გ. გოგიჩაიშვილი, ე. თურქია

პროგრამული უზრუნველყოფის რეალიზაცია Rational Rose ინსტრუმენტის ბაზაზე

თბილისი, 2009

უაკ 681.3

დამხმარე სახელმძღვანელო გათვალისწინებულია სისტემების ობიექტ-ორიენტირებული ანალიზისა და დაპროექტების ინსტრუმენტული საშუალების გასაცნობად.

აღწერილია ავტომატიზებული დაპროექტებისა და პროგრამული უზრუნველყოფის რეალიზაციის ინსტრუმენტული საშუალება Rational Rose, რომელიც უნიფიცირებული მოდელირების ენის გრაფიკული რედაქტორია.

დამხმარე სახელმძღვანელო განკუთვნილია მართვის ავტომატიზებული სისტემების სპეციალობის სტუდენტებისთვის.

რეცენზენტი: პროფ. გ. სურგულაძე

სარჩევი

შესავალი 4
1. სისტემების დაპროექტების არსი..... 5
2. სისტემების ავტომატიზებული დაპროექტებისა და დამუშავების ტექნოლოგია - CASE 9
3. სისტემების დაპროექტება UML ენის გამოყენებით.....12
3.1. CASE-ტექნოლოგიის ინსტრუმენტული საშუალება...12
RATIONAL ROSE 12
3.2. RATIONAL ROSE სისტემის მოდელების აღწერა.....16
3.3. პროგრამულ სისტემებთან მუშაობა 36
3.3.1 კლასთა დიაგრამიდან პროგრამული კოდის მიღება.....36
3.3.2. პროგრამული კოდიდან კლასთა დიაგრამის გენერაცია.....40
4. მითითებები საკურსო პროექტის შესასრულებლად 42
გამოყენებული ლიტერატურა.....61

შესავალი

თანამედროვე საინფორმაციო ტექნოლოგიები გვთავაზობს მართვის ავტომატიზებული სისტემების კონცეპტუალურ აგებას ვიზუალური მოდელირების პრინციპებით, რომელთა შესაძლებლობაშია ინსტრუმენტული საშუალებების გამოყენებით საპრობლემო არის ვიზუალური მოდელირება და ამ მოდელის ანალიზი სისტემის დამუშავების ყველა ეტაპზე.

ასეთი სისტემების ჯგუფში შედის სისტემების ავტომატიზებული დაპროექტებისა და დამუშავების ტექნოლოგია (CASE – Computer Aided System Engineering), მოდელზე ორიენტირებული არქიტექტურა (MDA-Model Driven Architecture) და დანართების სწრაფი დამუშავების ტექნოლოგია (RAD- Rapid Application Development), რომლებსაც საფუძვლად უდევს უნიფიცირებული მოდელირების ენა – UML (Unified Modeling Language).

UML ენა სტანდარტული ნოტაციაა პროგრამული სისტემების ვიზუალური მოდელირებისთვის. იგი, შემუშავებულია Object Managing Group (OMG) კონსორციუმის მიერ. ამ ტექნოლოგიების საშუალებით შესაძლებელია მართვის ობიექტის სტრუქტურის სრული სასიცოცხლო ციკლის ეტაპობრივი აღწერა, მოდელების გრაფიკული წარმოდგენიდან - პროგრამულ კოდამდე (მოთხოვნების განსაზღვრა, პროცესების მოდელირება და შესაბამისი პროგრამული ჩონჩხის მიღება).

1. სისტემების დაპროექტების არსი

საინფორმაციო სისტემების მართვის პროცესების ავტომატიზაცია, ძირითადად, ეხება ორგანიზაციის საქმეთა წარმოებისა და დოკუმენტბრუნვის ერთობლივი პროგრამული უზრუნველყოფის სისტემას. იგი, უზრუნველყოფს კონკრეტული ორგანიზაციისთვის ინფორმაციული ბაზის შევსებასა და ამ ინფორმაციის მართვას ორგანიზაციის მოთხოვნების შესაბამისად.

საწარმოების, ფირმების, კომპანიების, ბანკების, ბირჟის და ზოგადად, ორგანიზაციული სტრუქტურების ფუნქციონირების ხარისხის ამაღლება წარმოუდგენელია მათში მიმდინარე პროცესების ავტომატიზაციის გარეშე, რომლებიც დღესდღეობით ხორციელდება კომპიუტერის დახმარებით. ამავდროულად, ამ პროცესებში აქტიურ მონაწილეობას იღებს ადამიანი, რაც იწვევს ინფორმაციის დამუშავებისა და მართვის ავტომატიზებული (ადამიან-მანქანური) სისტემების შექმნის აუცილებლობას. ავტომატიზებული სისტემა არის პროგრამული პროდუქტი ანუ პროგრამული კომპლექსი, რომელიც ახორციელებს ინფორმაციის დამუშავებასა და მართვას. ამ პროცესების ავტომატიზაციას მოაქვს სასურველი ეფექტი, მხოლოდ იმ შემთხვევაში, როდესაც შექმნილი პროგრამული პროდუქტი, ან როგორც მას უწოდებენ, ავტომატიზებულ სისტემებში-პროგრამული უზრუნველყოფა, სრულად დააკმაყოფილებს მომხმარებლის მოთხოვნებსა და აუცილებლობებს.

მომხმარებლის მოთხოვნების განსაზღვრა და მისი ასახვა პროგრამულ უზრუნველყოფაში არის კომპლექსური პროცესი, რომელშიც მონაწილეობენ ბიზნეს-ანალიტიკოსები (ექსპერტები),

რომლებიც აყალიბებენ საავტომატიზაციო სისტემის ძირითად მოთხოვნებს და ტექნიკური დამმუშავებლები (დამპროექტებლები, პროგრამისტები), რომლებიც ამუშავებენ და ახორციელებენ ამ მოთხოვნების შესრულებას საინფორმაციო ტექნოლოგიის თვალსაზრისით. ამდენად, ნებისმიერი ეკონომიკური სისტემის ავტომატიზაციისთვის და პროგრამული მოდულის რეალიზაციისთვის, პირველადი პროცესი არის ამ სისტემის დაპროექტება და მოდელირება.

თანამედროვე ორგანიზაციული სისტემების უმეტესობა დღითიდღე ფართოვდება ინფრასტრუქტურული და ინფორმაციული თვალსაზრისით, შესაბამისად რთულდება ორგანიზაციული სისტემების მართვა. საჭირო ხდება მართვის ფუნქციების ზუსტი გადანაწილება შემსრულებლებზე, საქმის წარმოების ჯაჭვის უწყვეტი შესრულება და კონტროლი, შემსრულებლების დროული და მუდმივი ინფორმაციული ურთიერთგაცვლა და ა.შ [1].

ასეთ სისტემებს აკუთვნებენ რთული სისტემების კლასს. რთული სისტემა შეიცავს მრავალ სხვადასხვა ტიპის ობიექტს და შესაბამისად, ინფორმაციის დამუშავების რთულ პროცესებს.

შესაძლებელია გამოვყოთ რთული სისტემის რამდენიმე თვისება:

რთული სისტემები იერარქიულია და შედგება ურთიერთდამოკიდებული ქვესისტემებისგან, რომლებიც თავის მხრივ ასევე შესაძლებელია დაყოფილ იქნეს ქვესისტემებად ყველაზე დაბალ დონემდე. რთული სისტემის არქიტექტურა აიგება ასევე კომპონენტებისა და მათი იერარქიული დამოკიდებულებისგან. იმის

არჩევა, თუ რომელი კომპონენტი ითვლება ელემენტარულად (მარტივად), ფარდობითია და უმეტეს შემთხვევაში განიხილება კვლევის შეხედულების მიხედვით. ერთი დამკვირვებლისთვის დაბალი დონე შესაძლებელია აღმოჩნდეს მეორესთვის მაღალი და აშ.

იერარქიული სისტემები, როგორც წესი, შედგება სხვადასხვაგვარად კომბინირებული და ორგანიზებული რამდენიმე ტიპის ქვესისტემისგან. ნებისმიერი რთული მუშა სისტემა წარმოადგენს შედარებით მარტივი მუშა სისტემის განვითარების შედეგს. რთული სისტემის დაპროექტება ნულიდან არასოდეს არ იმუშავებს, ამდენად საჭიროა დავიწყოთ მარტივი მუშა სისტემიდან. სისტემის განვითარების პროცესში ობიექტები, თავდაპირველად განიხილება როგორც რთული ობიექტები, რომლებიც შემდგომში ხდება ელემენტარული და მათგან ხდება შედარებით რთული სისტემის აგება. შეუძლებელია თავიდანვე სწორად შეიქმნას ელემენტალური ობიექტები: მათზე თავიდან საჭიროა მუშაობა, რათა გავიგოთ მეტი, სისტემის რეალურ ქცევაზე და შემდგომ მოვახდინოთ მათი სრულყოფა.

რთული სისტემების დაპროექტებისას, როგორც წესი, სისტემა იშლება ქვესისტემებად და თითოეული ქვესისტემა განიხილება შემდგომ ცალკეულად (დეკომპოზიცია).

არსებობს სისტემების დეკომპოზიციის ორი ხერხი: სტრუქტურული (ანუ ფუნქციონალური) და ობიექტური (ანუ კომპონენტური). პროგრამული სისტემების ფუნქციონალური დეკომპოზიციისას მისი სტრუქტურის აღწერა წარმოებს ბლოკ-

სქემების სახით, რომელთა კვანძი წარმოადგენს "დასამუშავებელ ცენტრებს", ხოლო კავშირები კვანძებს შორის, აღწერს მონაცემთა მოძრაობას.

ობიექტური ანუ კომპოტენტური დეკომპოზიციისას სისტემა იყოფა (ერთმანეთთან ურთიერთქმედ აქტიურ არსებად), ობიექტები ცვლიან შეტყობინებებს კლიენტ-სერვერ გარემოს ფარგლებში. შეტყობინება, რომელიც შეუძლია მიიღოს ობიექტმა განისაზღვრება მის ინტერფეისში [3, 4].

როგორც წესი, საინფორმაციო სისტემების დაპროექტებასა და მოდელირებაში განიხილება საკვლევი ობიექტის საქმეთა წარმოების პროცესების (ბიზნეს-პროცესების), ამ პროცესებში მონაწილე როლებისა და რესურსების დამოკიდებულება, ინფორმაციული და ლოგიკური ანალიზი და ა.შ.

საგნობრივი სფეროს დაპროექტებისას წარმოებს სისტემის სასიცოცხლო ციკლის მოდელის დეტალური აღწერა. სასიცოცხლო ციკლის მოდელი შეიცავს სისტემის სტრუქტურას, შემადგენელ პროცესებს, დავალებებსა და ქმედებებს, პროცესის დინამიკას, ფუნქციურ ოპერაციათა დეკომპოზიციას, ინფორმაციული პროცესის ანალიზს და ა.შ.

მოდელი ძირითადი პრობლემის ანუ საგნობრივი სფეროს აბსტრაქციაა. სფერო (domain) არის ის ფაქტობრივი არე, საიდანაც მოდის პრობლემა. მოდელი შედგება ობიექტებისგან, რომლებიც ურთიერთქმედებს შეტყობინებებით. ობიექტს გააჩნია ორი ძირითადი თვისება: ატრიბუტები (მონაცემები) და ქცევა (ოპერაციები), ხოლო

ერთგვაროვან ობიექტთა ერთობლიობას უწოდებენ კლასს. სხვაგვარად, რომ ვთქვათ კლასი შეიცავს ობიექტის ატრიბუტებს და ობიექტის ქცევას ამ ატრიბუტებზე (მეთოდები, ფუნქციები).

მოდელირების ერთ-ერთი პრინციპია - არა ერთი, არამედ რამდენიმე მოდელის გამოყენება. ამასთან მოდელები შესაძლებელია აიგოს ერთმანეთისგან დამოუკიდებლად, თუმცა აუცილებელია ისინი იყოს ურთიერთდაკავშირებული.

ამ თვალსაზრისით, საგნობრივი სფეროს დაპროექტებისას ძირითადად ხდება შემდეგი მოდელების აღწერა:

1. ორგანიზაციის (საწარმოს) ბიზნეს-პროცესები;
2. ბიზნეს-პროცესების შემსრულებელი პირები და მათი ფუნქციები, რომელიც ექვემდებარება ავტომატიზაციას;
3. ბიზნეს-არსები და მათი მდგომარეობები;
4. ბიზნეს-წესები.

2. სისტემების ავტომატიზებული დაპროექტებისა და დამუშავების ტექნოლოგია - CASE

ნებისმიერი სისტემის ავტომატიზაციის საწყისი ეტაპია ორგანიზაციული სტრუქტურის, საქმეთა წარმოების პროცესებისა და ფუნქციების (ბიზნეს-პროცესების), დოკუმენტების მოძრაობისა და ორგანიზაციულ მოთხოვნებზე ინფორმაციის შეგროვება და ანალიზი. ინფორმაციის შეგროვებასა და ანალიზში ძირითადად, მონაწილეობს ბიზნეს-სფეროს ანალიტიკოსები (ექსპერტები) და ტექნიკური

დამმუშავებლები (დამპროექტებლები, პროგრამისტები). მათი ერთობლივი მუშაობისთვის აუცილებელია შეთანხმებული ტერმინოლოგია, რთული სისტემების დეკომპოზიციური მოდელების აგების შესაძლებობა, მოდელების აგებისთვის ინსტრუმენტული საშუალებების გამოყენების სიმარტივე, და მოქნილობა.

სწორედ, ამ პრინციპით ჩამოყალიბდა სისტემების ავტომატიზებული დაპროექტებისა და დამუშავების ტექნოლოგია (CASE – Computer Aided System Engineering), რომელიც ობიექტ-ორიენტირებული და ვიზუალური მოდელირების მეთოდოლოგიაზე დაფუძნებული სპეციალური კლასის პროგრამულ-ტექნოლოგიური საშუალებაა რთული სისტემების ავტომატიზაციისთვის [3].

CASE ტექნოლოგია ე.წ. CASE-ინსტრუმენტული საშუალებების (CASE-Tools) კომპლექსია, რომელშიც ინტეგრირებულია უნიფიცირებული მოდელირების ენის (UML - Unified Modeling Language) დიაგრამების აგებისა და ანალიზის გრაფიკული საშუალებები, პროგრამული და მონაცემთა ბაზების მართვის სისტემები, კოდის გენერაციის, ტესტირების, დოკუმენტაციის და პროექტების მართვის საშუალებები. ამგვარად, CASE ტექნოლოგია მოიცავს შემდეგ ძირითად კომპონენტებს:

– ანალიზისა და დაპროექტების გრაფიკულ საშუალებებს, რომლითაც იქმნება და რედაქტირდება იერარქიულად დაკავშირებული დიაგრამები;

– მონაცემთა ბაზების დაპროექტების საშუალებას, რომელიც უზრუნველყოფს მონაცემთა მოდელირებას და მონაცემთა ბაზების სქემების გენერაციას;

– გუნდური მუშაობისას სხვადასხვა საქმიანი აგენტების მიერ მიღებული ინფორმაციების სინქრონიზაციასა და მეტა-მონაცემების კონტროლს;

– კლიენტ-სერვერული კონფიგურაციის მართვას;

– დოკუმენტაციის, ტესტირებისა და პროექტის მართვის საშუალებებს;

– ღია არქიტექტურას, რაც ინფორმაციისა და ფუნქციების ექსპორტ-იმპორტის ორგანიზების საშუალებას იძლევა [1,5].

ფაქტობრივად, CASE ტექნოლოგია ფართო კონცეფციაა, რომელიც მოიცავს მოდელზე ორიენტირებულ სხვადასხვა მიდგომებს პროგრამული უზრუნველყოფის დასამუშავებლად. იგი მხარს უჭერს მოდელებიდან პროგრამულ კოდში გადასვლის ავტომატიზაციას.

CASE ტექნოლოგიის საფუძველი არის UML ენა, რაც საშუალებას იძლევა გრაფიკული ვიზუალური კომპონენტების საშუალებით შეიქმნას გრაფიკული და დოკუმენტირებული სისტემის მოდელი.

UML ენა სტანდარტული ნოტაციაა პროგრამული სისტემების ვიზუალური მოდელირებისთვის. იგი, შემუშავებულია Object Managing Group (OMG) კონსორციუმის მიერ. ამ ტექნოლოგიების საშუალებით შესაძლებელია მართვის ობიექტის სტრუქტურის სრული სასიცოცხლო ციკლის ეტაპობრივი აღწერა, მოდელების გრაფიკული წარმოდგენიდან - პროგრამულ კოდამდე (მოთხოვნების განსაზღვრა,

პროცესების მოდელირება და შესაბამისი პროგრამული ჩონჩხის მიღება) [7].

3. სისტემების დაპროექტება UML ენის გამოყენებით

ცნობილია UML ენის მრავალი ინსტრუმენტული საშუალება, მაგალითად, MS Visio, Paradigm+, ასევე ჩაშენებულია Java, Visual Studio პაკეტებში. ერთ-ერთი საინტერესო ინსტრუმენტული საშუალებაა Rational Rose. იგი, იძლევა მოდელების თანმიმდევრულად აგების საშუალებას და საკმაოდ მოქნილი ინსტრუმენტული საშუალებაა CASE ტექნოლოგიის გამოყენებისთვის - მოდელიდან (კლასთა მოდელის) პროგრამული კოდის გენერაციისა და რევერსიულ დაპროექტების (პროგრამული კოდიდან კლასთა მოდელის შექმნა) თვალსაზრისით. პროგრამულ სისტემებთან კავშირისთვის გამოყენებაშია დაპროგრამების ენების ფართო სპექტრი (Java, Visual Basic, C++, XML და ა.შ.), ასევე მონაცემთა მოდელებისა და მონაცემთა ბაზების მართვის სისტემის რევერსიული დაპროექტების აგების საშუალებები (MS SQL Server, Oracle და ა.შ.) [3].

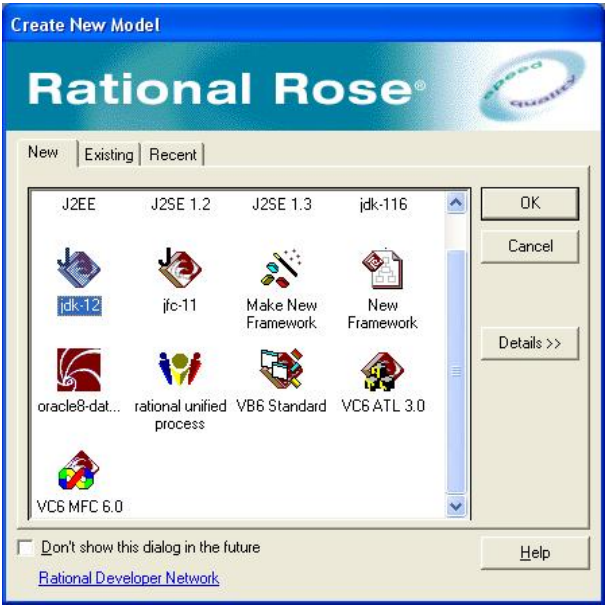
3.1. CASE-ტექნოლოგიის ინსტრუმენტული საშუალება RATIONAL ROSE

Rational Rose ინსტრუმენტის მოდელების დაპროექტების გარემოში შესასვლელად გამოიყენება ბრძანება Start-Programs-

Rational Rose Enteprise Edition (სურ. 3.1), რომლის შედეგად იხსნება დიალოგური ფანჯარა ახალი მოდელის შესაქმნელად (სურ. 3.2).

სურ. 3.1. Rational Rose ინსტრუმენტის გაშვება

იმ შემთხვევაში, თუ სისტემის დაპროექტება არ წარმოებს კონკრეტული პროგრამული სისტემის მიხედვით გამოიყენება ღილაკი “Cancel”, წინააღმდეგ შემთხვევაში საჭიროა არჩეულ იქნას შესაბამისი პროგრამული სისტემის სტანდარტული პლატფორმა (framework).



სურ. 3.2. დიალოგური ფანჯარა ახალი მოდელის შესაქმნელად

Rational Rose ინსტრუმენტში საავტომატიზაციო სისტემის დაპროექტება კლასიფიცირებულია სამ ძირითად ფაზად:

1. მოთხოვნილებათა განსაზღვრისა და ანალიზის ფაზა. ტექნიკურად, გამოყენებაშია Use Case View – შემთხვევების გამოყენების ბლოკი. ამ ეტაპზე ფაქტობრივად შესაძლებლობაშია UML ენის ექვსივე დიაგრამის (Use Case, Activity, Statechart, Sequence, Collaboration, Class Diagrams) აგება თითოეული პრეცედენტისთვის. ფაქტობრივად, Use Case View აღწერს როგორ გამოიყურება პროექტი გამოყენების თვალსაზრისით, ვის და სად შეაქვს მონაცემები, მონაცემების შეტანის შემდეგ რას აკეთებს პროგრამა და ვის გადასცემს შედეგს [2].

სისტემის use case view ნაწილი საშუალებას იძლევა შეიქმნას სისტემის სტრუქტურა, ძირითადი რესურსებისა და როლების ურთიერთდამოკიდებული ოპერაციების თვალსაზრისით. აქვე შესაძლებლობაშია აიგოს ამ ოპერაციების დეტალური, გაფართოებული ქვემოდელები.

2. დაპროექტების ფაზა. ტექნიკურად, ხორციელდება Logical View - ბლოკით, რომელიც ძირითად მოიცავს კლასთა დიაგრამებისა და კლასთაშორისი ასოციაციების აგებას, აღწერს პროგრამის ლოგიკას - კლასების, ობიექტების, კლასის თვისებების, მეთოდების და კლასთაშორისი ურთიერთდამოკიდებულების მიხედვით.

3. რეალიზაციისა და ტესტირების ფაზა. ამ ნაწილში შედის კომპონენტური და განთავსების დიაგრამების ფორმირება (Component View, Deployment View).

Component View აღწერს, რომელ ლოგიკურ კომპონენტებად ნაწილდება პროექტი, და რა არის მოთავსებული თითოეულ კომპონენტში. აქ ხდება მოდულების დაპროექტება და დამოკიდებულება მათ შორის, გადასვლები მთავარი პროგრამიდან ქვეპროგრამებში და ა.შ.

Deployment View აღწერს ფიზიკური კომპონენტების გამოყენებას სისტემაში. ინფრასტრუქტურა – მოწყობილობები, პროცესორები, სერვერი, ქსელი, კომუნიკაციური საშუალებები და ა.შ [2].

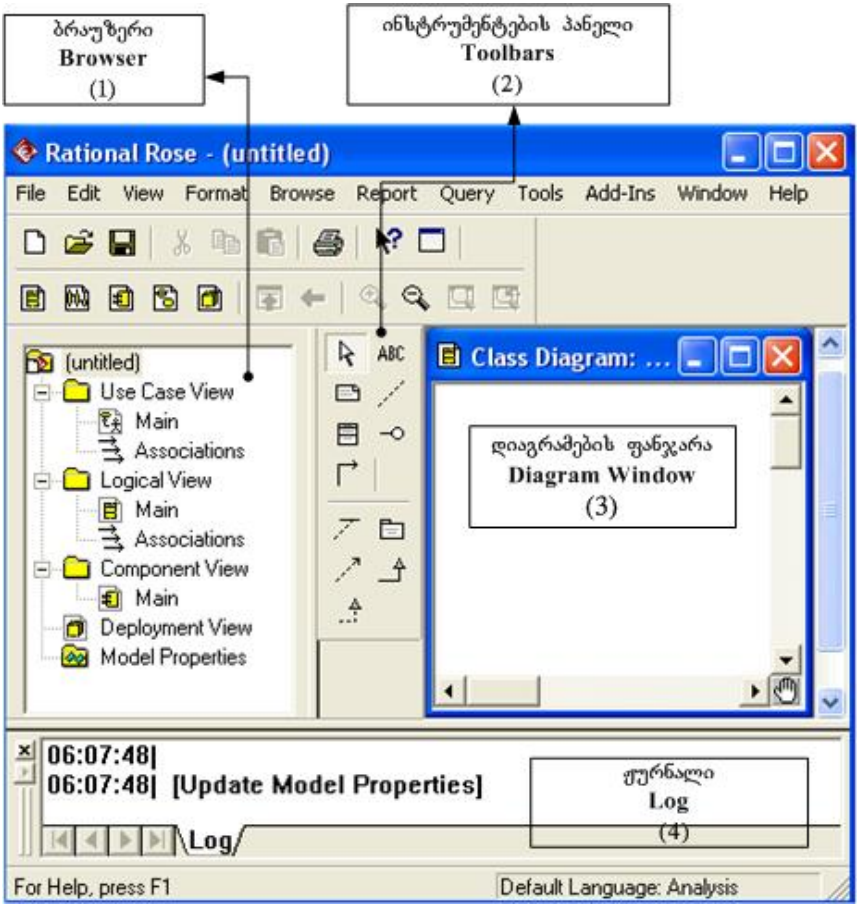
3.2. RATIONAL ROSE სისტემის მოდელების აღწერა

ამგვარად, Rational Rose სისტემაში დაჯგუფებულია დიაგრამები 4 ძირითად ბლოკად: Use Case View, Logical View, Component View და Deployment view, რომელიც მოთავსებულია ბრაუზერში (ნახ. 3.3.).

Rational Rose სისტემის ინტერფეისი შეიცავს შემდეგ ძირითად ელემენტებს: ბრაუზერი - Browser (1), ინსტრუმენტების პანელი - Toolbars (2), დიაგრამების ფანჯარა - Diagram Window (3), ჟურნალი - Log (4).

ბრაუზერში მოთავსებული თითოეული ბლოკი აგებულია იერარქიული ხის პრინციპით. დიაგრამების ფანჯრის ანუ სამუშაო გარემოს გამოტანა ხდება ფუნქციით – Main (მაუსის მარცხენა ღილაკის ორჯერ დაწკაპუნებით). თითოეულ დიაგრამას აქვს ინსტრუმენტული საშუალებების (ინსტრუმენტების პანელი) პიქტოგრამების (კომპონენტების) ჯგუფი.

თითოეულ პიქტოგრამას გააჩნია თვისებების ფანჯარა. მიმართვა მასზე ხდება კომპონენტის სამუშაო გარემოზე გადმოტანით და მარჯვენა ღილაკი მენიუდან ბრძანებაზე – Open spceification დაწკაპუნებით (რიგ შემთხვევაში, გამოძახება ასევე შესაძლებელია პიქტოგრამაზე მაუსის მარცხენა ღილაკის ორჯერ დაწკაპუნებითაც).



სურ.3.3. Rational Rose სისტემის ძირითადი ფანჯარა

განვიხილოთ თითოეული ბლოკი:

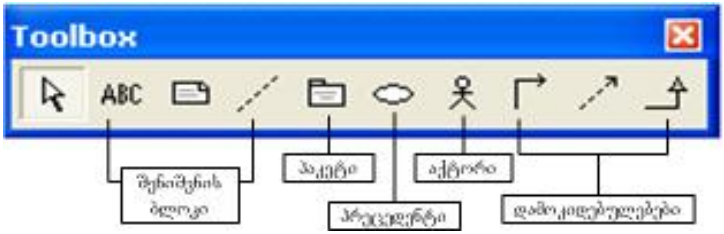
1. შემთხვევების გამოყენების ბლოკი - Use Case View

Use Case View ბლოკი სისტემის დაპროექტების საწყისი ეტაპია. მისი გამოყენება ძირითადად განკუთვნილია ანალიტიკოსების, მენეჯერებისა და პროგრამული სისტემის დამკვეთისთვის. მასში აღიწერება სისტემის ქცევის სცენარი, რომელსაც ასრულებს მომქმედი პირი. Use Case View ბლოკში უმთავრესია პრეცედენტების დიაგრამა - Use Case Diagram.

1.1. Use Case Diagram

Use Case დიაგრამის საშუალებით იქმნება იმ ოპერაციათა სია, რომელიც უნდა შეასრულოს სისტემამ. ხშირად მას ფუქნციების დიაგრამას უწოდებენ, ვინაიდან ამ დიაგრამის მიხედვით განისაზღვრება სისტემაზე მოთხოვნები და იქმნება სისტემის მიერ შესასრულებელ ფუნქციათა სიმრავლე.

სურათზე 3.4. ნაჩვენებია Use Case დიაგრამის ინსტრუმენტების პანელი.



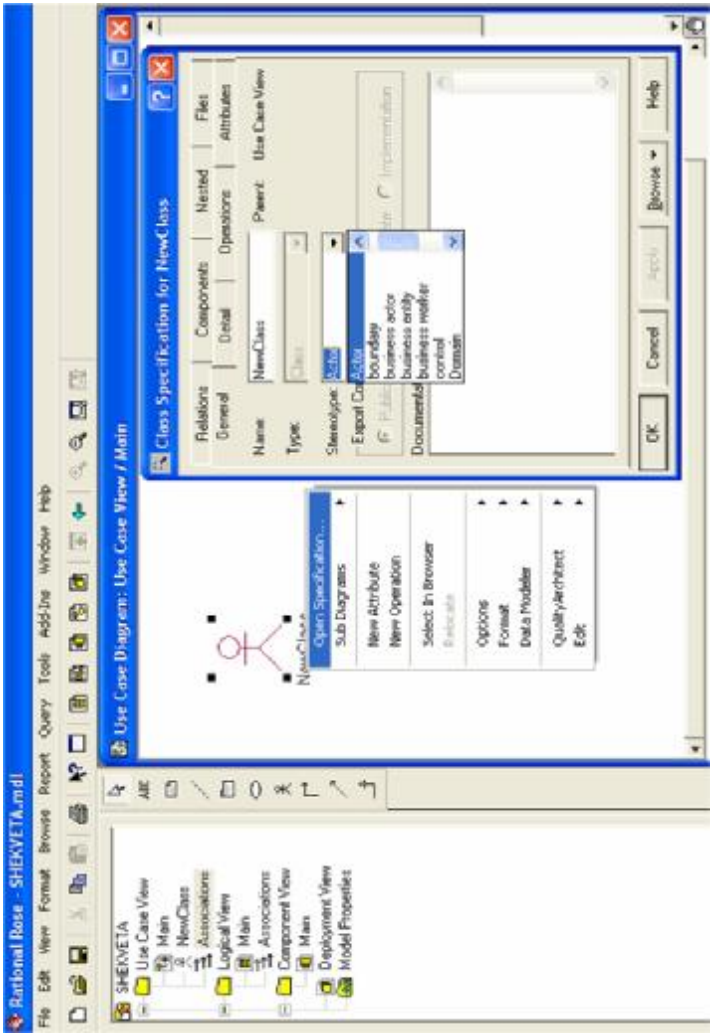
სურ. 3.4. ინსტრუმენტების პანელი - Use Case Diagram

Use Case დიაგრამაში აქცენტი კეთდება სამ ძირითად პიქტოგრამაზე, ესენია: ე.წ. კაცუნა, ოვალი და დამოკიდებულებები.

კაცუნა ზოგადად წარმოადგენს კლასს, თუმცა მისი გამოსახულება ანუ სტერეოტიპი მრავალგვარია (მაგალითად, აქტორი, ინტერფეისი, ცხრილი და ა.შ.). ამ კომპონენტის თვისებების ფანჯარა მოიცავს ატრიბუტების, ოპერაციებისა და სხვა დეტალურ აღწერებს.

3.5. სურათზე პიქტოგრამა-კაცუნას მაგალითზე ნაჩვენებია თვისებების ფორმირების ნიმუში.

პრეცედენტი გამოისახება ოვალის ფორმით. იგი წარმოადგენს იმ მოვლენათა ერთობლიობას, რომელიც სრულდება მაშინ, როდესაც აქტორი იყენებს სისტემას პროცესის შესასრულებლად. როგორც წესი, მოვლენის გამოყენება არ არის ინდივიდუალური ბიჯი ან ქმედება. იგი, შედარებით დიდი პროცესია და იშლება ქვედიაგრამებად. პრეცედენტები ტექნიკურად, შემდგომი ავტომატიზებული სისტემისთვის, შესაძლებელია წარმოვიდგინოთ, როგორც სისტემის დიალოგური ფორმები (პროგრამულ სისტემებში ეს ფორმები ასევე კლასებს წარმოადგენს).

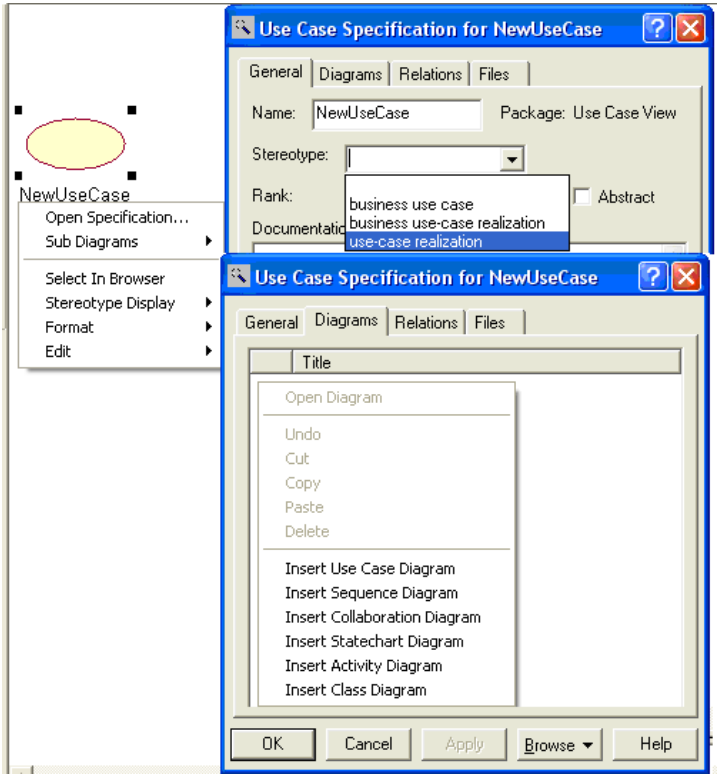


სურ.3.5. პიქტოგრამა-კაცუნას თვისებების ფორმირების ნიმუში

თითოეული დიალოგური ფორმის ფუნქციონირება, ლოგიკა და შემადგენელი კომპონენტები, რა თქმა უნდა, სხვადასხვაგვარია.

ამდენად, თითოეული პრეცედენტის აღწერა და ფუნქციონირების ლოგიკის განსაზღვრა წარმოებს ყველა შემადგენელი დიაგრამით.

პრეცედენტის თვისებების ფანჯარა შედგება ოთხი ძირითადი დანაყოფისგან: General, Diagrams, Relations, Files. ოფციაში Diagrams (მარჯვენა ღილაკი – open Specification- Diagrams- მარჯვენა ღილაკი) შესაძლებელია პრეცედენტის ყველა ჩამოთვლილი დიაგრამის შექმნა, რის საფუძველზეც სრულად აღიწერება კონკრეტული პრეცედენტი (სურ. 3.6).

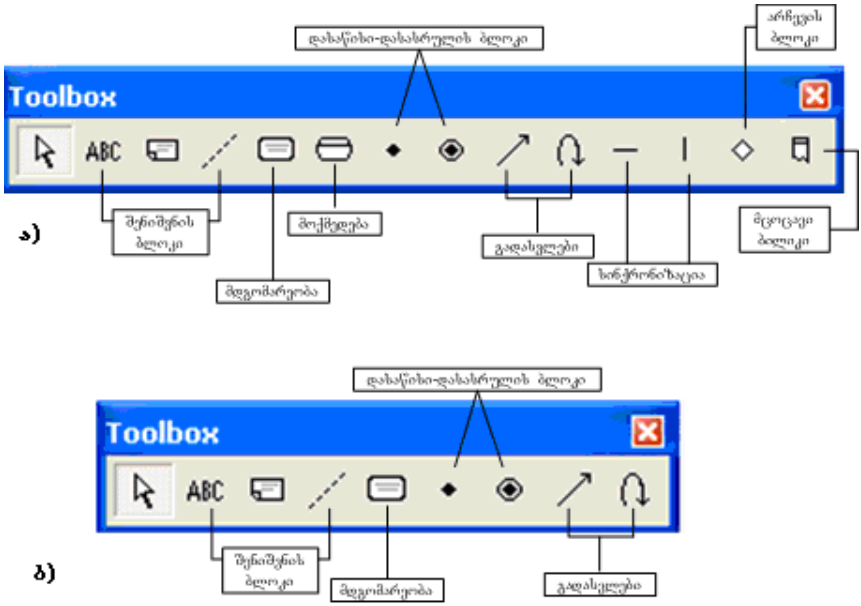


სურ.3.6. პიქტოგრამა- პრეცედენტის თვისებების ფორმირების ნიმუში

ობიექტ-ორიენტირებული დაპროექტების თვალსაზრისით, თითოეულ პრეცედენტში ინკაფსულირებულია შესაბამისი დიაგრამები.

1.2. აქტიურობისა და მდგომარეობათა დიაგრამა – Activity, Statechart Diagram

3.7 ა, ბ ნახაზზე მოცემულია აქტიურობისა და მდგომარეობათა დიაგრამის ინსტრუმენტული საშუალებების ფანჯარა თავისი აღწერილობით.



სურ. 3.7. აქტიურობისა და მდგომარეობათა დიაგრამის ინსტრუმენტული პანელი

აქტიურობის დიაგრამა, პრაქტიკულად, კონკრეტული ამოცანის ალგორითმია, რომელიც აღწერს ოპერაციების შესრულების მიმდევრობას და ამასთან ერთად უჩვენებს, რომელი კლასი ასრულებს კონკრეტულ ოპერაციას. ოპერაციები გამოისახება კომპონენტებით- მოქმედება და მდგომარეობა, რომელიც შესაძლებელია ასევე იშლებოდეს ოპერაციების ანუ აქტიურობის ქვედიაგრამად. ქვედიაგრამის აგების ფანჯრის გახსნა ხორციელდება მოქმედებაზე ან მდგომარეობაზე მარჯვენა ღილაკის დაჭერით - Sub Diagrams-New Activity Diagram.

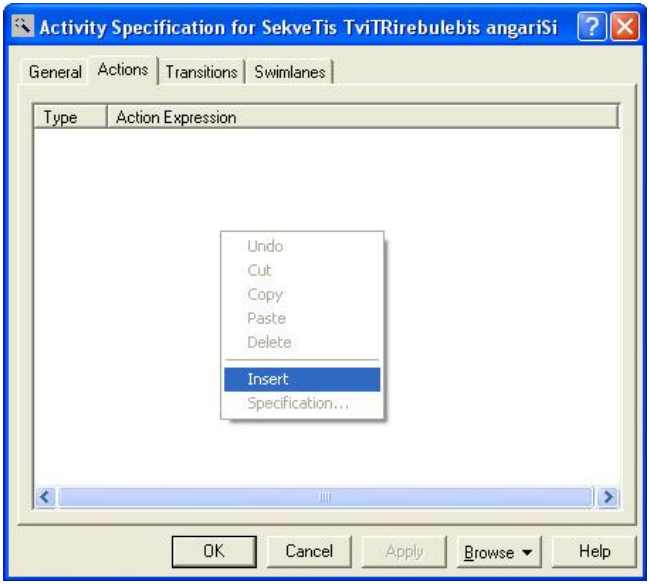
მდგომარეობათა დიაგრამა უჩვენებს სისტემის ან კონკრეტული პროცესის სიტუაციების მსვლელობისა და ქცევის ფუნქციონირებას - რა სიტუაციებში შესაძლებელია აღმოჩნდეს სისტემა ან პროცესი. ანუ რომელიმე ქმედების ან ოპერაციის გაშვებისას რა მდგომარეობაში შესაძლებელია აღმოჩნდეს სისტემა. მაგალითად, განვიხილოთ სიტუაცია ბანკომატში პლასტიკური ბარათის ჩადება და პინ-კოდის შეტანა. სისტემა ამოწმებს პლასტიკურ ბარათს და მითითებული პინ-კოდის სისწორეს. ამ შემთხვევაში სისტემა ლოდინის სიტუაციაშია და შესაძლებელია გადავიდეს ორ სხვადასხვა მდგომარეობაში:

1. იმ შემთხვევაში თუ მითითებული პინ-კოდი არასწორია სისტემაში შესვლა არ განხორციელდება;

2. იმ შემთხვევაში თუ მითითებული პინ-კოდი სწორია განხორციელდება სისტემაში შესვლა.

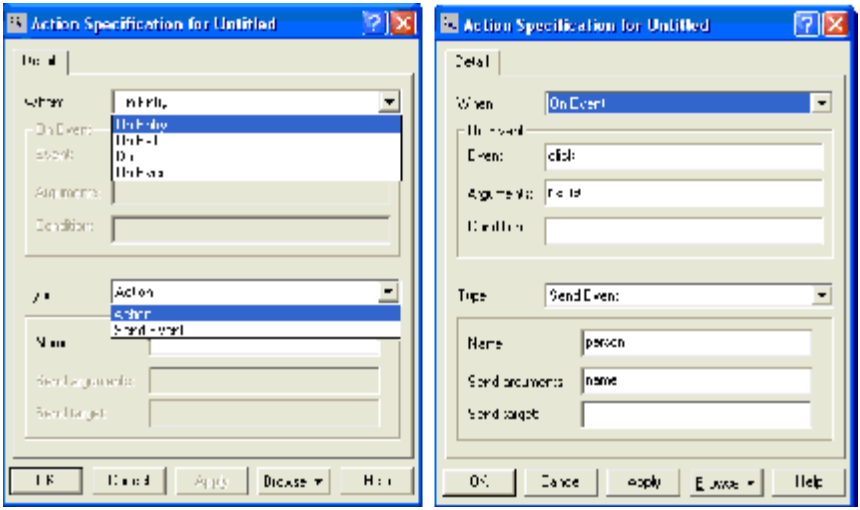
პროგრამულ სისტემაში ოპერაციის შესასრულებლად დამახასიათებელია მოვლენისა და არგუმენტების ტიპების მითითება. მოვლენა აღწერს როგორ მოხდეს ოპერაციის გამოძახება, მაგალითად მაუსის დაჭერისას, ინფორმაციის შეტანისას და ა. შ., ხოლო არგუმენტი მეთოდის პარამეტრებია.

მოქმედებისა და მდგომარეობის თვისებების ფანჯარაში Open Specification-Actions - მარჯვენა ღილაკით-Insert, შესაძლებელია ოპერაციის მოვლენის ჩასმა (სურ. 3.8), ხოლო ჩასმულ მოვლენაზე მაუსის 2-ჯერ დაჭერით ხდება მისი დეტალური აღწერა და თვისებების ფორმირება მოვლენის ტიპისა და არგუმენტის მიხედვით ანუ მიეთითება როდის და როგორ შესრულდეს კონკრეტული ოპერაცია (სურ. 3.9).

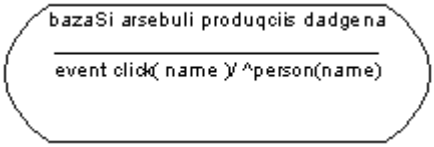


სურ.3.8. მოქმედებისა და მდგომარეობის თვისებების ფანჯარა

3.10 სურათზე ნაჩვენებია მოქმედების/მდგომარეობის კომპონენტის ფორმირების შედეგი.



სურ.3.9. მოვლენის დეტალური აღწერის დიალოგური ფანჯრები

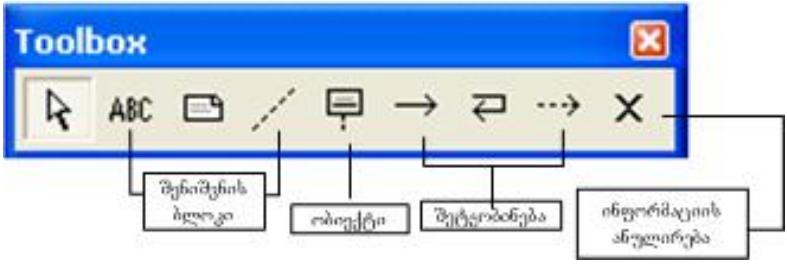


სურ.3.10. მოქმედების/მდგომარეობის კომპონენტის ფორმირების შედეგი

1.3. მიმდევრობითობის დიაგრამა - Sequence Diagram

მიმდევრობითობის დიაგრამა აღწერს გარკვეული პროცესის შესრულებისას ამ პროცესში მომქმედი ობიექტების ინფორმაციულ ურთიერთობას. მისი მთავარი ელემენტებია: ობიექტი, სასიცოცხლო ციკლის ხაზით ანუ მოქმედების პერიოდით, შეტყობინება და

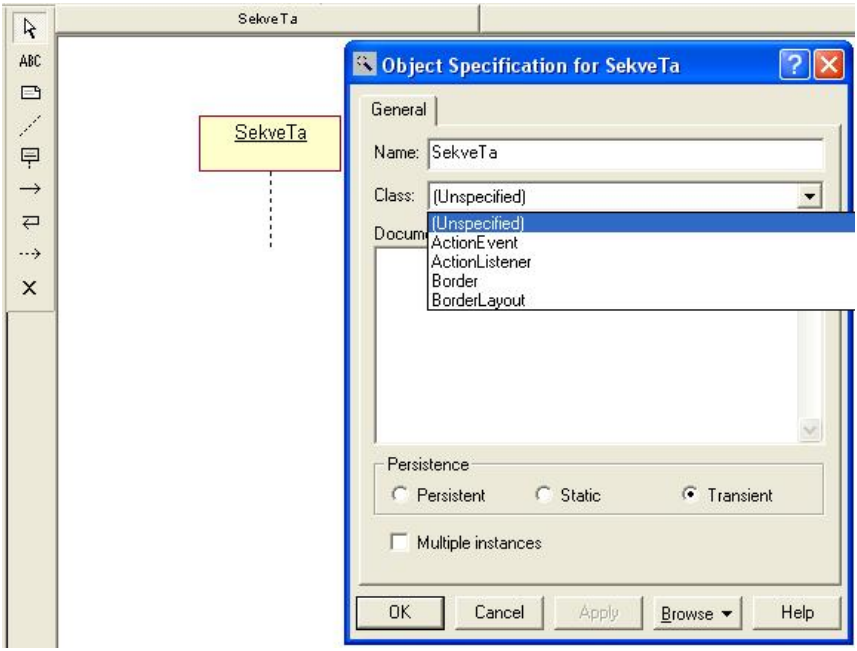
ინფორმაციის ანულირება. 3.11 სურათზე ნაჩვენებია მიმდევრობითობის დიაგრამის ინსტრუმენტების პანელი.



სურ.3.11. მიმდევრობითობის დიაგრამის ინსტრუმენტების პანელი

ობიექტი გრაფიკულად გამოისახება ობიექტის ბაზური კლასის სტერეოტიპის შესაბამისად. ობიექტის სასიცოცხლო ციკლის ხაზი ანუ მოქმედების პერიოდი – გამოსახავს ობიექტის არსებობას დროის გარკვეულ პერიოდში. იგი უჩვენებს დროის იმ მონაკვეთს, რომლის განმავლობაშიც ობიექტი ასრულებს მოქმედებას. ობიექტის თვისებების ფანჯარაში (მარჯვენა ღილაკი – open Specification) ხდება ობიექტის სახელისა და ბაზური კლასის მითითება (სურ. 3..12).

შეტყობინება წარმოადგენს კავშირს ობიექტებს შორის, უჩვნებს ინფორმაციის გადაცემას ერთი ობიექტიდან მეორეზე და მთავრდება მოქმედებით. დასაშვებია შეტყობინება სრულდებოდეს ერთი ობიექტის ფარგლებშიც (რეკურსია).



სურ. 3.12. ობიექტის თვისებების ფანჯარა

არსებობს შეტყობინების რამდენიმე სახესხვაობა, რაც გრაფიკულად ასევე სხვადასხვა სახეს იღებს.

– ჩვეულებრივი - Simple, რომელიც გამოიყენება შეტყობინების გადასაცემად;

– სინქრონული შეტყობინებისთვის- Synchronous;

– პროცედურის გამოძახებისთვის ან ოპერაციის შესრულებისთვის – Procedure call;

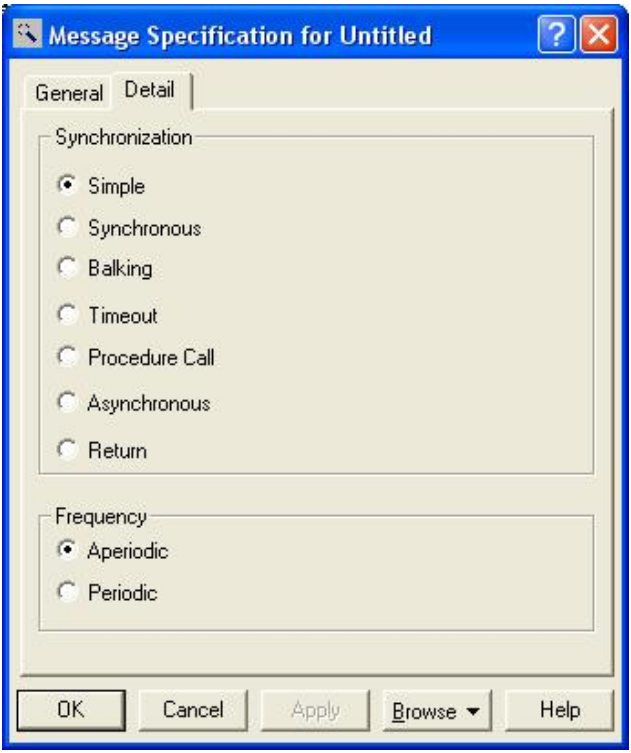
– შეტყობინების გადაცემის შეფერხებისთვის - balking;

– შეტყობინების მიღების ლოდინისთვის დროის უკმარისობის გამოსახვისთვის – Timeout;

– ასინქრონული - Asynchronous შეტყობინებისთვის. იგი, გამოიყენება კერძო შემთხვევებისთვის. მაგალითად, ოპერაციის შეწყვეტა ექსკლუზიური სიტუაციის დროს. ამ დროს გამომძახებელ ობიექტს გადაეცემა შეტყობინება შემდგომი პროცესის შესასრულებლად.;

– რეკურსიული შეტყობინებისთვის - Return.

შეტყობინების ტიპის მითითება ხდება შეტყობინების ისარზე მაუსის ორჯერ დაჭერით და ბლოკზე – Detail გადართვით (სურ. 3.13).



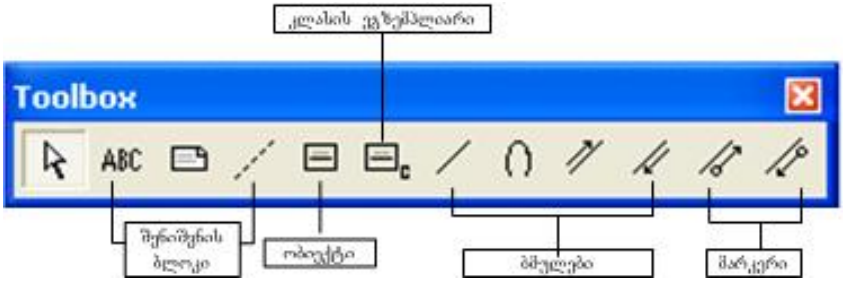
სურ.3.13. შეტყობინების ტიპის მითითების თვისებების ფანჯარა

1.4. კოოპერაციის დიაგრამა - Collaboration Diagram

კოოპერაციის დიაგრამის აგება ხდება მიმდევრობითობის დიაგრამის საფუძველზე და აღწერს ობიექტებს შორის დამოკიდებულებათა მიმართულებას.

კოოპერაციის დიაგრამა ავტომატურად მიიღება მიმდევრობითობის დიაგრამიდან F5 კლავიშზე დაჭერით.

3.14 სურათზე ნაჩვენებია კოოპერაციის დიაგრამის ინსტრუმენტების პანელი.

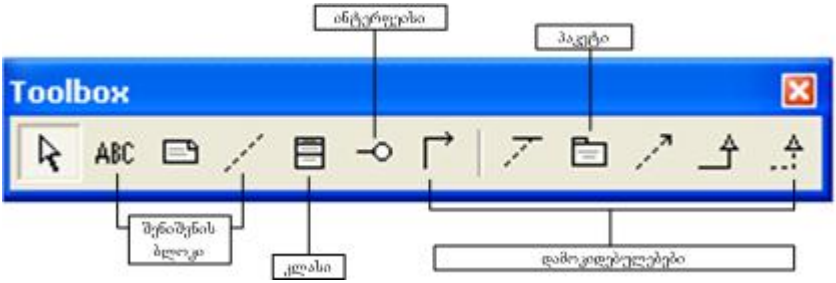


სურ.3.14. კოოპერაციის დიაგრამის ინსტრუმენტების პანელი

2. დაპროექტების ბლოკი - Logical View

კლასების დიაგრამა - Class Diagram

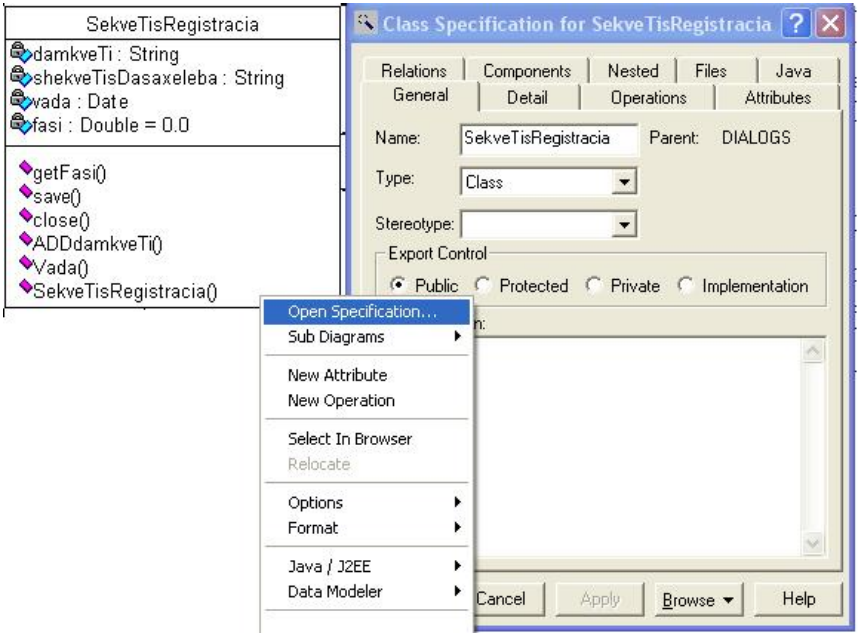
კლასების დიაგრამა, პრაქტიკულად ობიექტ-ორიენტირებული დაპროექტების ბირთვია. 3.15 სურათზე ნაჩვენებია კლასების დიაგრამის ინსტრუმენტების პანელი, აღწერილობით.



სურ.3.15. კლასთა დიაგრამის ინსტრუმენტების პანელი

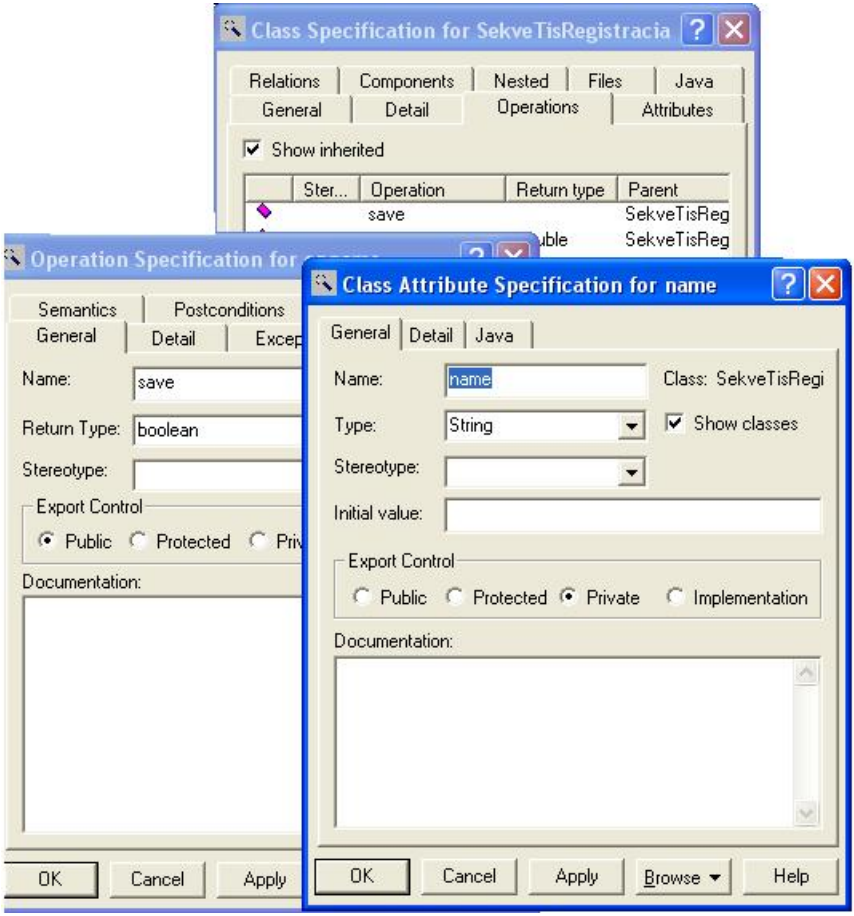
კლასი აღწერს სისტემის შემადგენელ ელემენტებს, რომელსაც აქვს სხვადასხვა თვისებები/მახასიათებლები (მაგალითად, თუ განვიხილავთ სახლს, როგორც სისტემას, მისი შემადგენელი ელემენტები იქნება – კედლები, კარები, ფანჯარა. თითოეულს გააჩნია მახასიათებელი, დავუშვათ სიგრძე, სიგანე, იღება, იკეტება და ა.შ.). მახასიათებლების აღწერა ხდება კლასის თვისებების ფანჯარაში (მარჯვენა ღილაკი – open Specification). კლასის თვისებების ფანჯარა ნაჩვენებია სურათზე 3.16.

ატრიბუტების ფორმირება წარმოებს თვისებების ფანჯრის ბლოკში “Attributes”, ხოლო მეთოდების აღწერა ხდება ბლოკში “Operations”. ატრიბუტებისა და მეთოდების ჩასმა ხდება შესაბამის ბლოკში ფუნქციით “Insert” (მარჯვენა ღილაკის კონტექსტური მენიუ). ჩასმულ ატრიბუტზე ან მეთოდზე ფუნქციის “Specification” გამოყენება დაწვრილებით აღწერს კონკრეტულ ატრიბუტს/მეთოდს (სურ. 3.17).



სურ.3.16. კლასის თვისებების ფორმირების დიალოგური ფანჯარა

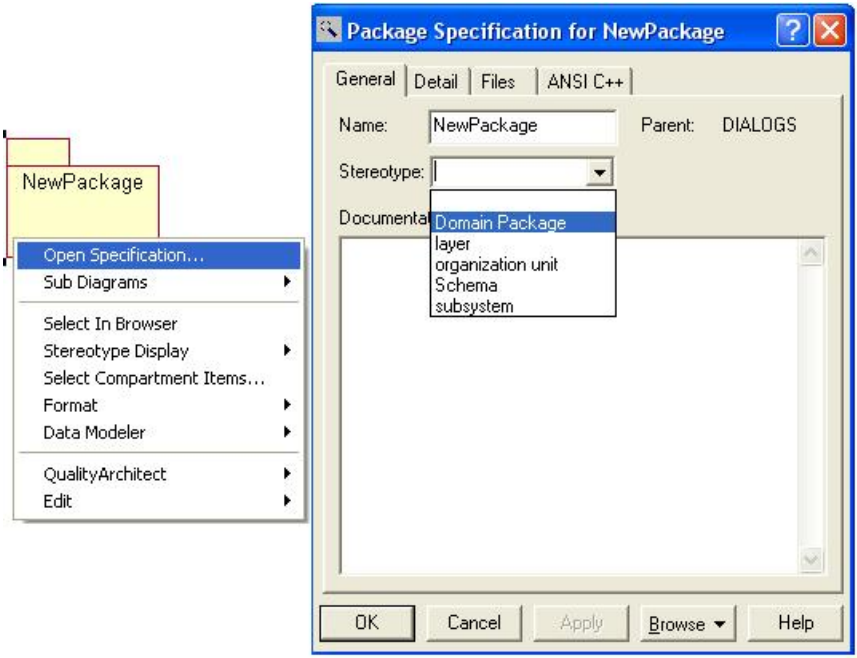
ნებისმიერი კლასის ფუნქციონირება შეუძლებელია სხვა კლასებთან დამოკიდებულების გარეშე. ობიექტ-ორიენტირებულ დაპროექტებაში ცნობილია კლასების კოოპერაცია (თანამშრმლობა), რომლის აღწერაც ხდება კლასების დიაგრამაში. სასურველია, რომ კლასების თითოეული დიაგრამა აღწერდეს მხოლოდ ერთ კოოპერაციას.



სურ.3.17. ატრიბუტებისა და ოპერაციების თვისებების ფანჯარა

კლასების დიაგრამის აგებისას ხშირად გამოიყენება პაკეტი (სტრუქტურულად ფორმირებისთვის), რომელსაც სხვადასვა სახის სტერეოტიპის მიღება შეუძლია (დირექტორია, ქვედირექტორია, ქვესისტემა და ა.შ.). მაგალითად, სისტემაში შესაძლებელია

არსებობდეს დიალოგური კლასები, რომელიც დავუშვათ მოთავსდება პაკეტში – “Dialogs”, დამხმარე კომპონენტური კლასები, რომელიც დავუშვათ მოთავსდება პაკეტში – “Service” და ა.შ. პაკეტის თვისებების ფანჯარა ნაჩვენებია 3.18. სურათზე.



სურ. 3.18. პაკეტის თვისებების ფანჯარა

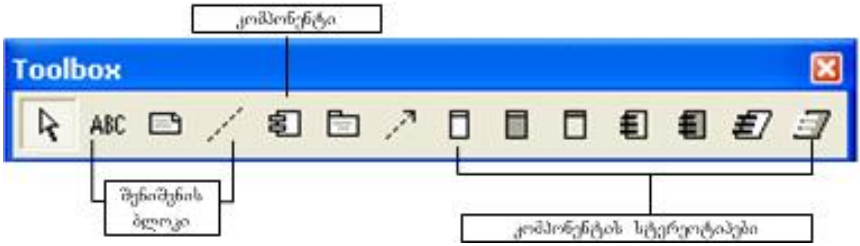
კლასების დიაგრამაში, თითოეული კლასისთვის შესაძლებელია კლასის ფუნქციონირების ლოგიკის ანუ ოპერაციების შესრულების თანამიმდევრობის აღწერა აქტიურობისა და მოქმედებების დიაგრამებით (მარჯვენა ღილაკი – Sub Diagrams).

3. რეალიზაციისა და ტესტირების ბლოკი - Component View, Deployment View

3.1. კომპონენტების დიაგრამა – Component Diagram

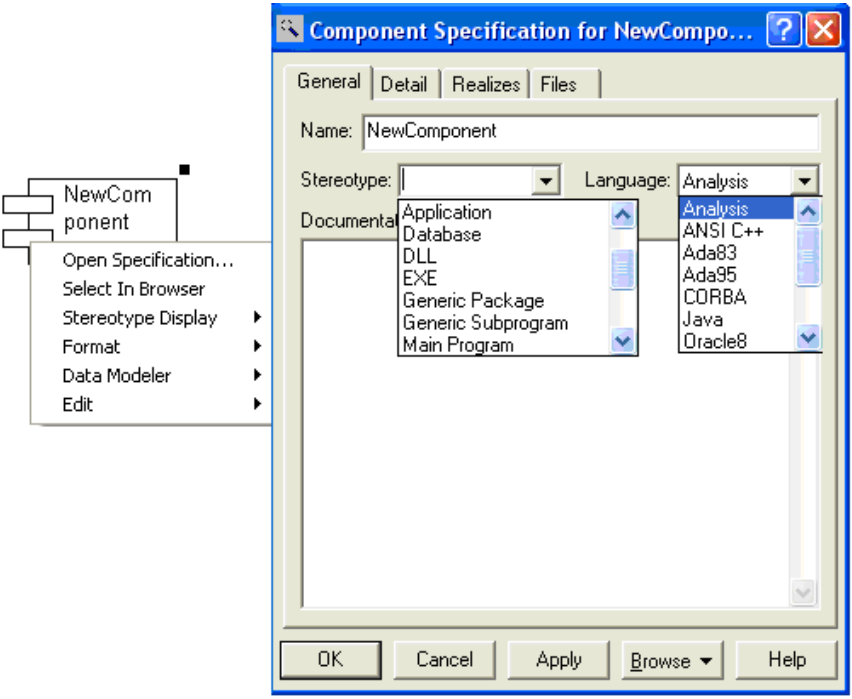
კომპონენტების დიაგრამით ხდება სისტემის მოდელის აღწერა ფიზიკურ დონეზე. კომპონენტად შესაძლებელია ჩაითვალოს სისტემის ნებისმიერი მსხვილი მოდულის ობიექტი, მაგალითად, ცხრილი, მონაცემთა ბაზა, ქვესისტემა, ფაილი, დამხმარე სხვა პროგრამული სისტემა ან დანართი. მასში ძირითადად მოთავსებულია ინფორმაცია პროგრამული კოდისა და დინამიკური (DLL) ბიბლიოთეკის შესახებ.

კომპონენტური დიაგრამის ძირითადი მიზანია გამოსახოს სისტემის შემადგენელი ლოგიკური ელემენტების მთლიანობა და კოდირების მართვის, კომპილაციის და დანართების განთავსების აღწერა. ამ ბლოკს უმეტესად იყენებს პროგრამისტები და ტექნიკური დამმუშავებლები. კომპონენტური დიაგრამის პრაქტიკულად, ერთადერთი ელემენტია კომპონენტი, თუმცა ინსტრუმენტების პანელზე (სურ. 3.19) მოთავსებულია კომპონენტი ძირითადი სტერეოტიპებით, მაგ., გამშვები პროგრამა, ქვეპროგრამა და ა.შ.



სურ.3.19. კომპონენტური დიაგრამის ინსტრუმენტების პანელი

კომპონენტის აღნიშნული ძირითადი სტერეოტიპების არჩევა ასევე შესაძლებელია კომპონენტის თვისებების ფანჯარაში, სადაც შესაძლებლობაშია დამატებითი სტერეოტიპის არჩევა და პროგრამული ენის მითითებაც (სურ. 3.20).

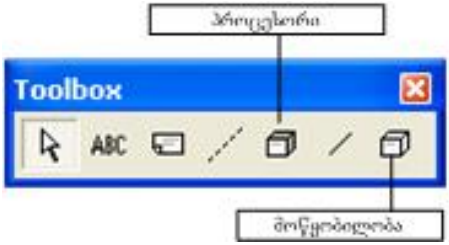


სურ. 3.20. კომპონენთის თვისებების ფანჯარა

3.2. განაწილების/განთავსების დიაგრამა - Deployment Diagram

დიაგრამის აგება ხორციელდება დაპროექტების დასკვნით ეტაპზე და გამოსახავს პროგრამული სისტემის ფიზიკურ წარმოდგენას. მასში აღიწერება, რომელ პლატფორმაზე იგება

სისტემა, როგორია სისტემის განაწილება ქსელური თვალსაზრისით (მაგ., პერსონალური, ლოკალური, კორპორაციული, გლობალური), როგორ არის განაწილებული სამუშაო ადგილები შესაბამისი ფუნქციების მიხედვით და ა.შ. გამოყენებაშია ორი ძირითადი ელემენტი პროცესორი/კვანძი და მოწყობილობა (სურ. 3.21).



სურ.3.21. განაწილების/განთავსების დიაგრამის ინსტრუმენტების პანელი

3.3. პროგრამულ სისტემებთან მუშაობა

UML ენის ერთ-ერთი მთავარი ღირებულებაა პროგრამული უზრუნველყოფის რეალიზაცია. Rational Rose ინსტრუმენტი კლასების დიაგრამიდან პროგრამული კოდის მიღებისა და პროგრამული კოდიდან კლასების დიაგრამის გენერაციის შესაძლებლობას იძლევა.

3.3.1 კლასთა დიაგრამიდან პროგრამული კოდის მიღება

პროგრამული კოდის გენერაცია ძირითადად წარმოებს ნაწილებიდან - Logical package და Component package.

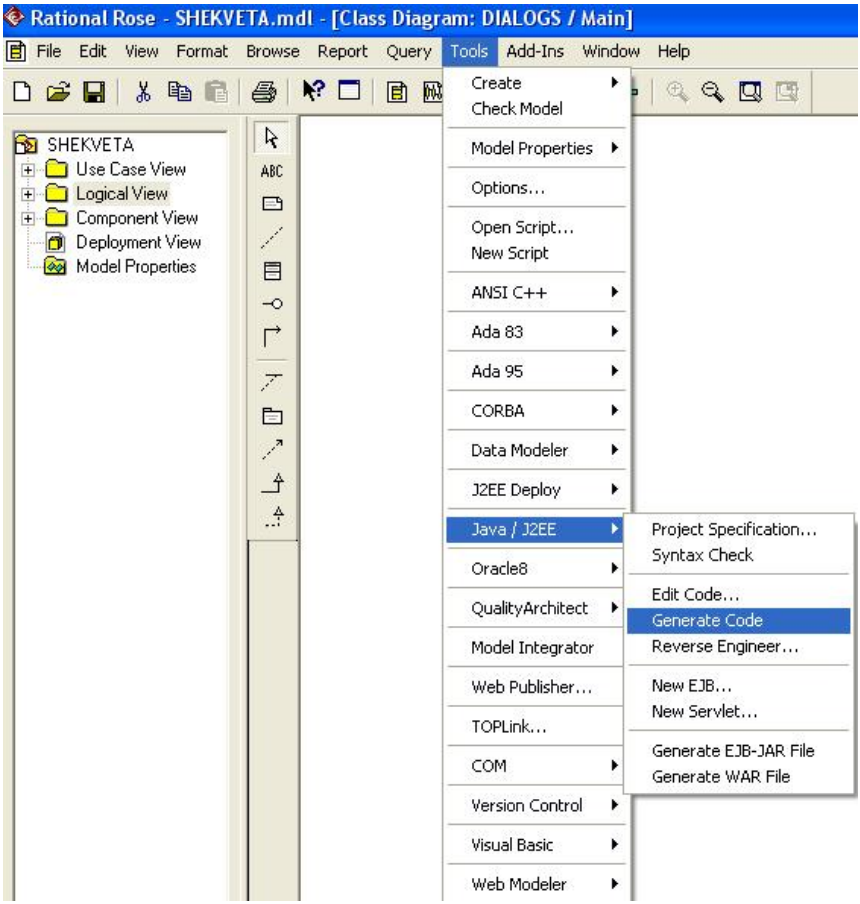
შესაძლებელია როგორც ცალკეული კლასის ან კომპონენტის შესაბამისი კოდის გენერაცია, ისე მთლიანად ლოგიკური ან კომპონენტური პაკეტებიდან პროგრამული კოდის გენერაცია.

აღსანიშნავია, რომ პროგრამული კოდის ავტომატური გენერაციის ქვეშ იგულისხმება მხოლოდ დიაგრამაში განსაზღვრული მოდელის შესაბამისი კლასის სტრუქტურის შექმნა, მონაცემთა ტიპებისა და მეთოდების მიხედვით, და არა პროცედურები და ფუნქციები, რაც მეთოდის ფუნქციონირებას და შინაარს აღწერს. ეს უკვე პროგრამისტის პრეროგატივაა.

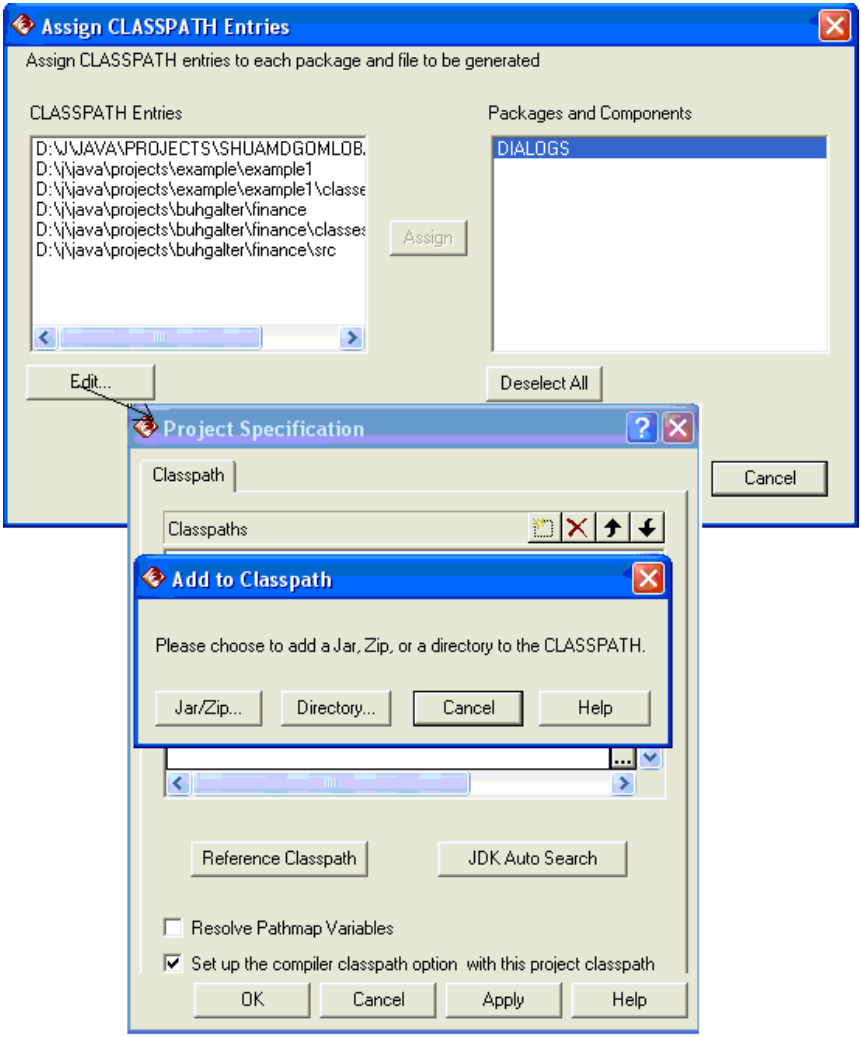
ფაქტობრივად, Rational Rose სისტემით დამუშავებული მოდელის (კლასებისა და კომპონენტების დიაგრამა) გენერაციით იქმნება შესაბამისი დიაგრამის ე.წ. კარკასული პროგრამული კოდი, ჩაწერილი მითითებული პროგრამული ენის სტილში.

პროგრამული კოდის გენერაციისთვის Tools მენიუში წარმოდგენილია პროგრამული სისტემების საკმაოდ ფართო სპექტრი, აქვე შედის მონაცემთა ბაზის გენერაციის საშუალებებიც ფუნქციით (Data Modeler).

მაგალითად, Java გარემოში კოდის გენერაციისთვის, ვახდენთ დასაგენერირებელი კლასის მონიშვნას, Tools მენიუში ვირჩევთ ფუნქციას Java/J2EE – Generate Code (სურ. 3.22), რის შედეგადაც ეკრანზე ჩნდება გენერირებული კლასის მისამართის (CLASSPATH) განსაზღვრა ანუ სად მოთავსდეს გენერირებული კლასი (სურ. 3.23).



სურ.3.22. Java გარემოში კოდის გენერაცია

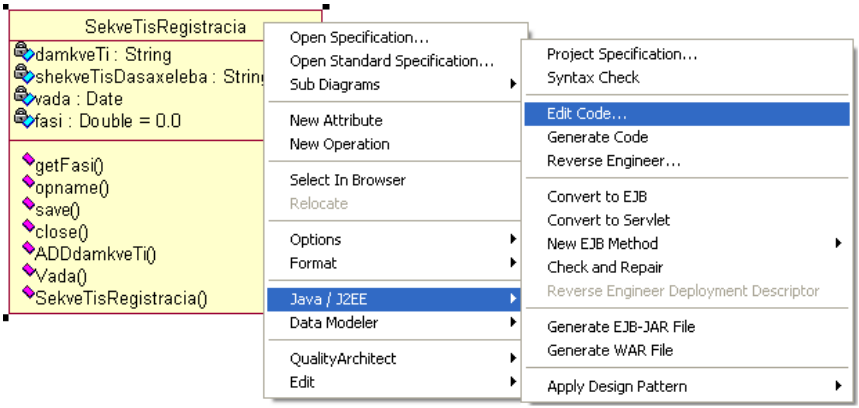


სურ. 3. 23. გენერირებული კლასის მისამართის (CLASSPATH) განსაზღვრის დიალოგური ფანჯარა

არსებობს მეორე გზაც, კოდის სწრაფი გენერაციისთვის:

კლასთა დიაგრმაზე, მარჯვენა ღილაკის მენიუდან Java/J2EE-Generate Code.

შესაძლებელია ასევე Java კოდის გამოტანა Rational Rose სისტემის სამუშაო ფანჯარაზეც - კლასთა დიაგრმაზე, მარჯვენა ღილაკის მენიუდან Java/J2EE-Edit Code ფუნქციით.

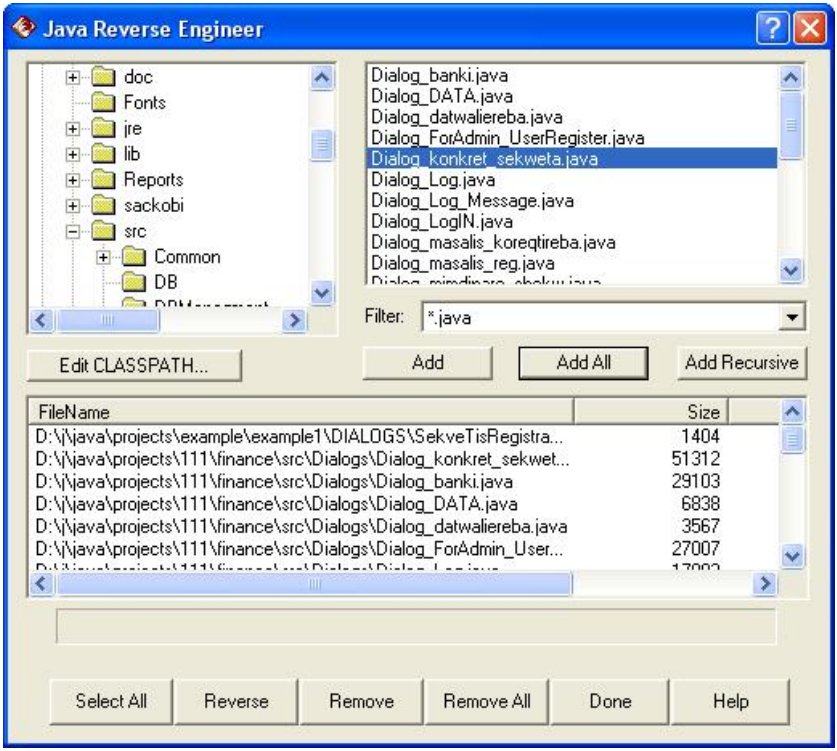


სურ.3.24. Java კოდის გამოტანა Rational Rose სისტემის სამუშაო ფანჯარაზე

3.3.2. პროგრამული კოდიდან კლასთა დიაგრამის გენერაცია

ნებისმიერი ავტომატიზებული საწარმოო პროცესი, წარმოების განვითარებისა და შესაბამის ცვლილებასთან ერთად საჭიროებს ამ ცვლილებების ასახვას უკვე დანერგილ ავტომატიზებულ სისტემაში.

პროგრამული კოდიდან კლასთა დიაგრამის გენერაცია გამოიყენება ორ ძირითად შემთხვევაში, რეალიზებული პროგრამული სისტემის სრულყოფისთვის და პროგრამული სისტემის დოკუმენტაციის შესაქმნელად, რაც უკუ-დაპროექტების (Reverse Engineering) ფუნქციით ხორციელდება. უკუ-დაპროექტების ოპერაცია სრულდება Tools მენიუს შესაბამისი პროგრამული სისტემიდან. მაგალითად, Java პროგრამისთვის - Tools მენიუში ვირჩევთ ფუნქციას Java/J2EE – Reverse Engineer...(სურ. 3.25).



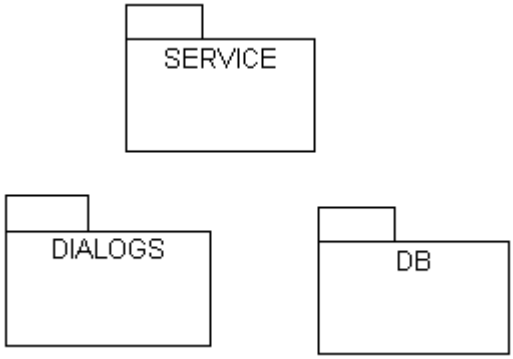
სურ. 3.25. Java პროგრამისთვის უკუ-დაპროექტების დიალოგური ფანჯარა

4. მითითებები საკურსო პროექტის შესასრულებლად

მოცემულია საკურსო დავალება: "ფირმაში შეკვეთის მიღების პროცესის მართვა".

სცენარი: შემოდის შეკვეთა, მიმღები (რეგისტრატორი) ახდენს შეკვეთის რეგისტრაციას. შეკვეთის მოთხოვნის შესაბამისად ხდება სარეალიზაციო პროდუქციის პარამეტრების დამუშავება. შეკვეთის ღირებულების დასადგენად რეგისტრატორი მიმართავს პროდუქციის მენეჯერს. იმ შემთხვევაში თუ საწყობში შეკვეთილი პროდუქციის რაოდენობა საკმარისია, პროდუქციის მენეჯერი მიმართავს ფინანსურ მენეჯერს გასაყიდი ფასის დასაფიქსირებლად და გადასცემს შეკვეთის ღირებულებას რეგისტრატორს, რომელიც საბოლოოდ აფორმებს ხელშეკრულებას დამკვეთთან. იმ შემთხვევაში თუ შეკვეთილი პროდუქციის რაოდენობა საწყობში არასაკმარისია პროდუქციის მენეჯერი მიმართავს მომარაგების განყოფილებას პროდუქციის შესაძენად. დასასრული ეტაპია საბოლოო ნაშთების გაანგარიშება და ფაქტობრივი ხარჯის მიხედვით შეკვეთის გაფორმება.

საწყის ეტაპზე, კლასების დიაგრამაში აღვწეროთ სარეალიზაციო სისტემის სამი პაკეტი – დიალოგური რეჟიმის კლასები (DIALOGS), სერვისული კლასები (SERVISE) და მონაცემთა ბაზების კლასები (DB), რომელშიც შემდგომში მოთავსდება შესაბამისი თემატიკით კლასთა დიაგრამები (სურ. 4.1). რის შემდეგაც ვახდენთ შემთხვევების გამოყენების დიაგრამის ანუ პრეცედენტების დიაგრამის აგებას (სურ. 4.2.).



სურ. 4.1. სარეალიზაციო სისტემის პაკეტების ფრაგმენტი

მოცემული სცენარის მიხედვით სისტემაში მონაწილე პირებია - გაყიდვების მენეჯერი, პროდუქციის მენეჯერი, მომარაგების მენეჯერი და ფინანსური მენეჯერი. ობიექტ-ორიენტირებული პროგრამული უზრუნველყოფის თვალსაზრისით, ჩამოთვლილი პირები წარმოადგენს სარეალიზაციო სისტემის განაწილებული სამუშაო ადგილების არსებობას, რაც გულისხმობს თითოეული პირისთვის მთლიანი სისტემის შესაბამისი დიალოგის არსებობას. ამგვარ განაწილებას, საინფორმაციო ტექნოლოგიებში საქმეთა მართვის პროცესს უწოდებენ.

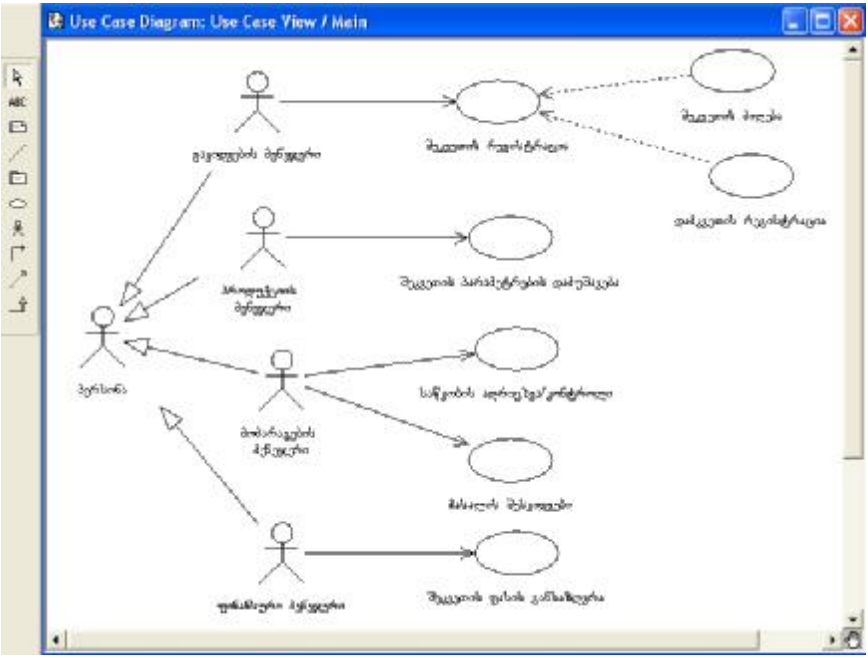
ამგვარად, მოცემული პირებისთვის ანუ აქტორებისთვის განისაზღვრება შემდეგი მოვლენები/პრეცედენტები:

1. გაყიდვების მენეჯერი - პრეცედენტი: შეკვეთის რეგისტრაცია/გაფორმება. შეკვეთის რეგისტრაციას თან უნდა ახლდეს ორი აუცილებელი ქვემოთხოვნა - დამკვეთის რეგისტრაცია (ან მითითება) და შეკვეთის მიღება;

2. პროდუქციის მენეჯერი. პრეცედენტი: შეკვეთის პარამეტრების დამუშავება, ინფორმაცია საწყობში არსებულ მარაგებზე.

3. მომარაგების მენეჯერი. პრეცედენტი: მასალის შესყიდვები, ინფორმაცია მიმწოდებლებზე.

4. ფინანსური მენეჯერი. პრეცედენტი: შეკვეთის ფასის განსაზღვრა, დამკვეთის მონაცემები.



სურ. 4.2. შემთხვევების გამოყენების დიაგრამის ფრაგმენტი

აღსანიშნავია, რომ თანამედროვე საინფორმაციო ტექნოლოგიებში შეძლებისდაგვარად ზოგადად უნდა მოხდეს თითოეული ობიექტის აღწერა, რათა თავიდან იქნას აცილებული

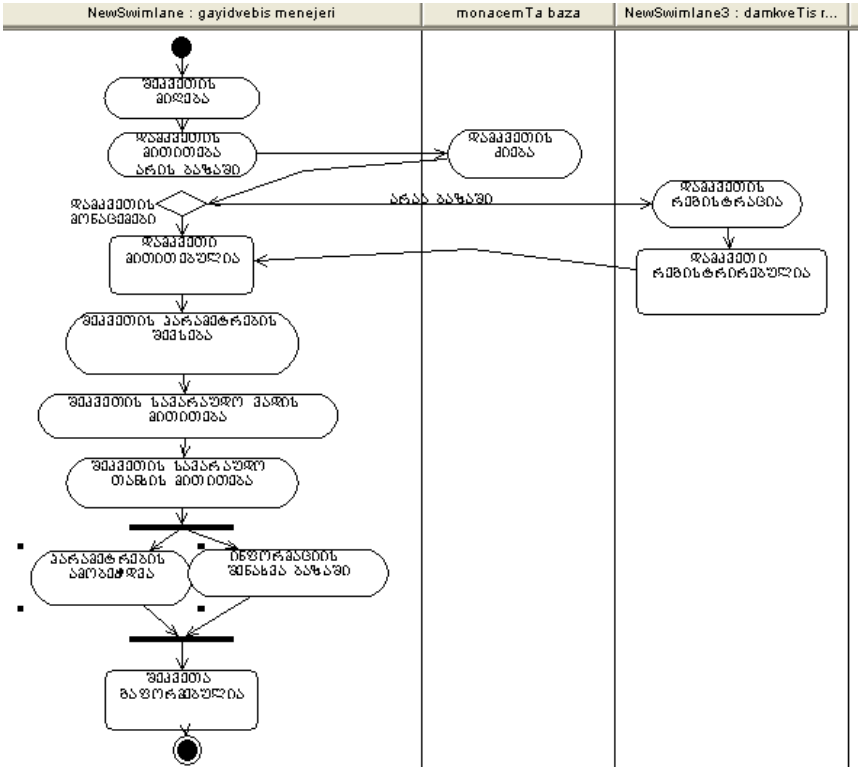
ობიექტის პარამეტრებისა და ოპერაციების დუბლირება. ამ თვალსაზრისით, ზემოთ ჩამოთვლილი პირები, შესაძლებელია მივაკუთვნოთ ერთ ზოგად კლასს პერსონა (მემკვიდრეობითობა).

შემდეგ ეტაპზე ვახდენთ შეკვეთის თითოეული პრეცედენტის დეტალურ აღწერას.

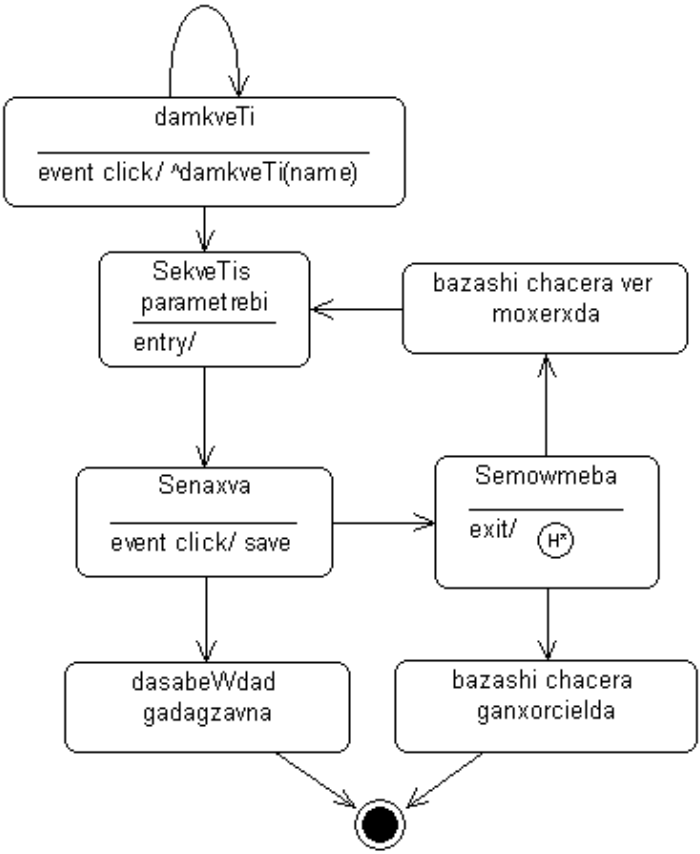
პრეცედენტი – შეკვეთის გაფორმება. შეკვეთის გაფორმება ხორციელდება შემდეგი ლოგიკით:

რეგისტრატორი შეკვეთის მიღებისას თავდაპირველად ახდენს დამკვეთის ძიებას მონაცემთა ბაზაში. იმ შემთხვევაში, თუ დამკვეთი არ მოიძებნა, ახდენს დამკვეთის რეგისტრაციას. შემდეგ ეტაპზე გადასცემს პროდუქციის მენეჯერს შეკვეთის პარამეტრებს შეკვეთის ღირებულების დასადგენად. პროდუქციის მენეჯერი ადგენს პროდუქციის გასაყიდ ფასს და გადასცემს შედეგს რეგისტრატორს. რეგისტრატორი ასრულებს პროცესს ორი მიმართულებით, იმ შემთხვევაში, თუ დამკვეთი არ არის თანახმა მიღებულ ღირებულებაზე აუქმებს შეკვეთას, წინააღმდეგ შემთხვევაში აფორმებს შეკვეთას.

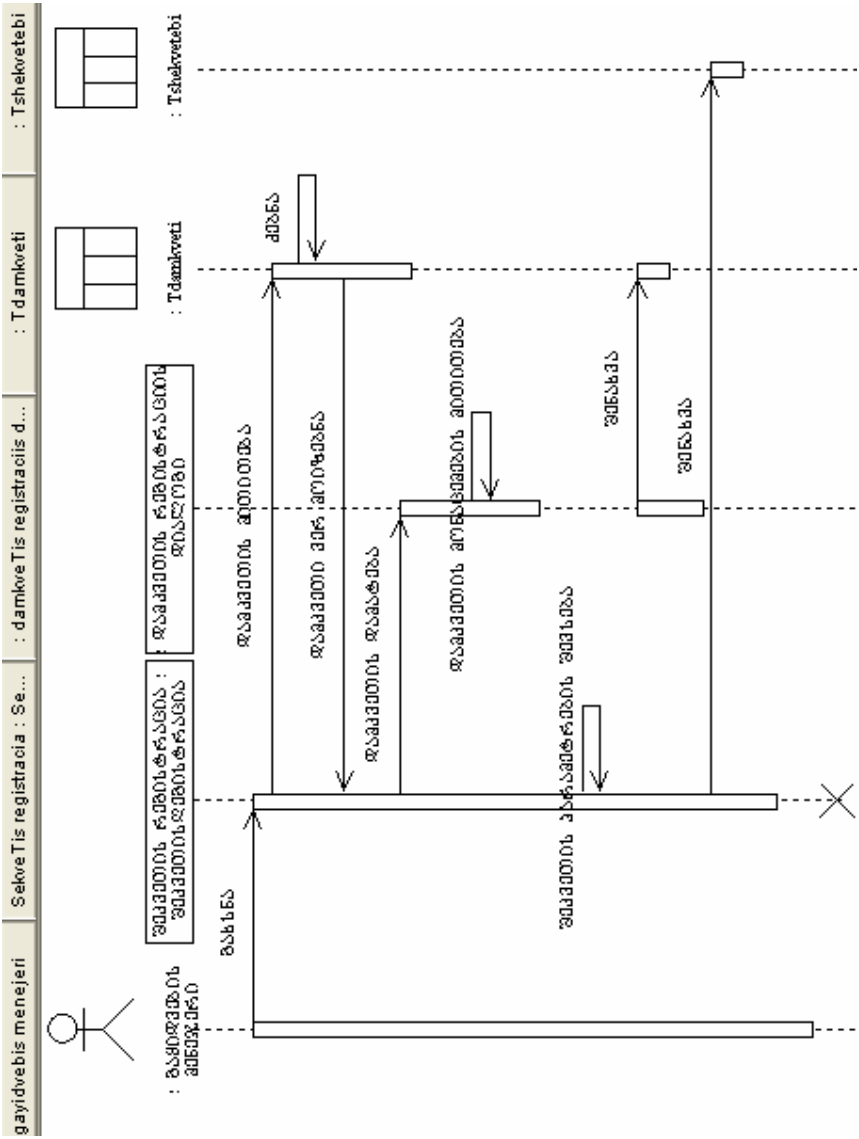
პრეცედენტისთვის - შეკვეთის გაფორმება, აღწერილი სცენარის საფუძველზე მიმდევრობით ვაგებთ შემდეგ დიაგრამებს: აქტიურობის (სურ. 4.3), მდგომარეობის (სურ. 4.4), მიმდევრობითობის (სურ. 4.5), კოოპერაციის (სურ. 4.6), კლასთა (სურ. 4.7) და კომპონენტურ (სურ. 4.8) დიაგრამებს. დასასრულს ვახდენთ კლასთა დიაგრამის შესაბამისი კოდის გენერაციას Java პროგრამულ სისტემაში (სურ. 4.9).



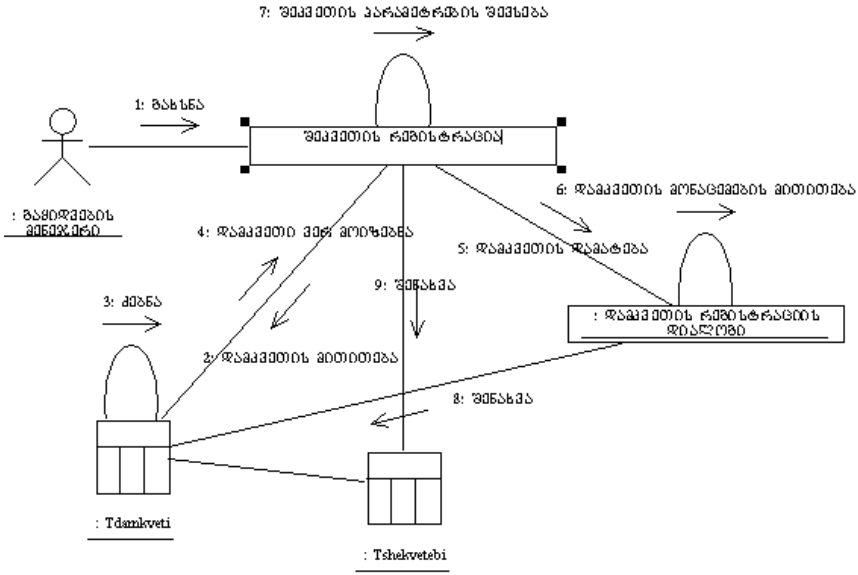
სურ. 4.3. აქტიურობის დიაგრამა პრეცედენტისთვის - შეკვეთის გაფორმება



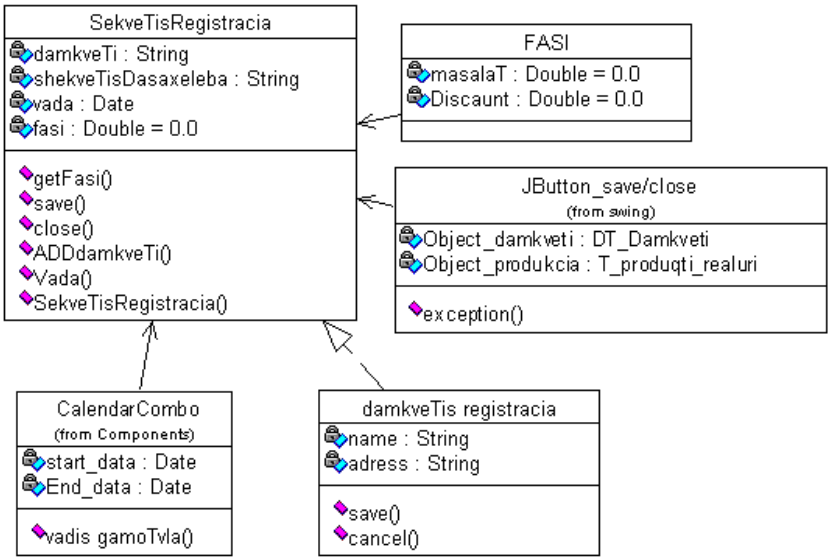
სურ. 4.4. მდგომარეობის დიაგრამა პრეცედენტისთვის - შეკვეთის გაფორმება



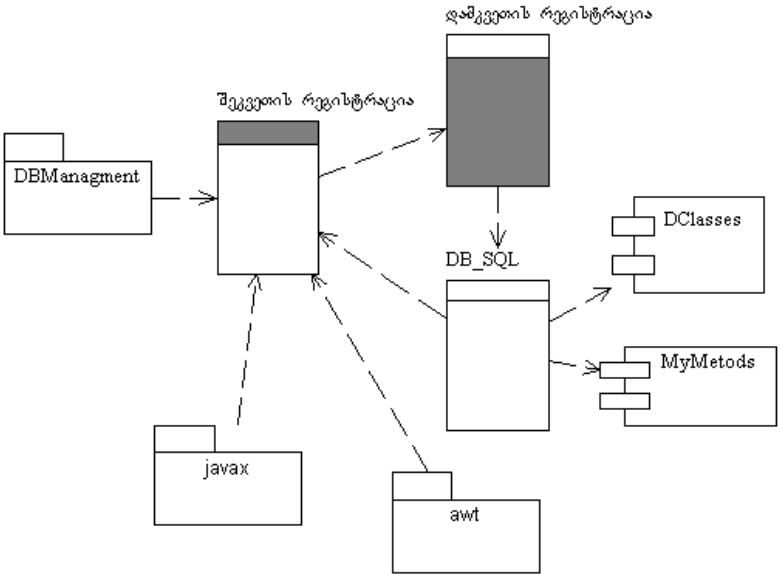
სურ. 4.5. მიმდევრობითობის დიაგრამა პრეცედენტისთვის - შეკვეთის გაფორმება



სურ. 4.6. კოოპერაციის დიაგრამა პრეცედენტისთვის - შეკვეთის გაფორმება



სურ. 4.7. კლასთა დიაგრამა პრეცედენტისთვის - შეკვეთის გაფორმება



სურ. 4.8. კომპონენტური დიაგრამა პრეცედენტისთვის - შეკვეთის გაფორმება

```
SekveTisRegistracia.java
//Source file: D:\\j\\java\\projects\\example\\example1\\DIALOGS\\SekveTisRegistracia.java
package DIALOGS;
import javax.swing.JDialog;
import java.util.Arrays;

public class SekveTisRegistracia extends JDialog
{
   private String damkveTi;
   private String shekveTisDasaxeleba;
   private Date vada;
   private Double fasi = 0.0;
   private String dasaxeleba;

   public SekveTisRegistracia()
   {        }
   public SekveTisRegistracia()
   {   }

   public Boolean save()
   {
    return null;
   }

   public Double getFasi()
   {
    return null;
   }

   public Boolean close()
   {
    return null;
   }

   public TdamkveTi ADDdamkveTi()
   {
    return null;
<
SekveTisRegistracia.java
```

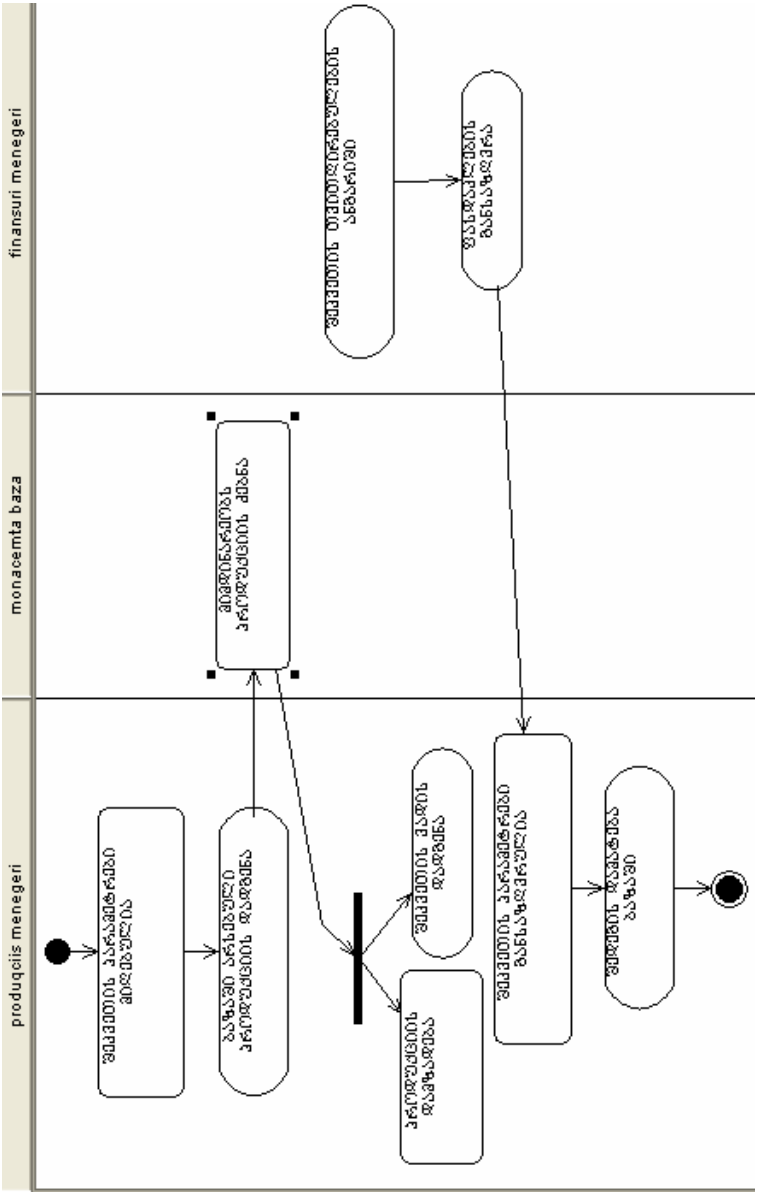
სურ. 4.9. კოდის გენერაციის ფრაგმენტი კლასისთვის – შეკვეთის რეგისტრაცია

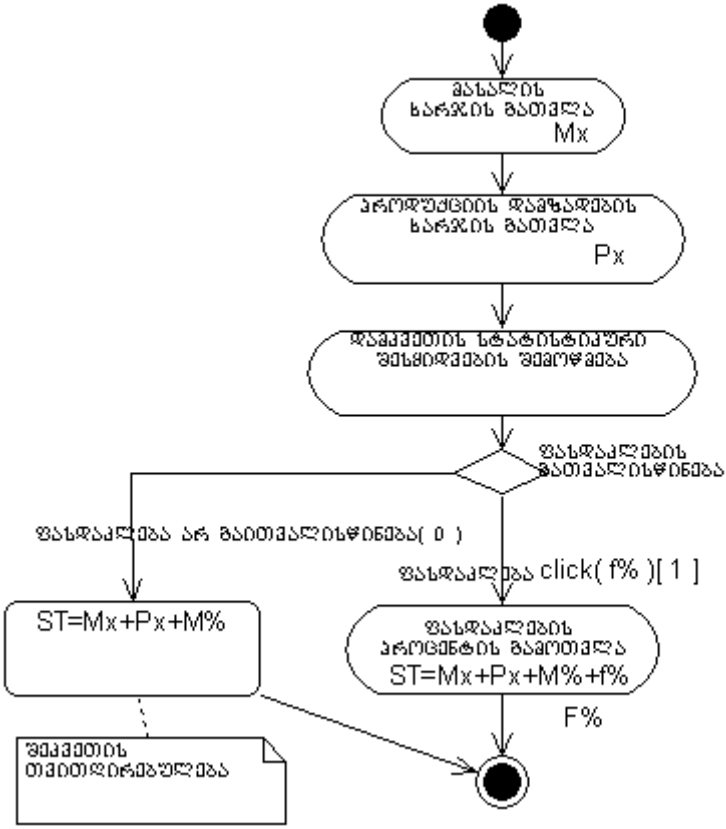
განვიხილოთ კიდევ ერთი პრეცედენტი - პროდუქციის პარამეტრების დამუშავება.

პრეცედენტის სცენარი შემდეგნაირია: პროდუქციის მენეჯერი იღებს შეკვეთის პარამეტრებს. ახდენს პროდუქციის ძიებას მონაცემთა ბაზაში. იმ შემთხვევაში, თუ დაკვეთილი პროდუქცია არ არის საწყობში ან პროდუქციის რაოდენობა არასაკმარისია, პროდუქციის

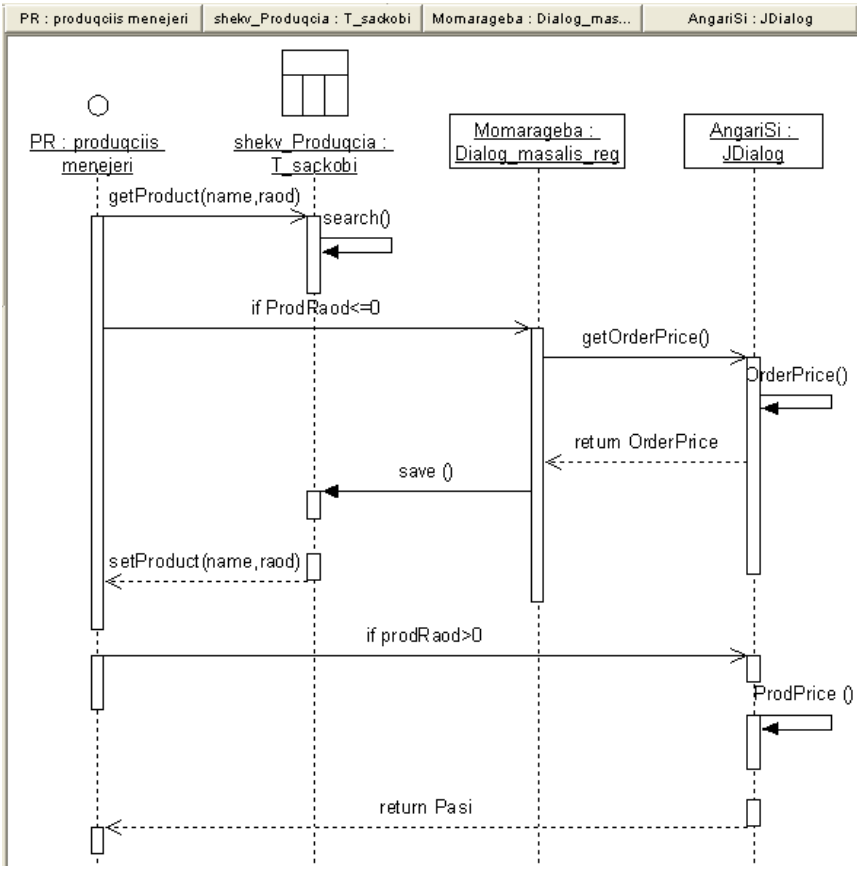
მენეჯერი მიმართავს მომარაგების მენეჯერს ახალი პროდუქციის შეძენის თაობაზე. მომარაგების მენეჯერი შესყიდვის დასაშვები ფასის შესახებ ინფორმაციას იღებს ფინანსური მენეჯერისგან, ახდენს პროდუქციის შესყიდვას და აფიქსირებს მონაცემთა ბაზაში. იმ შემთხვევაში, თუ დაკვეთილი პროდუქცია არის საწყობში და ამასთან პროდუქციის რაოდენობა საკმარისია, იგი მიმართავს ფინანსურ მენეჯერს გასაყიდი ფასის დასადგენად. ფინანსური მენეჯერი ახდენს ფასის გაანგარიშებას შესაძლო ფასდაკლების ან ფასის მომატების ფაქტორების გათვალისწინებით. ფასდაკლება შესაძლებელია გათვალისწინებულ იქნას მაგალითად, თუ დამკვეთი იმეორებს შეკვეთას, ან შეკვეთილი რაოდენობა დიდია. ფასის მომატების ფაქტორი შესაძლოა დამოკიდებულ იქნეს შეკვეთის ვადის დაჩქარებაზე ან სავალუტო კურსის ცვლილებაზე და ა. შ. ფასის დადგენის შემდეგ ფინანსური მენეჯერი გადასცემს შედეგს პროდუქციის მენეჯერს, რომელიც საბოლოოდ ასრულებს პროცესს.

ამგვარად, პრეცედენტისთვის - პროდუქციის პარამეტრების დამუშავება, აგებული დიაგრამები ნაჩვენებია სურათებზე 4.10-4.16. შეკვეთის ფასის ფორმირების ალგორითმი იხსნება მოქმედებაში - შეკვეთის თვითღირებულების ანგარიში, ქვედიაგრამის სახით (სურ. 4.).

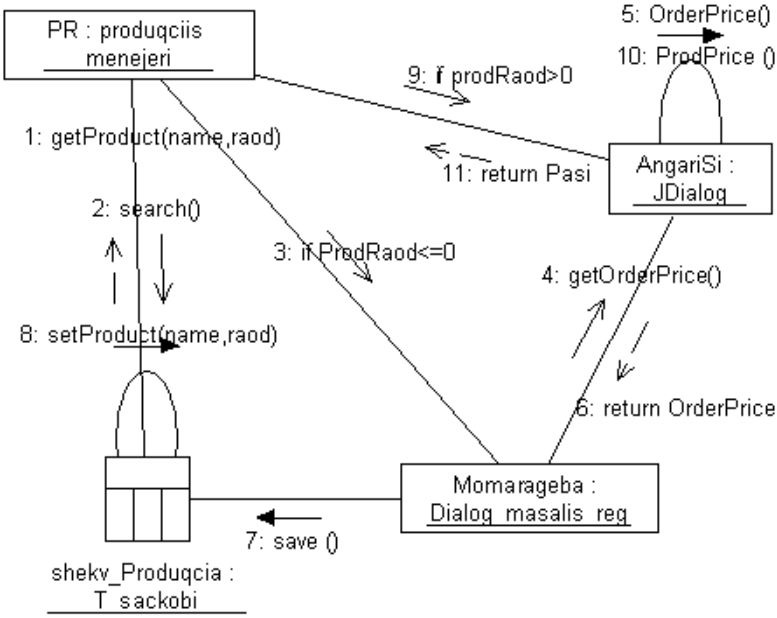

finansuri menegeri
monacemta baza
produqciis menegeri
ფასდადების პარამეტრების
მიმდინარეობს
ფასდადების
ბანსაზღვრა
ფასდადების
ბანსაზღვრა
ფასდადების
ბანსაზღვრა



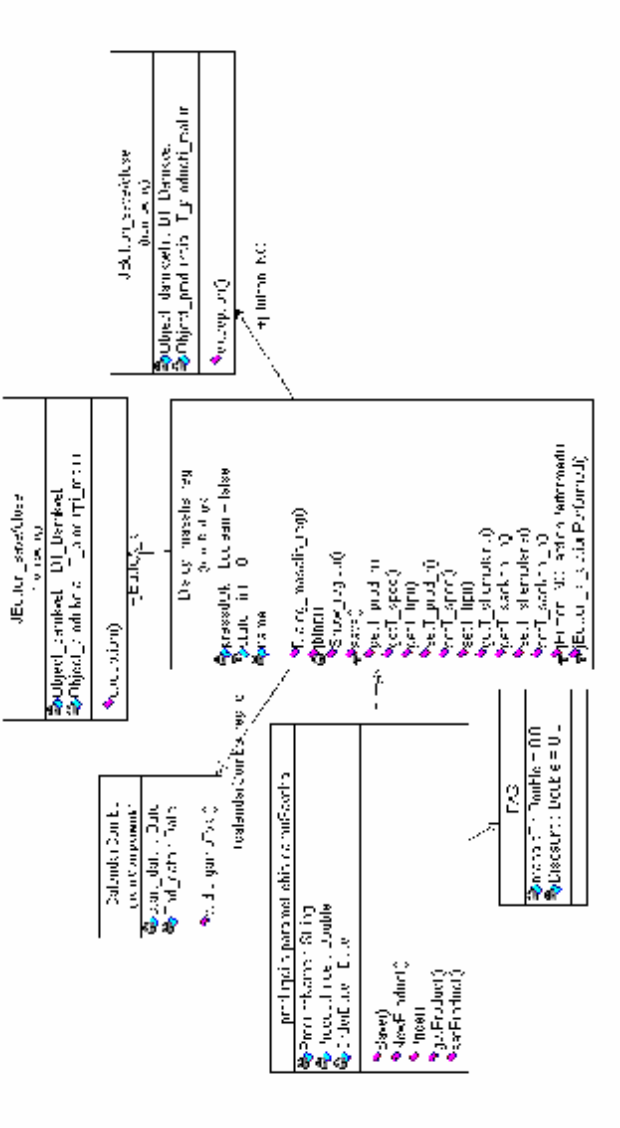
სურ. 4.11. შეკვეთის ფასის ფორმირების ქვედიაგრამის ფრაგმენტი პრეცედენტისთვის - პროდუქციის პარამეტრების დამუშავება



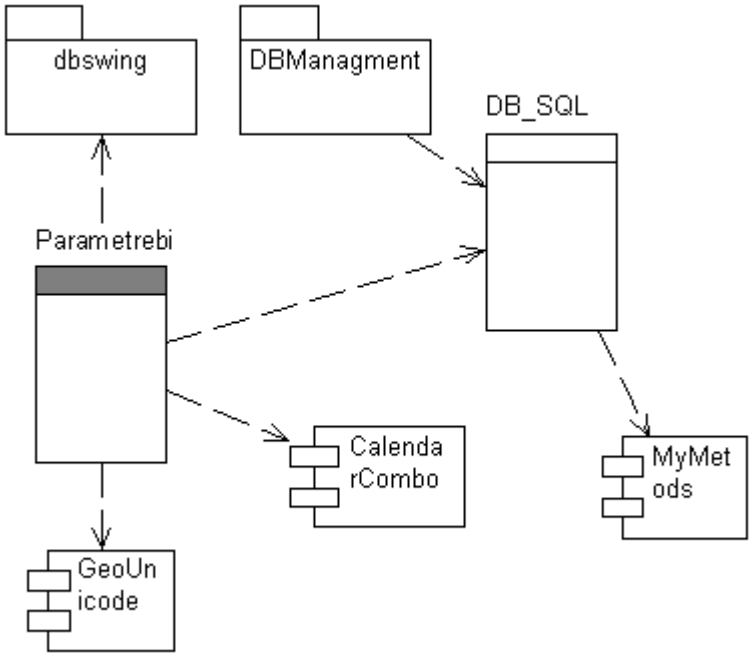
სურ. 4.12. მიმდევრობითობის დიაგრამის ფრაგმენტი პრეცედენტისთვის - პროდუქციის პარამეტრების დამუშავება



სურ. 4.13. კოოპერაციის დიაგრამის ფრაგმენტი პრეცედენტისთვის - პროდუქციის პარამეტრების დამუშავება



სურ. 4.14. კლასთა დიაგრამის ფრაგმენტი პრეცედენტისთვის - პროდუქციის პარამეტრების დამუშავება



სურ. 4.15. კომპონენტური დიაგრამის ფრაგმენტი პრეცედენტისთვის - პროდუქციის პარამეტრების დამუშავება

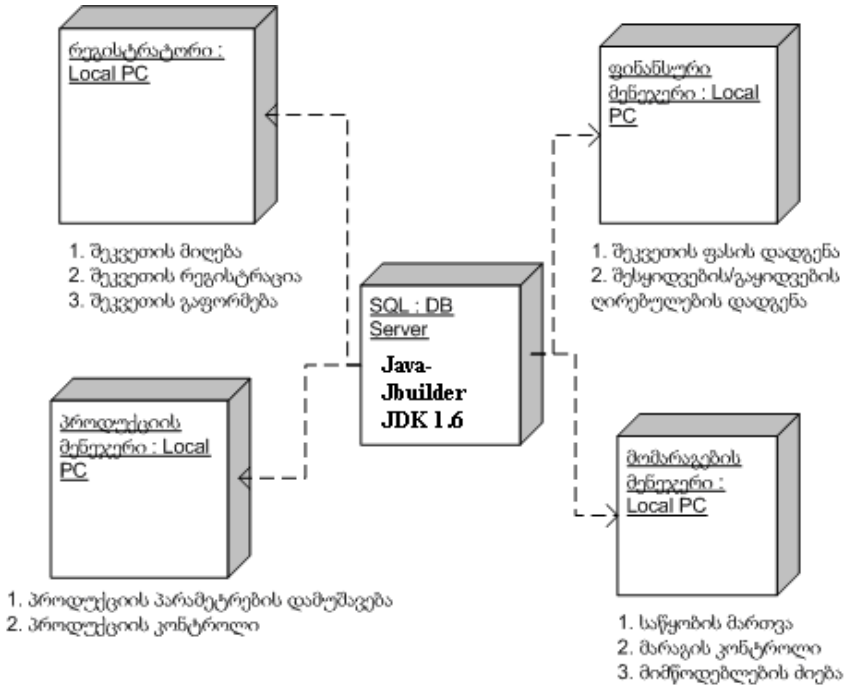
JDialog_ProduqciisParametrebisDamuSaveba.java

```java
package DIALOGS;

import Dialogs.Dialog_masalis_reg;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;

public class JDialog_ProduqciisParametrebisDamuSaveba
{
   private String ProductName;
   private Double ProductPrice;
   private Date OrderDate;

   /**
    * @roseuid 49C0EA8C036B
    */
   public JDialog_ProduqciisParametrebisDamuSaveba()
   {

   }
   public JButton_save/close Save()
   {
    return null;
   }

   public ActionEvent NewProduct()
   {
    return null;
   }
   public Double Price()
   {
    return null;
   }
```

JDialog_ProduqciisParametrebisDamuSaveba.java

სურ. 4.16. კოდის გენერაციის ფრაგმენტი კლასისთვის - პროდუქციის პარამეტრების დამუშავება

იმისათვის, რომ რეალიზებულ იქნეს ჩვენს მიერ წარმოდგენილი სცენარი, პროგრამულ ინსრუმენტად ვიყენებთ პროგრამულ პაკეტს Java, მონაცემთა ბაზის მართვის სისტემას MS Sql Server. ასევე, ჩვენს სისტემაში გათვალისწინებულია განაწილებული სამუშაო ადგილები. სურათზე 4.17 ნაჩვენებია სისტემაში მონაწილე როლების სამუშაო ადგილების მიხედვით განთავსების დიაგრამა შესაბამისი ფუნქციების მითითებით.



სურ. 4.17. სისტემის განთავსების დიაგრამის ფრაგმენტი

გამოყენებული ლიტერატურა

1. სურგულაძე გ., დოლიძე თ., ყვავაძე ლ. კომპონენტურ-ვიზუალური დაპროგრამება, სტუ, თბილისი 2006.

2. სურგულაძე გ., თურქია ე. ბიზნეს-პროცესების მართვის ავტომატიზებული სისტემების დაპროექტება, მონოგრაფია, სტუ, თბილისი, 2003.

3. Леоненков Л. самоучитель UML - http://khpi-iip.mipk.kharkiv.edu/library/case/leon/index.html - უკანასკნელად გადამოწმებულ იქნა 12.02.2009.

4. UML диаграммы в Rational Rose - http://www.interface.ru/home.asp?artId=4710 - უკანასკნელად გადამოწმებულ იქნა 10.02.2009.

5. Трофимов С.А. CASE-технологии. Практическая работа в Rational Rose, Москва 2002.