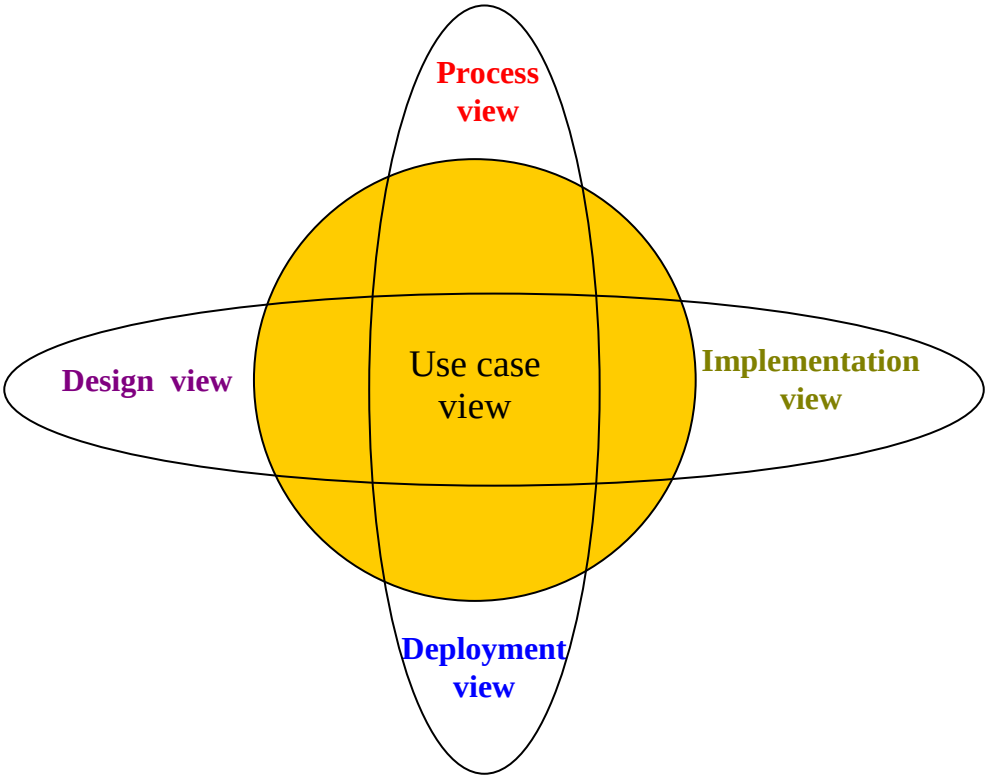
თ. სუხიაშვილი

მოდელირების უნიფიცირებული ენა (UML). პრაქტიკული ობიექტ-ორიენტირებული ანალიზი და დაპროექტება

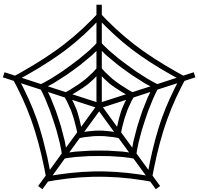


“ტექნიკური უნივერსიტეტი”

საქართველოს ტექნიკური უნივერსიტეტი

თეიმურაზ სუხიაშვილი

მოდელირების უნიფიცირებული ენა (UML). პრაქტიკული ობიექტ-ორიენტირებული ანალიზი და დაპროექტება



დამტკიცებულია სტუ – ს
სამეცნიერო-ტექნიკური საბჭოს მიერ

თბილისი 2018

უაკ 681.3.06.

ა ნ ო ტ ა ც ი ა

მოცემულია მართვის ავტომატიზებული სისტემების აგების მეთოდიკა ობიექტ – ორიენტირებული მიდგომით თანამედროვე ინფორმაციული ტექნოლოგიების გამოყენებით, მოდელირების უნიფიცირებული ენის (**UML**) ბაზაზე.

ახსნილია საპრობლემო სფეროს სტრუქტურული, ქცევითი და არქიტექტურული მოდელირების საკითხები. მოყვანილია მართვის ავტომატიზებული სისტემების დამუშავების ეტაპები რაციონალური უნიფიცირებული პროცესის გამოყენებით.

განკუთვნილია ინფორმატიკის სპეციალობის სტუდენტების, მაგისტრანტებისა და სპეციალისტებისათვის.

პროფ. თ. სუხიაშვილის რედაქციით

გამომცემლობა « ტექნიკური უნივერსიტეტი », 2018

ს ა რ ჩ ე ვ ი

თავი 1

მოდელირების მნიშვნელობა ავტომატიზებული სისტემის აგებისას

თავი 2

სტრუქტურული მოდელირების საფუძვლები

თავი 3

ქცევითი მოდელირების საფუძვლები

თავი 1
მოდელირების მნიშვნელობა ავტომატიზებული სისტემის აგებისას
1.1. მოდელირების პრინციპები

ავტომატიზებული სისტემის აგება გულისხმობს არსებული მართვის სისტემის შესწავლას და მისი ფუნქციონირების ადეკვატური მოდელის შექმნას. ყველა შემთხვევაში რთული სისტემების მოდელირება აუცილებელია, წინააღმდეგ შემთხვევაში ჩვენ ვერ შევძლებთ აღვითქვათ იგი როგორც ერთიანი მთლიანი, ჩვენ ვაგებთ მოდელს რათა უკეთ გავიგოთ დასამუშავებელი სისტემა და საშუალებას გვაძლევს გადავწყვიტოთ ამოცანები, რომლებიც განაპირობებენ ავტომატიზებული სისტემის ფორმირებას. ეს ამოცანებია:

- სისტემის ვიზუალირება მიმდინარე და ჩვენთვის სასურველ მდგომარეობაში;
- განისაზღვროს სისტემის სტრუქტურა და ქცევა;
- მივიღოთ შაბლონი, რომელიც საშუალებას მოგვცემს შემდეგ მოვახდინოთ სისტემის კონსტრუირება;
- აგებული მოდელების გამოყენებით, მოვახდინოთ მიღებული გადაწყვეტილების დოკუმენტირება.

მაგრამ საპრობლემო სფეროს მოდელირებისას უნდა გავითვალისწინოთ მოდელის შერჩევისა და აგების ოთხი ძირითადი პრინციპი:

1. ***მოდელის შერჩევა ახდენს განმსაზღვრელ გავლენას პრობლემის გადაწყვეტის მიდგომაზე და იმაზე თუ როგორ გამოისახება ეს გადაწყვეტილება.***
   თანამედროვე მართვის თეორიაში ამჟამად არსებობს უამრავი მოდელები, რომლებიც აღწერენ საპრობლემო სფეროს სხვადასხვა ინსტრუმენტალური საშუალებებით. ნებისმიერი შეიძლება

მეტნაკლებათ მოერგოს დასამუშავებელ სისტემას, მაგრამ ამავე დროს თითოეული მათგანი გავლენას ახდენს ავტომატიზებული სისტემის საბოლაო სახეზე და გასათვალისწინებელია თუ რამდენად დააკმაყოფილებს იგი მომხმარებელს.

2. ***ყოველი მოდელი შესაძლებელია რეალიზებულ იქნას აბსტრაქციის სხვადასხვა დონით.***
სისტემის დამუშავებისას გარკვეულ შემთხვვაში შეიძლება დაგვჭირდეს საპრობლემო სფეროს განზოგადებული წარმოდგენა, ხოლო სხვა შემთხვევაში მისი დეტალური აღწერა. ნებისმიერ შემთხვევაში უკეთესი მოდელი იქნება ის, რომელიც უზრუნველყოფს დეტალიზაციის საჭირო დონის ამორჩევას იმისგან დამოკიდებულებით, თუ ვინ და რა დანიშნულებით უყურებს სისტემას. ყველა შემთხვევაში მომხმარებლისათვის უფრო მეტ ინტერეს წარმოადგენს, თუ რისი გაკეთება შეუძლია მოცემული სისტემით, ხოლო დამმუშავებლისათვის, თუ როგორ გააკეთებს ამას.

3. ***საუკეთესო მოდელები ის არის, რომლებიც უფრო ახლოა რეალობასთან.***
ცხადია, ყველაზე უკეთესი შემთხვევაა, როდესაც აგებული მოდელი შეესაბამება რეალობას. მაგრამ იმ შემთხვევაში როდესაც ასეთი შესაბამობა დარღვეულია, უნდა შეგვეძლოს იმის დადგენა, რაში გამოიხატება განსხვავება და რა გამომდინარეობს აქედან. რამდენადაც მოდელი ყოველთვის ამარტივებს რეალობას, ამოცანა იმაშია, რომ ამ გამარტივებამ არ გამოიწვიოს რაიმე არსებითი დანაკარგები.

4. ***არ შეიძლება შევიზღუდოთ მხოლოდ ერთი მოდელის შექმნით. ყველაზე კარგი მიდგომა ნებისმიერი არატრივიალური სისტემის დამუშავებისას – გამოვიყენოთ რამოდენიმე მოდელების ერთობლიობა, თითქმის დამოუკიდებელნი ერთმანეთისაგან.***
რთული ორგანიზაციული სისტემების მოდელირებისას, მის ცალკეულ შემადგენელ ელემენტებს შორის რთული სემანტიკური

კავშირების არსებობის გამო, მათი სრულყოფილი ასახვისათვის მიზანშეწონილია რამოდენიმე მოდელის შექმნა, რომელიც აღწერს საპრობლემო სფეროს სხვადასხვა ხედვით და აბსტრაქციით. მოდელები შეიძლება შეიქმნან და შესწავლილ იქნან ერთმანეთისაგან დამოუკიდებლად, მაგრამ ამასთან ერთად ისინი რჩებიან ურთიერთ დაკავშირებულნი.

ასეთი მიდგომა სავსებით მისაღებია ობიექტ-ორიენტირებული სისტემებისათვის, რომლებშიც სისტემის არქიტექტურის გაგებისათვის გამოიყენება რამოდენიმე ურთიერთ შევსებადი სახეობები: პრეცედენტების თვალთახედვით (სისტემისადმი მოთხოვნების დასადგენად), პროექტირების თვალთახედვით (საპრობლემო სფეროს ლექსიკონის დასადგენად), პროცესების თვალთახედვით (სისტემაში მართვის ნაკადებისა და პროცესების განაწილების დასადგენად), რეალიზაციის თვალთახედვით, რომელიც საშუალებას მოგვცემს განვიხილოთ სისტემის ფიზიკური რეალიზება, და განლაგების თვალთახედვით, რომელიც გვეხმარება ყურადღება გავამახვილოთ სისტემური დაპროექტების საკითხებზე.

სისტემის ბუნებიდან გამომდინარე ზოგიერთი მოდელები მოცემული ობიექტისათვის შესაძლებელია უფრო მნიშვნელოვანი იყვნენ სხვებზე. ასე მაგალითად, ისეთი სისტემების დამუშავებისას, რომლებიც განკუთვნილნი არიან დიდი მოცულობის მონაცემების დამუშავებისათვის, უფრო მნიშვნელოვანია მოდელები, რომლებიც მიეკუთვნებიან სტატიკური პროექტირების თვალსაზრისს. მომხმარებლის იტერაქტიულ მუშაობაზე განსაზღვრულ სისტემებში, წინა რიგში დგება წარმოდგენა სტატიკური და დინამიკური პრეცენდენტების თვალსაზრისით. რეალური დროის სისტემებში უფრო არსებითია წარმოდგენა დინამიური პროცესების თვალსაზრისით.

1.2. ობიექტური მოდელირება და სისტემის არქიტექტურა

თანამედროვე სისტემების დამუშავებისას არსებობს რამოდენიმე მიდგომა მოდელირებისადმი. მათგან უმთავრესია ალგორითმული და ობიექტ - ორიენტირებული.

ალგორითმული მეთოდი წარმოადგენს ტრადიციულ მიდგომას პროგრამული უზრუნველყოფის შესაქმნელად. ძირითად სამშენებლო ბლოკებს წარმოადგენენ პროცედურები და ფუნქციები, ხოლო ყურადღება პირველ რიგში ენიჭება მართვის გადაცემის საკითხებს და დიდი ალგორითმების დეკომპოზიციას მცირეზე. ცუდი ამაში არაფერია თუ არ ჩავთვლით იმას, რომ სისტემები არც თუ ისე მარტივად ადაპტირდებიან მოთხოვნილების ცვლილებისას ან განზომილების გაზრდისას, რაც მთავარია მათი გამოყენებაც საკმაოდ რთულდება.

უფრო თანამედროვე მიდგომა პროგრამული უზრუნველყოფის დამუშავებისას არის ობიექტ- ორიენტირებული. აქ ძირითად სამშენებლო ბლოკის სახით გამოდის ობიექტი ან კლასი. ყველაზე ზოგადი აზრით ობიექტი არსებაა საპრობლემო სფეროს ლექსიკონიდან, ხოლო კლასი წარმოადგენს ერთი ტიპის მქონე ობიექტთა სიმრავლის აღწერას. ყოველი ობიექტი ხასიათდება მდგომარეობით (მას უკავშირებენ გარკვეულ მონაცემებს) და ქცევით (მასზე შეიძლება რაიმეს გაკეთება ან მას თვითონ შეუძლია გააკეთოს რაიმე სხვა ობიექტზე).

ობიექტ - ორიენტირებული მიდგომა პროგრამული უზრუნველყოფის დასამუშავებლად ამჟამად ყველაზე მიღებულია უბრალოდ იმიტომ, რომ მან დაანახა თავისი სარგებლიანობა ნებისმიერი განზომილებისა და სირთულის სისტემების აგებისას სხვადასხვა სფეროებში.

გარდა ამისა თანამედროვე დაპროგრამების ენების უმრავლესობა, ინსტრუმენტალური საშუალებები და ოპერაციული სისტემები წარმოადგენენ ამა თუ იმ ზომით ობიექტ-ორიენტირებულებს.

თუ ჩვენ მივიღებთ ობიექტ- ორიენტირებულ მიდგომას, მაშინ უნდა განისაზღვროს როგორი სტრუქტურა უნდა გააჩნდეს ობიექტ-ორიენტირებული არქიტექტურიდან.

არქიტექტურა ეს არსებითი გადაწყვეტილებების ერთობლიობაა, რომელიც ეხება:

- პროგრამული სისტემის ორგანიზაციას;
- სტრუქტურული ელემენტების ამორჩევას, რომელიც შეადგენს სისტემას და მათ ინტერფეისებს;
- ამ ელემენტების ქცევას, რომელიც სპეციფიცირებულია სხვა ელემენტებთან კოოპერაციაში;
- ამ სტრუქტურული და ქცევითი ელემენტებისაგან სულ უფრო მსხვილი ქვესისტემების შექმნას;
- არქიტექტურულ სტილს, რომელიც წარმართავს და განსაზღვრავს სისტემის მთელ ორგანიზაციას: სტატიკურ და დინამიკურ ელემენტებს, მათ ინტერფეისებს, კოოპერაციებს და მათი გამოყენების საშუალებებს.

პროგრამული სისტემის არქიტექტურა მოიცავს არა მარტო მის სტრუქტურულ და ქცევით ასპექტებს, არამედ გამოყენებას, ფუნქციონირებას, წარმადობას, მოქნილობას, განმეორებითი გამოყენების შესახლებლობას, სისრულეს, ეკონომიურ და ტექნოლოგიურ შეზღუდვებს და კომპრომისებს, ასევე ესთეთიკურ საკითხებს.

ყველას ვისაც კავშირი აქვს დასამუშავებელ სისტემასთან – მომხმარებლები, ანალიტიკოსები, დამმუშავებლები, მატესტირებლები, ტექნიკოსები და პროექტის მენეჯერები გააჩნიათ საკუთარი

ინტერესები და ყოველი მათგანი უყურებს შესაქმნელ სისტემას თავისებურად მისი სასიცოცხლო ციკლის სხვადასხვა მომენტში. ეს მიგვანიშნებს იმ ფაქტზე, რომ სისტემის სრულყოფილი აღწერისა და შესწავლისათვის აუცილებელია საპრობლემო სფეროს განხილვა სხვადასხვა თვალსაზრისით.

1.3. სისტემის სხვადასხვა წარმოდგენა

პროგრამული სისტემის არქიტექტურა ყველაზე ოპტიმალურად შესაძლებელია აღწერილი იყოს ხუთი ურთიერთ დაკავშირებული სახით ან წარმოდგენით, თითოეული მათგანი წარმოადგენს სისტემის ორგანიზაციის და სტრუქტურის ერთერთ შესაძლო პროექციას და ყურადღებას ამახვილებს მისი ფუნქციონირების განსაზღვრულ ასპექტზე. ყოველი მათგანი გულისხმობს სტრუქტურულ და ქცევით მოდელირებას (სტატიკური და დინამიკური არსებების მოდელირება).

შეხედულება პრეცენდენტების თვალსაზრისით მოიცავს პრეცენდენტებს, რომლებიც აღწერენ სისტემის ქცევას, მომხმარებლის დაკვირვებით. ეს სახე არ წარმოგვიდგენს პროგრამის ჭეშმარიტ ორგანიზაციას, არამედ იმ მამოძრავებელ ძალებს, რომლებისგანაც დამოკიდებულია სისტემური არქიტექტურის ფორმირება.

შეხედულება დაპროექტების თვალსაზრისით მოიცავს კლასებს, ინტერფეისებსა და კოოპერაციებს, რომლებიც ახდენენ ამოცანის და მისი გადაწყვეტის ლექსიკონის ფორმირებას. ეს შეხედულება პირველ რიგში აყენებს ფუნქციონალურ მოთხოვნებს, რომელიც წაეყენება სისტემას, ან იმ მომსახურებას, რომელიც მან უნდა წარუდგინოს მომხმარებლებს.

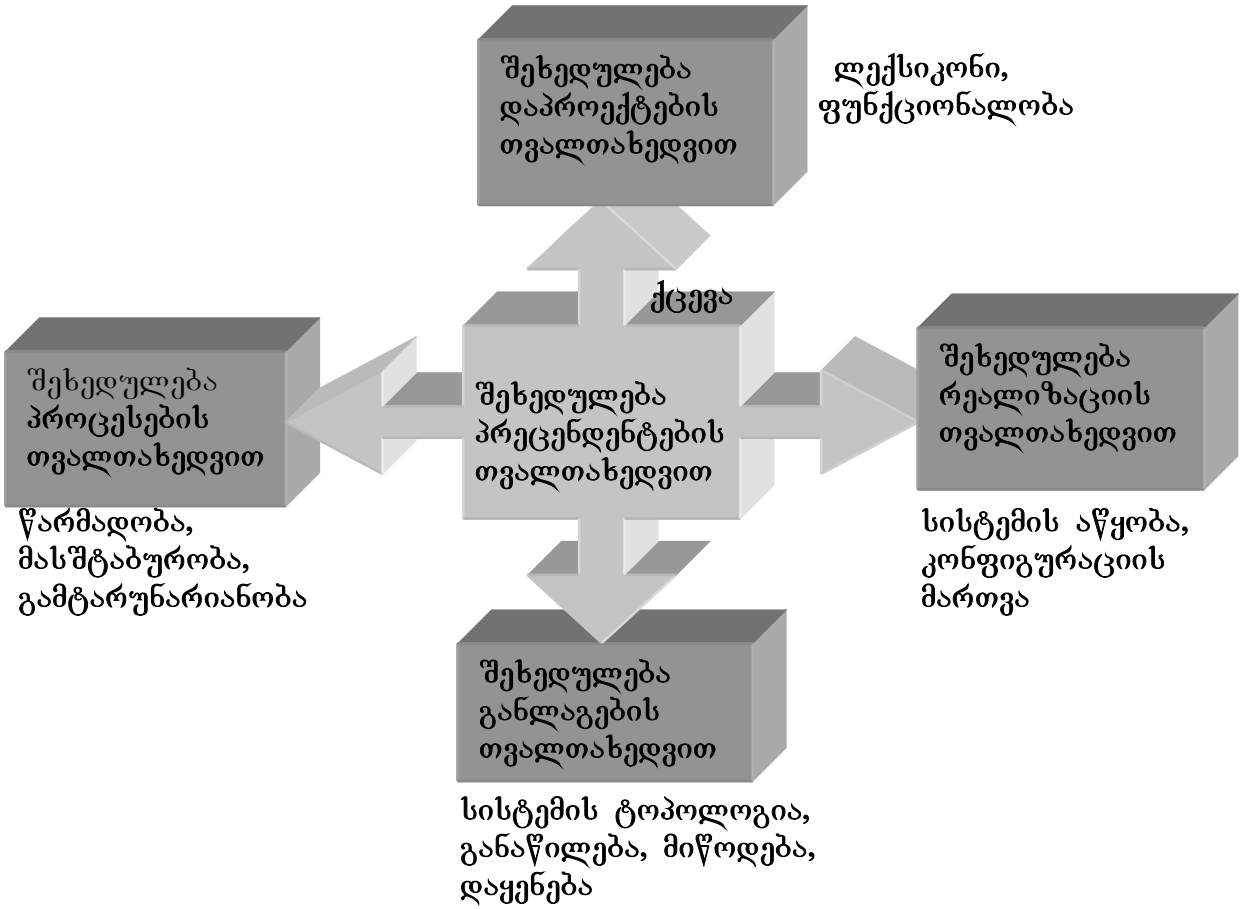
შეხედულება პროცესების თვალსაზრისით მოიცავს პროცესებსა და ძაფებს, რომლებიც ახდენენ პარალელიზმისა და სინქრონიზაციის ფორმირებას სისტემაში. ეს სახე უმთავრესად აღწერს სისტემის წარმადობას, მასშტაბურობას და გამტარუნარიანობას.

შეხედულება რეალიზაციის თვალსაზრისით მოიცავს კომპონენტებსა და ფაილებს, რომლებიც გამოიყენებიან საბოლაო პროგრამული პროდუქტის აწყობისა და გამოშვებისათვის. ეს სახე პირველ რიგში განკუთვნილია სისტემის კონფიგურაციის ვერსიების მართვისათვის, რომლებიც შედგენილია დამოუკიდებელი კომპონენტებისა და ფაილებისაგან.

შეხედულება განლაგების თვალსაზრისით მოიცავს კვანძებს, რომლებიც ახდენენ სისტემის აპარატული საშუალებების ტოპოლოგიის ფორმირებას, რომელზედაც იგი არის შესრულებული. პირველ რიგში იგი დაკავშირებულია სისტემის ფიზიკური შემადგენელი ნაწილების განაწილებასთან, შეკვეთასა და დაყენებასთან.

თითოეული ამ ჩამონათვალიდან, შეიძლება ჩაითვალოს სავსებით დამოუკიდებელ ასპექტად, ისე რომ, პირები რომლებსაც კავშირი აქვთ სისტემის დამუშავებასთან, შეუძლიათ ყურადღება გაამახვილონ არქიტექტურის მხოლოდ იმ ასპექტების შესწავლაზე, რომლებიც უშუალოდ მათ ეხებიან. მაგრამ არ უნდა დავივიწყოთ, რომ ეს სახეები ურთიერთქმედებენ ერთმანეთში.

ამრიგად, **სახე ან წარმოდგენა (View)** - ეს მოდელია, რომელიც განიხილება გარკვეული თვალსაზრისით: მათში გამოსახულია ერთი არსებები და გამოტოვებულია სხვები, რომლებსაც მოცემული თვალსაზრისით ინტერესი არ გააჩნიათ.



ნახ.1.

იმისათვის, რომ გამოვსახოთ სისტემა რომელიმე თვალთახედვით გამოიყენება დიაგრამები. ამჟამად განსაზღვრულია ცხრა ტიპის დიაგრამა, რომელთა კომბინირება შესაძლებელია საჭირო წარმოდგენის მისაღებათ. სისტემის სტატიკური ნაწილების განხილვისას გამოიყენება კლასების, ობიექტების, კომპონენტების და განლაგების დიაგრამები, ხოლო სისტემის დინამიურ ნაწილებთან სამუშაოდ გამოიყენება პრეცენდენტების, მიმდევრობის, კოოპერაციის, მდგომარეობის და მოღვაწეობის დიაგრამები.

1.4. მოდელირების უნიფიცირებული ენა(UML)

სისტემის გაგებისათვის აუცილებელია მოდელირება. ამასთან მარტო ერთი მოდელი არასდროს არ არის საკმარისი. ყოველი სისტემის გაგებისათვის, საჭიროა დიდი რაოდენობის ურთიერთ

დაკავშირებული მოდელების დამუშავება. პროგრამულ სისტემებთან მიმართებაში ეს ნიშნავს, რომ საჭიროა ენა, რომლის მეშვეობითაც შესაძლებელი იქნება სხვადასხვა კუთხით ავღწეროთ სისტემის არქიტექტურა, მისი დამუშავების ციკლის განმავლობაში.

ამჟამად სტანდარტულ ინსტრუმენტს პროგრამული უზრუნველყოფის „მონახაზის" შესაქმნელად წარმოადგენს მოდელირების უნიფიცირებული ენა - Unified Modeling Language (**UML**). მისი მეშვეობით შესაძლებელია პროგრამული სისტემების არტეფაქტების ვიზუალირება, სპეციფიცირება, კონსტრუირება და დოკუმენტირება.

UML შეიძლება გამოვიყენოთ ნებისმიერი სისტემის მოდელირებისათვის დაწყებული ინფორმაციული სისტემებიდან დამთავრებული განზოგადებული **WEB-** წინადადებით. იგი ძალიან გამოხატული ენაა. რომელიც საშუალებას იძლევა განვიხილოთ სისტემა ყველა მხრიდან. მიუხედავად გამოხატვის ფართო შესაძლებლობისა ეს ენა ადვილი გასაგები და გამოსაყენებელია.

ენა შესდგება ანბანისა და წესებისაგან, რომელთა მეშვეობით შესაძლებელია მასში შემავალი სიტყვების კომბინირება და მივიღოთ აზრობრივი კონსტრუქციები.

UML არ არის ვიზუალური დაპროგრამების ენა, მაგრამ მისი მეშვეობით შექმნილი მოდელები, შეიძლება უშუალოდ გადაყვანილ იქნას დაპროგრამების სხვადასხვა ენებზე. სხვა სიტყვებით რომ ვთქვათ **UML** მოდელი შეიძლება გამოვსახოთ ისეთ ენებზე როგორიც არის **JAVA, C++, Visual Basik** და რელაციურ მონაცემთა ბაზის ცხრილებზეც კი ან ობიქტ – ორიენტირებული მონაცემთა ბაზის მდგრად ობიექტებზე. ის მცნებები, რომლებიც უფრო ხელსაყრელია გადავცეთ გრაფიკულად, ასეთივე სახით წარმოიდგინებიან **UML**-ში, ხოლო ის მცნებები, რომლებიც უფრო

ხელსაყრელია ავღწეროთ ტექსტური სახით, გამოისახებიან დაპროგრამების ენების მეშვეობით.

მოდელის ასეთი ასახვა დაპროგრამების ენებზე საშუალებას გვაძლევს განვახორციელოთ პირდაპირი პროექტირება- **UML** მოდელიდან კონკრეტული ენის კოდების გენერაცია. შეიძლება გადაწყდეს უკუამოცანაც - მოვახდინოთ მოდელის რეკონსტრუირება არსებული რეალიზაციიდან.

დაპროგრამების ენებზე პირდაპირ ასახვასთან ერთად **UML**-ის გამომხატველობისა და ერთმნიშვნელობის მეშვეობით საშუალება გვეძლევა შევასრულოთ მოდელი, მოვახდინოთ სისტემის ქცევის იმიტირება და ვაკონტროლოთ არსებული სისტემები.

UML-ის ანბანი მოიცავს სამ სამშენებლო ბლოკს:

- არსები;
- მიმართებები:
- დიაგრამები.

არსები – ეს აბსტრაქციებია, რომლებიც წარმოადგენენ მოდელის ძირითად ელემენტებს. მიმართებები აკავშირებენ სხვადასხვა არსებებს; დიაგრამები ახდენენ ჩვენთვის საინტერესო არსთა ერთობლიობის დაჯგუფებას.

არსები. საპრობლემო სფეროს ასახვისათვის UML-ში ძირითადათ გამოიყენება სტრუქტურული და ქცევის არსები, რომლებიც წარმოადგენენ ენის ძირითად ობიექტ - ორიენტირებულ ბლოკებს. მათი მეშვეობით შესაძლებელია შევქმნათ კორექტული მოდელები.

სტრუქტურული არსები ეს **UML** ენაზე მოდელში არსებითი სახელებია. როგორც წესი, ისინი წარმოადგენენ მოდელის სტატიკურ ნაწილებს, რომლებიც შეესაბამებიან სისტემის კონცეპტუალურ და ფიზიკურ ელემენტებს.

არსებობს სტრუქტურული არსების შვიდი ნაირსახეობა:

- **კლასი** (**CLASS**) – ეს ობიექტთა ერთობლიობის აღწერაა საერთო ატრიბუტებით, დამოკიდებულებებით და სემანტიკით.
- **ინტერფეისი** (**Interface**) – ეს ოპერაციების ერთობლიობაა, რომლებიც განაპირობებენ მომსახურების ერთობლიობას, რომელსაც წარმოადგენს კლასი ან კომპონენტი. ინტერფეისს შეუძლია წარმოადგინოს კლასის ან კომპონენტის ყოფაქცევა მთლიანად ან ნაწილობრივ. იგი განსაზღვრავს ოპერაციის მხოლოდ სპეციფიკაციებს(სიგნატურას), მაგრამ არასდროს მათ რეალიზაციებს.
- **კოოპერაცია** (Collaboracion) განსაზღვრავს ურთიერთქმედებას, იგი წარმოადგენს როლებისა და სხვა ელემენტების ერთობლიობას, რომლებიც მუშაობენ რა ერთად ქმნიან გარკვეულ კოოპერაციულ ეფექტს. იგი არ დაიყვანება ელემენტთა უბრალო ჯამამდე. კოოპერაციას აქვს როგორც სტრუქტურული, ასევე ქცევითი ასპექტი, ერთი და იგივე კლასი შეიძლება მონაწილეობდეს რამოდენიმე კოოპერაციაში. მაშასადამე, ისინი წარმოადგენენ ქცევის სახეობების რეალიზაციას.
- **პრეცენდენტი** (**Use case**) - ეს სისტემის მიერ შესასრულებელი მოქმედებათა თანმიმდევრობის აღწერაა, რომელიც ქმნის ხილულ შედეგს და რომელიც მნიშვნელოვანია რომელიმე განსაზღვრული აქტიორისათვის(**Aktor**), პრეცენდენტი გამოიყენება ქცევის არსებების სტრუქტურირებისათვის. პრეცენდენტები რეალიზდებიან კოოპერაციის მეშვეობით. სამი შემდეგი არსება აქტიური კლასები, კომპონენტები და კვანძები კლასების მსგავსია, ისინი აღწერენ ობიექტების ერთობლიობას საერთო ატრიბუტებით, ოპერაციებით, დამოკიდებულებებით და სემანტიკით.
- **აქტიური კლასი** (**Active class**) უწოდებენ კლასს, რომლის ობიექტები ჩართულნი არიან ერთ ან რამოდენიმე პროცესში,

ამიტომ შეიძლება მოახდინონ მმართველი ზემოქმედების ინიციირება.

- **კომპონენტი** (**Component**) სისტემის ფიზიკური ნაწილია, რომელიც შეესაბამება ინტერფეისების გარკვეულ ნაკრებს და უზრუნველყოფს მის რეალიზაციას(მაგ. ფაილები, ბიბლიოთეკები, გვერდები, ცხრილები). კომპონენტი წარმოადგენს ლოგიკური ელემენტების ფიზიკურ წარმოდგენას, როგორიც არის კლასები, ინტერფეისები და კოოპერაციები.
- **კვანძი** (**Node**) - ეს არის რეალური (ფიზიკური) სისტემის ელემენტი, რომელიც არსებობს პროგრამული კომპლექსის ფუნქციონირებისას და წარმოადგენს გამოთვლით რესურს, გარკვეული მეხსიერებით და დამუშავების შესაძლებლობით. კომპონენტების ერთობლიობა შეიძლება განლაგდეს კვანძში, ასევე გადაადგილდეს ერთი კვანძიდან მეორეში.

ქცევითი არსებები წარმოადგენენ **UML** მოდელის დინამიურ შემადგენელს. ეს ენის ზმნებია, ისინი აღწერენ მოდელის ქცევას დროში და სივრცეში. ქცევითი არსებებია:

- **ურთიერთქმედება** (**Interaction**) – ეს არის ქცევა, რომლის არსი მდგომარეობს შეტყობინებების გაცვლაში ობიექტებს შორის კონკრეტული კონტექსის ფარგლებში გარკვეული მიზნების მისაღწევად. ურთიერთქმედების საშუალებით შესაძლებელია ავღწეროთ როგორც ცალკეული ოპერაცია, ისე ობიექტთა ერთობლიობის ქცევა.
- **ავტომატი** (**State machine**) – ეს ქცევის ალგორითმია, რომელიც განსაზღვრავს მდგომარეობათა თანმიმდევრობას, რომლებზედაც ობიექტები ან ურთიერთქმედება გადადის მთელი სასიცოცხლო ციკლის განმავლობაში სხვადასხვა მოვლენების საპასუხოდ. ავტომატის მეშვეობით შესაძლებელია ავღწეროთ ცალკეული კლასის ან კლასის კოოპერაციის ქცევა. ავტომატთან

დაკავშირებულია რიგი სხვა ელემენტებისა: მდგომარეობები, გადასვლები (ერთი მდგომარეობიდან მეორეში), მოვლენები (არსებები, რომლებიც გადასვლების ინიციალიზაციას ახდენენ) მოქმედებათა სახეები (რეაქცია გადასვლაზე). სემანტიკურათ ქცევითი არსების ეს ორივე ელემენტი ხშირათ არიან დაკავშირებული სხვადასხვა სტრუქტურულ ელემენტთან, პირველ რიგში – კლასებთან, კოოპერაციებთან და ობიექტებთან.

UML-ის არსებს მიაკუთვნებენ აგრეთვე დაჯგუფების და ანოტაციურ არსებს. დაჯგუფების არსებას უწოდებენ პაკეტს. პაკეტი წარმოადგენს ელემენტების ჯგუფებში ორგანიზაციის უნივერსალურ მექანიზმს. პაკეტში შესაძლებელია მოვათავსოთ სტრუქტურული, ქცევითი და სხვა დაჯგუფების არსები.

ანოტაციური არსები ეს **UML** მოდელის განმარტებითი ნაწილია. მას მიეკუთვნება კომენტარიები დამატებითი აღწერისათვის, მოდელის ნებისმიერი ელემენტის განმარტების ან შენიშვნისათვის. ძირითადათ გამოიყენება მხოლოდ ერთი ანოტაციური ელემენტი – შენიშვნა(Note).

მიმართებები. UML ენაში განსაზღვრულია ოთხი ტიპის მიმართება:

- დამოკიდებულება;
- ასოციაცია;
- განზოგადება;
- რეალიზაცია.

ეს მიმართებები წარმოადგენენ ძირითად დამაკავშირებელ სამშენებლო ბლოკებს **UML**-ში და გამოიყენება კორექტული მოდელების შესაქმნელად.

დამოკიდებულება (Dependency) – ეს სემანტიკური კავშირია ორ არსებას შორის. რომლის დროსაც ერთერთის ცვლილებას, დამოუკიდებულის, შეიძლება გავლენა მოახდინოს მეორეს, დამოკიდებულის, სემანტიკაზე.

ასოციაცია (Association) - სტრუქტურული მიმართებაა, რომლებიც აღწერენ კავშირების ერთობლიობას. კავშირი – ეს ობიექტებს შორის შეერთებაა. ასოციაციის ნაირსახეობას წარმოადგენს აგრეგირება – ასე ეძახიან სტრუქტურულ მიმართებას მთელსა და მის ნაწილს შორის.

განზოგადება (Generalization) – ეს არის მიმართება როდესაც ობიექტი(შვილობილი) შეიძლება წარმოვადგინოთ განზოგადებული ობიექტის(მშობელის) მაგიერ. მაშასადამე, შვილობილი მემკვიდრეობით ღებულობს თავისი მშობლის სტრუქტურასა და ქცევას.

რეალიზაცია (Realization) – ეს სემანტიკური მიმართებაა კლასიფიკატორებს შორის, რომლის დროსაც კლასიფიკატორი განსაზღვრავს კონტრაქტს, ხოლო მეორე გარანტიას იძლევა მის შესრულებაზე. რეალიზაციიც მიმასრთება გვხვდებაორ შემთხვევაში: პირველ რიგში ინტერფეისსა და მის მარეალიზებელ კლასებს ან კომპონენტებს შორის, მეორეს მხრივ, პრეცენდენტებსა და მათ მარეალიზებელ კოოპერაციებს შორის.

გარდა ამ ოთხი ტიპის მიმართებისა არსებობს აგრეთვე ვარიაციები მაგალითად დაზუსტება(**Refinement**), ტრასსირება(**Trace**), ჩართვა და გაფართოება (დამოკიდებულებისათვის).

დიაგრამები. დიაგრამა **UML**-ში ეს არის გრაფიკული წამოდგენა ელემენტების ერთობლიობისა, რომელიც გამოისახება შეკრული გრაფით, რომლის წვეროებია არსებები და წიბოები კი მიმართებები.

UML-ში გამოყოფენ ცხრა ტიპის დიაგრამებს, რომლებიც გამოიყენებიან სისტემის როგორც სტატიკური, ისე დინამიური ასპექტების ვიზუალირებისათვის:

კლასების დიაგრამაზე გამოისახება კლასები, ობიექტები, ინტერფეისები, კოოპერაციები და მათ შორის მიმართებები. ობიექტ-

ორიენტირებული სისტემების მოდელირებისას კლასების დიაგრამები გამოიყენება ყველაზე ხშირად. პროექტირების თვალსაზრისით კლასების დიაგრამა შეესაბამება სისტემის სტატიკურ სახეს.

ობიექტების დიაგრამაზე წარმოიდგინება ობიექტები და მიმართებები მათ შორის. ობიექტების დიაგრამაც სისტემის სტატიკურ სახეს ასახავს.

პრეცენდენტების დიაგრამაზე წარმოდგენილია პრეცენდენტები, აქტიორები და მიმართებები მათ შორის. პრეცენდენტების გამოყენების თვალსაზრისით მოცემული დიაგრამაც ასახავს სისტემის სტატიკურ სახეს.

მიმდევრობისა და კოოპერაციის დიაგრამები წარმოადგენენ ურთიერთქმედების დიაგრამების კერძო სახეს. მათზე წარმოიდგინება ობიექტებს შორის კავშირები და შეტყობინებები, რომლებითაც ხდება ინფორმაციის მიმოცვლა მოცემულ ობიექტებს შორის. ურთიერთქმედების დიაგრამები ასახავენ სისტემის დინამიკურ სახეს. ამასთან მიმდევრობის დიაგრამები გამოხატავენ შეტყობინებათა დროის მიხედვით მოწესრიგებას, ხოლო კოოპერაციის დიაგრამები – სტრუქტურულ ორგანიზაციას შეტყობინებათა მიმოცვლაში მონაწილე ობიექტებს შორის.

მდგომარეობათა დიაგრამაზე (Statechart diagrams) წარმოიდგინება ავტომატი, რომელიც მოიცავს მდგომარეობებს და გადასვლებს მათ შორის. ისინი ყურადღებას ამახვილებენ ობიექტის ქცევაზე, რომელიც დამოკიდებულია მოვლენათა თანმიმდევრობაზე. მდგომარეობათა დიაგრამები ასახავენ სისტემის დინამიკურ სახეს.

მოღვაწეობის დიაგრამაზე გამოისახება სისტემის შიგნით მართვის ნაკადის გადასვლები ერთი მოქმედებიდან მეორეზე. მოღვაწეობის დიაგრამები ასახავენ სისტემის დინამიკურ სახეს.

კომპონენტების დიაგრამაზე წარმოიდგინება კომპონენტების ორგანიზაცია და მათ შორის არსებული დამოკიდებულებები.

რეალიზაციის თვალსაზრისით კომპონენტების დიაგრამა მიეკუთვნება სისტემის სტატიკურ სახეს.

განლაგების დიაგრამაზე წარმოიდგინება სისტემის დასამუშავებელი კვანძების კონფიგურაცია და მათში განლაგებული კომპონენტები.

სისტემის სტატიკური ასპექტების ვიზუალირებისათვის გამოიყენება ოთხი სტრუქტურული დიაგრამა. სტრუქტურული დიაგრამების დასახელებები შეესაბამებიან ძირითადი არსებების სახელებს, რომლებიც გამოიყენებიან სისტემის მოდელირებისას. მსგავსად იმისა თუ როგორ და რა სახით განლაგდება შენობაში სახლის სტატიკური ასპექტები (კედლები, კარებები, ფანჯრები და ა.შ), ასევე ავტომატიზებული სისტემის სტატიკური ასპექტები ასახავენ კლასების, ინტერფეისების, კოოპერაციების, კომპონენტების, კვანძების და სხვა არსებების არსებობასა და განლაგებას.

სისტემის დინამიური ასპექტების ვიზუალირებისათვის გამოიყენება ქცევის ხუთი დიაგრამა. შეიძლება ჩაითვალოს, რომ სისტემის დინამიური ასპექტები წარმოადგენენ მის ცვალებად ნაწილებს. მაგალითად, საცხოვრებელი სახლის დინამიური ასპექტებია ჰაერისა და ხალხის ნაკადების გადაადგილება ოთახებში. ავტომატიზებული სისტემის დინამიური ასპექტები მოიცავენ მის ისეთ ელემენტებს, როგორიც არის შეტყობინებათა ნაკადი დროში და კომპონენტების ფიზიკური გადაადგილება ქსელში.

ქცევის დიაგრამები პირობითად იყოფიან ხუთ ტიპად სისტემის დინამიკის მოდელირების ძირითადი საშუალებების შესაბამისად:

- პრეცედენტების დიაგრამა აღწერს სისტემის ქცევის ორგანიზებას;
- მიმდევრობის დიაგრამა ყურადღებას ამახვილებს შეტყობინებათა დროის მიხედვით მოწესრიგებაზე;

- კოოპერაციის დიაგრამები ფოკუსირებულნი არიან შეტყობინებათა გამგზავნებისა და მიმღებთა სტრუქტურულ ორგანიზაციაზე;
- მდგომარეობათა დიაგრამები აღწერენ სისტემის მდგომარეობათა ცვლას მოვლენის პასუხად;
- მოღვაწეობის დიაგრამები ახდენენ მართვის გადაცემის დემონსტრირებას ერთი მოღვაწეობიდან მეორეზე.

ამრიგად, სისტემის ავტომატიზაცია მოითხოვს მის მოდელირებას. მოდელი კი ეს აბსტრაქციაა, რომელიც იქმნება იმისათვის, რომ უკეთ გავიგოთ სისტემა. სისტემა აღიწერება მოდელების ნაკრებით, რომლებიც შესაძლებლობის მიხედვით განიხილავენ მას სხვადასხვა თვალთახედვით. ჩვენ საშუალება გვეძლევა მოვახდინოთ სისტემების და ქვესისტემების მოდელირება როგორც ერთიანი მთლიანის და ამით ორგანულად გადავწყვიტოთ მასშტაბირების პრობლემა.

თავი 2
სტრუქტურული მოდელირების საფუძვლები
2.1. კლასები

კლასები - ეს ყველაზე მნიშვნელოვანი სამშენებლო ბლოკია ნებისმიერი ობიექტ-ორიენტირებული სისტემისათვის. მათი მეშვეობით აღიწერება პროგრამული, აპარატული ან სუფთა კონცეპტუალური არსებები.

სისტემის მოდელირება გულისხმობს არსებების იდენტიფიცირებას, რომლებიც მნიშვნელოვანია ამა თუ იმ თვალსაზრისით. მაგალითად, თუ თქვენ აშენებთ სახლს, მაშინ თქვენთვის როგორც სახლის მფლობელისათვის მნიშვნელოვანი იქნება კედლები, კარებები, ფანჯრები, კარადები და განათება. თითოეული ამ არსებათაგანი განსხვავდება სხვებისაგან და ხასიათდება თვისებების საკუთარი ნაკრებით. კედლებს აქვთ სიმაღლე, სიგანე, სიგრძე და ა.შ. კარებსაც აქვს სიმაღლე და სიგანე, მაგრამ გარდა ამისა გააჩნია მექანიზმი, რომლითაც ისინი იხსნებიან ერთ მხარეს. ფანჯარა წააგავს კარებს იმით, რომ ჩაშენებულია კედელში, დანარჩენში ცხადია ისინი განსხვავდებიან. ფანჯრებს აპროექტებენ ისე, რომ მათგან შესაძლებელი იყოს ხედვა და არა სიარული.

კედლები, კარებები და ფანჯრები იშვიათად არსებობენ თავისთვის, ამიტომ უნდა გადავწყვიტოთ თუ როგორ დაუკავშიროთ ისინი ერთმანეთს. თუ რომელ არსებს ავირჩევთ და როგორ კავშირებს დავამყარებთ მათ შორის, ცხადია დამოკიდებულია იმაზე თუ როგორ ვაპირებთ ოთახის გამოყენებას.

UML-ში ყველა ეს არსები მოდელირდება როგორც კლასები. კლასი ეს არსების აბსტრაქციაა, იგი წარმოადგენს არა ინდივიდუალურ ობიექტს, არამედ არსთა ერთობლიობას. აქედან გამომდინარე ჩვენ შეგვიძლია ჩავთვალოთ, რომ “კედელი” არის

ობიექტების კლასი, საერთო თვისებებით, როგორიც არის სიმაღლე, სიგანე, სიგრძე, სისქე, მზიდი კედელია თუ არა და ა. შ. ამასთან ცალკეული კედლები განიხილება როგორც კლასი "კედელი"-ს ცალკეული ეგზემპლიარები.

მაშასადამე, ***კლასს*** **(CLASS)** უწოდებენ ობიექტთა ერთობლიობის აღწერას საერთო ატრიბუტებით, ოპერაციებით, მიმართებებით და სემანტიკით.

ყოველ კლასს უნდა გააჩნდეს **სახელი**, რომელიც განასხვავებს მას სხვა კლასებისაგან. კლასის სახელი - ეს ტექსტური სტრიქონია. სახელის გარდა კლასის დახასიათებისათვის გამოიყენება ატრიბუტები, ოპერაციები და მოვალეობები.

ატრიბუტი - კლასის დასახელებული თვისებაა, რომელიც შეიცავს მნიშვნელობათა სიმრავლეს, რომელსაც ღებულობენ ამ თვისების ეგზემპლიარები. კლასს შეიძლება ქონდეს ატრიბუტების ნებისმიერი რაოდენობა ან საერთოდ არ ქონდეს. ატრიბუტი წარმოადგენს სამოდელო არსების გარკვეულ თვისებას, რომელიც საერთოა მოცემული კლასის ყველა ობიექტისათვის. მაგ. კლიენტის მოდელირებისას შეიძლება მიუთითოთ გვარი, სახელი, მისამართი, ტელეფონი და დაბადების თარიღი. მაშასადამე, ატრიბუტი წარმოადგენს ობიექტის მონაცემების ან მისი მდგომარეობის აბსტრაქციას. დროის ყოველ მომენტში ობიექტის ნებისმიერ ატრიბუტს გააჩნია სავსებით გარკვეული მნიშვნელობა. ატრიბუტის სახელი, ისევე როგორც კლასის, შეიძლება იყოს ნებისმიერი ტექსტური სტრიქონი.

ოპერაციას უწოდებენ მომსახურების რეალიზაციას, რომელიც შეიძლება მოვითხოვოთ მოცემული კლასის ნებისმიერი ობიექტისაგან ქცევაზე ზემოქმედებისათვის. სხვა სიტყვებით რომ ვთქვათ, ოპერაცია – ეს აბსტრაქციაა იმისა, თუ რისი გაკეთება შეიძლება ობიექტზე. კლასს შეიძლება ქონდეს ოპერაციების

ნებისმიერი რაოდენობა ან საერთოდ არ ქონდეს. ოპერაციის სახელი, ისევე როგორც კლასის, შეიძლება იყოს ნებისმიერი ტექსტური სტრიქონი. ოპერაციის აღწერისას შეიძლება მიუთითოთ პარამეტრები, მათი ტიპი და მნიშვნელობები, ფუნქციების შემთხვევაში – დასაბრუნებელი მნიშვნელობის ტიპი.

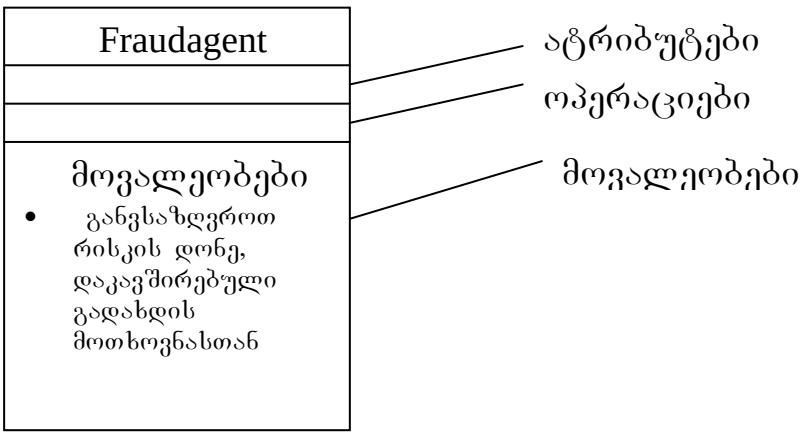
კლასის **მოვალეობები** - ეს თავისებურად კონტრაქტია, რომელსაც იგი უნდა დაემორჩილოს. განსაზღვრავთ რა კლასს, თქვენ უთითებთ, რომ ყველა მის ობიექტებს აქვთ ერთგვაროვანი მდგომარეობა და ქცევა. შესაბამისი ატრიბუტები და ოპერაციები წარმოადგენენ სწორედ იმ თვისებებს, რომლის მეშვეობითაც სრულდება კლასის მოვალეობები. მაგალითად, კლასი კედელი პასუხს აგებს ინფორმაციაზე სიმაღლის, სიგანისა და სისქის შესახებ და ა.შ.

კლასების მოდელირება უკეთესია დავიწყოთ სისტემის ლექსიკონში შემავალი არსებების მოვალეობების განსაზღვრიდან. პრინციპში კლასის მოვალეობების რაოდენობა შეიძლება იყოს ნებისმიერი, მაგრამ კარგათ სტრუქტურირებულ კლასს აქვს სულ მცირე ერთი მოვალეობა. მეორეს მხრივ მათი რაოდენობა არ უნდა იყოს ძალიან დიდი. მოდელის დაზუსტებისას კლასის მოვალეობები გარდაიქმნებიან ატრიბუტებისა და ოპერაციების ერთობლიობაში, რომლებმაც უნდა უზრუნველყონ მათი შესრულება.

კლასის გრაფიკული გამოსახვისას მიუთითებენ როგორც სახელს, ისე ატრიბუტებსა და ოპერაციებს. მოვალეობებს გამოსახავენ კლასის ქვედა ნაწილის სპეციალურ განყოფილებაში. ისინი შეიძლება მიუთითოთ აგრეთვე შენიშვნებში.

კლასების მეშვეობით ჩვეულებრივ გამოხატავენ აბსტრაქციებს, რომლებიც გამოიყოფა გადასაწყვეტი ამოცანიდან და გამოიყენება მის გადასაწყვეტად. ასეთი აბსტრაქციები წარმოადგენენ ჩვენი სისტემის ლექსიკონს ე.ი. წარმოადგენენ არსებებს, რომლებიც

მნიშვნელოვანია მომხმარებლებისა და დამმუშავებლისათვის. იმისათვის, რომ დავეხმაროთ მომხმარებელს გამოყოს ეს აბსტრაქციები უნდა მიმართოს პრეცენდენტების ანალიზს.



სისტემის ლექსიკონის მოდელირება მოიცავს შემდეგ ეტაპებს:

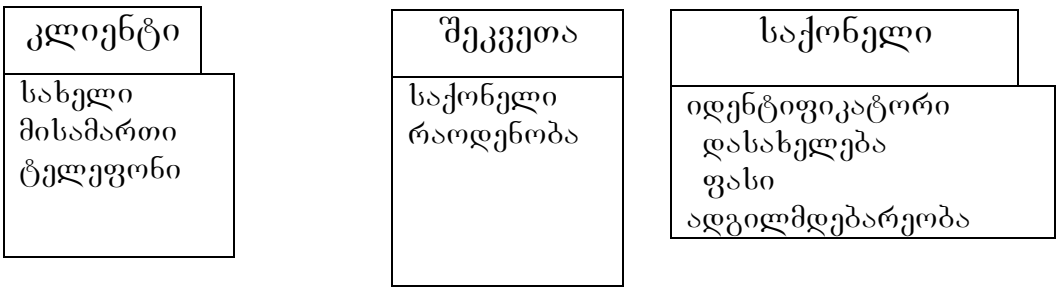
1. განვსაზღვროთ რომელ ელემენტებს იყენებენ მომხმარებლები და დამმუშავებლები ამოცანის აღწერისა და მისი გადაწყვეტისათვის.

2. სწორი აბსტრაქციების მოძებნისათვის გამოვიყენოთ პრეცენდენტების ანალიზი.

3. ყოველი აბსტრაქციისათვის დავადგინოთ მისი შესაბამისი მოვალეობების სიმრავლე.

4. დავამუშავოთ ატრიბუტები და ოპერაციები, რომლებიც აუცილებელია კლასების მიერ თავიანთი მოვალეობის შესასრულებლათ.

მაგ. საცალო ვაჭრობის სისტემისათვის გვექნება შემდეგი კლასები:



სისტემის დაპროექტებისას ჩვეულებრივ შეიძლება გვქონდეს კლასების გარკვეული სიმრავლე. ამიტომ უნდა ვიზრუნოთ მათ შორის მოვალეობების განაწილებაზე. უნდა ავერიდოთ ძალიან დიდი ან ძალიან მცირე კლასების შექმნას. ყოველი კლასი კარგათ უნდა აკეთებდეს ერთ გარკვეულ მოვალეობას.

მოვალეობების განაწილების მოდელირება სისტემაში მოიცავს შემდეგ ეტაპებს:

1. მოახდინეთ იმ კლასების იდენტიფიცირება, რომლებიც ერთობლივად აგებენ პასუხს გარკვეულ მოქმედებაზე.

2. განსაზღვრეთ ყოველი კლასის მოვალეობა.

3. შეხედეთ მიღებულ კლასებს როგორც ერთ მთლიანს, გამოყავით მათგან ისინი, რომლებსაც ძალიან ბევრი მოვალეობები აქვთ და დაყავით შედარებით მცირე კლასებათ – პირიქით, პატარა კლასები ელემენტარული მოვალეობებით, გააერთიანეთ უფრო დიდში.

4. გადაანაწილეთ მოვალეობები ისე, რომ ყოველი აბსტრაქცია გახდეს ავტონომიური.

5. განიხილეთ თუ როგორ კოოპერირებენ კლასები ერთმანეთთან და გადაანაწილეთ მოვალეობები ისეთი ანგარიშით, რომ არც ერთი კლასი კოოპერაციის ჩარჩოში არ აკეთებდეს ძალიან ბევრს ან ძალიან ცოტას.

მაგალითისათვის მოვიყვანოთ მოვალეობების განაწილება კლასებს შორის მოდელი, ხედი, კონტროლერი.

მოდელი
მოვალეობები • მოდელის მდგომარეობების მართვა

ხედი
მოვალეობები • გამოვსახოთ მოდელი ეკრანზე • ვმართოთ მზერის ველის ზომის ცვლილება და გადაადგილება

კონტროლერი
მოვალეობები • მოვახდინოთ მოდელის ცვლილებების სინქრონიზირება

კლასების მოდელირებისას უნდა დავიმახსოვროთ, რომ ყოველ კლასს უნდა შეესაბამებოდეს გარკვეული არსება და კონცეპტუალური აბსტრაქცია იმ სფეროდან, რომელთანაც საქმე აქვს მომხმარებელს ან დამმუშავებელს. კარგათ სტრუქტურირებულ კლასს აქვს შემდეგი თვისებები:

- წარმოადგენს გარკვეული მცნებების მკვეთრად გამოხატულ აბსტრაქციას საპრობლემო სფეროს ლექსიკონიდან;
- შეიცავს გარკვეულ მოვალეობების ნაკრებს და ასრულებს ყოველ მათგანს;
- უზრუნველყოფს მკვეთრ გამიჯვნას აბსტრაქციების სპეციფიკაციებსა და მათ რეალიზაციას შორის;
- გასაგები და მარტივია, მაგრამ ამავე დროს დასაშვებია გაფართოებისა და ადაპტაციის შესაძლებლობა.

კლასების გამოხატვისას დაცული უნდა იქნას შემდეგი წესები:

1. უჩვენეთ მხოლოდ ის თვისებები, რომლებიც მნიშვნელოვანია აბსტრაქციების გაგებისათვის მოცემულ კონტექსში.
2. დაყავით ატრიბუტების და ოპერაციების გრძელი ცხრილები ჯგუფებათ მათი კატეგორიების შესაბამისად;
3. უჩვენეთ ურთიერთდაკავშირებული კლასები ერთ და იმავე დიაგრამაზე.

2.2. მიმართებები

აბსტრაქციების აგებისას დავრწმუნდით, რომ კლასები იშვიათად არსებობენ ავტონომიურად. როგორც წესი ისინი სხვადასხვა საშუალებით ურთიერთქმედებენ ერთმანეთთან. ეს იმას ნიშნავს, რომ სისტემის მოდელირებისას თქვენ არა მარტო უნდა მოახდინოთ არსებების იდენტიფიცირება, არამედ უნდა აღწეროთ თუ როგორ

დამოკიდებულებაში არიან ერთმანეთთან. არსებობს მიმართებების სამი სახე, რომლებიც განსაკუთრებით მნიშვნელოვანია ობიექტ - ორიეტირებული მოდელირებისათვის:

- ***დამოკიდებულება***, რომლითაც აღიწერება კლასებს შორის არსებული გამოყენების მიმართებები.
- ***განზოგადება,*** რომლებიც აკავშირებენ განზოგადებულ კლასებს სპეციალიზირებულთან.
- ***ასოციაცია,*** რომლებიც აღწერენ ობიექტებს შორის სტრუქტურულ კავშირებს.

ყოველი მათგანი აბსტრაქციების სხვადასხვა სახით კომბინირების საშუალებას იძლევა.

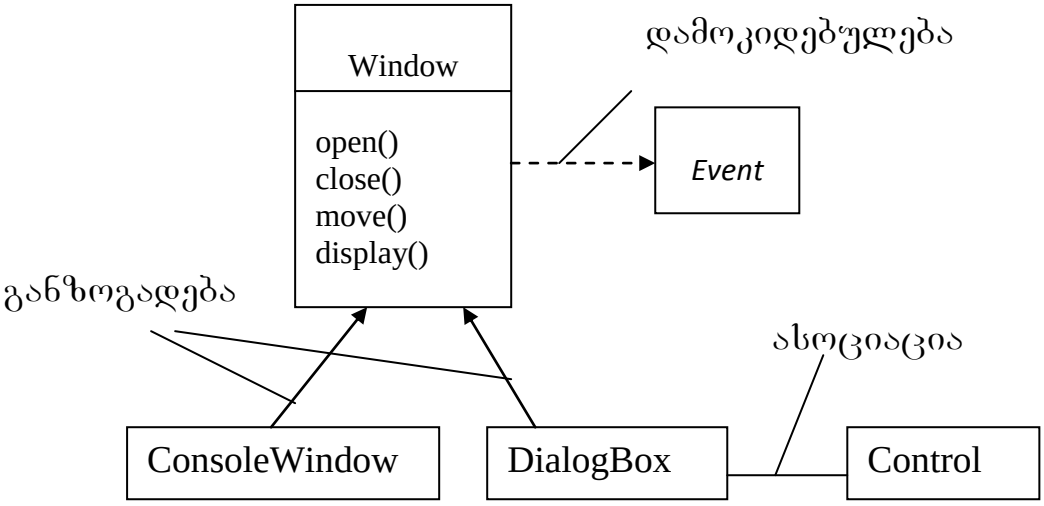
თუ ჩვენ ვაშენებთ სახლს, მაშინ ისეთი არსებები როგორიც არის კედლები, კარებები, ფანჯრები გახდებიან თქვენი ლექსიკონის ნაწილი. მათგან არც ერთი არ არსებობს თავისთავად. კედლები შეერთებულია სხვა კედლებთან, კარებები და ფანჯრები ჩაშენებულია კედლებში, რომლითაც უზრუნველყოფენ გასასვლელს და სინათლეს ადამიანებისათვის. ვაჯგუფებთ რა ამ ობიექტებს ვქმნით უფრო მაღალი დონის არსებას – ოთახს.

არსებებს შორის შეიძლება აღმოვაჩინოთ არა მარტო სტრუქტურული მიმართებები. ოთახში აუცილებლად არის ფანჯარა მაგრამ ის შეიძლება იყოს სხვადასხვა სახის. მაგ. გაუხსნელი ფანჯარა ან გასახსნელი, გარდა ამისა ზოგი იხსნება ზევით, ზოგიერთი ქვევით და შესაძლებელია გვერდზეც. მიუხედავად ჩამოთვლილი განსხვავებებისა მათთვის დამახასიათებელია საერთო თვისება: ყოველი ფანჯარა ჩაშენებულია კედელში, რათა შემოუშვას სინათლე და ჰაერი.

UML ენაში საშუალებები, რომლებითაც ელემენტები უკავშირდებიან ერთმანეთს, მოდელირდებიან მიმართებების სახით. ეს სამი მიმართება მოიცავს ელემენტების ურთიერთქმედების

საშუალებების უდიდეს ნაწილს, რაც კარგად გამოისახება ობიექტებს შორის კავშირების საშუალებებით, რომლებიც მიღებულია დაპროგრამების ობიექტ- ორიენტირებულ ენებში.

თითოეული დასახელებული მიმართებისათვის არსებობს გრაფიკული გამოსახვა, რომელიც ნაჩვენებია ქვემოთ მოყვანილ მაგალითზე. ეს ნოტაცია საშუალებას გვაძლევს მოვახდინოთ სამოდელირო მიმართებების ვიზუალირება გამოყენებელი დაპროგრამების ენისაგან დამოუკიდებლად, ისე რომ ხაზი გაესვას მათ ყველაზე მნიშვნელოვან შემადგენლებს: სახელს, დამაკავშირებელ არსებებს და თვისებებს.



დამოკიდებულება ეს არის გამოყენების მიმართება, რომლის მიხედვითაც ერთი ელემენტის სპეციფიკაციების ცვლილებას (მაგ. **ვენტ**), შეიძლება გავლენა იქონიოს მეორე ელემენტზე, რომელიც მას იყენებს (მოცემულ შემთხვევაში კლასი **Window**), ამასთან პირიქით არ არის აუცილებელი.

ყველაზე ხშირად დამოკიდებულებას იყენებენ კლასებთან მუშაობისას, იმისათვის რომ გამოვსახოთ ოპერაციათა სიგნატურაში ის ფაქტი, რომ ერთი კლასი იყენებს მეორეს არგუმენტის სახით ე.ი. ერთი კლასში მომხდარი ცვლილება გამოისახება მეორეს მუშაობაზე. მაგალითად, გათბობის მილები დამოკიდებულია გამათბობლისგან, რომელიც აცხელებს წყალს.

განზოგადებას ეს მიმართებაა საერთო არსებასა და მის კონკრეტულ გამოვლინებას შორის. მას კიდევ უწოდებენ მიმართებას "წარმოადგენს", იმის გამო რომ ერთი არსება უფრო ზოგადი წარმოადგენს მეორეს კერძო შემთხვევას. განზოგადება ნიშნავს იმას, რომ შვილობილი ობიექტი შეიძლება გამოყენებულ იქნას ყველგან, სადაც გვხვდება მშობელი კლასის ობიექტები, მაგრამ არა პირიქით. სხვა სიტყვებით რომ ვთქვათ შვილობილი შეიძლება დავაყენოთ მშობელის მაგიერ. ამასთან ის მემკვიდრეობით ღებულობს მშობელის თვისებებს, კერძოთ მის ატრიბუტებსა და ოპერაციებს. ხშირათ შვილობილს გააჩნია თავისი საკუთარი ატრიბუტები და ოპერაციები, გარდა იმისა რაც გააჩნია მშობელს. შვილობილი ღებულობს ოპერაციებს იმავე შემადგენლობით მშობლისაგან.

კლასს, რომელსაც არ გააჩნია მშობელი, მაგრამ აქვს შვილობილი უწოდებენ ბაზურს ან ფუძისეულს (ფესვს), ხოლო კლასს რომელსაც არ აქვს შვილობილი- ფოთლოვანს (leaf).

მემკვიდრეობის მიმართების მოდელირება წარმოებს შემდეგი თანმიმდევრობით:

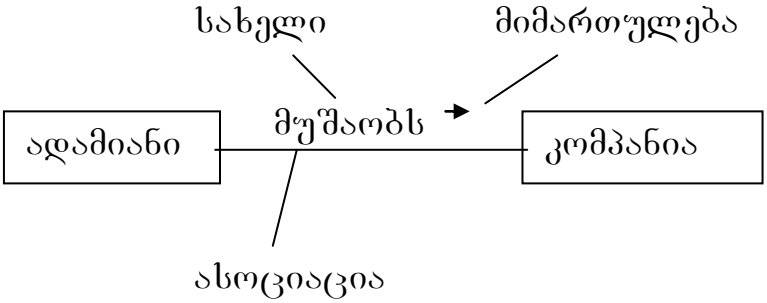
1. ვიპოვოთ ატრიბუტები, ოპერაციები და მოვლენები, რომლებიც საერთოა ორი ან მეტი კლასისათვის.
2. გამოიტანეთ ეს ელემენტები საერთო კლასში, რათა შევქმნათ ახალი კლასი. ამასთან ყურადღება მივაქციოთ იმას, რომ დონეები არ აღმოჩნდეს ძლიან ბევრი.
3. მიუთითოთ მოდელში, რომ შედარებით სპეციალიზებული კლასები მემკვიდრეობით ღებულობენ უფრო ზოგადის თვისებებს, რისთვისაც გამოვიყენოთ განზოგადების მიმართება მიმართული შვილობილიდან მის მშობელზე.

ასოციაციას უწოდებენ სტრუქტურულ მიმართებას, რომელიც გვიჩვენებს, რომ ერთი ტიპის ობიექტები გარკვეულად

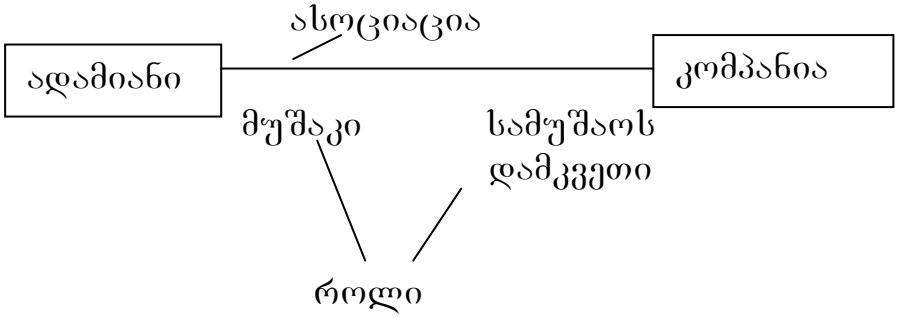
დაკავშირებული არიან მეორე ტიპის ობიექტებთან. დასაშვებია შემთხვევა, როდესაც ასოციაციის ორივე ბოლო ერთი და იმავე კლასს ეკუთვნის. ეს ნიშნავს, რომ კლასის რომელიმე ობიექტზე დასაშვებია დაუკავშირდეს სხვა ობიექტები იმავე კლასიდან. ასოციაციას, რომელიც აკავშირებს ორ კლასს უწოდებენ ბინარულს. შესაძლებელია, თუმცა იშვიათია, შეიქმნას ასოციაცია, რომელიც აკავშირებს ერთდროულად რამოდენიმე კლასს, მათ უწოდებენ ნ- არულს.

არსებობს ოთხი დამატება, რომელიც გამოიყენება ასოციაციასთან მიმართებაში.

დასახელება. ასოციაციას შეიძლება მიენიჭოს სახელი და მისი კითხვის მიმართულება.

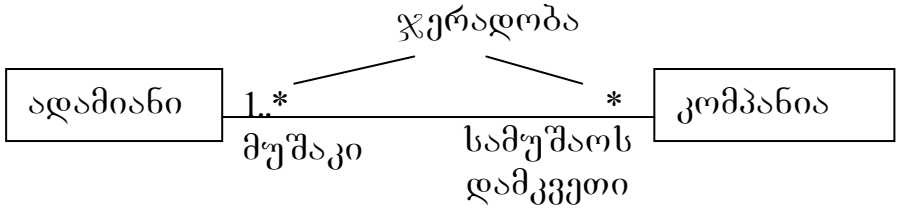


როლი. კლასი, რომელიც მონაწილეობს ასოციაციაში, აკისრია მასში გარკვეული როლი. შეგვიძლია მიუთითოთ ეს როლი.

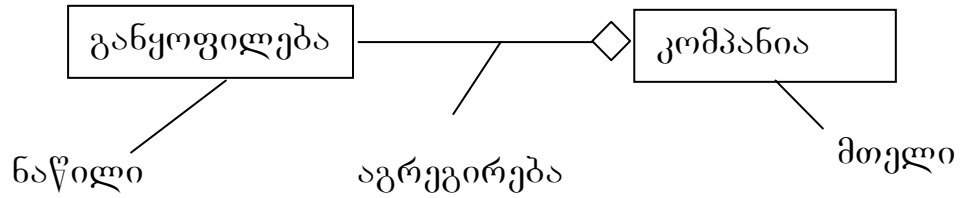


ჯერადობა. ხშირათ მოდელირებისას საჭიროა მიეთითოს, რამდენი ობიექტი შეიძლება დაუკავშირდეს ასოციაციის ერთი ეგზემპლიარით. ამ რიცხვს უწოდებენ ასოციაციის როლის

ჯერადობას. მას მიუთითებენ მნიშვნელობათა დიაპაზონის სახით. მაგ. ერთი (1), ერთი ან ნოლი (0..1), "ბევრი" (0..*), ერთი ან მეტი (1..*). შესაძლებელია მიეთითოს გარკვეული რიცხვი- (3). ქვემოთ მოყვანილი მაგალითიდან გამომდინარეობს, რომ ერთი ან მეტი მუშაკი მუშაობს ნებისმიერ (*) კომპანიაში.



აგრეგირება. უბრალო ასოციაცია ორ კლასს შორის გამოსახავს სტრუქტურულ მიმართებას თანაბარ არსებებს შორის, როდესაც ორივე კლასი იმყოფება ერთ კონცეპტუალურ დონეზე და არც ერთი არ არის უფრო მნიშვნელოვანი მეორესთან შედარებით. მაგრამ ამას გარდა გვხვდება შემთხვევები, როდესაც საჭიროა მოდელირება მიმართებისა "ნაწილი/მთელი". ერთერთ კლასს აქვს მეტი რანგი(მთელი) და შედგება რამოდენიმე უფრო მცირე რანგის ნაწილებისაგან. ასეთი ტიპის მიმართებას უწოდებენ აგრეგირებას. გრაფიკულად მას გამოხატავენ რომბით.



გარდა ზემოთ მოყვანილი მიმართებებისა, ობიექტთა დინამიური კოოპერაციების მოდელირებისას, ჩვენ გვხვდება კიდევ ორი სახის მიმართება:

- კავშირები – მიმართებები რომლის მეშვეობითაც შესაძლებელია შეტყობინებების გადაცემა;

- გადასვლები – რომლითაც აკავშირებენ მდგომარეობებს ავტომატში.

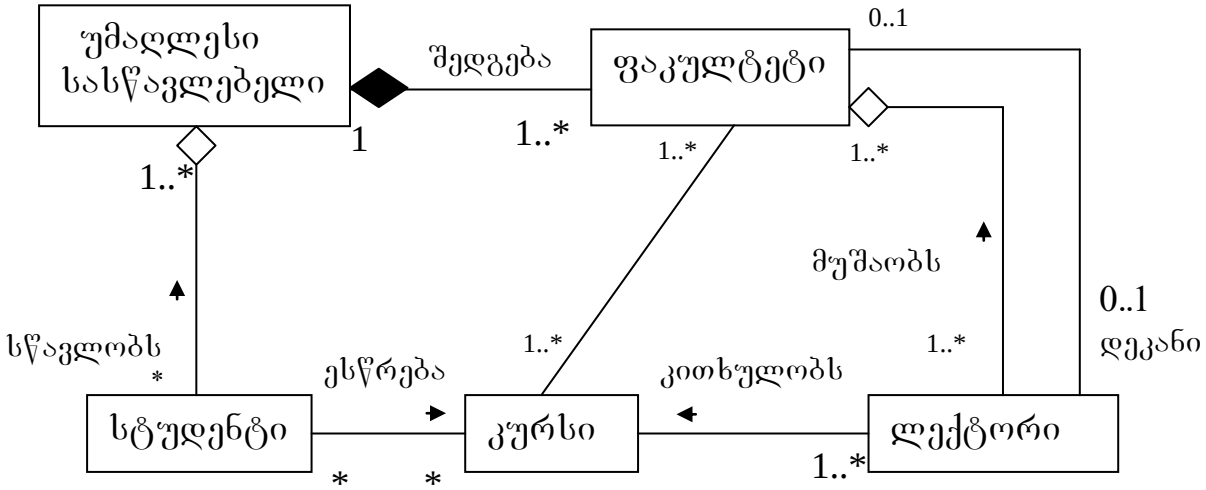
მოყვანილი მიმართებები განხილული იქნება შესაბამისი სტრუქტურული ელემენტების განხილვისას.

სტრუქტურული მიმართებების მოდელირება წარმოებს შემდეგნაირად:

1. განსაზღვრეთ ასოციაცია კლასების ყოველი წყვილისათვის. ეს იქნება შეხედულება ასოციაციაზე მონაცემების თვალსაზრისით.
2. თუ ერთი კლასის ობიექტები უნდა ურთიერთქმედებდნენ სხვა კლასის ობიექტებთან განსხვავებულად, ვიდრე ეს ოპერაციის პარამეტრებით არის განსაზღვრული, მიზანშეწონილია განისაზღვროს ამ კლასებს შორის ასოციაცია. ეს არის შეხედულება ასოციაციაზე ქცევის თვალსაზრისით.
3. ყოველი განსაზღვრული ასოციაციისათვის მიუთითეთ ჯერადობა და როლების დასახელება.
4. თუ ასოციაციის ერთერთი კლასი სტრუქტურულად ან ორგანიზაციულად წარმოადგენს მთელს ასოციაციის მეორე ბოლოში მყოფი კლასის მიმართ, მონიშნეთ ასეთი ასოციაცია როგორც აგრეგირება.

მოყვანილი მსჯელობის ვიზუალირებისათვის ქვემოთ მოყვანილია მაგალითი.

კლასებს ***სტუდენტი*** და ***კურსი*** არსებობს ასოციაცია, რომელიც გვიჩვენებს, რომ სტუდენტი ესწრება კურსებს. ყოველ სტუდენტს შეუძლია დაესწროს კურსების ნებისმიერ რაოდენობას, და ყოველ კურსზე შესაძლებელია მოდიოდეს სტუდენტების ნებისმიერი რაოდენობა.



ანლოგიურად კლასებს ***კურსი*** და ***ლექტორი*** განსაზღვრულია ასოციაცია, რომელიც გვიჩვენებს, რომ ლექტორი კითხულობს კურსს. ყოველი კურსისავის უნდა არსებობდეს ერთი ლექტორი მაინც, და ყოველ ლექტორს შეუძლია წარმართოს კურსების ნებისმიერი რაოდენობა (მათ შორის არც ერთი).

მიმართება კლასებს ***უმაღლესი სასწავლებელი*** და კლასებს ***სტუდენტი*** და ***ფაკულტეტი*** მცირედ განსხვავდებიან ერთმანეთისაგან, თუმცა ორივე წარმოადგენს აგრეგირების მიმართებას. უმაღლეს სასწავლებელში შესაძლებელია იყოს სტუდენტების ნებისმიერი რაოდენობა (ნოლის ჩათვლით), და ყოველი სტუდენტი შესაძლებელია სწავლობდეს ერთ ან რამოდენიმე უმაღლეს სასწავლებელში; უმაღლეს სასწავლებელი შესაძლებელია შედგებოდეს ერთი ან რამოდენიმე ფაკულტეტისაგან, მაგრამ ყოველი ფაკულტეტი ეკუთვნის მხოლოდ ერთ უმაღლეს სასწავლებელს. მიმართებას კლასებს შორის ***უმაღლესი სასწავლებელი*** და ***ფაკულტეტი*** უწოდებენ კომპოზიციურ აგრეგირებას.

ნახაზიდან ასევე ჩანს, რომ კლასებს ***ფაკულტეტი*** და ***ლექტორი*** დგინდება ორი ასოციაცია. ერთი მათგანი გვიჩვენებს, რომ ყოველი ლექტორი მუშაობს ერთ ან რამოდენიმე ფაკულტეტზე, და ყოველ ფაკულტეტზე უნდა იყოს სულ მცირე ერთი ლექტორი. მეორე ასოციაცია გვიჩვენებს, რომ ყოველ ფაკულტეტს მართავს მხოლოდ

ერთი ლექტორი – დეკანი. ამ მოდელის თანახმად, ლექტორი შესაძლებელია იყოს მხოლოდ ერთი ფაკულტეტის დეკანი, ამასთან ზოგიერთი ლექტორი არ არის დეკანი.

2.3. კლასების დიაგრამა

კლასების დიაგრამა ობიექტ - ორიენტირებული სისტემების მოდელირებისას გვხვდება ყველაზე ხშირად. ასეთ დიაგრამებზე უჩვენებენ კლასებს, ინტერფეისებს და მიმართებებს მათ შორის.

კლასების დიაგრამა გამოიყენება სისტემის სტატიკური სახით მოდელირებისათვის პროექტირების თვალსაზრისით. კლასების დიაგრამა წარმოადგენს საფუძველს ორი შემდეგი სახის დიაგრამისათვის – კომპონენტების და განლაგების.

ვაშენებთ რა სახლს ჩვენ ვახდენთ ძირითადი სამშენებლო ბლოკების კედლების, ფანჯრების, კარებების, იატაკის და ა.შ. ანალიზს. იმის მიუხედავად, რომ ყველა ეს არსებები ატარებენ სტრუქტურულ ხასიათს (მაგალითად, კედელი ხასიათდება სიმაღლით, სიგანით და სისქით), მათ აქვთ კიდევ ქცევითი თავისებურებანი (მაგალითად, კედლები უნდა უძლებდნენ გარკვეულ დატვირთვას, კარებები უნდა იღებოდნენ და იკეტებოდნენ). სახლის სტრუქტურული და ქცევითი ასპექტები არ შეიძლება განვიხილოთ იზოლირებულად. პირიქით, უნდა განვიხილოთ მათი ურთიერთქმედება. არქიტექტურული პროექტირების პროცესი იმაში მდგომარეობს, რომ ვაერთიანებთ რა ყველა ზემოჩამოთვლილ არსებებს, შევქმნათ ლამაზი სახლის მოდელი, რომელიც დააკმაყოფილებს ყველა ფუნქციონალურ და არაფუნქციონალურ მოთხოვნებს. ამასთან ნახაზები, რომლებიც იქმნება სახლის ვიზუალირებისათვის, წარმოადგენენ მისი ყველა

შემადგენელი ელემენტების და მათ შორის ურთიერთქმედებების გრაფიკულ გამოსახვას.

პროგრამული უზრუნველყოფის შექმნა წააგავს სახლის მშენებლობას. მიუხედავად პროგრამების აზრობრივი ბუნებისა საშუალებას გვაძლევს შევქმნათ საჭირო სამშენებლო ბლოკები. კლასების დიაგრამა გამოიყენება სწორედ ამ სამშენებლო ბლოკების და მათ შორის მიმართებების სტატიკური ასპექტების ვიზუალირებისათვის.

კლასების დიაგრამა ძირითადათ გამოიყენება შემდეგი მიზნებისათვის:

- **სისტემის ლექსიკონის მოდელირებისათვის**. იგი გულისხმობს იმას, თუ რომელი აბსტრაქცია წარმოადგენს სისტემის ნაწილს და რომელი არა. კლასების დიაგრამით ჩვენ შეგვიძლია განვსაზღვროთ ეს აბსტრაქციები და მათი მოვალეობები.
- **უბრალო კოოპერაციების მოდელირებისათვის**. კოოპერაცია ეს კლასების, ინტერფეისების და სხვა ელემენტების ერთობლიობაა, რომლებიც მუშაობენ ერთობლივად გარკვეული კოოპერაციული ქცევის უზრუნველსაყოფად, უფრო მნიშვნელოვანის ვიდრე მათი ელემენტების ჯამი. მაგ. განაწილებულ სისტემებში ტრანზაქციების სემანტიკის მოდელირებისას, ვერ გავიგებთ მიმდინარე პროცესებს, მხოლოდ ერთი კლასით, რადგან შესაბამისი სემანტიკა უზრუნველყოფილი ხდება ერთობლივად მომუშავე კლასებით. მოდელირების ეს საშულება განხილული იქნება კოოპერაციების დახასიათებისას.
- **მონაცემთა ბაზის ლოგიკური სქემის მოდელირებისათვის**. ლოგიკური სქემა შეიძლება წარმოვადგინოთ როგორც მონაცემთა ბაზის კონცეპტუალური პროექტის ნახაზი.

სამოდელო სისტემების დიდი ნაწილი შეიცავენ მდგრად ობიექტებს, ესე იგი ისეთებს, რომლებიც შესაძლებელია შევინახოთ

მონაცემთა ბაზაში და შემდეგ საჭიროებისამებრ გამოვიძახოთ. ამისათვის ყველაზე ხშირად იყენებენ რელაციურ, ობიექტ-ორიენტირებულ ან ჰიბრიდულ ობიექტ-რელაციურ მონაცემთა ბაზებს. მონაცემთა ბაზების ლოგიკური პროექტირებისათვის ხშირად იყენებენ დიაგრამებს "არსება-კავშირი" (**-ღ** დიაგრამები). მაგრამ თუ კლასიკურ **-ღ** დიაგრამებზე ძირითადი ყურადღება გამახვილებულია მხოლოდ მონაცემებზე, კლასების დიგრამით შესაძლებელია ქცევის მოდელირებაც.

ლოგიკური სქემის მოდელირება წარმოებს შემდეგნაირად:

1. მოვახდინოთ მოდელში შემავალი კლასების იდენტიფიცირება, რომელთა მდგომარეობაც უნდა შენახულ იქნას.

2. შევქმნათ კლასების დიგრამა და დავახასიათოთ ისინი როგორც მდგრადი, სტანდარტული მონიშნული მნიშვნელობით **persistent** (მდგრადი).

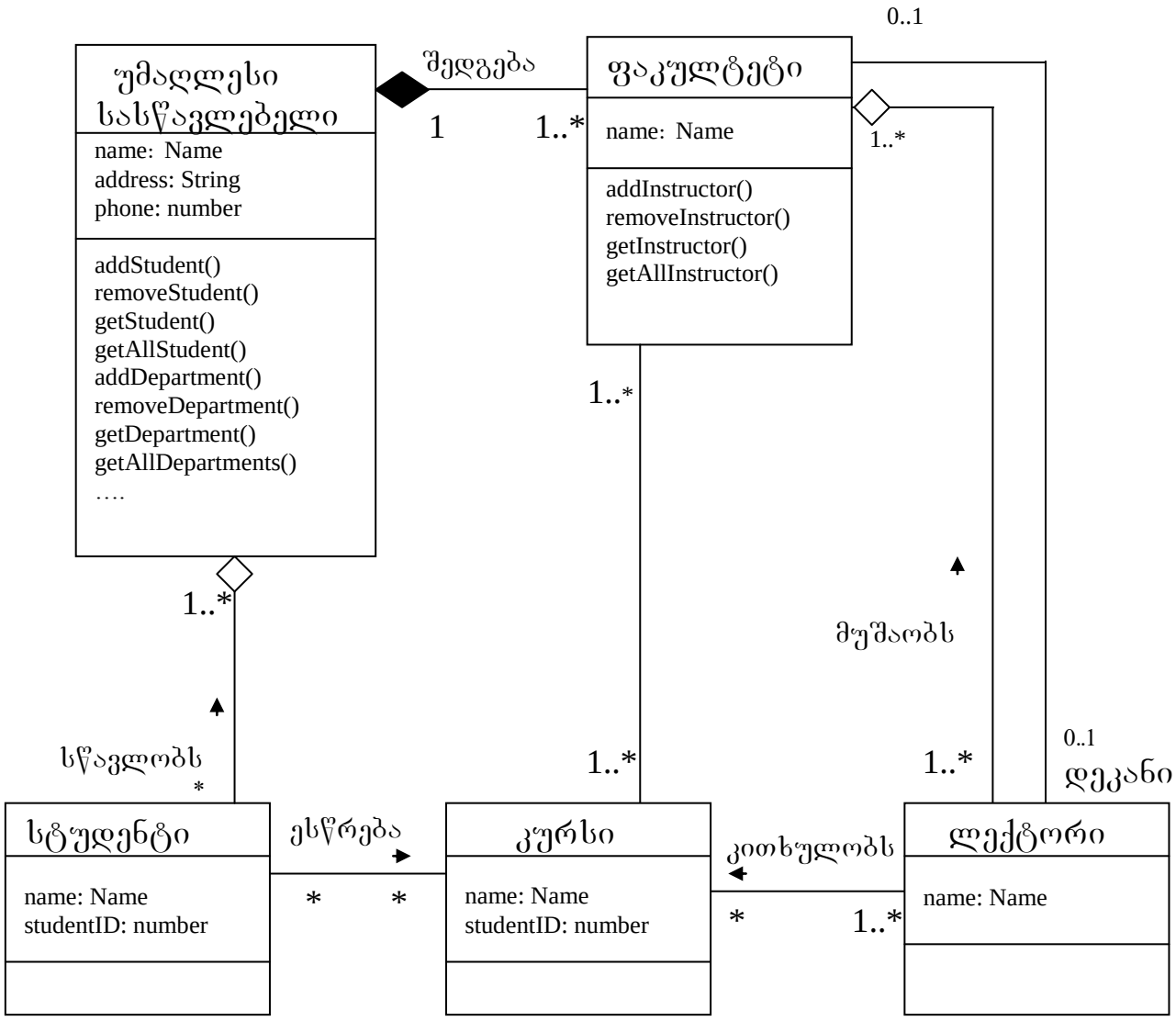
3. გავხსნათ კლასების სტრუქტურული თავისებურებანი. ეს ნიშნავს, რომ დეტალურად უნდა მიუთითოთ ატრიბუტები და განსაკუთრებული ყურადღება მივაქციოთ ასოციაციას და მათ ჯერადობას.

4. მოვძებნოთ ტიპიური სტრუქტურული სახეები, რომლებიც ართულებენ ფიზიკურ მონაცემთა ბაზის პროექტირებას, მაგ. ციკლური ასოციაციები, ასოციაცია ერთი ერთზე და **ნ**-არული ასოციაციები. საჭიროების შემთხვევაში შევქმნათ შუალედური აბსტრაქციები ლოგიკური სქემის გამარტივებისათვის.

5. განვიხილოთ ამ კლასების ქცევები, გავხსნათ ოპერაციები, რომლებიც მნიშვნელოვანია მონაცემებთან მიმართვისათვის და მათი მთლიანობის დასაცავად.

6. ეცადეთ გამოიყენოთ ინსტრუმენტალური საშუალებები, რომლებიც საშუალებას მოგვცემენ გარდავქმნათ ლოგიკური პროექტი ფიზიკურში.

ნახაზზე ქვევით მოყვანილია კლასების ერთობლიობა აღებული უმაღლესი სასწავლებლის საინფორმაციო სისტემიდან. ეს ნახაზი ფაქტიურად წარმოადგენს მანამდე მოყვანილი კლასების დიაგრამის გაფართოებას და შეიცავს საკმაო რაოდენობის დეტალებს ფიზიკური მონაცემთა ბაზის კონსტრუირებისათვის.



ყველა კლასი დიაგრამაზე მონიშნულია როგორც მდგრადი (persistent), ესე იგი მათი ეგზემპლიარები უნდა იყვნენ მონაცემთა ბაზაში. მოყვანილია აგრეთვე კლასის ატრიბუტები.

ორი კლასი (უმაღლესი სასწავლებელი და ფაკულტეტი) შეიცავენ რამოდენიმე ოპერაციას, რომლებიც უზრუნველყოფენ მათზე მანიპულირებას. ეს ოპერაციები ჩართული იქნა მათი

მნიშვნელობის გამო მონაცემთა მთლიანობის დაცვის თვალსაზრისით. ცხადია, არსებობენ სხვა ოპერაციებიც, რომლებიც სასურველი იყო განგვეხილა მსგავსი კლასების მოდელირებისას, მაგალითად დაკვეთა სტუდენტის კურსზე ჩარიცხვისათვის საჭირო წინასწარი ცოდნის შესახებ. მაგრამ ეს, უფრო ბიზნეს-წესებია, ვიდრე ოპერაციები მონაცემთა მთლიანობის დასაცავად, ამიტომ უკეთესი იქნება ისინი აბსტრაქციის უფრო მაღალ დონეზე მოვათავსოთ.

2.4. კლასების დამატებითი თვისებები

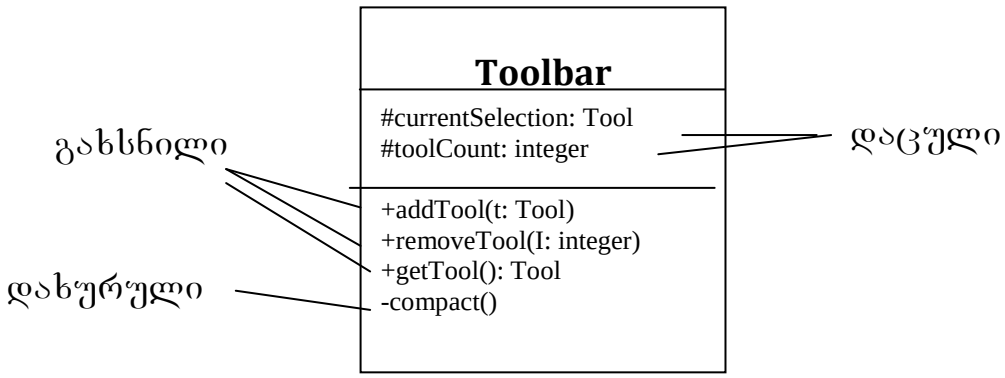
კლასები ობიექტ-ორიენტირებული სისტემების ყველაზე მნიშვნელოვანი სამშენებლო ბლოკებია. მაგრამ კლასები წარმოადგენენ მხოლოდ ერთ ნაირსახეობას -ის უფრო რთული სამშენებლო ბლოკის - კლასიფიკატორის. კლასიფიკატორი – ეს მექანიზმია, რომელიც აღწერს სტრუქტურულ და ქცევით თვისებებს. კლასიფიკატორების რიცხვს მიეკუთვნება კლასები, ინტერფეისები, მონაცემთა ტიპები, სიგნალები, კომპონენტები, კვანძები, პრეცედენტები და ქვესისტემები.

კლასიფიკატორები, და განსაკუთრებით კლასები, ხასიათდებიან დიდი რაოდენობის დამატებითი თვისებებით, გარდა იმ მარტივებისა, რომელიც განხილული იყო წინა თავებში. ამ თვისებებს მიეკუთვნება ხედვა, მოქმედების არე, ჯერადობა და სხვა.

ხედვა. ერთერთი დეტალი, რომელიც საკმაოდ მნიშვნელოვანია ატრიბუტებისა და ოპერაციებისათვის ეს არის მათი ხედვა. თვისების ხედვა მიუთითებს შეიძლება თუ არა იგი გამოყენებულ იქნას სხვა კლასიფიკატორების მიერ. ბუნებრივია, ეს გულისხმობს თვით კლასიფიკატორის ხედვას. ერთ კლასიფიკატორს შეუძლია “შეხედოს” მეორეს, თუ იგი იმყოფება პირველის მოქმედების

სფეროში და მათ შორის არსებობს პირდაპირი ან უშუალო მიმართება. – ში არსებობს ხედვის სამი დონე:

- **public(**გახსნილი) – ნებისმიერი გარე კლასი, რომელიც "ხედავს" მოცემულს შეუძლია ისარგებლოს მისი გახსნილი თვისებებით. აღინიშნება ნიშნით +(პლიუსი) ატრიბუტის ან ოპერაციის წინ.
- **proteqted**(დაცული) – ნებისმიერ შვილობილს მოცემული კლასისა შეუძლია ისარგებლოს მისი დაცული თვისებებით. აღინიშნება №(დიეზ) ნიშნით.
- **private**(დახურული) – მხოლოდ მოცემულ კლასს შეუძლია ისარგებლოს დახურული თვისებებით. აღინიშნება სიმბოლოთი –(მინუსი).



კლასის თვისებების ხედვას განსაზღვრავენ იმისათვის, რომ დამალონ მისი რეალიზაციის დეტალები და უჩვენოთ მხოლოდ ის თავისებურებანი, რომლებიც აუცილებელია მოვალეობების განსახორციელებლად. სწორედ ამაში მდგომარეობს ინფორმაციის დახურვის მიზეზი, ურომლისოდ შეუძლებელია საიმედო და მოქნილი სისტემის შექმნა. თუ ხედვის სიმბოლო არ არის მითითებული, ჩვეულებრივ იგულისხმება, რომ თვისება არის გახსნილი.

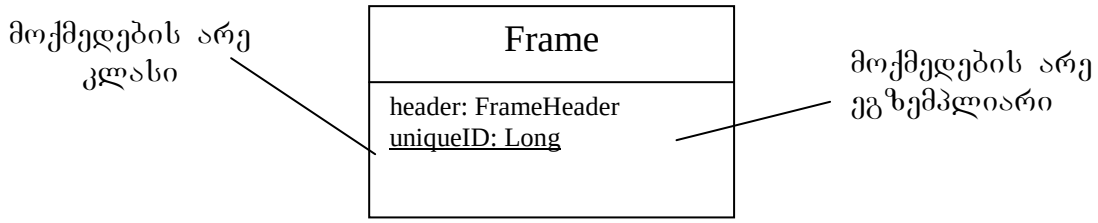
მოქმედების არე. კლასიფიკატორის ატრიბუტების და ოპერაციების კიდევ ერთ მნიშვნელოვან მახასიათებელს

წარმოადგენს მოქმედების არე. რომელიმე თვისებისათვის მოქმედების არის მითითებით აღნიშნავენ, გამოავლენს თუ არა იგი თავის თავს სხვა სახით კლასის ყოველ ეგზემპლიარში, თუ თვისების ერთი და იმავე მნიშვნელობა ერთობლივად გამოიყენება ყველა ეგზემპლიარის მიერ. – ში განსაზღვრულია მოქმედებათა არის ორი სახე:

- **instance** (ეგზემპლიარი) – კლასის ყოველ ეგზემპლიარს აქვს მოცემული თვისების საკუთარი მნიშვნელობა.
- **classifier** (კლასიფიკატორი) – კლასის ყველა ეგზემპლიარები ერთობლივად იყენებენ მოცემული თვისების საერთო მნიშვნელობას.

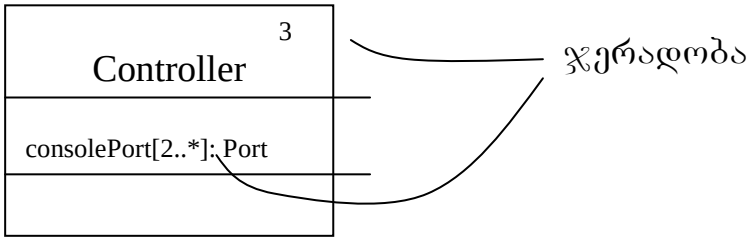
როგორც წესი, სამოდელო კლასების თვისებებს აქვთ მოქმედების სფერო ეგზემპლიარი. თვისებები მოქმედების არით კლასიფიკატორი ყველაზე ხშირად გამოიყენება დაკეტილი ატრიბუტების აღწერისათვის.

ქვემოთ მოყვანილ ნახაზზე თვისება, რომელსაც აქვს მოქმედების არე **classifier** ხაზგასმულია, ხოლო თუ ხაზგასმული არ არის იგულისხმება მოქმედების არე **instance**.



ჯერადობა. კლასებთან მუშაობისას იგულისხმება, რომ შეიძლება არსებობდეს მისი ნებისმიერი რაოდენობის ეგზემპლიარები. თუ რასაკვირველია ის არ არის აბსტრაქტული კლასი, რომელსაც საერთოდ არ გააჩნია ეგზემპლიარები, თუმცა მის შვილობილებს შეიძლება გააჩნდეთ ნებისმიერი რაოდენობით.

კლასის ეგზემპლიარების რაოდენობას უწოდებენ მის ჯერადობას. კლასის ჯერადობა მიეთითება გამოსახულებით ზედა მარჯვენა კუთხეში. ჯერადობა გამოიყენება ატრიბუტების მიმართაც. ატრიბუტის ჯერადობა იწერება გამოსახულების სახით კვადრატულ ფრჩხილებში და განთავსდება ატრიბუტის დასახელების შემდეგ.



2.5. ინტერფეისები, ტიპები და როლები

სისტემის დაპროექტებისას მნიშვნელოვანია ავაგოთ სისტემები ამოცანათა მკაფიო გამიჯვნით. ეს გულისხმობს, რომ სისტემის განვითარებისას ცვლილებას მის ერთ ნაწილში არ შეეხოს დანარჩენს. ამ მიზნის მისაღწევათ აუცილებელია სისტემის დამაკავშირებელი კვანძების სპეციფიცირება, რომლებსაც უკავშირდებიან დამოუკიდებლად ცვალებადი ნაწილები. დავადგენთ რა შემდეგში უკეთეს რეალიზაციას, თქვენ შეგეძლებათ შეცვალოთ ძველი ისე რომ არ შევაწუხოთ მომხმარებელი.

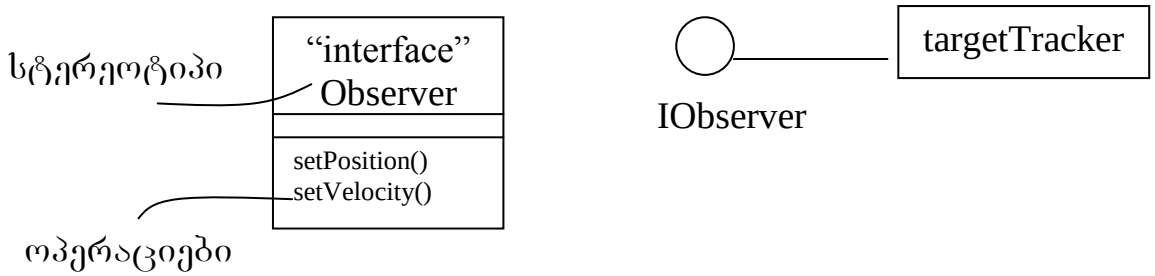
დამაკავშირებელი კვანძების მოდელირებისათვის გამოიყენება ინტერფეისები. **ინტერფეისები** – ეს ოპერაციების ერთობლიობაა, რომლებიც ახდენენ მომსახურების სპეციფიცირებას, რომელსაც წარმოადგენენ კლასები. ვაცხადებთ რა ინტერფეისს, ჩვენ გვეძლევა საშუალება დავადგინოთ აბსტრაქციის სასურველი ქცევა, მათი რეალიზაციისაგან დამოუკიდებლად. კლიენტებს შეუძლიათ დაეყრდნონ გამოცხადებულ ინტერფეისებს, ხოლო თქვენ შემდეგში შეგეძლებათ შექმნათ ან იყიდოთ მისი ნებისმიერი რეალიზაცია.

მთავარია მხოლოდ მან შეასრულოს მოვალეობები, გამოცხადებული ინტერფეისით.

თითქმის ყველა თანამედროვე დაპროგრამების ენები, მათ შორის **JAVA** და **CORBA IDL**, იყენებენ ინტერფეისების კონცეფციას.

ამრიგად, ინტერფეისებს უწოდებენ ოპერაციების ნაკრებს, რომლებიც კლასების მიერ გამოიყენება მომსახურების სპეციფიცირებისათვის.

გრაფიკულად ინტერფეისი წარმოიდგინება წრის სახით. გარდა ამისა, იგი შეიძლება წარმოვადგინოთ როგორც სტერეოტიპული კლასი, რათა გაიხსნას ოპერაციები და სხვა თვისებები.



ყოველ ინტერფეისს უნდა გააჩნდეს სხვებისაგან განსხვავებული სახელი. ინტერფეისის სახელი წარმოიდგინება ტექსტური სტრიქონის სახით. იმისათვის, რომ განასხვავონ კლასისაგან ინტერფეისის სახელს დასაწყისში ემატება I.

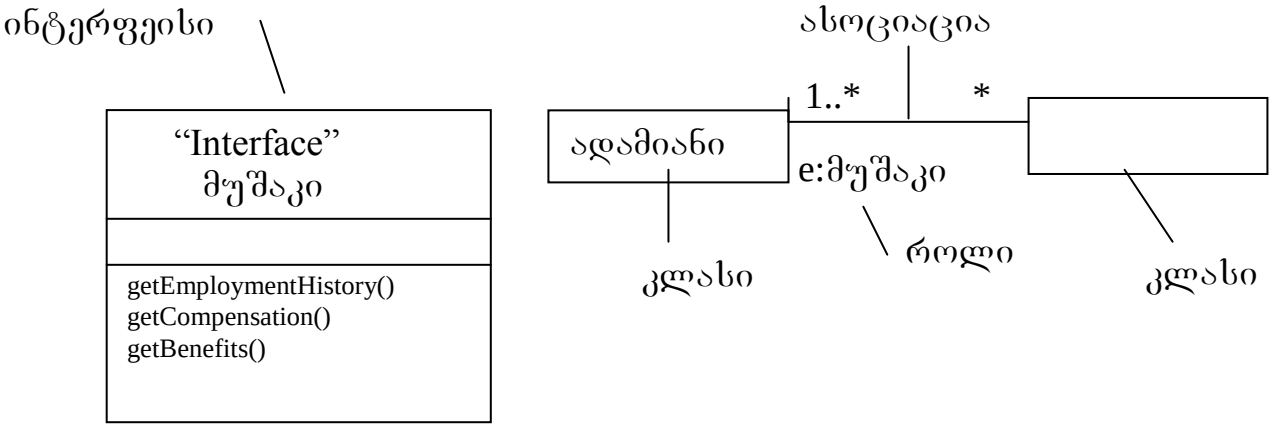
ინტერფეისი ახდენს კლასის სპეციფიცირებას, მაგრამ არ ახდენს არავითარ შეზღუდვას მის რეალიზაციაზე. კლასი შეიძლება რამოდენიმე ინტერფეისების რეალიზებას ახდენდეს. ამასთან ისინი მოვალეობას იღებენ შეასრულონ თავიანთი ყველა კონტრაქტები, უნდა შეიცავდნენ მეთოდებს ინტერფეისით გამოცხადებული ოპერაციების რეალიზებისათვის. ასევე კლასი შესაძლებელია დამოკიდებული იყოს რამოდენიმე ინტერფეისისაგან. ამ დროს ის ელოდება, რომ გამოცხადებული კონტრაქტები შესრულდებიან მათი რეალიზებადი გარკვეული კომპონენტების ნაკრებით. სწორედ ამიტომ ამბობენ, რომ ინტერფეისი წარმოადგენს სისტემაში

დამაკავშირებელ კვანძს. იგი განსაზღვრავს კონტრაქტის პირობებს, რომლის შემდეგაც ორივე მხარე კლიენტი და მომწოდებელი შეუძლიათ იმოქმედონ ერთმანეთისაგან დამოუკიდებლად, მთლიანად დაეყრდნონ რა ურთიერთ მოვალეობებს.

კლასი, რომელიც იყენებს ინტერფეისს, მასთან დაკავშირება ხდება დამოკიდებულების მიმართებით. შევქმნით რა ახალ ინტერფეისს, პირველ რიგში უყურებთ ოპერაციათა სიმრავლეს, რომელიც განსაზღვრავენ კლასის სერვისს. ინტერფეისი განსაზღვრავს კონტრაქტის პირობებს და კლასის ყველა ეგზემპლიარები უნდა იცავდნენ ამ პირობებს. ამასთან კონკრეტულ კონტექსში ეგზემპლიარს შეუძლია წარმოადგინოს მხოლოდ ის ინტერფეისები, რომლებიც მოცემულ სიტუაციას შეესაბამება. ეს ნიშნავს, რომ ყოველი ინტერფეისი განსაზღვრავს როლს, რომელსაც თამაშობს ობიექტი. როლი – ეს გარკვეული არსების ქცევაა კონკრეტულ კონტექსში ან სხვა სიტყვებით – პირი, რომლითაც აბსტრაქცია მიმართულია სამყაროს მიმართ. მაგალითად, განვიხილოთ კლასი ***ადამიანის*** ეგზემპლიარი. კონტექსისგან დამოკიდებულებით ამ კლასის ეგზემპლიარი შესაძლებელია თამაშობდეს მუშაკის, მყიდველის, მენეჯერის, მომღერლის და ა.შ. როლს. შესაბამისად, ობიექტი წარუდგენს სამყაროს ამა თუ იმ "სახეს" და მასთან ურთიერთქმედებაში მყოფი კლიენტები ელოდებიან მისგან შესაბამის ქცევას. მაგალითად, კლასი ***ადამიანის*** ეგზემპლიარი მენეჯერის როლში ფლობს სხვა თვისებებს, ვიდრე მას ექნება მომღერლის როლში.

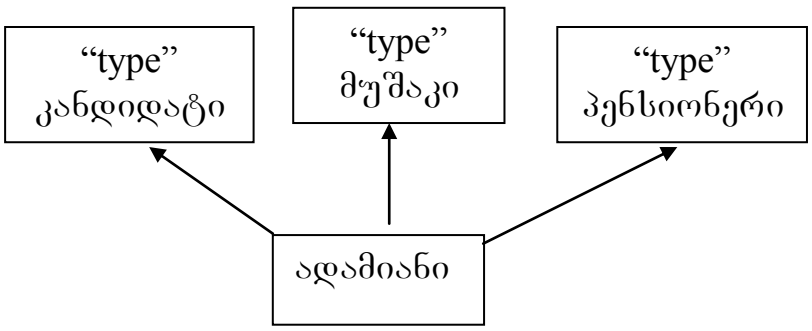
როლი, რომელსაც ერთი აბსტრაქცია თამაშობს მეორეს მიმართ, შეიძლება აღვწეროთ შევავსებთ რა ასოციაციის შესაბამის ბოლო წერტილს ინტერფეისის სახელით. მაგალითისათვის ნახაზზე ნაჩვენებია ინტერფეისი ***მუშაკი***, რომლის განსაზღვრა მოიცავს სამ ოპერაციას. კლასებს შორის **ადამიანი** და **კომპანია** არსებობს

ასოციაცია, რომლის კონტექსში **ადამიანი** თამაშობს **ე** როლს, რომელიც ეკუთვნის ტიპს **მუშაკი**. სხვა ასოციაციაში ეს კლასი შესაძლებელია "სხვა სახით იყოს წარმოდგენილი". მოცემულ შემთხვევაში, კლასი **ადამიანი** თამაშობს კლასისათვის **კომპანია** მუშაკის როლს და მოცემულ კონტექსში კომპანიისათვის ხილვადია და არსებითი იქნება მხოლოდ თვისებები, რომლებიც განისაზღვრებიან მოცემული როლით.



აბსტრაქციების სემანტიკის ფორმალური მოდელირებისათვის და მათი გარკვეულ ინტერფეისთან შესაბამისობისათვის გამოიყენება სტერეოტიპი **type**. ეს არის კლასის სტერეოტიპი, რომლის მეშვეობით განისაზღვრება ობიექტების მოქმედების არე ოპერაციებთან ერთად. ტიპის კონცეფცია მჭიდროდ არის დაკავშირებული ინტერფეისის კონცეფციასთან, მხოლოდ იმ განსხვავებით, რომ ტიპის აღწერა შესაძლებელია შეიცავდეს ატრიბუტებს, ხოლო ინტერფეისის აღწერა – არა.

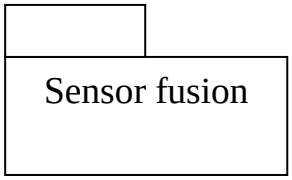
ქვედა ნახაზზე მოყვანილია როლები, რომელსაც კლასი ***ადამიანი*** თამაშობს რესურსების მართვის სისტემის კონტექსში. მოცემული ნახაზიდან ჩანს, რომ კლასი ***ადამიანის*** ეგზემპლიარები შესაძლებელია მივაკუთვნოთ სამიდან ერთ ტიპს – კანდიდატი, მუშაკი და პენსიონერი, რომლებიც წარმოიდგინებიან სტერეოტიპული კლასების სახით.



2.6. პაკეტები

დიდი სისტემების პროექტირება განსაზღვრავს პოტენციალურად დიდი რაოდენობის კლასებს, ინტერფეისებს, კვანძებს, კომპონენტებს, დიაგრამებს და სხვა ელემენტებს. მათი გაგების გამარტივებისათვის უნდა მოვახდინოთ ამ არსების ორგანიზება უფრო დიდ ბლოკებში, რომელსაც UML-ში უწოდებენ პაკეტებს(Packages).

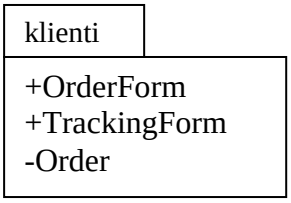
პაკეტი – ეს მოდელის ელემენტების ორგანიზების საშუალებაა უფრო დიდ ბლოკში, რომელთა მანიპულირება შემდგომში შესაძლებელია როგორც ერთიანი მთლიანის. შესაძლებლობა გვეძლევა ვმართოდ პაკეტში შემავალი არსების ხედვა, ისე, რომ ზოგიერთი იქნება გარედან ხედვადი, ხოლო სხვები – არა. ამის, გარდა პაკეტების საშუალებით შესაძლებელია წარმოვადგინოთ სისტემის არქიტექტურის სხვადასხვა სახე. კარგათ დაპროექტებული პაკეტი აჯგუფებს სემანტიკურად ახლოს მყოფ ელემენტებს, რომლებსაც აქვთ ტენდენცია იცვლებოდნენ ერთად. გრაფიკულად პაკეტი წარმოიდგინება შემდეგი სახით

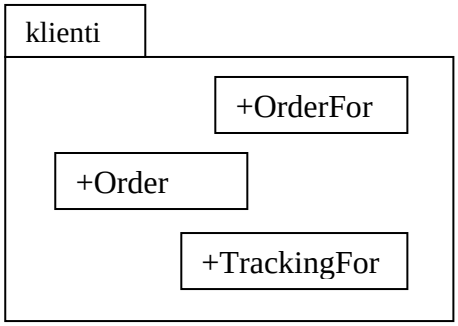


ყოველ პაკეტს უნდა გააჩნდეს დასახელება, რომელიც განასხვავებს მას სხვა პაკეტებისაგან. პაკეტის დასახელება – ტექსტური სტრიქონია.

პაკეტის შემადგებელი ელემენტები. პაკეტი როგორც ავღნიშნეთ შესაძლებელია შეიცავდეს სხვა ელემენტებს. ეს ნიშნავს, რომ ისინი გამოცხადებული უნდა იქნენ პაკეტის შიგნით. თუ პაკეტი იხურება, ისპობა მასზე მიკუთვნილი ელემენტები. ამასთან ელემენტი შესაძლებელია ეკუთვნოდეს მხოლოდ ერთ პაკეტს და უნდა ქონდეს თავისი უნიკალური სახელი. დაუშვებელია ერთი და იგივე ელემენტების არსებობა ერთნაირი დასახელებით, მაგრამ დაშვებულია სხვადასხვა ელემენტის არსებობა ერთი დასახელებით.

პაკეტებს, შესაძლებელია ეკუთვნოდეს სხვა პაკეტები, რაც საშუალებას იძლევა შევქმნათ მოდელის იერარქიული დეკომპოზიცია. მაგალითად, შესაძლებელია გვქონდეს კლასი Camera, რომელიც ეკუთვნის პაკეტს Vision, რომელიც თავის მხრივ არის პაკეტის Sensors შემადგენელი. შედგენილი დასახელება ამ კლასის იქნება Sensors::Vision::Camera. მაგრამ უნდა მოვერიდოთ პაკეტების ძალიან ღრმა შემცველობას – ორი-სამი დონე წარმოადგენს ზღვარს. პაკეტის შემცველობა შესაძლებელია წარმოვადგინოთ გრაფიკულად ან ტექსტური სახით:





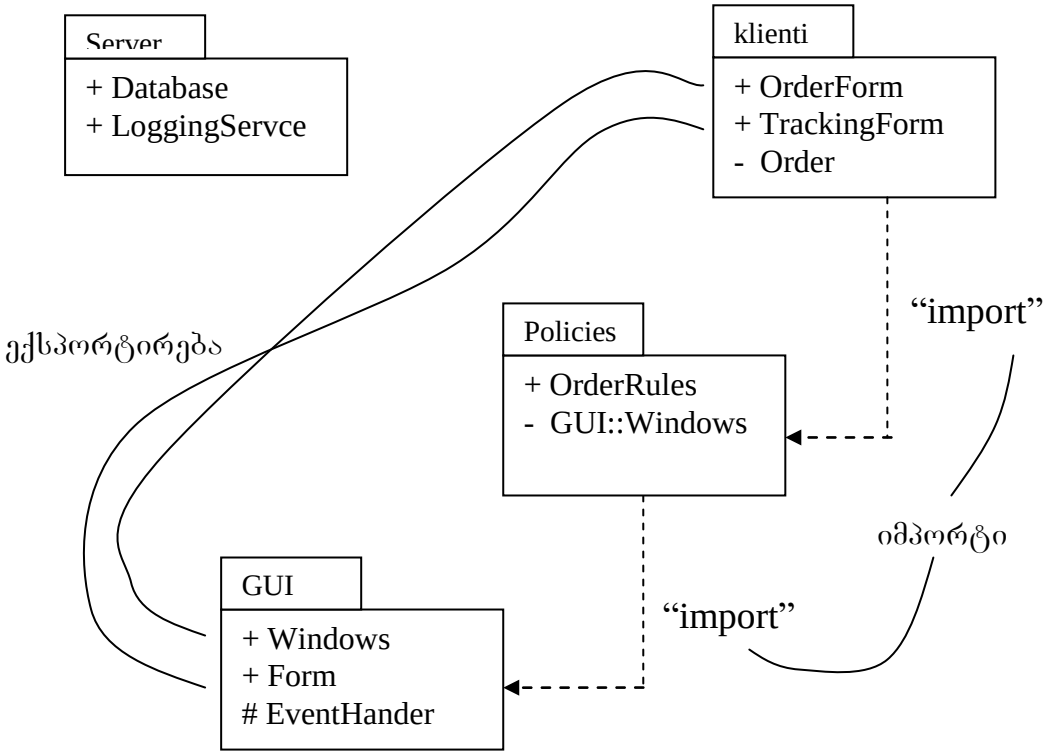
ხედვა. პაკეტის ელემენტების ხედვის კონტროლი შესაძლებელია ისევე, როგორც კლასის ოპერაციებისა და ატრიბუტების ხედვა. დუმილით ასეთი ელემენტები გახსნილია, ანუ ხედვადია ყველა ელემენტებისათვის ნებისმიერ პაკეტში, რომელიც ახდენს მის იმპორტირებას. დაცული ელემენტები ხედვადია მხოლოდ შვილობილებისათვის, ხოლო დახურული თავისი პაკეტის გარეთ საერთოდ არახედვადია.

პაკეტის შემადგენელი ელემენტების ხედვა აღინიშნება ხედვის სიმბოლოთი ამ ელემენტის დასახელების წინ. გახსნილი ელემენტებისათვის გამოიყენება სიმბოლო +(პლიუსი). პაკეტის ყველა გახსნილი ნაწილები შეადგენენ მის ინტერფეისს. ისევე როგორც კლასებისათვის დაცული ელემენტებისათვის გამოიყენება სიმბოლო #(დიეზი), ხოლო დახურულისათვის უმატებენ სიმბოლოს –(მინუსი). ბოლოს, დაცული ელემენტები ხედვადი იქნება მხოლოდ პაკეტებისათვის, რომლებიც მემკვიდრეობით არის მოცემულისათვის, ხოლო დახურული საერთოდ არახედვადია პაკეტის გარეთ, რომელშიც იგი გამოცხადებულია.

იმპორტი და ექსპორტი. დაუშვათ ჩვენ გვაქვს კლასი A ერთ პაკეტში, ხოლოკლასი B მეორეში და ორივე პაკეტი თანაბარია. ამასთან, A-ც და B-ც გამოცხადებულია ღიათ თავიანთ პაკეტებში. თავისუფალი მიმართვა ერთერთის მეორესთან არ არის შესაძლებელი, რადგან პაკეტის საზღვრები გაუმჭირვალეა. მაგრამ, თუ ერთი პაკეტი, რომელიც შეიცავს პაკეტს A, ახდენს B კლასის შემცველი პაკეტის იმპორტირებას, მაშინ A შეძლებს “შეხედოს” B-ს. მაგრამ B კი მაინც ვერ “შეხედავს” A-ს. იმპორტი საშუალებას აძლევს ერთი პაკეტის ელემენტებს მოახდინონ ერთმხვრივი მიმართვა მეორეს ელემენტებთან. UML-ში იმპორტის მიმართება მოდელირდება როგორც დამოკიდებულება სტერეოტიპით Import. სემანტიკურად მოფიქრებულ ბლოკებში აბსტრაქციების დაჯგუფებით და იმპორტის მეშვეობით მათთან მიმართვის კონტროლით, ჩვენ საშუალება გვეძლევა ვმართოდ რთული სისტემა, რომელიც ითვლის რამოდენიმე ათეულ აბსტრაქციას.

პაკეტის ღია ელემენტებს უწოდებენ ექსპორტირებულებს. ასე მაგალითად, ნახ.2.6.1. –ზე პაკეტი GUI ექსპორტირებას უწევს ორ კლასს – Windows და Form. კლასი EventHandler პაკეტის დაცული ნაწილია. ქსპორტირებული ელემენტები იქნებიან ხედვადი მხოლოდ

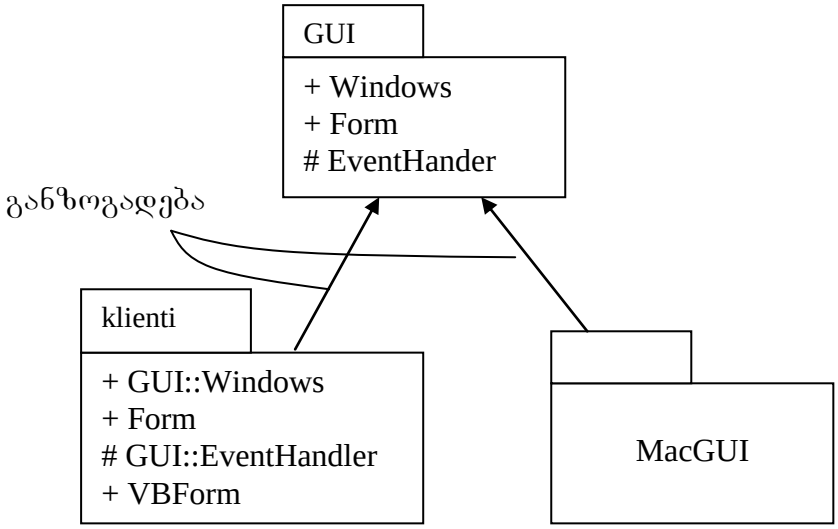
იმ პაკეტებისათვის, რომლებიც ახდენენ მოცემულის იმპორტირებას. როგორც ნახაზიდან ჩანს Policies ახდენს GUI-ს იმპორტირებას. მაშასადამე GUI::Window და GUI::Form იქნებიან მისგან ხედვადი. მაგრამ კლასი GUI::EventHandler არ იქნება ხედვადი, რადგან არის დაცული. მეორეს მხრივ, პაკეტი Server არ ახდენს GUI-ს იმპორტირებას, ამიტომ არა აქვთ უფლება GUI ელემენტებთან მიმართვისა. იმავე მიზეზით GUI-ს არ აქვს უფლება Server პაკეტის შემცველობასთან მიმართვისა.



ნახ.2.6.1. იმპორტი და ექსპორტი

დამოკიდებულება იმპორტი არ არის ტრანზიტიული. მოცემულ მაგალითში პაკეტი Client ახდენს Policies იმპორტირებას, ხოლო Policies – GUI-ს, მაგრამ ეს არ ნიშნავს, რომ Clienti ახდენს GUI-ს იმპორტირებას. მაშასადამე პაკეტიდან Client პაკეტი Policies ექსპორტირებულ ნაწილებთან მიმართვა დაშვებულია, ხოლო GUI-ს ექსპორტირებულ ნაწილებთან არა.

განზოგადება. პაკეტებს შორის განსაზღვრულია ორი ტიპის მიმართება: იმპორტზე დამოკიდებულების, რომელიც გამოიყენება პაკეტში ელემენტების იმპორტისათვის, რომლებიც ექსპორტირებულები არიან სხვა პაკეტებით და განზოგადების, რომელიც გამოიყენება კლასთა ოჯახის სპეციფიცირებისათვის. მაგალითად, როგორც ჩანს ნახ. 2.6.2.-დან GUI ექსპორტირებას უწევს ორ კლასს – Windows და Form. კლასი EventHandler პაკეტის დაცული ნაწილია. რსებობს პაკეტ GUI ორი სპეციალიზაცია WindowsGUI და MacGUI. ისინი მემკვიდრეობით იღებენ თავისი მშობლის ღია და დაცულ ელემენტებს. ღოგორც კლასების შემთხვევაში, პაკეტებს შესაძლებლობა აქვთ ჩაანაცვლონ მშობლის ელემენტები ან დაამატონ ახალი. მაგალითად, პაკეტი WindowsGUI შეიცავს კლასებს GUI::Window და GUI::EventHandler. არდა ამისა, მასში გადაწერილია კლასი Form და დამატებულია ახალი კლასი VBForm.



ნახ.2.6.2. პაკეტებს შორის განზოგადება

განზოგადებაში მონაწილე პაკეტები მიყვებიან ჩართვის იმავე პრინციპს, რასაც კლასები. სპეციალიზირებული პაკეტები (როგორიც არის WindowsGUI) შესაძლებელია გამოყენებულ იქნან ყველგან, სადაც დასაშვებია მათი მშობლების გამოყენება (GUI-ს მსგავსად).

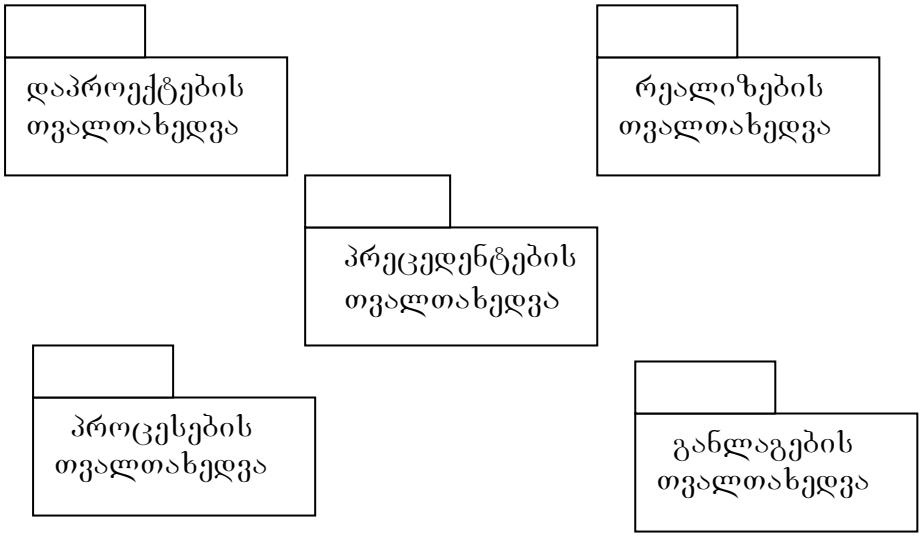
გამოყენების ტიპიური ხერხები. ველაზე ხშირად პაკეტებს იყენებენ მოდელირების ელემენტების ორგანიზებისათვის ჯგუფებში, რომელთანაც შემდეგ შესაძლებელია მუშაობა როგორც ერთიანი მთელის. მარტივი სისტემის შექმნისას, შესაძლებელია პაკეტები საერთოდ არ დაგვჭირდეს, რადგა ყველა ჩვენი აბსტრაქციები შესაძლებელია მოთავსდეს ერთ პაკეტში. აგრამ უფრო რთული სისტემების დამუშავებისას აღმოვაჩენთ, რომ ბევრი კლასები, კომპონენტები, კვაძები, ინტერფეისები და დიაგრამებიც კიბუნებრივად იყოფიან ჯგუფებათ. შწორედ ამ ჯგუფება ვამოდელირებთ ჯგუფებში.

ყველაზე ხშირად პაკეტების მეშვეობით ერთი ტიპის ელემენტებს აერთიანებენ ჯგუფებში. მაგალითად, კლასები და მათი მიმართებები სისტემის წარმოდგენიდან დაპროექტების თვალთახედვით შესაძლებელია გავაერთიანოდ შესაძლებელია დავყოთ რამოდენიმე პაკეტად და ვაკონტროლოთ მასთან მიმართვა იმპორტზე დამოკიდებულებით.

პაკეტებით შესაძლებელია აგრეთვე სხვადასხვა ტიპის ელემენტების ორგანიზება. ეს განსაკუთრებით ეხება ისეთ შემთხვევას, როდესაც სისტემა მუშავდება რამოდენიმე კოლექტივის მიერ, რომლებიც განლაგებულნი არიან სხვადასხვა ადგილზე. ამ შემთხვევაში პაკეტები გამოიყენებიან კონფიგურაციის მართვისათვის. განვალაგებთ რა მათში ყველა კლასებს და დიაგრამებს, სხვადასხვა კოლექტივის წევრებს დამოუკიდებლად შეეძლებათ ამოიღონ და მოათავსოთ უკან სასურველი ელემენტი.

პაკეტების გამოყენება მონათესავე ელემენტების დაჯგუფებისათვის საკმაოდ მნიშვნელოვანია – მის გარეშე რთული სისტემების დამუშავება შეუძლებელია. მაგრამ პროგრამული სისტემების არქიტექტურული სახეების განხილვისას იქმნება მოთხოვნა უფრო რთული ბლოკების შექმნისა. კერძოთ,

არქიტექტურული სახეებიც შესაძლებელია პაკეტების მეშვეობით დავამოდელიროთ. ამისათვის პირველ რიგში უნდა დადგინდეს თუ რომელი სახეებია ჩვენი სისტემისათვის მნიშვნელოვანი. ჩვეულებრივ ეს სახეებია - პრეცედენტების, დაპროექტების, პროცესების რეალიზაციისა და განლაგების თვალთახედვა. ასეთი დაჯგუფებისათვის, შესაბამის პაკეტებში მოვათავსებთ ელემენტებს და დიაგრამებს, რომლებიც აუცილებელია ყოველი ცალკეული სახეობის ვიზუალირებისა და სპეციფიცირებისათვის. აუცილებლობის შემთხვევაში ყოველი სახეობის ელემენტები შესაძლებელია დავაჯგუფოთ უფრო მცირე პაკეტებათ. ელემენტებს შორის სხვადასხვა სახეობიდან, ცხადია იარსებებს დამოკიდებულების მიმართება, ამიტომ ზოგადათ უნდა გავხსნათ ყველა სახეობები სისტემის ზედა დონეზე ყველა დანარჩენი სახეობებისათვის იმავე დონეზე. მაგალითისათვის, ნახ.2.6.3-ზე მოყვანილია ზედა დონის კანონიკური დეკომპოზიცია, რომელიც გამოიყენება ყველაზე რთული სისტემებისათვისაც.



ნახ.2.6.3. არქიტექტურული სახეობების მოდელირება

2.7. ეგზემპლიარები

ტერმინებს “სამ ოთახიანი სახლი” და “ჩემი სამ ოთახიანი სახლი” არსებობს ფუნდამენტალური განსხვავება. პირველი – ეს მხოლოდ აბსტრაქციაა, რომელიც აღწერს სახლის გარკვეულ ტიპს სხვადასხვა თვისებებით, მაშინ როდესაც მეორე წარმოადგენს კონკრეტულ ეგზემპლიარს ამ აბსტრაქციის, რომელიც არსებობს რეალურ სამყაროში და მის ყოველ თვისებას აქვს რეალური მნიშვნელობა.

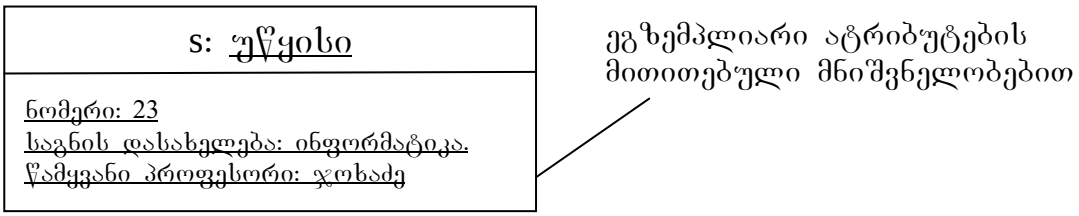
აბსტრაქცია აღწერს საგნის იდეალურ არსს, ეგზემპლიარი – მის კონკრეტულ მატერიალიზაციას. ერთ აბსტრაქციას შეიძლება გააჩნდეს რამოდენიმე ეგზემპლიარი. მოცემული ეგზემპლიარისათვის ყოველთვის არსებობს აბსტრაქცია, რომელიც განსაზღვრავს საერთო მახასიათებლებს ყველა მსგავსი ეგზემპლიარებისათვის.

ეგზემპლიარს (**Instance**) უწოდებენ აბსტრაქციის კონკრეტულ მატერიალიზაციას, რომლის მიმართ შეიძლება გამოყენებულ იქნას ოპერაციები და რომელსაც შეუძლია შეინახოს მათი შედეგები. ცნებები “ეგზემპლიარი” და “ობიექტი” პრაქტიკულად სინონიმებია. ეგზემპლიარს გამოხატავენ ხასგასმული სახელით.

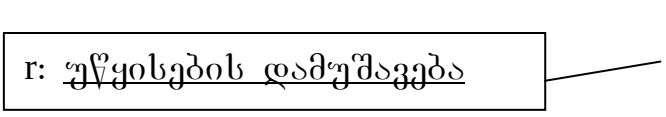
ჩვეულებრივ ობიექტს უწოდებენ კლასის კონკრეტულ მატერიალიზაციას. ობიექტები – ეს კლასების ეგზემპლიარებია. ობიექტი არა მარტო იკავებს ადგილს რეალურ სამყაროში, მათზე შესაძლებელია ასევე მანიპულირება. ოპერაციები, რომლებიც სრულდება ობიექტებზე, გამოცხადდება მის აბსტრაქციაში. ეს მიუთითებს იმაზე, რომ ყოველი ობიექტი ხასიათდება მდგომარეობით.

ობიექტის მდგომარეობას უწოდებენ მისი ყველა თვისებების ერთობლიობას და მათ მიმდინარე მნიშვნელობებს. თვისებათა რიცხვში შედის ობიექტთა ატრიბუტები. მაშასადამე, ობიექტის მდგომარეობა დინამიურია და მისი ვიზუალიზაციისას ფაქტიურად

აღიწერება მისი მდგომარეობა დროის მოცემულ მომენტში და სივრცის მოცემულ წერტილში. ვასრულებთ რა ობიექტზე ოპერაციას, ფაქტიურად ვცვლით მის მდგომარეობას. მაგრამ, აქვე უნდა აღინიშნოს, რომ ობიექტის გამოკითხვისას მდგომარეობა არ იცვლება. ობიექტის მდგომარეობათა შეცვლა შეიძლება გამოვსახოთ ურთიერთქმედების დიაგრამაზე, დავხაზავთ რა მას რამოდენიმეჯერ ან შესაძლებელია გამოვიყენოთ მოცემული პროცესის აღწერისათვის ავტომატი.



გარდა ამისა, შესაძლებელია გამოვაცხადოთ აქტიური კლასები(იხ. თავი 3), რომლებიც ახდენენ პროცესების მატერიალიზებას და გამოვყოთ აქტიური კლასების ეგზემპლიარები. ყოველი აქტიური ობიექტი წარმოადგენს მართვის ნაკადის ამოსავალ წერტილს და შეიძლება გამოვიყენოთ სხვადასხვა მართვის ნაკადების დასახელებისათვის.

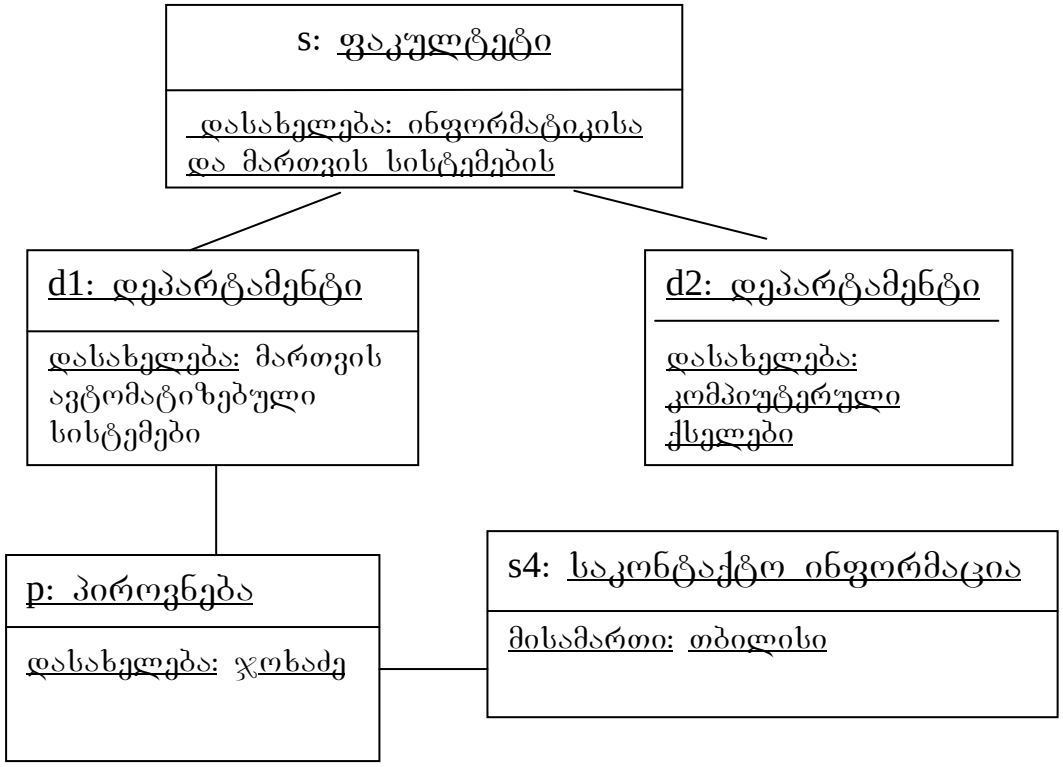


2.8. ობიექტების დიაგრამა

ობიექტების დიაგრამა საშუალებას იძლევა მოვახდინოთ იმ არსების ეგზემპლიართა მოდელირება, რომლებიც არსებობენ კლასების დიაგრამაზე. ობიექტების დიაგრამაზე ნაჩვენებია ობიექტების სიმრავლე და მათ შორის მიმართებები დროის რომელიღაც მომენტში, როგორც ეს ნაჩვენებია ნახ.2.7.1.-ზე.

ობიექტების დიაგრამა გამოიყენება სისტემის სტატიკური სახით მოდელირებისათვის პროექტირებისა და პროცესების

თვალთახედვით. ობიექტების დიაგრამით ახდენენ ობიექტების სტრუქტურის მოდელირებას.



ნახ.2.7.1.

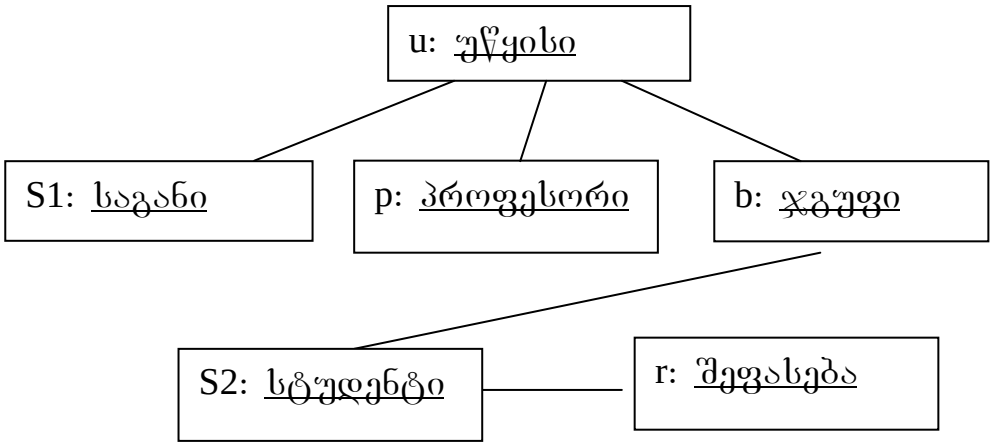
ობიექტთა სტრუქტურის მოდელირება გულისხმობს სისტემის ობიექტთა "სურათის" მიღებას დროის მოცემულ მომენტში. ობიექტების დიაგრამა წარმოადგენს დინამიური სცენარის სტატიკურ საფუძველს, რომელიც აღიწერება ურთიერთქმედების დიაგრამით.

ობიექტური სტრუქტურების მოდელირება ხორციელდება შემდეგნაირად:

1. მოახდინეთ იმ მექანიზმის იდენტიფიცირება, რომლის მოდელირებასაც აპირებთ. მექანიზმი წარმოადგენს გარკვეულ ფუნქციას ან სამოდელო სისტემის ქცევის ნაწილს, რომელშიც მონაწილეობს კლასები, ინტერფეისები და სხვა არსები.

2. ყოველი მექანიზმისათვის მოვახდინოთ კლასების, ინტერფეისების და კოოპერაციაში მონაწილე სხვა ელემენტების და მათ შორის მიმართებების იდენტიფიცირება.
3. განვიხილოთ მექანიზმის მუშაობის ერთ ერთი სცენარი. დავაფიქსიროთ ეს სცენარი დროის გარკვეული მომენტისათვის და გამოსახეთ ყველა ობიექტები, რომლებიც მონაწილეობენ მექანიზმში.
4. მიუთითეთ ყოველი ობიექტის მდგომარეობა და ატრიბუტების მნიშვნელობები, თუ ეს საჭიროა სცენარის გაგებისათვის.
5. მიუთითეთ აგრეთვე კავშირები ამ ობიექტებს შორის, რომლებიც წარმოადგენენ არსებული ასოციაციების ეგზემპლიარებს.

მაგალითისათვის ნახ.2.7.2.-ზე მოყვანილია ობიექტების ერთობლიობა, რომელიც აღწერს უმაღლეს სასწავლებელში სტუდენტთა მოსწრების აღრიცხვას.



ნახ. 2.7.2.

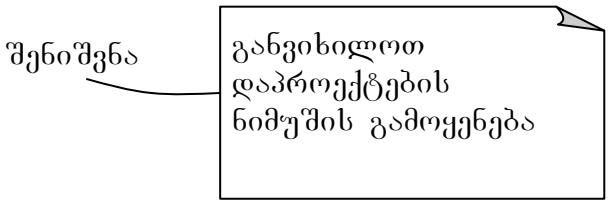
როგორც ნახაზიდან ჩანს, ერთი ობიექტი შეესაბამება უწყისს(u, კლასი უწყისის ეგზემპლიარი). ეს ობიექტი დაკავშირებულია ეგზემპლიართან s1 კლასისა საგანი, ეგზემპლიართან p კლასისა პროფესორი და ეგზემპლიართან b კლასისა ჯგუფი. თავის მხრივ, ობიექტი b დაკავშირებულია ობიექტთან s2 სტუდენტი, ხოლო ეს

უკანასკნელი ობიექტს r შეფასებები, რომლებშიც შეიტანება s2 სტუდენტის მონაცემები.

2.9. საერთო მექანიზმები

მუშაობა UML-ში საგრძნობლად მარტივდება მასში არსებული ოთხი მექანიზმის გამოყენებით. ეს არის სპეციფიკაციები, დამატებები, მიღებული დაყოფა და გაფართოების საშუალებები.

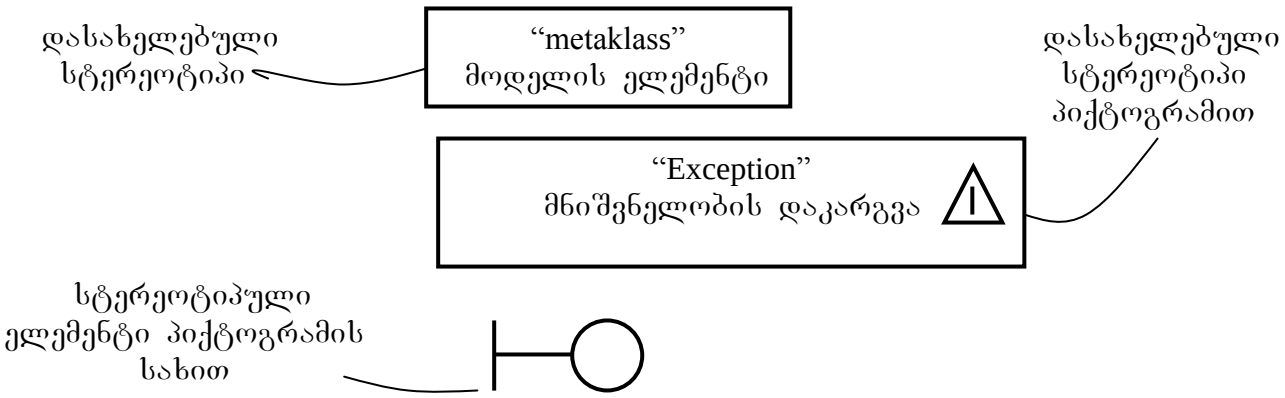
დამატების ყველაზე მნიშვნელოვანი სახეობაა შენიშვნა. იგი წარმოიდგინება გრაფიკული სიმბოლოთი შეზღუდვების გამოსახვისათვის ან კომენტარით, რომელიც უერთდება ელემენტს ან მათ ერთობლიობას. შენიშვნა გამოიყენება მოდელში დამატებითი ინფორმაციის ჩასართველად, მაგალითად, მოთხოვნების, მიმოხილვებისა და განმარტებების(იხ.ნახ2.9.1.). როგორც წესი შენიშვნებს იყენებენ იმისათვის, რომ თავისუფალ ფორმაში ჩავწეროთ დაკვირვება, მიმოხილვა ან განმარტება. ასეთი კომენტარების მოდელში ჩართვით საშუალება გვეძლევა დავამატოთ არტეფაქტები დამუშავების პროცესში. მოდელირების პროცესის გაგრძელებისას ის კომენტარები, რომლებიც კარგავენ აქტუალობას უნდა ჩამოვაშოროთ. გარდა იმ კომენტარებისა, რომლებიც შეიცავენ მნიშვნელოვან ინფორმაციას, რომელთა აღწერა მოდელზე ვერ ხერხდება.



ნახ.2.9.1

UML-ის საერთო მექანიზმები საშუალებას იძლევიან გავაფართოვოთ მისი შესაძლებლობები. მათ რიცხვში შედიან სტერეოტიპები, მონიშნული მნიშვნელობები და შეზღუდვები.

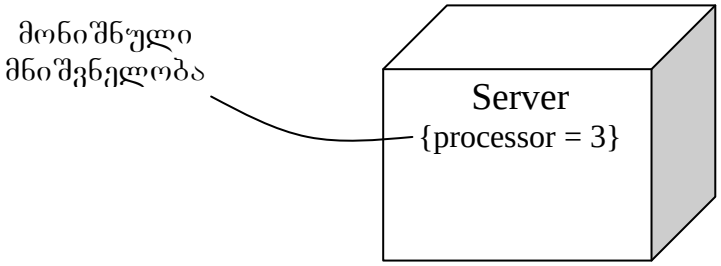
სტერეოტიპები აფართოვებენ UML-ს ლექსიკონს ახალი სამშენებლობლო ბლოკების შექმნით უკვე არსებულებიდან. სტერეოტიპი ეს არ არის შვილობილი კლასი, რასაც ჩვენ ვიყენებთ განზოგადებისას “მშობელი/შვილობილი”. იგი უფრო შეესაბამება გარკვეულ მეტატიპს, რამდენადაც სტერეოტიპი ქმნის ახალი კლასის ექვივალენტს UML-ს მოდელში. მიუწერთ რა სტერეოტიპს ისეთ ელემენტებს, როგორიც არის კვანძი ან კლასი, ჩვენ ფაქტიურად ვაფართოვებთ UML-ს. ვქმნით ახალ სამშენებლო ბლოკებს, რომლებიც გვანან არსებულს, მაგრამ აქვთ ახალი სპეციალური თვისებები, ახალი სემანტიკა. სტერეოტიპი გამოისახება დასახელებით პრჭყალებში, მოთავსებული სხვა ელემენტის თავზე. შესაძლებელია დაუნიშნოთ პიქტოგრამა, მოათავსონ მარჯვნივ დასახელებიდან ან გამოიყენონ როგორც ბაზური სიმბოლო სტერეოტიპული არსისათვის(იხ.ნახ.2.9.2.). UML-ში განსაზღვრული სტერეოტიპები მოყვანილია [1].



ნახ.2.9.2.

მონიშნული მნიშვნელობები აფართოვებენ UML-ს სამშენებლო ბლოკების თვისებებს, ელემენტთა სპეციფიკაციებში ახალი ინფორმაციის შეტანით. მონიშნული მნიშვნელობები გამოისახებიან სტრიქონის სახით, მოთავსებული სხვა ელემენტის დასახელების ქვეშ. სტრიქონი შედგება დასახელებისაგან(ნიშნული), გამყოფისაგან

(ტოლობის ნიშანი) და ამ ნიშნულის მნიშვნელობისაგან(იხ.ნახ2.9.3.). UML-ში განსაზღვრული მონიშნული მნიშვნელობები მოყვანილია [1].



ნახ.2.9.3.

შეზღუდვები აფართოვებენ UML-ს სამშენებლო ბლოკების სემანტიკას, მათი მეშვეობით შესაძლებელია შემოვიტანოთ ან შევცვალოთ არსებული წესები. აღნიშნულ მექანიზმებს იყენებენ რათა მიუსადაგოთ ენა საპრობლემო სფეროს კონკრეტულ მოთხოვნებს. შეზღუდვები გამოისახება სტრიქონის სახით ფიგურულ ფრჩხილებში. იგი შესაძლებელია დაუკავშიროთ ერთდროულად რამოდენიმე ელემენტს დამოკიდებულების მიმართებით. რეალური დროის სისტემების მოდელირებისას ხშირად არის საჭირო დროითი და სივრცითი შეზღუდვების გამოყენება.

UML-ის სამშენებლო ბლოკები – კლასები, ინტერფეისები, კოოპერაციები, კომპონენტები, კვანძები და ა.შ. საერთოა, მაგრამ მათი გამოყენება ნებისმიერი არსების მოდელირებისათვის გარკვეულ შემთხვევებში შეუძლებელია.

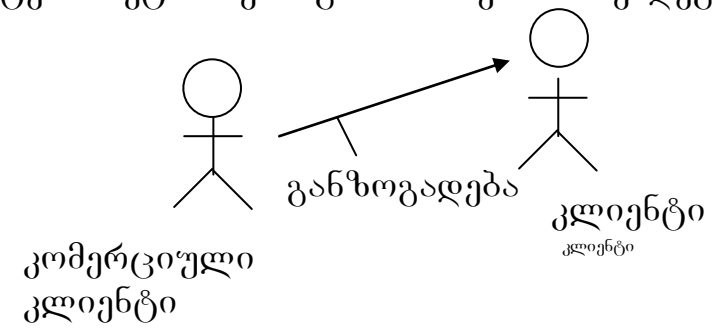
ახალი სამშენებლო ბლოკების შექმნისას(სტერეოტიპების გამოყენება), ახალი თვისებების დასამატებლად(მონიშნული მნიშვნელობის გამოყენება) და ახალი სემანტიკის დასამატებლად(შეზღუდვების გამოყენება), ცხადია პირველ რიგში უნდა დავრწმუნდეთ, რომ UML-ში არსებული საშუალებებით შეუძლებელია ჩვენი ჩანაფიქრის რეალიზება. ასევე არ არსებობს სტანდარტული სტერეოტიპი მისი გადაწყვეტისათვის.

თავი 3
ქცევითი მოდელირების საფუძვლები
3.1. პრეცენდენტები

პრეცენდენტს (**Use case**) უწოდებენ გარკვეული თანმიმდევრობის მოქმედებათა სიმრავლის აღწერას, რომელსაც ასრულებს სისტემა იმისათვის, რომ აქტიორს შეეძლოს მიიღოს განსაზღვრული შედეგი.

ყოველ პრეცედენტს უნდა გააჩნდეს სახელი, რომელიც განასხვავებს მას სხვა პრეცედენტებისაგან. გრაფიკულად პრეცედენტი გამოისახება ელიფსის სახით, რომლის შიგნით იწერება დასახელება. პრეცედენტის სახელი წარმოადგენს ტექსტურ სტრიქონს

პრეცედენტები და აქტიორები. აქტიორი წარმოადგენს დამაკავშირებელი როლების სიმრავლეს, რომელსაც პრეცენდენტების მომხმარებლები ასრულებენ მათთან ურთიერთობაში. ჩვეულებრივ აქტიორი წარმოადგენს როლს, რომელსაც თამაშობს პიროვნება, აპარატული მოწყობილობა ან სხვა სისტემა. აქტიორები გამოისახებიან შემდეგი სახით



აქტიორები პრეცედენტებს უკავშირდებიან მხოლოდ ასოციაციური კავშირით. ასოციაცია აქტიორსა და პრეცედენტს

შორის გვიჩვენებს, რომ ისინი ურთიერთობენ, შესაძლოა უგზავნიან ან ღებულობენ ერთმანეთისაგან შეტყობინებებს.

პრეცენდენდები და მოვლენათა ნაკადი. პრეცენდენტი აღწერს თუ რას აკეთებს სისტემა, მაგრამ არ განსაზღვრავს თუ როგორ აკეთებს იგი ამას. მოდელირების პროცესში ყოველთვის მნიშვნელოვანია გამოვყოთ შიდა და გარე წარმოდგენა. პრეცენდენტის აღწერაში სასურველია მიეთითოს, თუ როგორ და როდის იწყება და მთავრდება პრეცენდენტი, როდის ურთიერთქმედებს აქტიორებთან და რომელ ობიექტებთან იცვლებიან ინფორმაციით. როგორც წესი, სამუშაოს დაწყებისას მოვლენათა ნაკადებს აღწერენ ტექსტის სახით. სისტემისადმი მოთხოვნების დაზუსტების შესაბამისად გადადიან გრაფიკულ გამოსახვაზე ურთიერთქმედების დიაგრამის მეშვეობით.

პრეცენდენდები და სცენარები. ყოველ პრეცენდენტს შეესაბამება მოვლენათა სიმრავლე, რომლებიც განაპირობებენ მისი განხორციელების სხვადასხვა თანმიმდევრობას, ანუ სცენარს. შესაბამისად პრეცენდენტი აღიწერება არა ერთი, არამედ თანამიმდევრობათა სიმრავლით, რამდენადაც პრეცენდენტის ჩვენთვის საინტერესო ყველა დეტალის გამოხატვა ერთი თანმიმდევრობით შეუძლებელია. ამიტომ, სასურველია გამოვყოთ მთავარი(ძირითადი) მოვლენათა ნაკადი ალტერნატიულებისაგან. მაგალითად, პრეცენდენტს "მუშაკის დაქირავება" გააჩნია განხორციელების სხვადასხვა ვარიანტები:

- გადმოვიბიროთ სხვა კომპანიიდან;
- გადმოვიყვანოთ რომელიმე სხვა განყოფილებიდან;
- დავიქირავოთ ახალი მუშაკი.

ყოველი ვარიანტი გამოისახება თავისი თანმიმდევრობით, ანუ სცენარით. შესაბამისად თითოეული სცენარი წარმოადგენს ერთ შესაძლო ვარიანტს მოცემულ მოვლენათა ნაკადში. სცენარი – ეს

მოქმედებათა გარკვეული თანმიმდევრობაა, რომელიც წარმოადგენს სისტემის ქცევას. სცენარები ისეთივე დამოკიდებულებაში არიან პრეცენდენტებთან, როგორითაც ეგზემპლიარები კლასებთან, ე. ი. სცენარი – ეს პრეცენდენტის ეგზემპლიარია.

შედარებით რთული სისტემა შეიცავს რამოდენიმე ათეულ პრეცენდენტს, რომელთაგან თითოეული შეიძლება გაიშალოს რამოდენიმე ათეულ სცენარში. ყოველი პრეცენდენტისათვის შეიძლება გამოვყოთ ძირითადი სცენარები, რომელიც აღწერს ძირითად თანმიმდევრობას, და დამხმარე, რომლებიც აღწერენ ალტერნატიულ თანმიმდევრობებს.

პრეცედენტის აღწერა წარმოადგენს სპეციფიკაციას, რომელიც შედგება შემდეგი პუნქტებისაგან:

- დასახელება;
- მოკლე დახასიათება;
- მიზნები და შედეგები (მოქმედი პირის თვალთახედვით);
- სცენარების აღწერა (ძირითადის და ალტერნატიულის);
- სპეციალური მოთხოვნები (შეზღუდვები დროში ან სხვა რესურსებში);
- გაფართოება (კერძო შემთხვევები);
- კავშირები სხვა პრეცედენტებთან;
- მოღვაწეობის დიაგრამები (სცენარების თვალსაჩინო აღწერისათვის – აუცილებლობის შემთხვევაში).

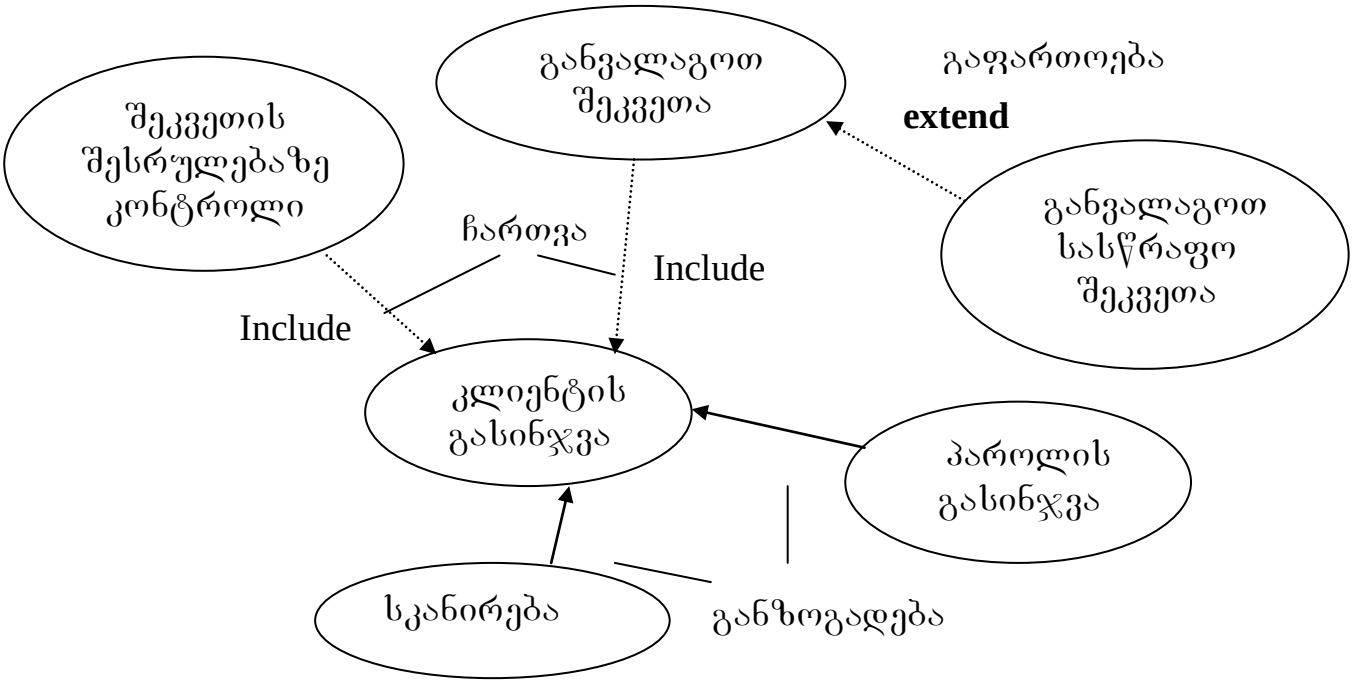
პრეცენდენტების ორგანიზება. პრეცენდენტების ორგანიზებისათვის მათ აერთიანებენ პაკეტებში ან განსაზღვრავენ მათ შორის განზოგადების, ჩართვისა და გაფართოების კავშირებს.

განზოგადება ნიშნავს, რომ პრეცენდენტი შვილობილი მემკვიდრეობით იძენს თავისი მშობლის ქცევას და სემანტიკას, შეუძლია ჩაენაცვლოს მას ან შეავსოს მისი ქცევა.

ჩართვის მიმართება პრეცენდენტებს შორის ნიშნავს, რომ ბაზური პრეცენდენტის გარკვეულ წერტილში თავმოყრილია მეორე პრეცენდენტის ქცევა. ჩართვადი პრეცენდენტი არასდროს არ არსებობს ავტონომიურად, არამედ განიხილება როგორც მომცველი პრეცენდენტის ნაწილი. შეიძლება ჩაითვალოს, რომ ბაზური პრეცენდენტი ითავსებს ჩართულების თვისებებს.

გაფართოების მიმართება გულისხმობს, რომ ბაზური პრეცენდენტი მოიცავს სხვა პრეცენდენტის ქცევას. ბაზური პრეცენდენტი შეიძლება იყოს ავტონომიურად, მაგრამ გარკვეულ პირობებში მისი ქცევა შეიძლება გაფართოვდეს მეორეს ხარჯზე.

მაგალითად, საბანკო სისტემაში შესაძლებელია არსებობა პრეცედენტის ***კლიენტის გასინჯვა***, რომელიც პასუხს აგებს კლიენტის პიროვნების შემოწმებაზე. აღნიშნული პრეცედენტი შესაძლებელია აღვწეროთ ნახ.3.1.1. -ზე მოყვანილი სახით.



ნახ.3.1.1.

როგორც მოყვანილი მაგალითებიდან ჩანს განზოგადება პრეცედენტებს შორის გამოისახება ისრიანი ხაზით, ჩართვის

მიმართება გამოისახება ისრიანი წყვეტილი ხაზით სტერეოტიპით **include**, ხოლო გაფართოების მიმართება იგივე სახით სტერეოტიპით **extend**.

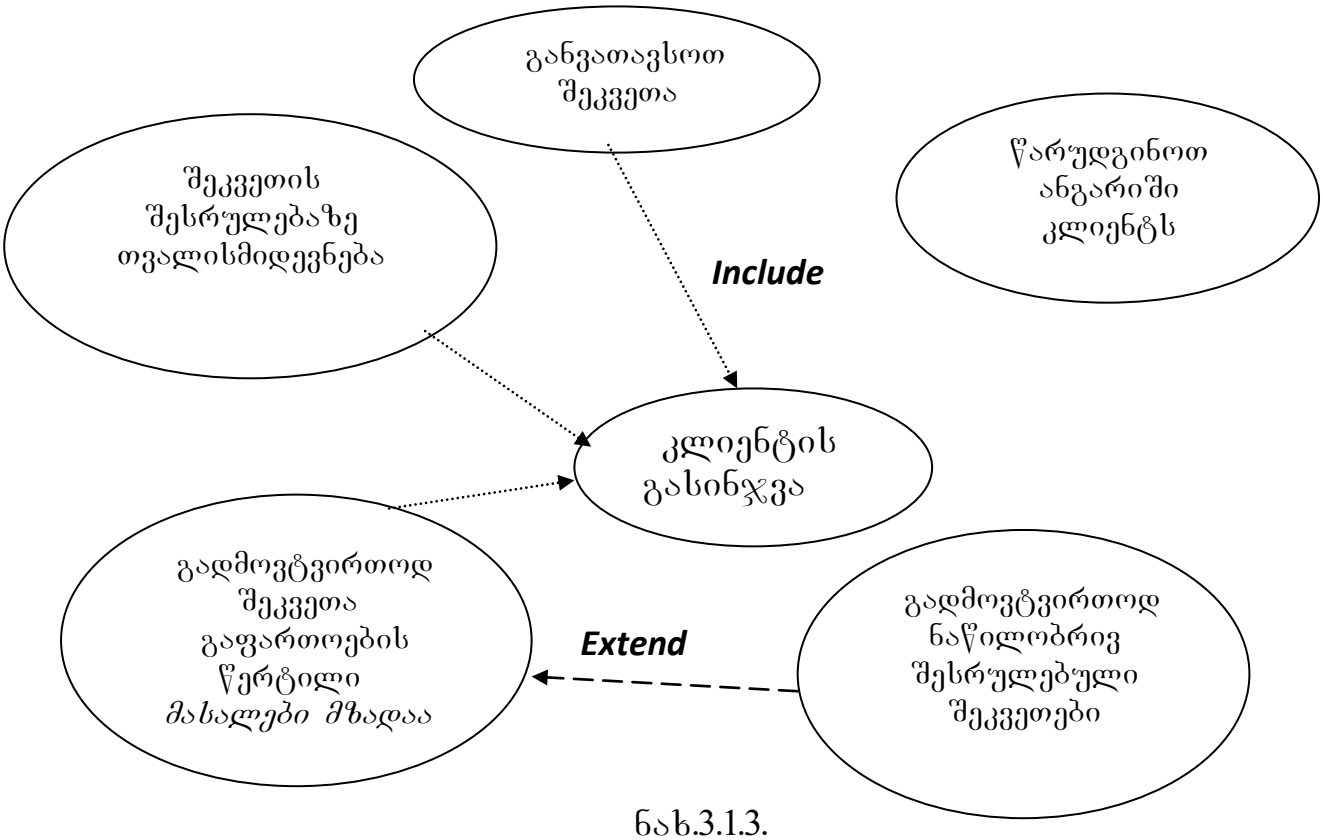
პრეცენდენდები წარმოადგენენ კლასიფიკატორებს(მექანიზმი, რომელსაც აქვს სტრუქტურული და ქცევითი თვისებები) და შეიძლება ქონდეთ ატრიბუტები და ოპერაციები. ატრიბუტები შეიძლება ჩაითვალოს ობიექტებათ პრეცენდენდების შიგნით, რომლებიც საჭიროა მისი გარე ქცევის აღწერისათვის, ხოლო ოპერაციები - სისტემის მოქმედებათ, რომლებიც საჭიროა მოვლენათა ნაკადის აღწერისათვის. ეს ობიექტები და ოპერაციები დაშვებულია ჩაირთოს ურთიერთქმედების დიაგრამებში, პრეცენდენტების ქცევის აღწერისათვის. ისევე როგორც სხვა კლასიფიკატორებს, პრეცედენტებს შესაძლებელია დაუკავშიროთ ავტომატები.

ამრიგად, პრეცენდენდები შეიძლება გამოვიყენოთ სისტემის ან კლასის მოდელირებისათვის. ელემენტის ქცევის მოდელირება ხდება შემდეგნაირად:

1. მოვახდინოთ მოცემულ ელემენტთან ურთიერთქმედებაში მყოფი აქტიორების იდენტიფიცირება.
2. მოვახდინოთ აქტიორების ორგანიზება, გამოვყოთ რა საერთო და სპეციალიზებული როლები.
3. ყოველი აქტიორისათვის განვიხილოთ ელემენტებთან მისი ურთიერთქმედების ძირითადი გზები. განვიხილოთ აგრეთვე ისეთი ურთიერთქმედებები, რომლებიც ცვლიან ელემენტის მდგომარეობებს.
4. განვიხილოთ აქტიორების ელემენტებთან ურთიერთქმედების ალტერნატიული საშუალებები.

5. მოვახდინოთ გამოვლენილი ქცევის ორგანიზება პრეცენდენტების სახით. გამოვიყენოთ რა ჩართვისა და გაფართოების მიმართებები საერთო და განსაკუთრებული ქცევის გამოყოფისათვის.

მაგალითად, საცალო ვაჭრობის სისტემა უნდა ურთიერთქმედებდეს კლიენტებთან, რომლებიც განალაგებენ შეკვეთებს და თვალი უნდა მიადევნონ მათ მოძრაობას. სისტემა გამოსცემს შესრულებულ შეკვეთებს და წარუდგენს ანგარიშებს კლიენტებს. ასეთი სისტემის მოდელირება შესაძლებელია პრეცენდენტების გამოცხადებით, რომელიც მოყვანილია ნახ.3.1.3. –ზე.



ნახ.3.1.3.

მოყვანილ მაგალითებში შესაძლებელია გამოვყოთ საერთო ქცევა (***კლიენტის გასინჯვა***) და ვარიაციები (***გადმოვტვირთოდ ნაწილობრივ შესრულებული შეკვეთები***). თითოეული მათგანისათვის უნდა ჩავრთოდ ქცევის სპეციფიკაცია ტექსტის სახით, ავტომატით ან ურთიერთქმედებით.

3.1.1. პრეცენდენტების დიაგრამა

პრეცენდენტების დიაგრამები გამოიყენებიან სისტემის მოდელირებისათვის პრეცენდენტების თვალთახედვით.

პრეცენდენტების ან გამოყენებით შემთხვევათა დიაგრამა(**Use case**) უწოდებენ დიაგრამას, რომელზედაც ნაჩვენებია პრეცენდენტებისა და აქტიორების ერთობლიობა და აგრეთვე მიმართებები მათ შორის.

პრეცენდენტების დიაგრამა ჩვეულებრივ მოიცავს თავისში:

- პრეცენდენტებს;
- აქტიორებს;
- დამოკიდებულების, განზოგადების და ასოციაციის მიმართებებს.

პრეცენდენტების დიაგრამები გამოიყენება სისტემის სტატიკური სახის მოდელირებისათვის პრეცენდენტების თვალთახედვიდან. ამისათვის გამოყენებით შემთხვევათა დიაგრამები ძირითადათ ორი საშუალებით გამოიყენებიან:

- ***სისტემის კონტექსის მოდელირებისათვის***, რომელიც გულისხმობს რომ ჩვენ სისტემას შემოვავლებთ წარმოსახვით ხაზს და გამოვავლენთ აქტიორებს, რომლებიც იმყოფებიან ამ ხაზის იქით და ურთიერთქმედებენ სისტემასთან. პრეცენდენტების დიაგრამა ამ ეტაპზე საჭიროა აქტიორებისა და მათი როლების სემანტიკის იდენტიფიცირებისათვის.
- ***სისტემისადმი მოთხოვნილებების მოდელირებისათვის.*** სისტემისადმი მოთხოვნილებების მოდელირება მიუთითებს, თუ რას უნდა აკეთებდეს სისტემა გარე მეთვალყურის თვალთაზრისით, იმისგან დამოუკიდებლად თუ როგორ უნდა აკეთებდეს იგი ამას. პრეცენდენტების დიაგრამა აქ საჭიროა სისტემის სასურველი ქცევის სპეციფიცირებისათვის. ისინი საშუალებას იძლევიან განვიხილოთ სისტემა როგორც "შავი ყუთი'. თქვენ ხედავთ ყველაფერს მის გარეთ, აკვირდებით მის

რეაქციას მოვლენაზე, მაგრამ არაფერი იცით მის შინაგან მოწყობაზე.

სისტემის კონტექსი. ნებისმიერი სისტემა თავის შიგნით შეიცავს გარკვეულ არსებს, მაშინ როდესაც სხვა არსები რჩებიან მის გარეთ. არსები სისტემის შიგნით პასუხს აგებენ ქცევის რეალიზებაზე, რომელსაც ელოდებიან არსები გარედან. არსები, რომლებიც იმყოფებიან სისტემის გარეთ და ურთიერთქმედებენ მასთან, შეადგენენ მის კონტექს. მაშასადამე კონტექს უწოდებენ სისტემის გარემოცვას.

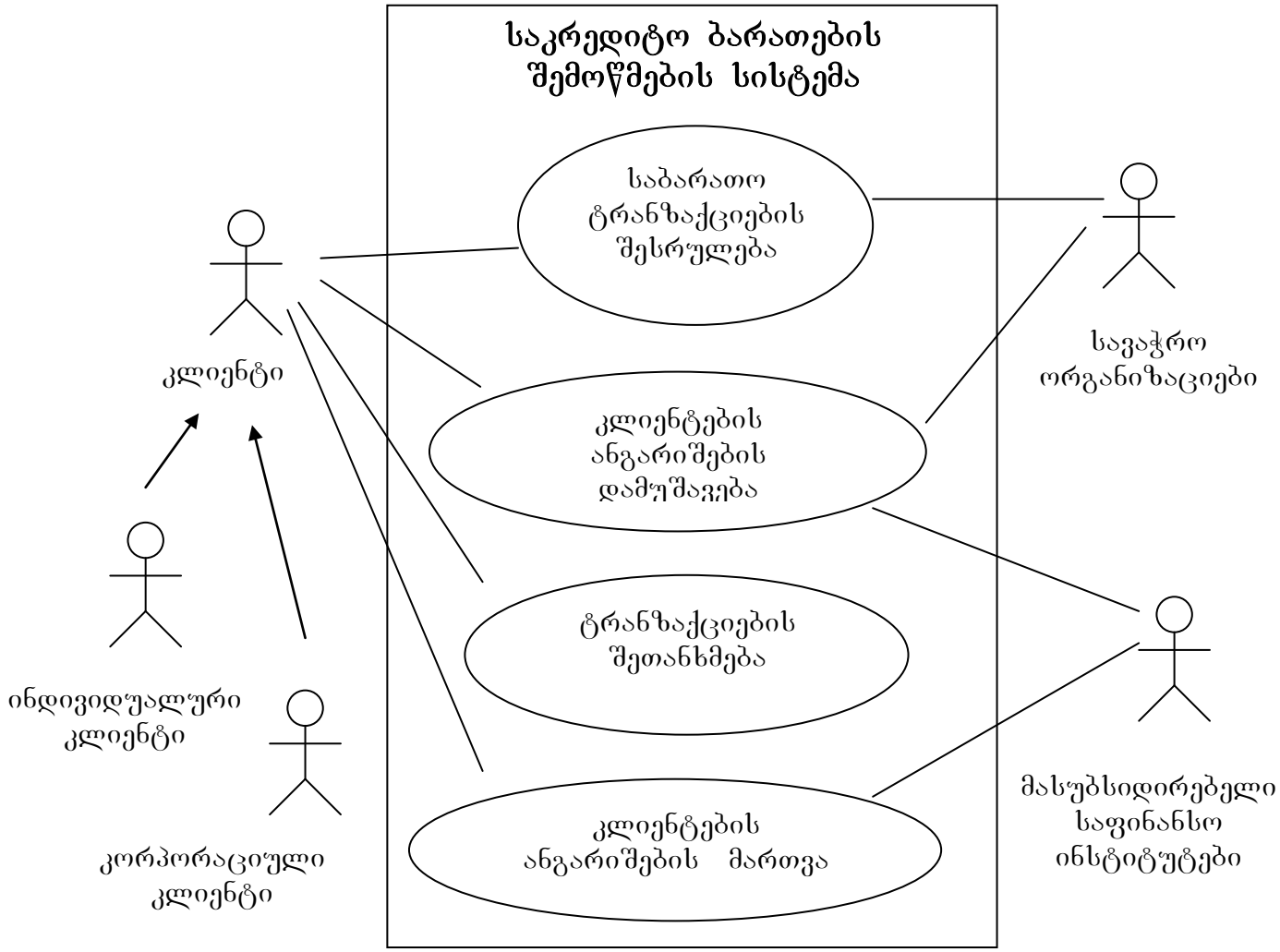
UML საშუალებას იძლევა მოვახდინოთ კონტექსის მოდელირება პრეცენდენტების დიაგრამის მეშვეობით, რომელშიც ყურადღება მიმართულია სისტემის გარემომცველ აქტიორებზე.. მნიშვნელოვანია სწორად განვსაზღვროთ აქტიორები, რამდენადაც ეს საშუალებას იძლევა ავღწეროთ არსებების კლასები, რომლებიც ურთიერთქმედებენ სისტემასთან.

სისტემის კონტექსის მოდელირება შედგება შემდეგი ბიჯებისაგან:

1. მოვახდინოთ სისტემის გარემომცველი აქტიორების იდენტიფიცირება. ამისათვის უნდა დადგინდეს ის ჯგუფები, რომლებისთვისაც სისტემის მონაწილეობაა საჭირო თავიანთი ამოცანების გადასაწყვეტად.
2. მოვახდინოთ მსგავსი აქტიორების ორგანიზება განზოგადებისა და სპეციალიზაციის მიმართებების გამოყენებით.
3. შემოვიტანოთ სტერეოტიპი ყოველი აქტიორისათვის თუ ეს გაადვილებს გაგებას.
4. მოათავსეთ აქტიორები პრეცენდენტების დიაგრამაზე და განსაზღვრეთ სისტემის პრეცენდენტებთან მათი კავშირის საშუალებები.

მაგალითად, ნახ.3.1.5. -ზე მოყვანილია სისტემის კონტექსი, რომელიც მუშაობს საკრედიტო ბარათებთან, სადაც ძირითადი

ყურადღება ექცევა მის გარემომცველ აქტიორებს. პირველ რიგში ეს კლიენტებია ორი სახის (ინდივიდუალური კლიენტი და კორპორაციული კლიენტი), რომლებიც შეესაბამებიან როლებს, რომლებსაც თამაშობენ ადამიანები სისტემასთან ურთიერთქმედებისას. ამ კონტექსში ნაჩვენებია ის აქტიორები, რომლებიც წარმოადგენენ სხვა ორგანიზაციებს, ისეთი როგორიც არის *სავაჭრო საწარმოები* (მათთან მყიდველები აწარმოებენ საბარათო ტრანზაქციებს, იძენს რა საგნებს ან მომსახურებას) და *მასუბსიდირებელი ფინანსური ინსტიტუტები* (ასრულებენ საკლირინგო პალატის როლს საბარათო ანგარიშებისათვის). რეალურ სამყაროში ბოლო ორი აქტიორი თვითონ იქნებიან პროგრამული სისტემები.



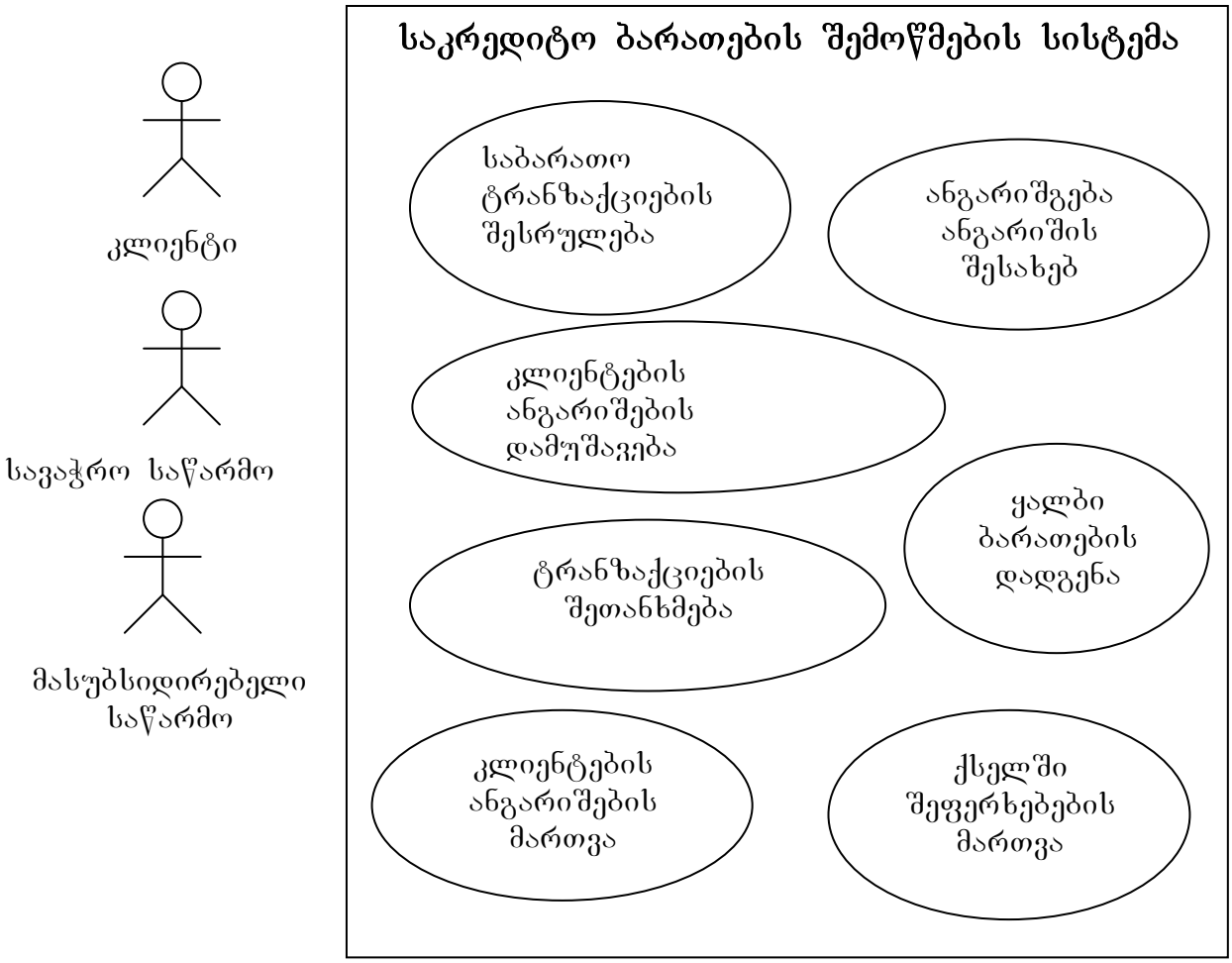
ნახ.3.1.5.

სისტემისადმი მოთხოვნები. მოთხოვნა(**Requirement**) – ეს პროექტის თავისებურება, თვისება ან სისტემის ქცევაა. მოთხოვნების დადგენისას ჩვენ გარკვეულ წილად ავღწერთ კონტრაქტების პირობებს, რომელიც დაიდება სისტემასა და მის გარეთ მყოფ არსებებს შორის, რომელშიც დეკლარირდება თუ რა უნდა აკეთოს სისტემამ. ამასთან ჩვენ არ გვაინტერესებს თუ როგორ შეასრულებს სისტემა დასახულ ამოცანას, არამედ ის თუ რას გააკეთებს იგი. კარგათ დაპროექტებულმა სისტემამ მთლიანად უნდა შეასრულოს ყველა მოთხოვნები, ამასთან უნდა აკეთოს ეს საიმედოთ.

მოთხოვნები შეიძლება გამოვსახოთ სხვადასხვანაირად, არასტრუქტურირებული ტექსტიდან დაწყებული, დამთავრებული ფორმალურ ენაზე წარმოდგენით. ფუნქციონალური მოთხოვნების უმეტესობა შეიძლება გამოვსახოთ გამოყენების პრეცენდენტების სახით, რაშიც გვეხმარება პრეცენდენტების დიაგრამები.

სისტემის მოთხოვნების მოდელირება ხდება შემდეგი სახით:

1. დავადგინოთ სისტემის კონტექსი, მოვახდენთ რა გარემომცველი აქტიორების იდენტიფიცირებას.
2. ყოველი აქტიორისათვის განვიხილოთ ქცევა, რომელსაც ის ელოდება ან მოითხოვს სისტემისაგან.
3. დავასახელოთ ეს საერთო ვარიანტები როგორც პრეცედენტები.
4. გამოვყოთ საერთო ქცევა ახალ პრეცედენტებში, რომელიც გამოიყენება სხვების მიერ; გამოვყოთ ქცევის ვარიაციები ახალ პრეცენდენტებში, რომლებიც აფართოვებენ მოვლენათა ძირითად ნაკადს.
5. მოვახდინოთ პრეცენდენტების დიაგრამაზე ამ პრეცენდენტების, აქტიორებისა და მათ შორის მიმართებების მოდელირება.
6. დავამატოდ პრეცენდენტებს შენიშვნები, რომლებიც აღწერენ არაფუნქციონალურ მოთხოვნებს.



ნახ.3.16.

ნახ.3.1.6. –ზე მოყვანილი ნახაზი აფართოვებს წინამორბედს. მართალია კავშირები აქტიორებსა და პრეცედენტებს შორის მასზე არ არის გამოსახული, მაგრამ გამოსახულია დამატებით პრეცედენტები, რომლებიც აღწერენ სისტემის ქცევის მნიშვნელოვან ელემენტებს. ეს დიაგრამა იმით არის მოხერხებული, რომ საშუალებას იძლევა მომხმარებლებმა და დამმუშავებლებმა ერთობლივად ჩამოაყალიბონ სისტემისადმი ფუნქციონალური მოთხოვნები. მაგალითად, პრეცედენტი ***ყალბი ბარათების დადგენა*** აღწერს ქცევას, რომელც მნიშვნელოვანია როგორც ***სავაჭრო ორგანიზაციებისათვის***, ისე ***მასუბსიდირებელი ფინანსური ინსტიტუტებისათვის***. მეორე პრეცედენტი – ***ანგარიშგება ანგარიშის შესახებ*** – ასევე აღწერს ქცევას, რომელიც მოითხოვება სისტემისაგან სხვა ორგანიზაციების მიერ თავიანთ კონტექსში.

მოთხოვნები, რომლებიც მოდელირდება პრეცედენტით *ქსელში შეფერხებების მართვა* განსხვავდება დანარჩენებისაგან, რამდენადაც წარმოადგენს სისტემის დამხმარე ქცევას, რომელიც აუცილებელია მისი საიმედო და უწყვეტი ფუნქციონირებისათვის.

3.2. ურთიერთქმედება

რთულ სისტემებში ობიექტები არ რჩებიან სტატიკურნი: ისინი ურთიერთქმედებენ ერთმანეთთან, ცვლიან რა შეტყობინებებს ერთმანეთს შორის.

ურთიერთქმედებას (**Interaction**) უწოდებენ ქცევას, რომელიც გამოიხატება მოცემულ კონტექსში მოცემული ერთობლიობის ობიექტებს შორის შეტყობინებების გაცვლაში, რის შედეგადაც მიიღწევა განსაზღვრული მიზანი. ურთიერთქმედებას ადგილი აქვს ყოველთვის, როდესაც ობიექტები დაკავშირებულნი არიან ერთმანეთთან (ეს სტრუქტურული კავშირები გამოისახებიან ობიექტების დიაგრამაზე).

შეტყობინება (**Message**) – ობიექტებს შორის მონაცემთა გაცვლის სპეციფიკაციაა, რომლის დროსაც გადაიცემა გარკვეული ინფორმაცია იმის გათვალისწინებით, რომ პასუხად განხორციელდება გარკვეული მოქმედება. ყველაზე ხშირად შეტყობინება დაიყვანება ოპერაციის გამოძახებამდე ან სიგნალის გაგზავნამდე, ამავე დროს მას შეუძლია შექმნას ან მოსპოს სხვა ობიექტები.

ურთიერთქმედების საშუალებით ახდენენ მართვის ნაკადების მოდელირებას ოპერაციების, კლასების, კომპონენტების, პრეცენდენტების ან მთლიანად სისტემის შიგნით. ოპერაციის კონტექსში იგი ვლინდება ოპერაციის პარამეტრების ურთიერთქმედებაში მის მიერ რეალიზებადი ალგორითმის

შესრულებისას. კლასების კონტექსში ურთიერთქმედებით შესაძლებელია კლასის სემანტიკის ვიზუალირება, მაგალითად უჩვენოთ, თუ როგორ ურთიერთქმედებენ კლასის ატრიბუტები ერთმანეთში, სხვა ობიექტებთან და კლასში განსაზღვრულ ოპერაციის პარამეტრებთან. პრეცედენტების კონტექსში ურთიერთქმედებით აღიწერება სცენარი, რომელიც თავის მხრივ წარმოადგენს პრეცედენტის მოქმედების ერთ ერთ ნაკადს.

ურთიერთქმედება საშუალებას გვაძლევს მოვახდინოთ ასეთი ნაკადების ანალიზი ორი კრიტერიუმით:

- ყურადღება გავამახვილოთ შეტყობინებების დროის მიხედვით მიმდევრობაზე;
- ყურადღების აქცენტირება ხდება ურთიერთდაკავშირებული ობიექტების სტრუქტურულ მიმართებებზე და შემდეგ განიხილება თუ როგორ გადასცემენ შეტყობინებებს ამ სტრუქტურის კონტექსში.

ურთიერთქმედებაში მონაწილე ობიექტები შეიძლება იყვნენ კონკრეტული არსებები ან პროტოტიპები. კონკრეტული არსების სახით ობიექტი წარმოადგენს რაიმეს, რეალურ სამყაროში არსებულს. მაგალითად, პ, კლასი ***ადამიანის*** ეგზემპლიარი, შესაძლებელია აღნიშნავდეს კონკრეტულ ადამიანს. პირიქით, პროტოტიპმა პ შესაძლებელია წარმოადგინოს კლასი ***ადამიანის*** ნებისმიერი ეგზემპლიარი.

ურთიერთქმედებას ადგილი აქვს ყოველთვის, როდესაც ობიექტები დაკავშირებულია ერთმანეთთან. კავშირი (**LINK**) წარმოადგენს სემანტიკურ შეერთებას ობიექტებს შორის. თუ ობიექტებს შორის არსებობს კავშირი, მაშინ ერთ ერთს შეუძლია გაუგზავნოს შეტყობინება მეორეს. ხშირად საკმარისია მიუთითოდ, რომ ასეთი გზა არსებობს. იმ შემთხვევაში თუ თქვენ გინდათ მისი

უფრო დაწვრილებითი სპეციფიცირება, შესაძლებელია კავშირის შევსება შემდეგი სტანდარტული სტერეოტიპებით:

association – უჩვენებს, რომ შესაბამისი ობიექტი ხილვადია ასოციაციისათვის;

self – შესაბამისი ობიექტი ხილვადია, რადგან წარმოადგენს დისპეჩერს ოპერაციისათვის;

global – შესაბამისი ობიექტი ხილვადია, რამდენადაც იმყოფება მოქმედებათა მომცავ სფეროში;

local - შესაბამისი ობიექტი ხილვადია, რამდენადაც იმყოფება მოქმედებათა ლოკალურ სფეროში;

parameter - შესაბამისი ობიექტი ხილვადია, რამდენადაც წარმოადგენს პარამეტრს.

მოქმედება, რომელიც წარმოადგენს შეტყობინების მიღების შედეგს შესრულებადი წინადადებაა, რომელიც ქმნის გამოთვლითი პროცედურის აბსტრაქციას. მოქმედებას შეუძლია მიგვიყვანოს მდგომარეობის შეცვლამდე.

UML საშუალებას გვაძლევს მოვახდინოთ რამოდენიმე სახის მოქმედების მოდელირება:

call(გამოძახება) – იძახებს ოპერაციას, რომელიც გამოიყენება ობიექტზე. ობიექტს შეუძლია გაუგზავნოს შეტყობინება თავისთავს, რაც მიგვიყვანს ოპერაციის ლოკალურ გამოძახებამდე.

return(დავაბრუნოთ) – აბრუნებს მნიშვნელობას გამომძახებელ ობიექტთან.

send(გავგზავნოთ) – აგზავნის სიგნალს ობიექტთან.

create(შეიქმნას) – ქმნის ახალ ობიექტს.

destroy(განადგურდეს) – ობიექტს სპობს. ობიექტს შეუძლია გაანადგუროს თავისი თავი.

როდესაც ობიექტი იძახებს ოპერაციას ან უგზავნის სიგნალს მეორე ობიექტს, შეტყობინებასთან ერთად შეიძლება გადავცეთ მისი

ფაქტიური პარამეტრები. ასევე, როდესაც ობიექტი უბრუნებს მართვას მეორე ობიექტს, შესაძლებელია მიუთითოთ დასაბრუნებელი მნიშვნელობა.

როდესაც ობიექტი უგზავნის შეტყობინებას მეორე ობიექტს, მიმღებს შეუძლია თავის მხრივ გაუგზავნოს შეტყობინება მესამე ობიექტს, ამ უკანასკნელმა მეოთხეს და ა.შ. შეტყობინებათა ასეთი ნაკადი გვაძლევს მიმდევრობას(**Sequence**).

ურთიერთქმედებაში მონაწილე ობიექტები არსებობენ მთელი ურთიერთქმედების მანძილზე. მაგრამ ზოგჯერ ობიექტები საჭიროა შეიქმნას (**create**) და განადგურდნენ (**destroy**). ეს ეხება კავშირებსაც: მიმართებები ობიექტებს შორის შეიძლება აღიძვრას ან გაქრეს. იმისათვის, რომ ავღნიშნოთ ობიექტების ან კავშირების ურთიერთქმედების პროცესში წარმოშობისა და გაქრობის ფაქტი, ელემენტს უერთებენ ერთერთ შემდეგ შეზღუდვას:

- **new**(ახალი) – გვიჩვენებს, რომ ეგზემპლიარი ან კავშირი წარმოიქმნება მომცავი ურთიერთქმედების შესრულების დროს;
- **destroyed**(მოსპობილი) – ეგზემპლიარი ან კავშირი ისპობა მომცავი ურთიერთქმედების შესრულების დასრულებამდე;
- **transient**(დროებითი) – ეგზემპლიარი ან კავშირი იქმნება მომცავი ურთიერთქმედების შესრულებისას და ისპობა მის დამთავრებამდე.

ურთიერთქმედებისას ობიექტების ატრიბუტების მნიშვნელობები, მათი მდგომარეობა ან როლი, როგორც წესი იცვლება. იგი შეიძლება გამოვხატოთ ობიექტის კოპიის შექმნით ატრიბუტების სხვა მნიშვნელობით, მდგომარეობით და როლით. ურთიერთქმედების დიაგრამაზე მათ აკავშირებენ შეტყობინების სტერეოტიპით **become**.

ყველაზე ხშირათ ურთიერთქმედებას იყენებენ მართვის ნაკადის მოდელირებისათვის, რომელიც ახასიათებს სისტემის ქცევას მთლიანობაში პრეცენდენტის, ერთი კლასის ან ცალკეული

ოპერაციის ჩათვლით. ამასთან კლასები, ინტერფეისები, კომპონენტები და მათ შორის კავშირები ახდენენ სისტემის სტატიკური ასპექტების მოდელირებას.

ახდენთ რა ურთიერთქმედების მოდელირებას, თქვენ ფაქტიურად აღწერთ მოქმედებათა თანმიმდევრობას, რომელსაც ასრულებენ ობიექტები.

მართვის ნაკადის მოდელირება შედგება შემდეგი ბიჯებისაგან:

1. განსაზღვრეთ ურთიერთქმედების კონტექსი, ეს იქნება სისტემა, ერთეული კლასი ან ცალკეული ოპერაცია;
2. აღწერეთ სცენა, რომელზედაც განხორციელდება ურთიერთქმედება. ამისათვის მოახდინეთ ობიექტების იდენტიფიცირება, რომლებიც გარკვეულ როლს ასრულებენ და დაადგინეთ მათი საწყისი თვისებები, მათ შორის ატრიბუტების მნიშვნელობა, მდგომარეობა და როლი.
3. თუ თქვენს მოდელში ყურადღება ენიჭება ობიექტების სტრუქტურულ ორგანიზაციას, მოახდინეთ მათი კავშირების იდენტიფიცირება, რომლებსაც აქვთ მიმართება მონაცემთა ცვლასთან ურთიერთქმედების დროს.
4. თუ ძირითადი ყურადღება ენიჭება დროით მოწესრიგებას, მოახდინეთ შეტყობინებების სპეციფიცირება, რომლებიც გადაიცემა ობიექტებს შორის. გამოყავით შეტყობინებები, ჩართეთ აღწერაში პარამეტრები და დასაბრუნებელი მნიშვნელობები.
5. ურთიერთქმედების არსებითი დეტალების გადაცემისათვის შესაძლებელია მიუთითოთ ყოველი ობიექტის მდგომარეობა და როლი დროის ნებისმიერ მომენტში.

მოყვანილი პოსტულატების რეალიზებისათვის გამოიყენება ურთიერთქმედების დიაგრამები.

3.2.1. ურთიერთქმედების დიაგრამა

ურთიერთქმედების დიაგრამა განსაზღვრულია ობიექტებს შორის კავშირების აღწერისათვის და გამოიყენება სისტემის დინამიკური ასპექტების მოდელირებისათვის. იგი შედგება მიმდევრობის და კოოპერაციის დიაგრამებისაგან. ამასთან მიმდევრობის დიაგრამა ყურადღებას ამახვილებს შეტყობინებების დროის მიხედვით მოწესრიგებაზე, ხოლო კოოპერაციის დიაგრამა-შეტყობინების გამგზავნი და მიმღები ობიექტების სტრუქტურულ ორგანიზაციაზე.

როგორც წესი, ურთიერთქმედების დიაგრამები შეიცავენ:

- ობიექტებს;
- კავშირებს;
- შეტყობინებებს.

მიმდევრობის დიაგრამა. მიმდევრობის დიაგრამის ასაგებათ საჭიროა განვალაგოთ ობიექტები, რომლებიც მონაწილეობენ ურთიერთქმედებაში X ღერძის ზედა ნაწილის გასწვრივ. ჩვეულებრივ ურთიერთქმედების ინიციატორი ობიექტი თავსდება მარცხნივ, ხოლო დანარჩენები მარჯვნივ (რაც უფრო შორს არის მით უფრო დამოკიდებული ობიექტია). შემდეგ ღერძის გასწვრივ განალაგებენ შეტყობინებებს, რომლებსაც ობიექტები აგზავნიან და ღებულობენ. ამასთან რაც უფრო გვიანია მით უფრო ქვევით არის. გარდა ამისა დიაგრამაზე უჩვენებენ ობიექტის სიცოცხლის ხაზს. ეს არის ვერტიკალური წყვეტილი ხაზი, რომელიც ასახავს ობიექტის არსებობას დროში. ობიექტების უმრავლესობა, რომლებიც წარმოდგენილი არიან ურთიერთქმედების დიგრამაზე, არსებობენ მათი ურთიერთქმედების განმავლობაში, ამიტომ მათ გამოხატავენ დიაგრამის ზედა ნაწილში, ხოლო მისი სიცოცხლის ციკლი იხაზება ზევიდნ ქვევით. ობიექტები შეიძლება იქმნებოდნენ

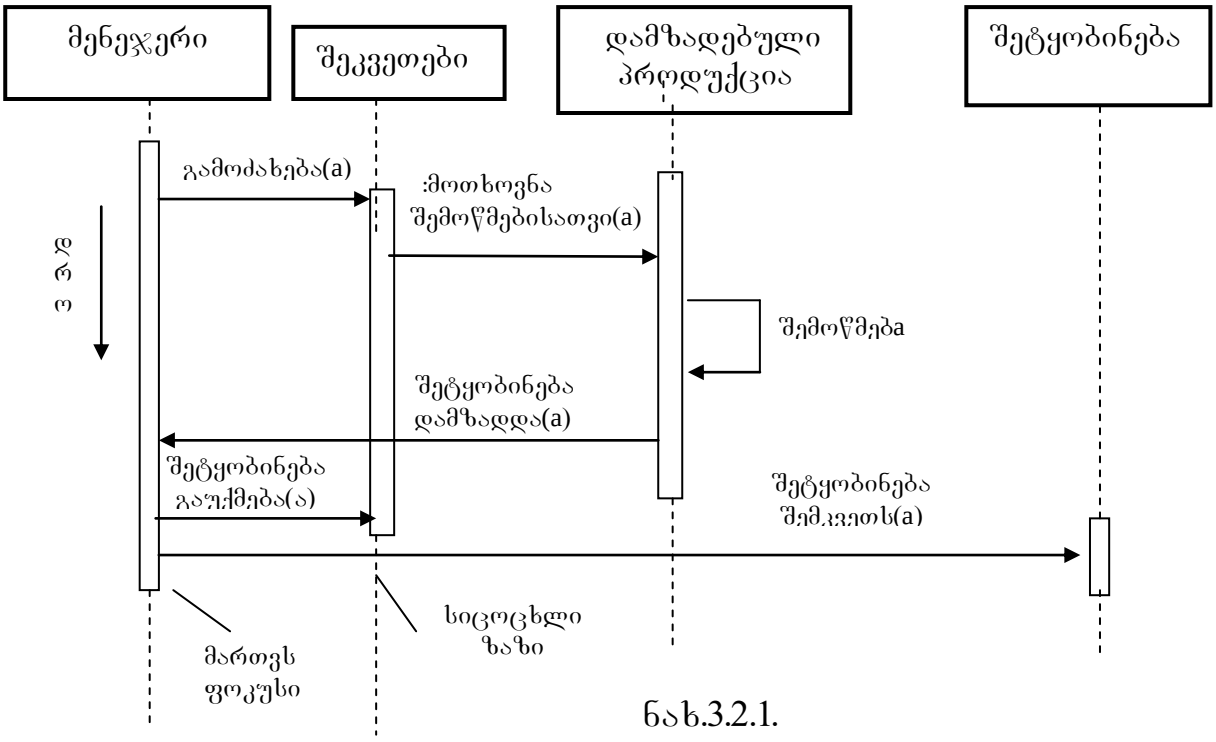
ურთიერთქმედების პერიოდშიც. მათი სიცოცხლის ციკლი მაშინ იწყება ცრეატე შეტყობინების მიღებით. ობიექტები ასევე შეიძლება დაიხურონ (მოისპონ) ურთიერთქმედების პროცესში, ასეთ შემთხვევაში მათი სიცოცხლის ციკლი მთავრდება destroy შეტყობინების მიღებით.

აღნიშნულ დიაგრამაზე უჩვენებენ აგრეთვე მართვის ფოკუსს. იგი გამოისახება სწორკუთხედით, რომელიც მიუთითებს დროის ინტერვალს, რომლის განმავლობაშიც ობიექტი ასრულებს გარკვეულ მოქმედებას უშუალოდ ან დამოკიდებული პროცედურით. სწორკუთხედის ზედა ხაზი გაუტოლდება დროის ღერძზე მოქმედების დაწყებას, ხოლო ქვედა - მის დამთავრებას.

შესაძლებელია მართვის ფოკუსის ჩართვა, გამოწვეული რეკურსიით (საკუთარ ოპერაციასთან მიმართვა) ან უკუგამოძახებით მეორე ობიექტის მხრიდან. ეს შეიძლება ვუჩვენოთ მართვის მეორე ფოკუსის აგებით, ოდნავ მარცხნივ თავისი მშობლისაგან (დასაშვებია ჩართვა ნებისმიერი სიღრმით). თუ მართვის ფოკუსის განლაგება საჭიროა მიეთითოს მაქსიმალური სიზუსტით, შეიძლება სწორკუთხედი დაიშტრიხოს, იმ დროის შესაბამისად, რომლის განმავლობაშიც მეთოდი ნამდვილად მუშაობს და არ გადასცემს მართვას მეორე ობიექტს.

მაგალითისათვის განვიხილოთ პრეცენდენტი “შეკვეთების დამზადებაზე თვალის მიდევნება” და აღვწეროთ სცენარი, რომელზედაც განხორციელდება ურთიერთქმედება. როგორც ავღნიშნეთ (§3.1.), ყოველ პრეცენდენტში პროცესი შესაძლებელია სხვადასხვა სცენარით განვითარდეს, რომლებიც მოვლენათა ნაკადებით არის განსაზღვრული. შესაბამისად ამისა, ერთ გამოყენებით შემთხვევას შეიძლება ქონდეს რამოდენიმე ურთიერთმოქმედების დიაგრამა, ვინაიდან მისი განხორციელების სცენარი შეიძლება შედგებოდეს ალტერნატიულ მოვლენათა

ნაკადებისაგან. ჩვენი მაგალითის შემთხვევაში პრეცენდენტის მოქმედება იწყება შეკვეთის მიღებით, რომელსაც თან ახლავს მოვლენათა ნაკადები, შეკვეთა დამზადდა ან არა. თითოეული ამ მოვლენის ფარგლებში პროცესი შესაძლებელია სხვადასხვა სცენარით განხორციელდეს. კერძოთ, მოხდეს შეკვეთის ამოღება დავალებიდან და შეტყობინების გაგზავნა შემკვეთთან ან ვადის დაწესება შეკვეთის დასამზადებლათ და შეკვეთის დამზადების პრიორიტეტის გაზრდა. თითოეული ეს სცენარი აღიწერება დამოუკიდებელი ურთიერთმოქმედების დიაგრამით.



ნახ.3.2.1.

ნახ.3.2.1. ნაჩვენებია ძირითადი ნაკადი და მოცემულ სქემაზე არ ჩანს თუ რა ხდება იმ შემთხვევაში, როდესაც შეკვეთა დამზადებული არ არის. აღნიშნული მოვლენაც ისევე როგორც ზემოთ მითითებულები შეიძლება ჩაითვალოს მოვლენათ, რომელიც განაპირობებს დამატებით - ალტერნატიულ სცენარს პრეცენდენტის განსახორციელებლად. შესაბამისად მისთვის უნდა ავაგოთ დამატებითი დიაგრამა.

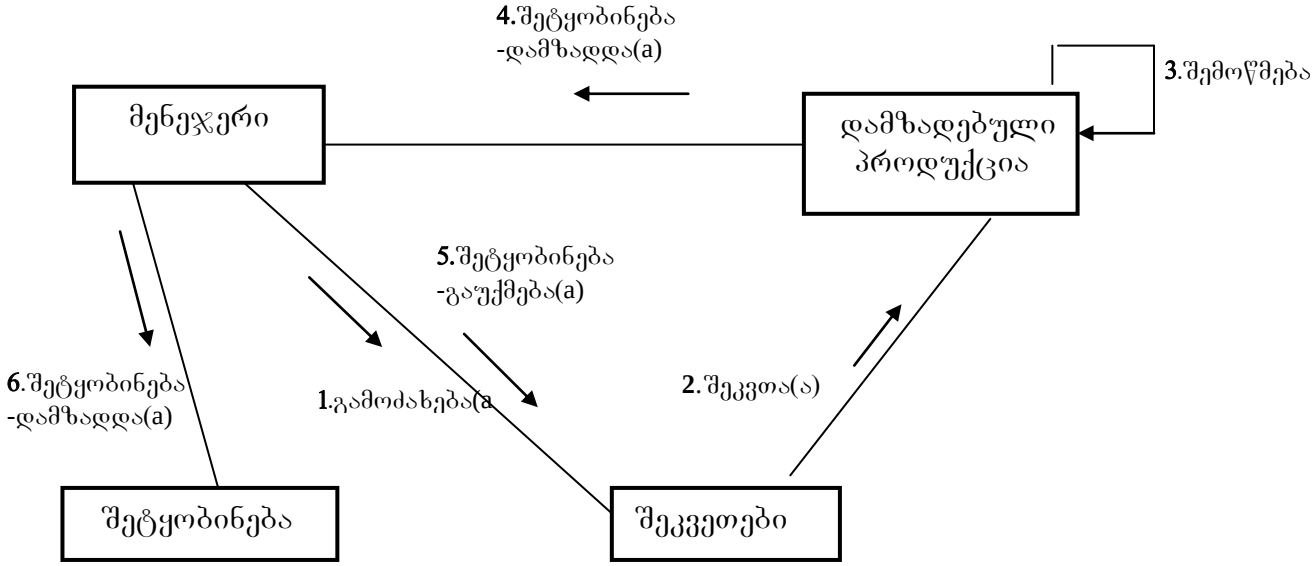
ყველა შემთხვევაში თავდაპირველად უნდა მოვახდინოთ მოცემულ პრეცედენტში მონაწილე ობიექტების იდენტიფიცირება, რომლებიც გარკვეულ როლს ასრულებენ და დავადგინოთ მათი საწყისი თვისებები, მათ შორის ატრიბუტების მნიშვნელობა, მდგომარეობა და როლი. ამის შემდეგ საშუალება გვეძლევა მოვახდინოთ მათ შორის კავშირების იდენტიფიცირება.

კოოპერაციის დიაგრამა. კოპერაციის დიაგრამა ყურადღებას ამახვილებს ურთიერთქმედებაში მონაწილე ობიექტების ორგანიზაციაზე. კოოპერაციის დიაგრამის შექმნისათვის ურთიერთქმედებაში მონაწილე ობიექტები უნდა განვალაგოთ გრაფის მწვერვალების სახით. შემდეგ კავშირები, რომლებიც ამ ობიექტებს აკავშირებენ, გამოისახებიან ამ გრაფის წიბოების სახით. კავშირებს უმატებენ შეტყობინებებს, რომლებსაც ობიექტები ღებულობენ ან აგზავნიან. ეს საშუალებას აძლევს მომხმარებელს მიიღოს ნათელი წარმოდგენა მართვის ნაკადზე.

კოოპერაციის დიაგრამებს გააჩნიათ ორი თვისება, რომლებიც განასხვავებენ მათ მიმდევრობის დიაგრამებისაგან:

- პირველი - ეს არის გზა. ერთი ობიექტის მეორესთან კავშირის აღწერისათვის. ამ კავშირის შუალედურ და ბოლო წერტილებს შეიძლება დაუკავშიროთ გზის სტერეოტიპი (მაგ. **Local**, რომელიც მიუთითებს, რომ დანიშნული ობიექტი წარმოადგენს ლოკალურს შეტყობინების გამგზავნის მიმართ).
- მეორე თვისება – შეტყობინების რიგითი ნომერია. დროის თანმიმდევრობის აღნიშვნისათვის შეტყობინებას შეიძლება დაესვას ნომერი, რომელიც თანდათან უნდა მატულობდეს ყოველი ახალი შეტყობინებისათვის. ჩართული შეტყობინების აღნიშვნისათვის გამოიყენება ათობითი ნოტაციები 1.1., 1.2. და ა.შ.

ნახ.3.2.2.-ზე მოყვანილია ზემოთ მოყვანილი მიმდევრობის დიაგრამის შესაბამისი კოოპერაციის დიაგრამა. დიაგრამაზე წარმოდგენილია ხუთი ობიექტი: ***მენეჯერი, სარჩელი, შეკვეთა, დამზადებული პროდუქცია, შეტყობინება***. მოვლენათა ნაკადი დანომრილია თანმიმდევრულად. მოქმედება იწყება იმით, რომ მენეჯერი აძლევს შეტყობინებას ***შეკვეთები*** და მიმართავს გადასცეს არსებული შეკვეთები ობიექტს ***დამზადებული პროდუქცია***. ეს უკანასკნელი ამოწმებს (პარამეტრი a), დამზადებულ შეკვეთებს ატყობინებს ***მენეჯერს***. ამის შემდეგ ობიექტი ***მენეჯერი*** უგზავნის შეტყობინებას ***შეკვეთები*** გაუქმოს - ამოიღოს დავალებიდან. ასევე ობიექტს ***შეტყობინება*** გაუგზავნოს დამკვეთს შეტყობინება შეკვეთის შესრულების შესახებ.



ნახ. 3.2.2.

მიმდევრობითი ნაკადების მოდელირების გარდა შესაძლებელია უფრო რთული ნაკადების მოდელირება, როგორიც არის იტერაცია და განშტოება. იტერაცია წარმოადგენს შეტყობინებათა განმეორებად თანმიმდევრობას. მისი მოდელირებისათვის შეტყობინების ნომრის წინ თანმიმდევრობაში ისმება იტერაციის

გამოსახულება მაგ. **i=1-n**. იტერაცია გვიჩვენებს , რომ შეტყობინება განმეორდება მოცემული გამოსახულების შესაბამისად. ანალოგიურად, პირობა წარმოადგენს შეტყობინებას, რომლის შესრულება დამოკიდებულია ბულის გამოსახულების გამოთვლის შედეგზე. პირობის მოდელირებისათვის შეტყობინების რიგითი ნომრის წინ ისმება გამოსახულება მაგ. **x>0**. ყველა ალტერნატიულ შტოებს ექნებათ ერთი და იგივე ნომერი, მაგრამ პირობა ყოველი შტოსათვის უნდა მივცეთ ისე, რომ ორი მათგანი მათ შორის არ სრულდებოდეს ერთდროულად. ისევე როგორც მიმდევრობის დიაგრამაზე, ერთ კოოპერაციის დიაგრამაზე შესაძლებელია უჩვენოთ მხოლოდ ერთი მართვის ნაკადი, ამიტომ როგორც წესი, ქმნიან ურთიერთქმედების რამოდენიმე დიაგრამას, რომელთაგანაც ერთი ითვლება ძირითადათ, ხოლო დანარჩენები აღწერენ ალტერნატიულ გზებს.

3.2.2. მოდელირების ტიპიური ხერხები

ამრიგად, სისტემის დინამიური ასპექტების მოდელირება ურთიერთქმედების დიაგრამების გამოყენებით შესაძლებელია სისტემის, ქვესისტემის, ოპერაციის ან კლასის კონტექსში. ურთიერთქმედების დიაგრამები შესაძლებელია დაუკავშიროთ აგრეთვე პრეცედენტებს (სცენარების მოდელირებისათვის) და კოოპერაციებს (ობიექტთა ერთობის დინამიური ასპექტების მოდელირებისათვის).

სისტემის დინამიური ასპექტების მოდელირებისათვის ურთიერთქმედების დიაგრამებს იყენებენ ორმაგათ:

- მართვის ნაკადების დროის მიხედვით მოწესრიგებისათვის. ამ მიზნით გამოიყენებენ მიმდევრობის დიაგრამებს, რომლებშიც აქცენტირება ხდება შეტყობინებების დროში გადაცემაზე, რაც

განსაკუთრებით სასარგებლოა პრეცენდენდების დინამიური ასპექტების მოდელირებისათის. უბრალო იტერაციები და განშტოებები მიმდევრობის დიაგრამებზე გამოისახება უფრო მოხერხებულად, ვიდრე კოოპერაციის დიაგრამაზე.

- მართვის ნაკადების სტრუქტურული ორგანიზაციის მოდელირებისათვის. ძირითადი ყურადღება ამ დროს ეთმობა ურთიერთქმოქმედ ეგზემპლიარებს შორის სტრუქტურულ მიმართებას, რომელთა გასწვრივაც გადაიცემა შეტყობინებები. რთული იტერაციები, განშტოებები და მართვის პარალელური ნაკადების ვიზუალირებისათვის კოოპერაციები უფრო მოხერხებულია ვიდრე მიმდევრობა.

მართვის ნაკადები დროში. მართვის ნაკადების დროის მიხედვით მოწესრიგება ხდება შემდეგნაირად:

1. დავადგინოთ ურთიერთქმედების კონტექსი, ეს იქნება სისტემა, ქვესისტემა, ოპერაცია ან პრეცენდენტის ერთ ერთი სცენარი.
2. განსაზღვრეთ ურთიერთმოქმედი ობიექტები და განალაგეთ მარცხნიდან მარჯვნივ, ისე რომ შედარებით მნიშვნელოვანი ობიექტები განლაგდნენ უფრო მარჯვნივ.
3. განსაზღვრეთ ყოველი ობიექტისათვის სიცოცხლის ხაზი. უფრო ხშირად ობიექტები არსებობენ მთელი ურთიერთმოქმედების განმავლ;ობაში. ხოლო იმ ობიექტებისათვის, რომლებიც ურთიერთმოქმედის პროცესში ისპობიან, სიცოცხლის ხაზზე ნათლად უჩვენეთ დაბადებისა და სიკვდილის წერტილები შესაბამისი სტერეოტიპით.
4. დაწყებული შეტყობინებიდან, რომელიც ურთიერთმოქმედების ინიციირებას ახდენს, განალაგეთ ყველა შემდეგი შეტყობინებები ზევიდან ქვევით ობიექტების სასიცოცხლო ხაზებს შორის, უჩვენეთ ყოველი შეტყობინების თვისება მაგ. მისი პარამეტრები.

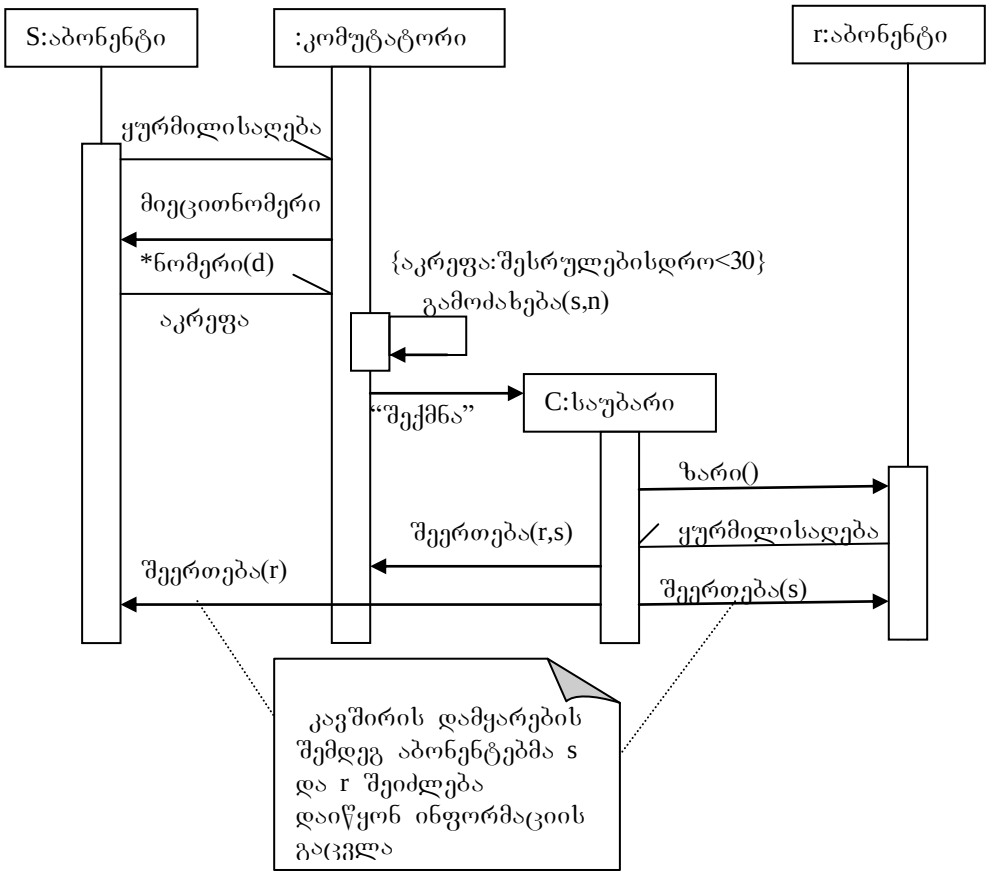
5. თუ საჭიროა შეტყობინებების ჩართვა ან გამოთვლის ზუსტი ინტერვალის მითითება, შეავსეთ ობიექტების სიცოცხლის ციკლი მართვის ფოკუსით.
6. თუ საჭიროა დროითი ან სივრცობრივი შეზღუდვების სპეციფიცირება შეავსეთ შეტყობინება დროითი აღნიშვნებით და დაუკავშირეთ შესაბამისი შეზღუდვები.
7. მართვის ნაკადების უფრო ფორმალური აღწერისათვის დაუკავშირეთ ყოველ შეტყობინებას წინა და შემდგომი პირობები.

მიმდევრობის ყოველ დიაგრამაზე შეიძლება მხოლოდ ერთი მართვის ნაკადის ჩვენება, ამიტომ როგორც წესი ქმნიან ურთიერთქმედების რამოდენიმე დიაგრამას, რომელთაგან ერთი ითვლება ძირითადათ, ხოლო დანარჩენები აღწერენ ალტერნატიულ გზებს და გამორიცხულ პირობებს. მიმდევრობის დიაგრამების ასეთი ერთობლიბა, შეიძლება გავაერთიანოთ პაკეტში, მივცეთ რა თვითეულს შესაბამისი სახელი.

მაგალითისათვის ნახ3.23.-ზე ნაჩვენებია მიმდევრობის დიაგრამა, სადაც აღწერილია მართვის ნაკადი, რომელიც ეხება უბრალო ორმხრივ სატელეფონო საუბრის ინიციირებას. აბსტრაქციის მოცემულ დონეზე გვაქვს ოთხი ობიექტი: ორი აბონენტი s და r, სატელეფონო კომუტატორი და ობიექტი c, რომელიც წარმოადგენს საუბრის მატერიალიზაციას აბონენტებს შორის. მიმდევრობა იწყება ერთერთი აბონენტის(s) მიერ კომუტატორთან სიგნალის გაგზავნით (***ყურმილისაღება***). კომუტატორი თავის მხრივ, უგზავნის აბონენტს სიგნალს ***მიუთითოს ნომერი***, რომლის შემდეგაც აბონენტი რამოდენიმეჯერ აგზავნის შეტყობინებას ***აკრიფეთ ციფრი***. აქვე უნდა ავღნიშნოთ, რომ ამ შეტყობინებას აქვს დროის შეზღუდვა (შესრულების დრო – ნაკლებია 30 წამზე). დიაგრამაზე არ არი ნაჩვენები , თუ რას აკეთებს სისტემა როდესაც პირობა არ სრულდება, ამისათვის შესაძლებელი იყო ჩაგვერთო ცალკე შტო ან

მოგვეყვანა სხვა დიაგრამა. შემდეგ კომუტატორი უგზავნის შეტყობინებას თავის თავს (გავგზავნოთ შეტყობინება) და ქმნის ობიექტს c კლასისა საუბარი, რომელსაც გადასცემს სამუშაოს დანარჩენ ნაწილს. ობიექტი c ურეკავს მეორე აბონენტს(r), რომელიც ასინქრონულად უგზავნის შეტყობინებას ***აიღოს ყურმილი***. ამის შემდეგ ობიექტი საუბარი ახსენებს კომუტატორს, რომ საჭიროა დაკავშირება, ხოლო კომუტატორი ატყობინებს ორივე ობიექტს, რომ ისინი დაკავშირებული არიან, რომლის შემდეგაც აბონენტებს შეუძლიათ დაიწყონ ინფორმაციის გაცვლა.

გარდა მოყვანილისა, c-ს გააჩნია დამატებითი მოვალეობა, დაკავშირებული საუბარზე ღირებულების აღრიცხვასთან, რომელიც გამოსახული უნდა იქნას ურთიერთქმედების სხვა დიაგრამაზე.



ნახ. 3.2.3.

მართვის ნაკადების სტრუქტურა. მართვის ნაკადების სტრუქტურული ორგანიზაციის მოდელირება შედგება შემდეგი ბიჯებისაგან:

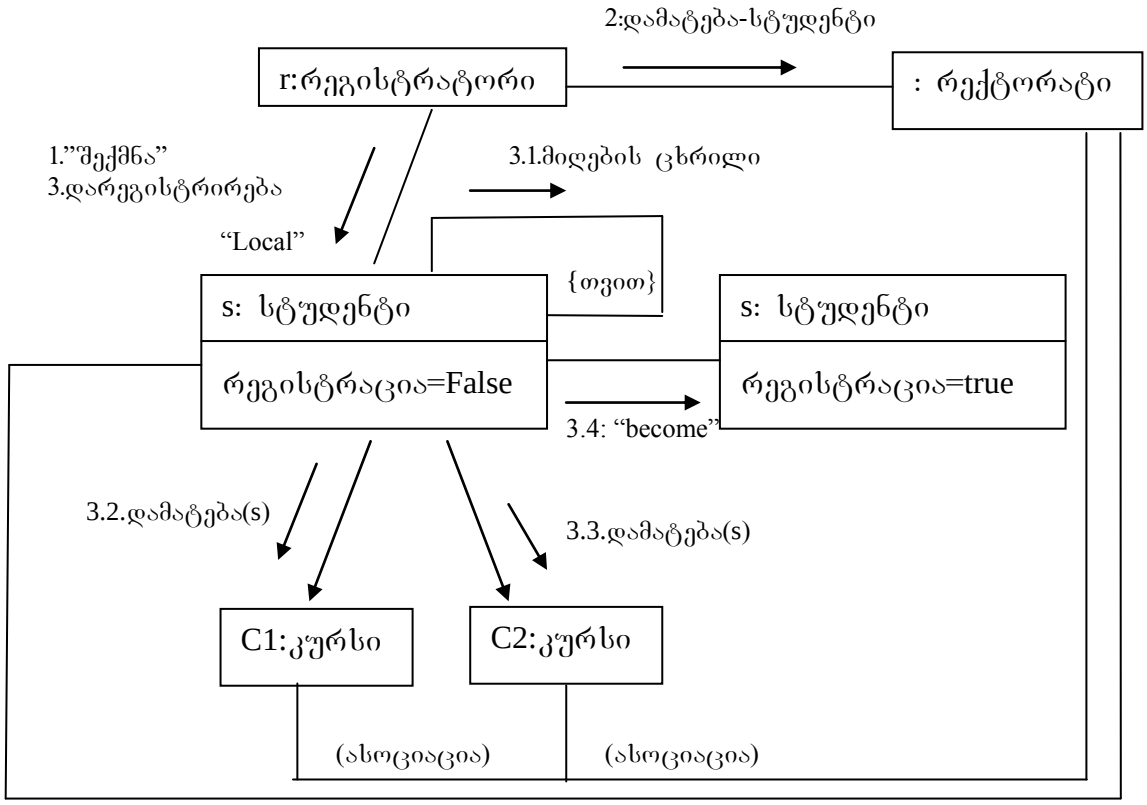
1. დავადგინოთ ურთიერთქმედების კონტექსი, ეს იქნება სისტემა, ქვესისტემა, ოპერაცია ან პრეცენდენტის ერთ ერთი სცენარი.
2. განსაზღვრეთ ურთიერთქმედების სცენა, დავადგენთ რა რომელი ობიექტები მონაწილეობენ მასში. განალაგეთ ისინი კოოპერაციის დიაგრამაზე გრაფის მწვერვალებად, ისე რომ მნიშვნელოვანი ობიექტები აღმოჩნდნენ დიაგრამის ცენტრში, ხოლო მათი მეზობლები ნაპირზე.
3. განსაზღვრეთ თითოეული ამ ობიექტის საწყისი თვისებები. თუ ატრიბუტის მნიშვნელობა, ობიექტების როლი და მდგომარეობა ურთიერთქმედებისას იცვლება, დიაგრამაზე მოათავსეთ დუბლიკატები ახალი მნიშვნელობებით და დააკავშირეთ ისინი შეტყობინების სტერეოტიპით **become** და **copy**, შესაბამისი რიგითი ნომრების დართვით.
4. დეტალურად ავღწეროთ კავშირები ობიექტებს შორის, რომელთა გასწვრივაც გადაიცემა შეტყობინებები. ამისათვის:

- თავიდან მიუთითეთ კავშირი ასოციაცია. ისინი ყველაზე მნიშვნელოვანია, რამდენადაც წარმოადგენენ სტრუქტურულ შეერთებებს.
- ამის შემდეგ მიუთითეთ დანარჩენი კავშირები, დაამატებთ რა მას გზის შესაბამის სტერეოტიპებს(როგორიც არის **global** ან **local**).

5. დაწყებული ურთიერთქმედების ინიციირების შეტყობინებიდან, დაუკავშირეთ ყველა დანარჩენ კვანძებს შეტყობინებები, მონიშნეთ რიგითი ნომრები.
6. თუ საჭიროა დროითი ან სივრცითი შეზღუდვების სპეციფიცირება დაუმატეთ შეტყობინებებს დროითი ნიშნები და დაუკავშირეთ საჭირო შეზღუდვები.

7. თუ საჭიროა მართვის ნაკადების უფრო ფორმალური აღწერა დაუკავშირეთ ყოველ შეტყობინებას წინა და შემდგომი პირობები.

ისევე როგორც მიმდევრობის დიაგრამაზე, აქაც ერთი კოოპერაციის დიაგრამით შეიძლება მხოლოდ ერთი მართვის ნაკადის ჩვენება, ამიტომ როგორც წესი ქმნიან ურთიერთქმედების რამოდენიმე დიაგრამას, რომელთაგან ერთი ითვლება ძირითადათ, ხოლო დანარჩენები აღწერენ ალტერნატიულ და განსაკუთრებულ პირობებს. კოოპერაციის დიაგრამების ასეთი ერთობლიბა, შეიძლება გავაერთიანოთ პაკეტში, მივცეთ რა თვითეულს შესაბამისი სახელი.



ნახ. 3.2.4.

მაგალითისათვის ნახ.3.2.4. – ზე მოყვანილია კოოპერაციის დიაგრამა, რომელიც აღწერს მართვის ნაკადს, დაკავშირებულს ახალი სტუდენტის რეგისტრაციასთან. დიაგრამაზე წარმოდგენილია ხუთი ობიექტი: ***რეგისტრატურა, სტუდენტი, ორი ობიექტი კურსი და რექტორატი (უმაღლესი სასწავლებელი).*** მოქმედება იწყება იმით,

რომ ***რეგისტრატურა*** ქმნის ობიექტს ***სტუდენტი*** და ამატებს მას ***უმაღლეს სასწავლებელს***(შეტყობინება ***დამატება სტუდენტი***), ხოლო შემდეგ აძლევს მას მითითებას დარეგისტრირდეს. ამის შემდეგ ობიექტი ***სტუდენტი*** უგზავნის შეტყობინებას თავისთავს ***მიღების ცხრილი***, მიღებული აქვს რა თავიდან ***კურსი,*** რომელზეც მას სურს ჩაწერა.

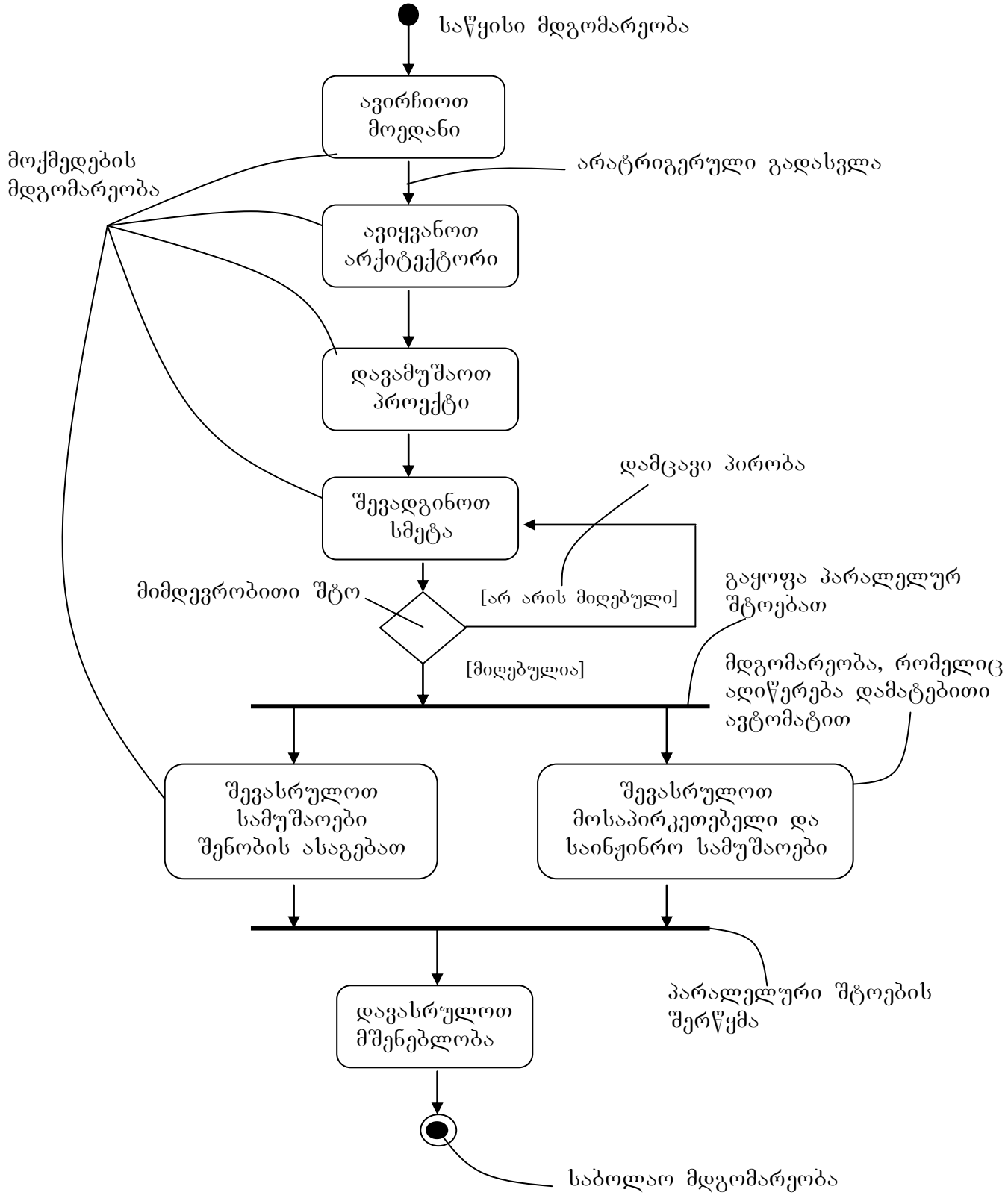
შემდეგ ობიექტი ***სტუდენტი*** ამატებს თავისთავს ***კურსის*** ყოველ ობიექტს. ბოლოს ისევ არის ნაჩვენები ობიექტი ***სტუდენტი*** ატრიბუტ ***დარეგისტრირების*** განახლებული მნიშვნელობით.

3.3. მოღვაწეობის დიაგრამა

მოღვაწეობის დიაგრამები ერთერთი იმათგანია, რომლებიც გამოიყენება სისტემის ქცევის დინამიური ასპექტების მოდელირებისათვის. როგორც წესი ისინი გამოიყენებიან გამოთვლითი პროცესების მიმდევრობითი და პარალელური ბიჯების მოდელირებისათვის. თუ ურთიერთქმედების დიაგრამაზე აქცენტირება ხდება მართვის ნაკადების გადასვლებზე ობიექტიდან ობიექტზე, მოღვაწეობის დიაგრამა აღწერს გადასვლებს ერთი მოქმედებიდან მეორეზე. ***მოღვაწეობა*** (**Activity**) - ეს გარკვეული ხანგრძლივობის შესრულბის ეტაპია ავტომატში. საბოლაოდ მოღვაწეობა დაიყვანება გარკვეულ ***მოქმედებამდე*** (**Action**), რომლებიც შედგენილია ელემენტარული გამოთვლებისაგან და მივყავართ სისტემის მდგომარეობის შეცვლამდე (იხ.ნახ.3.3.1.).

მოქმედება შეიძლება მდგომარეობდეს სხვა ოპერაციის გამოძახებაში, რაიმე სიგნალის გაგზავნაში, ობიექტის შექმნაში ან მოსპობაში ან უბრალო გამოსახულების გამოთვლაში. მოღვაწეობის დიაგრამა ზოგადათ შესდგება:

- მოღვაწეობის და მოქმედების მდგომარეობებისაგან;
- გადასვლებისაგან;
- ობიექტებისაგან.



ნახ.3.3.1.

მოქმედების მდგომარეობა და მოღვაწეობის მდგომარეობა. მოქმედების მდგომარეობა ისეთი მდგომარეობაა, რომლის შემდგომი დეკომპოზიცია შეუძლებელია. ეს ნიშნავს, რომ მათ შიგნით შეიძლება ხდებოდეს სხვადასხვა მოვლენები, მაგრამ მოქმედების მდგომარეობაში შესრულებადი სამუშაო არ შეიძლება შეწყვეტილ იქნას. ჩვეულებრივ დაშვებულია, რომ ერთი მოქმედების მდგომარეობის ხანგრძლივობა იკავებს ძალიან მცირე დროს.

მისგან განსხვავებით შეიძლება მოღვაწეობის მდგომარეობების შემდგომი დეკომპოზიცია, ამის შედეგად შესრულებადი მოღვაწეობა შეიძლება წარმოვადგინოთ სხვა მოღვაწეობის დიაგრამების სახით. მოღვაწეობის მდგომარეობა არ არის ელემენტარული, ანუ შეიძლება შეწყვეტილ იქნან. ამასთან იგულისხმება, რომ მათი დამთავრებისათვის საჭიროა საკმაო დრო. შეიძლება ჩაითვალოს, რომ მოქმედების მდგომარეობა – ეს მოღვაწეობის დიაგრამის კერძო სახეა, უფრო კონკრეტულად კი – ისეთი მდგომარეობა, რომლის შემდგომი დაშლა შეუძლებელია. რაც შეეხება მოღვაწეობის მდგომარეობას იგი წარმოდგენს შედგენილ მდგომარეობას, რომლის მართვის ნაკადი შეიცავს მხოლოდ სხვა მოღვაწეობებისა და მოქმედებების მდგომარეობებს. გრაფიკულად მოღვაწეობისა და მოქმედების მდგომარეობები გამოისახებიან ერთნაირად, მხოლოდ იმის განსხვავებით, რომ პირველს შეიძლება გააჩნდეს დამატებით მოქმედებები შესვლისას და გამოსვლისას ე. ი. მოქმედებები, რომლებიც სრულდებიან მოცემულ მდგომარეობაში შესვლისას და გამოსვლისას.

გადასვლები. ამრიგად, მოღვაწეობისა და მოქმედების მდგომარეობები – ეს ავტომატის მდგომარეობების კერძო სახეა. შევდივართ რა ერთერთ ასეთ მდგომარეობაში სრულდება შესაბამისი მოქმედება ან მოღვაწეობა, ხოლო გამოსვლისას მართვა გადაეცემა შემდეგ მოქმედებას ან მოღვაწეობას. ასეთი გადასვლების

აღწერისათვის გამოიყენება ***გადასვლები*** (**Transitions**), რომლებიც მიუთითებენ გზას ერთი მოქმედების მდგომარეობიდან მეორეში. გრაფიკულად იგი გამოისახება ისრიანი სწორი ხაზით. ასეთ გადასვლებს უწოდებენ გადასვლებს დასრულებისას ანუ არატრიგერულ გადასვლებს, რამდენადაც მართვა საწყის მდგომარეობაში მუშაობის დასრულებისას მყისიერად გადაეცემა შემდეგ.

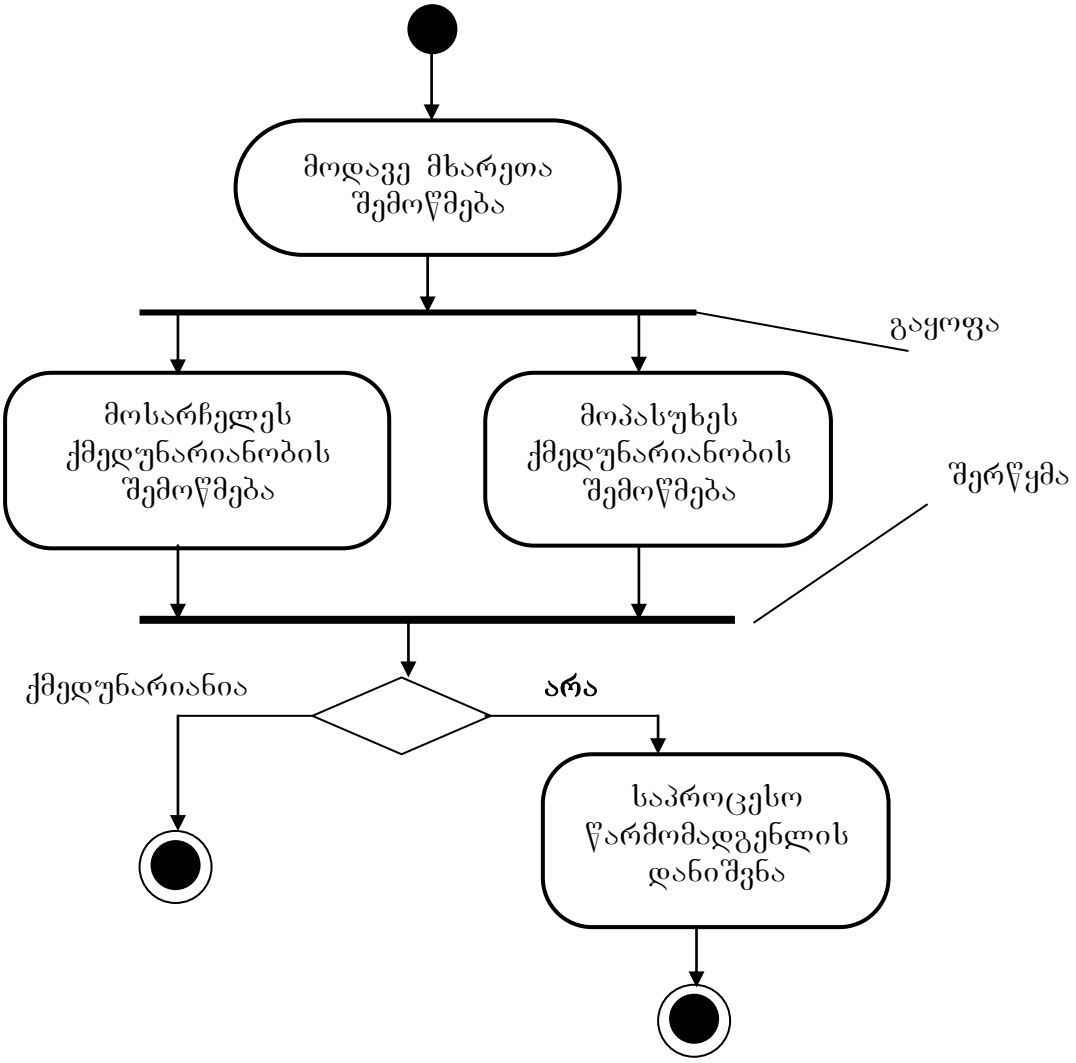
ასეთი მარტივი მიმდევრობითი გადასვლები გვხვდება ყველაზე ხშირად, მაგრამ მხოლოდ ისინი არასაკმარისია მართვის ნაკადების მოდელირებისათვის. ისევე როგორც ბლოქ – სქემებში შესაძლებელია განშტოების გამოყენება, რომელიც აღწერს შესრულების სხვადასხვა გზებს გამომდინარე ბულის გამოსახულების რომელიღაც მნიშვნელობიდან. განშტოებას აქვს ერთი შესასვლელი და ორი ან მეტი გამოსასვლელი.

მოხერხებულობისათვის შეგვიძლია ვისარგებლოთ გასაღებური სიტყვით else , იმ გადასვლის აღსანიშნავათ, რომელიც უნდა აირჩეს იმ შემთხვევაში როდესაც პირობები ყველა დანარჩენი გადასვლებისათვის არ სრულდება.

განშტოების მეშვეობით შეგვიძლია იტერაციის რეალიზებაც. ამისათვის უნდა შემოვიტანოთ მოქმედების ორი მდგომარეობა. პირველში ვადგენთ მთვლელის საწყის მნიშვნელობას, მეორეში იგი იზრდება, ხოლო განშტოებაში გამოთვლილი მნიშვნელობით დგინდება უნდა შეწყდეს იტერაცია თუ არა.

მარტივი და განშტოებადი გადასვლები მოღვაწეობის დიაგრამაზე გამოიყენება ყველაზე ხშირათ, მაგრამ გვხვდება აგრეთვე პარალელური ნაკადები. ასეთი პარალელური ნაკადების გაყოფისა და შერწყმის შესრულების აღნიშვნისათვის გამოიყენება სინქრონიზაციის ზღუდე, რომელიც იხაზება გამსხვილებული ვერტიკალური ან ჰორიზონტალური ხაზით. ყოველი პარალელურად

შესრულებადი მართვის ნაკადი არსებობს დამოუკიდებელი აქტიური ობიექტის კონტექსში, რომელიც მოდელირდება პროცესის სახით. მაგალითისათვის მოვიყვანოთ მოდავე მხარეთა ქმედუნარიანობის შემოწმების წესი სამოქალაქო სამართალწარმოებისას.



ნახ.3.3.2.

როგორც ნახაზიდან ჩანს გაყოფის წერტილი შეესაბამება ერთი მართვის ნაკადის გაყოფას ორ პარალელურზე. ამ წერტილში შეიძლება არსებობდეს ერთი შემავალი და ორი ან მეტი გამომავალი მიმდევრობა. გამომავალი მიმდევრობა წარმოადგენს მართვის ერთ დამოუკიდებელ ნაკადს. გაყოფის წერტილის შემდეგ მოქმედებები სრულდებიან პარალელურად. კონცეპტუალური თვალსაზრისით იგულისხმება ნამდვილი პარალელიზმი, მაშასადამე ერთდროული

შესრულება, მაგრამ რეალურ სისტემებში ეს შეიძლება შესრულდეს (როდესაც სისტემა განლაგებულია რამოდენიმე კვანძზე) და შეიძლება არა (იმ შემთხვევაში როდესაც სისტემა განლაგებულია მხოლოდ ერთ კვანძზე). უკანასკნელ შემთხვევაში ადგილი აქვს მიმდევრობით შესრულებას ნაკადებს შორის გადართვით, რაც იძლევა პარალელიზმის მხოლოდ ილუზიას.

მოყვანილი ნახაზიდან აგრეთვე ჩანს, რომ შერწყმის წერტილი წარმოადგენს რამოდენიმე პარალელური ნაკადის სინქრონიზაციის მექანიზმს. ამ წერტილში შედიან ორი ან მეტი მიმდევრობა და გამოდის მხოლოდ ერთი. შერწყმის წერტილში პარალელური ნაკადები სინქრონიზირდებიან ანუ ყოველი მათგანი იცდის სანამ ყველა დანარჩენი მიაღწევს ამ წერტილს რომლის შემდეგაც შესრულება გრძელდება ერთი ნაკადის ფარგლებში.

ბოლოს უნდა აღინიშნოს, რომ უნდა დაცული იქნას ბალანსი გაყოფის და შერწყმის წერტილებს შორის. ეს ნიშნავს, რომ ნაკადების რიცხვი, რომლებიც გამოდიან გაყოფის წერტილიდან, ტოლი უნდა იყოს იმ ნაკადების რაოდენობის, რომლებიც შემოდის მის შესაბამის შერწყმის წერტილში.

ბიზნეს - პროცესების მოდელირებისას ხშირად მიმართავენ დიაგრამის დაყოფას ჯგუფებად, რომელთაგანაც თითოეული წარმოადგენს ამა თუ იმ განყოფილების ან თანამდებობის პირის დასახელებას, რომელიც პასუხს აგებს მოცემულ სამუშაოთა შესრულებაზე. **UML**- ში ასეთ ჯგუფებს უწოდებენ ბილიკებს. ყოველ ბილიკს ენიჭება სახელი და გამოსახავს გარკვეულ პასუხიმგებლობას მასში მოთავსებულ სამუშაოებზე. აქტიურობის დიაგრამაზე, რომელიც დაყოფილია ბილიკებათ, ყოველი მოღვაწეობა მიეკუთვნება მხოლოდ ერთ ბილიკს, მაგრამ გადასვლებმა შეიძლება გადაკვეთონ ბილიკის საზღვრები.

მოღვაწეობის დიაგრამა შეიძლება დაუკავშიროთ მოდელის ნებისმიერ ელემენტს მისი ქცევის ვიზუალიზებისათვის, კერძოთ კლასებს, ინტერფეისებს, კომპონენტებს, კვანძებს, პრეცენდენტებს და კოოპერაციებს. ყველაზე ხშირათ მოღვაწეობის დიაგრამებს უკავშირებენ ოპერაციებს.

სისტემის დინამიური ასპექტების მოდელირებისას მოღვაწეობის დიაგრამები გამოიყენებიან ძირითადათ ორი საშუალებით:

- **სამუშაო პროცესის მოდელირებისათვის**. აქ ძირითადი ყურადღება ფიქსირდება მოღვაწეობაზე აქტიორების თვალთახედვიდან, რომლებიც თანამშრომლობენ სისტემასთან.
- **ოპერაციების მოდელირებისათვის**. ამ შემთხვევაში მოღვაწეობის დიაგრამები გამოიყენება როგორც ბლოქ - სქემები გამოთვლის დეტალების მოდელირებისათვის.

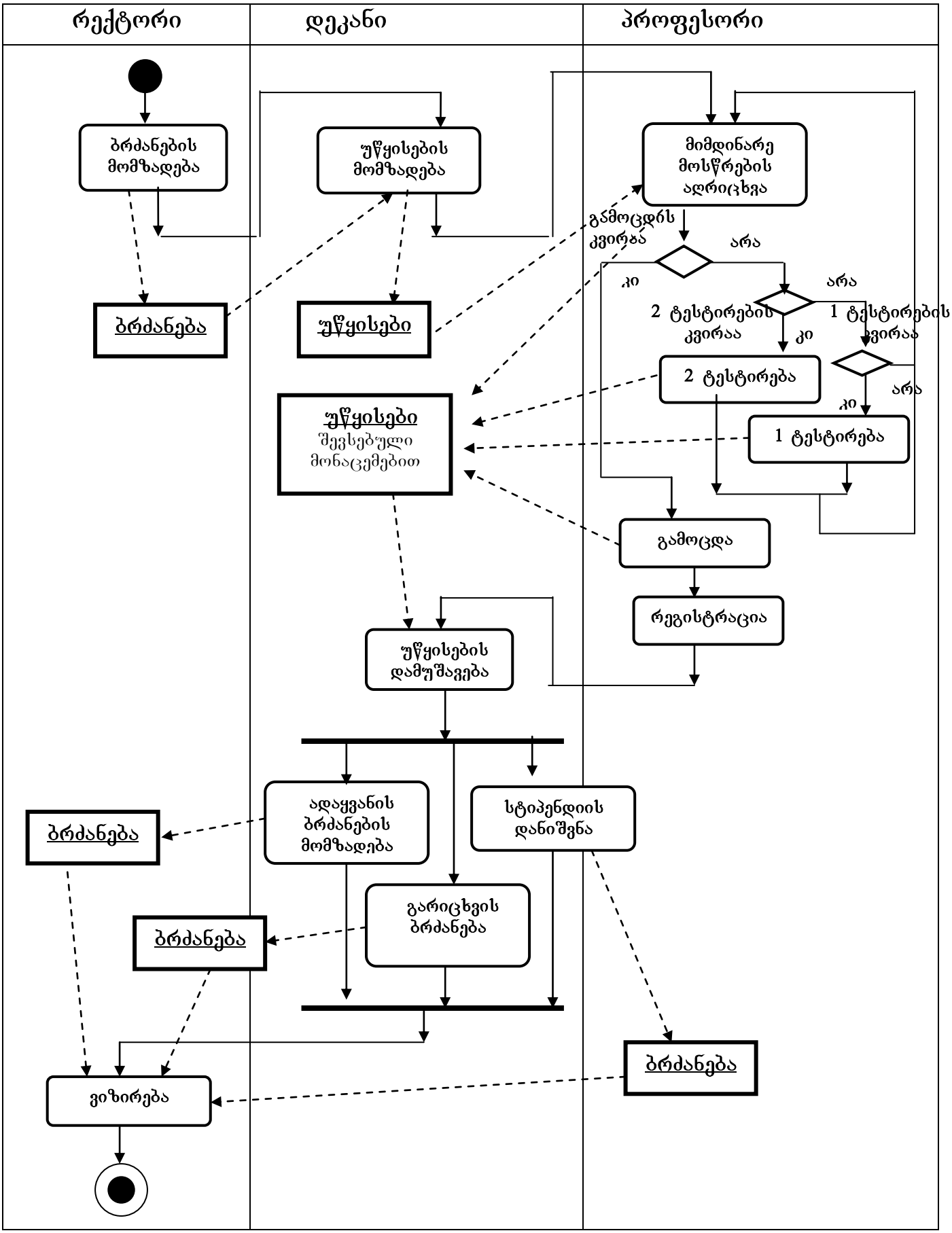
სამუშაო პროცესის მოდელის ასაგებად საჭიროა შემდეგი:

- გამოვყოთ სამუშაო პროცესის რომელიმე ნაწილი. რთული სისტემების პროექტირებისას, შეუძლებელია გამოვსახოთ ყველა საინტერესო თანმიმდევრობა ერთ დიაგრამაზე.
- ავირჩიოთ ობიექტები, რომლებზედაც დაკისრებულია პასუხისმგებლობა სამუშაო პროცესის ამა თუ იმ ნაწილზე. შევქმნათ ბილიკები ყოველი ასეთი ობიექტისათვის.
- მოვახდინოთ სამუშაო პროცესების საწყისი და საბოლაო მდგომარეობების პირობების იდენტიფიცირება. ეს დაგვეხმარება პროცესის საზღვრების მოდელირებისათვის.
- დაწყებული საწყისი მდგომარეობიდან ავღწეროთ მოღვაწეობა და მოქმედება, რომლებიც სრულდება დროის სხვადასხვა მომენტში, ამის შემდეგ გამოვსახოთ ისინი დიაგრამაზე მოღვაწეობის ან მოქმედების მდგომარეობის სახით.

- რთული მოქმედებები საჭიროა გამოიყოს მოღვაწეობის მდგომარეობაში და ყოველი ასეთი მდგომარეობისათვის შევადგინოთ მოღვაწეობის ცალკე დიაგრამა.
- გამოვსახოთ გადასვლები, რომლებიც აკავშირებენ ამ მოღვაწეობებისა და მოქმედებების მდგომარეობებს. თავიდან ყურადღება გავამახვილოთ მიმდევრობით ნაკადებზე, ხოლო შემდეგ გადავიდეთ განშტოებებზე და ბოლოს განვიხილოთ გაყოფა და შერწყმა.

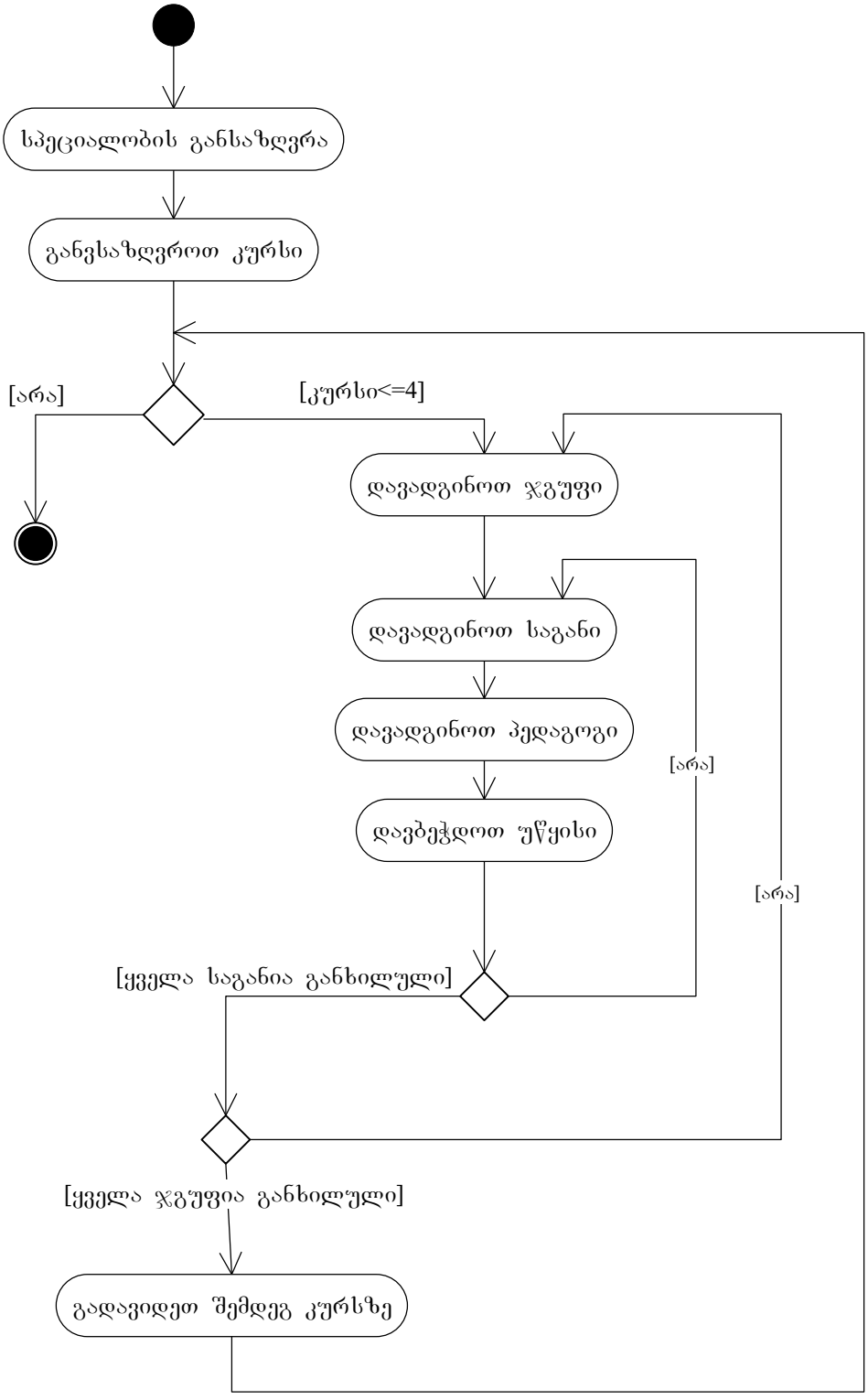
მაგალითისათვის ნახ.3.3.3.-ზე მოყვანილია აქტიურობის დიაგრამა, რომელიც ასახავს საქმეთა წარმოების ეტაპებს უმაღლეს სასწავლებელში სტუდენტთა მოსწრების პროცესს. როგორც ნახაზიდან ჩანს რექტორის ბრანების საფუძველზე, რომელიც განსაზღვრავს სასწავლო პროცესის ვადებს მიმდინარე წელს, დეკანატები ამზადებენ უწყისებს(სპეციალობის, კურსის, ჯგუფის, საგნისა და წამყვანი პედაგოგისათვის). უწყისები გადაეცემა პედაგოგებს, რომლებიც დადგენილ ვადებში ატარებენ – მიმდინარე მოსწრების აღრიცხვას, 1 შალედურ, 2 შუალედურ გამოცდებს და ბოლოს დასკვნით გამოცდას. მითითებული მონაცემებით შევსებული უწყისები რეგისტრირდებიან და გადაეცემა დეკანატში. მათი დამუშავების საფუძველზე მიიღება გადაყვანის, გარიცხვისა და სტიპენდიის დანიშვნის ბრძანებები.

დიაგრამაზე სამი ბილიკია რექტორი, დეკანატი და დეპარტამენტი - პროფესორი. დიაგრამაზე მოყვანილია როგორც მოქმედების, ისე მოღვაწეობის მდგომარეობები. უკანასკნელნი თავის მხრივ წარმოიდგინებიან მათი მოქმედების აღმწერი აქტიურობის დიაგრამებით, რომლებიც მოყვანილია ქვევით.

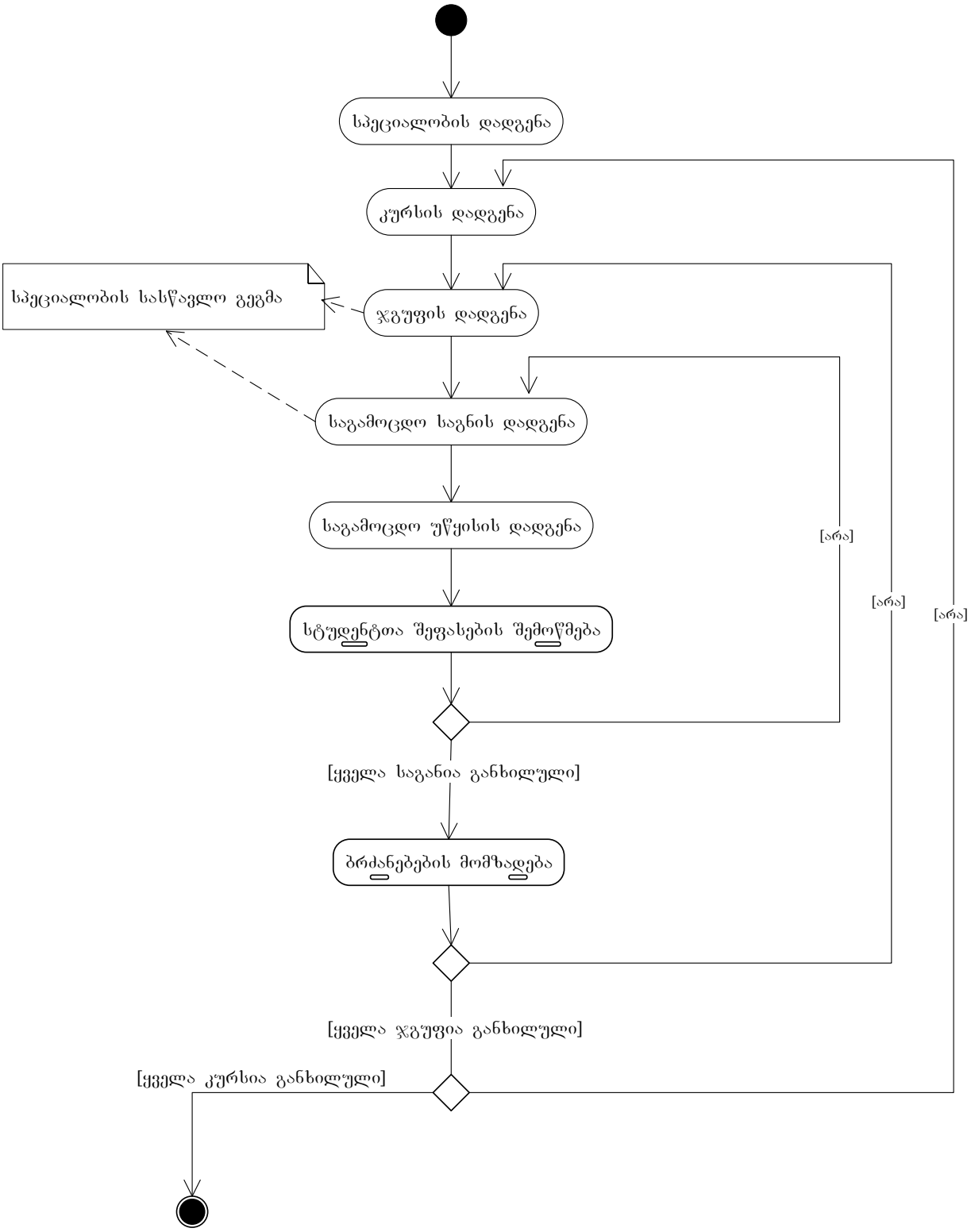

რექტორი
დეკანი
პროფესორი
ბრძანების მომზადება
უწყისების მომზადება
მიმდინარე მოსწრების აღრიცხვა
გამოცდის კვირაა
არა
კი
არა
2 ტესტირების კვირაა
კი
1 ტესტირების კვირაა
ბრძანება
უწყისები
2 ტესტირება
კი
არა
1 ტესტირება
უწყისები
შევსებული მონაცემებით
გამოცდა
რეგისტრაცია
უწყისების დამუშავება
ადაყვანის ბრძანების მომზადება
სტიპენდიის დანიშვნა
ბრძანება
გარიცხვის ბრძანება
ბრძანება
ბრძანება
ვიზირება

ნახ,3.3.3.

უწყისების მომზადება



ნახ.3.6.



ნახ.3.7.

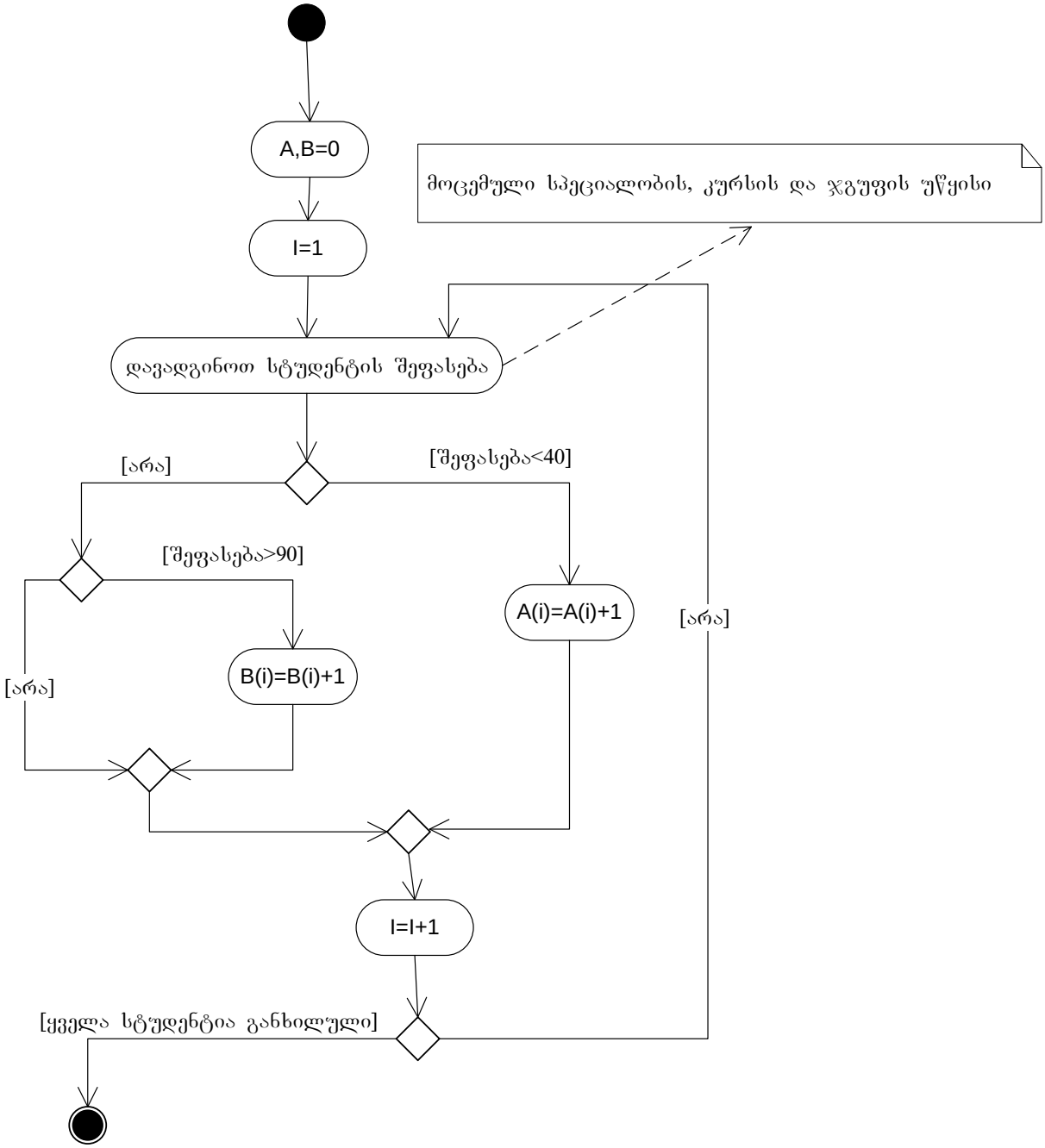
ოპერაციების მოდელირებისას მოღვაწეობის დიაგრამა ხდება შესასრულებელი მოქმედებების ბლოქ – სქემა. ოპერაციების მოდელირება შედგება შემდეგი ბიჯებისაგან:

- გამოვავლინოთ აბსტრაქციები, რომლებიც განეკუთვნებიან ოპერაციას.
- მოახდინეთ ოპერაციის საწყისი და საბოლაო მდგომარეობის პირობების იდენტიფიცირება.
- დაწყებული ოპერაციის საწყისი მდგომარეობიდან ავღწეროთ მოღვაწეობა და მოქმედება, რომლებიც სრულდება დროის სხვადასხვა მომენტში, ამის შემდეგ გამოვსახოთ ისინი დიაგრამაზე მოღვაწეობის ან მოქმედების მდგომარეობის სახით.
- აუცილებლობის შემთხვევაში გამოვიყენოთ განშტოების წერტილები პირობითი გადასვლების და იტერაციის აღწერისათვის.
- მხოლოდ იმ შემთხვევაში თუ ოპერაციის მფლობელი აქტიური კლასია და ამის აუცილებლობა წარმოიშობა, გამოვიყენოთ გაყოფისა და შერწყმის ოპერაციები შესრულების პარალელური ნაკადების შესრულებისათვის.

ზემოთ მოყვანილ მაგალითებიდან მნიშვნელოვანია შემოწმდეს სტუდენტთა შეფასებები და შეფასების საფუძველზე შევიტანოთ იგი შესაბამის ბრძანებაში.

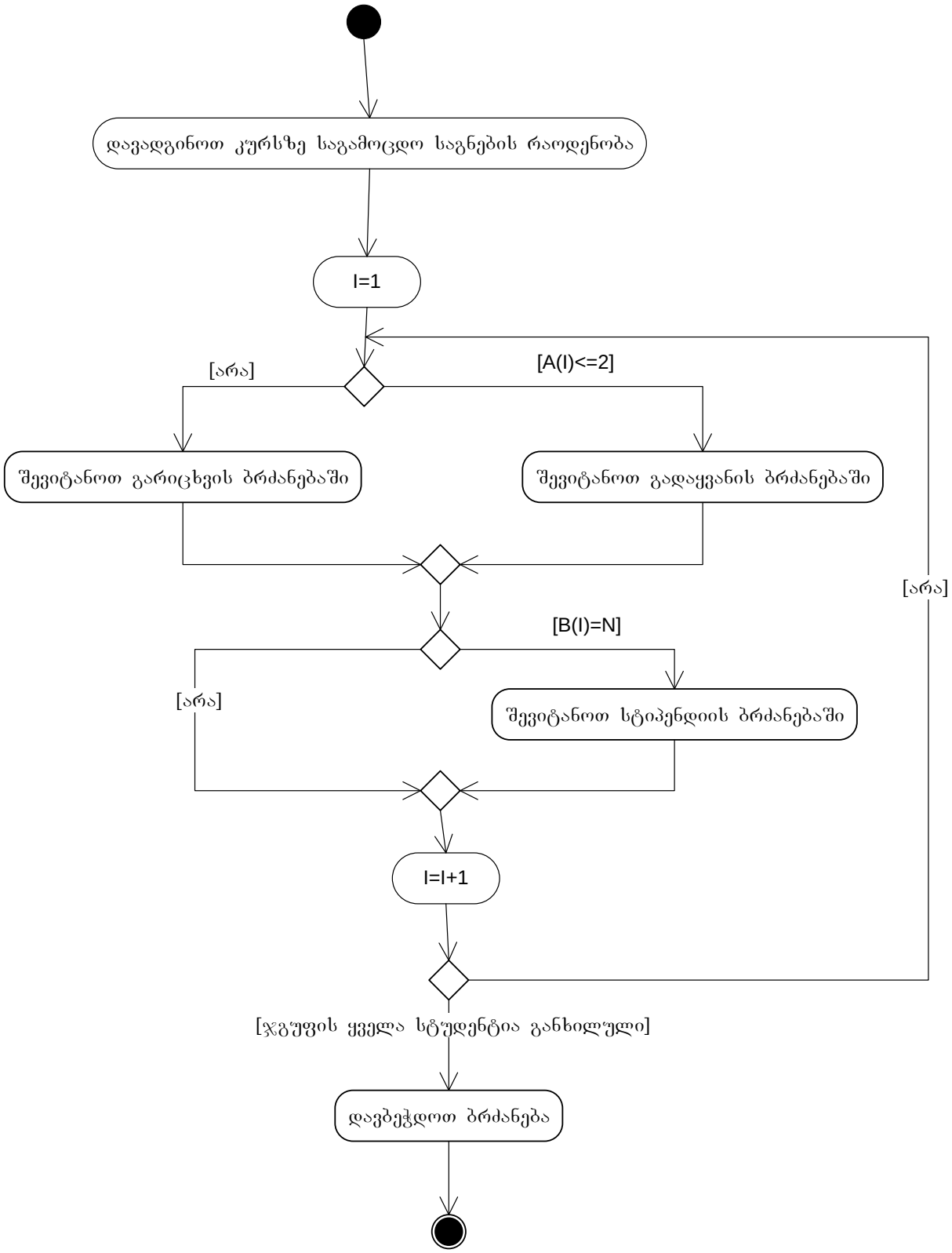
ქვემოთ მოყვანილია ორივე აღნიშნული ოპერაციის ამსახველი აქტიურობის დიაგრამა.

სტუდენტთა შეფასების შემოწმება



ნახ.3.8.

ბრძანების მომზადება

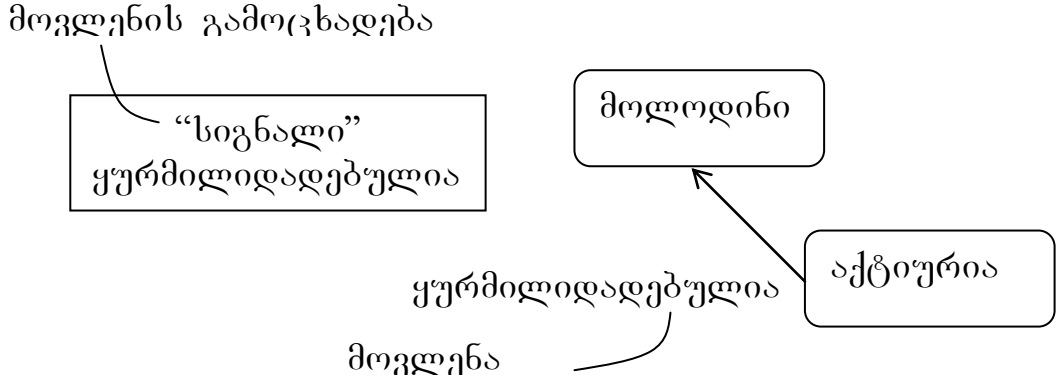


ნახ.3.9.

3.4. მოვლენები და სიგნალები

UML - ში ნებისმიერი წარმოსახვა, რომელიც შესაძლებელია მოხდეს, მოდელირდება როგორც მოვლენა.

მოვლენა – ეს არსებითი ფაქტის აღწერაა, რომელსაც აქვს გარკვეული ადგილი დროსა და სივრცეში. სიგნალის მიღება, დროითი შუალედის ამოწურვა და მდგომარეობის შეცვლა – ეს ასინქრონული მოვლენების მაგალითია, რომლებიც შეიძლება დადგეს ნებისმიერ მომენტში. გამოძახება – ეს სინქრონული მოვლენაა, რომელიც გამოიყენება რომელიმე ოპერაციის გასაშვებათ. ავტომატების კონტექსში მოვლენა ეს სტიმულია, რომელსაც შეუძლია გამოიწვიოს გადასვლა ერთი მდგომარეობიდან მეორეში (ნახ.1.).



ნახ.3.4.1.

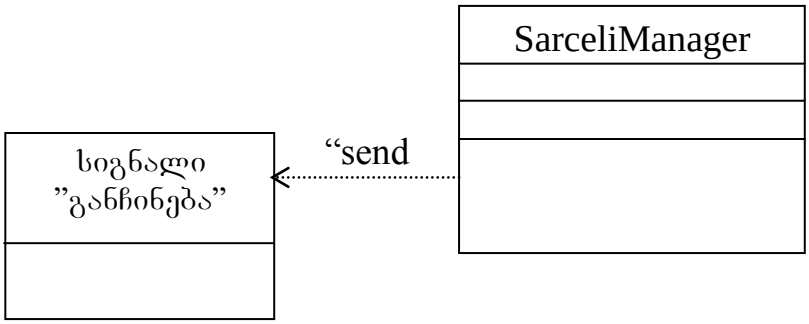
სიგნალი – ეს მოვლენის სახეა, რომელშიც სტიმული გადაეცემა ასინქრონულად ერთი ეგზემპლიარიდან მეორეს.

UML – ში შეიძლება მოვლენათა ოთხი ტიპის მოდელირება: სიგნალები, გამოძახება, დროის შუალედის ამოწურვა და მდგომარეობის შეცვლა.

სიგნალები. სიგნალი ეს დასახელებული ობიექტია, რომელიც ასინქრონულად აღიძვრება ერთი ობიექტით და მიიღება მეორეთი. გამორიცხვა, რომელიც ფართოდ გამოიყენება თანამედროვე დაპროგრამების ენების უმეტესობაში, ყველაზე გავრცელებული

სახეა შიდა სიგნალებს შორის. ობიექტების შექმნა და მოსპობაც სიგნალის ნაირსახეობას მიეკუთვნება.

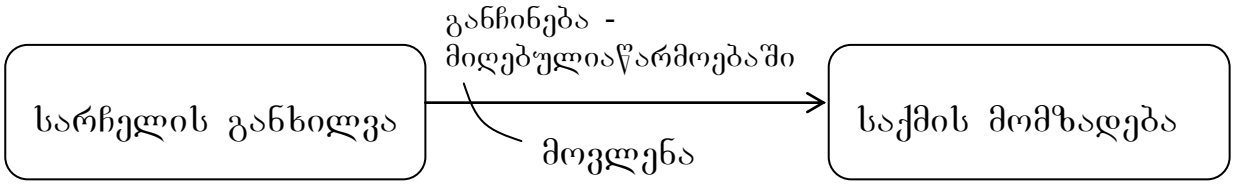
სიგნალი შეიძლება გაიგზავნოს როგორც მდგომარეობის გადასვლის მოქმედება ავტომატში ან როგორც შეტყობინების გაგზავნა ურთიერთქმედებისას. ოპერაციების შესრლებისას ასევე შეიძლება გაიგზავნოს სიგნალი. იმისათვის, რომ მიუთითოდ ოპერაციის მიერ სიგნალის გაგზავნა, შეიძლება ვისარგებლოთ მიმართებით დამოკიდებულება სტერეოტიპით **send**.



ნახ.3.4.2.

როგორც ნახაზიდან ჩანს სიგნალები მოდელირდებიან სტერეოტოპული კლასებით.

გამოძახების მოვლენები. თუ სიგნალის მოვლენა წარმოადგენს სიგნალის აღძვრას, გამოძახების მოვლენა განკუთვნილია ოპერაციის შესრულების აღწერისათვის. ორივე შემთხვევაში მოვლენას შეუძლია გამოიწვიოს მდგომარეობის შეცვლა ავტომატში. იმ დროს როდესაც სიგნალი წარმოადგენს ასინქრონულ მოვლენას, გამოძახების მოვლენა ჩვეულებრივ სინქრონულია. ეს ნიშნავს იმას, რომ როდესაც ერთი ობიექტი ინიციირებას ახდენს ოპერაციის შესრულებაზე მეორე ობიექტზე, რომელსაც გააჩნია თავისი ავტომატი, მართვა გადაიცემა გამომგზავნიდან მიმღებზე, ამუშავდება შესაბამისი გადასვლა, შემდეგ ოპერაცია სრულდება, მიმღები გადადის ახალ მდგომარეობაში და უბრუნებს მართვას გამომგზავნს.

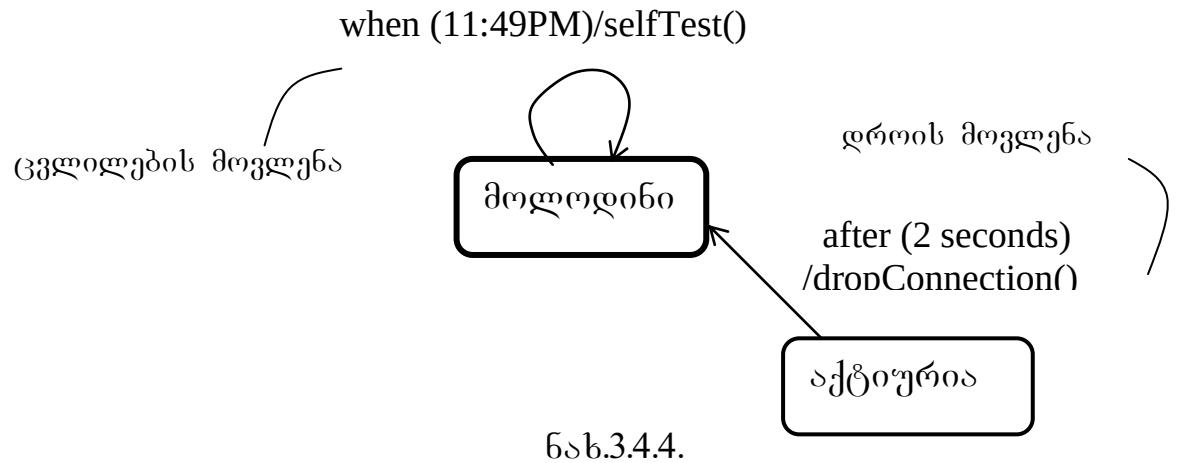


ნახ.3.4.3.

როგორც ნახ.3.4.3. – დან ჩანს ვიზუალურად სიგნალისა და გამოძახების მოვლენები არ განსხვავდებიან ერთმანეთისაგან. მაგრამ მოვლენის მიმღებმა რასაკვირველია იცის ამ განსხვავების შესახებ (ოპერაციის გამოცხადების საფუძველზე თავის ოპერაციათა ცხრილში). სიგნალი როგორც წესი მუშავდება ავტომატის დონეზე, ხოლო გამოძახების მოვლენა – მეთოდით. მოვლენიდან სიგნალზე ან ოპერაციაზე გადასასვლელად შესაძლებელია ვისარგებლოთ ინსტრუმენტალური პროგრამით.

დროითი და ცვლილების მოვლენები. დროითი მოვლენა წარმოადგენს დროითი შუალედის ამოწურვას. იგი წარმოიდგინება გასაღებური სიტყვით **after**(შემდეგ), მას მოსდევს გამოსახულება, რომელიც ითვლის დროის გარკვეულ შუალედს. გამოსახულება შეიძლება იყოს მარტივი ან რთული. თუ არ არის მითითებული, დროის ათვლა იწყება მიმდინარე მდგომარეობაში შესვლის მომენტიდან.

ცვლილების მოვლენით აღიწერება მდგომარეობის შეცვლა ან გარკვეული პირობის შესრულება. იგი წარმოიდგინება გასაღებური სიტყვით **when**, რომელსაც მოსდევს ბულის გამოსახულება. ასეთი გამოსახულება შესაძლებელია აბსოლუტური დროის მომენტის (მაგალითად, **when=11;59**) ან პირობის შესამოწმებლად (მაგალითად, **when altitude<1000**).

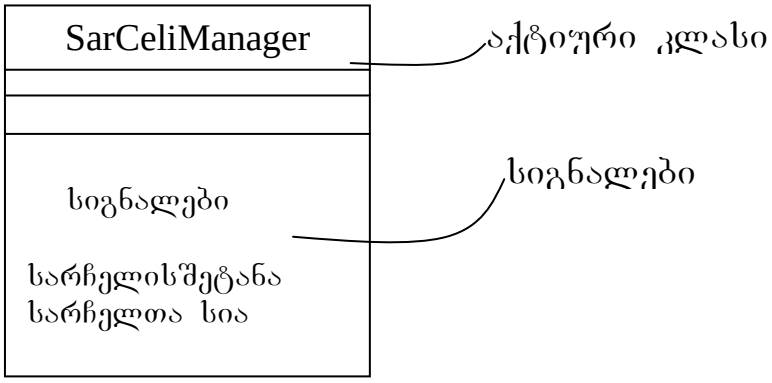


ნახ.3.4.4.

მოვლენის გაგზავნა და მიღება. სიგნალისა და გამოძახების მოვლენებში უკიდურეს შემთხვევაში მონაწილეობენ ორი ობიექტი: ობიექტი, რომელიც აგზავნის სიგნალს ანუ ოპერაციის ინიციირებას ახდენს, და ობიექტი, რომელსაც ეგზავნება მოვლენა. რამდენადაც სიგნალები თავისი ბუნებით ასინქრონულია, ხოლო ასინქრონული გამოძახებები თავისთავად წარმოადგენენ სიგნალებს, მოვლენათა სემანტიკა უკავშირდება აქტიური და პასიური ობიექტების სემანტიკას.

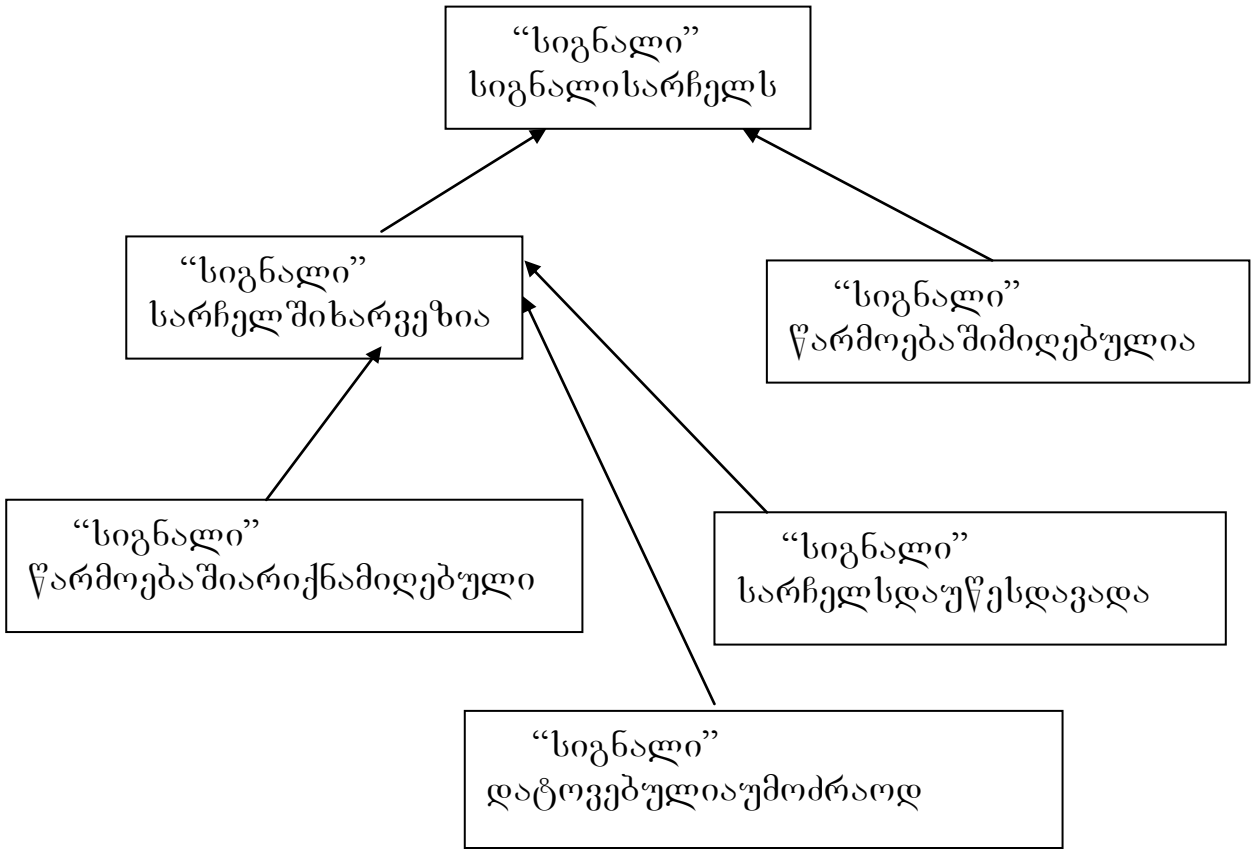
ნებისმიერი კლასის ეგზმლიარს შეუძლია გაგზავნოს სიგნალი მიმღებ ობიექტთან ან მოახდინოს მასში ოპერაციის ინიციირება. გაგზავნის რა სიგნალს მიმღებთან, იგი აგრძელებს თავის მართვის ნაკადს, არ ელოდება რა მისგან პასუხს. მაგალითად, აქტიორს (მოსარჩელეს) სამოქალაქო სამართალწარმოების სისტემაში სარჩელის შეტანიდან ხუთი დღის შემდეგ, შეუძლია გააგრძელოს თავისი საქმიანობა – შეასრულოს სხვა საპროცესო მოქმედებები იმისგან დამოუკიდებლად, თუ რა განაჩენს გამოიტანს მოსამართლე. პირიქით, თუ ობიქტი ახდენს ოპერაციის ინიციირებას, იგი უნდა დაელოდოს მიმღებისაგან პასუხს. იგივეა სამოქალაქო სამართალწარმოების სისტემაში, მოსამართლემ შესაძლებელია მოახდინოს ოპერაციის ***საქმის გახსნა*** ინიციირება კლასი სარჩელის რომელიღაც ეგზემპლიარში, ღებულობს რა ***განჩინებას*** სარჩელის წარმოებაში მიღების შესახებ, რომლითაც ფაქტიურად ცვლის მის

მდგომარეობას. რამდენადაც ეს სინქრონული გამოძახებაა, მოსამართლე დაელოდება, სანამ ოპერაცია დასრულდება.
ნებისმიერი კლასის ნებისმიერი ეგზემპლიარი შეიძლება იყოს მოვლენა სიგნალის ან მოვლენა გამოძახების მიმღები. თუ ეს სინქრონული მოვლენაა, გამგზავნი და მიმღები იმყოფებიან მდგომარეობაში რანდევუ ოპერაციის შესრულების მთელი ხანგრძლივობისას. ეს ნიშნავს, რომ გამგზავნის მართვის ნაკადი ბლოკირდება მიმღების მართვის ნაკადით, სანამ ოპერაცია არ დასრულდება. თუ ეს სიგნალია, გამგზავნი და მიმღები არ შედიან მდგომარეობაში რანდევუ: გამგზავნი აგზავნის სიგნალს, მაგრამ არ ელოდება პასუხს მიმღებისაგან. გამოძახების მოვლენები, რომლებსაც ღებულობს ობიექტი, მოდელირდება როგორც ოპერაციები ამ ობიექტის კლასებზე. დასახელებული სიგნალები, რომლებსაც ღებულობს ობიექტი, მოდელირდება ჩამონათვალის სახით კლასის დამატებით განყოფილებაში.



ნახ.3.4.5.

სისტემათა უმეტესობაში, რომლებიც იმართება მოვლენებით, მოვლენა სიგნალები ქმნიან სიგნალების იერარქიას. ნახ.3.4.6.-ზე მოყვანილია სიგნალების ოჯახი, რომლებიც შესაძლებელია დამუშავდეს სამოქალაქო სამართალწარმოების სიტემაში სარჩელების განხილვისას.



ნახ.3.4.6.

მოვახდენთ რა სიგნალთა იერარქიის მოდელირებას, შესაძლებლობა გვეძლევა მოვახდინოთ პოლიმორფული მოვლენების სპეციფიცირება. მაგალითად განვიხილოთ ავტომატი, რომელშიც გარკვეული გადასვლა ამუშავდება სიგნალი ***წარმოებაშიარიქნამიღებული***. რამდენადაც ეს სიგნალი იერარქიაში წარმოადგენს ფოთლოვანს, გადასვლა ინიციირდება მხოლოდ მის მიერ. მაგრამ დაუშვათ, რომ გვაქვს ავტომატი, რომელშიც გადასვლა ამუშავდება ***სარჩელშიხარვეზია*** სიგნალის მიღების შემდეგ. ასეთ შემთხვევაში გადასვლა პოლიმორფულია, რამდენადაც მას აღაგზნებს როგორც სიგნალი ***სარჩელშიხარვეზია,*** ასევე მისი ნებისმიერი სპეციალიზაცია (ნაირსახეობა) ***წარმოებაში არ იქნა მიღებული, დაუწესდა ვადა, დატოვებულიაუმოძრაოდ***.

სიგნალთა ოჯახის მოდელირება მოიცავს შემდეგ ეტაპებს:

1. განიხილეთ სიგნალების ყველა ნაირსახეობა, რომელზეც შესაძლებელია უპასუხოს აქტიური ობიექტების მოცემულმა სიმრავლემ.
2. გამოავლინეთ სიგნალების მსგავსი სახეობები და მოათავსეთ ისინი იერარქიაში "განზოგადება/სპეციალიზაცია" , მემკვიდრეობითობის გამოყენებით. იერარქიის მაღალ დონეებზე განლაგდება ზოგადი სიგნალები, ხოლო ქვედაზე – სპეციალიზირებულები.
3. განსაზღვრეთ პოლიმორფიზმის შესაძლებლობა ამ აქტიური ობიექტების ავტომატებში. პოლიმორფიზმის აღმოჩენისას, მოახდინეთ იერარქიის კორექტირება, დაამატეთ აუცილებლობის შემთხვევაში აბსტრაქტული სიგნალები.

გამორიცხვები. კლასების ან ინტერფეისების ქცევის ვიზუალირების მნიშვნელოვან ნაწილს წარმოადგენს გამორიცხვების სპეციფიცირება, რომლებსაც შეუძლიათ აღაგზნონ მისი ოპერაციები. თუ გაქვთ კლასი ან ინტერფეისი, მაშინ ოპერაციები, რომლებიც შესაძლებელია მათზე გამოვიძახოთ ნათლად ჩანს აღწერიდან, მაგრამ იმის გაგება თუ რომელ გამორიცხვებს აღაგზნებენ ისინი არ არის ადვილი, თუ ეს ნათლად არ არის მითითებული მოდელში.

გამორიცხვები წარმოადგენენ სიგნალების კერძო შემთხვევებს და მოდელირდებიან სტერეოტიპული კლასებით. გამორიცხვები შესაძლებელია მიუერთოდ ოპერაციების სპეციფიკაციებს. გამორიცხვების მოდელირება წარმოადგენს ოპერაციას, გარკვეული აზრით საწინააღმდეგოს სიგნალთა სიმრავლის მოდელირებისა. სიგნალთა ოჯახის მოდელირების ძირითადი მიზანია – მოვახდინოთ იმის სპეციფიცირება, თუ რომელი სიგნალები შეუძლია მიიღოს აქტიურმა ობიექტმა; გამორიცხვების მოდელირების მიზანია უჩვენოთ, რომელი გამორიცხვები შიძლება აღიძვრეს ობიექტით თავისი ოპერაციებიდან.

გამორიცხვები მოდელირება ხდება შემდეგნაირად:

1. ყოველი კლასისა და ინტერფეისისათვის და ყოველი მათში განსაზღვრული ოპერაციისათვის განიხილეთ, რომელი გამორიცხვების აღძვრა არის შესაძლებელი.
2. მოახდინეთ გამორიცხვების ორგანიზება იერარქიულად. მაღალ დონეებზე განვალაგოთ ზოგადი გამორიცხვები, ქვედაზე – სპეციალიზირებულები, საჭიროების შემთხვევაში შემოიტანეთ შუალედური გამორიცხვები.
3. მიუთითედ ყოველი ოპერაციისათვის, რომელი გამორიცხვები შეუძლია მან აღაგზნოს. ეს შეიძლება მიუთითოდ დიაგრამაზე (დამოკიდებულებით **send** ოპერაციიდან მის გამორიცხვამდე) ან მოვათავსოთ გამორიცხვების ჩამონათვალი ოპერაციათა სპეციფიკაციაში.

ნახ. 3.4.7. –ზე წარმოდგენილია გამორიცხვების იერარქიული მოდელი, რომლებიც შესაძლებელია აღიძვრას კლასით **SarCeliManager.** ამ იერარქიის თავში იმყოფება აბსტრაქტული კლასი **Exception,** ხოლო ქვევით სპეციალიზირებული გამორიცხვები. ნახაზზე მიღებულია შემდეგი აღნიშვნები:

X1 – სარჩელის შინაარსი აკმაყოფილებს საქართველოს სამოქალაქო კოდექსის(სსკ) 178-ე მუხლს;

X2 – სახელმწიფო ბაჟი შეტანილია(სსკ-ის 178-ე მუხლი);

X3 – სრულდება სსკ-ის 186-ე მუხლი.

X3 = X31 ∧ X32 ∧ X33 ∧ X34 ∧ X35 ∧ X36 ∧ X37 ;

სადაც,

X31 – სარჩელი არ ექვემდებარება სასამართლო უწყებას:

X32 - არსებობს სასამართლო გადაწყვეტილება ან განჩინება მოსარჩელის მიერ სარჩელზე უარის თქმის, მოპასუხის მიერ სარჩელის ცნობის ან მხარეთა მორიგების დამტკიცების შესახებ;

X33 – ამ ან სხვა სასამართლოს წარმოებაშია საქმე დავაზე იმავე მხარეებს შორის, იმავე საგანზე და იმავე საფუძვლით;

X34 – მხარეებს დადებული აქვთ ხელშეკრულება, რომ მათ შორის დავა გადასაწყვეტად გადაეცეს კერძო არბიტრაჟს;

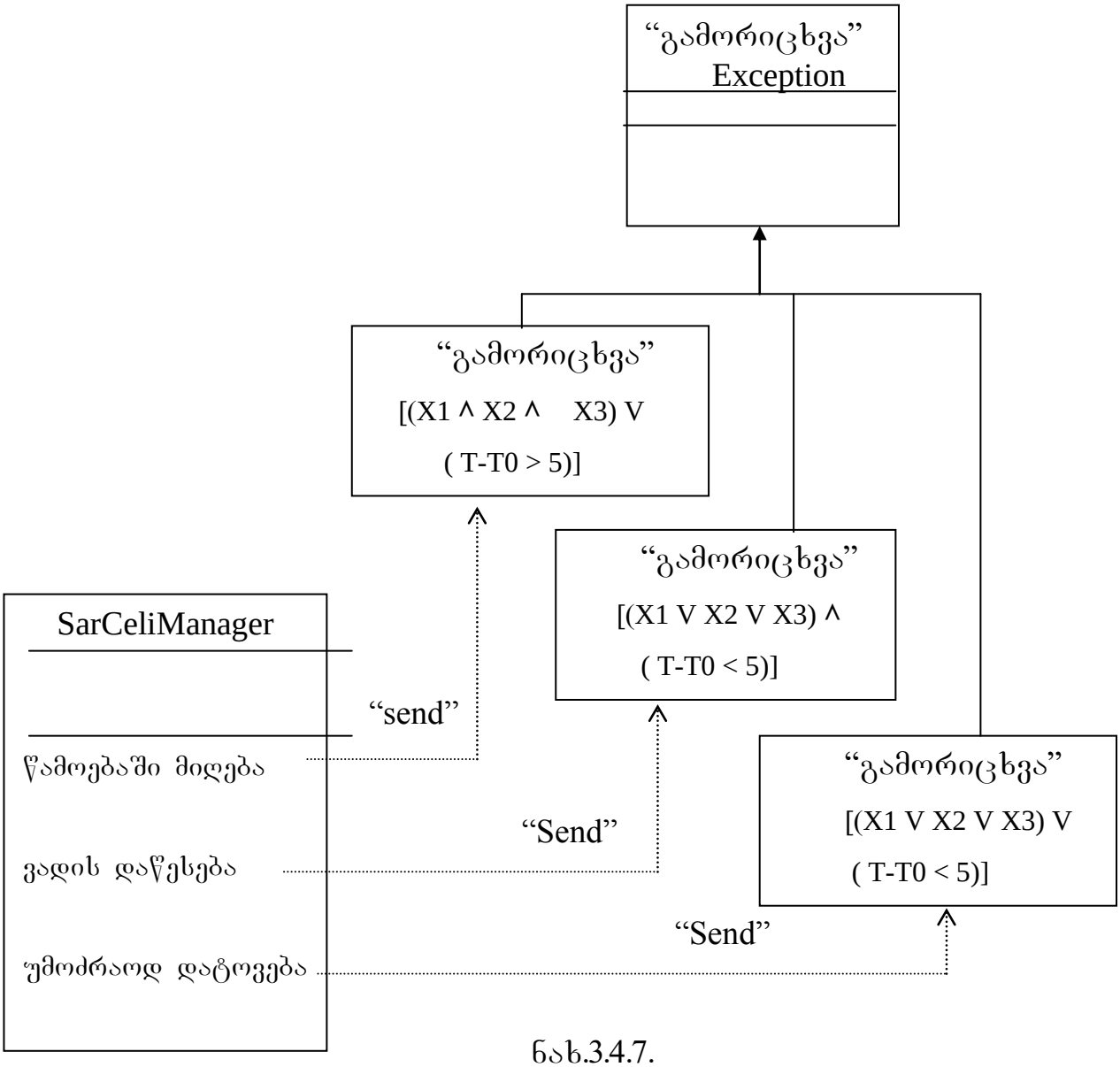
X35 – საქმე ამ სასამართლოს განსჯადი არ არის;

X36 – სარჩელი შეიტანა ქმედუუნარო პირმა;

X37 – დაინტერესებული პირის სახელით განცხადება შეიტანა პირმა, რომელსაც არა აქვს უფლებამოსილება საქმის წარმოებაზე.

T – მიმდინარე თარიღი;

T0 – მოსამართლეზე სარჩელის გადაცემის თარიღი.



ნახ.3.4.7.

3.5. ავტომატები

ურთიერთქმედებით შესაძლებელია ერთობლივად მომუშავე ობიექტის ქცევის მოდელირება. ავტომატი კი საშუალებას იძლევა მოვახდინოთ ცალკეული ობიექტის ქცევის მოდელირება. ავტომატი აღწერს ქცევას მიმდევრობითი მდგომარეობების სახით, რომლებზედაც გაივლის ობიექტი თავისი სიცოცხლის განმავლობაში.

ავტომატი გამოიყენება სისტემის დინამიური ასპექტების მოდელირებისათვის. როდესაც ხდება გარკვეული მოვლენა, ობიექტის მიმდინარე მდგომარეობიდან დამოკიდებულებით ადგილი აქვს ამა თუ იმ მოღვაწეობას. მოღვაწეობა ეს ავტომატის შიგნით არა ატომური გამოთვლებია, რომელიც გარკვეულ დროს იკავებს. მოღვაწეობის შედეგს წარმოადგენს გარკვეული მოქმედება, შედგენილი ატომური გამოთვლებისაგან, რომლებსაც მივყავართ მდგომარეობის შეცვლასთან ან მნიშვნელობის დაბრუნებასთან.

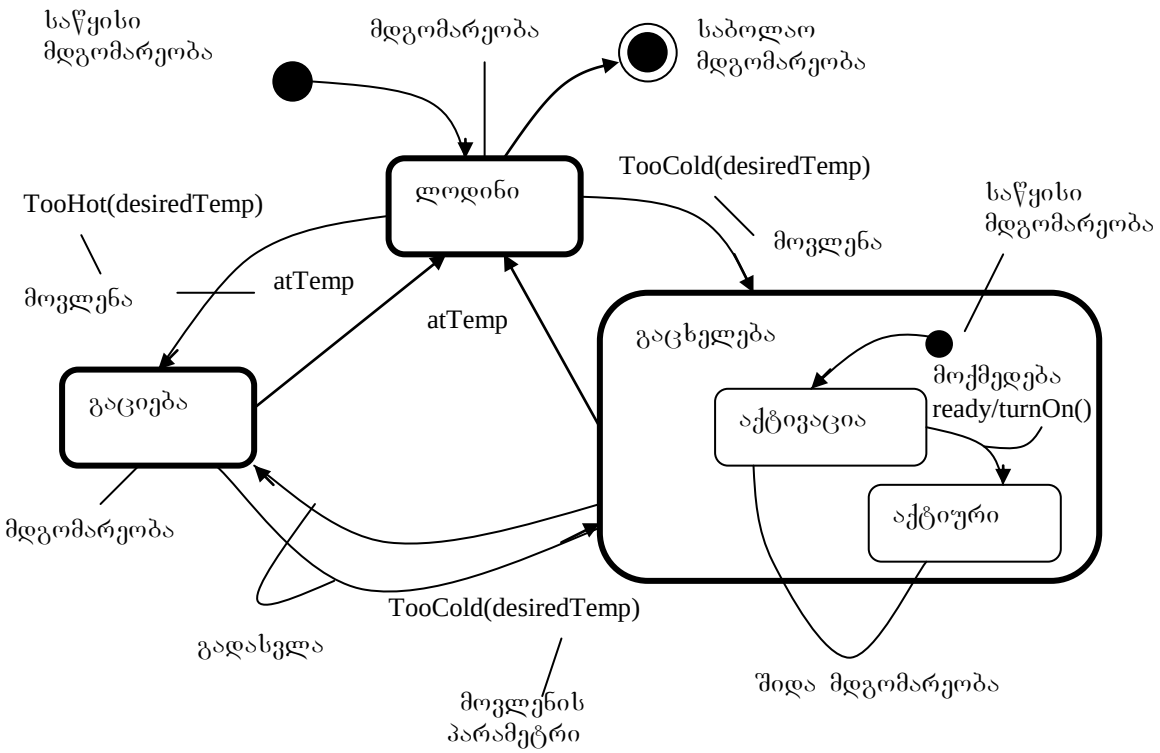
ავტომატის ვიზუალირება შესაძლებელია ორი საშუალებით:

- გამოვყოთ მართვის ნაკადის გადაცემა ერთი მოღვაწეობიდან მეორეზე (მოღვაწეობის დიაგრამის გამოყენებით);
- გამოვყოთ ობიექტების პოტენციალური მდგომარეობები და გადასვლები მათ შორის (მდგომარეობათა დიაგრამის სახით).

თავისი სიცოცხლის განმავლობაში ობიექტს შესაძლებელია შეხვდეს სხვადასხვა მოვლენები, როგორიც არის სიგნალი, მიმართვა გარკვეული ოპერაციის შესრულებაზე, ობიექტის შექმნა ან მოსპობა, დროის ინტერვალის გასვლა ან რომელიმე პირობის ცვლილება. მოვლენის საპასუხოდ ობიექტი ასრულებს გარკვეულ მოქმედებას, ანუ ატომურ გამოთვლებს, რომლებსაც მივყავართ ობიექტის

მდგომარეობის შეცვლამდე ან მნიშვნელობის დაბრუნებამდე. მაშასადამე, ობიექტის ქცევაზე გავლენას ახდენს მისი წარსული. ობიექტს შეუძლია "მიიღოს" მოვლენა, მოახდინოს რეაგირება მასზე მოქმედების შესრულებით, ხოლო შემდეგ შეიცვალოს თავისი მდგომარეობა. თუ ამის შემდეგ იგი მიიღებს სხვა მოვლენას, რეაქცია მასზე შესაძლებელია იყოს სხვაგვარი, იმ მდგომარეობიდან გამომდინარე, რომელშიც ის მოხვდა წინა მოვლენის შედეგად.

ავტომატები გამოიყენება მოდელის ნებისმიერი ელემენტის ქცევის მოდელირებისათვის, მაგრამ ყველაზე ხშირად – კლასების, პრეცენდენტების ან სისტემის მთლიანად.



ნახ.3.5.1.

UML-ში არის საშუალებები მდგომარეობების, გადასვლების, მოვლენების და მოქმედებების გრაფიკული წარმოდგენისათვის, როგორც ეს მოყვანილია ნახ.3.5.1.-ზე. ეს საშუალებას გვაძლევს მოვახდინოთ ობიექტის ქცევის ვიზუალირება ისე, რომ

წარმოვადგინოთ მისი სასიცოცხლო ციკლის ყველაზე მნიშვნელოვანი ელემენტები.

კონტექსტი. ყოველ ობიექტს გააჩნია სიცოცხლის გარკვეული დრო. ობიექტი იბადება, როდესაც მას ქმნიან, და ასრულებს არსებობას განადგურების შემდეგ. შუალედში მას შეუძლია იმოქმედოს სხვა ობიექტებზე (გაუგზავნოს შეტყობინება), ასევე თვითონ განიცადოს ზემოქმედება (წარმოადგენდეს შეტყობინების მიმღებს). ბევრ შემთხვევაში შეტყობინებები შეიძლება იყოს უბრალო სინქრონული გამოძახებები. მაგალითად, კლასი *კლიენტის* ეგზემპლიარმა შესაძლებელია მოახდინოს ოპერაცია *მივიღოთ ანგარიშის ბალანსი* კლასი *საბანკო ანგარიშის* ეგზემპლიარზე. იმისათვის, რომ მოვახდინოთ ასეთი ობიექტების ქცევის სპეციფიცირება, ავტომატი საჭირო არ არის, რამდენადაც დროის ნებისმიერ მომენტში მათი ქცევა არ არის დამოკიდებული მათ წარსულზე.

სხვა სახის სისტემებში არსებობენ ობიექტები, რომლებიც უნდა ახდენდნენ რეაგირებას ასინქრონულ სიგნალებზე და რომელთა მიმდინარე ქცევა დამოკიდებულია წარსულისაგან. რაკეტა "ჰაერი-ჰაერის" მიმართვის სისტემის ქცევა დამოკიდებულია მისი მიმდინარე მდგომარეობისაგან, მაგალითად *არ ფრინავს* (ცხადია რაკეტის გაშვება არ არის საჭირო თვითმფრინავიდან, რომელიც იმყოფება დედამიწაზე) ან *ძებნა* (არ შეიძლება მისი გაშვება, სანამ არ არის დადგენილი მიზანი).

თუ ობიექტებმა რეაგირება უნდა მოახდინონ ასინქრონულ მოქმედებებზე და მათი მიმდინარე მდგომარეობა დამოკიდებულია მათი წარსულისაგან, უკეთესი იქნება ასეთი ობიექტების ქცევა ავღწეროთ ავტომატების მეშვეობით. ამ კატეგორიას განეკუთვნება კლასების ეგზემპლიარები, რომლებსაც შეუძლიათ მიიღონ სიგნალები, აქტიური ობიექტების ჩათვლით. ავტომატებს იყენებენ

აგრეთვე სისტემების მოდელირებისათვის მთლიანობაში, განსაკუთრებით რეაქტიულის, ანუ ისეთების, რომლებმაც უნდა უპასუხონ სიგნალებს გარე აქტიორებისაგან.

მდგომარეობები. ობიექტის მდგომარეობა ეს სიტუაციაა მის ცხოვრებაში, რომლის განმავლობაშიც იგი აკმაყოფილებს გარკვეულ პირობებს, ახორციელებს გარკვეულ მოღვაწეობას ან ელოდება სხვა მოვლენას.

ობიექტი რჩება მოცემულ მდგომარეობაში დროის სასრულო პერიოდის განმავლობაში. მაგ. გამათბობელი სახლში შეიძლება იმყოფებოდეს ოთხიდან ერთერთ მდგომარეობაში: **ლოდინი** (ელოდება ბრძანებას "დავიწყო გაცხელება"); **აქტივაცია** (გაზი მიეწოდება, მაგრამ არ არის მიღწეული საჭირო ტემპერატურა); **აქტიური** (გაზი ჩართულია); **გამორთვა** (გაზი არ მიეწოდება).

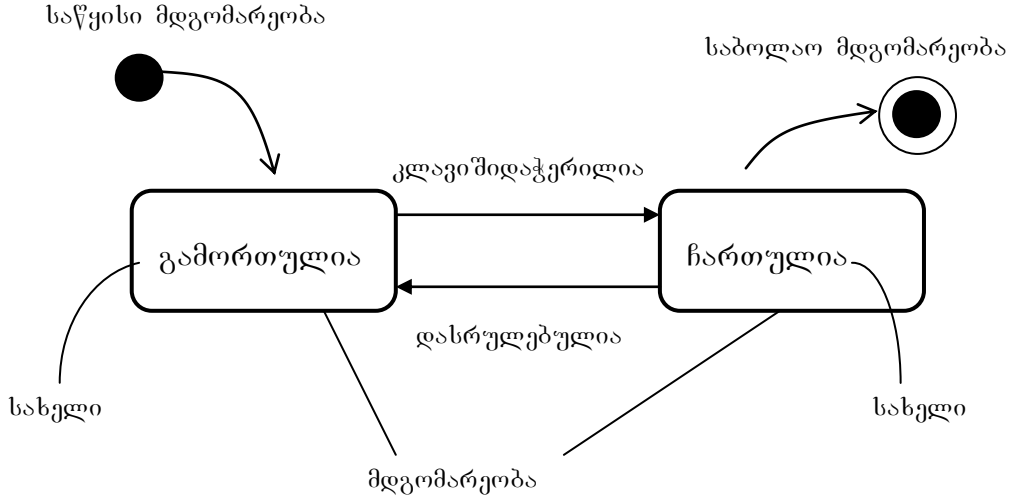
როდესაც ობიექტის ავტომატი იმყოფება მოცემულ მდგომარებაში, ამბობენ, რომ ობიექტიც იმყოფება ამ მდგომარეობაში.

მდგომარეობას განსაზღვრავენ შემდეგი ელემენტები:

- სახელი – ტექსტური სტრიქონი, რომელიც განასხვავებს ერთ მდგომარეობას ყველა დანარჩენისაგან;
- მოქმედება შესვლა/გამოსვლის დროს – მოქმედებები, რომლებიც სრულდება მდგომარეობაში შესვლისას და მისგან გამოსვლისას;
- შიდა გადასვლები – გადასვლები, რომლებიც მუშავდება მდგომარეობიდან გამოსვლის გარეშე;
- ქვემდგომარეობები – მდგომარეობების შიგნით შეიძლება არსებობდეს ქვემდგომარეობები, როგორც თანმიმდევრულად (აქტივიზირდებიან მიმდევრობით), ისე პარალელურად (აქტივიზირდებიან ერთდროულად);

• გადადებული მოვლენები – მოვლენების ცხრილი, რომლებიც არ არიან დამუშავებული მოცემულ მდგომარეობაში, გადადებულია და დაყენებულია რიგში ობიექტის მიერ დამუშავებისათვის, რომელიღაც სხვა მდგომარეობაში.

როგორც ნახ.3.5.2.-დან ჩანს, ობიექტის ავტომატში შესაძლებელია განსაზღვრული იყოს ორი სპეციალური მდგომარეობა. პირველ რიგში გვაქვს საწყისი მდგომარეობა, რომელშიც ავტომატი ან ქვემდგომარეობა იმყოფება დუმილით დროის საწყის მომენტში. მეორე, გვაქვს საბოლაო მდგომარეობა, რომელშიც მთავრდება ავტომატის ან მისი მომცავი მდგომარეობის შესრულება.



ნახ.3.5.2.

გადასვლები. გადასვლა ეს მიმართებაა ორ მდგომარეობას შორის, რომელიც გვიჩვენებს, რომ ობიექტმა, რომელიც იმყოფებოდა პირველ მდგომარეობაში, უნდა შეასრულოს გარკვეული მოქმედებები და გადავიდეს მეორე მდგომარეობაში, როგორც კი მოხდება მითითებული მოვლენა და დაკმაყოფილდება მითითებული პირობები. მდგომარეობის ასეთი ცვლილებისას ამბობენ, რომ ამუშავდა გადასვლა. სანამ გადასვლა არ ამუშავდა, ობიექტი იმყოფება საწყის მდგომარეობაში. ამუშავების შემდეგ ის იმყოფება მიზნობრივ მდგომარეობაში. მაგ. გამათბობელი შეიძლება გადავიდეს **ლოდინის**

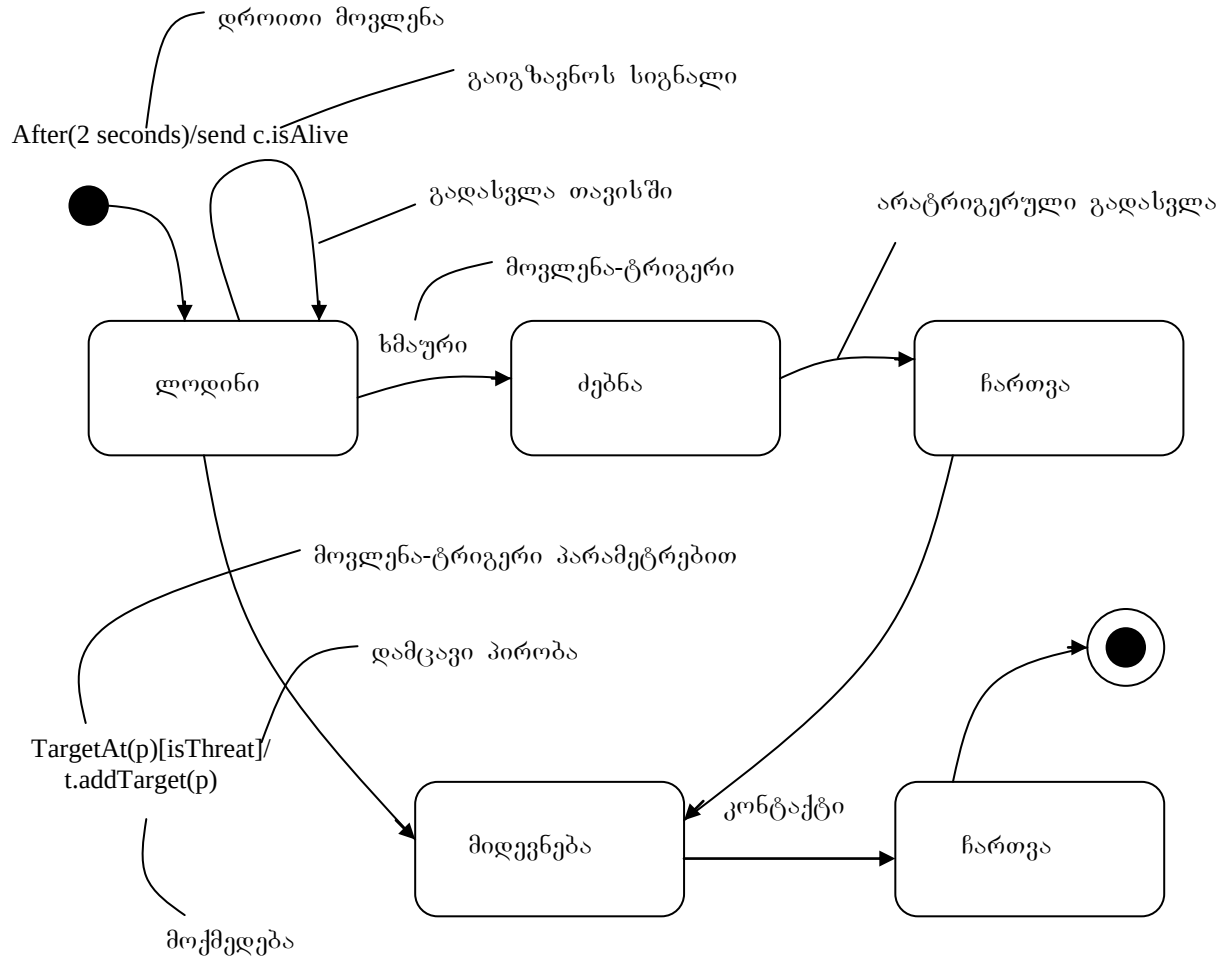
მდგომარეობიდან **აქტივაციის** მდგომარეობაში, როდესაც აღიძვრება მოვლენა (ძალიან ცივა) პარამეტრით (სასურველი ტემპერატურა). გადასვლას განმარტავენ ხუთი ელემენტით:

1. ***საწყისი მდგომარეობა***. მდგომარეობა, რომლიდანაც წარმოებს გადასვლა. თუ ობიექტი იმყოფება საწყის მდგომარეობაში, მაშინ საწყისიდან გადასვლა ამუშავდება, როდესაც ობიექტი მიიღებს მოვლენა-ტრიგერს, რომელიც ახდენს ამ გადასვლის ინიციირებას, ამასთან უნდა შესრულდეს დამცავი პირობა(თუ იგი მოცემულია);
2. ***მოვლენა-ტრიგერი***. მოვლენა, რომლის მიღებისას საწყის მდგომარეობაში მყოფ ობიექტზე შესაძლებელია ამუშავდეს გადასვლა (ამასთან უნდა სრულდებოდეს დამცავი პირობა);
3. ***დამცავი პირობა***. ბულის გამოსახულება, რომელიც გამოითვლება მოვლენა-ტრიგერის მიღებისას. თუ მნიშვნელობა ჭეშმარიტია, მაშინ გადასვლას ნება ეძლევა ამუშავდეს, თუ მცდარია- გადასვლა არ იმუშავებს. თუ ამ დროს არ არის მოცემული სხვა გადასვლა, რომლის ინიციირება იმავე მოვლენით ხდება, მაშინ მოვლენა იკარგება.
4. ***მოქმედება***. ატომური გამოთვლა, რომელსაც შეუძლია უშუალოდ იმოქმედოს ობიექტზე, რომელსაც გააჩნია ავტომატი ან მოახდინოს ირიბი ზემოქმედება სხვა ობიექტებზე, რომლებიც იმყოფებიან მის მხედველობაში.
5. ***მიზნობრივი მდგომარეობა***. მდგომარეობა, რომელიც ხდება აქტიური გადასვლის დამთავრების შემდეგ.

გადასვლას შეიძლება გააჩნდეს რამოდენიმე საწყისი მდგომარტეობა, ასევე რამოდენიმე მიზნობრივი მდგომარეობა.

მოვლენა-ტრიგერი. მოვლენა ეს არსებითი ფაქტის სპეციფიკაციაა, რომელიც ხდება სივრცეში და დროში. ავტომატების კონტექსში მოვლენა ეს გარკვეული სტიმულია,

რომელიც ახდენს გადასვლის ინიციირებას ერთი მდგომარეობიდან მეორეში. მოვლენების რიცხვში შედიან სიგნალები, გამოძახება, დროის მონაკვეთის ამოწურვა ან მდგომარეობის შეცვლა. სიგნალსა და გამოძახებას შეიძლება გააჩნდეს პარამეტრები, რომლის მნიშვნელობები მისაწვდომია გადასვლისათვის, მათ შორის დამცავი პირობის გამოთვლისას და მოქმედების შესრულებისას.



ნახ.3.5.3.

არსებობენ არატრიგერული გადასვლები, რომლებისთვისაც არ არის არავითარი მოვლენა-ტრიგერი. არატრიგერული გადასვლა, რომელსაც კიდევ უწოდებენ გადასვლას დასრულებისას, ინიციირდება გარე პირობებისგან დამოუკიდებლად, როდესაც მუშაობა საწყის მდგომარეობაში დამთავრდება (იხ.ნახ.3.5.3.).

დამცავი პირობები. დამცავი პირობები გამოიხატება ბულის გამოსახულებით, მოთავსებული კვადრატულ ფრჩხილებში, მოვლენა-

ტრიგერის შემდეგ. დამცავი პირობა ითვლება მხოლოდ მოვლენა-ტრიგერის აღძვრის შემდეგ, რომელიც ახდენს შესაბამისი გადასვლის ინიციირებას. ამიტომ ერთი მდგომარეობიდან შესაძლებელია რამოდენიმე გადასვლა ერთი და იგივე მოვლენა-ტრიგერით იმ შემთხვევაში, თუ არცერთი ორი დამცავი პირობიდან არ ხდებიან ერთდროულად ჭეშმარიტი.

დამცავი პირობა გამოითვლება ერთხელ ყოველი გადასვლისათვის მოვლენის დადგომის მომენტში, მაგრამ შესაძლებელია გამოითვალოს ხელახლა თუ გადასვლა ინიციირდება ხელმეორედ. ბულის გამოსახულებაში შეიძლება ჩავრთოთ პირობები, რომელშიც ფიგურირებენ ობიექტის მდგომარეობები (მაგ. **გამათბობელი in ლოდინი** ჭეშმარიტია, თუ ობიექტი გამათბობელი იმყოფება მდგომარეობაში **ლოდინი**).

მოქმედება. ეს ატომური გამოთვლაა. მოქმედებებს მიეკუთვნებიან ოპერაციათა გამოძახება (ობიექტის, რომელიც ფლობს ავტომატს, ასევე სხვა ნებისმიერი ხილვადი ობიექტის), სხვა ობიექტის შექმნა და მოსპობა, ან სიგნალის გაგზავნა ობიექტთან. სიგნალის გაგზავნის აღნიშვნისათვის განსაზღვრულია სპეციალური ნოტაცია – სიგნალის სახელს წინ უსწრებს გასაღებური სიტყვა **send**.

მოქმედება ყოველთვის ატომურია, ე.ი. არ შეიძლება შეწყდეს სხვა მოვლენით და შესაბამისად სრულდება დამთავრებამდე. ამით იგი განსხვავდება მოღვაწეობისაგან, რომლის შესრულება შესაძლებელია შეწყდეს სხვა მოვლენით.

3.5.1. მდგომარეობებისა და გადასვლების უფრო რთული ასპექტები

ზემოთ მოყვანილი ბაზური საშუალებებით შესაძლებელია მივიღოთ ავტომატები, რომლებიც აღწერენ ქცევის მოდელებს, რომელთა გრაფი შესდგება მხოლოდ წიბოებისაგან (გადასვლები) და წვეროებისაგან (მდგომარეობები).

მაგრამ ავტომატებს **UML**-ში აქვთ რიგი დამატებითი შესაძლებლობებისა, რომლებიც უზრუნველყოფენ უფრო რთული ქცევის მოდელების შექმნას. მათი გამოყენება ხშირად საშუალებას იძლევა შევამციროდ გადასვლებისა და მდგომარეობების რიცხვი. ასეთი დამატებითი შესაძლებლობების რიცხვს მიეკუთვნებიან *მოქმედებები შესვლისა და გამოსვლისას, შიდა გადასვლები, მოლვაწეობები და გადადებული მოვლენები.*

მოქმედებები შესვლისა და გამოსვლისას. მოდელირებისას ხშირად გვხვდება სიტუაციები, როდესაც აუცილებელია გარკვეული მოქმედების შესრულება დამოუკიდებლად იმისა რომელი გადასვლით იქნა შესრულებული შესვლა მდგომარეობაში. ასევე მდგომარეობიდან გამოსვლისას, საჭირო ხდება ერთი და იგივე მოქმედების შესრულება ნებისმიერი გადასვლისას. მოყვანილი ეფექტის მიღწევა შესაძლებელია უბრალო ავტომატების გამოყენებითაც, მოცემული მოქმედებების ასოციირებით ყოველ შემავალ და გამომავალ გადასვლებთან შესაბამისად. მაგრამ ასეთი მეთოდიკის დროს არ არის გამორიცხული შეცდომა – მთავარია არ დავივიწყოთ ეს მოქმედებები ყოველი ახალი მოქმედების დამატებისას. გარდა ამისა, მოდიფიკაციისას მოგვიხდება გადავსინჯოთ მდგომარეობასთან დაკავშირებული ყველა გადასვლები.

ამის თავიდან ასაცილებლად, სიმბოლოს, რომელიც აღნიშნავს მდგომარეობას, შეიძლება დაერთოს ნებისმიერი მოქმედება

შესვლისას (იგი აღინიშნება სიტყვით **entry**) და მოქმედება გამოსვლისას (აღინიშნება სიტყვით **exit**). მაშინ მითითებული მოქმედებები შესრულდებიან შესაბამისად მდგომარეობაში შესვლისას და მისგან გამოსვლისას.

შიდა გადასვლები. ვიმყოფებით რა რომელიღაც მდგომარეობაში, შესაძლებელია მივიღოთ მოვლენა, რომელიც სასურველია გადამუშავდეს მდგომარეობიდან გაუსვლელად. ასეთ დამუშავებას უწოდებენ შიდა გადასვლას. შიდა გადასვლასა და გადასვლას თავის თავში არის განსხვავება. თავის თავში გადასვლისას, მოვლენა ახდენს გადასვლის ინიციირებას, ხდება გამოსვლა მიმდინარე მდგომარეობიდან, სრულდება გარკვეული მოქმედება (თუ იგი სპეციფიცირებულია), რის შედეგადაც ვბრუნდებით საწყის მდგომარეობაში. რამდენადაც გადასვლისას თავის თავში ხდება გამოსვლა მდგომარეობიდან და ხელახალი შესვლა მასში, მაშინ სრულდება მოქმედება, ასოციირებული გადასვლასთან და გარდა ამისა მოქმედება მდგომარეობაში შესვლისას. დაუშვათ, რომ აუცილებელია დამუშავდეს მოვლენა, შესვლისას და გამოსვლისას მოქმედების აღუძვრელად. პრინციპში ეს შესაძლებელია გაკეთდეს უბრალო ავტომატებითაც, მაგრამ ამ დროს ყურადღებით უნდა ვიყოთ თუ რომელი გადასვლებისათვის უნდა შესრულდეს ეს მოქმედებები, ხოლო რომლისთვის არა.

ამის თავიდან ასაცილებლად სიმბოლოს, რომელიც აღნიშნავს მდგომარეობას შეიძლება დაერთოს შიდა გადასვლა (აღიწერება როგორც მოვლენა სიტყვით **newtarget**). თუ თქვენ იმყოფებით ასეთ მდგომარეობაში და ხდება მითითებული მოვლენა, მაშინ შესაბამისი მოქმედება სრულდება მდგომარეობიდან გამოსვლისა და ხელმეორედ შესვლის გარეშე, ანუ მოვლენა მუშავდება შესვლისა და გამოსვლის მოქმედებების აღუძვრელად.

მოღვაწეობა. როდესაც ობიექტი იმყოფება გარკვეულ მდგომარეობაში, ის ჩვეულებრივ უმოქმედოა და ელოდება გარკვეული მოვლენების აღძვრას. მაგრამ ხშირად საჭიროა უწყვეტად მიმდინარე პროცესების მოდელირება. ვიმყოფებით რა ასეთ მდგომარეობაში, ობიექტი დაკავებულია მანამდე, სანამ ეს მოღვაწეობა არ შეწყდება მოვლენით. გადასვლით **do** (შევასრულოთ) აღინიშნება ის სამუშაოები, რომლებიც უნდა შესრულდეს მდგომარეობის შიგნით მას შემდეგ როგორც კი დამუშავდება მდგომარეობაში შესვლის მოქმედება. მოღვაწეობა დაკავშირებული **do** გადასვლასთან შესაძლებელია იყოს სხვა ავტომატის სახელი. დასაშვებია მიმდევრობითი მოქმედებებიც, მაგ. **do**/op1;op2;op3. მითითებული მოქმედებების შესრულებისას მომცავ მდგომარეობას შეუძლია დაამუშაოს მოვლენები და არ არის გამორიცხული, რომ ეს გამოიწვევს მდგომარეობიდან გამოსვლას.

გადადებული მოვლენები. ნებისმიერ სამოდელო სიტუაციაში აუცილებელი ხდება გარკვეული მოვლენების გამოცნობა, ხოლო სხვების იგნორირება. გამოცნობილი მოვლენები მოდელირდებიან როგორც გადასვლის ტრიგერები, ხოლო იგნორირებადი მოდელში არ ჩაირთვებიან. მაგრამ ხანდახან საჭიროა მოვლენის გამოცნობა, მაგრამ მისი დამუშავების გადადება მომავლისათვის. ასეთი მოვლენები სპეციფიცირდებიან გადადებული მოვლენების მეშვეობით. გადადებული მოვლენები (**Deferred event**) – ეს მოვლენების ცხრილია, რომელთა აღძვრა კონკრეტულ მდგომარეობაში გადადებულია ისეთ მდგომარეობაში გადასვლამდე, სადაც ეს ცხრილი არ წარმოადგენს გადადებულ მოვლენას. ამ მომენტში მოვლენები შეიძლება დამუშავდნენ და მოახდინონ ამა თუ იმ გადასვლების ინიციირება, ისე თითქოს ისინი წარმოიშვნენ ამ მომენტში.

3.5.2. ქვემდგომარეობები

UML-ში ავტომატებს აქვთ თვისება, რომელიც საშუალებას იძლევა კიდევ უფრო გავამარტივოთ რთული ქცევის მოდელირება. ეს არის ქვემდგომარეობები (**Substate**) – მდგომარეობები, რომლებიც წარმოადგენენ სხვა მდგომარეობების ნაწილს. მაგ. გამათბობელი შეიძლება იმყოფებოდეს მდგომარეობაში **გაცხელება,** რომლის შიგნით არის კიდევ ერთი მდგომარეობა – **აქტივაცია**. ასეთ შემთხვევაში იტყვიან, რომ ობიექტი იმყოფება ერთდროულად მდგომარეობაში **გაცხელება** და **აქტივაცია**.

უბრალოს უწოდებენ ისეთ მდგომარეობას, რომელსაც არ გააჩნია შინაგანი სტრუქტურა. მდგომარეობას, რომელსაც გააჩნია ქვემდგომარეობები, ანუ ჩაშენებული მდგომარეობები, უწოდებენ შედგენილს. მას შესაძლებელია გააჩნდეს როგორც პარალელური (დამოუკიდებელი), ისე მიმდევრობითი ქვემდგომარეობები. შედგენილი მდგომარეობები გამოიძახება ისევე, როგორც მარტივი, მაგრამ აქვს დამატებითი გრაფიკული განყოფილება, რომელშიც ნაჩვენებია ჩართული ავტომატი. მდგომარეობათა ჩართვის სიღრმე არ არის შეზღუდული.

მიმდევრობითი ქვემდგომარეობები. ამ დროს ქვემდგომარეობები გადადიან თანმიმდევრობით. მიმდევრობითი ქვემდგომარეობების გამოყენება საშუალებას გვაძლევს გავამარტივოთ ამოცანის მოდელირება. ამ დროს გადასვლა მდგომარეობიდან, რომელიც იმყოფება შედგენილი მდგომარეობის გარეთ, შესაძლებელია განხორციელდეს როგორც თვით ამ მდგომარეობაში, ისე მის ნებისმიერ ქვემდგომარეობაში. თუ მიზნობრივი მდგომარეობა არის შედგენილი, მაშინ ჩადებულ ავტომატს უნდა გააჩნდეს საწყისი მდგომარეობა, სადაც მართვა ხვდება შედგენილ მდგომარეობაში შესვლისას მასთან ასოციირებული მოქმედების შესრულების შემდეგ

(თუ ასეთი არსებობს). თუ მიზნობრივს წარმოადგენს ერთერთი ჩართული მდგომარეობებიდან, მაშინ მართვა გადაეცემა მას, მაგრამ ისევ მომცავ შედგენილ მდგომარეობაში და ქვემდგომარეობაში შესვლის მოქმედების შესრულების შემდეგ.

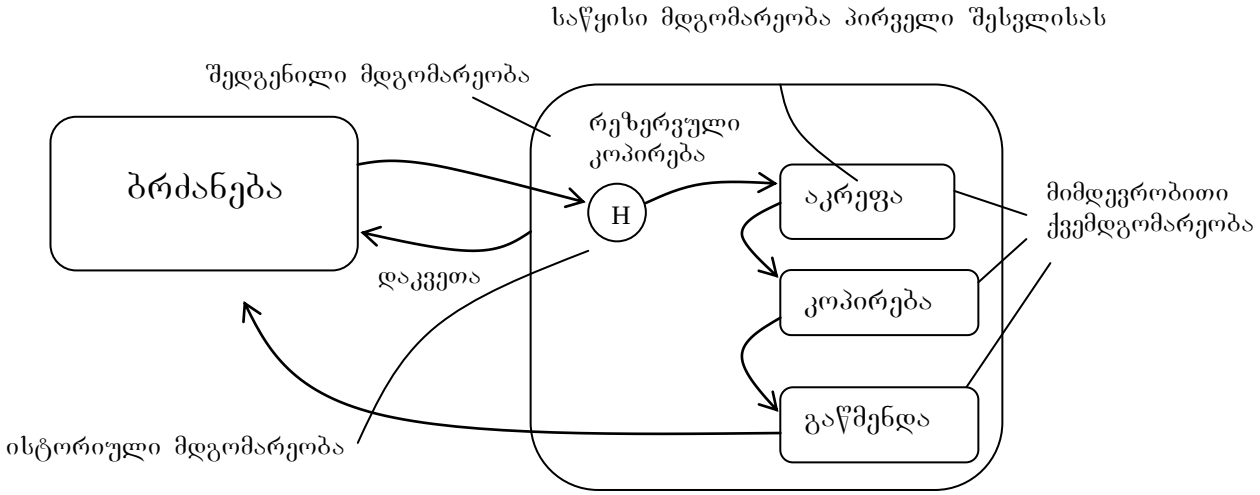
გადასვლისათვის, რომელიც გამომდინარეობს შედგენილი მდგომარეობიდან, საწყისი შეიძლება იყოს როგორც იგი თვითონ, ისე მისი რომელიმე ქვემდგომარეობა. ნებისმიერ შემთხვევაში მართვა თავიდან ტოვებს ჩართულ ქვემდგომარეობას (მაშინ სრულდება მისი მოქმედება გამოსვლისას, თუ იგი განსაზღვრულია), ხოლო შემდეგ შედგენილ მდგომარეობას (მაშინაც სრულდება მოქმედება გამოსვლისას). გადასვლა, რომლისთვისაც საწყისს წარმოადგენს შედგენილი მდგომარეობა, არსებითად წყვეტს ჩართული ავტომატის მუშაობას.

ჩართულ მიმდევრობით ავტომატს შეიძლება გააჩნდეს საწყისი და საბოლაო მდგომარეობები არა უმეტეს ერთისა.

ისტორიული მდგომარეობები. ავტომატი აღწერს ობიექტის დინამიურ ასპექტებს, რომლის მიმდინარე ქცევა დამოკიდებულია მისი წარსულისაგან. როდესაც გადასვლა ხდება შედგენილ მდგომარეობაზე, ჩართული ავტომატის მოქმედება იწყება მისი საწყისი მდგომარეობიდან (თუ რასაკვირველია გადასვლას არ მივყავართ რომელიმე ქვემდგომარეობაში). მაგრამ ხშირად საჭიროა დავამოდელიროთ ობიექტი ისე რომ მან დაიმახსოვროს ის ბოლო მდგომარეობა, რომელშიც ის იმყოფებოდა შედგენილი მდგომარეობიდან გამოსვლისას.

UML – ში ასეთი სიტუაციებისათვის გამოიყენება – ისტორიული მდგომარეობები (**History states**). ისტორიული მდგომარეობა საშუალებას აძლევს შედგენილ მდგომარეობას, რომელიც შეიცავს მიმდევრობით ქვემდგომარეობებს, დაიმახსოვროს რომელი ქვემდგომარეობა იყო მიმდინარე შედგენილ მდგომარეობიდან

გამოსვლის მომენტში. ისტორიული მდგომარეობა წარმოიდგინება წრით, რომელშიც იწერება სიმბოლო **H**.



ნახ.3.5.4.

ნახ.3.5.4. -ზე მოყვანილ მაგალითში სისტემას გააჩნია ორი მდგომარეობა, რომელთაგან ერთი შედგენილი მდგომარეობაა (შეიცავს მიმდევრობით ქვემდგომარეობებს). შედგენილ მდგომარეობაში პირველი შესვლისას ისტორია არ გვაქვს. ამიტომ ისტორიული მდგომარეობიდან გადასვლა ხდება მიმდევრობითი ქვემდგომარეობების საწყის ქვემდგომარეობაში ***აკრეფა.*** დაუშვათ, რომ მოვლენა ***დაკვეთა*** შემოვიდა, როდესაც აგენტი იმყოფებოდა მდგომარეობაში ***რეზერვული კოპირება*** და ქვემდგომარეობაში ***კოპირება.*** ამ დროს მართვა ტოვებს რა ორივე ამ მდგომარეობას ბრუნდება მდგომარეობაში ***ბრძანება.*** როდესაც მთავრდება მოქმედება მდგომარეობაში ***ბრძანება,*** არატრიგერული გადასვლა აბრუნებს აგენტს შედგენილი მდგომარეობის ***რეზერვული კოპირება*** ისტორიულ მდგომარეობაში. ამჯერად, რამდენადაც ჩართულ ავტომატს უკვე აქვს ისტორია, მართვა გადადის მდგომარეობაში ***კოპირება,*** გვერდს უვლის მდგომარეობას ***აკრეფა***, რამდენადაც ***კოპირება*** იყო ბოლო აქტიური ქვემდგომარეობა მდგომარეობიდან ***რეზერვული კოპირება*** გამოსვლისას.

H სიმბოლოთი აღინიშნება შედარებით ახლო ისტორია, რომელშიც დაიმახსოვრება წინამდებარე მდგომარეობა, მაგრამ შეიძლება განისაზღვროს შორეული (**deep**) ისტორია, რომელიც გამოიხატება სიმბოლოთი **H***. შორეული ისტორია საშუალებას იძლევა დავიმახსოვროთ ჩართული ქვეავტომატების საბოლაო მდგომარეობები ჩართვის ნებისმიერი დონისათვის.

პარალელური ქვემდგომარეობები. მიმდევრობითი ქვემდგომარეობები გვხვდება უფრო ხშირათ. მაგრამ ზოგიერთ სიტუაციებში აღიძვრება აუცილებლობა პარალელურ ქვემდგომარეობებში. ისინი საშუალებას იძლევიან მოვახდინოთ ორი ან მეტი ავტომატის სპეციფიცირება, რომლებიც სრულდებიან პარალელურად მომცავი ობიექტის კონტექსში. ორი პარალელური ქვემდგომარეობის შესრულება მიმდინარეობს ერთდროულად. ბოლოს და ბოლოს ყოველი ჩართული პარალელური ქვეავტომატი აღწევს თავის საბოლაო მდგომარეობას. თუ ერთ ერთი მივა ბოლო მდგომარეობაში უფრო ადრე ვიდრე მეორე, მაშინ მართვის ნაკადი დაელოდება, როდესაც მეორე ავტომატიც მივა თავის საბოლაო მდგომარეობამდე, ამის შემდეგ ორივე ნაკადი ხელახლა შეერწყმება ერთში.

თუ არის გადასვლა შედგენილ მდგომარეობაში, გაყოფილი პარალელურ ქვემდგომარეობებზე, მაშინ მართვის ნაკადი იყოფა იმდენ პარალელურ ნაკადათ რამდენი ქვემდგომარეობებიც არის. პირიქით, შედგენილი მდგომარეობებიდან (გაყოფილი პარალელურ ქვემდგომარეობებად) გადასვლისას, ნაკადები ერთიანდებიან ერთში. ეს სამართლიანია ყველა შემთხვევაში. თუ ყველა პარალელური ქვეავტომატები აღწევენ საბოლაო მდგომარეობას ან არის აშკარა გადასვლა მომცავი შედგენილი მდგომარეობიდან, ყველა პარალელური ნაკადები ერთიანდებიან.

ჩართულ პარალელურ ავტომატებს არ გააჩნიათ საწყისი, საბოლაო ან ისტორიული მდგომარეობები. მაგრამ მიმდევრობით ქვემდგომარეობებს, რომლებიც შედიან პარალელურების შემადგენლობაში, შეიძლება გააჩნდეთ ასეთი თვისებები.

ავტომატები გამოიყენება ობიექტის სასიცოცხლო ციკლის მოდელირებისათვის, განსაკუთრებით როდესაც ეს კლასი, პრეცენდენტი ან სისტემაა მთლიანობაში. ობიექტის სასიცოცხლო ციკლის მოდელირებისას განსაკუთრებული ყურადღება ენიჭება შემდეგი ელემენტების სპეციფიცირებას: მოვლენები, რომლებზედაც ობიექტი უნდა რეაგირებდეს, რეაქციები ასეთ მოვლენებზე, და ასევე წარსულის გავლენა ქცევაზე მიმდინარე მომენტში.

ობიექტის სასიცოცხლო ციკლის მოდელირება ხორციელდება შემდეგნაირად:

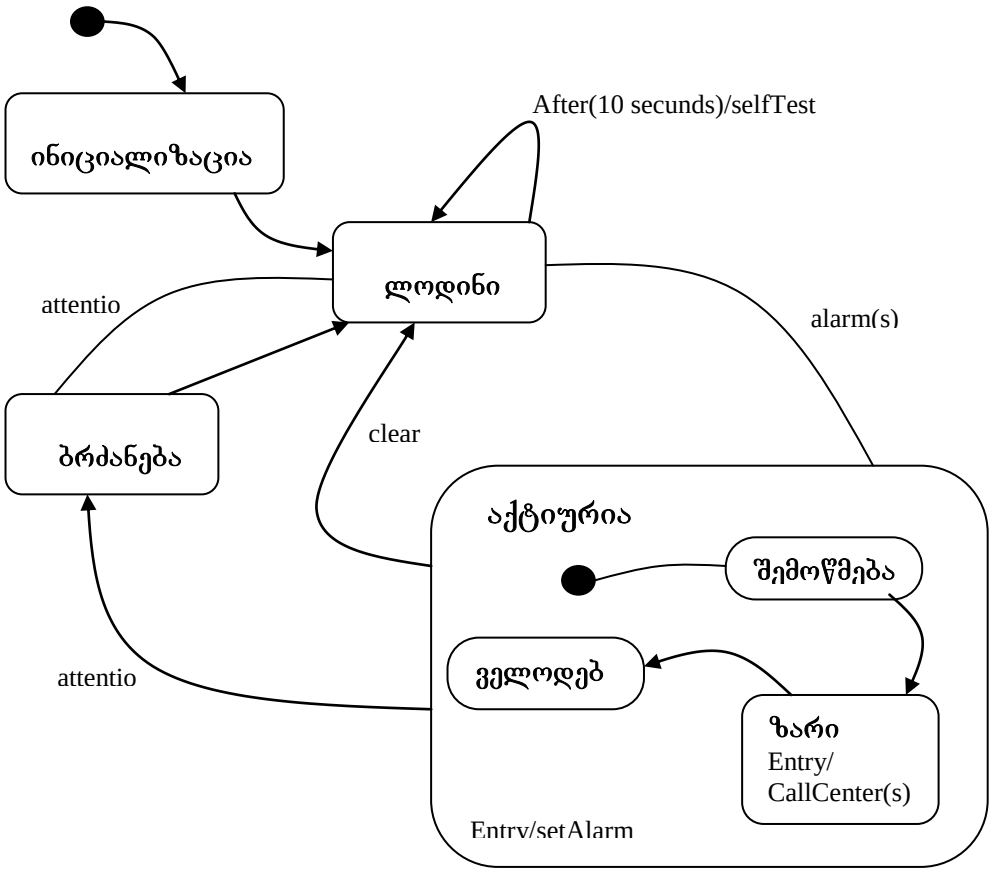
1. ამოვირჩიოთ კონტექსი ავტომატისათვის, ეს იქნება კლასი, პრეცენდენტი ან სისტემა მთლიანობაში. ამასთან
- თუ კონტექსს წარმოადგენს კლასი ან პრეცენდენტი, მოახდინეთ მეზობელი კლასების იდენტიფიცირება, რომლებზედაც შესაძლებელია მიმართვა მოცემულიდან, დამოკიდებულებებით და ასოციაციით (ასეთი მეზობლები არიან მოქმედებათა მიზნების როლის ან დამცავ პირობებში ჩასართველი კანდიდატები);
- თუ კონტექსს წარმოადგენს სისტემა მთლიანობაში, მაშინ ყურადღება უნდა გამახვილდეს მისი ქცევის ერთ რომელიმე მხარეზე.
2. დავადგინოთ ობიექტისათვის საწყისი და საბოლაო მდგომარეობები.
3. გადაწყვიტეთ რომელ ობიექტებზე შეუძლია ობიექტს რეაგირება. თუ ობიექტის ინტერფეისები უკვე სპეციფიცირებულია, მათში არის სწორედ ამ მოვლენების

აღწერა, წინააღმდეგ შემთხვევაში საჭიროა განვიხილოთ, რომელი ობიექტები უნდა განვიხილოთ, რომელი ობიექტები ურთიერთქმედებენ ობიექტთან მოცემულ კონტექსში და რომელი მოვლენები შეუძლიათ მათ გაგზავნონ.

4. გამოსახეთ ზედა დონის ყველა მდგომარეობები, საწყისიდან ბოლომდე, რომლებშიც შეიძლება იმყოფებოდეს ობიექტი. დააკავშირეთ ეს მდგომარეობები გადასვლებით, ინიციირებული ამა თუ იმ მოვლენებით, ხოლო შემდეგ შეავსეთ ეს გადასვლები მოქმედებებით.
5. მოახდინეთ შესვლისა და გამოსვლის მოქმედებების იდენტიფიცირება.
6. თუ საჭიროა, გაშალეთ ამ დროს გამოვლენილი მდგომარეობები, გამოიყენეთ რა ქვემდგომარეობების აპარატი.
7. დარწმუნდით, რომ ყველა მოვლენა, რომელიც გვხვდება ავტომატში, შეესაბამებიან იმას, რასაც ელოდება ობიექტის ინტერფეისი. საჭიროა დავრწმუნდეთ აგრეთვე იმაში, რომ მოვლენებს, რომლებიც ობიექტის ინტერფეისს ელოდებიან, ნახონ თავიანთი გამოსახვა ავტომატში. ბოლოს უნდა ვნახოთ ისეთი ადგილები, სადაც აზრი აქვს მოვლენების იგნორირებას.
8. დარწმუნდით, რომ ყველა მოქმედებები, რომლებიც მოხსენებულია ავტომატში, შეთავსებულია მომცავი ობიექტის მიმართებებით, მეთოდებით და ოპერაციებით.
9. მიყევით ავტომატის გადასვლებს ხელით ან ინსტრუმენტალური საშუალებით, შეამოწმეთ მოვლენათა მოსალოდნელი მიმდევრობები და რეაქციები მათზე. განსაკუთრებული ყურადღება უნდა მიაქციოთ მიღწეულ მდგომარეობებს და მდგომარეობებს, რომლებშიც ავტომატი შესაძლებელია “გაჩერდეს”(ჩიხებს).

10. შემოწმების შედეგად შეცვალეთ ავტომატის სტრუქტურა, ხელახლა გასინჯეთ იგი მოსალოდნელ მოვლენათა თანმიმდევრობაზე, რათა დარწმუნდეთ, რომ ობიექტის სემანტიკა შეიცვალა.

მაგალითისათვის ნახ.3.5.5.-ზე მოყვანილია ავტომატი სახლის უსაფრთხოების სისტემის კონტროლირებისათვის, რომელიც თვალყურს ადევნებს სხვადასხვა საზომს განთავსებულს სახლში.



ნახ.3.5.5.

ასეთი კონტროლერის სასიცოცხლო ციკლში არის ოთხი მდგომარეობა: **ინიციალიზაცია** (კონტროლერი შედის სამუშაო რეჟიმში), **ლოდინი** (კონტროლერი მზად არის სამუშაოდ და ელოდება სახიფათო სიგნალებს ან ბრძანებებს მომხმარებლისაგან), **ბრძანება** (კონტროლერი დაკავებულია მომხმარებლის ბრძანების დამუშავებით) და **აქტიურია** (კონტროლერი ამუშავებს განგაშის ბრძანებას).

როდესაც კონტროლერის ობიექტი იქმნება პირველად, იგი თავიდან გადადის მდგომარეობაში **ინიციალიზაცია**, ხოლო შემდეგ ყოველგვარი პირობის გარეშე – მდგომარეობაში **ლოდინი**. დაწვრილებით ეს ორი მდგომარეობა ნაჩვენები არ არის; ნაჩვენებია მხოლოდ გადასვლა თავის თავში დროის მოვლენით მდგომარეობისათვის **ლოდინი**. ასეთი სახის დროითი მოვლენები ტიპიურია ჩართული სისტემებისათვის, რომლებშიც ხშირად არის ტაიმერი, განკუთვნილი თვითდიაგნოსტიკის პროცედურის პერიოდული გაშვებისათვის.

მოვლენა **alarm**(განგაში) მიღებისას, რომელსაც გააჩნია პარამეტრი **s**, მომწოდის ამუშავების მაჩვენებელი, მართვა გადადის მდგომარეობიდან ლოდინი მდგომარეობაში აქტიური. მდგომარეობა **აქტიური** შესვლისას სრულდება მოქმედება შესვლისას **setAlarm** და მართვა ხვდება მდგომარეობაში **გასინჯვა** (სადაც ისინჯება ყალბი ხომ არ არის განგაში), შემდეგ მდგომარეობაში **ზარი** (კომპანიაში, რომელმაც დააყენა სიგნალიზაცია განგაშის რეგისტრაციისათვის) და ბოლოს მდგომარეობაში **ველოდები**. მდგომარეობიდან **ველოდები** და **აქტიურია** კონტროლერს შეუძლია გამოსვლა მხოლოდ განგაშის სიგნალის მოხსნის შემდეგ(მოქმედება **clear**) ან მომხმარებლისაგან მოვლენა **attention** მიღებით, რომელიც შესაძლებელია წინ უსწრებდეს ბრძანების გამოცემას.

თუ დავაკვირდებით, მოყვანილ მაგალითს არ გააჩნია საბოლაო მდგომარეობა. ესეც დამახასიათებელია ჩართული სისტემებისათვის, რომლებიც უნდა მუშაობდნენ უწყვეტად.

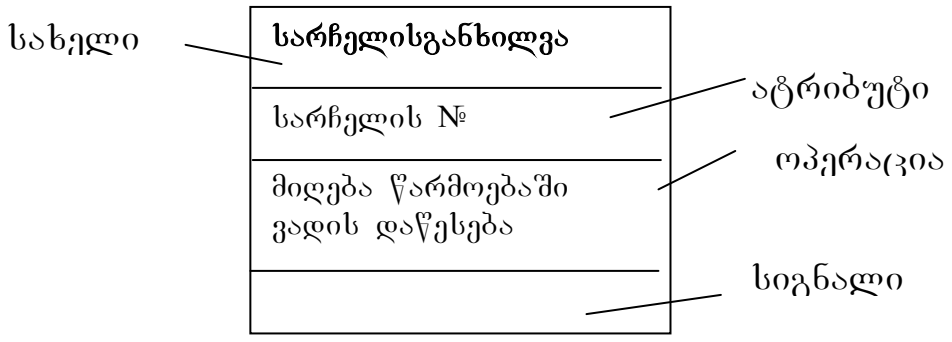
3.6. პროცესები და ძაფები

სხვადასხვა მოვლენები წარიმართებიან ერთდროულად. ამიტომ სისტემის მოდელირებისას, რომელიც განკუთვნილია რეალურ

სამყაროში სამუშაოდ, საჭიროა გავითვალისწინოთ მისი სახე პროცესების თვალსაზრისით, რომელშიც ძირითადი ყურადღება ეთმობა პროცესებს და ძაფებს, რომლებიც საფუძვლად უდევს პარალელიზმისა და სინქრონიზაციის მექანიზმებს.

UML-ში ყოველი დამოუკიდებელი მართვის ნაკადი მოდელირდება როგორც აქტიური ობიექტი, რომელიც აღწერს პროცესს ან ძაფს და აქვს უნარი მოახდინოს მმართველი ზემოქმედების ინიციირება. პროცესი (**Process**) – ეს რესურსებით უზრუნველყოფილი მართვის ნაკადია, რომელიც სრულდება სხვა პროცესების პარალელურად; ძაფი **(Thread**) – ეს შედარებით მცირე მართვის ნაკადია, რომელიც სრულდება სხვა ძაფების პარალელურად ერთ და იმავე პროცესის ფარგლებში.

აქტიური ობიექტი – ეს ობიექტია, რომელიც ფლობს პროცესს ან ძაფს და შეუძლია მმართველი ზემოქმედების ინიციირება. აქტიური კლასი – ეს კლასია, რომლის ეგზემპლიარები წარმოადგენენ აქტიურ ობიექტებს. გრაფიკულად იგი გამოიხატება შემდეგნაირად



პროცესები და ძაფები გამოისახებიან აქტიური კლასებით, ხოლო ურთიერთქმედების დიაგრამაზე ხშირად გვევლინებიან მიმდევრობის როლში.

მართვის ნაკადი. მიმდევრობით სისტემებში გვაქვს მხოლოდ ერთი მართვის ნაკადი. ეს ნიშნავს, რომ დროის ყოველ მომენტში

სრულდება ერთი და მხოლოდ ერთი მოქმედება. პარალელურ სისტემაში კი მართვის ნაკადები რამოდენიმეა, ე.ი. დროის ერთსა და იმავე მომენტში სრულდება სხვადასხვა მოქმედება. თითოეული ერთდროულად შესრულებადი მართვის ნაკადებიდან იწყება დამოუკიდებელ პროცესში ან ძაფში შესვლის წერტილიდან.

აქტიური კლასი გამოიყენება პროცესის წარმოსადგენად, რომლის კონტექსშიც სრულდება დამოუკიდებელი მართვის ნაკადი, რომელიც მუშაობს სხვების პარალელურად და რომელიც სარგებლობს მისი თანაბარი უფლებებით.

რეალურ პარალელიზმს შესაძლებელია მივაღწიოთ სამი საშუალებით: - გავანაწილოთ აქტიური ობიექტები სხვადასხვა კვანძებზე, - მოვათავსოთ აქტიური ობიექტები კვანძებზე რამოდენიმე პროცესორებით და ბოლოს მოვახდინოთ ორივე მეთოდის კომბინირება.

კლასები და მოვლენები. აქტიური კლასი წარმოადგენს დამოუკიდებელ მართვის ნაკადს, მაშინ როდესაც ჩვეულებრივი კლასი არ არის მასთან კავშირში. აქტიურებისგან განსხვავებით, ჩვეულებრივ კლასებს უწოდებენ პასიურებს, რადგან მათ არ აქვთ საშუალება მოახდინონ დამოუკიდებელი მართვის ნაკადის ინიციირება.

აქტიური კლასები გამოიყენებიან პროცესებისა და ძაფების მოდელირებისათვის. ტექნიკურად ეს ნიშნავს, რომ აქტიური ობიექტი – კლასის ეგზემპლიარი – ახდენს პროცესის ან ძაფის მატერიალიზებას. პარალელური სისტემების მოდელირებისას აქტიური ობიექტებით თქვენ ანიჭებთ სახელს ყოველ დამოუკიდებელ მართვის ნაკადს. როდესაც აქტიური ობიექტი იქმნება, გაიშვება მასთან ასოცირებული მართვის ნაკადი. როდესაც აქტიური ობიექტი ისპობა, ეს ნაკადი სრულდება.

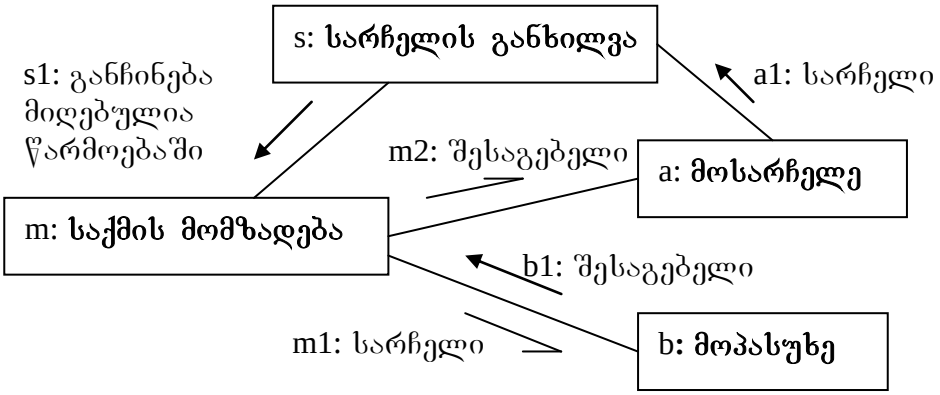
აქტიური კლასები ფლობენ იმავე თვისებებს, რასაც სხვა კლასები. მათ შესაძლებელია გააჩნდეთ ეგზემპლიარები, ატრიბუტები და ოპერაციები, ასევე მონაწილეობა მიიღონ დამოკიდებულების, განზოგადების და ასოციაციის მიმართებებში. ისინი შესაძლებელია რეალიზებულ იქნან კოოპერაციებით, ხოლო მათი ქცევა სპეციფიცირებულ იქნას ავტომატების მეშვეობით. დიაგრამებზე აქტიური ობიექტები გვხვდება ყველგან სადაც გვხვდება პასიურები.

კომუნიკაცია. ერთმანეთთან კოოპერირებადი ობიექტები ურთიერთქმედებენ შეტყობინებების გაცვლით. სისტემებში სადაც გვაქვს აქტიური და პასიური ობიექტები, მიზანშეწონილია განვიხილოთ ოთხი შესაძლო კომბინაცია.

პირველი, შეტყობინება შესაძლებელია გადაიცეს ერთი პასიური ობიექტიდან მეორეზე. იმის გათვალისწინებით, რომ დროის ნებისმიერ მომენტში არსებობს მხოლოდ ერთი მართვის ნაკადი, რომელიც გადის ორივე ობიექტზე, ასეთი ურთიერთქმედება შეესაბამება უბრალოდ ოპერაციის გამოძახებას.

მეორე, შეტყობინება შესაძლებელია გადაიცეს ერთი აქტიური ობიექტიდან მეორეზე. აქ ჩვენ საქმე გვაქვს პროცესებს შორის კომუნიკაციასთან, რომელიც შესაძლებელია განხორციელდეს ორი საშუალებით. პირველ ვარიანტში რომელიმე აქტიურ ობიექტს შეუძლია სინქრონულად გამოიძახოს მეორეს ოპერაცია. ასეთ საშუალებას აქვს სემანტიკა რანდევუ **(Rendezvous)**: გამომძახებელი ობიექტი მოითხოვს ოპერაციის შესრულებას და ელოდება, სანამ მიმღები მხარე მიიღებს გამოძახებას, შეასრულებს ოპერაციას და დააბრუნებს გარკვეულ ობიექტს (თუ ასეთი არის); შემდეგ ორივე ობიექტი აგრძელებენ მუშაობას ერთმანეთისაგან დამოუკიდებლად. გამოძახების შესრულების მთელი დროის განმავლობაში ორივე მართვის ნაკადი ბლოკირებულია. მეორე ვარიანტში ერთ აქტიურ

ობიექტს შეუძლია ასინქრონულად გაუგზავნოს სიგნალი მეორეს ან გამოიძახოს მისი ოპერაცია. ასეთი საშუალების სემანტიკა გვაგონებს საფოსტო ყუთს **(Mailbox**); გამომძახებელი მხარე აგზავნის სიგნალს ან იძახებს ოპერაციას, რის შემდეგ აგრძელებს შესრულებას. ამ დროს მიმღები მხარე იღებს სიგნალს ან გამოძახებას, როგორც კი იქნება ამისათვის მზად. სანამ ის ამუშავებს მოთხოვნას, ყველა ახლად შემოსული მოვლენები ან გამოძახებები დგებიან რიგში. მოახდენს რა რეაგირებას მოთხოვნაზე, მიმღები ობიექტი განაგრძობს თავის მუშაობას. საფოსტო ყუთის სემანტიკა მჟღავნდება იმაში, რომ ორივე ობიექტი არა სინქრონიზებულია, უბრალოდ ერთი უტოვებს შეტყობინებას მეორეს. **UML**-ში სინქრონული შეტყობინება გამოისახება მთლიანი ისრით, ხოლო ასინქრონული "ნახევარისრით" (იხ. ნახ.3.6.1).



ნახ.3.6.1.

მესამე, შეტყობინება შესაძლებელია გადაიცეს აქტიური ობიექტიდან პასიურზე. სიძნელე წარმოიშვება იმ შემთხვევაში, როდესაც ერთდროულად რამოდენიმე აქტიური ობიექტი გადასცემენ თავის მართვის ნაკადს ერთ და იმავე პასიურს. ასეთ შემთხვევაში საჭიროა ნაკადების სინქრონიზაციის ძალიან აკურატული მოდელირება, რომელსაც განვიხილავთ ქვევით.

მეოთხე, პასიურ ობიექტს შეუძლია გადასცეს შეტყობინება აქტიურს. თუ გავითვალისწინებთიმას, რომ ყოველი მართვის ნაკადი

ეკუთვნის რომელიმე აქტიურ ობიექტს, მაშინ ხდება ნათელი, რომ პასიური ობიექტის მიერ შეტყობინების გადაცემა აქტიურზე აქვს იგივე სემანტიკა, რაც შეტყობინებების გაცვლა ორ აქტიურ ობიექტს შორის.

სინქრონიზაცია. წარმოვიდგინოთ მრავალრიცხოვანი მართვის ნაკადები პარალელურ სისტემაში. როდესაც ნაკადი გადის რაიმე ოპერაციაზე, ჩვენ ვამბობთ, რომ ეს ოპერაცია არის შესრულების წერტილი. თუ ოპერაცია განსაზღვრულია რომელიმე კლასში, შეიძლება ითქვას, რომ შესრულების წერტილს წარმოადგენს ამ კლასის კონკრეტული ეგზემპლიარი. ერთ ოპერაციაში (შესაბამისად ერთ ობიექტში) შესაძლებელია ერთდროულად იმყოფებოდნენ რამოდენიმე მართვის ნაკადები, ასევე, ხდება ისე, რომ სხვადასხვა ნაკადები იმყოფებოდნენ სხვადასხვა ოპერაციებში, მაგრამ ერთ ობიექტში.

პრობლემა წარმოიშვება მაშინ, როდესაც ერთ ობიექტში იმყოფება ერთდროულად რამოდენიმე მართვის ნაკადი. თუ არ გამოვიჩენთ სიფრთხილეს, ნაკადებმა შეიძლება ხელი შეუშალონ ერთმანეთს, რაც მიგვიყვანს ობიექტის მდგომარეობის არაკორექტულ შეცვლამდე. ეს არის კლასიკური პრობლემა ურთიერთგამორიცხვისა. შეცდომები ასეთი სიტუაციების დამუშავებისას შეიძლება გახდეს სხვადასხვა სახის კონკურენციების მიზეზი ნაკადებს შორის.

ამ პრობლემის გადასაწყვეტად ობიექტ-ორიენტირებულ სისტემებში ოპერაციებზე, რომლებიც განსაზღვრულია კლასში, ენიჭება გარკვეული მასინქრონიზებელი თვისებები.

Sequential(მიმდევრობითი) – გამომძახებელი მხარე თავისი მოქმედების შესახებ კოორდინირებას უნდა ახდენდეს გამოსაძახებელ ობიექტში შესვლამდე, ისე რომ დროის ნებისმიერ მომენტში ობიექტის შიგნით იმყოფება ერთი მართვის ნაკადი.

რამოდენიმე მართვის ნაკადის არსებობისას ობიექტის სემანტიკისა და მთლიანობის გარანტია არ არის.

quarded(დაცული) - მართვის რამოდენიმე ნაკადისას ობიექტის სემანტიკა და მთლიანობა გარანტირებულია ობიექტის ყველა დაცულ ოპერაციაზე გამოძახებათა მოწესრიგების გზით. დროის ყოველ მომენტში ობიექტზე შესაძლებელია შესრულდეს მხოლოდ ერთი ოპერაცია.

Concurent(პარალელური) – მართვის რამოდენიმე ნაკადისას ობიექტის სემანტიკა და მთლიანობა გარანტირებულია იმით, რომ ოპერაცია განიხილება როგორც ატომური.

აქტიური ობიექტები თამაშობენ მნიშვნელოვან როლს სისტემის წარმოდგენისას პროცესების თვალსაზრისით. ასეთი წარმოდგენა მოიცავს პროცესებსა და ძაფებს, რომლებიც ქმნიან სისტემურ პარალელიზმსა და სინქრონიზაციას. ეს კი საშუალებას გვაძლევს გამოვსახოთ ასეთი წარმოდგენის სტატიკური და დინამიკური ასპექტები იმავე დიაგრამებით, რომლებიც გამოიყენება წარმოდგენისას პროექტირების თვალსაზრისით ე.ი. კლასების, ურთიერთქმედების, მოღვაწეობის და მდგომარეობის დიაგრამებით, იმ განსხვავებით რომ ძირითადი ყურადღება მათზე ეთმობა აქტიურ კლასებს, რომლებიც წარმოადგენენ პროცესებს და ძაფებს.

მართვის რამოდენიმე ნაკადი. სისტემის აგება რამოდენიმე მართვის ნაკადით – ადვილი ამოცანა არ არის. უნდა გადაწყდეს თუ როგორ გადავანაწილოთ სამუშაო პარალელურ აქტიურ ელემენტებს შორის, ასევე უნდა დადგინდეს კომუნიკაციისა და სინქრონიზაციის სწორი მექანიზმები სისტემის აქტიურ და პასიურ ობიექტებს შორის, რომელიც უზრუნველყოფს მათი ქცევის სისწორეს მართვის რამოდენიმე ნაკადის დროს.

მართვის რამოდენიმე ნაკადის მოდელირება ხდება შემდეგნაირად:

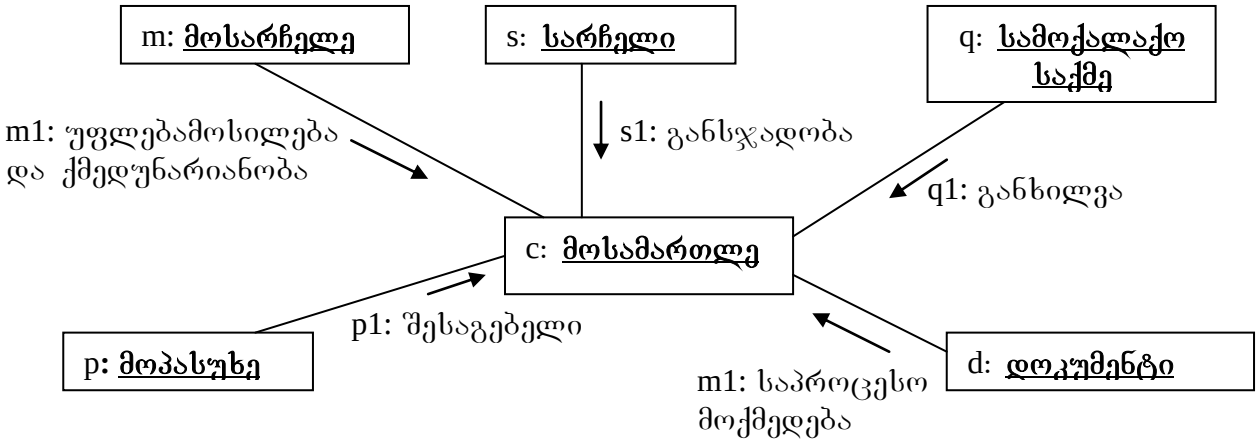
1. დავადგინოთ მოქმედებათა პარალელიზმის შესაძლებლობა და მოვახდინოთ მართვის ნაკადის მატერიალიზაცია აქტიური კლასის სახით. დავაჯგუფოთ აქტიური ობიექტების საერთო სიმრავლე აქტიურ კლასში.
2. განვიხილოთ მოვალეობების განაწილების ბალანსი აქტიურ კლასებს შორის, ხოლო შემდეგ განვიხილოთ, რომელ სხვა აქტიურ და პასიურ კლასებთან კოოპერირდება სტატიურად ყოველი მათგანი.
3. გამოხატეთ სტატიკური გადაწყვეტილებები კლასების დიაგრამის სახით, ნათლად გამოვყოთ აქტიური კლასები.
4. განიხილეთ, თუ როგორ კოოპერირდება დინამიურად თითოეული კლასი სხვა კლასებთან. გამოვხატოთ ეს გადაწყვეტილებები ურთიერთქმედების დიაგრამაზე. ნათლად მიუთითეთ აქტიური ობიექტები როგორც საწყისი წერტილები შესაბამისი მართვის ნაკადისა.
5. განსაკუთრებული ყურადღება მიაქციეთ აქტიურ ობიექტებს შორის კომუნიკაციებს. გამოიყენეთ საჭიროებისამებრ როგორც სინქრონული ასევე ასინქრონული შეტყობინებები.
6. ყურადღება მიაქციეთ აქტიური ობიექტების სინქრონიზაციას და იმ პასიურ ობიექტებს, რომლებთანაც ისინი კოოპერირდებიან. გამოიყენეთ ყველაზე მისადაგებული სემანტიკა – მიმდევრობითი, დაცული და პარალელური.

ნახ.3.6.2.-ზე მოყვანილია სამოქალაო სამართალწარმოების სისტემის ნაწილი პროცესების თვალთახედვით.

როგორც ნახაზიდან ჩანს რამოდენიმე ობიექტი ერთდროულად ურთიერთქმედებს ობიექტთან *მოსამართლე,* რომელსაც მივეცით სახელი c. შესაბამისად, c უნდა დავაპროექტოდ ისე, რომ მან შეინარჩუნოს თავისი სემანტიკა რამოდენიმე მართვის ნაკადისას.

პროცესებს შორის კომუნიკაცია. სისტემაში რამოდენიმე მართვის

ნაკადის ჩართვისას საჭიროა ასევე განვიხილოთ მექანიზმები, რომელთა მეშვეობით ობიექტები სხვადასხვა მართვის ნაკადიდან ურთიერთქმედებენ ერთმანეთთან. ინფორმაციის გასაცვლელად პროცესების საზღვრებიდან, ჩვეულებრივ გამოიყენება სხვა მექანიზმები.



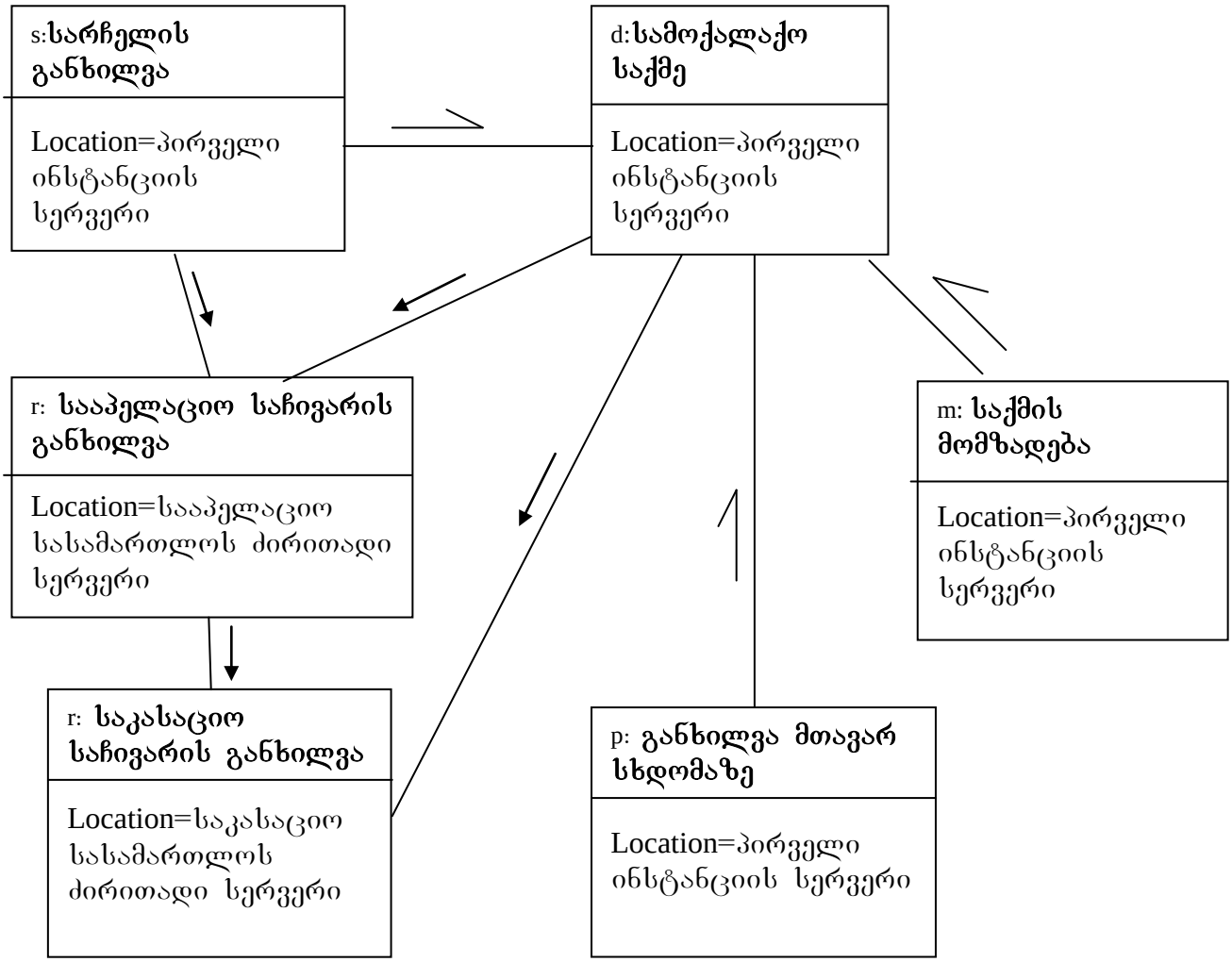
ნახ.3.6.2.

სირთულე პროცესებს შორის კომუნიკაციისა რთულდება კიდევ იმით, რომ განაწილებულ სისტემებში პროცესები შეიძლება სრულდებოდნენ სხვადასხვა კვანძებზე. არსებობს ორი კლასიკური მიდგომა პროცესებს შორის კომუნიკაციისა: შეტყობინების გადაცემა და დაშორებული პროცედურების გამოძახება. **UML-** ში ეს მექანიზმები მოდელირდებიან როგორც შესაბამისად სინქრონული და ასინქრონული მოვლენები.

პროცესებს შორის კომუნიკაციის მოდელირება ხდება შემდეგნაირად:

1. მოახდინეთ რამოდენიმე მართვის ნაკადის მოდელირება.
2. განიხილეთ ამ აქტიური ობიექტებიდან რომელი შეიძლება წარმოვადგინოთ პროცესების და რომელი ძაფების სახით. მათი განსხვავებისათვის, გამოვიყენოთ მისადაგებული სტერეოტიპი.

3. შეტყობინებების გაცვლა დავამოდელიროთ ასინქრონული, ხოლო დაშორებული პროცედურების – სინქრონული კომუნიკაციის მეშვეობით.



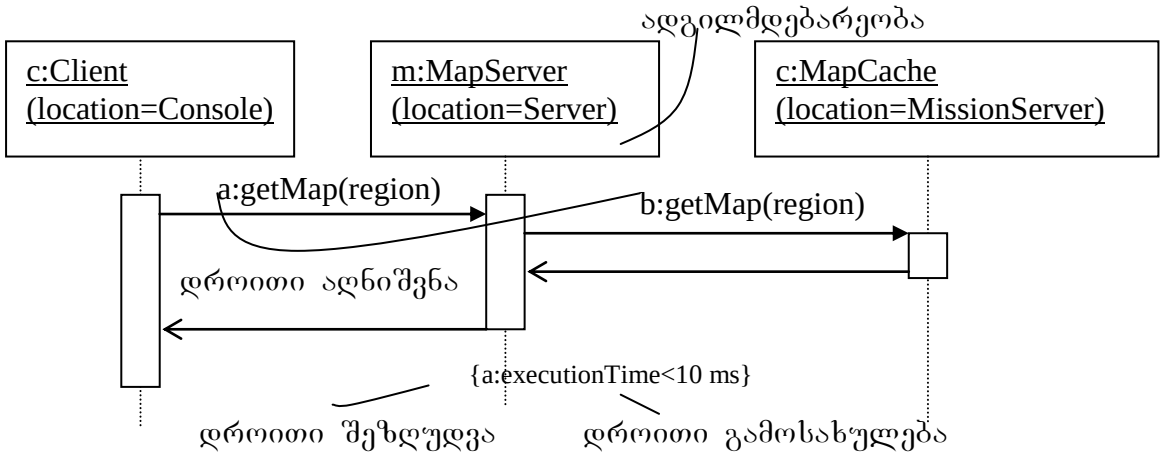
ნახ.3.6.4.

ნახ.3.6.4.-ზე მოყვანილია სამოქალაქო სამართალწარმოების განაწილებული სისტემის ნაწილი, რომელშიც პროცესები სრულდება სამ კვანძზე. როგორც ნახაზიდან ჩანს კომუნიკაცია ობიექტებს შორის *სარჩელის განხილვა, საქმის მომზადება, განხილვა მთავარ სხდომაზე* და *სამოქალაქო საქმე* ასინქრონულია, ხოლო *სამოქალაქო საქმე, სააპელაციო საჩივრის განხილვა* და *საკასაციო საჩივრის განხილვა* სინქრონული.

3.7. დრო და სივრცე

დროისა და სივრცის მოდელირება ერთ ერთი მნიშვნელოვანი ელემენტია ნებისმიერ განაწილებულ და რეალური დროის სისტემებში.

რეალური დროის სისტემებს უწოდებენ იმიტომ, რომ იგი თავის ფუნქციებს უნდა ასრულებდეს მკაცრად განსაზღვრულ აბსოლუტურ ან შეფარდებით დროის მომენტებში და ამაზე ხარჯავდეს წინასწარ განსაზღვრულ ან ხშირად შეზღუდულ დროს. არსებულ სისტემებს შორის არსებობს ისეთები, რომლებისთვისაც რეაქციის საჭირო დრო აღირიცხვება ნანო ან მილიწამებით. მაგრამ გვხვდება თითქმის რეალური დროის სისტემები, რომლებისთვისაც რეაქციისათვის დასაშვები დრო – წამები ან უფრო მეტიც არის.
განაწილებული სისტემების ქვეშ გაიგება ისეთი სისტემები, რომლის კომპონენტები შესაძლებელია განლაგდნენ სხვადასხვა კვანძებზე. კვანძები შეიძლება წარმოადგენდნენ სხვადასხვა პროცესორებს, დამონტაჟებული ერთ და იმავე კორპუსში, ან სხვადასხვა კომპიუტერებს, რომლებიც იმყოფებიან დედამიწის სხვადასხვა წერტილებში.



ნახ.3.7.1. დროითი შეზღუდვა და ადგილმდებარეობა

რეალური დროის სისტემების მოდელირების მოთხოვნილების დასაკმაყოფილებლად შემოტანილია გრაფიკული წარმოდგენა დროითი ნიშნულის, დროითი გამოსახულების, დროითი შეზღუდვის და ადგილმდებარეობის, როგორც ეს ნაჩვენებია ნახ.3.7.1.-ზე.

დროითი ნიშნული გამოიყენება დროის მომენტის აღნიშვნისათვის, რომელშიც მოხდა მოვლენა. იგი გამოისახება გამოსახულებით, დამოკიდებული სახელისაგან, რომელიც ენიჭება შეტყობინებას.

დროითი გამოსახულება ეს გამოსახულებაა, რომლის მნიშვნელობას წარმოადგენს აბსოლუტური ან შეფარდებითი დრო. *დროითი შეზღუდვა* ეს სემანტიკური მტკიცებაა შეფარდებითი ან აბსოლუტური დროის შესახებ. *მდებარეობა* ეს კომპონენტის განლაგებაა კვანძში.

დრო. რეალური დროის სისტემები, როგორც ეს დასახელებიდან გამომდინარეობს, მკაცრნი არიან დროის მიმართ. მოვლენები მათში შეიძლება წარმოებდეს რეგულარულად ან სპონტანურად, მაგრამ ნებისმიერ შემთხვევაში რეაქციის დრო მოვლენაზე უნდა იყოს განსაზღვრული ან აბსოლუტური ხანგრძლივობით, ან მოვლენის აღძვრის მომენტის მიმართ.

შეტყობინებების გადაცემა – ეს სისტემის ერთერთი დინამიური ასპექტია, ამიტომ სისტემის დროითი ფაქტორების მოდელირებისას შეიძლება ყოველ შეტყობინებას, რომლებიც მონაწილეობას ღებულობენ ურთიერთქმედებაში მივცეთ სახელი, რომელიც გამოიყენება როგორც დროითი ნიშნული.

დროითი ნიშნული – ეს გამოსახულებაა, რომელშიც გვხვდება შეტყობინების სახელი, რომელიც მონაწილეობას ღებულობს ურთიერთქმედებაში. თუ მოცემულია შეტყობინების სახელი, შეგვიძლია გამოვიყენოთ ნებისმიერი ამ შეტყობინების სამი ფუნქციიდან – **startTime, stopTime, executionTime**. თავის მხრივ, ეს

სამი ფუნქცია შეიძლება გამოვიყენოთ ნებისმიერი სირთულის დროითი გამოსახულებების აგებისათვის.

ადგილმდებარეობა. განაწილებული სისტემები თავისი ბუნებით შესდგებიან კომპონენტებისაგან, ფიზიკურად გაბნეულნი სხვადასხვა კვანძებზე. ბევრი სისტემებისათვის ადგილმდებარეობა (**Location**) კომპონენტების ფიქსირდება სისტემის დაყენების მომენტში. მაგრამ გვხვდება ისეთი სისტემებიც, რომლებშიც კომპონენტები მიგრირებენ ერთი კვანძიდან მეორეში.

ელემენტის ადილმდებარეობის მითითება შესაძლებელია ორი საშუალებით - მიუთითოდ კვანძის დამატებით განყოფილებაში ან ვისარგებლოთ მონიშნული მნიშვნელობით **Location** იმ კვანძის აღნიშვნისათვის რომელზეც განლაგდება კლასი.

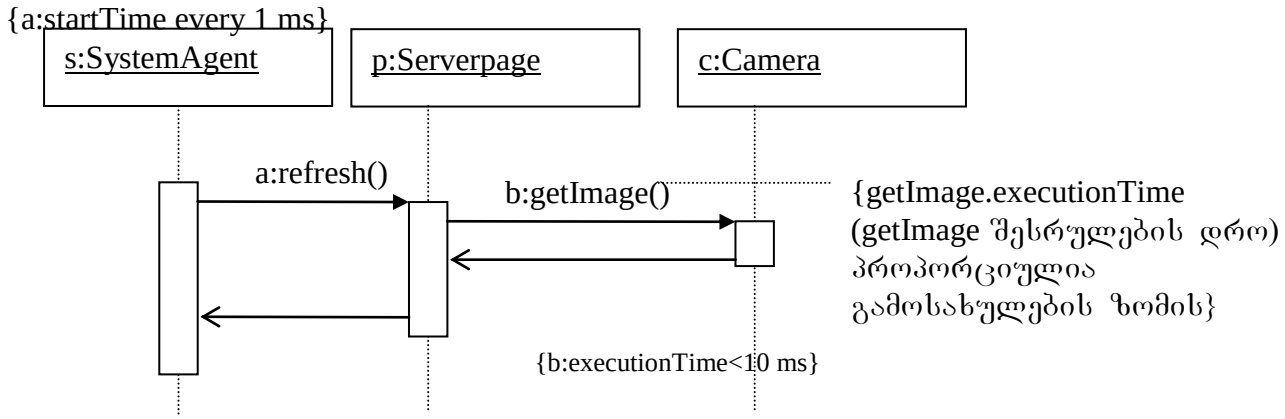
მოვლენის აბსოლუტური დრო, ინტერვალი მოვლენებს შორის და დრო, რომელიც საჭიროა მოქმედების შესრულებისათვის – აი სამი ძირითადი დროითი ასპექტი რეალური დროის სისტემებისათვის, რომელთა მოდელირებისას გამოიყენებიან დროითი შეზღუდვები.

დროითი შეზღუდვების მოდელირება ხორციელდება შემდგი სახით:

1. ყოველი მოვლენისათვის ურთიერთქმედებაში განვიხილოთ, იწყება იგი დროის განსაზღვრულ აბსოლუტურ მომენტში. დავამოდელიროთ ეს თვისება მოვლენაზე დროითი შეზღუდვის საშუალებით.
2. შეტყობინებების ყოველი ინტერესმქონე თანმიმდევრობისათვის ურთიერთქმედებაში განვიხილოთ, შეზღუდულია თუ არა მისი შესრულების დრო. დავამოდელიროთ ეს თვისება თანმიმდვრობაზე დროითი შეზღუდვის საშუალებით.

3. ყოველი დროში კრიტიკული ოპერაციისათვის განვიხილოთ მისი დროითი სირთულე. დავამოდელიროთ ეს სემანტიკა ოპერაციაზე დროითი შეზღუდვის საშუალებით.

მაგალითად, ქვევით მოყვანილ ნახაზზე მარცხენა შეზღუდვა ადგენს საწყის დროს განმეორებადი მოვლენის refresh გამოძახებისათვის. დროითი შეზღუდვა ნახაზის შუაში, ადგენს getImage გამოძახების მაქსიმალურ ხანგრძლივობას. ბოლოს, მარჯვენა შეზღუდვა ადგენს getImage გამოძახების მოვლენის დროით სირთულეს.



ნახ.3.7.2. დროითი შეზღუდვების მოდელირება

ობიექტების განაწილება. განაწილებული სისტემის ტოპოლოგიის მოდელირებისას მიზანშეწონილია განვიხილოთ ეგზემპლიარების როგორც კომპონენტების, ასევე კლასების ფიზიკური განლაგება. თუ ყურადღების ცენტრში არის გაშლილი სისტემის კონფიგურაციის მართვა, კომპონენტების განაწილების მოდელირება განსაკუთრებით მნიშვნელოვანია ისეთი ფიზიკური არსებების სპეციფიცირებისათვის, როგორიც არის შესრულებადი მოდულები, ბიბლიოთეკები და ცხრილები. თუ თქვენ უფრო გაინტერესებთ ფუნქციონალურობა, მასშტაბურობა და სისტემის

გამტარუნარიანობა, მაშინ უფრო მნიშვნელოვანია ობიექტთა განაწილების მოდელირება.

გადაწყვეტილება თუ როგორ გავანაწილოდ ობიექტები სისტემაში მნიშვნელოვანი პრობლემაა და მჭიდროდ არის დაკავშირებული პარალელიზმის საკითხებთან.

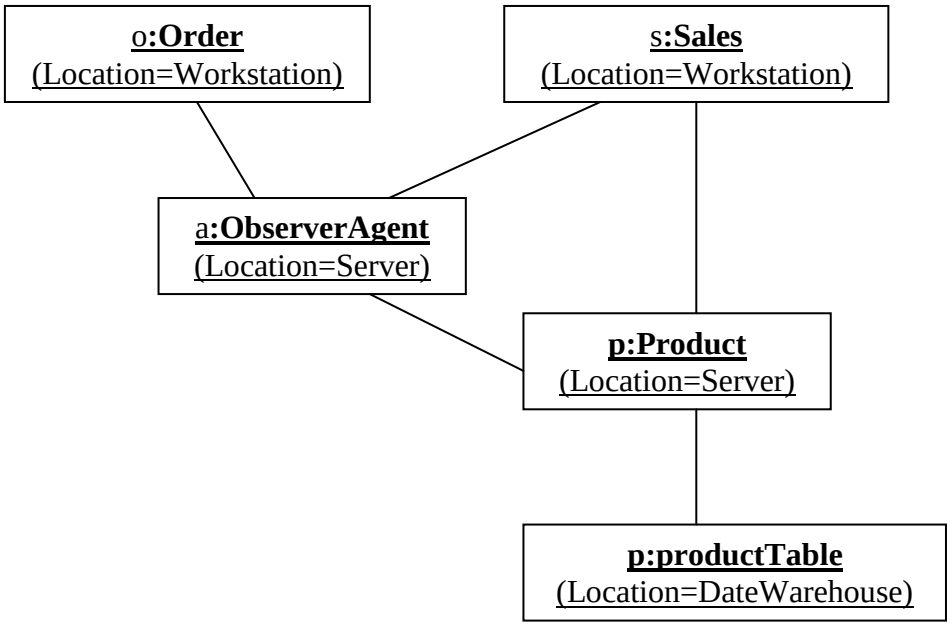
ობიექტების განაწილების მოდელირება შედგება შემდეგი ეტაპებისაგან:

1. ყოველი ინტერესმქონე კლასისათვის სისტემაში განვიხილოთ მიმართვების ლოკალურობა – სხვა სიტყვებით, გამოვავლინოთ ყველა მეზობლები და მათი ადგილმდებარეობა. ძლიერკავშირიანი ლოკალურობა ნიშნავს, რომ ლოგიკურად მეზობლური ობიექტები იმყოფებიან გვერდიგვერდ, ხოლო სუსტკავშირიანი – რომ ისინი ფიზიკურად დაშორებულნი არიან ერთმანეთისაგან. ეცადეთ განალაგოთ ობიექტები აქტიორების გვერდით, რომლებიც მანიპულირებენ მათზე.
2. შემდეგ განიხილეთ ტიპიური ურთიერთქმედებები ურთიერთდაკავშირებულ ობიექტთა სიმრავლეებს შორის. განალაგეთ ობიექტების სიმრავლე ურთიერთქმედების მაღალი ხარისხით გვერდიგვერდ, იმისათვის რომ შევამციროდ კომუნიკაციის ღირებულება. ობიექტები, რომლებიც სუსტად ურთიერთქმედებენ ერთმანეთს შორის, განვაცალკევოდ სხვადასხვა კვანძებზე.
3. შემდეგ განიხილეთ პასუხისმგებლობის განაწილება სისტემაში. გადავანაწილოდ ობიექტები ისე, რომ დავაბალანსოდ ყოველი კვანძის დატვირთვა.
4. არ დაგავიწყდეთ მომსახურების უსაფრთხოება, მობილურობა და ხარისხი და გავითვალისწინოთ ეს შეხედულებები ობიექტების განლაგების დროს.

5. გამოხატეთ ობიექტები დიაგრამაზე ერთი ორი შესაძლო საშუალებიდან:

- ჩართეთ ობიექტები უშუალოდ კვანძებში განლაგების დიაგრამაზე.
- ნათლად მიუთითედ ობიექტის მდგომარეობა მონიშნული მნიშვნელობის მეშვეობით.

ქვევით ნახაზზე მოყვანილია ობიექტების დიაგრამა, რომელიც ახდენს საცალო ვაჭრობის სისტემაში ობიექტების განაწილების მოდელირებას. ამ დიაგრამის ფასი იმაშია, რომ იგი საშუალებას იძლევა მოვახდინოთ ძირითადი ობიექტების ფიზიკური განლაგების ვიზუალირება.



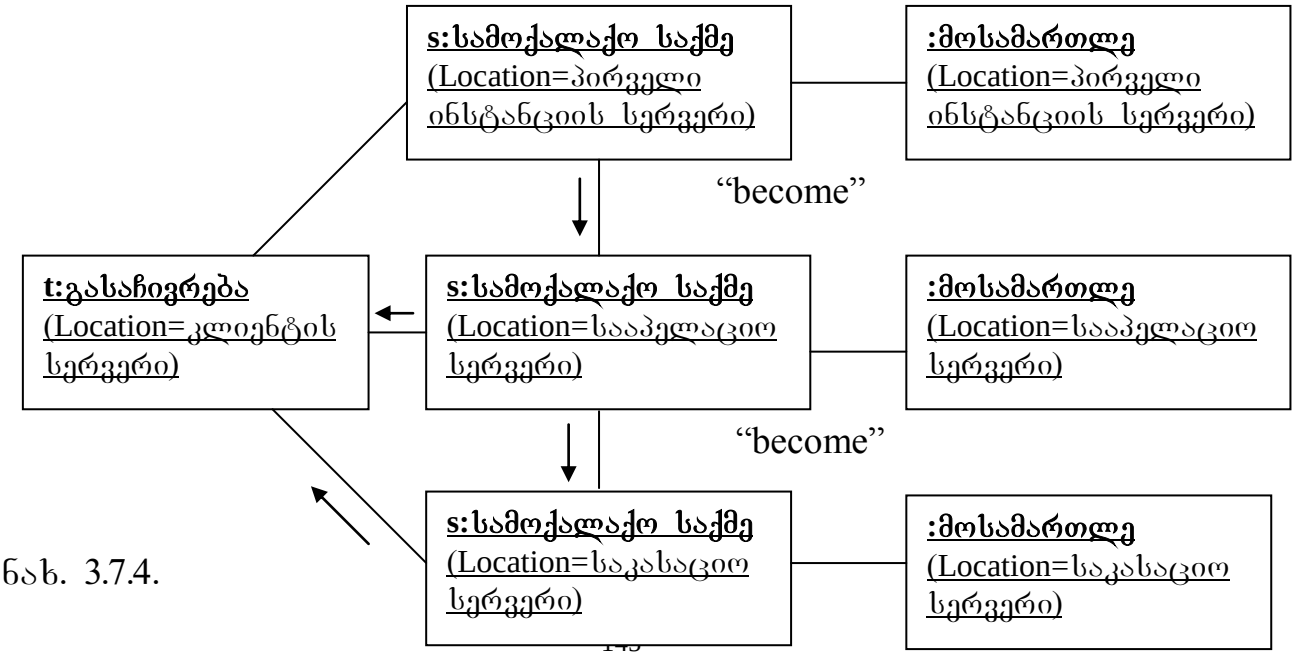
ნახ.3.7.3. ობიექტების განაწილების მოდელირება

როგორც ჩანს, ორი ობიექტი Order (დაკვეთა) და Sales (გაყიდვა) იმყოფებიან კვანძში Workstation (სამუშაო სადგური), ორი სხვა (ObserverAgent, დაკვირვების აგენტი, და Product, პროდუქტი) – კვანძში Server და ერთი (ProductTable, პროდუქტების ცხრილი) – კვანძში dataWarehouse (მონაცემების საცავი).

მიგრაციული ობიექტები. ბევრ განაწილებულ სისტემებში კომპონენტები და ობიექტები არ იცვლიან თავის ადგილმდებარეობას თავდაპირველი განლაგების შემდეგ. მაგრამ გვხვდება განაწილებული სისტემების კატეგორია, რომლებშიც სხვადასხვა არსებები გადაადგილდებიან.

პირველ რიგში, ობიექტები მიგრირებენ, რათა მიუახლოვდნენ აქტიორებს და სხვა ობიექტებს, რომლებთანაც ისინი კოოპერირდებიან სამუშაოს უკეთესი შესრულებისათვის. მაგალითად, სამოქალაქო სამართალწარმოების სისტემაში ***სამოქალაქო საქმე*** გადაადგილდება პირველი ინსტანციის სასამართლოდან სააპელაციოში და იქიდან შესაძლებელია საკასაციოში, თანსდევს რა მისსავე ფიზიკურ ანალოგს გასაჩივრების შესაბამისად.

მეორეს მხრივ, ობიექტები მიგრაციას განიცდიან ამა თუ იმ კვანძის მწყობრიდან გამოსვლის გამო ან დატვირთვის ბალანსირებისათვის სხვადასხვა კვანძებს შორის. ამიტომ სისტემის მდგრადობის უზრუნველსაყოფად ხდება ყველა ელემენტების გადაადგილება სხვა კვანძებზე. ამ დროს შესაძლებელია წარმადობის და გამტარუნარიანობის შემცირება, მაგრამ უზრუნველყოფილი იქნება მისი მდგრადი- ნორმალური მუშაობა.



ნახ. 3.7.4.

ნახ.3.7.4.-ზე მოყვანილია კოოპერაციის დიაგრამა, რომელიც წარმოადგენს სამოქალაქო საქმის მიგრაციას კვანძებს შორის. კლასის ***სამოქალაქო საქმე*** ეგზემპლიარი (სახელით **s)** მიგრაციას განიცდის კლიენტის გასაჩივრების შესაბამისად. გზაზე ეს ობიექტი ურთიერთქმედებს კლასი ***მოსამართლის*** ანონიმურ ეგზემპლიარებთან ყოველ კვანძზე და ბოლოს გასაჩივრების შედეგს აბრუნებს ობიექტზე ***გასაჩივრება*** , რომელიც განლაგებულია კვანძზე კლიენტის სერვერი.

ობიექტების მიგრაციის მოდელირება ხდება შემდეგნაირად:

1. აირჩიეთ მექანიზმი ობიექტების კვანძებს შორის ტრანსპორტირებისათვის.
2. წარმოადგინეთ დიაგრამაზე ობიექტის კვანძზე განლაგება, ნათლად მიუთითოდ მისი მდგომარეობა მონიშნული მნიშვნელობით.
3. სტერეოტოპული შეტყობინებებით become და copy წარმოადგინეთ ობიექტის გადაადგილება ახალ კვანძზე.
4. განიხილეთ სინქრონიზაციის საკითხები (კლონირებული ობიექტების კორექტული მდგომარეობის შენარჩუნება) და იდენტურობა (ობიექტის სახელის შენარჩუნება მისი გადაადგილებისას).

3.8. მდგომარეობათა დიაგრამები

მდგომარეობათა დიაგრამა გამოიყენება სისტემის დინამიური ასპექტების მოდელირებისათვის. ფაქტიურად იგი გვიჩვენებს ავტომატს. მის კერძო ნაირსახეობას წარმოადგენს მოღვაწეობის დიაგრამა, რომელშიც ყველა ან უმეტესი ნაწილი გადასვლებისა ინიციირდება მოღვაწეობის დამთავრების შედეგად საწყის მდგომარეობაში. მაშასადამე, ობიექტის სასიცოცხლო ციკლის მოდელირებისას სასარგებლოა როგორც მოღვაწეობის დიაგრამები, ისე მდგომარეობათა დიაგრამები. მაგრამ თუ მოღვაწეობის დიაგრამები გვიჩვენებენ მართვის ნაკადს მოღვაწეობიდან მოღვაწეობამდე, მდგომარეობათა დიაგრამაზე წარმოდგენილია მართვის ნაკადი მდგომარეობიდან მდგომარეობამდე.

მდგომარეობათა დიაგრამა გამოიხატება გრაფის სახით, მასზე ძირითადათ გამოისახება:

- მარტივი და შედგენილი მდგომარეობები;
- გადასვლები ასოციირებულ მოვლენებთან ან მოქმედებებთან ერთად.

მდგომარეობათა დიაგრამა შედგენილია ელემენტებისაგან, რომელიც გვხვდება ნებისმიერ ავტომატში და მათზე გამოიყენება ავტომატის ყველა მახასიათებლები. როგორც ყველა დიაგრამა, მდგომარეობათა დიაგრამაც შეიძლება შეიცავდეს შენიშვნებსა და შეზღუდვებს.

გამოყენების ტიპიური მაგალითები. როგორც ავღნიშნეთ მდგომარეობათა დიაგრამა გამოიყენება სისტემის დინამიური ასპექტების მოდელირებისათვის. უმეტესწილად ამაში იგულისხმება რეაქტიული ობიექტების ქცევის მოდელირება. რეაქტიულს უწოდებენ ისეთ ობიექტებს, რომელთა ქცევა ყველაზე კარგად გამოიხატება მისი რეაქციით საკუთარი კონტექსის გარეთ მომხდარ მოვლენებზე.

ყველაზე ხშირათ მდგომარეობათა დიაგრამებს იყენებენ რეაქტიული ობიექტების მოდელირებისათვის, განსაკუთრებით კლასების, პრეცენდენტების ან მთლიანად სისტემის. რეაქტიულ ობიექტს აქვს მკაფიოდ გამოხატული სასიცოცხლო ციკლი, როდესაც მიმდინარე ქცევა გამოწვეულია წარსულით. როგორც წესი რეაქტიული ელემენტები იმყოფებიან ლოდინის მდგომარეობაში, სანამ არ მიიღებს მოვლენას, ხოლო როდესაც ეს მოხდება მისი რეაქცია დამოკიდებულია წინმდებარე მოვლენებზე. მას შემდეგ რაც ობიექტი მოახდენს მასზე რეაგირებას, იგი ხელახლა გადადის შემდეგი მოვლენის ლოდინის მდგომარეობაში. ასეთი ობიექტებისათვის ინტერესს წარმოადგენს პირველ რიგში მდგრადი მდგომარეობები, მოვლენები რომლებიც ახდენენ გადასვლის ინიციირებას ერთი მდგომარეობიდან მეორეში და მოქმედებები, რომლებიც სრულდება მდგომარეობის შეცვლისას.

რეაქტიული ობიექტების მოდელირებისას უნდა მოვახდინოთ ძირითადათ სამი საგნის სპეციფიცირება: მდგრადი მდგომარობები, რომელშიც შესაძლებელია იმყოფებოდეს ობიექტი, მოვლენები, რომლებიც ახდენენ გადასვლების ინიციირებას და მოქმედებები, რომლებიც სრულდება მდგომარეობის ყოველი შეცვლისას. რეაქტიული ობიექტების მოდელირება გულისხმობს მთელი მისი სასიცოცხლო ციკლის მოდელირებას, დაწყებული შექმნის მომენტიდან და დამთავრებული მისი მოსპობით, განსაკუთრებული აქცენტით მდგრად მდგომარეობებზე, რომლებშიც შეიძლება იმყოფებოდეს ობიექტი.

მდგრადი მდგომაეობა ეს ისეთი მდგომარეობაა, რომელშიც ობიექტი იმყოფება განუსაზღვრელად დიდი დროის განმავლობაში. მოვლენებს შეუძლიათ აგრეთვე გადასვლების ინიციირება თავის თავში და მოახდინონ შიდა გადასვლები, როდესაც საწყისი და მიზნობრივი მდგომარეობები ერთმანეთს ემთხვევა. მოვლენებზე ან

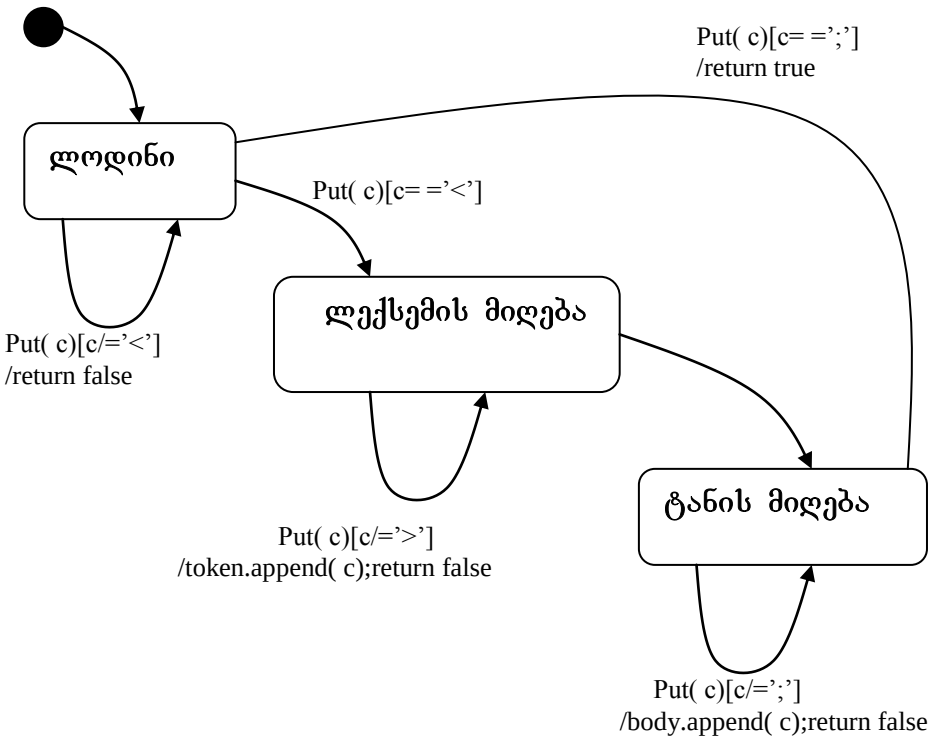
მდგომარეობის ცვლილებაზე რეაქციისას ობიექტმა შეიძლება შეასრულოს გარკვეული მოქმედება.

რეაქტიული ობიექტის ქცევის მოდელირებისას ჩვენ გვეძლევა საშუალება მოვახდინოთ მოქმედების სპეციფიცირება, მივაბამთ რა მას გადასვლასთან ან მდგომარეობის შეცვლასთან. ლიტერატურაში ავტომატი, რომლის ყველა მოქმედება მიბმულია გადასვლებთან, უწოდებენ **მილის** მანქანას, ხოლო ავტომატი რომლის მოქმედებები მიბმულია მდგომარეობებთან, **მურის** მანქანას. მდგომარეობათა დიაგრამის დამუშავებისას ჩვეულებრივ გამოიყენება **მილის** და **მურის** მანქანების კომბინაცია.

რეაქტიული ობიექტის მოდელირება შედგება შემდეგი პროცესებისაგან:

1. ავირჩიოთ ავტომატისათვის კონტექსი – კლასი, პრეცენდენტი ან სისტემა მთლიანობაში.
2. ავირჩიოთ ობიექტისათვის საწყისი და საბოლაო მდგომარეობები.
3. დაადგინეთ ობიექტის მდგრადი მდგომარეობები, ისეთი რომლებშიც ის შეიძლება იმყოფებოდეს განუსაზღვრელად დიდი დროის განმავლობაში. დაიწყეთ ზედა დონის მდგომარეობებით, ხოლო შემდეგ გადადით ქვემდგომარეობებზე.
4. მოახდინეთ მდგრადი მდგომარეობების მოწესრიგება ობიექტის მთელი სასიცოცხლო ციკლის განმავლობაში.
5. დაადგინეთ რომელ მოვლენებს შეუძლიათ მდგომარეობებს შორის გადასვლების ინიციირება. წარმოვადგინოთ ეს მოვლენები როგორც გადასვლების ტრიგერები.
6. დაუკავშირეთ მოქმედებები გადასვლებს(მილის მანქანა) და/ან მდგომარეობებს(მურის მანქანა).

7. განიხილეთ როგორ შეიძლება ავტომატის გამარტივება ქვემდგომარეობების, განშტოებების, შერწყმის და ისტორიული მდგომარეობებით.
8. გასინჯეთ, რომ ნებისმიერი მდგომარეობის მიღწევა შესაძლებელია მოვლენათა გარკვეული კომბინაციისას.
9. დარწმუნდით ჩიხური სიტუაციების არ არსებობაში, ანუ ისეთების რომლებიდანაც არ არის გადასვლა მოვლენის არც ერთი კომბინაციისას.
10. შეამოწმეთ ავტომატი ხელით ან ინსტრუმენტალური საშუალებით და დაადგინეთ როგორ იქცევა ის მოვლენათა მოსალოდნელი თანმიმდევრობისას და მათზე რეაქციისას.



ნახ.3.8.1.

ნახ.3.8.1.-ზე მოყვანილია მდგომარეობათა დიაგრამა მარტივი კონტექსურად თავისუფალი ენის გარჩევისათვის. ასეთი ენის მაგალითები შეიძლება ვნახოთ სისტემებში, რომლებშიც შემავალი ან გამომავალი ნაკადი შეადგენენ **XML**- შეტყობინებები. ასეთ

შემთხვევაში პროექტირდება ავტომატი სიმბოლოთა ნაკადის გარჩევისათვის, რომელიც დააკმაყოფილებს ენის სინტაკს:

შეტყობინება: ’ < ’ სტრიქონი ‘ > ’ შეტყობინება ‘ ; ‘

პირველი სტრიქონი წარმოადგენს ტეგს, მეორე – შეტყობინების ტანია. მოცემული სიმბოლოების ნაკადიდან ამოირჩევა მხოლოდ შეტყობინებები, რომლებიც აკმაყოფილებენ სინტაკსის წესებს.

ნახაზიდან ჩანს, რომ ავტომატისათვის გათვალისწინებულია სამი მდგრადი მდგომარეობა: **ლოდინი, ლექსემის მიღება** და **ტანის მიღება**. მდგომარეობათა დიაგრამა დაპროექტებულია მილის მანქანის სახით – მოქმედებები მიბმულია გადასვლებთან. სინამდვილეში ავტომატში არის მხოლოდ ერთი მოვლენა, რომელიც იმსახურებს ინტერესს – გამოძახება **put** ფაქტიური პარამეტრით **c** (სიმბოლოთი). მდგომარეობაში **ლოდინი** ავტომატი უკუაგდებს ყველა სიმბოლოს, რომლებიც არ შეესაბამებიან ლექსემის დასაწყისს (ეს სპეციფიცირებულია დამცავი პირობით). ლექსემის დასაწყისის აღმოჩენისას ობიექტის მდგომარეობა იცვლება **ლექსემის მიღებაზე**. იმყოფება რა ამ მდგომარეობაში, ავტომატი ინარჩუნებს ყველა სიმბოლოს, რომლებიც არ წარმოიდგინებიან როგორც ლექსემის დასასრული (ესეც სპეციფიცირებულია დამცავი პირობით). აღმოაჩენს რა ლექსემის დასასრულს, ობიექტი გადადის მდგომარეობაში **ტანის მიღება**. ამ მდგომარეობაში ავტომატი ინარჩუნებს ყველა სიმბოლოს, რომლებიც არ წარმოიდგინებიან როგორც შეტყობინების დასასრული (იხ. დამცავი პირობა). როგორც კი მიიღება შეტყობინების დასასრული, ობიექტის მდგომარეობა იცვლება მდგომარეობით **ლოდინი**. ბრუნდება მნიშვნელობა, რომელიც გვიჩვენებს, რომ შეტყობინება გარჩეულია და ავტომატი მზად არის შემდეგის მისაღებათ.

ყურადღება უნდა მივაქციოთ იმ გარემოებას, რომ ეს დიაგრამა აღწერს ავტომატს, რომელიც მუშაობს უწყვეტად – მასში არ არის საბოლაო მდგომარეობა.

ნახ.3.8.2.-ზე მოყვანილია მდგომაეობათა დიაგრამა სამოქალაქო სამართალწარმოების ერთერთი პრეცენდენტისათვის- "სარჩელის განხილვა". ერთერთი ძირითადი მოთხოვნა სამოქალაქო სამართალწარმოებისას შეტანილ სარჩელზე დროული რეაგირება და მისი განხილვაა დადგენილ ვადებში (ეს ვადა შეადგენს მარტივი საქმეებისათვის 2 თვეს და რთული საქმეებისათვის 5 თვეს). მოცემული მოთხოვნის შესრულება მოითხოვს ოპერატიულად ვიცოდეთ საქმის მიმდინარე მდგომარეობა და ის საპროცესო მოქმედებები, რომლებიც მოცემულ მდგომარეობაში უნდა იყოს გამოყენებული,

მოყვანილი დიგრამა აღწერს ავტომატს, რომლის საწყისია სარჩელის შემოტანა სასამართლოში, ხოლო მიზნობრივი მდგომარეობაა ***სარჩელი მიღებულია წარმოებაში.***

ქვემოთ მოყვანილია თითოეული გადასლის შესაბამისი მოვლენა-ტრიგერი, დამცავი პირობა და მოქმედება, რომელიც თან ახლავს ამ გადასვლას. მოცემულ შემთხვევაში ეს მოქმედებები წარმოადგენენ განჩინებებს, რომლებიც აფიქსირებენ სარჩელის ერთი მდგომარეობიდან მეორეში გადასვლას.

1. გადაეცა მოსამართლეს(5) [(X1 ^ X2 ^ X3) V (T-T0 > 5)] / განჩინება1

2. გადაეცა მოსამართლეს(5) [(X1 V X2 V X3) ^ (T-T0 < 5)] / განჩინება2

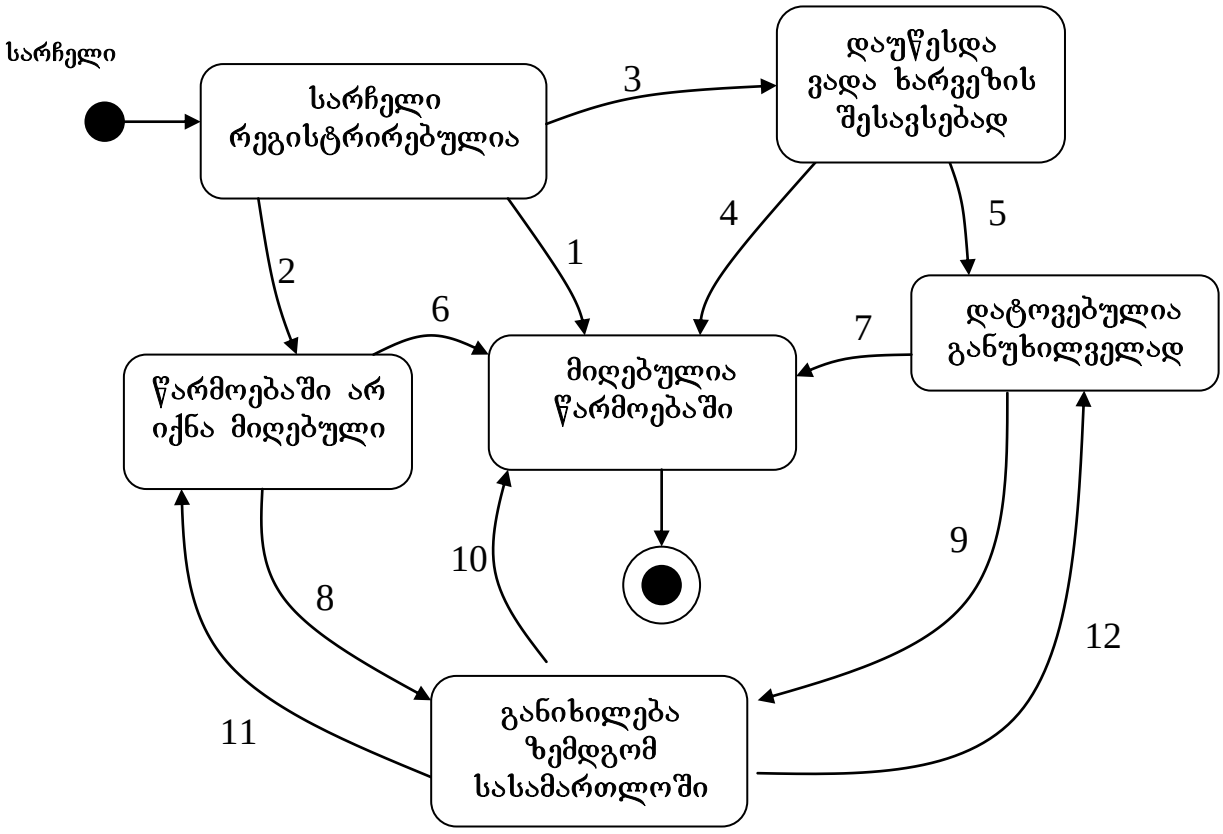
3. გადაეცა მოსამართლეს(5) [(X1 V X2 V X3) V (T-T0 < 5)] / განჩინება3

სადაც,

X1 – სარჩელის შინაარსი აკმაყოფილებს საქართველოს სამოქალაქო კოდექსის(სსკ) 178-ე მუხლს;

X2 – სახელმწიფო ბაჟი შეტანილია(სსკ-ის 178-ე მუხლი);

X3 – სრულდება სსკ-ის 186-ე მუხლი.



ნახ.3.8.2.

X3=X31 ^ X32 ^ X33 ^ X34 ^ X35 ^ X36 ^ X37 ;

X31 – სარჩელი არ ექვემდებარება სასამართლო უწყებას:

X32 - არსებობს სასამართლო გადაწყვეტილება ან განჩინება მოსარჩელის მიერ სარჩელზე უარის თქმის, მოპასუხის მიერ სარჩელის ცნობის ან მხარეთა მორიგების დამტკიცების შესახებ;

X33 – ამ ან სხვა სასამართლოს წარმოებაშია საქმე დავაზე იმავე მხარეებს შორის, იმავე საგანზე და იმავე საფუძვლით;

X34 – მხარეებს დადებული აქვთ ხელშეკრულება, რომ მათ შორის დავა გადასაწყვეტად გადაეცეს კერძო არბიტრაჟს;

X35 – საქმე ამ სასამართლოს განსჯადი არ არის;

X36 – სარჩელი შეიტანა ქმედუუნარო პირმა;

X37 – დაინტერესებული პირის სახელით განცხადება შეიტანა პირმა, რომელსაც არა აქვს უფლებამოსილება საქმის წარმოებაზე.

T – მიმდინარე თარიღი;

T0 – მოსამართლეზე სარჩელის გადაცემის თარიღი.

შემოსული სარჩელი თავდაპირველად ხვდება მდგომარეობაში ***რეგისტრაცია***, რომლის შემდეგ სარჩელი გადაეცემა მოსამართლეს, სწორედ აღნიშნული მოვლენა წარმოადგენს მოვლენა-ტრიგერს, რომელიც ინიციირებას უწევს მომდევნო სამ მდგომარეობაში გადასვლას – ***დაუწესდა ვადა ხარვეზის შესავსებათ***, ***მიღებულია წარმოებაში*** და ***უარი ეთქვა წარმოებაში მიღებაზე***. თითოეულ მათგანს გააჩნია დამცავი პირობა და მოქმედება, რომელიც მიბმულია შესაბამის გადასვლებთან (მილის მანქანა). მოსამართლეს განჩინების გამოტანისათვის გამოყოფილი აქვს 5 დღიანი ვადა, რაც მითითებულია მოვლენა- ტრიგერის პარამეტრში. განხილვის ვადის გადაჭარბების შემთხვევაში სარჩელი ნაკლოვანებების მიუხედავად ითვლება წარმოებაში მიღებულად. სწორედ ეს ფაქტი არის მითითებული 1-ლი გადასვლის დამცავ პირობაში.

მდგომარეობიდან ***დაუწესდა ვადა ხარვეზის შესავსებათ*** გადასვლით 4 (ხარვეზი შევსილია [T1 < T2] / განჩინება4) ხდება გადასვლა მდგომარეობაში ***მიღებულია წარმოებაში***, ხოლო გადასვლით 5 (ხარვეზი არ არის შევსილია [T1 < T2] / განჩინება5) მდგომარეობაში ***დატოვებულია განუხილველად.***

მდგომარეობებში ***უარი ეთქვა წარმოებაში მიღებაზე, დატოვებულია განუხილველად*** დასაშვებია კერძო საჩივრის შეტანა, რომელიც შეტანილი უნდა იყოს 12 დღის ვადაში და მას იხილავს იგივე სასამართლო. თუ სარჩელი დაკმაყოფილდა ამუშავდება 6(7) გადასვლა:

6. კერძო საჩივარი (12) & დაკმაყოფილდა [Ť < 12] / განჩინება6

7. კერძო საჩივარი (12) & დაკმაყოფილდა [Ť < 12] / განჩინება7

აქ Ť – თი აღნიშნულია კერძო საჩივრის შეტანის თარიღი. იმ შემთხვევაში თუ საჩივარი არ დაკმაყოფილდება ამუშავდება 8(9) გადასვლა, რითაც მოხდება გადასვლა მდგომარეობაში ***განიხილება ზემდგომ სასამართლოში***.

8. კერძო საჩივარი (12) & არ დაკმაყოფილდა [Ť < 12] / განჩინება8

9. კერძო საჩივარი (12) & არ დაკმაყოფილდა [Ť < 12] / განჩინება9

აღნიშნულ მდგომარეობას გააჩნია მოქმედება შესვლისა და გამოსვლისას:

entry. საჩივარი(სარჩელი) გადაიგზავნოს 5 დღის ვადაში ზემდგომ სასამართლოში;

exit. სარჩელი და მიღებული განჩინება საჩივარზე უბრუნდება სასამართლოს.

როგორც ნახაზიდან ჩანს თუ ზემდგომ სასამართლოში საჩივარი დაკმაყოფილდა ამუშავდება გადასვლა 10, წინააღმდეგ შემთხვევაში 11(12) გადასვლები:

10. კერძო საჩივარი დაკმაყოფილდა / განჩინება10

11. კერძო საჩივარი არ დაკმაყოფილდა / განჩინება11

12. კერძო საჩივარი არ დაკმაყოფილდა / განჩინება12

ასეთივე წესით აგებულ იქნა მდგომარეობათა დიაგრამები სამოქალაქო სართალწარმოების სხვა პრეცენდენტებისათვის და ბოლოს მთლიანად სისტემისათვის, რომელმაც გამოყენება პოვა სასამართლო სისტემის მართვის ავტომატიზებულ სისტემაში.

თავი 4
არქიტექტურული მოდელირების საფუძვლები

იმისათვის, რომ შეგვეძლოს ვიმსჯელოთ სისტემის სასურველ ქცევაზე, ვქმნით პრეცედენტების დიაგრამას. საპრობლემო სფეროს ლექსიკონს აღვწერთ კლასების დიაგრამის მეშვეობით. იმისათვის, რომ ვუჩვენოთ ლექსიკონში აღწერილი არსებები როგორ მუშაობენ ერთობლივად საჭირო ქცევის უზრუნველსაყოფად, ვსარგებლობთ მიმდევრობის, კოოპერაციის, მდგომარეობათა და მოღვაწეობის დიაგრამებით. საბოლაოდ ლოგიკური ნახაზები გარდაიქმნებიან რეალურ საგნებში, მაგალითად შესრულებად პროგრამებში, ბიბლიოთეკებში, ცხრილებში, ფაილებში და დოკუმენტებში.

ავტომატიზებული სისტემის არქიტექტურის პროექტირებისას საჭირო ხდება განვიხილოთ როგორც ლოგიკური, ისე მისი ფიზიკური ასპექტები. ლოგიკურ ელემენტებს განეკუთვნებიან ისეთი არსებები, როგორიც არის კლასები, ინტერფეისები, კოოპერაციები, ურთიერთქმედებები და ავტომატები, ხოლო ფიზიკურს – კომპონენტები (ლოგიკური არსებების ფიზიკური დაჯგუფება) და კვანძები (აპარატურა, რომელზედაც განლაგდებიან და სრულდებიან კომპონენტები).

4.1. კომპონენტები

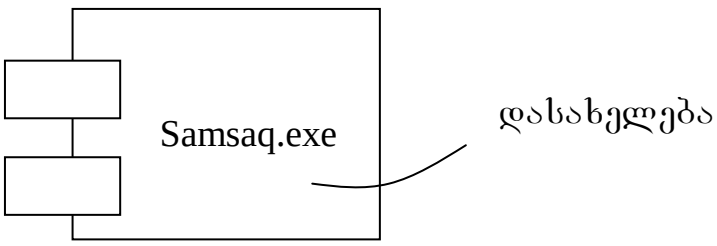
კომპონენტები გამოიყენებიან ფიზიკური არსებების მოდელირებისათვის. მათ მიეკუთვნებიან შესრულებადი მოდულები, ბიბლიოთეკები, ცხრილები, ფაილები და დოკუმენტები. ჩვეულებრივ კომპონენტი წარმოადგენენ ლოგიკური ელემენტების ფიზიკურ წყობას, ისეთების როგორიც არის კლასები, ინტერფეისები და კოოპერაციები.

ლოგიკური მოდელირება სრულდება საპრობლემო სფეროს ლექსიკონისა და სხვადასხვა არსებების ურთიერთქმედების აღწერისათვის. ფიზიკური მოდელირება გამოიყენება შესრულებადი სისტემის აგებისათვის. მაშინ როდესაც ლოგიკური არსებები იმყოფებიან მხოლოდ კონცეპტუალურ სამყაროში, ფიზიკურები განლაგდებიან ფიზიკურ კვანძებში და შეიძლება შესრულდნენ უშუალოდ, ან ირიბად მონაწილეობდნენ სისტემის მუშაობაში.

ამრიგად, ყველა ფიზიკური არსებები გამოისახებიან კომპონენტების სახით. კომპონენტი – ეს ფიზიკური არსებაა, რომელიც ახდენს გარკვეული ინტერფეისების რეალიზებას. მაშასადამე, ინტერფეისები ქმნიან ხიდს ლოგიკურ და ფიზიკურ მოდელებს შორის. მაგალითად, ლოგიკურ მოდელში შესაძლებელია რომელიმე კლასის ინტერფეისის სპეციფიცირება, რომელიც შემდეგ რეალიზებულ იქნება გარკვეული კომპონენტით.

მრავალი ოპერაციული სისტემა და დაპროგრამების ენები უშუალოდ ეთანხმებიან კომპონენტის მცნებას. ობიექტური ბიბლიოთეკები, შესრულებადი პროგრამები, კომპონენტები **COM+**, **Enterprise JavaBeans** – ყველა ეს არსებების მაგალითებია. კომპონენტები ასევე შესაძლებელია გამოყენებულ იქნან მუშა სისტემის ისეთი ელემენტების წარმოდგენისათვის როგორიც არის ცხრილები, ფაილები და დოკუმენტები.

ამრიგად, კომპონენტი ეს სისტემის ფიზიკური შესაცვლელად დასაშვები ელემენტია, რომელიც შეთავსებულია ინტერფეისების ერთ გარკვეულ ნაკრებთან და რომელიც უზრუნველყოფს რომელიმე სხვას რეალიზებას. გრაფიკულად კომპონენტი გამოისახება შემდეგი სახით



ბევრი მიმართებით კომპონენტები მსგავსია კლასების. ერთსაც და მეორესაც გააჩნია სახელი, შეუძლიათ ინტერფეისების რეალიზება, შევიდნენ დამოკიდებულების, განზოგადების და ასოციაციის მიმართებებში, მიიღონ მონაწილეობა ურთიერთქმედებაში. მაგრამ კომპონენტებსა და კლასებს შორის არის არსებითი განსხვავებაც:

- კლასები წარმოადგენენ ლოგიკურ აბსტრაქციებს, ხოლო კომპონენტები – ფიზიკურ არსებებს. მაშასადამე, კომპონენტები შესაძლებელია განლაგდნენ კვანძებში, ხოლო კლასები – არა.
- კომპონენტები წარმოადგენენ ლოგიკური არსებების ფიზიკურ წარმოსახვას და შესაბამისად, იმყოფებიან აბსტრაქციის სხვა დონეზე.
- კლასებს შეიძლება გააჩნდეთ ატრიბუტები და ოპერაციები. კომპონენტები ფლობენ მხოლოდ ოპერაციებს, რომლებიც მისაღწევია მათი ინტერფეისებიდან.

მნიშვნელოვანია მიმართება კომპონენტსა და ინტერფეისს შორის. ინტერფეისი ეს კლასების ან კომპონენტების მიერ წარმოდგენილი ოპერაციების ნაკრებია, რომლებიც აღწერენ მომსახურებას. ოპერაციული სისტემების ყველა პოპულარული კომპონენტური საშუალებები (როგორიც არის **COM+, CORBA და Enterprise JavaBeans**) იყენებენ ინტერფეისს სხვადასხვა კომპონენტების "დასაკავშირებლად".

ინტერფეისი, რომელიც რეალიზდება კომპონენტით, უწოდებენ ექსპორტირებულ ინტერფეისს. ეს ნიშნავს, რომ კომპონენტი მოცემული ინტერფეისიდან წარუდგენს რიგ მომსახურებებს სხვა კომპონენტებს. კომპონენტს შეუძლია ბევრი ინტერფეისების ექსპორტირება. ინტერფეისი, რომლითაც კომპონენტი სარგებლობს უწოდებენ იმპორტირებულს. ეს ნიშნავს, რომ კომპონენტი შეთავსებულია ინტერფეისთან და დამოკიდებულია მისგან თავისი

ფუნქციების შესრულებისას. კომპონენტს შეუძლია სხვადასხვა ინტერფეისების იმპორტირება, ამასთან მისთვის დასაშვებია ერთდროულად ექსპორტირება და იმპორტირება.

კონკრეტული ინტერფეისი შეიძლება ექსპორტირებული იყოს ერთი კომპონენტით და იმპორტირებული მეორეს მიერ. თუ ორ კომპონენტს შორის განლაგებულია ინტერფეისი მათი უშუალო ურთიერთ დამოკიდებულება იყოფა. კომპონენტი, რომელიც იყენებს მოცემულ ინტერფეისს, იფუნქციონირებს კორექტულად (განურჩევლად იმისა, რომელი კომპონენტით იქნება რეალიზებული ინტერფეისი). ცხადია, კომპონენტი შეიძლება გამოვიყენოთ გარკვეულ კონტექსში მხოლოდ მაშინ, როდესაც ყველა მისი იმპორტირებული ინტერფეისები ექსპორტირებულნი არიან რომელიღაც სხვა კომპონენტებით.

მთავარი ამოცანა ყოველი კომპონენტ - ორიენტირებული საშუალებისა ნებისმიერ ოპერაციულ სისტემაში - უზრუნველყოს დანართის აწყობის შესაძლებლობა შემცვლელი ორობითი ნაწილებისაგან. ეს ნიშნავს, რომ თქვენ შეგიძლიათ შექმნათ სისტემა კომპონენტებისაგან, ხოლო შემდეგ განავითაროთ ის, დაამატოთ რა ახალი კომპონენტები ან შეცვალოთ ძველი – დამატებითი კომპილაციის გარეშე. ამის მიღწევა შესაძლებელია ინტერფეისების მეშვეობით. მოვახდნთ რა ინტერფეისების სპეციფიცირებას შეგიძლიათ ჩასვათ უკვე მუშა სისტემაში მუშა კომპონენტი, რომელიც შეთავსებულია ამ ინტერფეისთან ან წარმოადგენს მას. სისტემა შესაძლებელია გავაფაროვოთ ახალი კომპონენტების ჩართვით, რომლებიც უზრუნველყოფენ ახალ მომსახურებას დამატებითი ინტერფეისების მეშვეობით, ასევე კომპონენტებით, რომლებსაც უნარი აქვთ გამოიცნონ და გამოიყენონ ეს ახალი ინტერფეისები. ასეთი სემანტიკა ხსნის თუ რა დგას კომპონენტის განსაზღვრის უკან. კომპონენტი ეს ფიზიკური

შესაცვლელი ნაწილია სისტემის, რომელიც შეთავსებულია ერთ ინტერფეისთან და რეალიზებას უწევს სხვებს. ყოველივე ზემოთ მოყვანილი მსჯელობა გვაძლევს საფუძველს გამოვყოთ კომპონენტების დამახასიათებელი ნიშნები:

- პირველ რიგში, კომპონენტს გააჩნია ფიზიკური ბუნება, იგი არსებობს რეალურად და არა კონცეფციის დონეზე.
- მეორე, კომპონენტი შეცვლას ექვემდებარება. ერთი კომპონენტის მაგიერ შესაძლებელია ჩაისვას მეორე, თუ იგი შეთავსებულია ინტერფეისების იმავე ნაკრებთან. ჩვეულებრივ დამატებისა და შეცვლის მექანიზმი კომპონენტისა შესრულებადი სისტემის ფორმირების მიზნით მომხმარებლისათვის გამჭვირვალეა და უზრუნველყოფილი ხდება ან ობიექტური მოდელებით (როგორიც არის **COM+ და Enterprise JavaBeans),** რომლებიც ხშირად სრულებით არ მოითხოვენ გარე ჩარევას, ან ინსტრუმენტალური საშუალებებით, რომლებიც მოცემული მექანიზმის ავტომატიზაციას ახდენენ.
- მესამე, კომპონენტი ეს სისტემის ნაწილია. კომპონენტი იშვიათად გამოდის სხვებისაგან დამოუკიდებლად: ჩვეულებრივ იგი მუშაობს სხვა კომპონენტებთან ერთად და ცხადია, მიესადაგება იმ არქიტექტურულ და ტექნოლოგიურ კონტექსს, რომლისთვისაც განკუთვნილია. კომპონენტი წარმოადგენს ლოგიკურად და ფიზიკურად შეძლებულს დაკავშირებისათვის, ანუ წარმოადგენს მნიშვნელოვან სტრუქტურულ და/ან ქცევით ფრაგმენტს გარკვეული სისტემის. კომპონენტი შესაძლებელია ხელმეორედ გამოვიყენოთ სხვადასხვა სისტემებში. მაშასადამე, კომპონენტები წარმოადგენენ ფუნდამენტალურ სამშენებლო ბლოკებს, რომლებისგან აიწყობა სისტემები. ეს განსაზღვრა რეკურსიულია - სისტემა, რომელიც განიხილება აბსტრაქციის ერთ დონეზე, შეიძლება იყოს მხოლოდ კომპონენტი უფრო მაღალ დონეზე.

- მეოთხე, როგორც აღნიშნული იყო, კომპონენტი შეთავსებულია ერთ ინტერფეისების ნაკრებთან და რეალიზებას უწევს მეორე ნაკრებს.

გამოყოფენ სამი სახის კომპონენტებს:

1. **განლაგების კომპონენტები**, რომლებიც აუცილებელია და საკმარისი შესრულებადი სისტემის აგებისათვის. მათ რიცხვს მიეკუთვნება დინამიურად დამაკავშირებელი ბიბლიოთეკები (**DLL**) და შესრულებადი პროგრამები (**EXE**). UML-ში განსაზღვრულია კომპონენტების ფართო სპექტრი, რომელიც მოიცავს როგორც კლასიკურ ობიექტურ მოდულებს **COM+, CORBA და Enterprise JavaBeans,** ისე ალტერნატიულებს დინამიურ Web-გვერდებს, მონაცემთა ბაზის ცხრილებს და შესრულებად მოდულებს, სადაც გამოიყენება კომუნიკაციის დახურული მექანიზმები.
2. **კომპონენტები – მუშა პროდუქტები**. ეს დამუშავების პროცესის გვერდითი შედეგია. მას შეიძლება მივაკუთვნოთ ფაილები პროგრამის საწყისი ტექსტებით და მონაცემებით, რომლებისგანაც იქმნება განლაგების კომპონენტები. ასეთი კომპონენტები არ ღებულობენ უშუალო მონაწილეობას შესრულებადი სისტემის მუშაობაში, მაგრამ წარმოადგენენ სამუშაო პროდუქტს, რომლებისგანაც იქმნება შესრულებადი სისტემა.
3. **შესრულების კომპონენტები**. ისინი იქმნებიან როგორც სისტემის მუშაობის შედეგი. მაგალითად შეიძლება მოვიყვანოთ ობიექტი **COM+**, რომლის ეგზემპლიარი იქმნება **DLL**- დან.

კომპონენტები შესაძლებელია დაჯგუფებულ იქნას პაკეტში და მათ შორის შესაძლებელია დამოკიდებულების, განზოგადების, ასოციაციის და რეალიზაციის მიმართებები.

UML-ში განსაზღვრულია ხუთი სტანდარტული სტერეოტიპი კომპონენტებთან მიმართებაში:

- **executable**(შესრულებადი) – განსაზღვრავს კომპონენტს, რომელიც შესაძლებელია შესრულდეს კვანძში.
- **Library**(ბიბლიოთეკა) – განსაზღვრავს სტატიკურ ან დინამიურ ობიექტურ ბიბლიოთეკას;
- **table**(ცხრილი) - განსაზღვრავს კომპონენტს, რომელიც წარმოადგენს მონაცემთა ბაზის ცხრილს.
- **file**(ფაილი) - განსაზღვრავს კომპონენტს, რომელიც წარმოადგენს დოკუმენტს, საწყისი ტექსტით და მონაცემებით.
- **document**(დოკუმენტი) - განსაზღვრავს კომპონენტს, რომელიც წარმოადგენს დოკუმენტს.

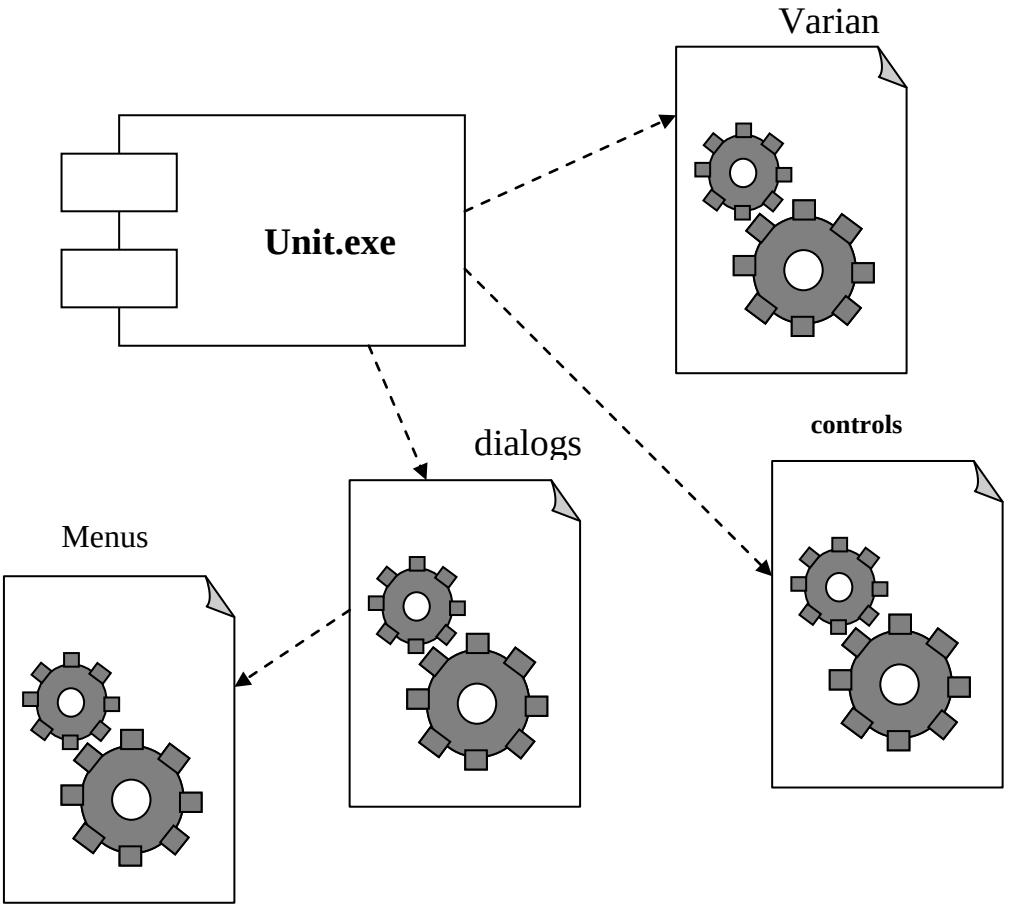
კომპონენტები ყველაზე ხშირად გამოიყენებიან განლაგების კომპონენტების მოდელირებისათვის, რომლებიც შეადგენენ სისტემის რეალიზაციას. ეს განსაკუთრებით ეხება ისეთ სისტემებს, რომლებიც შედგებიან რამოდენიმე კომპონენტისა და მასთან ასოცირებული ობიექტური ბიბლიოთეკებისაგან. ასეთ შემთხვევაში კომპონენტების მოდელირება სასარგებლო იქნება სისტემის ფიზიკური სეგმენტაციის თვალსაზრისით. კომპონენტების მოდელირების მნიშვნელობა კიდევ უფრო იზრდება, თუ სისტემის განვითარებასთან ერთად საჭირო ხდება მისი სხვადასხვა ნაწილების კონფიგურაციის მართვა და ვერსიების კონტროლი. ასეთი სახის გადაწყვეტილებებზე გავლენას ახდენს:

- რიგი ტექნიკური ფაქტორებისა, მაგალითად ოპერაციულ სისტემაში არსებული კომპონენტური საშუალებების შერჩევა;
- კონფიგურაციის მართვის საკითხები – სისტემის რომელი ნაწილები შეიძლება შეიცვალოს დროთა განმავლობაში;
- ხელმეორედ გამოყენების შესაძლებლობა - რომელი კომპონენტები შეიძლება იქნას გამოყენებული სხვა სისტემებში და პირიქით.

შესრულებადი პროგრამების და ბიბლიოთეკების მოდელირებისას ფიზიკური სისტემა უნდა დავყოთ ნაწილებათ. მხედველობაში უნდა მივიღოთ ტექნიკური ფაქტორები და ასევე შეხედულებები კონფიგურაციის და ხელმეორედ გამოყენების მართვის თვალსაზრისით. დავამოდელიროთ შესრულებადი პროგრამები და ბიბლიოთეკები როგორც კომპონენტები, გამოვიყენოთ რა შესაბამისი სტანდარტული ელემენტები. თუ რეალიზაციისათვის დაგვჭირდება ახალი კომპონენტები, შემოვიტანოთ შესაბამისი ახალი სტერეოტიპები. სისტემის მნიშვნელოვანი დამაკავშირებელი კვანძების მართვისათვის, დავამოდელიროთ ინტერფეისები, რომელთა რეალიზება უნდა მოახდინონ გარკვეულმა კომპონენტებმა, ხოლო სხვებმა - გამოიყენონ. დავამოდელიროთ მიმართებები შესრულებად პროგრამებს, ბიბლიოთეკებსა და ინტერფეისებს შორის. მაგალითისათვის ნახ. 4.1.-ზე მოყვანილია კომპონენტების ნაკრები სამოქალაქო სამართალწარმოების მართვისათვის. დიაგრამა შედგება ერთი შესრულებადი პროგრამისაგან (**Unit.exe**) და ბიბლიოთეკებისაგან (**Variants, Controls, Menus, Dialogs**). კომპონენტების გამოსახვისათვის გამოყენებულია სტანდარტული ელემენტები შესრულებადი პროგრამებისა და ბიბლიოთეკებისათვის.

დიდი სისტემებისათვის, რომლებიც რამოდენიმე კომპიუტერზე იმართება, დაგვჭირდება კომპონენტების განაწილების საშუალებების მოდელირება, დაუნიშნოთ რა თითოეულს კვანძი, რომელზედაც ის განლაგდება.

გარდა შესრულებადი პროგრამების და ბიბლიოთეკებისა არსებობს კომპონენტები, რომლებიც მნიშვნელოვანია სისტემის განლაგებისათვის.



ნახ.4.1.

მათ მიეკუთვნება მონაცემთა ფაილები, ოპერატიული კარნახის ფაილები, ინიცირებადი ფაილები და სისტემის დაყენებისა ან დახურვის ფაილები. ამ კომპონენტების მოდელირებაც სისტემის კონფიგურაციის მართვის მნიშვნელოვანი შემადგენელი ნაწილია.

4.2. განლაგება

კომპონენტები, რომელსაც ვიყენებთ, უნდა განლაგდნენ რომელიმე აპარატურაზე, წინააღმდეგ შემთხვევაში ისინი ვერ შესრულდებიან. ავტომატიზებული სისტემა ამიტომაც შედგება პროგრამული და აპარატული უზრუნველყოფისაგან.

ავტომატიზებული სისტემის არქიტექტურის პროექტირებისას საჭირო ხდება განვიხილოთ როგორც ლოგიკური, ისე მისი

ფიზიკური ასპექტები. ლოგიკურ ელემენტებს განეკუთვნებიან ისეთი არსებები, როგორიც არის კლასები, ინტერფეისები, კოოპერაციები, ურთიერთქმედებები და ავტომატები, ხოლო ფიზიკურს – კომპონენტები (ლოგიკური არსებების ფიზიკური დაჯგუფება) და კვანძები (აპარატურა, რომელზედაც განლაგდებიან და სრულდებიან კომპონენტები).

კვანძი – ფიზიკური ელემენტია, რომელიც არსებობს შესრულების დროს და წარმოადგენს გამოთვლით რესურს, რომელიც ჩვეულებრივ ფლობს როგორც მინიმუმ მეხსიერების გარკვეულ მოცულობას, ხოლო ხშირათ ასევე პროცესორსაც. გრაფიკულად კვანძი გამოისახება კუბის სახით. ყოველ კვანძს გააჩნია სახელი, რომელიც წარმოადგენს ტექსტურ სტრიქონს.

ობიექტების ან კომპონენტების სიმრავლე, რომლებიც მიწერილია კვანძზე როგორც ჯგუფზე, უწოდებენ განაწილების ელემენტს. კვანძებს შორის ყველაზე გავრცელებული მიმართებაა ასოციაცია. მოცემულ კონტექსში ასოციაცია წარმოადგენს კვანძების ფიზიკურ შეერთებას, მაგალითად ხაზი **Ethernet**, მიმდევრობითი არხი ან განმანცალკავებელი შინა. ასოციაცია შეიძლება გამოვიყენოთ თანამგზავრული ტიპის არაპირდაპირი შეერთების მოდელირებისას ორ დაშორებულ პროცესორს შორის.

კვანძები ძირითადათ გამოიყენებიან:

- პროცესორებისა და მოწყობილობების მოდელირებისათვის, რომლებიც ქმნიან ავტონომიური, კლიენტ-სერვერული და განაწილებული სისტემების ტოპოლოგიას.
- სისტემაში შემავალი კომპონენტების ფიზიკური განაწილების ვიზუალირებისა და სპეციფიცირებისათვის პროცესორებისა და კვანძების მიხედვით.

პროცესორი ეს კვანძია, რომელსაც შეუძლია მონაცემების დამუშავება, ანუ შეასრულოს კომპონენტი. მოწყობილობა ეს კვანძია,

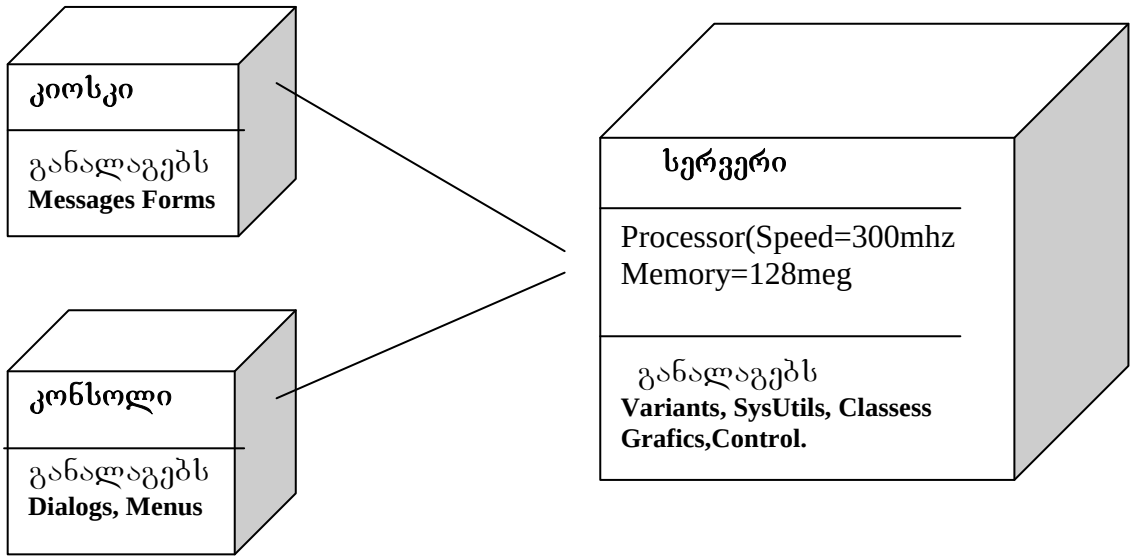
რომელსაც არ შეუძლია მონაცემების დამუშავება და საერთო ჯამში გამოიყენება რეალურ სამყაროსთან დაკავშირებული რაიმეს წარმოდგენისათვის.

პროცესორებისა და მოწყობილობების მოდელირებისას უნდა მოვახდინოთ გამოთვლითი ელემენტების იდენტიფიცირება სისტემის განლაგების თვალთახედვით წარმოდგენისას. თუ ეს ელემენტები წარმოადგენენ საერთო სახის პროცესორებსა და მოწყობილობებს მიუწეროთ მათ შესაბამისი სტანდარტული სტერეოტიპები. მაგრამ, თუ ეს პროცესორები და მოწყობილობები შედიან საპრობლემო სფეროს ლექსიკონში, შეუთავსეთ მათ შესაბამისი სტერეოტიპები. ისევე როგორც კლასების მოდელირებისას, დავადგინოთ ატრიბუტები და ოპერაციები, გამოყენებული თითოეული კვანძისათვის.

სისტემების ტოპოლოგიის მოდელირებისას ხშირათ სასარგებლოა კომპონენტების სისტემის შემადგენლობაში შემავალი პროცესორებსა და მოწყობილობებზე ფიზიკური განაწილების ვიზუალირება და სპეციფიცირება. ეს განსაკუთრებით შეეხება განაწილებულ სისტემებს, რათა გამოვრიცხოთ კომპონენტების განლაგების დუბლირების შესაძლებლობა. საკმაოდ გავრცელებულია შემთხვევა, როდესაც ერთი და იგივე კომპონენტები განლაგებულნი არიან ერთდროულად რამოდენიმე კვანძზე. ამიტომ სასურველია მივაწეროთ ყოველი მნიშვნელოვანი კომპონენტი შესაბამის კვანძს ან ჩამოვთვალოთ კომპონენტები, განლაგებულნი კვანძზე, დამატებით განყოფილებაში.

ნახ.4.2.-ზე წარმოდგენილია დიაგრამა, რომლებზეც მოყვანილია შესრულებადი პროგრამები განლაგებულნი კვანძების მიხედვით. სერვერი – ეს კვანძია სტერეოტიპით პროცესორი საერთო დანიშნულებით, კიოსკი და კონსოლი – კვანძებია სტერეოტიპით სპეციალიზირებული პროცესორები. ყოველი პროცესორისათვის

გამოყოფილია დამატებითი განყოფილება, მათზე განლაგებული კომპონენტების აღნიშვნისათვის. ამასთან სერვერი წარმოდგენილია სიჩქარის და მეხსიერების ატრიბუტებით.



ნახ.4.2.

აუცილებელი არ არის კომპონენტები კვანძების მიხედვით განაწილებული იყვნენ სტატიურად. შესაძლებელია კომპონენტების ერთი კვანძიდან მეორეში დინამიური მიგრაციის მოდელირება.
ბევრ განაწილებულ სისტემებში კომპონენტები და ობიექტები არ იცვლიან თავის ადგილმდებარეობას თავდაპირველი განლაგების შემდეგ. მაგრამ გვხვდება განაწილებული სისტემების კატეგორია, რომლებშიც სხვადასხვა არსებები გადაადგილდებიან.

პირველ რიგში, ობიექტები მიგრირებენ, რათა მიუახლოვდნენ აქტიორებს და სხვა ობიექტებს, რომლებთანაც ისინი კოოპერირდებიან სამუშაოს უკეთესი შესრულებისათვის. მაგალითად, სამოქალაქო სამართალწარმოების სისტემაში ***სამოქალაქო საქმე*** გადაადგილდება პირველი ინსტანციის სასამართლოდან სააპელაციოში და იქიდან შესაძლებელია საკასაციოში, თანსდევს რა მისსავე ფიზიკურ ანალოგს გასაჩივრების შესაბამისად (იხ.§ 3.7.).

მეორეს მხრივ, ობიექტები მიგრაციას განიცდიან ამა თუ იმ კვანძის მწყობრიდან გამოსვლის გამო ან დატვირთვის

ბალანსირებისათვის სხვადასხვა კვანძებს შორის. ამიტომ სისტემის მდგრადობის უზრუნველსაყოფად ხდება ყველა ელემენტების გადაადგილება სხვა კვანძებზე. ამ დროს შესაძლებელია წარმადობის და გამტარუნარიანობის შემცირება, მაგრამ უზრუნველყოფილი იქნება მისი მდგრადი- ნორმალური მუშაობა.

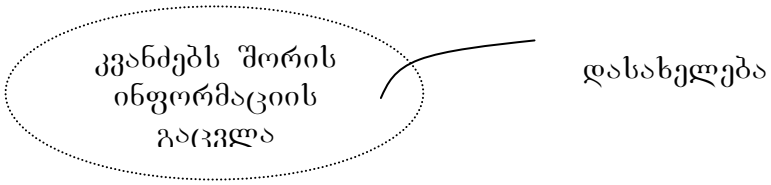
4.3. კოოპერაციები

სისტემის არქიტექტურის კონტექსში კოოპერაცია საშუალებას იძლევა მივანიჭოთ სახელი გარკვეულ კონცეპტუალურ ფრაგმენტს, რომელიც მოიცავს როგორც სტატიკურ, ისე დინამიურ ასპექტებს. კოოპერაციას უწოდებენ კლასების, ინტერფეისების და სხვა ელემენტების ერთობლიობას, რომლებიც მუშაობენ ერთობლივად კოოპერაციული ეფექტის უზრუნველსაყოფად, უფრო მნიშვნელოვანის, ვიდრე შემადგენელთა ჯამია.

კოოპერაციები გამოიყენება პრეცედენტებისა და ოპერაციების რეალიზაციის აღწერისათვის და სისტემის არქიტექტურულად - მნიშვნელოვანი მექანიზმების მოდელირებისათვის. კარგად სტრუქტურირებული სისტემების ბევრი ელემენტი სხვადასხვა კომბინაციაში მონაწილეობას ღებულობს სხვადასხვა მექანიზმების ფუნქციონირებაში. მექანიზმების მოდელირება ხდება კოოპერაციის მეშვეობით. კოოპერაცია წარმოადგენს სისტემის კონცეპტუალურ სამშენებლო ბლოკებს, რომლებიც შეიცავენ როგორც სტრუქტურულ, ისე ქცევით ელემენტებს. მაგალითად, შეიძლება განვიხილოთ ინფორმაციის მართვის განაწილებული სისტემა, რომლის მონაცემთა ბაზა განლაგდება რამოდენიმე კვანძზე. მომხმარებლის თვალთახედვით განახლება წარმოდგება როგორც ერთიანი ელემენტარული ოპერაცია. მაგრამ, თუ შევხედავთ შიგნიდან, ყველაფერი ასე მარტივად არ გამოჩნდება, რამდენადაც

განახლებაში მონაწილეობს რამოდენიმე მანქანა. სიმარტივის ილუზიის შექმნისათვის აუცილებელია შემოვიტანოთ ტრანზაქციის მექანიზმი, რომლის მეშვეობით კლიენტს საშუალება ექნება მიანიჭოს სახელი გარკვეულ ოპერაციას, რომელიც წარმოიდგინება ერთიანი ელემენტალური ოპერაციით, მიუხედავად იმისა, რომ რამოდენიმე მონაცემთა ბაზას ეხება. ასეთი მექანიზმების მუშაობაში შესაძლებელია მონაწილეობა მიეღო რამოდენიმე კოოპერირებულ კლასს, რომლებიც ერთობლივად უზრუნველყოფენ ტრანზაქციის სემანტიკას. ბევრი მათგანი ჩართული იქნება სხვა მექანიზმებშიც, მაგალითად ინფორმაციის მდგრადი შენახვის უზრუნველსაყოფად. ასეთი კლასების ერთობლიობა (სტრუქტურული შემადგენელი), აღებული ურთიერთქმედებით მათ შორის (ქცევითი შემადგენელი), ქმნის მექანიზმს, რომელსაც წარმოადგენს კოოპერაცია. კოოპერაცია არა მარტო ასახელებს სისტემურ მექანიზმებს, არამედ გამოიყენება პრეცედენტებისა და ოპერაციების რეალიზებისათვის.

გრაფიკულად კოოპერაცია გამოისახება ელიფსის სახით წყვეტილი საზღვრით.



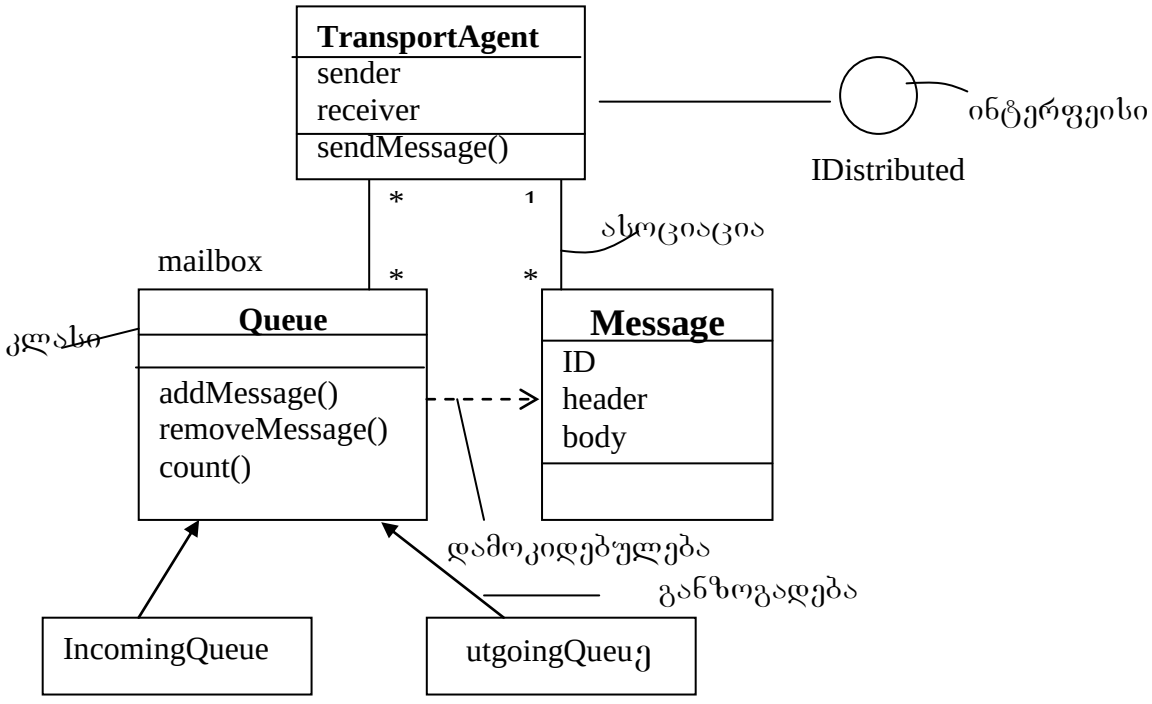
ნახ.4.3.

როგორც ავღნიშნეთ, კოოპერაციას გააჩნია ორი ასპექტი – სტრუქტურული და ქცევითი. სტრუქტურული შემადგენელი აღწერს კლასებს, ინტერფეისებს და სხვა ერთობლივად მომუშავე ელემენტებს. ქცევითი შემადგენელი კი აღწერს ამ ელემენტების ურთიერთქმედების დინამიკას.

კოოპერაციის სტრუქტურული შემადგენელი შეიძლება შეიცავდეს კლასებს, ინტერფეისებს, კომპონენტებს და კვანძებს. სტრუქტურული ელემენტების აღწერისათვის შეიძლება გამოვიყენოთ სტრუქტურული მოდელირების მთელი სპექტრი. ამავე დროს კოოპერაცია თვითონ არ

ფლობს არც ერთ სტრუქტურულ ელემენტს, იგი მხოლოდ მიმართავს კლასებს, ინტერფეისებს, კომპონენტებს და კვანძებს, რომლებიც გამოცხადებულნი არიან სხვა ადგილზე, ან იყენებენ მათ. ამიტომ კოოპერაცია ასახელებს სისტემური არქიტექტურის კონცეპტუალურ და არა ფიზიკურ ფრაგმენტს. უფრო მეტიც, ერთმა და იმავე ელემენტმა შეიძლება მონაწილეობა მიიღოს რამოდენიმე კოოპერაციაში.

თუ გვაქვს კოოპერაცია, რომელიც ასახელებს სისტემის კონცეპტუალურ ფრაგმენტს, შეგვიძლია გავხსნათ იგი და ვნახოთ მისი დამალული სტრუქტურული დეტალები. მაგალითად, ნახ.4.4.-ზე ნაჩვენებია, რომ კოოპერაციის ***კვანძებს შორის შეტყობინებები*** გახსნისას დგინდება კლასები, რომლებიც მოყვანილია მოცემულ კლასების დიაგრამაზე.



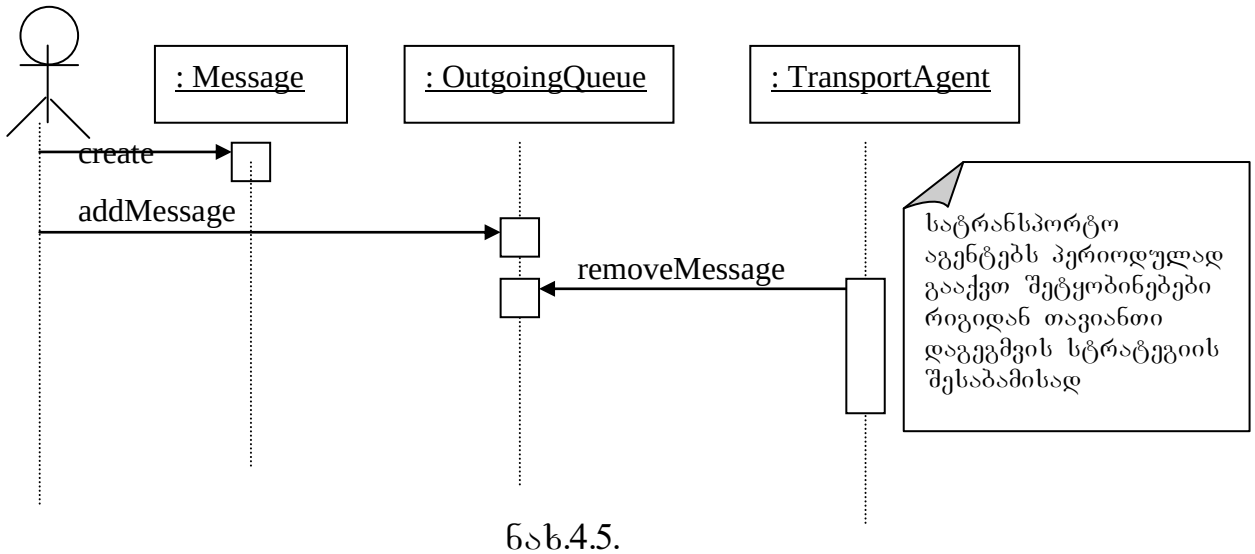
ნახ.4.4.

თუ სტრუქტურული შემადგენელი კოოპერაციისა ჩვეულებრივ გამოისახება კლასების დიაგრამის სახით, მისი ქცევითი შემადგენელი გამოისახება ურთიერთქმედების დიაგრამის სახით. ეს დიაგრამა აღწერს ურთიერთქმედებას, რომელიც შეესაბამება ქცევას, შედგენილს შეტყობინებების გაცვლაში ობიექტებს შორის გარკვეულ

კონტექსში. ურთიერთქმედების კონტექსს ადგენს მომცავი კოოპერაცია.

კოოპერაციის ქცევითი შემადგენელი შეიძლება აღვწეროთ ერთი ან რამოდენიმე დიაგრამით. თუ საჭიროა შეტყობინებების დალაგება დროის მიხედვით, გამოვიყენებთ ურთიერთქმედების დიაგრამას. ხოლო თუ ძირითადი აქცენტი კეთდება ობიექტებს შორის სტრუქტურულ მიმართებაზე, რომელიც წარმოიშობა ერთობლივი მოღვაწეობისას, გამოვიყენებთ კოოპერაციის დიაგრამას. შეიძლება გამოვიყენოთ ნებისმიერი სახის დიაგრამა, რამდენადაც უმეტეს შემთხვევაში ისინი სემანტიკურად ექვივალენტურნი არიან.

ეს ნიშნავს, რომ ვახდენთ რა ურთიერთქმედების მოდელირებას კლასების გარკვეული ერთობლიობის შიგნით როგორც კოოპერაციის, შეგვიძლია გავხსნათ იგი და გავეცნოთ ქცევის დეტალებს. ასე მაგალითად, თუ გავხსნით კოოპერაციას ***კვანძებს შორის შეტყობინება,*** დავინახავთ ურთიერთქმედების დიაგრამას, რომელიც მოყვანილია ნახ.4.5.-ზე.



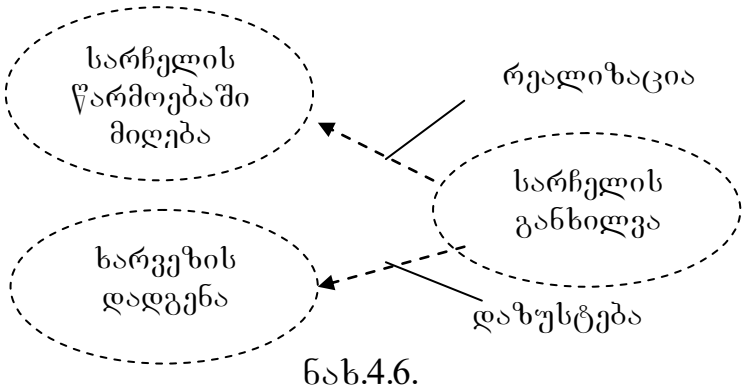
ნახ.4.5.

კოოპერაციები წარმოადგენენ სისტემური არქიტექტურის გულს, რამდენადაც სისტემის საფუძვლად დადებული მექანიზმები წარმოადგენენ მნიშვნელოვან საპროექტო გადაწყვეტებს. ყველა კარგათ სტრუქტურირებული ობიექტ-ორიენტირებული სისტემები

შედგებიან რეგულარულად მოწყობილი შედარებით არადიდი ზომის კოოპერაციების სიმრავლისაგან, ამიტომ მნიშვნელოვანია ვისწავლოთ მათი ორგანიზება. არსებობს ორი სახის მიმართება კოოპერაციებთან დამოკიდებულებაში, რომლებიც საჭიროა გათვალისწინებულ იქნას.

პირველ რიგში ეს არის მიმართება კოოპერაციასა და იმას, თუ რის რეალიზაციას ახდენს იგი. კოოპერაცია ახდენს კლასიფიკატორებისა ან ოპერაციების რეალიზებას. ეს ნიშნავს, რომ კოოპერაცია აღწერს შესაბამისი კლასიფიკატორისა ან ოპერაციის სტრუქტურულ ან ქცევით რეალიზაციას. მაგალითად, პრეცედენტი, რომელიც სისტემის მიერ შესრულებულ მოქმედებათა თანმიმდევრობას აღწერს, შეიძლება რეალიზებულ იქნას კოოპერაციით. ეს პრეცედენტი ასოციირებულ აქტიორებთან და მეზობელ პრეცედენტებთან ერთად წარმოადგენს კონტექსს კოოპერაციისათვის. ანალოგიურად კოოპერაციით შესაძლებელია რეალიზებულ იქნას ოპერაცია, რომელიც ასახელებს გარკვეული სისტემური მომსახურების რეალიზებას. ასეთ შემთხვევაში კონტექსი ფორმირდება ოპერაციითა და შესაბამისი პარამეტრებით. ასეთი მიმართება პრეცედენტსა ან ოპერაციას და რეალიზებად კოოპერაციას შორის მოდელირდება რეალიზების მიმართებით.

მეორე, არსებობს მიმართება თვით კოოპერაციებს შორის. კოოპერაციები შეიძლება აზუსტებდნენ სხვა კოოპერაციების აღწერას, რაც შეიძლება მოდელირებულ იქნას დაზუსტების მიმართებით.



ნახ.4.6.

მიმართების ეს ორი სახე ნაჩვენებია ნახ.4.6.-ზე. მასზე წარმოდგენილია კოოპერაცია ***სარჩელის განხილვა***, რომელიც რეალიზებას ახდენს პრეცედენტის ***სარჩელის წარმოებაში მიღება*** და აზუსტებს კოოპერაციას ***ხარვეზის დადგენა***.

კოოპერაციის ერთ ერთი დანიშნულება პრეცედენტების მოდელირებაა. სისტემის ანალიზისას ვხელმძღვანელობთ იმით, თუ როგორ შეიძლება მისი გამოყენება. გადავდივართ რა რეალიზაციის ეტაპზე, საჭირო ხდება იდენტიფიცირებული პრეცედენტების რეალიზება კონკრეტული სტრუქტურებისა და ქცევების სახით. საერთო ჯამში ყოველი პრეცედენტი რეალიზებულ უნდა იქნას ერთი ან რამოდენიმე კოოპერაციებით.

პრეცედენტების რეალიზების მოდელირება შედგება შემდეგი ეტაპებისაგან:

1. მოვახდინოთ იმ სტრუქტურული ელემენტების იდენტიფიცირება, რომლებიც შეადგენენ პრეცედენტის სემანტიკას.
2. მოვახდინოთ ამ სტრუქტურული ელემენტების ორგანიზება კლასების დიაგრამაში.
3. განვიხილოთ ცალკეული სცენარები, რომლებიც წარმოადგენენ მოცემულ პრეცედენტებს.
4. გამოვსახოთ ამ სცენარების დინამიკა ურთიერთქმედების დიაგრამაზე. ამისათვის ვისარგებლოთ მიმდევრობის და კოოპერაციის დიაგრამებით.
5. მოვახდინოთ ამ სტრუქტურული და ქცევითი ელემენტების ორგანიზება როგორც კოოპერაციისა, რომელიც შესაძლებელი იქნება დაუკავშიროთ პრეცედენტს რეალიზაციით.

4.4. კომპონენტების დიაგრამა

კომპონენტების დიაგრამა გამოიყენება ობიექტ-ორიენტირებული სისტემების ფიზიკური ასპექტების მოდელირებისათვის. ისინი გვიჩვენებენ კომპონენტების ორგანიზაციას და მათ შორის დამოკიდებულებებს.

კომპონენტების დიაგრამა გამოიყენება სისტემის სტატიკური სახის მოდელირებისათვის რეალიზაციის თვალთახედვით. აქ შედის კვანძზე განლაგებული ფიზიკური არსებების, მაგალითად პროგრამების, ბიბლიოთეკების, ცხრილების, ფაილების და დოკუმენტების მოდელირება. არსებითად, კომპონენტების დიაგრამა – ეს კლასების დიაგრამაა, ფოკუსირებულნი სისტემურ კომპონენტებზე.

გრაფიკულად კომპონენტების დიაგრამა ჩვეულებრივ შეიცავს კომპონენტებს, ინტერფეისებს და მიმართებებს მათ შორის (დამოკიდებულების, განზოგადების, ასოციაციის და რეალიზაციის).

სისტემის სტატიკური ასპექტების მოდელირებისას რეალიზაციის თვალთახედვით კომპონენტების დიაგრამები გამოიყენებიან:

- **საწყისი კოდის მოდელირებისათვის**. უმრავლესობა ობიექტ-ორიენტირებულ პროგრამირების ენებში კოდი იწერება დამუშავების ინტეგრირებულ გარემოში, რომლებიც ინახავენ საწყის ტექსტებს ფაილებში. კომპონენტების დიაგრამები შესაძლებელია გამოვიყენოთ ამ ფაილების კონფიგურაციის მართვის მოდელირებისათვის, რომლებიც წარმოადგენენ კომპონენტებს – სამუშაო პროდუქტებს.
- **შესრულებადი ვერსიების მოდელირებისათვის**. ვერსია – ეს შედარებით სრული და შეთანხმებული არტეფაქტების ნაკრებია, რომელიც წარედგინება შიდა ან გარე მომხმარებელს. სისტემისათვის, რომელიც შედგენილია კომპონენტებისაგან,

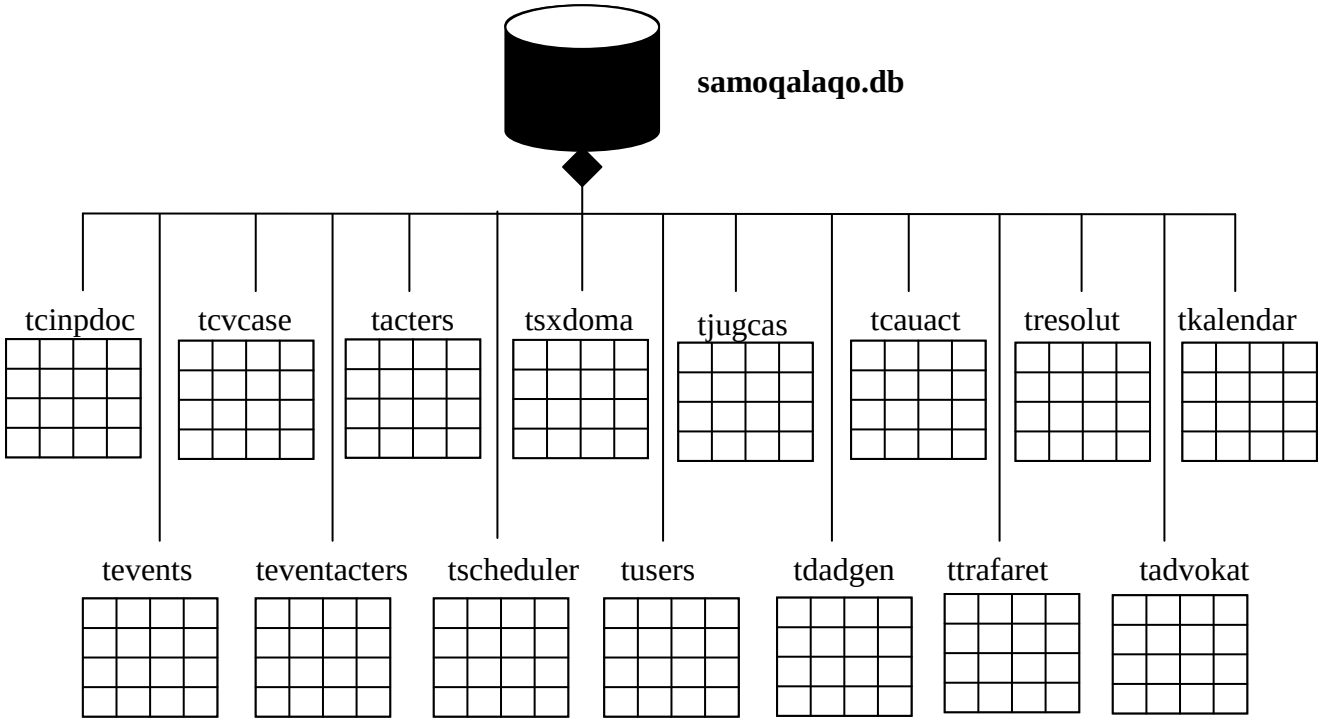
ვერსია უპირველეს ყოვლისა გულისხმობს იმ ნაწილებს, რომლებიც აუცილებელია დავაყენოთ მუშა სისტემის მისაღებათ. ვერსიების მოდელირებისას კომპონენტების დიაგრამის მეშვეობით ხდება გადაწყვეტილებების ვიზუალირება, სპეციფიცირება და დოკუმენტირება, რომლებიც მიიღება სისტემის ფიზიკური შემადგენლების, ესე იგი კომპონენტების განლაგების მიმართ.

- **მონაცემთა ბაზის ფიზიკური მოდელირებისათვის**. მონაცემთა ბაზის ლოგიკური სქემა აღწერს შენახული მონაცემების ლექსიკონს, ასევე მათ შორის კავშირის სემანტიკას. ფიზიკურად ყველაფერი ეს ინახება მონაცემთა ბაზაში. ლოგიკური სქემის ასახვა ობიექტ-ორიენტირებულ მონაცემთა ბაზაზე სიძნელეს არ წარმოადგენს, შედარებით რთულია ასახვა რელაციურზე. სიძნელე განპირობებულია იმით თუ როგორ გამოვსახოთ კლასები ცხრილების მეშვეობით და ოპერაციები, განსაზღვრულნი ლოგიკურ სქემაზე. ამ დროს ხელმძღვანელობენ შემდეგი მოსაზრებით- უბრალო ოპერაციები შექმნა, განახლება და ამოგდება რეალიზდება **SQL** და **ODBC** სტანდარტული საშუალებებით, ხოლო უფრო რთული ქცევა (მაგალითად, ბიზნეს წესები) გამოისახებიან ტრიგერებზე და შესანახ პროცედურებზე.

ადაპტური სისტემების მოდელირებისათვის. ზოგიერთი სისტემები აბსოლუტურათ სტატიკურნი არიან- მათი კომპონენტები ჩნდებიან, ღებულობენ მონაწილეობას შესრულებაში, ხოლო შემდეგ ქრებიან სცენიდან. სხვები უფრო დინამიურნი არიან. ისინი შეიცავენ კომპონენტებს, რომლებიც მიგრაციას განიცდიან. ასეთი სისტემების წარმოდგენისათვის გამოიყენება კომპონენტების დიაგრამა.

ნახ.4.8.-ზე მოყვანილია ცხრილები მონაცემთა ბაზისა აღებული სამოქალაქო სამართალწარმოების ინფორმაციული სისტემიდან. მონაცემთა ერთი ბაზა **სამოქალაქო** გამოსახულია კომპონენტის სახით სტერეოტიპით **database**, რომელიც შედგება ცხრილებისაგან

და ყველა ისინი გამოისახებიან კომპონენტების სახით სტერეოტიპით **interface**.



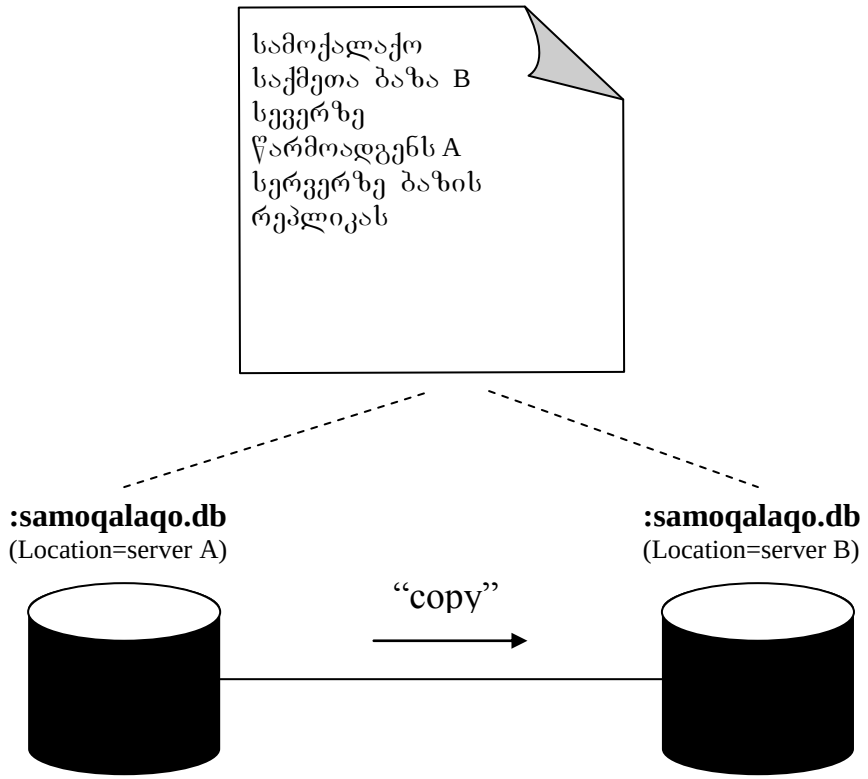
ნახ.4.8.

შესაძლებელია თითოეული ცხრილის შემადგენლობის სპეციფიცირება. კერძოთ, კომპონენტებს შეიძლება გააჩნდეთ ატრიბუტები, ამიტომ მონაცემთა ფიზიკური ბაზის მოდელირებისას ფართოდ გამოიყენება ატრიბუტები თითოეული ცხრილის აღწერისათვის. ასევე კომპონენტებს შეიძლება გააჩნდეთ ოპერაციები, რომლებითაც შეიძლება ვისარგებლოთ შენახული პროცედურების აღნიშვნისათვის.

ძირითადათ კომპონენტები მთელ თავის სიცოცხლეს ერთ კვანძზე ატარებენ, მაგრამ ზოგჯერ, განსაკუთრებით რთულ განაწილებულ სისტემებთან მუშაობისას, საჭიროა დინამიური წარმოდგენების მოდელირებაც. მაგალითად, სისტემამ შესაძლებელია მოახდინოს თავისი მონაცემთა ბაზის რეპლიცირება რამოდენიმე კვანძზე და გადაერთოს რეზერვულ სერვერზე მთავარის

მტყუნებისას. გლობალური განაწილებული სისტემის მოდელირებისას, რომელიც მუშაობს რეჟიმში 24×7 (შვიდი დღე კვირაში, 24 საათი დღეში) გამოიყენება მობილური აგენტები – კომპონენტები, რომლებიც მიგრირებენ ერთი კვანძიდან მეორეზე გარკვეული ოპერაციის შესასრულებლად. ასეთი დინამიური წარმოდგენების მოდელირებისათვის დაგვჭირდება კომპონენტების, ობიექტების და ურთიერთქმედების დიაგრამების კომბინირება.

ნახ.4.9.-ზე მოყვანილია მონაცემთა ბაზის რეპლიკაციის მოდელი, რომელიც წარმოდგენილია წინა ნახაზზე. გვაქვს კომპონენტ **samoqalaqo.db**-ს ორი ეგზემპლიარი და აქვთ სხვადასხვა მონიშნული მნიშვნელობები **Location**. შენიშვნაში ახსნილია, რომელი ეგზემპლიარი არის პირველადი და რომელი წარმოადგენს რეპლიკას.



ნახ.4.9.

4.5. განლაგების დიაგრამა

განლაგების ან გამოყენების დიაგრამა გამოიყენება ობიექტ-ორიენტირებული სისტემების ფიზიკური ასპექტების მოდელირებისათვის. კომპონენტების დიაგრამისაგან განსხვავებით იგი გვიჩვენებს კვანძების კონფიგურაციას, სადაც ხდება ინფორმაციის დამუშავება, და აგრეთვე თუ რომელი კომპონენტები არის განლაგებული ყოველ კვანძზე.

განლაგების დიაგრამა გამოიყენება სისტემის სტატიკური სახის მოდელირებისათვის განლაგების თვალთახედვით. ძირითადათ ამის ქვეშ იგულისხმება აპარატული საშუალებების ტოპოლოგიის მოდელირება, რომელზედაც სრულდება სისტემა.

ამრიგად, განლაგების დიაგრამაზე ნაჩვენებია დამმუშავებელი კვანძების კონფიგურაცია, რომელზედაც სრულდება სისტემა, და კომპონენტები, განლაგებულნი ამ კვანძებზე. განლაგების დიაგრამა შეიცავს კვანძებს და მიმართებებს (დამოკიდებულების და ასოციაციის) მათ შორის.

განლაგების დიაგრამას განსაკუთრებული მნიშვნელობა ენიჭება კლიენტ- სერვერული და მთლიანად განაწილებული სისტემების მოდელირებისას. კლიენტ-სერვერული სისტემა ტიპიური მაგალითია არქიტექტურის, სადაც ძირითადი ყურადღება ეთმობა მოვალეობების მკაფიო გამიჯვნას მომხმარებლის ინტერფეისსა, რომელიც არსებობს კლიენტზე, და სისტემის შენახულ მონაცემებს შორის, რომელიც არსებობენ სერვერზე. ამ დროს მნიშვნელოვანია გადაწყდეს თუ როგორ დავაკავშიროთ კლიენტები და სერვერები ქსელით, ასევე დავადგინოთ თუ როგორ არიან განაწილებულნი პროგრამული კომპონენტები კვანძებს შორის. განლაგების დიაგრამები საშუალებას გვაძლევენ მოვახდინოთ ასეთი სისტემების ტოპოლოგიის მოდელირება.

განაწილებულ სისტემებში გვაქვს ისეთი სისტემებიც, რომლებიც განაწილებულნი არიან გლობალურად და მოიცავენ სხვადასხვა დონის სერვერებს. ხშირად ასეთი ტიპის სისტემებზე დაყენებულია პროგრამული კომპონენტების სხვადასხვა ვერსიები, რომელთა ნაწილიც მიგრირებას ახდენს ერთი კვანძიდან მეორეზე. მსგავსი სისტემების მოდელირება მოითხოვს გადაწყვეტილებებს, რომლებიც ეხება სისტემური ტოპოლოგიის მუდმივ ცვლილებას.

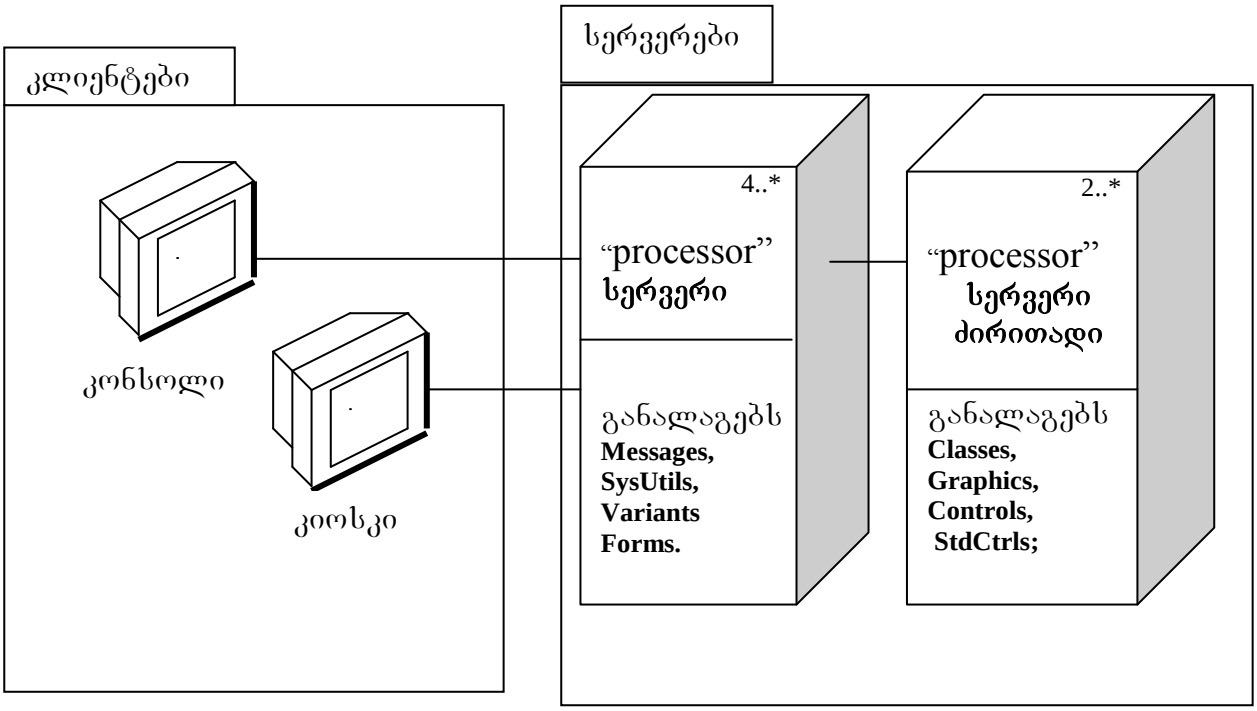
კლიენტ - სერვერული სისტემა. არსებობენ სისტემის სხვადასხვა ვარიაციები. მაგალითად, "თხელი" კლიენტი, რომლის გამოთვლითი რესურსები შეზღუდულია და ძირითადათ დაკავებულია მომხმარებელთან ურთიერთქმედებით და ინფორმაციის ასახვით. "თხელ" კლიენტებს შეიძლება არ გააჩნდეთ საკუთარი კომპონენტები. ამის მაგიერ ისინი ტვირთავენ კომპონენტებს სერვერიდან მოთხოვნილებისამებრ, ისე როგორც **Enterprise JavaBeans** – ის შემთხვევაში. მეორეს მხრივ, შეიძლება ავირჩიოთ "მსხვილი" კლიენტი, რომელსაც გამოთვლითი რესურსები მეტი აქვს და რომელსაც ამის გამო შეუძლია დაკავდეს არა მარტო ვიზუალირებით. ამორჩევა "თხელ" და "მსხვილ" კლიენტს შორის ეს არქიტექტორული გადაწყვეტაა, რომელზეც გავლენას ახდენს სხვადასხვა ტექნიკური, ეკონომიკური და პოლიტიკური ფაქტორები.

ნებისმიერ შემთხვევაში სისტემის გაყოფისას კლიენტურ და სერვერულ ნაწილად გვიხდება გადაწყვეტილების მიღება, თუ სად განვათავსოთ ფიზიკურად კომპონენტები და როგორ უზრუნველვყოთ პასუხისმგებლობის ბალანსი მათ შორის. მაგალითად, მართვის სისტემების უმრავლესობის არქიტექტურა ინფორმაციით სამდონიანია, ანუ მომხმარებლის ინტერფეისი, ბიზნეს-ლოგიკა და მონაცემთა ბაზა ფიზიკურად განცალკევებულია ერთმანეთისაგან. გადაწყვეტილება იმის შესახებ, თუ სად უნდა განლაგდეს ინტერფეისი და მონაცემთა ბაზა გასაგებია, ხოლო სად უნდა

იმყოფებოდნენ კომპონენტები, რომლებიც ბიზნეს - ლოგიკის რეალიზებას ახდენენ, შედარებით რთულია.

განლაგების დიაგრამა შესაძლებელია გამოვიყენოთ კლიენტ-სერვერული სისტემის ტოპოლოგიის აღწერისა და დადგენისათვის, ასევე პროგრამული კომპონენტების განაწილებისათვის კლიენტსა და სერვერს შორის.

კლიენტ-სერვერული სისტემის მოდელირებისას უნდა მოვახდინოთ კვანძების იდენტიფიცირება, რომლებიც წარმოადგენენ კლიენტისა და სერვერის პროცესორებს. გამოვყოთ ის მოწყობილობები, რომლებიც ასე თუ ისე გავლენას ახდენენ სისტემის ქცევაზე. სტერეოტიპების მეშვეობით დავამუშავოთ ვიზუალური აღნიშვნები პროცესორებისა და მოწყობილობებისათვის. ბოლოს, დავამოდელიროთ კვანძების ტოპოლოგია განლაგების დიაგრამაზე.

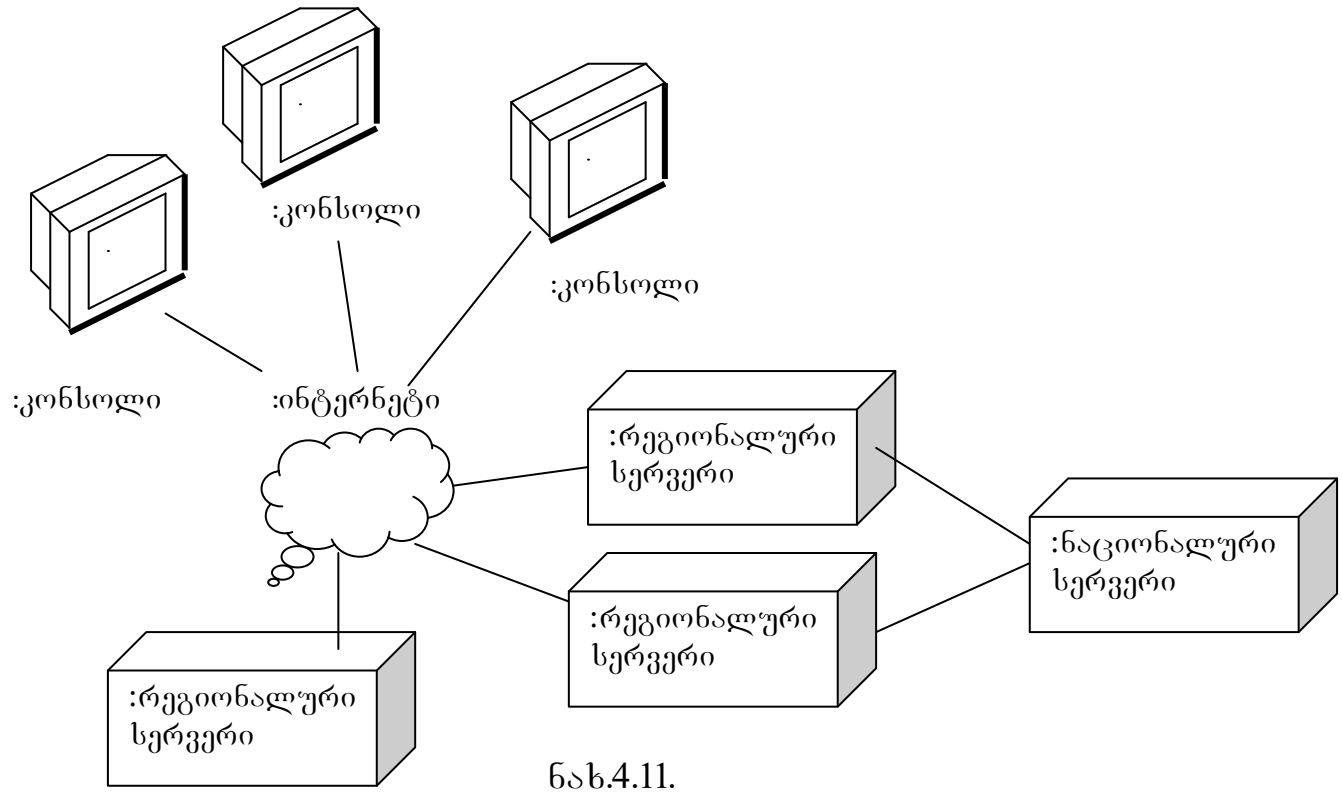


ნახ.4.10.

მთლიანად განაწილებული სისტემა. განაწილებული სისტემები შეიძლება იყვნენ:

- უბრალო ორპროცესორიანი;
- განშტოებულნი - განლაგებულნი მრავალ გეოგრაფიულად დაშორებულ კვანძებში.

უკანასკნელნი როგორც წესი არ არიან სტატიკურნი. კვანძები ჩნდებიან და ქრებიან პროცსორების მწყობრიდან გამოსვლის გამო. იქმნება ახალი, უფრო ჩქარი კავშირგაბმულების არხები, რომლებიც ფუნქციონირებენ ნელის პარალელურად, რომელთა ბოლოს და ბოლოს დემონტირებას ახდენენ. იცვლება არა მარტო სისტემის ტოპოლოგია, არამედ პროგრამული კომპონენტების განაწილება. მაგალითად მონაცემთა ბაზის ცხრილები შეიძლება გადაადგილდეს სერვერებს შორის რათა დაუახლოვოთ ისინი ინფორმაციის მომხმარებლებს.



ნახ.4.11.

მთლიანად განაწილებული სისტემის მოდელირებისას უნდა მოვახდინოთ მოწყობილობების და პროცესორების იდენტიფიცირება, ისევე როგორც კლიენტ-სერვერული სისტემის შემთხვევაში. განსაკუთრებულ ყურადღება უნდა მიექცეს კვანძების დაჯგუფებას, რისთვისაც შესაძლებელია პაკეტების გამოყენება. წარმოვადგენთ რა მოწყობილობებს და პროცესორებს განლაგების დიაგრამის სახით, ყველგან სადაც შესაძლებელია უნდა გამოვიყენოთ ინსტრუმენტალური საშუალებები სისტემის ქსელური ტოპოლოგიის გახსნისათვის.

თუ საჭიროა ყურადღების გამახვილება სისტემის დინამიკაზე, უნდა გამოვიყენოთ პრეცედენტების დიაგრამა ჩვენთვის მნიშვნელოვანი ქცევის სპეციფიცირებისათვის და მათი გახსნა განვახორციელოთ ურთიერთქმედების დიაგრამით.

ნახ.4.11.-ზე გამოსახულია მთლიანად განაწილებული სისტემის ტოპოლოგია. მასზე გამოსახულია სამი კონსოლი, რომლებიც დაკავშირებულნი არიან ინტერნეტთან. მეორეს მხრივ, გვაქვს სამი რეგიონალური სერვერი, რომლებიც ანხორციელებენ ინტერფეისს ნაციონალურ სერვერთან.

თავი 5
ავტომატიზებული სისტემის აგება და დამუშავების ორგანიზება

5.1. სისტემა და ქვესისტემა

სისტემის ავტომატიზაცია მოითხოვს მის მოდელირებას. მოდელი კი ეს აბსტრაქციაა, რომელიც იქმნება იმისათვის, რომ უკეთ გავიგოთ სისტემა. სისტემა აღიწერება მოდელების ნაკრებით, რომლებიც შესაძლებლობის მიხედვით განიხილავენ მას სხვადასხვა თვალთახედვით. ჩვენ საშუალება გვეძლევა მოვახდინოთ სისტემების და ქვესისტემების მოდელირება როგორც ერთიანი მთლიანის და ამით ორგანულად გადავწყვიტოთ მასშტაბირების პრობლემა.

ავტომატიზაციისას სისტემის აბსტრაქციები წარმოიდგინებიან მოდელების სახით, რომელთაგან თითოეული წარმოადგენს შედარებით დამოუკიდებელ, მაგრამ მნიშვნელოვან ასპექტს დასამუშავებელი სისტემის. ამ აბსტრაქციებიდან ჩვენთვის საინტერესო ნაკრების ვიზუალირებისათვის შეიძლება გამოვიყენოთ დიაგრამები. სისტემის არქიტექტურის ხუთი სხვადასხვა წარმოდგენა განსაკუთრებით სასარგებლოა დამუშავების პროცესის სხვადასდხვა წარმომადგენელთა მოთხოვნილებების დასაკმაყოფილებლად. მთლიანობაში ეს მოდელები იძლევიან სრულ წარმოდგენას სისტემის სტრუქტურისა და ქცევის შესახებ.

შემდეგში მოყვანილი დებულებების უკეთ გაგებისათვის შემოვიტანოთ ზოგიერთი განმარტებები.

სისტემა ეს ელემენტების ერთობლიობაა, ორგანიზებულნი გარკვეული სახით განსაზღვრული მიზნის შესასრულებლად. იგი აღიწერება მოდელების ნაკრებით, სხვადასხვა თვალთახედვით.

ელემენტების სიმრავლე დიდ სისტემებში შესაძლებელია დაიყოს უფრო მცირე ქვესისტემებათ, რომელთაგან თითოეული აბსტრაქციის

უფრო დაბალ დონეებზე შეიძლება განვიხილოთ როგორც დამოუკიდებელი სისტემა.

ქვესისტემა ეს წარმონაქმნია, რომელიც გამოიყენება რთული სისტემის დეკომპოზიციისათვის უფრო მარტივებათ, ერთმანეთზე სუსტად დამოკიდებულ შემადგენლებზე.

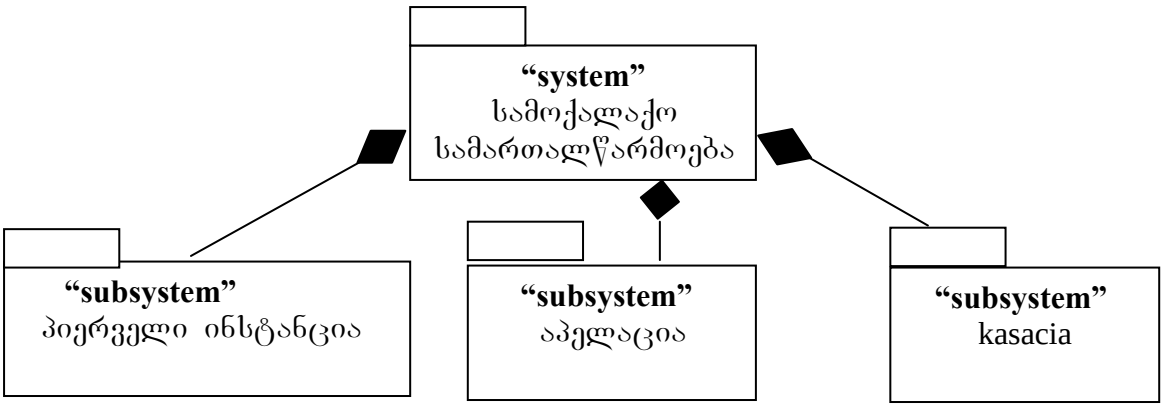
მოდელი ეს რეალობის გამარტივებაა, აბსტრაქციაა, რომელიც იქმნება სისტემის უკეთ გაგებისათვის.

სახე ან წარმოდგენა(View) - ეს მოდელია, რომელიც განიხილება გარკვეული თვალსაზრისით: მათში გამოსახულია ერთი არსებები და გამოტოვებულია სხვები, რომლებსაც მოცემული თვალსაზრისით ინტერესი არ გააჩნიათ.

სისტემა არის ის არსება, რომელსაც ვამუშავებთ და რომლისთვისაც ვადგენთ მოდელს. სისტემა მოიცავს ყველა არტეფაქტს, რომლებიც შეადგენენ ამ არსებას, მოდელისა და ამ მოდელის ელემენტების ჩათვლით, როგორიცაა კლასები, ინტერფეისები, კომპონენტები, კვანძები და კავშირები მათ შორის.

სისტემა გამოისახება სტერეოტიპული პაკეტის სახით. წარმოადგენს რა სტერეოტიპულ პაკეტს, იგი ფლობს გარკვეულ ელემენტებს. თუ ჩავიხედებით სისტემის შიგნით, შეიძლება დავინახოთ ყველა მისი მოდელი და ცალკეული ელემენტები. წარმოადგენს რა კლასიფიკატორს, სისტემას შეიძლება გააჩნდეს ეგზემპლიარები (შეიძლება არსებობდეს რამოდენიმე სისტემა განლაგებულნი სხვადასხვა ადგილზე), ატრიბუტები და ოპერაციები (სისტემის გარე აქტიორებს უნარი აქვთ იმოქმედონ სისტემაზე), პრეცედენტები, ავტომატები და კოოპერაციები. თითოელ მათგანს შეუძლია მონაწილეობა მიიღოს სისტემის ქცევის სპეციფიცირებაში.

ნახ. 5.1.-ზე წარმოდგენილია სამოქალაქო სამართალწარმოების სისტემა და მისი ქვესისტემები.



ნახ.5.1.

ის რაც აბსტრაქციის ერთ დონეზე გამოისახება სისტემად, მეორეზე უფრო მაღალ დონეზე შეიძლება განხილულ იქნას როგორც ქვესისტემა. ძირითადი დამოკიდებულება სისტემასა და ქვესისტემას შორის აგრეგირებაა. სისტემა შეიძლება შედგებოდეს ნული და მეტი ქვესისტემებისაგან. სისტემა ეს ყველაზე მაღალი დონის არსებაა მოცემულ კონტექსში. მისი შემადგენელი ქვესისტემები ყოფენ სისტემას გადაუკვეთავ ნაწილებათ.

იმ დროს როდესაც ქვესისტემა წარმოადგენს დიდი სისტემის ელემენტების სიმრავლის დაყოფას დამოუკიდებელ ნაწილებათ, მოდელი ეს აბსტრაქციის სიმრავლის დაყოფაა, რომელიც გამოიყენება ამ სისტემის ვიზუალირების, სპეციფიცირების, კონსტრუირების და დოკუმენტირებისათვის. განვალაგებთ რა სისტემას ქვესისტემებათ, ვახორციელებთ მის დამუშავებას და განლაგებას ერთმანეთისაგან დამოუკიდებლად. სისტემის ან ქვესისტემის აბსტრაქციებს კი ვყოფთ მოდელებათ იმისათვის, რომ უკეთესად გავიგოთ თუ რის დამუშავებას და განლაგებას ვაპირებთ.

მოდელი შეიძლება შეიცავდეს იმდენად ბევრ არტეფაქტებს, რომ დიდ სისტემაში მთელი მათი ერთობლიობის უეცრად მოცვა შეუძლებელი იყოს. სისტემური არქიტექტურის სახე შეიძლება წარმოვიდგინოთ როგორც მოდელის ერთერთი პროექცია. ყოველი

მოდელისათვის გათვალიწინებულია რიგი დიაგრამებისა რომელთა მეშვეობით მოხერხებულია მისი კუთვნილი არსებების მიმოხილვა. წარმოდგენა მოიცავს არსებების სიმრავლეს, რომლებიც შედის მოდელის შემადგენლობაში.

როგორც ავღნიშნეთ ძირითადი დამოკიდებულება სისტემასა და ქვესისტემას შორის ეს აგრეგირებაა. ხოლო მიმართებების სპეციფიცირება ისეთ ელემენტებს შორის, როგორის არის კლასები, ინტერფეისები, კომპონენტები და კვანძები ეს მოდელის მნიშვნელოვანი სტრუქტურული შემადგენელია. რაც შეეხება კონცეპტუალურ კავშირებს ელემენტებს შორის, რომლებიც არსებობენ სხვადასხვა მოდელებს შორის, ისინი შესაძლებელია წარმოვადგინოთ ტრასირების მიმართების სახით. ტრასირება არ შეიძლება წარმოვადგინოთ ელემენტების მიმართ ერთი მოდელის ფარგლებში. ტრასირება წარმოიდგინება სტერეოტიპული დამოკიდებულებით. ტრასირების დამოკიდებულება გამოიყენება იმისათვის რათა ვუჩვენოთ გზა მოთხოვნიდან რეალიზაციამდე, რომელზედაც განლაგებულია ყველა შუალედური ვერსიები.

5.2. სისტემის მოდელირება სხვადასხვა წარმოდგენების საფუძველზე

სისტემების და მოდელების მოყვანილი ცნებები გამოიყენება იმ ელემენტების ორგანიზაციისათვის, რომლებიც საჭიროა სისტემის არქიტექტურის ასახვისათვის. სისტემური არქიტექტურის მოდელირებისას, ვაერთიანებთ იმ გადაწყვეტილებებს, რომლებიც მიღებულია სისტემის მოთხოვნების, მისი ლოგიკური და ფიზიკური ელემენტების მიმართ. მოდელირდება სისტემის სტრუქტურული და ქცევითი ასპექტები, რომლებიც ახდენენ მათი სხვადასხვა წარმოდგენების ფორმირებას. ბოლოს, საჭიროა ყურადღება მიექცეს

ქვესისტემების დაკავშირებას და გადაწყვეტილებების ტრასირებას, დაწყებული მოთხოვნების ფორმულირებიდან განლაგების ეტაპამდე.

როგორც აღნიშნული იყო არსებობს ხუთი ურთიერთშევსებადი სახე, ან წარმოდგენა, რომლებიც განსაკუთრებით მნიშვნელოვანია სისტემის არქიტექტურის ვიზუალირების, სპეციფიცირების, კონსტრუირების და დოკუმენტირებისათვის. ეს წარმოდგენებია პრეცედენტების, პროექტირების, პროცესების, რეალიზაციის და განლაგების თვალთახედვით. ყოველი მათგანი ითვალისწინებს სტრუქტურულ და ქცევით მოდელირებას, ესე იგი სტატიკური და დინამიკური არსებების მოდელირებას. ერთობლიობაში ეს სახეობები საშუალებას იძლევიან გადავცეთ ყველაზე მნიშვნელოვანი გადაწყვეტილებები, რომლებიც ეხება სისტემას მთლიანობაში, ხოლო ცალკე თითოეული ყურადღებას ამახვილებს ერთ ასპექტზე, რომლის განხილვა ამ სახით მარტივდება.

იმისათვის, რომ გამოვსახოთ სისტემა რომელიმე თვალთახედვით გამოიყენება დიაგრამები. როგორც უკვე ნაჩვენები იყო განსაზღვრულია ცხრა ტიპის დიაგრამა, რომელთა კომბინირება შესაძლებელია საჭირო წარმოდგენის მისაღებათ. მაგალითად, სტატიკური ასპექტების ვიზუალირება სისტემის რეალიზაციის თვალთახედვით უფრო მოხერხებულია კომპონენტების დიაგრამის გამოყენებით, ხოლო დინამიკური ასპექტების იმავე თვალთახედვით – ურთიერთქმე0დების დიაგრამით. სტატიკური ასპექტები სისტემის მონაცემთა ბაზის შეიძლება გამოვსახოთ კლასების დიაგრამით, ხოლო დინამიკურის – კოოპერაციის დიგრამის მეშვეობით. ყველა შემთხვევაში მოგვიხდება დიაგრამების დამუშავება ინკრემენტულად (უმატებთ რა თითო ახალ ფრაგმენტს ყოველ ერთ ჯერზე) და იტერაციულად (ვიმეორებთ რა პროექტირების პროცესს ყოველი ახალი გაუმჯობესების შემდეგ).

სისტემის მოდელირებისას სხვადასხვა თვალთახედვით ფაქტიურად ხდება მისი კონსტრუირება რამოდენიმე განზომილებაში. წარმოდგენის ერთობლიობის სწორი არჩევა საშუალებას მოგვცემს უფრო სწორად განვსაზღვროთ სისტემასთან დაკავშირებული საკითხები, გამოვავლინოთ რისკი, რომელიც უნდა გათვალისწინებულ იქნას. თუ სახეები არჩეულია ცუდად ან ყურადღებას ამახვილებთ მხოლოდ ერთზე სხვების მხედველობაში მიუღებლად, მაშინ იზრდება საშიშროება ვერ შევნიშნოთ ისეთი საკითხები, რომელთა გაუთვალისწინებლობა ადრე თუ გვიან მთელ პროექტს დააყენებს საშიშროების წინაშე.

სისტემის მოდელირება სხვადასხვა წარმოდგენების გამოყენებით მოითხოვს ორი ამოცანის გადაწყვეტას:

- პირველ რიგში უნდა დადგინდეს წარმოდგენის რომელი სახეობები გამოხატავენ ყველაზე უკეთესად სისტემის არქიტექტურას;
- მეორე - ყოველი ამორჩეული სახეობის მიმართ განისაზღვროს, რომელი არტეფაქტებია საჭირო მისი ყველაზე არსებითი დეტალების გამოსახვისათვის. ეს არტეფაქტები უმრავლეს შემთხვევაში შედგებიან სხვადასხვა დიაგრამებისაგან.

პირველი ამოცანის გადაწყვეტისას გამოვდივართ დასამუშავებელი სისტემის მასშტაბებიდან. დაუშვათ ჩვენ ვამოდელირებთ უბრალო დანართს, რომელიც სრულდება ერთ კომპიუტერზე, ამ დროს დაგვჭირდება:

- წარმოდგენა გამოყენებით შემთხვევათა თვალთახედვით – პრეცედენტების დიაგრამა;
- წარმოდგენა პროექტირების თვალთახედვით – კლასების დიაგრამა (სტრუქტურული მოდელირებისათვის) და ურთიერთქმედების დიაგრამა (ქცევის მოდელირებისათვის).

დანარჩენი წარმოდგენები მოცემული შემთხვევისათვის არ დაგვჭირდება.

თუ სისტემა აწყობილია არქიტექტურით “კლიენტი-სერვერი”, სასურველი იქნება ჩავრთოთ სამუშაოში კომპონენტები და განლაგება რეალიზაციის კონკრეტული ფიზიკური დეტალების მოდელირებისათვის.

რთული განაწილებული სისტემის მოდელირებისას, უნდა გამოვიყენოთ არსებული ყველა დიაგრამა. ისინი საშუალებას მოგვცემენ გამოვხატოთ მისი არქიტექტურა და პროექტთან დაკავშირებული ტექნიკური რისკი. სასამართლო სამოქალაქო სამართლის სისტემისათვის გამოყენებულ იქნა შემდეგი წარმოდგენები:

- წარმოდგენა პრეცედენტების თვალთახედვით – პრეცედენტების დიაგრამა და აქტიურობის დიაგრამები (ქცევის მოდელირებისათვის);
- წარმოდგენა პროექტირების თვალთახედვით – კლასების დიაგრამა (სტრუქტურული მოდელირებისათვის), ურთიერთქმედების დიაგრამა (ქცევის მოდელირება), მდგომარეობათა დიაგრამა (ქცევის მოდელირება);
- წარმოდგენა პროცესების თვალთახედვით – კლასების დიაგრამა (სტრუქტურული მოდელირებისათვის) და ურთიერთქმედების დიაგრამა (ქცევის მოდელირება).
- წარმოდგენა რეალიზაციის თვალთახედვით – კომპონენტების დიაგრამა;
- წარმოდგენა განლაგების თვალთახედვით – განლაგების დიაგრამა.

ამრიგათ, მთლიანად სისტემის მოდელირებისათვის უნდა შესრულდეს შემდეგი მოქმედებები:

1. ***მოვახდინოთ სისტემის სახის სპეციფიცირება გამოყენებითი ვარიანტების ან პრეცენდენტების თვალსაზრისით.*** სტატიკური ასპექტების მოდელირებისათვის ვიყენებთ პრეცენდენტების დიაგრამას, ხოლო დინამიურის – ურთიერთქმედების, მდგომარეობათა და აქტიურობის დიაგრამებს. სამოქალაქო სამართალწარმოების სისტემისათვის დამუშავებული იქნა 90- მდე პრეცედენტების და აქტიურობის დიაგრამები, რომლებიც აღწერენ სამართალწარმოების პროცესს პირველი ინსტანციის, საოლქო და უზენაეს სასამართლოებში.
2. ***მოვახდინოთ სისტემის სახის სპეციფიცირება პროექტირების თვალსაზრისით.*** მასში შედიან კლასები, ინტერფეისები და კოოპერაციები, რომლებიც ახდენენ საგნობრივი სფეროს ლექსიკონის და წარმოდგენილი გადაწყვეტილების ფორმირებას. სტატიკური ასპექტების მოდელირებისათვის ვიყენებთ კლასებისა და ობიექტების დიაგრამებს, ხოლო დინამიურის – მიმდევრობის, მდგომარეობათა და აქტიურობის დიაგრამებს. სამოქალაქო სამართალწარმოების სისტემისათვის დამუშავდა კლასების დიაგრამა წარმოების სამივე დონისათვის (პირველი ინსტანციისათვის, აპელაცია და კასაცია).
3. ***მოვახდინოთ სისტემის სახის სპეციფიცირება პროცესების თვალსაზრისით.*** მასში შედიან პროცესები, რომლებიც ახდენენ სისტემაში პარალელიზმისა და სინქრონიზაციის მექანიზმის ფორმირებას. სტატიკური ასპექტების მოდელირებისათვის ვიყენებთ კლასებისა და ობიექტების დიაგრამებს, ხოლო დინამიურის – მიმდევრობის, მდგომარეობათა და აქტიურობის დიაგრამებს. განსაკუთრებული ყურადღება დაუთმეთ აქტიურ კლასებსა და ობიექტებს. სამოქალაქო სამართალწარმოებისას აქტიური კლასების ურთიერთქმედება დანარჩენ კლასებთან განხილულია მეორე თავში.

4. ***მოვახდინოთ სისტემის სახის სპეციფიცირება რეალიზაციის თვალსაზრისით.*** მასში შედიან კომპონენტები, რომლებიც გამოიყენებიან მზა ფიზიკური სისტემის აწყობისა და გამოშვებისათვის. სტატიკური ასპექტების მოდელირებისათვის ვიყენებთ კომპონენტების დიაგრამებს, ხოლო დინამიურის – ურთიერთქმედების, მდგომარეობათა და აქტიურობის დიაგრამებს. სამოქალაქო სამართალწარმოების სისტემის რეალიზებისათვის გამოყენებული იქნა განლაგების კომპონენტები, რომლებიც აუცილებელია და საკმარისი შესრულებადი სისტემის აგებისათვის. მათ რიცხვს მიეკუთვნება დინამიურად დამაკავშირებელი ბიბლიოთეკები **(DLL)** და შესრულებადი პროგრამები **(EXE)**. ისინი ძირითადათ შეიცავენ კლასიკურ ობიექტურ მოდულებს **COM+, CORBA** და **Enterprise JavaBeans**. რაც შეეხება კომპონენტებს – მუშა პროდუქტებს, რომლებშიც შედიან ფაილები პროგრამის საწყისი ტექსტებით და მონაცემებით მოყვანილია დანართში.
5. ***მოვახდინოთ სისტემის სახის სპეციფიცირება განლაგების თვალსაზრისით.*** მასში შედიან კვანძები, რომლებიც ახდენენ აპარატული საშუალებების ტოპოლოგიის ფორმირებას და რომლებზეც სრულდება სისტემა. სტატიკური ასპექტების მოდელირებისათვის ვიყენებთ განლაგების დიაგრამებს, ხოლო დინამიურის – ურთიერთქმედების, მდგომარეობათა და აქტიურობის დიაგრამებს. სამოქალაქო სამართალწარმოების სისტემის აპარატული საშუალებების შერჩევისა და ტოპოლოგიის დადგენის საკითხები განხილულია § 4.4.-ში.
6. ***მოვახდინოთ კოოპერაციის მეშვეობით პროექტირების ნიმუშების მოდელირება.*** სამოქალაქო სამართალწარმოების სისტემისათვის დადგენილი და დამუშავებული იქნა ისეთი კოოპერაციები როგორიც არის სასამართლოს განსჯადობა და უწყებრივი

ქვემდებარება, მოქმედ მხარეთა უფლებამოსილება და ქმედუნარიანობა, სასამართლო შეტყობინება და დაბარება, მოწმეთა ჩვენებების მიღება, სარჩელის განხილვა, მესამე პირის ჩართვა და სხვა. აღნიშნული კოოპერაციები აღწერენ სამოქალაქო სამართალწარმოების ცალკეულ საპროცესო მოქმედებებს და შესაბამისად წარმოადგენენ სამართალწარმოების პროექტირების ნიმუშებს. მოყვანილი კოოპერაციების აღწერა პრეცედენტების დიაგრამის სახით მოყვანილია დანართში.

სისტემის დამუშავების პროცესი გულისხმობს არქიტექტურის მიმდევრობით დაზუსტებას პრეცედენტების ანალიზის, იტერაციული და ინკრემენტული კვლევის საფუძველზე.

სისტემის სირთულის ზრდასთან ერთად აუცილებელი ხდება მისი დეკომპოზიცია უფრო მარტივ ქვესისტემებათ, რომელთაგან თითოეული შესაძლებელია დამუშავდეს დამოუკიდებლად, ხოლო შემდეგ თანდათან გავაერთიანოთ ისინი. სამოქალაქო სამართალწარმოების სისტემის დამუშავებისას გამოყოფილ იქნა წარმოების პროცესი პირველი ინსტანციის სასამართლოებში, წარმოების პროცესი სააპელაციო სასამართოებში და წარმოება საკასაციო სასამართლოში. დამუშავებული ნაწილები გაერთიანებული იქნენ ერთ მთლიან სამოქალაქო სამართალწარმოების სისტემაში.

ქვესისტემის მოდელირებისას ყოველი ქვესისტემისათვის ვახდენთ კონტექსის სპეციფიცირებას, ისევე როგორც ეს ხდება სისტემისათვის მთლიანად, ხოლო ყოველი ქვესისტემის არქიტექტურის მოდელირება წარმოებს ისევე როგორც ეს ხდება მთელი სისტემისათვის.

5.3. აბსტრაქციის სხვადასხვა დონეები

დამმუშავებლისათვის აუცილებელია განიხილოს სისტემა არა მარტო სხვადასხვა თვალთახედვით, არამედ აბსტრაქციის სხვადასხვა დონეებზე. მაგალითად, თუ გვაქვს კლასები, რომლებიც მოიცავენ საპრობლემო სფეროს ლექსიკონს, მაშინ პროგრამისტისათვის საჭირო იქნება თითოეული მათგანის შესწავლა შესაბამისი ატრიბუტებით, ოპერაციებითა და მიმართებებით. მეორეს მხრივ, ანალიტიკოსისათვის, რომელიც ათანხმებს საბოლაო მომხმარებელთან სისტემის გამოყენების სხვადასხვა ვარიანტებს, იგივე კლასები დასრულებული სისტემის დასჭირდებათ მაქსიმალურად გამარტივებული სახით. მაშასადამე, შეიძლება ითქვას, რომ პროგრამისტი მუშაობს შედარებით დაბალ, ხოლო ანალიტიკოსი და მომხმარებელი შედარებით მაღალ აბსტრაქციის დონეებზე. რამდენადაც დიაგრამები, არსებითად წარმოადგენენ მოდელის შემადგენელი ელემენტების გრაფიკულ წარმოდგენას, შეგვიძლია დავხაზოთ ერთი და იმავე მოდელის რამოდენიმე დიაგრამა, რომელთაგან თითოეული მალავს ან პირიქით, წარმოაჩენს ზოგიერთ ელემენტებს და შესაბამისად გვიჩვენებს დეტალიზაციის სხვადასხვა დონეს.

არსებობს ორი ძირითადი საშუალება სისტემის აბსტრაქციის სხვადასხვა დონეების მოდელირებისა:

- შეიძლება ავაგოთ ერთი და იმავე მოდელის დიაგრამები, რომლებიც ხასიათდებიან დეტალიზაციის სხვადასხვა დონით.
- შევქმნათ სხვადასხვა მოდელები აბსტრაქციის სხვადასხვა დონით და ტრასირების დიაგრამები ერთი მოდელიდან მეორეზე.

სისტემის მოდელირება აბსტრაქციის სხვადასხვა დონეებზე დეტალიზაციის სხვადასხვა დონის დიაგრამების გამოყენებით ხდება შემდეგნაირად:

1. თუ შესაქმნელია მოდელი რეალიზაციისათვის, მაშინ დაგვჭირდება ყველაზე დაბალი დონის დიაგრამები, ხოლო საბოლაო მომხმარებლისათვის ძირითადი კონცეფციების გადასაცემად, დაგვჭირდება მაღალდონიანი დიაგრამები, რომელშიც დეტალების უმრავლესობა დამალულია.
2. იმისგან დამოკიდებულებით, თუ რა მოთხოვნებია გასათვალისწინებელი, ვქმნით დიაგრამას შესაბამის აბსტრაქციის დონეზე. რისთვისაც ვმალავთ ან წარმოვაჩენთ არსებებს, რომლებიც ქმნიან მოდელს. კერძოთ, ვიყენებთ მხოლოდ იმ სამშენებლო ბლოკებს და მიმართებებს მათ შორის, რომლებიც შეესაბამებიან ამ მოთხოვნებს, დანარჩენებს უგულებელვყოფთ. ქცევის დიაგრამებზე უჩვენებთ მხოლოდ იმ შეტყობინებებს და გადასვლებს, რომლებსაც აქვთ მნიშვნელობა არსებული მოთხოვნების გადმოსაცემად.

ძირითადი უპირატესობა მოცემული მიდგომისა იმაშია, რომ მოდელირების პროცესში გამოიყენება სემანტიკური მცნებების საერთო ნაკრები. ძირითადი ნაკლოვანება კი იმაშია, რომ ცვლილებამ დიაგრამაზე აბსტრაქციის ერთ დონეზე, შეიძლება გამოიწვიოს სხვა დონეებზე არსებული დიაგრამების აქტუალობის დაკარგავა.

სისტემის პროექტირება აბსტრაქციის სხვადასხვა დონეებზე სხვადასხვადონიანი მოდელების გამოყენებით წარმოებს შემდეგნაირად:

1. ავირჩიოთ აბსტრაქციის საჭირო დონე და შევქმნათ ყოველი დონისათვის თავისი მოდელი.
2. მოდელები, რომლებიც უფრო მაღალ დონეზეა, უნდა შეიცავდნენ უფრო განზოგადებულ (უბრალო) აბსტრაქციებს, ხოლო დაბალ დონეებზე – დეტალიზებულს.

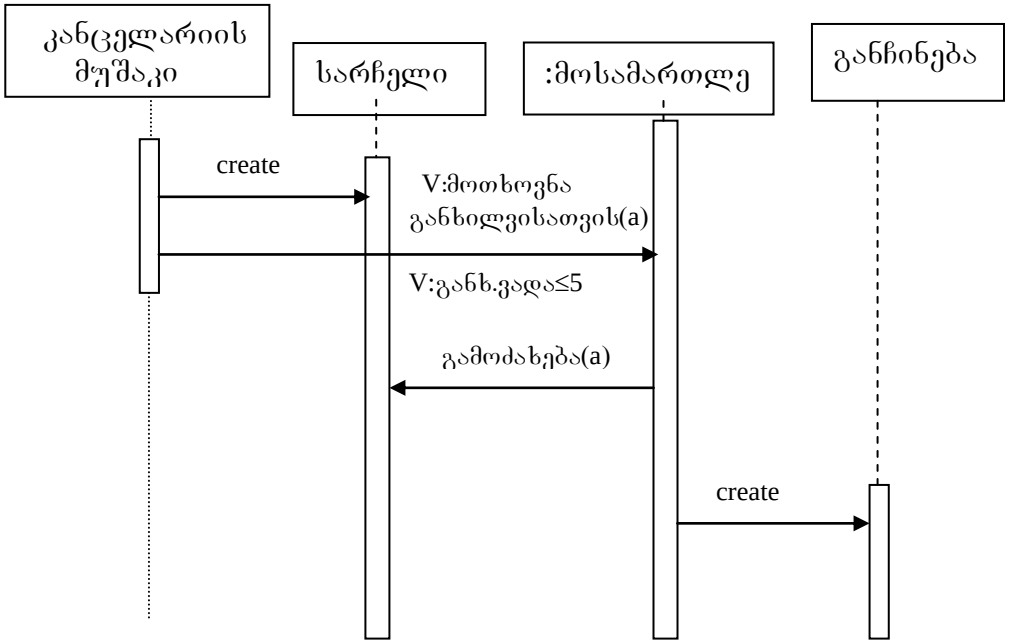
3. დავადგინოთ ტრასირების მიმართება სხვადასხვა დონის დამაკავშირებელ ელემენტებს შორის.
4. თუ ჩვენ ვეყრდნობით სისტემის იდეოლოგიას ხუთი არქიტექტურული სახეობით, მაშინ შესაძლებელია ოთხი ტიპიური სიტუაცია;

- პრეცედენტები და მათი რეალიზაცია. მოდელში პრეცედენტების თვალთახედვით პრეცედენტები ტრასირდებიან მოდელის კოოპერაციებთან პროექტირების თვალთახედვით.
- კოოპერაციები და მათი რეალიზაცია. კოოპერაციები ტრასირდებიან კლასებთან, რომლებიც ერთობლივად ფუნქციონირებენ მოცემული კოოპერაციის განსახორციელებლად.
- კომპონენტები და მათი პროექტირება. რეალიზაციის მოდელის კომპონენტები ტრასირდებიან პროექტირების თვალთახედვით მოდელის ელემენტებთან.
- კვანძები და მათი კომპონენტები. კვანძები განლაგების მოდელიდან ტრასირდებიან კომპონენტებთან რეალიზაციის მოდელიდან.

უპირატესობა ასეთი მიდგომისა იმაშია, რომ სხვადასხვა დონის აბსტრაქციის დიაგრამები ნაკლებათ არიან ერთმანათთან დაკავშირებული. ცვლილება ერთ მოდელში არ მოახდენს ძალიან დიდ გავლენას დანარჩენებზე. ნაკლოვანება კი იმაში მდგომარეობს, რომ საჭირო ხდება დამატებითი ძალისხმევა მოდელების და მათი დიაგრამების სინქრონიზაციაზე. ეს განსაკუთრებით იჩენს თავს თუ მოდელები მიეკუთვნებიან სასიცოცხლო ციკლის სხვადასხვა ფაზებს.

მაგალითისათვის განვიხილოთ სამოქალაქო სამართალწარმოების მოდელირების პროცესი. ერთ ერთ ძირითად პრეცედენტს ასეთ სისტემაში წარმოადგენს სარჩელის განხილვა. ანალიტიკოსს ან საბოლაო მომხმარებელს დასჭირდებათ

ურთიერთქმედების დიაგრამები აბსტრაქციის მაღალ დონეზე, რომლებიც სქემატურად გამოსახავენ ამ პროცესს ისე, როგორც ეს ნახ. 5.2. –ზე არის მოყვანილი.

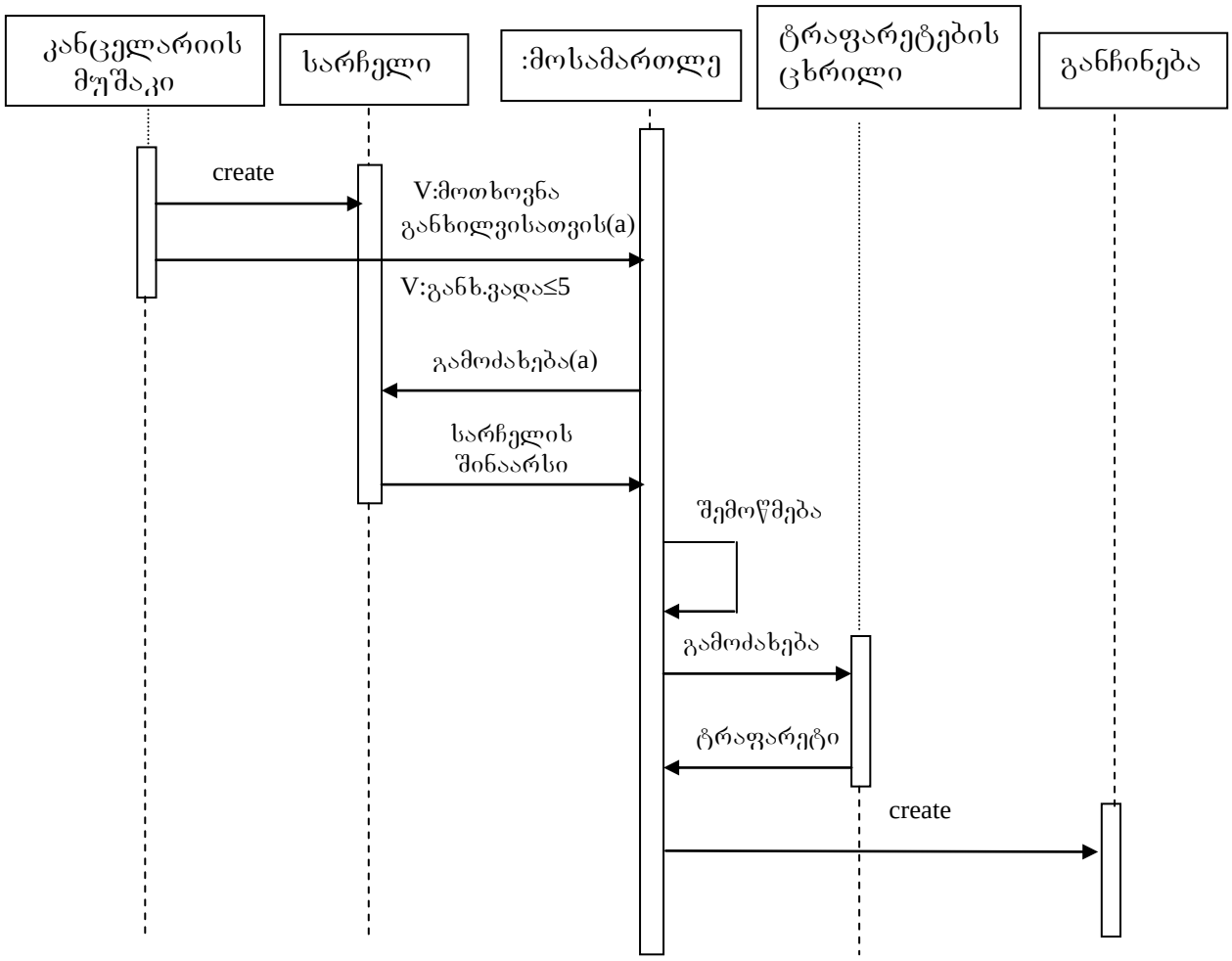


ნახ.5.2.

მეორეს მხრივ, პროგრამისტი, რომელიც პასუხს აგებს ამ სცენარის რეალიზებაზე, უნდა ისარგებლოს უფრო დამუშავებული დიაგრამით, რომელშიც უნდა გაფართოვდეს ზოგიერთი შეტყობინებები და დაემატოს ახალი მოქმედი პირები. ასეთი მაგალითი მოყვანილია ნახ. 5.3.-ზე.

ორივე დიაგრამა მიეკუთვნება ერთი და იმავე მოდელს, მაგრამ ახდენენ სხვადასხვა დეტალიზაციის დონის დემონსტრირებას.
კონცეპტუალური კავშირების მოდელირება ელემენტებს შორის, რომლებიც სხვადასხვა მოდელებში არიან, შესაძლებელია ტრასირების დამოკიდებულებით. ტრასირება წარმოიდგინება სტერეოტიპული დამოკიდებულებით. ხშირად ყურადღებას არ აქცევენ ასეთი დამოკიდებულების მიმართულებას, თუმცა ჩვეულებრივ ისარი მიუთითებს უფრო ადრეულ ობიექტს. უფრო

ხშირად ტრასირების მიმართება გამოიყენება იმისათვის, რომ უჩვენონ გზა მოთხოვნიდან რეალიზაციამდე.



ნახ.5.3.

5.4. რაციონალური უნიფიცირებული პროცესი

ავტომატიზებული სისტემის დამუშავება დადგენილ ვადებში და წინასწარ შედგენილი სმეტის ფარგლებში, რომელიც შეესაბამება მომხმარებლის მოთხოვნებს, მოითხოვს მოწესრიგებულ მიდგომას იმასთან, თუ როგორ უნდა განაწილდნენ სამუშაოები და პასუხისმგებლობები ორგანიზაციაში, რომელიც დაკავებულია წარმოების პროცესით. პროცესს უწოდებენ ბიჯების ნაწილობრივად მოწესრიგებულ სიმრავლეს, რომლებიც მიმართულია გარკვეული მიზნის მისაღწევად. დამუშავების არსებული მეთოდიკებიდან

საუკეთესოდ მიჩნეულია რაციონალური უნიფიცირებული პროცესი, რომელიც ყველაზე კარგად არის მისადაგებული **UML** ტექნოლოგიასთან და შესაძლებელია ადაპტირებულ იქნას სხვადასხვა დანიშნულების პროექტების ორგანიზებისათვის.

განვიხილოთ რაციონალური უნიფიცირებული პროცესის ძირითადი ელემენტები.

პროცესის ხასიათი. რაციონალური უნიფიცირებული პროცესი იტერაციულია. თუ გავითვალისწინებთ თანამედროვე სისტემების სირთულესა და განშლადობას, ასეთი წრფივი მიდგომა დამუშავებისადმი ხდება არარეალური. იტერაციული მიდგომა გულისხმობს თანდათანობით შეღწევას პრობლემის არსში მიმდევრობითი დაზუსტებითა და სულ უფრო მოცულობითი გადაწყვეტილების მიღებას რამოდენიმე ციკლის განმავლობაში. იტერაციული მიდგომისათვის დამახასიათებელია მოქნილობა, რომელიც საშუალებას იძლევა ჩავრთოთ ბიზნეს- მიზნებში ახალი მოთხოვნები ან ტაქტიკური ცვლილებები. მისი გამოყენება შესაძლებლობას იძლევა გამოვავლინოთ და გამოვრიცხოთ რისკი, რომელიც დაკავშირებულია პროექტთან დამუშავების ადრეულ ეტაპებზე.

სამუშაოს არსი რაციონალური უნიფიცირებული პროცესის ფარგლებში – ეს მოდელის შექმნა და თანხლებაა. მოდელები, რომლებიც გამოხატულია **UML** ენაზე, გვაძლევენ სემანტიკურად გაჯერებულ წარმოდგენას დასამუშავებელ პროგრამულ კომპლექსზე.

რაციონალური უნიფიცირებული პროცესი ეყრდნობა ობიექტ-ორიენტირებულ მიდგომას, ყოველი მოდელი ობიექტ - ორიენტირებულია. მოდელები, რომლებიც გამოიყენებიან რაციონალური უნიფიცირებული პროცესის ფარგლებში ეფუძნება ობიექტების, კლასების და მათ შორის მიმართებების მცნებებს, ხოლო საერთო ნოტაციის სახით გამოიყენება **UML** ნოტაცია.

რაციონალური უნიფიცირებული პროცესი ბაზირდება უბრალო და გასაგებ არქიტექტურაზე, რომელიც უზრუნველყოფს კონცეპტუალურ ერთიანობას დამუშავების მთელ პროცესში, ამასთან ადაპტირდება სხვადასხვა სიტუაციებთან.

რაციონალური უნიფიცირებული პროცესი უზრუნველყოფს ხარისხის და რისკის მართვის ობიექტურ კონტროლს პროექტის შესრულების ყველა სტადიაზე. ხარისხის კონტროლი წარმოადგენს პროცესის განუყოფელ ნაწილს, მოიცავს ყველა სახის და მონაწილის სამუშაოებს. ამასთან გამოიყენება შედარების ობიექტური კრიტერიუმები და მეთოდები. რისკის მართვა ჩართულია პროცესში, ისე რომ შესაძლო დაბრკოლებები პროექტის წარმატებით დასრულების გზაზე გამოვლინდება და გამოირიცხება ადრეულ ეტაპებზე, როდესაც რეაგირებისათვის საკმარისი დროა.

ფაზები. ფაზა – ეს დროის მონაკვეთია პროცესის ორ მნიშვნელოვან საყრდენ წერტილებს შორის, რომელშიც უნდა მიღწეულ იქნას მკაფიოდ გამოხატული მიზნები, მომზადებული იქნას ესა თუ ის არტეფაქტები და მიღებული იქნას გადაწყვეტილება- საჭიროა შემდეგ ფაზაზე გადასვლა თუ არა. რაციონალური უნიფიცირებული პროცესი შედგება ოთხი ფაზისაგან:

1. დასაწყისი(**Inception**) – პროექტის ბიზნეს- მიზნების განსაზღვრა.
2. გამოკვლევა(**Elaboration**) - გეგმის და პროექტის არქიტექტურის დამუშავება.
3. აგება(**Construction**) - სისტემის თანდათანობით შექმნა.
4. დანერგვა(**Transition**) - საბოლაო მომხმარებლისათვის სისტემის დაყენება.

დასაწყისისა და გამოკვლევის ფაზები მოიცავენ დამუშავების პროცესის სასიცოცხლო ციკლის საპროექტო სტადიებს; აგებისა და დანერგვის ფაზები მიეკუთვნებიან წარმოებას.

ყოველი ფაზის შიგნით ხდება რამოდენიმე იტერაცია. იტერაცია წარმოადგენს დამუშავების სრულ ციკლს, ანალიზის დროს მოთხოვნების გამომუშავებიდან - რეალიზებითა და ტესტირებით დამთავრებული. საბოლაო შედეგს წარმოადგენს მზა პროდუქტის გამოშვება.

ყველა ფაზები და იტერაციები გულისხმობენ გარკვეულ ძალისხმევას რისკის შესამცირებლად. ყოველი ფაზის ბოლოს არის მკაფიოდ გამოხატული საყრდენი წერტილი, სადაც ფასდება, რა დონით არის მიღწეული დასახული მიზნები და საჭირო ხომ არ არის პროცესში ცვლილებების შეტანა, სანამ გადავიდოდეთ შემდეგ ეტაპზე.

დასაწყისი. მოცემულ სტადიაზე განისაზღვრება სისტემის მიზნები და პროექტის საზღვრები. მიზნების ანალიზი მოიცავს წარმატების კრიტერიუმის გამომუშავებას, აუცილებელ რესურსებს და გეგმის შედგენას, რომელშიც გამოხატულია ძირითადი საყრდენი წერტილები.

საწყისი ფაზის დასასრულს კიდევ ერთხელ ყურადღებით შეისწავლება პროექტის სასიცოცხლო ციკლი და მიიღება გადაწყვეტილება, ღირს თუ არა დავიწყოთ ფართო მასშტაბიანი დამუშავება.

გამოკვლევა. მოცემულ ეტაპზე უნდა გაანალიზდეს საგნობრივი სფერო, დავამუშაოთ სისტემის არქიტექტურული საფუძვლები, შევადგინოთ პროექტის გეგმა და გამოვრიცხოთ ყველაზე საშიში რისკები. არქიტექტურული გადაწყვეტილებები უნდა მიღწეულ იქნას მაშინ, როდესაც გაირკვევა სისტემის სტრუქტურა მთლიანობაში, ე.ი. მოთხოვნათა უდიდესი ნაწილი უკვე ფორმულირებულია.

გამოკვლევის ფაზის დასასრულს დეტალურად შეისწავლება პროექტის მიზნები, მისი საზღვრები, არჩეული არქიტექტურა და

ძირითადი რისკების მართვის მეთოდები, რომლის შემდეგაც მიიღება გადაწყვეტილება იმის შესახებ დავიწყოთ აგება თუ არა.

აგება. აგების ფაზაში თანდათან ან იტერაციულად მუშავდება პროდუქტი, რომელიც შესაძლებელი იქნება დაინერგოს. მოცემულ ეტაპზე აღიწერება დარჩენილი მოთხოვნები და მიღების კრიტერიუმები, მთავრდება დამუშავება და პროგრამული კომპლექსის ტესტირება.

აგების ფაზის დასასრულს მიიღება გადაწყვეტილება პროგრამების და მომხმარებლების დანერგვისათვის მზადყოფნის შესახებ.

დანერგვა. დანერგვის ფაზაში პროგრამული უზრუნველყოფა გადაეცემა მომხმარებელს. ამ დროს ხშირად წარმოიშვება დამატებითი დამუშავების მოთხოვნის საკითხები სისტემის გაწყობასთან დაკავშირებით, შეცდომების გასასწორებლად, რომლებიც მანამდე არ იყო შემჩნეული და რიგი ფუნქციების საბოლაოდ გასაფორმებლად, რომელთა რეალიზებაც გადადებული იყო.

დანერგვის ფაზის ბოლოს დგინდება მიღწეულია თუ არა პროექტის მიზნები და საჭიროა თუ არა დავიწყოთ დამუშავების ახალი ციკლი.

იტერაციები. ყოველი ფაზა რაციონალური უნივერსალური პროცესის შეიძლება დაიყოს იტერაციებათ. იტერაცია ეს დასრულებული ეტაპია, რომლის შედეგად გამომუშავდება პროექტის ვერსია, რომელიც ახდენს დაგეგმილი ფუნქციების ნაწილის რეალიზაციას. შემდეგ ეს ვერსია იტერაციიდან იტერაციამდე ფართოვდება მზა პროდუქციის მიღებამდე. ყოველი იტერაციისას სრულდება განსაკუთრებული სამუშაო პროცესები, თუმცა სხვადასხვა ფაზებში ძირითადი დაწოლა ხდება განსხვავებულ

სამუშაოებზე. საწყის ფაზაში მთავარ ამოცანას წარმოადგენს მოთხოვნათა გამომუშავება, ხოლო დანერგვის ფაზაში – განლაგება.

დამუშავების ციკლები. ოთხივე ძირითად ფაზაზე გასვლას უწოდებენ დამუშავების ციკლს. ყოველი ციკლი მთავრდება სისტემის ვერსიის გენერაციით. პირველ გასვლას ყველა ოთხივე ფაზაზე უწოდებენ დამუშავების საწყის ციკლს. თუ ამის შემდეგ მუშაობა პროექტზე არ წყდება, მაშინ მიღებული პროექტის განვითარება გრძელდება და ხელახლა გაივლის იმავე ფაზებს: საწყის, გამოკვლევის, აგების, და დანერგვის. რის შედეგადაც სისტემა განიცდის ევოლუციას, ამიტომ ყოველ ციკლს, რომელიც მოსდევს საწყისს, უწოდებენ ევოლუციურს.

სამუშაო პროცესები. რაციონალური უნიფიცირებული პროცესი შედგება ცხრა სამუშაო პროცესისაგან:

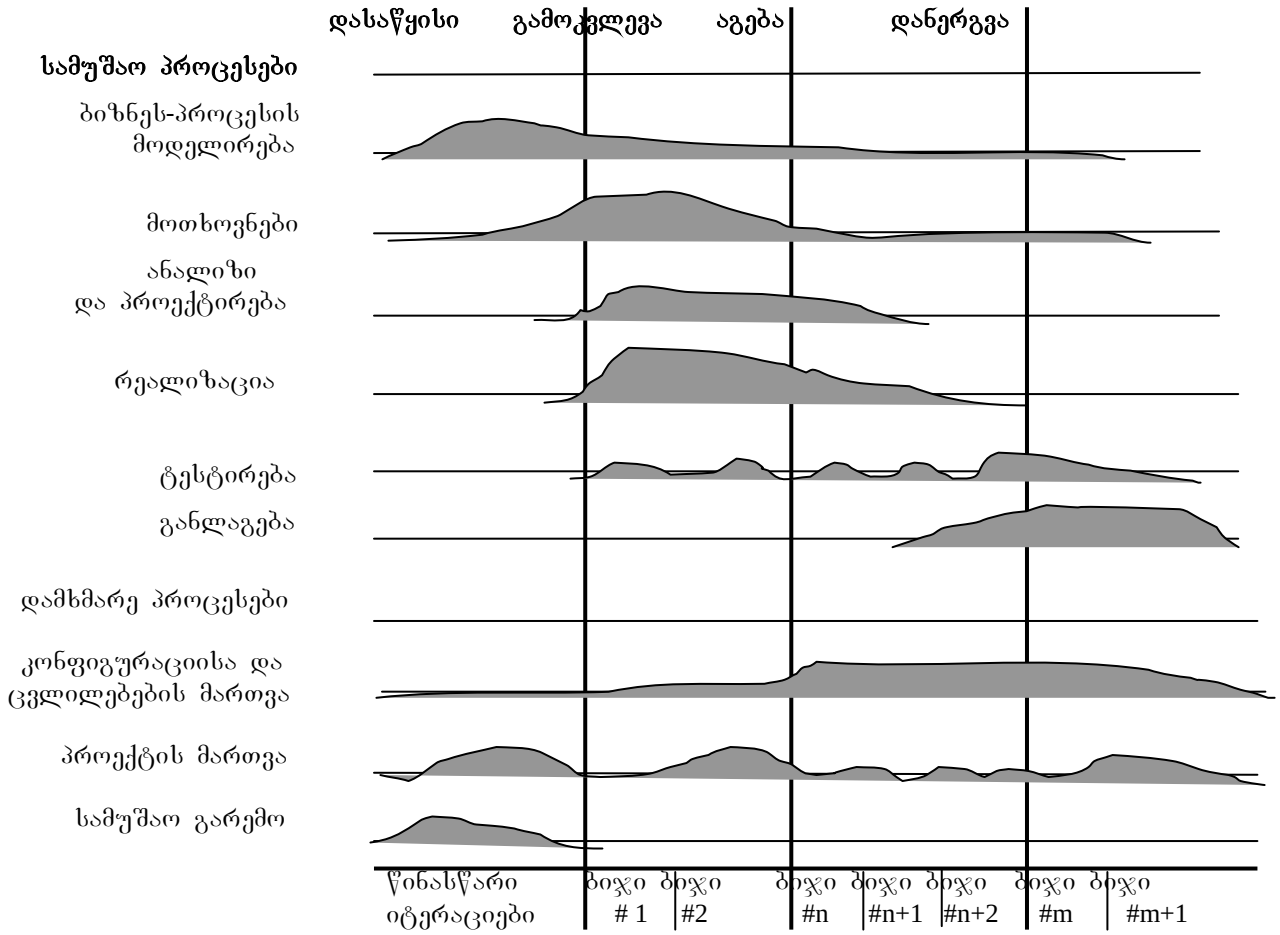
- ბიზნეს-მიზნების მოდელირება – აღიწერება ორგანიზაციის სტრუქტურა და დინამიკა;
- მოთხოვნების დამუშავება – აღიწერება პრეცენდენტებზე დაფუძნებული მოთხოვნათა დადგენის მეთოდი;
- დაპროექტება და ანალიზი – აღიწერება სისტემის სხვადასხვა სახის არქიტექტურა;
- რეალიზაცია – ხდება პროგრამების დამუშავება, ავტონომიური ტესტირება და ინტეგრაცია;
- ტესტირება – აღიწერება ტესტის სცენარები, პრეცენდენტები და მეტრიკა ცდომილებათა რიცხვის განსაზღვრისათვის;
- განლაგება – მოიცავს სისტემის კონფიგურირებას;
- კონფიგურაციის მართვა – ცვლილებების მართვა და პროექტის არტეფაქტების მთლიანობის უზრუნველყოფა;
- პროექტის მართვა – აღწერს სხვადასხვა სტრატეგიას იტერაციულ პროცესთან მუშაობისათვის;

- გარემოს მართვა – განიხილება ინფრასტრუქტურის საკითხები, რომლებიც აუცილებელია სისტემის მუშაობისათვის.

ყოველი სამუშაო პროცესის შიგნით თავმოყრილია ერთმანეთთან დაკავშირებული არტეფაქტები და მოღვაწეობები. **არტეფაქტები** – ეს გარკვეული დოკუმენტია, ანგარიში ან შესრულებადი პროგრამა, რომლებიც იქმნება, ხოლო შემდეგ გარდაიქმნება ან გამოიყენება. ტერმინით **მოღვაწეობა** აღიწერება ამოცანები – მოფიქრება, შესრულება, პროექტის ანალიზი, რომლებიც გადაწყდება თანამშრომლების მიერ არტეფაქტების შექმნის ან მოდიფიკაციის მიზნით, ასევე რეკომენდაციები და საშუალებები ამ ამოცანების გადასაწყვეტად.

არტეფაქტები. რაციონალურ უნივერსალურ პროცესში ყოველ მოღვაწეობასთან დაკავშირებულია არტეფაქტები, რომლებიც ან მიეწოდება შესასვლელზე, ან მიიღება გამოსასვლელზე. არტეფაქტები გამოიყენებიან როგორც საწყისი მონაცემები შემდგომი მოღვაწეობისათვის, შეიცავენ საცნობარო მონაცემებს პროექტის შესახებ.

მოდელი. მოდელი ეს არტეფაქტების ყველაზე მნიშვნელოვანი სახეა რაციონალურ უნივერსალურ პროცესში. მოდელი რეალობის გამარტივებაა; იგი იქმნება დასამუშავებელი სისტემის უკეთესი გაგებისათვის. რაციონალური უნიფიცირებული პროცესი მოიცავს ცხრა მოდელს, რომლებიც ერთობლივად მოიცავენ ყველა მნიშვნელოვან გადაწყვეტილებებს დასამუშავებელი სისტემის ვიზუალირების, სპეციფიცირების, კონსტრუირებისა და დოკუმენტირებისათვის:



- ბიზნეს-პროცესების მოდელი – ახდენს ორგანიზაციის აბსტრაქციის ფორმალიზებას;
- საპრობლემო სფეროს მოდელი – ახდენს სისტემის კონტექსის ფორმალიზებას;
- პრეცედენტების მოდელი – ახდენს სისტემისადმი ფუნქციონალური მოთხოვნების ფორმალიზებას;
- ანალიტიკური მოდელი (არ არის აუცილებელი) – ახდენს პროექტის იდეის ფორმალიზებას;
- საპროექტო მოდელი – ახდენს საპრობლემო სფეროს ლექსიკონის და გადაწყვეტის სფეროს ფორმალიზებას;
- პროცესების მოდელი (არ არის აუცილებელი) – ახდენს სისტემის პარალელიზმისა და სინქრონიზაციის მექანიზმების ფორმალიზებას;

- განლაგების მოდელი – ახდენს აპარატული საშუალებების ტოპოლოგიის ფორმალიზებას, რომლებზეც სრულდება სისტემა;
- რეალიზაციის მოდელი – აღწერს ნაწილებს, რომლებისგანაც ფიზიკურად აიწყობა სისტემა;
- ტესტირების მოდელი – ახდენს სისტემის გასინჯვისა და მიღების საშუალებების ფორმალიზებას.

სახე – ეს მოდელის ერთერთი პროექციაა. რაციონალურ უნივერსალურ პროცესში არსებობს ხუთი ერთმანეთთან მჭიდროდ დაკავშირებული სისტემური არქიტექტურის სახე: პროექტირების, პროცესების, განლაგების, რეალიზაციისა და პრეცენდენტების თვალსაზრისით.

სხვა არტეფაქტები. არტეფაქტები რაციონალურ უნიფიცირებულ პროცესში იყოფიან ორ ჯგუფად: ადმინისტრაციული და ტექნიკური. ტექნიკური არტეფაქტები, თავის მხრივ, იყოფიან ოთხ დიდ ქვეჯგუფად:

- მოთხოვნათა ჯგუფი – აღწერს, რას უნდა აკეთებდეს სისტემა;
- პროექტირების ჯგუფი – აღწერს, როგორ უნდა იქნას აგებული სისტემა;
- რეალიზების ჯგუფი – აღწერს, დამუშავებული პროგრამული კომპონენტების აწყობას.
- განლაგების ჯგუფი – შეიცავს ყველა მონაცემებს, რომლებიც აუცილებელია წარმოდგენილი სისტემის კონფიგურირებისათვის.

მოთხოვნათა ჯგუფში ჩართულია ინფორმაცია იმის შესახებ, თუ რას უნდა აკეთებდეს სისტემა. ამ არტეფაქტების შემადგენლობაში შეიძლება იყოს, არაფუნქციონალური მოთხოვნების, საპრობლემო სფეროს პრეცედენტების მოდელი და მომხმარებლის მოთხოვნათა გამოსახვის სხვა ფორმები, მათ შორის მაკეტები, ინტერფეისების პროტოტიპები, იურიდიული შეზღუდვები და ა.შ.

პროექტირების ჯგუფი შეიცავს ინფორმაციას იმის შესახებ, თუ როგორ უნდა იქნას აგებული სისტემა დროსა და ბიუჯეტში, განმეორებითი გამოყენების, ხარისხისადმი მოთხოვნებისა და ა.შ. შეზღუდვების გათვალისწინებით. აქ შედიან საპროექტო მოდელი, ტესტირების მოდელი და მომხმარებლის მოთხოვნათა გამოსახვის სხვა ფორმები, მათ შორის პროტოტიპები და შესრულებადი არქიტექტურები.

რეალიზაციის ჯგუფს მიეკუთვნება მთელი ინფორმაცია პროგრამული ელემენტების შესახებ, რომლებისგანაც შესდგება სისტემა, მათ შორის საწყისი კოდი პროგრამირების სხვადასხვა ენებზე, კონფიგურაციული ფაილები, მონაცემთა ფაილები, პროგრამული კომპონენტები და ა.შ., ასევე ინფორმაცია იმის შესახებ, თუ როგორ უნდა აიწყოს სისტემა.

განლაგების ჯგუფი ატყობინებს იმის შესახებ, თუ როგორ არის დაყოფილი პროგრამული კომპლექსი პაკეტებათ, როგორი სახით მიეწოდება, როგორ ყენდება და გაიშვება დამკვეთის მოედანზე.

ლიტერატურა

1. Джим Арлоу и Айла Нейштадт, **UML 2** н Унифицированный процесс. 2-е издание, Практичеслий объектно- ориентированный анализ и проектирование, Санкт-Петербург - Москва, 2008.
2. Буч Г., Рамбо Д., Джекобсон А. Язык **UML**. Руководство пользователя.// Серия “Объектно- ориентированные технологии в программировании”. Москва, 2004.
3. **Леоненков. Самоучитель UML. UML** Teach Yourself. **...** Особенности реализации языка **UML** в CASE-инструментарии Rational Rose 98/2000 · Заключение · Каталог · Индекс раздела. khpi-iip.mipk.kharkiv.edu/library/case/leon/index.html - 3k –
4. სასამართლო საქმეთა წარმოების ქსელური კომპიუტერული სისტემის დამუშავების პროექტი. მ. ახობაძე, გ. გოგიჩაიშვილი, გ. სურგულაძე, თ. სუხიაშვილი, გ. ღვინეფაძე // სტუ- ს შრომები, 2004, № 1(451) გვ.100- 104.
5. თ. სუხიაშვიული .სასამართლო სამოქალაქო სამართლის საქმეთა მდგომარეობათა ანალიზი სამართალწარმოების ავტომატიზებულ სისტემაში. // ინტელექტი, 2004.
6. სუხიაშვილი თ. ავტომატიზებული მართვის თეორიული საფუძვლები. მონოგრაფია. დამტკიცებულია სტუ-ს სამეცნიერო-ტექნიკური საბჭოს მიერ. გამომცემლობა “ტექნიკური უნივერსიტეტი”, 2005, 210 გვ.
7. სისტემების ობიექტ-ორიენტირებული ანალიზი. დამხმარე სახელმძღვანელო. დამტკიცებულია სტუ-ს სარედაქციო-საგამომცემლო საბჭოს მიერ. გამომცემლობა “ტექნიკური უნივერსიტეტი”, 2008, 70 გვ.
8. სისტემების ობიექტზე ორიენტირებული ანალიზი და დაპროექტება. სახელმძღვანელო. დამტკიცებულია სტუ-ს სარედაქციო-საგამომცემლო საბჭოს მიერ. გამომცემლობა “ტექნიკური უნივერსიტეტი”, 2009, 170 გვ.

9. სისტემების ობიექტზე ორიენტირებული ანალიზი. დამხმარე სახელმძღვანელო. ლაბორატორიული პრაქტიკუმი. დამტკიცებულია სტუ-ს სარედაქციო-საგამომცემლო საბჭოს მიერ. გამომცემლობა "ტექნიკური უნივერსიტეტი", 20013, 90 გვ.
10. სისტემების ობიექტზე ორიენტირებული დაპროექტება. დამხმარე სახელმძღვანელო. ლაბორატორიული პრაქტიკუმი. დამტკიცებულია სტუ-ს სარედაქციო-საგამომცემლო საბჭოს მიერ. გამომცემლობა "ტექნიკური უნივერსიტეტი", 20013, 95 გვ.
11. პროგრამული სისტემის დამუშავების CASE საშუალებები. დამხმარე სახელმძღვანელო. ლაბორატორიული პრაქტიკუმი. 20013, 130 გვ.
12. სისტემების ობიექტ-ორიენტირებული დაპროექტება. საკურსო პროექტის მეთოდური მითითება. თბილისი, "ტექნიკური უნივერსიტეტი". 2016, 98გვ.