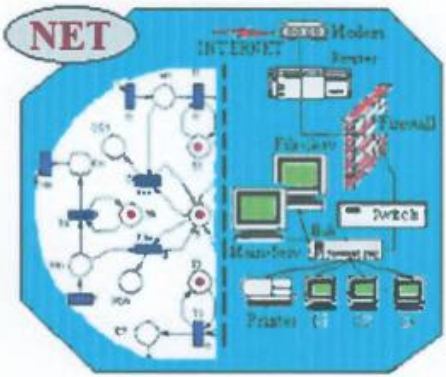
გია სურგულაძე, დავით გულუა

# ქსელური არქიტექტურები ბიზნესისათვის




„IT-კონსალტინგის ცენტრი“

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, დავით გულუა

# ქსელური არქიტექტურები ბიზნესისათვის

დამტკიცებულია:
სტუ-ს „IT კონსალტინგის“
სარედაქციო-საგამომცემლო
კოლეგიის მიერ 10.03.2017

თბილისი
2017

**უაკ 004.5**

განხილულია კორპორაციული კომპიუტერული ქსელების სერვერული სისტემების არქიტექტურა, მისი ინფრასტრუქტურის დაპროექტებისა და აგების ახალი, ინტენსიონალური და ექსენსიონალური მიდგომები, მთლიანი სისტემის წარმადობისა და საიმედოობის სრულყოფის მიზნით. დეტალურად არის წარმოდგენილი ვირტუალიზაციის, კლასტერული და სხვა ტექნოლოგიები, რომლებიც ხელს უწყობს მოცემული ამოცანის მაქსიმალური ეფექტურობით შესრულებას. წარმოდგენილია აგრეთვე სისტემების ადმინისტრირების საფუძვლები, ვებ-, ფაილური და მონაცემთა ბაზის სერვერების გამართვა. დეტალურადაა გადმოცემული ვებ-სერვერული ინფრასტრუქტურის აგებულება, მისი აწყობისა და მართვის საკითხები, საფოსტო და საკომუნიკაციო სერვერების გამართვა. შემოთავაზებულია MsVisual Studio.NET Framework 4.5 ინტეგრირებულ გარემოში ჰიბრიდული კომპიუტერული სისტემების (Windows- და Web-აპლიკაციების) დაპროგრამების ინსტრუმენტული საშუალება Windows Communication Foundation (WCF) ტექნოლოგიის ბაზაზე. დამხმარე სახელმძღვანელო განკუთვნილია ინფორმატიკისა და მართვის საინფორმაციო სისტემების სპეციალობის მაგისტრანტებისა და ბაკალავრიატის მაღალი კურსის სტუდენტებისათვის.

**რეცენზენტები:**

- პროფ. ე. თურქია
- ასოც.პროფ. ი. ქართველიშვილი

პროფ. გ. სურგულაძის რედაქციით

**© სტუ-ის „IT-კონსალტინგის სამეცნიერო ცენტრი", 2017**
ISBN 978-9941-0-9842-0

# შინაარსი

# შესავალი

თანამედროვე ინფორმაციული ტექნოლოგიების განვითარება კომპიუტერული ქსელების წინაშე ახალ ამოცანებს აყენებს. კორპორაციული ინფორმაციის მოცულობისა და ქსელებში ინფორმაციული ნაკადების ზვავისებრი ზრდის გამო ქსელში ინფორმაციის ტრადიციული მეთოდებით შენახვა, დამუშავება და გადატანა დანახარჯების მკვეთრ ზრდას იწვევს, ხოლო მთლიანი ქსელის ფუნქციონირების ეფექტურობა მკვეთრად ეცემა.

გასული საუკუნის ბოლოდან ინფორმაციულ ტექნოლოგიებში კორპორაციული ქსელების დაპროექტების და აგების ახალი მეთოდები მკვიდრდება, რომელთა განვითარების პიკსაც დღევანდელობა წარმოადგენს. უფრო ზუსტად, ახალი მეთოდოლოგია 80-იანი წლების დასაწყისში დროებით „დავიწყებული“ მეინფრეიმულ-სუპერკომპიუტერული არქიტექტურისა და რესურსების ვირტუალიზაციის ტექნოლოგიის ხელახალი აქტუალიზაციის შედეგია. პერსონალური კომპიუტერების განვითარებასთან ერთად თითქოს სუპერკომპიუტერების საჭიროება მოიხსნა, მაგრამ ინფორმატიკის ახალმა რეალობებმა ისინი ხელახლა დააბრუნა ასპარეზზე, ოღონდ მნიშვნელოვნად შეცვლილი სახით.

წიგნის 1-ელი ნაწილი ეხება თანამედროვე კორპორაციული ქსელების ისეთ მნიშვნელოვან სფეროს, როგორიცაა სერვერული ინფრასტრუქტურა. იგი ნებისმიერი დიდი კომპიუტერული ქსელის „გულია“ და მის გამართულ მუშაობაში მთავარ როლს თამაშობს. მეორე მხრივ, სწორედ სერვერები შეიცავს კომპიუტერული ქსელის ყველაზე ძვირადღირებულ რესურსებს და შესაბამისად, ყოველი სერვერული კომპონენტი თავის ღირებულებას მაქსიმალურად უნდა პასუხობდეს, რაც მათი დატვირთვების ეფექტურ განაწილებას მოითხოვს. მოცემულ ნაშრომში აღნიშნულ საკითხს ასევე საკმაო ყურადღება ექცევა [1-3].

პრაქტიკული ნაწილი რეალური კორპორაციული ქსელის მოდერნიზების პროცესში დასმულ ამოცანებს და მათი გადაწყვეტისთვის გამოყენებულ მეთოდებს ეხება [4-8]. ძირითადი აქცენტები თანამედროვე ინფორმაციული ტექნოლოგიების უმნიშვნელოვანეს მიმართულებებზე: ვირტუალიზაციასა და კლასტერულ არქიტექტურაზე კეთდება. განხილულია მეთოდებიც, რომელთა საშუალებითაც სერვერული სისტემების ძველი არქიტექტურის ახალ ტექნოლოგიებზე მორგება ხდება შესაძლებელი [9-12].

წიგნის მე-2 ნაწილში განხილულია კომპიუტერული ქსელური სისტემების ადმინისტრირების საფუძვლები. კერძოდ აქ გადმოცემულია გამოთვლითი რესურსების მართვის ცენტრალიზებული სისტემა - კატალოგური სერვისების სახით, მონაცემთა ბაზითა და სერვისის დომენური სტრუქტურით; წარმოდგენილია მნიშვნელოვანი საკითხები ქსელის მომხმარებელთა და გამოთვლითი რესურსების მართვის ამოცანების გადასაწყვეტად Windows გარემოში [13-16]. კერძოდ, განიხილება ჯგუფური პოლიტიკის მართვის კონკრეტული ამოცანები, როგორიცაა მაგალითად, ფოლდერების გადამისამართება (Folder Redirection), სისტემურ დისკოზე მიმართვის შეზღუდვა, პაროლების მართვა, დომენური ჯგუფის ჩასმა ლოკალურ ჯგუფში და ა.შ. ამავე ნაწილში შემოთავაზებულია ვებ-სერვერული ინფრასტრუქტურის აწყობისა და მართვის საკითხები (Internet Information Server-ს გარემოში), აგრეთვე ფაილური, ბეჭდვის და სკანირების სერვისების მართვა, მათი საიმედოობის უზრუნველყოფისა და რეპლიკაციების აგების საკითხები. ბოლოს განიხილება საფოსტო და საკომუნიკაციო სერვერების გამართვის საკითხები Microsoft Exchange Server-ის გარემოში [17-19].

წიგნის მე-3 ნაწილი ეხება ჰიბრიდული პროგრამული აპლიკაციების (დანართების) აგების საკითხებს (ორი ნაირსახეობაა ცნობილი: ვინდოუს სისტემები, რომელთაც ასევე სამაგიდო

აპლიკაციებს უწოდებენ და ვებ-აპლიკაციები, რომელთა გამოყენებაც ინტერნეტ ბრაუზერებიდანაა შესაძლებელი) [20-23].

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 მომხმარებლისთვის. მაგრამ თანამედროვე კომპიუტერული ტექნოლოგიების სამყაროში ამ ორი სახის აპლიკაციებს შორის საზღვრები სულ უფრო და უფრო იშლება [24].

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, ასევე გლობალურ ქსელებში. ამ ნაწილში დეტალურად განიხილება Windows Communication Foundation ტექნოლოგიის ძირითადი ცნებები, საბაზო სტრუქტურები და მათი გამოყენების მეთოდოლოგიური საკითხები, დაპროგრამების პრინციპების თვალსაზრისით. ემოთავაზებულია კლიენტ-სერვერული და ვებ-სერვისული აპლიკაციების აგების პრაქტიკული საკითხები.

დამხმარე სახელმძღვანელო, რომელიც ფაქტობრივად, არის „ქსელური არქიტექტურები ბიზნესისათვის“, სასარგებლო იქნება მაგისტრატურის და ბაკალავრიატის მაღალი კურსის სტუდენტებისათვის, აგრეთვე მართვის განაწილებული საინფორმაციო სისტემების დაპროექტებისა და აგების საკითხებით დაინტერესებული სპეციალისტებისათვის.

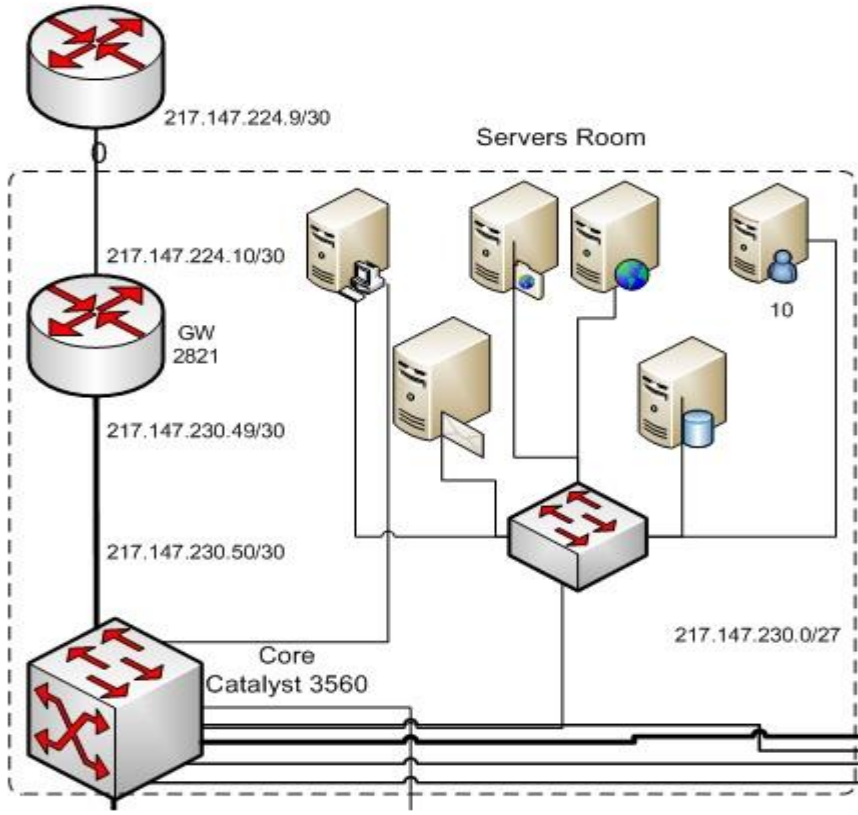
# ნაწილი I

# სერვერული სისტემები კორპორაციულ ქსელებში

# თავი 1: თანამედროვე სერვერული სისტემები

## 1.1 კორპორაციული ქსელის სერვერული სისტემის ზოგადი სტრუქტურა

კორპორაციული სერვერული სისტემა ორ ძირითად ქვესისტემად შეიძლება დავყოთ: საკუთრივ სერვერული და გარემოსთან ურთიერთობის ანუ ქსელური ქვესისტემები (ნახ.1.1 ).




ნახ.1.1. სერვერული და ქსელური ქვესისტემების ფრაგმენტი

სერვერული სისტემის ქსელური ქვესისტემა მოიცავს მაგისტრალურ და სხვა ტიპის ქსელურ აპარატურას (რუთერები, მე-3 დონის კომუტატორები, კონვერტირების მოწყობილობები და სხვა), რომლებიც ინტერნეტთან მიერთებას და კორპორაციული ქსელის სხვადასხვა სეგმენტების ურთიერთკავშირს უზრუნველყოფენ. სერვერული სისტემა ქსელის მომხმარებელთათვის განკუთვნილი მრავალფეროვანი სერვისების (ფაილ-სერვისი, მონაცემთა ბაზები, ელექტრონული ფოსტა, ვები და სხვა) განთავსებისა და მართვის ამოცანას ემსახურება.

სტანდარტული სერვერული სისტემა თითოეული სერვისისთვის, როგორც წესი, ცალკე ფიზიკური მანქანის ან მანქანების (სერვერების) გამოყოფას ითვალისწინებს. შეზღუდული რესურსების არსებობისას შეიძლება რამდენიმე სერვისი ერთსა და იმავე სერვერზე იყოს განთავსებული. არსებობს საკმაოდ მოქნილი მეთოდები მონაცემთა საიმედო შენახვისა და ინფორმაციასთან უწყვეტი წვდომის უზრუნველსაყოფად, თუმცა მოცემული მიდგომა მაინც საკმაოდ „ხისტია“ და გააჩნია რიგი ნაკლოვანებებისა, რომელთაგან უპირველესად რესურსების უყაირათო ხარჯვა უნდა მივიჩნიოთ. სერვერული სისტემის პირველადი არქიტექტურა მეტნაკლებად ითვალისწინებს ყოველი სერვერის წინაშე მდგარ სარესურსო მოთხოვნებს, თუმცა ორგანიზაციის ინფორმაციული მოთხოვნების გაზრდის კვალდაკვალ სერვერული სისტემების მათზე მორგება საკმაოდ შრომატევადი საქმეა და მნიშვნელოვან ფინანსურ და შრომით დანახარჯებათანაა დაკავშირებული.

ბოლო წლებში განვითარებული რამდენიმე ტექნოლოგიის (ვირტუალიზაცია, კლასტერული არქიტექტურა, ინფორმაციის მრავალდონური შენახვა) წყალობით აღნიშნულ ამოცანათა მეტი ეფექტურობით გადაჭრა გახდა შესაძლებელი. ქვემოთ მოცემულია თითოეული მათგანის მოკლე აღწერაა.

## 1.2 ვირტუალიზაცია

ვირტუალიზაციის თეორიული და პრაქტიკული ასპექტები ჯერ კიდევ გასული საუკუნის 70-იან წლებში იქნა დამუშავებული **IBM** ფირმის სუპერმანქანებისათვის. სისტემის ყოველ მომხმარებელს მთლიანი სერვერიდან გამოეყოფოდა რესურსების საკუთარი ნაკრები და უქმნიდა გამოყოფილ გამოთვლით სისტემასთან მუშაობის ილუზიას.

2000-იან წლებში იდეა ხელახლა გახდა აქტუალური, განსაკუთრებით სერვერულ სისტემებში, თუმცა ზოგადად ტერმინი „ვირტუალიზაცია“ ბევრად ფართო შინაარს ატარებს. ვირტუალურად შეიძლება ვაქციოთ გამოთვლითი სისტემის პრაქტიკულად ყველა კომპონენტი, როგორც აპარატული, ასევე პროგრამული (მაგალითად, ოპტიკური დისკამძრავები, ქსელური ინტერფეისები, ოპერატიული მეხსიერება, ოპერაციული სისტემები და ასე შემდეგ).

სერვერის ვირტუალიზაცია ერთ ფიზიკულ სერვერზე რამდენიმე ვირტუალური სერვერის გაშვებას გულისხმობს [12]. ვირტუალური მანქანები ან სერვერები წარმოადგენენ პროგრამებს, რომლებიც ეშვებიან ფიზიკური, ე.წ. "მასპინძელი" ოპერაციული სისტემის ფარგლებში (ჰოსტ-სერვერი). თავის მხრივ, ყოველი ვირტუალური მანქანა წარმოადგენს დამოუკიდებელ ოპერაციულ სისტემას, საკუთარი პროგრამებით და სერვისებით.

ვირტულიზაციის შედეგად მიღებული ეკონომიკური ეფექტი უდავოა. დიდი სერვერული სისტემა (მონაცემთა დამუშავების ცენტრი) ვრცელ ფართზეა განთავსებული და ელექტრო ენერგიის დიდ რაოდენობას მოიხმარს, განსაკუთრებით მაშინ, როცა ცენტრს გაციების სპეციალური სისტემები და დამატებითი ინფრასტრუქ-ტურა ემსახურება. ვირტუალიზაციის გამოყენება რამდენიმე ფიზიკური სერვერის ერთ მძლავრ სერვერზე „შეფუთვის“ საშუალებას იძლევა, რითაც იზოგება ადგილი და მცირდება

ელექტროენერგიის ხარჯი. ამასთან მცირდება აპარატურაზე საერთო დანახარჯებიც ნაკლები რაოდენობის სერვერთა არსებობის გამო.

შეგვიძლია ჩამოვთვალოთ ვირტუალური სისტემების სხვა უპირატესობებიც ფიზიკურ სისტემებთან შედარებით, განსაკუთრებით კომპიუტერული ქსელების სერვერული ინფრასტრუქტურის აგებისა და მართვის თვალსაზრისით:

- ერთ გამოთვლით სისტემაში ერთზე მეტი ოპერაციული სისტემის პარალელური მუშაობა;
- კრიტიკული რესურსების (პროცესორის დრო, ოპერატიული და გარე მეხსიერება) ოპტიმალური განაწილება ვირტუალურ მანქანებს შორის;
- ოპერაციული სისტემების სწრაფი გადატანა ფიზიკურ ჰოსტებს შორის. ოპერაციული სისტემების ინსტალაციისა და კონფიგურირების დროითი დანახარჯების შემცირება წინასწარ მომზადებული ვირტუალური სისტემების (იმიჯების) ხარჯზე.

გვერდს ვერ ავუვლით ძირითად ნაკლოვანებებსაც, რაც ვირტუალურ სისტემებს ახასიათებთ:

- სერვერულ რესურსებზე მაღალი მოთხოვნები - ვირტუალური მანქანა ფაქტობრივად დამოუკიდებელ ოპერაციულ სისტემას წარმოადგენს და გამოთვლითი რესურსებიც პრაქტიკულად ფიზიკური მანქანის მასშტაბებით სჭირდება, ანუ ბევრად მეტი, ვიდრე ამას სტანდარტული პროგრამები მოითხოვენ;
- დამოკიდებულება ერთ ფიზიკურ სერვერზე, რომლის მწყობრიდან გამოსვლაც მასზე არსებული ყველა ვირტუალური მანქანის მწყობრიდან გამოსვლის ტოლფასია.

ბოლო პრობლემის აღმოფხვრა ვირტუალიზაციისა და კლასტერული არქიტექტურის ურთიერთშერწყმით ხდება შესაძლებელი, რაც მოცემული ნაშრომის ერთერთ ძირითად საგანს წარმოადგენს.

ვირტუალიზაციის პროცესი საკმაოდ კომპლექსურია და რამდენიმე ამოცანის გადაჭრას ითხოვს:

**იზოლირება** ვირტუალიზაციის მნიშვნელოვანი კომპონენტია და ვირტუალური სისტემების სრულ ურთიერთდამოუკიდებლობაში მდგომარეობს. ერთი ვირტუალური მანქანის მწყობრიდან გამოსვლას მეორის მუშაობაზე არავითარი გავლენის მოხდენა არ შეუძლია. კერძოდ, ფიზიკურ სერვერზე განთავსებულ ვირტუალურ მანქანებს შორის მონაცემთა არანაირი ერთიანი სივრცე არ არსებობს. მათ შორის ინფორმაციის ნებისმიერი ტრანსფერი ჩვეულებრივი ქსელური ინტერფეისების გავლით ხორციელდება.

**ინკაპსულაცია** ნიშნავს მთლიანი ვირტუალური სისტემის ერთ (ან მეტ) ჩვეულებრივ კომპიუტერულ ფაილში ინტეგრირებას. ამგვარი სისტემები მეტად მოქნილია ფიზიკურ სისტემებს შორის გადატანის, კოპირებისა თუ არქივირების ოპერაციების შესრულებისას.

გამოთვლითი მანქანების ვირტუალიზაციისას განიხილავენ 2 მთავარ კომპონენტს: მასპინძელ (**Host**) და სტუმარ (**Guest**) სისტემებს, ამასთან მასპინძელი ერთია და როგორც წესი, ფიზიკურ გამოთვლით სისტემას წარმოადგენს, ხოლო სტუმარ-სისტემები მასზე განთავსებული 1 ან მეტი ვირტუალური მანქანებია.

ცხადია, ამგვარი მარტივი სტრუქტურა გამოსადეგია მხოლოდ დესკტოპ-სისტემებისთვის, სადაც მომხმარებელს რამდენიმე ოპერაციული სისტემის ერთდროული გამოყენება სჭირდება. იგი ასაგებად საკმაოდ ადვილია, თუმცა რამდენიმე ამოცანის გადაჭრას მაინც საჭიროებს. ეს ზოგადი ამოცანები ნებისმიერი ვირტუალური ინფრასტრუქტურის ფარგლებში წარმოიშობა და მათი განხილვა აუცილებელია:

- **სტუმარ-სისტემათა შაბლონების აგება** - არის პროცედურა, რომელსაც შეუძლია ვირტუალური სისტემის ასაგებად საჭირო დროითი დანახარჯების მნიშვნელოვნად შემცირება. ყოველი

სტუმარ-ოპერაციული სისტემისთვის (**Windows, LINUX, Mac OS X** და სხვა) იქმნება საბაზისო კონფიგურაცია, რომელიც შემდეგ ყოველი კონკრეტული მოთხოვნის მიხედვით ფართოვდება;

- **ვირტუალური სისტემების კოპირება ან კლონირება** – კოპირების ოპერაცია ვირტუალური სისტემის ჩვეულებრივი ასლის აგებას გულისხმობს და ნაკლებად ეფექტურია, რადგან მასპინძელ-სისტემის გარე მეხსიერების სწრაფ გავსებას იწვევს. ბევრად უფრო მოქნილი კლონირების მექანიზმი მომხმარებელს არჩევანს უტოვებს: სრული კლონირებისას ვირტუალური მანქანის სრული და დამოუკიდებელი ასლი იქმნება, ხოლო ბმული (ლინკირებული) კლონირება საწყის ვირტუალურ მანქანას ეყრდნობა (მის მოდიფიკაციას წარმოადგენს) და მეხსიერების მინიმალურ ხარჯვას სჯერდება;
- **სკრინშოტების შექმნა** – სკრინშოტი ვირტუალური სისტემის მიმდინარე მდგომარეობას ეწოდება და საჭიროების მიხედვით სისტემის მოცემული მდგომარეობის აღდგენას განაპირობებს;
- **ფიზიკური მანქანების ვირტუალიზაცია** – გულისხმობს არსებული ფიზიკური სისტემების ვირტუალურ ინფრასტრუქტურაში მიგრაციას ვირტუალური მანქანების სახით;
- **რესურსების განაწილება მასპინძელ და სტუმარ სისტემებს შორის** – გულისხმობს მასპინძელ-სისტემის რესურსების (პროცესორული სიმძლავრეები, ოპერატიული და გარე მეხსიერება, დისკამძრავები, ქსელური ინტერფეისები) სტუმარ-სისტემებზე გადანაწილებას. ეხება პირველ რიგში ქსელურ რესურსებს (სისტემის უნიკალური ქსელური **MAC**-მისამართი, **IP**-მისამართი).

ზემოთ აღწერილი ამოცანების პრაქტიკული რეალიზაციის პროცესი სადღეისოდ საკმაოდ შორს არის წასული. ვირტუალიზაციის პროგრამული პროდუქტები (ფირმა **VMWare**-ის **ESX Server, vSphere** და **Workstation**; ფირმა **Citrix**-ის **XenApp**, ფირმა **Microsoft**-ის **Hyper-V, Virtual Server, Virtual PC,** ფირმა **Oracle**-ის **VirtualBox)** მოიცავენ ინსტრუმენტების ფართო ნაკრებს სხვადასხვა

(მათ შორის კორპორაციული) მასშტაბის კომპიუტერული ქსელების სერვერულ სისტემათა ვირტუალიზაციისთვის. საყურადღებოა, რომ ამ ინსტრუმენტების ნაწილი არსებული ფიზიკური ინფრასტრუქტურის უმტკივნელოდ "გავირტუალების" საქმეს ემსახურება, რაც ორგანიზაციებს საშუალებას აძლევს თავიანთი სერვერული ინფრასტრუქტურის მოდერნიზაცია და ვირტუალურ რელსებზე გადაყვანა ძირითადი საწარმოო პროცესის შეწყვეტის გარეშე შეასრულონ.

საილუსტრაციოდ განვიხილოთ ვირტუალიზაციის ერთერთი სრული პროგრამული პაკეტი (ფირმა **VMWare**) და მისი მთავარი კომპონენტები:

- ჰაიპერვაიზორი **VMware ESX (ESXi) Server** - ჰოსტ-სისტემის მმართველი პროგრამა (იხილეთ ქვემოთ);
- **VMWare Converter** - ფიზიკური გამოთვლითი სისტემების ვირტუალიზაციის ან ვირტუალური მანქანების სხვადასხვა ფორმატებში კონვერტირების ინსტრუმენტი;
- **VMotion** – ვირტუალური მანქანების მიგრაცია სერვერებს შორის მათი გამორთვის გარეშე;
- **Virtual SMP** - სტუმარ-სისტემის 4-ზე მეტ პროცესორთან მუშაობის უზრუნველყოფა.

ბოლო წლებში ვირტუალიზაცია სერვერულ აპარატურასაც შეეხო. **x86**-არქიტექტურაზე მომუშავე პროცესორების ახალი თაობა (**Intel, AMD**) ვირტუალიზაციის აპარატურული მხარდაჭერითაა აღჭურვილი, რაც ერთიორად ამაღლებს ვირტუალური ოპერაციული სისტემების მუშაობის ეფექტურობას. სწორედ ამგვარი პროცესორებით აღჭურვილ სერვერულ სისტემებშია ყველაზე ეფექტური "აბსოლუტური ვირტუალიზაციის" განხორციელება, რომელიც ჰოსტ-მანქანად არა სტანდარტული ოპერაციული სისტემების, არამედ ე.წ. ჰაიპერვაიზორების გამოყენებას გულისხმობს. ჰაიპერვაიზორი სპეციალურად ვირტუალური

მანქანების მომსახურებისთვის შექმნილი მინი ოპერაციული სისტემაა. ვირტუალურ "მასპინძლად" სტანდარტული ოპერაციული სისტემების ჰაიპერვაიზორებით ჩანაცვლება მნიშვნელოვნად ზოგავს სერვერის რესურსებს. მიმდინარე ეტაპზე ყველაზე გავრცელებული ჰაიპერვაიზორებია **VMware ESX Server, Microsoft Hyper-V** და უფასო პროგრამა **Xen**.

## 1.3. ინფორმაციის საიმედო შენახვა სერვერულ სისტემებში

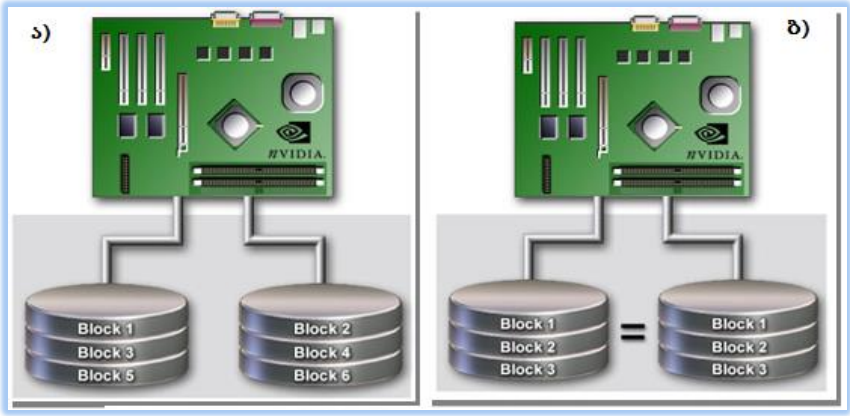
### 1.3.1. პროგრამულ-აპარატული უზრუნველყოფა

გარე მეხსიერების ის მოწყობილობები, რომლებიც დღევანდელ გამოთვლით სისტემებში გამოიყენება (ხისტი, ნახევარგამტარული, ოპტიკური), თავისთავად წარმოადგენენ ინფორმაციის საიმედო მატარებლებს ფუნქციონირების მრავალწლიანი გარანტიით, თუმცა ცხადია, მათი პირდაპირი გამოყენება განსაკუთრებით სერვერულ სისტემებში მიუღებელია. კრიტიკული ინფორმაციის (საბანკო და სადაზღვევო მონაცემთა ბაზები, სახელმწიფო და კორპორაციული ინფორმაცია) ერთ ეგზემპლარად შენახვა ინფორმაციის დაკარგვის თუნდაც მინიმალური რისკით გაუმართლებელია. სარეზერვო კოპირების და არქივაციის (იხ. ქვემოთ) ინსტრუმენტები შეიძლება ინფორმაციის კარგვისგან თავდაცვის ერთერთ მოხერხებულ მეთოდად მივიჩნიოთ, მათი გამოყენება მაინც არასაკმარისია ინფორმაციის საიმედო შენახვის პრაქტიკულად გარანტირებული უზრუნველყოფისთვის.

კომპიუტერული აპარატურის მწარმოებლები გვთავაზობენ სხვადასხვა საშუალებს ინფორმაციული სიჭარბის (**Redundancy**) მისაღწევად, რომელთაგან ყველაზე პოპულარულია გარე მეხსიერებაში განლაგებული ინფორმაციის დუბლირების მექანიზმები ეგრეთ წოდებული **RAID**-ტექნოლოგიის გამოყენებით, რომელსაც მოკლედ შევეხებით [3,5-7].

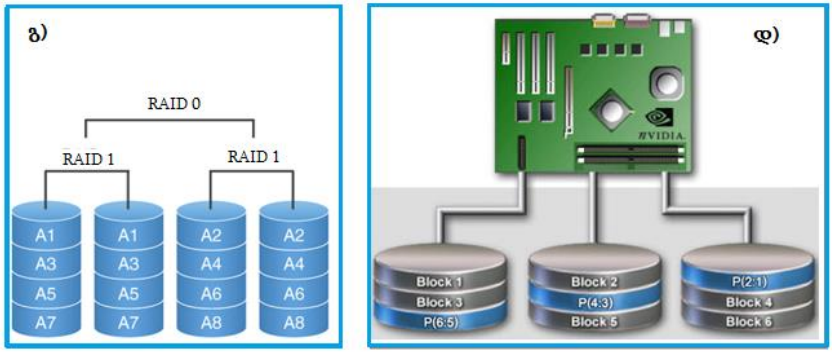
**RAID** გაიშიფრება როგორც Redundant Array of Independent Disks და დისკური მეხსიერებაში განთავსებულ ინფორმაციასთან მიმართვის დაჩქარების ან მისი შენახვის საიმედობის ამაღლების სხვადასხვა ალგორითმებს შეიცავს. სადღეისოდ არსებობს რამდენიმე **RAID**-არქიტექტურა **(RAID0…RAID6)** ან არქიტექტურათა კომბინაცია (მაგ. **RAID1+0**). ქართულ აიტი-სლენგზე ნახსენები ტერმინები შემდეგნაირად გამოითქმის: მე-10 მასივი - RAID10, მე-5 მასივი - RAID5 და ასე შემდეგ.

ყველაზე გავრცელებული RAID-სისტემები სქემატურად მოცემულია 1.2 ნახაზზე.




**ნახ.1.2. RAID-მასივთა ტიპები:**
**ა) RAID0 - სტრაიპი; ბ) RAID1 - სარკე;**

შევეხოთ მოკლედ ერთერთი, **RAID5**-არქიტექტურის მოქმედების პრინციპს, რომელიც 3-დან 32 ხისტ დისკთან მუშაობის საშუალებას იძლევა. არქიტექტურა ეფუძნება მარტივ ოპერატორს „გამომრიცხავი ან“ (**XOR**), რომელიც ბიტების მიმდევრობათა სიმრავლისგან ე.წ. „პარიტეტული ბლოკების“ მიღებისა და მათგან შემდგომ ბიტების რომელიმე საწყისი მიმდევრობის აღდგენის საშუალებას იძლევა.




ნახ.1.2. RAID-მასივთა ტიპები:
გ) RAID1+0 კომბინირებული სტრაიპი და სარკე;
დ) RAID5 - დისკური მასივი პარიტეტული ბლოკებით

ამასთან, სტანდარტული არითმეტიკული ოპერაციებისაგან განსხვავებით, აღნიშნული ოპერაცია არ იწვევს თანრიგების გადავსებას, რაც მისი გამოყენების ეკონომიურობას და მოხერხებულობას განაპირობებს. პარიტეტული ბლოკების ფორმირებისა და საწყისი ინფორმაციის აღდგენის ნიმუში იხილეთ 1.3 ნახაზზე.

| | Drive 1 | Drive 2 | Drive 3 | Drive 4 |
|---|---|---|---|---|
| **Stripe 1** | 0100 | 0101 | 0010 | 0011 |
| **Stripe 2** | 0010 | 0000 | 0110 | 0100 |
| **Stripe 3** | 0011 | 0001 | 1010 | 1000 |
| **Stripe 4** | 0110 | 0001 | 1101 | 1010 |

ნახ.1.3. პარიტეტული ბლოკები (ფერადი ფონით)

პარიტეტული ბლოკის ფორმირების მაგალითი (პირველი სტრაიპისთვის)

(0100) XOR (0101) XOR (0010) = (0011)

საწყისი ინფორმაციის აღდგენის მაგალითი (დაკარგულად ითვლება პირველი სტრაიპის პირველი ბლოკი)

(0101) XOR (0010) XOR (0011) = (0100)

**RAID5**-ის მთავარ ღირსებას თანაფარდობის „ინფორმაციის საიმედო შენახვა - სასარგებლო მეხსიერება“ მაღალი კოეფიციენტი წარმოადგენს. მაგალითად, 10-დისკიანი მასივისთვის სასარგებლო მეხსიერება მთლიანი მეხსიერების 90%-ს შეადგენს (ზოგიერთი წინაპირობის შესრულების შემთხვევაში), რაც დისკების „სარკული“ ასახვის 50%-თან შედარებით მაღალი მაჩვენებელია. თანამედროვე მონაცემთა საცავებში უპირატესობა სწორედ **RAID5**-ს ან მის მოდიფიცირებულ ვარიანტს, **RAID6**-ს ენიჭება.

### 1.3.2 ინფორმაციის მრავალდონიანი შენახვის მეთოდები

თანამედროვე კორპორაციულ ქსელებში გამოყენებული ინფორმაციის მოცულობა განუხრელად იზრდება. ინფორმაციის შენახვის ტექნოლოგიების მზარდი პროგრესის მიუხედავად, ახალი მეთოდების შემოღების აუცილებლობა დროის მცირე მონაკვეთებში ხდება საჭირო. სადღეისოდ კორპორაციული ინფორმაციის შენახვის ყველაზე პოპულარულ ტექნოლოგია შენახვის მრავალდონიანი არქიტექტურაა (**Multi Tier Storage**), რომელშიც გარკვეული წესების საფუძველზე თავმოყრილია სხვადასხვა მწარმოებლურობისა და ღირებულების მოწყობილობა ინფორმაციის შესანახად და მათი მმართველი ინტერფეისები. სადღეისოდ აღნიშნულ მოწყობილობათა შემდეგი კლასები შეიძლება გამოვყოთ (ნახ.1.4) [9]:

- **SSD (Solid State Drive)** - მყარსხეულიანი დისკური ტექნოლოგია, რომელიც ნახევარგამტარულ ტექნოლოგიას ეფუძნება და საყოველთაოდ ცნობილი ფლეშ-მეხსიერების სინონიმს წარმოადგენს. გამოირჩევა უაღრესად მაღალი მწარმოებლურობით და ასევე მაღალი ფასით;


ნახ.1.4. მონაცემთა მრავალდონიანი საცავის კომპონენტები:
ა) SATA და SAS-ინტერფეისები; ბ) მყარსხეულიანი დისკო (SSD);
გ) მაღალტევადი ოპტიკური დისკო: Blu Ray: დ) ლენტური
მეხსიერების კარადა

- **SAS/FC (Serial Attached SCSI/Fibre Channel)** - ხისტ დისკებზე აგებული ტექნოლოგია, რომელშიც ინფორმაცია კავშირის მაღალსიჩქარიანი არხით (მაგალითად, ოპტიკურბოჭკოვანი) გადაიცემა, ხოლო ინფორმაციის შენახვის ღირებულება საკმაოდ დაბალია;
- **SATA** - ხისტ დისკოებზე აგებული ტექნოლოგია, რომელშიც კომბინაცია „მწარმოებლურობა/ღირებულება" ყველაზე ოპტიმალურია;
- **LTO** - ლენტური ტექნოლოგია. ყველაზე იაფი და დაბალმწარმოებლური სისტემა ინფორმაციასთან მიმდევრობითი მიმართვით;
- **Blu Ray** - ინფორმაციის შენახვის ოპტიკური ტექნოლოგია დაბალი ღირებულებით და დამაკმაყოფილებელი მწარმოებლურობით.

კორპორაციულ ინფორმაციის სხვადასხვა ტიპის საცავებში განაწილებისას მთავარ ამოცანას მოთხოვნის მიხედვით მისი კლასიფიცირება წარმოადგენს. საჭიროა შესრულდეს ინფორმაციის სასიცოცხლო ციკლის ანალიზი, რომლის დროსაც ირკვევა, რომ დროის კონკრეტულ მომენტში კონკრეტული ინფორმაცია ხშირად ან ნაკლები სიხშირით იცვლება, ხოლო რაღაც მომენტიდან იგი პრაქტიკულად აღარ არის საჭირო.

ანალიზის პროცესში გასათვალისწინებელია დაგროვილი ინფორმაციის მოცულობაც. სწორედ ეს პარამეტრი განაპირობებს მრავალდონიან ინფორმაციულ საცავებში მონაცემთა მიგრაციის საჭიროებას და სიხშირეს, რისთვისაც სპეციალური პროგრამული უზრუნველყოფა არსებობს. იგი ავტომატურ რეჟიმში ასრულებს ინფორმაციის გადატანას მრავალდონიანი არქიტექტურის ფარგლებში, ინფორმაციის სტატუსის გათვალისწინებით.

თანამედროვე კორპორაციულ ქსელებში ინფორმაციის შენახვის და მართვის ყველაზე ოპტიმალურ სტრუქტურად სამდონიანი არქიტექტურა (**3-Tier Architecture**) ითვლება, რომელშიც ყოველ დონეს საკუთარი აგებულება და ამოცანები გააჩნია (ნახ.1.5).


**ნახ.1.5. ინფორმაციის შენახვის სამდონიანი არქიტექტურის ზოგადი სქემა**

პირველ, ე.წ. მწარმოებლურობის დონეზე (**Performance Tier, Tier 1**) პირველადი, ყველაზე ხშირად გამოსაყენებელი ინფორმაცია ინახება. მოცემული დონის ფიზიკური შემადგენლობა წარმოადგენს სწრაფქმედი **SAS/FC**-დისკების მასივს ინფორმაციის საიმედო შენახვის **RAID**-ინტერფეისებით. ამავე დონეზე მოიაზრება **SSD**-

დისკების გარკვეული რაოდენობაც განსაკუთრებით აქტუალური ინფორმაციული მასივების შესანახად, თუმცა ჯერჯერობით ამგვარი დისკების ხვედრითი წილი კორპორაციულ ინფორმაციულ საცავებში ჯერ კიდევ დაბალია, რასაც მათი სიძვირე განაპირობებს.

მეორე დონე (**Capacity Tier, Tier 2**), რომელსაც "მოცულობითი დონე" ეწოდება, კორპორაციის ძირითადი, შედარებით ნაკლებაქტუალური ინფორმაციული მასივების შენახვას ემსახურება და როგორც წესი, **SATA/RAID**-ტექნოლოგიით ორგანიზებული გარე მეხსიერების მასივების ერთობლიობას წარმოადგენს.

მესამე, არქივაციის დონე (**Archive Tier, Tier 3**) გრძელვადიანი არქივების შესანახადაა განკუთვნილი და ლენტური და ოპტიკური შემნახველი მოწყობილობების საფუძველზეა აგებული. ამასთან, როგორც წესი, პრიორიტეტი ლენტური მეხსიერების აპარატურას ენიჭება, როგორც მეხსიერების დიდი მოცულობის, ასევე მეხსიერების მოწყობილობათა (მეხსიერების კასეტები) დიდი, ავტონომიური სისტემების აგების შესაძლებლობათა გამო. არქივაციის დონეზე ინფორმაციის ცვლილება ან წაშლა მეტწილად მხოლოდ წინაწარ განსაზღვრულ, გრძელვადიან პერსპექტივაში ხდება.

აღწერილ დონეებს შორის ინფორმაციის ავტომატური მიგრაცია, როგორც აღვნიშნეთ, მონაცემთა მართვის წინასწარ განსაზღვრული წესებით სრულდება, შესაბამისი პროგრამული უზრუნველყოფის გამოყენებით.

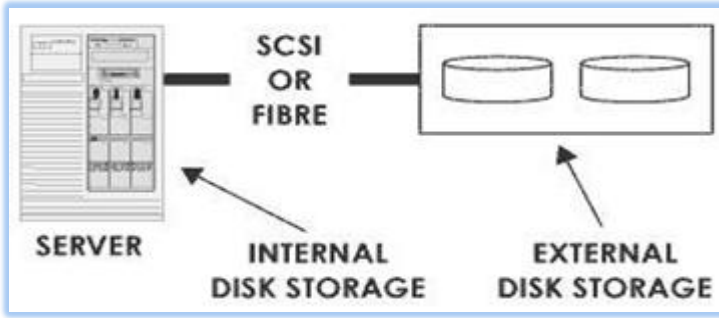
### 1.3.3 მონაცემთა საცავები

მონაცემთა შენახვის ზემოთ განხილული ფიზიკური ინფრა-სტრუქტურის დანიშნულებაა ინფორმაციის საიმედო და ოპტიმალური შენახვის უზრუნველყოფა და დროული მიწოდება მომხმარებლისთვის [14]. მეორე მოთხოვნა პირველზე არანაკლებ მნიშვნელოვანია, რადგან ინფორმაციის დაგვიანებით მიღება ხშირ შემთვევაში მისი არმიღების ტოლფასია. სადღეისოდ არსებობს

ინფორმაციის შენახვის ისეთი სისტემები, რომლებიც მომხმარებელს მონაცემებთან მაქსიმალურად ეფექტური მიმართვის და მათი საიმედო შენახვის საშუალებებს სთავაზობს.

განვიხილოთ რამდენიმე ყველაზე გავრცელებული სისტემა, კერძოდ, **DAS, NAS** და **SAN**-სისტემები.

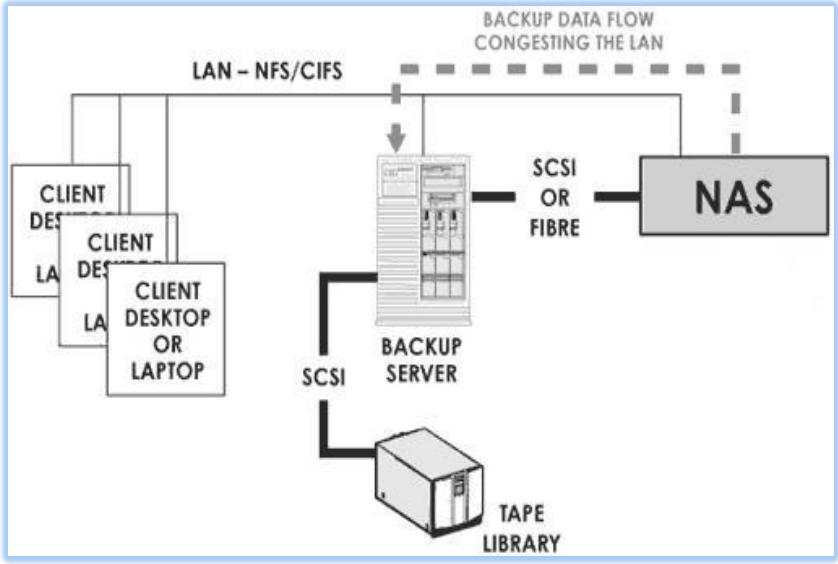
- **DAS (Direct Attached Storage)** - მოწყობილება შიგა მეხსიერებით, პირდაპირ უერთდება ძირითად კომპიუტერს და ძირითადად ლოკალური სარეზერვო კოპირების ამოცანებს ემსახურება. მარტივად რომ ვთქვათ, **DAS** არის ჩვეულებრივი ხისტი დისკო, რომელიც ჰოსტთან დასაკავშირებლად ფართოდ გავრცელებულ **SCSI**-ტექნოლოგიას **(Small Computer System Interface)** იყენებს (ნახ.1.6).




ნახ.1.6. DAS-სისტემის სქემა

**DAS**-სისტემა ვერ უზრუნველყოფს მონაცემთა სარეზერვო კოპირებას რამდენიმე ჰოსტიდან, მითუმეტეს მონაცემთა განაწილებას. ამგვარი მოწყობილობის დაყენება სარეზერვო კოპირების იაფფასიანი ვარიანტია, თუმცა გამოუსადეგარი, მეტნაკლებად დიდი ორგანიზაციის კომპიუტერულ ქსელში.

- **NAS (Network Attached Storage)** - "მონაცემთა ქსელური საცავი" კომპიუტერული ქსელის სრულუფლებიანი წევრია შესაბამისი საიდენტიფიკაციო მონაცემებით (**MAC** და **IP**-მისამართები) (ნახ.1.7).

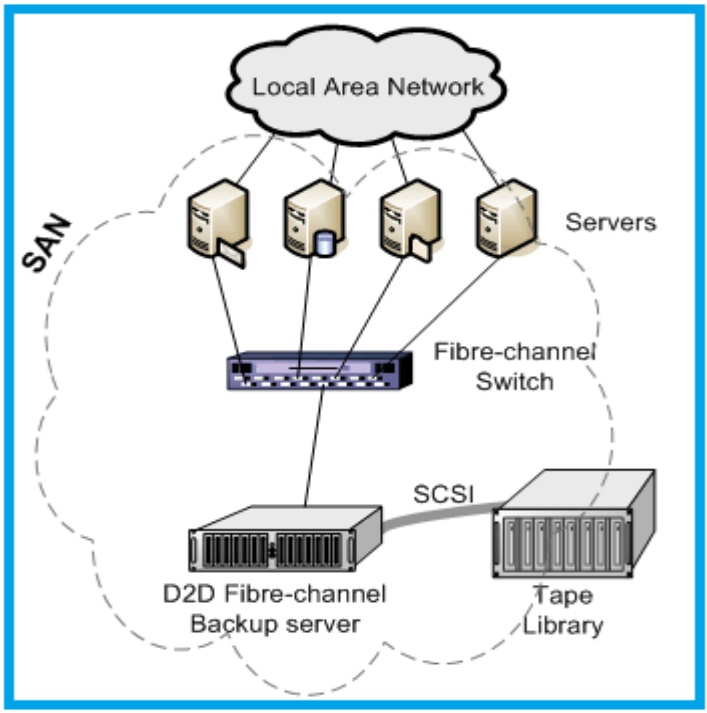



ნახ.1.7. NAS-სისტემის სქემა

**NAS**-სერვერის ძირითადი ფიზიკური კომპონენტი, როგორც წესი, 1 ან მეტი მეხსიერების ხისტი დისკოა, თუმცა ხანდახან მის კონფიგურაციაში ოპტიკული ან ლენტური მოწყობილობებიცაა გათვალისწინებული. **NAS**-მოწყობილობა **(NAS Appliance)** მარტივად ირთვება ქსელში და არის ფაილ-სერვერი საკუთარი მცირე ოპერაციული სისტემითა და მართვის ვებ-ბაზირებული ინტერ-ფეისით. **NAS**-სერვერები გარეგნულად **DAS**-მოწყობილობებს ჰგავს, მაგრამ პრინციპული განსხვავება მისგან ფაილებთან ქსელური წვდომის ეფექტური საშუალებებია.

- **SAN (Storage Area Network)** – „მონაცემთა საცავების ქსელი" მაღალი საიმედოობისა და სისწრაფის გამო სადღეისოდ მონაცემთა საიმედო შენახვის ყველაზე გავრცელებული ტექნოლოგიაა და მასობრივად ინერგება კორპორაციულ ქსელებში. იგი ეფუძნება ძალიან სწრაფი ოპტიკურ-ბოჭკოვანი კავშირის **Fibre Channel**-

ტექნოლოგიას, რომელშიც ინფორმაციის გადაცემის სიჩქარეა 1-2 გბ/წმ. ამასთან, მონაცემთა საცავის ქსელი ფიზიკურად გამოცალკევებულია ძირითადი კორპორაციული ქსელიდან, რაც ძლიერ ზრდის სარეზერვო ასლების შექმნისა და არქივაციის პროცედურების ეფექტურობას და ინფორმაციის შენახვის საიმედოობას (ნახ.1.8).




ნახ. 8 SAN-სისტემის სქემა

ამასთან, მნიშვნელოვანია ძირითადი ქსელური არხების განტვირთვის ფაქტორიც, რადგან სწორედ აღნიშნული პროცედურები (სარეზერვო კოპირება, არქივაცია) მოითხოვს ყველაზე დიდი ოდენობით ქსელურ რესურსებს და ინფორმაციული არხების მაღალ გამტარუნარიანობას.

სადღისოდ სერვერული სისტემების ბაზარზე **SAN**-საცავების საკმაოდ ბევრი ვარიანტი არსებობს, რაც დამკვეთი კომპანიების ერთმანეთისგან რადიკალური მოთხოვნების დაკმაყოფილებას უზრუნველყოფს.

მაგალითად, **Fibre Channel** ტექნოლოგიის გამოყენება შეიძლება 100 კილომეტრის რადიუსში (შედარებისთვის: ჩვეულებრივი **SCSI**-ინტერფეისების მოქმედების რადიუსი მხოლოდ 25 მეტრს შეადგენს).

**SAN**-ქსელის ცენტრალური კომპონენტია ოპტიკური არხის ადაპტერი (**Fibre Channel host bus adapter — FC HBA**), რომელიც სერვერებსა და მონაცემთა საცავებს შორის კავშირს უზრუნველყოფს. კერძოდ, იგულისხმება შემდეგი ფუნქციონალობა:

- ინტერფეისი სერვერებსა და მონაცემთა საცავებს შორის ნებისმიერი ქსელური ტოპოლოგიის ფარგლებში;
- მონაცემთა საცავების ცენტრალიზებული მართვა (კონფიგურება, მონიტორინგი და ქსელის კომპონენტების ანალიზი;
- დისკურ მასივებთან წვდომის უფლებების მართვა.

### 1.3.4 მონაცემთა დამუშავების ცენტრი

მონაცემთა საცავების ყველაზე მასშტაბური მოდიფიკაცია **მონაცემთა დამუშავების ცენტრია** (**Data Center**), რომელიც დიდი მოცულობის და მრავალფეროვანი ინფორმაციის (ტექსტური, მონაცემთა ბაზები, მულტიმედია) ეფექტურ და საიმედო შენახვას უზრუნველყოფს [14,17]. მონაცემთა დამუშავების ცენტრი შედარებით ნაკლებადაა გავრცელებული მისი მაღალი ღირებულების გამო. საკუთარი მონაცემთა ცენტრები გააჩნია მეტწილად დიდ კორპორაციებს და ინტერნეტ-სერვის-პროვაიდერებს, რომლებიც მომხმარებელს კარგად განვითარებულ ჰოსტინგის სერვისს სთავაზობს.

### 1.3.5. სარეზერვო კოპირება და არქივაცია

კორპორაციული ქსელის სერვერული სისტემა წარმოუდგენელია კარგად გამართული სარეზერვო კოპირებისა და არქივაციის სისტემის გარეშე. ინფორმაციის სარეზერვო საცავები გვაძლევს გარანტიას, რომ კორპორაციული ინფორმაცია არ დაიკარგება უკიდურეს შემთხვევაშიც კი, ხანძრის ან სტიქიური უბედურების შედეგად.

სარეზერვო კოპირების მრავალფეროვანი სისტემები არსებობს, რომელთაგან თითოეულს საკუთარი გამოყენების სფერო გააჩნია კომპიუტერული ქსელის მასშტაბებისა და შესასრულებელ ამოცანათა მოცულობის მიხედვით .

## 1.4 კლასტერული არქიტექტურა

კლასტერული არქიტექტურა სერვერულ სისტემებში (განსაკუთრებით მონაცემთა ბაზის სერვერებზე) ერთ-ერთ ყველაზე აპრობირებული ტექნოლოგიაა [17].

**კლასტერი** არის კომპიუტერების ჯგუფი, რომელიც შედგება ორი ან მეტი მანქანისაგან. ისინი ერთად მუშაობს იმისთვის, რომ უზრუნველყოს პროგრამების საერთო ნაკრების ან სერვისების ფუნქციონირება, რომლებსაც მომხმარებელი აღიქვამს როგორც ერთ მთლიანს. კომპიუტერები ფიზიკურად ერთიანდება აპარატურული საშუალებით ქსელის სახით ან საერთო შესანახი მოწყობილობებით.

კლასტერების პროგრამული სისტემა უზრუნველყოფს საერთო ინტერფეისს, ამასთან ერთად მართავს რესურსებს და კლასტერის შიგნით მყოფი კომპიუტერების დატვირთვას.

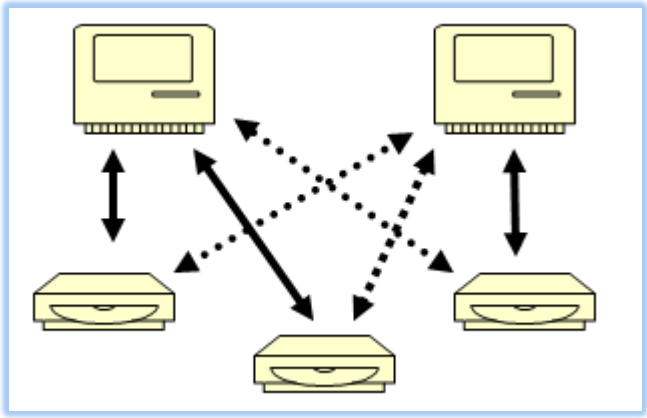
კლასტერი შედგება შემდეგი ძირითადი ელემენტებისაგან:

- **კვანძი** (როგორც წესი, ფიზიკური ან ვირტუალური სერვერი);
- **რესურსთა ჯგუფი:**
  - *ქვორუმ-რესურსი* (ხისტი დისკო კლასტერის კონფიგურაციის მონაცემთა ბაზით);

- *რესურსები* (ფიზიკური დისკო ლოკალური ან გარე მეხსიერების მასივიდან, IP-მისამართი, კომპიუტერის ქსელური სახელი (ვირტუალური სერვერი), სისტემური სერვისი, განაწილებული ფოლდერი, განაწილებული პროგრამა, ვირტუალური მანქანა.

კლასტერის ელემენტებს შორის დამოკიდებულებებს შემდეგი სახე აქვს: **რესურსი -> რესურსი** (მაგალიტად, **განაწილებული ფოლდერი -> ხისტი დისკო**). მოქმედებს იერარქიულობის პრინციპი.

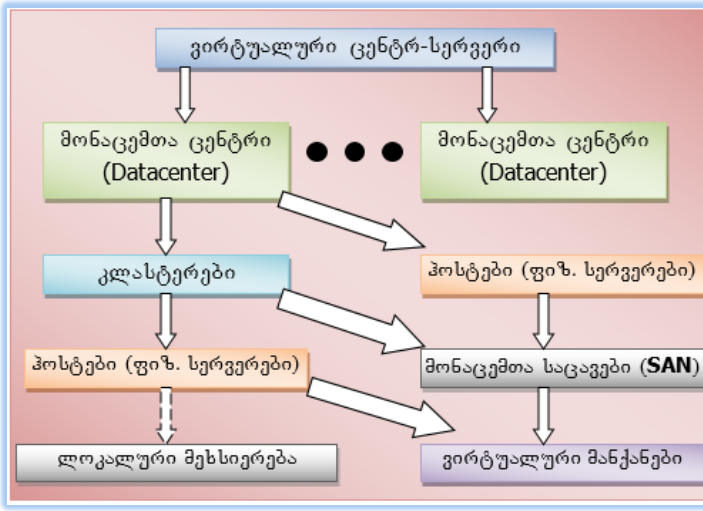
კლასტერის მუშაობის პრინციპი: კლასტერის რომელიმე კვანძის მწყობრიდან გამოსვლისას მასზე განთავსებული რესურსები ავტომატურად გადამისამართდება აქტიურ კვანძზე (**Failover**). სისტემა აგრძელებს მუშაობას (ნახ.1.9).



ნახ.1.9. კლასტერის მუშაობის პრინციპი

## 1.5. სერვერული სისტემის არქიტექტურა

ზემოთ აღწერილი ტექნოლოგიების კომბინირებული გამოყენება რთული სერვერული სისტემების დაპროექტებისა და აგების საშუალებას იძლევა. სერვერული სისტემის ზოგადი სტრუქტურა 1.10 ნახაზზეა მოცემული.




ნახ.1.1.10. ვირტუალური სერვერული სისტემის არქიტექტურა

როგორც ნახაზიდან ჩანს, ინტეგრირებული სერვერული არქიტექტურა შეიძლება გასცდეს კიდეც კორპორაციული ქსელის ფარგლებს და მონაცემთა ცენტრის სახით რამდენიმე ქსელის ინფორმაციული სივრცე მოიცვას, თუმცა უნდა აღინიშნოს, რომ პრაქტიკაში ამგვარი შემთხვევები შედარებით ნაკლებია.

# თავი 2. კორპორაციული სერვერული სისტემების აგება

## 2.1 ქსელური ინფრასტრუქტურა

წიგნის ავტორებმა, წინა თავში აღწერილი მეთოდების გამოყენებით, მონაწილეობა მიიღეს ერთ-ერთი მსხვილი ორგანიზაციის კორპორაციული ქსელის მოდერნიზაციის პროცესში. ორგანიზაციას გააჩნია კორპორაციული ქსელი, რომელიც რამდენიმე ფიზიკური ქვექსელისაგან შედგება

ექსპერიმენტის ფარგლებში გამოყენებული სერვერული ინფრასტრუქტურის აღწერა მოცემულია 2.1 ცხრილში.

სერვერული ინფრასტრუქტურა ცხრ.2.1

| მოწყობილობა | ტიპი | პროც. რაოდ. | ოპერატ. მეხს. | გარე მეხსიერება |
|---|---|---|---|---|
| **IBM System X3850 X5** | სერვერი | 4x8 Core | 64GB (upto 2TB) | 2x300GB (SAS RAID1) |
| **IBM System X3650 M2** | ბეკაპ-სერვერი | 1x8 Core | 16GB | 2x300GB (SAS RAID1) |
| **2xDell PowerEdge R410** | სერვერი | 1x4 Core | 16GB | 2x2TB (SAS) |
| **2xDell PowerEdge R410** | სერვერი | 1x4 Core | 16GB | 2x600GB (SAS) |
| **IBM System Storage DS4700** | მონაცემთა საცავი | - | - | 14x1TB (14TB SATA RAID5) |
| **Seagate BlackArmor 110** | NAS-სერვერი | - | - | 1TB |
| **IBM System Storage TS2900** | ლენტური მეხსიერება | - | - | 9x800GB (7,2TB Data Cartriges) |

გამოყენებული სერვერული პროგრამული უზრუნველყოფა წარმოდგენილია 2.2 ცხრილში.

სერვერული პროგრამული უზრუნველყოფა ცხრ.2.1

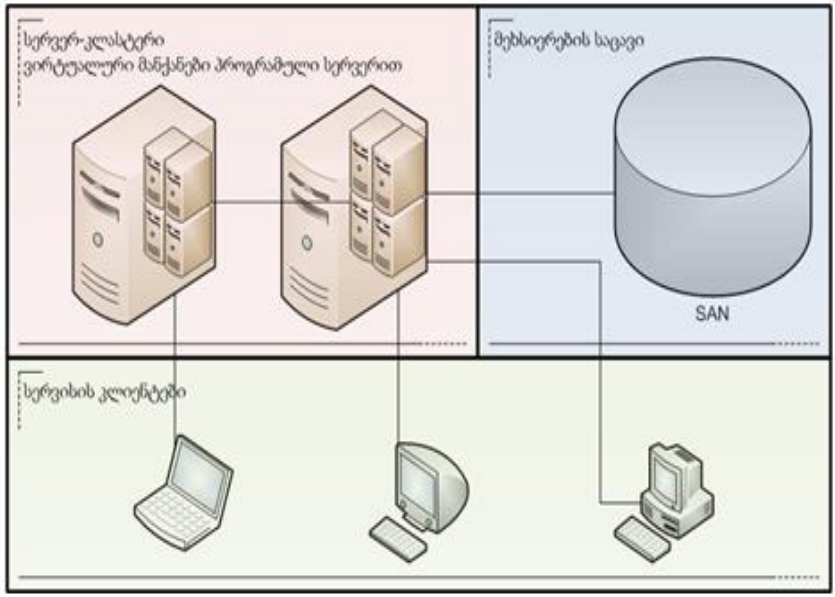
| პროგრამა | დანიშნულება |
|---|---|
| VMWare ESX | ჰაიპერვაიზორი |
| VMWare vSphere | ვირტუალური ინფრასტრუქტურა |
| IBM Fibre Storage Manager | მონაცემთა საცავების მართვა |
| IBM Tivoly Storage Manager | სარეზერვო ასლების ფორმირება, არქივაცია |

პროგრამული უზრუნველყოფის არჩევისას ძირითად კრიტერიუმებად მათი ფუნქციური შესაძლებლობების სიმრავლე იქნა მიჩნეული. **VMWare** ფირმის პროდუქტები სადღეისოდ ვირტუალიზაციის ბაზრის უპირობო ლიდერად ითვლება, ისევე როგორც ფირმა **IBM** - სერვერული სისტემების და ინფორმაციის უსაფრთხო და საიმედო შენახვის მოწყობილობათა ბაზარზე [5].

## 2.2 რესურსთა განაწილების მოდელები

სერვერული ინფრასტრუქტურის მოდელის ასაგებად გამოყენებულ იქნა ორი ტექნოლოგია: **ვირტუალიზაცია** და **კლასტერიზაცია.** პირველმა უზრუნველყო სერვისი „მრავალი ერთზე“, ხოლო მეორემ „ერთი მრავალზე“. ამ ორი სერვისის კომბინირებით სერვერული სისტემის მოდელის ოპტიმალური საწყისი სტრუქტურა ჩამოყალიბდა (ნახ.1.11).

მოცემული სტრუქტურა, უფრო ზუსტად კი მისი სერვერული ნაწილი განვაზოგადოთ პროგრამული პაკეტის **VMWare vSphere** მე-5 ვერსიის ბაზაზე, რომლის ერთერთი კომპონენტი, უტილიტა **vCenter** გრაფიკული ინტერფეისის მეშვეობით სერვერული სისტემის ყველა კომპონენტის მართვის საშუალებებს გვთავაზობს [5].

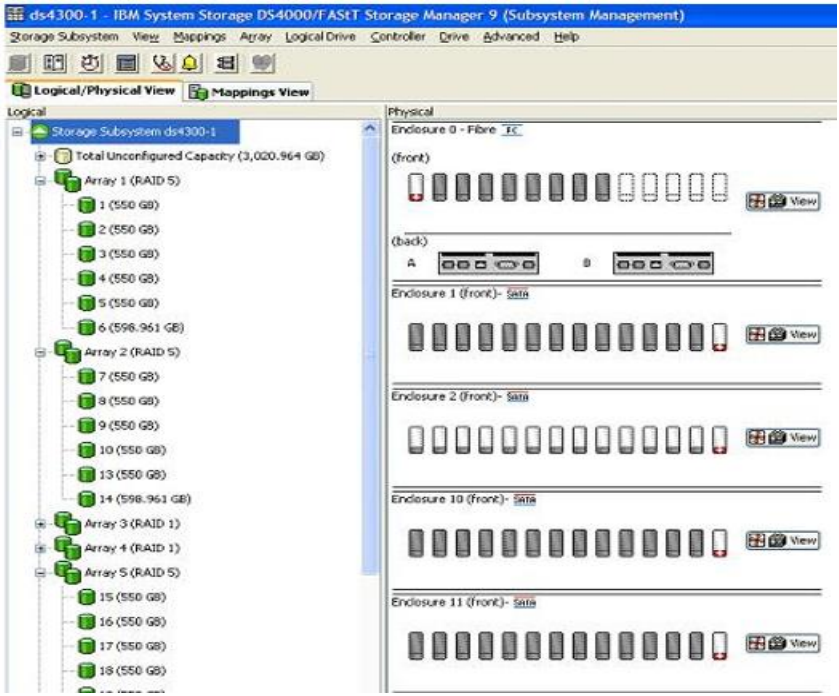



ნახ.1.11. კორპორაციული ქსელის ნიმუში ვირტუალიზაციისა და კლასტერიზაციის გამოყენებით

ამასთანავე უნდა შევნიშნოთ, რომ აღწერილი სისტემა ფიზიკურად ერთ ადგილასაა განთავსებული, რაც მონაცემთა დაკარგვის გარკვეულ რისკთანაა დაკავშირებული ხანძრის ან სხვა არასტანდარტული სიტუაციების შემთხვევაში, ამიტომ სერვერული სისტემის მოდელი გაფართოებულ იქნა მონაცემთა დამატებითი საცავით (**NAS**-სერვერი), რომელიც ძირითადი საცავიდან ფიზიკურად დაშორებულ ადგილასაა განთავსებული და განსაკუთრებულად კრიტიკული ინფორმაციის (მონაცემთა ბაზები) სარეზერვო ასლების შესანახად გამოიყენება. მონაცემთა საცავების დეტალური აღწერა იხილეთ მომდევნო პარაგრაფში.

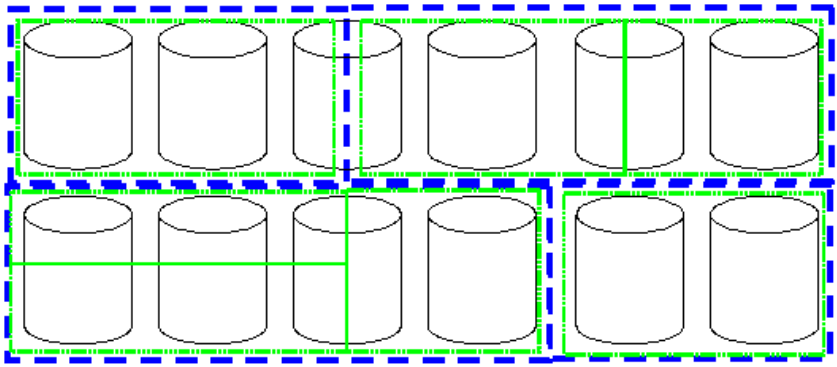
## 2.3 მონაცემთა საცავების მართვა

მონაცემთა საცავების მართვა განცალკევებული პროცესია და მონაცემთა საცავების საკუთარი შიგა სტრუქტურის აგებას გულისხმობს, რომელიც შემდგომ სხვადასხვა სერვისების მიერ სხვადასხვა მოცულობით და მონაცემთა საცავების სხვადასხვა ქსელის ფარგლებში **(Fibre Channel, iSCSI)** შეიძლება იქნეს გამოყენებული. შიგა სტრუქტურა უნდა აკმაყოფილებდეს საიმედოობისა და მონაცემებთან მაქსიმალურად სწრაფად მიმართვის პირობებს. საამისოდ, მონაცემთა საცავების მართვის პროგრამებს ინსტრუმენტების ფართო ნაკრები გააჩნია. განვიხილოთ ისინი პროგრამა **IBM Fibre Storage Manager v. 10**-ის მაგალითზე (ნახ.1.12).




ნახ.1.12. IBM Fibre Storage Manager - ინტერფეისის ფრაგმენტი

დისკური მასივის მართვის ალგორითმი შემდეგია: მთელი დისკური მასივი (ან მისი რაღაც ნაწილი) რომელიმე ტექნოლოგიის (მაგალითად, **RAID5**) მეშვეობით ერთ მთლიან ლოგიკურ დისკად ყალიბდება, ხოლო შემდეგ მმართველი პროგრამით შესაძლებელია მისი „დაშლა“ დისკური მასივის ლოგიკურ ტომებად (**LUN – Logical Unit Number)**, რაც 1.13 ნახაზზეა ასახული.



**ნახ.1.13. დისკური მასივის მართვის სქემა**

სწორედ ლოგიკური ტომი წარმოადგენს დისკური მეხსიერების იმ ერთეულს, რომელსაც ფიზიკური თუ ვირტუალური სერვერების ოპერაციული სისტემები ღებულობს და თავიანთი დისკური სივრცის ნაწილად (ლოკალურ დისკოებად) აღიქვამენ. ამგვარად, სერვერს „არ აინტერესებს“, მის განკარგულებაში გადმოსული ლოგიკური დისკო რეალურად რამდენ ფიზიკურ დისკოზეა განთავსებული. იგი მას საკუთარი ფიზიკური დისკივით იყენებს.

## 2.4. სარეზერვო კოპირების და არქივაციის მოდელები

**თეორია**. სარეზერვო ასლებისა (**BackUp**) და არქივების (**Archive**) შექმნა ინფორმაციის საიმედო შენახვას ემსახურება. ამ დროს საწყისი ინფორმაცია კოპირდება ბეკაპ-სერვერზე ან ლენტების არქივში. პირველი გზა სასარგებლოა ხშირად განახლებად მონაცემთათვის, როცა საწყისი მონაცემები არაერთხელ იცვლება და ინფორმაციის სწრაფ აღდგენაა საჭირო. მეორე დასრულებულ მონაცემთა (მაგალითად, დასრულებული პროექტების) შესანახადაა მიზანშეწონილი და მისი აღდგენაც დიდ დროს მოითხოვს.

ყველა ნორმალური ოპერაციული სისტემა შეიცავს მომხმარებელთა ცალკე ჯგუფს **BackUp Operators (Sicherungs-Operatoren)**, რომელშიც ადმინისტრატორთა პრივილეგიების არმქონე, მაგრამ სარეზერვო ასლების შესაქმნელად უფლებამოსილი მომხმარებლები შედიან.

### 2.4.1 ძირითადი სარეზერვო კოპირებისა და არქივაციის მოდელი

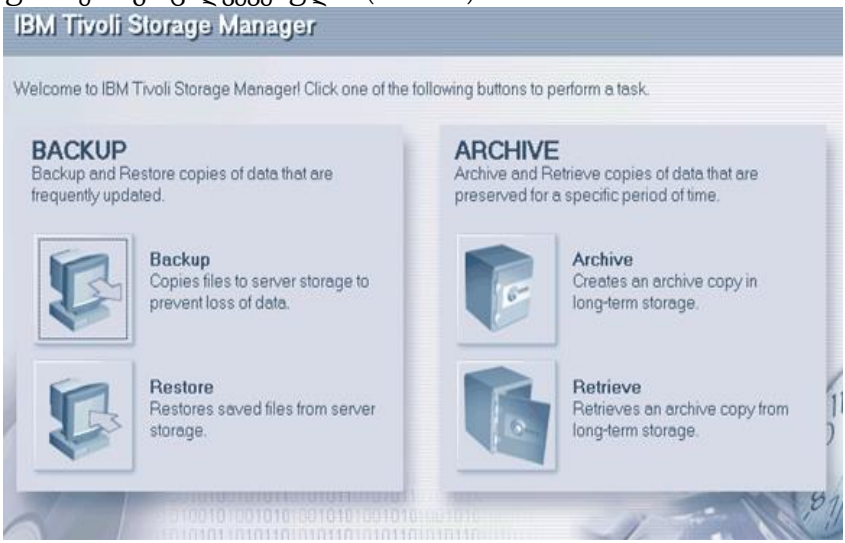
სარეზერვო კოპირებისა და არქივაციის მნიშვნელობა კორპორაციულ ქსელში უდავოა. ინფორმაციის სარეზერვო ასლების შექმნის გონივრულ პოლიტიკას შეუძლია კორპორაციული საქმიანობა თვით ყველაზე კრიზისული სიტაუაციებიდან იხსნას. შესაბამისად, ჩვენი კორპორაციული ქსელის სერვერის ინფრასტრუქტურის ჩამოყალიბებისას ამ საკითხზე განსაკუთრებული ყურადღება გავამახვილეთ.

რადგანაც სარეზერვო კოპირების ძირითადი მმართველი სერვერი ფირმა **IBM**-ის მიერ იქნა მოწოდებული და გამოყენებულ პროგრამულ უზრუნველყოფადაც ამავე ფირმის **Tivoly Storage Manager**, ტერმინოლოგიასაც ამავე ფირმის მიერ მიღებული

სტანდარტით ჩამოვაყალიბებთ, თუმცა იგი მხოლოდ დე-ფაქტო სტანდარტს წარმოადგენს.

განსხვავება სარეზერვო კოპირებასა და არქივაციას შორის ფირმა **IBM**-ის ტერმინოლოგიით ინფორმაციის შენახვისა და მასთან მიმართვის სიჩქარეში გამოისახება. სარეზერვო კოპირება ნიშნავს ინფორმაციის დუბლირებას მეხსიერების მატარებლებზე მიმართვის მცირე დროით (ხისტი და მყარსხეულიანი დისკოები), ხოლო არქივაცია გულისხმობს ინფორმაციის გადატანას ნელი მიმართვის მოწყობილობებზე, როგორიცაა, მაგალითად კასეტური მეხსიერება.

ნათქვამიდან გამომდინარე, სარეზერვო კოპირება მოიცავს შედარებით აქტუალურ ინფორმაციას, ხოლო არქივაცია - გრძელვადიან არქივებს, რომლებზეც მოთხოვნა კორპორაციულ ქსელში დაბალია. ორი პროცედურა პროგრამულ ინტერფეისშიც ურთიერთგანცალკევებულია (ნახ.1.14).




**ნახ.1.14. Tivoly Storage Manager-ის საწყისი ინტერფეისი სარეზერვო ასლებისა (Backup, Restore) და არქივაციის (Archive, Retrieve) ინტერფეისებით**

პროგრამა **Tivoly Storage Manager** არის სარეზერვო კოპირებისა და არქივაციის სპეციალიზებული სერვერი ტრადიციულად **AIX**-ოპერაციული სისტემის ბაზაზე და **TSM**-ის სერვერული მოდულით.

კლიენტ-პროგრამები **(TSM Client)** დამუშავებულია პრაქტიკულად ყველა ოპერაციული სისტემისათვის.

**Tivoly Storage Manager** იყენებს შემდეგ საბაზო ტერმინოლოგიას:

- კვანძი - გამოთვლითი სისტემა, რომელიც იყენებს სარეზერვო ასლების აგებისა და არქივაციის სერვისს;
- სარეზერვო ასლების ფორმირების მეთოდები:
  - **სრული** (ანუ **სრული ინკრემენტული**) - იგება სისტემის მთლიანი სარეზერვო ასლი;
  - **ინკრემენტული** - სარეზერვო ასლები აიგება მხოლოდ იმ რესურსებისათვის, რომელებიც შეიცვალა ბოლო სარეზერვო კოპირების შემდეგ;
  - **დიფერენციალური** - სარეზერვო ასლები აიგება მხოლოდ იმ რესურსებისათვის, რომელებიც შეიცვალა ბოლო **სრული** სარეზერვო კოპირების შემდეგ.
- სარეზერვო ასლის ტიპი:
  - **აქტიური** - რესურსი არსებობს ორიგინალის სახით;
  - **პასიური** - რესურსის ორიგინალი არ არსებობს.
- სარეზერვო ასლებში ფაილების სტატუსის განსაზღვრის ალგორითმი ინკრემენტული კოპირების შედეგად:
  - სარეზერვო სერვერზე არსებული ფაილები და კატალოგები ჩანაცვლდება განახლებული ასლებით და ინარჩუნებს სტატუსს **„აქტიური";**
  - ორიგინალში ახალშექმნილი ფაილებისა და კატალოგების ასლები გადაკოპირდება სარეზერვო სერვერზე და იღებენ სტატუსს **„აქტიური";**
  - ორიგინალში წაშლილი კატალოგები და ფაილები სარეზერვო სერვერზე ღებულობენ სტატუსს **„პასიური".**

➢ სარეზერვო ასლების და არქივების მართვის პოლიტიკა (განისაზღვრება არქივაციის სერვერის ადმინისტრატორის მიერ):

• **კოპირების სიხშირე (Copy Frequency)** - სარეზერვო ასლის აღების სიხშირე;

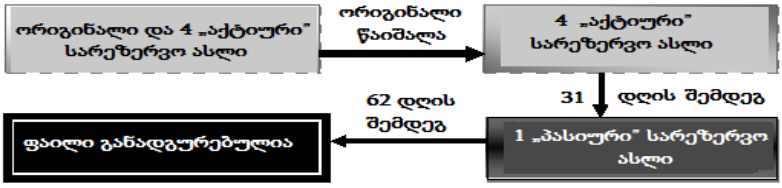
• **მონაცემთა ვერსიების რაოდენობა (Versions Data Exists)** – „აქტიური" სარეზერვო ასლების აღების „სიღრმე" (მაგალითად 4 ნიშნავს, რომ სერვერზე ინახება „აქტიური" რესურსების ბოლო 4 სარეზერვო ასლი);

• **პასიურ მონაცემთა ვერსიების რაოდენობა (Versions Data Deleted)** - სარეზერვო კოპირების სიღრმე „პასიური" რესურსებისათვის;

• **Retain Extra Versions** – „აქტიური" ასლების „სიცოცხლის ხანგრძლივობა" ორიგინალის წაშლის შემდეგ (მაგლითად, 31 დღე). ამ დროის შემდეგ ზედმეტი ასლები იშლება, ხოლო დანარჩენები ღებულობს „პასიური" ასლის სტატუსს;

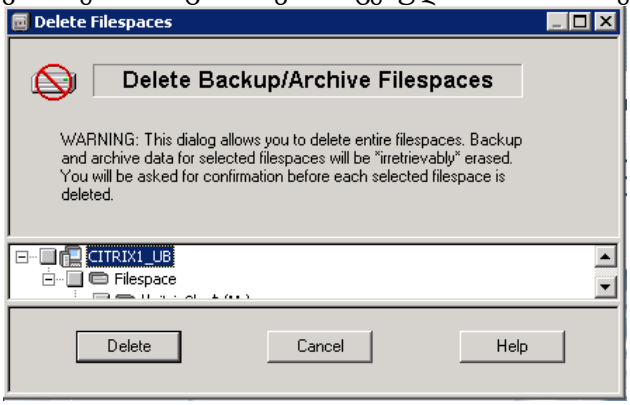
• **Retain only Version** – „პასიური" ასლების „სიცოცხლის ხანგრძლივობა" სარეზერვო ასლებისა და არქივებისათვის (მაგალითად, 62 და 1825 დღე).

მოცემული ტერმინოლოგიის გამოყენებით სარეზერვო კოპირების სერვისს შეუძლია შემოგვთავაზოს ინფორმაციის სასიცოცხლო ციკლის სქემა, რომლის ნიმუშიც მოცემულია 1.15 ნახაზზე.

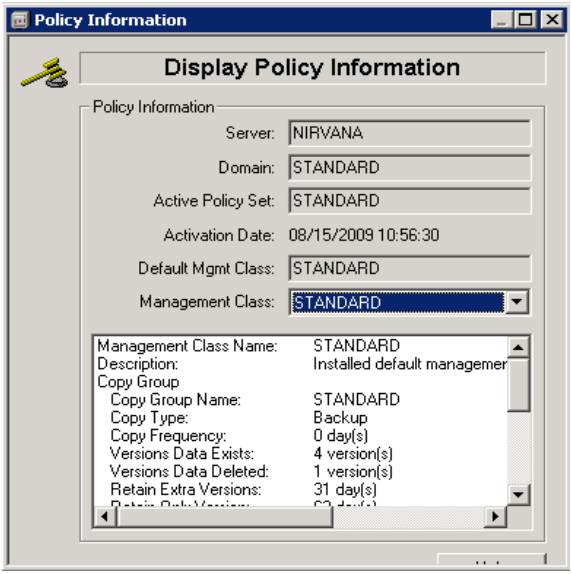



**ნახ.1.15. ინფორმაციის სასიცოცხლო ციკლის ნიმუში სარეზერვო კოპირების სისტემაში**

**Tivoly Storage Manager**-ის ინტერფეისის ფრაგმენტები ჩვენი კორპორაციული ქსელის სერვერული სისტემის სარეზერვო კოპირების ამოცანისთვის მოცემულია 1.16 ნახაზზე.




ა)




ბ)

ნახ.1.16. ა) სარეზერვო კოპირების ობიექტთა არჩევის დიალოგი; ბ) ინფორმაცია სარეზერვო კოპირების მიმდინარე პოლიტიკის შესახებ.

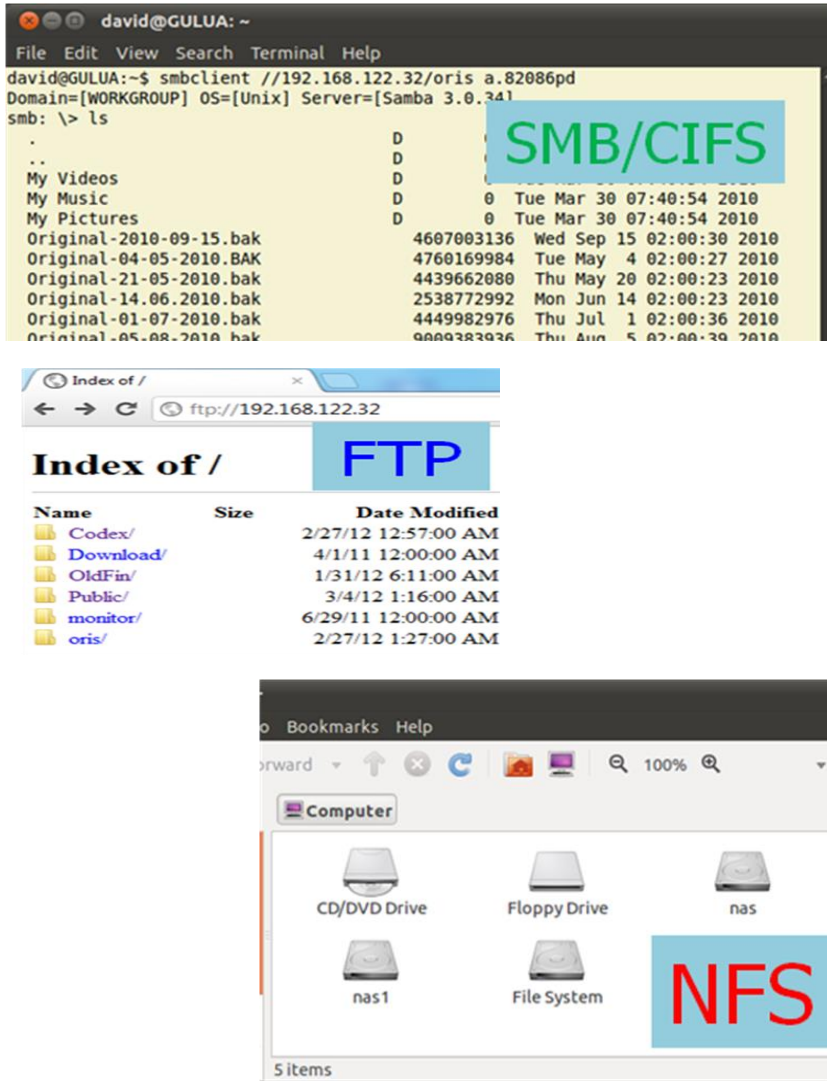
როგორც ზემოთ აღვნიშნეთ, არქივაციის პოლიტიკა ბევრად ხანგრძლივ დროის მონაკვეთზეა გათვლილი (ჩვენს სისტემაში პასიური ასლების სიცოცხლის ხანგრძლივობა მაგნიტურ კასეტებზე 1825 დღეს შეადგენს), თუმცა არქივაციის პოლიტიკის შემუშავების წესები სარეზერვო კოპირების პოლიტიკის ანალოგიური წესებისგან პრინციპულად არ განსხვავდება.

### 2.4.2 დამატებითი სარეზერვო კოპირების მოდელი

დამატებითი სარეზერვო კოპირების ამოცანას ემსახურება ფირმა **Seagate**-ის **NAS**-სერვერი **BlackArmor 110**, რომელიც 1 ტერაბაიტი ინფორმაციის შენახვის საშუალებას იძლევა. სერვერის სასარგებლო თვისებათაგან შეიძლება გამოვყოთ შემდეგი:

- ლოკალური ქსელის სრულუფლებიანი წევრი (საკუთარი დინამიკური ან სტატიკური, შიგა ან გარე **IP**-მისამართი);
- რესურსებისა და მომხმარებელთა მართვის საკუთარი პროგრამული უზრუნველყოფა (უმეტესად ვებ-ინტერფეისით);
- სარეზერვო კოპირების დამატებითი ფუნქციები:
  - მონაცემთა ან სისტემების აღდგენა ლოკალური ქსელიდან;
  - დისკური სივრცის დინამიკური გაფართოება გარე ხისტი დისკოებით, **OneTouch**-ტექნოლოგიის გამოყენებით;
- მომხმარებელთა სხვადასხვა სააღრიცხვო ჩანაწერებით სარეზერვო ასლებთან დიფერენცირებული წვდომის უფლებებით მიმართვის საშუალება;
- მრავალრიცხოვანი, ყველაზე გავრცელებული ფაილური სერვისების და ფაილური სისტემების **(CIFS, NFS, FTP)** მხარდაჭერა.

ჩვენი სისტემისთვის **NAS**-სერვერი გამოყენებულ იქნა განსაკუთრებით კრიტიკული ინფორმაციის (მონაცემთა ბაზები, ელექტრონული ფოსტა) სარეზერვო კოპირებისთვის (ნახ.1.17).




ნახ.1.17 NAS-სერვერზე განთავსებულ სარეზერვო ასლებთან წვდომის ინტერფეისები SMB/CIFS, FTP და NFS-სისტემების ფა\რგლებში

## 2.5 ვირტუალური ქსელების აგება

სერიოზული კორპორაციული ქსელის ვირტუალიზებული სერვერული არქიტექტურა, როგორც წესი, მცირე ოდენობის მძლავრ ფიზიკურ ჰოსტებს და მათზე განთავსებულ მრავალრიცხოვან ვირტუალურ მანქანებს შეიცავს. ფაქტობრივად, ქსელის ყოველი სერვისის განკარგულებაშია ერთი ან მეტი ვირტუალური მანქანა, რომლებიც საკმაოდ რთული სქემითაა ურთიერთდაკავშირებული. მაგალითად, კატალოგური სერვისი (**AD, NIS**) მოითხოვს რამდენიმე სერვერს საიტების შენახვისა და რეპლიკაციის ამოცანებისათვის. იგივე შეიძლება ითქვას ვებ-სერვისის და მონაცემთა ბაზების კლასტერიზაციისა და რეპლიკაციის ამოცანებზე.

ნათქვამიდან გამომდინარე, უშუალოდ ვირტუალური სერვერული სისტემის ფარგლებში სერვერთა ჯგუფების შექმნისა და მათი ურთიერთგამიჯვნის საჭიროება წარმოიშობა. მოცემული ამოცანისთვის, უპირველეს ყოვლისა, სტანდარტული ქსელური მეთოდები გამოიყენება.

ზოგადად, ვირტუალური მანქანა **ვირტუალური მოწყობილობის (Virtual Appliance)** ნაირსახეობაა, ხოლო ეს უკანასკნელი წინასწარ აგებულ და კონფიგურირებული, შესასრულებლად გამზადებული პროგრამაა, რომელიც თავის შესაფერის ოპერაციულ სისტემიანა შეიძლება ვირტუალურ ინფრასტრუქტურაში ჩაინერგოს. ვირტუალური მოწყობილობა შეიძლება იყოს თვით ოპერაციული სისტემა, თამაში და სხვა.

ვირტუალური მანქანების და ჰოსტების გარკვეული სიმრავლისგან მიიღება **ვირტუალური ქსელები (VM Network).**

## 2.6. VMWare vSphere-ის გრაფიკული ინსტრუმენტები

სქემები (Maps) ინფრასტრუქტურის პროგრამის გრაფიკული კომპონენტია, რომელიც სქემატურად გამოსახავს დამოკიდებულებებს ინფრასტრუქტურის კომპონენტებს შორის. დამოკიდებულებათა ტიპებია:

- **ჰოსტი -> ვირტუალური მანქანა** (ყოველი ვირტუალური მანქანა ფიზიკურად ერთ ჰოსტზეა განთავსებული, ყოველი ჰოსტი შეიცავს 0, 1 ან მეტ ვირტუალურ მანქანას;
- **ჰოსტი -> ვირტუალური ქსელი;**
- **ჰოსტი -> მონაცემთა საცავი** (ყოველი ჰოსტს გააჩნია მიმართვის უფლება მონაცემთა საცავის ყოველ ელემენტზე, თუ ფილტრი არ აყენია;
- **ვირტუალური მანქანა -> ვირტუალური ქსელი;**
- **ვირტუალური მანქანა -> მონაცემთა საცავი** (ყოველი ვირტუალური მანქანა განთავსებულია მონაცემთა საცავის 0 ან 1 ელემენტზე. პირველ შემთხვევაში დისლოკაციის ადგილია ჰოსტის ლოკალური მეხსიერება.

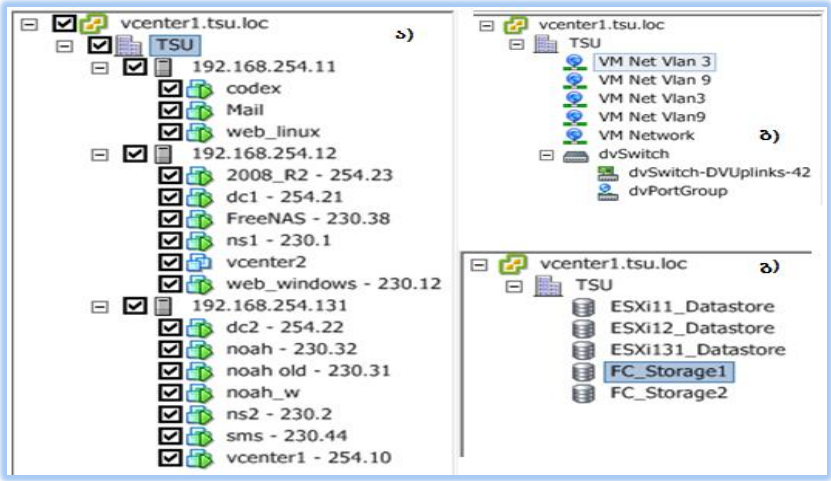
ინფრასტრუქტურის კიდევ ერთი ელემენტია რესურსთა ნაკრები (**Resource Pool**), რომელიც პროცესორის სიმძლავრეებს და ოპერატიული მეხსიერების ბლოკებს ვირტუალურ მანქანებზე ანაწილებს და ჰოსტებისგან დამოუკიდებელია.

## 2.7. სერვერული სისტემების ნიმუშები

ზემოაღწერილი ინსტრუმენტების საშუალებით ჩამოვაყალიბოთ წიგნის ავტორთა მონაწილეობით შექმნილი კორპორაციული ქსელის სერვერული სისტემის ერთიანი სტრუქტურა, რომელიც ქვემოთმოყვანილ იერარქიულ სქემაზე იქნება დაფუძნებული (ნახ.1.18).

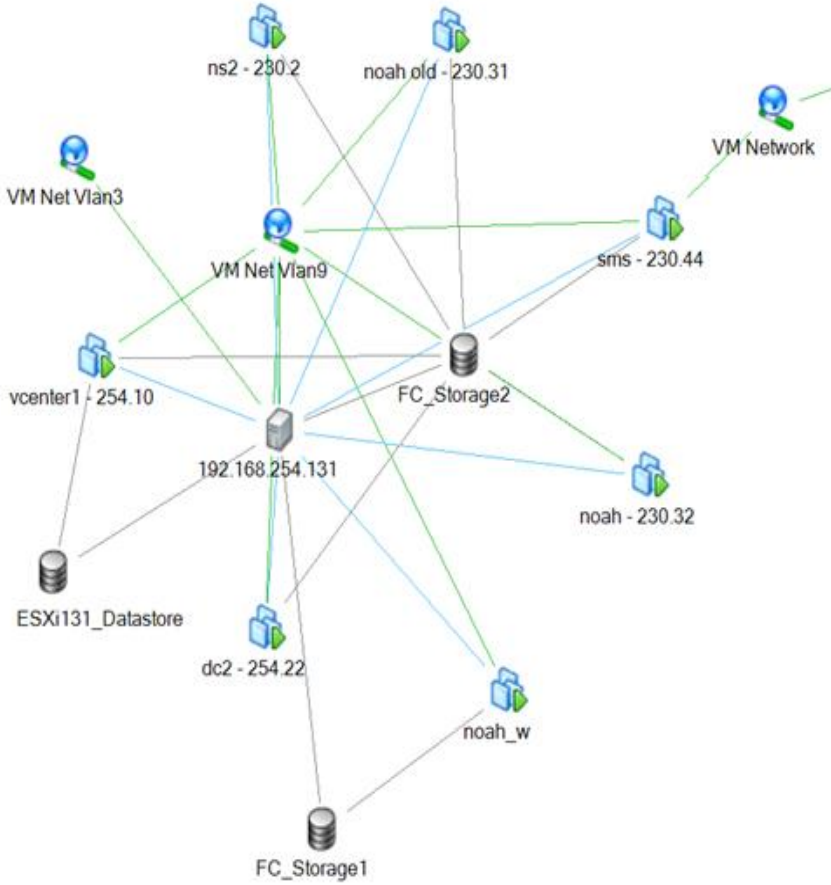

ნახ.18 კორპორაციული ქსელის ვირტუალური სერვერული სისტემის ზოგადი სქემა

1.18 ნახაზზე მოცემულია **vSphere**-სისტემაში შექმნილი ვირტუალური სერვერული სისტემის სტრუქტურა, აღწერილი იერარქიული სქემის საფუძველზე. კერძოდ, მოცემულია სერვერული სისტემის მონაცემთა საცავების, ვირტუალური ქსელების და ვირტუალური მანქანების ნაკრები.




ნახ.1.18 ა) ვირტუალური მანქანები; ბ) ვირტუალური ქსელები; გ) მონაცემთა საცავები

ახლა განვიხილოთ სერვერული სისტემის „რუქები“, რომლებიც ვირტუალური სერვერული სისტემის სხვადასხვა რაკურსიდან დანახვის საშუალებას გვაძლევენ. 1.19 ნახაზზე გამოსახული რუკა გვთავაზობს კორპორაციული ქსელის ფრაგმენტს ელემენტთა სრული ურთიერთკავშირებით.

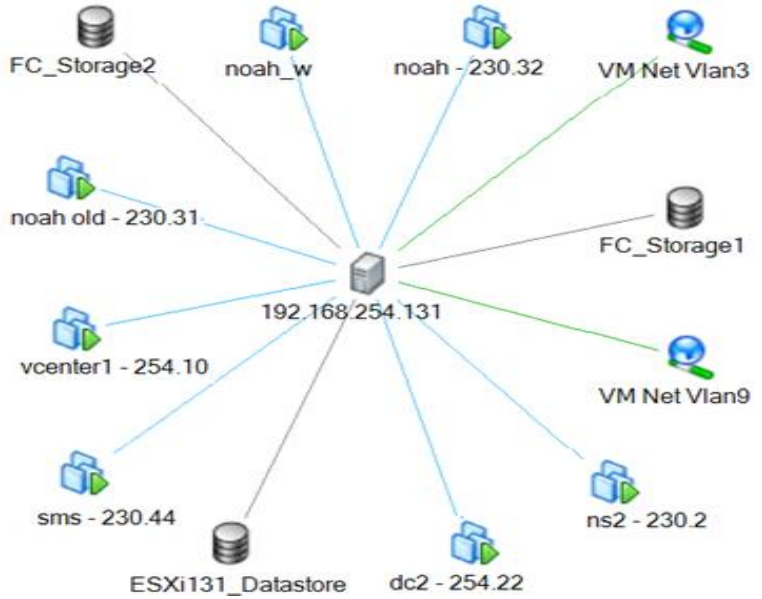



ნახ.1.19 კორპორაციული ქსელის ფრაგმენტი კომპონენტთა შორის სრული კავშირებით

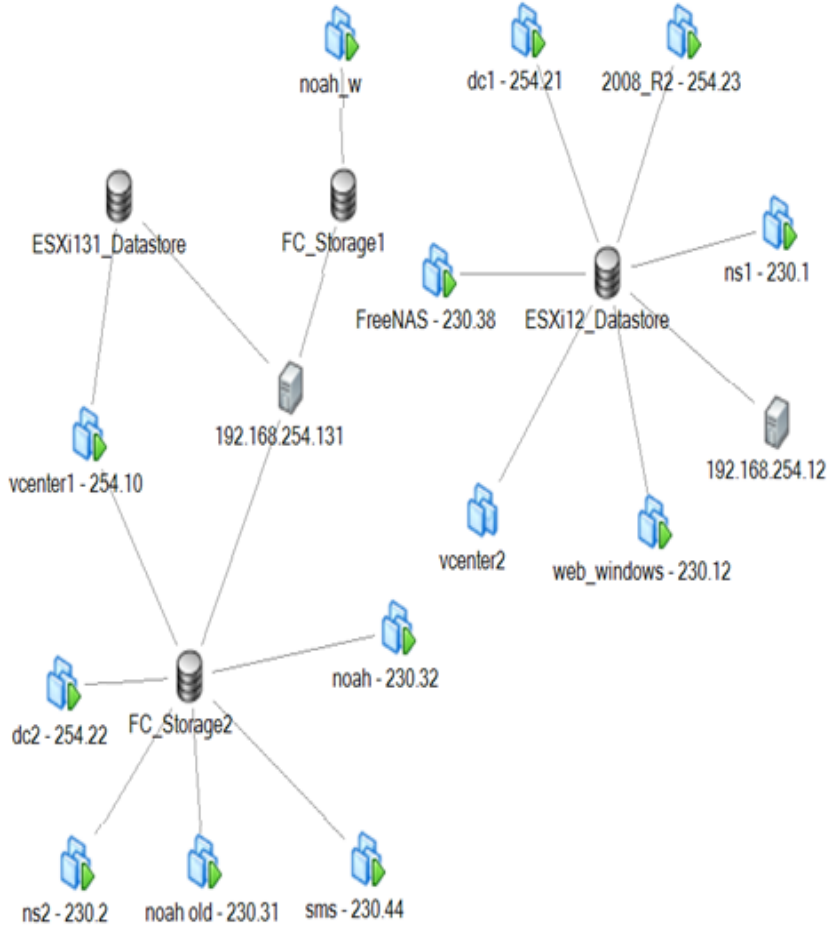
როგორც სქემა გვიჩვენებს, მოცემულ ფრაგმენტში 4 ტიპის კომპონენტია ასახული:

- **ფიზიკური ჰოსტი** (192.168.254.131);
- **მონაცემთა საცავი** (ESXi131_Datastore, FC_Storage1, FC_Storage2), რომლის პირველი წარმომადგენელიც ჰოსტის ლოკალური მეხსიერებაა, ხოლო ბოლო ორი ლოგიკური ტომებია მონაცემთა საცავიდან;
- **ვირტუალური მანქანა** (noah_w, noah-230.32 etc);
- **ვირტუალური ქსელი** (VM Net Vlan3, VM Net Vlan9, VM Network).

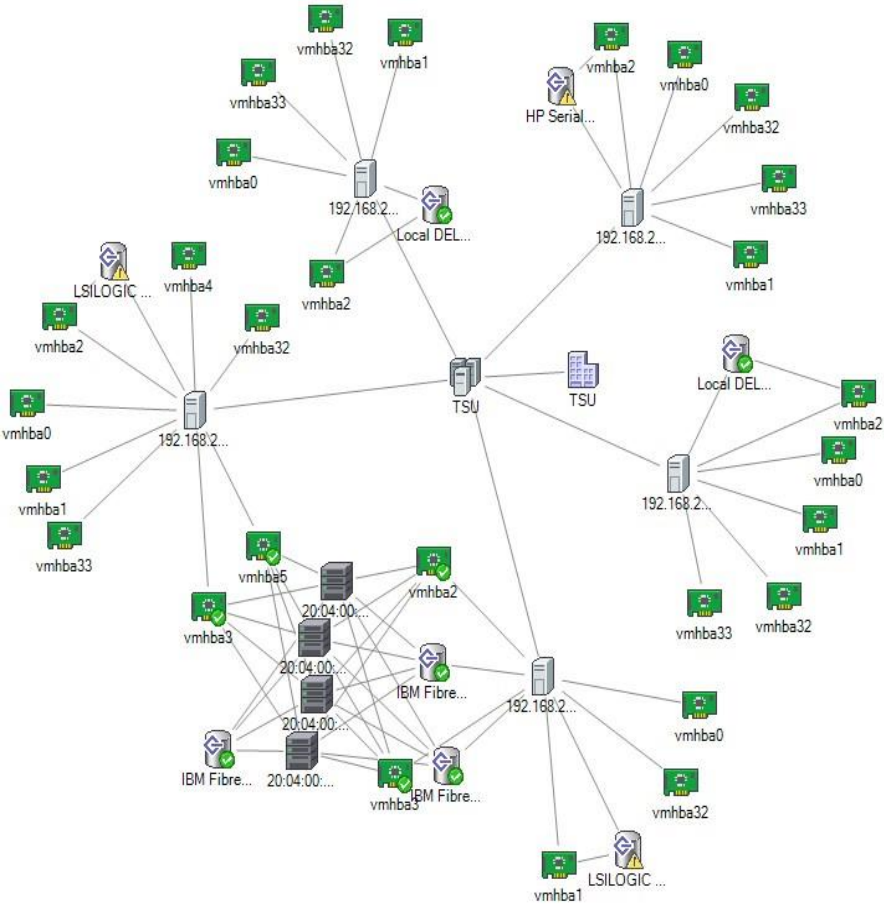
მორიგი „რუქები“ (ნახ.1.20 და 1.21) წარმოგვიდგენს სერვერული სისტემის ფრაგმენტებს ფიზიკური ჰოსტების და მონაცემთა საცავებისათვის სისტემის სხვა ელემეტებთან მათი კავშირების ჩვენებით.




**ნახ.1.20. კორპორაციული ქსელის ფრაგმენტი ფიზიკური ჰოსტისთვის**




ნახ.1. 21 კორპორაციული ქსელის ფრაგმენტი მონაცემთა საცავებისთვის




ნახ.1.22 კორპორაციული ქსელის ფრაგმენტი ოპტიკურბოჭკოვანი ინფრასტრუქტურისთვის

წიგნის ბოლოში მოცემულია დანართი_1, რომელშიც წარმოდგენილია მონაცემთა ბაზების კლასტერული არქიტექტურის შექმნა კორპორაციული ქსელის ვირტუალურ სერვერულ სივრცეში.

## პირველი ნაწილის დასკვნა

ნაშრომის მოცემულ ნაწილში განხილულია თანამედროვე ინფორმაციული ტექნოლოგიების უმნიშვნელოვანესი მიმართულება - კორპორაციული კომპიუტერული ქსელების სერვერული სისტემების არქიტექტურა.

ქსელურ გარემოში შენახული ინფორმაციის მოცულობა და კრიტიკულობა გახდა მთავარი მამოძრავებელი ძალა გასული საუკუნის ბოლოდან დღემდე დამკვიდრებული ახალი მეთოდებისა სერვერული სისტემების დაპროექტებისა და აგების პროცესში.

აღწერილია ახალი მიდგომები სერვერული ინფრასტრუქტურის დაპროექტებისა და აგების დანახარჯების შესამცირებლად მთლიანი სისტემის წარმადობის და საიმედოობის შემცირების გარეშე. ისეთი მეთოდები, როგორიცაა ვირტუალიზაცია, კლასტერული ტექნოლოგია და სხვ., ხელს უწყობს მოცემული ამოცანის მაქსიმალური ეფექტურობით შესრულებას.

**VMWare** ფირმის პროგრამულ სისტემათა ნაკრები **vSphere** ერთერთი საუკეთესო ინსტრუმენტია აღწერილი ამოცანების შესასრულებლად. ეს და სხვა დამატებითი პროგრამული პროდუქტები გამოიყენება კორპორაციული ქსელის სერვერული სისტემის სხვადასხვა კომპონენტების დაპროექტებისა და აგების დროს და მათ შორის დამოკიდებულებათა აგების მეთოდების მოდელირებისათვის (მონაცემთა საცავებისა და ვირტუალური ქსელების სახით).

განხილული თემატიკა პროგრამული უზრუნველყოფის მოწინავე მწარმოებელთა შეუნელებელი ყურადღების საგანია. ამ მიმართულებით ახალი ტექნოლოგიების განვითარებას და დანერგვას ახლო მომავალში უნდა ველოდოთ.

# ნაწილი II

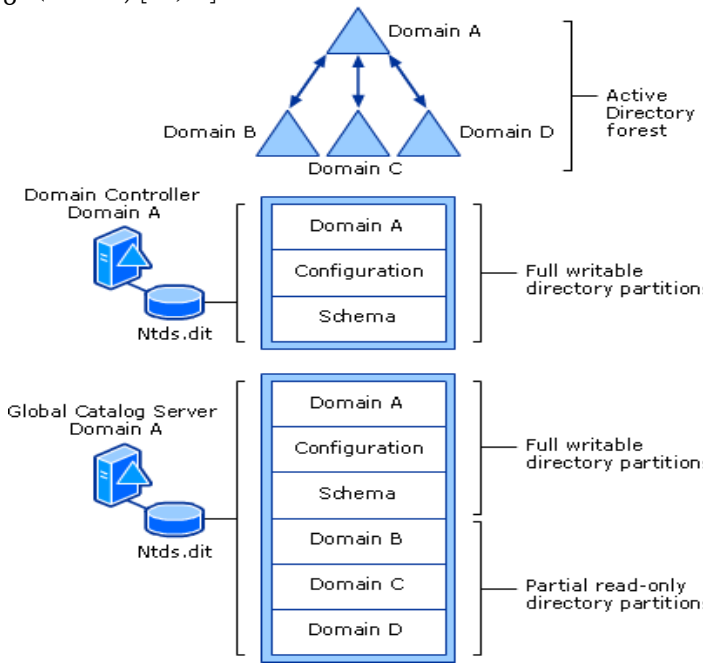
# სერვერული სისტემების ადმინისტრირება კორპორაციულ ქსელებში

# თავი 3. სისტემების ადმინისტრირების საფუძვლები

## 3.1. კატალოგური სერვისი - გამოთვლითი რესურსების მართვის ცენტრალიზებული სისტემა

### 3.1.1. Active Directory: Windows-ბაზირებული კატალოგური სერვისი

კატალოგების სამსახურების ძირითადი დანიშნულებაა ქსელური უსართხოების მართვა. ქსელური უსაფრთხოების საფუძველია მომხმარებლის, მომხმარებელთა ჯგუფებისა და კომპიუტერების სააღრიცხვო ჩანაწერების მონაცემთა ბაზა (accounts), რომელთა საშუალებითაც ხორციელდება ქსელურ რესურსებზე დაშვებების მართვა (ნახ.3.1) [17,18].




ნახ.3.1. კატალოგური სერვისის ზოგადი სქემა

კატალოგური სქემის აგება აუცილებელია შედარებით დიდი ზომის ლოკალურ ქსელებში, სადაც ვერ გამოიყენება მოდელი **„სამუშაო ჯგუფი“**, რომელიც ყველა სხვა მოდელზე პრიმიტიულია.

„სამუშაო ჯგუფის“ დანერგვა მცირე ქსელებშია მიღებული (3–10 კომპიუტერი). იგი ეფუძნება არქიტექტურას, რომელშიც ქსელის ყოველ კომპიუტერს (ოპერაციული სისტემებით Windows NT/2000-/XP/2003/Vista/2008/7) აქვს სააღრიცხვო ჩანაწერთა საკუთარი, ლოკალური მონაცემთა ბაზა და მისი დახმარებით ხორციელდება დაშვებების მართვა მოცემული, კონკრეტული კომპიუტერის რესურსებზე. სააღრიცხვო ჩანაწერების ლოგიკური მონაცემთა ბაზა (SAM – Security Account Manager) ინახება ოპერაციული სისტემის რეესტრში და იგი სრულებით იზოლირებულია სხვა კომპიუტერების ანალოგიური ბაზებისგან.

მოდელი „სამუშაო ჯგუფი“ შედარებით მარტივი აგებულებით გამოირჩევა და ადმინისტრირების მინიმალურ ხარჯებს მოითხოვს, მაგრამ იმ ქსელებში, სადაც მრავალი კომპიუტერი გამოიყენება, ქსელური რესურსების მართვის პროცესი რთულდება. მომხმარებლებს უწევთ ყოველ კომპიუტერზე საკუთარი სააღრიცხვო ჩანაწერების ხელით შექმნა ერთიდაიმავე პაროლით, რაც ინფორმაციის შენახვისა და მასთან მიმართვის სიცხადეს მეტად ამცირებს.

შედარებით დიდი ქსელების სამართავად დამუშავებულია **დომენური მოდელი,** რომელიც მოიცავს კატალოგურ სერვისს და ამ სერვისის ერთიან მონაცემთა ბაზას. დომენური მოდელის სამოქმედო სივრცე შეიძლება იყოს მთლიანი კორპორაციული ქსელის ან სულაც ქსელთა სიმრავლე.

დომენური მოდელი იყენებს სპეციალიზებულ სერვერებს, რომელთაც **დომენის კონტროლერები** ეწოდება.

დომენური მოდელის ძირითადი ტერმინოლოგია მოცემულია ქვემოთ:

**ერთიანი კატალოგი** არის ინფორმაციის ნაკრები ცალკეულ მომხმარებელთა და მათი ჯგუფების, ქსელში შემავალი კომპიუტერების, ქსელური პრინტერების, საერთო ფაილური რესურსებისა და სხვა ელემენტების შესახებ. ყველა ამ ელემენტს *ობიექტი* ეწოდება.

ობიექტს გააჩნია თვისებები (ატრიბუტები). მაგალითად, მომხმარებლის კატალოგში შენახული ატრიბუტები შეიძლება იყოს სახელი, ტელეფონის ნომერი, მისამართი, სახელი სისტემაში შესასვლელად, პაროლი, ჯგუფები, რომლებშიც მოხმარებელი შედის და სხვ.

Windows-ბაზირებულ ოპერაციულ სისტემებში დომენური სისტემის სამართავად გამოიყენება სისტემა **Active Directory (AD),** რომელიც არა მარტო საბაზო (მომხმარებელი, მომხმარებელთა ჯგუფი, კომპიუტერი), არამედ კომპლექსური ობიექტების, ე.წ. „ორგანიზაციული ქვედანაყოფების“ შექმნისა და მართვის საშუალებას, აგრეთვე კატალოგური სისტემის ფიზიკური და ლოგიკური დეკომპოზიციის საშუალებას იძლევა („საიტების“ სახით).

კორპორაციული ქსელის კატალოგური სერვისი Active Directory უზრუნველყოფს შემდეგ ფუნქციონალობას:

- **ერთიანი რეგისტრაცია ქსელში.** მომხმარებლები შეიძლება დარეგისტრირდნენ ქსელში ერთი სახელითა და პაროლით და ამასთან მიიღონ დაშვება ყველა ქსელურ რესურსთან და სამსახურთან (ქსელური ინფრასტრუქტურების სამსახური, ფაილების სამსახური და შეჯგუფების სამსახური, დანართების და მონაცემთა ბაზის სერვერები და ა.შ.). ამასთან, აღნიშნული სახელით და პაროლით მომხმარებელს შეუძლია თავის კომპიუტერში მოხვედრა გათიშული ქსელის პირობებშიც, ანუ როცა დომენის კონტროლერთან კავშირი არ არის (სააღრიცხვო ჩანაწერის კეშირების საფუძველზე);
- **ინფორმაციის უსაფრთხოება.** აუთენტიფიკაციის საშუალებები და რესურსებთან დაშვების მართვა;

- **ცენტრალიზებული დაცვა**. ქსელის ადმინისტრატორებს შეუძლიათ ერთიანად მართონ ყველა კორპორაციული რესურსი;
- **ადმინისტრირება ჯგუფური პოლიტიკის გამოყენებით.** კომპიუტერის დატვირთვისას ან მომხმარებლის რეგისტრაციისას სისტემაში ხორციელდება ჯგუფური პოლიტიკის მოთხოვნები, მათი კონფიგურაცია ინახება ჯგუფური პოლიტიკის ობიექტებში (GPO – Group Policy Objects) და გამოიყენება კომპიუტერების მომხმარებლის სააღრიცხვო ჩანაწერებში;
- **ინტეგრაცია DNS-თან.** კატალოგების სამსახურის ფუნქციონირება მთლიანად დამოკიდებულია დომენურ სახელთა სერვისის (DNS) სერვისების მუშაობაზე. თავის მხრივ სერვერების DNS-ებს შეუძლია შეინახონ ზონების ინფორმაცია Active Directory-ის მონაცემთა ბაზაში;
- **კატალოგის გაფართოებადობა.** ადმინისტრატორს შეუძლია კატალოგების სქემაში დაუმატოს ახალი ობიექტთა კლასები ან დაუმატოს ახალი ატრიბუტები არსებულ კლასებში;
- **მასშტაბირება**. Active Directory-ის სამსახურმა შეიძლება მოიცვას როგორც ერთი დომენი, ასევე მრავალი დომენი. დომენების **ხეთა** გაერთიანება ქმნის **ტყეს;**
- **ინფორმაციის რეპლიკაცია**. მოცემული სერვისის დახმარებით დომენური ინფორმაცია სინქრონულად ინახება რამდენიმე დომენის კონტროლერზე. რამდენიმე კონტროლერის არსებობა დომენებში უზრუნველყოფს მტყუნებათა მიმართ მდგრადობას და ქსელური დატვირთვების განაწილების შესაძლებლობას იძლევა.

### 3.1.2. კატალოგური სერვისის მონაცემთა ბაზა

დომენის კონტროლერებზე კატალოგური ინფორმაცია ინახება სპეციალური ფორმატის მონაცემთა ბაზის სახით. სერვისის განვითარების საწყის ეტაპზე კატალოგების სამსახურებისათვის შემუშავებულ იქნა სტანდარტი X.500, რომელიც ხის სტრუქტურის

მქონე, იერარქიული, მასშტაბირებადი ცნობარების შესაქმნელად იყო გათვალისწინებული, ობიექტთა კლასებისა და მათი ატრიბუტების ნაკრების გაფართოების შესაძლებლობით.

X.500 სტანდარტის პრაქტიკული რეალიზაცია არაეფექტური გამოდგა არასაკმარისი მწარმოებლურობის გამო, ამიტომ აღნიშნული სტანდარტის ბაზაზე შემუშავებული იქნა კატალოგის მონაცემთა ბაზის გამარტივებული ვერსია, LDAP (Lightweight Directory Access Protocol), რომელშიც შენაჩუნდა X.500-ის ყველა ეფექტური ფუნქცია და ამასთან სტანდარტის პრაქტიკული რეალიზების საშუალება მისცა მომხმარებელს.

სადღეისოდ LDAP არის ქსელური კატალოგების მონაცემებთან დაშვების სტანდარტული მეთოდი და მრავალი პროდუქტის საფუძველია (აუთენტიფიკაციის სისტემები, ელექტრონული ფოსტის პროგრამები, ელექტრონული კომერციის სისტემები და სხვა). LDAP-სერვისები დამუშავებულია პრაქტიკულად ყველა გავრცელებული ოპერაციული სისტემისათვის და დიდია მისი როლი ღრუბლოვან ტექნოლოგიებშიც, რადგან ეს უკანასკნელი კატალოგურ სერვისს ინტენსიურად საჭიროებს.

დღეისათვის ბაზარზე LDAP-ის 60-ზე მეტი კომერციული სერვერია წარმოდგენილი. მათი 90% არის LDAP-კატალოგების ავტონომიური სერვერი, ხოლო დანარჩენი – სხვა დანართების კომპონენტების სახით არის წარმოდგენილი. ჩვენ ნაშრომში გამოყენებულია ერთ-ერთი მათგანი – Active Directory.

### 3.1.3. კატალოგური სერვისის დომენური სტრუქტურა

მოცემულ პარაგრაფში შევეხოთ კატალოგური სერვისის სტრუქტურულ ერთეულებს (დომენი, ხე, ტყე, ორგანიზაციული ერთეული, გლობალური კატალოგი), აგრეთვე განვიხილოთ კატალოგის ობიექტთა სახელდების წესები.

**დომენი** არის კატალოგური სერვისის ძირითადი ერთეული. იგი აყალიბებს ადმინისტრაციული პასუხისმგებლობის სფეროს. დომენის მონაცემთა ბაზა შეიცავს მომხმარებლის სააღრიცხვო ჩანაწერებს, ჯგუფებსა და კომპიუტერებს. კატალოგური სერვისის ფუნქციების დიდი ნაწილი სწორედ დომენურ დონეზე მუშაობს, კერძოდ: მომხმარებლის აუტენტიფიკაცია, რესურსებთან დაშვების მართვა, სამსახურების მართვა, რეპლიკაციების მართვა, უსაფრთხოების პოლიტიკა.

დომენების სახელები Active Directory-ში ფორმირდება იმავე სქემით, როგორც დომენურ სახელთა სივრცის (DNS) სახელები და ეს არ არის შემთხვევითი ფაქტი. DNS-ის სამსახური წარმოადგენს კომპონენტ-დომენების ძებნის პროცესის საინიციალიზაციო მოვლენას დომენის კონტროლერებზე.

**დომენის კონტროლერი** სპეციალური სერვერია, რომელიც ინახავს Active Directory-ის მონაცემთა ბაზის დომენური ნაწილის შესაბამის მონაცემებს. მათი ძირითადი ფუნქციებია:

- Active Directory-ის მონაცემთა ბაზის შენახვა (კატალოგურ ინფორმაციასთან წვდომა, მართვა და მოდიფიკაცია);
- Active Directory-ში ცვლილებათა სინქრონიზაცია. კერძოდ, კატალოგში ცვლილების შეტანა შეიძლება დომენის ნებისმიერი კონტროლერიდან; ნებისმიერი ცვლილება სინქრონულად დაფიქსირდება დომენის სხვა კონტროლერებზეც;
- მომხმარებლის აუთენტიფიკაცია – ნებისმიერი დომენის კონტროლერი ახორციელებს კლიენტ-სისტემაზე დარეგისტრირებული მომხმარებლის უფლებამოსილების შემოწმებას.

სადღეისოდ დომენის კონტროლერები თანაბარუფლებიანი სერვერებია, ხოლო წინა ვერსიებში PDC (Primary Domain Controller) და BDC (Backup Domain Controller) სერვერების სახით არსებობდა.

**ხე** არის დომენების კრებული, რომელიც გამოიყენებს ერთიან დამაკავშირებელ სივრცეს. ამ შემთხვევაში „შვილობილი“ დომენი მემკვიდრეობით იღებს „მშობელი“ დომენის სახელს.

შვილობილი დომენი ავტომატურად აყენებს **ორმხრივ, ტრანზიტულ, ნდობით დამოკიდებულებებს** მშობელ დომენთან, რაც ნიშნავს, რომ ერთი დომენის რესურსები შეიძლება გამოყენებულ იქნას სხვა დომენების მომხმარებელთა მიერაც. ზოგადად, ამ დროს დომენებს შორის ნდობითი დამოკიდებულებები შეიძლება იყოს **ორმხრივი** ან **ცალმხრივი**. დამოკიდებულება ცალმხრივია როცა **A** დომენი ენდობა **B**-ს. ამ დროს **B**-დომენში მოხვედრილი მომხმარებელი თავისი სააღრიცხვო ჩანაწერით **A**-ში მოხვედრასაც ახერხებს და არა პირიქით. უკუკავშირს ცალკე დამოკიდებულების დამყარება ჭირდება. ამასთან, ნდობითი დამოკიდებულებები შეიძლება იყოს როგორც ტრანზიტული, ასევე **არატრანზიტული** (**A->B->C** დამოკიდებულება არ მოქმედებს).

ყველა დომენურ მომხმარებელს ერთ დომენში მაინც უნდა ჰქონდეს სააღრიცხვო ჩანაწერი. თუ მომხმარებელს სააღრიცხვო ჩანაწერი არა აქვს, იგი დომენში შესვლას მხოლოდ მაშინ შეძლებს, როცა ამ დომენს ნდობითი დამოკიდებულება აქვს იმ დომენთან, რომელშიც მომხმარებელი შედის.

**ტყე** მეტნაკლებად მსხვილი სტრუქტურაა Active Directory-ში. ტყე აერთიანებს დომენურ ხეებს სხვადასხვა სახელთა სივრცეებით და ამით უზრუნველყოფს ობიექტთა ერთიან სქემის არსებობას (სქემა – განსაზღვრული ტიპების ან კლასების კრებულია). ტყეში ყველა დომენს შორის ორმხრივი, ტრანზიტული ნდობითი დამოკიდებულებაა დამყარებული, რაც ნებისმიერი დომენის მომხმარებლებს ყველა დანარჩენ დომენებზე დაშვებით უზრუნველყოფს შესაბამისი დაშვების უფლების არსებობის შემთხვევაში. სტანდარტულად, ტყეში შექმნილი პირველი დომენი ფესვურ დომენად ითვლება. სწორედ მასში ინახება ე.წ. **AD**-სქემა.

**ორგანიზაციული ერთეული (Organiazational Units, OU)** მოთავსებულია Active Directory-ის ერთერთი დომენის ფარგლებში და უფლებათა დელეგირების ამოცანას ემსახურება. ორგანიზაციული ერთეული დომენის გარეთ არ არსებობს. მისი გამოყენება განსაკუთრებით მოხერხებულია დიდი ორგანიზაციის ერთიანი დომენის აგებისას, როდესაც დომენის ადმინისტრატორს ფიზიკურად აღარ შეუძლია მთლიანი დომენის ყველა სააღრიცხვო ჩანაწერის მომსახურება. ამ დროს დომენის ადმინისტრატორი ასრულებს დელეგირებას, ანუ საკუთარი უფლებების გადაცემას სხვა მომხმარებელზე, რომელსაც შემდგომ დომენში საკუთარი „წილი" ეძლევა მასში ცვლილებების შეტანის უფლებით. ორგანიზაციული ერთეული მთლიანად დომენს დაქვემდებარებული სტრუქტურაა, მას არ შეუძლია კატალოგური ინფორმაციის მიღება სხვა დომენიდან.

**გლობალური კატალოგი** არის დომენური სივრცის ყველა ობიექტის ერთიანი ბაზა. სტანდარტულად, დომენების კონტროლერები შეიცავს ინფორმაციას მხოლოდ საკუთარი ობიექტების შესახებ, მაშინ, როდესაც გლობალური კატალოგის სერვერი წარმოადგენს დომენის კონტროლერს, რომელშიც განთავსებულია ინფორმაცია დომენთა ტყის ყოველი ობიექტის შესახებ. ამასთან, უნდა აღინიშნოს, რომ გლობალურ კატალოგში ობიექტთა ატრიბუტების მხოლოდ ნაწილი ხვდება – ყველაზე მნშვნელოვანი ატრიბუტების სახით.

კატალოგურ სერვისში მნიშვნელოვანია **ობიექტების სახელების განმასხვავებელი მექანიზმი (Distinguished Name, DN),** რომელიც კატალოგის ნებისმიერი ობიექტის დომენთა ტყის მასშტაბით ცალსახად იდენტიფიცირების საშუალებას იძლევა. **DN** არის ქვესახელების ერთობლიობა, რომელთა კომბინაცია გვაძლევს სწორედ ობიექტის უნიკალურ იდენტიფიკატორს.

მაგალითისათვის ავიღოთ მომხმარებელი **testuser** დომენიდან **sangu.ge,** რომელიც განთავსებული დომენის მომხმარებელთა

სტანდარტულ კონტეინერში **„Users“.** მომხმარებლის უნიკალური სახელი შემდეგი ფორმით იქნება წარმოდგენილი:

**DC=ge, DC=sangu, OU=Professors, CN=Users, CN=testuser**

სადაც

- **DC (Domain Component)** – პირველი და მეორე დონის დომენური სახელები (მაგალითად, უნივერსიტეტის ან მისი რომელიმე ფაკულტეტის დომენი);
- **OU (Organizational Unit)** – ორგანიზაციული ერთეული (უნივერსიტეტის პროფესორები);
- **CN (Common Name)** – მომხმარებელთა სტანდარტული ჯგუფი (**Users**) და მასში შემავალი მომხმარებლის სახელი (**testuser**).

**Active Directory**-ის სახელთა განაწილების ეფექტური სქემის დაპროექტება ძალიან საპასუხისმგებლო საქმეა კორპორაციული ქსელის უსაფრთხოების თვალსაზრისითა და იმ ფაქტორის გათვალისწინებით, რომ უკვე ჩამოყალიბებული სახელთა სტრუქტურის შეცვლა ძნელ, ხანდახან კი შეუძლებელ ამოცანას წარმოადგენს. მაგალითად, ოპერაციული სისტემა Windows 2000-ის ფარგლებში ზედა დონის დომენის სახელის შეცვლა საერთოდ შეუძლებელია, ხოლო Windows 2003-ში ტექნიკურად საკმაოდ ძნელი ამოცანაა.

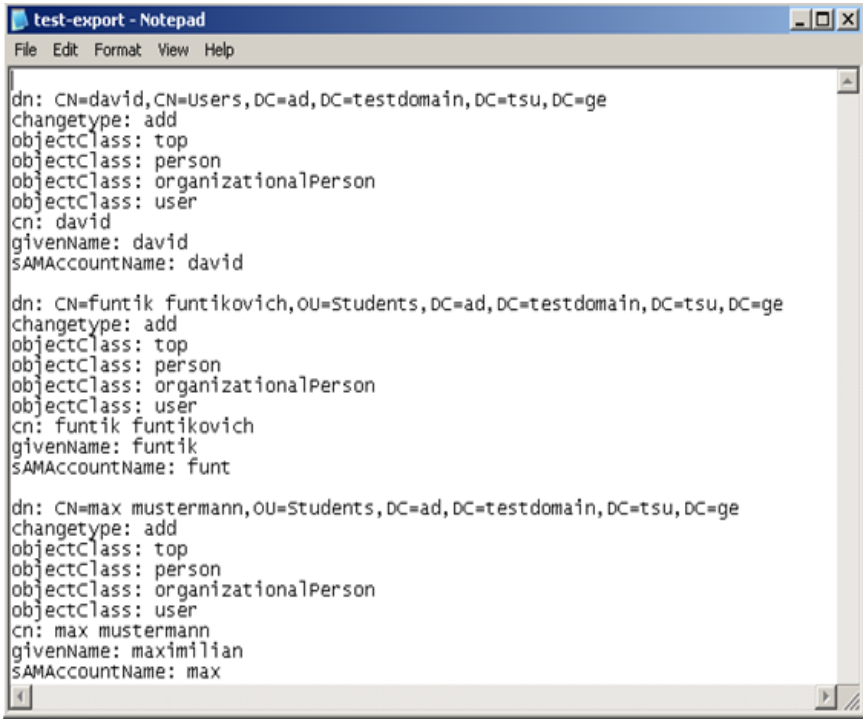
ამიტომ სახელთა სივრცის დაპროექტებისას შემდეგი ფაქტორები უნდა იქნეს გათვალისწინებული:

- ზედა დონის დომენების სახელების გულდასმით შერჩევა;
- კომპანიაში კომუნიკაციების ხარისხი (კავშირი ცალკეულ ქვედანაყოფებთან და ფილიალებთან);
- კომპანიის ორგანიზაციული სტრუქტურა;
- კომპიუტერების მომხმარებელთა რაოდენობის ზრდის ტემპების პერსპექტივა.

### 3.1.4. მოხმარებელთა მიგრაცია LDAP-ბაზებში

დომენურ მონაცემთა ბაზებში (LDAP) მომხმარებელთა მიგრაციისათვის აუცილებელი შემავალი და გამომავალი ინფორმაცია LDF-ფორმატის ფაილებში ინახება. ქვემოთ მოცემულია მომხმარებელთა ექსპორტის ბრძანება და შედეგად მიღებული ფაილის ფრაგმენტი.

```
ldifde -f test-export.ldf -s server1 -d
„dc=ad,dc=testdomain,dc=tsu,dc=ge“ -p subtree -r
„(&(objectCategory=person)(objectClass=User)(givenname=*))“ -l
„cn,givenName,objectclass,samAccountName“
```



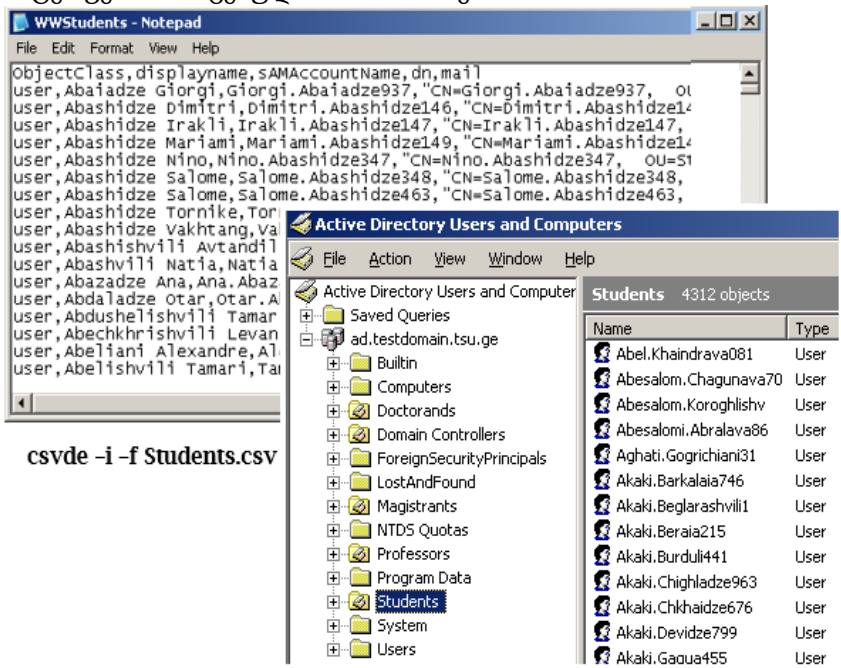

ნახ.3.2. მოხმარებელთა ექსპორტი LDF-ფაილებში

მომხმარებელთა იმპორტის ბრძანებას შემდეგი სახე შეიძლება გააჩნდეს:

```
ldifde i -f test-import.ldf -s server1
```

იგივე პროცედურების შესასრულებლად გარე უტილიტებიც გამოიყენება. მაგალითად, უტილიტა **CSVDE** ასრულებს **LDAP**-ბაზაში მოხმმარებლების ექსპორტისა და იმპორტის პროცედურებს **CSV**-ფორმატის ფაილებიდან.

**CSV**-ფაილის ფრაგმენტი, უტილიტის გამოყენების ნიმუში, და შესრულების შედეგის ასახვა დომენურ მომხმარებელთა მართვის ინტერფეისში მოცემულია 3.3 ნახაზზე.




ნახ.3.3. მოხმარებელთა იმპორტი CSV-ფაილებიდან

და ბოლოს, მოვიყვანოთ პატარა **Shell**-სკრიპტის ნიმუში იმის საილუსტრაციოდ, თუ როგორ შეიძლება დავაყენოთ **LDAP**-ბაზასთან სამუშაო უტილიტა **LINUX**-ბაზირებულ ოპერაციულ სისტემაში და შევასრულოთ დომენური ინფორმაციის ძიება.

```
apt-get install ldap-utils
ldapsearch -v -x -H ldap://ad.testdomain.tsu.ge -b 'cn=Admin,
ou=Students, o=Students'
```

## 3.2. ჯგუფური პოლიტიკის მართვა Windows-ის გარემოში

**Windows-სისტემის (Active Directory) ჯგუფური პოლიტიკა** (Group Policy) ცალკეულ მომხმარებელთა, ჯგუფების და გამოთვლითი რესურსების მართვის ამოცანას ემსახურება. მისი საშუალებით ორგანიზაციის IT-ინფრასტრუქტურის ფარგლებში ყველა მომხმარებელმა „იცის“, **რა უფლებები გააჩნია, რომელი ფუნქციები ეზღუდება** და **რისგან არის დაზღვეული**.

ჯგუფური პოლიტიკის გამოყენების სივრცე პრაქტიკულად ყველა იმ რესურსს მოიცავს, რომელზე კომპანია Microsoft-ის ოპერაციული სისტემები მუშაობს: სერვერები, პერსონალური კომპიუტერები, ნოუტბუქები და სხვა მობილური მოწყობილობები.

პოლიტიკა შეიძლება განისაზღვროს Active Directory-დომენის, ორგანიზაციული ერთეულის (Organisational Unit – OU) ან ლოკალური გამოთვლითი სისტემისთვის. ამასთან, ჯგუფური პოლიტიკა კუმულატიურია და მოქმედების შემდეგი პრიორიტეტები გააჩნია:

- ლოკალური ჯგუფური პოლიტიკა;
- დომენის ჯგუფური პოლიტიკა;
- ზემდგომი ორგანიზაციული ერთეულის ჯგუფური პოლიტიკა;

- მომდევნო დონის ორგანიზაციული ერთეულის ჯგუფური პოლიტიკა;
- ...

ამრიგად, თუ მაგალითად, მომხმარებელს ლოკალური ჯგუფური პოლიტიკით ინტერნეტში გასვლა ეკრძალება, ხოლო დომენის პოლიტიკით ნებადართული აქვს, იგი შეძლებს ინტერნეტთან წვდომის განხორციელებას.

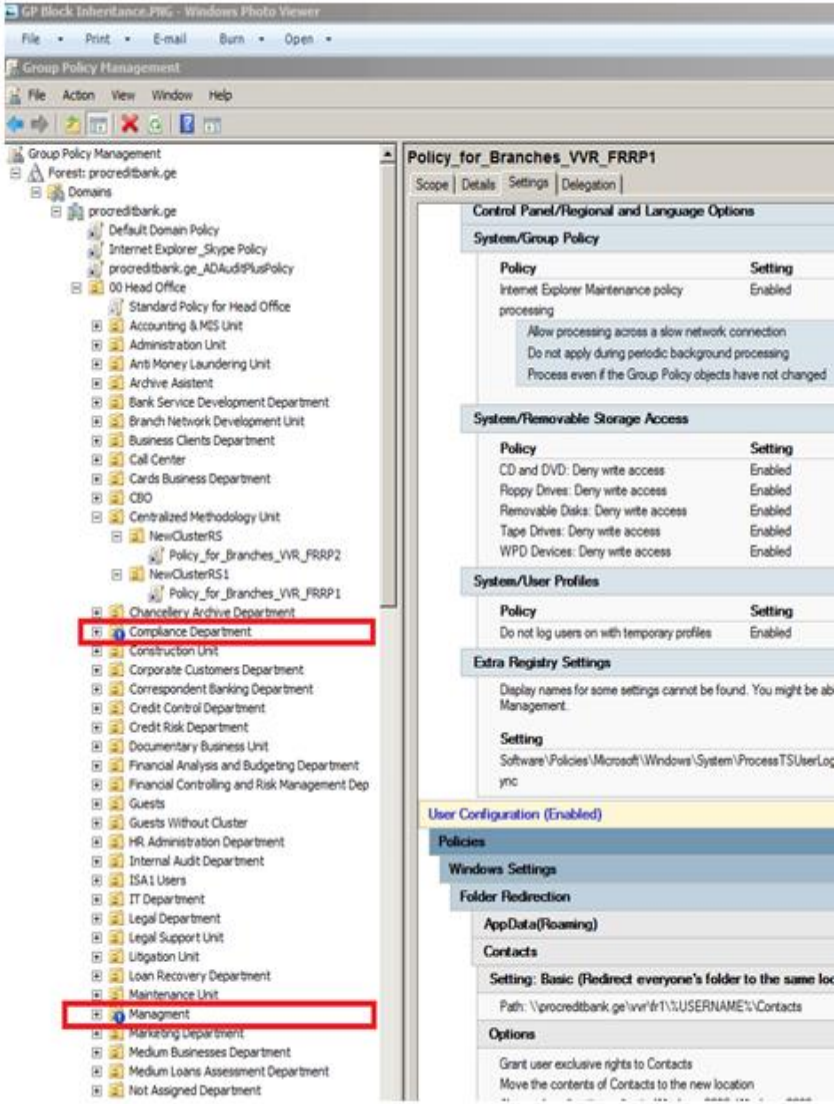
აუცილებელია აღინიშნოს, რომ ზემოთ მოყვანილი პრიორიტეტების სქემაში ხანდახან ხელოვნურად ცვლილებების შეტანა ხდება საჭირო. ვთქვათ, თუ რომელიმე OU-ს დონეზე ზემდგომი OU-ებიდან ან დომენიდან „ჩამოსული“ ჯგუფური პოლიტიკები უნდა დაიბლოკოს ახალი პოლიტიკა „სუფთა ფურცლიდან“ უნდა განისაზღვროს, OU-ზე მაუსის მარჯვენა ღილაკის დაკლიკვის შემდეგ გამოიძახება ბრძანება Block Inheritance (ნახ.3.4).

მეორე მხრივ, თუ გვსურს ზემდგომი დომენის ან OU-ს ჯგუფური პოლიტიკა ქვედა დონის OU-ებს „თავს მოვახვიოთ“, გამოიყენება ბრძანება: მაუსის მარჯვენა ღილაკი ჯგუფურ პოლიტიკაზე -> (ნახ.3.5).

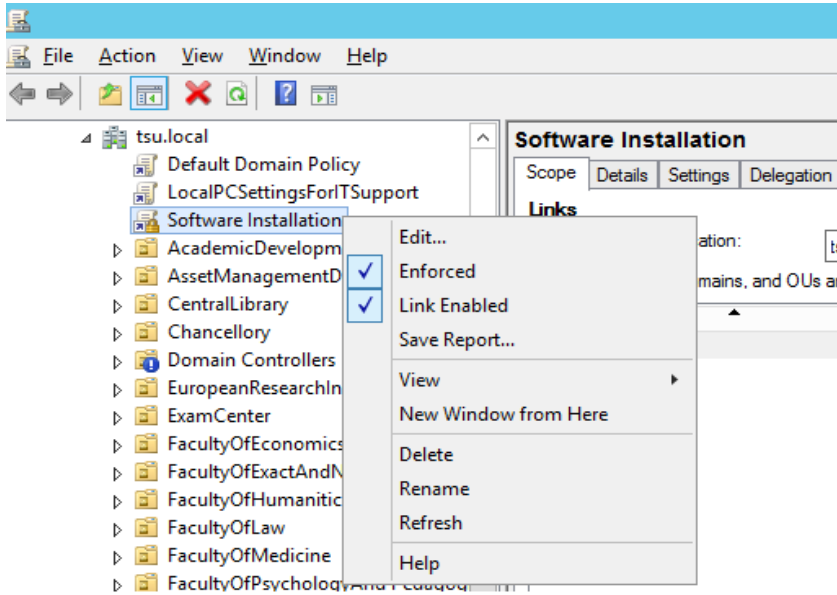
ჯგუფური პოლიტიკის მართვისთვის Windows-ის სპეციალური უტილიტა (Snap-in) gpedit.msc გამოიყენება, რომელიც ბრძანებათა სტრიქონიდან შეიძლება გაეშვას. მოვიყვანოთ კიდევ რამდენიმე ბრძანება ჯგუფური პოლიტიკის ეფექტური მართვისთვის:

- **gpupdate /force** - ახალი ჯგუფური პოლიტიკის ძალაში შესვლა მიმდინარე გამოთვლით სისტემაში;
- **gpresult /z** - მიმდინარე ჯგუფური პოლიტიკების ნახვა;
- **secpol.msc** – უსაფრთხოების პოლიტიკის მმართველი;

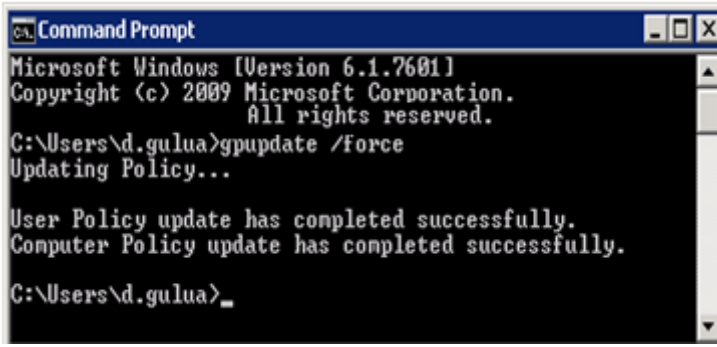
ქვემოთ, სურათზე მოცემულია უტილიტა gpupdate-ის შესრულების ნიმუში (ნახ.3.6).




ნახ.3.4. OU-ები დაბლოკილი ზედა დონის ჯგუფური პოლიტიკებით




ნახ.3.5. ჯგუფური პოლიტიკის გავრცელება ქვემდგომ OU-ებზე




ნახ.3.6. უტილიტა gpupdate

ჯგუფური პოლიტიკის საშუალებით უამრავი წესის განსაზღვრა და მართვა შეიძლება. მოვიყვანოთ რამდენიმე მაგალითი:

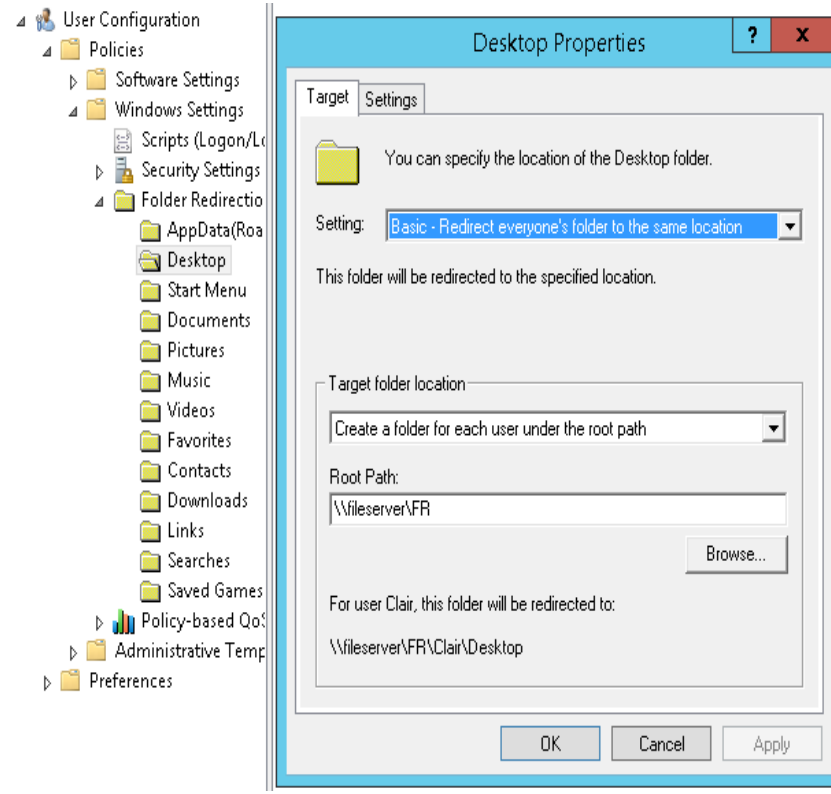
- ქსელური ფაილური რესურსების გამოყოფა და მართვა;
- დომენური ჯგუფის ან მომხმარებლის ჩასმა ლოკალურ ჯგუფში;
- პაროლების პოლიტიკის შემუშავება;
- ინტერნეტთან წვდომის მართვა;
- და სხვა მრავალი...

განვიხილოთ ჯგუფური პოლიტიკის რამდენიმე ნიმუში. **ქსელური ფაილური რესურსების გამოყოფისა და მართვის** ამოცანებიდან ყველაზე მნიშვნელოვანია **ფოლდერების გადამისა-მართება (Folder Redirection),** რომელსაც მნიშვნელოვანი მომხმარებლის მონაცემთა საიმედო შენახვა ევალება.

როგორც წესი, ორგანიზაციის თანამშრომლებს ჩვევად აქვთ, მნიშვნელოვანი ფოლდერები და ფაილები Windows-ის სამუშაო მაგიდაზე, ანუ დესკტოპზე განათავსონ და არ შეიწუხონ თავი მათი სარეზერვო ასლების შექმნით. ასეთ შემთხვევაში მნიშვნელოვანი მონაცემების დაკარგვის შანსი დიდია, ამიტომ იქმნება ჯგუფური პოლიტიკა, რომელიც მომხმარებლის **დესკტოპს, My Documents, My Pictures, My Music** და სხვა ფოლდერებს ავტომატურად შეუცვლის ადგილმდებარეობას, ანუ მომხმარებელს ჰგონია, რომ მისი ფაილები დესკტოპზეა, ფიზიკურად კი ისინი საიმედო ფაილ-სერვერზეა განთავსებული. შესაბამისი ჯგუფური პოლიტიკის განსაზღვრა სრულდება შემდეგნაირად:

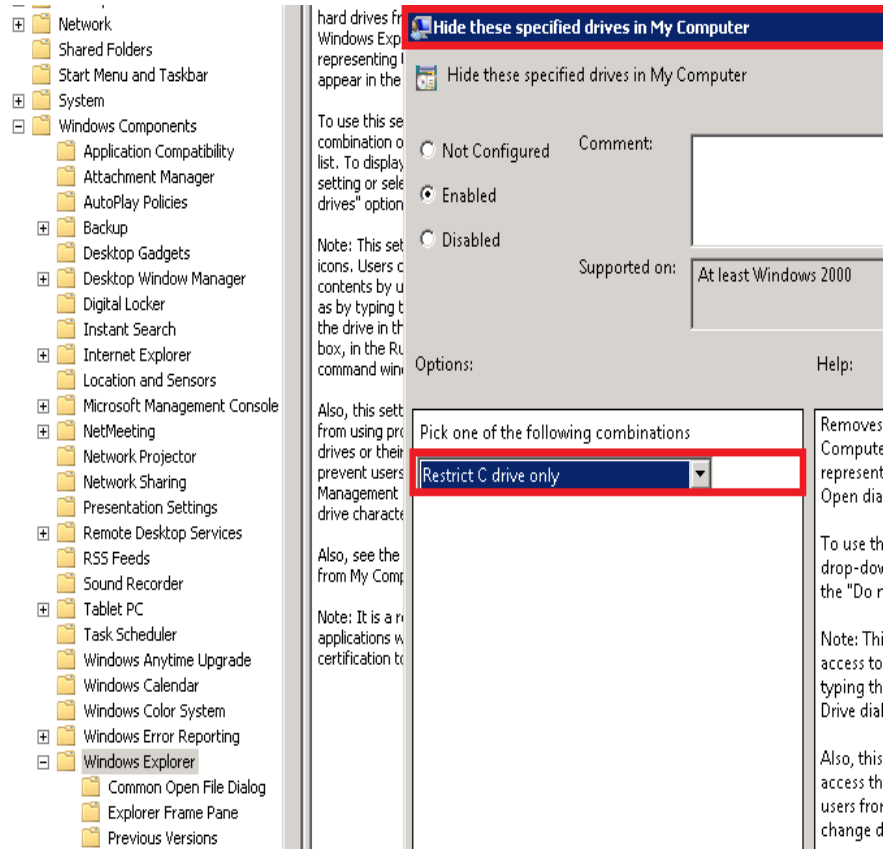
**User Configuration -> Windows Settings -> Folder Redirection**

იხილეთ შესაბამისი სურათი ჯგუფური პოლიტიკის მართვის ინტერფეისიდან (ნახ.3.7).



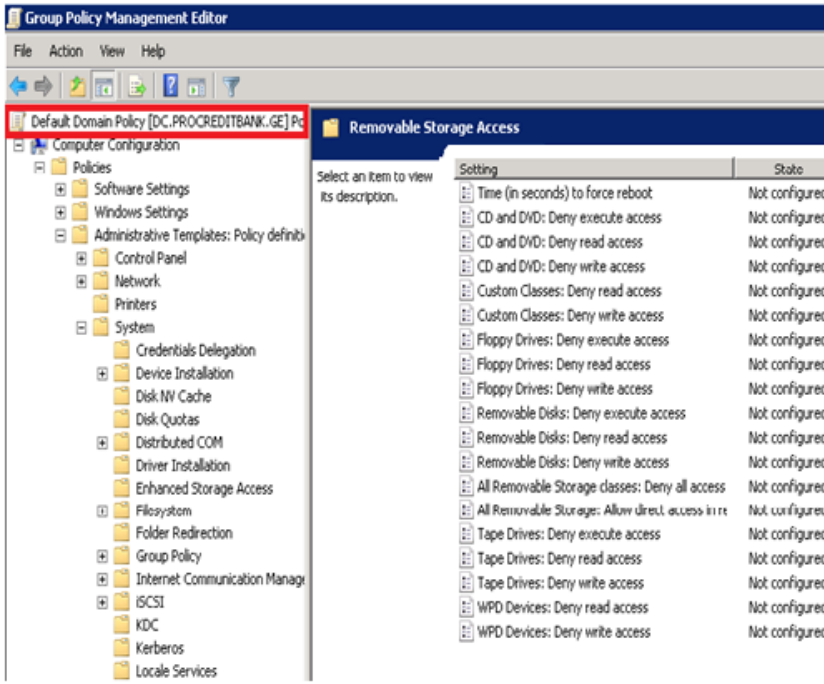

**ნახ.3.7. ჯგუფური პოლიტიკა ფოლდერების გადამისამართებისთვის**

მომდევნო მაგალითში წარმოდგენილია ჯგუფური პოლიტიკა, რომელიც მომხმარებლებს **სისტემურ C:\ დისკთან წვდომას უზღუდავს,** რაც ხშირად მნიშვნელოვანია უსაფრთხოების თვალსაზრისით (ნახ.3.8).




**ნახ.3.8. სისტემურ დისკზე წვდომის შეზღუდვის ჯგუფური პოლიტიკა**

შეზღუდვა შეიძლება ასევე შეეხოს კომპიუტერის ყველა „მოხსნად“ (Removable) მოწყობილობას. მომდევნო სურათზე ნაჩვენებია ჯგუფური პოლიტიკის ნიმუში CD/DVD-დისკების, ფლეშ-დისკების, გარე ხისტი დისკების, მეხსიერების ბარათებისა და ამ ტიპის სხვა მოწყობილობების ბლოკირებისთვის (ნახ.3.9).



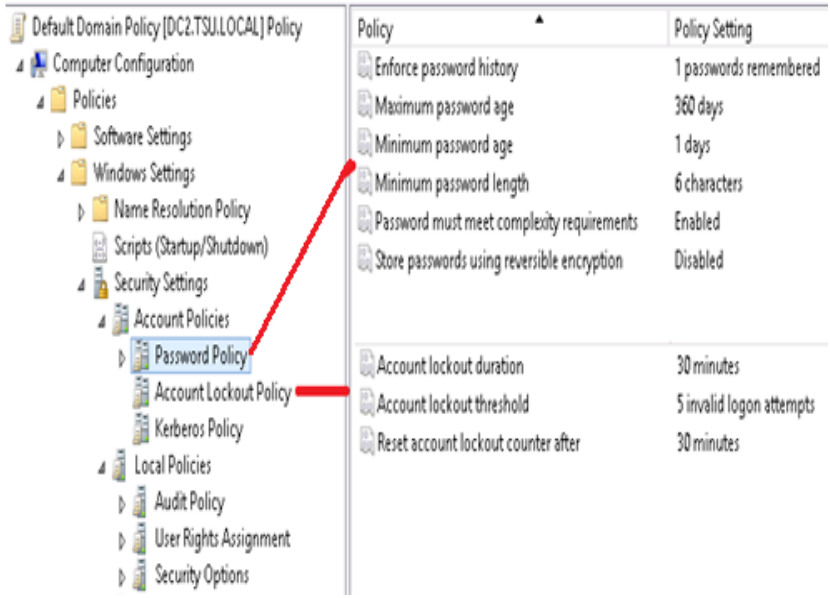

**ნახ.3.9. მოხსნადი მოწყობილობების ბლოკირების ჯგუფური პოლიტიკა**

ჯგუფური პოლიტიკის მომდევნო მაგალითში მომხმარებლებს ეკრძალებათ კომპიუტერთან უშუალოდ მისი განლაგების ადგილზე (ლოკალურად) მიერთება. მოცემული ჯგუფური პოლიტიკა ასევე უსაფრთხოების ერთერთ ზომას წარმოადგენს და შემდეგი ბრძანებით განისაზღვრება:

**Computer Configuration -> Windows Settings -> Security Settings -> Local Policies -> User Rights Assignment -> Deny Log on Locally**

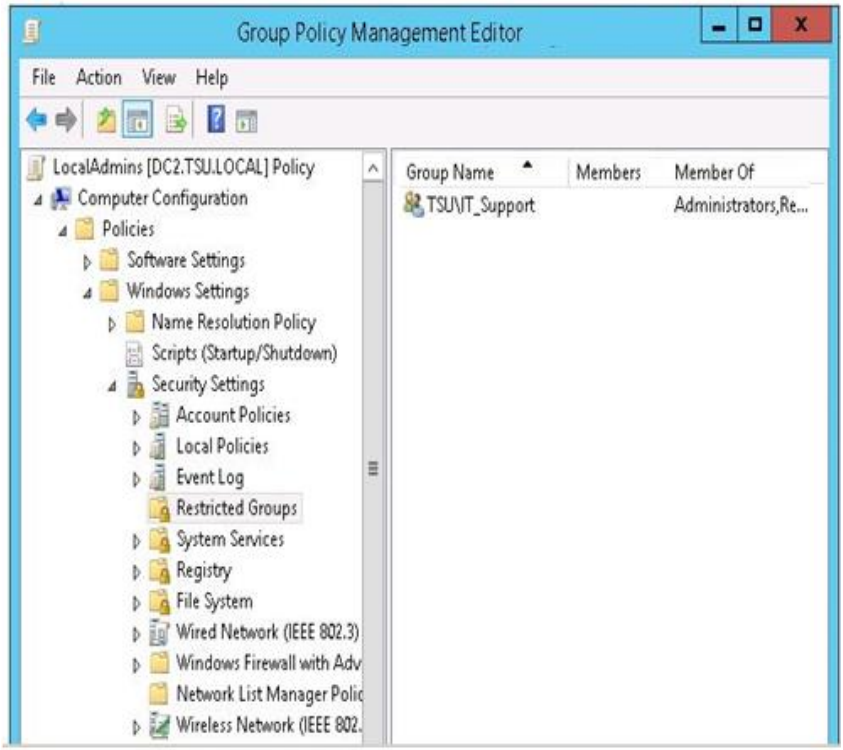
**უსაფრთხო პაროლების** გამოყენება ყველა ორგანიზაციის IT-ინფრასტრუქტურის უმნიშვნელოვანეს პრობლემათაგანია. მომდევნო ჯგუფური პოლიტიკა აიძულებს მომხმარებელს,

სისტემაში შესასვლელად გამოიყენოს მინიმუმ 6-სიმბოლოიანი პაროლი, სადაც ერთი მაინც დიდი ლათინური სიმბოლო, პატარა ლათინური სიმბოლო და არაბული ციფრი იქნება წარმოდგენილი (ნახ.3.10).



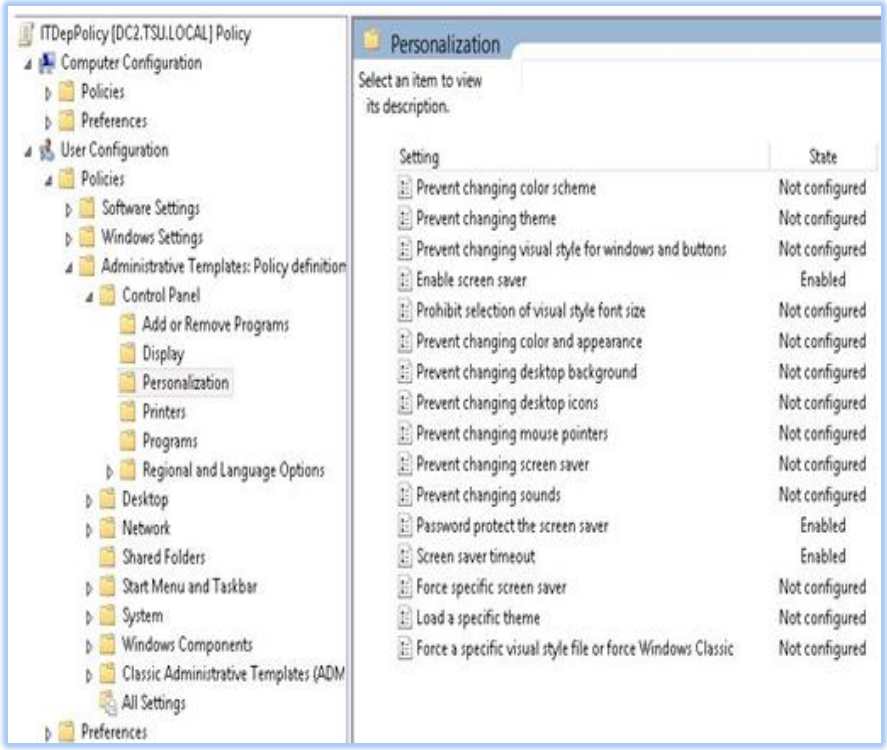

ნახ.3.10. პაროლების მართვის ჯგუფური პოლიტიკა

მომდევნო ჯგუფური პოლიტიკა ორგანიზაციის კომპიუტერებზე, ადმინისტრატორთა ლოკალურ ჯგუფში IT-მხარდაჭერის გლობალური, დომენური ჯგუფის ავტომატურად ჩასმას გულისხმობს. მოცემული პოლიტიკის მიზანს კომპიუტერებთან IT-მხარდაჭერის ჯგუფის თათამშრომელთა უპრობლემოდ დაშვება წარმოადგენს (ნახ.3.11).




ნახ.3.11. დომენური ჯგუფის ლოკალურ ჯგუფში ჩასმის ჯგუფური პოლიტიკა

ბოლო მაგალითში მოყვანილი ჯგუფური პოლიტიკის მეშვეობით, თუ მომხმარებლის კომპიუტერი გარკვეული დროის (მაგალითად, 10 წუთის) განმავლობაში უმოქმედოა, ეკრანი ავტომატურად „Lock“-რეჟიმში გადავა („დაილოქება“) და მუშაობის გასაგრძელებლად სახელის და პაროლის ხელახლაშეყვანა იქნება საჭირო (ნახ.3.12).




**ნახ.3.12. ეკრანის ავტომატური „დალოქვის“ ჯგუფური პოლიტიკა**

დასკვნის სახით შეიძლება ითქვას, რომ ჯგუფური პოლიტიკა Windows-გარემოში შეუცვლელი ინსტრუმენტია და მისი ეფექტური გამოყენება მკვეთრად ამაღლებს მთლიანი IT-ინფრასტრუქტურის სარგებლიანობას.

## 3.3. Web-ის, ფაილური და მონაცემთა ბაზის სერვერების გამართვა

### 3.3.1. ვებ-სერვერული ინფრასტრუქტურის აწყობა და მართვა

#### 3.3.1.1. ვებ-ინფრასტრუქტურის აგებულება

თანამედროვე ელექტრონულ ინფორმაციულ პროცესებში ვებ-ტექნოლოგიის როლი განუხრელად იზრდება. მე-20 საუკუნის 40-იან წლებში რამდენიმე მეცნიერის (ვანევარ ბუში, ტედ ნელსონი, დაგლას ენგელბარტი) მიერ ჰიპერტექსტური ინფორმაციული მოდელის შექმნამ, ხოლო 1990 წელს CERN-ში ტიმ ბერნერს-ლის მიერ World Wide Web (WWW)-ტექნოლოგიის დამუშავებამ უდიდესი ციფრული ინფორმაციული სივრცის - მსოფლიო აბლაბუდის შექმნას ჩაუყარა საფუძველი. დღეს როგორც გლობალური, ასევე ლოკალური (კორპორაციული) ინფორმაციული სისტემების დიდი ნაწილი ვებ-ტექნოლოგიის საფუძველზე იქმნება, ხოლო ძველი, დესკტოპ-აპლიკაციების ახალი ვერსიებიც, როგორც წესი, იმავე მიმართულებით ვითარდება.

მე-20 საუკუნის 90-იანი წლების მსოფლიო აბლაბუდა ინფორმაციული კონტენტის მხოლოდ *გამოყენებას* (ვებ-გვერდის წაკითხვა, ფაილების და პროგრამების ჩამოტვირთვა და ა.შ.) ითვალისწინებდა, თუმცა საუკუნეთა გასაყარზე შექმნილი ახალი კონცეფციის (WEB 2.0) მიხედვით, ვებ-მომხმარებელი იმავდროულად *ვებ-კონტენტის რედაქტორიცაა*, ანუ ვებ-გვერდის გარეგნული სახის შეცვლაში მონაწილეობს. WEB 2.0 *რედაქტირებადი აბლაბუდაა*, ანუ ინფორმაციული სივრცე, რომლის ხარისხიც უმჯობესდება მომხმარებელთა რაოდენობის ზრდასთან ერთად.

WEB2.0-ის სახეებია *მულტიმედია-პორტალები* (youtube, myvideo.ge…), *სოციალური ქსელები* (Facebook, ok.ru…), *ბლოგოსფერო* (blogger.com, wordpress.com…), *ვიკი-პორტალები* (wikipedia...) და სხვა მრავალი.

ვებ-ტექნოლოგიებზე მომუშავე მრავალრიცხოვან სპეციალისტთა ჯგუფები (პირველ რიგში *W3C - ვების განვითარების კონსორციუმი*) მიღწეულს არ სჯერდებიან და მსოფლიო აბლაბუდის ახალი შესაძლებლობებით აღჭურვისთვის იღწვიან. კერძოდ, ე.წ. *სემანტიკური ვების* კონცეფციით, რომელიც ჯერ უფრო თეორიულ დონეზეა დამუშავებული, ვებ-სივრცემ მომხმარებელს სემანტიკური სერვისები უნდა შესთავაზოს, ანუ უბრალოდ თუ ვიტყვით, მის პერსონალურ ასისტენტად უნდა იქცეს.

მაგალითად, სემანტიკური ვები მიიღებს მომხმარებლისგან მოთხოვნას: *„სად შევიძინო ყველაზე იაფად ავტომანქანა Toyota, ჩემგან 50 კილომეტრის რადიუსში, 2002-დან 2008 წლამდე გამოშვების, თეთრი ან ბორდოსფერი, 7-10 ლიტრი ბენზინის ხარჯით 100 კილომეტრზე, ტყავის სალონით, კონდიციონერით, მარცხენა საჭიანი?“* და გასცემს მას საუკეთესო პასუხს.

მსოფლიო აბლაბუდის ჩონჩხი, მისი არქიტექტურა უცვლელია შექმნის დღიდან და შემდეგი კომპონენტებისგან შედგება:

- **ჰიპერტექსტური მონიშვნების ენა HTML** (Hypertext Markup Language) – ვებ-კონტენტის შენახვის ფორმატი;
- **პროტოკოლი http** (Hyper Text Transport Protocol) – ქსელში ინფორმაციის ტრანსპორტირების საშუალება;
- **ვებ-სერვერი** (Apache, IIS, lightppd, nginx…) – ვებ-კონტენტის საცავი (HTML-ფაილები, ფოლდერები, ვებ-აპლიკაციები);
- **ვებ-ბრაუზერი (**Firefox, Internet Explorer/ Microsoft Edge, Opera, Chrome, Safari…**)** – ვებ-კონტენტთან მიმართვის და ინფორმაციის მიღების საშუალება**;**
- **ინტერნეტ-პროგრამირების ენები** (JavaScript, PHP, ASP.NET…) - მსოფლიო ქსელისათვის პროგრამული პროდუქტების შექმნის ინსტრუმენტები.

შევეხოთ თითოეულს უფრო დეტალურად.

HTML-ენა ვებსივრცეში ინფორმაციის წარმოდგენის სტანდარტული ფორმატია, რომელიც 1990 წლიდან მუდმივ მოდერნიზაციას და სტანდარტიზაციას განიცდის. ენის ბოლო ვერსია, HTML5 სხვა ფუნქციებთან ერთად მულტიმედიური ინფორმაციის (ვიდეო, აუდიო, რუქები და სხვა) გამოტანის საშუალებებსაც შეიცავს. HTML-დოკუმენტი არის მონიშვნის სპეციალური საშუალებების (ტეგების) იერარქიულად დალაგებული ნაკრები, რომლის წაკითხვისა და ფორმატირებულად ასახვისათვის სპეციალური პროგრამა, ვებ-ბრაუზერი გამოიყენება.

აღსანიშნავია, რომ თანამედროვე ვებდოკუმენტებში *კონტენტი* (შინაარსი) და *ფორმატირება* (გარეგნული სახე) ურთიერთ-გამიჯნულია. კონტენტი უშუალოდ HTML-დოკუმენტში ინახება, ხოლო ფორმატირების ელემენტები, როგორც წესი, ე.წ. ჩადგმულ CSS (Cascade Style Sheet) ფაილებშია აღწერილი. ქვემოთ მოცემულია HTML და მასთან დაკავშირებული CSS-ფაილების ფრაგმენტები:

CSS-ფაილი **test.css**

```
body {background-color: Yellow;}
        h1 {
        text-align: center;
        color: blue;}
```

HTML-ფაილი ჩადგმული CSS-ით

```
<!DOCTYPE html>
<html> <head>
  <title>Main HTML-Document</title>
  <link rel="stylesheet" type="text/css" href="B:|Web|test.css">
</head>
<body>
<h1>ქართული ფილმების არქივი</h1>
<table border="5">
```

```
<tr>    <th>ფილმის დასახელება</th>
        <th>რეჟისორი</th>    </tr>
<tr>    <td style="color:red">დარიკო</td>
        <td>სიკო დოლიძე</td> </tr>
<tr>    <td>ცისფერი მთები</td>
        <td>ელდარ შენგელაია</td></tr>
</table> </body> </html>
```

ვებკონტენტის, ანუ *ვებგვერდების* შესაქმნელად მრავალი პროგრამა არსებობს. სწავლების პროცესში უბრალო ტექსტურ რედაქტორებს მიმართავენ (მაგალითად, Notepad++), ხოლო რეალური ვებსაიტების შესაქმნელად იყენებენ ისეთ პროფესიონალურ პროდუქტებს, როგორიცაა Adobe Dreamweaver, Microsoft Expression Web, CoffeeCup HTML Editor და სხვ.

**http-პროტოკოლი** არის *ვებბრაუზერსა* და *ვებსერვერს* შორის კავშირის საშუალება, რომელიც საამისოდ ბრძანებათა საკუთარ ნაკრებს შეიცავს, ხოლო ვებრესურსის ადგილსამყოფელის დასადგენად იყენებს უნიფიცირებულ რესურსთა ლოკატორს (URL). URL-მისამართი რამდენიმე ელემენტის კომბინაციისგან შედგება. ესენია: *პროტოკოლი, ჰოსტის DNS-სახელი* ან *IP-მისამართი, პორტის ნომერი, ვებ-ფოლდერი*. მოვიყვანოთ ამგვარად მიღებული URL-ის ნიმუში:

http://lockss.cms.hu-berlin.de:8081/BatchAuConfig

**ვებ-სერვერი** მსოფლიო ქსელის მონაცემთა განთავსების და მართვის ამოცანას ემსახურება. იგი სერვერული ოპერაციული სისტემის გაფართოება ან დამოუკიდებელი პროგრამული უზრუნველყოფაა და შექმნილია პრაქტიკულად ყველა გავრცელებული ოპერაციული სისტემისთვის. ყველაზე გავრცელებული ვებსერვერებიდან შეიძლება დავასახელოთ Apache, IIS (Internet Information Services), Nginx, Google Web Server... .

**ვებ-ბრაუზერი** ვებსერვერიდან გამოძახებული კონტენტის ფორმატირებული ასახვის (გამოტანის) საშუალებაა და შექმნილია ყველა პოპულარული ოპერაციული სისტემისთვის (Windows, MAC OS X, LINUX…). ბრაუზერებს გააჩნია ინფორმაციის ქეშირების საშუალება მონაცემებთან მიმართვის დასაჩქარებლად. ისტორიულად პირველი ვებბრაუზერებიდან ფართოდ გავრცელდა Lynx (რომელიც ტექსტურ, ანუ კონსოლის რეჟიმში მუშაობდა), Mosaic და Netscape Navigator. სადღეისოდ ყველაზე გავრცელებული ვებბრაუზერებია Google Chrome, Mozilla Firefox (და მისი კლონი Debian LINUX-ისთვის Iceweasel), MS Edge, Internet Explorer, Opera, Safari...

**დაპროგრამების ინსტრუმენტების** გამოყენებით ვებინფრასტრუქტურის დინამიკური მართვა ხორციელდება. ვებპროგრამირების ენები ორ ძირითად ჯგუფად იყოფა:

- **სერვერული ვებპროგრამები** (Server-side Scripting Languages) – სრულდება ვებსერვერის მხარეს და მიზნად ისახავს დინამიკური HTML-კონტენტის გენერაციას. მომხმარებელი გზავნის მოთხოვნას ბრაუზერიდან ვებსერვერისაკენ, რომელიც სერვერის მხარეს მუშავდება (მაგალითად, ამოკრეფს ინფორმაციას მონაცემთა ბაზებიდან) და საბოლოო სახით (გენერირებული HTML-დოკუმენტი) უბრუნდება მომხმარებელს. სერვერული ვებპროგრამების კოდი ბრაუზერში უხილავია. სერვერული ვებპროგრამირების ენებიდან შეიძლება დავასახელოთ: Perl, PHP, ASP.NET, JAVA-Servlet, XML, Python, Ruby;
- **კლიენტის ვებპროგრამები** (Client-side Scripting) – სრულდება ვებბრაუზერში და მიზნად ისახავს HTML-კოდზე მოქმედებების შესრულებას (ფორმის ან შინაარსის შეცვლა) ან მომხმარებლის მიერ განხორციელებულ ქმედებათა (მაგალითად, ღილაკზე მაუსის დაკლიკვა) დამუშავებას. ისინი ინტეგრირებულია ვებბრაუზერებში ანუ წაკითხვადია კლიენტის მხარეს (Browser source code), რაც მათი

გამოყენების უსაფრთხოებას ამცირებს. კლიენტის ვებპროგრამირების ცნობილი ენებია: **JavaScript**, **VBScript, JAVA-Applet...**

ზემოაღწერილი კომპონენტების საფუძველზე აიგება ორგანიზაციის ვებინფრასტრუქტურა (ვებაპლიკაციების ფრეიმვორკი). ცნობილია ოპერაციული სისტემების, მონაცემთა ბაზების მართვის სისტემების, ვებსერვერების და ვებდაპროგრამების სისტემების რამდენიმე კომბინაცია, რომლებიც ფართოდაა ბაზარზე გავრცელებული. მოვიყვანოთ მაგალითები:

- Windows, Internet Information Server, MS SQL Server, ASP.NET;
- LINUX, Apache Server, MySQL, PHP (LAMP-სერვერი);
- Windows, Apache, MySQL, PHP (WAMP-სერვერი);
- IBM WebSphere – AIX, Java, XML, Web-Services;

და სხვ.

თანამედროვე ვებსივრცეში ფართოდ გავრცელდა *კონტენტის მართვისა* (CMS – Content Management Systems) და მის ბაზაზე სხვადასხვა საპრობლემო სფეროს (დისტანციური სწავლება, ელექტრონული კომერცია, ონლაინ ბიბლიოთეკა და სხვ.) მართვის სისტემები, რომლებიც წარმოადგენს მეტწილად უფასო, „ღია კოდზე“ (OpenSource) მომუშავე ვებაპლიკაციებს, წინასწარ მომზადებული ფუნქციების ნაკრებით, რაც ვებდეველოპმენტისათვის დასახარჯ დროს, ხშირ შემთხვევაში, მნიშვნელოვნად ამცირებს. ქვემოთ ამგვარი სისტემების მაგალითებია მოყვანილი:

- *სტანდარტული ვებკონტენტის მართვის სისტემაა:* Joomla, Drupal, Wordpress და სხვ.;
- *სასწავლო პროცესის მართვის* სისტემებიდან (LMS – Learning Management Systems) შეიძლება დავასახელოთ: Moodle და ILIAS;
- *ბიბლიოთეკების მართვის* (LMS – Library Management Systems) ყველაზე ცნობილი ელექტრონული სისტემებია: Evergreen და Koha;

- *ელექტრონული კომერციის* (E-commerce) სფეროშიც (ონლაინ-მაღაზიები, ონლაინ-აუქციონები) მრავალი პროგრამა მუშაობს. მაგალითისათვის PrestaShop პროგრამა შეიძლება დავასახელოთ.

ერთ-ერთ ქვეთავში ჩვენ შევეხებით სრულფუნქციური, მზა ვებპორტალის უმოკლეს დროში მიღების პროცედურას MsSharePoint სისტემის მაგალითზე, რომლის ერთ-ერთი ფუნქციაც კონტენტის მართვაა.

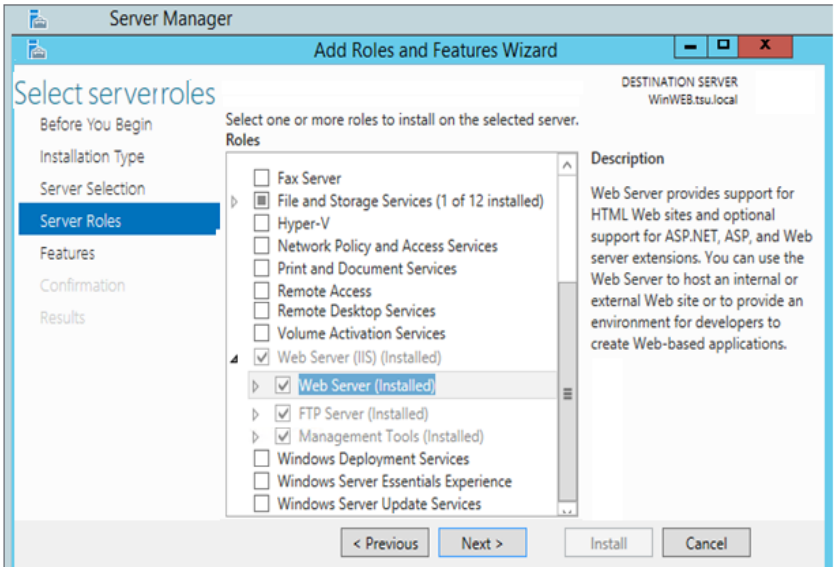
ყურადღების გარეშე ვერ დავტოვებთ **ვებ-ჰოსტინგის მართვის სისტემებსაც,** რომელთა დანერგვა პირველ რიგში აუცილებელია *ვებ-ჰოსტინგ-პროვაიდერებისთვის,* აგრეთვე ისეთი ორგანიზაციებისთვის, რომელთა ვებ-ინფრასტრუქტურა ათობით და ასობით ვებ-გვერდისგან შედგება. ვებ-ინფრასტრუქტურის მართვის ავტომატიზაციის ისეთი პროგრამები, როგორიცაა Cpanel, ISPConfig და სხვები მნიშვნელოვნად ამცირებს სისტემის ადმინისტრატორის დროით დანახარჯებს ვებ-ინფრასტრუქტურის მართვის პროცესში.

### 3.3.1.2. ვებ-სერვერის აქტივაცია და მართვა Internet Information Server-ის გარემოში

პროგრამა Internet Information Server (IIS) არის Windows Server ოპერაციული სისტემის ერთ-ერთი **როლი**, ვებგვერდების შექმნის, შენახვის და მართვის ინსტრუმენტებით. მისი აქტივაცია Server Manager-პროგრამის Add Roles and Features ფუნქციითაა შესაძლებელი (ნახ.3.13).

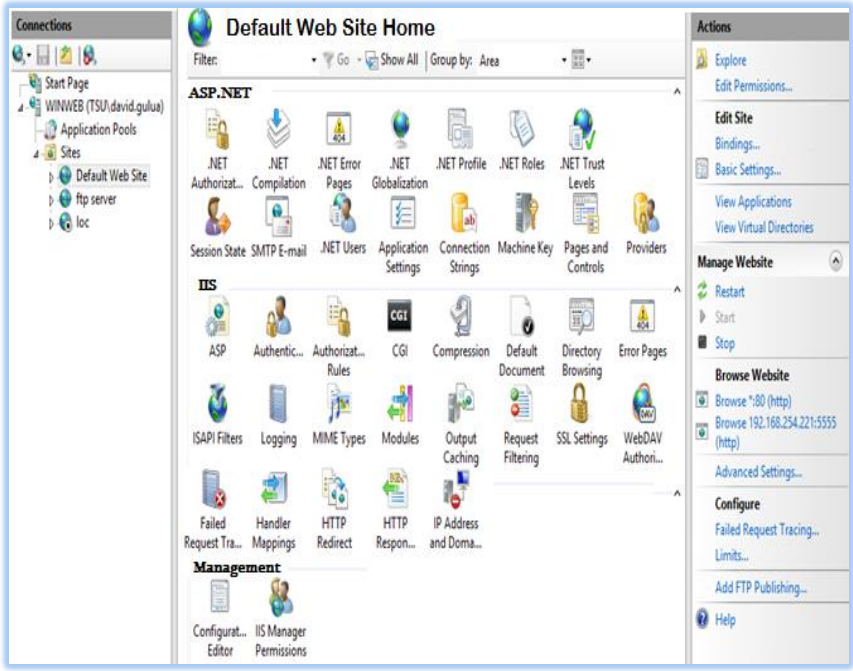
აქტივაციის წარმატებული შესრულების ნიშანია ვებბრაუზერში ვებსერვერის გვერდის გამოტანა, რომელიც გავრცელებულ ენებზე ესალმება მომხმარებელს (ნახ.3.14).

უშუალოდ ვებსერვერზე Internet Information Services (IIS) Manager პროგრამის გაშვებით ვებადმინისტრატორს შეუძლია შეუდგეს ვებსაიტებისა და ვებაპლიკაციების სერვერზე ატვირთვას და მართვას (ნახ.3.15).




ნახ.3.13. IIS-ის აქტივაცია Windows 2012 Server-ის გარემოში


ნახ.3.14. IIS-სერვერის საწყისი გვერდი ვებ-ბრაუზერში



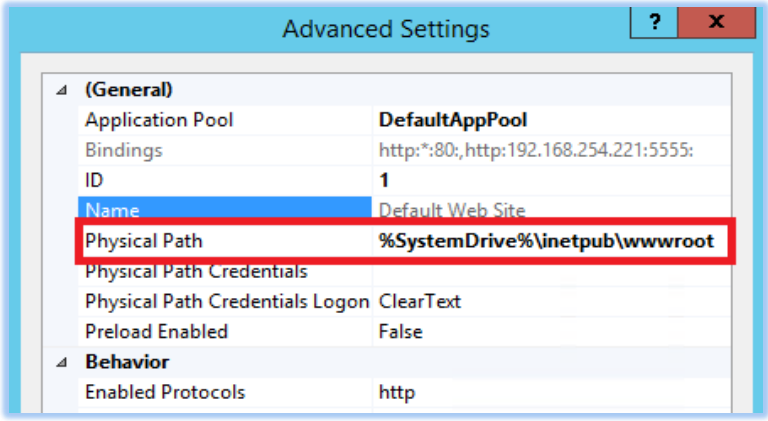

ნახ.3.15. IIS-ის მართვის ინტერფეისი

წარმოდგენილი ინტერფეისის ყველა ფუნქციის განხილვა მოცემული წიგნის ფარგლებს სცილდება. გამოვყოფთ რამდენიმე ძირითად მომენტს:

ვებგვერდების ნაგულისხმევ (Default) საცავს სერვერის ფაილურ სისტემაში ფოლდერი %System Drive%\inetpub\wwwroot წარმოადგენს. ვებგვერდების და ვებაპლიკაციების ლოკაციის შესაცვლელად გამოიყენება ბრძანება:

*ვებ-გვერდზე* ***მაუსის მარჯვენა ღილაკი -> Manage Website -> Advanced Settings -> პარამეტრი Physical Path***

3.16 ნახაზზე მოცემულია ეს ფანჯარა.




ნახ.3.16. ვებგვერდის ადგილმდებარეობის შეცვლა IIS-ში

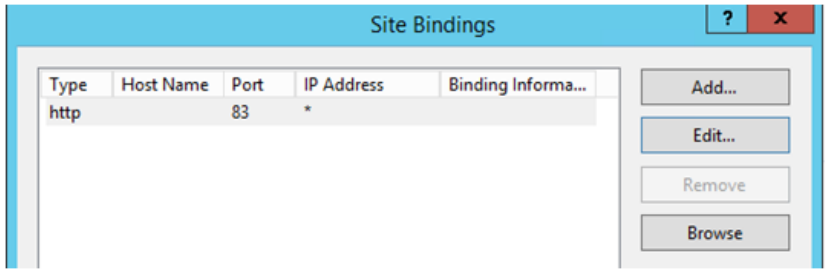
ვებ-რესურსთან მიმართვის აუცილებელ წინაპირობას ვებ-სერვერზე თავისუფალი პორტის არსებობა წარმოადგენს. http-პროტოკოლი სტანდარტულად მე-80 პორტს იყენებს, ანუ URL-ში პორტის ნომრის არარსებობა მე-80 პორტის გამოყენებაზე მიუთითებს. ორ ვებ-აპლიკაციას საერთო პორტის გამოყენება არ შეუძლია. კონფლიქტის აღმოსაფხვრელად ორი გზა არსებობს:

- ერთ-ერთი ვებ-აპლიკაციის გადამისამართება *ალტერნატიულ* (მაგალითად, 81-ე, 82-ე და ა.შ.) პორტზე;
- ვებაპლიკაციისათვის *ვირტუალური ჰოსტის* სახელის (Host Name) განსაზღვრა და იმავე (მე-80) პორტის პარალელური გამოყენება.

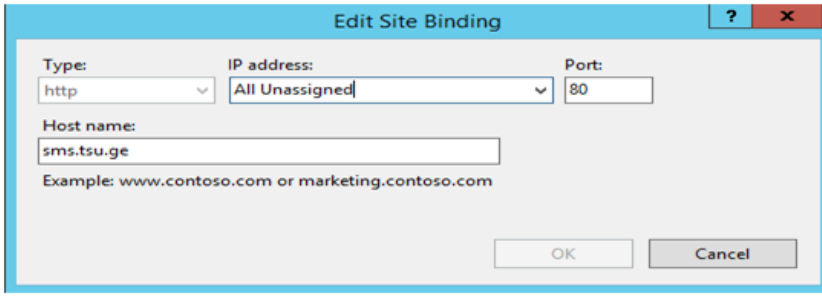
ორივე შემთხვევაში გამოიყენება ბრძანება:

*ვებ-გვერდზე* ***მაუსის მარჯვენა ღილაკი -> Edit Bindings***

ბრძანების ინტერფეისის ნიმუშები მოცემულია 3.17 და 3.18 ნახაზებზე.




ნახ.3.17. ვებ-აპლიკაციისთვის ალტერნატიული პორტის არჩევა




ნახ.3.18. ვირტუალური ჰოსტის სახელის განსაზღვრა

### საკონტროლო დავალებები:

1. შეასრულეთ ვებ-სერვერის როლის აქტივაცია Windows Server 2012-ის გარემოში. გამოიტანეთ IIS-ის საწყისი ფანჯარა ბრაუზერში;
2. შექმენით ახალი ფოლდერი IIS-ში. შექმენით ახალი HTML-ფაილი და დააკოპირეთ ახალ ფოლდერში. ბრაუზერის გამოყენებით დარწმუნდით, რომ ფაილი ხელმისაწვდომია.

## 3.3.2. ფაილური, ბეჭდვის და სკანირების სერვისის მართვა

### 3.3.2.1. ფაილური სერვისის ძირითადი ფუნქციები

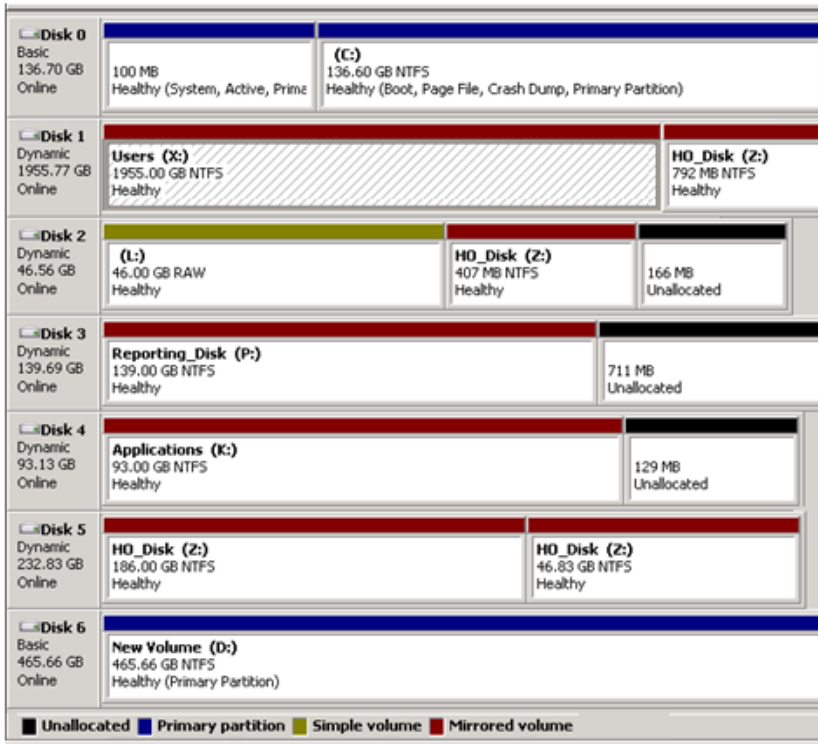
**ფაილური სერვისი** ორგანიზაციის ინფორმაციული ინფრასტრუქტურის საბაზო კომპონენტია. მისი დანიშნულება ფაილური რესურსების (ფოლდერები, ტექსტური და მულტიმედია ფაილები) საიმედოდ შენახვა, მართვა და მოხმარებლისთვის მიწოდებაა.

სადღეისოდ, Windows-სისტემებში ფაილური სერვისი შემდეგ ძირითად ამოცანებს განაგებს:

- ხელმისაწვდომი *მეხსიერების კვოტების* დადგენა მომხმარებლებისათვის;
- *ფაილების ინდექსირება* (Index Service) ძებნის ოპერაციათა დასაჩქარებლად;
- საერთო *ქსელური დისკოების* მართვა;
- *მომხმარებელთა პროფილების* მართვა;
- საერთო ფაილური რესურსების გამოყენების *მონიტორინგი* და *ფილტრაცია.*

ორგანიზაციის ფაილურ სერვისს **ფაილ-სერვერი** ემსახურება. ოპერაციული სისტემა Windows Server-ის ფაილური სერვისის გასააქტიურებლად (ფაილ-სერვერის შესაქმნელად) აუცილებელია შესაბამისი სერვერული როლის გააქტიურება, რომელსაც File and Storage Services ეწოდება და რამდენიმე ქვეროლს მოიცავს (ზოგიერთ მათგანს ქვემოთ შევეხებით).

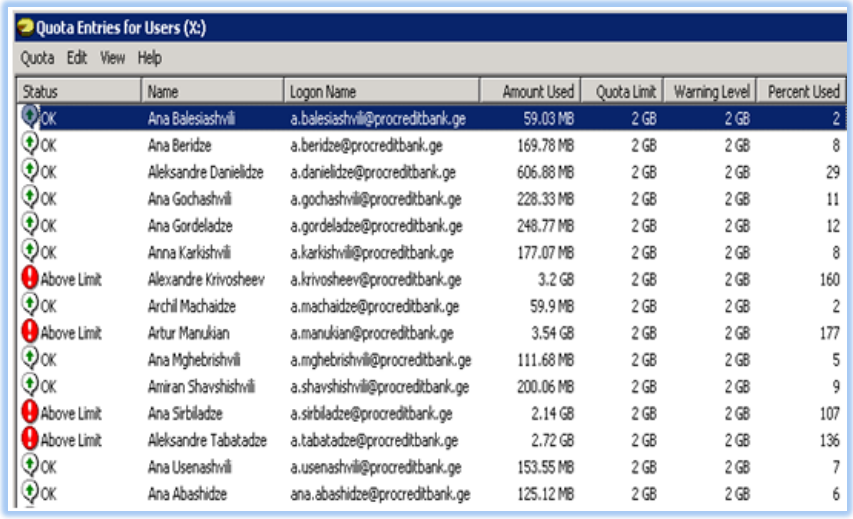
ფაილ-სერვერის მთავარი მახასიათებელია გარე მეხსიერების დიდი მოცულობა. ოპერაციული სისტემა Windows Server შეიცავს როგორც ლოკალური და ქსელური, ასევე **მონაცემთა საცავების ლოგიკური ტომების** (Data Storage Volumes) ინტეგრირების საშუალებებს, ისე, რომ სისტემური აპლეტის Disk Management ფანჯარაში ხშირად უამრავი დისკის დაფიქსირებაა შესაძლებელი (ნახ.3.19).




ნახ.3.19. ლოგიკური დისკები Windows Disk Management-ში

თითოეულ ლოგიკურ დისკოზე ფაილური სერვისის რესურსების მომხმარებლებზე განაწილება **კვოტირების ფუნქციის** თანხლებით სრულდება, რომელიც Windows ოპერაციული სისტემის როგორც სერვერულ, ისე დესკტოპ-ვერსიებშია და ***დისკოზე მაუსის მარჯვენა ღილაკის*** *დაკლიკვით,* ***Properties-ბრძანების*** *არჩევითა და* ***გადამრთველ Quota****-ზე გადასვლით შეიძლება იქნეს გამოძახებული (ნახ.3.20).*

აქ ჩანს, რომ ყოველი მომხმარებლისათვის 2GB ფაილური რესურსია გამოყოფილი, თუმცა ორგანიზაციის ზოგიერთი დეპარტამენტი (მაგალითად, ბუღალტერია, რეპორტინგი), მუშაობის სპეციფიკიდან გამომდინარე, შეიძლება ცალკე დისკურ რესურსებს და დამატებით გარე მეხსიერებასაც ფლობდეს.

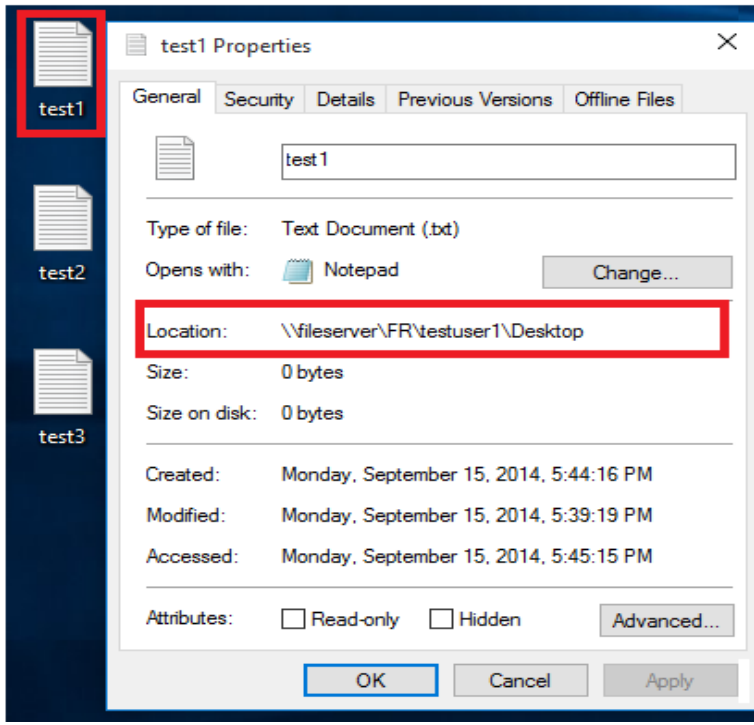



| Status | Name | Logon Name | Amount Used | Quota Limit | Warning Level | Percent Used |
|---|---|---|---|---|---|---|
| OK | Ana Balesiashvili | a.balesiashvili@procreditbank.ge | 59.03 MB | 2 GB | 2 GB | 2 |
| OK | Ana Beridze | a.beridze@procreditbank.ge | 169.78 MB | 2 GB | 2 GB | 8 |
| OK | Aleksandre Danielidze | a.danielidze@procreditbank.ge | 606.88 MB | 2 GB | 2 GB | 29 |
| OK | Ana Gochashvili | a.gochashvili@procreditbank.ge | 228.33 MB | 2 GB | 2 GB | 11 |
| OK | Ana Gordeladze | a.gordeladze@procreditbank.ge | 248.77 MB | 2 GB | 2 GB | 12 |
| OK | Anna Karkishvili | a.karkishvili@procreditbank.ge | 177.07 MB | 2 GB | 2 GB | 8 |
| Above Limit | Alexandre Krivosheev | a.krivosheev@procreditbank.ge | 3.2 GB | 2 GB | 2 GB | 160 |
| OK | Archil Machaidze | a.machaidze@procreditbank.ge | 59.9 MB | 2 GB | 2 GB | 2 |
| Above Limit | Artur Manukian | a.manukian@procreditbank.ge | 3.54 GB | 2 GB | 2 GB | 177 |
| OK | Ana Mghebrishvili | a.mghebrishvili@procreditbank.ge | 111.68 MB | 2 GB | 2 GB | 5 |
| OK | Amiran Shavshishvili | a.shavshishvili@procreditbank.ge | 200.06 MB | 2 GB | 2 GB | 9 |
| Above Limit | Ana Sirbiladze | a.sirbiladze@procreditbank.ge | 2.14 GB | 2 GB | 2 GB | 107 |
| Above Limit | Aleksandre Tabatadze | a.tabatadze@procreditbank.ge | 2.72 GB | 2 GB | 2 GB | 136 |
| OK | Ana Usenashvili | a.usenashvili@procreditbank.ge | 153.55 MB | 2 GB | 2 GB | 7 |
| OK | Ana Abashidze | ana.abashidze@procreditbank.ge | 125.12 MB | 2 GB | 2 GB | 6 |

**3.20. საერთო დისკის კვოტირების ინტერფეისი**

**საერთო ქსელური დისკების** მართვის ამოცანა გულისხმობს არსებული საშუალებების გათვალისწინებით მომხმარებლების ან მათი ჯგუფებისთვის გარკვეული ოდენობის მეხსიერების გამოყოფას ე.წ. *ქსელური დისკის* სახით.

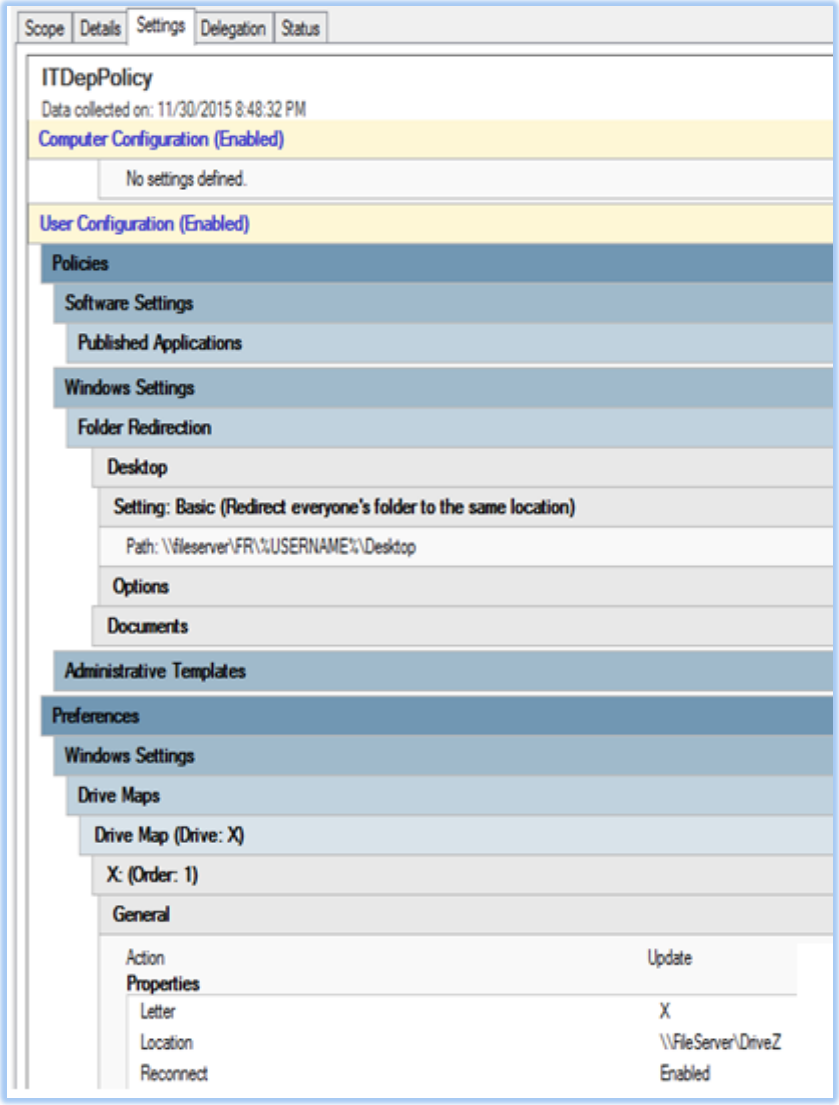
**მომხმარებელთა პროფილების** მართვის ამოცანაა მათ პერსონალურ კომპიუტერებზე დისკური მეხსიერების იმ უბნების საიმედო შენახვა, რომლებშიც ისინი ყველაზე მნიშვნელოვან ინფორმაციას განათავსებენ. იგულისხმება ოპერაციული სისტემის სამუშაო მაგიდა (დესკტოპი), დოკუმენტების, სურათების, აუდიო და ვიდეოფოლდერები და სხვ. **კატალოგების გადამისამართების** (Folder Redirection) ინსტრუმენტი მომხმარებელს უქმნის ილუზიას, რომ ყველა საჭირო ფაილი მის პერსონალურ კომპიუტერშია განთავსებული, მაშინ, როცა დესკტოპების, დოკუმენტების და სხვა პერსონალური ფოლდერების რეალური ადგილსამყოფელი ფაილსერვერია (ნახ.3.21).




**ნახ.3.21. კატალოგების გადამისამართების (Folder Redirection) ასახვა Windows 10-ის დესკტოპზე**

ზემოთ განხილულ ორივე შემთხვევაში ფაილური რესურსების მომხმარებელთა განკარგულებაში გადასაცემად აღარ არის საჭირო სპეციალური სკრიპტების წერა და ყოველ ლოკალურ ოპერაციულ სისტემაში მათი ავტომატურად გაშვება.

გამოიყენება ბევრად უფრო ადვილი ხერხი - დომენური ინფრასტრუქტურის (Active Directory) *ჯგუფური პოლიტიკა* (Group Policy), რომლის ფუნქციებიც ერთერთ წინა თავშია აღწერილი. ამჯერად წარმოვადგინოთ ჯგუფური პოლიტიკის ნიმუში *კატალოგების გადამისამართებისათვის* (ნახ.3.22).




3.22. ჯგუფური პოლიტიკის (Group policy) ფრაგმენტი კატალოგების გადამისამართებისთვის

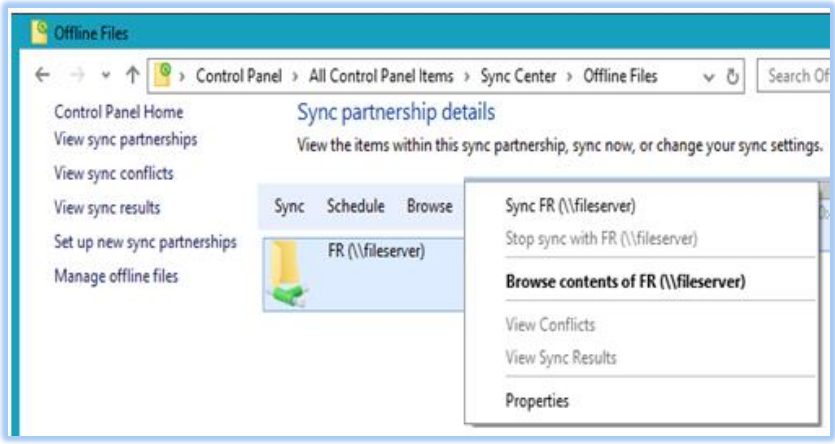
ფაილური სერვისის გამოყენება მომხმარებელს **ონლაინ** ან **ოფლაინ-რეჟიმებში** შეუძლია. ეს უკანასკნელი საშუალებას იძლევა მივმართოთ ქსელურ რესურსებს გათიშული ქსელის პირობებში, რისთვისაც კლიენტ-მანქანის კეშ-მეხსიერებაში შენახული ინფორმაცია გამოიყენება.

**ფაილური კეში** (Client Side Cash) ქსელური რესურსის (მაგალითად, ფოლდერი) იერარქიულ სტრუქტურას ინახავს და კლიენტ-კომპიუტერზე განთავსებულია **%SystemRoot%\CSC** ფოლდერში. ფაილ-სერვერთან კავშირის აღდგენის შემდეგ კეშირებული ინფორმაცია სერვერთან მონაცემების სინქრონიზაციას შეასრულებს.

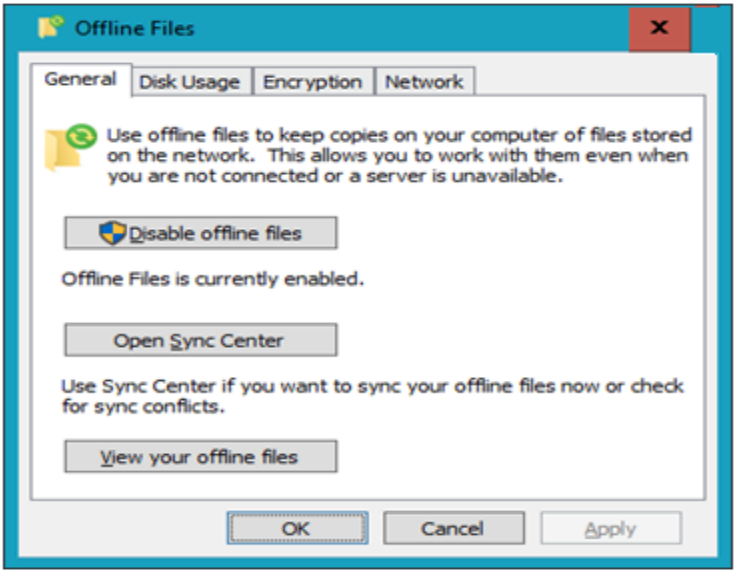
ოფლაინ-რეჟიმში მუშაობის საშუალებას Windows-ის უტილიტა Sync Center იძლევა, რომელსაც რამდენიმე ფუნქცია აქვს:

- ოფლაინ-კეშის აქტივაცია;
- სინქრონიზაციის გრაფიკის შედგენა;
- ოფლაინ-მეხსიერების ქვოტირება საერთო ლოკალური მეხსიერების ფარგლებში.

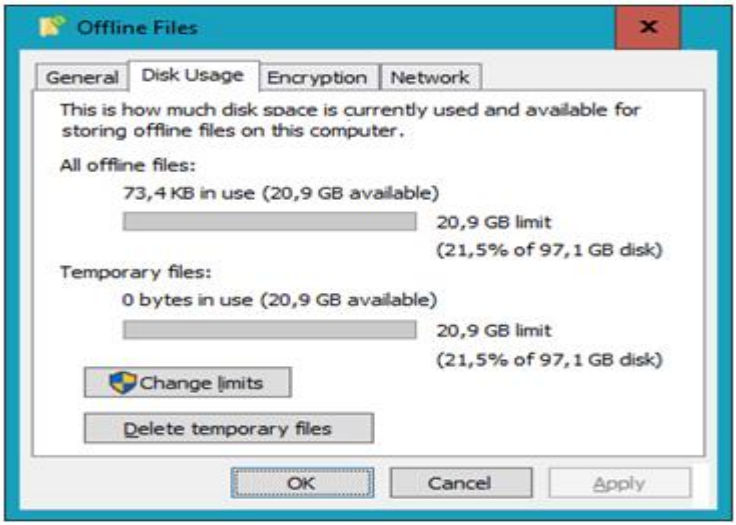
და სხვა (ნახ.3.23 - 3.25).




**ნახ.3.23. კლიენტსა და ფაილ-სერვერს შორის მონაცემთა სინქრონიზაციის ინტერფეისი**



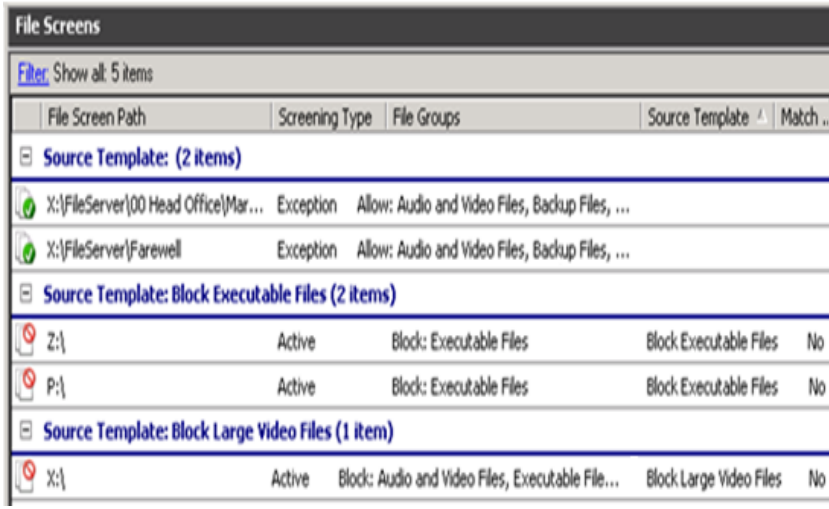

ნახ.3.24. ოფლაინ-რეჟიმის მართვა




ნახ.3.25. ოფლაინ-რეჟიმის ფაილური ქვოტის მართვა

უშუალოდ ფაილ-სერვერის მხარეს, ფაილური სერვისის სამართავად გამოიყენება პროგრამა File Server Resource Manager (სისტემური აპლეტი fsrm.msc). მისი დახმარებით ფაილური სერვისის ოპტიმალურ მდგომარეობაში შენარჩუნება შეიძლება, ისე, რომ ერთი მხრივ, მომხმარებლებმა საჭირო ფაილური რესურსების ნაკლებობა არ იგრძნონ, ხოლო მეორე მხრივ, მეხსიერება საკმარისად გაიცეს და ყოველგვარი ზედმეტი კონტენტისაგან თავისუფალი იყოს.

3.26 ნახაზზე მოცემულია ერთ-ერთი ინტერფეისი სახელით File Screens, რომელიც ფაილ-სერვერის სხვადასხვა დისკოებზე სხვადასხვა ტიპის ფაილების დაშვება/შეზღუდვის პოლიტიკას განსაზღვრავს. აქ ფაილ-სერვერის Z: და P:-დისკოებზე აკრძალულია შესრულებადი (.exe, .com) ფაილების განთავსება, ხოლო X:-დისკოზე, ორიოდე გამონაკლისის გარდა (განყოფილება allow) შეუძლებელია აუდიო, ვიდეო და შესრულებადი ფაილების კოპირება.

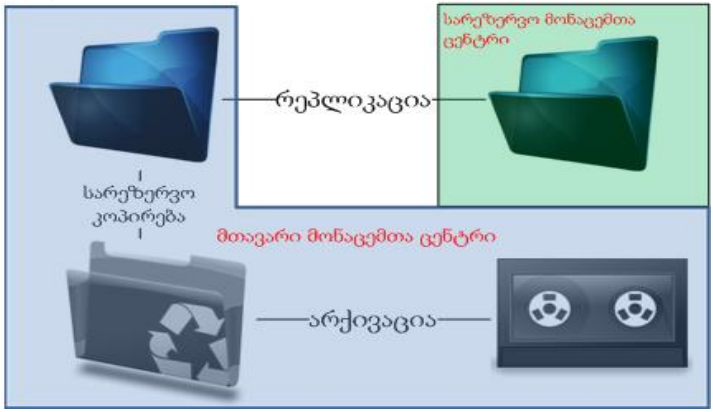



ნახ.3.26. ფაილ-სერვერის დისკების მართვის ინტერფეისის ფრაგმენტი

### 3.3.2.2. ფაილური სერვისის სარგებლიანობის და საიმედობის უზრუნველყოფა

ზოგადად, ინფორმაციის სარგებლიანობასა და საიმედოობაში მისთვის *წვდომის უწყვეტობის* უზრუნველყოფას და სერვისის *უმოქმედობის დროის მინიმიზაციას* გულისხმობენ. *ფაილური სერვისის* სარგებლიანობის ასამაღლებლად შემდეგი მეთოდები გამოიყენება (ნახ.3.27):

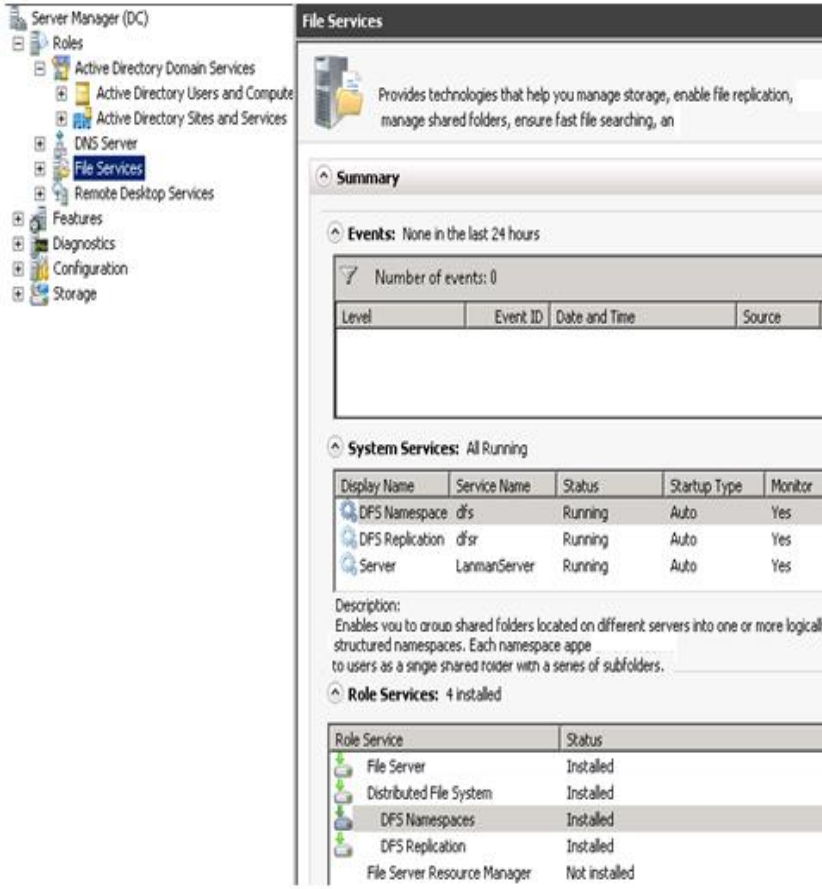
- სარეზერვო კოპირება და არქივაცია;
- განაწილებული ფაილური სისტემა (DFS) და ფაილთა ტომების რეპლიკაცია.



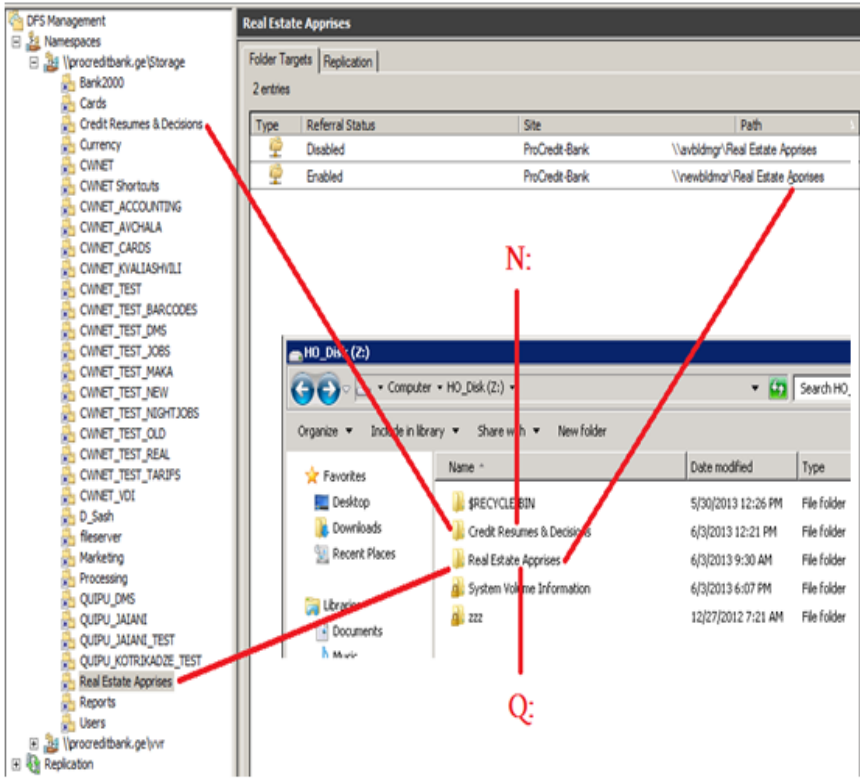

**ნახ.3.27. რეპლიკაციის, სარეზერვო კოპირების და არქივაციის საწიმუშო სქემა**

ფაილური და სხვა რესურსების სარეზერვო კოპირების და არქივაციის თემა ცალკე პარაგრაფში განიხილება. *განაწილებული ფაილური სისტემის* ინსტრუმენტებით (Windows 2008 Distributed File System – DFS) სხვადასხვა ადგილებზე განთავსებული (მაგალითად, ორ დამოუკიდებელ ფაილ-სერვერზე) ფაილური რესურსებისაგან დამოუკიდებელი *ვირტუალური ფაილური სივრცეა* შექმნილი

(Namespace მექანიზმის საშუალებით), რომელთანაც მომუშავე მომხმარებელს „არ აინტერებს“ ფოლდერების და ფაილების ფიზიკური მდებარეობა და აგრძელებს საკუთარ რესურსებთან მიმართვას მაშინაც, როცა რომელიმე სერვერი მწყობრიდან გამოდის (ნახ.3-28,29).




**ნახ.3.28. DFS-ფუნქციის აქტივაცია Windows Server-ში**

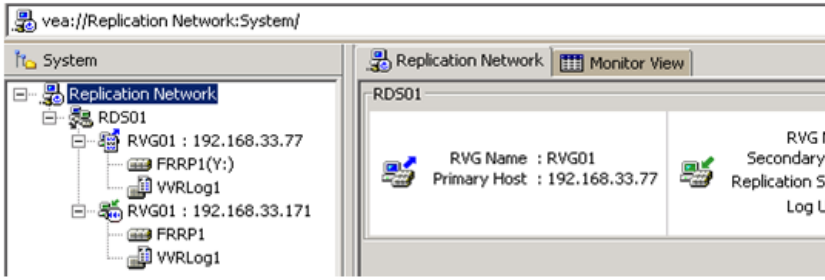



ნახ.3.29. DFS-სერვისის მართვის ინტერფეისი

**ფაილური რეპლიკაციის** დამსახურებით კრიტიკული ინფორმაციული მასივები, კერძოდ მომხმარებელთა პროფილები და საერთო სარგებლობის დისკოები, მუდმივად მინიმუმ ორი სინქრონული ასლის სახით ინახება (მაგალითად, ცენტრალურ და სარეზერვო მონაცემთა ცენტრებში). ერთ-ერთი მხარის გათიშვისას ავტომატურად ფაილთა მეორე რეპლიკა ირთვება, რაც ფაილურ რესურსებთან შეფერხებელ მიმართვას უზრუნველყოფს.

IT-ინფრასტრუქტურაში რეპლიკაციისათვის სპეციალური პროგრამული უზრუნველყოფა გამოიყენება (მაგალითად, Symantec Veritas Volume Replication), რომელიც DFS-ის რეპლიკაციის მოდულისაგან განსხვავებით მუშაობს არა ფაილური სისტემის, არამედ უფრო საიმედო, *მონაცემთა ტომების* და *ფიზიკური დისკოების* დონეზე.

რეპლიკაციის საშუალებით მომხმარებელთა მონაცემების აქტუალური ასლები სინქრონულად მრავლდება, მაგალითად, ცენტრალურ და სარეზერვო მონაცემთა ცენტრებს შორის (ნახ.3.30).




**ნახ.3.30. რეპლიკაცის ნიმუში: 192.168.33.77-დან 192.168.33.171-ისკენ**

DFS და VVR სისტემების ერთიანი მოქმედება ფაილური სერვისის სარგებლიანობას შესამჩნევად ამაღლებს, თუმცა ოპერაციული სისტემის ფარგლებში სარეპლიკაციო ოპერაციების განხორციელება (Application Replication) ტვირთავს სერვერის რესურსებს და შედარებით არაეფექტურია. მეტი სარეპლიკაციო ეფექტი მიიღება **აპარატული რეპლიკაციის** (Block Replication) განხორციელებით, რომელიც სრულდება უშუალოდ მონაცემთა საცავებს შორის (მწარმოებლები: HP, IBM, EMC, NetApp და სხვ.) და ბევრად სწრაფია.

### 3.4. საკონტროლო დავალებები:

1. შეასრულეთ ფაილური სერვისის როლის აქტივაცია Windows Server-ში;

2. შეასრულეთ ფაილურ სერვერზე ერთერთი ლოგიკური დისკის კვოტირება. გაუწერეთ საერთო დისკური კვოტა ყველა მომხმარებელს. გაუზარდეთ კვოტა 2-3 მომხმარებელს;

3. Active Directory Group Policy-ს გარემოში შექმენით ახალი ჯგუფური პოლიტიკა, რომელიც დომენის მომხმარებლებს უზრუნველყოფს ქსელური X:-დისკით. შეამოწმეთ პოლიტიკის მოქმედება მომხმარებლის კომპიუტერზე;

4. Active Directory Group Policy-ს გარემოში შექმენით ახალი ჯგუფური პოლიტიკა, რომელიც დომენის მომხმარებლებს უზრუნველყოფს კატალოგის გადამისამართების (Folder Redirection) ფუნქციით. გადამისამართების ობიექტებად აირჩიეთ დესკტოპი და დოკუმენტების ფოლდერი. შეამოწმეთ პოლიტიკის მოქმედება მომხმარებლის კომპიუტერზე;

# თავი 4. საფოსტო და საკომუნიკაციო სერვერების გამართვა

## 4.1. საფოსტო და საკომუნიკაციო სერვისების ზოგადი მიმოხილვა

სწრაფი და საიმედო კომუნიკაცია 21-ე საუკუნეში საქმის წარმოების ერთ-ერთი უმთავრესი წინაპირობაა. სადღეისოდ ინტერნეტის ფიზიკური მონაცემები (კარგად განვითარებული ქსელური და კაბელური ინფრასტრუქტურა, ინფორმაციის გადაცემის საიმედო პროტოკოლები) უკვე იძლევა დიდი მოცულობის ინფორმაციის მთელი პლანეტის ფარგლებში გადაცემის საშუალებას, თუმცა ეს პრობლემის მხოლოდ ერთი ნაწილია. არანაკლებ მნიშვნელოვანია ინფორმაციის გაცვლის პროცესის იმგვარად ორგანიზება, რომ კომპიუტერული სისტემების და ზოგადად ინტერნეტის მომხმარებელმა მაქსიმალურად მრავალფეროვანი ინფორმაციის (ტექსტი, გამოსახულება, ხმა, ვიდეო) გაცვლა შესძლონ და ამასთან, მაღალი დონის უსაფრთხოებით და საიმედობით [14].

ნებისმიერი თანამედროვე, მეტნაკლებად დიდი ორგანიზაციის გამართული მუშაობისთვის სტაბილური **საკომუნიკაციო სისტემების** არსებობა აუცილებელია. ითვლება, რომ კორპორაციულ ქსელში კომუნიკაციის შემდეგი სისტემები უნდა არსებობდეს:

- ელექტრონული ფოსტა;
- სატელეფონო სერვისი (ქალაქის და მობილური);
- მყისიერ შეტყობინებათა სერვისი;

პროგრამული უზრუნველყოფის ბაზარზე მრავალი საკომუნიკაციო პროგრამული უზრუნველყოფა არსებობს. გასული საუკუნის 90-იანი წლების პირველი ნახევრიდან, ანუ ინტერნეტის განვითარების კვალდაკვალ, შეტყობინებათა გადაცემის სერვისები გამოდის ვიწრო ინფორმაციული სეგმენტებიდან (სამეცნიერო კვლევითი ორგანიზაციები, სამხედრო საქმე) და **ელექტრონული ფოსტის** სახელით იმკვიდრებს თავს მომხმარებელთა ფართო წრეში.

ელექტრონული ფოსტა ორგანიზაციის საკომუნიკაციო ინფრასტრუქტურის პირველადი, ყველაზე „ოფიციალური“ კომპონენტია.

გამართული **სატელეფონო კავშირის** არსებობა აუცილებელია როგორც ორგანიზაციის შიგა, ისე გარე კომუნიკაციისათვის. ამასთან, თანამედროვე ორგანიზაციის სატელეფონო სერვისი დამატებით მოთხოვნებსაც უნდა პასუხობდეს. კლიენტთა მომსახურების განყოფილება (*ქოლ-ცენტრი*), თანამშრომელთა მხარდაჭერის განყოფილება (*ჰელდესკი*) ის სტრუქტურული ერთეულებია, სადაც თანამშრომელთა ვალდებულება შემოსული ზარების მაქსიმალური რაოდენობით მიღება და მომსახურებაა, რისთვისაც ზარების ოპტიმალური გადანაწილებაა საჭირო.

ამგვარი ამოცანების გადასაჭრელად ადრე ანალოგური სატელეფონო ინფრასტრუქტურა გამოიყენებოდა, რომელსაც სადღეისოდ ინტენსიურად ანაცვლებს VoIP-ტექნოლოგია (Voice over IP) IP-პროტოკოლის გამოყენებით და აქედან გამომდინარე, ორგანიზაციის კომპიუტერულ ინფრასტრუქტურასთან ინტეგრირების გაუმჯობესებული მეთოდებით. VoIP ზოგადი ცნებაა და არამარტო სატელეფონო კომუნიკაციის, არამედ დაკვირვების, სასიგნალო და სხვა სისტემებშიც გამოიყენება.

აპარატული და პროგრამული უზრუნველყოფის ბაზარზე სადღეისოდ რამდენიმე წამყვანი სატელეფონო სისტემაა წარმოდგენილი. მათი უმრავლესობა პროპრიეტარულია (ფასიანია) როგორც აპარატულად, ასევე პროგრამულად, თუმცა არსებობს ეფექტური სისტემები Open Source-კოდით, სადაც მომხმარებელს მხოლოდ აპარატული პლატფორმის ხარჯების გაღება უწევს. პირველი ჯგუფის პროდუქტებიდან ლიდერობს კომპანია Cisco, რომელიც ტელეფონიის სრულფასოვან ინფრასტრუქტურას აწარმოებს დაწყებული ქსელური მოწყობილობებითა და სერვერებით და

დასრულებული IP-ტელეფონებით. მეორე ჯგუფის სისტემებიდან საქართველოში ყველაზე მეტად Asterisk-სისტემებია გავრცელებული.

ამის პარალელურად, უწყვეტად ვითარდება **მყისიერ შეტყობინებათა** (Instant Messaging) სერვისებიც. 1988 წელს დამუშავებული პროტოკოლი IRC (Internet Relay Chat) ამ მიმართულებით „პირველი მერცხალი“ იყო. მის ბაზაზე იმპლემენტირებული პროგრამები (პირველ რიგში miRC) ყველაზე პოპულარული მყისიერი კომუნიკაციის (Instant Messaging) საშუალებას წარმოადგენდა საქართველოში ინტერნეტის განვითარების გარიჟრაჟზე.

ჩვენს საუკუნეში მყისიერ შეტყობინებათა სერვისებმა უდავო პროგრესი განიცადა. გამოვიდა მრავალი პოპულარული პროგრამა (ICQ, Windows Live Messenger, Yahoo! Messenger, Skype, ooVoo, AIM, ICQ, MSN, Jitsi, XMPP). გარდა ამისა, მყისიერ შეტყობინებათა მოდულები პოპულარული პროგრამული პროდუქტების და ონლაინ-სისტემების (მაგალითად, სოციალური ქსელები) განუყრელ ნაწილად იქცა. სადღეისოდ მათ პრაქტიკულად ნებისმიერი ტიპის მონაცემებთან სწრაფად და საიმედოდ მუშაობა შეუძლია.

უნდა ითქვას, რომ მყისიერ შეტყობინებათა (Instant Messaging) სერვისის საზოგადოდ ცნობილი პროგრამული უზრუნველყოფა (Skype და სხვა) შიგა, კორპორაციული ქსელის ფარგლებში ნაკლებად გამოსადეგია. თუმცა შესაძლებელია მყისიერ შეტყობინებათა ცნობილი სერვისების მორგება კორპორაციულ ინფორმაციულ ინფრასტრუქტურაზე, მაგრამ სისტემის ადმინისტრატორებს უჭირთ აკონტროლონ Skype-ის, Google Talk-ის და სხვა ამგვარი სისტემების მომხმარებლები. *შიგა მყისიერ შეტყობინებათა სერვისები* (მაგალითად, MS Lync), მათი გლობალური კოლეგებისგან განსხვავებით, პირდაპირ ორგანიზაციის საკომუნიკაციო მოთხოვნებზეა მორგებული და გარე სამყაროსთან ყოველგვარი კავშირის გარეშე მუშაობს. მიმდინარე ეტაპზე მსგავსი სისტემები მრავალი ოპერაციული სისტემისათვის არსებობს.

## 4.2. საფოსტო სერვისის მართვა Microsoft Exchange Server-ის გარემოში

### 4.2.1. Exchange Server-ის სტრუქტურა

**ელექტრონული ფოსტა** შიგა და გარე კომუნიკაციის ძირითადი ფორმაა. მიუხედავად იმისა, რომ სადღეისოდ უამრავი კომპანია სხვადასხვა სერვისების და მათ შორის, ელექტრონული ფოსტის სამართავად მომხმარებელს „ღრუბლოვან“ სივრცეებს სთავაზობს, დიდი ორგანიზაციებისთვის ელექტრონული ფოსტის შიგა ინფრასტრუქტურის განვითარება მაინც აუცილებელ ამოცანად რჩება. ამასთან, უნდა აღინიშნოს, რომ თანამედროვე ელექტრონული ფოსტის სერვისები მოიცავს არამხოლოდ ელექტრონული წერილების, არამედ კონტაქტების, კალენდრების და სამსახურეობრივი ამოცანების მომსახურების საშუალებებს.

პროგრამული უზრუნველყოფის ბაზარზე კორპორაციული საფოსტო სერვისების მართვის მრავალი სისტემა არსებობს, რომელთაგან ზოგი დაბალი ფასით გამოირჩევა და მცირე ლოკალურ ქსელებზეა გათვლილი (მაგალითად, პოლონური პროგრამა Code two public folders), ზოგიც - ყოვლისმომცველი ფუნქციურობით. ჩვენ მეორე ვარიანტზე შეჩერდებით და წარმოავდგენთ კომპანია Microsoft-ის პროგრამას Ms Exchange Server, რომელიც კორპორაციული ელ-ფოსტის მართვის მძლავრ საშუალებებს ფლობს.

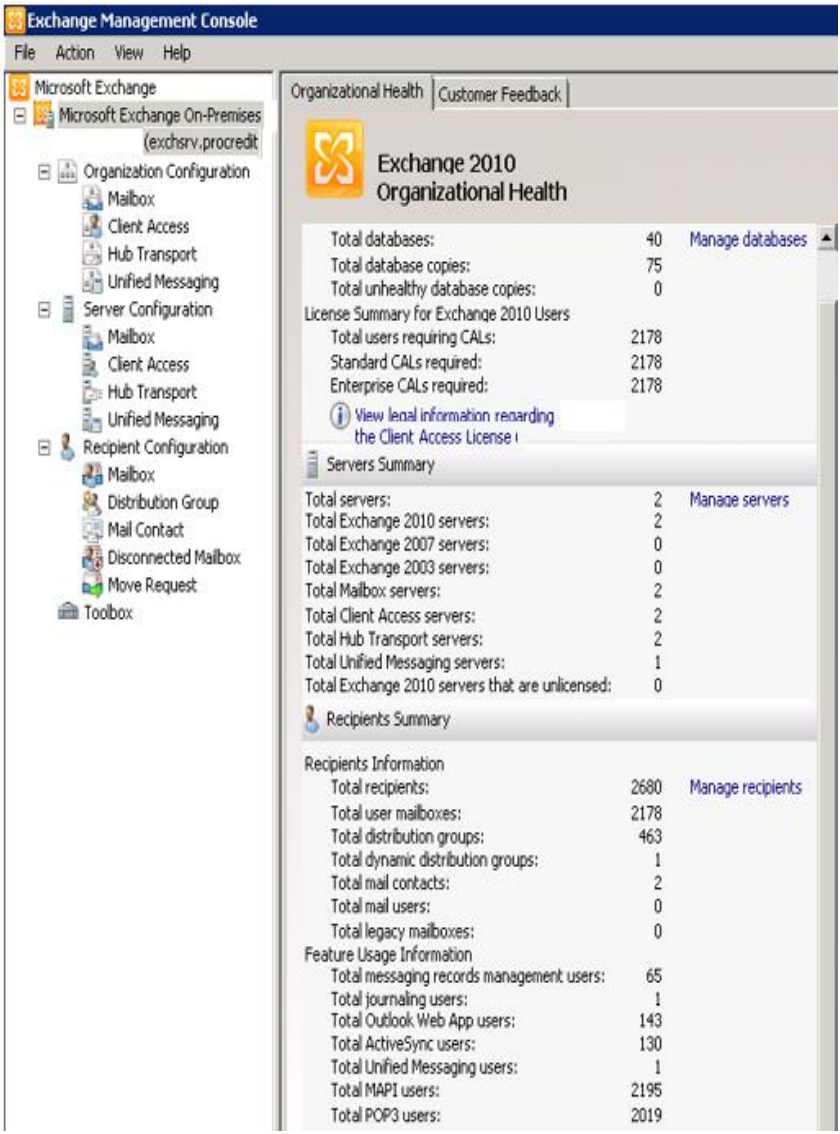
შევეხოთ პირველ რიგში სისტემის აპარატულ-პროგრამულ მოთხოვნებს და ელექტრონული ფოსტის მონაცემთა ბაზების შენახვის ინსტრუმენტებს. Ms Exchange Server რესურსების მიმართ პრეტენზიული პროგრამაა. სისტემის წარმატებული ინსტალაციის და საწარმოო რეჟიმში გაშვების შემდეგ ადვილი შესამჩნევია, რომ სერვერის პროცესორისა და განსაკუთრებით ოპერატიული მეხსიერების მოზრდილ წილს პროცესი Store.exe (Ms Exchange Server-ის მთავარი შესრულებადი ფაილი) ფლობს, ამიტომ სისტემის

ადმინისტრატორებს მსგავსი სისტემების მართვისას მწარმოებლურობის მაჩვენებლებზე მუდმივად ყურადღების გამახვილება უწევთ.

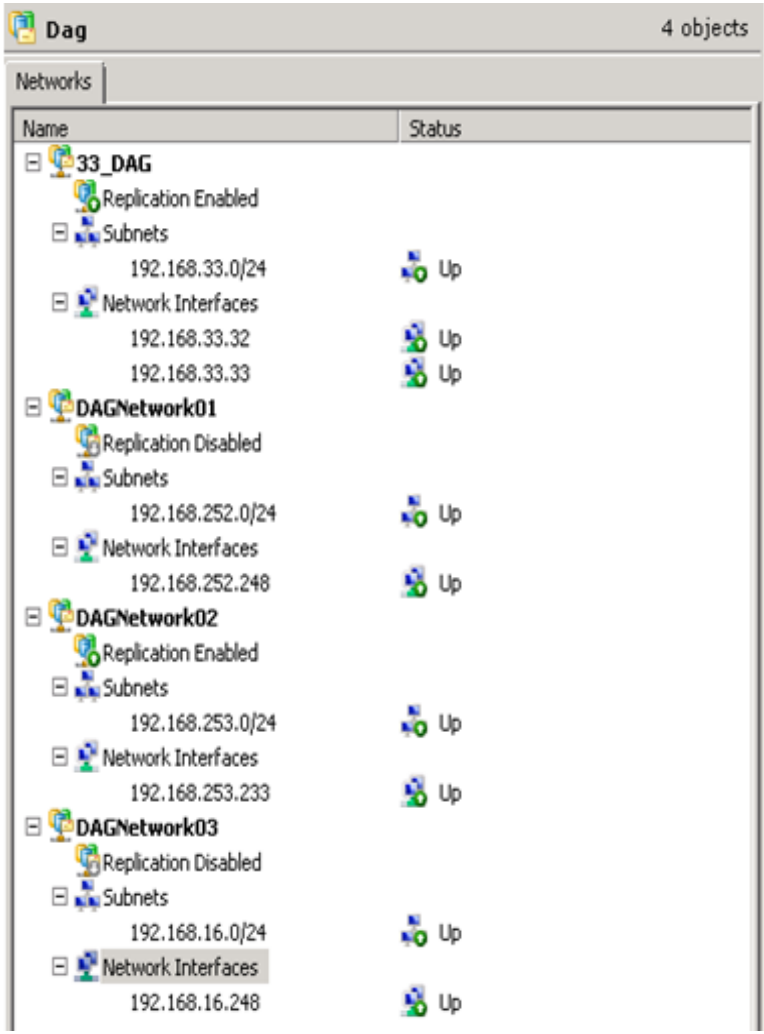
Ms Exchange Server სისტემა აუთენტიფიკაციას, როგორც წესი, დომენურ სისტემაში ასრულებს, ანუ ელ-ფოსტის სერვერის მომხმარებლები იგივე დომენური მომხმარებლებია.

სერვერზე ინფორმაცია იერარქიულად ინახება: ელ-ფოსტის ფაილურ საცავს მეილების მონაცემთა ბაზა ანუ **სთორი** (Store) ეწოდება, რომელიც თავის მხრივ, გარკვეული რაოდენობის, მომხმარებელთა ელექტრონული ფოსტის მისამართებთან დაკავშირებული *საფოსტო ყუთებისგან* (Mailboxes) შედგება. სწორედ სთორების შექმნისას ითვალისწინებენ დაწესებულების ორგანიზაციულ სტრუქტურას: მაგალითად, ცალკე სთორებში შეიძლება იყოს თავმოყრილი უმაღლესი და საშუალო რგოლის მენეჯერების, აგრეთვე სხვადასხვა დეპარტამენტის რიგით თანამშრომელთა საფოსტრო ყუთები. ამგვარი მიდგომა გარე მეხსიერების და სხვა რესურსების, აგრეთვე ელექტრონული ფოსტის გამოყენების განსხვავებული წესების დადგენის საშუალებას იძლევა. ქვემოთ, სურათზე ნაჩვენებია Ms Exchange-ის მართვის პანელის (Exchange Management Console) საწყისი ფანჯარა ინფორმაციით Exchange-სერვერების, ელ-ფოსტის მონაცემთა ბაზების (სთორების) და საფოსტო ყუთების (მეილბოქსების) შესახებ (ნახ.4.1).

სურათის მიხედვით, Exchange-სერვისი 2 Exchange-სერვერზე მუშაობს, რაც მთლიანად სისტემის სარგებლიანობის ამაღლებას ემსახურება. სპეციალური, DAG-სერვისი (Database Availability Group) უზრუნველყოფს სთორების და მეილბოქსების რეპლიკაციას სერვერებს შორის, რომელთაგან ერთი მუდმივად აქტიურ, ხოლო მეორე - პასიურ მდგომარეობაშია. მთავარი სერვერის გათიშვისას DAG-სერვისი მართვას ავტომატურად სათადარიგო სერვერს გადასცემს. ქსელური კავშირები Exchange-სერვერებს შორის დუბლირებულია, რაც ასევე ამაღლებს მთლიანი სისტემის სარგებლიანობას (ნახ.4.2).

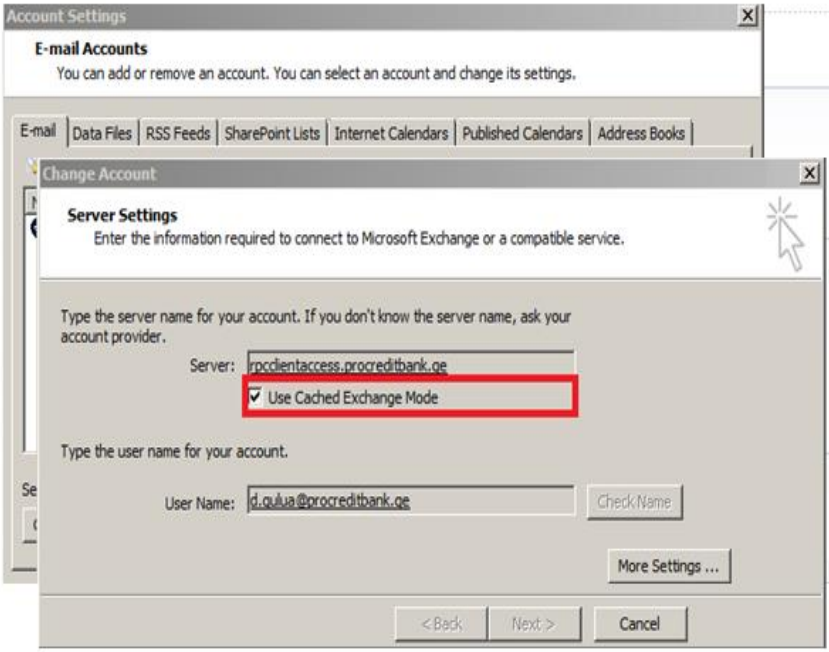



ნახ. 4.1. Exchange Management Console-ს მთავარი ფანჯარა

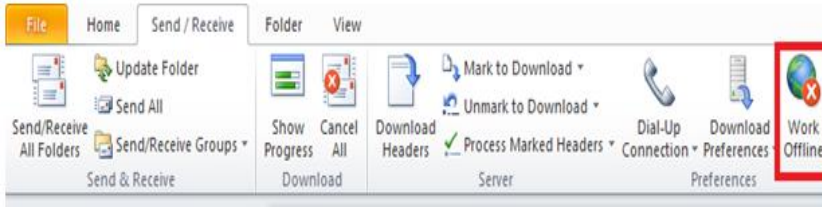



ნახ.4.2. DAG-სერვისის მართვის ინტერფეისი

Exchange Server მომხმარებელს სერვერთან **პირდაპირ მიმართვის** (online) და **კეშის** (Cached Exchanged Mode) რეჟიმებში მუშაობის საშუალებას აძლევს. მეორე რეჟიმი მოხერხებულია, როცა მოხმარებელს ელ-ფოსტის სერვერთან მუდმივი ან/და სწრაფი კავშირის საშუალება არ აქვს. ასეთ შემთხვევაში ელექტრონული წერილები, კონტაქტები, კალენდარი, ამოცანები და მომხმარებლის სხვა რესურსები ლოკალურ კეშში, კერძოდ .ost-ფაილებში იყრის თავს და სერვერთან პერიოდულ სინქრონიზაციას ასრულებს (ზოგადად, ლოკალური მეილბოქსები კომპიუტერზე .pst-ფორმატის ფაილებში ინახება). 4.3-ა,ბ ნახაზებზე ნაჩვენებია ელ-ფოსტის კლიენტ-პროგრამაში (Ms Outlook) კეშის რეჟიმის აქტივაციისა და ოფლაინ-რეჟიმზე გადართვის დიალოგები.

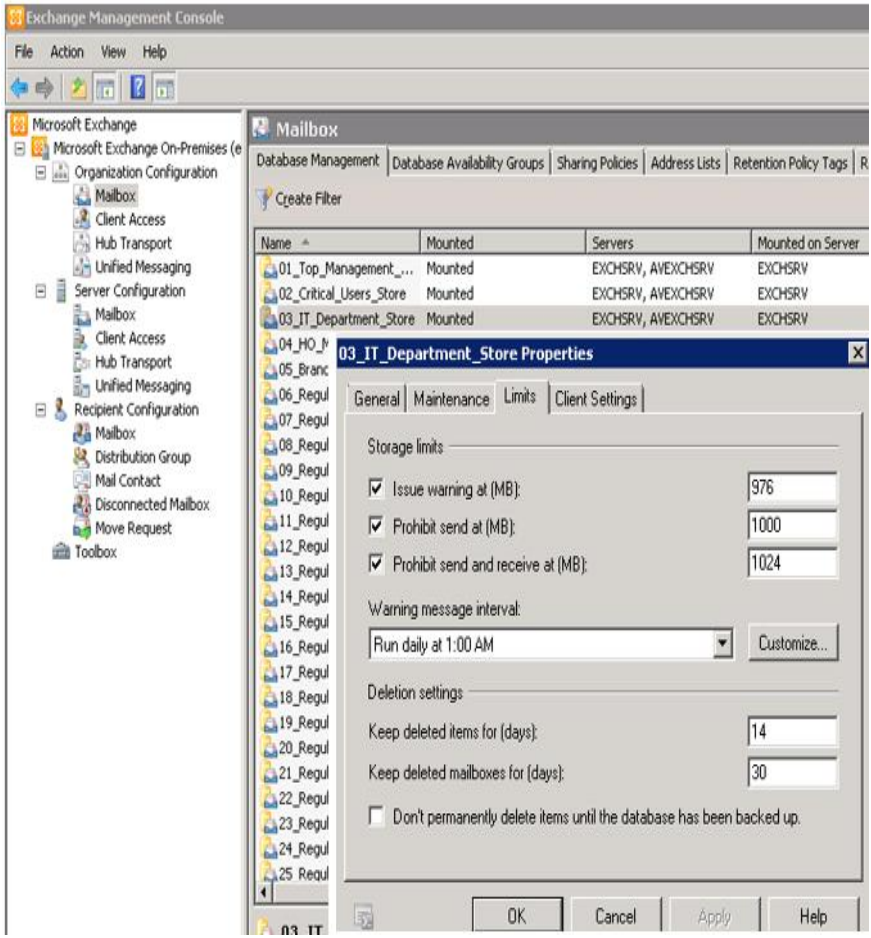



ნახ.4.3-ა. კეშის რეჟიმის აქტივაცია Ms Outlook-ში

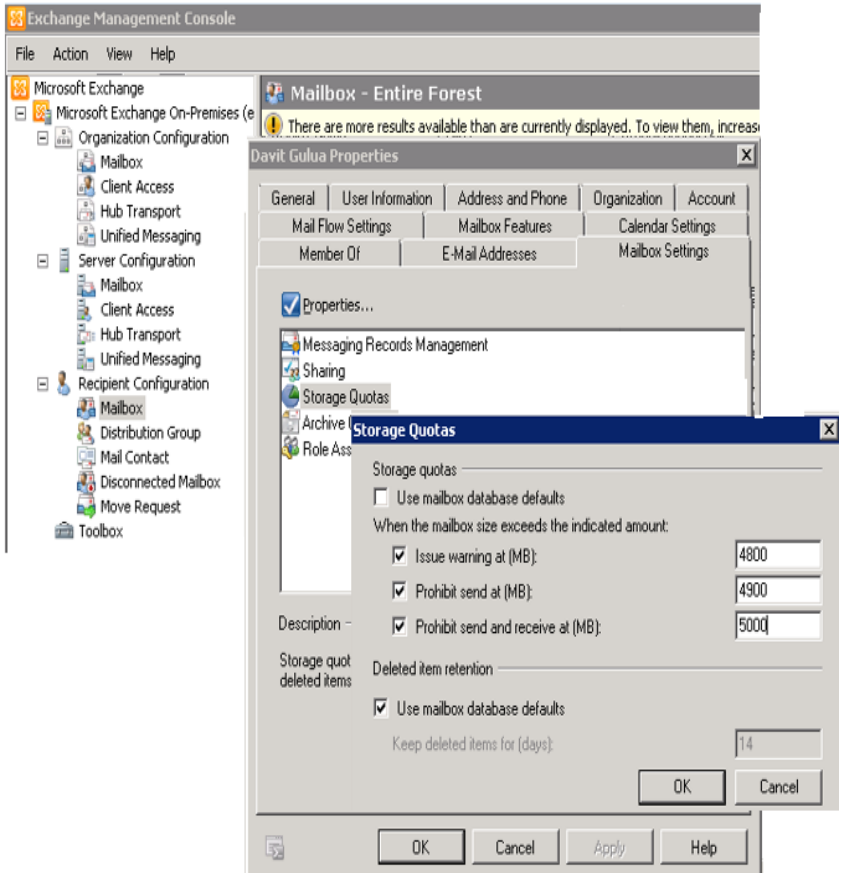



ნახ.4.3-ბ. ოფლაინ-რეჟიმის აქტივაცია MS Outlook-ში

როგორც ფაილურ სერვერს, ელ-ფოსტის სერვერსაც აქვს მეხსიერების კვოტირების ფუნქცია, რომელიც ყოველი მომხმარებლისთვის ელექტრონული წერილებისთვის გარკვეული დისკური სივრცის გამოყოფას ითვალისწინებს (მაგალითად, 500 მბ-დან **სტანდარტული** მომხმარებლებისათვის, 2,5 გბ-მდე **ტოპ-მენეჯმენტისათვის**). ცალკე ფუნქციაა **ონლაინ არქივიც** (მაგალითად, 5 გბ დამატებითი მეხსიერების მოცულობით), რომელშიც ელ-ფოსტის ძირითადი საცავიდან, გარკვეული წესების მიხედვით, მეილების გადატანა ხორციელდება ძირითადი საცავის განტვირთვის მიზნით. მომდევნო ორ ნახაზზე ნაჩვენებია გარე მეხსირების კვოტების განსაზღვრის დიალოგები მთლიანი მეილ-სთორის ანუ ყველა მეილბოქსისა და კონკრეტული მეილ-ბოქსისთვის (ნახ.4.4-ა,ბ).




ნახ.4.4-ა. მეხსიერების კვოტირება მეილ-სთორისთვის




ნახ.4.4-ბ. მეხსიერების კვოტირება მეილ-ბოქსისთვის

### 4.2.2. წესების მართვა MS Exchange Server-ის გარემოში

ელექტრონული ფოსტის სერვერზე ადმინისტრირების სხვადასხვა ამოცანის გადაწყვეტა შეიძლება. პირველ რიგში აღვნიშნოთ, რომ MsExchange არის ორგანიზაციის დომენურ ინფრასტრუქტურაში (Active Directory) ინტეგრირებული სისტემა.

Exchange-ის მომხმარებლის მრავალი თვისება AD-დან მოდის. საკმარისია აღვნიშნოთ, რომ დომენური სააღრიცხვო ჩანაწერით რეგისტრაციისას მომხმარებლი ავტომატურად გადის რეგისტრაციას Exchange-სერვერზეც. ორი სისტემის ინტეგრაციის მაღალ ხარისხზე შემდეგი ფაქტებიც მიუთითებს:

- ახალი Exchange-მომხმარებლის (საფოსტო ყუთის) **შექმნისას** AD-მომხმარებელიც **ავტომატურად იქმნება**;
- Exchange-მომხმარებელის (საფოსტო ყუთის) **წაშლისას** AD-მომხმარებელიც **ავტომატურად იშლება;**
- AD-მომხმარებელის **წაშლისას** Exchange-მომხმარებელიც (საფოსტო ყუთი) **ავტომატურად იშლება.**

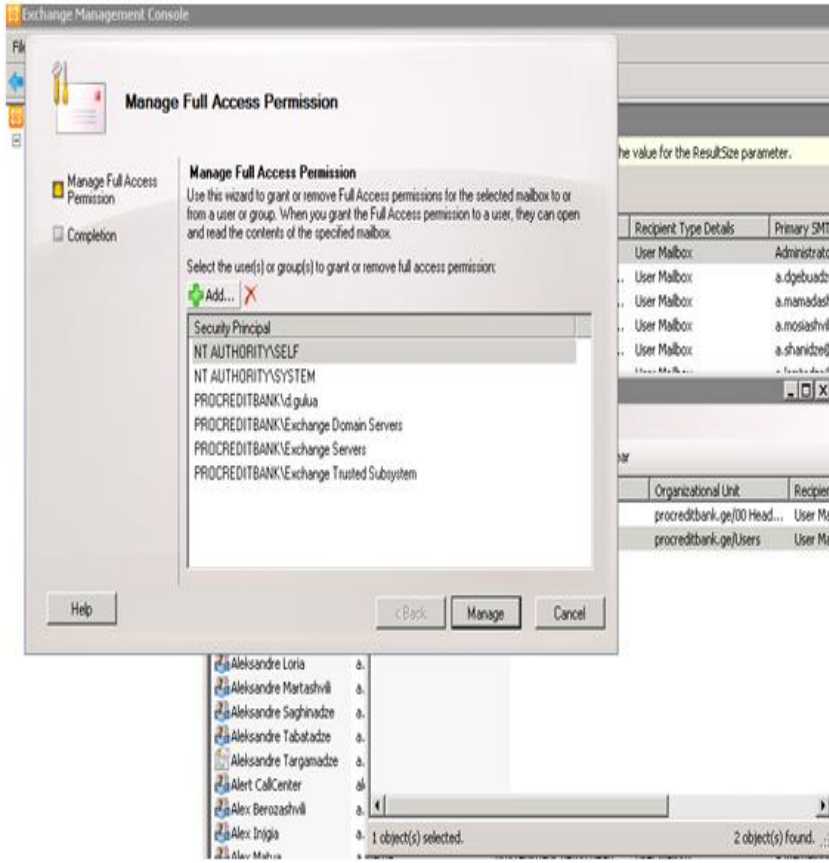
ერთადერთი გამონაკლისი: ახალი AD-მომხმარებელის **შექმნისას** Exchange-მომხმარებელი (საფოსტო ყუთი) **ავტომატურად არ იქმნება**.

ახლა ორიოდე ყველაზე მარტივი ფუნქციაც გავარჩიოთ:

თუ მომხმარებლის მეილ-ბოქსზე სრული წვდომის დამყარება (მ.შ. ჩვენს ელ-ფოსტის კლიენტის ფანჯარაში მისი ელექტრონული წერილების გამოტანა) გვჭირდება, საჭიროა გამოვიძახოთ ბრძანება:

*მაუსის მარჯვენა ღილაკი მეილ-ბოქსზე -> Manage Full Access Permission*

და Add-ღილაკით დავამატოთ ჩვენი (ან სხვა მომხმარებლის) დომენური სააღრიცხვო ჩანაწერი (ნახ.4.5).

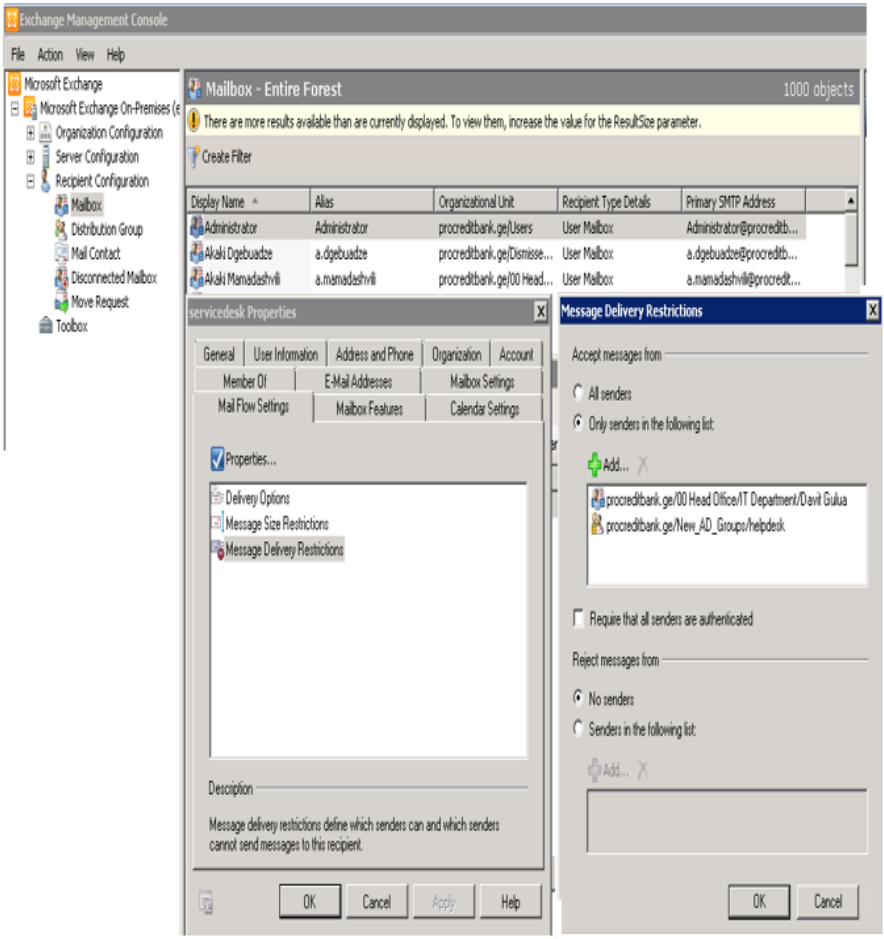



ნახ.4.5. სრული უფლებების ჩართვა მეილ-ბოქსზე

თუ საჭიროა იმ მომხმარებელთა რაოდენობის შეზღუდვა, რომელთაც მოცემული მეილის მისამართზე მოწერა შეუძლიათ, საჭიროა შემდეგი ბრძანების შესრულება:

*მაუსის მარჯვენა ღილაკი მეილ-ბოქსზე -> Properties -> Mail Flow Settings -> Message Delivery Restrictions*

რის შემდეგაც გამოსულ ფანჯარაში All Senders ჩავანაცვლოთ ბრძანებით: Only Senders in the following list და ავირჩიოთ სასურველი დომენური მომხმარებლები (ნახ.4.6).




ნახ.4.6. მეილ-ბოქსზე გამოგზავნის უფლებათა შეზღუდვა

მართვის შედარებით რთული ოპერაციების განსახორციელებლად Ms Exchange Server *წესების* განსაზღვრის საშუალებას იძლევა, რომლის დახმარებითაც ელ-ფოსტის სერვერზე მომხმარებლის პრაქტიკულად ყველა მოქმედების გაკონტროლება შეიძლება.

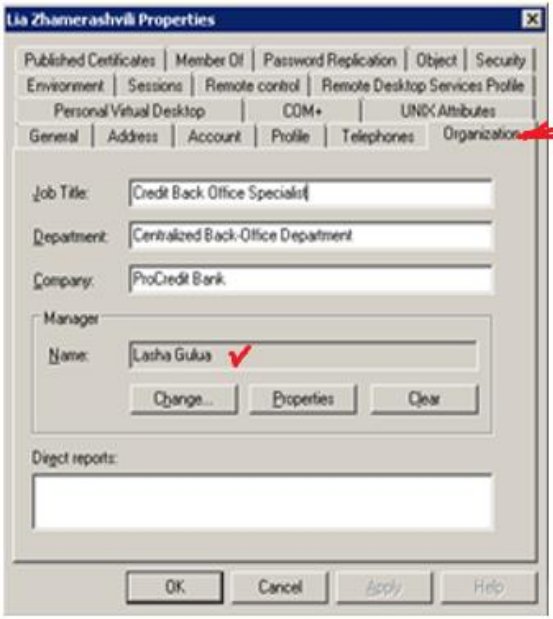
განვიხილოთ მაგალითი, რომელიც ორგანიზაციის გარეთ გამავალი და ორგანიზაციაში შემომავალი ელექტრონული წერილების კონტროლს ეხება. ამგვარი წესის აუცილებლობა ნათელია: მონაცემთა გაჟონვას ან არაკვალიფიციურ მიმოწერას ორგანიზაციისთვის სერიოზული ფინანსური და საიმიჯე ზარალის მიყენება შეუძლია. შიგა IT ინფრასტრუქტურის ფარგლებს გარეთ გამავალი და შემომავალი ელექტრონულ წერილებს უფლებამოსილი პირები (როგორც წესი, თანამშრომლების უშუალო ხელმძღვანელები) აკონტროლებენ (ამ პროცესს **მოდერაციასაც** უწოდებენ).

მოდერაცია ავტომატური პროცესია, ანუ Exchange Server-ის ფარგლებიდან გამავალი ნებისმიერი მეილი დასტურის ან უარყოფისთვის თავდაპირველად მოდერატორს მისდის და გზას მხოლოდ დადებით პასუხის შემთხვევაში გააგრძელებს. მოდერატორის ვინაობა მომხმარებლის დომენური საიდრიცხვო ჩანაწერის მართვის დიალოგში (გადამრთველი Organization, ველი Manager) დგინდება. ქვემოთ, 4.7 ნახაზზე ნაჩვენებია Manager-ის განსაზღვრის დიალოგი დომენური მომხმარებლისთვის, ხოლო 4.8 ნახაზზე გააქტიურებულია Ms Exchange-ის ე.წ. *ტრანსპორტირების წესი* (Transport Rule):

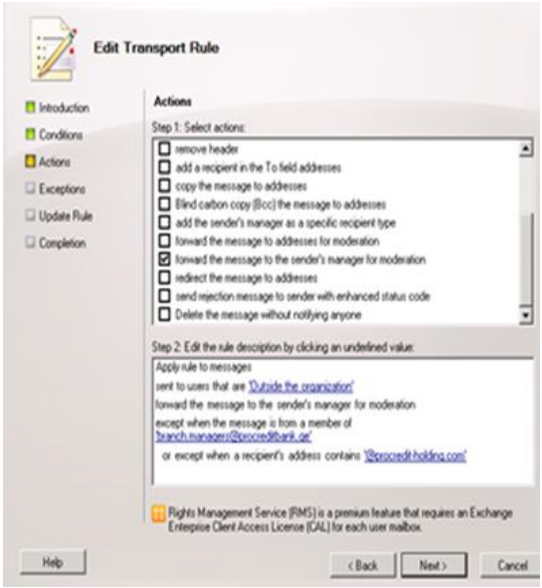
***Forward the Message to the senders Manager for moderation***

რომელიც მოდერაციის ოპერაციის შესრულებას უზრუნველყოფს.

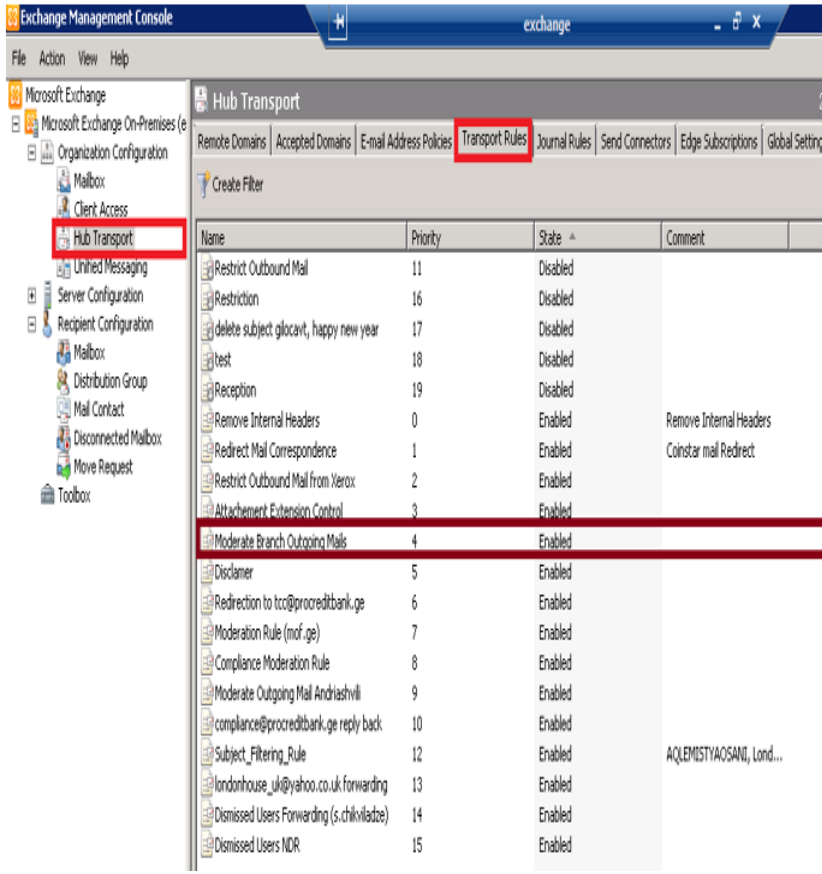
4.9 ნახაზზე გამოტანილია Exchange Server-ის წესების მართვის ინტერფეისი, რომელშიც ჩვენ მიერ განსაზღვრული წესია გამოყოფილი (*Exchange Management Console -> Moderate Branch Outgoing Mail*).




ნახ.4.7. Organization, ველი Manager: მოდერატორის ვინაობის განსაზღვრა




ნახ.4.8. Ms Exchange-ის ტრანსპორტირების წესის გააქტიურება




ნახ.4.9. AD-მენეჯერის მიერ გარე მეილის მოდერაციის დიალოგები

### 4.3. საკონტროლო დავალებები:

1. შეასრულეთ პროგრამა Ms Exchange Server 2013-ის ინსტალაცია;

2. შექმენით 2-3 ახალი ელ.ფოსტის მონაცემთა ბაზა (სთორი). ყოველ მათგანში მოათავსეთ 2-3 დომენური მომხმარებლის მეილბოქსი;

3. შეასრულეთ მეილ-ბოქსებზე გაცემული გარე მეხსიერების კვოტირება მთლიანი სთორისა და ცალკეული მეილ-ბოქსებისთვის;

4. შექმენით ახალი წესი (Rule), რომლის მიხედვითაც ხელმძღვანელი მის დაქვემდებარებაში მყოფი თანამშრომლების მიერ „გარეთ“ გაგზავნილი მეილების მოდერაციას შეასრულებს.

# ნაწილი III

# კლიენტ-სერვერული და სერვისორიენტირებული სისტემები კორპორაციული ქსელებისათვის

ბიზნესპროცესების შესრულებისას ერთ–ერთი მნიშვნელოვანი ასპექტია კომუნიკაცია (ურთიერთობა) აპლიკაციებს შორის, კლიენტებსა და სერვერებს შორის, აგრეთვე სამუშაო–პროცესებსა (workflow) და ჰოსტ–დანართებს შორის. აქ გავეცნობით თუ როგორ შეიძლება ბიზნესპროცესების გამოყენებით გამარტივდეს და კოორდინაცია გაეწიოს კომუნიკაციათა სხვადასხვა სცენარებს. აპლიკაციის სახით განიხილება პროექტების აგება პროდუქციის წარმოებისა და ბიზნესის სფეროს მაგალითებისათვის, რომლებშიც ინფორმაციის („მოთხოვნა-პასუხის“) გადაცემა მათ ფილიალებს ან სტრუქტურულ ერთეულებს შორის ხორციელდება კლიენტ-სერვერული ან სერვისორიენტირებული არქიტექტურის პრინციპებით [20].

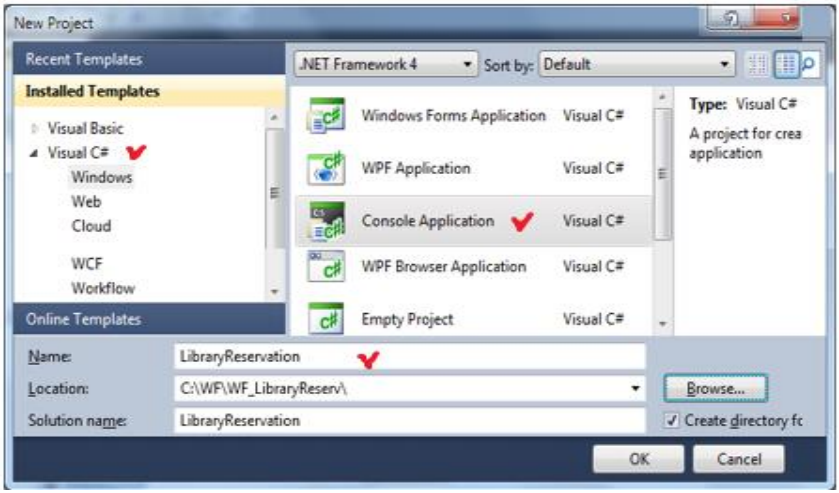
# თავი 5. კომუნიკაცია აპლიკაციებს შორის Ms WCF-ის ბაზაზე

## 5.1. ინფორმაციის „გადაცემის“ და „მიღების“ ქმედებები

მთავარი ქმედებები, რომლებიც კომუნიკაციისათვის 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 [20,25]. ამ თავში ავაგებთ კონსოლის მარტივ აპლიკაციას, რომელიც გამოიყენებს კომუნიკაციას იგივე აპლიკაციის სხვა ბიზნესპროცესთან (ჯერ „ბიბლიოთეკის“ მაგალითზე, რადგან იგი სტუდენტებისათვის კარგად ნაცნობი სფეროა, შემდეგ კი განვიხილავთ სხვა საპრობლემო სფეროებსაც).

- **ახალი პროექტის შექმნა**

შევქმნათ ახალი პროექტი LibraryReservation სახელით, ჩვეულებრივ Windows -> Console Application რეჟიმში (ნახ.5.1).



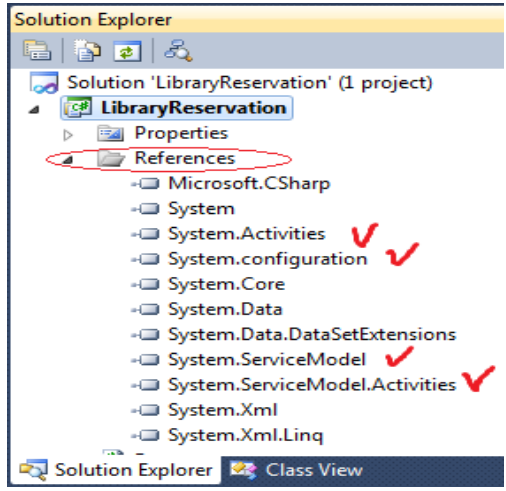

ნახ.5.1

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.5.2

➢ 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:

```
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 5.1 ლისტინგში. შევიტანოთ იგი LibraryReference namespace–ს შიგნით. წავშალოთ ცარიელი კლასი Reservation, რომელიც წინა ბიჯზე შეიქმნა.

```
// ----- ლისტინგი_ 1 .1 ----------------------
// განისაზღვროს ფილიალის მონაცემტა სტრუქტურა
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, რომელიც 5.2 ლისტინგშია მოცემული.

```
//----- ლისტინგი_5.2 -------------
// მოთხოვნის შეტყობინების განსაზღვრა: ReservationRequest------
[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.

- **კონტრაქტის შეტყობინება [MessageContract]**

ვინაიდან ReservationRequest კლასი გამოყენებულ იქნება გამომავალი შეტყობინების განსაზღვრისათვის, 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 ურთიერთ-ქმედების პლატფორმას. SOAP არის სტანდარტული პროტოკოლი, რომლის მხარდაჭერაც აქვს WCF-ს.

არსებობს აგრეთვე MessageBodyMember ატრიბუტი მის ყოველ public თვისებაზე. ეს აუცილებელია WCF შრისათვის, რათა სწორად დაფორმატდეს SOAP შეტყობინება.

ახლა შევიტანოთ რეალიზაციის კოდი ReservationResponse კლასისითვის, რომელიც 5.3 ლისტინგზეა მოცემული.

```
//------ ლისტინგი_5.3 ---------
// მოთხოვნაზე საპასუხო შეტყობინების განსაზღვრა: ReservationResponse
[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 შეკვეთისათვის. ის შეიცავს ასევე Branch კლასს, რომელიც ასახავს იმ ფილიალს, რომელმაც შესარულა ეს მოთხოვნა.

- **სერვისის კონტრაქტი [ServiceContract]**

WCF-ის ბოლო წერტილის განსაზღვრისათვის არსებობს ინფორმაციის სამი პორცია, რომლებიც მითითებულ უნდა იქნას: მიერთება (binding), მისამართი და კონტრაქტი.

**მიერთება** მიუთითებს იმ პროტოკოლს, რომელიც გამოიყენება (მაგალითად, HTTP, TCP ან სხვ.).

**მისამართი** მიუთითებს, თუ სად უნდა ვიპოვოთ ბოლო წერტილი, და მისამართის ტიპს, რომლის გამოყენება დამოკიდებულია მიერთებაზე. მაგალითად, HTTP მიერთებისას უნდა მიეთითოს URL, ხოლო TCP-სთვის 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 სერვისმეთოდების მიერ პარამეტრების სახით.

ახლა დავამატოთ ინტერფეისის განსაზღვრება, რომელიც 5.4 ლისტინგზეა მოცემული იმავე Reservation.cs ფაილში.

```
//----- ლისტინგი_5.4 ------------------
// სერვისის კონტრაქტის განსაზღვრა: ILibraryReservation
// რომელიც შეიცავს ორ მეთოდს: RequestBook() და 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 – კლიენტისკენ.

დავაკომპილიროთ პროექტი (F5) და გავმართოთ კოდი.

### ➢ აპლიკაციის კონფიგურაცია

როგორც ადრე ვთქვით, ჩვენ გვექნება ამ *აპლიკაციის რამდენიმე ასლი (კოპიო),* რომლებიც ასახავს *სხვადასხვა ფილიალის* განთავსებას. ფილიალთა მონაცემები ინახება კონფიგურაციის ფაილებში. **აპლიკაციის ყოველ ასლს ექნება თავისი კონფიგურაციის ფაილი, რომელიც შეიცავს თავის სპეციფიკურ ატრიბუტებს**.

LibraryReservation პროექტის Solution Explorer-დან ვირჩევთ Add->NewItem. დიალოგურ ფანჯარაში General ჯგუფში ვირჩევთ Application Configuration File. ფაილის სახელი ავტომატურად არის App.config ().

კონფიგურაციის ფაილში შევიტანოთ საჭირო მონაცემები 5.5 ლისტინგის მსგავსად.

```
<!— ლისტინგი_5.5 ———— –>
<?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 სხვა პორტების გამოყენებაც, თუ ეს საჭიროა.

### ➢ ბიზნეს–პროცესების განსაზღვრა [Defining the Workflows]

შემდეგი ბიჯი არის კლიენტის და სერვერის სამუშაო პროცესების განსაზღვრა. მათ შორის კომუნიკაცია ხორციელდება Send და Receive ქმედებათა გამოყენებით. ამ პროექტში ჩვენ გამოვიყენებთ workflow-ის კოდირებას, ნაცვლად დიზაინერისა .xaml ფაილის გენერაციისათვის [21].

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;
```

### ➢ კლიენტი – მოთხოვნების გაგზავნა [Client–SendRequest]

პირველ რიგში უნდა განისაზღვროს სამუშაო პროცესი (workflow), რომლითაც მოთხოვნა გადაეცემა სხვა ფილიალს. ReservationWF კლასის კოდი შევცვალოთ 5.6 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 პროცესისთვის
    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 და იგი ადრე იყო განსაზღვრული ReservationResponse კლასით.

კონსტრუქტორი აგრეთვე განსაზღვრავს ზოგიერთ ლოკალურ ცვლადებს. მათი გადაცემა არ ხდება ბიზნესპროცესიდან (ან ბიზნესპროცესში), 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 პროცესისათვის, 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 ცვლადებისგან, რომლებიც შევქმენით. ისინი იქმნება სახელდებული კლასების სახით, რომლებიც მოგვიანებით იქნება გამოყენებული.

➢ გაგზავნის ქმედება [Send Activity]

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 დანიშნულების ადგილას შეტყობინების მიღების დროს. Content თვისება შეინახავს მიმთითებელს შეტყობინებაზე (ReservationRequest კლასი), რომელიც უნდა გაიგზავნოს.

## 5.2. 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 5.7 ლისტინგში.

```
//——— ლისტინგი_5.7 ——————————
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
// ფილიალისთვის, 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 ბიზნეს–პროცესის განსაზღვრება 5.8 ლისტინგის შესაბამისად. ეს უნდა ჩაჯდეს იქ, სადაც მითითებულია (// SendRequest სამუშაო პროცესის განსაზღვრა).

```
// ——— ლისტინგი_5.8 ——----------
// 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 ერთობლიობისგან. 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 მიმდევრობისგან. კერძოდ, ის შეიცავს Send და ReceiveReply ქმედებებს, რომელთა მოთავსებით CorrelationScope ქმედებაში შესაძლებელი ხდება შემოსული მოთხოვნისა და საპასუხო შეტყობინების კორელაციის განსაზღვრა მოცემული ბიზნეს–პროცესისთვის.

WriteLine ქმედება ემატება Send ქმედების შემდეგ, რათა მიეთითოს, რომ მოთხოვნა იქნა გაგზავნილი.

### ➢ პასუხის მიღების ქმედება [ReceiveReply Activity]

ReceiveReply ქმედება ასოცირდება გაგზავნის Send ქმედე-ბასთან. 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 თუ სად ინახება საპასუხო შეტყობინება (ReservationResponse კლასი). ამით ყენდება სამუშაო ნაკადის Response გამომავალი არგუმენტი. იგი მისაწვდომი იქნება ჰოსტ–აპლიკაციისთვის როცა ვორკფლოვ–პროცესი დასრულდება.

### ➢ სერვერი – მოთხოვნის დამუშავება [Server–ProcessRequest]

ახლა განვსაზღვროთ სამუშაო პროცესი, რომელიც სრულდება მოთხოვნის დასამუშავებლად სხვა ფილიალიდან. ReservationWF.cs

ფაილში დავამატოთ კლასის განსაზღვრა 5.9 ლისტინგის შესაბამისად.

```
// —— ლისტინგი_5.9 ——————————————————
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, თუ არა. RequestHandle გამოიყენება შემოსულ მოთხოვნის და პასუხის კორელაციისთვის.

### ➢ მიღების ქმედება [Receive Activity]

Receive ქმედების სახელმინიჭებული ეგზემპლარი (receive-Request – 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.

### ➢ 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 5.10 ლისტინგზე.

// —— ლისტინგი_5.10 ——————

```
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 შევიტანოთ სამუშაო პროცესის განსაზღვრება, როგორც 5.11 ლისტინგშია. (ეს უნდა ჩაემატოს //სამუშაო პროცესის განსაზღვრა ProcessRequest).

```
// —— ლისტინგი_5.11 ——————
// ProcessRequest ბიზნესპროცესის 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, გამოცხადებულია 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.

### ➢ 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 5.12 ლისტინგში.

```
// ── ლისტინგი_5.12 ──────────
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
  {
  // შემავალი და გამომავალი არგუმენტების 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()
  {
  // ამ ბიზნესპროცესის ცვლადების განსაზღვრა----
  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
              { 7Request = receiveRequest,
                Content = SendContent.Create
                (new InArgument<ReservationResponse>(response))
              }
             }
            };
          }
        }
      }
```

## 5.3. აპლიკაციის რეალიზაცია

ბოლო ბიჯი პროექტის ასაგებად არის ჰოსტ–აპლიკაციის 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 აპლიკაციისთვის.

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;
```

- **ServiceHost კლასი [WorkflowServiceHost]**

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.

შვიტანოთ 5.13 ლისტინგის კოდი როგორც main() ფუნქციის რეალიზაცია Program კლასისთვის.

```
// --ლისტინგი 5_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 ღებულობს ფილიალის სახელს, რომელიც ეკრანზე გამოიტანება. ვინაიდან ჩვენ საქმე გვაქვს რამდენიმე აპლიკაციასთან, ეს მექანიზმი საშუალებას მოგვცემს თვალყური ვადევნოთ თუ რომელი რომელია.

- **სერვისი - WorkflowService კლასი**

შემდეგ იგი ქმნის WorkflowService კლასს. Body თვისებისთვის ის იყენებს ProcessRequest კლასის ახალ ეგზემპლარს, რომელიც განსაზღვრავს პროცესს შემავალი მოთხოვნების დასამუშავებლად.

- **ბოლო წერტილი [Endpoint]**

სერვისის კლასი აგრეთვე განსაზღვრავს ბოლო წერტილს, კონტრაქტის IlibraryReservation სერვისის გამოყენებით, URL–ით, რომელიც შეიცავს პორტის ნომრის ცვლადს და BasicHttpBinding

კლასს. დასასრულ, WorkflowServiceHost კლასი კონკრეტიზირდება 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.

### ➢ სამუშაო პროცესის გამომძახებელი [WorkflowInvoker]

ახლა საჭიროა კოდის დამატება მოთხოვნის ინიციალიზებისათვის. შევიტანოთ 5.14 ლისტინგის კოდი ოღონდაც wsh.Open() მეთოდის გამოძახების შემდეგ.

```
// ---- ლისტინგი 5_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 სამუშაო ასლი და თვალყური ვადევნოთ შემოსულ შეტყობინებებს. კოდის დანარჩენი ნაწილი ცნობილია ჩვენთვის, ის ჰგავს 4–7 თავების მასალას. ჯერ იქმნება ლექსიკონი შემავალი არგუმენტებისთვის. 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 მუშა პროცესი დასრულდება. დასასრულ, WorkflowServiceHost დაიხურა მუშა პროცესის დასრულებამდე.

ქვემოთ მოცემულია მთლიანი Program.cs პროგრამის ლისტინგი 5.15.

```
// ----- ლისტინგი_5.15 ------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 პროცესისთვის
```

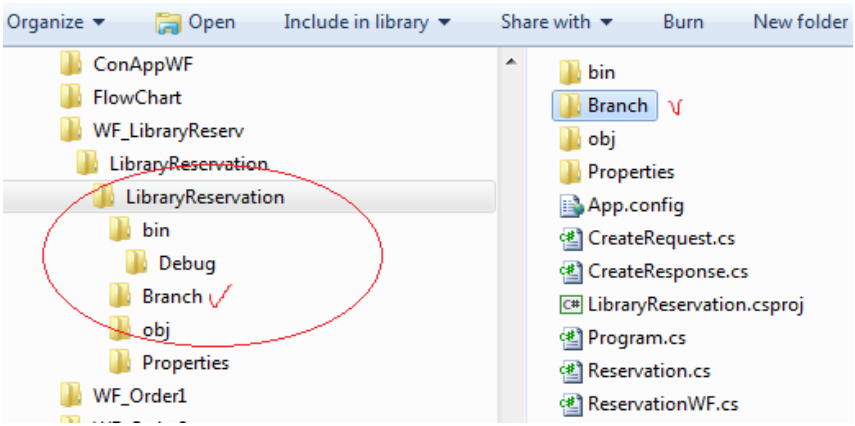
```
IDictionary<string, object> input = new Dictionary<string, object>
 {  { "Title" , "Gone with the Wind ქარწაღებული" },
    { "Author", "Margaret Mitchell" },
    { "ISBN", "9781416548898" }
 };
 // SendRequest სამუშაო პროცესის გამოძახება --------
  IDictionary<string, object> output =
     WorkflowInvoker.Invoke(new 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();
 }
}
}
```

### ➢ აპლიკაციის ამუშავება [Running the Application]

F5-ით ავამუშავოთ აპლიკაცია და გავმართოთ კოდი. ჩენ შგვიძლია აპლიკაციის ორი ეგზემპლარის ამუშავება და მათ დასჭირდება განსხვავებული კონფიგურაციის ფაილები. ამიტომაც შეიძლება მათთვის შერჩეულ იქნას სხვადასხვა პორტის ნომრები.

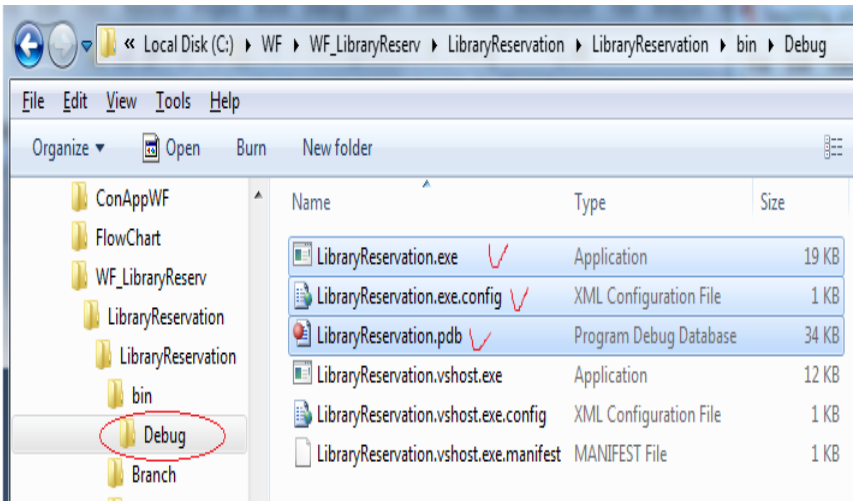
### ➢ ბიბლიოთეკის ფილიალის კონფიგურირება

გავხსნათ Windows Explorer და LibraryReservation ფოლდერში ჩავამატოთ ახალი ქვეკატალოგი Branch (ფილიალი) იხ. ნახ.5.4-ა.



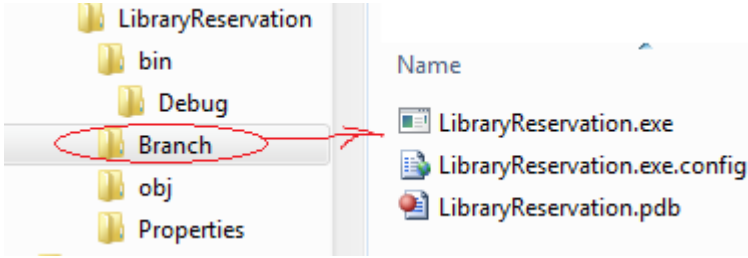

ნახ.5.4-ა

Debag-კატალოგში მოვნიშნოთ სამი ფაილი (ნახ.5.4-ბ) და კოპირებით გადავიტანოთ Branch ფოლდერში.




ნახ.5.4-ბ

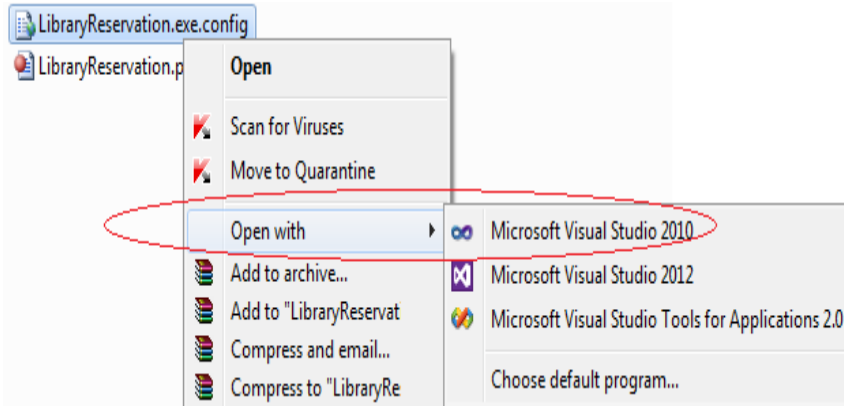
კოპირების შემდეგ Branch–ში მივიღებთ:




ნახ.5.4–გ

გავხსნათ LibraryReservation.exe.config ფაილი რომელიმე ტექსტურ რედაქტორში, მაუსის მარჯვენა ღილაკით Open with… და მაგალითად, Visual Studio 2010-ში (ნახ.5.4–დ).

შევცვალოთ კონფიგურაციის ფაილის ტექსტი შემდეგი კოდის 5.16_ლისტინგით.




ნახ.5.4–დ

```
<!– ლისტინგი_5.16 ——— –>
<?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.

## ➢ მოსალოდნელი შედეგები [Expected Results]

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 შემდეგი 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 შემდეგი 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:

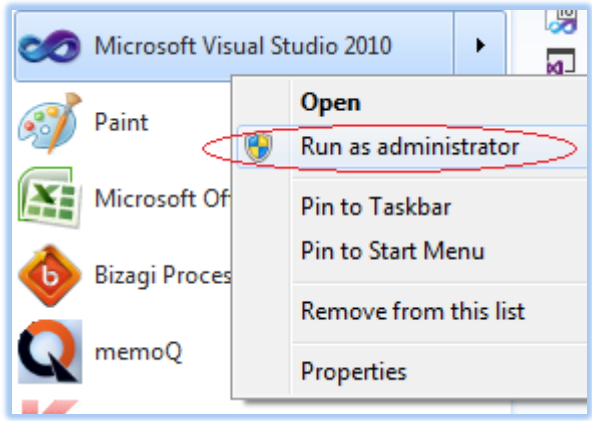
Southwest Regional
Waiting for requests, press ENTER to send a request.
Request sent; waiting for response
Response received from Central Library
Response received from the Central Library branch
Press ENTER to exit
Got request from: Central Library
Requesting: Gone with the Wind
Sending response to: Central Library

Enter–ღილაკით შეიძლება დაიხუროს აპლიკაციები.

### 5.4. პორტებთან მიმართვის შესაძლებლობა

Windows Vista და Windows 7 ოპერაციული სისტემების უსაფრთხოების ამაღლების მიზნით, შსაძლებელია, 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.5.5).




ნახ.5.5. ადმინისტრატორის უფლებით ამუშავება

ფილიალის აპლიკაციის ამუშავებისას აგრეთვე შეიძლება „ადმინისტრატორის უფლებებით“ სარგებლობა.

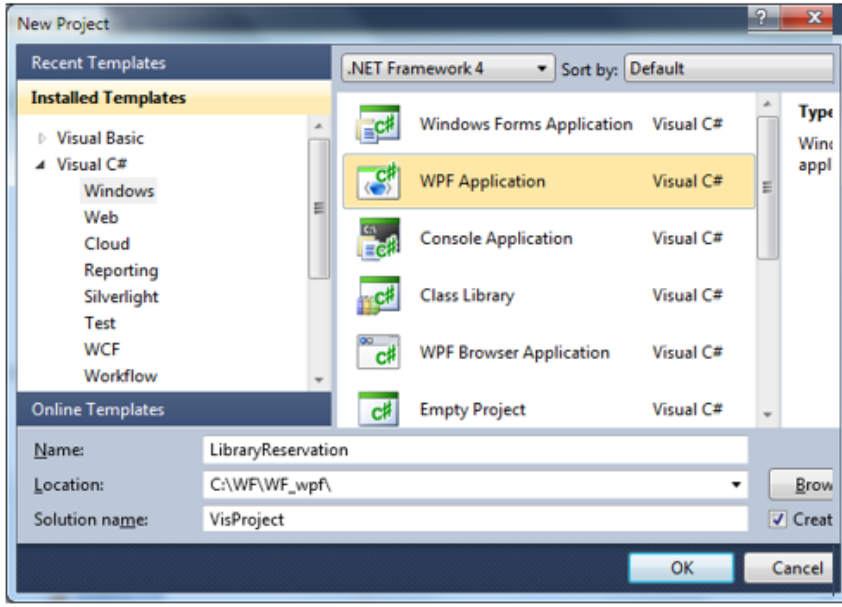
# თავი 6: კომუნიკაცია ჰოსტ-აპლიკაციასთან

ამ თავში ნაჩვენები იქნება 1-ელი თავის პროექტის მსგავსი გადაწყვეტა, ოღონდაც კონსოლის რეჟიმის ნაცვლად გამოყენებულ იქნება Windows Presentation Foundation (WPF) აპლიკაცია [23,25,26].

აქამდე ჩვენ მიერ აგებულ პროექტში ჰოსტი მარტივად იძახებდა სამუშაო პროცესს და დამთავრების შემდეგ ასახავდა შედეგებს. ახალი პროექტისათვის საჭირო იქნება უფრო მეტი კავშირი workflow-პროცესსა და Host-აპლიკაციას შორის. საბედნიეროდ, WF 4.5 აქვს კარგი შესაძლებლობები ამ მიზნის მისაღწევად.

## 6.1. WPF პროექტის შექმნა

1. შევქმნათ ახალი პროექტი WPF აპლიკაციის სახით. პროექტის სახელი იყოს LibraryReservation და Solution-ის VisProject.




ნახ.2.1. 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

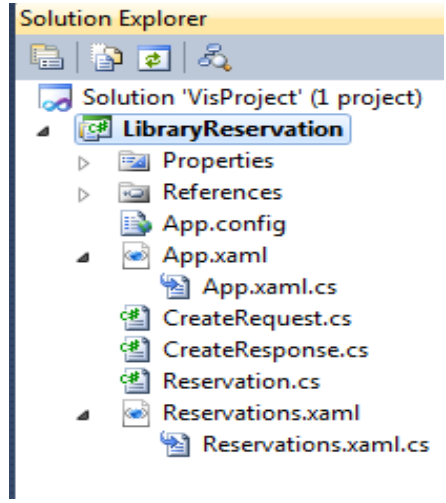
3. Solution Explorer-ში გენერირდება ფაილი სახელით Windows1.xaml, რომელიც შევცვალოთ Reservations.xaml-ით.

App.xaml ფაილი განსაზღვრავს ვინდოუსის startup-ს. ესაა ახლა Windows1 ფაილი, რომელიც უნდა შევცვალოთ ასე:

StartupUri="Reservations.xaml"

## ➢ Class-ების გამოყენება წინა თავიდან

მე-5 თავის LibraryReservation ფოლდერიდან App.config, CreateRequest.cs, CreateResponse.cs და Reservation.cs ფაილები კოპირებით გადმოვიტანოთ VisProject-ის LibraryReservation ფოლდერში. შემდეგ Solution Explorer-ის LibraryReservation–ზე მაუსის მარჯვენა ღილაკით Add -> Existing Item და პროექტს მივუერთოთ კოპირებული ფაილები. მივიღებთ (ნახ.6.2).




ნახ.6.2

- **Window Form –ის განსაზღვრა**

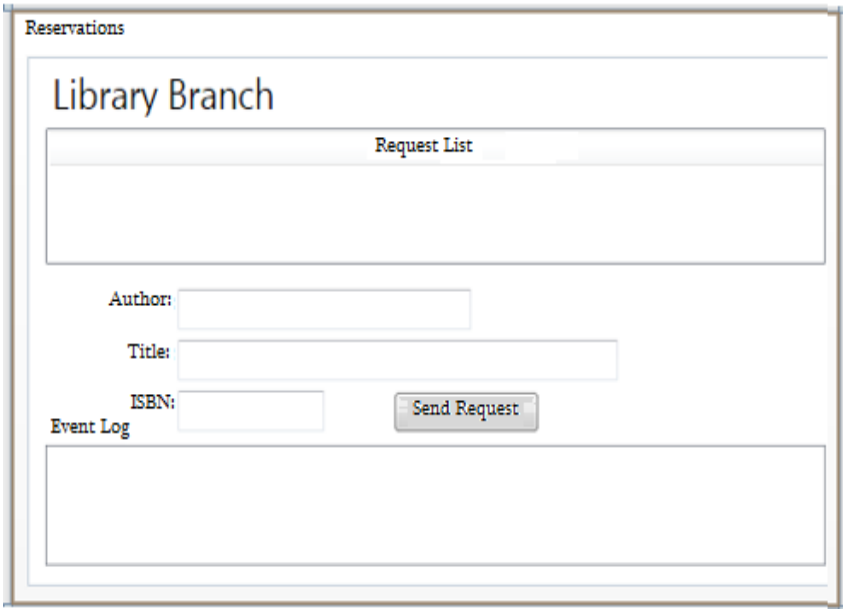
გავხსნათ Reservations.xaml ფაილი და ავირჩიოთ XAML tab. ჩავანაცვლოთ კოდი შემდეგი 6.1 ლისტინგის ტექსტით:

**<!– – ლისტინგი_6.1 – – – –>**

```
<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.6.3).




ნახ.6.3

ფორმის ზედა ნაწილში Request List ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად სხვა ფილიალში გამოიყენება ველები ფორმის შუაში. 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 გამოიტანებოდა 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 6.2 ლისტინგში.

**// --- ლისტინგი_6.2 ----- 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 კონტროლისთვის, რომელმაც უნდა ასახოს ეს მოვლენები. ამ მეთოდს იყენებს ApplicationInterface კლასი. AddEvent() მეთოდს იყენებს აპლიკაცია მაშინ, როცა მას სჭირდება მოვლენის დამატება.

- **ListBoxTextWriter–ის რეალიზაია [ Implementing ListBoxTextWriter]**

დავამატოთ პროექტს კლასი ListBoxTextWriter.cs, რომლის რეალიზაცია 6.3 ლისტინგშია მოცემული.

**// ------ ლისტინგი_6.3 ------ ListBoxTextWriter.cs -------**

```
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 ApplicationInterface სტატიკურ 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 ქმედებისთვის. და სტატიკური ApplicationInterface კლასის გამო, ListBoxTextWriter კლასს შეუძლია წვდომა lstEvents ელემენტზე აპლიკაციის გარედანაც კი.

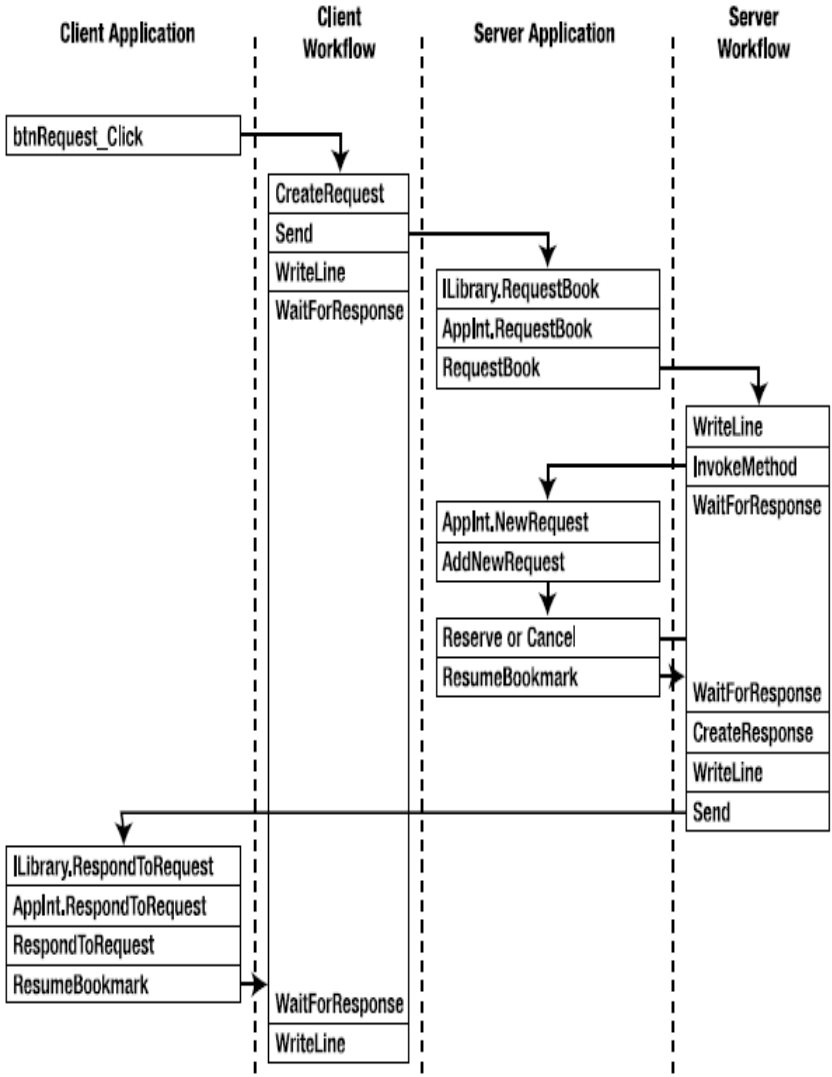
ასე რომ, არსებობს სამი გზა lstEvents მართვის ელემენტში ტექსტის დასამატებლად:

- ✓ აპლიკაციის შიგნიდან, გამოიყენება ლოკალური AddEvent () მეთოდი;
- ✓ აპლიკაციის გარედან, გამოიყენება ApplicationInterface კლასის AddEvent () მეთოდი;
- ✓ WriteLine ქმედებიდან, მიეთითება TextWriter თვისება ListBoxTextWriter-ზე.

- **workflow-პროცესების რეალიზაცია**

6.4 ნახაზზე ნაჩვენებია ზოგადი ლოგიკა და შეტყობინებათა ნაკადი. ამ სქემის ელემენტები მოგვიანებით იქნება ახსნილი, ამჯერად კი განხილულ იქნება ძირითადი კონცეფციები.

ეს მცირეთი განსხვავდება პროცესებისგან, 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.


Client Application
Client Workflow
Server Application
Server Workflow
btnRequest_Click
CreateRequest
Send
WriteLine
WaitForResponse
ILibrary.RequestBook
AppInt.RequestBook
RequestBook
WriteLine
InvokeMethod
WaitForResponse
AppInt.NewRequest
AddNewRequest
Reserve or Cancel
ResumeBookmark
WaitForResponse
CreateResponse
WriteLine
Send
ILibrary.RespondToRequest
AppInt.RespondToRequest
RespondToRequest
ResumeBookmark
WaitForResponse
WriteLine

ნახ.6.4

- **შეტყობინებათა მიღება (Listening for Messages)**

6.4 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 წინა 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. 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 მუშა პროცესთან 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.

- **სერვისის კონტრაქტის რეალიზაცია**

სერვისის კონტრაქტი განსაზღვრავს მხოლოდ ხელმისაწვდომ მეთოდებს, იგი არ უზრუნველყოფს მათ 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. მისი კოდის რეალიზაცია ნაჩვენებია 6.4 ლისტინგში.

// ------- **ლისტინგი 6.4** ------ ClientService.cs ----------

```
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 იქნება 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 უნდა მოთავსდეს კონსტრუქტორის წინ. ასე:

```
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 6.5 ლისტინგში MainWindow კლასისთვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

**//---- ლისტინგი 6.5 ------ *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.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),bBindi
ng, 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, მიბმის და კონტრაქტის სამეულს.

Unloaded მოვლენის დამმუშავებელი უბრალოდ ხურავს ServiceHost-ს, ასე რომ მეტი აღარ მოხდება შეტყობინებების მიღება.

- **სანიშნეები (Bookmarks)**

სანიშნეები იძლევა საშუალებას შეაჩეროს მუშა 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 მუშა 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 WaitForInput.cs კლასი, რომლის 6.6 ლისტინგი მოცემულია ქვემოთ.

// ----- **ლისტინგი 6.6 -------- WaitForInput.cs** ----------

```
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 იქნს 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 ინკაფსულაციისთვის, 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 კლასები შესრულებულია დაცული კლასების სახით, ძირითადად მათი გაფართოების შეზღუდვის მიზნით.

= = = =

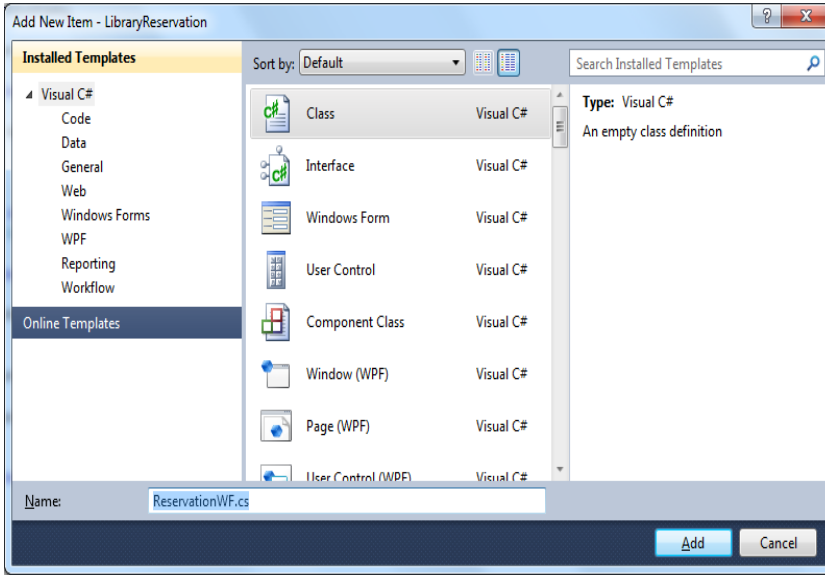
მომხმარებლის ეს ქმედება გამოიყენებს NativeActivity საბაზო კლასს (CodeActivity-ის ნაცვლად), იმიტომ რომ იგი აძლევს მას NativeActivityContext -თან მიმართვის საშუალებას, რომელიც აუცილებელია სანიშნის შესაქმნელად. იგი ასევე იყენებს შაბლონის ვერსიას (კლასში <T> აღნიშვნაა). შემავალი არგუმენტი ასახავს მონაცემებს, რომლებიც გადაეცემა მუშა 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 მუშა პროცესი აღდგენილია, მაშინ ეს უკუ-გამოძახების მეთოდი სრულდება. ყურადსაღებია, რომ უკუ-გამოძახების მეთოდი ობიექტს ღებულობს მესამე პარამეტრის სახით. ესაა მონაცემები, წარმოდგენილი აპლიკაციით. იგი ინახება შემავალ არგუმენტში, რაც ხელმისაწვდომია მუშა პროცესისთვის.

აქტიურობები (ქმედებები), რომლებიც იყენებს სანიშნეებს, უნდა იქნას გადატვირთული (override), რომ CanInduceIdle თვისება აბრუნებდეს true მნიშვნელობას. ეს კი უზრუნველყოფს მუშა პროცესს, რომ გადავიდეს ლოდინის მდგომარეობაში მანამ, სანამ სანიშნით მოხდება მისი აღდგენა.

### 6.2. SendRequest მუშა პროცესის რეალიზაცია

ახლა შევასრულოთ მუშა პროცესების რეალიზაცია. Solution Explorer-ის LibraryReservation-ზე მარჯვენა ღილაკით ავირჩიოთ Add -> Class. სახელი ReservationWF.cs (ნახ.6.5). კოდის რეალიზაცია მოცემულია 6.7 ლისტინგში.




ნახ.6.5

// ---- ლისტინგი_6.7 -------- ReservationWF.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 LibraryReservation
{
 // ეს ფაილი განსაზღვრავს ორ workflow პროცესს:
 // SendRequest - ახალი მოთხოვნის ინიცილიზაცია და
 // ProcessRequest  - შემომავალი მოთხოვნების დამუშავება
  public sealed class SendRequest : Activity
```

```
    {  // შემავალი და გამომავალი არგუმენტების განსაზღვრა
        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()
{
 // ცვლადების განსაზღვრა workflow პროცესისთვის ---
 Variable<ReservationRequest> request =
 new Variable<ReservationRequest> { 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
               {
                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 პროცესის დამატების ადგილი----
}
```

ამ workflow 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 გამოსატანად ეკრანზე. ეს გამოიწვევს ტექსტის ასახვას lstEvents მართვის ელემენტში.

სხვა განსხვავება იმაშია, რომ მომხმარებლის ქმედება WaitForInput გამოიყენება Receive ქმედების ნაცვლად. აპლიკაცია მიიღებს საპასუხო შეტყობინებას უშუალოდ. 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**

ProcessRequest პროცესის განსაზღვრება მოცემულია 6.8 ლისტინგში. ჩავამატოთ ეს კოდი ReservationWF.cs ფაილში.

```
// ------- ლისტინგი_6.8----------------
public sealed class ProcessRequest : Activity
{
 public InArgument<ReservationRequest> request { get; set; }
 public InArgument<TextWriter> Writer { get; set; }
 public ProcessRequest()
  {
   // ამ workflow-ის  ცვლადების განსაზღვრა ----
   Variable<ReservationResponse> response =
     new Variable<ReservationResponse> {Name="response"};
   Variable<bool> reserved = new Variable<bool> {
               Name = "Reserved" };
   Variable<string> address = new Variable<string>
            { Name = "Address" };

   // ProcessRequest პროცესის 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<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 workflow პროცესი განსხვავდება წინა თავში რეალიზებული ვერსიისგან. 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 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 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 წინა 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.

### 6.3. აპლიკაციის რეალიზაცია

შემდეგი ბიჯი არის აპლიკაციის რეალიზაცია. არსებობს მოვლენათა რამდენიმე დამმუშავებელი (event handlers), რომელთა რეალიზაცია აუცილებელია, აგრეთვე მეთოდები, რომლებიც გამოიძახება სტატიკური ApplicationInterface კლასით.

- **მხარდაჭერა მუშა პროცესების ეგზემპლარებისთვის**

აპლიკაცია თვალყურს უნდა ადევნებდეს workflow პროცესის ეგზემპლარებს, ამიტომაც მას შეუძლია გნაახლოს 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 workflow პროცესს SendRequest და ProcessRequest, ამიტომაც საჭირო იქნება ლექსიკონის ორი ობიექტი. დავამატოთ კონსტრუქტორში კოდი ამ ობიექტების ინიციალიზებისთვის:

```
_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 გამომახება, რომელიც ქმნის ReservationRequest კლასს, ალტერნატიული კონსტრუქტორის გამოსაყენებლად, რომელიც ღებულობს მეხუთე პარამეტრს RequestID -თვის. დავამატოთ მუქი სტრიქონი კოდის შემდეგ ტექსტში:

```
// ReservationRequest კლასისი შექმნა და
// საწყისი არგუმენტებით შევსება
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    // !!!
   );
```

- **მოვლენათა დამმუშავებელი (Event Handlers)**

ახალი მოთხოვნის შესაქმნელად მომხმარებელი შეავსებს ***ავტორის, სათაურის, ISBN*** ველებს და აამოქმედებს Send Request ღილაკს. ამ მოვლენის ღილაკის რეალიზება მოცემულია 6.9 ლისტინგში, Reservations.xaml.cs ფაილში.

**// --- ლისტინგი 6.9 -----*Click Event -ის რეალიზება* -----**

```
private void btnRequest_Click(object sender, RoutedEventArgs e)
  {  // ობიექტის ლექსიკონის აწყობა პარამეტრების გადასაცემად---
     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 Workflow-Application-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; მუშა 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 გააგრძელოს რეაგირება მოვლენებზე (და შემავალ შეტყობინებებზე). Run() მეთოდის გამოყენება წყვეტს სწორედ ამ ამოცანას.

- - - - - - - - -

მოთხოვნების სიის ფორმაზე მოთავსებულია ღილაკები Reserve და Cancel, რომლებსაც იყენებს მომხმარებელი იმის მისათითებლად, თუ რომელი ელემენტი იყო გამოყენებული.

6.10 ლისტინგი აღწერს ამ ღილაკებისთვის მოვლენათა დამმუშავებლების რეალიზაციას. დავამატოთ ეს მეთოდები Reservations კლასში.

**// -- ლისტინგი 6.10 -- *Reserve და Cancel ღილაკების რეალიზაცია* ----**

```
// Reserve ღილაკის დაკლიკვის მოვლენის დამმუშავებელი ----
private void Reserve(object sender, RoutedEventArgs e)
 {
  // instanceID მიღება Tag თვისებიდან ----
  FrameworkElement fe = (FrameworkElement)sender;
  Guid id = (Guid)fe.Tag;
  ResumeBookmark(id, true);
 }
```

```
// Cancel ღილაკის დაკლიკვის მოვლენის დამმუშავებელი ---
private void Cancel(object sender, RoutedEventArgs e)
{
   // instanceID მიღება Tag 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);
    }
}
```

ეს მოვლენის დამმუშავებლები იღებს სამუშაო პროცესის ეგზემპლარის ID-ს ღილაკის Tag თვისებიდან. შემდეგ იძახებენ ResumeBookmark() მეთოდს, მიაწოდებს true-ს ან false-ს, იმისდა მიხედვით, თუ რომელი ღილაკი იყო ამოქმედებული.

ResumeBookmark() მეთოდი მიიღებს WorkflowApplication-ს _incomingRequests-კოლექციიდან და გამოიძახებს მის Resume-Bookmark() მეთოდს. გადაეცემა სანიშნის სახელი (bookmark name) და მნიშვნელობა, რომელშიც ეგზემპლარი განახლდება (resumed).

## 6.4. ApplicationInterface მეთოდები

ჩვენ განვსაზღვრეთ ApplicationInterface კლასის სამი მეთოდი. ახლა უნდა უზრუნველვყოთ მათი რეალიზაცია MainWindow კლასში. ამ მეთოდების რეალიზაცია მოცემულია 6.11 ლისტინგში.

**// -- ლისტინგი 6.11-- ApplicationInterface კლასის მეთოდების რეალიზაცია -**

```
public void RequestBook(ReservationRequest request)
 {
  // ობიექტის ლექსიკონის აწყობა პარამეტრების გადასაცემად---
   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 RespondTo-Request მეთოდია მითითებული. ეს ხდება მაშინ, როცა სხვა ფილიალები აგზავნის უკან პასუხს შემოსულ მოთხოვნაზე. იგი ღებულობს WorkflowApplication-ს _outgoingRequests კოლექციიდან და აღადგენს სანიშნეს, გამავალს ReservationResponse კლასში.

AddNewRequest() გამოიძახება ProcessRequest მუშა პროცესით, როცა მიიღება ახალი შეტყობინება. ეს ხდება InvokeMethod ქმედების დახმარებით. იგი უბრალოდ დაამატებს ჩანაწერს ListView-კონტრო-ლის RequestList-ელემენტში. ვინაიდან ის გამოძახებულ უნდა იქნას მუშა პროცესის შესრულებად ნაკადში, Dispatcher კლასი გამოიყენებს შესასრულებლად Add () მეთოდს main window-ის შესრულებადი ნაკადით. Reservations.xaml.cs-ის სრული რეალიზაცია მოცემულია 6.12 ლისტინგში.

// --- ლისტინგი 6.12 --- *Reservations.xaml.cs საბოლოო 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)
{
  // config ფაილის გახსნა და ფილიალის სახელის და
  // ქსელური მისამართის მიწოდება
  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));

  //  Endpoint-ის დამატება ---
  string szAddress = "http://localhost:" + adr + "/ClientService";
  System.ServiceModel.Channels.Binding bBinding = new
                    BasicHttpBinding();
  _sh.AddServiceEndpoint(typeof(ILibraryReservation),
     bBinding, szAddress);
  // ServiceHost-ის გახსნა  შეტყობინებების სიისათვის ---
  _sh.Open();
  // ListBoxTextWriter-ის ტესტირება ---
  //ListBoxTextWriter lbtw = new ListBoxTextWriter();
  //lbtw.Write("This is a test");
}
private void Window_Unloaded(object sender, RoutedEventArgs e)
{
```

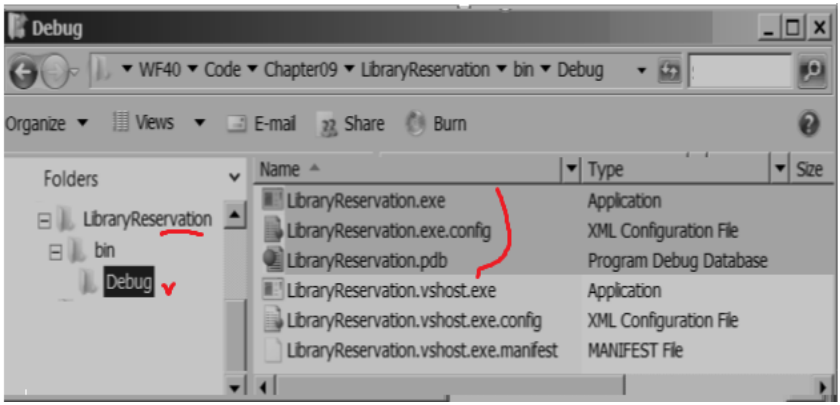
```
        // host-ის სერვისის დასასრული ---
        _sh.Close();
    }
private void btnRequest_Click(object sender, RoutedEventArgs e)
 {
   // ობიექტის ლექსიკონის აწყობა პარამეტრების გადასაცემად---
Dictionary<string, object> parameters = new Dictionary<string, object>();
        parameters.Add("Author", 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();
    }
    //  Reserve ღილაკის დაკლიკვის მოვლენის დამმუშავებელი ----
    private void Reserve(object sender, RoutedEventArgs e)
    {
        // instanceID მიღება Tag თვისებიდან ----
        FrameworkElement fe = (FrameworkElement)sender;
        Guid id = (Guid)fe.Tag;
        ResumeBookmark(id, true);
    }
     //  Cancel ღილაკის დაკლიკვის მოვლენის დამმუშავებელი ----
    private void Cancel(object sender, RoutedEventArgs e)
    {
        // instanceID მიღება Tag 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 RequestBook(ReservationRequest request)
    {
        // ობიექტის ლექსიკონის აწყობა პარამეტრების გადასაცემად---
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);
        }
    }
}
```

## 6.5. აპლიკაციის ამუშავება

როგორც წინა თავის შემთხვევაში, აქაც საჭიროა აპლიკაციის რამდენიმე ასლის (კოპიოს) ერთად გაშვება, თითოეული თავისი კონფიგურაციის ფაილის ვერსიით. თავიდან საჭიროა F6 კლავიშის ამოქმედება Solution-ის აღსადგენად და კომპილატორის შენიშვნების აღმოსაფხვრელად. შევქმნათ ახალი ქვეფოლდერი LibraryReservation-ფოლდერში, რომელიც იძახებს ფილიალებს. შემდეგ დავაკოპიროთ ფილიალის ფოლდერში ფაილები, რომლებიც 6.6 ნახაზზეა ნაჩვენები.



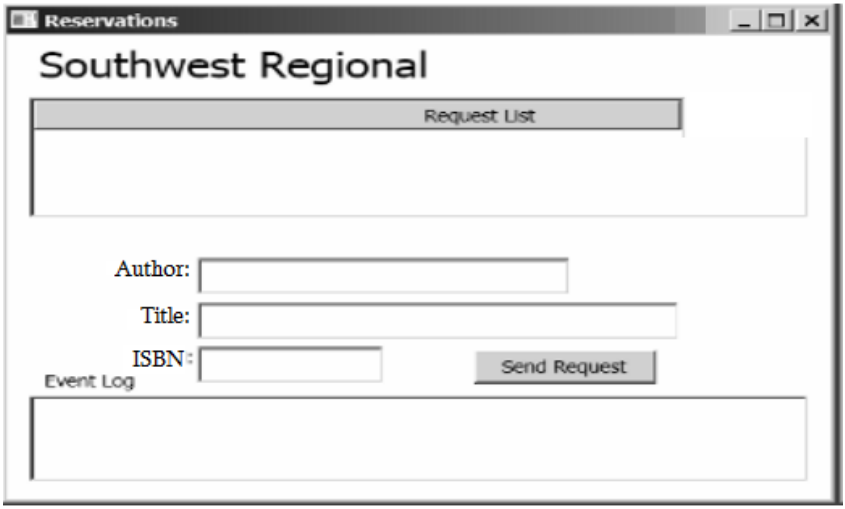

ნახ.6.6

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 (იხ. წინა თავი).

ფილიალის ფოლდერში LibraryReservation.exe ფაილი ორჯერ დავკლიკოთ. აპლიკაცია ასე გამოიყურება (ნახ.6.7).

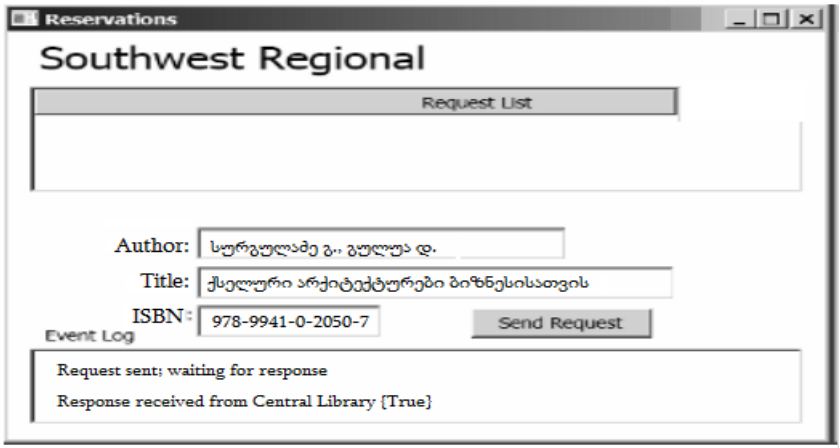



ნახ.6.7

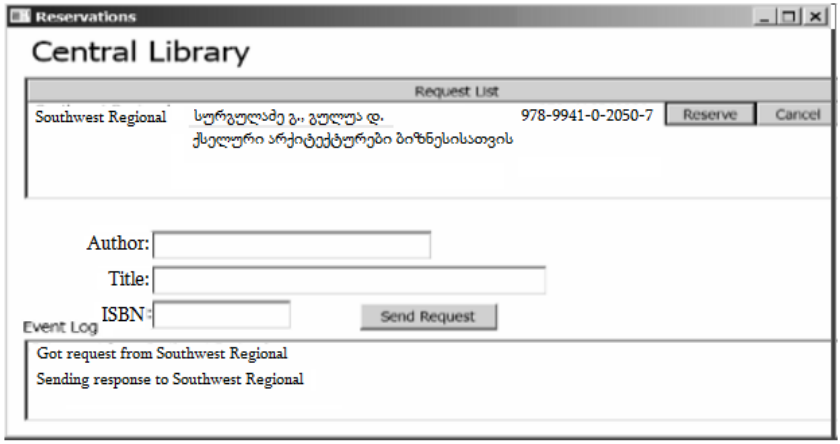
Visual Studio-ში F5-ით გავმართოთ აპლიკაცია. ანალოგიური ფანჯარა უნდა მივიღოთ, ოღონდ სათაურით - ცენტრალური ბიბლიოთეკა. ფანჯრები განვაცალკევოთ, ხედვის გასაუმჯობესებლად.

ერთ-ერთ აპლიკაციაში შევიტანოთ 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 6.8 და 6.9 ნახაზების სახე.




ნახ.6.8. მოთხოვნის გაგზავნა ფილიალი ბიბლიოთეკიდან




ნახ.6.9. მოთხოვნის დამუშავება ცენტრალურ ბიბლიოთეკაში

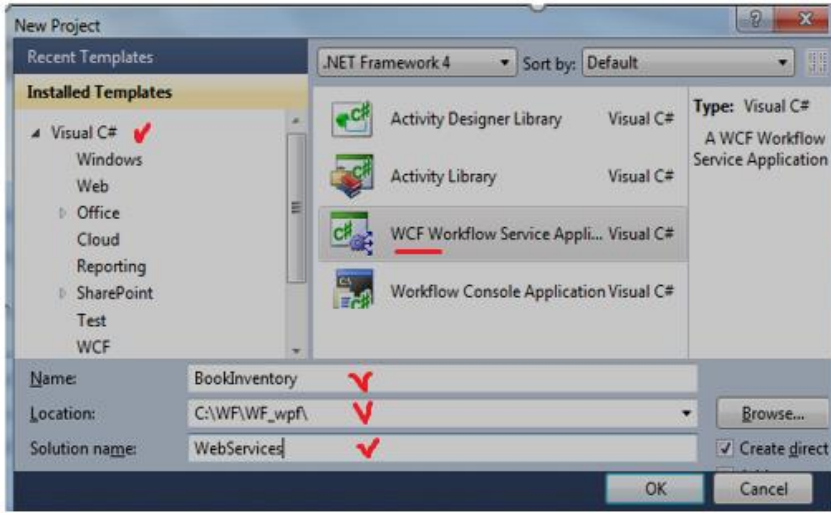
ვცადოთ რამდენიმე მოთხოვნის გაგზავნა ორივე ფანჯრიდან. ასევე შევამოწმოთ Cancel ღილაკი და დავრწმუნდეთ, რომ საპასუხო შეტყობინება მოვლენათა ჟურნალში (მეორე აპლიკაციისთვის) არის [false].

# თავი 7. Web-სერვისები

Workflow (ბიზნეს-) პროცესები შეიძლება განთავსდეს ვებ-სერვისში, რომელიც უზრუნველყოფს იდეალურ საშუალებას სამუშაო პროცესის გადაწყვეტილების მისაწოდებლად არა-სამუშაო პროცესის კლიენტებისათვის, 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) ტექნოლოგიაში, ჩვენ შეგვიძლია ადვილად შევქმნათ WCF სერვისები [20,21,25].

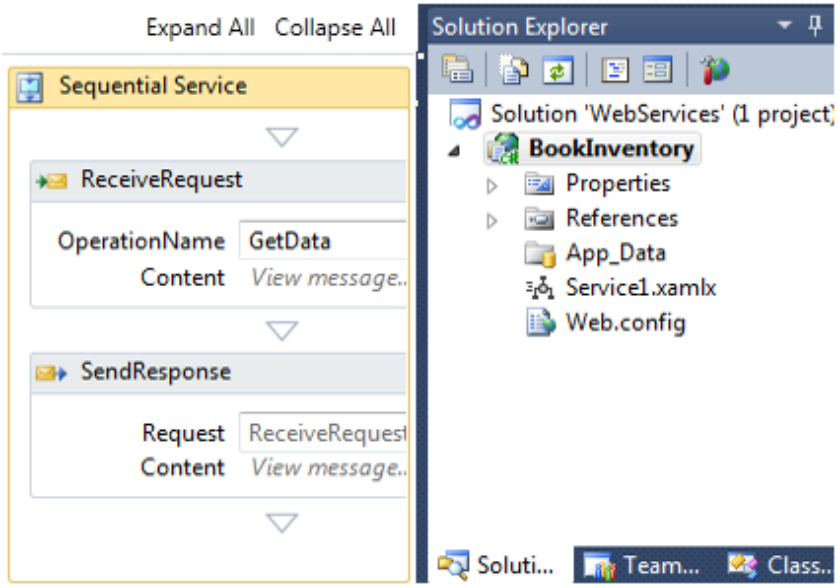
## 7.1. Workflow-ის სერვისის შექმნა

ავამუშავოთ Visual Studio 2015 და შევქმნათ ახალი პროექტი WCF Workflow Service Application template გამოყენებთ. შევიტანოთ პროექტის სახელი BookInventory და Solution-ის სახელი WebServices.




ნახ.7.1. WCF Workflow Service აპლიკაციის შექმნა

შეიქმნება საინიციალიზაციო workflow Sequence ბლოკი, რომელიც შეიცავს Receive და SendReply ქმედებებს, როგორც 7.2 ნახაზზეა ნაჩვენები.




ნახ.7.2. საწყისი ბიზნესპროცესის 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 შექმნის საწყის ბიზნესპროცესს ფაილში 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”.*

## 7.2. სერვისის კონტრაქტის განსაზღვრა

Solution Explorer-ში, მარჯვენა ღილაკით BookInventory პროექტზე ავირჩიოთ: Add->Class სახელით BookInfo.cs, რომლის ტექსტი მოცემულია 7.1 ლისტინგში.

```
// ----- ლისტინგი 7.1 --- BookInfo.cs ---
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
namespace BookInventory
{
  // სეერვისის კონტრაქტის განსაზღვრა: IbookInventory,
  // რომელიც შეიცავს ერთ LookupBook() მეთოდს----
  [ServiceContract]
  public interface IBookInventory
    {
        [OperationContract]
        BookInfoList LookupBook(BookSearch request);
    }
     // BookSearch მოთხოვნის შეტყობინების განსაზღვრა ---
    [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,
    // 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 სახადასხვა თვისებები, საჭირო წიგნის მოსაძებნად, მაგალითად, ავტორს და დასახელებას. ის აბრუნებს უკან BookInfoList კლასს, რომელიც შეიცავს BookInfo კლასების კოლექციას.

F6 ამოქმედებით აიგება Solution.

*) 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–შრისათვის რათა სწორად დაფორმატდეს SOAP შეტყობინება.

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 თუ 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.

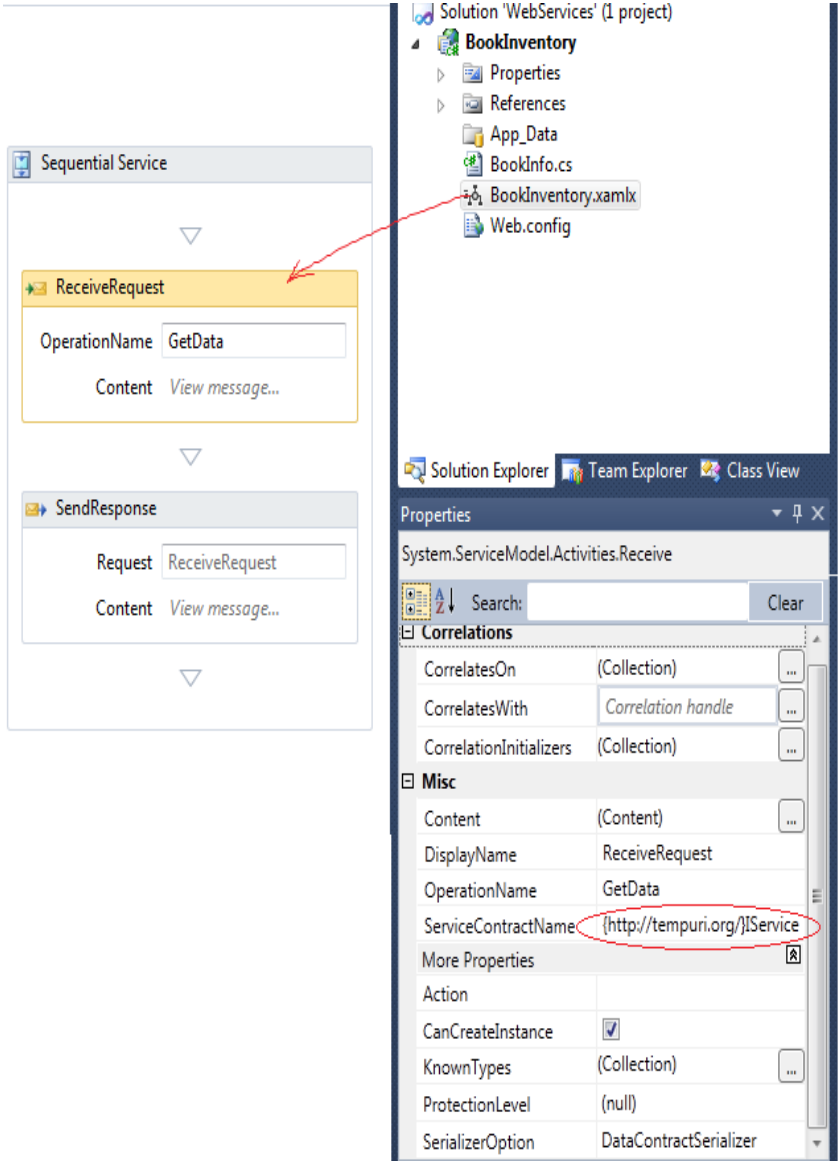
**კონტრაქტი** განისაზღვრება ServiceContract–ით, რომელიც არის ინტერფეისი. იგი განსაზღვრავს მეთოდებს, რომლებიც მიწვდომადია ბოლო წერტილში.

ამგვარად, ჩვენ განვსაზღვრეთ შეტყობინებები, რომლებიც გადაიცემა სერვის–მეთოდების მიერ პარამეტრების სახით.

### 7.3. Receive და SendReply კონფიგურირება

გავხსნათ BookInventory.xamlx ფაილი და ავირჩიოთ ქმედება ReceiveRequest (ნახ.7.3).

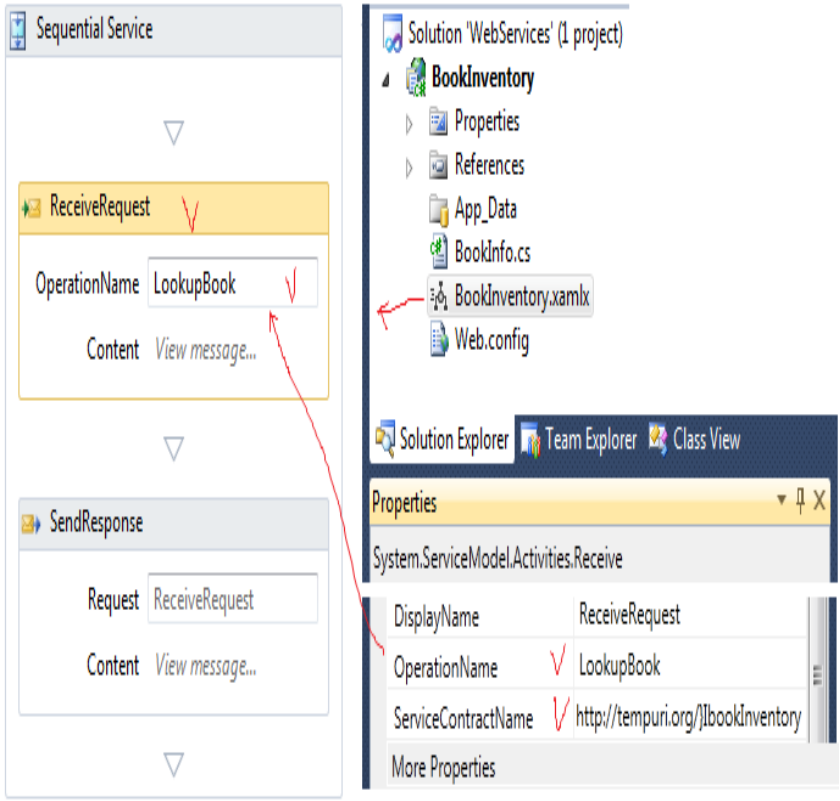
თვისებათა ფანჯარაში ServiceContract თვისებას აქვს default-მნიშვნელობა {http://tempuri.org/}IService. შევცვალოთ Iservice სტრიქონი IbookInventory-ით. შევიტანოთ OperationName როგორც LookupBook.




ნახ.7.3. საწყისი მდგომარეობა

ეკრანზე WorkflowService დიზაინერში, ქვედა მარცხენა კუთხეში დავკლიკოთ Variables ღილაკი. გამოჩნდება შაბლონი ორი ცვლადის შესაქმნელად (ნახ.7.4).

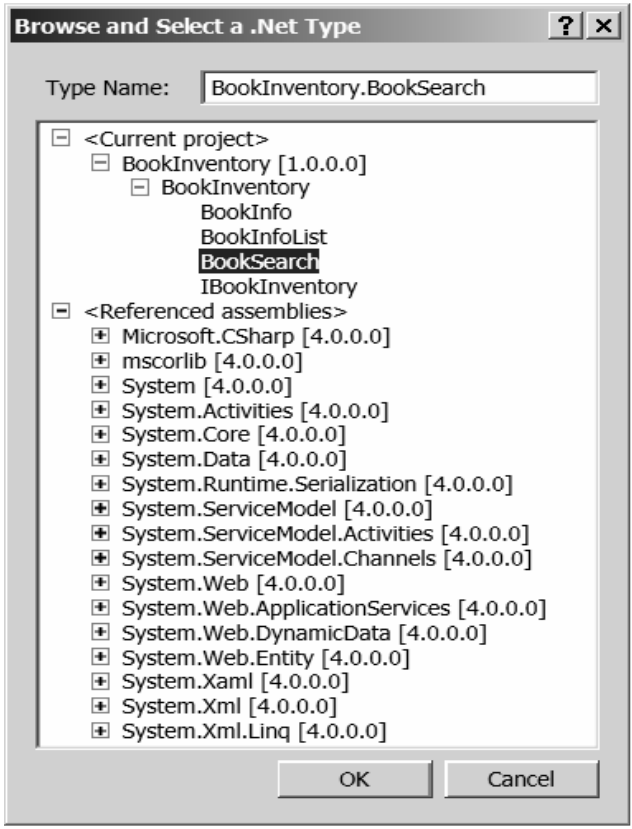
გადასამუშავებელი ცვლადი (handle variable) გამოიყენება პასუხის კორელაციისთვის იმ ეგზემპლართან, რომელმაც გააგზავნა მოთხოვნა.




ნახ.7.4. საბოლოო მდგომარეობა

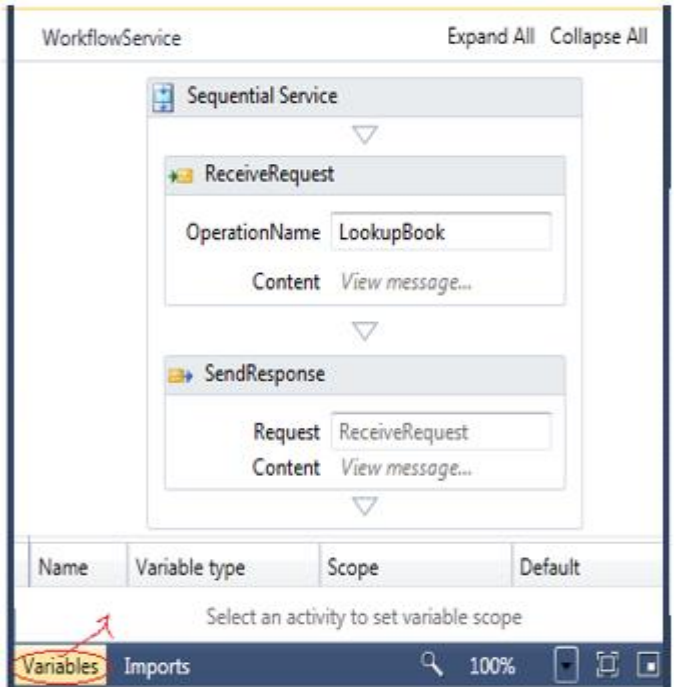
მონაცემთა ცვლადი იქმნება გადასაცემი მონაცემების (ინფორმაციის) მიზნით. გავასუფთავოთ ცვლადების არე (data) და შევქმნათ ორი ახალი ცვლადის სახელი.

პირველისთვის ავირჩიოთ სახელი Name search და ტიპი - Browse for Typs. ახალ დიალოგურ ფანჯარაში BookInventory ნაკრები გავაფართოვოთ BookSearch-ით (ნახ.7.5).




ნახ.7.5

მეორე ცვლადისათვის შევიტანოთ Name: result. ტიპი შეირჩევა Browse-დან, BookInfoList კლასით. მიიღება 7.6 ნახაზზე მოცემული შემთხვევა.




ნახ.7.6-ა. 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

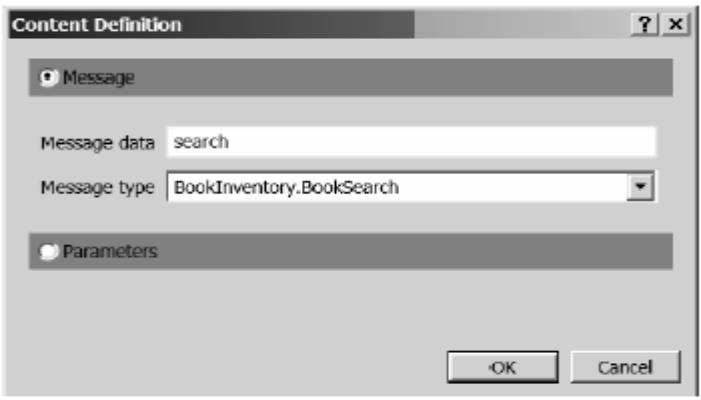
Variables Imports 100%

ნახ.7.6-ბ. ცვლადები განისაზღვრა სამუშაო პროცესისთვის

მუშა პროცესების დიზაინერში „ReceiveRequest“ (მოთხოვნების მიღების) ქმედებას აქვს view message (შეტყობინების ნახვის) ლინკი შინაარსის თვისებისათვის. მისი ამოქმედებით იხსნება დიალოგური ფანჯარა შემავალი შეტყობინების დასადგენად (შეიძლება ასევე სამწერტილიანი ღილაკის გამოყენებაც, თვისების გვერდით). შესასვლელი განისაზღვრება ორი ხერხით: შეტყობინებით ან პარამეტრების ერთობლიობით.

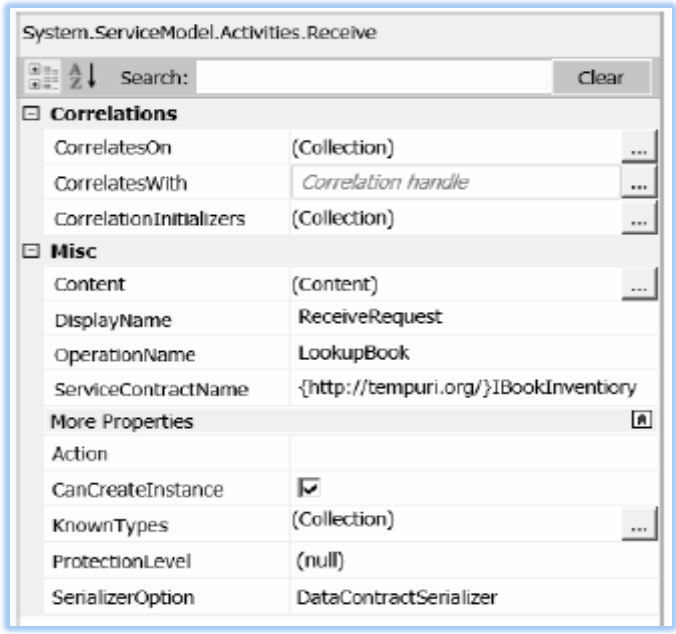
ამ თავში, მოგვიანებით ჩვენ განვიხილავთ მეორე ხერხსაც. ახლა დავაკვირდეთ, რომ რადიობუტონის გადამრთველი შეტყობინებისთვის სწორადაა არჩეული.

Message data თვისებისათვის შევიტანოთ **search.** ის მიუთითებს, რომ შემომავალი შეტყობინება უნდა ინახებოდეს search ცვლადში. Message ტიპისთვის ვირჩევთ BookInventory.BookSearch. დიალოგური ფანჯარა მოცემულია 7.7 ნახაზზე.




ნახ.7.7. შემავალი შეტყობინების განსაზღვრა

თვისებების ფანჯარა უნდა გამოიყურებოდეს 7.8 ნახაზზე ნაჩვენები სახით.



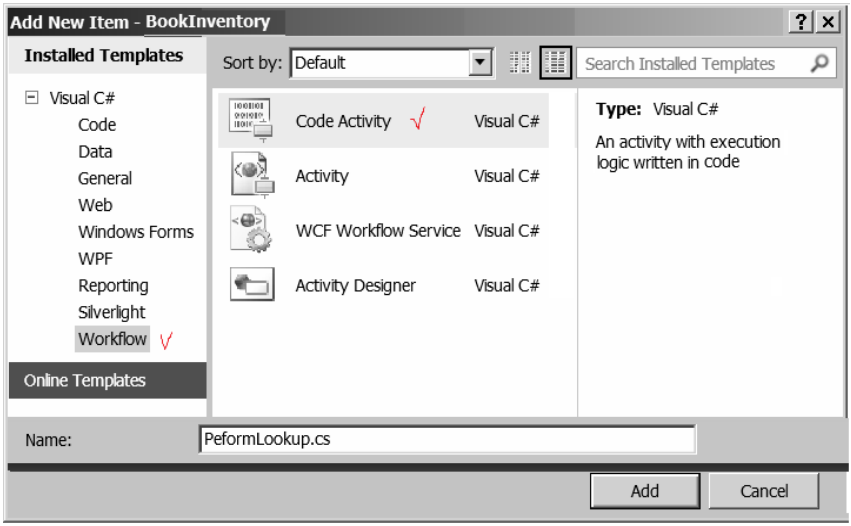

**ნახ.7.8. Receive ქმედების თვისებათა ფანჯარა**

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.

## 7.4. PerformLookup აქტიურობის შექმნა

ამ პროექტისთვის შევქმნით მომხმარებლის ქმედებას (აქტიუ-რობას) “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 კატეგორიისთვის ვირჩევთ Code Activity-ს. Name-ში შევიტანთ PerformLookup.cs სახელს (ნახ.7.9).




**ნახ.7.9. მომხმარებლის ქმედების შექმნა**

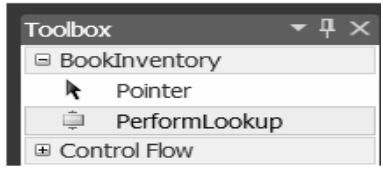
შევიტანოთ PerformLookup ქმედების რეალიზაციისთვის 7.2 ლისტინგში მოცემული კოდი.

**//--- ლისტინგი_7.2 ----- PerformLookup -------**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;
namespace BookInventory
{
   // მომხმარებლის ქმედება ქმნის BookInfoList კლასს,
   // რომელიც არის BookInfo კლასის კოლექცია.  იგი იყენებს
```

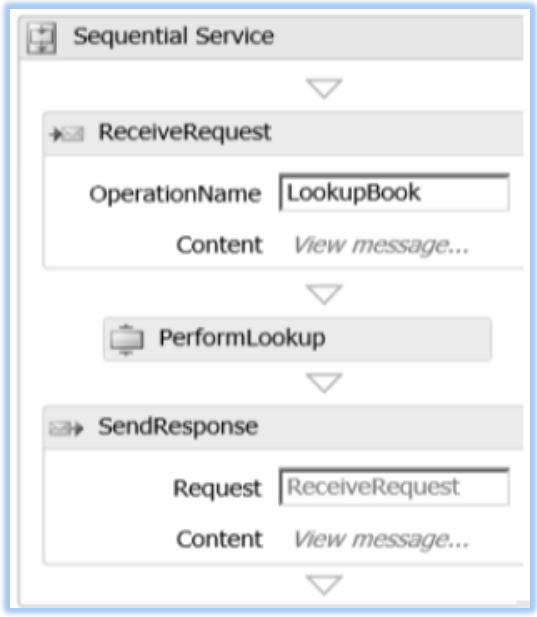
```
        // (BookSearch კლასის) შემავალ პარამეტრებს
        //  შესაბამისი ელემენტების მოსაძებნად ("lookup").
        // BookInfoList კლასი ბრუნდება გამომავალ პარამეტრში
    public sealed class PerformLookup : CodeActivity
    {
     public InArgument<BookSearch> Search { get; set; }
     public OutArgument<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.7.10).



ნახ.7.10

გადმოვიტანოთ PerformLookup ქმედება „ReceiveRequest” და „SendResponse” ქმედებებს შორის, როგორც ეს 7.11 ნახაზზეა ნაჩვენები.




ნახ.7.11

ავირჩიოთ PerformLookup ქმედება. Properties-ის ფანჯარაში, BookList თვისებისთვის შევიტანოთ result; Search თვისებისთვის კი search.

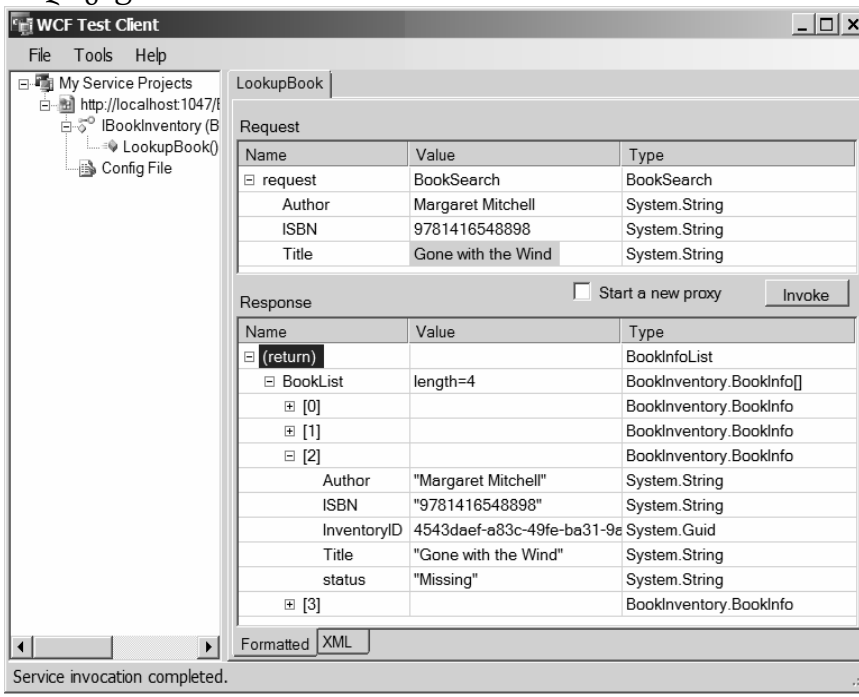
## 7.5. სერვისის ტესტირება

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 7.12 ნახაზის მარცხენა პანელზე.

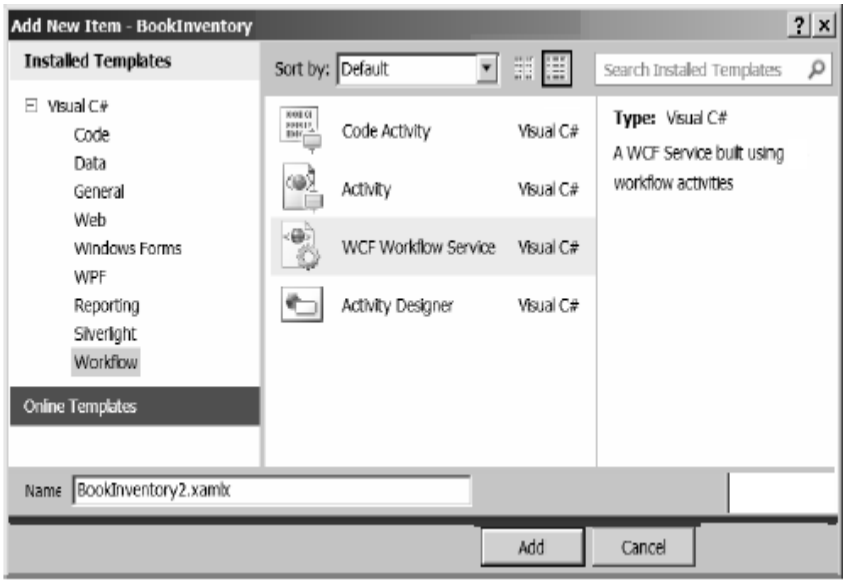
LookupBook() მეთოდზე მაუსის 2-ჯერ დაკლიკვით მარჯვენა პანელის ზედა ნაწილში გამოიყოფა ადგილი შემოსული შეტყობინების განსათავსებლად. იგი მზადაა რთული შეტყობინებებისათვისაც, რომლებიც შეიცავს კლასების და თვისებების კოლექციებს.

შევიტანოთ ავტორი, ISBN-ნომერი, დასახელება. შემდეგ ავამოქმედოთ Invoke ღილაკი. გამოჩნდება შედეგები, რომლებიც ანალოგიურია 7.13 ნახაზის.




ნახ.3.13. WCF Client ტესტის სერვისის შედეგების ნახვა

სერვისი აბრუნებს BookInfo-ს ოთხ კლასს. 7.14 ნახაზზე მესამე ჩანაწერი გაფართოებულია, რათა დავინახოთ დაბრუნებული მონაცემების მაგალითი. მოცემულ კონკრეტულ ელემენტს აქვს სტატუსი Missing (დაკარგული, ამოვარდნილი).




ნახ.7.14. WCF მუშა პროცესის სერვისის შექმნა

შენიშვნა: თუ .xamlx ფაილი არის მომდინარე 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.

## 7.6. პარამეტრების გამოყენება

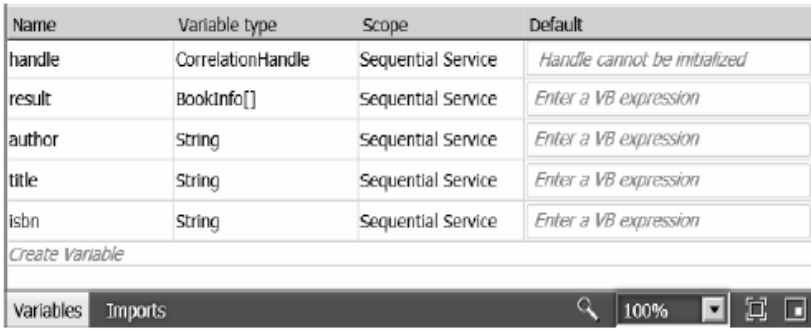
წინა პროექტის მიხედვით ვებსერვისში შესვლა განისაზღვრებოდა როგორც კლასი MessageContract ატრიბუტით. ესაა ტიპური გზა WCF სერვისის გამოსაძახებლად. მიუხედავად ამისა, იმის მაგივრად, რომ შეიქმნას შეტყობინების ერთი კლასი, რომელიც ყველა შემავალ მონაცემს შეიცავდეს, შესაძლებელია მათი გადაცემა ბიზნეს პროცესების სერვისებისათვის ცალკეული პარამეტრების სახით. ამის სადემონსტრაციოდ შევქმნათ მეორე იდენტური სერვისი, რომელიც გამოიყენებს პარამეტრებს შეტყობინებათა მაგივრად.

## 7.7. 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. 7.15 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 ტიპისათვის (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. 7.15 ნახაზზე ნაჩვენებია ცვლადების სია.




ნახ.7.15. ცვლადების სია

Properties-ის ფანჯარაში ServiceContract თვისებისათვის, შევცვალოთ IService კონტრაქტი წიგნით. ოპერაციის სახელში ჩავწეროთ LookupBook2.

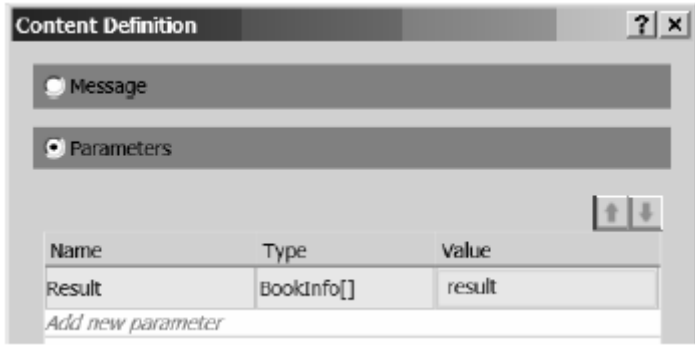
შენიშვნა: პირველი სერვისის შექმნისას CanCreateInstance თვისება დაყენდა true-ში შაბლონით. მეორე სერვისისთვის იგი არის false-ში. შეამოწმეთ და დარწმუნდით, რომ ის გადაყვანილია true-ში.

ავირჩიოთ „ReceiveRequest" ქმედება (მოთხოვნის მიღება) და ავამოქმედოთ Content ლინკი. ამჯერად დავაყენოთ გადამრთველი პოზიციაში „პარამეტრები".

პარამეტრები ყენდება ცვლადების 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 (ნახ.7.16).


ნახ.7.16. ReceiveRequest პარამეტრების სია

დავაჭიროთ „SendResponse" ქმედების Content ლინკს. ავირჩიოთ გადამრთველი პარამეტრები და დავაჭიროთ ლინკს „Add new parameter". შევიტანოთ Name როგორც Result; ტიპისთვის ავირჩიოთ BookInventory.BookInfo[] ჩამოსაშლელი სიიდან. დიალოგურ ფანჯარას აქვს 7.17 ნახაზზე ნაჩვენები სახე.




ნახ.7.17. SendResponse პარამეტრების სია

## 7.8. მოდიფიცირებული PerformLookup ქმედების შექმნა

მომხმარებლის PerformLookup ქმედება, რომელიც ჩვენ შევქმენით პირველი სერვისისთვის, 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. შაბლონიდან ვირჩევთ Code Activity-ს და სახელისათვის Name: PerformLookup2.cs. ამ ქმედების რეალიზაცია ნაჩვენებია 7.3 ლისტინგში.

**// ლისტინგი_ 7.3 ----PerformLookup2.cs. რეალიზაცია----**

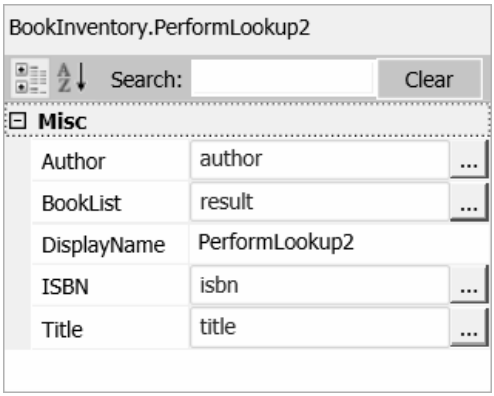
```
using System;
using System.Collections.Generic;
using System.Activities;
namespace BookInventory
{
   //  მომხმარებლის ქმედება ქმნის BookInfo  მასივს და
   //  იყენებს შემავალი პარამეტრების  მოსაძებნად ("lookup").
   // BookInfo  მასივი ბრუნდება გამომავალ პარამეტრსი.
 public sealed class PerformLookup2 : 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 „Receive-Request" და „SendResponse" ქმედებებს შორის. თვისებების ფანჯარაში შევიტანოთ შესაბამისი მნიშვნელობები, როგორც 7.18 ნახაზზეა ნაჩვენები.

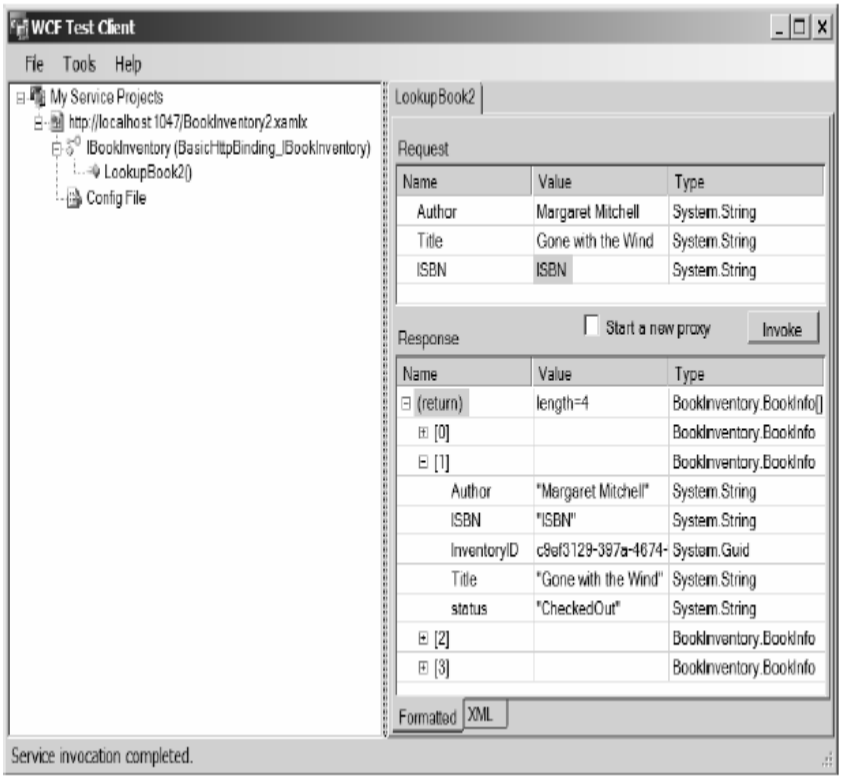



**ნახ.3.18. PerformLookup2 ქმედების Properties ფანჯარა**

## 7.9. 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 ამოქმედდეს ისე, როგორც პირველი სერვისის შემთხვევაში. 2-ჯერ დავკლიკოთ LookupBook2() მეთოდი, შევიტანოთ მოთხოვნის მონაცემები, და დავაჭიროთ Invoke ღილაკს. შედეგებს ექნება 7.19 ნახაზზე ნაჩვენები სახე.




ნახ.7.19. WCF Test Client

ფორმატი მცირედით განსხვადება პირველი სერვისისგან, მაგრამ ფუნქციონირებს ძირითადად მის მსგავსად. ნახაზზე გაფართოებულია მეორე ჩანაწერი, რათა გამოჩნდეს, რომ შემოწმებულ იქნა ეს ეგზემპლარი.

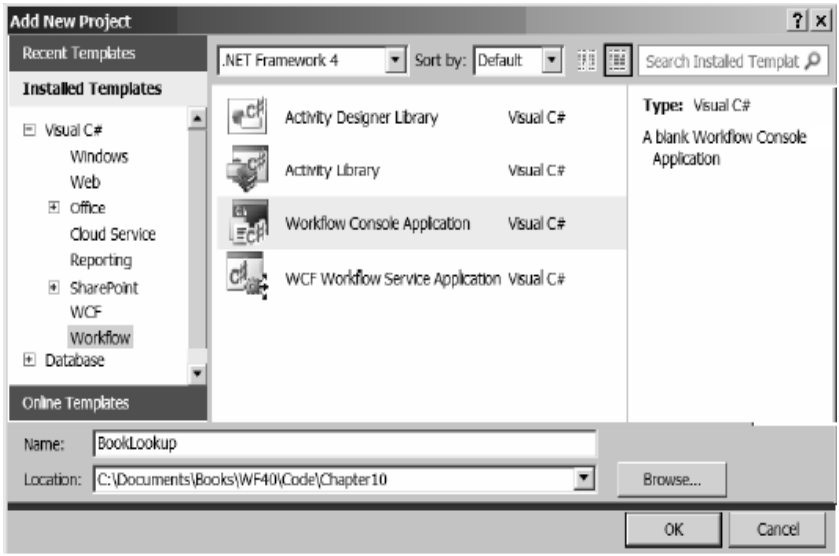
მეორე სერვისისათვის არ შეგვიქმნია კონტრაქტი მომსახურებისთვის. ჩვენ მხოლოდ განვსაზღვრეთ პარამეტრები, რომლებიც გადასცა და უკან დაიბრუნა სერვისმა. კონტრაქტი სერვისის მიწოდებისთვის იქმნება ავტომატურად.

შენიშვნა:

Receive/SendReply წყვილის განსაზღვრისას, გვეძლევა არჩევანის უფლება: შეტყობინებები 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. წინააღმდეგ შემთხვევაში ფიქსირდება საკმაოდ ხანგრძლივი განსაკუთრებული შემთხვევა შესრულების პროცესში.

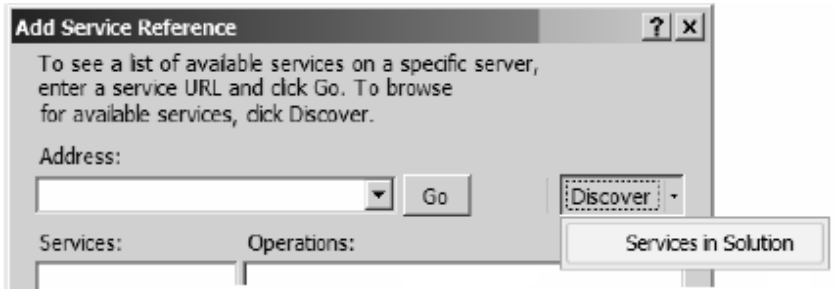
## 7.10. კლიენტის ბიზნესპროცესის შექმნა

ახლა უნდა შევქმნათ Client Workflow , რომელიც გამოიძახებს Web-სერვისებს. Solution Explorer-ში მარჯვენა ღილაკით ვირჩევთ Add -> New Project. შემდეგ ავირჩევთ Workflow Console Application და შეგვაქვს პროექტის სახელი: BookLookup (ნახ.7.20).



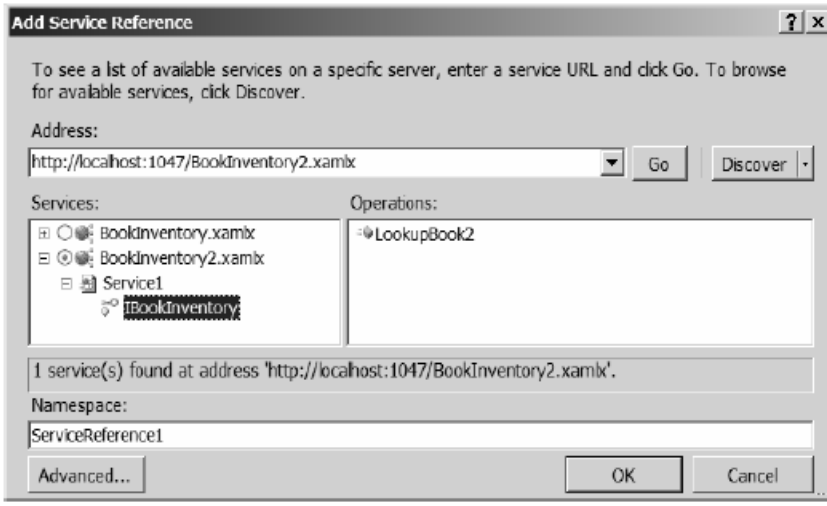

ნახ.7.20. console application სამუშაო პროცესის დამატება

Solution Explorer-ში BookLookup project-ზე მარჯვენა ღილაკის დაჭერით და Add Service Reference არჩევით, უნდა მივიღოთ 7.21 ნახაზი.



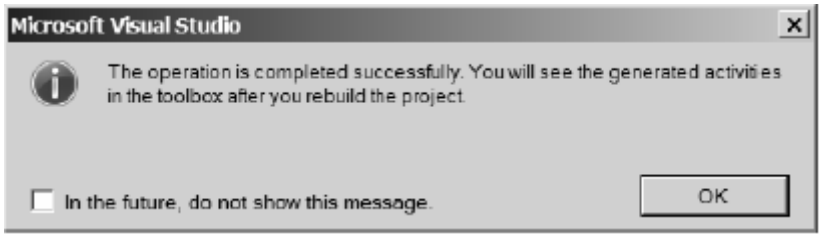

ნახ.7.21. არსებული სერვისების ძებნა

დავაჭიროთ Discover ლინკის ღილაკს და ავიჩიოთ სერვისები Solution-ში. 7.22 ნახაზზე დიალოგის სიაში ჩანს ორი ჩვენ მიერ შექმნილი სერვისი BookInventory პროექტში.



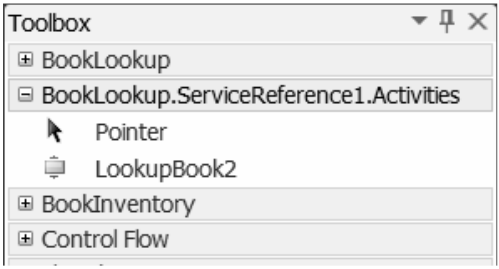

ნახ.7.22. სასურველი სერვისის მოძებნა

შესაძლებელია ამ სერვისების გაფართოვება, რათა დანახულ იქნას მეთოდები, გათვალისწინებული თითოეულში. ავირჩიოთ მეორე, BookInventory2.xamlx და OK. რამდენიმე წამის შემდეგ უნდა გამოჩნდეს დიალოგური ფანჯარა (ნახ.7.23).




ნახ.7.23. 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 7.24 ნახაზის მაგვარი.




ნახ.7.24. განახლებული Toolbox სერვისების გარსით

ინსტრუმენტების პანელზე BookLookup.ServiceReference1 .Activities სახელსივრცე შეიცავს მომხმარებლის ქმედებას ყოველი მეთოდისთვის სერვისში. ჩვენ შემთხვევაში არის მხოლოდ ერთი: LookupBook2.

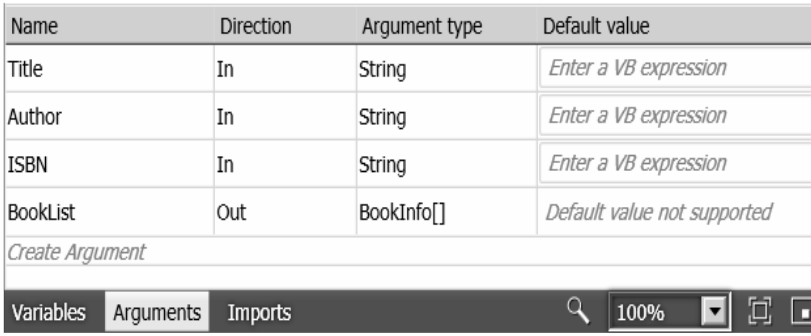
Solution Explorer-ში მაუსის მარჯვენა ღილაკით BookLookup პროექტზე დავაჭიროთ და ავირჩიოთ Set as Startup Project.

## 7.11. მუშა პროცესის განსაზღვრა

გადავიტანოთ LookupBook2 ქმედება მუშა პროცესზე. შემდეგ საჭიროა არგუმენტების დაყენება რათა შესაძლებელი იყოს საძებნი კრიტერიუმების გადაცემა მუშა პროცესში და შედეგების დაბრუნება. დააჭირეთ Arguments მართვის ელემენტს.

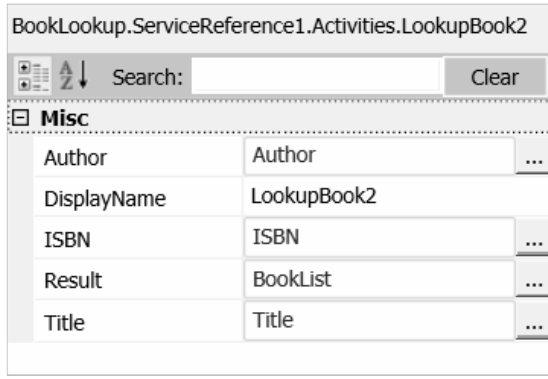
დავამატოთ სამი String შემავალი არგუმენტი სახელით Title, Author და ISBN. დავამატოთ გამომავალი არგუმენტი , სახელით BookList. არგუმენტის ტიპისთვის ავირჩიოთ Array Of[T]. გამოსულ დიალოგურ ფანჯარაში ვირჩევთ Browse for Types და ავირჩევთ

BookInfo კლასს BookLookup.ServiceReference1.BookInventory ნაკრებიდან. არგუმენტების სია მოცემულია 7.25 ნახაზზე.




ნახ.7.25. მუშა პროცესის არგუმენტები

ავირჩიოთ “LookupBook2” ქმედება; Properties ფანჯარაში შევიტანოთ თვისებების values , როგორც 7.26 ნახაზზეა ნაჩვენები.




ნახ.7.26. LookupBook2 თვისებები

### 7.12. ჰოსტ აპლიკაციის რეალიზაცია

გავხსნათ Program.cs ფაილი LookupBook პროექტში. ამ ფაილის რეალიზაცია ნაჩვენებია 7.4 ლისტინგში.

**//---- ლისტინგი 7.4 --- 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)
    {
     // ლექსიკონის შექმნა შემავალი არგუმენტებით workflow-სთვის
       IDictionary<string, object> input = new Dictionary<string, object>
        {
           { "Author" , "სურგულაძე გ., გულუა დ." },
           { "Title" , "ქსელური არქიტექტურები ბიზნესისათვის" },
           { "ISBN" , "978-9941-0-050-7" }
        };

      //  workflow-ის შესრულება ---
     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. სამუშაო პროცესი აბრუნებს უკან BookInfo ობიექტების მასივს. ამ კოდს გამოაქვს ეკრანზე ამ მასივის შინაარსი.

## 7.13. 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
```

= = = = =

## დასკვნა

ბიზნესპროცესები (workflows) ხშირად განაწილებულია სხვადასხვა აპლიკაციაში და სხვადსხვა სერვერზეც კი. ამიტომაც კომუნიკაცია არის სამუშაო პროცესის დიზაინის მნიშვნელოვანი ნაწილი. პროექტში, მაგალითად, ბიბლიოთეკის სხვადასხვა ფილიალები ურთიერთქმედებს ერთმანეთთან, რათა მოითხოვოს 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, ჰოსტ-აპლიკაციას შეუძლია ასევე მიიღოს WCF-შეტყობინება უშუალოდ.

მიუხედავად იმისა, რომ ბიზნესპროცესის ქმედებებს არ აქვს მომხმარებლის ინტერფეისი, ისინი ხშირად საჭიროებს ურთიერთობას ჰოსტაპლიკაციასთან, ან აპლიკაციის განახლებას, ან მოითხოვოს მონაცემები, რომლებიც შეიტანება მომხმარებელთა მიერ. ამ თავში ვიყენებდით სანიშნეს (Bookmark), რათა შეჩერებულიყო სამუშაო პროცესი მომხმარებელთა მონაცემების შეტანამდე. აპლიკაციას ადვილად შეუძლია სამუშაო პროცესის განახლება, სადაც იგი შეწყდა. გამოიყენებოდა ასევე WriteLine ქმედება, კონსოლზე ტექსტის გამოსატანად. შემდეგ კი 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. მუშა პროცესი შეიძლება გამოყენებულ იქნას როგორც სერვისის შესაქმნელად და მათ გამოსაყენებლად. მეთოდები, რომლებიც უზრუნველყოფილია Web-სერვისებით, ხდება ტრადიციული, სტანდარტული ქმედებები, რომელთა გამოყენება შესაძლებელია მუშა პროცესებში.

# თავი 8. კორპორაციული Web-აპლიკაციების პროგრამული რეალიზაცია

წინამდებარე თავებში გადმოცემული იყო ორგანიზაციული მართვის ბიზნესპროცესების ავტომატიზაციის მიზნით, კერძოდ ბიბლიოთეკის საპრობლემო სფეროს მაგალითზე, კლიენტ-სერვერული და სერვისორიენტირებული არქიტექტურის სისტემების დაპროგრამების საფუძვლები. ამჯერად შემოთავაზებულ იქნება სხვადასხვა კორპორაციული ობიექტისათვის Web-აპლიკაციის მოდელირების და პროგრამული რეალიზაციის მაგალითები. ბიზნესპროცესების ქსელური სისტემების ასაგებად განხილულია მოდელირების, დაპროექტების და დაპროგრამების თანამედროვე ჰიბრიდული ტექნოლოგიები [20-22,26-28].

## 8.1. მარკეტინგული ბიზნესპროცესების მენეჯმენტის მხარდამჭერი საინფორმაციო სისტემა

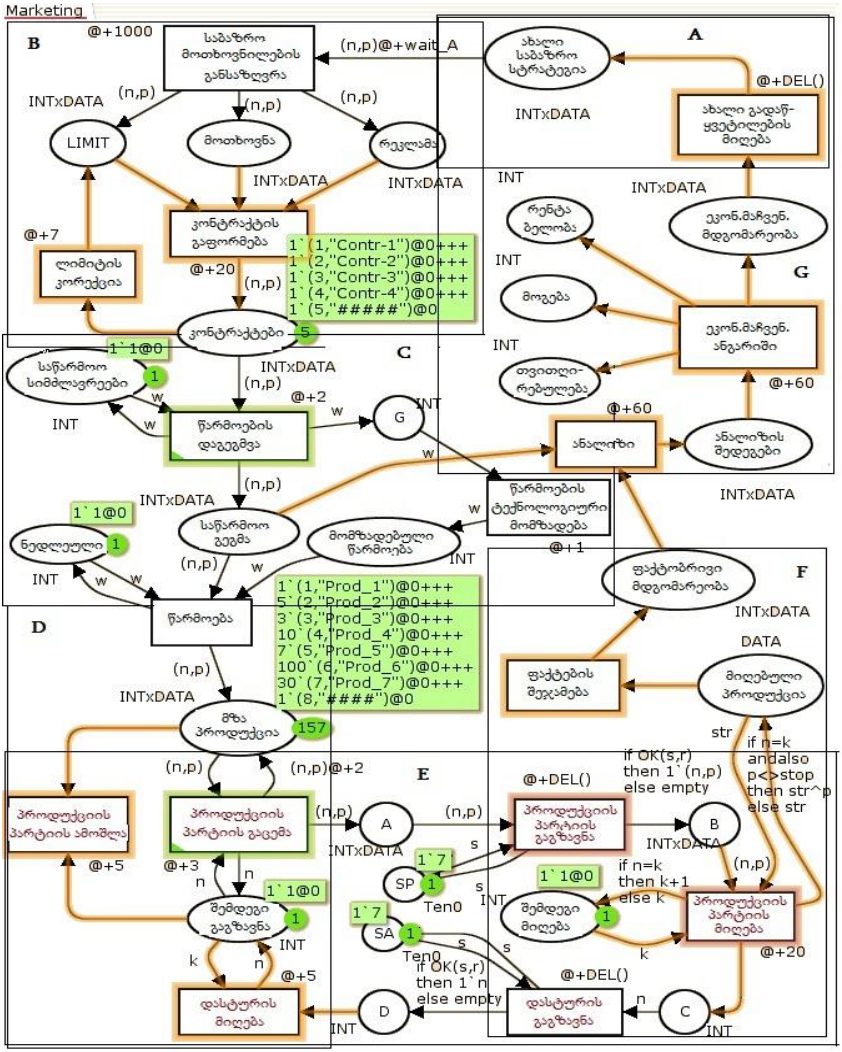
### ➢ მარკეტინგული პროცესების CPN- მოდელი

პროდუქციის მწარმოებელ ფირმებში საინფორმაციო ტექნოლოგიების გამოყენება ხდება როგორც წარმოების დაგეგმვის, აღრიცხვის, ანალიზისა და კონტროლის ამოცანების გადასაწყვეტად, ისე საკონსტრუქტორო და ტექნოლოგიური პროცესების შესასრულებლად. მენეჯერის ძირითადი ფუნქციაა გადაწყვეტილების მიღება კონკრეტული სიტუაციის ანალიზის საფუძველზე, თავისი პროფესიონალური გამოცდილების, თეორიულ-პრაქტიკული მომზადებისა და ხშირად ინტუიციის გამოყენებით.

წარმოების მენეჯმენტი დამოკიდებულია ობიექტურ და დროულ ინფორმაციაზე, რომელიც გროვდება, მუშავდება, ინახება და ვრცელდება თანამედროვე სამეცნიერო მეთოდებისა და ტექნიკური საშუალებების გამოყენებით.

მარკეტინგული მენეჯმენტის პროცესების ზოგადი წარმოდგენის მიზნით ჩვენ განვიხილავთ 8.1 ნახაზზე ნაჩვენებ

საილუსტრაციო სქემას, რომელიც აღწერს მარკეტინგული ბიზნეს-პროცესების მოდელს პეტრის ფერადი ქსელების, კერძოდ CPN-ინსტრუმენტის გამოყენებით [30-32].




ნახ.8.1. მარკეტინგული პროცესების CPN-მოდელი

პროდუქციის საწარმოო ფირმის მარკეტინგული პროცესების მოდელირებისათვის გვაქვს შემდეგი ძირითადი იერარქიული მოდულები: 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 სტრატეგიისთვის (A) და ა.შ. ციკლურად [32].

ჩვენი მიზანია ზემოაღწერილი იერეარქიული მოდულებიდან გამოვყოთ, მაგალითად, E-ბლოკი, რომელიც აღწერს პროდუქციის მიწოდების პროცესს კლიენტებზე, და მოვახდინოთ „მიმწოდებელი-დამკვეთი" ობიექტებისთვის პროდუქციის მიწოდების პროცესის მოდელირება შეტყობინებების გაცვლის იმიტაციით, კლიენტ-სერვერ არქიტექტურის პრინციპების საფუძველზე.

- **პროგრამული რეალიზაცია**

ბიზნესპროცესების შესრულებისას ერთ–ერთი მნიშვნელოვანი საკითხია კომუნიკაცია „მიმწოდებელ-დამკვეთ" აპლიკაციებს შორის, ან კლიენტებსა და სერვერებს შორის. ასეთი კავშირების რეალიზაცია შესაძლებელია ბიზნესპროცესებსა და ჰოსტ-დანართებს შორის.

აპლიკაციის მაგალითის სახით განიხილება პროექტის აგება 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,

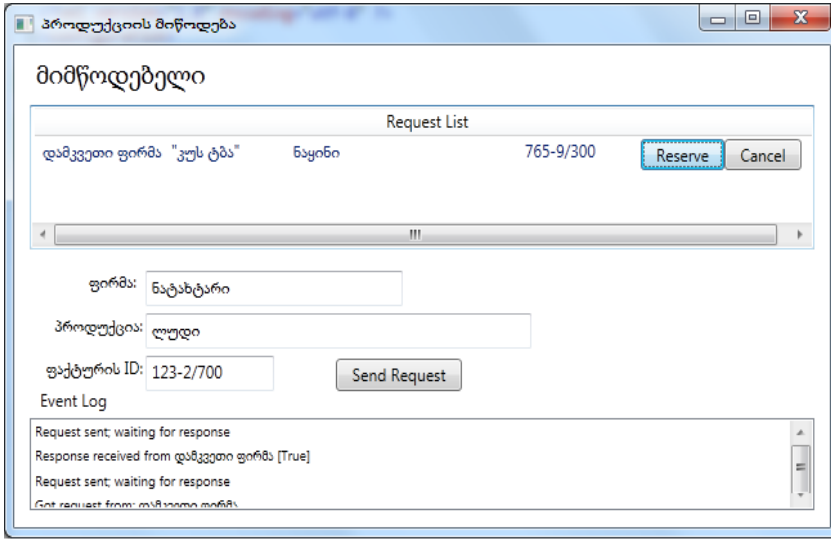
რითაც ეს „ტრანზაქცია” დასრულებულად ითვლება. მიმწოდებლის იგივე აპლიკაციას შეუძლია მოთხოვნის გაგზავნა სხვა დამკვეთთან და ასევე საპასუხო შეტყობინების მიღება მათგან. თუ საპასუხო შეტყობინება არ დაბრუნდა მიმწოდებელთან, ეს ნიშნავს, რომ შეკვეთა არაა შესრულებული და შესაძლებელია პროდუქციის იგივე პარტია კვლავ გაიგზავნოს დამკვეთთან.

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 [21]. ჩვენ ავაგებთ მარტივ WPF-აპლიკაციას (Windows Presentation Foundation), რომელიც გამოიყენებს კომუნიკაციას, მაგალითად ორ სხვადასხვა აპლიკაციის (მიმწოდებელი და დამკვეთი) ბიზნესპროცესებს შორის [23].

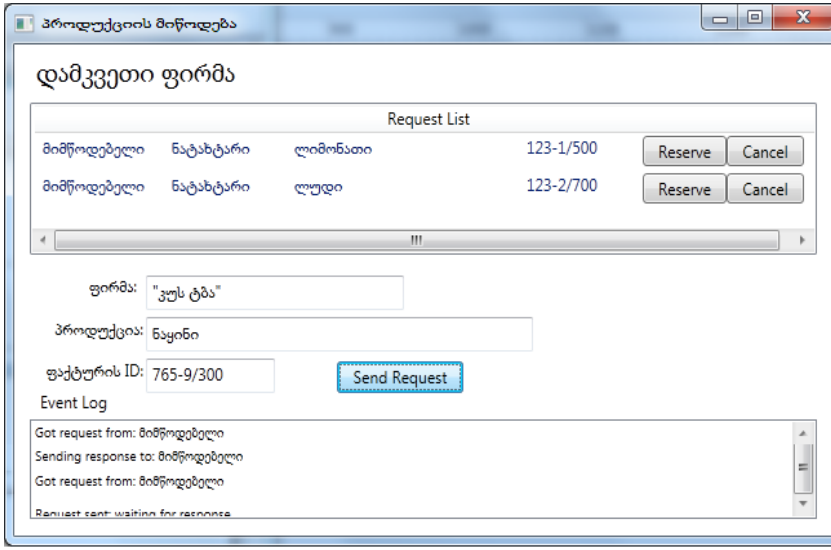
8.2 და 8.3 ნახაზებზე მოცემულია საილუსტრაციო ფრაგმენტები „მიმწოდებელ-დამკვეთ˜ აპლიკაციებს შორის, თუ როგორი შეიძლება იყოს მათი ინტერფეისები. მაგალითად, ფირმა „ნატახტარი” აგზავნის შეკვეთილი „ლიმონათის პარტიას”, ფაქტურით „123-1/500, ამოქმედდა „Send Request” ღილაკი და დამკვეთი ფირმის ინტერფეისზე „Request List”-ში გამოჩნდა სტრიქონი ამის შესახებ.

პრინციპში, ეს ნიშნავს პროდუქციის ადგილზე მიტანას (ჩვენ მაგალითში ეს მყისიერად მოხდა, თუმცა შესაძლებელია გარკვეული დაყოვნების დროის გამოყენებაც, შემთხვევით რიცხვთა გენერატორის დახმარებით, რადგან პროცესი სტოქასტურია) [20]. 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.




ნახ.8.2. მიმწოდებლის ინტერფეისი




ნახ.8.3. დამკვეთის ინტერფეისი

ეს შეტყობინება მიმწოდებლის ინტერფეისზე აისახება მოვლენათა რეგისტრაციის, „Event Log" ლისტბოქსში. 8.1 ლისტიგნში მოცემულია „მიმწოდებლის" ინტერფეისის პროგრამული აპლიკაციის კონფიგურაციის ფაილი (App.config). აქ ყურადსაღებია პორტის ნომრები (Address, Request Address), რომლებიც „დამკვეთი ფირმისთვის" იგივეა, ოღონდ შებრუნებული.

<-- ლისტინგი_8.1: --- 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.8.2 და 8.3). ინფორმაციის მომზადება და გადაცემა, ასევე შეტყობინების გაგზავნა შესაძლებელია ორივე მიმართულებით. 8.2 და 8.3 ლისტინგებში ნაჩვენებია შესაბამისი პროგრამული კოდები.

// -- ლისტინგი_8.2: --- 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_8.3: --- 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);
    }
   }
}
```

8.4 ლისტინგში მოცემულია კლიენტის სერვისის კლასის კოდის ფრაგმენტი.

**// -- ლისტინგი_8.4: --- 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);
      }
   }
}
```

ამგვარად, თანამედროვე საინფორმაციო ტექნოლოგიების საფუძველზე, როგორიცაა მაიკროსოფთის WPF, Workflow და WCF პაკეტები, შესაძლებელია როგორც მომხმარებელთა მოქნილი და ეფექტური ინტერფეისების დაპროექტება, ასევე მობილური კლიენტ-სერვერ და სერვის-ორიენტირებული სისტემების აგება.

## 8.2. საკრედიტო ბიზნესპროცესები მენეჯმენტის მხარდამჭერი საინფორმაციო სისტემა

აპლიკაციის მაგალითის სახით განიხილება პროექტის აგება საწარმოო ფირმებსა და ბანკებს შორის, რომელშიც მოთხოვნილი ინფორმაცია (მაგალითად, იურიდიული პირის მიერ საკრედიტო განაცხადის წარდგენა) გადაეცემა ბანკს. იგივე აპლიკაციას შეუძლია მოთხოვნის გაგზავნა სხვა (სხვა ბანკში) და ასევე პასუხის გაცემა სხვა ორგანიზაციების მოთხოვნებზე [33].

მთავარი ქმედებები, რომლებიც განხილულ სისტემაში კომუნიკაციისათვის გამოიყენება, აქაც არის Send და Receive ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply). ეს ქმედებები გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და სამეთვალყურეოდ [20-25].

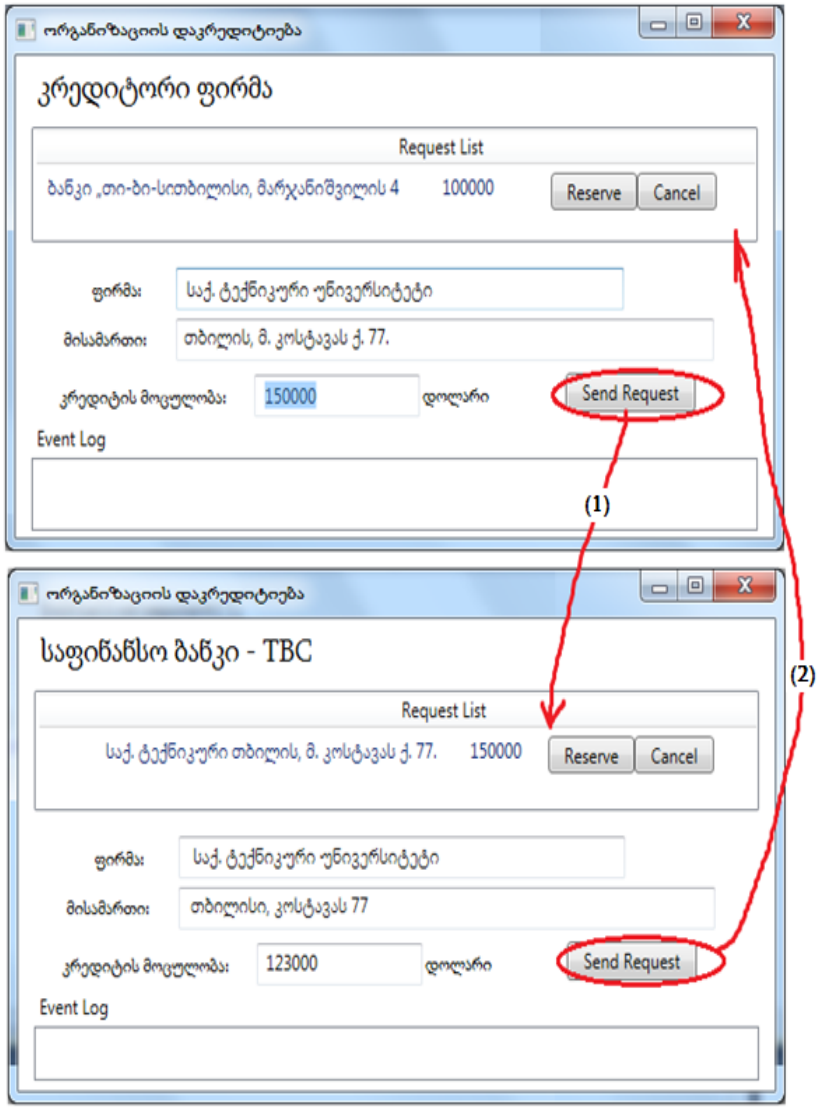
წინა პარაგრაფის მსგავსად ავაგებთ მარტივ WPF-აპლიკაციას. იგი გამოიყენებს კომუნიკაციას სხვადასხვა აპლიკაციათა ბიზნესპროცესებს შორის, რომლებიც შეიძლება განთავსდეს ვებ-სერვისში, რაც უზრუნველყოფს იდეალურ საშუალებას ბიზნეს პროცესის გადაწყვეტილების მისაწოდებლად არა-მუშა პროცესის კლიენტებისათვის, როგორიცაა ვებ-აპლიკაციები.

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.

- **Window Form –ის განსაზღვრა**

ავაგოთ მომხმარებელთა ინტერფეისის ფორმა, რომელიც გამოდგება, მაგალითად ბანკებისთვის. მისი მაკეტი მოცემულია 8.4 ნახაზზე. შესაბამისი ფორმა აგებულია XAML ფაილის სახით დიზაინის რეჟიმში, როგორც ზემოთ აღვნიშნეთ, WPF ტექნოლოგიის გამოყენებით.

ინტერფეისის დაპროექტება და პროგრამული რეალიზაცია ხდება Visual Studio.NET Framework 4.0/5 გარემოში, C#.NET ენის საფუძველზე. გამოიყენება ინტერფეისის დიზაინისთვის XAML და პროგრამის ლოგიკისთვის C# ენები. 8.5 ლისტინგში მოცემულია XAML ფაილის ტექსტი.




ნახ.8.4. მომხმარებლის ინტერფეისები

Crediting.xaml ფაილის 1_ლისტინგის ტექსტი ასეთია:

```
<!----- ლისტინგი_8.5 ---------->
<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 დაკრედიტება" 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.

- **სერვისის პროგრამული რეალიზაცია**

ჩვენი სისტემის ClientService.cs კოდის რეალიზაცია ნაჩვენებია 8.6 ლისტინგში.

```
// -------- ლისტინგი_8.6 ---- 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 ჩვენს მიერ. ყოველი მეთოდი უბრალოდ იძახებს ApplicationInterface კლასის შესაბამის მეთოდს.

ApplicationInterface.cs ფაილი მოცემულია 8.7 ლისტინგში.

**// ---- ლისტინგი_8.7--- 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 ინება ამ მეთოდების რეალიზება Crediting.xaml.cs ფაილში.

- **ServiceHost -ის რეალიზაცია**

აპლიკაციისთვის აუცილებელია ServiceHost-ის რეალიზაცია შემავალი შეტყობინებების მისაღებად. იგი თავსდება პროექტის Crediting.xaml.cs ფაილში კონსტრუქტორის წინ კლასის წევრის სახით. ტექსტი მოცემულია 8.8 ლისტინგში.

**//---- ლისტინგი_8.8----------**

```
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 8.9 ლისტინგში MainWindow კლასისთვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

**// --- ლისტინგი 8.9----------**

```
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 შეტყობინებების 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, მიბმის 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 აღსადგენად და კომპილატორის შენიშვნების აღმოსაფხვრელად.

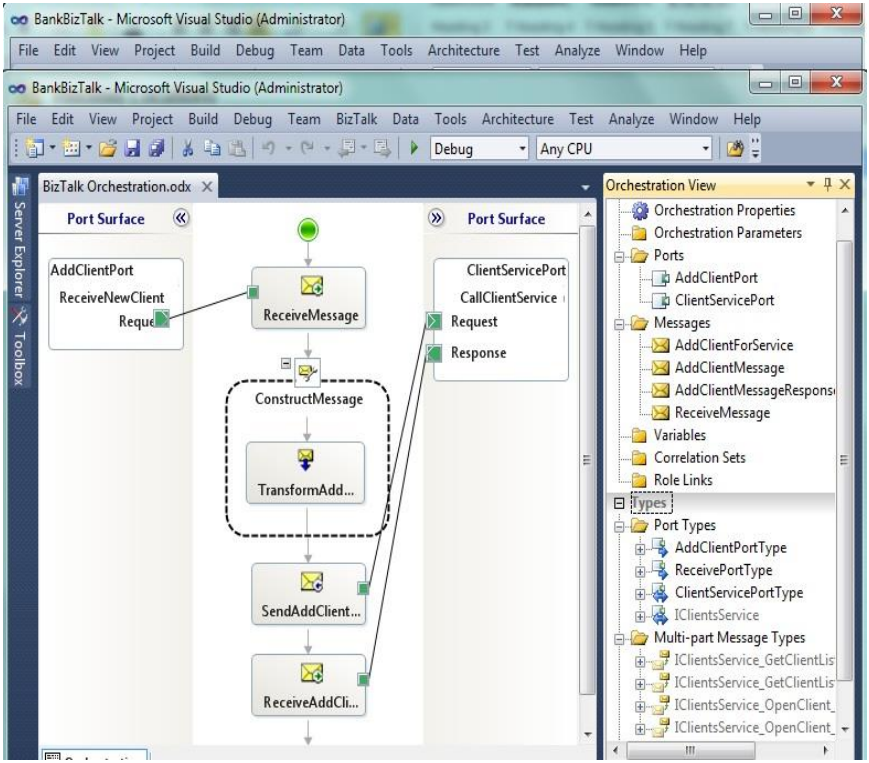
შევქმნათ ახალი ფოლდერი BankRequest -ფოლდერის ქვეშ, რომელიც იძახებს ბანკებს. შემდეგ დავაკოპიროთ ბანკის ფოლდერში ფაილები, რომლებიც ზემოთ შევქმენით;

```
FirmsCrediting.exe    // Application
FirmsCrediting.exe.config   // XAML Configuration file
FirmsCrediting.pdb   // program debug database
```

სისტემის ამუშავების შემდეგ გვექნება ორი მუშა ფანჯარა, მაგალითად, ერთი კრედიტორი ფირმის, მეორე საფინანსო ბანკის. მათ შორის შესაძლებელი იქნება ინფორმაციის და შეტყობინებების გაცვლა, რისი მიღწევაც გვინდოდა (ნახ.8.4).

### 8.3. სერვის-აპლიკაციების პროგრამული რეალიზაცია და გამოყენება Ms BizTalk Server გარემოში

Ms BizTalk სერვერი საშუალებას იძლევა მიიღოს და დაამუშაოს შეტყობინებები [34]. აპლიკაციაში დამატებული ორკესტრაციის ფაილი Bank_orchestration.odx.




ნახ.8.5. Ms BizTalk Server-ის ორკესტრაციის ფაილი

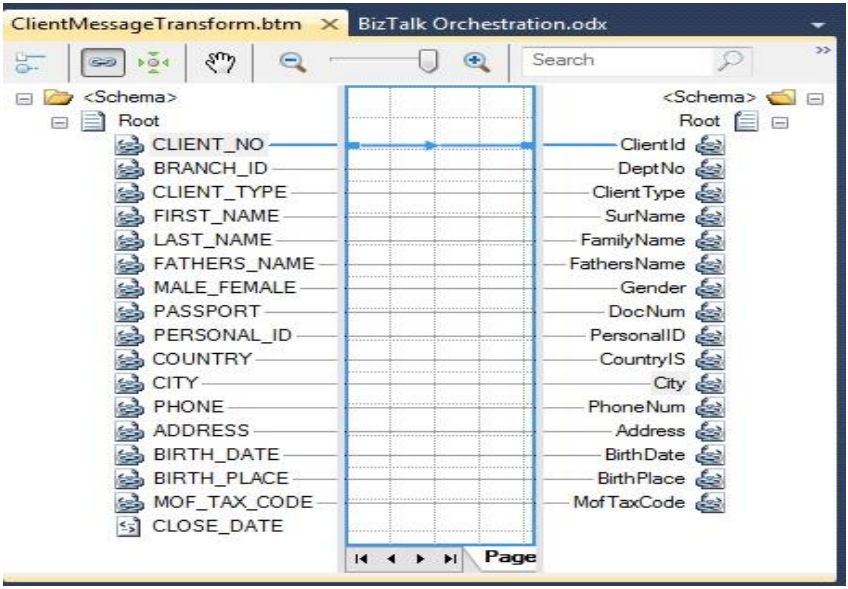
შეტყობინებების მიღება ხდება XML ფორმატში. XML-ის დასამუშავებლად იქმნება სქემის ფაილი Clients.xsd.

```
<?xml version="1.0" encoding="utf-16"?>
    <xs:schema xmlns="http://BankBizTalk.Client"
        xmlns:b="http://schemas.microsoft.com/BizTalk/2003"
        targetNamespace="http://BankBizTalk.Client"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="Root">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="CLIENT_NO" type="xs:unsignedInt" />
         <xs:element name="BRANCH_ID" type="xs:short" />
         <xs:element name="CLIENT_TYPE" type="xs:short" />
         <xs:element name="FIRST_NAME" type="xs:string" />
         <xs:element name="LAST_NAME" type="xs:string" />
         <xs:element name="FATHERS_NAME" type="xs:string" />
         <xs:element name="MALE_FEMALE" type="xs:boolean" />
         <xs:element name="PASSPORT" type="xs:string" />
         <xs:element name="PERSONAL_ID" type="xs:string" />
         <xs:element name="COUNTRY" type="xs:string" />
         <xs:element name="CITY" type="xs:string" />
         <xs:element name="PHONE" type="xs:string" />
         <xs:element name="ADDRESS" type="xs:string" />
         <xs:element name="BIRTH_DATE" type="xs:string" />
         <xs:element name="BIRTH_PLACE" type="xs:string" />
         <xs:element name="MOF_TAX_CODE" type="xs:string" />
         <xs:element name="CLOSE_DATE" type="xs:dateTime" />
        </xs:sequence>
       </xs:complexType>
      </xs:element>
</xs:schema>
```

მონაცემების მიღება ხდება პორტების საშალებით. პორტისას მიეთთება შეტყობინების ტიპი (Message Type). შეტყობინება კი თავის

მხრივ მიეთითება ის სქემა რომლის შესაბამის XML-ის მიღებაც ხდება, Port>Message>Schema. ამ შემთხვევაში მითითებულია Clients.xsd. შეტობინების მიღების შემდეგ უნდა მოხდეს XML -ის ფორმირება, რა ფორმატითაც საჭიროა მისი გადაცემა ვებ-სერვისზე. ამისათვის გამოიყენება Transform კომპონენტი. მის შემავალ შეტყობინების ტიპში მიეთითება Clients.xsd და Clients4Service.xsd.

ამ სქემებს შორის ურთიერთკავშირი აიგება 8.6 ნახაზზე მოცემული ინსტრუმენტის საშუალებით. მიღებული XML-ის გადაცემა ხდება ClientsService ვებ-სერვისისთვის. ამისათვის ორკესტრაციაში ემატება გაგზავნის შეტყობინება, რომელიც გადასცემს ახალ პორტს კლიენტის მონაცემებს. ვებ-სერვისის გამძახებისთვის საჭიროა პროექტს დაემატოს კავშირი ვებ-სერვისთან. ამისათვის გამოიყენება AddGeneratedItems ხელსაწყო. 8.7 ნახაზზე მოცემულია ამ ინსტრუმენტის გამოძახება:




ნახ.8.6. XML ტრანსფორმაციის ინსტრუმენტი


ნახ.8.7. AddGeneratedItems ხელსაწყო

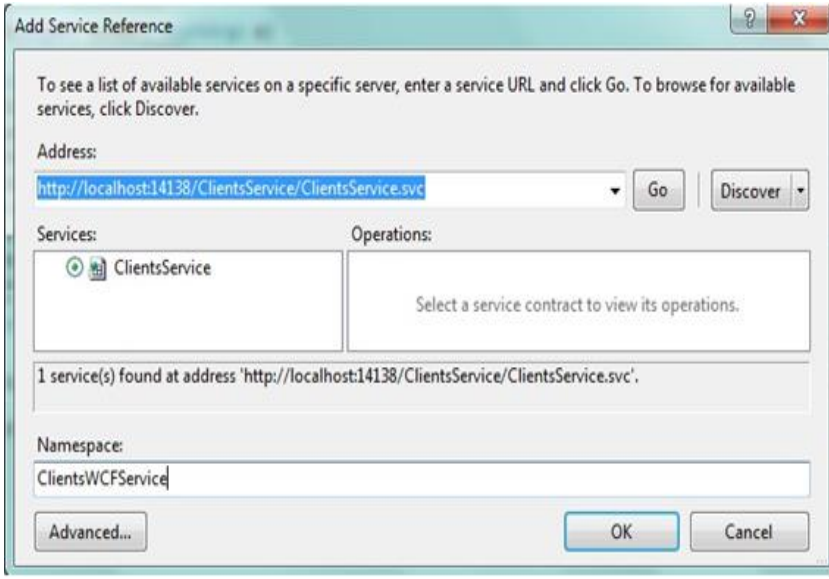
პორტის მიმღებ შესასვლელზე გადაეცემა AddClientMessage შეტყობინება. ამ ოპერაციის შესრულების შემდეგ გამოიძახება ვებ-სერვისი, რომელიც უზრუნველყოფს ბაზაში/სისტემაში ახალი კლიენტის დამატებას. ოპერაციის დასრულების შემდეგ ვებ-სერვისი აბრუნებს შეტყობინებას:

BankBizTalk. ClientsService_tempuri_org.OpenClientResponse.

ამ შეტყობინების მიღების შემდეგ ორკესტრაციაში ჩადებული ლოგიკა ასრულებს მუშაობას.

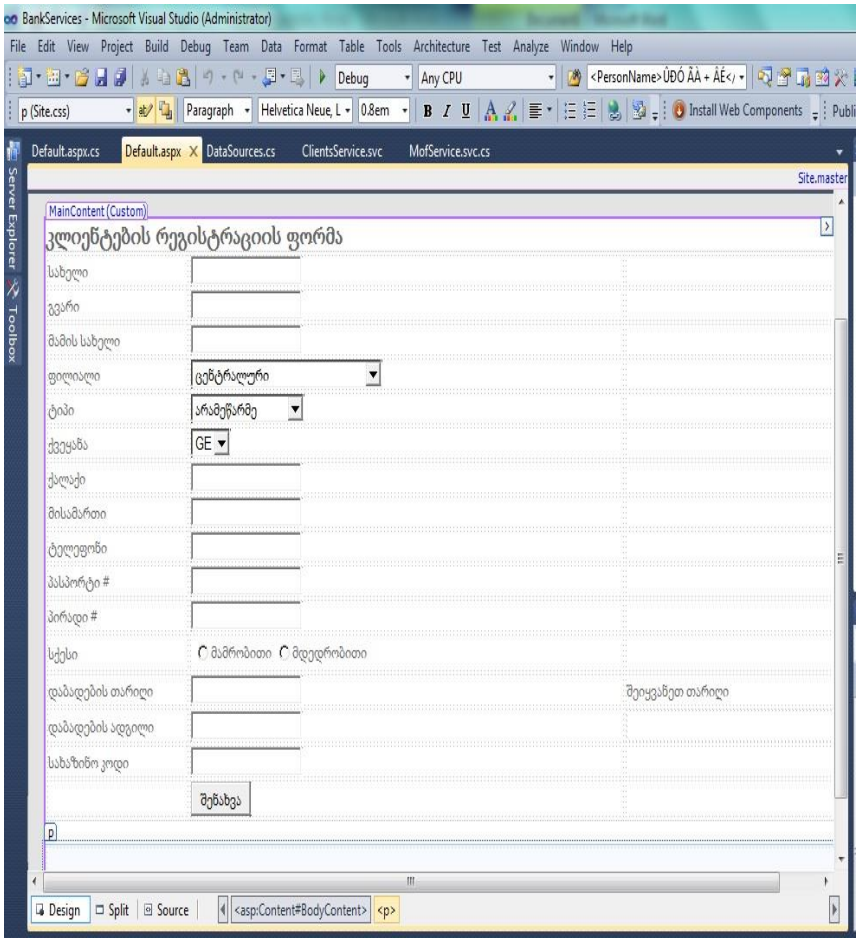
## 8.4. Web-სისტემის მომხმარებელთა ინტერფეისების აგება სერვისების რეალიზაციით

ვებ ინტერფეისის ფორმა შექმნილია ASP.NET აპლიკაციაში. იგი შედგება ვებგვერდებისა და კოდის ფაილებისგან [34,36]. ეს აპლიკაცია მონაცემების ბაზაში წვდომისათვის გამოიყენებს ვებ-სერვისებს. ვებ სერვისები დამატებული Add Service Reference დიალოგური ფანჯრის საშალებით. ამ ფანჯრის სურათი მოცემულია 8.8 ნახაზზე.




ნახ.8.8. Add Service Reference დიალოგური ფანჯარა.

ამ ფანჯარაში გაიწერება ვებ-სერვისი URL. იქმნება სპეციალური სახელების სივრცე (Namespace) და ხდება ვებ-სერვისთან დასაკავშრებელი კლასების გენერაცია. ახალი კლიენტების რეგისტრაციის ფორმა, რომელიც გამოიყენება მათი რეგისტრაციისათვის სისტემაში, მონაცემების ბაზაში შესანახად, მოცემულია 8.9 ნახაზზე.




**ნახ.8.9. მომხმარებლის ინტერფერისი ASP.NET გარემოში**

ვებ-გვერდი შედგება HTML და კოდის ფაილებისგან. კოდში გამოყენებულია C# ენა. მომხმარებლის მონაცემების შეტანის შემდეგ „შენახვა“ ღილაკზე დაჭერით მოხდება მონაცემების გადაგზავნა სერვერზე და პროცედურის გამომახება AddClient_Click. მოცემულია ღილაკის Click მეთოდის 8.10 ლისტინგი:

```
//--- ლისტინგი_8.10 ---------------------------
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 ვებ-სერვისი ClientsWCFService და მისი მეთოდი OpenClient. ამ მეთოდს გადაეცემა ახალი მომხმარებლის მონაცემები, ვებ-სერვისი კი უზრუნველყოფს ამ მონაცემების ასახვას ბაზაში.

## 8.5. დასკვნა

კორპორაციულ და კორპორაციათაშორის საინფორმაციო სისტემებში აუცილებელია თანამედროვე ინტეგრაციული საშუალებების გამოყენება, რაც უზრუნველყოფს სერვის–ორიენტირებული არქიტეტურის რეალიზაციას, რომელიც დაფუძნებულია ორგანიზაციის ინტეგრაციის არხზე. მსგავსი მიდგომა თავის მხრივ განაპირობებს ელექტრონული ბიზნეს–პროცესების შესრულების სისწრაფის ზრდას, ეფექტიანობას, საიმედოობას და უსაფრთხოებას;

თანამედროვე ინფორმაციული ტექნოლოგიების, კერძოდ CASE-ინსტრუმენტების გამოყენებით კორპორაციული მენეჯმენტისა და ინტერკორპორაციული ვებ-აპლიკაციების აგების პროცესში, მნიშვნელოვნად უმჯობესდება პროგრამული უზრუნველყოფის ხარისხი და საგრძნობლად მცირდება დაპროექტების, მისი იმპლემენტაციისა და რეინჟინერინგის პერიოდები. დიდი საინფორმაციო სისტემების მონაცემთა ბაზების სტრუქტურების დაპროექტებისა და აგების პროცესების ავტომატიზება, აგრეთვე მისი შემდგომი რესტრუქტურიზაციის პრობლემების მოქნილად გადაწყვეტის საშუალებას იძლევა.

## ლიტერატურა:

1. ბოტჰე კ. (გერმ.), სურგულაძე გ., დოლიძე თ., შონია ო., სურგულაძე გიორგი. სახელმძღ., ISBN 99940-14-11-0. თანამედროვე პროგრამული პლატფორმები და ენები: (Windows, Unix, Linux, C/C++, Java, XML). სტუ, თბილისი, 2003. -252 გვ.

2. გაბედავა ო., პოჩოვიანი ს. სერვერული ტექნოლოგიები. სახელმძღვანელო. სტუ, თბ.,2011. 255 გვ.

3. Storage_area_network. http://en.wikipedia.org/wiki/Storage_-area_network

4. Data_centers. http://en.wikipedia.org/wiki/Data_centers

5. Ziegler M. Virtualisierung. Desktop zentral. VMWares Konzept der Virtual Desktop Infrastructure; iX 8/2006, S. 100

6. Sittel F. Storage-Area-Network an der Humboldt-Universität. cms-journal, Nr.22; 05.11.2001

7. Sittel F. Fileservice für die Institute auf Basis eines Storage Area Network. cms-journal, Nr.24; 17.04.2003

8. Рассел Ч., Кроуфорд Ш., Джеренд Дж. Microsoft Windows 2003 – Справочник Администратора. Пер. с Англ. М. Издательство «СП ЭКОМ»б 2004. 1392 с. ISBN 5-9570-0016-7

9. Клее Б. Многоуровненное хранение в целях экономии затрат. Журнал сетевых решении LAN. Ноябрь 2010

10. Хомутский Ю. Шесть шагов для снижения расходов на эксплуатацию ЦОД. Журнал сетевых решении LAN. Ноябрь 2010

11. http://itc.ua/articles/balansirovka_nagruzki__tehnologiya_sozdaniya_puleneprobivaemyh_sajtov_1842

12. სურგულაძე გ., დოლიძე თ., გულუა დ. ვირტუალური სისტემების მოდელირება კორპორაციულ ქსელებში. სტუ, თბილისი, 2009

13. ადამია ვ., არაბული ნ., ცირამუა ზ. კომპიუტერული ქსელები (ნაწ.1). სტუ, თბ., 2009. -208 გვ.

14. სურგულაძე გ., პეტრიაშვილი ლ. მონაცემთა საცავის აგების ტექნოლოგია ინტერნეტული ბიზნესის სისტემებისთვის. სტუ.

„ტექნიკური უნივერსიტეტი“. 2005. -205 გვ., www.gtu.ge/katedrebi-/kat94/pdf/WareHouse-19.pdf

15. Surguladze G., Petriashvili L., Shonia O. The Visual, Objectoriented Modelling, Design, Analysis and Implementation of Distributed Business Processes using .NET Technologies and Petri Networks. Bulletin of Georg.Acad.of Science. v.172,N2, 2005

16. სურგულაძე გ., გულუა დ., ურუშაძე ბ., კაშიბაძე მ. (2013). ორგანიზაციის საინფორმაციო ინფრასტრუქტურის ავტომატიზების თანამედროვე მეთოდები. სტუ-ს შრ.კრებ. „მას“ N1(14). - გვ. 109-114

17. სურგულაძე გ., გულუა დ., მაისურაძე გ. (2011). კორპორატიული ქსელების „ღრუბლოვანი" სერვისების მონაცემთა საცავების დაპროექტების მეთოდები. სტუ-ს შრ.კრებ. „მას“ N2(11). - გვ. 89-92

18. http://en.wikipedia.org/wiki/World_Wide_Web

19. http://office.microsoft.com/en-us/windows-sharepoint-services-help/working-with-sharepoint-lists-part-1-HA001119988.aspx

20. სურგულაძე გ. დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. სტუ. „IT-კონსალტინგის ცენტრი“. თბ., 2016. -272 გვ.

21. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: WCF ტექნოლოგია (ნაწ.3). სტუ, თბ., 2015.

22. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: Workflow ტექნოლოგია (ნაწ.2). სტუ, თბ., 2015.

23. სურგულაძე გ. კორპორაციული მენეჯმენტის სისტემების Windows დეველოპმენტი: WPF ტექნოლოგია (ნაწ.1). სტუ, თბ., 2014.

24. Мак-Дональд М. WPF: Windows Presentation Fundation в .NET 3.5 с примерами на C# 2008 для профессионаюв. 2-е издание: Пер. с англ. - М. : ООО "И.Д. Вильяме". 2008

25. 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.

26. Petzold Ch. Applications=Code+Markup. A Guide to the MicroSoft Windows Presentation Foundation. St-Petersburg. 2008

27. სურგულაძე გ., ოხანაშვილი მ., სურგულაძე გ. მარკეტინგის ბიზნეს_პროცესების უნიფიცირებული და იმიტაციური მოდელირება. მონოგრ., სტუ. თბ., 2009

28. სურგულაძე გ., ოხანაშვილი მ., კაშიბაძე მ., ნეფარიძე მ. თანამედროვე საინფორმაციო ტექნოლოგიები მარკეტინგული პროცესების და წარმოების მენეჯმენტში. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(17), თბილისი, 2014, გვ. 64-71.

29. Anurag S. WCF: From a Beginner's perspective & a Tutorial. V, 4 Apr 2013. http://www.codeproject.com/Articles/566691/WCF-From-a-Beginners-perspective-a-Tutorial

30. Jensen K., Kristensen M.L., Wells L. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. University of Aarhus. Denmark. 2007

31. CPN Tools. www.daimi.au.dk/CPNTools/

32. სურგულაძე გ., გულუა დ. განაწილებული სისტემების ობიექტ-ორიენტირებული მოდელირება უნიფიცირებული პეტრის ქსელებით. მონოგრაფია. სტუ, „ტექნიკური უნივერსიტეტი“. თბილისი, 2005

33. სურგულაძე გ., ფხაკაძე ც., კეკენეძე ა. ორგანიზაციული მართვის ბიზნესპროცესების მოდელირება და დაპროექტება. მონოგრაფია. ISBN 978-9941-0-8259-7. სტუ. „IT-კონსალტინგის ცენტრი“. თბ., 2016. -204 გვ.

34. სურგულაძე გ., ბულია ი. კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება. მონოგრ., სტუ. თბ., 2012. -324 გვ.

35. სურგულაძე გ. ვიზუალური დაპროგრამება C#_2010 ენის ბაზაზე. სტუ, თბ., 2011

**დანართი_1**

## მონაცემთა ბაზების კლასტერული არქიტექტურის შექმნა კორპორაციული ქსელის ვირტუალურ სერვერულ სივრცეში

ამოცანის შესასრულებლად აუცილებელი ფიზიკური და პროგრამული რესურსების ნუსხა მოცემულია 2.1 და 2.2 ცხრილებში, ხოლო თავად ამოცანა შემდეგ პუნქტებს მოიცავს:

- ფიზიკური ჰოსტებისა და მონაცემთა საცავების გამართვა მათ შორის სრული კავშირების დამყარებით (ორივე ჰოსტის ორივე საცავთან დაკავშირება):
  - უტილიტა IBM DS Storage Manager;
- ორი ახალი ვირტუალური მანქანის შექმნა წინასწარ შექმნილი შაბლონიდან (**VM Template**) ოპერაციული სისტემით **Windows 2008 R2 Enterprise** და მათი გამართვა:
  - 2 ვირტუალური ქსელური ინტერფეისის შექმნა, რომელთაგან პირველი განკუთვნილია კლასტერისა და ქსელის ურთიერთობისთვის და ღიაა, ხოლო მეორე კლასტერის კვანძთაშორის კომუნიკაციას ემსახურება და ჩაკეტილია (არ გააჩნია ქსელში გასასვლელი ჭიშკარი ანუ გეითვეი). კლასტერის კვანძთა შიგა ქსელს **Heartbeat- (გულისცემის) ქსელსაც** უწოდებენ;

▪ ***<u>შენიშვნა:</u>*** *მეორე ვირტუალური მანქანის ქსელური კონფიგურაცია ადაპტერების წაშლას და ხელახლა დამატებას მოითხოვს*

  - **Windows Failover Cluster**-ის ამუშავების წინაპირობას კლასტერის კვანძების წინასწარ შექმნილ **დომენურ ინფრა-სტრუქტურაში (Active Directory)** ინტეგრირებაა. კლასტერული სერვისის გაშვება და მომსახურება კვანძებზე დომენური (გლობალური) სააღრიცხვო ჩანაწერებით აუთენტიკაციას მოითხოვს;

- ვირტუალური მანქანის ადგილსამყოფელი SAN-საცავია. აუცილებელია ვირტუალური მანქანების **მიგრაციის** ფუნქციის ტესტირება ჰოსტებსა და საცავებს შორის.

- 2 ახალი ვირტუალური დისკის შექმნა ერთ-ერთ ვირტუალურ მანქანაზე:
  - ერთ-ერთ ვირტუალურ მანქანაში დისკების ონლაინ-რეჟიმში გადაყვანა, ინიციალიზაცია, ტომების შექმნა და ჭდეების მინიჭება;
  - კლასტერის კვორუმ-დისკი;
  - კლასტერული მონაცემთა ბაზის საცავი.

- **Windows Failover Cluster**-ის აქტივაცია ვირტუალურ მანქანებზე სერვერის ფუნქციათა (**Features**) სიიდან;
- **.NET Framework** -ის ფუნქციის გააქტიურება ორივე კვანძზე;
- **Windows SQL Server Cluster**-ის ინსტალაცია კლასტერის ერთ-ერთ კვანძზე. ინსტალაცია სრულდება ყველა არჩეულ კვანძზე.
  - **Cluster Validation Test**

- **Windows SQL Server Cluster**-ის ახალი კვანძის დამატება კლასტერის მეორე კვანძზე;
- **Cluster Shared Disks;**

დანართი_2

## Web-ის უსაფრთხოების და სერვისის მომხმარებელთა ინტერფეისების რეალიზაცია

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 [34,35]:

```
<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>
```


ნახ. დ2-1

გადაეცა წარმოებას 12.03.2017 წ. ხელმოწერილია დასაბეჭდად 21.03.2017 ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 10,5. ტირაჟი 100 ეგზ.

სტუ-ს „IT კონსალტინგის ცენტრი“ (თბილისი, მ.კოსტავას 77)