ის მიუთითებს იმაზე, რომ ახალი კლასი ListBoxTextWriter, რომელიც იქნა რეალიზებული, უნდა იქნას გამოყენებული ამ ტექსტის დისპლეიზე გამოსატანად. ეს გამოიწვევს ტექსტის ასახვას lstEvents მართვის ელემენტში.

სხვა განსხვავება ისაა, რომ მომხმარებლის ქმედება WaitForInput გამოიყენება Receive ქმედების ნაცვლად. აპლიკაცია მიიღებს საპასუხო შეტყობინებას უშუალოდ (პირდაპირ). როცა მიღებულ იქნება პასუხი, მაშინ აპლიკაცია აღადგენს სამუშაო პროცესს, რომელიც მიმდინარეობს ReservationResponse კლასში.

ყურადსაღებია, რომ მომხმარებლის ქმედება განისაზღვრება როგორც WaitForInput <ReservationResponse>, მიუთითებს რა, რომ გადასაცემი მონაცემები იქნება ReservationResponse კლასის.

➢ **ProcessRequest სამუშაო პროცესის რეალიზაცია (Implementing the ProcessRequest Workflow)**

ProcessRequest მუშა პროცესი განსაზღვრება მოცემულია 20.8 ლისტინგში. ჩავამატოთ ეს კოდი ReservationWF.cs ფაილში.

// ------------- ლისტინგი 20.8 -------------------

```
public sealed class ProcessRequest : Activity
 {
  public InArgument<ReservationRequest> request{get;set;}
  public InArgument<TextWriter> Writer { get; set; }
  public ProcessRequest()
    {
     // Define the variables used by this workflow
  Variable<ReservationResponse> response =
        new Variable<ReservationResponse> { Name =
         "response" };
   Variable<bool> reserved = new Variable<bool> { Name =
         "Reserved" };
 Variable<string> address = new Variable<string> { Name=
         "Address" };
   // Define the ProcessRequest workflow
 this.Implementation = () => new Sequence
    {
      DisplayName = "ProcessRequest",
     Variables = { response, reserved, address },
     Activities =
       {
        new WriteLine
         {
          Text = new InArgument<string>(env => "Got
                request from: " +
            request.Get(env).Requester.BranchName),
          TextWriter = new InArgument<TextWriter> (env =>
            Writer.Get(env))
         },
       new InvokeMethod
         {
```

```
      TargetType = typeof(ApplicationInterface),
      MethodName = "NewRequest",
      Parameters =
        {
         new InArgument<ReservationRequest>(env =>
             request.Get(env))
        }
     },
 new WaitForInput<bool>
   {
    BookmarkName = "GetResponse",
    Input = new OutArgument<bool>(env =>
        reserved.Get(env))
   },
 new CreateResponse
   {
    Request = new InArgument<ReservationRequest>
        (env => request.Get(env)),
    Reserved = new InArgument<bool>(env =>
         reserved.Get(env)),
    Response = new OutArgument<ReservationResponse>
        (env => response.Get(env))
   },
 new WriteLine
   {
    Text = new InArgument<string>(env => "Sending
         response to: " +
         request.Get(env).Requester.BranchName),
    TextWriter = new InArgument<TextWriter> (env =>
         Writer.Get(env))
   },
 new Send
   {
    OperationName = "RespondToRequest",
         ServiceContractName = "ILibraryReservation",
    EndpointAddress = new InArgument<Uri>(
      env => new Uri("http://localhost:" +
```

```
            request.Get(env).Requester.Address +
                "/ClientService")),
        Endpoint = new Endpoint
            { Binding = new BasicHttpBinding()
            },
          Content = SendContent.Create
         (new InArgument<ReservationResponse>(response))
        }
       }
      };
     }
    }
```

ეს სამუშაო პროცესი განსხვავდება წინა თავში რეალიზებული ვერსიისაგან. იმის მაგივრად, რომ დაწყება იყოს Receive ქმედებით, რათა მიღებულ იქნას შემავალი მოთხოვნა, ReservationRequest გადასცემს სამუშაო პროცესს შემავალი არგუმენტის გამოყენებით. WriteLine ქმედება, რომელიც მოსდევს მას, ცნობს შემავალ მოთხოვნას.

InvokeMethod ქმედება ჩვენ ადრე გამოვიყენეთ, რათა გამოგვეძახა შეკვეთის ფასდაკლების საანგარიშო მეთოდი. ახლა გამოვიყენოთ იგი მონაცემთა გადასაცემად აპლიკაციაში. ApplicationInterface კლასი მოხერხებულადაა შესრულებული ამ მიზნით. იგი უზრუნველყოფს მუშა პროცესს, რათა განხორციელდეს გამოძახება აპლიკაციაში. InvokeMethod ქმედება იძახებს ApplicationInterface კლასის NewRequest () მეთოდს ReservationRequest კლასში გადასაცემად.

გავხსნათ ApplicationInterface.cs ფაილი და დავამატოთ მეთოდი, რომელიც უბრალოდ იძახებს AddNewRequest ()-ს აპლიკაციაში:

```
public static void NewRequest(ReservationRequest request)
    {  if (_app != null)
          _app.AddNewRequest(request);
    }
```

შემდეგი აქტიურობაა მომხმარებლის WaitForInput ქმედება, რომელიც გამოიყენებოდა SendRequest სამუშაო პროცესში.

ამჯერად იგი ელოდება Bool-შესატანი მითითება, იყო თუ არა დაჯავშნული დასახელება (სათაური). CreateResponse და WriteLine ქმედებები იგივეა, რაც იყო მე-8 თავში. აქ გამოიყენებოდა SendReply ქმედება, ვინაიდან იგი იყო დაკავშირებული საწყის Receive ქმედებასთან.

ამ პროექტში, ვინაიდან არაა არავითარი Receive ქმედება, ჩვენ გამოვიყენებთ Send ქმედებას. საყურადღებოა, რომ EndpointAddress აწყობილია მისამართის გამოყენებით (პორტის ნომერი), რომელიც გათვალისწინებულია შესატან მოთხოვნაში.

20.3. აპლიკაციის რეალიზაცია

შემდეგი ბიჯი არის აპლიკაციის რეალიზაცია. არსებობს მოვლენათა რამდენიმე დამმუშავებელი (event handlers-обработчики события), რომელთა რეალიზაცია აუცილებელია, აგრეთვე მეთოდები, რომლებიც გამოიძახება სტატიკური ApplicationInterface კლასით.

20.3.1. მხარდაჭერა სამუშაო პროცესების ეგზემპლარებისათვის

აპლიკაცია თვალყურს უნდა ადევნებდეს სამუშაო პროცესის ეგზემპლარებს, ამიტომაც მას შეუძლია განაახლოს სწორი ეგზემპლარი. ამის შესრულება შესაძლებელია მარტივად ობიექტის ლექსიკონით.

გახსენით Reservations.xaml.cs ფაილი და დაამატეთ კლასის წევრები მომხმარებლის ServiceHost-ის ქვემოთ:

```
private IDictionary<Guid, WorkflowApplication> _incomingRequests;
private IDictionary<Guid, WorkflowApplication> _outgoingRequests;
```

ისინი იყენებს სამუშაო პროცესის ეგზემპლარის იდენტიფიკატორს, როგორც ლექსიკონის გასაღებს და WorkflowApplication ობიექტს, როგორც მნიშვნელობას. ვინაიდან აპლიკაცია ამუშავებს ორივე სამუშაო პროცესს SendRequest და ProcessRequest, ამიტომაც საჭირო იქნება ლექსიკონის ორი ობიექტი. დავამატოთ კონსტრუქტორში კოდი ამ ობიექტების ინიციალიზებისათვის:

```
_incomingRequests = new Dictionary<Guid, WorkflowApplication>();
_outgoingRequests = new Dictionary<Guid, WorkflowApplication>();
```

საჭიროა კიდევ ერთი მცირე ცვლილება მომხმარებლის CreateRequest ქმედებაში. სამუშაო პროცესის ეგზემპლარის ID გამოყენებულ უნდა იქნას როგორც ReservationRequest კლასის RequestID ველი. აპლიკაცია მას გამოიყენებს პროცესის განახლების დროს. გავხსნათ CreateRequest.cs ფაილი და შევცვალოთ გამოძახება, რომელიც ქმნის ReservationRequest კლასს, ალტერნატიული კონსტრუქტორის გამოსაყენებლად, რომელიც იღებს მეხუთე პარამეტრს RequestID -თვის. დავამატოთ მუქი სტრიქონი კოდის შემდეგ ტექსტში:

```
// Create a ReservationRequest class and populate
    // it with the input arguments
   ReservationRequest r = new ReservationRequest
    (
      Title.Get(context),
      Author.Get(context),
      ISBN.Get(context),
      new Branch
```

```
                {
       BranchName = app.Settings["Branch Name"].Value,
        BranchID = new Guid(app.Settings["ID"].Value),
        Address = app.Settings["Address"].Value
       },
     context.WorkflowInstanceId    // !!!
   );
```

20.3.2. მოვლენათა დამმუშავებელი

ახალი მოთხოვნის შესაქმნელად მომხმარებელი შეავსებს ***ავტორის, სათაურის, ISBN*** ველებს და აამოქმედებს Send Request ღილაკს. ამ მოვლენის ღილაკის რეალიზება მოცემულია 20.9 ლისტინგში, Reservations.xaml.cs ფაილში.

// --- ლისტინგი 20.9 -----*Click Event -ის რეალიზება* -----

```
private void btnRequest_Click(object sender,
                                RoutedEventArgs e)
{
  // Setup a dictionary object for passing parameters
   Dictionary<string, object> parameters = new
                      Dictionary<string, object>();
   parameters.Add("Author", txtAuthor.Text);
   parameters.Add("Title", txtTitle.Text);
   parameters.Add("ISBN", txtISBN.Text);
   parameters.Add("Writer", new
               ListBoxTextWriter(lstEvents));
   WorkflowApplication i = new WorkflowApplication(new
              SendRequest(), parameters);
  _outgoingRequests.Add(i.Id, i);
   i.Run();
 }
```

ამ მეთოდის პირველი ნაწილი ჩვენთვის ნაცნობია. იგი იყენებს ობიექტის ლექსიკონს შემავალი არგუმენტების შესანახად, რომლებიც უნდა გადაეცეს სამუშაო პროცესს. შემდეგ იგი ქმნის WorkflowApplication-ს, რომლის კონსტრუქტორსაც გადაეცემა პარამეტრები:

– სამუშაო პროცესების დეფინიცია
– ობიექტის ლექსიკონი, რომელიც შეიცავს შემავალ არგუმენტებს:

WorkflowApplication შემდეგ ემატება _outgoingRequests კოლექციას. ბოლოს, ეგზემპლარი გაიშვება Run () მეთოდით.

რეკომენდაცია:

წინა პროექტებში ჩვენ ვიყენებდით WorkflowInvoker კლასის Invoke () მეთოდს, რათა დაწყებულიყო WorkflowApplication.

ეს მიდგომა იწყებს სამუშაო პროცესს სინქრონულად; სამუშაო პროცესი შესრულდება გამომძახებლის შესრულების ნაკადში (caller's thread). ეს ნიშნავს, რომ აპლიკაცია ბლოკირებულია მანამ, სანამ სამუშაო პროცესი არ გადავა ლოდინის რეჟიმში. მაგრამ ეს ის არაა, რაც ჩვენ გვინდა ამ პროექტში. ჩვენ გვინდა, რომ სამუშაო პროცესი დაიწყოს თავის საკუთარ შესრულების ნაკადში, მაშინ როცა აპლიკაციას შეუძლია განაგრძოს რეაგირება მოვლენებზე (და შემავალ შეტყობინებებზე). Run () მეთოდის გამოყენება წყვეტს სწორედ ამ ამოცანას.

მოთხოვნების სიის ფორმაზე მოთავსებულია ღილაკები Reserve და Cancel, რომელთაც იყენებს მომხმარებელი იმის მისათითებლად, თუ რომელი ელემენტი იყო გამოყენებადი.

20.10 ლისტინგი აღწერს ამ ღილაკებისათვის მოვლენათა დამმუშავებლების რეალიზაციას. დავამატოთ ეს მეთოდები Reservations კლასში.

// -- ლისტინგი 20.10 -- *Reserve და Cancel ღილაკების რეალიზაცია* ---

```
   // Handle the Reserve button click event
 private void Reserve(object sender, RoutedEventArgs e)
  {
   // Get the instanceID from the Tag property
   FrameworkElement fe = (FrameworkElement)sender;
   Guid id = (Guid)fe.Tag;
   ResumeBookmark(id, true);
  }
  // Handle the Cancel button click event
 private void Cancel(object sender, RoutedEventArgs e)
  {
   // Get the instanceID from the Tag property
    FrameworkElement fe = (FrameworkElement)sender;
    Guid id = (Guid)fe.Tag;
    ResumeBookmark(id, false);
  }
 private void ResumeBookmark(Guid id, bool bReserved)
```

```
{
 WorkflowApplication i = _incomingRequests[id];
   try
    {
      i.ResumeBookmark("GetResponse", bReserved);
    }
   catch (Exception e)
    {
     AddEvent(e.Message);
    }
}
```

ეს მოვლენის დამმუშავებლები იღებს სამუშაო პროცესის ეგზემპლარის ID-ს ღილაკის Tag თვისებიდან. შემდეგ იძახებენ ResumeBookmark () მეთოდს, მიაწოდებს true-ს ან false-ს, იმის მიხედვით, თუ რომელი ღილაკი იყო ამოქმედებული. ResumeBookmark () მეთოდი მიიღებს WorkflowApplication-ს _incomingRequests-კოლექციიდან და გამოიძახებს მის ResumeBookmark () მეთოდს. გადაეცემა სანიშნის სახელი (bookmark name) და მნიშვნელობა, რომელშიც ეგზემპლარი განახლედება (resumed).

20.4. ApplicationInterface მეთოდები

ჩვენ განვსაზღვრეთ ApplicationInterface კლასის სამი მეთოდი. ახლა უნდა უზრუნველვყოთ მათი რეალიზაცია MainWindow კლასში. ამ მეთოდების რეალიზაცია მოცემულია 20.11 ლისტინგში.

//--- ლისტინგი 20.11---ApplicationInterface კლასის მეთოდების რეალიზაცია ----

```
public void RequestBook(ReservationRequest request)
 {
  // Setup a dictionary object for passing parameters
  Dictionary<string, object> parameters = new
              Dictionary<string, object>();
   parameters.Add("request", request);
   parameters.Add("Writer", new
              ListBoxTextWriter(lstEvents));
  WorkflowApplication i =
           new WorkflowApplication(new
        ProcessRequest(), parameters);
  request.InstanceID = i.Id;
  _incomingRequests.Add(i.Id, i);
```

```
      i.Run();
   }

public void RespondToRequest(ReservationResponse response)
 {
  Guid id = response.RequestID;
  WorkflowApplication i = _outgoingRequests[id];
  try
    {
     i.ResumeBookmark("GetResponse", response);
    }
  catch (Exception e2)
      {
        AddEvent(e2.Message);
      }
 }
public void AddNewRequest(ReservationRequest request)
   {
     this.requestList.Dispatcher.BeginInvoke
                (new Action(() =>
           this.requestList.Items.Add(request)));
   }
```

RequestBook() მეთოდი ანალოგიურია btnRequest_Click () მეთოდის. იგი გამოიძახება მაშინ, როცა შემავალი შეტყობინება მიღებულია ServiceHost -დან და სერვის კონტრაქტის RequestBook მეთოდი მითითებულია. ის აგებს ობიექტის ლექსიკონს ერთი არგუმენტის შესანახად, ქმნის WorkflowApplication-ს, ამატებს მას _incomingRequests კოლექციაში, ხოლო შემდეგ აამოქმედებს სამუშაო პროცესს.

RespondToRequest () მეთოდი ასევე გამოიძახება ServiceHost-დან მიღებული შეტყობინებით. იგი გამოიძახება მაშინ, როცა RespondToRequest მეთოდია მითითებული. ეს ხდება მაშინ, როცა სხვა ფილიალები აგზავნიან უკან პასუხს შემოსულ მოთხოვნაზე. იგი იღებს WorkflowApplication-ს _outgoingRequests კოლექციიდან და აღადგენს სანიშნეს, გამავალს ReservationResponse კლასში.

AddNewRequest () გამოიძახება ProcessRequest სამუშაო პროცესით, როცა მიიღბა ახალი შეტყობინება. ეს ხდება InvokeMethod ქმედების დახმარებით. იგი უბრალოდ დაამატებს ჩანაწერს ListView-კონტროლის RequestList-ელემენტში. ვინაიდან ის გამოძახებულ უნდა იქნას სამუშაო პროცესის შესრულებად ნაკადში, Dispatcher კლასი

გამოიყენებს შესასრულებლად Add () მეთოდს main window-ის შესრულებადი ნაკადით. Reservations.xaml.cs-ის სრული რეალიზაცია მოცემულია 20.12 ლისტინგში.

```
// --- ლისტინგი 20.12 --- Reservations.xaml.cs-ის სრული რეალიზაცია -----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
using System.ServiceModel.Description;
using System.ServiceModel.Channels;
using System.Activities;
using System.Xml.Linq;
using System.Configuration;

namespace LibraryReservation
{
   public partial class MainWindow : Window
   {
      private ServiceHost _sh;
      private IDictionary<Guid, WorkflowApplication> _incomingRequests;
      private IDictionary<Guid, WorkflowApplication> _outgoingRequests;

      public MainWindow()
      {
         InitializeComponent();
```

```
    ApplicationInterface._app = this;
    _incomingRequests = new Dictionary<Guid, WorkflowApplication>();
    _outgoingRequests = new Dictionary<Guid, WorkflowApplication>();
}

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Open the config file and get the name for this branch
    // and its network address
    Configuration config = ConfigurationManager
        .OpenExeConfiguration(ConfigurationUserLevel.None);
    AppSettingsSection app =  (AppSettingsSection)config.GetSection("appSettings");
    string adr = app.Settings["Address"].Value;
    // Display the Branch name on the form
    lblBranch.Content = app.Settings["Branch Name"].Value;

    // Create the ServiceHost
    _sh = new ServiceHost(typeof(ClientService));

    // Add the Endpoint
    string szAddress = "http://localhost:" + adr + "/ClientService";
    System.ServiceModel.Channels.Binding bBinding = new BasicHttpBinding();
    _sh.AddServiceEndpoint(typeof(ILibraryReservation),
        bBinding, szAddress);
    // Open the ServiceHost to listen for messages
    _sh.Open();
    // Test the ListBoxTextWriter
    //ListBoxTextWriter lbtw = new ListBoxTextWriter();
    //lbtw.Write("This is a test");
}
private void Window_Unloaded(object sender, RoutedEventArgs e)
{
    // Terminate the service host
    _sh.Close();
}

private void btnRequest_Click(object sender, RoutedEventArgs e)
```

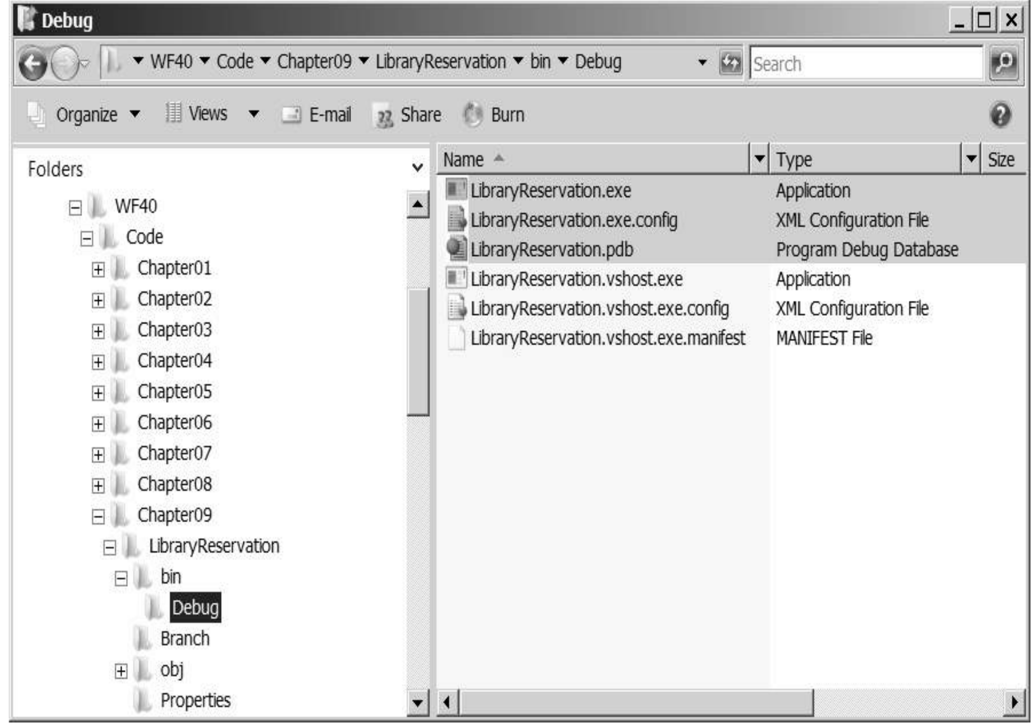
```
{
    // Setup a dictionary object for passing parameters
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("Author", txtAuthor.Text);
    parameters.Add("Title", txtTitle.Text);
    parameters.Add("ISBN", txtISBN.Text);
    parameters.Add("Writer", new ListBoxTextWriter(lstEvents));

    WorkflowApplication i =
        new WorkflowApplication(new SendRequest(), parameters);

    _outgoingRequests.Add(i.Id, i);
    i.Run();
}

// Handle the Reserve button click event
private void Reserve(object sender, RoutedEventArgs e)
{
    // Get the instanceID from the Tag property
    FrameworkElement fe = (FrameworkElement)sender;
    Guid id = (Guid)fe.Tag;
    ResumeBookmark(id, true);
}
// Handle the Cancel button click event
private void Cancel(object sender, RoutedEventArgs e)
{
    // Get the instanceID from the Tag property
    FrameworkElement fe = (FrameworkElement)sender;
    Guid id = (Guid)fe.Tag;
    ResumeBookmark(id, false);
}
private void ResumeBookmark(Guid id, bool bReserved)
{
    WorkflowApplication i = _incomingRequests[id];
    try
    {
        i.ResumeBookmark("GetResponse", bReserved);
```

```
        }
        catch (Exception e)
        {
            AddEvent(e.Message);
        }
    }
    public void RequestBook(ReservationRequest request)
    {
        // Setup a dictionary object for passing parameters
        Dictionary<string, object> parameters = new Dictionary<string, object>();
        parameters.Add("request", request);
        parameters.Add("Writer", new ListBoxTextWriter(lstEvents));

        WorkflowApplication i = new WorkflowApplication(new ProcessRequest(), parameters);
        request.InstanceID = i.Id;
        _incomingRequests.Add(i.Id, i);
        i.Run();
    }
    public void RespondToRequest(ReservationResponse response)
    {
        Guid id = response.RequestID;
        WorkflowApplication i = _outgoingRequests[id];
        try
        {
            i.ResumeBookmark("GetResponse", response);
        }
        catch (Exception e2)
        {
            AddEvent(e2.Message);
        }
    }
    public void AddNewRequest(ReservationRequest request)
    {
        this.requestList.Dispatcher.BeginInvoke
            (new Action(() => this.requestList.Items.Add(request)));
    }
    public ListBox GetEventListBox()
```

```
    {
       return this.lstEvents;
    }
   private void AddEvent(string szText)
    {
       lstEvents.Items.Add(szText);
    }
  }
}
```

20.5. აპლიკაციის ამუშავება

როგორც წინა თავში, აქაც საჭიროა აპლიკაციის რამდენიმე კოპიოს ერთად გაშვება, თითოეული თავისი კონფიგურაციის ფაილიის ვერსიით. თავიდან საჭიროა F6 კლავიშის ამოქმედება solution-ის (გადაწყვეტის) აღსადგენად და კომპილატორის შენიშვნების აღმოსაფხვრელად. შევქმნათ ახალი ფოლდერი LibraryReservation-ფოლდერის ქვეშ, რომელიც იძახებს ფილიალებს. შემდეგ დავაკოპიროთ ფილიალის ფოლდერში ფაილები, რომლებიც 20.7 ნახაზზეა ნაჩვენები.



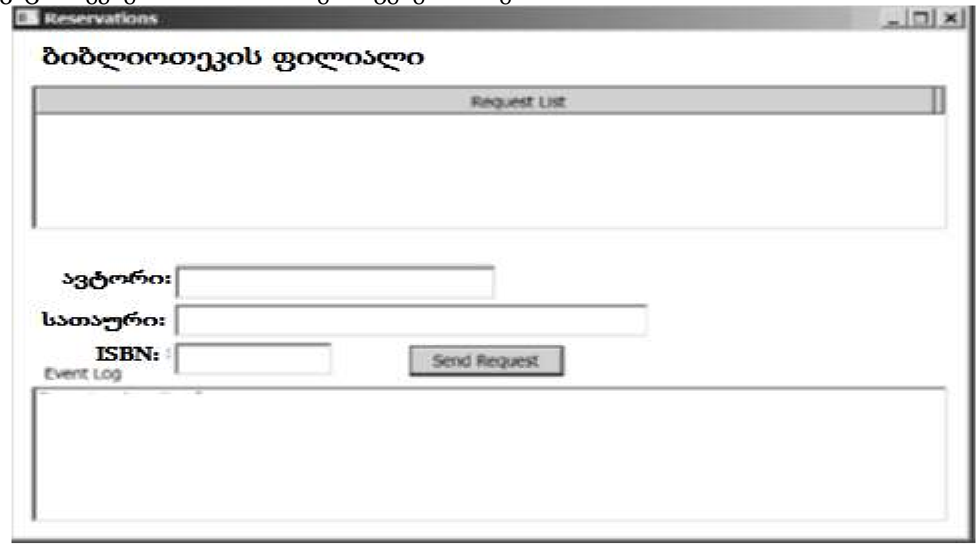
ნახ.20.7

გავხსნათ LibraryReservation.exe.config ფაილი (ფილიალის ქვეფოლდერში) და შევასწოროთ შემდეგნაირად:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
      <add key="Branch Name" value="Southwest Regional"/>
      <add key="ID" value="{CA62F4ED-FACF-4835-8468-16CAAC298F4C}"/>
      <add key="Address" value="8730"/>
      <add key="Request Address" value="8000"/>
  </appSettings>
</configuration>
```

შენიშვნა: თუ შედეგი შეცდომითაა მიღებული, უნდა ვცადოთ აპლიკაციის გაშვება ადმინისტრატორის უფლებებით.

ფილიალის ფოლდერში LibraryReservation.exe ფაილი ორჯერ „დავკლიკოთ“. აპლიკაციას ექნება 20.8 ნახაზზე ნაჩვენები სახე.



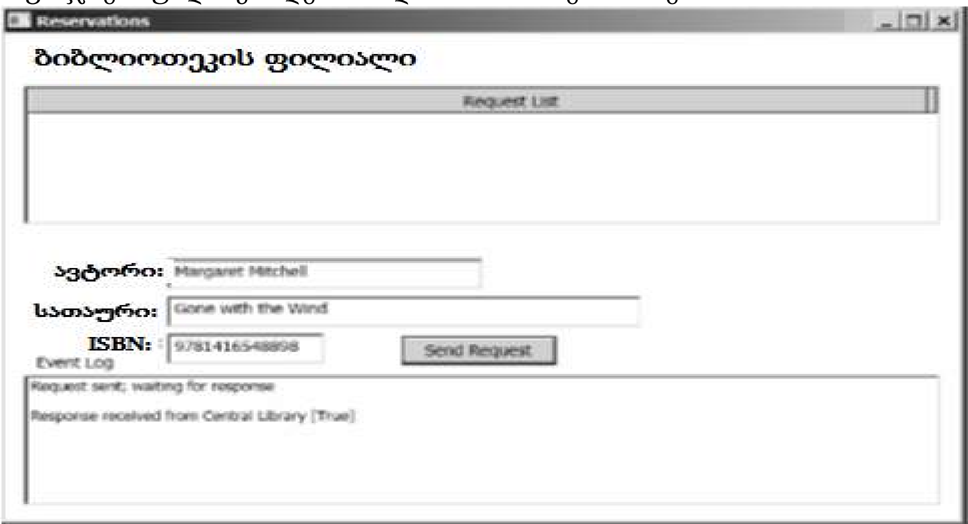
ნახ.20.8

Visual Studio-ში F5-.ით გავმართოთ აპლიკაცია. ანალოგიური ფანჯარა უნდა მივიღოთ, ოღონდ სათაურით - ცენტრალური ბიბლიოთეკა. ხედვის გასაუმჯობესებლად ფანჯრები განვაცალკევოთ.

ერთ-ერთ აპლიკაციაში შევიყვანოთ ავტორი, სათაური და ISBN და ავამოქმედოთ ღილაკი „მოთხოვნის გაგზავნა“. მოთხოვნა უნდა გამოჩნდეს მეორე ფანჯრის

მოთხოვნების სიაში. დააჭირეთ Reserve ღილაკს მეორე აპლიკაციაში. გამოჩნდება შეტყობინება პირველი ფანჯრის მოვლენების ჟურნალში, რომ პასუხი მიღებულია.

ფანჯრებს უნდა ჰქონდეს 20.9 და 20.10 ნახაზების სახე.



ნახ.20.9. მოთხოვნის გაგზავნა ფილიალი ბიბლიოთეკიდან

ნახ.20.10. მოთხოვნის დამუშავება ცენტრალურ ბიბლიოთეკაში

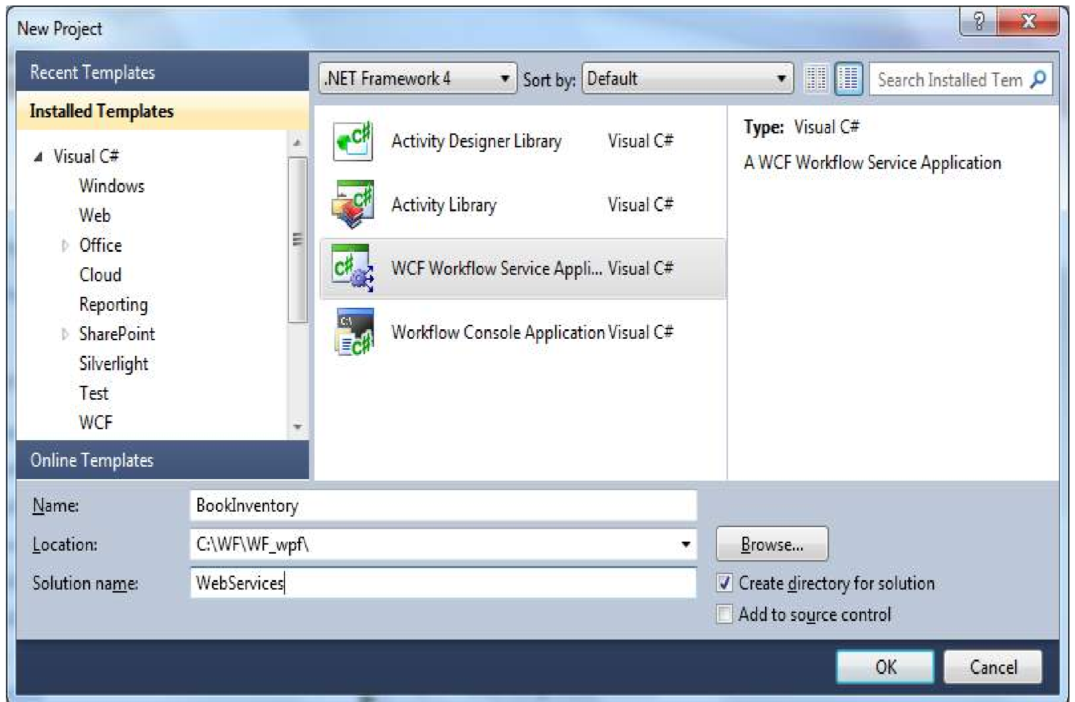
ვცადოთ რამდენიმე მოთხოვნის გაგზავნა ორივე ფანჯრიდან. ასევე შევამოწმოთ Cancel ღილაკი და დავრწმუნდეთ, რომ საპასუხო შეტყობინება მოვლენათა ჟურნალში (მეორე აპლიკაციისთვის) არის [false].

XXI თავი
Web - სერვისები

სამუშაო პროცესები შეიძლება განთავსდეს ვებსერვისში, რომელიც უზრუნველყოფს იდეალურ საშუალებას სამუშაო პროცესის გადაწყვეტილების მისაწოდებლად არამუშა პროცესის კლიენტებისთვის, როგორიცაა ვებ-აპლიკაციები. ვებ-სერვისი იღებს მოთხოვნას, ასრულებს მის სათანადო გადამუშავებას და აბრუნებს პასუხს. ეს, ბუნებრივია, სრულდება Receive და Send ქმედებებით, რომლებიც ჩვენ წინა თავებში განვიხილეთ. ვინაიდან ეს აქტიურობები ინტეგრირებულია Windows Communication Foundation (WCF) -თან, ჩვენ შეგვიძლია ადვილად შევქმნათ WCF სერვისები.

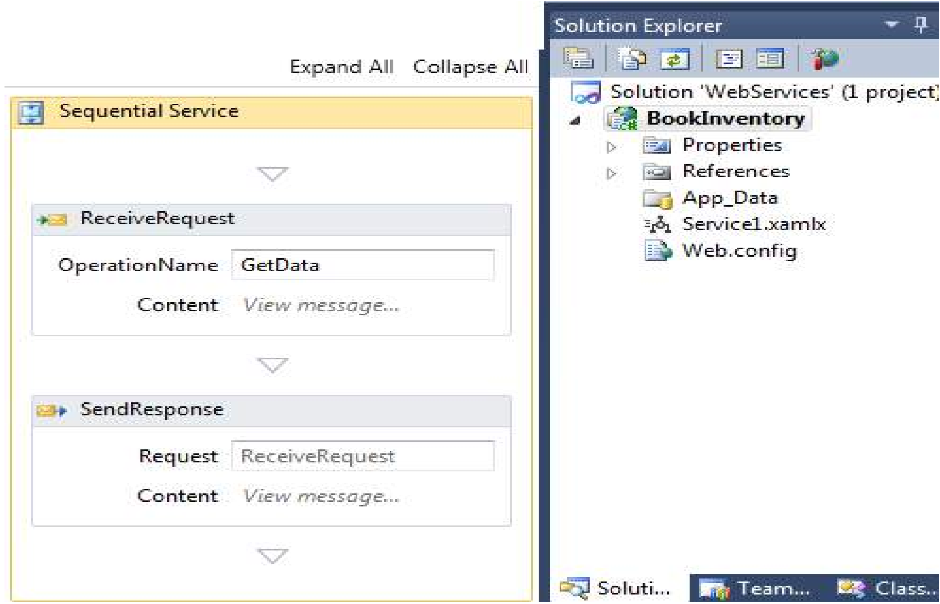
21.1. სამუშაო პროცესის სერვისის შექმნა

ავამუშავოთ Visual Studio და შევქმნათ ახალი პროექტი WCF Workflow Service Application template გამოყენებით. შევიტანოთ პროექტის სახელი BookInventory და solution-ის სახელი WebServices.



ნახ.21.1. *WCF*-ის ბიზნესპროცესის სერვისის აპლიკაციის შექმნა

შეიქმნება საინიციალიზაციო workflow Sequence ბლოკი, რომელიც შეიცავს Receive და SendReply ქმედებებს, როგორც 21.2 ნახაზზეა ნაჩვენები.



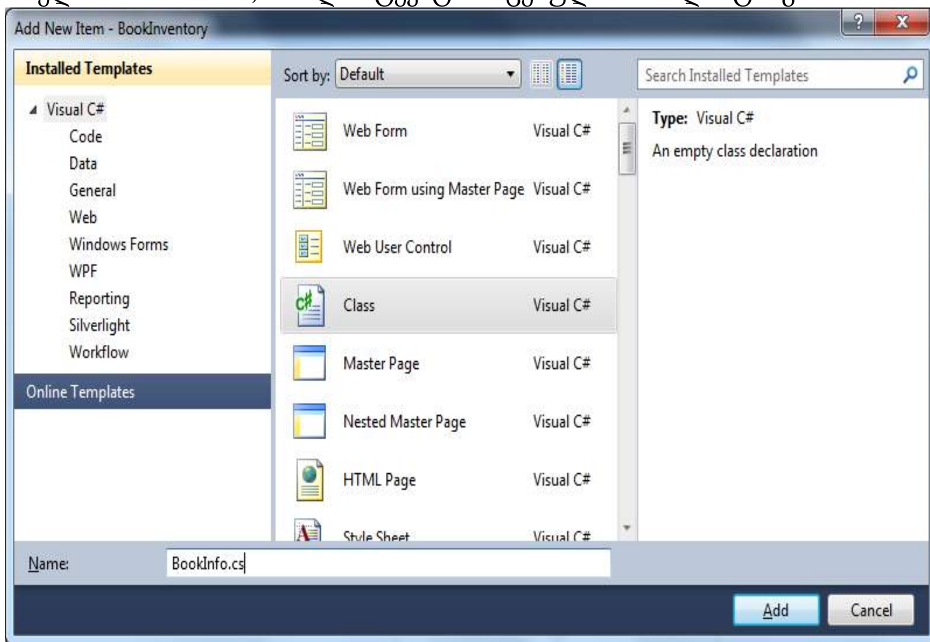
ნახ.21.2. ბიზნესპროცესის საწყისი თანამიმდევრობა

თავიდან საჭიროა ამ ქმედებების კონფიგურაცია სერვისის კონტრაქტის განსაზღვრის მიზნით, რომელსაც ისინი დააკმაყოფილებს. შემდეგ დავამატოთ დამუშავების სამუშაო პროცესი, რომელიც განხორციელდება Receive და SendReply ქმედებებს შორის.

შაბლონი ფაილში შექმნის საწყის სამუშაო პროცესს (ბიზნესპროცესს) სახელით Service1.xamlx. შევცვალოთ Solution Explorer-ში ეს სახელი BookInventory.xamlx -ით. *„სერვისი, რომლის შექმნაც გვინდა, ძებნის მითითებულ წიგნს და აბრუნებს უკან ყოველი ასლის მდგომარეობას, რომელიც ბიბლიოთეკას ეკუთვნის“.*

21.2. სერვისის კონტრაქტის განსაზღვრა

Solution Explorer-ში, მარჯვენა ღილაკით BookInventory პროექტზე ავირჩიოთ: Add->Class სახელით BookInfo.cs, რომლის ტექსტი მოცემულია 21.1 ლისტინგში.



ნახ.21.2

```
// ----- ლისტინგი 21.1 -----
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
namespace BookInventory
{
  public interface IBookInventory
    {
        [OperationContract]
        BookInfoList LookupBook(BookSearch request);
    }
    /*****************************************************/
    // მოთხოვნის შეტყობინების განსაზღვრა - BookSearch
    [MessageContract(IsWrapped = false)]
    public class BookSearch
```

```
{
    private String _ISBN;
    private String _Title;
    private String _Author;
    public BookSearch()
    {
    }
public BookSearch(String title, String author,
                  String isbn)
    {
        _Title = title;
        _Author = author;
        _ISBN = isbn;
    }
    #region Public Properties
    [MessageBodyMember]
    public String Title
    {
        get { return _Title; }
        set { _Title = value; }
    }

    [MessageBodyMember]
    public String Author
    {
        get { return _Author; }
        set { _Author = value; }
    }

    [MessageBodyMember]
    public String ISBN
    {
        get { return _ISBN; }
        set { _ISBN = value; }
    }
    #endregion Public Properties
}
```

```
/*****************************************************/
// BookInfo კლასის განსაზღვრა
[MessageContract(IsWrapped = false)]
public class BookInfo
{
    private Guid _InventoryID;
    private String _ISBN;
    private String _Title;
    private String _Author;
    private String _Status;

    public BookInfo()
    {
    }

  public BookInfo(String title, String author,
             String isbn, String status)
    {
        _Title = title;
        _Author = author;
        _ISBN = isbn;
        _Status = status;
        _InventoryID = Guid.NewGuid();
    }

    #region Public Properties
    [MessageBodyMember]
   public Guid InventoryID
    {
        get { return _InventoryID; }
        set { _InventoryID = value; }
    }
    [MessageBodyMember]
   public String Title
    {
        get { return _Title; }
        set { _Title = value; }
```

```
    }
    [MessageBodyMember]
   public String Author
    {
        get { return _Author; }
        set { _Author = value; }
    }
    [MessageBodyMember]
   public String ISBN
    {
        get { return _ISBN; }
        set { _ISBN = value; }
    }

    [MessageBodyMember]
    public String status
    {
        get { return _Status; }
        set { _Status = value; }
    }
    #endregion Public Properties
}
/*******************************************************/
// საპასუხო შეტყობინების განსაზღვრა - BookInfoList, რომელიც
// არის BookInfo კლასის სია
[MessageContract(IsWrapped = false)]
public class BookInfoList
{
    private List<BookInfo> _BookList;

    public BookInfoList()
    {
        _BookList = new List<BookInfo>();
    }
    [MessageBodyMember]
    public List<BookInfo> BookList
    {
```

```
            get { return _BookList; }
        }
    }
}
```

სერვისის კონტრაქტი IbookInventory შეიცავს ერთადერთ მეთოდს LookupBook(). იგი მონაცემებს გადასცემს BookSearch კლასს, რომელსაც აქვს სხვადასხვა თვისება, საჭირო წიგნის მოსაძებნად, მაგალითად ავტორს და დასახელებას. ის აბრუნებს უკან BookInfoList კლასს, რომელიც შეიცავს BookInfo კლასების კოლექციას.

F6 ამოქმედებით აიგება გადაწყვეტა (solution).

შენიშვნა:

MessageContract ატრიბუტი მიუთითებს, რომ ეს კლასი ჩართულ იქნება SOAP ბარათში. SOAP–ის გამოყენების დროს შეტყობინებები გადაიცემა XML–ის მსგავსი ფორმატირებადი ენით. ეს უზრუნველყოფს კლიენტებსა და სერვერს შორის მაღალი ხარისხის ურთიერთქმედების პლატფორმას. SOAP არის სტანდარტული პროტოკოლი, რომლის მხარდაჭერაც აქვს WCF-ს.

არსებობს აგრეთვე MessageBodyMember ატრიბუტი მის ყოველ public-თვისებაზე. ეს აუცილებელია WCF-შრისთვის რათა სწორად დაფორმატდეს SOAP შეტყობინება.

WCF-ის ბოლო წერტილის განსაზღვრისთვის არსებობს ინფორმაციის სამი პორცია, რომლებიც მითითებულ უნდა იქნას: მიერთება (binding), მისამართი და კონტრაქტი.

– **მიერთება** მიუთითებს იმ პროტოკოლს, რომელიც გამოიყენება (მაგ.,HTTP, TCP ან სხვ.);

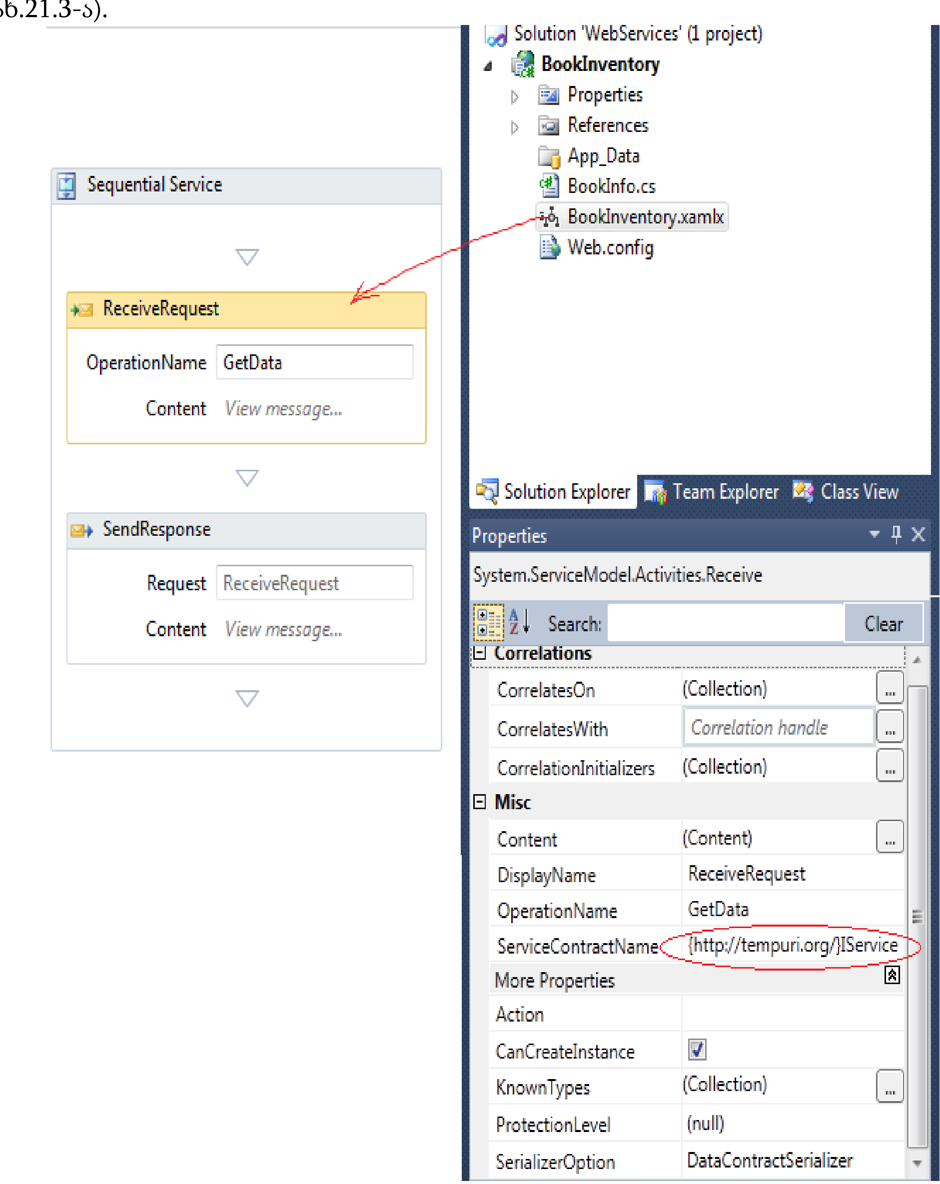
– **მისამართი** მიუთითებს სად უნდა ვიპოვოთ ბოლო წერტილი, და მისამართის ტიპს, რომლის გამოყენება დამოკიდებულია მიერთებაზე. მაგალითად, HTTP–მიერთებისას უნდა მიეთითოს URL, ხოლო TCP–თვის მისამართი იქნება სერვერის სახელი ან IP–მისამართი.

– **კონტრაქტი** განისაზღვრება ServiceContract–ით, რომელიც არის ინტერფეისი. იგი განსაზღვრავს მეთოდებს, რომლებიც მიწვდომადია ბოლო წერტილში.

ამგვარად, ჩვენ განვსაზღვრეთ შეტყობინებები, რომლებიც გადაიცემა სერვის–მეთოდების მიერ პარამეტრების სახით.

21.3. Receive და SendReply კონფიგურირება

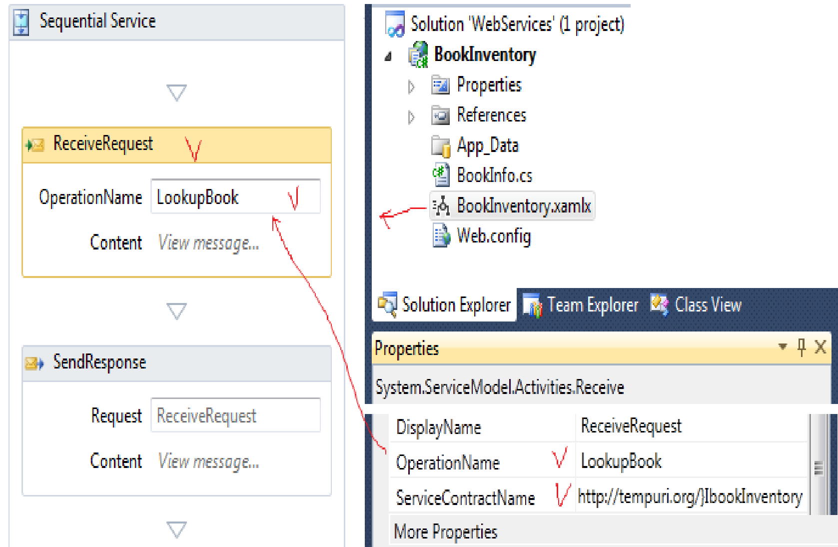
გავხსნათ BookInventory.xamlx ფაილი და ავირჩიოთ “ReceiveRequest” ქმედება (ნახ.21.3-ა).



ნახ.21.3-ა. საწყისი

თვისებათა ფანჯარაში ServiceContract თვისებას აქვს default-მნიშვნელობა {http://tempuri.org/}IService. შევცვალოთ Iservice სტრიქონი **IbookInventory**-ით. შევიტანოთ OperationName როგორც **LookupBook.**

ეკრანზე WorkflowService-დიზაინერში ქვედა მარცხენა კუთხეში დავკლიკოთ Variables ღილაკი. გამოჩნდება შაბლონი ორი ცვლადის შესაქმნელად (ნახ.21.3-ბ).

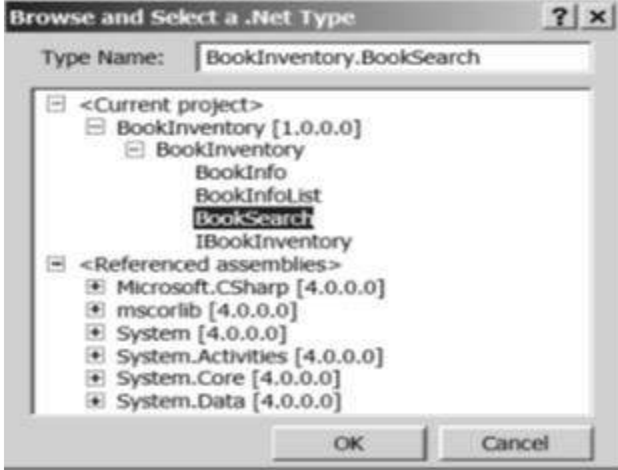


ნახ.21.3-ბ. საბოლოო

გადასამუშავებელი ცვლადი (handle variable) გამოიყენება პასუხის კორელაციისთვის იმ ეგზემპლართან, რომელმაც გააგზავნა მოთხოვნა.

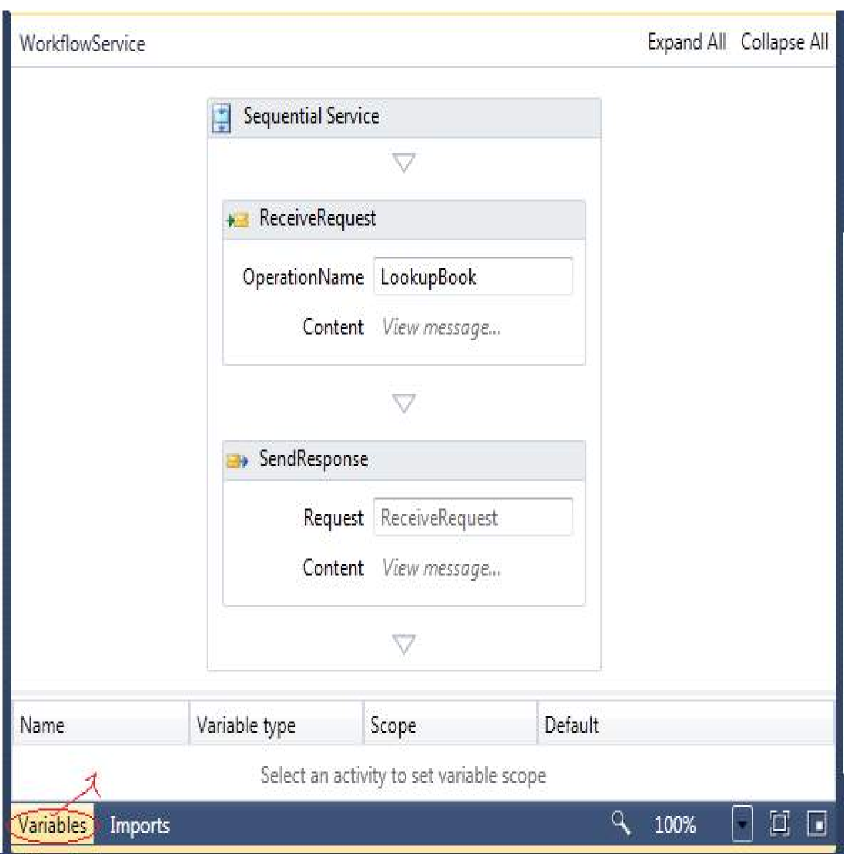
მონაცემთა ცვლადი იქმნება გადასაცემი მონაცემების (ინფორმაციის) მიზნით. გავასუფთავოთ ცვლადების არე (data) და შევქმნათ ორი ახალი ცვლადის სახელი.

პირველისთვის ავირჩიოთ სახელი Name search და ტიპი - Browse for Typs. ახალ დიალოგურ ფანჯარაში BookInventory ნაკრები გავაფართოვოთ BookSearch-ით (ნახ.21.4).

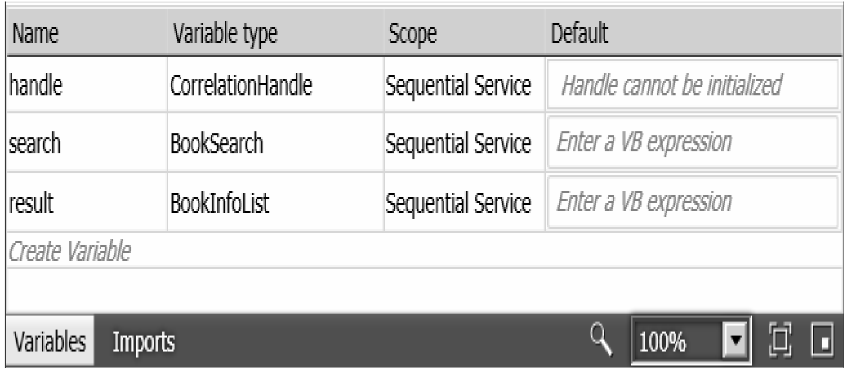


ნახ.21.4

მეორე ცვლადისათვის შეიტანეთ Name: result. ტიპი შეირჩევა Browse-დან, BookInfoList კლასით (ნახ.21.5). მიიღება 21.6 ნახაზზე მოცემული შემთხვევა.



ნახ.21.5. ცვლადები



Name	Variable type	Scope	Default
handle	CorrelationHandle	Sequential Service	Handle cannot be initialized
search	BookSearch	Sequential Service	Enter a VB expression
result	BookInfoList	Sequential Service	Enter a VB expression
Create Variable			

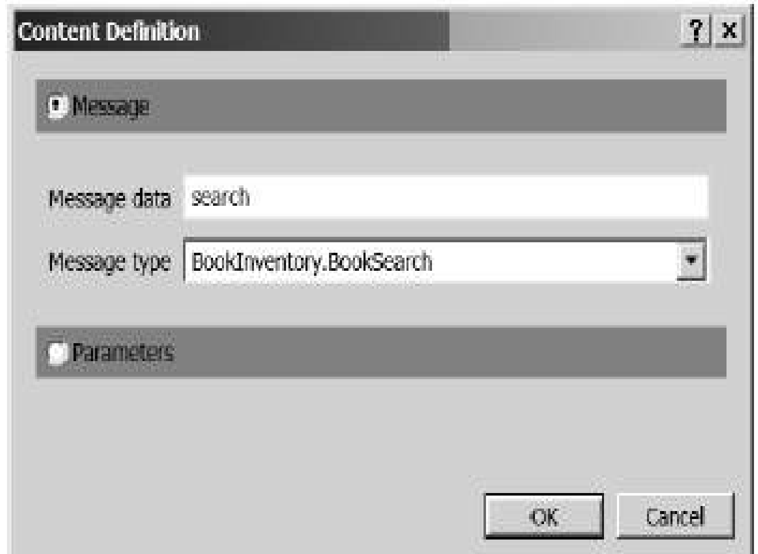
Variables Imports 100%

ნახ.21.6. ცვლადები განისაზღვრა ბიზნესპროცესისთვის

სამუშაო პროცესების დიზაინერში „ReceiveRequest“ (მოთხოვნების მიღების) ქმედებას აქვს view message (შეტყობინების ნახვის) ლინკი შინაარსის თვისებისთვის. მისი ამოქმედებით იხსნება დიალოგური ფანჯარა შემავალი შეტყობინების დასადგენად (შეიძლება ასევე სამ-წერტილიანი ღილაკის გამოყენებაც, თვისების გვერდით). შესასვლელი განისაზღვრება ორი ხერხით: შეტყობინებით ან პარამეტრების ერთობლიობით.

ამ თავში, მოგვიანებით ჩვენ განვიხილავთ მეორე ხერხსაც. ახლა დავაკვირდეთ, რომ რადიობუტონის გადამრთველი შეტყობინებისათვის სწორადაა არჩეული.

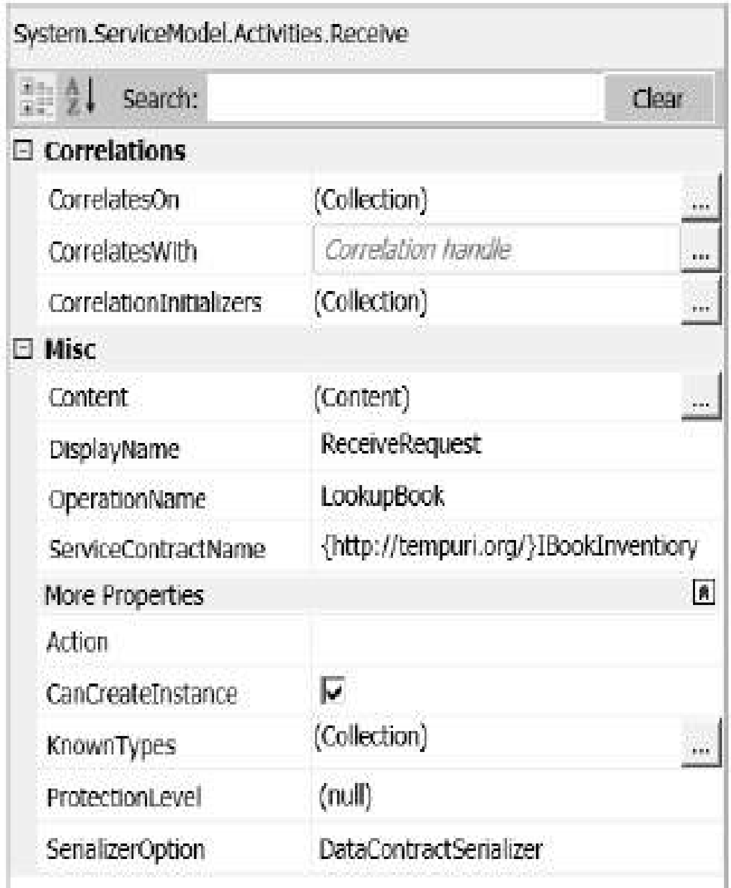
Message data თვისებისთვის შევიტანოთ **search.** ის მიუთითებს, რომ შემომავალი შეტყობინება უნდა ინახებოდეს search ცვლადში. Message ტიპისთვის ვირჩევთ BookInventory.BookSearch. დიალოგური ფანჯარა მოცემულია 21.7 ნახაზზე.



ნახ.21.7. შემავალი შეტყობინების განსაზღვრა

თვისებების ფანჯარა უნდა გამოიყურებოდეს 21.8 ნახაზზე ნაჩვენები სახით.

ვირჩევთ ქმედებას „SendResponse“ და ავამოქმედებთ view message ლინკს. კვლავ შევამოწმოთ, რომ არჩეულია შეტყობინების გადამრთველი. Message data თვისებისთვის შევიტანოთ result, ხოლო Message type თვისებისთვის ვირჩევთ BookInfoList კლასს.

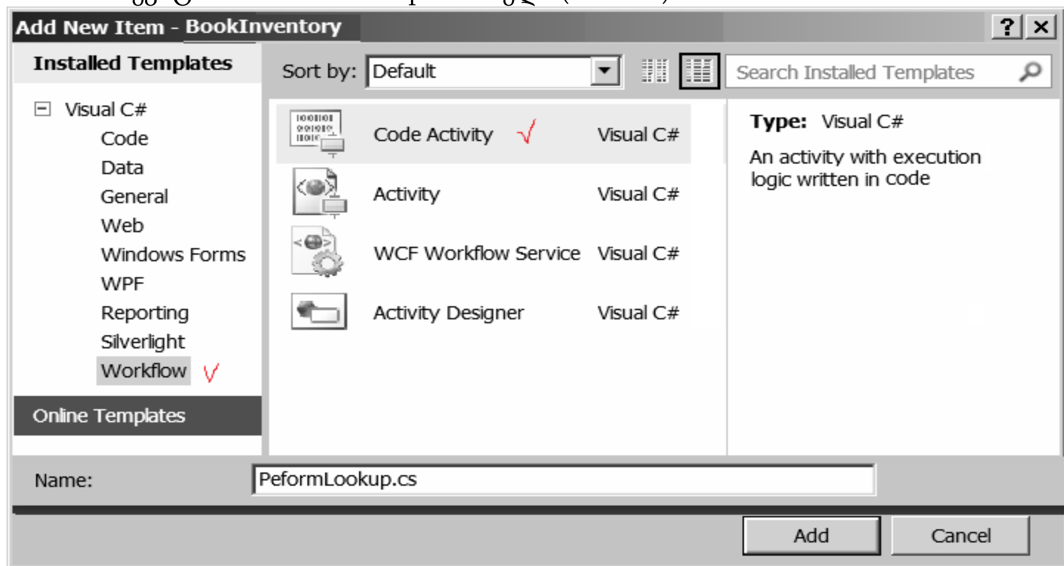


ნახ.21.8. Receive ქმედების თვისებათა ფანჯარა

21.4. PerformLookup აქტიურობის შექმნა

ამ პროექტისათვის შევქმნით მომხმარებლის ქმედებას (აქტიურობას) "lookup"-ის (ძებნის) შესასრულებლად. ფაქტობრივად, მარტივი იქნება ხისტად-კოდირებული მონაცემების დაბრუნება. რეალურ სიტუაციაში მან, ალბათ, უნდა შეასრულოს მონაცემთა ბაზისადმი მოთხოვნა საჭირო მონაცემების მისაღებად.

Solution Explorer-ში BookInventory პროექტზე მარჯვენა ღილაკით ვირჩევთ Add ➤ New Item და დიალოგში Workflow კატეგორიისათვის ვირჩევთ Code Activity-ს. Name-შოი შევიტანთ PerformLookup.cs სახელს (ნახ.21.9).



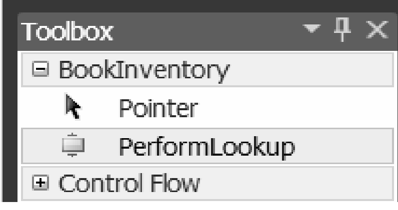
ნახ.21.9. მომხმარებლის ქმედების შექმნა

შევიტანოთ PerformLookup ქმედების რეალიზაციისთვის 21.2 ლისტინგში მოცემული კოდი.

```
// --------- ლისტინგი 21.2 ---------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
namespace BookInventory
{
    // ქმედება ქმნის BookInfoList კლასს, რომელიც არის
```

```
    // BookInfo კლასების კოლექცია. იგი იყენებს (BookSearch კლასის)
    //შემავალ პარამეტრებს შესაბამისი ელემენტების მოსაძებნად.
    // BookInfoList კლასი ბრუნდება გამომავალ პარამეტრში.
   public sealed class PerformLookup : CodeActivity
    {
        public InArgument<BookSearch> Search { get; set; }
        public OutArgument<BookInfoList> BookList { get; set; }

       protected override void Execute(CodeActivityContext context)
        {
         string author = Search.Get(context).Author;
         string title = Search.Get(context).Title;
         string isbn = Search.Get(context).ISBN;

         BookInfoList l = new BookInfoList();

        l.BookList.Add(new BookInfo(title, author, isbn, "Available"));
        l.BookList.Add(new BookInfo(title, author, isbn, "CheckedOut"));
        l.BookList.Add(new BookInfo(title, author, isbn, "Missing"));
        l.BookList.Add(new BookInfo(title, author, isbn, "Available"));
        BookList.Set(context, l);
       }
    }
}
```

F6-ით განვახორციელოთ აპლიკაციის აღდგენა. გავხსნათ BookInventory.xamlx ფაილი. გასათვალიწინებელია, რომ მომხმარებლის PerformLookup ქმედება არის ToolBox-ზე (ნახ.21.10).



ნახ.21.10

გადმოვიტანოთ PerformLookup ქმედება „ReceiveRequest" და „SendResponse" ქმედებებს შორის, როგორც ეს 21.11 ნახაზზეა ნაჩვენები.

ნახ.21.11

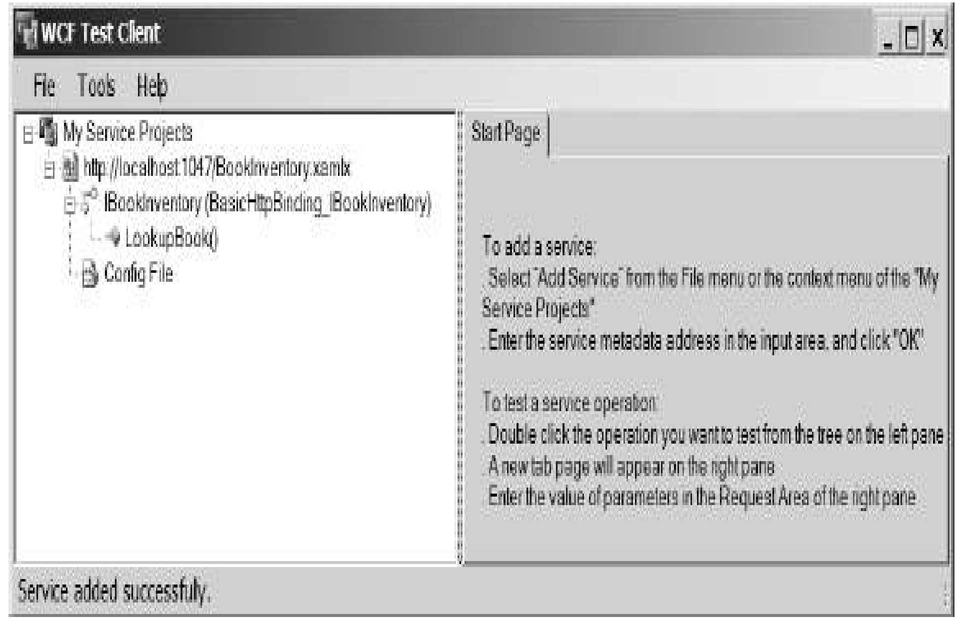
ავირჩიოთ PerformLookup ქმედება. Properties-ის ფანჯარაში, BookList თვისებისთვის შევიტანოთ result; Search თვისებისათვის კი search.

21.5. სერვისის ტესტირება

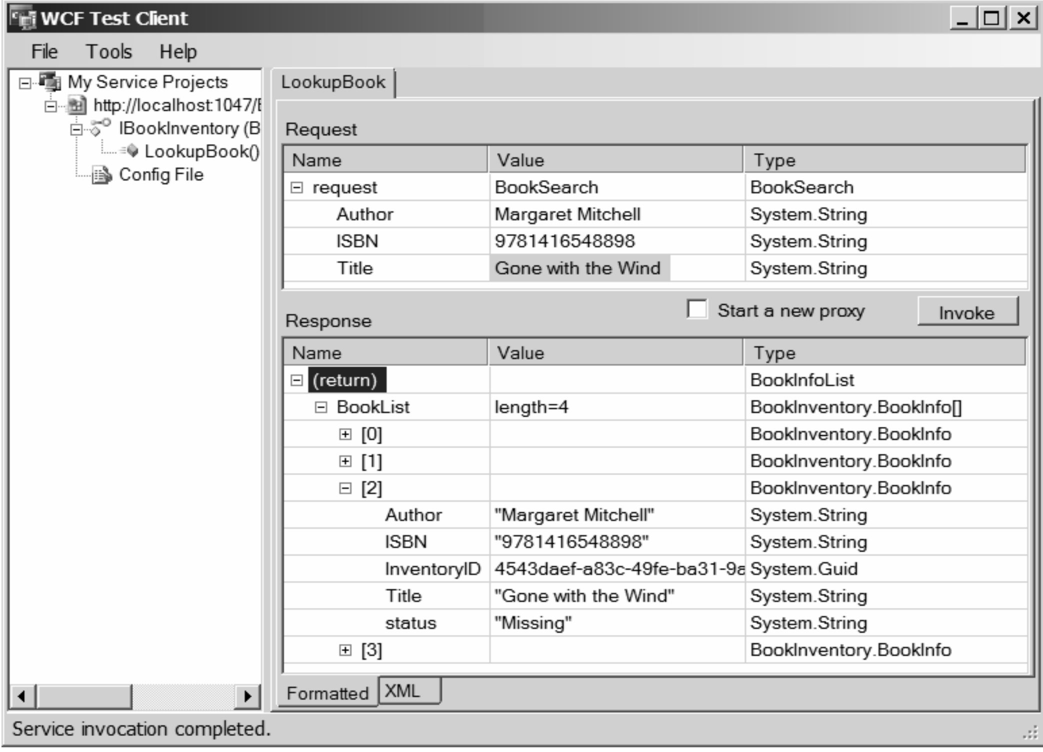
F5-ით ჩავატეროთ სერვისის გამართვა (debug). ვინაიდან ეს Web-სერვისია, Visual Studio ავტომატურად აამუშავებს WCF Test Client-ს. ეს მეტად მოსახერხებელი უტილიტაა. იგი ჩატვირთავს Web-სერვისებს და აღმოაჩენს მეთოდებს, რომლებიც გათვალისწინებულია. ისინი ჩანს 21.12 ნახაზის მარცხენა პანელზე.

LookupBook() მეთოდზე მაუსის 2-ჯერ დაჭერით მარჯვენა პანელის ზედა ნაწილში გამოიყოფა ადგილი შემოსული შეტყობინების განსათავსებლად. იგი მზადაა რთული შეტყობინებებისთვისაც, რომლებიც შეიცავს კლასებისა და თვისებების კოლექციებს.

შევიტანოთ ავტორი, ISBN-ნომერი, დასახელება. შემდეგ ავამოქმედოთ Invoke ღილაკი. გამოჩნდება შედეგები, რომლებიც ანალოგიურია 21.13 ნახაზზე ნაჩვენებისა.

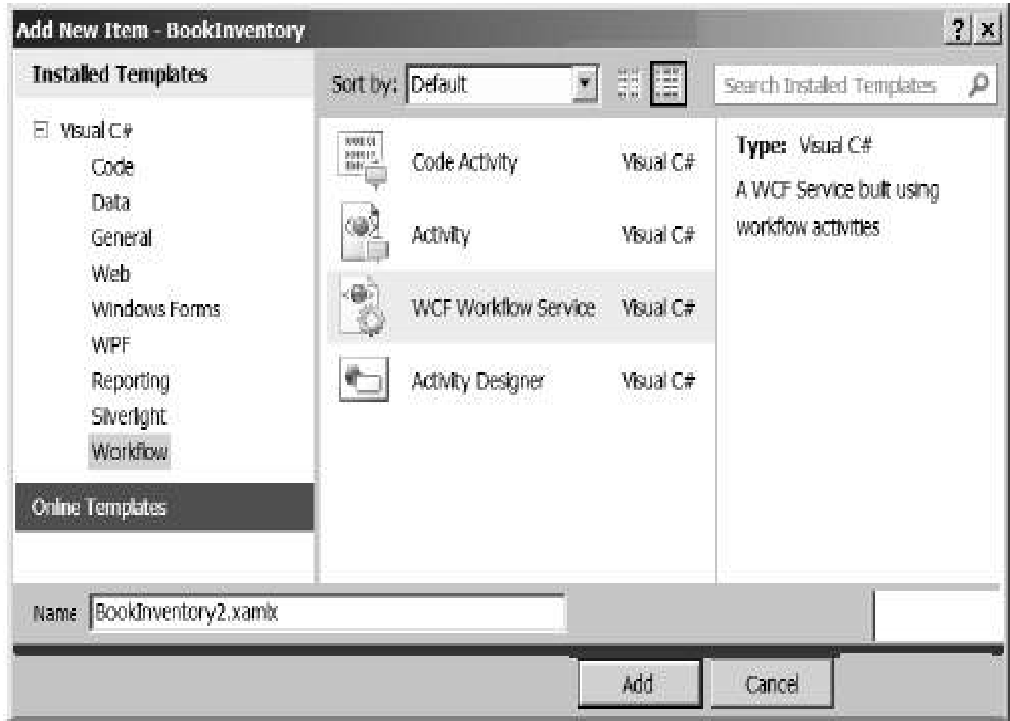


ნახ.21.12. **WCF Test Client** ინიციალიზაციის ფანჯარა



ნახ.21.13. **WCF Client** ტესტის სერვისის შედეგების ნახვა

სერვისი აბრუნებს BookInfo-ს ოთხ კლასს. 21.14 ნახაზზე მესამე ჩანაწერი გაფართოებულია, რათა დავინახოთ დაბრუნებული მონაცემების მაგალითი. მოცემულ კონკრეტულ ელემენტს აქვს სტატუსი Missing (დაკარგული, ამოვარდნილი).



ნახ.21.14. WCF სამუშაო პროცესის სერვისის შექმნა

შენიშვნა: თუ .xamlx ფაილი არის მიმდინარე ფაილი Visual Studio-ში, როცა F5-ვაჭერთ, მაშინ WCF Test Client გაიშვება, როგორც აქაა ნაჩვენები. თუ სხვა ფაილია აქტიური, მაშინ გამოჩნდება შესაბამისი კატალოგი და შედეგები. ის უნდა დაიხუროს და გააქტიურდეს ჩვენთვის საჭირო .xamlx ფაილი.

21.6. პარამეტრების გამოყენება

წინა პროექტის მიხედვით ვებსერვისში შესვლა განისაზღვრებოდა როგორც კლასი MessageContract ატრიბუტით. ესაა ტიპური გზა WCF სერვისის გამოსაძახებლად. მიუხედავად ამისა, იმის მაგივრად, რომ შეიქმნას შეტყობინების ერთი კლასი, რომელიც ყველა შემავალ მონაცემს შეიცავდეს, შესაძლებელია მათი გადაცემა სამუშაო პროცესების სერვისებისათვის ცალკეული პარამეტრების სახით. ამის სადემონსტრაციოდ შევქმნათ მეორე იდენტური სერვისი, რომელიც გამოიყენებს პარამეტრებს შეტყობინებათა მაგივრად.

21.7. მეორე სერვისის შექმნა

Solution Explorer-ში მარჯვენა ღილაკს ვაჭერთ BookInventory პროექტზე და ვირჩევთ

Add ➤ New Item.

ამ დიალოგში ვირჩევთ WCF Workflow Service შაბლონს, რომელიც Workflow კატეგორიაშია. 21.14 ნახაზზე ნაჩვენებია ეს, სახელით Name: **BookInventory2.xamlx**.

სამუშაო პროცესის დიზაინერში ქვემოთ მარცხნივ ავამოქმედოთ ცვლადების ღილაკი Variables. რამდენიმე ცვლადი უკვე შექმნილია პირველი სერვისის შექმნის დროს. წავშალოთ data ცვლადები და შევქმნათ ახალი ცვლადი, სახელით result. ცვლადის ტიპისთვის (type) ავირჩიოთ ArrayOf<T>. გამოჩნდება დიალოგური ფანჯარა <T> ტიპის ასარჩევად. ვირჩევთ Browse-ს და BookInventory კრებულიდან ვირჩევთ BookInfo კლასს. კიდევ დავამატოთ სამი String ტიპის ცვლადი სახელებით: ავტორი, დასახელება და ISBN. 21.15 ნახაზზე ნაჩვენებია ცვლადების სია.

Name	Variable type	Scope	Default
handle	CorrelationHandle	Sequential Service	Handle cannot be initialized
result	BookInfo[]	Sequential Service	Enter a VB expression
author	String	Sequential Service	Enter a VB expression
title	String	Sequential Service	Enter a VB expression
isbn	String	Sequential Service	Enter a VB expression
Create Variable			

Variables Imports 100%

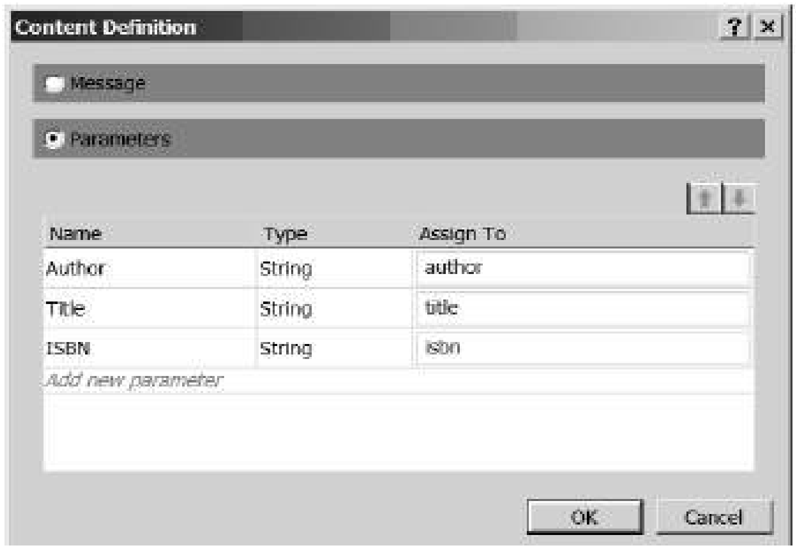
ნახ.21.15. ცვლადების სია

Properties-ის ფანჯარაში ServiceContract თვისებისთვის, შევცვალოთ IService კონტრაქტი წიგნით. ოპერაციის სახელში ჩავწეროთ LookupBook2.

შენიშვნა: პირველი სერვისის შექმნისას CanCreateInstance თვისება დაყენდა true-ში შაბლონით. მეორე სერვისისათვის იგი არის false-ში. შეამოწმეთ და დარწმუნდით, რომ ის გადაყვანილია true-ში.

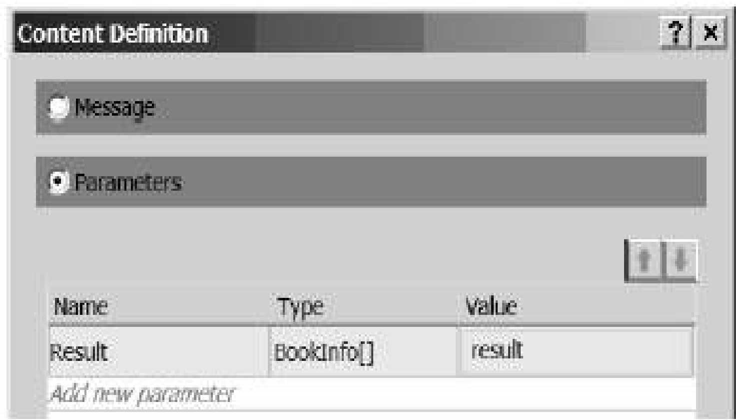
ავირჩიოთ „ReceiveRequest" ქმედება (მოთხოვნის მიღება) და ავამოქმედოთ Content ლინკი. ამჯერად დავაყენოთ გადამრთველი პოზიციაში „პარამეტრები".

პარამეტრები ყენდება ცვლადებისა და არგუმენტების ანალოგიურად. ავამოქმედოთ ლინკი „Add new parameter". შევიტანოთ Name როგორც ავტორი და დავაყენოთ Assign ავტორის თვისებით. დავამატოთ კიდევ ერთი პარამეტრი, სახელით Title და Assign დასახელებაზე. დავამატოთ მესამე პარამეტრი სახელით ISBN და Assign isbn-ზე. შევსებული გვერდს ექნება ასეთი სახე (ნახ21.16).



ნახ.21.16. ReceiveRequest პარამეტრების სია

დავაჭიროთ „SendResponse" ქმედების Content ლინკს. ავირჩიოთ გადამრთველი პარამეტრები და დავაჭიროთ ლინკს „Add new parameter". შევიტანოთ Name როგორც Result; ტიპისათვის ავირჩიოთ BookInventory.BookInfo[] ჩამოსაშლელი სიიდან. დიალოგურ ფანჯარას აქვს 21.17 ნახაზზე ნაჩვენები სახე.



ნახ.21.17. SendResponse პარამეტრების სია

21.8. მოდიფიცირებული PerformLookup ქმედების შექმნა

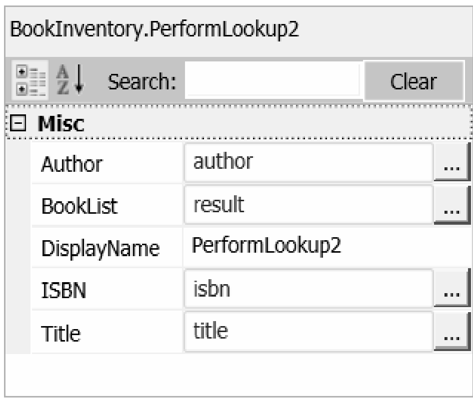
მომხმარებლის PerformLookup ქმედება, რომელიც ჩვენ შევქმენით პირველი სერვისისათვის, იყენებს BookSearch კლასს როგორც შესასვლელ არგუმენტს და აბრუნებს უკან BookInfoList კლასს. ახლა ჩვენ უნდა შევქმნათ სხვა სამომხმარებლო ქმედება, რომელიც იყენებს ცალკეულ პარამეტრებს. Solution Explorer-ში მაუსის მარჯვენა ღილაკით ვაჭერთ BookInventory პროექტზე და ვირჩევთ Add ➤ New Item. ვირჩევთ შაბლონიდან Code Activity-ს და სახელისათვის Name: PerformLookup2.cs. ამ ქმედების რეალიზაცია ნაჩვენებია 21.3 ლისტინგში.

```
// ლისტინგი --21.3 ----PerformLookup2.cs. რეალიზაცია----
using System;
using System.Collections.Generic;
using System.Activities;
namespace BookInventory
{
  // მომხმარებლის ქმედება ქმნის BookInfo მასივს და იყენებს შემავალ პარამეტრებს
  // შესაბამისი ეგზემპლარების მოსაძებნად. BookInfo მასივი ბრუნდება გამომავალ
  // პარამეტრში
    public sealed class PerformLookup2 : CodeActivity
    {
      public InArgument<String> Title { get; set; }
      public InArgument<String> Author { get; set; }
      public InArgument<String> ISBN { get; set; }
```

```
    public OutArgument<BookInfo[]> BookList{get;set; }
    protected override void Execute(CodeActivityContext context)
      {
          string author = Author.Get(context);
          string title = Title.Get(context);
          string isbn = ISBN.Get(context);

          BookInfo[] l = new BookInfo[4];

         l[0] = new BookInfo(title, author, isbn, "Available");
         l[1] = new BookInfo(title, author, isbn, "CheckedOut");
         l[2] = new BookInfo(title, author, isbn, "Missing");
         l[3] = new BookInfo(title, author, isbn, "Available");
        BookList.Set(context, l);
      }
    }
}
```

ეს კოდი მუშაობს ისევე, როგორც პირველი, იმისგან განსხვავებით, რომ შემავალი არგუმენტები მიეწოდება ცალ-ცალკე, და შედეგები ბრუნდება მასივად, და არა კლასში. F6-ის დაჭერით განვაახლოთ solution.

ავირჩიოთ BookInventory2.xamlx ფაილი და გადმოვიტანოთ PerformLookup2 ქმედება ინსტრუმენტების პანელიდან „ReceiveRequest" და „SendResponse" ქმედებებს შორის. თვისებების ფანჯარაში შევიტანოთ შესაბამისი მნიშვნელობები, როგორც 21.18 ნახაზზეა ნაჩვენები.

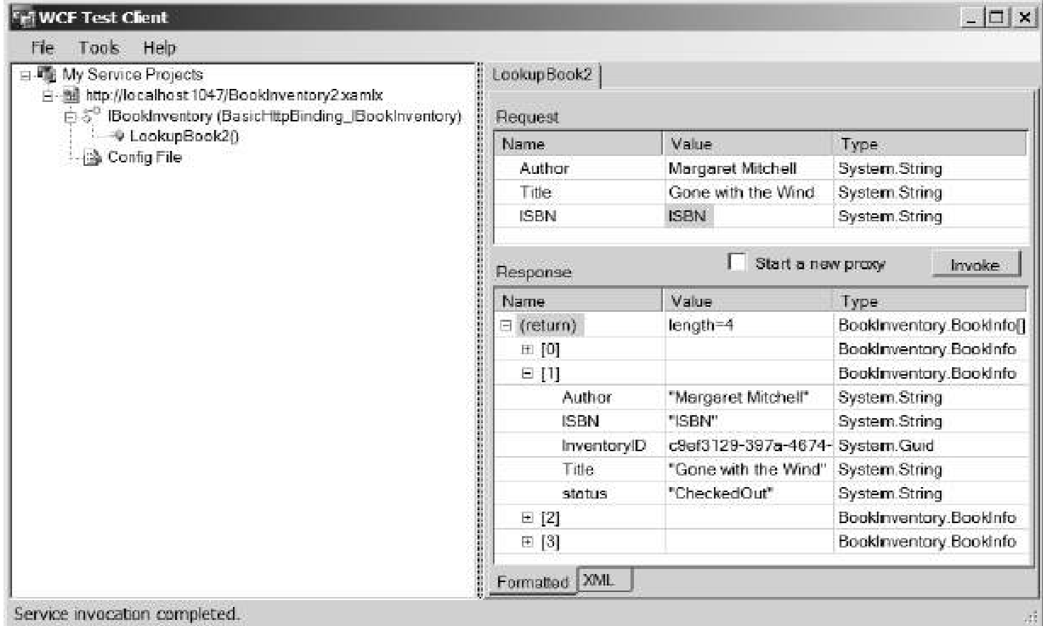


ნახ.21.18. PerformLookup2 ქმედების Properties ფანჯარა

21.9. სერვისის ხელახალი ტესტირება

დავრწმუნდეთ, რომ BookInventory2.xamlx ფაილი არის აქტიური Visual Studio-ში და F5-ით გავუშვათ დებაგის პროცესი კოდის გასამართად.

WCF Test Client უნდა ამოქმედდე ისე, როგორც პირველი სერვისის შემთხვევაში. 2-ჯერ დავკლიკოთ LookupBook2() მეთოდი, შევიტანოთ მოთხოვნის მონაცემები, და დავაჭიროთ Invoke ღილაკს. შედეგებს ექნება 21.19 ნახაზზე ნაჩვენები სახე.



ნახ.21.19. WCF Test Client

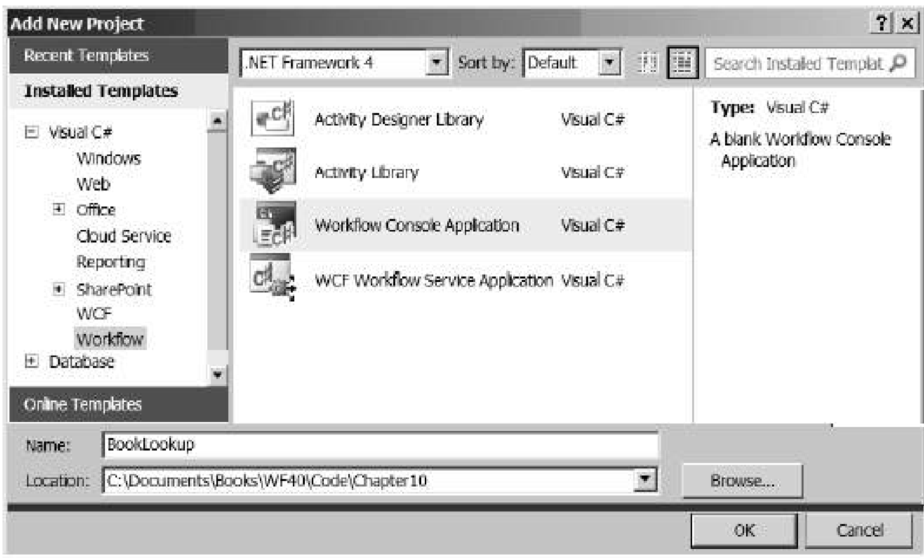
ფორმატი მცირედით განსხვადება პირველი სერვისისაგან, მაგრამ ფუნქციონირებს ძირითადად მის მსგავსად. ნახაზზე გაფართოებულია მეორე ჩანაწერი, რათა გამოჩნდეს, რომ შემოწმებულ იქნა ეს ეგზემპლარი.

მეორე სერვისისთვის არ შეგვიქმნია კონტრაქტი მომსახურებისათვის. ჩვენ მხოლოდ განვსაზღვრეთ პარამეტრები, რომლებიც გადასცა და უკან დაიბუნა სერვისმა. კონტრაქტი სერვისის მიწოდებისთვის იქმნება ავტომატურად.

შენიშვნა: Receive/SendReply წყვილის განსაზღვრისას, გვეძლევა არჩევანის უფლება: შეტყობიმებები ან პარამეტრები. ამ ორი ვარიანტის შერევა დაუშვებელია. თუ ვიყენებთ პარამეტრებს ქმედების მისაღებად, მაშინ არ შეიძლება შეტყობინების გამოყენება SendReply ქმედებისთვის. ამავდროულად, პარამეტრების გამოყენებისას ტიპებს არ უნდა ჰქონდეს ატრიბუტი MessageContract. საწინააღმდეგო შემთხვევაში ფიქსირდება საკმაოდ ხანგრძლივი განსაკუთრებული შემთხვევა შესრულების პროცესში.

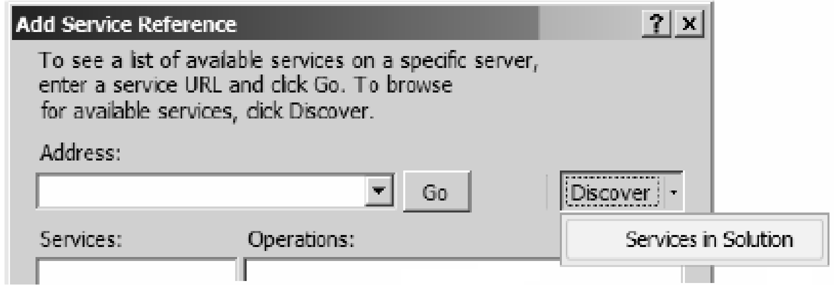
21.10. კლიენტის Workflow პროცესის შექმნა

ახლა უნდა შევქმნათ Client Workflow , რომელიც გამოიძახებს Web-სერვისებს. Solution Explorer-ში მარჩვენა ღილაკით ვირჩევთ Add ➤ New Project. შემდეგ ავირჩევთ Workflow Console Application და შეგვაქვს პროექტის სახელი: BookLookup (ნახ.21.20).



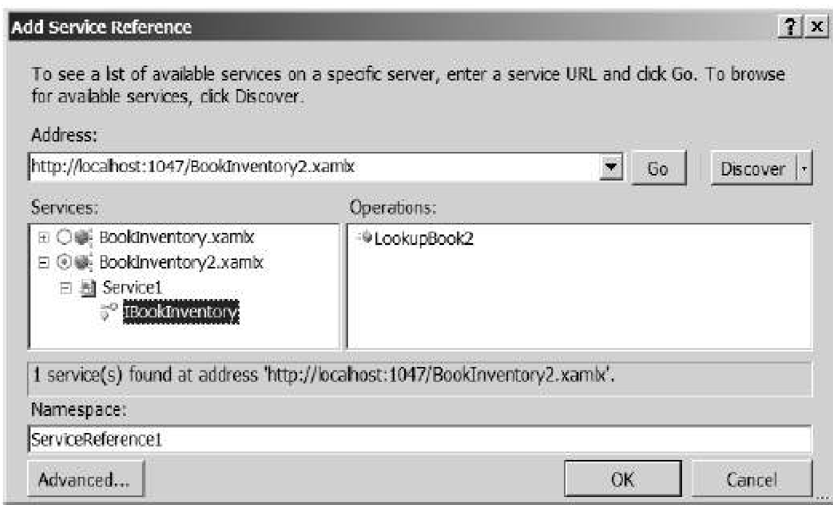
ნახ.21.20. console application მუშა პროცესის დამატება

Solution Explorer-ში BookLookup project-ზე მარჯვენა ღილაკის დაჭერით და Add Service Reference არჩევით, უნდა მივიღოთ 21.21 ნახაზი.



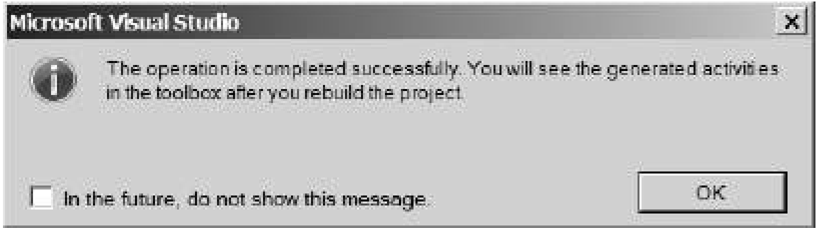
ნახ.21.21. არსებული სერვისების ძებნა

დავაჭიროთ Discover ლინკის ღილაკს და ავიჩიოთ სერვისები Solution-ში.21.22 ნახაზზე დიალოგის სიაში ჩანს ორი ჩვენ მიერ შექმნილი სერვისი BookInventory პროექტში.



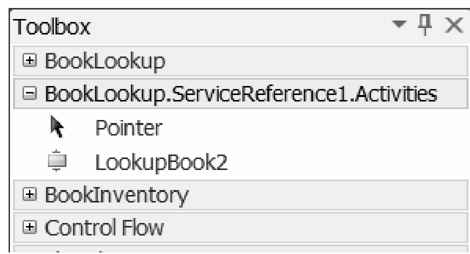
ნახ.21.22. სასურველი სერვისის მოძებნა

შესაძლებელია ამ სერვისების გაფართოვება, რათა დანახულ იქნას თითოეულში გათვალისწინებული მეთოდები. ავირჩიოთ მეორე, BookInventory2.xamlx და OK. რამდენიმე წამის შემდეგ უნდა გამოჩნდეს დიალოგური ფანჯარა (ნახ.21.23).



ნახ.21.23. დასრულებული დიალოგის პროცესი

ეს გაგვაგებინებს, რომ მიმართვა სერვისზე იქნა დამატებული პროექტში. F6-ით აღდგენილ იქნება solution. ფაილი Window1.xaml უნდა იქნას ასახული. თუ არა, მაშინ გავხსნათ იგი. ინსტრუმენტების პანელის ზედა ნაწილში უნდა იყოს 21.24 ნახაზის მაგვარი.



ნახ.21.24. განახლებული Toolbox სერვისების გარსით

ინსტრუმენტების პანელზე BookLookup.ServiceReference1 .Activities სახელსივრცე შეიცავს მომხმარებლის ქმედებას ყოველი მეთოდისათვის სერვისში. ჩვენ შემთხვევაში არის მხოლოდ ერთი: LookupBook2.

Solution Explorer-ში მაუსის მარჯვენა ღილაკით BookLookup პროექტზე დავაჭიროთ და ავირჩიოთ Set as Startup Project.

21.11. Worrkflow პროცესის განსაზღვრა

გადავიტანოთ LookupBook2 ქმედება სამუშაო პროცესზე. შემდეგ საჭიროა არგუმენტების დაყენება რათა შესაძლებელი იყოს საძებნი კრიტერიუმების გადაცემა სამუშაო პროცესში და შედეგების დაბრუნება. დააჭირეთ Arguments მართვის ელემენტს.

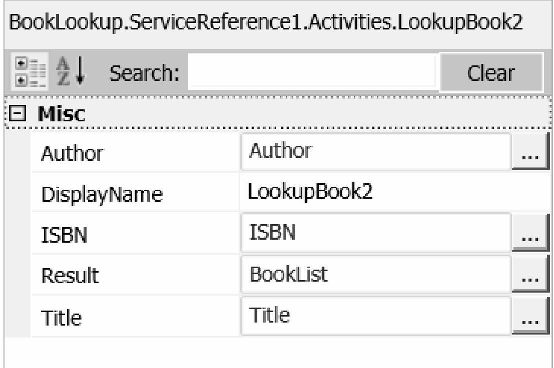
დავამატოთ სამი String შემავალი არგუმენტი სახელით Title, Author და ISBN. დავამატოთ გამომავალი არგუმენტი, სახელით BookList. არგუმენტის ტიპისთვის ავირჩიოთ Array Of[T]. გამოსულ დიალოგურ ფანჯარაში ვირჩევთ Browse for Types და ავირჩევთ BookInfo კლასს BookLookup.ServiceReference1.BookInventory ნაკრებიდან. არგუმენტების სია მოცემულია 21.25 ნახაზზე.

Name	Direction	Argument type	Default value
Title	In	String	Enter a VB expression
Author	In	String	Enter a VB expression
ISBN	In	String	Enter a VB expression
BookList	Out	BookInfo[]	Default value not supported
Create Argument			

Variables Arguments Imports 100%

ნახ.21.25. სამუშაო პროცესის არგუმენტები

ავირჩიოთ "LookupBook2" ქმედება; Properties ფანჯარაში შევიტანოთ თვისებების values , როგორც 21.26 ნახაზზეა ნაჩვენები.



ნახ.21.26. LookupBook2 თვისებები

21.12. ჰოსტ აპლიკაციის რეალიზაცია

გავხსნათ Program.cs ფაილი LookupBook პროექტში. ამ ფაილის რეალიზაცია ნაჩვენებია 21.4 ლისტინგში.

```
//- ლისტინგი 21.3 ---   Program.cs ფაილი ------------
using System;
using System.Linq;
using System.Activities;
using System.Activities.Statements;
using System.Collections.Generic;
using BookLookup.ServiceReference1;

namespace BookLookup
{
  class Program
   {
    static void Main(string[] args)
     {
      // create dictionary with input arguments
      // for the workflow
      IDictionary<string, object> input = new
                   Dictionary<string, object>
           {
               { "Author" , "Margaret Mitchell" },
               { "Title" , "Gone with the Wind" },
```

```
                { "ISBN" , "1234567890123" }
            };

          // execute the workflow
        IDictionary<string, object> output =
        WorkflowInvoker.Invoke(new Workflow1(), input);

        BookInfo[] l = output["BookList"] as BookInfo[];
           if (l != null)
            {
             foreach (BookInfo i in l)
              {
              Console.WriteLine("{0}: {1}, {2}", i.Title,
                        i.status, i.InventoryID);
                  }
            }
           else
             Console.WriteLine("No items were found");

             Console.WriteLine("Press ENTER to exit");
             Console.ReadLine();
         }
     }
}
```

ეს კოდი გადასცემს Autor-, Title- და ISBN-ში არგუმენტებს ობიექტის ლექსიკონის საშუალებით (იხ. თავი 4). სამუშაო პროცესი აბრუნებს უკან BookInfo ობიექტების მასივს. ამ კოდს გამოაქვს ეკრანზე ამ მასივის შინაარსი.

21.13. აპლიკაციის ამუშავება

F5-ით გაიშვება პროგრამა. შედეგებს უნდა ჰქონდეს ასეთი სახე:

= = = = Using PickUsing Pick

Gone with the Wind: Available, 58ab51cd-2796-4b32-a7be-21170f1e922b

Gone with the Wind: CheckedOut, 64406a94-a6ef-45a7-8373-066f5f991134

Gone with the Wind: Missing, a37186ec-faa7-4e6b-8226-484f17075998

Gone with the Wind: Available, e34d39e5-aafa-4fd3-8000-664809b7e98d

Press ENTER to exit

21.14. Pick-ის გამოყენება (არჩევა)

WF 4.0/5 უზრუნველყოფს ქმედებას, სახელით Pick, რომელსაც აქვს რამდენიმე შტო (PickBranch). ყოველი შტო შეიცავს ტრიგერის თვისებას და აქციის თვისებას. თითოეული მათგანი ასრულებს ქმედებას (ან ქმედებათა მიმდევრობას). როდესაც Pick ქმედება შესრულდა, ტრიგერის ყველა ქმედება დაწყებულია. როგორც კი ამათგან ერთი ქმედება დასრულდება, მისი შესაბამისი აქციები შესრულებულია და ყველა სხვა შტო გაუქმებულია.

ეს სასარგებლოა შესაბამისი აქციების განსაზღვრისათვის რომელიმე მოვლენის საფუძველზე. მაგალითად, შეიძლება მონაცემთა მიღების (Receive) ქმედების გამოყენება ტრიგერისათვის.

ყოველ შტოს შეიძლება ჰქონდეს Receive ქმედება, რომელიც ელოდება სხვა შეტყობინებას. რომლის საფუძველზეც მიღებულია შეტყობინება, იმის შესაბამისი აქცია შესრულდება.

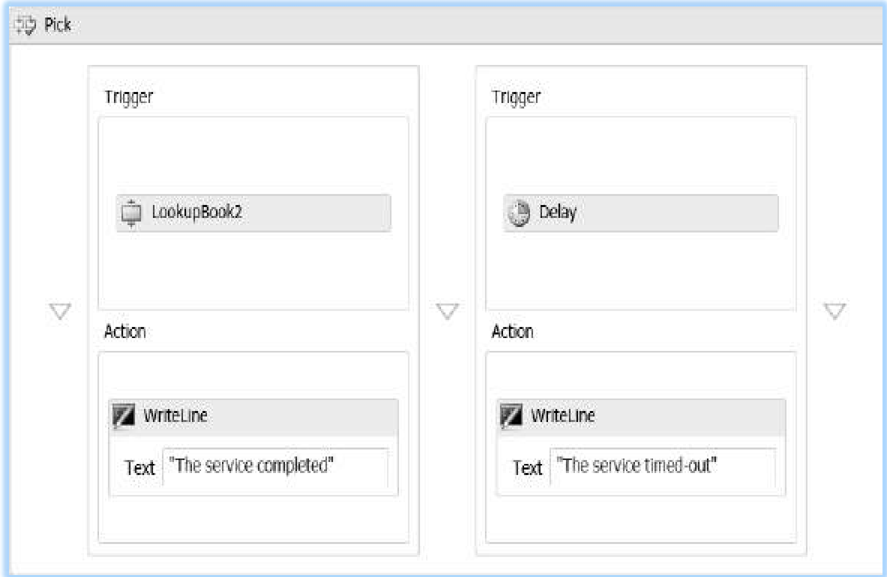
ამ პროექტისთვის გამოყენებულ იქნება Pick ქმედება, რათა უზრუნველყოფილ იქნას ტაიმ-აუტის ფუნქცია სამუშაო პროცესისათვის.

რჩევა. თუ თქვენ იცნობთ სამუშაო პროცესის წინა ვერსიებს, ეს ეკვივალენტურია, მაგალითად, Listen ქმედებისა.

გავხსნათ Window1.xaml ფაილი, მაუსის მარჯვენა ღილაკით „LookupBook2" ქმედებაზე დავაჭიროთ და ავირჩიოთ Cut (ამოჭრა). გადავიტანოთ Pick ქმედება სამუშაო პროცესზე. მაუსის მარჯვენა ღილაკით დავაჭიროთ პირველი განყოფილების Trigger-ს და ავირჩიოთ ბრძანება Paste (ჩასმა).

გადავიტანოთ დაყოვნების ქმედება (Delay) მეორე შტოს Triggeg-ის განყოფილებაში. დააყენეთ მისი ხანგრძლივობის (Duration) თვისება TimeSpan.FromSeconds(5).

გადავიტანოთ WriteLine ქმედება ყოველი აქციის სექციაში და მივცეთ Text-თვისება „სრული სერვისი“ და „სერვიი ტაიმ-აუტით“. მუშა პროცესი უნდა გამოიყურებოდეს 21.27 ნახაზზე მოცემული სახით.



ნახ.21.27. სამუშაო პროცესი timeout ლოგიკით

F5-ით ავამუშავოთ აპლიკაცია. შედეგები უნდა იყოს იდენტური პროგრამის ბოლო გაშვებისა გამორიცხვის დამატებული შეტყობინებით („სრული სერვისი"). ტაიმ-აუტის ფუნქციის შესამოწმებლად გავხსნათ BookInventory2.xamlx ფაილი. გადმოვიტანოთ დაყოვნების Delay ქმედება SendResponse ქმედების დაწყების წინ და დავაყენოთ მისი ხანგრძლივობის Duration თვისება TimeSpan.FromSeconds(7). F5-ით ავამუშავოთ აპლიკაცია. 5 წამის დაყოვნების შემდეგ შედეგებს ექნება ასეთი სახე:

The service timed-out
No items were found
Press ENTER to exit

რეზიუმე

სამუშაო პროცესები ხშირად განაწილებულია სხვადასხვა აპლიკაციაში და სხვადასხვა სერვერზეც კი. ამიტომაც კომუნიკაცია არის მნიშვნელოვანი ნაწილი სამუშაო პროცესის დიზაინისა. პროექტში, მაგალითად, ბიბლიოთეკის სხვადასხვა ფილიალი ურთიერთქმედებს ერთმანეთთან, რათა მოითხოვონ ელემენტები, რომლებიც ექვემდებარება გადაცემას. Send და Receive ქმედებები (და მათი „კოლეგები" ReceiveReply და SendReply) უზრუნველყოფს შეტყობინებათა მოსახერხებელი ფორმით გადაცემას და მიღებას. ეს ქმედებები ეყრდნობა WCF-ს შეტყობინებების გადასაცემად და შეუძლია გამოიყენოს რიგი პროტოკოლებისა, როგორიცაა HTTP ან TCP. მიუხედავად ამისა,

ჰოსტაპლიკაციას შეუძლია ასევე მიიღოს WCF-შეტყობინება უშუალოდ, როგორც ეს წინა თავში იყო ნაჩვენები.

მიუხედავად იმისა, რომ მუშა პროცესის ქმედებებს არ აქვს მომხმარებლის ინტერფეისი, ისინი ხშირად საჭიროებენ ურთიერთობას ჰოსტაპლიკაციასთან, ან აპლიკაციის განახლებას, ან მოითხოვოს მონაცემები, რომლებიც შეიტანება მომხმარებელთა მიერ. ამ თავში ვიყენებდით სანიშნეს (Bookmark), რათა შეჩერებულიყო სამუშაო პროცესი მომხმარებელთა მონაცემების სეტანამდე. აპლიკაციას ადვილად შეუძლია სამუშაო პროცესის განახლება, სადაც იგი შეწყდა. გამოიყენებოდა ასევე WriteLine ქმედება, კონსოლზე ტექსტის გამოსატანად (თავი 1). მე-2 თავში ნაჩვენები იყო ამ ქმედების გამოყენება აპლიკაციის სიაში (listbox).

Web-სერვისების გამოყენება ხდება დაპროექტებისათვის სულ უფრო პოპულარული მიდგომა. საქმე შეიძლება დაწყებულ იქნას კონტრაქტიდან მომსახურების მიღებაზე, ან უბრალოდ განისაზღვროს შემავალი და გამომავალი პარამეტრები, სამუშაო პროცესების კონსტრუქტორის გამოყენებით. სამუშაო პროცესი შეიძლება გამოყენებულ იქნას როგორც სერვისის შესაქმნელად და მათ გამოსაყენებლად. მეთოდები, რომლებიც უზრუნველყოფილია Web-სერვისებით, ხდება ტრადიციული, სტანდარტული ქმედებები, რომელთა გამოყენება შესაძლებელია სამუშაო პროცესებში.

VII ნაწილი

მონაცემთა მენეჯმენტი

XXII თავი. კორპორაციათა მართვის სისტემების მონაცემთა საცავები 668

XXIII თავი. მონაცემთა ბაზების მართვის სისტემები ………………………………….. 747

XXIV თავი. სივრცითი მონაცემთა ტიპები SQL Serever-ში ……………………..……. 758

XXII თავი
კორპორაციათა მართვის სისტემების
მონაცემთა საცავები

კომპიუტერული და ქსელური ინდუსტრიის განვითარებამ ბოლო ათწლეულებში მნიშვნელოვან შედეგებს მიაღწია, რამაც თითქმის მთლიანად შეცვალა მართვის საინფორმაციო სისტემების აგების ტექნოლოგია და ინსტრუმენტული საშუალებანი, გაჩნდა ახალი ცნებები და ტერმინები, რომლებიც საფუძვლად თანამედროვე კონცეფციებსა და პროექტებს ედება [35,38].

ორგანიზაციული სისტემების მართვის პრობლემებისა და ამოცანების გადასაწყვეტად საჭირო ხდება ახალი კომპიუტერული და ინფორმაციული ტექნოლოგიების გამოყენება, რომელთა შორის ერთ-ერთი აქტუალური და მნიშვნელოვანი მიმართულებაა მონაცემთა საცავების (data warehouse) აგება. ბოლო პერიოდში იგი ითვლება, განსაკუთრებით პერსპექტიულ და დინამიკურ მიმართულებად მართვის საინფორმაცო სისტემების დაპროექტებაში, იგი უკავშირდება მრავალდონიან განაწილებულ საინფორმაციო სისტემის შექმნას.

მონაცემთა საცავი განიხილება როგორც რომელიმე კორპორაციის, კონკრეტული ორგანიზაციის ან დიდი საწარმოსთვის განკუთვნილი სპეციალური სუპერ ბაზა, სადაც მიმდინარე ოპერატიული სამუშაოს შესრულებისას თავს იყრის ქრონოლოგიურ ინფორმაციათა მთელი სპექტრი, რომელთა დანიშნულებაცაა მომხმარებლისთვის ინტერნეტ გვერდებზე მიზნობრივად განლაგებული ტექსტური, გრაფიკული და აუდიო ვიზუალური საინფორმაციო ბლოკების მიწოდება.

მონაცემთა მენეჯმენტის ამოცანები განიხილება ორგანიზაციული მართვის საინფორმაციო სისტემის მონაცემთა საცავის, მონაცემთა ბაზების და თვით მონაცემების მოდელირების, შენახვის, დამუშავების და გადაცემის, ასევე ინფორმაციის დაცვის საკითხების საფუძველზე.

22.1. მონაცემთა მოდელები მულტიმედიური სისტემებისათვის

მონაცემთა მულტიმედიური ბაზების მართვის სისტემების (MMDBMS - MultiMedia DataBase Management Systems) ძირითადი ცნებების განვითარება უნდა დაიწყოს მონაცემთა მოდელების დისკუსიით. მონაცემთა მოდელი არის ერთგვარი სახე (ფორმალური) ენისა, რომელშიც მომხმარებელს შეუძლია სინამდვილის ფრაგმენტების აღწერა. მონაცემთა მოდელს ასევე მიეკუთვნება ოპერაციები (მეთოდები) მონაცემთა: წარმოების, წაკითხვის, მიერთების (ლინკის), მოწესრიგების (დალაგების), ცვლილების, ძებნის და წაშლის შესახებ. მონაცემთა მოდელი განსაზღვრავს მომხმარებლის

თვალსაზრისს მონაცემებზე. აქ მისი რეალიზაცია დაფარულია და მონაცემთა ბაზების მართვის სისტემის მიერ შესაძლებელია მისი განხორციელება სხვადასხვა გზით.

დღეისათვის, ტექნიკისა და ტექნოლოგიების განვითარების მოცემულ დონეზე, პრაქტიკულად ფართოდ გამოიყენება ე. კოდის რელაციური მოდელი, იერარქიულ და ქსელურ კლასიკურ მოდელებთან შედარებით [66,67].

ამ პერიოდში შემოთავაზებულ იქნა აგრეთვე სხვა, ახალი სახის მოდელებიც, რომლებსაც ექსპერიმენტულადაც იკვლევდნენ მწარმოებლურობის ამაღლების თვალსაზრისით, რელაციურთან შედარებით. ასეთი მოდელების მაგალითებია სემანტიკური, ფუნქციონალური და ობიექტ ორიენტირებული [68,73]. მაგრამ მათი რეალიზაცია დაკავშირებული იყო დიდ დანახარჯებთან, რაც რელაციური მოდელის უპირატესობას ამტკიცებდა, მისი სიმარტივის გამო. ამ თვალსაზრისით უკეთესი კონცეფცია გახდა ისევ რელაციური მოდელის გაფართოებით მიღებული შედეგები, როგორიც იყო „ობიექტრელაციური“ მონაცემთა ბაზის მართვის სისტემები (ორ-მბმს) [32]. ისინი, თავიანთ შესაძლებლობათა გამო, მისაღები გახდა ასევე მულტიმედიური სისტემებისთვისაც. ამგვარად, მნიშვნელოვანია გამოკვლეულ იქნას მედიამონაცემთა შესაბამისი მონაცემთა ტიპების ისეთი განსაზღვრება, რომელიც ხელს შეუწყობს შემდგომში მათ ჩაშენებას განსხვავებულ მონაცემთა მოდელებში ან, სხვა სიტყვებით რომ ვთქვათ, მოითხოვება ამ მონაცემთა ტიპების განსაზღვრება მონაცემთა მოდელებისაგან დამოუკიდებლად. ჩვენ ქვემოთ განვიხილავთ ამ საკითხს ობიექტრელაციური და ობიექტორიენტირებული ბაზებისთვის.

ტრადიციული მონაცემთა მოდელების შემოთავაზება ფორმატირებული მონაცემებისა და მათი კავშირების სამართავად ასევე გამოყენებადია მულტიმედიური მბმს-თვისაც. როგორც ცნობილია, მედიაობიექტები (ტექსტი, გრაფიკა, სურათი, აუდიო და ვიდეო ფაილები) გვხდება ყოველთვის მათ აღწერასთან ერთად. ამასთანავე ისინი დაკავშირებულია ერთმანეთთან და სხვა ნებისმიერ ფორმატირებულ მონაცემებთან. ეს შეიძლება შემდეგი ფორმებით გამოვლინდეს:

- ატრიბუტები (არსების ან ობიექტების). ობიექტი (პერსონა, მანქანა და ა.შ.) შეიძლება დამატებით აღიწეროს მისი სურათის, ხელმოწერის, ხმის საშუალებით. მედიაობიექტი აღადგენს გამოსახული ობიექტის თვისებას;
- კომპონენტები კომპლექსური ობიექტებისათვის. დოკუმენტი შედგება ცვლადი რაოდენობის მედიაობიექტისაგან: ტექსტის ბლოკების, სურათების, გრაფიკის და შესაძლო აკუსტიკური კომენტარებისაგან. ამჯერად მედიაობიექტები არის არა მხოლოდ ატრიბუტები, არამედ თვით ობიექტებიც. ამათ და დოკუმენტის ობიექტების შორის არსებობს რელაციური კავშირები, აგრეგატული დამოკიდებულების მსგავსად. შეიძლება განვიხილოთ ასევე სინქრონიზაციის დამოკიდებულება ვიდეოფილმის სურათებსა და ხმის ბილიკს შორის;

• ერთი და იმავე ინფორმაციის ალტერნატიული წარმოდგენა. ხშირად ერთიდაიგივე ინფორმაცია აისახება სხვადასხვა მედიის საშუალებით, მაგალითად, ცხრილებისა და გრაფიკების სახით. არაა ყოველთვის შესაძლებელი ერთი სახის ინფორმაციის მიღება (გარდაქმნით) სხვა სახიდან გონივრული ხარჯებით. ამიტომ უფრო მისაღებია ორივე, როგორც მედიაობიექტები, იქნას ცხადად შენახული.

ამ მედიაობიექტებს შორის ეკვივალენტობის დამოკიდებულების ისეთი სახე უნდა გამოიყენებოდეს, რომელიც გამომავალი მოწყობილობის წვდომის საშუალებითა და მომხმარებლის მიდრეკილებით მბმს-თან, შეძლებს ერთი ან მეორე სახის არჩევას.

მონაცემთა მოდელი უნდა ცნობდეს დამოკიდებულებათა ამ განსხვავებულ ტიპებს მედიაობიექტებსა და ფორმატირებულ მონაცემებს შორის, ან თვით მედიაობიექტებს შორის, რაც შემდეგ პროგრამის შესრულებისას ოპერაციებმა უნდა გაითვალისწინოს.

22.1.1. მონაცემთა ახალი ტიპები

ძირითადი ამოცანა მდომარეობს მონაცემთა ტიპების (ან კლასების) განსაზღვრაში მედიამონაცემებისთვის [32]. შემოთავაზებული ოპერაციების სიმრავლე, განსაკუთრებული ყურადღებით უნდა შეირჩეს ტექსტისათვის, რასტრული სურათისა და გრაფიკისათვის და ა.შ.

მონაცემთა ტიპი Text. ახალი საბაზო მონაცემთა ტიპის განსაზღვრა განვიხილოთ Text ტიპის მაგალითზე.

ცხადია, რომ მონაცემთა ტიპი Text უნდა იყოს მეტი, ვიდრე char[]. ამიტომაც Text ტიპის ობიექტს საკმაო რაოდენობის ოპერაციები მიეწოდება, რომლებიც იღებენ განსხვავებულ ინფორმაციას ამ ობიექტიდან:

```
interface Text {
    public int length ();
    public int alphabet ();  // 0 == ISO Latin-1, . . .
    public int alphabetSize ();
    public int language (); // 0 == English, 1 == German, …
    public char charAt (int n);
    public byte[] getASCII ();
    public byte getEBCDIC ();
    public String getUnicode ();
    . . .
    };
```

Text -ს აქვს ყოველთვის ერთი სიგრძე (გამოყენებული სიმბოლოების რაოდენობა). ტექსტი ხშირ შემთხვევაში განიხილება როგორც სტრიქონსტრუქტურირებული,

რომელშიც კიდევ უკვე განსაზღვრულია ქვეტიპი. სტრიქონებად დაყოფა შეიძლება ინფორმაციის შემცველი იყოს როგორც ლექსებში. დაყოფა სიტყვებად დამოკიდებულია ენაზე და სიმბოლოების ერთობლიობაზე, მაგრამ, პრინციპულად, ყოველ ტექსტში შესაძლებელია. ამ სტრუქტურებიდან გამომდინარე, შეიძლება შემდეგი ოპერაციების განხილვა, რომლებიც სისტემისა და ინსტრუმენტისთვისაც ცნობილია:

```
// გამოაქვს შედეგად სტრიქონების რაოდენობა ტექსტში
public int lineCount ();

// გამოაქვს შედეგად სიტყვების რაოდენობა ტექსტში
public int wordCount ();

// გამოაქვს ტექსტიდან სტრიქონები მითითებული ნომრით
public char [ ] line (int lineNo);

// გამოაქვს ტექსტიდან სიტყვა მითითებული ნომრით
public char [ ] word (int wordNo);
```

ცხადია, რომ Text ტიპის ობიექტისთვის მრავალი ოპერაციის შეთავაზებაა შესაძლებელი, რომლებიც ტექსტის მთლიანი თუ ცალკეული ნაწილების დასამუშავებლად იქნება გამოყენებული. მაგალითად,

```
public boolean print (Printer p);
public boolean display (window w);
```

ეს ოპერატორები შედეგად იძლევა მხოლოდ დასტურს, რომ მონაცემთა გამოტანის პროცესი დასრულდა წარმატებით.

განვიხილოთ ცვლილების ოპერაციების მაგალითები:

```
public void replaceLine (int lineNo, char [] newLine);
public void insertLine (int lineNo, char [] newLine);
public void concatenate (Text t);
```

Text ტიპის ობიექტის საწარმოებლად საჭიროა ოპერაციის მომზადება, რომელიც სხვა ტიპის ობიექტებიდან შეძლებს Text ტიპის ობიექტში ასახვას. მაგალითად, java ენაზე შეიძლება დაიწეროს კლასის შემდეგი კოდი კონსტრუქტორით:

```
class TextClass implements Text {
   int length,
```

```
        int charLength,
        int code,         // 0 == ASCII, 1 == EBCDIC, . . .
        int formatter,    // 0 == none, 1 == PostScript, . . .
        byte endOfLine,
        byte [] characters
    };
    . . .
}
```

ასეთი ოპერაციით მოითხოვება შესაძლოდ ფართო გამოყენება, იგი საშუალებას იძლევა ტექსტის მრავალი სახე სხვადასხვა სისტემური გარემოდან გადასცეს (მონაცემთა ბაზის) ობიექტს, Text ტიპისას.

აქ განხილულ ოპერაციებს Text ტიპისათვის ჰქონდა მხოლოდ საილუსტრაციო ხასიათი. ისინი გვიჩვენებს, რომ საჭიროა მსგავსი ოპერაციების მომზადება მონაცემთა ასეთი ტიპის გამოყენებისას. მხოლოდ ასეთი ტექსტობიექტებით იქნება შესაძლებელი მულტიმედიურ მონაცემთა ბაზებში მუშაობა. საჭიროა სხვა ოპერაციების შემუშავებაც, რომლებიც განსაზღვრულ სამუშაო გარემოსთვის იქნება მორგებული. ეს შესაძლებელია უშუალოდ ტიპის გაფართოებით მოხდეს ან მისი სპეციალიზაციით.

მონაცემთა ტიპი Image აღიწერება ხშირად გამოყენებადი ოპერაციების საშუალებით. მისი შინაგანი სტრუქტურის შესახებ მომხმარებელთა უმრავლესობას ზოგადი წარმოდგენა აქვს, რომ იგი შედგება ფერთა ცხრილისა და პიქსელების მატრიცისაგან. არსებული Image-ობიექტების წასაკითხად საჭიროა შესაბამისი ოპერაციების მომზადება, რომლებიც შეძლებს ცალკეულ კომპონენტზე მიმართვას:

```
interface Image {
  public int height ();
  public int width ();
  . . .
 }
```

დანარჩენ ოპერაციებს შეუძლია სურათების სტატისტიკური შეფასება. მაგალითად,

```
public int pixelcount (byte [] pixelvalue);
```

ითვლის სიხშირეს, რომლითაც განსაზღვრული პიქსელის მნიშვნელობა ხდება.

სურათის წაკითხვისას მონაცემთა ბაზიდან შესაძლებელია მისი შემადგენელი ელემენტარული ნაწილების (პიქსელების) გამოძახება, ასევე შესაძლებელია გამოსახულების დართვა გამოთვლითი გარემოს სპეციფიურ მონაცემთა სტრუქტურაზე. მაგალითად, SUN-კომპიუტერებზე რასტრული გამოსახულება სათანადო დროის მანძილზე იმართება მონაცემთა სტრუქტურაზე, სახელით Pixrect, რომელიც შეიცავს მრავალ ზემოაღნიშნულ ნაწილს. ეს შეიძლება ასევე პირდაპირ იქნას წარმოებული:

public Pixrect getPixrect ();

ინტერაქტიულ გარემოში სურათის ნახვა ყველაზე გავრცელებული გამოყენების სახეა, ამიტომ, ყოველი შემთხვევისათვის იგი უნდა იყოს მხარდაჭერილი საკუთარი ოპერაციით:

public boolean display (Device);

სურათის ცვლილება შესაძლებელია და მისი განხორციელებისას ყურადღება უნდა გამახვილდეს მთლიანობის შენარჩუნებაზე: მხოლოდ პირველადი მონაცემების ცვლილება შესაბამისი რეგისტრაციის მონაცემების ადაპტაციის გარეშე, ან პირიქით პროცესის დროს, უნდა იქნას თავიდან აცილებული. შემდეგი ოპერაციები ასახავს განახლების მაგალითს, ასეთი მრავალი შეიძლება იყოს.

public Image window (int x0, int y0, int x1, int y1);

ჩამოიჭრება ფანჯრის გარეთ მდებარე სურათის ნაწილი,

```
public Image replaceColormap {
   Code encoding;
int colormapLength;
int colormapDepth,
int [] [] colormap
};
```

სურათის ფერების ცხრილის შეცვლა შემდეგი სახით;

```
public Image replacePixelvalue {
   int x, int y,
   byte [] pixelvalue
};
```

იცვლება ცალკეული პიქსელის მნიშვნელობა.

როგორც ტექსტური ტიპისათვის იქნა ახსნილი, აქაც ეს ოპერაციები შეიძლება იყოს რეალიზებული პროცედურების სახით (ანუ შედეგის ტიპით void), თუ ეს სათანადო გადაწყვეტად იქნება მიჩნეული.

Image ტიპის მონაცემთა ობიექტის საწარმოებლად შემოთავაზებული ოპერაცია

```
class ImageClass implements Image {
    public ImageClas {
        int height,
        int width,
        int depth,
        float aspectRatio,
        Code encoding,
        int colormapLength,
        int colormapDepth,
```

```
        int [] [] colormap,
        byte [] pixelmatrix
    };
. . .}
```

მონაცემთა ტიპები Graphic, Sound, Video ანალოგიურად განისაზღვრება. აქ მათი განხილვა არაა წარმოდგენილი, ვინაიდან ახალ ასპექტებს არ შეიცავს.

მულტიმედიური მონაცემთა ტიპების გაფართოება მათი რეზიუმირებისათვის (შინაარსის მოკლე მიმოხილვისათვის) ხდება იმავე პრინციპით, როგორც ეს იყო ოპერაციების განსაზღვრის დროს, განსაკუთრებით შედარების ოპერაციებისათვის.

რადგან რეზიუმეების დროს საქმე ეხება დამოკიდებულ კომპონენტებს, რომლებიც მულტიმედიური ობიექტების გარეშე არ არსებობს, ისინი კაპსულირებული იქნება ობიექტთან. ეს მოითხოვს დამატებით ოპერაციებს მულტიმედიური ობიექტებისათვის. ქვემოთ ნაჩვენებია მაგალითი Image ტიპისათვის, რომლის მსგავსი იქნება ასევე სხვა მულტიმედია ტიპებისთვისაც. გამოსახულება გვიჩვენებს პროცედურებს.

```
interface Image {
    . . .
    public void newDescr (String descr);
    }
```

შეაქვს სურათის რეზიუმე descr-ში და ცვლის აქ მონაცემებს.

```
public void extendDescr (String descr);
```

აფართოვებს ძველ რეზიუმეს (თუ არსებობს) descr -ში მოცემულს.

```
public int descrLength ( );
```

იძლევა რეზიუმის სიგრძეს.

```
public String getDescr ( );
```

გამოაქვს რეზიუმე სტრიქონის ფორმატში.

```
public boolean contains (String query);
```

იკვლევს, რეზიუმე ხომ არ შეიცავს query-ს, ანუ ძებნის გამოსახულებას, რომელიც query-ს შეესაბამება. ესაა წარმომადგენელი შედარების ოპერაციებიდან, რომლებიც სურათებისთვის და სხვა მულტიმედიატიპებისათვის შეიძლება იყოს განსაზღვრული.

22.1.2. კლასთა იერარქიის აგება განზოგადების საფუძველზე

მულტიმედიური მონაცემთა ტიპების რეზიუმეების ოპერაციების განზოგადება შესაძლებელია გენერალიზაციის იერარქიის აგებით, რომელშიც „მშობელი" მონაცემთა ტიპი (სუპერკლასი) MediaObject არის მოთავსებული.

მისთვის განსაზღვრულია ოპერაციები, რომლებიც ყველა მედიაობიექტისთვისაა გამოყენებადი. შეიძლებოდა ასევე, მაგალითად, ერთი (ვირტუალური ან აბსტრაქტული) მეთოდის output აგება, რომლის კონკრეტული რეალიზაცია იქნებოდა თითოეული არსებული მულტიმედიური მონაცემთა ტიპი.

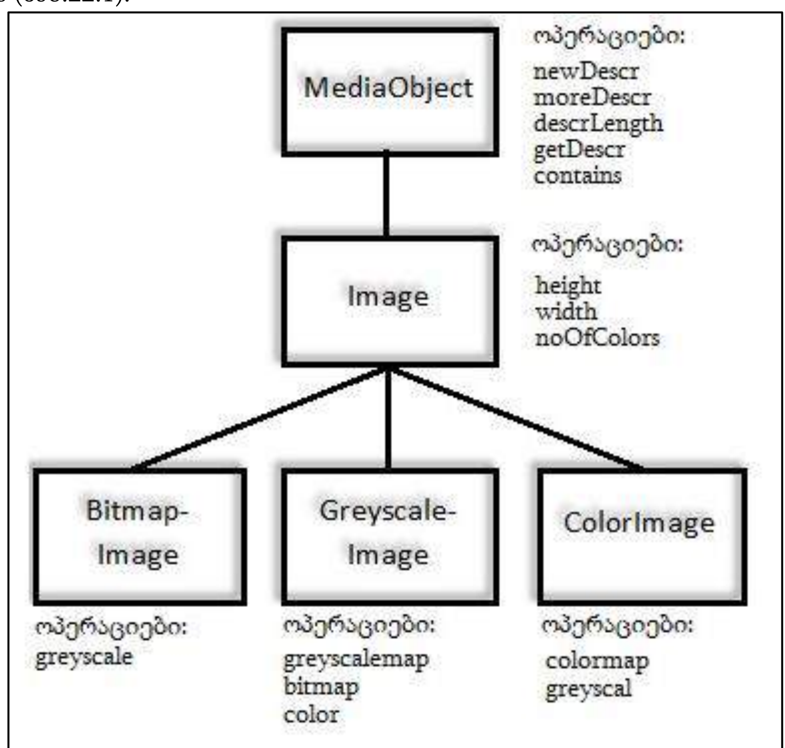
განზოგადების იერარქიის შემდგომი გამოყენება შესაძლებელია, რათა სუბკლასების დახმარებით მოხერხდეს ერთგვარი სიბინძურის თავიდან აცილება, რომელსაც აქამდე Image -სთვის განსაზღვრული ოპერაციები მიუთითებდნენ.

მაგალითად, არსებობს სურათები Pixel-ის სიღმით 1, რომლებიც არ იყენებენ, ფერთა ცხრილებს. Pixel მნიშვნელობით 0 , ნიშნავს თეთრს, ხოლო 1 კი - შავს. პრინტერებისა და მონიორიების დიდი კლასისათვის (Bitmap-დისპლეები) ასეთი სურათები მეტად მნიშვნელოვანია.

მათთვის ისეთი ოპერაციების განსაზღვრა, როგორიცაა colormapLength და getColormap არის უაზრო.

ამავე დროს არსებობს სხვა ოპერაციები, რომელთა შემოტანა ასეტი ტიპის Bitmap-სურათებისათვის არის სასარგებლო [GS83].

ამგვარად, შემოთავაზებულია სამი ტიპის სურათებისათვის ცალკე სუბკლასების წარმოდგენა (ნახ.22.1).



ნახ.22.1. კლასთა იერარქია რასტრული სურათებისათვის

Image-ს სუბკლასები ასახავს განსხვავებულ ინფორმაციულ შინაარსს რასტრული გამოსახულებებისათვის. თუ წარმოვიდგენთ ერთ ფოტოსურათს როგორც Bitmap-სურათს, მერე როგორც შავ-თეთრს და, ბოლოს როგორც ფერადს, ინტუიციურად მივხვდებით, რომ ფერადი სურათი მეტ ინფორმაციას შეიცავს, ხოლო Bitmap-სურათი-ყველაზე ნაკლებს.

ფერადი სურათის გარდაქმნას შავ-თეთრში ანგარიშობენ ფერის სიკაშკაშის მნიშვნელობით ან წაკითხულ იქნება სპეციალური ცხრილიდან, და ის დაკავშირებულია ყოველთვის ინფორმაციის დაკარგვასთან. იგივეა შავ-თეთრი სურათის გადაყვანისას Bitmap-სურათში. ასეთი ინფორმაციული დანაკარგები მხედველობაში უნდა იქნას მიღებული, როცა მათი წარმოდგენა იქნება საჭირო სრულიად განსაკუთრებულ გამომტან მოწყობილობებზე. ამისათვის არის გათალისწინებული ოპერაციები greyscale ColorImage-სთვის და bitmap GreyscaleImage--თვის.

ყოველი რასტრული სურათი შეიძლება იყოს Bitmap-სურათი, შავ-თეთრი ან ფერადი. იგი მიეკუთვნება ყოველთვის მხოლოდ ერთ სუბკლასს. ამიტომაც თითოეული ფლობს თავის განსაკუთრებულ კონსტრუქტორებს. აქ მიღებული რასტრული სურათების სპეციალიზაციები ამიტომ არის გადაუკვეთადი და სრული. მონაცემთა ბაზების სისტემას უნდა შეეძლოს ასეთი მთლიანობის პირობის დაცვის გარანტირება.

22.1.3. SQL/MM: მულტიმედიური მონაცემები და სტანდარტები

მულტიმედიურ მონაცემთა ტიპების დაპროექტების პროცესის უნიფიკაციის საკითხებმა ასახვა პოვა ჯერ SQL:1999 გაფართოებაში, შემდეგ კი ერთიან საერთაშორისო ISO/IEC 13249 SQL/MM სტანდარტში [65]. სტანდარტი ხუთ პაკეტად დაიყო: ფრეიმვორკი (ან ჩარჩო, სტრუქტურა), სრული ტექსტი, ვექტორული გრაფიკა, რასტრული სურათები და მონაცემთა ინტელექტუალური ანალიზი (Data Mining).

ამასთანავე, SQL:1999 სინტაქსის ტერმინების გამოყენების თვალსაზრისით, კლასების ცნებისათვის გამოყენებულ იქნება „მომხამარებელის მეირ განსაზღვრული ტიპები" (UDT – User-Defined Types) ხოლო ოპერაციებისათვის - „მომხმარებლის მიერ განსაზღვრული ფუნქციები" (UDF – User-defined Functions).

- **SQL/MM სრული ტექსტი**

ეს პაკეტი განსაზღვრავს UDT FullText-ს ტექსტური მონაცემებისათვის და სხვა UDT FT_Pattern-ს საძებნი შაბლონებისათვის (ან მოდელებისთვის). FullText მიუთითებს ოთხ საძებნ მეთოდზე, რომელთაგან ორი განსხვავდება მხოლოდ მათი პარამეტრების ტიპებით. კერძოდ, ესაა ან სიმბოლოების ჯაჭვი (სტრიქონი) ან FT_Pattern-ის ტიპის შაბლონი. მეთოდების სახელები გადატვირთვადია (overloaded).

ორი Contains-მეთოდი ახორციელებს ლოგიკურ ძებნას. შედეგის სახით მიიღება მხოლოდ „დიახ“ ან „არა“. ხოლო ორი Rank-მეთოდი კი, მეორე მხრივ, განსაზღვრავს რიცხვს მცოცავი წერტილით, რომელიც გამოიყენება პოზიციონირებისათვის (ranking).

ამას გარდა FullText-ს აქვს ორი კონსტრუქტორი. ერთი აწარმოებს ტექსტის ობიექტს სტრიქონიდან, მეორე კი დამატებით არეგისტრირებს ტესქტის ენას. დასასრულ, არსებობს კიდევ ერთი ფუნქცია FullText_to_Character სტრიქონების მისაღებად, ტექსტის გამოტანის მიზნით.

SQL:1999 სინტაქსით UDT-განსაზღვრება სტანდარტულად შემდეგი სახით ჩაიწერება:

```
create type FullText as (
      Contents character varying (FT_MaxTextLength),
     Language character varying (FT_MaxLanguageLength),
. . .
)
method Contains (pattern FT_Pattern)
        returns integer
method Contains (pattern character varying (FT_MaxTextLength))
        returns integer
method Rank (pattern FT_Pattern)
        returns double precision
method Rank (pattern character varying (FT_MaxTextLength))
        returns double precision
method FullText (String character varying (FT_MaxTextLength))
        returns FullText
method FullText (
        String . . . ,
        Language character  varying (FT_MaxTextLength)
)
        returns FullText;
create cast (FullText as
       character varying (FT_MaxTextLength)
       with FullText_to_Character);
create type FT_Pattern as
       character varying (FT_MaxPatternLength);
```

FT_Pattern მნიშვნელობები უნდა იყოს გამოსახულებები, აღწერილი ბეკუს-ნაურის ფორმის ენის საშუალებით. შეფასება აღიწერება წესების საშუალებით ენის სიმბოლოების მიხედვით. განვიხილოთ მაგალითები ამ საკითხებზე.

ტექსტური ობიექტი აქ მოიცემა aText სახით და შინაარსით: „ეს სექცია წარმოადგენს SQL/MM სტანდარტს. ამ სტანდარტით განსაზღვრულია მულტიმედიური ობიექტების ტიპები და ქვეპროგრამები“.

უმარტივესი ძებნის შაბლონი არის ერთი სიტყვა:

aText.Contains(‘ “სექცია“ ‘) = 1

ეს გამოსახულება ასახავს შემთხვევას, როცა შედეგი „ჭეშმარიტია“ („true“ ანუ 1).

შესაძლებელია ძებნის განხორციელება სიტყვების სიმრავლითაც, სადაც გამოყენებულ იქნება სიტყვებსშორისი შემავსებლები (placeholder, wildcards), მაგალითად „ქვედა ტირე“.

aText.Contains(‘ “სექცია_“ ‘) = 0.

შედეგი მიღებულა ამჯერად „მცდარი“ (false ანუ 0) მნიშვნელობით.

ძებნის შაბლონებში შესაძლებელია აგრეთვე თეზაურუსის გამოყენებაც, მასზე მიმართვით. იგი გაფართოებული შაბლონით აფორმირებს ახალ სიტყვას, მაგალითად, მსგავსი სიტყვები, განზოგადებული სიტყვები, სპეციალური სიტყვები, სინონიმები ან წარმოებული:

```
aText.Contains ( ‘
        thesaurus “Informatik”
        expand synonym term of “Norm”
  ‘ ) = 1
```

აქ იგულისხმება, რომ „ინფორმატიკის“ თეზაურუსი სიტყვისათვის „Norm“ პოულობს სინონიმს „Standard“ (გერმანული ენა). სიტყვათა მიმდევრობა და დაცილება შესაძლებელია განისაზღვროს კონტექტური შაბლონის ფორმით:

```
aText.Contains ( ‘
        (“მოხსენება“)  near “სტანდარტი”
        within 0 sentences in order
  ‘ ) = 1
```

დასასრულ, შესაძლებელია ასევე ე.წ. კონცეფციური შაბლონის (concept pattern) განხილვა ტექსტის მნიშვნელობის მისამართად:

```
aText.Contains ( ‘
        is about “Internationaler Standart
        yur Volltextsuche”
  ‘ ) = 1
```

ამ დროს ღიადაა დატოვებული, თუ is about როგორ იქნება რეალიზებული.

ყველა ზემოგანხილული მაგალითი იყო ცალკეული ძებნის-ფრაზები. შესაძლებელია მათი კომბინაციური გამოყენებაც, კონიუნქციებისა და სხვა ლოგიკური ოპერაციების გაერთიანების საფუძველზე.

შეიძლება განვიხილოთ უმარტივესი შაბლონი, რომელიც მოთხოვნის მაგალითის კონტექსტში იქნება ნაჩვენები:

```
select * from myDocs
where Doc.Rank(‘ “ Standard” ‘) > 0.8
```

ამგვარად, სისტემა უნდა აკონტროლებდეს (შეფასების თავალსაზრისით), რომ მოხდა ინფორმაციის სწორი ამოცნობა, ანუ სიტყვის, წინადადების, აბზაცის მიხედვით (კონტექსტური შაბლონებით). გაფართოებული შაბლონებით კი მიიღწევა სწორი (ანალოგიური, მსგავსი) სიტყვების პოვნა.

- SQL/MM სივრცითი ტიპი (Spatial)

მონაცემთა სივრცითი ტიპი Spatial გამოიყენება “graphic” ტიპის ობიექტებისათვის [74, 117, 152].

განიხილება UDT-ები 2D-მონაცემებისა (წერტილი, წრფეები, სიბრტყეები) და მათი კოლექციებისათვის. მათ შეიცავს პროგრამები სივრცითი მონაცემების მანიპულაციის, ძებნისა და შედარებისათვის, აგრეთვე კონვერტირებისათვის UDT და სიმბოლოებს ან ორობით წარმოდგენებს შორის.

ცალკეული ტიპები ორგანიზებულია განზოგადებული იერარქიით. ფესვის ტიპს ეწოდება ST_Geometry. ყოველ გეომეტრიულ ობიექტს მიეკუთვნება ერთი SRID (Spatial reference system identifier), რომელიც ახასიათებს სივრცით ათვლის სისტემას. იგი ეფუძნება ცნობილ ათვლის სისტემებს: გეოგრაფიული საკოორდინატო სისტემა (განედი და გრძედი), საპროექციო საკოორდინატო სისტემა (X-ით, Y-ით და Z-ით). ისინი აღიწერება, მაგალითად ასეთი სახის BNF-ით:

```
<geographic system> ::= <projected cs> | <geocentric cs>
<geographic cs> ::= GEOGCS <left delimiter>
   <double quote> <name> <double quote> <comma>
   <datum> <comma>
   <prime meridian> <comma>
   <angular unit>
<right delimiter>
```

ST_Geometry ტიპის კოლექციის ელემენტებისათვის და ST_Geometry ველის შიგნით უნდა იქნას შერჩეული ერთი და იგივე ათვლის სისტემა.

დეტალურად რომ დავახასიათოთ, არსებობს შემდეგი ტიპები:

- „ნულგანზომილებიანი“ - განიხილება წერტილი: ST_Point;
- ერთგანზომილებიანი ობიექტები არის ზოგადად ST_Curve, რომელშიც შესაძლებელია ქვეტიპების შექმნა, რომლებიც განსაზღვრავს ინტერპოლაციას ცალკეულ წერტილებს შორის: ST_LineString – წრფივი ინტერპოლაციაა; ST_CircularString – წრიული ფორმისა და ST_CompoundString – შერეული ფორმისაა;
- ორგანზომილებიანი ობიექტები - ST_Surface, სადაც ST_CurvePolygon განისაზღვრება გარე და შიგა ST_CompoundString საზღვრებით, მაშინ, როცა ST_CurvePolygon იყენებს მხოლოდ ST_LineString საზღვრებს.

დამატებით არსებობს აგეთვე კოლექციის ობიექტები, რომლებიც აკეთებს იმას, რაც სხვა სისტემებისგან ცნობილია „დაჯგუფების“ (grouping) სახით. ისინი მოითხოვს, როგორც უკვე აღინიშნა, ერთნაირ ათვლის სისტემას ყველა ელემენტისათვის. არსებობს ST_MultiPoint, ST_MultiCurve, ST_MultiLineString, ST_MultiSurface და ST_MultiPolygon, რომელთა სახელები ცხადად მიუთითებს, თუ ელემენტის რომელი სახისთვის არის თითოეული დანიშნული.

ST_Geometry-სა და ქვეტიპებზე განსაზღვრულია შემდეგი მეთოდები: საშუალო (წერტილთა სიმრავლის საშუალო), სხვაობა და გაერთიანება, მანძილის გაანგარიშება, ტესტირება (contains, overlaps, touches, croses,…), და ათვლის სისტემის განსაზღვრა.

ამასათანავე ხდება აგრეთვე რამდენიმე სხვა მეთოდის დამატება ქვეტიპებისათვის, როგორიცაა, მაგალითად, length ST_Curve-თვის, area და perimeter ST_Surface-თვის.

- **SQL/MM სტილი სურათი/ფოტო (Styl Image)**

განვიხილოთ სურათის (ან ფოტოს) სტილის სტანდარტის გამოყენების საკითხი. თანამედროვე სტანდარტების შემოთავაზებით, მაგალითად, UDT-ს SI_StillImage გამოიყენება სურათის (ფოტოს) მონაცემებისათვის, SI_Feature - სურათების თვისებებისთვის და SI_FeatureList ასეთი თვისებების სიისათვის. SI_StillImage-ს შემთხვევაში, საინტერესოა, რომ ატრიბუტების სია, ანუ შინაგანი წარმოდგენა ღიაა და შესაძლებელია მასზე მიმართვა:

```
create type SI_StillImage  as (
    SI_content binary large object (SI_MaxContLength),
    SI_contentLength integer,
    SI_format character varying (8),
    SI_height integer,
    SI_width integer,
    . . .
)
```

ატრიბუტი SI_content შეიცავს სათაურს, ფერის ცხრილებს და სხვა მსგავს მონაცემებს, ანუ სარეგისტრაციო მონაცემებს და აღწერის მონაცემებსაც. „Container"-ის სახით განიხილება მთლიანი სურათი.

ატრიბუტი SI_format ის ფორმატია, რომლითაც სურათი ინახება კონტეინერში, მონაცემთა მასისვის სუფიქსის მსგავსად. განასხვავებენ „მხარდამჭერ" და „მომხმარებლის_მიერ_ განსაზღვრულ" ფორმატებს. მხარდამჭერი ფორმატები იკითხება თვით მბმს-ის მიერ, ისე, რომ ხდება BLOB-ის შინაარსის ინტერპრეტაცია. მაგალითად, მას შეუძლია სურათის თვისებების ამოღება. მომხმარებლის მიერ განსაზღვრული ფორმატები კი, თავის მხრივ, იმართება მბმს-ით მხოლოდ როგორც სახელები. ინტერპრეტაციის საკითხი რჩება აპლიკაციის გადასაწყვეტი.

ცხდად ჩანს, რომ მონაცემთა დამოუკიდებლობა აქ არ თამაშობს განსაკუთრებულ როლს. ანუ ის მოთხოვნები, რომლებიც ზემოთ იქნა განსაზღვრული სხვა ტიპებისათვის, ნაწილობრივ ხორციელდება. როგორც ჩანს, იგი უფრო ორიენტირებულია ფაილური სისტემების (მასივების) მოდელებზე, ვიდრე მონაცემთა ბაზებზე ტიპებით.

ეს საკითხი შემდგომ აისახება ასევე ოპერატორებზეც. SI_StillImage-ს აქვს ორი კონსტრუქტორი, რომელთაგან ერთი შესასვლელზე ელოდება უბრალოდ BLOB-ს, ხოლო მეორე, პირიქით, BLOB-ის გარდა სხვა ფორმატის მონაცემებს.

ამიტომ არსებობს ცვლილებათა ორი მეთოდი: პირველი ჩაანაცვლებს შიგთავსს (Content-ს) ახალი BLOB-ით, და შეიძლება იმედი ვიქონიოთ, რომ სხვა ატრიბუტებიც მლიანობას მოერგება. მეორე მეთოდი ცვლის ფორმატს და ამან უნდა მოახდინოს Content-ის (შიგთავსის) შინაარსის კონვერტირება.

არსებობს აგრეთვე წაკითხვის ორი მეთოდი მინიატურული სურათის (Thumbnails) საწარმოებლად:

```
method SI_StillImage (
    content binary large object (SI_MaxContLength)
) returns SI_StillImage

method SI_StillImage (
    content binary large object (SI_MaxContLength)
format character varying (. . .)
) returns SI_StillImage

method SI_setContent (
    content binary large object (SI_MaxContLength)
) returns SI_StillImage

method SI_changeFormat (
```

```
    targetFormat character varying (. . .)
) returns SI_StillImage
```

ფორმატის ცვლილების მეთოდები გამოიყენება მხოლოდ არსებული (მხარდაჭერილი) ფორმატებისთვის.

აღწერილი მონაცემებისათვის შესაძლებელია სურათის მახასიათებლების მიღება და შენახვა ცალკე სვეტებში სურათის გვერდით. SI_Feature საბაზო ტიპს აქვს შემდეგი ქვეტიპები:

- SI_AverageColor - ასახავს მთლიან სურათს ერთ ფერში, უბრალოდ საშუალო ფერით;
- SI_ColorHistogram - მართავს ფერთა ჯგუფების სიხშირეებს [32];
- SI_PositionalColor - მიიღება სურათის დაშლით მართკუთხედებად, რომლითაც განისაზღვრება შემდეგ საშუალო ფერი;
- SI_Texture - შეიცავს განმეორებადი ელემენტების სიდიდეებს, სიკაშკაშის ვარიაციებს და დომინირებად მიმართულებებს.

ყველა მახასიათებელი ხელმისავდომია SI_Score მეთოდით, რომელიც ითვლის სურათის დისტანციას მახასიათებლებში და იძლევა შედეგს 0 -:-1 ინტერვალში.

SI_Feature-ს ყველა ქვეტიპს აქვს მახასიათებლების ამოღების ფუნქცია (იმავე სახელით, როგორიც აქვს ქვეტიპს), რომელიც სურათს გამოიყენებს არგუმენტის სახით და იძლევა ქვეტიპის ობიექტს შედეგის სახით.

SI_AverageColor და SI_ColorHistogram ქვეტიპთა ობიექტებს შეუძლია ამასთანავე უშუალოდ იქნას აგებული კონსტანტების ჩაწერის საშუალებით. საილუსტრაციო მაგალითი მოცემულია საშუალო ფერის სინტაქსის აღწერით.

```
create type SI_Feature
method SI_Score (image SI_StillImage)
    returns double precision

create type SI_AverageColor under SI_Feature as
      (SI_ AverageColorSpec SU_Color)
method SI_ AverageColor (
      ReadValue integer,
      GreenValue integer,
      BlueValue integer
)  returns SI_AverageColor

create function SI_AveageColor (Image SI_StillImage)
    returns SI_AverageColor
```

სურათი შეიძლება აღიწეროს სხვადასხვა მახასიათებლით (თვისებებით), და მისი საშუალებით შეუძლებელია შედარების თვითშეფასების განხორციელება, ამიტომ

შემოთავაზებულია შესაძლებლობა, ისინი გაერთიანდეს ერთიან თვისება-მნიშვნელობის წყვილების სიაში. იგი ქმნის ტიპს SI_FeatureList. მას აქვს აგრეთვე SI_Score მეთოდი, რომელიც შეწონილ საშუალო მნიშვნელობას იძლევა ცალკეული Score შეფასებისათვის:

```
  self.SI_Features[1].SI_Score(img) * self.SI_Weights[1]
+ self.SI_Features[2].SI_Score(img) * self.SI_Weights[2] + . . .
/ (self.SI_Weights[1] + self.SI_Weights[2] + . . .)
```

კონსტრუქტორის მიერ ხდება ზუსტად ერთი თვისებისა და ერთი წონის სიის ინიციალიზაცია, რომელიც SI_Append მეთოდით ამატებს მომდევნო თვისებას თავის წონასთან ერთად.

```
create type SI_FeatureList as (
      SI_Features SI.Feature array[SI_MaxFeatureNumber],
      SI_Weights double precision array[SI_MaxFeatureNumber]
  )
method SI_FeatureList (firstFeature SI_Feature, weight double precision)
    returns SI_FeatureList
method SI_Append (feature SI_Feature, weight double precision)
    returns SI_FeatureList
```

განვიხილოთ ერთი მაგალითი, რომელშიც შეჯამებულია აღწერილი შესაძლებლობები SQL-ის კონტექსტში. წარმოდგენილია Logos რელაცია, რომელიც სურათის ერთი Logo ატრიბუტით უნდა იქნას გამოკვლეული. შესადარებელი ობიექტის სახით გამოიყენება bspLogo, რომლიდანაც მიიღება ტექსტურა და ფერთა ჰისტოგრამა. ამ მახასიათებლებიდან აიგება თვისებათა სია, რომელშიც ჯერ შეიტანება ტექსტურა წინით 0.8 (კონსტრუქტორი) და შემდეგ ფერთა ჰისტოგრამა წონით 0.2 (SI_Append მეთოდი). შედგენილი თვისებების სიიდან გამოიძახება SI_Score, და, კერძოდ, ველით Logo პარამეტრის სახით. მსგავსებისთვის შეწონილი საშუალო მნიშვნელობა უნდა იყოს 0.7-ზე მეტი.

```
select * from Logos
wher  SI_FeatureList (
          SI_Texture (bspLogo), 0.8
    ).SI_Append (
        SI_ColorHistogram (bspLogo, 0.2)
    ).SI_Score (Logo) > 0.7
```

დასასრულ, შეიძლება მოკლედ შევაჯამოთ SQL/MM სტანდარტის წინადადებები.

Spatial და Still Image-ს გამოყენების დროს გვხვდება ყველგან ST ან SI პრეფიქსები, რომლებიც Full Text შემთხვევაში არ არსებობდა. Full Text -ს აქვს ფუნქცია Rank, რომელიც

Still Image-სთვის გვხვდება Score-ს სახით. ვინაიდან სპეციალიზიბა ხდება ტიპებზე, შეიძლებოდა ასევე MM_Object-თვის განზოგადება ან მსგავსი ხერხის შემოთავაზება, რომლის დროსაც ერთი (ვირტუალური) MM_Score მეთოდი თავის ადგილს იპოვნის.

22.2. ობიექტრელაციური მულტიმედიური მონაცემთა ბაზის სისტემები

წინა პარაგრაფში შემოტანილ იქნა მონაცემთა ახალი ტიპები მულტიმედიური ობიექტებისათვის და იყო მცდელობა, შეძლებისდაგვარად, განხორციელებულიყო ჯერ არარსებული კავშირების ასახვა მონაცემთა მოდელზე. ამჟამად ჯერ კიდევ არ არსებობს მულტიმედიურ მონაცემთა (მაგალითად, მულტიმედიური დოკუმენტები აგრეგაციული სტრუქტურებით) ობიექტების მოდელირების საშუალებები [32].

შემოთავაზებულია მონაცემთა მოდელების იმ შესაძლებლობათა გამოყენება, რომლებშიც დამოკიდებულებები და აგრეგაციები (სხვადასხვა ფორმით) ხელმისაწვდომია. ამას მივყავართ კითხვამდე, თუ როგორ იქნება ახალი მონაცემთა ტიპები ჩაშენებული ამ მონაცემთა მოდელების კონტექსტში და როგორი სახე ექნება შემდგომ მათ გამოყენებას.

ეს საკითხი, პირველ რიგში, გამოკვლეულ იქნება რელაციური ბაზების სისტემის მაგალითზე. დღეს ბევრს საუბრობენ ობიექტრელაციურ სისტემებზე, ამასთანავე მათი გაფართოებები არის ძალზე სასარგებლო მულტიმედიური ობიექტებისათვის. ასეთ სისტემების საფუძველი ყოველთვის რელაციური მოდელებია. მათ უდავოდ აქვს უდიდესი პრაქტიკული მნიშვნელობა.

ჩვენ ქვემოთ დავახასიათებთ ჯერ მათ იმ განსაკუთრებულ თვისებებს, რომლებიც მულტიმედიისათვისაა მთავარი, შემდეგ ნაჩვენები იქნება, როგორ პროექტდება სქემები, რომლებიც მულტიმედიური ობიექტების ჩადგმულ მონაცემთა ტიპებს გამოიყენებს. და ბოლოს, ნაჩვენები იქნება, თუ როგორ შეიძლება SQL-მოთხოვნების ენით, რომელიც ასეთი სისტემების მნიშვნელოვანი ენაა, განხორციელდეს მონაცემებთან მიმართვა ობიექტრელაციურ მონაცემთა ბაზებში.

- **მოკლე დახასიათება**

ობიექტრელაციური მონაცემთა ბაზების სისტემა (ორმბს) წარმოადგენს მცდელობას, რელაციური მბს ისე განვითარდეს და ახალი კონცეფციებით გამდიდრდეს, რომ მან შეძლოს, მინიმუმ, ფუნქციონალობათა ერთი ნაწილის შემოთავაზება, რომელიც აქამდე ცნობილი იყო ობიექტორიენტირებული მბს-თვის. ესენია, კერძოდ, ობიქტთა იდენტურობა, მომხმარებელთა მიერ განსაზღვრებადი მონაცემთა ტიპები და ტიპთა იერარქიები. თუმცა ძირითადი სტრუქტურა ყველა მონაცემთა ორგანიზაციისა და მონაცემთა შენახვისა ისევ ცხრილები რჩება. ასე რომ, ყველა ეს ახალი გაფართოება უნდა დაექვემდებაროს, როგორც ადრე, ამ (წინა) კონცეფციას.

მულტიმედიური მონაცემებისათვის თანამედროვე სტანდარტებით განიხილება დიდი ობიექტები (large objects), ორი ფორმატით: სიმბოლოების ობიექტი (character large object - CLOB) და ბინარული ობიექტები (binary large object - BLOB).

არსებობს გარკვეული შეზღუდვები ამ ტიპის ატრიბუტებისათვის: ისინი არ შეიძლება იყოს განსაზღვრული როგორც პირველადი გასაღებური ატრიბუტები, UNIQUE ან მეორეული გასაღებური ატრიბუტები. აგრეთვე არ შეიძლება მათი გამოყენება GROUP BY ან ORDER BY კონსტრუქციებში.

განსაკუთრებით მნიშვნელოვანია სტრუქტურირებული, მომხარებალთა მიერ განსაზღვრული ტიპები (UDT's). ეს ტიპები შეიცავს ატრიბუტებს და მეთოდებს. ასევე ძველ ფუნქციებს და პროცედურებს, რომლებიც 1996 წლიდად SQL92-ის სტანდარტის გაფართოებით იქნა განსაზღვრული, შეეძლოთ ასეთი ტიპის პარამეტრებით სარგებლობა [70].

პროცედურები განსაზღვრული იყო ისე, რომ ისინი შედეგის მნიშვნელობას უკან არ აბრუნებდა, მაგრამ შემავალი და გამომავალი პარამეტრები შეიძლება ჰქონოდა.

ფუნქციები, პირიქით, აბრუნებს უკან მიღებულ შედეგებს და აქვს მხოლოდ შემავალი პარამეტრები.

ტიპები შეიძლება იყოს ორგანიზებული იერარქიებად. ამავე დროს, ტიპები მიეკუთვნება ეგზემპლარულს (instantiable), თუ მათ შეუძლია უშუალოდ შედეგის მნიშვნელობების მოცემა. საწინააღმდეგო შემთხვევაში ტიპები აბსტრაქტული ან ვირტუალურია, რაც ნიშნავს, რომ მათ შეუძლია მხოლოდ ქვეტიპების მნიშვნელობებზე მითითება. ქვეტიპების განსაზღვრა ამით თავიდან აცილებულია, რადგანაც ტიპი თვითონ ასრულებს ამ ფუნქციას.

იერარქიულობის ეს პრინციპი არის ურთიერთშენაცვლების: სადაც ტიპის მნიშვნელობა შეიძლება მოთავსდეს, იქ შეიძლება ასევე ქვეტიპების მნიშვნელობათა არსებობა. იმისათვის, რომ ეს შესრულდეს, ტიპის ატრიბუტები და მეთოდები ასევე წვდომადი უნდა იყოს მისი ყველა ქვეტიპისათვის, ანუ მემკვიდრეობითობის თვისება უნდა იყოს რეალიზებული.

განვიხილოთ მაგალითი ტიპისა და ქვეტიპებისათვის.

```
create type Student
    under Person
     as (
        MatrNr  integer,
        Studienfach varchar(30),
    )
    instance method Durchschnittsnote()
         return real
         language SQL
```

```
        deterministic
        contains SQL
    . . . ;
```

მეთოდები მხოლოდ ცხადდება ტიპების სღწერაში, ანუ განისაზღვრება მათი სიგნატურები. რეალიზაცია შეიძლება მოგვიანებით განხორციელდეს (ბრძანება create method). ჩვეულებისამებრ, განასხვავებენ ეგზემპლარულ მეთოდებს (instance method) კლასთა მეთოდებს (static method).

დამატებითი ინფორმაცია მეთოდისათვის შემდეგია:

- returns - განსაზღვრავს უკან დასაბრუნებელ მნიშვნელობის ტიპს. იგი აღწერილია create-method ბრძანებაში;
- language SQL – მიუთითებს, რომ მეთოდის ტანის რეალიზაცია მთლიანად ხორციელდება SQL -ში;
- deterministic – მიუთითებს, რომ უშუალოდ ერთმანეთის მომდევნო გამოძახებები ერთი და იმავე პარამეტრების მნიშვნელობებით, ყოველთვის ერთსა და იმავე შედეგს იძლევა;
- contains SQL – მიუთითებს, რომ მეთოდის რეალიზაცია შეიცავს SQL-ბრძანებას, რომელიც არ ეხება ეგზემპლარის ან კლასის მონაცემებს, არამედ მხოლოდ სხვა მონაცემებს ეხება. ალტერნატიული ინფორმაციაა no SQL, როცა არაა მოცემული SQL-ბრძანება; reads SQL data, როცა ეგზემოლარის ატრიბუტები მხოლოდ უნდა წაკითხულ იქნას; modifies SQL data, როცა ატრიბუტები ასევე უნდა შეიცვალოს;
- returns null on null input - მიუთითებს, რომ შედეგი არ ბრუნდება უკან, როცა შემავალი პარამეტრია Nullwert (zero values);

ერთი ტიპის ყოველი ატრიბუტისათვის ორი მეთოდი ავტომატურად გენერირდება:

დამკვირვებელი (Observer) პასუხისმგებელია წაკითხვის წვდომაზე.

instance method **MatrNr** ()

```
    returns integer
    language SQL
    deterministic
    contains SQL
```

ვინაიდან მეთოდის გამოძახებისას პარამეტრების გარეშე ყოველთვის ხდება ფრჩხილების გამოტოვება, ამ მეთოდის ვიზუალური გამოძახება არ განსხვავდება ატრიბუტებთან ნორმალური წვდომისაგან: Student1.MatrNr.

მუტატორი ახდენს ატრიბუტის მნიშვნელობის ცვლილებას (ჩანაცვლებას):

instance method **MatrNr** (new value)

```
    returns Student
    self as result
```

```
language SQL
deterministic
contains SQL
returns null on null input
```

შემდეგი მაგალითი უფრო გამოკვეთილად შეესაბამება მულტიმედიურ შემთხვევას:

```
create type ImageType
    under <Supertyp>
    as  (< Attributliste> )
    static method countImages ()
    returns integer
    language SQL
    deterministic
    contains SQL
instance method height ()
    returns integer
    language SQL
    deterministic
    rads SQL data
```

კიდევ ერთხელ, საბოლოოდ უნდა გაესვას ხაზი იმას, რომ კორტეჟები და ცხრილები შეუცვლელად თამაშობს მნიშვნელოვან როლს: იგი უშუალოდ დაკავშირებულია კორტეჟების შეტანასთან ცხრილებში, სხვა ობიექტები ან მნიშვნელობები არ იქმნება. ასევე ყოველთვის შესაძლებელია მოთხოვნები ცხრილებისადმი, რომლებიც გამოსასვლელზე იძლევა ისევ ცხრილებს.

- **ობიექტრელაციური ბაზის სტრუქტურები მულტიმედია ობიექტისათვის**

ობიექტრელაციურ ბაზის სქემაში დასაშვებია მონაცემთა ახალი ტიპების გამოყენება მულტიმედიური ობიექტებისათვის, როგორც მნიშვნელობათა არეები (domains). ატრიბუტები შეიძლება იყოს ტიპებით text, image და სხვ. მაგალითად, ყოველი ამომრჩევლისათვის კორტეჟში უშუალოდ ჩაისმება პირადი ნომერი, გვარი, დაბადების თარიღი, და ა.შ., ასევე ფოტო (სურათი), თითების ანაბეჭდი, ხმა და მოხდება მისი შენახვა:

```
Voter ( Pers_N integer,
        PersName varchar (50),
        BirthsDate date,
        Adress varchar (100),
           . . .
```

PersFoto **image,**
PersFingerprints **image,**
PersVoice **sound**)

დავარქვათ ამ სტრუქტურას, პირობითად, რელაციური სქემა (ტიპი_1), რათა შემდგომში შემოტანილი სხვა ტიპისაგან განვასხვავოთ.

ნორმალურ ფორმათა თეორიის საფუძველზე, სქემათა ოპტიმალური სტრუქტურის მისაღებად, ხშირად საჭიროა ერთი რელაციის დეკომპოზიცია ორ ან მეტ რელაციად, რომლებშიც მოხდება შესაბამისი კორტეჟების გადანაწილება, პირველადი და მეორეული გასაღებური ატრიბუტების განსაზღვრა და ა.შ.

ჩვენი მაგალითისათვის შესაძლებელია ამომრჩევლის პირადი ტექსტური მონაცემების ერთ რელაციაში შენახვა, ხოლო მულტიმედიური მონაცემებისა მეორეში. ქვემოთ ნაჩვენებია ეს შემთხვევა:

Person (Pers_N integer,
PersLastName varchar (50),
PersFirstName varchar (30),
BirthsDate date,
Adress varchar (100),
. . .
)
Voters_MM_Daten(Pers_N integer,
PersFoto **image,**
PersFingerprints **image,**
PersVoice **sound**)

მიღებული სტრუქტურა არის რელაციური სქემა (ტიპი_2). ამ შემთხვევაში ამომრჩევლის ვინაობა (გვარი, სახელი,...) უკავშირდება მულტიმედიური მონაცემების რელაციას პირველადი გასაღებური ატრიბუტით Pres_N. ამ ორი რელაციის საფუძველზე მოთხოვნების დასამუშავებლად საჭირო იქნება „Join" გაერთიანების ოპერაციის გამოყენება.

არსებობს შემთხვევები, როდესაც რელაციებს შორის არსებობს m:n დამოკიდებულება. ამ დროს საჭიროა რელაციებსშორის კავშირების რეალიზაცია დამატებითი ინდექსების საფუძველზე, მათ შორის პირველადი და მეორეული გასაღებებითაც. განვიხილოთ მაგალითი, რომელიც ასახავს მაჟორიტარი დეპუტატის (Majority_Deputy) განაწილებას რეგიონის (Region) მხარის (Area) საარჩევნო უბნებზე (Polling_Districts). ერთი დეპუტატი კენჭს იყრის რამდენიმე საარჩევნო უბანზე, ასევე ერთ საარჩევნო უბანზე რამდენიმე კანდიდატია, ანუ საქმე გვაქვს m:n დამოკდებულებასთან:

Majority_Deputy (majority_deputyID integer,
Deputy_Name varchar (50),

```
        . . .
        Deputy_Foto image
       )
Polling_District ( polling_districtID integer,
        Polling_DistrictNr varchar (20),
        polling_district_AddressName varchar (100),
        areaID integer,
        RegionID integer
       )
Results_List (polling_districtID integer,
        majority_deputyID integer,
        number_votes integer,
        Percentage double,
        Position integer
       )
```

მივიღეთ რელაცირი სქემა (ტიპი_3). ამგვარად, შეიძლება შევაჯამოთ შედეგები:

- სამივე ტიპის რელაციური სქემა მულტიმედიურ ობიექტებსა და არსებს შორის შეიძლება აისახოს 1: 1, 1 : n, m : n დამოკიდებულებათა სახით, სპეციალური სემენტიკის გარეშე;
- მულტიმედიური ობიექტები შეიძლება გამოვლინდეს როგორც ატრიბუტები ან როგორც არსები;
- მიმართვა ზოგიერთ რელაციურ სქემაზე შეიძლება იყოს მოუხერხებელი და მოითხოვდეს რამდენიმე გაერთიანების ოპერაციის (Join) გამოყენებას.

22.3. მოთხოვნების დამუშავება ობიექტრელაციურ მონაცემთა ბაზებში

რელაციურ მონაცემთა ბაზების შესახებ, რომლებშიც ტრადიციული, ტექსტური ტიპის ატრიბუტების გარდა არის აგრეთვე მულტიმედიური ტიპებიც, როგორიცაა image, graphics და ა.შ., აქამდე ზედაპირულად იყო დახასიათებული. აქვე ნახსენები იყო მომხმარებლისათვის მოუხერხებელი გამოყენების საშუალება გაერთიანების ოპერაციის სახით.

იმისათის, რომ აქ შედარებით ზუსტი სურათის გადმოცემა მოხერხდეს, საჭიროა წარმოდგენილ იქნას ახალ ტიპებზე განსაზღვრული ოპერაციების სავარაუდო ჩანერგვა (embedding) რელაციურ მოთხოვნის ენაში. ამას გვთავაზობს SQL ენა, რომელიც სტანდარტის სახით დამკვიდრდა მოთხოვნათა ენებისათვის.

განვიხილოთ მარტივი, ორატრიბუტიანი რელაციის მაგალითი:

Aerial_image (Nr integer,
Picture image)

კორტეჟების შესატანად ბაზაში საჭიროა SQL-ენის insert-ოპერატორის გამოყენება. ამ დროს ცალკეული ატრიბუტების მნიშვნელობები გადაეცემა როგორც კონსტანტები ან პროგრამის ცვლადები. მაგალითად,

insert into Aerial_image
values (:nr, image(:pr, :cm));

სადაც pr-ში ინახება სურათი, ხოლო cm-ში ფერთა ცხრილები.

გამოსახულებაში ორი წერტილის შემდეგ დგას პროგრამული ცვლადები, რითაც ისინი სინტაქსურად განსხვავდება რელაციების და ატრიბუტების სახელებისაგან.

value წინადადების პირველი ჩანაწერი ეკუთვნის ატრიბუტის ნომერს და უნდა იყოს integer ტიპის. მეორე ჩანაწერი ეკუთვნის სურათს და ტიპი image. კომპილატორი ცნობს image ტიპის ოპერაციების პარამეტრებისა და შედეგის ტიპებს და შეუძლია შესაბამისი შემოწმების ჩატარება. ეს პრინციპი ძალაშია შემდგომი განხილული ოპერატორებისთვისაც.

თუ კორტეჟი სურათის ატრიბუტებით ერთხელ უკვე შენახულია მონაცემთა ბაზაში, მაშინ შესაძლებელია მათი SQL-ის update ბრძანებით ცვლილება.

update Aerial_image
set Picture = Picture.replaceColormap(:yuv, 4096, 24, : cm)
where Nr = 1286;

შინაარსობრივი მონაცემების მისაღებად შესაძლებელია შემდეგი სახის ბრძანების ჩაწერა:

update Aerial_image
set Picture = Picture.newDescr (
„მოედანს, რომლის ცენტრში ძეგლი და გარშემო\n
ყვავილებია, უერთდება ხუთი ქუჩა“
where Nr = 1234;

განსაზღვრული კორტეჟების მოსაძებნად (განსაზღვრული სურათებისთვის) შესაძლებელია ჩვეულებრივი SQL გამოსახულებების გამოყენება. ატრიბუტთა მნიშვნელობების შედარება კონტანტებთან, რომელიც ამ დროს მთავარ როლს თამაშობს, დასაშვებია, უპირველეს ყოვლისა, მხოლოდ სტანდარტული ტიპებისათვის. მულტიმედიური ტიპებისათვის კი არსებობს სპეციალური შედარების ოპერაციები.

პროგრამაზე გადაცემის დროს image ტიპის ატრიბუტებს არ შეუძლია უშუალოდ პროგრამის ცვლადებზე მინიჭება, რადგან ცვლადთა ტიპები სხვადასხვა დანართში შეიძლება სხვადასხვა იყოს. პირიქით, კომპონენტების ამორჩევა და მასთან დაკავშირებული ტიპების შერჩევა (ადაპტაცია) ხდება ცხადად შესაბამისი image - ოპერაციებით:

```
select Picture.getPixrect(), Picture.getColormap()
into :pr,  :cm
from Aerial_image
where Nr = :k;
```

ამ მაგალითში k-ცვლადში მოცემულია სურათის ნომერი, რომელშიც სურათი (Picture) იქნება გამოძახებული მონაცემთა ბაზიდან Pixrect ფორმატში. შსაძლებელია ასევე სურათის ატრიბუტების თვისებათა შერჩევა, თუ იგი წინასწარ image-ოპერაციების გამოყენებით მონაცემთა ტიპებისთვის შეიქმნა, რომლებზეც განსაზღრულია შედარების ოპერაციები:

```
select Picture.height(),  Picture.width()
into :hoehe,  :breite
from  Aerial_image
where Picture.pixelcount( :dunkelbraun) < 1000
and Picture.noOfColors() > 4095;
```

შეიძლება ალტერნატიული ვარიანტის განხილვაც image - მონაცეთა ტიპის შედარების ოპერაციისთვის:

```
select Picture.description(), . . .
from Aerial_image
where Picture.contains("ცენტრში ძეგლი");
```

ეს, უპირველეს ყოვლისა, უჩვენებს მომხმარებლის ინტერფეისის სიმარტივეს. ზემოთ აღწერილი select-ბრძანება იძლევა სურათს ნომრით Nr1234, ვინაიდან ამ აღწერაში არის სიტყვები „ცენტრში ძეგლი" ნახსენები, და იგი როგორც საძებნი გამოსახულება „ცენტრში ძეგლი", ისე მოიაზრება.

შემდეგ, შენახვისა და მოთხოვნის შემთხვევებში შეიძლება გამოიცეს შეცდომის შეტყობინება, როგორიცაა, მაგალითად, „ამ სიტყვას არ ვიცნობ" ან „ეს ტექსტი არ მესმის" და სხვ. ესაა ის ღირებულება, რომლითაც მარტივი ტექსტების შედარებისაგან განსხვავებით მომხმარებლის ინტერფეისი საძებნი პროცედურის უკეთესი ხარისხით გამოირჩევა.

instert – ბრძანებაში დაუშვებელია კომბინაცია values-წინადადებისა (კონსტან-ტების ან პროგრამის ცვლადების მონაცემები) და ქვეარჩევის (subselect) (ატრიბუტების ახალი მნიშვნელობების შეკრება მონაცემთა ბაზისთან მიმართვის შესახებ). ასეთი პროცედურა უფრო მაშინაა საჭირო, როდესაც სურათის ნაწილი, ამოღებული მონაცემთა ბაზიდან, უნდა იქნას შენახული სხვა კორტეჟში.

ამ ახალი კორტეჟის ფორმატირებული ატრიბუტები მიიღება არა მონაცემთა ბაზიდან, არამედ პროგრამებიდან. მიზანშეწონილია კონსტანტების მიწოდება select-წინადადებაში, რაც არც ძალიან გასაგებად გამოიყურება:

```
insert into Aerial_image
        select  :neueNr, Bild window (50,50,100,100)
        from Aerial_image
        where Nr = :alteNr;
```

update - ბრძანების სემანტიკა, სპეციალური set-წინადადება, არის მნიშვნელობის ჩანაცვლება (აღდგენა), და არა მნიშვნელობის მოდიფიკაცია. მნიშვნელობის მინიჭების კონსტრუქციის მარჯვენა ნაწილში შეიძლება ნებისმიერი სირთულის გამოსახულება იყოს, რომელსაც უნდა ჰქონდეს სწორად შერჩეული შედეგის ტიპი. მაგალითად:

```
update  Aerial_image
apply Picture.replaceColormap ( :cm);
where Nr = 1234;
```

SQL - ბრძანებები პროგრამული ენების სინტაქსის მსგავსად პროცედურების გამოძახებას ჰგავს, და არა ფუნქციურ გამოსახულებას, რომელთაც მნიშვნელობა აქვს. ამიტომაც ბრძანებები, როგორც მაგალითად ქვემოთაა ნაჩვენები, ძალზე მგრძნობიარეა მონაცემთა ახალი ტიპების მიმართ, რათა თავიდან იქნას აცილებული შრომატევადი დუბლირების პროცესები:

```
var := select . . . from . . . where . . . ;
writeScreen(select Picture.pixelmatrix ( ) );
```

მიზანშეწონილია, რომ მეთოდები აღიწეროს დეტალურად. ეს ნიშნავს იმას, რომ მოხდეს განსხვავება წასაკითხსა და ჩასაწერს შორის, და მულტიმედიური კონტექსტის თვალსაზრისით, უპირველეს ყოვლისა, განასხვავონ დროზე დამოკიდებული და დამოუკიდებელი პროცესები.

რელაციური მონაცემთა ბაზები, მოთხოვნების დამუშავების თვალსაზრისით, წლების განმავლობაში ასრულებს განსაკუთრებულ როლს, სხვა ახალი მონაცემთა ბაზებისაგან განსხვავებით. მათი გაფართოება ახალი ტიპის, მულტიმედიალური მონაცემებით ძალზე ეფექტურია და პრაქტიკულად ღირებული. მომხმარებელს, მათი გამოყენების მიზნით, სჭირდება ამ მიმართულებით დამატებითი ცოდნის მიღება.

22.4. ობიექტორიენტირებული მულტიმედიური მონაცემთა ბაზის სისტემები

პირველი ნაშრომები მულტიმედიური მონაცემთა ბაზების შესახებ გამოჩნდა 80-ან წლებში, როდესაც მონაცემთა რელაციური მოდელების თემატიკა, როგორც ახალი მეცნიერული მიმართულება, ძალზე აქტუალური და დომინირებადი იყო [67, 71-73]. გასაოცარი არაა, რომ ამ დროს მულტიმედიური მონაცემთა ბაზების სისტემების პირველი პროექტები მოიაზრებოდა როგორც აუცილებლად ობიექტ ორიენტირებული

სისტემები [74,152]. მაგრამ ამ პერიოდში ვერ მოხერხდა ასეთი სისტემების პრაქტიკული რეალიზაცია.

ამ პრობლემებზე მუშაობა და მნიშვნელოვანი მიღწევები მიღებულ იქნა 90-ანი წლებიდან, როდესაც ODMG (Object Data Management Group) ჯგუფმა წარმოადგინა რეალაციური ბაზებისათვის SQL-ის მსგავსი ეფექტური კონცეფცია [76].

- **მონაცემთა მულტიმედიური ტიპების ჩაშენება**

ობიექტორიენტირებულ სისტემებში მულტიმედიური მონაცემთა ტიპების ასახვა ხდება უშუალოდ როგორც კლასები. ეგზემპლარები უნდა იყოს ინკაფსულირებული, ისე, რომ წვდომის განხორციელება შეიძლებოდეს მხოლოდ კლასის მეთოდებზე.

როგორც ობიექტრელაციურ მოდელში, ეს კლასები შეიძლება იყოს ორგანიზებული განზოგადებული (გენერალიზებული) იერარქიით. ისინი გამორიცხავს საერთო მეთოდების განმეორებად განსაზღვრებებს და ნებას იძლევა მონაცემთა სპეციალური ტიპების შესატანად, რომლებისთვისაც დამატებითი მეთოდებია განსაზღვრული.

მონაცემთა ტიპები Text, Image, ასევე Graphics, Audio და Video განიხილება როგორც MediaObject-ის ქვეკლასები, და მემკვიდრეობით იღებს მის მეთოდებს. ისინი აფართოებენ ამ მეთოდებს საკუთარი ოპერაციებით, რომლებიც მორგებულია სპეციალურ მულტიმედიურ მონაცემთა ტიპებს, მაგალითად, image-ს დროს height, ან Text-ის დროს length.

ORION სისტემა ან MIM (Multimedia Information Manager)-ით აღნიშნული კლასების კოლექცია წარმოადგენს კიდევ ერთ შრეს MediaObject-სა და Text ან Image ტიპებს შორის, რომლების შეიცავს კლასებს „წრფივი მედიაობიექტები“ და „სივრცითი მედიაობიექტები“. პირველი მოიცავს ტექსტებს და აუდიოს, ხოლო მეორე სურათებს, გრაფიკას და ვიდეოს. მართლაც, ამ მეთოდებს, როგორიცაა length და height, შეუძლია ფაქტობრივად განსაზღვრულ იქნას ამ კლასებზე და შემდეგ მემკვიდრეობით იქნას გამოყენებული. სხვა მხრივ უფრო მნიშვნელოვნად ჩანს კლასიფიკაცია როგორც „დროზე დამოკიდებული“ და „დროზე დამოუკიდებელი“. ამიტომ გადაწყვეტა ასეთი შუალედური შრის შესახებ ჯერჯერობით არ განიხილება.

კლასების იერარქიის გენერალიზაცია და აგება საშუალებას იძლევა მოვახდინოთ არა მხოლოდ მულტიმედიური ობიექტების სასარგებლო ჩანერგვა, არამედ აგრეთვე ობიექტებისაც (Entities), რომლებსაც ისინი ასახავს ან აღწერს.

წინა პარაგრაფში ობიექტრელაციური მოდელების დისკუსიამ გვიჩვენა, რომ სათანადო შემთხვევა სხვადასხვა is.presented_in დამოკიდებულება ამოდელირებს და ეს ძებნის დროს ყველა ცხადად უნდა იქნას გათვალისწინებული.

გენერალიზაცია, პირიქით ნებას იძლევა, რომ ამომრჩეველი, დეპუტატობის_კანდიდატი და საარჩევნო_უბნის_თანამშრომელი ზემნიშვნელობით

(Superclass) გავაერთიანოთ, როგორიცაა „პიროვნება" და მათი ეგზემპლარები სურათებით დავაკავშიროთ.

რომელი ზე მნიშვნელობა მიესადაგება დასმულ მიზანს, დამოკიდებულია იმაზე, თუ რომელი ობიექტებია სურათებზე, ტექსტებში, გრაფიკებზე და ა.შ. უფრო რელევანტური (შესაბამისი) გამოსაყენებლად. ყველაზე ზოგად შემთხვევაში, შეძლებისდაგვარად, თუ მოცემულია პრესიდან სურათი (ფოტო), საჭიროა ისევ „Object" კლასზე მიმართვა.

ობიექტორიენტირებულ მოდელს შეუძლია მე-3 ტიპის სქემის (იხ. წინა პარაგრაფი) უკეთესად წარმოდგენა, ვიდრე რელაციურ მოდელს. როგორი მდგომარეობაა სქემის სხვა ტიპებისათვის კლასის ეგზემპლარები გამოისახება კორტეჟების სახით ატრიბუტების მნიშვნელობებზე, რომელთა მნიშვნელობათა არეები დადგენილია კლასების განსაზღვრებაში. კლასიკური რელაციური მოდელისაგან განსხვავებით მნიშვნელობათა ეს არეები არ უნდა იყოს არჩეული ერთი წინასწარგანსაზღვრული სიმრავლიდან, არამედ შესაძლებელია ასევე იყოს ნებისმიერი თვითგანსაზღვრებადი კლასი.

მაგალითად, employee კლასს შეუძლია თავისი ეგზემპლარებისათვის განსაზღვროს ატრიბუტი (ეგზემპლარის ცვლადი) Portrait, რომელთა მნიშვნელობები შეიძლება იყოს GreyscaleImage კლასის ეგზემპლარები. ეს შეესაბამება სქემის 1-ელ ტიპს რელაციური ბაზისათვის.

ობიექტორიენტირებული სისტემები არ მოითხოვს ნორმალიზაციას თავისი კლასებისა და ეგზემპლარებისათვის, ასე რომ ნებადართულია ატრიბუტები რამდენიმე მნიშვნელობით. აუცილებლობის შემთხვევაში საჭიროა ტიპების კონსტრუქტორის, როგორიცაა list ან set გამოყენება. შესაბამისად იგი მე-2 ტიპის სქემასთან შედარებით ოდნავ ჭარბია. Portrait ატრიბუტს შეუძლია აგრეთვე მარტივად ჰქონდეს რამდენიმე სურათი.

მრავალი ობიექტორიენტირებული სისტემა არ განსხვავდება სუფთად ობიექტის ატრიბუტებითა და კომპონენტებით. ხშირად აგრეგაცია გამოისახება მარტივად ატრიბუტთა დახმარებით. ODMG-მოდელი მკაფიოდ გამოყოფს ატრიბუტებს დამოკიდებულებებისაგან და თუ ატრიბუტებს შეუძლია მხოლოდ მნიშვნელობების მიღება, ობიექტები უკავშირდება ერთმანეთს დამოკიდებულებებით.

ობიექტორიენტირებული სისტემების სუსტი ადგილია სიმრავლეებზე ორიენტაციის არარსებობა და მასთან დაკავშირებული ძებნა. ეს გასაგები იყო, ამიტომაც ცდილობდნენ ობიექტორიენტირებული ბაზების მართვის სისტემებისათვის მოთხოვნების ენის განსაზღვრას [77].

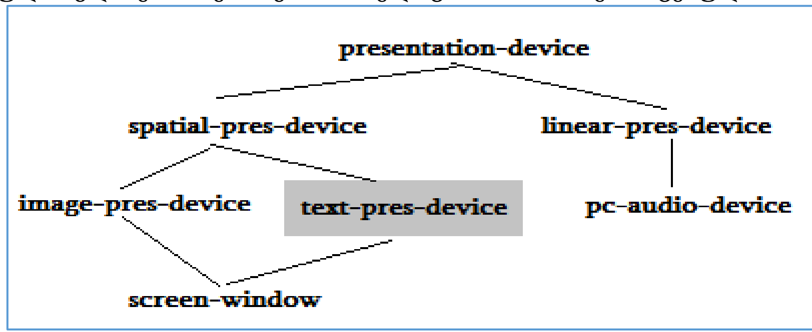
OMDG-მოდელში არსებობს ამისთვის OQL-ენა (Object Query Language), რომელიც ორიენტირებულია SQL-ზე და ზოგიერთ შემთხვევაში სრულად თავსებადიცაა. მას მოაქვს თან ყველა სასურველი თვისება, მაგრამ არ შეესაბამება ობიექტებზე პირველად წვდომას, რისთვისაც იყო შექმნილი ოო-ბაზების სისტემები: იგულისხმება ირიბი

მიმართვა. ვინაიდან OQL ენა ძალზე რთულია, მისი გამოყენება ნაკლებად გვხვდება პროგრამულ პროდუქტებში.

➢ **მულტიმედიური ინფორმაციული მენეჯერი (MIM)**

ობიექტორიენტირებული მულტიმედიური მონაცემთა ბაზის სისტემების კონკრეტული მაგალითი განვიხილოთ ORION სისტემის მაგალითზე, რომელიც ასევე ცნობილია მულტიმედიალური ინფორმაციული მენეჯერის (MIM – Multimedia Information Manager) სახელით [77-79]. ესაა ობიექტ ორიენტირებული კლასების ბიბლიოთეკა. ასეთი ობიექტორიენტირებული მონაცემთა ბაზების სისტემა მომხმარებლისთვის აადვილებს მულტიმედიური ობიექტები დაიყვანონ სუბკლასებამდე და შემდეგ ისარგებლონ MIM ბიბლიოთეკის ოპერაციებით მემკვიდრეობითობის მექანიზმის გამოყენებით [77].

MIM-ის განსაკუთრებული თავისებურება მდგომარეობს იმაში, რომ ყველა მოწყობილობა წარმოიდგინება როგორც ობიექტები ORION-ში, კერძოდ, როგორც შეტანა/გამოტანის, ასევე მეხსიერების მოწყობილობები. გამოტანის მოწყობილობისთვის მომზადებულია კლასების იერარქია, რომელიც 22.2 ნახაზზეა მოცემული.



ნახ.22.2. MIM-ში კლასთა იერარქია გამოსატანი მოწყობილობებისთვის

გრაფის წიბოები ასახავს მემკვიდრეობითობის კავშირებს, სადაც სუბკლასები მოთავსებულია კლასების ქვეშ. მონიშნული კლასები text-pres-device არის მაგალითი გაფართოებისა, რომელიც თვით მომხმარებელმა აირჩია. ასეთი კლასი არ ეკუთვნის MIM-ბიბლიოთეკას.

კლასთა ეგზემპლარები ამ იერარქიაში არაა განაწილებული ცალსახად გამომტანი მოწყობილობებისათვის, არამედ ისინი სპეციფიცირებულია:

- სადაა მოწყობილობაზე ასახული (მაგალითად, ეკრანის რომელ ნაწილში);
- მედიალური ობიექტის რომელი ნაწილი (ამონაჭერი) არის ასახული.

შედეგად შესაძლებელია რამდენიმე ეგზემპლარის მიცემა, რომლებიც ერთი და იმავე ფიზიკურ მოწყობილობას ასახავს.

შეიძლება ასევე მათი ინტერპრეტაცია შეზღუდვებით როგორც „გამოცემის ფორმატი“ მედიური ობიექტებისათვის. მაგალითად, კლასი spatial-pres-device ასე განმარტავს მისი ეგზემპლარის შემდეგ ატრიბუტებს:

upper-left-x,

upper-left-y,

width,

height.

ამ ატრიბუტების მნიშვნელობები შეესაბამება მედიური ობიექტის ამონაჭერს (მაგალითად, რასტრულ სურათს), რომელიც უნდა გამოიცეს სივრცით გამომტან მოწყობილობაზე.

ქვეკლასში screen-window სპეციფიცირდება ატრიბუტები:

win-upper-left,

win-upper-right,

win-width,

win-height

დამატებით ეკრანის ნაწილზე, რომელიც უნდა იქნას გამოყენებული ასახვისთვის.

screen-window -სთვის განსაზღვრულია ასევე მეთოდები, კერძოდ:

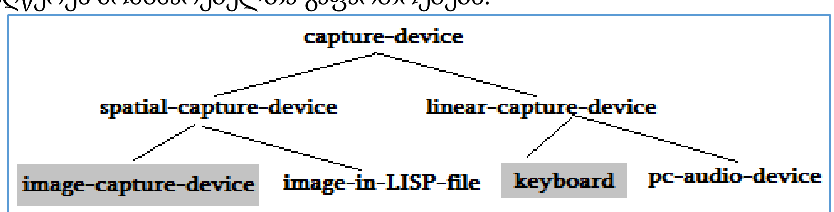
present,

capture,

present-pres,

რომლებიც გამოიყენება განსაზღვრული მულტიმედიური ობიექტების გამოსატანად და, აუცილებლობის შემთხვევაში, გამოიძახება წაკითხვის აღსადგენადაც.

ანალოგიურად არის MIM-ში განსაზღვრული კლასთა იერარქია შესატანი მოწყობილობებისთვის (ნახ.22.3). აქ მონიშნული კლასები, image-capture-device და keyboard აღწერეს მომხმარებელთა გაფართოებებს.



ნახ.22.3. MIM-ში კლასთა იერარქია შესატანი მოწყობილობებისათვის

აქაც, როგორც წინა შემთხვევაში, კლასის ეგზემპლარები მეტია, ვიდრე მხოლოდ სპეციფირებული მოწყობილობები. ისინი ასევე მიუთითებს თუ მულტიმედიური

ობიექტის რომელი ნაწილი განიხილება და როგორაა მოწყობილობა დაკომპლექტებული. შედეგად შესაძლებელია ერთი ფიზიკური მოწყობილობისათვის კვლავ capture-device ტიპის რამდენიმე ეგზემპლარის მიცემა.

spatial-capture-device სუბკლასის ეგზემპლარები მიუთითებს შემდეგ ატრიბუტებს:

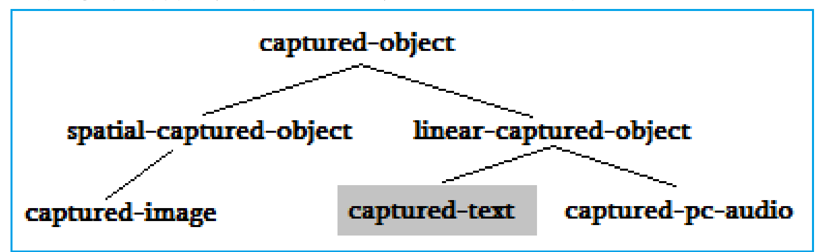
upper-left-x,
upper-left-y,
width,
height.

იგი სპეციფიცირებას უკეთებს სივრცითი მულტიმედიური ობიექტის ამონაჭერს, რომელიც მიიღება შემტანი მოწყობილობით ან ჩანაცვლებით. მომხმარებლის მიერ განსაზღვრულ image-capture-device კლასს შეუძლია

cam-width,
cam-height,
bits-per-pixel

ატრიბუტების და capture მეთოდის წარმოდგენა.

დამახსოვრებული მულტიმედიური ობიექტები ორგანიზებულია ასევე კლასთა იერარქიის სახით. ამ დროს კვლავ განასხვავებენ სივრცით და წრფივ მულტიმედიურ ობიექტებს. რასტრული სურათები და გრაფიკები მიეკუთვნება სივრცითს, ხოლო ტექსტი და აუდიო კი წრფივს. ქვეკლასები ასახულია 22.4 ნახაზზე.



ნახ.22.4. MIM-ში დამახსოვრებული მედიური ობიექტების
კლასთა იერარქია

captured-object კლასებში ყველა ქვეკლასისთვის განსაზღვრულია შემდეგი ატრიბუტები:

storage-object – მიუთითებს storage-device კლასის ერთ ეგზემპლარზე, რომელიც ქვემოთ შემოიტანება.

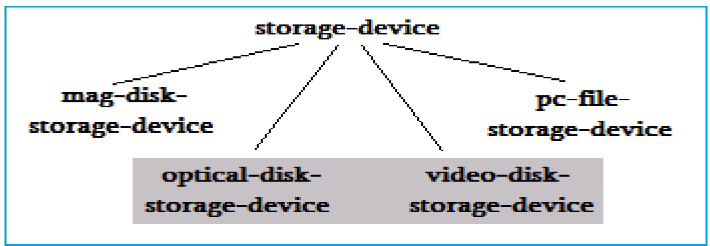
logical-measure - განსაზღვრავს საწყისი მონაცემების ელემენტარულ ერთეულებს მომხმარებლის თვალსაზრისით, რომლებიც არ უნდა ემთხვეოდეს სისტემტექნიკურ ერთეულებს (ბიტი, ბაიტი ან მეხსიერების სიტყვა (memory word)). ასეთი ლოგიკური ზომის ერთეულებია, მაგალითად, წამები აუდიოს დროს, ან კადრები - ვიდეოს დროს.

დამოკიდებულება ფიზიკურ და ლოგიკურ ზომის ერთეულებს შორის აისახება phys-logic-ratio ატრიბუტში. ეს იქნება მაშინ ბაიტი/წამში აუდიოსათვის ან ბაიტი/კადრი ვიდეოსათვის.

spatial-captured-object სუბკლასი იძლევა ატრიბუტებს width, height და row-major. უკანასკნელი გვიჩვენებს, დამახსოვრება მოხდა სტრიქონულად თუ სვეტურად.

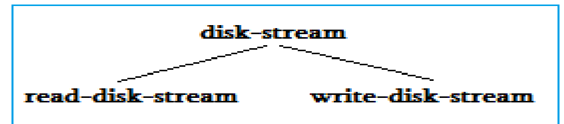
და ბოლოს, არსებობს ატრიბუტი bits-per-pixel, რომელიც ცხადყოფს, რომ საქმე ეხება ტიპურ სარეგისტრაციო მონაცემებს.

დამახსოვრებისთვის არსებობს მოწყობილობები, რომლებიც შესატან და გამოსატან მოწყობილობათა მსგავსად ნაწილობრივ გამოყენებაშია და storage-device კლასის ეგზემპლარების საშუალებით წარმოიდგინება (ნახ.22.5).



ნახ.22.5. MIM-ში შენახვის მოწყობილობების
კლასთა იერარქია

და ბოლოს, კლასებში არსებობს კიდევ ორგანიზებული მონაცემთა ობიექტები, რომლებიც ასახავს ცალკეულ წაკითხვის ან ჩაწერის პროცესებს (ნახ.22.6).



ნახ.22.6. MIM-ში წაკითხვის და ჩაწერის პროცესების
კლასთა იერარქია

მათი წარმოება ხდება დინამიკურად და შეტანის ან გამოტანის დამთავრების შემდეგ ისევ იშლება. ამავდროულად, disk-stream შეიცავს ორივე შემთხვევისათვის საჭირო storage-object ატრიბუტს, რომელიც მიმართავს storage-device კლასის წასაკითხ ან ხელახლაჩასაწერ ეგზემპლარს. read-disk-stream -ში არსებობს დამატებთ read-block-list ატრიბუტი, რომელც აღწერს პოზიციონირების მარკირებასა და აიდენტიფიცირებს მულტიმედიური ობიექტის მომდევნო წასაკითხ ბლოკს. მსგავს როლს ასრულებს write-block-list ატრიბუტი write-disk-stream -ში.

ამჯერად ილუსტრირებული გვაქვს აღნიშნული კლასების ურთიერთმოქმედება და მეთოდების ჩადგმული გამოძახება რასტრული სურათის გამოცემის მაგალითზე.

დავუშვათ, რომ არსებობს car კლასი (ავტომობილი), რომელიც თავისი ეგზემპლარებისათვის განსაზღვრავს ატრიბუტებს image (სურათი) ტიპით captured-image და output_ device (გამოსატანი მოწყობილობა) ტიპით image-pres-device.

ავტომანქანის აღწერას ეკუთვნის აგრეთვე გამოსატანი მოწყობილობა, რომელზეც მანქანის სურათი კარგად უნდა აისახოს. ამის გარდა ეს კლასი შეიცავს ასევე მეთოდს show_image (სურათის ჩვენება), რომელიც ახორციელებს დამახსოვრებული სურათის გამოცემას წინასწარგანსაზღვრულ გამოსატან მოწყობილობაზე. ამის მისაღწევად, საჭიროა show-image მეთოდმა გააგზავნოს present შეტყობინება ობიექტისაკენ, რომელიც წარმოადგენს გამოსატან მოწყობილობას და მას ერთ პარამეტრში დაუსახელოს გამოსატანი სურათი. თუ ასეთი გადაცემის ან გამოძახების ბრძანებები LISP-ენის სახის მეთოდოლოგიის სინტაქსით აიგება, მიმღები ობიექტი და პარამეტრი სპეციფიცირდება, მაშინ show-image-ს შესაბამისი ბრძანება შეიძლება ასე ჩაიწეროს:

(present output-device Image)

output-device ატრიბუტით იდენტიფიცირებული image-pres-device კლასის ეგზემპლარი ასრულებს ამის შემდეგ მის present მეთოდს. მისი ატრიბუტები upper-lef-x, upper-lef-y, width და height უთითებს სურათის რომელი ნაწილი (ამონაჭერი) იხილება. ეს მართკუთხა ამონაჭერი გაითვლება წრფივ კოორდინატებში. ეს შესაძლებელია მხოლოდ captured-image ეგზემპლარზე წვდომის საშუალებით, რომელიც იდენტიფი-ცირებულია image პარამეტრით, რადგან აქ row-major ატრიბუტი მხოლოდ გვეუბნება, რომ რეგისტრაციის მონაცემები ხელმისაწვდომია. ამჯერად შენახული სურათი წასაკითხად იხსნება:

(open-for-read Image [start-offset])

Image-ს საშუალებით დასახელებული capture-image ეგზემპლარი ასრულებს open-for-read მეთოდს. ამ დროს იგი აწარმოებს ახალ read-disk-stream ეგზემპლარს, რომლის სახელს (ობიექტის იდენტიფიკატორს) იგი image-pres-device -ს უკან უგზავნის. თუ შესაძლებელია, სპეციფიკაცია start-offset ემსახურება read-block-list -ის ინიციალიზაციას. გამოსატანი მოწყობილობა აგზავნის ამჯერად (present-ის შესრულების გაგრძელება) ამ ნაკადისთვის (straem) შეტყობინებას წაკითხვის შესახებ:

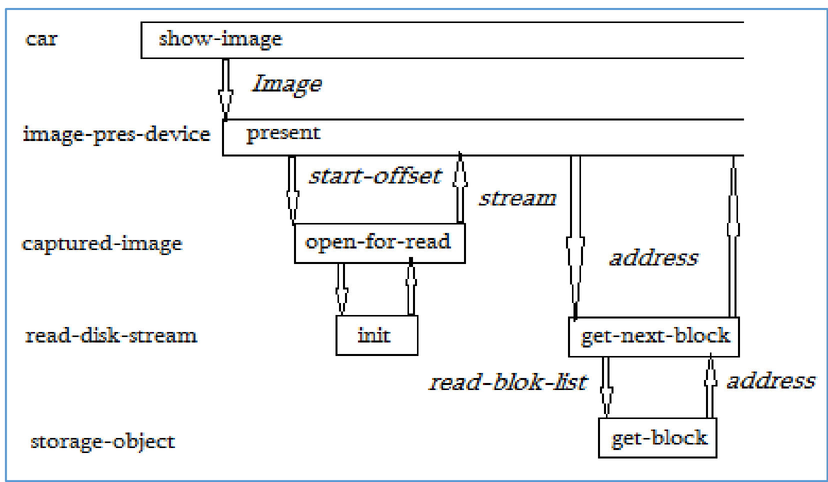
(get-next-block read-disk-stream)

შემდეგ ნაკადი (straem) თვითონ აგზავნის ისევ:

(get-block storage-object read-block-list)

ამ გამოძახებით მიიღება ბუფერის მისამართი, სადაც ინახება სასურველი ბლოკი. შემდეგ ნაკადი ზრდის მაჩვენებელს და აწვდის image-pres-device–ს უკან ბუფერის მისამართს, როგორც get-next-device მეთოდის შედეგს. 22.7 ნახაზი ასახავს ზემოაღწერილ პროცესს გრაფიკულად. მარცხენა მხარეს შემოტანილია კლასის სახელები, რომელთაგან

თითოეული ზუსტად ერთ ეგზემპლარში მონაწილეობს. მართკუთხედები აღნიშნავს მეთოდის შესრულებას, რომლებიც დახრილი (italic) სიმბოლოებით აღწერილი პარამეტრებით გამოიძახება და ასევე კურსივით აღწერილ მნიშვნელობებს აბრუნებს.



ნახ.22.7. MIM-მეთოდების ჩადგმული გამოძახება
სურათის გამოცემის დროს

შემდეგი მსვლელობისას გამომტანი მოწყობილობა გადასცემს წაკითხული ბლოკის შინაარსს ტექნიკურ უზრუნველყოფას და აცნობებს ნაკადს, რომ ბუფერის სახელები კვლავ ხელმისაწვდომია:

(free-block read-disk-stream)

სურათის ყველა ბლოკის წაკითხვისა და გათავისუფლების შემდეგ (close-read read-disk-stream)-ით დაიხურება წაკითხვის პროცესი. ნაკადი შეიძლება კვლავ წაიშალოს.

captured-object კლასი უზრუნველყოფს თავისი ეგზემპლარებისათვის კიდევ სხვა მეთოდებს. make-captured-object-version-ის საშუალებით იწარმოება შენახული მულტიმედია ობიექტის ახალი ვერსია და ირიბად ასევე დაკავებული მეხსიერების მოწყობილობის ახალი ვერსია (storage-device). ძველი და ახალი ვერსიები თავიდან იკავებს დისკზე ერთსა და იმავე ბლოკებს.

დასასრულ, არსებებოს კიდევ delete-captured-object და delete-part-of-captured-object მეთოდები. უკანასკნელი ელოდება პარამეტრებს start-offset და delete-count, რომლებიც მიუთითებს, რომელი ბაიტ - პოზიციიდან და რამდენი ბაიტი უნდა იქნას წაშლილი. ეს ოპერაცია მოითხოვს სიფრთხილეს, ვინაიდან იგი იმავდროულად არ სრულდება სარეგისტრაციო მონაცემების ცვლილებით. ასეთი ოპერაცია არაა გათვლილი სისტემის საბოლოო მომხმარებელზე.

ამგვარად, ORION სისტემა MIM-თან ერთად ასახავს ყველაზე სრულყოფილ წინადადებას დღემდე არსებულ მულტიმედიურ მონაცემთა ბაზების მართვის სისტემებს შორის. იგი მომხმარებელს სთავაზობს დიდ მოქნილობას, რათა არსებული კლასების იერარქია საკუთარი სუბკლასების სპეციალიზებით გაფართოვდეს, რაც დამატებითი დამუშავებისა და წვდომის ოპერაციების გამოყენების შესაძლებლობას იძლევა.

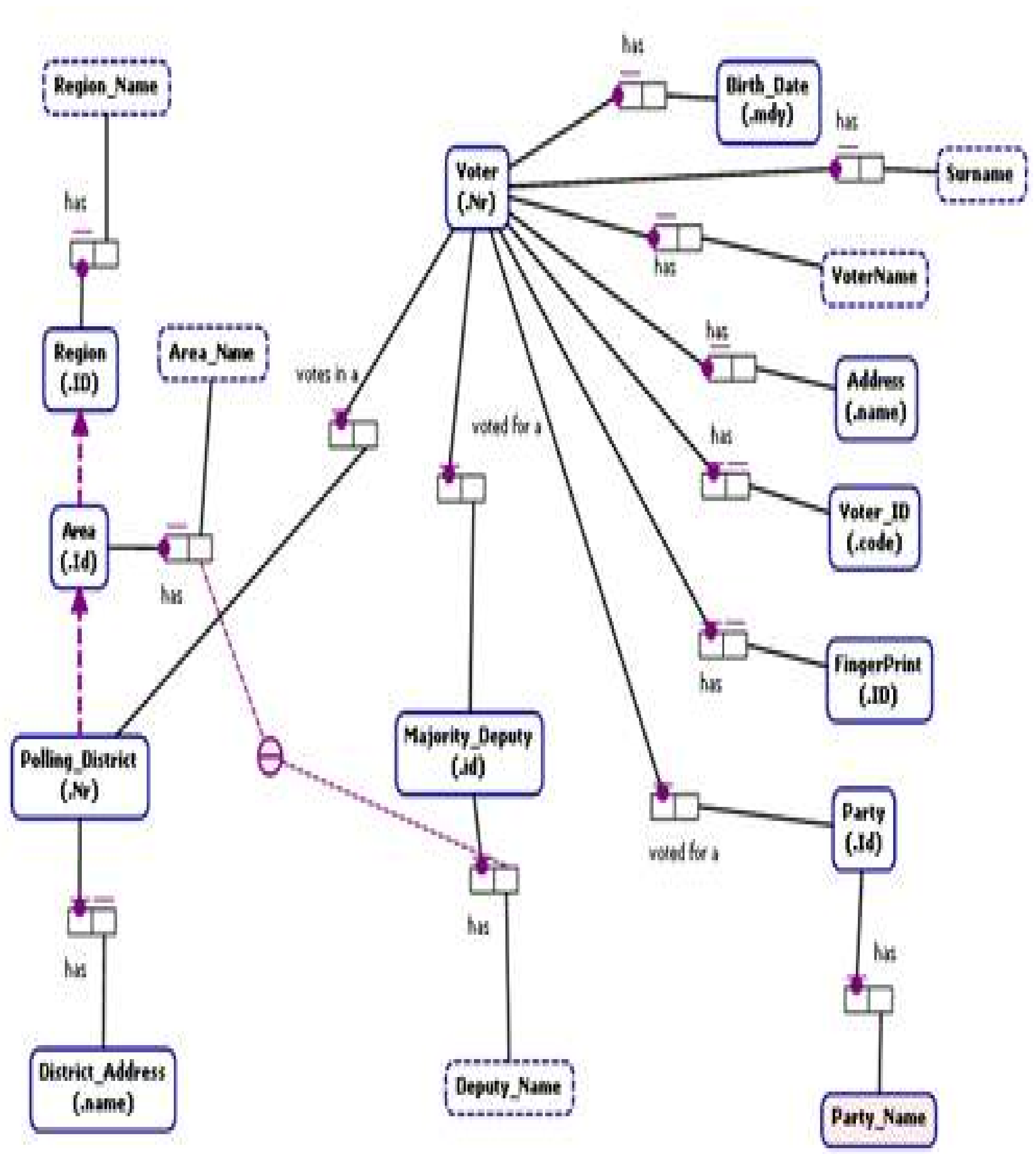
ამგვარად, ობიექტრელაციური და ობიექტორიენტირებული მონაცემთა ბაზების მართვის სისტემების შედარება გვიჩვენებს, რომ ზოგადად არსებობს ერთიანობა მათში ახალი ტიპების განსაზღვრისა და მართვისათვის, ან ობიექტორიენტირებული სისტემის კონტექსტში (მაგალითად, ORION), ან რეალიურ სისტემაში (მაფალითად, SQL / MM).

22.5. მულტიმედიური რელაციური ბაზის ოპტიმალური სტრუქტურის დაპროექტება

ელექტრონული საარჩევნო სისტემის მონაცემთა რელაციური ბაზის კონცეპტუალური სქემის დასაპროექტებლად ჩვენ გამოვიყენეთ ობიექტ-როლური მოდელირების მეთოდი და ინსტრუმენტული საშუალება (იხ. თავი 2). ავტომატიზებული პროცესი მომხმარებლის მიერ სემანტიკუი ფაქტების აღწერით იწყებოდა, რომლის საფუძველზე ფორმირდებოდა კონცეპტუალური სქემა (1-ელი დონე) ORM დიაგრამის სახით (ნახ.22.8), ობიექტებით, პრედიკატებით (კავშირები ობიექტებს შორის), ატრიბუტებით და მათი მნიშვნელობებით.

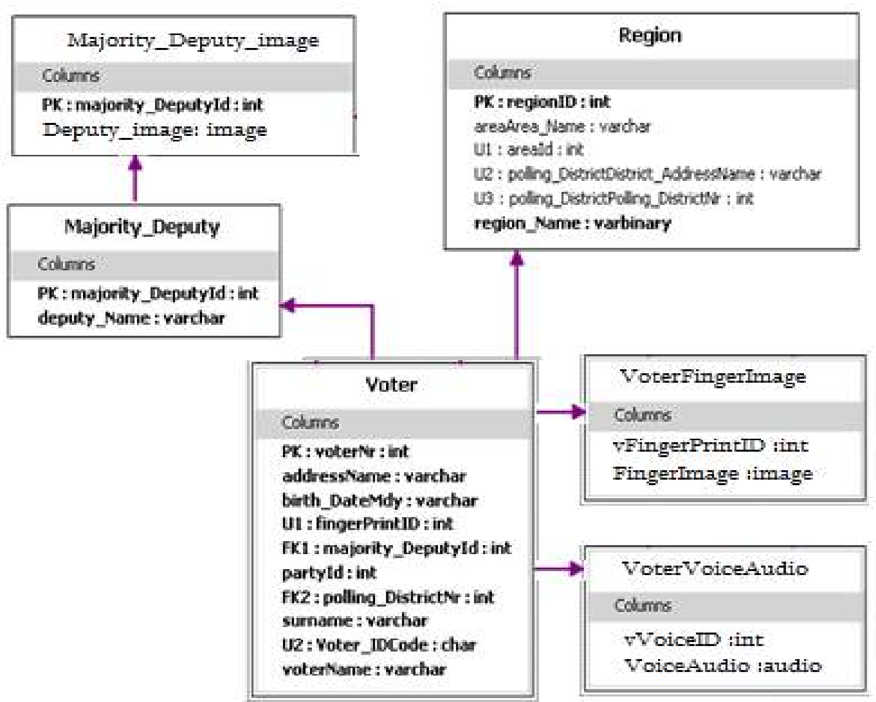
ვინაიდან საპრობლემო სფეროს აღწერა სისტემური ანალიტიკოსის, ან საბოლოო მომხმარებლის მიერ ხდება (ან ორივეს თანამშრომლობით), სემანტიკური ფაქტების სიმრავლე, შემდეგ ORM დიაგრამა და ბოლოს, ERM სქემა შეიძლება იყოს განსხვავებული. ანუ მიიღება ეკვივალენტური კონცეპტუალური სქემები. რომელია მათ შორის ოპტიმალური, ან უკეთესი სისტემის მონაცემთა ბაზის საბოლოო რეალიზაციისათვის? შესაძლებელია თუ არა დაპროექტების წინა სტადიებზე განისაზღვროს უკეთესი ვარიანტი და გამოირიცხოს არაპერსპექტიული ვარიანტები? როგორ შევაფასოთ მოსალოდნელი შედეგი წინასწარ?

ამგვარად, წინამდებარე პარაგრაფში გვინდა წარმოვადგინოთ ამ საკითხის გადაჭრის გზა, მიდგომა და ფორმალიზებული ალგორითმული სქემები.

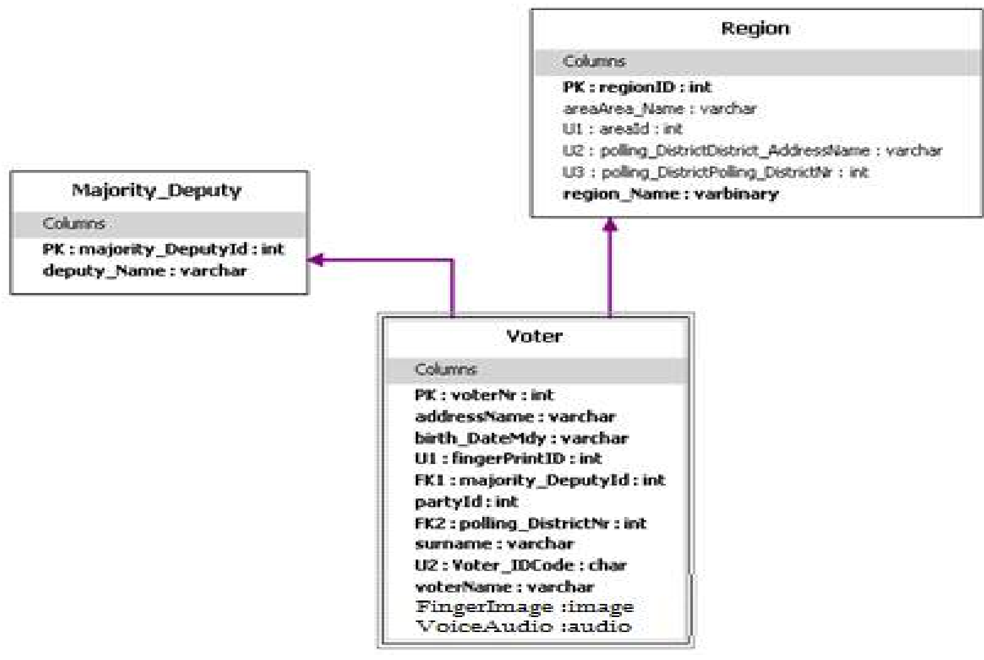


ნახ.22.8. კონცეპტუალური სქემა ORM დიაგრამით

22.9 და 22.10 ნახაზებზე ნაჩვენებია ორი ეკვივალენტური კონცეპტუალური (მე-2 დონე) ERM სქემა. მათ შორის განსხვავება ცხრილების (Tables) რაოდენობაშია, რომლებიც ავტომატიზებული პროცედურების ბოლოს, მაგალითად, SQL Sever-ის DDL-ფაილებად გარდაიქმნება ფიზიკური რეალიზაციის მიზნით.



ნახ.22.9. ER-მოდელის 1-ელი ვარიანტი ექვსი ცხრილით



ნახ.22.10. ER-მოდელის მე-2 ვარიანტი სამი ცხრილით

ER-დიაგრამების შედარება გვიჩვენებს, რომ ORM-მოდელის შეცვლამ გაამარტივა ER-მოდელი, კერძოდ, ექვსი ცხრილის ნაცვლად მივიღეთ სამი. ე.ი. ოპტიმიზაცია, ამ შემთხვევაში, არის ობიექტის ER მოდელის აგების პროცესი ცხრილების რაოდენობის მინიმიზაცია, როცა სემანტიკური მთლიანობა არ ირღვევა (!), ანუ ეს გარკვეული ინფორმაციული სიჭარბის აღმოფხვრაა [73].

ER-მოდელის ცხრილების ოპტიმალური შემადგენლობის განსაზღვრის ამოცანა კავშირშია რელაციური ბაზის სტრუქტურის ნორმალიზაციის საკითხთან [69]. ამ ამოცანის გადაწყვეტა და მისი ალგორითმის შემუშავება მოცემულია ქვემოთ.

დავუშვათ, მოცემულია საპრობლემო სფეროს აღწერის ატრიბუტთა $U = \bigcup_i U_i$ სიმრავლე. მონაცემთა ბაზის $F^0(\overline{A})$ საწყისი სქემა შეიძლება გამოვსახოთ უნივერსალური დამოკიდებულების სახით: $\overline{S}^0 = \{\overline{R} = < U, P >\}$, სადაც $\overline{R}$ დამოკიდებულებათა უნივერსუმია, ხოლო P სემანტიკურ შეზღუდვათა კლასები ან ფუნქციონალურ დამოკიდებულებათა (ფდ) ერთობლიობა [59, 66-68].

ამოცანის მიზანია $\overline{S}^0$ სქემის ეკვივალენტური $\overline{S}$ სქემის კონსტრუირება $\overline{S} = \{R_i = < U_i, P_i >\}$, სადაც R_i არის $\overline{R}$ -ის პროექცია, ხოლო P_i ეთანადება ფდ-ის განსაზღვრულ კლასს, მაგალითად, ფდ, სრული-ფდ (სფდ), ტრანზიტული-ფდ (ტფდ), ფსევდოტრანზიტული-ფდ (ფტფდ), მრავალსახა (მსდ), ზოგადი არაფუნქციონალური (ზად).

ნორმალურ ფორმათა (ნფ) თეორიის გამოყენებით ხორციელდება უნივერსუმის მიმდევრობითი დეკომპოზიცია შემდეგი სქემით:

ანფ -> 1ნფ -> 2ნფ -> 3ნფ -> 4ნფ -> 5ნფ -> . ? . -> ბნფ ,

სადაც ანფ - არანორმალიზებული ფორმაა, 1ნფ - პირველი ნფ, . . . , ბნფ - ბინარული ნფ.

ძირითადი სქემის განშტოებაში შეიძლება განვიხილოთ ბოის-კოდის ნფ (3ნფ-დან), პირველი რიგის იერარქიული დეკომპოზიცია (4ნფ-დან), ურთიერთდამოკიდებულებანი (5ნფ-დან) და ა.შ. ნიშანი „ ? " მიუთითებს იმაზე, რომ დამოკიდებულებათა თვისებების კვლევა გრძელდება, მათი შემდგომი ოპტიმიზაციის მიზნით.

დღეისათვის მრავლად არსებობს სხვა სახის ნფ-ებიც, მაგრამ ნორმალიზაციის კლასიკურ თეორიაში ისინი ნაკლებადაა ასახული [59,66].

დამოკიდებულებათა დეკომპოზიციის დროს ბნფ-ები მიიღება სქემის დაპროექტების ცალკეულ ეტაპზე. არადეკომპონირებადი დამოკიდებულებისთვის კი საჭიროა ხელოვნურად ფიქტიური ატრიბუტის (ნატურალურ რიცხვთა სასრული სიმრავლე) შემოტანა. ინფორმაციის სემანტიკური მთლიანობის უზრუნველყოფა დეკომპონირებულ დამოკიდებულებათა შორის ხორციელდება ინდექსური კავშირების დუბლირების საშუალებით, ე.ი. შემოტანილია გარკვეული სიჭარბე. რაც მაღალია ნფ-ის რიგი, მით ნაკლებია ინფორმაციული და მეტია ინდექსური სიჭარბე:

1nf->2nf->. . . ->bnf, 1nf -> 2nf ->. . . ->bnf,

V1-inf>=V2-inf>=...>=Vb_inf, V1-ind<=V2-ind<=...<=Vb-ind

აქედან გამომდინარე, დაისვა ოპტიმალური სიჭარბის განსაზღვრის კომპრომისული ამოცანა განახლების დროის მინიმიზაციის მოთხოვნით. ამოცანის გადაწყვეტის შედეგად შესაძლებელი გახდა მონაცემთა ბაზის სქემის დამოკიდებულებათა ოპტიმალური ნფ-ების დადგენა, რის შემდეგაც ავტომატიზებულად დაპროექტდება ნორმალურ ფორმათა სტრუქტურები.

დავუშვათ, რომ მოცემულია სემანტიკურად თავსებადი ფდ-ების სიმრავლე:

$$\begin{cases} R_1(k_1, k_2, \ldots k_{n1}, A_1, A_2, \ldots, A_{a1}) \\ R_2(k_1, k_2, \ldots k_{n2}, B_1, B_2, \ldots, B_{a2}) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ R_l(k_1, k_2, \ldots k_{nl}, Z_1, Z_2, \ldots, Z_{al}) \end{cases} \tag{22.1}$$

სადაც k ატრიბუტების გასაღებური, ხოლო A-:-Z არაგასაღებური ნაწილებია.

სემანტიკური მთლიანობა შედეგია იმ ფაქტისა, რომ სიმრავლე მიღებულია ერთი უნივერსუმის დეკომპოზიციით. თუ ჩავთვლით, რომ

$$k_1, k_2, \ldots k_{n1} \supseteq k_1, k_2, \ldots k_{n2} \supseteq k_1, k_2, \ldots k_{nl},$$

(1) სისტემის კომპოზიციით, მაშინ შესაძლებელია ერთი შედარებით დაბალი ნფ-ის მიღება:

$$R(k_1 \ldots k_{n1}, A_1 \ldots A_{a1}, B_1 \ldots B_{a2}, \ldots Z_1 \ldots Z_{al}) \tag{22.2}$$

დავუშვათ აგრეთვე, რომ წინასწარ ცნობილია R_i-ის ცვლილების რაოდენობა μ_i დროის განსაზღვრულ ინტერვალში და მართებულია შემდეგი მოწესრიგება.

$$\mu_1 \geq \mu_2 \geq \ldots \geq \mu_l$$

(1) და (2) გამოსახულებებისთვის განახლებათა მოცულობები შესაბამისად გამოითვლება შემდეგნაირად:

$$Q_{dec} = \sum_{i=1}^{l} \mu_i * (n_i + a_i) \text{ da } Q_{com} = \mu_1 * (n_1 + \sum_{j=1}^{l} (a_j - r))$$

სადაც r ატრიბუტების ის რაოდენობაა, რომლითაც სრულდება შეერთების („Join“) ოპერაცია. შემდგომში შეიძლება იგნორირება.

თუ დავუშვებთ, რომ (1) და (2) ნფ-ებს შორის არსებობს შუალედური ნფ, მაშინ მისთვის განახლებათა მოცულობა შეადგენს:

$$Q = \sum_{j=1}^{s} \mu_j * (n_j + \sum_{k=1}^{l} a_k)$$

სადაც S ფდ-ების რაოდენობაა შუალედურ ნფ-ში. მართებულია შემდეგი უტოლობა:

$$\mu_1 *(n_1 + \sum_{k=1}^{l} a_k) \geq \ldots \geq \sum_{j=1}^{s} \mu_j *(n_j + \sum_{k=1}^{l} a_k) \geq \ldots \geq \sum_{i=1}^{s} \mu_i *(n_i + a_i) \quad (22.3)$$

სადაც უტოლობის მარცხენა მხარე ეთანადება (i-1)ნფ-ს, ნაპირა მარჯვენა მხარე - (i+1)ნფ-ს, ხოლო ცენტრალური - i ნფ-ს, სადაც $i >= 4$.

გავაანალიზოთ დეტალურად ორი მოსაზღვრე ნფ, მაგალითად, i და i+1. (3)_დან შეიძლება მივიღოთ:

$$\sum_{j=1}^{s} \mu_j n_j + \sum_{j=1}^{s} \sum_{k=1}^{l} \mu_j a_k \geq \sum_{i=1}^{l} \mu_i n_i + \sum_{i=1}^{l} \mu_i a_i \quad (22.4)$$

აქედან მართებულია შემდეგი გამოსახულებანი:

$$\sum_{i=1}^{l} \mu_i n_i - \sum_{j=1}^{s} \mu_j n_j = \sum_{i=s+1}^{l} \mu_i n_i \quad (22.5)$$

$$\sum_{j=1}^{s} \sum_{k=1}^{l} \mu_j a_k - \sum_{i=1}^{l} \mu_i a_i = \sum_{j=1}^{s} \sum_{k=1, j\neq k}^{l} \mu_j a_k - \sum_{i=s+1}^{l} \mu_i a_i \quad (22.6)$$

თუ (5) და (6) ტოლობათა მარჯვენა ნაწილებს ჩავსვამთ (4)-ში, მივიღებთ:

$$\sum_{j=1}^{s} \sum_{k=1, j\neq k}^{l} \mu_j a_k \geq \sum_{i=s+1}^{l} \mu_i n_i + \sum_{i=s+1}^{l} \mu_i a_i \quad (22.7)$$

უტოლობის ორივე მხარე გავყოთ $\sum_{i=s+1}^{l} \mu_i a_i$ - ზე, გვექნება:

$$\frac{\sum_{j=1,k=1}^{s} \sum_{j\neq k}^{l} \mu_j a_k}{\sum_{i=s+1}^{l} \mu_i a_i} \geq \frac{\sum_{i=s+1}^{l} \mu_i n_i}{\sum_{i=s+1}^{l} \mu_i a_i} + \frac{\sum_{i=s+1}^{l} \mu_i a_i}{\sum_{i=s+1}^{l} \mu_i a_i} \quad (22.8)$$

ვინაიდან $[1:l] = [1:s] \cup [s+1:l]$ ამიტომ :

$$\frac{\sum_{j=1}^{s} \sum_{k=1, j\neq k}^{l} \mu_j a_k}{\sum_{i=s+1}^{l} \mu_i a_i} = \frac{\sum_{j=1}^{s} \sum_{k=1, j\neq k}^{s} \mu_j a_k}{\sum_{i=s+1}^{l} \mu_i a_i} + \frac{\sum_{j=1}^{s} \mu_j \sum_{k=s+1}^{l} a_k}{\sum_{i=s+1}^{l} \mu_i \sum_{i=s+1}^{l} a_i}$$

ამგვარად, (8)-დან მივიღებთ ვედეკინდ-სურგულაძის მოდელს [69]:

$$\frac{\sum_{j=1}^{s}\sum_{k=1, j\neq k}^{l}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i}+\frac{\sum_{j=1}^{s}\mu_j}{\sum_{i=s+1}^{l}\mu_i}\geq\sum_{i=s+1}^{l}\frac{n_i}{a_i}+1 \qquad (22.9)$$

სადაც $l \geq 2, s \geq 1$ და $l > s$.

პრაქტიკაში ხშირად გამოიყენება შემთხვევა, როცა l=2 და s=1, მაშინ (9) იღებს შემდეგ სახეს:

$$\frac{\mu_1}{\mu_2}\geq\frac{n_2}{a_2}+1 \qquad (22.10)$$

რაც, როგორც ცნობილია, არის ვონგ-ვედეკინდის მოდელი [82].

იგი არის (22.9) გამოსახულების კერძო შემთხვევა. (22.10)-ის გამოყენების დიაპაზონია მე-3ნფ-მდე, ხოლო (22.9)-ისა მთელი დიაპაზონი ნფ-ებისა, ამგვარად იგი უნივერსალურია.

ახლა გამოვიკვლიოთ შემთხვევა, როდესაც კორტეჟის არაგასაღებური ატრიბუტების მნიშვნელობათა ცვლილების სიხშირე მაღალია, ხოლო გასაღებურისა _ დაბალი. დავუშვათ, l=2, s=1 და მოცემულია რელაციათა სქემები:

$$\begin{cases} R_1(k_1,k_2,\ldots k_{n1},A_1,A_2,\ldots,A_{a1}) \\ R_2(k_1,k_2,\ldots k_{n2},B_1,B_2,\ldots,B_{a2}) \\ \\ R_{12}(k_1,k_2,\ldots k_{n1},A_1,A_2,\ldots,A_{a1},B_1,B_2,\ldots,B_{a2}) \end{cases}$$

რომლებშიც მართებულია შემდეგი პირობები:

$$k_1,\ldots k_n \supseteq k_1,\ldots k_{n2} \text{ da } \mu_1 > \mu_2 \qquad (22.11)$$

(3.9)-დან გამომდინარე, მოცემული R_1, R_2 და R_{12} სქემებისათვის, გასაღებურ ატრიბუტთა მნიშვნელობების ცვლილების მაღალი სიხშირის დროს მიზანშეწონილია R_1 da R_2 დამოკიდებულებათა კომპოზიცია R_{12}- ში, თუ სრულდება პირობა:

$$\mu_1(n_1+a_1)+\mu_2(n_2+a_2)>\mu_1(n_1+a_1+a_2).$$

აქედან გამომდინარეობს, რომ:

$$\frac{n_2}{a_2}>\frac{\mu_1}{\mu_2}-1.$$

თუ განიხილება არაგასაღებური ატრიბუტების მნიშვნელობათა ცვლილება გასაღებური ატრიბუტების ცვლილების გარეშე, მაშინ მართებულია შემდეგი გამოსახულება:

$$\mu_1 a_1 + \mu_2 a_2 > \mu_1 (a_1 + a_2).$$

აქედან გამომდინარეობს, რომ:

$$\mu_2 > \mu_1.$$

რაც ეწინააღმდეგება (3.11)-ს.

ამგვარად, დამოკიდებულებათა სქემები, რომლებისთვისაც დომინირებადია არაგასაღებურ ატრიბუტთა ნაწილის ცვლილება, მიზანშეწონილია გამოისახოს მაღალი რიგის ნფ-ებით.

- სქემის გარდასახვა კონცეპტუალურ დონეზე შეიძლება გამოყენებულ იყოს იმისათვის, რომ უფრო ნათელი გახდეს კონცეპტუალური მოდელი ან გაუმჯობესდეს მონაცემთა ბაზის აპლიკაციის ხარისხი;

- მონაცემთა ბაზის დამოკიდებულებანი უნდა წარმოდგენილი იქნეს სხვადასხვა რიგის ნორმალური ფორმებით (3ნფ-:-ბნფ), განახლების სიხშირესა და კავშირების ტიპებზე დამოკიდებულებით მოცემულ კონტექსტში;

- მოცემული μ--თვის შეიძლება (3.9) გამოსახულებით დადგინდეს მოსახერხებელი (ოპტიმალური) ნფ-ები;

- თუ დამოკიდებულებათა გასაღებური ატრიბუტების ცვლილებების სიხშირე მაღალია, მაშინ მათთვის სასურველია დაბალი რიგის ნფ-ების გამოყენება, ხოლო თუ არაგასაღებურ ატრიბუტთა ცვლილების სიხშირე დომინირებადია, მაშინ - შედარებით მაღალი რიგის ნფ-ებისა.

22.6. მონაცემთა ბაზების ახალი ტექნოლოგიები

22.6.1. დოკუმენტორიენტირებული მონაცემთა ბაზა

დოკუმენტორიენტირებული მონაცემთა ბაზა (Document-Oriented Database) არის მონაცემთა ბაზების მართვის სისტემა (მბმს), რომელიც გამოიყენება დოკუმენტების (მონაცემთა იერარქიული სტრუქტურების) შესანახად და რეალიზებულია NoSQL მიდგომის საშუალებით [117,118].

დოკუმენტ-ორიენტირებული მბმს-ას საფუძვლად უდევს დოკუმენტების საცავი (document Store), რომელსაც აქვს ხის სტრუქტურა. ხის სტრუქტურა იწყება ფესვური კვანძით და შეიძლება შეიცავდეს რამდენიმე შიგა კვანძს და ფოთლების კვანძს.

ფოთლების კვანძი შეიცავს მონაცემებს, რომლებიც დოკუმენტის დამატების დროს შეიტანება ინდექსებში, რაც უზრუნველყოფს, რთული სტრუქტურების შემთხვევაშიც კი, მოიძებნოს გზა საჭირო მონაცემებისკენ.

მოთხოვნის საფუძველზე API (Application Programming Interface) ახორციელებს დოკუმენტების და მათი ნაწილების ძებნას.

დოკუმენტები შეიძლება იყოს ორგანიზებული (დაჯგუფებული) კოლექციებში. ეფექტური ინდექსირების მიზნით სასურველია კოლექციებში მსგავსი სტრუქტურების დოკუმენტების გაერთიანება.

დოკუმენტორიენტირებული მონაცემთა ბაზები გამოიყენება შინაარსის მართვის სისტემებში (CMS - Content Management System), საგამომცემლო საქმეში, დოკუმენტების საძიებო სისტემებში და სხვ. ასეთი ბაზების მართვის სისტემების მაგალითებია: MongoDB, CouchDB, Couchbase, MarkLogic, eXist, IBM Lotus Notes და სხვ.[117-120].

დოკუმენტორიენტირებული მონაცემთა ბაზების მთავარი ცნებაა „დოკუმენტი", რომელიც განისაზღვრება როგორც მონაცემთა ინკაფსულაცია ინფორმაციის კოდირების სტანდარტული ფორმატებისა და მეთოდების გამოყენების საფუძველზე. ასეთი ფორმატებია: XML, JSON, BSON, YAML. ზოგ შემთხვევაში შესაძლებელია PDF, Ms Office და მსგავსი დოკუმენტების ბინარული ფორმატით შენახვაც [121].

დოკუმენტი მონაცემთა ბაზაში მისამართდება უნიკალური გასაღების საშუალებით. ხშირად ეს გასაღები მარტივი სტრიქონია, რომელიც შეიძლება იყოს URI (Unified Resource Identifier) ან გზა (path) დოკუმენტამდე. ასეთი გასაღების ან მისი ინდექსის საშუალებით მოიძებნება დოკუმენტი ბაზაში და შესაძლებელია მისი სწრაფად ამოღება.

დოკუმენტური ბაზის დამახასიათებელია სიტყვა-გასაღების (მნიშვნელობის-გასაღების) მარტივად განსაზღვრა მოთხოვნილი დოკუმენტების მოსაძებნად. მონაცემთა ბაზას აქვს სპეციალური API ანუ მოთხოვნების ენა, რომელიც უზრუნველყოფს დოკუმენტების მიღებას მათი შინაარსის (content) მიხედვით.

API არის აპლიკაციების დაპროგრამების ინტერფეისი. იგი შეიცავს მზა კლასების, პროცედურების, ფუნქციების, სტრუქტურებისა და კონსტანტების ერთობლიობას, რომელსაც წარმოადგენს დანართი (ბიბლიოთეკა ან სერვისი) ან ოპერაციული სისტემა. იგი გამოიყენება პროგრამისტების მიერ აპლიკაციის შექმნისას.

ქვემოთ ჩვენ დეტალურად შევეხებით NoSQL მონაცემთა ბაზების საკითხებს და კონკრეტულად, MongoDB პაკეტის შესაძლებლობებს.

22.6.2. გრაფულ-ორიენტირებული მონაცემთა ბაზა

გრაფული მონაცემთა ბაზის მართვის სისტემებისთვის დამახასიათებელია მონაცემთა გრაფული მოდელი, ანუ ინფორმაციის შენახვა ხდება არა „ცხრილებით" (tables) და ატრიბუტებით (როგორც რელაციური ბაზებში), არამედ გრაფული სტრუქტუ-რებით, ანუ კვანძებით (nodes) და მათ შორის კავშირებით (გრაფის წიბოები - edges). ასეთი კავშირები შეიძლება რამდენიმე დონეს მოიცავდეს (ღრმა კავშირები) და ისინი ძალზე აქტუალურია დიდი სოციალური პროექტების (ქსელების), ბიოინფორმატიკის, რთული მარშრუტების, სემანტიკური ქსელის (Web, HTTP გვერდებით) და სხვა სფეროს ამოცანების გადასაწყვეტად.

გრაფული მონაცემთა ბაზა არის ქსელური მოდელის (ან RDF-მოდელის) რეალიზაციის ნაირსახეობა [117]. მისი კონცეფცია ჯერ კიდევ 80-იან წლებში გამოჩნდა, ხოლო პირველი გრაფული რეალიზაცია 2007 წელს, Neo4j სისტემის სახით. დღეისთვის უკვე არსებობს რამდენიმე ათეული ასეთი ბაზებისა, მაგალითად: ArangoDB, OrientDB, MarkLogic, Oracle Spatial and Graph და სხვ.

RDF (Resource Description Framework) - რესურსის აღწერის გარემო შეიქმნა WWW კონსორციუმის მიერ როგორც მონაცემთა აღწერის მოდელი - მეტამონაცემებით.

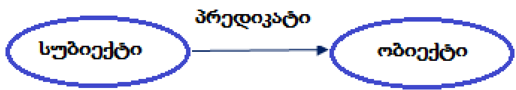
რესურსი RDF-ში შეიძლება იყოს ნებისმიერი არსი, როგორც ინფორმაციული (მაგალითად, ვებგვერდი, გამოსახულება), ასევე არაინფორმაციული (მაგალითად, ადამიანი, მანქანა ან აბსტრაქტული ცნება). რესურსის შესახებ გამონათქვამის მტკიცებას აქვს სამეულის (Triple Stores - სამადგილიანი შენახვა) სახე:

„სუბიექტი — პრედიკატი — ობიექტი"

მაგალითად, მტკიცება „დოლიძე არის პროფესორი", RDF-ის ტერმინოლოგიით ჩაიწერება ასე:

სუბიექტი — „დოლიძე", პრედიკატი — „აქვს თანამდებობა", ობიექტი — „პროფესორი".

გრაფიკულად იგი 22.11 ნახაზზეა მოცემული



ნახ.22.11. RDF-ის სამეული

ამგვარად, RDF-ის მტკიცებულებები (ფაქტები) ქმნის ორიენტირებულ გრაფს, რომელშიც კვანძებია სუბიექტები და ობიექტები, ხოლო წიბოები - მათ შორის მიმართებები. RDF არის მონაცემთა აბსტრაქტული მოდელი, ანუ იგი აღწერს მოცემულ სტრუქტურას, დამუშავების ხერხებს და მონაცემთა ინტერპრეტაციებს.

ქვემოთ მოცემულია NoSQL ტიპის ოჯახის ზოგიერთი პოპულარული მბმს, რომლებიც გრაფულ ან ჰიბრიდულ ბაზებს მიეკუთვნება [118, 122].

AllegroGraph – დამუშავებულია W3C სტანდარტით Common Lisp ენაზე Triple Store (პრედიკატული სამეული) სახით მონაცემთა RDF მოდელისთვის, Windows, Linux და Mac OSX -თვის, კლიენტის ინტერფეისებით: Java, Python, Ruby, Perl, C#, Clojure და Common Lisp.

ArangoDB – დაწერილია C++ და JavaScript ენებზე მულტიმოდელური ბაზის სახით. გამოიყენება როგორც key/value, document, და graph data ბაზები და აქვთ ერთი საერთო მოთხოვნების ენა. 2011 წლამდე გამოდიოდა AvocadoDB სახელით.

DataStax Enterprise Graph (DSE) – აგებულია Java ენაზე Web-საიტებისა და მობილური ტექნიკისთვის. სერვერის მხარეს Backend-ის სახით იყენებს Apache Cassandra-ს. შეუძლია დაამუშაოს წამში პეტაბაიტი ინფორმაცია და ერთდროულად მოემსახუროს ათას მომხმარებელს. ბაზა განაწილებულია კვანძების კლასტერებში და აქვს მასშტაბირებადი არქიტექტურა [117]. მასში ჩადგმულია OLAP ანალიზის და გრაფში ძებნის მხარდაჭერა. აქვს უსაფრთხოების დამატებითი პარამეტრები კონფიდენციალური მონაცემებისთვის.

MarkLogic – მულტიმოდელური ბაზაა სემანტიკური გრაფით და RDF სამეულით. ინახავს დოკუმენტებს JSON (JavaScript Object Notation) და XML ფორმატებში. აქვს ჩადგმული საძიებო სისტემა, ACID მახასიათებლების მქონე ტრანზაქციები – მაღალი წვდომადობა და ავარიული აღდგენის უნარი, გარანტირებული უსაფრთხოება, მოქნილობა და მასშტაბურობა.

Neo4j – კომპანია Neo Technology-ის პროდუქტი, დამუშავებულია java ენაზე და ერთ-ერთი ყველაზე პოპულარული გრაფული ბაზაა. აპლიკაციების დაპროგრამების ინტერფეისი მონაცემთა ბაზისთვის რეალიზებულია მრავალი ენისთვის, მათ შორის: Java, Python, Ruby, PHP და სხვ.

OrientDB – გრაფული- და დოკუმენტ-ორიენტირებული ბაზის სისტემაა, დამუშავებულია Java ენაზე Orient Technologies LTD ფირმის მიერ Windows, Linux, Mac და სხვა ოპერაციული სისტემებისთვის. მოთხოვნების ენისათვის აქვს SQL-ის მხარდაჭერა (ამიტომაც მას NewSQL ბაზასაც მიაკუთვნებენ). იგი არ იყენებს JOIN ოპერაციას. მის მაგივრად აქვს სუპერ-სწრაფი მუდმივი მიმთითებლები ჩანაწერებს შორის, რომლებიც გრაფული ბაზებისთვისაა დამახასიათებელი. ეს უზრუნველყოფს ჩანაწერების ცალკეული ან მთლიანი ხეების და გრაფების გადასინჯვას რამდენიმე მილიწამის ფარგლებში.

Stardog – არის კროსპლატფორმული, სემანტიკური გრაფული ბაზის სიტემა, რეალიზებული Java ენაზე, RDF-ის და OWL (Web Ontology Language)-ის მხარდაჭერით [32]. OWL ენით აღიწერება კლასები და მიმართებები მათ შორის, რომლებიც დამახასიათებელია ვებ-დოკუმენტებისა და აპლიკაციებისთვის.

22.6.3. NewSQL მონაცემთა ბაზები

NewSQL არის თანამედროვე რელაციური ბაზების მართვის სისტემების კლასი, რომელიც გაფართოებული ფუნქციონალობის საფუძველზე უზრუნველყოფს NoSQL ბაზების მსგავს მწარმოებლურობას ტრანზაქციების ოპერატიული დამუშავებისათვის [123]. ამასთანავე იგი ინარჩუნებს ACID პრინციპებს.

განსაკუთრებით საყურადღებოა აქ მონაცემთა შენახვის პრინციპულად ახალი პლატფორმების შექმნა, რომლებიც ორიენტირებულია განაწილებული არქიტექტურის და მრავალნაკადურ სისტემებზე.

ასეთი მიდგომის ერთ-ერთი პოპულარული მონაცემთა ბაზაა MemSQL (რელაციური ბაზა ოპერატიული მეხსიერებით [In-Memory Storage] Linux ოპერაციული სისტემისთვის) [124] (ნახ.22.12). იგი იყენებს SQL ენას. კოდის გენერაცია სრულდება C++ ენაზე. ანუ MemSQL სერვერზე გაგზავნილი მოთხოვნები გარდაიქმნება C++ -ზე და კომპილირდება GCC-ს დახმარებით.

ნახ.22.12

MemSQL თავსებადია MySQL-თან. აპლიკაციები შეიძლება შეერთდეს MemSQL სისტემასთან ODBC/JDBC სტანდარტებით, ასევე დრაივერებით და MySQL-ის მომხმარებლებით.

გარდა ზემოთ აღნიშნულისა, ლიტერატურულ წყაროებში განიხილავენ NewSQL-ის ტიპის შემდეგ ბაზებს: NuoDB, VoltDB, OrientDB, Clustrix, ScaleDB, dbShards და სხვ [125] (ნახ.22.13).

ნახ.22.13

22.14 ნახაზზე მოცემულია მონაცემთა ტრადიციული SQL ბაზების, NoSQL და NewSQL ბაზების შედარება ოთხი ძირითადი თვისებით (Properties) [124].

COMPARISON :

PROPERTIES	TRADITIONAL SQL	NOSQL	NEWSQL
ACID PROPERTY	✓	✗	✓
IN MEMORY DB	✗	✓	✓
BIG DATA	✗	✓	✓
RDBMS	✓	✗	✓

ნახ.22.14

როგორც ვხედავთ, NewSQL მონაცემთა ბაზა აერთიანებს ტრადიციული (რელაციური) და NoSQL (არარელაციური) ბაზების საუკეთესო თვისებებს, ამიტომაც იგი ძალზე პერსპექტიულია სამომავლო პროექტებისათვის.

ერთ-ერთი საინტერესო გადაწყვეტა ამ თვალსაზრისით არის MySQL და NoSQL ბაზების ინტეგრაციის საკითხი, რომელსაც მომდევნო პარაგრაფში განვიხილავთ.

22.6.4. NoSQL მონაცემთა ბაზა MySQL ბაზასთან ერთად

ერთ-ერთი აქტუალური მიმართულება მონაცემთა რელაციური ბაზების მწარმოებლურობის (ეფექტიანობის) ამაღლების მიზნით არის NoSQL ბაზების პრინციპების ჩანერგვა რელაციურ სისტემებში. ამის კონკრეტული მაგალითია Oracle კორპორაციის მიერ C-ენაზე დაწერილი კროსპლატფორმული პროდუქტი InnoDB, რომელიც გამოიყენება MySQL მონაცემთა ბაზების მართვის სისტემაში (5.5 ვერსიიდან) [118].

InnoDB არის მონაცემთა საცავი (database engine, storage engine), ბაზების მართვის სისტემის პროგრამული კომპონენტი, რომელიც გამოიყენება მონაცემთა ბაზის შენახვის, განახლების და ინფორმაციის ძებნის (create, update, delete, read) მექანიზმების სამართავად.

ამგვარად, Oracle-მ მოახერხა MySQL-ის (v.5.6) InnoDB-ში NoSQL-ის შესაძლებლობების დამატებით 9-ჯერ აემაღლებინა ტრანზაქციათა დამუშავების ეფექტურობა [119].

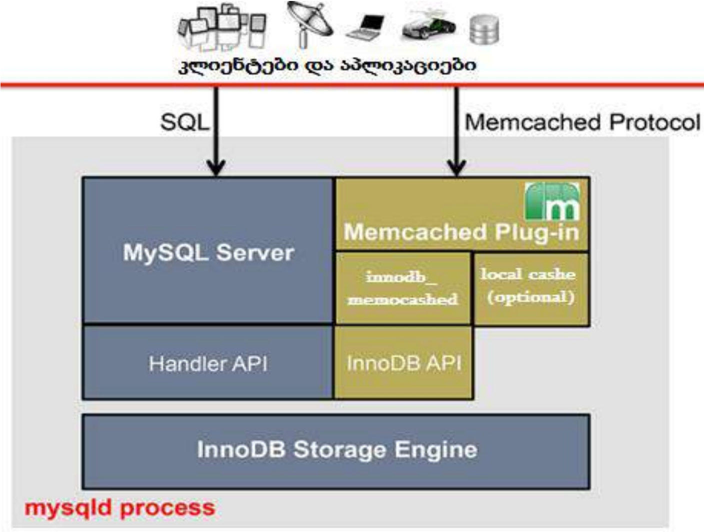
აქ რეალიზებულია NoSQL-ის ინტერფეისები MySQL და MySQL Cluster მონაცემთა ბაზებთან. ისინი სრულიად უვლის გვერდს SQL შრეს, მის სინტაქსურ ანალიზს და ოპტიმიზაციას. მონაცემები შეიძლება უშუალოდ ჩაიწეროს MySQL-ის ცხრილებში 9-ჯერ უფრო სწრაფად, ACID პრინციპების დაცვით.

როგორაა NoSQL რეალიზებული MySQL-ში ?

MySQL 5.6 უზრუნველყოფს მარტივ, პირდაპირ *გასაღები-მნიშვნელობა* ურთიერთქმედებას InnoDB-ს მონაცემებთან ცნობილი API-ის ქეშმეხსიერების დახმარებით.

ქეშმეხსიერება (Memcached) წარმოადგენს საერთო დანიშნულების განაწილებული მეხსიერების კეშირების სისტემას [118]. იგი ხშირად გამოიყენება მონაცემთა ბაზების მქონე დინამიკური ვებსაიტების დასაჩქარებლად მონაცემებისა და ობიექტების კეშირების დახმარებით ოპერატიულ მეხსიერებაში. აქ მნიშვნელოვნად მცირდება, მაგალითად, აპლიკაციის ინტერფეისით ინფორმაციის წაკითხვა მონაცემთა გარე წყაროებიდან. API შესაძლებლობას იძლევა აგრეთვე Memcached-ისა და კლიენტების სტანდარტული ბიბლიოთეკით (C++, Java, Python, PHP და სხვა ენებისათვის) კეშირებულ იქნას მონაცემები ხელმისაწვდომი სერვერების ოპერატიულ მეხსიერებაში და შემდგომ მრავალჯერ იქნას გამოყენებული.

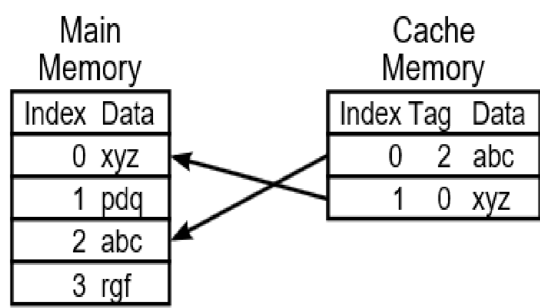
სისტემის რეალიზაცია მოცემულია 22.15 ნახაზზე [119].



ნახ.22.15

MySQL-ის გაფართოებულ ვერსიაში NoSQL შესაძლებლობებით კლიენტების და აპლიკაციების მოთხოვნები შედის სისტემაში SQL ენაზე (პირდაპირ MySQL Server-ზე) ან Memcached პროტოკოლით (InnoDB-ს გავლით).

მეორე შემთხვევაში აპლიკაციებისთვის გარანტირებულია მაღალი სისწრაფის კითხვა/ჩაწერის ოპერაციების შესრულება, ვინაიდან მონაცემები ინახება ქეშმეხსიერებაში (ნახ.22.16).



ნახ.22.16

ქეშმეხსიერება (Cache memory) არის აპარატურული ან პროგრამული ნაწილის კომპონენტი, რომელიც იქმნება მონაცემთა დროებით შესანახად, მათი სწრაფად, მრავალჯერადი გამოყენების მიზნით. აქ ინახება წინასწარ გათვლილი ან ამორჩეული მონაცემთა ერთობლიობა, რომელთა ხშირად გამოყენების ალბათობა მაღალია. ძირითად მეხსიერებასთან შედარებით ქეშმეხსიერება არაა დიდი ზომის.

22.6.5. MariaDB ბაზა MySQL-ის Open Source ალტერნატივა

მონაცემთა ბაზა MariaDB არის MySQL-ის ახალი ალტერნატიული free-ვარიანტი, რომელიც შექმნა მაიკლ ვიდენიუსმა 2009 წელს. იგი იყო ავტორი MySQL-ისაც (1995), რომელიც Oracle კორპორაციამ შეიძინა 2008 წელს და გახადა იგი კომერციული პროდუქტი [40]. (მ. ვიდენიუსის უფროსი ქალიშვილია მაია -MySQL, ხოლო უმცროსი - მარია).

MariaDB თავსებადია MySQL-თან, უზრუნველყოფს შესაბამისობას API-სთან და MySQL-ის ბრძანებებთან. მასში დამატებულია მონაცემთა საცავის (storage engine) ქვესისტემა XtraDB, რომელიც ცვლის MySQL-ის InnoDB-ს [117, 126]. ეს საცავი მაღალმწარმოებლურია InnoDB-სთან შედარებით, აქვს მეხსიერების მართვის და ინფორმაციის ნაკადების შეტანა/გამოტანის უფრო სრულყოფილი მექანიზმები, რეალიზებულია ტრანზაქციების ACID მოთხოვნები (Atomicity, Consistency, Isolation, Durability) და აქვს MVCC (MultiVersion Concurrency Control - მონაცემთა ბაზასთან წვდომის პარალელური უზრუნველყოფა) არქიტექტურა.

იმისათვის, რომ დავაყენოთ MariaDB მონაცემთა ბაზა, უნდა შესრულდეს შემდეგი პირობები:

1) გამართული უნდა იყოს ლინუქსის ოპერაციული სისტემა ფიზიკურ ან ვირტუალურ მანქანაზე (2 CORE, 2 GB RAM, 20 GB HDD). რაც შეეხება Linux-ის

დისტრიბუციას, არჩევანი დიდია (მაგალითად, CentOS-ს, რომელიც დიდი პოპულარობით სარგებლობს) [127];

2) გამართულ ოპერაციულ სისტემას წვდომა უნდა ჰქონდეს ინტერნეტში (ბაზის დაყენების დროს).

მას შემდეგ რაც ოპერაციული სისტემა გამართულია და ინტერნეტში წვდომაც გვაქვს, შეგვიძლია დავიწყოთ მონაცემთა ბაზის დაყენება (აღნიშნული ინსტრუქცია გათვლილია „CentOS 6 64-bit"-სთვის).

ოპერაციულ სისტემაში შევდივართ „root" მომხმარებლით და ტერმინალში ვწერთ შემდეგ ბრძანებებს [128]:

1) touch /etc/yum.repos.d/MariaDB.repo რაც შექმნის MariaDB.repo ფაილს /etc/yum.remos.d/ დირექტორიაში;

2) vi/etc/yum.repos.d/MariaDB.repo vi ედიტორით გავხსნათ MariaDB.repo ფაილი კლავიატურაზე „i" ღილაკის გამოყენებით გადავიდეთ „insert" რეჟიმში და ფაილში ჩავწეროთ შემდეგი :

- [mariadb]
- name = MariaDB
- baseurl = http://yum.mariadb.org/5.5/centos6-amd64
- gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
- gpgcheck=1

კლავიატურაზე „Esc" ღილაკის გამოყენებით გადავიდეთ ბრძანების რეჟიმში ვწერთ „wq" და ვაწვებით „Enter" ღილაკს, რის შემდეგადაც ჩვენი შეყვანილი ინფორმაცია შეინახება „MariaDB.repo" ფაილში.

3) yum -y install MariaDB MariaDB-server (დაიწყება ბაზის ინსტალაცია).

4) /etc/init.d/mysql start (მონაცემთა ბაზის გაშვება)

5) როდესაც მონაცემთა ბაზა გაეშვება „mysql" ბრძანებით შეგვიძლია შევიდეთ მონაცემთა ბაზის ტერმინალში სადაც გაუშვებს mysql ის ბრძანებებს:

- show databases;
- quit;

6) მას შემდეგ, რაც მონაცემთა ბაზას წარმატებით დავუკავშირდით და ყველაფერმა იმუშავა, აუცილებელია უსაფრთხოების პარამეტრების გამართვა, რისთვისაც ვუშვებთ შემდეგ ბრძანებას და დაკვირვებით გავივლით შემდგომ ეტაპებს:

mysql_secure_installation

/usr/bin/mysql_secure_installation: line 379: find_mysql_client: command not found

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB

SERVERS IN PRODUCTION USE! PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current

password for the root user. If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.
Enter current password for root (enter for none):
OK, successfully used password, moving on...
Setting the root password ensures that nobody can log into the MariaDB
root user without the proper authorisation.
Set root password? [Y/n] Y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
... Success!
Remove anonymous users? [Y/n] y
... Success!
Normally, root should only be allowed to connect from 'localhost'. This
ensures that someone cannot guess at the root password from the network.
Disallow root login remotely? [Y/n] y
... Success!
By default, MariaDB comes with a database named 'test' that anyone can
access. This is also intended only for testing, and should be removed
before moving into a production environment.
Remove test database and access to it? [Y/n] y
- Dropping test database...
... Success!
- Removing privileges on test database...
... Success!
Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.
Reload privilege tables now? [Y/n] y
... Success!
Cleaning up...
All done! If you've completed all of the above steps, your MariaDB
installation should now be secure.
Thanks for using MariaDB!

7) /etc/init.d/mysql restart (კონფიგურაციის გავლის შემდეგ აუცილებელია მონაცემთა ბაზის გადატვირთვა).

8) chkconfig mysql on (სისტემის ჩატვირთვისას მონაცემთა ბაზა ავტომატურად რომ გაეშვას)

9) mysql -u root –p (მონაცემთა ბაზის ტერმინალში შესასვლელად -u პარამეტრით გადავცემთ მომხმარებლის სახელს, ხოლო -p პარამეტრით პაროლს).

22.6.6. MongoDB ბაზა და მისი ინსტალაცია

MongoDB არის დოკუმენტზე ორიენტირებული NoSQL მონაცემთა ბაზა [129]. მისი ინსტალიაციის მიზნით კომპიუტერზე აუცილებელია იგივე წინაპირობები და ეტაპები, რაც MariaDB-სთვის:

1) სისტემაში შევდივართ „root“ მომხმარებლით
2) touch /etc/yum.repos.d/mongodb.repo
3) vi /etc/yum.repos.d/mongodb.repo vi ედიტორის საშუალებით mongodb.repo ფაილში ჩავწრეოთ შემდეგი :

- [mongodb]
- name=MongoDB Repository
- baseurl=http://downloads-distro.mongodb.org/repo/redhat/os/x86_64/
- gpgcheck=0
- enabled=1

4) yum –y install mongo-10gen mongo-10gen-server (მონაცემთა ბაზის დაყენება)
5) service mongod start (მონაცემთა ბაზის სერვისის გაშვება)
6) service mongod status (მონაცემთა ბაზის სტატუსის შემოწმება)
7) mongostat (მონაცემთა ბაზაში მიმდინარე პროცესების ნახვა)

ამით MongoDB-ს დაყენება დამთავრებულია. იმისათვის, რომ მონაცემთა ბაზაში მუშაობა შევძლოთ, ოპერაციული სისტემის ტერმინალზე უნდა ავკრიფოთ mongo და შევიდეთ მონაცემთა ბაზის ტერმინალში, სადაც უშუალოდ mongo-ს ბრძანებების გაშვებას შევძლებთ. mongo (მონაცემთა ბაზის კლიენტი) ბრძანების გაშვებისას ოპერაციული სისტემა ავტომატურად მიმართავს „localhost:27017“ და ცდის ბაზასთან დაკავშირებას. მომდევნო თავებში დეტალურად გავეცნობით ამ ბაზის ფუნქციონირების პრინციპებს და მუშაობის შესაძლებლობებს.

22.6.7. Hadoop – „დიდ მონაცემთა“ ახალი ტექნოლოგია

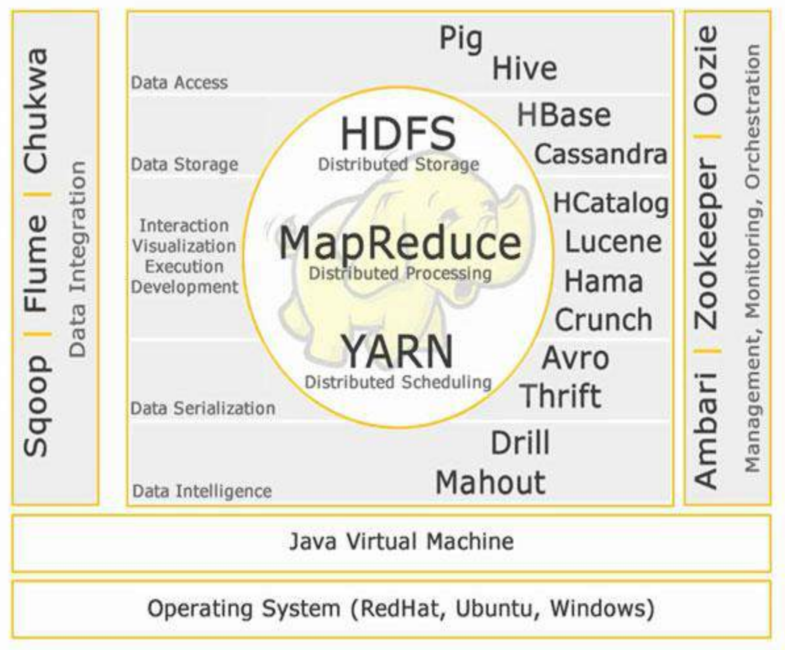
ჩვენ ვცხოვრობთ ინფორმაციის ეპოქაში. 2013 წლისთვის ციფრული სამყაროს ზომა 4.4 ზეტაბაიტი იყო, 2020 წლისთვის კი ნავარაუდებია ინფორმაციის მოცულობის ათმაგი ზრდა, 44 ზეტაბაიტამდე (44 მილიარდი ტერაბაიტი) [130].

ამ მოცულობის ინფორმაციიდან გარკვეული ნაწილი ისეთ კორპორაციებზე მოდის, როგორიცაა ნიუ იორკის საფონდო ბირჟა – 4-5 ტერაბაიტი დღეში [131], Facebook.com – ჯამში 240 მილიარდი ფოტო, თვეში 7 პეტაბაიტი ზრდის მახასიათებლით [132], Ancestry.com – 10 პეტაბაიტი მოცულობის გენეალოგიის ბაზა [133].

ეს ადამიანის შექმნილი მონაცემებია, მაგრამ ბოლო ათწლეულში ტენდენცია შეიცვალა, დღეს უკვე ინფორმაციის უდიდეს ნაწილს ადამიანების ნაცვლად კომპიუტერული ტექნიკა აგენერირებს.

ბოლო წლებში აქტუალური გახდა ტერმინი IOT (Internet Of Things), რაც თავის თავში მოიცავს ყველა იმ აპარატს და კომპიუტერულ ტექნიკას, რაც მიმდინარე დროში დიდი რაოდენობით მონაცემებს აგენერირებს. მაგალითად, მანქანის GPS სისტემები, სხვადასხვა სენსორები და ყველა ის ტექნიკა, რაც ძირითადი ფუნქციონირების პარალელურად წარმოშობს დიდი რაოდენობის დამხმარე ინფორმაციას (metadata).

ამ რაოდენობის მონაცემების დამუშავებას სრულიად განსხვავებული სისტემა სჭირდება არა მხოლოდ რელაციური და არარელაციური ბაზების დონეზე, არამედ იმ სერვერული არქიტექტურის დონეზე, სადაც ვაყენებთ მონაცემთა ბაზებს. დღეისათვის საუკეთესო გამოსავალი დააპროექტა Apache Software Foundation-მა, სახელით Hadoop. Hadoop არის უფასო, ჯავაზე დაფუძნებული პლატფორმა, რომელიც შექმნილია დიდი ზომის მონაცემთა ნაკადის დასამუშავებლად (ნახ.1.33) [37].



ნახ.22.17

Hadoop ეკოსისტემაში სხვადასხვა პროდუქტებია გაერთიანებული, ბირთვად კი სამი ძირითადი კომპონენტი აქვს:

- HDFS (Hadoop Distributed File System) - განაწილებული ფაილური სისტემა მონაცემების შესანახად;
- Map Reduse - მთავარი კომპონენტი განაწილებული გამოთვლების ჩასატარებლად;
- YARN (Yet Another Resource Negotiator) - განაწილებული გარემოს მართვა.

ბოლო ათწლეულში მონაცემთა ბაზების მართვის სისტემების განვითარება ხორციელდებოდა რამდენიმე მიმართულებით.

მთავარი იყო *რელაციური ბაზების*, როგორც უმრავლესი მართვის საინფორმაციო სისტემების ძირითადი კომპონენტის სრულყოფა მწარმოებლურობის თვალსაზრისით. ვითარდებოდა მოთხოვნების სწრაფად დამუშავების ოპტიმიზაციის მეთოდები და ალგორითმები.

მეორე მიმართულება გახდა NoSQL („არარელაციური" ან „არა მხოლოდ რელაციური") ტიპის ბაზების მართვის სისტემების შექმნა და სრულყოფა, განსაკუთრებით „დიდ მონაცემთა" დასამუშავებლად, სადაც რელაციური ბაზები არაეფექტურია, მათი Join და სხვა რელაციური ოპერაციებით ცხრილების დამუშავების პროცესის დიდი დროის გამო. ასეთია, მაგალითად, დიდ კორპორაციულ საცავებში დოკუმენტებთან მუშაობის სისტემები.

მესამე მიმართულება ეხება გრაფული ბაზების შექმნას და განვითარებას. მათთვის დამახასიათებელია ინფორმაციის შენახვა გრაფული სტრუქტურებით და აქტუალურია დიდი სოციალური ქსელების, ბიოინფორმატიკის და სხვა სფეროების ამოცანების გადასაწყვეტად.

ბოლოს, შეიძლება ვახსენოთ ჰიბრიდული, NewSQL მონაცემთა ბაზების მართვის სისტემები, სადაც ინტეგრირდება რელაციური, NoSQL-ის და გრაფული ბაზების კონცეფციები.

ყველა მიმართულება აქტუალურია და სწრაფად ვითარდება მსოფლიოს წამყვან უნივერსიტეტებსა და პროგრამული სისტემების ფირმებში. რომელი ტიპის ბაზას აირჩევს მომხმარებელი, დამოკიდებულია კონკრეტული მართვის საინფორმაციო სისტემის მიზნებსა და ფუნქციებზე.

22.7. მონაცემთა არარელაციური ბაზების განვითარების ძირითადი მოთხოვნები და ტენდენციები

ტექნოლოგიური განვითარებისა და ამოცანების თვისობრივად შეცვლის გამო გაჩნდა ისეთი მოთხოვნები, რომლებსაც რელაციური მონაცემთა ბაზები ვეღარ აკმაყოფილებს [92,95].

სინამდვილეში არის ორი ძირითადი პრობლემა, რაც ამ მოდელს აქვს:

- **მონაცემების წარმოდგენა და დამუშავება**

რელაციურ მოდელში მონაცემები სხვა სტრუქტურით გვაქვს წარმოდგენილი, აპლიკაციაში კი სრულიად განსხვავებული სტრუქტურით მოვიხმართ.

მონაცემები როგორც წარმოდგენილი გვაქვს ბაზაში, აბსოლუტურად განსხვავდება იმისგან, როგორც გვაქვს ოპერატიულ მეხსიერებაში, რასაც შემდეგ უკვე ვიყენებთ.

რეალურად, ჩვენ აპლიკაციაში ძალიან დიდი რესურსი ეთმობა შემდეგ ოპერაციებს:

- წავიკითხოთ მონაცემები ბაზის სხვადასხვა ცხრილებიდან;
- გავაერთიანოთ ისინი აპლიკაციისთვის საჭირო ობიექტში;
- დავამუშაოთ და გამოვიყენოთ ეს ობიექტი;
- დავშალოთ შექმნილი ობიექტი და ცვლილებები შესაბამის ცხრილებში გავუშვათ.

რეალურად, მონაცემთა ბაზის ოპერაციებიდან ძალიან დიდი დრო ეთმობა ამ მიმდევრობების მრავალჯერ გაშვებას.

22.7.1. ვერტიკალური და ჰორიზონტალური სკალირება

ბოლო პერიოდში ინფორმაციისა და პარალელური მოთხოვნების რაოდენობა იმდენად გაიზარდა, რომ საჭირო გახდა არქიტექტურის სრულიად შეცვლა.

ტრადიციულ რელაციურ ბაზებში თუ იზრდება მოთხოვნების დატვირთვა, ვაძლიერებთ სერვერს (vertical scale).

გარკვეული ზღვრის შემდეგ სერვერის განვითარება საკმაოდ ძვირი ჯდება, თანაც ნაკლებად ეფექტურია. ადამიანები მივიდნენ იმ დასკვნამდე, რომ მცირე რაოდენობის მძლავრ სერვერებს ჯობია დიდი რაოდენობის იაფფასიანი სერვერები (horizontal scale), რომლებიც იმუშავებს როგორც ერთი კლასტერი (ნახ.22.18).

ნახ.22.18

22.7.2. მონაცემთა შენახვის ტიპები

NoSQL ბაზებში მონაცემების შენახვის 4 ძირითადი ტიპი არსებობს:

1. **Key-Value Store** – აქვს დიდი ჰეშ ცხრილი გასაღებებისთვის და მათი მნიშვნელობებისათვის {მაგალითად - Riak, Amazon S3 (Dynamo), Redis…}

2. **Document-based Store**- ინახავს იარლიყიანი ელემენტებისაგან (tagged elements) შემდგარ დოკუმენტებს {მაგალითად - MongoDB, CouchDB…}

3. **Column-based Store**- მეხსიერების თითოეული ბლოკი ინახავს მხოლოდ ერთი სვეტის ინფორმაციას {მაგალითად, HBase, Cassandra}

4. **Graph-based**-ქსელური ტიპის მონაცემთა ბაზა, რომელიც იყენებს წიბოებსა და კვანძებს მონაცემების შესანახად და წარმოსადგენად {მაგალითად - Neo4J}.

განვიხილოთ თითოეული მათგანი.

➢ **Key-Value Store**

Key-Value ყველაზე მარტივი სტრუქტურაა: გვაქვს უნიკალური პარამეტრი key და გვაქვს შესაბამისი მნიშვნელობა - value (ცხრ.22.1).

ცხრ.22.1

გავრცელებული სისტემები	Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB, Amazon SimpleDB, Riak
ტიპური გამოყენება	Content caching (განაწილებულ სერვერებზე დიდი ოდენობის ინფორმაციის დამუშავებაზე ორიენტირება), ლოგ–სერვერები...
მოდელი	Key-Value წყვილების კოლექცია
უპირატესობა	სწრაფი ძებნა
სისუსტე	შენახულ მონაცემებს არ აქვს სქემა

სიმარტივისთვის რომ წარმოვიდგინოთ, შესაბამის ლოგიკას რელაციურ ბაზაში ორი სვეტით ავაწყობდით, – Primary key და მასთან დაკავშირებული სვეტი მონაცემებისთვის. განსხვავება ის იქნებოდა რომ რელაციურ ბაზაში წინასწარ უნდა მიგვეთითებინა მონაცემების სვეტის ტიპი, NoSQL ბაზებში კი ეს მნიშვნელობა შეიძლება იყოს რიცხვი, ტექსტი, სურათი და ამასთან ერთად, არ არის აუცილებელი, რომ ყველა სტრიქონში ერთსა და იმავე ტიპის ინფორმაციას ვინახავდეთ.

მონაცემების ამგვარი შენახვა გვაძლევს იმის საშუალებას, რომ ერთი ცხრილის სხვადასხვა ჩანაწერი კლასტერის სხვადასხვა კვანძზე მოვათავსოთ. როგორც წესი, ეს გადანაწილება გასაღების მნიშვნელობის მიხედვით ხდება.

Key-Value ტიპის ბაზებში რთულია ატომარობისა და კონსისტენტურობის დაცვა – სანამ ერთი მომხმარებელი ჩანაწერს ანახლებს, სხვა მომხმარებელმა შეიძლება წასაკითხად მიაკითხოს იმავე ჩანაწერს. ამგვარ შემთხვევებში გვაქვს ორი ვარიანტი,– მივაწოდოთ მომხმარებელს ჩანაწერის ბოლო განახლებული ვერსია, ან მივაწოდოთ ყველა არსებული ვერსია და თავად მივცეთ იმის საშუალება, რომ გაარკვიოს, რომელი ვერსიის გამოყენება ურჩევნია. ამგვარად, აპლიკაციის მხარეს მეტ–ნაკლებად შესაძლებელი ხდება კონსისტენტურობის დაცვაც, მაგრამ თუ დასმული ამოცანისთვის კრიტიკულია ბაზის ტრანზაქციულობა, მაშინ key-value strorage საიმედო გადაწყვეტილება ვერ იქნება.

როგორც წესი, key-value მეთოდი იყენებს ჰეშ ცხრილებს, რომლებშიც ინფორმაცია ლოგიკურ გაერთიანებებად(bucket) არის დაყოფილი. რეალურად, გასაღები არის ჩვენს გასაღებს + Bucket მნიშვნელობების კონკატენაციის ჰეში –; hash (Bucket+ Key)

CAP თეორემას თუ მივუბრუნდებით, ცხადი ხდება, რომ key-value მეთოდი იდეალურია Availability და Partition მხარდაჭერებისათვის, მაგრამ საგრძნობლად მოიკოჭლებს Consistency კონსისტენტურობის კუთხით [135-137].

Key-value storage-ის ძალიან კარგი გამოყენებაა სესიის ინფორმაციის შენახვა, როგორც key - სესიის იდენტიფიკატორი და value - სესიასთან დაკავშირებული ინფორმაციის ნაკრები.

ასევე საინტერესო მაგალითია მომხმარებლის პირადი ინფორმაციის შენახვა, სადაც key გასაღები არის მომხმარებლის უნიკალური იდენთიფიკატორი, value მნიშვნელობა კი მთელი ის ინფორმაცია, რაც ახასიათებს მომხმარებელს.

22.2 ცხრილში Key-value პრინციპით შეტანილია საქართველოს ბანკის ფილიალები დასახლებების მიხედვით.

უნდა გავითვალისწინოთ, რომ თუ გასაღებად ტექსტურ მნიშვნელობებს ავირჩევთ, დროთა განმავლობაში გაგვიჭირდება უნიკალურობის დაცვა.

ცხრ..22.2

Key	Value
ლანჩხუთი	{"ჟორდანიას ქუჩა #101"
ქობულეთი	{"ნინოშვილის ქუჩა #1"}
თბილისი	{"ვეკუას ქუჩა #1თბილისი","პუშკინის ქუჩა #3თბილისი","კოსტავას ქუჩა #24თბილისი","თაბუკაშვილის ქუჩა #38თბილისი"}

Key-value ტიპის მონაცემთა ბაზა მონაცემების დასამუშავებლად მომხმარებელს შემდეგ ფუნქციონალს სთავაზობს:

- Get(key), აბრუნებს პარამეტრად მიღებული key გასაღების შესაბამის value მნიშვნელობას.
- Put(key, value), აკავშირებს მნიშვნელობას გასაღებთან.
- Multi-get(key1, key2, .., keyN), აბრუნებს მიღებული გასაღებების შესაბამისი მნიშვნელობების მიმდევრობას.
- Delete(key), მონაცემთა ბაზიდან ამოშლის შესაბამის ჩანაწერს.

„გასაღები-მნიშვნელობა“ მოდელს თუ, შევადარებთ რელაციურ მოდელს შემდეგ შესაბამისობებს მივიღებთ:

- Table -> bucket
- Row -> key-value
- Rowid -> key

➢ **Document-based Store**

მონაცემების წარმოდგენის დოკუმენტზე ორიენტირებული ტიპები, key-value ტიპის მსგავსად, ინფორმაციას ინახავს. განსხვავება ისაა, რომ დოკუმენტებად შენახულ ინფორმაციას გარკვეული სტრუქტურა და წარმოდგენის განსხვავებული სახე აქვს (ცხრ.22.3).

ცხრ.22.3

გავრცელებული სისტემები	CouchDB, MongoDb
ტიპური გამოყენება	ლოგ–სერვერები, ვებ აპლიკაციები(Key-value–ს დახვეწილი ვერსია)
მოდელი	Key-Value წყვილების კოლექციების კოლექციები (ჩადგმული კოლექციები)
უპირატესობა	ფუნქციონირებს არასრული ინფორმაციის შემთხვევაშიც
სისუსტე	ტრანზაქციების წარმადობა; არ აქვს სტანდარტული სინტაქსი მოთხოვნების ჩამოსაყალიბებლად.

Document-based მონაცემთა ბაზები ინფორმაციას List<Object> სიის სახით ინახავს. ამგვარ სიებში შესაძლებელია ჩაიწეროს მრავალი სხვადასხვა სახის მონაცემი.

მაგალითისათვის Document-based მონაცემთა ბაზები შეგვიძლია შევადაროთ ფოლდერს, რომელშიც Ms Word-ის მრავალი ფაილია მოთავსებული, თითოეულ ფაილში კი სხვადასხვა სტრუქტურა და შიგთავსია. თითოეული ფოლდერი ჩვენთვის დოკუმენტების კოლექცია იქნება (collection).

დოკუმენტებზე ორიენტირებულ მოდელს თუ რელაციურ მოდელს შევადარებთ, შემდეგ შესაბამისობებს მივიღებთ:

Table -> collection
Row -> BSON document
Column -> BSON Field
Rowed -> _id
Index -> Index
Join -> ჩადგმული დოკუმენტი (Embedded Document)
Partition -> Shard
Partition Key -> Shard Key

Document-based მოდელის გამოყენების საინტერესო მაგალითია CMS ძრავებსა და ყველა იმ დავალებაში, სადაც გვიწევს, ბრაუზერს ყოველ მიმართვაზე თავიდან დავაგენერირებინოთ კოდი. ცხადია, ბევრად უკეთეს წარმადობას მივიღებთ, თუ უკვე დაგენერირებულ სკრიპტს შევინახავთ და ბრაუზერს არ მოუწევს ყოველ ჯერზე ხელახალი დაგენერირება.

- **Column-based Store**

Column-based Store პრინციპი შემუშავებულ იქნა მრავალ მანქანაზე გადანაწილებული დიდი რაოდენობის მონაცემების დასამუშავებლად (ცხრ.22.4).

ცხრ.22.4

გავრცელებული სისტემები	Cassandra, HBase, Riak
ტიპური გამოყენება	განაწილებული ფაილური სისტემები Hadoop Distributed File System (HDFS)
მოდელი	სვეტი –> სვეტების გაერთიანება
უპირატესობა	სწრაფი ძიება, განაწილებული გარემოს საუკეთესო მხარდაჭერა
სისუსტე	დაბალი დონის API

22.7.3. განაწილებული გარემო: replication და sharding

- **replication ტექნოლოგია**

რეპლიკაცია არის პროცესი, რომლის დროსაც მონაცემები სინქრონიზირდება ერთი, მთავარი სერვერიდან რამდენიმე სათადარიგო სერვერზე [138]. რეპლიკაციის დროს მთავარ (primary) სერვერზე ჩაწერისა და კითხვის ოპერაციების განხორციელება არის შესაძლებელი, ხოლო სათადარიგო სერვერებიდან კი მხოლოდ წაკითხვისა.

რეპლიკაციას რამდენიმე ძირითადი დადებითი მხარე აქვს:

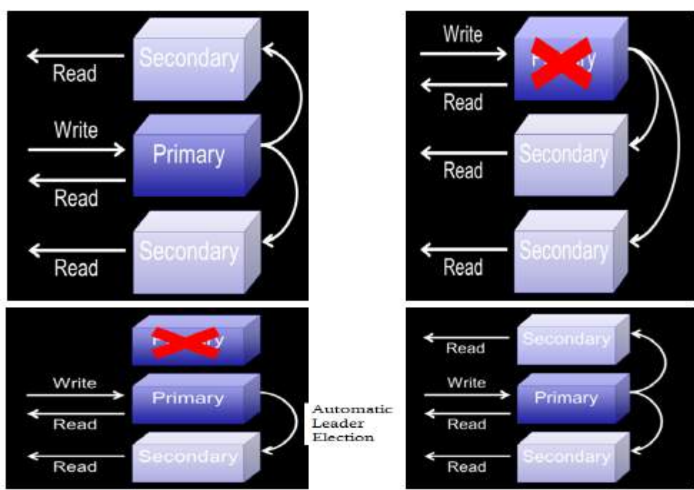
- *მონაცემთა კითხვის გაუმჯობესება* – რამდენ დუბლირებული სერვერიც არის დაკონფიგურირებული, იმდენი დამოუკიდებელი წყაროდანაა შესაძლებელი ინფორმაციის პარალელურ რეჟიმში წაკითხვა;
- *Disaster Recovery* – პრობლემის ან გეგმიური სამუშაოების დროს შესაძლებელია რომელიმე სათადარიგო (standby) სერვერი გახდეს ძირითადი (მთავარი - primary);
- აღარ არის საჭირო backup-ებისა და ინდექსების რეორგანიზაციის გამო მონაცემთა ბაზის გაჩერება ან შენელება;
- რეპლიკა სეტები აპლიკაციის დონეზე არ ჩანს.

მთავარ სერვერთან კავშირის დაკარგვის ან გათიშვის შემთხვევაში სათადარიგო სერვერები აწყობს არჩევნებს და ირჩევს ახალ primary სერვერს.

რეპლიკა სეტების არჩევნებში მონაწილეობს წინასწარ განსაზღვრული სერვერები – არბიტრები. არბიტრი არ შეიცავს მომხმარებელთა ინფორმაციას, მისი ერთადერთი დანიშნულება არჩევნებში ხმის მიცემაა.

თუ Replica set-ში გვაქვს ლუწი რაოდენობის კვანძი, მაშინ არბიტრის დამატებით ვიღებთ კენტი რაოდენობის წევრებს, რითაც ვაღწევთ ხმათა უმრავლესობას არჩევნებში.

ხშირად რეპლიკაციას სამი რეპლიკა სეტისგან აწყობენ (ნახ.22.19) [118].



ნახ.22.19

რეპლიკაციის აწყობა დიდ სირთულეებთან არ არის დაკავშირებული, მთავარ სერვერზე ვქმნით რეპლიკაციის გარემოს:

mongod --port "PORT" --dbpath „ბაზის მისამართი" --replSet "რეპლიკა სეტის ინსტანსის სახელი";

ამის შემდეგ ვუკავშირდებით უკვე შექმნილ გარემოს და ვუშვებთ რეპლიკაციის პროცესს : rs.initiate().

რეპლიკა სეტის კონფიგურაციის შემოწმება შეგვიძლია ბრძანებებით rs.conf() და rs.status().

ახალი წევრების დასამატებლად ვუშვებთ

rs.add(HOST_NAME:PORT)

სტრუქტურის ბრძანებას.

თუ არაფერს არ შევცვლით, კითხვისათვის კლიენტები ძირითად სერვერს მიმართავენ, მაგრამ აპლიკაციიდან შეგვიძლია ავირჩიოთ „Read Preference“ და კითხვა განვახორციელოთ მეორადიდან.

Read Preference პარამეტრი საგრძნობლად აუმჯობესებს წარმადობას, მაგრამ ცხადია, პრობლემა ხდება კონზისტენტურობა,– სანამ ძირითად(primary) სერვერზე განხორციელებული ცვლილება სათადარიგო(secondary) ბაზამდე მიაღწევს, კლიენტები ძველ ინფორმაციას კითხულობენ. ეს პრობლემა გადაჭრილია majority პარამეტრის შემოღებით. ამ პარამეტრით ვუთუითებთ, მინიმუმ რამდენ კვანძზე უნდა ჩაიწეროს ინფორმაცია, რომ ტრანზაქცია წარმატებულად ჩაითვალოს. ძირითად სერვერზე ინფორმაციის რამდენიმე ვერსია გვექნება, მაგრამ მომხმარებლებს მივაწვდით მხოლოდ იმ ვერსიას, რომელიც დასრულეულია, ანუ სათადარიგო სერვერების საჭირო რაოდენობაზე უკვე ჩაწერილია.

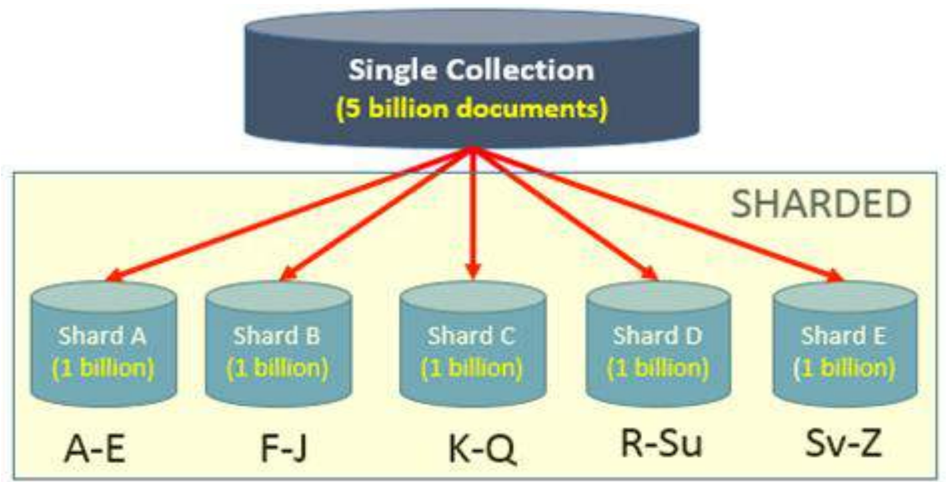
- **sharding ტექნოლოგია**

sharding არის სერვერებზე მონაცემთა თანაბარი განაწილების პროცესი. ტერმინი „Shard“ ნიშნავს ერთი მთლიანის პატარა ნაწილს. ამ ტექნოლოგიით ხორციელდება მონაცემთა დიდი ზომის უმართავი ბაზების დანაწევრება (database partitioning) მცირე, სწრაფ და ადვილად მართვად ნაწილებად.

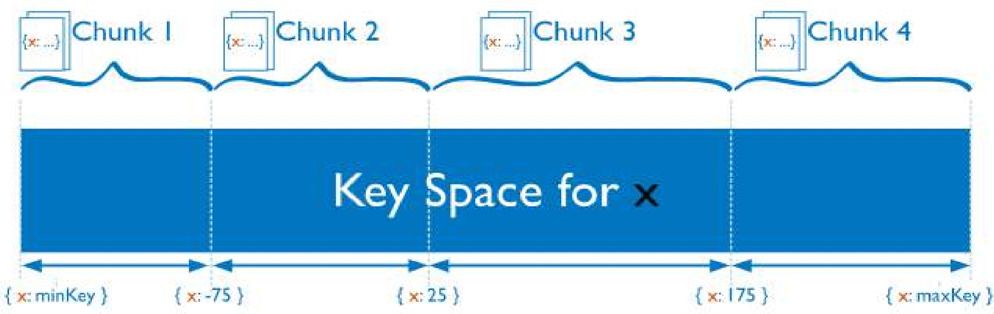
ტექნიკურად არც sharding-ის აწყობაა რთული. მთავარი სირთულე shard key გასაღების სწორად არჩევაა. ამ გასაღების მიხედვით ნაწილდება დოკუმენტები შესაბამის Shard-ებზე.

მაგალითისათვის შეგვიძლია მოვიყვანოთ 5 ბილიონი ჩანაწერის მქონე კოლექცია, რომელიც გადანაწილებულია 5 Shard-ზე, გასაღებად კი აღებულია ტექსტური ველი. გასაღების ამგვარი არჩევის შემთხვევაში მარტივად შეგვიძლია ახალი Shard-ის დამატება რომელიმე ძველ Shard-ზე ინფორმაციის გადანაწილებით (ნახ.22.20).

დოკუმენტები ერთიანდება ფიქსირებული ზომის ფრაგმენტებში (chunk-ებში), რომლის სტანდარტული (default) ზომა 64 მეგაბაიტია. რამდენიმე chunk ერთიანდება ერთ shard-ბლოკში (ნახ.22.21) [129,139]. მონაცემები დაყოფილია დიაპაზონებად, რომლებიც განისაზღვრება გასაღებური სივრცით (Key Space).

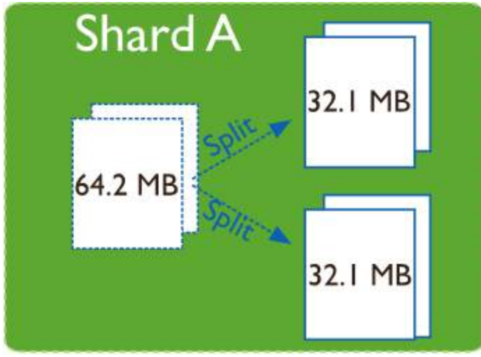


ნახ.22.20



ნახ.22.21

თითოეული 64 მბ-ანი ფრაგმენტი (Chunk) შეიძლება დაიყოს Shard-ბლოკში შედარებით მცირე ზომის ფრაგმენტებად (ნახ.22.22).



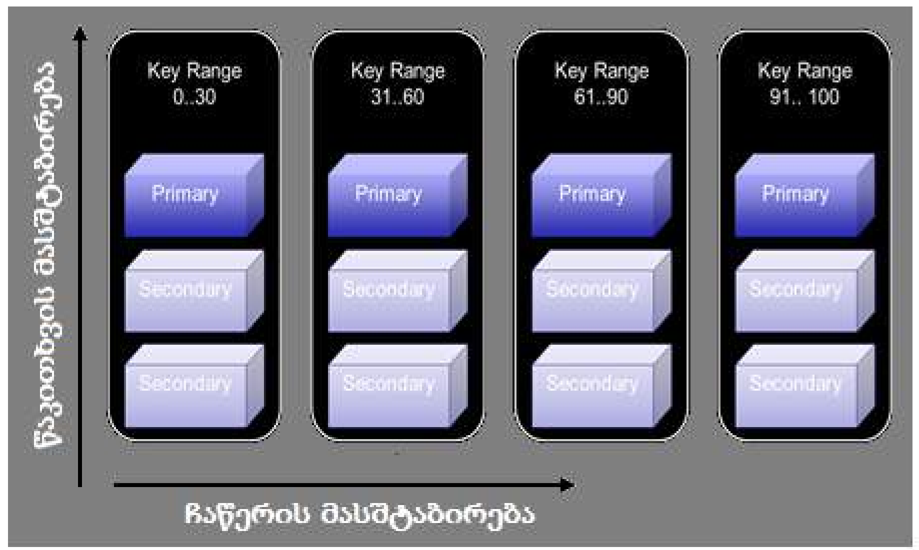
ნახ.22.22

მონაცემთა რაოდენობის გაზრდის შემდეგ ახალი shard სერვერის დასამატებლად საჭიროა ერთი ან რამდენიმე არსებული shard-იდან ახალ სერვერზე გადატანილ იქნას ახალი დაყოფის შესაბამისი chunk-ები.

MongoDB chunk-ების ბალანსირების პროცესს ავტომატურად, კლიენტებისგან დამოუკიდებლად ასრულებს. ასევე, მომხმარებლისგან დამოუკიდებლად წყდება, თუ რომელი shard-ებია პასუხისმგებელი კლიენტის მოთხოვნაზე. ამისათვის არსებობს მოთხოვნების მარშრუტიზატორი (query router), რომელიც კლიენტისგან იღებს მოთხოვნას, წინასწარ შენახული მეტა ინფორმაციის (metadata) მიხედვით არჩევს შესაბამის shard სერვერებს და კლიენტს უბრუნებს საბოლოო აწყობის შედეგს.

- **replication + sharding განაწილებული გარემო**

როგორც ცნობილია, მონაცემთა ბაზების წარმადობა ტექნიკური თვალსაზრისით, დამოკიდებულია სამ ძირითად მახასიათებელზე: პროცესორი, ოპერატიული მეხსიერება და მყარი დისკო. რეპლიკაციისა და შარდინგის დროს კი მოთხოვნის ზრდასთან ერთად შესაძლებელია ასობით და ათასობით ახალი მანქანის დამატება და თითოეული მათგანი გააუმჯობესებს აღნიშნული მახასიათებლების მნიშვნელობებს (ნახ.22.23).



ნახ.22.23

მონაცემთა ბაზის ადმინისტრატორები ხშირად საჭიროზე მეტ ოპერატიულ მეხსიერებას და პროცესორებს უყოფენ აპლიკაციას. ამ დროს თავს იჩენს ე.წ. bottleneck (ბოთლის შევიწროებული ყელის) პრობლემა, დისკოების სიმცირეში. სერვერი დიდ დროს კარგავს დისკოდან ინფორმაციის ამოკითხვაზე.

ეს პრობლემა განაწილებულ გარემოში მუშაობისას მინიმუმამდეა დაყვანილი:

- გვჭირდება სწრაფი კითხვა? – ვამატებთ ახალ რეპლიკა სეტს, ანუ ბაზის ახალ ასლს (კოპიოს);
- გვჭირდება სწრაფი ჩაწერა/წაკითხვა? – sharding ტექნოლოგიით უფრო მცირე ნაწილებად ვყოფთ მონაცემთა ბაზას, შესაბამისად, უფრო მეტი დისკო წერს და კითხულობს ერთდროულად.

22.7.4. MySQL და MongoDB ბაზების შედარება

რელაციური და არარელაციური მოდელები საკმაოდ განსხვავდება ერთმანეთისგან. რელაციურ მოდელისთვის არსებობს სქემა და მონაცემები განთავსებულია მის დაკავშირებულ ცხრილებში (Tables). ცხრილი შეიცავს სტრიქონებს (Rows) და სვეტებს (Columns). ცხრილები ერთმანეთს უკავშირდება პირველადი (PrimaryKey) და მეორეული (ForeignKey) გასაღებების საშუალებით. როდესაც მოთხოვნის საფუძველზე იძებნება რაიმე ინფორმაცია, შესაბამისი ჩანაწერები მიიღება რამდენიმე ცხრილის შეერთების (join), პროექციის, შეზღუდვის, უნიკალურობის (სიმრავლის) და/ან სხვა ოპერაციების მიმდევრობით (ან პარალელური) შესრულების საფუძველზე. მსგავსია ჩაწერის და მოდიფიკაციის პროცედურებიც, რომლებიც უნდა მოხდეს რამდენიმე ცხრილში ერთდროულად (ბაზის მთლიანობის შესანარჩუნებლად).

NoSQL ბაზებს, რელაციურთან შედარებით, აქვს სრულიად განსხვავებული მოდელი. მაგალითად: დოკუმენტზე ორიენტირებული NoSQL ბაზები მონაცემებს ინახავს JSON (JavaScript Object Notation) ფორმატში [117]. თითოეული JSON დოკუმენტი შეიძლება განვიხილოთ როგორც ობიექტი, რომელსაც მიიღებს აპლიკაცია. ის შეიძლება შეიცავდეს რელაციური მოდელის რამდენიმე ცხრილის გადაბმით მიღებულ ინფორმაციას ერთ დოკუმენტში/ობიექტში, რაც უზრუნველყოფს ჩაწერა/წაკითხვის ოპერაციების წარმადობის გაუმჯობესებას.

მაგალითად, MongoDB არის open-source მონაცემთა ბაზის სისტემა. იგი ინახავს მონაცმებს JSON ტიპის დოკუმენტებში, რომელთა სტრუქტურაც შეიძლება იცვლებოდეს. ინფორმაცია ინახება ერთად. MongoDB იყენებს დინამიურ სქემებს, რაც ნიშნავს, რომ შესაძლებელია ჩანაწერების შექმნა სტრუქტურის (ველების, მნიშვნელობათა ტიპების) წინასწარი განსაზღვრის გარეშე. შემდგომ შეიძლება ჩანაწერების სტრუქტურის (ანუ დოკუმენტების) მარტივად შეცვლა ახალი ველის დამატებით ან არსებულის წაშლით.

ეს მოდელი გვაძლევს საშუალებას წარმოვადგინოთ იერარქიული კავშირები და სხვა უფრო რთული სტრუქტურები შედარებით მარტივად. დოკუმენტებს კოლექციაში არ სჭირდება იდენტური ველები და მონაცემთა დენორმალიზაცია არის აქ ჩვეულებრივი მოვლენა. MongoDB ბაზის სისტემა შეიქმნა მაღალი წვდომადობისა და მასშტაბირების რეალიზაციის მიზნით, ფლობს რეპლიკაციას და ავტომატურ სეგმენტაციას (auto-sharding).

MySQL და MongoDB ბაზებში მრავალი საერთო ტერმინი და ცნებაა (ცხრ.22.5) [134].

ტერმინების შედარება ბაზებში ცხრ.22.5

MySQL	MongoDB
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents, linking

Collection არის mongo დოკუმენტების ჯგუფი. იგი RDBMS ცხრილების ეკვივა-ლენტია.

Document – ცხრილის ძირითადი ნაწილია. დოკუმენტები არის დინამიკური. MongoDB და SQL ბაზები გვთავაზობს ფუნქციების მდიდარ კომპლექსს (ცხრ22.6) [47].

ფუნქციები და შესაძლებლობები ცხრ.22.6

დასახელება	MySQL	MongoDB
Rich Data Model	No	Yes
Dynamic Schema	No	Yes
Typed Data	Yes	Yes
Data Locality	No	Yes
Field Updates	Yes	Yes
Easy for Programmers	No	Yes
Complex Transactions	Yes	No
Auditing	Yes	Yes
Auto-Sharding	No	Yes

MongoDB ბაზას აქვს მაღალფუნქციური მოთხოვნების ენა (query language), რომელსაც შეუძლია აგრეთვე ტექსტებთან და სივრცით მონაცემებთან (გეოსისტემები) მუშაობა. აღნიშნული ფუნქციების გამოყენება შესაძლებელია მონაცემთა მრავალი სახის ტიპებთან, ვიდრე რელაციურ ბაზებში.

ქვემოთ მოყვანილია რამდენიმე მარტივი მოთხოვნის ფორმირების მაგალითი MySQL და MongoDB ბაზებთან მუშაობის დროს:

- „შევიტანოთ products მონაცემთა ბაზაში (ნახ.22.19) ახალი პროდუქტის შესახებ ინფორმაცია (სტრიქონი), რომელშიც გვაქვს ოთხი ველი: პროდუქტის იდენტიფიკატორი (pr_ID), დასახელება (Name), ფასი (price) და პროდუქტის კატეგორიის იდენტიფი-კატორი (cat_ID)“.

Field	Type	Length/Values[1]
pr_ID	INT	4
Name	VARCHAR	30
prize	DECIMAL	
cat_ID	INT	2

ნახ.22.19

- MySQL-ში:

```
INSERT INTO products (pr_ID, Name, price, cat_ID)
VALUES (101, 'არაჟანი', 5.80, 6)
```

- MongoDB-ში:

```
db.products.insert ({
    pr_ID: 101,
    Name: 'არაჟანი',
    price: 5.80,
    cat_ID: 6
})
```

➢ „ვნახოთ (ავირჩიოთ) ყველა პროდუქტის მონაცემები“:

- MySQL-ში:

```
SELECT  * FROM  products
```

- MongoDB-ში:

```
db.products.find()
```

➢ „შევცვალოთ (გავზარდოთ) მე-6 კატეგორიის („რძის ნაწარმი“) პროდუქტების ფასი 20 % -ით“:

- MySQL-ში:

```
UPDATE products SET price = price*0.2
WHERE cat_ID = 6
```

- MongoDB-ში:

```
db.products.update(
   { cat_ID: 6 },
   { $set: { price: price*0.2 } }
   { multi: true }
)
```

➢ „წავშალოთ products ბაზაში ალკოჰოლური სასმელების მონაცემები (ალკ_სასმელის იდენტიფიკატორია cat_ID = 4)“.

- MySQL-ში:

```
DELETE FROM products
WHERE cat_ID = 4
```

- MongoDB-ში:

```
db.products.remove (
   ( cat_ID:  4  }
)
```

და ა.შ.

MongoDB ორგანიზაციებს აძლევს საშუალებას უფრო სწრაფად შექმნას აპლიკაციები, დაამუშაოს განსხვავებულ ტიპთა დიდი მოცულობის ჩანაწერები, აპლიკაციათა მასშტაბირების მართვა განახორციელოს უფრო ეფექტურად. ამავდროულად, მონაცემთა ბაზის დამუშავება (დეველოპმენტი) და განახლება საკმაოდ გამარტივებულია.

მონაცემთა ობიექტრელაციური მოდელის (სქემის) ცვლილება, მაგალითად, MySQL-ში, მოითხოვს განახლების შედარებით დიდ დროს, როდესაც MongoDB-ს მონაცემთა მოქნილი მოდელის გამო ასეთი პროცედურები სწრაფად ხორციელდება.

მომდევნო პარაგრაფში ჩვენ უფრო დეტალურად განვიხილავთ MongoDB ბაზის სისტემაში მუშაობის საკითხებს.

22.7.5. მონაცემთა ბაზის აგება MongoDB სისტემაში

MongoDB-ში ბაზას ცხადად არ ვქმნით, ვირჩევთ მიმდინარე ბაზას (რომელიც ახალი ბაზის შექმნის დროს რეალურად არც არსებობს) და არჩეულ ბაზაში უბრალოდ ვამატებთ ახალ ჩანაწერებს. თვითონ მონაცემთა ბაზა პირველივე მონაცემის ჩაწერის პარალელურად, ავტომატურად იქმნება [118].

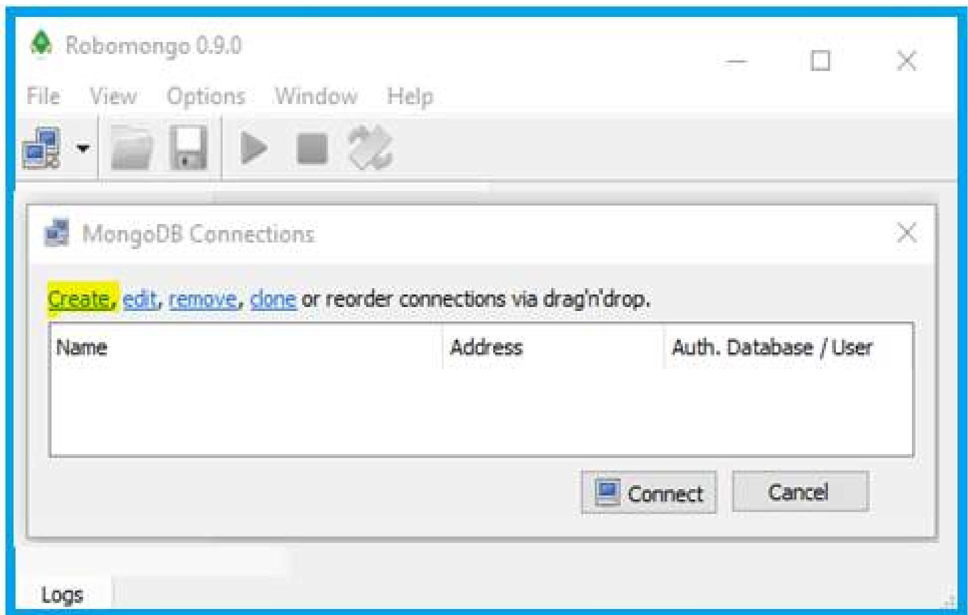
1-ელ დანართში მოყვანილია საცდელი ბაზის შექმნის მაგალითი MongoDB სისტემისთვის [140]. კონკრეტული ოპერაციებისა და მაგალითების განხილვამდე შევქმნათ moviesDB მონაცემთა ბაზა დანართის მიხედვით.

moviesDB მონაცემთა ბაზაში მხოლოდ ერთი კოლექცია გვაქვს - movies, თუმცა, ცხადია, რეალურ სისტემაში ამ კოლექციის გარდა შესაძლოა, გვქონოდა music, games, users, actors … სხვადასხვა კოლექციები. ამ კოლექიაში თავმოყრილია მთელი ის ინფორმაცია, რაც ფილმის შესახებ ინახება.

22.7.5.1. RoboMongo პლატფორმა

RoboMongo პლატფორმა არის ათზე მეტი ალტერნატიული ინსტრუმენტიდან ერთ-ერთი ყველაზე გავრცელებული GUI (Graphical User Interface) გრაფიკული ინტერფეისი MongoDB მონაცემთა ბაზასთან სამუშაოდ [118,141].

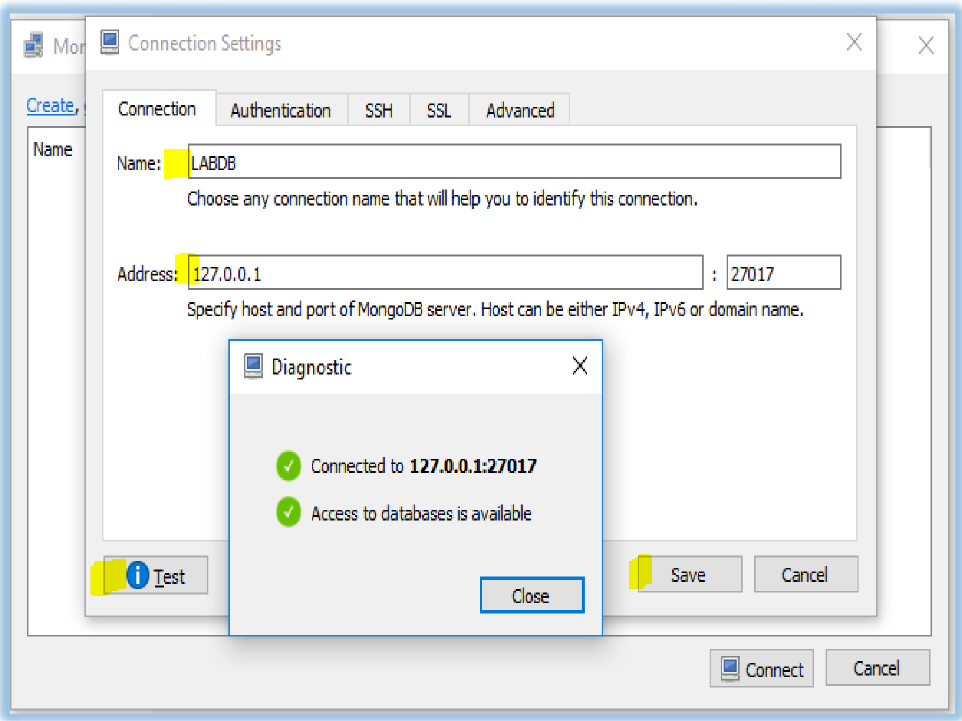
RoboMongo პლატფორმა მოხერხებულობითა და სიმარტივით გამოირჩევა. თავდაპირველად ვამატებთ ახალ კავშირს MongoDB სისტემასთან (ნახ. 22.20).



ნახ. 22.20

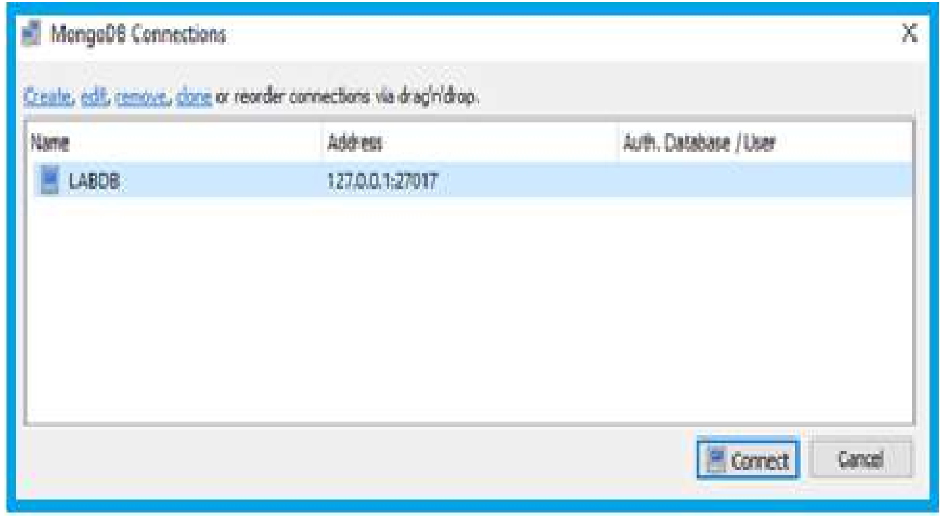
დიალოგის ფანჯარაში (ნახ. 22.21) შევავსებთ საჭირო ველებს:

- Name - კავშირის სახელი (შეგვიძლია ნებისმიერი სახელის დარქმევა);
- Address - სერვერის IP მისამართი და პორტი, რომელზეც ელოდება კლიენტებისგან მოსულ მოთხოვნებს. ჩვენ შემთხვევაში ჩავწერეთ 127.0.0.1 - რადგან სერვერი ჩვენივე კომპიუტერზე გვაქვს დაყენებული. პორტი კი უცვლელად დავტოვეთ (default – 27017).



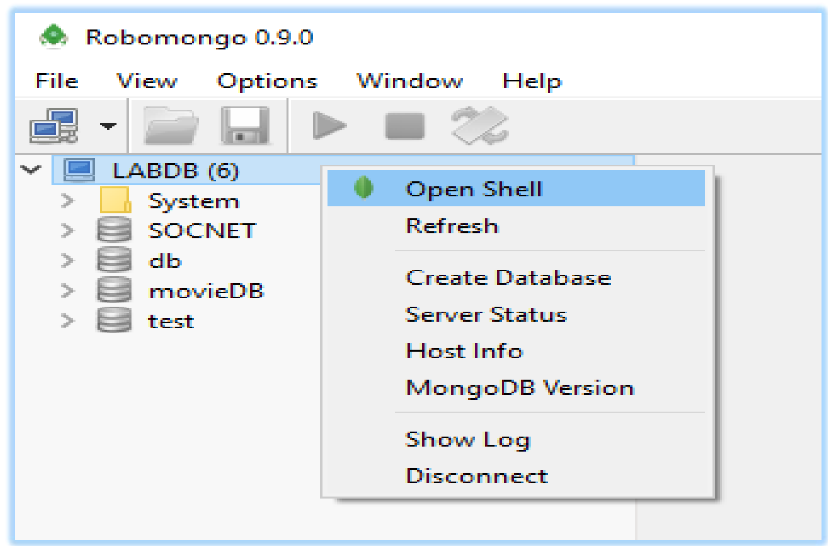
ნახ. 22.21

კავშირის შექმნამდე (Save) შეგვიძლია შევამოწმოთ მისი ვალიდურობა Test ღილაკზე დაჭერით. შემდეგ, 22.22 ნახაზზე ვხსნით ახლად შექმნილ კავშირს.



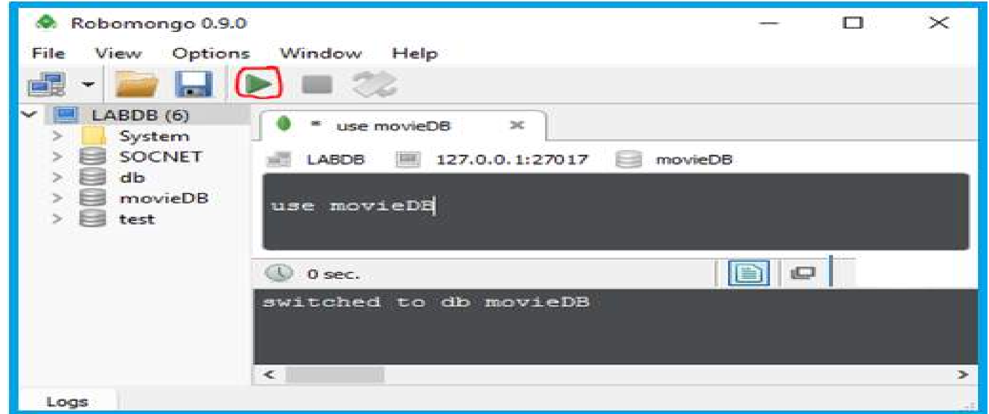
ნახ. 22.22

22.23 ნახაზზე ჩანს უკვე დაკავშირებულ სერვერთან ბრძანებების რეჟიმში გადასვლის მეთოდი.



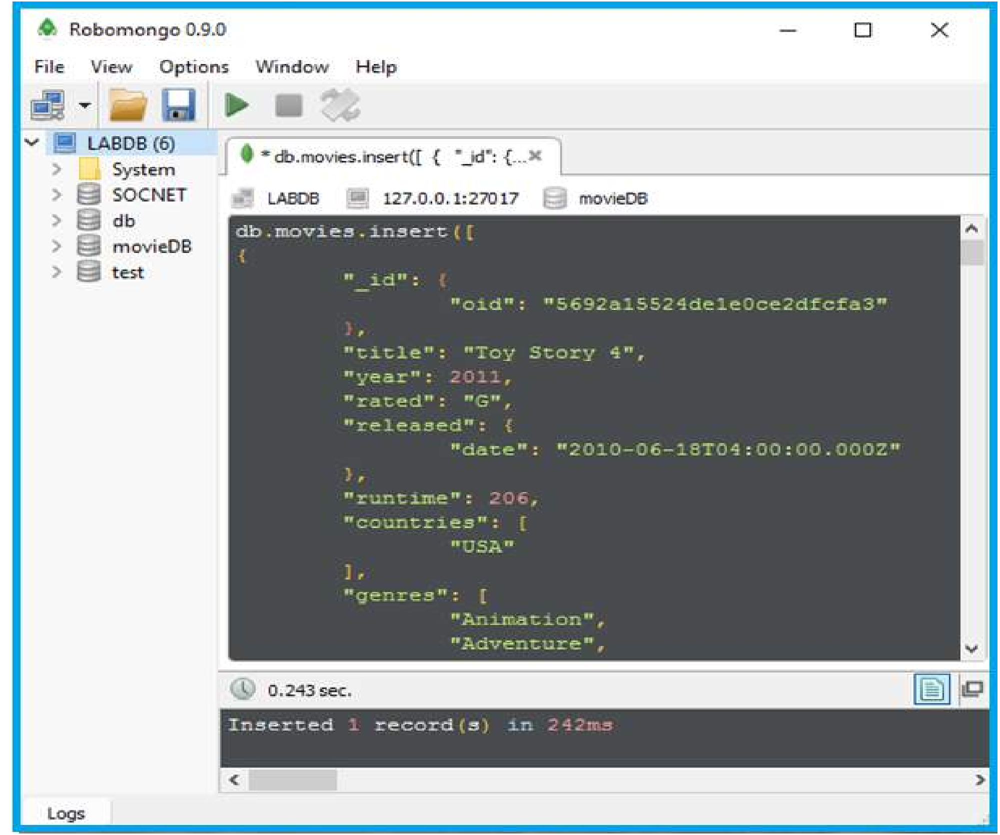
ნახ. 22.23

შემდეგ 22.24 ნახაზზე მოცემულია პირველი ბრძანების - ბაზასთან დაკავშირების მაგალითი



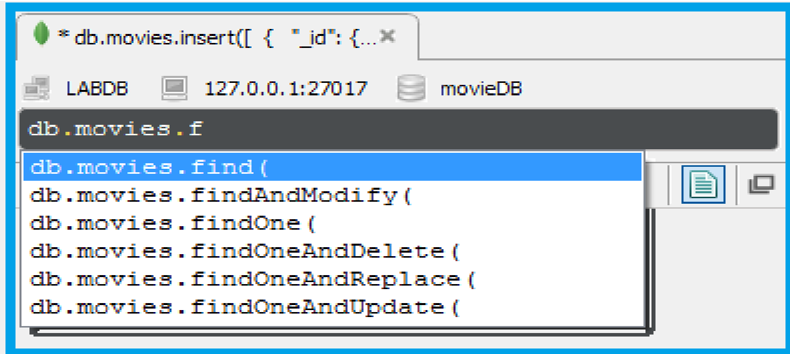
ნახ. 22.24

შემდეგ ეტაპზე დავამატოთ 1-ელ დანართში ნაჩვენები მონაცემები movies კოლექციაში (ნახ.22.25).



ნახ. 22.25

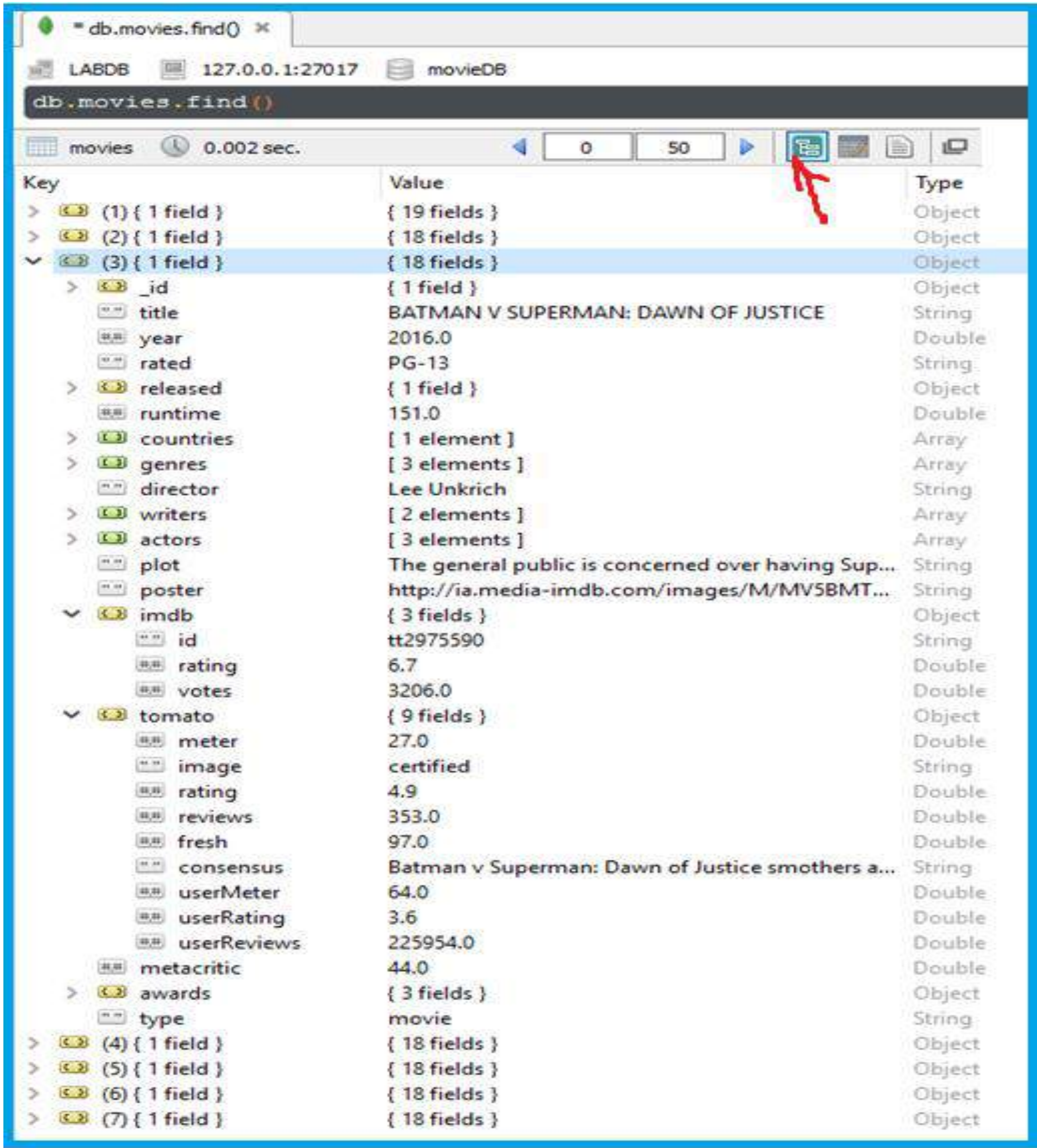
შემდეგ, 22.26 ნახაზზე ჩანს RoboMongo გარემოს intellisense შესაძლებლობა - შემოგვთავაზოს შესაძლო ვარიანტები კოდის წერის დროს.



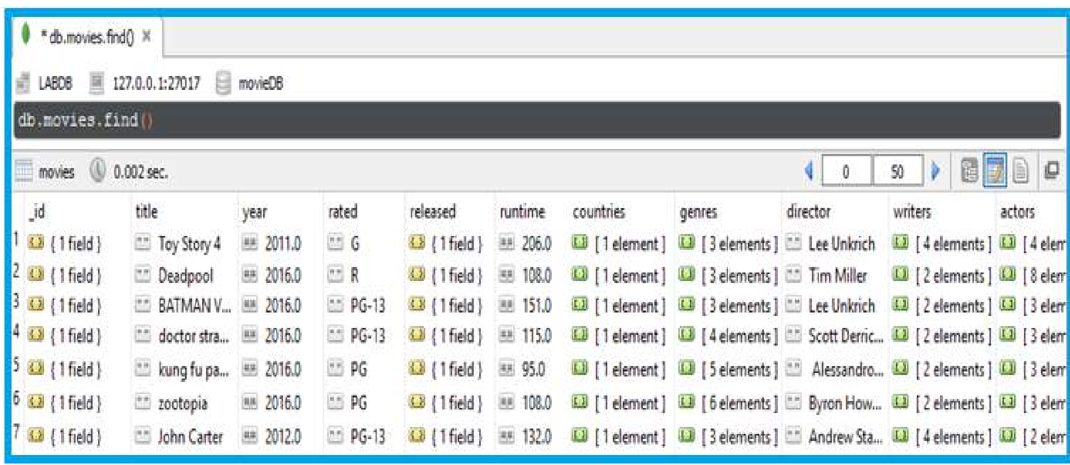
ნახ. 22.26

db.movies.find() ბრძანება გამოიტანს მიმდინარე ბაზის (movieDB) movies კოლექციის ყველა ჩანაწერს. შედეგის გამოტანა შესაძლებელია სამი სხვადასხვა ფორმატით:

- View results in tree mode - ხისებრი სტრუქტურა (ნახ. 22.27);
- View results in table mode - ცხრილის სტრუქტურა (ნახ. 22.28);
- View results in text mode – JSON სტრუქტურა (ნახ. 22.29).



ნახ.22.27. ხისებრი სტრუქტურა



	_id	title	year	rated	released	runtime	countries	genres	director	writers	actors
1	{ 1 field }	Toy Story 4	2011.0	G	{ 1 field }	206.0	[1 element]	[3 elements]	Lee Unkrich	[4 elements]	[4 elem
2	{ 1 field }	Deadpool	2016.0	R	{ 1 field }	108.0	[1 element]	[3 elements]	Tim Miller	[2 elements]	[8 elem
3	{ 1 field }	BATMAN V...	2016.0	PG-13	{ 1 field }	151.0	[1 element]	[3 elements]	Lee Unkrich	[2 elements]	[3 elem
4	{ 1 field }	doctor stra...	2016.0	PG-13	{ 1 field }	115.0	[1 element]	[4 elements]	Scott Derric...	[2 elements]	[3 elem
5	{ 1 field }	kung fu pa...	2016.0	PG	{ 1 field }	95.0	[1 element]	[5 elements]	Alessandro...	[2 elements]	[3 elem
6	{ 1 field }	zootopia	2016.0	PG	{ 1 field }	108.0	[1 element]	[6 elements]	Byron How...	[2 elements]	[3 elem
7	{ 1 field }	John Carter	2012.0	PG-13	{ 1 field }	132.0	[1 element]	[3 elements]	Andrew Sta...	[4 elements]	[2 elem

ნახ. 22.28. ცხრილის სტრუქტურა

```
db.movies.find()
```


```
/* 1 */
{
    "_id" : {
        "oid" : "5692a15524de1e0ce2dfcfa3"
    },
    "title" : "Toy Story 4",
    "year" : 2011.0,
    "rated" : "G",
    "released" : {
        "date" : "2010-06-18T04:00:00.000Z"
    },
    "runtime" : 206.0,
    "countries" : [
        "USA"
    ],
    "genres" : [
        "Animation",
        "Adventure",
        "Comedy"
    ],
    "director" : "Lee Unkrich",
    "writers" : [
        "John Lasseter",
        "Andrew Stanton",
        "Lee Unkrich",
        "Michael Arndt"
    ],
    "actors" : [
        "Tom Hanks",
        "Tim Allen",
        "Joan Cusack",
        "Ned Beatty"
    ],
 "plot" : "The toys are mistakenly delivered to a day-care
  center instead of the attic right before Andy leaves
 "poster" : "http://ia.media-imdb.com/images/M/MV5BMTgxOTY4Mjc
  0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
  "imdb" : {
      "id" : "tt0435761",
      "rating" : 8.4,
      "votes" : 500084.0
  },
  "tomato" : {
```


ნახ. 22.29. JSON სტრუქტურა

22.7.5.2. ბრძანებებისა და ოპერაციების მაგალითები MongoDB ბაზაში

- **ინფორმაციის ამოღების მაგალითები**

მონაცემთა ბაზის შექმნისა და მისი მონაცემებით შევსების შემდეგ (მაგალითად, movieDB - ფილმების მონაცემთა ბაზა) განვიხილოთ სხვადასხვა ოპერაციის მაგალითები ამ სისტემის გამოყენებით.

- მიმდინარე ბაზის არჩევა:

```
use movieDB
```

- ძიება ფილმის სათაურის მიხედვით:

```
db.movies.find({"title":'Deadpool'})
```

- ძიება ფილმის გამოშვების თარიღის მიხედვით:

```
db.movies.find( { year: { $gt: 2010 } } )
db.movies.find( { year: { $gt: 2010, $lt: 2012 } } ),
```

სადაც $gt და $lt შედარების ოპერატორებია (Greater Then, Less, Then)

- ფილმის ძიება IMDB რეიტინგის მიხედვით (ამ მაგალითში ჩანს, როგორ უნდა მივწვდეთ ჩადგმული დოკუმენტის ველებს)

```
db.movies.find(
  {
     "imdb.rating": {$gt:8}
  }
)
```

- იმ ფილმების ძიება სადაც ფავორიტი მსახიობებიდან ერთ-ერთი მაინც თამაშობს:

```
db.movies.find(
  {
    actors: { $in: [ "Tom Hanks", "Gina Carano" ] }
  }
)
```

- ძიება უნიკალური ნომრის მიხედვით:

```
db.movies.find(
  {
    "_id.oid" : "5692a15524de1e0ce2dfcfa3"
  }
)
```

- ძიება ჩადგმული დოკუმენტების მასივში:

შემდეგი ფრაგმენტი დაგვიბრუნებს ყველა იმ ფილმს, რომელზეც დატოვებულია კონკრეტული მომხმარებლის კომენტარი:

```
db.movies.find(
 {
   reviews: {
        $elemMatch: {
          name: "parvesh"
        }
   }
 }
)
```

- მოწესრიგება (სორტირება):

```
db.movies.find().sort( { year: 1 } )
```

- რაოდენობის შეზღუდვა:

```
db.movies.find().sort( { year: 1 } ).limit(1)
```

ამ ბრძანებით შეგვიძლია ამოვიღოთ ინფორმაცია ყველაზე ძველ ფილმზე

- სასურველი ველების პროექცია:

```
db.movies.find( { year: { $gt: 2010 } }, { title: 1, year: 1 } )
{ title: 1, year: 1 }
```

ამ სახის ფორმატით შევძლებთ მხოლოდ სასურველი ველების დაბრუნებას (ნახ:22.30)

db.movies.find({ year: { $... × | * db.movies.find() × | * db.bios.find({ _id: { $i... ×

LABDB | 127.0.0.1:27017 | movieDB

```
db.movies.find( { year: { $gt: 2010 } }, { title: 1, year: 1 } )
```

movies | 0.001 sec. | 0 | 50

```
/* 1 */
{
    "_id" : {
        "oid" : "5692a15524de1e0ce2dfcfa3"
    },
    "title" : "Toy Story 4",
    "year" : 2011.0
}

/* 2 */
{
    "_id" : {
        "oid" : "589cbda9c0b9fec62febf274"
    },
    "title" : "Deadpool",
    "year" : 2016.0
}

/* 3 */
{
    "_id" : {
        "oid" : "589cc22cc0b9fec62febf275"
    },
    "title" : "BATMAN V SUPERMAN: DAWN OF JUSTICE",
    "year" : 2016.0
}

/* 4 */
{
    "_id" : {
        "oid" : "589cc417c0b9fec62febf276"
    },
    "title" : "doctor strange",
    "year" : 2016.0
}

/* 5 */
{
```

ნახ. 3.11

ანალოგიურად, შეგვიძლია ამოვიღოთ მხოლოდ სასურველი ველები ჩადგმული დოკუმენტების ძებნითაც:

```
db.movies.find( { "imdb.rating": {$gt:8}}, { title: 1, "imdb.rating": 1} )
{ title: 0, "imdb.rating": 0}
```

ამ შემთხვევაში მივიღებდით ყველა ველს, ამ ორის გარდა:

```
/* 1 */
{
   "_id" : {
      "oid" : "5692a15524de1e0ce2dfcfa3"
   },
   "title" : "Toy Story 4",
   "imdb" : {
      "rating" : 8.4
   }
}
/* 2 */
{
   "_id" : {
      "oid" : "589cbda9c0b9fec62febf274"
   },
   "title" : "Deadpool",
   "imdb" : {
      "rating" : 8.1
   }
}
/* 3 */
{
   "_id" : {
      "oid" : "589cc846c0b9fec62febf278"
   },
   "title" : "zootopia",
   "imdb" : {
      "rating" : 8.1
   }
}
```

➢ ოპერაციების მაგალითები წიგნების კოლექციისათვის

- ერთმანეთში ჩადგმული დოკუმენტები

```
var book = {
    title: 'MongoDB: The Definitive Guide',
    authors: [
        { lastName: 'Chodorow', firstName: 'Kristina' },
{ lastName: 'Dirolf', firstName: 'Michael' }
    ],
    tags: ['NoSQL', 'Database', 'BigData', 'Programming'],
    pages: 195,
    published: 2010
};
```

- Insert

```
db.books.save(book);
// primary key '_id'
// generated by client driver
// e.g. 4fba97070f318c1e73763350
book._id;
```

- Update

```
db.books.update({title: /Good Parts/},
                {$inc: {pages: 3}});
db.books.update({title: /in Action/},
{$set: {publisher: 'Manning'}},
false, true);
db.books.update({},
{$addToSet: {tags: 'Programming'}},
false, true);
```

- Delete:

```
db.books.remove({_id: mybook._id});  // წაშლა კონკრეტული
                                     // იდენტიფიკატორის   მიხედვით
db.books.remove({tags: 'Cooking'});
db.books.remove();              // კოლექციის გასუფთავება
```

- **მარტივი მოთხოვნები**

```
db.books.find();     // შედეგში აისახება ყველა წიგნისგან შემდგარი სია
db.books.count();  // შედეგად დაბრუნდება დოკუმენტების საერთო
                   // რაოდენობა
db.books.find().count();  // შედეგად დაბრუნდება დოკუმენტების
                          // საერთო რაოდენობა
 db.books.findOne({  // უნიკალური იდენტიფიკატორის მიხედვით ძებნა
_id: ObjectId("4fba97190f318c1e73763353")
});
    // ძიება სხვადასხვა პარამეტრების მიხედვით
db.books.find({ title: 'JavaScript Patterns' });
db.books.find({ title: /^MongoDB/ });
db.books.find({ title: /^MongoDB/,  pages: {$gt: 200} });
// ძიება ჩადგმულ დოკუმენტებიან სტრუქტურაში
db.books.find({
'authors.lastName': 'Katz'});
db.books.find({
'authors.lastName':
   {$in: ['Katz', 'McCaw']} });
 db.books.find({
$or: [
    {'authors.lastName': 'Katz'},
                {'authors.lastName': 'McCaw'}
            ]});
// პროექცია, სორტირება, ზღვარი
db.books.
find({/* all */},
{title: 1, pages: 1}).
sort({title: 1}).
limit(4);
```

- **ბრძანებები:**

```
db.runCommand({count: 'books',
query: {published: 2012}});
db.runCommand({distinct: 'books', key:'tags'});
db.runCommand({group: {
```

```
ns: 'books',
key: { published: true },
$reduce: function (obj, prev) {
            prev.pages += obj.pages;
},
initial: { pages: 0 }
}});

db.runCommand({ dropDatabase: 1 });
db.runCommand({ getLastError: 1 });
db.runCommand({ serverStatus: 1 });
db.runCommand({ shutdown: 1 });
```

- Indexes

```
db.books.ensureIndex({"title": 1}, {unique: true});
db.books.ensureIndex({"authors.lastName": 1});
db.books.ensureIndex({"tags": 1});
db.books.getIndexes();
db.books.dropIndex('title_1');
```

- Import / Export

```
mongoexport -d test -c books > mongo.books.txt
mongo test --eval "db.books.remove()"
 mongoimport -d test -c books --file books.txt
mongoexport -d test -c books --jsonArray > books.json
mongoimport -d test -c books --jsonArray < books.json
```

XXIII თავი
მონაცემთა ბაზების მართვის სისტემები

23.1. მონაცემთა ბაზების მართვის სისტემა Ms SQL Serever

გამოყენებითი სფეროს აპლიკაციის მონაცემთა ბაზა (Database) შედგება ურთიერთდაკავშირებული ცხრილებისაგან (Tables). ეს კავშირები ძირითადად რეალიზებულია პირველადი (Primary key - PK) და მეორეული (Foreign key - FK) გასაღებებით. შესაძლებელია ინდექსების გამოყენებაც (ინდექსური ფაილების შესაქმნელად).

ჩვენ ვგულისხმობთ, რომ მკითხველს აქვს საწყისი წარმოდგენა რელაციურ ბაზებთან სამუშაოდ და, კერძოდ, MsSQLServer-თან. ამიტომ აქ ჩვენ მოკლედ განვიხილავთ ზოგიერთ საკითხს (გამეორების თვალსაზრისით), რაც დაგვჭირდება პროგრამული აპლიკაციების გასამართად, მომხმარებელთა ინტერფეისების სამუშაოდ მონაცემთა ბაზებთან.

23.1.1. მონაცემთა ძირითადი ტიპები

მონაცემთა ბაზის ცხრილი (Table), როგორც ვიცით სვეტების (ატრიბუტების) და სტრიქონებისგან (კორტეჟებისგან) შედგება. იგი სტრუქტურაა, რომლის ელემენტები შეიძლება მონაცემთა სხვადასხვა ტიპით განისაზღვროს (ნახ.23.1).

R1=„Student"

ID მთელრიცხვა	A1 სტრიქონული	A2 ნამდვილ-რიცხვა	. . . სხვა	An თარიღი
101	აბესაძე ავთანდილ	19		21/05/76
102	ბურძგლა დიმიტრი	27		03/01/59
115	. . .	. . .	. . .	. . .

MsSQL Server-ში მონაცემთა ტიპები მრავალფეროვანია [10]. ერთი სვეტის (ატრიბუტის) ყველა მნიშვნელობა ერთი ტიპისაა. გამონაკლისია მხოლოდ SQL_VARIANT ტიპი, რომელშიც შესაძლებელია ერთდროულად რამდენიმე ტიპის მონაცემის შენახვა. მაგალითად, რიცხვითი, სტრიქონული ან თარიღის ტიპის მონაცემები. მონაცემთა ტიპები იყოფა შემდეგ კატეგორიებად:

- რიცხვითი ტიპები;
- სიმბოლური ტიპები;
- დროითი ტიპები (თარიღი და დრო);
- სხვა ტიპები.

რიცხვითი ტიპები **ცხრ.23.1**

მონაცემთა ტიპი	ბაიტი	აღწერა
INT	4	მთელრიცხვა: -2^{31}-:-$2^{31}-1$
SMALLINT	2	მთელრიცხვა: -2^{15}-:-2^{15}-1
TINYINT	1	მთელრიცხვა: 0 - 255
BIGINT	8	მთელრიცხვა: -2^{63}-:-2^{63}-1
DEC(p,[s]) ან NUMERIC	5-:-17	p -სიზუსტის წილადი -2^{38} +1 238 -1 s –თანრიგები წერტილის მარჯვნივ
		მცოცავწერტილიანი
REAL		დადებითი 2,23E –308 -:- 1,79E +308, უარყოფ. –1,18E –38 -:- –1,18E +38
		მცოცავწერტილიანი
FLOAT[(p)]	4	if p < 25
	8	if p >= 25
MONEY	8	ფულის ტიპი: -2^{63} -:- 2^{63} -1
SMALLMONEY	4	ფულის ტიპი: -2^{31} -:- 2^{31} -1

სიმბოლური ტიპები **ცხრ. 23.2**

მონაცემთა ტიპი	ბაიტი	აღწერა
CHAR[(n)]	min 1	ფიქსირებული სიგრძის სტრიქონი n = 1-:- 8000 (სიმბოლო)
VARCHAR[(n)]	min 1	ცვლადი სიგრძის სტრიქონი 0 < n < 8000 (სიმბოლო)
NCHAR[(n)]	min 2	ფიქს.სიგრძის Unicode სტრიქონი n = 1-:- 4000 (სიმბოლო)
NVARCHAR[(n)]	min 2	ცვლადი სიგრძის Unicode სტრიქონი 0 < n < 4000 (სიმბოლო)

დროითი ტიპები (თარიღი და დრო) **ცხრ.23.3**

მონაცემთა ტიპი	ბაიტი	აღწერა
DATETIME	4	თარიღი და დრო დიაპაზონით: 01/01/1753 -:- 31/12/9999
SMALLDATETIME	2	თარიღი და დრო დიაპაზონით: 01/01/1900 -:- 06/06/2079
DATE	3	თარიღი დიაპაზონით: 01/01/0001-:-31/12/9999 მაგ.: ‘mmm dd yyyy’ (‘Jan 07 2016’).

		SET DATEFORMAT-ით იცვლება
TIME	3-:-5	დრო 100 ნანოსეკ. სიზუსტით. მაგ.: 'hh:mm' ('21:45')
DATETIME2	6-:-8	თარიღი და დრო დიდი სიზუსტით
DATETIMEOFFSET	6-:-8	თარიღი და დრო დროის სარტყელით

ორობითი და ბიტური ტიპები **ცხრ.23.4**

მონაცემთა ტიპი	ბაიტი	აღწერა
BINARY[(n)]	=n	ფიქსირ. სიგრძის ბიტების სტრიქონი 0 < n < 8000
VARBINARY[(n)]	n-მდე	ცვლადი სიგრძის ბიტების სტრიქო-ნი 0 < n < 8000
BIT	1 ბიტი	ლოგიკური მნიშვნელობა: FALSE, TRUE და NULL

დიდი ობიექტების ტიპები **ცხრ.23.5**

მონაცემთა ტიპი	აღწერა
VARCHAR(max)	LOB : ობიექტი 2GB - მდე
NVARCHAR(max)	LOB : ობიექტი 2GB - მდე
VARBINARY(max)	BLOB: ატრიბუტით FILESTREAM შეინახება მონაცემები NTFS ფაილურ სისტემაში

UNIQUEIDENTIFIER - გლობალური უნიკალური იდენტიფიკატორების (Global Unique Identifier, GUID) შენახვის ტიპი;

TIMESTAMP - დროითი შტამპი, რომელიც აფიქსირებს დროს სტრიქონის ყოველი ცვლილებისას;

HIERARCHYID - ინახავს სრულ იერარქიას (მაგ.: თანამშრომელთა იერარქია, კატალოგში ფოლდერების იერარქია და ა.შ.);

SPARSE – „მეჩხერი" სვეტების (შეიცავს ბევრ NUL მნიშვნელობას) შენახვის მოცულობის ოპტიმიზაცია.

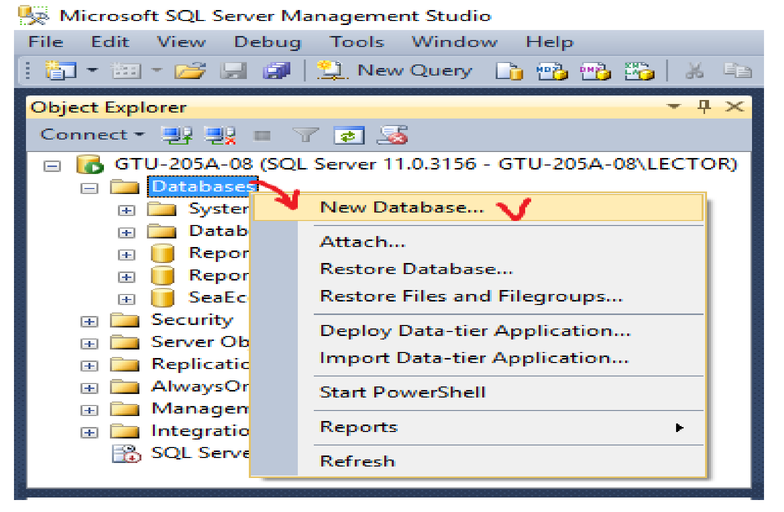
23.1.2. მონაცემთა ბაზის ობიექტების შექმნა

მონაცემთა ბაზის ობიექტები არის ფიზიკური (დისკებზე), როგორიცაა ფაილები და ფაილთა ჯგუფები, ან ლოგიკური - მომხმარებელთა წარმოდგენები მონაცემთა ბაზის შესახებ. ლოგიკური ობიექტების მაგალითებია ცხრილები (Tables), სვეტები (Columns) და ვირტუალური ცხრილები (Views - წარმოდგენები).

მონაცემთა ბაზის ობიექტი, რომელიც პირველ რიგში უნდა შეიქმნას, არის თვით მონაცემთა ბაზა. Database Engine კომპონენტი მართავს სისტემურ და მომხმარებელთა

მონაცემთა ბაზებს. სისტემური ბაზები იქმნება მონაცემთა ბაზის ინსტალირების დროს, ხოლო მომხმარებელთა ბაზები კი - თვით ავტორიზებული მომხმარებლის მიერ.

მონაცემთა ბაზის შექმნა ორი მეთოდითაა შესაძლებელი. პირველი, SQL Server Management Studio-ს დახმარებით, რომლითაც ხდება დიალოგურ რეჟიმში პროცესების წარმართვა (ნახ.23.1, 23.2.).



ნახ.23.1. ახალი ბაზის შექმნის დაწყება

Database name: University

Owner: <default>

Use full-text indexing

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize
University	Rows ...	PRIMARY	4	By 1 MB, Unlimited
University_log	Log	Not Applicable	1	By 10 percent, Unlimited

OK Cancel

ნახ.23.2. იქმნება University ბაზა

მეორე, Transact-SQL ენის CREATE DATABASE ინსტრუქციის დახმარებით:

```
CREATE DATABASE db_name
     [ON [PRIMARY] {file_spec1},.]
```

```
    [LOG ON {file_spec2},.]
        [COLLATE collation_name]
  [FOR {ATTACH ATTACH_REBUILD_LOG}]
```

ჩვენ აქ Transact-SQL ენას დეტალურად არ განვიხილავთ.

საჭიროა აღვნიშნოთ, რომ ერთ სისტემას შეუძლია 32 767 მონაცემთა ბაზის მართვა. ყველა ბაზა ინახება ფაილებში, რომლებიც შეიძლება ცხადად იყოს მითითებული ადმინისტრატორის მიერ ან არაცხადად იყოს წარმოდგენილი სისტემის მიერ. თუ CREATE DATABASE ინსტრუქცია შეიცავს ON პარამეტრს, მაშინ ბაზის ყველა ფაილი ცხადად არის გამოცხადებული.

დიდი ბაზებისთვის სასურველია ფაილთა ჯგუფების გამოყენება. ფაილი ინახავს ერთი ბაზის მონაცემებს. ფაილთა ჯგუფები საშუალებას იძლევა გადანაწილდეს მონაცემები სხადასხვა დისკებზე, შესრულდეს სარეზერვო დუბლირება და მონაცემთა ბაზის ნაწილის აღდგენის პროცედურა.

PRIMARY პარამეტრი მიუთითებს პირველ (მნიშვნელოვან) ფაილზე, რომელიც შეიცავს სისტემურ ცხრილებს და სხვა საჭირო შიგა ინფორმაციას ბაზის შესახებ.

COLLATE ოფციაში მიეთითება მოწესრიგების მიმდევრობა.

FOR ATTACH ოფცია მიუთითებს, რომ მონაცემთა ბაზა იქმნება უკვე არსებული ფაილების მიერთებით.

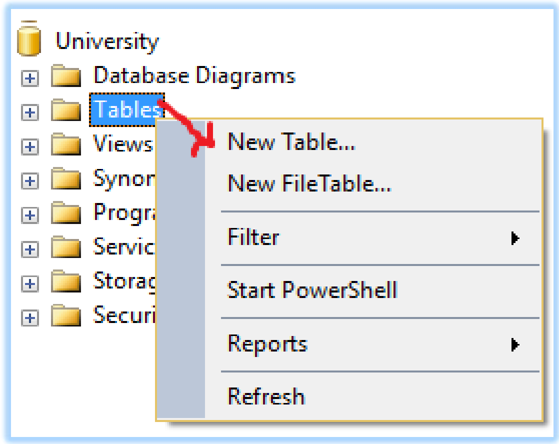
არსებული მონაცემთა ბაზის **მომენტალური სურათის** შექმნა CREATE DATABASE ინსტრუქციით (როცა ბაზაში დასრულებულია გარკვეული ტრანზაქციები და საჭიროა დუბლის შექმნა):

```
CREATE DATABASE database_snapshot_name
    ON (NAME = logical_file_name,
        FILENAME = 'C:\temp\file_name') [,...n ]
       AS SNAPSHOT OF source_database_name
```

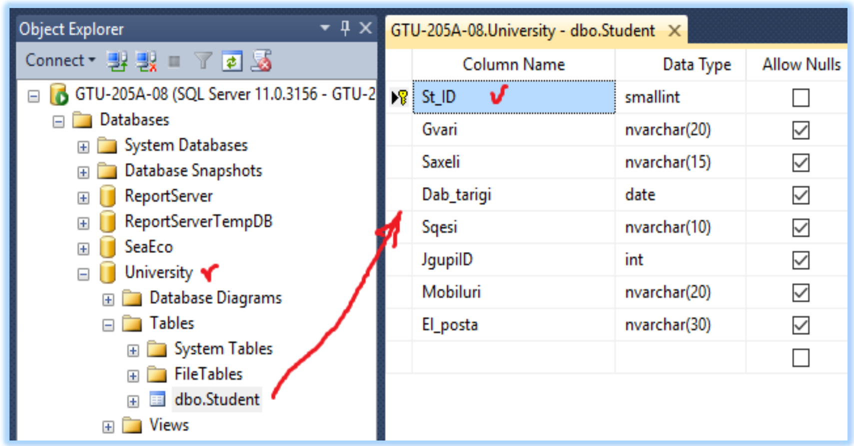
ამისათვის აქ გამოიყენება სტრიქონი AS SNAPSHOT OF. თავიდან უნდა შეიქმნას ბაზის მომენტალური სურათის შესანახი კატალოგი, მაგალითად, C:\temp\... . (NTFS - ფაილურ სისტემაში).

უნდა აღინიშნოს, რომ მომენტალური სურათი შეიცავს მონაცემთა ბაზის მხოლოდ შეცვლილ ნაწილს და ყოველი გადაღებული კადრი არ მოითხოვს დიდ დისკურ მეხსიერებას.

მონაცემთა ბაზის ცხრილის (Table) შექმნა ხორციელდება ინტერაქტიულ რეჟიმში SQL Server Management Studio-ს დახმარებით (ნახ.2.3). მომხმარებელს შეაქვს ცხრილის ატრიბუტები მათი ტიპების და სხვა მახასიათებლების მითითებით (ნახ.23.4). იქმნება რელაციური ცხრილის სტრუქტურა (მაგალითად, Student).

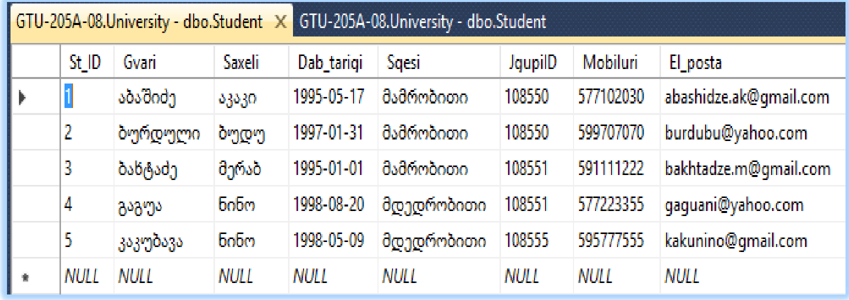


ნახ.23.3. ცხრილის შექმნის დაწყება



ნახ.23.4. Student ცხრილის შექმნა

23.5 ნახაზზე მოცემულია Student ცხრილის შევსების მაგალითი რამდენიმე სტრიქონით.



GTU-205A-08.University - dbo.Student | GTU-205A-08.University - dbo.Student

St_ID	Gvari	Saxeli	Dab_tariqi	Sqesi	JgupiID	Mobiluri	El_posta
1	აბაშიძე	აკაკი	1995-05-17	მამრობითი	108550	577102030	abashidze.ak@gmail.com
2	ბურდული	ბუდუ	1997-01-31	მამრობითი	108550	599707070	burdubu@yahoo.com
3	ბახტაძე	მერაბ	1995-01-01	მამრობითი	108551	591111222	bakhtadze.m@gmail.com
4	გაგუა	ნინო	1998-08-20	მდედრობითი	108551	577223355	gaguani@yahoo.com
5	კაკუბავა	ნინო	1998-05-09	მდედრობითი	108555	595777555	kakunino@gmail.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

ნახ.23.5. Student ცხრილის ფრაგმენტი

ცხრილების შექმნის მეორე გზა Transact-SQL ენის CREATE TABLE ინსტრუქციაა, რომლ;ის საბაზო ფორმა ასეთია:

CREATE TABLE table_name

(col_name1 type1 [NOT NULL | NULL)

[{, col_name2 type2 [NOT NULL | NULL]} ...])

აქ col_name ველის სახელია, ხოლო type - შესაბამისი ველის ტიპი.

ცხრილების მაქსიმალური რაოდენობა ერთ ბაზაში შეზღუდულია მონაცემთა ბაზის ობიექტების რაოდენობით, რომელთა რაოდენობა არ უნდა აღემატებოდეს 2 მილიარდს. აქ შედის ცხრილები (Tables), წარმოდგენები (Views), შენახვადი პროცედურები (Store procedures), ტრიგერები (Triggers) და შეზღუდვები (Constraints).

23.1.3. დეკლარაციული მთლიანობის შეზღუდვები და დომენები

მონაცემთა ბაზების მართვის სისტემის ერთ-ერთი ყველაზე მნიშვნელოვანი მოთხოვნაა მონაცემთა მთლიანობის უზრუნველყოფა. შეზღუდვებს, რომლებიც გამოიყენება მონაცემთა შესამოწმებლად მათი ცვლილების ან დამატებისას, უწოდებენ მთლიანობის უზრუნველყოფის შეზღუდვებს (Integrity constraints) [73].

მონაცემთა მთლიანობის უზრუნველყოფას გამოყენებით პროგრამაში ახორციელებს მომხმარებელი ან თვით მონაცემთა ბაზების მართვის სისტემა. ამ უკანასკნელ შემთხვევაში გვაქვს შემდეგი უპირატესობები:

- მაღლდება მონაცემთა საიმედოობა;
- მცირდება დაპროგრამების დრო;
- მარტივდება ტექნიკური მომსახურება.

მთლიანობის უზრუნველსაყოფად მონაცემთა ბაზების მართვის სისტემაას აქვს ორი ტიპის შეზღუდვა: დეკლარაციული შეზღუდვები და პროცედურული შეზღუდვები (ტრიგერებით რეალიზებადი).

დეკლარაციული შეზღუდვები განისაზღვრება DDL ენის CREATE TABLE და ALTER TABLE ინსტრუქციებით, სვეტების ან ცხრილების დონეზე [72]. ყოველ დეკლარაციულ შეზღუდვას ენიჭება სახელი. იგი ენიჭება ცხადად CONSTRAINT ოფციის გამოყენებით CREATE TABLE ან ALTER TABLE ინსტრუქციაში.

დეკლარაციული შეზღუდვები შეიძლება დაჯგუფდეს შემდეგ კატეგორიებში [72]:

- DEFAULT ;
- UNIQUE;
- PRIMARY KEY;
- CHECK;
- FOREIGN KEY.

დომენი (domain) მონაცემთა ბაზის ცხრილში სვეტის (ველის) დასაშვებ მნიშვნელობათა ერთობლიობაა. მათთვის, ჩვეულებისამებრ, გამოიყენება მონაცემთა ტიპები: INT, CHAR, DATE და სხვ. დომენის მთლიანობის უზრუნველყოფის მიზნით ასეთი მეთოდი არაა საკმარისი [73]. მაგალითად, უნივესრსიტეტის (University) მონაცემთა ბაზის ლექტორთა (Lector) ცხრილში ველისათვის ქალაქის კოდი (zip) შეიძლება როგორც SMALLINT, ისე CHAR(5) ტიპების გამოყენება, მაგრამ არც ერთი არ იქნება ზუსტი. პირველში რიცხვთა დიაპაზონი მოიცავს დადებითთან ერთად უარყოფით მნიშვნელობებსაც. მეორეში კი 5-სიმბოლოიანი მნიშვნელობა შეიძლება შედგებოდეა ალფაბეტის სიმბოლური, რიცხვითი და სხვა მნიშვნელობისგან. ჩვენ კი გვჭირდება მხოლოდ დადებით რიცხვთა დიაპაზონი ინტერვალში 00001 -:- 00099.

მეტნაკლები სიზუსტით დომენების მთლიანობის უზრუნვეყოფა შესაძლებელია CHECK შეზღუდვით ინსტრუქციაში CREATE TABLE ან ALTER TABLE.

Transact-SQL ენას ამ მიზნით დომენებისათვის აქვს მონაცემთა ტიპების ფსევდონიმების შექმნის შესაძლებლობა CREATE TYPE ინსტრუქციით.

მონაცემთა ტიპების ფსევდონიმი (alias data type) - მომხმარებლის მიერ განსაზღვრული სპეციალური ტიპია, რომელიც გამოიყენება არსებული საბაზო ტიპების საფუძველზე. მისი შექმნის ზოგადი სინტაქსი ასეთია:

```
CREATE TYPE [type_schema_name.] type_name
{[FROM base_type[(precision[ , scale ])] [NULL | NOT NULL]]
   | [EXTERNAL NAME assembly_name [.class_name]]}
```

ჩვენ მიერ ზემოთ განხილული კონკრეტული მაგალითისათვის გვექნება შემდეგი ტექსტი:

```
USE University;
CREATE TYPE zip
FROM SMALLINT NOT NULL;
```

ახლადშექმნილი ფსევდონიმური ტიპის გამოყენაბა შეგვიძლია ასე:

```
USE University;
CREATE TABLE Lector
   (LectorID INT NOT NULL,
    Lector_name CHAR(20) NOT NULL,
    city CHAR(20),
    zip_code ZIP,
    CHECK (zip_code BETWEEN 101 AND 199));
```

ამ მაგალითში Lector ცხრილის zip_code ველისათვის ტიპი განისაზღვრება zip ფსევდონიმური ტიპით. ამ ველის დასაშვებ მნიშვნელობათა შეზღუდული დიაპაზონი იქნება მთელი რიცხვები 101-199, რაც CHECK -ითაა მოცემული.

CREATE TYPE ინსტრუქციით კი ასეთი ტიპის შექმნა შემდეგნაირად ხორციელდება:

```
USE University;
CREATE TYPE Lector_table AS TABLE
    (gvari VARCHAR(30), xelpasi DECIMAL(8,2));
```

მომხმარებლის მიერ შექმნილი მონაცემთა ცხრილურ ტიპს Lector_table აქვს ორი ველი gvari და xelpasi.

ძირითადი სინტაქსური განსხვავება ცხრილურ და ფსევდონიმურ მონაცემთა ტიპებს შორის არის AS TABLE წინადადების არსებობა. მომხმარებლის მიერ განსაზღვრული ცხრილური ტიპები გამოიყენება მაშინ, როდესაც საჭიროა ცხრილურ მნიშვნელობათა პარამეტრების უკან დაბრუნება.

23.2. მონაცემთა ბაზების უსაფრთხოების სისტემა

მონაცემთა რელაციური ბაზების მართვის სისტემის ერთ-ერთი მნიშვნელოვანი კომპონენტია Database Engine, რომლის მომხმარებელთა ინტერფეისის დახმარებითაც საგრძნობლად მარტივდება სისტემასთან მუშაობა. აგრეთვე აქვს სხვადასხვა ინსტრუმენტი მონაცემთა ბაზის ობიექტების შესაქმნელად, დანართების ასაწყობად და სისტემური ადმინისტრირების ამოცანების სამართავად.

ჩვენ განვიხილავთ Database Engine კომპონენტის სისტემას მონაცემთა უსაფრთხოების უზრუნველყოფის მიზნით. ასეთია ავტორიზაციის (მონაცემთა ბაზასთან სანქცირებული მიმართვის) საკითხი, აუტენთიფიკაცის (მონაცემებთა მომხმარებელთა წვდომის პრივილეგიების) საკითხი, აგრეთვე მონაცემთა მოდიფიკაციის თანხლების შესაძლებლობები Database Engine კომპონენტით.

ამგვარად, მონაცემთა ბაზების უსაფრთხოების დაცვის ძირითადი კონცეფციებია:

- აუტენთიფიკაცია (მოხმარებლის სახელი, პაროლი);
- შიფრაცია (ინფორმაციის კოდირება, კრიპტოგრაფია);
- ავტორიზაცია (ბაზის რესურსების ფლობის ნებართვა);
- ცვლილებების მონიტორინგი (ბაზაში ყველა მიმართვის და ცვლილების ფიქსირება და დოკუმენტირება).

SQL Server სისტემაში მონაცემთა უსაფრთხოების მოდელი შედგება სამი განსხვავებული კატეგორიისაგან:

❖ პრინციპალები (principals) - სუბიექტებია, რომელთაც აქვთ წვდომის ნებართვა ბაზის განსაზღვრულ არსებთან (entities). არსებობს ასევე Windows-ჯგუფები და SQL Server-როლები. მომხმარებელს მიენიჭება შესაბამისი ჯგუფის ან როლის სააღრიცხვო ჩანაწერი, რაც მის უფლებებს განსაზღვრავს;

❖ დაცული ობიექტები (securables) - რესურსებია, რომლებზეც წვდომა რეგულირდება ბაზის ავტორიზაციის სისტემით;

❖ ნებართვები (permissions) - ყოველ დაცულ ობიექტს აქვს მასთან დაკავშირებული ნებართვა, რომელიც წარედგინება პრინციპალს.

23.2.1. აუტენთიფიკაცია

Database Engine კომპონენტის უსაფრთხოების სისტემა შედგება ორი განსხვავებული ქვესისტემისგან:

- Windows უსაფრთხოების სისტემები;
- SQL Server უსაფრთხოების სისტემები.

Windows უსაფრთხოების სისტემა განსაზღვრავს უსაფრთხოებას ოპერაციული სისტემის დონეზე. ესაა მეთოდი, რომლის დახმარებითაც მომხმარებელი შედის Windows სისტემაში.

SQL Server უსაფრთხოების სისტემა განსაზღვრავს დამატებით უსაფრთხოებას, რომელიც საჭიროა მონაცემთა ბაზის დონეზე. ესაა ხერხი, რომლითაც მომხმარებელი უკვე Windows სისტემაშია და ცდილობს მონაცემთა ბაზის სერვერთან დაკავშირებას.

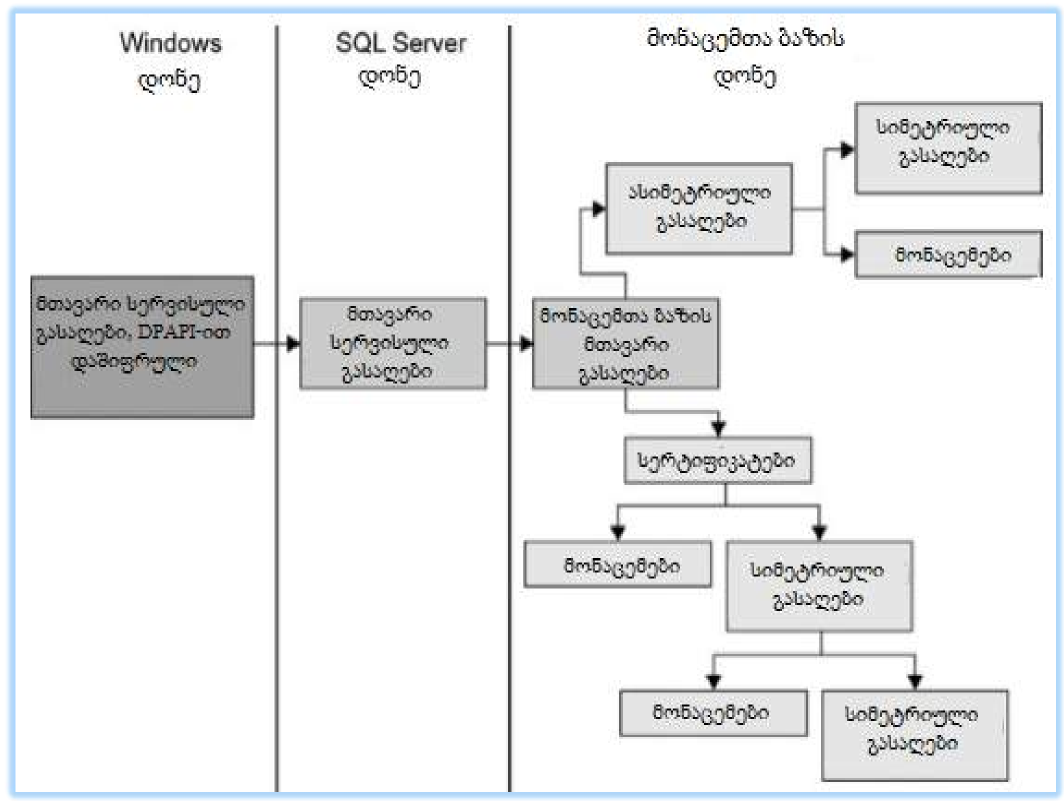
23.2.2. მონაცემთა შიფრაცია

შიფრაცია არის მონაცემების კოდირება (მაგალითად, კრიპტოგრაფიის მეთოდებით), რომლის დროსაც გაუგებარი ხდება მათი შინაარსი. Database Engine კომპონენტი უზრუნველყოფს მონაცემთა დაცვას შიფრაციის იერარქიული დონეების მიხედვით და გასაღებების მართვის ინფრასტრუქტურით. შიფრაციის ყოველი დონე იცავს მის მომდევნო დონეს, იყენებს სერტიფიკატების, სიმეტრიული და ასიმეტრიული გასაღებების კომბინაციას (ნახ.23.6).

მთავარი სერვისული გასაღები მართავს ყველა დანარჩენ გასაღებს და სერტიფიკატებს. იგი იქმნება ავტომატურად Database Engine კომპონენტის დაყენებისას. ეს გასაღები დაშიფრულია API- Windows მონაცემთა დაცვის ინტერფეისის დახმარებით (DPAPI – Data Protection API) [11].

მონაცემთა ყოველ ბაზას აქვს თავისი ერთი მთავარი გასაღები, რომელიც იქმნება CREATE MASTER KEY ინსტრუქციით. ეს გასაღები დაცულია სისტემის მთავარი სერვისული გასაღებით, რომელსაც შეუძლია ავტომატურად მისი გაშიფრვა.

მონაცემთა ბაზის მთავარი გასაღებით შესაძლებელია მომხმარებელთა გასაღებების შექმნა: სიმეტრიული, ასიმეტრიული და სერტიფიკატი. სიმეტრიულის დროს ორივე მხარეს (გადამცემი, მიმღები) აქვს ერთი გასაღები, ხოლო ასიმეტრიულის დროს - სხვადასხვა. პირველი უფრო მარტივია, მეორე კი - საიმედო.



ნახ.23.6. შიფრაციის იერქრქიული კომპონენტი [11]

SQL Server-ში გამოიყენება მონაცემთა შიფრაციის ორი ხერხი: შიფრაცია სვეტების დონეზე და გამჭვირვალე შიფრაცია (სიმეტრიული გასაღების საფუძველზე). უფრო დეტალურად შეიძლება ამ საკითხების გაცნობა [73,88].

XXIV თავი
სივრცითი მონაცემთა ტიპები SQL Server მონაცემთა ბაზაში

21-ე საუკუნის დასაწყისიდან ბიზნესის, გეოსისტემებისა და სხვა სფეროებში მზარდი მოთხოვნილება გაჩნდა და მნიშვნელოვანი განვითარება დაიწყო სივრცითი მონაცემების მოდელირების, მონაცემთა ბაზაში შენახვისა და მათი ასხვის ტექნოლოგიების ტექნოლოგიებმა. ამ პროცესებზე განსაკუთრებით დიდი გავლენა იქონია „მაიკროსოფთის" კორპორაციის Virtual Earth სისტემის და GPS ინსტრუმენტების (Global Positioning Systems – ნავიგაციის გლობალური სისტემა და ადგილმდებარეობის განსაზღვრა) გამოჩენამ [11].

მოდელები. სივრცითი მონაცემების ასახვის მიზნით დღეისათვის გამოიყენება ორი სახის მოდელი:

- გეოდეზიური სივრცითი მოდელები და
- ბრტყელი სივრცითი მოდელები.

პლანეტები, როგორც რთული ობიექტები შეიძლება წარმოდგენილ იყოს სფეროიდების სახით. ამის ერთ-ერთი კარგი მაგალითია გლობუსი – დედამიწის მოდელი. მისი ზედაპირის კონკრეტული წერტილი აღიწერება განედით (პარალელები, ეკვატორის ჩრდილოეთით და სამხრეთით) და გრძედით (მერიდიანები, მაგალითად, 0-ოვანი მერიდიანის დასავლეთით ან აღმოსავლეთით). ასეთ მოდელებს გეოდეზიურს უწოდებენ.

ბრტყელ სივრცით მოდელებში, მაგალითად დედამიწის ასახვის მიზნით, გამოიყენება ორგანზომილებიანი რუკები.

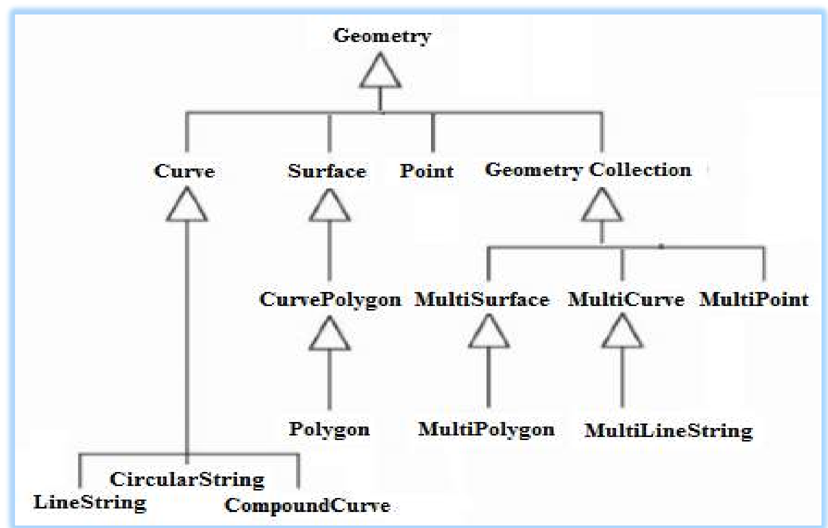
ამგვარად, ჩვენ განვიხილავთ სივრცითი მონაცემების ორ ტიპს:

- GEOMETRY – ბრტყელი სივრცითი მოდელი;
- GEOGRAPHY – გეოდეზიური მოდელი.

მონაცემთა ასეთი ტიპები რეალიზებულია და გამოიყენება მონაცემთა ბაზების მართვის სისტემის, SQL Server 2012/14 ვერსიებში.

24.1. მონაცემთა ტიპი GEOMETRY

ტერმინი - გეომეტრიული ობიექტი შემოიტანა OGC (Open Geospatial Consortium) კონსორციუმმა, რათა შესაძლებელი ყოფილიყო ისეთი სივრცითი თვისებების ასახვა, როგორიცაა წერტილები და წრფეები. ამიტომაც გეომეტრიული ობიექტი მონაცემებს წარმოადგენს ორგანზომილებიან სივრცეში წერტილების, წრფეებისა და მრავალ-კუთხედების სახით. გეომეტრიული ობიექტის მონაცემთა ტიპს აქვს ქვეტიპები, როგორც ეს 24.1 ნახაზზეა ნაჩვენები [11].



ნახ.24.1. ტიპების იერარქია GEOMETRY ფესვური ტიპით

როგორც ნახაზიდან ჩანს, ქვეკლასები დაყოფილია ორ კატეგორიად: საბაზო გეომეტრიულ ქვეკლასებად და ერთგვაროვან კოლექციათა ქვეკლასებად.

საბაზო გეომეტრიული ქვეკლასები შედგება Point, LineString და Polygon ქვეკლასებისაგან, ხოლო ერთგვაროვანი კოლექციები - MultiPoint, MultiLineString და MultiPolygon ქვეკლასებისგან, მათ საკუთარი თვისებებიც აქვს.

ახლა განვიხილოთ სივრცითი მონაცემების აღნიშნული ტიპების მოკლე დახასიათება:

- **Point (წერტილი)** – არის ორგანზომილებიანი გეომეტრიული ობიექტი კოორდინატთა X და Y მნიშვნელობებით, ამ ტიპის ეგზემპლარს შეიძლება ჰქონდეს ორი დამატებითი კოორდინატა: დონე (Z) და ზომა (M). Point ტიპის ეგზემპლარები გამოიყენება რთული სივრცითი ტიპების ასაგებად;
- **MultiPoint (მრავალწერტილოვანი)** – არის კოლექცია ნულოვანი ან მეტი რაოდენობის წერტილებისა. არაა აუცილებელი მრავალწერტილოვან ტიპში წერტილები იყოს განსხვავებული;
- **LineString (ტეხილი ხაზი)** – არის გეომეტრიული ობიექტი, რომელსაც აქვს სიგრძე და შეინახება წერტილების მიმდევრობით, რომელიც განსაზღვრავს ხაზის გზას. ტეხილ ხაზს აქვს საკონტროლო წერტილები (გადახრის ადგილი). ტეხილი ხაზი მარტივია (simple), თუ იგი არ კვეთს თავის თავს. თუ ტეხილი ხაზის საწყისი და საბოლოო წერტილები ემთხვევა, იგი ჩაკეტილია (closed). ჩაკეტილ, მარტივ ტეხილს უწოდებენ რგოლს (ring).
- **MultiLineString (მრავალტეხილი ხაზები)** – არის კოლექცია ნულოვანი ან მეტი რაოდენობის ტეხილი ხაზის.

- **Polygon (მრავალკუთხედი)** – არის ორგანზომილებიანი გეომეტრიული ფიგურა ზედაპირით. მრავალკუთხედი ინახება მიმდევრობითი წერტილების სახით, რომლებითაც განისაზღვრება მისი გარეგანი შემომსაზღვრელი პერიმეტრი (ring) და შიგა მრავალკუთხედები (ნული ან რამდენიმე). გარე და შიგა ფიგურები იძლევა მრავალკუთხედის საზღვრებს, ხოლო სივრცე მათ შორის კი - განსაზღვრავს შიგა ნაწილს.
- **MultiPolygon ()** – მრავალკუთხედების კოლექციაა (0 ან მეტი).
- **GeometryCollection** – გეომეტრიული ეგზემპლარების კოლექციაა 0 ან მეტი გეომეტრიული ობიექტით. ამ ტიპის ეგზემპლარი შეიძლება შეიცავდეს GEOMETRY ტიპის ნებისმიერ ქვეტიპს.

24.1 ნახაზიდან ჩანს, რომ არსებობს სამი ქვეტიპი, რომლისთვისაც შეიძლება ეგზემპლარების შექმნა: CircularString, CompoundCurve და CurvePolygon. ეს ქვეტიპები ახალია SQL Server 2012 -ში და გამოიყენება სივრცით მონაცემებთან სამუშაოდ.

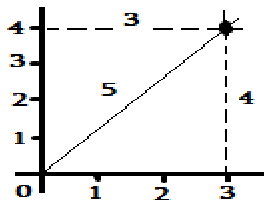
24.2. მონაცემთა ტიპი GEOGRAPHY

როგორც აღინიშნა, გეომეტრიული ტიპი მონაცემებს ინახავს X და Y კოორდინატებით, ხოლო გეოგრაფიული მონაცემები კი – GPS სისტემის განედისა და გრძედის გამოყენებით. გრძედი არის ჰორიზონტალური კუთხე (-180^0-:- $+180^0$) დიაპაზონში, ხოლო განედი – ვერტიკალური კუთხე (-90^0-:- $+90^0$) დიაპაზონში.

გეომეტრიულისგან განსხვავებით გეოგრაფიული ტიპისთვის უნდა მიეთითოს კონკრეტული კოორდინატთა სივრცითი სისტემა შესაბამისი მთელრიცხვა იდენტიფიკატირით SRID (Spatial Reference ID). მონაცემთა GEOMETRY ტიპის ეგზემპლარები გამოყენება აგრეთვე GEOGRAPHY ტიპისათვის.

განსხვავება GEOMETRY და GEOGRAPHY ტიპებს შორის, როგორც ადრე აღვნიშნეთ ისაა, რომ გეომეტრიული მონაცემები გამოიყენება ბრტყელ სივრცით მოდელებში, ხოლო გეოგრაფიულ კი გეოდეზიურ მოდელებში.

ძირითადი განსხვავება სივრცით მონაცემთა ამ მოდელებს შორის ისაა, რომ GEOMETRY ტიპის მონაცემებისათვის მანძილები და ფართობები მიეთითება განზომილების იმავე ერთეულებში, როგორც ეს გამოიყენებოდა ეგზემპლარებია კოორდინატებში. მაგალითად, მანძილი ორ (0,0) და (3,4) წერტილს შორის იქნება 5 ერთეული (ნახ.24.2). მართკუთხა სამკუთხედის ჰიპოტენუზა – პითაგორას თეორემის საფუძველზე.



ნახ.24.2. ორ წერტილს შორის სიგრძე
გეომეტრიულ კოორდინატებში

GEOGRAPHY ტიპის მონაცემებისათვის გამოიყენება ელიფსოიდური კოორდინატები, რომლებიც იზომება განედისა და გრძედის კუთხეებში. ამასთანავე, ამ ტიპის მონაცემებზე გაითვალისწინება გარკვეული შეზღუდვები, მაგალითად, ასეთი ტიპის ყოველი ეგზემპლარი უნდა მოთავსდეს ერთი ნახევარსფეროს შიგნით.

მონაცემთა გარე ფორმატები. SQL Server მონაცემთა ასახვის მიზნით იყენებს სამი ტიპის გარე ფორმატს მათი რეალიზაციის ფორმისაგან დამოუკიდებლად:

- WKT (Well-Known Text – ცნობილი ტესტური ფორმატი) –გამოიყენება სივრცით სტრუქტურათა სივრცითი კოორდინატების სისტემების ასახვისათვისა და გარდაქმნებისათვის კოორდინატთა სისტემებს შორის;
- WKB (Well-Known Binary – ცნობილი ორობითი ფორმატი) – არის WKT-ს ორობითი ეკვივალენტი;
- GML (Geography Markup Language – გეოგრაფიული ფორმატირების ენა) – არის XML ენის გრამატიკა, განსაზღვრული OGC კონსორციუმის მიერ, გეოგრაფიული თვისებების გამოსახვისათვის.

ესაა მონაცემთა გაცვლის ღია ფორმატი გეოგრაფიული ტრანზაქციების შესასრულებლად ინტერნეტში.

საილუსტრაციოდ განვიხილოთ WKT ფორმატის მაგალითები:

– POINT(3,4) – მნიშვნელობები მიუთითებს X და Y კოორდინატებს;

– LINESTRING(0 0, 3 4) – პირველი ორო რიცხვი საწყისი წერტილის X და Y კოორდინატების, ხოლო მესამე და მეოთხე – ბოლო წერტილისა;

– POLYGON(300 0, 150 0, 150 150, 300 150, 300 0) – რიცხვების ყოველი წყვილი არის წერტილი მრავალკუთხედზე. ჩვენ შემთხვევაში პირველი და ბოლო წერტილები ემთხვევა.

24.3. სივრცით მონაცემებთან მუშაობა

SQL Server -ში სივრცითი მონაცემებისათვის გამოიყენება, როგორც აღვნიშნეთ, ორი ტიპი: GEOMETRY და GEOGRAPHY. ესაა მომხმარებლის მიერ განსაზღვრული მონაცემთა ტიპები, რომლებიც რეალიზდება SQL Server დანართის სახით CLR გარემოს გამოყენებით. ამ ტიპებს აქვს ქვეტიპები, რომლებიც, რომლებიც შეიძლება იყოს ეგზემპლარირებადი ან არაეგზემპლარირებადი.

ეგზემპლარირებადი ქვეტიპების შემთხვევაში შესაძლებელია ეგზემპლარების შექმნა და მათთან მუშაობა. ეს ეგზემპლარები შეიძლება შენახული იყოს როგორც ცხრილის სვეტები, ან როგორც ცვლადების მნიშვნელობები ან პარამეტრები.

არაეგზემპლარირებადი ქვეტიპებისათვის ეგზემპლარების შექმნა არ შეიძლება. კლასთა იერარქიაში ფესვური (აბსტრაქტული) კლასი არაეგზემპლარირებადია.

24.3.1. GEOMETRY ტიპის მონაცემებთან მუშაობა

მაგალითი_1: ცხრილი „უალკოჰოლო_სასმელები“ შედგება სამი სვეტისაგან. პირველი - id გენერირდება სისტემის მიერ, მეორე - name გამოიყენება სამელის დასახელების შესანახად, მესამე - shape შეიცავს ინფორმაციას ბაზრის სფეროს ფორმის შესახებ, რომელშიც მყიდველები უფრო მეტ უპირატესობას ანიჭებენ რომელიმე კონკრეტულ სამელს. პირველი სამი INSERT ინსტრუქცია ქმნის სამ სფეროს, რომლებშიც კონკრეტული სასმელია პრიორიტეტული. ყოველი ეს სამი სფერო არის მრავალკუთხედი. მეოთხე INSERT ინსტრუქცია სვამს წერტილს, ვინაიდან ეს სპეციფიკური სასმელი იყიდება მხოლოდ ერთ ადგილას.

24.1 ლისტინგზე მოცემულია ცხრილის აგების Transact_SQL პროგრამის ტექსტი.

```
USE sample;
CREATE TABLE beverage_markets
 ( id INTEGER IDENTITY(1,1),
   name VARCHAR(25), shape GEOMETRY);
   INSERT INTO beverage_markets
   VALUES ('Coke1', GEOMETRY::STGeomFromText
    ('POLYGON ((0 0, 150 0, 150 150, 0 150, 0 0))', 0));
   INSERT INTO beverage_markets
   VALUES ('Pepsi', GEOMETRY::STGeomFromText
('POLYGON ((300 0, 150 0, 150 150, 300 150, 300 0))', 0));
   INSERT INTO beverage_markets
   VALUES ('7UP', GEOMETRY::STGeomFromText
('POLYGON ((300 0, 150 0, 150 150, 300 150, 300 0))', 0));
   INSERT INTO beverage_markets
   VALUES ('Almdudler', GEOMETRY::STGeomFromText
('POINT (50 0)', 0));
```

ლისტინგი 24.1

ლისტინგიდან ჩანს მონაცემთა გეომეტრიულ ტიპებთან მუშაობის მეთოდი `GEOMETRY::STGeomFromText()`. ეს სტატიკური მეთოდი გამოიყენება გეომეტრიული ფიგურების (მრავალკუთხედი, წერტილები) კოორდინატების ჩასასმელად.

ტიპს შეიძლება ჰქონდეს მეთოდების ორი განსხვავებული ჯგუფი: სტატიკური მეთოდები და კლასის ეგზემპლარების მეთოდები. პირველი გამოიყენება მთლიანი კლასისათვის, ხოლო მეორე - კლასის სათანადო ეგზემპლარებისათვის. ამ ჯგუფთა მეთოდების გამოძახება სხვადასხვანაირად ხდება.

სტსტიკურისთვის გამოიყენება სიმბოლო „ :: “, ხოლო ეგზემპლარის მეთოდისათვის კი – „ . “, მაგალითად:

GEOMETRY :: STGeomFromText;

@g.STContains;

გარდა აღნიშნულისა, გამოიყენება ასევე სამი სტატიკური მეთოდებიც:

- STPointFromText() – დააბრუნებს POINT მონაცემთა ტიპის ეგზემპლარის WKT წარმოდგენას;
- STLineFromText() – დააბრუნებს LINESTRING მონაცემთა ტიპის ეგზემპლარის WKT წარმოდგენას, სიმაღლის და ზომის მნიშვნელობათა დამატებით;
- STPolyFromText() – დააბრუნებს MULTIPOLYGON მონაცემთა ტიპის ეგზემპლარის WKT წარმოდგენას, სიმაღლისა და ზომის მნიშვნელობათა დამატებით;

სივრცით მონაცემებზე მოთხოვნები სრულდება ისევე, როგორც რელაციურ მონაცემებზე. ჩვენი მაგალითისათვის, დავუშვათ, რომ ვირჩევთ ინფორმაციას beverage_markets ცხრილის shape სვეტიდან.

24.2 ლისტინგში აგებულია მოთხოვნა, რომელიც განსაზღვრავს „არსებობს თუ არა Almdudler სასმელის გამყიდველი მაღაზია იმ რაიონში, სადაც პრიორიტეტულია სასმელი Coce“.

```
DECLARE @g geometry;
DECLARE @h geometry;
SELECT @h = shape FROM beverage_markets
       WHERE name ='Almdudler';
SELECT @g = shape FROM beverage_markets
       WHERE name = 'Coke';
SELECT @g.STContains(@h);
```

ლისტინგი 24.2

ეს მოთხოვნა აბრუნებს პასუხს 0-ს, რაც ნიშნავს, რომ ამ რაიონში ასეთი მაღაზია არაა.

STContains() მეთოდი დააბრუნებს 1-ს, თუ GEOMETRY ტიპის მონაცემის ერთი ეგზემპლარი შეიცავს ამავე ტიპის მეორე ეგზემპლარს, რომელიც მეთოდის პარამეტრშია მითითებული.

24.3 ლისტინგში ასახულია მოთხოვნა, რომელიც განსაზღვრავს „ shape სვეტის სიგრძეს და წარმოდგენას იმ მაღაზიისათვის, რომელიც ყიდის სასმელს Almdudler“.

```
SELECT id, shape.ToString() AS wkt, shape.STLength() AS length
    FROM beverage_markets
    WHERE name = 'Almdudler';
```

ლისტინგი 24.3

შედეგი გამოიტანება შემდეგი სახით:

Id	wkt	Length
4	POINT (500)	0

ამ მაგალითში STLength() მეთოდი აბრუნებს GEOMETRY მონაცემთა ტიპის ელემენტის საერთო სიგრძეს. 0 ნიშნავს, რომ მნიშვნელობა წერტილია. ToString() მეთოდი აბრუნებს სტრიქონს wkt ფორმატში, რომელიც შეიცავს მოცემული ეგზემპლარის ლოგიკურ წარმოდგენას.

24.4 ლისტინგში განიხილება STIntersects() მეთოდის გამოყენება. კერძოდ, მოთხოვნაა: „განისაზღვროს იკვეთება თუ არა რაიონი, რომელშიც იყიდება Coce, რაიონთან, რომელშიც იყიდება Pepsi”.

```
USE sample;
DECLARE @g geometry;
DECLARE @h geometry;
SELECT @h = shape FROM beverage_markets WHERE name = 'Coke';
SELECT @g = shape FROM beverage_markets WHERE name = 'Pepsi';
SELECT @g.STIntersects(@h);
```

ლისტინგი 24.4

STIntersects() მეთოდის გამოყენების შედეგად მიიღება 1 (TRUE), რაც ნიშნავს, რომ ეს ორი გეომეტრიული ფიგურა, რომლებიც ასახავს გაყიდვების სფეროებს, იკვეთება.

24.5 ლისტინგში განიხილება STIntersection() მეთოდის გამოყენება.

```
USE   sample;
DECLARE @poly1 GEOMETRY = 'POLYGON ((1 1, 1 4, 4 4, 4 1, 1 1))';
DECLARE @poly2 GEOMETRY = 'POLYGON ((2 2, 2 6, 6 6, 6 2, 2 2))';
DECLARE Sresult GEOMETRY;
SELECT Sresult = @poly1.STIntersection(@poly2);
SELECT Sresult.STAsText();
```

ლისტინგი 24.5

შედეგად მიიღება შემდეგი სტრიქონი (დამრგვალებული მნიშვნელობებით):

POLYGON ((22, 42, 44, 24, 22))

მეთოდი STIntersection() აბრუნებს ობიექტს, რომელიც ასახავს წერტილებს, სადაც GEOMETRY მონაცემთა ტიპის ეგზემპლარი იკვეთება ამავე ტიპის სხვა ეგზემპლართან. ამიტომაც. ამ მოთხოვნის საფუძველზე მიიღება მართკუთხედი, რომელზეც იკვეთება მრავალკუთხედი (გამოცხადებული @poly1 ცვლადით) მეორე მრავალკუთხედთან (გამოცხადებული @poly2 ცვლადით).

STAsText() მეთოდი აბრუნებს შედეგს wkt ფორმატში.

24.3.2. GEOGRAPHY ტიპის მონაცემებთან მუშაობა

GEOGRAPHY ტიპის მონაცემების დასამუშავებლადაც ასევე გამოიყენება იგივე სტატიკური და ეგზემპლარის მეთოდები, რაც იყო GEOMETRY მონაცემთა ტიპებისთვის. 2.6 ლისტინგში ნაჩენებია ერთი მაგალითი:

```
USE AdventureWorks;
SELECT SpatialLocation, City
    FROM Person.Address
```

ლისტინგი 24.6

ამ მოთხოვნის შესრულების შედეგები ნაჩვენებია ქვემოთ, SpatialLocation, City ცხრილში.

AdventureWorks მონაცემთა ბაზის Address ცხრილი შეიცავს SpatialLocation სვეტს, რომელიც არის GEOGRAPHY მონაცემთა ტიპის. ჩვენი მაგალითში მოითხოვება იმ თანამშრომელთა გეოგრაფიული მდებარეობის დადგენა, რომლებიც ცხოვრობენ ქალაქ დალასში.

SpatialLocation	City
0xE6100000010C4DD260393369404026C0A31BF73458C0	Dallas
0xE6100000010C10A810D1886240403A0F0653663158C0	Dallas
0xE6100000010C4346160AA26440406340F0E64F3 B58C0	Dallas
0xE6100000010C107E16DAAD6540403DA892EAD52C58C0	Dallas
0xE6100000010C8044A1422D5F4040F66D784F983758C0	Dallas
0xE6100000010C8E345943826A4040839B00B8E03358C0	Dallas
0xE6100000010CAA5BBD5FAB69404087866D198D3C58C0	Dallas

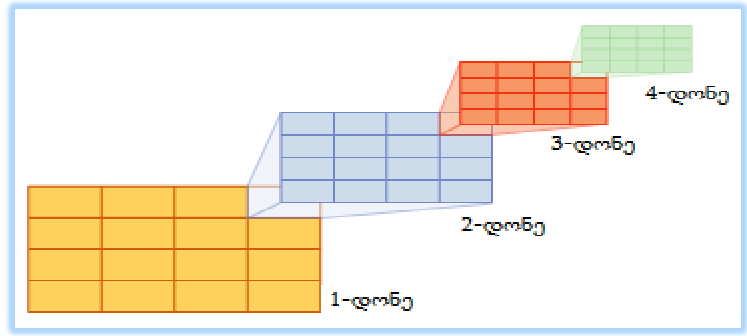
ამ ცხრილის SpatialLocation სვეტის მნიშვნელობებში ასახულია მისამართების გრძედი და განედი 16-ით კოდში თითოეული თანამშრომლისთვის ქალაქ დალასიდან.

შემდგომში ჩვენ განვიხილავთ ამ მაგალითს SQL Server Management Studio -ს გამოყენებით.

24.4. სივრცით ინდექსებთან მუშაობა

ინდექსირება, როგორც ცნობილია, გამოიყენება მონაცემებთან სწრაფი მიმართვის განსახორციელებლად. სივრცით მონაცემებთან მიმართვის დასაჩქარებლად კი საჭიროა ე.წ. სივრცითი ინდექსები, რომლებიც განისაზღვრება ცხრილის სვეტისთვის GEOMETRY ან GEOGRAPHY მონაცემთა ტიპით.

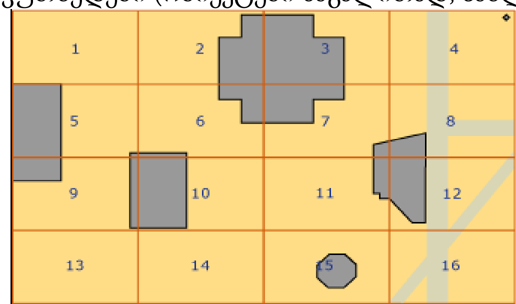
SQL Server-ში ასეთი ინდექსების ასაგებად გამოიყენება B-ხეები. ინდექსები აქ ასახავს ორ განზომილებას B-ხეების წრფივ რიგში. სივრცით ინდექსში მონაცემების მოთავსებამდე სისტემა ყოფს სივრცეს იერარქიულად ერთგვაროვანი წესით. ინდექსის შექმნის პროცესი სივრცეს ყოფს ოთხდონიანი ბადისებრი იერარქიის სახით (ნახ.24.8) [12,13]. ცხრილს უნდა ჰქონდეს კლასტერიზებული პირველადი გასაღები.



ნახ.24.8. სივრცითი ინდექსის ოთხდონიანი ბადისებრი იერარქია

ყველა დონის ბადეს უნდა ჰქონდეს უჯრედების ერთნაირი რაოდენობა (მაგალითად, 4x4 ან 8x8) და ყველა უჯრა ერთი ზომისაა. ნახაზზე ნაჩვენებია ბადის ზედა მარჯვენა უჯრის დეკომპოზიცია ბადის იერარქიის ზედა დონეებზე. დეკომპოზიცია სრულდება ყველა უჯრედისათვის. ამგვარად, მთლიანი სივრცის დეკომპოზიცია 4x4 ბადეზე ქმნის 65 536 უჯრედს მეოთხე დონეზე.

24.9 ნახაზზე ნაჩვენებია უჯრედების ნუმერაციის მაგალითი და ბადეზე განლაგებული მრავალკუთხედები (ობიექტები მაგალითად, სახლები, ქუჩები და ა.შ.).



ნახ.24.9. სივრცით ბადეზე ობიექტების განთავსება

ბადის სიმკვრივე განისაზღვრება უჯრედების რაოდენობით, რაც მეტია, მით უფრო მკვრივია იგი. მაგალითად, ბადე 8x8 (64 უჯრედით) უფრო მკვრივია, ვიდრე ბადე 4x4 (16 უჯრედით).

სივრცითი ინდექსების შესაქმნელად გამოიყენება ინსტრუქცია CREATE SPATIAL INDEX, რომლის GRIDS წინადადებით შეიძლება განსხვავებული სიმკვრივეების მითითება სხვადასხვა დონეზე (LOW - 4x4, MEDIUM – 8x8, HIGH. – 16x16).

გამოიყენება დამატებით ოფციები და წინადადებები:

- GEOMETRY_GRID – ეს წინადადება განსაზღვრავს გეომეტრიული ობიექტის ბადის ტესელაციის (tessellations) გამოყენებულ სქემას. ამ პროცესში ობიექტი თავსდება ბადისებრ იერარქიაში, უკავშირდება ბადის უჯრედებს, რომლებთანაც მას აქვს შეხება. სვეტს უნდა ჰქონდეს გეომეტრიული ტიპი;
- BOUNDING_BOX – ეს ოფცია განსაზღვრავს კომპლექტს ოთხი რიცხვითი მნიშვნელობით (მართკუთხედის ოთხი კოორდინატისთვის): Xmin და Ymin (ქვედა მარცხენა კუთხე) და Xmax და Ymax (ზედა მარჯვენა კუთხე). ეს პარამეტრი გამოიყენება მხოლოდ GEOMETRY_GRID წინადადებისთვის;
- GEOGRAPHY_GRID – ეს წინადადება განსაზღვრავს გეოგრაფიული ობიექტის ბადის ტესელაციის (tessellations) სქემას. სვეტს უნდა ჰქონდეს გეოგრაფიული ტიპი;

24.7 ლისტინგში ნაჩვენებია სივრცითი ინდექსის შექმნის მაგალითი beverage_markets ცხრილის shape სვეტისათვის.

სივრცითი ინდექსის შექმნა შეიძლება მახოლოდ მაშინ, თუ სივრცითი სვეტის მქონე ცხრილისათვის ცხადად არის განსაზღვრული პირველადი გასაღები. ამ მიზეზით ლისტინგის პირველ ALTER TABLE ინსტრუქციაში ეს შეზღუდვაა განსაზღვრული. შემდეგ ინსტრუქციაში CREATE SPATIAL INDEX იქმნება სივრცითი ინდექსი, გამოიყენება GEOMETRY_GRID.

```
USE sample;
GO
ALTER TABLE beverage_markets
      ADD CONSTRAINT prim_key PRIMARY KEY(id);
      GO
      CREATE SPATIAL INDEX i_spatial_shape
            ON beverage_markets(shape)
            USING GEOMETRY_GRID
             WITH (BOUNDING_BOX = (xmin=0, ymin=0, xmax=500, ymax=200),
             GRIDS = (LOW, LOW, MEDIUM, HIGH),
             PAD_INDEX = ON);
```

ლისტინგი 24.7

პარამეტრი BOUNDING BOX იძლევა საზღვრებს, რომელშიც მოთავსდება shape სვეტის ეგზემპლარი. GRIDS პარამეტრში მიეთითება ბადის სიმკვრივე ტესელაციის სქემის ყოველ დონეზე. პარამეტრი PAD_INDEX კავშირშია FILEFACTOR პარამეტრთან, რომელიც განსაზღვრავს პროცენტებში სივრცის თავისუფალ მოცულობას ინდექსის გვერდების საერთო მოცულობიდან. PAD_INDEX პარამეტრი კი მიუთითებს, რომ FILEFACTOR პარამეტრის მნიშვნელობა გამოიყენება როგორც ინდექსის გვერდებთან, ისე ინდექსში მონაცემთა გვერდებთან.

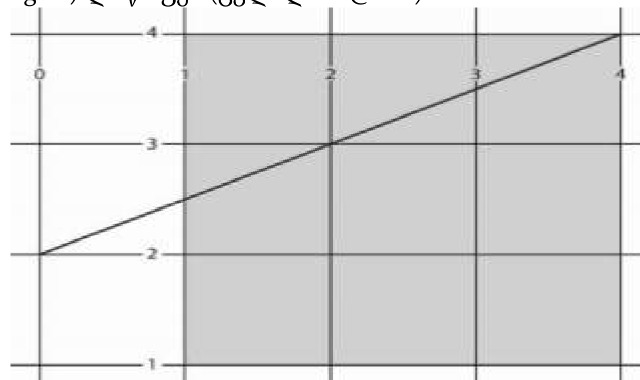
24.5. ინფორმაციის ასახვა სივრცითი მონაცემების შესახებ

SQL Server Management Studio 2012 ვერსიიდან ზევით გაფართოვდა სივრცითი მონაცემების ასახვის შესაძლებლობები გრაფიკულ ფორმატში. განვიხილოთ ეს საკითხები GEOMETRY და GEOGRAPHY მონაცემების ტიპებზე, შესაბამისად 24.8 და 24.9 ლისტინგების საფუძველზე.

```
USE sample;
DECLARE @rectangle1 GEOMETRY = 'POLYGON((1 1, 1 4, 4 4, 4 1, 1 1))';
DECLARE @line GEOMETRY = 'LINESTRING (0 2, 4 4)';
SELECT @rectangle1
UNION ALL
SELECT @line
```

ლისტინგი 24.8

ამ მოთხოვნის შესრულების შედეგების მისაღებად პანელზე ვირჩევთ Spatial Results ჩანართს (Results-ის მარჯვნივ). შედეგად ვიღებთ 24.10 ნახაზზე მიღებულ მართკუთხედს (ცვლადით @rectangle1) და წრფეს (ცვლადით @line).



ნახ.24.10. შედეგის გრაფიკული წარმოდგენა (24.8 ლისტინგის მიხედვით)

GEOMETRY ტიპის რამდენიმე ობიექტის ერთდროულად ასახვის მიზნით (ერთი ცხრილის რამდენიმე სტრიქონი), 24.8 ლისტინგში გამოყენებულია ორი SELECT ინსტრუქცია, რომლებიც ერთიანდება UNION ALL წინადადებით.

24.9 ლისტინგში ნაჩვენებია მოთხოვნა GEOGRAPHY მონაცემთა ტიპის გრაფიკული ასახვის სადემონსტრაციოდ.

```
USE AdventureWorks;
SELECT SpatialLocation, City
   FROM Person.Address
   WHERE City = 'Dallas';
```

ლისტინგი 24.9

აქაც ვიყენებთ შედეგების პანელზე Spatial Results ჩანართს და მივიღებთ 24.11 ნახაზსს.

ნახ.24.11. შედეგის გრაფიკული წარმოდგენა
(2.9 ლისტინგის მიხედვით)

24.6. სიახლეები სივრცით მონაცემებთან სამუშაოდ

SQL Server 2012 და უფრო ახალ ვერსიებში გაუმჯობესდა სივრცით მონაცემებთან მუშაობის შესაძლებლობები კერძოდ, შემოტანილია:

- წრის რკალის ახალი ქვეტიპები;
- ახალი სივრცითი ინდექსები;
- ახალი სისტემური შენახვადი პროცედურები, დაკავშირებული სივრცით მონაცემებთან.

განვიხილოთ თითოეული მათგანი:

➢ **წრის რკალის ახალი ქვეტიპები.** ეფუძნება ANSI SQL/MM სტანდარტს. წრის რკალი ზოგადად არის მრუდის ჩაკეტილი სეგმენტი ორგანზომილებიან სიბრტყეზე. ამგვარად, რკალი წრის სეგმენტია, რომელიც მოიცემა დამოუკიდებლად ან წრფის სეგმენტთან ერთად. ამასთანავე იგი შეიძლება გამოყენებულ იქნას როგორც საფუძველი ახალი ტიპის მრავალკუთხედისათვის, რომელიც შეიცავს ერთ ან მეტ რკალურ კომპონენტს.

წრის რკალები მხარდაჭერილია GEOMETRY და GEOGRAPHY მონაცემთა ტიპებით და განისაზღვრება WKT, WKB და GML ფორმატებში.

არსებობს წრის რკალების სამი ახალი ტიპი:

- **CircularString;**

24.6 ნახაზზე GEOMETRY ტიპის იერარქიიდან ჩანს, რომ CircularString ტიპი არის Curve-ს ქვეტიპი. ამიტომაც CircularString ტიპის ობიექტები მრუდის საბაზო ქვეტიპია. CircularString ტიპის ობიექტის განსაზღვრისათვის საჭიროა მინიმუმ სამი წერტილის მოცემა. პირველი მიუთითებს რკალის დასაწყისს, მეორე - მის ბოლოს, ხოლო მესამე უნდა იყოს რკალის რომელიმე ადგილზე. CircularString ტიპის ეგზემპლარები შეიძლება გაერთიანდეს, სადაც წინა მრუდის ბოლო წერტილი ხდება მომდევნო მრუდის საწყისი წერტილი. 24.10 ლისტინგში ნაჩვენებია CircularString ტიპის ობიექტის განსაზღვრა @g ცვლადის გამოყენებით.

```
DECLARE @g GEOMETRY;
SET @g =  GEOMETRY::STGeomFromText
       ('CIRCULARSTRING(0 -12.5, 0 0, 0 12.5)', 0);
```

ლისტინგი 24.10

- **CompoundCurve;**

განსაზღვრავს ახალ შედგენილ მრუდებს, რომლებიც შედგება ან მხოლოდ CircularString ტიპის ეგზემპლარებისაგან ან CircularString და LineString ეგზემპლარებისაგან. ყოველი წინა კომპონენტის ბოლო წერტილი უკავშირდება მომდევნო საწყის წერტილს. 24.11 ლისტინგში ნაჩვენებია შედგენილი მრუდის შექმნა CircularString ტიპის რამდენიმე სხვადასხვა ობიექტით.

```
DECLARE @g GEOGRAPHY;
SET @g = GEOGRAPHY::STGeomFromText('
COMPOUNDCURVE(CIRCULARSTRING(0 -23.43778, 0 0, 0 23.43778),
CIRCULARSTRING(0 23.43778, -45 23.43778, -90 23.43778),
CIRCULARSTRING(-90 23.43778, -90 0, -90 -23.43778),
CIRCULARSTRING(-90 -23.43778, -45 -23.43778, 0 -23.43778) )' ,4326);
```

ლისტინგი 24.11

აქ იქმნება GEOGRAPHY მონაცემთა ტიპის ეგზემპლარი, რომელიც მიენიჭება @g ცვლადს. ეს ცვლადი შედგება CompoundCurve ტიპის ეგზემპლარისაგან, რომელიც თავის მხრივ შედგება CircularString ტიპის ეგზემპლარებისაგან. საყურადღებოა STGeomFromText() მეთოდის ბოლო არგუმენტის მნიშვნელობა 4326. ესაა SRID (Spatial reference ID) იდენტიფიკატორი GEOGRAPHY მონაცემთა ტიპისათვის და შეესაბამება WGS_82 კოორდინატთა სივრცით სისტემას [11].

- **CurvePolygon.**

CurvePolygon ტიპის ობიექტი შედგება LineString და CircularString ტიპის ობიექტებისაგან, აგრეთვე CompoundCurves ტიპის ობიექტებისგან. 24.6 ნახაზიდან ჩანს, რომ CurvePolygon არის უშუალო ქვეტიპი Surfface ტიპისა და სუპერტიპი Polygon ტიპისა. ასეთი წრის შიგნით CurvePolygon ობიექტის მომდევნო კომპონენტის პირველი წერტილი ემთხვევა მომდევნო კომპონენტის ბოლო წერტილს.

- **ახალი სივრცითი ინდექსები**

SQL Server 2012 -ში GEOMETRY и GEOGRAPHY ტიპის მონაცემებისათვის დამატებულია ახალი სივრცითი ინდექსი auto_grid_index. მისი ფუნქციონალობა ორივესთვის მსგავსია, ამიტომ მხოლოდ ერთს განვიხილავთ.

ამ ინდექსში გამოიყენება ტესელაციის რვა დონე სხვადასხვა ზომის ობიექტის უკეთესი აპროქსიმაციის მიზნით.

24.12 ლისტინგში ნაჩვენებია ავტომატური ინდექსის შექმნა გეომეტრიული ობიექტისათვის.

```
CREATE SPATIAL INDEX auto_grid_index
    ON beverage_markets(shape)
    USING GEOMETRY_AUTO_GRID
        WITH (BOUNDING_BOX = (xmin=0, ymin=0, xmax=500, ymax=200),
        CELLS_PER_OBJECT = 32, DATA_COMPRESSION = page);
```

ლისტინგი 24.12

24.7. ახალი სისტემური შენახვადი პროცედურები სივრცითი მონაცემებისთვის

SQL Server 2012 -ში სივრცითი ტიპის მონაცემებისათვის დამატებულია ახალი სისტემური შენახვადი პროცედურები:

- sp_help_spatial_geometry_index;
- sp_help_spatial_geography_index;

რომელთა გამოყენების სინტაქსიც ორივესი მსგავსია.

sp_help_spatial_geometry_index სისტემური შენახვადი პროცედურა აბრუნებს უკან გეომეტრიული ობიექტის სივრცითი ინდექსის მითითებული თვისებების ერთობლიობის დასახელებებს და მნიშვნელობებს. შედეგები აისახება ცხრილურ ფორმატში. აგრეთვე შესაძლებელია ძირითად თვისებათა ერთობლიობის ან ინდექსის ყველა თვისების მოთხოვნა. 24.13 ლისტინგში მოცემულია ამ სისტემური შენახვადი პროცედურის კოდი.

```
DECLARE @query geometry
 ='POLYGON((-90.0 -180.0, -90.0 180.0, 90.0 180.0, 90.0 -180.0, -90.0 -180.0))' ;
EXEC sp_help_spatial_geometry_index 'beverage_markets',
                                              'auto_grid_index', 0, @query;
```

ლისტინგი 24.12

ამ მაგალითში sp_help_spatial_geometry_index სისტემური პროცედურა ასახავს auto_grid_index სივრცითი ინდექსის თვიებებს, რომელიც შეიქმნა 24.12 ლისტინგით.

VIII ნაწილი

ბიზნესაპლიკაციების რეალიზაცია დაპროგრამების ჰიბრიდული ტექნოლოგიებით

XXV თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „უნივერსიტეტი“ 774

XXVI თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „საფინანსო ბანკი“....... 793

XXVII თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „სატრანსპორტო გადაზიდვები“ 801

XXVIII თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროში „ელექტრონული არჩევნები“ 817

XXIX თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროსში „შავი ზღვის ეკოლოგია“ 872

XXX თავი. WPF-აპლიკაციის აგება საპრობლემო სფეროსათვის „მარკეტინგი“ 900

XXV თავი
WPF-აპლიკაციის აგება საპრობლემო სფეროში „უნივერსიტეტი“

საუნივერსიტეტო მართვის საინფორმაციო სისტემის შექმნა ან არსებული სისტემის მოდიფიკაცია ინტეგრაციის პრინციპების საფუძველზე მოითხოვს ამ საპრობლემო სფეროს ობიექტორიენტირებული, პროცესორიენტირებული და სერვისორიენტირებული მიდგომების კომპლექსურ გამოყენებას [24]. სისტემის შესაბამისი ინფრასტრუქტურის დასამუშავებლად კი აუცილებელია დღეისათვის არსებული ისეთი სტანდარტებისა და მეთოდოლოგიების გამოყენება, როგორიცაა BSI, ITIL, COBIT [25], რაც საბოლოოდ უზრუნველყოფს უსაფრთხო განაწილებული ინფორმაციული სისტემის შექმნას, მის შემდგომ მასშტაბირებასა და განვითარებას.

მეორე მხრივ, ინტეგრაციის პროცესში სისტემის ცალკეული კომპონენტების დასამუშავებლად დროითი პარამეტრებისა და პროგრამული პროდუქტის ხარისხის გასაუმჯობესებლად აუცილებელი ხდება თანამედროვე CASE ტექნოლოგიების გამოყენება, როგორიცაა მაგალითად, Enterprise Architect (UML-ის ინსტრუმენტული საშუალება), Natural Object Role Modeling Architect (NORMA) და სხვ., რომლებიც მაიკროსოფთის Visual Studio .NET Framework პაკეტის სამუშაო გარემოს თავსებადია [14,26,27].

წინამდებარე პარაგრაფში გადმოცემულია საუნივერსიტეტო მართვის საინფორმაციო სისტემის საპილოტო ვერსიის მაგალითზე მონაცემთა განაწილებული ბაზისა და მომხმარებელთა ინტერფეისების დაპროექტების და პროგრამული რეალიზაციის პროცედურების დეტალური აღწერა ზემოაღნიშნული ახალი ტექნოლოგიებისა და სერვისორიენტირებული არქიტექტურის საფუძველზე [16].

პროგრამული უზრუნველყოფის სასიცოცხლო ციკლის ეტაპების შესაბამისად (ანალიზი, დაპროექტება, დეველოპმენტი, ტესტირება, დანერგვა), საუნივერსიტეტო მართვის საინფორმაციო სისტემის შექმნა, UML-ტექნოლოგიით, ითვალისწინებს სისტემის ფუნქციონალური და არაფუნქციონალური მოთხოვნილებების განსაზღვრას, ობიექტორიენტირებული ანალიზისა და დაპროექტების განხორციელებას, შესაბამისად სისტემასთან მომხმარებელთა მუშაობის ინტერაქტიული სცენარების, კლასთა ასოსიაციებისა და მდგომარეობათა დიაგრამების აგებით სხვადასხვა შემთხვევით მოვლენათა შესაბამისად.

ესაა ცოდნა სამართავი ობიექტის შესახებ, მისი სტატიკური (მდგომარეობათა სიმრავლე) და დინამიკური (ქცევათა სიმრავლე) მოდელებით. თუ ობიექტორიენტირებული მოდელირების ტერმინებით ვისარგებლებთ, დასმული ამოცანის გადაწყვეტის „გასაღებს“ კლასების, ობიექტების, კლასთაშორისი კავშირების, ობიექტროლური და არსთა დამოკიდებულების მოდელებისა და სხვა სახის დიაგრამების

აგება წარმოადგენს, ხოლო შემდეგ, კლასთა-ასოციაციებისა და არსთა-დამოკიდებულების დიაგრამათა საფუძველზე განხორციელდება მიზნობრივი სისტემის პროგრამული კოდების რეალიზაციის ავტომატიზებული პროცესი ტესტირებით [28].

ამგვარად, ჩვენ განვიხილავთ კონკრეტული მართვის საინფორმაციო სისტემის („უნივერსიტეტი") აგების ამოცანების სპექტრს და ამ პროცესში გამოვყოფთ პროგრამული სისტემის ტესტირების ფუნქციურ ამოცანას და მის რეალიზაციას [16].

25.1. სისტემის ობიექტროლური მოდელის დაპროექტება

განვიხილოთ ზოგადად „უნივერსიტეტის" კლასის ობიექტისათვის მონაცემთა განაწილებული ბაზის დაპროექტების ამოცანა. უნივერსიტეტის საპრობლემო სფეროს არაფორმალიზებული აღწერის ობიექტებია (ტერმინთა ლექსიკონი): ფაკულტეტი, დეპარტამენტი, სტუდენტი, ლექტორი, საგანი (აკადემიური დისციპლინები, რომლებიც იკითხება შესაბამისი კათედრებსა და სპეციალობებზ), სასწავლო გეგმები, სილაბუსები (პროგრამები), ლექციები, პრაქტიკული და ლაბორატორიული სამუშაოები, აუდიტორიები, გამოცდები, ტესტირება და ა.შ.

ობიექტროლური მოდელირების თეორიის წესების თანახმად საჭიროა აიგოს უნივერსიტეტის სასწავლო პროცესის შესაბამისი ORM-დიაგრამა. ამისათვის ფაქტ-შეზღუდვების ერთობლიობით (რომელსაც ადგენს სისტემური ანალიტიკოსი და საბოლოო მომხმარებელი), რომლებშიც ასახულია საპრობლემო სფეროს შესახებ ცოდნა (კლასებისა და ობიექტების ძირითადი ტერმინები და ქცევის წესები, დებულებით დადგენილი კანონზომიერებები და სხვ.), გადაიტანება Ms_Visual Studio.NET Framework სამუშაო გარემოში, ობიექტ-როლური მოდელის, NORMA-პაკეტის (Natural ORM Architect) ინტერფეისზე [14,15]. მაგალითად, ასეთი ფაქტები შეიძლება იყოს:

f1 : ლექტორს აქვს გვარი

f2 : ლექტორი მუშაობს დეპარტამენტში

f3 : ლექტორს აქვს თანამდებობა

f4 : ლექტორს აქვს ტელეფონი

f5 : ლექტორს აქვს ელ-ფოსტა

f6 : ლექტორს აქვს ხარისხი

f7 : ლექტორი კითხულობს საგანს

f8 : ლექტორი ასწავლის #-ჯგუფს

f8 : სტუდენტი არის #-ჯგუფში

. . .

f50 : ლექტორი მუშაობს #-დეპარტამენტში

f51 : დეპარტამენტი ეკუთვნის #-ფაკულტეტს

f52 : #-ჯგუფი ეკუთვნის #-დეპარტამენტს

f53 : ფაკულტეტს აქვს დასახელება

. . .

f100 : სტუდენტი არ შეიძლება იყოს ერთზე მეტ ჯგუფში

f101 : ლექტორი არ შეიძლება იყოს სრულ შტატზე ერთზე მეტ დეპარტამენტში

f102 : ლექტორი დროის ერთ მომენტში არ შეიძლება იყოს ორ სხვადასხვა აუდიტორიაში

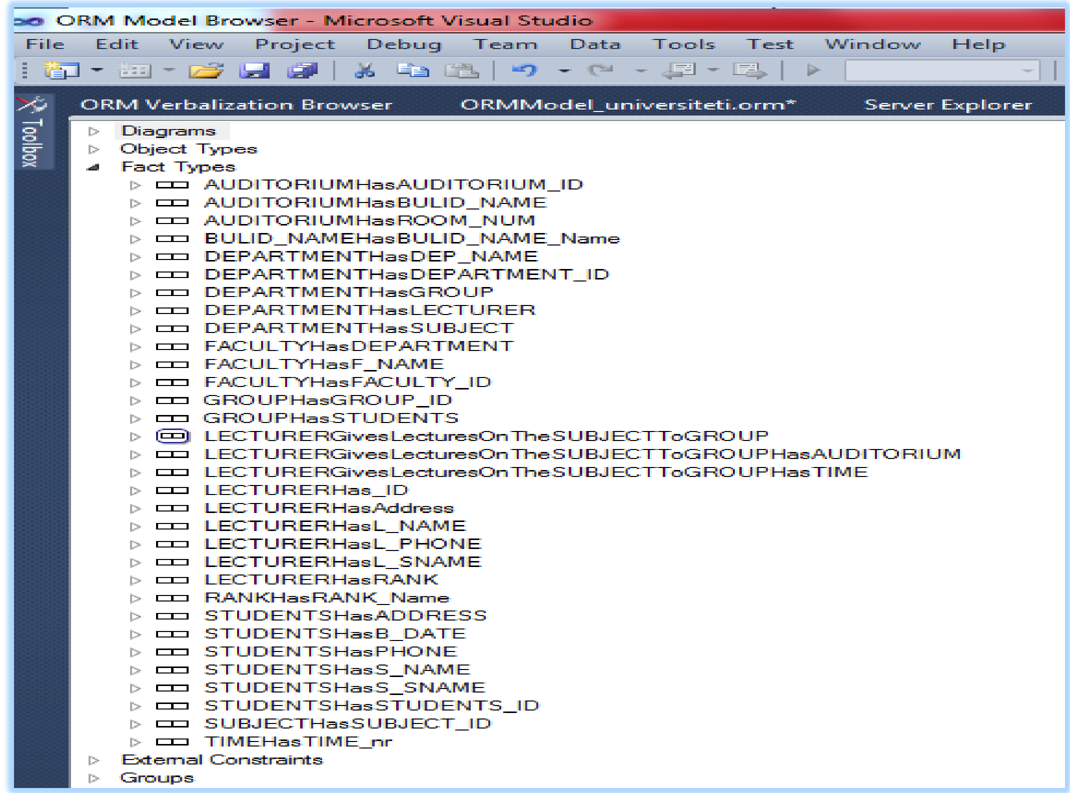
და ა.შ.

25.1 ნახაზზე წარმოდგენილია ობიექტროლური მოდელირების თეორიაში არსებული ზოგიერთი შეზღუდვის მაგალითი, რომლებიც გამოყენებულია ჩვენ მაგალითში.

Internal Uniqueness Constraint	შიგა უნიკალურობა: ერთ ან მეტ როლში მონაწილეობა ხდება არა უმეტეს ერთხელ;
External Uniqueness Constraint	გარე უნიკალურობა: ობიექტის უნიკალურობა განისაზღვრება ორი ობიექტით;
Objectified Fact Type	ბუდის ტიპის ობიექტი: ობიექტი თამაშობს მხოლოდ ერთ როლს და ეს როლი არ არის სავალდებულო;
Frequency Constraint	სიხშირის შეზღუდვა: ობიექტმა შეიძლება მიიღოს ჩამოთვლილი მნიშვნელობებიდან ერთ-ერთი.
. . .	

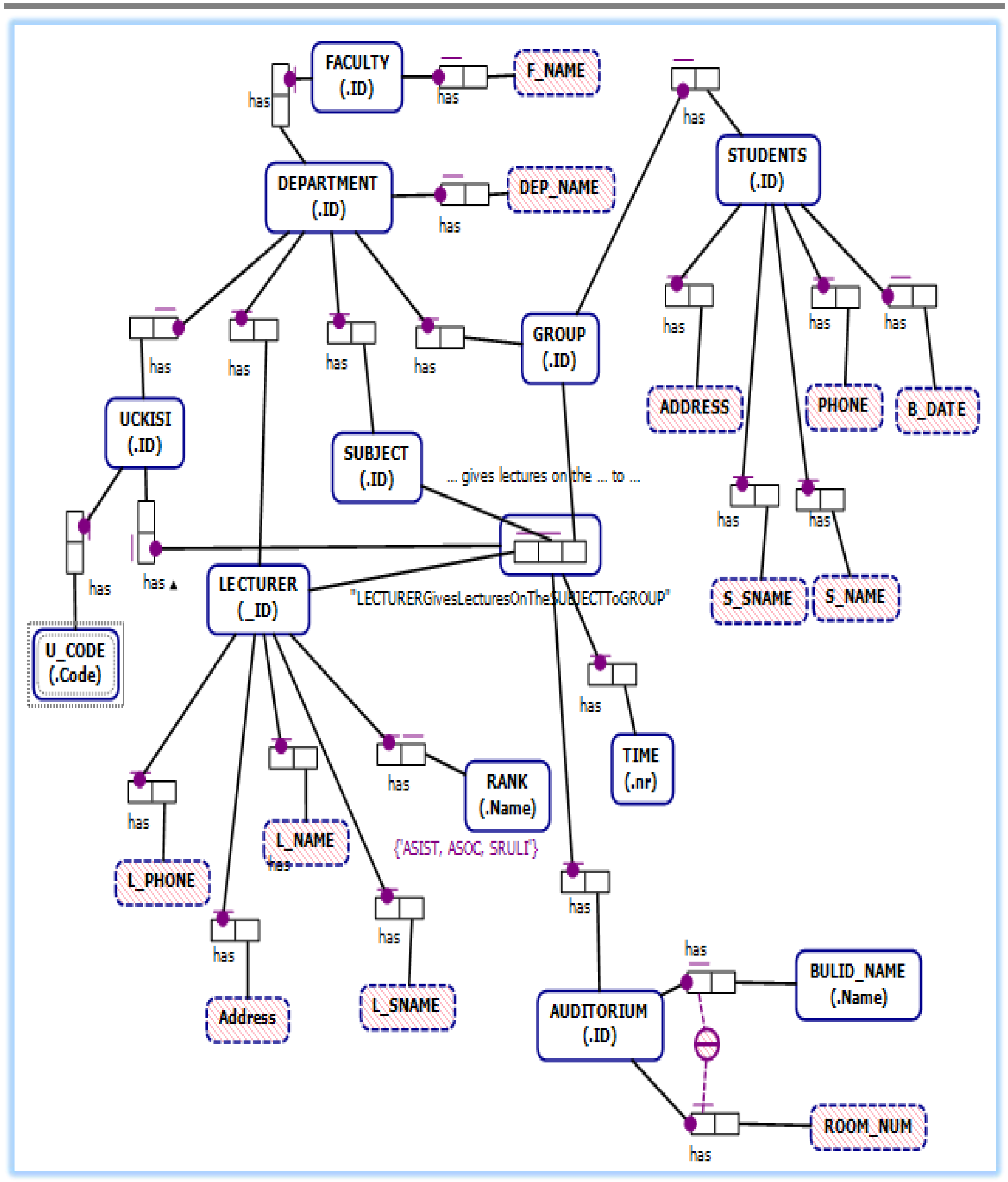
ნახ.25.1. შეზღუდვების აღწერის მაგალითები ORM-დიაგრამაზე

შეზღუდვების აღწერა ხდება კატეგორიალური მიდგომის საფუძველზე, რომელიც აერთიანებს სალაპარაკო ენის ფორმალური გრამატიკის და ლოგიკურ-ალგებრულ წესებს. შედეგად მიიღება პრედიკატები, რომლებიც შეიძლება იყოს ერთ-, ორ- ან n-ადგილიანი [67]. 25.2 ნახაზზე მოცემულია Visual Studio.NET სამუშაო გარემოში ORM ინსტრუმენტის გამოყენებით მიღებული შედეგები „უნივერსიტეტის" ფაქტების შეტანის საფუძველზე.



ნახ.25.2. „უნივერსიტეტის“ ფაქტების აღწერის ფრაგმენტი

ზემოჩამოთვლილი ფაქტებიდან NORMA ინსტრუმენტი გვაძლევს შემდეგი სახის ORM-დიაგრამას (ნახ.25.3). აქ შესაძლებელია ახალი ფაქტის დამატება, არსებულის მოდიფიკაცია / წაშლა.

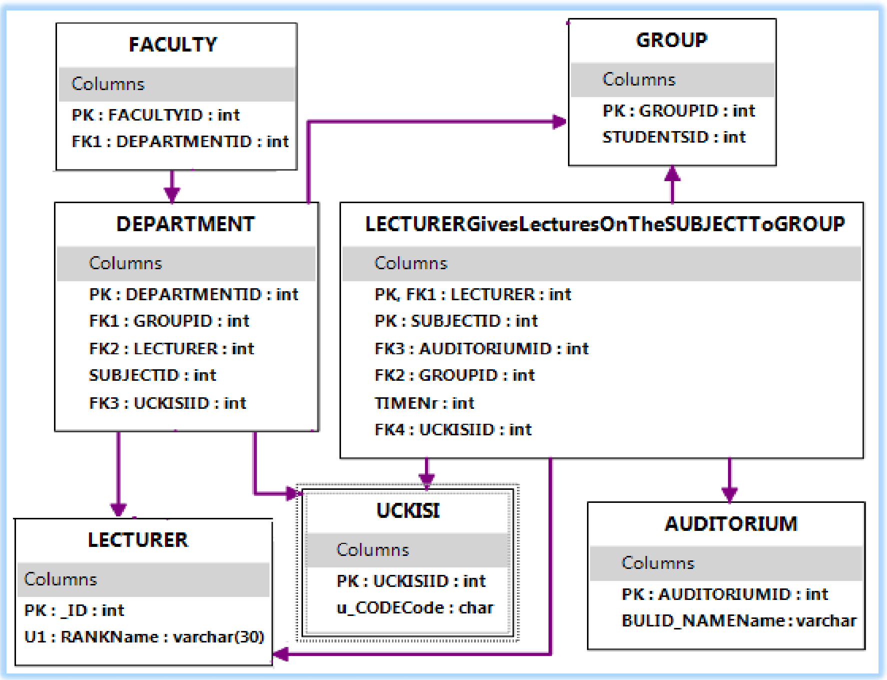


ნახ.25.3. „უნივერსიტეტის“ ORM დიაგრამის ფრაგმენტი
(საპრობლემო სფეროს კონცეპტუალური მოდელი 1)

25.2. არსთადამოკიდებულების ER მოდელის დაპროექტება

მომდევნო ეტაპზე განხორციელდება ობიექტროლური მოდელის ავტომატიზებული გადაყვანა არსთადამოკიდებულების მოდელში. სისტემის დამპროექტებელი გააქტიურებს NORMA პაკეტის მენიუდან გენერაციის პროცედურას. ORM-დიაგრამიდან გენერირებული ERM-დიაგრამა მოცემულია 25.4 ნახაზზე.

შესაბამისად გამოკვეთილია შვიდი ობიექტი (არსი - Entity): ფაკულტეტი (Faculty), დეპარტამენტი (Department), ჯგუფი (Group), სტუდენტი (Students), ლექტორი (Lecturer), საგანი (Subject), აუდიტორია (Auditorium), საგამოცდო_უწყისი (Uckisi).

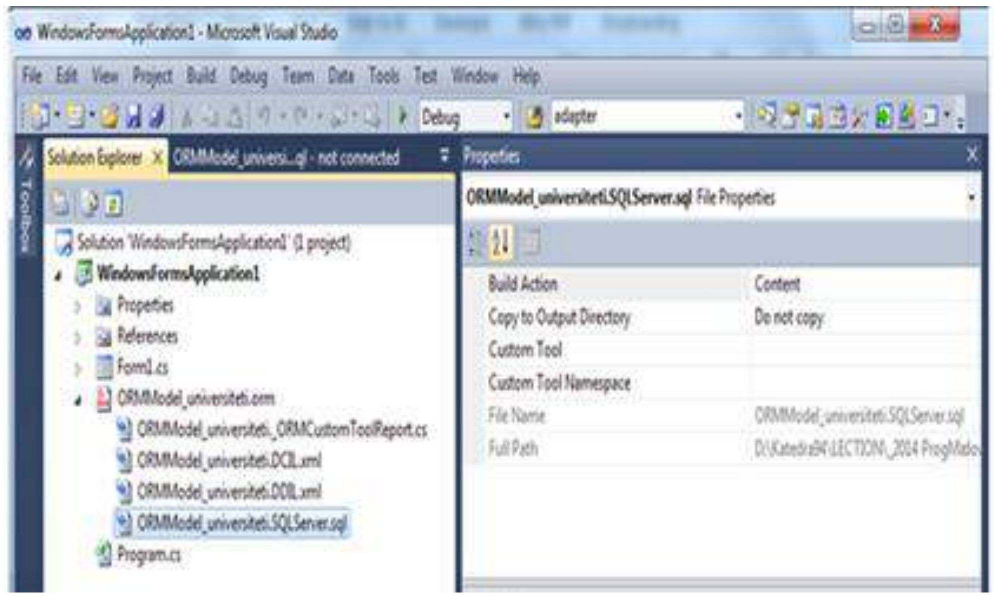


ნახ.25.4. ORM დიაგრამიდან გენერირებული ER მოდელი
(საპრობლემო სფეროს კონცეპტუალური მოდელი 2)

დიალოგში შესაძლებელია ER-მოდელის ობიექტების (Tables) განლაგების შეცვლა, რათა ვიზუალურად უფრო მოხერხებული მდებარეობა მიიღოს თითოეულმა, ობიექტთაშორისი კავშირების რაც შეიძლება ნაკლები გადაკვეთებით. ცხრილებში ჩანს ობიექტის სახელი და ატრიბუტთა დასახელებები, ტიპების მითითებით. აგრეთვე ასახულია პირველადი (PK) და მეორეული (FK) გასაღებური ატრიბუტები და ა.შ.

25.3. მონაცემთა ბაზის სერვერზე განთავსება

მონაცემთა ბაზის კონცეპტუალური ERM სქემის აგების შემდეგ საჭიროა მის საფუძველზე სისტემის მიერ დაიწეროს შუალედური ტექსტური ტიპის DLL–ფაილი, რომელიც მომავალში SQL Server მონაცემთა ბაზების მართვის სისტემამ უნდა გამოიყენოს. ამგვარად, ER-დიაგრამიდან, ცხრილებითა და ატრიბუტებით, ავტომატურად გენერირდება .DDL ფაილები, რომლებიც შემდგომ სტრუქტურულად მოთავსდება SQL Server მონაცემთა ბაზაში. 25.5 ნახაზზე ნაჩვენებია ამ ეტაპის პროცესის ინიცირების დიალოგური სქემა.



ნახ.25.5. DDL ფაილის გენერაცია ER-მოდელიდან

25.1 ლისტინგში მოცემულია ავტომატურად გენერირებული DDL-ფაილის ტექსტის ფრაგმენტი.

```
!-- Listing_25.1 -----DDL file --------------------------
CREATE SCHEMA ORMModel1
GO
GO
CREATE TABLE ORMModel1.FACULTY
(  FACULTYID INTEGER IDENTITY (1, 1) NOT NULL,
   DEPARTMENTID INTEGER NOT NULL,
   CONSTRAINT FACULTY_PK PRIMARY KEY(FACULTYID)  )
```

```
GO
CREATE TABLE ORMModel1.DEPARTMENT
(  DEPARTMENTID INTEGER IDENTITY (1, 1) NOT NULL,
   GROUPID INTEGER NOT NULL,
   LECTURER INTEGER NOT NULL,
   SUBJECTID INTEGER IDENTITY (1, 1) NOT NULL,
   CONSTRAINT DEPARTMENT_PK PRIMARY KEY(DEPARTMENTID)   )
GO
CREATE TABLE ORMModel1."GROUP"
(  GROUPID INTEGER IDENTITY (1, 1) NOT NULL,
   STUDENTSID INTEGER IDENTITY (1, 1) NOT NULL,
   CONSTRAINT GROUP_PK PRIMARY KEY(GROUPID)       )
GO
CREATE TABLE ORMModel1.LECTURER
(  "_ID" INTEGER IDENTITY (1, 1) NOT NULL,
   RANKName NATIONAL CHARACTER VARYING(30) NOT NULL,
   CONSTRAINT LECTURER_PK PRIMARY KEY("_ID"),
   CONSTRAINT LECTURER_UC UNIQUE(RANKName),
   CONSTRAINT LECTURER_RANKName_RoleValueConstraint1 CHECK (RANKName IN (N'ASIST,
ASOC, SRULI'))  )
GO
CREATE TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP
(  LECTURER INTEGER NOT NULL,
   SUBJECTID INTEGER IDENTITY (1, 1) NOT NULL,
   AUDITORIUMID INTEGER NOT NULL,
   GROUPID INTEGER NOT NULL,
   TIMENr INTEGER NOT NULL,
   CONSTRAINT LECTURERGivesLecturesOnTheSUBJECTToGROUP_PK PRIMARY KEY(LECTURER,
SUBJECTID)     )
GO
CREATE TABLE ORMModel1.AUDITORIUM
(  AUDITORIUMID INTEGER IDENTITY (1, 1) NOT NULL,
   BULID_NAMEName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
   CONSTRAINT AUDITORIUM_PK PRIMARY KEY(AUDITORIUMID)         )
GO
```

```
ALTER TABLE ORMModel1.FACULTY ADD CONSTRAINT FACULTY_FK FOREIGN KEY
(DEPARTMENTID) REFERENCES ORMModel1.DEPARTMENT (DEPARTMENTID) ON DELETE NO ACTION
ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.DEPARTMENT ADD CONSTRAINT DEPARTMENT_FK1 FOREIGN KEY
(GROUPID) REFERENCES ORMModel1."GROUP" (GROUPID) ON DELETE NO ACTION ON UPDATE NO
ACTION
GO
ALTER TABLE ORMModel1.DEPARTMENT ADD CONSTRAINT DEPARTMENT_FK2 FOREIGN KEY
(LECTURER) REFERENCES ORMModel1.LECTURER ("_ID") ON DELETE NO ACTION ON UPDATE NO
ACTION
GO
ALTER TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP ADD CONSTRAINT
LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK1 FOREIGN KEY (LECTURER) REFERENCES
ORMModel1.LECTURER ("_ID") ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP ADD CONSTRAINT
LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK2 FOREIGN KEY (GROUPID) REFERENCES
ORMModel1."GROUP" (GROUPID) ON DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.LECTURERGivesLecturesOnTheSUBJECTToGROUP ADD CONSTRAINT
LECTURERGivesLecturesOnTheSUBJECTToGROUP_FK3 FOREIGN KEY (AUDITORIUMID)
REFERENCES ORMModel1.AUDITORIUM (AUDITORIUMID) ON DELETE NO ACTION ON UPDATE NO
ACTION
GO
GO
```

შეიძლება ჩვთვალოთ, რომ ამ DDL ფაილის კოპირებით Ms SQL Server–ში შეიქმნება შესაბამისი ბაზა, ცხრილებით, ატრიბუტებით და კავშირებით.

რა თქმა უნდა, შესაძლებელია აქაც დამპროექტებლის ჩარევა ბაზის სტრუქტურის ზოგიერთი კომპონენტის შესასწორებლად, საჭიროებისამებრ.

25.4. ბიზნესპროცესის სერვისის შექმნა

როგორც ცნობილია, ბიზნესპროცესი შეიძლება განთავსდეს ვებსერვისში, რომელიც უზრუნველყოფს ბიზნესპროცესის გადაწყვეტილების (შედეგის) მიწოდებას კლიენტებისათვის (ვებაპლიკაციებისათვის). ვებსერვისი იღებს მოთხოვნას კლიენტისგან, ასრულებს მის დამუშავებას და უბრუნებს პასუხს. ეს პროცედურები სრულდება Receive და Send ქმედებებით.

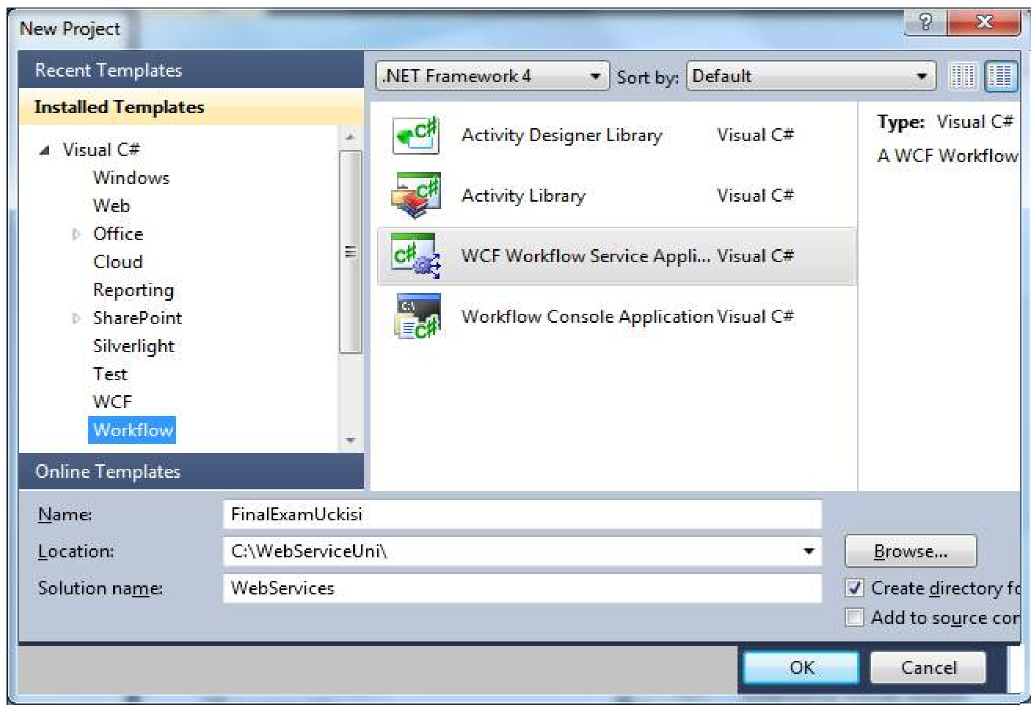
განვიხილოთ ჩვენი მაგალითის რეალიზაცია ჰიბრიდული ტექნოლოგიების, WF (Workflow Foundation) და WCF ბაზაზე (Windows Communication Foundation) [69]. ისევე, როგორც WPF (Windows Presentation Foundation) ტექნოლოგია, Visual Studio .NET Framework 4.0/4.5 გარემოში ქმნის მომხმარებელზე ორიენტირებულ მაღალი ხარისხის დიზაინის აპლიკაციებს C#.NET (ლოგიკის ნაწილი) და XAML (დიზაინის ნაწილი) ენების საფუძველზე [16-18].

ავამუშავოთ Visual Studio და შევქმნათ ახალი პროექტი WCF Workflow Service Application template გამოყენებთ. შევიტანოთ პროექტის სახელი FinalExamUckisi და solution-ის სახელი WebServices (ნახ.25.6).

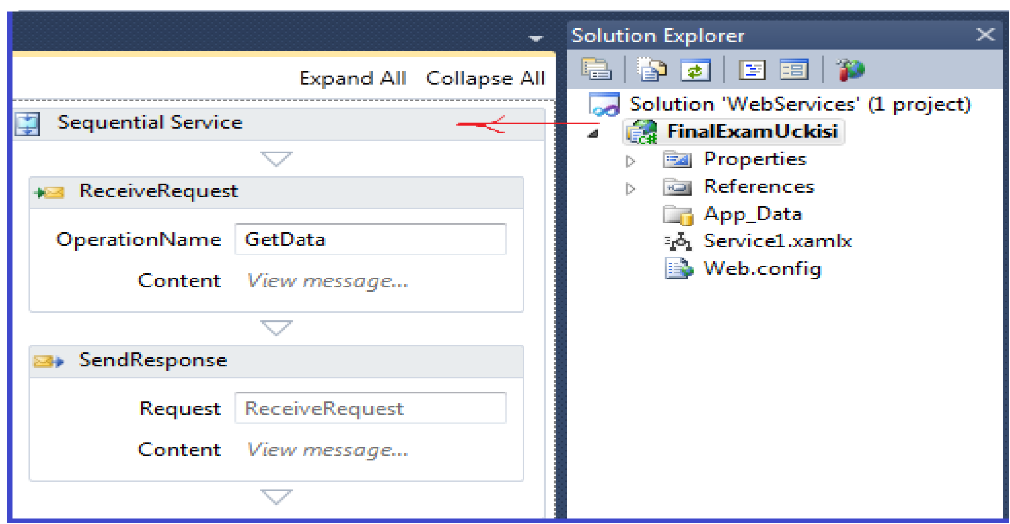
შეიქმნება საინიციალიზაციო workflow Sequence ბლოკი, რომელიც შეიცავს Receive და SendReply ქმედებებს, როგორც 25.7 ნახაზზეა ნაჩვენები.

თავიდან საჭიროა ამ ქმედებების კონფიგურაცია სერვისის კონტრაქტის განსაზღვრის მიზნით, რომელსაც ისინი დააკმაყოფილებს. შემდეგ დავამატოთ დამუშავების ბიზნესპროცესი, რომელიც განხორციელდება Receive და SendReply ქმედებებს შორის.

შაბლონი შექმნის საწყის ბიზნესპროცესს ფაილში, სახელით Service1.xamlx. შევცვალოთ Solution Explorer-ში ეს სახელი FinalExamUckisi.xamlx -ით.



ნახ.25.6. Visual Studio.NET –ში WCF-პროექტის შექმნა



ნახ.25.7. FinalExamUckisi პროექტის workflow Sequence ბლოკი

სერვისი, რომელიც მაგალითის სახით უნდა შევქმნათ, „საფინალო გამოცდისათვის", ძებნის შესაბამის უწყისებს მითითებული აკადემიური ჯგუფის, ლექტორისა და აკადემიური საგნის მიხედვით. ელექტრონული საგამოცდო უწყისები ეკუთვნის ფაკულტეტის დეკანატს, ხოლო მათი დამუშავება ხდება დეპარტამენტთა ლექტორების მიერ".

25.5. სერვისის კონტრაქტის განსაზღვრა

WCF სისტემებში იყენებენ კონტრაქტების სამ დონეს: მონაცემთა კონტრაქტი, შეტყობინებათა კონტრაქტი და სერვისის კონტრაქტი [3,16].

მონაცემთა კონტრაქტის დანიშნულებაა შეთანხმება კლიენტსა და სერვისს შორის ერთმანეთთან გასაცვლელ მონაცემებზე (ითვალისწინებენ მონაცემთა სტრუქტურებს, პარამეტრებს, მოწესრიგებას და ა.შ.).

შეტყობინებათა კონტრაქტი უზრუნველყოფს SOAP (Simple Object Access Protocol) შეტყობინებების კონტროლს, რომელიც გამოიყენება ქსელში სხვადასხვა შეტყობინების გასაცვლელად XML ფორმატში. იგი აგრეთვე უზრუნველყოფს ინფორმაციის გაცვლის უსაფრთხოებას შეტყობინებების დონეზე.

სერვისის კონტრაქტი (ანუ კონტრაქტი მომსახურებისათვის) განსაზღვრავს ოპერაციათა სახეებს, რომლებსაც უზრუნველყოფს სერვისი. იგი კლიენტს აწვდის ასევე ინფორმაციას: შეტყობინებაში მონაცემთა ტიპების შესახებ, ოპერაციათა ადგილმდებარეობის შესახებ, ინფორმირების პროტოკოლისა და სერიალიზაციის ფორმატის შესახებ, შეტყობინებათა გაცვლის შაბლონების შესახებ (ცალმხრივი, ორმხრივი ან კითხვა/პასუხის ტიპებით).

ჩვენი პროექტისთვის სერვისის კონტრაქტის ასაგებად უნდა შევქმნათ საგამოცდო უწყისის ინფორმაციის კლასი C# ნაზე - UckisiInfo.cs. ამისთვის Solution Explorer-ში მარჯვენა ღილაკით FinalExamUckisi პროექტზე ვირჩევთ Add->Cla სახელით UckisiInfo.cs, რომლის ტექსტი მოცემულია 25.2 ლისტინგში.

```
// ----- ლისტინგი_25.2 – Service Contract ---
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;

namespace FinalExamUckisi
{  // ---- სერვისის კონტრაქტის განსაზღვრა --- IfinalExamUckisi,  რომელიც
   // --- შედგება ერთი მეთოდისგან  -  LookupUckisi () ------
    [ServiceContract]
  public interface IFinalExamUckisi
    {   [OperationContract]
        UckisiInfoList LookupUckisi(UckisiSearch request);
    }
//-- მოთხოვნის შეტყობინების განსაზღვრა , UckisiSearch ---
    [MessageContract(IsWrapped = false)]
   public class UckisiSearch
     {  private String _JGUPI;
        private String _Sagani;
        private String _Lectori;
        public UckisiSearch() {  }
        public UckisiSearch(String sagani, String lectori,
                           String jgupi)
        {
            _Sagani  = sagani;
            _Lectori = lectori;
            _JGUPI   = jgupi;       }
        #region Public Properties
        [MessageBodyMember]
        public String Sagani
        {   get { return _Sagani; }
            set { _Sagani = value; }    }
        [MessageBodyMember]
```

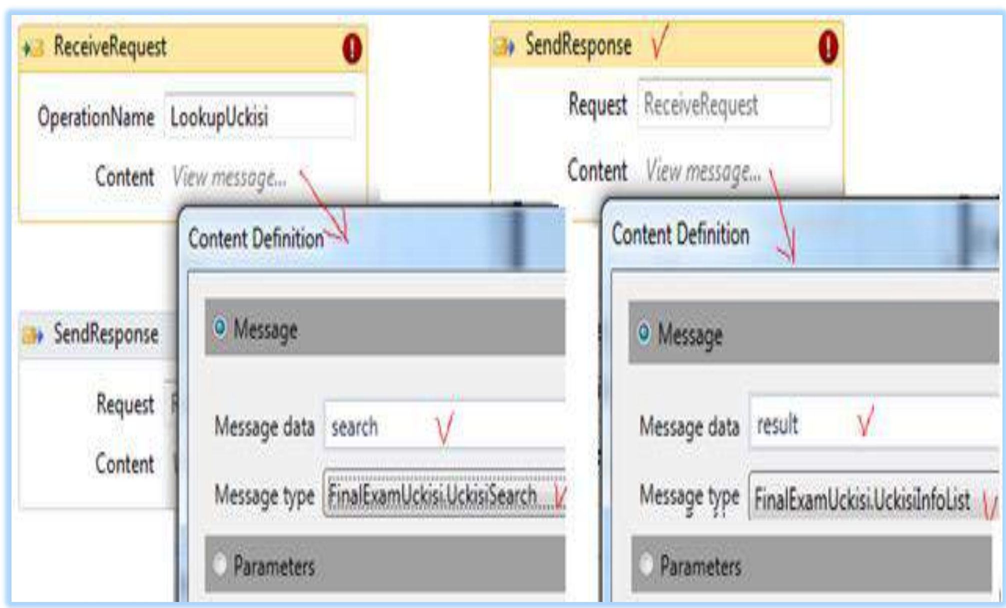
```
    public String Lectori
    {   get { return _Lectori; }
        set { _Lectori = value; }   }
    [MessageBodyMember]
    public String JGUPI
    {   get { return _JGUPI; }
        set { _JGUPI = value; }     }
    #endregion Public Properties
}

// --- UckisiInfo კლასის განსაზღვრა -----
    [MessageContract(IsWrapped = false)]
public class UckisiInfo
{   private Guid _ExamUckisiID;
    private String _JGUPI;
    private String _Sagani;
    private String _Lectori;
    private String _Status;
    public UckisiInfo() {   }
    public UckisiInfo(String sagani, String lectori,
                      String jgupi, String status)
    {   _Sagani = sagani;
        _Lectori = lectori;
        _JGUPI = jgupi;
        _Status = status;
        _ExamUckisiID = Guid.NewGuid();    }
    #region Public Properties
    [MessageBodyMember]
    public Guid ExamUckisiID
    {   get { return _ExamUckisiID; }
        set { _ExamUckisiID = value; }     }
    [MessageBodyMember]
    public String Sagani
    {   get { return _Sagani; }
        set { _Sagani = value; }      }
    [MessageBodyMember]
   public String Lectori
```

```
        {   get { return _Lectori; }
            set { _Lectori = value; }     }
        [MessageBodyMember]
       public String JGUPI
        {   get { return _JGUPI; }
            set { _JGUPI = value; }       }
        [MessageBodyMember]
       public String status
        {   get { return _Status; }
            set { _Status = value; }      }
        #endregion Public Properties
    }
    //--საპასუხო შეტყობინების განსაზღვრა --- UckisiInfoList---
       [MessageContract(IsWrapped = false)]
    public class UckisiInfoList
    {   private List<UckisiInfo> _UckisiList;
        public UckisiInfoList()
        {  _UckisiList = new List<UckisiInfo>(); }
        [MessageBodyMember]
        public List<UckisiInfo> UckisiList
        {    get { return _UckisiList; }    }
    }
}
```

სერვისის კონტრაქტი IFinalExamUckisi შეიცავს ერთადერთ მეთოდს LookupUckisi(). იგი მონაცემებს გადასცემს UckisiSearch კლასს, რომელსაც აქვს სხვადასხვა თვისება, საჭირო საგამოცდო უწყისის მოსაძებნად, მაგალითად, ლექტორი, საგანი, ჯგუფი. ის აბრუნებს უკან UckisiInfoList კლასს, რომელიც შეიცავს UckisiInfo კლასების კოლექციას. F6 ამოქმედებით აიგება გადაწყვეტა (solution). ამგვარად, ჩვენ განვსაზღვრეთ შეტყობინებები, რომლებიც გადაიცემა სერვისმეთოდების მიერ პარამეტრების სახით.

25.6. Receive და SendReply კონფიგურირება

შემდეგ ეტაპზე FinalExamUckisi.xamlx-ში ReceiveRequest ქმედებისათვის OperationName თვისებაში ვათავსებთ LookupUckisi სახელს. WorkflowService-დიზაინერში შევქმნით ორ ახალ ცვლადს (Variables): search ცვლადი, შემომავალი შეტყობინების შესანახად და result ცვლადი, გამოსატანი შედეგისათვის (ნახ.25.8). Message ბუტონი უნდა იყოს ჩართული და ტიპებიც არჩეული.



ნახ. 25.8. შემავალი მოთხოვნის შეტყობინებისა და გამომავალი
შედეგის ცვლადების განსაზღვრა

25.7. PerformLookup ქმედების აგება (ძებნის შესრულება)

ძებნის განსახორციელებლად (მონაცემთა ბაზებთან მიმართვის მიზნითაც) ვქმნით ახალ კლასს PerformLookup.cs, რომელიც Solution Explorer-ში FinalExamUckisi-პროექტისთვის აირჩევა Add->NewItem, Workflow-კატეგორიაში, Code Activity-ით. ამ ახალი კლასის ტექსტი მოცემულია 25.3 ლისტინგში.

```
// ---- ლისტინგი_25.3 --- PerformLookup ----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
namespace FinalExamUckisi
{
 public sealed class PerformLookup : CodeActivity
  {
   public InArgument<UckisiSearch> Search { get; set; }
```

```
public OutArgument<UckisiInfoList> UckisiList {get;set;}
protected override void Execute(CodeActivityContext
                                                    context)
 { string lectori = Search.Get(context).Lectori;
   string sagani = Search.Get(context).Sagani;
   string jgupi = Search.Get(context).JGUPI;

   UckisiInfoList l = new UckisiInfoList();

  l.UckisiList.Add(new UckisiInfo(sagani, lectori,
        jgupi, "Available"));
  l.UckisiList.Add(new UckisiInfo(sagani, lectori,
        jgupi, "CheckedOut"));
  l.UckisiList.Add(new UckisiInfo(sagani, lectori, jgupi,
        "Missing"));
  l.UckisiList.Add(new UckisiInfo(sagani, lectori, jgupi,
        "Available"));
    UckisiList.Set(context, l);        }
 }
}
```

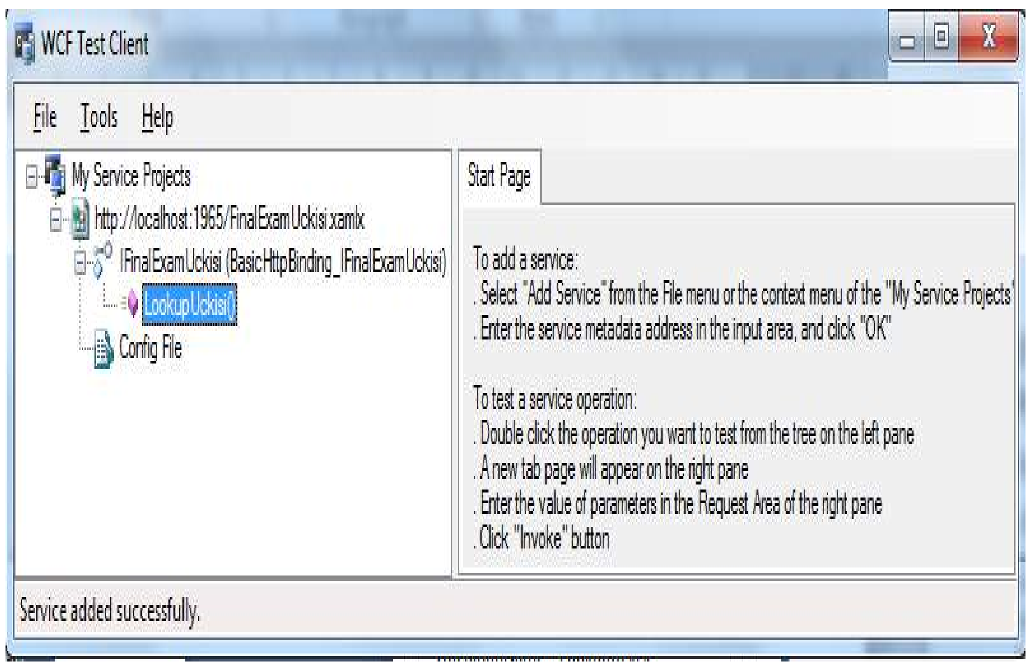
PerformLookup ქმედება ჩაემატება ინსტრუმენტების პანელზე. იგი უნდა გადმოვიტანოთ „ReceiveRequest" და „SendResponse" ქმედებებს შორის შესასრულებლად. ამასთანავე მისი Search თვისებისათვის ჩავწერეთ search, ხოლო UckisiList თვისებისათვის კი - result.

25.8. სერვისის ტესტირება

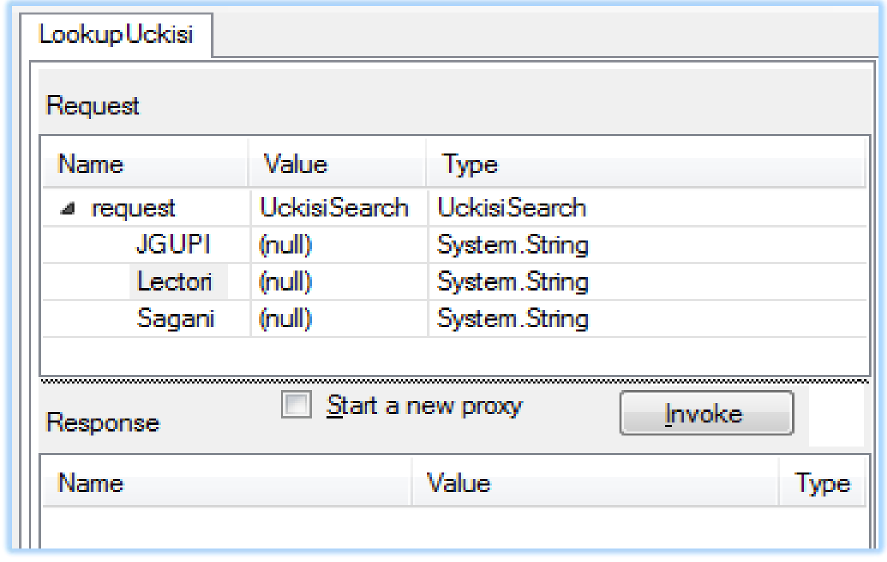
აპლიკაციის საბოლოო გამართვამდე უნდა მოვახდინოთ აგებული სერვისების ტესტირება. F5-ით ავამუშავოთ სერვისის გამართვის პროცედურა (debug). ვინაიდან ეს Web-სერვისია, Visual Studio ავტომატურად აამუშავებს WCF Test Client-ს [20,28].

ეს მეტად მოსახერხებელი უტილიტაა. იგი ჩატვირთავს Web-სერვისებს და აღმოაჩენს მეთოდებს, რომლებიც გათვალისწინებულია. ეს შედეგი ჩანს 25.9 ნახაზის მარცხენა პანელზე.

LookupUckisi() მეთოდზე 2-ჯერ დაჭერით მარჯვენა პანელის ზედა ნაწილში გამოიყოფა ადგილი შემოსული შეტყობინების განსათავსებლად (ნახ.25.10).

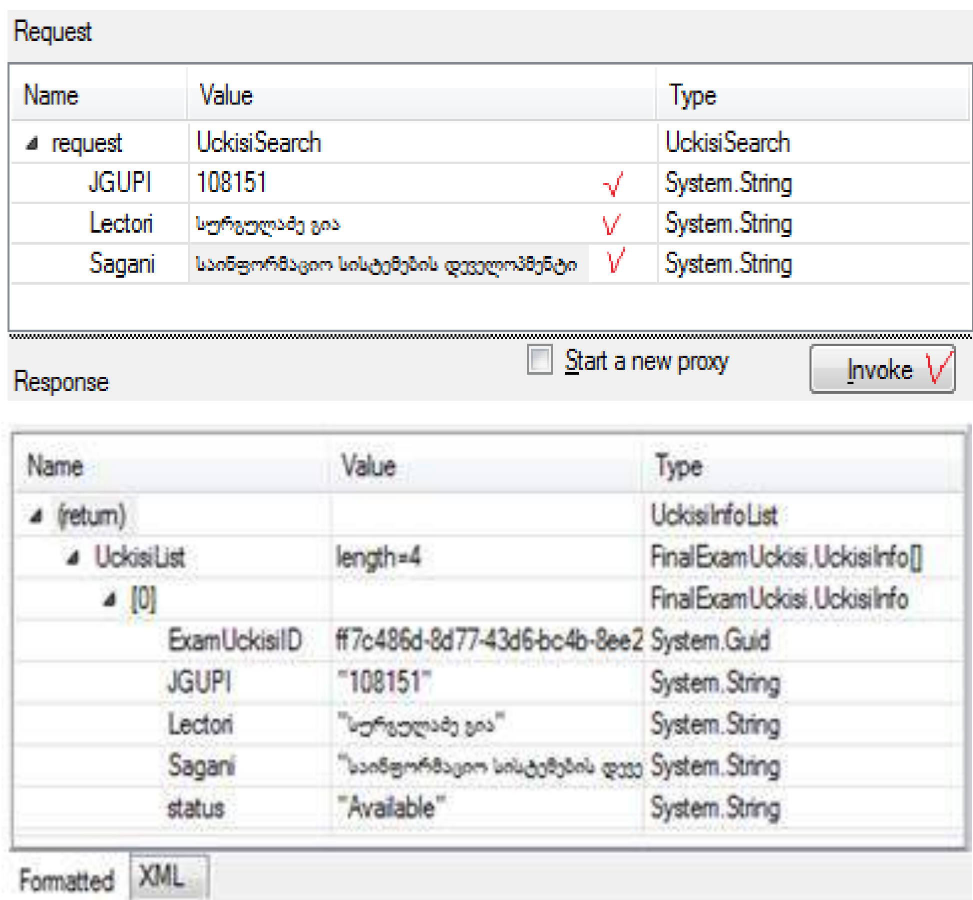


ნახ.25.9. ტესტირების პროცედურა



ნახ.25.10. საწყისი მონაცემების შესატანი
ფანჯარა

შევიტანოთ კონკრეტული მნიშვნელობები Lectori, JGUPI, Sagani და ავამოქმედოთ Invoke ღილაკი. ცხრილებში გამოჩნდება შედეგები (ნახ.25.11).



ნახ.25.11. WCF Client Test უტილიტით სერვისის
ტესტირების შედეგების ნახვა

25.12 ნახაზზე ნაჩვენებია Request და Response ცხრილების შესაბამისი ფაილები XML ფორმატში.

LookupUckisi

Request


```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1" xmlns="http://schemas.microsoft.com/ws/2005/05
      /addressing/none">http://tempuri.org/IFinalExamUckisi/LookupUckisi</Action>
  </s:Header>
  <s:Body>
    <JGUPI xmlns="http://tempuri.org/">108151</JGUPI>
    <Lectori xmlns="http://tempuri.org/">სურგულაძე გია</Lectori>
    <Sagani xmlns="http://tempuri.org/">საინფორმაციო სისტემების დეველოპმენტი</Sagani>
  </s:Body>
</s:Envelope>
```


Response


```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header />
  <s:Body>
    <UckisiList xmlns="http://tempuri.org/" xmlns:a="http://schemas.datacontract.org/2004
      /07/FinalExamUckisi" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
     <a:UckisiInfo>
      <a:ExamUckisiID>ff7c486d-8d77-43d6-bc4b-8ee25d371757</a:ExamUckisiID>
      <a:JGUPI>108151</a:JGUPI>
      <a:Lectori>სურგულაძე გია</a:Lectori>
      <a:Sagani>საინფორმაციო სისტემების დეველოპმენტი</a:Sagani>
      <a:status>Available</a:status>
     </a:UckisiInfo>
```


Formatted | XML

ნახ.25.12. ტესტირების შედეგების XML ტექსტები

XXVI თავი
WPF აპლიკაციის აგება საპრობლემო სფეროში „საფინანსო ბანკი“

26.1 ინფორმაციის გაცვლის პროგრამული რეალიზაციის ამოცანა SOA-ისთვის

ბიზნესპროცესების შესრულებისას ერთ-ერთი მნიშვნელოვანი ასპექტია ურთიერთობა (კომუნიკაცია) აპლიკაციებს შორის, კლიენტებსა და სერვერებს შორის, აგრეთვე სამუშაო პროცესებსა და ჰოსტდანართებს შორის [4].

წინამდებარე პარაგრფში აღწერილი გვაქვს, თუ როგორ შეიძლება ბიზნეს-პროცესების გამოყენებით გამარტივდეს და კოორდინაცია გაეწიოს კომუნიკაციათა სხვადასხვა სცენარს. აპლიკაციის მაგალითის სახით განიხილება პროექტის აგება საწარმოო ფირმებსა და ბანკებს შორის, რომელშიც მოთხოვნილი ინფორმაცია (მაგალითად, იურიდიული პირის მიერ საკრედიტო განაცხადის წარდგენა) გადაეცემა ბანკს. იმავე აპლიკაციას შეუძლია მოთხოვნის გაგზავნა (სხვა ბანკში) და ასევე პასუხის გაცემა სხვა ორგანიზაციების მოთხოვნაზე.

მთავარი ქმედებები, რომლებიც კომუნიკაციისთვის გამოიყენება, არის Send და Receive ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply). ეს ქმედებები გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და სამეთვალყურეოდ [18,35].

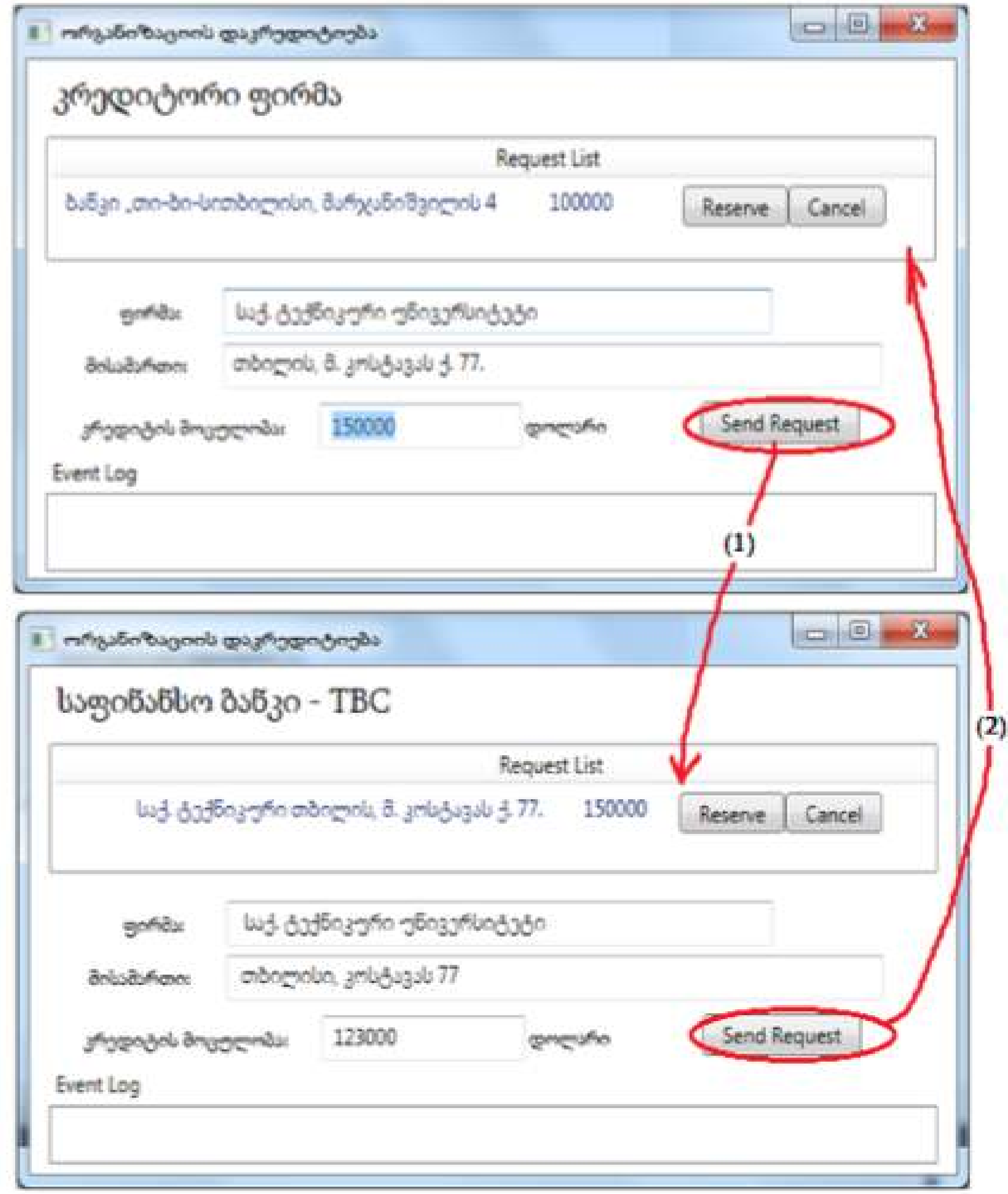
ჩვენ ავაგებთ მარტივ WPF-აპლიკაციას (Windows Presentation Foundation), რომელიც გამოიყენებს კომუნიკაციას სხვადასხვა აპლიკაციას ბიზნესპროცესებს შორის. ბიზნესპროცესები შეიძლება განთავსდეს ვებსერვისში, რომელიც უზრუნველყოფს იდეალურ საშუალებას სამუშაო პროცესის გადაწყვეტილების მისაწოდებლად არამუშა პროცესის კლიენტებისათვის, როგორიცაა ვებაპლიკაციები.

ვებსერვისი იღებს მოთხოვნას, ასრულებს მის სათანადო გადამუშავებას და აბრუნებს პასუხს. ეს, ბუნებრივია, სრულდება Receive და Send ქმედებებით. ვინაიდან ეს აქტიურობები ინტეგრირებულია Windows Communication Foundation (WCF) -თან, ჩვენ შეგვიძლია ადვილად შევქმნათ WCF სერვისები [18].

26.2. Window Form –ის განსაზღვრა

ავაგოთ მომხმარებელთა ინტერფეისის ფორმა, რომელიც გამოდგება, მაგალითად ბანკებისათვის. მისი მაკეტი მოცემულია 26.1 ნახაზზე. შესაბამისი ფორმა აგებულია XAML ფაილის სახით დიზაინის რეჟიმში, როგორც ზემოთ აღვნიშნეთ, WPF ტექნოლოგიის გამოყენებით. ინტერფეისის დაპროეტება და პროგრამული რეალიზაცია ხდება Visual Studio.NET Framework 4.5 გარემოში, C#.NET ენის საფუძველზე. გამოიყენება ინტერფეისის დიზაინისთვის XAML და პროგრამის ლოგიკისათვის C# ენები. ქვემოთ, 26.1 ლისტინგში მოცემულია XAML ფაილის ტექსტი. ფორმის ზედა ნაწილში

„მოთხოვნების სია“ (Request List) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად, მაგალითად, ბანკში გამოიყენება ველები ფორმის შუაში. აქ მიეთითება: - „ფირმა“, - „მისამართი“ და - მოთხოვნილი „კრედიტის მოცულობა“.



ნახ.26.1. მომხმარებელთა ინტერფეისები

შემდეგ ავამოქმედოთ ღილაკი „მოთხოვნის გაგზავნა“ (Send Request). ქვედა მარცხენა კუთხეში Event Log ასახავს ბიზნესპროცესის შეტყობინებას ისე, როგორც კონსოლის რეჟიმშია.

Crediting.xaml ფაილის ლისტინგი ასეთია:

```
<!– – ლისტინგი_26.1 – – –  –>
<Window x:Class="FirmsCrediting.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="ორგანიზაციის დაკრედიტიება" Height="480" Width="650"
      Loaded="Window_Loaded" Unloaded="Window_Unloaded">
   <Grid>
      <Label Height="40" HorizontalAlignment="Left" Margin="12,0,0,0"
         Name="lblBranch" FontSize="20" VerticalAlignment="Top"
            Width="462" FontStretch="Expanded"
             FontFamily="Sylfaen">კრედიტორი ფირმა</Label>
      <ListView x:Name="requestList" Margin="12,42,12,5" Height="150"
             VerticalAlignment="Top" ItemsSource="{Binding}">
         <ListView.View>
           <GridView>
             <GridViewColumn Header="Request List" Width="610">
               <GridViewColumn.CellTemplate>
                 <DataTemplate>
                   <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding Requester.BranchName}" Width="100"/>
        <TextBlock Text="{Binding FirmName}" Width="95"/>
        <TextBlock Text="{Binding Adress}" Width="180"/>
        <TextBlock Text="{Binding CreditQ}" Width="90"/>
        <Button Content="Reserve" Tag="{Binding InstanceID}"
                    Click="Reserve" Width="65"/>
        <Button Content="Cancel"  Tag="{Binding InstanceID}"
                    Click="Cancel" Width="60"/>
                   </StackPanel>
                 </DataTemplate>
               </GridViewColumn.CellTemplate>
             </GridViewColumn>
           </GridView>
         </ListView.View>
```

```
    </ListView>
   <Label Height="30" Margin="27,0,0,205" Name="label5"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="73" HorizontalContentAlignment="Right"
          Content="ფირმა:" FontFamily="Sylfaen"></Label>
    <Label Height="30" Margin="27,0,0,176" Name="label2"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="77" HorizontalContentAlignment="Right"
          Content="მისამართი:" FontFamily="Sylfaen"></Label>
    <Label Height="30" Margin="13,0,0,142" Name="label3"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="151" HorizontalContentAlignment="Right"
          Content="კრედიტის მოცულობა:" FontFamily="Sylfaen">
    </Label>
    <TextBox Height="25" Margin="121,0,0,210"  Name="txtFirmName"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="400" /> <TextBox Height="25" Margin="121,0,0,180"
          Name="txtAdress" VerticalAlignment="Bottom"
          HorizontalAlignment="Left" Width="468" />
    <TextBox Height="25" Margin="0,0,329,147" Name="txtCreditQ"
          VerticalAlignment="Bottom" HorizontalAlignment="Right"
          Width="125" />
    <Button Height="23" Margin="477,0,0,150" Name="btnRequest"
          VerticalAlignment="Bottom" HorizontalAlignment="Left"
          Width="98" Click="btnRequest_Click">Send Request</Button>
    <Label Height="27" HorizontalAlignment="Left" Margin="11,0,0,121"
          Name="label4" VerticalAlignment="Bottom" Width="76">
          Event Log</Label>
    <ListBox Margin="12,0,12,12" Name="lstEvents" Height="111"
          VerticalAlignment="Bottom" FontStretch="Condensed"
          FontSize="10" />
     <Label Content="დოლარი" Height="28" HorizontalAlignment="Left"
          Margin="302,269,0,0"  Name="label1" VerticalAlignment="Top"
          Width="67" FontFamily="Sylfaen" />
  </Grid>
</Window>
```

ფორმის ზედა ნაწილში „მოთხოვნების სია“ (Request List) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად მაგალითად, ბანკში გამოიყენება ველები ფორმის შუაში. აქ მიეთითება „ფირმა“, „მისამართი“ და მოთხოვნილი „კრედიტის მოცულობა“, შემდეგ გაგზავნის ღილაკი „მოთხოვნის გაგზავნა“ (Send Request). ქვედა მარცხენა კუთხეში Event Log ასახავს ბიზნესპროცესის შეტყობინებას ისე, როგორც კონსოლის რეჟიმშია.

26.3. სერვისის პროგრამული რეალიზაცია

ჩვენი სისტემის ClientService.cs კოდის რეალიზაცია ნაჩვენებია 26.2 ლისტინგში.

```
// ----- ლისტინგი 26.2 --- ClientService.cs ---------------
using System;
using System.ServiceModel;   // !!!
namespace FirmsCrediting
{
   public class ClientService : ICreditReservation
   {
      public void RequestCredit(CreditingRequest request)
      {
         ApplicationInterface.RequestCredit(request);
      }
      public void RespondToRequest(CreditingResponse response)
      {
         ApplicationInterface.RespondToRequest(response);
      }
   }
}
```

ეს რეალიზაცია იყენებს ApplicationInterface სტატიკურ კლასს, რომელიც უკვე შექმნილია ჩვენ მიერ. ყოველი მეთოდი უბრალოდ იძახებს ApplicationInterface კლასის შესაბამის მეთოდს.

ApplicationInterface.cs ფაილი მოცემულია 26.3 ლისტინგში.

```
//  ---- ლისტინგი 26.3 --- ApplicationInterface.cs ფაილისთვის ----
using System;
using System.Windows.Controls;
using System.Activities;
namespace FirmsCrediting
{
```

```
 public static class ApplicationInterface
  {
    public static MainWindow _app { get; set; }
    public static void AddEvent(String status)
     {
      if (_app != null)
       {
         new ListBoxTextWriter(_app.GetEventListBox()).WriteLine(status);
        }
    }
 public static void RequestCredit(CreditingRequest request)
   {
     if (_app != null)
         _app.RequestCredit(request);
   }
 public static void RespondToRequest(CreditingResponse response)
   {
     if (_app != null)
          _app.RespondToRequest(response);
     }
   public static void NewRequest(CreditingRequest request)
    {
      if (_app != null)
         _app.AddNewRequest(request);
    }
  }
}
```

ეს მეთოდები, თავის მხრივ, იძახებს შესაბამის მეთოდებს აპლიკაციაში სტატიკური მიმთითებლის გამოყენებით. საჭირო იქნება ამ მეთოდების რეალიზება Crediting.xaml.cs ფაილში.

26.4. ServiceHost -ის რეალიზაცია

აპლიკაციისთვის აუცილებელია ServiceHost-ის რეალიზაცია შემავალი შეტყობინებების მისაღებად. იგი პროექტის Crediting.xaml.cs ფაილში თავსდება კონსტრუქტორის წინ კლასის წევრის სახით (ლისტინგი_26.4).

```
//-- ლისტინგი_26.4 ------
public partial class MainWindow : Window
```

```
{
    private ServiceHost _sh;   // !!!
    public MainWindow()
    {   InitializeComponent();
        ApplicationInterface._app = this;
    }
 ...
```

ServiceHost იწყება მაშინ, როცა ფანჯარა ჩატვირთულია და იხურება, როცა ფანჯარა ამოტვირთულია. მეთოდების დამატება ნაჩვენებია 26.5 ლისტინგში MainWindow კლასისთვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

```
 // --- ლისტინგი 26.5 ------------------
private void Window_Loaded(object sender, RoutedEventArgs e)
 {
   // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და მისი
   // ქსელური მისამართი
  Configuration config = ConfigurationManager
                 .OpenExeConfiguration(ConfigurationUserLevel.None);
  AppSettingsSection app = (AppSettingsSection)config
                     .GetSection("appSettings");
  string adr = app.Settings["BranchAddress"].Value;
      // ფილიალის სახელის გამოტანა ფორმაზე
      lblBranch.Content = app.Settings["Branch Name"].Value;

      // ServiceHost-ის შექმნა
      _sh = new ServiceHost(typeof(ClientService));

     // დასასრულის წერტილის (Endpoint) დამატება
     string szAddress = "http://localhost:" + adr + "/ClientService";
     System.ServiceModel.Channels.Binding bBinding =
            new BasicHttpBinding();
    _sh.AddServiceEndpoint(typeof(ICreditReservation),bBinding,
                  szAddress);
    // ServiceHost-ის გახსნა შეტყობინების მისაღებად (listen)
   _sh.Open();
 }
private void Window_Unloaded(object sender, RoutedEventArgs e)
```

```
{
  // service host-ის დატოვება
    _sh.Close();
}
```

მოვლენის დამმუშავებელი Loaded ხსნის კონფიგურაციის ფაილს და ათავსებს ფილიალის სახელს lblBranch -მართვის ელემენტში, ამიტომაც ფორმა ასახავს ფირმის სახელს. შემდეგ იქმნება ServiceHost თანამგზავრი (passing) ClientService კლასის. შემდეგ იგი აკონფიგურირებს დასასრულის წერტილს ServiceHost-თვის, იყენებს რა ცნობილი მისამართის, მიზმისა და კონტრაქტის სამეულს.

Unloaded მოვლენის დამმუშავებელი უბრალოდ ხურავს ServiceHost-ს, ასე რომ მეტი აღარ მოხდება შეტყობინებების მიღება.

აპლიკაციის ამუშავება: საჭიროა აპლიკაციის რამდენიმე კოპიოს ერთად გაშვება, თითოეული თავისი კონფიგურაციის ფაილის ვერსიით. თავიდან საჭიროა F6 კლავიშის ამოქმედება Solution-ის (გადაწყვეტის) აღსადგენად და კომპილატორის შენიშვნების აღმოსაფხვრელად.

შევქმნათ ახალი ფოლდერი BankRequest -ფოლდერის ქვეშ, რომელიც იძახებს ბანკებს. შემდეგ დავაკოპიროთ ბანკის ფოლდერში ფაილები, რომლებიც ზემოთ შევქმენით;

```
FirmsCrediting.exe    // Application
FirmsCrediting.exe.config   // XML Configuration file
FirmsCrediting.pdb   // program debug database
```

სისტემის ამუშავების შემდეგ გვექნება ორი სამუშაო ფანჯარა, მაგალითად, ერთი კრედიტორი ფირმის, მეორე საფინანსო ბანკის. მათ შორის შესაძლებელი იქნება ინფორმაციისა და შეტყობინებების გაცვლა, რისი მიღწევაც გვინდოდა (ნახ.26.1).

წიგნის N1 დანართში წარმოდგენილია Web-სერვისების პროგრამული რეალიზაცია მომხმარებელთა ინტერფეისების აგებისა და Web-ის უსაფრთხოების დაცვისათვის ASP.NET ტექნოლოგიის გამოყენებით.

XXVII თავი
WPF აპლიკაციის აგება საპრობლემო სფეროში „სატრანსპორტო გადაზიდვები"

27.1 მულტიმოდალური გადაზიდვების მართვის საინფორმაციო სისტემის აგების ამოცანა

განიხილება ტვირთების მულტიმოდალური გადაზიდვების მართვის ავტომატიზებული სისტემის მონაცემთა ბაზის დაპროექტების, მისი პროგრამული რეალიზაცია და მომხმარებელთა ინტერფეისების აგების საკითხები დაპროექტების CASE - და დაპროგრამების ჰიბრიდული ტექნოლოგიებით [8].

წინამდებარე პარაგრაფში შემოთავაზებულია მულტიმოდალური გადაზიდვების (გემი, რკინიგზა, ავტო- და საჰაერო ტრანსპორტი) საპრობლემო სფეროს კონცეპტუალური სქემები კლიენტის (ტვირთის მფლობელი), ტვირთის (გადაზიდვის ობიექტი) და მიმწოდებელის (გადამზიდავი) ცხრილებით, ობიექტ-როლური და არსთა დამოკიდებულების მოდელირების (ORM/ERM) ინსტრუმენტებით Visual Studio.NET გარემოში და Ms SQL Server პაკეტით [22,23]. აგებულია მონაცემთა ბაზის განახლების ფუნქციების ინტერფეისი დაპროგრამების (WPF, C#, XAML) ინტეგრირებულ გარემოში [1].

მულტიმოდალური გადაზიდვების პროცესი, რომლის ძირითადი მიზანი ტვირთების ტრანსპორტირებაა მიმწოდებლიდან დამკვეთამდე, არის მომსახურების განაწილებული სისტემა. მარტივად რომ წარმოვიდგინოთ, მიმწოდებელი (Supplier_ID) აგზავნის ტვირთს (Freight_ID) დამკვეთის (Cleint_ID) მისამართზე (Client_Address) [22].

როგორც ცნობილია, კლიენტსა (ტვირთის მფლობელი) და მიმწოდებელს (გადამზიდავი) შორის ხელშეკრულებას აფორმებს ექსპედიტორი (შუამავალი), რომელსაც აქვს საჭირო ინფორმაცია ადგილობრივი და საერთაშორისო გადაზიდვების აგენტების, მარშრუტებისა და შესაბამისი ფასების შესახებ (ამ უკანასკნელის ცვლილებების შესახებაც) და სხვ.

ქვემოთ მოცემული გვაქვს მულტიმოდალური გადაზიდვების პროცესის ინფრასტრუქტურის ძირითადი ობიექტებისა და მათი თვისებების (ატრიბუტების) სემანტიკური აღწერა, რაც მომავალში გამოყენებული იქნება ავტომატიზებული სისტემის მონაცემთა ბაზის ასაგებად [30].

ტვირთი: იდენტიფიკატორი, ტიპი, მდგომარეობა, შეფუთვის ტიპი, ერთეულის ზომები (სიგრძე, სიგანე, სიმაღლე), ერთეულის მოცულობა, ჯამური მოცულობა, ერთეულის წონა, ერთეულის რაოდენობა, ჯამური წონა, უსაფრთხოობა, საბაჟო კოდი, გამგზავნი, მიმღები, საწყისი მდებარეობა, საბოლოო მდებარეობა და სხვ.;

კლიენტი: იდენტიფიკატორი, დასახელება/ვინაობა, იურიდიული/ფიზიკური პირი, მისამართი, ტელეფონი, ელ_მისამართი და სხვ.;

მიმწოდებელი: იდენტიფიკატორი, დასახელება, იურიდიული/ფიზიკური პირი, მისამართი, ტელეფონი, ელ_მისამართი, ფაქსი, ტრანსპორტის სახე და სხვ.;

გემი: იდენტიფიკატორი, ტიპი, ამწეთი/უამწეო, მდგომარეობა, სასაწყობო ლიმიტი, ტვირთამწეობა, ტვირთმოცულობა, ადგილმდებარეობა და სხვ.;

თვითმფრინავი: იდენტიფიკატორი, ტიპი, მდგომარეობა, ტვირთმოცულობა, გადასაზიდი ერთეულის დასაშვები ზომები (სიგრძე, სიგანე, სიმაღლე) ადგილ-მდებარეობა და სხვ.;

ავტოტრანსპორტი: იდენტიფიკატორი, ტიპი, მდგომარეობა, ტვირთმოცულობა, გადასაზიდი ერთეულის დასაშვები ზომები (სიგრძე, სიგანე, სიმაღლე), გადასაზიდი ერთეულის დასაშვები წონა, მაქსიმალური დატვირთვა, ადგილმდებარეობა და სხვ.;

სარკინიგზო სატვირთო ვაგონი: იდენტიფიკატორი, ტიპი, ტვირთამწეობა, მოცულობა, დასაშვები დატვირთვა, ადგილმდებარეობა, მიმწოდებლის იდენტიფი-კატორი, მდგომარეობა და სხვა.;

საწყობი: იდენტიფიკატორი, სახე, ფართობი, სართული, დაკავებულობის პროცენტი, დასაშვები დატვირთვა, ადგილმდებარეობა, მისამართი, მიკუთვნება რაიონზე და სხვ.;

გადაზიდვის ხელშეკრულება კლიენტთან: იდენტიფიკატორი, საწყისი მდებარეობა, თარიღი_1, საბოლოო მდებარეობა, თარიღი_2, გადაზიდვის ღირებულება, გადახდილი თანხა, გადახდის_თარიღი, მდგომარეობა და სხვ.

27.2. მონაცემთა ბაზის დაპროექტება და რეალიზაცია

მულტიმოდალური გადაზიდვების საპრობლემო სფეროს მონაცემთა ბაზის ასაგებად საჭიროა მისი ობიექტების, ობიექტთაშორისი რელაციური კავშირების (პრედიკატების) და კონკრეტული ინფორმაციის გადატანა ბაზაში. ამისათვის ჩვენ ვიყენებთ ობიექტროლური მოდელირების CASE ინსტრუმენტულ საშუალებას, როგორიცაა Natural ORM Archchitect [14]. ამ სფეროში არსებული ფაქტების აღწერა ხორციელდება კატეგორიალური მიდგომისა (სალაპარაკო ენის გრამატიკული წესები) და ალგებრულ-ლოგიკური თეორიის კანონების საფუძველზე [15]. მაგალითად,

- **f1: კლიენტს აქვს ტვირთი;**
- **f2: კლიენტს აქვს იდენტიფიკატორი;**
- **f3: ტვირთს აქვს იდენტიფიკატორი;**
- **f4: ტვირთს აქვს გადასატანი მისამართი;**
- **f5: მიმწოდებელს აქვს ტრანსპორტი;**

და ა.შ.

27.1 ცხრილში მოცემულია ჩვენი ობიექტების შესაბამისი ფაქტების აღწერის ფრაგმენტი NORMA გარემოში.

ფაქტის ტიპები (პრედიკატები) ცხრ.27.1

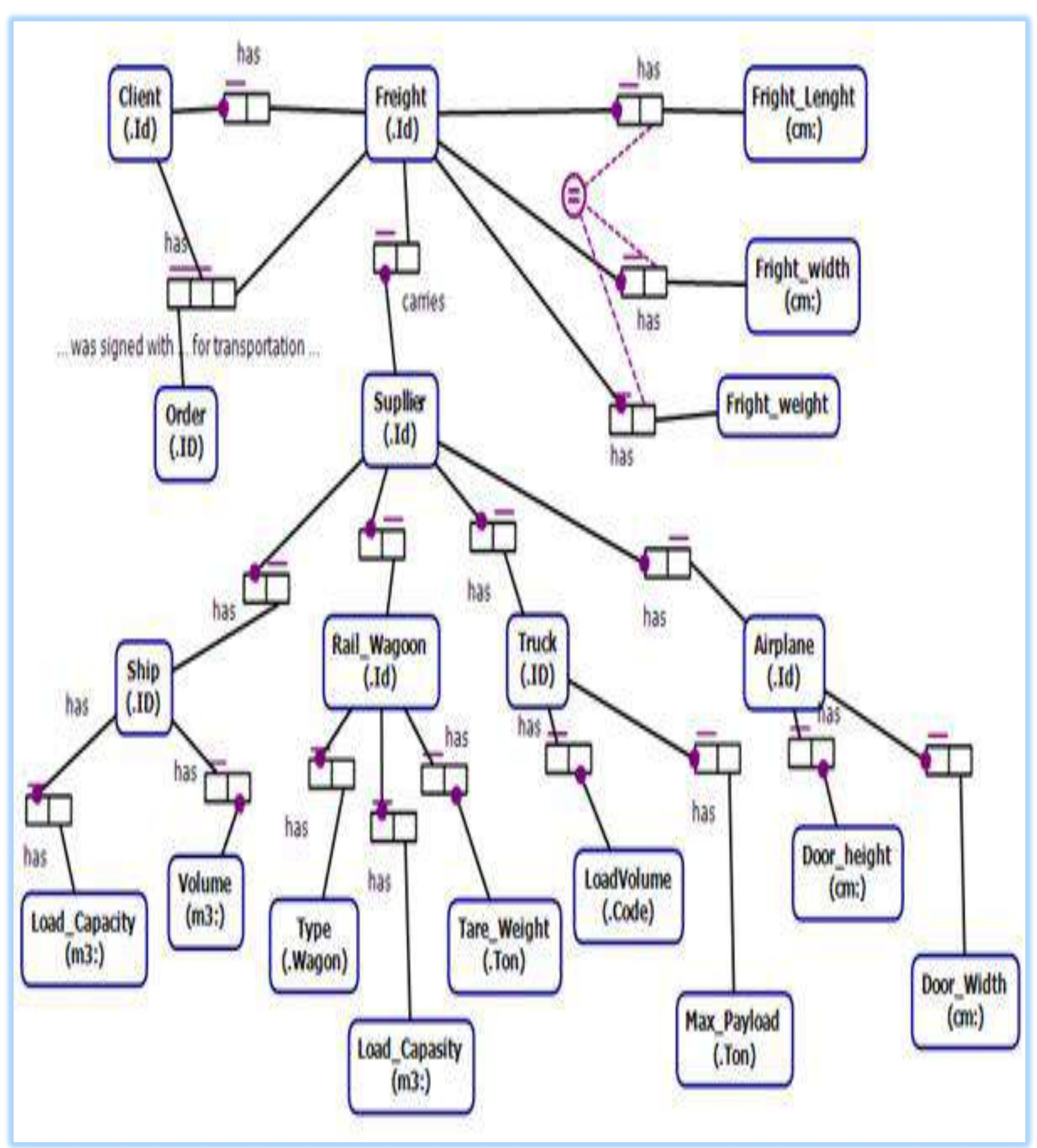
Fact Types

- AirplaneHasAirplane_Id
- AirplaneHasDoor_height
- AirplaneHasDoor_Width
- ClientHasClient_Id
- ClientHasFreight
- Door_heightHascmValue
- Door_WidthHascmValue
- FreightHasFreight_Id
- FreightHasFright_Lenght
- FreightHasFright_weight
- FreightHasFright_width
- FreightIsConvayedBySupllier
- Fright_LenghtHascmValue
- Fright_weightHas[Missing]
- Fright_widthHascmValue
- Load_CapacityHasm3Value
- Load_CapasityHasm3Value
- LoadVolumeHasLoadVolume_Code
- Max_PayloadHasMax_Payload_Ton
- OrderHasOrder_ID
- OrderWasSignedWithClientForTransportationFreight
- Rail_WagoonHasLoad_Capasity
- Rail_WagoonHasRail_Wagoon_Id
- Rail_WagoonHasTare_Weight
- Rail_WagoonHasType
- ShipHasLoad_Capacity
- ShipHasShip_ID
- ShipHasVolume
- SupllierCarriesFreight
- SupllierHasAirplane
- SupllierHasRail_Wagoon
- SupllierHasShip
- SupllierHasSupllier_Id
- SupllierHasTruck
- Tare_WeightHasTare_Weight_Ton

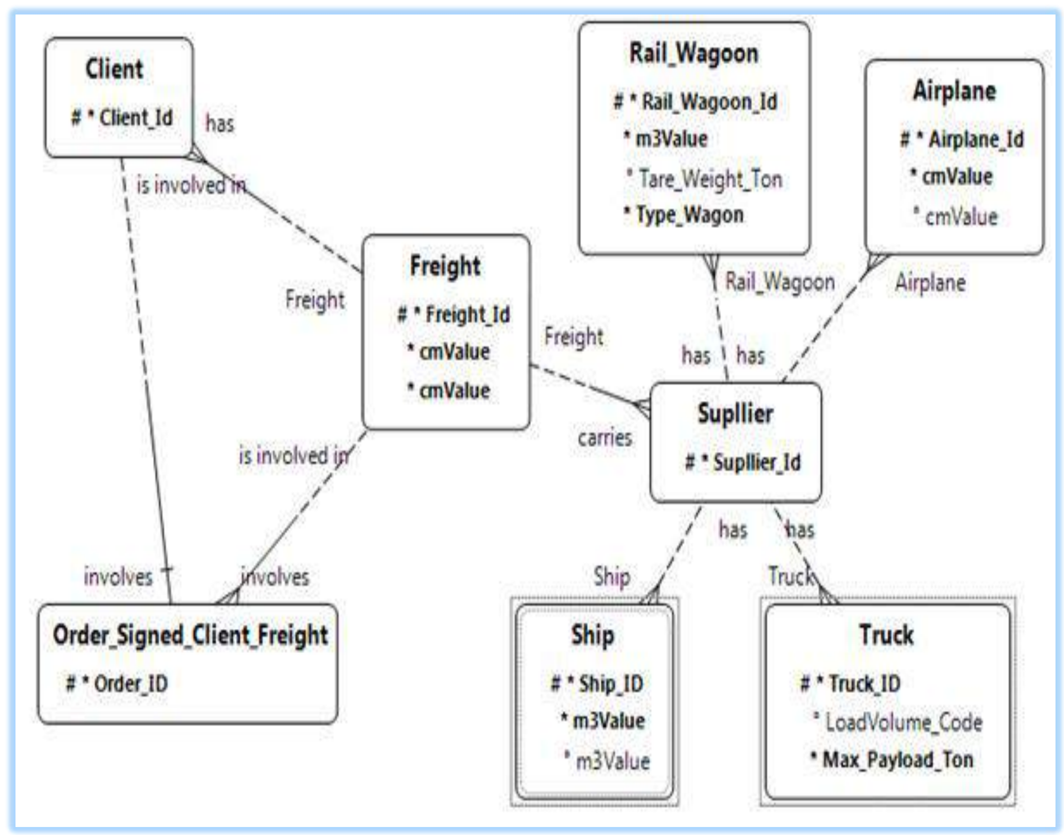
მულტიმოდალური გადაზიდვების საპრობლემო სფეროს ფაქტების აღწერისა და მათი Visual Studio.NET + NORMA ინტეგრირებულ სამუშაო გარემოში გადატანის შემდეგ ვიღებთ 27.1 ნახაზზე ნაჩვენებ კონცეპტუალურ სქემას, რომელიც ობიექტროლური მოდელია.

დიდი მართკუთხედებით აისახება ობიექტები (მაგალითად, Client, Freight, Airplane და სხვა) და მათი თვისებები (Load_Capacity, Volume, Freight_Lenght და სხვა), პატარა მართკუთხედებით კი – პრედიკატები მათ შორის (მაგალითად, „Ship has Load_Capacity" და ა.შ.). აქ „მრგვალთავა" კავშირის ხაზებით და პრედიკატებზე `ხაზგასმით" გვეძლევა დამატებითი ინფორმაცია ობიექტებსშორის მრავლობითი კავშირების შესახებ, როგორიცაა მაგალითად, 1:1, 1:N და M:N.

აქვე შესაძლებელია მივიღოთ, ე.წ. რიჩარდ ბარკერის დიაგრამა, რომელშიც ასეთი კავშირები ობიექტებს შორის უკეთესად ჩანს (ნახ.27.2). იგი ჩვეულებრივი არსთა-დამოკიდებულების მოდელია, რომელსაც იყენებენ Oracle CASE მიმდევრები [17].



ნახ.27.1. საპრობლემო სფეროს კონცეპტუალური სქემა:
ობიექტ-როლური მოდელი

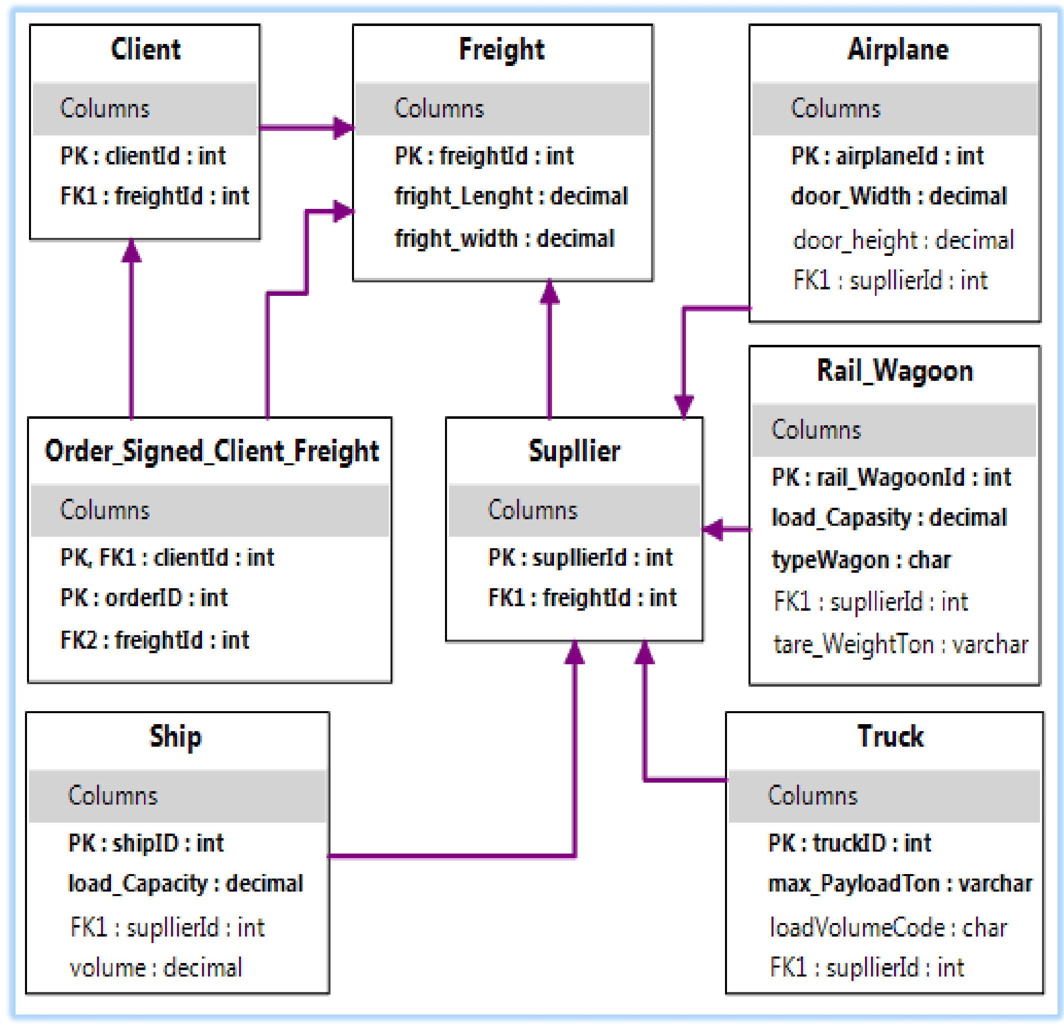


ნახ.27.2. ბარკერის დიაგრამა (Barker ER view)

ჩვენი შემდეგი ბიჯი დაკავშირებულია OღM დიაგრამით მიღებული კონცეპტუალური სქემიდან ეკვივალენტური ER მოდელის დაპროექტებასთან. ეს პროცესი დიალოგურ რეჟიმში ხორციელდება, სადაც მომხმარებელს შეუძლია სისტემის მიერ შემოთავაზებულ სქემაში შეიტანოს საჭირო ცვლილებები.

27.3 ნახაზზე ნაჩვენებია მულტომოდალური გადაზიდვების საპრობლემო სფეროს ER სქემა. იგი ახლოსაა რელაციური ბაზების დაპროექტების ობიექტ-ორიენტირებული მიდგომის ტრადიციულ დიაგრამებთან, რომლებიც აიგება MsVisio, Enterprise Architect, Rational Rose და სხვა ინსტრუმენტებით [27].

ჩვენ აღვწერეთ კონცეპტუალური სქემის ავტომატიზებული ფორმირების პროცესი ORM->ERM. აქვე შეიძლება აღვნიშნოთ, რომ 27.3 ნახაზზე მოცემულია მხოლოდ ფრაგმენტი და მისი სრული ვერსიის მიღება შესაძლებელია პირველ ეტაპზე ფაქტების თანდათანობით დამატებით.



ნახ.27.3. კონცეპტუალური სქემა (ER Model)

27.2 ცხრილში მოცემულია სისტემის მიერ ფორმირებული მულტიმოდალური გადაზიდვების საპრობლემო სფეროს კონცეპტუალური სქემის შესაბამისი ფაქტების ვერბალიზაციის ლისტინგი.

ფაქტების ვერბალიზაცია ცხრ.27.2

Freight is an entity type. Reference Scheme: Freight has Freight_Id. Reference Mode: .Id. Fact Types: Freight has Freight_Id. Freight is convayed by Supllier. Supllier carries Freight. Freight has Freight_Lenght. Freight has Freight_width. Freight has Freight_weight. Client has Freight. Order was signed with Client for transportation Freight. **Client** is an entity type. Reference Scheme: Client has Client_Id. Reference Mode: .Id. Fact Types: Client has Client_Id. Client has Freight. Order was signed with Client for transportation Freight. Each Client has exactly one Freight. It is possible that more than one Client has the same Freight. **Truck** is an entity type. Reference Scheme: Truck has Truck_ID. Reference Mode: .ID. Fact Types: Truck has Truck_ID. Truck has LoadVolume. Truck has Max_Payload. Supllier has Truck. Each Supllier has some Truck. For each Truck, at most one Supllier has that Truck.	**Supplier** is an entity type. Reference Scheme: Supllier has Supplier_Id. Reference Mode: .Id. Fact Types: Supllier has Supplier_Id. Freight is convayed by Supplier. Supplier carries Freight. Supplier has Ship. Supplier has Rail_Wagoon. Supplier has Truck. Supplier has Airplane. **Order** is an entity type. Reference Scheme: Order has Order_ID. Reference Mode: .ID. Fact Types: Order has Order_ID. Order was signed with Client for transportation Freight. For each Order and Client,that Order was signed with that Client for transportation at most one Freight. This association with Order, Client provides the preferred identification scheme for Order was signed with Client for Transportation Freight. **Ship** is an entity type. Reference Scheme: Ship has Ship_ID. Reference Mode: .ID. Fact Types: Ship has Ship_ID. Supplier has Ship. Ship has Load_Capacity. Ship has Volume. Each Supplier has some Ship. For each Ship, at most one Supllier has that Ship. It is possible that the same Supllier has more than one Ship.

Airplane is an entity type. Reference Scheme: Airplane has Airplane_Id. Reference Mode: .Id. Fact Types: Airplane has Door_height. Airplane has Door_Width. Airplane has Airplane_Id. Supllier has Airplane. Each Supllier has some Airplane. For each Airplane, at most one Supllier has that Airplane. It is possible that the same Supllier has more than one Airplane.	**Rail_Wagoon** is an entity type. Reference Scheme: Rail_Wagoon has Rail_Wagoon_Id. Reference Mode: .Id. Fact Types: Rail_Wagoon has Rail_Wagoon_Id. Supplier has Rail_Wagoon. Rail_Wagoon has Type. Rail_Wagoon has Load_Capasity. Rail_Wagoon has Tare_Weight. Each Supllier has some Rail_Wagoon. For each Rail_Wagoon, at most one Supllier has that Rail_Wagoon. It is possible that the same Supplier has more than one Rail_Wagoon.

მომდევნო ბიჯზე, მიღებული ER-მოდელის შესაბამისად, სისტემა გვაძლევს DDL-ფაილს (მონაცემთა აღწერის ენა), რომელიც ჩვენ შემთხვევაში გამოიყენება SQL Server ბაზის ასაგებად [30].

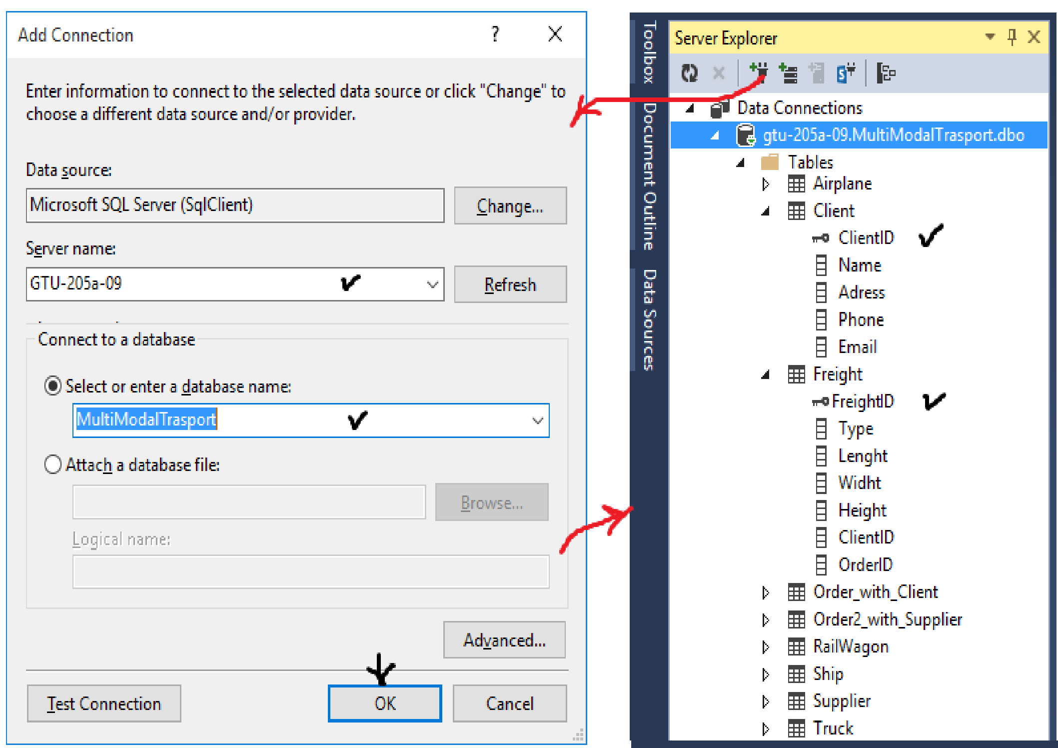
27.3. მონაცემთა ბაზასთან მუშაობის ინტერფეისის აგება

მულტიმოდალური გადაზიდვების ბაზის დაპროექტებისა და მისი DLL-ფაილით SQL Server 2012-ში რეალიზაციის შემდეგ, საჭიროა აიგოს მომხმარებლის ინტერფეისის პროგრამა Visual Studio.NET 2013/15 ინტეგრირებულ გარემოში დაპროგრამების ჰიბრიდული ტექნოლოგიის გამოყენებით, WPF (Windows Presentation Foundation) [8].

ჰიბრიდული ტექნოლოგიის კონცეფცია გულისხმობს, რომ ამ მეთოდოლოგიით შესაძლებელია როგორც ვინდოუსის, ისე ვებაპლიკაციების აგება (შედეგები აისახება ვინდოუსის ფორმაზე ან რომელიმე ინტერნეტ ბრაუზერში).

თავდაპირველად საჭიროა ახალი პროექტის შექმნა Visual Studio.NET-ში, შემდეგ კი ჩვენი მონაცემთა ბაზის მიერთება ამ პროექტთან. 27.4 ნახაზზე ნაჩვენებია ბაზის მიერთების (Data Connections) პროცედურა.

განახლდება სერვერის სახელი (მაგალითად, GTU-205a-09) და აირჩევა მასში მონაცემთა ბაზის სახელი (MultiModalTransport), ბოლოს OK და შედეგად მივიღებთ ნახაზის Server Explorer-ში და Airplane, Client, Freight სხვა ცხრილებს.



ნახ.27.4. მონაცემთა ბაზის დაკავშირება C#.NET პროგრამასთან

მონაცემთა ბაზის ცხრილებთან მუშაობის ინტერფეისის სქემა წინასწარ შეთანხმებულია პროგრამული სისტემის მომხმარებელთან. მომხმარებელს, რომელმაც არ იცის მონაცემთა ბაზასთან მუშაობა დაპროგრამების ენების გამოყენებით (მაგალითად, DDL და DML), მაშინ მისთვის უნდა აიგოს დიალოგში სამუშაო ინტერფეისი, რომელიც ადვილად შეასრულებს ბაზაში მონაცემების ძებნის (Select) ან ცხრილების განახლების ოპერაციებს (Add, Update, Delete).

27.5 ნახაზზე ნაჩვენებია ასეთი ინტერფეისის ერთ-ერთი ვარიანტი, რომელშიც შესაძლებელია მომხმარებელმა, როგორც ბაზის ადმინისტრატორმა, განახორციელოს თავისი ფუნქციები (რეალურ მონაცემთა ბაზასთან უშუალოდ წვდომის გარეშე), ამჯერად, მაგალითად, კლიენტების ბაზასთან.

ნახ.27.5. ინტერფეისის საილუსტრაციო მაგალითის
მაკეტი

ამ ფორმის დიზაინი აიგება Visual Studio.NET გარემოში WPF ინსტრუმენტების პანელის და XAML-ენის გამოყენებით. შესაბამისი კოდის ფრაგმენტი მოცემულია 27.1 ლისტინგში.

```
<--  ლისტინგი  27.1  ----  XAML ფაილი  ------>
<Window x:Class="WpfAppSQL_ListView.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title ="ტვირთების კლიენტების სია"
            Height="375" Width="520" Loaded="Window_Loaded"
                  Background="White">
      . . .
      <GridView>
         <GridViewColumn Header="კლიენტის_ID"
             DisplayMemberBinding=
                     "{Binding Path=ClientID}"></GridViewColumn>
         <GridViewColumn Header="გვარი" DisplayMemberBinding=
                     "{Binding Path=Name}"></GridViewColumn>
```

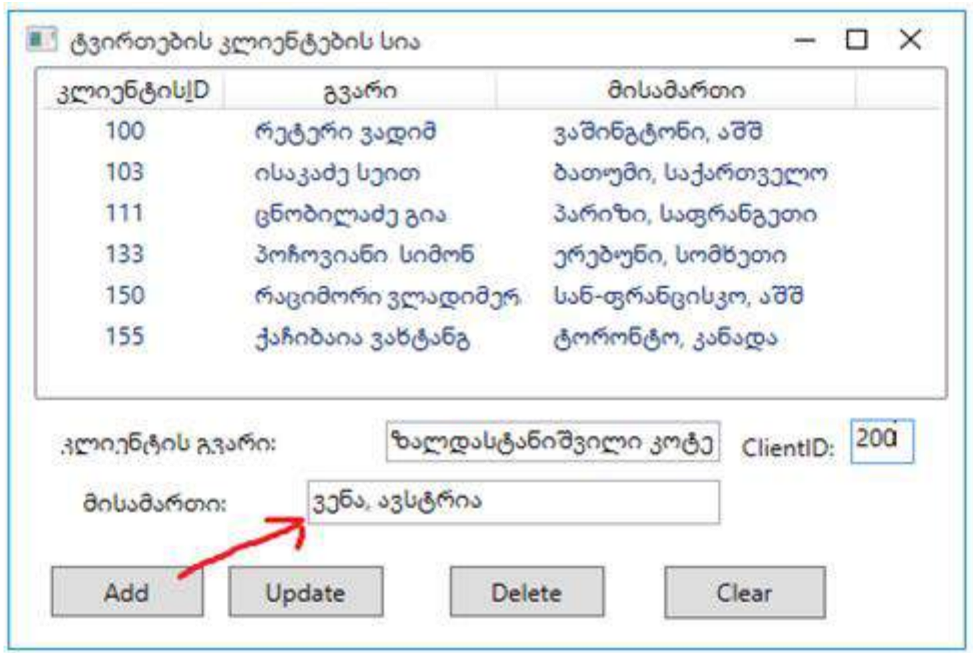
```
      <GridViewColumn Header="მისამართი"
                        DisplayMemberBinding=
                "{Binding Path=Adress}"></GridViewColumn>
   </GridView>
      . . .
</Window>
```

ინტერფეისის ფორმაზე ჩანს LisBox სამი სვეტით, კლიენტების სიაში რამდენიმე მონაცემის გამოსატანად, სამი TextBox, საძებნი ან დასამატებელი სტრიქონის მონაცემების ჩასაწერად და ოთხი Button, ბაზაში სხვადასხვა ფუნქციის შესასრულებლად (Add, Update, Delete). Clear ღილაკი გამოიყენება მონაცემთა შესატანი ტექსტბოქსების გასაწმენდად.

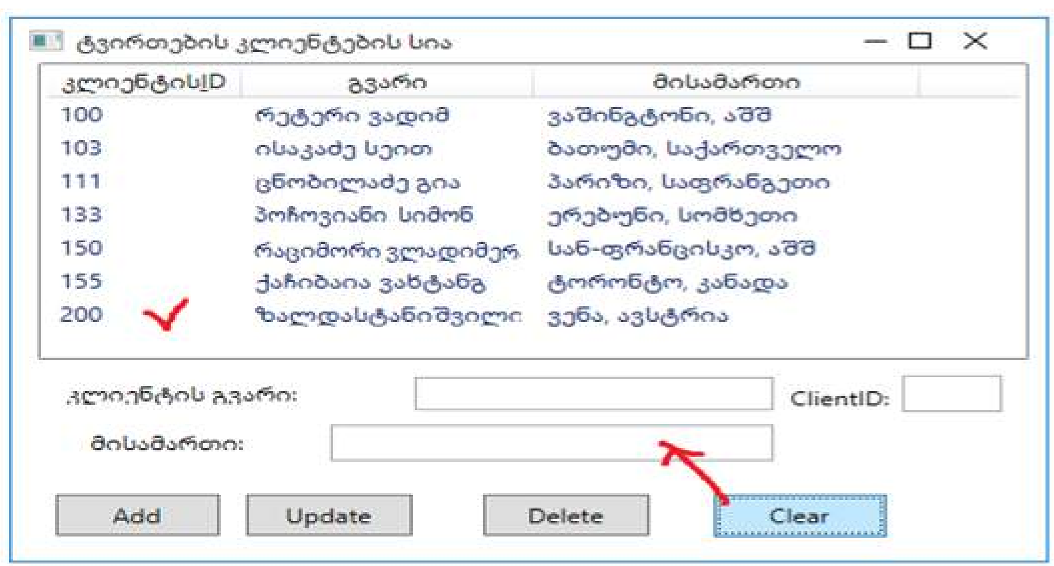
საჭიროა პროგრამის მუშაობის ლოგიკის დაპროგრამება C#.NET ენაზე, კერძოდ, უნდა დაიწეროს მეთოდები, რომლებიც მოემსახურება შესაბამისი ღილაკების ამოქმედებით ინიციალიზებული მოვლენების დამუშავებას.

განვიხილოთ მაგალითები.

დავამატოთ მონაცემთა ბაზას ახალი კლიენტი (ნახ.27.6-ა). უნდა შეივსოს კლიენტის_გვარი, კლიენტის ID და მისამართი.



ნახ.27.6-ა. ახალი ჩანაწერის დამატება (Add ღილაკით)



ნახ.27.61-ბ. ახალი ჩანაწერი დაემატა და ტექსტბოქსები გაიწმინდა

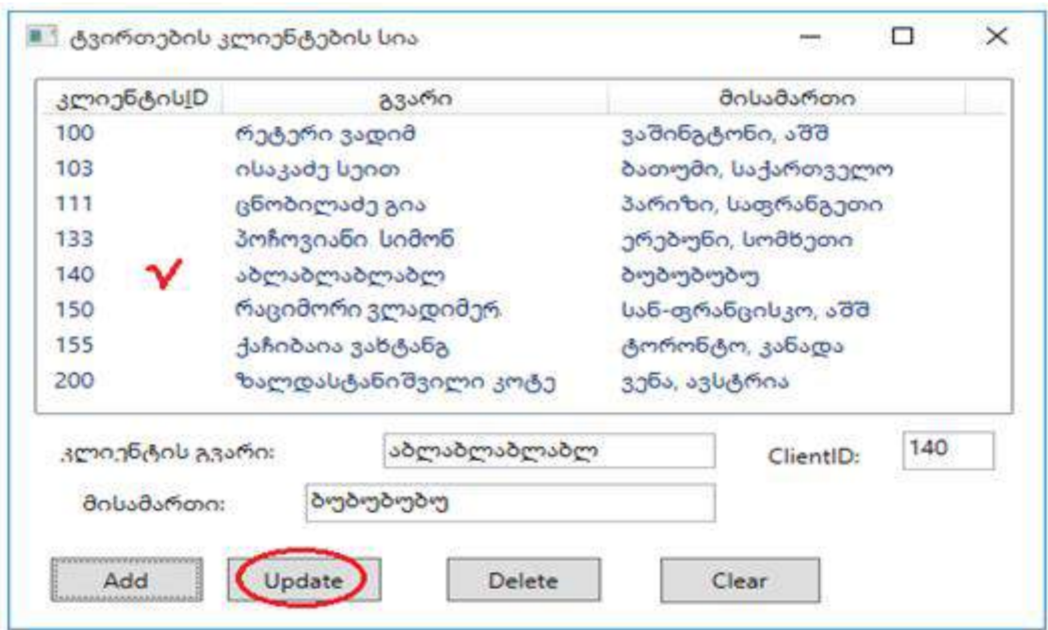
Add – ღილაკის პროგრამის ტექსტი მოცემულია 27.2 ლისტინგში.

```
// --- ლისტინგში_27.2 – Add ---------------
private void btnAdd_Click(object sender, RoutedEventArgs e)
  {
    string Name = textBox1.Text;
    string Adress = textBox2.Text;
    string ClientID = textBox3.Text;
    SqlConnection con = new SqlConnection (@"Data Source=GTU-205a-09;
       Initial Catalog=MultiModalTrasport;Integrated Security=True");
    con.Open();
    SqlCommand comm = new SqlCommand("insert into Client(Name,Adress,
        ClientID) values(@Name, @Adress, @ClientID)", con);

    comm.Parameters.AddWithValue("@Name", textBox1.Text);
    comm.Parameters.AddWithValue("@Adress", textBox2.Text);
    comm.Parameters.AddWithValue("@ClientID", textBox3.Text);
    comm.ExecuteNonQuery();
    con.Close();
    ShowData();
}
```

შენიშვნა: SqlConnection სტრიქონით ხდება MultiModalTrasport მონაცემთა ბაზასთან ავტომატური დაკავშირება, ხოლო SqlCommand–ით ხორციელდება Insert ოპერაციის შესრულება. აქ ShowData()მეთოდია, რომელიც უზრუნველყოფს შედეგების ასახვას ინტერფეისის ფორმაზე; მისი შესაბამისი C#- პროგრამის კოდი ნაჩვენებია 27.3 ლისტინგში.

```
//--- ლისტინგი_27.3 --- ShowData() ----
public void ShowData()
{ SqlConnection con = new SqlConnection(@"Data Source=GTU-205a-09;
      Initial Catalog=MultiModalTrasport;Integrated Security=True");
  con.Open();
  SqlCommand comm = new SqlCommand("Select ClientID, Name, Adress from Client", con);
  DataTable dt = new DataTable();
  SqlDataAdapter da = new SqlDataAdapter(comm);
  da.Fill(dt);
  listView1.DataContext = dt.DefaultView;
}
```

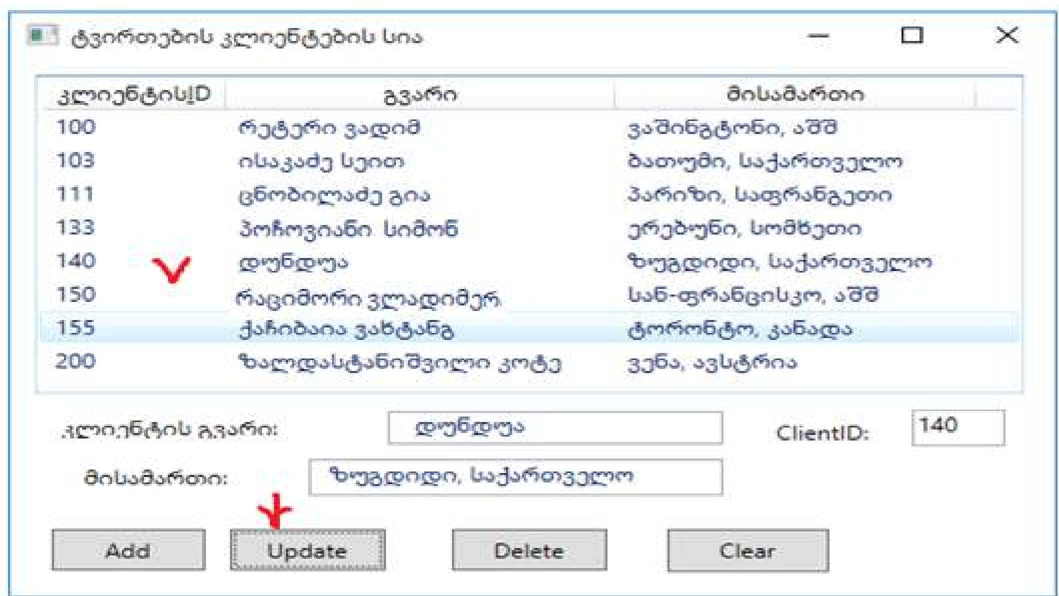
მონაცემთა ბაზაში საჭიროა შეიცვალოს ინფორმაცია, შეცდომაა 140-კლიენტის ჩანაწერში (ნახ.27.7-ა). საჭიროა Update ღილაკის დაპროგრამება, რომლის ტექსტი 27.4 ლისტინგშია მოცემული.



ნახ.27.7-ა. მონაცემთა შეცვლა – ბაზის განახლება
(Update ღილაკით)

```
//---  ლისტინგი 27.4 ---  Update ----------
private void btnUpdate_Click(object sender, RoutedEventArgs e)
 {
  if (listView1.SelectedItems.Count > 0)
   {
    DataRowView drv = (DataRowView)listView1.SelectedItem;
    string id = drv.Row[0].ToString();
    SqlConnection con = new SqlConnection(@"Data Source=GTU-205a-09;
         Initial Catalog=MultiModalTrasport;Integrated Security=True");
    con.Open();
    SqlCommand comm = new SqlCommand("update Client set
         Name=@Name,Adress=@Adress where ClientID=@ClientID", con);
   comm.Parameters.AddWithValue("@ClientID", id);
   comm.Parameters.AddWithValue("@Name", textBox1.Text);
   comm.Parameters.AddWithValue("@Adress", textBox2.Text);
   comm.ExecuteNonQuery();
     con.Close();
     ShowData();
  }
}
```

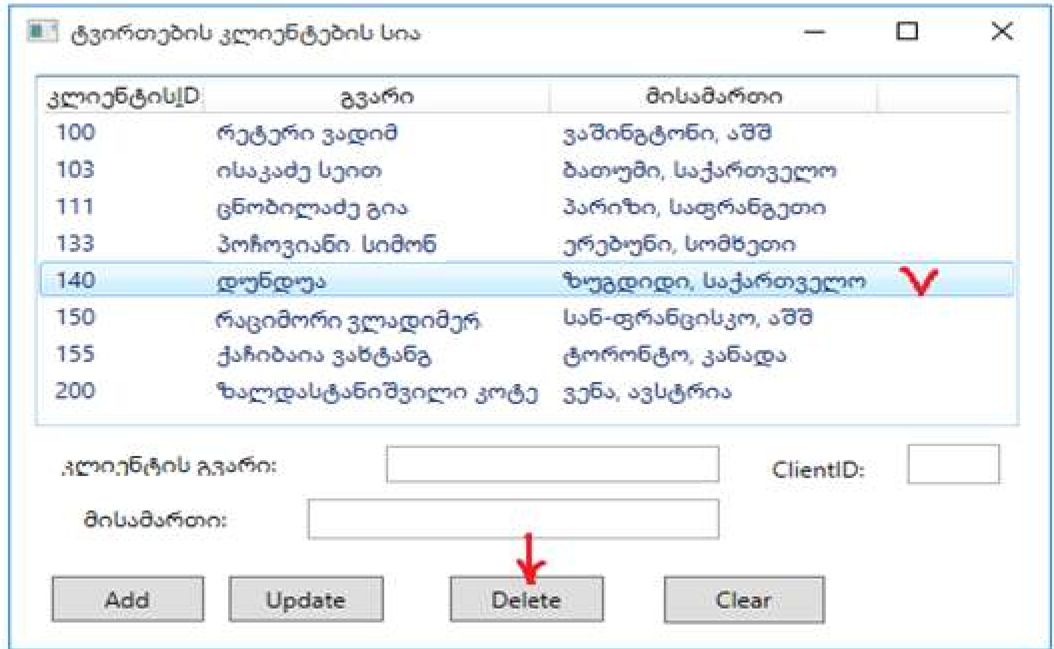
შედეგად მივიღებთ 27.7-ბ ნახაზზე მოცემულ სიტუაციას, ბაზაში 140-ე კლიენტის გვარი და მისამართი განახლებულია.



ნახ.27.7-ბ. მონაცემები შეიცვალა ბაზაში ჩაწერით

დავუშვათ, რომ გვჭირდება კლიენტის მონაცემების წაშლა, რომლის გვარია „დუნდუა".

უნდა მოვნიშნოთ ეს სტრიქონი (ნახ.27.8-ა).



ნახ.27.8-ა. წასაშლელად მომზადებული სტრიქონი

ამჯერად უნდა დაპროგრამირდეს ელეტე ღილაკი. კოდის ტექსტი მოცემულია 27.5 ლისტინგში.

```
//-----ლისტინგი 27.5---Delete --------
private void btnDelete_Click(object sender, RoutedEventArgs e)
 {
  if (listView1.SelectedItems.Count > 0)
   {
    DataRowView drv = (DataRowView)listView1.SelectedItem;
    string id = drv.Row[0].ToString();
    SqlConnection con = new SqlConnection(@"Data Source=GTU-205a-09;
        Initial Catalog=MultiModalTrasport;Integrated Security=True");
    con.Open();
    SqlCommand comm = new SqlCommand("delete from Client where
                              ClientID=@ClientID", con);
     comm.Parameters.AddWithValue("@ClientID", id);
```

```
    comm.ExecuteNonQuery();
    ShowData();
  }
}
```

ახალი მონაცემების შეტანის დროს Add ღილაკის ამოქმედების შემდეგ თუ ტექსტბოქსებში დარჩა უკვე შეტანილი მონაცემები, მაშინ ისინი უნდა გასუფთავდეს, გამზადდეს ახალი მონაცემების შესატანად. ამის ფუნქციას ასრულებს Clear ღილაკი, რომლის კოდის ტექსტი მოცემულია 27.6 ლისტინგში.

```
//---- ლისტინგი 27.6 -- Clear -----------
private void btnClear_Click(object sender, RoutedEventArgs e)
  {
    textBox1.Text = "";
    textBox2.Text = "";
    textBox3.Text = "";
  }
```

XXVIII თავი
WPF-აპლიკაციის აგება საპრობლემო სფეროში „ელექტრონული არჩევნები“

ელექტრონული საარჩევნო სისტემა მიეკუთვნება რთული და დიდი სისტემების კლასს, რომლის ობიექტორიენტირებული მოდელირების, ანალიზის, დაპროექტებისა და შემდგომი პროგრამული რეალიზაციის საკითხები მეტად მნიშვნელოვანია. იგი ეფუძნება ქვეყნის დემოკრატიული პრინციპებისა და საზოგადოების მაღალზნეობრივი კულტურის განვითარებას.

განიხილება დაცული სახელმწიფო ქსელის აგების კონცეფცია და მისი არქიტექტურა. დაპროექტებულია ელექტრონული საარჩევნო სისტემისათვის საჭირო და აუცილებელი მულტიმედიური მონაცემთა რელაციური ბაზები. ამ თვალსაზრისით განიხილება მოდელირების კატეგორიალური მეთოდების და დაპროექტების ობიექტორიენტირებული, CASE-ტექნოლოგიების გამოყენება.

სისტემური ანალიზის შედეგად შერჩეულია ელექტრონული საარჩევნო სისტემისათვის აუტენთიფიკაციის სხვადასხვა ტიპის კომბინირება და მათი ინტეგრირება მულტიმედიურ მონაცემთა ბაზებში. ესენია, თითის ანაბეჭდის სკანირების კომპონენტი, ხმის აუდიო ჩანაწერი, ბიომეტრული ფოტოსურათი და ელექტრონული ხელმოწერა [40,84]. ზემოხსენებული მოდულების ინტეგრაციით იზრდება უსაფრთხოება და ამავდროულად პირდაპირპროპორციულად იმატებს საიმედოობისა და ნდობის ხარისხი. დამუშავებულია კლიენტ-სერვერ არქიტექტურის ლოგიკურად ერთიანი და ფიზიკურად განაწილებული მონაცემთა რელაციური ბაზების სისტემა ობიექტ-როლური მოდელირების პრინციპების საფუძველზე და შესაბამისი გრაფულ-ანალიზური ინსტრუმენტების გამოყენებით [76,77].

შემუშავებულია აგებული სისტემის მომხმარებელთა ინტერფეისები, ინსტრუქციები, დანერგვისა და ექსპლუატაციის პროცესების ორგანიზაციული, ტექნიკური და იურიდიული ასპექტები [85,87].

28.1 ელექტრონული არჩევნების მართვის საინფორმაციო სისტემის აგების ამოცანა

ელექტრონული საარჩევნო სისტემის ძირითადი მიზანია [40]:

- საარჩევნო სიების სრულყოფა;
- საარჩევნო სიაში არსებული ყველა ამომრჩევლის შესახებ ამომწურავი ინფორმაციის მოგროვება და გამდიდრება;
- საარჩევნო პროცესის ავტომატიზაცია, რაც სამომავლოდ გაუიაფებს სახელმწიფოს არჩევნების ჩატარების ხარჯებს.

ნაშრომში ხორციელდება ელექტრონული საარჩევნო სისტემის ბიზნესპროცესების მართვის დაპროექტება და რეალიზაცია სერვის-ორიენტირებული არქიტექტურით.

მიზნის მისაღწევად განიხილება შემდეგი ძირითადი ამოცანები:

- არსებული თანამედროვე ელექტრონული საარჩევნო სისტემების ანალიზი და შესაბამისი ინფორმაციული ტექნოლოგიების კლასიფიკაცია, ობიექტ-, პროცეს- და სერვისორიენტირებული დაპროექტების პრინციპებით;

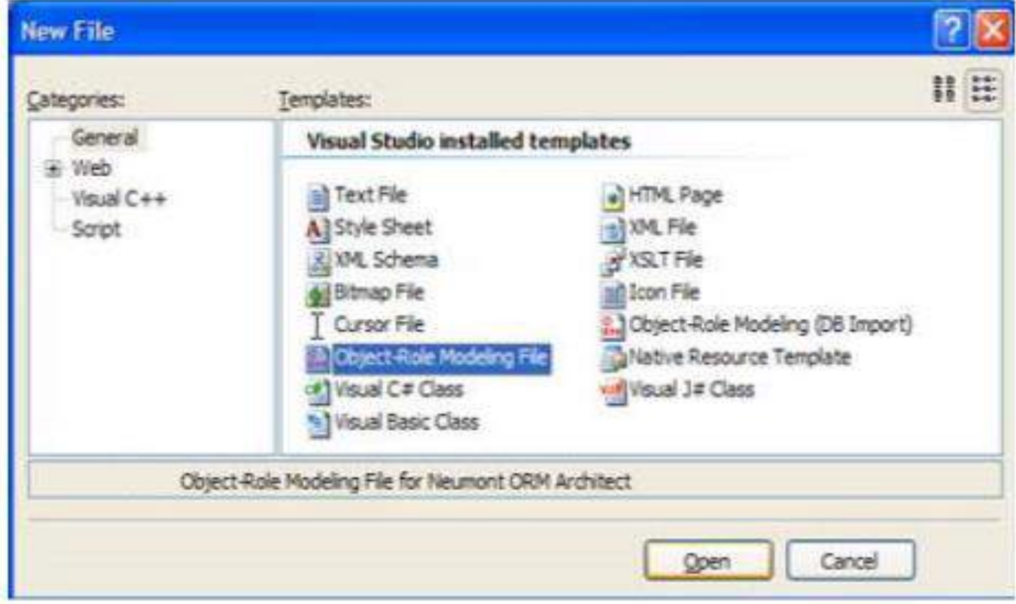
- მულტიმედიური მონაცემთა ბაზების დაპროექტება და აგება კლიენტ-სერვერული ტექნოლოგიის გამოყენებით SQL Server-ის ბაზაზე;

- სერვისორიენტირებული მონაცემთა განაწილებული ბაზების სტრუქტურების დასაპროექტებლად ობიექტროლური მოდელების (ORM) აგება და კვლევა რევერსიული CASE ტექნოლოგიების გამოყენებით;

- პროექტის შედეგების საფუძველზე ესპერიმენტული პროგრამული სისტემის რეალიზაცია .NET პლატფორმაზე, C#.NET, Natural ORM Architect და SQL Server პროგრამული პაკეტების გამოყენებით.

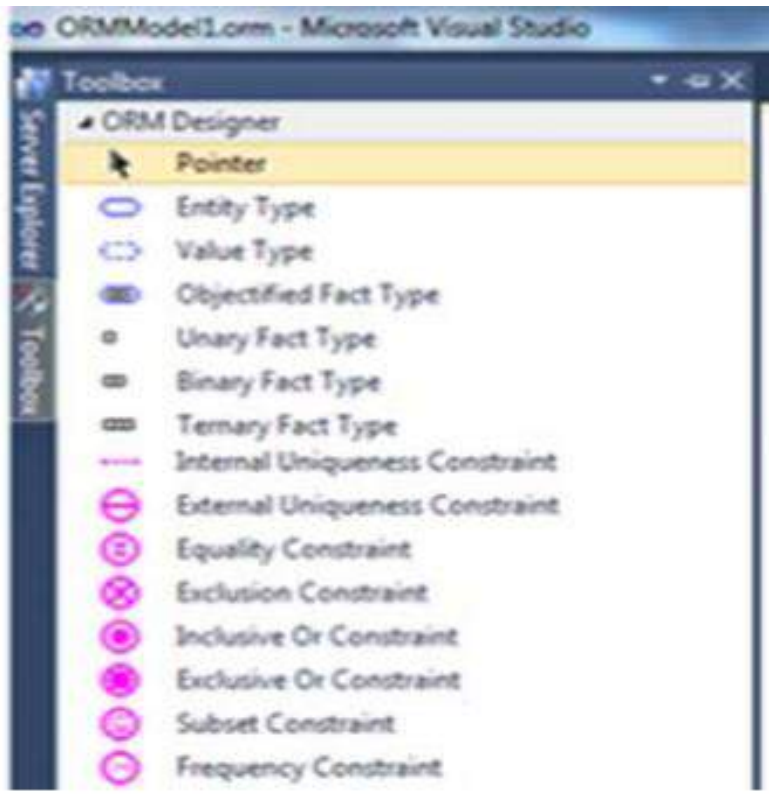
28.2 ელექტრონული არჩევნების სისტემის კონცეპტუალური მოდელის აგება ORM/ERM ტექნოლოგიით

მონაცემთა ბაზების ავტოამტიზებულ რეჟიმში დაპროექტება შესაძლებელია ობიექტროლური ORM (Object-Role Modeling) მოდელირების ტექნოლოგიის საშუალებით. Natural ORM Architect (NORMA) ინსტრუმენტი წარმოადგენს Microsoft Visual Studio.NET Framework-ის plugin-ს [76]. ORM-დიაგრამის შესაქმნელად საჭიროა General კატეგორიიდან Object-Role Modeling File template-ის ამორჩევა (ნახ.28.1) [77].



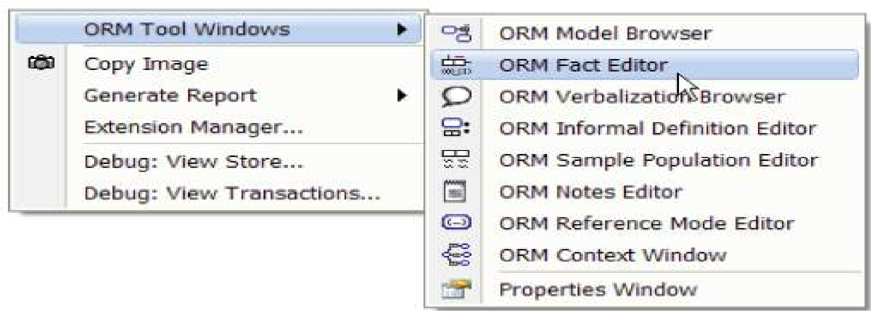
ნახ.28.1. ORM ფაილის გახსნა

ORM-დიაგრამის აგება ხდება Document Window ფანჯარაში, სადაც ხელთ გვაქვს დიაგრამის ასაგებად საჭირო ყველა ინსტრუმენტი: არსი (Entity), კავშირი (Connector) და შეზღუდვები (Constraints). ეს ნაჩვენებია 28.2 ნახაზზე.



ნახ.28.2. ORM-დოკუმენტის ფანჯარა

ფაქტების შესატანად საჭიროა FACT EDITOR-ის გააქტიურება (ნახ.28.3).



ნახ.28.3. ORM Fact Editor-ის ამორჩევა

თავდაპირველად ხდება საპრობლემო სფეროს (კვლევის ობიექტის) მოთხოვნილებათა ანალიზი, ტექნიკური დავალების განსაზღვრის მიზნით, საიდანაც ჩამოყალიბდება ფაქტები.

ფაქტი არის კატეგორიული ცნება, რომელშიც აისახება ობიექტის სემანტიკური (შინაარსობრივი) შედგენილობა და სტრუქტურა [77]. იგი ენის გრამატიკისა და ლოგიკური ალგებრის სიმბიოზია.

ფორმალური სახით ფაქტის აღწერა ხდება შემდეგი სქემით:

„ობიექტი_1“ პრედიკატი „ობიექტი_2“ (ან „მნიშვნელობა“)

ესაა ორადგილიანი პრედიკატის ჩაწერის მაგალითი. შესაძლოა 3,4 და ზოგადად n-ადგილიანი პრედიკატების გამოყენებაც. სწორედ ამ ელემენტარული ფაქტების საშუალებით განისაზღვრება ORM-მოდელი. FACT EDITOR-ის ფანჯარაში შეგვაქვს დანარჩენი ფაქტები:

f1: Voter has VoterName

f2: Voter has FingerPrint

f3: Voter voted for a Majoritary_Deputy

f4: Voter has Voter_ID

f5: Voter has Address

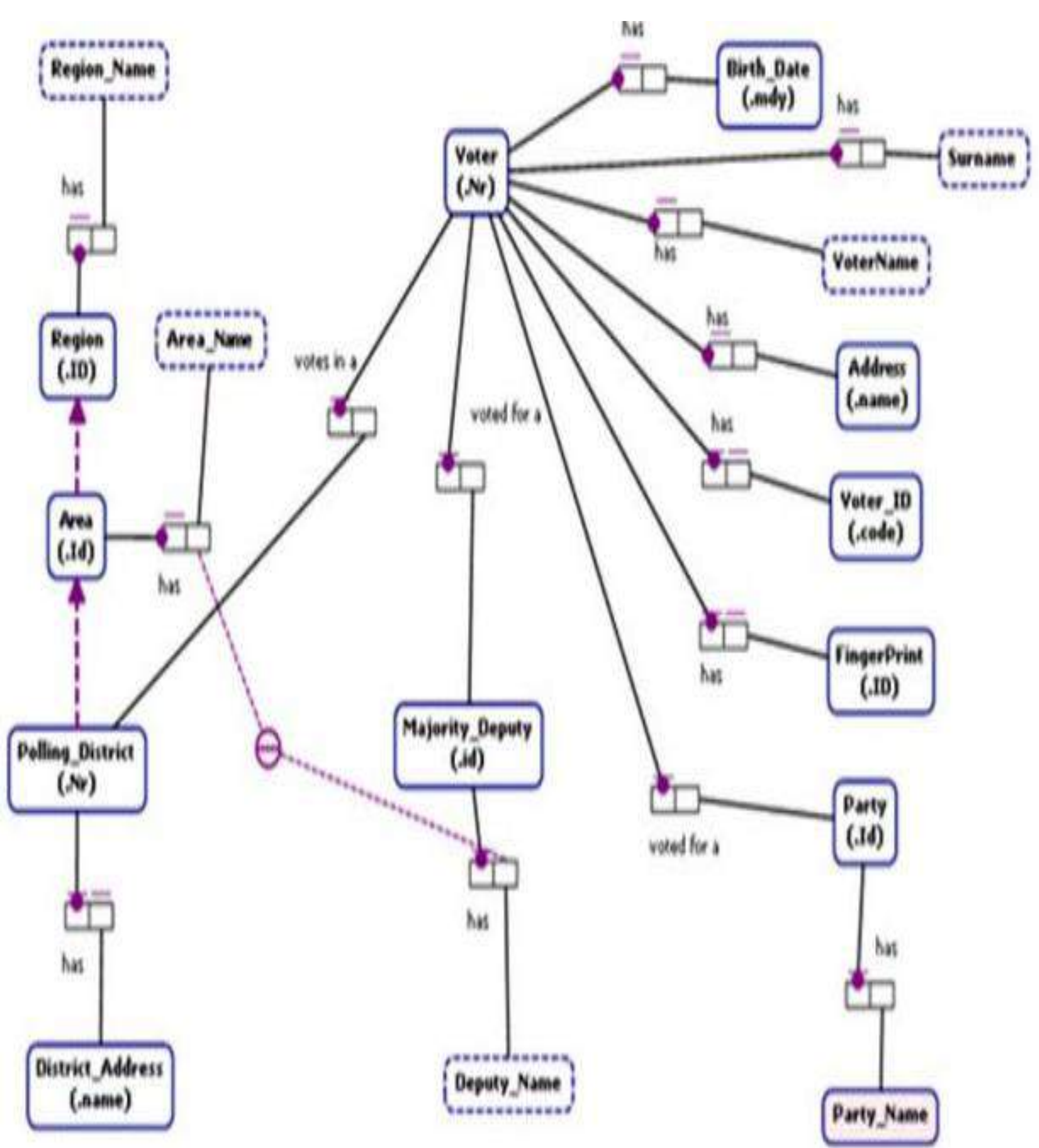
f6: Voter has Surname

f7: Majoritary_Deputy has Deputy_Name

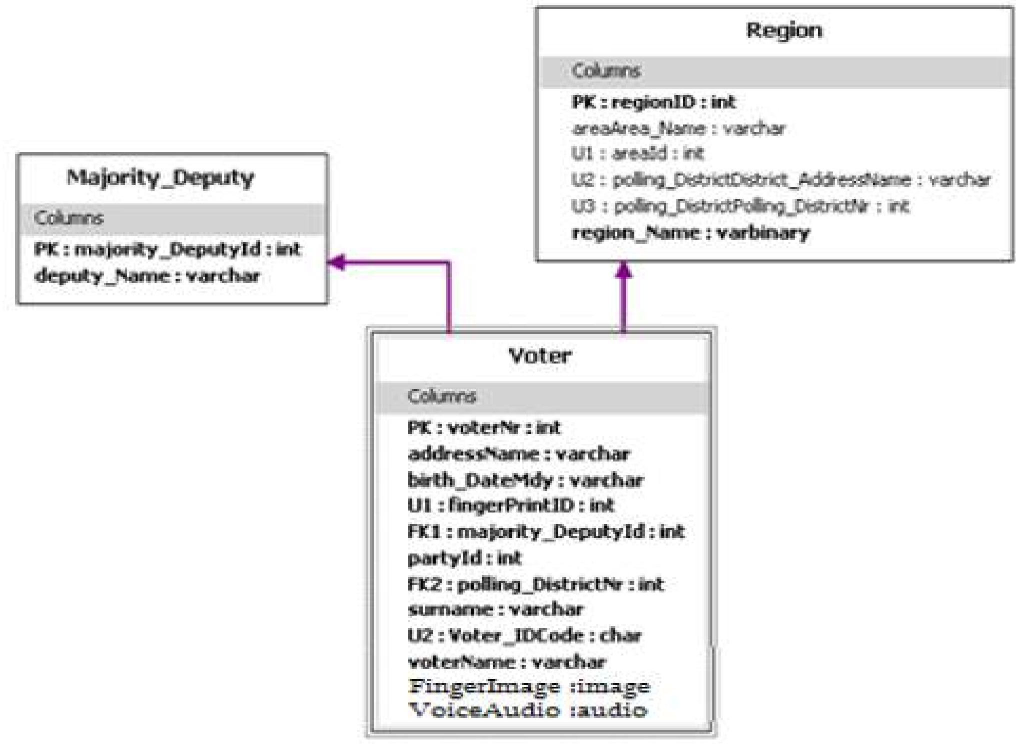
f8: Voter votes in a Polling_District . . . და ა.შ.

ზემომოყვანილი ფაქტების შეტანის შემდეგ ORM-დიაგრამა მიიღებს 28.4 ნახაზზე ნაჩვენებ სახეს.

აგებული დიაგრამიდან ავტომატიზებულ რეჟიმში ვახდენთ არსთა დამოკიდებულების მოდელის ER (Entities-Relationship Model) კონსტრუირებას Extension manager-ის საშუალებით და მიიღება ER-მოდელის დიაგრამა, რომელიც ნაჩვენებია 28.5 ნახაზზე.



ნახ.28.4. ORM-დიაგრამის ფრაგმენტი



ნახ.28.5. ავტომატიზებულ რეჟიმში მიღებული ER-მოდელის
ფრაგმენტი 3 ცხრილით

დიაგრამაზე გამოსახულია სამი არსი (Entities) და ორი ლოგიკური რელაციური კავშირი, რომელთა პროგრამული რეალიზაცია ხორციელდება ცხრილების (Tables) და ცხრილთაშორისი კავშირებით პირველადი (Primary key) და მეორეული (Foreign key) გასაღებების გამოყენებით.

ჩვენ მაგალითში ამ მიზნით გამოყენებულია:

– PK: majority_DeputyId,
– PK: regionId,
– PK: voterNr,
– FK1: majority_DeputyId,
– FK2:pooling_DistrictNr,
– U1:fingerPrintID,
– U2:voter_IDCode.

Natural ORM Architect პაკეტის საშუალებით, ასევე ავტომატიზებულ რეჟიმში, ვახდენთ მონაცემთა აღწერის ენის შესაბამისი DDL-კოდის გენერაციას.

Visual Studio.NET პაკეტის Solution Explorer-ის საშუალებით შესაძლებელია SQL Server მონაცემთა ბაზისათვის ჩვენ მიერ გენერირებული კოდის ნახვა.

მონაცემთა ბაზის ეს DDL-ფაილის ფრაგმენტი შემდეგნაირად გამოიყურება:

```
CREATE SCHEMA ORMModel1
GO
CREATE TABLE ORMModel1.Voter
( voterNr INTEGER NOT NULL,
  fingerPrintID INTEGER IDENTITY (1, 1) NOT NULL,
  Voter_IDCode NATIONAL CHARACTER(4000) NOT NULL,
  voterName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  surname NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  addressName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  partyId INTEGER IDENTITY (1, 1) NOT NULL,
 birth_DateMdy NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  majority_DeputyId INTEGER NOT NULL,
  polling_DistrictNr INTEGER NOT NULL,
  CONSTRAINT Voter_PK PRIMARY KEY(voterNr),
  CONSTRAINT Voter_UC1 UNIQUE(fingerPrintID),
  CONSTRAINT Voter_UC2 UNIQUE(Voter_IDCode)
 )
GO
CREATE TABLE ORMModel1.Majority_Deputy
( majority_DeputyId INTEGER IDENTITY (1, 1) NOT NULL,
  deputy_Name NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  CONSTRAINT Majority_Deputy_PK PRIMARY KEY(majority_DeputyId)
)
GO
CREATE TABLE ORMModel1.Region
(  regionID INTEGER IDENTITY (1, 1) NOT NULL,
   region_Name BINARY VARYING(MAX) NOT NULL,
   areaId INTEGER IDENTITY (1, 1),
   polling_DistrictDistrict_AddressName NATIONAL CHARACTER
                        VARYING(MAX),
   polling_DistrictPolling_DistrictNr INTEGER,
   areaArea_Name NATIONAL CHARACTER VARYING(MAX),
   CONSTRAINT Region_PK PRIMARY KEY(regionID),
   CONSTRAINT Region_Area_MandatoryGroup CHECK(areaId IS NOT
```

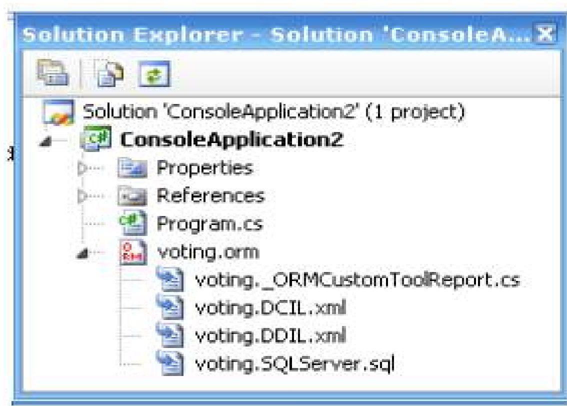
```
  NULL AND areaArea_Name IS NOT NULL OR
      polling_DistrictDistrict_AddressName IS NULL AND
    polling_DistrictPolling_DistrictNr IS NULL AND areaId IS NULL AND
      areaArea_Name IS NULL), CONSTRAINT
      Region_Polling_District_MandatoryGroup CHECK
    (polling_DistrictDistrict_AddressName IS NOT NULL AND
     polling_DistrictPolling_DistrictNr IS NOT  NULL OR
     polling_DistrictDistrict_AddressName IS NULL AND
     polling_DistrictPolling_DistrictNr IS NULL)
  )
GO
CREATE VIEW ORMModel1.Region_UC1 (areaId)
WITH SCHEMABINDING
AS
SELECT  areaId
FROM    ORMModel1.Region
WHERE areaId IS NOT NULL
GO
CREATE UNIQUE CLUSTERED INDEX Region_UC1Index ON
ORMModel1.Region_UC1(areaId)
GO
CREATE VIEW ORMModel1.Region_UC2
(polling_DistrictDistrict_AddressName)
WITH SCHEMABINDING
AS
SELECT polling_DistrictDistrict_AddressName
FROM   ORMModel1.Region
WHERE polling_DistrictDistrict_AddressName IS NOT NULL
GO
CREATE UNIQUE CLUSTERED INDEX Region_UC2Index ON
ORMModel1.Region_UC2 (polling_DistrictDistrict_AddressName)
GO CREATE VIEW ORMModel1.Region_UC3
(polling_DistrictPolling_DistrictNr)
WITH SCHEMABINDING
AS
SELECT polling_DistrictPolling_DistrictNr
FROM   ORMModel1.Region
```

```
WHERE polling_DistrictPolling_DistrictNr IS NOT NULL
GO
CREATE UNIQUE CLUSTERED INDEX Region_UC3Index ON
ORMModel1.Region_UC3 (polling_DistrictPolling_DistrictNr)
GO
ALTER TABLE ORMModel1.Voter ADD CONSTRAINT Voter_FK1
FOREIGN KEY (majority_DeputyId) REFERENCES
ORMModel1.Majority_Deputy(majority_DeputyId) ON
 DELETE NO ACTION ON UPDATE NO ACTION
GO
ALTER TABLE ORMModel1.Voter ADD CONSTRAINT Voter_FK2
FOREIGN KEY (polling_DistrictNr) REFERENCES
ORMModel1.Region(polling_DistrictPolling_DistrictNr)ON
DELETE NO ACTION ON UPDATE NO ACTION
GO
```

28.3. მონაცემთა ბაზის აგების პროგრამული რეალიზაციის ავტომატიზებული პროცედურა

Natural ORM Architect პროგრამული პაკეტის დახმარებით ავტომატიზებულ რეჟიმში მოვახდინეთ DDL კოდის გენერირება, რათა აგვეგო დასაპროექტებელი სისტემის რელაციური სქემა.

Solution Explorer-ში ნაჩვენებია კოდი, რომელიც დაგენერირდა SQL Server-ისათვის (ნახ.28.6). აქ ილუსტრირებულია ConsoleApplication2 პროექტის სტრუქტურა, რომლის voting.orm ფაილი რამდენიმე კომპონენტისგან (.cs, .xml და .sql ფაილები) შედგება [40].



ნახ.28.6. VisualStudio.NET გარემოში პროექტი ConsoleApplication2

პროგრამის ტექსტი, რომელიც ავტომატიზებულ რეჟიმში იქნა მიღებული და აღწერს ჩვენი კვლევის ობიექტის ანუ ელექტრონული საარჩევნო სისტემის მონაცემთა მოდელის შინაარსს, შემდეგი სახისაა:

ელ_საარჩევნო სისტემის ფაქტები ცხრ.28.6

Voter is an entity type.
Reference Scheme: Voter has Voter_Nr.
Reference Mode: .Nr.
Fact Types:
Voter has Voter_Nr.
Voter has VoterName.
Voter has Surname.
Voter has Address.
Voter has FingerPrint.
Voter voted for a Party.
Voter voted for a Majority_Deputy.
VoterName.
Voter has Surname.
Each Voter has exactly one Surname.
It is possible that more than one Voter has the same Surname.
Address is an entity type.
Reference Scheme: Address has Address_name.
Reference Mode: .name.

Fact Types:
Address has Address_name.
Voter has Address.
Voter has Address.
Each Voter has exactly one Address.
It is possible that more than one Voter has the same Address.
Surname is a value type.
Portable data type: Text: Variable Length.

Fact Types:
Voter has Surname.
FingerPrint is an entity type.
Reference Scheme: FingerPrint has FingerPrint_ID.
Reference Mode: .ID.

Fact Types:
FingerPrint has FingerPrint_ID.
Voter has FingerPrint.
Voter has FingerPrint.
Each Voter has exactly one FingerPrint.
For each FingerPrint, at most one Voter has that FingerPrint.
Voter voted for a Party.
Each Voter voted for a exactly one Party.
It is possible that more than one Voter voted for a the same Party.
Party is an entity type.
Reference Scheme: Party has Party_Id.
Reference Mode: .Id.

Fact Types:
Party has Party_Id.
Voter has Voter_ID.
Voter has Birth_Date.
Voter votes in a Polling_District.
VoterName is a value type.
Portable data type: Text: Variable Length.

Fact Types:
Voter has VoterName.
Voter has VoterName.
Each Voter has exactly one VoterName.
It is possible that more than one Voter has the same
Reference Scheme: Majority_Deputy has Majority_Deputy_id.
Reference Mode: .id.

Fact Types:
Majority_Deputy has Majority_Deputy_id.
Voter voted for a Majority_Deputy.
Majority_Deputy has Deputy_Name.
Voter voted for a Majority_Deputy.
Each Voter voted for a exactly one Majority_Deputy.
It is possible that more than one Voter voted for a the same Majority_Deputy.
Deputy_Name is a value type.
Portable data type: Text: Variable Length.

Fact Types:
Majority_Deputy has Deputy_Name.
Majority_Deputy has Deputy_Name.
Each Majority_Deputy has exactly one Deputy_Name.
It is possible that more than one Majority_Deputy has the same Deputy_Name.
Voter_ID is an entity type.
Reference Scheme: Voter_ID has Voter_ID_code.
Reference Mode: .code.

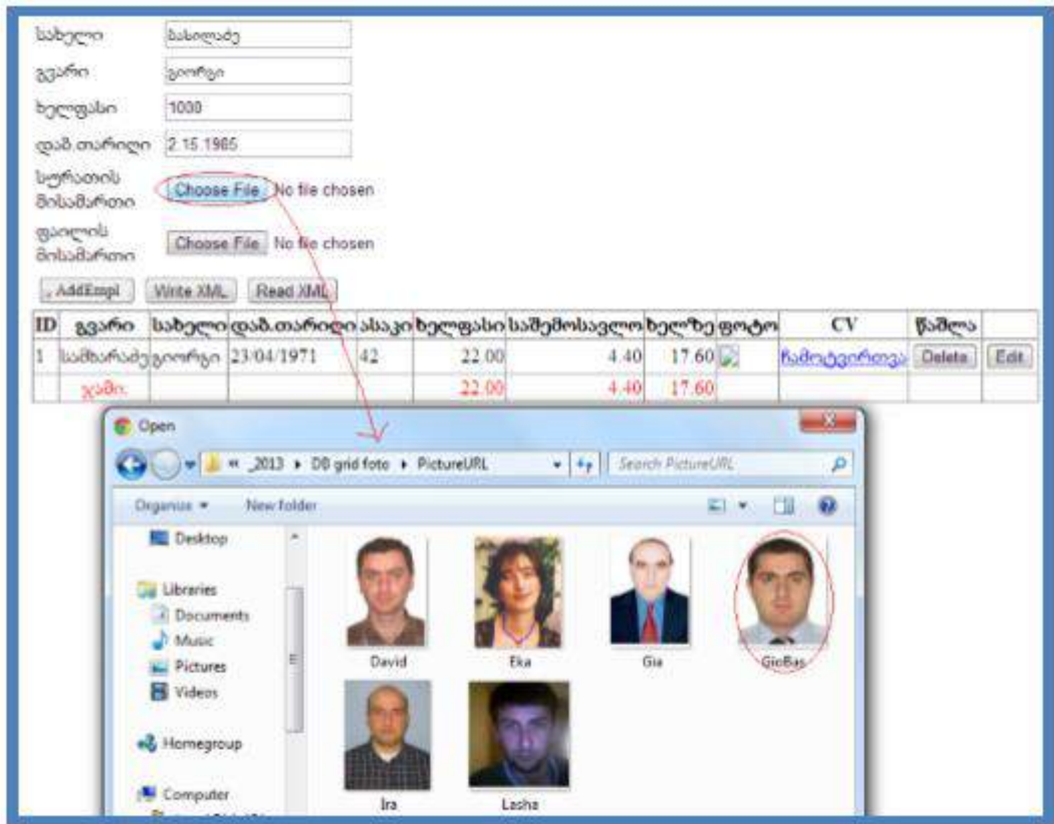
Fact Types:
Voter_ID has Voter_ID_code.
Voter has Voter_ID.
Voter has Voter_ID.
Each Voter has exactly one Voter_ID.
For each Voter_ID, at most one Voter has that Voter_ID.
Birth_Date is an entity type.
Reference Scheme: Birth_Date has Birth_Date_mdy.
Reference Mode: .mdy.

Fact Types:
Birth_Date has Birth_Date_mdy.
Voter has Birth_Date.
Voter has Birth_Date.
Each Voter has exactly one Birth_Date.
It is possible that more than one Voter has the same

Voter voted for a Party. Party has Party_Name. Party has Party_Name. **Each** Party has **exactly one** Party_Name. **It is possible that more than one** Party has **the same** Party_Name. Majority_Deputy **is an entity type.** Region has Region_Name. **Each** Area is an instance of Region. Region_Name **is a value type.** **Portable data type:** Raw Data: Variable Length. **Fact Types:** Region has Region_Name. Region has Region_Name. **Each** Region has **exactly one** Region_Name. **It is possible that more than one** Region has **the same** Region_Name. Area **is an entity type.** **Reference Scheme:** Area has Area_Id. **Reference Mode:** .Id. **Fact Types:** Area has Area_Id. Area has Area_Name. **Each** Polling_District is an instance of Area. **Each** Area is an instance of Region. Area_Name **is a value type.** **Portable data type:** Text: Variable Length. **Fact Types:** Area has Area_Name. Area has Area_Name. **Each** Area has **exactly one** Area_Name. **It is possible that more than one** Area has **the same** Area_Name. Polling_District **is an entity type.** **Reference Scheme:** Polling_District has Polling_District_Nr. **Reference Mode:** .Nr.	Birth_Date. Region **is an entity type.** **Reference Scheme:** Region has Region_ID. **Reference Mode:** .ID. **Fact Types:** Region has Region_ID. **Fact Types:** Polling_District has District_Address. **Each** Polling_District is an instance of Area. Polling_District has Polling_District_Nr. Voter votes in a Polling_District. District_Address **is an entity type.** **Reference Scheme:** District_Address has District_Address_name. **Reference Mode:** .name. **Fact Types:** District_Address has District_Address_name. Polling_District has District_Address. Polling_District has District_Address. **Each** Polling_District has **exactly one** District_Address. **For each** District_Address, **at most one** Polling_District has **that** District_Address. Voter votes in a Polling_District. **Each** Voter votes in a **exactly one** Polling_District. **It is possible that more than one** Voter votes in a **the same** Polling_District. **C** Area has Area_Name; Majority_Deputy **ontext:** has Deputy_Name. **In this context, each** Area_Name, Deputy_Name **combination is unique.** Model Error: Constraint 'ExternalUniquenessConstraint1' in model 'ORMModel1' requires a join path. **Each** Area is an instance of Region. **Each** Polling_District is an instance of Area.

28.4. სისტემის პროგრამული რეალიზაცია Visual Studio.NET Framework გარემოში

28.6 ნახაზზე ნაჩვენებია ინტერნეტ ბრაუზერში მომუშავე სისტემის ინტერფეისი - ამომრჩეველთა რეგისტრაცია. გრაფიკული და ტექსტური ინფორმაციის შესატანად გამოყენებულია შესაბამისი დიალოგური პროცედურები.



ნახ.28.6. ინტერფეისის ფრაგმენტი ფოტო ინფორმაციის მიერთებით

Write XML ღილაკით განახლებული ცხრილი ჩაიწერება XML-ფაილში (მონაცეთა ბაზაში). Read XML ღილაკი უზრუნველყოფს ამ ბაზის ხელახლად წაკითხვას. 28.7 ნახაზზე ნაჩვენებია XML ფაილის ლისტინგი.

DBGrid-foto - Microsoft Visual Studio

File Edit View Website Build Debug Team XML Data Tools Test

lectors.xml × Default.aspx Default.aspx.cs

```xml
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Lectors>
    <ID>1</ID>
    <FirstName>გიორგი</FirstName>
    <LastName>სამხარაძე</LastName>
    <Salary>2200</Salary>
    <BirthDate>1971-04-23T00:00:00+04:00</BirthDate>
    <ImagePath>~/PictureURL/Roman.bmp</ImagePath>
    <DocPath>~/DocURL/CV-Roman.docx</DocPath>
  </Lectors>
  <Lectors>
    <ID>31</ID>
    <FirstName>ბასილაძე</FirstName>
    <LastName>გიორგი</LastName>
    <Salary>1000</Salary>
    <BirthDate>1985-05-12T00:00:00+04:00</BirthDate>
    <ImagePath>~/PictureURL/GioBas.bmp</ImagePath>
    <DocPath>~/DocURL/CV-BasilazeGio.docx</DocPath>
  </Lectors>
  <Lectors>
    <ID>41</ID>
    <FirstName>გაბინაშვილი</FirstName>
    <LastName>ლაშა</LastName>
    <Salary>500</Salary>
    <BirthDate>1997-10-22T00:00:00+04:00</BirthDate>
    <ImagePath>~/PictureURL/Lasha.bmp</ImagePath>
    <DocPath>~/DocURL/CV-GabinLasha.docx</DocPath>
  </Lectors>
</NewDataSet>
```

ნახ.28.7. XML ფაილის შიგთავსის სტრუქტურა

28.5. მომხმარებელთა ინტერფეისების დამუშავება

დაპროექტებული და აგებულია ელექტრონული საარჩევნო სისტემის მხარდამჭერი პროგრამული უზრუნველყოფა, რომლის ჩაწერაც მოხდება საარჩევნო უბნებზე არსებულ რეგისტრატორთათვის განკუთვნილ კომპიუტერებზე.

ელექტრონული რეგისტრაციის ფორმა შექმნილია ახალი ტექნოლოგიების გამოყენებით, როგორიცაა Windows Presentation Foundation (WPF) და Metro Style App, რომელიც დღეისათვის ინოვაციას წარმოადგენს და გამოიყენება Microsoft Windows 8-ში [87]. ჩვენ შევეცადეთ, რომ პროგრამა ყოფილიყო შედარებით მარტივი და ადვილად სამართავი, რათა ნებისმიერ საარჩევნო უბნის რეგისტრატორს გაადვილებოდა მასთან მუშაობა და მომხდარიყო დროის მაქსიმალურად გამოყენება. ასევე გათვალისწინებულია მომხმარებელთა ინტერფეისების დამუშავება საქართველოს სომეხი, აზერბაიჯანელი, ოსი და აფხაზი ეროვნების წარმომადგენელთათვის [103,40].

აღნიშნული პროგრამული უზრუნველყოფის გამოყენება გვაძლევს გარანტიას სრულად გამოვრიცხოთ ისეთი ტერმინები და მისგან გამომდინარე შექმნილი უხერხულობები, როგორიც არის საარჩევნო სიების გაყალბება, მკვდარი სულები საარჩევნო სიებში, გამოტოვების შემთხვევები საარჩევნო სიებში და არჩევნების მიმდინარეობის პროცესში - „ კარუსელები “ (რაც გამოიხატება ერთი ამომრჩევლის ან ამომრჩეველთა ჯგუფის მიერ რამდენიმე საარჩევნო უბანზე საკუთარი ხმის დაფიქსირებაში).

პროგრამა არ იძლევა უფლებას, რომ ერთმა ადამიანმა ერთზე მეტ საარჩევნო უბანზე გაიაროს რეგისტრაცია და მისცეს ხმა. იმ შემთხვევაში, თუ ამომრჩეველს უკვე გავლილი აქვს რეგისტრაცია და შესაბამისად მისი კონსტიტუციური უფლება - მისცეს ხმა სასურველ კანდიდატს უკვე აღსრულებულია, ხელმეორედ ნებისმიერ საარჩევნო უბანზე მისვლა აღმოჩენილი იქნება პროგრამის მიერ და ეცნობება საუბნო საარჩევნო კომისიის რეგისტრანტ წევრს იმის შესახებ, თუ რა დროს და რომელ უბანზე გაიარა უკვე რეგისტრაცია ამა თუ იმ ამომრჩეველმა.

კომისიის წევრი ვალდებულია აღკვეთოს ხელმეორედ ხმის მიცემის ფაქტი და აცნობოს სამართალდამცავ ორგანოებს ზემოაღნიშნული შესაბამისი რეაგირებისათვის.

პროგრამული უზრუნველყოფისათვის გამოვიყენეთ უახლესი ტექნიკა და თანამედროვე ტექნოლოგიები: თითის ანაბეჭდის სკანერი, ელექტრონული ხელმოწერის პანელი, ხმის ჩამწერი და შემდეგ მისი ამომცნობი სისტემები და ფოტოაპარატი, აგრეთვე ბიომეტრული სურათების შედარების სისტემები.

ამ ყველაფრის ხარჯზე, თუ მონაცემები არ იქნება გატარებული მონაცემთა ბაზაში თქვენ არ მოგეცემათ არჩევნებში მონაწილეობის უფლება. პროგრამა ითვალისწინებს გარდა პირადი ნომრით ამომრჩევლის იდენტიფიცირებისა, ისეთი დამატებითი იდენტიფიკატორების შემოტანას და მათ რეალიზაციას, როგორიცაა ამომრჩევლის თითის ანაბეჭდი, ამომრჩევლის ხმა, ასევე ხელმოწერა და ბიომეტრული სურათი.

ახლა დეტალურად განვიხილოთ პროგრამის მუშაობის ძირითადი პრინციპი:

ცენტრალურ საარჩევნო კომისიას ექნება წინასწარ ფორმირებული მონაცემთა ბაზა, თუ რომელ საარჩევნო უბანზე რომელი კომისიის წევრი არის მიმაგრებული და რა ფუნქცია-მოვალეობები აქვს ნაკისრი.

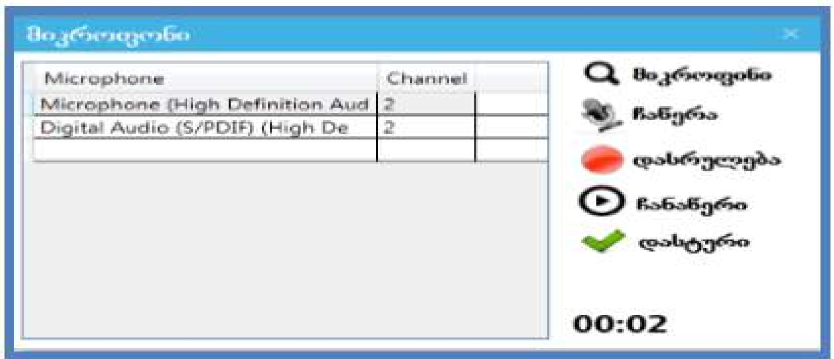
საარჩევნო უბნის გახსნისას ამომრჩევლთა მიღების დაწყებამდე, როდესაც რეგისტრატორი გახსნის პროგრამას, პირველ რიგში თვითონ გაივლის იდენტიფიკაციას და როგორც კი მისი აუთენტურობა დადგინდება, როგორც რეგისტრატორი, ამის შემდეგ მოხდება მოთხოვნა Access Code(AC)-ის, რომელიც იქნება უნიკალური და წინასწარ დალუქული კონვერტით ექნება მიღებული საუბნო კომისიის თავჯდომარეს.

უნდა შედგეს აღნიშნული კონვერტის გახსნის ოქმი, რომელზეც კომისიის თავმჯდომარესთან ერთად ხელმომწერები იქნებიან თავჯდომარის მოადგილე და საუბნო კომისიის წევრები. აღნიშნული AC კოდის შეტანის შემდეგ მიეცემათ უფლება, დაიწყონ ამომრჩეველთა მიღება და რეგისტრაცია. ყოველივე ეს აძლიერებს და ამყარებს უსაფრთოების ხარისხს და მეტ დამაჯერებლობას სძენს ელეტრონულ საარჩევნო სისტემას.

ამომრჩევლის უბანზე მისვლისას ხდება მისთვის ბიომეტრული სურათის გადაღება ვებ-კამერის მეშვებით, რეგისტრატორს შეუძლია ერთი ან რამდენიმე სურათის გადაღება და არჩევანის მოხდენა. ვებკამერა იმართება პროგრამიდან და აკეთებს დროებით ჩანაწერებს კომპიუტერში, სადაც ინახება მოქალაქის სურათი თავისივე უნიკალური დასახელებით, შერჩეული სურათის დაშლა ხდება ბიტებად და მონაცემთა ბაზაში ჩაწერა, ხოლო დანარჩენი სურათები ავტომატურად იშლება მყარი დისკიდან. კამერის სამართავად პროგრამული უზრუნველყოფა იყენებს კომპიუტერში არსებულ დრაივერს, რომელსაც პოულობს და ავტომატურად აყენებს (ნახ.28.8).

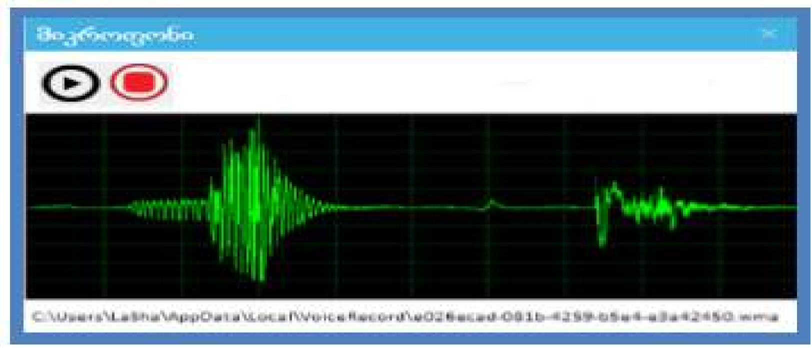
ნახ.28.8. მოქალაქის სურათი თავისივე უნიკალური დასახელებით

ხმის ჩასაწერად და მიკროფონზე წვდომის განსახორციელებლად გამოვიყენეთ ყველასათვის კარგად ცნობილი და გამოცდილი ბიბლიოთეკა, როგორიცაა Naudio. მისი მეშვეობით ხდება წვდომა მიკროფონზე და ძიება ყველა არსებული მიკროფონის, რომელიც მიერთებულია კომპიუტერს (ნახ.28.9) [40,87].



ნახ.28.9. კომპიუტერთან მიერთებული მიკროფონების ნუსხა

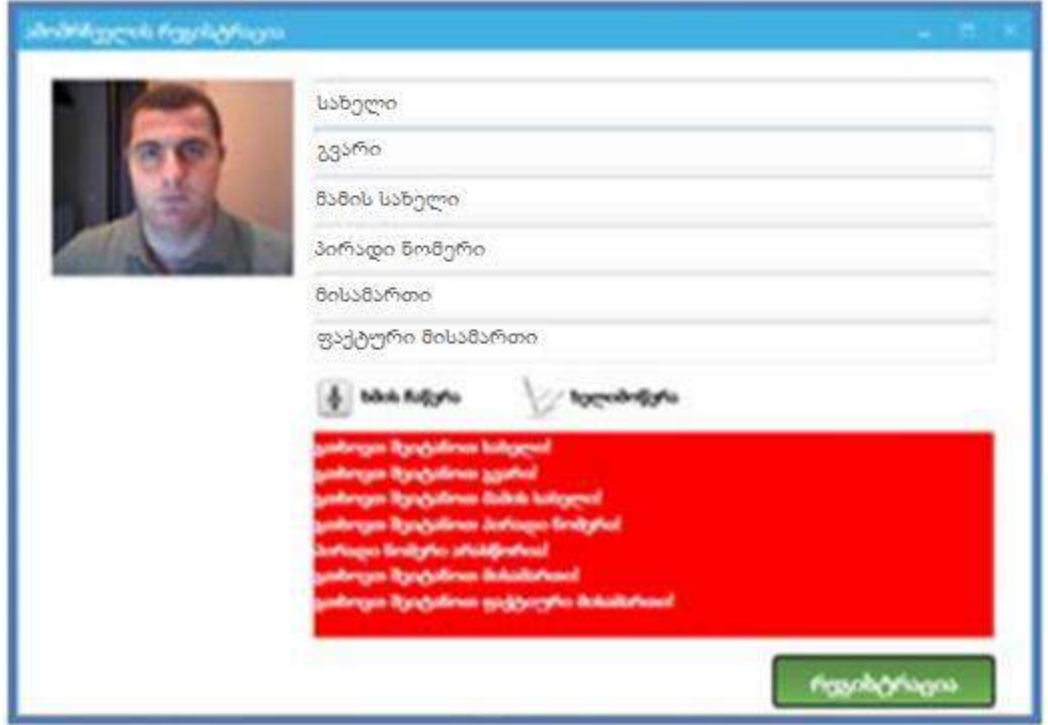
რეგისტრატორმა უნდა აირჩიოს მიკროფონი და ჩაიწეროს ხმა, რომელიც ასევე დროებით ჩაწერას ახდენს კომპიუტერის მეხსიერებაში. ჩაწერის შემდეგ შესაძლებელია ჩაწერილი ხმის მოსმენა და მისი გრაფიკულად ნახვა (ნახ.28.10).



ნახ.28.10. ხმის გრაფიკული გამოსახულება

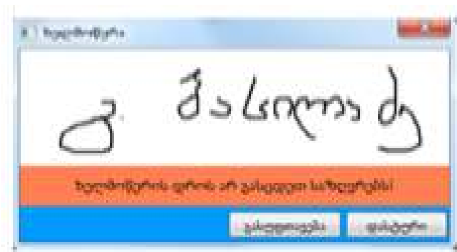
პროგრამული უზრუნველყოფა შეიცავს აგრეთვე მთელ რიგ ვალიდაციებს, რომლებიც არ მისცემს რეგისტრატორს რაიმე სახის შეცდომის დაშვების უფლებას ამომრჩეველთა რეგისტრაციის დროს. კერძოდ, სახელის, გვარის, მამის სახელის, მისამართის და ფაქტური მისამართის შეტანა და რეგისტრაცია მოხდება წინასწარ განსაზღვრული ფონტით-Sylfaen. რაც შეეხება პირად ნომერს, იგი აუცილებლად იქნება 11 ნიშნა რიცხვითი მონაცემი.

აღნიშნული შეცდომების დაშვების დროს მივიღებთ შემდეგი სახის გამაფრთხილებელ დიალოგურ ფანჯარას (ნახ.28.11).



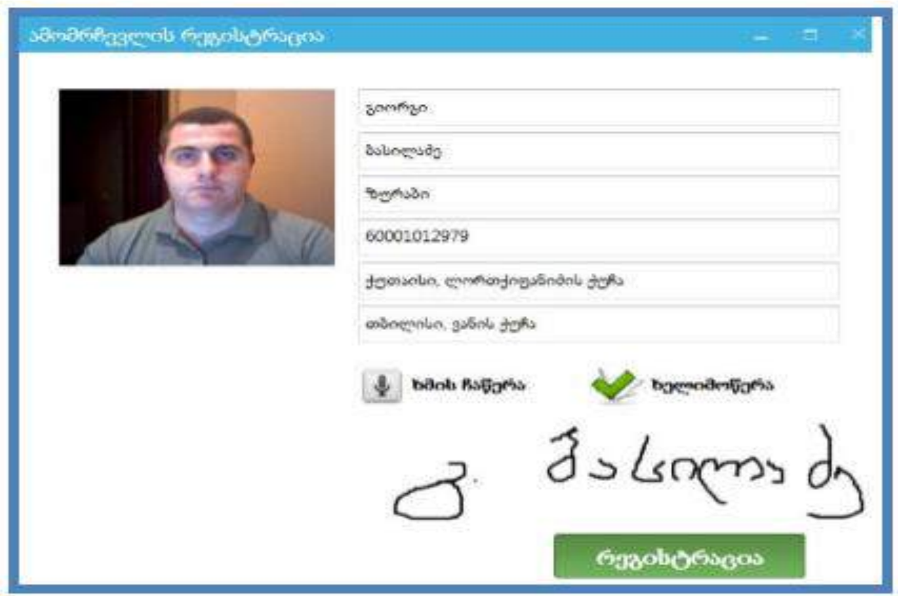
ნახ.28.11. გამაფრთხილებელი ფანჯარა

პროგრამა გამორიცხავს ამომრჩევლის შესახებ არასრული ინფორმაციის ბაზაში ჩანაწერის გაკეთებას, რაც თავის მხრივ ხელს უშლის არჩევნებზე დარეგისტრირებული ამომრჩეველთა რიცხვის ხელოვნურ ზრდას. პროგრამული უზრუნველყოფა ითვალისწინებს აგრეთვე ელექტრონული ხელმოწერის შეტანა-შენახვის მოდულს (ნახ.28.12) და თითის ანაბეჭდის შეტანა-შენახვა იდენტიფიცირებას.



ნახ.28.12. ელექტრონული ხელმოწერის შეტანა

პროექტის ფარგლებში შემუშავებულია მომხმარებელთა ინსტრუქციებიც [40]. მაგალითად, 28.13 ნახაზი გვიჩვენებს რეგისტრაციის ფორმის შევსებულ, საბოლოო ვარიანტს.



ნახ.28.13. რეგისტრაციის ფორმა

28.6. კომუნიკაცია: WCF ტექნოლოგია

ბიზნესპროცესების შესრულებისას ერთ-ერთი მნიშვნელოვანი ასპექტია ურთიერთობა (კომუნიკაცია) აპლიკაციებს, კლიენტებსა და სერვერებს, აგრეთვე სამუშაო პროცესებსა და ჰოსტდანართებს შორის.

აპლიკაციის მაგალითის სახით განიხილება პროექტის აგება საარჩევნო სისტემისათვის, რომელშიც მოთხოვნილი ინფორმაცია (მაგალითად, ამომრჩევლის ან დეპუტატის შესახებ) გადაიცემა ფილიალებს (საარჩევნო სისტემის იერარქიული რგოლები) შორის.

მთავარი ქმედებები, რომლებიც კომუნიკაციისათვის გამოიყენება, არის Send და Receive ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply). ეს ქმედებები გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და მონიტორინგისათვის.

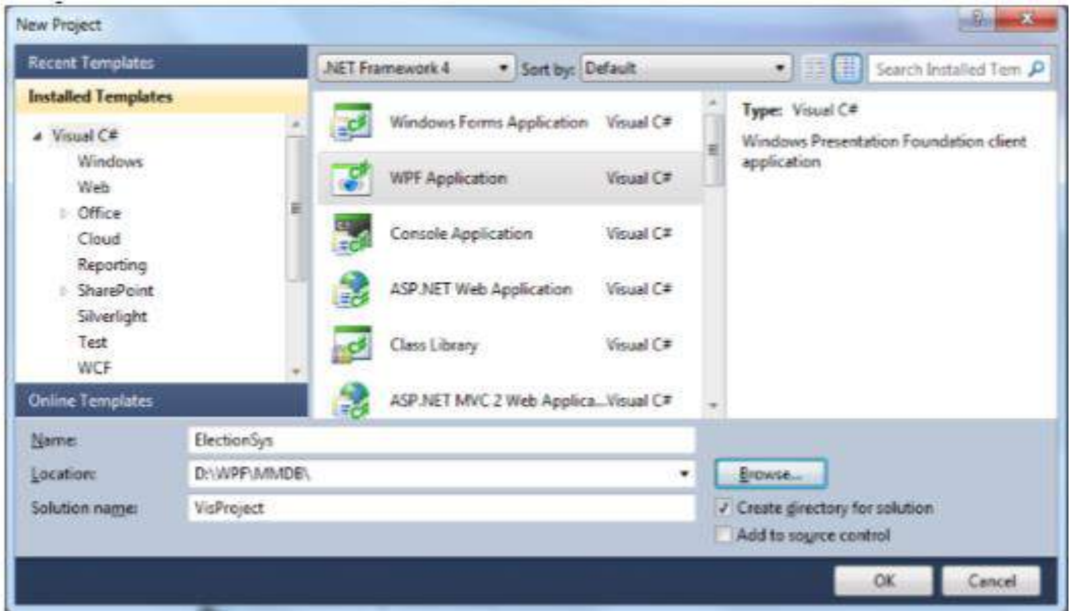
ავაგოთ მარტივი WPF-აპლიკაცია (Windows Presentation Foundation), რომელიც გამოიყენებს კომუნიკაციას სხვადასხვა აპლიკაციის ბიზნესპროცესებს შორის.

➢ **WPF პროექტის შექმნა.**

1. ახალი პროექტის შექმნა WPF აპლიკაციის სახით იწყება პროექტის სახელის ElectionSys და Solution-ის დასახელების VisProject შერჩევით (ნახ.28.14).

2. Solution Explorer-ში ElectionSys პროექტზე მაუსის მარჯვენა ღილაკით ავირჩიოთ Add Reference და .NET ცხრილიდან დავამატოთ შემდეგი კავშირები:

- System.Activities
- System.Configuration
- System.ServiceModel
- System.ServiceModel.Activities



ნახ.28.14. WPF აპლიკაციის შექმნა

3. Solution Explorer-ში გენერირდება ვინდოუსის ფაილი სახელით Window1.xaml, რომელსაც ვცვალით Election.xaml - ით.

App.xaml ფაილი განსაზღვრავს ვინდოუსის startup-ს. ესაა ახლა Windows1 ფაილი, რომელიც უნდა შევცვალოთ ასევე: StartupUri=" Election.xaml".

➢ **ახალი config-ფაილის და Class-ების შექმნა**

რამდენიმე აპლიკაციის კონფიგურირების მიზნით (საარჩევნო ფილიალების რაოდენობის შესაბამისად) შეიქმნება App.config ფაილების დუბლები, რომლებშიც მოთავსებულ იქნება შესაბამისი ფილიალის ფიზიკური მონაცემები. ElectionSys– პროექტის Solution Explorer–დან ვირჩევთ Add New Item ➤. დიალოგურ ფანჯარაში General ჯგუფში ვირჩევთ Application Configuration File. ფაილის სახელი ავტომატურად

არის App.config(). კონფიგურაციის ფაილში შევიტანთ საჭირო მონაცემებს 28,1 ლისტინგის მსგავსად.

```
<!— ლისტინგი_28.1 ———— –>
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Branch Name" value="Regioni GURIA"/>
    <add key="ID" value="{43E6DADD-4751-4056-8BB7-
                        7459B5C361AB}"/>
    <add key="Address" value="8000"/>
    <add key="Request Address" value="8730"/>
  </appSettings>
</configuration>
```

AppSettings სექციას აქვს მნიშვნელობები ფილიალის სახელი, ID (უნიკალური იდენტიფიკატორი) და მისამართი (პორტის ნომერი, რომელსაც აპლიკაცია იყენებს). მოთხოვნის მისამართი განსაზღვრავს პორტის ნომერს, საითაც იქნება მოთხოვნები გაგზავნილი. შესაძლებელია სხვა პორტების გამოყენებაც, თუ ეს აუცილებელი იქნება.

შემდეგ საჭიროა შეიქმნას კლასი, რომელიც განსაზღვრავს შეტყობინებებს აპლიკაციებს შორის. Solution Explorer-იდან Add -> Class და ჩავწეროთ კლასის სახელი Election.cs. ამ ფაილში დავამატოთ სახელსივრცეები (namespaces):

```
using System.Runtime.Serialization;
using System.ServiceModel;
```

Election.cs ფაილში განვსაზღვროთ სამი კლასი:

- Branch: განსაზღვრავს მონაცემებს საარჩევნო სისტემის ფილიალის მდებარეობის შესახებ;
- ElectionRequest: განსაზღვრავს ფილიალის მოთხოვნას;
- ElectionResponse: განსაზღვრავს პასუხს მოთხოვნის შესაბამის ფილიალისათვის. 28.2 ლისტინგში მოცემულია შესაბამისი კოდი:

```
//--- ლისტინგი_28.2 --- Election.cs ------
using System;
using System.Runtime.Serialization;
using System.ServiceModel;
namespace ElectionSys
{ /*********************************************/
  // ფილიალის მონაცემთა სტრუქტურის განსაზღვრა
```

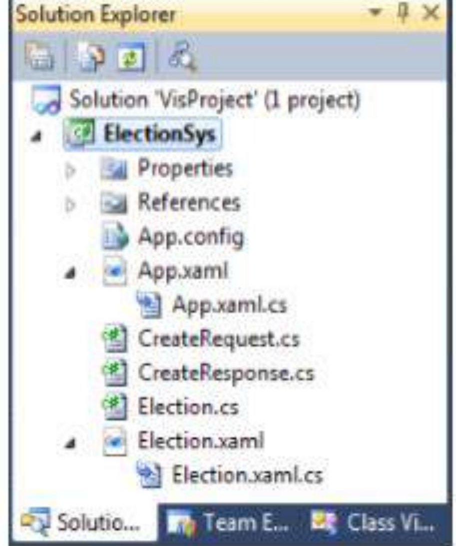
```
 /*********************************************/
public class Branch
 {    public String BranchName { get; set; }
      public String Address { get; set; }
      public Guid BranchID { get; set; }
      #region Constructors
      public Branch()  {  }
      public Branch(String name, String address)
        {
           BranchName = name;
           Address = address;
           BranchID = Guid.NewGuid();
       }
      public Branch(String name, String address, Guid id)
       {
          BranchName = name;
          Address = address;
           BranchID = id;
       }
     public Branch(String name, String address, String id)
      {
          BranchName = name;
        Address = address;
        BranchID = new Guid(id);
     }
     #endregion Constructors
   }
/****************************************************/
// მოთხოვნის შეტყობინების  განსაზღვრა, ElectionRequest
 /****************************************************/
[MessageContract(IsWrapped = false)]
public class ElectionRequest
 {
   private String _Region;
   private String _ArealName;
   private String _MajorDeputy;
   private Guid _RequestID;
```

```
   private Branch _Requester;
   private Guid _InstanceID;
   #region Constructors
   public ElectionRequest() {  }
public ElectionRequest(String arealname, String majordeputy,
        String region, Branch requester)
 {
   _ArealName = arealname;
   _MajorDeputy = majordeputy;
   _Region = region;
   _Requester = requester;
   _RequestID = Guid.NewGuid();
 }
public ElectionRequest(String arealname, String majordeputy,
         String region, Branch requester, Guid id)
     {   _ArealName = arealname;
         _MajorDeputy = majordeputy;
         _Region = region;
         _Requester = requester;
         _RequestID = id;
        }
      #endregion Constructors
      #region Public Properties
     [MessageBodyMember]
       public String Title
         {  get { return _ArealName }
            set { _ArealName = value; }
         }
      [MessageBodyMember]
       public String ISBN
        {  get { return _Region; }
            set { _Region = value; }
        }
      [MessageBodyMember]
        public String Author
         {  get { return _MajorDeputy; }
             set { _MajorDeputy = value; }
```

```
        }
       [MessageBodyMember]
        public Guid RequestID
          {  get { return _RequestID; }
             set { _RequestID = value; }
          }
        [MessageBodyMember]
          public Branch Requester
        {   get { return _Requester; }
        set { _Requester = value; }
      }
      [MessageBodyMember]
       public Guid InstanceID
            {  get { return _InstanceID; }
               set { _InstanceID = value; }
             }
         #endregion Public Properties
   }
/****************************************************/
// მოთხოვნის შეტყობინების განსაზღვრა:ElectionResponse
/****************************************************/
 [MessageContract(IsWrapped = false)]
 public class ElectionResponse
   {
     private bool _Reserved;
     private Branch _Provider;
     private Guid _RequestID;
     #region Constructors
     public ElectionResponse() {  }
     public ElectionResponse(ElectionRequest request, bool reserved,
                          Branch provider)
      {  _RequestID = request.RequestID;
         _Reserved = reserved;
         _Provider = provider;
       }
      #endregion Constructors
      #region Public Properties
```

```
    [MessageBodyMember]
     public bool Reserved
      {   get { return _Reserved; }
          set { _Reserved = value; }
      }
      [MessageBodyMember]
       public Branch Provider
         {  get { return _Provider; }
            set { _Provider = value; }
         }
      [MessageBodyMember]
       public Guid RequestID
         {   get { return _RequestID; }
             set { _RequestID = value; }
         }
      #endregion Public Properties
   }
/****************************************************/
// სერვისის კონტრექტის განსაზღვრა, IElectionSys
// რომელიც ორი მეთოდისგან შედგება: RequestElinfo() და
// RespondToRequest()
/****************************************************/
[ServiceContract]
public interface IElectionSys
 {
    [OperationContract(IsOneWay = true)]
    void RequestElinfo(ElectionRequest request);

    [OperationContract(IsOneWay = true)]
   void              RespondToRequest(ElectionResponse
response);
  }
 }
```

შედეგად Solution Explorer-ს ექნება ასეთი სახე (ნახ.28,15).



ნახ.28.15

➢ Window Form –ის განსაზღვრა

გავხსნათ Election.xaml ფაილი და ავირჩიოთ XAML tab. ჩავანაცვლოთ კოდი შემდეგი 28.3 ლისტინგის ტექსტით:

```
<!-- ლისტინგი 28.3 -- Election.xaml --- -->
<Window x:Class="ElectionSys.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml
        /presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ელ-საარჩევნო სისტემა" Height="480" Width="650"
    Loaded="Window_Loaded" Unloaded="Window_Unloaded">
 <Grid>
    <Label Height="40" HorizontalAlignment="Left"
       Margin="12,0,0,0" Name="lblBranch" FontSize="22"
       VerticalAlignment="Top" Width="276"
       FontStretch="Expanded">ოლქი: ოზურგეთი</Label>
    <ListView x:Name="requestList" Margin="12,42,12,5"
       Height="150" VerticalAlignment="Top"
       ItemsSource="{Binding}"
       SelectionChanged="requestList_SelectionChanged">
    <ListView.View>
     <GridView>
       <GridViewColumn Header="Request List" Width="610">
         <GridViewColumn.CellTemplate>
           <DataTemplate>
             <StackPanel Orientation="Horizontal">
               <TextBlock Text="{Binding
                    Requester.BranchName}" Width="100"/>
```

```
            <TextBlock Text="{Binding MajorDeputy}"
                  Width="95"/>
            <TextBlock Text="{Binding ArealName}"
                  Width="180"/>
            <TextBlock Text="{Binding Region}"
                  Width="90"/>
            <Button Content="Reserve" Tag="{Binding
              InstanceID}" Click="Reserve" Width="65"/>
            <Button Content="Cancel" Tag="{Binding
              InstanceID}" Click="Cancel" Width="60"/>
           </StackPanel>
          </DataTemplate>
         </GridViewColumn.CellTemplate>
        </GridViewColumn>
       </GridView>
      </ListView.View>
     </ListView>
   <Label Height="30" Margin="15,0,0,210" Name="label5"
         VerticalAlignment="Bottom"
         HorizontalAlignment="Left" Width="148"
          HorizontalContentAlignment="Right">მაჟორიტარი დეპუტატი:</Label>
  <Label Height="30" Margin="34,0,479,180" Name="label2"
           VerticalAlignment="Bottom"
           HorizontalContentAlignment= "Right">საარჩევნო ოლქი:</Label>
  <Label Height="30" Margin="45,25,0,150" Name="label3"
            VerticalAlignment="Bottom"
            HorizontalAlignment="Left" Width="60"
            HorizontalContentAlignment="Right">რეგიონი:</Label>
  <TextBox Height="25" Margin="169,0,0,210"
            Name="txtMajorDeputy"
            VerticalAlignment="Bottom"
            HorizontalAlignment="Left" Width="200" />
  <TextBox Height="25" Margin="0,0,161,180"
            Name="txtArealName"
            VerticalAlignment="Bottom"
            HorizontalAlignment="Right" Width="300" />
```

```
        <TextBox Height="25" Margin="168,0,0,150"
                   Name="txtRegion"
                   VerticalAlignment="Bottom"
                   HorizontalAlignment="Left" Width="100" />
        <Button Height="23" Margin="497,0,0,150"
                   Name="btnRequest"
                   VerticalAlignment="Bottom"
                   HorizontalAlignment="Left" Width="98"
               Click="btnRequest_Click">Send Request</Button>
        <Label Height="27" HorizontalAlignment="Left"
               Margin="15,0,0,137" Name="label4"
               VerticalAlignment="Bottom"
               Width="76">Event Log</Label>
        <ListBox Margin="12,0,12,12" Name="lstEvents"
               Height="130" VerticalAlignment="Bottom"
               FontStretch="Condensed" FontSize="10" />
    </Grid>
</Window>
```

<!– – ლისტინგი 28.3.-ის დასასრული -----

შემდეგ ავირჩიოთ Design Tab-ს. ფორმას უნდა ჰქონდეს შემდეგი სახე (ნახ.28.16). ფორმის ზედა ნაწილში მოთხოვნების სია (RequestList) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად რეგიონული ცენტრში გამოიყენება ველები ფორმის შუაში. აქ მიეთითება მაგალითად, მაჟორიტარი დეპუტატის გვარი_ს., საარჩევნო_ოლქი და რეგიონი, შემდეგ გაგზავნის ღილაკი „მოთხოვნის გაგზავნა“ (Send Request). „მოვლენათა რეგისტრაცია“ (Event Log) ქვედა მარცხენა კუთხეში ასახავს ბიზნესპროცესების შეტყობინებებს.

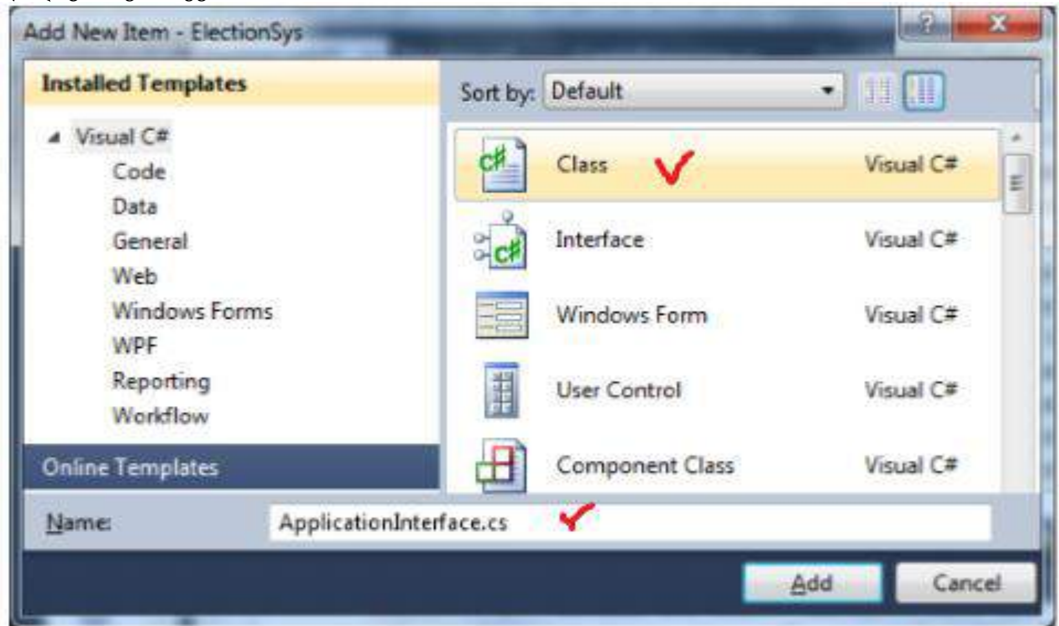
ნახ.28.16. ინტერფეისი „საარჩევნო ოლქი“

➢ ტექსტის ჩამწერის რეალიზაცია

WriteLine ქმედებისათვის, რომელიც გამოიყენება ტექსტის გამოსატანად ეკრანზე, არ დაგვიყენებია თვისება TextWriter. კონსოლის რეჟიმში ტექსტი ავტომატურად გამოიტანება, ფორმაზე გამოსატანად კი უნდა გავეცნოთ TextWriter კლასს.

➢ აპლიკაციის სტატიკური მიმთითებლის უზრუნველყოფა

თავიდან უნდა შეიქმნას სტატიკური კლასი, რომელიც უზრუნველყოფს აპლიკაციის ფანჯარასთან წვდომას. Solution Explorer-ში ElectionSys–ზე მაუსის მარჯვენა ღილაკით ვირჩევთ Add -> Class (ნახ.28.17).



ნახ.28.17. Class-ის არჩევა

კლასის სახელია ApplicationInterface.cs, რომლის პროგრამული რეალიზაცია მოცემულია 28.4 ლისტინგში.

```
// ------- ლისტინგი_28.3 ---- ApplicationInterface.cs --------
using System;
using System.Windows.Controls;
using System.Activities;
namespace ElectionSys
{
  public static class ApplicationInterface
```

```
 {
   public static MainWindow _app { get; set; }
   public static void AddEvent(String status)
    {
       if (_app != null)
        {
        new ListBoxTextWriter(_app.GetEventListBox())
               .WriteLine(status);
      }
    }
public static void RequestElinfo(ElectionRequest request)
 {
    if (_app != null)
      _app.RequestElinfo(request);
 }
 public static void RespondToRequest(ElectionResponse response)
   {
     if (_app != null)
      _app.RespondToRequest(response);
   }
 public static void NewRequest(ElectionRequest request)
    {
     if (_app != null)        _app.AddNewRequest(request);
    }
   }
 }
```

ApplicationInterface კლასს აქვს სტატიკური მიმთითებელი (_app) აპლიკაციის ფანჯარაზე (MainWindow კლასი). სტატიკური AddEvent() მეთოდი ქმნის ListBoxTextWriter კლასის ეგზემპლარს, რომელიც შემდგომში იქნება რეალიზებული და იძახებს მის WriteLine() მეთოდს.

გავხსნათ Election.xaml.cs ფაილი და დავამატოთ შემდეგი სახელსივრცეები:

```
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
using System.ServiceModel.Description;
using System.ServiceModel.Channels;
using System.Activities;
```

```
using System.Xml.Linq;
using System.Configuration;
```

დავამატოთ MainWindow()-ში კონსტრუქტორის შემდეგი კოდი: ApplicationInterface._app = this;

აქ this აინიციალიზებს _app მიმართვას ApplicationInterface კლასში. ვინაიდან იგი სტატიკური კლასია, მასში იქნება მხოლოდ ერთი ეგზემპლარი, რომელსაც ექნება მიმართვა MainWindow კლასზე. დავამატოთ შემდეგი მეთოდები Election.xaml.cs ფაილში.

```
public ListBox GetEventListBox()
 {
    return this.lstEvents;
 }
private void AddEvent(string szText)
  {
    lstEvents.Items.Add(szText);
  }
```

GetEventListBox() მეთოდი აბრუნებს უკან მიმთითებელს ListBox–ის ფაქტური კონტროლისათვის, რომელმაც უნდა ასახოს ეს მოვლენები. ამ მეთოდს იყენებს ApplicationInterface კლასი. AddEvent () მეთოდს იყენებს აპლიკაცია მაშინ, როცა მას სჭირდება მოვლენის დამატება.

- **ListBoxTextWriter–ის რეალიზაია**

დავამატოთ პროექტს კლასი ListBoxTextWriter.cs, რომლის რეალიზაცია 28.5 ლისტინგშია მოცემული.

```
// – – ლისტინგი_28.5 --- ListBoxTextWriter.cs ------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Windows.Controls;
namespace ElectionSys
 {
   public class ListBoxTextWriter : TextWriter
    {
      const string textClosed = "This TextWriter must be opened before use";
```

```
    private Encoding _encoding;
    private bool _isOpen = false;
    private ListBox _listBox;
    public ListBoxTextWriter()
    {
       // Get the static list box
       _listBox = ApplicationInterface._app.GetEventListBox();
      if (_listBox != null)
         _isOpen = true;
    }
   public ListBoxTextWriter(ListBox listBox)
      {
        this._listBox = listBox;
        this._isOpen = true;
      }
     public override Encoding Encoding
     {
     get
     {
      if (_encoding == null)
        {
         _encoding = new UnicodeEncoding(false, false);
        }
      return _encoding;
    }
  }
public override void Close()
  {
    this.Dispose(true);
  }
protected override void Dispose(bool disposing)
   {
      this._isOpen = false;
      base.Dispose(disposing);
    }
  public override void Write(char value)
    {
```

```
      if(!this._Open)
     throw new ApplicationException(textClosed); ;

    this._listBox.Dispatcher.BeginInvoke
       (new Action(() =>
         this._listBox.Items.Add(value.ToString())));
   }
  public override void Write(string value)
   {
     if (!this._isOpen)
        throw new ApplicationException(textClosed);
              ;
if (value != null)
   this._listBox.Dispatcher.BeginInvoke
         (new Action(() =>
                this._listBox.Items.Add(value)));
   }
 public override void Write(char[] buffer, int index,
                   int count)
  {
   String toAdd = "";
   if (!this._isOpen)
      throw new ApplicationException(textClosed); ;
   if (buffer == null || index < 0 || count < 0)
       throw new ArgumentOutOfRangeException("buffer");
   if ((buffer.Length - index) < count)
      throw new ArgumentException("The buffer is too small");
          for (int i = 0; i < count; i++)
             toAdd += buffer[i];
      this._listBox.Dispatcher.BeginInvoke
        (new Action(() => this._listBox.Items.Add(toAdd)));
   }
  }
}
```

ListBoxTextWriter კლასი არის მიღებული აბსტრაქტული TextWriter კლასიდან და უზრუნველყოფს Write() მეთოდის განხორციელებას, რომელიც ამატებს სტრიქონს

ListBox-ში (თუ გსურთ განახორციელოთ Write() მეთოდის სამი გადატვირთვა, იმისთვის რომ იყოს მიღებული, როგორც char ან string ან char [] მასივი).

არსებული კონსტრუქტორი იყენებს სტატიკურ ApplicationInterface კლასს MainWindow-ის lstEvents კონტროლის მისაღებად. იგი ასევე უზრუნველყოფს კონსტრუქტორს, რომელშიც ListBox შეიძლება შესრულდეს. ეს კონსტრუქტორი გამოიყენება ApplicationInterface კლასის AddEvent () მეთოდით.

ListBox-ის Add() მეთოდი ხორციელდება აპლიკაციის შესრულების ნაკადის (thread) შემდეგაც. იგი აკეთებს ამას Dispatcher-ის BeginInvoke() მეთოდის გამოყენებით, რომელიც ასოცირდება lstEvents მართვის ელემენტთან. ეს საშუალებას აძლევს მეთოდს, იმუშაოს სხვადასხვა ნაკადებიდან გამოძახების დროსაც.

იმის გამო, რომ ListBoxTextWriter კლასი არის მიღებული TextWriter-დან, შეიძლება მისი მითითება, როგორც TextWriter თვისებისა ნებისმიერი WriteLine ქმედებისათვის. და სტატიკური ApplicationInterface კლასის გამო, ListBoxTextWriter კლასს შეუძლია წვდომა lstEvents ელემენტზე აპლიკაციის გარედანაც კი.

ასე რომ, არსებობს სამი გზა lstEvents მართვის ელემენტში ტექსტის დასამატებლად:

- აპლიკაციის შიგნიდან, გამოიყენება ლოკალური AddEvent () მეთოდი;
- აპლიკაციის გარედან, გამოიყენება ApplicationInterface კლასის AddEvent () მეთოდი;
- WriteLine ქმედებიდან, მიეთითება TextWriter თვისება ListBoxTextWriter-ზე.

28.7. ბიზნესპროცესის რეალიზაცია

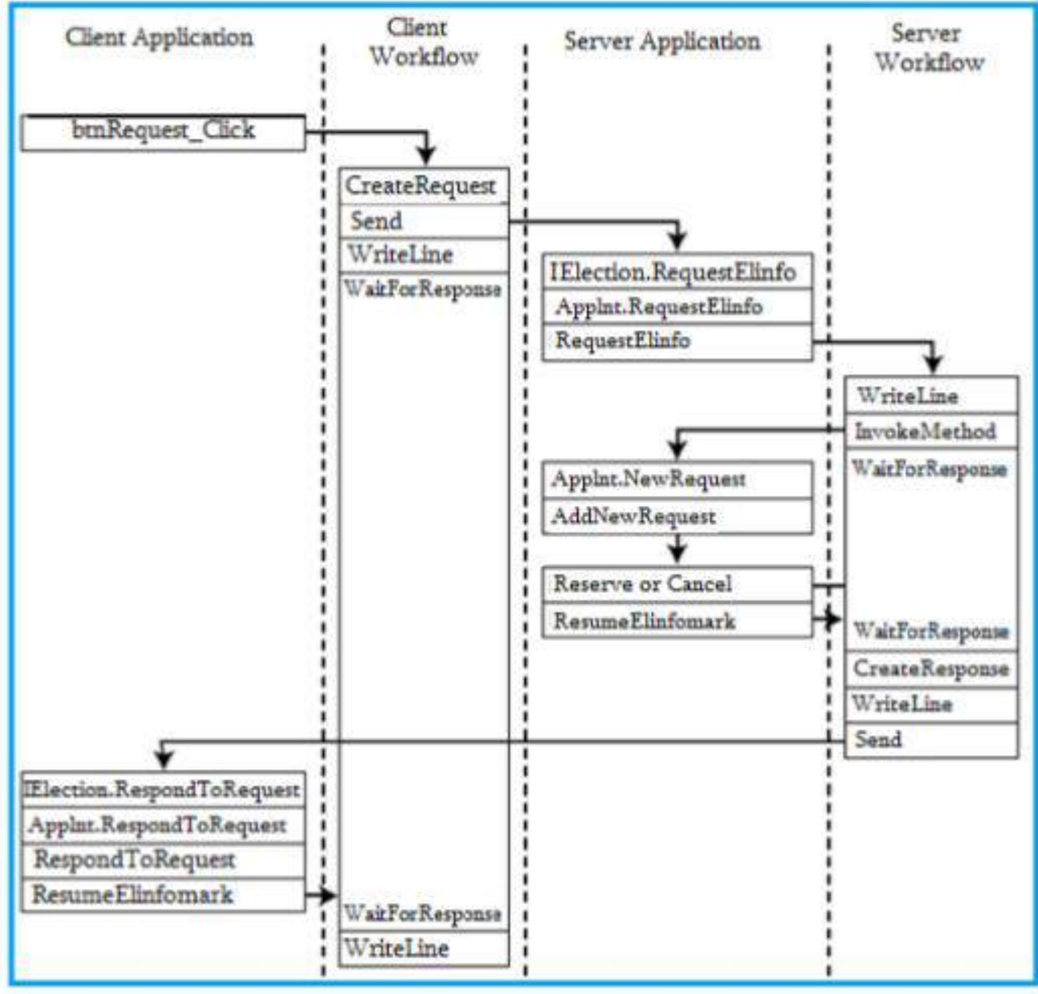
28.18 ნახაზზე ნაჩვენებია ზოგადი ლოგიკა და შეტყობინებათა ნაკადი. ამ სქემის ელემენტები მოგვიანებით იქნება ახსნილი, ამჯერად კი განხილულ იქნება ძირითადი

გავხსნათ Election.cs ფაილი, რომელშიც გამოჩნდება ინტერფეისის შემდეგი განსაზღვრება:

```
[ServiceContract]
public interface IElectionSys
 {
   [OperationContract]
    void Elinfo(ElectionRequest request);
    [OperationContract]
   void RespondToRequest(ElectionResponse response);
 }
```

საჭიროა მცირე ცვლილების ჩატარება. OperationContract-ს უნდა დაემატოს (IsOneWay = true). ქვემოთ ნაჩვენებია ეს:

```
[ServiceContract]
public interface IElectionSys
  {
    [OperationContract(IsOneWay = true)]
    void RequestElinfo(ElectionRequest request);

   [OperationContract(IsOneWay = true)]
   void RespondToRequest(ElectionResponse response);
 }
```



ნახ.28.18. მოდულების მუშაობის ლოგიკა და
შეტყობინებათა ნაკადი

შეტყობინება იგზავნება ბიზნესპროცესთან ერთად, მაგრამ პასუხი მიიღება ServiceHost-ით აპლიკაციის შიგნით. ასე, რომ ეს არაა ტქნიკური ორმხრივი საუბარი. არსებობს შეტყობინებები ორივე მიმართულებით. რადგან გაგზავნის და მიღების საბოლოო წერტილები სხვადასხვაა, WCF ამას აფიქსირებს როგორც ცალკე ერთმიმართულებიანი შეტყობინებები.

28.8. სერვისის კონტრაქტის რეალიზაცია

სერვისის კონტრაქტი განსაზღვრავს მხოლოდ ხელმისაწვდომ მეთოდებს, იგი არ უზრუნველყოფს მათ იმპლემენტაციას (რეალიზაციას). ჩვენი პროექტისთვის აუცილებელია ამ საკითხის გადაწყვეტა. ამიტომაც, Solution Explorer-ში ElectionSys -ზე მარჯვენა ღილაკით ვირჩევთ Add Class. მივუთითებთ კლასის სახელს ClientService.cs. მისი კოდის რეალიზაცია ნაჩვენებია 28.6 ლისტინგში.

```
//---- ლისტინგი 28.6 ----- ClientService.cs -----
using System;
using System.ServiceModel;
namespace ElectionSys
 {
   public class ClientService : IElectionSys
     {
       public void RequestElinfo(ElectionRequest request)
         {
           ApplicationInterface.RequestElinfo(request);
         }
       public void RespondToRequest(ElectionResponse response)
         {
            ApplicationInterface.RespondToRequest(response);
         }
     }
 }
```

ეს რეალიზაცია იყენებს ApplicationInterface სტატიკურ კლასს, რომელიც უკვე შექმნილია ჩვენ მიერ. ყოველი მეთოდი უბრალოდ იძახებს ApplicationInterface კლასის შესაბამის მეთოდს. გავხსნათ ApplicationInterface.cs ფაილი და დავამატოთ შემდეგი მეთოდები:

```
public static void RequestElinfo(ElectionRequest request)
  {
     if (_app != null)
   _app.RequestElinfo(request);
```

```
}
public static void  RespondToRequest(ElectionResponse response)
 {
    if (_app != null)
   _app.RespondToRequest(response);
 }
```

ეს მეთოდები თავის მხრივ იძახებს შესაბამის მეთოდებს აპლიკაციაში სტატიკური მიმთითებლის გამოყენებით. საჭირო ინება ამ მეთოდების რეალიზება Election.xaml.cs ფაილში, რასაც მოგვიანებით დავუბრუნდებით.

28.9. ServiceHost -ის რეალიზაცია

აპლიკაციისთვის აუცილებელია ServiceHost-ის რეალიზაცია შემავალი შეტყობინებების მისაღებად (მოსასმენად). გავხსნათ Election.xaml.cs ფაილი და დავამატოთ შემდეგი კლასის წევრები:

```
private ServiceHost _sh;
```

იგი უნდა მოთავსდეს კონსტრუქტორის წინ. ასე:

```
public partial class MainWindow : Window
 {
    private ServiceHost _sh;
    public MainWindow()
     {
        InitializeComponent();
       ApplicationInterface._app = this;
     }
```

ServiceHost იწყება მაშინ, როცა ფანჯარა ჩატვირთულია და იხურება, როცა ფანჯარა ამოტვირთულია. მეთოდების დამატება ნაჩვენებია 28.7 ლისტინგში MainWindow კლასისათვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

```
// -- ლისტინგი 28.7 --- The Loaded and Unloaded Event Handlers ----
private void Window_Loaded(object sender, RoutedEventArgs e)
 {
   // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და
  //  მისი ქსელური მისამართი
   Configuration config = ConfigurationManager.OpenExeConfiguration
                        (ConfigurationUserLevel.None);
    AppSettingsSection app = (AppSettingsSection)config.GetSection ("appSettings");
    string adr = app.Settings["Address"].Value;
```

```
  // ფილიალის სახელის გამოტანა ფორმაზე
  lblBranch.Content = app.Settings["Branch Name"].Value;
  // ServiceHost-ის შექმნა
   _sh = new ServiceHost(typeof(ClientService));
  // დასასრულის წერტილის (Endpoint) დამატება
    string szAddress = "http://localhost:" + adr + "/ClientService";
   System.ServiceModel.Channels.Binding bBinding = new BasicHttpBinding();
 _sh.AddServiceEndpoint(typeof(ILibraryReservation),bBinding, szAddress);
// ServiceHost-ის გახსნა შეტყობინებების მისაღებად (listen)
  _sh.Open();    // ListBoxTextWriter -ის ტესტირება
  //ListBoxTextWriter lbtw = new ListBoxTextWriter();
 //lbtw.Write("ეს არის ტესტი - This is a test");
}
private void Window_Unloaded(object sender, RoutedEventArgs e)
 {   // service host-ის დატოვება
    _sh.Close();
}
```

მოვლენის დამმუშავებელი Loaded ხსნის კონფიგურაციის ფაილს და ათავსებს საარჩევნო უბნის (ფილიალის) სახელს lblBranch - მართვის ელემენტში, ამიტომაც ფორმა ასახავს ლოკალური ფილიალის სახელს. შემდეგ იქმნება ServiceHost თანამგზავრი (passing) ClientService კლასისა, რომელიც ახლახანს შევქმენით როგორც მისი რეალიზაცია. შემდეგ იგი აკონფიგურირებს დასასრულის წერტილს ServiceHost-თვის, იყენებს რა ცნობილი მისამართის, მიბმისა და კონტრაქტის სამეულს. Unloaded მოვლენის დამმუშავებელი უბრალოდ ხურავს ServiceHost-ს, ასე რომ, მეტი აღარ მოხდება შეტყობინებების მიღება.

28.10. SendRequest ბიზნესპროცესის რეალიზაცია

ახლა შევასრულოთ სამუშაო პროცესების რეალიზაცია. Solution Explorer-ის ElectionSys ზე მარჯვენა ღილაკით ავირჩიოთ Add -> Class. სახელი ElectionWF.cs. კოდის რეალიზაცია მოცემულია 28.8 ლისტინგში.

```
// --- ლისტინგი 28.8 --- ElectionWF.cs ---
using System;
using System.Activities;
using System.Activities.Statements;
using System.ServiceModel.Activities;
using System.ServiceModel;
```

```
using System.ServiceModel.Channels;
using System.Runtime.Serialization;
using System.Xml.Linq;
using System.IO;

namespace ElectionSys
{
  // ეს ფაილი შეიცავს ორი workflow-ის განსაზღვრას:
  // SendRequest - აინიციალიზირებს  ახალ მოთხოვნებს,
  // ProcessRequest - ამუშავებს შემოსულ მოთხოვნებს
 public sealed class SendRequest : Activity
   { // შემავალი და გამომავალი არგუმენტების განსაზღვრა----
 public InArgument<string> ArealName { get; set; }
  public InArgument<string> MajorDeputy { get; set; }
  public InArgument<string> Region { get; set; }
  public InArgument<TextWriter> Writer { get; set; }
  public OutArgument<ElectionResponse> Response {get; set;}
  public SendRequest()
   {
        // ცვლადების განსაზღვრა ამ workflow -სთვის ---
     Variable<ElectionRequest> request = new Variable<ElectionRequest>
               {Name="request" };
    Variable<string> requestAddress = new Variable<string> { Name =
               "RequestAddress" };
    Variable<bool> reserved = new Variable<bool> { Name =  "Reserved" };
    // SendRequest workflow -ის განსაზღვრა ---
     this.Implementation = () => new Sequence
       {
         DisplayName = "SendRequest", Variables = {request, requestAddress,
                    reserved
        },
   Activities =
    {
       new CreateRequest
        {
          ArealName = new InArgument<string>
              (env =>  ArealName.Get(env)),
```

```
      MajorDeputy = new InArgument<string>
          (env =>  MajorDeputy.Get(env)),
     Region = new InArgument<string>
          (env => Region.Get(env)),
     Request = new OutArgument<ElectionRequest>
           (env => request.Get(env)),
    RequestAddress = new OutArgument<string>
         (env => requestAddress.Get(env))
   },
   new Send
    {
      OperationName = "RequestElinfo",
      ServiceContractName = "IElectionSys",
      Content = SendContent.Create
         (new InArgument<ElectionRequest>(request)),
      EndpointAddress = new InArgument<Uri>
            (env => new Uri("http://localhost:" +
            requestAddress.Get(env) + "/ClientService")),
      Endpoint = new Endpoint
        {
          Binding = new BasicHttpBinding()
        },
     },
   new WriteLine
    {
      Text = new InArgument<string>
          (env => "Request sent; waiting for response"),
      TextWriter = new InArgument<TextWriter>
          (env =>  Writer.Get(env))
     },
   new WaitForInput<ElectionResponse>
    {
       ElinfomarkName = "GetResponse",
       Input = new OutArgument<ElectionResponse>
            (env => Response.Get(env))
   },
new WriteLine
```

```
    {
      Text = new InArgument<string> (env => "Response received from " +
      Response.Get(env).Provider.BranchName + " [" +
      Response.Get(env).Reserved.ToString() + "]"),
      TextWriter = new InArgument<TextWriter>
                    (env => Writer.Get(env))
    },
   }
  };
 }
}       // აქ უნდა დაემატოს 28.8 ლისტინგი - დასასრული"
}
```

უნდა აღვნიშნოთ, რომ ყოველ WriteLine ქმედებას აქვს დამატებითი თვისება:

TextWriter = new ListBoxTextWriter()

ის მიუთითებს იმაზე, რომ ახალი კლასი ListBoxTextWriter, რომელიც იქნა რეალიზებული, უნდა იქნას გამოყენებული ამ ტექსტის დისპლეიზე გამოსატანად. ეს გამოიწვევს ტექსტის ასახვას lstEvents მართვის ელემენტში.

განსხვავება იმისა, რომ მომხმარებლის ქმედება WaitForInput გამოიყენება Receive ქმედების ნაცვლად. აპლიკაცია მიიღებს საპასუხო შეტყობინებას უშუალოდ (პირდაპირ). როცა მიღებულ იქნება პასუხი, მაშინ აპლიკაცია აღადგენს სამუშაო პროცესს, რომელიც მიმდინარეობს ElectionResponse კლასში.

ყურადსაღებია, რომ მომხმარებლის ქმედება განისაზღვრება როგორც WaitForInput <ElectionResponse>, მიუთითებს რა, რომ გადასაცემი მონაცემები იქნება ElectionResponse კლასის.

28.11. ProcessRequest ბიზნესპროცესის რეალიზაცია

ProcessRequest ბიზნესპროცესი განსაზღვრება მოცემულია 28.9 ლისტინგში. ჩავამატოთ ეს კოდი ElectionWF.cs ფაილში.

```
// ---- ლისტინგი 28.9 ---------ElectionWF.cs დამატება ------
public sealed class ProcessRequest : Activity
 {
  public InArgument<ElectionRequest> request {get; set;}
  public InArgument<TextWriter> Writer { get; set; }
  public ProcessRequest()
```

```
{
// ცვლადების განსაზღვრა ამ workflow-სთვის ---
Variable<ElectionResponse> response =      new
           Variable<ElectionResponse> { Name = "response" };
Variable<bool> reserved = new Variable<bool> { Name = "Reserved" };
Variable<string> address = new Variable<string> { Name = "Address" };
// ProcessRequest workflow-ს განსაზღვრა ---
this.Implementation = () => new Sequence
 {
    DisplayName = "ProcessRequest",
    Variables = { response, reserved, address },
    Activities =
     {
      new WriteLine
       {
        Text = new InArgument<string>(env => "Got request from: " +
                request.Get(env).Requester.BranchName),
        TextWriter =  new InArgument<TextWriter>
               (env  => Writer.Get(env))
    },
    new InvokeMethod
     {
      TargetType = typeof(ApplicationInterface),
      MethodName = "NewRequest",
      Parameters =
       {
        new InArgument<ElectionRequest>(env => request.Get(env))
       }
     },
  new WaitForInput<bool>
     {
       ElinfomarkName = "GetResponse",
       Input = new OutArgument<bool>(env => reserved.Get(env))
     },
  new CreateResponse
     {
       Request = new InArgument<ElectionRequest>
```

```
                (env => request.Get(env)),
          Reserved = new InArgument<bool>(env =>  reserved.Get(env)),
          Response = new OutArgument<ElectionResponse>
                   (env => response.Get(env))
         },
      new WriteLine
        {
          Text = new InArgument<string>(env => "Sending response to: " +
              request.Get(env).Requester.BranchName),
          TextWriter = new InArgument<TextWriter>
               (env => Writer.Get(env))
        },
 new Send
  {
    OperationName = "RespondToRequest",ServiceContractName =
     "ILibraryReservation", EndpointAddress = new InArgument<Uri>
                  (env => new Uri("http://localhost:" +
                   request.Get(env).Requester.Address +
                  "/ClientService")),
    Endpoint = new Endpoint
      {
        Binding = new  BasicHttpBinding()
      },
   Content = SendContent.Create
             (new InArgument<ElectionResponse>(response))
          }
        }
      };
  }
 }
}
```

იმის მაგივრად, რომ დაწყება იყოს Receive ქმედებით, რათა მიღებულ იქნას შემავალი მოთხოვნა, ElectionRequest გადასცემს სამუშაო პროცესს შემავალი არგუმენტის გამოყენებით. WriteLine ქმედება, რომელიც მოსდევს მას, ცნობს შემავალ მოთხოვნას.

InvokeMethod ქმედება გამოვიყენოთ მონაცემთა გადასაცემად აპლიკაციაში. ApplicationInterface კლასი მოხერხებულადაა შესრულებული ამ მიზნით. იგი უზრუნველყოფს სამუშაო პროცესს, რათა განხორციელდეს გამოძახება აპლიკაციაში.

InvokeMethod ქმედება იძახებს ApplicationInterface კლასის NewRequest() მეთოდს ElectionRequest კლასში გადასაცემად.

გავხსნათ ApplicationInterface.cs ფაილი და დავამატოთ მეთოდი, რომელიც უბრალოდ იძახებს AddNewRequest ()-ს აპლიკაციაში:

```
public static void NewRequest(ElectionRequest request)
  {
    if (_app != null)
          _app.AddNewRequest(request);
  }
```

შემდეგი აქტიურობაა მომხმარებლის WaitForInput ქმედება, რომელიც გამოიყენებოდა SendRequest სამუშაო პროცესში.

ამჯერად იგი ელოდება Bool-შესატან მითითებას, იყო თუ არა დაჯავშნული დასახელება (სათაური). CreateResponse და WriteLine ქმედებები იგივეა. აქ გამოიყენებოდა SendReply ქმედება, ვინაიდან იგი იყო დაკავშირებული საწყის Receive ქმედებასთან.

ამ პროექტში, ვინაიდან არაა არავითარი Receive ქმედება, ჩვენ გამოვიყენებთ Send ქმედებას. საყურადღებოა, რომ EndpointAddress აწყობილია მისამართის გამოყენებით (პორტის ნომერი), რომელიც გათვალისწინებულია შესატან მოთხოვნაში.

28.12. აპლიკაციის რეალიზაცია

შემდეგი ბიჯი არის აპლიკაციის რეალიზაცია. არსებობს მოვლენათა რამდენიმე დამმუშავებელი (event handlers), რომელთა რეალიზაცია აუცილებელია, აგრეთვე მეთოდები, რომლებიც გამოიძახება სტატიკური ApplicationInterface კლასით.

28.13. ბიზნესპროცესების ეგზემპლარების მხარდაჭერა

აპლიკაცია ახდენს ბიზნესპროცესის ეგზემპლარების მონიტორინგს, ამიტომაც მას შეუძლია განაახლოს სწორი ეგზემპლარი. ამის შესრულება შესაძლებელია მარტივად ობიექტის ლექსიკონით. გავხსნათ Election.xaml.cs ფაილი და დავამატოთ კლასის წევრები მომხმარებლის ServiceHost _sh სტრიქონის ქვემოთ:

```
private IDictionary<Guid, WorkflowApplication> _incomingRequests;
private IDictionary<Guid, WorkflowApplication> _outgoingRequests;
```

ისინი იყენებს ბიზნესპროცესის ეგზემპლარის იდენტიფიკატორს, როგორც ლექსიკონის გასაღებს და WorkflowApplication ობიექტს, როგორც მნიშვნელობას. ვინაიდან აპლიკაცია ამუშავებს ორივე სამუშაო პროცესს SendRequest და ProcessRequest, ამიტომაც საჭირო იქნება ლექსიკონის ორი ობიექტი. დავამატოთ კონსტრუქტორში კოდი ამ ობიექტების ინიციალიზებისათვის:

```
_incomingRequests = new Dictionary<Guid, WorkflowApplication>();
_outgoingRequests = new Dictionary<Guid, WorkflowApplication>();
```

საჭიროა კიდევ ერთი მცირე ცვლილება მომხმარებლის CreateRequest ქმედებაში. ბიზნესპროცესის ეგზემპლარის ID გამოყენებულ უნდა იქნას როგორც ElectionRequest კლასის RequestID ველი. აპლიკაცია მას გამოიყენებს პროცესის განახლების დროს. გავხსნათ CreateRequest.cs ფაილი და შევცვალოთ გამოძახება, რომელიც ქმნის ElectionRequest კლასს, ალტერნატიული კონსტრუქტორის გამოსაყენებლად, რომელიც იღებს მეხუთე პარამეტრს RequestID -თვის. დავამატოთ მუქი სტრიქონი კოდის შემდეგ ტექსტში:

```
// -- ElectionRequest კლასის შექმნა და მისი შევსება
// შესატანი არგუმენტებით -----
ElectionRequest r = new ElectionRequest
    ( ArealName.Get(context),
      MajorDeputy.Get(context),
      Region.Get(context),
     new Branch
     {
       BranchName = app.Settings["Branch Name"].Value,
       BranchID = new Guid(app.Settings["ID"].Value),

       Address = app.Settings["Address"].Value
     },
  context.WorkflowInstanceId  // ეს დაემატა!!!
  )
```

28.14. მოვლენათა დამმუშავებელი (Event Handlers)

ახალი მოთხოვნის შესაქმნელად მომხმარებელი შეავსებს:

მაჟორ_დეპუტატის_გვარის, ოლქის_დასახელების, რეგიონის_სახელის

ველებს და აამოქმედებს Send Request ღილაკს. ამ მოვლენის ღილაკის რეალიზება მოცემულია 28.10 ლისტინგში, Election.xaml.cs ფაილში.

```
// --- ლისტინგი 28.10----- Click Event ის რეალიზება -----
private void btnRequest_Click(object sender,RoutedEventArgs e)
 {
   // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად ---
  Dictionary<string, object> parameters = new Dictionary<string, object>();
  parameters.Add("MajorDeputy", txtMajorDeputy.Text);
  parameters.Add("ArealName", txtArealName.Text);
```

```
  parameters.Add("Regioni", txtRegioni.Text);
  parameters.Add("Writer", new ListBoxTextWriter(lstEvents));
 WorkflowApplication i = new WorkflowApplication(new SendRequest(),
                parameters);
    _outgoingRequests.Add(i.Id, i);
     i.Run();
}
```

ამ მეთოდის პირველი ნაწილი ჩვენთვის ნაცნობია. იგი იყენებს ობიექტის ლექსიკონს შემავალი არგუმენტების შესანახად, რომლებიც უნდა გადაეცეს ბიზნესპროცესს. შემდეგ იგი ქმნის WorkflowApplication-ს, რომლის კონსტრუქტორსაც გადაეცემა პარამეტრები:

ბიზნესპროცესების დეფინიცია

ობიექტის ლექსიკონი, რომელიც შეიცავს შემავალ არგუმენტებს

WorkflowApplication-ი შემდეგ ემატება _outgoingRequests კოლექციას. ბოლოს, ეგზემპლარი გაიშვება Run () მეთოდით.

მოთხოვნების სიის ფორმაზე მოთავსებულია ღილაკები Reserve და Cancel, რომელთაც იყენებს მომხმარებელი იმის მისათითებლად, თუ რომელი ელემენტი იყო გამოყენებადი.

28.11 ლისტინგი აღწერს ამ ღილაკებისათვის მოვლენათა დამმუშავებლების რეალიზაციას. დავამატოთ ეს მეთოდები Election.xaml.cs კლასში.

// -- ლისტინგი 28.11 --Reserve და Cancel ღილაკების რეალიზაცია --

```
 // -- Reserve ღილაკის მოვლენის დამმუშავებელი ---
private void Reserve(object sender, RoutedEventArgs e)
  {
    // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ----
   FrameworkElement fe = (FrameworkElement)sender;
   Guid id = (Guid)fe.Tag;
   ResumeBookmark(id, true);
  }
  // -- Cancel ღილაკის მოვლენის დამმუშავებელი ---
    private void Cancel(object sender, RoutedEventArgs e)
     {
       // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ----
      FrameworkElement fe = (FrameworkElement)sender;
      Guid id = (Guid)fe.Tag;         ResumeBookmark(id, false);
     }
```

```
private void ResumeBookmark(Guid id, bool bReserved)
  {
    WorkflowApplication i = _incomingRequests[id];
    try
      {
        i.ResumeBookmark("GetResponse", bReserved);
      }
   catch (Exception e)
     {
       AddEvent(e.Message);
     }
  }
```

მოვლენის დამმუშავებლები იღებს ბიზნესპროცესის ეგზემპლარის ID-ს ღილაკის Tag თვისებიდან. შემდეგ იძახებს ResumeBookmark () მეთოდს, მიაწოდებს true-ს ან falseს, იმის მიხედვით, თუ რომელი ღილაკი იყო ამოქმედებული.

ResumeBookmark() მეთოდი მიიღებს WorkflowApplication-ს _incomingRequests კოლექციიდან და გამოიძახებს მის ResumeBookmark() მეთოდს. გადაეცემა სანიშნის სახელი (bookmark name) და მნიშვნელობა, რომელშიც ეგზემპლარი განახლდება (resumed).

28.15. ApplicationInterface მეთოდები

ჩვენ განვსაზღვრეთ ApplicationInterface კლასის სამი მეთოდი. ახლა უნდა უზრუნველვყოთ მათი რეალიზაცია MainWindow კლასში. გავხსნათ Election.xaml.cs ფაილი და ჩავამატოთ ამ მეთოდების რეალიზაცია 28.12 ლისტინგის მიხედვით.

//--ლისტინგი 28.12 --ApplicationInterface კლასის მეთოდების რეალიზაცია--

```
public void RequestElinfo(ElectionRequest request)
 {
   // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად ---
  Dictionary<string, object> parameters = new Dictionary<string, object>();
  parameters.Add("request", request);
  parameters.Add("Writer", new  ListBoxTextWriter(lstEvents));
 WorkflowApplication i = new
          WorkflowApplication(new ProcessRequest(), parameters);
  request.InstanceID = i.Id;
 _incomingRequests.Add(i.Id, i);
   i.Run();
```

```
}
public void RespondToRequest(ElectionResponse response)
{
  Guid id = response.RequestID;
  WorkflowApplication i = _outgoingRequests[id];
   try
    {
      i.ResumeBookmark("GetResponse", response);
    }
   catch (Exception e2)
     {
       AddEvent(e2.Message);
     }
    }
 public void AddNewRequest(ElectionRequest request)
 {
    this.requestList.Dispatcher.BeginInvoke
             (new Action(() =>  this.requestList.Items.Add(request)));
 }
```

RequestBook () მეთოდი ანალოგიურია btnRequest_Click () მეთოდის. იგი გამოიძახება მაშინ, როცა შემავალი შეტყობინება მიღებულია ServiceHost -დან და სერვისის კონტრაქტის RequestBook მეთოდი მითითებულია. ის აგებს ობიექტის ლექსიკონს ერთი არგუმენტის შესანახად, ქმნის WorkflowApplication-ს, ამატებს მას _incomingRequests კოლექციაში, ხოლო შემდეგ აამოქმედებს სამუშაო პროცესს.

RespondToRequest () მეთოდი ასევე გამოიძახება ServiceHost-დან მიღებული შეტყობინებით. იგი გამოიძახება მაშინ, როცა RespondToRequest მეთოდია მითითებული. ეს ხდება მაშინ, როცა სხვა ფილიალები აგზავნის უკან პასუხს შემოსულ მოთხოვნაზე.

იგი იღებს WorkflowApplication-ს _outgoingRequests კოლექციიდან და აღადგენს სანიშნეს, გამავალ ElectionResponse კლასში.

AddNewRequest() გამოიძახება ProcessRequest მუშა პროცესით, როცა მიიღბა ახალი შეტყობინება. ეს ხდება InvokeMethod ქმედების დახმარებით. იგი უბრალოდ დაამატებს ჩანაწერს ListView-კონტროლის RequestList-ელემენტში. ვინაიდან ის გამოძახებულ უნდა იქნას ბიზნესპროცესის შესრულებად ნაკადში, Dispatcher კლასი გამოიყენებს შესასრულებლად Add() მეთოდს main window-ის შესრულებადი ნაკადით. Election.xaml.cs-ის სრული რეალიზაცია მოცემულია 28.13 ლისტინგში.

```
// -- ლისტინგი 28.13 -- Election.xaml.cs საბოლოო სახე -----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
 using System.ServiceModel.Description;
using System.ServiceModel.Channels;
using System.Activities;
using System.Xml.Linq;
using System.Configuration;
namespace ElectionSys
  {
    public partial class MainWindow : Window
     { private ServiceHost _sh;
      private IDictionary<Guid, WorkflowApplication>
_incomingRequests;
      private IDictionary<Guid, WorkflowApplication>
_outgoingRequests;
      public MainWindow()
       { InitializeComponent();
         ApplicationInterface._app = this;
        _incomingRequests = new Dictionary<Guid, WorkflowApplication>();
        _outgoingRequests = new Dictionary<Guid,  WorkflowApplication>();
       }
private void Window_Loaded(object sender, RoutedEventArgs e)
```

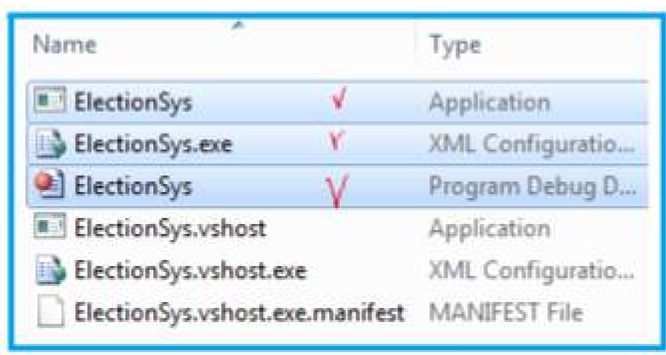
```
   {   // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და
      // მისი ქსელური მისამართი ---
     Configuration config = ConfigurationManager.OpenExeConfiguration
                     (ConfigurationUserLevel.None);
     AppSettingsSection app = (AppSettingsSection)config.GetSection("appSettings");
     string adr = app.Settings["Address"].Value;
  // ფილიალის სახელის გამოტანა ფორმაზე ----
    lblBranch.Content = app.Settings["Branch Name"].Value;
 // ServiceHost-ის შექმნა ----
    _sh = new ServiceHost(typeof(ClientService));
// დასასრულის წერტილის (Endpoint) დამატება ----
    string szAddress = "http://localhost:" + adr +  "/ClientService";
    System.ServiceModel.Channels.Binding bBinding = new BasicHttpBinding();
_sh.AddServiceEndpoint(typeof(IElectionSys), bBinding,  szAddress);
   // ServiceHost-ის გახსნა შეტყობინებების მისაღებად (listen) ---
 _sh.Open();
 }
private void Window_Unloaded(object sender, RoutedEventArgs e)
   {     // service host-ის დატოვება
    _sh.Close();
   }
  // ----- მოვლენათა დამმუშავებელი -----Event Handler-----
private void btnRequest_Click(object sender,RoutedEventArgs e)
  {
 // Setup a dictionary object for passing parameters
    Dictionary<string, object> parameters = new Dictionary<string, object>();
    parameters.Add("MajorDeputy", txtMajorDeputy.Text);
    parameters.Add("ArealName", txtArealName.Text);
    parameters.Add("Regioni", txtRegioni.Text);
    parameters.Add("Writer", new ListBoxTextWriter(lstEvents));
   WorkflowApplication i =  new WorkflowApplication(new SendRequest(), parameters);
     _outgoingRequests.Add(i.Id, i);
     i.Run();
 }
  // -- Reserve ღილაკის მოვლენის დამმუშავებელი ---
private void Reserve(object sender, RoutedEventArgs e)
   {  // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ----
```

```
    FrameworkElement fe = (FrameworkElement)sender;
    Guid id = (Guid)fe.Tag;   ResumeBookmark(id, true);
   }

  // -- Cancel ღილაკის მოვლენის დამმუშავებელი ---
 private void Cancel(object sender, RoutedEventArgs e)
  {   // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ---
    FrameworkElement fe = (FrameworkElement)sender;
    Guid id = (Guid)fe.Tag;
    ResumeBookmark(id, false);
  }

 private void ResumeBookmark(Guid id, bool bReserved)
  {
     WorkflowApplication i = _incomingRequests[id];
     try
      {
         i.ResumeBookmark("GetResponse", bReserved);
      }
    catch (Exception e)
       {            AddEvent(e.Message);
       }
    }

public void RequestElinfo(ElectionRequest request)
    {  // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად --
    Dictionary<string, object> parameters = new Dictionary<string,
                  object>();
     parameters.Add("request", request);
     parameters.Add("Writer", new ListBoxTextWriter(lstEvents));
     WorkflowApplication i = new WorkflowApplication(new ProcessRequest(), parameters);
     request.InstanceID = i.Id;
     _incomingRequests.Add(i.Id, i);
       i.Run();
   }
public void RespondToRequest(ElectionResponse response)
  {
```

```
     Guid id = response.RequestID;
    WorkflowApplication i = _outgoingRequests[id];
     try
      {
         i.ResumeBookmark("GetResponse", response);
      }
   catch (Exception e2)
      {
         AddEvent(e2.Message);
      }
   }
 public void AddNewRequest(ElectionRequest request)
  {
    this.requestList.Dispatcher.BeginInvoke (new Action(() =>
            this.requestList.Items.Add(request)));
    }
public ListBox GetEventListBox()
    {
      return this.lstEvents;
    }
    private void AddEvent(string szText)
       {
         lstEvents.Items.Add(szText);
       }
     }
  }
```

28.16. აპლიკაციის ამუშავება

პროგრამული სისტემის ასამუშავებლად საჭიროა აპლიკაციის რამდენიმე ასლის (კოპიოს) ერთად გაშვება, თითოეული თავისი კონფიგურაციის ფაილის ვერსიით. თავიდან საჭიროა F6 კლავიშის ამოქმედება solution-ის (გადაწყვეტის) აღსადგენად და კომპილატორის შენიშვნების აღმოსაფხვრელად. შევქმნათ ახალი ფოლდერი ElectionSys-ფოლდერის ქვეშ, რომელიც იმახებს ფილიალებს. შემდეგ დავაკოპიროთ ფილიალის ფოლდერში ფაილები, რომლებიც 28.19 ნახაზზეა ნაჩვენები.



ნახ.28.19. ფილიალის ფოლდერში ფაილების კოპირება

გავხსნათ ElectionSys.exe.config ფაილი (ფილიალის ქვეფოლდერში) და შევასწოროთ შემდეგნაირად:
<?xml version="1.0" encoding="utf-8" ?> <configuration> <appSettings> <add key="Branch Name" value="Olqi Ozurgeti"/> <add key="ID" value="{43E6DADD-4751-4056-8BB7-7459B5C361AB}"/>
<add key="Address" value="8730"/> <add key="Request Address" value="8000"/>
</appSettings> </configuration>

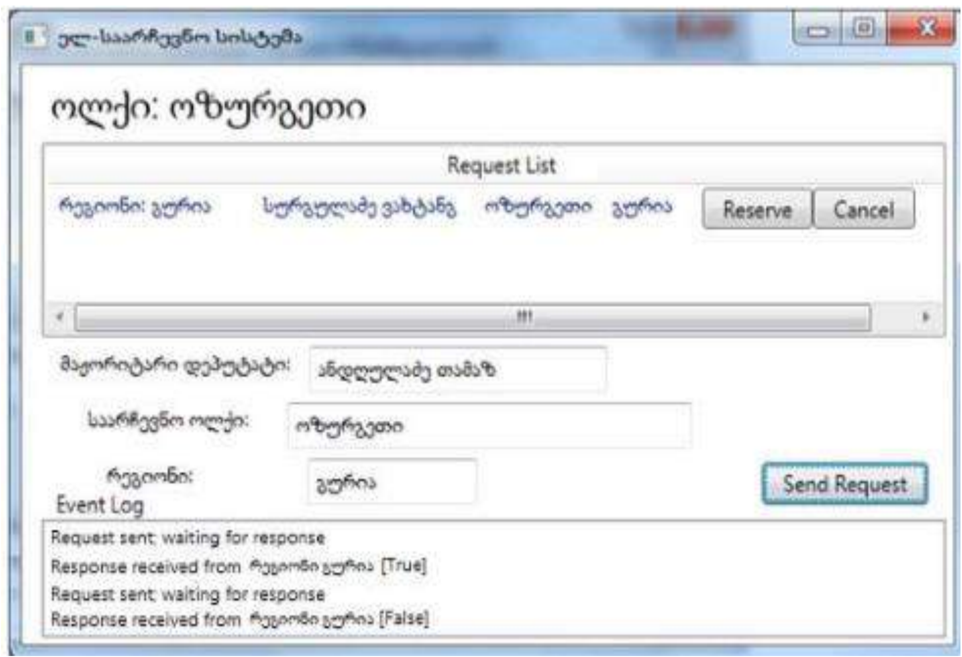
შენიშვნა: თუ შედეგი მიღებულია შეცდომით, უნდა ვცადოთ აპლიკაციის გაშვება ადმინისტრატორის უფლებებით (!).

ფილიალის ფოლდერში ElectionSys.exe ფაილი ორჯერ დავკლიკოთ. აპლიკაცია ასე გამოიყურება (ნახ.28.20).

ნახ.28.20. ფილიალის ფორმა

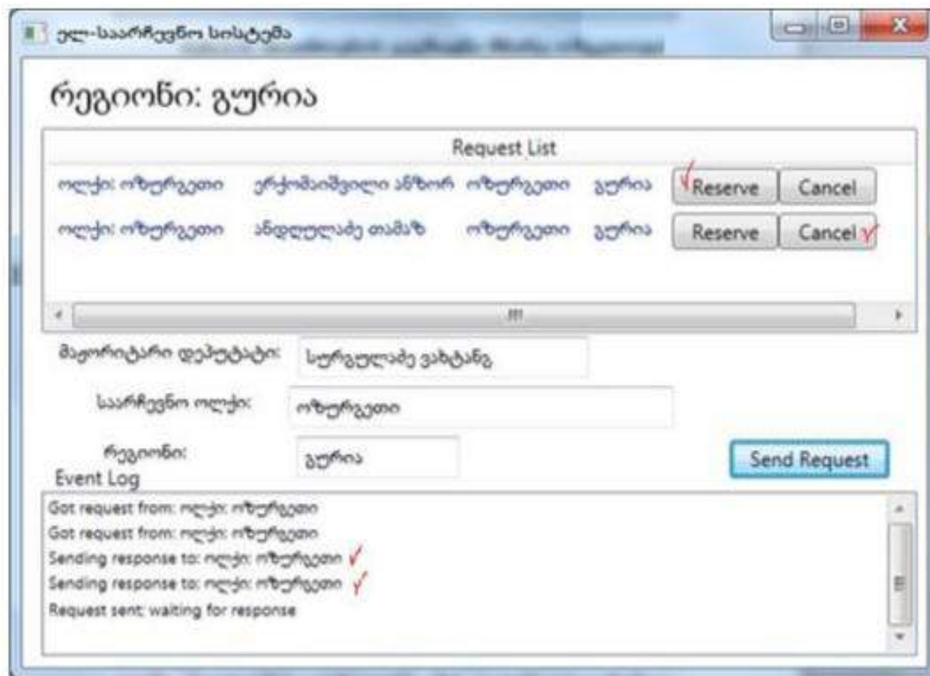
Visual Studio-ში F5-ით გავმართოთ აპლიკაცია. ანალოგიური ფანჯარა უნდა მივიღოთ, ოღონდ სათაურით - „საარჩევნო ოლქი“ (ან „რეგიონალური საარჩევნო კომისია“ ან „ცენტრალური საარჩევნო კომისია“ და ა.შ.).

ერთ-ერთ აპლიკაციაში შევიტანოთ მაჟორიტარი დეპუტატის გვარი, მხარე (ოლქი), რეგიონი და ავამოქმედოთ ღილაკი „მოთხოვნის გაგზავნა“ (ნახ.28.21).



ნახ.28.21. მოთხოვნის გაგზავნა ოლქიდან „ოზურგეთი“

მოთხოვნა უნდა გამოჩნდეს მეორე ფანჯრის მოთხოვნების სიაში. დააჭირეთ Reserve ღილაკს მეორე აპლიკაციაში (ნახ.28.22). გამოჩნდება შეტყობინება პირველი ფანჯრის მოვლენების ჟურნალში, რომ პასუხი მიღებულია.



ნახ.28.22. მოთხოვნის დამუშავება რეგიონში „გურია“

ვცადოთ რამდენიმე მოთხოვნის გაგზავნა ორივე ფანჯრიდან. ასევე შევამოწმოთ Cancel ღილაკი და დავრწმუნდეთ, რომ საპასუხო შეტყობინება მოვლენათა ჟურნალში (მეორე აპლიკაციისათვის) იქნება [False].

28.17. დასკვნა

ამრიგად, ჩატარებული სამუშაოების შედეგად შეიძლება შემდეგი სახის დასკვნის გამოტანა:

- ელექტრონული საარჩევნო სისტემა, როგორც კორპორაციული მართვის ობიექტი ელექტრონული მთავრობის შემადგენლობაში, მიეკუთვნება რთული და დიდი სისტემების კლასს. მისმა კომპლექსურმა ანალიზმა გვიჩვენა, რომ ასეთი სისტემების დასაპროექტებლად აუცილებელია ობიექტ-ორიენტირებული, პროცესორიენტირებული და სერვის-ორიენტირებული მოდელირების მეთოდების გამოყენება;
- საქართველოში არსებული ტრადიციული საარჩევნო სისტემის დიაგნოსტიკური გამოკვლევის საფუძველზე, აგრეთვე საზღვარგარეთული დემოკრატიული ინსტიტუტების გამოცდილების გათვალისწინებით ამ სფეროში განსაზღვრულ იქნა ძირითადი პრობლემები და ამოცანები, შემუშავდა მათი გადაწყვეტის გზები და ერთიანი ელქტრონული საარჩევნო სისტემის აგების კონცეფცია;

- ელექტრონული საარჩევნო სისტემის დაპროექტება, აპარატურული და პროგრამული რეალიზაცია უნდა განხორციელდეს მულტიმედიური საშუალებების, განაწილებული, რელაციური ბაზების, კლიენტ-სერვერ არქიტექტურისა და უსაფრთხო, საიმედო ქსელების ბაზაზე;

- ელექტრონული საარჩევნო სისტემის მულტიმედიური მონაცემთა რელაციური ბაზების ეფექტურად დასაპროექტებლად და ასაგებად მიზანშეწონილია თანამედროვე CASE-ტექნოლოგიების გამოყენება, მოდელირების კატეგორიალური თეორიისა და დაპროექტების ობიექტორიენტირებული მეთოდების საფუძველზე;

- კლიენტსერვერ არქიტექტურის ლოგიკურად ერთიანი და ფიზიკურად განაწილებული მონაცემთა რელაციური ბაზების სისტემის დამუშავება შესაძლებელი გახდა ობიექტროლური მოდელირების პრინციპების საფუძველზე და შესაბამისი გრაფულ–ანალიზური ინსტრუმენტების გამოყენებით;

- შემუშავებულია აგებული სისტემის მომხმარებელთა ინტერფეისები, ინსტრუქციები, დანერგვისა და ექსპლუატაციის პროცესების ორგანიზაციული, ტექნიკური და იურიდიული ასპექტები. განსაზღვრულია სისტემისათვის საჭირო ტექნიკური საშუალებები და მათი შესაძლებლობები.

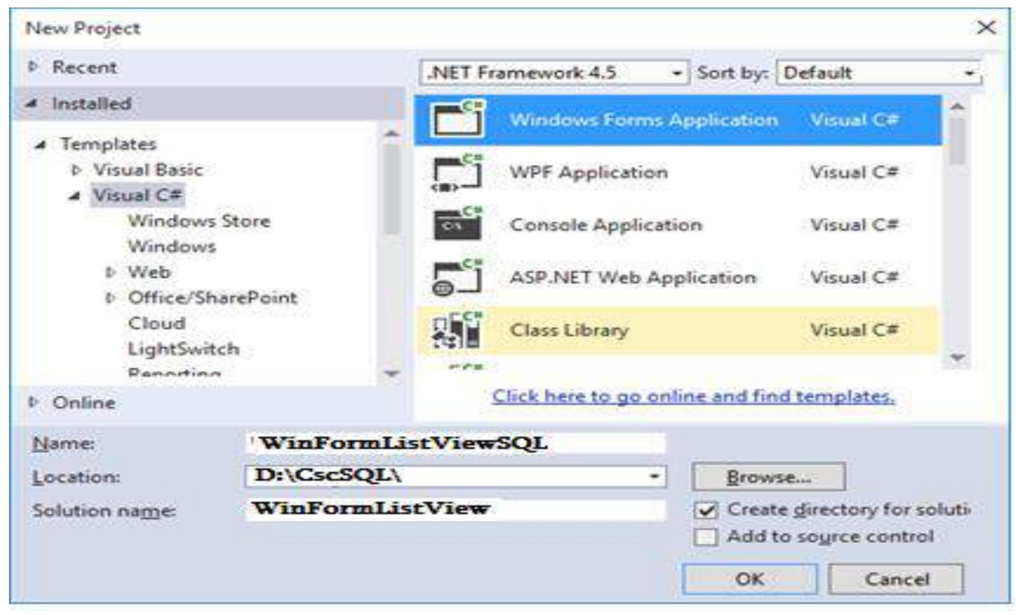
XXIX თავი
WPF-აპლიკაციის აგება საპრობლემო სფეროსში „შავი ზღვის ეკოლოგია“

29.1. შავი ზღვის საქართველოს მდინარეების მონაცემთა ბაზის აგებისა და განახლების ინტერფეისის დამუშავების ამოცანა

განიხილება ვიზუალური დაპროგრამების C# ენის ListView კლასის საფუძველზე მონაცემთა მენეჯმენტის საკითხი, SQL Server 2012 ბაზის განახლების მეთოდების Insert, Update და Delete ასაგებად.

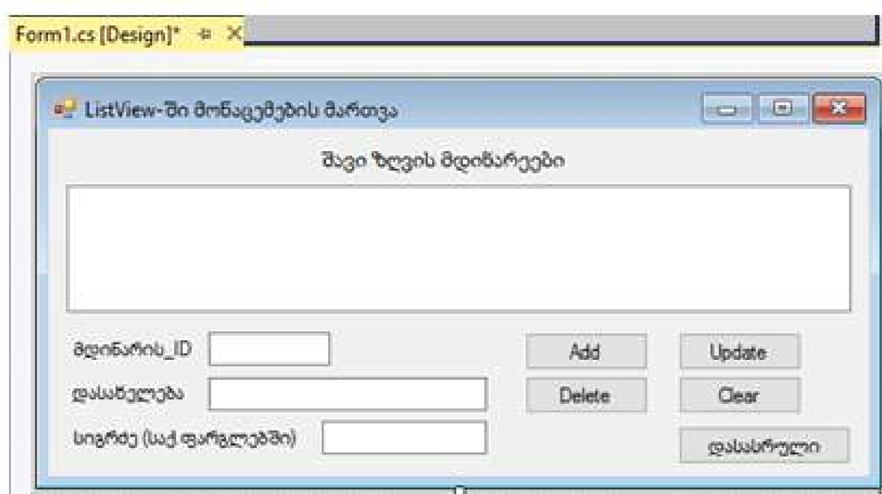
პროგრამული პროექტის აგების მაგალითისათვის განვიხილოთ ამოცანა საქართველოს აკვატორიაში შავი ზღვის მდინარეების მონაცემთა ბაზის შექმნისა და მისი ცხრილების შევსების და განახლების ინტერფეისული პროგრამის დამუშავება.

Visual Studio.NET 2013/15 გარემოში შევქმნათ ახალი პროექტი სახელით: WinFormListViewSQL (ნახ.29.1).



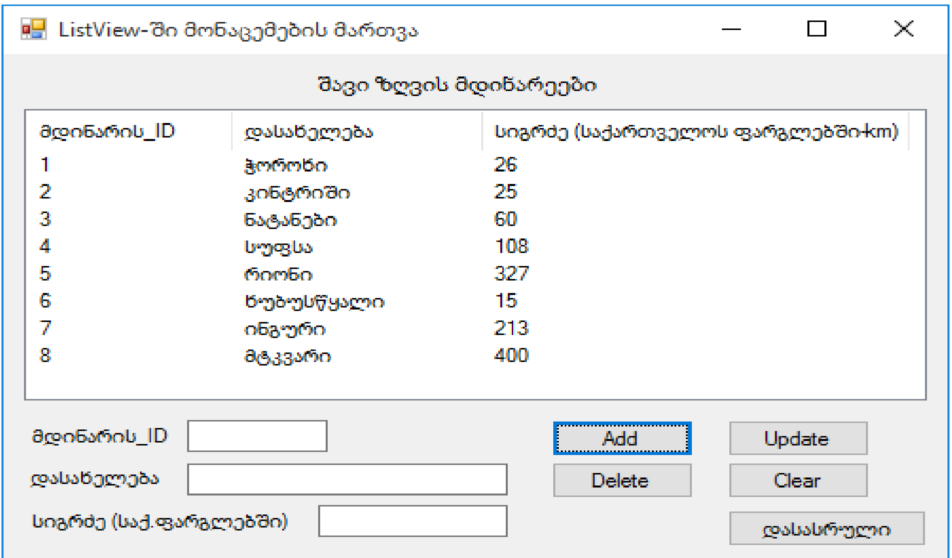
ნახ.29.1

ჩვენ მიერ დაპროექტებული ინტერფეისის ფორმა მოცემულია 29.2 ნახაზზე.



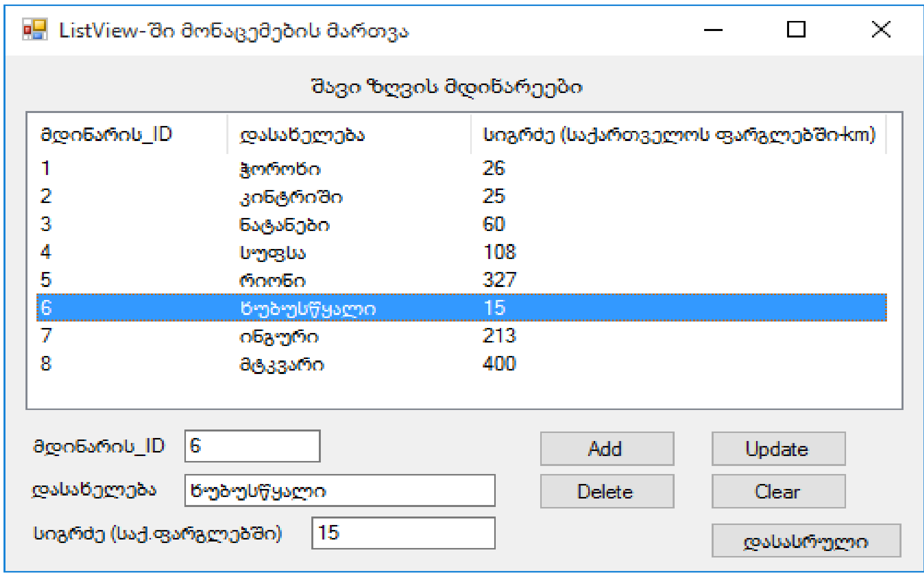
ნახ.29.2

მონაცემთა ბაზის ჩანაწერებიგამოიტანება ListView სვეტებში (ნახ.29.3).



ნახ.29.3

მე-6 სტრიქონში შეცდომაა და საჭიროა ცვლილების განხორციელება Update მეთოდით. ვირჩევთ სტრიქონს (ნახ.29.4).



ნახ.29.4

„ხუბუსწყალი" შევცვალოთ „ხობისწყალი"-თ (ნახ.29.5).

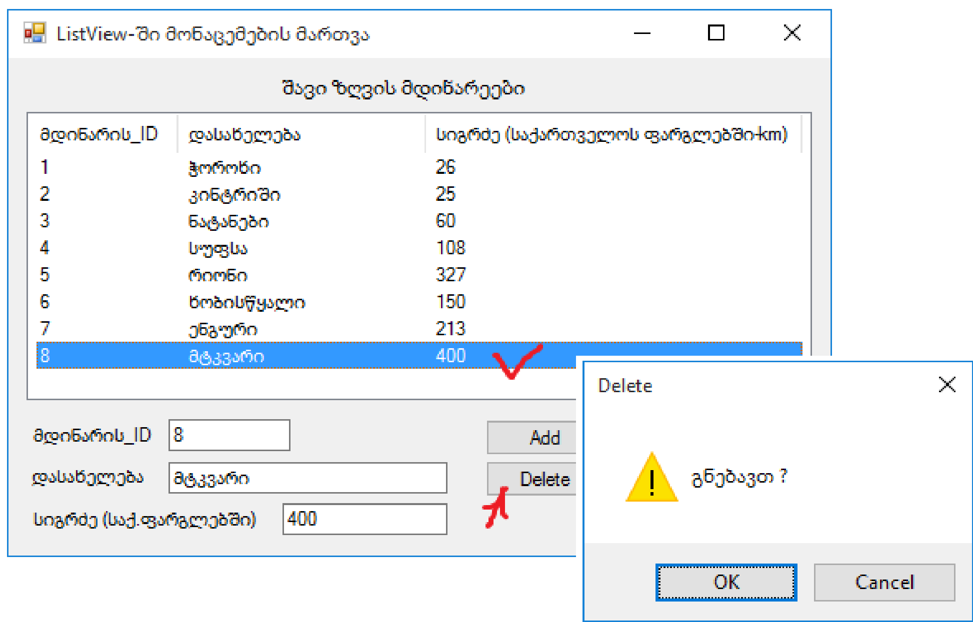
შავი ზღვის მდინარეები

მდინარის_ID	დასახელება	სიგრძე (საქართველოს ფარგლებში-km)
1	ჭოროხი	26
2	კინტრიში	25
3	ნატანები	60
4	სუფსა	108
5	რიონი	327
6	ხობისწყალი	150
7	ინგური	213
8	მტკვარი	400

მდინარის_ID Add Update

ნახ.29.5

დარჩა ერთი შესასწორებელი სტრიქონი - მდინარის სახელი „ინგური" უნდა შეიცვალოს სახელით „ენგური", ხოლო „მტკვარი" არაა შავი ზღვის მდინარე, ამიტომ ის უნდა წაიშალოს სიიდან Delete მეთოდით (ნახ.29.6).



ნახ.29.6

29.1_ლისტინგში მოცემულია ListView-ში მონაცემების დამატების, ცვლილებისა და წაშლის მეთოდების კოდები.

```
// -- ლისტინგი_29.1 ---- ListView-ის  Add, Update Delete მეთოდები --
using System;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WinFormListViewSeqD
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
      // ListView-ის თვისებები -----
```

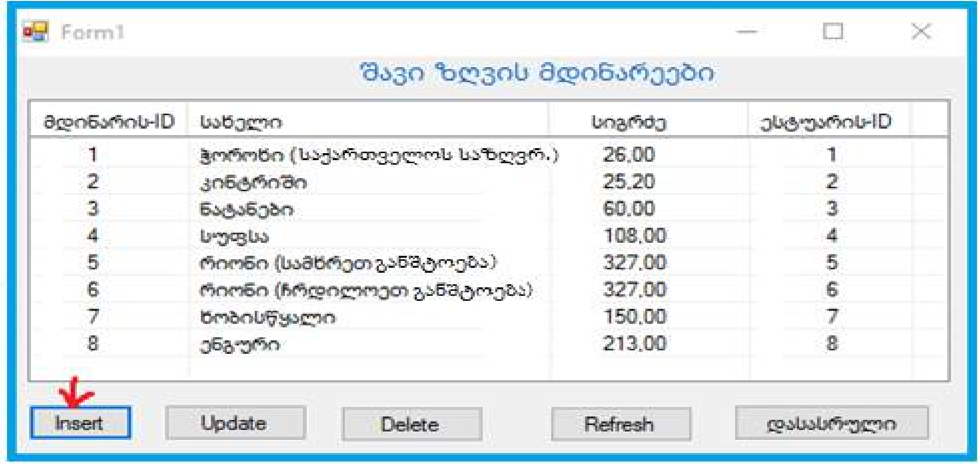
```
        listView1.View = View.Details;
        listView1.FullRowSelect = true;
        // ListView-ის ველების სიგანე-----
        listView1.Columns.Add("მდინარის_ID", 50);
        listView1.Columns.Add("დასახელება", 150);
        listView1.Columns.Add("სიგრძე (საქართველოს
                   ფარგლებში-კმ)",50);
      }
 // Add
private void add(String id, String name, String length)
 {
   //row---
   String[] row = {id, name, length};
   ListViewItem item = new ListViewItem(row);
   listView1.Items.Add(item);
 }
// update-----
private void update()
 {
   listView1.SelectedItems[0].SubItems[0].Text =
         idTxt.Text;
   listView1.SelectedItems[0].SubItems[1].Text =
         nameTxt.Text;
   listView1.SelectedItems[0].SubItems[2].Text =
         lengthTxt.Text;
  // clear txt
   idTxt.Text = "";
   nameTxt.Text = "";
   lengthTxt.Text = "";
 }
  // delete -----
  private void delete()
   {
    if (MessageBox.Show("გნებავთ წაშლა ?", "DELETE",
                 MessageBoxButtons.OKCancel,
           MessageBoxIcon.Warning) == DialogResult.OK)
      {
```

```
listView1.Items.RemoveAt(listView1.SelectedIndices[0]);
       // clear txt
        idTxt.Text = "";
        nameTxt.Text = "";
        lengthTxt.Text = "";
      }
   }
private void button1_Click(object sender, EventArgs e)
  {
    add(idTxt.Text, nameTxt.Text, lengthTxt.Text);
    // clear txt
      idTxt.Text = "";
      nameTxt.Text = "";
      lengthTxt.Text = "";
  }
private void button2_Click(object sender, EventArgs e)
  {
      update();
  }
private void button3_Click(object sender, EventArgs e)
  {
      delete();
  }
private void button4_Click(object sender, EventArgs e)
  {
    listView1.Items.Clear();
    // clear txt
    idTxt.Text = "";
    nameTxt.Text = "";
    lengthTxt.Text = "";
   }
 private void listView1_MouseClick(object sender,
            MouseEventArgs e)
   {
idTxt.Text=listView1.SelectedItems[0].SubItems[0].Text;
nameTxt.Text=listView1.SelectedItems[0].SubItems[1]
       .Text;
```

```
lengthTxt.Text=listView1.SelectedItems[0].SubItems[2]
        .Text;
    }
private void button5_Click(object sender, EventArgs e)
    {
        Close();
    }
   }
}
```

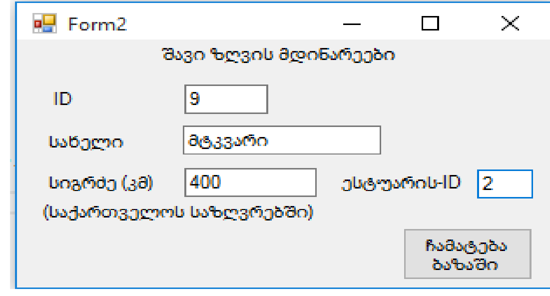
ამოცანა_2: ახლა განვიხილოთ ListView-ში მონაცემების გამოტანა SQL Server-დან და მასში ცვლილებების განხორციელება.

29.7 ნახაზზე მოცემულია ListView საწყისი ფანჯარა ბაზიდან ამოღებული სტრიქონებით:



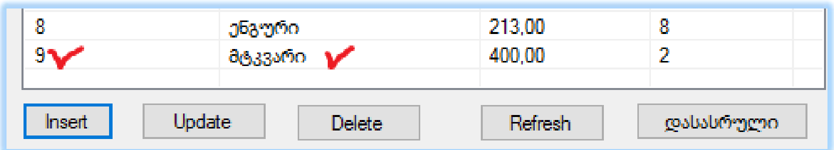
ნახ.29.7

Insert ღილაკის არჩევით გამოიტანება 29.8 ფანჯარა და ვამატებთ ახალ სტრიქონს :



ნახ.29.8

მდინარე მტკვარი ჩაემატა SQL Server ბაზაში და განახლდა ListView სია თავიდან ახალი ვერსიით (ნახ.29.9).



ნახ.29.9

- Insert მეთოდის შესაბამისი კოდი მოცემულია 29.2 ლისტინგში

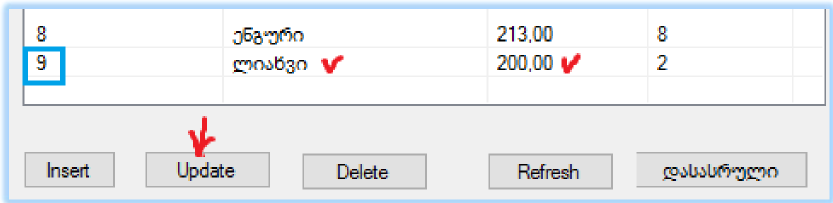
```
public partial class Form1 : Form
{  // SQL Server ბაზასთან მიერთება----
  SqlConnection con = new SqlConnection(@"Data Source=gtu-205A-08;Initial
Catalog=SeaEco;Integrated Security=True");
...
//... ლისტინგი_29.2 ---- Insert() ------
private void button1_Click(object sender, EventArgs e)
      {
          Form2 frm = new Form2(nID+1, null);
          //დამატება მოვლენის დასაჭერად Form2-დან---
          frm.UpdateListView += new
               EventHandler(frm_UpdateListView);
          frm.Show();
      }
  void frm_UpdateListView(object sender, EventArgs e)
        {
         //როგორც კი მონაცემები დაემატება Form2-ში,
        // მოვლენა განაახლებს ListView1-ს----
           ListViewLoad();
        }
```

- Delete მეთოდის შესაბამისი კოდი მოცემულია 29.3 ლისტინგში

```
   //.. ლისტინგი_29.3 ---- Delete
   private void button4_Click(object sender, EventArgs e)
        {
            SqlDataAdapter dass = new SqlDataAdapter();
            con.Open();
            DataTable dt = new DataTable();
            string deleete_r = listView1.SelectedItems[0].SubItems[0].Text;
```

```
        try
          {
             dass.DeleteCommand = new
          SqlCommand(@"DELETE FROM River WHERE riverID =
                @riverID", con);

      dass.DeleteCommand.Parameters.Add("@riverID",
          SqlDbType.Int).Value = int.Parse(deleete_r);
      dass.DeleteCommand.ExecuteNonQuery();
      con.Close();
          }
        catch (Exception ex)
          {
              MessageBox.Show(ex.Message);
          }
          ListViewLoad();
      }
```

29.10 ნახაზზე მოცემულია Update მეთოდის შედეგი, ანუ დავდექით მე-9 ჩანაწერზე და ავამოქმედეთ Update ღილაკი. ამ მოვლენამ აამუშავა Update() მეთოდი. შევიტანეთ მდინარე „ლიახვი 200 კმ“. კოდი ნაჩვენებია 29.4 ლისტინგში.



ნახ.29.10

```
// ლისტინგი_29.4 - -- Update() ьуЕщвш ---
private void button3_Click(object sender, EventArgs e)          {
            Form2 frm = new Form2(nID, listView1);
            frm.UpdateListView += new EventHandler(frm_UpdateListView);
            frm.Show();
        }
```

შეცვლილი ბაზის ხელახალი ჩატვირთვა ListView ფანჯარაში ხორციელდება შემდეგი კოდით (ლისტინგი 29.5).

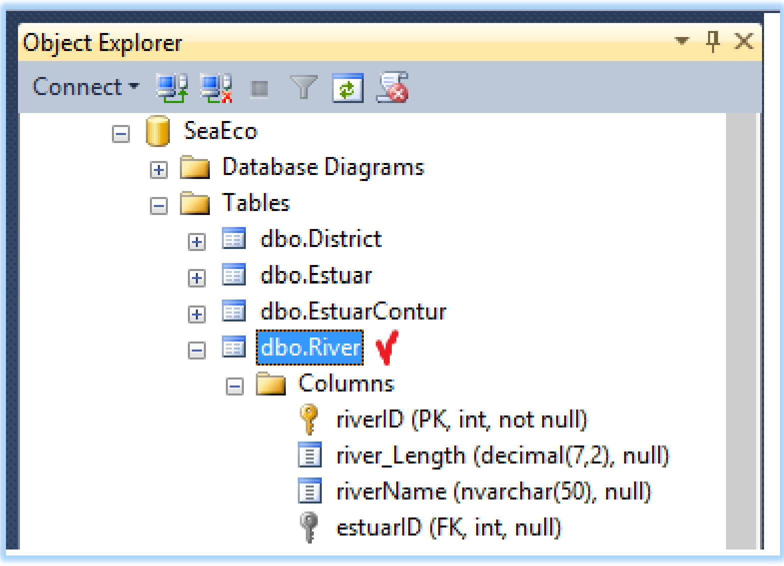
```
// -- ლისტინგი_29.5 ---- ---
void ListViewLoad()
```

```
{
  SqlDataAdapter da = new SqlDataAdapter("SELECT riverID,
    riverName, river_Length, estuarID FROM River", con);
  DataTable dt = new DataTable();
  // მონაცემების გადატანა ბაზიდან ცხრილში ---
   da.Fill(dt);
  // ListView-ს შევსება ცხრილიდან---
   listView1.Items.Clear();
   listView1.View = View.Details;
   listView1.GridLines = true;

   foreach (DataRow dr in dt.Rows)
   {
     ListViewItem lvi = new
           ListViewItem(dr["riverID"].ToString());
    lvi.SubItems.Add(dr["riverName"].ToString());
    lvi.SubItems.Add(dr["river_Length"].ToString());
    lvi.SubItems.Add(dr["estuarID"].ToString());
    listView1.Items.Add(lvi);
    // გლობალური ცვლადი (int), გამოცხადებულია დასაწისში---
    nID =int.Parse( dr["riverID"].ToString());
   }
}
```

29.2. მონაცემთა ბაზის განახლება ADO.NET დრაივერისა და DataGridView კლასის გამოყენებით

ამოცანა_3. მოცემულია Ms SQL Server მონაცემთა ბაზა (მაგალითად, „შავი ზღვის ეკოლოგიური სისტემა“), რომლის ერთ-ერთი ცხრილი (Table) არის River.dbo (მდინარეები). საჭიროა ავაგოთ C# პროექტი (მომხმარებლის ინტერფეისი), რომელიც მონაცემთა ბაზიდან ამოიღებს მდინარეების მონაცემებს DataGridView ცხრილში, შეძლებს Insert, Update და Delete ოპერაციების განხორციელებას. გამოყენებულ უნდა იქნას ADO.NET დრაივერის საშუალებები.

29.11 ნახაზზე ნაჩვენებია საწყისი მონაცემთა ბაზა, რომელიც Ms SQL Server 2012 ვერსიაშია რეალიზებული. (ა) შეესაბამება ბაზის ცხრილების იერარქიას და აქვე ჩანს River ცხრილის სტრუქტურაც, მონაცემთა შესაბამისი ტიპებით. (ბ)-ზე მოცემულია ჩანაწერები, რომელთა შეტანა მოხდა წინასწარ (თუმცა ჩვენი პროგრამისათვის არაა აუცილებელი მისი არსებობა. შესაძლებელია იგი შეივსოს ინტერფეისიდან).



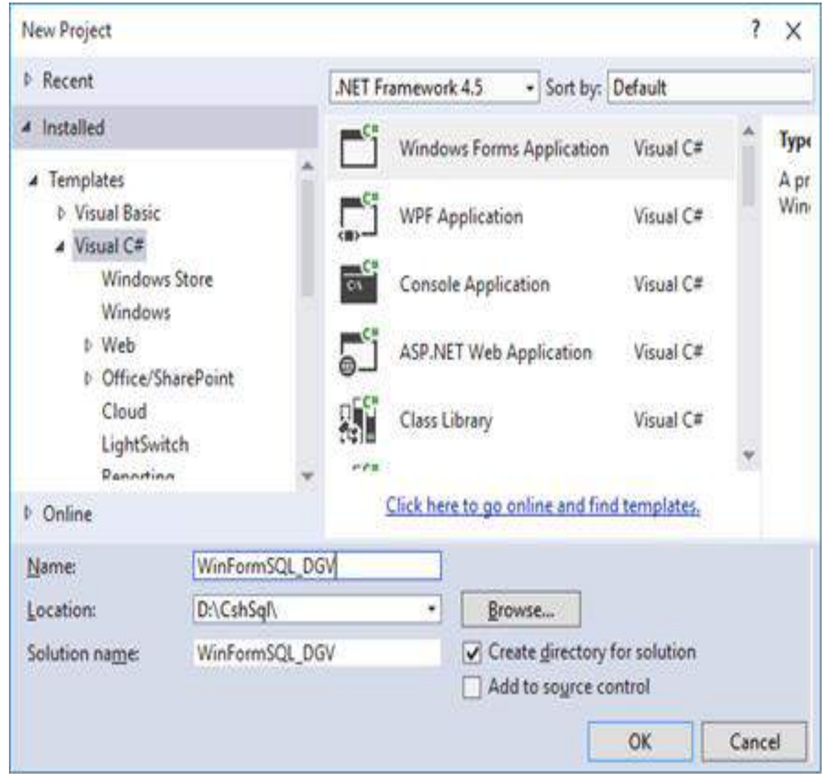
ნახ.29.11-ა. მონაცემთა ბაზა Ms SQL Server-ში

GTU-205A-08.SeaEco - dbo.River | GTU-205A-08.SeaEco - dbo.River

	riverID	river_Length	riverName	estuarID
▶	1	26,00	ჭოროხი (საქართველოს საზღვრებში)	1
	2	25,20	კინტრიში	2
	3	60,00	ნატანები	3
	4	108,00	სუფსა	4
	5	327,00	რიონი (სამხრეთ განშტოება)	5
	6	327,00	რიონი (ჩრდილოეთ განშტოება)	6
	7	150,00	ხობისწყალი	7
	8	213,00	ენგური	8
*	NULL	NULL	NULL	NULL

ნახ.29.11-ბ. River (მდინარეების) საწყისი ცხრილი
Ms SQL Server-ში

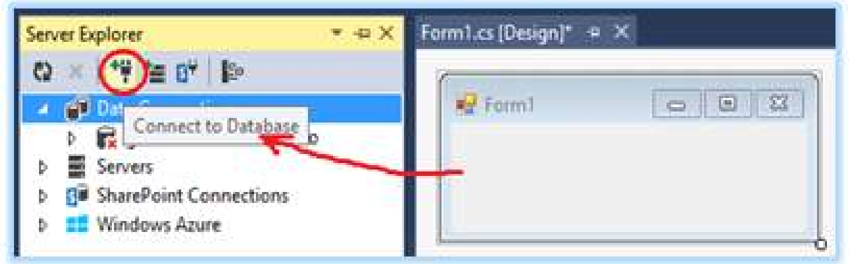
დავიწყოთ ახალი პროექტის აგება Ms Visual Studio.NET 2013/15 სამუშაო გარემოში. 29.12 ნახაზზე ნაჩვენებია პროექტის შექმნის პროცედურა.



ნახ.29.12. WinFormSQL_DGV პროექტის შექმნა

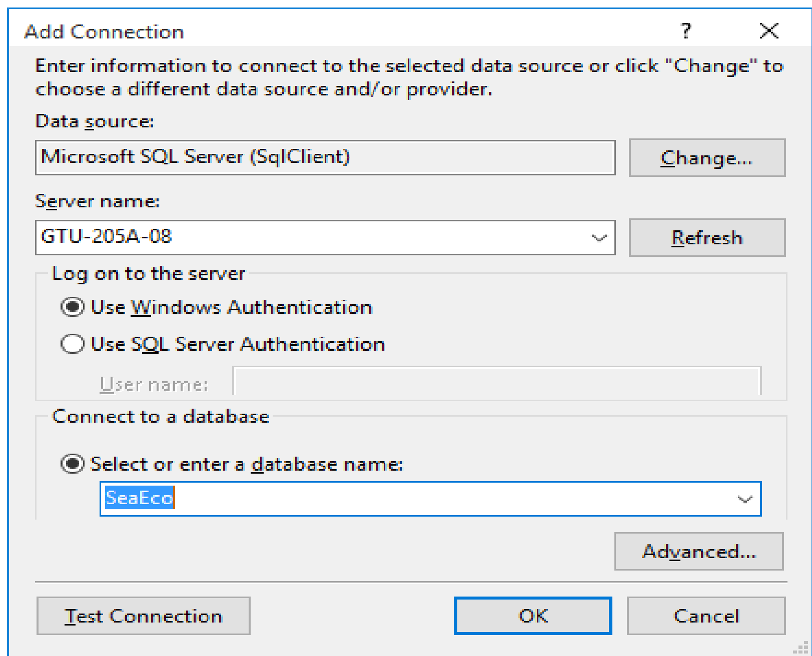
ვირჩევთ პროექტის სახელს (Name), მისი შენახვის ადგილს (Browse-ს დახმარებით) და Solutionname-ს. შემდეგ OK.

საჭიროა პროექტისთვის განვახორციელოთ მონაცემა ბაზის მიერთება (Connect to Database), რაც 29.13 ნახაზზეა ნაჩვენები.



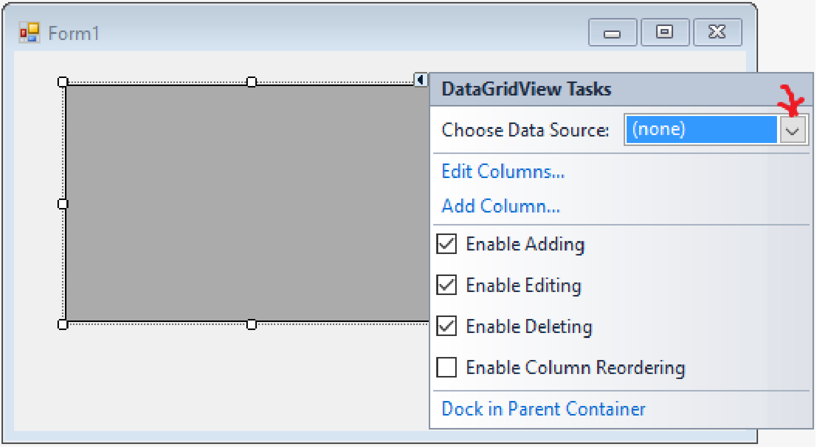
ნახ.29.13. Connect Database ამოქმედება

გამოიტანება ახალი ფანჯარა (ნახ.29.14), რომელშიც უნდა შეირჩეს შესაბამისი წყარო (Data Source), სერვერი (Server name) და მონაცემთა ბაზა (Database name).



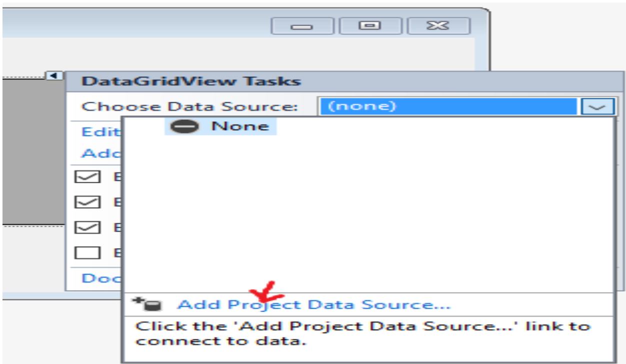
ნახ.29.14. მონაცემთა წყაროს, სერვერისა და ბაზის არჩევა

ინსტრუმენტების პანელიდან ფორმაზე გადავიტანოთ DataGridView ელემენტი და ზედა მარჯვენა კუთხე პატარა ისრით ავამოქმედოთ. მივიღებთ 29.15 ნახაზს.



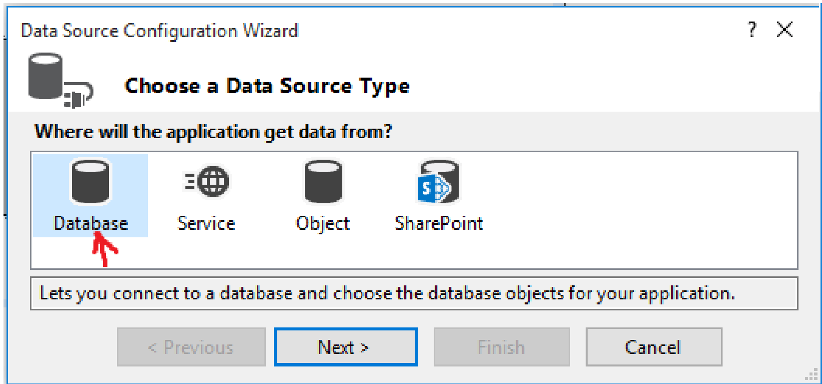
ნახ.29.1.5. პარამეტრების განსაზღვრა

ნახაზზე ჩანს, რომ ჩამატების, რედაქტირებისა და წაშლის ოპერაციები ნებადართულია (ჩეკბოქსები მონიშნულია). ავირჩიოთ Choose Data Source კომბობოქსის ღილაკი, მივიღებთ 29.16 ნახაზს.



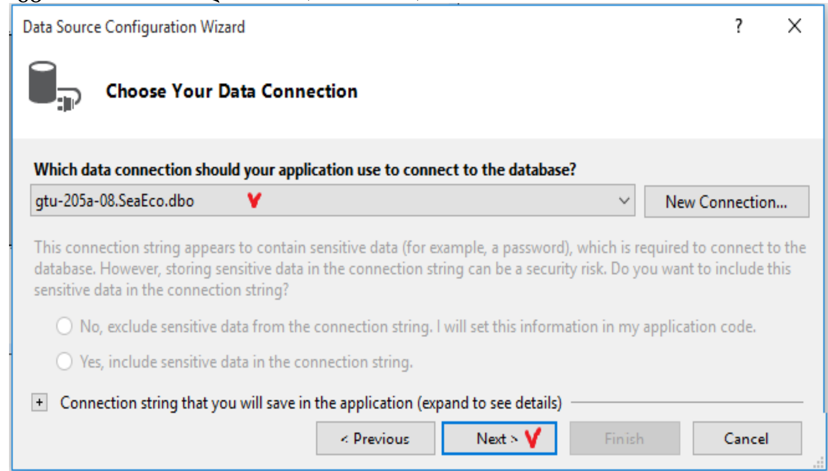
ნახ.29.16. მონაცემთა წყაროს პროექტის დამატება

ავამოქმედოთ Add Project Data Source და გადავიდეთ 29.17 ნახაზზე.



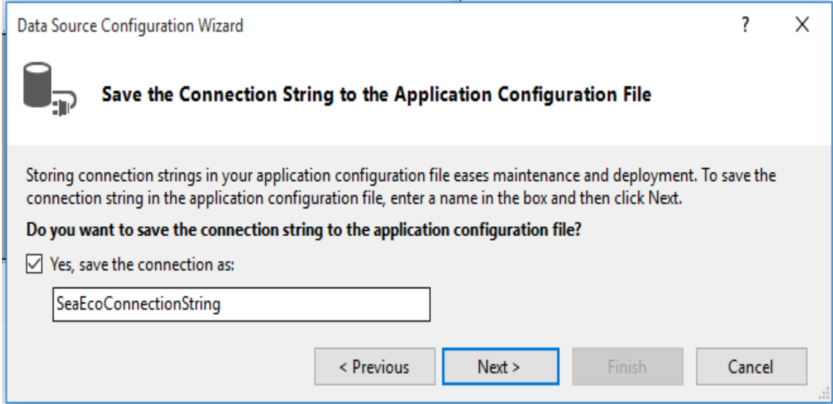
ნახ.29.17. მონაცემთა წყაროს ტიპის არჩევა

ვირჩევთ Database-ს და Next (ნახ.29.18).



ნახ.29.18. მონაცემთა Connection (მიერთების) შერჩევა

Connection პარამეტრი ყველა კომპიუტერს ექნება თავისი. მისი განსაზღვრა შესაძლებელია Server Explorer-იდან (ჩვენ შემთხვევაში იგი არის: GTU-205a-08.SeaEco.dbo). ბოლოს Next და გადავალთ 29.19 ნახაზზე.



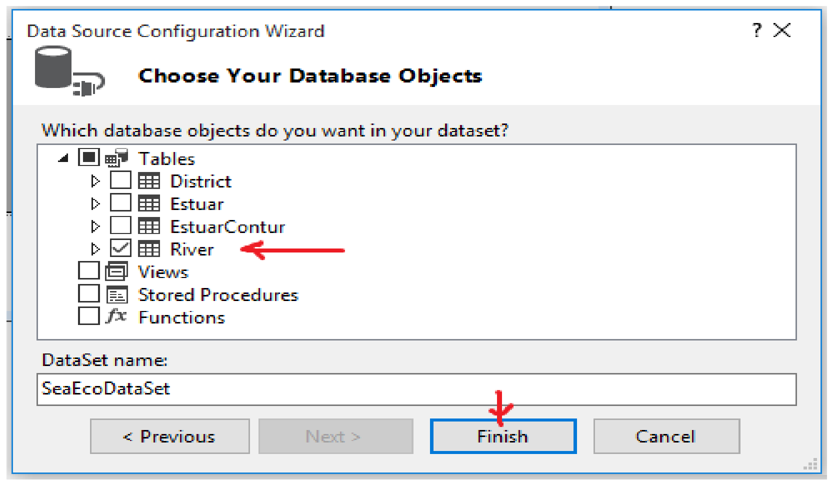
ნახ.29.19. Connection String-ის შენახვა აპლიკაციის კონფიგურაციის ფაილში

შემდეგ გამოიძახება მონაცემთა ბაზის ობიექტების არჩევის ფანჯარა (ნახ.29.20).

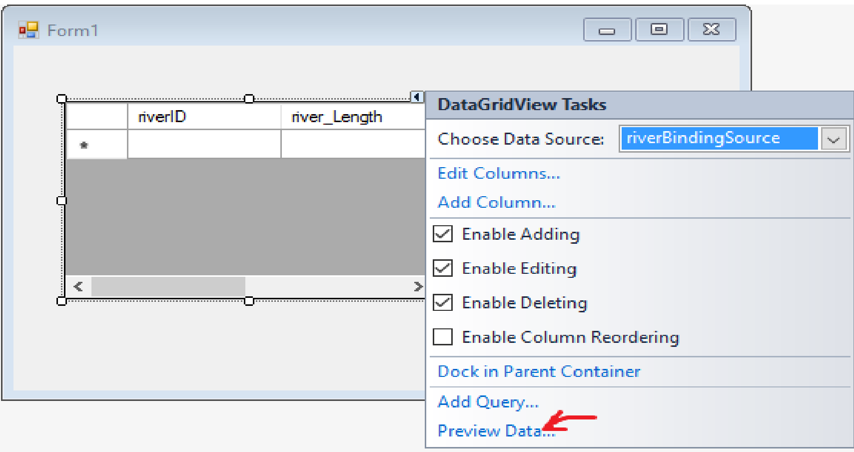
29.21 ნახაზზე ნაჩვენებია მონაცემთა წყაროს (Data Source) განსაზღვრის შედეგის მნიშვნელობა, ჩვენ შემთხვევაში იგი არის:

riverBindingSource.

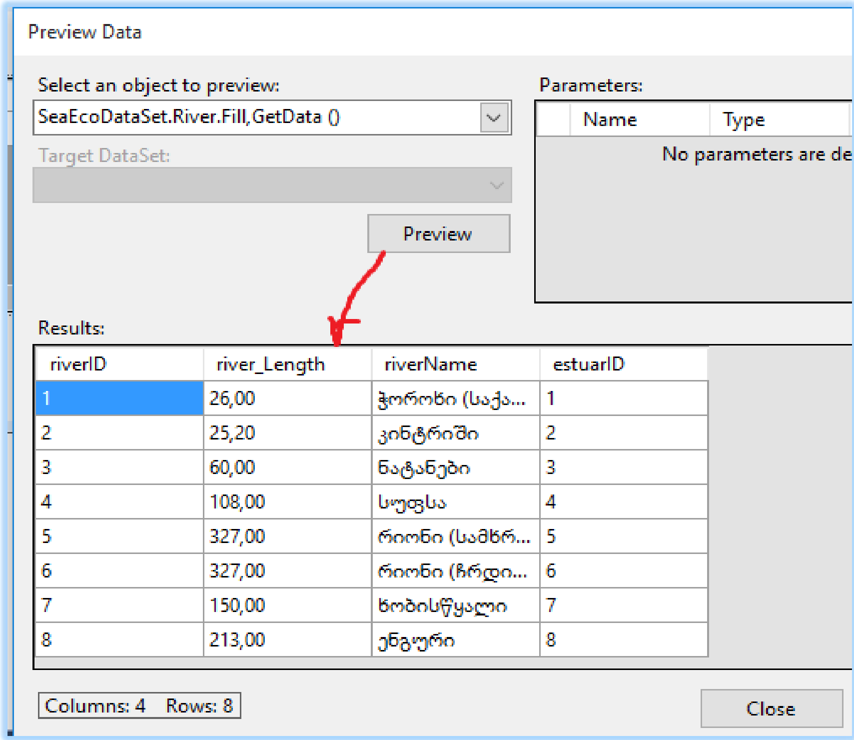
აქვე შეიძლება გამოვიყენოთ Preview Data ლინკი და დავათვალიეროთ წინასწარ მიერთებული ბაზის ცხრილის ჩანაწერები (ნახ.29.22).



ნახ.29.20. ობიექტების მონიშვნა (მაგალითად, River)

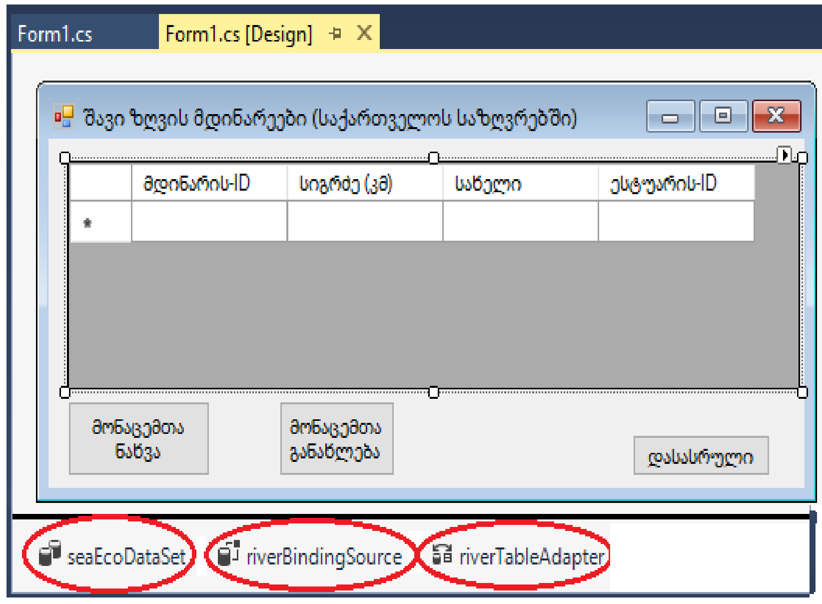


ნახ.29.21. Data Source შედეგი: “riverBindingSource”



ნახ.29.22. Preview Data ცხრილი

ბოლოს, Form1-ს Properties-ში შევუცვალოთ სახელი „შავი ზღვის მდინარეები (საქართველოს საზღვრებში)“, ინსტრუმენტების პანელიდან გადმოვიტანოთ სამი ღილაკი (Button1,2.3), დავარქვათ სახელები. მიიღება 29.23 ნახაზი.



ნახ.29.23. პროექტის ძირითადი ინტერფეისი

ახლა გადავიდეთ ღილაკების ფუნქციების დაპროგრამებაზე, ანუ უნდა განხორციელდეს SQL Server -შესაბამისი ბაზის ცხრილის ჩანაწერების დათვალიერება, ჩამატება, შეცვლა და წაშლა.

თავდაპირველად ჩავამატოთ სახელსივრცის სტრიქონი:

```
using System.Data.SqlClient;
```

შემდეეგ გამოვაცხადოთ გლობალური ცვლადები:

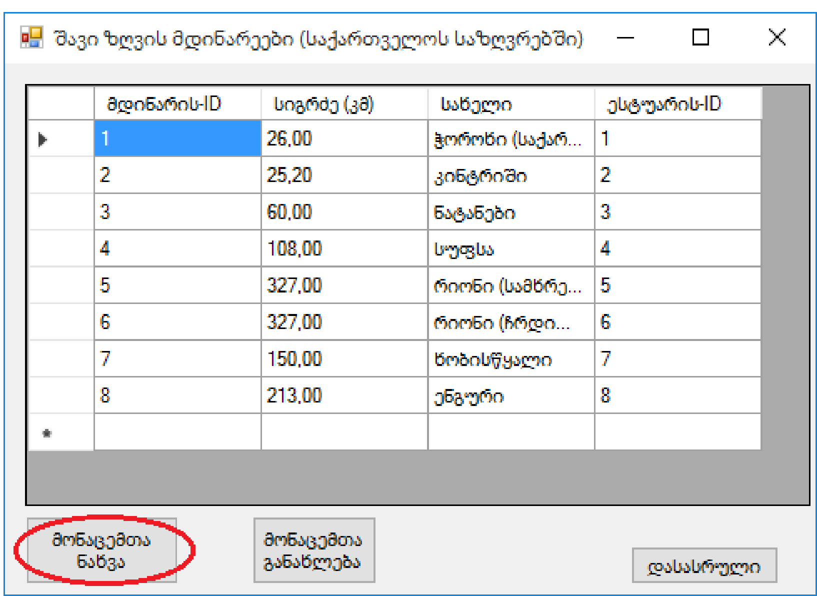
```
SqlDataAdapter sda;
SqlCommandBuilder scb;
DataTable dt;
```

„მონაცემთა ნახვის“ ღილაკისთვის (button1) გვექნება შემდეგი კოდი:

```
private void button1_Click(object sender, EventArgs e)
 {
   SqlConnection con = new SqlConnection("Data Source=
                GTU-205A-08;Initial Catalog=SeaEco;
                Integrated Security=True");
   sda = new SqlDataAdapter(@"SELECT riverID,
                    river_Length, riverName,
                    estuarID from River", con);
```

```
  dt = new DataTable();
  sda.Fill(dt);
  dataGridView1.DataSource = dt;
}
```

პროგრამის ამუშავებით, თუ მასში არაა შეცდომები, მიიღება 29.24 ნახაზი.



ნახ.29.24. „მონაცემთა ნახვის" (DataShow) ღილაკის ამოქმედებით მიღებული შედეგი

„მონაცემთა განახლების" ღილაკის კოდი (Insert, Update, Delete) ნაჩვენებია ქვემოთ:

```
private void button2_Click(object sender, EventArgs e)
  {
    scb = new SqlCommandBuilder(sda);
    sda.Update(dt);
  }
```

მთლიანი პროგრამის კოდი მოცემულია 29.6 ლისტინგში.

```
// --- ლისტინგი_29.6 --- Insert, Update, Delete for DataGridView_SQL----
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;
namespace WinFormSQL_DGV
{
  public partial class Form1 : Form
   {
      SqlDataAdapter sda;
      SqlCommandBuilder scb;
      DataTable dt;
      public Form1()
       {
          InitializeComponent();
       }
    private void Form1_Load(object sender, EventArgs e)
     {

     }
       // მონაცემთა ნახვა
    private void button1_Click(object sender, EventArgs e)
    {
      SqlConnection con = new SqlConnection("Data
           Source=GTU-205A-08;Initial Catalog=SeaEco;
                 Integrated Security=True");
      sda = new SqlDataAdapter(@"SELECT riverID,
                    river_Length, riverName, estuarID
                    from River", con);
      dt = new DataTable();
      sda.Fill(dt);
      dataGridView1.DataSource = dt;
    }

   // განახლება
   private void button2_Click(object sender, EventArgs e)
     {
```

```
            scb = new SqlCommandBuilder(sda);
            sda.Update(dt);
        }

        private void button3_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

29.25 ნახაზზე მოცემულია არსებულ ცხრილში ორი მნიშვნელობის (ესტუარისID) შეცვლა (ახალი რიცხვები ნაჩვენებია წრეში), შემდეგ 1-ელი და მე-2 ღილაკების ამოქმედებით ბაზაში ჩაიწერება ეს შეცვლილი მნიშვნელობები (შეიძლება დავხუროთ პროგრამა და თავიდან ავამუშავოთ).

შავი ზღვის მდინარეები (საქართველოს საზღვრებში)

მდინარის-ID	სიგრძე (კმ)	სახელი	ესტუარის-ID
1	26,00	ჭოროხი (საქარ...	5
2	25,20	კინტრიში	2
3	60,00	ნატანები	3
4	108,00	სუფსა	4
5	327,00	რიონი (სამხრე...	1
6	327,00	რიონი (ჩრდი...	6
7	150,00	ხობისწყალი	7
8	213,00	ენგური	8

მონაცემთა ნახვა 2 | მონაცემთა განახლება 1 | დასასრული

ნახ.29.25. Update ცვილების განხორციელება

შავი ზღვის მდინარეები (საქართველოს საზღვრებში)

მდინარის-ID	სიგრძე (კმ)	სახელი	ქსტუარის-ID
1	26,00	ჭოროხი (საქარ...	1
2	25,20	კინტრიში	2
3	60,00	ნატანები	3
4	108,00	სუფსა	4
5	327,00	რიონი (სამხრე...	5
6	327,00	რიონი (ჩრდი...	6
7	150,00	ხობისწყალი	7
8	213,00	ენგური	8
9	400,00	მტკვარი	5

მონაცემთა ნახვა | მონაცემთა განახლება | დასასრული

ნახ.29.26. Insert ოპერაციის განხორციელება

ბოლოს ნაჩვენებია სტრიქონის წაშლის (Delete) ოპერაცია. იგი ხორციელდება, მაგალითად, „**მდინარის-ID**" შესაბამისი სტრიქონის მონიშვნით და შემდეგ კომპიუტერის კლავიატურის Delete- ღილაკის ამოქმედებით (ნახ.29.27).

შავი ზღვის მდინარეები (საქართველოს საზღვრებში)

მდინარის-ID	სიგრძე (კმ)	სახელი	ქსტუარის-ID
1	26,00	ჭოროხი (საქარ...	1
2	25,20	კინტრიში	2
3	60,00	ნატანები	3
4	108,00	სუფსა	4
5	327,00	რიონი (სამხრე...	5
6	327,00	რიონი (ჩრდი...	6
7	150,00	ხობისწყალი	7
8	213,00	ენგური	8
9	400,00	მტკვარი	5

ღილაკი: DELETE

მონაცემთა ნახვა | მონაცემთა განახლება | დასასრული

ნახ.29.27. Delete ოპერაციის განხორციელება

29.3. შავის ზღვის კონცეპტუალური ORM მოდელის აგება ეკოლოგიური პარამეტრებისათვის

ჩვენი კვლევის საპრობლემო სფეროა შავი ზღვის ეკოლოგიური სისტემა, კერძოდ მისთვის მონაცემთა ბაზის დაპროექტება. საწყის ეტაპზე საჭიროა განისაზღვროს ის ობიექტები, რომლებიც აღწერს სინტაქსურად და სემანტიკურად ზღვის ეკოსისტემის ძირითად პარამეტრებს. ჩვენ მიერ ჩატარებული სისტემური ანალიზის საფუძველზე, რომელიც ხორციელდებოდა მიმდინარე საანგარიშო პერიოდში, გამოიკვეთა შემდეგი ობიექტები:

- ზღვა (SeaID, Name, Length_EastWest, Length_NorthSouth,Area, Water_volume, Average_depth, Max_depth);
- სანაპირო_ზოლი (...);
- **მდინარე** (RiverID, Name, Length_inGeorgia,);
- მდინარე (RiverID, მდინარის_დასახელება, წყალშემკრები_აუზის_ფართობი, კმ^2, აბსოლუტური_ნიშნული, მ, მდინარის_სიგრძე, კმ, საშუალო_ქანობი, i, აუზის_საშუალო_სიმაღლე_მონაკვეთზე, მ, ჩამონადენის_საშუალო_მოდული, ლ/წმ.კმ^2, საშუალო წლიური ხარჯი მ^3/წმ);
- **ესტუარი** (EstuarID, RiverID, CoordGPSx, CoordGPSy, Area,);
- **მოწყვლადი_უბანი** (Vulnerable_districtsID, CoordGPSx, CoordGPSy, Area, T1/T2, pH, TDS);
- GPS_კოორდინატები (CoordGPSx, CoordGPSy);
- **სენსიტიური_უბანი** (SensitiveAreasID, CoordGPSx, CoordGPSy)
- **უბანი** (DistrictID, Name, CoordGPSx, CoordGPSy, Area, T1/T2, pH, TDS);
- წყლის_სინჯის_ფაქტორები (WaterTestID, WaterT1, AirT2, Water_acidityPH, WaterSalinityTDS);
- ეკოლოგიური_პარამეტრი (...);
- რაოდენობრივი_მახასიათებელი (...);
- თვისობრივი_მახასიათებელი (...);
- შავიზღვის_ეკოლოგიური_პრობლემები (...);
- ეკო_უსაფრთხოების_ღონისძიება (ActionID, Name, DateBegin, DateEnd, …);
- საზღვაო_პორტი (PortID, DistrictsName, CoordGPSx, CoordGPSy);

და სხვ.

მონაცემთა ბაზის დაპროექტება უნდა განვახორციელოთ ობიექტროლური მოდელირების ინსტრუმენტისა და მისი პრინციპების საფუძველზე [1,2]. ინსტრუმენტის სახით ვიყენებთ Natural ORM Architect პაკეტს, რომელიც თავსებადია Visual Studio.NET Framework ინტეგრირებულ სისტემასთან [1].

კონცეპტუალური მოდელი (ORM) ან სქემა არის საპრობლემო სფეროს ძირითად ტერმინთა ერთობლიობა და მათ შორის კავშირები, რომლებიც ასახავს საკვლევი სფეროს ბიზნეს პროცესებს და ბიზნეს-წესებს. იგი თეორიულად ეფუძნება კატეგორიალური მიდგომის (ენის გრამატიკული წესები) მათემატიკური ლოგიკის (ალგებრის) ერთობლივ გამოყენებას [3].

ასეთი მიდგომა ჩადებულია NORMA-ინსტრუმენტში, რომელიც დამკვეთ-მომხმარებლის ცოდნას დასაპროექტებელი ობიექტების შესახებ გადაიტანს ე.წ. ობიეტების, მათი თვისებების და პრედიკატების (ბინარული,... , n-არული) სახით.

ობიექტების აღწერა მომხმარებლის მიერ ხდება NORMA პაკეტის სამუშაო ინტერფეისით და შეიტანება ჯერ ერთი ობიექტი, შემდეგ მეორე და ა.შ.

ბოლოს თვით NORMA-სისტემა გვაძლევს ინტეგრირებულ კონცეპტუალურ მოდელს, რომელიც 29.30 ნახაზზეა ნაჩვენები.

ჩვენ საილუსტრაციოდ აღვწერეთ სამი ობიექტი: „მდინარე“ (River), „ესტუარი“ (Estuar) და „უბანი“ (District). მათ შორის კავშირები აგებულია „has“ („is”, „works” და სხვ.) პრედიკატებით.

პრედიკატები არის ფაქტები, მაგალითად:

f1: River has RiverName

f2: River has RiverLength

f3: River has Estuar

f4: Estuar has River

. . .

f15: District has Category

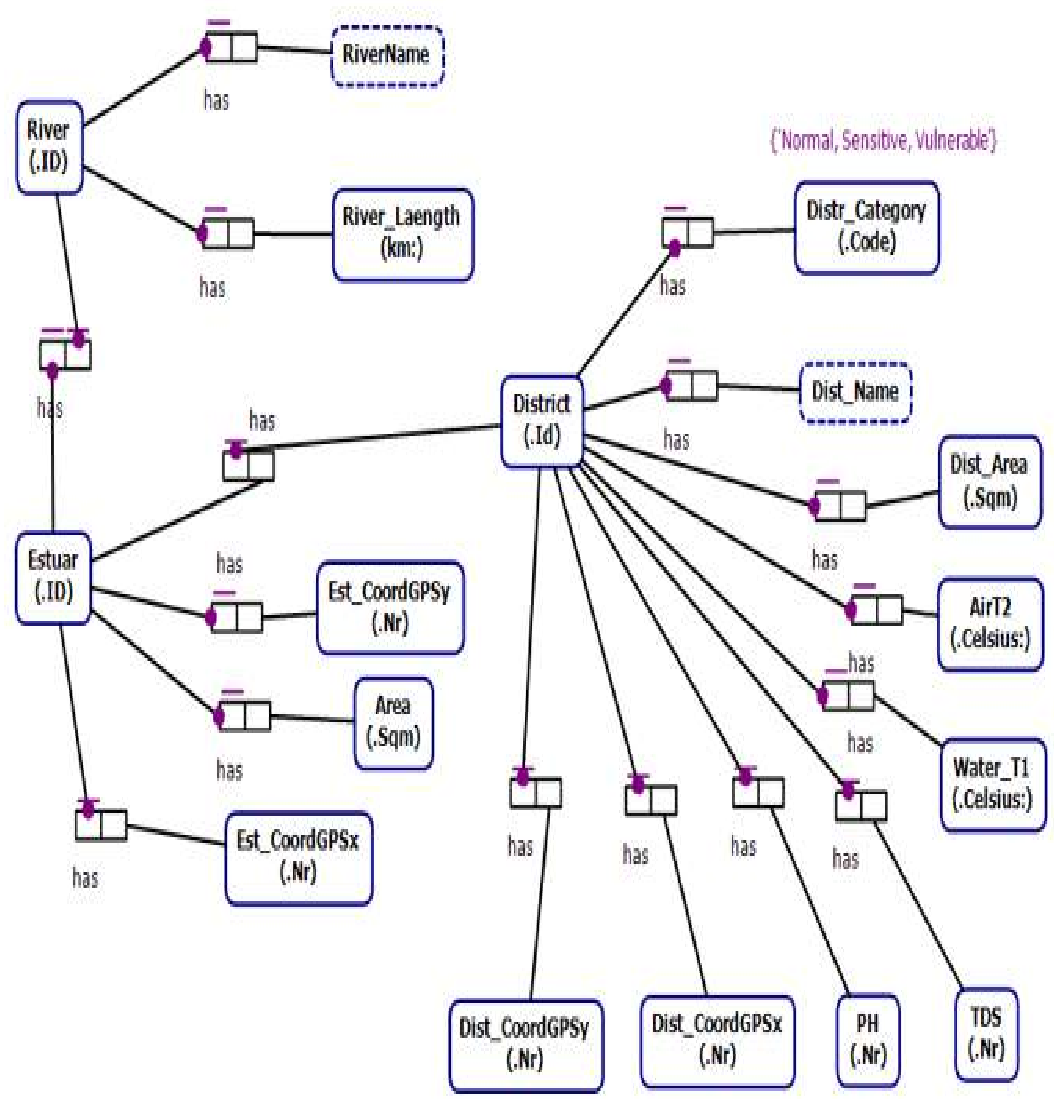
f16: District_Category is Normal or Sensitive or Vulnerable

და ა.შ.

ობიექტს „უბანი“ აქვს პარამეტრი „უბნის_კატეგორია“, რომელიც არის მნიშვნელობა სიმრავლიდან {ნორმალური, სენსიტიური, მოწყვლადი}. თუ რომელი იქნება კონკრეტული უბანი, დამოკიდებულია მისი ეკოლოგიური პარამეტრების მნიშვნელობებზე.

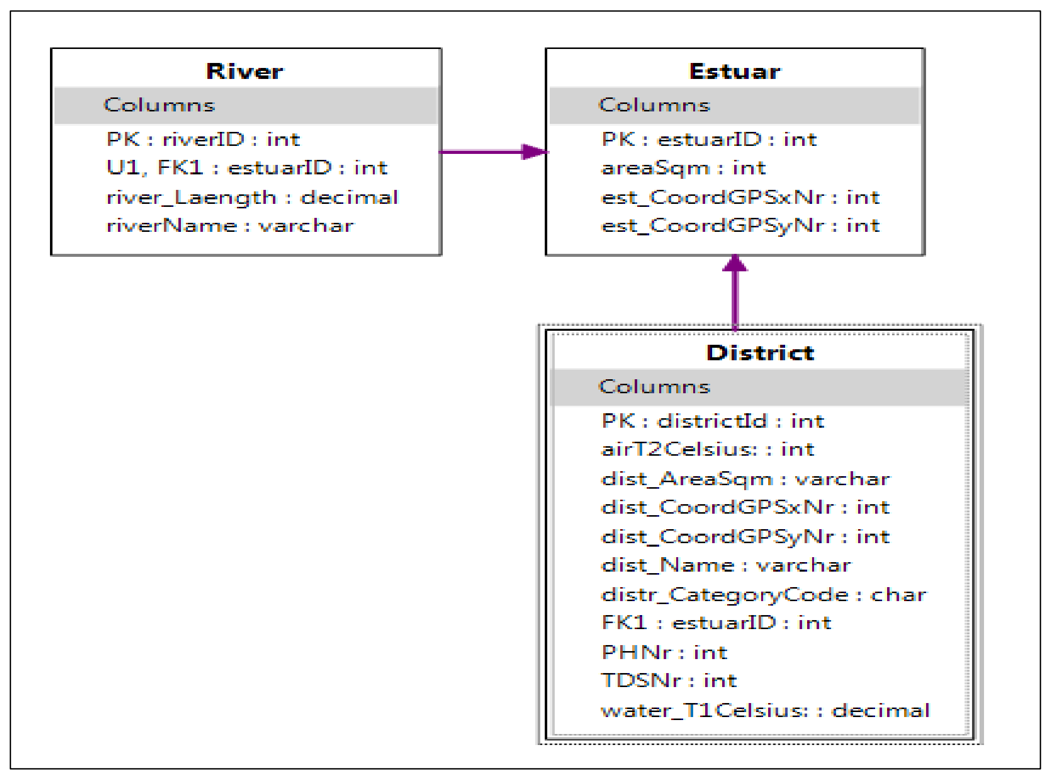
შეიძლება ითქვას, რომ ORM-მოდელირების ინსტრუმენტის გამოყენებით პირველი დონის კონცეპტუალური სქემის აგება შეუძლია არაპროგრამისტ (მონაცემთა ბაზების აგების არმცოდნე) მომხმარებელსაც. მან იცის საპრობლემო სფეროს არსი, ამოცანები, ფუნქციები და ამიტომ, იგი, შედარებით მცირე კონსულტაციის შემდეგ, NORMA გარემოში ადვილად საქმიანობს - გადააქვს თავისი ცოდნა კომპიუტერში. შედეგად მიიღება მონაცემთა ბაზის ORM მოდელი (ნახ.29.30).

რა თქმა უნდა, შესაძლებელია შედეგში იყოს უზუსტობები, რომლებიც ქსელის მოდიფიკაციის რეჟიმში ადვილად სწორდება თვით მომხმარებლის მიერ მანამ, სანამ არ მიიღება საბოლოო მომხმარებლისათვის მისაღები კონცეპტუალური მოდელი.

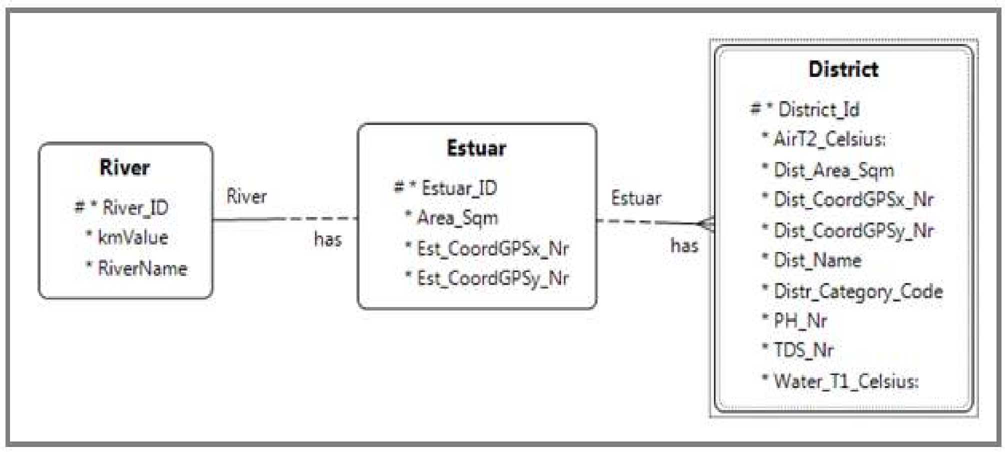


ნახ.29.30. შავი ზღვის ეკოლოგიური ბაზის ORM სქემის ფრაგმენტი

შემდეგი ეტაპი ეხება ORM მოდელის (კონცეპტუალური სქემის) საფუძველზე არსთა დამოკიდებულების მოდელის (Entity-Relationship Model), ანუ ERM მეორე დონის კონცეპტუალური სქემის დამუშავებას (პირველის საფუძველზე). 29.31 ნახაზზე ნაჩვენებია კლასიკური ვარიანტი ამ სქემისა, რომელიც მივიღეთ უშუალოდ Visual Studio.NET გარემოში ავტომატურად NORMA პაკეტიდან. 29.32 ნახაზზე მოცემულია ალტერნატიული ვარიანტი - ბარკერის მოდელი (მას აქტიურად იყენებს ფირმა Oracle) [4]



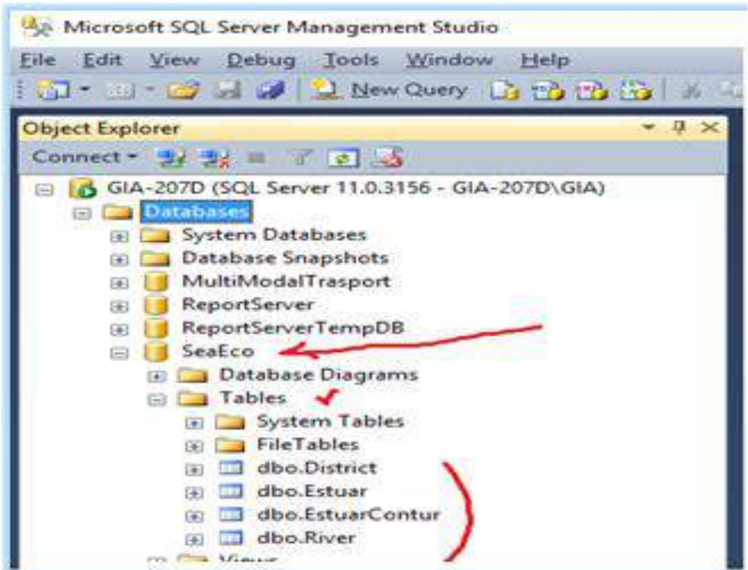
ნახ.29.31. ERM სქემის ფრაგმენტი (კლასიკური მოდელი)



ნახ.29.32. ბარკერის კონცეპტუალური მოდელი

29.4. შავი ზღის ეკოლოგიური პარამეტრების Ms SQL Server ბაზა

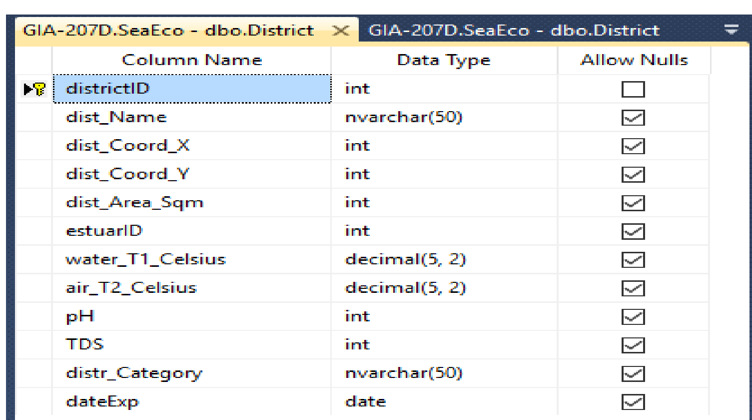
29.33 ნახაზზე ნაჩვენებია ჩვენ მიერ MsSQL Server პაკეტით აგებული მონაცემთა რელაციური ბაზა „SeaEcfo".



ნახ.29.33. MsSQLServer-სამუშაო გარემო

ბაზის დემოვერსია შედგება ოთხი ცხრილისგან (Tables). ეს ცხრილებია:

- District.dbo (ნახ.29.34),
- Estuar.dbo (ნახ.29.35),
- EstuarContur.dbo (ნახ.29.36) და
- River.dbo (ნახ.29.37).



GIA-207D.SeaEco - dbo.District

Column Name	Data Type	Allow Nulls
districtID	int	☐
dist_Name	nvarchar(50)	☑
dist_Coord_X	int	☑
dist_Coord_Y	int	☑
dist_Area_Sqm	int	☑
estuarID	int	☑
water_T1_Celsius	decimal(5, 2)	☑
air_T2_Celsius	decimal(5, 2)	☑
pH	int	☑
TDS	int	☑
distr_Category	nvarchar(50)	☑
dateExp	date	☑

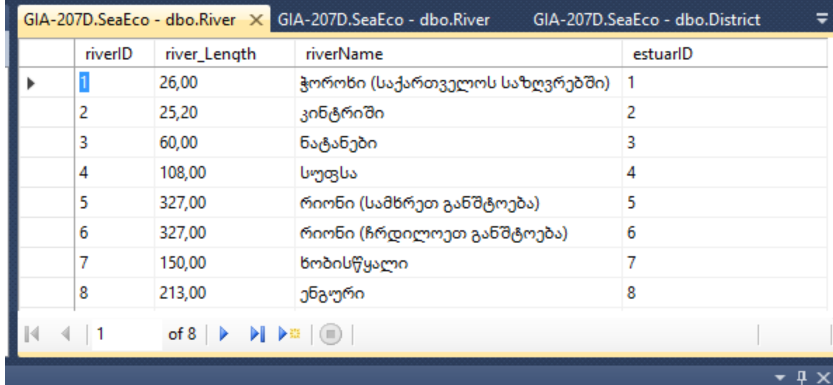
ნახ.29.34. უბნის ცხრილის სტრუქტურა

GIA-207D.SeaEco - dbo.District | GIA-207D.SeaEco - ER_Model SQL Server*

districtID	dist_Name	dist_Coord_X	dist_Coord_Y
1	სარფი	41526956	41548731
2	კვარიათი_1	41545542	41561587
3	კვარიათი_2	41554651	41563841
4	გონიო	41574588	41565589
5	ჭოროხი-მარცხენა	41596952	41569943
6	ჭოროხი-მარჯვენა	41607866	41577288
7	ადლია	41614371	41583944
8	ბათუმი (დელფინარიუმთან)	41649103	41621114
9	ბათუმი (დასაწყისი)	41650823	41666129
10	ბათუმი (ბუნზე)	41662161	41678955
11	მახინჯაური (რკინიგზის სადგურთან)	41677322	41694925
12	. . .	41723714	41727073
32	ყულევი	42259918	41637102
33	ანაკლია (რეპერთან)	42382543	41577101
34	ანაკლია (სასტუმროსთან)	42382744	41563028
35	ანაკლია (მდ. ენგურის მარცხენა ნაპირი)	42389302	41560674

1 of 35

ნახ.29.35. უბნების ცხრილი


GIA-207D.SeaEco - dbo.River | GIA-207D.SeaEco - dbo.River | GIA-207D.SeaEco - dbo.District

riverID	river_Length	riverName	estuarID
1	26,00	ჭოროხი (საქართველოს საზღვრებში)	1
2	25,20	კინტრიში	2
3	60,00	ნატანები	3
4	108,00	სუფსა	4
5	327,00	რიონი (სამხრეთ განშტოება)	5
6	327,00	რიონი (ჩრდილოეთ განშტოება)	6
7	150,00	ხობისწყალი	7
8	213,00	ენგური	8

1 of 8

ნახ.29.36. მდინარეების ცხრილი

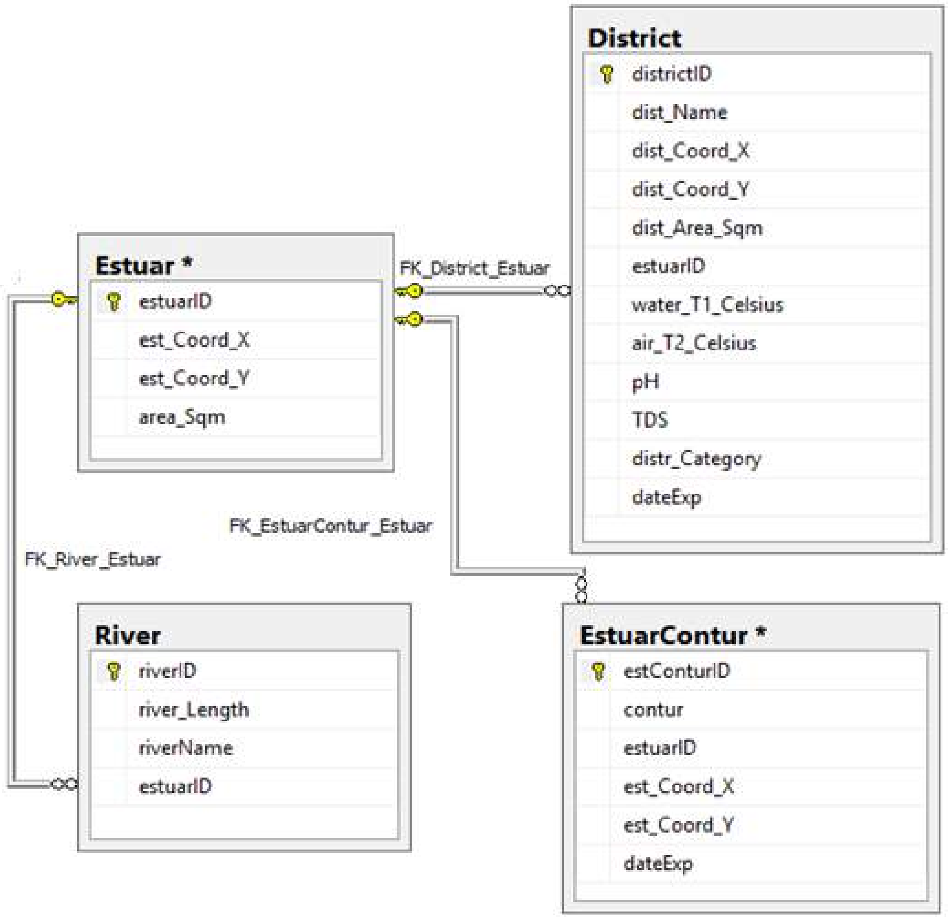
GIA-207D.SeaEco - dbo.Estuar | GIA-207D.SeaEco - dbo.Estuar*

estuarID	est_Coord_X	est_Coord_Y	area_Sqm
1	41602473	41571921	5465
2	41797895	41766410	861
3	41907237	41769259	1130
4	42017175	41752815	1488
5	42140385	41655681	20390
6	42172443	41645811	14551
7	42272625	41634087	1009
8	42386917	41564536	1379

1 of 8

ნახ.29.37. ესტუარების ცხრილი

29.38 ნახაზზე წარმოდგენილია აგებული მონაცემთა ბაზის კონცეპტუალური მოდელი რეალური ცხრილებით და ბაზის ატრიბუტებით. მითითებულია პირველადი (PrimaryKey) და მეორეული (ForeignKey) გასაღებები, რითაც განხორციელებულია კავშირები მონაცემებს შორის [5].



ნახ.29.38. ზღვის ეკოსისტემის სადემონსტრაციო ბაზის
სტრუქტურის ფრაგმენტი

წიგნის მე-3 დანართში მოცემულია საქართველოს შავი ზღვის აკვატორიაში მდინარათა ესტუარების ეკოლოგიური მდგომარეობის მონიტორინგის სისტემის პროექტის ერთი ნაწილი, რომელიც წყლის სინჯების ექსპერტთა ვებ-სერვისის რეალიზებას ეხება.

XXX თავი
WPF აპლიკაციის აგება საპრობლემო სფეროში „მარკეტინგი“

განიხილება მარკეტინგული პროცესების მენეჯმენტის, მისი მოდელური და პროგრამული რეალიზაციის საკითხები ბიზნესპროცესების მოდელირების ნოტაციის (BPMN), პეტრის ფერადი ქსელების (CPN) და კლიენტ-სერვერ არქიტექტურის დაპროგრამების ახალი ტექნოლოგიების (WPF, Workflow, WCF) ბაზაზე [10,16-18,43,60].

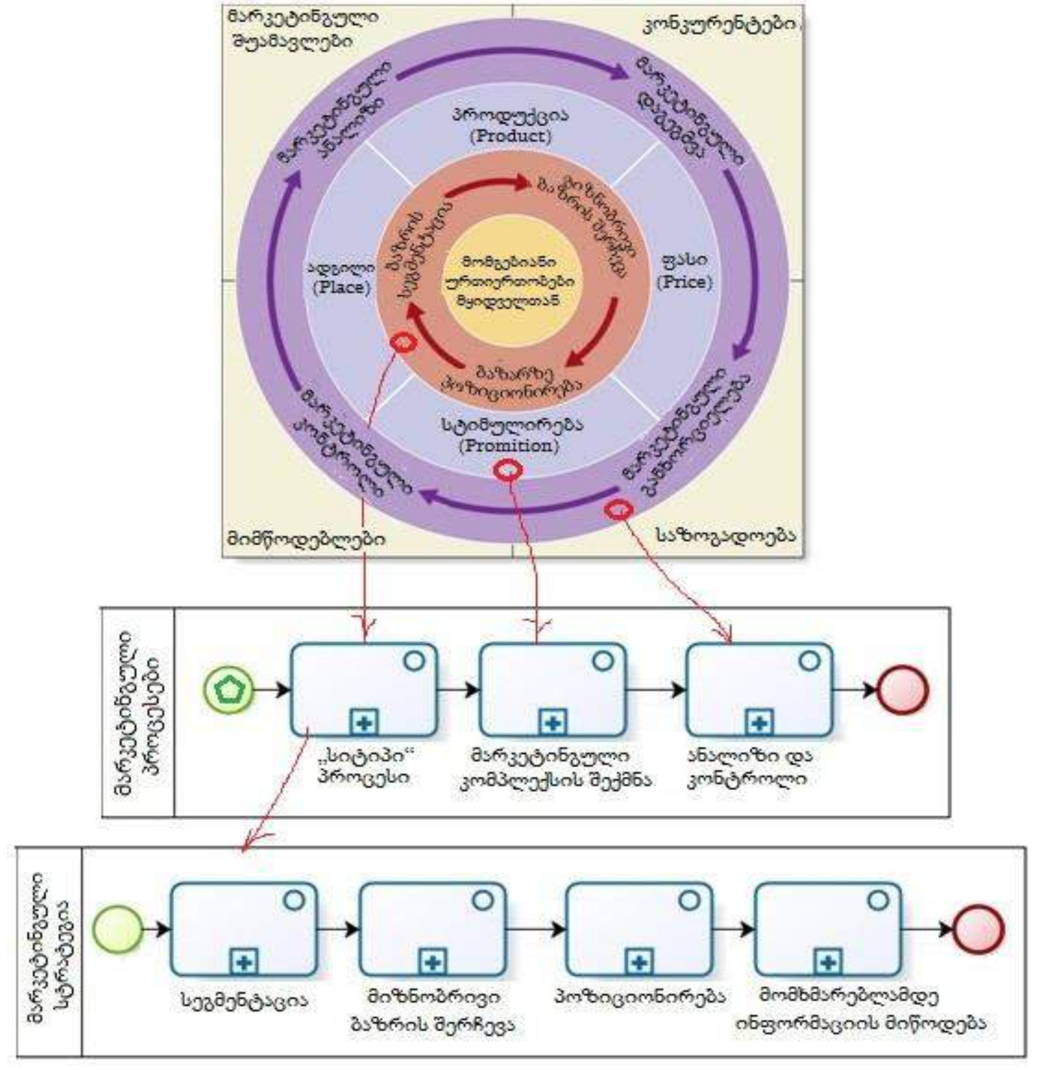
ძირითადი საწარმოო-ეკონომიკური მაჩვენებლები, როგორიცაა პროდუქციის წარმოების, რეალიზაციის, თვითღირებულების გეგმის შესრულება, რენტაბელობა, პროდუქციის ხარისხი, შრომის ნაყოფიერება, საშუალო ხელფასი და ა.შ. წარმოადგენს იმ მონაცემებს, რომლებიც მოქცეულია ხელმძღვანელობის კონტროლის ქვეშ და რომელთა განსაზღვრული მნიშვნელობების მისაღწევად წარიმართება მათი ყოველდღიური საქმიანობა. მენეჯერი თავის ფუნქციებს კარგად შეასრულებს იმ შემთხვევაში, თუ იგი სრულად ფლობს სიტუაციას, თუ აქვს უტყუარი ინფორმაცია საწარმოო საქმიანობის შესახებ.

საქმიანი პროცესების ავტომატიზაციის ტექნოლოგია (workflow) არის საწარმოს მართვის პროცესების პროგრამული მხარდაჭერა. იგი აერთიანებს რამდენიმე საინფორმაციო ტექნოლოგიას, როგორიცაა ელექტრონული ფოსტა, პროექტების მართვის სისტემა, მონაცემთა ბაზების მართვის სისტემა, ობიექტორიენტირებული პროგრამირება და CASE-ტექნოლოგიები [46,80].

30.1. მარკეტინგის მენეჯმენტის ბიზნესპროცესები და მათი მოდელირების ნოტაცია

კორპორაციის მენეჯმენტის სფეროში მნიშვნელოვანი ადგილი უკავია მარკეტინგს, როგორც ბიზნესის ფილოსოფიის ცნებას [142]. მარკეტინგი, როგორც მმართველობითი კონცეფცია, ჩვენს ქვეყანაში, საბაზრო რეფორმების განვითარებასთან, ერთად საკმაოდ ფართოდ გავრცელდა ეკონომიკის სხვადასხვა დარგში. მისი ერთ-ერთი ძირითადი მიმართულებაა ბაზრის მოთხოვნილების შესაბამისი პროდუქტის შეთავაზება.

მარკეტინგის როლი და საქმიანი ფუნქციურობის ზოგადი მოდელი კარგადაა წარმოდგენილი ფილიპ კოტლერის ფუნდამენტურ ნაშრომებში. 30.1 ნახაზზე მოცემულია მის მიერ შემოთავაზებული მოდელი, კერძოდ, მარკეტინგული სტრატეგიის და მარკეტინგული კომპლექსის მართვის საკითხები [43]. ესაა მარკეტინგული მენეჯმე-ნტის მთავარი საქმიანობის შემაჯამებელი სურათი. მყიდველი მოთავსებულია ცენტრში. სისტემის მიზანია ძლიერი და მომგებიანი ურთიერთობების ჩამოყალიბების ხელშეწყობა მყიდველებთან.



ნახ.30.1. მარკეტინგის კოტლერისეული მოდელის
ასახვა BPMN ნოტაციაში

შემდეგ მოდის მარკეტინგული სტრატეგია – მარკეტინგული ლოგიკა, რომლის საშუალებით კომპანია ამ მომგებიანი ურთიერთობების ჩამოყალიბებას იმედოვნებს. ბაზრის სეგმენტაციის, მიზნობრივი ბაზრის შერჩევისა და პოზიციონირების საშუალებით კომპანია წყვეტს, თუ რომელ მყიდველს როგორ მოემსახურება. იგი მთლიანი ბაზრის იდენტიფიცირებას ახდენს, შემდეგ მას სეგმენტებად ყოფს. მათ შორის ყველაზე პერსპექტიულს ირჩევს და მის მომსახურებასა და დაკმაყოფილებაზე ამახვილებს ყურადღებას.

მარკეტინგული სტრატეგიის ხელმძღვანელობით იგი ქმნის მარკეტინგულ კომპლექსს, რომელიც შედგება კომპანიის მიერ კონტროლირებადი ფაქტორებისგან: პროდუქტი, ფასი, ადგილი და სტიმულირება.

საუკეთესო მარკეტინგული სტრატეგიისა და კომპლექსის მისაღწევად კომპანია ჩაბმულია მარკეტინგული ანალიზის, დაგეგმვის, რეალიზაციისა და კონტროლის საქმიანობებში. ეს კი კომპანიას საშუალებას აძლევს თვალყური ადევნოს საბაზრო გარემოს და მის მონაწილეებს. ნაშრომში მოკლედ განიხილება მარკეტინგული ბიზნესპროცესების ასახვის საკითხები BPMN – ახალი სტანდარტული გრაფო-ანალიზური ენის საშუალებით [20,43].

ბიზნესპროცესების მოდელირების ინიციატივა (BPMI), რომელიც ბიზნეს-პროცესების მოდელირების საზოგადოების დიდი სეგმენტია, მივიდა კონსესუსამდე და წარმოადგინა BPMN-ენა, როგორც ბიზნეს პროცესების მოდელირების მსოფლიო სტანდარტი [143,144]. BPMN–ის შექმნა იყო მნიშვნელოვანი ნაბიჯი ფრაგმენტულობის შესამცირებლად, რომელიც არსებობს ბიზნესპორცესების მოდელირების მრავალ ინსტრუმენტთან და ნოტაციასთან. BPMN 2.0 ვერსია გამოქვეყნდა 2011 წლის იანვარში.

BPMN-ის ძირითადი მიზანი და დანიშნულებაა იყოს ადვილად გასაგები ყველა ბიზნესმომხმარებლისთვის, დაწყებული ბიზნესანალიტიკოსებიდან, რომლებიც ქმნიან პროცესების დაწყებით ნახაზებს, ასევე ტექნიკური დეველოპერებისთვის, რომლებიც ნერგავენ ისეთ ტექნოლოგიას, რომელმაც უნდა შეასრულოს ეს პროცესები და დამთავრებული ბიზნესგარემოს იმ წარმომადგენლებით, რომლებმაც უნდა განახორციელონ ამ პროცესების მართვა და მონიტორინგი. BPMN ასევე მხარდაჭერილია შიგა მოდელით, რომლითაც შესაძლებელია BPEL4WS-ის განხორციელებადი გენერაცია. ამგვარად, BPMN ქმნის სტანდარტიზებულ ხიდს ბიზნესპროცესების მონახაზს, გეგმასა და ამ პრცესების პროგრამულ რეალიზაციას შორის. BPMN – ბიზნესპროცესების დიაგრამაა, რომელიც დაფუძნებულია ე. წ. „flowcharting" ტექნიკაზე. ის არის შექმნილი ბიზნესპროცესების ოპერაციის გრაფიკული მოდელების ასაგებად.

ბიზნესპროცესების დიაგრამა (BPD) შედგება გრაფიკული ელემენტებისგან, რომლითაც შესაძლებელია მარტივად შევქმნათ დიაგრამა, რომელიც იქნება ადვილად გასაგები ბიზნესანალიტიკოსებისათვის. ელემენტები ისეა არჩეული, რომ იყოს ადვილად გარჩევადი და იყენებენ ისეთ ფიგურებს, რომლთა უმრავლესობა მათთვის ნაცნობია. მაგალითად, ქმედებები მართკუთხედებია და გადაწყვეტილებები – რომბისებრი და ა.შ.

შეგვიძლია ხაზი გავუსვათ, რომ BPMN-ის განვითარების ერთ-ერთი სტიმული იყო ისეთი მარტივი მექანიზმის ჩამოყალიბება ბიზნესპორცესების მოდელირებისათვის, რომ შესაძლებელი ყოფილყო ბიზნესპროცესების სირთულის მართვა. ამ მოთხოვნების დასა-კმაყოფილებლად საჭიროა ნოტაციის გრაფიკული ასპექტების ორგანიზება სპეციფიკურ კატეგორიებში. ეს ქმნის ნოტაციის კატეგორიების მცირე ნაკრებს ისე, რომ ბიზნეს–

პროცესების დიაგრამის (BPD) წამკითხველმა, მარტივად შეიცნოს ელემენტის ძირითადი ტიპები და გაიგოს დიაგრამა.

არსებობს ელემენტების ოთხი ძირითადი კატეგორია:

- ნაკადის ობიექტები (Flow Objects);
- შემაერთებელი (დამაკავშირებელი) ობიექტები (Connecting Objects);
- პროცესის მონაწილენი (Swimlanes) და
- არტიფაქტები (ნიმუშები - Artifacts).

ნაკადის ობიექტები ძირითადი გრაფიკული ელემენტებია, რომლებიც განსაზღვრავს ბიზნესპროცესების ყოფაქცევას. არსებობს სამი სახის ნაკადის ობიექტი:

- ხდომილებები (Events);
- ქმედებები (Activities) და
- გასასვლელები (Gateways).

ნაკადის ობიექტების ერთმანეთთან ან სხვა ინფორმაციასთან დაკავშირების (შეერთების) სამი გზა არსებობს. ასეთი დამაკავშირებელი (შემაერთებელი) ობიექტებია:

- მიმდევრობითი შესრულების ნაკადი (Sequence Flow);
- შეტყობინების ნაკადი (Message Flow) და
- გაერთიანება (Association);

პროცესის მონაწილეების საშუალებით მოდელირების ძირითადი ელემენტების დაჯგუფების ორი გზაა: გუბეები (Pools) და ბილიკები (Lanes);

არტიფაქტები (ნიმუშები) გამოიყენება პროცესის შესახებ დამატებითი ინფორმაციის უზრუნველსაყოფად. არსებობს სამი სტანდარტიზებული არტიფაქტი, მაგრამ მოდელირების ინსტრუმენტებს შეუძლია დაამატოს რამდენიც საჭიროა იმდენი არტიფაქტი. BPMN-ს აქვს დამატებითი შესაძლებლობები სტანდარტიზება გაუკეთოს არტიფაქტების დიდ სიმრავლეს საერთო გამოყენებისათვის. არტიფაქტების არსებული სიმრავლე მოიცავს: მონაცემების ობიექტს (Data Object); ჯგუფს (Group) და ანოტაციას (შენიშვნა –Annotation).

30.2. საწარმოო რესურსების მართვის (ERP) სისტემა და მისი დანერგვის პროცესები

60-იანი წლების დასაწყისში გამოთვლითი სისტემების პოპულარობის გაზრდამ წარმოშვა იდეა გამოეყენებინათ ისინი საწარმოს პროცესების დასაგეგმად. მისი შექმნა აუცილებელი გახადა საწარმოში მომარაგების დაგეგმვასთან, საწყობის (მარაგების) მართვასთან და კონტროლთან დაკავშირებული პრობლემების გამო. მსგავსი პრობლემების თავიდან ასაცილებლად შემუშავდა MRP (Material requirement planning) მეთოდოლოგია. ამ სისტემის შემდგომმა განვითარებამ წარმოშვა სისტემა MRP2 (Manfactory reasource planning) [145]. აბრევიატურიდან გამომდინარე, ეს სისტემა შექმნილი იყო საწარმოს ყველა რესურსის ეფექტური დაგეგმისათვის, ფინანსური და ადამიანური რესურსების ჩათვლით.

სტანდარტი MRP2 შემუშავდა ამერიკის შეერთებულ შტატებში APICS-ის (American Production and inventory control society) მიერ. ამ სისტემის განვითარებამ 1990 წელს წარმოშვა ERP (Enterprise reasource planning - იარპი) სისტემა.

ERP არის ინფორმაციული სისტემა, რომელიც ერთიან ფუნქციურ გარემოს ქმნის და მენეჯმენტს საშუალებას აძლევს მართოს კომპანია მსოფლიოში საუკეთესო ბიზნეს-პრაქტიკების და სტანდარტების გამოყენებით.

იარპი სისტემა ერთიან უწყვეტ ჯაჭვში აერთიანებს ფინანსური მენეჯმეტის, ბუღალტერიის, წარმოების, მატერიალური მარაგების, დაგეგმარების, გაყიდვების, შესყიდვების, დისტრიბუციის, მარკეტინგის და სხვა ბიზნეს ერთეულების პროცესებს. იგი ახდენს კომპანიის მართვის კომპლექსურ ავტომატიზაციას, მაგრამ ეს მხოლოდ კომპიუტერული სისტემების დანერგვა როდია. იარპი სისტემის დანერგვა ახალ მმართველობით კონცეპციაზე გადასვლაა, რომელიც თავის თავში გულისხმობს მართვის ახალი სტანდარტების და ინსტრუმენტების გამოყენებას. იგი დაფუძნებულია ცენტრალურ მონაცემთა ბაზის სისტემაზე და მოიცავს კომპანიის ყველა ბიზნეს-ოპერაციას.

იარპი სისტემის დახმარებით კომპანია შეძლებს მოთხოვნების და მომარაგების განჭვრეტას და ბალანსირებას, შრომითი რესურსების ეფექტურად გამოყენებას, საჭირო ინფორმაციის ოპერატიულად მიღებას, ხარჯების და შემოსავლების დაგეგმვას და ანა-ლიზს, აღრიცხვიანობის მოწესრიგებას, თვითღირებულების და მოგების კონტროლს. მისი მიზანი არის მომსახურების გაუმჯობესება, პროდუქტიულობის გაზრდა, ფასების შემცირება და ის ასევე ქმნის საძირკველს ეფექტური მომარაგებისა და ელექტრონული კომერციისათვის. ეს კეთდება გეგმის განვითარების საფუძველზე ისე, რომ საჭირო რესურსები – სამუშაო ძალა, საქონელი და ფული იყოს ხელმისაწვდომი საჭირო რაოდენობით, საჭირო დროს.

დაწყებით ეტაპზე ERP სისტემა აღმოცენდა საწარმო ორგანიზაციაში, დღეს კი მან დაფარა ყველა ბიზნესპროცესი და ფუნქცია. ტიპური ERP მოდული შეიცავს: წარმოებას, მომარაგების ჯაჭვს, საწყობის მართვას, ფინანსებს, CRM და HR. იარპი სისტემა არის ყველაზე ძვირადღირებული პროგრამული უზრუნველყოფა. სხვა სიტყვებით, ERP აერთიანებს ყველა მონაცემს და პროცესებს ორგანიზაციის ერთ გაერთიანებულ სისტემაში.

ERP სისტემის მიზანია:

- კომპანიაში აღრიცხვიანობის მოწესრიგება და კონსოლიდაცია;
- სწორი მენეჯერული გადაწყვეტილებების მიღების შესაძლებლობა ზუსტ და აქტუალურ ინფორმაციაზე დაყრდნობით;
- კომპანიის განვითარების სცენარების დაგეგმარებისა და მოდელირების შესაძლებლობა;
- ფინანსური საკითხების სწრაფი და ეფექტური გადაწყვეტა;

- ხარჯების და შემოსავლების დაგეგმვა/ფაქტობრივი-ანალიზი, გადასახადების კონტროლი;
- რეალური თვითღირებულების და მოგების კონტროლის შესაძლებლობა;
- წარმოების პროცესის ეფექტურობის გაზრდა და გამოშვებული პროდუქციის მომგებიანობის გაზრდა;
- კომპანიის საქმიანობის გამჭვირვალობა კომპანიის მენეჯმენტისთვის და მფლობელებისთვის;
- მომხმარებლებთან მომსახურების გაუმჯობესება;
- პროდუქტიულობა, ფასების შემცირება.

ზემოთ აღწერილი იყო იარპი სისტემის უპირატესობა. ხშირად საქართველოში აიგივებენ MRP2 და ERP სისტემებს, რაც არასწორია. ამიტომ მნიშვნელოვანი იქნება თუ მოვიყვანთ ამ ორი სხვადასხვა კლასის სისტემის მახასაითებლების შედარებას. აქვე უნდა აღინიშნოს, რომ MRP2 და ERP სისტემებისათვის ძირითადი გამოყენების სფეროა წარმოება. ისინი უთუოდ ბაზრის მოთხოვნილების შესაბამისად ვითარდება. ემატებათ ახალი ფუნქციები და გადადიან ახალ ტექნოლოგიურ პლატფორმაზე.

ამასთან განსხვავებები MRP2-სა და ERP-ს შორის არსებობს სწორედ წარმოების დაგეგმვის სფეროში. აღნიშნული განსხვავება უკავშირდება დაგეგმვის რეალიზაციის სიღრმეს, რაც განპირობებულია ამ სისტემების ორიენტაციით ბაზრის სხვადასხავა სეგმენტებზე.

ERP – (დიდი) სისტემები იქმნება მრავალფუნქციური და ტერიტორიულად განაწილებული სამრეწველო კორპორაციებისათვის (მაგალითად, ჰოლდინგები, ტრანსნაციონალური კომპანიები, საფინანსო-სამრეწველო ჯგუფები და სხვა).

MRP2 სისტემები ორიენტირებულია საშუალო საწარმოების ბაზარზე, რომლებსაც არ სჭირდება ERP-სისტემებისათვის დამახასიათებელი სიმძლავრეები.

MRP2 და ERP სისტემებს შორის განსხვავება შესაძლოა გამოისახოს შემდეგი ფორმულით:

ERP = MRP2 + ნებისმიერი ტიპის წარმოების რეალიზაცია +
+ რესურსების დაგეგმვის ინტეგრირება კომპანიის საქმიანობის სხვადასხვა მიმართულებით +
+ მრავალრგოლიანი დაგეგმვა

რა თქმა უნდა, MRP2-სისტემების უმრავლესობა ვითარდება სიღრმისეული დაგეგმვის უზრუნველყოფის პოზიციებიდან გამომდინარე და ზოგიერთი პარამეტრის მიხედვით უახლოვდება ERP სისტემებს. მაგრამ „ზოგიერთი პარამეტრის მიხედვით“ არ ნიშნავს „ყველა პარამეტრს“, ამიტომ „ERP“ ტერმინის გამოყენებას სიფრთხილე სჭირდება.

საინფორმაციო მმართველობითი სისტემების თანამედროვე ბაზარი შედგება ლიდერი-სისტემების სამეულისაგან (ზოგიერთი შეფასებებით ხუთეულისაგან), რომლებიც საკუთრივ ERP-კლასის სისტემებს მიეკუთვნება.

უპირობო ლიდერებია გერმანული კომპანიის SAP AG-ს სისტემები – SAP R/3, ამერიკული კომპანიის Oracle-ს სისტემა Oracle Applications და ჰოლანდიური Baan-კომპანიის მიერ შემუშავებული სისტემა Baan. ხანდახან ამ „ელიტარულ" ნუსხას უმატებენ J.D. Edwards და PeopleSoft კომპანიების მიერ წარმოებულ OneWorld სისტემას. მათ იარპი ბაზრის 64% უკავია და მთავრ როლს თამაშობენ ახალი ბაზრების ფორმირებასა და იარპი სისტემების ფუნქციურობის განვითარებაში. ინდუსტრიის ანალიტიკოსები ვარაუდობენ, რომ მომავალში სულ უფრო მეტი კომპანიები დაიწყებს ERP სისტემების გამოყენებას და გაგრძელდება გაერთიანებები იარპის მთავარ შემქმნელებს შორის.

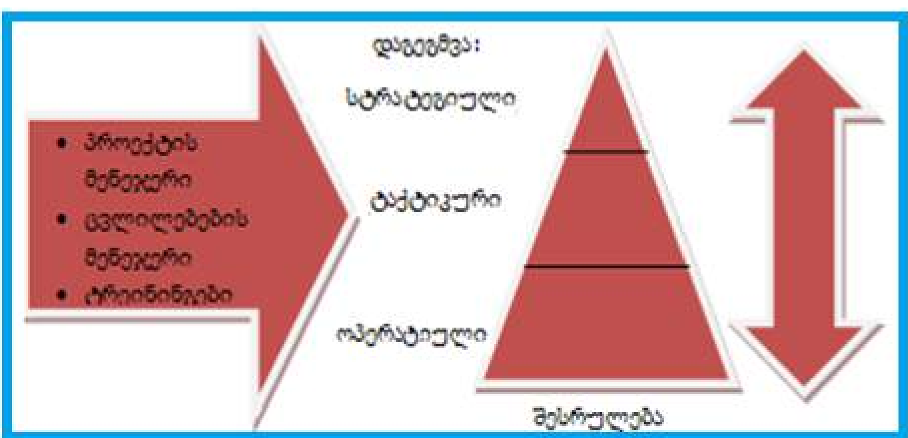
რაც შეეხება MRP2 სისტემებს, აქ გადაწყვეტათა დიდი რაოდენობა შეიმჩნევა, რაც ფუნქციურობის და ტექნოლოგიურ თავისებურებათა უნიკალური თავსებადობის ნიმუშებია. ისინი ერთმანეთისაგან განსხვავდება საწარმოო, ფინანსური და სხვა ფუნქციების დამუშავების ხარისხით. ამიტომაც კონსულტანტების დახმარებით საწარმოს შეუძლია შეარჩიოს ისეთი სისტემები, რომლებიც ყველაზე მეტად პასუხობს მათ მოთხოვნებს. აქედან გამომდინარე, MRP2 არ არის სისტემის არასრულყოფილების ნიშანი, არამედ იმის მაჩვენებელია, რომ სისტემა ორიენტირებულია საშუალო საწარმოთა ბაზარზე.

ჩვენ მოკლედ განვიხილეთ მარკეტინიგული პროცესების მართვის ძირითადი ამოცანები, მათი ავტომატიზაციის მნიშვნელობა ინფორმაციული ტექნოლოგიების გამოყენებით, აგრეთვე საერთაშორისო სტანდარტებით აღიარებული საწარმო რესურსების დაგეგმვის ერთიანი სისტემა (ERP). კომპანიაში არსებული სტარატეგიების ხელშემშლელი ფაქტორების ანალიზის საფუძველზე, მათი გადაჭრის მიზნით განსაკუთრებული ყურადღება ექცევა ბიზნესპროცესების მოდელირების ნოტაციას (BPMN), როგორც ამჟამად ყველაზე ეფექტურ პროცესორიენტირებულ ინსტრუმენტს. იგი გამოიყენება UML-თან ერთად, მაგალითად, Enterprise Architect, Visual Paradigm და სხვა პროგრამულ სისტემებში [20].

ERP სისტემის დანერგვის თვალსაზრისით არსებობს რამდენიმე კრიტიკული საკითხი, რომელიც ყურადღებით უნდა განვიხილოთ ასეთი პროექტების წარმატებისათვის. 30.2 ნახაზზე ნაჩვენებია ის ფაქტორები, რომლებიც მთავარ როლს თამაშობს იარპი სისტემის დანერგვის პროექტში [43].

ნაჩვენებია ამ სისტემის დანერგვის სამი დონე: *სტრატეგიული, ტაქტიკური* და *ოპერაციული.* თავის მხრივ თითოეული დონე მოიცავს კრიტიკულ ფაქტორებს, სამივე დონე არის ერთმანეთზე დამოკიდებული და აუცილებელია თითოეულ დონეზე კარგი მენეჯმენტი.

სამივე დონე არის ერთმანეთზე დამოკიდებული და აუციებელია თითოეულ დონეს მართავდეს კომპეტენტური და გამოცდილი მენეჯერი.



ნახ.30. 2. ERP სისტემის დანერგვის პროექტი

სტრატეგიულ დონეზე ხდება სისტემის სკრინინგი და პროექტის დაგეგმვა. ამ ამოცანების შესრულებას ხელმძღვანელობს პროექტის მენეჯერი;

ტაქტიკურ დონეზე ხდება GAP ანალიზი – ბიზნესის შიგა პროცესების განხილვა და მოდელირება, ბიზნესპროცესების რეინჟინერინგი და კომპანიის შიგა რეკონსტრუქცია;

ოპერაციულ დონეზე ხდება პერსონალის სწავლება და პროგრამის ტესტირება რეალური სცენარებით.

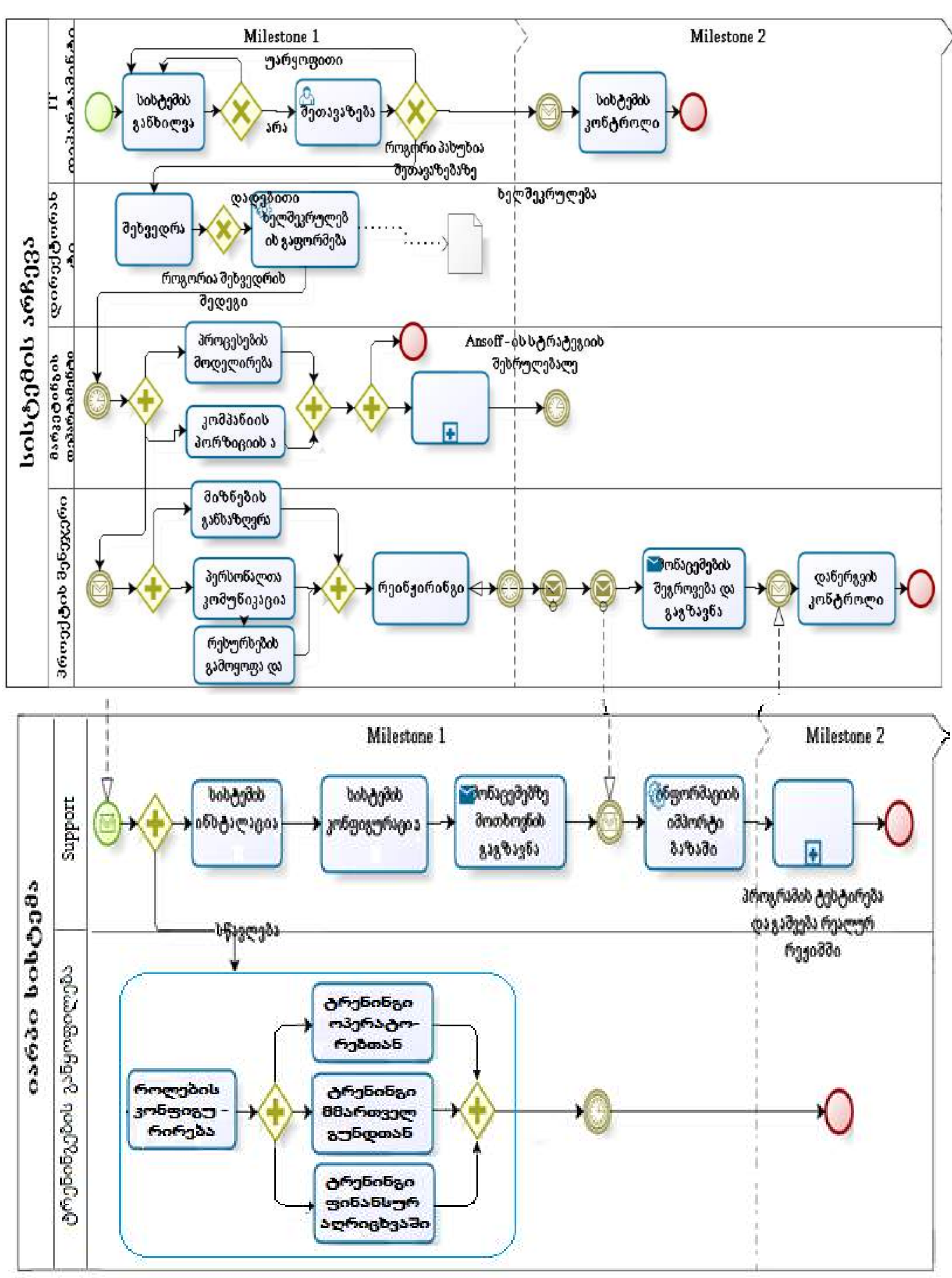
დანერგვის ეს ამოცანები უნდა შეასრულოს ორგანიზაციამ იარპი სისტემის წარმატებით დანერგვის უზრუნველსაყოფად. ეს ამოცანები დაწვრილებით არის განხილული და მათი განხილვისთვის გამოყენებულია პროცესების მოდელირების უახლესი სტანდარტი „BPMN" – ბიზნესპროცესების მოდელირების ნოტაცია (ნახ.30.3).

- **პროექტის დაგეგმვა**

ინფორმაციული სისტემის დანერგვის პროცესის დაგეგმვა არის ყველაზე მეტად მნიშვნელოვანი იარპი პროექტის წარმატებისთვის.

ეს არის გადამწყვეტი დონე, სადაც ხდება პროექტის მიზნის განსაზღვრა, კლიენტებთან და პერსონალთან კომუნიკაცია, დანერგვის პროცესის დაგეგმარება, ამ ფაზაზე ხდება დეტალების შესწავლა, დგინდება ვადები და რესურსები, რომელიც საჭიროა პროცესის გასამართად, ხდება როლების გამოყოფა, პასუხისმგებლობის განაწილება, პროექტის ხედვის და მიზნების განსაზღვრა, რაც ორგანიზაციის მომავალ მიმართულებებს აყალიბებს და რომელიც უნდა გაითვალისწინოს მთელმა კომპანიამ.

ამ დონეზე ხდება იარპი სისტემის სტრატეგიის განხილვა და საწარმოზე ახალი პროგრამული პაკეტის ზემოქმედების განსაზღვრა.



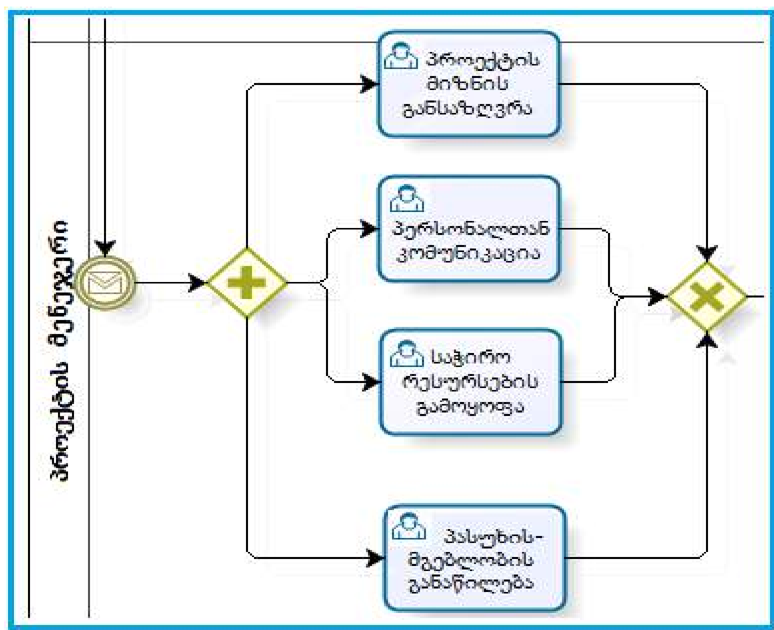
ნახ.30.3. ERP სისტემის დანერგვის პროცესები

პროექტის დაგეგმვისას დიდი წვლილი მიუძღვის ზემდგომი მენეჯერის კვალიფიკაციას და მხარდაჭერას და ეს ფაქტორი განისაზღვრება, როგორც წარმატების ყველაზე მთავარი და გადამწყვეტი წინაპირობა.

მისი მოქმედებები მოიცავს პროექტის მიზნების, საჭიროებების, და სარგებლის განხილვას, საჭირო რესურსების თავმოყრას, სწრაფი გადაწყვეტილებების მიღებას, პერსონალისა და კლიენტების ახალი პროგრამისადმი დადებითად განწყობას და მოსალოდნელ ცვლილებების საჭიროებებში და მისგან გამოწვეულ სარგებელში მათ დარწმუნებას. ხშირად კომპანიები ვერ აფასებს დროს და რესურსებს, რომელიც საჭიროა ახალი იარპი სისტემის დასანერგად. ისმის კითხვა, როგორ გამოვითვალოთ დრო და რე-სურსები, რომელიც იქნება ამ პროცესისათვის აუცილებელი?

e2b ტექნოლოგიების მარკეტინგის დირექტორის, ჯეიმს მელორის აზრით, დრო, რომელიც ERP სისტემის დანერგვისთვისაა საჭირო, შეგვიძლია შევაფასოთ პროგრამული უზრუნველყოფის ფასის 100-ზე გაყოფით. მაგალითად, თუ პროგრამის ფასი არის 20.000$, მაშინ საჭირო იქნება დაახლოებით 200 თანამშრომელი და 200 საათი, თუ დანერგვის პროცესში გამოვიყენებთ სერთიფიცირებულ კონსულტანტებს. ეს რიცხვი გაორმაგდება თუკი თავად შევეცდებით დავნერგოთ ნაკლებად პროფესიონალი ასისტენტების დახმარებით [146].

პროექტის დაგეგმვის პროცესი განხილულია BPMN მოდელირების სტანდარტით, ეს არის ძირითადი პროცესის ამოცანები, რომელსაც პროექტის მენეჯრი ასრულებს (ნახ.30.4).



ნახ.30.4. დაგეგმვის პროცესი

- **სისტემის სკრინინგი**

ERP სისტემის არჩევა არის ყველაზე რთული ამოცანა და ყველაზე სარისკო გადაწყვეტილება, რომლის წინაშეც შეიძლება კომპანია დადგეს. საწარმომ უნდა აირჩიოს პროგრამული უზრუნველყოფის შერჩევის ექსპერტები და მეთოდები.

სისტემის არჩევასთან დაკავშირებული დრო ზრდის იარპი სისტემის დანერვის პროექტის ხანგრძლიობას. სისტემის არჩევის შემდეგ ხდება მისი მოდულების დეტალურად განხილვა ანუ სისტემის სკრინინგი.

არსებობს რამდენიმე შეცდომა, რომელიც ხშირად ხდება დანერგვის პროცესის თითოეულ ამოცანაში და რომელიც ძირეულია სისტემის შემდგომი წარმატებით ექსპლოატაციისათვის. მაგალითად, ერთ-ერთი ასეთი პრობლემაა სისტემის ძირითად მახასიათებლებთან დაკავშირებული გაურკვევლობა.

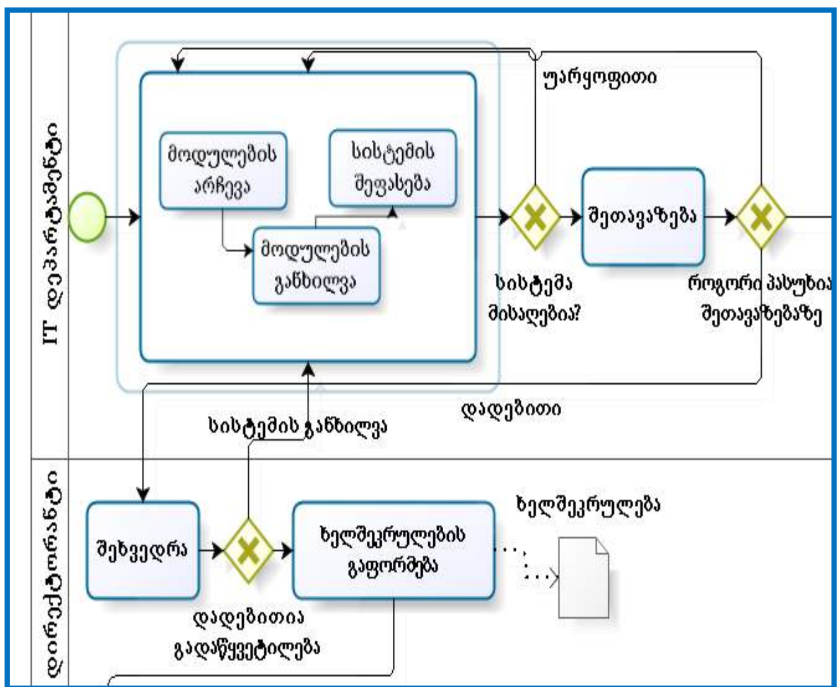
წლიური ERP მიმოხილვებიდან ჩანს, რომ ამ სისტემის მომხმარებლების მხოლოდ 46 პროცენტს აქვთ გარკვეული ცოდნა, თუ თავიანთი იარპი სისტემის, რომელ მახასიათებლებს იყენებენ [147]. „ეს არის შოკის მომგვრელი", ამბობს ჯონ ჰობერი, მორგან ფრანკლინის კორპორაციის დირექტორი, „თუკი მხედველობაში მივიღებთ, რომ მილიონობით კომპანია დებს ინვესტიციას იარპი სისტემებში. მახასიათებლების ცოდნის გარეშე ისინი ხელიდან უშვებენ შესაძლებლობას – მოახდინონ პროცესების სრული ავტომატიზაცია, შეასრულონ ფუნქციები უფრო სწრაფად და მიაღწიონ დასახულ მიზნებს".

უნდა გაკეთდეს მახასიათებლების სია, ERP სისტემის თუ რომელი მახასიათებელი იქნება გამოყენებული და რომელი იქნება ყველაზე მეტად სასარგებლო. არჩეული სისტემის სკრინინგი არის სწორედ პროგრამის თითოეული ფუნქციის გათვალისწინებით, საჭირო მოდულების ჩამოყალიბება და არჩევა, რაც დამოკიდებულია ორგანიზაციის შინაარსზე, მის შიგა სტრუქტურაზე.

წინასწარი სკრინინგის პროცესები აღწერილია ბიზნესპროცესების მოდელირების ნოტაციის (BPMN) გამოყენებით. პირველად უნდა მოხდეს მოდულების განხილვა, შემდეგ მისი შეფასება, ეს ამოცანა არის „loop" ამოცანა, ანუ ის სათითაოდ შეაფასებს ყველა მოდულს და შემდეგ ექსკლუზიური გასასვლელის საშუალებით მოახდენს შესაფერისი მოდულის არჩევას (ნახ.30.5).

- **GAP ანალიზი**

ორგანიზაციის მენეჯერებს ხშირად არ აქვთ ზუსტად განსაზღვრული კომპანიაში მიმდინრე ბიზნესპოცესები და ის, თუ როგორ უნდა განავითაროს ეს პროცესები მაქსიმალური მოგების და ეფექტურობის მისაღწევად. ამ პრობლემის გადასაჭრელად საუკეთესო გამოსავალია, რომ კომპანიამ იარპი სისტემის დანერგვამდე ჩაატაროს ორგანიაციაში მიმდინარე პროცესების შიგა აუდიტი და GAP ანალიზი.



ნახ.30.5. ERP სისტემის სკრინინგი

GAP – ინსტრუმენტია, რომლის დახმარებითაც კომპანიებს შეუძლია ერთმანეთს შეადაროს მიმდინარე წარამატება და მოსალოდნელი შესაძლებლობები. ამ ანალიზის მთავრი შეკითხვა არის – „სად ვართ“ და „სად გვინდა, რომ ვიყოთ“.

ამ ორ ფაქტორს შორის არსებული განსხვავება არის გეპი (GAP – ინგლისური სიტყვაა და ნიშნავს ნაპრალს). იგი ყველაზე სასარგებლო ინსტრუმენტია მარკეტინგის მენეჯერისათვის, შექმნას სწორი მარკეტინგული სტრატეგია და სწორად გამოიყენოს იგი ტაქტიკური ინსტრუმენტების მეშვეობით.

გეპ ანალიზი უჩვენებს ქვედა ფაზას, სადაც დარჩება კომპანია, თუ მენეჯმენტი საჭირო ზომებს არ მიიღებს, და – იმ ზედა ფაზას, რომელსაც უნდა რომ თავისი საქმიანობით მიაღწიოს კომპანიამ.

შემდეგი ნაბიჯი არის ამ ორ ფაზას შორის არსებული ნაპრალის ამოვსება. ამისთვის წარმატებულად გამოიყენება Ansoff_ის მატრიცა [148]. მარკეტინგის ეს სტრატეგიული მენეჯმენტის ანალიზური ინსტრუმენტი პირველად გამოქვეყნდა 1957 წელს ამერიკაში რუსმა მეცნიერმა იგორ ანსოფმა სტატიაში „დივერსიფიკაციის სტრატეგია“ [149]. მას იყენებენ მარკეტოლოგები კომპანიის შემოსავლიანობის გაზრდის მიზნით.

Ansoff_ის მატრიცა გვთავაზობს მიზნების მიღწევის რამდენიმე სტრატეგიულ გადაწყვეტილებას (ნახ.30.6). არსებობს ოთხი მთავარი სტრატეგია:

პროდუქტი / ბაზარი	არსებული	ახალი
არსებული	ბაზარზე შეღწევადობა	პროდუქტის განვითარება
ახალი	ბაზრის განვითარება	დივერსიფიკაცია

ნახ.30.6. Ansoff-ის მატრიცა

- **ბაზარზე შეღწევადობის გაზრდა:** აქ ხდება არსებული პროდუქტის, არსებულ ბაზარზე შეთავაზება. ეს ნიშნავს კომპანიის წლიური შემოსავლის გაზრდას პროდუქტის პრომოუშენით და ბრენდის პოზიციონირებით, არ ხდება პროდუქტის შეცვლა, ახალი კლინეტების მოძებნა.
- **ბაზრის განვითარება**: აქ ხდება არსებული პროდუქტის შეთავაზება ახალ ბაზარზე. ეს ნიშნავს, რომ პროდუქტი რჩება იგივე, მაგრამ ბაზარი ფართოვდება. მაგალითად, პროდუქტის ექსპორტი ან ახალ რეგიონში მისი გატანა.
- **პროდუქციის განვითარება:** ეს არის ახალი პროდუქტის შექმნა და მისი შეთავაზება არსებული მომხმარებლისათვის. აქ ხდება ახალი და მეტად განვითარებული პროდუქტის შეთავაზება არსებულის მაგივრად.
- **დივერსიფიკაცია:** იგი გულისხმობს ახალ ბაზარზე შესვლას ახალი პროდუქტით.

არსებობს ორგვარი დივერსიფიკაცია: პირველი, როდესაც ახალი პროდუქტი კომპანიის არსებულ ბაზრებზე გატანილ პროდუქტისაგან საგრძნობლად განსხვავდება და მეორე, როდესაც ახალი პროდუქტი თვისობრივად ძალიან არ განსხვავდება არსებულისაგან.

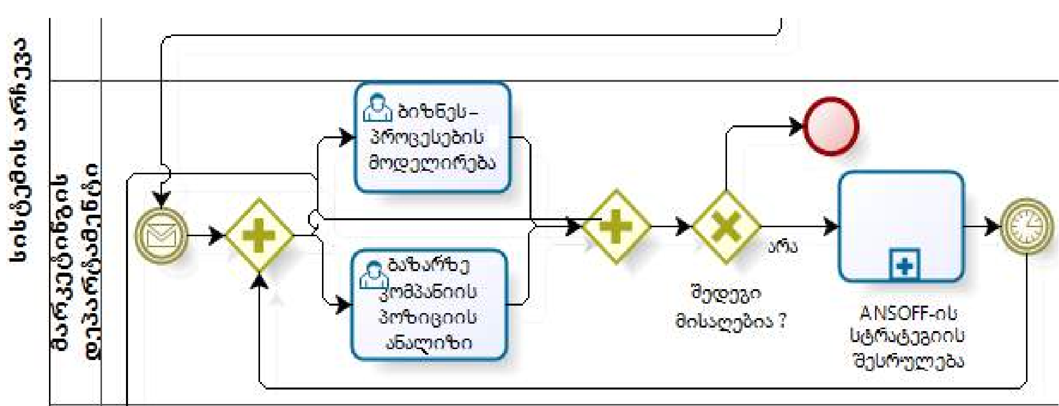
GAP-ის ნაპრალის ამოვსება შესაძლებელია ტაქტიკური ფაქტორების – მარკეტინგული მიქსის ელემენტების გამოყენებით, მაგალითად: ფასის და პრომოუშენის ისეთი ცვლილებით, რომ კომპანია ავიდეს იმ ფაზაზე, რომელიც იქნება ყველაზე მეტი სარგებლის მომტანი. მართალია ეს პროცესი დიდ თანხებთან არის დაკავშირებული, მაგრამ ბიზნესპროცესების წარმატებით ავტომატიზაცია ამ მარკეტინგული ანალიზის გარეშე რთულია.

ERP სისტემის დანერგვის ამ დონეზე, ხდება იარპი სისტემის დანერგვასთან დაკავშირებული პროცესების მოდელირება და განსაზღვრა თუ როგორ იმუშავებს სისტემა არა ტექნიკური, არამედ ბიზნესამოცანების შესრულების თვალსაზრისით და თუ როგორ იფუნქციონირებს ბიზნესპროცესები იარპი სისტემის პაკეტის გამოყენების შემდეგ.

ბიზნესპროცესების მოდელირება არის სრული აღწერა, თუ როგორ დანერგავს საწარმო იარპი სისტემის პაკეტს ბიზნეს სიტუაციების და პროცესების მხარდასაჭერად.

პროგრამის დანერგვის ამ ამოცანას ასრულებს მარკეტინგის მენეჯერი. იგი მოიცავს ორ ქვეამოცანას:

- პირველია კომპანიაში GAP ანალიზის ჩატარება, ანუ ბაზარზე კომპანიის პოზიციის ანალიზი და
- მეორე – საწარმოში არსებული და მოსალოდნელი პროცესების მოდელირება (ნახ. 30.7) [43].



ნახ.30.7. GAP ანალიზი

სისტემის კონფიგურაცია

ERP სისტემის დანერგვა არის კომპლექსური ამოცანა და უმეტესი კომპანია ირჩევს დაიქირაოს კვალიფიციური კონსულტანტები პროგრამის დანერგვასა და სისტემის კონფიგურირებაში დასახმარებლად. აღიარებულია, რომ პროექტის წარმატება დამოკიდებულია კონსულტანატების შესაძლებლობებზე და უნარზე, რადგან ორგანიზაციაში მხოლოდ მათ აქვთ პროგრამის საფუძვლიანი ცოდნა.

იარპი სისტემის დანერგვის პროცესში ჩართულია სამი მონაწილე: ორგანიზაცია, პროგრამული უზრუნველყოფის კომპანია და კონსულტანტები.

როდესაც ახალ ტექნოლოგიასთან გვაქვს საქმე, მაშინ ხშირად კრიტიკულია ორგანიზაციას ჰქონდეს პროგრამის მომწოდებელ კომპანიისგან საუკეთესო მხარდაჭერა.

„Gartner Group“ IT-კომპანიამ დაასაბუთა, რომ კონსულტანტების ღირებულება და პროგრამის ღირებულება ისე შეეფარდება ერთმანეთს, როგორც 3:1, და რომ ის არის კრიტიკული წარმატების ფაქტორი, რაც უნდა იყოს ყურადღებით მართვადი და კონტროლირებადი.

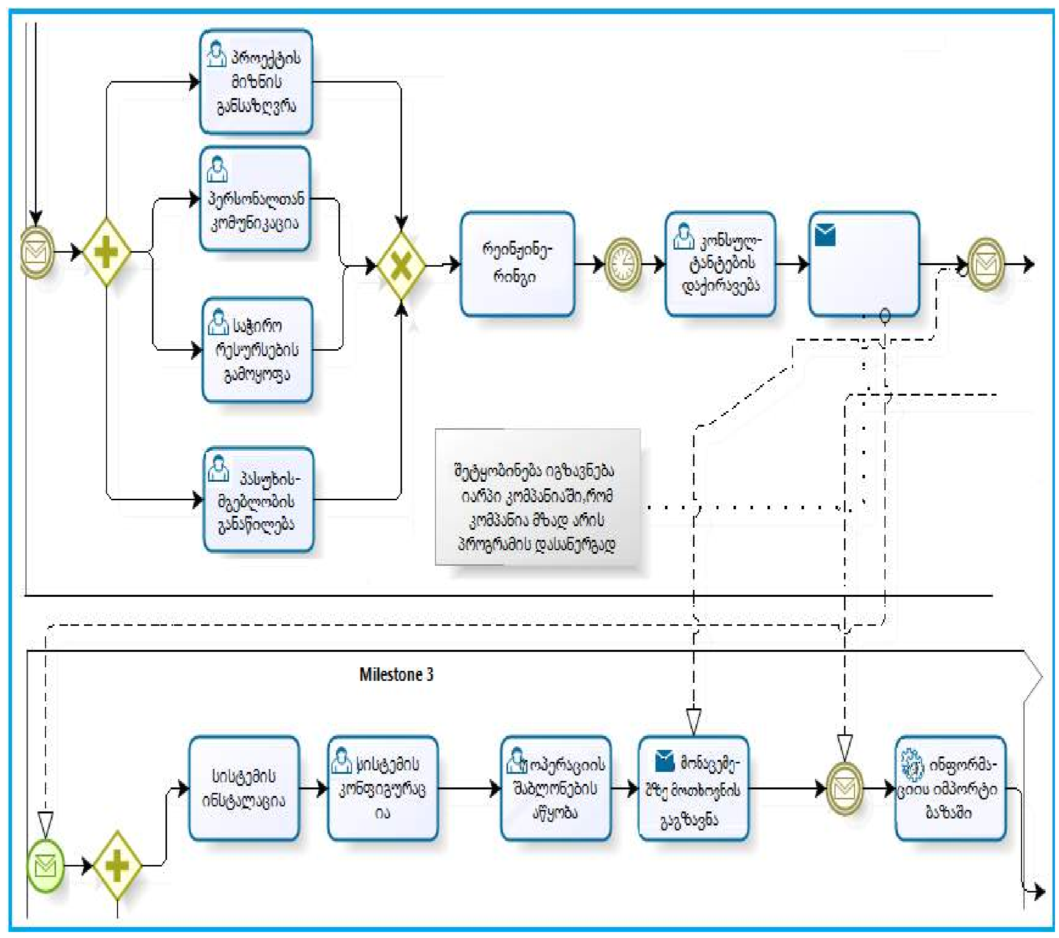
ამიტომ სისტემის კონფიგურაციის პროცესის პირველი ამოცანა არის კვალიფიციური კონსულტანტების დაქირავება.

შემდეგი ამოცანაა პროგრამის კონფიგურაცია, რომელიც კვალიფიციური კონსულტანტების დახმარებით შესრულდება.

პროგრამის კონფიგურაციის ძირითადი ამოცანებია:

- სერვერთან კავშირების მოწყობა;
- ოპერაციის შაბლონების აწყობა;
- ორგანიზაციის სპეციფიკურ მოთხოვნებზე პროგრამის მორგება.

კლიენტთა მოთხოვნილებებზე მორგება არ ნიშნავს ERP პაკეტის მოდიფიცირებას, არამედ – ამ პროგრამაში არსებული ყველა კომპანიისთვის გამოსადეგი პარამეტრის მოწყობას და კონფიგურირებას. ამ დონეზე ასევე ხდება საჭირო და გამოსადეგი მონაცემების შეგროვება და მათი იმპორტი პროგრამის ცენტრალურ მონაცემთა ბაზაში. ეს თანმიმდევრული პროცესები აღწერილია ბიზნესპროცესების მოდელირების ნოტაციის დახმარებით (ნახ.30.8).



ნახ.30.8. სისტემის კონფიგურაცია

- ბიზნესპროცესების რეინჟინერინგი (BPR)

ძირითადად მთავარი დაბრკოლება ERP სისტემის დანერგვის დროს არის ცვლილებებთან დაკავშირებული წინააღმდეგობა. აუცილებელია ამ წინააღმდეგობის შემცირება და ჩვენება, თუ რა სარგებლობა მოაქვს მათთვის ამ ცვლილებებს. ცვლილებების მენეჯერმა აუცილებელია მოამზადოს კომპანიის თანამშრომლები იარპი სისტემის დანერგვისათვის.

არსებობს ERP სისტემის დანერგვის ორი მთავარი ვარიანტი:

1. მოხდეს იარპი პაკეტის მოდიფიკაცია ისე, რომ დააკმაყოფილოს და შეეწყოს ორგანიზაციის მოთხოვნებს;

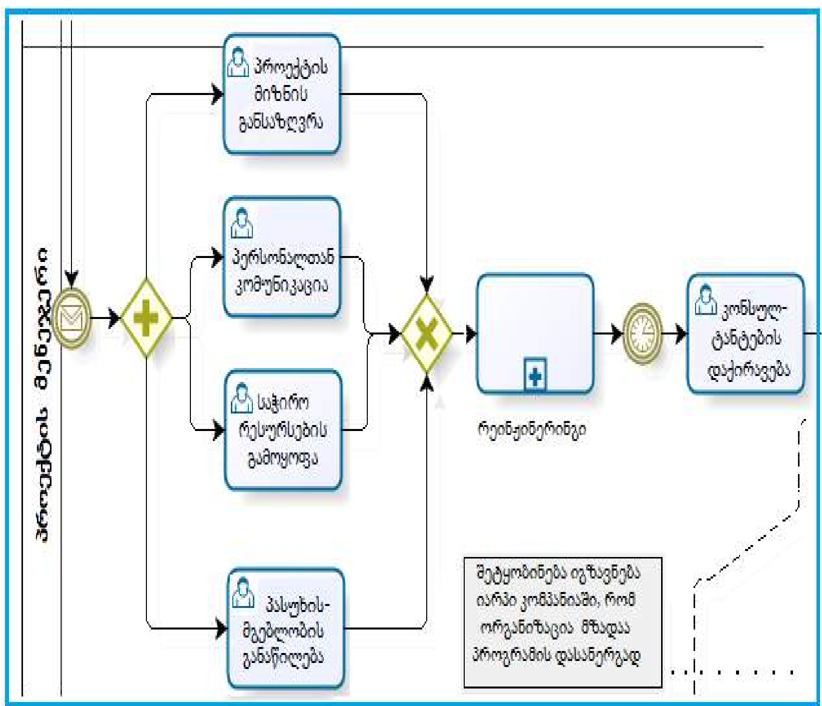
2. დაინრეგოს იარპი სისტემის პაკეტი სტანდარტული ჩარჩოებიდან მინიმალური გადახრებით. რადგან ERP სისტემა არის შექმნილი ინდუსტრიაში არსებული საუკეთესო პრაქტიკებზე დაყრდნობით, მისი წარმატებით ინსტალაციისთვის უმჯობესი იქნება კომპანიის ყველა პროცესი შეეთანხმოს და შეეწყოს იარპი მოდელს.

კვლევებმა აჩვენა, რომ საუკეთესო სისტემის პაკეტსაც კომპანიის მოთხვნების მხოლოდ 70%-ის დაკმაყოფილება შეუძლია. ამიტომ სრულყოფილად წარმატებული იარპი პროგრამის შექმნის საწინდარი კომპანიის ბიზნესპროცესების რედიზაინია. იმ კომპანიებს, რომლებიც არ ეთანხმება ამ ფილოსოფიას, ხშირად ხვდება სირთულეები.

სისტემატურ ცვლილებებთან დაკავშირებული წინააღმდეგობა არის მთავარი დაბრკოლება იარპი სისტემის დანერგვის დროს, რადგან მას უარყოფითად და შიშით ხვდება კომპანიის პერსონალი. აუცილებელია ამ წინააღმდეგობის შემცირება და ჩვენება, თუ რა სარგებლობა მოაქვს მათთვის ამ ცვლილებებს. ცვლილებების მენეჯერმა აუცილებელია მოამზადოს კომპანიის თანამშრომლები იარპი სისტემის დასანერგად.

არსებობს შეკითხვა ამ კუთხით, როდის უნდა მოხდეს კომპანიის ბიზნეს-პროცესების რეინჟინერინგი? იარპი პაკეტის დანერგვამდე, დანერგვის პროცესის დროს თუ დანერგვის შემდეგ. ამაზე არ არსებობს ერთი პასუხი, რომელიც ყველა კომპანიისთვის გაამართლებს. ეს დამოკიდებულია კომპანიის სპეციფიკასა და ბიზნეს სიტუაციაზე.

ERP სისტემის დანერგვა ორგანიზაციაში არის ძალიან დიდი პროექტი და შესაბამისად შეცდომები იქნება დაშვებული და გეგმაც ვერ შესრულდება თუკი სწორად არ შეირჩევიან თანამშრომლები, რომლებიც ამ პროცესში იქნებიან ჩართული. ცვლილებები შეეხება თანამშრომელთა რაოდენობას და მათ პასუხისმგებლობას (ნახ.30.9).

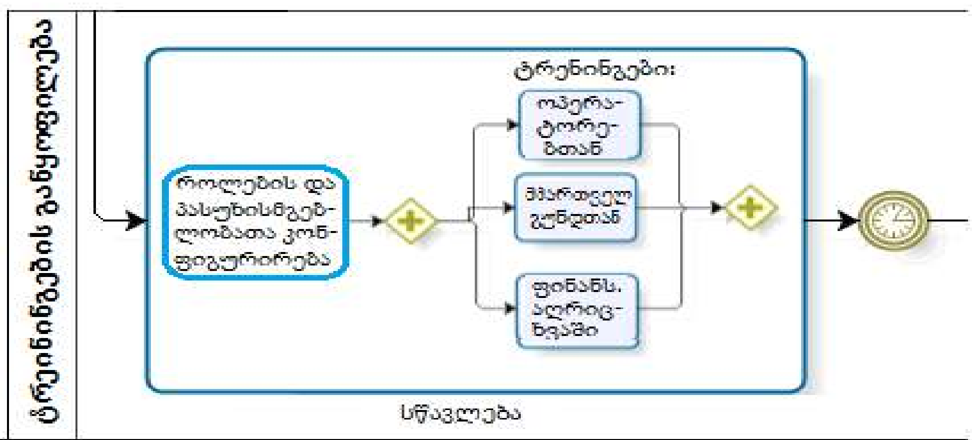


ნახ.30.9. ბიზნეს პროცესების რეინჟირინგი (BPR)

➢ გუნდის სწავლება

ERP სისტემა არის კომპლექსური სისტემა და მოითხოვს ხანგრძლივ ტრენინგებს. შესაბამისი ტრენინგების ნაკლებობა, არის მთავარი მიზეზი იარპი პროექტის მარცხისა. ეს ასევე იწვევს პროგრამის მიმართ პერსონალის უარყოფით განწყობას, რადგან მათ არ იციან იგი. იარპი სისტემის დანერგვა საბოლოო მომხმარებლის მომზადების გარეშე არის დროის და ფულადი რესურსების კარგვა. „სრულყოფილი ტრენინგების გარეშე იარპი სისტემიდან საბოლოოდ მიიღებთ ექსელის ძალიან ძვირადღირებულ ვერსიას“ ამბობს კევინ ჰერინგი GSI_ის ყოფილი პრეზიდენტი და იარპი სისტემის სპეციალისტი [150].

გუნდის სწავლების მოდელში გამოყენებულია პარალელური გსასვლელი, რაც ნიშნავს, რომ პროცესი მიმდინარეობს პარალელურად, ორგანიზაციის სხვადასხვა დონეზე საჭიროა განსხვავებული ტრენინგების ჩატარება და სანამ არ იქნება ყველა ტრენინგი ჩატარებული მოლარე-ოპერატორებთან, ფინანსურ განყოფილებაში და მენეჯმენტთან, პროცესი არ მთავრდება (ნახ. 30.10).



ნახ.30.10. გუნდის სწავლება

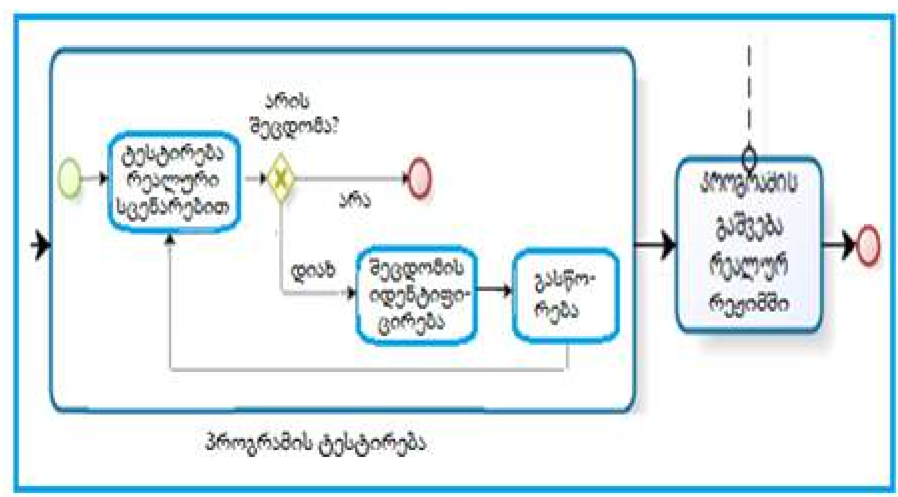
➢ ტესტირება

ბოლო ეტაპზე აუცილებელია დანერგილი პროგრამის ტესტირება რეალური სცენარებით.

ამ მოდელში გამოყენებული გასასვლელით ნაჩვენებია, რომ სანამ ტესტირების შედეგად მიღებული ყველა შეცდომა არ გასწორდება, მანამდე პროცესი არ სრულდება.

ტესტირება ეხმარება კომპანიას თავიდან აიცილოს პრობლემები, რომლებიც უარყოფითად მოქმედებს მომხმარებელზე.

ტესტირების პროცესის მოდელი ნაჩვენებია 30.11 ნახაზზე.



ნახ.30.11. ტესტირება

➢ **რეალურ რეჟიმში მუშაობა**

ამ დონეზე მნიშვნელოვანია, რომ შეფასდეს საბოლოო მომხმარებელთან ჩატარებული ტრეინინგები. ეს არის იარპი პაკეტის დანერგვის ბოლო ეტაპი და შეიცავს ორ მთავარ ბიჯს: სისტემის გააქტიურებას და ძველი სისტემიდან ახალ სისტემაზე გადასვლას.

ამგვარად, ბიზნესპროცესების მოდელირების ინსტრუმენტის Bizagi Process Modeler მეშვეობით წარმოდგენილ იქნა მარკეტინგული დაგეგმვისა და ERP სისტემის დანერგვის ყველა ბიზნესპროცესის დიაგრამა.

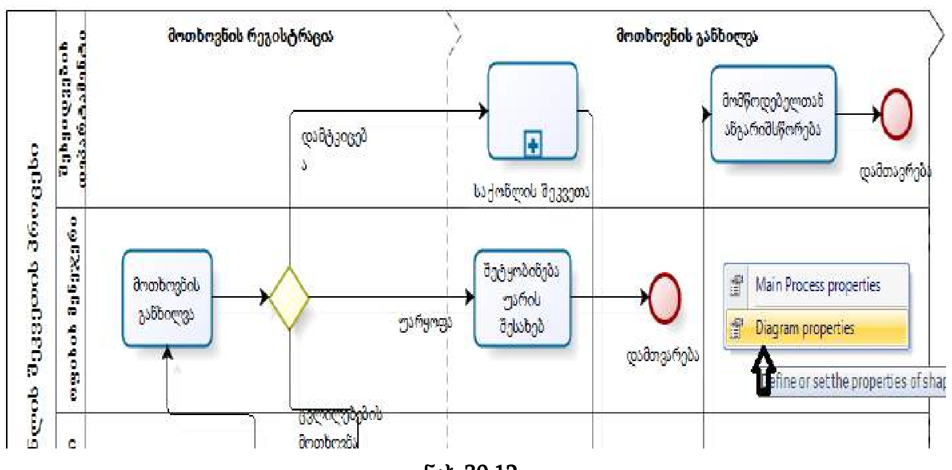
30.3. ბიზნესპროცესის დოკუმენტირება და სიმულაცია

➢ **ბიზნესპროცესის დოკუმენტირება**

Bizagi Proces Modeler საშუალებას იძლევა მოვახდინოთ პროცესის სრულყოფილი დოკუმენტირება [43]. დოკუმენტირება შეგვიძლია, როგორც მთლიანად დიაგრამის, სადაც შევძლებთ აღვწეროთ პროცესი, მისი დასახელება და ავტორი, ბიზნესწესები, მიზანი და ნებისმიერი დამატებითი დახასიათება, ასევე მოვახდინოთ პროცესში მონაწილე თითოეული ელემენტის დოკუმენტირება. ასევე მოდელერი საშუალებას აძლევს მომხმარებელს თავად დაამატოს არტიფაქტები მისთვის მოსახერხებელი ფორმებით.

- **პროცესების დოკუმენტირება**

განვიხილოთ მოდელირებული ბიზნესპროცესის „საქონლის შეკვეთა“ დოკუმენტირების მაგალითი. ამისთვის მოდელის ცარიელ ადგილზე მაუსის მარჯვენა ღილაკის დახმარებით ავირჩიოთ „**Diagram Properties**” (ნახ.30.12)



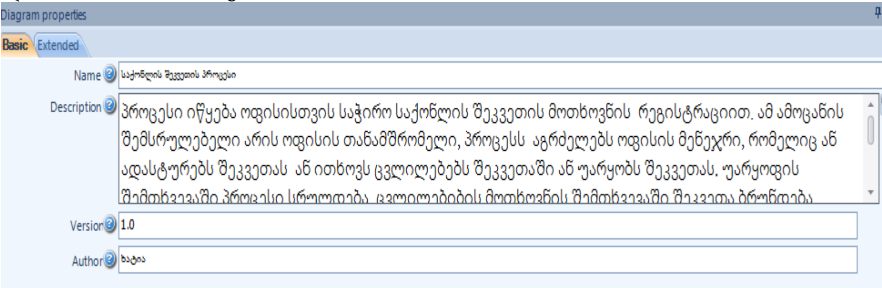
ნახ.30.12

შედეგად გამოჩნდება **დიაგრამის თვისებების** ფანჯარა, რომელსაც, თავის მხრივ, აქვს ორი ჩანართი: ძირითადი და გაფართოებული ჩანართი.

ძირითადი ჩანართი (**Basic**) : შეიცავს ძირითად ინფორმაციას, მათ შორის, პროცესის სახელს, მის აღწერას, ვერსიას და პროცესის ავტორს.

გაფართოებული ჩანართი (**Extended**) : ქმნის დამატებით ელემენტებს, რათა უზრუნველყოს ყველა საჭირო ინფორმაცია სრულყოფილი დოკუმენტაციისთვის.

ძირითადი ჩანართის **Name** ველში მიუთითეთ პროცესის სახელი, **Description** ველში პროცესის აღწერა და **Author** ველში პროცესის ავტორი. შევსებული ეს ფორმა მოცემულიაა 30.13 ნახაზზე.



ნახ.30.13

- **ელემენტების თვისებები**

როგორც აღვნიშნეთ შეგვიძლია მოვახდინოთ, როგორც მთლიანად დიაგრამის, ასევე პროცესში მონაწილე თითოეული ელემენტის დოკუმენტირება. იმ ინფორმაციის მითითება, რომელსაც თითოეული ელემენტი შეიცავს გვაძლევს შესაძლებლობას ნაბიჯ-ნაბიჯ მივიღოთ პროცესში არსებული დეტალური ინფორმაცია.

თითოეულ ელემენტს აქვს საკუთარი თვისებები და ეს თვისებები დამოკიდებულია ელემენტის ტიპზე. მას აქვს ოთხი ინფორმაციული და ფუნქციური ჩანართი:

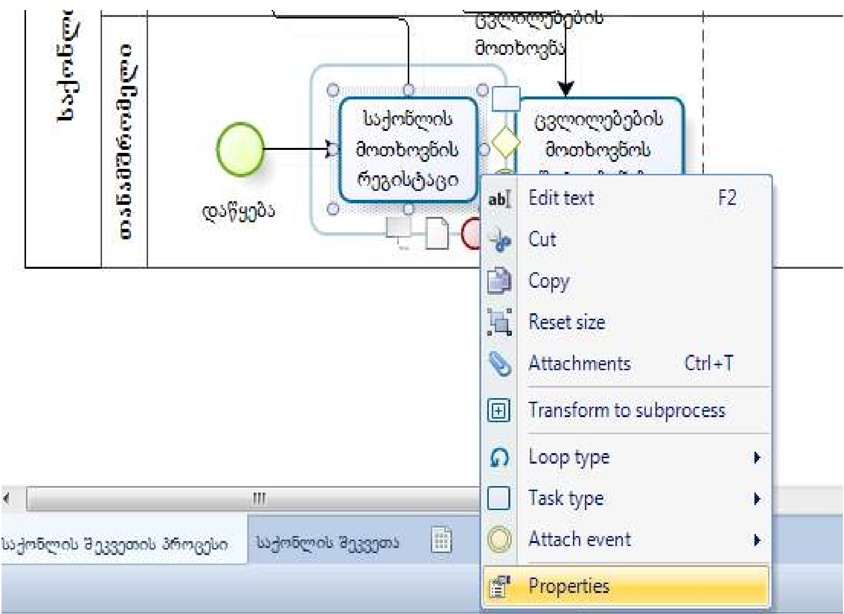
ძირითადი ჩანართი (**Basic**) : შეიცავს ძირითად ინფორმაციას, მათ შორის, ელემენტის სახელს, მის აღწერას და შემსრულებელს.

გაფართოებული ჩანართი (**Extended**) : ქმნის დამატებით ელემენტებს, რათა უზრუნველყოს ყველა საჭირო ინფორმაცია სრულყოფილი დოკუმენტაციისთვის.

დამატებითი ფუნქციის ჩანართი (**Advanced**): იგი კონკრეტული BPMN ატრიბუტებისთვის ვრცელდება.

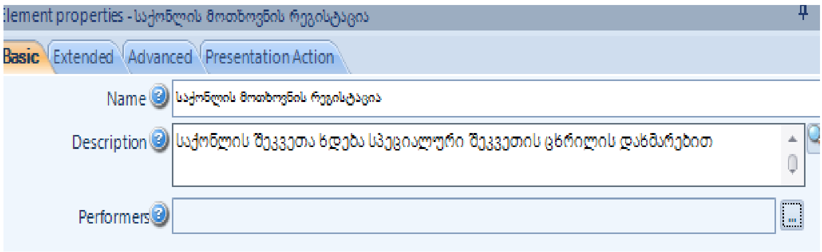
პრეზენტაციის ნაწილი (**Presentation action**) : განსაზღვრავს თუ რა იქნება ნაჩვენები პრეზენტაციის რეჟიმში.

მოვახდინოთ პროცესის პირველი მოქმედების „საქონლის მოთხოვნის რეგისტრაციის" ელემენტის დოკუმენტირება. ამისთვის მოვნიშნოთ ეს ამოცანა და მაუსის მარჯვენა ღილაკის დახმარებით ავირჩიოთ properties ველი (ნახ.30.14).



ნახ.30.14

გაიხსნება **ელემენტის თვისებების** ახალი ფანჯარა, სადაც გააქტიურებულია ძირითადი ფანჯარა. ელემენტის სახელი ავტომატურად არის მითითებული, დავამატოთ მოქმედების აღწერა (ნახ.30.15).

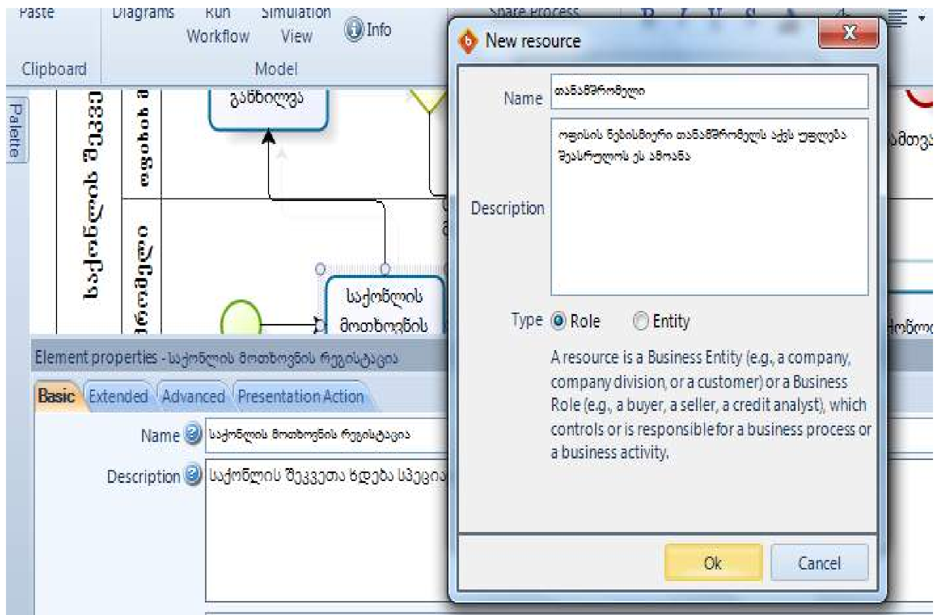


ნახ.30.15

შემსრულებლების დასამატებლად მივმართოთ **performers** ველთან არსებულ ღილაკს . შედეგად გაიხსნება რესურსების არჩევის (**select resources**) ახალი ფანჯარა, ავირჩიოთ ღილაკი , შედეგად გამოჩნდება რესურსების (**resources**) ფანჯარა, ავირჩიოთ შემსრულებლების დამატების ღილაკი და **Name** ველში ჩავწეროთ „საქონლის მოთხოვნის რეგისტრაციის“ ამოცანის შემსრულებლის სახელი.

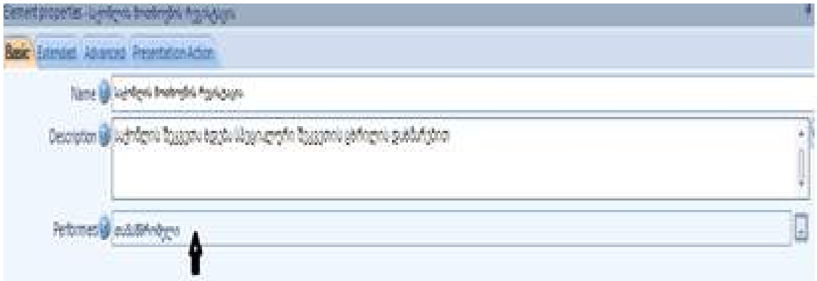
ამ ამოცანას, როგორც უკვე აღვნიშნეთ, ასრულებს ოფისის თანამშრომელი, **Type** ველში მიუთითოთ **Role** და ავირჩიოთ ღილაკი **Ok** (ნახ.30.16).

რესურსების ფანჯარაში დაემატება ერთი შემსრულებელი „თანამშრომელი“. ავირჩიოთ ღილაკი **Ok** რის შედეგადაც დავბრუნდებით რესურსების არჩევის (**select resources**) ფანჯარაში. მაუსის დახმარებით შემსრულებლად მოვნიშნოთ „**თანამშრომელი**“ და მივუთითოთ ღილაკს **Ok.**



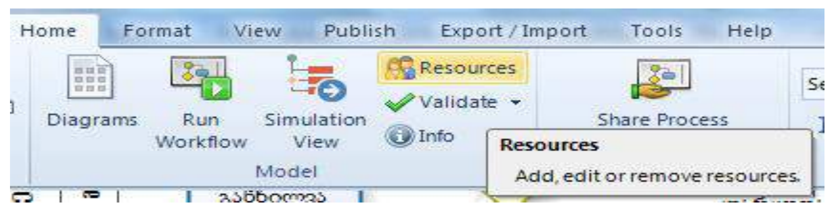
ნახ.30.16

შედეგად ელემენტის თვისებების ფანჯრის შემსრულებლების (**performers**) ველში ავტომატურად მიეთითება „თანამშრომელი“ (ნახ.5.6).



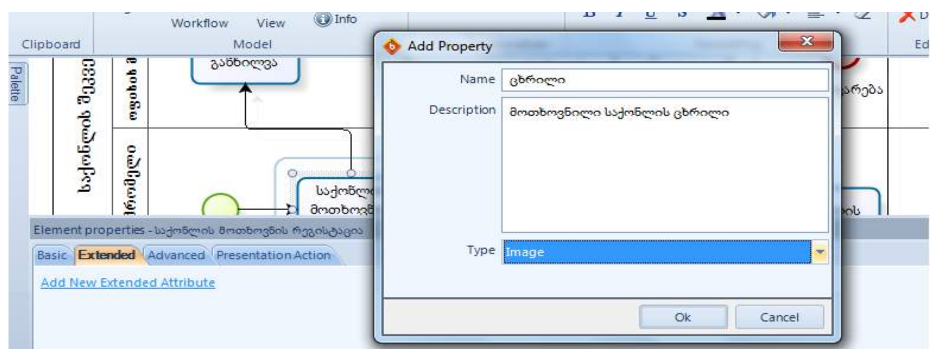
ნახ.30.17

შემსრულებლების დამატება ანალოგიურად ხდება **Home** მოდულის **Resources** ჩანართში (ნახ.30.18). **Home** მოდულის გამოყენებით დაამატეთ ორი შემსრულებელი: ოფისის მენეჯერი და გაყიდვების დეპარტამენტი.



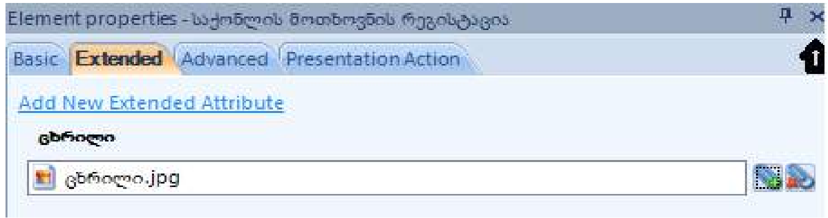
ნახ.30.18

ელემენტის თვისებებში დავამატოთ ინფორმაცია გაფართოებული ჩანართის (**Extended**) გამოყენებით. ამისთვის ელემენტის თვისებების ფანჯარაში ავირჩიოთ **extended** ჩანართი და მაუსის დახმარებით ავირჩიოთ **add new extended attribute**, რის შედეგადაც ეკრანზე გამოვა ახალი ფანჯარა **Add Property** სადაც **Name** ველში მივუთითოთ ატრიბუტის სახელი მაგალითად, „ცხრილი“ და **Description** ველში ატრიბუტის აღწერა, მაგალითად, „შეკვეთილი საქონლის ცხრილი“. ატრიბუტს აქვს თორმეტი ტიპი, მათგან **Type** ველში ავირჩიოთ **image** ტიპი და მივუთითოთ **ok** ღილაკი. შევსებული **Add Property** ფანჯარა (ნახ.30.19).



ნახ.30.19

შედეგად ელემენტის თვისებების extended ჩანართში გამოჩნდება ახალი ველი, დამატების ღილაკის დახმარებით შეგვიძლია მოქმედებას მივამაგროთ სასურველი სურათი. მაუსის დახმარებით მივმართოთ დამატების ღილაკს და ავირჩიოთ სურათი. დავხუროთ ელემეტის თვისებების ფანჯარა ღილაკის გამოყენებით (ნახ.30.20).

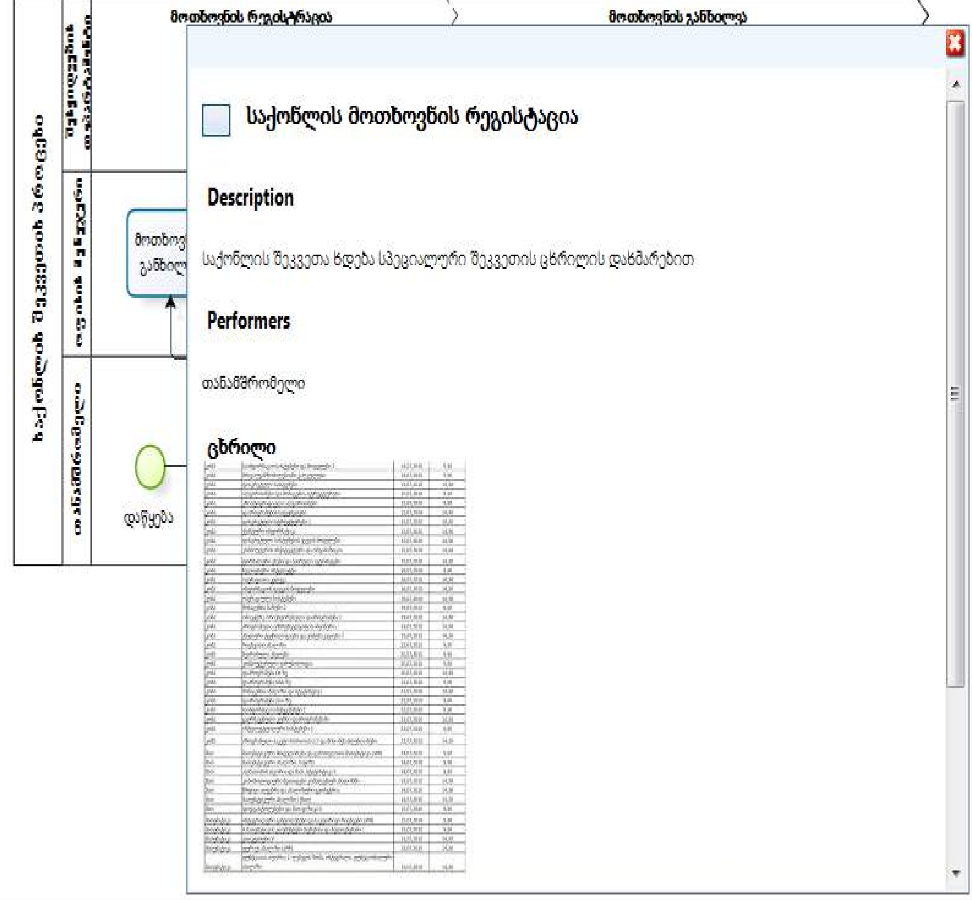


ნახ.30.20

დიაგრამა გადავიყვანოთ პრეზენტაციის რეჟიმში ღილაკის დახმარებით (ნახ.30.21).

ნახ.30.21

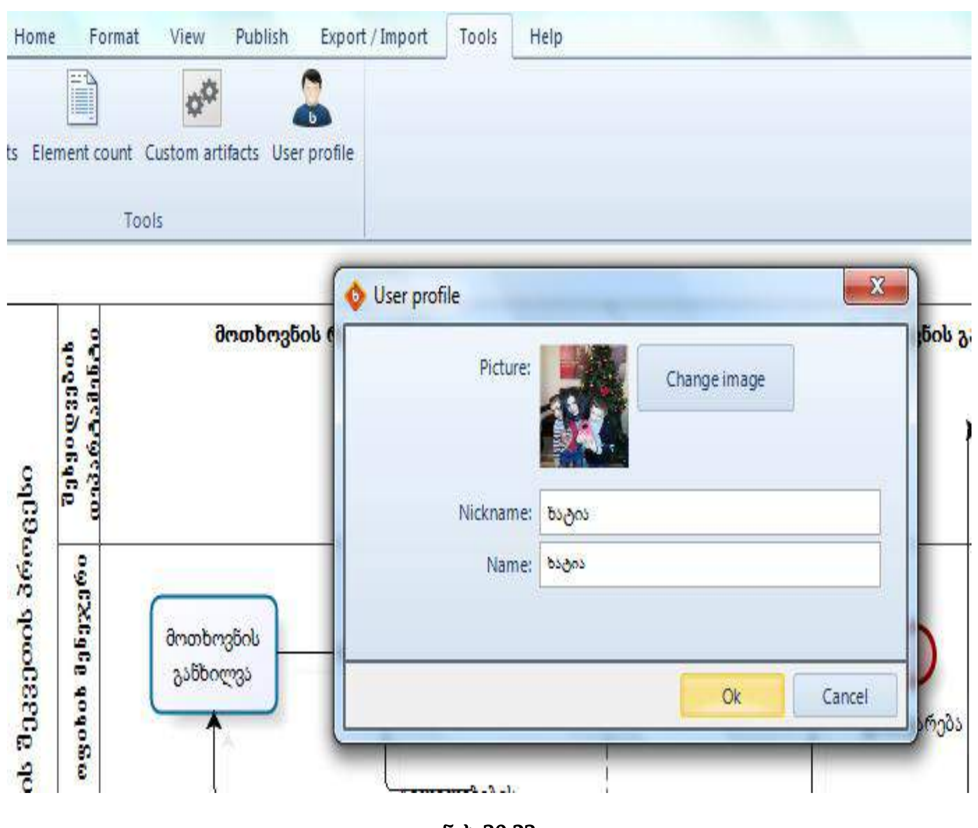
მოვნიშნოთ „საქონლის მოთხოვნის რეგისტრაციის“ მოქმედება; შედეგად გამოჩნდება მასზე მითითებული სრული ინფორმაცია (ნახ.30.22).



ნახ.30.22

- **მომხმარებლის შექმნა და თანამოქმედების მოდული**
  - **მომხმარებლის პროფაილის შექმნა**

მომხმარებლის პროფაილის შესაქმნელად ავირჩიოთ **Tools/User profile**, **Change Image** ღილაკის გამოყენებით ავირჩიოთ პროფილის სურთი, შევავსოთ **Nickname** და **Name** ველები და მივუთითოთ ღილაკს **Ok** (ნახ.30.23).



ნახ.30.23

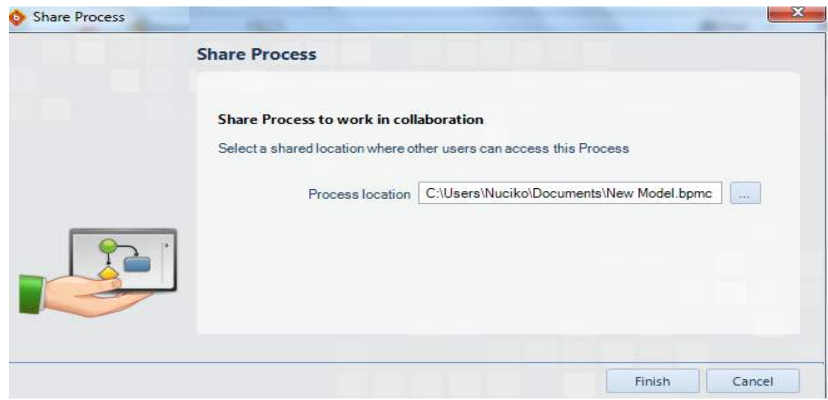
- **თანამოქმედების მოდული**

თანამოქმედების მოდული საშუალებას აძლევს კომპანიის მონაწილეებს გუნდურად მიიღონ მონაწილეობა პროცესის განსაზღვრაში. პროცესის შექმნის ფაზაზე მრავალი მომხმარებელი მუშაობს ერთდროულად მოდელზე, რაც უზრუნველყოფს პროცესის განსაზღვრის უმაღლეს ხარისხს.

Bizagi–ის გუნდური თანამშრომლობის მოდული საშუალებას აძლევს მომხმარებლებს, რომ შეცვალონ და გააუმჯობესონ პროცესი, მოაწყონ ონლაინ დისკუსიები და დააკომენტარონ პროცესები რეალურ დროში, რაც ხილვადი იქნება ყველა მონაწილესთვის. გუნდური თანამშრომლობისთვის აუცილებელია პროცესის

მოდელი ინახებოდეს იმ სივრცეში, რომელიც გუნდისთვის იქნება მისაწვდომი. ეს ნიშნავს, რომ მოდელის გაზიარებისთვის საჭიროა ქსელთან კავშირი.

გუნდური თანამოქმედების მოდულის გასააქტიურებლად ავირჩიოთ **Home/Team collaboration/Share Process,** ... ღილაკით მივუთითოთ მოდელის ადგილმდებარეობა და მივმართოთ **Finish** ღილაკს (ნახ.30.24).



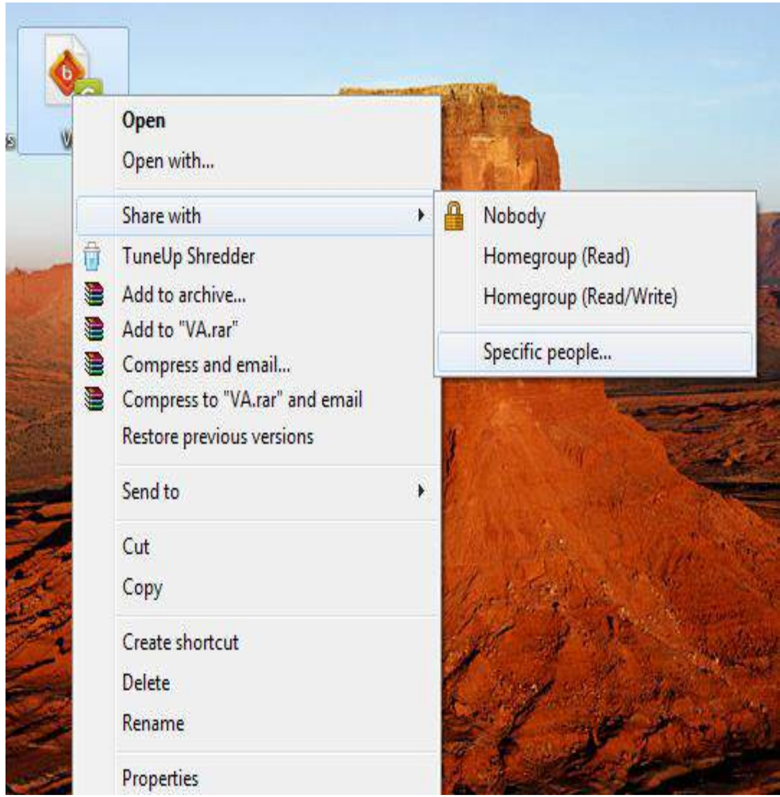
ნახ.30.24

მას შემდეგ, რაც მოდელს შევინახავთ, Bizagi Pocess Modeler-ი ქმნის .bpmc ფაილს ფოლდერით, რომელიც შეიცავს სპეციალურ ფაილს, გუნდური თანამშრომლობის მოდულისთვის.

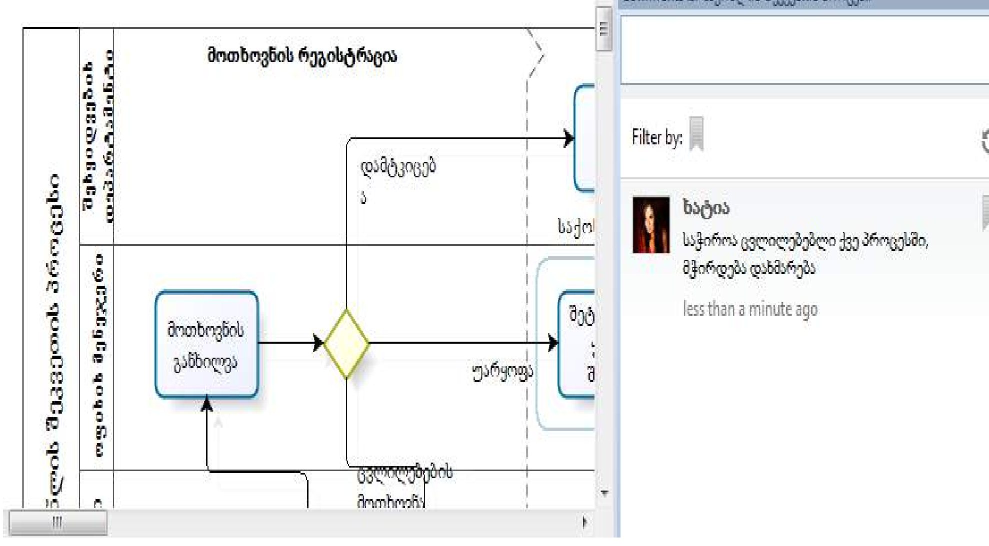
შემდეგ უნდა ავირჩიოთ გუნდი ან პიროვნებები, რომელთაც მივცემთ ნებართვას და წვდომას ამ ფაილთან. ამისთვის ავირჩიოთ მოცემული ფაილი, მივმართოთ მაუსის მარჯვენა ღილაკს, ავირჩიოთ **Share with/Specific people** და მოვძებნოთ სასურველი გუნდი (ნახ. 30.25).

ამის შემდეგ შესაძლებელია ჩვენ და ჩვენს მიერ არჩეულმა გუნდმა ერთად გავხსნათ მოდელი და ვიმუშაოთ პარალელურ რეჟიმში.

ურთიერთობა ხორციელდება კომენტარების ფანჯრის დახმარებით, რომელიც გამოჩნდება ეკრანის მარჯვენა მხარეს (ნახ.30.26). შეტყობინებების მიღება ხორციელდება რეალურ დროში, ასე რომ, როგორც კი შევიტანთ შეტყობინებას და მივუთითებთ **Enter** ღილაკს, მეორე მხარე მომენტალურად შეძლებს მის წაკითხვას.

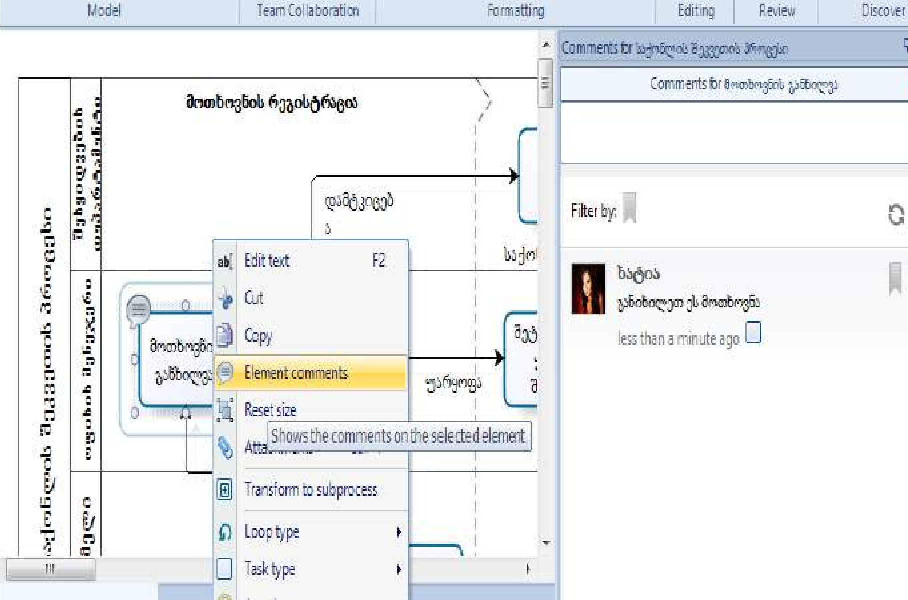


ნახ.30.25



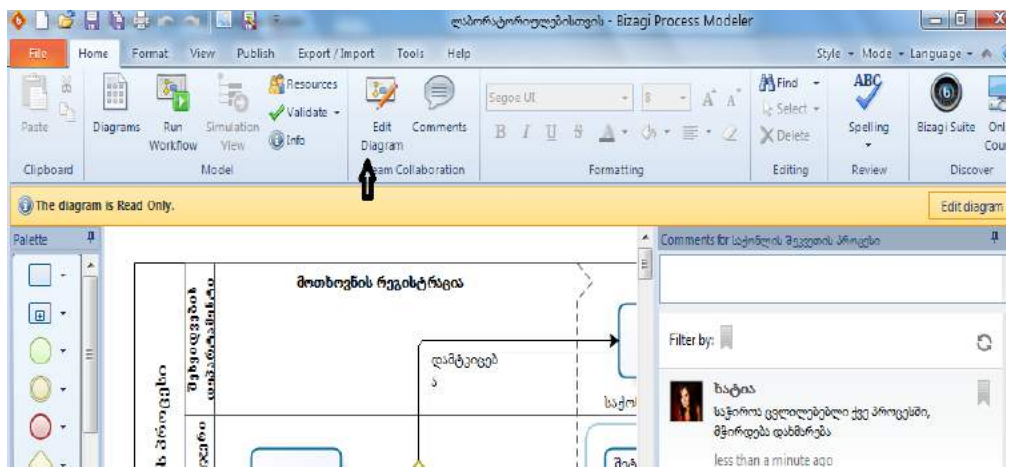
ნახ.30.26

გარდა დიაგრამის დაკომენტარებისა, შესაძლებელია თითოეულ ელემენტს გაუკეთოთ კომენტარი (ნახ.30.27).



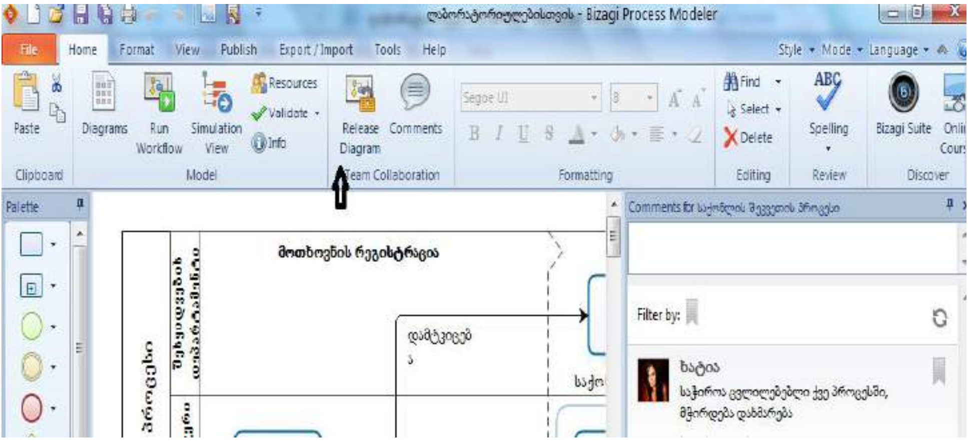
ნახ.30.27

სტანდარტულად ყველა დიაგრამა არის კითხვის რეჟიმში. დიაგრამის შესაცვლელად ან განახლებისთვის მივმართოთ ღილაკს **Edit Diagram**, რომელიც მდებარეობს **Home** მოდულის **Team Collaboration** ჩანართში (ნახ.30.28).



ნახ.30.28

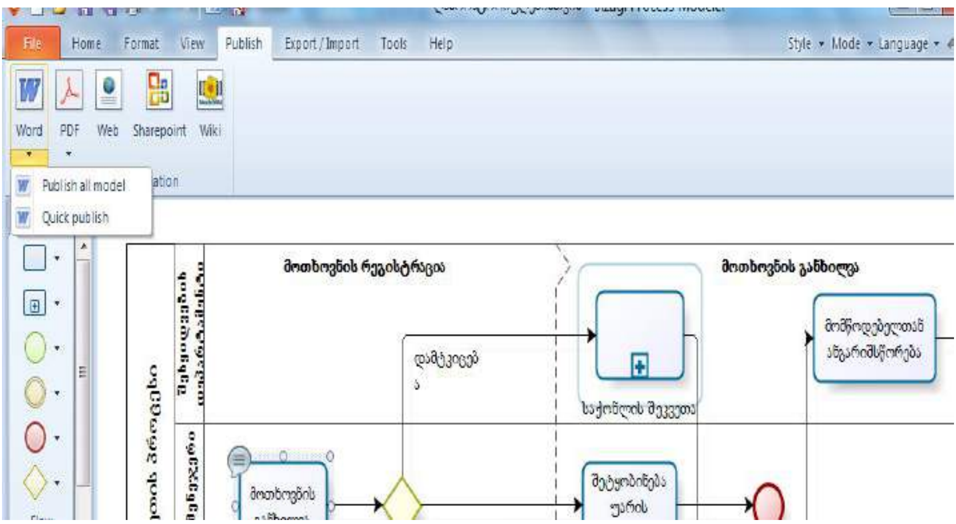
მას შემდეგ რაც გავაკეთებთ სასურველ ცვლილებებს უნდა ავირჩიოთ **Release Diagram** ღილაკი, რომელიც მდებარეობს **Home** მოდულის **Team Collaboration** ჩანართში (ნახ.30.29).



ნახ.30.29

- **დიაგრამის ექსპორტი Word და PDF**

დიაგრამის ექსპორტი მარტივად არის შესაძლებელი Word და PDF ფაილში, შეგვიძლია ავირჩიოთ **Publish all model** (გამოიყენება დიაგრამის ელემენტებით და დოკუმენტირებით სრულად ექსპორტირებისთვის) ან **Quick publish** (სწრაფი ექსპორ-ტირებისთვის) (ნახ.30.30).



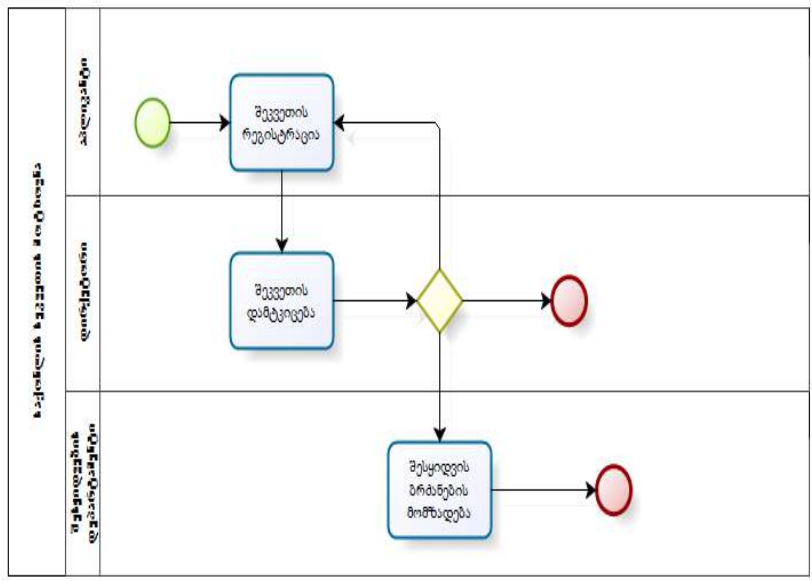
ნახ. 30.30

➢ 5.3. სიმულაცია (იმიტაციური მოდელი)

სიმულაცია არის ინსტრუმენტი მოდელის შესრულების შესაფასებლად, რათა შეამციროს და აღმოფხვრას გაუთვალისწინებელი ფაქტორები და თავიდან აიცილოს ადამიანური, მატერიალური და დროითი რესურსების სიჭარბე ან ნაკლებობა და უზრუნველყოს ამ რესურსების ოპტიმალური გამოყენება. Bizag-ის სიმულაციის გამოყენებით სუკეთესო სცენარის შესაქმნელად აუცილებელია პროცესი იყოს დასრულებული, სხვა შემთხვევაში შედეგი არ იქნება სანდო.

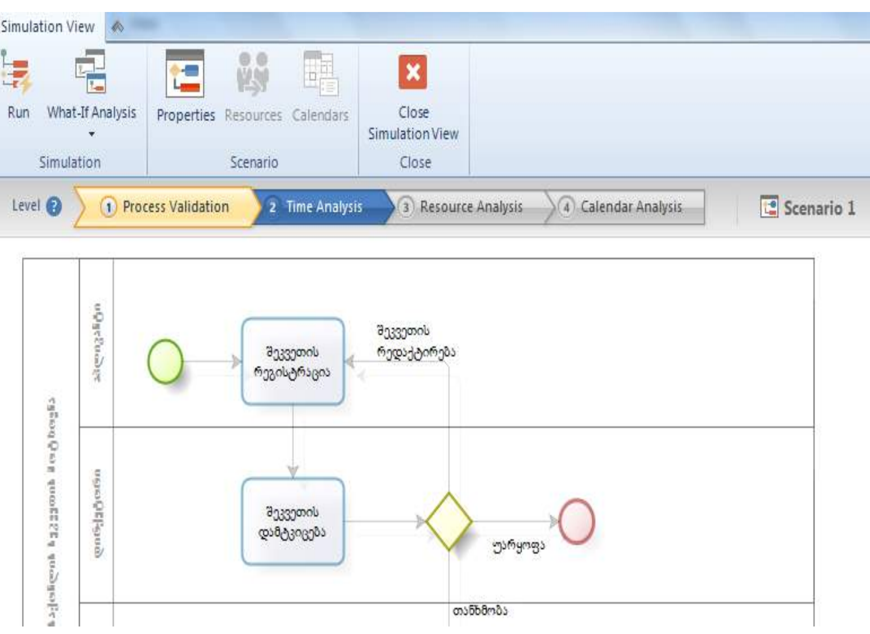
• 5.3.1. ახალი ბიზნესპროცესი

სიმულაცია მოვახდინოთ ბიზნესპროცესის მოდელის სცენარზე, რომელსაც ჰყავს სამი მონაწილე: აპლიკანტი, რომელიც ასრულებს საქონლის მოთხოვნას, დირექტორი, რომელიც ამ მოთხოვნას ან უარყოფს (რის შემდეგაც პროცესი სრულდება) ან ითხოვს ცვლილებას, რის შემდეგაც მოთხოვნა უბრუნდება აპლიკანტს, ან ამტკიცებს, რის შედეგადაც შესყიდვის ბრძანების მომზადების ქვეპროცესს ასრულებს შესყიდვების დეპარტამენტი და პროცესი სრულდება. ავაგოთ ბიზნესპროცესის ეს მოდელი (ნახ.30.31).



ნახ.30.31

პროცესის სიმულაციისთვის ძირითად მენიუში აირჩიეთ ღილაკი „Simulation View" , შედეგად მოდელი გადავა მხოლოდ კითხვის რეჟიმში (ნახ.30.32)



ნახ.30.32

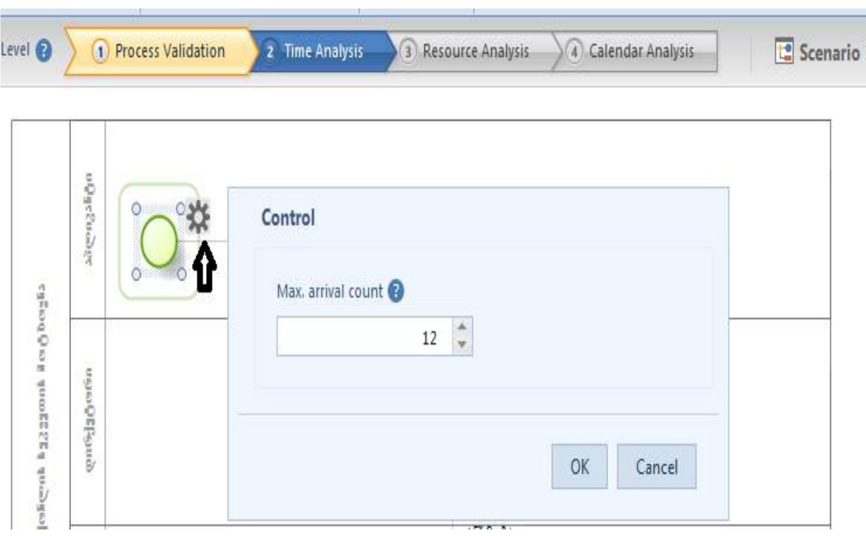
- **Bizagi-ის სიმულაციის დონეები**

სიმულაციის დონეების სტრიქონში მოცემულია ოთხი დონე (ნახ.30.33). სიმულაციის პირველ დონეზე (Process Validation) ხდება ბიზნესპროცესის შემოწმება და დარწმუნება იმაში რომ ყველა სიმბოლო გაივლის მიმდევრობის ნაკადს და მოქმედებს ისე, როგორც არის მოსალოდნელი.

ნახ.30.33

ამ დონეზე შრომითი, მატერიალური დროითი რესურსების პარამეტრების მითითება არ არის საჭირო, ეს დაგვჭირდება შემდგომ დონეებში. Bizagi-ს სიმულაციის მოდული გვთავაზობს რეალურ დროში პროცესის შესრულების ანიმაციას იმისთვის, რომ მარტივად მოხდეს პრობლემის იდენტიფიცირება.

ამ დონეზე ხდება მხოლოდ გეითვეის და საწყისი ხდომილებისთვის ინფორმაციის მინიჭება. მოვნიშნოთ საწყისი ხდომილება და მაუსის ღილაკით ავირჩიოთ მისი მენიუ , შედეგად გამოჩნდება ახალი ფანჯარა, სადაც შევიყვანოთ Max. Arivval count-ის პარამეტრი სურვილისამებრ. (სიმულაცია დასრულდება მაშინ, როდესაც სცენარის ხანგძლივობა მიაღწევს Max. Arivval count პარამეტრს) (ნახ.30.34)

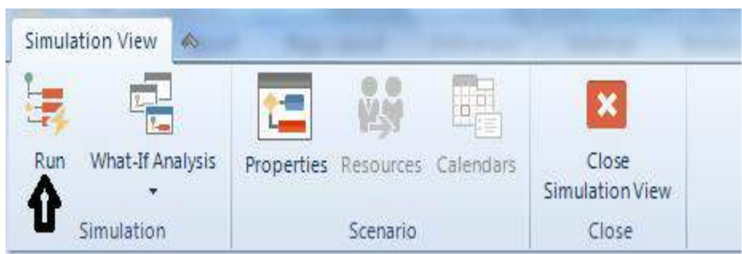


ნახ.30.34

გეითვეისთვის მონაცემების მინიჭება: გეითვეის აქვს აქტივაციის ალბათობა, რომელიც განისაზღვრება 0% - დან 100% მდე. მოვნიშნოთ გეითვეი, მაუსის ღილაკით ავირჩიოთ მისი მენიუ და ღილაკის დახმარებით დავაყენოთ სასურველი ალბათობები (ნახ.30.35)

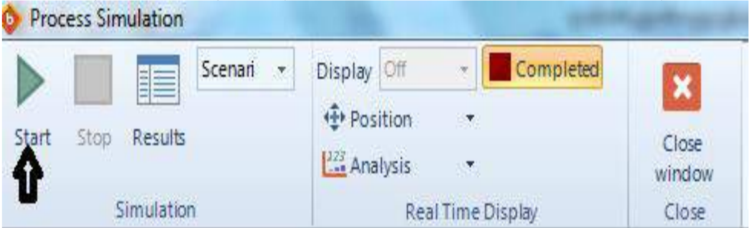
ნახ.30.35

მას შემდეგ, რაც მოთხოვნილი მონაცემები უკვე განსაზღვრულია, ავირჩიოთ **Run** ღილაკი სიმულაციის შესასრულებლად (ნახ.30.36).



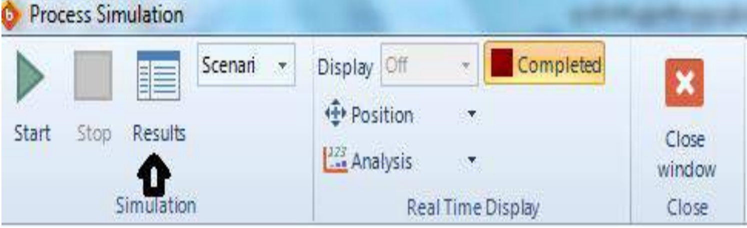
ნახ.30.36

სიმულაციის დასაწყებად ახალ ფანჯარაში ვირჩევთ **Start** ღილაკს (ნახ.30.37).



ნახ.30.37

როდესაც სიმულაცია დასრულდება, შედეგის ნახვისთვის ავირჩიოთ ძირითად მენიუში არსებული Results - ღილაკი (ნახ.30.38). შედეგი გადავიყვანოთ ექსელში, დავხუროთ შედეგების ფანჯარა და სიმულაციის ძირითად მენიუში ავირჩიოთ ღილაკი Close Window. დავბრუნდებით Bizagi-ს სიმულაციის ძირითად ფანჯარაში.



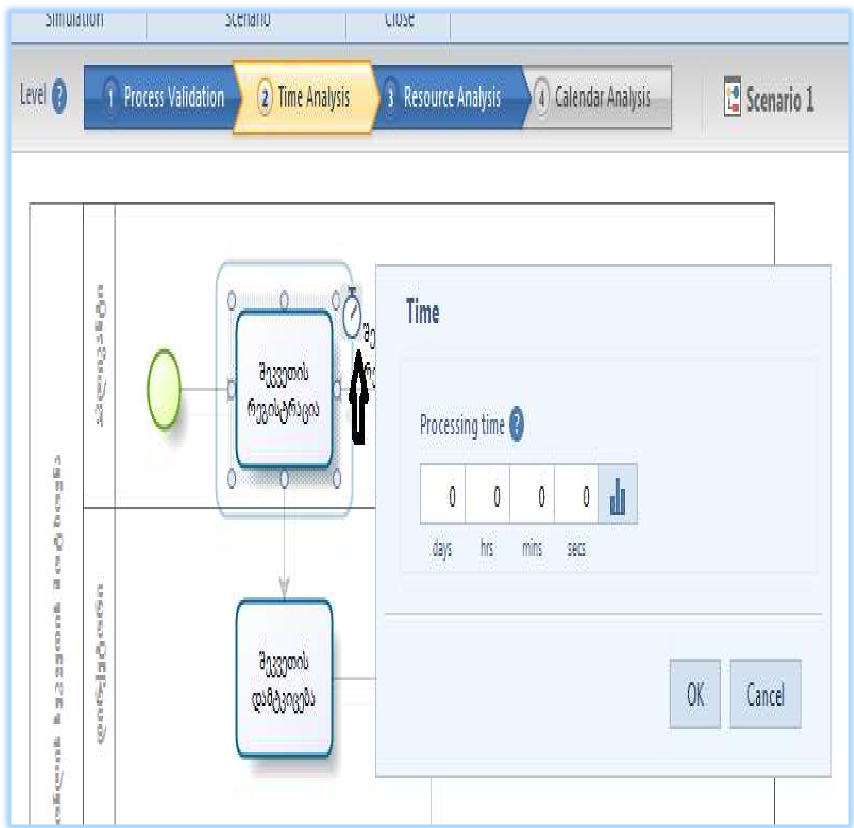
ნახ.30.38

ავირჩიოთ სიმულაციის მეორე დონე (**Time Analysis**), როგორც 30.39 ნახაზზეა მოცემული. დროის ანალიზის დახმარებით გავზომავთ დროს, რომელიც სჭირდება პროცესს დასასრულებლად. მონაცემები, რომლებიც უნდა შევიტანოთ, არის სავარაუდო დრო თითოეული ამოცანისთვის. სიმულაციის შედეგად მივიღებთ სიმბოლოებისთვის მაქსიმალურ და მინიმალურ დროს.

ნახ.30.39

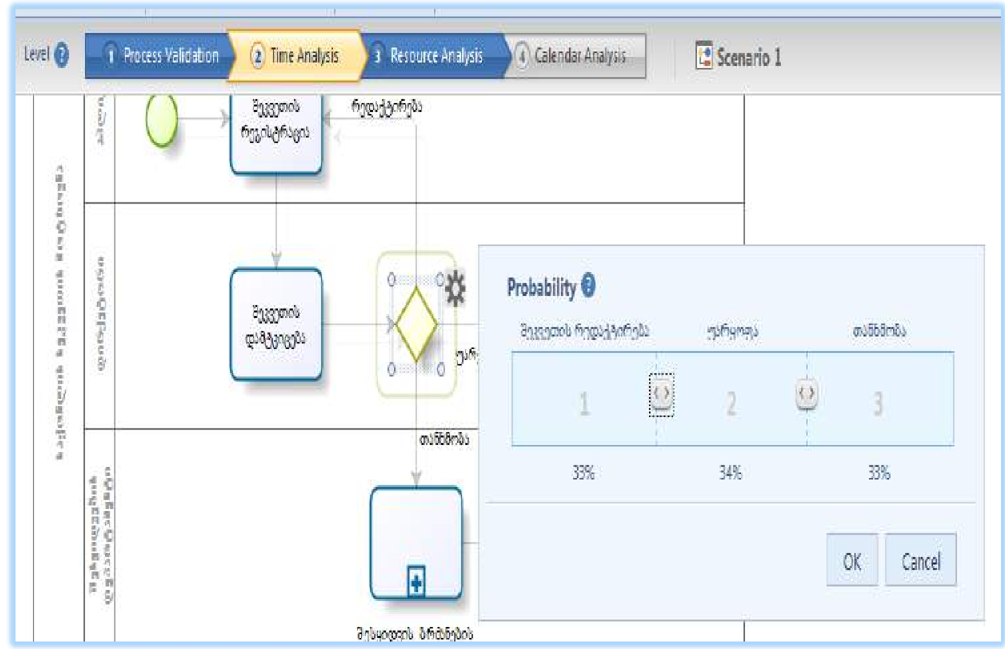
ამ დონის არჩევის შემდეგ თითოეულ საწყის მოვლენას და გეითვეის მარჯვენა ზედა კუთხეში გაუჩნდება ინსტრუმენტის ნიშანი, ხოლო მოქმედებათა ელემენტებს – საათის ნიშანი, მასზე მაუსის მარჯვენა ღილაკის მიტანით გამოჩნდება დროითი

რესურსის მითითების ფანჯარა, სადაც დრო შეგვიძლია მივუთითოთ დღეების, საათების, წუთების ან წამების საშუალებით (ნახ.30.40).



ნახ.30.40

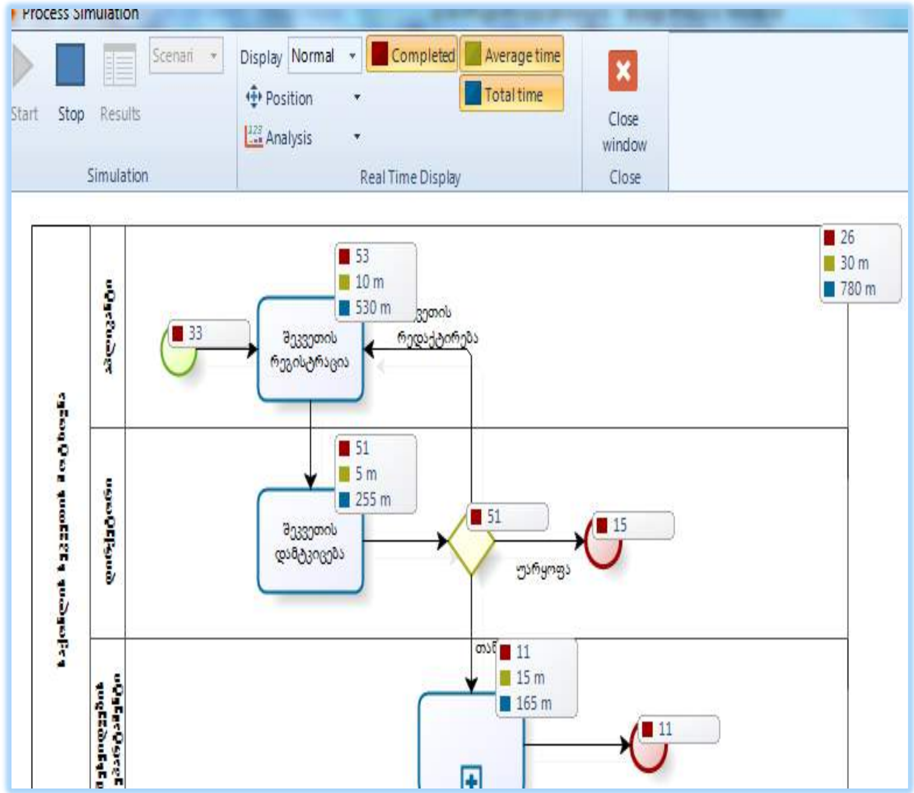
მოვნიშნოთ გეითვეი და მაუსის ღილაკით მივუთითოთ ინსტრუმენტების ნიშანს, შედეგად გამოჩნდება სამი შესაძლო მოქმედების მოხდენის ალბათობა, პროცენტულად თანაბრად გამოსახული (ნახ.30.41). მოვახდინოთ თანხმობის ალბათობის გაზრდა 50%-მდე და მივუთითოთ ღილაკს OK.



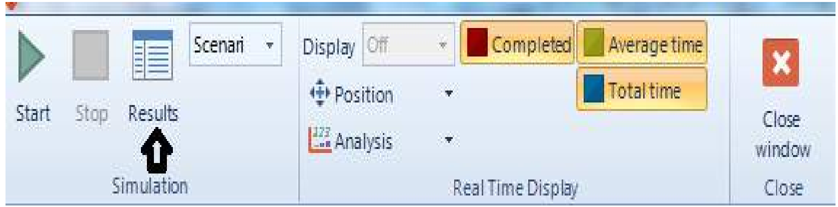
ნახ.30.41

როდესაც მოქმედებებზე და გეითვეიზე შევავსებთ მონაცემებს სიმულაციის დასაწყებად, ძირითად მენიუში ავირჩიოთ ღილაკი Run. გამოჩნდება ახალი ფანჯარა, სადაც მივუთითოთ ღილაკს Start. დაიწყება სიმულაცია და ეკრანზე გამოჩნდება პროცესის შესრულების ანიმაციური ხედი (ნახ.30.42).

ჩვენ შეგიძლია ნებისმიერ დროს მივუთითოთ Stop ღილაკს სიმულაციის შესაწყვეტად. შედეგის ნახვისთვის სიმულაციის ფანჯრის ძირითად მენიუში მივმართოდ ღილაკს Recults (ნახ.30.43).



ნახ.30.42



ნახ.30.43

გამოვა შედეგების ფანჯარა, სადაც არის მინიმალური საშუალო, მაქსიმალური და მთლიანი დრო. შეგვიძლია შედეგების დაბეჭვდა და ექპორტი ექსელში (ნახ.30.44).

Scenario information

Name Scenario 1

Time unit Minutes

Name	Type	Instances completed	Instances started	Min. time	Max. time	Avg. time
საქონლის სეკვეთის მოტხოვნა	Process	1 000	1 000	15m	2h	30m 1s
NoneStart	Start event	1 000				
შეკვეთის რეგისტრაცია	Task	1 533	1 533	10m	10m	10m
შეკვეთის დამტკიცება	Task	1 533	1 533	5m	5m	5m
ExclusiveGateway	Gateway	1 533	1 533			
NoneEnd	End event	531				
NoneEnd	End event	469				

ნახ.30.44

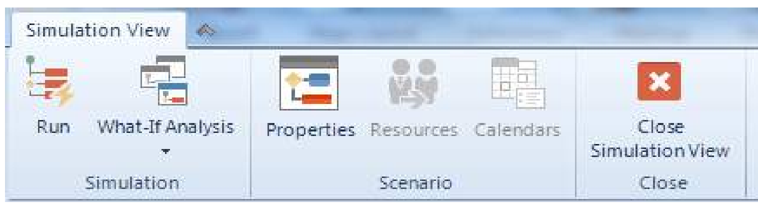
ავირჩიოთ close windows.

გავააქტიუროთ სიმულაციის მესამე დონე Resource Analysis (რესურსების ანალიზი) (ნახ.30.45). ამ დონეზე მოთხოვნილი მონაცემები არის მატერიალური და ადამიანური რესურსები, რომლებიც სჭირდება თითოეულ მოქმედებას. შედეგი არის ანალოგიური, როგორიც იყო დროითი ანალიზის დრო.

ნახ.30.45

მაუსის ღილაკით მოვნიშნოთ მოქმედება. გამოჩნდება მოქმედებაზე თანდართული პატარა მენიუ, რომლიდანაც ავირჩიოთ ღირებულების მენიუ.

სიმულაციის მოდულის დასასრულებლად ავირჩიოთ **Close Simulation View** (ნახ.30.46) ღილაკი.



ნახ.30.46

შედეგად გადავალთ მოდელერის ძირითად ფანჯარაში და სანამ დავხურავთ მოდელერს შევინახოთ ცვლილებები **File/Save.**

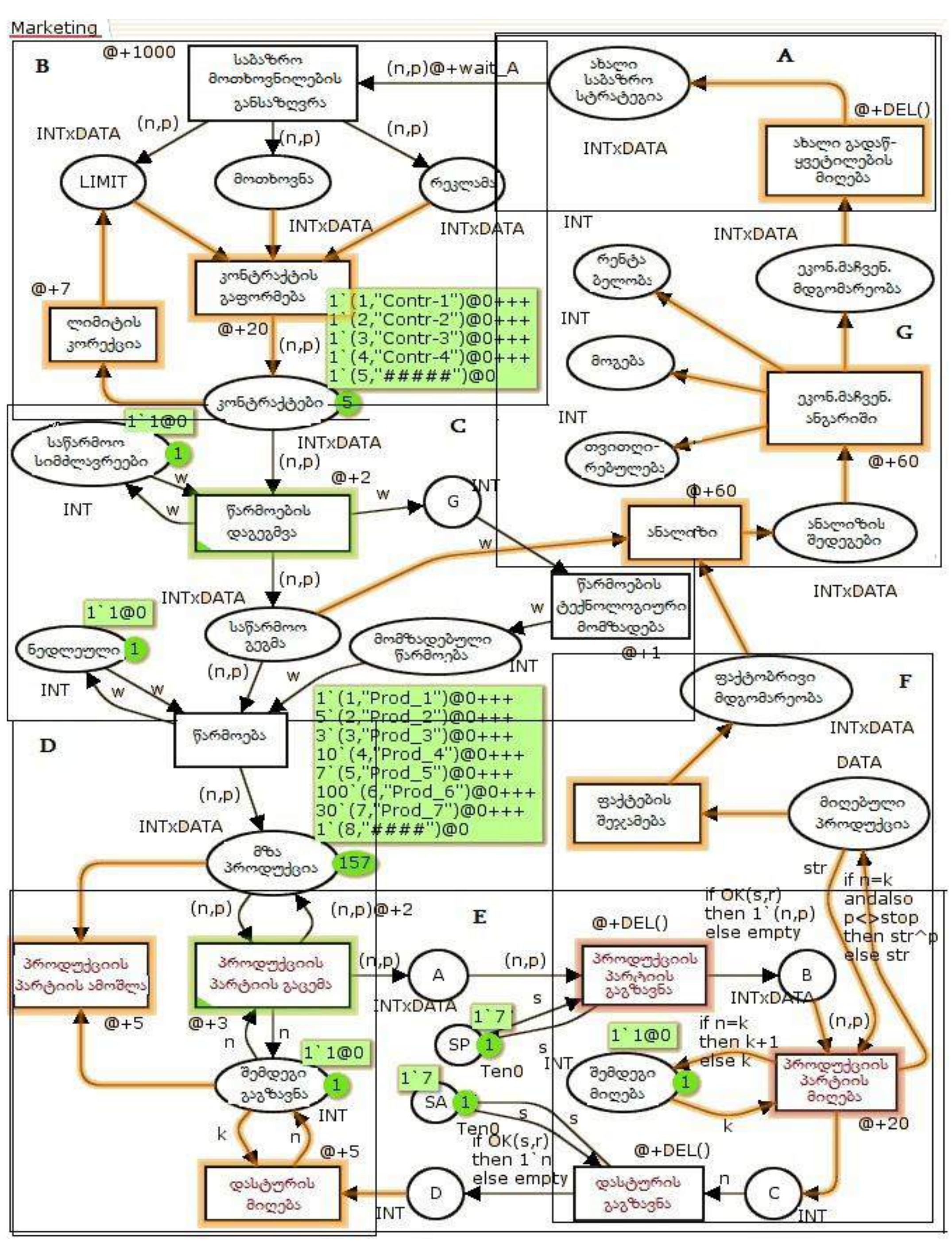
30.4. მარკეტინგული პროცესების მოდელირება პეტრის ფერადი ქსელებით

განვიხილოთ მარკეტინგული პროცესების მოდელირების საილუსტრაციო მაგალითი პეტრის ფერადი ქსელების (CPN-Coloured Petri Net) ინსტრუმენტის გამოყენებით [41,61,62].

პროდუქციის საწარმოო ფირმის მარკეტინგული პროცესების მოდელირებისათვის გვაქვს შემდეგი ძირითადი იერარქიული მოდულები (ნახ.30.47): ახალი საბაზრო სტრატეგიის ფორმირება (A), საბაზრო მოთხოვნების განსაზღვრა (B); პროდუქციის წარმოების დაგეგმვა (C); წარმოების ტექნიკური მომზადება და პროდუქციის წარმოება (D); პროდუქციის გაცემა (სასაწყობო მეურნეობა) და პროდუქციის გადაგზავნა (ტრანსპორტირება) (E), პროდუქციის მიღება და დამკვეთის ინფორმირება (F); ფაქტობრივი მდგომარეობის აღრიცხვა, საწარმოო და სარეალიზაციო გეგმების შესრულების ანალიზი, ეკონომიკური მაჩვენებლების ანგარიში და ანალიზი (G); გადაწყვეტილების მიღება ახალი საბაზრო სტრატეგიისათვის (A) და ა.შ. ციკლურად.

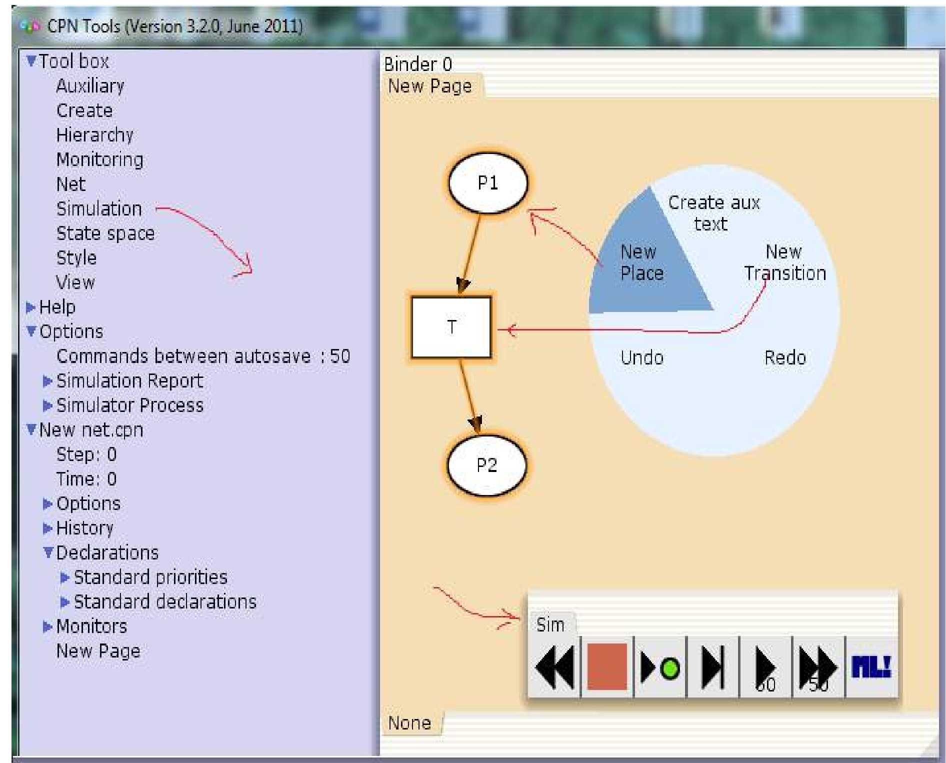
ჩვენი მიზანია ზემოაღწერილი იერეარქიული მოდულებიდან გამოვყოთ, მაგალითად, E ბლოკი, რომელიც აღწერს პროდუქციის მიწოდების პროცესს კლიენტებზე და მოვახდინოთ „მიმწოდებელი–დამკვეთი" პროცესის მოდელირება შეტყობინებების გაცვლის იმიტაციით, კლიენტ–სერვერ არქიტექტურის პრინციპების საფუძველზე (შემდეგ კი დავაპროგრამოთ ეს ბიზნესპროცესი).

მოკლედ განვიხილოთ CPN ინსტრუმენტის სამუშაო გარემო, მისი დახმარებით მოდელის აგების და შემდეგ ბიზნესპროცესის იმიტაციური რეჟიმის ამოქმედება.



ნახ.30.47. მარკეტინგული პროცესების პეტრის ქსელის მოდელი (CPN)

30.48 ნახაზზე ნაჩვენებია CPN-ის მომხმარებლის საწყისი ინტერფეისი (ვუშვებთ, რომ CPN პაკეტი დაინსტალირებულია კომპიუტერში [62]. იგი უფასო ვერსიაა და ფართოდ გამოიყენება აშშ, ევროპის, ჩინეთისა და სხვა ქვეყნების უნივერსიტეტებში).



ნახ.30.48. CPN სამუშაო გარემო

პეტრის ქსელის პოზიციების (Places), გადასასვლელების (Transitions) და რკალების (Arcs) აგება (მაუსის მარჯვენა ღილაკით), შემდეგ მარკერების დამატება და იმიტაციური პროცესის (Simulation) ამუშავება მარტივად ხორციელდება.

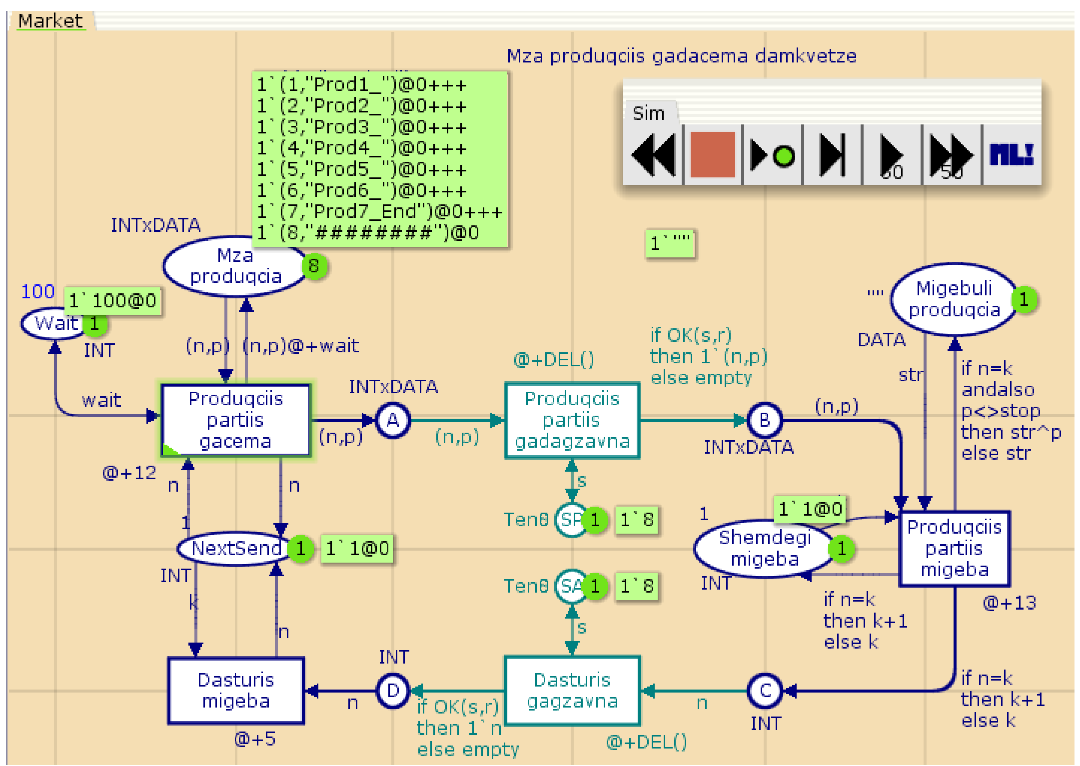
ნახაზის მარცხენა ნაწილში ჩანს პეტრის ქსელის კონკრეტული პარამეტრების მნიშვნელობები. მაგალითად, საწყის მდგომარეობაში პოზიცია „მზა პროდუქცია" შეიცავს INTxDATA ტიპის ფერად მარკერთა 8-ელემენტიან სიმრავლეს (საინიციალიზაციო მარკირება) (ნახ.30.49).

30.50 ნახაზზე ნაჩვენებია აგებული E ბლოკის (მარტივი შემთხვევის) მაგალითი, რომლისათვისაც ჩავატარეთ კონკრეტული ექსპერიმენტები.

{1`(1,"Prod1"), 1`(2, "Prod2"), 1`(3, "Prod3"), 1`(4, "Prod4"), . . . , 1`(8, „##### ") }. აქ ბოლო, მე-8 ელემენტი შეესაბამება დასასრულის იდენტიფიკაციას -Stop.

```
▼Marketing-4.cpn
   Step: 0
   Time: 0
  ▶History
  ▼Declarations
    ▼colset INT =int timed;
    ▼colset DATA = string;
    ▼colset INTxDATA = product INT*DATA timed;
    ▼var n, k, w, d, wait: INT;
    ▼var p, str: DATA;
    ▼val wait_A=8760;
    ▼val stop = "#####";
    ▼colset Ten0 = int with 0..12;
    ▼colset Ten1 =int with 1..12;
    ▼var s: Ten0;
    ▼var r: Ten1;
    ▼fun OK(s:Ten0, r:Ten1) = (r<=s);
    ▼colset NetDelay = int with 25..75;
    ▼fun DEL() = NetDelay.ran();
```

ნახ.30.49. „პროდუქციის მიწოდების“ ბიზნესპროცესის პეტრის ქსელის ფრაგმენტი (საწყისი პოზიცია)



ნახ.30.50. CPN-ის პროგრამული ენის ფრაგმენტი

„1“-იანი ყოველი ელემენტის დასაწყისში (მას კოეფიციენტი ეწოდება), რომელიც მიუთითებს, რომ პოზიციაშია არაუმეტეს 1 ცალი მოცემული ფერის მონაცემი (ანუ არსებობს მხოლოდ ერთი პროდუქტი ნომრით „Prod1“, რომლის ფერი - რიგითი ნომერია 1). ამ შემთხვევაში გვაქვს მონაცემთა ელემენტების *სიმრავლე.*

30.47 ნახაზზე ნაჩვენები E ბლოკის რთული შემთხვევისათვის გვექნებოდა 157 ელემენტი (1+5+3+10+7+100+30+1), რომლებიც 7 სხვადასხვა (მარკერების ფერის) დამზადებული პროდუქტის რაოდენობას, ანუ *მულტისიმრავლეს* ასახავს.

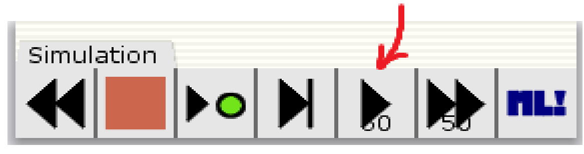
30.50 ნახაზზე პროცესების შესრულების დრო (დაყოვნება) აისახება გადასავლელთან სიმბოლოს და დროის ერთეულის (მაგალითად, @+12, @+wait) მითითებით, სადაც wait წინასწარ განსაზღვრული კონსტანტაა.

ამავე ნახაზზე ასახულია არადეტერმინირებული ლოგიკური გამოსახულება (პირობის ბლოკი) ფერადი პეტრის ქსელის რკალებზე, რომელიც გადასასვლელთა გაშვების სხვადასხვა პირობებს და შედეგებს ასახავს, ანუ ლოგიკური პირობის ჭეშმარიტებისას გადასასვლელს განსხვავებული მნიშვნელობა მიეწოდება (ან გადასასვლელიდან განსხვავებული მნიშვნელობა გამოვა), მცდარობისას – განსხვავებული.

მაგალითად, გადასასველს „პროდუქციის პარტიის გადაგზავნა“ გამოსასვლელ რკალზე აქვს ლოგიკური პირობა - თუ გამოგზავნილი პროდუქციის ნომერი (n) ემთხვევა კლიენტის კონტრაქტით მისაღებ პროდუქციის ნომერს (k), მაშინ გვაქვს “true”, წინააღმდეგ შემთხვევაში ”false“, რაც იმას ნიშნავს, რომ საჭირო პროდუქცია არაა მოსული. თუ ყველაფერი წესრიგშია, მაშინ მიმღები უგზავნის მწარმოებელს შეტყობინებას გადასასვლელით „დასტურის გამოგზავნა“.

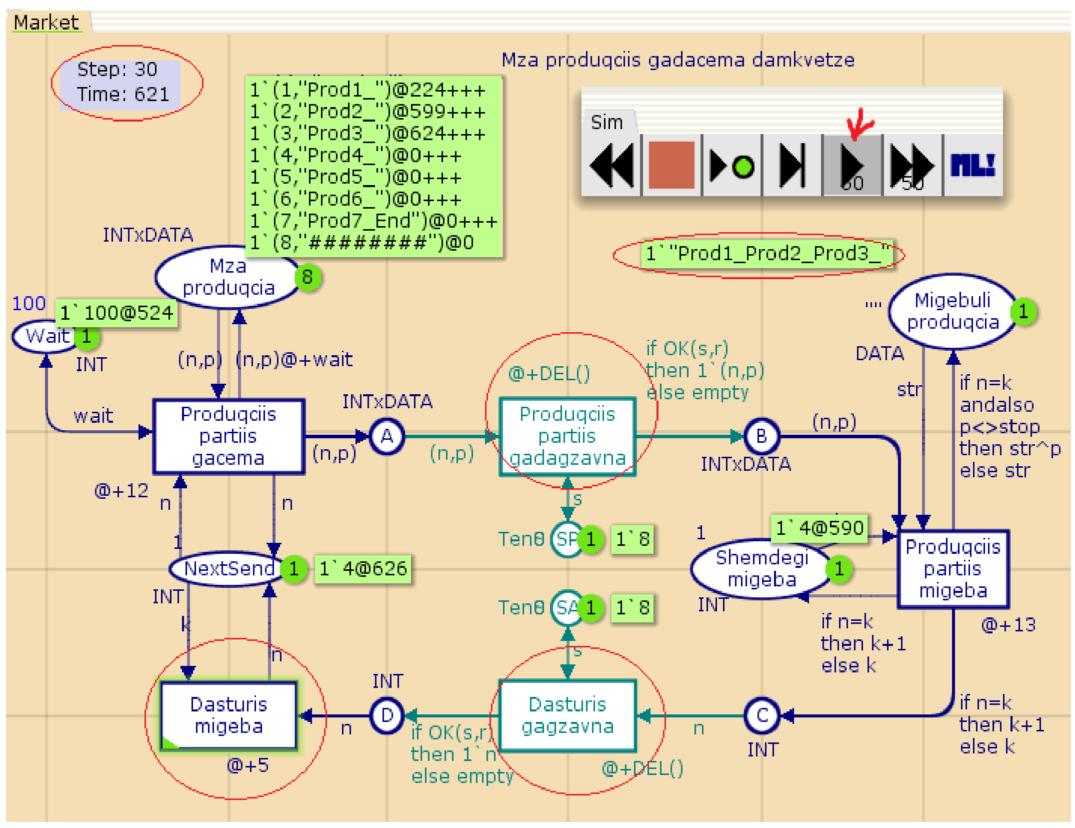
პროდუქციის და შეტყობინების გადაცემათა ქსელში შემთხვევითი პროცესის არსებობა განპირობებულია დაყოვნების ცვლადი დროის გამო, რაც აისახება colset NetDelay=int with 25..75, fun DEL( ) =NetDelay.ran( ) random-ფუნქციით. ლოგიკური პირობის მნიშვნელობა სხვადასხვა შემთხვევებში სხვადასხვანაირად განისაზღვრება. ინტერაქტიულ სიმულატორებში ჭეშმარიტება-მცდარობას თავად მომხმარებელი განსაზღვრავს, ავტომატური სიმულაციისას – შემთხვევით სიდიდეთა გენერატორი.

30.51 ნახაზზე სიმულატორის ღილაკებით იმართება იმიტაციური პროცესი და მიიღება შუალედური და საბოლოო შედეგბი.



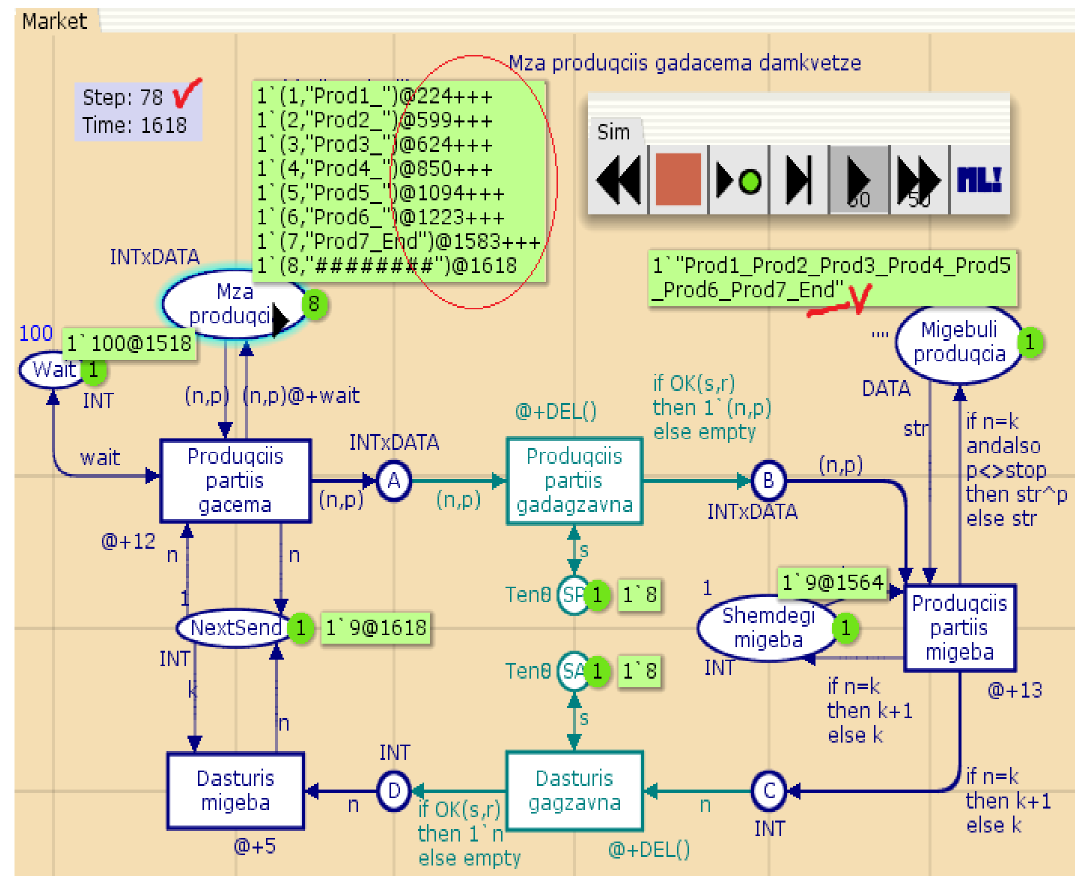
ნახ.30.51

მაგალითად, 30.52 ნახაზზე ნაჩვენებია პროცესის დინამიკის ფრაგმენტი 30-ე ბიჯზე. აქ მიმდევრობით ხდება „ფერადი მარკერების" (სხვადასხვა სახის პროდუქციის) გაგზავნა დამკვეთებზე. წითელი წრეხაზებით აღნიშნული გვაქვს პარალელურად შესრულებადი პროცესები (გადასასვლელების გახსნა). სქემაზე ჩანს ასევე დროის მომენტები და დამკვეთთან უკვე მისული პროდუქციის დასახელებები.



ნახ.30.52. იმიტაციური პროცესი (შუალედური ბიჯი = 30)

30.53 ნახაზზე ნაჩვენებია დასრულებული ბიზნესპროცესი, ანუ ყველა შეკვეთილი პროდუქტი გადაცემულია დამკვეთზე, რაც ვიზუალურად ფიქსირდება "End"-ით.



ნახ.30.53. პროცესი დასრულდა (ბიჯი = 78)

30.5. CPN ქსელის მდგომარეობათა სივრცის ანალიზი პროდუქციის რეალიზაციის პროცესისათვის

მარკეტინგის პროცესისთვის მზა პროდუქციის დამკვეთებზე მიწოდების (რეალიზაციის) გეგმის შესრულების (აღრიცხვის) შესაბამისი CPN პეტრის ქსელის ანალიზის ამოცანა წყდება მისი მდგომარეობათა სივრცის საფუძველზე.

მდგომარეობათა სივრცე (State Space) არის კვლევის ობიექტის შესაბამისი მოდელის ყველა შესაძლო მდგომარეობის ერთობლიობა. თვით მდგომარეობა, როგორც ეს პეტრის კლასიკურ ქსელებშია მიღებული, ასახავს მარკერთა განაწილებას ქსელის პოზიციების მიხედვით, ანუ მარკირებებს [60]. ქსელის რომელიმე გადასასვლელის ამუშავების (გაშვების) შემდეგ ხდება მის შესასვლელ და გამოსასვლელ პოზიციებში მარკერთა რაოდენობის ცვლილებები. ამ დროს ქსელი გადადის ახალ მდგომარეობაში.

ასეთი პროცესი შეიძლება რომელიმე ბიჯზე დაიბლოკოს, ანუ ჩიხში შევიდეს, რაც იმის მაუწყებელია, რომ ასეთი მოდელი და მისი შესაბამისი რეალური ობიექტი ვერ მიაღწევს მიზანს, საბოლოო შედეგს. ამგვარად ქსელი ყოფილა არასაკმარისად მდგრადი და იგი მოითხოვს კორექტირებას.

ჩვენ შემთხვევაში საქმე გვაქვს დამკვეთებზე მზა პროდუქციის მიწოდებასთან, რომლის გეგმაც კონტრაქტების საფუძველზე იქნა შედგენილი და მისი შესრულება აუცილებელია (რათა არ მოხდეს ხელშეკრულების დარღვევასთან დაკავშირებული საჯარიმო სანქციების ამოქმედება).

ჩვენი მოდელის ფრაგმენტის საფუძველზე, რომელიც წინა პარაგრაფში განვიხილეთ, ხდება მზა პროდუქციის გაცემა საწყობიდან, შემდეგ ტრანსპორტირება და დამკვეთამდე მიტანა. დამკვეთი, პროდუქციის მიღებისთანავე აგზავნის დასტურის შეტყობინებას და მიმწოდებელი ამის შემდეგ ზრუნავს მომდევნო პარტიის დამზადებასა და მიწოდებაზე.

არაა გამორიცხული შემთხვევები, რომ პროდუქციის პარტია ვერ მივიდეს დროულად დამკვეთთან (გარკვეული ობიექტურ-სუბიექტური მიზეზების გამო), ან დაიკარგოს დასტურის შეტყობინება. ასეთ შემთხვევებში საჭიროა ინფორმაციის დროულად გამოკვლევა და არშესრულებული პროცედურის გამეორება.

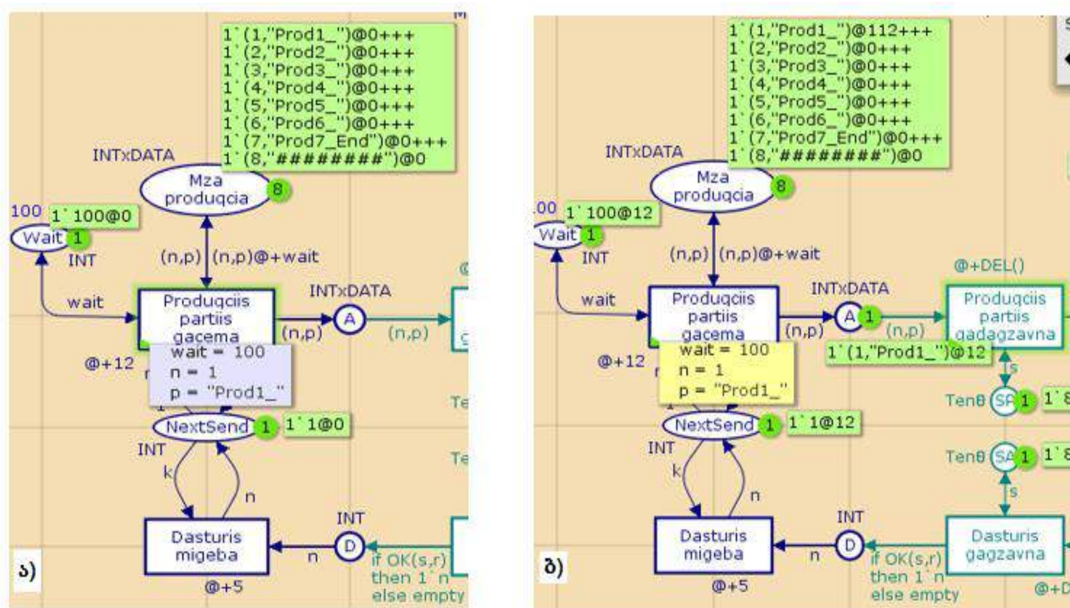
ფერადი პეტრის ქსელის გადასასვლელები, როგორებიცაა Produqciis partiis gacema, Produqciis partiis gadagzavna, Produqciis partiis migeba, dasturis gagzavna და ა.შ. ხასიათდება დროითი დაყოვნებებით, რომლებიც ან კონსტანტური მნიშვნელობისაა, ან შემთხვევითი რიცხვების დიაპაზონიდან აიღება სისტემის მიერ.

ამგვარად, CPN-ინსტრუმენტით შესაძლებელია მდგომარეობათა სივრცის ანგარიშის მთლიანი პროცესის სრული ავტომატიზაცია, რაც მნიშვნელოვნად აჩქარებს ქსელის დიაგნოსტიკის პროცესს მისი რეალურ ობიექტთან ადეკვქტურობის შესახებ, ანუ რამდენად სწორად ასახავს მოდელი რეალური ობიექტის ყოფაქცევას.

მდგომარეობათა სრული სივრცე აისახება ორიენტირებული გრაფით, რომელშიც მწვერვალები შეესაბამება ქსელის დასაშვებ მარკირებებს, ხოლო რკალები – მოვლენებს დამაკავშირებელი ელემენტებით. ე.ი. M1 მდგომარეობიდან (მარკირებიდან) სისტემა გადადის M2 მდგომარეობაში, როდესაც არსებობს რკალი დამაკავშირებელი (n,p)-ელემენტით, სადაც n - ფერადი მარკერია, ხოლო p - ინფორმაციული ნაწილი.

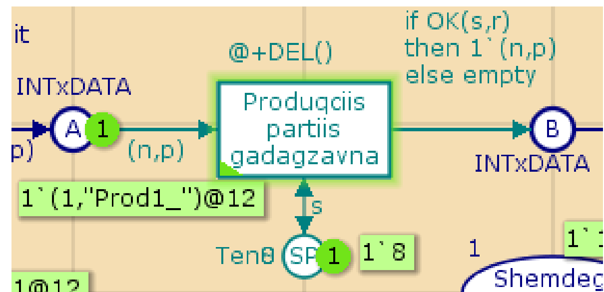
30.54 ნახაზზე ნაჩვენებია პეტრის ქსელის საწყისი მდგომარეობის ფრაგმენტი {n=1, p="Prod1"} ელემენტით (ა), ხოლო (ბ) შეესაბემება პეტრის ქსელის ახალ მარკირებას პირველი ბიჯის შემდეგ. აქ შესამჩნევია, რომ A-პოზიციაში გაჩნდა ახალი, 1 მარკერი, რომლის ფერი=1, მონაცემი="Prod1". ამასთანავე ეს მარკერი მოვიდა ქსელის ამუშავებიდან t=12 დროითი ერთეულის (მაგ., წუთი) შემდეგ (ვინაიდან Produqciis partiis gacemis გადასასვლელის დროითი დაყოვნებაა @+12).

ახლა გააქტიურდა Produqciis partiis gadagzavnis გადასასვლელი და შესაძლებელია ასევე Produqciis partiis gacemis გადასასვლელის ხელახალი გაშვებაც. ეს ორივე პროცესი შეიძლება შესრულდეს პარალელურად, ისინი ერთმანეთს ხელს არ უშლის.



ნახ.30.54. ა) საწყისი მარკირება და ბ) მარკირება პირველი ბიჯის შემდეგ

30.55 ნახაზზე ნაჩვენებია Produqciis partiis gadagzavnis გადასასვლელის აქტიური მდგომარეობა. აქ მარკერები არის A და SP პოზიციებშიც.



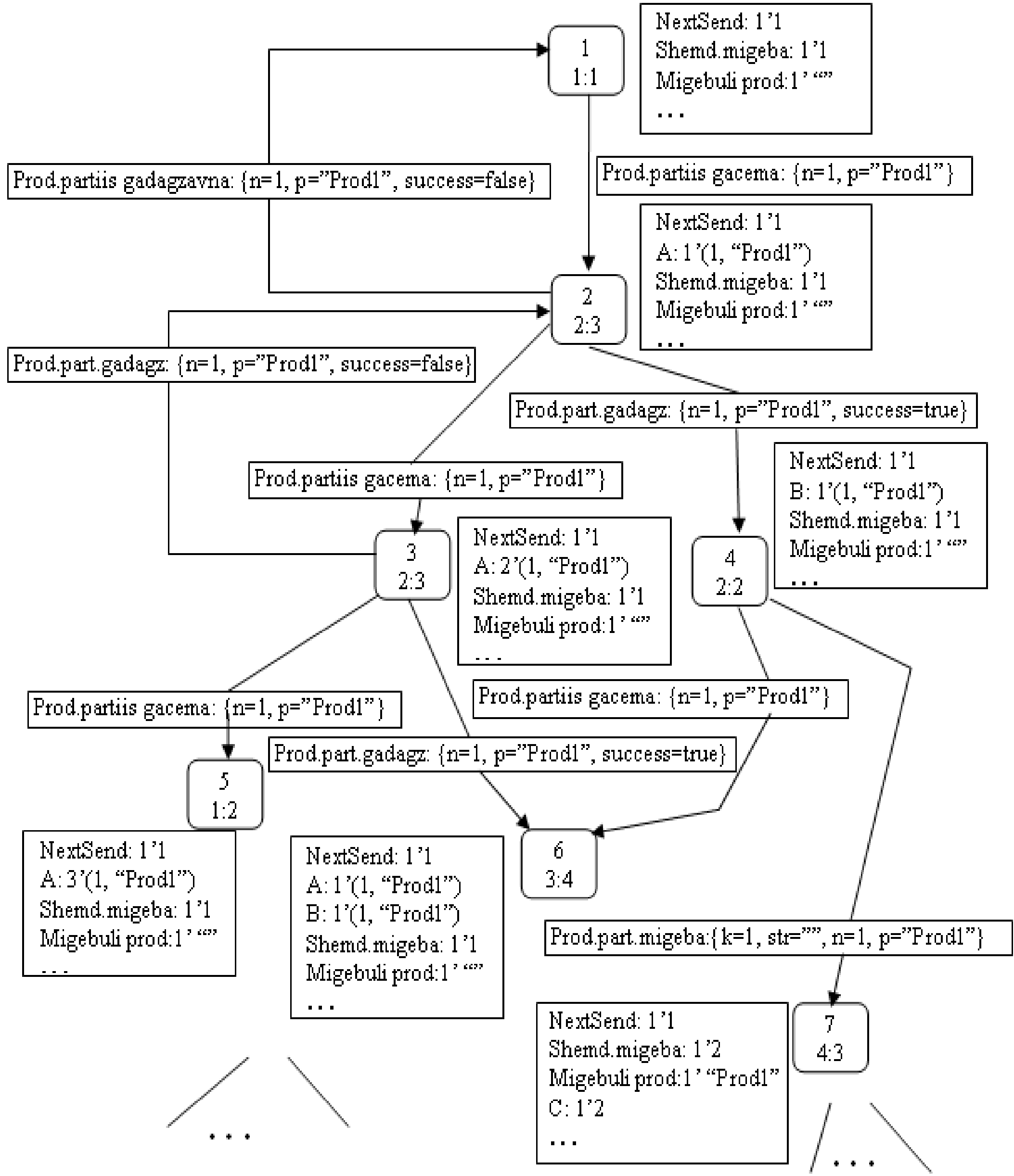
ნახ.30.55

ამ გადასასვლელიდან B–პოზიციაში შემავალი რკალი ლოგიკურ პირობას აკონტროლებს, ანუ დასაშვებია ორი შემთხვევა:

TP+ = (Produqciis_partiis_gadagzavna, <n=1,p=”Prod1”, success=true>),

TP- = (Produqciis_partiis_gadagzavna, <n=1,p=”Prod1”, success=false>).

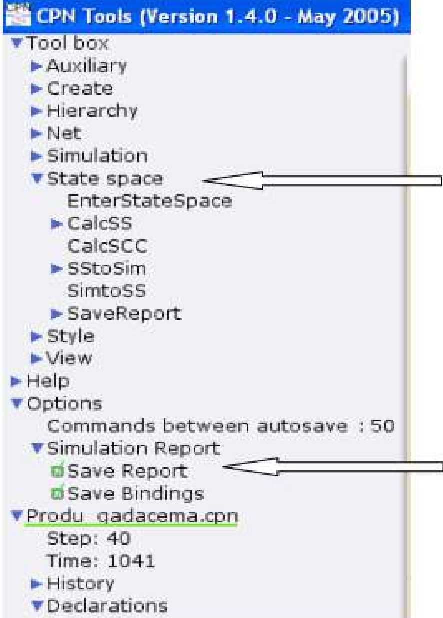
30.56 ნახაზზე ნაჩვენებია აღწერილი პროცესის შესაბამისად ჩვენი ქსელის მდგომარეობათა სივრცის ფრაგმენტი, რომელიც, როგორც აღვნიშნეთ, ორიენტირებული გრაფითაა წარმოდგენილი.



ნახ.30.56. მდგომარეობათა სივრცის ფრაგმენტი CPN-მოდელისათვის

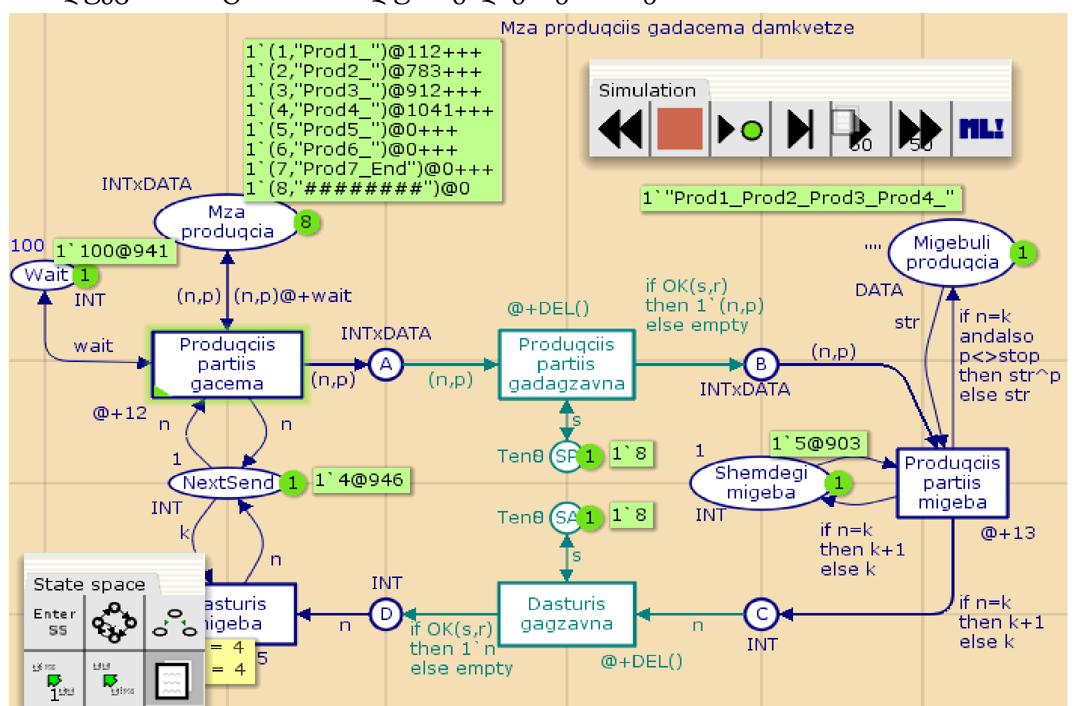
30.57 ნახაზზე მოცემულია იმიტაციური მოდელირებისა (სიმულაციის) და მდგომარეობათა სივრცის ანალიზის რეპორტების მომზადების ინსტრუმენტი. აღნიშნული CheckBox-ის ჩართვის შემთხვევაში,

რეპორტები ავტომატურად მოთავსდება C:\temp საქაღალდეში, რომელიც წინასწარ უნდა შეიქმნას.



ნახ.30.57

30.58. ნახაზზე მოცემულია CPN-ქსელის ფუნქციონირების ფრაგმენტი 4 პროდუქციის პარტიის ნორმალური გადაგზავნისათვის.



ნახ.30.58. State Space - ინსტრუმენტით

აქვე მოცემულია ჩვენ მიერ აგებული პეტრის ქსელის მოდელზე ჩატარებული ექსპერიმენტის შედეგედ მიღებული სტატისტიკური მონაცემების ლისტინგი:

```
Statistics    // მდგომარეობათა სივრცის სტატისტიკა
---------------------------------------------
 State Space   // მდგომარეობათა სივრცე
   Nodes:  55  // მწვერვალების რაოდენობა
   Arcs:   58  // რკალების რაოდენობა
   Secs:   0   // ერთი ბიჯის შემდეგ დრო 0-ია
   Status: Partial  // ნაწილობივაა შესრულებული
 Scc Graph    // მკაცრად დაკავშირებულ ელემენტთა გრაფი
    Nodes:  55
    Arcs:   58
    Secs:   0
Boundedness Properties // მარკერების რაოდენობა პოზიციებში
                         (max, min)
---------------------------------------------
  Best Integers Bounds  Upper Lower
  Market'A 1              1     0 //არის 1  ან 0 მარკერი
  Market'B 1              1     0
  Market'C 1              1     0
  Market'D 1              1     0
  Market'Migebuli_produqcia 1
                          1   1
  Market'Mza_produqcia 1  8   8 //ყოველთვის 8 მარკერია
  Market'NextSend 1       1   1 // ყოველთვის 1 მარკერია
  Market'SA 1             1   1
  Market'SP 1             1   1
  Market'Shemdegi_migeba 1
                          1   1 // ყოველთვის 1 მარკერია
  Market'Wait 1           1   1
Best Upper Multi-set Bounds //მულტი-სიმრავლის კავშირები
Market'A 1          1`(4,"Prod4_") // გადაცემულია მე-4
Market'B 1          1`(4,"Prod4_") // გადაცემულია მე-4
Market'C 1          1`5          // შემდეგი უნდ იყოს მე-5
Market'D 1          1`5          // შემდეგი უნდ იყოს მე-5
Market'Migebuli_produqcia 1
      1`"Prod1_Prod2_Prod3_Prod4_"
Market'Mza_produqcia 1 // აქ ყოველთვის 7 ფერის პროდუქციაა
    1`(1,"Prod1_")++
    1`(2,"Prod2_")++
    1`(3,"Prod3_")++
    1`(4,"Prod4_")++
    1`(5,"Prod5_")++
    1`(5,"Prod5_")++
    1`(6,"Prod6_")++
```

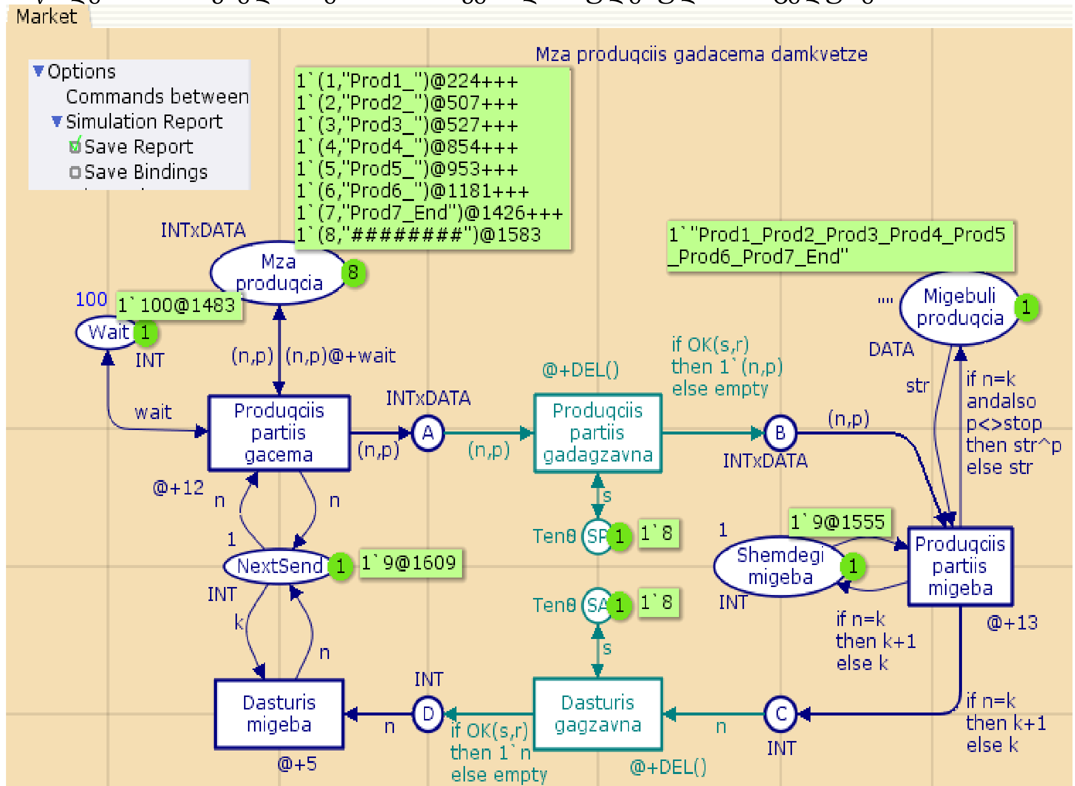
```
     1`(7,"Prod7_End")++
     1`(8,"########")
Market'NextSend 1    1`4++      // აქ ბოლო იყო მე-4 და
                     1`5        // შემდეგი უნდა იყოს მე-5
Market'SA 1          1`8   // აქ ყოველთვის არის 8 მარკერი
Market'SP 1          1`8   // აქ ყოველთვის არის 8 მარკერი
Market'Shemdegi_migeba 1
                     1`5   //  შემდეგი უნდა იყოს მე-5
Market'Wait 1        1`100 // დაყოვნების დრო მუდმივია: 100

Best Lower Multi-set Bounds
Market'A 1       empty //ცარიელია (ანუ პოზიციაში შეიძლება
Market'B 1       empty        // არ იყოს მარკერი)
Market'C 1       empty
Market'D 1       empty
Market'Migebuli_produqcia 1
             1`"Prod1_Prod2_Prod3_Prod4_"
Market'Mza_produqcia 1
     1`(1,"Prod1_")++
     1`(2,"Prod2_")++
     1`(3,"Prod3_")++
     1`(4,"Prod4_")++
     1`(5,"Prod5_")++
     1`(6,"Prod6_")++
     1`(7,"Prod7_End")++
     1`(8,"########")
Market'NextSend   1   empty
Market'SA         1    1`8
Market'SP         1    1`8
Market'Shemdegi_migeba 1  1`5
Market'Wait       1    1`100

Home Properties
---------------------------------------------
Home Markings:  Initial Marking is not a home marking
Liveness Properties
--------------------------------------------------
Dead Markings:  33 [55,54,53,52,51,...]
Dead Transitions Instances: None
Live Transitions Instances: None
Fairness Properties
--------------------------------------------------
  No infinite occurrence sequences.
```

აღნიშნული სტატისტიკა ემსახურება ქსელის მუშაობის პროცესის ისტორიის დამახსოვრებას, პოზიციების მდგომარეობათა, გადასასვლელების გაშვებებისა და მარკერების მოძრაობის შესახებ. ისინი შემდგომი ანალიზისთვის გამოიყენება.

30.6. იმიტაციური მოდელირების პროცესის ლისტინგი პროდუქციის მიწოდების CPN ქსელისათვის

კომპანიის წარმოების ეფექტიანად ფუნქციონირებისა და საბაზრო-ეკონომიკურ პირობებში მისი სტაბილური არსებობისთვის ერთ-ერთი მთავარი ბირთვია მარკეტინგული სამსახური. იგი იკვლევს პროცესებს, თუ რამდენად ეფექტურად ხორციელდება ბიზნესის პრაქტიკულად ყველა რგოლი და მოიცავს მისაღები ზომების კომპლექსს, რაც უზრუნველყოფს პროდუქციის კონკურენტუნარიანი მდგომარეობის შექმნას ბაზარზე. ასეთი პროცესების იმიტაციური მოდელირება და ანალიზი ხელს უწყობს კომპიუტერული ექსპერიმენტების ჩატარებას და გარკვეული დასკვნების გამოტანას გადაწყვეტილების მიღების პროცესში.

30.59 ნახაზზე მოცემული გვაქვს მარკეტინგის დეპარტამენტის პროდუქციის მიწოდების CPN-ქსელის შესაბამისი სქემა დასრულებული პროცედურებით.



ნახ.30.59. საბოლოო მდგომარეობა და იმიტაციის შედეგად
Save Report-ის მიღება

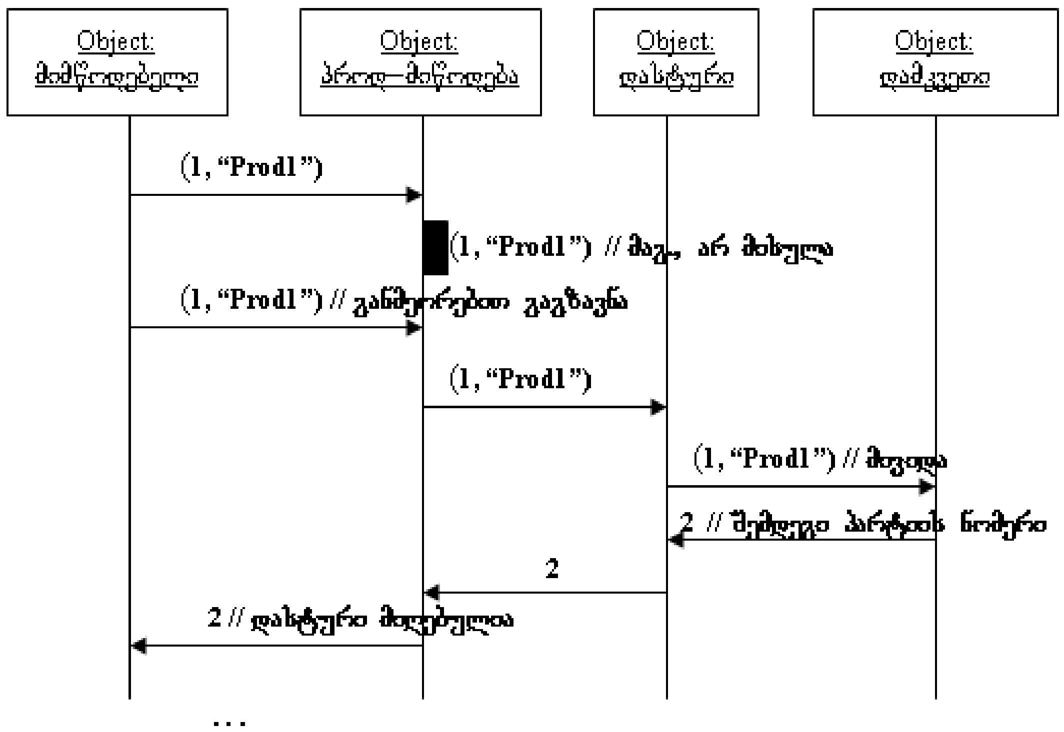
ქვემოთ მოცემულია ჩვენი ქსელის მუშაობის ამსახველი იმიტაციური პროცესის რეპორტი:

იმიტაციური პროცესის ფრაგმენტის Save Report-ის ლისტინგი:

```
ბიჯი  დრო         გადასასვლელი
1     0     Produqciis_partiis_gacema @ (1:Market)
 - wait = 100
 - n = 1
 - p = "Prod1_"

2     12    Produqciis_partiis_gadagzavna @ (1:Market)
 - n = 1
 - p = "Prod1_"
 - s = 8
 - r = 9
3     112   Produqciis_partiis_gacema @ (1:Market)
 - wait = 100
 - n = 1
 - p = "Prod1_"
4     124   Produqciis_partiis_gadagzavna @ (1:Market)
 - n = 1
 - p = "Prod1_"
 - s = 8
 - r = 6
5     158   Produqciis_partiis_migeba @ (1:Market)
 - str = ""
 - k = 1
 - n = 1
 - p = "Prod1_"
6     171   Dasturis_gagzavna @ (1:Market)
 - n = 2
 - s = 8
 - r = 4
7     206   Dasturis_migeba @ (1:Market)
 - k = 1
 - n = 2
8     211   Produqciis_partiis_gacema @ (1:Market)
 - wait = 100
 - n = 2
 - p = "Prod2_"
9     223   Produqciis_partiis_gadagzavna @ (1:Market)
 - n = 2
 - p = "Prod2_"
 - s = 8
 - r = 4
. . . . . . . . .  და ა.შ.

81    1576  Dasturis_gagzavna @ (1:Market)
 - n = 9
 - s = 8
 - r = 2
82    1619  Dasturis_migeba @ (1:Market)
 - k = 9
 - n = 9
```

ამგვარად, იმიტაციური პროცესი მოიცავს 82 ბიჯს.

აღნიშნული პროცესების შესრულება უნიფიცირებული მოდელირების ენის UML-ტექნოლოგიაში მოგვაგონებს შეტყობინებათა (Messages) მართვას ინტერაქტიურობის დინამიკურ მოდელში, რომელსაც მიმდევრობითობის დიაგრამით (Sequence-D) ვიცნობთ. 30.60 ნახაზზე მოცემული გვაქვს ასეთი დიაგრამის ფრაგმენტი:



ნახ.30.60. იმიტაციური პროცესის ეკვივალენტური
მიმდევრობითობის დიაგრამა

30.7. მარკეტინგული პროცესების პროგრამული რეალიზაცია კლიენტსერვერული არქიტექტურით

ამჯერად უნდა გადავწყვიტოთ საკითხი, როგორ დავაპროგრამოთ მარკეტინგული მენეჯმენტის ის ბიზნესპროცესები, რომელთა მოდელი ავაგეთ პეტრის ქსელის CPN ინსტრუმენტით. კლიენტსერვერული ბიზნესპროცესების შესრულებისას ერთ-ერთი მნიშვნელოვანი საკითხია კომუნიკაცია „მიმწოდებელ-დამკვეთ” აპლიკაციებს შორის, ან კლიენტებსა და სერვერებს შორის. ასეთი კავშირების რეალიზაცია შესაძლებელია

ბიზნესპროცესებსა და ჰოსტდანართებს შორის. ქვემოთ განვიხილავთ ჩვენი მაგალითის შესაბამისი პროგრამების ფრაგმენტებს.

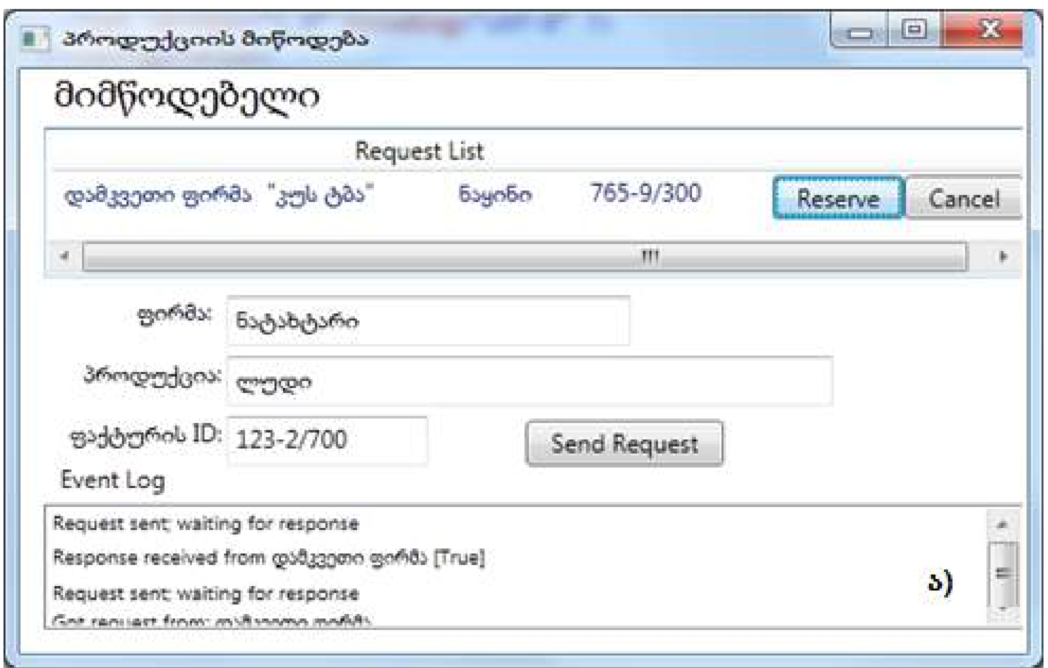
აპლიკაციის მაგალითის სახით ვიხილავთ პროექტის აგებას პროდუქციის მიმწოდებელ ფირმასა და დამკვეთ ორგანიზაციას შორის. კერძოდ, მიმწოდებელი (Firm) აგზავნის პროდუქციის პარტიას (Product), შესაბამისი თანმხლები დოკუმენტით, ფაქტურით (Invoice) დამკვეთთან. როდესაც დამკვეთი ფირმა მიიღებს პროდუქციას, იგი საპასუხო შეტყობინებას უბრუნებს მიმწოდებელს, რითაც ეს „ტრანზაქცია" დასრულებულად ითვლება. მიმწოდებლის იმავე აპლიკაციას შეუძლია მოთხოვნის გაგზავნა სხვა დამკვეთთან და ასევე საპასუხო შეტყობინების მიღება მათგან. თუ საპასუხო შეტყობინება არ დაბრუნდა მიმწოდებელთან, ეს ნიშნავს, რომ შეკვეთა არაა შესრულებული და შესაძლებელია პროდუქციის იგივე პარტია კვლავ გაიგზავნოს დამკვეთთან (საჯარიმო სანქციების პრევენციის მიზნით).

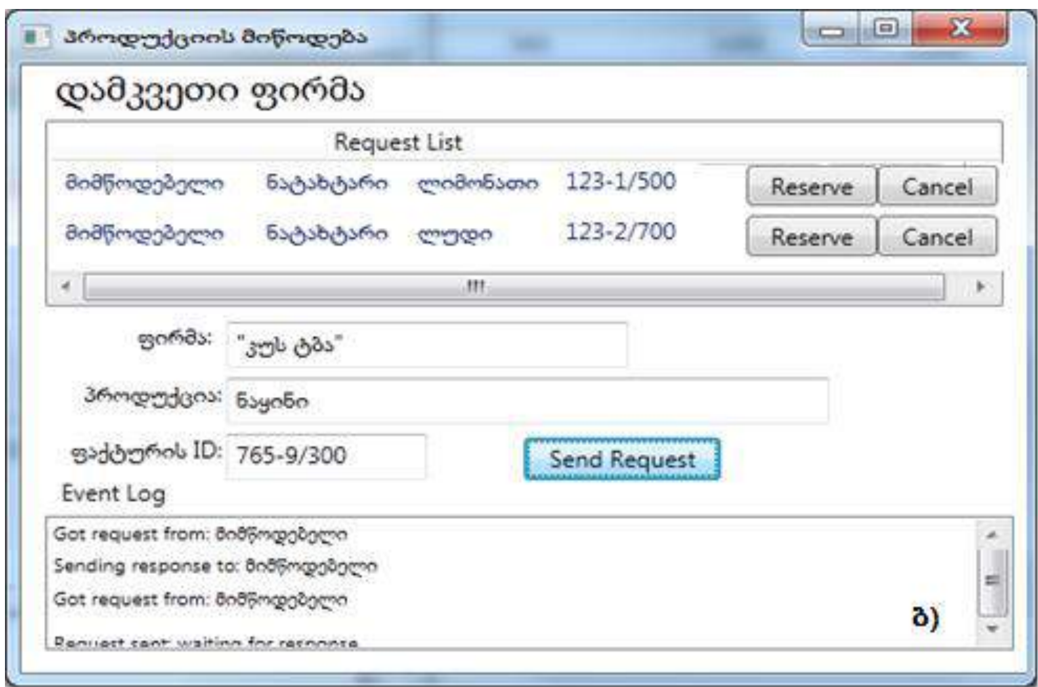
მთავარი ქმედებები, რომლებიც კომუნიკაციისათვის გამოიყენება არის Send და Receive ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply). ეს ქმედებები გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და სამეთვალყურეოდ [17,18]. ჩვენ ავაგებთ მარტივ WPF აპლიკაციას (Windows Presentation Foundation), რომელიც გამოიყენებს კომუნიკაციას, მაგალითად ორ სხვადასხვა აპლიკაციის (მიმწოდებელი და დამკვეთი) ბიზნესპროცესებს შორის.

30.61-ა,ბ ნახაზებზე მოცემულია საილუსტრაციო ფრაგმენტები „მიმწოდებელ-დამკვეთ" აპლიკაცებს შორის, თუ როგორი შეიძლება იყოს მათი ინტერფეისები.

მაგალითად, ფირმა „ნატახტარი" აგზავნის შეკვეთილი „ლიმონათის პარტიას", ფაქტურით „123-1/500", ამოქმედდა „Send Request" ღილაკი და დამკვეთი ფირმის ინტერფეისზე „Request List"-ში გამოჩნდა სტრიქონი ამის შესახებ. ეს ნიშნავს პროდუქციის ადგილზე მიტანას (ჩვენ მაგალითში ეს მყისიერად მოხდა, თუმცა შესაძლებელია გარკვეული დაყოვნების დროის გამოყენებაც, შემთხვევით რიცხვთა გენერატორის დახმარებით, რადგან პროცესი სტოქასტურია) [1]. გარკვეული დროის შემდეგ, მიმწოდებელი აგზავნის მეორე შეკვეთას, „ლუდის პარტიას", ფაქტურით „123-2/700" და ა.შ.

დამკვეთი ფირმა, პროდუქციის პარტიის მიღების შემთხვევაში, იყენებს „Reserve" ღილაკს, რაც უზრუნველყოფს მიმწოდებლის ინფორმირებას პროდუქციის ამ პარტიის მიღების შესახებ. ეს შეტყობინება მიმწოდებლის ინტერფეისზე აისახება მოვლენათა რეგისტრაციის, „Event Log" ლისტბოქსში. 30.1 ლისტინგში მოცემულია „მიმწოდებლის" ინტერფეისის პროგრამული აპლიკაციის კონფიგურაციის ფაილი (App.config). აქ ყურადსაღებია პორტის ნომრები (Address, Request Address), რომლებიც „დამკვეთი ფირმისთვის" იგივეა, ოღონდ შებრუნებული.





ნახ.30.61-ა,ბ. „მიმწოდებელი-დამკვეთი“ აპლიკაციის
ინტერფეისები

```
<-- ლისტინგი–30.1: --- App.config ---
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Branch Name" value="მიმწოდებელი"/>
    <add key="ID" value="{43E6DADD-4751-4056-8BB7-7459B5C361AB}"/>
    <add key="Address" value="8730"/>
    <add key="Request Address" value="8000"/>
  </appSettings>
</configuration>
```

ორივე პროგრამული აპლიკაცია, გაიშვება „Run as administrator" რეჟიმში. ერთ კომპიუტერზე (ექსპერიმენტისათვის) ერთდროულად ჩანს ორი ფანჯარა (ნახ.30.61-ა,ბ). ინფორმაციის მომზადება და გადაცემა, ასევე შეტყობინების გაგზავნა შესაძლებელია ორივე მიმართულებით. 30.2 და 30.3 ლისტინგებში ნაჩვენებია.

```
// -- ლისტინგი_30.2--- CreateRequest.cs   ---
using System;
using System.Activities;
using System.Configuration;
namespace ProductDelivery  // პროდუქციის მიწოდება
{
  public sealed class CreateRequest : CodeActivity
    {
       public InArgument<string> Product { get; set; }
       public InArgument<string> Firm { get; set; }
       public InArgument<string> InvoiceID { get; set; }
       public OutArgument<ReservationRequest> Request {get; set;}
       public OutArgument<string> RequestAddress {get; set;}

  protected override void Execute(CodeActivityContext context)
   {
     // config ფაილია გახსნა და Request Address-ის მიწოდება
    Configuration config = ConfigurationManager
       .OpenExeConfiguration(ConfigurationUserLevel.None);
     AppSettingsSection app =
            AppSettingsSection)config.GetSection("appSettings");
    // ReservationRequest კლასის შექმნა და მისი შევსება არგუმენტებით
```

```
      ReservationRequest r = new ReservationRequest
        (
          Product.Get(context),
          Firm.Get(context),
          InvoiceID.Get(context),
          new Branch
           {
             BranchName = app.Settings["Branch Name"].Value,
             BranchID = new Guid(app.Settings["ID"].Value),
             Address = app.Settings["Address"].Value
           },
          context.WorkflowInstanceId
         );

       // მოთხოვნის შენახვა OutArgument-ში
         Request.Set(context, r);

      // მისამართის შენახვა OutArgument-ში
       RequestAddress.Set(context, app.Settings["Request Address"].Value);
    }
  }
}
```

// -- ლისტინგი_30.3 --- CreateResponse. cs ---

```
using System;
using System.Activities;
using System.Configuration;
namespace ProductDelivery
 {
  public sealed class CreateResponse : CodeActivity
   {
    public InArgument<ReservationRequest> Request {get; set;}
    public InArgument<bool> Reserved { get; set; }
    public OutArgument<ReservationResponse> Response {get; set;
   }

protected override void Execute(CodeActivityContext context)
```

```
 {
   // config ფაილის გახსნა ---
   Configuration config = ConfigurationManager
             .OpenExeConfiguration(ConfigurationUserLevel.None);
   AppSettingsSection app =
             (AppSettingsSection)config.GetSection("appSettings");

   // ReservationResponse კლასის შექმნა და მისი შევსება ----
      ReservationResponse r = new ReservationResponse
      (
         Request.Get(context),
         Reserved.Get(context),
         new Branch
          {
            BranchName = app.Settings["Branch Name"].Value,
            BranchID = new Guid(app.Settings["ID"].Value),
            Address = app.Settings["Address"].Value
          }
         );
    // პასუხის შენახვა OutArgument-ში
       Response.Set(context, r);
   }
  }
}
```

30.4 ლისტინგში მოცემულია კლიენტის სერვისის კლასის კოდის ფრაგმენტი.

```
// -- ლისტინგი_30.4 --- ClientService.cs   ---
using System;
using System.ServiceModel;

namespace ProductDelivery
{
  public class ClientService : IProductDelivery
   {
    public void RequestProduct(DeliveryRequest request)
      {
         ApplicationInterface.RequestProduct(request);
      }
```

```
  public void RespondToRequest(DeliveryResponse response)
      {
         ApplicationInterface.RespondToRequest(response);
      }
   }
 }
```

30.8. დასკვნა

გამოკვლეულია პროდუქციის მწარმოებელი ორგანიზაციის (ფირმის) მარკეტინგის ფუნქციები, ამ სფეროში კომპიუტერული ტექნიკისა და ახალი ინფორმაციული ტექნოლოგიების დანერგვის აქტუალურობა. დასაბუთდა იმიტაციური მოდელირების გამოყენების ეფექტურობა. აუცილებელი გახდა კომპიუტერული ტექნოლოგიების გამოყენებით მარკეტინგის მართვის პროცესების მოდელირება და შესაბამისი პროგრამული პაკეტების მოძიება და შემუშავება;

იმიტაციური მოდელირების დახმარებით პეტრის ქსელების ინსტრუმენტით შესაძლებელია რთული, დინამიკური პროცესების ანალიზი, სტატისტიკური მონაცემების დამუშავება, მთლიანად ქსელის ან ცალკეული ქვექსელების უსაფრთხოების კვლევა;

ინფორმაციული სისტემების დაპროექტებისა და მათი პროგრამული რეალიზების თანამედროვე ტექნოლოგიების გამოყენება ობიექტორიენტირებული, პროცესორიენ-ტირებული და სერვისორიენტირებული მიდგომების საფუძველზე მნიშვნელოვნად აუმჯობესებს განაწილებული მართვის საინფორმაციო სისტემების დაპროექტებისა და აგების პროცესს;

დაპროექტებული მართვის საინფორმაციო სისტემის ეფექტური პროგრამული რეალიზაცია მომხმარებელთა ინტერფეისების ჩათვლით, უნდა განხორციელდეს დაპროგრამების ჰიბრიდული ახალი ტექნოლოგიებით .NET პლატფორმაზე, კერძოდ, WPF, WF და WCF პაკეტებით;

მონაცემთა საცავების, ან ბაზების შერჩევა და გამოყენება დაპროექტებული მართვის საინფორმაციო სისტემისთვის უნდა განხორციელდეს თანამედროვე რელაციური, NoSQL ან ჰიბრიდული ტიპის (NewSQL) მონაცემთა ბაზების მართვის სისტემების ანალიზის საფუძველზე.

ლიტერატურა:

1. ჩოგოვაძე გ., ფრანგიშვილი ა., გოგიჩაიშვილი გ., დიდმანიძე ვ., სურგულაძე გ. (2016). მართვის ავტომატიზებული სისტემები და პროგრამული ინჟინერია: ინოვაციები საუნივერსიტეტო განათლების სფეროში. სტუ შრ.კრებ. „მართვის ავტომატიზებული სისტემები" N1(21). თბ., გვ. 9-24.

2. ჩოგოვაძე გ. (2003). ინფორნაცია (ინფორმაცია, საზოგადოება, ადამიანი). საქართველო, თბ., „ნეოსტუდია".

3. ფრანგიშვილი ა., სურგულაძე გ., ვაჭარაძე ი. (2009). ბიზნეს-პროგრამების ექსპერტულ შეფასებებში გადაწყვეტილებათა მიღების მხარდამჭერი მეთოდები და მოდელები. სტუ. მონოგრ., თბ., „ტექნიკური უნივერსიტეტი".

4. სურგულაძე გ., ბულია ი. (2012). კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. მონოგრ., ISBN 978-9941-20-165-3. სტუ. თბ., „ტექნიკური უნივერ-სიტეტი".

5. სურგულაძე გ., ურუშაძე ბ. (2014). საინფორმაციო სისტემების მენეჯმენტის საერთაშორისო გამოცდილება (BSI, ITIL, COBIT). დამხმ.სახელმძღვ., ISBN 978-9941-20-458-6. სტუ. თბ., „ტექნიკური უნივერსიტეტი".

6. BSI-Standard 100-1: Managementsysteme für Informations-sicherheit (ISMS). Bundesamt für Sicherheit in der Informationstechnik, (BSI) Godesberger Allee 185-189, 53175 Bonn, 2008/2013.

7. BSI-Standard 100-2: IT-Grundschutz-Vorgehensweise. (BSI) Godesberger Allee 185-189, 53175 Bonn, 2008/2013.

8. BSI-Standard 100-3: Risikoanalyse auf der Basis von IT-Grundschutz. Bundesamt für Sicherheit in der Informationstechnik, Bonn. 2011.

9. BSI-Standard 100-4: Notfallmanagement Version 1.0 Inhaltverzeichnis. Bundesamt für Sicherheit in der Informationstechnik, (BSI) Godesberger Allee 185-189, 53175 Bonn, 2008/2013.

10. Мак-Дональд М. (2008). WPF: Windows Presentation Fundation в .NET 3.5 с примерами на C# 2008 для профессионалов. 2-е изд.: Пер. с англ. - М. : ООО "И.Д. Вильямс".

11. Petzold Ch. (2008). Applications=Code+Markup. A Guide to the MicroSoft Windows Presentation Foundation. St-Petersburg.

12. Уотсон К.,Нейгел К., Педерсен Я., ХаммерР., Джон Д., Скиннер М., Уайт Э. (2009). Visual C# 2008: базовый курс. : Пер. с англ. - М. : ООО "И.Д. Вильямс".

13. Долженко А.И. (2011). Разработка приложений на базе WPF и Silverlight. –М. www.intuit.ru/department/se/dawpfsl/

14. Снетков В.М. (2008). Практикум прикладного программирования на C# в среде VS.NET 2008. www.intuit.ru/department/se/prcsharp08/

15. Eberhardt C. (2009). WPF DataGrid Practical Examples. http://www.codeproject.com-/Articles/30905/WPF-DataGrid-Practical-Examples.

16. სურგულაძე გ. (2014).. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: WPF ტექნოლოგია (ნაწ.1). სტუ, თბ. „IT კონსალტინგის ცენტრი".

17. სურგულაძე გ. (2015). კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: Workflow ტექნოლოგია (ნაწ.2). სტუ, თბ. „IT კონსალტინგის ცენტრი".

18. სურგულაძე გ. (2016.). კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: WCF ტექნოლოგია (ნაწ.3). სტუ, თბ., „IT კონსალტინგის ცენტრი".

19. ITIL moving towards Enterprise Architecture. http://blogs.msdn.com/b/mikewalker-/archive/2007/07/06/itil-moving-towards-enterprise-architecture.aspx?Redirected=true

20. ITILv3. Глоссарий терминов и определений, ITIL® V3 Glossary Russian Translation. v0.92, 30 Apr 2009.

21. Скрипник Д. (2012). ITIL : IT Service Management по стандартам V.3.1. http://www.intuit.ru/studies/courses/2323/623/info. გად.10.01.14

22. COBIT Overview ISACA. http://www.isaca.org/knowledge-center/cobit/Pages/Overview.aspx. გადამოწ.14.01.14

23. COBIT 5 for Information Security. http://www.isaca.org/COBIT/Documents/COBIT-5-for-Information-Security-Introduction.pdf.

24. Скрипник Д. (2012). Управление ИТ на основе COBIT 4.1. М., www.intuit.ru-/studies/courses/3704/946/info.

25. Booch G., Jacobson I., rambaugh J. (1996). Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara,

26. The Unified Modeling Language. http://www.uml-diagrams.org/ უკანასკნ. გადამოწმ. 10.05.14

27. UML: Basics Principles and Background. http://sourcemaking.com/uml

28. Principles behind the Agile Manifesto. http://agilemanifesto.org/princip-les.html.

29. Бек К. (2008). Шаблоны реализации корпоративных приложений. Экстремальное программирование: Пер. с англ. М.: Вильямс.

30. Амблер С. (2005). Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. Библиотека программиста. СПб.: Питер.

31. Rumpe B. (2012). Agile Modellierung mit UML. Berlin, „Springer". 2-te Auflage.

32. სურგულაძე გ., გულიტაშვილი მ., კაკულია ი., ჩერქეზიშვილი გ., ჯავახიშვილი ი. (2010). პროგრამული სისტემების სასიცოცხლო ციკლის პროცესის მოდელირება უნივერსალური და ექსტრემალური პროგრამირების პრინციპების კომპრომისული გადაწყვეტით. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“, N 1(8). გვ.63-70.

33. გოგიჩაიშვილი გ., სუხიაშვილი თ. (2012). სისტემების ობიექტორიენტირებული ანალიზი და დაპროექტება. სტუ, თბ. „ტექნიკური უნივერსიტეტი“.

34. სურგულაძე გ., პეტრიაშვილი ლ. (2007). მონაცემთა საცავის აგების ტექნოლოგია ინტერნეტული ბიზნესის სისტემებისათვის. მონოგრ., ISBN 99940-36-7-7. სტუ.. თბ., „ტექნიკური უნივერსიტეტი“.

35. სურგულაძე გ. (2016). დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. სტუ. „IT-კონსალტინგის ცენტრი“, თბ., -272 გვ.

36. ჩოგოვაძე გ., სურგულაძე გ., შონია ო. (1996). მონაცემთა და ცოდნის ბაზების აგების საფუძვლები. სახელმძღვანელო. თსუ, „განათლება”, თბ., „თსუ“.

37. ჩოგოვაძე გ., გოგიჩაიშვილი გ., სურგულაძე გ., შეროზია თ., შონია ო. (2001). მართვის ავტომა¬ტიზებული სისტემების დაპროექტება და აგება (თეორიული და პრაქტიკული ინფორმატიკა). სახელმძღ., სტუ, თბ., -750 გვ.

38. გოგიჩაიშვილი გ., ფრანგიშვილი ა., სურგულაძე გ. (2007). ინფორმატიკა, პროგრამული ტექნოლოგიები და მათი განვითარების და სწავლების თანამედროვე მიმართულებანი. სტუ-ს შრ.კრ., „მას”, N 1(2), თბ., გვ.7-15

39. სურგულაძე გ., გულიტაშვილი მ., კივილაძე ნ. (2015). Web-სისტემების ტესტირება, ვალიდაცია და ვერიფიკაცია. მონოგრ. სტუ. „IT-კონსალტინგის ცენტრი“. თბ.

40. მეიერ-ვეგენერი კ., სურგულაძე გ., ბასილაძე გ. (2014). საინფორმაციო სისტემების აგება მულტიმედიურ მონაცემთა ბაზებით. ISBN 978-9941-20-468-5. სტუ, თბ., „ტექნიკური უნივერსიტეტი“.

41. გოგიჩაიშვილი გ., ბოლხი გ., სურგულაძე გ., პეტრიაშვილი ლ. (2013). მართვის ავტომატიზებული სისტემების ობიექტ-ორიენტირებული დაპროექტების და მოდელირების ინსტრუმენტები (MsVisio, WinPepsy, PetNet, CPN). სახელმძღვ., ISBN 99940-56-77-8. სტუ.. თბ., „ტექნიკური უნივერსიტეტი“.

42. სურგულაძე გ., გულიტაშვილი მ., ჩერქეზიშვილი. (2011). Web აპლიკაციების დამუშავების პროცესის მოდელირება UML/2 ტექნოლოგიით. Intern. Science Conf.“Automated Control Systems & new IT”, 20-22 Mai. GTU, Tbilisi, გვ. 180-184

43. სურგულაძე გ., ქრისტესიაშვილი ხ., სურგულაძე გ. (2015). საწარმოო რესურსების მენეჯმენტის ბიზნესპროცესების მოდელირება და კვლევა. მონოგრ., ISBN 978-9941-20-557-6. სტუ. თბ., „ტექნიკური უნივერსიტეტი“.

44. სურგულაძე გ., ფხაკაძე ც., კეკენაძე ა. (2016). ორგანიზაციული მართვის ბიზნესპროცესების მოდელირება და დაპროექტება. მონოგრ., ISBN 978-9941-0-8259-7. სტუ, თბ., „IT კონსალტინგის ცენტრი“.

45. სურგულაძე გ. თურქია ე. (2003). ბიზნესპროცესების მართვის ავტომატი-ზებული სისტემების დაპროექტება. მონოგრ., ISBN 99940-14-81-1. სტუ, თბილისი. „ტექნიკური უნივერსიტეტი“.

46. თურქია ე. (2010). ბიზნესპროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. სტუ. თბ., „ტექნიკური უნივერსიტეტი“.

47. Кознов Д.В. (2009). Введение в программную инженерию. http://www. intuit.ru/department/se/inprogeng.

48. Scrum.org. https://www.scrum.org/Resources

49. Information Systems Examinations Board (ISEB). ttp://www.bcs.org/.

50. Foundation Certificate in IT Service Management. https:// www.exin.com/NL/en-/exams/?exam=itil-v3-foundation.

51. სურგულაძე გ., თურქია ე., ქაჩლიშვილი თ., ფხაკაძე ც. (2014). საფინანსო კორპორაციის ბიზნეს-პროცესების მენეჯმენტი ITIL მეთოდოლოგიის საფუძველზე (რეპორტების ავტომატიზაცია). სტუ-ს შრ.კრ. „მას“. 2, 18, თბ., გვ.51-56.

52. Carnegie Mellon Software Engineering Institute. http://www.sei.cmu.edu/

53. Software quality assurance. https://en.wikipedia.org/wiki/Software_quality_assurance

54. Sommerville I. (2010). Softwear engineering, 9th ed., ISBN 978-0137035151. Addison-Wesley.

55. სურგულაძე გ., თურქია ე. (2016). პროგრამული სისტემების მენეჯმენტის საფუძვლები. სახელმძღვ., სტუ, თბ., -350 გვ. http://www.gtu.ge/katedrebi/kat94/pdf/sql.pdf

56. ბიტარაშვილი მ., სურგულაძე გ. (2012). საგადასახადო სამართალდარღვევის საქმის წარმოების სისტემის ბიზნესპროცესების მოდელირება UML2 ტექნოლოგიით. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N2(13), თბ., გვ. 217–222.

57. სურგულაძე გ., ოხანაშვილი მ., კაშიბაძე მ., ნეფარიძე მ. (2014). თანამედროვე საინფორმაციო ტექნოლოგიები მარკეტინგული პროცესების და წარმოების მენეჯმენტში. სტუ-ს შრ.კრ. „მართვის ავტომატ.სისტ.“. N1(17), თბ., გვ. 64-71.

58. საქართველოს საგადასახადო კოდექსი. თბ., 17 სექტ., 2010.

59. სურგულაძე გ., ბიტარაშვილი მ. (2013). ბიზნესპროცესების UML-მოდელირება და პროგრამული რეალიზაცია Workflow Foundation ტექნოლოგიით საგადასახდო დავების სისტემის მაგალითზე. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(14), თბ., გვ. 229-233.

60. სურგულაძე გ., ოხანაშვილი მ., სურგულაძე გ. (2009). მარკეტინგის ბიზნეს_პროცესების უნიფიცირებული და იმიტაციური მოდელირება. მონოგრ., სტუ. თბ., „ტექნიკური უნივერსიტეტი“.

61. Jensen K., Kristensen M.L., Wells L. (2007). Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. University of Aarhus. Denmark.

62. CPN Tools. www.daimi.au.dk/CPNTools/

63. Collins M.J. Beginning WF: Windows Workflow in .NET 4.0. ISBN-13 (pbk): 978-1-4302-2485-3 Copyright © 2010. USA. http://www.ebooks-it.net/ebook/beginning-wf.

64. Уотсон К.,Нейгел К., Педерсен Я., ХаммерР., Джон Д., Скиннер М., Уайт Э.(2009). Visual C# 2008: базовый курс. : Пер. с англ. - М. : ООО "И.Д. Вильямс".

65. ჩოგოვაძე გ. (2015). ფიქრები მომავალზე. თბ., -198 გვ.

66. ფრანგიშვილი ა., სამხარაძე რ. (2002). ენერგოსისტემების მართვის ექსპერტული სისტემების აგების თეორია. მონოგრ., თბ., „მეცნიერება˜.

67. Прангишвили А., Прокопьев С. (2005). Информационные технологии согласования управленческих решений по выбору целей и стратегий в конфликтологии. Georgian Electronic Scientific Journal. #3(7). http://gesj.internet-academy.org.ge.

68. ჩოგოვაძე გ. (2006). გლობალანსი. მოსკოვი. „ზ. წერეთელის შემოქმედებითი სახელოსნოები”.

69. ჩოგოვაძე გ. (2009). ბიოსფეერია. მოსკოვი. „ზ. წერეთელის შემოქმედებითი სახელოსნოები”.

70. სურგულაძე გ., თურქია ე., თოფურია ნ. (2015). ჰიბრიდული აპლიკაციების დაპროგრამების ავტომატიზაცია. საერთაშ. სამეცნიერო კონფერენცია „საინფორმაციო და კომპიუტერ. ტექნოლოგიები, მოდელირება და მართვა“ (მიძღვ. ივ.ფრანგიშვილი-85 წლისთავისადმი). სტუ, თბ., გვ. 69-73 66.

71. სურგულაძე გ., ოხანაშვილი მ., კაშიბაძე მ., ნეფარიძე მ. (2014). თანამედროვე საინფორმაციო ტექნოლოგიები მარკეტინგული პროცესების და წარმოების მენეჯმენტში. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(17), თბ., გვ. 64-71.

72. სამხარაძე რ., გაჩეჩილაძე ლ. (2016). SQL სერვერი. სტუ. თბ., „ტექნიკური უნივერსიტეტი“.

73. Петкович Д. (2013). Microsoft SQL Server 2012. Руководство для начинающих: Пер. с англ. - СПб.: БХВ-Петербург.

74. Create Spatial Index (Transact-SQL). https://msdn.microsoft.com/en-us/library/bb934196.aspx

75. Laudon K.C., Laudon J.P. (2014). Management Information Systems Managing the Digital Firm. 13th Edit., Global Edition. New York University. Azimuth Information Systems. published by Pearson Education. England.

76. Halpin T. (2005). ORM 2 Graphical Notation, Neumont University. http://www.orm.net/ pdf/ORM2_TechReport1.pdf.

77. სურგულაძე გ., ვედეკინდი ჰ., თოფურია ნ. (2006). განაწილებული ოფის-სისტემების მონაცემთა ბაზების დაპროექტება და რეალიზაცია Value at Risk ტექნოლოგიით. სტუ, თბ., „ტექნიკური უნივერსიტეტი".

78. სურგულაძე გ., თოფურია ნ., ბაკურია კ., ლომიძე მ. (2014). საინფორმაციო სისტემის დაპროექტება ობიექტროლური მოდელირებისა და სერვისორიენტირებული არქიტექტურის ბაზაზე. სტუ-ს შრ.კრ., „მას".N1(17). თბ., გვ. 32-45.

79. Barker R. (1990). CASE Method: Entity Relationship Modelling. Reading, MA: Addison-Wesley Professional.

80. სურგულაძე გ. (2011). ვიზუალური დაპროგრამება C#_2010 ენის ბაზაზე. სტუ, თბ., „ტექნიკური უნივერსიტეტი".

81. Surguladze G., Topuria N., Petriashvili L., Surguladze Giorgi. (2015). Modelling of Designing a Conceptual Schema for Multimodal Freight Transportation Information System. WASET, World Academy of Scientific, Engineering and Technology, v.9, N11. ISSN 1307-6892, Spain, pp. 204-207.

82. გოგიჩაიშვილი გ., სურგულაძე გიორგი. (2014). მულტიმოდალური გადაზიდ-ვების ბიზნესპროცესების ავტომატიზებული მართვის კონცეფცია. სტუ-ს შრ.კრ. „მას" N2(18). გვ.45-50.

83. სურგულაძე გიორგი. (2015). მულტიმოდალური გადაზიდვების ბიზნეს-პროცესების მართვის სისტემის ინფრასტრუქტურა და მისი იმიტაციური მოდელი. სტუ-ს შრ.კრ.„მას", 2(20). თბ., გვ.108-123.

84. სურგულაძე გ., თოფურია ნ., ბასილაძე გ., კივილაძე ნ., ნეფარიძე მ. (2014). ელექტრონული საარჩევნო სისტემა მულტიმედიური მონაცემთა ბაზებით და კლიენტ-სერვერული არქიტექტურით. GESJ: Computer Science and Telecommunications. N2(42). გვ.39-86.

85. სურგულაძე გ., თოფურია ნ., ბასილაძე გ., ურუშაძე ბ., ლომიძე მ., გაბინაშვილი ლ. (2013). პროგრამული სისტემების მენეჯმენტი მულტიმედიური აპლიკაციების დასაპროექტებლად და ასაგებად. VI საერთ. სამეცნ.პრაქტ. კონფ. „ინტერნეტი და საზოგადოება". აკ.წერეთლის სახ.უნივ. ქუთაისი, გვ. 66-70.

86. სურგულაძე გ., თოფურია ნ. (2007). მონაცემთა ბაზების მართვის სისტემები: ობიექტროლური მოდელირება (ORM/ERM, SQL Server). სახელმძღ., სტუ, თბ., „ტექნიკური უნივერსიტეტი".

87. ბასილაძე გ., სურგულაძე გ., გაბინაშვილი ლ. (2013). მულტიმედიური ელექტრონული საარჩევნო სისტემის პროგრამული უზრუნველყოფის დამუშავება. სტუ-ს შრ.კრ. „მას". N1(14), თბ., გვ. 234-239.

88. SQL Server Security. https://msdn.microsoft.com/en-us/library/bb669074(v=vs.110).-aspx

89. Clarke R. Electronic Data Interchange (EDI): An Introduction. http://www.roger clarke.com/EC/EDIIntro.html

90. Sokol K.P. (1995). From Edi to Electronic Commerce: A Business Initiative. McGraw-Hill.

91. სურგულაძე გ., თოფურია ნ., გავარდაშვილი ა. (2016). მონაცემთა ბაზის დაპროექტების ავტომატიზაცია შავი ზღვის ეკოლოგიური სისტემისათვის. სტუ-ს შრ. კრ. „მართვის ავტომატიზებული სისტემები". No 1(21), თბ., გვ. 165-168.

92. სურგულაძე გ., კივილაძე გ. (2016). ტრანზაქციის იზოლირების დონეები რელაციურ და არარელაციურ მონაცემთა ბაზებში. მე-3 საერთ.კონფ. „კომპიუტინგი-/ინფორმატიკა, განათლების მეცნიერებები". თბ., გვ.161-168.

93. სურგულაძე გ., პეტრიაშვილი ლ., თოფურია ნ. (2016). მულტიმოდალური გადაზიდვების სერვის-ორიენტირებული სისტემის აგება CASE და SHAREPOINT ტექნოლოგიებით. მე-3 საერთ.კონფ. „კომპიუტინგი/ინფორმატიკა, განათლების მეცნიერებები". თბ., გვ.155-160.

94. Surguladze G., Gavardashvili A., Topuria N. (2016). Determination of the Ecological Parameters of the Black Sea and Designing its Multimedia Base based on the Object-Role Modeling. XXVII Intern.Scientific Conf., `Problems of Decision Making under Uncertaintie". Kiev, pp.65-68.

95. სურგულაძე გ., კივილაძე ნ., კივილაძე გ. (2016). კორპორაციული აპლიკაციების აგება დაპროგრამების სერვისორიენტირებული ტექნოლოგიით. სტუ-ს შრ. კრ. „მართვის ავტომატიზებული სისტემები". No 1(21), თბ., გვ. 230-235.

96. სურგულაძე გ., ოდიშარია კ., ფხაკაძე ც., კეკენაძე ა., ჩერქეზიშვილი გ. (2016). საფინანსო ორგანიზაციის ბიზნესპროცესებისა და IT-სამსახურის ინფორმაციული უსაფრთხოების რისკების შეფასება. სტუ-ს შრ. კრ. „მართვის ავტომატიზებული სისტემები". No 1(21), თბ., გვ. 230-235.

97. გოგიჩაიშვილი გ., სურგულაძე გ., თოფურია ნ., პეტრიაშვილი ლ. (2015). მულტიმოდალური გადაზიდვების მართვის ავტომატიზებული სისტემის აგება დაპროექტების CASE- და დაპროგრამების ჰიბრიდული ტექნოლოგიებით. სტუ-ს შრ. კრ. „მართვის ავტომატიზ. სისტემები“. No2(20), თბ., გვ. 96--107.

98. სურგულაძე გ., კაიშაური თ., ნარეშელაშვილი გ., მაისურაძე გ. (2015). პროგრამული აპლიკაციის დამუშავების სასიცოცხლო ციკლი VisualStudio.NET Framework-ის ახალ ვერსიებში ჰიბრიდული აპლიკაციების დაპროგრამების ავტომატიზაცია. სტუ-ს შრ. კრ. „მართვის ავტომატ. სისტემები“. No2(20), თბ., გვ. 212-222.

99. სურგულაძე გ., თურქია ე., თოფურია ნ. (2015). ჰიბრიდული აპლიკაციების დაპროგრამების ავტომატიზაცია. საერთაშ.სამეცნ. კონფ., „საინფორმაციო და კომპიუტერ. ტექნოლოგიები, მოდელირება და მართვა“. ივ.ფრანგიშვილის 85-წლ. სტუ, თბ., გვ. 69-73.

100. Floyd R.W. (1979). Paradigms of Programming. Communications of the ACM. #8, vol.22, August ‘79. Stanford University. USA.

101. Bobrow D.G. (1984). If Prolog is the answer, what is the question. 5-th Generation of Computer Systems, pp. 138-145, Tokyo, Japan, November ‘84. Inst. for New Generation Computer Technology (ICOT). North-Holland.

102. Shriver B.D. (1986). Software paradigms. IEEE Software, 3(1):2, January ‘86.

103. Wegner P. (1990). Concepts and paradigms of object-oriented programming. {OOPS messenger}, 1(1): 7-87, August ‘90.

104. Budd T.A. (1995). Multy-Paradigm Programming in LEDA. Addison-Wesley, Reading, Massachusets.

105. გოგიჩაიშვილი გ., სურგულაძე გ., შონია ო. (1997). დაპროგრამების მეთოდები C & C++. სახელმძღვ.. სტუ. თბ., „ტექნიკური უნივერსიტეტი“.

106. Troelsen A., Japikse P. (2015) C# 6.0 and the .NET 4.6 Framework. 7-th Edition. Copyright by Andrew Troelsen and Philip Japikse. Springer Science+Business Media NewYork.

107. Perkins B., Hammer J.V., Reid J.D. (2016). Beginning Visual C# 2015 Programming. Published by John Wiley & Sons, Inc. Indianapolis, IN 46256 . Copyright © 2016.

108. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) W3C Recommendation 27 April 2007. http://www.w3.org/TR/soap12-part1/#intro

109. Fielding R.T. (2000). Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine. http://www.ics.uci.edu/~fielding/pubs/-dissertation/top.htm

110. Скопин И. (2004). Основы менеджмента программных проектов. Новосибирский Гос.Унив., Россия '04. INTUIT. http://www.intuit.ru/studies/courses/38/38/info

111. Surguladze G., Petriaschvili L., Surguladze G. (2015). Decision Support System for Optimization of Seaport Resources with considering multimodal transportation. III internat. Scientific Conf., "Computing / Informatics, Education Sciences, Teacher Education. Batumi. pp.139-143.

112. Meyer-Wegener K., Surguladze G., Basiladze G. (2015). Construction of Information Systems with Multimedia Databases. III internat. Scientific Conf., "Computing / Informatics, Education Sciences, Teacher Education. Batumi. pp.43.

113. Distributed Management Task Force. http://www.dmtf.org/

114. Surguladze G., Turkia E., Topuria N., Basiladze G. (2012). Automation of Business-Processes of an Election System. VI Intern. Conf. (AICT 2012). Application of Information and Communic. Technologies. ISBN 978-1-4673-1740-5. Tb., Georgia '12. pp. 308-312.

115. სურგულაძე გ., ბოტჰე კ., კაშიბაძე მ. (2001). მემკვიდრეობითობა მართვის ინფორმაციული სისტემების დაპროგრამებაში: მონაცემთა ბაზებიდან UML-ტექნოლოგიამდე. საერთ.კონფ. შრ.კრებ. N4(437). სტუ. თბ., გვ. 55-62.

116. სურგულაძე გიორგი. (2016). მულტიმოდალური გადაზიდვების ბიზნეს-პროცესების კლასებისა და მდგომარეობათა დიაგრამების დაპროექტება. სტუ. შრ.კრ. „მას" 2(22). თბ., გვ.101-122.

117. Fowler A. (2015). NoSQL For Dummies®. Published by: John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com Copyright © 2015, New Jersey.

118. სურგულაძე გ., კივილაძე გ. (2017). შესავალი NoSQL მონაცემთა ბაზებში (MongoDB). ISBN 978-9941-0-9642-6. სტუ. თბ., „ITკონსალტინგის ცენტრი".

119. Morgan A., Lord M. (2014). NoSQL with MySQL. http://www.drdobbs.com/-database/nosql-with-mysql/240167115.

120. Document-oriented Database. Clusterpoint. Retrieved on 2015-10-08. https://www.-clusterpoint.com/

121. Гранков М.В., Жуков А.И. (2013). Системы управления Базами данных. Донской гос.техн. Университет. Ростов-на-Дону.

122. https://en.wikipedia.org/wiki/Graph_database

123. Stonebraker M. (2011). New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps. June 2011. http://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext

124. Choudhary S. (2014) NewSQL: Best of Both "OldSQL" and "NoSQL". http://www.-slideshare.net/sushantbchoudhary/newsql-the-best.

125. Glushkov I. (2015.). NewSQL overview. http://www.slideshare.net/IvanGlushkov-/newsql-overview.

126. MariaDB. https://en.wikipedia.org/wiki/MariaDB.

127. Cobbaut P. (2015). Linux Fundamentals. http://linux-trai¬ning¬.be-/linuxfun.pdf

128. სურგულაძე გ., კივილაძე გ., კახელი ბ. (2016). NoSQL მონაცემთა ბაზების განვითარების პერსპექტივები და პრობლემები მართვის საინფორმაციო სისტემებში. სტუ, შრ.კრ. „მას" 2(22), თბ., გვ.230-239.

129. MongoDB manual – http://docs.mongodb.org/manual/

130. The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. http://bit.ly/digital_universe.

131. Groenfeldt T. (2013). At NYSE, The Data Deluge Overwhelms Traditional Databases. http://www.forbes.com/sites/tomgroenfeldt/2013-/02/14/at-nyse-the-data-deluge-overwhelms-traditio¬nal-data¬bases/#64¬4-feb072eb7

132. Miller R. Facebook Builds Exabyte Data Centers for Cold Storage. http://bit.-ly/facebook_exabyte

133. Ancestry.com. Comp.Facts. Ancestry.com/corporate/aboutancestry/company-facts

134. https://www.mongodb.com/compare/mongodb-mysql

135. CAP theorem. https://en.wikipedia.org/wiki/CAP_theorem.

136. Grigorik I. (2010). Weak Consistency and CAP Implications. Igvita. Google-Co. 7 https://www.igvita.com/2010/06/24/weak-consistency-and-cap-implications/

137. Greiner R. (2014). CAP Theorem: Explained. http://robertgreiner.com/2014/06/cap-theorem-explained/

138. Fowler A. (2016). The State of NoSQL 2016: A quick guide to the NoSQL landscape. Kindle Edition, 52 p., Published February 5th 2016 by Adam Fowler.

139. Sharding. MongoDB Manual. (2017). Vers.3.4. https://docs.mongodb.com/manual-/sharding/#sharded-cluster

140. Parvesh Tandon, RestFul API using node.js and mongodb. https://github.com/mis-tertandon/node-express-hbs

141. AlternativeTo – Crowdsourced software recommendations. http://alternati-veto.net/software/robomongo/

142. კოტლერი ფ. მარკეტინგის საფუძვლები. თარგ.ინგ. თბ., 1993.

143. White S.A. Introduction to BPMN. IBM Corporation. United States. Oct. 2006. http://www.omg.org/bpmn/Documents/OMG _BPMN_Tutorial.pdf

144. Owner M. and Raj J. BPMN and Business Process Management. 2003.

145. MRP. http://en.wikipedia.org/wiki/Manufacturing_resource_planning.

146. Mallory J. (2016). ERP History. e2b-teknologies. http://e2btek.com/ibm-general-ledger-financial-control-1956/

147. Enterprise Resource Planning. MorganFranklin Consulting: strategy and execution-focused business consulting firm. http://www.morganfranklin.com/commercial-solutions/-information-management-and-technology/enterprise-resource-planning

148. Ansoff I. (1957). How to Use an Ansoff Matrix. https://www.smartdraw.com/ansoff-matrix/

149. Ansoff, I.: Strategies for Diversification, Harvard Business Review, Vol. 35 Issue 5,Sep-Oct 1957, pp. 113-124

150. https://www.getgsi.com/management-team

151. სურგულაძე გ., ბულია ი., თურქია ე. ვებ-აპლიკაციების დამუშავება მონაცემთა ბაზების საფუძველზე (ADO.NET, ASP.NET, C#). ISBN 978-9941-14-289-5. სტუ, თბ., 2009. 172 გვ.

152. Spatial Indexes Overview. https://msdn.microsoft.com/en-us/en-ue/library/bb895-265.aspx#decompose.

153. სურგულაძე გ., თოფურია ნ., გავარდაშვილი ა. (2016). ვებ-სერვისის რეალიზაცია შავი ზღვის მდინარეთა ესტუარების მონიტორინგის სისტემისათვის. სტუ-ს შრ. კრ. „მართვის ავტომატიზებული სისტემები“. No 2(22), თბ., გვ. 190-193.

154. სუპატაშვილი გ., ქაჯაია გ. (2001). გარემო და ადამიანი. თსუ გამომცემლობა, თბილისი.

155. Gavardashvili A.G. (2013). The Program Software to Create United Database of Black Sea Ecological Characteristics. Collected Papers of Water Management Iinstitute of Georgian Technical University, # 68, Tbilisi, pp. 27-32 .

156. Gavardashvili G.V., Chakhaya G.G., Diakonidze R.V., Tsulukidze L.N., Supatashvili T.L. (2011). The Results and Analysis of Studies Carried out in 2011 in the Black Sea Water Area within the Boundaries of Georgia. 3 rd Bi-annual BS Scientific Conference and UP-GRADE BS-SCENE Project Joint Conference. Odessa, Ukraine, 1-4 Nov., pp. 205.

დანართი_1

Web-ის უსაფრთხოების და სერვისის მომხმარებელთა ინტერფეისების რეალიზაცია

- **Web-ის უსაფრთხოება ASP.NET-ში. სერვერის, კლიენტების, ფორმების და როლების აუთენტიფიკაცია**

ვებ აპლიკაციებში ხშირად საჭიროა მომხმარებელთა იდენტიფიკაცია და მათთვის სხვადასხვა რესურსებზე წვდომის უფლების განსაზღვრა. ASP.NET-ში არსებობს აუთენტიფიკაციის რამდენიმე მეთოდი: Windows, Forms, Passport. ეს მოთოდები განისაზღვრება კონფიგურაციის ფაილში authentication ტეგის mode ატრიბუტის საშუალებით [4, 151]:

```
<configuration>
        <system.web>
                <authentication />
        </system.web>
</configuration>
```

მეთოდები:

```
<authentication mode="Windows" />
<authentication mode="Forms" />
<authentication mode="Passport" />
<authentication mode="None" />
```

Web–აპლიკაციაზე მომხმარებლის წვდომის უფლებას განსაზღვრავს authentication ელემენტი, ხოლო აპლიკაციის გარკვეულ ნაწილებზე განისაზღვრება authorization ელემეტით: deny და allow ქველემენტების საშუალებით:

*** – განსაზღვრავს ყველა მომხმარებელს,**

? – ანონიმურ მომხმარებლებს.

მაგალითად, ანონიმურ მომხმარებელთათვის საიტზე წვდომის უფლების გასათიშად ვებ-საიტის კონფიგურაციის ფაილში ჩაიწერება:

```
<configuration>
 <system.web>
  <authorization>
   <deny users="?" />
  </authorization>
 </system.web>
</configuration>
```

შესაძებელია ცალკეულ მომხმარებელზე და მომხმარებელთა ჯგუფებზე წვდომის უფლების მინიჭება. შემდეგი ჩანაწერი ნიშნავს, რომ წვდომის უფლება აქვთ someone და Admins ჯგუფში შემავალ მომხმარებლებს:

```
<authorization>
   <allow users="someone" />
   <allow roles="Admins" />
   <deny users="*" />
</authorization>
```

შესაძლებელია აგრეთვე ცალკეულ ვებ-გვერდებზე წვდომის უფლებების დაყენებაც:

```
<location path="webpage.aspx">
 <authorization>
  <allow roles="managers" />
  <deny users="?" />
 </authorization>
</location>
```

წინა ფრაგმენტი გვიჩვენებს, რომ webpage.aspx წვდომის უფლება აქვთ მხოლოდ managers ჯგუფის მომხმარებლებს.

Forms მეთოდით მომხმარებელთა ავტორიზაციის დროს საჭიროა მომხმარებლის სარეგისტრაციო გვერდის მითითება:

```
<configuration>
 <system.web>
  <authentication mode="Forms">
   <forms name=".ASPXCOOKIE" loginUrl="login.aspx" protection="All"
                    timeout="30" path="/"></forms>
  </authentication>
 </system.web>
</configuration>
```

ვებ საიტზე მომხმარებლების ავტორიზაციის მაგალითი:

```
// ფაილი Web.config:
<configuration>
  <system.web>
     <authentication mode="Forms">
       <forms name=".ASPXUSERDEMO" loginUrl="login.aspx" protection="All" timeout="60" />
     </authentication>
     <authorization>
```

```
      <deny users="?" />
     </authorization>
   <globalization requestEncoding="UTF-8" responseEncoding="UTF-8" />
  </system.web>
</configuration>
      // ფაილი Default.aspx:
<%@ Import Namespace="System.Web.Security " %>
<html>
 <script language="C#" runat="server">
   void Page_Load(Object Src, EventArgs E )
    { Welcome.Text = "Hello, " + User.Identity.Name;
    }
   void Signout_Click(Object sender, EventArgs E)
    { FormsAuthentication.SignOut();
      Response.Redirect("login.aspx");
    }
 </script>
 <body>
  <h3><font face="Verdana">Using Cookie Authentication</font></h3>
  <form runat="server" ID="Form1">
   <h3><asp:label id="Welcome" runat="server" /></h3>
   <asp:button text="Signout" OnClick="Signout_Click" runat="server" ID="Button1"
NAME="Button1" />
  </form>
 </body>
</html>

// ფაილი Login.aspx:
<%@ Import Namespace="System.Web.Security " %>
<html>
 <script language="C#" runat="server">

   void Login_Click(Object sender, EventArgs E) {
//authenticate user: this samples accepts only one user with a
//name of someone@www.contoso.com and a password of 'password'
if((UserEmail.Value=="someone")&&(UserPass.Value=="password"))
 {
   FormsAuthentication.RedirectFromLoginPage(UserEmail.Value,
                        PersistCookie.Checked);
 }
else
  { Msg.Text = "Invalid Credentials: Please try again"; }
```

```
}
</script>
<body>
  <form runat="server" ID="Form1">
  <h3><font face="Verdana">Login Page</font></h3>
  <table>
   <tr>
     <td>Email:</td>
     <td><input id="UserEmail" type="text" runat="server"
          NAME="UserEmail" /></td>
     <td
     </td>
   </tr>
   <tr>
     <td>Password:</td>
     <td><input id="UserPass" type="password" runat="server"
          NAME="UserPass" /></td>
     <td>
     </td>
   </tr>
   <tr>
    <td>Persistent Cookie:</td>
    <td><ASP:CheckBox id="PersistCookie" runat="server" />
    </td>
    <td></td>
   </tr>
  </table>
  <asp:button text="Login" OnClick="Login_Click"
        runat="server" ID="Button1" NAME="Button1" />
  <asp:Label id="Msg" ForeColor="red" Font-Name="Verdana"
        Font-Size="10" runat="server" />
 </form>
</body>
</html>
```

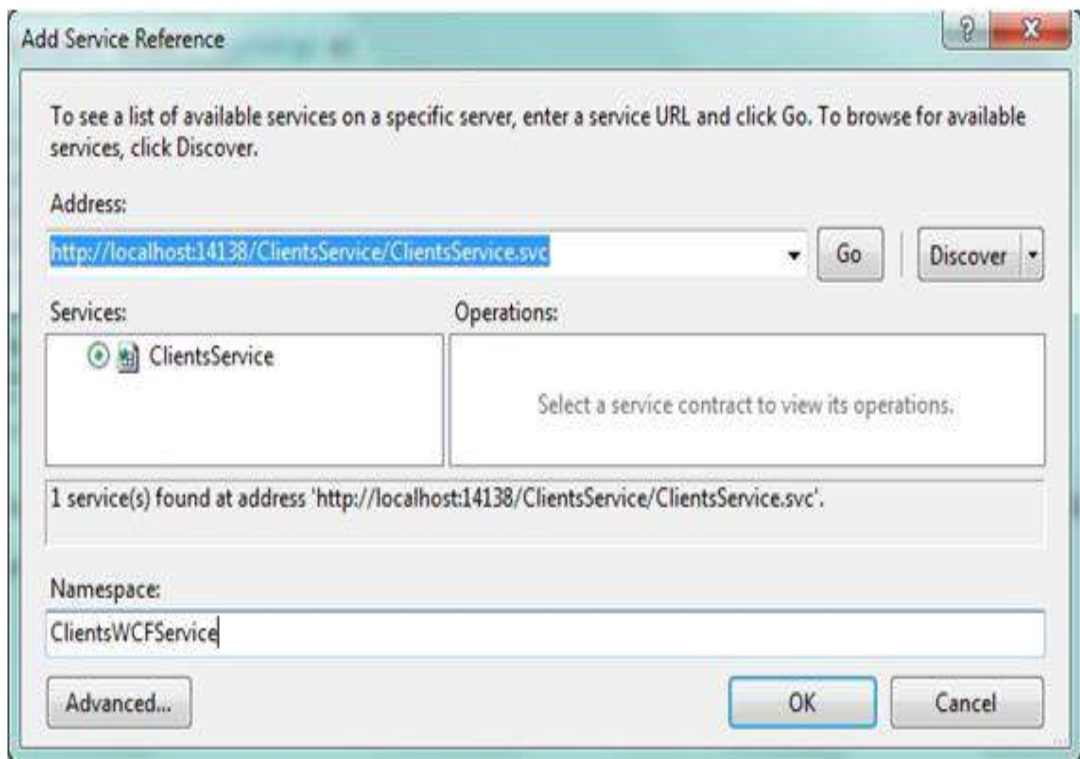
Login Page

Email: [ ]
Password: [ ]
Persistent Cookie: ☐
[Login]

ნახ. დ1-1

- Web-სისტემის მომხმარებელთა ინტერფეისების აგება (სერვისების რეალიზაცია)

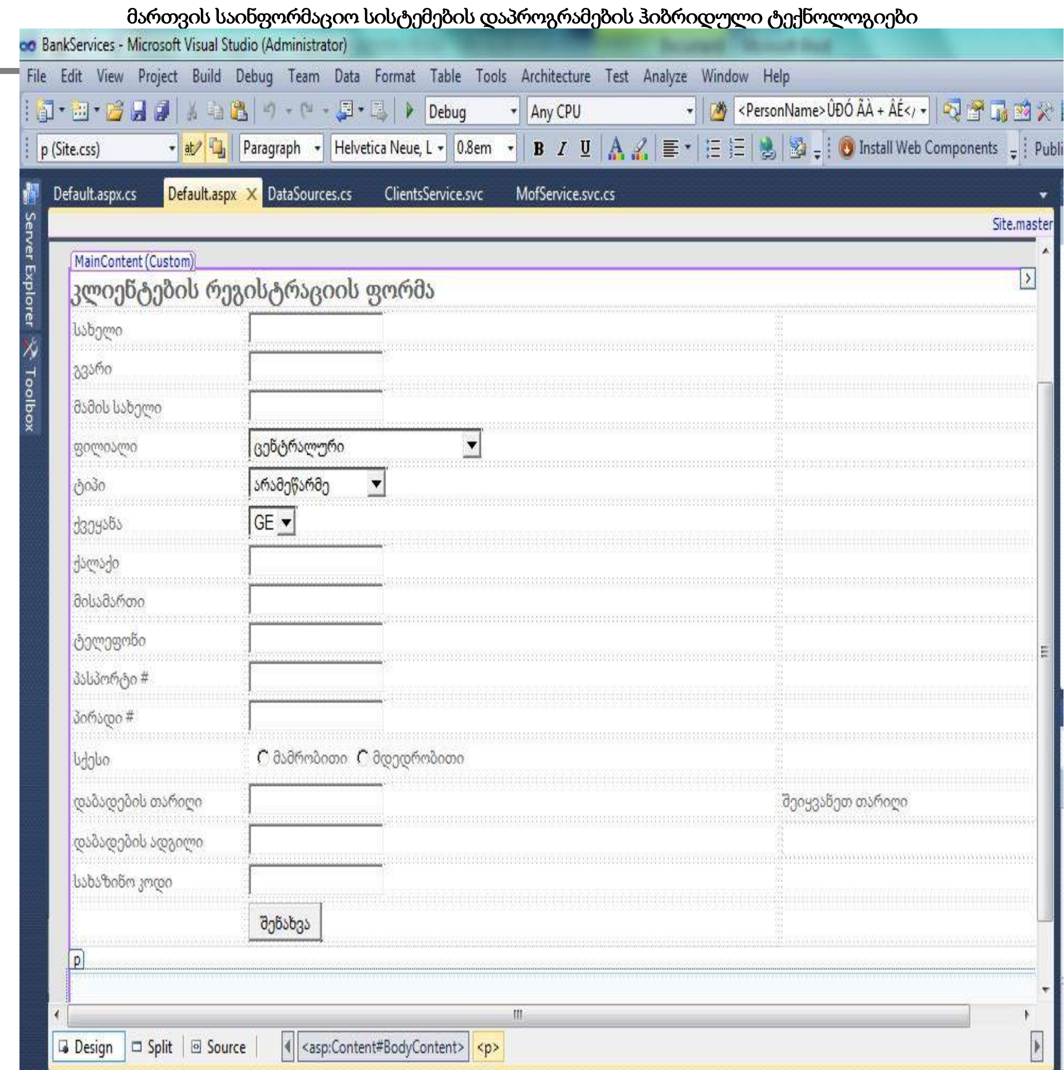
ვებ ინტერეფისის ფორმა შექმნილია ASP.NET აპლიკაციაში. იგი შედგება ვებ-გვერდებისა და კოდის ფაილებისგან. ეს აპლიკაცია მონაცემების ბაზაში წვდომისათვის გამოიყენებს ვებ-სერვისებს. ვებ სერვისები დამატებული Add Service Reference დიალოგური ფანჯრის საშალებით. ამ ფანჯრის სურათი მოცემულია დ1-2 ნახაზზე.



ნახ. დ1-2. Add Service Reference დიალოგური ფანჯარა.

ამ ფანჯარაში გაიწერება ვებ-სერვისი URL. იქმნება სპეციალური სახელების სივცრე (Namespace) და ხდება ვებ-სერვისთან დასაკავშრებელი კლასების გენერაცია. ახალი კლიენტების რეგისტრაციის ფორმა, რომელიც გამოიყენება მათი რეგისტრაციისათვის სისტემაში, მონაცემების ბაზაში შესანახად, მოცემულია დ1-3 ნახაზზე.

Web-გვერდი შედგება HTML და კოდის ფაილებისგან. კოდში გამოყენებულია C# ენა. მომხმარებლის მონაცემების შეტანის შემდეგ „შენახვა“ ღილაკზე დაჭერით მოხდება მონაცემების გადაგზავნა სერვერზე და პროცედურის გამოძახება AddClient_Click. მოცემულია ღილაკის Click მეთოდის ლისტინგი_დ1 :



ნახ. დ1-3. მომხმარებლის ინტერფეისი ASP.NET გარემოში

```
// ლისტინგი_დ1 ---------------------------
  protected void AddClient_Click(object sender, EventArgs e)
{
    ClientsWCFService.ClientsServiceClient service = new
                                        ClientsWCFService.ClientsServiceClient();
   string firstName = txtFirstName.Text;
   string lastName = txtLastName.Text;
   string fatherName = txtFatherName.Text;
   int brachID = Convert.ToInt32(ddlBranch.SelectedValue);
```

```
        byte type = Convert.ToByte(ddlType.SelectedValue);
        string country = ddlCountry.SelectedValue;
        string city = txtCity.Text;
        string address = txtAddress.Text;
        string phone = txtPhone.Text;
        string passport = txtPassport.Text;
        string personalID = txtPersonalID.Text;
        bool gender =  rdlGender.SelectedValue=="0"?false:true;
        DateTime birtDate = DateTime.Parse(txtBirtDate.Text);
        string birtPlace = txtBirtPlace.Text;
        string taxCode = txtTaxCode.Text;
        try
        {
          int clientNo = service.OpenClient(firstName, lastName, fatherName, brachID, type,
                             gender, passport, personalID, country, city, address, phone, birtPlace,
                             birtDate, taxCode);
          if (clientNo != -1)
            Response.Redirect(String.Format("~/Result.aspx?clientId={0}", clientNo));
        }
        catch (Exception ex)
        {
          throw ex;
        }
      }
```

ამ პროცედურაში გამოიძახება ვებსერვისი ClientsWCFService და მისი მეთოდი OpenClient. ამ მეთოდს გადაეცემა ახალი მომხმარებლის მონაცემები. ვებსერვისი კი უზრუნველყოფს ამ მონაცემების ასახვას ბაზაში.

დანართი 2

მონაცემთა ბაზის აგების ექსპერიმენტული მაგალითი MongoDB სისტემაში

დანართში მოყვანილია საცდელი ბაზის შექმნის მაგალითი MongoDB სისტემისთვის [67,118].

moviesDB (ფილმების) მონაცემთა ბაზაში მხოლოდ ერთი კოლექცია გვაქვს - movies, თუმცა, ცხადია, რეალურ სისტემაში ამ კოლექციის გარდა შესაძლოა, გვქონოდა music, games, users, actors და სხვა კოლექცია. ამ კოლექციაში თავმოყრილია მთელი ის ინფორმაცია, რაც ფილმის გარშემო ინახება:

- "_id" - უნიკალური იდენტიფიკატორი (ჩადგმული დოკუმენტით)
- "title" - ფილმის სათაური
- "year" - ფილმის გამოშვების წელი
- "rated" - ფილმის შეფასება
- "released" - საიტზე ფილმის დამატების თარიღი (ჩადგმული დოკუმენტით)
- "runtime" - ფილმის ხანგრძლივობა წუთებში
- "countries" - მწარმოებელი ქვეყნების სია
- "genres" - ფილმის ჟანრების სია
- "director" - ფილმის რეჟისორი
- "writers" - სცენარისტები
- "actors" - მსახიობების სია
- "plot" - ფილმის მოკლე შინაარსი
- "poster" - ფილმის პოსტერის ბმული
- "imdb" - ჩადგმული დოკუმენტი IMDB კინოკრიტიკოსების რეიტინგის შესანახად
- "tomato" - ჩადგმული დოკუმენტი Rotten Tomatoes მაყურებლების რეიტინგის შესანახად
- "metacritic" – Metacritic რეიტინგის შესანახად
- "awards" - ჩადგმული დოკუმენტი ჯილდოების შესანახად
- "type" - სრულმეტრაჟიანი/მოკლემეტრაჟიანი/სერიალი...
- "reviews" - ჩადგმული დოკუმენტების მასივი საიტის მომხმარებლების კომენტარების შესანახად.

MongoDB-ში ბაზას ცხადად არ ვქმნით, ვირჩევთ მიმდინარე ბაზას (რომელიც ახალი ბაზის შექმნის დროს რეალურად არც არსებობს) და არჩეულ ბაზაში უბრალოდ ვამატებთ ახალ ჩანაწერებს. თვითონ მონაცემთა ბაზა პირველივე მონაცემის ჩაწერის პარალელურად, ავტომატურად იქმნება.

```
use movieDB
db.movies.insert([
{
	"_id": {
		"oid": "5692a15524de1e0ce2dfcfa3"
	},
	"title": "Toy Story 4",
	"year": 2011,
	"rated": "G",
	"released": {
		"date": "2010-06-18T04:00:00.000Z"
	},
	"runtime": 206,
	"countries": [
		"USA"
	],
	"genres": [
		"Animation",
		"Adventure",
		"Comedy"
	],
	"director": "Lee Unkrich",
	"writers": [
		"John Lasseter",
		"Andrew Stanton",
		"Lee Unkrich",
		"Michael Arndt"
	],
	"actors": [
		"Tom Hanks",
		"Tim Allen",
		"Joan Cusack",
		"Ned Beatty"
	],
	"plot": "ყველა ბავშვს სჯერა, რომ როდესაც ის თავის სათამაშოებს მარტო ტოვებს, ისინი ცოცხლდება და თავის საქმეებს აკეთებს. ეს ანიმაციური ფილმი მათ თავიანთ აზრებში კიდევ ერთხელ დაარწმუნებს.",
```

"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
"imdb": {
"id": "tt0435761",
"rating": 8.4,
"votes": 500084
},
"tomato": {
"meter": 99,
"image": "certified",
"rating": 8.9,
"reviews": 287,
"fresh": 283,
"consensus": "Deftly blending comedy, adventure, and honest emotion, Toy Story 3 is a rare second sequel that really works.",
"userMeter": 89,
"userRating": 4.3,
"userReviews": 602138
},
"metacritic": 92,
"awards": {
"wins": 56,
"nominations": 86,
"text": "Won 2 Oscars. Another 56 wins \u0026 86 nominations."
},
"type": "movie",
"reviews": [
{
"date": {
"date": "2017-02-13T04:00:00.000Z"
},
"name": "parvesh",
"rating": 8.9,
"comment": "My first review for Toy Story 3, hoping it will execute while trying for the very first time."
},

```
		{
			"date": {
				"date": "2017-02-13T04:00:00.000Z"
			},
			"name": "Prabhash",
			"rating": 8.9,
			"comment": "My first review for Toy Story 3, hoping it will execute while trying for the very first time."
		},
		{
			"date": {
				"date": "2017-02-11T04:00:00.000Z"
			},
			"name": "praveen",
			"rating": 6.7,
			"comment": "My first review for Toy Story 3, hoping it will execute while trying for the very first time."
		}
	]
},{
	"_id": {
		"oid": "589cbda9c0b9fec62febf274"
	},
	"title": "Deadpool",
	"year": 2016,
	"rated": "R",
	"released": {
		"date": "2016-06-18T04:00:00.000Z"
	},
	"runtime": 108,
	"countries": [
		"USA"
	],
	"genres": [
		"Comics character",
		"Adventure",
		"Action"
```

],
"director": "Tim Miller",
"writers": [
"Rhett Reese",
"Paul Wernick"
],
"actors": [
"Ryan Reynolds",
"Morena Baccarin",
"Ed Skrein",
"T.J. Miller",
"Gina Carano",
"Leslie Uggams",
"Stefan Kapičić",
"Brianna Hildebrand"
],
"plot": "ყოფილი სპეცდანიშნულების რაზმის წევრი მონაწილეობას იღებს ექსპერიმენტში, რომელიც მას შესაძლებლობას აძლევს მიიღოს წარმოუდგენელი ძალა, სისწრაფე და უნარი სწრაფი განკურნებისა. ",
"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
"imdb": {
"id": "tt1431045",
"rating": 8.1,
"votes": 585141
},
"tomato": {
"meter": 99,
"image": "certified",
"rating": 6.9,
"reviews": 287,
"fresh": 241,
"consensus": "Fast, funny, and gleefully profane, the fourth-wall-busting Deadpool.",
"userMeter": 90,
"userRating": 4.3,
"userReviews": 181719

},
"metacritic": 92,
"awards": {
"wins": 5,
"nominations": 12,
"text": "wo Golden Globe Award nominations for Best Motion Picture – Musical or Comedy and Best Actor – Motion Picture Musical or Comedy."
},
"type": "movie"
},{
"_id": {
"oid": "589cc22cc0b9fec62febf275"
},
"title": "BATMAN V SUPERMAN: DAWN OF JUSTICE",
"year": 2016,
"rated": "PG-13",
"released": {
"date": "2016-03-19T04:00:00.000Z"
},
"runtime": 151,
"countries": [
"USA"
],
"genres": [
"Action",
"Adventure",
"Sci-Fi"
],
"director": "Lee Unkrich",
"writers": [
"Chris Terrio",
"David S. Goyer"
],
"actors": [
"Amy Adams",
"Henry Cavill",
"Ben Affleck"

],
"plot": "იმის შიშით, რომ მეტროპოლისის სუპერგმირის ქმედებები ისევ დარჩება უკონტროლოდ, გოთემ-სითის რაინდი გამოუცხადებს სუპერმენს ომს. სანამ ისინი არჩევენ საქმეებს და აყენებენ კაცობრიობას არჩევანის წინაშე: თუ რომელი სუპერგმირი უფრო სჭირდებათ მათ, ჩნდება ახალი, გაცილებით საშიში საფრთხე",
"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
"imdb": {
"id": "tt2975590",
"rating": 6.7,
"votes": 3206
},
"tomato": {
"meter": 27,
"image": "certified",
"rating": 4.9,
"reviews": 353,
"fresh": 97,
"consensus": "Batman v Superman: Dawn of Justice smothers a potentially powerful story -- and some of Americas most iconic superheroes -- in a grim whirlwind of effects-driven action.",
"userMeter": 64,
"userRating": 3.6,
"userReviews": 225954
},
"metacritic": 44,
"awards": {
"wins": 6,
"nominations": 26,
"text": "Actor of the Year, Most Original Poster, Best Body of Work"
},
"type": "movie"
},{
"_id": {
"oid": "589cc417c0b9fec62febf276"
},
"title": "doctor strange",

"year": 2016,
"rated": "PG-13",
"released": {
"date": "2016-11-04T04:00:00.000Z"
},
"runtime": 115,
"countries": [
"USA"
],
"genres": [
"Sci-Fi",
"Fantasy",
"Adventure",
"Action"
],
"director": "Scott Derrickson",
"writers": [
"Jon Spaihts",
"Scott Derrickson"
],
"actors": [
"Benedict Cumberbatch",
"Chiwetel Ejiofor",
"Rachel McAdams"
],
"plot": "მას შემდეგ, რაც მისი კარიერა განადგურდა, ბრწყინვალე მაგრამ ქედმაღალი ექიმი ახალ გამოწვევას იღებს ცხოვრებაში, როდესაც ჯადოქარი მას მფარველობაში აიყვანს და წვრთნის, რათა სამყარო დაიცვას ბოროტებისაგან. ",
"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
"imdb": {
"id": "tt1211837",
"rating": 7.8,
"votes": 191181
},
"tomato": {
"meter": 27,

"image": "certified",
"rating": 4.9,
"reviews": 353,
"fresh": 97,
"consensus": "Batman v Superman: Dawn of Justice smothers a potentially powerful story -- and some of Americas most iconic superheroes -- in a grim whirlwind of effects-driven action.",
"userMeter": 64,
"userRating": 3.6,
"userReviews": 225954
},
"metacritic": 44,
"awards": {
"wins": 6,
"nominations": 38,
"text": "Oscar, Best Visual Effects"
},
"type": "movie"
},{
"_id": {
"oid": "589cc696c0b9fec62febf277"
},
"title": "kung fu panda 3",
"year": 2016,
"rated": "PG",
"released": {
"date": "2016-01-29T04:00:00.000Z"
},
"runtime": 95,
"countries": [
"USA"
],
"genres": [
" Animation",
"Action",
"Adventure",
"Comedy",

"Family"
],
"director": " Alessandro Carloni",
"writers": [
" Jonathan Aibel",
"Glenn Berger"
],
"actors": [
" Jack Black",
"Dustin Hoffman",
"Bryan Cranston"
],
"plot": ”კუნგ ფუ პანდა 3” ისევ დიდ თავგადასავალს გვპირდება. ამჯერად პო და მისი ისევ გამოჩენილი ღვიძლი მამა ლი მიემგზავრებიან საიდუმლო პანდების ადგილსამყოფელში, სადაც ისინი ბევრ წინააღმდეგობას აწყდებიან. მათ მთავარ საშიშროებას ცბიერი და ზებუნებრივით დაჯილდოვებული კაი წარმოადგენს. პანდა პოს რთული მისია აკისრია – მან სხვა პანდებს საბრძოლო ხელოვნება უნდა ასწავლოს. არც თუ ისე მარტივი დავალება აქვთ ფუმფულა პანდებს",
"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
"imdb": {
"id": "tt2267968",
"rating": 7.2,
"votes": 83809
},
"tomato": {
"meter": 87,
"image": "certified",
"rating": 6.8,
"reviews": 153,
"fresh": 133,
"consensus": "Kung Fu Panda 3 boasts the requisite visual splendor, but like its rotund protagonist, this sequels narrative is also surprisingly nimble, adding up to animated fun for the whole family.",
"userMeter": 79,
"userRating": 3.9,
"userReviews": 98794

```
        },
        "metacritic": 44,
        "awards": {
                "wins": 0,
                "nominations": 6,
                "text": "Best Animated Feature, Most Wanted Pet"
        },
        "type": "movie"
},{
        "_id": {
                "oid": "589cc846c0b9fec62febf278"
        },
        "title": "zootopia",
        "year": 2016,
        "rated": "PG",
        "released": {
                "date": "2016-04-04T04:00:00.000Z"
        },
        "runtime": 108,
        "countries": [
                "USA"
        ],
        "genres": [
                "Animation",
                "Adventure",
                "Comedy",
                "Crime",
                "Family",
                "Mystery"
        ],
        "director": "Byron Howard",
        "writers": [
                "Byron Howard",
                "Rich Moore"
        ],
        "actors": [
                "Ginnifer Goodwin",
```

"Jason Bateman",
"Idris Elba"
],
"plot": " მოგესალმებით ცხოველთა ქალაქში - თანამედროვე ქალაქი, დასახლებული სხვადასხვა ცხოველებით, დიდი სპილოები პატარა თაგვები. ქალაქი იყოფა რაიონებად და აქ ასევე არის ელიტარული უბანი საჰარის ტერიტორიაზე და ასევე არასტუმართმოყვარე ტუნდრათაუნი. ქალაქში არის ახალი პოლიციელი, მხიარული კურდღელი ჯუდი ჰოპსი, რომელიც სამუშაოს პირველი დღიდანვე აცნობიერებს თუ რა ძნელია იყო პატარა და ფუმფულა დიდ და ძლიერ პოლიციელებს შორის. ჯუდი ცდილობს პირველი შესაძლებლობისთანავე წარმოაჩინოს საკუთარი თავი იმის მიუხედავად, რომ მისი მეწყვილე არის მელია ნიკ უაიდლი. მათ ერთად მოუწევთ რთული საქმის გამოძიება, რის გამოაშკარავებამაც შეიძლება საფრთხე შეუქმნას მთელი ქალაქის მაცხოვრებლებს. ",
"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",
"imdb": {
"id": "tt2948356",
"rating": 8.1,
"votes": 262258
},
"tomato": {
"meter": 98,
"image": "certified",
"rating": 8.1,
"reviews": 241,
"fresh": 236,
"consensus": "Kung Fu Panda 3 boasts the requisite visual splendor, but like its rotund protagonist, this sequels narrative is also surprisingly nimble, adding up to animated fun for the whole family.",
"userMeter": 92,
"userRating": 4.4,
"userReviews": 95658
},
"metacritic": 44,
"awards": {
"wins": 26,
"nominations": 52,

"text": "Best Animated Feature Film of the Year, Best Motion Picture - Animated"
},
"type": "movie"
},{
"_id": {
"oid": "589d733a296ba85b1bc3bee6"
},
"title": "John Carter",
"year": 2012,
"rated": "PG-13",
"released": {
"date": "2012-03-09T04:00:00.000Z"
},
"runtime": 132,
"countries": [
"USA"
],
"genres": [
"Action",
"Adventure",
"Sci-Fi"
],
"director": "Andrew Stanton",
"writers": [
"John Lasseter",
"Andrew Stanton",
"Lee Unkrich",
"Michael Arndt"
],
"actors": [
"Andrew Stanton",
"Mark Andrews"
],
"plot": "ამერიკის სამოქალაქო ომის ვეტერანი ჯონ კარტერი, მარსზე აღმოჩნდება, სადაც ტყვედ ხვდება მტრულად განწყობილ ოთხმეტრიან აბორიგენებთან. კარტერი არამარტო თავად უნდა გადარჩეს, არამედ პრინცესა დეა ტორისიც უნდა გადაარჩინოს. ",
"poster": "http://ia.media-imdb.com/images/M/MV5BMTgxOTY-4Mjc0MF5BMl5BanBnXkFtZTcwNTA4MDQyMw@@._V1_SX300.jpg",

```
	"imdb": {
		"id": "tt0401729",
		"rating": 6.6,
		"votes": 217518
	},
	"tomato": {
		"meter": 51,
		"image": "certified",
		"rating": 5.7,
		"reviews": 219,
		"fresh": 111,
		"consensus": "While John Carter looks terrific and delivers its share of pulpy
thrills, it also suffers from uneven pacing and occasionally incomprehensible plotting and
characterization.",
		"userMeter": 60,
		"userRating": 3.5,
		"userReviews": 113966
	},
	"metacritic": 92,
	"awards": {
		"wins": 2,
		"nominations": 7,
		"text": "Top Box Office Films, Best Original Score for a Fantasy/Science
Fiction/Horror Film"
	},
	"type": "movie"
}
])
```

დანართი_3

Web-სერვისის რეალიზაცია შავი ზღვის მდინარეთა ესტუარების მონიტორინგის სისტემისათვის

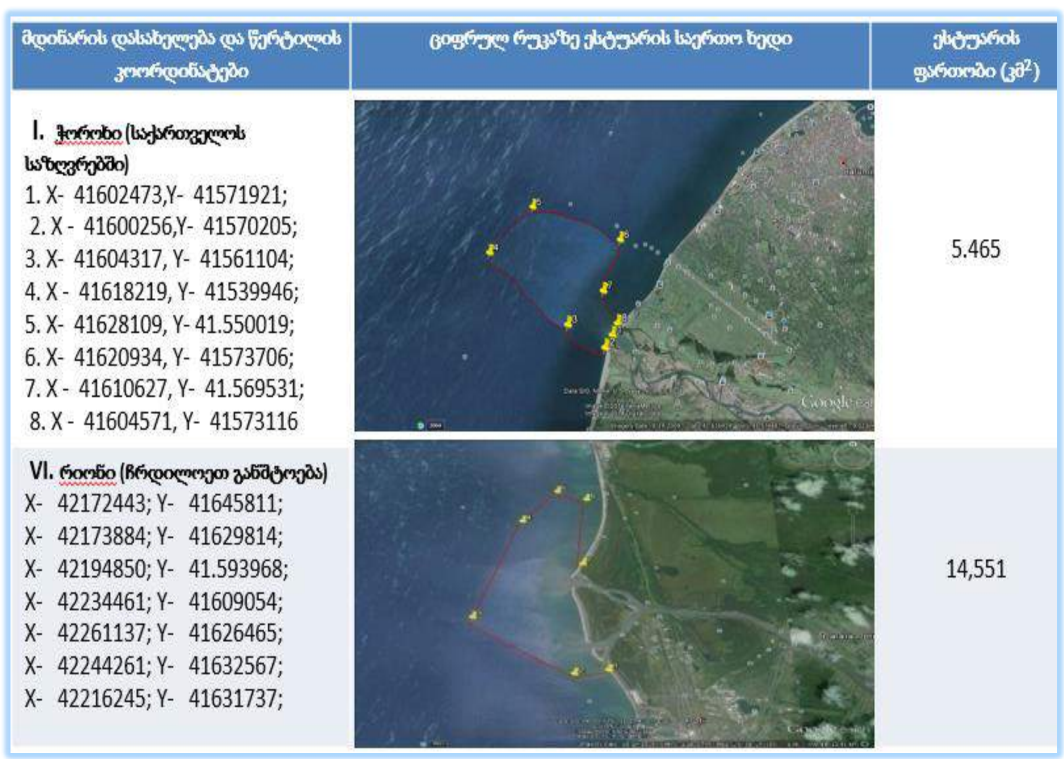
დანართში გადმოცემულია სტუ-ს „მართვის ავტომატიზებული სისტემების" დეპარტამენტში შესრულებული სამეცნიერო-კვლევითი სამუშაოს შედეგები [152], კერძოდ, საქართველოს შავი ზღვის აკვატორიაში მდინარეთა ესტუარების საველე-სამეცნიერო კვლევის შედეგების სერვერზე დისტანციურად განთავსების, ანალიზისა და მონიტორინგის ამოცანების გადაწყვეტა. სისტემა საშუალებას გვაძლევს გამოვლენილ იქნას სანაპირო ზოლის ახალი მოწყვლადი უბნები შესაბამისი ფართობებით გეოგრაფიული GPS კოორდინატების მიხედვით. სისტემა რეალიზებულია Ms SharePoint Server-ის, Business Data Connectivity და Infopath-ის დინამიკური ფორმების საშუალებით.

შეიძლება ითქვას, რომ ბოლო წლებში კლიმატის ცვლილების გათვალისწინებით იმატა შავი ზღვის აკვატორიაში მდინარეთა კალაპოტებში წყალდიდობათა წარმოშობის სიხშირემ, რომელთა ესტუარებში და მის მიმდებარე ტერიტორიებზე ხშირია დატბორვა, ზღვის სანაპირო ზოლის ეროზია (აბრაზია), რაც მთავრდება ძალზე უარყოფითი ეკოლოგიური შედეგით, კერძოდ, მიმდინარეობს ზღვის მიერ საქართველოს საზღვრებში სანაპირო ზოლის - ხმელეთის მიტაცება და უფრო შიგნით ხმელეთის სიღრმეში ზღვის შემოსვლა [153].

ყოველივე ზემოთ აღნიშნულის გათვალისწინებით, შავი ზღვის ეკოლოგიური საკითხების მეცნიერული კვლევა და მისი პროგნოზირება საქართველოსათვის მეტად აქტუალურია, იგი წარმოადგენს ქვეყნის სტრატეგიულ მიმართულებას, ხოლო შავი ზღვის სანაპირო ზოლისა და მისი მიმდებარე ტერიტორიების შენარჩუნება და დაცვა ქვეყნის მთავრობას აღიარებული აქვს როგორც სახელმწიფოს პრიორიტეტული მიმართულება [154].

საველე-სამეცნიერო კვლევის მიზანს წარმოადგენდა შავი ზღვის აუზის საქართველოს ზემოთ დასახელებული ძირითადი მდინარეების ესტუარების ფართოებების დაზუსტება GPS-ს კოორდინატებში და მათი დატანა ციფრულ რუკებზე (ნახ.დ3-1), ზღვის წყლისა და ჰაერის ტემპერატურის დაფიქსირება, ასევე ზღვის მოწყვლად უბნებსა და ესტუარებში ზღვის წყლის ანალიზის აღება და მათი ქიმიური ლაბორატორიული გამოკვლევა [155].

საველე კვლევების ჩატარება, კერძოდ სხვადასხვა მაჩვენებლების დაფიქსირება GPS-კოორდინატების მიხედვით მეტად მოხერხებული იქნება, თუ მონაცემების შეტანა შესაძლებელი იქნება ტერიტორიულად დაშორებული კომპიუტერიდან მონაცემები შეტანისთანავე აისახება კორპორატიული პორტალის ვებგვერდზე და ტერიტორიულად დაშორებულ SQL Server-ის ბაზაში [91, 152].



მდინარის დასახელება და წერტილის კოორდინატები	ციფრულ რუკაზე ესტუარის საერთო ხედი	ესტუარის ფართობი (კმ²)
I. ჭოროხი (საქართველოს საზღვრებში) 1. X- 41602473,Y- 41571921; 2. X - 41600256,Y- 41570205; 3. X- 41604317, Y- 41561104; 4. X - 41618219, Y- 41539946; 5. X- 41628109, Y- 41.550019; 6. X- 41620934, Y- 41573706; 7. X - 41610627, Y- 41.569531; 8. X - 41604571, Y- 41573116		5.465
VI. რიონი (ჩრდილოეთ განშტოება) X- 42172443; Y- 41645811; X- 42173884; Y- 41629814; X- 42194850; Y- 41.593968; X- 42234461; Y- 41609054; X- 42261137; Y- 41626465; X- 42244261; Y- 41632567; X- 42216245; Y- 41631737;		14,551

ნახ. დ3-1. შავი ზღვის აუზის საქართველოს მდინარეების: ჭოროხის და რიონის ესტუარის GPS-კოორდინატების მონაცემები

მონიტორინგის სამსახურის ექსპერტები აღნიშნულ საკონტროლო წერტილებში იღებენ წყლის სინჯებს ანალიზისათვის და შემდეგ ანალიზის შედეგებს გადასცემენ სისტემის ცენტრალურ სერვერს მობილური ტექნიკით [152].

დ3-2 ნახაზზე წარმოდგენილია ცხრილი, რომელშიც ასახულია ეკოლოგიური მაჩვენებლების მნიშვნელობები, შეტანილი საკონტროლო წერტილებიდან. აქ მონაცემების შეტანის დრო ფიქსირდება ავტომატურად.

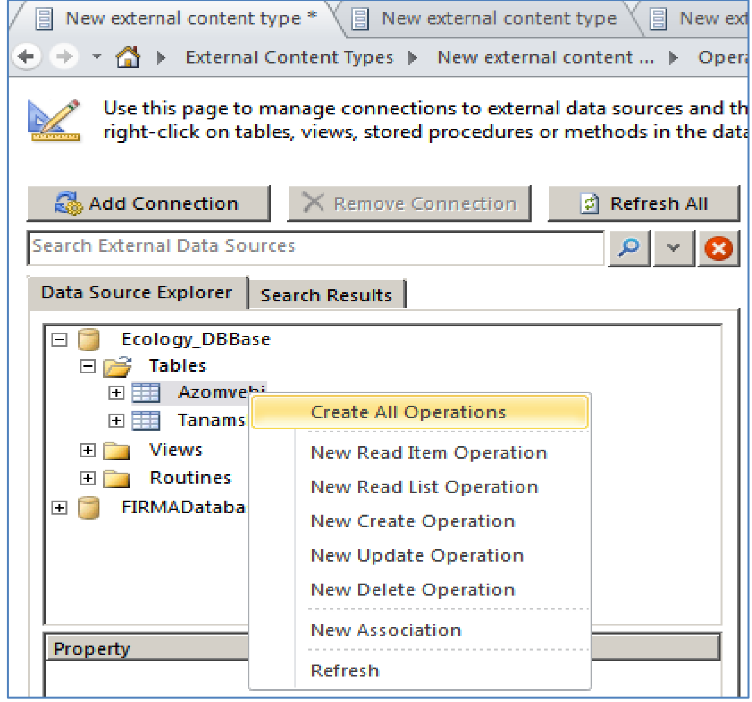
შემდეგ ეტაპზე საჭიროა MsSQL Server-ის მონაცემთა ბაზის დაკავშირება ვებ-პორტალთან. ამ ამოცანის გადასაჭრელად გამოყენებულია Sharepoint Designer-ი. დ3-3 ნახაზზე ასახულია მონაცმთა ბაზასთან დაკავშირება პორტალის External List-თან.

omvebi* | WIN-K8FK5Q18...bo.Azomvebi*

asaxeleba, GPS_X, GPS_Y, Temp, Mjavianoba, Marilianoba, GETDATE() AS Dro, Tanamshromeli_ID

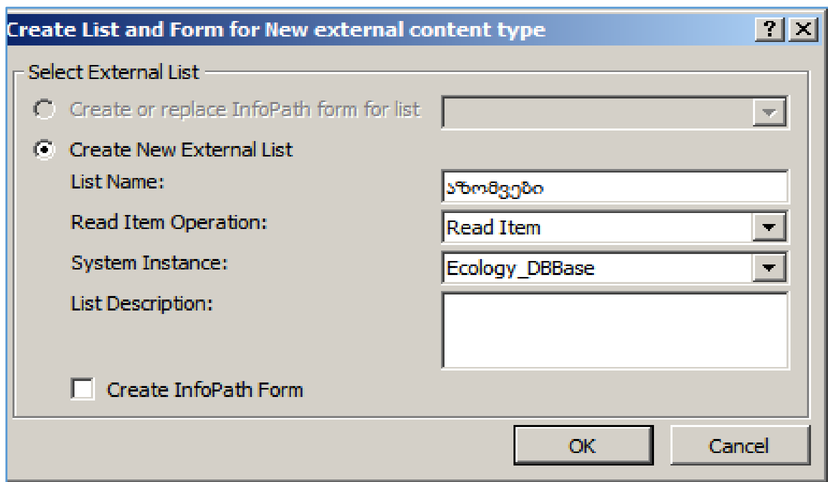
dasaxeleba	GPS_X	GPS_Y	Temp	Mjavianoba	Marilianoba	Dro	Tanams
sarfi	41526956	41548731	45.90	67.89	45.78	2016-06-22 06:24:58.037	NULL
kvariati_1	41545542	41561587	67.00	67.00	34.00	2016-06-22 06:24:58.037	1
kvariati_2	41554651	41563841	NULL	NULL	NULL	2016-06-22 06:24:58.037	NULL
gonio	41574588	41565589	NULL	NULL	NULL	2016-06-22 06:24:58.037	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

ნახ. დ3-2.



ნახ. დ3-3. მონაცემთა ბაზის სერვერთან დაკავშირება

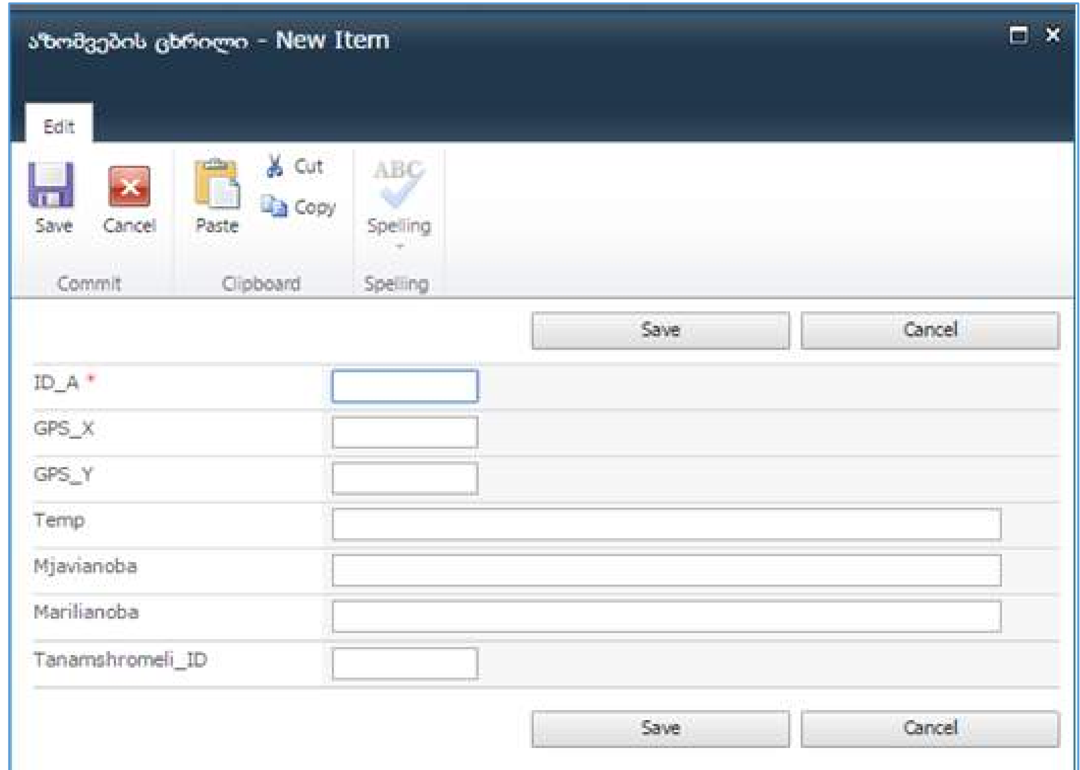
დ3-4 ნახაზზე ნაჩვენებია გარე სიის შექმნის პროცესი, რომელიც უკვე დაკავშირებულია მონაცემთა ბაზასთან.



ნახ.3

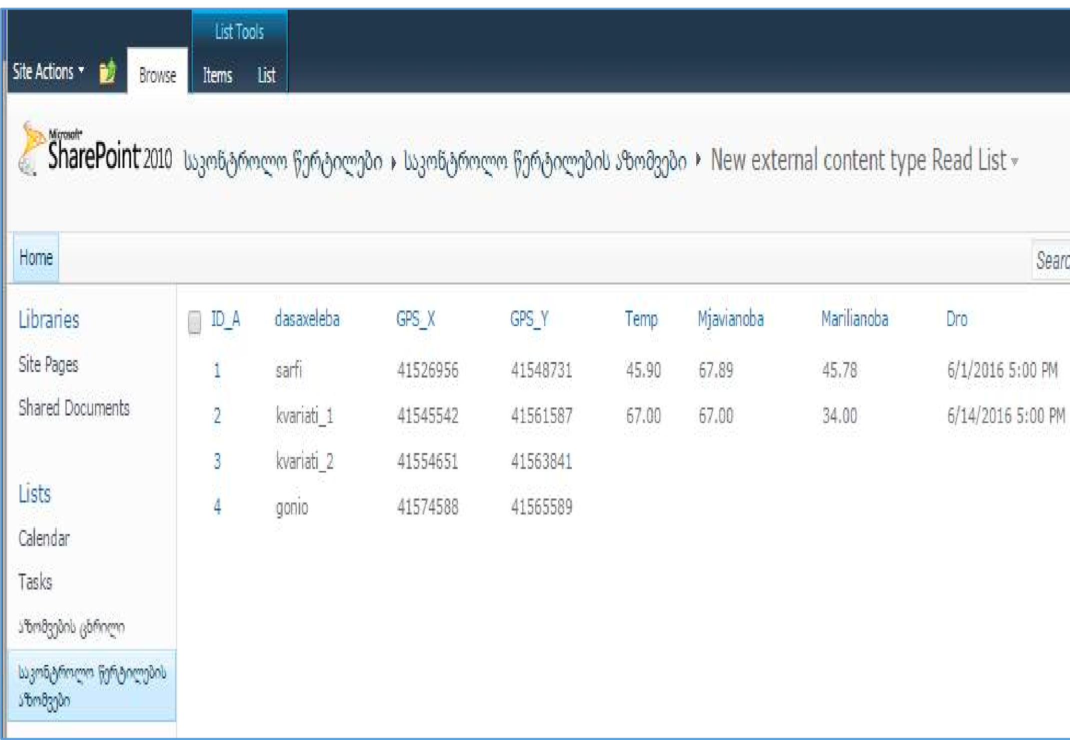
მონაცემების შეტანის პროცესი:

დ3-5 ნახაზზე ნაჩვენებია მონაცემების შეტანისთვის აუცილებელი დიალოგური ფანჯარა:



ნახ. დ3-5. მონაცემების შეტანის დიალოგური ფანჯარა

დ3-6 ნახაზზე მოცემულია ორგანიზაციის ვებ-პორტალზე ასახული საკონტროლო წერტილების აზომვების ცხრილი.



ID_A	dasaxeleba	GPS_X	GPS_Y	Temp	Mjavianoba	Marilianoba	Dro
1	sarfi	41526956	41548731	45.90	67.89	45.78	6/1/2016 5:00 PM
2	kvariati_1	41545542	41561587	67.00	67.00	34.00	6/14/2016 5:00 PM
3	kvariati_2	41554651	41563841				
4	gonio	41574588	41565589				

ნახ. დ3-6. აზომვის ფიქსირებული მნიშვნელობები საკონტროლო წერტილებში

ამგვარად, ყოველივე ზემოთ აღნიშნული საშუალებას მოგვცემს კომპლექსურად შევაფასოთ შავი ზღვის თანამედროვე ეკოლოგიური პრობლემები და დაიგეგმოს მისი სანაპირო ზოლისა და მიმდებარე ტერიტორიების ეკოლოგიური უსაფრთხოების ღონისძიებები.

Index/ინდექსი:

ACID -	Atomicity, Consistency, Isolation, Durability	711,713,715
AM -	Agile Modeling	126,234,294, 296,300,305,
API -	Application Programming Interface	23,25,321,709, 710,715,716, 757,
APICS -	American Production and inventory control society	905
AS -	Applied Software	18
ASP -	Active Server Pages	26,37,39,54,56 ,230,316,318, 319,365,800
BAM -	Business Activity Monitoring	226
BAML -	Binary Application Markup Language	345
BCM -	Business Continuity Management	188,189,190, 191,192
BCP -	Business Continuity Plan	189
BIA -	Business Impact Analysis	192,193
BPD -	Business Process Diagram	230,903,904
BPEL -	Business Process Execution Language	230,232,902
BPM -	Business Process Management	221,222,
BPMN -	Business Process Modeling Notation	229,230,231, 900,901,902, 903,906, 907,909,910
BPR -	Business Process Reengineering	915,916
BSC -	Balanced Score Card	151
BSI -	British Standards Institution	15,126,127. 128.129,132, 133,141,143, 774
BU -	Business unit	159

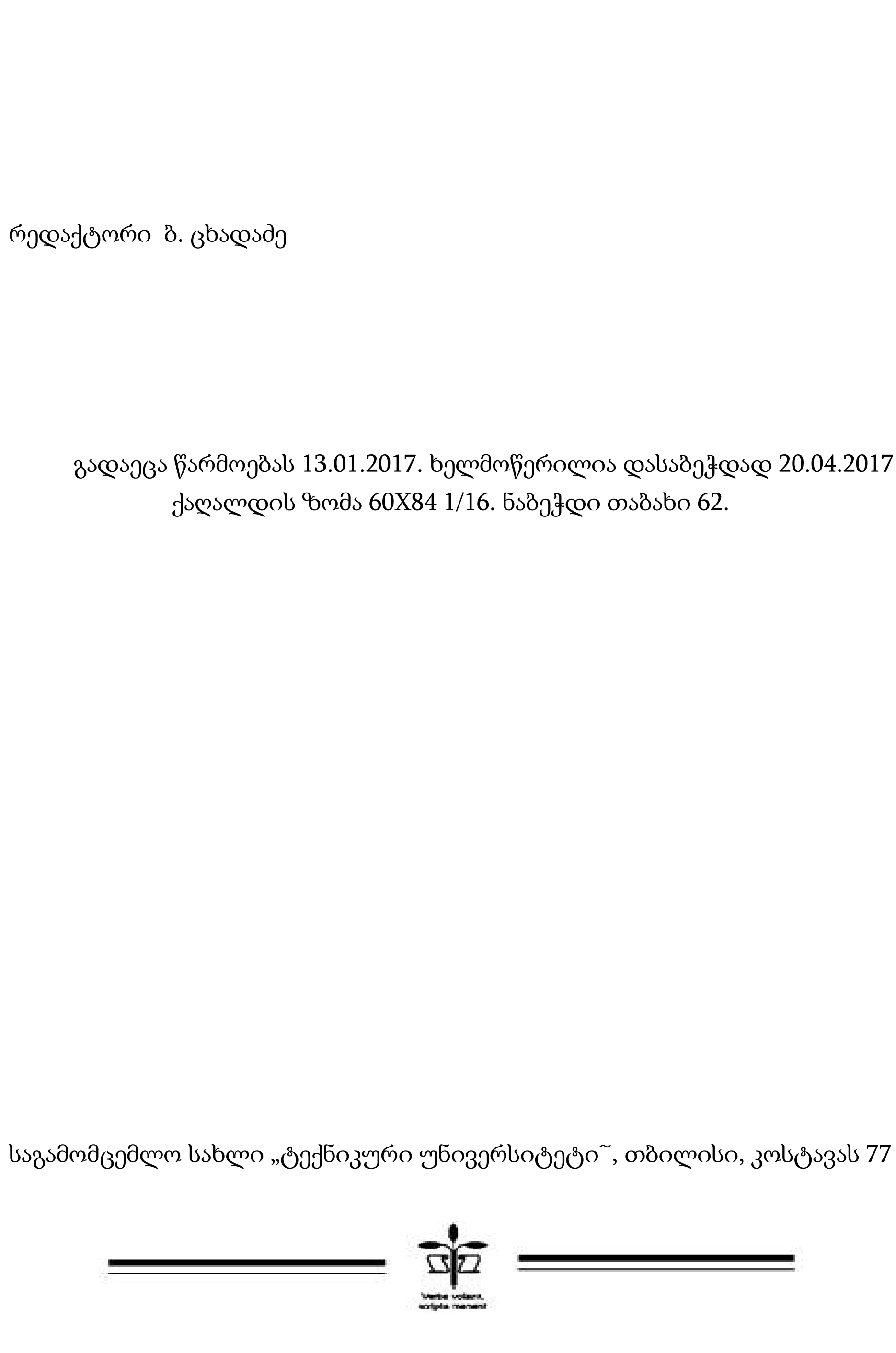
CASE -	Computer Aided System Engineering	107,126,230, 234,243,250, 296,304,314, 774,801,802, 817,818,871, 900,
CLR -	Common Language Runtime	26,27,49,348, 478,761
COBIT -	Control Objectives for Information and related Technology	15,126,127, 133,134,212, 213,214,215, 216,774,
CRM -	Customer Relationship Management;	904
CPN -	Coloured Petri Netzwork	230,900,937, 950,952,
CSF -	Critical Success Factor -	151
CSS -	Cascading Style Sheets	55
CTS -	Common Type System	27
DBMS -	Data Base Management System	668, 731
DDL -	Data Definition Language	753,780,782, 809,825
DML -	Data Manipulation Language	809
DPAPI -	Data Protection API	757
EA -	Enterprise Architect	236,250,254, 303,311,312, 774,805,906,
EAI -	Enterprise Application Integration	217,219,221,
EDI -	Electronic Data Interchange	217,221,222
ERM -	Entity Relation Model	701,702,779, 780,801,818, 895,
ERP -	Enterprise Resource Planning	40
ESB -	Enterprise Service Bus	220,222,
Hadoop -	BigData Platform	718,719,720, 725

HR -	Human Resources	904
HTML -	HyperText Markup Language	27,35,55,227, 318,325,463
HTTP -	HyperText Transfer Protocol	40,41,55,219, 228,644,710
GUI -	Graphical User Interface	26,735,
IDE -	Integrated development environment	230,482
ISACA	Information Systems Audit and Control Association	127,134,212,
ISEB -	Information Systems Examination Board	146,962
ISMS -	Information Security Management System	128,131,132, 134,200,202, 205,207,
ISO -	International Organization for Standardization	128,130,131, 132,133,137, 141,145,
IT -	Information Technology	146
ITIL -	Information Technology Infrastructure Library	15,40,126,133, 134,146,147, 152,158,164, 168,176,180, 184,188,190, 774
ITSCM -	IT Service Continuity Management	188,190,192, 193,194,195, 197,198,200
ITSM	IT Service Management	149,159,177, 208
KPI -	Key Performance Indicator	151,152,209, 226
Linux -	Computer Operating System	23,27,710,711, 712,715,
LINQ -	Language-Integrated Query	364,505,
MariaDB -	Data Base Management System	715
MDA -	Model driven Architecture	230

MemSQL -	Memory Storage Structured Query Language	712
M_o_R -	Management of Risks	194
MRP -	Material requirement planning	904
MRP2 -	Manfactory reasource planning	904
BizTalk -	Microsoft BizTalk_Server	222,223,224, 225
MSF -	Microsoft Solution Framework	110,111,
MVVM -	Model View Viewmodel	55
MVC -	Model View Controller	55
MVCC	MultiVersion Concurrency Control	716
MySQL	Data Base Management System	713
NewSQL	New Structured Query Language	711
NoSQL	No Structured Query Language	708,709,710, 713
OLA -	Operational-Level Agreement	203,204
OMG -	Object Management Group	236,237
OMT -	Object-modeling technique	236
OOSE -	Object-oriented software engineering	236
ORM -	Object Role Modeling	230,701,775, 776,802,818, 822,825,893, 895
PDCA -	Life Cycle model: „Plan", „Do", „Check", „Act"	137
RFC -	Request for Change	209
RF -	Fast Recovery	196
RG -	Gradual Recovery	195
RIm -	Immediate recovery	196
RIn -	Intermediate Recovery	196
SAP -	Systems, Applications & Products in Data Processing	40,222,906
Scrum -	Agile method	23,300,
SDP -	Service Design Package	155,156,176
SE -	Software Engineering	18,20,126,234, 304,

SKMS	Service Knowledge Management System	156
SLA -	Service Level Agreement	179,197,200, 204206
SLR -	Service Level Requirements	177,179,199, 206
SME -	Small and Medium-sized Enterprise	227
SOA -	Service Oriented Arcitecture	220,222,223, 793,
SOAP	Simple Object Access Protocol	40,41,43,572, 643,784
SPO -	Service Provisioning Optimization	172,195
SQL -	Structured Query Language	686,690,692, 702,
T-SQL -	Transact-SQL	750,751,753, 754
UML -	Unified Modeling Language	19,36,77,78, 82,126,229, 236,237,238, 240,261,273
URL -	Uniform Resource Locator	42,573,592, 643,974
VCD -	Variable Cost Dynamics	175
WCF -	Windows Communication Foundation	15,37,41,44, 315,316,572, 582,591,611, 637,651.653, 783,789,793, 851,958
WF -	Workflow Foundation	15,38,39,480, 559,560,561, 565,783,962

WPF -	Windows Presentation Foundation	15,38,46,48, 315,316,319, 320,322,324, 326,328,330, 331,336,338, 344,350,365, 370,373,378, 381,388,394, 398,409,426, 433,440,445, 450,463,472, 600,774,793, 801,808,817, 872,900
XAML -	Extensible Application Markup Language	39,45,49,318, 334,345,349, 494,602,784, 794,836,842
XML -	Extensible Markup Language	39,45,218,224, 227,228,230, 231,318,344, 709,711,784, 791,828
XP -	Extreme Programming	314



რედაქტორი ბ. ცხადაძე

გადაეცა წარმოებას 13.01.2017. ხელმოწერილია დასაბეჭდად 20.04.2017.
ქაღალდის ზომა 60X84 1/16. ნაბეჭდი თაბახი 62.

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი˜, თბილისი, კოსტავას 77