გია სურგულაძე

g.surguladze@gtu.ge

სტუ-ს ინფორმატიკის ფაკულტეტის „მართვის ავტომატიზებული სისტემების“ მიმართულების ხელმძღვანელი, სრული პროფესორი, ტექნიკის მეცნიერებათა დოქტორი. 300-მდე სამეცნიერო-პედაგოგიური ნაშრომის ავტორი, მათ შორის 53 წიგნი, 37 ელექტრონული სახელმძღვანელო, 200-ზე მეტი სამეცნიერო სტატია/თეზისი/პროექტი - პრაქტიკული და გამოყენებითი ინფორმატიკის სფეროში. ბერლინის ჰუმბოლდტის და ნიურნბერგ-ერლანგენის უნივრსიტეტების მიწვეული პროფესორი (1991-2013 წლებში). მსოფლიო ბანკის და USAID-ის ექსპერტი, ბიზნეს-ანალიტიკოსი.

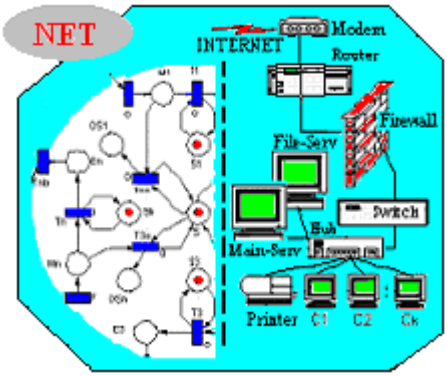
ირაკლი ბულია

i.bulia@gtu.ge

სტუ-ს ინფორმატიკის ფაკულტეტის „მართვის ავტომატიზებული სისტემების“ მიმართულების მოწვეული ასოცირებული პროფესორი. კათედრის ყოფილი კურსდამთავრებული, მაგისტრი და დოქტორანტი. აკადემიური დოქტორის ხარისხი დაიცვა 2012 წ. სტუ-ში. არის 3 წიგნისა და 15 სამეცნიერო ნაშრომის ავტორი. საქართველოს „რესპუბლიკა“ ბანკის IT-განვითარების განყოფილების უფროსი. აქვს თეორიული და პრაქტიკული მუშაობის გამოცდილება დაპროგრამების ობიექტ-ორიენტირებული, ვიზუალური და სერვის-ორიენტირებული ენების C#, C++,VB, ASP, ADO გამოყენებით .NET-პლატფორმაზე.

გია სურგულაძე,
ირაკლი ბულია

კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება



„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, ირაკლი ბულია

კორპორაციულ Web-აპლიკაციათა ინტეგრაცია და დაპროექტება

დამტკიცებულია:
სტუ-ს სარედაქციო-
საგამომცემლო
საბჭოს მიერ

თბილისი
2012

უაკ 004.5

განიხილება კორპორაციულ და კორპორაციათაშორის სისტემებში ინფორმაციის გაცვლის თანამედროვე საშუალებები მათი ინტეგრაციის მიზნით. გაანალიზებულია მეთოდები, არქიტექტურა, პრინციპები, მოდელები, რომლებიც აიოლებს ინტეგრაციის საკითხებს, ხდის უფრო ეფექტურს. მათი საშუალებით შესაძლებელია დამოუკიდებელ სისტემებს და აპლიკაციებს შორის მონაცემთა გადაცემა და ტრანსფორმაცია, ბიზნესპროცესების მენეჯმენტი და ავტომატიზაცია. კორპორაციებში არსებულ პროგრამულ უზრუნველყოფებს შორის ინტეგრაციის საკითხის გადაწყვეტა შემოთავაზებულია თანამედროვე მეთოდების, კერძოდ სერვისზე ორიენტირებული არქიტექტურის იმპლემენტაციით, რომლის განხორციელებისთვის გამოიყენება სხვადასხვა ტექნოლოგია, როგორიცაა: Web-სერვისები, ორგანიზაციის სერვისული არხი (ESB), BPMN/UML, BizTalk, XML, CPN, WPF/XAML და სხვ. აღნიშნული ტექნოლოგიები უზრუნველყოფს ინტეგრაციულ სისტემაში ჰიბრიდული აპლიკაციების შექმნას და მათ ურთიერთქმედებას, სხვადასხვა ფორმატის მონაცემთა გაცვლას, სისტემის ფუნქციონირების მდგრადობას, მოქნილობას, მასშტაბირებას.

მონოგრაფია განკუთვნილია მართვის საინფორმაციო სისტემების (Management Information Systems) დოქტორანტების, მაგისტრანტების, აგრეთვე Web-ტექნოლოგიების შექმნის საკითხებით დაინტერესებული სპეციალისტებისა და მკითხველთათვის.

რეცენზენტები:

- პროფ. ზურაბ გასიტაშვილი (სტუ)
- პროფ. გიორგი გოგიჩაიშვილი (სტუ)
- ასოც.პროფ. დავით გულუა (თსუ)

პროფ. გია სურგულაძის რედაქციით

ISBN 978-9941- 20-165-3

Georgian Technical University

Gia Surguladze, Irakli Bulia

INTEGRATION AND BUILDING OF WEB-APPLICATIONS FOR ENTERPRISES

Supported by DAAD (Germany)

ISBN 978-9941- 20-165-3

ABSTRACT

The work considers modern means of information exchange on the basis of their integration in the enterprise and inter-enterprise systems. There are discussed methods, architectures, principles, models that facilitate the integration issues, making them more efficient. They serve in data transfer and conversion, management and automation of business processes between different systems and applications. Establishing of the integration between the existing software in enterprises, is offered by modern methods, in particular through the implementation of Service Oriented Architecture, which is realized by means of different technologies such as: Electronic Data Interchange(EDI), Enterprise Service Bus (ESB), XML, BPEL. These technologies enable the integration between interacting applications across multiple platforms, data interchange of different formats, reliability, flexibility, scalability.

The work presents the problems of inter-corporate business process management. The questions of business processes integration, information composition and synchronization related to multi-application environment are suggested. Considering the inter and intra-corporate management aspects, main attention is given to the strategies of software product development in the organizational systems. To provide the Integration of the horizontal and vertical management in the inter-corporate applications, the possibilities of service-oriented architecture and its practical realization examples are described. Based on the business-processes model, which occurs between the Tax Service of Ministry of Finance and the Banks, formation of the incapsulated business-functions in the web-servises and the inter-corporate application process functioning examples are shown.

Questions of a unified modeling of inter-corporate business processes with service-oriented architecture and display them in the time-stochastic Petri nets, with the aim of further research. The task of transformation models is proposed based on the tools BPMN -> Activity-D -> PetNet. Investigated the temporal characteristics of business process execution system that provides the basis for correct decision making to further improve the service providing organization. In order to implement such a

system using BizTalk platform in the working environment MsVisualStudio.NET.

The problems of modeling and studying the processing of requests in the banking corporation based on service applications and systems analysis. Introduced the concept of building an integrated automated UML-based standards and service-oriented architecture. It is proposed the use of colored Petri nets (CPN) to construct a simulation model of inter-corporate service processes and study the temporal characteristics of its functioning.

Object-role modeling problem for intercorporate automated systems and its implementation is considered in the article. The creation concept for the distributed databases for various financial structures on the basis of Service-oriented architecture is offered. The web-application forms on Ms Visual Studio.NET platform by The NORMA software and Ms SQL Server package are realized as a result.

Proposed for financial banks, large shopping centers and other objects the service- and e-commerce- processes information flows required for the object - oriented analysis and design, the user interfaces of software implementation of the integrated use of modern information technology. Information objects for meta description is given in accordance with the SQL-standard. The deployment of the technology described in the distributed data warehouse.

The Concept of effective system selection & implementation was formulated, as pre-implementation stage we introduced 3 different stages which are 1) assessing organization’s readiness to implement an ERP system 2) Selecting a desirable ERP system using different analytical tools and 3) Company diagnostics to ensure that company’s goals and objectives are met with implementing the system. Also, system realization means have been established. So, ERP system 4 stage implementation methodology was elaborated, in contrast with existing implementation methodologies, main emphasis was made on pre-implementation stages necessary for successful system implementation.

მადლიერება

გვინდა განსაკუთრებული მადლიერება გამოვთქვათ ჩვენი ძირითადი კონსულტანტის, ***პროფესორ ზურაბ ჩხაიძის*** მიმართ, რომელიც ფასდაუდებელ რჩევებს გვაძლევდა საბანკო სფეროში ახალი ინფორმაციული ტექნოლოგიების გამოყენების მიზნით. იგი იყო ი. ბულიას აკადემიური დოქტორის ხარისხის მისანიჭებელი დისერტაციის პირველი ოპონენტი და წინამდებარე მონოგრაფიის ერთ-ერთი პირველი მკითხველი. მისი მოულოდნელი გარდაცვალებით უდიდესი დანაკლისი განიცადა როგორც საქართველოს ტექნიკურმა უნივერსიტეტმა და ჩვენმა კათედრამ (იგი „მართვის ავტომატიზებული სისტემების“ კათედრის კურსდამთავრებული, ასპირანტი და შემდგომ მისი ლექტორი იყო), ისე ქართულმა საბანკო საზოგადოებამ (წლების განმავლობაში იგი ხელმძღვანელობდა ინფორმაციული ტექნოლოგიების დეპარტამენტს „ვითიბი“ ბანკში, იყო სტუს „საბანკო საქმის“ კათედრის თანადამაარსებელი, ხელმძღვანელი).

მადლობას ვუხდით ორგანიზაციული მართვის დეპარტამენტის უფროსს, ბატონ გიორგი გოგიჩაიშვილს და „მართვის ავტომატიზებული სისტემების" მიმართულების კოლეგებს, განსაკუთრებით სრულ პროფესორ ეკატერინე თურქიას და ასოცირებულ პროფესორ ნინო თოფურიას, რომელთა მხარდაჭერა და საქმიანი რჩევები ხელს გვიწყობდა არაერთი პრაქტიკული პრობლემის გადაწყვეტის საქმეში და მიზნის მიღწევაში. მადლობა ასევე ჩვენს სტუდენტებს და მაგისტრანტებს, რომლებიც გულმოდგინედ გვეხმარებოდნენ პროგრამული სისტემების რეალიზაციისა და მათი ტესტირების ამოცანების შესრულებაში.

შინაარსი

შესავალი

ელექტრონული ბიზნესის მართვის თანამედროვე საინფორმაციო სისტემებში პროგრამულ უზრუნველყოფათა უმრავლესობა დაკავშირებულია ერთმანეთთან. მათ შორის წარმოებს ინფორმაციის მიმოცვლა. ეს აპლიკაციები შესაძლებელია განთავსებული იყოს როგორც კორპორაციის შიგა, ისე გლობალურ ქსელში. ორგანიზაციები ფლობს პროგრამებს, რომლებიც სხვადასხვა მიზანს ემსახურება. ერთ–ერთი მთავარია ელექტრონული სახით მონაცემების გაცვლა (Electronic Data Interchange) როგორც კომპანიებს, ისე სახელმწიფო სტრუქტურებსა და ტერიტორიულად დაშორებულ განყოფილებებს შორის. ყველა ამ პროგრამას ესაჭიროება გარკვეული სახის კავშირი სხვა სისტემებთან, რაშიც მონაცემების გაცვლა-დამუშავება იგულისხმება, გარკვეული ბიზნესლოგიკის შესაბამისად.

ბიზნესპროცესების მართვის სხვადასხვა ტიპის აპლიკაციები, როგორიცაა პროდუქციის მიწოდების (სასაწყობო მარაგების მართვის და მიწოდების ორგანიზება), კლიენტებთან ურთიერთქმედების სისტემები (არსებულ და პოტენციურ კლიენტებთან), ბუღალტრული აღრიცხვის, საწარმოთა რესურსების მართვის, Business Intelligence-ს სისტემებთან (OLAP), ადამიანური რესურსების მართვის, ჯანდაცვის და სხვა აპლიკაციები, როგორც წესი, საჭიროებს ერთმანეთთან ურთიერთობას, მონაცემების გაცვლის და სხვადასხვა ბიზნესპროცესის მართვის კუთხით. ეს სისტემები მუშაობს სხვადასხვა ოპერაციული სისტემის სერვერებზე, სხვადასხვა სახის მონაცემთა ბაზებზე, შესრულებულია დაპროგრამების განსხვავებული ენების, განსხვავებული ტექნოლოგიების გამოყენებით ან მოძველებულ სისტემებზე, რომელთა

განვითარება და მხარდაჭერაც აღარ ხდება. ამის გამო რთულია მათ შორის პროცესების მართვა, ავტომატიზაცია. ასევე ხდება მონაცემების დუბლირება, მათი სხვადასხვა სისტემაში განთავსების გამო.

ზემოაღნიშნულიდან გამომდინარე განსაკუთრებით აქტუალურია კორპორაციაში ბიზნესპროცესების მართვის ელექტრონული სისტემების ინტეგრაციის (Enterprise Application Integration) პრობლემის გადაწყვეტა, რომლის მიზანია ორგანიზაციაში არსებული სისტემების დაკავშირება, აპლიკაციების ურთიერთკავშირის გამარტივება და ბიზნესპროცესების ავტომატიზაციის უზრუნველყოფა. ასეთი კავშირების სარეალიზაციოდ იქმნება პროგრამები, რომლებიც ასრულებს შუალედურ რგოლს ამ სისტემებს შორის. მსგავსი ბროკერული აპლიკაციების რაოდენობა მით უფრო იზრდება, რაც უფრო იზრდება ორგანიზაციაში სისტემების და, ასევე, გარე ორგანიზაციებთან არსებული კავშირები. საჭირო ხდება ამ ბროკერული აპლიკაციების შექმნა, იზრდება მათი ადმინისტრირების, მონიტორინგის, პროგრამული უზრუნველ-ყოფის ცვლილებების მართვის სამუშაოები, რაც იწვევს ინფორმაციული ტექნოლოგიების რესურსების დიდი რაოდენობით მოხმარებას. საჭირო ხდება როგორც ადამიანური რესურსების, ასევე ინფრასტრუქტურული, პროგრამული უზრუნველყოფების მუდმივი განახლება.

ელექტრონული ბიზნესის სისტემები განსხვავებული ფორმატის მონაცემებს იძლევა გარე სისტემებთან დასაკავშირებლად. შესაძლებელია ეს იყოს ტექსტური ან ორობითი ფაილი, მონაცემები იყოს XML ფორმატში, Web-სერვისების საშუალებით იძლეოდეს მონაცემებზე წვდომის საშუალებას, ან ბაზის სხვადასხვა პროცედურის და ფუნქციის საშუალებით, ამასთანავე მონაცემთა გადაცემა ხდებოდეს TCP,

HTTP პროტოკოლების გამოყენებით ან მონაცემთა გადაცემის სხვა საშუალებებითაც. საჭიროა განსხვავებული ინტეგრაციული არქიტექტურის შექმნა, რათა კომპანიამ უფრო მარტივად უზრუნველყოს ინტეგრაციის ამოცანები ნაკლები რესურსების გამოყენებით.

ამოცანა არის ისეთი სისტემის შექმნა, რომელიც უზრუნველყოფს არსებული სისტემების ინტეგრაციას, როგორც ორგანიზაციებს შორის, ისე ორგანიზაციის შიგნით, რომელიც უზრუნველყოფს სხვადასხვა ფორმატის მონაცემთა მიღება-გადაცემა-დამუშავებას, დიდ წარმადობას და საიმედოობას.

ნაშრომის მიზანია კორპორაციული და ინტერ–კორპორაციული ბიზნესპროცესების მართვის Web-აპლიკაციების დაპროექტება და რეალიზაცია ელექტრონული სისტემების ინტეგრაციის შესაძლებლობით და სერვისზე ორიენტირებული არქიტექტურით.

მიზნის მისაღწევად ნაშრომში განიხილება შემდეგი ძირითადი ამოცანები:

- არსებული თანამედროვე ინტეგრაციის სისტემების ანალიზი და შესაბამისი ინფორმაციული ტექნოლოგიების კლასიფიკაცია, ობიექტზე, პროცესზე და სერვისზე ორიენტირებული დაპროექტების პრინციპებით;

- შიგაკორპორაციული და კორპორაციათაშორისი ბიზნესპროცესების ტრადიციული და სერვისზე ორიენტირებული მოდელების აგება სისტემური ანალიზის საფუძველზე, BPMN და UML ტექნოლოგიების ბაზაზე. მათი შედარებითი ანალიზი;

- Web-აპლიკაციების დასაპროექტებლად ორგანიზაციის სერვისული ბიზნესპროცესების ობიექტზე ორიენტირებული მოდელირება კლასების, ობიექტების და Web-მეთოდების ფორმალიზაციის საფუძველზე, პოლიმორფიზმის, მემკვიდრეო–ბითობის და ინტერფეისული თვისებების გათვალისწინებით;

- სერვისზე ორიენტირებული კორპორაციული Web-აპლიკა–ციების მონაცემთა განაწილებული ბაზების სტრუქტურების დასაპროექტებლად ობიექტ-როლური მოდელების (ORM) აგება და კვლევა რევერსიული CASE ტექნოლოგიების გამოყენებით;

- სერვისზე ორიენტირებული არქიტექტურის კორპორაციული სისტემის ბიზნესპროცესების უნიფიცირებული მოდელების ასახვის (BPMN -> Activity-D -> PetNet) ალგორითმების შემუშავება სტოქასტური, დროითი პეტრის ქსელის გრაფებში, მათი პროცესების შესრულების ეფექტურობის შეფასების და სერვისული უზრუნველყოფის შემდგომი სრულყოფის გადაწყვეტილების მიღების მიზნით;

- სერვისზე ორიენტირებული არქიტექტურის ინტერკორ–პორაციული სისტემის ინფორმაციული ნაკადების გაცვლის სერვისული ბიზნეს-პროცესების იმიტაციური მოდელის აგება და მისი ფუნქციონირების დროითი მახასიათებლების კვლევა ფერადი პეტრის ქსელების (CPN) გამოყენებით;

- პროექტის შედეგების საფუძველზე ექსპერიმენტული პროგრამული სისტემის რეალიზაცია Ms Visual Studio .NET პლატფორმაზე, ASP.NET, ADO.NET, MsSQL_Server, C#.NET, Natural ORM Architect და BizTalk პროგრამული პაკეტების გამოყენებით.

კვლევის შედეგების გამოყენება შესაძლებელია საფინანსო-საბანკო ორგანიზაციების, ფინანსთა სამინისტროს შემოსავლების სამსახურის, დიდი სავაჭრო ცენტრების და სხვა ობიექტების სერვის-პროცესებისა და ელექტრონული კომერციის მართვის საინფორმაციო სისტემებში.

კვლევის პროცესში ვიყენებდით უახლეს და ტრადიციულ ინფორმაციულ ტექნოლოგიებს, როგორიცაა სისტემური ანალიზის მეთოდი, ობიექტზე ორიენტირებული მოდელირების, ანალიზის და დაპროექტების მეთოდები, უნიფიცირებული მოდელირების ენა და მისი რეალიზაციის ინსტრუმენტები, პეტრის ქსელების

მათემატიკური მოდელი, გრაფთა თეორია, იმიტაციური მოდელირება ფერადი პეტრის ქსელებით, ობიექტზე ორიენტირებული დაპროგრამების თეორია, მონაცემთა განაწილებული ბაზების თეორია, ობიექტ-როლური მოდელირება, კლიენტ-სერვერული არქიტექტურა, სერვისზე ორიენტირებული არქიტექტურა და სხვა.

ნაშრომის მეცნიერული სიახლე მდგომარეობს საფინანსო–საბანკო ინტერკორპორაციული მენეჯმენტის ბიზნესპროცესების მართვის Web-აპლიკაციების დასაპროექტებლად და სარეალიზაციოდ სერვისზე ორიენტირებული, უნიფიცირებული მოდელების და პროცესზე ორიენტირებული მეთოდების დამუშავება ელექტრონული სისტემების ინტეგრაციის მიზნით.

- აგებულ იქნა ელექტრონული ბიზნესის მართვის სერვისზე ორიენტირებული მოდელები პროცესზე ორიენტირებული ტექნოლოგიების გამოყენებით და ჩატარდა მათი შედარებითი ანალიზი თანამედროვე ტრადიციულ მოდელებთან;

- სერვისორიენტირებული მოდელების ბაზაზე დაპროექტებული Web–აპლიკაციისთვის აგებულ იქნა სტოქასტიკურ–დროითი პეტრის ქსელის მათემატიკური მოდელი, რომლის საფუძველზე ჩატარდა ბიზნესპროცესების უნიფიცირებული მოდელების კველვა;

- შემუშავებულ იქნა საფინანსო–საბანკო ორგანიზაციების სერვისზე ორიენტირებული ბიზნესპროცესების მართვის მოდელები კლასების პოლიმორფიზმითა და მემკვიდრეობითობით, ასევე ობიექტ–როლური მოდელები მონაცემთა ბაზების ავტომატიზებული აგების მიზნით რევერსული CASE ტექნოლოგიების გამოყენებით.

მიღებულ შედეგებს აქვს პრაქტიკული ღირებულება, კერძოდ ის შეიძლება გამოყენებულ იქნას როგორც საფინანსო–საბანკო და კომერციულ დაწესებულებებში, ისე ნებისმიერი დარგის

კორპორაციულ ორგანიზაციებში, რომლებიც სერვისზე ორიენტირებული არქიტექტურის პრინციპებით ფუნქციონირებას გამოიყენებს. განსაკუთრებით ღირებულია ასეთი მიდგომა ინტერკორპორაციული მენეჯმენტის ბიზნეს–პროცესების მართვისთვის.

წიგნის **პირველი თავი** ეხება ელექტრონული ბიზნესის მართვის თანამედროვე საინფორმაციო სისტემების განხილვას. გადმოცემულია დღეისათვის მსოფლიოში არსებული ტრადიციული, ობიექტორიენტირებული, პროცესორიენტირებული და სერვისორიენტირებული მიდგომებით შექმნილი საინფორმაციო სისტემები და მათი შესაბამისი ინფორმაციული ტექნოლოგიები.

ჩატარებულია მათი კრიტიკული ანალიზი კორპორაციული და ინტერკორპორაციული მენეჯმენტის სერვისული ბიზნესპროცესების მოდელირების, დაპროექტების და Web–აპლიკაციების პროგრამული რეალიზაციის მიზნით, მათი მოქნილი ინტეგრაციის თვალსაზრისით. ამ ამოცანის რეალიზაციისათვის საჭიროა გამოყენებულ იყოს ინტეგრაციის აპრობირებული მეთოდები, პროგრამული უზრუნველყოფის არქიტექტურის მოდელები, სტანდარტები, რომლებიც უზრუნველყოფს მოქნილობას, მასშტაბირებას, არაერთგვაროვანი და კომპლექსური სისტემების ურთიერთკავშირს.

ნაშრომის **მეორე თავში** განიხილება საფინანსო–საბანკო კორპორაციული მენეჯმენტის სერვისული ბიზნესპროცესების მოდელირება ტრადიციული და Web–სერვისული მეთოდებით. კერძოდ, შემუშავებულია ინტერკორპორაციული აპლიკაციების ჰორიზონტალური და ვერტიკალური ინტეგრაციის მართვის მოდელები და მეთოდები სერვისზე ორიენტირებული არქიტექტურის ბაზაზე [2].

სერვისორიენტირებული მიდგომა ობიექტორიენტირებულ მეთოდოლოგიაზე რეალიზებული სისტემის გაფართოების ხერხია, ამდენად პროგრამული უზრუნველყოფის სისტემის მოდელირებისას, ძირითადად, ისევ გამოიყენება ობიექტზე ორიენტირებული დაპროექტების და მოდელირების ტექნოლოგია (პროტოტიპული პრინციპი, ვერსიების მართვა და ა.შ.), როგორც სისტემის შინაარსობრივი, ისე Web-სერვისის ფუნქციონალური აღწერისთვის. სისტემის სერვისზე ორიენტირებული არქიტექტურის მხარე აღიწერება ბიზნეს–პროცესების მოდელირების ნოტაციისა და ბიზნეს-პროცესების შესრულების ენის საფუძველზე.

ბიზნესპროცესების მოდელირების ნოტაცია ობიექტზე ორიენტირებულ მოდელირებას აფართოებს სისტემის სერვისზე ორიენტირებული ფუნქციონირების აღწერით, როგორც მრავალაპლიკაციური, ისე ჰორიზონტალური და ვერტიკალური დაპროექტების ჭრილში. Web-სერვისების გამოყენებით, მრავალაპლიკაციურ რეჟიმში ბიზნესპროცესების წარმოების ერთიანი ელექტრონული სისტემის ფუნქციონირების საილუსტრაციოდ, განვიხილავთ ფინანსთა სამინისტროს შემოსავლების სამსახურისა და საფინანსო ბანკების ინფორმაციული კომპოზიციის მაგალითს. ამ ორი მასშტაბური სტრუქტურის ფუნქციონალური ურთიერთობა იკვეთება გადასახადის გადამხდელი კლიენტის ბანკში ანგარიშის გახსნის, დახურვის, ცვლილებების, საგადასახადო დავალებების ოპერაციების წარმოების დროს. აღნიშნული ამოცანებისთვის შემუშავებულია ბიზნესპროცესების მოდელები (BPMN) და უნიფიცირებული მოდელირების ენის აქტიურობათა და მიმდევრობითობის დიაგრამები (UML).

მესამე თავში გადმოცემულია ინტერკორპორაციული სერვისების ბიზნესპროცესების აქტიურობათა დიაგრამების

პროექტირების საკითხები, რომლებიც დინამიკურ მოდელებს მიეკუთვნება. ისინი გამოკვლეულია მიზეზ–შედეგობრივი პროცესების შესრულების დროის მახასიათებლების საფუძველზე, პეტრის სტოქასტური და ფერადი ქსელების გამოყენებით (CPN).

მეოთხე თავში ვიხილავთ სერვისზე ორიენტირებული კორპორაციული Web-აპლიკაციების მონაცემთა განაწილებული ბაზების სტრუქტურის დაპროექტების პრობლემებს და მათი გადაწყვეტის ახალი ტექნოლოგიებს, კერძოდ, შემუშავებულია კორპორაციული მართვის სისტემის ობიექტ-როლური მოდელი (ORM) კატეგორიალური ანალიზის საფუძველზე და Natural ORM Architect ინსტრუმენტის გამოყენებით. ასევე ჩატარდა კვლევა ავტომატიზებულ რეჟიმში აგებული MsSQL Server მონაცემთა ბაზების სტრუქტურის ადექვატურობის შემოწმებაზე რევერსიული CASE ტექნოლოგიის გამოყენებით Ms VisualStudio.NET გარემოში.

მეხუთე თავში გადმოცემულია Web-აპლიკაციების პროგრამული რეალიზაციის მაგალითები, ექსპერიმენტული სადემონსტრაციო ელექტრონული ბიზნესის სისტემები. კერძოდ, განხილულია საფინანსო–საბანკო სისტემები, სავაჭრო–კომერციული ცენტრების, პროდუქციის შესყიდვა–მიწოდების კონტრაქტების რეალიზაციის სისტემები და ა.შ. ელექტრონული ბიზნესის მომხმარებელთა ინტერფეისების პროგრამული გადაწყვეტით .NET Framework 4.0 პლატფორმაზე და ASP.NET, ADO.NET, C#, XML, MsSQL_Server, BizTalk, DataWarehouse, Enterprise Architect პაკეტების ბაზაზე, ობიექტორიენტირებული, პროცესორიენტირებული და სერვისორიენტირებული არქიტექ-ტურის მოთხოვნათა გათვალისწინებით.

მეექვსე თავი ეხება კორპორაციათა მართვის სისტემის სრულყოფის ახალი ინფორმაციული ტექნოლოგიების, კერძოდ, საწარმოო რესურსების მართვის (ERP) და კლიენტებთან ურთიერთბების მართვის (CRM) სისტემების განხილვას და მათ

პრაქტიკულ გამოყენებას. წარმოდგენილია ასეთი სისტემების სტრუქტურები და მათი დაპროექტების მეთოდები.

ბიზნესპროცესების ინტეგრაცია და მათი ოპტიმიზაცია კორპორაციებს ხელს უწყობს ინტეგრაციის პროცესის რეალიზააციის მაღალი ხარჯების შემცირებაში. ამავდროულად მცირდება სპეციალური პროგრამული უზრუნველყოფის შეძენის აუცილებლობა მესამე, გარე მხარისგან, რაც ხშირად არაეფექტურია და იწვევს ფინანსური რესურსის გადაჭარბებულ ხარჯვას. განიხილება ასევე კორპორაციაში ადამიანური რესურსების მართვის პრობლემები და მათი გადაწყვეტის გზები ზემოაღნიშნული ინფორმაციული ტექნოლოგიების ბაზაზე.

მეშვიდე თავში მოცემულია კორპორაციული აპლიკაციების დეველოპმენტის ახალი ტექნოლოგიის Windows Presentation Foundation (WPF) აღწერა, რომელიც ითვლება პროგრამული ინჟინერიის ერთ-ერთ მძლავრ მეთოდოლოგიურ და ინსტრუმენტულ საშუალებად. მაიკროსოფთის ამ პროდუქტის საშუალებით ხორციელდება ჰიბრიდული სისტემების შექმნა თანამედროვე გრაფიკული დიზაინით, იშლება ზღვარი Windows- და Web- აპლიკაციებს შორის. აქ განიხილება WPF სისტემის არქიტექტურა, მისი საბაზო ელემენტები, XAML და C# ენები, როგორც კორპორაციული ჰიბრიდული სისტემების დაპროექტების (დიზაინის) და პროგრამული რეალიზაციის ძირითადი ინსტრუმენტული საშუალებანი.

წიგნის თითოეულ თავში მრავლადაა მოცემული მაგალითები და სადემონსტრაციო პროგრამები, როგორც პროგრამული აპლიკაციების (დანართების) ასაგებად, ისე მათი ფუნქციონეირების საილუსტრაციოდ.

დასასრულ გვინდა აღვნიშნოთ, რომ კორპორაციული სისტემების ინტეგრაციის პრობლემების გადაწყვეტა დღეს განსაკუთრებით პრობლემატურია მთელს მსოფლიოში, იგი

აქტუალური და ახალი მიმართულებაა მეცნიერული და პრაქტიკული ღირებულების თვალსაზრისით. იგი სწრაფად ვითარდება და მოიცავს მრავალ დარგობრივ სფეროს. ინფორმაციული ტექნოლოგიები და პროგრამული ინჟინერია კი ამ კუთხით ასრულებს კორპორაციული მართვის სისტემების ზოგადი თეორიისა და პრაქტიკულ-ექსპერიმენტული რეალიზაციის ფუნქციებს.

დასასრულ, გვინდა დიდი მადლობა გადავუხადოთ წინასწარ ჩვენს მკითხველ-კოლეგებს და სტუდენტებს საქმიანი შენიშვნებისა და სამომავლო რეკომენდაციებისთვის, რამეთუ საქართველოში ამ მიმართულების განვითარება გარდაუვალია, განსაკუთრებით მასოფლიო გლობალიზაციის პროცესში.

I თავი: კორპორაციათა მართვის პრობლემები და Web-აპლიკაციების აგების თანამედროვე ტექნოლოგიები

1.1. თანამედროვე სისტემებში ინტეგრაციის, მონაცემთა გადაცემის და დამუშავების ტექნოლოგიები

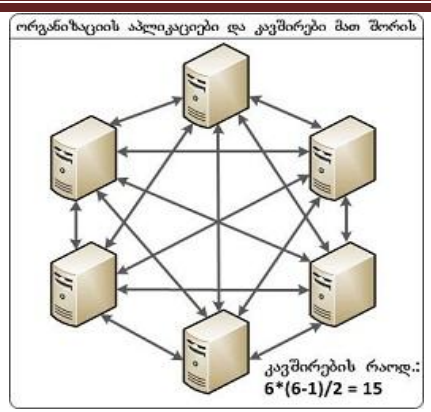
თანამედროვე საინფორმაციო სისტემების პროგრამული უზრუნველყოფები უმეტესად დაკავშირებულია ერთმანეთთან. მათ შორის იცვლება ინფორმაცია [1,2]. ეს აპლიკაციები შესაძლებელია განთავსდეს როგორც კორპორაციის შიგა, ისე გლობალურ ქსელში. ორგანიზაციები ფლობს პროგრამებს, რომლებიც სხვადასხვა მიზანს ემსახურება. ელექტრონული სახით მონაცემები იცვლება (Electronic Data Interchange), როგორც კომპანიებს შორის, ისე სახელმწიფო სტრუქტურებსა და ტერიტორიულად დაშორებულ განყოფილებებთან [9]. ყველა ამ პროგრამას ესაჭიროება გარკვეული სახის კავშირი სხვა სისტემებთან, რაშიც იგულისხმება მონაცემების გაცვლა-დამუშავება, გარკვეული ბიზნეს-ლოგიკის შესაბამისად.

სხვადასხვა ტიპის აპლიკაციები, როგორიცაა პროდუქციის მიწოდების (სასაწყობო მარაგების მართვის და მიწოდების ორგანიზება), კლიენტებთან ურთიერთქმედების სისტემები (არსებულ და პოტენციურ კლიენტებთან), ბუღალტრული აღრიცხვის, საწარმოთა რესურსების მართვის, Business Intelligence-ს სისტემებთან (OLAP), ადამიანური რესურსების მართვის, ჯანდაცვის და სხვა აპლიკაციებს, როგორც წესი, საჭიროებს ერთმანეთთან ურთიერთობას, მონაცემების გაცვლის და სხვადასხვა ბიზნესპროცესების მართვის კუთხით [10-13]. ეს სისტემები მუშაობს სხვადასხვა ოპერაციული სისტემის სერვერებზე, სხვადასხვა სახის მონაცემთა ბაზებზე, შესრულებულია დაპროგრამების განსხვავებული ენების,

განსხვავებული ტექნოლოგიების გამოყენებით, ან მოძველებულ სისტემებზე, რომელთა განვითარება და მხარდაჭერაც აღარ ხდება. რის გამოც რთულია მათ შორის პროცესების მართვა, ავტომატიზაცია. ასევე ხდება მონაცემების დუბლირება, მათი სხვადასხვა სისტემებში განთავსების გამო.

სისტემების ინტეგრაციის (Enterprise Application Integration) ამოცანაა დააკავშიროს და გაამარტივოს ამ კომპანიაში არსებული სისტემების, აპლიკაციების ურთიერთკავშირი და უზრუნველყოს ბიზნესპროცესების ავტომატიზაცია [14].

ამ კავშირების უზრუნველყოფისთვის იქმნება სხვადასხვა პროგრამები, რომლებიც ასრულებს დამაკავშირებელ, შუალედურ რგოლს ამ სისტემებს შორის. მსგავსი ბროკერული აპლიკაციების რაოდენობა იზრდება, რაც უფრო იზრდება ორგანიზაციაში სისტემების და ასევე გარე ორგანიზაციებთან არსებული კავშირები. საჭირო ხდება ამ ბროკერული აპლიკაციების შექმნა, იზრდება მათი ადმინისტრირების, მონიტორინგის, პროგრამული უზრუნველყოფის ცვლილებების მართვის სამუშაოები, რაც იწვევს ინფორმაციული ტექნოლოგიების რესურსების დიდი ოდენობით მოხმარებას. საჭირო ხდება როგორც ადამიანური რესურსების, ასევე ინფრასტრუქტურული, პროგრამული უზრუნველყოფების მუდმივი განახლება. თითოეული ბროკერული აპლიკაციის შექმნა არის ხანგრძლივი და შრომატევადი პროცესი. თუ ორგანიზაციას სჭირდება N რაოდენობის აპლიკაციების ერთმანეთთან დაკავშირება ამ აპლიკაციებს შორის ბროკერების რაოდენობამ შეიძლება მიაღწიოს N(N-1)/2 რიცხვს, რაც მაგალითად 10 აპლიკაციის შემთხვევაში არის 45, რაც საკმაოდ დიდი რიცხვია და ზემოთქმულიდან გამომდინარე რესურსების დიდ ხარჯებთან არის დაკავშირებული (ნახ.1.1).



ნახ.1.1

სხვადასხვა სისტემები განსხვავებული ფორმატის მონაცემებს იძლევა გარე სისტემებთან დასაკავშირებლად. შესაძლებელია ეს იყოს ტექსტური ან ორობითი ფაილი, მონაცემები იყოს XML ფორმატში, Web-სერვისების საშუალებით იძლეოდეს მონაცემებზე წვდომის საშუალებას, ან ბაზის სხვადასხვა პროცედურების და ფუნქციების საშუალებით, ამასთანავე მონაცემთა გადაცემა ხდებოდეს TCP, HTTP პროტოკოლების გამოყენებით, თუ მონაცემთა გადაცემის სხვა სახის საშუალებებითაც. საჭიროა სხვა სახის ინტეგრაციული არქიტექტურის შექმნა, რათა კომპანიამ უფრო მარტივად უზრუნველყოს ინტეგრაციის ამოცანები, ნაკლები რესურსების გამოყენებით.

ამოცანა მდგომარეობს ისეთი სისტემის შექმნაში, რომელიც უზრუნველყოფს არსებული სისტემების ინტეგრაციას, როგორც სხვადასხვა ორაგნიზაციებს შორის, ასევე ორგანიზაციის შიგნით, რომელიც უზრუნველყოფს სხვადასხვა ფორმატის მონაცემების მიღება-გადაცემა-დამუშავებას, მაღალ წარმადობას და საიმედოობას [18,75].

აპლიკაციების ინტეგრაციას აქვს სამი დანიშნულება:

- მონაცემთა ინტეგრაცია, რაც უზრუნველყოფს მონაცემების იდენტურობას სისტემებს შორის;
- აპლიკაციათა მიმწოდებლებზე დამოუკიდებლობა. უზრუნ-ველყოფს, რომ აპლიკაციის ცვლილების შემთხვევაში, ბიზნესპროცესი და ბიზნესწესების ხელახალი შექმნა არ იყოს საჭირო;
- ფასადის/ინტერფეისის შექმნა, რაც უზრუნველყოფს აპლიკაციებთან ერთიერთობის ერთიანი ინტერფესის შექმნას, რომელიც საშუალებას იძლევა აპლიკაციებთან კომუნიკაცია შესრულდეს მათი შიგა სტრუქტურების შესწავლის გარეშე.

ამ ამოცანის რეალიზაციისათვის საჭიროა გამოყენებული იყოს ინტეგრაციის აპრობირებული მეთოდები, პროგრამული უზრუნველყოფის არქიტექტურის მოდელები, სტანდარტები, რომლებიც უზრუნველყოფს მოქნილობას, მასშტაბირებას, არაერთგვაროვანი და კომპლექსური სისტემების ურთიერთკავშირს.

1.1.1. ორგანიზაციის აპლიკაციების ინტეგრაცია (**Enterprise Application Integration**)

ორგანიზაციის აპლიკაციების ინტეგრაცია (EAI) არის ტექნოლოგია, რომლის დანიშნულებაცაა საწარმოს აპლიკაციების ერთმანეთთან დაკავშირება, მათი ერთიან ბიზნეს-პროცესში ჩართვა, ასევე მათ შორის გადაცემული მონაცემების ფორმატის ტრანსფორმაცია [14,15,22].

EAI მეთოდის გამოყენება ხდება როდესაც საჭიროა [34]:

- აპლიკაციის აპლიკაციასთან დაკავშირება (Application to Application);
- ადამიანის სისტემასთან კავშირი (Person to System);
- ბიზნესის ბიზნესთან კავშირი (Business to Business).

EAI მეთოდის საშუალებით ხდება აპლიკაციებს შორის პირდაპირი კავშირების დამყარების აუცილებლობის გამორიცხვა [32]. მისი რეალიზაციისას ხდება სხვადასხვა მეთოდების, სქემების და სხვა პროგრამული უზრუნველყოფების გამოყენებით სხვადასხვა აპლიკაციების პროცესორიენტირებული ინტეგრაცია. განსხვავებული ფორმატის მონაცემების მიღება და ტრანსფორმაცია სპეციალური ადაპტერების საშუალებით სრულდება [36]. EAI საშუალებას იძლევა ცენტრალიზებულად განისაზღვროს ბიზნეს-პროცესების ლოგიკა, რაც საშუალებას იძლევა გარკვეული ბიზნეს-ლოგიკის ცვლილების შემთხვევაში, აპლიკაციების გადაკეთების გარეშე განხორციელდეს ახალი წესების იმპლემენტაცია.

ინტეგრაციის არხი ზრუნავს მონაცემები დამუშავდეს წინასწარ განსაზღვრული წესების შესაბამისად და შედეგები გადამისამართდეს კონკრეტული ბიზნესპროცესის შესასრულებლად [33].

1.1.2. სერვისზე ორიენტირებული არქიტექტურა (Service Oriented Architecture)

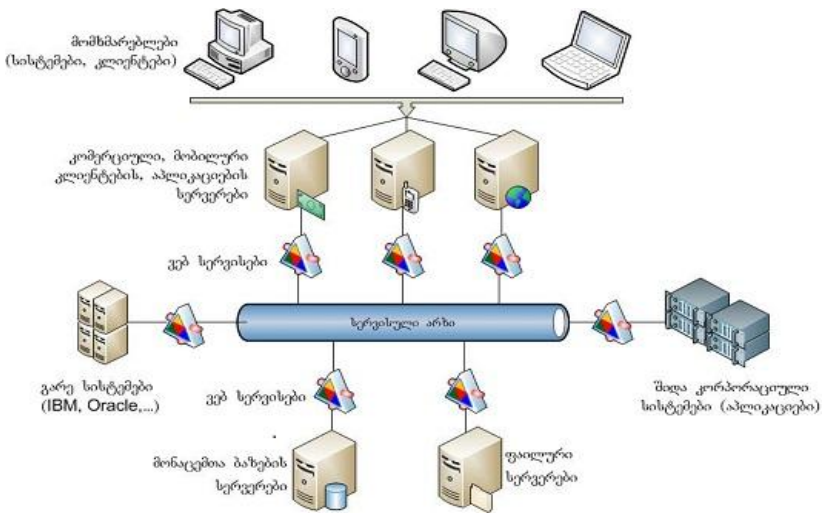
სერვისზე ორიენტირებული არქიტექტურა არის ერთმანეთთან ურთიერთმოქმედი სისტემების დაპროექტების მეთოდოლოგია [16]. EAI-ის შესაძლებლობები ფართოვდება სერვისორიენტირებული არქიტექტურის გამოყენებით, რომელიც სხვადასხვა აპლიკაციებსა და სხვადასხვა ორგანიზაციებს შორის სისტემების ინტეგრაციას და ბიზნესამოცანების შესრულებას უზრუნველყოფს [32]. SOA-ს შემთხვევაში სხვადასხვა კომპონენტები განიხილება, როგორც გარკვეული მომსახურების მომწოდებელი ან მომხმარებელი სერვისები. მისი საშუალებით ხდება განაწილებულ მონაცემთა ბაზების, Web- და სხვა სერვისების სტრუქტურირება გარკვეული ბიზნესპროცესების შესასრულებლად.

მაგალითად, ბანკში კლიენტისთვის სესხის გაცემა შედგება სხვადასხვა პროცესისგან: ბანკის კლიენტის შექმნა, ანგარიშის გახსნა, სასესხო ხელშეკრულების შექმნა და სხვა. ეს პროცესები შესაძლებელია სრულდებოდეს სხვადასხვა აპლიკაციის მიერ.

SOA-ს საშუალებით კი ეს პროცესი შესრულდება გარკვეული მიმდევრობის და ბიზნესპირობების დაცვით, რასაც ორკესტრაცია უზრუნველყოფს [19,22]. ერთიდაიგივე სერვისები შესაძლებელია გამოყენებულ იქნას სხვადასხვა ამოცანისთვის [9]. საბოლოოდ, ასეთი მოდგომა საშუალებას იძლევა დაიზოგოს რესურსები პროგრამულ უზრუნველყოფის შექმნაზე, ახალი ამოცანებისთვის უკვე არსებული სერვისების გამოყენების საშუალებით.

1.1.3. ორგანიზაციის სერვისული არხი (Enterprise Service Bus)

ESB განაწილებული სისტემების არქიტექტურული მოდელია [15]. სხვადასხვა აპლიკაციებს შორის კავშირის დასამყარებლად ამ მოდელში გამოიყენება შუალედური პროგრამული საშუალება, რომელიც უზრუნველყოფს ტრანზაქციული ოპერაციების შესრულებას, შეტყობინებათა მარშრუტიზაციას, ტრანსფორმაციას, მონაცემების დაცვას [27]. მისი საშუალებით შესაძლებელია სერვის-ორიენტირებული არქიტექტურის რეალიზაცია (ნახ.1.2). იგი შედგება დამოუკიდებელი კომპონენტებისგან, რომლებიც განთავსებულია სერვისულ არხში და საჭიროების შემთხვევაში ურთიერთქმედებს ერთმანეთთან. სხვადასხვა სისტემები შეტყობინებებს გადასცემს ორგანიზაციის სერვისულ არხს, რომელიც ახდენს მათ მიწოდების დაყოვნებას, პრიორიტეტიზაციას, გადავადებას ბუფერში მიმღები სერვისის განთავისუფლებამდე, მონაცემების სისწორის შემოწმებას, მონაცემების ტრანსფორმაციას, პროცესების კონტროლსა და შეცდომებზე რეაგირებას [30].



ნახ.1.2

1.1.4. მონაცემების ელექტრონული გაცვლა (Electronic Data Interchange)

მონაცემების ელექტრონული გაცვლა (EDI) აღნიშნავს მონაცემების ელექტრონულ დამუშავებას ელექტორნული ტრანსპორტირების მეთოდების გამოყენებით. მასში მონაწილეობენ სხვადასხვა ორგანიზაციების გამოყენებითი სისტემები. EDI სტანდარტები განსაზღვრავენ მონაცემთა გაცვლის მეთოდებს და რეგულაციას ორგანიზაციებს ან მათ ფილიალებს შორის. მისი დანიშნულებაა ციფრული ინფორმაციის გადაცემის სტანდარტიზება, სხვადასხვა დანიშნულების კომპიუტერული სისტემების პროგრამული ურთიერთქმედების უზრუნველყოფა. EDI გულისხმობს რამდენიმე ორგანიზაციას შორის შეთანხმებას გარკვეული მნიშვნელობის, ფორმატის, კავშირების საშუალებით მონაცემთა გაცვლაზე [17]. EDI-ის გამოყენების ძირითადი სარგებლობა არის მონაცემების ელექტრონული გადაცემის მაღალი

სიჩქარე, ადამიანის ჩარევის გარეშე ბიზნესს მონაცემების გადაცემა, ადამიანური შეცდომების გამორიცხვა.

არსებობს მონაცემთა გადაცემის საერთაშორისო სტანდარტები, მაგალითად SWIFT-საბანკო გადარიცხვებისთვის, GAEB-მშენებლობაში, VDA-საავტომობილო ინდუსტრიაში და სხვ. [17]. მონაცემების გადაცემა სრულდება ასევე გავრცელებული პროტოკოლებით: SMTP, HTTP, FTP და სხვ. [18].

1.1.5. ბიზნესის პროცესების მენეჯმენტი (Business Process Management)

ბიზნესპროცესების მენეჯმენტი (BPM) მოიცავს საწარმოო პროცესების იდენტიფიკაციას, დოკუმენტირებას, დაგეგმვას, მართვას და გაუმჯობესებას [20]. იგი მიმართულია არამხოლოდ ტექნიკურ, არამედ ასევე ორგანიზაციული საკითხების მოსაგვარებლად, სტრატეგიული მიმართულებების, ორგანიზაცი–ული კულტურის, მონაწილე მხარეების ინტეგრაციის და მართვის საკითხების დასახვეწად.

ბიზნესპროცესების ავტომატიზაციის პრობლემები იყოფა სამ ჯგუფად:

- კომპანიის შიგნით პროგრამული უზრუნველყოფების დაკავშირება, Enterprise Application Integration (EAI);
- სხვადასხვა კომპანიის პროგრამულ უზრუნველყოფებს შორის კავშირის დამყარება, Business-to-Business (B2B) ინტეგრაცია;
- ბიზნესპროცესების ავტომატიზაციის განზოგადებული მიდგომის უზრუნველყოფა, რომელიც განსაზღვრულია ბიზნესის პროცესების მენეჯმენტის (BPM) მიხედვით.

ბიზნეს პროცესების მენეჯმენტის მიზანია ორგანიზაციაში არსებული პროცესების გაუმჯობესება, უკეთესი მომსახურების, კლიენტების კმაყოფილების, პროდუქციის ხარისხის გაუმ–ჯობესების, წარმოების ეფექტურობის ამაღლების გზით [19,21].

1.1.6. აპლიკაციათა ინტეგრაციის სისტემები

სისტემების დაკავშირებისთვის ცალკეული ბროკერული აპლიკაციების შექმნა და მათი მართვის არაეფექტურობის გამო, უმჯობესია გარკვეული სისტემის გამოყენება, რომელიც უზრუნველყოფდა სხვადასხვა ფორმატის და არხით მიღებული მონაცემების დამუშავებას, ტრანსლიაციას, და მის გადაცემას.

ამ სისტემის დანიშნულებაა სხვადასხვა ტექნოლოგიებზე, პლატფორმებზე შექმნილი განსხვავებული ბიზნეს აპლიკაციების დაკავშირება, ბიზნესპროცესების ავტომატიზაცია და მათი მონიტორინგი [17,29,75].

ინტეგრაციის ამოცანების გადაწყვეტა შესაძლებელია სხვადასხვა სისტემის გამოყენებით. ეს სისტემები უზრუნველყოფს ორგანიზაციის სერვისული არხის (ESB), სერვისორიენტირებულ არქიტექტურის (SOA), მონაცემთა ელექტრონული გაცვლის (EDI), ორკესტრაციის, პროცესების მართვის ინსტრუმენტების საშუალებით სერვისების ინტეგრაციას, ბიზნესპროცესების დამუშავებას და ავტომატიზაციას (BPM), სერვისების, მონაცემების, პროგრამული უზრუნველყოფების დაკავშირებას. ასეთ სისტემებს შორის შესაძლებელია გამოიყოს მსოფლიოში წამყვანი პროგრამული უზრუნველყოფის მწარმოებელ კომპანიების პროდუქტები, როგორიცაა:

- JBoss Enterprise SOA Platform - RedHat-ისგან
- Oracle Enterprise Service Bus - Oracle-ისგან
- BizTalk Server – Microsoft-ისგან
- WebSphere Process Server – IBM-ისგან
- SAP Exchange Infrastructure – SAP AG-სგან
- webMethods - Software-სგან AG

ეს პროდუქტები ძირითადად განსხვავდება ოპერაციული სისტემების, მონაცემთა ბაზების ტიპების მხარდაჭერის

შესაძლებლობებით. ამ სისტემებში ინტეგრირებულია სხვადასხვა მზა ადაპტერები, რომლებიც სხვადასხვა აპლიკაციების დაკავშირების საშუალებას იძლევა, რომელთაც მონაცემთა სხვადასხვა ფორმატი გააჩნია. ასევე არის ბიზნესპროცესების სქემების აგების, ავტომატიზაციის შესაძლებლობა. აგრეთვე პროცესებზე მონიტორინგის, სხვადასხვა პარამეტრების გაზომვის ინსტრუმენტები (Business Activity Monitoring); მონაცემების გადაცემის, ტრანსფორმაციის დროს წარმოშობილი შეცდომების დიაგნოსტიკა. ასევე კონფიგურაციების საშუალებით სისტემის ფუნქციონირების, ბიზნესწესების პარამეტრიზიცია. მათ ფუნქციებს მიეკუთვნება მონაცემების დაცვის უზრუნველყოფა, უსაფრთხოება, მაღალი წარმადობა, საიმედოობა. ამ სისტემების გამოყენების უპირატესობებია:

- სისტემის ელემენტების შექმნის და ბიზნესლოგიკის დამუშავების სიმარტივე;
- მონაცემების და პროცესების მონიტორინგის შესაძლებლობა, ასევე მონაცემთა ანალიზი;
- ბიზნესის მომხმარებლის მიერ ბიზნეს-ლოგიკის ცვლილების შესაძლებლობა;
- მონაცემთა უსაფრთხო მიღება-გადაცემა;
- სერვერების ჯგუფების შექმნა დატვირთვების გასანაწილებლად და საიმედოობის გასაზრდელად და სხვა.

შეიძლება დავასკვნათ, რომ მზარდი მოთხოვნა ინტეგრაციულ სისტემებზე, მონაცემთა დამუშავების ავტომატიზაციის ამოცანების, სქემების, მიდგომების, პრინციპების მუდმივი ცვლილება, ასევე ქსელური და აპარატურული ტექნოლოგიების შესაძლებლობების ზრდა, განაპირობებს ამ კუთხით წარმოებული პროდუქტების ფუნქციონალურ განვითარებას.

ამრიგად, რთულ კორპორაციულ და კორპორაციათაშორის სისტემებში აუცილებელია თანამედროვე ინტეგრაციული საშუალებების გამოყენება, რომლებიც უზრუნველყოფს სერვისზე ორიენტირებული არქიტეტურის რეალიზაციას, რომელიც დაფუძნებული იქნება ორგანიზაციის ინტეგრაციის არხზე. მსგავსი მიდგომა, თავის მხრივ, განაპირობებს ელექტრონული ბიზნეს–პროცესების მსვლელობის სისწრაფის ზრდას, ეფექტურობას, საიმედოობას, უსაფრთხოებას. Ms_BizTalk წარმოადგენს ორგანიზაციის სერვისული არხის იმპლემენტაციის პლატფორმას, რომელიც გამოიყენება სხვადასხვა აპლიკაციებს შორის კავშირის დასამყარებლად. გრაფიკული ინტერფეისის საშუალებებით შესაძლებელია ბიზნეს პროცესების ორკესტრაცია. რეალიზებულია პროცესებზე მონიტორინგის ბიბლიოთეკები, როგორც პროცესების რაოდენობრივად შესაფასებლად, ისე გარკვეულ მოვლენებზე სარეაგირებლად. BizTalk-ის დახმარებით შესაძლებელია პროცესების ავტომატიზაციის და ინდუსტრიული სტანდარტების უზრუნველყოფა, რაც საშუალებას იძლევა შემცირდეს დანახარჯები და კომპლექსურობა B2B კავშირების დასამყარებლად.

1.2. კორპორაციული სისტემების ობიექტორიენტირებული, პროცესორიენტირებული და სერვისორიენტირებული დაპროექტება

კორპორაციული მენეჯმენტის საინფორმაციო სისტემები მიეკუთვნება დიდი და რთული სისტემების კლასს. კორპორაცია მრავალი საწარმოსა და ორგანიზაციისაგან შედგება, რომელთაც გააჩნია დამოუკიდებლობის საკმაოდ მაღალი ხარისხი/დონე და ამავდროულად, ორიენტირებულია კონკრეტული მიზნების შესრულებაზე. ამ მიზნების შესრულებისათვის კორპორაცია საჭიროებს ნებისმიერი სახის ინფორმაციის დაუყოვნებლივ

ფლობას და ამ ინფორმაციის მართვის შესაძლებლობას მის შემადგენლობაში მყოფი ყველა საწარმოსა და ორგანიზაცის საქმიანობის შესახებ. ასეთი კოორდინაციის არსებობა შესაძლებელია მხოლოდ ეფექტური ცენტრალიზებული კომუნიკაციების (კორპორატიული ქსელის) არსებობის შემთხვევაში [15]. კორპორაციული ქსელი ორგანიზაციის ინფრასტრუქტურაა, რომელიც უზრუნველყოფს კორპორაციული ინფორმაციის ურთიერთმიღწევადობას ორგანიზაციის სხვადასხვა ობიექტებსა და სუბიექტებს შორის.

კორპორაციული ქსელი ფუნქციონალური თვალსაზრისით არის ეფექტური გარემო აქტუალური ინფორმაციის გადასაცემად, რომელიც გადაწყვეტილებათა მისაღებადაა საჭირო. სისტემურ–ტექნიკური თვალსაზრისით ქსელი წარმოადგენს ერთ მთლიან სტრუქტურას, რომელიც რამდენიმე ურთიერთდაკავშირებული და ურთიერთმოქმედი დონისაგან შედგება: ინტელექტუალური შენობა; კომპიუტერული ქსელი; ტელეკომუნიკაციები; კომპიუტერული პლატფორმები; შუალედური პროგრამული უზრუნველყოფები (middleware); დანართები.

ქსელის შექმნისას ერთ–ერთ მთავარ პრინციპს წარმოადგენს ტიპური გადაწყვეტილებებისა და სტანდარტული უნიფიცირე–ბული კომპონენტების მაქსიმალური გამოყენება.

საერთო სისტემურ დანართებს მიეკუთვნება ინდივიდუალური სამუშაოს ავტომატიზაციის საშუალებები, რომლებიც გამოიყენება მომხმარებელთა სხვადასხვა კატეგორიების მიერ და ორიენტირებულია ტიპური საოფისე ამოცანების გადაწყვეტაზე. ესენია – ტექსტური პროცესორები, ელექტრონული ცხრილები, გრაფიკული რედაქტორები, კალენდრები, ჯიბის წიგნაკი და სხვ. როგორც წესი, საერთოსისტემური დანართები წარმოადგენს ლოკალიზებულ

პროგრამულ პროდუქტს, მარტივია ასათვისებლად და გამოსაყენებლად, რადგან ორიენტირებულია საბოლოო მომხმარებელზე.

სპეციალიზებული დანართები ორიენტირებულია ისეთი ამოცანების ამოხსნაზე, რომელთა საერთო სისტემური დანართების მეშვეობით ავტომატიზაცია რთულია ან შეუძლებლია. როგორც წესი, სპეციალიზებულ დანართებს კომპანიები იძენენ, უკვეთავენ ან თვითონ ამუშავებენ სპეციუალურად საქმიანობის სფეროს მიხედვით.

ხშირ შემთხვევებში სპეციალიზებული დანართები მუშაობის პროცესში მიმართავს საერთო სისტემურ დანართებს – ფაილურ სერვისებს, მონაცემთა ბაზებს, ელექტრონულ ფოსტას და სხვ. კორპორაციული ქსელი იძლევა ახალი დანართების მარტივი ინტეგრაციისა და მათი ეფექტური ფუნქციონირების საშუალებას [41]. მაღალი დონის კორპორაციული საინფორმაციო სისტემა უნდა ფლობდეს ღია ინფრასტურქტურას და მთავარ მახასიათებლებად უნდა გააჩნდეს - მწარმოებლურობა, მასშტაბურობა, უსაფრთხოება და მართვადობა.

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 [15].

ასეთი სისტემების ჯგუფს მიეკუთვნება 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) [32,38,42].

1.2.1. ობიექტორიენტირებული დაპროექტება (UML)

სისტემების დაპროექტებისთვის UML ენა უნივერსალურ ტექნოლოგიას წარმოადგენს. იგი სტანდარტული ნოტაციაა პროგრამული სისტემების ობიექტორიენტირებული და ვიზუალური მოდელირებისთვის.

შემუშავებულია Object Managing Group (OMG) კონსორციუმის მიერ, თუმცა, ძირითად ავტორებად და იდეოლოგებად ითვლებიან: გრედი ბუჩი (Grady Booch), ჯიმ რამბო (Jim Rumbaugh) და აივერ ჯაკობსონი (Ivar Jacobson) [32].

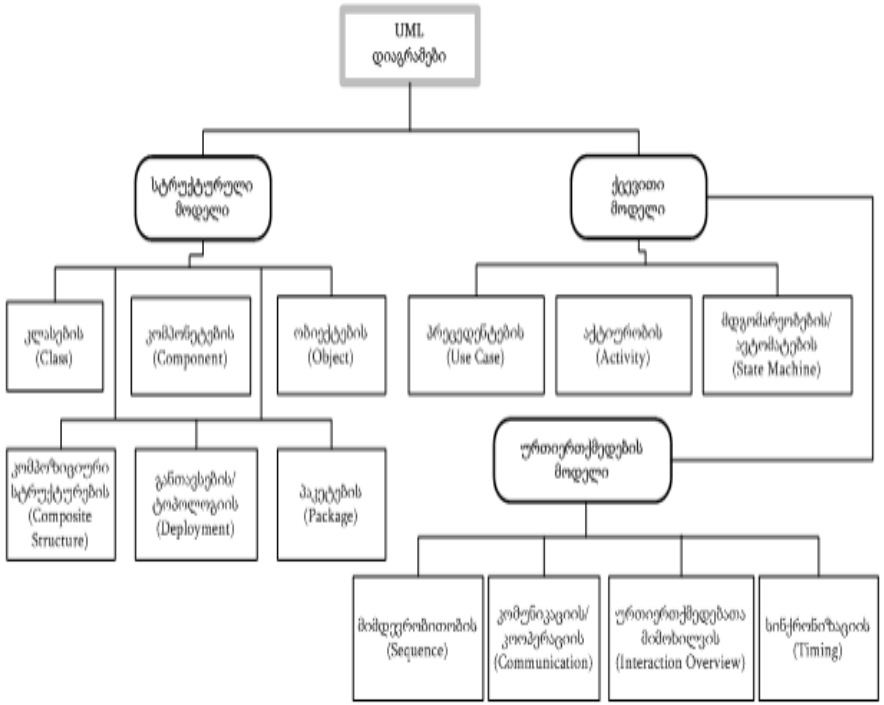
UML ტექნოლოგიის საშუალებით შესაძლებელია მართვის ობიექტის სტრუქტურის სრული სასიცოცხლო ციკლის ეტაპობრივი აღწერა, მოდელების გრაფიკული წარმოდგენიდან - პროგრამულ კოდამდე (მოთხოვნების განსაზღვრა, პროცესების მოდელირება და შესაბამისი პროგრამული ჩონჩხის მიღება) [42].

გარდა ობიექტორიენტირებული და ვიზუალური მოდელირების პრინციპებისა, ამ ტექნოლოგიის მნიშვნელოვან მიმართულებად იქცა გრაფიკული მოდელების პროგრამულ კოდად გენერაციისა და ე.წ. უკუდაპროექტების (რევერსიული) პროცესების რეალიზაცია (Model Driven Generation-MDG).

პროგრამულ ინჟინერიაში საინფორმაციო სისტემების დაპროექტებისა და მოდელირების თვალსაზრისით, UML ენა უნივერსალურ ტექნოლოგიად ითვლება და განკუთვნილია, როგორც პროგრამული სისტემების დამპროექტებლების, ისე ბიზნეს-ანალიტიკოსებისთვის.

ამ ტექნოლოგიის აქტიურმა გამოყენებამ აუცილებელი გახადა UML ენის განვითარება და ვიზუალური მოდელების

ახალი ვერსიების შექმნა (მაგ., UML2.0). UML ენის სტანდარტის მიხედვით, დიაგრამები კლასიფიცირებულია სტრუქტურული და ქცევითი მოდელების დიაგრამებად (ნახ.1.3).



ნახ.1.3. UML ენის დიაგრამების სტრუქტურა

სტრუქტურული მოდელის დიაგრამა (Structural Diagrams) განსაზღვრავს სისტემის მოდელის სტატიკურ არქიტექტურას. იგი, გამოიყენება სისტემის კლასების, ობიექტების, ინტერფეისების, ფიზიკური კომპონენტებისა და სისტემის ელემენტების ურთიერთ-დამოკიდებულების აღწერისათვის.

სტრუქტურული მოდელის დიაგრამების ძირითად შემადგენლობაშია: კლასების - Class, კომპონენტების - Component,

ობიექტების - Object, განთავსების/ტოპოლოგიის-Deployment, კომპოზიციური სტრუქტურების - Composite structure (UML2.0), პაკეტების -Package დიაგრამები.

ქცევითი მოდელის დიაგრამები (behavior diagrams) ასახავს სისტემის მოქმედებას რეალურ გარემოში და დინამიკური ნაკადების მართვას, რაც შესაძლებელს ხდის აწარმოონ დაკვირვება ოპერაციებისა და მოვლენების შესრულების ეფექტებსა და შედეგებზე.

ქცევითი მოდელის დიაგრამებია: პრეცედენტების/ შემთხვევების გამოყენების - UseCase, აქტიურობის - Activity, მდგომარეობის/ავტომატების - State Machine დიაგრამები.

ქცევითი მოდელი ასევე ნაწილდება **ურთიერთქმედების მოდელის** კატეგორიად, რომელიც შეიცავს - კომუნიკაციის/ კოოპერაციის - Communication, ურთიერთქმედებათა მიმოხილვის- Interaction_overview (UML2.0), მიმდევრობი-თობის - Sequence, სინქრონიზაციის - Timing (UML2.0) დიაგრამებს [15].

UML ენის ინსტრუმენტების ერთ-ერთი მთავარი ღირებულებაა პროგრამული უზრუნველყოფის რეალიზაცია, კერძოდ, კლასთა დიაგრამიდან პროგრამული კოდის გენერაცია. ეს პროცესი 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.

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.

საინფორმაციო და Web-ტექნოლოგიების შემდგომმა განვითარებამ საჭირო გახადა არსებული ობიექტორიენტირებული მიდგომის სრულყოფა პროცესორიენტირებული მიდგომით.

პროცესზე ორიენტირებული მიდგომის საფუძველზე ფაქტობრივად იცვლება ობიექტორიენტირებული დაპროექტების ძირითადი პრინციპები. მაგალითად, ობიექტზე ორიენტირებული მიდგომის მთავარი ფოკუსი – მონაცემი და საინფორმაციო მოდელი, პროცესორიენტირებულ მიდგომაში იცვლება პროცესითა და საპროცესო მოდელით [15].

პროცესორიენტირებული დაპროექტების მიმართულება წარმოშვა Web-ინტერფეისზე ბაზირებული სისტემების რეალიზაციის აუცილებლობამ, რაც განაწილებული ეკონომიკური სისტემების აგების ახალმა ხედვამ განაპირობა.

ეს ეხება კორპორაციული სისტემების გლობალურ გაფართოებას, განაწილების გეოგრაფიული მასშტაბების ზრდას, დისტანციურ მართვას და ა.შ.

1.2.2. ბიზნესპროცესების მოდელირება BPMN

ბიზნესპროცესების მოდელირების თვალსაზრისით UML ენა დღესდღეობით მოქნილ ტექნოლოგიად ითვლება. თუმცა, პროცეს-ორიენტირებული და სერვისორიენტირებული მიდგომის თვალსაზრისით UML ენის შემქმნელების მიერ (OMG - ObjectManagementGroup) განვითარდა და დამუშავდა ვიზუალიზაციის სპეციალური დამატებითი ელემენტების ინსტრუმენტული საშუალება, რომელიც ბიზნეს-პროცესების მართვის ნოტაციითაა (BPMN- Business Process Modeling Notation) ცნობილი [43].

ბიზნესპროცესების მართვის ნოტაციის მთავარი არსი ობიექტორიენტირებული მიდგომის ტრანსფორმაციაა პროცეს-ორიენტირებულ მიდგომაზე, რაც ბიზნესმოდელისა და საინფორმაციო მოდელის სინქრონიზაციის საშუალებას იძლევა.

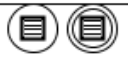
ბიზნესპროცესების მართვის ნოტაცია არის ე.წ. სტანდარტიზაციის ხიდი ბიზნესის პროცესების დაპროექტებასა და იმპლემენტაციას შორის.

ფაქტობრივად, BPMN სტანდარტი პირველ დონეზე ნაწილდება დაპროექტების ძირითადი ასპექტების მიხედვით - ორგანიზაციული სტრუქტურა, ფუნქციონალური დეკომპოზიცია და მონაცემთა მოდელი, რომელთა მთავარი ელემენტებია:

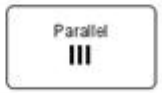
1. ობიექტთა ნაკადი;
2. დამაკავშირებელი ობიექტები და ლოგიკური ელემენტები;
3. როლები, მცოცავი ბილიკებით;
4. ხელოვნური ობიექტები - მონაცემთა ობიექტები, ჯგუფები და ანოტაცია.

განვიხილოთ თითოეული მათგანი დეტალურად.

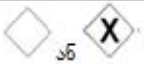
1. ობიექტთა ნაკადის ფორმირებისას შესაძლებელია ბიზნეს-პროცესის აღწერა ორ დონეზე. პირველი დონე ეს არის მეტა-მოდელი, ანუ სრული, ზოგადი ბიზნეს-პროცესი, ხოლო მეორე დონეში აღიწერება პროცესის ცალკეული ეტაპები ანუ ქვეპროცე-სები (ნახ. 1.4-1.5).

მოვლენები	გამოსახულება
ჩვეულებრივი (plain events)	
შეტყობინება (message events)	
წამზომი (timer events)	
შეცდომა (error events)	
შეწყვეტა (cancel events)	
კომპენსაცია (compensation events)	
ბიზნეს-წესები /პირობები (conditional events)	
ბმული (link events)	
კომპლექსური (multiple events)	
სიგნალი (signal events)	
შეწერება (terminate events)	

ნახ.1.4. BPMN ნოტაციის ობიექტთა ნაკადის ელემენტები

BPMN ელემენტები	გამოსახულება		
	ჩვეულებრივი	პარალელური	ციკლური
ქმედება/ოპერაცია			
ქვე-პროცესი			

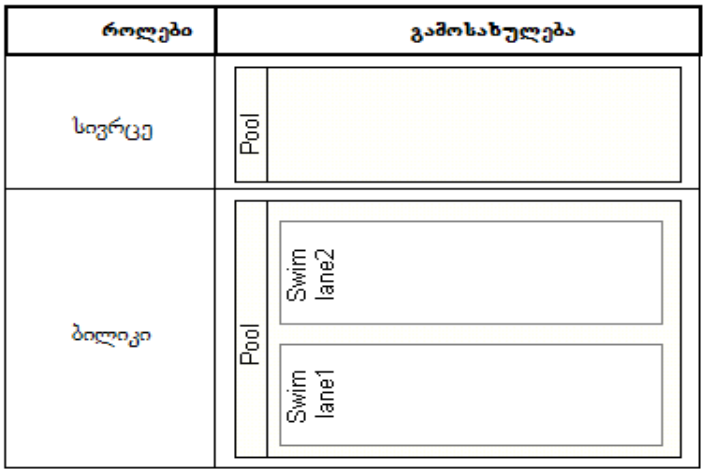
ნახ.1.5. BPMN-ნოტაციის ელემენტები

ლოგიკური ელემენტები	გამოსახულება
ლოგიკური „ან“ ოპერატორის გამორიცხვა მონაცემთა მართვისას (Data XOR)	
ლოგიკური „ან“ ოპერატორის გამორიცხვა მოვლენათა მართვისას (Event XOR)	
ლოგიკური „და“ ოპერატორი (AND)	
ლოგიკური „ან“ ოპერატორი (OR)	
ლოგიკური „რთული“ ოპერატორი (COMPLEX)	

ნახ. 1.6. BPMN ნოტაციის ლოგიკური ელემენტები

3. როლები მცოცავი ბილიკებით. იგი, გამოიყენება პროცესებისა და სისტემების დეკომპოზიციისთვის და წარმოადგენს ორგანიზაციული მთლიანობის მოდელს. მისი შემადგენელი ელემენტებია - სივრცე და ბილიკი.

როგორც წესი, მცოცავი ბილიკები გამოიყენება ქმედებების დაჯგუფებისთვის ფუნქციებისა და როლების მიხედვით. სივრცეში ხდება ქმედებათა ცალკეული მოდულური პროცესების (activities) ჩასმა სხვადასხვა ბიზნესარსებისა ან როლების აღწერისთვის, ხოლო ბილიკები მოდულური პროცესების ვირტუალური გამყოფია, ე.წ. საზღვარი ცალკეულ ქმედებათა დიაგრამებს შორის (ნახ. 1.7).



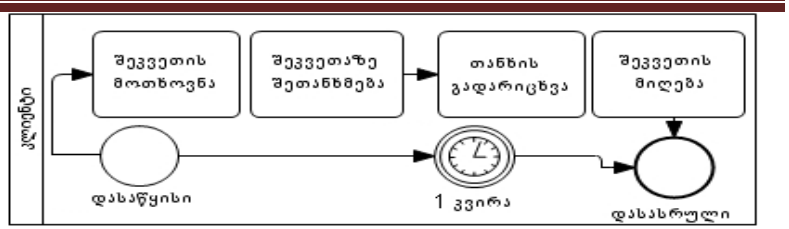
ნახ.1.7. BPMN ნოტაციის როლური ელემენტები

4. ხელოვნური ობიექტები - მონაცემთა ობიექტები, ჯგუფები და ანოტაცია.

ბიზნესპროცესების მოდელირების ნოტაცია (BPMN- Business Process Modeling Notation) საშუალებას იძლევა აიგოს როგორც სისტემის საქმიანი პროცესების ცალკეული მოდელები, ისე პროექტების მართვის დოკუმენტბრუნვის და საქმეთა წარმოების პროცესების ინტეგრალური სურათი, ანუ განზოგადებული მეტა-მოდელი [47].

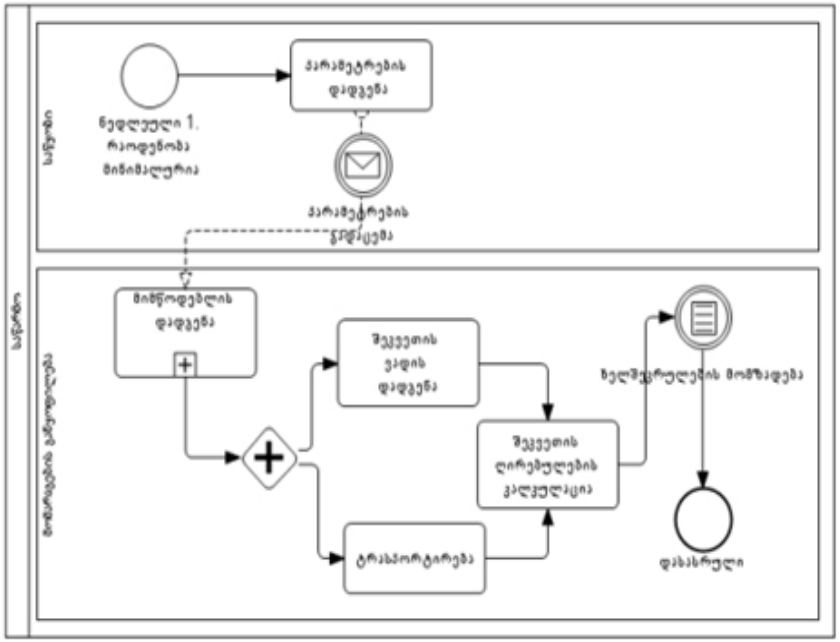
ბიზნესპროცესების მოდელირებისა და შესრულების ენები საშუალებას იძლევა გრაფიკულად აიგოს გამჭოლი ბიზნესპროცესები. არსებობს სამი ძირითადი ტიპი გამჭოლი მოდელის ქვემოდელების ფარგლებში:

1. კერძო (შიგა) ბიზნესპროცესი, რომელიც აღწერს ტექნოლოგიურ პროცესს ანუ საქმიან ნაკადს. კერძო ბიზნესპროცესის მოდელის ფრაგმენტი წარმოდგენილია 1.8 ნახაზზე.



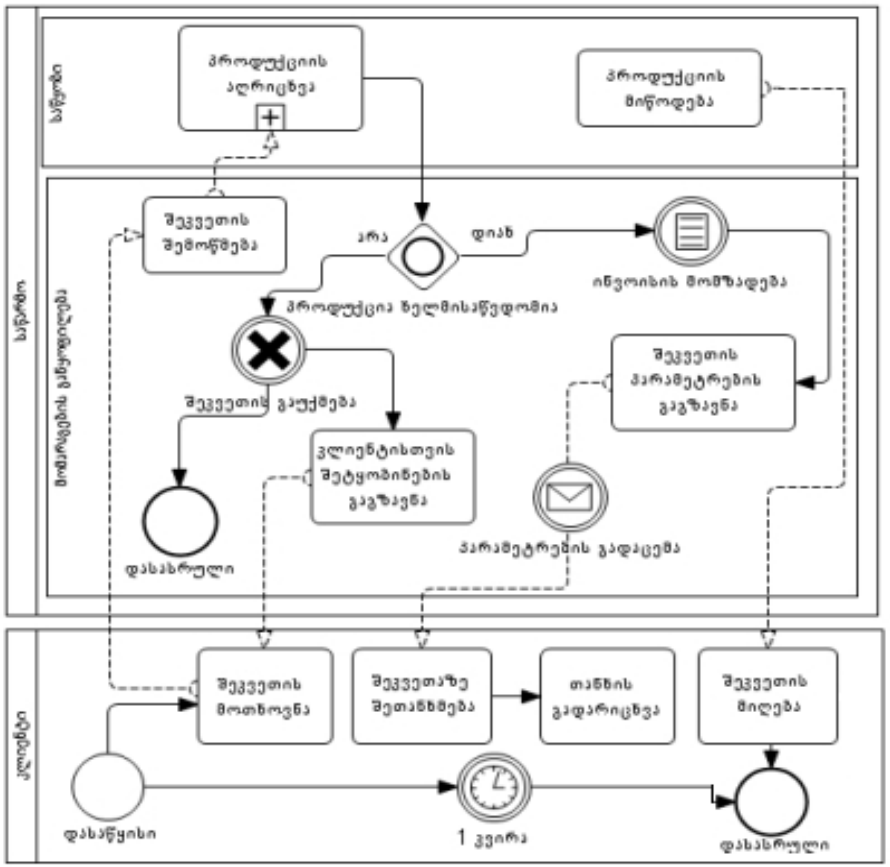
ნახ.1.8. კერძო ბიზნესპროცესის მოდელის ფრაგმენტი

2. აბსტრაქტული (ღია) ბიზნესპროცესი. იგი, აღწერს დამოკიდებულებას ორ ან მეტ კერძო პროცესს შორის ან პროცესსა და რესურსს შორის. აბსტრაქტულად ითვლება მხოლოდ ის პროცესები, რომელთა ქმედება აუცილებლად უკავშირდება კერძო ბიზნესპროცესს. ამდენად, აბსტრაქტული პროცესი ასახავს იმ შეტყობინებათა გადაცემის თანამიმდევრობას, რომლებიც ურთიერთქმედებს კონკრეტულ ბიზნესპროცესთან (ნახ.1.9).



ნახ.1.9. აბსტრაქტული ბიზნესპროცესის მოდელის ფრაგმენტი

3. ერთობლივი (გლობალური) ბიზნესპროცესი, რომელიც ასახავს ურთიერთქმედებას ორ ან მეტ ბიზნესობიექტს შორის და აერთიანებს აბსტრაქტულ ბიზნესპროცესებს. იგი წარმოადგენს ფაქტობრივად მეტა-მოდელს, რომელიც ქმნის კონკრეტული ბიზნესსტრუქტურის ერთიან სურათს (ნახ.1.10).



ნახ.1.10. ერთობლივი ბიზნესპროცესის მოდელის ფრაგმენტი

ბიზნესპროცესების მოდელირებისა და შესრულების ენებში მოდელირების ძირითად სემანტიკურ ერთეულად განიხილება ოპერაციები და შეტყობინებები, რის შედეგადაც წარმოებს დანართების სხვადასხვა ფუნქციონალური მოდულების ანუ სერვისების ურთიერთკავშირი.

ბიზნესპროცესების რეალიზაციის ენის საფუძველზე იწარმოება ორგანიზაციული პროცესების დოკუმენტაცია, ვიზუალიზაცია, მათი კომუნიკაციის მხარდაჭერა და თავსებადობა ვებ-სერვისული და სერვის-ორიენტირებული მიდგომის ფარგლებში.

ბიზნესპროცესების მოდელირების ნოტაციაში პრიორიტეტულია მოდელირების გრაფიკული ელემენტების ვიზუალური მხარე და დიაგრამების თავსებადობა.

ამ თავსებადობის საფუძველია ბიზნესპროცესების მოდელირების ენა (BPML - Business Process Modeling Language) და ბიზნესპროცესების შესრულების ენა (BPEL - Business Process Execution Language), რომელიც ბაზირებულია XML (Extensible Markup Language) ენაზე და წარმოადგენს ბიზნესპროცესების გრაფიკულად ასახვისა და მათი ურთიერთქმედების პროტოკოლების ფორმალური აღწერის ენას, რაც ბიზნესმოდელისა და საინფორმაციო მოდელის სინქრონიზაციის საშუალებას იძლევა [48].

ბიზნესპროცესების მოდელირების ნოტაციის ინსტრუმენტულ საშუალებად დღესდღეობით არსებული და განვითარებადი სისტემებია: Business Process Visual Architect, Active Modeler Avantage, ILOG JViews BPMN Modeler და ა.შ.

1.2.3. სერვის-ორიენტირებული არქიტექტურა (SOA)

სერვის-ორიენტირებული არქიტექტურა (Service Oriented Architecture) ახალი ხედვაა განაწილებული საინფორმაციო სისტემების ავტომატიზაციაში, რაც სხვადასხვა პროგრამულ დანართებში ცალკეულად დამუშავებული ავტომატიზებული ბიზნესპროცესების კომპოზიციისა და ინტეგრაციის საშუალებას იძლევა ერთ მთლიან სისტემაში. იგი, წარმოადგენს კომპონენტების ურთიერთქმედების მოდელს, რომელიც აკავშირებს დანართების სხვადასხვა ფუნქციონალურ მოდულებს (ვებ-სერვისებს) და საერთო ინტერფეისში მუშაობის საშუალებას იძლევა [15, 44, 45].

სერვის-ორიენტირებული მიდგომის არსია არსებული და მომავალი სხვადასხვა ფუნქციონალური, მასშტაბური საინფორმაციო სისტემების დანართების ურთიერთქმედება და ორკესტრირება ერთ საინფორმაციო გარემოში, ხოლო წვდომა სხვადასხვა საინფორმაციო სისტემების დანართებზე ხორციელდება ვებ-სერვისების საშუალებით. სერვის-ორიენტირებული მიდგომა, ძირითადად, საინფორმაციო ტექნოლოგიების არქიტექტურის შექმნის სტილია, რომლის იდეოლოგიით, ცალკეულად რეალიზებული სტანდარტული ბიზნესფუნქციები წარმოდგენილია ურთიერთდაკავშირებული ვებ-სერვისების სახით, რომელთა ერთობლივი გამოყენება და გამოძახება ხორციელდება კორპორაციული ან გლობალური ქსელით.

განსაზღვრული ბიზნესპროცესების შესრულებისთვის სერვისების გამოძახების თანმიმდევრობის ვიზუალური მოდელირება ხდება ბიზნესპროცესების მოდელირების ენის გამოყენებით, თანამეროვე სტანდარტით - ბიზნესპროცესების მოდელირების ნოტაცია (Business Process Modeling Notation -

BPMN), ხოლო ამ თანმიმდევრობის აღწერა ხორციელდება ბიზნესპროცესების შესრულების ენის (Business Process Execution Language-BPEL) გამოყენებით.

ბიზნესპროცესების შესრულების ენა გამოიყენება ასევე ტექნოლოგიური პროცესების ნაკადებისა (Workflow) და მონაცემთა ნაკადების (Data flow) ლოგიკური სინთეზისა და კოორდინაციის საშუალებად. ტექნიკური გამოყენების თვალსაზრისით იგი განსაზღვრავს თუ როგორ მოხდეს XML (exteblite Markuap Language) შეტყობინების გაგზავნა მოშორებულ სერვისებთან, როგორ განხორციელდეს XML მონაცემთა სტრუქტურის მართვა და მოშორებული სერვისებიდან XML შეტყობინებათა ასინქრონულად მიღება [46].

პროგრამული ტექნოლოგიების მწარმოებელი თანამედროვე, წამყვანი კომპანიები აქტიურად უჭერენ მხარს სერვის-ორიენტირებული არქიტექურის, ვებ-სერვისული ინტერფეისებისა და BPEL ენის გამოყენებას.

1.3. ბიზნესპროცესების მოდელების ინტეგრაცია

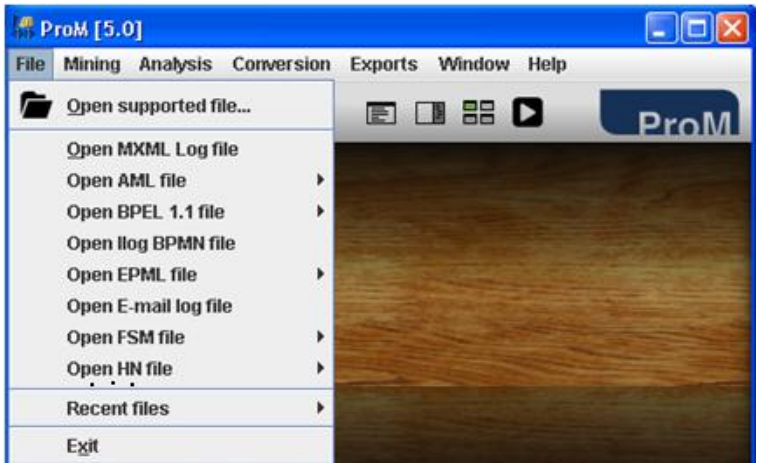
პროგრამული ინჟინერიის კლასის ტექნოლოგიები (CASE, MDA, RAD, BPMN) პროგრამული კოდის გენერაციის გარდა, XML და BPEL ენების გამოყენებით ცალსახად უჭერს მხარს სხვადასხვა ტიპის მოდელების თავსებადობის, ურთიერთტრანსფორმაციისა და დიაგრამების კონვერტაციის პროცესებს [15].

XML ენა ასრულებს მონაცემთა კოოპერაციის ერთგვარ ხიდს სხვადასხვა საინფორმაციო სისტემებში. იგი, საკმაოდ მოქნილი ენაა, რომლის შესაძლებლობაშია დამუშავდეს საკუთარი ტეგები, მონაცემთა სტრუქტურები და სქემები. XML ტექნოლოგია დამუშავებულია სტრუქტურულ მონაცემთა მართვისთვის და

იძლევა მონაცემთა განსაზღვრის, გადაცემისა და ინტერპრეტაციის საშუალებებს სხვადასხვა დანართებში, მოდელირების ენებში, მაღალი დონის პროგრამული ტექნოლოგიებში, მონაცემთა ბაზის მართვის სისტემებსა და ტექსტურ ფაილებში [34].

BPEL ენა (ბიზნესპროცესების რეალიზაციის ენა) ბაზირებულია XML ენაზე და ბიზნესპროცესების ფორმალური აღწერის, ტრანსფორმაციის, კოოპერაციისა და გენერაციის საშუალებას იძლევა. BPEL ენა ძირითადად, მუშაობს BPMN სტანდარტის მოდელებთან (Business Process Visual Architect, ActiveModelerer Avantage, Oryx-editor, ILOG BPMN Modeler).

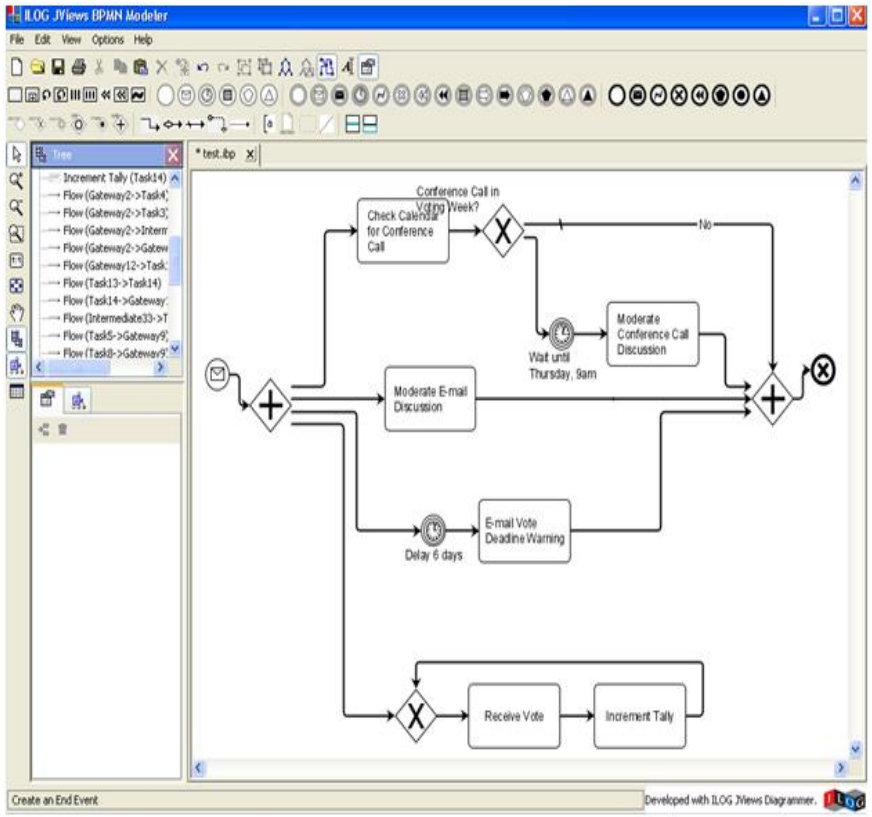
მოდელების ტრანსფორმაციისა და კონვერტაციის თვალსაზრისით ერთ-ერთი მძლავრი ინსტრუმენტია ProM რედაქტორი, რომელიც ბიზნესპროცესების სხვადასხვა ტიპის მოდელებისა და დიაგრამების, ურთიერთტრანსფორმაციის ანალიზის, კომპოზიციის, გარდაქმნის და ექსპორტის საშუალებას იძლევა (ნახ.1.11).



ნახ.1.11. ProM რედაქტორი

ProM რედაქტორი ბიზნესქმედებათა მონიტორინგის ბიზნესოპერაციების მართვის, ბიზნესპროცესების ანალიზის ერთგვარი პლატფორმაა CASE, Workflow, BPMN სტანდარტის დიაგრამების, სხვადასხვა ტიპის პეტრის ქსელის მოდელების, XML დოკუმენტების ურთიერთგარდაქმნისა და ანალიზისთვის [15].

განვიხილოთ ILOG BPMN Modeler სისტემაში აგებული BPMN მოდელის (ნახ. 1.12) კონვერტირება პეტრის ქსელში.



ნახ.1.12. BPMN მოდელი ILOG BPMN Modeler სისტემაში

ProM რედაქტორში ფუნქციით - Open ILOG BPMN File იხსნება შესაბამის ინსტრუმენტში შექმნილი BPMN მოდელი.

რედაქტორის ექსპორტის ფუნქციაში, გათვალისწინებულია კონვერტირებული დიაგრამების დოკუმენტაციის შექმნა ***.Dot ფაილში და ყოველი პროცესების ლოგირება, რომელიც ქმნის შემდეგი ტიპის MXML Log ფაილს (ნახ.1.13).

პეტრის ქსელის მიღებამდე, მოდელი საჭიროებს შუალედურ გარდაქმნას Workflow ჯგუფის ენის მოდელში (YAWL workflow), რის საფუძველზეც კონვერტირდება სხვადასხვა სპეციფიკის პეტრის ქსელში (ნახ.1.12) [15].

```
<Process>
    <ProcessInstance id="5">
        <AuditTrailEntry>
            <WorkflowModelElement>
                receive_application_3
            </WorkflowModelElement>
            <EventType>complete</EventType>
            <Timestamp>
                2008-02-29T15:20:01.050+01:00
            </Timestamp>
            <Originator>MoeW</Originator>
        </AuditTrailEntry>
        ...
    </ProcessInstance>
        ...
</Process>
```

ნახ.1.13

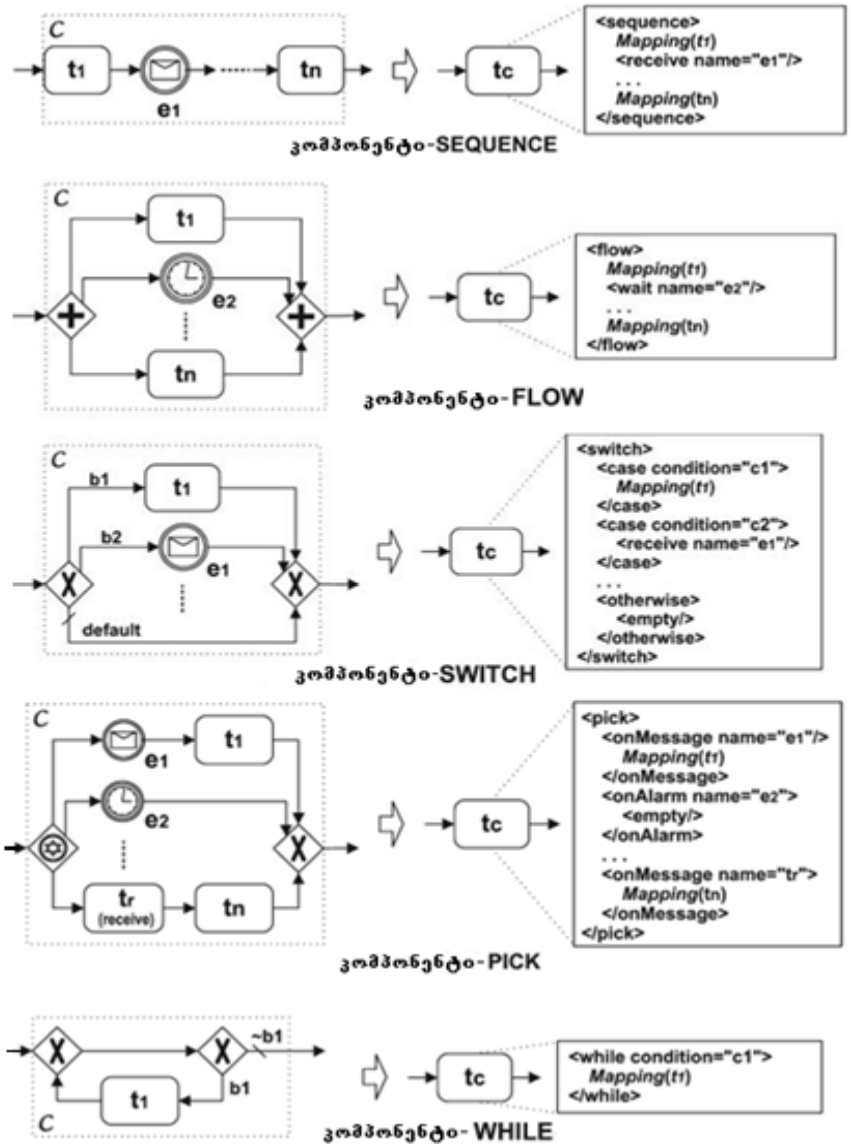
აქვე, შესაძლებელია მოდელების, მათ შორის პეტრის ქსელის მოდელის პარამეტრების მოდიფიკაცია და ანალიზი მენიუდან Analyze, ატრიბუტების, კვანძებისა და გადასვლების მიხედვით [15]. ეს საკითხი სცილდება ჩვენი თემის ფარგლებს, ხოლო ამგვარი პროცესების სიმულაციური ანალიზის ამოცანა განხილული გვექნება ფერადი პეტრის ქსელების (CPN) ინსტრუმენტის მაგალითზე [6,48,51].

BPMN მოდელების კონვერტაცია ხორციელდება XPDL (XML Processing Description Language) და BPEL ენების, როგორც სინტაკსური ანალიზატორების საფუძველზე. XPDL აღწერს ბიზნესპროცესების შინაარს, ხოლო BPEL ენა ბიზნესპროცესების ურთიერთქმედებას [73].

1.14 ნახაზზე წარმოდგენილია BPMN სტანდარტის ძირითადი ლოგიკური კომპონენტების (Sequence - მიმდევრობითობა, Flow-ნაკადი, Switch - გადამრთველი, Pick - მინიშნება, While-ციკლის ოპერატორი) BPEL კოდირების ნიმუში.

ბიზნესპროცესების აღწერას XPDL ენაზე შემდეგი სახე აქვს:

```
digraph G {
compound = true;
node [height=".2",width=".2",fontname="Arial",fontsize="8"];
        Gateway9
[label="",shapefile="attgrappa.bpmn.GatewayShape",shape="custom"];
Task4 [shape="box",label="Moderate E-mail Discussion"];
Task5 [shape="box",label="E-mail Vote Deadline Warning"]; End37}
```



ნახ.1.14. BPMN კომპონენტების BPEL კოდი

XML, XPDL, BPEL ჯგუფის ენების შექმნის საფუძველზე, თანამედროვე საინფორმაციო ტექნოლოგიებში შემოდის ბიზნესპროცესების ინტეგრაციის ძირეული კონცეფციები - ორკესტრირება (Orchestration), ქორეოგრაფია (Choreography) და ონტოლოგია (Ontology), რაც ძირითადად, განეკუთვნება სისტემების პროცეს-ორიენტირებული და სერვის-ორიენტირებული მიდგომით რეალიზაციას [1,15].

ბიზნესპროცესი სერვის-ორიენტირებული მიდგომის საფუძველზე აღიწერება ვებ-სერვისის სახით.

ორკესტრირება, პროცეს-ორიენტირებული მიდგომის თვალსაზრისით, არის სისტემის სხვადახვა სახის ბიზნესპროცესების ანუ ვებ-სერვისების კომპოზიცია, რომელთა შესრულების მიმდევრობას მართავს ბიზნესლოგიკა (მიზნობრივი ფუნქცია). ორკესტრირების აღწერის ენად გამოიყენება XPDL და BPEL ენები.

ქორეოგრაფია უზრუნველყოფს XML და WSDL ენების საშუალებით სხვადასხვა პლატფორმაზე და პროგრამულ ენაზე აღწერილი ბიზნესპროცესების (ვებ-სერვისების) აღქმადობასა და გამოყენებას.

ონტოლოგია ემსახურება ბიზნს-პროცესების მთლიანობას. იგი, ამავდროულად, სისტემის შესახებ ცოდნის ინფორმაციული მთლიანობაა და წარმოადგენს ერთგვარ კონცეპტუალურ სქემას, სისტემის ელემენტების აღწერის, ტერმინების, მათ შორის დამოკიდებულებათა და ამ დამოკიდებულებათა წესების ერთობლივი წარმოდგენით, ონტოლოგიის აღწერის ენებია RDF, RDF/XML (Resource Description Framework), OWL (Web Ontology Language) და სხვა.

1.4. ვებ–სერვისების აგება VisualStudio.NET–გარემოში: ASP.NET, Web Service

ASP.NET (Active Server Pages – აქტიური სერვერული გვერდები) – არის NET-პლატფორმის ნაწილი და ტექნოლოგია, რომელიც დინამიკურად ქმნის დოკუმენტებს Web-სერვერზე, როცა ისინი მოითხოვება HTTP-ს საშუალებით.

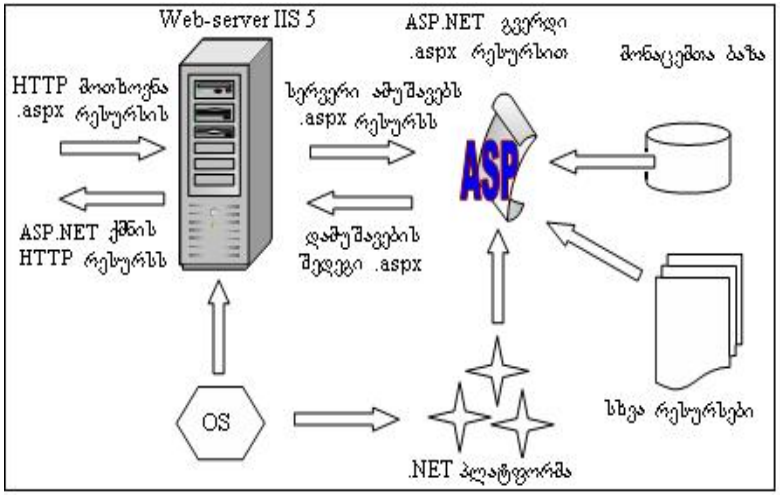
ASP.NET ტექნოლოგია ანალოგიურია PHP, ColdFusion და სხვ. ტექნოლოგიების, მაგრამ მათ შორის მნიშვნელოვანი განსხვავებაცაა. ASP.NET, როგორც მისი დასახელება გვიჩვენებს, შეიქმნა სპეციალურად NET პლატფორმასთან სრული ინტეგრაციის მიზნით, რომლის ნაწილიც ითვალისწინებს C# ენის მხარდაჭერას.

როგორც ცნობილია, Web-გვერდების დასაპროგრამებლად გამოიყენება ისეთი სცენარების ენები, როგორიცაა VBScript ან JScript. ეს სკრიპტული ენები მუშაობდა, მაგრამ ხშირად გარკვეულ პრობლემებს უქმნიდა დაპროგრამების "ნამდვილი" ენების პროგრამისტებს სხვადასხვა ადმინისტრატორული დავალებათა შესრულებისას, რაც საბოლოო ჯამში აისახებოდა სისტემის მწარმოებლურობის დაქვეითებაში.

მაღალგანვითარებული ენების გამოყენების შემთხვევაში, მაგალითად, შესაძლებელია მუშაობის პროცესის უზრუნველყოფა სრული სერვერული ობიექტური მოდელით. ASP.NET ახორციელებს მიმართვას გვერდის ყველა მმართველ ელემენტთან, როგორც ზოგადად გარემოს ობიექტებთან. სერვერის მხარესაც კი ხორციელდება წვდომა .NET–ის ყველა საჭირო კლასთან.

გვერდების მართვის ელემენტები ფუნქციონალურია და ფაქტობრივად, შესაძლებელია ყველაფრის გაკეთება, რასაც Windows-ის ფორმის კლასებთან ვაკეთებდით, რაც უფრო მოქნილს ხდის სისტემას. ამის გამო ASP.NET-ის გვერდებს, რომლებიც ქმნის

HTML შედგენილობას, ხშირად უწოდებენ Web-ფორმებს. ASP.NET გამოიყენებს ინტერნეტის ინფორმაციულ სერვერს (IIS – Internet Information Server) HTTP მოთხოვნებზე საპასუხო შინაარსის მისაწოდებლად. ASP.NET გვერდები მოთავსებულია ფაილებში .aspx გაფართოებით. მისი საბაზო არქიტექტურა მოცემულია 1.15 ნახაზზე [12].



ნახ.1.15. საბაზო არქიტექტურა

ASP.NET-ის დამუშავების დროს მისაწვდომია .NET-ის ყველა კლასი, სპეციალური კომპონენტები, შექმნილი C# ან სხვა ენებზე, მონაცემთა ბაზები და ა.შ. ფაქტობრივად, სახეზეა ყველა ის შესაძლებლობა, რომელსაც იყენებს C# დანართის აგებისას. ე.ი., C#-ის გამოყენება ASP.NET-ში ეფექტურს ხდის დანართის შესრულებას.

ASP.NET ფაილი შეიძლება შეიცავდეს ნებისმიერ ელემენტს შემდეგი სიიდან:

- ინსტრუქციის დამუშავება სერვერისთვის;
- კოდები ენებზე: C#, VB.NET, Jscript.NET ან სხვ., რომელთა მხარდაჭერა ხდება .NET პლატფორმით;
- შინაარსი ნებისმიერი ფორმით, რომელიც გენერირდება რესურსის სახით HTML–ით;
- ASP.NET -ის ჩადგმული სერვერული მართვის ელემენტები.

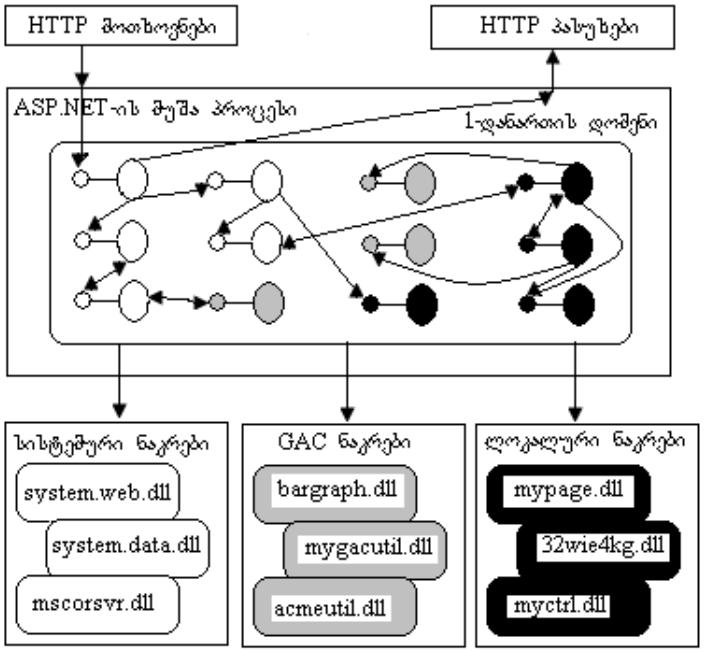
ამრიგად, შესაძლებელია ASP.NET ფაილის არსებობა, რომელიც მხოლოდ ერთი სტრიქონისგან შედგება, მაგალითად, Hello, ყოველგვარი სხვა კოდის ან ინსტრუქციის გარეშე. ამის საფუძველზე იქმნება დასაბრუნებელი HTML გვერდი, რომელშიც მოთავსებულია აღნიშნული ტექსტი.

ASP.NET-ის ბირთვია .NET-ის კლასების ერთობლიობა, რომელიც ემსახურება HTTP მოთხოვნების დამუშავებას. ზოგიერთი მათგანი ამ კლასებიდან განსაზღვრულია სისტემურ ნაკრებში (Assembly), როგორც .NET-ის საბაზო კლასების ბიბლიოთეკის ნაწილი.

ზოგი შეიძლება მოთავსებული იყოს გლობალური კეშის ნაკრებში (GAC – Global Assembly Cache), ზოგიც შეიძლება ჩაიტვირთოს ლოკალური ნაკრებიდან, რომელიც განთავსებულია დანართის ვირტუალურ კატალოგში.

ყველა კლასი, რომლებიც ემსახურება HTTP მოთხოვნებს, ჩაიტვირთება დანართის დომენში (დანართის არე – Application area) ASP.NET-ის მუშა პროცესში, და ურთიერთქმედებს ერთმანეთთან, აფორმირებს მოთხოვნებზე პასუხებს.

1.16 ნახაზი ასახავს ASP.NET-ის საბაზო არქიტექტურას დონეების მიხედვით. იგი 1.15 ნახაზის დეტალურად ასახსნელადაა შემოტანილი.



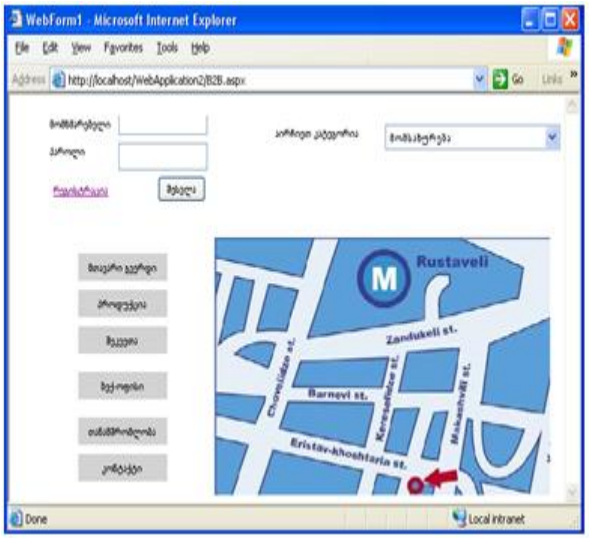
ნახ.1.16. ASP.NET საბაზო არქიტექტურა დონეების მიხედვით

ASP.NET-ის ძირითადი სიახლე მის წინამორბედებთან შედარებით ისაა, რომ აქ ყველა არსი აისახება კლასის საშუალებით, რომელიც ნაკრებიდან ჩაიტვირთება. დანართების აგება ASP.NET-ში ხდება კლასების კონსტრუირებით, რომლებიც ურთიერთმოქმედებს სხვა კლასებთან. ზოგი კლასი იწარმოება პლატფორმის საბაზო კლასებიდან, ზოგს შეუძლია ინტერფეისების რეალიზება ამ პლატფორმაში, ზოგიც ურთიერთქმედებს პლატფორმის საბაზო კლასებთან მათი მეთოდების გამოძახების გზით. ASP.NET-ის კლასიკური სინტაკსი ისევ შენარჩუნებულია,

ოღონდაც მისი კოდები, რომლებიც ინახება ნაკრების ფაილებში სერვერის მხარეს, გარდაქმნილია კლასების განსაზღვრების სახით.

განვიხილოთ პროდუქციის შეკვეთისა და შესყიდვის ინტერაქტიული ვებ-გვერდი (B2B-business-to-business, B2C-business-to-consumer მოდელი), რომლის სტრუქტურა მოიცავს: *შეკვეთის ორგანიზებისა და მიწოდების სერვისს; პრეისკურანტებს/ელექტრონულ კატალოგს; პროდუქციის მიმწოდებლებთან თანამშრომლობას; მარკეტინგის სამსახურს; პროდუქციის შეძენისთვის საკრედიტო ბარათით თანხის გადახდის სერვისს.*

1.23 და 1.243 სურათებზე ნაჩვენებია საწყისი გვერდი, რომელიც შეიცავს საიტის ნავიგაციას (ბმულები ვებ-გვერდებთან კავშირისთვის) და საიტზე შეკვეთის გაფორმების ფორმას ბმულით "**შეკვეთა**". საიტის ნავიგაციის სარეალიზაციოდ, შესაძლებელია გამოვიყენოთ როგორც HTML/Jscript კოდი, ისე სპეციალურად შექმნილი XML ფორმატის ფაილი და პროგრამული კოდი.



ნახ. 1.17. ვებ-საიტის საწყისი გვერდის ფრაგმენტი

ნახ.1.18. შეკვეთის გაფორმების ფრაგმენტი

ბმულით (Hyperlink) ვებ-გვერდების გამოსაძახებლად ვებ-ფორმის დიზაინერში გადავდივართ ბლოკში HTML, რომლის სინტაკსი შემდეგნაირია:

```
<asp:hyperlink id="HyperLink_Registr" NavigateUrl="Registracia.aspx"
style="Z-INDEX: 108; LEFT: 52px; POSITION: absolute; TOP: 194px"
runat="server" Height="16px" Width="69px" Font-Size="XX-Small">
რეგისტრაცია</asp:hyperlink>
```

ხოლო, ღილაკით (Button) ვებ-გვერდის გამოძახების შემთხვევაში, ღილაკის მეთოდში ვწერთ შემდეგ ბრძანებას:

```
private void Button_shekveta_Click(object sender, System.EventArgs e)
{
        Response.Redirect("Order.aspx");
}
```

ვებ-გვერდის ელემენტების ყველა ტეგი ვებ-ფორმის HTML ბლოკში იწყება პრეფიქსით <asp:, რაც ნიშნავს, რომ ASP .NET- ის ელემენტები განსხვადება HTML-ის ელემენტებისგან, ისინი განთავსებულია სერვერზე, ხოლო HTML-ის ჩვეულებრივი ელემენტები გამოიყენება მხოლოდ კლიენტური ვებ-გვერდების აწყობისთვის. შეკვეთის ვებ-გვერდი შეიცავს პროდუქციის შეძენისთვის შეკვეთაზე ხელშეკრულების, ანგარიშ-ფაქტურის გაფორმებისა და საკრედიტო ბარათით თანხის გადახდის სერვისებს.

1.5. სერვისორიენტირებული არქიტექტურის რეალიზაციის საშუალება Ms BizTalk Server 2010

კომპანიების საინფორმაციო ტექნოლოგიების განვითარება სულ უფრო მიმდინარეობს სერვისზე ორიენტირებული არქიტექტურის (SOA) მიმართულებით. BizTalk_Server სისტემის დანიშნულებაა სხვადასხვა ტექნოლოგიებზე, პლატფორმებზე შექმნილი განსხვავებული ბიზნეს აპლიკაციების დაკავშირება, ბიზნეს პროცესების ავტომატიზაცია და მათი მონიტორინგი [1,21]. ინტერკორპორაციულ სისტემებში BizTalk–ის გამოყენების უპირატესობები შემდეგია:

- სისტემის ელემენტების შექმნის და ბიზნესლოგიკის დამუშავების სიმარტივე;
- მონაცემთა და პროცესების ანალიზის და მონიტორინგის შესაძლებლობა;
- ხარვეზების დიაგნოსტიკის სიმარტივე;

- ბიზნესმომხმარებლის მიერ ბიზნესლოგიკის ცვლილების შესაძლებლობა;
- მონაცემთა უსაფრთხო მიღება-გადაცემა;
- სერვერების ჯგუფების შექმნა დატვირთვების გასანაწილებლად და საიმედოობის გასაზრდელად.

განვიხილოთ BizTalk-ის შედგენილობა უფრო დეტალურად, რომელიც სხვადასხვა ქვესისტემებს მოიცავს.

1.5.1. მონაცემთა მიღებისა და გადაცემის ქვესისტემა

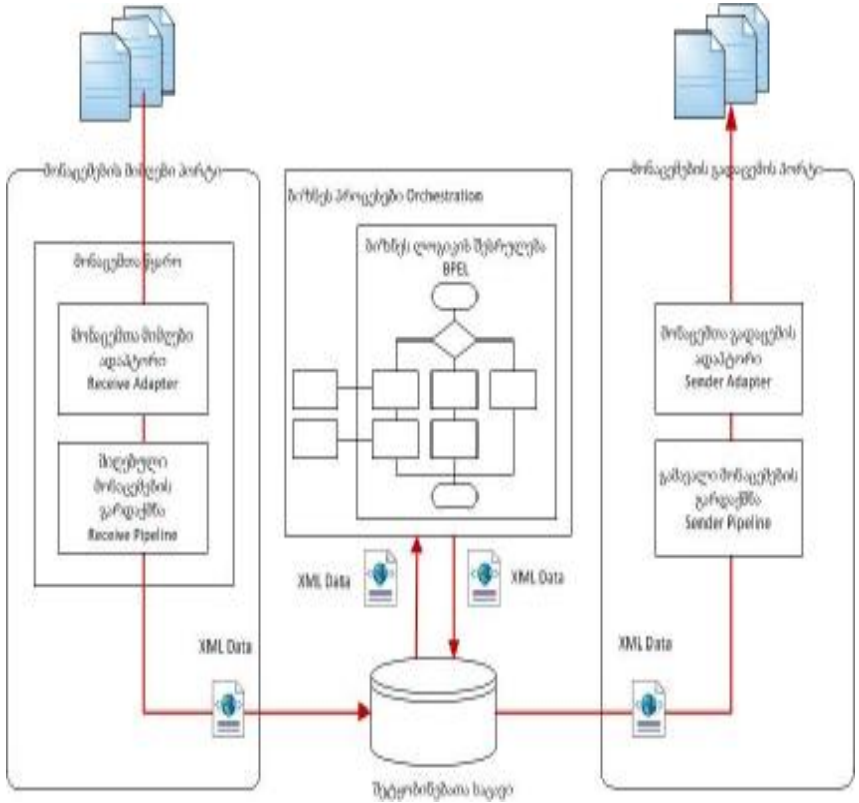
მისი დანიშნულებაა მონაცემების მიღება სხვადასხვა წყაროებიდან და მათი გადამისამართება გამავალ წყაროებში. ამ სისტემის კონფიგურაცია შესაძლებელია ადმინისტრირების საშუალებით BizTalk Administration. მიმღებ პორტზე ადაპტერის საშუალებით მონაცემების მიღება რაიმე ფიზიკური წყაროდან (ფაილი, ვებ-სერვისი, Sharepoint, ელ-ფოსტა). ადაპტერი მიღებულ ინფორმაციას გარდაქმნის XML ფორმატში. XML-ად გარდაქმნის ეტაპზე შესრულდება მონაცემების შიფრაცია, დეკოდირება, შესაბამისობაზე შემოწმება და სხვა მოქმედებები.

შემდეგ ეტაპზე ხდება XML დოკუმენტის გარდაქმნა საჭირო ფორმატში. ეს საჭიროა დამოუკიდებლად იმისგან, თუ რა სახის ინფორმაცია იყო წარმოდგენილი. რადგანაც ბიზნეს–ლოგიკას დასამუშავებლად მონაცემები გადაეცემა XML ფორმატში.

შემდგომ შეტყობინება ხვდება შეტყობინებათა ბაზაში და განისაზღვრება, დამუშავდეს ბიზნეს ლოგიკის შესაბამისად, თუ გადაიგზავნოს სხვაგან. შეტყობინებათა გაგზავნის ეტაპები შეტყობინების მიღების ანალოგიურია. სისტემის აღწერისას უნდა აღინიშნოს, რომ მისი ელემენტების შექმნა შესაძლებელია როგორც სტანდარტული ბლოკების საშუალებით, ასევე ელემენტების მოთხოვნილებათა მიხედვით (ადაპტერები, რომლებიც

არასტანდარტული პროტოკოლებით იღებს შეტყობინებას, ასევე Pipeline, რომელიც შეასრულებს, მაგალითად, მონაცემების არქივაციას მათ გაგზავნამდე. შეტყობინებების დამუშავება ნიშნავს შემავალი ნაკადების გარდაქმნას რამდენჯერმე და ახალი ნაკადების ფორმირებას.

1.19 ნახაზზე მოცემულია BizTalk სერვერის არქიტექტურა და შეტყობინებათა დამუშავების პროცესის სქემა.



ნახ.1.19. BizTalk Server-ის არქიტექტურა და შეტყობინებათა დამუშავების სქემა

1.5.2. ორკესტრაცია, წესების მექანიზმი და ხარვეზების დიაგნოსტიკა

ორკესტრაცია. შეტყობინების მიღების შემდეგ შეტყობინებათა ბაზაში იწყება მისი დამუშავება ბიზნესლოგიკის მიხედვით. იგი შეიძლება შედგებოდეს სხვადასხვა საფეხურებისგან, მაგალითად, შეტყობინებათა ტრანსფორმაცია და ახალი შეტყობინების ფორმირება. ოპერაციები შესრულდება როგორც მიმდევრობით, ისე პარალელურად, განტოტვილი ლოგიკით და ციკლების საშუალებით, ტრანზაქციების შესაძლებლობით ჩაშენებული ბიზნესლოგიკის შესასრულებლად. ბიზნეს ლოგიკის განსახორციელებლად შესაძლებელია შეიქმნას ერთი ან რამდენიმე შეტყობინება, რომლებიც ჩაიწერება შეტყობინებათა ბაზაში და შემდგომ მოხდება გადაცემა გამავალ პორტებზე ან სხვა ბიზნეს–ლოგიკაზე.

წესების მექანიზმი. ეს მექანიზმი საშუალებას იძლევა განისაზღვროს წესების კრებული, რომლებიც გამოიყენება გამავალი შეტყობინების მოდერნიზაციისა თუ სხვა მონაცემების შექმნისთვის. წესების ჩაშენება შესაძლებელია ბიზნესლოგიკაში, და შემდგომ შესაძლებელია მათი ცვლილება, პარამეტრიზაციის საშუალებით.

ხარვეზების დიაგნოსტიკა. ნებისმიერი სერვისის გაუმართავი მუშაობის დიაგნოსტორება, შეცდომების გამოვლენა და კორექტირება არის საკმაოდ შრომატევადი ამოცანა. რადგან აპლიკაციების უდიდესი ნაწილი პროგრამული კოდის წერის გარეშე იქმნება, შესაბამისად Visual Studio-ს საშუალებით მათი გამართვა შეუძლებელია.

ამ პრობლემის გადასაჭრელად BizTalk-ში შედის ინსტრუმენტები, რომელთა საშუალებითაც შეიძლება დადგენა, თუ

როგორ მუშაობს BizTalk, რა შეტყობინება შემოვიდა, როგორ დამუშავდა, რომელი კომპონენტები იქნა ამოქმედებული.

BizTalk Administration უტილიტის საშუალებით შესაძლებელია დადგინდეს შეტყობინება და ელემენტთა ეგზემპლარები, რომელთა დამუშავებისას წარმოიქმნა შეცდომა. ასეთ შეტყობინებათა დამუშავება შეჩერდება და წყდება საერთოდ. ასევე შესაძლებელია ამ უტილიტის საშუალებით მიეთითოს, თუ რა ეტაპებზე მოხდეს შეტყობინებების მონიტორინგი და რამდენად დეტალურად. ამ უტილიტის საშუალებით შესაძლებელია მოინახოს რა შეტყობინება საიდან იქნა მიღებული, რომელი კომპონენტის მიერ იყო დამუშვებული და რა შეტყობინებები შეიქმნა, ასევე შესაძლებელია შეტყობინების შიგთავსის ნახვა.

1.5.3. ბიზნესაქტიურობის მონიტორინგი

Business Activity Monitoring (BAM) - ბიზნესმომხმარებელს სჭირდება ეფექტურობის კოეფიციენტის განსაზღვრა (Key Performance Indicators (KPIs), რომლითაც იზომება ბიზნეს–მიზნების შესრულების დონე, აგრეთვე სხვა ბიზნესპროცესების მაჩვენებლები. ეს ქვესისტემა საშუალებას იძლევა შეაგროვოს სხვადასხვა ინფორმაცია ბიზნესპროცესზე და შექმნას მონაცემთა კუბები ბიზნესმომხმარებლის მხრიდან გასაანალიზებლად. მონაცემების ფორმირება შესაძლებელია როგორც რეალურ დროში, ასევე განსაზღვრული მომენტისთვის. Analysis Service–ს შეუძლია წარმოადგინოს მონაცემები სხვდასხვა სახით, გაფილტროს, დააჯგუფოს როგორც საჭირო იქნება პროცესების შეფასებისთვის, შესაბამის გადაწყვეტილებათა მისაღებად.

BAM შედგება 3 კომპონენტისგან: აგრეგაციის, აქტიურობის ძებნის და შეტყობინებების სერვისის. აგრეგაციის საშუალებით შესაძლებელია ბიზნესპროცესების პარამეტრების გაზომვა

(მაგალითად, თვის განმავლობაში ფილიალების მიხედვით გაყიდვების რაოდენობა) და მომხმარებლებისთვის სხვადასხვა კრიტერიუმების შესაბამისად გრაფიკული ასახვა. აქტიურობათა ძებნის საშუალებით შესაძლებელია დადგინდეს აღიძრა თუ არა სისტემაში გარკვეული ტიპის პროცესი (მაგალითად, განსაზღვრულმა კლიენტმა გააკეთა თუ არა შეკვეთა, მოხდა თუ არა განსაზღვრული ოდენობის საქონლის შეკვეთა და სხვა.). შეტყობინებათა სერვისით კი შესაძლებელია მომხმარებელს მიუვიდეს შეტყობინება, თუ სრულდება ესა თუ ის პირობა, რომლის მონიტორინგიც ხდება, აქტიურობის ძებნის თუ აგრეგაციების საშუალებით.

ამრიგად, Ms_BizTalk Server არის სტრატეგიული ინსტრუმენ-ტი, რომელიც გამოყენებადს ხდის იმ ინფრაქსტრუქტურას, რომელიც მომარაგებულია ინტერნეტის მიერ ელექტრონული ბიზნესაპლიკაციებისთვის ადვილი ქსელურობის, მომხმარებ-ლების გაზრდილი მომსახურებისთვის, მიწოდების ჯაჭვის მართვისთვის, პროდუქტიული კომუნიკაციისათვის და სხვა.

ინფორმაციული ტექნოლოგიები სრულყოფს სწრაფქმედებას, არსებულ ინფორმაციაზე წვდომას. მაგრამ იმისათვის, რომ შეიქმნას ახალი ინფორმაცია და გამოვიყენოთ არსებული ინფორმაცია უფრო ახალი და მეტად ინოვაციური გზით, საჭიროა შესაბამისი ინსტრუმენტი, რომელსაც შეეძლება ინფორმაციის შეფასება აქამდე გამოუკვლეველი გზებით.

MsBizTalk Server არის ერთ-ერთი ასეთი ინსტრუმენტი. შეცდომა იქნება, თუ მას მარტივ ტექნოლოგიად განვიხილავთ.

ქვემოთ ახსნილია თუ როგორ ითავსებს ეს ახალი პროგრამული უზრუნველყოფა იმ უპირატესობებს, რომელსაც მცირე და საშუალო საწარმოები (SME) აღწევს, მისი ეფექტური გამოყენებით.

ბიზნესმენისთვის, Web-სერვისის დეველოპერისთვის, ბიზნესანალიტიკოსისთვის, ტექნოლოგიის მგეგმავისთვის, პროექტების მენეჯერისთვის, აპლიკაციის დიზაინერისა და შემსრულებლისთვის MsBizTalk Server მეტად გამოსაყენებელი პროგრამული უზრუნველყოფაა. იგი საშუალებას იძლევა მისი ინფრასტრუქტურით აიგოს წარმატებული ელექტრონული საწარმოების კავშირი.

ეს ახალი თაობის პროგრამული უზრუნველყოფა მთლიანად იყენებს გაფართოებადი მარკირების ენის (XML) შესაძლებლობებს. ინტერნეტის საშუალებით ინფორმაციისა და მონაცემების გადაცემა შესაძლებელია ძალიან გართულდეს, განსაკუთრებით მაშინ, როდესაც კომპანიების აპლიკაციები დაწერილია სპეციალურ უნიკალურ ფორმატში, რომელთა გადაცემაც რთულია. ბიზნესმონაცემების და პროცესების აღსაწერად ერთი საერთო "ტექნიკური ლექსიკონის" არარსებობა, აუცილებელს ხდის ისეთი ენის გამოყენებას, მაგალითად XML-ისა, რომელიც ზუსტად განსაზღვრავს ჩვეულებრივ კოდს/ ყველა იმ მონაცემის მნიშვნელობას, რაც ფართო ქსელში გადაიცემა.

კორპორაციები, ორგანიზაციები, აპლიკაციები და ინდივიდუალური პირები შესაძლებელია ერთმანეთს გაცილებით უფრო ეფექტურად დაუკავშირდნენ, თუ ისინი შეთანხმდებიან ინფორმაციის მნიშვნელობის სტრუქტურაზე. XML სპეციალურად შექმნილია იმისათვის, რომ გაადვილოს ასეთი კომუნიკაცია, MsBizTalk Server კი უზრუნველყოფს მტკიცე საფუძველს, რომლისთვისაც XML არის სტრატეგიული ხელსაწყო. შიძლება აღინიშნოს, რომ MsBizTalk Server-ის დანერგვისათვის XML-ის ცოდნა არ არის საჭირო. საჭიროა HTML-ის და რელაციური ბაზების მცირეოდენი ცოდნა, ასევე მცირე მონდომება იმისათვის, რომ გასაგები იყოს პროდუქტის შესაძლებლობები მონაცემთა

მარტივი ინტეგრაციისათვის, რესურსების და ცოდნის მართვისათვის.

კომპანია მაიკროსოფტმა შექმნა MsBizTalk Server საწარმოო აპლიკაცია, რომელიც ინტეგრირებულია .NET სერვერ ოჯახში (www.microsoft.com) და ქმნის ხელსაწყოთა და სერვისების კომპლექტს, როგორიცაა მართვა, ადმინისტრაცია, დაგეგმვა და მონიტორინგი. იგი მოიცავს შემდეგს:

- BizTalk Server Orchestration Designer: შექმნის ხელსაწყო, რომელიც საშუალებას აძლევს ბიზნესანალიტიკოსს, დეველოპერსა და IT პროფესიონალს, იმოქმედოს ჩვეულებრივ პლატფორმაზე;
- BizTalk Editor: ქმნის XML დოკუმენტს და ცვლილებები შეაქვს მასში;
- BizTalk Mapper: გარდაქმნის ერთ XML დოკუმენტს სხვა XML სქემად. (სქემა არის განსხვავებული გზა, რომლითაც ხდება XML დოკუმენტების სტრუქტურიზაცია და მონაცემთა გადაცემის განსაზღვრის გაადვილება);
- BizTalk Messaging Manger: მინიმალური დაპროგრამების საშუალებით შესაძლებელს ხდის ინფორმაციის და აპლიკაციების გადაცემას;
- BizTalk Framework: მისი საშუალებით ხდება მიღებული და გაცემული მონაცემების მარშრუტიზაცია, მონიტორინგი და ანალიზი.
- BizTalk Administration Tool: მისი საშუალებით ხდება მომხმარებლის სპეციფიკური ინფორმაციის ენდ-ტო-ენდ ინტეგრაცია და ადვილი ონლაინ ანალიტიკური პროცესინგი.
- Pipeline Editor: ახდენს დოკუმენტის პროცესებზე დაკვირვებას დასაწყისიდან მის დასრულებამდე.

MsBizTalk Server-ს გააჩნია სტანდარტული ინტერნეტ ტექნოლოგიების მხარდაჭერა, მაგალითად EDI (ელექტრონული მონაცემების ურთიერთმოქმედება), მრავალი გადაცემები და პროტოკოლები (HTTP, SMTP), ჩვეულებრივი და კერძო ფაილების ფორმატები. BizTalk Server-ის გამოყენებით შესაძლებელია:

• მონაცემთა გაცვლის გამარტივება: (XML-ის არცოდნის მიუხედავად) შესაძლებელია შთამბეჭდავი დოკუმენტების შექმნა და მათი მოდიფიკაცია ნებისმიერი ტექსტური რედაქტორის გამოყენებით;

• დოკუმენტების გადათარგმნა, გაფილტვრა და მარშრუტიზაცია მისი შემცველობის გათვალისწინებით;

• ონლაინური ტრანზაქციების ადვილად გამოყენება, ხმოვანი და ვიდეო ნაკადების გადაცემის საშუალებით;

• მბეჭდავი მომხმარებლების ინტერფეისის ინტეგრაცია ვებ ბრაუზერებში და მონაცემთა ბაზების საშუალებით შესყიდვებისა და შეკვეთის პროცესების სისტემების განვითარება;

• აპლიკაციების შექმნა, რომელთაც შეუძლია ორ ან მეტ ჰეტეროგენულ მონაცემთა ბაზასთან შუამავლობა და ინფორმაციის ჭკვიანურად გამოყენება, რომელიც არის გამიზნული ინდივიდუალური მომხმარებლისათვის;

• წერტილოვანი დაკვირვება მონაცემებზე, კლიენტებს შორის სანდოობის ჩამოყალიბებისათვის;

• დაცული კომუნიკაციის აგება, რომელიც უზრუნველყოფს საზოგადო გასაღების დაშიფვრას, ციფრულ ხელმოწერებს და დაშიფვრას მრავალგამიზნული ინტერნეტ შეტყობინების გაფართოებისათვის (S/MIME), გაუმჯობესებულ ბმულებს კლიენტებთან, მიმწოდებლებთან და პარტნიორებთან.

შესაჯამებლად შეგვიძლია ვთქვათ რომ, MsBizTalk Server არის ტექნოლოგიური გადაწყვეტა, რომელიც აუმჯობესებს ეფექტურობას და ახდენს ბიზნესიდეებისა და სტრატეგიების განახლებას. იგი აძლიერებს ელექტრონული საწარმოების თვისებებს, ოპერაციული სისტემისა და პროგრამირების ენის დამოუკიდებლობას.

1.6. ბიზნესპროცესების რეალიზაციის ინსტრუმენტული საშუალებანი: JavaNetBeans, BPEL

Web-დანართების სარეალიზაციოდ, დღესდღეობით აქტიურ გამოყენებაშია Java (Java NetBeans) და Microsoft .Net პლატფორმები [15].

Java NetBeans სისტემა Sun Microsoft Systems კორპორაციისა და NetBeans გაერთიანების მიერ შექმნილი Java ტექნოლოგიის ავტომატიზებული სისტემების დამუშავების ინტეგრირებული გარემოა (IDE), რომელიც არის მრავალფუნქციონალური დანართების ერთობილიობა და უზრუნველყოფს Java Platform Standard Edition (Java SE), Java Platform Enterprise Edition (Java EE) და Java Platform Micro Edition (Java ME) პლატფორმების კომპლექსურ მხარდაჭერას [33, 34].

Java NetBeans შეიცავს პროგრამული ინსტრუმენტების ფართო სპექტრს, მათ შორის აღსანიშნავია ვიზუალური დაპროგრამების, სერვისორიენტირებული არქიტექტურის დანართების (XML, BPEL), პირდაპირი და რევერსიული დაპროექტებისთვის (BPD, BPMN, UML) მოდელური არქიტექტურის (Model-driven architecture MDA) ინსტრუმენტული საშუალებები.

პროგრამული ტექნოლოგიების მწარმოებელი თანამედროვე წამყვანი კომპანიები აქტიურად უჭერენ მხარს სერვისზე ორიენტირებული არქიტექურის, Web-სერვისული ინტერფეისებისა BPMN სტანდარტისა და BPEL ენის გამოყენებას. ამ თვალსაზრისით, Java NetBeans სისტემა ერთ-ერთი მოქნილი ინსტრუმენტია.

ბიზნესპროცესების შესრულების ენა, BPMN სტანდარტის მიხედვით აგებული მოდელების პროგრამული კოდში გენერაციის საშუალებას იძლევა. იგი გამოიყენება, როგორც ვებ-სერვისული

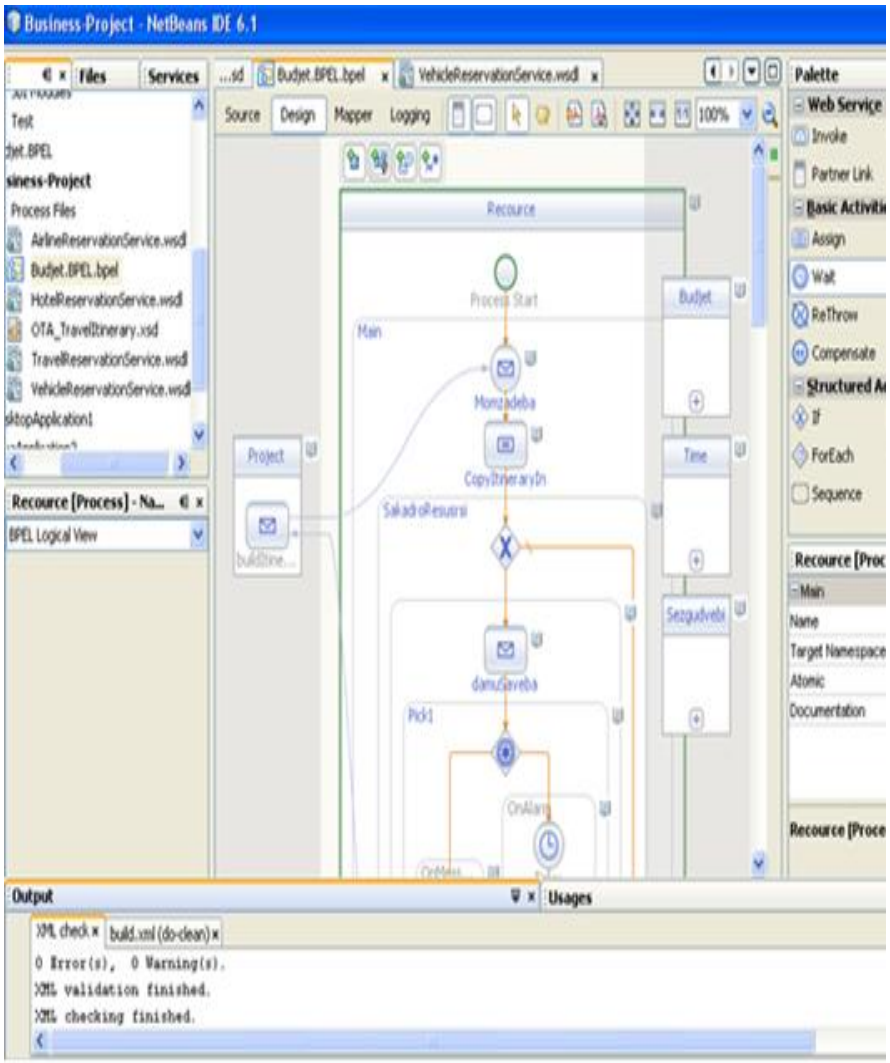
ფორმით რეალიზებული სხვადახვა ბიზნესპროცესების გამოძახების თანამიმდევრობის აღწერისთვის, ისე სისტემის ტექნოლოგიური პროცესების ნაკადებისა (WorkFlow) და მონაცემთა ნაკადების (DataFlow) ლოგიკური სინთეზისა და კოორდინაციისთვის [34].

ტექნიკური გამოყენების თვალსაზრისით, იგი განსაზღვრავს როგორ მოხდეს XML შეტყობინების გაგზავნა მოშორებულ სერვისებთან, როგორ განხორციელდეს XML მონაცემთა სტრუქტურის მართვა და მოშორებული სერვისებიდან XML შეტყობინებათა ასინქრონულად მიღება (ნახ.1.20).

Microsoft .Net პლატფორმა. Web-სერვისებისა და Web-დანართების შექმნისა და განთავსების მოქნილი საშუალებაა Microsoft .NET Framework - პლატფორმა, რომელიც ბაზირებულია მაღალმწარმოებლური, სხვადასვა პროგრამული ენის გამოყენების სტანდარტზე. მისი ერთ-ერთი ღირებულებაა არსებული Web-დანართებისა და Web-სერვისების ინტეგრაციის შესაძლებლობა ახალ პროექტში.

.NET Framework შედგება სამი ძირითადი ნაწილისგან - საერთო ენობრივი გამოყენების გარემო (common language runtime), კლასების უნიფიცირებული ბიბლიოთეკა და ASP კომპონენტური ვერსია - ASP.NET [11,12].

ASP.NET (Active Server Pages - აქტიური სერვერული გვერდები) - .NET ტექნოლოგიის ნაწილია, რომელიც გამოიყენება მასშტაბური კლიენტ-სერვერული Web-დანართების ასაგებად. ინტერაქტიული Web-საიტის ადვილად შესაქმნელად იგი შეიცავს მზა მართვის ელემენტების სიმრავლეს და შესაძლებლობას იძლევა შეიქმნას დინამიკური HTML გვერდები.



ნახ.1.20. Java NetBeans სისტემის BPEL ვიზუალური რედაქტორი

ASP.NET-ის დამუშავების დროს მისაწვდომია .NET-ის ყველა კლასი, სპეციალური კომპონენტები, შექმნილი C# ან სხვა ენებზე, მონაცემთა ბაზები და ა.შ. ფაქტობრივად, სახეზეა ყველა ის შესაძლებლობა, რომელსაც იყენებს C# დანართის აგებისას. C#- ის გამოყენება ASP.NET- ში ეფექტურს ხდის დანართის შესრულებას.

ASP.NET ფაილი შეიძლება შეიცავდეს ინსტრუქციების დამუშავებას სერვერისთვის, C#, VB.NET, Jscript.NET ან სხვა პროგრამული ენების კოდებს, რომელთა მხარდაჭერა ხდება .NET პლატფორმით, ნებისმიერი ფორმის შინაარსს, რომელიც გენერირდება რესურსის სახით HTML-ით, ASP.NET - ის ჩადგმული სერვერული მართვის ელემენტებს და ა.შ.

1.7. კორპორაციათა ბიზნესპროცესების ინტეგრაციის ამოცანა სერვისორიენტირებული სისტემების ასაგებად

ამოცანა მდგომარეობს ისეთი სისტემის შექმნაში, რომელიც უზრუნველყოფს არსებული სისტემების ინტეგრაციას, როგორც სხვადასხვა ორაგნიზაციებს შორის, ასევე ორგანიზაციის შიგნით, რომელიც უზრუნველყოფს სხვადასხვა ფორმატის მონაცემების მიღება-გადაცემა-დამუშავებას, მაღალ წარმადობას და საიმედოობას.

აპლიკაციების ინტეგრაციას ძირითადად აქვს სამი დანიშნულება:

- მონაცემთა ინტეგრაცია, რაც უზრუნველყოფს მონაცემების იდენტურობას სისტემებს შორის;
- აპლიკაციათა მიმწოდებლებზე დამოუკიდებლობა. უზრუნველყოფს, რომ აპლიკაციის ცვლილების შემთხვევაში, ბიზნესპროცესი და ბიზნეს წესების ხელახალი შექმნა არ იყოს საჭირო;

- ფასადის/ინტერფეისის შექმნა, რაც უზრუნველყოფს აპლიკაციებთან ერთიერთობის ერთიანი ინტერფესის შექმნას, რომელიც საშუალებას იძლევა აპლიკაციებთან კომუნიკაცია შესრულდეს მათი შიგა სტრუქტურების შესწავლის გარეშე.

დისერტაციის მიზანია კორპორაციული და ინტერკორპორაციული ბიზნესპროცესების მართვის ვებ-აპლიკაციების დაპროექტება და რეალიზაცია ელექტრონული სისტემების ინტეგრაციის შესაძლებლობით და სერვის-ორიენტირებული არქიტექტურით.

მიზნის მისაღწევად ნაშრომში განიხილება შემდეგი ძირითადი ამოცანები:

- არსებული თანამედროვე ინტეგრაციის სისტემების ანალიზი და შესაბამისი ინფორმაციული ტექნოლოგიების კლასიფიკაცია, ობიექტ-, პროცეს- და სერვისორიენტირებული დაპროექტების პრინციპებით;

- შიგაკორპორაციული და კორპორაციათაშორისი ბიზნეს-პროცესების ტრადიციული და სერვისორიენტირებული მოდელების აგება სისტემური ანალიზის საფუძველზე, BPMN და UML ტექნოლოგიების ბაზაზე. მათი შედარებითი ანალიზი;

- Web-აპლიკაციების დასაპროექტებლად ორგანიზაციის სერვისული ბიზნესპროცესების ობიექტორიენტირებული მოდელირება კლასების, ობიექტების და Web-მეთოდების ფორმალიზაციის საფუძველზე, პოლიმორფიზმის, მემკვიდრეობი–თობის და ინტერფეისული თვისებების გათვალისწინებით;

- სერვისორიენტირებული კორპორაციული Web-აპლიკაციების მონაცემთა განაწილებული ბაზების სტრუქტურების დასა–პროექტებლად ობიექტ-როლური მოდელების (ORM) აგება და კვლევა რევერსიული CASE ტექნოლოგიების გამოყენებით;

- სერვისორიენტირებული არქიტექტურის კორპორაციული სისტემის ბიზნესპროცესების უნიფიცირებული მოდელების ასახვის (BPMN -> Activity-D -> PetNet) ალგორითმების შემუშავება სტოქასტური, დროითი პეტრის ქსელის გრაფებში, მათი პროცესების შესრულების ეფექტურობის შეფასების და სერვისული უზრუნველყოფის შემდგომი სრულყოფის გადაწყვეტილების მიღების მიზნით;

- სერვის-ორიენტირებული არქიტექტურის ინტერკორპორა–ციული სისტემის ინფორმაციული ნაკადების გაცვლის სერვისული ბიზნესპროცესების იმიტაციური მოდელის აგება და მისი ფუნქციონირების დროითი მახასიათებლების კვლევა ფერადი პეტრის ქსელების (CPN) გამოყენებით;

- პროექტის შედეგების საფუძველზე ესპერიმენტული პროგრამული სისტემის რეალიზაცია Ms Visual Studio .NET პლატფორმაზე, ASP.NET, ADO.NET, MsSQL_Server, C#.NET, Natural ORM Architect და BizTalk Server პროგრამული პაკეტების გამოყენებით.

II თავი: კორპორაციული ბიზნესპროცესების მოდელირება Web-აპლიკაციებისთვის სერვისორიენტირებული არქიტექტურით

2.1. ინტერკორპორაციული სისტემის საინფორმაციო ბიზნესპროცესების აღწერის BPMN დიაგრამები ტრადიციული და Web-სერვისული მეთოდებით

ინტერკორპორაციული სისტემის მაგალითისათვის განვიხილავთ ფინანსთა სამინისტროს შემოსავლების სამსახურისა და საფინანსო ბანკების ერთიან ელექტრონულ სისტემას [2].

ბანკსა და შემოსავლების სამსახურს შორის ინფრომაციის გაცვლა ხდება პაკეტებით. ეს პაკეტები აგებულია XML-ფაილების საფუძველზე, რაც მნიშვნელოვნად ეფექტურს ხდის ინტერნეტის გამოყენებას ამ სფეროში

მაგალითად, შემოსავლების სამსახური ბანკში აგზავნის გარკვეული დანიშნულების საინფორმაციო შეტყობინების XML-ფაილს. ბანკის მხარეს არის რეალიზებული Web-სერვისი AcceptMessageFromMOF. ეს XML შეიცავს ინფორმაციას ყადაღების დადების/გაწვევის, ინკასოების დადების/ ცვლილების/გაწვევის, კლიენტების რეგისტრაციაზე ან დახურვაზე დასტურის შეტყობინებებს (ცხრ.2.1).

ბანკიც თავის მხრივ აგზავნის ერთ XML-ს რომელიც შეიცავს ინფორმაციას ახალი კლიენტების რეგისტრაციის, კლიენტების დახურვის დასტურის, ინკასოს თანხების, კლიენტების ტიპების ცვლილების შესახებ.

MOF სერვისის მეთოდების გამოძახებით ბანკი აგროვებს ინფრომაციას დროებით ცხრილებში და შემდგომ იძახებს შემოსავლების სამსახურის Web-სერვისს AcceptMessageFromBank, რომელსაც პარამეტრად გადაეცემა ბანკიდან გაგზავნილი ინფორმაცია.

Web-სერვისი	Web-მეთოდი	ცხრ.2.1
MOF	**AcceptMessageFromMOF** - ბანკის სერვისი, რომელსაც იძახებს შემოსავლების სამსახური. გამოიყენება ბანკებისთვის ინფორმაციის მისაწოდებლად. მონაცემები გადაეცემა XML-ის სახით, XML-შეიცავს მონაცემებს ახალი/შეცვლილი/გასაწვევი ინკასოების, ახალი/გამოსათხოვი ყადაღების, დასტურს კლიენტის გახსნის შესახებ.	
	AcceptMessageFromBank - შემოსავლების სამსახურის სერვისი, რომლითაც ბანკიდან მიეწოდება მონაცემები შემოსავლების სამსახურს	
	AddClient - აგზავნის შემოსავლების სამსახურში შეტყობინებას ახალი კლიენტის ბანკში დარეგისტრირების შესახებ. ინფორმაცია იწერება დროებით XML-ში	
	CloseClient - აგზავნის შემოსავლების სამსახურში შეტყობინებას კლიენტის და მისი ყველა ანგარიშის დახურვის შესახებ	
	CloseClientResponse - ტექნიკური შეტყობინება შემოსავლების სამსახურიდან კლიენტის დახურვის შეტყობინებაზე	
	ClientTypeChange - აგზავნის შემოსავლების სამსახურში შეტყობინებას კლიენტის ტიპის შეცვლის შესახებ (ანუ თუ კლიენტი გახდება გადამხდელი/მეწარმე)	
	IsClientHonestPayer - შემოსავლების სამსახურიდან მიღებული XML-იდან, ამოიღება დასტური კლიენტის გახსნის შესახებ. (თუ მასზე შემოსავლების სამსახურში არ არის რეგისტრირებული რაიმე დავალიანება ყადაღა/ინკასო)	
	GetOrderRecalls- იძახებს შემოსავლების სამსახურიდან მიღებული ინკასოების გაწვევების ან ცვლილებების შესახებ მონაცემებს (კლიენტის იდენტიფიკატორებს და ინკასოს	

	რედაქტირებულ თანხას, რა თანხის გადარიცხვაც უნდა მოხდეს კლიენტის ანგარიშებიდან. შესაძლოა ვალდებულების თანხა განულდეს, რაც ნიშნავს, რომ კლიენტს მოეხსნა ინკასო)
	RequestAmountFromMOF- სისტემა ამ სერვისის საშუალებით ეკითხება შემოსავლების სამსახურს, თუ რა თანხა უნდა ჩამოჭრას კლიენტს. უგზავნის XML კლიენტების იდენტიფიკატორებით
	ReceiveApprovedAmount - შემოსავლების სამსახურიდან მიღებული ინფორმაციიდან (XML), რომელიც დაბრუნდა GetXML მეთოდით, კლიენტის თანხების დამუშავება
Client	**AddClient**- გადაეცემა ახალი კლიენტის მონაცემები, ამატებს კლიენტს ბაზაში და აბრუნებს დამატებული კლიენტის იდენტიფიკატორს
	ChangeClientStatus - გადაეცემა კლიენტის იდენტიფიკატორი და კლიენტის სტატუსი გახდება „აქტიური“, „დახურული“
	ChangeType- გადაეცემა კლიენტის იდენტიფიკატორი და კლიენტის ტიპი ხდება „ფიზიკური პირი“ ან „იურიდიული ტიპი“
Accounts	**CloseAccount**- გადაეცემა ანგარიშის იდენტიფიკატორი და ხდება მისის სტატუსის შეცვლა „დახურულზე“
	CheckSaldoAvailable- გადაეცემა ანგარიშის იდენტიფიკატორი და აბრუნებს ინფორმაციას თუ რა თანხაა ამ ანგარიშზე
	UnblockAccounts - გადაეცემა კლიენტის ინდენტიფიკატორი და ახდენს მისი ანგარიშების სტატუსის ცვლილებას „აქტიურზე“. კლიენტი ჩვეულებრივად შეძლებს ანგარიშების გამოყენებას
	BlockAccounts - გადაეცემა კლიენტის იდენტიფიკატორი და

	სისტემა ასრულებს ამ კლიენტის ანგარიშების სტატუსის გადაყვანას „მხოლოდ ბიუჯეტურში“, რის შემდეგაც კლიენტი ვეღარ გამოიყენებს ამ ანგარიშებს, გარდა თანხის ბიუჯეტში გადასარიცხად
Transaction	**AddTransaction** - ახალი ტრანზაქციის დამატება სტატუსით „დაუდასტურებული“
	ApproveTransaction - გადაეცემა ტრანზაქციის იდენტიფიკატორი და ახდენს მოცემული ტრანზაქციის სტატუსის გადაყვანას „დადასტურებული“
	MakeTransaction - გადაეცემა ტრანზაქციის სტატუსი და ახდენს მოცემული ტრანზაქციის სტატუსის გადაყვანას „დასრულებული“-ში
Cards	**UnblockCard**- პლასტიკური ბარათის დაბლოკვა ბარათების საპროცესინგო ცენტრში (ბარათზე შეიზღუდება ოპერაციების წარმოება: გადარიცხვა, თანხის გატანა და ა.შ.)
	BlockCard - პლასტიკურ ბარათზე აღდგება ოპერაციების წარმოების შესაძლებლობა

განვიხილოთ აღნიშნული ბიზნესპროცესების (კლიენტების რეგისტრაცია, ინკასაცია, ყადაღის დადება/მოხსნა და სხვ. პროცესები) მოდელირება BPMN ინსტრუმენტის საფუძველზე, ტრადიციული მიდგომით და Web-სერვისების გამოყენებით.

აღნიშნული ტრადიციული და Web–სერვისებზე ორიენტირებული BPMN–სქემები მოცემულია 2.1–:2.5 ნახაზებზე (a-ტრადიციული ფორმატი , b-სერვისორიენტირებული).

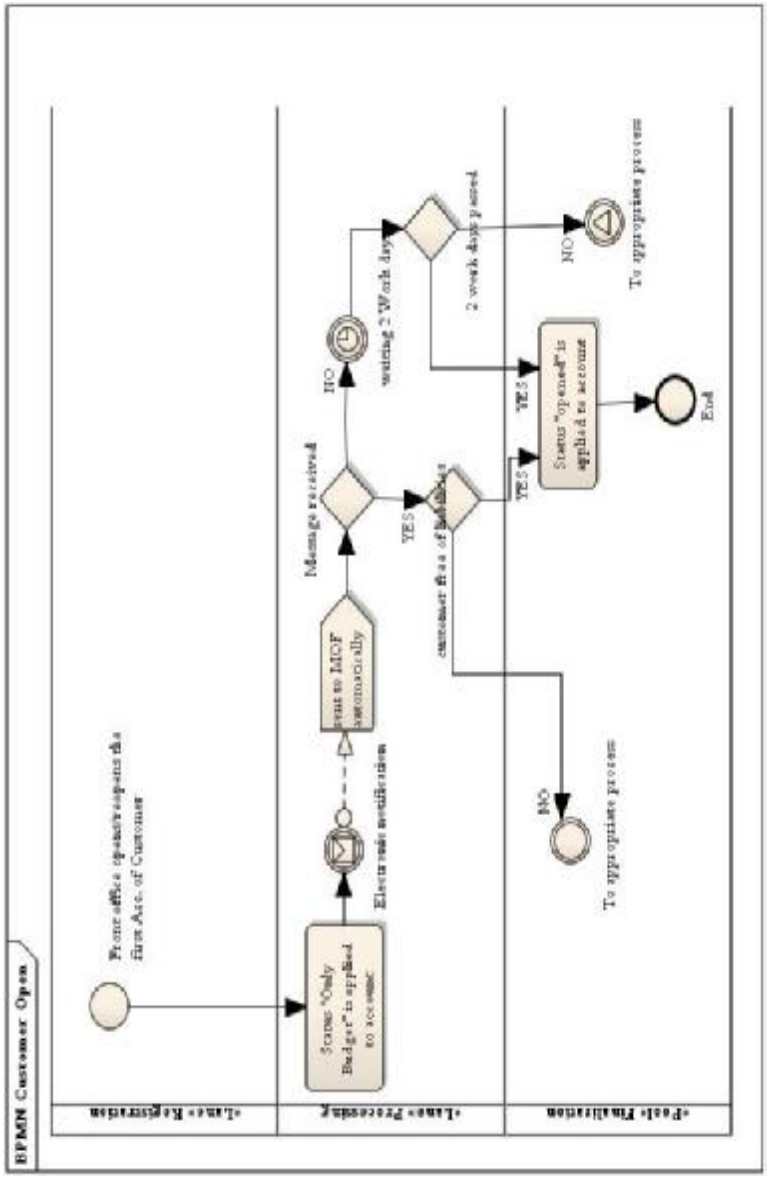
გავაანალიზოთ მათი დადებითი და უარყოფითი მხარეები. შევაფასოთ სისტემის ეფექტიანობა მისი მართვის სქემაში Web-სერვისების შემოტანით. პროცესების შესაფასებლად, შემდგომში

გამოვიყენებთ პეტრის ქსელების ინსტრუმენტს, რომელიც გვაძლევს ექსპერიმენტების ჩატარების საშუალებას.

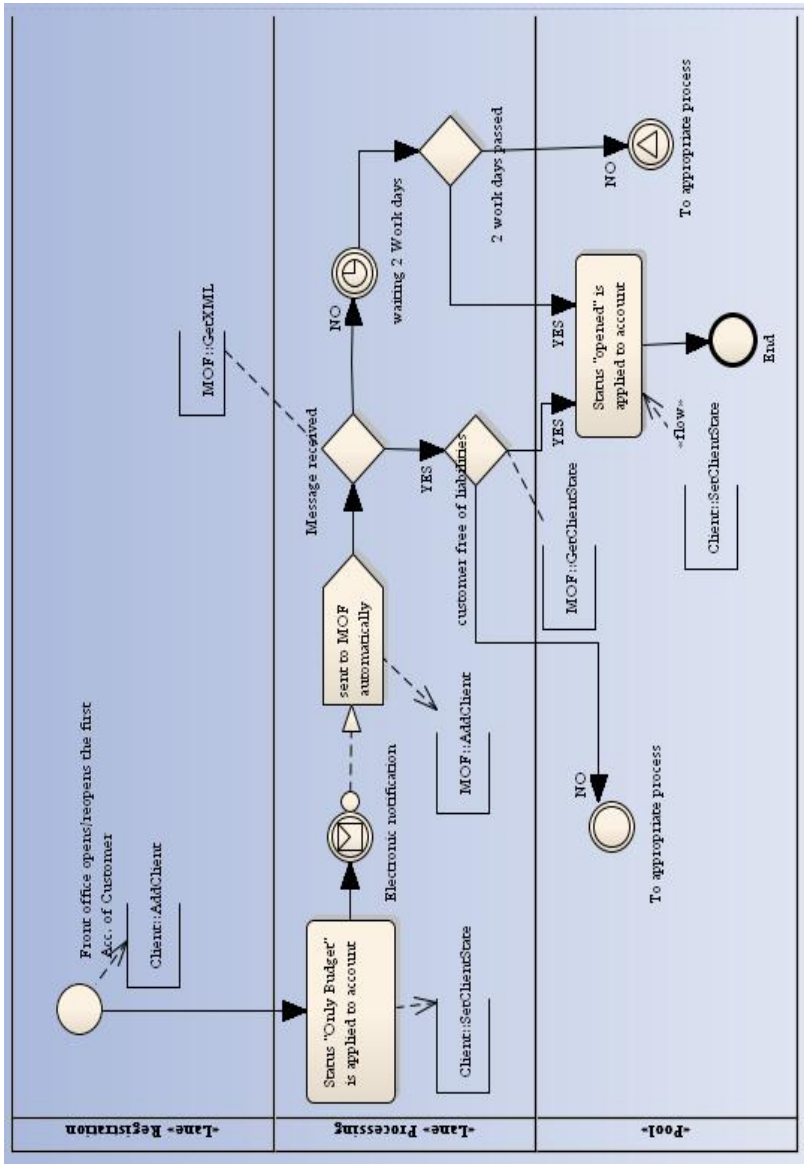
ამგვარად, 2.1 ნახაზზე ნაჩვენებია საფინანსო ბანკში „კლიენტის ანგარიშის გახსნის“ სერვისის ბიზნესპროცესის აღწერა. როგორც აღვნიშნეთ, a-ტრადიციული და b-სერვისზე ორიენტირებული:

a) კლიენტის რეგისტრაცია ხდება ოპერატორის მიერ, კლიენტების მართვის პროგრამის საშუალებით. ახალ კლიენტებს ენიჭება სტატუსი „მხოლოდ ბიუჯეტური“. ახალი კლიენტის მონაცემები ინახება ცხრილში, რომელშიც არსებულ მონაცემებსაც Window-ის სერვისი გარკვეული პერიოდულობით აფორმირებს XML სახით და ამ მონაცემებს უგზავნის შემოსავლების სამსახურის Web-სერვისს. ამ მონაცემებს შემოსავლების სამსახური ამოწმებს, აქვს თუ არა კლიენტს რაიმე ვალდებულება და შემდეგ ბანკის Web-სერვისს უგზავნის XML ფორმატის მონაცემებს, რომელიც შეიცავს კლიენტის იდენტიფიკატორს. ეს მონაცემები იწერება ცხრილში და შემდეგ Windows სერვისი მოახდენს ამ შეტყობინების შესაბამისად კლიენტის სტატუსის შეცვლას „აქტიურზე“ (თუ არ აქვს ვალდებულება).

b) ბანკის კლიენტის გახსნა შესაძლებელია სხვადასხვა პროგრამების საშუალებით, საჭიროა რომ განისაზღვროს კლიენტის პარამეტრების მინიმალური ოდენობა, რაც კლიენტის შექმნისთვის არის საჭირო. იქმნება კლიენტების Web-სერვისი, რომელსაც გააჩნია რამდენიმე მეთოდი, კლიენტების მონაცემების სამართავად. მათ შორის არის კლიენტების შექმნის მეთოდი. ამ მეთოდის გამოძახება შესაძლებელია სხვადასხვა პროგრამის მიერ, დამოუკიდებლად მათი ტექნოლოგიებისა.



ნახ.2.1-a. Client Open(): ტრადიციული



ნახ.2.1-ბ. Client Open(): სერვისები

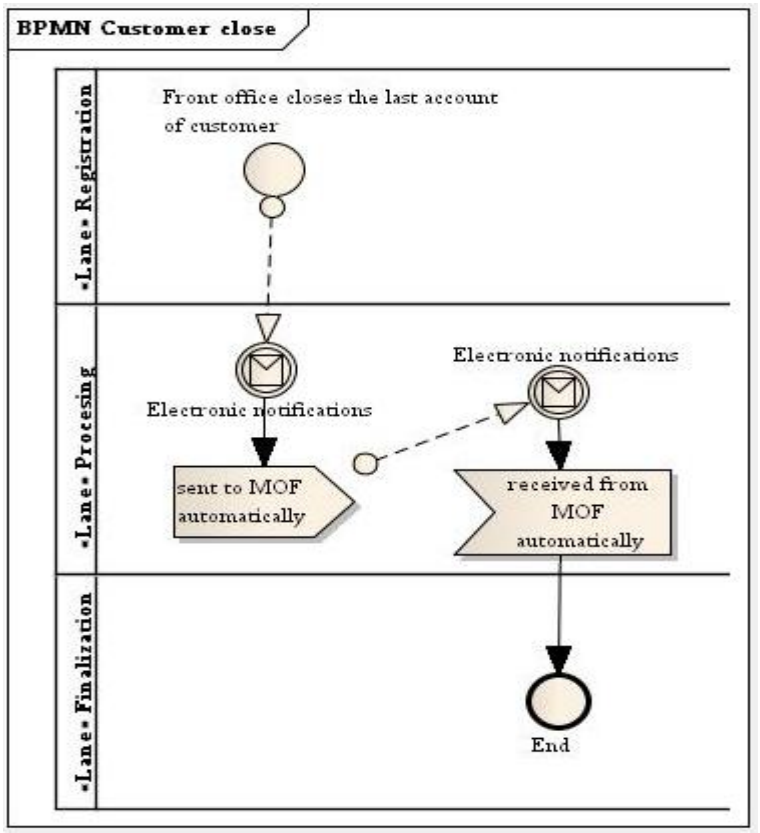
ახალი კლიენტის დამატების მეთოდი იძახებს კლიენტების სტატუსის ცვლილების მეთოდს, რომელიც კლიენტის სტატუსს აყენებს „მხოლოდ ბიუჯეტურზე“. ასევე იძახებს მეთოდს, რომელიც ამზადებს კლიენტის მონაცემებს შემოსავლების სამსახურში მონაცემების გასაგზავნად.

ეს მონაცემები იწერება ცხრილში. შემდეგ Windows-სერვისი გარკვეული პერიოდულობით კითხულობს ახალ მონაცემებს ამ ცხრილიდან და იძახებს შემოსავლების სამსახურის Web-სერვისს და გადასცემს კლიენტის მონაცემებს შესამოწმებლად.

შემოსავლების სამსახური, ამ მონაცემების გადამოწმების შემდეგ, ბანკის მხარეს იძახებს Web-სერვისს Accept_Message, რომელსაც გადმოეცემა შემოწმებული კლიენტის მონაცემები. ამ მონაცემების ჩაწერა ხდება ცხრილში და შემდეგ Windows- სერვისი მიღებული მონაცემების საფუძველზე შეუცვლის კლიენტს, Web-სერვისის გამოძახების საშუალებით, სტატუსს „აქტიურზე“.

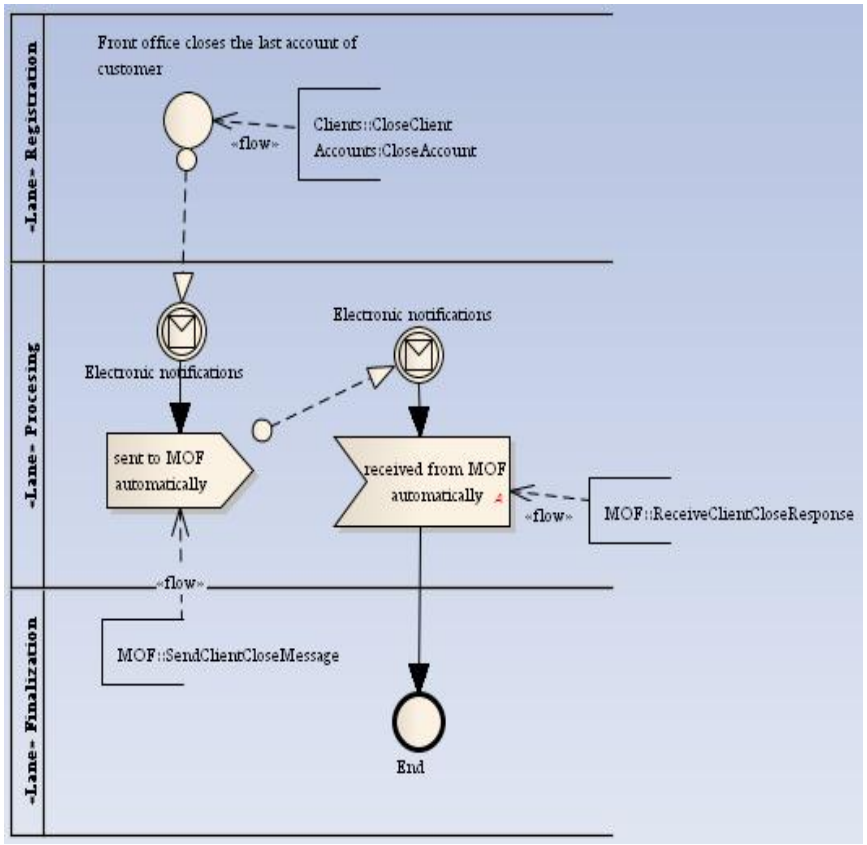
2.2–ა,ბ ნახაზებზე მოცემულია კლიენტის ანგარიშის დახურვის მოთხოვნის შესრულების ბიზნესპროცესები, ტრადიციული და სერვისორიენტირებული შემთხვევებისთვის:

a) კლიენტის დახურვა ხდება ოპერატორის მიერ, კლიენტების მართვის პროგრამის საშუალებით. ასევე ხდება მისი ანგარიშების დახურვა სათითაოდ. დახურული კლიენტის მონაცემები იწერება ცხრილში, რომელსაც Windows-სერვისი აფორმირებს XML-ის სახით და ასრულებს გაგზავნას შემოსავლების სამსახურში Web-სერვისის გამოძახებით.



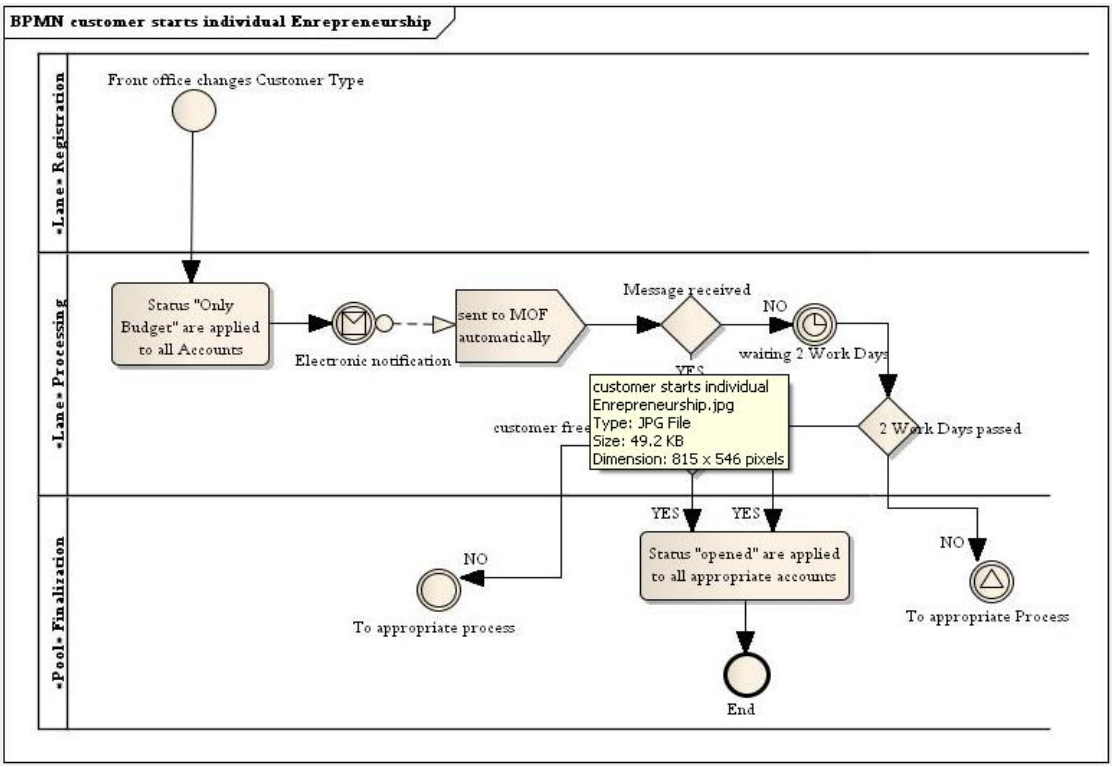
ნახ.2.2-a. Client Close(): ტრადიციული

b) კლიენტის ანგარიშის დახურვის შემდეგ, მოწმდება დარჩა თუ არა კლიენტს ღია ანგარიშები, თუ არცერთი ანგარიში აღარ არის აქტიური, ხდება კლიენტის მონაცემების ჩაწერა ცხრილში და Windows-სერვისი გადააგზავნის შემოსავლების სამსახურში შეტყობინებას კლიენტის დახურვის თაობაზე. შემოსავლების სამსახური უგზავნის დასტურს ბანკის Web-სერვისის გამოძახების საშალებით.

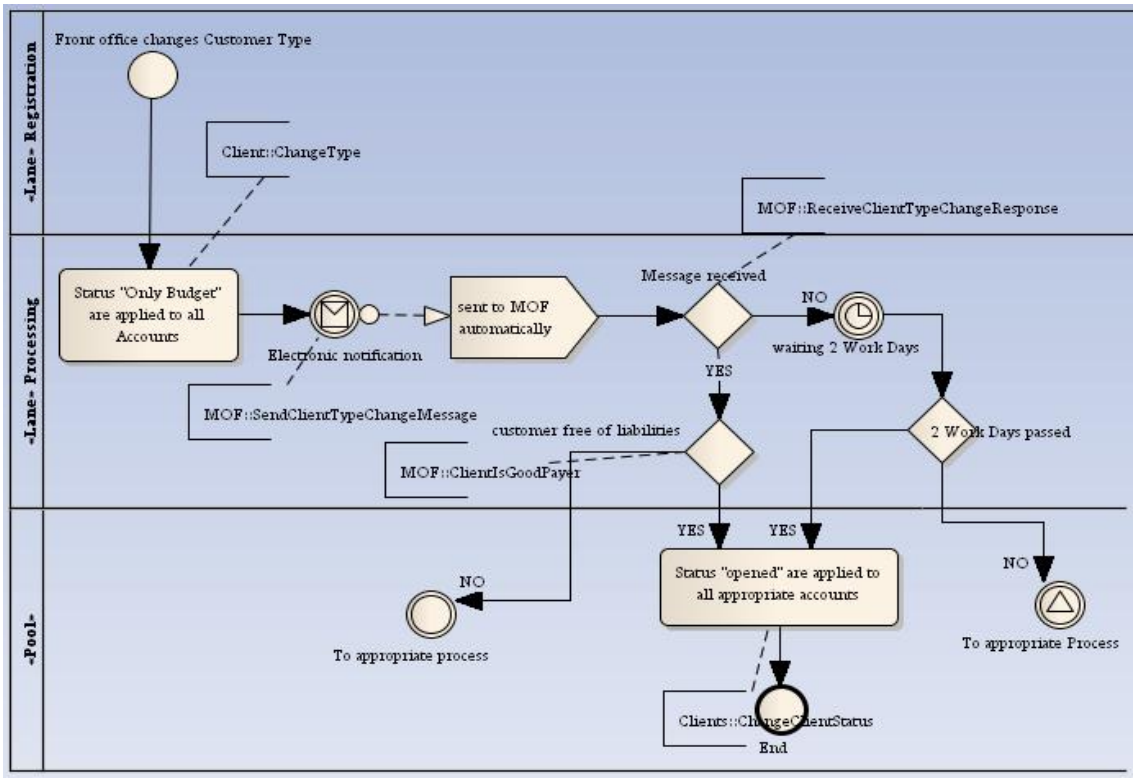


ნახ.2.2-b. Client Close(): სერვისებით

2.3–ა,ბ ნახაზებზე ასახულია კლიენტის სტატუსის ცვლილების მოთხოვნის ბიზნესპროცესის სქემები („არაგადამხდელი“ <—> „გადამხდელი“):



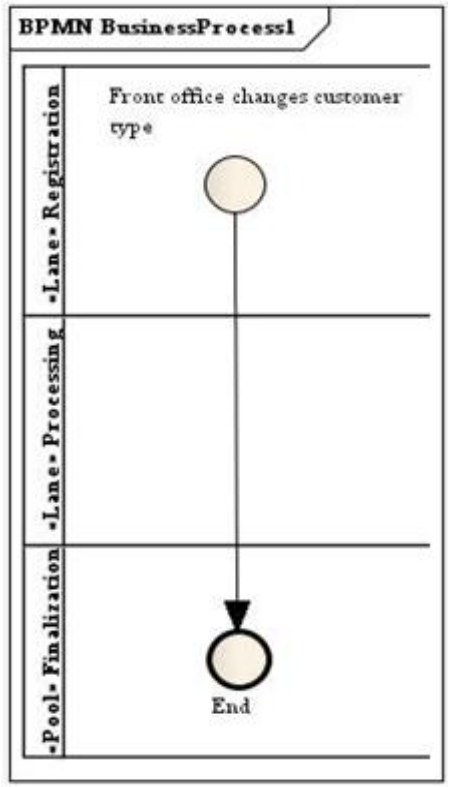
ნახ.2.3-a. Client Status(): ტრადიციული



ნახ.2.3-b. Client Status(): სერვისებით

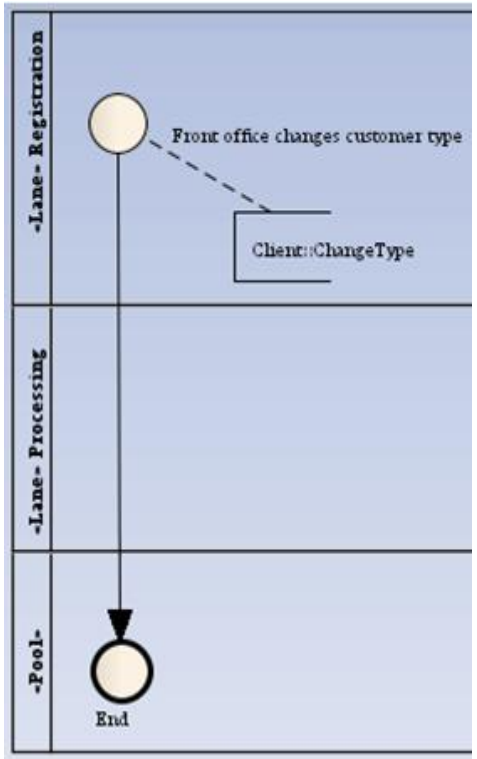
2.4–ა,ბ ნახაზებზე ასახულია კლიენტის ანგარიშის ტიპის ცვლილების მოთხოვნის ბიზნესპრცესის სქემები.

a) კლიენტის ტიპის ცვლილება ხდება ოპერატორის მიერ, კლიენტის განცხადების საფუძველზე. კლიენტს ეცვლება სტატუსი გადამხდელზე, ან პირიქით-არაგადამხდელზე. ამ ცვლილების შესახებ შეტყობინება უნდა გაეგზავნოს შემოსავლების სამსახურს. კლიენტის ტიპის შეცვლის შეტყობინება იწერება ცხრილში და Windows-სერვისი ახდენს XML მონაცემების ფორმირებას და Web-სერვისის გამოძახებას. შემოსავლების სამსახური აგაზავნის დასტურს ბანკის მხარეს Web-სერვისის გამოძახების საშუალებით.



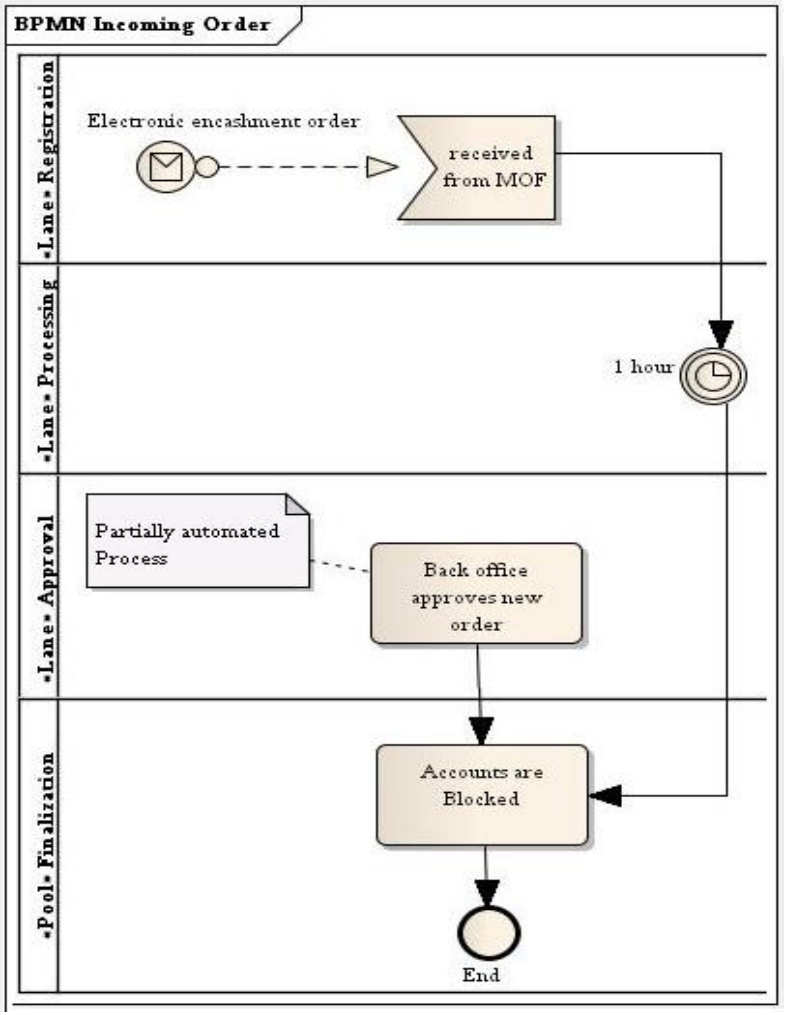
ნახ.2.4-a. Client ChangeType(): ტრადიციული

b) კლიენტის ტიპის შეცვლა შესაძლებელია გაკეთდეს სხვადასხვა აპლკაციების საშუალებით. კლიენტის მონაცემთა ბაზაში კლიენტის ტიპის ცვლილება ხორციელდება კლიენტების სერვის-მეთოდის გამოძახებით, რომელიც ცვლის კლიენტის ტიპს, და გასაგზავნ შეტყობინებათა ცხრილში წერს მონაცემებს, რომელსაც Windows- სერვისი შემდგომ გააგზავნის შემოსავლების სამსახურში. შემოსავლების სამსახური იძახებს ბანკის Web-სერვისს და გადმოცემს დასტურს კლიენტის ტიპის ცვლილებაზე. თუ კლიენტს გააჩნია რაიმე ვალდებულება, კლიენტის ანგარიშებს დაუყენდებათ სტატუსი „მხოლოდ ბიუჯეტური“.

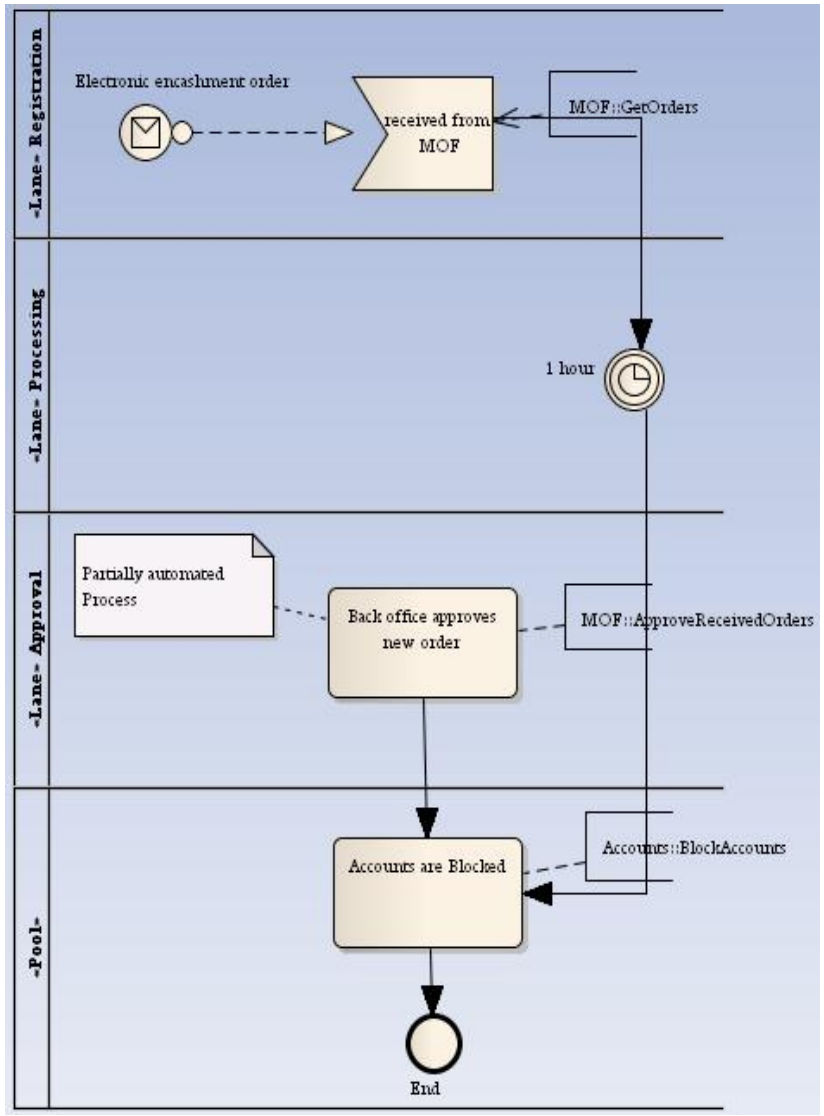


ნახ.2.4-b. Client ChangeType(): სერვისებით

2.5–ა,ბ ნახაზებზე ასახულია კლიენტის ანგარიშებზე ინკასოს წარმოდეგენის მოთხოვნის ტრადიციული და სერვისული ბიზნესპრცესის სქემები.



ნახ.2.5-a. Incoming Order(): ტრადიციული



ნახ.2.5-b. Incoming Order(): სერვისებით

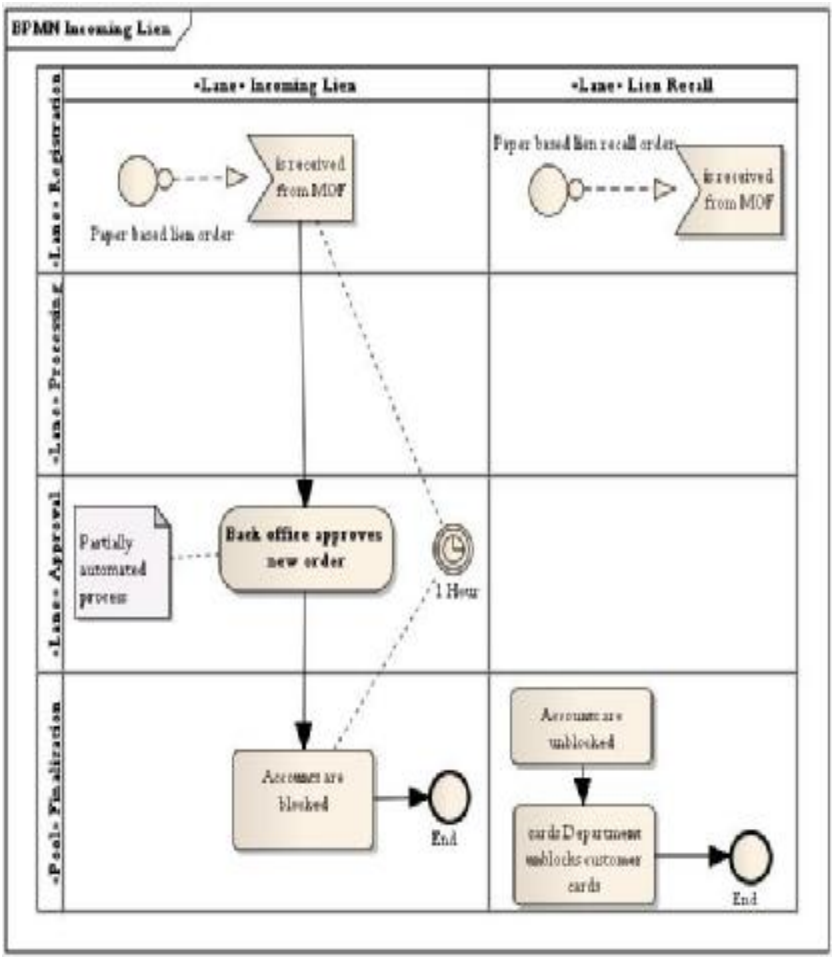
ა) კლიენტის ანგარიშებზე ინკასოს წარმოდეგნისთვის შემოსავლების სამსახური აფორმირებს XML ფორმატის შეტყობინებას, რომელიც შეიცავს კლიენტის მონაცემებს და თანხის ოდენობას. ეს შეტყობინება იგზავნება ბანკის Web-სერვის, რომელიც ცხრილში ჩაწერს ამ მონაცემებს. ოპერატორი ინკასოების მართვის მოდულში ნახულობს შემოსული ინკასოების სიას და ადასტურებს. დადასტურებულ ინკასოებზე Windows სერვისი კლიენტის ანგარიშებს ანიჭებს სტატუს „მხოლოდ ბიუჯეტურს“.

ბ) კლიენტის ანგარიშებზე ინკასოს წარმოგენისთვის შემოსავლების სამსახური აფორმირებს XML ფორმატის შეტყობინებას, რომელიც შეიცავს კლიენტის მონაცემებს და თანხის ოდენობას. ეს შეტყობინება იგზავნება ბანკის Web-სერვისს, რომელიც ცხრილში ჩაწერს ამ მონაცემებს. ოპერატორი ინკასოების მართვის მოდულში ნახულობს შემოსული ინკასოების სიას და ადასტურებს. დადასტურებულ ინკასოებზე Windows სერვისი კლიენტის ანგარიშებს ანიჭებს სტატუსს „მხოლოდ ბიუჯეტური“.

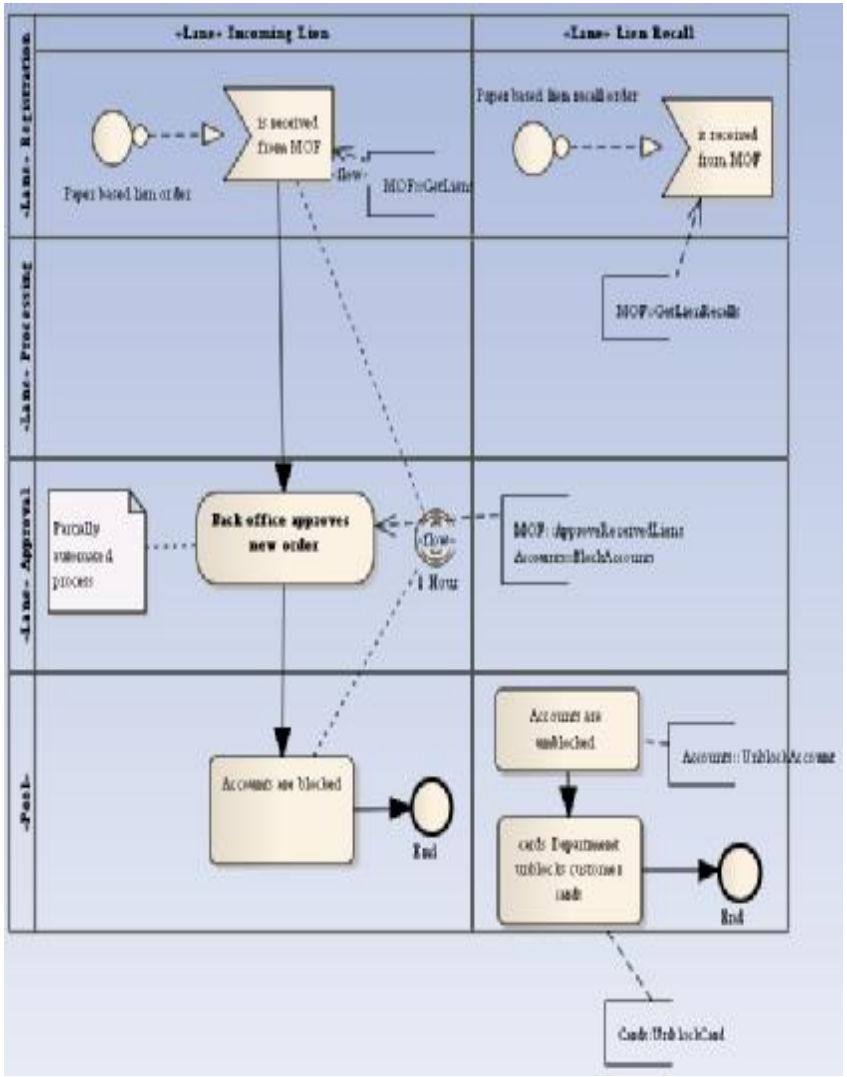
2.6–ა,ბ ნახაზებზე ასახულია კლიენტის ანგარიშებზე ყადაღის წარმოდეგენის მოთხოვნის ტრადიციული და სერვისული ბიზნესპრცესის სქემები.

ა) კლიენტის ანგარიშებზე ყადაღის წარმოგენისთვის შემოსავლების სამსახური აფორმირებს XML ფორმატის შეტყობინებას, რომელიც შეიცავს კლიენტის მონაცემებს და თანხის ოდენობას. ეს შეტყობინება იგზავნება ბანკის Web-სერვისს, რომელიც ცხრილში ჩაწერს ამ მონაცემებს. ოპერატორი ინკასოების მართვის მოდულში ნახულობს შემოსული ინკასოების სიას და ადასტურებს. დადასტურებულ ინკასოებზე Windows სერვისი კლიენტის ანგარიშებს ანიჭებს სტატუს „მხოლოდ ბიუჯეტური“. ყადაღების გაწვევისთვის შემოსავლების სამსახური აფორმირებს XML შეტყობინებას და უგზავნის ბანკის Web-სერვისს. ბანკში

ხდება XML-ის დამუშავება და შესაბამის კლიენტის ანგარიშებზე შეზღუდვის მოხსნა და ბართების განბლოკვა.



ნახ.2.6-a. Incoming recall Lien(): ტრადიციული



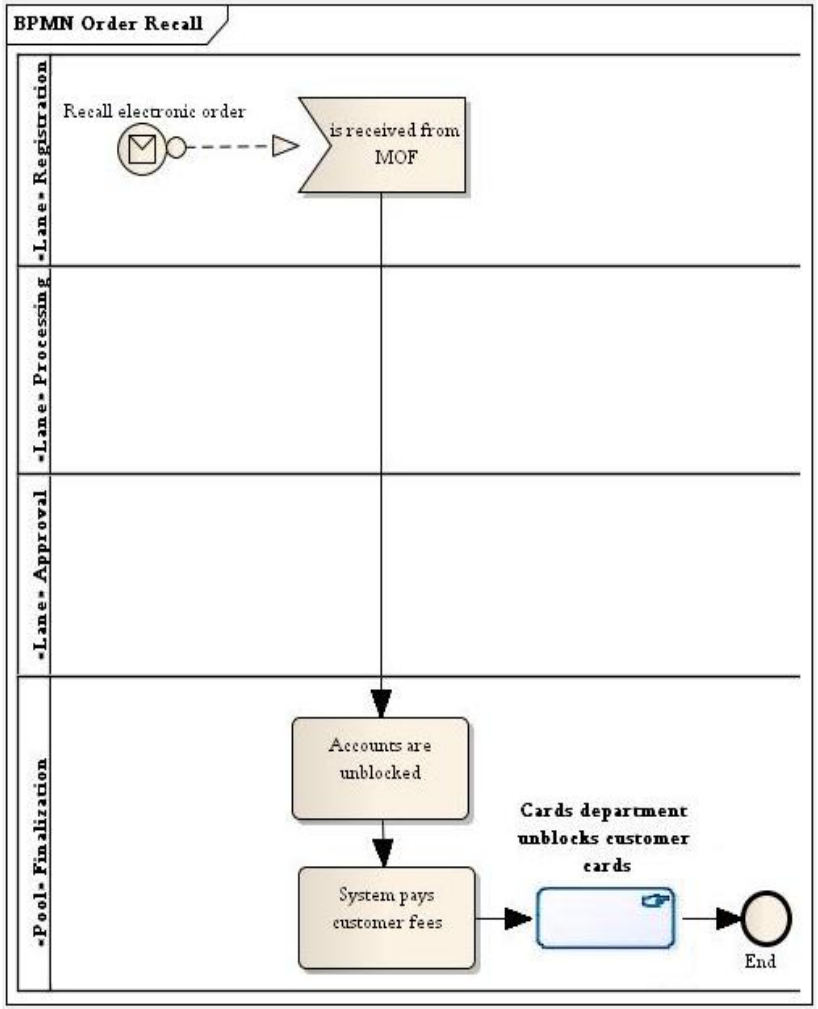
ნახ.2.6-b. Incoming recall Lien(): სერვისებით

ბ) შემოსავლების სამსახურიდან ყადაღის გაწვევა გაეგზავნება ბანკის Web-სერვისს XML-ის სახით. იწერება ცხრილში, შემდეგ Windows სერვისი იძახებს Web-სერვისს GetLienRecall, რომელიც აბრუნებს შემოსულ ყადაღას გაწვევებს. გამოიძახება ვებ–სერვისი UnblockAccounts, რომელიც ანგარიშებს აძლევს სტატუსს „აქტიური“, შემდეგ გამოიძახება UnblockCards, რომელიც ახდენს ბარათების განბლოკვას.

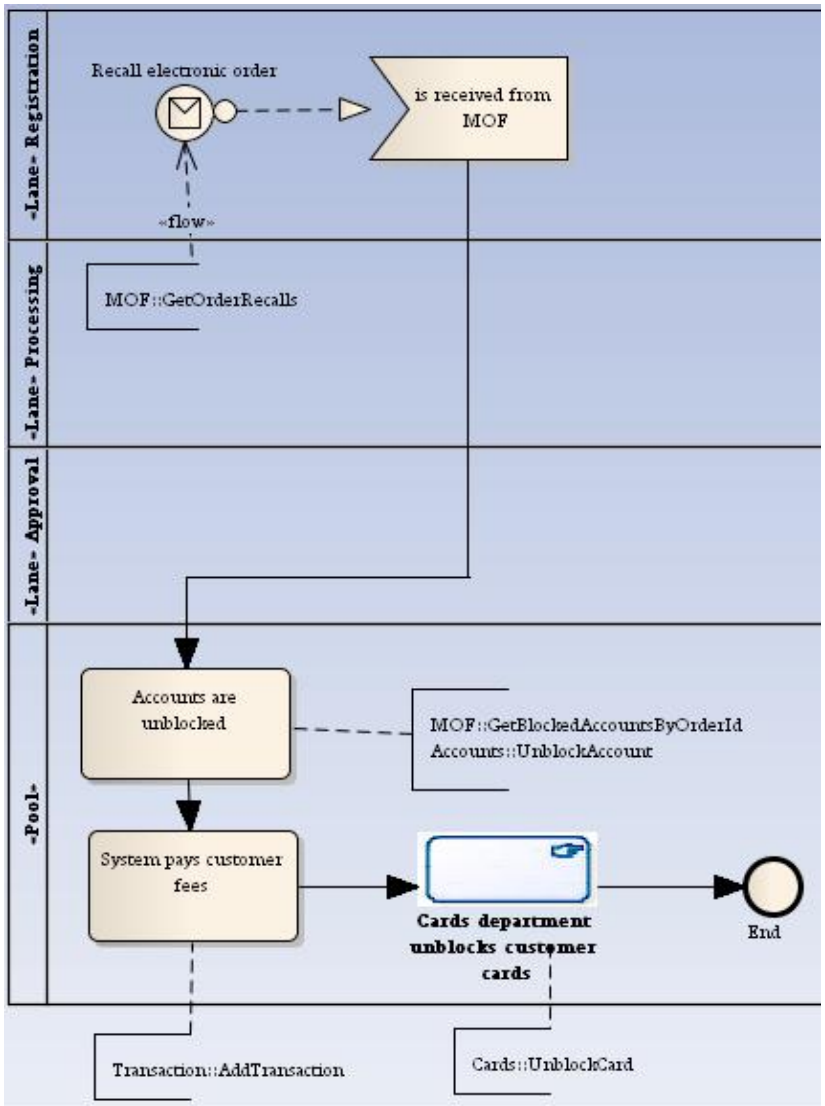
შემოსული ყადაღების სია გამოიძახება მომხმარებლის ინტერფეისიდან Web-სერვისი GetLiens გამოძახებით, ბანკის თანამშრომელი დაამტკიცებს ამ ყადაღების სიას, რის შემდეგაც გამოიძახება Web-სერვისი ApproveLiens და BlocackAccounts, BlockCards, რომლებიც ბლოკავენ ანგარიშებს, ანიჭებენ სტატუსს „მხოლოდ ბიუჯეტური“ და ბლოკავენ ბარათებს.

2.7–ა,ბ ნახაზებზე ასახულია ინკასოების გაწვევის მოთხოვნის ტრადიციული და სერვისული ბიზნესპრცესის სქემები.

ა) ინკასოების გაწვევა შემოსავლების სამასხურიდან იგზავნება XML-ის სახით, რომელიც გადმოეცემა ბანკის Web-სერვისს AcceptMessage. ეს ინფორმაცია იწერება ბაზაში და შემდეგ Windows სერვისი იძახებს ბაზის პროცედურებს, რომლებიც ასრულებს ანგარიშების სტატუსის ცვლილებას. ბარათების დეპარტამენტის თანამშრომელი კი გაწვევების სიის მიხედვით ახდენს ბარათების განბლოკვას მანუალურად, პროგრამის საშალებით.



ნახ.2.7-a. Incoming Order Recall(): ტრადიციული

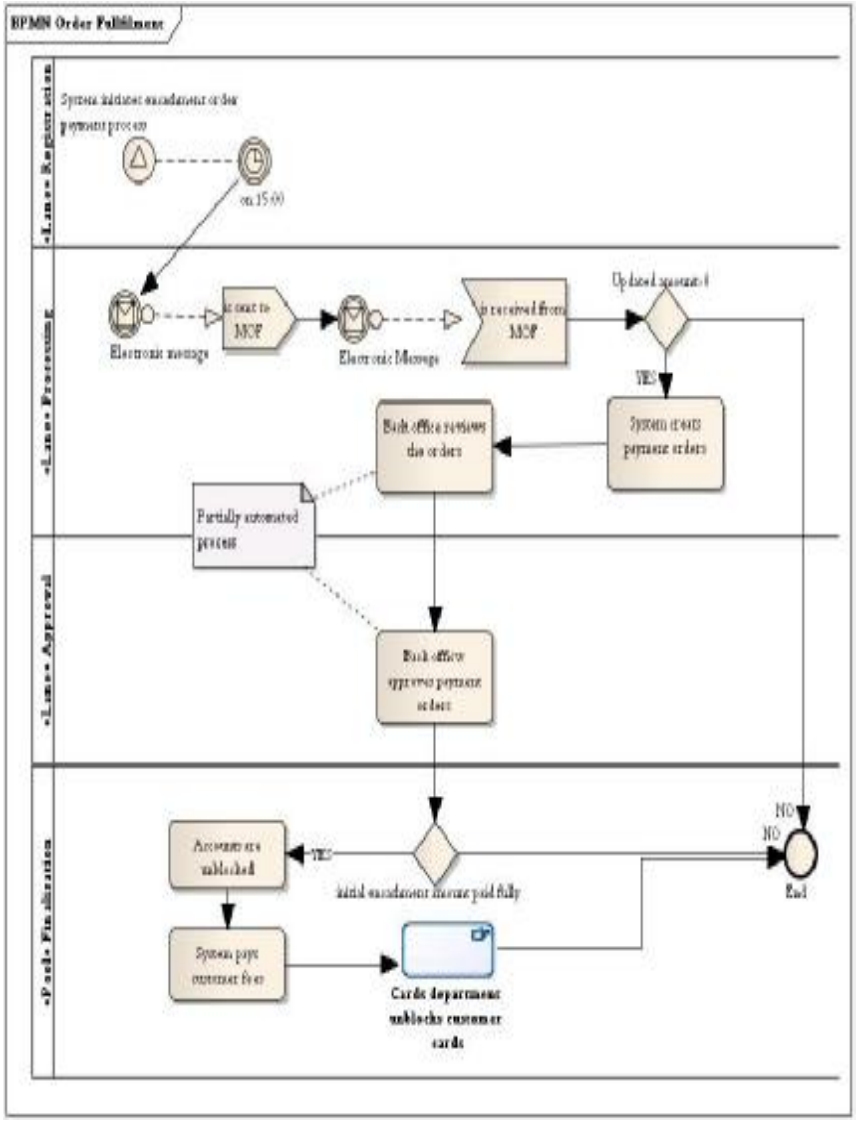


ნახ.2.7-b. Incoming Order Recall(): სერვისებით

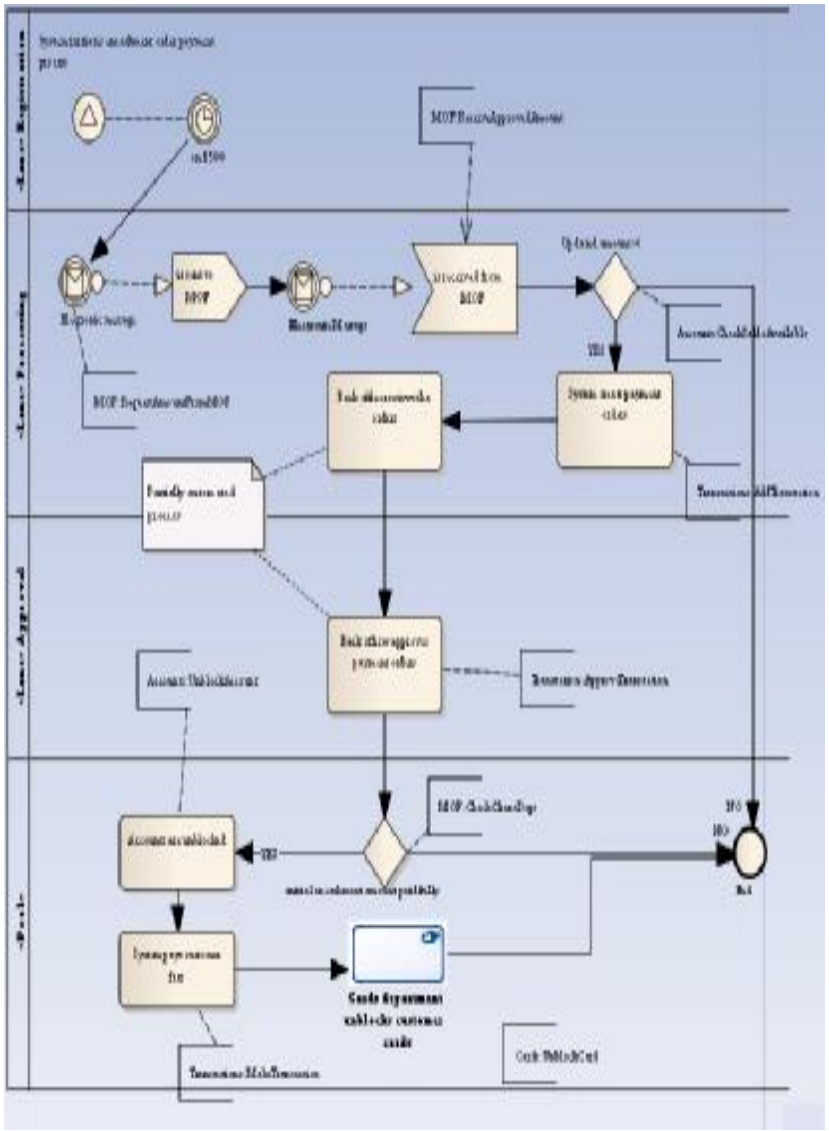
ბ) ინკასოების გაწვევა ბანკს გადმოეცემა XML-ის სახით AcceptMessage პროცედურა. მიღებული XML-ის მონაცემები იწერება ცხრილში და შემდგომ გამოიძახება GetOrderRecalls Web-სერვისი, ამ სიის მიხედვით ხდება უკვე ინკასოდადებულ ანგარიშების მოძებნა, GetBlockedAccouontsByOrderID მეთოდი აბრუნებს თითოეული ინკასოსთვის დაბლოკილ ანგარიშის მონაცემებს, ამ ანგარიშების სიის მიხედვით ხდება UnblockAccounts მეთოდის გამოძახება და ანგარიშების სტატუსის შეცვლა „აქტიურზე“. თუ ანგარიში არის საბარათე, მაშინ გამოიძახება UnblockCard Web-სერვისის მეთოდი, რომელიც განბლოკავს ბარათებს.

2.8–ა,ბ ნახაზებზე ასახულია ბანკში მიღებული ინკასოების დამუშავების მოთხოვნის ტრადიციული და სერვისული ბიზნესპრცესის სქემები.

ა) ბანკში მიღებული ინკასოების დამუშავება იწყება წინასწარ განსაზღვრულ დროს. Windows სერვისი ბაზის ცხრილებში ნახულობს დასამუშავებელი ინკასოების სიას. ამ მონაცემების საფუძველზე ქმნის XML-ს რომელიც შეიცავს ინკასოების ნომრებს, ეს XML ეგზავნება შემოსავლების სამსახურს, რომელიც ასევე XML უბრუნებს ბანკის Web-სერვისს, რომელიც შეიცავს ინკასოების ნომრებს და შესაბამის თანხას. ეს ინფორმაცია იწერება ბაზაში. ბანკის თანამშრომელი პროგრამის საშუალებით ნახულობს ინკასოების სიას, ამტკიცებს მას, რის შემდეგაც გამოიძახება ბაზის პროცედურები, რომლებიც ატარებენ ტრანზაქციებს, იხდიან ანგარიშებიდან თანხებს შემოსავლების სამსახურის ანგარიშზე. თუ თანხის დაფარვა მოხდება სრულად, შეიცლვლება ანგარიშის სტატუსი „აქტიურზე“ და მოხდება ბარათების განბლოკვა თანამშორლის მიერ.



ნახ.2.8-a. Incoming Order Fulfilment (): ტრადიციული



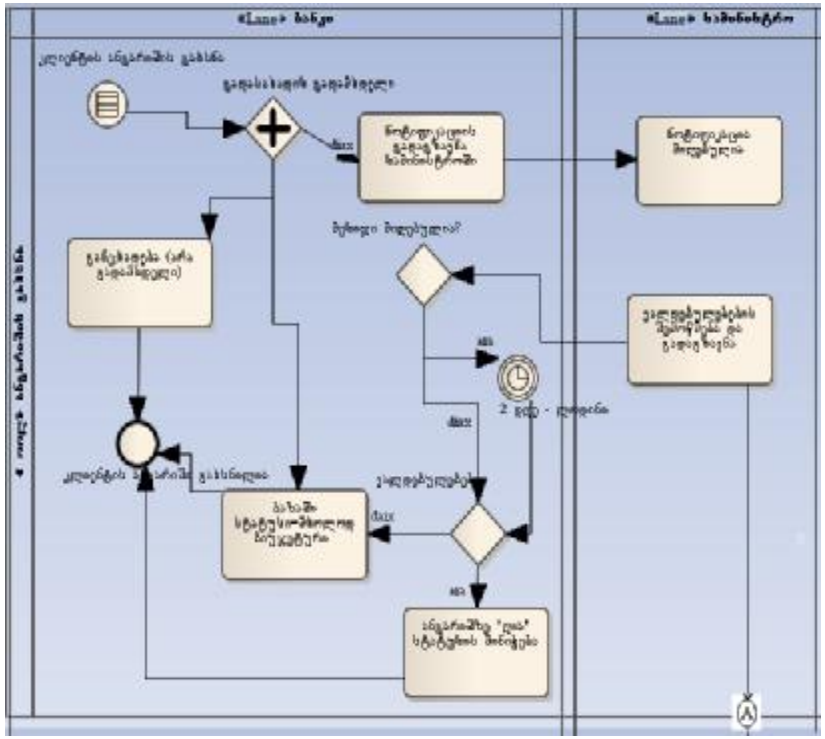
ნახ.2.8-b. Incoming Order Fulfilment (): სერვისებით

ბ) ბანკში მიღებული ინკასოების დამუშავება იწყება წინასწარ განსაზღვრულ დროს. Windows სერვისი ბაზის ცხრილებში ნახულობს დასამუშავებელი ინკასოების სიას, იძახებს GetActiveOrder-ს მეთოდს, მიღებული ინკასოების ნომრები გადაეცემა RequestAmountFromMof მეთოდს, რომელიც აფორმირებს XML შეტყობინებას და იძახებს შემოსავლების სამსახურის Web-სერვისს. შემოსავლების სამახური XML-ის სახის შეტყობინებას უბრუნებს ბანკის Web-სერვისს, რომელიც შეიცავს ინკასოების ნომრებს და შესაბამის თანხას. ეს ინფორმაცია იწერება ბაზაში. ბანკის თანამშრომელი პროგრამის საშუალებით ნახულობს ინკასოების სიას, გამოიძახება Web-სერვისის მეთოდი Get, ამტკიცებს მას, რის შემდეგაც გამოიძახება ბაზის პროცედურები, რომლებიც ატარებენ ტრანზაქციებს, იხდიან ანგარიშებიდან თანხებს შემოსავლების სამსახურის ანგარიშზე. თუ თანხის დაფარვა მოხდება სრულად, შეიცლვლება ანგარიშის სტატუსი „აქტიურზე“ და მოხდება ბარათების განბლოკვა თანამშორლის მიერ.

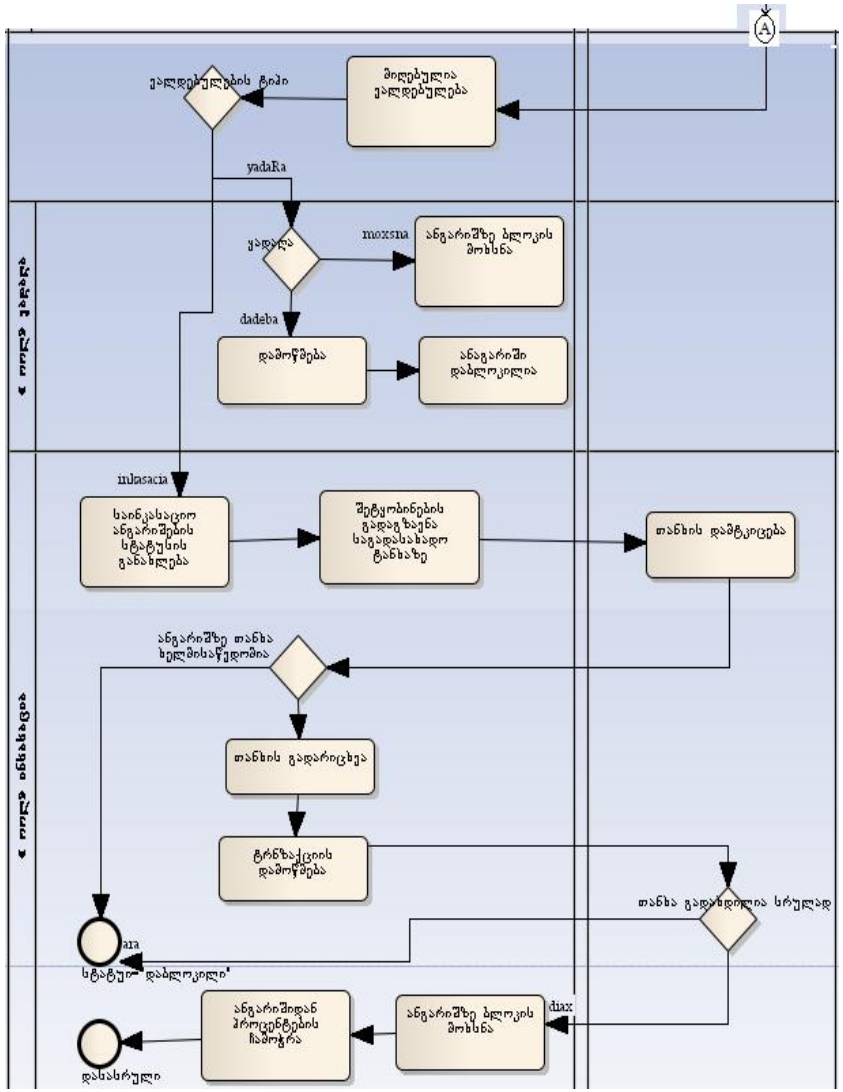
2.9-ა,ბ ნახაზებზე ილუსტრირებულია ინტერკორპორაციულ აპლიკაციებში მომქმედი გლობალური ბიზნესპროცესების BPMN ნოტაციით აგებული მოდელი, ტრადიციული და Web-სერვისების გამოყენებით, შესაბამისად.

მნიშვნელოვანია, რომ რამდენიმე ოპერაციის (მაგ., კლიენტის ანგარიშის გახსნა, დახურვა, ცვლილება, ყადაღის დადება, ყადაღის მოხსნა, ინკასო) წარმოებისთვის, გამოყენებაშია ერთი ტიპის ბიზნესპროცესები (მაგ., setclientState, blockaccount, unblockaccount), რაც სერვისორიენტირებული მიდგომის გამოყენებით შესაძლოა გავაერთიანოთ ერთ Web-სერვისში.

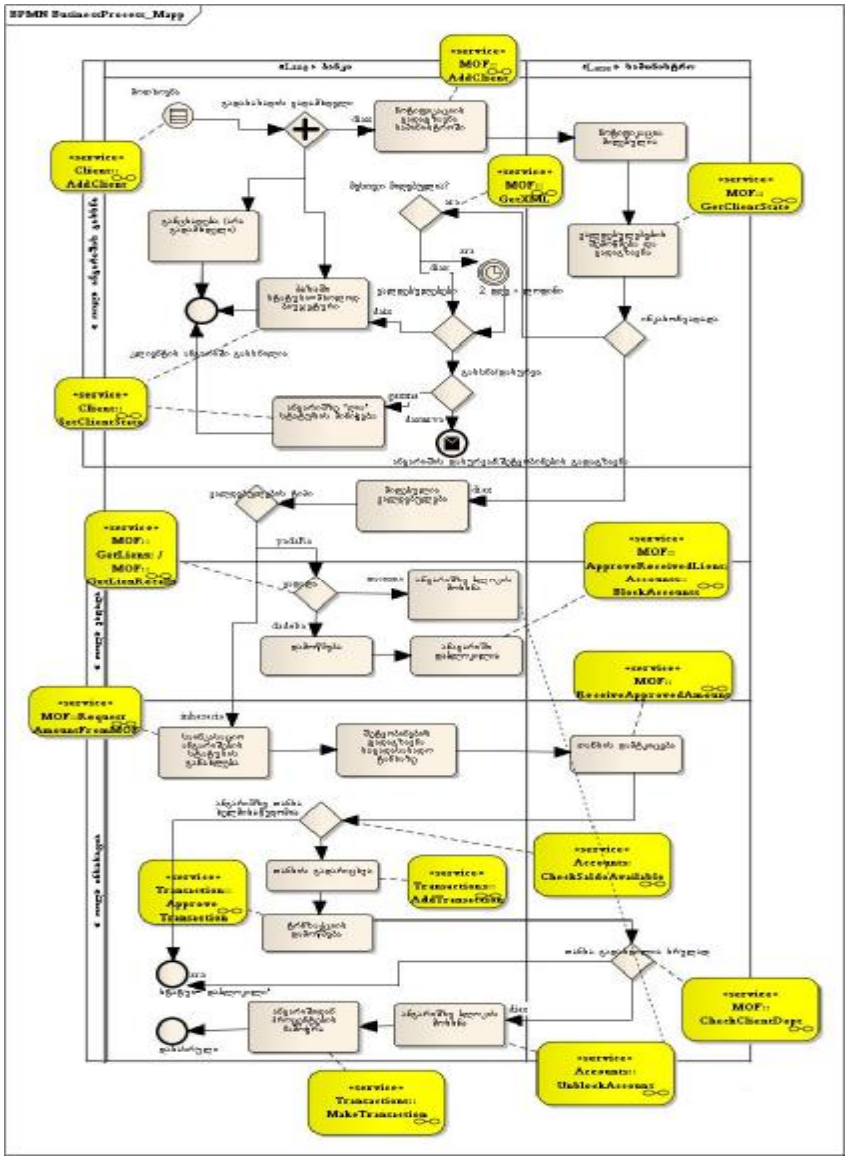
ამავდროულად, უკვე რეალიზებული Web-სერვისი (მაგალითად, კლიენტის ანგარიშის მოდიფიკაცია), შესაძლებელია ასევე გამოვიყენოთ სხვა აპლიკაციის რეალიზაციის (მაგალითად, კლიენტისთვის ინტერნეტ-ბანკინგის ჩართვა) ან სხვადასხვა აპლიკაციების ერთობლივი მუშაობის დროს (მაგალითად, ბანკის საბაჟო სტრუქტურასთან ჰორიზონტალური ინტეგრაცია).



ნახ.2.9-a. BusinessProcess_Map(): ტრადიციული (დასაწყისი)



ნახ.2.9-a. BusinessProcess_Map(): ტრადიციული (გაგრძელება)



ნახ.2.9-ბ. BusinessProcess_Map(): სერვისებით

2.2. ინტერკორპორაციული აპლიკაციების ჰორიზონტალური და ვერტიკალური ინტეგრაციის მართვა სერვისორიენტირებული არქიტექტურის ბაზაზე

კორპორაციული სტრუქტურის კომპანიების ფუნქციონირების ერთ-ერთი ძირითადი მახასიათებელია ბიზნეს მოთხოვნების მუდმივი ცვლილება, რასაც იწვევს სერვისების გაუმჯობესების, შიგა იერარქიული სტრუქტურის რეორგანიზაციის, გარე სისტემებთან თანამშრომლობის და სხვ. ფაქტორები.

ამ ტიპის ცვლილებების მხარდაჭერა და ასახვა კომპანიის ფუნქციონალური მართვის პროგრამულ პროდუქტებში არის ურთულესი პროცესი, რადგან მუშა პროგრამულ აპლიკაციაში ახალი ბიზნეს-მოთხოვნების ფუნქციონალის ჩაშენებასთან ერთად, დიდია რისკ-ფაქტორები ცვლილებების გავლენის საზღვრების დადგენაში, რომელიც უშუალოდ ეხება იერარქიულად დამოკიდებულ კლასებს, მეთოდებსა და მონაცემებს.

საქმიანი პროცესების მართვა ხშირ შემთხვევაში წარმოებს როგორც კომპანიის შიგა სტრუქტურის ფარგლებში, ისე მრავალორგანიზაციული მასშტაბით. ბიზნეს-ფუნქციების გადანაწილების ამგვარი სპეციფიკა საჭიროებს სხვადასხვა აპლიკაციებში დამუშავებული ინფორმაციის კომპოზიციასა და სინქრონიზაციას, რაც ჰორიზონტალური და ვერტიკალური ინტეგრაციის პრობლემებს მიეკუთვნება. ჰორიზონტალური და ვერტიკალური ინტეგრაციის კონცეფცია მნიშვნელოვანი ფატქორია ინტრა– და ინტერკორპორაციული აპლიკაციების ბიზნესპროცესების მართვის პროცესში.

ობიექტორიენტირებული დაპროგრამების მეთოდი არასაკმარისი აღმოჩნდა სხვადასხვა ტიპის აპლიკაციების ინტეგრაციის პობლემების გადასაჭრელად. Web–აპლიკაციების პროგრამული დანართების, ბიზნესპროცესების მართვის ჰორიზონტალური და ვერტიკალური ინტეგრაციის პრობლემების გადასაწყვეტად მნიშვნელოვანია შემდეგი ინფოტექნოლოგიები: SOA, XML ენა, მოდელებით მართვადი არქიტექტურა (MDA), კომპოზიციური დანართების აგების ტექნოლოგია და სხვ. [2,15].

სერვისორიენტირებული არქიტექტურა, რომლის მთავარი არსია პროცესზე ორიენტირება, აერთიანებს სხვადასხვა კომპაქტურ ბიზნესპროცესებს, წარმოდგენილს Web-სერვისების სახით პოლიმორფიზმის პრინციპით და ახდენს მათ ორკესტრირებას. განსაკუთრებით მოქნილს და რაც მთავარია შესაძლებელს ხდის ინფორმაციის კომპოზიციისა და სინქრონიზაციის მართვას მრავალაპლიკაციური მუშაობის რეჟიმში. სერვისორიენტირებული არქიტექტურის ბირთვია Web-სერვისი. Web-სერვისის ფორმირება ხდება განზოგადებული სახით, რაც ლოგიკურად მსგავს სხვადასხვა აპლიკაციებში გამოყენების შესაძლებლობას იძლევა. Web-სერვისის გამოყენება აღმოფხვრავს პროგრამული კოდის განმეორებადობას (ტიპური დავალებების ავტონომიურად არსებობის თვალსაზრისით), აჩქარებს პროგრამული სისტემების შექმნის დროს და ამცირებს ცვლილებების გავლენის რისკებს.

სერვისორიენტირებული არქიტექტურა აფართოებს ობიექტორიენტირებული ტექნოლოგიის გამოყენების საზღვრებს და ობიექტორიენტირებული სისტემების ინტეგრაციის შესაძლებლობას ქმნის. მთავარი განსხვავება ამ ორ

ტექნოლოგიას შორის, პრაქტიკულად, მეთოდების განზოგადებისა და წვდომის ასპექტში იკვეთება.

პრინციპული განსხვავება ობიექტორიენტირებულ და სერვისორიენტირებულ მიდგომებს შორის არის ორიენტირება ბიზნესპროცესზე და არა ობიექტზე. ამ პრინციპით, Web-სერვისში ინკაფსულირებულია ბიზნეს-ფუნქცია აბსტრაქციის მაღალი დონით, რაც უნიფიცირებული პროცესის შაბლონად შექმნის შესაძლებლობას ქმნის. სერვისორიენტირებული არქიტექტურის ერთ-ერთი პრინციპია ტიპიზირებული Web-სერვისების ბიბლიოთეკის არსებობა, მასში ამ შაბლონების შენახვა Web-სერვისის სახით და მისი მრავალჯერადად გამოყენება. ეს არის ობიექტორიენტირებული ტექნოლოგიის ობიექტების ბიბლიოთეკის ანალოგია, სადაც ობიექტი წარმოდგენილია პროცესის სახით და გატანილია სერვერზე. ამავდროულად, სერვისორიენტირებულ არქიტექტურის პრაქტიკული რეალიზაციაა არსებული Web-სერვისების ნაკრებისგან (უმეტესად ობიექტორიენტირებულ სისტემებში შექმნილი) კონკრეტული ბიზნეს-სცენარის აწყობა, რაც ორკესტრირებისა და ქორეოგრაფიის კონცეფციით ხორციელდება (იხ.თავი 1).

მაგალითად, მონაცემთა ბაზასა და პროგრამულ დანართს შორის ტრიგერული ფუნქციების (დამატება, განახლება, ძებნა, წაშლა) წარმოებისთვის საჭიროა მონაცემთა ბაზის მხარეს შეიქმნას პროცედურა, ხოლო პროგრამული სისტემის მხარეს დაიწეროს ამ პროცედურის წაკითხვისა და აპლიკაციასთან ურთიერთობის მეთოდები. ეს პროცესი იდენტურია ყოველი ცხრილისთვის და უფრო ზედა დონეზე ყოველი აპლიკაციისთვის, რომელიც მონაცემთა ბაზასთან მუშაობს. ობიექტორიენტირებულ პლატფორმებში ამ ტიპის

პროცესისთვის იქმნება ზოგადი კლასი, რომელიც მართავს მონაცემთა ბაზასა და აპლიკაციას შორის მონაცემთა მანიპულირებას ერთი პროექტის ფარგლებში (პოლიმორფიზმი). ამ პროცესის Web-სერვისის სახით გარდაქმნისას, უკვე შესაძლებელია ნებისმიერი პროექტისა და ნებისმიერი პლატფორმის დონეზე (XML-WSDL ენების საშუალებით) ერთხელ შექმნილი კლასის გამოყენება.

პრაქტიკული გამოყენების თვალსაზრისით Web-სერვისი შეიძლება შედარდეს სკრიპტულ ენაზე დაწერილ პორტლეტს, სხვადასხვა ტიპის მარტივი ფუნქციონალობის კომპონენტებს (მაგ., კალკულატორი, ინტერნეტით თანხის გადახდის ფუნქცია, აპლიკაციის მართვის ელემენტები) შინაარსობრივად უფრო გაფართოებული (ბიზნესფუნქციაზე აყვანილი) და გამოსაყენებლად საკმაოდ მოქნილი (პლატფორმა-დამოუკიდებელი) შესაძლებლობებით.

სერვისორიენტირებული მიდგომა ობიექტორიენტირებულ მეთოდოლოგიაზე რეალიზებული სისტემის გაფართოების ხერხია, ამდენად პროგრამული უზრუნველყოფის სისტემის მოდელირებისას ძირითადად, ისევ გამოიყენება ობიექტ-ორიენტირებული დაპროექტებისა და მოდელირების ტექნოლოგია (პროტოტიპული პრინციპი, ვერსიების მართვა და ა.შ.), როგორც სისტემის შინაარსობრივი, ისე Web-სერვისის ფუნქციონალური აღწერისთვის. სისტემის სერვის-ორიენტირებული არქიტექტურის მხარე აღიწერება ბიზნესპროცესების მოდელირების ნოტაციისა და ბიზნეს-პროცესების შესრულების ენის საფუძველზე. ბიზნესპროცესების მოდელირების ნოტაცია ობიექტორიენტირებულ მოდელირებას აფართოებს სისტემის სერვისორიენტირებული ფუნქციონირების

აღწერით, როგორც მრავალაპლიკაციური, ისე ჰორიზონტალური და ვერტიკალური დაპროექტების ჭრილში.

Web-სერვისების გამოყენებით, მრავალაპლიკაციურ რეჟიმში ბიზნესპროცესების წარმოების ერთიანი ელექტრონული სისტემის ფუნქციონირების თვალსაჩინოებისთვის, განვიხილავთ ფინანსთა სამინისტროს შემოსავლების სამსახურისა და საფინანსო ბანკების ინფორმაციული კომპოზიციის მაგალითს. ამ ორი მასშტაბური სტრუქტურის ფუნქციონალური ურთიერთობა იკვეთება გადასახადის გადამხდელი კლიენტის ბანკში ანგარიშის გახსნის, დახურვის, ცვლილებების, საგადასახადო დავალებების ოპერაციების წარმოების დროს.

იმ შემთხვევაში თუ ბანკი კლიენტის ანგარიშზე ახდენს გახსნა/ დახურვა/ცვლილებების ოპერაციებს (გადასახადის გადამხდელი კლიენტის შემთხვევაში), ვალდებულია აცნობოს და შემდგომ მიიღოს ინფორმაცია ფინანსთა სამინისტროს შემოსავლების სამსახურიდან კლიენტის მიმდინარე ვალდებულების შესახებ. გამომდინარე მიღებული ინფორმაციიდან ბანკი ანიჭებს კლიენტის ანგარიშს შესაბამის სტატუსს, რის საფუძველზეც შესაძლებელია განხორციელდეს ან დარჩეს უცვლელად გახსნა/დახურვა/ ცვლილებების ოპერაცია.

შემოსავლების სამსახურის მხრიდან საბანკო სისტემასთან ურთიერთობა აისახება სხვადასხვა ტიპის საგადასახადო დავალებების ფინანსური ოპერაციების უზრუნველსაყოფად (მაგ., ყადაღა, ინკასო და სხვ.). პერიოდულად, შემოსავლების სამსახურიდან ეგზავნება ბანკს კლიენტის ფინანსური ვალდებულების ნოტიფიკაცია (დაკისრება ან განთავისუფლება), რის საფუძველზეც ბანკი ახდენს კლიენტის ანგარიშის დაბლოკვა/განბლოკვის ოპერაციებს.

მონაცემებისა და ნოტიფიკაციების გაცვლა ამ ორ სტრუქტურას შორის წარმოებს XML ენაზე. აპლიკაციებმა უნდა უზრუნველყოს როგორც ამ ინფორმაციის მიღება, ისე სინქრონიზაცია და შედეგებზე ასახვა, მაქსიმალურად ავტომატიზებულ რეჟიმში.

სერვისორიენტირებული მიდგომით, აღწერილი ბიზნეს-პროცესების სარეალიზაციოდ (ნახ.2.9–ბ), გამოყენებაშია ცხრ.2.1 სახის Web-სერვისები (იხ. თავი_1, $_2.1).

მნიშვნელოვანია, რომ რამდენიმე ოპერაციის (მაგალითად, კლიენტის ანგარიშის გახსნა, დახურვა, ცვლილება, ყადაღის დადება, ყადაღის მოხსნა, ინკასოს წარმოება) წარმოებისთვის, გამოყენებაშია ერთი ტიპის ბიზნესპროცესები (მაგალითად, setclientState, blockaccount, unblockaccount), რაც სერვის-ორიენტირებული მიდგომის გამოყენებით შესაძლოა გავაერთიანოთ ერთ Web-სერვისში.

ამავდროულად, უკვე რეალიზებული Web-სერვისი (მაგალითად, კლიენტის ანგარიშის მოდიფიკაცია), შესაძლებელია ასევე გამოვიყენოთ სხვა აპლიკაციის რეალიზაციის (კლიენტისთვის ინტერნეტ-ბანკინგის ჩართვა) ან სხვადასხვა აპლიკაციის ერთობლივი მუშაობის დროს (მაგალითად, ბანკის საბაჟო სტრუქტურასთან ჰორიზონტალური ინტეგრაცია).

დასასრულ შეიძლება დავასკვნათ, რომ ობიექტ-ორიენტირებული აპლიკაციების Web-სერვისებით ფორმირება განსაკუთრებით მოქნილი, მოსახერხებელი და გაცილებით საიმედოა როგორც ახალი პროგრამული პროდუქტების შექმნისას, ისე არსებულ სისტემაში ცვლილებების გატარებისას. Web-სერვისი, შეიძლება განვიხილოთ, როგორც ხიდი ობიექტ-ორიენტირებულ და პროცესორიენტირებულ ტექნოლოგიებს შორის, რაც კომპანიათაშორისი და კომპანიის მსხვილ

სტრუქტურათაშორისი საქმიანი პროცესების ინტეგრაციასა და მრავალაპლიკაციურ მართვას უზრუნველყოფს.

თუმცა, პროცეს-ორიენტირებული მიდგომის მრავალმხრივ რეკომენდირებული სერვისორიენტირებული არქიტექტურის გამოყენებას IT სპეციალისტებს შორის არაცალსახა შეფასება აქვს. პრაქტიკიდან გამომდინარე, სერვისორიენტირებულ არქიტექტუ-რას ახლავს რიგი სირთულეები, რაც უკავშირდება Web-სერვისების ტიპიზირებას და ტიპიზირებული Web-სერვისების მასშტაბურ რაოდენობას, არასაკმარის ინსრუმენტულ საშუალებებს, სტანდარტიზაციისა და ცენტრალიზაციის პრობლემების წარმოქმნას.

მიუხედავად ამ სირთულეებისა, აპლიკაციების ინტეგრაციის თვალსაზრისით სერვისორიენტირებული არქიტექტურა და პროცესორიენტირებული მიდგომა აქტუალური, მომქმედი და განვითარებადია, რასაც მხარს უჭერს Microsoft და Java ტექნოლოგიები.

2.3. Web-სერვისული ბიზნესპროცესების და სცენარების დაპროექტება UML-ის Activity და Sequence დიაგრამებით

კორპორაციული მენეჯმენტის კომპიუტერული სისტემების პროგრამული უზრუნველყოფის დასაპროექტებლად დიდი ყურადღება ექცევა დღეს მოდელირების უნიფიცირებული ენის (UML) ტექნოლოგიის გამოყენებას [32].

პროგრამული სისტემების შექმნა მის ცალკეულ ეტაპებზე ხორციელდება მოდელირების სხვადასხვა ტიპის დიაგრამებით, როგორიცაა გამოყენებით შემთხვევათა (UseCase), აქტიურობათა (Activity), მიმდევრობითობის (Sequence), კლასებისა (Class) და კლასთაშორისი (ClassAssotiation), მდგომარეობათა (State) და სხვა სახის დიაგრამები.

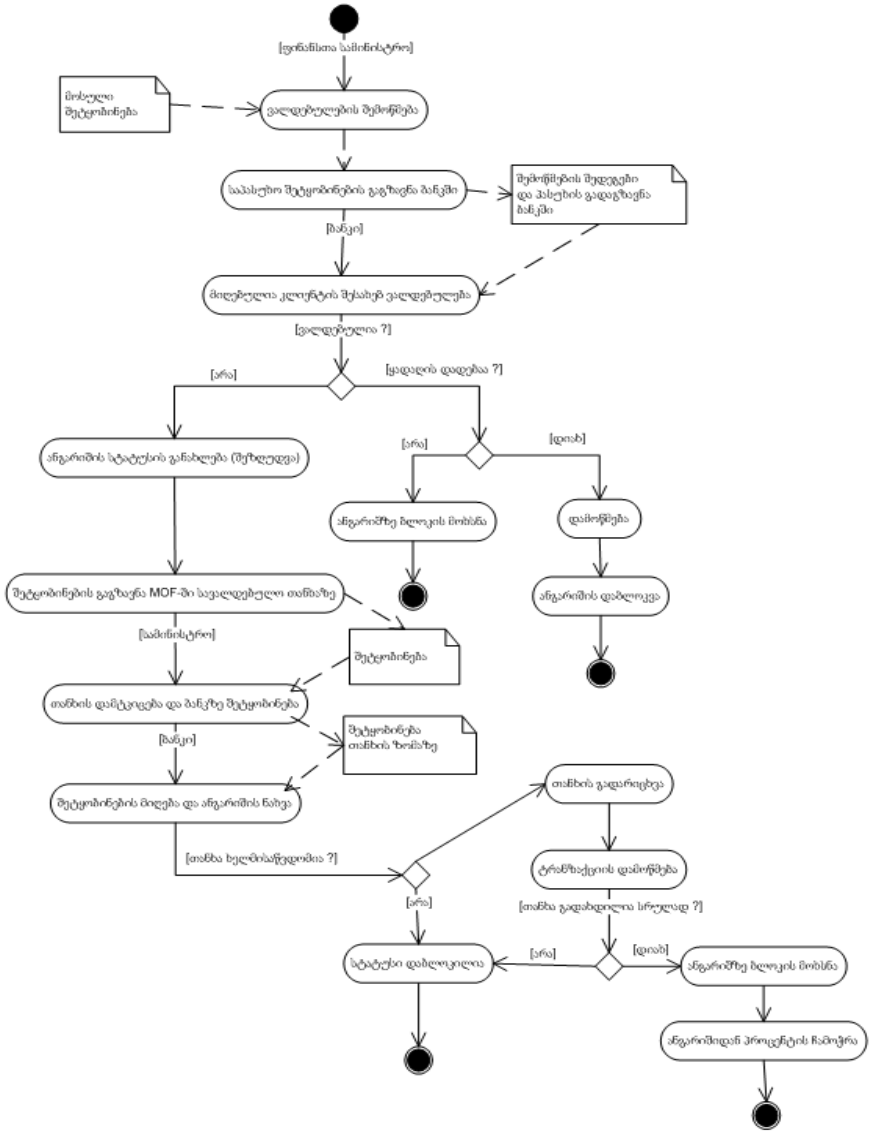
ჩვენ წარმოდგენილი გვაქვს ამოცანა განაწილებული კორპორაციული ობიექტების (მაგალითად, საფინანსო ბანკები, ფინანსთა სამინისტროს შემოსავლების სამსახური და ა.შ.) ბიზნესპროცესების დასაპროექტებლად უნიფიცირებული მოდელირების ენის (UML) გამოყენებით და კომპიუტერზე დასაპროგრამებლად ობიექტორიენტირებული და სერვისორიენტირებული პრინციპებით.

კონკრეტულად ჩვენ განვიხილავთ საფინანსო-საგადასახადო სისტემის ფუნქციურ ამოცანებს შემოსავლების სამსახურისა და საფინანსო ბანკების ელექტრონული სისტემების (აპლიკაციების) არსებობის პირობებში.

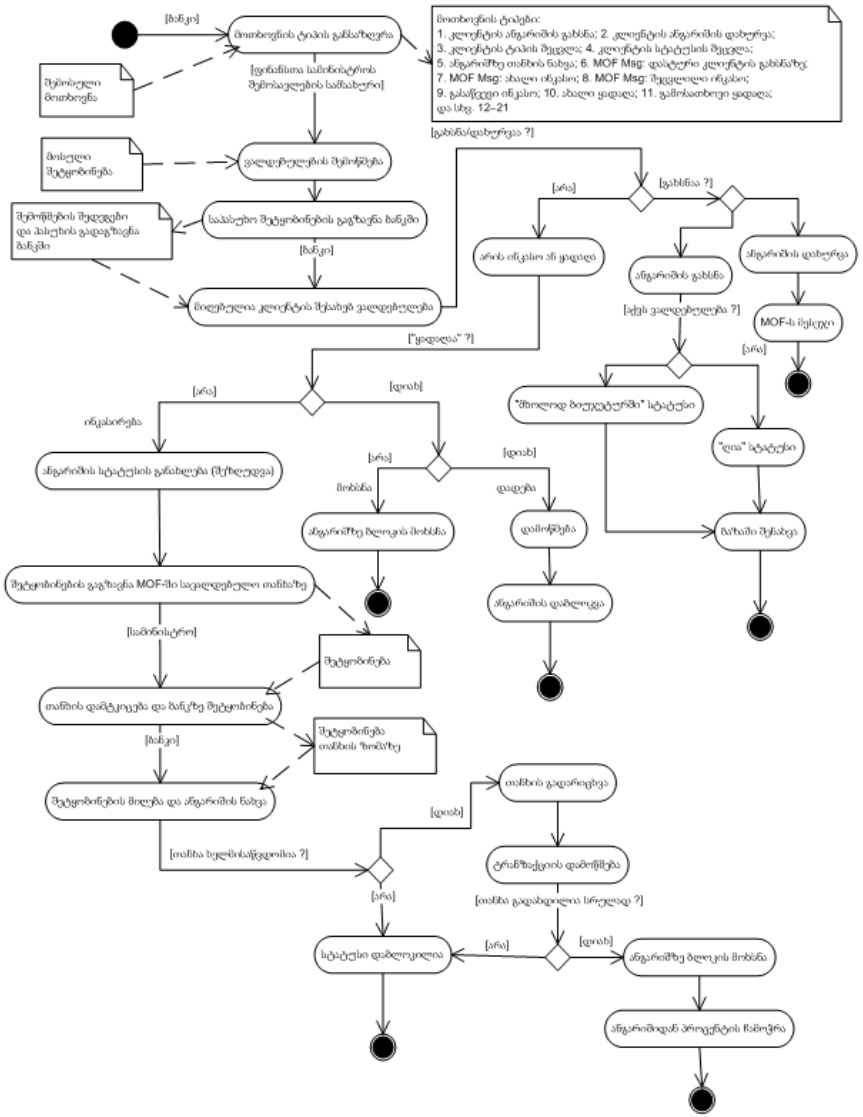
წინა პარაგრაფში ჩვენ მიერ აღწერილი იყო ასეთი ამოცანების გადაწყვეტის ბიზნესპროცესები BPMN მოდელირების ინსტრუმენტის საშუალებით (ნახ.2.1 - 2.9). ახლა განვიხილოთ ზოგიერთი აღნიშნული სერვის-პროცესის აქტიურობათა დიაგრამის და ავტომატიზებულ რეჟიმში დასმული ამოცანის შესრულების სცენარების, ანუ მიმდევრობითობის დიაგრამების აგება UML-ტექნოლოგიით.

2.10–ა,ბ ნახაზებზე მოცემულია BPMN მოდელის (ნახ.9–ბ) ბიზნესპროცესების ასახვის შედეგები UML-ის აქტიურობის დიაგრამებით (ორი ვარიანტი). მე–2 ვარიანტი მიღებულია პირველის გაფართოებით, ახალი დაზუსტებული ინფორმაციის შესაბამისი პროცედურების ინტეგრაციის საფუძველზე არსებულთან.

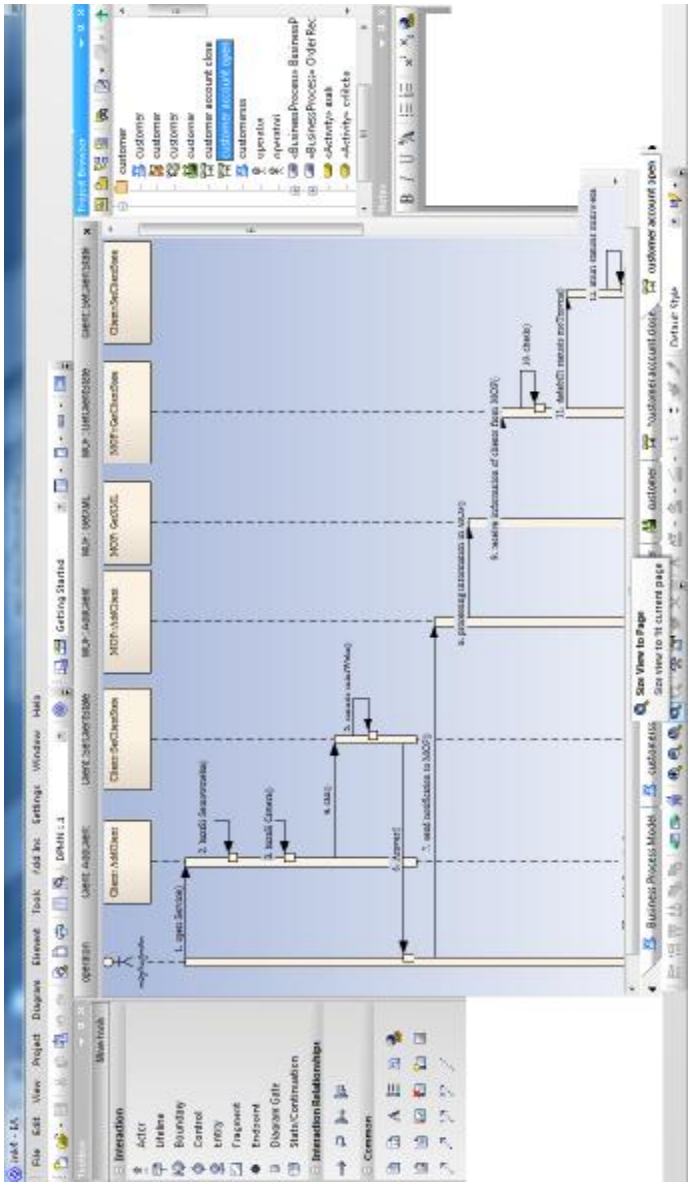
2.11–2.12 ნახაზებზე წარმოდგენილია ბანკში კლიენტების ანგარიშების გახსნის და დახურვის სერვისული ფუნქციების სცენარების მიმდევრობითობის დიაგრამები. ასეთივე წესით მოხდება დანარჩენი სერვის–ფუნქციების მოდელირება ინტერაქტიული დიაგრამებით.



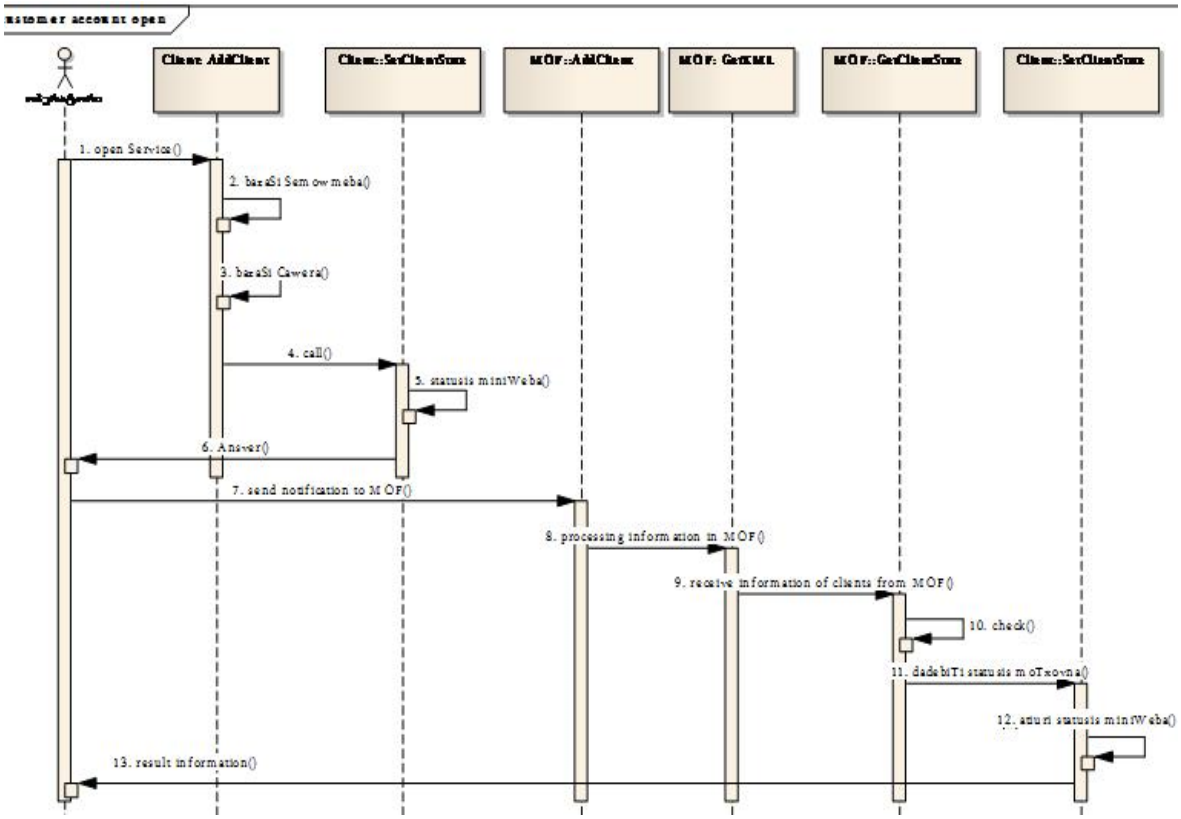
ნახ.2.10–ა. Activity-D: კლიენტის რეგისტრაცია და ინკასო (ვარიანტი_1)



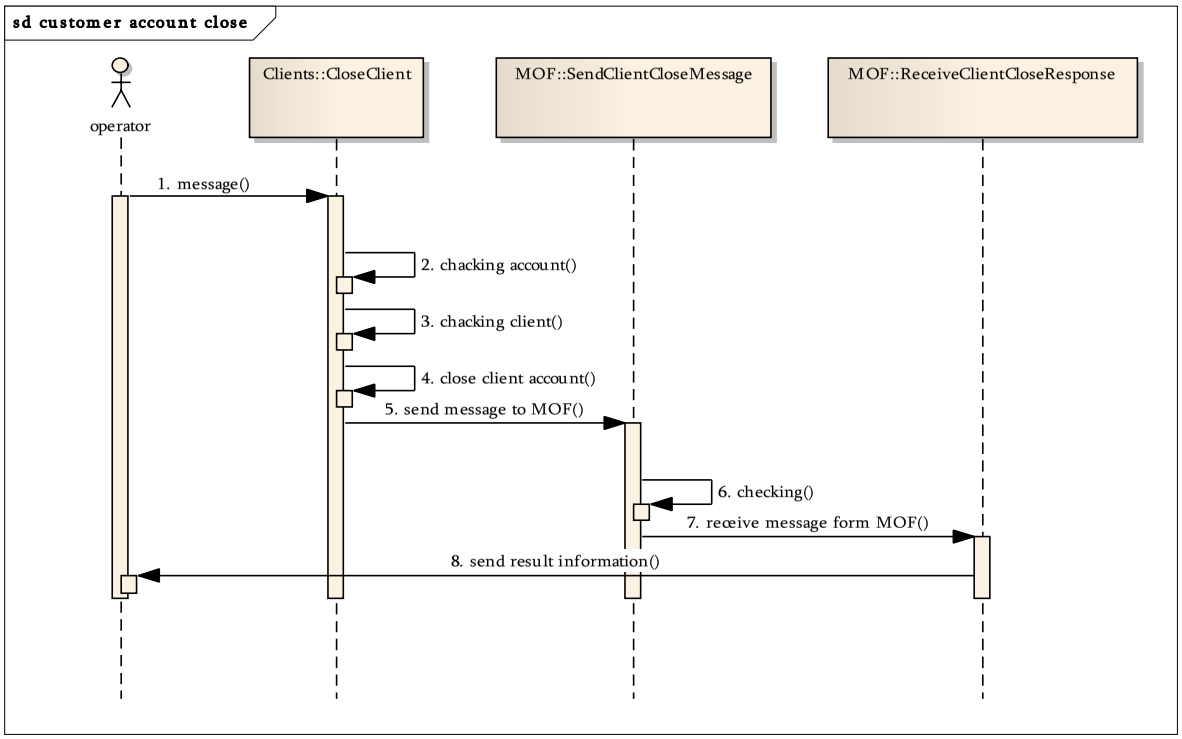
ნახ.2.10–ბ. Activity-D: კლიენტის რეგისტრაცია და ინკასო (ვარიანტი_2, დაზუსტებული მოდელი)



ნახ.2.11-ა. Enterprise Architect ინსტრუმენტის გარემო და Sequence-დიაგრამა



ნახ.2.11-ბ. მიმდევრობითობის დიაგრამა: customer account open



ნახ.2.12. მიმდევრობითობის დიაგრამა: customer account close

ჩვენი მიზანია შემდგომში გამოვიკვლიოთ ალტერნატიული აქტიურობის დიაგრამების დინამიკური პროცესები, მათი შესრულების დროითი მახასიათებლების ანალიზით. ამისათვის მომდევნო თავში განვიხილავთ დინამიკური პროცესების მოდელირებისა და ანალიზის ისეთ ინსტრუმენტს, როგორიცაა პეტრის ქსელები. ჩვენ გამოვიყენებთ სტოქასტიკურ, დროით პეტრის ქსელებს.

III თავი: მონაცემთა ბაზების დაპროექტების ობიექტ-როლური ტექნოლოგია

3.1. ობიექტ-როლური მოდელირება (ORM) განაწილებული სისტემებისთვის

კორპორაციის განაწილებული ინფორმაციული სისტემების ფუნქციონალობის განსაზღვრა კონცეპტუალურ დონეზე ხდება, სადაც გამოიყენება ისეთი კონცეფციები და ენა რომელიც ადვილი გასაგებია ადამიანისათვის [36]. მონაცემთა ბაზის დაპროექტება მოიცავს საპრობლემო არის ფორმალური მოდელის აგებას ანუ მისი აღწერის (საუბრის) ფორმირებას *(universe of discourse Uod)* სათანადოდ ამის გაკეთება კი დამოკიდებულია *Uod*-ის კარგ ცოდნაზე. ობიექტ-როლური მოდელირება (ORM) ამარტივებს დაპროექტების პროცესს, იყენებს რა ბუნებრივ, სალაპარაკო ენას, ასევე ინტუიციურ დიაგრამებს, რომელთა შევსებაც შეიძლება მაგალითების საშუალებით და შესაძლებელია ინფორმაციის შემოწმება მარტივ, ელემენტარულ ფაქტებზე დაყრდნობით. ვინაიდან მოდელი გამოსახულია ისეთ ბუნებრივ ტერმინებში, როგორიცაა ობიექტი და როლი, იგი უზრუნველყოფს მოდელირების კონცეპტუალურ მიდგომას [49,50].

ობიექტ-როლური მოდელირების ადრეული ვერსია 1970-იან წლებში გამოჩნდა ევროპაში. აქ აღწერილი მეთოდი დამუშავებულია ავსტრალიასა და აშშ-ში. ასოციაციური ენა *FORML (Formal Object-Role Modeling Language)* თანმხლებია *Microsoft Visio for Enterprise Architect* (*VEA*) პაკეტის, რომელიც *Visual Studio.NET*-ის შემადგენელი ნაწილია.

კონცეპტუალური მოდელირება მიიღწევა არსთა დამოკიდებულების *(ER)* მოდელითაც, თუმცა (*ER*) მოდელი შეიძლება გამოვიყენოთ მას შემდეგ, რაც დაპროექტების პროცესი

დამთავრებულია. იგი ნაკლებადაა შესაფერისი ფორმულირებისათვის, განახლებასა და პროექტის შემდგომი გაფართოებისათვის. *ER*-დიაგრამა შორსაა ბუნებრივი ენისაგან, ვერ ხერხდება შევსება ამა თუ იმ მოვლენის ფაქტით, დამალულია ინფორმაცია იმ სემანტიკური დომენების შესახებ, რომლებიც ქმნის მოდელს.

ამრიგად, კონცეპტუალური მოდელირების განვითარებულ ტექნიკას წარმოადგენს ობიექტ-როლური მოდელირება. სწორედ *ORM*-ს შეუძლია უზრუნველყოს სხვადასხვა პროფესიის ადამიანების შეთანხმებული მუშაობა, რომელთა მომზადების დონე ინფორმაციული სისტემების დაპროექტების სფეროში შეიძლება მნიშვნელოვად განსხვავდებოდეს.

ზემოაღწერილი მიზეზებიდან გამომდინარე, კონცეპტუალური მოდელირებისათვის ჩვენ ვირჩევთ *ORM*-ს. საინფორმაციო სისტემების სასიცოცხლო ციკლი მოიცავს რამდენიმე სტადიას: ტექნიკურ-ეკონომიკური დასაბუთება, მოთხოვნათა ანალიზი, მონაცემებისა და ოპერაციების კონცეპტუალური დაპროექტება; ლოგიკური დაპროექტება; გარე დაპროექტება; მაკეტირება; შიგა დაპროექტება და იმპლემენტაცია; ტესტირება და შესწორების შეტანა; მომსახურება (თანხლება).

ORM-ის კონცეპტუალური მოდელირების სქემის პროცედურა (*CSDP*) ყურადღებას ამახვილებს მონაცემების ანალიზზე და დაპროექტებაზე. კონცეპტუალური სქემა აღწერს აპლიკაციის ინფორმაციულ სტრუქტურას: ფაქტების ტიპები, რომლებიც წარმოადგენს ინტერესის სფეროს; მასზე არსებული შეზღუდვები და შესაძლოა წარმოქმნის წესები, რათა მივიღოთ ესა თუ ის ფაქტი სხვა ფაქტებიდან.

მრავალმასშტაბიანი აპლიკაციისათვის Uod-ი ყოფს მას მოხერხებულ მოდულებად, *CSDP*-ს მიმართავს თითოეული მათგანი და შემდეგ ხდება მიღებული კონცეპტუალური

ქვესქემების გაერთიანება ერთ გლობალურ კონცეპტუალურ სქემად. თვითონ *CSDP*-ი შედგება შვიდი ბიჯისაგან. იხ. ცხრ.3.1.

ცხრ.3.1

1. ელემენტარული ფაქტების ფორმირება და მათი ადექვატურობის შემოწმება;
2. ფაქტების ტიპებისათვის დიაგრამის აგება და სისრულის შემოწმება;
3. იმ ობიექტთა ტიპების შემოწმება, რომლებიც უნდა გაერთანდეს და მათი მათემატიკური წარმომავლობის დაფიქსირება;
4. დაემატოს უნიკალურობის შეზღუდვა და შემოწმდეს ფაქტების ტიპების ოპერანდების რაოდენობა;
5. დაემატოს როლების იძულებითი შეზღუდვები და შემოწმდეს მათი ლოგიკური წარმომავლობა;
6. დაემატოს ელემენტები, სიმრავლეთა შედარება და ქვეტიპის შეზღუდვები;
7. დაემატოს სხვა შეზღუდვები და მოხდეს საბოლოო შემოწმება.

ბიჯი_1: *CSDP*-ს ყველაზე მნიშვნელოვანი სტადიაა, სადაც ხდება სხვადასხვა სახის ინფორმაციის შეგროვება, ბუნებრივ სალაპარაკო ენაზე. ასეთი ინფორმაცია ხშირად არის შემავალი და გამომავალი ფორმები, შეიძლება იყოს ხელნაწერი. წინააღმდეგ შემთხვევაში მოდელის დამპროექტებელი მუშაობს უშუალოდ კლიენტთან, რათა ზუსტად ჩამოყალიბდეს, თუ რა მოეთხოვება სისტემას. აუცილებელია ექსპერტის არსებობა, რომელიც იცნობს აპლიკაციას.

ეს ფაქტი ასე შეიძლება ჩაიწეროს:

- **f1 თანამშრომელს, ნომრით 35, აქვს გვარი 'ქართველიშვილი'**
- **f2 თანამშრომელი, ნომრით 7, მუშაობს კონტრაქტით თარიღამდე '12.31.12'**

თითოეული ფაქტი არის ბინარული დამოკიდებულება ორ ობიექტს შორის. მუქი შრიფტით გამოყოფილია ლოგიკური პრედიკატი, რომელიც ახდენს ობიექტების იდენტიფიცირებას

ნაჩვენებია კურსივით. იმ შემთხვევაში თუ განისაზღვრება ობიექტის მხოლოდ ერთი თვისება, საქმე გვაქვს ერთადგილიან პრედიკატთან (unary fact). პრედიკატს შეიძლება ჰქონდეს (1,2,3,..) ოპერანდი, თუმცა რადგან პრედიკატი ელემენტარულია 3-4 ოპერანდზე მეტი იშვიათად გვხვდება. უმრავლეს შემთხვევაში პრედიკატი არის ორობითი. ასეთი პრედიკატებისათვის არსებობს ინვერსული პრედიკატი. ისე, რომ ფაქტი შეიძლება წავიკითხოთ ორივე მიმართულებით.

ბიჯი_2: აქ ხდება ფაქტების ტიპებისათვის დიაგრამის აგება. ობიექტები გამოისახება ელიფსებით, პრედიკატები მართკუთხედებით, მნიშვნელობის ტიპი გამოისახება წყვეტილი ელიფსით. პრედიკატი იკითხება მარცხნიდან-მარჯვნივ და ზემოდან-ქვემოთ, მანამ, სანამ არ შეხვდება ნიშანი "<<", რომელიც ცვლის წაკითხვის მიმართულებას საწინააღმდეგო მიმართულებით. შემდეგ ბიჯებზე ხდება შეზღუდვების დაწესება.

ORM დიაგრამაში გამოყენებული შეზღუდვები [51]:

<u>იძულების შეზღუდვები:</u>

- ობიექტი ასრულებს ზუსტად გარკვეულ როლს.

როლების დიზუნქცია არის იძულებითი. თითოეული ობიექტი ობიექტთა ტიპების ნაკრებიდან უნდა ასრულებდეს მხოლოდ ერთ როლს.

<u>უნიკალურობის შეზღუდვები:</u>

ერთ ან მეტ როლში მონაწილეობა ხდება არა უმეტეს ერთხელ.

როლების გარე უნიკალურობის შეზღუდვა. წყვილის გამორიცხვის შეზღუდვა.

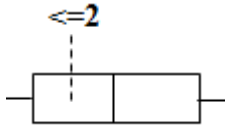
<u>სიმრავლეების შედარების შეზღუდვები:</u>

⊆ პირველი ობიექტის სიმრავლე ყოველთვის უნდა იყოს მეორის ქვესიმრავლე.

= პირველი ობიექტის სიმრავლე ყოველთვის უნდა იყოს მეორის ტოლი.

⊄ პირველი ობიექტის სიმრავლე არ შედის მეორეში.

სიხშირის შეზღუდვა.



ობიექტმა რამდენჯერ შეიძლება შეასრულოს ეს როლი

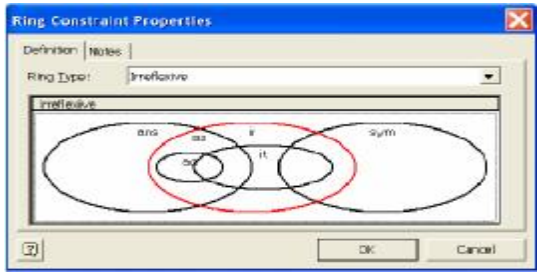
ბუდის ტიპის ობიექტი.

ობიექტი თამაშობს მხოლოდ ერთ როლს და ეს როლი არ არის სავალდებულო.

ქვეტიპი.

ერთი ობიექტი არის მეორის ქვეტიპი

წრიული შეზღუდვები



ანტირეფლექსურობა	Oir	iff for all x, ~xRx
სიმეტრიულობა	Osym	iff for all x, y, xRy → yRx
ასიმეტრიულობა	Oas	iff for all x, y, xRy → ~yRx
ანტისიმეტრიულობა	Oans	iff for all x, y, x ≠ y & xRy → ~yRx
ანტიტრანზიტულობა	Oit	iff for all x, y, z, xRy & yRz → ~xRz
აციკლურობა	Oac	iff for all x, y, z, xRy & yRz → ~zR

ახლა განვიხილოთ განაწილებული კორპორაციული სისტემებისთვის მონაცემთა ბაზების ობიექტორიენტირებული მოდელირებისა და ობიექტორიენტირებული დაპროექტების ავტომატიზებული პროცესების დამუშავების ამოცანა, რაც საგრძნობლად ამცირებს სისტემების ინფორმაციული და პროგრამული პაკეტების აგების დროს და, ამასთანავე ორიენტირებულია გამოყენებითი სფეროს მომხმარებელზე [49].

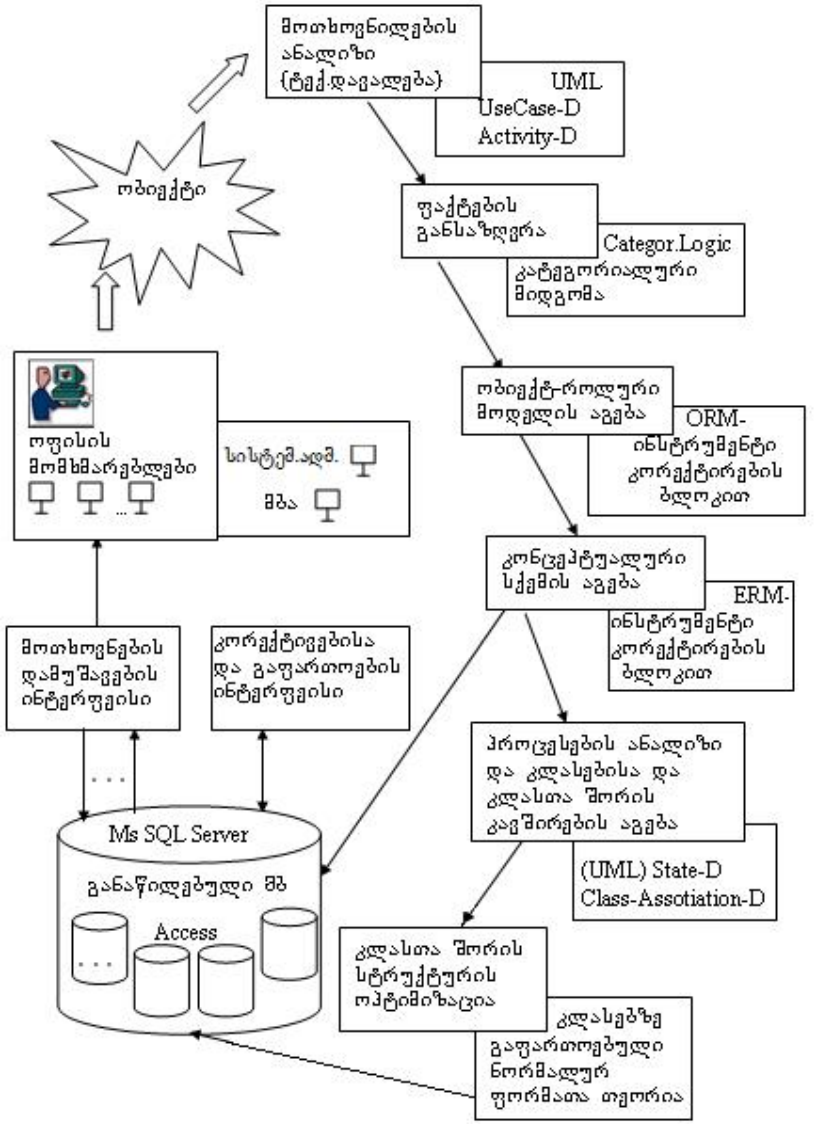
3.1 ნახაზზე მოცემულია საპრობლემო სფეროს მოდელირებისა და მონაცემთა ბაზაში ავტომატიზებული ასახვის ამოცანის ძირთადი ეტაპებისა და მათი რეალიზაციის ინსტრუმენტული საშუალებების სქემა.

ცოდნა, რომელიც საპრობლემო სფეროს შესახებ გააჩნია მომხმარებელს, სპეციალური ინტერფეისების საშუალებით, რომელთა საფუძველს ფორმალური ენის გრამატიკის კატეგორიები და ლოგიკურ-ალგებრული მეთოდები შეადგენს, გადაეცემა ობიექტ-როლური მოდელირების კომპიუტერულ პროგრამას.

მისი დახმარებით აიგება სემანტიკური სტრუქტურები *ORM*–დიაგრამათა სახით. შემდგომ ეტაპზე ავტომატიზებული პროცედურების გამოყენებით ფორმირდება არსთა-დამოკიდებულების მოდელი, ანუ *ER*-დიაგრამები, ცხრილებითა და ატრიბუტებით. მის საფუძველზე ავტომატურად გენერირდება *.DDL* ფაილები, რომლებიც შემდგომ სტრუქტურულად მოთავსდება *Ms SQL Server* მონაცემთა განაწილებულ ბაზაში.

როგორც აღვნიშნეთ, აქ პრობლემურად ისმება, ER-მოდელის შესაბამისი კლასების დიაგრამის "სტრუქტურის ოპტიმიზაციის" საკითხი ანუ მონაცემთა ბაზების დაპროექტების თეორიის ტერმინით - ნორმალიზაციის ამოცანა.

აქ იგულისხმება შემდეგი: უნდა განისაზღვროს თითოეული კლასის მონაცემთა ოპტიმალური სტრუქტურა (მონაცემთა მოდელი), კლასთა შორის კავშირების გათვალისწინებით და დამოკიდებულებათა რელაციების ნორმალიზაციის მეთოდების გამოყენებით.



ნახ.3.1

ოპტიმიზაციის კრიტერიუმად შეიძლება განვიხილოთ რელაციური ცხრილების ინფორმაციული და ინდექსური მონაცემების მოცულობათა მინიმიზაცია, და მათთან მიმართვისა და განახლების დროის შემცირება. მონაცემთა ერთიანი ლოგიკური სტრუქტურების მთლიანობის უზრუნველყოფით (დაცვით).

ნაშრომი ეფუძნება *UML*–ტექნოლოგიისა და მონაცემთა ლოგიკური სტრუქტურების რელაციურ დამოკიდებულებათა თეორიის კლასიკურ საკითხებს. ასეთი ინტეგრირებული მიდგომა დასმული ამოცანის გადასაწყვეტად უზრუნველყოფს, ერთის მხრივ, მონაცემთა სტრუქტურების ოპტიმიზაციას ნორმალურ ფორმათა თეორიის საფუძველზე და, მეორეს მხრივ, დაპროექტების ობიექტორიენტირებული პრინციპების მხარდაჭერას *Case*-ტექნოლოგიების გამოყენებით (დაპროგრამების გუნდური ტექნოლოგიები), რაც აქტუალური და მნიშვნელოვანი ამოცანაა.

საინფორმაციო სისტემების სასიცოცხლო ციკლი, როგორც ცნობილია მოიცავს რამდენიმე სტადიას: ტექნიკურ-ეკონომიკური დასაბუთება, მოთხოვნათა ანალიზი, მონაცემებისა და ოპერაციების კონცეპტუალური დაპროექტება; ლოგიკური დაპროექტება; გარე დაპროექტება; მაკეტირება; შიდა დაპროექტება და შესრულება; ტესტირება და შესწორების შეტანა; მომსახურება (თანხლება).

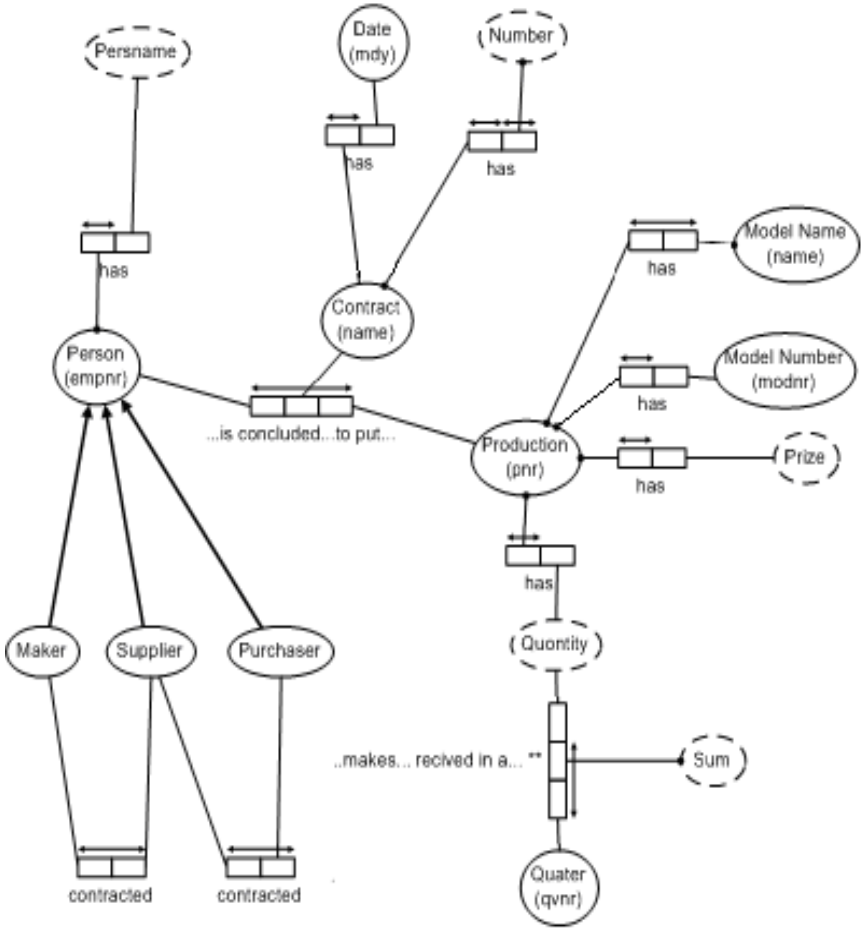
ORM-ის კონცეპტუალური მოდელირების სქემის პროცედურა *(CSDP)* ყურადღებას ამახვილებდა მონაცემთა ანალიზზე და დაპროექტებაზე. კონცეპტუალური სქემა აღწერს აპლიკაციის ინფორმაციულ სტრუქტურას: ფაქტების ტიპები, რომლებიც წარმოადგენს ინტერესის სფეროს; მასზე არსებული შეზღუდვები და შესაძლოა წარმოქმნის წესები, რათა მივიღოთ ესა თუ ის ფაქტი სხვა ფაქტებიდან. ახლა განვიხილოთ მაგალითები.

მაგალითის სახით განვიხილოთ კორპორაციული სისტემების ერთ-ერთი მნიშვნელოვანი ამოცანა – "კონტრაქტების გაფორმება" სერვისის (ან პროდუქციის) დამკვეთ და მიმწოდებელ ორგანიზაციებს შორის.

ამ ამოცანის შესაბამისი *ORM*-დიაგრამის აგების პროცესში, პირველ რიგში, "კონტრაქტის დადების" პროცედურის ტექსტური (შინაარსობრივი) ფორმიდან უნდა შევარჩიოთ და ჩამოვაყალიბოთ ფაქტების ერთობლიობა. მაგალითად, ოფის-ობიექტზე "კონტრაქტის დადების" ამოცანისთვის გვექნება შემდეგი ფაქტები:

f1 - პერსონას (სუბიექტს) აქვს სახელი;
f2 - დამამზადებელი არის სუბიექტი;
f3 - მიმწოდებელი არის სუბიექტი;
f4 - მყიდველი არის სუბიექტი;
f5 - სუბიექტმა დადო კონტრაქტი პროდუქციის შესყიდვაზე;
f6 - კონტრაქტს აქვს ნომერი;
f7 - კონტრაქტს აქვს სახელი;
f8- კონტრაქტს აქვს თარიღი;
f9 - სერვისს (პროდუქციას) აქვს მოდელის დასახელება;
f10 - სერვისს (პროდუქციას) აქვს მოდელის ფასი;
f11 - სერვისს (პროდუქციას) აქვს მოდელის ნომერი;
f12 - სერვისს (პროდუქციას) აქვს რაოდენობა;
f13 - სერვისი (პროდუქცია) გარკვეული რაოდენობით და გარკვეული ფასით გაიყიდა კვარტალში;
f14 - თანხა არის ფასი, გამრავლებული რაოდენობაზე.

ORM-დიაგრამაზე გამოსათვლელი ფაქტის ტიპი აღინიშნება " * "-ით. ოფის-ობიექტზე კონტრაქტის დადების *ORM*-დიაგრამა ნაჩვენებია 3.2 ნახაზზე.

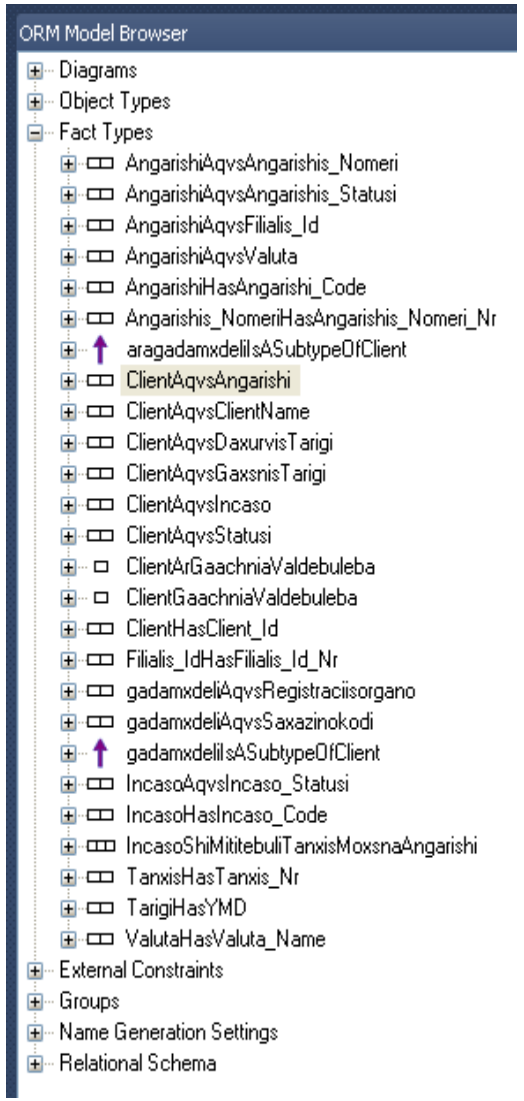


ნახ.3.2. "კონტრაქტის" *ORM*-მოდელის ფრაგმენტი

3.2. ინტერკორპორაციული სერვისორიენტირებული სისტემის მონაცემთა ბაზების დაპროექტება და რეალიზაცია

ახლა კონკრეტულ ამოცანაზე განვიხილოთ კორპორაციული მართვის საინფორმაციო სისტემის მონაცემთა ბაზების დამუშავების საკითხი ზემოაღწერილი მეთოდოლოგიის მიხედვით. კერძოდ, კონცეპტუალური მოდელირების ახალი ტექნოლოგიის სახით ვიყენებთ ობიექტ-როლურ მოდელირებას (ORM) [7,36]. იგი პრაქტიკულად სემანტიკური მოდელირების ინსტრუმენტია ფაქტებზე დაყრდნობით. ბუნებრივი ენის და ინტუიციური დიაგრამების (რომელთა შევსებაც ხდება მაგალითებით) გამოყენება და ასევე საპრობლემო სფეროს აღწერა ელემენტარული ფაქტების საფუძველზე, საგრძნობლად ამარტივებს მოდელის აგების და დაპროექტების პროცესს. კონცეპტულურის მოდელის ავტომატური დაპროექტებისათვის წარმოდგენილია ობიექტ-როლური მოდელირების მეთოდი (ORM), Natural ORM Architect მოდელირების ინსტრუმენტით [51].

თავდაპირველად ხდება საპრობლემო არის, ანუ კორპორაციათაშორის არსებული სისტემის მოთხოვნილებათა ანალიზი და ტექნიკური დავალების განსაზღვრა. არსებული ბიზნესპროცესებისა და ბიზნესწესების საფუძველზე ჩამოყალიბდება ფაქტები, ანუ იმ კანონზომიერებათა ერთობლიობა, რომელიც უნდა აისახოს მონაცემთა ბაზის მოდელში (ნახ.3.3). ამგვარად, მიღებული სემანტიკური ფაქტების სიმრავლის ფიზიკური გადატანით ავტომატიზებული დაპროექტების Natural ORM Architect (NORMA) ინსტრუმენტით Ms Visual Studio.NET გარემოში, განისაზღვრება ჩვენი სისტემის ობიექტ-როლური მოდელი, რომლის მიხედვითაც შემდგომ აიგება ERM-დიაგრამა.



ნახ.3.3. პრედიკატები საწყისი ფაქტების საფუძველზე NORMA ინსტრუმენტის გარემოში

საფინანსო ბანკების და შემოსავლების სამსახურის კორპორაციის ურთიერთობების საპრობლემო სფერო აღიწერება, მაგალითად, შემდეგი სახის ინფორმაციით: „ბანკში კლიენტის რეგისტრაცია ხდება ოპერატორის მიერ, "კლიენტების მართვის" პროგრამის საშუალებით. ახალ კლიენტს ენიჭება სტატუსი „მხოლოდ ბიუჯეტური". ამ კლიენტის შესახებ შეტყობინება ეგზავნება შემოსავლების სამსახურს. შემოსავლების სამსახური ამოწმებს კლიენტის მონაცემებს. კერძოდ, განსაზღვრავს აქვს თუ არა ამ კლიენტს რაიმე ვალდებულება (დავალიანება ბიუჯეტის მიმართ ან სხვ.) და შემდეგ ბანკს უგზავნის შესაბამის საპასუხო შეტყობინებას. ამ შეტყობინების შესაბამისად ხდება კლიენტის სტატუსის შეცვლა „აქტიურზე" (თუ არ აქვს ვალდებულება).

კლიენტის ანგარიშებზე ინკასოს (ან ყადაღას) წარმოდეგნისთვის შემოსავლების სამსახური აფორმირებს შეტყობინებას, რომელიც შეიცავს კლიენტის მონაცემებს და თანხის ოდენობას. ეს შეტყობინება ეგზავნება ბანკის ვებ-სერვისს. ოპერატორი ინკასოების (ან ყადაღების) მართვის მოდულში ნახულობს შემოსული ინკასოების სიას და ამოწმებს მათ. დადასტურებულ ინკასოებზე Windows სერვისი კლიენტის ანგარიშებს ანიჭებს „მხოლოდ ბიუჯეტურ" სტატუსს და ა.შ.

ელემენტარული ფაქტების ერთობლიობა შეიძლება ასე წარმოვადგინოთ:

f1- კლიენტს აქვს გვარი;

f2 - კლიენტს აქვს ანგარიში ;

f3-კლიენტს აქვს ტიპი;

f4-ანგარიშს აქვს ანგარიშის ნომერი ;

f5-ანგარიშს აქვს ვალუტა;

f6-კლიენტს აქვს ინკასო; f7-ინკასოს აქვს ინკასოს სტატუსი;

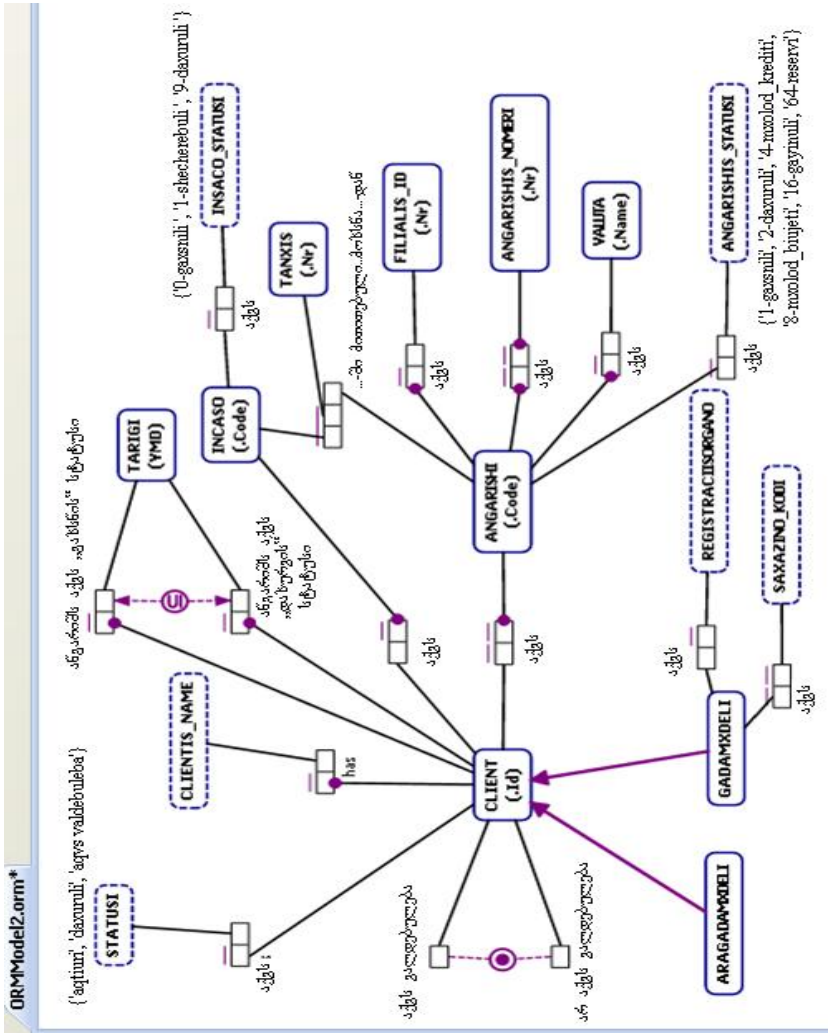
f8-ანგარიშებს აქვს ანგარიშების სტატუსი;

f9-ანგარიშს აქვს ფილიალი;

f10-კლიენტს აქვს სტატუსი;

f11-ანგარიშიდან ხდება ინკასოში მითითებული თანხის მოხსნა და ა.შ.

ამ ფაქტების საფუძველზე NORMA პროგრამული ინსტრუმენტით აგებული შესაბამისი ORM-დიაგრამა წარმოდგენილია 3.4 ნახაზზე.



ნახ.3.4. ORM-მოდელის ფრაგმენტი

ORM–დიაგრამაზე ობიექტები გამოსახულია ელიფსებში, წყვეტილი ელიფსი აღნიშნავს მნიშვნელობას (value), პრედიკატები გამოსახულია მართკუთხედებში. გამოყენებულია ბინარული და ტერნარული პრედიკატები. შეზღუდვები ნახაზზე გამოსახულია სპეც-აღნიშვნებით.

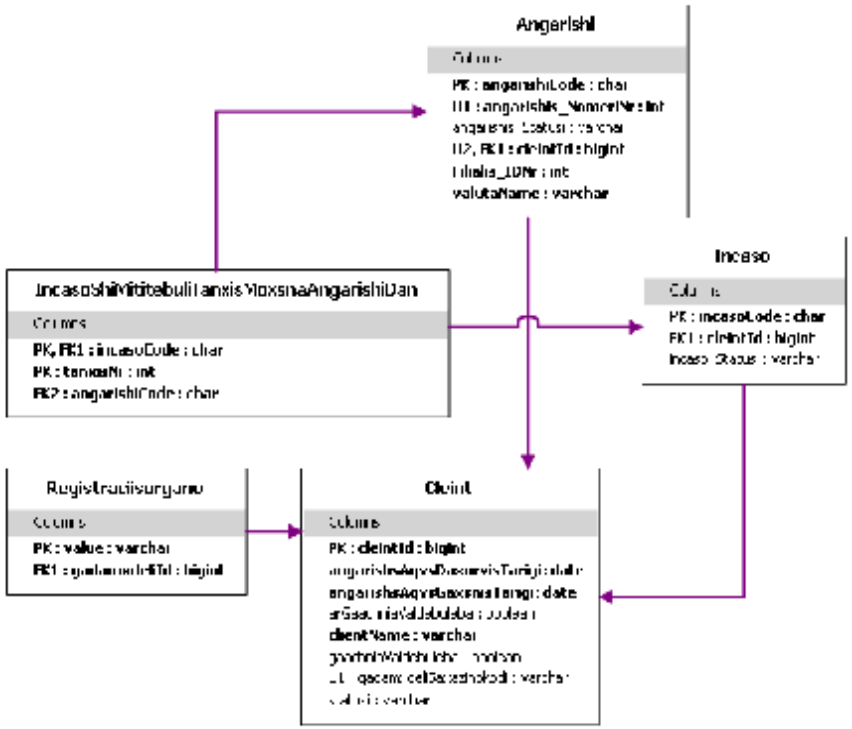
ჩვენ მიერ პროგრამულ ინსტრუმენტში შეტანილ ფაქტებს ობიექტებისა და მათი სემანტიკური კავშირების შესახებ აქვს 3.2 ცხრილში ნაჩვენები სახე.

ცხრ.3.2

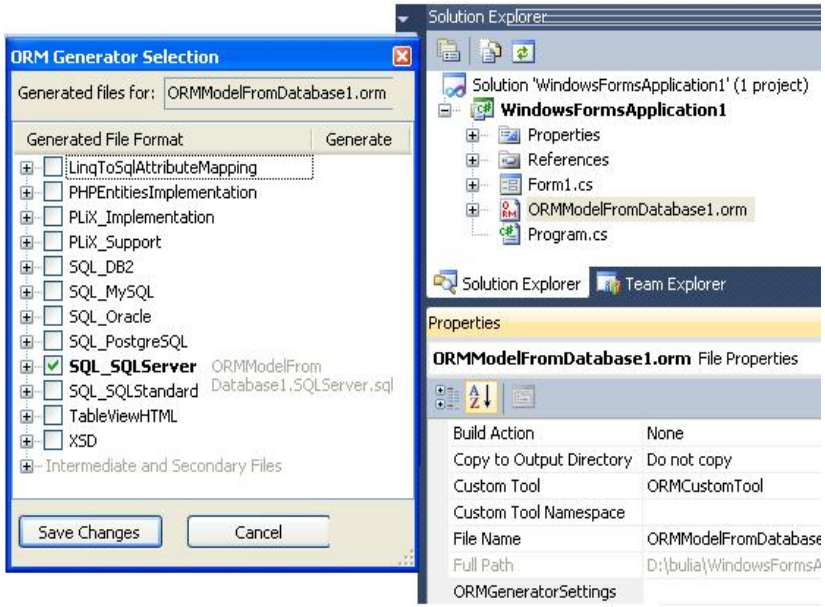
Fact Types: Client aqvs ClientName. **Each** Client aqvs **exactly one** ClientName. **It is possible that more than one** Client aqvs **the same** ClientName. Gadamxdeli **is an entity type**. **Reference Scheme:** Client has Client_Id. **Reference Mode:** .Id.	**Fact Types:** Gadamxdeli aqvs Registraciisorgano. **For each** Registraciisorgano, **at most one** gadamxdeli aqvs **that** Registraciisorgano. **It is possible that the same** gadamxdeli aqvs **more than one** Registraciisorgano. Tarigi **is an entity type**. **Reference Scheme:** Tarigi has YMD. **Reference Mode:** YMD.
Fact Types: **Each** gadamxdeli is an instance of Client. Gadamxdeli aqvs Saxazinokodi. Gadamxdeli aqvs Registraciisorgano. Aragadamxdeli **is an entity type**. **Reference Scheme:** Client has Client_Id. **Reference Mode:** .Id.	**Fact Types:** Gadamxdeli aqvs Saxazinokodi. **Each** gadamxdeli aqvs **at most one** Saxazinokodi. **For each** Saxazinokodi, **at most one** gadamxdeli aqvs **that** Saxazinokodi. Registraciisorgano **is a value type**. **Portable data type:** Text: Variable Length.
Fact Types: **Each** aragadamxdeli is an instance of Client. Saxazinokodi **is a value type**. **Portable data type:** Text: Variable Length.	**Fact Types:** Filialis_Id has Filialis_Id_Nr. Angarishi aqvs Filialis_Id. Angarishi aqvs Filialis_Id. **Each** Angarishi aqvs **exactly one** Filialis_Id. **It is possible that more than one** Angarishi aqvs **the same** Filialis_Id. Angarishis_Nomeri **is an entity type**. **Reference Scheme:** Angarishis_Nomeri has Angarishis_Nomeri_Nr. **Reference Mode:** .Nr.
და ა.შ.	

MsVisualStudio.Net გარემოში NORMA პროგრამული პაკეტის საშუალებით ORM-დიაგრამიდან ავტომატურად ვღებულობთ ERM არსთა დამოკიდებულებათა მოდელს (ნახ.3.5).

შემდეგ ბიჯზე ხორციელდება ERM–დიაგრამიდან ავტომატიზებულ რეჟიმში MsSQL Server მონაცემთა ბაზის DDL–ფაილის გენერირება. ამისათვის Ms Visual Studio.NET გარემოში, Solution Explorer და Properties–ში აირჩევა ORM-GeneratorSetting. 3.6 ნახაზზე ნაჩვენებია ეს პროცედურა MsSQL Server ბაზის ასარჩევად.



ნახ.3.5. ERM-დიაგრამის ფრაგმენტი



3.6. Ms SQL Server მონაცემთა ბაზის არჩევა

შემდეგ MsVisualStudio.Net გარემოში სისტემის დახმარებით ავტომატურად ვაგენერირებთ DDL–ფაილს, რომლის ფრაგმენტი მოცემულია ქვემოთ:

— ERM დიაგრამის შესაბამისი DDL ფაილის ტექსტი --------

```
CREATE SCHEMA ORMModel1
GO
CREATE TABLE ORMModel1.Cleint
(
   cleintId BIGINT CHECK (cleintId IN (N'aqtiuri',
N'daxuruli', N'aqvs valdebuleba'))
NOT NULL,
   clientName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
   aqvsDaxurvisTarigi DATETIME NOT NULL,
   aqvsGaxsnisTarigi DATETIME NOT NULL,
```

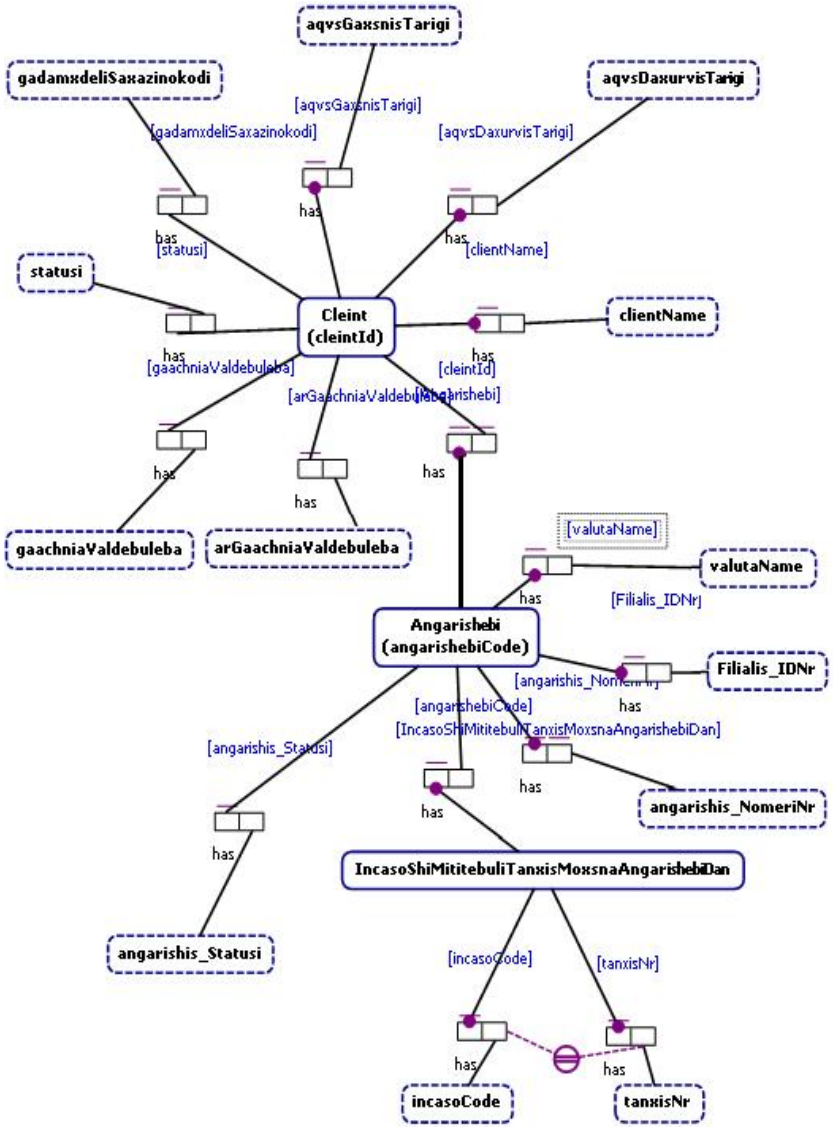


```
   gadamxdeliSaxazinokodi NATIONAL CHARACTER
VARYING(MAX),
   statusi NATIONAL CHARACTER VARYING(MAX) CHECK
(statusi IN (N'aqtiuri', N'daxuruli', N'aqvs
valdebuleba')),
   gaachniaValdebuleba BIT,
   arGaachniaValdebuleba BIT,
   CONSTRAINT Cleint_PK PRIMARY KEY(cleintId)
)
GO
CREATE TABLE ORMModel1.Angarishebi
(
   angarishebiCode NATIONAL CHARACTER(4000) NOT NULL,
   angarishis_NomeriNr INTEGER NOT NULL,
   cleintId BIGINT CHECK (cleintId IN (N'aqtiuri',
N'daxuruli', N'aqvs valdebuleba'))
. . . და ა.შ.
```

3.3. მონაცემთა მოდელის რევერსული დაპროექტება

ORM2 ნოტაცია MsVisualStudio.Net პაკეტის საშუალებით ითვალისწინებს რევერსული დაპროექტების პროცესსაც. შესაძლებელია ERM მოდელიდან აღვადგინოთ ORM მოდელი და გამოვიკვლიოთ მათი სინტაკსურ-სემანტიკური სისრულე. შემდეგ შევადაროთ მიღებული და ადრე დაპროექტებული დიაგრამები ერთმანეთს. ექსპერტული შეფასებების საფუძველზე შესაძლებელი იქნება ცდომილებების მიგნება და შემდეგ მათი კორექტირება.

რევერსული დაპროექტების შედეგი, ანუ რეალურად მომუშავე SQL Server ბაზის DDL-ფაილიდან მიღებული ORM დიაგრამა გამოსახულია 3.7 ნახაზზე.



ნახ.3.7. რევერსიული ხერხით მიღებული ORM დიაგრამა

მიღებული დიაგრამის ანალიზმა ცხადყო, რომ საჭიროა ORM-დიაგრამაში გარკვეული ცვლილებების შეტანა. კერძოდ, აუცილებელია ტერნარული პრედიკადის შეცვლა. ჩასამატებელია ტოლობის შეზღუდვა, რომელიც გვიჩვენებს, რომ "ინკასოში" აუცილებელია მითითებული იყოს თანხის რაოდენობა და ა.შ.

შეიძლება აღვნიშნოთ, რომ თანამედროვე ინფორმაციული ტექნოლოგიების, კერძოდ CASE-ინსტრუმენტების გამოყენებით კორპორაციული მენეჯმენტისა და ინტერკორპორაციული ვებ-აპლიკაციების აგების პროცესში მნიშვნელოვნად უმჯობესდება პროგრამული უზრუნველყოფის ხარისხი და საგრძნობლად მცირდება დაპროექტების, მისი იმპლემენტაციისა და რეინჟინერინგის პერიოდები. ნაშრომში შემოთავაზებული კონცეფციის საფუძველზე შესაძლებელია დიდი საინფორმაციო სისტემების მონაცემთა ბაზების სტრუქტურების დაპროექტების პროცესის ავტომატიზება და აგება, აგრეთვე მისი შემდგომი რესტრუქტურიზაციის პრობლემის მოქნილად გადაწყვეტა.

ამით დავასრულეთ კორპორაციული კომპიუტერული სისტემების სერვისორიენტირებული პროცესებისთვის საჭირო მონაცემთა ბაზის სტრუქტურის ავტომატიზებული დაპროექტების და რეალიზაციის საკითხის განხილვა.

ასეთი მიდგომა, ჩვენი თვალსაზრისით, ხელს შეუწყობს როგორც ბიზნესპროცესების ავტომატიზებული დამუშავების სისტემის ინფორმაციული უზრუნველყოფის სწრაფად აგებას, ასევე ამ სისტემის თანხლების პროცესში, ახალი ამოცანების ინტეგრაციის საკითხის ოპერატიულად გადაწყვეტას.

IV თავი: დინამიკური ბიზნესპროცესების კვლევა პეტრის ქსელებით

4.1. ბიზნესპროცესების მოდელირების, ანალიზის და შეფასების ინსტრუმენტი: პეტრის ქსელები

პეტრის ქსელები (Petri Network) ესაა სისტემის სტატისტიკისა და დინამიკის კვლევის ინსტრუმენტი, კერძოდ მათი ყოფაქცევის მოდელირებისა და ანალიზისათვის [52-54]. პეტრის ქსელები თანამედროვე საინფორმაციო სისტემების მოდელირებისა და ანალიზის ერთ-ერთი უმნიშვნელოვანესი ინსტრუმენტია, რომელსაც წარმატებით იყენებს მსოფლიოს მრავალი ქვეყნის სასწავლო და კომერციული დაწესებულება.

დღეისათვის არსებულ ფორმალურ მეთოდებს შორის პეტრის ქსელებს განსაკუთრებული ადგილი უკავია, როგორც განაწილებული სისტემების თეორიული კვლევის შესაძლებლობებით, ასევე პრაქტიკული გამოყენების სფეროთა სიმრავლით.

მრავალრიცხოვანი მეცნიერულ კვლევების შედეგად შეიქმნა პეტრის ქსელების სხვადასხვა კლასები, რომლებსაც ერთმანეთთან მჭიდრო კავშირი აქვს და მრავალი ცალკეული ტიპის პეტრის ქსელებისაგან შედგება, რაც აქტუალურს ხდის პეტრის ქსელების სტანდარტიზაციის პროცესის ამოცანას.

განსაკუთრებით საყურადღებოა პეტრის ქსელების გამოყენება პარალელური პროცესების მქონე რთულ ობიექტებში, რომლებშიც პროცესები მიმდინარეობს გარკვეულ მიზეზ–შედეგობრივი კავშირებით. როგორც ცნობილია სისტემების მოდელირებისა და ანალიზის ამოცანების გადასაწყვეტად ფართოდ გამოიყენება ისეთი მექანიზმები, როგორებიცაა მასობრივი მომსახურების და იმიტაციური მოდელირების თეორიები. ამასთანავე შეიძლება აღინიშნოს, რომ მათი

გამოყენების ეფექტურობა, გამომსახველობითი და ანალიტიკური მხარეები მკვეთრად განსხვავებულია და დამოკიდებულია როგორც თვით ინსტრუმენტის შესაძლებლობაზე, ასევე ობიექტის სირთულეზე (პარამეტრების რიცხვზე და სხვა).

ფერად პეტრის ქსელებში (Coloured Petri Nets) კარგადაა შერწყმული პეტრის ქსელებისა და დაპროგრამების თეორია (იერარქიულობა, მოდულურობა – დიდი სისტემების მოდელირებისთვის), რაც მის დიდ პრაქტიკულ ღირებულებასაც განაპირობებს თანამედროვე ინფორმაციულ ტექნოლოგიათა გამოყენების მრავალ სფეროში, განსაკუთრებით ბიზნესის და მარკეტინგის მენეჯმენტის ამოცანების გადასაწყვეტად [55,56].

უპირატესობა:

- შეიძლება წარმოდგენილი იქნას, როგორც გრაფიკული, ასევე ანალიტიკური ფორმით;
- უზრუნველყოფს ავტომატიზებულ ანალიზს;
- აქვს საკუთარი მოდელირების ენა (CPN_ML: www.smlnj.org), რომელზეც შესაძლებელია ახალი ფუნქციების შექმნა;
- იძლევა სისტემის აღწერის ერთი დეტალიზაციის დონიდან სხვაზე გადასვლის საშუალებას.

ნაკლი:

- ინსტრუმენტის ინტერფეისი რთულია და მოითხოვს მომხმარებლისგან დროს მასში გასარკვევად;
- CPN-ის ძირითად ბირთვს არ აქვს მოდელირებადი სისტემის დროითი მახასიათებლების აგების და გრაფიკული გაფორმების საშუალება, მაგრამ იგი ადვილად იყენებს არსებულ პაკეტებს (მაგ., ორ- და სამგანზომილებიან გრაფიკას [57].

4.2. სერვის-ორიენტირებული არქიტექტურის ინტერკორპორაციული ვებ-აპლიკაციების ბიზნესპროცესების მოდელირება და კვლევა პეტრის ქსელებით

წინამდებარე პარაგრაფში გადმოცემულია სერვის-ორიენტირებული არქიტექტურის ბაზაზე კორპორაციათაშორისი ბიზნესპროცესების უნიფიცირებული მოდელირების საკითხები და მათი ასახვა სტოქასტური, დროითი პეტრის ქსელების გრაფებით მათი შემდგომი კვლევის მიზნით.

შემოთავაზებულია მოდელების ტრანსფორმაციის ამოცანა:

BPMN -> Activity-D -> PetNet

ინსტრუმენტების ბაზაზე. გამოკვლეულია სისტემაში ბიზნესპროცესების შესრულების დროითი მახასიათებლები, რაც იძლევა სწორი გადაწყვეტილების მიღების საფუძველს ორგანიზაციის სერვისული უზრუნველყოფის შემდგომი სრულყოფის მიზნით.

წარმოდგენილია კონკრეტული ინტერკორპორაციული საინფორმაციო სისტემის სერვის-აპლიკაციების პროგრამული უზრუნველყოფის დაპროექტების და რეალიზაციის საკითხები შემოსავლების სამსახურსა და საფინანსო ბანკებს შორის სერვის-ორიენტირებული არქიტექტურის ბაზაზე. სერვისების განთავსება და მათი გამოყენება მიზანშეწონილია განხორციელდეს MsBizTalk Server–ის ან ადგილობრივი წარმოების ანალოგიური პროდუქტის საფუძველზე. ასეთი ქსელის მწარმოებლურობის ეფექტურობის შესაფასებლად ჩვენ ნაშრომში მოდელირება (იმიტაციური) და ანალიზი (სიმულაციის პროცესი) ჩატარდება პეტრის სტოქასტური ქსელების გამოყენებით [5,40].

არსებობს ერთი პრობლემა, როგორიცაა ჩვენ მიერ აღწერილი ბიზნესპროცესების აქტიურობის დიაგრამის (ნახ.2.10-ა,ბ) ეფექტურობის შეფასების შესაძლებლობა. აქტიურობათა დიაგრამა

დინამიკური პროცესის აღწერის მოდელია. სამწუხაროდ, მას არ გააჩნია რაოდენობრივი შეფასების კრიტერიუმები და ინსტრუმენტი. თეორიაში ასეთი ამოცანები წყდება სისტემების იზომორფიზმზე დაყრდნობით, კერძოდ ისეთი მათემატიკური მოდელის საფუძველზე, როგორიცაა პეტრის დროით-სტოქასტიკური ქსელები. საჭიროა აქტიურობის დიაგრამის შესაბამისი პეტრის ქსელის გრაფის აგება, ანუ მოდელირება და შემდეგ ამ მოდელის დინამიკის ანალიზის ჩატარება დროით პარამეტრებში.

ბიზნესპროცესებს, როგორც 1–ელ თავში აღვნიშნეთ, აღწერენ BPMN–ენის და UML–ის Activity-D დიაგრამის საშუალებით. ეს პროცესები დინამიკური ობიექტებია, რომელთა მოდელირება და ანალიზი შესაძლებელია პეტრის ქსელებით, როგორც იზომორფული მათემატიკური ინსტრუმენტი [40, 42].

აქ შევეხებით პრაქტიკული ღირებულების მქონე თეორიულ საკითხებს, რომლებიც ეყრდნობა, ერთის მხრივ, ობიექტ-ორიენტირებულ მოდელირებას და მეორეს მხრივ, თვით პეტრის ქსელების გამოყენებას. ამოცანის გადაწყვეტა სამი ეტაპისგან შედგება:

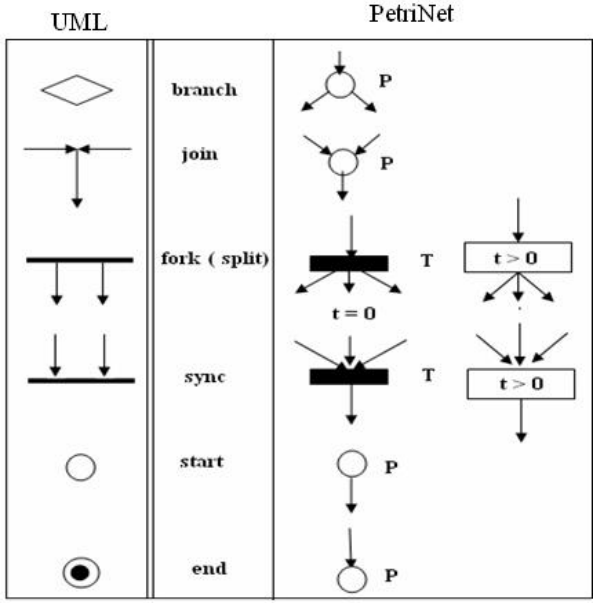
1. შეიქმნას სამუშაო პროცესებისა და ოპერაციების ტრანსფორმაციის (მოდელირების) თეორიული საფუძვლები პეტრის ქსელის გრაფებში;

2. აიგოს სისტემური პეტრის ქსელის გრაფებით წარმოდგენილი პროცესების დიაგრამები გრაფო-ანალიზური ინსტრუმენტების საშუალებით;

3. ჩატარდეს აგებული პეტრის ქსელების მოდელების მანქანური ანალიზი (შესაძლოა მასობრივი მომსახურების მეთოდების გამოყენება).

აქტიურობის დიაგრამაზე (ნახ.2.10–ა) გამოიყენება გრაფიკული ელემენტები: საწყისი და საბოლოო კვანძები,

მოქმედება, შედგენილი-მოქმედება (იერარქიულად ჩადგმული პროცესი), განშტოება და შეერთება. ეს ელემენტები პეტრის ქსელებში მოდელირდება გრაფის პოზიციებით (Pi). დაყოფისა (fork, split) და გაერთიანების (join, sync) ელემენტები კი მოდელირდება პეტრის ქსელის გადასასვლელებით (Tj). 4.1 ნახაზზე ნაჩვენებია ეს იზომორფული ელემენტები.



ნახ.4.1. UML და PetriNet იზომორფული ელემენტები

პეტრის ქსელის პოზიციებზე, გადასასვლელებზე ან/და რკალებზე დროითი დაყოვნების განსაზღვრას (მაგალითად, თითოეული ბიზნესპროცესის შესრულების დრო, მოთხოვნების მოსვლის ინტენსივობა და ა.შ.) დროითი პეტრის ქსელის ტიპი შემოაქვს, დაყოვნების დროთა ალბათურ განაწილებას – სტოქასტური პეტრის ქსელის ტიპი [40]. ამასთანავე, პეტრის ქსელის გადასასვლელი შეიძლება იყოს ორი სახის: მყისიერად

შესრულებადი (დროის დაყოვნების გარეშე) და დაყოვნებით (დროითი).

თანამედროვე ინფორმაციული ტექნოლოგიები ეფუძნება, ერთის მხრივ კომპიუტერებისა და სატელეკომუნიკაციო აპარატურის მაღალ შესაძლებლობებს, მეორეს მხრივ ახალ პროგრამულ სისტემებს და ინფორმაციის დამუშავების ახალ ტექნოლოგიებს. წინა პლანზეა წამოწეული ქსელური, განაწილებული ინფორმაციული სისტემები, რომლებიც ფუნქციონირებს გლობალურ (ინტერნეტი) ან ლოკალურ ქსელებში, ან ორივეში ერთად. ეს საშუალებას გვაძლევს მაქსიმალურად გავმიჯნოთ მომხმარებელთა ფუნქციები და უზრუნველყოთ ინფორმაციული ბაზის დაცვა არასანქცირებული მიმართვებისაგან.

თანამედროვე მიდგომა მსგავსი სისტემების დაპროექტებისა ეფუძნება UML ტექნოლოგიას, რომელშიც სამუშაო ადგილების მოდელირებისათვის გამოიყენება აქტიურობათა დიაგრამა. ვიზუალურად აქტიურობის დიაგრამა გამოიხატება გრაფის სახით, რომელსაც გააჩნია მწვერვალები და წიბოები:

$$\mathbf{D} = \langle \mathbf{S}a, \mathbf{S}m, \mathbf{R}, \mathbf{O} \rangle,$$

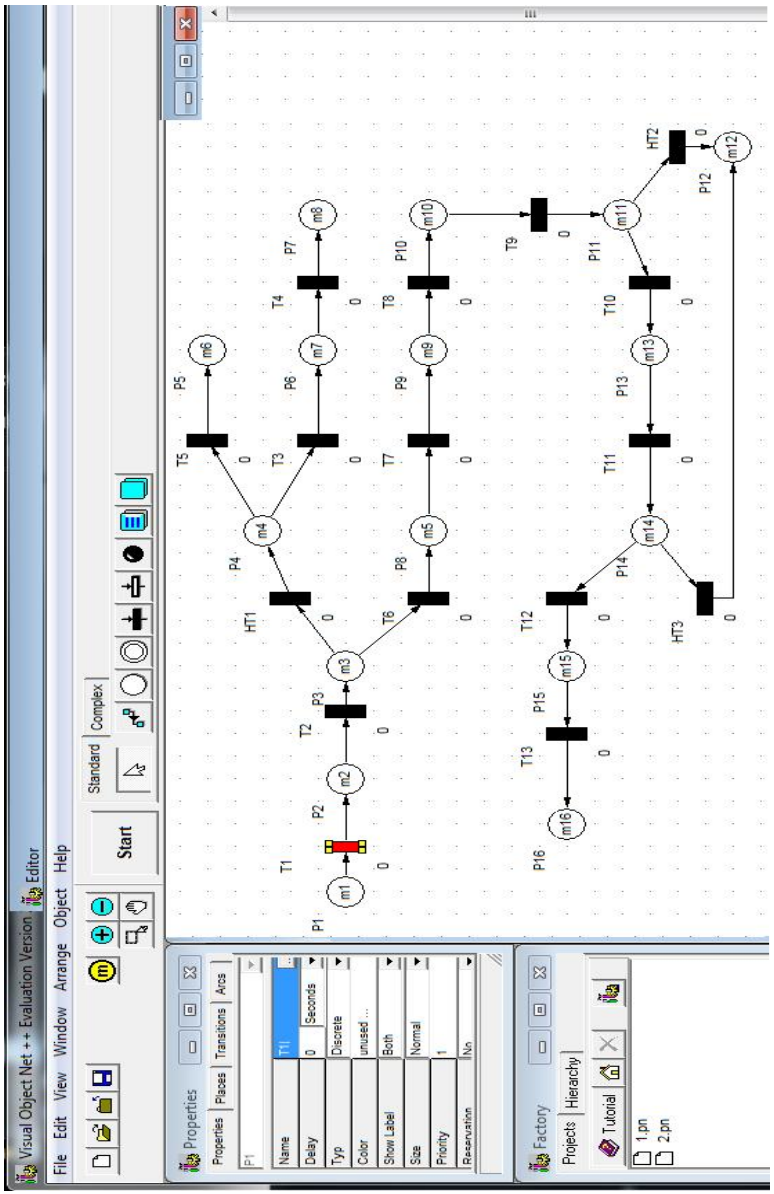
სადაც **S**a – აქტიურობის (პროცესის) და **S**m – მოქმედებათა (პროცედურების) მდგომარეობებია, **R** – გადასვლები და **O** - ობიექტება.

მოქმედების მდგომარეობა ისეთი მდგომარეობაა, რომლის შემდგომი დეკომპოზიცია შეუძლებელია. მისგან განსხვავებით, შეიძლება აქტიურობის მდგომარეობების შემდგომი დეკომპოზიცია, ამის შედეგად შესრულებადი აქტიურობა შეიძლება წარმოვადგინოთ სხვა აქტიურობის დიაგრამების სახით. სემანტიკურად აქტიურობის დიაგრამა ეკვივალენტურია აქტიურობის გრაფის ტრანზიტული გაფართოებისა, მანამდე სანამ არ დარჩება მხოლოდ მოქმედებები. შევდივართ რა ერთერთ ასეთ მდგომარეობაში, სრულდება შესაბამისი მოქმედება ან აქტიურობა,

ხოლო გამოსვლისას მართვა გადაეცემა შემდეგ მოქმედებას ან აქტიურობას.

4.2 ნახაზზე ილუსტრირებულია PetEdit პროგრამული პაკეტის გარემოში 2.10-ა ნახაზის აქტიურობის დიაგრამის ტრანსფორმაციის მაგალითი შესაბამის პეტრის ქსელში. როგორც ნახაზიდან ჩანს, პეტრის ქსელის სქემაზე გაჩნდა დამატებითი ელემენტები: დამხმარე-პოზიცია (Help Position - HP) და დამხმარე-გადასასვლელი (Help Transition - HT). ისინი აუცილებელია სქემის შესაკვრელად, როდესაც მოსაზღვრეა ორი პოზიცია ან ორი გადასასვლელი. ქვემოთ ცხრილებში მოცემულია პეტრის ქსელის პოზიციებისა (ცხრ.4.1) და გადასასვლელების (ცხრ.4.2) შინაარსობრივი მნიშვნელობები.

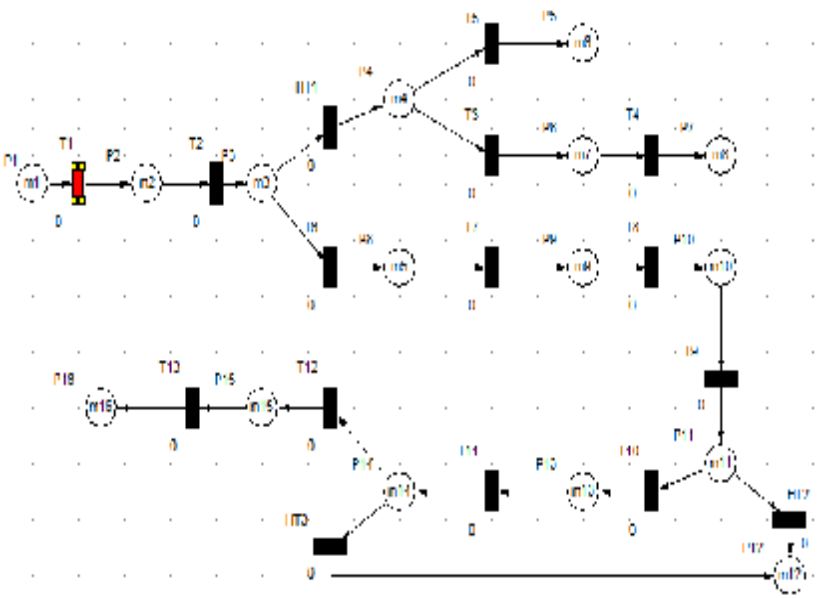
P_N	პოზიციის დანიშნულება	ცხრ.4.1
P1	MOF-ში შემოსულია მოთხოვნა ვალდებულებებზე	
P2	MOF -ში მიღებულია შემოწმების შედეგები	
P3	ბანკში მიღებულია კლიენტის შესახებ ვალდებულება	
P4	კლიენტისთვის მიღებულია მოთხოვნა ”ყადაღის” შესახებ	
P5	„ყადაღა” მოიხსნება ანუ კლიენტის ანგარიშზე ბლოკის მოხსნა	
P6	მოთხოვნა ”ყადაღის” დადებაზე დამოწმებულია	
P7	კლიენტის ანგარიში დაბლოკილია	
P8	MOF-ში მოსულია კლიენტზე სავალდებულო თანხის მოთხოვნა	
P9	სავალდებულო თანხა დადგენილია	
P10	ბანკში შემოსულია სავალდებულო თანხის ზომა	
P11	ანგარიშზე თანხის რაოდენობა	
P12	”დაბლოკილი” ანგარიში	
P13	გადარიცხული თანხა	
P14	ანგარიშზე დარჩენილი თანხა	
P15	”ბლოკმოხსნილი” ანგარიში	
P16	კლიენტის საბოლოო ანგარიშის მდგომარეობა	



ნახ.4.2. პეტრის ქსელის გრაფის ფრაგმენტი აქტიურობის დიაგრამისათვის: ნახ.2.10–ა

T_N	გადასასვლელის დანიშნულება	ცხრ.4.2
T1	კლიენტის ვალდებულების შემოწმება MOF-ში	
T2	შეტყობინების გადაგზავნა ბანკში კლიენტის შესახებ	
HT1	1-ელი დამხმარე გადასასვლელი, როცა არის მოთხოვნა ”ყადაღა”	
T3	ბანკში კლიენტისათვის ”ყადაღის” დამოწმება	
T4	კლიენტის ანგარიშის დაბლოკვა	
T5	კლიენტის ანგარიშზე ბლოკის მოხსნა	
T6	კლიენტის ანგარიშის სტატუსის განახლება (შეზღუდვა)	
T7	შეტყობინების გადაგზავნა MOF-ში სავალდებულო თანხაზე	
T8	MOF-ში თანხის დამტკიცება და ბანკზე შეტყობინება	
T9	შეტყობინების მიღება და კლიენტის ანგარიშის ნახვა	
HT2	მე-2 დამხმარე გადასასვლელი, როცა ანგარიშზე მოწმდება ხელმისაწვდომი თანხა	
T10	თანხის გადარიცხვა	
T11	ტრანზაქციის დამოწმება	
T12	ანგარიშზე ბლოკის მოხსნა	
HT3	მე-3 დამხმარე გადასასვლელი, როცა ანგარიშზე არაა სრული თანხა	
T13	ანგარიშის პროცენტის ჩამოჭრა	

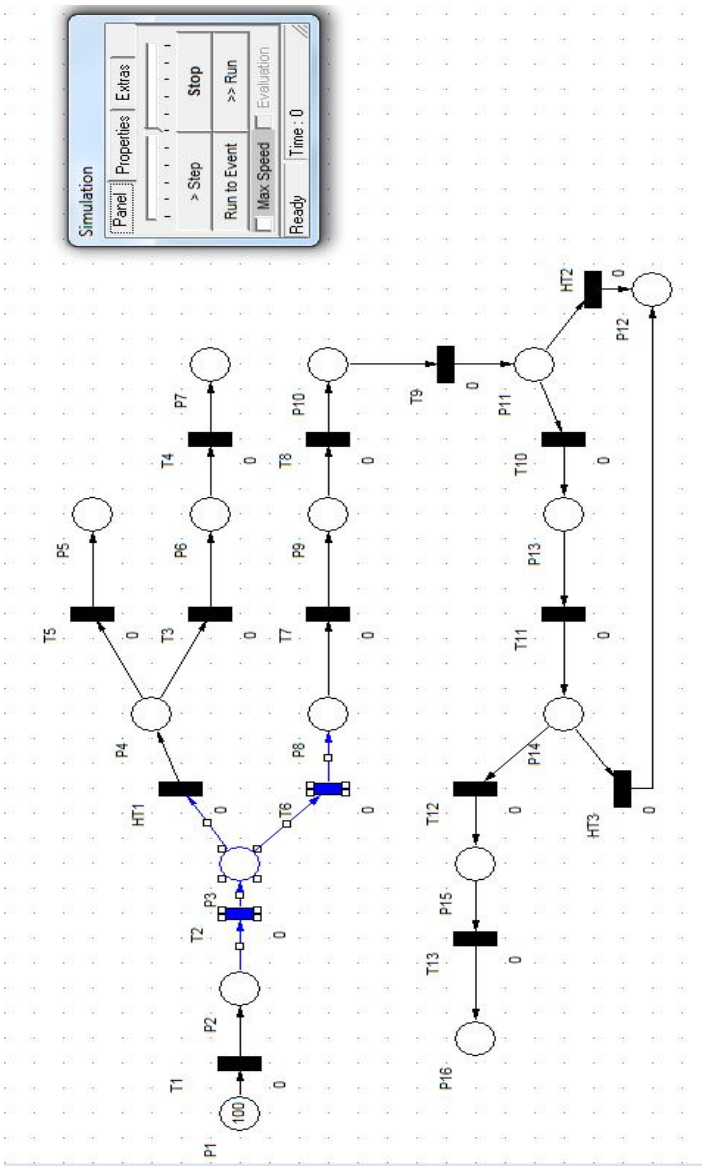
4.3. ნახაზზე ნაჩვენებია თვით პეტრის ქსელის გრაფი.



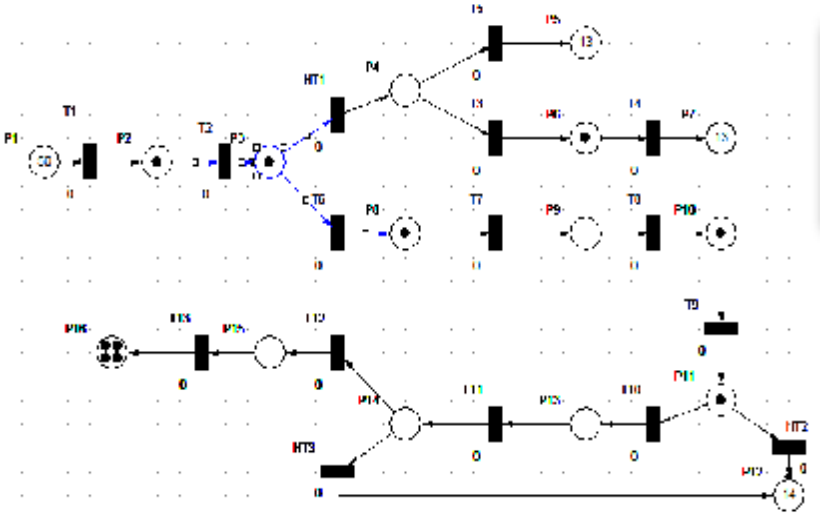
ნახ.4.3. პეტრის ქსელის გრაფი

ჩავატაროთ მიღებული პეტრის ქსელის გრაფის ანალიზი მისი იმიტაციური მოდელირების შესაძლებლობების ფარგლებში.

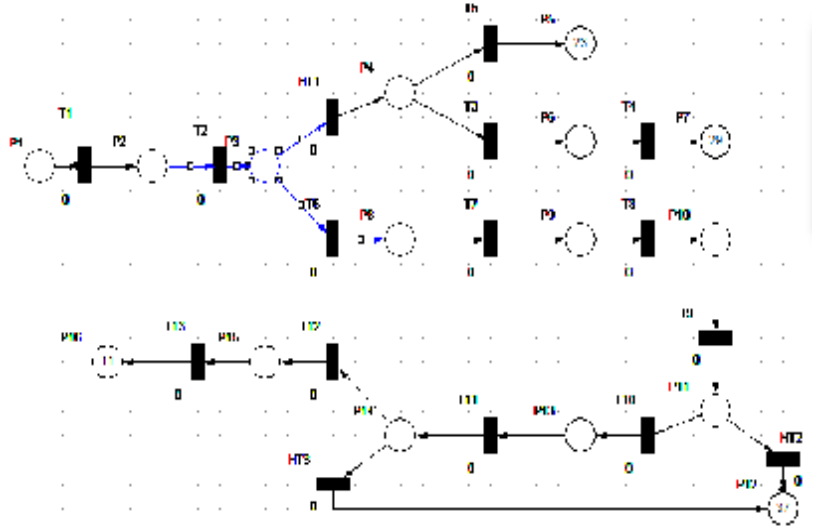
4.4–4.6 ნახაზებზე მოცემულია ქსელის საწყისი მდგომარეობა (100 მოთხოვნით შესასვლელზე), შუალედური მდგომარეობა (50 მოთხოვნა დამუშავდა ან მუშავდება) და საბოლოო მდგომარეობა (ყველა მოთხოვნა დამუშავდა).



ნახ.4.3. საწყისი მდგომარეობა(მონიშნულია P3-კონფლიქტური პოზიცია)



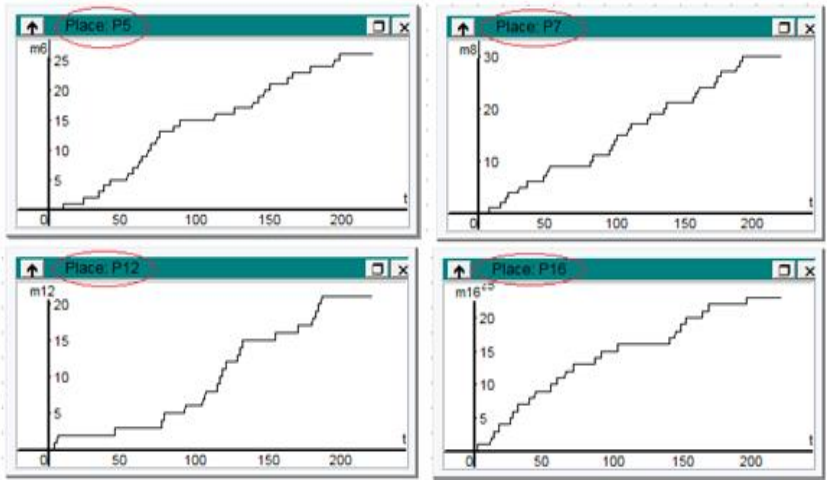
ნახ.4.4. შუალედური მდგომარეობა: 50 მოთხოვნა დასამუშავებელია, 44-დამუშავდა, 6–„დაუმთავრებელ წარმოებაშია“



ნახ.4.5. საბოლოო მდგომარეობა: ყველა მოთხოვნა შესრულდა

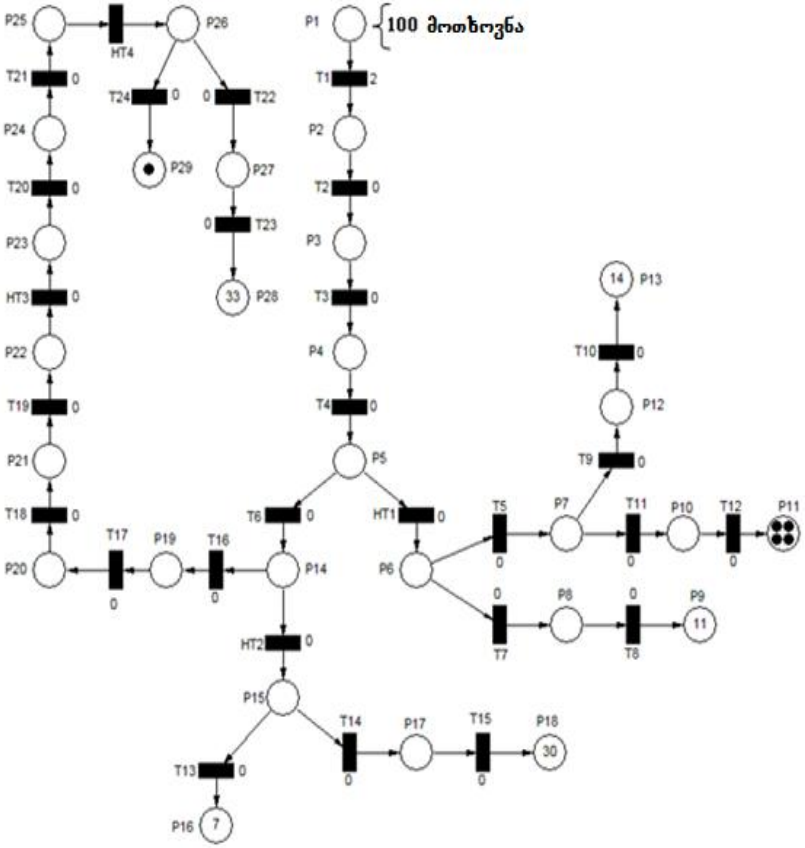
მოთხოვნის ტიპები, რომლებიც გამოიყენება საბანკო კორპორაციაში შემდეგია: *კლიენტის ანგარიშის გახსნა, ანგარიშის დახურვა, კლიენტის სტატუსის შეცვლა, კლიენტის ტიპის შეცვლა, ინკასოს ბლოკის დადება/მოხსნა, ყადაღას დადება/მოხსნა, კლიენტის ანგარიშზე თანხის დადგენა, ინკასოს თანხის გადარიცხვა ბიუჯეტში* და ა.შ.

სისტემაში შემოსული 100 მოთხოვნა პეტრის ქსელის გრაფში მოძრაობისას გადანაწილდა სტოქასტურად (P3, P4, P11, P14 კონფლიქტურ პოზიციებში) და საბოლოო შედეგში დაფიქსირდა: P5=23 (ყადაღა მოხსნილია 23 მოთხოვნაში), P7=29 (ანგარიში დაიბლოკა 29 შემთხვევაში), P12=37 (დაბლოკილი ანგარიშის დატოვება 37-ჯერ, როცა არასაკმარისი თანხაა ბლოკის მოსახსნელად), P16=11 (ანგარიშის საბოლოო მდგომარეობის ნახვა 11-ჯერ, %-ის ჩამოჭრის შემდეგ) და ა.შ. 4.6 ნახაზზე გამოტანილია საკონტროლო პოზიციების (P5, P7, P12 და P16) დროითი დიაგრამები, აგებული პეტრის ქსელის სიმულატორით.



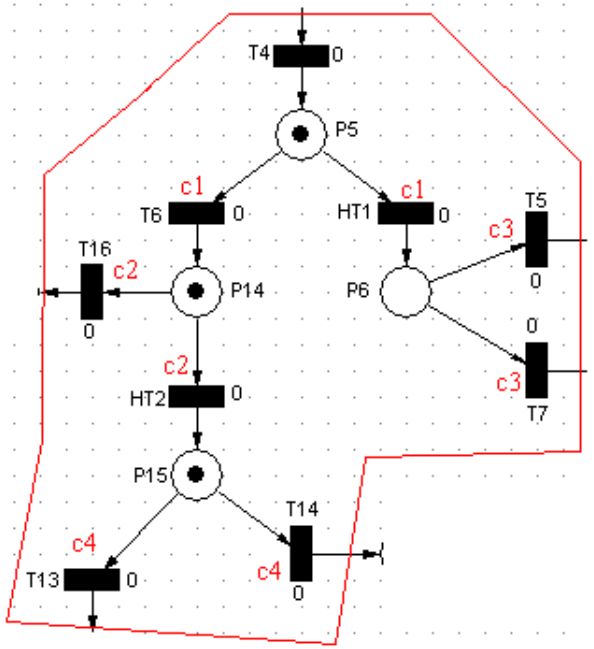
ნახ.4.6. P5, P7, P12 და P16 პოზიციების დროითი დიაგრამები

4.7 ნახაზზე მოცემულია „*კლიენტის რეგისტრაცია/ინკასოს*“ (იხ.ნახ.2.10–ბ) ალტერნატიული დაზუსტებული (გაფართოებული) აქტიურობის დიაგრამის იზომორფული პეტრის ქსელი. 4.3 ცხრილში აღწერილია გადასასვლელთა დანიშნულება. 4.8 ნახაზზე ნაჩვენებია ამ ქსელის ფრაგმენტი კონფლიქტური (ურთიერთ გამომრიცხავი) გადასასვლელელობით.



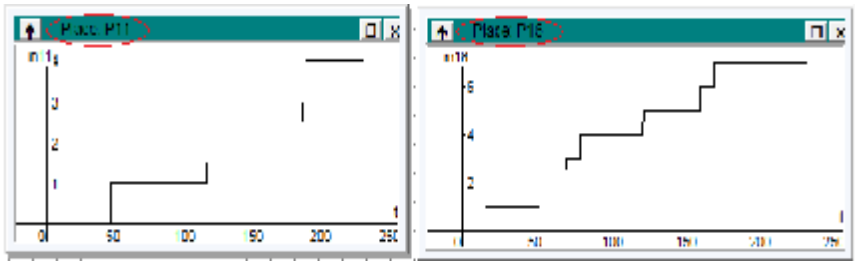
ნახ.4.7. ალტერნატიული აქტიურობის დიაგრამის (ნახ.2.10–ბ) პეტრის ქსელის გრაფი

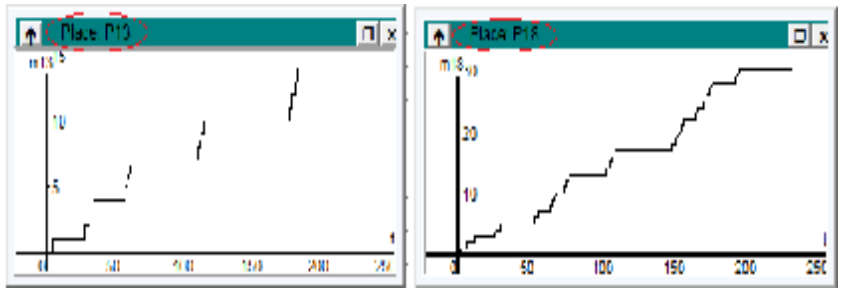
T_N	გადასასვლელის დანიშნულება	ცხრ.4.3
T1	მოთხოვნის ტიპის განსაზღვრა ბანკში	
T2	კლიენტის ვალდებულების შემოწმება MOF-ში	
T3	შეტყობინების გადაგზავნა ბანკში კლიენტის შესახებ	
T4	მიღებულია კლიენტის შესახებ ვალდებულება ბანკში	
HT1	1-ელი დამხმარე გადასასვლელი, როცა არის მოთხოვნა ”გახსნა/დახურვა”	
T5	ანგარიშის გახსნა	
T6	არის ინკასო ან ყადაღა	
T7	ანგარიშის დახურვა	
T8	MOF-ს ეგზავნება შეტყობინება დახურვის შესახებ	
T9	"მხოლოდ ბიუჯეტურში" სტატუსის მიცემა	
T10	მონაცემთა ბაზის შეზღუდული სტატუსის ცხრილში შენახვა	
T11	"ღია" სტატუსის მიცემა	
T12	მონაცემთა ბაზაში შენახვა ”ღია” სტატუსით	
HT2	მე-2 დამხმარე გადასასვლელი, როცა არის მოთხოვნა ”ყადაღა”	
T13	ანგარიშზე ”ყადაღას” ბლოკის მოხსნა	
T14	ანგარიშზე ”ყადაღას” ბლოკის დადების დამოწმება	
T15	ანგარიშის დაბლოკვა ”ყადაღაზე”	
T16	კლიენტის ანგარიშის სტატუსის განახლება (შეზღუდვა)	
T17	შეტყობინების გადაგზავნა MOF-ში სავალდებულო თანხაზე	
T18	MOF-ში თანხის დამტკიცება და ბანკზე შეტყობინება	
T19	შეტყობინების მიღება და კლიენტის ანგარიშის ნახვა	
HT3	მე-3 დამხმარე გადასასვლელი, როცა ანგარიშზე არაა სრული თანხა	
T20	თანხის გადარიცხვა	
T21	ტრანზაქციის დამოწმება	
HT4	მე-4 დამხმარე გადასასვლელი, როცა მოწმდება თანხა გადახდილია თუ არა სრულად	
T22	კლიენტის ანგარიშზე ბლოკის მოხსნა, თანხა გადახდილია სრულად	
T23	ანგარიშიდან %-ის ჩამოჭრა	
T24	კლიენტის ანგარიში დაბლოკილია	

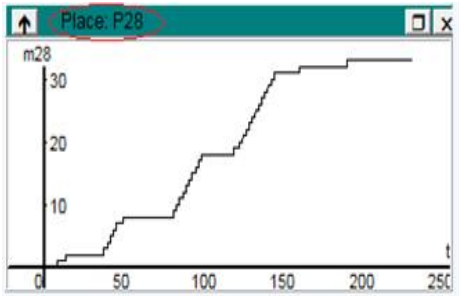


ნახ.4.8. Ci-ურით აღნიშნულია კონფლიქტური გადასასვლელები

4.9 ნახაზზე კი მოცემულია ქსელის იმიტაციით მიღებული დროითი დიაგრამები.







საკონტროლო პოზიციები:

P9	შეტყობინება დახურვაზე	11
P11	მმ-სი "ღია" სტატუსის მიცემა	4
P13	ინკასო "მხოლოდ ბიუჯეტში"	14
P16	ყადაღას ბლოკის მოხსნა	7
P18	ყადაღას დაბლოკვა	30
P28	ინკასოს ბლოკის მოხსნა	33
P29	ინკასოზე დაბლოკილი დარჩა	1

ნახ.4.9. დროითი დიაგრამები დაზუსტებული პეტრის ქსელისთვის (ცხრილში ასახულია მოთხოვნათა ტიპების მიხედვით სერვისების შესრულების შემთხვევითი განაწილება)

4.3. კორპორაციული განაწილებული სისტემის სერვისული რესურსების მართვის მახასიათებლების კვლევა

კორპორაციული განაწილებული სისტემების ქვეშ, როგორც აღნიშნული გვქონდა, ჩვენ ვიხილავთ ორგანიზაციათაშორის ელექტრონული საქმისწარმოების პროცესებს. მაგალითად, ფინანსთა სამინისტროს შემოსავლების სამსახური, სამოქალაქო და საჯარო რეესტრები, საფინანსო ბანკები და ა.შ. ესაა ინტერკორპორაციული სისტემა, რომელსაც გააჩნია განაწილებული მონაცემთა საცავი და პროგრამული პაკეტები, კლიენტ-სერვერ არქიტექტურის კონფიგურაცია ჰომოგენური (ან ჰეტეროგენული) კომპიუტერული რესურსებით, სერვის-ორიენტირებული არქიტექტურით. სისტემის მიზანია მომხმარებელთა მოთხოვნების (უშუალოდ ინტერნეტიდან) მაქსიმალური დაკმაყოფილება, რაც უზრუნველყოფს მის ეფექტიან ფუნქციონირებას და სტაბილურობას.

მასობრივი მომსახურების თეორიის მიხედვით, პირობითად „მომსახურე ორგანოს" სახით შეიძლება განვიხილოთ კორპორაციული ორგანიზაციის (ბანკი, შემოსავლების სამსახური ან სხვ.) თანამშრომელი (მომხმარებელთან უშუალო კონტაქტი) ან კომპიუტერული ქსელის სერვერზე განთავსებული სერვისები (პროგრამული პროდუქტები მონაცემთა ბაზებით). ორივე შემთხვევაში პროცესი მსგავსი მოდელით აიგება (მოთხოვნების ნაკადი, მომსახურების დრო, რიგების სიგრძე და ა.შ.), ოღონდაც თვით ამ მაჩვენებელთა მნიშვნელობები იქნება განსხვავებული [54].

ასეთი მულტიპროცესორული ქსელური კონფიგურაციის სისტემების დაპროექტებისას საჭიროა მრავალი მახასიათებლის გათვალისწინება, რომელთა ოპტიმალური მნიშვნელობების შერჩევა ძალზე მნიშვნელოვანია და ამავე დროს რთულიც. ამ

სიდიდეთა ოპტიმიზაცია არა მარტო გაზრდის კომპიუტერული ქსელის წარმადობას, არამედ შეამცირებს მის შესაქმნელად საჭირო ხარჯებსაც. მნიშვნელოვანია გავითვალისწინოთ ისეთი მომენტები, როგორიცაა სიმძლავრეების, საერთო რესურსების და ა.შ. ოპტიმალური განაწილება [55].

კომპიუტერულ ქსელებში მიმდინარე მოვლენების (დინამიკური პროცესების) მოდელირებისათვის მოსახერხებელია პეტრის ქსელების გამოყენება, რაოდენობრივი მახასიათებლების ანალიზისათვის კი - მასობრივი მომსახურების სისტემების თეორია [56].

ჩვენი მიზანია შევიმუშავოთ სერვის-ორიენტირებული არქიტექტურის კომპიუტერული სისტემისთვის სერვისების რეალიზაციის ალგორითმული სქემები და შესაბამისი პროგრამული პაკეტები, რომელთა დანიშნულებაცაა ქსელის მომხმარებელთა მოთხოვნების დაკმაყოფილება ამ სერვისებით და კომპიუტერული ქსელის სიმძლავრეების ანალიზი და მათი ოპტიმალური განაწილება.

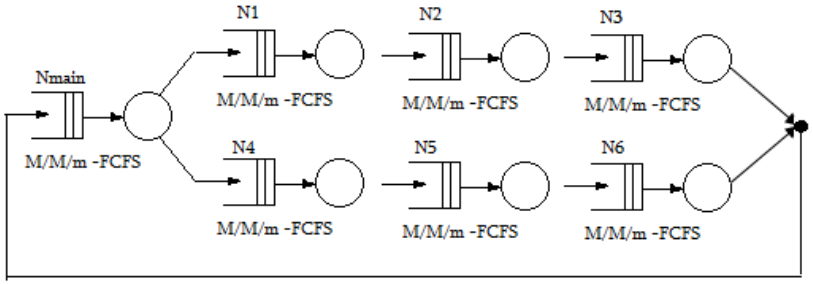
4.3.1. პროცესების კვლევა სტატიკურ რეჟიმში მასობრივი მომსახურების მეთოდებით

პირველ ეტაპზე ავაგოთ და გამოვიკვლიოთ განაწილებული სისტემის სერვისული რესურსების მართვის პროცესის მასობრივი მომსახურების მოდელი სტაციონარული რეჟიმისათვის [55].

განვიხილოთ კორპორაციული ქსელის მოდელი, სადაც არის რამდენიმე მომხმარებელი და რამდენიმე სერვერი (მომსახურე) სერვისებით. დავუშვათ, რომ სერვერთაგან ერთ-ერთი ასრულებს გამანაწილებლის ფუნქციას, ე.ი. იღებს მომხმარებლისაგან მოთხოვნას და უგზავნის მას მომსახურებისათვის იმ სერვერს, რომელიც თავისუფალია. თუ ყველა სერვერი დაკავებულია,

მოთხოვნა დგება რიგში და ელოდება ერთ-ერთი მათგანის განთავისუფლებას.

სერვერი (Ni, i=1,6), მიიღებს რა მოთხოვნას გამანაწილებელი სერვერიდან (Nmain), ემსახურება მას გარკვეული სერვისებით და შედეგებს უბრუნებს ისევ გამანაწილებელ სერვერს, რომელიც, თავის მხრივ პასუხს აგზავნის მომხმარებელთან (ნახ.4.10).



ნახ.4.10.

უნდა ვიგულისხმოთ, რომ მოთხოვნები მომხმარებლებისგან მოდის უწყვეტად, გარკვეული სიხშირით. თითოეული სერვერი ერთეული მოთხოვნის მომსახურებას ანდომებს გარკვეულ დროს. იმ შემთხვევაში როდესაც, მოთხოვნათა ფორმირების სიხშირე დიდია, გამანაწილებელ სერვერთან წარმოიქმნება რიგი. თუკი მოთხოვნათა ფორმირების სიხშირე ძალზე დიდია, ქსელი შეიძლება გადაიტვირთოს და ვეღარ შეძლოს ფუნქციონირება [56].

ჩვენი მიზანია ქსელის არსებული პარამეტრების მეშვეობით დავადგინოთ მისი მუშაობის კრიტიკული წერტილი, შევარჩიოთ ისეთი მახასიათებლები, რომლებიც უზრუნველყოფს მის ნორმალურ ფუნქციონირებას და შევქმნათ პროგრამული პროდუქტი, რომელიც ყოველივე ამას განახორციელებს. მასობრივი მომსახურების თეორიის თვალსაზრისით ზემოთ აღწერილი სისტემა არის M/M/m ტიპის [55].

განვიხილოთ მახასიათებლები და მათ შორის კავშირები, რომლებიც გააჩნია ქსელს. აქვე უნდა აღვნიშნოთ, რომ ქსელის ფუნქციონირებას განვიხილავთ სტაციონარულ რეჟიმში. ამ შემთხვევაში, როგორც ცნობილია, გარკვეულ იდეალიზაციასთან გვაქვს საქმე.

რეალურად დროის ყოველ t მომენტში სისტემაში არსებობს მოთხოვნათა რაღაც k რაოდენობა. ალბათობა იმისა, რომ დროის მოცემულ t მომენტში სისტემაში იმყოფება k მოთხოვნა, აღვნიშნოთ Pk(t)-თი. ჩვენ უნდა ვიგულისხმოთ, რომ t-ს ზრდასთან ერთად ალბათობა Pk(t) თანდათან მუდმივი ხდება. ამ შემთხვავაში Pk(t)-ს ნაცვლად შეიძლება გამოვიყენოთ Pk, რომელიც უკვე აღარ არის დროის ფუნქცია. ეს დაშვება არ გულისხმობს იმას, რომ სისტემა არ გადადის ერთი მდგომარეობიდან მეორეში, რა თქმა უნდა, დროის მიხედვით იცვლება ქსელში არსებული მოთხოვნების რაოდენობა, მაგრამ ალბათობა იმისა, რომ სისტემაში საკმარისად დიდი დროის გასვლის შემდეგ იმყოფება k მოთხოვნა, გამოიხატება Pk-თი.

სერვისულ პროგრამულ პაკეტებში ფუნქციების დასაპროგრამებლად გამოვიყენოთ აღნიშნული კლასიკური მოდელები. ამგვარად, სერვერების რაოდენობით, შემოსულ მოთხვნათა ინტენსივობით და დროით, რომელსაც ანდომებს სერვერი თითოეული მოთხოვნის მომსახურებას, შეგვეძლება დავადგინოთ ქსელის სხვადასხვა მახასიათებელი.

აღვნიშნოთ მოთხოვნათა მოსვლის ინტენსივობა λ-თი, ხოლო თითოეული მოთხოვნის მომსახურების დროს თს-ით. ამ შემთხვევაში ერგოდიულობის პირობა არის: $\lambda * Ts < 1$.

ქსელს გააჩნია შემდეგი მახასიათებლები:

1. მოძრაობის ინტენსივობა: $u = \lambda * Ts$.
2. სერვერის დატვირთვა: $\rho = u / m$.

იმისათვის, რომ სისტემა იყოს სტაბილური, სერვერს უნდა შეეძლოს თავი გაართვას მოთხოვნათა მოსვლის საშუალო

ინტენსივობას, ეს კი ნიშნავს, რომ მოძრაობის ინტენსივობა უნდა იყოს სერვერთა რაოდენობაზე ნაკლები, ან რაც იგივეა, სერვერის დატვირთვა უნდა იყოს ერთზე ნაკლები, ე.ი. $u < m$ ან $\rho < 1$.

M/M/n სახის სისტემების კვლევისას მნიშვნელოვანი ადგილი უკავია ერლანგის ფუნქციას. ეს ფუნქცია განსაზღვრავს იმის ალბათობას, რომ ყველა სერვერი დაკავებულია, და იმავდროულად იმის ალბათობასაც, რომ მოსულ მოთხოვნას მოცდა მოუწევს [56]. ერლანგის ფუნქციისთვის გამოვიყენებთ გამოსახულებას:

$$Ec(m,u) = (u^m / m!) / (u^m / m! + (1-\rho)\sum_{k=0}^{m-1}(u^m / m!))$$

მომხმარებლისთვის დიდი მნიშვნელობა აქვს მოთხოვნის რიგში დგომის (მოცდის) საშუალო დროს, იგი გამოითვლება ფორმულით:

$$Tw = \frac{Ec(m,u)Ts}{m(1-\rho)}$$

აუცილებელია განვსაზღვროთ მოთხოვნის სისტემაში ყოფნის საშუალო დრო: Tq = Tw + Ts.

ალბათობა იმისა, რომ მოთხოვნის სისტემაში ყოფნის დრო ნაკლებია t-ზე დამოკიდებულია u = m-1, თუ არა. თუ ეს პირობა სრულდება, მაშინ ადგილი აქვს შემდეგ ტოლობას:

$$P(\text{სისტემაში ყოფნის დრო} < t) = 1 - (1 + \frac{t}{Ts} Ec(m,u)) e^{\frac{-t}{Ts}}$$

წინააღმდეგ შემთხვევაში:

$$P(\text{სისტემაში ყოფნის დრო} < t) = 1 + \frac{B + Ec(m,u)}{B} e^{\frac{-t}{Ts}} + \frac{Ec(m,u)}{B} e^{-(m-u)\frac{t}{Ts}}$$

სადაც B=m-1-u.

დროის ყოველ მომენტში ქსელში იარსებებს მოთხოვნათა გარკვეული რაოდენობა. რაც ნაკლები მოთხოვნაა ქსელში, მით უკეთ ფუნქციონირებს იგი. ალბათობა იმისა, რომ ქსელში არის k მოთხოვნა არის Pk სადაც

$$P_k = \frac{u}{k!} P_0$$

როცა k <= m და

$$P_k = \frac{u^k}{m! m^{k-m}} P_0$$

როცა $k >= m$.

P_0 არის ალბათობა იმისა, რომ ქსელში საერთოდ არაა მოთხოვნა.

ეს რაც შეეხებოდა ალბათობებს. თვით სისტემაში არსებულ მოთხოვნათა რაოდენობა კი არის Lq, სადაც

$$Lq = u + \frac{pEc(m,u)}{1-\rho}$$

თუკი ქსელში არის m ან m-ზე ნაკლები მოთხოვნა, მაშინ იმ მოთხოვნების რაოდენობა, რომლებიც რიგში დგანან 0-ის ტოლია, ხოლო თუ ვიცით, რომ x მოთხოვნა რიგში დგას, მაშინ მთლიანად სისტემაში იქნება x+m მოთხოვნა. ასე, რომ გვაქვს შემდეგი მახასიათებლები:

ალბათობა იმისა, რომ არცერთი მოთხოვნა არ იცდის:

$$\text{P(არცერთი მოთხოვნა არ იცდის)} = \sum_{k=0}^{m} P_k$$

ალბათობა იმისა, რომ x მოთხოვნა დგას რიგში:

$$\text{P(x მოთხოვნა იცდის)} = P_{x+m} \quad \text{სადაც } x>m$$

მომლოდინე მოთხოვნათა საშუალო რიცხვი:

$$Lw = \frac{pEc(m,u)}{1-\rho}$$

რეჟიმს. ჩვენ მიერ შექმნილი პროგრამული საშუალება სწორედ ამ სიდიდეებს და ფორმულებს იყენებს ქსელის პარამეტრების ანალიზისათვის და მათი ოპტიმალური მნიშვნელობის შერჩევისათვის.

იგი, იღებს რა ინფორმაციას ქსელში მოთხოვნების მოსვლის სიხშირეზე, სერვერთა რაოდენობასა და თითოეული მოთხოვნის მომსახურების დროზე, ანგარიშობს ისეთ პარამეტრებს როგორიცაა მოთხოვნის რიგში დგომის დრო, ბუფერში მოთავსებული მომლოდინე მოთხოვნათა რაოდენობა, სერვერის დატვირთვა და მოძრაობის ინტენსივობა, სხვადასხვა ალბათობები და ა.შ.

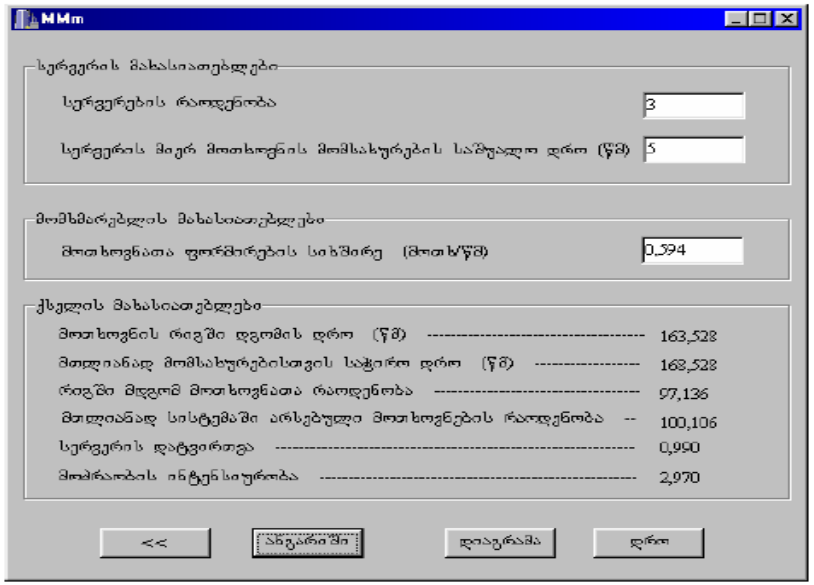
გარდა ამისა, გამოითვლის მოცემულ პირობებში ოპტიმალური მუშაობისათვის საჭირო პარამეტრებს და აგებს მათ შორის დამოკიდებულებათა გრაფიკებს.

4.11 ნახაზზე მოცემულია C++ ენის ინსტრუმენტით აგებული მომხმარებლის ინტერფეისი, რომელიც მუშაობს ვიზუალური და ტრადიციული დაპროგრამების კომპონენტების რევერსული ტექნოლოგიით [28].

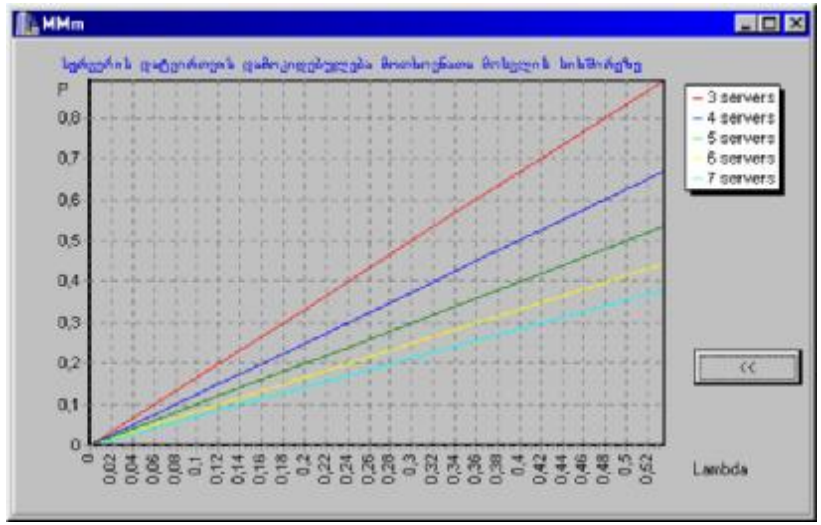
როგორც ნახაზიდან ჩანს, მომხმარებელს შეუძლია შეიტანოს (და ცვალოს) სამი პარამეტრის მნიშვნელობა: სერვერების რაოდენობა, მომსახურების საშუალო დრო და მოთხოვნათა რაოდენობის ინტენსიურობა.

ღილაკით „ანგარიში“ სისტემა გაიანგარიშებს ქსელის ძირითად მახასიათებლებს, კერძოდ: სერვერის დატვირთვა, მოძრაობის ინტენსიურობა, მოთხოვნის რიგში დგომის დრო, მთლიანად მომსახურებისთვის საჭირო დრო, რიგში მდგომ მოთხოვნათა რაოდენობა, სისტემაში მყოფ მოთხოვნათა საერთო, რაოდენობა.

ღილაკით „დიაგრამა“ გამოიტანება გაანგარიშების შედეგად მიღებული გრაფიკები. კერძოდ, 4.12 ნახაზზე მოცემულია პროგრამულად მიღებული დიაგრამა სერვერის დატვირთვის დამოკიდებულებისა მოთხოვნათა მოსვლის სიხშირეზე სერვერების სხვადასხვა რაოდენობისათვის (მაგალითად, 3-:-7).

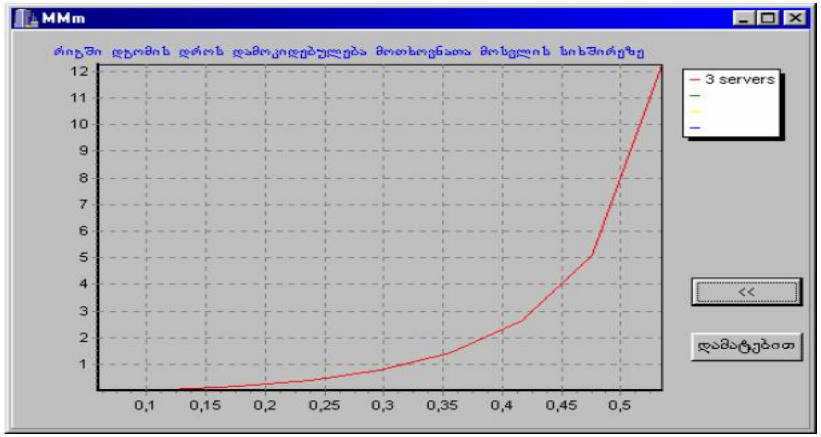


ნახ.4.11

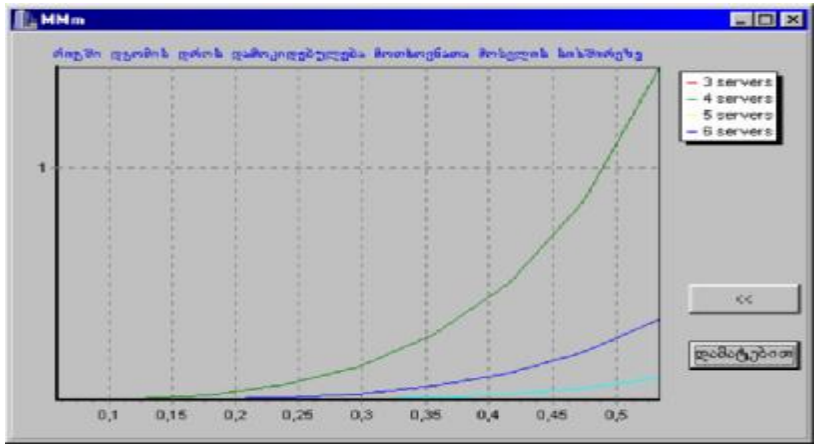


ნახ.4.12

4.13 და 4.14 ნახაზებზე კი მოცემულია დიაგრამები მოთხოვნათა რიგში დგომის დროის დამოკიდებულებისა მოთხოვნათა მოსვლის სიხშირეზე სერვერების სხვადასხვა რაოდენობის (მაგ., 3-:-6) შემთხვევაში. ბოლო დიაგრამებიდან კარგად ჩანს, თუ როგორ იკლებს მოთხოვნათა რიგში დგომის დრო მომსახურე არხების მომატებით.



ნახ.4.13



ნახ.4.14

4.3.2. პროცესების კვლევა დინამიკურ რეჟიმში პეტრის ქსელებით

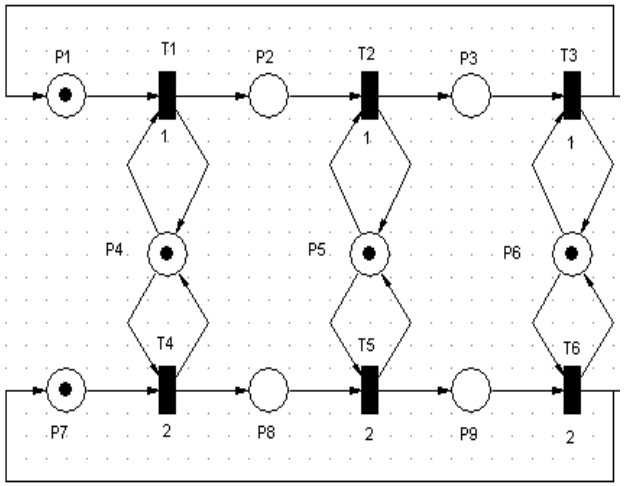
ახლა გავაანალიზოთ კომპიუტერული ქსელის მოქმედება დინამიკურ რეჟიმში. ამ შემთხვევაში სერვერების მიერ კლიენტთა მოთხოვნების დაკმაყოფილების პროცესი შეიძლება მოდელირებულ იქნას ტრანზიტული დროითი პეტრის ქსელის (Timed Transition Petri Net) საშუალებით [40].

პეტრის ქსელის დროითი გაფართოება ჩვენს შემთხვევაში იქნება ალბათური (სტოქასტური). ამგვარად, ასეთი ქსელის ანალიზი შესაძლებელია მარკოვის მეთოდების გამოყენებით, რომელშიც დრო ექსპონენციალურადაა განაწილებული [52,58].

სტოქასტური პეტრის ქსელის მისაღებად საჭიროა „პოზიცია-გადასასვლელების ქსელს" დაემატოს გადასასვლელთა გაშვების (დაყოვნების, მოლოდინის) დროთა მომენტები (მაგალითად, μ1, μ2, . . . , μn). განვიხილოთ კერძო მაგალითი კორპორაციული ქსელისათვის, ორი სერვერით და სამი კლიენტით.

4.15 ნახაზზე მოცემულია სტოქასტური პეტრის ქსელის საწყისი მდგომარეობა. მარკერის არსებობა S1(p1,p2,p3) და S2(p7,p8,p9) სერვერებში ნიშნავს მათ მზადყოფნაზე კლიენტების მომსახურებისათვის.

დავუშვათ, რომ C(p4,p5,p6) კლიენტის პოზიციებში მარკერები მუდმივადაა, ე.ი. მოთხოვნები არსებობს და ისინი ელოდება სერვერის მომსახურებას. როგორც აღვნიშნეთ, Tj გადასასვლელის გახსნის დროა (ანუ მომსახურების დაყოვნების დრო). Tj-ური გადასასვლელის გახსნის საშუალო დრო იქნება 1/μ, სადაც μ გადასასვლელის გახსნის ინტენსივობაა.



ნახ.4.15. სტოქასტური პეტრის ქსელის საწყისი მდგომარეობა

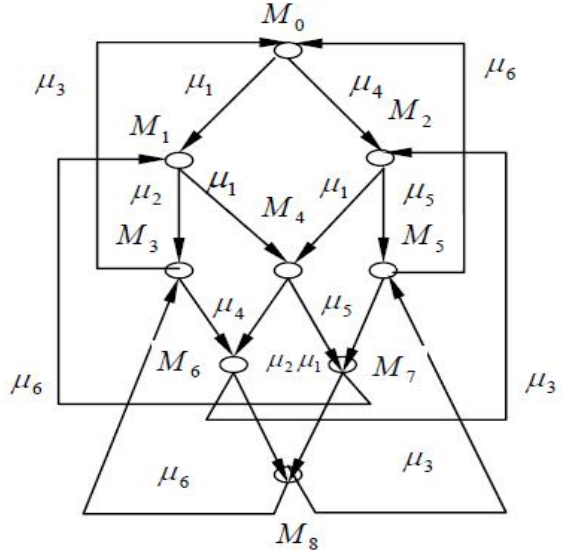
სისტემის მდგომარეობები, ანუ მარკერების სიმრავლე შეიძლება ასე ჩაიწეროს:

p პოზიციები და t გადასასვლელები:

M1__100111001
M2__010111001
M3__100111100
M4__010111001
M5__100111010
M6__001111100
M7__010111010
M8__001111010

სადაც Mi, 0<=i<=K მდგომარეობებია (მარკირებები); Tj, 1<=j<=L გადასასვლელები; μ_j, 1<=j<=L _ დაყოვნების დრო გადასასვლელის გასაღებად. ჩვენს შემთხვევაში m=3 და n=2, ამიტომ კომბინაცია იქნება $m^n = 9$.

გადასასვლელების გახსნის ორგანიზება, როცა სისტემა ყველა მდგომარეობას გადის ნაჩვენებია 4.16 ნახაზზე, რომელსაც პეტრის ქსელის მიღწევადობის გრაფს უწოდებენ.



ნახ.4.16 პეტრის ქსელის მიღწევადობის გრაფი

ასეთი სტოქასტური პეტრის ქსელის რაოდენობრივი ანალიზი შეიძლება განხორციელდეს შესაბამისი მარკოვის პროცესების ანალიზით. განვიხილოთ მარკოვის ჯაჭვის მაგალითი, რისთვისაც ამ ნახაზზე გრაფის რკალებზე მივამაგროთ გადასასვლელების გაშვების μ კოეფიციენტები.

ჩვენთვის საინტერესოა დავადგინოთ სისტემის თითოეულ მდგომარეობაში გადასვლის ალბათობები, ამისათვის საჭიროა შევადგინოთ კოლმოგოროვის განტოლებათა სისტემა [58]:

P5*μ6+P3*μ3-P0*(μ1+μ4)=0

P7*μ6+P0*μ1-P1*(μ2+μ4)=0

P0*μ4+P6*μ3-P2*(μ1+μ5)=0

P6*μ3+P0*μ4-P3*(μ1+μ5)=0

P1*μ4+P2*μ1-P4*(μ2+μ5)=0

P2*μ5+P8*μ3-P5*(μ1+μ6)=0

P3*μ4+P4*μ2-P6*(μ3+μ5)=0

P4*μ5+P5*μ1-P7*(μ2+μ6)=0

P6*μ5+P7*μ2-P8*(μ3+μ6)=0

P0+P1+P2+P3+P4+P5+P6+P7+P8=1

მაგალითად, თუ დავუშვებთ, რომ μ1=3, μ2=5, μ3=2, μ4=3, μ5=1, μ6=7, მაშინ ალბათობათა მნიშვნელობები, შესაბამისად იქნება:

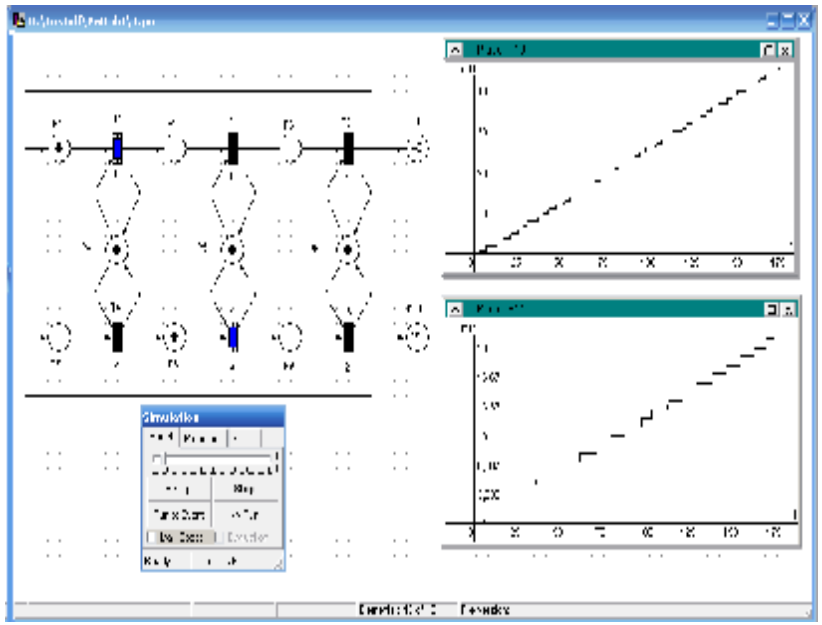
P0=0.11; P1=0.05; P2=0.07;
P3=0.26; P4=0.06; P5=0.02;
P6=0.37; P7=0.01; P8=0.05.

უნდა აღინიშნოს, რომ კომპიუტერული ქსელისათვის ერთი მდგომარეობიდან მეორეში გადასვლის ინტენსივობა μ არის სერვერის მიერ შესაბამისი კლიენტის მოთხოვნის მომსახურებისათვის საჭირო დროის შებრუნებული სიდიდე, ე.ი. 1/Ts. ამ განტოლებათა სისტემის ამოხსნით გაუსის მეთოდით მივიღებთ სისტემის ერთი მდგომარეობიდან მეორეში გადასვლის ალბათობებს $\mathbf{P_0, P_1, P_2, \ldots, P_8}$.

4.15 ნახაზზე წარმოდგენილი პეტრის ქსელის გრაფისთვის PetEdit რედაქტორში ავაგოთ შესაბამისი მოდელი. სერვერებისთვის შევირჩიოთ პირობითად განსხვავებული მწარმოებლურობა, კერძოდ ერთი ამუშავებს მოთხოვნებს 1 წმ-ში,

მეორე კი - 2 წამში (ამ მნიშვნელობების ცვლილებით შესაძლებელია შემდგომი ექსპერიმენტების ჩატარება).

4.17 ნახაზზე ნაჩვენებია მიღებული პეტრის ქსელის მოდელი და სიმულაციის შედეგები. P10 და P11 პოზიციები ასახავს სერვერების მიერ შესრულებული პროცედურების ჯამურ რაოდენობას.



ნახ.4.17. პეტრის ქსელის სიმულაციის რეჟიმი მახასიათებლებით

როგორც დიაგრამებიდან ჩანს, პირველი სერვერის სწრაფქმედების, ან სერვისების დამუშავების პროცედურების ხანგრძლივობა თითქმის ორჯერ ნაკლებია. ამიტომაც შედეგები P10 პოზიციაში ორჯერ მეტია. თუ სერვერული სისტემებისთვის მოხდება სერვისული ოპერაციების დამუშავების დროის წინასწარ განსაზღვრა, მაშინ შედეგებიც შესაბამისად აისახება.

4.4. სერვისებით მოთხოვნების დამუშავების პროცესების იმიტაციური მოდელირება

განხილულია საფინანსო კორპორაციაში მოთხოვნების დამუშავების პროცესების კვლევა სისტემური ანალიზის და სერვისული აპლიკაციების გამოყენების საფუძველზე. განიხილება ინტეგრირებული მართვის ავტომატიზებული სისტემის აგების კონცეფცია UML სტანდარტებით და კლიენტ-სერვერ არქიტექტურით. შემოთავაზებულია პეტრის ქსელების გამოყენებით იმიტაციური მოდელის აგების მაგალითი.

განვიხილოთ ინტერკორპორაციული საინფორმაციო სისტემის სერვის-აპლიკაციებს შორის ინფორმაციის გაცვლის პროცესის მოდელირების და ანალიზის საკითხები, კერძოდ საფინანსო ბანკებს და ფინანსთა სამინისტროს შემოსავლების სამსახურს შორის სერვის-ორიენტირებული არქიტექტურის ბაზაზე. ასეთი ქსელის ანალიზისათვის ვიყენებთ იმიტაციური მოდელირების ფერადი პეტრის ქსელის, CPN (Colored Petri Net) ინსტრუმენტს [6,56].

საკვლევი სისტემის მოდელის ასაგებად საჭირო მოთხოვნილებათა განსაზღვრა მოხდა სერვისების ბიზნესპროცესების და ბიზნესწესების შესწავლის საფუძველზე და შესაბამის აქტიურობათა დიაგრამების აგებით (ნახ.2.10).

სერვის-ფუნქციები, რომლებიც განთავსებულია შემოსავლების სამსახურისა და საფინანსო ბანკების სერვერებზე, შეიძლება წარმოვადგინოთ ობიექტორიენტირებული დაპროგრამების საფუძველზე როგორც კლასები და მათი მეთოდები (ცხრ.2.1).

როგორც ბიზნესპროცესების ანალიზმა გვიჩვენა, საფინანსო ბანკებსა და შემოსავლების სამსახურს შორის, როგორც ინტერკორპორაციული ორგანიზმისთვის, დამახასიათებელია შეტყობინებათა და მონაცემთა გაცვლის

სერვისების ხშირი გამოყენება (ყოველდღიურად ბანკმა შეიძლება მიიღოს (ან გადასცეს) 1000-ზე მეტი მოთხოვნა კლიენტების შესახებ).

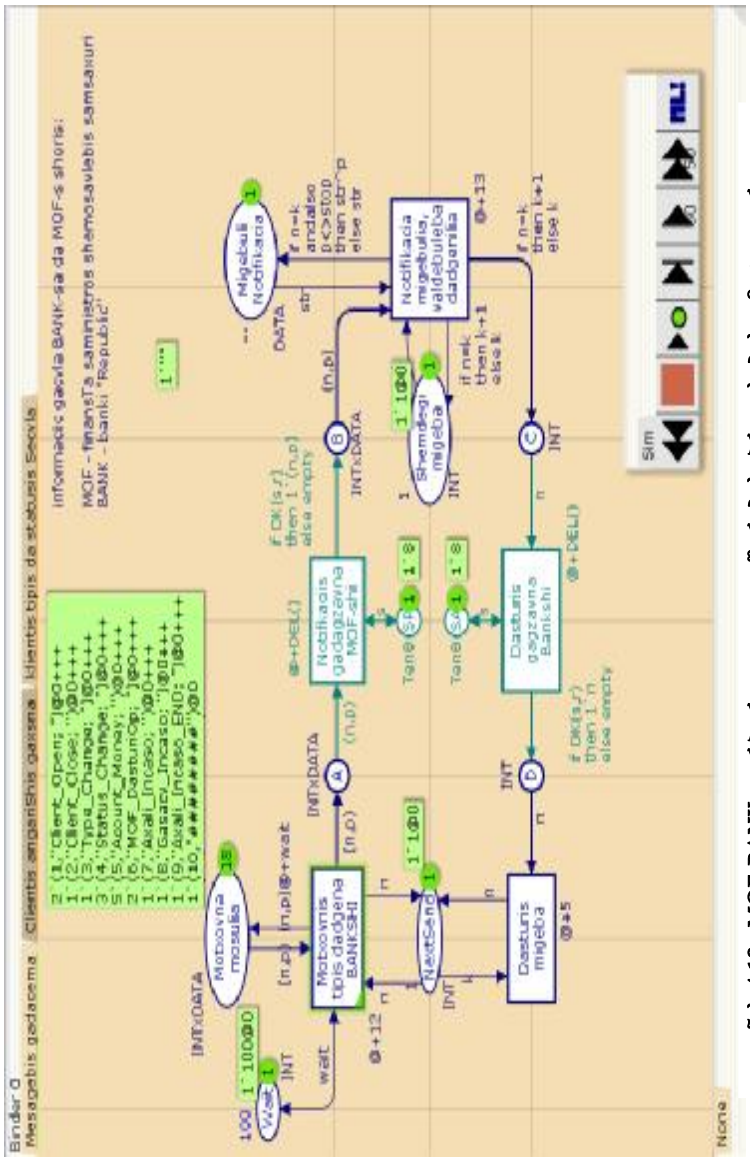
ასეთი ინფორმაციის მენეჯმენტი მოითხოვს საიმედო აღრიცხვისა და რისკების გამორიცხვის პროცედურების გათვალისწინებას. შეტყობინებათა ერთობლიობა, რომელიც მუდმივად გადაიცემა ქსელის საშუალებით, არ უნდა დაიკარგოს და ყოველი მათგანი უნდა ექვემდებარებოდეს მკაცრ კონტროლს, უნდა შეიძლებოდეს აღდგენა, ანუ განმეორებითი პროცედურის შესრულება.

კორპორაციებს შორის ასეთი სერვისული მოთხოვნების დამუშავების მართვის პროცესის მოდელირება ჩატარდა CPN ინსტრუმენტით. 4.18 ნახაზზე მოცემულია ასეთი ქსელის ფრაგმენტი ჩვენი სისტემისათვის.

აქ გადასასვლელ ბლოკებში ნაჩვენებია, მაგალითად, ***მოთხოვნის ტიპის დადგენა ბანკში, ნოტიფიკაციის გადაგზავნა MOF-შემოსავლების სამსახურში, ნოტიფიკაცია მიღებულია და ვალდებულება დადგენილია, MOF-დან დასტურის გადაგზავნა ბანკში, დასტურის მიღება ბანკში.*** თითოეული მათგანი უნდა გაიშალოს დამოუკიდებელი პეტრის ქსელით და მოხდეს მათი ანალიზი, ამასთანავე შეიქმნება ერთიანი იერარქიული სისტემა ჩადგმული პეტრის ქვექსელებით.

CPN-ინსტრუმენტი იყენებს ობიექტორიენტირებული, ვიზუალური დაპროგრამების პრინციპებს, მისი ენა ML საშუალებას იძლევა აღიწეროს ქსელის ფერადი კომპონენტები (მარკერები), ცვლადები, კონსტანტები და თვით პოზიციების, გადასასვლელებისა და რკალების ტექსტური აღწერები, რაც ერთგვარ კომფორტს ქმნის ქსელის წასაკითხად და გასაგებად.

ქსელის ყოველი პოზიციის გვერდით შეიძლება აისახოს მოცემულ მომენტში შემავალი ფერადი მარკერები.



ნახ.4.18. MOF-BANK კორპორაციული კავშირების პროცესების იმიტაციური მოდელი CPN -ის გარემოში: „მოთხოვნების დამუშავება“

საინიციალიზაციო მარკირება ხაზგასმული ტექსტის სახით გამოიტანება. მაგალითად, საწყის მდგომარეობაში პოზიცია „*მოთხოვნა მოსულია*“ შეიცავს INTxDATA ტიპის ფერად მარკერთა 9-ელემენტიან სიმრავლეს (საინიციალიზაციო მარკირება):

{ 2’(1, `კლიენტის_გახსნა“), 1’(2, `კლიენტის დახურვა“), 1’(3, „ტიპის ცვლილება“), 3’(4, „სტატუსის ცვლილება“), 5’(5, „ანგარიშზე თანხა”), 2’(6, „MOF-დან დასტური”) და ა.შ. }. აქ ბოლო, მე-10 ელემენტი შეესაბამება დასასრულის იდენტიფიკაციას - stop.

პირველი რიცხვი სტრიქონში: კოეფიციენტია, რომელიც მიუთითებს, რომ პოზიციაში არის არაუმეტეს 1 ცალი მოცემული ფერის მონაცემი (ანუ არსებობს მხოლოდ ერთი მოთხოვნა ნომრით „*ტიპის ცვლილება*“, რომლის ფერია - რიგითი ნომერი 3). ამ შემთხვევაში გვაქვს მონაცემთა ელემენტების სიმრავლე.

მეორე მაგალითი, პოზიცია ”*შემოსული_მოთხოვნები*” შედგება 18 ელემენტისგან (2+1+1+3+5+2+1+1+1+1), რომლებიც 9 სხვადასხვა (მარკერების ფერის) მოთხოვნათა ტიპების რაოდენობას, ანუ მულტისიმრავლეს ასახავს.

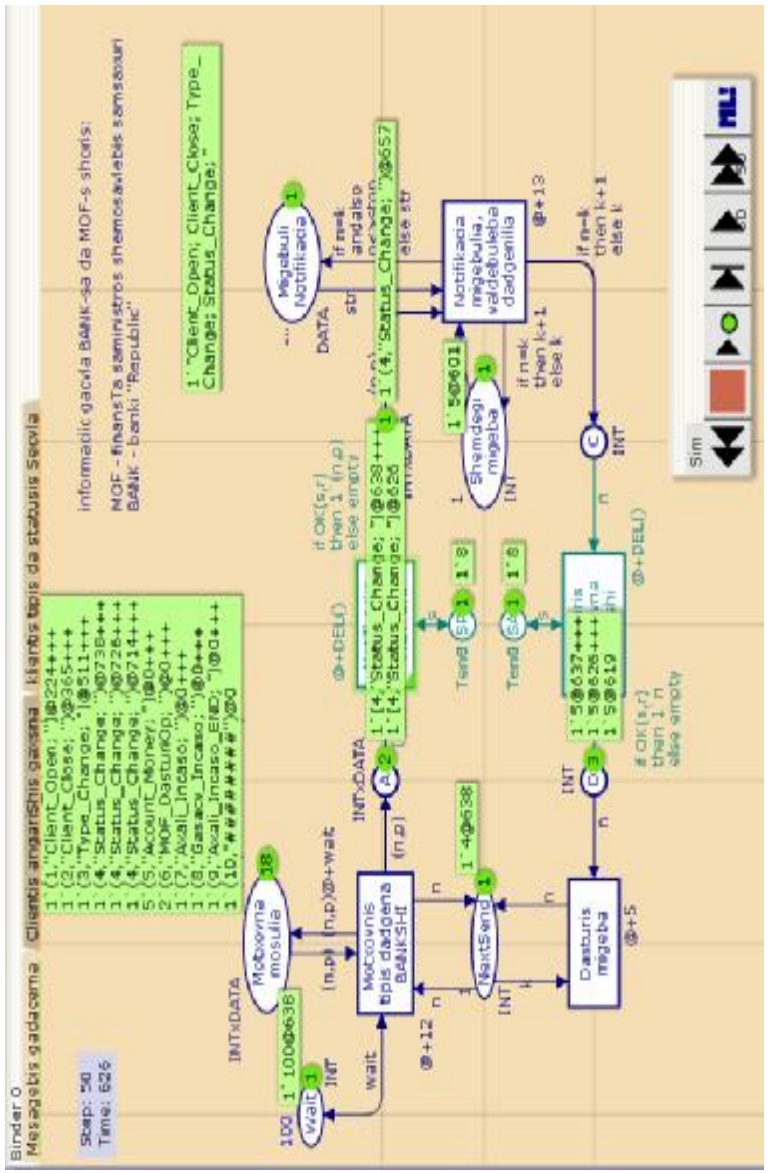
პროცესების შესრულების დრო (დაყოვნება) აისახება გადასავლელთან სიმბოლოს და დროის ერთეულის (მაგალითად, @+7, @+wait) მითითებით, სადაც wait წინასწარ განსაზღვრული კონსტანტაა.

ამავე ნახაზზე ასახულია არადეტერმინირებული ლოგიკური გამოსახულება (პირობის ბლოკი) ფერადი პეტრის ქსელის რკალებზე, რომლებიც გადასასვლელთა გაშვების სხვადასხვა პირობებს და შედეგებს ასახავს, ანუ ლოგიკური პირობის ჭეშმარიტებისას გადასასვლელს განსხვავებული მნიშვნელობა მიეწოდება (ან გადასასვლელიდან განსხვავებული მნიშვნელობა გამოვა), მცდარობისას – განსხვავებული. მაგალითად, გადასასველს „*ნოტიფიკაციის* გადაგზავა *MOF-ში*“

გამოსასვლელ რკალზე აქვს ლოგიკური პირობა - თუ გამოგზავნილი ნოტიფიკაციის ნომერი (n) ემთხვევა კლიენტის ანგარიშის ნომერს (k), მაშინ გვაქვს ”true”, წინააღმდეგ შემთხვევაში „false“, რაც იმას ნიშნავს, რომ საჭირო ნოტიფიკაცია არაა მისული MOF-ში. თუ ყველაფერი წესრიგშია, მაშინ MOF (მიმღები) უგზავნის ბანკს შეტყობინებას გადასასვლელით „*დასტურის_გამოგზავნა*“.

ნოტიფიკაციის და შეტყობინების გადაცემათა ქსელში შემთხვევითი პროცესის არსებობა განპირობებულია დაყოვნების ცვლადი დროის გამო, რაც აისახება colset NetDelay=int with 25..75, fun DEL( ) =NetDelay.ran( ) random-ფუნქციით. ლოგიკური პირობის მნიშვნელობა სხვადასხვა შემთხვევებში სხვადასხვანაირად განისაზღვრება. ინტერაქტიულ სიმულატორებში ჭეშმარიტება-მცდარობას თავად მომხმარებელი განსაზღვრავს, ავტომატური სიმულაციისას – შემთხვევით რიცხვთა გენერატორი.

4.19 ნახაზზე ნაჩვენები გვაქვს ჩვენი ამოცანის პეტრის ქსელის ფრაგმენტი 50-ე ბიჯის და პროცესის დასრულების შემდეგ. ჩანს მარკერების შეცვლილი მდგომარეობა. თავიდან გაიშვება გადასასვლელი „*მოთხოვნის ტიპის დადგენა ბანკში*“ (ნახ.4.18 გადასასვლელი გააქტიურებულია - მწვანე ფერის ჩარჩო), ვინაიდან მის შესასვლელ პოზიციაში „*მოთხოვნა_მოსულია*“ მზადაა მარკერები. ესაა სიგნალი იმის შესახებ, რომ 1-ელი მოთხოვნით გათვალისწინებული შეტყობინება გაიშვება ქსელში “*ნოტიფიკაციის_გადაცემა_MOF-ში*”. გააქტიურდება ეს გადასასვლელი და მარკერი გადავა “*გაგზავნის*” A-პოზიციაში (n=1, p=”Motxovna_1”). ტრანსპორტირების გარკვეული დროის შემდეგ (სტოქასტიკური დრო: @+DEL( )) ნოტიფიკაცია მიაღწევს დამკვეთამდე და ა.შ.

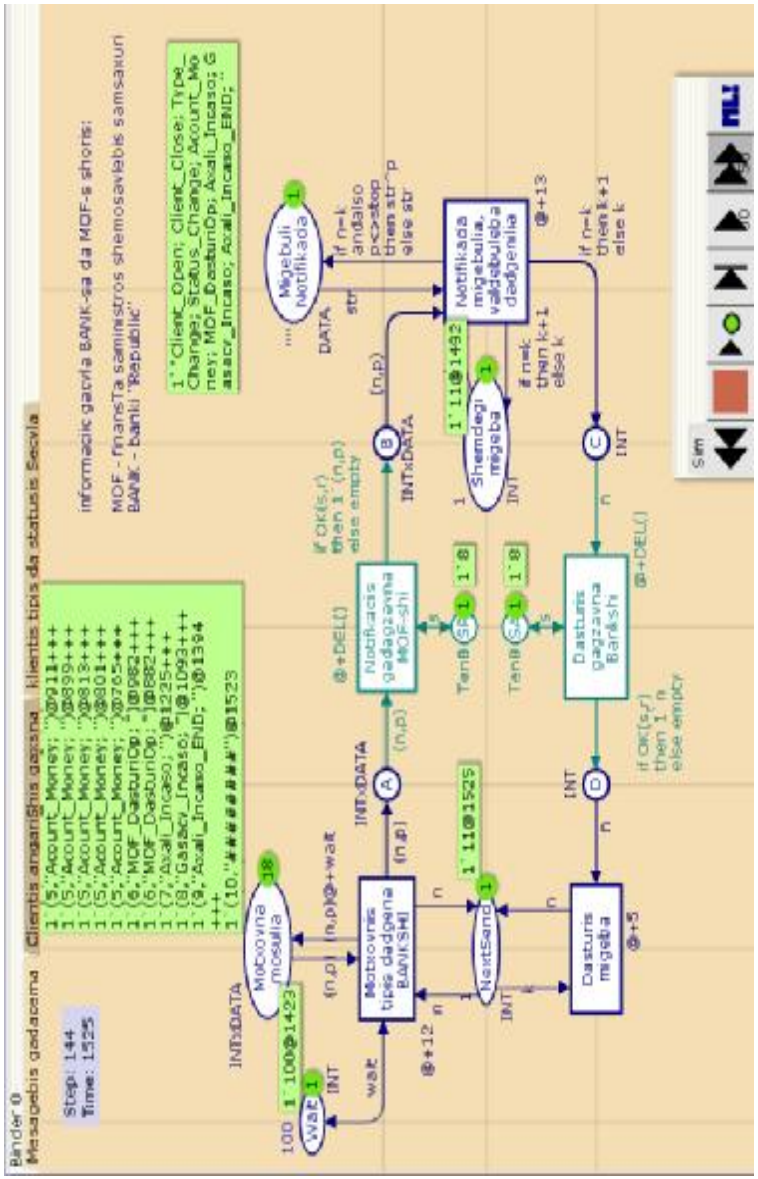


ნახ.4.19. ობიექტორიენტირებული მოდელირების შუალედური ეტაპი (ბიჯი=50)

ჩვენი დროითი CPN -მოდელით შეიძლება გამოვიკვლიოთ კორპორაციათა შორის (მაგალითად, ბანკსა და შემოსავლების სამსახურს შორის) შეტყობინებათა გაცვლის პროცესის მახასიათებლები. შეტყობინებათა განმეორებითი გადაცემის დაყოვნების დროის (wait) სხვადასხვა მნიშვნელობისათვის. ხანმოკლე დაყოვნება ზრდის შანსს განმეორებითი გადაგზავნების თავიდან ასაცილებლად. იგი ასევე ზრდის შანსს, რომ ოპერაცია Dasturis_migeba გადაიდოს, რადგან პროცესი Motxovnis tipis dadgena bankshi დაკავებულია განმეორებითი გადაგზავნით. გრძელი დაყოვნება ნიშნავს, რომ საჭირო იქნება დიდხანს ცდა, სანამ ბანკი დარწმუნდება, რომ შეტყობინება ან დასტური იქნა დაკარგული. სიმულაციის პროცესში, სხვადასხვა wait-მნიშვნელობით შეიძლება დადგინდეს ოპტიმალური მნიშვნელობა განმეორებითი გადაცემის დაყოვნებისათვის. 4.20 ნახაზზე პროცესი დასრულებულია.

როგორც აღვნიშნეთ, არაა გამორიცხული შემთხვევები, რომ შეტყობინება ვერ მივიდეს დროულად დანიშნულების ადგილას (გარკვეული ობიექტურ-სუბიექტური მიზეზების გამო), ან დაიკარგოს დასტურის შეტყობინება. ასეთ შემთხვევებში საჭიროა ინფორმაციის დროულად გამოკვლევა და არშესრულებული პროცედურის გამეორება.

პეტრის ქსელის გადასასვლელები, როგორებიცაა Motxovnis_tipis _dadgena_bankshi, Notifikaciis_gadagzavna_MOF-shi, Notifikacia_migebulia, valdebuleba_ dadgenilia, dasturis gagzavna bankshi და ა.შ. ხასიათდება დროითი დაყოვნებებით, რომლებიც ან კონსტანტური მნიშვნელობისაა, ან შემთხვევითი რიცხვების დიაპაზონიდან აიღება სისტემის მიერ. ამგვარად, CPN-ინსტრუმენტით შესაძლებელია მდგომარეობათა სივრცის ანგარიშის მთლიანი პროცესის სრული ავტომატიზაცია, რაც მნიშვნელოვნად აჩქარებს ქსელის დიაგნოსტიკის პროცესს მისი რეალურ ობიექტთან ადეკვატურობის შესახებ, ანუ რამდენად სწორად ასახავს მოდელი რეალური ობიექტის ყოფაქცევას.



ნახ.4.20. იმიტაციური მოდელირების დასასრული (ბიჯი=144)

დგომარეობათა სრული სივრცე – ორიენტირებული გრაფით აისახება, რომელშიც მწვერვალები შეესაბამება ქსელის დასაშვებ მარკირებებს, ხოლო რკალები – მოვლენებს დამაკავშირებელი ელემენტებით. ე.ი. M1 მდგომარეობიდან (მარკირებიდან) სისტემა გადადის M2 მდგომარეობაში, როდესაც არსებობს რკალი დამაკავშირებელი (n,p)-ელემენტით, სადაც n-ფერადი მარკერია, ხოლო p-ინფორმაციული ნაწილი.

პეტრის ქსელის ახალ მარკირებისთვის პირველი ბიჯის შემდეგ A-პოზიციაში გაჩნდა ახალი, 1-მარკერი, რომლის ფერი=1, მონაცემი="Client_Open". ამასთანავე ეს მარკერი მოვიდა ქსელის ამუშავებიდან t=12 დროითი ერთეულის (მაგ., წუთი) შემდეგ (ვინაიდან Motxovnis_tipis_dadgena_bankshi გადასასვლელის დროითი დაყოვნებაა @+12).

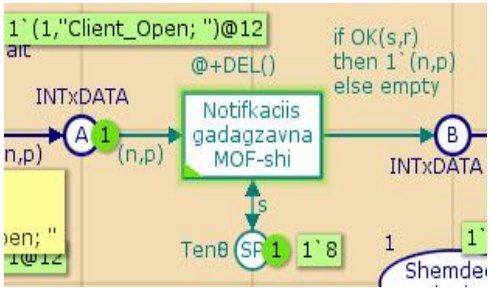
ახლა გააქტიურდა Notifikaciis_gadagzavna გადასასვლელი და შესაძლებელია ასევე Motxovnis_tipis_dadgena_bankshi გადასასვლელის ხელახალი გაშვებაც. ეს ორივე პროცესი შეიძლება შესრულდეს პარალელურად, ისინი ერთმანეთს ხელს არ უშლის.

Notifikaciis_gadagzavna გადასასვლელიდან B–პოზიციაში შემავალი რკალი აკონტროლებს ლოგიკურ პირობას, ანუ დასაშვებია ორი შემთხვევა (ნახ.4.21-ა):

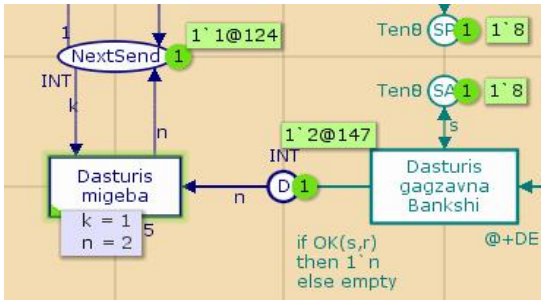
TP + =(Notifikaciis_gadagzavna, <n=1,p="Client_Open", success=true>),
TP - = (Notifikaciis_gadagzavna, <n=1,p=" Client_Open", success=false>).

ეს ორი დამაკავშირებელი ელემენტი TP+ და TP- იმყოფება კონფლიქტში ერთმანეთთან, ანუ ერთის შესრულება მეორეს გამორიცხავს. პირველით მოდელირდება ქსელში ნოტიფიკაციის წარმატებით გადაცემა, ხოლო მეორეთი კი – ამ შეტყობინების დაკარგვა.

ნახ.4.21-ა



მარკერი გადადის D-პოზიციაში და გააქტიურდება Dasturis_Migeba გადასასვლელი, რაც იმის მაუწყებელია, რომ ბანკის ამ ნოტიფიკაციაზე მოვიდა პასუხი MOF-დან (ნახ.4.21-ბ).



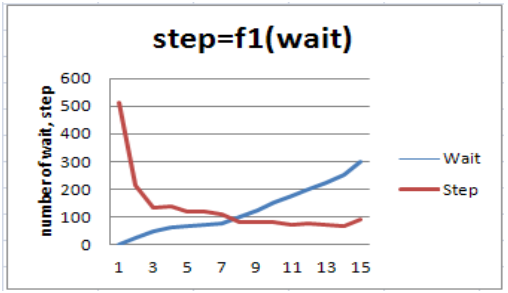
ნახ.4.21-ბ

შესაბამისად, ბანკი გააგრძელებს დასტურით მიღებული ინფორმაციის საფუძველზე მოქმედებას. მაგალითად, თუ ამ კლიენტს აქვს "*ვალდებულება*", მაშინ მას უხსნიან ანგარიშს, ოღონდ სტატუსით "*მხოლოდ ბიუჯეტური*". თუ არ აქვს ვალდებულება, მაშინ - "*ღია*" სტატუსით, და ა.შ. ქსელში გრძელდება სხვა მოთხოვნების (ნოტიფიკაციების) დამუშავება.

პეტრის ქსელის იმიტაციური მოდელის საფუძველზე (ნახ.4.20) პროცესების კვლევის შედეგად, როდესაც ვცვლით მოთხოვნების ანალიზის და შესრულების დასაწყისის დაყოვნების დროს (wait - პოზიცია) ინტერვალში [0-:300], მივიღეთ 4.4 ცხრილი და 4.22 ნახაზზე ნაჩვენები დიაგრამები.

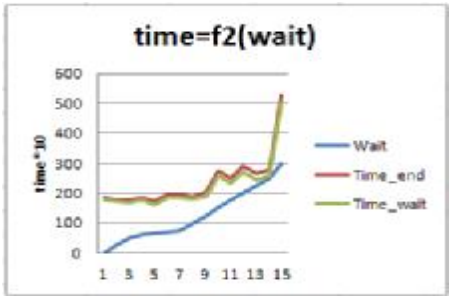
ცხრ.4.4

N	Wait	Step	Time_end	Time_wait
1	0	512	1856	1773
2	25	213	1759	1734
3	50	134	1771	1690
4	60	139	1833	1773
5	65	118	1714	1588
6	70	120	1977	1864
7	75	111	1956	1839
8	100	83	1887	1786
9	125	79	2036	1907
10	150	80	2729	2579
11	175	73	2514	2339
12	200	77	2893	2693
13	225	71	2686	2461
14	250	67	2794	2594
15	300	89	5289	4989



ნახ.4.22. იმიტაციის შედეგები: wait დროის ცვლილებით.

1: 15 ექსპერიმენტის ნომერი;
wait-დაყოვნების დრო;
step-ბიჯების რაოდენობა;
Time- სერვისების დამუშავების საბოლოო დრო



V თავი: კორპორაციული Web-აპლიკაციების მომხმარებელთა ინტერფეისების დაპროექტება და რეალიზაცია

5.1. ბიზნეს-პროცესების მართვის ინტერნეტული სისტემის აგება ASP+C# NET-ტექნილოგიით

ინტეგრაციული პროცესები და იმ მნიშვნელოვანი ინფორმაციის გამოყენების სურვილი, რომელიც დაგროვილია ისეთ საერთაშორისო საინფორმაციო ქსელში, როგორიცაა Internet-ი, საშუალებას გვაძლევს შევქმნათ ისეთი კომერციული ობიექტი, რომელიც იაფი სატრანსპორტო საკომუნიკაციო საშუალებების გამოყენებით მაქსიმალურ მოგებას მოგვიტანს [6].

თუმცა, ზემოთ აღნიშნული ობიექტის შექმნისას გვხვდება მთელი რიგი პრობლემებისა, რომელთა გადაჭრაც შესაძლებელია თანამედროვე საინფორმაციო და საკომუნიკაციო ტექნოლოგიების გამოყენებით. ინტერნეტ ქსელში, იგი წარმოადგენს სხვადასხვა სახის ბიზნეს ოპერაციების წარმოების კომპლექსურ სისტემას.

განაწილებული კორპორაციული სისტემების დაპროექტების ტექნიკური რეალიზაციის მხარე მოითხოვს მისი ცალკეული კვანძების ფუნქციური ანალიზის საფუძველზე აპარატული და პროგრამული უზრუნველყოფების ამოცანების გადაწყვეტას.

აპარატულში იგულისხმება კომპიუტერული და ქსელური ტექნიკა, რომლის საფუძველზედაც უნდა მოხდეს სისტემის გლობალურ / ლოკალურ ქსელში ფიზიკურად ჩართვის ორგანიზება. პროგრამული კი შეესაბამება ქსელში ჩართულ ოპერაციული სისტემის, პლატფორმის, საერთო-სერვისული გარემოს და კერძო-ფუნქციური პაკეტების ერთობლიობას [10].

ინტერკორპორაციულ ქსელებში ელექტრონული ფუნქციური კავშირების განსახორციელებლად საჭიროა შიგა და გარე

ინფორმაციული ნაკადების ანალიზის ჩატარება. არსებობს შემდეგი სახის ინფორმაციული ნაკადები (Workflow):

კორესპოდენცია და წერილები; ნორმატიული აქტები და კანონები; კონტრაქტები (ხანგრძლივი) და შეკვეთები (ერთჯერადი); კრედიტები და განვადებები, მონიტორინგი და კონტროლი, აგრეთვე ვალდებულებათა ფაქტობრივი შესრულებები და მათი ანალიზის მასალები; ინტერნეტიდან მიღებული ინფორმაცია (ფურცლები); აუდიო და ვიდეო ინფორმაცია (ელექტრონული გამოფენები, სალონები, პროდუქციის კატალოგები); საბანკო ანგარიშები, ბუღალტრული აღრიცხვა; კადრების აღრიცხვისა და შრომითი დასაქმების დოკუმენტაცია;

- ინფორმაცია პარტნიორებისა და კონკურენტების შესახებ;
- ინფორმაცია შიგა და გარე სერვისების შესახებ;
- სტატისტიკური ანალიზის მასალები.
- და სხვ.

ინფორმაციული ნაკადების მოცულობათა საანგარიშოდ მიღებულია ინფორმაციის ერთეულად:

- I_T : ერთი ნაბეჭდი A4 ფორმატის ფურცლის ტექსტური ინფორმაციის სიდიდე;
- I_A : ერთი აუდიო ინფორმაციის სიდიდე;
- I_V : ერთი ვიდეო ინფორმაციის სიდიდე;

პირობითად მივიღოთ, რომ I_T=4 Kb, I_A=20 Kb, I_V=30 Kb. ინფორმაციული ნაკადების ზომები დამოკიდებული იქნება კომერციული ობიექტების მასშტაბებზე. ინფორმაციული ნაკადების მოცულობების საანგარიშოდ შეიძლება ჩავატაროთ მიახლოებითი, გასაშუალებული გათვლები (თვის, კვარტლის, წლის და ხანგრძლივი პერიოდისთვის), რომელთა საფუძველზე შესაძლებელი იქნება საერთო ინფორმაციული ფონდის

მოცულობის შეფასება და მონაცემთა განაწილებული საცავის ფიზიკური მოწყობილობების საჭირო მეხსიერების დადგენა.

5.1 ნახაზზე მოცემულია ინტეგრირებული საინფორმაციო მართვის სისტემის გაშლილი ზოგადი სქემა. საპრობლემო სფერო (სს), ჩვენს შემთხვევაში განაწილებული კორპორაციული სისტემა - კომპლექსური ობიექტია.

სისტემის მომხმარებლები (სმ) კლასიფიცირდება მათი ფუნქციური დანიშნულების მიხედვით (მაგალითად, სისტემის ადმინისტრატორი, მონაცემთა საცავის ადმინისტრატორი, საბოლოო მომხმარებლები – ხელმძღვანელები და ფუნქციურ ქვედანაყოფთა სპეციალისტები და ა.შ.).

I_1 – ინტერფეისია საპრობლემო სფეროს და სისტემის მომხმარებლებს შორის.

საანგარიშო ფორმულები შემდეგი სახისაა:

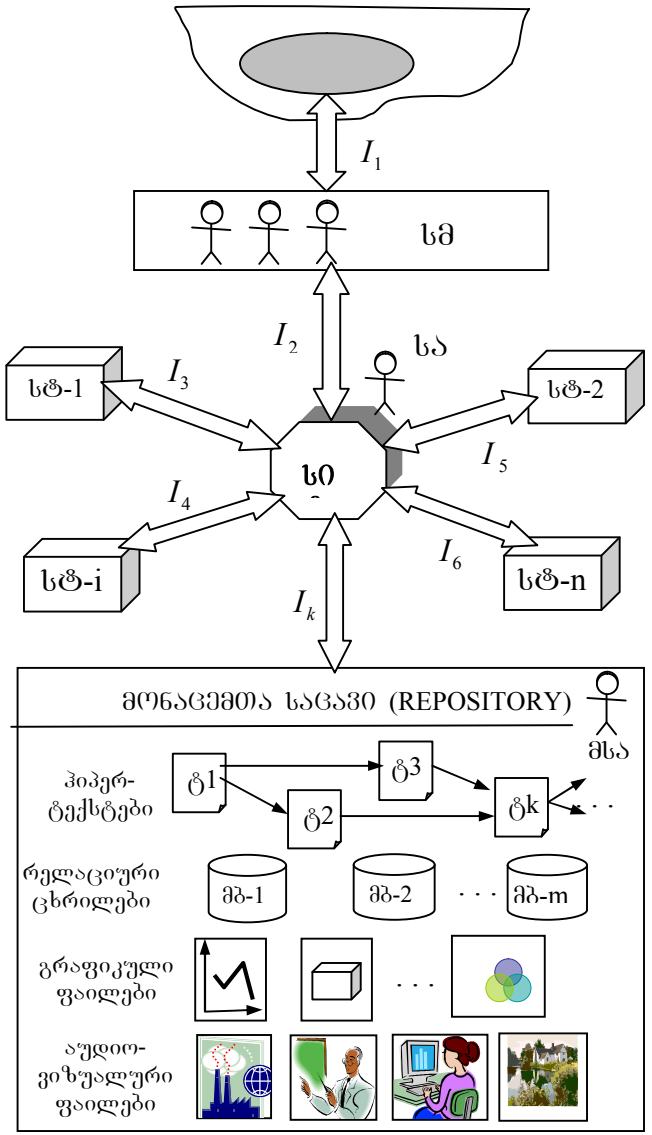
$$V_{jT} = k_j * \sum_{i=1}^{n} R_i * I_T \text{ , სადაც}$$

V_{jT} არის ტექსტური სახის ინფორმაციის თვიური, კვარტალური და წლიური დოკუმენტების ჯამური მოცულობა მეგაბაიტებში;
k_j – თვიური, კვარტალური და წლიური კოეფიციენტი (1, 3, 12);
R_i – ტექსტური დოკუმენტის A4-ფურცლების რაოდენობა;

აუდიო ინფორმაციული ნაკადებისათვის შესაბამისად გვექნება:

$$V_{jA} = k_j * \sum_{i=1}^{m} A_i \text{ , სადაც}$$

A_i – აუდიო ინფორმაციის ფაილის ზომაა;



ნახ.5.1

საჭიროა გავითვალისწინოთ ხმის გადაცემის მახასიათებელი, რომელიც საშუალოდ წარმოშობს 64 Kbit/sec წარმადობის ინფორმაციულ ნაკადებს.

ვიზუალურისათვის შესაბამისად გვექნება:

$$V_{jV} = k_j * \sum_{i=1}^{m} V_i \text{ , სადაც}$$

V_i – ვიდეო ინფორმაციის ფაილის ზომაა;

საჭიროა გავითვალისწინოთ, რომ ვიდეო გამოსახულების გადაცემა არქივირების გარეშე წარმოშობს 9.216 Mbit/sec, ხოლო არქივირებით 1.5 Mbit/sec წარმადობის ინფორმაციულ ნაკადებს.

მთლიანად ინფორმაციული ნაკადების ჯამური მოცულობა იქნება:

$$S = T_i * K_i * \sum_{j=1}^{n} V_j^i \text{ , სადაც}$$

T_i - i-ური კომერციული ობიექტის არსებობის მთლიანი პერიოდია (წლები) ;

K_i - i-ურ კომერციულ ობიექტზე ფილიალების რაოდენობაა;

V_j^i - i-ური კომერციული ობიექტის j-ური სახის ინოფორმაციული ნაკადის მოცულობა.

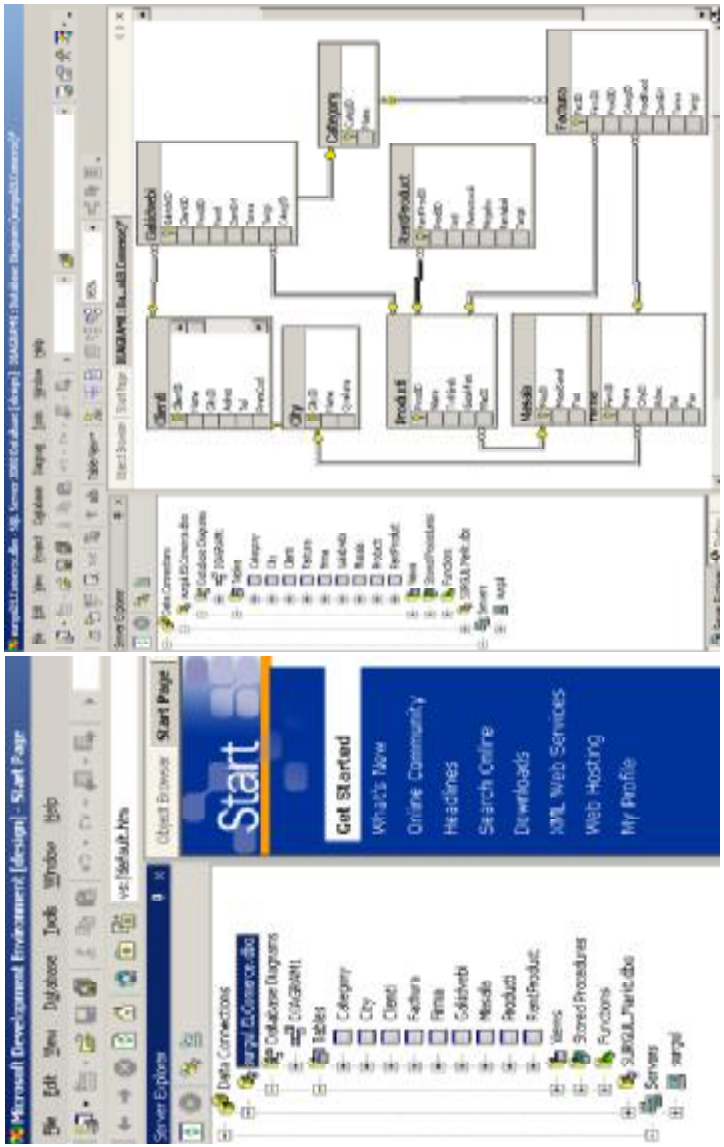
ექსპერტული ინფორმაციის საფუძველზე, როგორც ჩვენი პირობითი გათვლებიდან გამომდინარეობს, ერთ კორპორაციულ

ობიექტზე დაახლოებით 10 წლიანი არსებობის პერიოდში მონაცემთა საცავისათვის საშუალოდ დაგვჭირდება 250 GB–იანი მეხსიერება (გაითვალისწინება ძირითადი სტატისტიკური და ისტორიული ფაილების შენახვაც).

დიდი კორპორაციებისთვის ეს რიცხვი უნდა გამრავლდეს ობიექტების რაოდენობაზე და დაემატოს მათი მენეჯმენტისათვის საჭირო ინფორმაციული ნაკადი. ამგვარად, მონაცემთა საცავისათვის საჭირო ფიზიკური მეხსიერება მიაღწევს რამდენიმე ტერაბაიტ-მოცულობას.

მონაცემთა ბაზების ცხრილებიდან საჭრო ინფორმაციის ამოსაღებად გამოიყენება SQL (Structured Query Language სტრუქტურირებული მოთხოვნების ენა). ამისათვის იწერება მოთხოვნათა (Query) ფირმა მაიკროსოფტის მიერ დამუშავებულ SQL-ენის სტანდარტის შესაბამისად.

5.2 ნახაზზე მოცემულია ელექტრონული კომერციის სისტემის (ELCommerce) რეალიზებული მონაცემთა ბაზის ერთ-ერთი ფრაგმენტი MsSQL Server პაკეტის მაგალითზე.

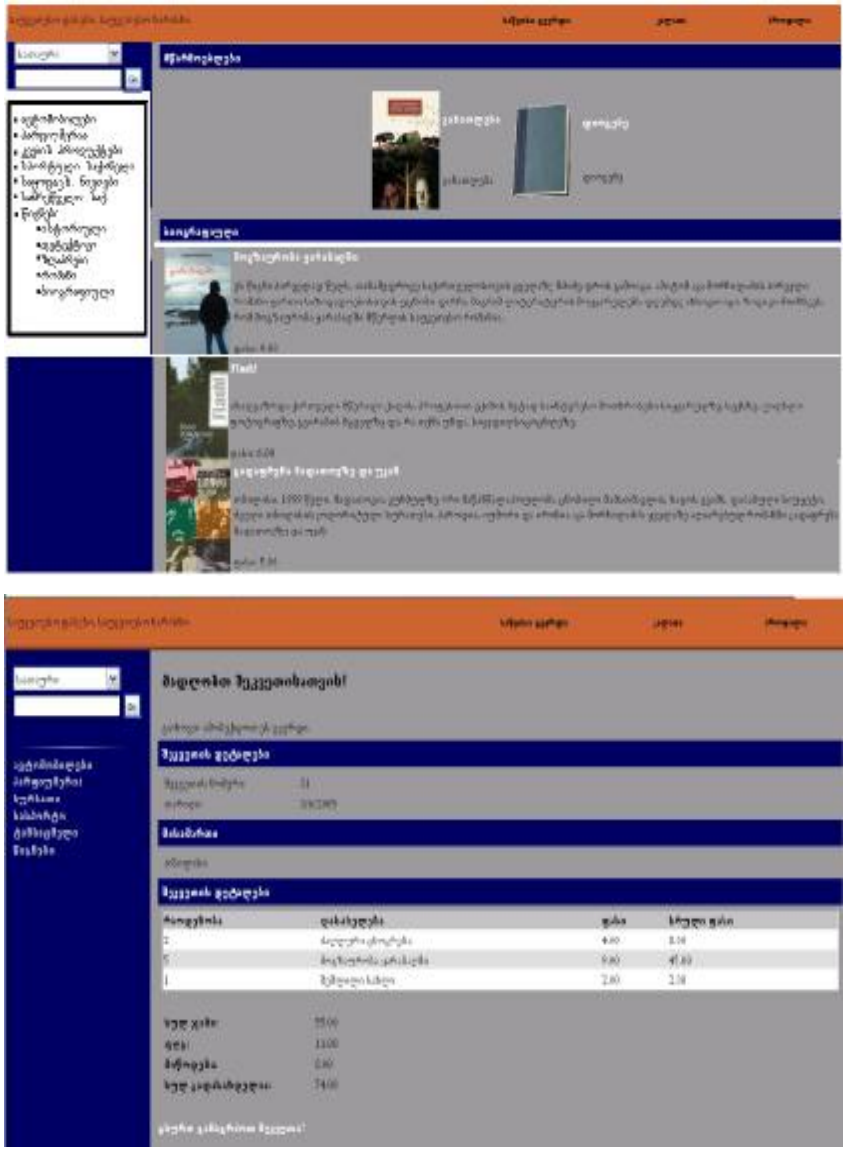


ნახ.5.2. მონაცემთა ბაზის სტრუქტურა

Internet ტექნოლოგიების გამოყენება, რომლის ერთ-ერთ სახესაც Web-გვერდების აგების ახალი ASP.NET ტექნოლოგია წარმოადგენს საშუალებას გვაძლევს შევქმნათ ისეთი საინფორმაციო გვერდი, რომელიც მომხმარებელს საშუალებას აძლევს შეარჩიოს მისთვის სასურველი სერვისი/პროდუქცია.

ASP.NET წარმოადგენს Web-გვერდების და მათზე მიბმული კოდის ფაილების ერთობლიობას. ამ მოდულში ხდება მომხმარებლის მიერ სხვადასხვა სერვისის/პროდუქციის ნახვა, ძებნა, არჩევა, შეკვეთა, მომხმარებლის რეგისტრაცია სისტემაში და პირადი მონაცემები შეტანა. იგი არის პლატფორმა, რომელიც საშუალებას გვაძლევს შევქმნათ მაღალი კლასის Web-პროგრამები. აგრეთვე იძლევა პროგრამირების ახლებური მოდელისა და ინფრასტრუქტურის საშუალებებს რათა შევქმნათ უფრო საიმედო, მასშტაბური და მდგრადი პროგრამები. ამავე დროს იგი უზრუნველყოფს გამოყენებითი სისტემების ეფექტურობას და მობილურობას Windows–ისა და სხვა პროგრამული პლატფორმებისაგან დამოუკიდებლად. 5.3 ნახაზზე მოთავსებულია ინტერნეტში განთავსებული სავაჭრო ცენტრის საინფორმაციო გვერდი.

Web-გვერდი ASP.NET სისტემაში, რომელიც SQL Server-თან კავშირშია, აღიწერება C# კოდით [10]. C# დაპროგრამების ენა .NET პლატფორმის ყველაზე მძლავრი ინსტრუმენტია, რომელიც კლასების საფუძველზე აგებს აპლიკაციებს. სისტემა იყენებს აგრეთვე XML და HTML ენებს. XML-მონაცემთა ფორმატირების გაფართოებადი ენა) თანამედროვე ინფორმაციული ტექნოლოგიების გამოყენებისას მეტად აქტუალურია [39]. ის არის სისტემების ინტეგრაციის მეტაენა, რომლითაც შესაძლებელია სხვადასხვა ტიპის მონაცემთა გადაცემა განსხვავებულ აპლიკაციებს შორის პლატფორმისგან დამოუკიდებლად.



ნახ.5.3 სავაჭრო ცენტრის საინფორმაციო გვერდი

XML დოკუმენტი არის ჩვეულებრივი ASCII ფაილი, რომელიც თავისუფლად გადაიცემა ინტერნეტში. იგი ტექსტური ტიპის პროტოკოლური ფაილია, რომელსაც ადვილად კითხულობს ადამიანიც და მანქანაც. XML დოკუმენტის ფორმატირების საშუალებას იძლევა XSL (eXtensible Stilesheet Language) სტილური ცხრილი. იგი იძლევა ბრაუზერის ეკრანზე ელემენტების გამოსახვის პროცესის მართვის და დოკუმენტში საჭირო ფრაგმენტების ძიების საშუალებას. სტილური ცხრილების დოკუმენტი მოიცავს აგების წესების ერთობლიობას, რომელთაგან თითოეული წესი დაყოფილია ცალკეულ ბლოკად, ორგანიზაციული ტეგებით.

5.2. სერვის–აპლიკაციების პროგრამული რეალიზაცია და გამოყენება Ms BizTalk Server გარემოში

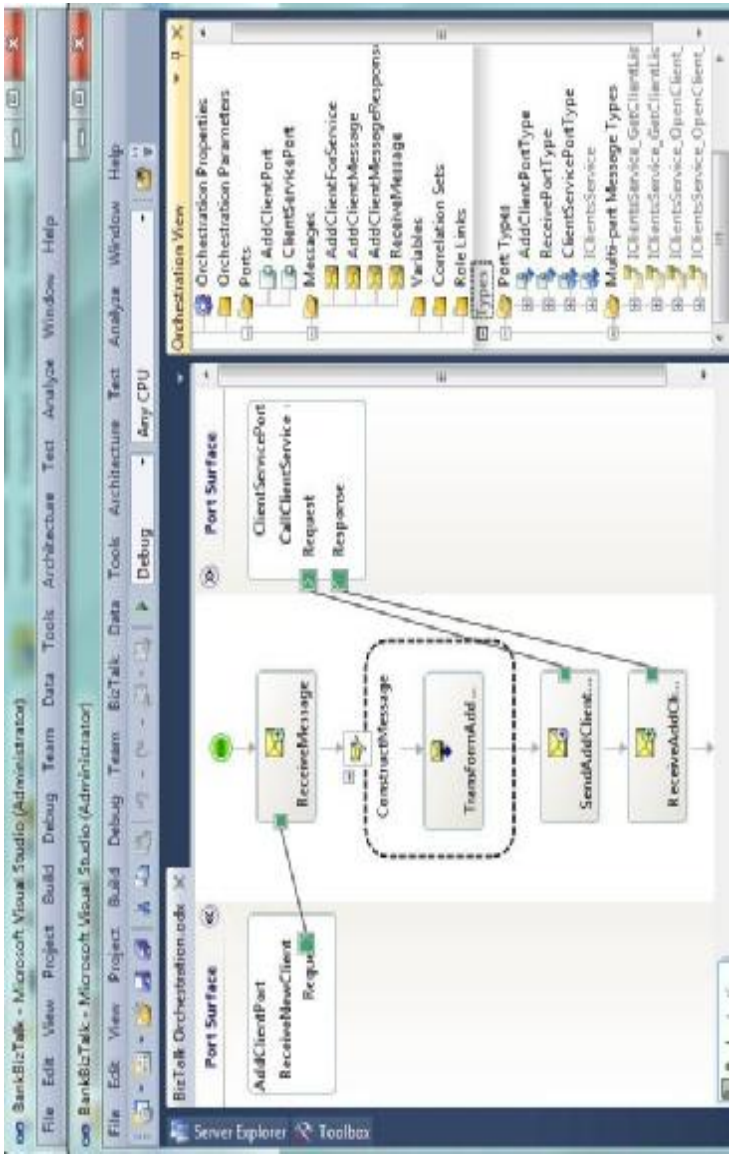
BizTalk სერვერი საშუალებას იძლევა მიიღოს და დაამუშაოს შეტყობინებები. აპლიკაციაში დამატებული ორკესტრაციის ფაილი Bank_orchestration.odx (ნახ.5.4).

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

```
<?xml version="1.0" encoding="utf-16"?>
        <xs:schema xmlns="http://BankBizTalk.Client"
        xmlns:b="http://schemas.microsoft.com/BizTalk/2003"
        targetNamespace="http://BankBizTalk.Client"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Root">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="CLIENT_NO" type="xs:unsignedInt" />
     <xs:element name="BRANCH_ID" type="xs:short" />
     <xs:element name="CLIENT_TYPE" type="xs:short" />
```

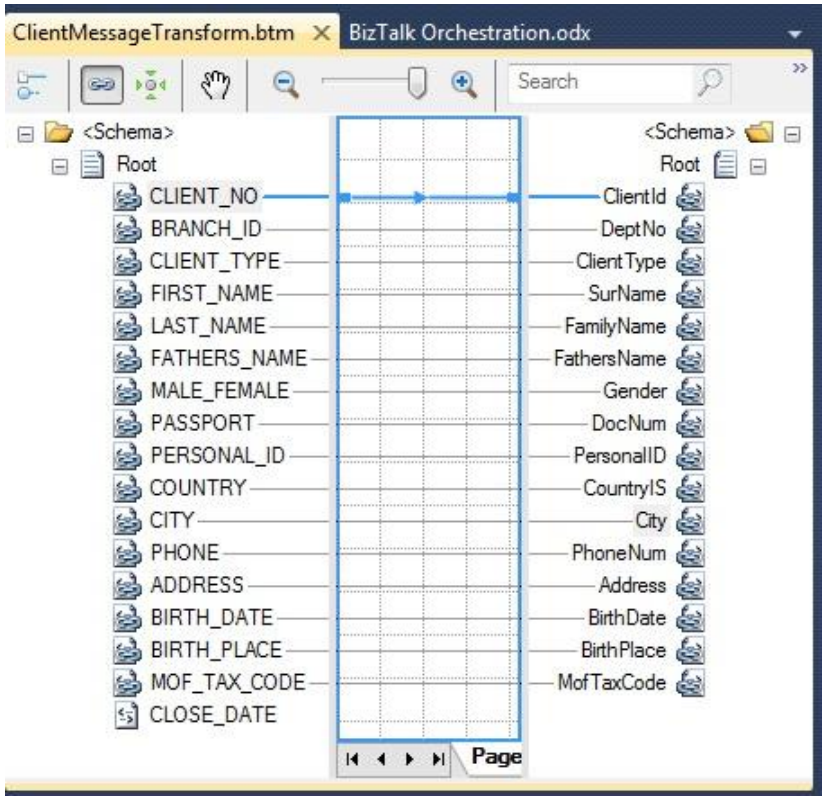
```
        <xs:element name="FIRST_NAME" type="xs:string" />
        <xs:element name="LAST_NAME" type="xs:string" />
        <xs:element name="FATHERS_NAME" type="xs:string" />
        <xs:element name="MALE_FEMALE" type="xs:boolean" />
        <xs:element name="PASSPORT" type="xs:string" />
        <xs:element name="PERSONAL_ID" type="xs:string" />
        <xs:element name="COUNTRY" type="xs:string" />
        <xs:element name="CITY" type="xs:string" />
        <xs:element name="PHONE" type="xs:string" />
        <xs:element name="ADDRESS" type="xs:string" />
        <xs:element name="BIRTH_DATE" type="xs:string" />
        <xs:element name="BIRTH_PLACE" type="xs:string" />
        <xs:element name="MOF_TAX_CODE" type="xs:string" />
        <xs:element name="CLOSE_DATE" type="xs:dateTime" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

მონაცემების მიღება ხდება პორტების საშალებით. პორტისას მიეთთება შეტყობინების ტიპი (Message Type). შეტყობინება კი თავის მხრივ მიეთთება ის სქემა რომლის შესაბამის XML-ის მიღებაც ხდება, Port>Message>Schema. ამ შემთხვევაში მითითებულია Clients.xsd. შეტობინების მიღების შემდეგ უნდა მოხდეს XML -ის ფორმირება რა ფორმტაითაც საჭიროა რომ იგი გადაეცეს Web-სერვის. ამისათვის გამოიყენება Transform კომპონენტი. მის შემავალ შეტყობინების ტიპში მიეთითება Clients.xsd და Clients4Service.xsd.



ნახ.5.4. BizTalk ორკესტრაციის ფაილი

ამ სქემებს შორის ურთიერთკავშირი აიგება ნახაზზე 5.5 მოცემული ინსტრუმენტის საშუალებით. მიღებული XML-ის გადაცემა ხდება ClientsService Web-სერვისისთვის. ამისათვის ორკესტრაციაში ემატება გაგზავნის შეტყობინება, რომელიც გადაცემს ახალ პორტს კლიენტის მონაცემებს. Web-სერვისის გამძახებისთვის საჭიროა პროექტს დაემატოს კავშირი Web-სერვისთან.



ნახ.5.5. XML ტრანსფორმაციის ხელსაწყო

ამისათვის გამოიყენება AddGeneratedItems ხელსაწყო. ნახაზზე 5.6 მოცემულია ამ ხელსაწყოს გამოძახება:

ნახ.5.6. AddGeneratedItems ხელსაწყო

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

BankBizTalk. ClientsService_tempuri_org.OpenClientResponse. ამ შეტყობინების მიღების შემდეგ ორკესტრაციაში ჩადებული ლოგიკა ასრულებს მუშაობას.

5.3. საფინანსო ბანკების Web-აპლიკაციის ინტერფეისების დამუშავება Internet-Intranet გარემოში .NET-პლატფორმაზე

განიხილება კორპორაციული მართვის სისტემების ვებ-აპლიკაციების დაპროექტების და რეალიზაციის საკითხები. საფინანსო ბანკის კლიენტთა მომსახურების მაგალითზე ილუსტრირებულია მათი ინტერფეისული კომპონენტების აწყობისა და მონაცემთა სერვერული ბაზების ორგანიზების ამოცანები. სისტემა დამუშავებულია .NET-პლატფორმაზე, C#, ASP.NET, ADO.NET და SQL Server ობიექტორიენტირებული ინსტრუმენტების გამოყენებით [9,11].

კორპორაციული სისტემები და მათი მართვის მექანიზმები ხასიათდება განსაკუთრებული სირთულით, დიდი ინფორმაციული ნაკადების ოპერატიულად დამუშავებისა და გადაწყვეტილების მიღების მცირე დროის არსებობის თვალსაზრისით, რაც აუცილებლად მოითხოვს ამ ორგანიზაციაში თანამედროვე საინფორმაციო ტექნოლოგიების დანერგვას.

ნაშრომში განსაკუთრებული ყურადღება ექცევა საფინანსო ბანკებში (და არა მხოლოდ აქ) საინფორმაციო-მომსახურების სისტემების დაპროექტებას WEB ტექნოლოგიით, ნაცვლად სტანდარტული Windows დანართებისა (აპლიკაციებისა). აქტუალურად მიგვაჩნია ასეთი სისტემების შემუშავება .NET-პლატფორმაზე, C#, ADO.NET და ASP.NET დაპროგრამების ვიზუალური, ობიექტორიენტირებული, ინტეგრირებული ინსტრუმენტების გამოყენებით [12,60].

დასაპროექტებელი მართვის ინფორმაციული სისტემის ძირითადი მოთხოვნები ასე განისაზღვრა:

- ინფორმაციის უსაფრთხოების და დაცვის მაღალი დონე;
- სისტემასთან ურთიერთობის გამარტივებული და მისი მომსახურების სიადვილე;

- სისტემის მომხმარებლებთან ურთიერთობის მეგობრული ინტერფეისის არსებობა.

ნაშრომში შემოთავაზებულია საბანკო სისტემაში ვებ-ტექნოლოგიაზე დაფუძნებული ინფორმაციული სისტემების დაპროექტების და მისი შემდგომი რეალიზაციის საკითხები. ისეთ საფინანსო ორგანიზაციის ფილიალებში, როგორიცაა ბანკი, ინტერნეტ- ინტრანეტის პირობებში გაიზრდება უსაფრთხოება და სისტემის დაცვა. ინტრანეტი იცავს ორგანიზაციას თავისი შიდა ფაილებისა და კონფიდენციალური ინფორმაციის წვდომისაგან გარე პირთათვის. იგი ხშირად გამოიყენება ფაილების და მეილების ორგანიზაციის წევრთათვის ერთობლივი წვდომისათვის, და ამავე დროს გარე პირთათვის იგივე ინფორმაციის ბლოკირებისათვის.

კორპორაციის შიგა Web-პროგრამები, რომელიც შეზღუდულია სპეციფიკური მომხმარებლისა თუ კომპიუტერებისათვის, დიდად გამოსაყენებელია ფინანსურ ბანკებში, რამეთუ შიგა ქსელის დამონტაჟებით და ინტრანეტის გაყვანით, ყველა აუცილებელი ოპერაცია სრულდება და ამავდროულად ვირუსების, ტროიანების და ა.შ. საფრთხე მკვეთრად მცირდება.

ჩვენ ვიხილავთ კლიენტ-სერვერულ სისტემას, სადაც გვაქვს მხოლოდ ერთი დიდი სერვერი, რომელიც მოთავსებულია ბანკის (პირობითად) სათაო ოფისში. დანარჩენ ფილიალებში მოთავსებული კომპიუტერები კი წარმოადგენს კლიენტებს. ანუ საქმე გვაქვს ცენტრალიზებულ სისტემასთან, რაც მკვეთრად ამცირდებს ადმინისტრირების ხარჯებს.

Windows-პროგრამების შემთხვევაში სისტემის „მწყობრიდან გამოსვლის" სიხშირე გაცილებით მაღალია, ამიტომაც საჭიროებს მუდმივ თვალყურისდევნებას ადმინისტრატორის მიერ.

ამ შემთხვევაში აუცილებელია პროგრამული უზრუნველ-ყოფის დაყენება ყველა მომხმარებლის კომპიუტერზე. ინსტალირების პროცესი მოითხოვს დროს, ხოლო ახალი ვერსიების გამოსვლა კი ხელს უწყობს ამ დროის გაზრდას. რადგან როდესაც პროგრამული უზრუნველყოფის ახალი ვერსია გამოდის საჭიროა მისი ყოველ კომპიუტერზე ხელახალი დაინსტალირება, რაც თავისთავად გამოიწვევს დროის ხარჯვას.

Web-პროგრამის შემთხვევაში ყველა ეს პრობლემა იხსნება. რადგან ვებ პროგრამა ჩაწერილია სერვერზე, და პროგრამისტთა ერთი ჯგუფიც კი საკმარისია, რათა ყველა ხარვეზი ადგილზევე, სერვერზევე აღმოიფხვრას.

როდესაც ვინდოუსის პროგრამებთან გვაქვს საქმე, ისიც უნდა გავითვალისწინოთ, რომ იმ კომპიუტერზე, სადაც ინსტალირდება ეს პროგრამა, უნდა იყოს დაინსტალირებული მთელი რიგი სხვა პროგრამები, რათა ამ უკანასკნელმა იფუნქციონიროს. ჩვენს მიერ შემოთავაზებული Web-პროგრამისთვის კი სულერთია, კლიენტის კომპიუტერზე რომელი ოპერაციული სისტემა იქნება დაინსტალირებული, ის ერთნაირად კარგად იმუშავებს, როგორც Windows-ის, ასევე Linux და Unix ოპერაციულ სისტემებში [39].

Web-პროგრამა აგებული გვაქვს Visual Web Dedevoper 2005 Express-ის საშუალებით, რომელიც არის Microsoft Visual Studio 2005 ოჯახის წევრი და ASP.NET-ზე ვებ პროგრამების დაწერის საუკეთესო საშუალებას იძლევა. როგორც Express გამოცემა, ის წარმოადგენს Visual Studio Standard–ის უფრო გაუმჯობესებულ ვარიანტს [9].

Visual Web Developer სპეციალურად აწყობილია ვებ პროგრამების დასაწერად და იყენებს ახალ Web-პროფილს, რომელიც მენიუს და ფანჯრის Web-პროგრამირებისთვის

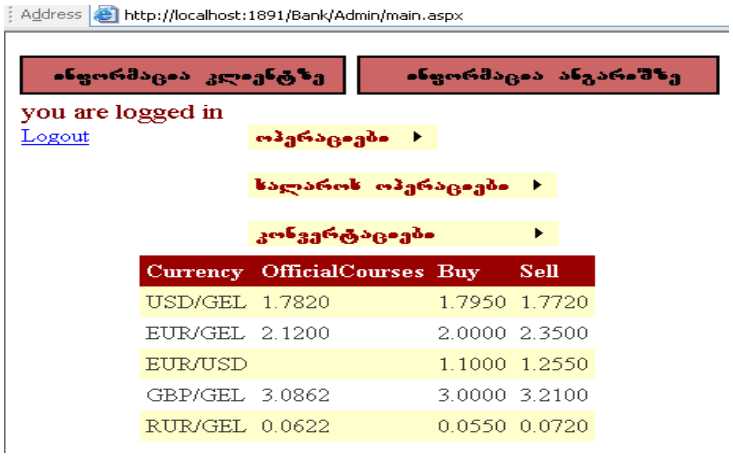
ოპტიმიზირებულ ვარიანტებს გვთავაზობს. პროგრამირების გარემო შეიცავს HTML კოდის რედაქტორს, ვებ გვერდების გაუმჯობესებულ დიზაინერს, ახალ საპროექტო სისტემას, მონაცემებთან მუშაობის უკეთეს მხარდაჭერას და XHTML-ის მთლიან მხარდაჭერას. ერთიანად ეს თვისებები აძლევს პროგრამისტს საშუალებას სწრაფად, ადვილად და ეფექტურად გამოიმუშაოს ვებ პროგრამა.

ვებ პროგრამის კოდის ძირითადი ნაწილი დაწერილია C#-ზე და ASP.NET-ზე. კოდის დინამიკური ნაწილი ASP-ზე, ხოლო სტატიკური ნაწილი HTML-ზე. მონაცემთა სერვერ-ბაზად შერჩეულია MS SQL Server, ხოლო დაპროგრამების ენად C#.NET. MsSQL Server პაკეტი კორპორაციული სისტემებისთვის დღეისათვის მეტად ეფექტურია მასშტაბურობის, სწრაფქმედების და მწარმოებლურობის თვალსაზრისით [60].

ახლა განვიხილოთ კონკრეტული საილუსტრაციო მაგალითი ჩვენი სისტემიდან, კერძოდ კლიენტებისათვის საბანკო ანგარიშზე თანხის შეტანის (ან გატანის) მაგალითისათვის. იმისათვის, რომ სისტემა დაცული იყოს არასანქციური შეღწევებისაგან, გამოვიყენეთ აუტენტიფიკაციის მექანიზმი, რისთვისაც პირველ რიგში მომხმარებელი შეიტანს თავისი იდენტიფიკატორს და პაროლს.

სწორი აუტენტიფიკაციის შემდეგ მომხმარებელი მიიღებს სისტემის მთავარ გვერდს, რომელზეც გამოსახულია ვალუტის გაცვლის კურსი, მენიუ და ღილაკები (ნახ.5.7).

„ინფორმაცია კლიენტზე" ღილაკით ოპერატორი გადავა Web-გვერდზე, რომელზეც მას შეუძლია იხილოს ინფორმაცია კონკტერულ კლიენტზე. ღილაკით „ინფორმაცია ანგარიშზე" გამოყენებისას ოპერატორი მოხვდება გვერდზე, რომელზეც მას შეუძლია ნახოს კლიენტის ანგარიშები.

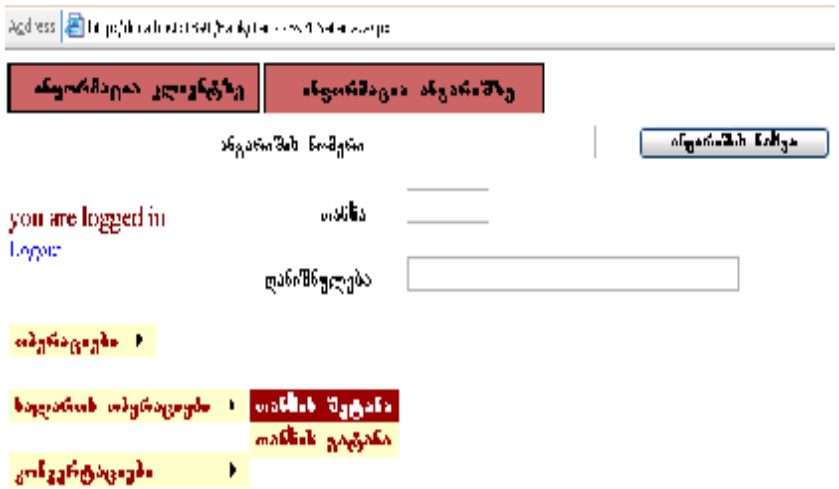


ნახ.5.7. ინფორმაცია კლიენტზე - მთავარი გვერდი

თუ ოპერატორს სჭირდება ინფორმაცია კლიენტის ანგარიშების შესახებ, მას შეუძლია გადავიდეს გვერდზე „ინფორმაცის ანგარიში", შესაბამისი სახელწოდების ღილაკით. კლიენტის მონაცემების (სახელი და გვარი) შეტანით, სპეციალურად განკუთვნილ ტექსტურ ველებში, ის იხილავს ინფორმაციას კლიენტის ანგარიშების შესახებ. გამოტანილი იქნება შემდეგი ინფორმაცია: სახელი, გვარი, ანგარიშის ნომერი, ანგარიშის ტიპი, ბალანსი, ოვერდრაფტი, კრედიტი და ვალუტის ტიპი (ნახ.5.8).

ნახ.5.8. გვერდი: "ინფორმაცია ანგარიშზე"

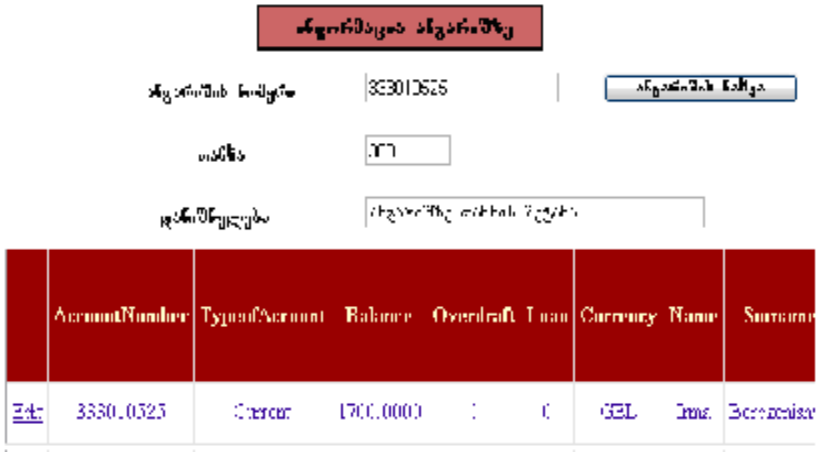
თუ კლიენტს სურს თანხის შეტანა ანგარიშზე, ამ შემთხვევაში ოპერატორი უნდა გადავიდეს გვერდზე „თანხის შეტანა". აქ ოპერატორს აქვს საშუალება გადაამოწმოს კლიენტის ანგარიშები, რათა დადასტურდეს, რომ ის ანგარიში, რომელზეც კლიენტს შემოაქვს ფული, არსებობს. ეს შესაძლებელია ანგარიშის ნომრის მითითებით და შემდეგ ღილაკით „ანგარიშის ნახვა". გამოტანილი იქნება ინფორმაცია მითითებული ანგარიშის შესახებ. თანხის შეტანისას ველში „თანხა" ოპერატორმა უნდა მიუთითოს თანხის მნიშვნელობა და დანიშნულება (ნახ.5.9).



ნახ.5.9. გვერდი: "თანხის შეტანა"

თანხის შეტანის დროს კლიენტის ანგარიშზე არსებული ბალანსი იცვლება. ოპერატორმა უნდა მოახდინოს ბალანსის რედაქტირება, რაც შესაძლებელია Edit ღილაკით. შესატანი თანხის

მითითებით და update ღილაკით ოპერატორი ახორციელებს ცვლილებების შეტანას მონაცემთა ბაზაში კლიენტის ბალანსის შესახებ, კერძოდ, შეტანილი თანხა ემატება არსებულ ბალანსს (ნახ.5.10).



ნახ.5.10. თანხის შეტანის საბოლოო შედეგი

იგივე ხდება თანხის გატანისას, ოღონდ საპირისპირო ნიშნით. არსებულ ბალანსს აკლდება გამოტანილი თანხა.

დასასრულ, შეიძლება დავასკვნათ, რომ ვებ-დანართების აგების პროცესების ავტომატიზაციით მიიღება მაღალი ხარისხის საიმედო პაკეტები. განსაკუთრებით ახალი საინფორმაციო ტექნოლოგიების, როგორიცაა NET-პლატფორმა და C#, ASP.NET.

5.4. Web-აპლიკაცია: სოციალური მომსახურების სისტემის კომპიუტერული მონიტორინგის სისტემა

განხილულია სოციალური მომსახურების სისტემისთვის ბენეფიციართა აღრიცხვისა და კომპიუტერული მონიტორინგის ლოგიკურად ერთიანი მონაცემთა ბაზის სტრუქტურების ავტომატიზებული დაპროექტების საკითხები ობიექტ-როლური მოდელირების (ORM/ERM) მეთოდით. არსებული სისტემის ბიზნეს-პროცესების ანალიზის და UML-მოდელირების საფუძველზე შემოთავაზებულია სოციალური მომსახურების საპრობლემო სფეროს Web-სერვისული სისტემის შემუშავების კონცეფცია და მომხმარებელთა ვიზუალური ინტერფეისის რეალიზაცია .NET პლატფორმაზე ASP/XML/C# ინსტრუმენტული საშუალებებით [8].

საქართველოს შრომის, ჯანმრთელობისა და სოციალური დაცვის სამინისტროს სტრუქტურაში ერთ-ერთი მნიშვნელოვანი ადგილი უჭირავს სოციალური მომსახურების სააგენტოს, რომლის Web-გვერდის ww.ssa.gov.ge ფრაგმენტი ნაჩვენებია 5.11 ნახაზზე. დღეისათვის სააგენტო უზრუნველყოფს 20-მდე სხვადასხვა სახის სოცილური მომსახურებისა და გასაცემლის ადმინისტრირებას, მის სააღრიცხვო სიაში 1,8 მილიონზე მეტი ბენეფიციარია.

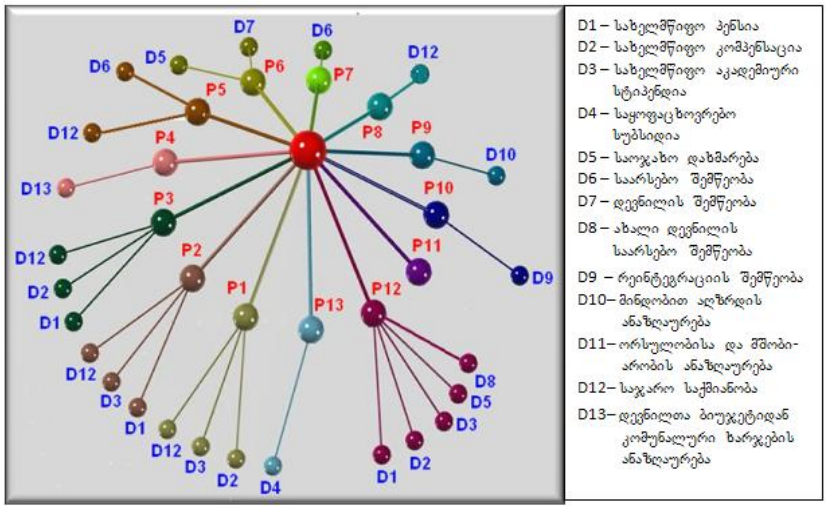
მიღებულია პოლიტიკური გადაწყვეტილება იმის შესახებ, რომ ყველა სახის სოციალური მომსახურება განხორციელდეს სოციალური მომსახურების სააგენტოს მიერ. შესაბამისად დადგა მისი მენეჯმენტის რეფორმის საკითხი, ანუ რეინტეგრაციის პრობლემის გადასაწყვეტად, და სოციალურ სფეროში მიმდინარე პროცესების დაგეგმვის, აღრიცხვისა და სხვადასხვა ჭრილში გაანალიზების მიზნით, ზოგადად, მომსახურების დონის ასამაღლებლად და ჰუმანური პრინციპების (ბიუროკრატიულის ნაცვლად) დასანერგად.

ნახ.5.11. ww.ssa.gov.ge

იმისათვის, რომ ახალი სისტემის მომზადების და დანერგვის პროექტი იყოს წარმატებული, აუცილებელია წინასწარ ჩამოყალიბდეს და მკაფიოდ განისაზღვროს ამ სისტემის მიმართ რეალურად არსებული ბიზნეს მოთხოვნები, რომელთა საფუძველზეც განხორციელდება საპროექტო და დეველოპმენტის ამოცანები [61].

თანამედროვე საინფორმაციო ტექნოლოგიები შესაძლებლობას იძლევა პროცეს-ორიენტირებული, ობიექტორიენტირებული მეთოდებისა და მოდელირების ვიზუალური ინსტრუმენტების საშუალებით განხორციელდეს რთული ბიზნეს-პროცესებისათვის მართვის სისტემის მოდელის აგება და შემდგომი კვლევა. მხედველობაში გვაქვს ისეთი პროგრამული პლატფორმები და სისტემები, როგორიცაა მაგალითად, ლოგიკურად ერთიანი განაწილებული მონაცემთა ბაზები, .NET-ის ინტეგრირებული ენები, ობიექტ-როლური მოდელი-ORM, უნიფიცირე-ბული მოდელირების ენა UML, ბიზნეს-პროცესების მოდელირების ნოტაცია - BPMN, ფერადი პეტრის ქსელები – CPN და სხვ. [12,15,36,56].

სოციალური მომსახურების სისტემაში, მაგალითად, ერთ-ერთი მნიშვნელოვანი საკითხია - განისაზღვროს, ეკუთვნის თუ არა სოციალური ბენეფიტის მაძიებელს ესა თუ ის სოციალური ბენეფიტი, რადგანაც პერსონა შეიძლება უკვე ღებულობდეს რომელიმე დახმარებას (მონიტორინგის პრობლემა). სულ განიხილება 13 სახის სოციალური ბენეფიტი (ნახ.5.12):



ნახ.5.12. ბენეფიტების ურთიერთგამორიცხვის მოდელი

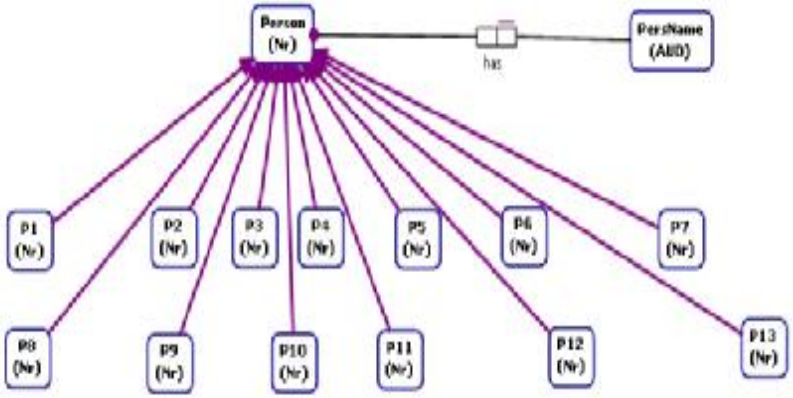
მოდელის ცენტრში წარმოდგენილია ობიექტი – პერსონა, რომელსაც შეიძლება ჰქონდეს რომელიმე ბენეფიტი (Pi , i=1,13). პერსონას შეიძლება ჰქონდეს რამდენიმე სხვადასხვა ბენეფიტი. საქართველოში არსებული დღევანდელი კანონმდებლობით ზოგიერთი ბენეფიტი გამორიცხავს სხვა სახის ბენეფიტის მიცემას ერთიდაიმავე ბენეფიციარზე.

გამორიცხვის ასეთი დამოკიდებულება ნახაზზე ასახულია Dj , j=1,13 კავშირით შესაბამის გამომრიცხავ Pi - თან. მაგალითად, თუ პერსონა ეწევა საჯარო საქმიანობას P12 (მსახურობს

სახელმწიფო ორგანიზაციაში), მაშინ იგი ვერ მიიღებს D1, D2, D3, D5 და D8 სახის ბენეფიტებს. ობიექტ-როლური მოდელის (ORM) ასაგებად განხორციელდა საპრობლემო სფეროს აღწერა ფაქტების საფუძველზე, მაგალითად:

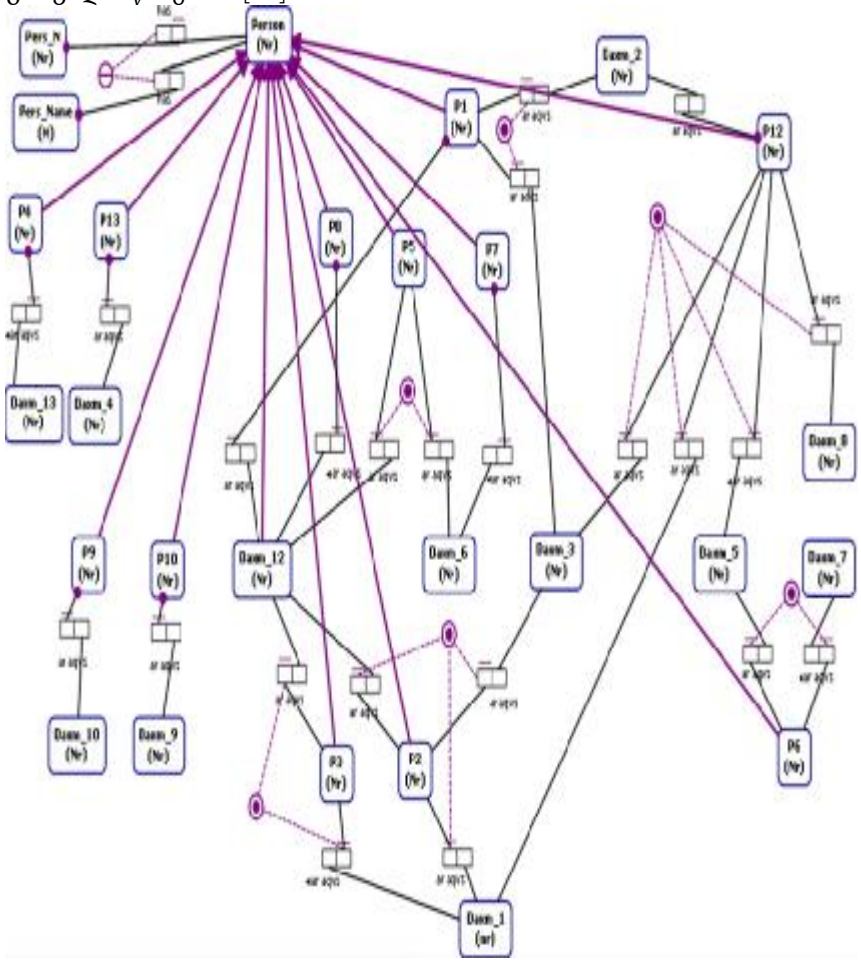
F1: პერსონას ***აქვს*** გვარი;
F2: პერსონას ***აქვს*** პირადობის მოწმობა;
F3 პერსონას გვარი და პირადობის მოწმობა უნიკალურად ***აღწერს*** პერსონას;
F4 სოციალური ბენეფიტი P1-ის მფლობელი პერსონა ***არის*** ობიექტ Person-ის ქვეტიპი;
. . .
F16: სოციალური ბენეფიტი P13-ის მფლობელი პერსონა ***არის*** ობიექტ Person-ის ქვეტიპი;
F17: სოციალური ბენეფიტის P1-ის მფლობელი პერსონას ***არ უნდა ჰქონდეს*** სახელმწიფო კომპენსაცია D2;
და ა.შ..

ობიექტ-როლურ მოდელში გამოყენებული შეზღუდვები ნათლად აღწერს საპრობლემო სფეროს დამახასიათებელ პროცესებს. 5.13 ნახაზზე წარმოდგენილი ქვეტიპის შეზღუდვა საშუალებას იძლევა ობიექტი Person დავყოთ 13 ნაწილად.



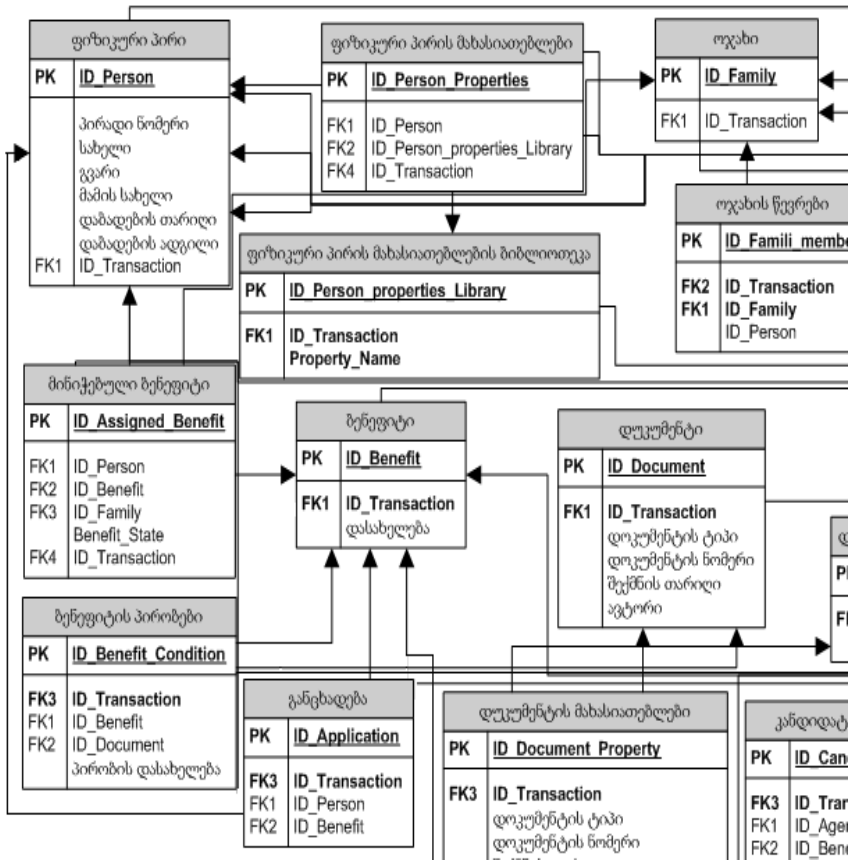
ნახ. 5.13. ORM-დიაგრამა Person ქვეტიპის შეზღუდვით

5.14 ნახაზზე მოცემული ORM-დიაგრამა აღწერს ურთიერთგამომრიცხავი სოციალური ბენეფიტების ერთობლიობას თითოეული შემთხვევისათვის, შეზღუდვათა ტიპების გათვალისწინებით [36].



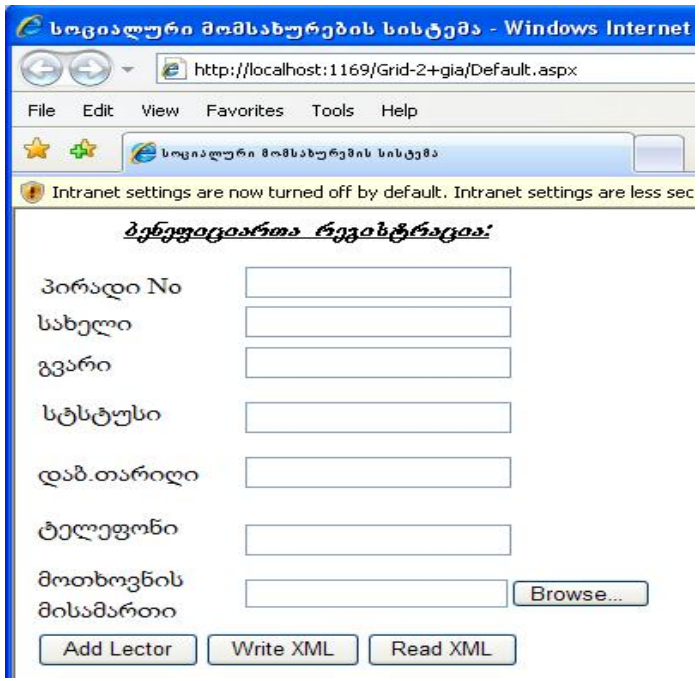
ნახ.5.14. ORM დიაგრამის ფრაგმენტი სოცბენეფიტების მოდელისთვის

5.15 ნახაზზე ილუსტრირებულია არსთა-დამოკიდებულების ER-მოდელის ფრაგმენტი, რომელიც საპრობლემო სფეროს კონცეპტუალური სქემის როლს ასრულებს ჩვენი სისტემისთვის. მის საფუძველზე აიგება MsSQL Server-ის DDL-ფაილი და შეიქმნება თვით მონაცემთა ბაზა.



ნახ.5.15. ER დიაგრამის ფრაგმენტი

მონაცემთა ბაზაში ინფორმაციის, კერძოდ ბენეფიციართა მონაცემების შესატანად და შემდგომ სხვადასხვა ფუნქციონალობის შესასრულებლად, დამუშავდა მომხმარებელთა სპეციალური ინტერფეისები, რომლებიც კლიენტ-სერვერულ არქიტექტურაზეა გათვლილი. 5.16 ნახაზზე ნაჩვენებია, მაგალითად, ბენეფიციართა რეგისტრაციის ინტერფეისი, რომლის გამოყენება შეუძლია როგორც სოცსამსახურის თანამშრომელს, ასევე თვით მოქალაქეს. მაგალითისათვის, WriteXML-ღილაკის კოდი მოცემულია 5.1 ლისტინგში.



ნახ.5.16. ბენეფიციართა რეგისტრაციის ინტერტფეისი

```
// ლისტინგი_5.1 ————————————————————————
protected void Button2_Click(object sender, EventArgs e)
  {
    DataSet ds = Session["MyDataSet"] as DataSet;
    ds.WriteXml(Request.PhysicalApplicationPath +
                                          "\\lectors.xml");
    //Response.Redirect("~/lectors.xml");
  }
```

სოციალური მომსახურების სისტემების კომპიუტერიზაცია ბევრად შეუწყობს ხელს როგორც სახელმწიფოს ერთიანი ხელისუფლების შექმნას და გამჭვირვალე, დემოკრატიული პროცესების დანერგვას, ასევე სოციალურ სფეროში არსებული მნელადფორმალიზებადი ბიზნეს-პროცესების მოწესრიგებას, რაც საბოლოო ჯამში, აამაღლებს ჰუმანური და სამართლიანი გადაწყვეტილებების მიღების დონეს.

5.5. Web-ის უსაფრთხოება ASP.NET-ში. სერვერის, კლიენტების, ფორმების და როლების აუთენტიფიკაცია

ვებ აპლიკაციებში ხშირად საჭიროა მომხმარებელთა იდენტიფიკაცია და მათთვის სხვადასხვა რესურსებზე წვდომის უფლების განსაზღვრა. ASP.NET-ში არსებობს აუთენტიფიკაციის რამდენიმე მეთოდი: Windows, Forms, Passport. ეს მოთოდები განისაზღვრება კონფიგურაციის ფაილში `authentication` ტეგის `mode` ატრიბუტის საშუალებით [12]:


```
<configuration>
      <system.web>
            <authentication/>
      </system.web>
</configuration>
```

მეთოდები:

```
<authentication mode="Windows" />
<authentication mode="Forms" />
<authentication mode="Passport" />
<authentication mode="None" />
```

ვებ–აპლიკაციაზე მომხმარებლის წვდომის უფლებას განსაზღვრავს `authentication` ელემენტი, ხოლო აპლიკაციის გარკვეულ ნაწილებზე განისაზღვრება `authorization` ელემეტით: deny და allow ქველემენტების საშუალებით:

*** – განსაზღვრავს ყველა მომხმარებელს,**
? – ანონიმურ მომხმარებლებს.

მაგალითად, ანონიმურ მომხმარებელთათვის საიტზე წვდომის უფლების გასათიშად Web-საიტის კონფიგურაციის ფაილში ჩაიწერება:

```
<configuration>
 <system.web>
  <authorization>
   <deny users="?" />
  </authorization>
 </system.web>
</configuration>
```

შესაძებელია ცალკეულ მომხმარებელზე და მომხმარებელთა ჯგუფებზე წვდომის უფლების მინიჭება. შემდეგი ჩანაწერი ნიშნავს, რომ წვდომის უფლება აქვთ someone და Admins ჯგუფში შემავალ მომხმარებლებს:

```
<authorization>
    <allow users="someone" />
    <allow roles="Admins" />
    <deny users="*" />
</authorization>
```

შესაძლებელია აგრეთვე ცალკეულ Web-გვერდებზე წვდომის უფლებების დაყენებაც:

```
<location path="webpage.aspx">
 <authorization>
  <allow roles="managers" />
  <deny users="?" />
 </authorization>
</location>
```

წინა ფრაგმენტი გვიჩვენებს, რომ webpage.aspx წვდომის უფლება აქვთ მხოლოდ managers ჯგუფის მომხმარებლებს.

Forms მეთოდით მომხმარებელთა ავტორიზაციის დროს საჭიროა მომხმარებლის სარეგისტრაციო გვერდის მითითება:

```
<configuration>
 <system.web>
  <authentication mode="Forms">
   <forms name=".ASPXCOOKIE" loginUrl="login.aspx"
protection="All"
            timeout="30" path="/"></forms>
  </authentication>
 </system.web>
</configuration>
```

Web- საიტზე მომხმარებლების ავტორიზაციის მაგალითი:

// ფაილი Web.config:

```
<configuration>
```



```
  <system.web>
      <authentication mode="Forms">
        <forms name=".ASPXUSERDEMO"
loginUrl="login.aspx" protection="All" timeout="60" />
      </authentication>
      <authorization>
        <deny users="?" />
      </authorization>
    <globalization requestEncoding="UTF-8"
responseEncoding="UTF-8" />
  </system.web>
</configuration>
```

// ფაილი Default.aspx:

```
<%@ Import Namespace="System.Web.Security " %>
<html>
 <script language="C#" runat="server">
   void Page_Load(Object Src, EventArgs E )
     { Welcome.Text = "Hello, " + User.Identity.Name;
     }
    void Signout_Click(Object sender, EventArgs E)
      { FormsAuthentication.SignOut();
        Response.Redirect("login.aspx");
      }

 </script>
 <body>
  <h3><font face="Verdana">Using Cookie
Authentication</font></h3>
  <form runat="server" ID="Form1">
   <h3><asp:label id="Welcome" runat="server" /></h3>
   <asp:button text="Signout" OnClick="Signout_Click"
runat="server" ID="Button1" NAME="Button1" />
  </form>
 </body>
</html>
```

// ფაილი Login.aspx:

```
<%@ Import Namespace="System.Web.Security " %>
```

```
<html>
 <script language="C#" runat="server">

    void Login_Click(Object sender, EventArgs E) {
 //authenticate user: this samples accepts only one user
 // with a name of someone@www.contoso.com and a password
 // of 'password'
if((UserEmail.Value=="someone")&&
                        (UserPass.Value=="password"))
  {
    FormsAuthentication.RedirectFromLoginPage
                (UserEmail.Value,PersistCookie.Checked);
  }
 else
   { Msg.Text = "Invalid Credentials: Please try again";
     }
 }
 </script>
 <body>
    <form runat="server" ID="Form1">
   <h3><font face="Verdana">Login Page</font></h3>
   <table>
    <tr>
      <td>Email:</td>
      <td><input id="UserEmail" type="text"
           runat="server" NAME="UserEmail" /></td>
      <td
      </td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><input id="UserPass" type="password"
            runat="server" NAME="UserPass" /></td>
      <td>
      </td>
    </tr>
  <tr>
  <td>Persistent Cookie:</td>
```

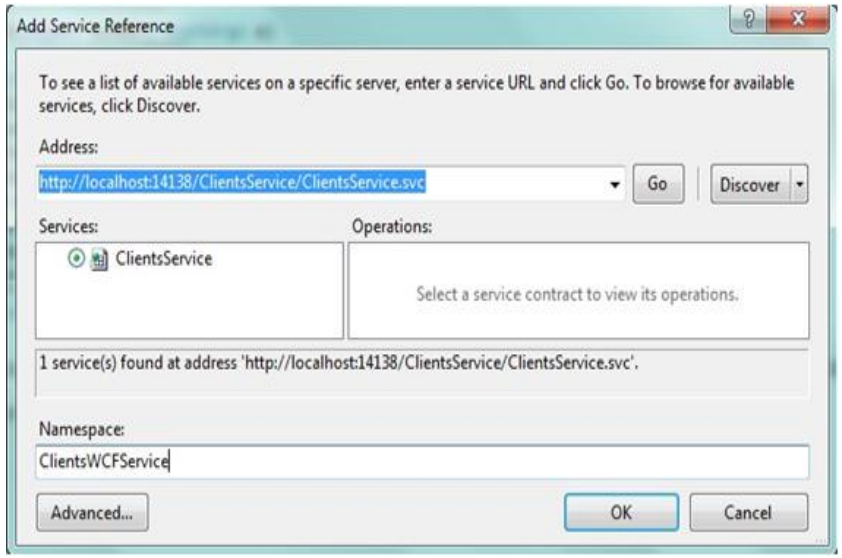


```
  <td><ASP:CheckBox id="PersistCookie" runat="server" />
     </td>
     <td></td>
    </tr>
   </table>
   <asp:button text="Login" OnClick="Login_Click"
           runat="server" ID="Button1" NAME="Button1" />
   <asp:Label id="Msg" ForeColor="red" Font-
Name="Verdana"
           Font-Size="10" runat="server" />
  </form>
 </body>
</html>
```

ნახ.5.17

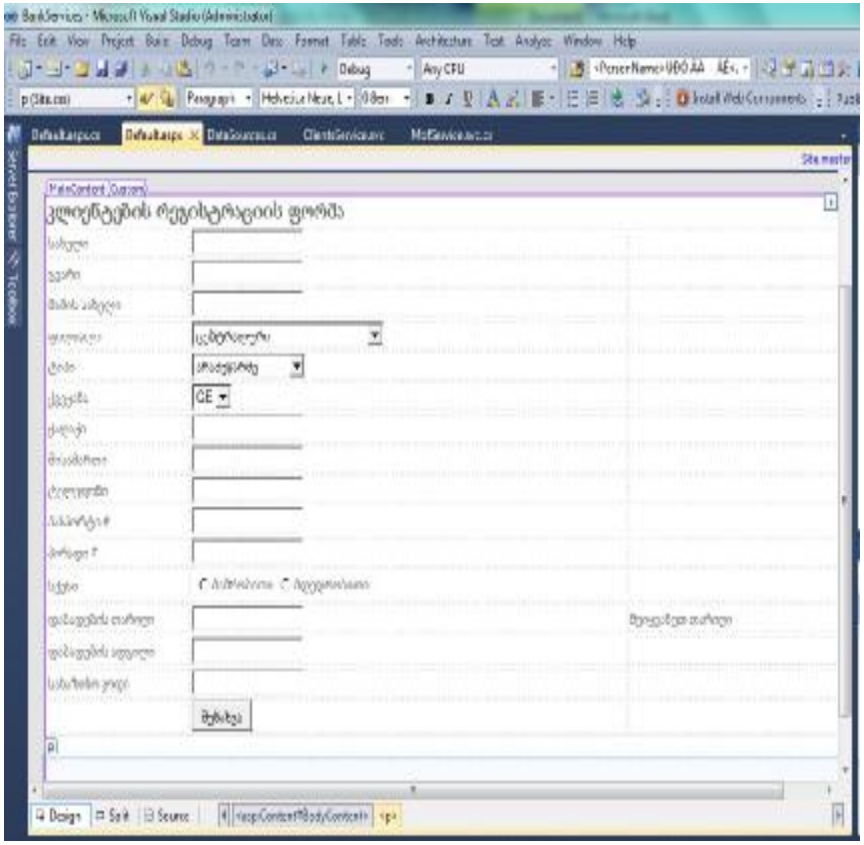
5.6. Web-სისტემის მომხმარებელთა ინტერფეისების აგება სერვისების რეალიზაციით

Web-ინტერეფისის ფორმა შექმნილია ASP.NET აპლიკაციაში. იგი შედგება Web-გვერდებისა და კოდის ფაილებისგან. ეს აპლიკაცია მონაცემების ბაზაში წვდომისათვის გამოიყენებს Web-სერვისებს. Web-სერვისები დამატებული Add Service Reference დიალოგური ფანჯრის საშალებით. ამ ფანჯრის სურათი მოცემულია 5.18 ნახაზზე.



ნახ.5.18. Add Service Reference დიალოგური ფანჯარა.

ამ ფანჯარაში გაიწერება Web-სერვისი URL. იქმნება სპეციალური სახელების სივრცე (Namespace) და ხდება Web-სერვისთან დასაკავშრებელი კლასების გენერაცია. ახალი კლიენტების რეგისტრაციის ფორმა, რომელიც გამოიყენება მათი რეგისტრაციისათვის სისტემაში, მონაცემების ბაზაში შესანახად, მოცემულია 5.19 ნახაზზე.



ნახ.5.19. მომხმარებლის ინტერფერისი ASP.NET გარემოში

Web-გვერდი შედგება HTML და კოდის ფაილებისგან. კოდში გამოყენებულია C# ენა. მომხმარებლის მონაცემების შეტანის შემდეგ „შენახვა" ღილაკზე დაჭერით მოხდება მონაცემების გადაგზავნა სერვერზე და პროცედურის გამოძახება AddClient_Click. მოცემულია ღილაკის Click მეთოდის 5.2 ლისტინგი:

// ლისტინგი_5.2 ----------

```
protected void AddClient_Click(object sender, EventArgs e)
       {
          ClientsWCFService.ClientsServiceClient service =
          new ClientsWCFService.ClientsServiceClient();
            string firstName = txtFirstName.Text;
            string lastName = txtLastName.Text;
            string fatherName = txtFatherName.Text;
int brachID = Convert.ToInt32(ddlBranch.SelectedValue);
byte type = Convert.ToByte(ddlType.SelectedValue);
            string country = ddlCountry.SelectedValue;
            string city = txtCity.Text;
            string address = txtAddress.Text;
            string phone = txtPhone.Text;
            string passport = txtPassport.Text;
            string personalID = txtPersonalID.Text;
bool gender =  rdlGender.SelectedValue=="0"?false:true;
DateTime birtDate = DateTime.Parse(txtBirtDate.Text);
            string birtPlace = txtBirtPlace.Text;
            string taxCode = txtTaxCode.Text;
     try
      {
        int clientNo = service.OpenClient(firstName,
        lastName, fatherName, brachID, type, gender,
        passport, personalID, country, city, address,
        phone, birtPlace, birtDate, taxCode);

        if (clientNo != -1)
           Response.Redirect(String.Format("~/Result.aspx?
                              clientId={0}", clientNo));
      }
      catch (Exception ex)
        { throw ex; }
     }
```

ამ პროცედურაში გამოიძახება Web-სერვისი ClientsWCFService და მისი მეთოდი OpenClient. ამ მეთოდს გადაეცემა ახალი მომხმარებლის მონაცემები. Web-სერვისი კი უზრუნველყოფს ამ მონაცემების ასახვას ბაზაში.

VI თავი: კორპორაციათა საწარმოო რესურსების მართვის და ინფორმაციის ინტეგრაციის ტექნოლოგიები

წინამდებარე თავში განხილულია კორპორაციათა მართვის სისტემის სრულყოფის ახალი ინფორმაციული ტექნოლოგიები. კერძოდ, საწარმოო რესურსების მართვის (ERP) და კლიენტებთან ურთიერთობების მართვის (CRM) სისტემები [62].

ჩამოყალიბდა საწარმოო რესურსების ეფექტურად დაგეგმვისა და მართვის ამოცანა, რომელიც მდგომარეობს შემდეგში: კორპორაციებისთვის ძალიან რთულია საქმიანობის საკუთარ სტრატეგიასა და მიზნებთან ზედმიწევნით თანხმობაში წარმართვა, წვდომა საჭირო ინფორმაციაზე რეალურ დროში ძირითადად არ არსებობს. შესაბამისად შეუძლებელია პრობლემატური საკითხების ადრეულ ეტაპზე იდენტიფიცირება. შედეგად ისისნი ხელიდან უშვებს და იკარგება მრავალი შესაძლებლობანი, რაც მხოლოდ მოგვიანებით ხდება ცნობილი. ამასთანავე, კორპორატიული მიზნების მიღწევა არის ძალზედ რთული, როდესაც ადამიანური რესურსი ორგანიზაციულ მიზნებთან არაა თანხვედრაში.

რაც შეეხება კორპორაციათა ხარჯების ზოგად დონეს, საოპერაციო თვალსაზრისით იგი არის მაღალი, რაც მათი მოუქნელობის, პროცესების დაბალი სტანდარტიზაციით, დაბალი ეფექტიანობითა და ადაპტაციის უნარით არის გამოწვეული. აგრეთვე საინფორმაციო და კოლაბორაციული ფუნქციები ვერ ვრცელდება მთლიან კორპორაციაზე.

საინფორმაციო ტექნოლოგიების გარეშე, კორპორაციებს არ გააჩნიათ ორგანიზაციის ხედვა, კერძოდ, ფინანსური და მენეჯერული აღრიცხვიანობის ფუნქციონალობა მოისუსტებს, რომ აღარაფერი ვთქვათ ბიზნეს ანალიტიკურ შესაძლებლობებზე.

შედეგად, ყველაფერ ამას კომპანია მიჰყავს დაბალ შემოსავლიანობამდე, სუსტ ფინანსურ და აგრეთვე რისკების სუსტ მართვამდე.

ბიზნესპროცესების ინტეგრაციისა და ოპტიმიზაციის გარეშე კორპორაციებს აქვს მაღალი ინტეგრაციის ხარჯები და, როგორც წესი, უხდება პროგრამული უზრუნველყოფის შეძენა მესამე, გარე მხარისგან, რაც ხშირად არაეფექტურია და იწვევს ფინანსური რესურსის გადაჭარბებულ ხარჯვას.

რაც შეეხება ადამიანური რესურსების მართვას კორპორაციაში, ძალზედ რთულია ჩამოყალიბდეს კარიერისა და განვითარების გეგმები საქმის მოწინავე შემსრულებლებთათვის, ადამიანებისთვის, რომლებიც მართლაც ფასეულ რესურსს წარმოადგენს კომპანიისთვის. ასეთი გეგმების არსებობის შემთხვევაშიც კი, ისინი არ არის ნათლად განსაზღვრული, ანუ არაეფექტურია. ამ მხრივ, მოწინავე შემსრულებლების საქმიანობა არ არის დაკავშრებული საკომპენსაციო პროგრამებთან. აგრეთვე, თანამშრომლებისთვის რთულია მოიპოვონ კომპანიაში ინფორმაცია, რომელიც მათ სჭირდებათ ყოველდღიურ საქმიანობაში.

ზემოხსენებული ამოცანების ცალკეული გადაჭრა შესაძებელია მრავალი გზით, კომპანიები დიდ დროს უთმობს ამ ამოცანებს, თუმცა მათი ერთიანად გადაწყვეტა ინტეგრირებული გზით მხოლოდ და მხოლოდ საწარმოო სისტემის ERP დანერგვითაა შესაძებელი.

6.1. კორპორაციათა საწარმოო რესურსების მართვის სისტემა (ERP)

ბოლო პერიოდის განმავლობაში მსხვილ კომპანიათა დიდი ნაწილი ტრანზაქციათა დამუშავებისას იყენებს ERP (Enterprise Resource Planning) სისტემებს. მეორეს მხრივ, დიდი მნიშვნელობა

ენიჭება მონაცემთა საცავებს, რომლებიც მნიშვნელოვან სამსახურს უწევს კომპანიის ხელმძღვანელებს გადაწყვეტილებათა მიღებაში [66]. დიდი კომპანიის მიერ წარმატებული პროექტების განსახორციელებლად საჭირო გახდა, ერთის მხრივ, ERP-სისტემების და მეორეს მხრივ, მონაცემთა საცავების პარალელურად გამოყენების აუცილებლობა [14,62].

ERP სისტემის განვითარება დაიწყო 1990 წლიდან, რის აუცილებლობაც განაპირობა სამმა ძირითადმა ტენდენციამ:

- მომდევნო წლისათვის არსებული ფინანსური პრობლემები;
- ბიზნესპროცესების, როგორც ძირითადი სტრატეგიის რეინჟინირება;
- ERP - სისტემის ფუნქციონალური და ტექნიკური მიღწევები.

კლიენტ-სერვერ არქიტექტურის ბაზაზე წარმოქმნილი ინტეგრირებული ERP სისტემა ტრანზაქციათა დამუშავებისას პროგრამული უზრუნველყოფის ეფექტური საშუალებაა. სისტემის ნაკლად შეიძლება ჩავთვალოთ ის, რომ არ არის გათვალისწინებული მოთხოვნათა ხარისხის განსაზღვრა. ამ დროს იგი ძალიან მნიშვნელოვანია კომპანიის ხელმძღვანელისათვის, რათა ხელი შეუწყოს გადაწყვეტილებათა მიღებას.

იმისათვის, რომ ERP სისტემა იყოს სრულყოფილი სახით წარმოდგენილი, პროგრამული რესურსების მწარმოებლები აერთიანებენ ERP სისტემას მონაცემთა საცავებთან [14].

ინტეგრირებული ანალიზი მონაცემებს აერთიანებს რამდენიმე სისტემაში: დამკვეთების მართვა ურთიერთქმედების რეჟიმში CRM - Customer Relationship Management, ადამიანური რესურსების მართვა, ფინანსები, ელექტრონული კომერცია. ეს ტექნოლოგია აგრეთვე გვაძლევს შესაძლებლობას გავაფართოვოთ ინფორმაციის ანალიზური დამუშავება, რის შედეგადაც

კორპორაციას შეეძლება სწრაფად მოახდინოს რეაგირება ბაზარზე არსებულ მუდმივად ცვალებად სიტუაციებზე.

ინტეგრირებული ანალიზის ჩატარებისას შესაძლებელია მივიღოთ შემდეგი შედეგები:

- აერთიანებს ინფორმაციას მთელი კორპორაციის მასშტაბით;
- საშუალებას გვაძლევს ინდივიდუალურად შევარჩიოთ Business Inteligence მეთოდი სისტემის ყოველი მომხმარებლისათვის;
- მოიცავს ყველაზე თანამედროვე ანალიზურ მოდელს, რომლის დანიშნულებაცაა გადაწყვეტილების მიღებაში ხელის შეწყობა.

ინტეგრირებული ანალიზი წარმოებას უზრუნველყოფს აუცილებელი ანგარიშგების სახეობით და ინსტრუმენტული საშუალებებით, რომლებიც კორპორაციის ხელმძღვანელს ხელს უწყობს მნიშვნელივან გადაწყვეტილებატა მიღებაში. მაგალითად ფინანსური ინფორმაცია წარმოდგენილია ბევრ ERP პროდუქტში ტრანზაქციული ფორმით, ხოლო მომხმარებელს სჭირდება სწრაფად მიიღოს ინფორმაცია სხვადასხვა დეტალიზაციის დონეზე. ამის გარდა ტრადიციული ERP სისტემა იყენებს რელაციურ ბაზებს, რომელთა სტრუქტურაც საშუალებას გვაძლევს შემომავალი მონაცემები წარმოვადგინოთ მხოლოდ ერთი განზომილებით, ხოლო მათი მრავალგანზომილებიანი წარმოდგენა მოითხოვს მაღალტექნოლოგიურ ტექნიკურ კვალიფიკაციას, რომელთა მწარმოებლურობა ოპერატიულ რეჟიმში ხშირ შემთხვევაში დაბალია.

მონაცემთა მრავალგანზომილებიანი წარმოდგენა კარგადაა შესაძლებელი მონაცემთა საცავებში [14]. ასეთი სისტემები შედგება როგორც წესი სამი კომპონენტისაგან:

- მონაცემთა წარმოდგენა საცავში საჭირო ფორმატით;

- რეპოზიტორები, როგორც ინფორმაციის ოპერატიული წყარო;
- ანალიზური დონე, რომლის საშუალებითაც მომხმარებელს შეეძლება განსაზღვროს მონაცემთა ტიპი.

ადრე ბევრი კომპანია ქმნიდა საკუთარ ინტერფეისებს, რომელთა დახმარებითაც უკავშირდებოდა ERP სისტემას, როგორც დამოუკიდებელ ანალიზურ დანართს. ამ დროს იქმნებოდა პრობლემები იმის გამო, რომ სხვა მწარმოებლებს მუდმივად უნდა მოეხდინა საბაზო არქიტექტურის შევსება. ამ პრობლემის მოსაგვარებლად მნიშვნელოვან როლს თამაშობს მონაცემთა საცავები, როგორც ERP სისტემის საინფორმაციო არქიტექტურის ცენტრალური კომპონენტი.

6.2. კლიენტებთან ურთიერთობის მართვის სისტემა (CRM)

კორპორაციის ფინანსური მდგომარეობის, მომსახურე პერსონალის, დამკვეთებზე და შემსრულებლებზე მონაცემთა ერთიანად წარმოდგენა ერთ-ერთი რთული ამოცანაა. მის გადასაწყვეტად შეიძლება სხვადასხვა სახის სისტემის გამოყენება.

CRM (Customer Relationship Management) – სისტემა მოიპოვებს ინფორმაციას კლიენტებთან მომსახურე თანამშრომლებზე და მათ ურთიერთკავშირზე კონკრეტულ დამკვეთთან. ადამიანური რესურსების მართვის სისტემაში იგივე ინფორმაცია წარმოდგენილია სხვა ფორმატით, ფინანსურ სისტემაშიც განსხვავებული ფორმატით და ა.შ. საცავების გამოყენებისას მონაცემთა ასეთი სახით წარმოდგენა არ ხდება, რადგან მოხდება მათი სხვადასხვა დონეზე გაერთიანება, მაგალითად:

- სრული და ურთიერთშეთანხმებული ინფორმაციის წარმოდგენა ფინანსებზე, დამკვეთებზე და თანამშრომლებზე;
- წარმოების და დაგეგმვის მაჩვენებლების წარმოდგენა;
- ინსტრუმენტთა ინფრასტრუქტურათა წარმოდგენის პროგრამული უზრუნველყოფა;
- პრობლემათა გადაჭრის ეფექტურობის ამაღლება, გამოყენებულ სისტემათა და ინტეგრირებულ ინსტრუმენტთა შემცირების ხარჯზე.

6.1 ნახაზზე ნაჩვენებია სხვადასხვა სახის მიდგომა ფირმა-მწარმოებელზე:

ნახ.6.1

6.3. ERP და CRM - შედარებითი ანალიზი

ERP სისტემის ახალი ვერსიის გამოჩენამ შესაძლებელი გახადა ანალიზის პროცესის ჩატარების დროს ცვლილებების შეტანა საინფორმაციო პროდუქტის კონკრეტულ დანართში.

როგორც ნახაზიდან ჩანს ანალიზის პროცესის ჩატარება ეტაპობრივად განიცდის განვითარებას სრულყოფისაკენ, რომელიც თაობებად არის წარმოდგენილი და საბოლოოდ იღებს ინტეგრირებულ სახეს.

აღსანიშნავია ის გარემოება, რომ ბოლო პერიოდში მსხვილმა კომპანია მწარმოებლებმა, როგორებიცაა SAP და Peoplesoft, მეორე თაობის ანალიზური სისტემის გამოყენებით მიაღწიეს უდიდეს წარმატებებს ეკონომიკისა და ბიზნესის სფეროში.

ფირმა SAP ბაზარზე გამოჩნდა მონაცემთა საცავებით და უშვებდა პროდუქტს დასახელებით Business Information Warehouse (BW). მის კომპონენტებს შორის იყო ასობით მონაცემთა ბაზა, მონაცემთა პროგრამული მართვის სქემები, ინტერაქტიული კვლევები, ეფექტურობის მაჩვენებლები და ა. შ.

ინტეგრირებული ანალიზური სისტემის გამოყენებით მან შესაძლებელი გახადა თითოეული შემადგენელი კომპონენტის ექსპლუატაციის ღირებულების შემცირება, რაც დამკვეთთათვის მეტად მნიშვნელოვანი აღმოჩნდა.

Peoplesoft კომპანიამ რამდენიმე წლის წინ დაიწყო ინტეგრირებული ანალიზური სისტემის გამოყენება. მეშვიდე ვერსიაში Peoplesoft–ის დანართის პაკეტში დამატებული იყო პროდუქტთა სია, რომელიც უზრუნველყოფდა წარმოების ეფექტურ მართვას (EPM – Enterprise Performance Management). თავიდან EPM მოდელი ინტეგრირებული იყო Peoplesoft არქიტექტურასთან, სადაც გამოყენებულ იყო არქიტექტურული სტანდარტები და მონაცემთა დამუშავების ორიგინალური ინსტრუმენტები.

ფირმა მუდმივად ავითარებს EPM მოდულის შესაძლებლობებს. იგი დღეისათვის შედგება რამდენიმე კომპონენტისაგან: მონაცემთა გარდაქმნა და ჩატვირთვა (ETL),

მონაცემთა საცავები, მონაცემთა დამუშავების ანალიზური ინსტრუმენტი, ანალიზური დანართი, რომელიც გამოიყენება დაგეგმვის, ბიუჯეტის და ბალანსის მაჩვენებლებისათვის.

ყველა ჩამოთვლილი ანალიზური კომპონენტის გამოყენება საშუალებას გვაძლევს შევაფასოთ კომპანიის მდგომარეობა სხვადასხვა დონეებზე. მაგალითად, რომელიმე პროდუქტის გასაღების დონე ცალკეულ რეგიონებში; გაყიდული პროდუქტის ღირებულება და კონკრეტული სახეობა; საქონლის რეალიზაციისათვის საკომისიო გადასახადები; შემოსავლები.

საბოლოოდ შეიძლება ვთქვათ, რომ ERP მოდული არის ერთ-ერთი საუკეთესო საშუალება ინტეგრირებულ მონაცემთა ანალიზისათვის. ცხადია, რომ დანართი, რომელიც მუშავდებოდა კონკრეტული მომხმარებლისათვის, წარსულს ჩაბარდა და მისი ადგილი დაიკავა მზა პროდუქტმა, რომელიც განკუთვნილია წარმოების რესურსების მართვისათვის.

მსხვილ საწარმოთა ანალიზური ERP დანართის დამკვეთები ცალსახად განსაზღვრავენ, თუ რა მიმართულებით სჭირდებათ განვითარება. ინტეგრირებული ანალიზური სისტემის რეალიზაციით ფირმებს აქვს შესაძლებლობა ოპტიმალურად განსაზღვრონ და მართონ საწარმოო რესურსები.

6.4. პროცესების ავტომატიზაცია და კლიენტთა მონაცემების ანალიზი

განიხილება ორგანიზაციული საკითხები, რომლებიც უკავშირდება საინფორმაციო დანართის სპეციალურ კლასს და ორიენტირებულია არა ტრანზაქციათა ოპერატიულ დამუშავებაზე (On-Line Transaction Processing – OLTP), არამედ მათ ოპერატიულ ანალიტიკურ დამუშავებაზე (On-Line Analitical Processing OLAP) [63]. ამ ორი განსხვავებული სახის სისტემის საშუალებით

შესაძლებელი ხდება აბსოლიტურად განსხვავებული ამოცანების გადაჭრა. კორპორაციული საინფორმაციო – OLTP სისტემები იქმნება იმისათვის, რათა ხელი შეუწყოს კორპორაციათა ყოველდღიურ საქმიანობას და წინა პლანზე წარმოაჩინოს აქტუალური მონაცემები. OLAP-სისტემები კი ემსახურება კორპორაციის ან მისი ცალკეული კომპონენტების საქმიანობის ანალიზს, რის საშუალებითაც აკეთებს პროგნოზს კორპორაციის მომავალ მდგომარეობაზე. ამისათვის კი საჭიროა კორპორაციის მუშაობის შესახებ (როგორც წარსულში ასევე არსებულ მდგომარეობაში) მრავალრიცხოვან მონაცემთა დაგროვება.

მონაცემთა ოპერატიულ ანალიზური დამუშავების სისტემა განსხვავდება სტატისტიკური სისტემისაგან, რომელიც ხელს უწყობს გადაწყვეტილების მიღებას (Decision Support System – DSS) იმდენად, რამდენადაც OLAP- სისტემა საშუალებას აძლევს ანალიტიკოსებს მოთხოვნათა კლასი მოაწესრიგოს დინამიკურად, რაც მნიშვნელოვნად აიოლებს დასმული ანლიტიკური ამოცანის გადაწყვეტას.

DSS-უზრუნველყოფს ანგარიშების წარმოდგენას, რომელიც შეესაბამება წინასწარ ფორმულირებულ წესებს და OLAP სისტემის მუშაობას უწყობს ხელს. სისტემაში შემოსული ახალი მოთხოვნის დასაკმაყოფილებლად საჭიროა მისი ფორმალური აღწერა, დაპროგრამება და შემდგომ შესრულებაზე გაშვება.

OLAP სისტემის თემატიკა ფართო სპექტრისაა. მიუხედავად ამისა გვხვდება ისეთი პრობლემები, რომლებიც თან ახლავს მსხვილ კორპორაციულ სისტემებს, იგი მოიცავს რამდენიმე მონაცემთა ბაზას, სადაც თავმოყრილია მოქმედებათა სახვადასხვა სფეროები. ერთი და იგივე მონაცემები შესაძლოა წარმოდგენილი იყოს სახადასხვა სახით, რაც ხშირ შემთხვევაში მათ შეუთავსებლობას იწვევს.

როგორც ცნობილია, OLAP-სისტემის საშუალებით ხდება კორპორაციის, ისტორიულ მონაცემთა ურთიერთშეთანხმებული დამუშავება, ამას გარდა ოპერატიულ ანალიტიკურ დამუშავებისათვის საჭირო ხდება კორპორაციის გარე მონაცემთა წყაროს გამოყენება, რომელიც იკავებს მოთხოვნათა შესაბამის სხვადასხვა ფორმატს. ასეთი სახის მსჯელობის საფუძველზე საჭირო ხდება მონაცემთა საცავების კონცეფციის, როგორც საგნობრივ-ორიენტირებული, ინტეგრირებული, ქრონოლოგიურ მონაცემთა ერთობლივი წარმოდგენის საშუალების გამოყენება, რათა ვაწარმოოთ ორგანიზებული მართვა. მონაცემთა საცავის (Datawarehouse - DWH) კონცეფციის ძირითადი საფუძველი ეყრდნობა ორ მთავარ იდეას:

1. დაცალკევებულ, დეტალიზებულ (იმ თვალსაზრისით, რომ ისინი აღწერს კონკრეტულ ფაქტებს, თვისებებს, მოვლენებს და ა.შ.) მონაცემთა ინტეგრაცია ერთ საერთო საცავში. ინტეგრაციის პროცესში აუცილებელია მოხდეს დეტალურ ინფორმაციათა შეთანხმება. მონაცემები შეიძლება გამოყენებულ იქნას კორპორაციის ისტორიული არქივიდან, ოპერატიულ მონაცემთა ბაზიდან და გარე წყაროებიდან;

2. მონაცემთა ნაკადი, რომელიც გამოიყენება ოპერატიული დამუშავებისათვის და მონაცემთა ნაკადი, რომელიც გამოიყენება გადასაწყვეტი ამოცანის ანალიზისათვის.

მონაცემთა საცავის კონცეფცია ისეა განსაზღვრული, რომ ოპერატიულ ანალიზური დამუშავება შესაძლებელია ვაწარმოოთ ქსელის ნებისმიერ კვანძში, იმისდა დამოუკიდებლად, თუ სად იმყოფება ძირითადი საცავი.

მონაცემთა საცავში ასახულია კორპორაციის განვითარების შესახებ თავმოყრილი ურთიერთშეთანხმებული ინფორმაცია. აქ ნათლადაა ასახული კორპორაციის წარმატების და

წარუმატებლობის, მათი დამკვეთების და პარტნიორების შესახებ ინფორმაცია, აგრეთვე საბაზრო მდგომარება, რაც გვაძლევს საშუალებას დავინახოთ კორპორაციის წარსული, მიმდინარე საქმიანობა და განვსაზღვროთ მისი მომავალი.

ერთ-ერთი მნიშვნელოვანი საკითხია მომხმარებელთა აუტენტიფიკაცია, მონაცემთა დაცვა, როდესაც ოპერატიულ მონაცემთა ბაზიდან და გარე წყაროებიდან მათი განთავსება ხდება მონაცემთა საცავში, და მონაცემთა დაცვა ქსელში გადაცემისას. ყველა ეს საკითხი მონაცემთა საცავში გათვალისწინებულია და უსაფრთხო.

მეტამონაცემთა როლი OLAP სისტემაში მნიშვნელოვან ადგილს იკავებს. მაგალითად, თუ კორპორაციის მენეჯერი სისტემას უყენებს მოთხოვნას, მან ჯერ უნდა გაანალიზოს, როგორი სახის ინფორმაცია აინტერესებს, იგი რამდენად აქტუალურია, შეიძლება თუ არა ამ ინფორმაციაზე დაყრდნობა და რა დრო დასჭირდება პასუხის ფორმირებას.

OLAP სისტემის მომხმარებელისათვის საჭიროა შემდეგი სახის მეტაინფორმაცია:

- მონაცემთა სტრუქტურის და მათი ურთიერთდამოკიდებულების აღწერა;
- ინფორმაცია საცავში არსებული მონაცემების შესახებ;
- ინფორმაცია მონაცემთა წყაროების შესახებ და მათ საიმედოობაზე;
- ინფორმაცია მონაცემთა განახლების პერიოდის შესახებ;
- ინფორმაცია მონაცემთა მფლობელების შესახებ;
- სტატისტიკური შეფასება, თუ რა დროშია შესაძლებელი მოთხოვნათა დაკმაყოფილება.

6.5. ERP სისტემის რეალიზაციის სტადიები

განიხილება სისტემის ეფექტური შერჩევისა და დანერგვის საკითხები. შემოთავაზებულია სამი ეტაპი, რომელთა გავლაც აუცილებელია სისტემის წარმატებით დანერგვის განსახორციელებლად. ეს ეტაპებია:

1) ორგანიზაციის მზადყოფნის შეფასება ERP სისტემის დასანერგად;

2) სასურველი, ოპტიმალური ERP სისტემის შერჩევა სხვადასხვა ანალიტიკური საშალებების გამოყენებით;

3) კომპანიის შესწავლის, ანუ დიაგნოსტიკის ეტაპი.

4) სისტემის რეალიზაციის საშუალებების ჩამოყალიბება.

ამგვარად, ERP სისტემის ოთხ–ეტაპიანი დანერგვის მეთოდოლოგია, რომელიც განსხვავდება არსებული დანერგვის მეთოდოლოგიებისგან ძრითადად იმით, რომ აქცენტი კეთდება დანერგვის წინა ეტაპებზე, რათა მოხდეს სისტემის წარმატებული დანერგვა (ნახ.6.2).

ნახ.6.2. ERP რეალიზაციის სტრატეგიული მოდელი

6.6. კორპორაციათა ოპერატიული ანალიზის სისტემის დონეები

მსხვილ კორპორაციათა ანალიზისათვის გამოიყენება სამდონიანი OLAP სისტემა, რომელიც შემდეგ ეტაპებს მოიცავს:

პირველ დონეზე რეალიზებულია კორპორაციული მონაცემთა საცავი, რომლის საფუძველსაც წარმოადგენს თანამედროვე რელაციური მონაცემთა ბაზის მართვის სისტემა [14]. რელაციური მბმს უზრუნველყოფს ეფექტურ დაცვას და დიდი მოცულობის მონაცემთა მართვას, მაგრამ არც თუ კარგად შეესაბამება OLAP სისტემის მოთხოვნებს, კერძოდ პრობლემა მდგომარეობს მონაცემთა მრავალგანზომილებიან წარმოდგენაში;

მეორე დონეზე რეალიზებულია მონაცემთა ბაზა, რომლის საფუძველსაც წარმოადგენს მრავალგანზომილებიან მონაცემთა ბაზების მართვის სისტემა. ასეთ სისტემად მოიაზრება Oracle Express Server. მონაცემთა ბაზის ფორმირება სრული სახით არ არის აუცილებელი, საკმარისია იგი შეიცავდეს მიმმართველს მონაცემთა საცავისკენ და შესაძლებელს ხდიდეს ინფორმაციის მიღებას;

მესამე დონეზე განთავსებულია განსაზღვრული რაოდენობის კლიენტთა სამუშაო ადგილი, რომელზეც მოთავსებულია მონაცემთა ოპერატიული ანალიზის ჩასატარებელი საშუალება;

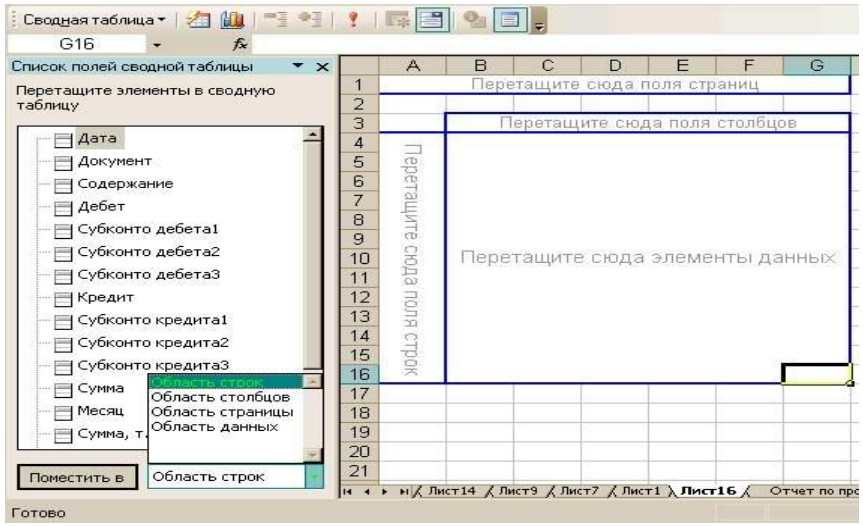
როგორც ზემოთ აღვნიშნეთ, OLAP საშუალებას გვაძლევს დიდი კომერციული ობიექტების მონაცემთა ბაზებს გავუკეთოთ ორგანიზება.

OLAP ინსტრუმენტით მონაცემები ისეა ორგანიზებული, რომ აადვილებს მონაცემთა მართვის ანალიზს და ამცირებს საჭირო მონაცემთა მოძებნის დროს. ეს ინსტრუმენტი საშუალებას

გვაძლევს რამდენიმე წამის განმავლობაში ავაგოთ დიდი მოცულობის მონაცემთა მასივების შესაბამისი რთული ანგარიშები.

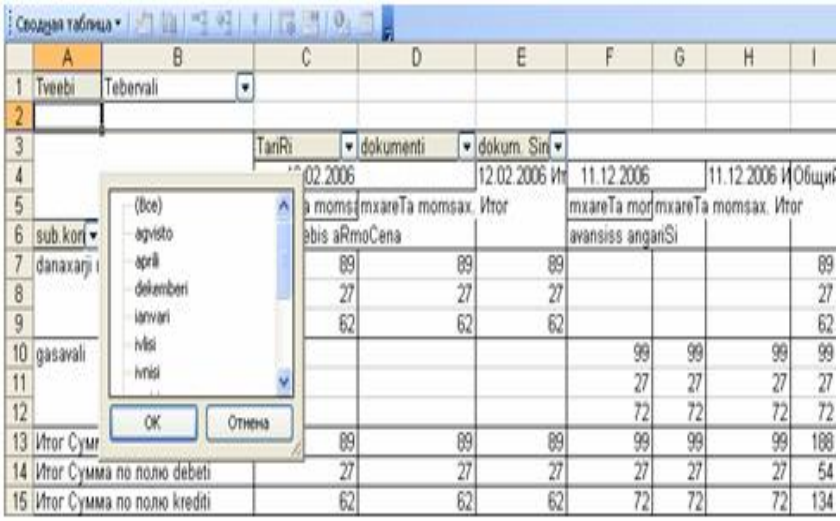
კომპიუტერული ტექნოლოგიების განვითარება დღეს ნებისმიერ მომხმარებელს საშუალებას აძლევს პერსონალური კომპიუტერის გამოყენებით მიიღოს ისეთივე შედეგი, როგორსაც OLAP-კუბი გვაძლევს.

მაგალითად, Microsoft Excel - ის დინამიკური ცხრილები არის OLAP-კუბის სახე, რომლის საწყისი ცხრილი წარმოდგენილია 6.3 ნახაზზე [64].



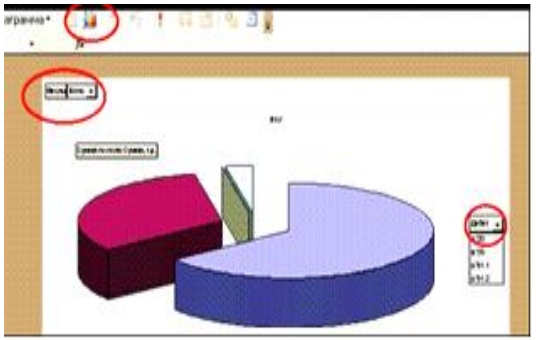
ნახ.6.3

მარცხენა მხარეს მოთავსებულია მონაცემთა ბაზაში არსებული ველები, რომელთა მიხედვით შესაძლებელია ვაწარმოოთ ანალიზი და გავფილტროთ მოთხოვნათა შესაბამისად. მაგალითად, თვეების მიხედვით შეგვიძლია ვნახოთ შემოსავალ-გასავლის ანალიზი (ნახ.6.4).



ნახ.6.4

ასევე შეგვიძლია წარმოვადგინოთ არსებულ მონაცემთა დამოკიდებულება დიაგრამის საშუალებით (ნახ.6.5). დიაგრამა ინტერაქტიულია – ჩვენ შესაძლებლობა გვაქვს შევარჩიოთ, თუ რომელი მონაცემი დავფაროთ და რომელი გამოვაჩინოთ.



ნახ.6.5

მონაცემთა მრავალგანზომილებიანი მოდელის გამოყენება, რომელსაც განსაკუთრებული ადგილი უკავია მონაცემთა სრულყოფილი ანალიზისათვის, არა მარტო გვაძლევს სტატისტიკურ ინფორმაციას საწარმოთა საქმიანობის შესახებ, არამედ პოულობს კანონზომიერებას სხვადასხვა პროცესებს შორის.

OLAP ინსტრუმენტი ანალიტიკოსებს და საწარმოთა ხელმძღვანელებს უზრუნველყოფს აუცილებელი ინფორმაციით, რათა მართვის სფეროში მიიღონ ეფექტური გადაწყვეტილება [65].

VII თავი: ჰიბრიდული აპლიკაციების აგების WPF-ტექნოლოგია

7.1. Windows Presentation Foundation ტექნოლოგია

აპლიკაციების (დანართების) ორი ნაირსახეობაა ცნობილი: ვინდოუს სისტემები, რომელთაც ასევე სამაგიდო აპლიკაციებს უწოდებენ და ვებ-აპლიკაციები, რომელთა გამოყენებაც ინტერნეტ ბრაუზერებიდანაა შესაძლებელი [67, 68].

ეს დანართები იქმნება .NET Framework -ის ორი სხვადასხვა პაკეტით. პირველი - Windows Forms კომპონენტებით და მეორე ASP.NET -ის საშუალებით. ორივეს აქვს თავისი უპირატესობები და ნაკლოვანებანი. კერძოდ, სამაგიდო დანართები ძალზე მოქნილი და რეაქციულია, ხოლო Web-დანართები ინტერნეტის საშუალებით იძლევა დისტანციური წვდომის საშუალებას ერთდროულად მრავალი მომხმარებლისთვის.

მაგრამ თანამედროვე კომპიუტერული ტექნოლოგიების სამყაროში ამ ორი სახის აპლიკაციებს შორის საზღვრები სულ უფრო და უფრო იშლება [69].

Web-სამსახურების და WCF (Windows Communication Foundation) სერვის-ორიენტირებული არქიტექტურის აგების საშუალებების გაჩენამ განაპირობა სამაგიდო- და ვებ-დანართების ფუნქციონირების შესაძლებლობა ერთიან განაწილებულ გარემოში,

სადაც მონაცემთა გაცვლა ხორციელდება როგორც ლოკალურ, ასევე გლობალურ ქსელებში.

7.1 ნახაზზე ნაჩვენებია ეს გამაერთიანებელი პროცესი.

ნახ.7.1

Windows Presentation Foundation (WPF) არის ერთ-ერთი ასეთი გამაერთიანებელი ტექნოლოგია, რომელიც საშუალებას იძლევა ისეთი დანართის ასაგებად, სადაც ლიკვიდირებული იქნება განხეთქილება სამაგიდო აპლიკაციასა და ინტერნეტს შორის.

WPF-დანართს, როგორც ეს ნაჩვენები იქნება ქვემოთ, შეუძლია ფუნქციონირება როგორც სამაგიდო აპლიკაციას, ასევე როგორც ვებ-აპლიკაციას ბრაუზერის შიგნით.

არსებობს ასევე WPF-ის შეზღუდული ვერსია, სახელით Silverlight, რომლითაც შესაძლებელია ვებ-დანართში დინამიკური მდგენელის დამატება.

აქ განიხილება შემდეგი საკითხები:

- რას წარმოადგენს WPF;
- WPF-ის საბაზო სტრუქტურა;
- WPF-ის ძირითადი ცნენები;
- დაპროგრამება WPF-ის გამოყენებით.

7.1.1. WPF ტექნოლოგიის შესაძლებლობები

WPF (ადრე ცნობილი იყო სახელით Avalon) არის ტექნოლოგია, რომელიც საშუალებას იძლევა დაიწეროს პლატფორმაზე დამოუკიდებელი აპლიკაციები, დიზაინისა და ფუნქციონალური შესაძლებლობების ცხადი დაყოფით. იგი ეფუძნება ადრე არსებულ ისეთ ტექნოლოგიათა გაფართოებულ ცნებებს და კლასებს, როგორიცაა Windows Forms, ASP.NET, XML, GDI+ და ა.შ. ასეთი მსგავსება კარგად ჩანს ვებ-დანართის აწყობისას .NET Framework გარემოში [1,2]. გარდა ამისა WPF-ში მრავლადაა ახალი ფუნქციონალური საშუალებები პროგრამისტებისა და მომხმარებლებისთვის, რაც მისი, როგორც ახალი ტექნოლოგიის განხილვის უფლებას იძლევა.

7.1.2. WPF -ის შესაძლებლობები დიზაინერებისთვის

მომხმარებელთა ინტერფეისების დასაპროექტებლად WPF-ში გამოიყენება დანართების ფორმატირების გაფართოებული ენა XAML (Extensible Application Markup Language) [3]. იგი მსგავსია ASP.NET-ის ფორმატირების ენის, რომელიც იყენებს XML-სინტაკსს და შეუძლია მომხმარებლის ინტერფეისს დაუმატოს მართვის ელემენტები დეკლარაციული, იერარქიული სახით. ამასთანავე მას შეუძლია მერთვის ელემენტების შექმნა, რომლებიც შეიცავს სხვა მართვის ელემენტებს, რაც მნიშვნელოვანია როგორც ინტერფეისის კონსტრუირებისთვის, ისე დანართის ფუნქციონალური შესაძლებლობებისთვის.

XAML-ენა ბევრად მძლავრია, ვიდრე ASP.NET და არაა შეზღუდული HTML ენის შესაძლებლობებით მონაცემთა ვიზუალიზაციის დროს ინტერფეისებში [4]. ის დამუშავდა სპეციალურად დღეისათვის არსებული მძლავრი გრაფიკული კარტების გათვალისწინებით DirectX-ის საფუძველზე [5]. მაგალითად:

- ვექტორული გრაფიკა და კოორდინატები მცოცავი მძიმით - უზრუნველყოფს ობიექტების მასშტაბირებას, ბრუნვას და ტრანსფორმირებას ხარისხის დაუკარგავად;
- ორ- და სამგანზომილებიანი შესაძლებლობები ვიზუალიზაციის სრულყოფისთვის;
- შრიფტების დამუშავების და ვიზუალიზაციის სრულყოფილი საშუალებები;
- გრადიენტული, უწყვეტი და ტექსტური შევსება გამჭვირვალობის არააუცილებელი ეფექტებით მომხმარებლის ინტერფეისის ობიექტებისთვის;
- ანიმაციის კადრების დაშლის ტექნოლოგია, რომელიც გამოიყენება ყველა სიტუაციაში, მათ შორის მომხმარებლის მიერ გენერირებულ მოვლენებში, მაგალითად, ღილაკის დაჭერის იმიტაცია;
- მრავალჯერადი მოხმარების რესურსები, რომელთა გამოყენება შეიძლება მართვის ელემენტების დინამიკური სტილიზაციისთვის.

დიზაინერებისთვის განსაკუთრებული ინტერესი აქვს მაიკროსიფთის ინსტრუმენტს Expression Blend (EB), რომლითაც შესაძლებელია XAML-ფაილების შექმნა და შემდგომ მათი გადატანა დანართებში. EB-ში შესაძლებელია პროექტების შექმნა და რედაქტირება ისევე, როგორც VisualStudio-ს Solution Explorer-ში. ამგვარად, Expression Blend სასურველი, მაგრამ არააუცილებელი კომპონენტია WPF-დანართების ასაგებად. ის VS სტანდარტულ

ვერსიას არ მოჰყვება, მაგრამ შეიძლება მისი დემო-ვერსიის ჩამოტვირთვა microsoft.com/expression/products/overview.aspx?key=blend -დან.

7.1.3. WPF -ის შესაძლებლობები C# - დეველოპერებისთვის

აპლიკაციების დეველოპერები პროექტებს ქმნიან VC (Visual Studio) და VCE (Visual C# Express - არის VC-ს შეზღუდული უფასო ვერსია) გარემოში სამუშაოდ.

WPF-ში გამოიყენება გამოყოფილი კოდის მოდელი, როგორც ASP.NET-ში. მაგალითად, Button მართვის ელემნტისთვის მოვლენის დამმუშავებლის (პროგრამის) მიმაგრება შეიძლება მისი XML ელემენტისთვის Click ატრიბუტის დამატებით. ეს ატრიბუტი მიუთითებს მოვლენის დამმუშავებლის სახელზე შესაბამისი XAML-გვერდის გამოყოფილი კოდის ფაილში, რომელიც შეიძლება დაწერილი იყოს C#-ზე.

WPF დანართებში მართვის ელემენტების მანიპულირება შეიძლება Windows Forms მსგავსად, რომლის დროსაც გამოიყენება დაპროგრამების ხერხები მომხმარებლის ინტერფეისების ასაგებად. გამოყოფილი კოდის საშუალებით შეიძლება შეიქმნას მართვის ელემენტის ეგზემპლარი, დაყენდეს თვისებები, მიებას მოვლენათა დამმუშავებლები და დაემატოს ეს მართვის ელემენტი ფორმაზე. ამ პროცესს შეუძლია მთლიანად გამორიცხოს XAML-კოდი.

ასეთ შემთხვევაში გამოყოფილი კოდის ზომა იქნება გაცილებით დიდი, ვიდრე შესაბამისი დეკლარაციული XAML-ის კოდი და ამასთანავე იკარგება დიზაინსა და ფუნქციონალურ შესაძლებლობებს შორის ცხადი საზღვარი, რაც არასახარბიელოა. ამიტომ სასურველია მომხმარებლის ინტერფეისზე მართვის ელემენტების განლაგება განხორციელდეს XAML-ით.

7.2. WPF-ის არქიტექტურა და კლასები

Windows Presentation Foundation (WPF) ეფუძნება ვიზუალიზაციის ვექტორულ სისტემას და ორიენტირებულია კლიენტების ვებ-აპლიკაციების (დანართების) დამუშავებაზე Microsoft.NET პლატფორმისთვის. ამ თავში განიხილება WPF-ის არქიტექტურა, ძირითად კლასთა იერარქია, მომხმარებლის ინტერფეისის აგების საკითხები, კონტროლის ელემენტების გამოყენების თავისებურებანი, შედგენილობის მართვის ძირითადი ელემენტები, მათი თვისებები და დეკლარაციული აღწერა. წარმოდგენილ ინფორმაციას - რესურსების, სტილის და შაბლონების შესახებ აქვს შესავალი ხასიათი WPF-ის საკმაოდ ეფექტურ და მრავალფეროვან კონსტრუქციებისთვის.

Windows Presentation Foundation (WPF) - ესაა კლიენტების Windows-დანართების აგების სისტემა Microsoft.NET ტექნოლოგიისთვის ვიზუალური შესაძლებლობებით. WPF-ით შეიძლება აიგოს ფართო სპექტრი როგორც ავტონომიური დანართებისა, ასევე ვებ-ბრაუზერში განთავსებული აპლიკაციებისა.

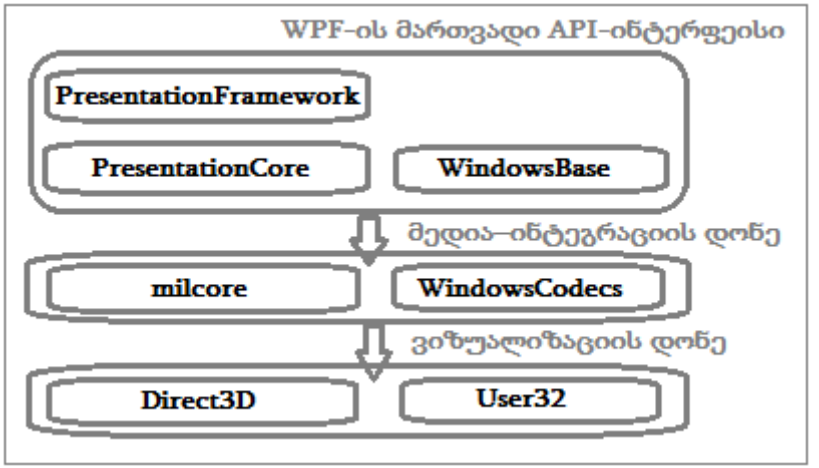
WPF-ის საფუძველია ვიზუალიზაციის ვექტორული სისტემა, რომელიც გათვლილია თანამედროვე გრაფიკულ საშუალებებზე. ვიზუალური ინტერფეისის ასაგებად გამოიყენება XAML ენა, მართვის ელემენტები, მონაცემებთან მიბმა, მაკეტები, 2- და 3-განზომილებიანი გრაფიკა, ანიმაცია, სტილები და შაბლონები, დოკუმენტები და ტექსტები, მულტიმედია და გაფორმებები.

WPF-ის გრაფიკულ ტექნოლოგიას საფუძვლად უდევს DirectX. განსხვავებით Windows Forms-სგან, სადაც გამოიყენება GDI/GDI+. WPF-ის მწარმოებლურობა უფრო მაღალია, ვიდრე GDI+-ის, გრაფიკის აპარატურული დამაჩქარებლის გამოყენების გამო DirectX -ში.

WPF უზრუნველყოფს მომხმარებლის მაღალი დონის ინტერფეისს და იძლევა შემდეგ შესაძლებლობებს:

- ვებ-ისმაგვარი მოდელის აწყობა, რომელიც უზრუნველყოფს მართვის ელემენტების განლაგებას და მოწესრიგებას შინაარსის მიხედვით;
- ხატვის მრავალფუნქციურ მოდელს გრაფიკული კომპონენტების საფუძველზე (საბაზო ფორმები, ტექსტური ბლოკები, გრაფიკული ინგრედიენტები);
- მოდელი ფორმატირებული ტექსტით, რომელიც უზრუნველყოფს ფორმატირებული სტილიზებული ტექსტის ასახვას მომხმარებლის ინტერფეისის ნებისმიერ ნაწილში, ტექსტის კომბინირებას სიებთან, ნახატებთან და სხვა ინტერფეისულ ელემენტებთან;
- ანიმაციის მოცემა დეკლარაციული დესკრიპტორებით;
- აუდიო-ვიზუალური გარემოს მხარდაჭერა ნებისმიერი აუდიო- და ვიდეოფაილების შესასრულებლად;
- სტილები და შაბლონები, რომლებიც უზრუნველყოფს ფორმატირების და (მართვის ელემენტების) ვიზუალიზების მართვის სტანდარტიზებას, აგრეთვე ამ შედეგების მრავალჯერ გამოყენება პროექტის სხვადასხვა ადგილას;
- ბრძანებები, რომლებითაც ისინი განისაზღვრება ერთ ადგილას, მაგრამ მრავალჯერადად უკავშირდება დანართის სხვა ელემენტებს;
- მომხმარებლის დეკლარაციული ინტერფეისი, რომელიც ფანჯრების ან გვერდების შინაარსს აღწერს XAML ენის საშუალებით.

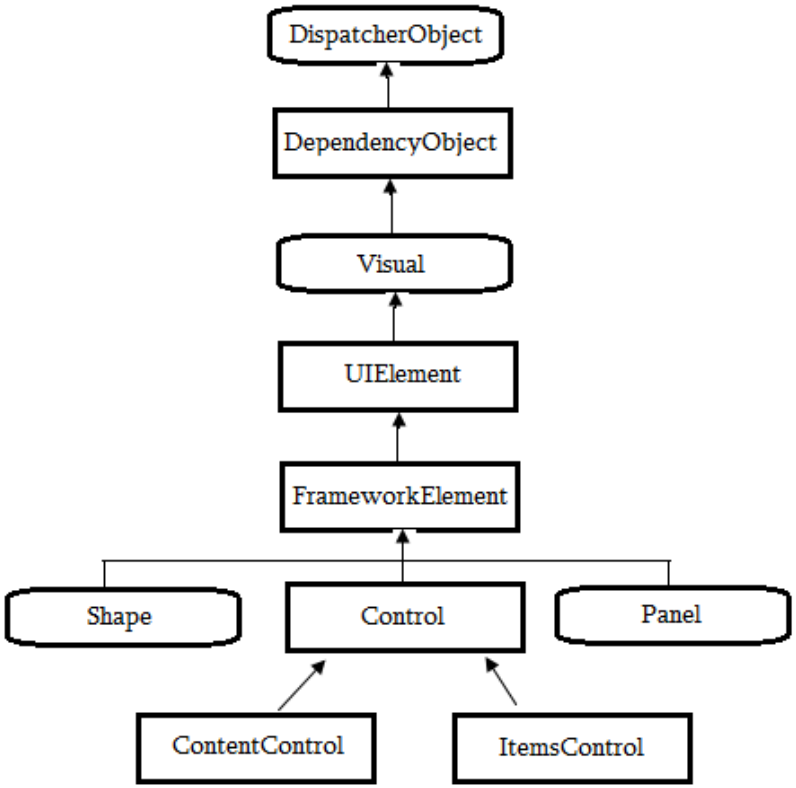
WPF არქიტექტურის ძირითადი კომპონენტები მოცემულია 6.2 ნახაზზე.



ნახ.7.2. WPF –ის არქიტექტურა

- PresentationFramework კომპონენტი შეიცავს WPF–ის ზედა დონის ტიპებს, როგორიცაა ფანჯრების, პანელების და სხვა ელემენტების წარმოდგენა;
- PresentationCore შეიცავს საბაზო ტიპებს, როგორიცაა UIElement და Visual, რომელთაგანაც იწარმოება მართვის ყველა ფორმა და ელემენტები;
- WindowsBase შეიცავს განსხვავებულ ტიპებს, რომელთა გამოყენება შეიძლება WPF–ის გარეთ. მაგალითად, DispatchObject და DependencyObject კომპონენტები;
- milcore კომპონენტი არის WPF ვიზუალიზაციის ბირთვი;
- WindowsCodecs ქვედა დონის API-ინტერფეისია გამოსახულებათა აგების მხარდასაჭერად;
- Direct 3D არის ასევე ქვედა დონის API-ინტერფეისი, რომლითაც ხორციელდება WPF –ის მთელი გრაფიკის ვიზუალიზაცია;
- User32 გამოიყენება იმის დასადგენად, თუ რომელ პროგრამას ეკრანის რომელი უბანი აქვს მიცემული.

WPF–ის არქიტექტურა განსაზღვრავს ძირითად სახელსივრცეებს კლასთა იერარქიისთვის. მართვის ელემენტების საბაზო ერთობლიობა განსაზღვრავს სისტემის კლასთა საკვანძო იერარქიებს. 7.3 ნახაზზე აბსტრაქტული კლასები ნაჩვენებია ოვალებით, ხოლო კონკრეტული კლასები – მართკუთხედებით.



ნახ.7.3. WPF ფუნდამენტური კლასები: აბსტრაქტული (მრგვალკუთხა) და კონკრეტული (მართკუთხა)

ობიექტთა უმრავლეოსობა WPF–ში იწარმოება DispatcherObject აბსტრაქტული კლასიდან. WPF ეფუძნება შეტყობინებათა გაცვლის სისტემას, რომლებიც მომხმარებლის ინტერფეისისთვის ფორმირდება ერთ ნაკადში, რომელიც იმართება და კონტროლდება დისპეტჩერის მიერ. DispatcherObject კლასი უზრუნველყოფს დანართის ინტერფეისის ყოველი ელემენტისთვის ნაკადში შესრულების შემოწმებას და წვდომას დისპეტჩერისკენ.

WPF კლასები ღებულობს მხარდაჭერას დამოკიდებულების თვისებებისგან, რომელიც წარმოიშვება DependencyObject კლასისგან.

Visual კლასი ერთეულოვანი ობიექტია, რომელიც აინკაფსულირებს ხატვის ინსტრუქციებს და დეტალებს, აგრეთვე საბაზო ფუნქციებს. WPF–ის ინტერფეისული ელემენტები წარმოქმნილ უნდა იქნას Visual კლასისგან.

მომხმარებელთა ყველა მართვის ელემენტი შთამომავალია UIElement ან FrameworkElement კლასების. UIElement კლასი უზრუნველყოფს ისეთ ფუნქციონალობას, როგორიცაა დაკომპლექტება, შეტანა, ოკუსი და მოვლენები. FrameworkElement-კლასი UIElement-ის ფუნქციონალობას უმატებს ველების განსაზღვრას, გასწორებას, მონაცემთა დაკავშირების მხარდაჭერას, ანიმაციას და სტილებს.

Shape კლასი საბაზოა ისეთი გეომეტრიული ფიგურების ასაგებად, როგორიცაა მართკუთხედი, ელიფსი, მრავალკუთხედი, წრფე და გზა.

Control კლასი განსაზღვრავს მართვის ელემენტებს, რომელთაც შეუძლია მომხმარებელთან ურთიერთობა. ესაა ღილაკები, სიები, ტექსტური ელემენტები.

ContentControl და ItemsControl კლასები საბაზოა მართვის ელემენტებისთვის, რომელთაც შეიძლება ჰქონდეს ერთადერთი მნიშვნელობა ან კოლექცია მნიშვნელობებისა, შესაბამისად.

Panel კლასი საბაზოა ყველა კონტეინერული ელემენტთა-შემადგენლობისთვის, რომლებიც შეიცავს ერთ ან მეტ შვილობილ ელემენტს.

7.3. ინტერფეისის დაკომპლექტება

აპლიკაციის მომხმარებლის ინტერფეისის დაპროექტების დროს აუცილებელია ფანჯარაში (ფორმაზე) ან გვერდზე საჭირო მართვის ელემენტების ფორმირება და შესაბამისი თვისებების განსაზღვრა, ანუ შინაარსის ორგანიზების ჩატარება. ამ პროცესს უწოდებენ დაკომპლექტებას (შედგენას).

WPF-ში დაკომპლექტება ხორციელდება სხვადასხვა კონტეინერით. ყოველ მათგანს თავისი საკუთარი დაკომპლექტების ლოგიკა აქვს. ზოგი ალაგებს ელემენტებს მიმდევრობით სტრიქონში, ზოგი ალაგებს მათ უხილავი უჯრების ბადეში.

ფანჯარას და გვერდს WPF-ში შეუძლია შეიცავდეს მხოლოდ ერთ ელემენტს -კონტეინერს. კონტეინერში შეიძლება განთავსდეს მომხმარებლის ინტერფეისის განსხვავებული ელემენტები და სხვა კონტეინერები. WPF-ში განლაგება განისაზღვრება გამოყენებული კონტეინერის ტიპით. ეს კონტეინერებია; პანელები, წარმომებული System.Windows.Controls.Panel აბსტრაქტული კლასიდან.

აპლიკაციებში გამოიყენება შემდეგი კლასები:

- Grid და UniformGrid – განალაგებს ელემენტებს უხილავი ცხრილის სტრიქონებსა და სვეტებში, შესაბამისად;
- StackPanel – განალაგებს ელემენტებს ჰორიზონტალურ ან ვერტიკალურ სვეტში.
- WrapPanel –განალაგებს ელემენტებს ხელმისაწვდომ სივრცეში, ერთ სტრიქონად ან სვეტად;
- DockPanel - განალაგებს ელემენტებს ერთ-ერთი სასაზღვრო გვერდის შეფარდებით;
- Frame – ანალოგიურია StackPanel-ის, მაგრამ უფრო მოსახერხებელია გვერდების გადასასვლელბის ორგანიზებისთვის.

Grid არის WPF-ის ყველაზე მძლავრი კონტეინერი. ის, რასაც სხვა კონტეინერები ასრულებენ ცალკ-ცალკე, შეიძლება Grid-ში შერულდეს. იგი იდეალური ინსტრუმენტია ფანჯრის (გვერდის) დასაყოფად შედარებით მცირე ზომის არეებად, რომელთა მართვა განხორციელდება სხვა პანელებით. Grid ანაწილებს ელემენტებს უხილავი ბადის სტრიქონებსა და სვეტებში. ბადის ერთ უჯრაში მიზანშეწონილია ერთი ელემენტის მოთავსება, რომელიც, საჭიროების შემთხვევაში, თვითონ შეიძლება იყოს სხვა კონტეინერი, რომელშიც განლაგდება საკუთარი მართვის ელემენტთა ჯგუფი.

StackPanel - ერთ-ერთი უმარტივესი კონტეინერია. იგი ალაგებს თავის შვილობილ ელემენტებს ერთ სტრიქონში ან სვეტში.

UniformGrid კონტეინერი, Grid-ისგან განსხვავებით, მოითხოვს მხოლოდ სტრიქონების და სვეტების რაოდენობის მითითებას, და აფორმირებს ერთი ზომის უჯრებს, რომელთაც დაკავებული აქვს ფანჯრის (გვერდის) ან ჩადგმული კონტეინერის ელემენტის მთელი ხელმისაწვდომი არე.

WrapPanel აწესრიგებს ელემენტების განლაგებას პანელზე Orientation თვისების შესაბამისად, ჰორიზონტალურად (Horizontal) ან ვერტიკალურად (Vertical). სტრიქონის ან სვეტის შევსების შემდეგ მომდევნო ელემენტი გადადის ახალ სტრიქონზე ან სვეტზე.

DockPanel უახლოვებს მართვის ელემენტებს მის რომელიმე მხარეს, Dock თვისების შესაბამისად, რომელიც იქნება Left, Right, Top ან Bottom. ელემენტის სიმაღლე განისაზღვრება MaxHeight პარამეტრით.

Frame მართვის ელემენტია შიგთავსისთვის, რომელიც იძლევა შესაძლებლობას შიგთავსზე ან მის ასახვაზე გადასასვლელად. Frame შეიძლება მოთავსდეს სხვა შიგთავსის შიგნით, როგორც სხვა ელემენტები. შიგთავსი შეიძლება იყოს .NET Framework-ის და HTML-ფაილების ნებისმიერი ტიპი.

ჩვეულებისამებრ, Frame გამოიყენება შიგთავსის ჩასაწყობად, რომელიც განსაზღვრავს გადასასვლელებს გვერდებზე.

დაკომპლექტების თვისებები განისაზღვრება კონტეინერით, მაგრამ შვილობილი ელემენტებიც ახდენს მასზე გალენას. დაკომპლექტების პანელები მუშაობს შვილობილ ელემენტებთან შეთანხმებით, შემდეგი თვისებების საუძველზე:

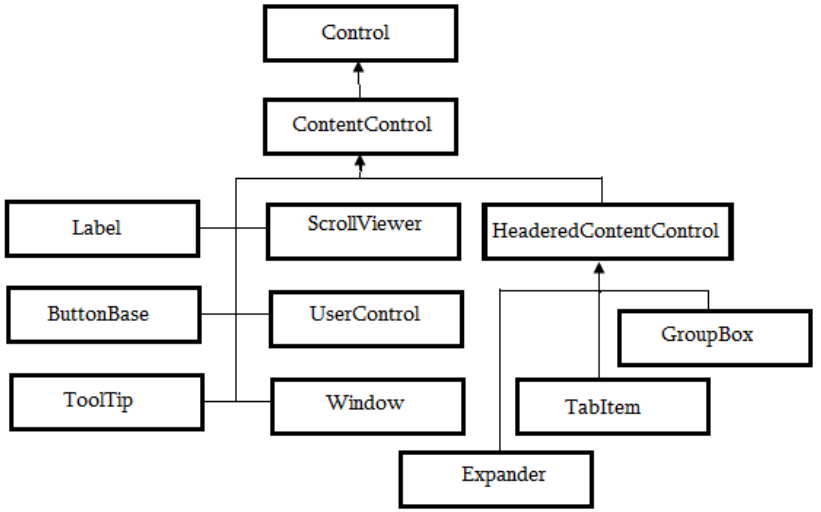
- HorizontalAlignment და VerticalAlignment - განსაზღვრავს, თუ როგორ პოზიციონირებს შვილობილი ელემენტი კომპლექტის შიგნით, როცა არსებობს დამატებითი ჰორიზონტალური/ვერტიკალური სივრცე;
- Margin - ამატებს ცარიელ სივრცეს ელემენტის ირგვლივ;
- MinWidth და MaxWidth აყენებს ელემენტის მაქსიმალურ ზომებს;
- Width и Height – ცხადად აყენებს ელემენტის ზომებს.

WPF-ში არსებობს კონტეინერები, რომლებსაც აქვს მართვის ელემენტები მოცემული კოორდინატების შესაბამისად, რომლებიც ზომებითაა მოცემული. ეს კონტეინერებია Canvas და InkCanvas. ისინი ძირითადად გამოიყენება გრაფიკული პრიმიტივების და ფიგურების ვიზუალიზაციისთვის.

7.4. მართვის ელემენტები და შიგთავსები

მართვის ელემენტების დანიშნულებაა მომხმარებელის ინტერაქტიული კავშირის მხარდაჭერა. მათ შეუძლია ფოკუსის და შემავალი მონაცემების მიღება კლავიატურიდან ან მაუსიდან.

შიგთავსის მართვის ელემენტები სპეციალიზებული ტიპის მართვის ელემენტებია, რომლებიც ინახავს განსაზღვრულ შიგთავსს - ერთ ან რამდენიმე ელემენტს. შიგთავსის ყველა მართვის ელემენტი მემკვიდრეებია ContentControl კლასისა (ნახ.6.4).



ნახ. 7.4. შიგთავსის მართვის ელემენტების იერარქია

ContentControl კლასი მემკვიდრეა System.Windows.Control კლასის, რომელიც მას და მის შვილობილ კლასებს ანიჭებს საბაზო ნახასიათებლებს, რომლებიც:

- საშუალებას იძლევა განისაზღვროს შიგთავსი მართვის ელემენტის შიგნით;
- საშუალებას იძლევა განისაზღვროს გადასვლების მიმდევრობა Tab-კლავიშით;
- ხელს უწყობს ფონის, წინა პლანის და ჩარჩოს ხატვას;
- ხელს უწყობს ტექსტური შინაარსის ზომის და შრიფტის ფორმატირებას.

მართვის ელემენტებს აქვს ფონი (ელემენტის ზედაპირი) და ტექსტი (ფონზე მოთავსებული). მათი ფერები WPF-ში განისაზღვრება Background და Foreground თვისებებით. ისინი იყენებს ფუნჯს - ობიექტი Brush. იგი უზრუნველყოფს ფონის და

ტექსტის შევსებას მთლიანი ფერით (ფუნჯის კლასი SolidColorBrush), ან გრადიენტულით, მაგალითად, LinearGradientBrush ფუნჯის კლასის დახმარებით.

მაგალითად, ღილაკის ფონის მისაცემად Brush ობიექტით აუცილებელია SolidColorBrush ფუნჯის Color თვისებას მიენიჭოს ფერის მნიშვნელობა, მაგალითად, Blue - ლურჯი.

```
<Button> ღილაკი A
   <Button.Background>
      <SolidColorBrush
Color="Blue"></SolidColorBrush>
   </Button.Background>
</Button>
```

WPF-ში შესაძლებელია მოკლე ფორმის გამოყენებაც. მაგალითად, იგივეს ექნება ასეთი სახე:

```
<Button Background="Black"
        Foreground="Red">ღილაკი A</Button>
```

ამ დროს WPF-ის სინტაქსური ანალიზატორი ავტომატურად შექმნის SolidColorBrush ობიექტებს მითითებული ფერებით და გამოიყენებს ამ ობიექტებს ფონისა და ტექსტისათვის.

თუ საჭიროა გრადიენტული ტიპის LinearGradientBrush ფუნჯის გამოყენება, მაშინ ფუნკის ობიექტი უნდა შეიქმნას დამოუკიდებლად:

```
<Button>Кнопка A
  <Button.Background>
    <LinearGradientBrush>
    <LinearGradientBrush.GradientStops>
       <GradientStop Offset="0.0"
Color="Blue"></GradientStop>
       <GradientStop Offset="1.0"
Color="Red"></GradientStop>
    </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Button.Background>
</Button>
```


ფუნჯის დახმარებით იხატება ჩარჩო მართვის ელემენტის ირგვლივ. ეს ხორციელდება BorderBrush და BorderThickness თვისებებით. ამ დროს BorderBrush თვისება იღებს არჩეულ ფუნჯს, ხოლო BorderThickness – ჩარჩოს სიგანეს:

```
    <Button Width="80" Height="30"
Background="Red"
    BorderBrush="Blue" BorderThickness="5">ღოლაკი
A</Button>
```

WPF ტექნოლოგია უზრუნველყოფს გამჭვირვალობას. თუ ფორმაზე ან გვერდზე განლაგებულია რამდენიმე ელემენტი ერთიმეორეზე, და მათთვის განსაზღვრულია სხვადასხვა ხარისხის გამჭვირვალობა, მაშინ ქვედა ელემენტები (წარწერებით) გამოჩნდება ზედა ელემენტების შიგნით. ეს საშუალებას იძლევა შეიქმნას მრავალშრიანი ანომაციური ობიექტები. გამჭვირვალობა შეიძლება მოიცეს ორი ხერხით:

- თვისებით Opacity (არაგამჭვირვალე), რომელიც იღებს მნიშვნელობას საზღვრებში 0-1. აქ 1 არის სრულიად არაგამჭვირვალე, ხოლო 0 - სრულად გამჭვირვალე;
- ნახევრადგამჭვირვალე ფერის გამოყენებით, ალფა-არხის მნიშვნელობის მიწოდებით. თუ ალფა-არხის მნიშვნელობა 255-ზე ნაკლები, მაშინ ის ნახევრადგამჭვირვალეა.

Control კლასი განსაზღვრავს შრიფტების თვისებათა ერთობლიობას:

- FontFamily – შრიფტის სახელი მართვის ელემენტში;
- FontSize – შრიფტის ზომა (1/96 დიუმი);
- FontStyle – ტექსტის დახრის მითითება;
- FontWeight – ტექსტის წონა;
- FontStretch – სიდიდე, რომლის მიხედვითაც გაიწელება ან შეიკუმშება ტექსტი.

მართვის ელემენტისთვის ტექსტის შრიფტის არჩევისას უნდა მიეთითოს შრიფტების ოჯახის სრული სახელი:

```
<Button FontFamily="Times New Roman" FontSize="18">ღოლაკი A</Button>
```

ნახევრადსქელი, დახრილი შრიფტის ასარჩევად უნდა მიეცეს შემდეგი თვისებები:

```
<Button FontFamily="Times New Roman"
FontSize="18" FontStyle="Italic" FontWeight="Bold"> ღილაკი A</Button>
```

WPF-ის მრავალი მართვის ელემენტი არის შიგთავსის მართვის ელემენტები. ესაა: Label, Button, CheckBox და RadioButton.

Label - ჭდე: არის შიგთავსის უმარტივესი მართვის ელემენტი. ამ ფუნქციისთვის გამოიყენება Target თვისება, რომელსაც ენიჭება მისაბმელი გამოსახულება. აქ უნდა მიეთითოს სხვა მართვის ელემენტი, რომელზეც გადავა ფოკუსი სწრაფი წვდომის ღილაკის დაჭერისას.

```
<Label Target="{Binding ElementName = txtA}">არჩევა_A</Label>
<TextBox Name="txtA">ტექსტის არჩევა</TextBox>
```

ჭდის ტექსტში რომელიმე ასოზე ხაზგასმა მიუთითებს სწრაფი წვდომის ღილაკზე. ასეთ მნემონიკურ ბრძანებებში ერთდროულად მუშაობს <Alt+A>, სადაც A ხაზგასმული ასოა. ჩვენი მაგალითისათვის ფოკუსი გადაეცემა მართვის ელემენტს TextBox, რომლის სახელია txtA.

WPF-ში განსაზღვრულია სამი კლასი: Button, CheckBox და RadioButton, რომლებიც ButtonBase კლასის მემკვიდრეებია. ButtonBase კლასი განსაზღვრავს Click მოვლენას და უმატებს იმ ბრძანებათა მხარდაჭერას, რომლებიც უზრუნველყოფს ღილაკების მიერთებას დანართის ამოცანებისათვის. აქვეა ClickMode თვისება,

რომელიც განსაზღვრავს თუ როდის აგენერირებს ღილაკი Click-მოვლენას მაუსის მოქმედების საპასუხოდ. გამოუცხადებლად ესაა ClickMode.Release მნიშვნელობა, რომელიც ნიშნავს მოვლენის გენერირებას მაუსის ღილაკის დაჭერის ან აშვების დროს. ClickMode.Press - მხოლოდ დაჭერის დროს, ClickModeHover - როცა მაუსის კურსორი ღილაკზე დადგება და შეჩერდება.

Button კლასსი ამატებს ორ თვისებას: IsCancel და IsDefault. როცა IsCancel = true, მაშინ ღილაკი იმუშავებს როგორც ფანჯრის შეცვლა, <Esc>-ის მსგავსად. თუ IsDefault= true, მაშინ ღილაკი ითვლება აქტიურად გამოუცხადებლად.

CheckBox და RadioButton კლასები არის ToggleButton კლასის მემკვიდრეები, რომელიც ასახავს ღილაკს ორი მდგომარეობით: „დაჭერილია“ და „აშვებულია“. ამ კლასში არის მოვლენები: Checked, Unchecked და Intermediate, რომლებიც გენერირდება ღილაკის ჩართვის, ამორთვის ან განუსაზღვრელ მდგომარეობაში გადასვლისას.

CheckBox ღილაკისთვის ჩართვა ნიშნავს „ალმის“ (‘ v ’) დაყენებას. IsChecked თვისებას, რომელიც ToggleButton-იდანაა ნაანდერძევი, შეუძლია სამი მნიშვნელობის მიღება: true (ჩართული), false (გამორთული), null (განუსაზღვრელი, რომელიც ჩანს როგორც ჩამუქებული ფანჯარა, და გამოიყენება როგორც შუალედური მდგომარეობა). XAML-აღწერა CheckBox-ის ამ სამი მდგომარეობისა ნაჩვენებია ქვემოთ:

```
<CheckBox Height="16" Name="checkBox1"
      Width="120" IsChecked="False"
          ClickMode="Release">არჩევა A</CheckBox>
<CheckBox Height="16" Name="checkBox2"
      Width="120" IsChecked="True"
          ClickMode="Press">არჩევა B</CheckBox>
<CheckBox Height="16" Name="checkBox3"
      Width="120" IsChecked="{x:Null}"
          ClickMode="Hover">არჩევა C</CheckBox>
```

RadioButton ღილაკისთვის დამატებულია GroupName თვისება, რომელიც უზრუნველყოფს გადამრთველების განლაგების მართვას ჯგუფში. ჯგუფიდან შეიძლება მხოლოდ ერთი გადამრთველის არჩევა.

```
   <GroupBox Header="რადიოღილაკების ჯგუფი"
Height="100"
Name="groupBox1" Width="200">
  <StackPanel>
   <RadioButton Height="16" Name="radioButton2"
Width="120">არჩევა D</RadioButton>
   <RadioButton Height="16" Name="radioButton1"
 Width="120">არჩევა E</RadioButton>
   <RadioButton Height="16" Name="radioButton3"
Width="120">არჩევა F</RadioButton>
  </StackPanel>
</GroupBox>
```

კონტექსტური ფანჯრის (მაუსის კურსორის მიტანისას მართვის ელემენტზე გამოჩნდება პატარა ფანჯარა ტექსტით, ხოლო კურსორის მოცილებისას - გაქრება) გამოტანა შეიძლება `ToolTip` კლასის ან თვით მართვის ელემენტის `ToolTip` თვისების გამოყენებით.

```
<Button ToolTip="ღილაკი A კონტექსტური ტექსტით
                 "></Button>
```

ScrollViewer მართვის ელემენტი უზრუნველყოფს შიგთავსის დათვალიერებას ტექსტის ეკრანზე გადახვევით. ამ ელემენტით იქმნება ტექსტების გადახვევის შესაძლებლობის მქონე პანელები.

UserControl ელემენტის დანიშნულებაა მომხმარებლის კონტროლის ელემენტების შექმნა, რომლებშიც ერთიანდება რამდენიმე სხვა ელემენტი.

Windows ელემენტი შიგთავსის მართვის ელემენტია და გამოიყენება დანართის ყველა ფანჯრის შესაქმნელად.

HeaderedContentControl ელემენტი მემკვიდრეა ContentControl კლასის და მშობელია იმ ელემენტებისა, რომელტაც შიგთავსის გარდა აქვს სათაურის არე.

GroupBox ელემენტი არის ფანჯარა სათაურით და გამოიყენება დაკავშირებული ელემენტების დაჯგუფებისათვის, მაგალითად, რადიოღილასკები.

TabItem მართვის ელემენტი ასახავს გვერდს TabControl კლასისთვის. ეს ელემენტი ასახავს იმ ჩადგმულ გვერდს TabControl პანელში, რომელიც აქტიურია ამ წუთში.

Expander მართვის ელემენტი უზრუნველყოფს შიგთავსის განსაზღვრული უბნის გამოჩენას ან დამალვას.

7.5. ტექსტური მართვის ელემენტები

WPF-ში განსაზღვრულია შემდეგი ტექსტური მართვის ელემენტები: TextBlock, TextBox, RichTextBox და PasswordBox. ელემენტი PasswordBox მემკვიდრეა Control კლასის,ხოლო ელემენტები TextBox და RichTextBox - მემკვიდრეა TextBase კლასის.

ელემენტი TextBlock გამოიყენება მცირე ზომის ტექსტისთვის. ელემენტი TextBox ინახავს ტექსტის სტრიქონს. RichTextBox ელემენტს შეუძლია FlowDocument-ის შიგთავსის შენახვა, რომელშიც შეიძლება იყოს ელემენტთა რთული კომბინაცია. ელემენტი PasswordBox შეიცავს ტექსტის სტრიქონს, იყენებს SecureString ელემენტს დასაცავად ზოგიერთი სახის თავდასხმებისგან. ელემენტი TextBox, ჩვეულებრივად ინახავს ერთ სტრიქონს: <TextBox Name="txtA">ტექსტის არჩევა</TextBox>.

თუ საჭიროა მრავალსტრიქონიანი წარმოდგენის შექმნა, მაშინ თვისებას TextWrapping მიენიჭება Wrap მნიშვნელობა. ასეთი TextBox-ისთვის შეიძლება მინიმალური და მაქსიმალური სტიქონების რაოდენობის მითითება, MinLines და MaxLines თვისაებათა გამოყენებით.

7.6. სიების მართვის ელემენტები

ListBox და ComboBox - სიების მართვის ელემენტებია. ისინი მემკვიდრეებია ItemsControl კლასისა Selector კლასის გავლით.

ItemsControl კლასი საბაზოა სიების მართვის ყველა ელემენტისათვის და ითვალისწინებს სიის ელემენტების შევსების ორ ხერხს. პირველში ხდება ელემენტის დამატება უშუალოდ Items კოლექციაში, რომლის დროსაც გამოიყენება კოდი ან XAML. მეორე ხერხით ხდება მონაცემთა მიბმა ItemsSource თვისებით (მონაცემთა წყარო). Selector კლასს აქვს თვისებები, რომლებიც თვალყურს ადევნებს მოცემულ დროის მომენტში გამოყოფილ სიის ელემენტს (SelectedItem) ან მის პოზიციას (SelectedIndex).

ელემენტთა დასამატებლად ListBox-ში შეიძლება ჩაიდოს ListBoxItem-ის ელემენტები ListBox-ში, როგორც ეს ქვემოთაა ნაჩვენები ფერთა სიის შესადგენად (მწვანე, ცისფერი, ყვითელი, წითელი):

```
<ListBox>
   <ListBoxItem>მწვანე</ListBoxItem>
    <ListBoxItem>ცისფერი</ListBoxItem>
    <ListBoxItem>ყვითელი</ListBoxItem>
   <ListBoxItem>წითელი</ListBoxItem>
</ListBox>
```

მართვის ელემენტი ListBox ინახავს თავის კოლექციაში ყოველ ჩადგმულ ობიექტს. ამავდროულად, ListBoxItem-ს შეუძლია არა მხოლოდ სტრიქონების შენახვა, არამედ ნებისმიერი თავისუფალი ელემენტის.

მართვის ელემენტი ComboBox მსგავსია ListBox-ის, ოღონდ ვიზუალიზაციისთვის გამოიყენებს ჩამოშლად სიას, რომლიდანაც მომხმარებელი ირჩევს მხოლოდ ერთ ელემენტს.

7.7. სპეციალიზებული მართვის ელემენტები

WPF-ში არის მართის ელემენტები, რომლებიც იყენებს მნიშვნელობათა დიაპაზონებს: ScrollBar, ProgressBar და Slider. მართვის ეს ელემენტები მემკვიდრეებია RangeBase კლასის, რომელშიც განსაზღვრულია ისეთი თვისებები, როგორიცაა Value (ელემენტის მიმდინარე მნიშვნელობა), Minimum და Maximum დასაშვები მნიშვნელობები.

ScrollBar მართვის ელემენტი იძლევა გადახვევის ზოლს გადაადგილებადი ელემენტით, რომლის პოზიცია შეესაბამება განსაზღვრულ მნიშვნელობას.

ProgressBar მართვის ელემენტი უჩვენებს ხანგრძლივი ამოცანის შესრულების მსვლელობას.

Slider მართვის ელემენტი გამოიყენება რიცხვითი მნიშვნელობის მოსაცემად კურსორის გადაადგილების შესაბამისად გადახვევის სახაზავზე.

7.8. ბრძანებები

WPF-ის ბრძანებათა მოდელი იძლევა შემდეგ შესაძლებლობებს:

- მოვლენათა დელეგირება შესაფერის ბრძანებებზე;
- მართვის ელემენტის ჩართული მდგომარეობის შენარჩუნება სინქრონიზებული სახით შესაბამისი ბრძანების მდგომარეობის დახმარებით.

ბრძანებათა მოდელი შეიცავს შემდეგ კომპონენტებს:

- ბრძანებები, რომლებიც წარმოადგენს დანართის ამოცანას, მათი წვდომის თვალყურის დევნის მექანიზმით პროგრამის შესრულების დროს;
- ბრძანებათა მიბმა, რომელიც გულისხმობს ბრძანების მიერთებას დანართის ლოგიკასთან, რომელიც პასუხისმგებელია მომხმარებლის ინტერფეისის გარკვეული უბნის მომსახურებაზე;
- ბრძანების წყარო, რომელსაც იგი აინიცირებს;
- ბრძანების მიზნობრივი ობიექტები - დანართის ელემენტები, რომლებისთვისაც არის გამიზნული ეს ბრძანება.

WPF-ის ბრძანებათა კლასები უნდა უჭერდეს მხარს ICommand ინტერფეისს.

```
public interface ICommand
    {
        void Execute(object par);
        bool CanExecute(object par);
        event EventHandler CanExecuteChanged;
    }
```

მეთოდი Execute() განსაზღვრავს ბრძანების რეალიზაციას დანართში. ეს შეიძლება იყოს დანართის ლოგიკის კოდი ან მოვლენის ამოქმედება, რომელიც მუშავდება დანართის სხვა კლასში. მეთოდი CanExecute() აბრუნებს ინფორმაციას ბრძანების წვდომის მდგომარეობაზე. Execute() და CanExecute() მეთოდები შეიძლება გამოძახებულ იქნას **par** პარამეტრით, რომელიც გამოიყენება დამატებითი ინფორმაციის გადასაცემად. CanExecuteChanged მოვლენა გამოიძახება ბრძანების წვდომის მდგომარეობის შეცვლისას. ამ შემთხვევისთვის გამოიძახება CanExecute() მეთოდი და მოწმდება ბრძანების მდგომარეობა.

WPF-ში არსებობს ბიბლიოთეკა საბაზო ბრძანებების, რომლებიც ხელმისაწვდომია შემდეგი სტატიკური კლასების სტატიკური თვისებებიდან:

- ApplicationCommands – წარმოდგენილია ზოგადი ბრძანებები, მაგ., Create, Open, Colpy, Store და სხვ. ;
- Navigation Commands – ბრძანებები ნავიგაციისთვის;
- Editing Commands – ბრძანებები დოკუმენტაციის რედაქტირებისთვის;
- Media Commands – მულტიმედიასთან სამუშაო ბრძანებები.

ბრძანების ინიციალიზაციისთვის საჭიროა ბრძანების წყაროს გამოყენება, ხოლო მასზე საპასუხოდ - ბრძანების მიბმა მისი შესრულების გადამისამართებასთან ჩვეულებრივ მოვლენის დამმუშავებელზე.

დავუშვათ, რომ **შექმნა** ბრძანების წყარო არის ღილაკი New. Command თვისების დახმარებით ღილაკს ვაბამთ New ბრძანებას.

```
<Button Name="New" Content="შექმნა" Width="200"
                Command="New"/>
```

შემდეგ ბიჯზე აუცილებელია ბრძანების მიბმა Window ფანჯარასთან.

```
<Window.CommandBindings>
        <CommandBinding Command="New"
                Executed="CommandBinding_Executed"

        CanExecute="CommandBinding_CanExecute"/>
</Window.CommandBindings>
```

მიბმის დროს CommandBinding ობიექტისთვის, გარდა New ბრძანებისა, საჭიროა დამმუშავებლების მითითება Execute() და CanExecute( ) მეთოდებისთვის, შესაბამისად Executed და CanExecute თვისებებით. დამმუშავებელი CommandBinding_Executed რეალიზებას გაუკეთებს **შექმნა**-ბრძანების ბიზნეს-ლოგიკას, ხოლო დამმუშავებელი CommandBinding_CanExecute კი - ბრძანების წვდომის შემოწმებას.

მომხმარებლის ბრძანების შექმნისას მიზანშეწონილია RoutedCommand კლასის გამოყენება, რომელიც რეალიზაციას უკეთებს ICommand ინტერფეისს.

საილუსტრაციოდ განვიხილოთ **რედაქტირება**-ბრძანების კონსტრუირება. შევქმნათ DataCommands კლასი.

```
public class DataCommands
{
     private static RoutedCommand edit;

    public static RoutedCommand Edit
    {
        get { return DataCommands.edit; }
    }
```

```
    static DataCommands()
    {
        InputGestureCollection inputs = new
InputGestureCollection();
        inputs.Add(new KeyGesture(Key.E,
ModifierKeys.Control, "Ctrl+E"));
        Edit = new RoutedCommand("Edit",
typeof(DataCommands), inputs);
    }
}
```

DataCommands კლასში გამოცხადებულია RoutedCommand კლასის ეგზემპლარის სტატიკური ველი (edit) და თვისება (Edit). სტატიკური კონსტრუქტორი ქმნის **შექმნა**-ბრძანების (Edit) ეგზემპლარს, იყენებს რა InputGestureCollection-კლასის (inputs)-ეგზემპლარს. ეს კლასი არის KeyGesture ობიექტების მოწესრიგებული კოლექცია, რომლებიც განსაზღვრავს „ცხელ“ ღილაკს ბრძანების გამოსაძახებლად. მაგალითში ესაა „Ctrl+E”. შემდეგ აუცილებელია ბრძანებისთვის დაყენდეს წყარო და მიებას ბრძანება, როგორც ეს ადრე იყო განხილული.

7.9. რესურსები

ობიექტური რესურსი - ესაა .NET-ობიექტი, რომელიც განისაზღვრება ერთ ადგილას და გამოიყენება რამდენიმე სხვა ადგილას დანართში. ეს რესურსი განისაზღვრება XAML ფორმატირების ენაში [4,8]. ობიექტურ რესურსებს აქვს რიგი მნიშვნელოვანი მახასიათებლები:

- ეფექტურობა კოდის მრავალჯერადი გამოყენების პრინციპის რეალიზაციის გამო;
- მოხერხებული თანხლება ფორმატირების დაბალი დონის დეტალების გადატანის გამო ერთ ცენტრალურ ადგილას;
- ადაპტაციის უნარი ინფორმაციის დინამიკურად ცვლილების შესაძლებლობის გამო, რომელიც კოდისგან გამოყოფილია.

ყოველი ელემენტი, რომელიც მემკვიდრეა FrameworkElement კლასის (ნახ.7.2) შეიცავს Resources თვისებას, რომელშიც ინახება რესურსების Resourcesdictionary-ლექსიკონური კოლექცია. აქ შეიძლება ინახებოდეს ნებისმიერი ტიპის ობიექტი. ყოველ ელემენტს აქვს წვდომა როგორც საკუთარ რესურების კოლექციასთან, ასევე თავისი მშობლების რესურსების კოლექციებთან. ხშირად რესურსები ინახება ფანჯრის ან გვერდის დონეზე მათი ერთობლივად გამოყენების უზრუნველსაყოფად ყველა მართვის ელემენტის მიერ.

რესურსები განისაზღვრება Resource სექციაში შესაბამისი ელემენტისთვის, მაგ., ფანჯრისთვის:

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

ყოველი რესურსი ჩაიწერება ასეთი სახით:

```
<TypeResource  x:Key="რესურსის გასაღები" >
        ...
</ TypeResource >
```

(TypeResource)- ობიექტის თვისებების მისაცემად შეიძლება გამოყენებულ იქნას ატრიბუტთა სინტაქსი ან ელემენტთა თვისებათა სინტაქსი. XAML ელემენტები მიმართავენ რესურსებს გასაღებით (KeyResource), რომელიც განისაზღვრება ატრიბუტით x:Key. რესურსის გასაღები მოიცემა StaticResource-ს გამოყენებით.

მაგალითად, დანართში შექმნილია გრადიენტული ფუნჯი ღილაკის ფერით შესავსებად, რომელიც გამოყენებულ უნდა იქნას ინტერფეისის სხვა ელემენტებში. მაშინ გრადიენტული ფუნჯი განსაზღვრული უნდა იყოს როგორც რესურსი:

```
<Window.Resources>
   <LinearGradientBrush x:Key="BrushButton"
         EndPoint="0,1" StartPoint="0,0">
     <GradientStop Color="#FFD2E995" Offset="0.9"/>
     <GradientStop Color="#FF74EDB3" Offset="0.55"/>
     <GradientStop Color="#FF72D8A7" Offset="0.75"/>
     <GradientStop Color="#FF233CC6" Offset="0.25"/>
   </LinearGradientBrush>
</Window.Resources>
<StackPanel>
    <Button x:Name="New" Content="შექმნა" Width="200"
         Command="New" Margin="154.5,0"
        Background="{StaticResource BrushButton}"/>
</StackPanel>
```

გრადიენტულ ფუნჯს აქვს `LinearGradientBrush` ტიპი და მისთვის მოცემულია გასაღები `BrushButton`. ღილაკის XAML-აღწერაში ატრიბუტი `Background` განისაზღვრება გაფართოებული ფორმატით `{StaticResource BrushButton}`.

ჩვენ მაგალითში გამოყენებულია სტატიკური რესურსი. იგი ამოიღება რესურსების კოლექციიდან მხოლოდ ერთხელ. ამასთანავე, ნებისმიერი ცვლილებები რესურსის ობიექტში შეიმჩნევა მყისიერად. თუ დანართის ფუნქციონირების პროცესში შექმნილი რესურსი შეიძლება შეიცვალოს, მაშინ გამოყენებულ უნდა იქნას დინამიკური რესურსი - DynamicResource. იგი განისაზღვრება რესურსების კოლექციაში ყოველთვის, როცა ამის საჭიროება აღმოცენდება.

რესურსები შეიძლება განისაზღვროს ელემენტის, კონტეინერის, ფანჯრის ან გვერდის, დანართის, სისტემის ან ნაკრების დონეზე.

რესურსების ხელმეორედ გამოსაყენებლად იქმნება რესურსების ლექსიკონები. ესაა XAML-ფაილი რესურსების შესანახად. თუ მაგალითად, ჩვენ მიერ შექმნილი გრადიენტული ფუნჯის რესურსს გადავიტანთ ლექსიკონში, მაშინ რესურსების ლექსიკონის MyDictionary.xaml ფაილს ექნება შემდეგი სახე:


```
<ResourceDictionary
  xmlns="http://schemas.microsoft.com/winfx/2006/
                                      xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <LinearGradientBrush x:Key="BrushButton"
                  EndPoint="0,1" StartPoint="0,0">
       <GradientStop Color="#FFD2E995" Offset="0.9"/>
       <GradientStop Color="#FF74EDB3" Offset="0.55"/>
       <GradientStop Color="#FF72D8A7" Offset="0.75"/>
       <GradientStop Color="#FF233CC6" Offset="0.25"/>
    </LinearGradientBrush>
</ResourceDictionary>
```

რესურსების ლექსიკონის გამოსაყენებლად იგი გაერთიანებულ უნდა იქნას რესურსების კოლექციასთან, მაგალითად ფანჯრის:

```
<Window.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary
                 Source="MyDictionary.xaml" />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Window.Resources>
```

რესურსების ლექსიკონის გაერთიანება სხვა ობიექტის რესურსეთან წარმოებს ResourceDictionary.MergedDictionaries თვისების მოცემით. MergedDictionaries კოლექცია - ესაა ResourceDictionary ობიექტების კოლექცია, რომლებიც უნდა დაემატოს რესურსებუს კოლექციას. დასამატებელი რესურსების ლექსიკონები მოიცემა როგორც Source ატრიბუტის მნიშვნელობები. განხილულ მაგალითში ესაა რესურსების ლექსიკონის ფაილი MyDictionary.xaml.

7.10. სტილები

სტილები - ესაა თვისებათა მნიშვნელობების კოლექციები, რომლებიც გამოიყენება ელემენტისთვის. ისინი საშუალებას იძლევა განისაზღვროს ფორმატირების მახასიათებლების ზოგადი ერთობლიობა და გამოყენებულ იქნას მთელი დანართის ფარგლებში შეთანხმებადობის უზრუნველსაყოფად. WPF-ში სტილებს შეუძლია დამოკიდებულებათა ნებისმიერი თვისების დაყენება. მათი გამოყენება შეიძლება რომელიმე ელემენტის ვიზუალური ქცევის სტანდარტიზაციისთვის. WPF-ის სტილებს აქვს ტრიგერები, რომლებიც იძლევა საშუალებას ელემენტის სტილის შესაცვლელად სხვა თვისებების შეცვლის დროს. სტილები იყენებს შაბლონებს მართვის ელემენტების სტანდარტული ვიზუალური წარმოდგენის ხელახლა განსაზღვრისთვის.

სტილი იქმნება Style კლასის ბაზაზე, რომლის თვისებები მოცემულია 7.1 ცხრილში.

Style კლასის თვისებები **ცხრ.7.1**

სახელი	აღწერა
BasedOn	აბრუნებს ან იძლევა განსაზღვრულ სტილს, რომელიც საბაზოა მიმდინარე სტილისთვის
Dispatcher	აბრუნებს Dispatcher ობიექტს, რომელთანაც კავშირშია ეს DispatcherObject ობიექტი
IsSealed	აბრუნებს მნიშვნელობას, რომელიც მიუთითებს, არის თუა არა სტილი მხოლოდ წაკითხვის
Resources	აბრუნებს ან იძლევა რესურსების კოლექციას, რომლებიც გამოყენებულ იქნება მოცემული სტილის ხილვადობის არეში
Setters	აბრუნებს Setter და EventSetter ობიექტების კოლექციას
TargetType	აბრუნებს ან იძლევა ტიპს, რომლისთვისაც დანიშნულია ეს სტილი
Triggers	აბრუნებს TriggerBase ობიექტთა კოლექციას, რომლებიც იყენებს თვისებების მნიშვნელობებს მოცემული პირობრბის საფუძველზე

მაგალითად, დანართში საჭიროა მრავალჯერადი გამოყენების ღილაკები, რომელთაც აქვს ღია-ცისფერი ფონი და ცისფერი ჩარჩო.

```
<Style x:Key="ButtonStyle" TargetType="Button">
    <Setter Property="Background"
                        Value="LightBlue"/>
    <Setter Property="BorderBrush" Value="Blue" />
</Style>
```

სახელი მიეცემა ატრიბუტით x:Key (ButtonStyle). ატრიბუტი TargetType განსაზღვრავს ტიპს, რომლისთვისაც გამოყენებულ იქნება სტილი (Button). ყოველი Setter-ობიექტი აყენებს ელემენტში ერთ თვისებას, რომელიც აუცილებლად უნდა იყოს დამოკიდებულების თვისება.

თვისებების დაყენება წარმოებს Property ატრიბუტის დახმარებით, რომელიც განსაზღვრავს თვისების სახელს, და Value - მის მნიშვნელობას.

სტილში შეიძლება EventSetters ობიექტის დამატება, რომელსაც მიებმება მოვლენა გარკვეული დამმუშავებლისთვის.

ტრიგერები იძლევა სტილის შეცვლის უფლებას განსაზღვრული პირობების შესრულებისას.

დავამატოთ ButtonStyle ღილაკის სტილში ტრიგერი, რომელიც დააფორმირებს წითელ ფონს, როცა მაუსის კურსორი დადგება ღილაკზე.

```
<Style x:Key="ButtonStyle" TargetType="Button">
    <Setter Property="Background"
                        Value="LightBlue"/>
    <Setter Property="BorderBrush" Value="Blue" />
    <Style.Triggers>
       <Trigger Property="IsMouseOver"
                                Value="true">
          <Setter Property="Background"
                                Value="Red" />
       </Trigger>
    </Style.Triggers>
</Style>
```

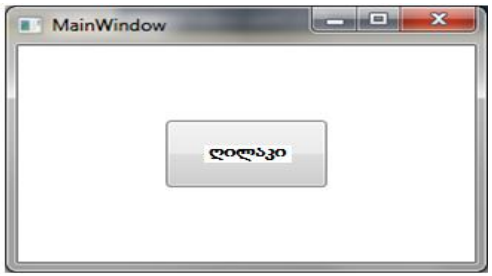
ტრიგერში მითითებული უნდა იყოს მაიდენტიფიცირებელი თვისება (ჩვენ მაგალითში IsMouseOver), რომელიც უნდა ვაკონტროლოთ, და მნიშვნელობა, რომელსაც უნდა ველოდოთ (მაგალითად, true). როცა გამოჩნდება აუცილებელი მნიშვნელობა, ყენდება თვისება, რომელიც განისაზღვრება Setter ობიექტით.

ჩვენ მაგალითში, როცა IsMouseOver = true, ანუ მაუსის კურსორი მიყვანილია ღილაკზე, მაშინ Background თვისებას მიენიჭება მნიშვნელობა Red, ანუ ღილაკის ფონი ხდება წითელი. როცა კურსორი გამოდის ღილაკის ზონიდან, მაშინ ტრიგერის ამუშავების პირობა ირღვევა და ღილაკის ფონი გადადის საწყის მდგომარეობაში.

თვისების ცვლილების მომლოდინე ტრიგერის გარდა არსებობს მოვლენათა ტრიგერები (EventTrigger), რომლებიც ელოდება განსაზღვრულ მოვლენათა აღმოცენებას.

7.11. შაბლონები

WPF-აპლიკაციების დაპროექტების დროს ფანჯრები ან გვერდები არის კონტეინერები, რომლებშიც განლაგებულია სხვა კონტეინერები და ინტერფეისის სხვადასხვა ელემენტები (ჭდეები, ტექსტური ბლოკები, ღილაკები, სიები და სხვა მართვის ელემენტები). მაგალითად, ფანჯარა ერთი ღილაკით (ნახ.6.5).

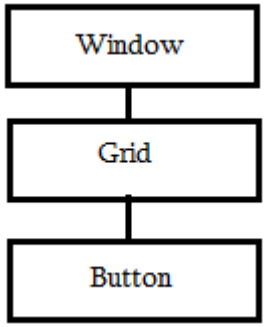


ნახ.7.5

XAML-დოკუმენტი შექმნილი ფანჯრისთვის შემდეგი სახისაა.

```
<Window x:Class="WpfApplication1.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/
                                    xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/
                                          2006/xaml"
     Title="MainWindow" Height="200" Width="300">
     <Grid>
        <Button Content="ღილაკი" Width="100"
             Height="50"/>
     </Grid>
</Window>
```

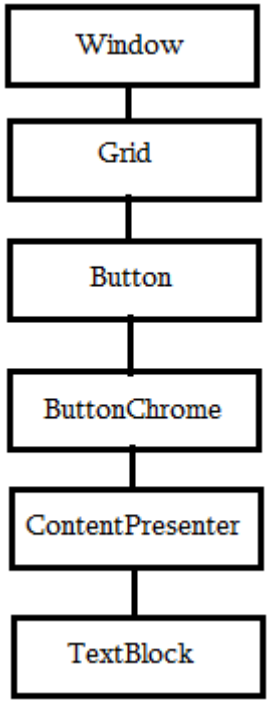
ფანჯრის XAML-დოკუმენტში მოცემულია სამი ობიექტი: Window, Grid და Button. ეს ელემენტები ქმნის ფანჯრის ლოგიკურ ხეს (ნახ.7.6).



ნახ.7.6. ფანჯრის ლოგიკური ხე

WPF-ში ლოგიკური ხე დეტალიზირდება ვიზუალირი ხის დახმარებით, რომელიც ასახავს ლოგიკური ხის ელემენტებს უფრო მცირე ფრაგმენტების სახით. მაგალითად, ღილაკი, ვიზუალური ხის დონეზე, ინკაპსულირებულია მართკუთხედის სახით,

რომელიც შეიცავს საზღვარს (ButtonChrome კლასი), შინაარსს (ContentPresenter კლასი) და ბლოკს ტექსტით (TextBlock კლასი) ნახ.6.7.



ნახ.7.7. ფანჯრის ვიზუალური ხე

საერთოდ, არსებობს არაერთი ხერხი ლოგიკური ხის გაფართოებისა ვიზუალურ ხემდე. მაგალითად, ელემენტი Button, რომელიც შიგთავსის ელემენტია, შეუძლია შეიცავდეს ნებისმიერ სხვა ელემენტს, რაც აისახება მის ვიზუალურ ხეში.

ვიზუალური ხე საშუალებას იძლევა შეიცვალოს მისი ელემენტები სტილების დახმარებით და შეიქმნას ახალი შაბლონები მართვის ელემენტებისათვის.

WPF-ში არსებობს მართვის ელემენტების შაბლონები, მონაცეთა შაბლონები და პანელის სპეციალური შაბლონები.

ControlTemplate - მართვის ელემენტების შაბლონი გამოიყენება ამ ელემენტების გამოსახვის (წარმოდგენის) და მათი ვიზუალური ქცევის მოსაცემად.

DataTemplate - მონაცემთა შაბლონი გამოიყენება მონაცემთა ამოსაღებად ობიექტიდან და მათი ასახვისთვის შიგთავსის მართვის ელემენტში.

Hierarchical DataTemplate - პანელების შაბლონები გამოიყენება ელემენთა კომპლექტების მართვისათვის სიის ტიპის მართვის ელემენტებში.

7.11.1. მართვის ელემენტთა შაბლონები

მართვის ყოველ ელემენტს აქვს Template-თვისება, რომელიც განსაზღვრავს შაბლონს მისი ვიზუალიზაციისთვის. თუ ეს თვისება ცხადად არაა მოცემული, მაშინ გამოიყენება მართვის ელემენტის სტანდარტული შაბლონი, რომელიც WPF-შია განსაზღვრული. მართვის ელემენტის მომხმარებლის შაბლონის შესაქმნელად აუცილებელია განისაზღვროს ControlTemplate ობიექტი.

```
<ControlTemplate x:Key="შაბლონის_გასაღები"
                 TargetType="ელემენტის_ტიპი">
    ...
</ControlTemplate>
```

ატრიბუტი x:Key განსაზღვრავს გასაღებს, რომლითაც მიმართავენ შაბლონს, ხოლო TargetType განსაზღვრავს ელემენტის ტიპს, რომლისთვისაც იქმნება შაბლონი.

ღილაკის უმარტივესი შაბლონის მაგალითი მოცემულია ქვემოთ:.

```
<ControlTemplate x:Key="ButtonTemplate"
                 TargetType="Button">
```

```
    <Border BorderBrush="Blue" BorderThickness="3"
                CornerRadius="25"
   Background="Azure" TextBlock.Foreground="Green">
        <ContentPresenter
            HorizontalAlignment="Center"
            VerticalAlignment="Center"/>
    </Border>
</ControlTemplate>
```

ContentPresenter ელემენტი განსაზღვრავს მართვის ელემენტის შიგთავსის ჩასმის ადგილს კონტეინერში (მაგალითად, კონტეინერი არის ჩარჩო – Border).

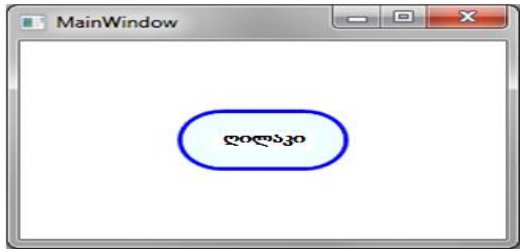
მართვის ელემენტის შაბლონი შეიძლება განთავსდეს ფანჯრის რესურსების კოლექციაში.

```
<Window.Resources>
        <ControlTemplate x:Key="ButtonTemplate"
                                TargetType="Button">
        ...
        </ControlTemplate>
</Window.Resources>
```

შაბლონის მიბმა მართვის ელემენტთან ხორციელდება ამ ელემენტის Template თვისების მიცემით.

```
<Grid>
   <Button Content="ღილაკი" Width="100" Height="50"
               Template="{StaticResource
                             ButtonTemplate}"/>
</Grid>
```

Template თვისება განისაზღვრება განლაგების (layout) გაფართოებით სტატიკურ რესურსზე (StaticResource) მიმართვისთვის რესურსის გასაღებით (ButtonTemplate). პროექტის ამუშავებით ღილაკს ექნება ასეთი სახე (ნახ.6.8).



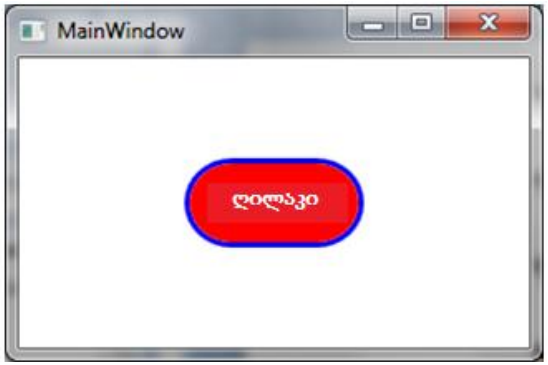
ნახ.7.8. ღილაკის წარმოდგენა შაბლონის გამოყენებით

განხილული შაბლონი განსაზღვრავს ღილაკის სტატიკურ ვიზუალურ წარმოდგენას. დინამიკური ვიზუალური ქცევის დასამატებლად შეიძლება შაბლონთა ტრიგერების გამოყენება. მაგალითად, დავამატოთ შაბლონში ტრიგერი, რომელიც შეცვლის ღილაკის ფონის შევსებას წითელი ფერით, როცა მასზე მაუსის კურსორი დადგება, ამასთანავე შეცვლის წარწერის ფერს თეთრით.

```
<ControlTemplate x:Key="ButtonTemplate"
                              TargetType="Button">
    <Border Name="Border" BorderBrush="Blue"
          BorderThickness="3"
          CornerRadius="25" Background="Azure"
        TextBlock.Foreground="Green">
        <ContentPresenter
           HorizontalAlignment="Center"
           VerticalAlignment="Center"/>
    </Border>

    <ControlTemplate.Triggers>
       <Trigger Property="IsMouseOver"
                                  Value="true">
       <Setter TargetName="Border"
             Property="Background" Value="Red" />
       <Setter TargetName="Border"
  Property="TextBlock.Foreground" Value="White" />
```

```
        </Trigger>
    </ControlTemplate.Triggers>
</ControlTemplate>
```

როცა ამოქმედდება მოვლენა MouseOver თვისება IsMouseOver გახდება true, შედეგად იმუშავებს ტრიგერი და შეიცვლება ღილაკის თვისებები Background და TextBlock.Foreground შესაბამისად Red და White-ით. Background და TextBlock.Foreground თვისებების დაყენებისას TargetName-ელემენტში მიეთითება Border სახელი, რომელიც მინიჭებული აქვს Border-ელემენტს შაბლონში. 7.9 ნახაზზე ნაჩვენებია ღილაკის წარმოდგენა, როცა მასზე მაუსის კურსორია მოთავსებული.



ნახ.7.9

ტრიგერების გამოყენებით შეიძლება განისაზღვროს მართვის ელემენტების ვიზუალური ქცევის რეაქცია აუცილებელ მოვლენებზე.

7.11.2. მონაცემთა შაბლონები

მონაცეთა შაბლონი - ესაა XAML-ფორმატირების ნაწილი, რომელიც განსაზღვრავს, თუ როგორ უნდა აისახოს მონაცემების მიმაგრებული ობიექტი. მონაცემთა შაბლონს შეიძლება ჰქონდეს ელემენტების ნებისმიერი კომბინაცია და უნდა შეიცავდეს ერთ ან მეტ მიბმის გამოსახულებას.

მონაცემთა შაბლონებს აქვს შემდეგი ელემენტები:

- შიგთავსის მართვის ელემენტები ContentTemplate თვისებით, რომელიც გამოიყენება ნებისმიერი Content-ში მოთავსებული შიგთავსის ვიზუალიზებისთვის;
- სიისებური მართვის ელემენტები ItemTemplate თვისებით, კოლექციის ელემენტების ვიზუალიზაციისთვის, რომელიც მითითებულია როგორც მონაცემთა წყარო.

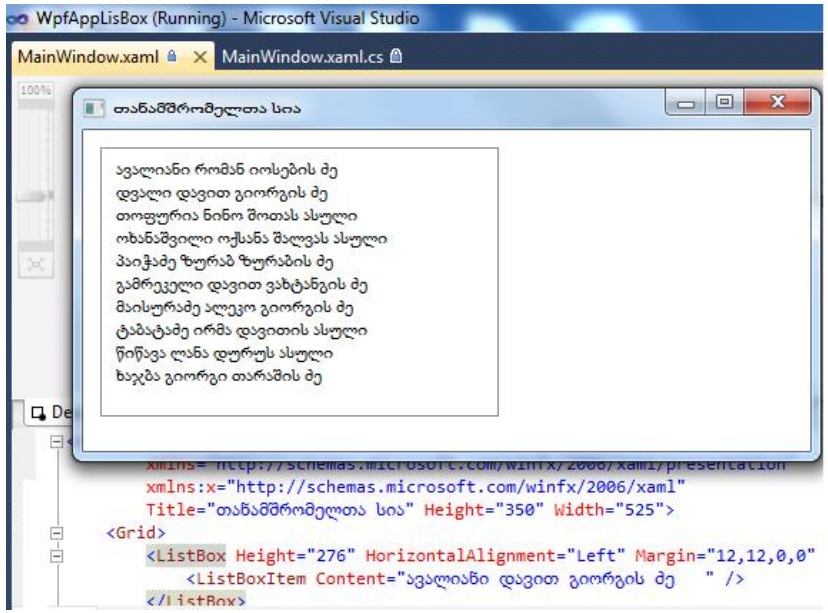
მაგალითად, დავამუშავოთ მონაცემთა შაბლონი ListBox სიისთვის. სიაში უნდა აისახოს მონაცემები თანამშრომლის შესახებ: გვარი, სახელი და მამის სახელი. მონაცემთა წყაროდ listBoxEmployees სიისთვის განიხილება კოლექცია Employees, რომელიც შეიცავს Employee კლასს. Employee კლასის თვისებებია:

- Surname –გვარი;
- Name - სახელი;
- Patronymic – მამის სახელი.

მონაცემთა შაბლონს listBoxEmployees სიისთვის აქვს შემდეგი სახე:

```
<ListBox Name="listBoxEmployees" >
    <ListBox.ItemTemplate>
        <DataTemplate>
          <StackPanel Orientation="Horizontal">
            <TextBlock Text="{Binding Path=Surname}" />
                <TextBlock Text="{Binding Path=Name}" />
                <TextBlock Text="{Binding
                                    Path=Patronymic}" />
          </StackPanel>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

პროგრამის ამუშავებით ფანჯარაში გამოიტანება სია (ნახ.7.10).



ნახ.7.10. სიის წარმოდგენა მომხმარებელური მონაცემთა შაბლონით

მონაცემთა შაბლონის მრავალჯერადი გამოყენების მიზნით ის უნდა მოიცეს ფანჯრის ან დანართის რესურსის სახით, შაბლონის გასაღების მითითებით (ListBoxEmployee).

```
<Application.Resources>
        <DataTemplate x:Key="ListBoxEmployee" >
        ...
        </DataTemplate>
</Application.Resources>
```


ამავე დროს სიის XAML-აღწერაში ItemTemplate თვისებისთვის მოიცემა ფორმატის გაფართოება სტატიკურ რესურსზე.

```
<ListBox Grid.Row="1" Name="listBoxEmployees"
        ItemTemplate="{StaticResource
ListBoxEmployee}" />
```

მონაცემთა შაბლონი არის ეფექტური ინსტრუმენტი მართვის ელემენტთა ვიზუალური ასახვის ცვლილებისთვის.

7.12. XAML ენის საფუძვლები

მომხმარებელთა ინტერფეისების აგება WPF- და Silverlight-დანართებისთვის (აპლიკაციებისთვის) ხორციელდება XAML (Extensible Application Markup Language - აპლიკაციების გაფართოებადი ფორმატირების ენა) ენის გამოყენებით. XAML-დოკუმენტი შეიცავს ფორმატს, რომელიც აღწერს დანართის ფანჯრის (ან გვერდის) გარეგან სახეს და ქცევას, ხოლო მასთან კავშირში მყოფი C# კოდის ფაილები კი - დანართის ლოგიკას. XAML-ენა უზრუნველყოფს დანართის დიზაინის პროცესის (გრაფიკული ნაწილი) გამოყოფას ბიზნეს-ლოგიკის (პროგრამული კოდი) დამუშავების პროცესისგან, დიზაინერებსა და დეველოპერებს შორის [69,70].

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

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

• XAML-დოკუმენტის ყოველი ელემენტი აისახება .NET კლასის რომელიღაც ეგზემპლარში. ასეთი ელემენტის სახელი ზუსტად შეესაბამება კლასის სახელს. მაგალითად, <Button> ელემენტი ემსახურება WPF-ინსტრუქციას Button-კლასის ობიექტის აგების მიზნით;

• XAML-ის ელემენტები შეიძლება ერთმანეთში ჩალაგდეს. ელემენტების ჩალაგების ფორმატი ასახავს ინტერფეისის ელემენტების ჩალაგებას;

• კლასის თვისებები განისაზღვრება ატრიბუტებით ან ჩალაგებული დესკრიპტორების დახმარებით სპეციალური სინტაკსით.

XAML-ენა ხასიათდება თვითაღწერადობით. XAML-დოკუმენტში ყოველი ელემენტი არის ტიპის სახელი (მაგალითად, Button, Window ან Page) მოცემული სახელსივრცის ჩარჩოებში.

ელემენტთა ატრიბუტები გამოიყენება შესაბამისი ობიექტების თვისებების (მაგალითად, Name, Height, Width და ა.შ.) და მოვლენების (Click, Load და ა.შ) მოსაცემად.

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

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


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

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

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

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

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

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

მეორე სახელსივრცე გამოიყენება XAML-ის სპეციფიური ლექსემების („საკვანძო სიტყვები“) ჩასასმელად. 7.1 ცხრილში მოცემულია შედარებით ხშირად გამოყენებადი ასეთი სიტყვები.

ცხრილი 7.1. XAML-ის საკვანძო სიტყვები

საკვანძო სიტყვა	დანიშნულება
x:Array	წარმოადგენს .NET-ის მასივის ტიპს XAML-ზე
x:ClassModifier	უზრუნველყოფს კლასის ტიპის ხილვადობის (internal ან public) განსაზღვრას, რომელიც Class საკვანძო სიტყვითაა აღნიშნული
x:FieldModifier	უზრუნველყოფს ტიპის წევრის ხილვადობის (internal, public, private ან protected) განსაზღვრას ფესვის ნებისმიერი სახელმინიჭებული ელემენტისთვის. სახელმინიჭებული ელემენტი განისაზღვრება საკვანძო სიტყვით Name

x:Key	უზრუნველყოფს გასაღების მნიშვნელობის დაყენებას XAML ელემენტეისთვის, რომელიც უნდა მოთავსდეს ლექსიკონის ელემენტში
x:Name	უზრუნველყოფს C#-ით გენერირებული სახელის მითითებას მოცემული XAML ელემენტისთვის
x:Null	წარმოადგენს null-მიმთითებელს
x:Static	უზრუნველყოფს ტიპის სტატიკურ წევრზე მიმართვას
x:Type	XAML-ეკვივალენტია C#-ია typeof ოპერაციის (იძახებს System.Type მითითებული სახელის საფუძველზე)
x:TypeArgument	უზრუნველყოფს ელემენტის დაყენებას, როგორც განზოგადებული ტიპისას განსაზღვრული პარამეტრებით

WPF-დანართებში საბაზო სახელსივრცეთა გარდა იყენებენ ასევე სპეციალურს, რომლებიც არააუცილებელია:

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

Window ობიექტის ატრიბუტებში შეიძლება დაემატოს შემდეგი XAML-აღწერები:

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

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

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

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

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

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

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

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

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

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

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

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

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

```
<Grid Name="grid">

</Grid>
```

მარტივი თვისებები XAML-დოკუმენტში მოიცემა შემდეგი სინტაკსის შესაბამისად:

თვისების_სახელი = "მნიშვნელობა"

მაგალითად, **Name = "grid1"**

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

მშობელი.თვისების_სახელი

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

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

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

ფორმატირების გაფართოებები იყენებს შემდეგ სინტაკსს:

{ფორმატირების_გაფართოების_კლასი არგუმენტი}

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

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


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

ფორმატირების გაფართოებები შეიძლება გამოყენებულ იქნას როგორც ჩალაგებული თვისებები.

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

განმსაზღვრელი_ტიპი.თვისების_სახელი

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

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

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

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

მოვლენის_სახელი = "მოვლენის_დამმუშავებლის_მეთოდეის_სახელი"

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

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

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

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

7.13. WPF აპლიკაციის შექმნა

კორპორაციული აპლიკაცია (დანართი) პროგრამაა, რომელიც რეალიზაციას უკეთებს განსაზღვრულ ბიზნესამოცანას (ბიზნესფუნქციას). დანართი უნდა მუშაობდეს მონაცემებთან, რომლებიც ინახება საინფორმაციო სისტემის მონაცემთა ბაზაში.

დანართის არქიტექტურა მოიცავს **წარმოდგენის შრეს, ბიზნესლოგიკის შრეს** და **მონაცემთა შრეს.** დანართის თითოეული შრის ფუნქციონალობა ბევრადაა დამოკიდებული საინფორმაციო სისტემის საგნობრივ სფეროზე, თუმცა არსებობს აგრეთვე ზოგადი, ფუძემდებლური ფუნქციები, რომლებიც ახასიათებს პრაქტიკულად ნებისმიერ კორპორაციულ დანართს.

ამგვარად, აპლიკაციაში უნდა დამუშავდეს წარმოდგენის შრე, რომელიც უზრუნველყოფს მომხმარებლის ინტერფეისს სისტემასთან. ინტერფეისი შეიძლება შეიქმნას Windows-ფანჯრებით და WPF გვერდებით, რომლებიც შეივსება მართვის სხვადსხვა ვიზუალური ელემენტებით [70].

მართვის ელემენტები უნდა უზრუნველყოფდეს სისტემის ფუნქციონალობის ვიზუალურ წარმოდგენას მომხმარებლისათვის, აწარმოებდეს შესატანი მონაცემების ვერიფიკაციას და ურთიერთქმედებდეს ბიზნესკლასებთან.

ბიზნესლოგიკის შრე უნდა უზრუნველყოფდეს დანართის ძირითად ფუნქციონალობას: დააფორმიროს ბიზნესკლასები, რეალიზება გაუკეთოს მონაცემთა დამუშავების ალგორითმებს, უზრუნველყოს მონაცემებთან მიერთება და მათი კეშირება. ამ შრის რეალიზაცია შეიძლება განხორციელდეს კლასების საფუძველზე, რომლებიც ბიზნესლოგიკას არეალიზებენ ინტერფეისული ელემენტების კლასთა მეთოდებით, ან მონაცემთა მოდელის კლასთა მეთოდებით.

მონაცემთა შრე უნდა უზრუნველყოფდეს დანართის ურთიერთქმედებას ბაზის მონაცემებთან. კორპორაციულ აპლიკაციებში ამისათვის ყველაზე მიზანშეწონილია გამოყენებულ იქნას პლატფორმა ADO.NET Entity Framework და მოდელი EDM (Entity Data Model). EDM მოდელი აღწერს მონაცემთა სტრუქტურას ფიზიკური შენახვის ფორმისგან დამოუკიდებლად.

კორპორაციული დანართების დაპროექტების საკითხების შესასწავლად განვიხილოთ ძირითადი მიდგომები ინფორმაციული სისტემის ცალკეული ფუნქციების ასაგებად, რომელიც უზრუნველყოფს კომპანიის თანამშრომელთა მონაცემების დამუშავებას.

მაგალითისთვის აქ გამოვიყენებთ მონაცემთა ბაზას TitlePresonal, ცხრილებისა და ველების მცირე რაოდენობით, ხოლო აპლიკაციის ფუნქციონალობა ითვალისწინებს კომპანიის თანამშრომელთა მონაცემების შეტანას, კორექტირებას და წაშლას. ასაგები დანართი უნდა უზრუნველყოფდეს შემდეგი მონაცემების

შენახვას და გადამუშავებას: გვარი, სახელი, სქესი, დაბადების_თარიღი, თანამდებობა, ტელეფონი, ელ_ფოსტა.

აპლიკაციის ფუნქციებია:

- თანამშრომელთა მონაცემების დათვალიერება;
- ახალი თანამშრომლის მონაცემთა შეტანა;
- თანამშრომლის მონაცემთა რედაქტირება;
- თანამშრომლის მონაცემების წაშლა;
- მონაცემთა მოძებნა თანამშრომლის შესახებ.

შევქმნათ .NET Framework 4 გარემოში ახალი დანართის WPF-პროექტი სახელით WpfApplProject.

7.14. WPF აპლიკაცია მონაცემთა ბაზებით

განიხილება „არსთა-დამოკიდებულების“ მოდელის ძირითადი დებულებები, მისი საბაზო კომპონენტები: არსები (ობიექტები), ასოციაციები (კავშირები) და თვისებები (ატრიბუტები). სასწავლო მაგალითზე ნაჩვენებია მოდელის აგების პროცესი არსებული ბაზის გამოყენებით. აგებული PageEmployee დანართის გვერდისა და მონაცემთა მოდელისთვის ხორციელდება მონაცემთა მიბმა ინტერფეისის ელემენტებთან: ტექსტბოქსის, კომბობოქსის, ლისტბოქსის და თარიღის. განიხილება დანართის ურთიერთქმედების ოპერაციების დაპროექტების საკითხები მონაცემთა ბაზასთან: მონაცეთა რედაქტირება, დამატება და წაშლა. აღიწერება მონაცემთა შემოწმების შესაძლებლობა მათი შეტანისას, მომხმარებლელთა შემოწმების წესების გამოყენებით.

Entity Data Model (EDM) - არსთა მონაცემთა მოდელი („არსთა-კავშირების“ მოდელი). **EDM** მოდელი წარმოადგენს

ძირითად ცნებათა ერთობლიობას, რომელიც აღწერს მონაცემთა სტრუქტურას მისი კომპიუტერის მეხსიერებაში ფიზიკურად შენახვის ფორმისგან დამოუკიდებლად. EDM მოდელში აღწერილ მონაცემებს შეიძლება ჰქონდეს განსხვავებული სტრუქტურები: რელაციური, ტექსტური ფაილების, XML-ის, ელექტრონული ცხრილების და რეპორტების. EDM მოდელი აღწერს მონაცემთა სტრუქტურას არსებისა და კავშირების საფუძველზე, რომლებიც დამოუკიდებელია შენახვის სქემებისგან. ასეთი მიდგომის საფუძველზე მონაცემთა ფიზიკური დამახსოვრება განცალკევებულია დანართისაგან და არ მოქმედებს მის დამუშავებაზე. ამის უზრუნველყოფა ხდება იმის გამო, რომ არსები და კავშირები აღწერს მონაცემთა სტრუქტურებს ისე, როგორც ეს სჭირდება დანართს. EDM მოდელი კონცეპტუალური მოდელია, რომელიც აღწერს მონაცემთა სტრუქტურებს არსების და კავშირების სახით.

EDM მოდელი იყენებს სამ ძირითად ცნებას მონაცემთა სტრუქტურის აღსაწერად:

- არსის ტიპი;
- ასოციაციის ტიპი;
- თვისება.

არსის ტიპი გამოიყენება მონაცემთა სტრუქტურის აღსაწერად EDM მოდელის დახმარებით. კონცეპტუალურ მოდელში არსთა ტიპები აიგება თვისებებისგან და აღწერს ზედა დონის ძირითად კონცეპტუალურ ელემენტთა სტრუქტურას, როგორიცაა მაგალითად, თანამშრომლები და როლები. არსის ტიპი არის შაბლონი არსებისთვის. არსი არის განსაზღვრული ობიექტი (მაგალითად, რომელიმე თანამშრომელი და მისი როლი ბიზნეს-პროცესში). ყოველ არსს უნდა ჰქონდეს უნიკალური გასაღები არსთა ერთობლიობის შიგნით. არსთა ერთობლიობა არის

განსაზღვრული ტიპის არსის ეგზემპლართა კოლექცია. არსთა ერთობლიობები (და ასოციაციათა ერთობლიობები) ლოგიკურად დაჯგუფებულია არსთა კონტეინერებში.

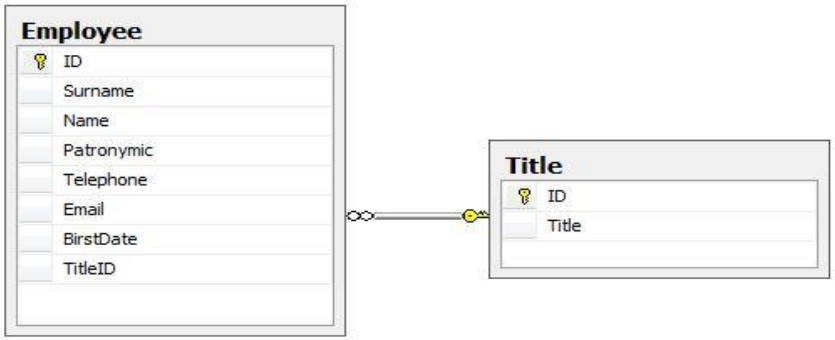
ასოციაციის ტიპი გამოიყენება კავშირთა აღწერის ასაგებად EDM მოდელში. კონცეპტუალურ მოდელში ასოციაცია ასახავს კავშირს ორი ტიპის არსებს შორის. ყოველ ასოციაციას აქვს ორი ბოლო წერტილი, რომლებიც განსაზღვრავს არსთა ტიპებს. ისინი მონაწილეობს ასოციაციაში. ყოველი ბოლო წერტილი განსაზღვრავს მის ჯერადობას, რაც მიუთითებს არსების შესაძლო რაოდენობაზე ასოციაციის ამ ბოლო წერტილში.

არსთა ტიპები შეიცავს თვისებებს, რომლებიც განსაზღვრავს მათ სტრუქტურას და მახასიათებლებს. მაგალითად, არსის ტიპს „თანამშრომელი“ შეიძლება ჰქონდეს თვისებები: **ID, გვარი, სახელი, სქესი, დაბ_თარიღი, თანამდებობა, ტელეფონი, ელ_მისამართი** და ა.შ.

თვისებები კონცეპტუალურ მოდელში ანალოგიურია პროგრამული აპლიკაციის კლასებისა ან მონაცემთა ბაზების ატრიბუტების. თვისებები შეიძლება შეიცავდეს პრიმიტიულ მონაცემებს (სტრიქონი, მთელი რიცხვი, თარიღი, ლოგიკური მნიშვნელობა) ან სტრუქტურირებულ მონაცემებს (რთული ტიპი).

კონცეპტუალური მოდელი არის მონაცემთა სტრუქტურის სპეციფიკური ასახვა არსებისა და კავშირების სახით. 7.11 ნახაზზე ნაჩვენებია კონცეპტუალური მოდელის გამოსახვა სქემით, ბაზისათვის TitlePersonal – „თანამშრომელი“, ორი ტიპის არსით

Employee – თანამშრომელი და Title –როლი/თანამდებობა), და ასოციაციური კავშირით 1:* (ერთი:მრავალთან).



ნახ.7.11. მონაცემთა ბაზის კონცეპტუალური მოდელი

Employee ცხრილი შეიცავს თანამშრომლის მონაცემებს. მისი ატრიბუტებია:

- ID – თანამშრომლის იდენტიფიკატორი;
- Surname – გვარი;
- Name – სახელი;
- Sex –სქესი;
- BirstDate – დაბადების თარიღი;
- Telephone – ტელეფონი;
- Email – ელ_მისამართი;
- TitleID – გარე გასაღები Title ცხრილთან დასაკავშირებლად.

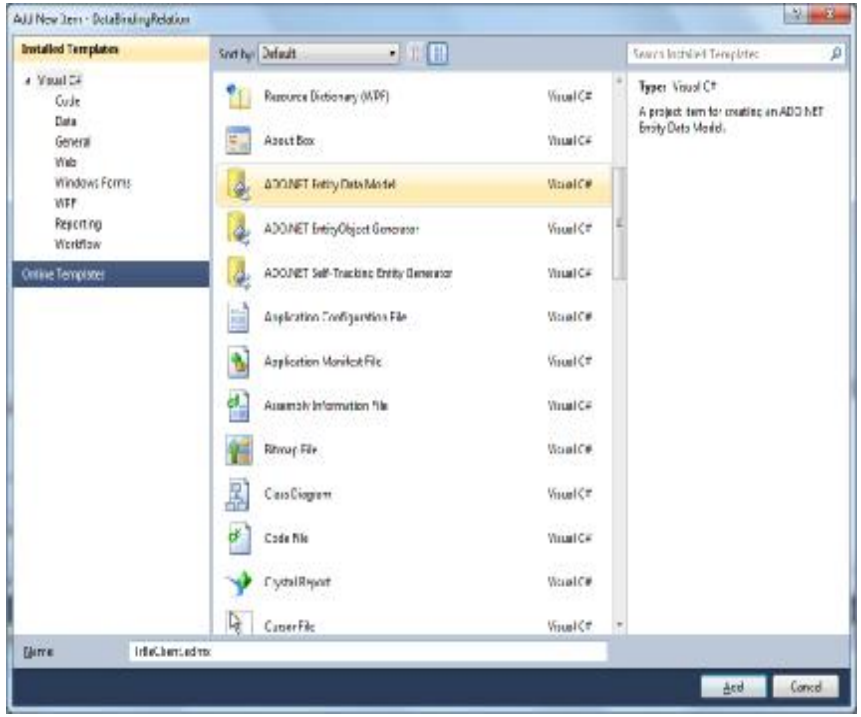
Title ცხრილი თანამდებობათა ცნობარია, რომელიც ამ ორგანიზაციას აქვს. იგი შეიცავს შემდეგ ატრიბუტებს:

- ID – თანამდებობის კოდი;
- Title – თანამდებობის დასახელება.

EDM-მოდელის შესაქმნელად პროექტში Solution Explorer-დან ჩავამატოთ ახალი ელემენტი (ნახ.7.12):

Add -> NewItem -> ADO.NET EDM

და File Name-ში მივცეთ სახელი: TitleClient.

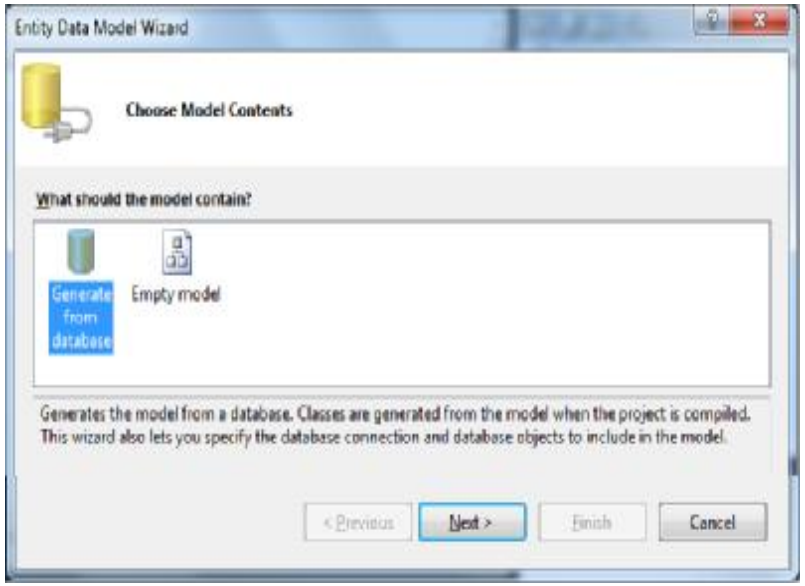


ნახ.7.12. პროექტში EDM მოდელის დამატება

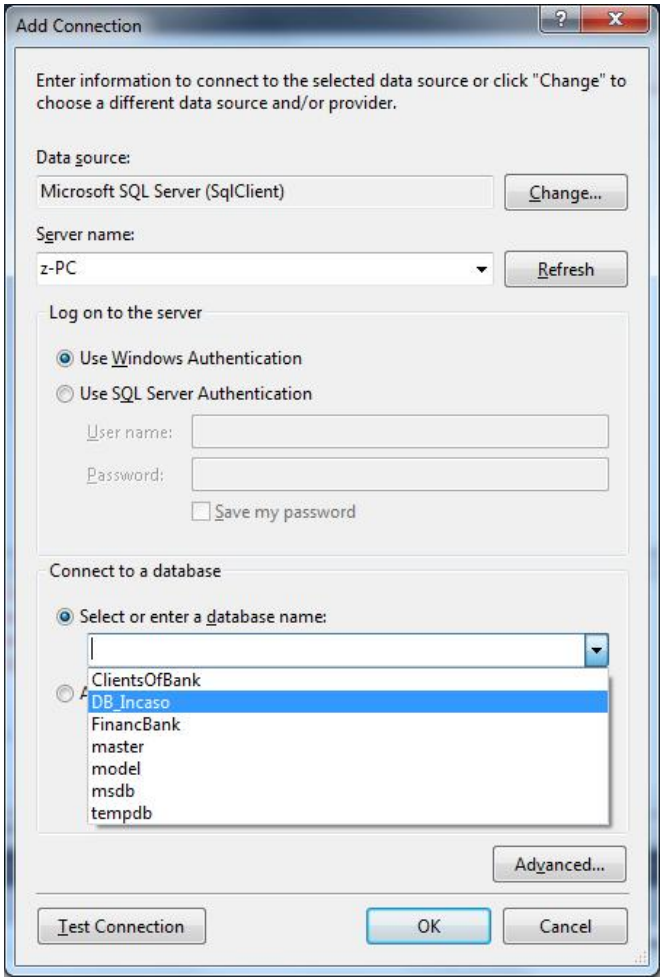
EDM მოდელის შექმნისას 7.13 ნახაზზე ნაჩვენებ Wizard-ში მივუთითოთ „შექმნა მონაცემთა ბაზიდან“, რის შემდეგაც გამოვა 7.14 ნახაზზე ნაჩვენები ფანჯარა. აქ, მონაცემთა ბაზასთან მისაერთებლად, უნდა მივცეთ:

სერვერის_სახელი.მონაცემთა_ბაზის_სახელი

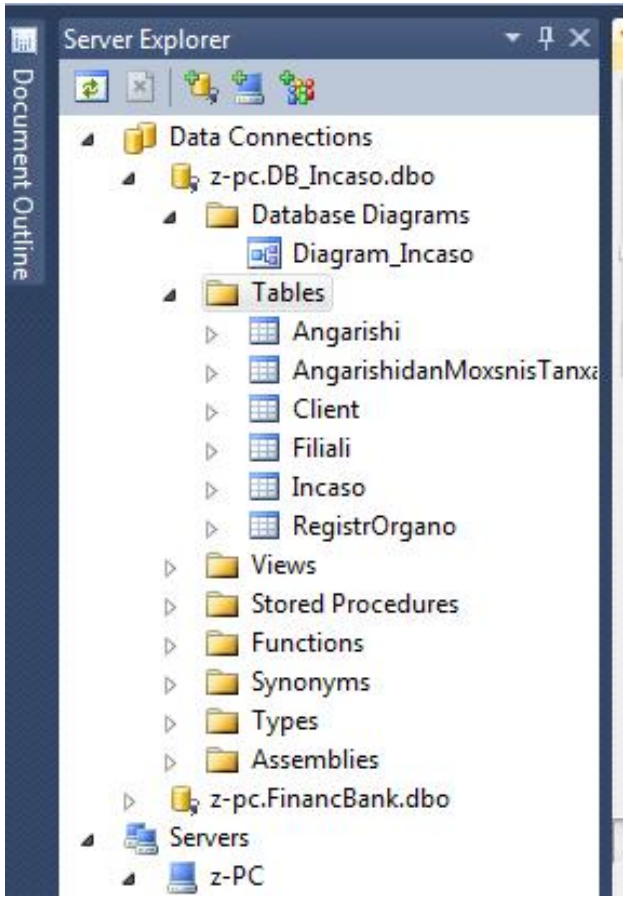
და მონაცემთა მოდელის სახელი - TitlePersonEntities (მიერთების პარამეტრების შენახვა Config-ში). Next ღილაკით გამოიტანება 7.15 ფანჯარა, რომლის ჩეკბოქსშიც ვირჩევთ მონაცემთა ბაზის ობიექტებს (მაგ.,Employee და Title), მივუთითებთ ობიექტების ფორმირებას მხოლობით ან მრავლობით რიცხვში.



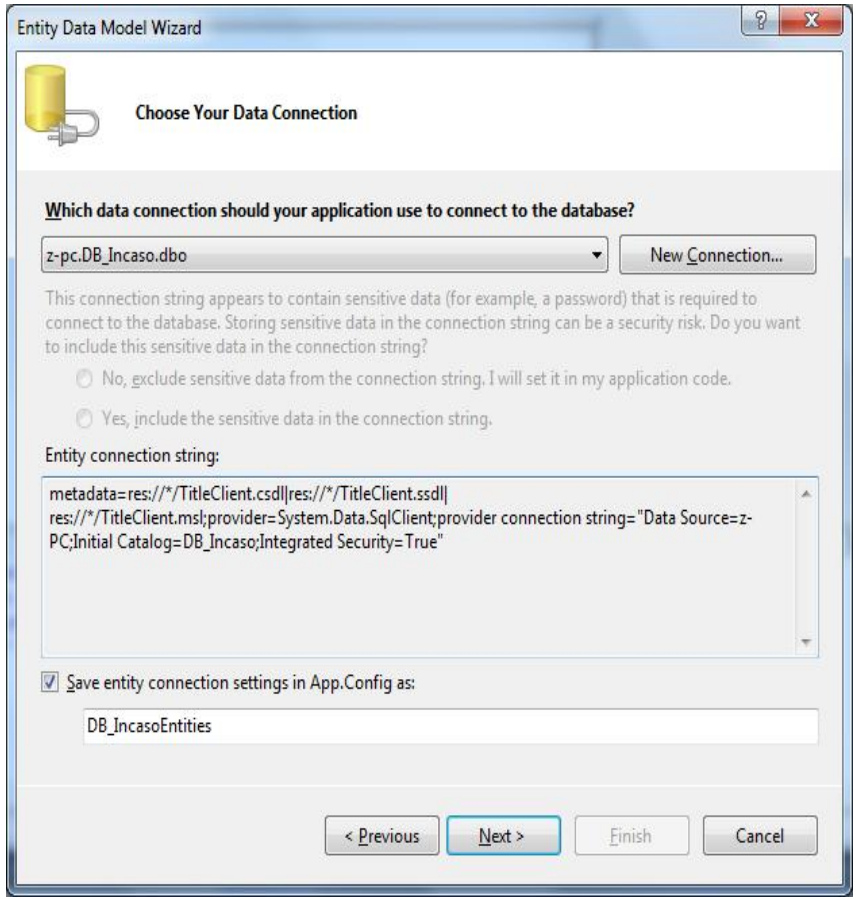
ნახ.7.13. EDM მოდელის შექმნა მონაცემთა ბაზიდან



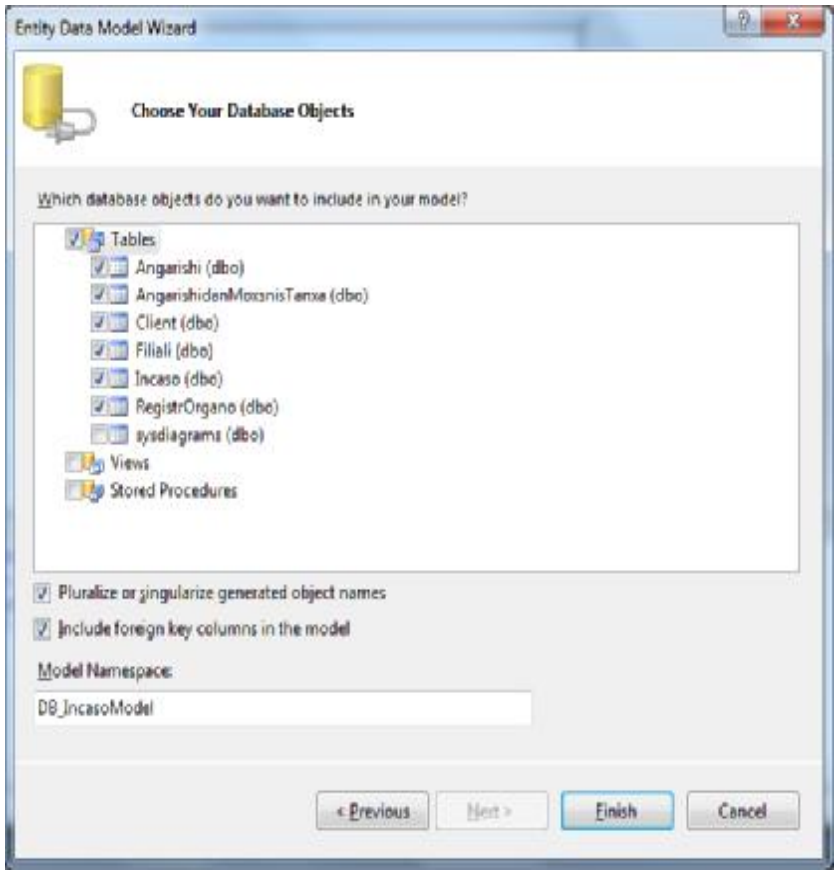
ნახ.7.14-ა. მონაცემთა ბაზასთა მიერთება: ეტაპი_1



ნახ.7.14-ბ. მონაცემთა ბაზასთა მიერთება: ეტაპი_ 2

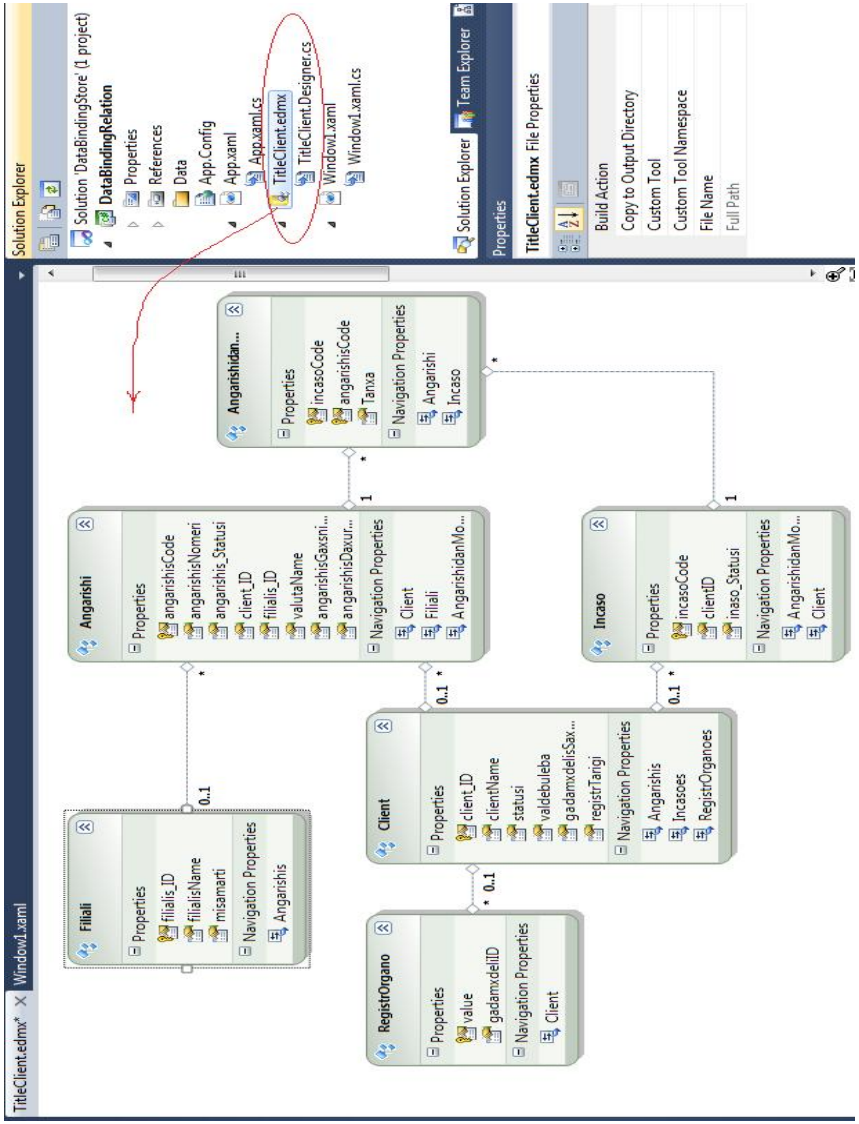


ნახ.7.14-გ. მონაცემთა ბაზასთა მიერთება: ეტაპი_3



ნახ. 7.15. მონაცემთა ბაზის ცხრილების არჩევა

EDM მოდელის შექმნის შედეგად პროექტში დამატებული იქნება TitleEmployee.edmx ფაილი (ნახ.7.16). მიმართებები საჭირო ბიბლიოთეკებზე და კონფიგურაციის ფაილი.



ნახ. 7.16. პროექტი EDM მოდელით

ავტომატურად გენერირებული კლასი `DB_IncasoEntities`, რომელიც მემკვიდრეა ObjectContext კლასის, ასახავს `TitleClient` მონაცემთა ბაზის არსებს, შეიცავს თვისებებს, რომლებიც ამოდელირებს Tanamshromeli და Tanamdeboba ცხრილებს, კავშირებს მათ შორის.

მონაცემთა მოდელის შექმნისას პროექტში ავტომატურად მოხდა კონფიგურაციის App.Config ფაილის შექმნა, რომელიც შეიცავს მონაცემთა ბაზასთან მიერთების სტრიქონს.

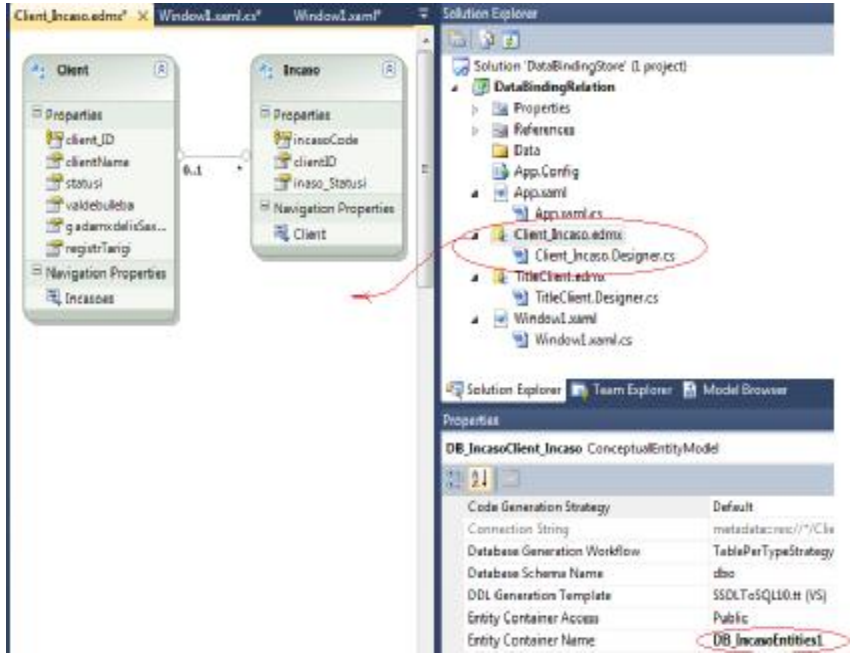
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="DB_IncasoEntities"
    connectionString="metadata=&amp;quot;res://*/
    TitleClient.csdl|&amp;#xD;&amp;#xA;
    res://*/TitleClient.ssdl|res://*/
    TitleClient.msl&amp;quot;
    provider=System.Data.SqlClient;provider connection
    string=&amp;quot;Data Source=z-PC;Initial
    Catalog=DB_Incaso;&amp;#xD;&amp;#xA;
    Integrated
    Security=True;MultipleActiveResultSets=True&amp;quot;"
    providerName="System.Data.EntityClient" />

    <add name="DB_IncasoEntities1"
     connectionString="metadata=res://*/
    Client_Incaso.csdl|res://*/Client_Incaso.ssdl|res://*/
    Client_Incaso.msl;provider=System.Data.SqlClient;
    provider connection string=&amp;quot;
    Data Source=z-PC;Initial Catalog=DB_Incaso;Integrated
    Security=True;MultipleActiveResultSets=True&amp;quot;"
    providerName="System.Data.EntityClient" />
  </connectionStrings>
</configuration>
```

Solution Explorer-ში Edm მოდელის C#-ის TitleClient.Designer.cs -კოდის ფრაგმენტი ასეთი სახისაა:

```
using System;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using System.Data.EntityClient;
using System.ComponentModel;
using System.Xml.Serialization;
using System.Runtime.Serialization;

[assembly: EdmSchemaAttribute()]
#region EDM Relationship Metadata

[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_Angarishi_Client",
    "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
    typeof(DataBindingRelation.Client), "Angarishi",

System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.Angarishi), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_Angarishi_Filiali",
    "Filiali",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
    typeof(DataBindingRelation.Filiali), "Angarishi",

System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.Angarishi), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_AngarishidanMoxsnisTanxa_Angarishi",
    "Angarishi",
System.Data.Metadata.Edm.RelationshipMultiplicity.One,
```



```
    typeof(DataBindingRelation.Angarishi),
"AngarishidanMoxsnisTanxa",
System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.AngarishidanMoxsnisTanxa),
true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_AngarishidanMoxsnisTanxa_Incaso",
    "Incaso",
System.Data.Metadata.Edm.RelationshipMultiplicity.One,
    typeof(DataBindingRelation.Incaso),
"AngarishidanMoxsnisTanxa",

System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.AngarishidanMoxsnisTanxa),
true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_Incaso_Client",
    "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
    typeof(DataBindingRelation.Client), "Incaso",

System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.Incaso), true)]
[assembly: EdmRelationshipAttribute("DB_IncasoModel",
"FK_RegistrOrgano_Client",
    "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
    typeof(DataBindingRelation.Client), "RegistrOrgano",

System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    typeof(DataBindingRelation.RegistrOrgano), true)]
#endregion
namespace DataBindingRelation
{
 ... }
```



ნახ.7.17. EDM მოდელი: Client_Incaso

```
// Client_Incaso.Designer.cs ----------------------
using System;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using System.Data.EntityClient;
using System.ComponentModel;
using System.Xml.Serialization;
using System.Runtime.Serialization;

[assembly: EdmSchemaAttribute()]
#region EDM Relationship Metadata

[assembly:
EdmRelationshipAttribute("DB_IncasoClient_Incaso",
"FK_Incaso_Client",
```



```
      "Client",
System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
         typeof(DataBindingRelation.Client),
      "Incaso",
System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
         typeof(DataBindingRelation.Incaso), true)]

#endregion

namespace DataBindingRelation
{
...
}
```

7.15. მართვის ელემენტების მიბმა მონაცემთა წყაროსთან

აპლიკაციის სამუშაოდ მონაცემთა ბაზასთან აუცილებელია Window1 კლასის კოდში გამოცხადდეს შექმნილი `EDM` მოდელის `DataEntitiesEmployee` მონაცემთა კონტექსტის სტატიკური თვისება. დანართის დაპროექტების ამ ეტაპზე ეს თვისება შეიძლება იყოს მხოლოდ ყველასთვის მისაწვდომი, ანუ - public. შემდგომში ამ თვისებას გამოიყენებს სხვა კლასები, ოღონდ `PageEmployee` კლასის ეგზემპლარის შექმნის გარეშე, ამიტომაც იგი უნდა იყოს სტატიკური.

```
public static TitlePresonalEntities
DataEntitiesEmployee { get; set; }
public static DB_IncasoEntities1 DataEntitiesClient {
get; set; }
```

ასევე აუცილებელია გამოცხადდეს ListEmployee კოლექცია დანართის სამუშაოდ Employee ობიექტთა კოლექციასთან.

```
public PageEmployee()
{
      InitializeComponent();
    DataEntitiesEmployee = new
TitlePersonalEntities();
    ListEmployee = new
ObservableCollection<Employee>();
}

public Window1()
{
   InitializeComponent();
      DataEntitiesClient = new DB_IncasoEntities1();
  ListClient = new ObservableCollection<Client>();
public static DB_IncasoEntities1 DataEntitiesClient {
get; set; }

}
```

დანართისთვის მონაცემთა ფორმირება ბაზიდან მოხდება PageEmployee გვერდის ჩატვირთვის დროს. ამისათვის XAML-დოკუმენტში Page დავამატოთ თვისება Loaded:

Loaded="Page_Loaded"

PageEmployee კლასის კოდში ჩავრთოთ დამმუშავებელი Page_Loaded.

```
private void Page_Loaded(object sender,
RoutedEventArgs e)
{
    ObjectQuery<Employee> employees =
DataEntitiesEmployee.Employees;
    var queryEmployee = from employee in employees
                        orderby employee.Surname
                        select employee;
     foreach (Employee emp in queryEmployee)
    {
```



```
        ListEmployee.Add(emp);
    }
    DataGridEmployee.ItemsSource = ListEmployee;
}

private void Page_Loaded(object sender,
RoutedEventArgs e)
{
    ObjectQuery<Client> clients =
DataEntitiesClient.Clients;
    var queryClient = from Client in clients
                      orderby client.clientName
                      select client;
     foreach (Client cln in queryClient)
    {
        ListClient.Add(cln);
    }
    DataGridClient.ItemsSource = ListClient;
}
```

clients ველს აქვს ObjectQuery<Client> ტიპი. ObjectQuery<T> კლასი არის მოხოვნა, რომელიც აბრუნებს ტიპიზირებულ არსთა კოლექციას ელემენტთა ნებისმიერი რაოდენობით. ავაგოთ მოთხოვნა Linq-ტექნოლოგიის საშუალებით:

```
var queryClient = from client in clients
            orderby client.clientName
            select client;
```

მოთხოვნის შედეგები მოვაწესრიგოთ „გვარებით“ (orderby client.clientName). შემდეგ ვაფორმირებთ ListClient კლიენტთა კოლექციას და მონაცემთა წყაროს DataGridClient ბადისთვის.

```
foreach (Client cln in queryClient)
    {
        ListClient.Add(cln);
    }
DataGridClient.ItemsSource = ListClient;
```

პროექტირების შედეგად დანართში ფორმირებულია კოლექცია მონაცემებით TitleClient ბაზის Client ცხრილიდან.

შემდგომი ამოცანაა DataGridClient ბადის აწყობა მონაცემთა კორექტული ასახვისათვის. თავიდან მოდიფიცირება გავუკეთოთ DataGrid-ის ზოგად აღწერას.

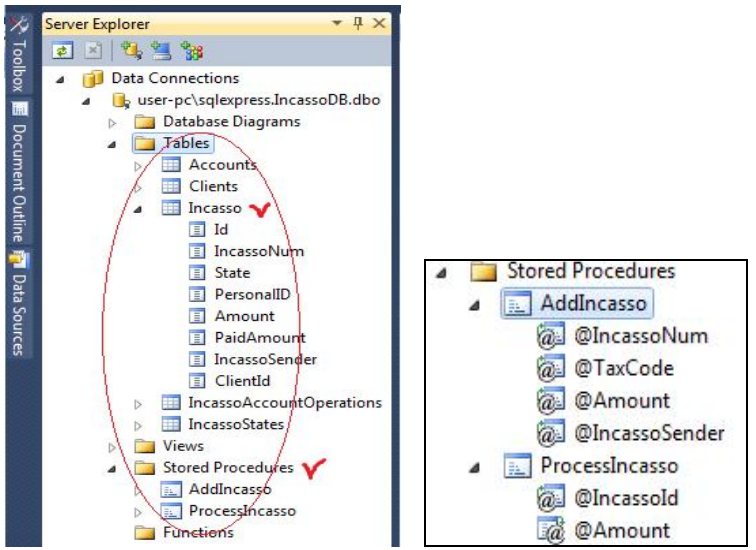
```
<DataGrid Name="DataGridClient"
ItemsSource="{Binding}"
AutoGenerateColumns="False"
HorizontalAlignment="Left"
MaxWidth="1000" MaxHeight="295"
RowBackground="#FFE6D3EF"
AlternatingRowBackground="#FC96CFD4"
BorderBrush="#FF1F33EB"
BorderThickness="3"
IsReadOnly="True"
RowHeight="25"
Cursor="Hand">
```

- განვსაზღვროთ მიბმა მონაცემთა წყაროსთვის:
  `ItemsSource="{Binding}"`
- გავაუქმოთ სვეტების ავტომატური გენერაცია:
  `AutoGenerateColumns="False"`
- დავადგინოთ მიბმა გვერდის მარცხენა საზღვართან:
  `HorizontalAlignment="Left"`
- განვსაზღვროთ ბადის მაქსიმალური ზომები:
  `MaxWidth="1000" MaxHeight="295"`
- დავადგინოთ ბადის შევსების ძირითადი და ალტერნატიული ფერები:
  `RowBackground="#FFE6D3EF"`
  `AlternatingRowBackground="#FC96CFD4"`
- დავადგინოთ ბადის ჩარჩოსთვის ფერი და ხაზის სისქე:
  `BorderBrush="#FF1F33EB"`
  `BorderThickness="3"`
- დავადგინოთ ბადის სტრიქონთა სიმაღლე:
  `RowHeight="25"`
- შევცვალოთ კურსორის ფორმა მაუსის ისრის მიტანისას DataGridEmployee ცხრილზე: Cursor="Hand"

7.16. WPF-აპლიკაციის რეალიზაცია IncasoDB ბაზით და სერვისორიენტირებული არქიტექტურით

პროგრამული აპლიკაციის დეველოპინგი ნაშრომის ექსპერიმენტული ნაწილისთვის შესრულდა MsSQL Server მონაცემთა ბაზის და MsVisual Studio.NET Framework 4.0 ინტეგრირებული პაკეტის გარემოში, WPF-ტექნოლოგიის, C# და XAML ენების საფუძველზე.

7.18 ნახაზზე ნაჩვენებია Incasso.DB მონაცემთა ბაზის სტრუქტურა Accounts, Clients, Incasso, IncassoAccountOperations და IncassoStates ცხრილებით და Stored Procedures სერვისებით.



ნახ.7.18–ა. IncassoDB ბაზა და შენახვადი პროცედურები

	Id	IncassoNum	State	PersonalID	Amount	PaidAmount	IncassoSender	ClientId
▶	39	89-66	0	01010056897	2500,0000	NULL	1	1
	40	89-66	0	01010056897	250,0000	NULL	1	1
	41	89-66	0	01010056897	300,0000	NULL	1	1
	42	856-63	1	NULL	10,0000	NULL	1	NULL
	43	856-63	1	NULL	50,0000	NULL	1	NULL
	44	856-62	1	NULL	50,0000	NULL	1	NULL
	45	106-62	1	NULL	50,0000	NULL	1	NULL
	46	5-99	0	05981001125	45,0000	NULL	1	2
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

(1)

	Id	IncassoId	AccountId	Amount
▶	26	39	2	500,0000
	27	40	1	250,0000
	28	41	1	50,0000
	29	46	4	20,0000
	30	46	5	25,0000
*	NULL	NULL	NULL	NULL

(2)

ნახ.7.18–ბ. Incasso (1) და IncassoAccountOperations (2) ცხრილების ფრაგმენტები

მონაცემთა ბაზის შენახვად პროცედურებში (Stored Procedures: AddIncasso, ProcessIncasso) რეალიზებულია სერვის-ფუნქციები. ქვემოთ, 1-ელ და მ–2 ლისტინგებში ნაჩვენებია ამ პროცედურათა ტექსტები:


```
-- ==ლისტინგი_1====== AddIncasso ===============
-- Author:           <Author,,Name>
-- Create date: <Create Date,,>
-- Description:      ახალი ინკასოს შემოსვლა
-- ============================================
ALTER PROCEDURE [dbo].[AddIncasso]
    @IncassoNum varchar(15),
    @TaxCode varchar(15),
    @Amount money,
    @IncassoSender tinyint
AS
BEGIN
-- SET NOCOUNT ON added to prevent extra result sets from
-- interfering with SELECT statements.
       SET NOCOUNT ON;

       DECLARE @State  tinyint = 0 -- გადაუხდელი
       DECLARE @ClientId int
       DECLARE @PaidAmount money
       Declare @incasoId int
       DECLARE @PersonalID varchar(12)
       --კლიენტის მოძებნა პირადი ნომერის მიხედვით
       SELECT @ClientId = Id, @PersonalID = PersonalID FROM
dbo.Clients where TaxCode  = @TaxCode

       IF ISNULL(@ClientId,0)=0
       SET  @State = 1--არ არის ბანკის კლიენტი/არ არის
                    --გადასახდელი
    -- Insert statements for procedure here
       INSERT INTO [dbo].[Incasso]
           ([IncassoNum]
           ,[State]
           ,[PersonalID]
           ,[Amount]
           ,[IncassoSender]
           ,[ClientId])
     VALUES
           (@IncassoNum
           ,@State
```

```
           ,@PersonalID
           ,@Amount
           ,@IncassoSender
           ,@ClientId)
       if @@ROWCOUNT = 1 RETURN SCOPE_IDENTITY()
       RETURN -1
END

ProcessIncasso
— == ლისტინგი_2 ======== ProcessIncasso ========
--declare @Amount money
--exec [dbo].[ProcessIncasso] 16, @Amount OUTPUT
--select @Amount
-- =============================================
-- Author:          <Author,,Name>
-- Create date: <Create Date,,>
-- Description:     <Description,,>
-- =============================================
ALTER PROCEDURE [dbo].[ProcessIncasso]
       @IncassoId int,
       @Amount money OUTPUT
AS
BEGIN
-- SET NOCOUNT ON added to prevent extra result sets from
-- interfering with SELECT statements.
       SET NOCOUNT ON;
       Declare @ClientId int
       Declare @IncassoAmount money
       Declare @AccountId int
       Declare @Balance money

       SELECT @ClientId = ClientId, @IncassoAmount = Amount
FROM Incasso where Id = @IncassoId and State = 0

       DECLARE @amount2Pay money = 0
       DECLARE @amount2Collect money = 0
       SET @amount2Collect  = @IncassoAmount
       BEGIN TRY
              DECLARE account_cursor CURSOR FOR
              SELECT AccountId, Balance
```



```
              FROM dbo.Accounts
              WHERE ClientId = @ClientId;

              OPEN account_cursor

              FETCH NEXT FROM account_cursor
              INTO @AccountId, @Balance

              WHILE @@FETCH_STATUS = 0
              BEGIN
                     --select @AccountId, @Balance
                     SET @amount2Pay = 0
              if @Balance > 0 and @amount2Collect > 0
                 BEGIN
                   if @Balance > @amount2Collect
                     BEGIN
                       SET @amount2Pay = @amount2Collect
                             SET @amount2Collect = 0
                     END
                    else
                     BEGIN
                       SET @amount2Pay = @Balance
SET @amount2Collect = @amount2Collect - @Balance
                     END

  ================================================
  -- Update Account Balance
  ================================================
  --SELECT AccountId, Balance, Balance - @amount2Pay
  --FROM dbo.Accounts
  --WHERE AccountId = @AccountId
       UPDATE dbo.Accounts
       SET Balance = Balance - @amount2Pay
       WHERE AccountId = @AccountId
  ================================================
  -- Update Account Balance
  ================================================
  -- Insert Incasso Operation
  ================================================
```

```
INSERT INTO dbo.IncassoAccountOperations (IncassoId,
                                        AccountId, Amount)
VALUES (@IncassoId, @AccountId, @amount2Pay)
       ================================================
       -- Insert Incasso Operation
       ================================================
       END

--SELECT @Balance Balance, @amount2Collect Amount2Collect,
@amount2Pay amount2Pay
       if @amount2Collect  = 0 BREAK
              FETCH NEXT FROM account_cursor
              INTO @AccountId, @Balance
       END

              CLOSE account_cursor;
              DEALLOCATE account_cursor;
       END TRY
       BEGIN CATCH
              RETURN -1
       END CATCH
       SELECT @Amount = SUM(Amount)
       FROM dbo.IncassoAccountOperations
       WHERE  IncassoId = @IncassoId
       RETURN @Amount
END
```

7.19 ნახაზებზე მოცემულია Incasso.DB ბაზის დანარჩენი ცხრილების ფრაგმენტები ჩანაწერებით.

IncassoStates: Quer...lexpress.IncassoDB)

	StateId	Name
▶	0	გადასახდელი
	1	არ არის გადასახ...
	2	ნაწილობრივ გად...
	3	გადახდილი
*	NULL	NULL

ნახ.7.19–ა. ინკასოს მდგომარეობათა ცხრილი (IncassoStates)

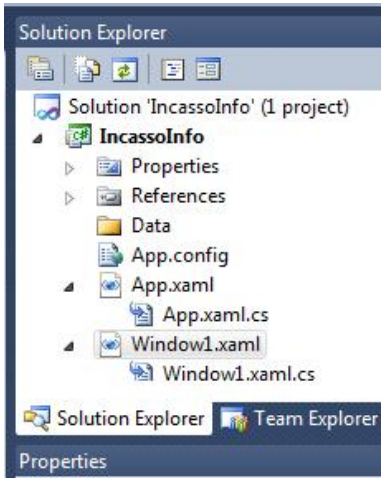
Accounts: Query(us...lexpress.IncassoDB) | dbo.Accounts: Tabl

	AccountId	AccountNum	ClientId	Balance
▶	1	108897301	1	0,0000
	2	108897305	1	0,0000
	3	108891459	1	0,0000
	4	2005689458	2	0,0000
	5	2005685812	2	335,0000
	6	2005680001	2	890,0000
	7	2005680002	2	1380,0000
	8	6098745201	3	5428,0000
	9	6098745202	3	125,0000
	10	20007801	4	350,0000
	11	20007802	4	2560,0000
	12	39770100	5	16740,0000
	13	908700010	6	25,0000
	14	918700050	7	630,0000
*	NULL	NULL	NULL	NULL

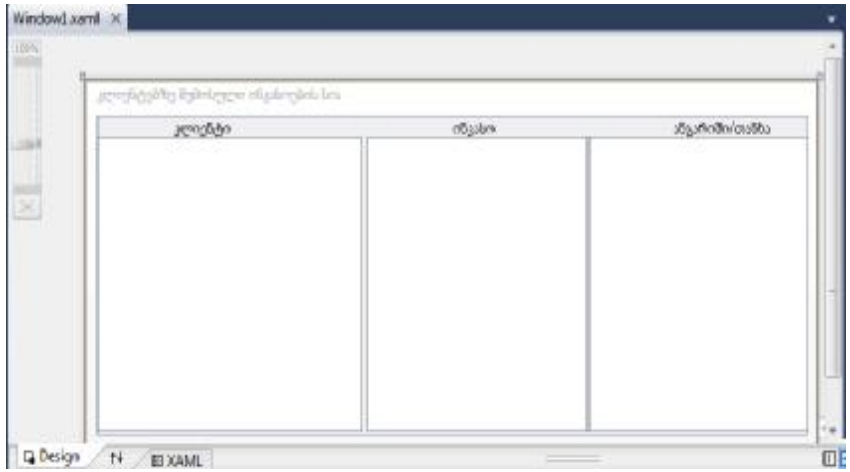
ნახ.7.19–ბ. ანგარიშების ცხრილი (Accounts)

ნახ.7.19–გ. კლიენტების ცხრილი (Clients)

7.20 ნახაზზე წარმოდგენილია VisualStudio.NET გარემოში აგებული IncassoInfo-აპლიკაცის სტრუქტურა (ა) და სისტემის მომხმარებლის ერთ-ერთი ინტერფეისის ფრაგმენტი (ბ).

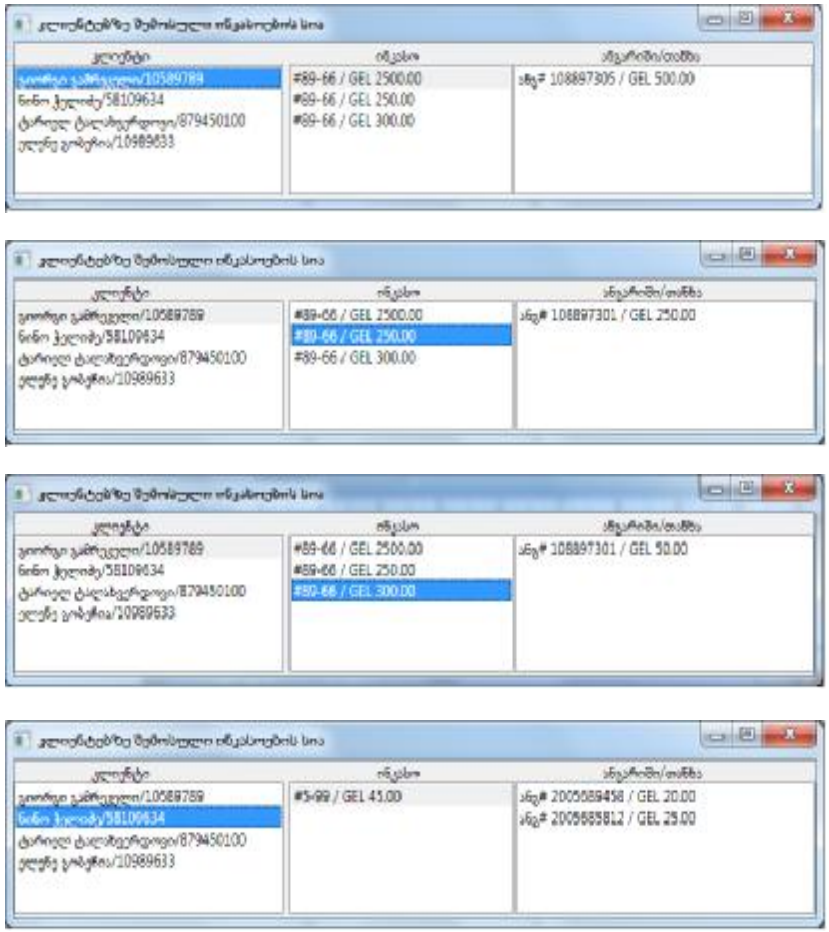


ნახ.7.20–ა



ნახ.7.20–ბ

7.21 ნახაზზე ილუსტრირებულია აპლიკაციის ამუშავების შემდეგ მიღებული შედეგები ბანკში კლიენტების მიხედვით ინკასოს მდგომარეობა.



ნახ.7.21. ინტერფეისის მუშაობის ფრაგმენტი:
„კლიენტი–>ინკასო–>ანგარიში“

პროგრამული აპლიკაციის ინტერფეისი ჰიბრიდული WPF-ტექნოლოგიის საფუძველზე რეალიზებულია XAML-ენის გამოყენებით (ტექსტი მოცემულია მე-3 ლისტინგში).

```
<-- ლისტინგი-3:   Window1.xaml  ფაილი ინტერფეისის ასაგებად ------- -->
<Window x:Class="DataBindingRelation.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/
                          xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Height="406" Width="726"
   Title="კლიენტებზე შემოსული ინკასოების სია"
   MinHeight="300"
   MinWidth="300"
   WindowStartupLocation="CenterScreen"
   xmlns:local="clr-namespace:DataBindingRelation"
   >
 <Window.Resources>
    <SolidColorBrush x:Key="ControlColorBrush"
        Color="{x:Static SystemColors.ControlColor}" />
 </Window.Resources>
 <Grid Background="{StaticResource ResourceKey = ControlColorBrush}">
          <Grid Name="ClientIncasso">
             <Grid.RowDefinitions>
                <RowDefinition />
                <RowDefinition Height="Auto" />
             </Grid.RowDefinitions>
             <Grid>
                <Grid.RowDefinitions>
                   <RowDefinition Height="Auto" />
                   <RowDefinition />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                   <ColumnDefinition Width="238" />
                   <ColumnDefinition Width="204" />
                   <ColumnDefinition Width="252*" />
                </Grid.ColumnDefinitions>
```



```
             <TextBlock HorizontalAlignment ="Center" Margin="69,0,87,0"
                Width="82">კლიენტი</TextBlock>
             <ListBox Grid.Row="1" Margin="0,0,0,3"
                ScrollViewer.VerticalScrollBarVisibility="Auto"
                ItemsSource="{Binding}"
                DisplayMemberPath="Name"
                IsSynchronizedWithCurrentItem="True"
               DataContext="{Binding}" Width="238" />

             <TextBlock Grid.Column="1" HorizontalAlignment="Center"
                   Margin="82,0,76,0" Width="46">ინკასო</TextBlock>
             <ListBox Grid.Row="1" Margin="0,0,1,3"
                ScrollViewer.VerticalScrollBarVisibility="Auto"
                ItemsSource="{Binding Path=ClientIncasso}"
                DisplayMemberPath="IncassoAmount"
                IsSynchronizedWithCurrentItem="True"
                DataContext="{Binding}" Width="198"
                HorizontalAlignment="Right" Grid.Column="1" />

            <TextBlock Grid.Column="2" HorizontalAlignment="Center"
                Margin="76,0,87,0" Width="99">ანგარიში/თანხა</TextBlock>
             <ListBox Grid.Row="1" Grid.Column="2" Margin="0,0,0,3"
                ScrollViewer.VerticalScrollBarVisibility="Auto"
                ItemsSource="{Binding Path=ClientIncasso/Incasso_Incasso_
                                                                  Details}"
                DisplayMemberPath="AccAmount"   />
          </Grid>
       </Grid>
    </Grid>
</Window>
```

აპლიკაციის ლოგიკური ნაწილი რეალიზებულია C#-პროგრამის ტექსტში, რომელიც მე-4 ლისტინგშია ასახული.

```
//— ლისტინგი_4: —— Window1.xaml.cs ————————————
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
```

```
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Data;
using System.Data.SqlClient;

namespace DataBindingRelation
{
  public partial class Window1 : Window
   {
    public Window1()
     {
      InitializeComponent();
      FillClientIncassoData();
     }

    void FillClientIncassoData()
     {
      // App.config ფაილიდან ამოიღება ბაზასთან მიბმის სტრიქონი
       String connectionString = System.Configuration.
       ConfigurationManager.ConnectionStrings["Incasso"].
                                              ConnectionString;
      // გამოყოფილ მონაცემთა შესანახი ადგილის შექმნა -----
       DataSet dataSet = new DataSet();
     // გამოყოფილი ადგილის შევსება მონაცემთა ბაზიდან -----
using (SqlConnection conn = new SqlConnection
                                        (connectionString))
    {
      SqlCommand selectCommand = conn.CreateCommand();
      SqlDataAdapter adapter = new SqlDataAdapter
                                         (selectCommand);

       // მონაცემები ჩაიტვირთება Clients ცხრილიდან -----
    selectCommand.CommandText = "SELECT FirstName +  ' ' +
        LastName + '/' + TaxCode as [Name],  * FROM Clients";
    adapter.Fill(dataSet, "Clients");
```



```
       // მონაცემები ჩაიტვირთება Incasso ცხრილიდან -----
    selectCommand.CommandText = "SELECT '#' + IncassoNum +
                     ' / GEL '+ Cast(Amount as varchar(15))
                   as IncassoAmount, * FROM Incasso";
    adapter.Fill(dataSet, "Incasso");

       // მონაცემთა ასახვა sql-მოთხოვნით -----
selectCommand.CommandText = @"select  N'ანგ# '+ A.AccountNum
+ ' / GEL '  + Cast(IO.Amount as nvarchar(15)) AccAmount,
IO.IncassoId FROM IncassoAccountOperations IO left Join
Accounts A on A.AccountId = IO.AccountId";

     adapter.Fill(dataSet, "Incasso_Details");
    }

 // ClientIncasso დამოკიდებულების შექმნა Clients და Incasso
// ცხრილებს შორის ----
dataSet.Relations.Add("ClientIncasso",
                   dataSet.Tables["Clients"].Columns["Id"],

dataSet.Tables["Incasso"].Columns["ClientId"]);

// დამოკიდებულების შექმნა Incasso და Incasso_Details ცხრილებს
// შორის განსხვავებული ხერხით ----
   DataColumn parentColumn =
              dataSet.Tables["Incasso"].Columns["Id"];
  DataColumn childColumn = dataSet.Tables["Incasso_Details"].
                                          Columns["IncassoId"];
  DataRelation relation = new DataRelation(
                "Incasso_Incasso_Details",
                parentColumn, childColumn);
            dataSet.Relations.Add(relation);

   // Client–ის მონაცემთა მიმაგრება ინტერფეისთან – – – –
     ClientIncasso.DataContext = dataSet.Tables["Clients"];
    }
  }
}
```

აპლიკაციის კონფიგურაციის ფაილის ტექსტი ნაჩვენებია მე-5 ლისტინგში:

<— ლისტინგი_5: ---- App.config ფაილი ---------- -->

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="Incasso" connectionString="Data
                    Source=USER-PC\SQLEXPRESS;Initial
                Catalog=IncassoDB;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

და ბოლოს, აპლიკაციის App.xaml ფაილის ტექსტი მოცემულია მე–6 ლისტინგში:

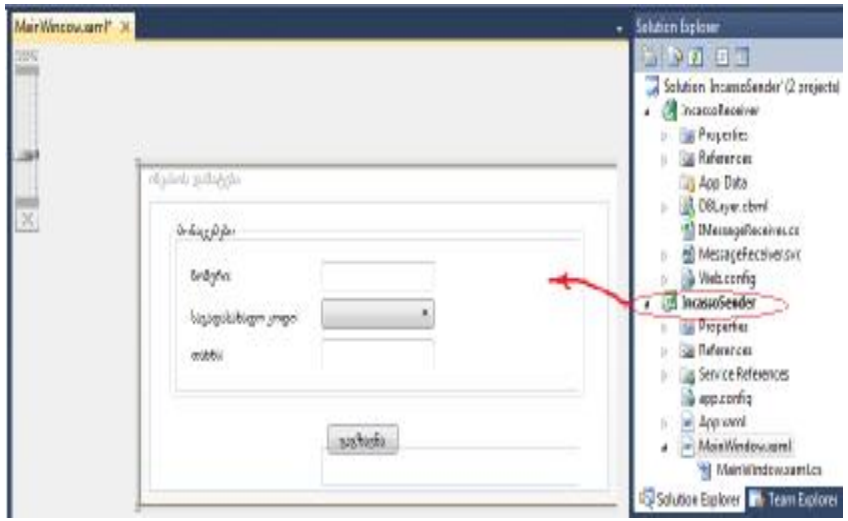
<— ლისტინგი_6: ---- App.xaml ფაილი ---- ---- >

```
<Application x:Class="DataBindingRelation.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="Window1.xaml">
    <Application.Resources>

    </Application.Resources>
</Application>
```

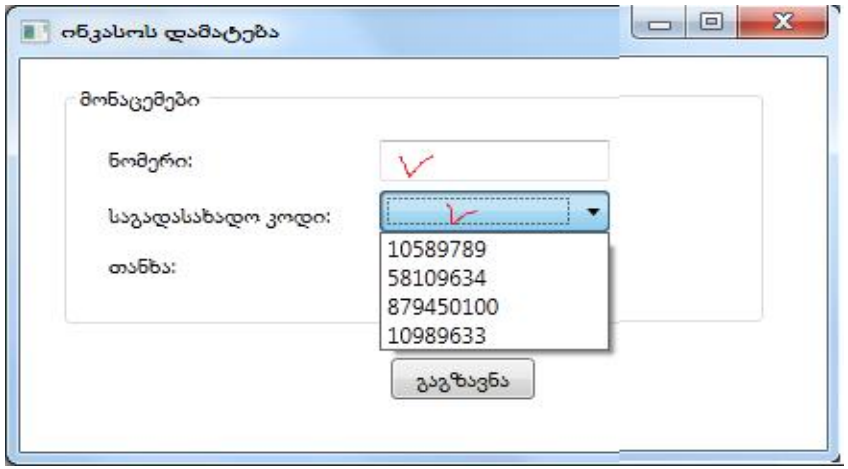
7.22 ნახაზზე ნაჩვენებია კლიენტისათვის ახალი ინკასოს დამატების სერვისით შესასრულებელი ინტერფეისის ფორმა. IncassoSender პროექტში MainWindow.xaml ფაილი ასახავს ფორმის დიზაინს.

ინკასოს ნომრის და თანხის მოცულობის შეტანის და საგადასახადო კოდის არჩევის შემდეგ (ნახ.7.23–ა,ბ) „გაგზავნა“ ღილაკით ინფორმაცია გადაეცემა ბანკს.

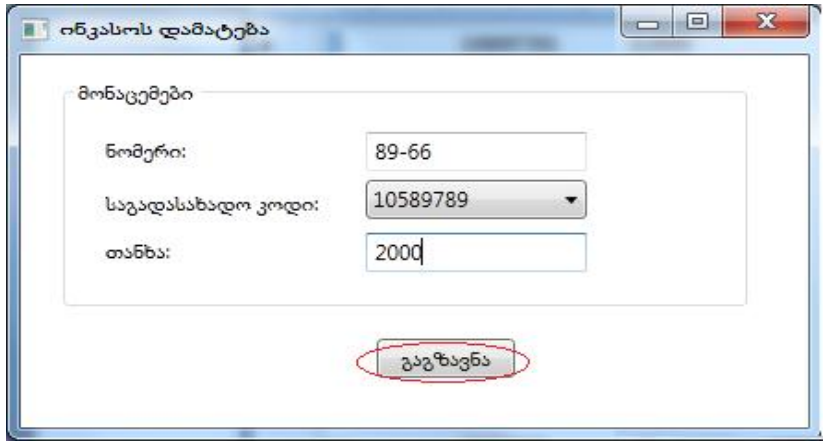
თუ შესაბამისი კლიენტის ანგარიშზე არაა საკმარისი თანხა, მაშინ ბრუნდება შეტყობინება: „ინკასოს გადახდე ვერ შესრულდა“ (ნახ.7.25).



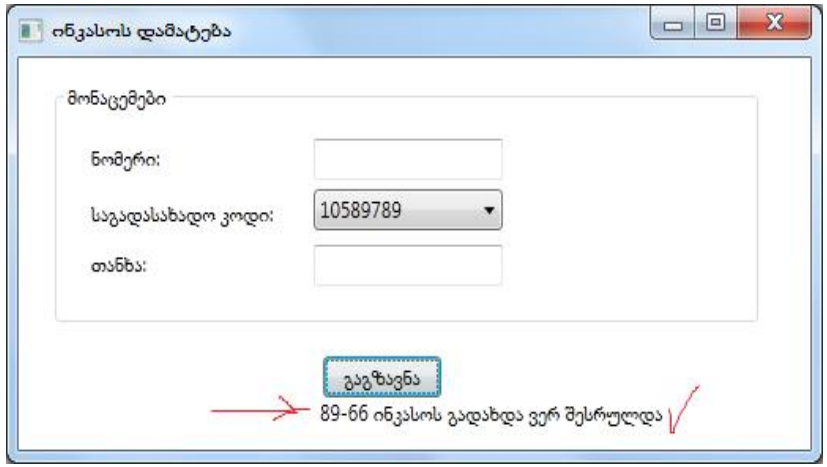
ნახ.7.22. ინტერფეისი



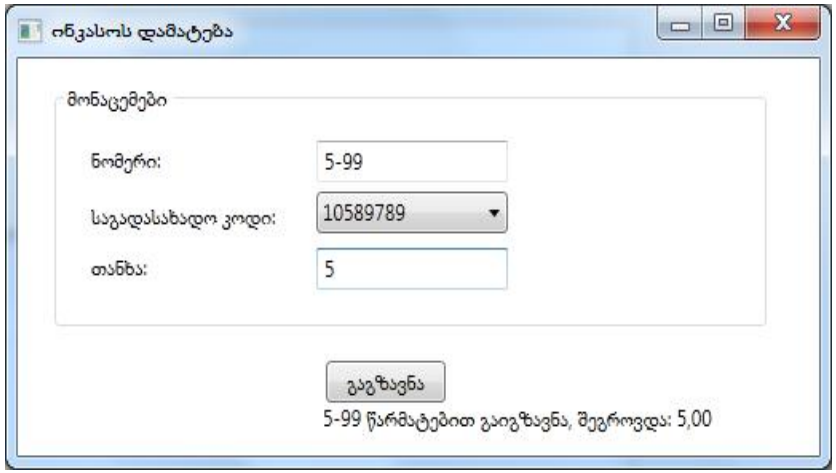
ნახ.7.23–ა. მონაცემების არჩევა



ნახ.7.23–ბ. ინკასოს შეტყობინების გაგზავნა



ნახ.7.24–ა. ინკასო გადახდა ვერ შესრულდა



ნახ.7.24–ბ. ინკასო გადახდა წარმატებით დასრულდა

დასასრულს შეიძლება აღვნიშნოთ, რომ ობიექტ-ორიენტირებული აპლიკაციების Web-სერვისებით ფორმირება განსაკუთრებით მოქნილი, მოსახერხებელი და გაცილებით საიმედოა, როგორც ახალი პროგრამული პროდუქტების შექმნისას, ისე არსებულ სისტემაში ცვლილებების გატარებისას.

Web-სერვისი წარმოადგენს დამაკავშირებელ რგოლს ობიექტ-ორიენტირებულ და პროცეს-ორიენტირებულ ტექნოლოგიებს შორის, რაც კომპანიათაშორისი და კომპანიის მსხვილ სტრუქტურათაშორისი საქმიანი პროცესების ინტეგრაციასა და მრავალაპლიკაციურ მართვას უზრუნველყოფს.

7.17. მარშრუტიზირებადი მოვლენები

WPF-აპლიკაციები იერერქიული ბუნებისაა. მათ აქვს მართვის ელემენტები, რომლებიც თვითონ შეიცავს სხვა მართვის ელემენტებს, რომელთაც ასევე აქვს სხვა მართვის ელემენტები და ა.შ. [68,70].

მარშრუტიზირებადი მოვლენა არის ისეთი მექანიზმი, რომელიც საშუალებას იძლევა, რომ მოვლენა, რომელსაც აქვს გავლენა იერარქიის ერთ ელემენტზე, ჰქონდეს ასევე გავლენა ამავე იერარქიის სხვა ელემენტებზე, და ამასთანავე არ იყოს საჭირო რთული კოდის დაწერა.

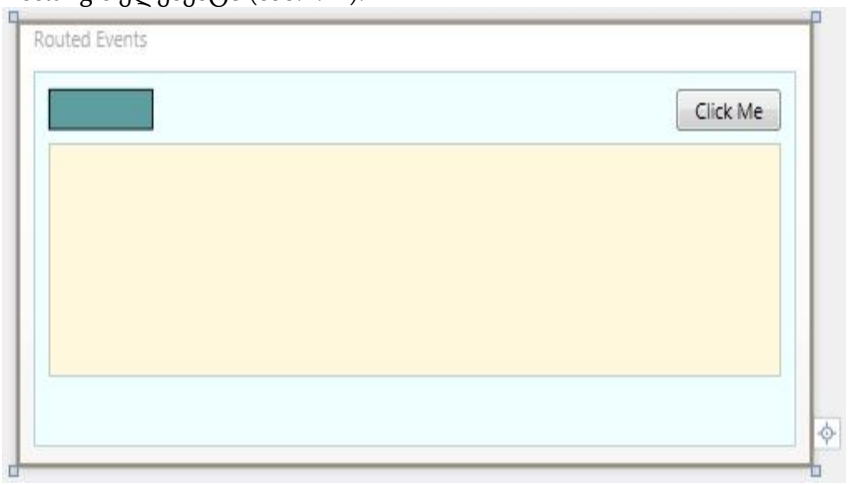
ამის საუკეთესო მაგალითია სიტუაცია, როდესაც მომხმარებლებს ეძლევათ საშუალება დანართში იმუშაონ მაუსის დახმარებით, რაც ძალზე ხშირია. როდესაც მომხმარებელი კლიკავს ღილაკს დანართში, ჩვეულებისამებრ საჭიროა, რომ დანართმა როგორღაც იმოქმედოს ამ დაკლიკვის მოვლენაზე. ერთ-ერთი ვარიანტი, რომელიც ცნობილია Windows Form და ASP.NET დანართებიდან, არის ამ ღილაკისთვის მოვლენის დამმუშავებლის კოდის დაწერა (Button_Click(…) {..}), რომელშიც მითითებულია ის ქმედებები, რომლებიც უნდა შესრულდეს ამ ღილაკის დაკლიკვის საპასუხოდ.

ასეთი მიდგომა ერთგვარად ზღუდავს დამმუშავებლის შესაძლებლობებს და ხშირად Windows Form-ში ქმნის საკმაოდ რთულ, გაურკვეველ კოდს. მარტივი მაგალითისთვის დავუშვათ, რომ ფორმაზე ღილაკია. ასეთ დროს რომელმა მართვის ელემენტმა უნდა იმოქმედოს დაკლიკვაზე და შექმნას მოვლენა - ფანჯარამ თუ ღილაკმა, თუ ორივემ ერთად ? თუ არსებობს მოთხოვნა, რომ მოვლენა შექმნას ორივემ და ამასთანავე გარკვეული თანამიმდევრობით, მაშინ Windows Form დანართში უნდა დაიწეროს სპეციალური, საკმაოდ რთული კოდი.

WPF-ში მაუსის ღილაკის დაკლიკვა მართვის ელემენტებისთვის (მათ შორის Button და Window) რეალიზდება მარშრუტიზირებადი მოვლენის სახით, რაც სრულად გამორიცხავს

ზემოხსენებულ პრობლემას. მარშრუტიზირებადი მოვლენები გენერირდება ყველა ობიექტის მიერ განსაზღვრული მიმდევრობით იერარქიაში. ამავე დროს დამმუშავებელს ეძლევა სრული კონტროლის საშუალება იმაზე, თუ როგორ იმოქმედოს მათზე.

მაგალითად, განვიხილოთ მართვის ელემენტი Window, რომელიც შეიცავს Grid ელემენტს, რომელიც თავის მხრივ შეიცავს Rectangle ელემენტს (ნახ.7.24).



ნახ.7.24

თუ შესრულდა დაკლიკვა Rectangle მართვის ელემენტზე, მაშინ ადგილი ექნება შემდეგი მიმდევრობის მოვლენებს:

1. მაუსის ღილაკის დაკლიკვის მოვლენის გენერაცია Window-ში.
2. მაუსის ღილაკის დაკლიკვის მოვლენის გენერაცია Grid-ში.
3. მაუსის ღილაკის დაკლიკვის მოვლენის გენერაცია Rectangle-ში.

ამით პროცესი არ მთავრდება და შემდეგ მოხდება:

4. მაუსის ღილაკის დაკლიკვის კიდევ ერთი მოვლენის გენერაცია Rectangle-ში.

5. მაუსის ღილაკის დაკლიკვის კიდევ ერთი მოვლენის გენერაცია Grid-ში.

6. მაუსის ღილაკის დაკლიკვის კიდევ ერთი მოვლენის გენერაცია Window-ში.

დამმუშავებელს შეუძლია მოვლენაზე რეაგირება აღნიშნული მიმდევრობის ნებისმიერ წერტილში, ანუ უბრალოდ დაამატოს მოვლენის დამუშავების შესაბამისი მეთოდი.

მას შეუძლია ასევე ამ მიმდევრობის შეწყვეტა მოვლენის დამუშავების ჩარჩოს ნებისმიერ წერტილში.

XAML-კოდს აქვს ასეთი სახე:

```
<Window
    x:Class="Ch34Ex02.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/
                  xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Routed Events" Height="400" Width="800"

    MouseDown="Generic_MouseDown"
    PreviewMouseDown="Generic_MouseDown"
    MouseUp="Window_MouseUp">

 <Grid Name="contentGrid" MouseDown="Generic_MouseDown"
  PreviewMouseDown="Generic_MouseDown" Background="Azure">
   <Grid.RowDefinitions>
      <RowDefinition Height="332*" />
      <RowDefinition Height="29*" />
   </Grid.RowDefinitions>
      <Rectangle Name="clickMeRectangle"
          Margin="10,10,0,0" Height="23"
        HorizontalAlignment="Left" VerticalAlignment="Top"
          Width="70" Stroke="Black"
        MouseDown= "Generic_MouseDown" PreviewMouseDown=
                "Generic_MouseDown"
        Fill="CadetBlue" />
   <Button Name="clickMeButton" Margin="0,10,10,0"
         Height="23"
```



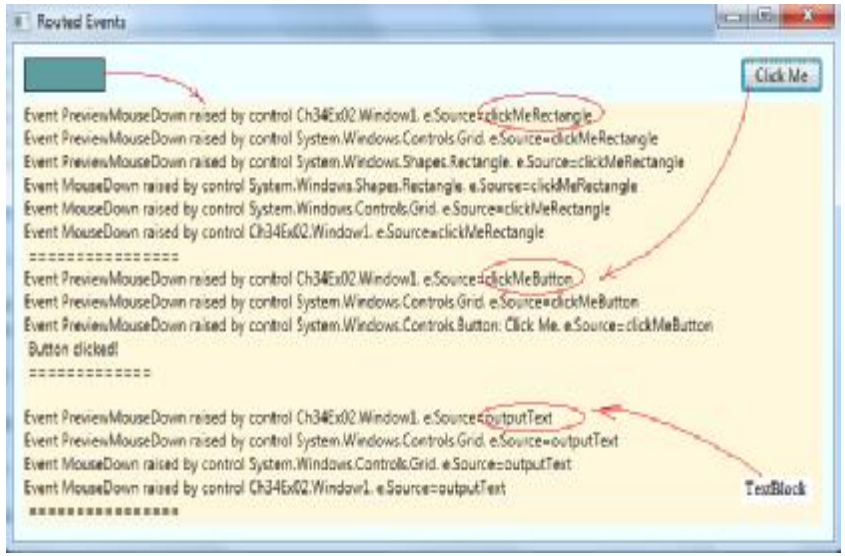
```
        HorizontalAlignment="Right" VerticalAlignment="Top"
           Width="70"
        MouseDown="Generic_MouseDown"
        PreviewMouseDown="Generic_MouseDown"
        Click="clickMeButton_Click">Click Me</Button>
         <TextBlock  Name="outputText" Margin="10,40,10,10"
              Background="Cornsilk" />
   </Grid>
</Window>
```

3. შევცვალოთ კოდი Window1. xaml.cs ფაილში შემდეგი სახით:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Media.Animation;   // დაემატა
namespace Ch34Ex02
{
public partial class Window1 : Window
 {
    private void Generic_MouseDown(object sender,
MouseButtonEventArgs e)
     {
       outputText.Text = string.Format("{0}Event {1}
raised by control {2}. e.Source={3}\n",
           outputText.Text, e.RoutedEvent.Name,
sender.ToString(), ((FrameworkElement)e.Source).Name);
      }
```

```
 private void Window_MouseUp(object sender,
                              MouseButtonEventArgs e)
  {
    outputText.Text = string.Format("{0}
             ================\n", outputText.Text);
  }
 private void clickMeButton_Click(object sender,
                                  RoutedEventArgs e)
  {
   outputText.Text = string.Format("{0} Button clicked!\n
          =============\n\n", outputText.Text);
   }
 }
}
```

4. ავამუშავოთ დანართი, მივიღებთ 7.25 ნახაზზე ნაჩვენებ შედეგს:



ნახ.7.25

დასკვნები

მონოგრაფიის ფარგლებში ჩატარებული საპროექტო–კვლევითი სამუშაოების შედეგების საფუძველზე შესაძლებელია შემდეგი დასკვნების გაკეთება:

1. მზარდი მოთხოვნა ინტეგრაციულ სისტემებზე, მონაცემთა დამუშავების ავტომატიზაციის ამოცანების, სქემების, მიდგომების, პრინციპების მუდმივი ცვლილება, ასევე ქსელური და აპარატურული ტექნოლოგიების შესაძლებლობების ზრდა, განაპირობებს Web-აპლიკაციების დაპროექტების და რეალიზაციის სისტემების პროგრამული უზრუნველყოფის განვითარებას სერვისორიენტირებული არქიტექტურით;

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

3. ობიექტორიენტირებული აპლიკაციების Web-სერვისებით ფორმირება განსაკუთრებით მოქნილი, მოსახერხებელი და გაცილებით საიმედოა, როგორც ახალი პროგრამული პროდუქტების შექმნისას, ისე არსებულ სისტემაში ცვლილებების გატარებისას. Web-სერვისი, შეიძლება განვიხილოთ, როგორც ხიდი ობიექტ-ორიენტირებულ და პროცეს-ორიენტირებულ ტექნოლოგიებს შორის, რაც კომპანიათაშორისი და კომპანიის მსხვილ სტრუქტურათაშორისი საქმიანი პროცესების ინტეგრაციასა და მრავალაპლიკაციურ მართვას უზრუნველყოფს;

4. სერვისორიენტირებულ არქიტექტურას ახლავს რიგი სირთულეები, რაც უკავშირდება Web-სერვისების ტიპიზირებას და ტიპიზირებული Web-სერვისების მასშტაბურ რაოდენობას, არასაკმარის ინსტრუმენტულ საშუალებებს, სტანდარტიზაციისა და ცენტრალიზაციის პრობლემების წარმოქმნას. მიუხედავად ამ სირთულეებისა, აპლიკაციების ინტეგრაციის თვალსაზრისით სერვისორიენტირებული არქიტექტურა და პროცეს-ორიენტირებული მიდგომა აქტუალური, მომქმედი და განვითარებადია, რასაც მხარს უჭერს MicroSoft და Java ტექნოლოგიები;

5. სისტემა MsBizTalk, როგორც ორგანიზაციის სერვისული არხის იმპლემენტაციის პლატფორმა, უზრუნველყოფს სხვადასხვა აპლიკაციებს შორის კავშირის დამყარებას. გრაფიკული ინტერფეისის საშუალებებით ახორციელებს ბიზნესპროცესების ორკესტრაციას, მონიტორინგს, როგორც პროცესების რაოდენობრივი შეფასებისთვის, ასევე გარკვეულ მოვლენებზე რეაგირებისთვის. BizTalk-ის დახმარებით შესაძლებელია პროცესების ავტომატიზაციის და ინდუსტრიული სტანდარტების უზრუნველყოფა, რაც საშუალებას იძლევა შემცირდეს დანახარჯები და კომპლექსურობა B2B კავშირების დასამყარებლად;

6. კორპორაციული მართვის ბიზნესპროცესების მოდელირებისა და ანალიზისთვის ეფექტურად გამოიყენება სტოქასტურ-დროითი პეტრის ქსელები, როგორც დინამიკური პროცესების იმიტაციური მოდელირების ინსტრუმენტი. აგებული პეტრის ქსელის მოდელების ვარიანტების ანალიზის საფუძველზე შესაძლებელია გარკვეული დასკვნების გაკეთება ობიექტზე ბიზნესპროცესების ეკვივალენტურ აქტიურობათა დიაგრამების

პროგრამული რეალიზაციის ეფექტიანობის შესახებ. აგრეთვე, პეტრის ქსელზე გადასასვლელთა (აქტიურობის დიაგრამაზე შესრულებადი პროცესების) დროითი პარამეტრების შერჩევით და „კონფლიკტური" გადასასვლელების გარკვეული ლოგიკური პირობების შემოტანით, შესაძლებელი ხდება გასაანალიზებელი შედეგების ხარისხის ამაღლება და ცალკეული სერვის-პროცესების დროითი მაჩვენებლების ანალიზი;

7. კორპორაციათაშორისი საინფორმაციო მართვის სისტემების რეალიზაციის მიზნით სერვის-ორიენტირებული არქიტექტურით, განსაკუთრებით მნიშვნელოვანია მონაცემთა გაცვლის საშუალებების ფუნქციონირების ეფექტურობის ანალიზი, რაც წარმატებით ხორციელდება ფერადი პეტრის ქსელების (CPN) ინსტრუმენტით. მასში კარგადაა შერწყმული პეტრის სტოქასტური ქსელებისა და ობიექტ-ორიენტირებული დაპროგრამების თეორიის ძირითადი პრინციპები (იერარქიულობა, მოდულურობა, მემკვიდრეობითობა, პოლიმორფიზმი – დიდი სისტემების მოდელირებისთვის), რაც მის დიდ პრაქტიკულ ღირებულებასაც განაპირობებს თანამედროვე ინფორმაციულ ტექნოლოგიათა გამოყენების მრავალ სფეროში;

8. თანამედროვე ინფორმაციული ტექნოლოგიების, კერძოდ CASE-ინსტრუმენტების გამოყენებით კორპორაციული მენეჯმენტისა და ინტერკორპორაციული ვებ-აპლიკაციების აგების პროცესში, მნიშვნელოვნად უმჯობესდება პროგრამული უზრუნველყოფის ხარისხი და საგრძნობლად მცირდება დაპროექტების, მისი იმპლემენტაციისა და რეინჟინერინგის პერიოდები. დიდი საინფორმაციო სისტემების მონაცემთა ბაზების სტრუქტურების დაპროექტებისა და აგების პროცესების

ავტომატიზება, აგრეთვე მისი შემდგომი რესტრუქტურიზაციის პრობლემების მოქნილად გადაწყვეტის საშუალებას იძლევა.

9. კორპორაციული მენეჯმენტის მართვის საინფორმაციო სისტემების ჰიბრიდული პაკეტების შესაქმნელად, მაღალხარისხოვანი დიზაინის პროგრამების თვალსაზრისით, დღეისათვის ერთ-ერთი მნიშვნელოვანი ინსტრუმენტია მაიკროსოფტის ფირმის ინტეგრირებული Ms Visual Studio .NET FrameWork 4.0 პაკეტის Windows Presentation Foundation პლატფორმა. კორპორაციული მენეჯმენტის სისტემის მომხმარებელთა ინტერფეისების შემუშავება და პროგრამული რეალიზაცია ამ პლატფორმაზე საშუალებას იძლევა ერთიანი, ინტეგრირებული სისტემის შესაქმნელად, ეფექტური და ამავდროულად უსფრთხო ფუნქციონირების პირობების უზრუნველყოფით;

10. XAML და C# ენების ბაზაზე მიიღება Windows- და Web-აპლიკაციების პროგრამული დანართები, რომელთა ინტეგრაცია კორპორაციების მონაცემთა საცავებთან (ან მონაცემთა განაწილებულ ბაზებთან) სამუშაოდ, ობიექტ-ორიენტირებულ, პროცეს-ორიენტირებულ და, განსაკუთრებით, სერვის-ორიენტირებული არქიტექტურის კომპლექსური გამოყენების საფუძველზე იძლევა პროგრამული ინჟინერიის დისციპლინის განვითარების თვისობრივად ახალ დონეს, რაც უშუალოდ აისახება როგორც პროგრამული სისტემების პროდუქტიულობის ხარისხზე, ასევე მათი თანხლებისა და განვითარების პროცესების სრულყოფაზე, სასიცოცხლო ციკლის დროითი მაჩვენებლების გაუმჯობესებაზე.

ლიტერატურა:

1. ბულია ი. თანამედროვე სისტემებში ინტეგრაციის, მონაცემთა გადაცემის და დამუშავების ტექნოლოგიები. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N2(11), თბილისი, 2011, გვ. 139–144.

2. თურქია ე., ბულია ი., გიუტაშვილი მ. ინტერ-კორპორაციული აპლიკაციების ჰორიზონტალური და ვერტიკალური ინტეგრაციის მართვა სერვის-ორიენტირებული არქიტექტურის ბაზაზე. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(12). 2012. გვ. 57–62.

3. ბულია ი. ბიზნეს-აპლიკაციების აგების თანამედროვე ინფორმაციული ტექნოლოგიები. საერთაშ. სამეცნ. ტექნ. კონფ.: „მართვის ავტომატიზ. სისტ. და ახალი_ინფორმაციული ტექნოლოგიები“. სტუ, თბილისი, 20-22 მაისი, 2011

4. Surguladze G., Topuria N., Burchuladze A., Bulia I. Modelling and automation of monitoring and control processes for transporting oil products. Intern.Conf. of Comp. Inf.and Comp. Technologies, Modelling, Control. Tbilisi, Georgia, Nov.1-4, 2010. p.37-39

5. სურგულაძე გ., ბულია ი., კაშიბაძე მ., შურღაია ი. სერვის-ორიენტირებული არქიტექტურის ინტერკორპორაციული ვებ-აპლიკაციების ბიზნეს-პროცესების მოდელირება და კვლევა პეტრის ქსელებით. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(12). 2012. გვ. 63–72.

6. სურგულაძე გ., ბულია ი., ოხანაშვილი მ., ქრისტესიაშვილი ხ. კორპორაციული მენეჯმენტის ბიზნეს-პროცესების მოდელირება და კვლევა ფერადი პეტრის ქსელებით. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(12). 2012. გვ. 73–82.

7. სურგულაძე გ., თოფურია ნ., ბულია ი., ინტერკორპორაციული სერვის-ორიენტირებული სისტემის მონაცემთა ბაზების დაპროექტება და რეალიზაცია. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(12). 2012. გვ. 83–89.

8. სურგულაძე გ., თოფურია ნ., მარკოსიანი დ., ბულია ი. სოციალური მომსახურების სისტემაში მონიტორინგის პროცესების ავტომატიზაცია. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. - N 1(8), 2010. გვ. 175–180.

9. სურგულაძე გ., ბერძენიშვილი ი., ვაჭარაძე ი., ხელაძე ნ., ბულია ი. კორპორაციული მართვის სისტემის ვებ-აპლიკაციის დამუშავება ინტერნეტ-ინტრანეტ გარემოში .Net-პლატფორმაზე. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. - N1, 2006. გვ. 159–162.

10. სურგულაძე გ., პეტრიაშვილი ლ., ყვავაძე ლ., ბულია ი. ბიზნეს-პროცესების მართვის ინტერნეტული სისტემის დაპროექტება და რეალიზაცია ASP და C# .NET-ტექნილოგიებით. საქ.მეცნ.აკად., “მეცნიერება და ტექნოლოგიები”, N10-12. თბილისი, 2005. გვ. 45–49.

11. სურგულაძე გ., თურქია ე., ბულია ი. ვებ-აპლიკაციების აგება ASP & ADO & C# პაკეტებით .NET პლატფორმაზე. ISBN 978-9941-14-365-6. სტუ, თბ., 2009. 86 გვ.

12. სურგულაძე გ., თურქია ე., ბულია ი. ვებ-აპლიკაციების დამუშავება მონაცემთა ბაზების საფუძველზე (ADO.NET, ASP.NET, C#). ISBN 978-9941-14-289-5. სტუ, თბ., 2009. 172 გვ.

13. Krafzig D., Banke K., Slama D. Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall. 2004

14. სურგულაძე გ., პეტრიაშვილი ლ. მონაცემთა საცავის აგების ტექნოლოგია ინტერნეტული ბიზნესის სისტემებისათვის. სტუ. თბ., 2005

15. თურქია ე. ბიზნეს–პროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. სტუ. თბ., 2010

16. თურქია ე., გიუტაშვილი მ. კორპორაციულ სისტემებში ინტელექტუალური რესურსების მენეჯმენტი. სტუ. თბ., 2008

17. Clarke R. Electronic Data Interchange (EDI): An Introduction. http://www.roger clarke.com/EC/EDIIntro.html

18. Sokol K. P. From Edi to Electronic Commerce: A Business Initiative. McGraw-Hill. 1995

19. Freund J. Zwölf Fragen zum Business-Process-Management. http://wiki. computerwoche.de/doku.php/soa_bpm/bpm_faq

20. Seeley R. Business-side often drives BPM initiatives today, http://searchsoa. techtarget. com/news/1323855/Business-side-often-drives-BPM-initiatives-today

21. Loesgen B., Young C., Eliasen J., Colestock S., Kumar A. Microsoft BizTalk Server 2010.

22. Magal S.R., Word J. Essentials of Business Processes and Information Systems. Wiley

23. Erl T. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall.

24. Chappell D. Enterprise Service Bus: Theory in Practice. O'Reilly Media

25. Sarang P., Jennings F., Juric M., Loganathan R. SOA Approach to Integration: XML, Web services, ESB, and BPEL in real-world SOA projects. Packt Publishing.

26. Goel A. Enterprise Integration EAI vs. SOA vs. ESB, http://ggatz.com/images/Enterprise_20Integration_20-_20SOA_20vs_20 EAI_20vs_ 20ESB.pdf.

27. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley.

28. Ruh A. W., Maginnis X. F., Brown J. W. Enterprise Application Integration: A Wiley Tech Brief. Wiley.

29. Bussler Ch. B2B Integration: Concepts and Architecture. Springer.

30. Oracle Enterprise Application Integration Services. http://www.oracle.com/ us/products/consulting/resource-library/enterprise-application-integration-069324.pdf.

31. Understanding Enterprise Application Integration - The Benefits of ESB for EAI. http://www.mulesoft.org/enterprise-application-integration-eai-and-esb.

32. Буч Г., Рамбо Дж., Джакобсон А. Язык UML. Руководство пользователя. Пер.с англ., Питер, 2004.

33. Jablonski, S., Petrov, I., Meiler, C., Mayer, U. Guide to Web Application and Platform Architectures, Germany, Springer-Verlag Berlin And Heidelberg Gmbh, 2004.

34.http://www.microsoft.com/bi. უკანასკნელად გადამოწმებულ იქნა–22.01.2012

35.http://www.sdgcomputing.com/glossary.htm. უკანასკნელად გადამოწმებულ იქნა–22.03.2012

36. ვედეკინდი ჰ., სურგულაძე გ., თოფურია ნ. განაწილებული ოფის-სისტემების მონაცემთა ბაზების დაპროექტება და რეალიზაცია UML-ტექნოლოგიით. მონოგრ., სტუ, თბ., 2006

37. სურგულაძე გ., შონია ო., ყვავაძე ლ. მონაცემთა განაწილებული ბაზების მართვის სიტემები. სტუ, თბ., 2004.

38. სურგულაძე გ., დაპროგრამების ვიზუალური მეთოდები და ინსტრუმენტები (UML, MsVisio,C++Builder). სტუ, თბ., 2005.

39. ბოტჰე კ., სურგულაძე გ., დოლიძე და სხვ.. თანამედროვე პროგრამული პლატფორმები და ენები (WindowsNT, Unix, Linux, C++, Java, XML). სტუ. თბ.,2003

40. სურგულაძე გ., გულუა დ. განაწილებული სისტემების ობიექტ_ორიენტირებული მოდელირება უნიფიცირებული პეტრის ქსელებით. თბილისი: ტექნიკური უნივერსიტეტი, 2005, გვ. 210.

41. http://www.sparxsystems.com.au/platforms/business_process_ modeling.html გადამოწმდა – 15.04.2012

42. რესიგი ვ., სურგულაძე გ., გულუა დ, „ვიზუალური ობიექტ–ორიენტირებული დაპროგრამების მეთოდები". სტუ, თბილისი, 2002.

43. Jablonski S. Projekt-ForFlow. www.ai4.uni-bayreuth.de/en/ research/ projects /010_ forflow/index.html.

44. http://webservices.xml.com/lpt/a/ws/2003/09/30/soa.html. გადამო-წმდა–25.04.2012

45. Service-Oriented Architecture: A Primer. Michael S. Pallos. http://www.bijonline.com/PDF/SOAPallos.pdf. გადამოწმდა–25.04.2012

46. თურქია ე. ელექტრონული ბიზნესისა და ელექტრონული კომერციის სისტემების მოდელირება და დაპროექტება. სტუ, თბილისი, 2009.

47. Open Management Group, Business Process Management Initiative: Business Process Modeling Notation (BPMN), http://www.bpmi.org, 2006

48. Surguladze G., Turkia E., Topuria N., Giutashvili M. Development and Research of the Computer System Models Supporting the Human Resource Selection for the Project Management, 3 rd Intern. Conf., Computational Intelligence (CI'09), Tbilisi, Georgia 2009

49. სურგულაძე გ., თოფურია ნ., ყვავაძე ლ. განაწილებული მონაცემთა ბაზების აგების ავტომატიზაცია .NET გარემოში. სტუ შრ.კრ."მას" N1(2), 2007. გვ.105-108

50. Николаишвили В., Сургуладзе Г. Топурия Н., Кашибадзе М. Категориальный подход разработки абстрактных моделей данных для объектно-ориентированных, реляционных баз данных. Тез.Докл.Интерн.Конф. Киев, 2006.

51. Halpin T., ORM 2 Graphical Notation, Neumont University, 2005. http://www.orm.net/ pdf/ORM2_TechReport1.pdf

52.Bolch G., Greiner S., DeMeer H., Travedi K. Queueing Networks and Markov Chains. John Wiley&Sons.Inc., 2000.

53.Питерсон Дж. Теория Сетей Петри и моделлирование систем. Перевод с английского. Москва, «Мир», 1983

54. Reisig W. Elements of Distributed Algorithms : Modeling and Analysis with Petri Nets. Berlin. Heidelberg. New York et al. Springer, 1998

55. CPN Tools. www.daimi.au.dk/CPNTools/. გადამოწმებულია 1.10.08

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

57. http://www.gnuplot.info გადამოწმებულია 10.10.08

58. ბოლხი გ., სურგულაძე გ., პეტრიაშვილი ლ., ჩიხრაძე ბ. მულტიპროცესორული სისტემების რესურსების მართვის პროგრამული უზრუნველყოფის დამუშავება Borland_C++Builder ინსტრუმენტით. სტუ-ს შრ., N 4(437), თბილისი, 2001

59. ჩოგოვაძე გ., გოგიჩაიშვილი გ., სურგულაძე გ., შეროზია თ., შონია ო. მართვის ავტომატიზებული სისტემების დაპროექტება და აგება. სტუ, თბ., 2001

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

61. მახარაძე კ., კვანჭახაძე დ., სურგულაძე გ. სოციალური მომსახურების სააგენტოს განვითარების სადიაგნოსტიკო პროექტის ანგარიში. USAID საჯარო სამსახურის რეფორმის (PAR) პროგრამის ფარგლებში. კომპანია BIT. თბ., 2009. www.bit.ge

62. სურგულაძე გ., ოხანაშვილი მ., სურგულაძე გ. მარკეტინგის ბიზნეს-პროცესების უნიფიცირებული და იმიტაციური მოდელირება. სტუ. თბილისი. 2009.

63. ოხანაშვილი მ., პეტრიაშვილი ლ. მარკეტინგული პროცესების იმიტაციური მოდელის აგება აგენტური მოდელირების გამოყენებით. სტუ შრ.კრ. N1(2) 2007.

64. ოხანაშვილი მ., შარაშიძე თ. პროდუქციის წარმოების ინფორმაციულ-ტექნოლოგიური პროცესების უნიფიცირებული და იმიტაციური მოდელირება. სტუ შრ.კრ. N1(4) გვ.108_113 2008.

65. სურგულაძე გ., თურქია ე., ოხანაშვილი მ., სურგულაძე გ. მარკეტინგული პროცესების მართვის ერთი მოდელის შესახებ ფერადი პეტრის ქსელებით. სტუ შრ.კრ. N2(5), 2008. გვ. 9_16.

66. ჯავახაძე გ. ბიზნეს-პროგრამების მართვისას გადაწყვეტილებათა მიღების მხარდამჭერი მოდელები და მეთოდები. სტუ. თბილისი 2003.

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

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

69. Уотсон К.,Нейгел К., Педерсен Я., ХаммерР., Джон Д., Скиннер М., Уайт Э. Visual C# 2008: базовый курс. : Пер. с англ. - М. : ООО "И.Д. Вильяме", 2009

70. Долженко А.И. Разработка приложений на базе WPF и Silverlight. –М., 2011, www.intuit.ru/department/se/dawpfsl/

71. Алексеев А.А. Технологии Microsoft для создания RIA-приложений (Rich Internet Applicartions). –М. Intuit. 2011. http://www.intuit.ru/department/se/tmsria/

72. Eberhardt C. WPF DataGrid Practical Examples. 2009. http://www.codeproject.com/Articles/30905/WPF-DataGrid-Practical-Examples

73. Surguladze G., Turkia E., Gulua D. Perfection of Object-Oriented Projecting with a Process-Oriented Approach. Internation. Conference.

Educat, science and economics at univ.Integrat.to intern.educ.area. Płock, Poland, 2008

74. Surguladze G., Topuria N., Petriashvili L. Construction of Multi-dimensional Analysis Packet of Commercial Objects with Decision Cube Components. Educat, science and economics at univ.Integrat.to intern.educ.area. Płock, Poland, 2008.

75. Surguladze G., Turkia E., Topuria N., Lominadze T., Giutashvili M. Towards an Integration of Process-Modeling: from Business Method: from Business-Content to the Software Implementation. IV Intern. Conf. Problems of Cybernetics and Informatics (PCI' 2012). vol.1. Baku, Azerbaijan, Sept.12-14, 2012. 16-19 pp.

იბეჭდება ავტორთა მიერ
წარმოდგენილი სახით

გადაეცა წარმოებას 20.10.2012 წ. ხელმოწერილია დასაბეჭდად 20.12.2012 წ. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 20,5. ტირაჟი 100 ეგზ.

საგამომცემლო სახლი ,,ტექნიკური უნივერსიტეტი"
თბილისი, მ. კოსტავას 77