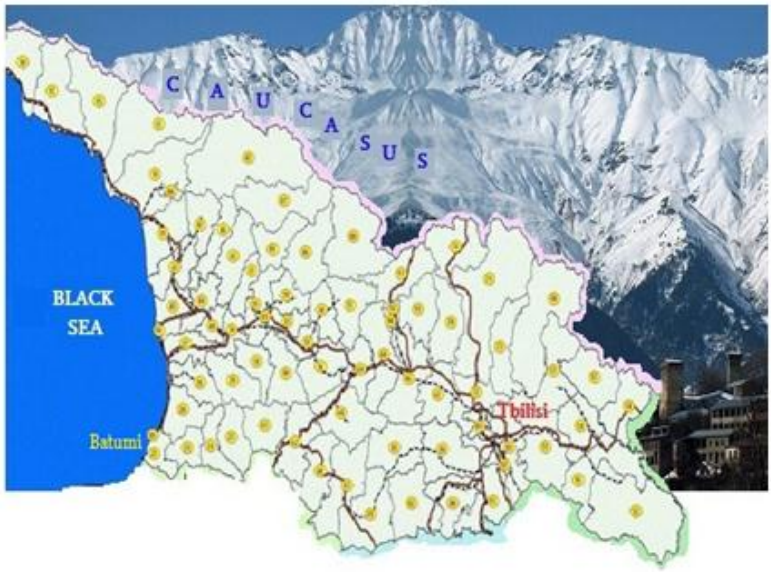
კლაუს მეიერ-ვეგენერი,
გია სურგულაძე,
გიორგი ბასილაძე

საინფორმაციო სისტემების აგება მულტიმედიური მონაცემთა ბაზებით



„ტექნიკური უნივერსიტეტი“

საქართველოს ტექნიკური უნივერსიტეტი

კლაუს მეიერ-ვეგენერი, გია სურგულაძე,
გიორგი ბასილაძე

საინფორმაციო სისტემების აგება მულტიმედიური მონაცემთა ბაზებით

(ელექტრონული საარჩევნო სისტემის პროექტი)

დამტკიცებულია საქართველოს
ტექნიკური უნივერსიტეტის
სარედაქციო-საგამომცემლო საბჭოს
მიერ 06.06.2014, ოქმი №2

თბილისი 2014

უაკ 004.5

განიხილება კორპორაციულ სისტემებში თანამედროვე ინფორმაციული ტექნოლოგიების საშუალებების, მათ შორის მონაცემთა მულტიმედიური საცავების დაპროექტების, პროგრამული და აპარატურული რეალიზაციის პრობლემები და მათი გადაწყვეტის გზები სერვისორიენტირებული არქიტექტურის ბაზაზე. შემოთავაზებულია ელექტრონული საარჩევნო სისტემის, როგორც რთული და დიდი სისტემის მოდელირების, ობიექტორიენტირებული დაპროექტების, ობიექტორიენტირებული ანალიზის და შემდგომი პროგრამული რეალიზაციის საკითხები. გაანალიზებულია დასმული პრობლემების გადაწყვეტის საზღვარგარეთული გამოცდილება, არსებული მეთოდები, არქიტექტურა, პრინციპები, მოდელები, რომლებიც აუცილებელია ელექტრონული საარჩევნო სისტემის წარმატებით დასანერგად.

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

რეცენზენტები:

- სრული პროფესორი - გ. გოგიჩაიშვილი
- სრული პროფესორი - რ. სამხარაძე
- სრული პროფესორი - ა. ქუთათელაძე

პროფ. გ. სურგულაძის რედაქციით

ISBN 978-9941-20-468-5

Georgian Technical University

Klaus Meyer-Wegener, Gia Surguladze,

Giorgi Basiladze

CONSTRUCTION OF INFORMATION SYSTEMS WITH MULTIMEDIA DATABASES

Supported by DAAD (Germany)

ISBN 978-9941- 20-468-5

ABSTRACT

Review of the multimedia tools in corporate systems, including the data storage design and solving software and equipment problems and implementing solutions for the modern client - server architecture base.

Is proposed the electronic voting system as a complex and large system of modeling, object - oriented software design, object – oriented analysis and object – oriented implementation issues in process. The problem-solving experience from overseas, the existing methods, architecture, principles and models, which are necessary for the successful introduction of an electronic voting system, are analyzed.

The existing traditional voting system and its accompanying problems in Georgia are reviewed. For modeling business processes of the existing and electronic voting systems BPMN (Business Process Modeling Notation) technology has been used. Corresponding, process oriented models are created and complexities and opportunities of their reliable functioning assessed. Practical experience of the advantages and disadvantages in the electronic voting system of the leading countries has been examined.

The concept of building a state level safe network and its architecture are proposed. The necessary relational multimedia data bases are designed for the electronic voting system. In this point of view the usage of categorical modeling methods in design and object – oriented, CASE (Computer Aided System Engineering) - technologies are reviewed.

The categories of authentication have been reviewed; and based on a deep analysis the combination of different types of authentication and their integration into multimedia database have been selected for the electronic voting system. These are: fingerprints scanner, voice audio recorder, biometric photo, and electronic signature. Through the integration of these modules increases security, as well as trust and faith.

Logically whole client - server architecture and physically separated relational database system is developed. Designing structure of the multimedia relational database took place in automatic regime based on grammar-theory (categorical analysis), logical algebra, object-role

modeling principles and appropriate graph-analytical tools on the basis of Natural ORM Architect system. Created for assessing the functionality of managing business processes of the computer-aided voting system, namely for changing information affectively between central office and its branches, the mathematical apparatus of Coloured Petri nets is used. This is an example of imitative modeling and analysis.

The principles of safe and reliable network operations in the proposed electronic voting system are reviewed for modern communication base facilities. The methods of projecting and Router configurations on the basis of VPN (Virtual Private Network) e-governance are offered. Aspects of inclusion of central election commission, and district and precinct commissions into one united governmental net were discussed. The means of designing and implementing server through the client – server architecture are discussed. Server operational systems and applications, which are necessary for developing electronic voting system, are offered.

The experimental pilot software versions of the multimedia electronic voting system is implemented on the basis of new information technologies, such as Visual Studio.Net, SQL Server, ORM / ERM, CPN, VPN technologies and software packages. The modern technology – Windows Presentation Foundation (WPF) and Metro Style App, is a real innovation and is used by Microsoft Windows 8. The complex of program systems uses the modern declarative (Script) XAML language (eXtensible Application Markup Language) to develop design of the system and to describe the logic behind the functioning of the object oriented C# language. The interfaces, instructions, implementation and exploitation processes of organizational, technical and legal aspects have been developed for consumers of the construction system. The procedural changes, which should be implemented on the approved election districts, are described. Change of the protocol described and approved election date will take place. The necessary technical means and their capabilities for the system are defined. An approximate budget and estimate calculation for the electronic voting system, as well as its support with IT technologies, is prepared.

Klaus.Meyer-Wegener@fau.de

Prof. Dr. Meyer-Wegener Klaus

University of Erlangen and Nuremberg, Faculty of Engineering, Department of Computer Sciences. Chair for Computer Science 6 - „Data Management”

კლაუს მეიერ-ვეგენერი

ნიურნბერგ-ერლანგენის უნივერსიტეტის „მონაცემთა მენეჯმენტის (ბაზების)“ კათედრის გამგე, ტექნიკის მეცნიერებათა დოქტორი, პროფესორი (გერმანია)

Prof. Dr. Surguladze Gia

Georgian Technical University, Faculty of Informatics and Management Systems, Department of „Software Engineering”

გია სურგულაძე

საქართველოს ტექნიკური უნივერსიტეტი, ინფორმატიკისა და მართვის სისტემების ფაკულტეტი. „პროგრამული ინჟინერიის“ დეპარტამენტის სრული პროფესორი, ტექნიკის მეცნიერებათა დოქტორი. IT კონსალტინგის სამეცნიერო ცენტრის ხელმძღვანელი.

gsurg@gmx.net

gbasiladze@yahoo.com

Asoc.Prof. Dr. Basiladze Giorgi

Georgian Technical University, Faculty of Informatics and Management Systems, Department of „Software Engineering”

გიორგი ბასილაძე

საქართველოს ტექნიკური უნივერსიტეტი, ინფორმატიკისა და მართვის სისტემების ფაკულტეტი. „პროგრამული ინჟინერიის“ დეპარტამენტის ასოცირებული პროფესორი, აკადემიური დოქტორი. ფლობს ინგლისურ, გერმანულ, ბერძნულ და რუსულ ენებს.

შინაარსი

შესავალი ---------- 11

1 თავი. მულტიმედიური მონაცემთა საცავების დაპროექტების პრობლემები და მათი გადაჭრის გზები ელექტრონული საარჩევნო სისტემის ასაგებად ---------- 19

1.1. კვლევის ობიექტის დახასიათება, როგორც მართვის რთული და დიდი სისტემა ---------- 19

1.2. მონაცემთა საცავების სტრუქტურა და შედგენილობა მულტიმედიური ელემენტების გამოყენებით ---------- 21

1.3. კლიენტ-სერვერული არქიტექტურა კორპორაციულ სისტემებში ---------- 34

1.4. მულტიმედიური მონაცემების აუთენტიფიკაცია ------- 41

1.5. თანამედროვე დაცული ქსელის აგება VPN-ის გამოყენებით ---------- 50

1.6. ბიზნესპროცესების მოდელირება და მართვა (BPMN ინსტრუმენტული საშუალებით) ---------- 52

1.7. სერვერული ოპერაციული სისტემების მიმოხილვა------ 62

1.8. აპლიკაცია Active Directory, როგორც ტექნოლოგია და მისი დანერგვის გეგმარება ---------- 72

1.9. ობიექტ-როლური მოდელირება (ORM) განაწილებული სისტემებისთვის ---------- 78

1.10. ბიზნესპროცესების მოდელირების, ანალიზის და შეფასების ინსტრუმენტი: პეტრის ქსელები---------- 88

1.11. მულტიმედიური მონაცემთა ბაზების დაპროექტების და გამოყენების ამოცანა ელექტრონული საარჩევნო სისტემის ასაგებად ---------- 90

II თავი. მულტიმედიური მონაცემთა ბაზების დამუშავება ელექტრონული საარჩევნო სისტემისთვის ---------- 93

2.1. ტრადიციული და ელექტრონული საარჩევნო სისტემების ბიზნესპროცესების აღწერა BPMN ენაზე -- 93

2.2. სისტემის კონცეპტუალური მოდელის ავტომატიზებული აგება ORM/ERM ტექნოლოგიით --- 97

2.3.	მონაცემთა ბაზის აგების პროგრამული რეალიზაციის ავტომატიზებული პროცედურა ----------	112
2.4.	სისტემის ტექნიკური უზრუნველყოფის მახასიათებლების განსაზღვრა და შეფასება ----------	118
2.5.	VPN ქსელის დაპროექტება, აგება და უსაფრთხოების ნორმების უზრუნველყოფა ----------	125
2.6.	ელექტრონული საარჩევნო სისტემის დამუშავების პროცესების იმიტაციური მოდელირება ფერადი პეტრის ქსელით ----------	130
2.7.	სისტემის ფუნქციონირების ორგანიზების ასპექტები---	135
III	თავი. მონაცემთა მოდელები და მულტიმედიური ბაზები ----------	143
3.1.	მონაცემთა მოდელები მულტიმედიისთვის ----------	143
3.1.1.	მონაცემთა ახალი ტიპები ----------	145
3.1.2.	კლასთა იერარქიის აგება განზოგადოების საფუძველზე ----------	152
3.1.3.	SQL/MM: მულტიმედიური მონაცემები და სტანდარტები ----------	154
3.1.3.1.	SQL/MM: სრული ტექსტი (FullText) ----------	155
3.1.3.2.	SQL/MM: სივრცითი ტიპი (Spatial) ----------	158
3.1.3.3.	SQL/MM: სტილი სურათი/ფოტო (Styl Image) ----------	160
3.2.	ობიექტრელაციური მულტიმედიური მონაცემთა ბაზის სისტემები ----------	165
3.2.1.	მოკლე დახასიათება ----------	166
3.2.2.	ობიექტრელაციური ბაზის სტრუქტურა მულტიმედია ობიექტისთვის ----------	171
3.2.3.	მოთხოვნების დამუშავება ობიექტრელაციურ მონაცემთა ბაზებში ----------	174
3.3.	ობიექტორიენტირებული მულტიმედიური მონაცემთა ბაზის სისტემები ----------	179
3.3.1.	მულტიმედიურ მონაცემთა ტიპების ჩაშენება ----------	179

3.3.2.	მულტიმედიური ინფორმაციული მენეჯერი (MIM) -----	182
3.4.	მულტიმედიური რელაციური ბაზების ოპტიმალური სტრუქტურის დაპროექტება ----------	192
IV	**თავი. სისტემის პროგრამული რეალიზაცია და ექსპლუატაციის მხარდაჭერა ----------**	203
4.1.	ელექტრონული საარჩევნო სისტემის კომპონენტების დამუშავება Visual Studio.NET Framework 4.0 გარემოში--	203
4.1.1.	DBGrid-foto აპლიკაცია ----------	203
4.1.2.	DBGrid-foto აპლიკაციის დიზაინის Default.aspx კოდი --	213
4.1.3.	DBGrid-foto აპლიკაციის ლოგიკის Default.aspx.sc კოდი -	219
4.2.	მომხმარებელთა ინტერფეისების დამუშავება ----------	229
4.3.	მომხმარებელთა ინსტრუქციები ----------	235
4.4.	შედეგების ანალიზი და შემდგომი სინქრონიზაცია ----	242
4.5.	სისტემის განვითარების კონცეფცია ----------	244
V	**თავი. კლიენტ-სერვერ არქიტექტურის პროგრამული რეალიზაცია ----------**	**247**
5.1.	კომუნიკაცია: WCF - ტექნოლოგია ----------	247
5.1.1.	WPF პროექტის შექმნა ----------	248
5.1.2.	ახალი config-ფაილის და Class-ების შექმნა ----------	249
5.1.3.	WindowForm-ის განსაზღვრა ----------	255
5.1.4.	ტექსტის ჩამწერის რეალიზაცია (TextWriter) ----------	257
5.1.5.	აპლიკაციის სტატიკური მიმთითებლის უზრუნველყოფა ----------	258
5.1.6.	ListBoxTextWriter–ის რეალიზაცია ----------	260
5.1.7.	მუშა პროცესების რეალიზაცია ----------	264
5.1.8.	შეტყობინებათა მიღება ----------	264
5.1.9.	სერვისის კონტრაქტის რეალიზაცია ----------	266
5.1.10.	ServiceHost -ის რეალიზაცია ----------	268
5.1.11.	სანიშნეები (Bookmarks) ----------	270
5.1.12.	SendRequest მუშა პროცესის რეალიზაცია ----------	272
5.1.13.	ProcessRequest მუშა პროცესის რეალიზაცია ----------	275

5.1.14.	აპლიკაციის რეალიზაცია ----------	278
5.1.15.	მუშა პროცესების ეგზემპლარების მხარდაჭერა ----------	279
5.1.16.	მოვლენათა დამმუშავებელი (Event Handlers) ----------	280
5.1.17.	ApplicationInterface მეთოდები ----------	279
5.1.18.	აპლიკაციის ამუშავება ----------	288
VI	**თავი. სერვისორიენტირებული არქიექტურის პროგრამული რეალიზაცია ----------**	**292**
6.1.	ბიზნესპროცესის სერვისის შექმნა ----------	292
6.2.	სერვისის კონტრაქტის განსაზღვრა ----------	294
6.3.	Receive და SendReply კონფიგურირება ----------	299
6.4.	PerformLookup აქტიუროზის შექმნა ----------	303
6.5.	სერვისის ტესტირება ----------	307
6.6.	პარამეტრების გამოყენება ----------	311
6.7.	მეორე სერვისის შექმნა ----------	311
6.8.	მოდიფიცირებული PerformLookup2 ქმედების შექმნა და სერვისის ტესტირება ----------	316
6.9.	კლიენტის ბიზნესპროცესის შექმნა ----------	322
6.10.	ბიზნესპროცესის განსაზღვრა ----------	326
6.11.	ჰოსტ-აპლიკაციის რეალიზაცია და ამუშავება ----------	328
6.12.	Pick-ის გამოყენება	330
6.13.	რეზიუმე	332
VII	**დასკვნა ----------**	334
	ლიტერატურა ----------	337

შესავალი

დემოკრატიული სახელმწიფოს მშენებლობისა და სამართლიანი საზოგადოების ჩამოყალიბების პროცესები ბევრადაა დამოკიდებული მრავალ ობიექტურ და სუბიექტურ ფაქტორზე, რომელთაგან ერთ-ერთი მთავარი ქვეყნის საარჩევნო სისტემის რეფორმაა. იგი განსაკუთრებით მნიშვნელოვანია ჩვენი ქვეყნისთვისაც, რომელიც ფართო პოლიტიკური სპექტრითა და მოსახლეობის მაღალი უნდობლობის ფაქტორებით ხასიათდება.

1990 წლიდან დაწყებული საქართველოში მრავალი არჩევნები ჩატარდა, როგორც ადგილობრივი თვითმართველობის, ასევე საპარლამენტო და საპრეზიდენტო. ჩატარდა რამდენიმე რეფერენდუმიც, თუმცა მოსახლეობა, მიუხედავად ამდენი ჩატარებული არჩევნებისა, დიდ უნდობლობას გამოხატავს ამ პროცესების მიმართ. ვერ მოხერხდა ქართული საარჩევნო კანონმდებლობის და საარჩევნო სისტემის დახვეწა. აღარაფერს ვამბობთ საარჩევნო სიაზე და მის ფორმირებაზე, სადაც დღეის მდგომარეობითაც უამრავი ხარვეზია. გარდა ამისა, არსებული სისტემით არჩევნების ჩატარება სახელმწიფოსათვის დიდ ხარჯებთან არის დაკავშირებული.

ამგვარად, ელექტრონული საარჩევნო სისტემის დაპროექტება, აგება და შემდგომი სრულყოფა უახლესი ინფორმაციული ტექნოლოგიების, განსაკუთრებით მულტიმედიური საშუალებებისა და საიმედო, უსაფრთხო ქსელების ბაზაზე, მეტად აქტუალურია.

მონოგრაფიის მიზანია ელექტრონული საარჩევნო სისტემის კონცეფციის შემუშავება, მისი დაპროექტება და შესაბამისი პროგრამული სისტემის აგება მულტიმედიური მონაცემთა ბაზების და კლიენტ-სერვერ არქიტექტურის საფუძველზე სახელმწიფოს ერთიანი დაცული ქსელის დაპროექტების და აგების ბაზაზე.

მიზნის მისაღწევად ნაშრომში განიხილება შემდეგი ძირითადი ამოცანები:

- არსებული ტრადიციული საარჩევნო სისტემის ანალიზი საერთაშორისო გამოცდილების საფუძველზე. პრობლემების გამოკვეთისა და მათი გადაჭრის გზების ერთიანი კონცეფციის შემუშავება, პოლიტიკური, სოციალური, ეკონომიკური, კულტურული და ტექნიკურ-ტექნოლოგიური ასპექტების გათვალისწინებით;

- ტრადიციული და ელექტრონული საარჩევნო სისტემების ბიზნესპროცესების მოდელების აგება სისტემური ანალიზის საფუძველზე, BPMN და UML ტექნოლოგიების ბაზაზე. მათი შედარებითი ანალიზი და დასაპროექტებელი მულტიმედიური საარჩევნო სისტემის ფუნქციურ მოთხოვნილებათა განსაზღვრა;

- თანამედროვე ელექტრონული საარჩევნო სისტემის მონაცემთა მულტიმედიური ბაზების ანალიზი და შესაბამისი ინფორმაციული ტექნოლოგიებით ინფრასტრუქტურული სქემის შემუშავება, სისტემის არაფუნქციურ მოთხოვნილებათა განსაზღვრა;

- ელექტრონული საარჩევნო სისტემის მონაცემთა განაწილებული ბაზების სტრუქტურების დასაპროექტებლად ობიექტ-როლური მოდელების (ORM) აგება და კვლევა რევერსიული CASE ტექნოლოგიების გამოყენებით;

- ელექტრონული საარჩევნო სისტემის ფუნქციონირების დროს, ამომრჩეველთა ნაკადების კვლევა ფერადი პეტრის ქსელების (CPN) გამოყენებით;

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

ნაშრომის კვლევის ობიექტია ელექტრონული საარჩევნო სისტემა, მულტიმედიური მონაცემთა ბაზები და შესაბამისი მართვის საინფორმაციო სისტემები.

კვლევის მეთოდების სახით განიხილება: სისტემური ანალიზის მეთოდი; მონაცემთა განაწილებული ბაზების თეორია; მულტიმედიური მონაცემთა ბაზების თეორია; ობიექტ-ორიენტირებული მოდელირების, ანალიზის და პროექტირების მეთოდები; უნიფიცირებული მოდელირების ენა და მისი რეალიზაციის ინსტრუმენტები; პეტრის ქსელების მათემატიკური მოდელი; იმიტაციური მოდელირება ფერადი პეტრის ქსელებით; ობიექტორიენტირებული დაპროგრამების თეორია; ობიექტ-როლური მოდელირება. კლიენტ-სერვერული არქიტექტურა; სერვისორიენტირებული არქიტექტურა.

ნაშრომის ორიგინალობა მდგომარეობს ელექტრონული საარჩევნო სისტემის, როგორც დიდი და რთული საინფორმაციო სისტემის პროგრამული უზრუნველყოფის კომპლექსურ შემუშავებაში ახალი ინფორმაციული ტექნოლოგიების ბაზაზე. კერძოდ:

1. აიგო მულტიმედიური ელექტრონული საარჩევნო სისტემის მოდელები პროცესორიენტირებული ტექნოლოგიების გამოყენებით და ჩატარდა მათი შედარებითი ანალიზი თანამედროვე და ტრადიციულ მოდელებთან;

2. შემუშავდა მულტიმედიური საიმედო მონაცემთა ბაზების აგების ტექნოლოგია. ასეთ ბაზებში განთავსდება დამატებითი დაცვის იდენტიფიკატორების ინფორმაცია – თითის ანაბეჭდები, ელექტრონული ხელმოწერები, ბიომეტრული სურათები, ამომრჩევლის ხმა.

3. პირველად ნაშრომში ავტომატიზებულ რეჟიმში დაპროექტდა საპრობლემო სფეროს, ელექტრონული საარჩევნო სისტემის კონცეპტუალური და მონაცემთა განაწილებული

რელაციური ბაზის ლოგიკური და ფიზიკური მოდელები კატეგორიული მიდგომისა და ობიექტროლური მოდელირების გამოყენებით;

4. ელექტრონული საარჩევნო სისტემისათვის აიგო პეტრის ქსელის მათემატიკური (იმიტაციური) მოდელი, რომლის საფუძველზე ჩატარდა საარჩევნო სისტემის დინამიკური პროცესების კველვა.

მიღებულ შედეგებს აქვს პრაქტიკული ღირებულება, კერძოდ, ის შეიძლება გამოყენებულ იქნას სახელმწიფოს მიერ (ელექტრონული მთავრობის სისტემაში) და მოხდეს მისი დანერგვა შესაბამის საკანონმდებლო ბაზაში ცვლილებების შეტანის შემდეგ.

ნაშრომის ძირითადი შინაარსი მოხსენებული იყო სამ საერთაშორისო სამეცნიერო–ტექნიკურ კონფერენციაზე (2011-2013 წლებში): 1. Intern. Sc.Conf.: “Automated Control Systems & new IT” - სტუ-2011, 2. Application of information and Communication Technologies (PCI’ 2012) - თსუ-2012, 3. International Scientific and Practical Conference - ქუთაისი-2013, აგრეთვე ინფორმატიკისა და მართვის სისტემების ფაკულტეტის „მართვის ავტომატიზებული სისტემების“ დეპარტამენტის სამეცნიერო სემინარების სხდომაზე, დაცულია აკადემიური დოქტორის ხარისხის მისანიჭებელი დისერტაცია.

წიგნის **პირველი თავი** ეხება ელექტრონული საარჩევნო სისტემის დახასიათებას. იგი განიხილება, როგორც მართვის რთული და დიდი სისტემა. გადმოცემულია SQL Server-ის გამოყენებით მონაცემთა ბაზების აგების მეთოდები და საშუალებები. ყურადღება გამახვილებულია ისეთი მულტიმედიური მონაცემების აუთენტიფიკაციაზე, როგორიცაა თითის ანაბეჭდები, ბიომეტრული სურათები, ელექტრონული ხელმოწერები, თვალის ბადურის სკანირება, ხმის აუდიო ჩანაწერი და ა.შ.

განხილულია VPN ტექნოლოგია, როგორც უსაფრთხო და დაცული ქსელის აგების საშუალება. მიმოხილულია ოპერაციული სისტემები და მათი სერვერული ვერსიებისათვის გავრცელებული სერვისები. გადმოცემულია ზოგადი ინფორმაცია ბიზნეს-პროცესების მოდელირების და მართვის (BPMN), ობიექტროლური მოდელირების (ORM) და ბიზნესპროცესების მოდელირების, ანალიზისა და შეფასების ინსტრუმენტი ფერადი პეტრის ქსელების შესახებ.

მეორე თავში განიხილება საარჩევნო სისტემების ელექტრონული და ტრადიციული მეთოდები. კერძოდ, შემუშავებულია მართვის მოდელები მულტიმედიური მონაცემთა ბაზების ასაგებად და სახელმწიფო უსაფრთხო ქსელის პროექტირებისა და აგების არქიტექტურა და გეგმა.

საარჩევნო სისტემებისათვის აგებულია ბიზნესპროცესების მოდელები (BPMN) როგორც ტრადიციული, ასევე ელექტრონული. გამოკვლეულია მიზეზშედეგობრივი პროცესების შესრულების დროის მახასიათებლები პეტრის ფერადი ქსელების გამოყენებით (CPN). ელექტრონული საარჩევნო სისტემის მონაცემთა განაწილებული მულტიმედიური ბაზების სტრუქტურის დასაპროექტებლად შემუშავებულია ობიექტ-როლური მოდელი (ORM). ჩატარდა კვლევა რევერსული CASE ტექნოლოგიის გამოყენებით.

განხილულია ელექტრონული საარჩევნო სისტემისათვის საჭირო ტექნიკური უზრუნველყოფის საშუალებები და მათი შეფასებები. მოცემულია ვირტუალური კერძო ქსელის (VPN) დაპროექტება, აგება და უსაფრთხოების ნორმები. განხილულია ელექტრონული საარჩევნო სისტემის ფუნქციონირების ორგანიზების ასპექტები.

ნაშრომის **მესამე თავი** წარმოადგენს მონაცემთა მულტიმედიის მოდელების ანალიზის და მულტიმედიური მონაცემთა ბაზების შედგენილობის და სტრუქტურის ელემენტების დახასიათებას, განიხილება მონაცემთა ახალი ტიპები მულტიმედიური სისტემებისთვის, მათ საფუძველზე კი - განზოგადებულ კლასთა იერარქიის აგების ამოცანა. საერთაშორისო ISO/IES 13249 სტანდარტების საფუძველზე ამ სფეროში, წარმოდგენილია SQL / MM ნორმებით განსაზღვრული მონაცემთა მულტიმედიური ტიპები, როგორიცაა ტექსტები, სივრცითი გამოსახულებები, სურათები და სხვა. დახასიათებულია ობიექტრელაციური და ობიექტორიენტირებული მულტიმედიური მონაცემთა ბაზების სისტემები, მათი შედარებითი ანალიზი. შემოთავაზებულია რელაციური მონაცემთა ბაზის ოპტიმალური სტრუქტურის დაპროექტების ავტორისეული ალგორითმული გადაწყვეტის შედეგები.

მეოთხე თავში გადმოცემულია ელექტრონული საარჩევნო სისტემისათვის დამუშავებული პროგრამული უზრუნველყოფის სტრუქტურა, მისი ინტერფეისული და კოდური ნაწილები. შემოთავაზებულია SQL Server-ის ბაზაზე აგებული მონაცემთა ბაზების ფაილები. აღწერილია მომხმარებელთათვის შექმნილი ინსტრუქციები და პროგრამული უზრუნველყოფის მოხმარების წესები. გადმოცემულია ელექტრონული საარჩევნო სისტემის შედეგების ანალიზი და მისი გამოყენების სამომავლო პერსპექტივები. განიხილება შედეგების საბოლოო სინქრონიზაცია საქართველოს სამოქალაქო რეესტრის მონაცემთა ბაზებთან.

წიგნის **მეხუთე თავი** ეხება საინფორმაციო სისტემის პროგრამულ რეალიზაციას მულტიმედიურ მონაცემთა ბაზებით და მისი ექსპლუატაციის მხარდაჭერას სერვისორიენტირებული არქიტექტურით. მაიკროსოფთის ახალი ინფორმაციული ტექნოლოგიების WPF, WF და WCF საფუძველზე, რომლებიც

გამოიყენება ჰიბრიდული საინფორმაციო სისტემების ასაგებად (Windows და Web პროექტები), შემუშავებულია საპილოტო ვერსიის პროგრამული რეალიზაცია კლიენტ-სერვერ არქიტექტურით. წარმოდგენილია პროგრამული კოდების დამუშავების ტექნოლოგია C#.NTET და XAML ენების ბაზაზე.

დასასრულ, გვინდა აღვნიშნოთ, რომ მონოგრაფიის ძირითადი საკითხები, როგორიცაა მულტიმედიური მონაცემთა ბაზების აგების პრონციპები, ახალი კომპიუტერული ტექნიკის და ინფორმაციული ტექნოლოგიების ეფექტური გამოყენება, პროცეს-ორიენტირებული და სერვისორიენტირებული მიდგომების საფუძველზე, წლების განმავლობაში მუშავდებოდა ავტორების მიერ როგორც საქართველოს ტექნიკურ უნივერსიტეტში, ასევე გერმანიაში, კერძოდ, ერლანგენ-ნიურნბერგის უნივერსიტეტში („მონაცემთა მენეჯმენტის“ კათედრაზე, რომლის გამგეცაა ჩვენი თანაავტორი, პროფესორი კლაუს მეიერ-ვეგენერი). აქვე 2014 წლის იანვარში მოხდა წინამდებარე ნაშრომის ბოლო ვერსიის განხილვა ჩვენი კათედრების საერთო სამეცნიერო სემინარზე.

ვფიქრობთ, რომ წიგნში ასახული მასალა საინტერესო იქნება ჩვენი კოლეგების, სტუდენტების და მკითხველისთვის.

მზად ვართ მოვისმინოთ და განვიხილოთ მათი შენიშვნებიც, რამეთუ სოციალური სისტემების საინფორმაციო ტექნოლოგიების შექმნა აუცილებლად მოითხოვს მუდმივ განვითარებას და სრულყოფას.

ავტორები

I თავი

მულტიმედიური მონაცემთა საცავების დაპროექტების პრობლემები და მათი გადაჭრის გზები ელექტრონული საარჩევნო სისტემის ასაგებად

1.1. კვლევის ობიექტის დახასიათება, როგორც მართვის რთული და დიდი სისტემა

დემოკრატიული სახელმწიფოს მშენებლობისა და სამართლიანი საზოგადოების ჩამოყალიბების პროცესები ბევრადაა დამოკიდებული მრავალ ობიექტურ და სუბიექტურ ფაქტორზე, რომელთაგან ერთ-ერთი მთავარი ქვეყნის საარჩევნო სისტემის რეფორმაა.

იგი განსაკუთრებით მნიშვნელოვანია ჩვენი ქვეყნისთვისაც, რომელიც ფართო პოლიტიკური სპექტრითა და მოსახლეობის მაღალი უნდობლობის ფაქტორებით ხასიათდება. აქტუალურად მიგვაჩნია როგორც პოლიტიკური, ასევე ეკონომიკური და ტექნიკურ-ტექნოლოგიური გადაწყვეტილებანი ამ სფეროს პრობლემების გადასაჭრელად.

განვიხილოთ ცნება „E-voting” – „ელექტრონული საარჩევნო სისტემა”, რომელიც საშუალებას გვაძლევს ჩატარდეს რეფერენდუმები და არჩევნები ელექტრონული მოწყობილობების და სისტემის მხარდაჭერით. არსებობს რამდენიმე სახის ელექტრონული საარჩევნო სისტემა. ესენია:

1. სისტემა, რომელიც არ მოითხოვს საარჩევნო უბნის ინტერნეტში ან სხვა რომელიმე ქსელში ჩართვას. ამ შემთხვევაში ინფორმაციის ჩაწერა ხდება ეგრეთ წოდებულ „პირდაპირ

ელექტრონულ ჩამწერში’’ და არ გადასცემს ინფორმაციას ფიზიკური მიერთების გარეშე.

2. სისტემა, რომელიც საჭიროებს საარჩევნო უბნის ინტერნეტში ჩართულობას და ამომრჩევლის არჩევანს გზავნის ცენტრალური საარჩევნო კომისიის დიდ მონაცემთა საცავში. ასეთი ტიპის სისტემები იყენებს ჩვეულებრივ პერსონალურ კომპიუტერებს ან მობილურ ტელეფონებს (Smart phones).

3. სისტემა, რომელიც არ საჭიროებს საარჩევნო უბნის მოწყობას. ამ შემთხვევაში ამომრჩეველს შესაძლებლობა აქვს, მსოფლიოს ნებისმიერი წერტილიდან მისცეს ხმა მისთვის სასურველ კანდიდატს ინტერნეტის და დაშვების კოდის (Access code) მეშვეობით.

უნდა აღინიშნოს, რომ მესამე სისტემას ჰყავს მრავალი მოწინააღმდეგე, რადგან ვერ ხერხდება თავად ამომრჩევლის სრული იდენტიფიცირება და უსაფრთხოების ნორმების სრული დაცვა. ყველა თანხმდება იმაზე, რომ აუცილებელია ხმის მიცემა განხორციელდეს მხოლოდ და მხოლოდ საარჩევნო უბანზე დადგენილი წესების სრული დაცვით, რაც ჩვენ თვალსაზრისსაც ემთხვევა.

მრავალ ქვეყანაში განიხილება არსებული საარჩევნო სისტემების ელექტრონული საარჩევნო სისტემით ჩანაცვლების სხვადასხვა მოდელი და შესაძლებლობა. ყველა თანხმდება ელექტრონული საარჩევნო სისტემის მიმართ შემდეგ მოთხოვნებზე:

ელექტრონული საარჩევნო სისტემა უნდა იყოს თავისუფალი, ხმის საიდუმლოების დამცველი. საიდუმლო და დაცულ სისტემასთან წვდომა შეუძლებელი იქნება ელექტრონული თვალსაზრისით. ამ მიმართულების დახვეწაზე მუშაობს მრავალი მოწინავე ქვეყანა და ორგანიზაცია: ავსტრალია,

ავსტრია, ბელგია, ბრაზილია, კანადა, ესტონეთი, ევროკავშირი, საფრანგეთი, გერმანია, ინდოეთი, ირლანდია, ნორვეგია, პორტუგალია, ესპანეთი, შვეიცარია, ჰოლანდია, დიდი ბრიტანეთი, აშშ და სხვები.

მაგალითად, აშშ-ში დაიქირავეს მსოფლიოში ცნობილი ჰაკერები, რომლებსაც დაევალათ იმ ხვრელების აღმოჩენა, საიდანაც შესაძლებელი გახდებოდა საარჩევნო სისტემაში შეღწევა და მისი განადგურება ან რაიმე სახის ზემოქმედების მოხდენა. შედეგებით აშშ-ს ადმინისტრაცია კმაყოფილი დარჩა, რაც უახლოეს მომავალში აისახება არსებულ საარჩევნო სისტემაზე ელექტრონულის ჩანაცვლებით.

1.2. მონაცემთა საცავების სტრუქტურა და შედგენილობა მულტიმედიური ელემენტების გამოყენებით

მონაცემთა საცავის (Data Warehouse-DWH) იდეა, რომელიც წარმოადგენს მართვის საინფორმაციო სიტემების დაპროექტების და რაეალიზაციის ერთ-ერთ უახლეს ტექნოლოგიას, აქტუალური გახდა 90-იანი წლებიდან. ამ იდეის ფუძემდებლად მოიხსენიება ამერიკელი მეცნიერი ვ.ჰ. ინმონი.

მონაცემთა საცავი აღიწერებოდა, როგორც „მონაცემთა სუპერმარკეტი”, „სუპერ მონაცემთა ბაზა”. ამ მიმართულებით პირველი პროექტი „ევროპის ბიზნეს ინფორმაციული სისტემა” 1988 წელს IBM ფირმის მიერ განხორციელდა. გლობალურ მონაცემთა შენახვა და მათი ანალიზი სავაჭრო კომპანიების მუშაობისას არა მარტო ამაღლებს მუშაობის ეფექტურობას, არამედ უძლებს მკაცრ კონკურენციას, რომელიც საერთაშორისო ბაზარზე ყოველწლიურად ძლიერდება.

მონაცემთა საცავის ფართო გამოყენება, მსოფლიო ბიზნეს-გაერთიანებაში ადასტურებს იმ ფაქტს, რომ ამ ტექნოლოგიაზე დაყრდნობით მოგება ყოველწლიურად იზრდება ათობით მილიარდი დოლარით. ბანკებში ჩატარებულმა ანალიზმა აჩვენა, რომ საბაზო ტრანზაქციების და სასურველ მონაცემებზე არსებული ინფორმაციის მოცულობა ძალიან დიდია. პირველადი ანალიზისათვის უნდა მომზადდეს გაფართოებული მონაცემები და მიეცეს ინდექსაცია. ამისათვის საჭიროა ფართო კომპიუტერული რესურსები, რომელიც საშუალებას იძლევა მცირე დროის განმავლობაში შესრულდეს ძებნა რამდენიმე ცხრილში, რომელიც ათობით მილიონ ჩანაწერს შეიცავს და განხორციელდეს მონაცემთა შერჩევა.

მონაცემთა საცავი არის კომპლექსური სისტემა, რომელიც შედგება შემდეგი ძირითადი ფუნქციური ბლოკებისაგან:

- მონაცემთა განაწილებული, რელაციური ბაზების მართვის სისტემები;

- ინფორმაციის წყაროებიდან ოპერატიულ მონაცემთა ჩატვირთვის და გარდაქმნის საშუალება;

- საცავის დაპროექტების მეთოდური და ინსტრუმენტული საშუალებანი;

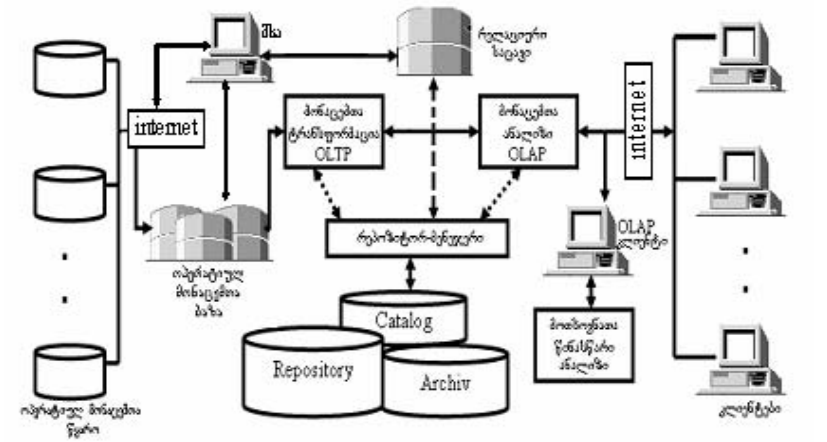
- საცავის აგებისა და მოდიფიკაციის საშუალებანი;

- საცავის მეტამონაცემთა იერარქიული ორგანიზების ჰიპერლინკური საშუალებანი;

- საცავის ფუნქციურ მომხმარებელთა მოთხოვნების წინასწარი ანალიზისა და ტრანზაქციების ეფექტურად შესრულების დაგეგმვის საშუალებანი;

- საცავის საინფორმაციო ბლოკებისა და არქივის ოპერატიული ანალიზის ინსტრუმენტული საშუალებანი.

1.1 ნახაზზე მოცემულია განაწილებული ავტომატიზებული მართვის სისტემის მონაცემთა საცავის ზოგადი სქემა [10].



ნახ.1.1. მონაცემთა საცავის ზოგადი სქემა

მონაცემთა საცავის მუშაობის პრინციპი ასეთია: პირველ ეტაპზე DWH-ს ტექნოლოგიის გამოყენების საშუალებით, რელაციურ ბაზებში ერთად თავმოყრილი მონაცემები ლაგდება გარკვეული სტრუქტურული თანამიმდევრობით, ხდება მათი „დაწმენდა”. მეორე ეტაპზე წარმოებს მათი ტექნოლოგიური დამუშავება მონაცემთა ოპერატიული ანალიზის OLAP-ტექნოლოგიის გამოყენებით. ხოლო მესამე ეტაპზე ეს მონაცემები მომხმარებლებს მიეწოდებათ ინტერნეტის საშუალებით.

მონაცემთა საცავის ცნება აქტუალური გახდა 2000 წლიდან და იგი დღემდე, საინფორმაციო სიტემების დაპროექტებაში, პერსპექტიულ მიმართულებად ითვლება. საწყის ეტაპზე საინფორმაციო სისტემების დაპროექტება ვითარდებოდა „ინფოლოგიკური“ გზით, ხოლო ახალი მიმართულების განვითარების ტენდენციამ ფართო გამოვლინება პოვა „მონაცემთა საცავის“ სახით.

ინფორმატიკაში როგორც ბევრი ტერმინი, ასევე ეს ცნებაც მოკლებულია ზუსტ განმარტებას, რაც პრაქტიკაში წარმოქმნის შეუთავსებლობას. არსებობს განმარტების მრავალგვარი ახსნა, რომელთაგან ერთ–ერთი ასეთია:

მონაცემთა საცავი არის რომელიმე ორგანიზაციისთვის განკუთვნილი სპეციალური ბაზა, სადაც მიმდინარე ოპერატიული სამუშაოს შესრულებისას თავს იყრის ქრონოლოგიურ ინფორმაციათა მთელი სპექტრი, რომელთა დანიშნულებაცაა მომხმარებელს მიაწოდოს ინტერნეტ გვერდებზე მიზნობრივად განლაგებული საინფორმაციო ბლოკები.

საავტომატიზაციო ობიექტის განაწილებული სისტემის მონაცემთა საცავში განსათავსებელი საინფორმაციო ბლოკები შეივსება ოპერატიულ მონაცემთა ბაზებიდან, მათი წინასწარი დამუშავების საფუძველზე (მეთოდური საფუძვლები წინა პარაგრაფში იყო განხილული). როგორც ვიცით, ოპერატიულ ბაზაში საწყისი მონაცემები თავს იყრის გარე ორგანიზაციებიდან ინტერნეტის საშუალებით ან სხვა სახის კომუნიკაციებიდან.

საინფორმაციო ბლოკები ან, უფრო ზოგადად, მონაცემთა მანქანური ფაილები განსხვავებული ტიპებისაა: ტექსტები და სუპერტექსტები (H), ცხრილები (T), გრაფიკები (G), აუდიო (A) და ვიზუალური (V) მასალა და ა.შ.

მათემატიკური მოდელი შეიძლება ჩავწეროთ ტეტრადით:

$$B = < R, M, S, F >, \text{ სადაც}$$

R საცავის კლასიფიცირებული ობიექტებია.

R = {}, . აქ i კლასიფიკაციის ტიპია და იგი შეესაბამება ზემოაღნიშნულ ცვლადებს: {H, T, G, A, V}.

მონაცემთა საცავებთან ურთიერთობის და მათი გამოყენების თვალსაჩინო მაგალითია საძიებო სისტემები.

ინფორმაციის საძიებო სისტემების ძირითადი დანიშნულება არის დოკუმენტების მოძიება, რომლებიც სწორედ მონაცემთა საცავებშია ასახული. ტრადიციულად, დოკუმენტებში იგულისხმება ტექსტური დოკუმენტები. თუმცა დღეისათვის გაძლიერებულია ეს მოთხოვნა და იგი გადაიზარდა მულტიმედიალური დოკუმენტების (სურათები, ვიდეოების, აუდიო და ჰიპერტექსტური დოკუმენტები) ძიებაში ან მაგალითად სამუშაო და ექსპერტული ჯგუფების ძიება განსაზღვრულ კომპეტენტურ ჭრილში.

ამასთანავე გაიზარდა ინფორმაციის საძიებო სისტემების არეალიც და ყველაზე აქტუალური დღეისათვის გახლავთ ინტერნეტი (World Wide Web). გაჩნდა აუცილებლობა იმისა, რომ იმ უდიდესი მოცულობის მონაცემებიდან ეფექტურად (სწრაფად) და მიზანმიმართულად (ამომწურავი ინფორმაცია) მოიძიო ესა თუ ის საჭირო დოკუმენტი.

ამასთანავე კონკრეტული ასახვა ძიებაში შეიძლება იყოს შემდეგი სახით კლასიფიცირებული:

1. რომელ ინფორმაციულ მასივში იძებნება ესა თუ ის დოკუმენტი ?

თავისთავად გამორჩეული და განსხვავებულია ინტერნეტის მართულ მონაცემთა მასივებში ძიება. განსხვავება მდგომარეობს ინფორმაციის მოცულობასა და მის ჰეტეროგენულობაში.

დიდ დროს და ძალისხმევას მოითხოვს ინტერნეტში მოძიებული სხვადასხვა ავტორის, სხვადასხვა ხარისხის და სხვადასხვა გაფართოების (ფორმატი) დოკუმენტების გადახარისხება.

ამისათვის შეიქმნა ბევრად უკეთ მართული და კონკრეტულ საქმიანობაზე მორგებული - ინტრანეტი, რათა კონტროლის, ხარისხისა და გაფართოების გათვლისწინებით შესამჩნევი ყოფილიყო მასში ძიების ეფექტურობა.

2. რა ტიპის დოკუმენტი იძებნება ?

ძირითადად ხდება ტექსტური დოკუმენტების ძიება, თუმცა სხვა გაფართოების დოკუმენტებიც არსებობს, რომელიც არანაკლებ საჭირო ინფორმაციის შემცველი შეიძლება იყოს. მაგალითად: HTML–გაფართოების მონაცემები, PDF-გაფართოება და ა.შ. ასევე შესაძლებელია მოძიებული იქნას სურათები, ვიდეო- ან აუდიო-მასალა.

3. რამდენად კარგად არის ახსნილი სასურველი დოკუმენტი საძიებელ სიტყვაში?

არსებობს შემთხვევები, როდესაც მაძიებელმა დაზუსტებით იცის რას ეძებს, ხოლო ასევე არსებობს შემთხვევები, როდესაც სასურველი დოკუმენტის ზუსტად ასახვას ვერ ახერხებს ესა თუ ის პირი და ამიტომაც ზოგჯერ არასასურველ შედეგს ვიღებთ ხოლმე.

მოსაზრებები ფუნქციური MMDBVS-ის შესახებ გარკვეული დროის წინ გაჩნდა. ყველაზე ადრეული შეხედულებები, რაც გამოქვეყნდა ამ საკითხთან მიმართებით შედგებოდა მხოლოდ ორი გვერდისაგან, რომელიც სტავროს ქრისტოდულაკისმა, წარმოშობით ბერძენმა მეცნიერმა სადისკუსიოდ SIGMOD-ის კონფერენციაზე 1985 წელს გამოიტანა. ამ საკითხის წამოწევით ორი გადასაჭრელი ამოცანა დაისვა:

- არაფორმატირებული მონაცემების მენეჯმენტი;
- არაფორმატირებული მონაცემების მონიტორინგის და ჩაწერა-შენახვის მოწყობილობების მართვა.

აღნიშნული საკითხების გადასაჭრელად არ არსებობდა საკმარისი არც გამოცდილება და არც ცოდნა. ერთი კი ფაქტი იყო, რომ მედია მონაცემები უნდა ჩაწერილიყო და შენახულიყო.

სტავროს ქრისტოდულაკისის მტკიცებით, ეჭვს აღარ იწვევდა, რომ უნდა შექმნილიყო მონაცემთა საცავები MMDBVS, სადაც მოხდებოდა არა მხოლოდ ფორმატირებული მონაცემების

შენახვა, არამედ არაფორმატირებულისაც კი. მან განავითარა აღნიშნული პრობლემა და ჩამოაყალიბა მათი გადაჭრისათვის აუცილებელი რამდენიმე პუნქტი:

1. პროგრამული უზრუნველყოფის არქიტექტურა:

პროგრამული უზრუნველყოფის არქიტექტურა შეიძლება სხვადასხვა გზით განხორციელდეს იმისათვის რომ MMDBMS რეალიზაცია მოხდეს. ერთი მხრივ, ეს არის არსებული DBMS განგრძობა-გავრცობა, რომელიც მიბმული იქნება მასივებში შეტანილ ცვლილებებთან და ახლადიმპლემენტირებულ ბაზასთან მოვა თანხვედრაში. მეორე მხრივ, შეიძლება მედია სპეციფიკის მქონე სპეციალური DBMS და სტანდარტული DBMS-სათვის საერთო კომბინირებული ინტერფეისის შექმნა. შესრულებისას არსებობს გარკვეული პრობლემები, რადგან ყველა განსაკუთრებით გავრცელებული ფუნქცია, რომლებიც მედია მონაცემებსა და ფორმატირებულ მონაცემებს ეხება, შესაძლებელია მხოლოდ რეალიზებული იქნას ინტეგრაციულ დონეზე. ყოველივე ეს კი საბოლოოდ შესაძლებელს ხდის შევქმნათ სრულიად ახალი DBMS, რომელიც უმსხვილეს ხარჯებთან არის დაკავშირებული, მაგრამ სრული ოპტიმიზაციის საშუალებას მოგვცემს.

2. შემადგენლობის დამისამართება:

ინფორმაციის ძებნა ყოველთვის ხორციელდება სწორად ფორმატირებულ მონაცემებში ან ტექსტებში, რაც ნამდვილად არ არის რეალური პრობლემა. მედია ობიექტების თვისებები და სტრუქტურები თავად იპყრობს ყურადღებას და კვალიფიკაციას მოითხოვს ზომების მსგავსების გამო. ამიტომ ავტომატური კონტენტის ანალიზი პრაქტიკულად შეუძლებელია, თუმცა შეიძლება ძალიან განსხვავებული გზის ე.წ. „თვისებების" გამოყენება და ძებნა მისი მეშვეობით.

3. შესრულება (ეფექტურობა):

უნდა არსებობდეს ახალი პირობების დაშვება და მეთოდები ახალი მონაცემთა ტიპებისათვის (მაგალითად, ელექტრონული ხელმოწერა, ბიომეტრული სურათი, თითის ანაბეჭდები და ა.შ.), ტექნიკური მოწყობილობების არქიტექტურა, ფიზიკურ მონაცემთა ბაზის დიზაინი და მოთხოვნათა ოპტიმიზაციის დროს გათვალისწინებული უნდა იქნას ახალ მონაცემთა ტიპები, მათი ზომა და განსაზღვრული ოპერაციები.

4. ინტერფეისი:

როდესაც მომხმარებლისათვის დამუშავდება საბოლოო ინტერფეისი MMDBVS-ისათვის, მას უნდა შეეძლოს ფუნქციონირება სხვადასხვა მოწყობილობებთან, რომლებიც საჭირო გახდება სხვადასხვა მედია მონაცემების შეტანა-გამოტანის ოპერაციის შესასრულებლად.

5. ოპერაციები:

რამდენადაც კარგად არის შესაძლებელი, MMDBVS ოპერაციებმა უნდა შესთავაზოს მომხმარებელს ინფორმაციის მოპოვების შესაძლებლობა მედიაობიექტებიდან და მათი წარმოდგენა ფორმატირებული ფორმით. ასევე უნდა გაგრძელდეს ოპერაციები მედია-გარდაქმნა ინსტრუქციის მიხედვით.

6. მრავალმომხმარებლიანი სისტემა, აღდგენა, დაშვების კონტროლი, მხარდაჭერის ვერსიები:

რაც განასხვავებს ტრადიციულ DBMS მარტივი ფაილური სისტემებისაგან, ასევე უნდა იყოს ხელმისაწვდომი MMDBVS. ძირითადი განმასხვავებელი საკითხი არის სინქრონიზაციის შემდგომ გრანულებში და მათზე დაშვების მართვაში. აქედან

გამომდინარე ენიჭება მომხმარებელს უფლება, შეცვალოს ექსკლუზიური მედიაობიექტები თუ მხოლოდ მისი ნაწილები.

7. დიდი მოცულობის შემნახველი მოწყობილობები:

მართვის ახალი შენახვის მოწყობილობები, განსაკუთრებით ოპტიკური დისკები უდავოდ ამოცანაა MMDBVS-სათვის. ასეთი მოწყობილობები მოიცავს ასლების გაკეთების თანმიმდევრულ კონტროლს. საჭირო და აუცილებელია მოვახდინოთ მათი გამოყენებისას შეკუმშვის მეთოდების ჩართვა, რათა დაზოგილ იქნას სივრცე შემნახველ მოწყობილობაზე.

8. ინფორმაციის ძიების მეთოდები და ტექნიკა:

გასაღებური სიტყვის გამოყოფა და ძიება სრულ ტექსტში შეიძლება ძალიან სასარგებლო მეთოდი იყოს. ეს მოითხოვს, რომ აპრობირებული, ინფორმაციის საძიებელი ცნობილი ტექნიკა შეიძლება ინტეგრირებული იქნას ჩვეულებრივ DBMS-ში, როგორც ეს აქამდეც ხდებოდა.

9. გამოყენებადი პროტოტიპები:

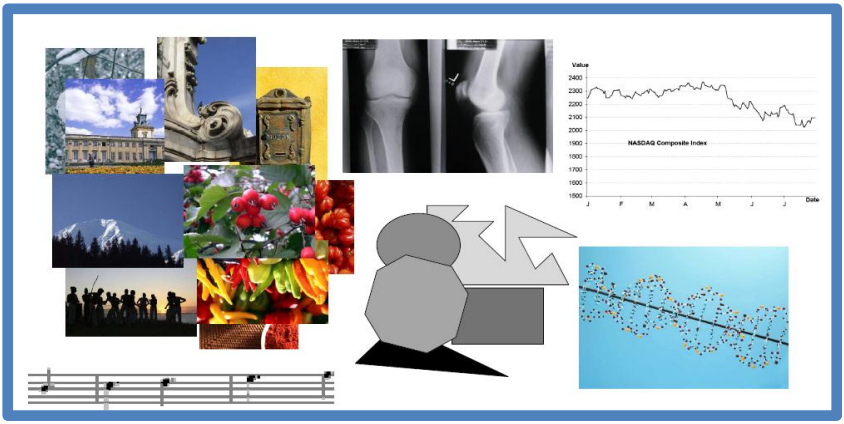
წამყვანი პროტოტიპები უნდა განხორციელდეს სწრაფად, რითაც შეიძლება შეგროვდეს გამოცდილება.

ბოლო წლებისათვის მულტიმედია (ტექსტის, გრაფიკული მონაცემის, სურათის, ვიდეოს და აუდიოს კომბინაცია) გახდა კომპიუტერული მეცნიერებისათვის ცენტრალური კვლევის საგანი.

სწრაფი ტექნოლოგიური პროგრესი მულტიმედიური მონაცემების წარმომქმნელი მოწყობილობების (ციფრული კამერები) და მულტიმედია მონაცემების მომხმარებელი მოწყობილობები (DVD პლეერები) განაპირობებს მულტიმედია მონაცემების რაოდენობის დიდად გაზრდას. ამ საქმეში დიდ როლს თამაშობს ინტერნეტიც.

რას ნიშნავს მულტიმედია მონაცემი ?

ეს არის მონაცემთა ტიპი, ბლოკების შემადგენელი ელემენტები, რომლებიც ახასიათებს და აღწერს ბუნებაში მომხდარ მოვლენებს. მულტიმედია მონაცემების ტიპებია: ტექსტური, გრაფიკული, სურათი, ვიდეო, აუდიო. განვიხილოთ თითოეული



მათგანი დეტალურად (ნახ.1.2).

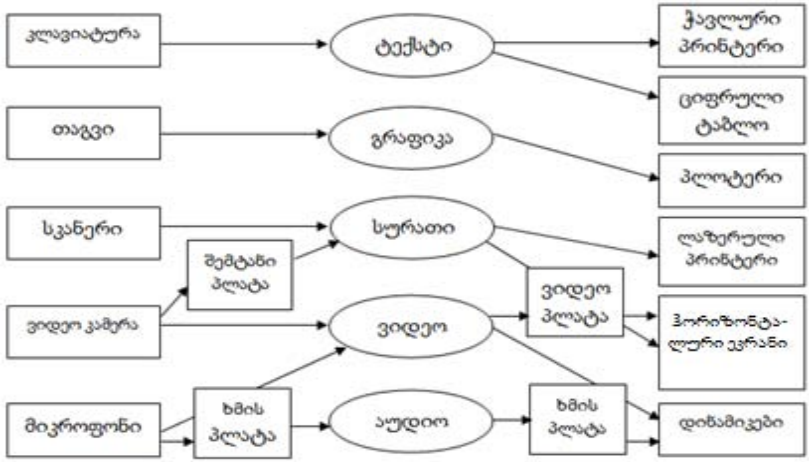
ნახ.1.2. მულტიმედია მონაცემების ტიპები

- **ტექსტური ინფორმაცია:** როდესაც ვსაუბრობთ ტექსტური ფაილებზე, გვახსენდება „ASCII” კოდირება. ამ ფორმის დოკუმენტების შენახვა და მართვა ხდება ძალიან კარგად და მარტივად მონაცემთა ბაზებში.
- **გრაფიკული ინფორმაცია:** ამ ტიპის მონაცემები გამოისახება სპეციალური სტრუქტურით და აგებულებით 2D, 3D, 4D გაფართოებით. იგი მოიცავს სხვადასხვა ფორმატებს, მაგალითად CAD და CAM.
- **სურათი:** უამრავი ვარიანტია სურათების შემთხვევაში, რაც გამოისახება სურათის ხარისხში, ზომაში, კომპლექსურობაში,

შეკუმშულობაში. იგი შეიძლება წარმოდგენილი იქნას შემდეგ ფორმატებში: jpg, png, bmp, tiff.

- **ვიდეო:** ერთ-ერთი ყველაზე დიდი და შრომატევადი მულტიმედია მონაცემთა ტიპია ციფრული ვიდეო. ციფრული ვიდეო ინახება გარკვეული თანმიმდევრული წყობით, მისი რეზოლუციის და ზომის გათვალისწინებით 1 მბ. ზომამდე. უდიდესი მიღწევა იყო მულტიმედია მონაცემების სტანდარტიზაცია და შეკუმშვა [MPEG-1/2/4 (AVC), JPEG (ეს დასახელება მიანიჭეს 1992 წელს ISO-ნორმით), MP3]. მულტიმედია მონაცემების შეკუმშვამ გამოიწვია ინფორმაციის მოცულობის დიდად შემცირება.
- **აუდიო:** მზარდად პოპულარული მონაცემების ტიპი არის აუდიო მონაცემები. ისიც საკმაო მოცულობითია. მაგალითად, ერთი წუთი ხმა შესაძლებელია იყოს 2-3 მბ. შესაძლებელია მისი შეკუმშვაც.

1.3 ნახაზი ასახავს მულტიმედია ინფორმაციის შეტანა-გამოტანის საშუალებებს.



ნახ.1.3.

რამდენად განსხვავებულია მულტიმედია ინფორმაცია?

კონცეპტუალურად შესაძლებელი უნდა იყოს მულტიმედია მონაცემების იმავე გზით შენახვა და დაარქივება, როგორც ნებისმიერი სხვა მონაცემის (რიცხვების, თარიღების, სიმბოლოების). თუმცა აქ არის რამდენიმე გამოწვევა, რომელიც წარმოდგება მულტიმედია მონაცემებიდან.

სწორედ ახლა განვიხილავთ მედიაობიექტებს და მონაცემებს, რომელთა სამართავადაც გამოვიყენებთ მულტიმედია მონაცემთა ბაზებს. მულტიმედია საგნის აღწერილობა მოითხოვს მის შესანახ ფორმატში გადაყვანას, კოდირებას და სტანდარტიზაციას.

ზოგი მედიაობიექტისათვის არსებობს მზა სამართავი ან არქივაციის სისტემა, ხოლო ზოგისთვის – არა. მათ შორის აღსანიშნავია და დიდ ინტერესს იწვევს დაჭერის ოპერაციები, რომელთა შემდგომი გამოყენება ხდება მულტიმედია მონაცემთა ბაზების მმართველ სისტემებში.

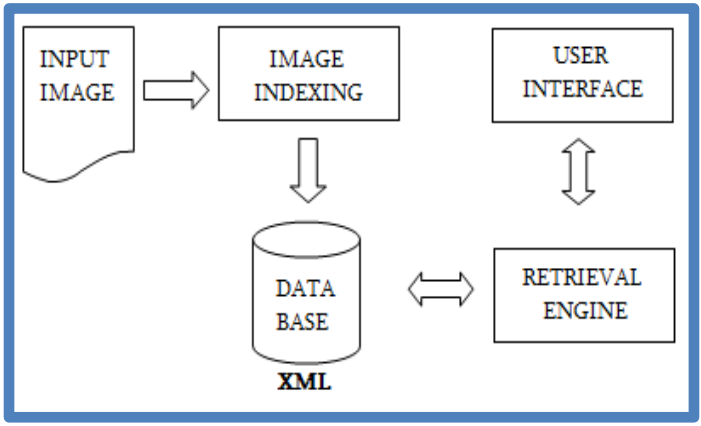
მონაცემთა მოძიების ძირითადი მიდგომები:

მონაცემთა მენეჯმენტს აქვს დიდი ხნის ისტორია და მრავალი მიდგომა, რაც გამოისახება კომპიუტერულ სისტემებში საკითხების დივერსიფიკაციისას მონაცემთა ტიპების მიხედვით. მისი კლასიფიცირება შეიძლება შემდეგი კატეგორიების მიხედვით:

- **სტანდარტული მონაცემთა ბაზების სისტემა:** ეს არის ფართოდ გავრცელებული სტრუქტურულ მონაცემთა მართვის და ძებნის საკითხი. ყველა მონაცემი ასეთ მონაცემთა ბაზაში უნდა ეთანხმებოდეს წინამდებარედ აღწერილ სტრუქტურას და შეზღუდვებს. მონაცემთა ბაზის მოთხოვნის ფორმულირებისათვის მომხმარებელმა დაწვრილებით უნდა აღწეროს, რომელი

მონაცემთა ობიექტი არის საძიებელი, რომელი გასაფართოებელი ან პრედიკატული. ძირითადად, მონაცემთა ბაზის აღწერილობის ენა არის ხელოვნური სახეობა, სადაც სინტაქსური შეზღუდვა და სიტყვათა მარაგი ისეთივეა, როგორც SQL-ში.

- **ინფორმაციის საძიებელი სისტემა (IR):** IR სისტემა გამოიყენება ძირითადად დიდი ზომის ტექსტური კოლექციების ძებნისათვის, სადაც მონაცემთა შინაარსი აღწერილია ინდექსებად საკვანძო სიტყვების ან ტექსტური შინაარსის გამოყენებით. მაგალითად, სურათები და ვიდეოები უნდა აღვწეროთ სიტყვებით ან მეტადატით (ნახ.1.4).
- **შინაარსზე დაყრდნობილი საძიებო სისტემა (CBR):** ეს მიდგომა გამოიყენება მულტიმედია ობიექტების მოძიებისას მონაცემთა ბაზების დიდ კოლექციაში თავისი თვისებრიობიდან გამომდინარე.
- **გრაფი ან შეფარდებითი ხე:** ამ მიდგომის მიზანია მოიძიოს ობიექტის ქვეგრაფები დიდ გრაფებში აღნიშნული მოდელის მიხედვით.



ნახ.1.4. IR სისტემის სქემა

1.3. კლიენტ-სერვერული არქიტექტურა კორპორაციულ სისტემებში

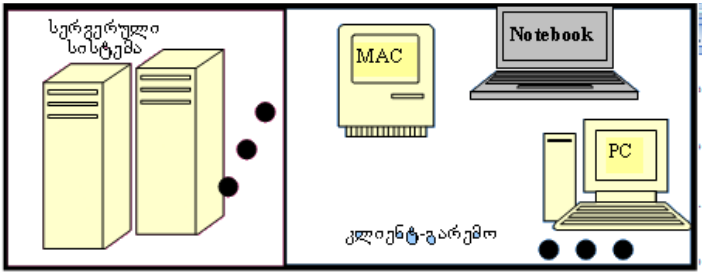
განიხილება კორპორაციული სისტემების კლიენტ-სერვერული არქიტექტურა, რომლის სერვერ-ფარმი (ტერმინალ-სერვერი) მომხმარებელთა სამუშაო სესიების მართვასა და კლიენტთა მიმართვების ფიზიკურ სერვერებზე თანაბარ განაწილებას უზრუნველყოფს, ხოლო კლასტერის (ფაილ-სერვერის) ამოცანას საერთო რესურსების (ვირტუალური IP-მისამართები და ქსელური სახელები, საერთო სერვისები და პროგრამები, განაწილებული კატალოგები და სხვა) საიმედო შენახვა და გარე მეხსიერებასთან (მაგ. SAN-დისკების მასივი) მიმართვის ორგანიზება წარმოადგენს.

პროგრამული სერვერის (Application Server) აგება და მართვა კორპორაციულ ქსელებში ერთ-ერთ უმნიშვნელოვანეს და აქტუალურ ამოცანას წარმოადგენს.

გასული საუკუნის 90-იან წლებამდე ამგვარი სერვერის განთავსება, კონფიგურირება და მართვა მხოლოდ სუპერ- და მინი- კომპიუტერების ბაზაზე იყო შესაძლებელი, რადგან ზოგადად პროგრამული სერვერი ნებისმიერი სხვა ტიპის სერვერზე მეტ რესურსებს მოითხოვს.

ბოლო ათწლეულში ტენდენცია შეიცვალა. პერსონალური კომპიუტერების გამოთვლით სიმძლავრეთა შეუქცევადი ზრდის შედეგად მათი ყველა ტიპის სერვერული ამოცანების გადასაწყვეტად გამოყენებაც გახდა შესაძლებელი და სადღეისოდ უკვე შეიძლება ითქვას, რომ სერვერორიენტირებულმა პერსონალურმა კომპიუტერმა, აპარატურისა და ადმინისტრირების ბევრად მცირე დანახარჯების წყალობით, სერვერულ სექტორშიც სუპერ- და მინიმანქანების ხვედრითი წილი ფუნდამენტურად შეამცირა.

ამასთან უნდა აღინიშნოს ისიც, რომ დროთა განმავლობაში პერსონალური კლიენტ-მანქანების მრავალი ახალი ტიპის შექმნის მიუხედავად (რომელთაც ხშირად საკმარისი ოდენობის საკუთარი გამოთვლითი რესურსებიც გააჩნია და ქსელში სერვერებისგან დამოუკიდებელი შეუძლია იყოს), თანამედროვე პროგრამ-სერვერული სისტემების აგების ოპტიმალურ იდეოლოგიად ძველებურად ცენტრალიზებული მიდგომა მიიჩნევა, რაც სქემატურად 1.5. ნახაზზეა ნაჩვენები.



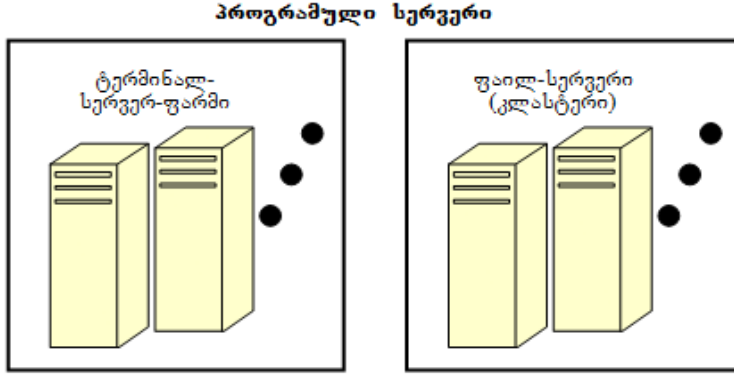
ნახ.1.5.

ამგვარი მიდგომა პროგრამული კომპლექსების მთლიანად სერვერის მხარეს განთავსებას გულისხმობს, ისევე როგორც სუპერ- და მინიმანქანების შემთხვევაში. ჩამოვთვალოთ პროგრამული სერვერის ძირითადი მახასიათებლები:

- გამოყენებითი პროგრამების დიდი რაოდენობა;
- ერთდროულად შესრულებადი პროგრამების მაღალი რიცხვი;
- ცალკეული, უნიკალური მონაცემთა ბაზები გამოყენებითი პროგრამების უმრავლესობისთვის;
- ინტენსიური ქსელური ტრაფიკი.

ზემოთ მოყვანილი სია პროგრამული სერვერების დიდ „პრეტენზიულობაზე” მიუთითებს. შეიძლება ითქვას, რომ პროგრამული სერვერი ყველა სხვა სერვერზე მეტი ოდენობით

მოითხოვს გამოთვლით რესურსებს. მას სჭირდება სწრაფი პროცესორებიც (უმჯობესია მულტიპროცესორული სისტემები), დიდი ოდენობით ოპერატიული და გარე მეხსიერება და მაღალი გამტარუნარიანობის ქსელური ინტერფეისები. რამდენადაც ცალკეული პერსონალური, თუნდაც სერვერ-სპეციფიკური კომპიუტერებისთვის ამგვარი მოთხოვნები ძნელი შესასრულებელია, პროგრამულ სერვერები სადღეისოდ კომპლექსური სახით აიგება (ნახ.1.6.).



ნახ.1.6. პროგრამული სერვერის ზოგადი სქემა

ოპტიმალური პროგრამული სერვერი სასურველია პირველი დონის ორ სერვერულ სისტემაზე იყოს დაფუძნებული: ტერმინალ-სერვერსა და ფაილ-სერვერზე. ორივე მათგანი, როგორც წესი, ერთზე მეტ ფიზიკურ სერვერს მოიცავს (მათ ლოგიკურ ერთობლიობას ტერმინალ-სერვერისათვის სერვერ ფარმი, ხოლო ფაილ-სერვერისთვის – კლასტერი ეწოდება), რაც მთლიანი სისტემის უსაფრთხოების უმთავრეს გარანტს წარმოადგენს, რადგან ორივე შემთხვევაში რომელიმე ფიზიკური

სერვერის მწყობრიდან გამოსვლისას მის ამოცანებს ავტომატურად მისი „კოლეგა” გადაიბარებს და სისტემის მუშაობის საერთო შეფერხება მხოლოდ მისი სისწრაფის შენელებით შემოიფარგლება.

სერვერ-ფარმი მომხმარებელთა სამუშაო სესიების (Client Sessions) მართვასა და კლიენტთა მიმართვების ფიზიკური სერვერებზე თანაბარ განაწილებას (Load Balancing) უზრუნველყოფს, ხოლო კლასტერის ამოცანას საერთო რესურსების (ვირტუალური IP-მისამართები და ქსელური სახელები, საერთო სერვისები და პროგრამები, განაწილებული კატალოგები და სხვა) საიმედო შენახვა და გარე მეხსიერებასთან (მაგ. SAN-დისკების მასივი) მიმართვის ორგანიზება წარმოადგენს.

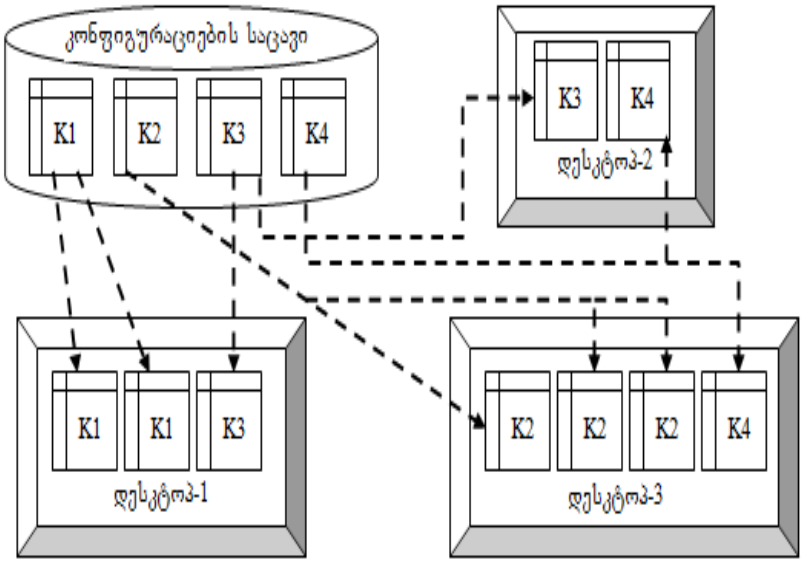
სერვერ-ფარმებზე პროგრამული სერვერის ორგანიზებისთვის საჭიროა იმ ინფრასტრუქტურის მოხაზვა, რომელშიც პროგრამული სერვერის კომპონენტები იმუშავებს.

პროგრამული სერვერის ძირითადი ამოცანებიდან შეიძლება დავასახელოთ:

- ერთიანი ინფრასტრუქტურის აგება სერვერზე განთავსებული პროგრამული პაკეტების მართვისთვის;
- პროგრამებისთვის აუცილებელი რესურსების ცენტრალიზებული გაცემა და მართვა;
- ლიცენზიების კონტროლი.

პირველი ამოცანის გადასაწყვეტად განვსაზღვროთ პროგრამული სერვერის ფუნდამენტური ელემენტი, რომელსაც ფესვური კონტეინერი ანუ კონფიგურაცია (Configuration) ვუწოდოთ. კონფიგურაცია მმართველი და ინტერფეისის ელემენტების, პროგრამებზე, გამოყენებულ რესურსებსა და დოკუმენტაციაზე მაჩვენებლების ერთობლიობას წარმოადგენს და ფოლდერების, გამოყენებითი პროგრამების, სისტემური პროცედურებისა და სხვა ინტერფეისული და არაინტერფეისული ელემენტებისგან აიგება.

კონფიგურაცია ჩონჩხია პროგრამული პროფილების ასაგებად, სხვა სიტყვებით, კონფიგურაციების სია წარმოადგენს გამოყენებითი პროგრამების საცავს (რეპოზიტორს), სადაც კონფიგურაციების სხვადასხვა ქვესიმრავლეებით აიგება დამოუკიდებელი მომხმარებლის ინტერფეისები – დესკტოპები (ნახ.1.7).



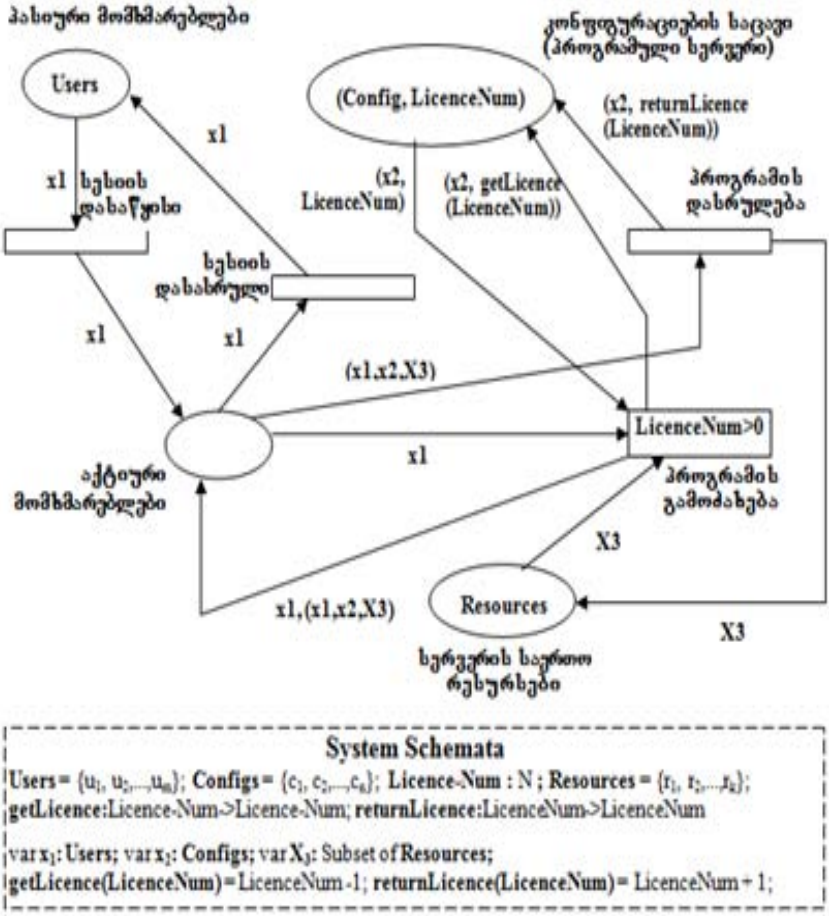
ნახ.1.7. პროგრამული სერვერის ზოგადი სქემა

კონფიგურაციათა საცავი წარმოადგენს სიმრავლეს (დომენი) K={K1,K2,…,Kn}, ხოლო პროგრამული პროფილი (დესკტოპი) არის მულტისიმრავლე (კომპლექტი) მოცემული სიმრავლის ელემენტებზე Di{K1,K1,…,K1,K2,…,K2,…}, სადაც i=1..n და მასში ყოველი მოცემული კონფიგურაცია შეიძლება შედიოდეს ერთხელ, n-ჯერ, ან საერთოდ არ შედიოდეს.

დესკტოპების მულტისიმრავლეთა სიმრავლეს:

$$D = \{D1,D2,\ldots,Dn\}$$

პროგრამული სერვერი ვუწოდოთ. 1.8 ნახაზზე წარმოდგენილია ამგვარი სერვერის მუშა მოდელის მაგალითი სისტემური პეტრის ქსელების გარემოში.



ნახ.1.8. პროგრამული სერვერის ზოგადი სქემა

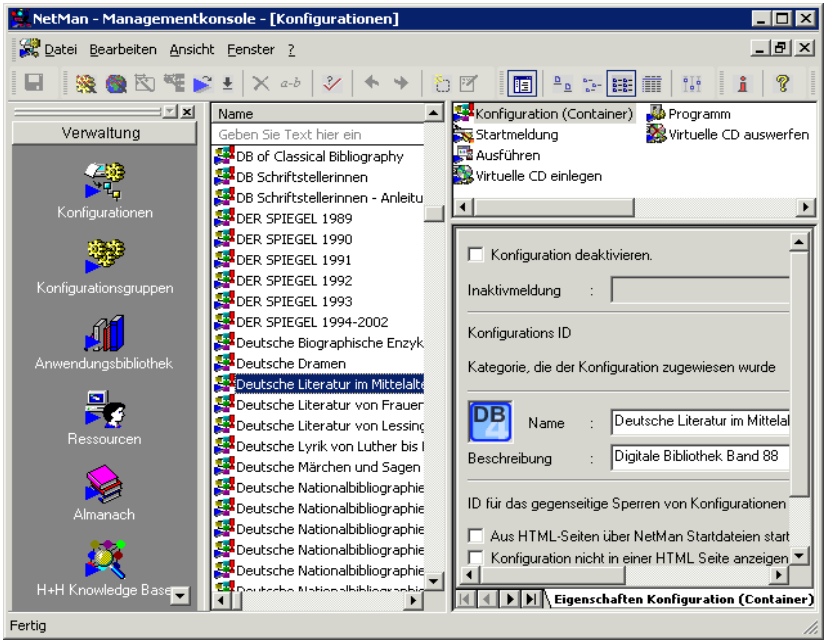
სერვერ-ფარმების მართვის Windows-ორიენტირებული სისტემებიდან შეიძლება გამოვყოთ Citrix Meta Frame და Windows 2003 terminal Server, რომლებიც ნაწილობრივ პროგრამული სერვერების მართვის ინსტრუმენტებსაც მოიცავენ, თუმცა სერიოზული პროგრამული სერვერების ასაგებად გამოიყენება სპეციალიზებული პროგრამული პროდუქტები. მაგალითად, გერმანული ფირმა H+H-ს მიერ წარმოებული პროგრამული პაკეტი NetMan პროგრამულ სერვერებზე ან თუნდაც ლოკალურ კომპიუტერებზე გამოყენებითი პროგრამების სამართავად მეტად მოხერხებულია ინსტრუმენტია.

პროგრამული პაკეტი პროგრამული სერვერებისთვის ზემოთ აღწერილ ყველა აუცილებელ მოთხოვნებს აკმაყოფილებს. იგი შეიცავს მომხმარებელთა, სამუშაო ადგილების (Workstations), საერთო რესურსებისა და პროგრამული ბიბლიოთეკების მართვის ინსტრუმენტების სრულ ნაკრებს. მაგალითისთვის შეიძლება დავასახელოთ CD_ROM-ებზე შენახული პროგრამული პროდუქტების სერვერზე მოქნილი გადატანის შესაძლებლობა (ვირტუალური კომპაქტ-დისკების შექმნა), რომელსაც NetMan იმავე ფირმის პროგრამულ პაკეტ Virtual CD-სთან ერთობლივი მუშაობით ახორციელებს.

NetMan-ის ბოლო ვერსიების (NetMan XP, NetMan 3.0) უმნიშვნელოვანეს შენაძენად ინტერნეტ-ტექნოლოგიების ინტეგრირება უნდა მივიჩნიოთ. პროგრამის ვებ-კომპონენტები უზრუნველყოფს პირველ რიგში, პროგრამული სერვერის არქიტექტურის ინტერნეტისთვის გასაგებ ფორმატში (HTML) გადაყვანას, აგრეთვე ქმნის დამატებით ვებ-ინფრასტრუქტურას ინტერნეტ-გარემოში პროგრამული კომპლექსების ეფექტური აგებისა და მართვისთვის.

ეს მეორე გარემოება NetMan-ის კორპორაციული ქსელების საზღვრებს გარეთ გატანის და პროგრამული სერვერების

გლობალური (მაგალითად, კორპორაციათაშორისი) ქსელებისთვის აგების საშუალებასაც იძლევა. პროგრამა NetMan-ის მუშაობის მაგალითი 1.9 ნახაზზეა წარმოდგენილი.



ნახ.1.9. პროგრამა NetMan-ის მართვის კონსოლი

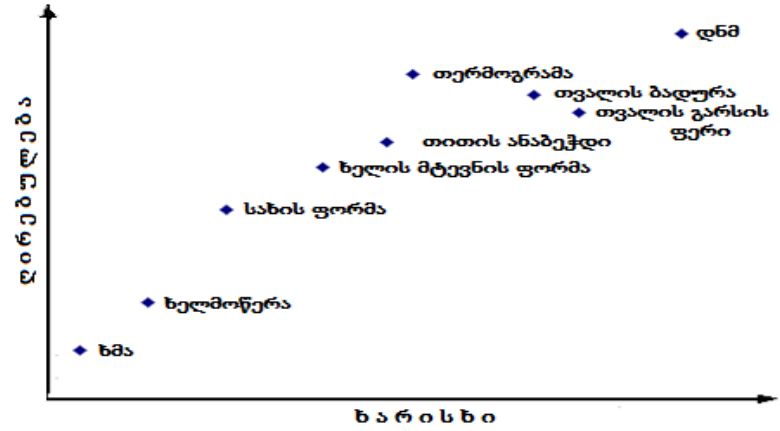
1.4. მულტიმედიური მონაცემების აუთენტიფიკაცია

უკანასკნელ პერიოდში პიროვნების იდენტიფიკაციის ბიომეტრული სისტემები იწვევს მზარდ ინტერესს, რაც აიხსნება არა მარტო მათი დინამიკური განვითარებით, არამედ მთელ მსოფლიოში მუდმივად გართულებული პოლიტიკური სიტუაციით. წვდომის, მართვის ინტეგრირებულ სისტემებში ბიომეტრული აუთენტიფიკაციის ტექნოლოგიების გამოყენება

მნიშვნელოვანი ნაბიჯია სხვადასხვა ობიექტის უსაფრთხოების უზრუნველსაყოფად.

აუთენტიფიკაციის ხარისხის და საიმედობის ამაღლების მიზნით მეტად აქტუალურია მრავალფაქტორიანი აუთენტიფიკაციის სისტემების შექმნა. ასეთ სისტემებში თანამედროვე პერიოდში აქტიურად გამოიყენება ხელოვნური ინტელექტის, კერძოდ, სახეთა შეცნობის მეთოდი, რომელიც ეფუძნება ნეირონულ ქსელებს. ამდენად, მიზანშეწონილია ნეირონული ქსელის განსწავლის ალგორითმის და მის საფუძველზე აგებული ბიომეტრული აუთენტიფიკაციის სისტემის დამუშავება.

ბიომეტრული სისტემების დახასიათებისთვის გამოიყენება სტატიკური მაჩვენებლები, როგორიცაა: სისტემაში დარეგისტრირებული სუბიექტის წვდომაზე მტყუნება და სისტემაში დაურეგისტრირებელი სუბიექტის დაშვება. უსაფრთხოების თვალსაზრისით მეორე მაჩვენებელი გაცილებით კრიტიკულია (ნახ.1.10.).



ნახ. 1.10. იდენტიფიკაციის სისტემების შეფასების სქემა

აღნიშნულის მიხედვით, ბიომეტრული მეთოდების სორტირება საუკეთესოდან უარესისაკენ შეიძლება შემდეგი სახით: დნმ; თვალის ფერადი გარსი, თვალის ბადურა; თითის ანაბეჭდი, სახის თერმოგრამა, ხელის მტევნის ფორმა; სახის ფორმა, ხელის მტევნებზე ვენების განლაგება; ხელმოწერა, კლავიატურის ხელწერა და ა.შ.

დნმ მიხედვით აუთენტიფიკაციის ტექნოლოგია, მართალია გამოირჩევა მაღალი საიმედობით, მაგრამ ამავდროულად ძვირადღირებულია და გარკვეულ დროს მოითხოვს, რის გამოც მისი გამოყენება მიზანშეწონილია მაღალი დონის აუთენტიფიკაციის სისტემებში და ვერანაირად ვერ მოხერხდება მისი გამოყენება არჩევნების დროს.

თვალის ფერადი გარსის მიხედვით აუთენტიფიკაციის ტექნოლოგია არ მოითხოვს მომხმარებლისგან განსაკუთერებულ ძალისხმევას, ვინაიდან თვალის ვიდეოგამოსახულება შეიძლება იყოს სკანირებული ერთი მეტრის მანძილზე, რაც ბანკომატებში ასეთი სკანირების გამოყენების შესაძლებლობას იძლევა. თუმცა, ამ ტექნოლოგიას გააჩნია ისეთი უარყოფითი მხარეები, როგორიცაა: იდენტიფიკაციის ალგორითმების სირთულე და მხედველობაზე მავნე ზემოქმედების შესახებ არსებული აზრის გამო შექმნილი დისკომფორტი. ამგვარად, ამ ტექნოლოგიის გამოყენება ვერ მოხერხდება ელექტრონული საარჩევნო სისტემისათვის.

თვალის ბადურის მიხედვით აუთენტიფიკაციის ტექნოლოგია იყენებს დაბალი ინტენსივობის ინფრაწითელ სხივებს. თვალის ბადურის სკანერებმა პოვა დიდი გამოყენება წვდომის მართვის ზესაიდუმლო სისტემებში, ვინაიდან აღნიშნული აუთენტიფიკაციის საშუალება ხასიათდება დარეგისტრირებული მომხმარებლის წვდომაში მტყუნების დაბალი პროცენტული მაჩვენებლით და შეცდომითი წვდომის თითქმის ნულოვანი პროცენტით. უარყოფით მხარეს მიეკუთვნება

წაკითხვის სირთულე, საიდენტიფიკაციო პარამეტრების არამდგრადობა, იდენტიფიკაციის ალგორითმების სირთულე, მხედველობაზე მავნე ზემოქმედების შესახებ არსებული აზრის გამო შექმნილი დისკომფორტი.

აუთენტიფიკაცია თითის ანაბეჭდის მიხედვით საკმაოდ მარტივი განსახორციელებელია. აღნიშნული ტექნოლოგიის უპირატესობა არის გამოყენების სიმარტივე, მოხერხებულობა და საიმედოობა, შეცდომების ალბათობის სიმცირე. გარდა ამისა, იდენტიფიკაციის მოწყობილობა საკმაოდ კომპაქტურია. აღნიშნული მეთოდის უარყოფითი მხარეებიდან აღსანიშნავია იდენტიფიკაციის ალგორითმების სირთულე, უშუალო კონტაქტის აუცილებლობა მოწყობილობასთან. ამ ტექნოლოგიას გააჩნია ისეთი მოქნილობა, რაც აუცილებლად უნდა გამოვიყენოთ ელექტრონული საარჩევნო სისტემის ფორმირებისას.

სახის, ხელის თერმოგრამა უნიკალურია. იგი ხასიათდება მუდმივობით, რაც ზრდის მეთოდის საიმედობის ხარისხს, მაგრამ, ამავე დროს, ეს მეთოდი ძვირადღირებულია ტექნიკური თვალსაზრისით. ამრიგად, ზემოთ ხსენებული მეთოდი უნდა გამოვრიცხოთ.

ბიომეტრული ტექნოლოგია ხელის მტევნის გეომეტრიის მიხედვით, გამოიყენება რვა ათასზე მეტ ორგანიზაციაში. მისი საიმედოობა საკმაოდ დიდია. მოწყობილობა Handkey სკანირებას უკეთებს ხელის როგორც შიდა, ასევე გვერდით მხარეს. უარყოფითი მხარეებიდან აღსანიშნავია: გათვლილია მარჯვენა ხელისთვის, გამოიყენება მხოლოდ პინ-კოდით, მოითხოვს უშუალო კონტაქტს მოწყობილობასთან.

სახის გეომეტრიის მიხედვით იდენტიფიკაციის ტექნოლოგია ერთ-ერთი სწრაფად განვითარებადი მიმართულებაა ბიომეტრულ ინდუსტრიაში. აღნიშნული მეთოდი ადამიანების მიერ ერთმანეთის შეცნობასთან დაახლოებულია, რაშიც

გამოიხატება მისი მიმზიდველობა. მისი განვითარება დაკავშირებულია მულტიმედიური ვიდეოტექნოლოგიების სწრაფ ზრდასთან. უარყოფითი მხარეებიდან აღსანიშნავია: დამოკიდებულება განათებაზე, დამოკიდებულება თავის მდგომარეობაზე, ტყუპების შემთხვევაში წარმოქმნილი პრობლემები.

ჩამოთვლილი უარყოფითი მხარეებიდან, საარჩევნო უბნებზე შეგვიძლია მოვაგვაროთ განათების პრობლემა და თავის მდგომარეობაც. ერთადერთი რაც გვრჩება პრობლემებიდან არის ტყუპების შემთხვევა, რაც მარტივად რეგულირდება ბიომეტრიულ ფოტოსურათს+თითის ანაბეჭდი.

ხელმოწერის მიხედვით აუთენტიფიკაცია ერთ-ერთი გავრცელებული მეთოდია. აღნიშნული ტექნოლოგიის არსი მდგომარეობს ხელმოწერის დინამიკის ანალიზის მიხედვით ციფრული კოდის ფორმირებაში. აქ გასათვალისწინებელია დროებითი მახასიათებლები, როგორიცაა საწერი ინსტრუმენტის დაჭერის ხარისხი და დინამიკა. თანამედროვე პერიოდში მზა პროდუქტები ვერ აკმაყოფილებს ყველა მოთხოვნას, მათთვის დამახასიათებელია არც თუ ისე მაღალი საიმედოობა, მაგრამ მისი გამოყენება საკმაოდ პერსპექტიულია.

რაც შეეხება კლავიატურის ხელწერას, იგი არ მოითხოვს სპეციალურ მოწყობილობას, თუმცა წვდომის მართვის საკმარის დონეს ვერ აკმაყოფილებს.

აუთენტიფიკაციის სისტემები ხმის მიხედვით მოხერხებულია პრაქტიკული თვალსაზრისით, მაგრამ ხასიათდება იდენტიფიკაციის დაბალი სიზუსტით.

მიგვაჩნია, რომ ბოლო სამი მეთოდიდან მესამე მეთოდის არსებობა ხელს არ შეუშლის საარჩევნო სისტემის ფუნქციონირებას და პირიქით, შეაგროვებს დამატებით

ინფორმაციას ამომრჩეველზე, რომელიც შეგვიძლია აღვიქვათ როგორც მეტადატად.

ყოველივე ზემოაღნიშნულიდან გამომდინარე შევაფასოთ ბიომეტრული მეთოდები მახასიათებლების მაღალი, საშუალო და დაბალი მაჩვენებლებით (ცხრ.1.1)

ბიომეტრული მეთოდების შეფასების ცხრილი ცხრ.1

	დნმ	თვალის ფერადი გარსი	თვალის ბადური	თითის ანაბეჭდი	ხახის, ხელის თერმოგრამა	ხელის მტევნის ფორმა	ხახის ფორმა	ხელმოწერა	ხმა
უნიკალობა	მ	ს	ს	ს	მ	ს	მ	მ	ს
ადამიანის მზაობა იდენტიფიკაციის გასავლელად	დ	ს	დ	ს	ს	მ	მ	მ	მ
საიმედოობა	მ	მ	ს	მ	მ	დ	დ	მ	დ
ხანდაზმულობა	მ	მ	ს	მ	მ	ს	ს	მ	დ
აპარატურის უზრუნველყოფის მაჩვენებლები	მ	მ	ს	მ	ს	მ	დ	ს	დ
აპარატურის სირთულის კოეფიციენტი	მ	მ	ს	მ	მ	დ	დ	მ	დ

აღნიშნული ანალიზიდან გამომდინარე, შეიძლება გამოვყოთ ორი ყველაზე უფრო მოთხოვნადი ტექნოლოგია. ეს არის: ბიომეტრული იდენტიფიკაცია თითის ანაბეჭდით და ბიომეტრული იდენტიფიკაცია სახის ფორმის მიხედვით. თუმცა,

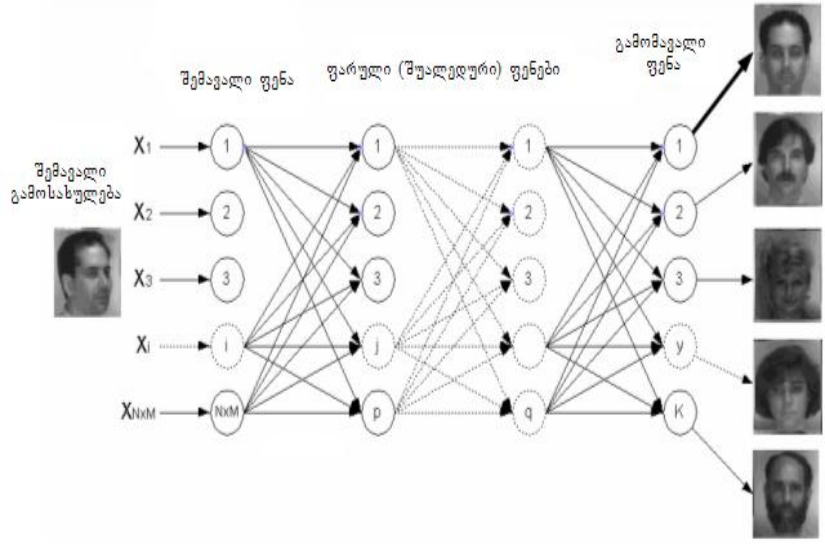
თითის ანაბეჭდით იდენტიფიკაციის შემთხვევაში შეიძლება იყოს მტყუნებები, რაც დაკავშირებულია პაპილარული (უზორების) დეფორმაციასთან ჭრილობების ან დამწვრობის შემთხვევაში. აღნიშნული პრობლემის საუკეთესო გადაწყვეტაა მრავალფაქტორიანი იდენტიფიკაციის სისტემების გამოყენება, როგორიცაა, მაგალითად, თითის ანაბეჭდის, სახის ფორმის გეომეტრიის და ელექტრონული ხელმოწერის მიხედვით.

აღნიშნულ შემთხვევაში შეცდომები მკვეთრად მცირდება, ხოლო სისტემის საიმედოობის ხარისხი იზრდება გამოყენებული ფაქტორების რაოდენობის პროპორციულად. რაც შეეხება სახის ფორმის მიხედვით ბიომეტრულ იდენტიფიკაციას, თანამედროვე სამყაროში აუთენტიფიკაციის სისტემებში აქტიურად გამოიყენება სახეთა შეცნობის მეთოდი, რომელიც ეფუძნება ნეირონულ ქსელებს.

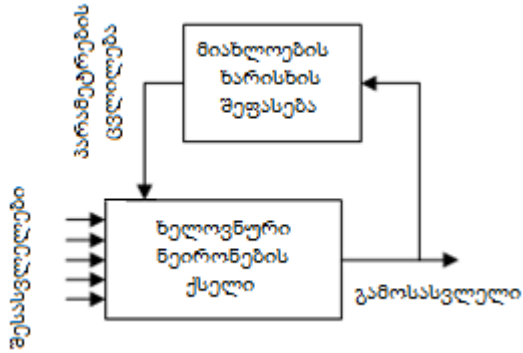
შეცნობის ამოცანებში შეიძლება გამოვყოთ ორი ეტაპი: საწყისი მონაცემების შეცნობისათვის მოხერხებულ ფორმაში მოყვანა და თვით შეცნობა, რაც მოიაზრებს ობიექტის განსაზღვრულ კლასზე მიკუთვნებას. ამდენად, მნიშვნელოვანია ობიექტების სიახლოვის, მსგავსების ცნების სწორი ფორმულირება, რისთვისაც მოუხერხებელია კლასიკური მათემატიკური მეთოდების გამოყენება. ბიომეტრული აუთენტიფიკაციის ამოცანებში ხელოვნური ნეირონული ქსელი საკმაოდ ფართოა და ღრმა, ამდენად მას აქვს ათობით შესასვლელი და გამოსასვლელი, ასევე ათობით ფენა.

1.11 ნახაზზე წარმოდგენილია მრავალფენიანი ნეირონული ქსელი გამოსახულების კლასიფიკაციისათვის.

აქედან გამომდინარე ხელოვნური ნეირონული ქსელის განსწავლის სქემა შეიძლება წარმოვადგინოთ 1.12. ნახაზზე.

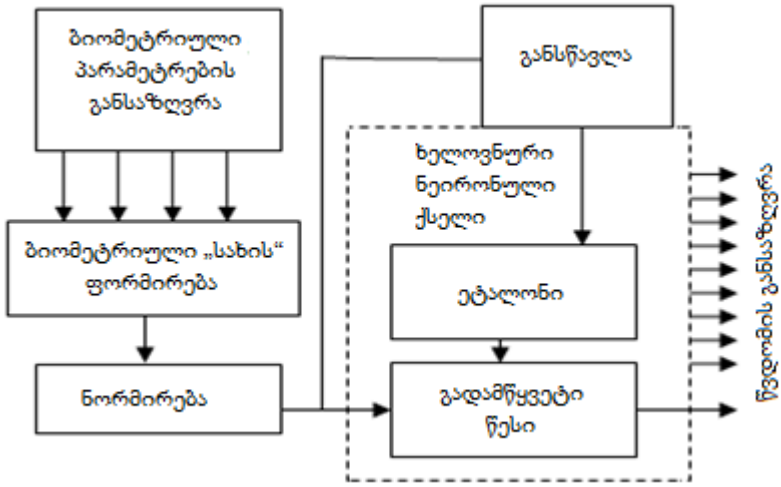


ნახ.1.11. მრავალფენიანი ნეირონული ქსელი გამოსახულების კლასიფიკაციისათვის



ნახ.1.12. ნეირონული ქსელის განსწავლის მართვის სქემა

ნეირონული ქსელის დინამიკური სისტემის ასაგებად საჭიროა განისაზღვროს: ობიექტი, რომელიც ასრულებს ნეირონული ქსელის შემავალი სიგნალის როლს; ობიექტი, რომელიც ასრულებს ნეირონული ქსელის გამომავალი სიგნალის როლს, რომელიც შეიძლება იყოს თვით გადაწყვეტილება ან მისი რაიმე მახასიათებელი; ნეირონული ქსელის სასურველი გამომავალი სიგნალი; შუალედური ფენების რაოდენობა და კავშირი ფენებს შორის, წონითი კოეფიციენტები; სისტემის ცდომილების ფუნქცია. ბიომეტრული აუთენტიფიკაციის სისტემის სქემა წარმოდგენილია 1.13. ნახაზზე.



ნახ.1.13. ბიომეტრიული აუთენტიფიკაციის სისტემის სქემა

ნეიროქსელური მეთოდი უზრუნველყოფს გამოსახულების სწრაფ და საიმედო შეცნობას. თუმცა ამ მეთოდის გამოყენებისას სამგანზომილებიანი ობიექტების გამოსახულებებთან მიმართებით

წარმოიქმნება სივრცეში განთავსებასა და განათებასთან დაკავშირებული პრობლემები.

ბიომეტრული აუთენტიფიკაცია მულტიმედიურ მონაცემთა ბაზებში არსებულ ჩანაწერებს ვერ განსაზღვრავს აბსოლუტური სიზუსტით, ამდენად ბიომეტრული აუთენტიფიკაციის სისტემები ამოიცნობს მომხმარებელს გარკვეული ალბათობით. სიზუსტის თვალსაზრისით ხშირ შემთხვევებში ბიომეტრული აუთენტიფიკაცია გამოიყენება მომხმარებლის პაროლთან ან სხვა მეთოდთან კომბინაციაში.

1.5. თანამედროვე დაცული ქსელის აგება VPN-ის გამოყენებით

VPN (ვირტუალური კერძო ქსელი). იგი უზრუნველყოფს დამოუკიდებელი დაცული ქსელის შექმნას ინტერნეტის ან სხვა ღია არხების მეშვეობით. აღნიშნული ტექნოლოგია საშუალებას გვაძლევს შევქმნათ ცენტრალური საარჩევნო სისტემის საკუთარი კორპორაციული ქსელი ნებისმიერ საარჩევნო უბანზე. VPN კლიენტის პროგრამული უზრუნველყოფა იტვირთება დაშორებულ კომპიუტერზე და ინტერნეტის მეშვეობით უკავშირდება კორპორატიულ ქსელში განთავსებულ VPN სერვერს ან როუტერს. VPN ტექნოლოგია ოპტიმალურია ელექტრონული საარჩევნო სისტემისათვის შექმნილი ქსელის სამართავად. ცენტრალური საარჩევნო კომისიიდან გაუსვლელად წვდომა გვექნება შიდა ქსელის ყველა რესურსთან, როგორიც არის მულტიმედიური მონაცემთა ბაზები და ა.შ. VPN (ვირტუალური ლოკალური ქსელის) შესაქმნელად საჭიროა არსებობდეს ფიზიკური არხი, როგორც საკაბელო, ასევე რადიოსიხშირული სპექტრის გამოყენებით. ჩვენს შემთხვევაში გამოყენებული იქნება მხოლოდ და მხოლოდ საკაბელო არხები, ამაზე ზემოთ

მოგახსენეთ. VPN ორი ნაწილისაგან შედგება. ესენია - „შიგა“ და „გარე“ ქსელი. პირველ შემთხვევაში, იქმნება ერთი ლოკალური ქსელი, რომლის მოქმედების რადიუსი, მაქსიმუმ 100 მეტრია და რომელიც უშუალოდ საარჩევნო უბანზე აიგება, ხოლო „გარე“ ქსელი ამ ქსელს დააკავშირებს ცენტრალურ სერვერებთან. VPN-მა, თავის მხრივ, შეიძლება გამოიყენოს DSL ტექნოლოგიაც, რასაკვირველია აქაც საჭიროა ინტერნეტის მაღალი სიჩქარე, რათა მოხდეს მონაცემთა შეუფერხებელი გადაცემა. მონაცემები და მარშრუტიზაციის შემცველი თავსართები საჯარო ქსელის მეშვეობით დაშიფრული სახით გაიგზავნება, რასაც ტუნელირება ეწოდება. აღნიშნული ქსელის ასაგებად უნდა გამოვიყენოთ IPsec ტექნოლოგია, რომელიც ყველაზე სანდო და უსაფრთხოა თვით VPN-ის ტიპებს შორის.

რა არის IPsec და როგორ ფუნქციონირებს იგი ?

IPsec (IP security) არის OSI მოდელის მე-3 დონის პროტოკოლი. თუმცა, მიუხედავად ამისა, ამ ორ პროტოკოლს შორის ძალიან ცოტა მსგავსებაა []. GRE-ს ერთი უპირატესობა IPsec-თან შედარებით ის არის რომ IPsec-ს მხოლოდ TCP/IP მოდელის პროტოკოლების მხარდაჭერა გააჩნია და მას არ შეუძლია ჩვეულებრივ გადაიტანოს ისეთი პროტოკოლები, როგორიცაა IPX და AppleTalk. თუმცა, რადგანაც GRE წარმოადგენს IP პროტოკოლს, ამიტომაც შესაძლებელია GRE ტექნოლოგიის გვირაბის შექმნა IPsec VPN კავშირის დროს, რათა მოხდეს Non -TCP/IP ტრაფიკის დაცვა.

IPsec წარმოადგენს რამდენიმე სტანდარტის კომბინაციას. მაშინ, როცა GRE-ერთ-ერთი ტიპი VPN კავშირის, არ უზრუნველყოფს ტრაფიკის ძლიერ დაცვას, IPsec სპეციალურად შეიქმნა დაუცველ ქსელში მონაცემების უსაფრთხო გადაცემის უზრუნველსაყოფად. IPsec-ის ფრეიმვორქი უზრუნველყოფს სამ ძირითად ფუნქციას:

- მონაცემთა კონფიდენციალურობა
- მონაცემთა ინტეგრარულობა
- მონაცემთა იდენტიფიკაცია

მონაცემთა კონფიდენციალურობა გულისხმობს მონაცემთა დაცვას მოსმენითი ტიპის შეტევის დროს. მისი განხორციელება შესაძლებელია დაშიფვრის გამოყენებით.

IPsec-ს გააჩნია DES, 3DES, და AES დაშიფვრის ალგორითმების მხარდაჭერა. მონაცემთა ინტეგრარულობა გულისხმობს პაკეტის შემოწმებას, მოხდა თუ არა პაკეტში მოთავსებულ ინფორმაციაში ჩარევა.

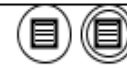
მის განსახორციელებლად გამოიყენება ჰეშირების ფუნქციები, მაგალითად MD5 და SHA.

მონაცემთა იდენტიფიკაცია გამოიყენება პაკეტისა და მოწყობილობის იდენტიფიკაციისათვის. ჰეშირების ფუნქციები გამოიყენება იმ მოწყობილობის იდენტურობის შემოწებისათვის, რომელიც აგზავნის IPsec პაკეტებს. მოწყობილობის იდენტიფიკაცია გამოიყენება იმისათვის, რომ ვაკონტროლოთ, რომელ მახლობელ მოწყობილობებს შეუძლიათ IPsec კავშირის დამყარება ადგილობრივ მოწყობილობასთან.

1.6. ბიზნესპროცესების მოდელირება და მართვა (BPMN ინსტრუმენტული საშუალება)

ბიზნესპროცესების მოდელირების თვალსაზრისით დღესდღეობით UML ენა მოქნილ ტექნოლოგიად ითვლება. თუმცა, პროცესორიენტირებული და სერვის-ორიენტირებული მიდგომის თვალსაზრისით UML ენის შემქმნელების მიერ (OMG - ObjectManagementGroup) განვითარდა და დამუშავდა ვიზუალიზაციის სპეციალური დამატებითი ელემენტების

ინსტრუმენტული საშუალება, რომელიც ბიზნესპროცესების მართვის ნოტაციითაა (BPMN- Business Process Modeling Notation) ცნობილი [48,87].

მოვლენები	გამოსახულება
ჩვეულებრივი (plain events)	
შეტყობინება (message events)	
ნამზომი (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)	
შეჩერება (terminate events)	

ნახ.1. 14. BPMN ნოტაციის ობიექტთა ნაკადის ელემენტები

ბიზნესპროცესების მართვის ნოტაციის მთავარი არსი. ობიექტორიენტირებული მიდგომის ტრანსფორმაციაა პროცეს-ორიენტირებულ მიდგომაზე, რაც ბიზნესმოდელისა და

საინფორმაციო მოდელის სინქრონიზაციის საშუალებას იძლევა. ბიზნესპროცესების მართვის ნოტაცია წარმოადგენს ე.წ. სტანდარტიზაციის ხიდს ბიზნესპროცესების დაპროექტებისა და იმპლემენტაციას შორის.

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

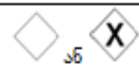
1. ობიექტთა ნაკადი;
2. დამაკავშირებელი ობიექტები და ლოგიკური ელემენტები;
3. როლები, მცოცავი ბილიკებით;
4. ხელოვნური ობიექტები – მონაცემთა ობიექტები, ჯგუფები და ანოტაცია.

განვიხილოთ თითოეული მათგანი დეტალურად.

1. ობიექტთა ნაკადი. ობიექტთა ნაკადის ფორმირებისას შესაძლებელია ბიზნესპროცესის აღწერა ორ დონეზე. პირველი დონე ეს არის მეტა-მოდელი, ანუ სრული, ზოგადი ბიზნეს-პროცესი, ხოლო მეორე დონეში აღიწერება პროცესის ცალკეული ეტაპები ანუ ქვეპროცესები (ნახ. 1.15-1.16).

BPMN ელემენტები	გამოსახულება		
	წეველებრივი	პარალელური	ციკლური
ქმედება/ოპერაცია	Task	Parallel	Loop
ქვე-პროცესი	SubProcess	Parallel SP	Loop SP

ნახ.1.15. BPMN-ნოტაციის ელემენტები

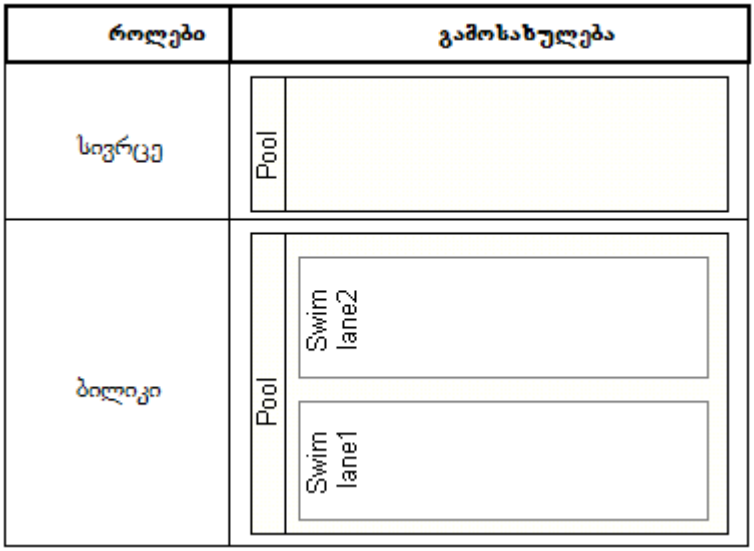
ლოგიკური ელემენტები	გამოსახულება
ლოგიკური „ან“ ოპერატორის გამორიცხვა მონაცემთა მართვისას (Data XOR)	
ლოგიკური „ან“ ოპერატორის გამორიცხვა მოვლენათა მართვისას (Event XOR)	
ლოგიკური „და“ ოპერატორი (AND)	
ლოგიკური „ან“ ოპერატორი (OR)	
ლოგიკური „რთული“ ოპერატორი (COMPLEX)	

ნახ. 1.16. BPMN ნოტაციის ლოგიკური ელემენტები

2. დამაკავშირებელი ობიექტები და ლოგიკური ელემენტები. აქ განიხილება ე.წ. მიმდევრობითობის დიაგრამა, რომელიც განშტოვდება ოპერაციათა ნაკადის, შეტყობინებათა ნაკადის და ასოციაციების სახეობებად.

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. 1.17).



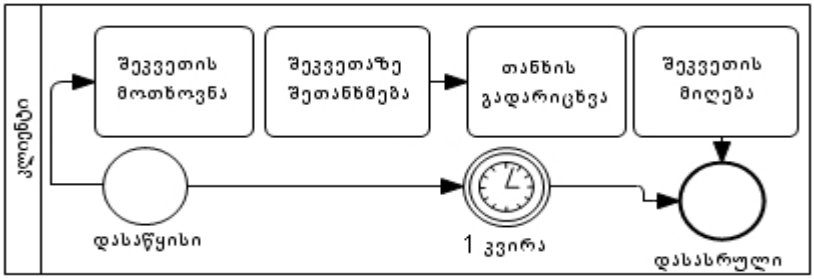
ნახ.1.17. BPMN ნოტაციის როლური ელემენტები

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 [16].

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

• კერძო (შიგა) ბიზნესპროცესი, რომელიც აღწერს ტექნოლოგიურ პროცესს ანუ საქმიან ნაკადს. კერძო ბიზნეს-პროცესის მოდელის ფრაგმენტი წარმოდგენილია 1.18 ნახაზზე.

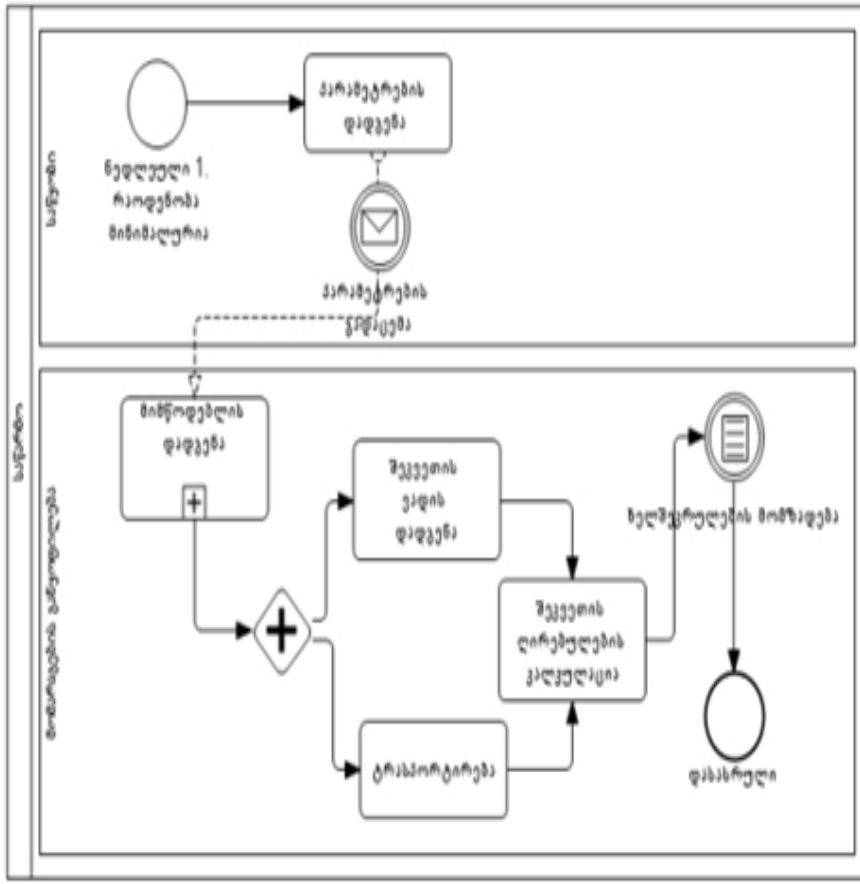


ნახ.1.18. კერძო ბიზნესპროცესის მოდელის ფრაგმენტი

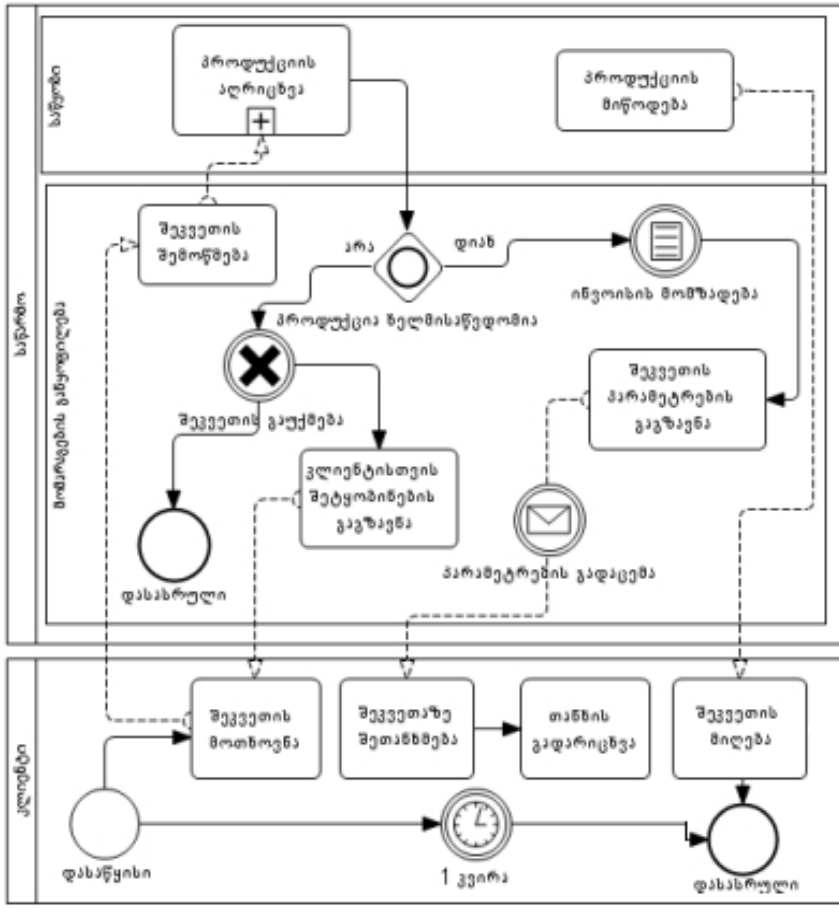
• აბსტრაქტული (ღია) ბიზნესპროცესი. იგი, აღწერს დამოკიდებულებას ორ ან მეტ კერძო პროცესს შორის ან პროცესსა და რესურსს შორის. აბსტრაქტულად ითვლება მხოლოდ ის პროცესები, რომელთა ქმედება აუცილებლად უკავშირდება კერძო ბიზნესპროცესს. ამდენად, აბსტრაქტული პროცესი ასახავს იმ შეტყობინებათა გადაცემის თანამიმდევრობას, რომლებიც ურთიერთქმედებს კონკრეტულ ბიზნესპროცესთან (ნახ. 1.19).

• 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. 1.20).

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



ნახ.1.19. აბსტრაქტული ბიზნეს-პროცესის მოდელის ფრაგმენტი



ნახ.1.20. ერთობლივი ბიზნესპროცესის
მოდელის ფრაგმენტი

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

ბიზნესპროცესების მოდელირების ნოტაციაში პრიორიტეტულია მოდელირების გრაფიკული ელემენტების ვიზუალური მხარე და დიაგრამების თავსებადობა. ამ თავსებადობის საფუძველი არის ბიზნესპროცესების მოდელირების (BPML – Business Process Modeling Language) და შესრულების ენა (BPEL – Business Process Execution Language), რომელიც ბაზირებულია XML (Extensible Markup Language) ენაზე და წარმოადგენს ბიზნესპროცესების გრაფიკული ასახვისა და მათი ურთიერთქმედების პროტოკოლების ფორმალური აღწერის ენას, რაც ბიზნესმოდელისა და საინფორმაციო მოდელის სინქრონიზაციის საშუალებას იძლევა [16].

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

სერვისორიენტირებული არქიტექტურა (Service Oriented Architecture, SOA) 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 ფუნქციურ მოდულებს (ვებ-სერვისებს) და საერთო ინტერფეისში მუშაობის საშუალებას იძლევა [14, 44, 46].

სერვისორიენტირებული მიდგომის არსია არსებული და მომავალი სხვადასხვა ფუნქციური, 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. მათი 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 ბიზნეს-პროცესების 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.

1.7. სერვერული ოპერაციული სისტემების მიმოხილვა

კომპიუტერული პლატფორმა მოიცავს კომპიუტერულ ტექნიკას, ტექნიკის არქიტექტურას და პროგრამული უზრუნველყოფის კომბინაციას, რომელიც საშუალებას იძლევა მოხდეს დამატებითი პროგრამული უზრუნველყოფის გაშვება. ტიპური პლატფორმები მოიცავს კომპიუტერის არქიტექტურას, ოპერაციულ სისტემას, პროგრამირების ენებს და მასთან დაკავშირებული ინტერფეისებს.

პლატფორმა არის გადამწყვეტი ელემენტი პროგრამული უზრუნველყოფის დამუშავებისას. პლატფორმა შეიძლება უბრალოდ განისაზღვროს როგორც პროგრამული უზრუნველყოფის „საძირკველი“. პლატფორმის პროვაიდერი სთავაზობს პროგრამისტებს პროდუქტს, რომელზეც მოხდება ლოგიკურ კოდზე დაწერილი პროგრამის გაშვება. ლოგიკური კოდი მოიცავს bytecode-ებს, source code-ებს და მანქანურ კოდებს. ეს ფაქტობრივად ნიშნავს, რომ აღსრულების პროგრამა არ არის შეზღუდული ტიპის ოპერაციული სისტემა. იგი ძირითადად შეიცვალა მანქანურ-დამოუკიდებელი ენებით.

1. **ოპერაციული სისტემები:**
    - AmigaOS, AmigaOS 4
    - FreeBSD, NetBSD, OpenBSD
    - Linux
    - Mac OS X
    - Microsoft Windows

- OS/2
- Solaris
- Unix
- IBM VM/370, VM/BSEP, VM/SEP, VM/XA, VM/ESA, z/VM
- Google Chrome OS

2. პროგრამული სტრუქტურა:
   - Adobe AIR
   - Java, JDK and JRE
   - Mono
   - Mozilla Prism XUL and XULRunner
   - .NET Framework
   - Oracle Database
   - Vexi
   - .NET
   - Java
   - SAP

ნახ.1.21.

დეტალურად განვიხილოთ ყველაზე გავრცელებული სამი ოპერაციული სისტემა:

ლინუქსი (Linux) – წარმოადგენს მრავალფუნქციურ მძლავრ უფასო ოპერაციულ სისტემას, რომელიც იუნიქსის (UNIX) მაგვარი ოპერაციული სისტემების ერთ-ერთი ნაირსახეობაა პერსონალური კომპიუტერებისთვის [40]. ოპერაციული სისტემა UNIX შეიქმნა სამოციან წლებში, კომპანია ბელ ლაბსის (Bell Labs, ამჟამად AT&T Bell Laboratories) მიერ [59] .

UNIX-ის შექმნისას გათვალისწინებული იქნა მისი მულტიპლატფორმულ ოპერაციულ სისტემად ჩამოყალიბება, ანუ შესაძლებელი უნდა ყოფილიყო მისი სხვადასხვა არქიტექტურის მანქანებზე უპრობლემოდ მუშაობა. სწორედ მისმა მულტიპლატფორმულობამ განაპირობა UNIX-ის უდიდესი პოპულარულობა. UNIX-ის პოპულარულობასთან ერთად სხვადასხვა კომპანიებმა დაიწყეს მის ბაზაზე ოპერაციული სისტემების შექმნა.

Linux არის თავისუფალად გავრცელებადი GPL ლიცენზიაზე დამყარებული ოპერაციული სისტემა, რომელიც შეიქმნა UNIX მაგვარი ოპერაციული სისტემა Minix-ის ბაზაზე. მისი თავდაპირველი ვერსიის შემქმნელია ფინეთის ჰელსინკის უნივერსიტეტის სტუდენტი ლინუს ტორვალდსი (Linus Torvalds) [40]. საგულისხმოა ის ფაქტი, რომ სტუდენტმა ახალი ოპერაციული სისტემა საკურსო ნაშრომის ფარგლებში შექმნა.

Linux შექმნილია მრავალრიცხოვან UNIX პროგრამისტთა და ინტერნეტის ქსელში მომუშავე ენთუზიასტთა დახმარებით. მასში არ არის გამოყენებული AT&T UNIX-ის პროგრამული კოდი, ასევე არც ერთი სხვა UNIX-ის პროგრამული კოდი. Linux-ის მეტი წილი პროგრამები შექმნილია GNU Free Software Foundation პროექტის ჩარჩოში, კემბრიჯში, მასაჩუსეტსი, თუმცა მის შექმნაში ასევე მონაწილეობა თითქმის მთელი მსოფლიოს პროგრამისტებმა

მიიღეს. აღსანიშნავია ისიც, რომ Linux-ის ბაზაზე შექმნილი პროგრამების უმეტესობა არის ღია კოდით (open source) და ამასთან უფასო. 0.01 ვერსიის Linux-ის გამოჩენისას არასდროს არ გაკეთებულა ოფიციალური განცხადება მისი შექმნის თაობაზე, 0.01 ვერსიის პროგრამული წყარო, პროგრამის კოდის ნორმალური შესრულების შესაძლებლობასაც კი არ იძლეოდა და მიუთითებდა იმაზე, რომ მომხმარებელი მიმართავდა Minix მანქანას. ეს მისი (Linux) კომპილირებისა თუ სრულყოფის შესაძლებლობას იძლეოდა.

Linux თავდაპირველად შეიქმნა ინტელის 32-ბიტიანი x86-ბაზაზე შექმნილ 386 ან უფრო მძლავრ პროცესორებზე სამუშაოდ. ამჟამად Linux ასევე შემდეგი პლატფორმის მანქანებზეც მუშაობს: Compaq Alpha AXP, Sun SPARC, UltraSPARC, Motorola 680x0, PowerPC, PowerPC64, ARM, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, Intel IA-64, DEC VAX, AMD x86-64, CRIS.

ნახ.1.22. ლინუქსის ეკრანი (KDE გარემო)

Linux წარმატებით შეიძლება გამოვიყენოთ, როგორც სამუშაო სადგურად, ასევე სერვერის პლატფორმად. დღესდღეობით Linux UNIX-ისმაგვარი ოპერაციული სისტემების ღირსეული კლონია. Linux სრულყოფილი მრავალფუნქციური მრავალამოცანიანი ოპერაციული სისტემაა, რომელიც წარმატებით ართმევს თავს თანამედროვე საინფორმაციო ტექნოლოგიების მზარდ მოთხოვნილებებს. მასზე შესაძლებელია X Windows-ის გრაფიკულ გარემოში და console (ბრძანებათა სტრიქონში) მოხერხებული მუშაობა. Linux-ის მომხმარებელთათვის არსებობს თანამედროვე გრაფიკული და საოფისე პროგრამები, ასევე მათთვის ინტერნეტში სრულყოფილი მუშაობისთვის უამრავი პროგრამა, რომლითაც შესაძლებელია ვებ-გვერდების დათვალიერება, ელექტრონული ფოსტის მიღება და გაგზავნა, IRC, ICQ, Yahoo Messenger, AOL Instant Messenger, MSN Messenger და სხვა პოპულარული საკომუნიკაციო საშუალებების გამოყენება.

პროგრამირებით დაინტერესებულ პირთათვის არანაკლები რაოდენობით არსებობს პროფესიული დონის გრაფიკული ინტერფეისიანი პროგრამული პროდუქტები. ამასთან პროგრამული უზრუნველყოფის სრული სპექტრი გამოიყენება Linux-ის როგორც ინტერნეტ, ბეჭდვით თუ ფაილ სერვერად. Linux-ისთვის არსებობს უმძლავრესი პროგრამული პროდუქტები მონაცემთა ბაზების შესაქმნელად, რომლის ნათელი მაგალითია კომპანია Oracle, რომელიც თავის პროდუქციას სხვა ოპერაციულ სისტემებთან ერთად ასევე Linux პლატფორმისათვის ქმნის.

ამჟამად Linux და მის ბაზაზე შექმნილი პროგრამები საკმაოდ დეტალურად არის დოკუმენტირებული. ეს ინფორმაცია უფასოა და ხელმისაწვდომი შესაბამის ინტერნეტგვერდებზე. Linux-ის პრინციპია: მომხმარებელს უფასოდ მიეწოდოს რაც შეიძლება მეტი ინფორმაცია და პროდუქტი, რაც ხელსაყრელი ალტერნატივაა მომხმარებლისთვის.

Windows - „ფანჯრები“ — დღეისათვის არსებული ერთ-ერთი ყველაზე პოპულარული ოპერაციული სისტემა შექმნილია კომპანია Microsoft-ის მიერ [89]. ბოლო 15 წლის განმავლობაში იგი ყველაზე დიდი პოპულარობით სარგებლობს ბაზარზე. ვინდოუსი მუშაობს შემდეგ პლატფორმებზე: x86, AMD64, IA-64, ასევე DEC Alpha, MIPS და PowerPC. ეს პროგრამა (Windows 1.0x) ამერიკელმა სტუდენტმა ბილ გეითსმა გამოიგონა [90]. იგი MsDos-ისთვის მომხმარებლის გრაფიკული ინტერფეისი იყო.

განვიხილოთ დღეისათვის ფართოდ გავრცელებული ოპერაციული სისტემები: Windows 7, Windows server 2008 [91,92].

განსხვავებით წინამორბედი Windows Vista-სგან, რომელშიც შეტანილი იყო მრავალი სიახლე, Windows 7-ში ყურადღება ექცევა იმ მოწყობილობებთან თავსებადობას, რომლებთანაც წინა სისტემაც იყო თავსებადი. Microsoft-ის მიერ 2008 წელს გამართულ პრეზენტაციებზე წარდგენილი იქნა ახალი სისტემის რამდენიმე სიახლე, როგორიც იყო შეხების ტექნოლოგია, სწრაფად გაშვების განსხვავებული დაფა, რომელსაც Superbar ეწოდება, შინაური ქსელის სისტემა HomeGroup, და გაუმჯობესებები სწრაფ-მოქმედების მხრივ. ახალ სისტემაში შეტანილი რამდენიმე პროგრამა, რომელიც უკვე ვრცელდებოდა Windows-ის წინა ვერსიასთან ერთად (მაგალითად, Windows Calendar, Windows Mail, Windows Movie Maker და Windows Photo Gallery) არ იქნა შეტანილი Windows7-ში – მომხმარებელს შეუძლია მათი თავად გადმოტვირთვა და ინსტალირება პროგრამების კრებულის Windows Live Essentials ფარგლებში.

დაიწყო მუშაობა სისტემაზე „Longhorn", რომლის წარდგენაც 2003 წლისათვის იყო დაგეგმილი. ამ ვერსიაში თავდაპირველად შეიტანეს გადადებული პროექტის რამდენიმე ელემენტი. საბოლოოდ, Longhorn-ს ეწოდა Windows Vista და იგი ოფიციალურად გამოვიდა 2007 წლის იანვარში [93]. Windows Vista-

ზე მუშაობის პარალელურად, Blackcomb-ს ეწოდა *Vienna,* თუმცა 2007 წლის 20 ივლისს ცნობილი გახდა, რომ სახელწოდება შეცვლილი იქნა და ამჯერად ეს იყო *Windows 7*.

Windows Server 2008 Microsoft-ის სერვერული ოპერაციული სისტემა [92]. ეს ოპერაციული სისტემა წარმოადგინეს, როგორც Windows 2003 Server-ის შემცვლელი.Windows Server 2008-ის განახლებული ვერსია Windows Server 2008 R2 გამოვიდა.

Windows Server 2008-ის სერვისებშიც არის საკმაოდ ბევრი ცვლილებები. მაგალითად IIS ვებ სერვერში, რომელიც თითქმის თავიდან-ბოლომდე ხელახლა დაიწერა იმისათვის, რომ გაუმჯობესებულიყო წარმადობა და დაცვა, სრულყოფილია Distributed File System, რომელსაც ახლა უკვე შეუძლია დაჰოსტოს მრავალი DFS root-ები ერთ სერვერზე. გაუმჯობესებულია ასევე Terminal სერვერი, Active Directory, Print სერვერი და ა.შ. ამ პროდუქტმა რამდენჯერმე შეიცვალა სახელი. მასში ჩაშენებულია ახალი და განახლებული შესაძლებლობები:

- Internet Information Services (IIS) v7.0 – გაუმჯობესებულია IIS-ის ვერსია;
- წინა ვერსიებთან შედარებით გაუმჯობესებულია default დაცულობა. ჩაშენებულია firewall და default-ად გამორთულია სერვისების უმეტესობა;
- მნიშვნელოვანი ცვლილებებია Message Queuing-ში;
- Manage Your Server – სამართავი ხელსაწყო, რომელიც საშუალებას აძლევს ადმინისტრატორს აირჩიოს თუ რა ფუნქციონირება ექნება სერვერს;
- გაუმჯობესებულია Active Directory;
- გაუმჯობესებულია Group Policy და მისი ადმინისტრირება უფრო გაადვილებულია;

- გაუმჯობესებულია დისკების მართვა, როგორიცაა shadow ფაილების backup-ის შესაძლებლობა, ასევე გახსნილი ფაილების backup-ის შესაძლებლობა;
- გაუმჯობესებულია scripting და command line ხელსაწყოები.

Windows Server-ს აქვს რამდენიმე გამოცემა. ყოველი მათგანი განკუთვნილია სხვადასხვა მიზნებისთვის. ყველა გამოცემას შეუძლია შეასრულოს შემდეგი ფუნქციები: გაანაწილოს ფაილები და პრინტერები, იმუშაოს როგორც აპლიკაციის სერვერად, მოგვაწოდოს იმეილ სერვისები, აუთენტიფიკაცია გაუკეთოს მომხმარებელს, იმუშაოს როგორც X.509 სერტიფიკატის სერვერად, მოგვაწოდოს LDAP სერვისები, იმუშაოს stream სერვერად და ა.შ.

განვიხილოთ რამდენიმე მათგანი:

1. **Windows Small Business Server** [94].

SBS შეიცავს Windows Server-ს და სხვა დამატებით ტექნოლოგიებს, რომლებიც გათვალისწინებულია პატარა ქსელისთვის. მაგალითად, ერთი-ერთი ასეთი ტექნოლოგიაა Remote Web Workplace.

SBS-ის სტანდარტული გამოცემა შეიცავს SharePoint სერვისებს, Microsoft Exchange სერვერს, Fax სერვერს და Active Directory-ს მომხმარებელთა სამართავად. ეს პროდუქტი ასევე გვთავაზობს firewall-ს, DHCP სერვერს და NAT როუტერს.

SBS-ის პრემიუმ გამოცემა შეიცავს ყველაფერს, რაც აქვს სტანდარულ გამოცემას. ასევე შეიცავს Microsoft SQL Server 2000-ს და Microsoft Internet Security and Acceleration Server 2004-ს.

SBS-ს აქვს საკუთარი ტიპის CAL (Client Access License), რომელიც არის განსხვავებული და ღირს უფრო ძვირი ვიდრე Windows Server 2003-ის სხვა ვერსიებში.

SBS სერვერს აქვს შემდეგი შეზღუდვები:

- მხოლოდ ერთ კომპიუტერზე დომეინში შეიძლება იყოს დაყენებული Windows Server 2003 for Small Business Server;

- Windows Server 2003 for Small Business Server უნდა იყოს root-ი Active Directory-ში;
- Windows Server 2003 for Small Business Server-ს არ აქვს "ნდობა" სხვა დომეინებთან;
- Windows Server 2003 for Small Business Server-ს აქვს 75 user-ის ან device-ს ლიმიტი. ეს დამოკიდებულია CAL-ის ტიპზე;
- Windows Server 2003 for Small Business Server-ს აქვს 4GB RAM(Random Access Memory)-ის ლიმიტი;
- Windows Server 2003 for Small Business Server-ის დომეინს არ შეუძლია ჰქონდეს child დომეინები;
- Terminal სერვერის ინსტალაციის შემდეგ, მხოლოდ ერთდროული 2 RDP კავშირის ლიმიტია;
- იმისათვის რომ აღარ გვქონდეს ეს ლიმიტები, მაშინ უნდა დავაყენოთ Windows Small Business Server 2003 R2 პაკეტი.

2. Web გამოცემა

Windows Server 2003-ის Web გამოცემა ძირითადად განკუთვნილია ვებ-აპლიკაციების, ვებ-გვერდების, XML სერვისების ასაწყობად და დასაჰოსტად. ეს გამოცემა არ საჭიროებს CAL-ებს და Terminal სერვერის რეჟიმი არ არის ამ გამოცემაში, მაგრამ Windows Server 2003 ვებ-გამოცემაში შეგვიძლია გამოვიყენოთ Remote Desktop-ი remote ადმინისტრირებისათვის. მხოლოდ 10 file-sharing კავშირი შეიძლება იყოს ერთდროულად. ამ გამოცემაში შეუძლებელია დაყენდეს Microsoft SQL Server და Microsoft Exchange, მაგრამ Service Pack 1-ის ინსტალაციის შემდეგ უკვე შეიძლება დაყენდეს MSDE და SQL Server 2005 Express. ასევე ამ გამოცემას არ აქვს UDDI-ს მხარდაჭერა.

Windows Server 2003 Web გამოცემას აქვს მაქსიმუმ 2 პროცესორის მხარდაჭერა და 2GB ოპერატიული მეხსიერების. ამ გამოცემას არ შეუძლია იმუშაოს დომეინ კონტროლერად. ეს არის

ერთადერთი ვერსია Windows Server 2003-ის რომელიც არ საჭიროებს CAL-ებს.

3. სტანდარტული გამოცემა

Windows Server 2003-ის სტანდარტული გამოცემა გათვლილია პატარა და საშუალო ზომის ქსელებისთვის. სტანდარტულ გამოცემას აქვს ფაილების და პრინტერების განაწილების შესაძლებლობა, გვთავაზობს დაცულ ინტერნეტ კავშირს. ამ გამოცემას აქვს 4 პროცესორისა და 4GB RAM-ის მხარდაჭერა, ხოლო მის 64-ბიტიან ვერსიას კი – 32GB RAM-ის და ასევე Non-Uniform Memory Access (NUMA)-ს მხარდაჭერა. 32-ბიტიანი Windows Server 2003-ის სტანდარტული გამოცემის გადმოწერა უფასოდ შეუძლიათ სტუდენტებს Microsoft-ის DreamSpark საიტიდან.

4. Enterprise გამოცემა

Windows Server 2003-ის Enterprise გამოცემა გათვლილია საშუალო ან დიდი ზომის ქსელებისთვის. ამ გამოცემას აქვს 8 პროცესორის მხარდაჭერა, eight-node clustering-ის მხარდაჭერა Microsoft Cluster Server (MSCS)-ის გამოყენებით. ასევე მას აქვს 32GB RAM-ის მხარდაჭერა PAE-ს ჩართვის შემდეგ. Enterprise გამოცემა ასევე გამოდის 64-ბიტიანები. მათ აქვთ 1TB ოპერატიულის მხარდაჭერა. ორივეს 32-ბიტიანებსაც და 64-ბიტიანებსაც აქვთ Non-Uniform Memory Access (NUMA)-ს მხარდაჭერა.

ამ გამოცემას აქვს შემდეგი შესაძლებლობების მხარდაჭერა:

- Microsoft Metadirectory Services (MMS)-ების მხარდაჭერა, რომლის საშუალებითაც ხდება მრავალი დირექტორიების, მონაცემთა ბაზების და ფაილების ინტეგრაცია Active Directory-ასთან;

- აქვს Hot Add Memory-ის მხარდაჭერა, რაც იმას ნიშნავს, რომ შეგვიძლია ჩავამატოთ ან ამოვიღოთ RAM სისტემიდან გადატვირითვის ან გამორთვის გარეშე;
- აქვს WSRM (Windows System Resource Manager)-ის მხარდაჭერა.

5. Datacenter გამოცემა

Windows Server 2003-ის Datacenter გამოცემა განკუთვნილია ისეთი გარემოსთვის, რომელიც მოითხოვს მაღალ დაცულობას და საიმედოობას. Windows Server 2003-ის Datacenter გამოცემა არის როგორც 32-ბიტიანი, ასევე 64-ბიტიანი. 32-ბიტიანს აქვს 32 პროცესორის მხარდაჭერა, ხოლო 64-ბიტიანს – 64 პროცესორის. 32-ბიტიან არქიტექტურას აქვს 128GB RAM-ის ლიმიტი, ხოლო 64 ბიტიანს - 2TB.

Windows Server 2003-ის Datacenter გამოცემას ასევე აქვს NUMA-ს და 8-node clustering-ის მხარდაჭერა. მას ასევე აქვს უკეთესი მხარდაჭერა Storage Area Networks (SAN)-ისგან.

1.8. აპლიკაცია-Active Directory, როგორც ტექნოლოგია და მისი დანერგვის გეგმარება

Active Directory არის Windows NT ოპერაციული სისტემების ოჯახის სერვერის დამოუკიდებელი ნაწილი, აქტიური კატალოგი [95,96]. იგი პროცესებისა და სერვისების ნაკრებია, რომელიც შეიმუშავა მაიკროსოფთის ფირმამ დომენების ქსელისთვის Windows-ის ბაზაზე [96,97]. მონაცემთა დამუშავების ორგანიზაციული მეთოდი ახარისხებს ინფრომაციას და გვაძლევს საშუალებას სწრაფად და ეფექტურად მოვიძიოთ სასურველი ინფორმაცია. კატალოგური სერვისები ხშირ შემთხვევაში შედარებულია სატელეფონო წიგნთან. ესაა ინფორმაციის

ერთობლიობა, რომელიც ორგანიზებული და დახარისხებულია შემდეგი კრიტერიუმებით: გვარი, სახელი, ტელეფონის ნომერი, ქალაქი და ქვეყანა. რადგან ეს ინფორმაცია ამ კერძო გზით არის ორგანიზებული, ჩვენ სწრაფად შეგვიძლია ვიპოვოთ ნებისმიერი ადამიანის ტელეფონის ნომერი და სხვა დამატებითი ინფორმაცია, რაც ამ სატელეფონო წიგნშია მოცემული.

კატალოგები, რა თქმა უნდა, არ არის რაღაც ახალი, ისინი გამოიყენება ზოგადად, როგორც წიგნები. მაგრამ ქსელური ტერმინი დირექტორია (სერვისების კატალოგი), ქსელური ტექნოლოგიის მნიშვნელოვანი ტერმინია.

რა არის სერვისების კატალოგი (Directory Service)?

Active Directory არ არის პირველი სერვისების კატალოგი, რომელიც გამოჩნდა ბაზარზე. უბრალოდ ქსელები და შესაბამისად მისი სერვისები არ იყო იმდენად მნიშვნელოვანი რამდენიმე წლის წინ, როგორც დღეს. მართალია, ადრეც არსებობდა დიდი ბიზნესი განსაკუთრებით დიდი მოცულობის მონაცემებით, მაგრამ პერსონალური კომპიუტერების მასობრივმა გავრცელებამ და მისმა გამოყენებამ ეს მაჩვენებელი მნიშვნელოვნად გაზარდა.

განვიხილოთ Directory service. სერვისების კატალოგიის მიზანია მოაწესრიგოს დიდი და პატარა ქსელები. იგი უზრუნველყოფს გამარტივებულ მიდგომას ქსელსა და რესურსების აღმოჩენაში. დირექტორიებში მომხმარებელს საშუალება ეძლევა სწრაფად და მარტივად მოიძიოს საჭირო ინფორმაცია. Active Directory არის Microsoft-ის პასუხი დღევანდელი ქსელური საჭიროებებისათვის:

1. ორგანიზებული მიდგომა;
2. მარტივი ადმინისტრირება;
3. მომხმარებელთა ტოპოლოგიის საჭიროების გაუქმება;
4. ზრდის პოტენციალი;
5. სტანდარტიზაცია;

6. ქსელის კონტროლი;
7. გამარტივებული WAN-ის მართვა;

- **Active Directory-ის ლოგიკური სტრუქტურა**

ვიდრე Active Directory-ის შევისწავლით, თვალი გადავავლოთ მის ლოგიკურ სტრუქტურას. იმისათვის რომ ეფექტურად დავგეგმოთ, განვახორციელოთ და ადმინისტრირება გავუწიოთ Active Directory-ის, მისი ლოგიკური სტრუქტურა კარგად უნდა გვქონდეს გათავისებული.

Active Directory დაშენებულია დომეინურ დონეზე (Windows domain networks) [97]. დომეინი კი ეს არის კომპიუტერების და მომხმარებლების ლოგიკური დაჯგუფება ადმინისტრირებისა და უსაფრთხოების მიზნით.

მაგალითად, Windows NT-ში, უნდა დაგვეკარგა ძალიან დიდი დრო, რომ აღმოგვეფხვრა დომეინის შეცდომები. ნებისმიერი ზომის Windows NT ქსელისთვის უნდა გქონოდათ რამდენიმე დომეინი. NT დომეინები იყო ძალიან შეზღუდული ადმინისტრირების კუთხით და დიდი სირთულეები მოჰქონდა როგორც მომხმარებლებისთვის, ასევე ადმინისტრატორებისათვის.

Windows Active Directory დომეინის მოდელი არის განსხვავებული რამ. იგი ისეა ორგანიზებული (განლაგებული) როგორც „ხეები ტყეში“. როდესაც ვაინსტალირებთ Active Directory-ის, ჩვენ ვქმნით ახალ დომეინს ტყეში. ახალი დომეინი ხდება ძირი დომეინი. თუ ჩვენ დავაინსტალირებთ დამატებით დომეინს, ისინი იქმნება ტყის საფუძველიდან.

ჩვენ შეგვიძლია შევქმნათ დამატებითი ხე დომეინები ერთ ტყეში. ანუ სხვა სიტყვებით რომ ვთქვათ, ტყე შეიძლება შეიცავდეს ერთ ან ერთზე მეტ ხე დომეინებს. Active Directory უპირატესობას ანიჭებს რამდენიმე დომეინს სხვადასხვა ორგანიზაციული

ერთეულებით (ოე). ორგანიზაციულ ერთეულებად შეგვიძლია ჩავთვალოთ ფოლდერები, რომლებიც შეიცავს მნიშვნელოვან ინფორმაციას.

ორგანიზაციული ერთეული არის კონტეინერი და ინახავს ყველა ტიპის Active Directory რესურსს, როგორიცაა: მომხმარებელი, კომპიუტერი, პრინტერი და სხვა.

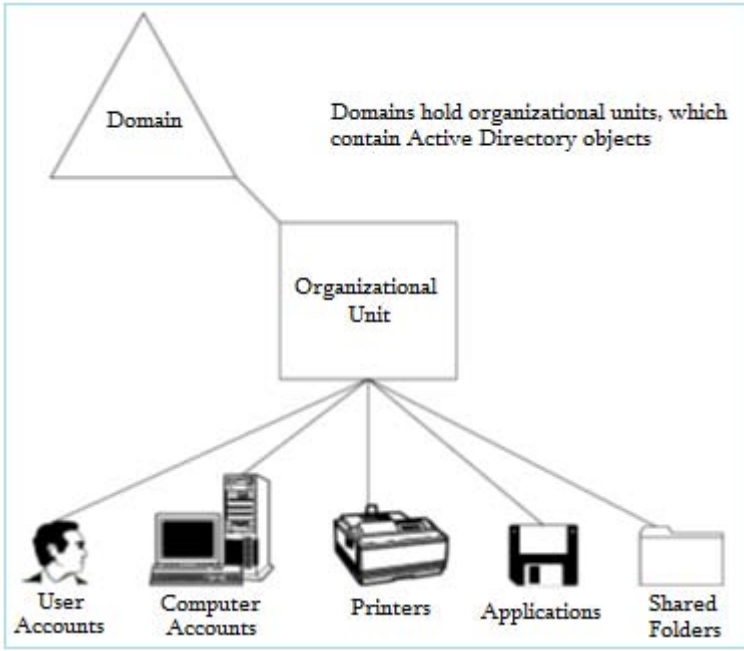
საუკეთესო საშუალება რაც ორგანიზაციულ ერთეულს აქვს, არის უსაფრთხოება, ადმინისტრაციული კონტროლი ოე-ს დონეზე. უკვე შეგვიძლია მრავალი NT ქსელი, რომელსაც აქვს რამდენიმე დომეინი, შევცვალოთ ერთი დომეინით და მასში შემავალი რამდენიმე ორგანიზაციული ერთეულით (ნახ. 1.23).

შესაბამისად, ლოგიკურ სტრუქტურაში გვაქვს დომეინი, ორგანიზებული ერთეულები და ობიექტები, თითოეული მათგანის განმსაზღვრელი ატრიბუტებით. Active Directory ასევე იყენებს საიტებს, მაგრამ ეს საიტები არ მოიაზრება Active Directory-ის ლოგიკური სტრუქტურის ან იერარქიის ნაწილად.

Active Directory-ში საიტები შექმნილია მხოლოდ რეპლიკაციისათვის და ტრაფიკის კონტროლის მიზნით.

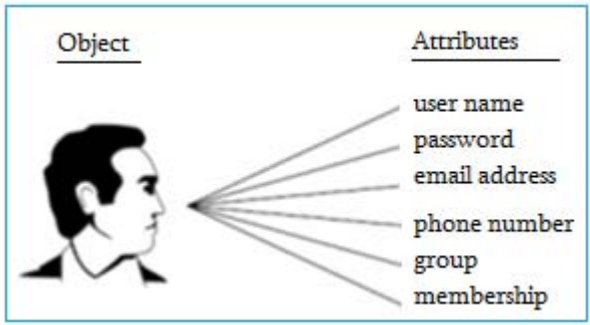
საიტი თავისი განმარტებით არის ფიზიკური მდებარეობა კომპიუტერების და მომხმარებლების დომეინის შემადგენლობაში, რომელიც თავის მხრივ არის კომპიუტერების და მომხმარებლების ლოგიკური ჯგუფი.

მნიშვნელოვანია აღინიშნოს, რომ საიტი და დომეინი არ არის ურთიერთდაკავშირებული.



ნახ. 1.23-ა.

Active Directory შეგვიძლია წარმოვიდგინოთ როგორც მონაცემთა ბაზა, ობიექტებით და ატრიბუტებით.



ნახ.1.23-ბ. ობიექტი და ატრიბუტები

საიტი შეიძლება შეიცავდეს რამდენიმე დომეინს ან დომეინი ირგებდეს მრავალ საიტს. რადგან დომეინი არის ლოგიკური ადმინისტრაციული დონე ქსელურ არქიტექტურაში, საიტი არის ფიზიკური დონე. Active Directory-ში საიტები დაფუძნებულია IP subnet-ებზე და მათ შეუძლია იყოს გეგმარების და კონფიგურირების ამრევი.

- **DNS და LDAP**

Active Directory არის სრულად თავსებადი Domain Name System (DNS) და Lightweight Directory Access Protocol (LDAP) სისტემებთან. DNS არის სახელის გაფართოების მეთოდი, იმისათვის რომ მოხდეს IP მისამართებზე ჰოსტების სახელების მინიჭება.

DNS გამოიყენება TCP/IP ქსელებში და არის სისტემის გაფართოების სახელი, რომელიც გამოიყენება მთლიანად ინტერნეტში. იგი ააქტიურებს ჰოსტის სახელს, მაგალითად, www.microsoft.com და წყვეტს, რომ იგი იყენებდეს TCP/IP მისამართს, მაგალითად, 131.107.2.200.

კომპიუტერები კომუნიკაციისათვის იყენებს IP მისამართებს. ადამიანებისათვის რთულია IP მისამართების დამახსოვრება, ისინი არიან მეტყველებაზე დაფუძნებული არსებები. ამიტომ DNS გვაძლევს საშუალებას, ჰოსტების სახელები დავიმახსოვროთ ადამიანურ ენაზე, მაგალითად, microsoft.com.

რა კავშირი აქვს DNS-ს Active Directory-სთან ?

Active Directory აგებულია DNS-ზე, სინამდვილეში Active Directory-ის სახელები არის DNS სახელები.

- **Windows Domain Controllers**

Windows დომეინ კონტროლერები ფუნქციონირებს როგორც თანაბარი დონის დომეინები. ეს ნიშნავს იმას, რომ აღარ არის

პირველადი დომეინ კონტროლერი. ყველა დომეინ კონტროლერი არის უბრალოდ „დომეინ კონტროლერი“ და ყველანი თანაბარი დონისაა. სწორედ ამიტომ შეგვიძლია გამოვიყენოთ ნებისმიერი დომეინ კონტროლერი Active Directory-ში ცვლილებების შესატანად. ეს არის მართვის საუკეთესო საშუალება. იმ შემთხვევაშიც კი, თუ ერთი რომელიმე კონტროლერი გამოვა მწყობრიდან, პრობლემა არ ჩნდება, რადგან სხვა კონტროლერები ჩვეულ რიტმში აგრძელებს მუშაობას და ქსელი ფუნქციონირებს ნორმალურად.

- **გლობალური კატალოგ სერვერები**

ზოგი დომეინ კონტროლერი არის გლობალური კატალოგ სერვერი. იგი დამოკიდებულია იმაზე, თუ როგორაა ჩვენი ქსელი დაკონფიგურირებული. შესაძლებელია გვქონდეს რამდენიმე გლობალური კატალოგ სერვერი. ეს სერვერები ასრულებს 2 მთავარ ფუნქციას:

გლობალური კატალოგ სერვერები შეიცავს სრულ ასლს (კოპიო) ყველა Active Directory ობიექტებისას მათ დომეინში და ნაწილობრივ ასლს ყველა Active Directory ობიექტებისას სხვა დომეინში, რომელიც ტყეშია.

გლობალური კატალოგ-სერვერები გამოიყენება მომხმარე-ბელთა დაშვებისათვის სისტემაში. ისინი ემსახურება მომხმარებელთა Log-ინებს და ამით გვაწოდებენ ინფორმაციას უნივერსალური ჯგუფების შესახებ.

1.9. ობიექტ-როლური მოდელირება (ORM) განაწილებული სისტემებისთვის

განაწილებული ინფორმაციული სისტემების ფუნქციურობის განსაზღვრა კონცეპტუალურ დონეზე ხდება, სადაც გამოიყენება

ისეთი კონცეფციები და ენა, რომელიც ადვილი გასაგებია ადამიანისათვის [37]. 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. ობიექტროლური 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].

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

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.

ORM-ის კონცეპტუალური მოდელირების სქემის პროცედურა (*CSDP*) ყურადღებას ამახვილებს მონაცემების ანალიზზე და დაპროექტებაზე [52].

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 (ცხრ.1.2).

ცხრ.1.2

1. ელემენტარული ფაქტების ფორმირება და მათი ადეკვატურობის 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 ‘12.31.14’**

თითოეული ფაქტი არის ბინარული დამოკიდებულება ორ ობიექტს შორის. მუქი შრიფტით გამოყოფილია ლოგიკური პრედიკატი, რომელიც ახდენს ობიექტების იდენტიფიცირებას და 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. ასე, 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 [52]:

<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.

სიმრავლეების შედარების შეზღუდვები:

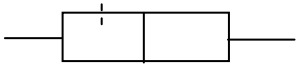
⊆ - პირველი ობიექტის სიმრავლე ყოველთვის უნდა იყოს მეორის ქვესიმრავლე.

= - პირველი ობიექტის სიმრავლე ყოველთვის უნდა იყოს მეორის ტოლი.

⊄ - პირველი ობიექტის სიმრავლე არ შედის მეორეში.

<=2 **სიხშირის შეზღუდვა**

ობიექტმა რამდენჯერ შეიძლება შეასრულოს ეს როლი



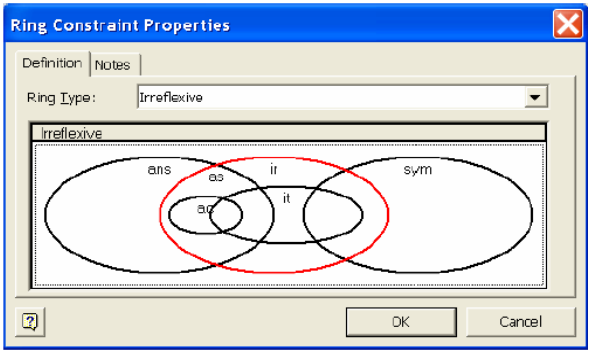
ბუდის ტიპის ობიექტი

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

ქვეტიპი

ერთი ობიექტი არის მეორის ქვეტიპი

წრიული შეზღუდვები:



ნახ. 1.23.

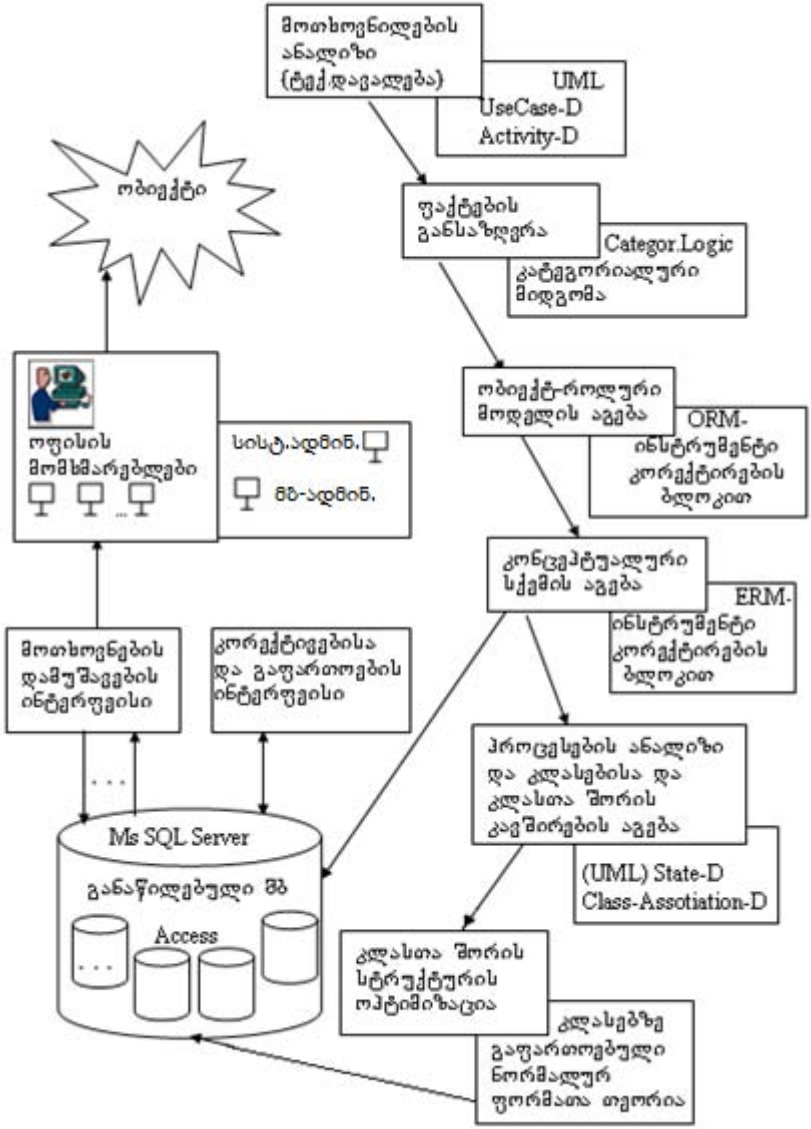
ანტირეფლექსურობა	**Oir**	**iff for all x, ˜xRx**
სიმეტრიულობა	**Osym**	**iff for all x, y, xRy → yRx**
ასიმეტრიულობა	**Oas**	**iff for all x, y, xRy → ˜yRx**
ანტისიმეტრიულობა	**Oans**	**iff for all x, y, x ≠ y & xRy → ˜yRx**
ანტიტრანზიტულობა	**Oit**	**iff for all x, y, z, xRy & yRz → ˜xRz**
აციკლურობა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

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

1.24 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 *.DD*L ფაილები, რომლებიც შემდგომ სტრუქტურულად მოთავსდება *Ms SQL Server* მონაცემთა განაწილებულ ბაზაში.



ნახ.1.24. საპრობლემო სფეროს მოდელირების ეტაპები

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. 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) [69].

ნაშრომი ეფუძნება *UML*–ტექნოლოგიისა და მონაცემთა ლოგიკური სტრუქტურების რელაციურ დამოკიდებულებათა თეორიის კლასიკურ საკითხებს. ასეთი ინტეგრირებული მიდგომა დასმული ამოცანის გადასაწყვეტად უზრუნველყოფს, ერთი მხრივ, მონაცემთა სტრუქტურების ოპტიმიზაციას ნორმალურ ფორმათა თეორიის საფუძველზე და, მეორე მხრივ, დაპროექტების ობიექტ-ორიენტირებული 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;

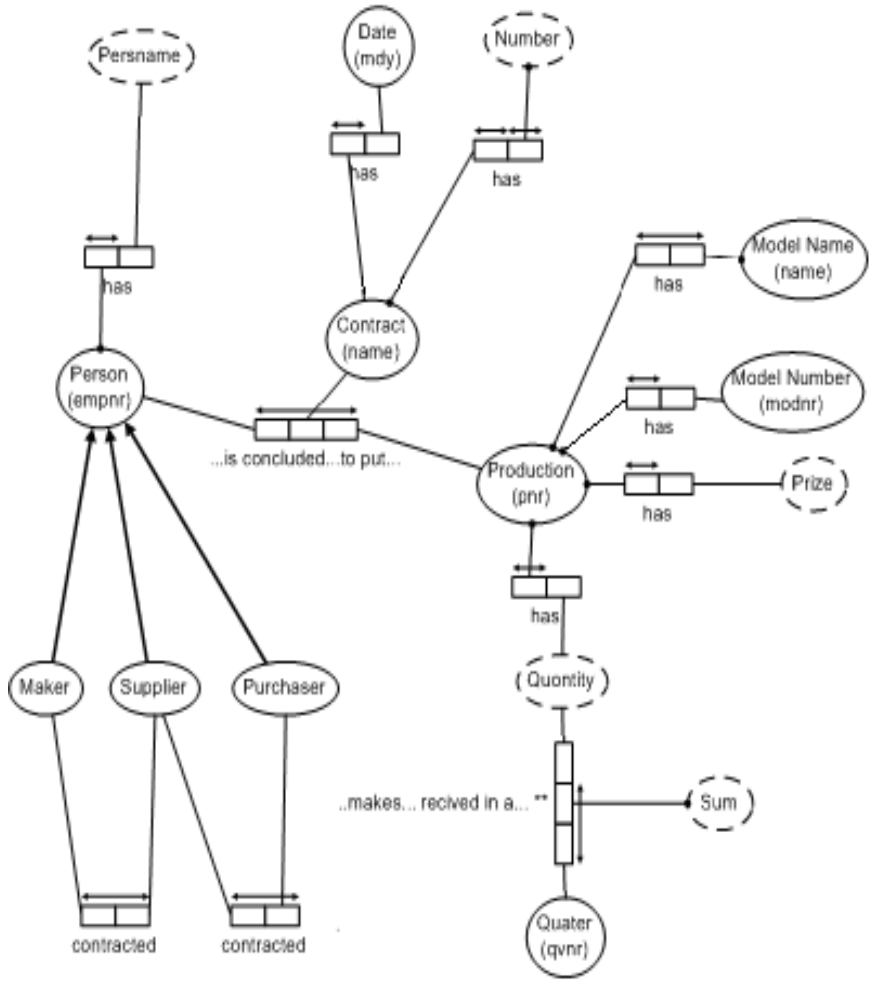
f11 - სერვისს (პროდუქციას) აქვს მოდელის ნომერი;

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 1.25. ნახაზზე.



ნახ.1.25. „კონტრაქტის" *ORM*-მოდელის ფრაგმენტი

1.10. 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 მეცნიერული კვლევების შედეგად შეიქმნა პეტრის ქსელების სხვადასხვა კლასები, რომლებსაც ერთმანეთთან მჭიდრო კავშირი აქვს და მრავალი ცალკეული ტიპის პეტრის ქსელისაგან 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).

ფერადი 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 ავტომატიზებული ანალიზის შესაძლებლობას;
- 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 (მაგალითად, ორ- და სამგანზომილებიან გრაფიკას) [57].

1.11. მულტიმედიური მონაცემთა ბაზების დაპროექტებისა და გამოყენების ამოცანა ელექტრონული საარჩევნო სისტემის ასაგებად

ამოცანა მდგომარეობს ისეთი ელექტრონული საარჩევნო სისტემის შექმნაში, რომელიც უზრუნველყოფს არსებული სისტემის პირველ ეტაპზე ნაწილობრივ სახეცვლილებას და მოამზადებს ნიადაგს სრული ჩანაცვლებისათვის.

ახალი ელექტრონული საარჩევნო სისტემის ძირითადი დანიშნულებაა მოხდეს:

- საარჩევნო სიების სრულყოფა;
- საარჩევნო სიაში არსებული ყველა ამომრჩევლის შესახებ ამომწურავი ინფორმაციის მოგროვება და გამდიდრება;
- საარჩევნო პროცესის ავტომატიზაცია, რაც სამომავლოდ გაუიაფებს სახელმწიფოს არჩევნების ჩატარების ხარჯებს;

დისერტაციის მიზანია ელექტრონული საარჩევნო სისტემის ბიზნესპროცესების მართვის დაპროექტება და რეალიზაცია სერვისორიენტირებული არქიტექტურით.

მიზნის მისაღწევად ნაშრომში განიხილება შემდეგი ძირითადი ამოცანები:

- არსებული თანამედროვე ელექტრონული საარჩევნო სისტემების ანალიზი და შესაბამისი ინფორმაციული ტექნოლოგიების კლასიფიკაცია, ობიექტ-, პროცეს- და სერვის-ორიენტირებული დაპროექტების პრინციპებით;

- საარჩევნო სისტემის ბიზნესპროცესების ტრადიციული და სერვისორიენტირებული მოდელების აგება სისტემური ანალიზის საფუძველზე, BPMN და UML ტექნოლოგიების ბაზაზე;

- მულტიმედიური მონაცემთა ბაზების დაპროექტება და აგება კლიენტ-სერვერული ტექნოლოგიის გამოყენებით SQL Server-ის ბაზაზე;

- სერვისორიენტირებულ მონაცემთა განაწილებული ბაზების სტრუქტურების დასაპროექტებლად ობიექტროლური მოდელების (ORM) აგება და კვლევა რევერსიული CASE ტექნოლოგიების გამოყენებით;

- კლიენტ-სერვერ არქიტექტურით სისტემის ინფორმაციული ბიზნესპროცესების იმიტაციური მოდელის აგება და მისი ფუნქციონირების დროითი მახასიათებლების კვლევა ფერადი პეტრის ქსელების (CPN) გამოყენებით;

- ელექტრონული საარჩევნო სისტემისათვის ერთიანი სახელმწიფო დაცული ქსელის დაპროექტება და აგება VPN ტექნოლოგიის გამოყენებით;

- პროექტის შედეგების საფუძველზე ესპერიმენტული პროგრამული სისტემის რეალიზაცია .NET პლატფორმაზე, C#.NET, Natural ORM Architect და SQL Server პროგრამული პაკეტების გამოყენებით.

II თავი
მულტიმედიური მონაცემთა ბაზების დამუშავება ელექტრონული საარჩევნო სისტემისთვის

2.1. ტრადიციული და ელექტრონული საარჩევნო სისტემების ბიზნესპროცესების აღწერა BPMN ენაზე

გთავაზობთ ტრადიციული და ელექტრონული საარჩევნო სისტემების ბიზნესპროცესებს, რომლებიც დამუშავებულია ავტორების მიერ და ასახულია BPMN–მოდელებით (ნახ.2.1, 2.2).

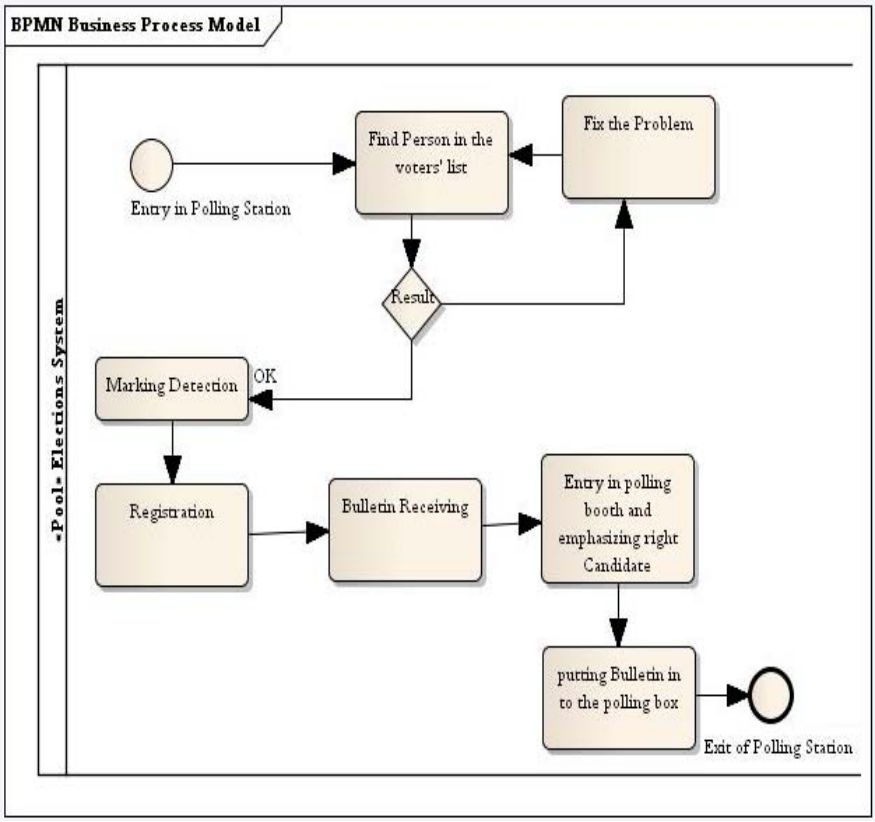
ტრადიციული სისტემით გათვალისწინებულია შემდეგი პროცედურები: 1. ამომრჩეველთა ერთიან სიაში თავის პოვნა; 2. მარკირების დეტექცია; 3. რეგისტრაციის გავლა; 4. საარჩევნო ბიულეტენის მიღება; 5. საარჩევნო კაბინაში შესვლა და სასურველი პირის შემოხაზვა; 6. ბიულეტენის საარჩევნო ურნაში ჩაგდება; 7. საარჩევნო უბნის დატოვება.

პირველი მოდელი გვიჩვენებს თუ რა პროცედურების გავლა უწევს საარჩევნო უბანზე მისულ ამომრჩეველს:

„მოქალაქე შედის საარჩევნო უბანზე და ეძებს თავის სახელსა და გვარს საარჩევნო უბანზე გამოკრულ სიაში. დადებითი შედეგის შემთხვევაში იგი აგრძელებს საარჩევნო პროცედურას და შედის მარკირების პროცედურაზე. უარყოფითი შედეგის შემთხვევაში, იგი ბრუნდება და იწყებს პრობლემის გამოსწორებას.

მარკირების შემდეგ მოქალაქე მიდის რეგისტრატორთან და გადის რეგისტრაციის პროცედურას, რაც გულისხმობს სახელის, გვარის და პირადი ნომერის ქაღალდის სიებში პოვნას და იდენტიფიცირებას. ამის შემდეგ იგი იღებს ბიულეტინს და მიემართება საარჩევნო უბნებზე მოწყობილ კაბინაში, ხაზავს

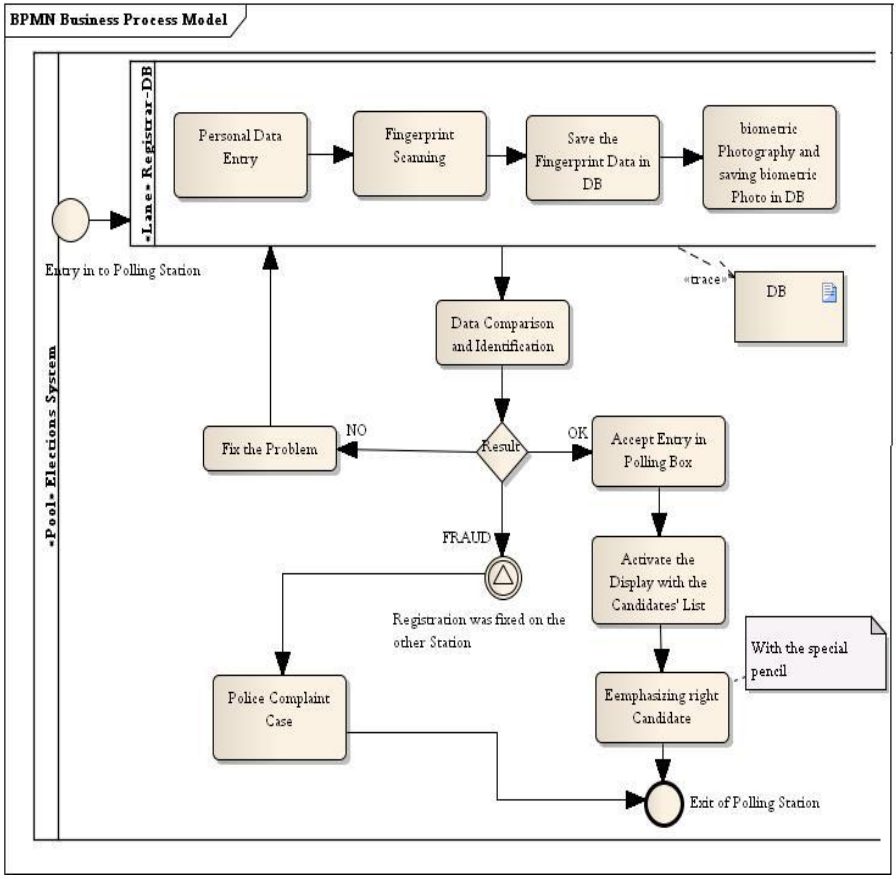
სასურველ კანდიდატს, ათავსებს კონვერტში ბიულეტინს და აგდებს საარჩევნო ყუთში. მოქალაქე გადის საარჩევნო უბნიდან“.



ნახ.2.1. ტრადიციული სარჩევნო სისტემა

ელექტრონული საარჩევნო სისტემა კი მოიცავს შემდეგ პროცედურებს: 1. დარეგისტრირება ნებისმიერ რეგისტრატორთან დადგენილი წესის დაცვით, მიიღებს დაშვებას საარჩევნო კაბინაში, სადაც გააქტიურდება სენსორული ეკრანი, რომელზეც გამოტანილი იქნება კანდიდატების ჩამონათვალი; 3. ხმის მიცემა სასურველი კანდიდატის გასწვრივ შემოიხაზება ნომერი სპეციალური კალმით; 4. საარჩევნო უბნის დატოვება.

2.2 ნახაზი ასახავს ელექტრონულ საარჩევნო სისტემას, რომელიც საბოლოო პროდუქტია და მოიცავს შემდეგ პროცედურებს: დარეგისტრირება ნებისმიერ რეგისტრატორთან შემდეგი წესის დაცვით, ტექსტური მონაცემების მონაცემთა ბაზაში შეყვანის შემდეგ თითის სკანირება და ანაბეჭდის ბაზაში გადატანა; ბიომეტრული ფოტოსურათის გადაღება და მონაცემთა ბაზის შესაბამის ველში მისი დამახსოვრება; ხმის ჩაწერა, ბაზაში ასახვა და ელექტრონული ხელმოწერის განხორციელება; ეს პროცედურა თავის თავში გულისხმობს არა მხოლოდ რეგისტრაციას, არამედ იდენტიფიკაციასაც. მას შემდეგ რაც წარმატებით გაივლის მოქალაქე იდენტიფიცირებას, მიიღებს დაშვებას საარჩევნო კაბინაში და გააქტიურდება სენსორული ეკრანი, რომელზეც გამოტანილი იქნება კანდიდატების ჩამონათვალი; მოქალაქე ირჩევს სასურველ კანდიდატს და მის გასწვრივ შემოხაზავს ნომერს სპეციალური კალმით; ამის შემდეგ ტოვებს საარჩევნო უბანს. იმ შემთხვევაში თუ მოქალაქის იდენტიფიცირების პროცედურისას წარმოიქმნება პრობლემები, იგი ბრუნდება უკან და მოხდება პრობლემის აღმოფხვრა, ხოლო იმ შემთხვევაში თუ ვერ გაივლის იდენტიფიცირებას, ანუ აღმოჩნდება რომ ის უკვე სხვა საარჩევნო უბანზეა ნამყოფი და დარეგისტრირებული, იგი მხილებული იქნება და აღნიშნული კეისის გარკვევაში ჩაერთვება პოლიცია და სამართალდამცავი ორგანოები.



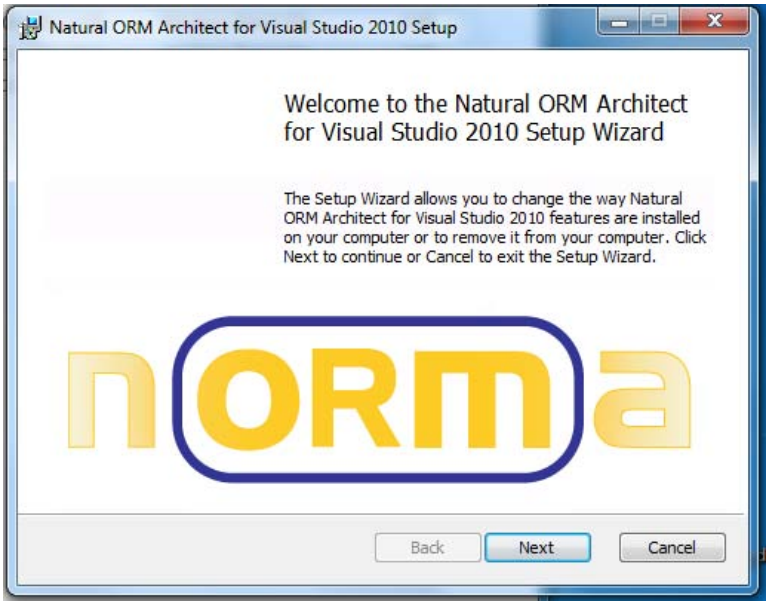
ნახ.2.2. ელექტრონული სარჩევნო სისტემა

2.2. სისტემის კონცეპტუალური მოდელის ავტომატიზებული აგება – ORM/ERM ტექნოლოგიით

მონაცემთა ბაზების ავტოამტიზებულ რეჟიმში დაპროექტება შესაძლებელია ობიექტროლური ORM (Object-Role Modeling) მოდელირების ტექნოლოგიის საშუალებით. Natural ORM Architect (NORMA) ინსტრუმენტი წარმოადგენს Microsoft Visual Studio.NET Framework-ის plugin-ს. მისი ჩამოტვირთვა შეიძლება საიტიდან:

www.ormfoundation.org/files/folders/norma_the_software/default.aspx

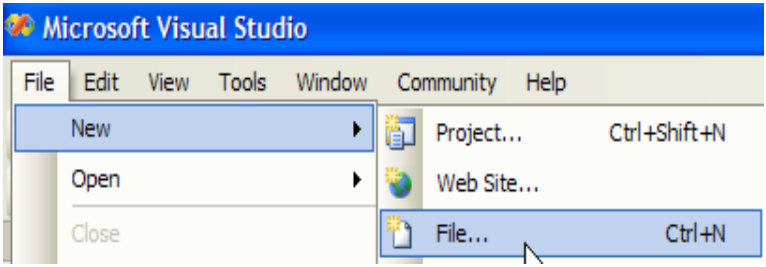
2.3 ნახაზზე მოცემულია სისტემის ინსტალაციის რეჟიმის დასაწყისი.



ნახ.2.3. NORMA ინსტრუმენტის ინსტალაციის რეჟიმი

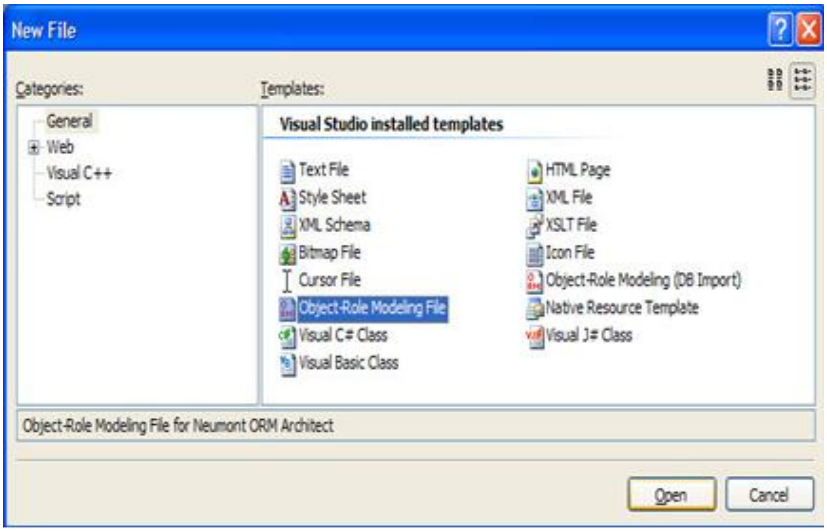
პაკეტის ინსტალაციის შემდეგ ფაილის გახნა ხდება ბრძანებით:

File --> New --> File



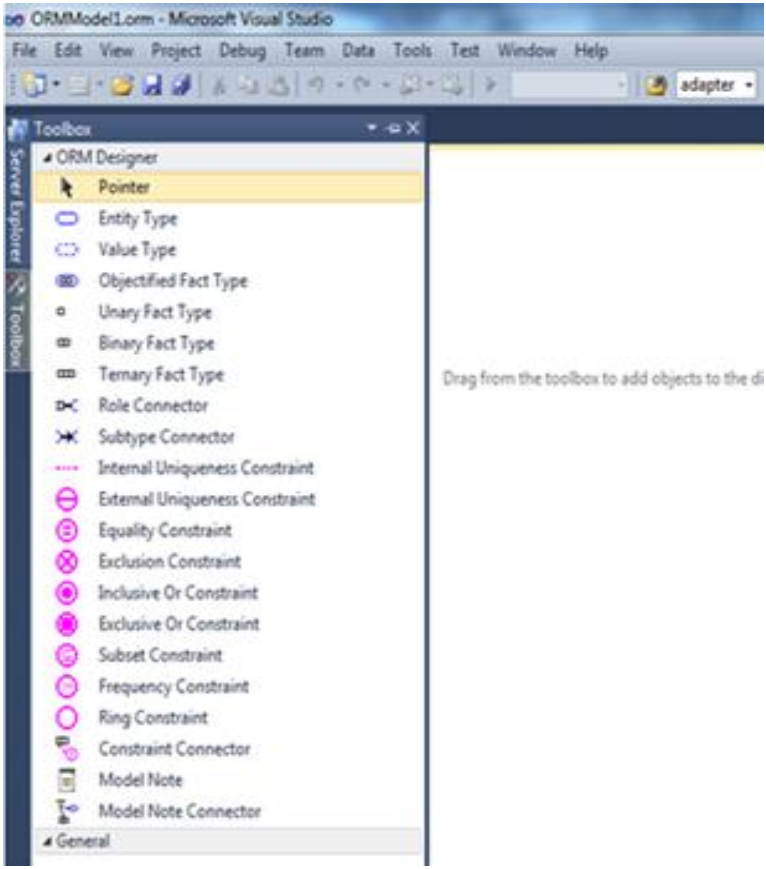
ნახ.2.4. ახალი ფაილის გახნა

Orm-დიაგრამის შესაქმნელად საჭიროა General კატეგორიიდან Object-Role Modeling File template-ის ამორჩევა.



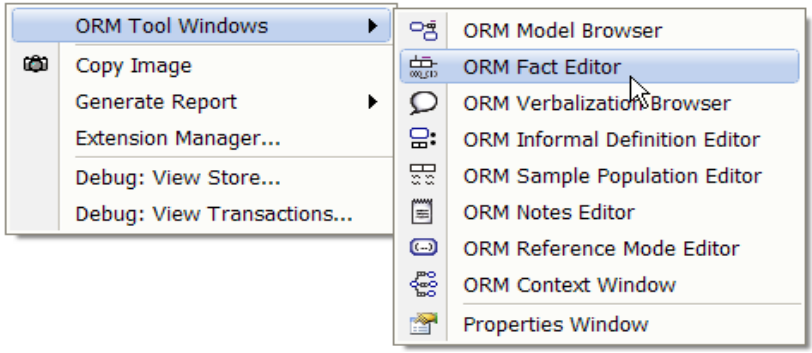
ნახ.2.5. ORM-ფაილის გახსნა

Orm-დიაგრამის აგება ხდება Document Window ფანჯარაში, სადაც სახეზეა დიაგრამის ასაგებად საჭირო ყველა ინსტრუმენტი: არსი (Entity), კავშირი (Connector) და შეზღუდვები (Constraints).



ნახ.2.6. ORM-დოკუმენტის ფანჯარა

ფაქტების შესატანად საჭიროა FACT EDITOR-ის გააქტიურება:



ნახ.2.7. ORM Fact Editor-ის ამორჩევა

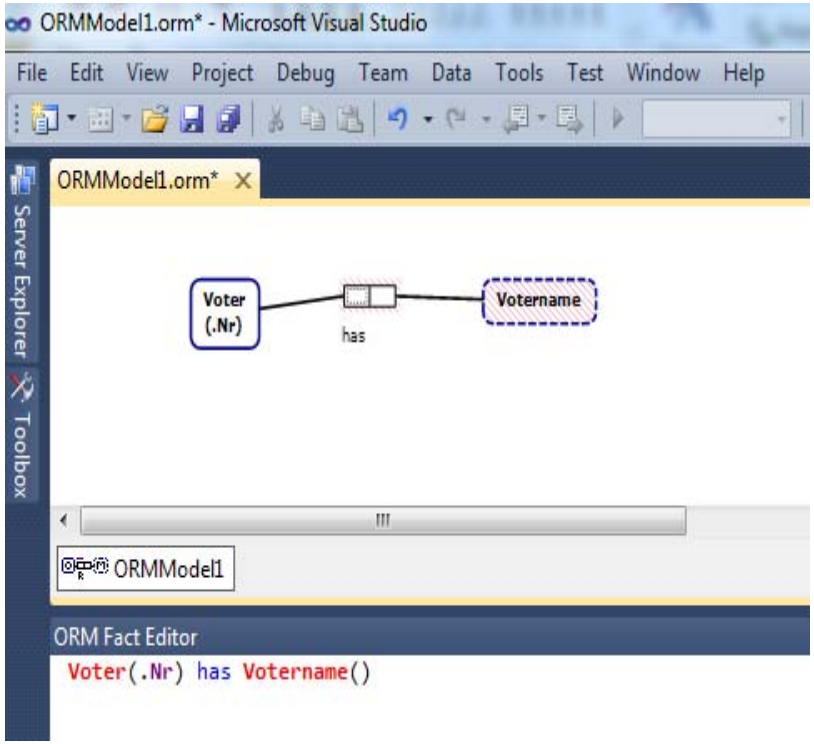
თავდაპირველად ხდება საპრობლემო სფეროს (კვლევის ობიექტის) მოთხოვნილებათა ანალიზი, ტექნიკური დავალების განსაზღვრის მიზნით, საიდანაც ჩამოყალიბდება ფაქტები.

ფაქტი არის კატეგორიული ცნება, რომელშიც აისახება ობიექტის სემანტიკური (შინაარსობრივი) შედგენილობა და სტრუქტურა. იგი ენის გრამატიკისა და ლოგიკური ალგებრის სიმბიოზია. ფორმალური სახით ფაქტის აღწერა ხდება შემდეგი სქემით:

„ობიექტი_1“ პრედიკატი „ობიექტი_2“ (ან „მნიშვნელობა“)

ესაა ორადგილიანი პრედიკატის ჩაწერის მაგალითი. შესაძლოა 3,4 და ზოგადად n-ადგილიანი პრედიკატების გამოყენებაც.

სწორედ ამ ელემენტარული ფაქტების საშუალებით განისაზღვრება ORM-მოდელი, რომლის მიხედვითაც შემდგომ აიგება ORM-დიაგრამა.



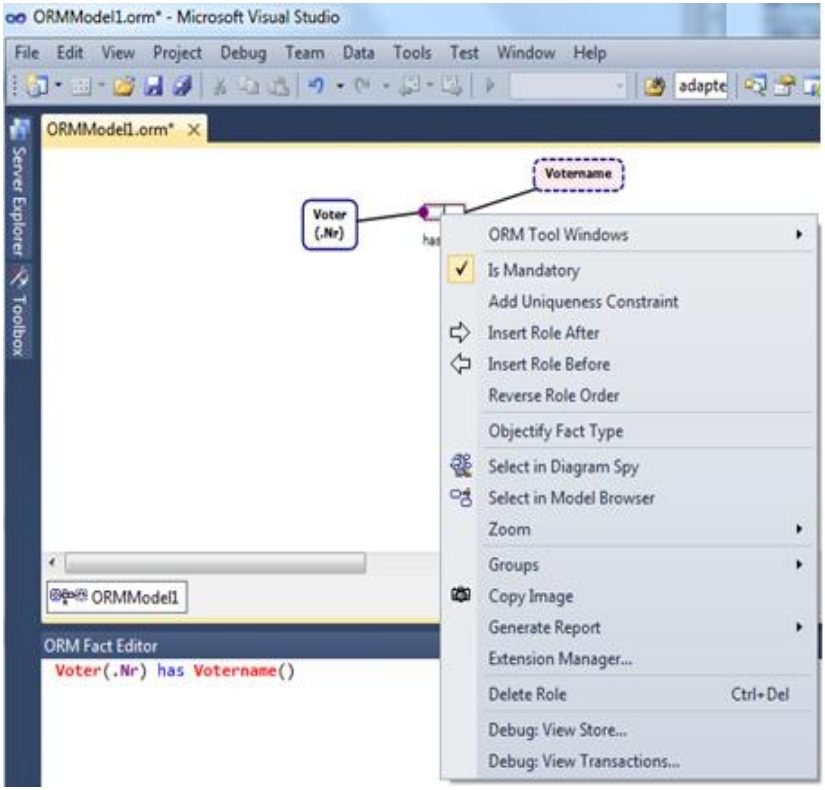
ნახ.2.8. ფაქტების შეტანა:

„ობიექტი“ 2-ადგილიანი პრედიკატი (has) „მნიშვნელობა“

ჩვენს მაგალითში ფაქტის სემანტიკური კონსტრუქცია აღწერს, რომ „ამომრჩეველს“ აქვს „გვარი_სახელი“.

სქემაზე შესაძლებელია სემანტიკური კონსტრუქციების დამატება, გაფართოება და შეზღუდვების დაყენება.

მაგალითად, ქვემოთ ნახაზზე ნაჩვენებია შეზღუდვის ჩამატება ჩვენს სქემაზე.

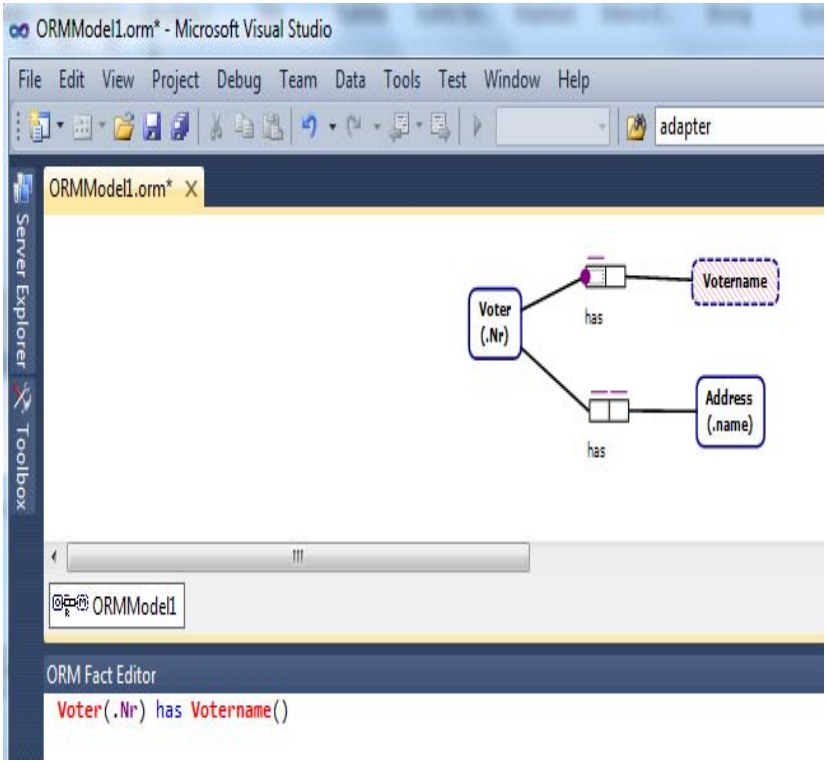


ნახ.2.9. „იძულების“ შეზღუდვის დამატება

„იძულების“ შეზღუდვით ხდება იმ კანონზომიერების განსაზღვრა, რომლის მიხედვითაც ერთ კონკრეტულ ამომრჩეველს (მოქალაქეს) აქვს მხოლოდ ერთი „ვინაობა“ (ამომრჩევლის ერთი ხმა). ასეთი შეზღუდვა გრაფიკულად აღინიშნება პრედიკატზე მრგვალი ბურთულით.

2.10 ნახაზზე ნაჩვენებია „უნიკალურობის“ შეზღუდვის გამოყენების მაგალითი, რომელიც ასახავს ფაქტს, რომ „ამომრჩეველს“ აქვს მხოლოდ ერთი (უნიკალური) „მისამართი“

(საარჩევნო უბნის და ნომრის უნიკალური იდენტიფიცირების მიზნით).



ნახ.2.10. „უნიკალურობის“ შეზღუდვის დამატება

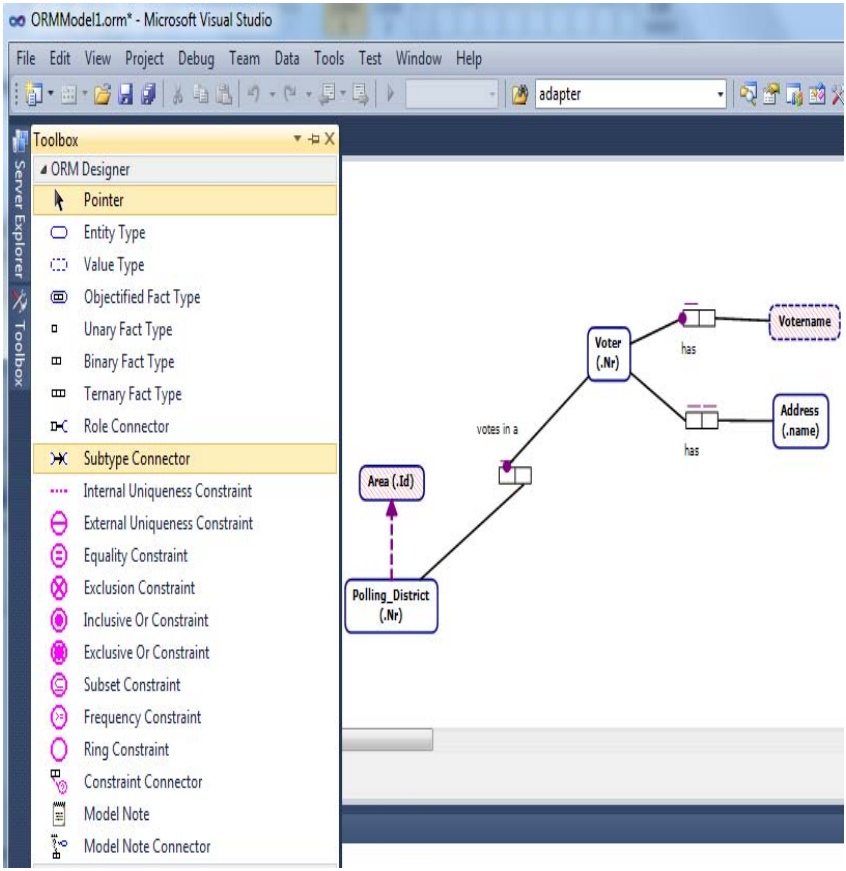
2.11 ნახაზზე მოცემულია „ქვეტიპის“ შეზღუდვის მაგალითი. ამგვარად, FACT EDITOR-ის ფანჯარაში შეგვაქვს დანარჩენი ფაქტები:

f1 Voter ***has*** VoterName

f2 Voter ***has*** FingerPrint

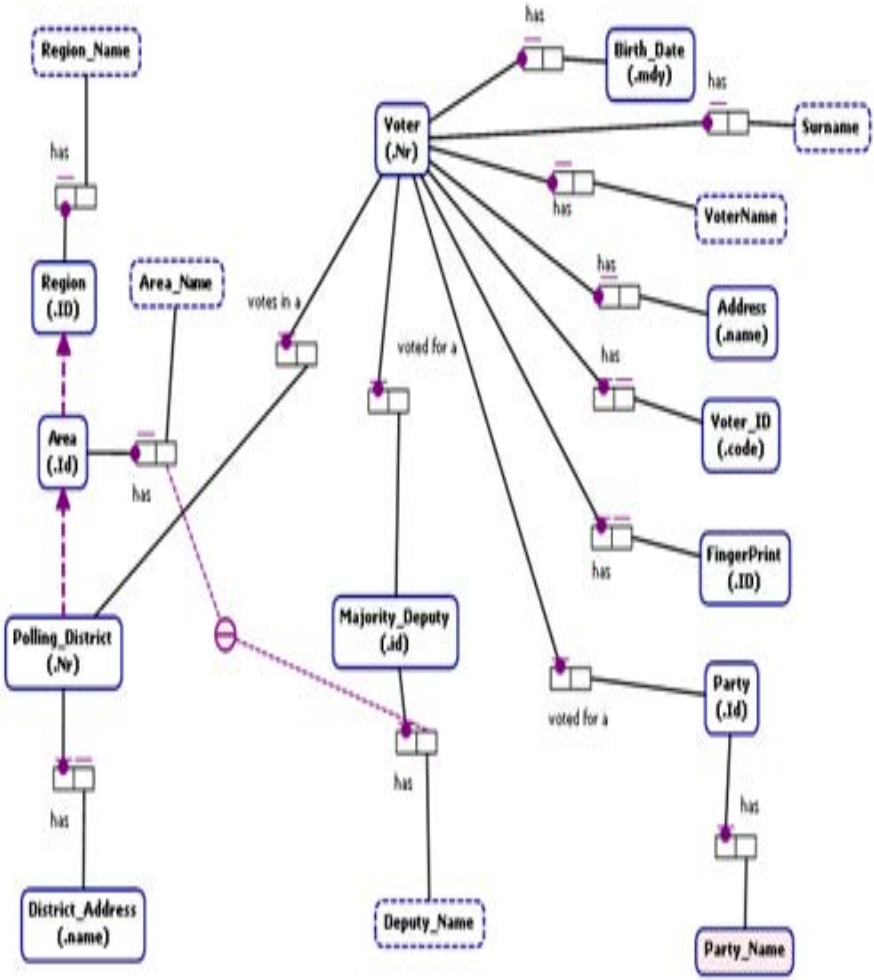
f3 Voter ***voted*** for a Majoritary_Deputy

f4 Voter ***has*** Voter_ID
f5 Voter ***has*** Address
f6 Voter ***has*** Surname
f7 Majoritary_Deputy ***has*** Deputy_Name
f8 Voter ***votes*** in a Polling_District …..
და ა.შ.



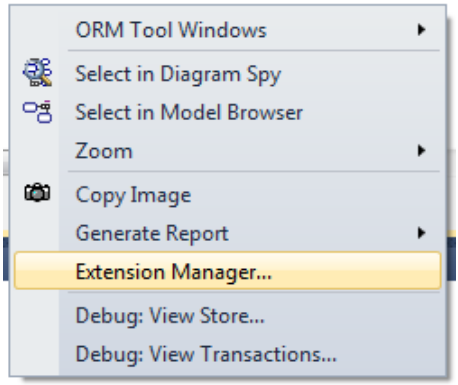
ნახ.2.11. „ქვეტიპის“ შეზღუდვის დამატება

ზემომოყვანილი ფაქტების შეტანის შემდეგ ORM-დიაგრამა მიიღებს შემდეგ სახეს:

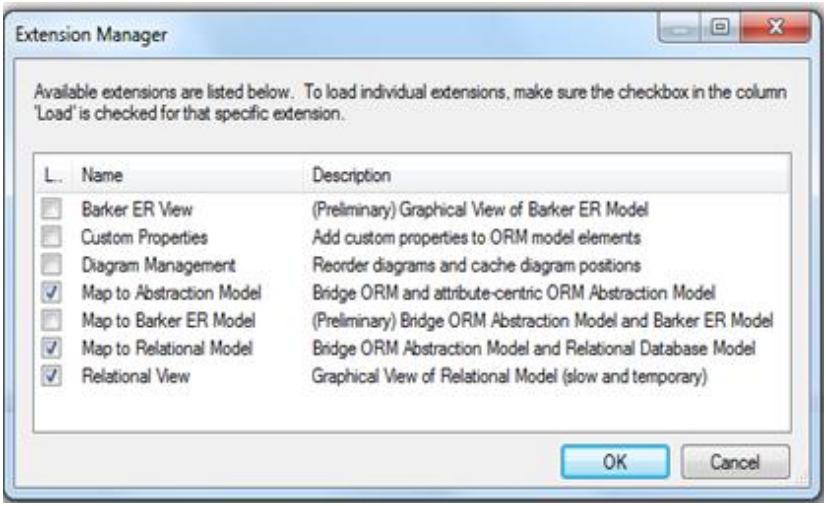


ნახ.2.12. ORM-დიაგრამის ფრაგმენტი

აგებული დიაგრამიდან ავტომატიზებულ რეჟიმში ვახდენთ არსთა-დამოკიდებულების მოდელის ER (Entities-Relationship Model) კონსტრუირებას Extension manager-ის საშუალებით (ნახ.2.13).

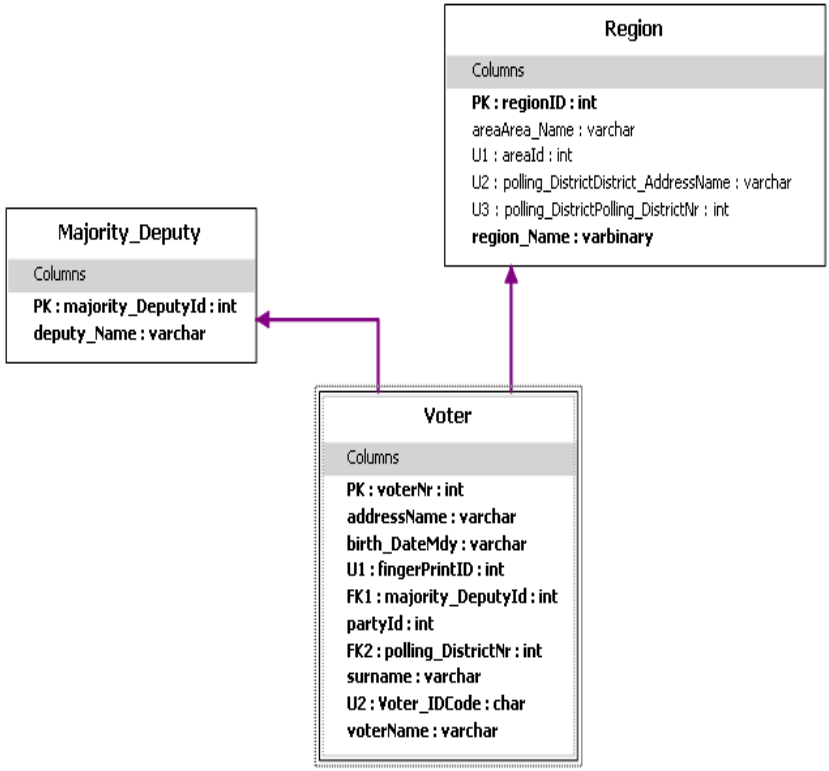


ნახ.2.13. Extension manager-ის გააქტიურება



ნახ.2.14. ER-მოდელის აგება ავტომატიზებულ რეჟიმში

ER-მოდელის შესაბამისი დიაგრამა ნაჩვენებია 2.15 ნახაზზე.



ნახ.2.15. ავტომატიზებულ რეჟიმში მიღებული ER-მოდელის ფრაგმენტი

დიაგრამაზე გამოსახულია სამი არსი (Entities) და ორი ლოგიკური რელაციური კავშირი, რომელთა პროგრამული რეალიზაცია ხორციელდება ცხრილების (Tables) და ცხრილთაშორისი კავშირებით პირველადი (Primary key) და მეორეული (Foreign key) გასაღებების გამოყენებით.

ჩვენს მაგალითში ამ მიზნით გამოყენებულია:

PK: majority_DeputyId,
PK: regionId,
PK: voterNr,
FK1: majority_DeputyId,
FK2:pooling_DistrictNr,
U1:fingerPrintID,
U2:voter_IDCode.

Natural ORM Architect პაკეტის საშუალებით, ასევე ავტომატიზებულ რეჟიმში, ვახდენთ მონაცემთა აღწერის ენის შესაბამის DDL-კოდის გენერაციას.

Visual Studio.NET პაკეტის Solution Explorer-ის საშუალებით შესაძლებელია SQL Server მონაცემთა ბაზისთვის ჩვენ მიერ გენერირებული კოდის ნახვა. მონაცემთა ბაზის ეს DDL-ფაილი შემდეგნაირად გამოიყურება:

```
CREATE SCHEMA ORMModel1
GO
GO
CREATE TABLE ORMModel1.Voter
(
  voterNr INTEGER NOT NULL,
  fingerPrintID INTEGER IDENTITY (1, 1) NOT NULL,
 Voter_IDCode NATIONAL CHARACTER(4000) NOT NULL,
 voterName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
 surname NATIONAL CHARACTER VARYING(MAX) NOT NULL,
 addressName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
 partyId INTEGER IDENTITY (1, 1) NOT NULL,
 birth_DateMdy NATIONAL CHARACTER VARYING(MAX) NOT NULL,
```

```
    majority_DeputyId INTEGER NOT NULL,
    polling_DistrictNr INTEGER NOT NULL,
    CONSTRAINT Voter_PK PRIMARY KEY(voterNr),
    CONSTRAINT Voter_UC1 UNIQUE(fingerPrintID),
    CONSTRAINT Voter_UC2 UNIQUE(Voter_IDCode)
)
GO

CREATE TABLE ORMModel1.Majority_Deputy
(
 majority_DeputyId INTEGER IDENTITY (1, 1) NOT NULL,
deputy_Name NATIONAL CHARACTER VARYING(MAX) NOT NULL,
CONSTRAINT Majority_Deputy_PK PRIMARY
KEY(majority_DeputyId)
)
GO

CREATE TABLE ORMModel1.Region
(
    regionID INTEGER IDENTITY (1, 1) NOT NULL,
    region_Name BINARY VARYING(MAX) NOT NULL,
    areaId INTEGER IDENTITY (1, 1),
    polling_DistrictDistrict_AddressName NATIONAL CHARACTER
                        VARYING(MAX),
    polling_DistrictPolling_DistrictNr INTEGER,
    areaArea_Name NATIONAL CHARACTER VARYING(MAX),
      CONSTRAINT Region_PK PRIMARY KEY(regionID),
      CONSTRAINT Region_Area_MandatoryGroup CHECK (areaId IS
            NOT NULL AND areaArea_Name IS NOT NULL OR
    polling_DistrictDistrict_AddressName IS NULL AND
```

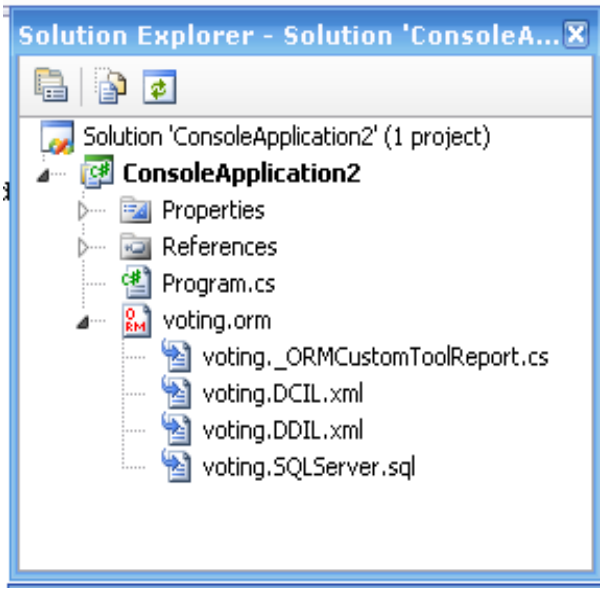
```
polling_DistrictPolling_DistrictNr IS NULL AND areaId IS NULL AND
areaArea_Name IS NULL),
    CONSTRAINT Region_Polling_District_MandatoryGroup CHECK
        (polling_DistrictDistrict_AddressName IS NOT NULL AND
polling_DistrictPolling_DistrictNr IS NOT NULL OR
polling_DistrictDistrict_AddressName IS NULL AND
polling_DistrictPolling_DistrictNr IS NULL)
)
GO

CREATE VIEW ORMModel1.Region_UC1 (areaId)
WITH SCHEMABINDING
AS
        SELECT  areaId
        FROM    ORMModel1.Region
        WHERE areaId IS NOT NULL
GO

CREATE UNIQUE CLUSTERED INDEX Region_UC1Index ON
            ORMModel1.Region_UC1(areaId)
GO

CREATE VIEW ORMModel1.Region_UC2
                    (polling_DistrictDistrict_AddressName)
WITH SCHEMABINDING
AS
        SELECT polling_DistrictDistrict_AddressName
        FROM   ORMModel1.Region
        WHERE polling_DistrictDistrict_AddressName IS NOT NULL
GO
```

```
CREATE UNIQUE CLUSTERED INDEX Region_UC2Index ON
ORMModel1.Region_UC2(polling_DistrictDistrict_AddressName)
GO

CREATE VIEW ORMModel1.Region_UC3
                    (polling_DistrictPolling_DistrictNr)
WITH SCHEMABINDING
AS
        SELECT polling_DistrictPolling_DistrictNr
        FROM   ORMModel1.Region
        WHERE polling_DistrictPolling_DistrictNr IS NOT NULL
GO

CREATE UNIQUE CLUSTERED INDEX Region_UC3Index ON
ORMModel1.Region_UC3(polling_DistrictPolling_DistrictNr)
GO

ALTER TABLE ORMModel1.Voter ADD CONSTRAINT Voter_FK1
        FOREIGN KEY (majority_DeputyId) REFERENCES
ORMModel1.Majority_Deputy (majority_DeputyId) ON DELETE NO
ACTION ON UPDATE NO ACTION
GO

ALTER TABLE ORMModel1.Voter ADD CONSTRAINT Voter_FK2
FOREIGN KEY (polling_DistrictNr) REFERENCES ORMModel1.Region
(polling_DistrictPolling_DistrictNr) ON DELETE NO ACTION ON
UPDATE NO ACTION
GO
GO
```

2.3. მონაცემთა ბაზის აგების პროგრამული რეალიზაციის ავტომატიზებული პროცედურა

Natural ORM Architect პროგრამული პაკეტის დახმარებით ავტომატიზებულ რეჟიმში ხდება DDL კოდის გენერირება, რათა აგვეგო დასაპროექტებელი სისტემის რელაციური სქემა. Solution Explorer-ში ნაჩვენებია კოდი, რომელიც დაგენერირდა SQL Server-ისათვის (ნახ.2.16).

ნახაზზე ილუსტრირებულია ConsoleApplication2 პროექტის სტრუქტურა, რომლის voting.orm ფაილი რამდენიმე კომპონენტისგან (.cs, .xml და .sql ფაილები) შედგება.



ნახ.2.16.VisualStudio.NET გარემოში პროექტი ConsoleApplication2

პროგრამის ტექსტი, რომელიც ავტომატიზებულ რეჟიმში იქნა მიღებული და აღწერს ჩვენი კვლევის ობიექტის ანუ ელექტრონული საარჩევნო სისტემის მონაცემთა მოდელის შინაარსს, შემდეგი სახისაა:

Voter **is an entity type**.
Reference Scheme: Voter has Voter_Nr.
Reference Mode: .Nr.

Fact Types:
Voter has Voter_Nr.
Voter has VoterName.
Voter has Surname.
Voter has Address.
Voter has FingerPrint.
Voter voted for a Party.
Voter voted for a Majority_Deputy.
Voter has Voter_ID.
Voter has Birth_Date.
Voter votes in a Polling_District.
VoterName **is a value type**.
Portable data type: Text: Variable Length.

Fact Types:
Voter has VoterName.
Voter has VoterName.
Each Voter has **exactly one** VoterName.
It is possible that more than one Voter has **the same** VoterName.
Voter has Surname.
Each Voter has **exactly one** Surname.
It is possible that more than one Voter has **the same** Surname.
Address **is an entity type**.
Reference Scheme: Address has Address_name.
Reference Mode: .name.

Fact Types:
Address has Address_name.
Voter has Address.
Voter has Address.
Each Voter has **exactly one** Address.
It is possible that more than one Voter has **the same** Address.
Surname **is a value type**.
Portable data type: Text: Variable Length.

Fact Types:
Voter has Surname.
FingerPrint **is an entity type**.
Reference Scheme: FingerPrint has FingerPrint_ID.
Reference Mode: .ID.

Fact Types:
FingerPrint has FingerPrint_ID.
Voter has FingerPrint.
Voter has FingerPrint.
Each Voter has **exactly one** FingerPrint.
For each FingerPrint, **at most one** Voter has **that** FingerPrint.
Voter voted for a Party.
Each Voter voted for a **exactly one** Party.
It is possible that more than one Voter voted for a **the same** Party.
Party **is an entity type**.
Reference Scheme: Party has Party_Id.
Reference Mode: .Id.

Fact Types:
Party has Party_Id.
Voter voted for a Party.
Party has Party_Name.

Party has Party_Name.
Each Party has **exactly one** Party_Name.
It is possible that more than one Party has **the same** Party_Name.
Majority_Deputy **is an entity type**.
Reference Scheme: Majority_Deputy has Majority_Deputy_id.
Reference Mode: .id.

Fact Types:
Majority_Deputy has Majority_Deputy_id.
Voter voted for a Majority_Deputy.
Majority_Deputy has Deputy_Name.
Voter voted for a Majority_Deputy.
Each Voter voted for a **exactly one** Majority_Deputy.
It is possible that more than one Voter voted for a **the same** Majority_Deputy.
Deputy_Name **is a value type**.
Portable data type: Text: Variable Length.

Fact Types:
Majority_Deputy has Deputy_Name.
Majority_Deputy has Deputy_Name.
Each Majority_Deputy has **exactly one** Deputy_Name.
It is possible that more than one Majority_Deputy has **the same** Deputy_Name.
Voter_ID **is an entity type**.
Reference Scheme: Voter_ID has Voter_ID_code.
Reference Mode: .code.

Fact Types:
Voter_ID has Voter_ID_code.
Voter has Voter_ID.
Voter has Voter_ID.
Each Voter has **exactly one** Voter_ID.
For each Voter_ID, **at most one** Voter has **that** Voter_ID.

Birth_Date **is an entity type**.
Reference Scheme: Birth_Date has Birth_Date_mdy.
Reference Mode: .mdy.

Fact Types:
Birth_Date has Birth_Date_mdy.
Voter has Birth_Date.
Voter has Birth_Date.
Each Voter has **exactly one** Birth_Date.
It is possible that more than one Voter has **the same** Birth_Date.
Region **is an entity type**.
Reference Scheme: Region has Region_ID.
Reference Mode: .ID.

Fact Types:
Region has Region_ID.
Region has Region_Name.
Each Area is an instance of Region.
Region_Name **is a value type**.
Portable data type: Raw Data: Variable Length.

Fact Types:
Region has Region_Name.
Region has Region_Name.
Each Region has **exactly one** Region_Name.
It is possible that more than one Region has **the same** Region_Name.
Area **is an entity type**.
Reference Scheme: Area has Area_Id.
Reference Mode: .Id.

Fact Types:
Area has Area_Id.
Area has Area_Name.

Each Polling_District is an instance of Area.
Each Area is an instance of Region.
Area_Name **is a value type**.
Portable data type: Text: Variable Length.

Fact Types:
Area has Area_Name.
Area has Area_Name.
Each Area has **exactly one** Area_Name.
It is possible that more than one Area has **the same** Area_Name.
Polling_District **is an entity type**.
Reference Scheme: Polling_District has Polling_District_Nr.
Reference Mode: .Nr.

Fact Types:
Polling_District has District_Address.
Each Polling_District is an instance of Area.
Polling_District has Polling_District_Nr.
Voter votes in a Polling_District.
District_Address **is an entity type**.
Reference Scheme: District_Address has District_Address_name.
Reference Mode: .name.

Fact Types:
District_Address has District_Address_name.
Polling_District has District_Address.
Polling_District has District_Address.
Each Polling_District has **exactly one** District_Address.
For each District_Address, **at most one** Polling_District has **that** District_Address.
Voter votes in a Polling_District.
Each Voter votes in a **exactly one** Polling_District.
It is possible that more than one Voter votes in a **the same** Polling_District.

Context: Area has Area_Name; Majority_Deputy has Deputy_Name.

In this context, each Area_Name, Deputy_Name combination is unique.
Model Error: Constraint 'ExternalUniquenessConstraint1' in model 'ORMModel1' requires a join path.
Each Area is an instance of Region.
Each Polling_District is an instance of Area.

2.4. სისტემის ტექნიკური უზრუნველყოფის მახასიათებლების განსაზღვრა და შეფასება

1990 წლიდან საქართველოში უამრავი არჩევნები ჩატარდა, როგორც ადგილობრივი თვითმართველობის, ასევე საპარლამენტო და საპრეზიდენტო. ჩატარდა რამდენიმე რეფერენდუმიც, თუმცა მოსახლეობა, მიუხედავად ამდენი ჩატარებული არჩევნებისა, დიდ უნდობლობას გამოხატავს ყველა არჩევნების მიმართ. ვერ მოხერხდა ქართული საარჩევნო კანონმდებლობის და საარჩევნო სისტემის დახვეწა. აღარაფერს ვამბობთ საარჩევნო სიაზე და მის ფორმირებაზე, სადაც დღეის მდგომარეობითაც უამრავი ხარვეზია. გარდა ამისა, არსებული სისტემით არჩევნების ჩატარება სახელმწიფოსათვის დიდ ხარჯებთან არის დაკავშირებული.

მაგალითად, განვიხილოთ მცირეოდენი ინფორმაცია საქართველოში 2012 წელს ჩატარებული არჩევნების შესახებ. შეიძლება აღვნიშნოთ, რომ სისტემა აშკარად იხვეწება და მისი სრულყოფისთვის ბევრი კარგი და დიდი საქმეა გაკეთებული თუ წამოწყებული. მოგეხსენებათ, რომ 2012 წლის არჩევნებისათვის საქართველოს მასშტაბით შეიქმნა 3,766 საარჩევნო უბანი, მათ შორის გამონაკლის შემთხვევაში – 71, საზღვარგარეთ – 45 და 2 უბანი – ავღანეთში ჰელმანდისა და შუკვანის პროვინციებში. აღნიშნული უბნების ზემდგომი ინსტანციაა საოლქო საარჩევნო

უბნები, რომელიც სულ 84-ია საქართველოს მასშტაბით, აქედან 10 თბილისშია. სულ ამომრჩეველთა სიებში რეგისტრირებული იყო 3 613 851 ამომრჩეველი, აქედან 1 025 455 – თბილისში.

რადგან საარჩევნო უბნების რაოდენობა არ არის უცვლელი და იგი რეგულირდება საარჩევნო კოდექსის მე-16 მუხლის შესაბამისად, ვიმედოვნებთ, რომ მისი რიცხვი გაიზრდება, რაც პირდაპირპროპორციულად ნიშნავს საქართველოს მოქალაქეთა რაოდენობის მატებას.

მუხლი 16. საარჩევნო უბნები

1. კენჭისყრის ჩასატარებლად და ხმების დასათვლელად საარჩევნო ოლქი იყოფა საარჩევნო უბნებად.

2. საარჩევნო უბანი იქმნება არანაკლებ 20 და არა უმეტეს 1500 ამომრჩევლისთვის. საარჩევნო უბნებს ქმნის, მათ საზღვრებსა და ნომრებს ადგენს შესაბამისი საოლქო საარჩევნო კომისია არაუგვიანეს არჩევნების წლის 1 ივლისისა და 2 დღის ვადაში აქვეყნებს სათანადო ინფორმაციას უბნების საზღვრების მითითებით. საოლქო საარჩევნო კომისია ადგილობრივი თვითმმართველობის ორგანოთა მონაცემების საფუძველზე ადგენს და აზუსტებს საარჩევნო უბანში შემავალი ყველა საცხოვრებელი შენობა-ნაგებობის ნუსხასა და მისამართს, აგრეთვე იმ შენობა-ნაგებობათა ნუსხასა და მისამართებს, რომლებიც საარჩევნო ადმინისტრაციამ შეიძლება გამოიყენოს საარჩევნო მიზნებისთვის. პარლამენტის რიგგარეშე არჩევნების შემთხვევაში საარჩევნო უბნები იქმნება არჩევნებამდე არა უგვიანეს 40 დღისა. (22.11.2007 N 5500)

2012 წლის არჩევნებისათვის ცენტრალური საარჩევნო კომისიის ინფორმაციაზე დაყრდნობით დაიწყო ფართომასშტაბიანი პროექტის „საოლქო საარჩევნო კომისიებში საარჩევნო

პროცესის მართვის ელექტრონული სისტემის” ეტაპობრივი განხორციელება. ამ სამუშაოს მიზანი იყო საარჩევნო ადმინისტრაციაში არსებული განვითარებული საინფორმაციო ტექნოლოგიების ინფრასტრუქტურაზე დაყრდნობით პროგრამული სისტემის შექმნა, რომელიც ხელს შეუწყობდა, ერთი მხრივ, საარჩევნო ოლქში განსახორციელებელი სამუშაოების უკეთ შესრულებას, ხოლო, მეორე მხრივ, საშუალებას მისცემდა ცენტრალურ საარჩევნო კომისიას ერთიანად ედევნებინა თვალყური საოლქო საარჩევნო კომისიებში მიმდინარე პროცესებისადმი და, საჭიროების შემთხვევაში, მოვლენებზე დროული რეაგირება მოეხდინა რეალური დროის რეჟიმში.

პირველ ეტაპზე შეიქმნა შემდეგი მოდულები: „საარჩევნო კალენდარი“, „საჩივრების რეესტრი“, „ადამიანური რესურსების მართვა“, „არჩევნების შედეგები“ და „საინფორმაციო გვერდი“.

პარალელურად ხდებოდა შესაბამისი მონაცემების ბაზების სტრუქტურის განსაზღვრა და მათში რეალური მონაცემების შეტანა. ასევე სისტემატურად მიმდინარეობდა მონიტორინგი საოლქო საარჩევნო კომისიებში ახალი VPN-ქსელის მონტაჟის შესახებ.

კიდევ ერთი სასიამოვნო სიახლე იქნა ტესტირების რეჟიმში გაშვებული, უბნებსა და ოლქებში ამომრჩეველთა იდენტიფიცირებისათვის შეიქმნა ამომრჩეველთა სიის კომპიუტერული საძიებო პროგრამა ოპერაციული სისტემების Windows-სა და Android-ისათვის. აღნიშნული პროგრამა ჩაიწერა პორტატული და პლანშეტური ტიპის კომპიუტერებზე (2,000-მდე ერთეული) და საჭიროებისამებრ გადანაწილდა საარჩევნო ოლქებში უბნებზე გადასაცემად.

ყოველივე ზემოხსენებულისათვის სახელმწიფოს მხრიდან საჭირო ინვენტარისა და სხვა საგნების შესყიდვის მიზნით წინასაარჩევნო პერიოდში გამოცხადდა 20 ტენდერი, მათ შორის 8

ელექტრონული და 12 გამარტივებული ელექტრონული ტენდერი. გაფორმებულ იქნა 20 ხელშეკრულება, რომელთა ჯამური თანხა შეადგენდა 1 377 517 ლარს.

2012 წლის საპარლამენტო არჩევნების სრულად ჩასატარებლად საარჩევნო ღონისძიებებისათვის სახელმწიფო შესყიდვების პროცესში გაფორმდა 312 ხელშეკრულება და შესყიდულ იქნა 2 806 829 ლარის საქონელი და მომსახურება.

აქვე შეიძლება აღვნიშნოთ, რომ თანხის იგივე რაოდენობა იქნება საჭირო სავარაუდოდ მომდევნო არჩევნებისთვისაც, რომელიც სახელმწიფომ უნდა გაიღოს იმ დემოკრატიული პროცესისათვის, რასაც არჩევნების ჩატარება ჰქვია.

იმისათვის, რომ მოხდეს ელექტრონული საარჩევნო სისტემის დანერგვა და არჩევნების ჩატარება, ვიდრე ტექნიკური საშუალებების შემდეგ ჩამონათვალს გაგაცნობდეთ, აუცილებლად მიგვაჩნია მცირეოდენი ცვლილებები საარჩევნო საოლქო მოწყობის შესახებ.

კერძოდ, სასურველია მოხდეს ოლქებსა და ცენტრალურ საარჩევნო კომისიას შორის მხოლოდ ინფორმაციულ დონეზე გამსხვილება და ნაცვლად 84 ოლქისა, სერვერების განთავსება მოხდეს რეგიონული მოწყობის მიხედვით, თუმცა 84 ოლქი დარჩეს ხელუხლებელი. ეს საშუალებას მოგვცემს მაქსიმალურად ეფექტურად მოვახდინოთ სისტემის მართვა და რაც ყველაზე მნიშვნელოვანია, მინიმალური დანახარჯებით მივაღწიოთ საუკეთესო შედეგს.

გთავაზობთ საქართველოს მასშტაბით რეგიონების ჩამონათვალს, სადაც უნდა განთავსდეს სერვერები:

1. თბილისი,
2. კახეთი,
3. მცხეთა-თიანეთი,
4. შიდა ქართლი,

5. ქვემო ქართლი,
6. სამცხე-ჯავახეთი,
7. იმერეთი,
8. რაჭა-ლეჩხუმი, ქვემო სვანეთი,
9. სამეგრელო-ზემო სვანეთი,
10. აფხაზეთი,
11. გურია,
12. აჭარა.

სერვერების განსათავსებლად და ასაწყობად შესაძლებელია გამოყენებულ იქნას საქართველოს იუსტიციის სახლები.

რეგიონებს შორის, მისი სიდიდიდან და ამომრჩეველთა რაოდენობიდან გამომდინარე, ცალკე გამოვყავით თბილისი, სადაც თითოეულ ოლქში, ასეთი 10-ია, განვათავსებთ თითო სერვერს.

ახლა კი გთავაზობთ იმ ტექნიკის ჩამონათვალს, რაც არჩევნების ელექტრონული წესით ჩატარებისათვის უბნების აღსაჭურვად არის საჭირო, გარდა ქსელის მოწყობისა. ესენია:

- სვიჩი-DGS-1024D 24-PORT 1000,
- პორტატული კომპიუტერი (ხარჯების შესამცირებლად შესაძლებელია ეს იყოს Netbook-ებიც) შემდეგი მონაცემებით- 15.6" Intel i3/ 2.0GHz 2GB,
- სპეციალური ვიდეო თვალი + მიკროფონი-USB 2.0 Color Box Camera, IP კამერა-D-LINK DCS-33-4 Mounting Bracket,
- თითის ანაბეჭდის სკანერი,
- ციფრული ხელმოწერის წამკითხველი მოწყობილობა.

აქვე აღვნიშნავთ, რომ ელექტრონული საარჩევნო სისტემის დანერგვის მეორე ეტაპი მოიცავს ქაღალდის ბიულეტენების საერთოდ ამოღებას არჩევნების პროცესიდან და მას ჩაანაცვლებს სპეციალური Touch screen-ით აღჭურვილი კომპიუტერული ბლოკი-Direct Recording Electronic(DRE), რომელიც განთავსდება

უბნებზე სპეციალურად დამონტაჟებულ კაბინებში, სადაც ამომრჩეველი ეკრანზე სპეციალური კალმით შემოხაზავს სასურველ კანდიდატს. მოგახსენებთ სრულ რაოდენობრივ ჩამონათვალს:

1. სერვერი – 11 ერთეული +10 ერთეული თბილისისათვის,
2. Cisco როუტერი – 21 ერთეული,
3. პორტატული კომპიუტერი –3766*3 = 11298 ერთეული,
4. ვიდეო თვალი – 11298 ერთეული,
5. ელექტრონული ხელმოწერისა და თითის ანაბეჭდის სკანერი – 11298 ერთეული,
6. IP კამერა – 3766 ერთეული,
7. Cisco Wireless როუტერი(Small Business) – 3766 ერთეული,
8. ქსელის მოწყობა, დემონტაჟი, დამატებითი ხარჯები.

შეიძლება გამოითქვას აზრი იმის თაობაზე, რომ ელექტრონული წესით არჩევნების ჩატარება სახელმწიფოს ამ მონაცემების ფონზე ძვირი დაუჯდება.

თუმცა არგუმენტად, გარდა იმისა, რომ სამართლიანი და გამჭვირვალე არჩევნები არის დემოკრატიის შუქურა, საქართველოს სახელმწიფოსათვის ერთჯერადად გაღებული ეს თანხა არ იქნება პრობლემა და კატასტროფა, იმ ფონზე როცა ყოველწლიურად განათლების სამინისტრო ყველა საჯარო სკოლის პირველკლასელ მოსწავლეს „ნეტბუკებს” ურიგებს.

ამონარიდი ექსპრესნიუსის ოფიციალური ვებ გვერდიდან: როგორც „ექსპრესნიუსს” განათლების სამინისტროდან აცნობეს, 2012–2013 სასწავლო წლისთვის საქართველოს სკოლებისთვის შექმნილი ნეტბუკები 41272 პირველკლასელს და მათ 3078 დამრიგებელს საჩუქრად გადაეცემა.

2.17 ნახაზი გვიჩვენებს თუ რა ტექნიკური საშუალებებია საჭირო ელექტრონული წესით არჩევნების ჩასატარებლად.

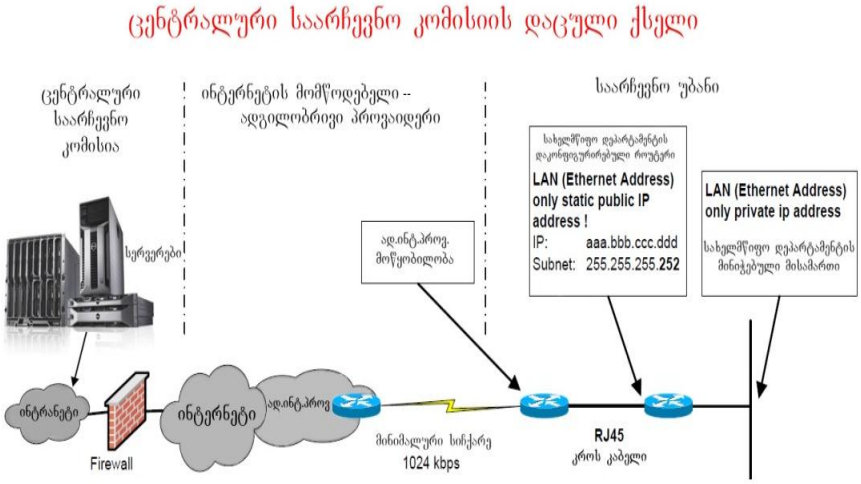
ნახ.2.17

აღნიშნული ტექნიკის კალკულაციას, სავარაუდოდ, ექნება 2.1 ცხრილში მოცემული სახე (დღევანდელი ფასებით).

#	დასახელება	რაოდე-ნობა	ერთეულის ფასი	საერთო ფასი
1	სერვერი HP Proliant DL 360 G7	21	5500	115500
2	Cisco როუტერი	21	1200	25200
3	პორტატული კომპიუტერი	11298	450	5084100
4	ვიდეო თვალი	11298	15	169470
5	ელექტრონული ხელ-მოწერისა და თითის ანაბეჭდის სკანერი	11298	190	2146620
6	IP კამერა	3766	130	489580
7	CiscoWireless როუტერი (Small Business)	3766	220	828520
	ჯამი			**8.858.990 ლარი**
8	ქსელის მოწყობა, დემონტაჟი, დამატებითი ხარჯები			მთლიანი ღირებულების 5% = 442950
	საბოლოო ჯამი			**9.301.940 ლარი**

2.5. VPN ქსელის დაპროექტება, აგება და უსაფრთხოების ნორმების უზრუნველყოფა

ცენტრალური საარჩევნო კომისიისათვის აიგება ელექტრონული საარჩევნო სისტემის მხარდამჭერი კომპიუტერული ქსელი, რომლის გამართვასა და ფუნქციონირებაზე, ასევე უსაფრთხოებაზე პასუხისმგებელია ინფორმატიზაციის სახელმწიფო დეპარტამენტი. ქსელის მოწყობისათვის საჭიროა ჩატარდეს ტენდერი ადგილობრივ ინტერნეტ პროვაიდერებს შორის, რომელიც უზრუნველყოფს საარჩევნო უბნებს ინტერნეტით (მინიმალური სიჩქარე 10 მგბ.წ) (ნახ.2.18).



ნახ.2.18.

სასურველია მოხერხდეს ყველა საარჩევნო უბნის ოპტიკურ-ბოჭკოვანი კაბელით ჩართვა ინტერნეტში. აქვე მინდა აღვნიშნო, რომ სავსებით შესაძლებელია ამგვარი ქსელის აგების ვალდებულება იკისროს კომპანია მაგთიკომმა (მაგთიკომმა შექმნა ერთიანი სამთავრობო ქსელი). ეს რაც შეეხებოდა ფიზიკური ქსელის აგებას და მასში ყველა სერვერის და მომხმარებლის კომპიუტერის ჩართვას, რომელიც ელექტრონული წესით არჩევნების ჩასატარებლად არის საჭირო. ხოლო რაც შეეხება ამ ქსელის გამართულობას და უსაფრთხოებას, განვიხილოთ კონკრეტული ტექნოლოგია, რომელიც დააკმაყოფილებს ზემოთ ხსენებულ ორივე მოთხოვნას. სწორედ ასეთი ტექნოლოგიაა- VPN (ვირტუალური კერძო ქსელი). იგი უზრუნველყოფს დამოუკიდებელი დაცული ქსელის შექმნას ინტერნეტის ან სხვა ღია არხების მეშვეობით. აღნიშნული ტექნოლოგია საშუალებას გვაძლევს შევქმნათ ცენტრალური საარჩევნო სისტემის საკუთარი კორპორატიული ქსელი ნებისმიერ საარჩევნო უბანზე. VPN კლიენტის პროგრამული უზრუნველყოფა იტვირთება დაშორებულ კომპიუტერზე და ინტერნეტის მეშვეობით უკავშირდება კორპორატიულ ქსელში განთავსებულ VPN სერვერს ან როუტერს. VPN ტექნოლოგია ოპტიმალურია ელექტრონული საარჩევნო სისტემისათვის შექმნილი ქსელის სამართავად. ცენტრალური საარჩევნო კომისიიდან გაუსვლელად წვდომა წვდომა გვექნება შიდა ქსელის ყველა რესურსთან, როგორიც არის მულტიმედიალური მონაცემთა ბაზები, და ა.შ. VPN-ის შესაქმნელად საჭიროა არსებობდეს ფიზიკური არხი, როგორც საკაბელო, ასევე რადიოსიხშირული სპექტრის გამოყენებით. ჩვენს შემთხვევაში გამოყენებული იქნება დიდი ქალაქების დონეზე, როგორებიცაა თბილისი, ქუთაისი, ბათუმი, რუსთავი, გორი, თელავი, ფოთი, ოზურგეთი, ზუგდიდი მხოლოდ და მხოლოდ საკაბელო არხები, ხოლო დაბებსა და სოფლებში

რადიოსიხშირული ინტერნეტი, რაც ზემოთ იყო აღნიშნული. ქსელი ორი ნაწილისაგან შედგება, ესენია: „შიგა“ და „გარე“ ქსელი. პირველ შემთხვევაში, იქმნება ერთი ლოკალური ქსელი, რომლის მოქმედების რადიუსი, მაქსიმუმ 100 მეტრია და რომელიც უშუალოდ საარჩევნო უბანზე აიგება, ხოლო მეორე – „გარე“ ქსელი, რომელიც ამ ქსელს დააკავშირებს ცენტრალური საარჩევნო კომისიის სერვერებთან. VPN-მა, თავის მხრივ, შეიძლება გამოიყენოს DSL ტექნოლოგიაც, რასაკვირველია აქაც საჭიროა ინტერნეტის მაღალი სიჩქარე, რათა მოხდეს მონაცემთა შეუფერხებელი გადაცემა. მონაცემები და მარშრუტიზაციის შემცველი თავსართები საჯარო ქსელის მეშვეობით დაშიფრული სახით გაიგზავნება, რასაც ტუნელირება ეწოდება. აღნიშნული ქსელის ასაგებად უნდა გამოვიყენოთ IPsec ტექნოლოგია, რომელი ყველაზე სანდო და უსაფრთხოა თვით VPN-ის ტიპებს შორის.

სხვა VPN პროტოკოლებთან შედარებით, IPsec არის ყველაზე პოპულარული და ყველაზე გამოყენებადი, რადგანაც მისი როგორც გამართვა, ასევე დიაგნოსტიკა არის ძალიან მარტივი.

იმისათვის, რომ VPN (ვირტუალური კერძო ქსელის) ტექნოლოგია გამოვიყენოთ ელექტრონული საარჩევნო სისტემის მხარდამჭერი ქსელის აგებისას, საჭიროა თითოეული საარჩევნო უბანი აღიჭურვოს კომპანია Cisco-ს მიერ წარმოებული როუტერებით, მაგალითად, Cisco 2800 Series Integrated Services Routers. თითოეული Cisco 2800 უნდა დაკონფიგურირდეს შემდეგნაირად:

```
VPN-CECTBI-1# show running-config
Building configuration....
Current configuration : 4068 bytes
!
version 12.3
no service pad
```

```
service timestamps debug datetime msec
service timestamps log datetime msec
service password-encryption
!
hostname VPN-CECTBI-1
!
boot-start-marker
boot-end-marker
!
enable secret 5 $1$9BB/************************************/
!
username admin password 7 **********************
memory-size iomem 25
aaa new-model
!
!
aaa authentication login USERLIST local
aaa session-id common
ip subnet-zero
!
!
ip cef
ip dhcp excluded-address 192.168.3.1
!
ip dhcp pool PRIVATE_DHCP
   import all
   network 192.168.3.0 255.255.255.0
   default-router 192.168.3.1
!
!
no ip domain lookup
ip multicast-routing
ip ids po max-events 100
!
```

```
no ftp-server write-enable
voice-card 0
 no dspfarm
!
!--- IPSec configuration
!
crypto ipsec client ezvpn VPN1
 connect auto
 group VPN1 key cisco123
 mode network-extension
 peer 10.32.152.26
 username VPN-CECTBI-1 password  **********
!
interface FastEthernet0/0
 description === private interface ===
 ip address 192.168.3.1 255.255.255.0
 duplex auto
 speed auto
 crypto ipsec client ezvpn VPN1 inside
!
interface FastEthernet0/1
 no ip address
 duplex auto
 speed auto
 shutdown
!
interface Serial0/0/0
 description === public interface ===
 ip address 10.32.150.46 255.255.255.252
 crypto ipsec client ezvpn VPN1
!
ip classless
ip route 0.0.0.0 0.0.0.0 10.32.150.45
!
ip http server
```

```
no ip http secure-server
!
control-plane
!
dial-peer cor custom
!
line con 0
 exec-timeout 0 0
line aux 0
line vty 0 4
 login authentication USERLIST
!
End
```

სადაც VPN-CECTBI-1# აღნიშნავს ცენტრალური საარჩევნო კომისიის, თბილისის ნომერ პირველ საარჩევნო უბანს.

ამრიგად, ჩვენ უზრუნველყოფთ უმაღლესი დონის უსაფრთხო ქსელის მოწყობას თითოეულ საარჩევნო უბანზე Cisco როუტერების ანალოგიური კონფიგურირებით.

2.6. ელექტრონული საარჩევნო სისტემის პროცესების იმიტაციური მოდელირება ფერადი პეტრის ქსელით

მონაცემთა ბაზებს შორის ინფორმაციის გაცვლა ხორციელდება ასინქრონულად სერვისორიენტირებული არქიტექტურით.

მოთხოვნათა ტიპები განისაზღვრება ამოცანათა სახეებით, მაგალითად:

- ამომრჩეველთა რეგისტრაციის პროცესი,
- მათი ცენტრალური სერვერის ბაზაში განთავსება,
- მონიტორინგის და კონტროლის ამოცანები დუბლირების აღმოსაფხვრელად,
- ამომრჩეველთა ხმის მიცემის პროცედურა და ა.შ.

ამ ბიზნესპროცესების საფუძველზე ხორციელდება შეტყობინებათა და მონაცემთა პაკეტების გაცვლის სერვისების ხშირი გამოყენება (წინასაარჩევნო და საარჩევნო პერიოდში ცენტრალური საარჩევნო ოფისის სერვერ მანქანებზე შეიძლება მიიღონ (ან გადასცენ) რამდენიმე მილიონზე მეტი მოთხოვნა ამომრჩეველთა შესახებ).

ასეთი ინფორმაციის მენეჯმენტი მოითხოვს საიმედო აღრიცხვის და რისკების გამორიცხვის პროცედურების გათვალისწინებას. შეტყობინებათა ერთობლიობა, რომელიც მუდმივად გადაიცემა ქსელის საშუალებით, არ უნდა დაიკარგოს და ყოველი მათგანი უნდა ექვემდებარებოდეს მკაცრ კონტროლს, უნდა შეიძლებოდეს აღდგენის და არქივირების ქმედებათა შესრულება.

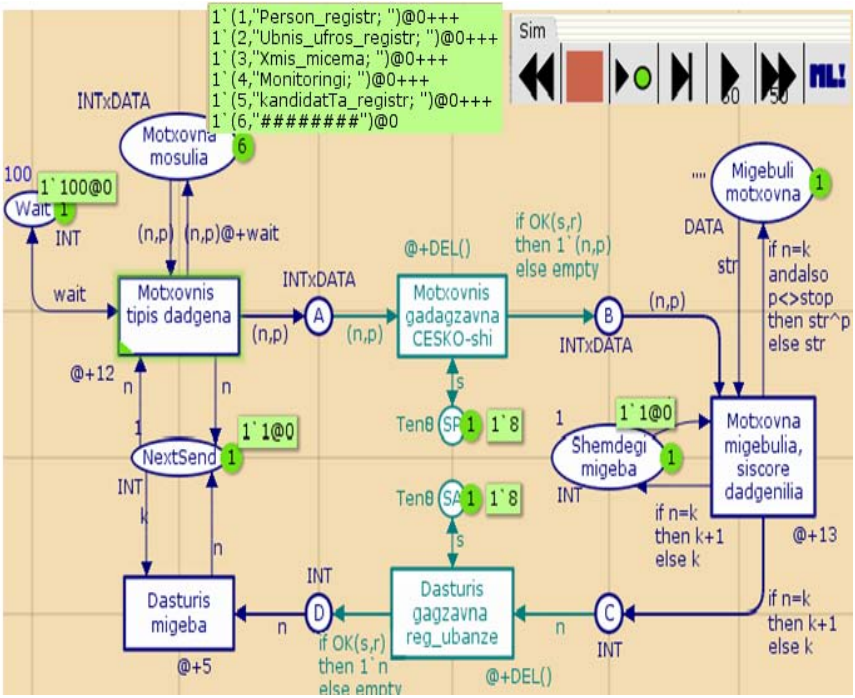
სწორედ ამიტომ ასეთი სერვისული მოთხოვნების დამუშავების მართვის პროცესის იმიტაციური მოდელირება განხორციელებულია პეტრის ფერადი ქსელების CPN-ინსტრუმენტით (Coloured Petri Net).

თავდაპირველად ხდება მოთხოვნის ტიპის დადგენა, თუ რა სახის ინფორმაციის გაგზავნა მოითხოვეს და მხოლოდ ამის შემდეგ ხდება დასტურის (შეტყობინების) დაბრუნება შესრულებულ მოთხოვნაზე.

2.19 ნახაზზე მოცემულია პეტრის ფერადი ქსელის ფრაგმენტი ჩვენი სისტემისათვის. აქ გადასასვლელ ბლოკებში ნაჩვენებია, მაგალითად, მოთხოვნის ტიპის დადგენა, შეტყობინების გადაგზავნა ცენტრალურ საარჩევნო ოფისში, დასტურის მიღება საარჩევნო უბანზე.

CPN-ინსტრუმენტი იყენებს ობიექტორიენტირებულ, 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.



ნახ.2.19. CPN ქსელის ფრაგმენტი

ქსელის ყოველი პოზიციის გვერდით შეიძლება აისახოს მოცემულ მომენტში შემავალი ფერადი მარკერები. საინიციალიზაციო მარკირება ხაზგასმული ტექსტის სახით გამოიტანება. მაგალითად, საწყის მდგომარეობაში პოზიცია „*მოთხოვნის ტიპის დადგენა*“ შეიცავს INTxDATA ტიპის ფერად მარკერთა 5-ელემენტიან სიმრავლეს (საინიციალიზაციო მარკირება):

{1’(1, „მოქალაქის_რეგისტრაცია“),1’ (2, „უბნის_უფროსის _რეგისტრაცია“), 1’(3, „ხმის მიცემა“), 1’(4, „მონიტორინგი“), 1’(5, „კანდიდატთა_რეგისტრაცია”) და ა.შ. }. ბოლო, მე-6 ელემენტი: *1’(6, „#########”)* შეესაბამება დასასრულის იდენტიფიკაციას - **stop** (ნახ. 2.20).

```
Declarations
colset INT = int timed;
colset DATA = string;
colset INTxDATA = product INT * DATA timed;
var n, k, wait: INT;
var p, str: DATA;
val stop = "#########";
colset Ten0 = int with 0..10;
colset Ten1 = int with 1..10;
var s: Ten0;
var r: Ten1;
fun OK(s:Ten0,r:Ten1) = (r<=s);
colset NetDelay = int with 25..75;
fun DEL() = NetDelay.ran();
```

ნახ.2.20.

1-ლი რიცხვი სტრიქონში *1’(3, „ხმის მიცემა“)* – კოეფიციენტია, რაც ნიშნავს, რომ პოზიციაში არის არაუმეტეს 1 ცალი მოცემული ფერის მონაცემი (ანუ არსებობს მხოლოდ ერთი მოთხოვნა ნომრით „*ხმის მიცემა*“, რომლის ფერია – რიგითი ნომერი 3). ამ შემთხვევაში გვაქვს მონაცემთა ელემენტების სიმრავლე.

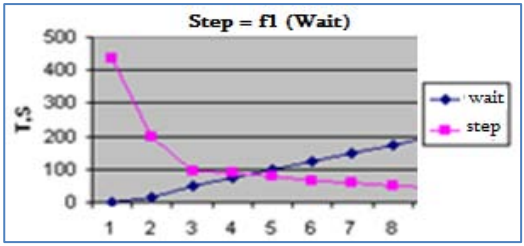
მეორე მაგალითი, პოზიცია ”*მოთხოვნ_მოსულია*” შედგება 6 ელემენტისგან (1+1+1+1+1+1), რომლებიც 5 სხვადასხვა (მარკერების ფერის) მოთხოვნათა ტიპების რაოდენობას, ანუ მულტისიმრავლეს ასახავს. პროცესების შესრულების დრო (დაყოვნება) აისახება გადასავლელთან სიმბოლოს და დროის ერთეულის (მაგალითად, @+12, @+wait, @+DEL) მითითებით, სადაც wait წინასწარ განსაზღვრული კონსტანტაა (mag., 100), DEL – კი

შემთხვევით რიცხვთა გენერატორით ფორმირებული (random-ფუნქცია), რიცხვია [25..75] ინტერვალში. იმიტაციის პროცესში ეს რიცხვი ავტომატურად აირჩევა (სტოქასტური მოვლენის მოდელირება).

ჩვენი დროითი CPN -მოდელით შეიძლება გამოვიკვლიოთ „ცესკოს“ სათავო ოფისსა და ფილიალებს შორის შეტყობინებათა გაცვლის პროცესის მახასიათებლები, მათი განმეორებითი გადაცემის დაყოვნების დროის (wait) სხვადასხვა მნიშვნელობისთვის.

ხანმოკლე დაყოვნება ზრდის შანსს განმეორებითი გადაგზავნების თავიდან ასაცილებლად. იგი ასევე ზრდის შანსს, რომ ოპერაცია Dasturis_migeba გადაიდოს, რადგან პროცესი Motxovnis tipis dadgena დაკავებულია განმეორებითი გადაგზავნით.

გრძელი დაყოვნება ნიშნავს, რომ საჭირო იქნება დიდხანს ცდა, ფილიალი დარწმუნდება, რომ შეტყობინება ან დასტური იქნა დაკარგული. იმიტაციის პროცესში, სხვადასხვა wait-მნიშვნელობით შეიძლება დადგინდეს ოპტიმალური მნიშვნელობა განმეორებითი გადაცემის დაყოვნებისათვის (ნახ. 2.21).



ნახ.2.21.

2.7. სისტემის ფუნქციონირების ორგანიზების ასპექტები

ტრადიციული და ელექტრონული საარჩევნო სისტემების დეტალური ანალიზის საფუძველზე შემოთავაზებულია უშუალოდ „არჩევნების დღის“ რეგლამენტი, ფუნქციონირების თვალსაზრისით. უკვე არსებული, ტრადიციული საარჩევნო სისტემით ჩატარებული არჩევნების პროცედურული სქემა და ახალი, ჩვენ მიერ შემოთავაზებული ელექტრონული საარჩევნო სისტემის მიხედვით ჩატარებული არჩევნების მოდელი. განვიხილოთ ისინი დეტალურად და მოვახდინოთ მათი შედარება.

ტრადიციული სისტემით გათვალისწინებულია შემდეგი პროცედურა:

1. ამომრჩეველი, რომელიც მიდის საარჩევნო უბანზე, ეძებს საკუთარ თავს ამომრჩეველთა ერთიან სიაში, რომელიც გამოკრულია, როგორც წესი, საარჩევნო უბნებზე;

2. პოვნის შემდეგ შედის უშუალოდ საარჩევნო დარბაზში და გადის მარკირების დეტექციას;

3. გადის რეგისტრაციას საარჩევნო უბანზე, რაც გულისხმობს ამომრჩევლის მიერ ხელმოწერით დადასტურებას;

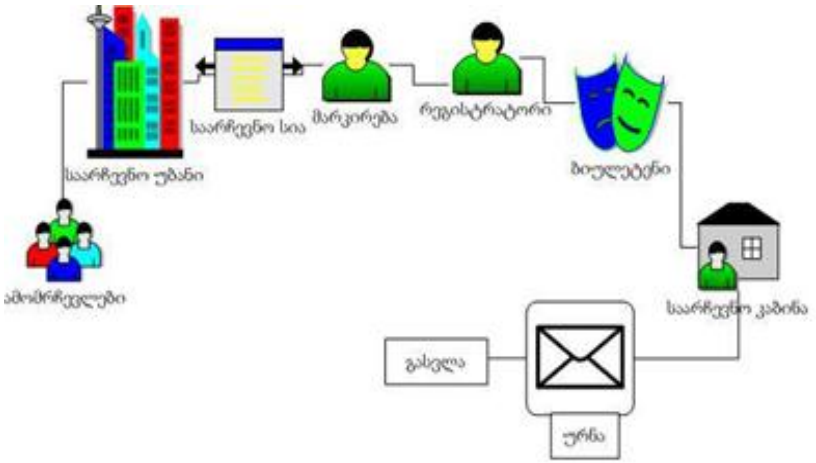
4. იღებს საარჩევნო ბიულეტენს;

5. შედის საარჩევნო კაბინაში და ხაზავს სასურველ პირს;

6. ათავსებს ბიულეტენს კონვერტში და აგდებს საარჩევნო ურნაში;

7. ტოვებს საარჩევნო უბანს.

2.22 ნახაზზე ეს პროცესი ასახულია სქემატურად.



ნახ. 2.22.

ელექტრონული სისტემით არჩევნების ჩატარებისას, სასურველად მიგვაჩნია გაიზარდოს რეგისტრატორთა რიცხვი და ნაცვლად 3-ისა გახდეს 5, რათა თავიდან იქნას აცილებული (ხელოვნურად თუ არახელოვნურად) რიგების წარმოქმნა საარჩევნო უბნებზე.

თუმცა აქვე დავსძენთ, რომ ფერადი პეტრის ქსელების საფუძველზე ჩატარებულმა ექსპერიმენტმა (გათამაშებულმა მოდელმა) კარგი შედეგი გვაჩვენა და სავსებით საკმარისია სამი რეგისტრატორიც.

ჩვენი ელექტრონული საარჩევნო სისტემა მოიცავს შემდეგ პროცედურებს:

1. რეგისტრაციის გავლა ნებისმიერ რეგისტრატორთან შემდეგი წესის დაცვით, ტექსტური მონაცემების მონაცემთა ბაზაში შეყვანის შემდეგ უნდა მოხდეს თითის სკანირება და ანაბეჭდის ბაზაში გადატანა; ელექტრონული წესით ხელმოწერის განორციელება და მისი მულტიმედიურ მონაცემთა ბაზაში

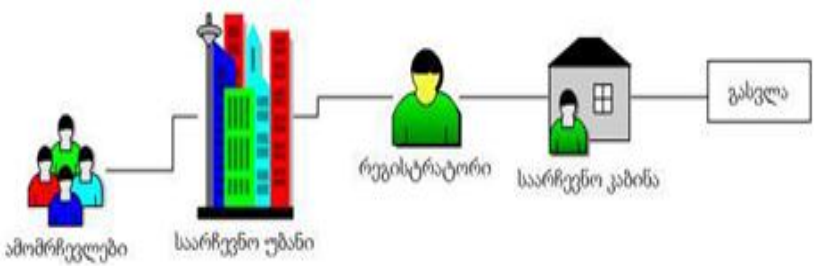
გადატანა; ბიომეტრული ფოტოსურათის გადაღება, მონაცემთა ბაზის შესაბამის ველში მისი დამახსოვრება და აუდიო ფაილის, რომელიც ამომრჩევლის ხმას ჩაიწერს, მისი მულტიმედიალურ ბაზაში, შესაბამის ველში ჩაგდება;

2. იდენტიფიცირების წარმატებით გავლის შემთხვევაში მოქალაქე მიიღებს ბიულეტინს;

3. შედის საარჩევნო კაბინაში და ხაზავს სასურველ პირს;

4. ათავსებს ბიულეტენს კონვერტში და აგდებს საარჩევნო ურნაში;

5. ტოვებს საარჩევნო უბანს (ნახ.2.23).



ნახ. 2.23.

იმ შემთხვევაში, თუ იდენტიფიცირების შემდეგ აღმოჩნდება, რომ მოქალაქეს უკვე გავლილი აქვს სხვა რომელიმე უბანზე რეგისტრაცია, იგი მხილებული იქნება არჩევნების გაყალბების მცდელობაში და საარჩევნო კომისიის შეცდომაში შეყვანის მცდელობის გამო მოხდება მისი დაკავება სამართალდამცავი პირების მიერ და კანონით გათვალისწინებული ზომების გატარება.

მეტი თვალსაჩინოებისათვის შემდგომში შემოთავაზებულ იქნება ტრადიციული და ელექტრონული საარჩევნო სისტემების მოდელები, აღწერილ იქნება არჩევნების დღის განმავლობაში ამომრჩევლის მიერ შესასრულებელი ყველა ნაბიჯი.

ახლა განვიხილოთ ცენტრალური და საოლქო საარჩევნო კომისიათა ოფისების მუშაობა.

საოლქო საარჩევნო კომისიებში უნდა მოეწყოს მედია დარბაზები, სადაც სრულიად გამჭვირვალედ მოხდება დაკვირვება არჩევნების მიმდინარეობაზე. აქ თავს მოიყრიან საოლქო საარჩევნო კომისიის წევრები, პარტიების, როგორც უმრავლესობის, ასევე უმცირესობის პარტიული წარმომად-გენლები, არასამთავრობო სექტორის წარმომადგენლები, მოწვეული უცხოელი დამკვირვებლები და, რაც ყველაზე მთავარია, ჟურნალისტები.

საოლქო საარჩევნო კომისიებში მოხდება შესაბამისი რაოდენობის დიდი ზომის მონიტორების დაყენება, რამდენი საარჩევნო უბანიც შედის ამა თუ იმ საარჩევნო ოლქში და თითოეულზე მიებმება საარჩევნო უბნებზე დამონტაჟებული IP კამერები.

ეს საშუალებას მოგვცემს ვაკონტროლოთ და ჩავიწეროთ უბნებზე არჩევნების მიმდინარეობა, გავაკონტროლოთ და საჯარო გავხადოთ მოსახლეობის აქტივობა და უსაფრთხოებისა და წესრიგის შესახებ მოვამზადოთ და შევიტანოთ შენიშვნები არჩევნების შესახებ მომზადებულ დასკვნაში.

ანალოგიური პრინციპით მოხდება ცენტრალურ საარჩევნო კომისიაში მუშაობის წარმართვა, ერთი სხვაობით, აქ სრულიად საქართველოს მასშტაბით და უცხოეთში მოწყობილი უბნებიდანაც მივიღებთ ანალოგიურ ვიდეო გამოსახულებას, რომელიც მულტიმედიურ მონაცემთა ბაზაში მოთავსდება და მოხდება მისი არქივაცია.

ასევე მოხდება საქართველოს იუსტიციის სამინისტროს საჯარო რეესტრის მიერ შექმნილი დინამიკური ციფრული რუკების გამოყენება, რომელზეც დატანილი იქნება საქართველოს მასშტაბით მიმობნეული საარჩევნო უბნები, რომელთა ცალკეული

ნაწილების გადიდებით მივიღებთ საარჩევნო უბნების მისამართებს და იმ ვიდეო სიგნალებს, რომელსაც IP კამერები მოგვაწოდებს. რუკას ექნება დაახლოებით ასეთი გრაფიკული სახე (ნახ.2.24-2.25).

გვინდა ასევე შემოგთავაზოთ ელექტრონული საარჩევნო სისტემის ფუნქციონირების ორგანიზების სტრატეგია და მონაცემთა გაცვლისა და მოძრაობის თანმიმდევრობა.

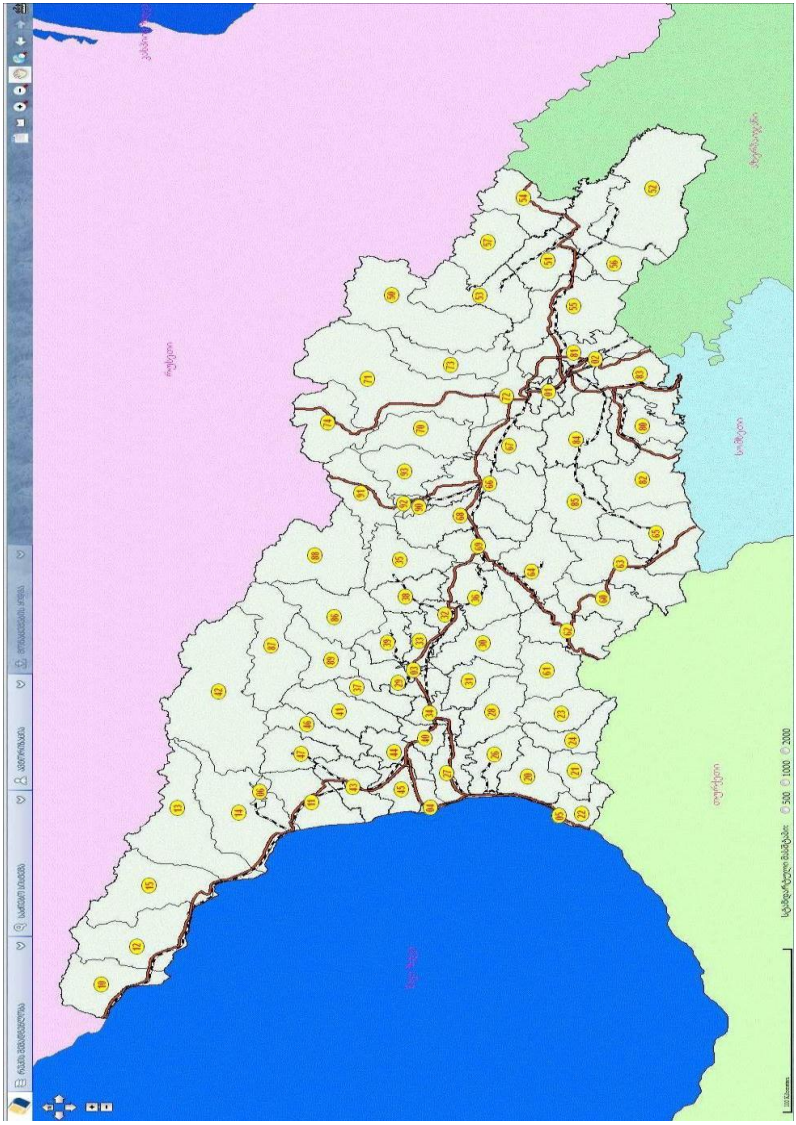
ინფორმაციის პირველადი წყარო როგორც ზემოთ აღვნიშნეთ, არის საარჩევნო უბნებზე მოქალაქეთა რეგისტრაციისას მიღებული მონაცემები, რომლებიც შესაბამისი თანმიმდევრობითა და წესების დაცვით მოთავსდება მულტიმედიურ მონაცემთა ბაზებში.

აღნიშნული მულტიმედიური ბაზები განთავსებული იქნება საოლქო საარჩევნო უბნებზე არსებულ სერვერებზე.

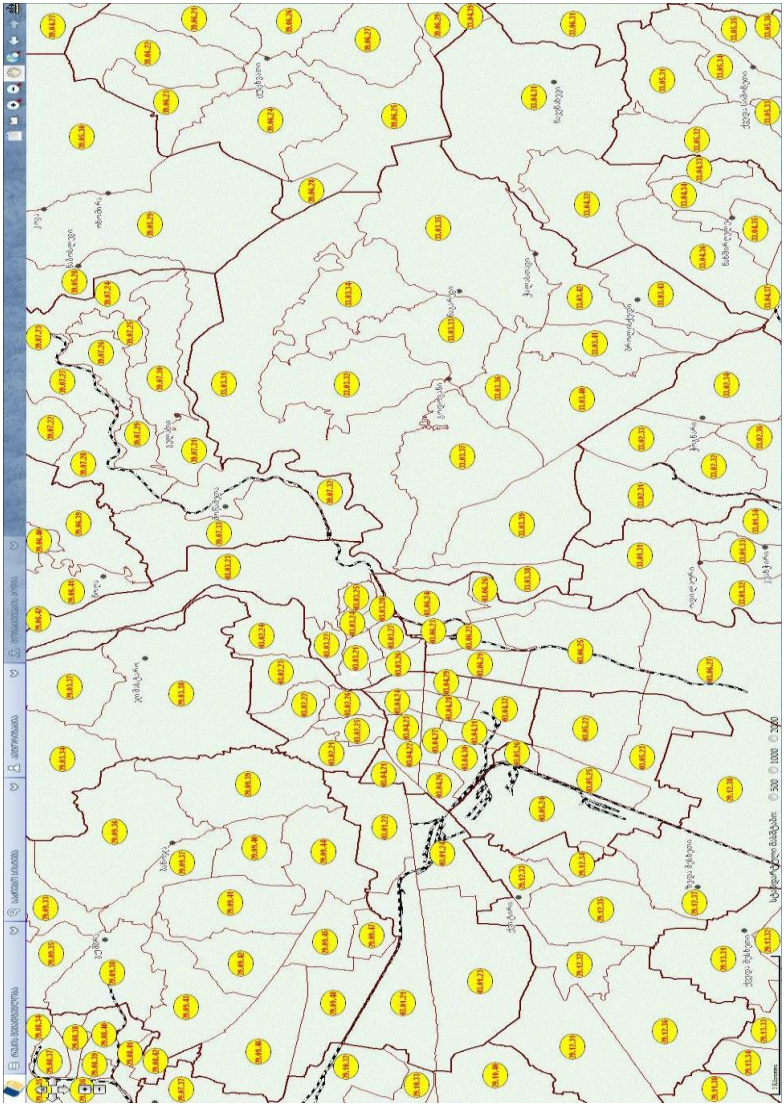
მიგვაჩნია, რომ დაუშვებელია უბნების პირდაპირი მიბმა ცენტრალური საარჩევნო კომისიის სერვერებზე და უნდა მოხდეს გარკვეული დროის მონაკვეთებში საოლქო საარჩევნო უბნების სერვერებიდან ინფორმაციის ტრანსფერი ცენტრალური საარჩევნო კომისიის სერვერებზე არსებულ მონაცემთა საცავებში.

აღნიშნული პროცედურა თავიდან აგვაცილებს ცენტრალური საარჩევნო კომისიის სერვერების „გაჭედვას“ და ფუნქციონირების შეჩერებას, რაც შეიძლება გამოწვეული იყოს ტრაფიკის გადატვირთვით.

ტრაფიკის გადატვირთვის თავიდან აცილების მიზნით ასევე გამოვიყენებთ და გავმართავთ აღნიშნულ სერვერებზე **SAN** ტექნოლოგიას.



ნახ.2.24. დინამიკური ციფრული რუკა
საარჩევნო უბნებით



ნახ.2.25. გადიდებული რუკა ინფორმაციით

SAN (Storage Area Network) დღეისათვის ფუნქციურად დომინანტური ტექნოლოგიაა მაღალი საიმედოობისა და სისწრაფის გამო. მასში გამოყენებულია ძალიან სწრაფი ოპტიკურ-ბოჭკოვანი შეერთება (Fiber Channel), ხოლო მონაცემთა შენახვის სისტემა განცალკევებულია ორგანიზაციის საერთო ქსელიდან.

ასეთი მოწყობის პირობებში იზრდება რეზერვირებისა და არქივაციის სიჩქარეც და საიმედოობაც.

მონაცემთა შენახვის აღნიშნული სისტემების მრავალ-ფეროვნება უზრუნველყოფს დამკვეთი კომპანიის მოთხოვნების დაკმაყოფილებას წვრილმანების გათვალისწინებითაც კი.

შემოთავაზებული მონაცემთა შენახვის სისტემით შესაძლებელია შემდეგი ამოცანების გადაწყვეტა:

- მონაცემთა ავტომატური რეზერვაცია და საჭიროების შემთხვევაში სწრაფი აღდგენა;
- დიდი მოცულობის მონაცემთა ცენტრალიზაცია და მართვის გაადვილება.

იშვიათად გამოყენებადი ინფორმაციის არქივაცია და საიმედო შენახვა რამდენიმე ათეული წლის განმავლობაში.

III თავი
მონაცემთა მოდელები და მულტიმედიური ბაზები

3.1. მონაცემთა მოდელები მულტიმედიისთვის

მონაცემთა მულტიმედიური ბაზების მართვის სისტემების (MMDBMS - MultiMedia DataBase Management Systems) ძირითადი ცნებების განვითარება უნდა დაიწყოს მონაცემთა მოდელების დისკუსიით. მონაცემთა მოდელი არის ერთგვარი სახე (ფორმალური) ენისა, რომელშიც მომხმარებელს შეუძლია სინამდვილის ფრაგმენტების აღწერა. მონაცემთა მოდელს მიეკუთვნება ასევე ოპერაციები (მეთოდები) მონაცემთა: წარმოების, წაკითხვის, მიერთების (ლინკის), მოწესრიგების (დალაგების), ცვლილების, ძებნის და წაშლის შესახებ. მონაცემთა მოდელი განსაზღვრავს მომხმარებლის თვალსაზრისს მონაცემებზე. აქ მისი რეალიზაცია დაფარულია და მონაცემთა ბაზების მართვის სისტემის მიერ შესაძლებელია მისი განხორციელება სხვადასხვა გზით.

დღეისათვის, ტექნიკისა და ტექნოლოგიების განვითარების მოცემულ დონეზე, პრაქტიკულად ფართოდ გამოიყენება ედგარ კოდის რელაციური მოდელი, იერარქიულ და ქსელურ კლასიკურ მოდელებთან შედარებით [66,67].

ამ პერიოდში შემოთავაზებულ იქნა აგრეთვე სხვა, ახალი სახის მოდელებიც, რომლებსაც ექსპერიმენტულადაც იკვლევდნენ მწარმოებლურობის ამაღლების თვალსაზრისით, რელაციურთან შედარებით. ასეთი მოდელების მაგალითებია სემანტიკური, ფუნქციური და ობიექტორიენტირებული [68,73]. მაგრამ მათი რეალიზაცია დაკავშირებული იყო დიდ დანახარჯებთან, რაც რელაციური მოდელის უპირატესობას ამტკიცებდა მისი სიმარტივის გამო. ამ თვალსაზრისით უკეთესი კონცეფცია გახდა

ისევ რელაციური მოდელის გაფართოებით მიღებული შედეგები, როგორიც იყო „ობიექტრელაციური“ მონაცემთა ბაზის მართვის სისტემები (ორ-მბმს) [32]. ისინი, თავიანთ შესაძლებლობათა გამო, მისაღები გახდა ასევე მულტიმედიური სისტემებისთვისაც. ამგვარად, მნიშვნელოვანია გამოკვლეულ იქნას მედიამონაცემთა შესაბამისი მონაცემთა ტიპების ისეთი განსაზღვრება, რომელიც ხელს შეუწყობს შემდგომში მათ ჩაშენებას განსხვავებულ მონაცემთა მოდელებში ან, სხვა სიტყვებით რომ ვთქვათ, მოითხოვება ამ მონაცემთა ტიპების განსაზღვრება მონაცემთა მოდელებისაგან დამოუკიდებლად. ჩვენ ქვემოთ განვიხილავთ ამ საკითხს ობიექტრელაციური და ობიექტორიენტირებული ბაზებისთვის.

ტრადიციული მონაცემთა მოდელების შემოთავაზება ფორმატირებული მონაცემების და მათი კავშირების სამართავად ასევე გამოიყენება მულტიმედიური მბმს-თვისაც. როგორც ცნობილია, მედიაობიექტები (ტექსტი, გრაფიკა, სურათი, აუდიო და ვიდეო ფაილები) გვხვდება ყოველთვის მათ აღწერასთან ერთად. ამასთანავე ისინი დაკავშირებულია ერთმანეთთან და სხვა ნებისმიერ ფორმატირებულ მონაცემებთან. ეს შეიძლება შემდეგი ფორმებით გამოვლინდეს:

- ატრიბუტები (არსების ან ობიექტების). ობიექტი (პერსონა, მანქანა და ა.შ.) შეიძლება დამატებით აღიწეროს მისი სურათის, ხელმოწერის, ხმის საშუალებით. მედიაობიექტი აღადგენს გამოსახული ობიექტის თვისებას;
- კომპონენტები კომპლექსური ობიექტებისთვის. დოკუმენტი შედგება ცვლადი რაოდენობის მედიაობიექტისგან: ტესქსტის ბლოკების, სურათების, გრაფიკის და შესაძლო აკუსტიკური კომენტარებისგან. ამჯერად მედიაობიექტები არის არა მხოლოდ ატრიბუტები, არამედ თვით ობიექტებიც. ამათ და დოკუმენტის ობიექტების შორის არსებობს რელაციური კავშირები,

აგრეგატული დამოკიდებულების მსგავსად. შეიძლება განვიხილოთ ასევე სინქრონიზაციის დამოკიდებულება ვიდეოფილმის სურათებს და ხმის ბილიკს შორის;

- ერთი და იგივე ინფორმაციის ალტერნატიული წარმოდგენა. ხშირად ერთი და იგივე ინფორმაცია აისახება სხვადასხვა მედიის საშუალებით, მაგალითად, ცხრილების და გრაფიკების სახით. არაა ყოველთვის შესაძლებელი ერთი სახის ინფორმაციის მიღება (გარდაქმნით) სხვა სახიდან გონივრული ხარჯებით. ამიტომ უფრო მისაღებია ორივე, როგორც მედიაობიექტი, იქნას ცხადად შენახული. ამ მედიაობიექტებს შორის ეკვივალენტობის დამოკიდებულების ისეთი სახე უნდა გამოიყენებოდეს, რომელიც გამომავალი მოწყობილობის წვდომის საშუალებით და მომხმარებლის მიდრეკილებით მბმს-თან, შეძლებს ერთი ან მეორე სახის არჩევას.

მონაცემთა მოდელი უნდა ცნობდეს დამოკიდებულებათა ამ განსხვავებულ ტიპებს მედიაობიექტებსა და ფორმატირებულ მონაცემებს შორის ან თვით მედიაობიექტებს შორის, და შემდეგ პროგრამის შესრულებისას ოპერაციებმა უნდა გაითვალისწინოს ისინი.

3.1.1. მონაცემთა ახალი ტიპები

ძირითადი ამოცანა მდომარეობს მონაცემთა ტიპების (ან კლასების) განსაზღვრაში მედიამონაცემებისთვის [32]. შემოთავაზებული ოპერაციების სიმრავლე, განსაკუთრებული ყურადღებით უნდა შეირჩეს ტექსტისთვის, რასტრული სურათისა და გრაფიკისთვის და ა.შ.

მონაცემთა ტიპი Text. ახალი საბაზო მონაცემთა ტიპის განსაზღვრა განვიხილოთ Text ტიპის მაგალითზე.

ცხადია, რომ მონაცემთა ტიპი Text უნდა იყოს მეტი, ვიდრე char[]. ამიტომაც Text ტიპის ობიექტს საკმაო რაოდენობის

ოპერაციები მიეწოდება, რომლებიც იღებენ განსხვავებულ ინფორმაციას ამ ობიექტიდან:

```
interface Text {
    public int length ();
    public int alphabet ();  // 0 == ISO Latin-1, . . .
    public int alphabetSize ();
    public int language (); // 0 == English, 1 == German, …
    public char charAt (int n);
    public byte[] getASCII ();
    public byte getEBCDIC ();
    public String getUnicode ();
    . . .
    };
```

Text-ს აქვს ყოველთვის ერთი სიგრძე (გამოყენებული სიმბოლოების რაოდენობა). ტექსტი ხშირ შემთხვევაში განიხილება როგორც სტიქონსტრუქტურირებული, რომელშიც უკვე განსაზღვრულია ქვეტიპი. სტრიქონებად დაყოფა შეიძლება ინფორმაციის მატარებელი იყოს როგორც ლექსებში. დაყოფა სიტყვებად დამოკიდებულია ენაზე და სიმბოლოების ერთობლიობაზე, მაგრამ პრინციპში ის ყოველ ტექსტში შესაძლებელია. ამ სტრუქტურებიდან გამომდინარე, შეიძლება შემდეგი ოპერაციების განხილვა, რომლებიც სისტემისთვისაც და ინსტრუმენტისთვისაც ცნობილია:

```
// გამოაქვს შედეგად სტრიქონების რაოდენობა ტექსტში
public int lineCount ();

// გამოაქვს შედეგად სიტყვების რაოდენობა ტექსტში
public int wordCount ();

// გამოაქვს ტექსტიდან სტრიქონები მითითებული ნომრით
public char [ ] line (int lineNo);
```

```
// გამოაქვს ტექსტიდან სიტყვა მითითებული ნომრით
public char [ ] word (int wordNo);
```

ცხადია, რომ Text ტიპის ობიექტისთვის მრავალი ოპერაციის შეთავაზებაა შესაძლებელი, რომლებიც ტესქტის მთლიანი თუ ცალკეული ნაწილების დასამუშავებლად იქნება გამოყენებული. მაგალითად,

```
public boolean print (Printer p);
public boolean display (window w);
```

ეს ოპერატორები შედეგად იძლევა მხოლოდ დასტურს, რომ მონაცემთა გამოტანის პროცესი დასრულდა წარმატებით.

განვიხილოთ ცვლილების ოპერაციების მაგალითები:

```
public void replaceLine (int lineNo, char [] newLine);
public void insertLine (int lineNo, char [] newLine);
public void concatenate (Text t);
```

Text ტიპის ობიექტის საწარმოებლად საჭიროა ოპერაციის მომზადება, რომელიც სხვა ტიპის ობიექტებიდან შეძლებს Text ტიპისაში ასახვას. მაგალითად, java ენაზე შეიძლება დაიწეროს კლასის შემდეგი კოდი კონსტრუქტორით:

```
class TextClass implements Text {
    int length,
    int charLength,
    int code,          // 0 == ASCII, 1 == EBCDIC, . . .
    int formatter,     // 0 == none, 1 == PostScript, . . .
    byte endOfLine,
    byte [] characters
};
. . .
}
```

ასეთი ოპერაციით მოითხოვება შესაძლოდ ფართო გამოყენება. იგი საშუალებას იძლევა ტექსტის მრავალი სახე

სხვადასხვა სისტემური გარემოდან გადასცეს (მონაცემთა ბაზის) Text ტიპის ობიექტს.

აქ განხილულ ოპერაციებს Text ტიპისათვის ჰქონდა მხოლოდ საილუსტრაციო ხასიათი. ისინი გვიჩვენებს, რომ საჭიროა მსგავსი ოპერაციების მომზადება მონაცემთა ასეთი ტიპის გამოყენებისას. მხოლოდ ასეთი ტექსტობიექტებით იქნება შესაძლებელი მულტიმედიურ მონაცემთა ბაზებში მუშაობა. საჭიროა სხვა ოპერაციების შემუშავებაც, რომლებიც განსაზღვრულ სამუშაო გარემოზე იქნება მორგებული. ეს შესაძლებელია უშუალოდ ტიპის გაფართოებით ან მისი სპეციალიზაციით მოხდეს.

მონაცემთა ტიპი Image აღიწერება ხშირად გამოყენებადი ოპერაციების საშუალებით. მისი შინაგანი სტრუქტურის შესახებ მომხმარებელთა უმრავლესობას ზოგადი წარმოდგენა აქვს, რომ იგი შედგება ფერთა ცხრილისა და პიქსელების მატრიცისგან. არსებული Image-ობიექტების წასაკითხად საჭიროა შესაბამისი ოპერაციების მომზადება, რომლებიც შეძლებს ცალკეულ კომპონენტზე მიმართვას:

```
interface Image {
  public int height ();
  public int width ();
  . . .
 }
```

დანარჩენ ოპერაციებს შეუძლია სურათების სტატისტიკური შეფასებების გაკეთება. მაგალითად,

```
public int pixelcount (byte [] pixelvalue);
```

ითვლის სიხშირეს, რომელიც განსაზღვრული პიქსელის მნიშვნელობა ხდება.

სურათის წაკითხვისას მონაცემთა ბაზიდან შესაძლებელია მისი შემადგენელი ელემენტარული ნაწილების (პიქსელების)

გამოძახება, ასევე შესაძლებელია გამოსახულების გაკეთება გამოთვლითი გარემოს სპეციფიურ მონაცემთა სტრუქტურაზე. მაალითად, SUN-კომპიუტერებზე რასტრული გამოსახულება გარკვეული დროის განმავლობაში იმართება მონაცემთა სტრუქტურაზე, სახელით Pixrect, რომელიც შეიცავს მრავალ ზემოაღნიშნულ ნაწილს. ეს შეიძლება ასევე პირდაპირ იქნას წარმოებული:

public Pixrect getPixrect ();

ინტერაქტიულ გარემოში სურათის ნახვა ყველაზე გავრცელებული გამოყენების სახეა, ამიტომ, ყოველი შემთხვევისთვის იგი უნდა იყოს მხარდაჭერილი საკუთარი ოპერაციით:

public boolean display (Device);

სურათის ცვლილება შესაძლებელია და მისი განხორციელებისას ყურადღება უნდა გამახვილდეს მთლიანობის შენარჩუნებაზე: მხოლოდ პირველადი მონაცემების ცვლილება შესაბამისი რეგისტრაციის მონაცემების ადაპტაციის გარეშე, ან პირიქით პროცესის დროს, უნდა იქნას თავიდან აცილებული. შემდეგი ოპერაციები ასახავს განახლების მაგალითს, ასეთი შესაძლებელია იყოს მრავალი.

public Image window (int x0, int y0, int x1, int y1);

ჩამოიჭრება ფანჯრის გარეთ მდებარე სურათის ნაწილი,

```
public Image replaceColormap {
   Code encoding;
int colormapLength;
int colormapDepth,
int [] [] colormap
};
```

სურათის ფერების ცხრილის შეცვლა შემდეგი სახით;

```
public Image replacePixelvalue {
   int x, int y,
   byte [] pixelvalue
};
```

იცვლება ცალკეული პიქსელის მნიშვნელობა.

როგორც ტექსტური ტიპისთვის იქნა ახსნილი, აქაც ეს ოპერაციები შეიძლება იყოს რეალიზებული პროცედურების სახით (ანუ შედეგის ტიპით void), თუ ეს სათანადო გადაწყვეტად იქნება მიჩნეული.

Image ტიპის მონაცემთა ობიექტის საწარმოებლად შემოთავაზებული ოპერაცია

```
class ImageClass implements Image {
    public ImageClas {
        int height,
        int width,
        int depth,
        float aspectRatio,
        Code encoding,
        int colormapLength,
        int colormapDepth,
        int [] [] colormap,
        byte [] pixelmatrix
    };
. . .}
```

მონაცემთა ტიპები Graphic, Sound, Video ანალოგიურად განისაზღვრება. აქ მათი განხილვა არაა წარმოდგენილი, ვინაიდან ახალ ასპექტებს არ შეიცავს.

მულტიმედიური მონაცემთა ტიპების გაფართოება მათი რეზიუმირებისათვის (შინაარსის მოკლე მიმოხილვისთვის) ხდება იმავე პრინციპით, როგორც ეს იყო ოპერაციების განსაზღვრის დროს, განსაკუთრებით შედარებითი ოპერაციებისათვის.

რადგან რეზიუმეების დროს საქმე ეხება დამოკიდებულ კომპონენტებს, რომლებიც მულტიმედიური ობიექტების გარეშე არ არსებობს, ისინი კაპსულირებული იქნება ობიექტთან. ეს მოითხოვს დამატებით ოპერაციებს მულტიმედიური ობიექტებისთვის. ქვემოთ ნაჩვენებია მაგალითი Image ტიპისთვის, რომლის მსგავსი იქნება ასევე სხვა მულტიმედია ტიპებისთვისაც. გამოსახულება გვიჩვენებს პროცედურებს.

```
interface Image {
    . . .
    public void newDescr (String descr);
    }
```

შეაქვს სურათის რეზიუმე descr-ში და ცვლის აქ მონაცემებს.

public void **extendDescr** (String descr);

აფართოებს ძველ რეზიუმეს (თუ არსებობს) descr -ში მოცემულს.

public int **descrLength** ( );

იძლევა რეზიუმის სიგრძეს.

public String **getDescr** ( );

გამოაქვს რეზიუმე სტრიქონის ფორმატში.

public boolean **contains** (String query);

იკვლევს, რეზიუმე ხომ არ შეიცავს query-ს, ანუ ძებნის გამოსახულებას, რომელიც query-ს შეესაბამება. ესაა წარმომადგენელი შედარების ოპერაციებიდან, რომლებიც სურათებისთვის და სხვა მულტიმედიატიპებისთვის შეიძლება იყოს განსაზღვრული.

3.1.2. კლასთა იერარქიის აგება განზოგადოების საფუძველზე

მულტიმედიური მონაცემთა ტიპების რეზიუმეების ოპერაციების განზოგადება შესაძლებელია გენერალიზაციის იერარქიის აგებით, რომელშიც „მშობელი“ მონაცემთა ტიპი (სუპერკლასი) MediaObject არის მოთავსებული.

მისთვის განსაზღვრულია ოპერაციები, რომლებიც ყველა მედიაობიექტისთვისაა გამოყენებადი. შეიძლებოდა ასევე, მაგალითად, ერთი (ვირტუალური ან აბსტრაქტული) მეთოდის output აგება, რომლის კონკრეტული რეალიზაცია იქნებოდა თითოეული არსებული მულტიმედიური მონაცემთა ტიპი.

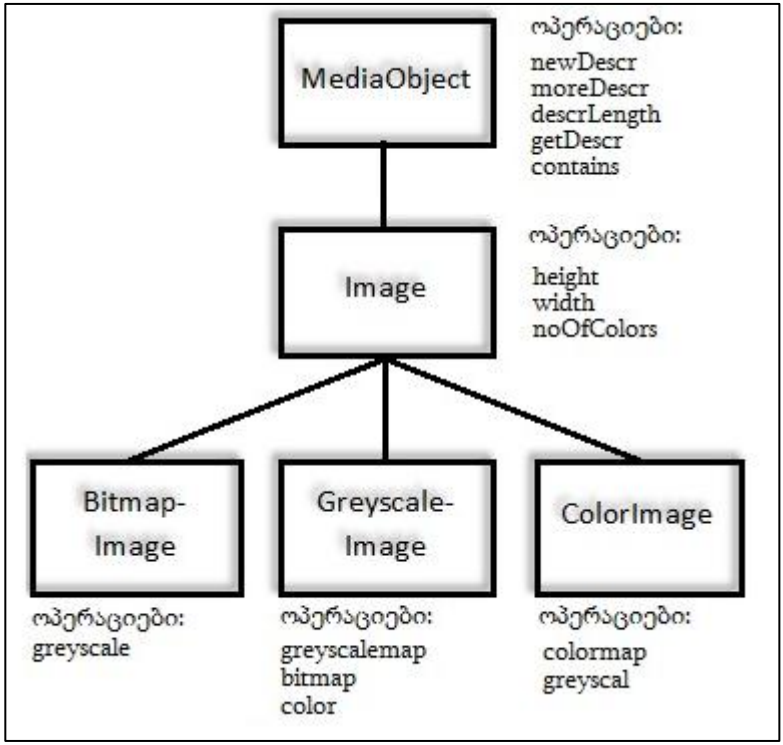
განზოგადების იერარქიის შემდგომი გამოყენება შესაძლებელია რათა სუბკლასების დახმარებით მოხერხდეს ერთგვარი სიბინძურის თავიდან აცილება, რომელსაც აქამდე Image -სთვის განსაზღვრული ოპერაციები მიუთითებდნენ.

ასე მაგალითად, არსებობს სურათები Pixel-ის სიღმით 1, რომლებიც არ იყენებენ ფერთა ცხრილებს. Pixel მნიშვნელობით 0 , ნიშნავს თეთრს, ხოლო 1 კი – შავს. პრინტერებისა და მონიტორების დიდი კლასისათვის (Bitmap-დისპლეები) ასეთი სურათები მეტად მნიშვნელოვანია.

მათთვის ისეთი ოპერაციების განსაზღვრა, როგორიცაა colormapLength და getColormap არის უაზრო.

ამავე დროს არსებობს სხვა ოპერაციები, რომელთა შემოტანა ასეთი ტიპის Bitmap-სურათებისთვის არის სასარგებლო [GS83].

ამგვარად, შემოთავაზებულია სამი ტიპის სურათებისთვის ცალკე სუბკლასების წარმოდგენა (ნახ.3.1).



ნახ.3.1. კლასთა იერარქია რასტრული სურათებისთვის

Image-ს სუბკლასები ასახავს განსხვავებულ ინფორმაციულ შინაარსს რასტრული გამოსახულებებისთვის. თუ წარმოვიდგენთ ერთ ფოტოსურათს როგორც Bitmap-სურათს, მერე როგორც შავ-თეთრს და, ბოლოს როგორც ფერადს, ინტუიციურად მივხვდებით, რომ ფერადი სურათი მეტ ინფორაციას შეიცავს, ხოლო Bitmap-სურათი – ყველაზე ნაკლებს.

ფერადი სურათის გარდაქმნას შავ-თეთრში ანგარიშობენ ფერის სიკაშკაშის მნიშვნელობით ან წაიკითხება სპეციალური ცხრილიდან და ის დაკავშირებულია ყოველთვის ინფორმაციის დაკარგვასთან. იგივეა შავ-თეთრი სურათის გადაყვანისას Bitmap-

სურათში. ასეთი ინფორმაციული დანაკარგები მხედველობაში უნდა იქნას მიღებული, როცა მათი წარმოდგენა იქნება საჭირო სრულიად განსაკუთრებულ გამომტან მოწყობილობებზე. ამისათვის არის გათალისწინებული ოპერაციები greyscale ColorImage-სთვის და bitmap GreyscaleImage--თვის.

ყოველი რასტრული სურათი შეიძლება იყოს Bitmap-სურათი, შავ-თეთრი ან ფერადი და იგი მიეკუთვნება ყოველთვის მხოლოდ ერთ სუბკლასს. ამიტომაც თითოეული ფლობს თავის განსაკუთრებულ კონსტრუქტორებს. აქ მიღებული რასტრული სურათების სპეციალიზაციები ამიტომ არის გადაუკვეთადი და სრული. მონაცემთა ბაზების სისტემას უნდა შეეძლოს ასეთი მთლიანობის პირობის დაცვის გარანტირება.

3.1.3. SQL/MM: მულტიმედიური მონაცემები და სტანდარტები

მულტიმედიურ მონაცემთა ტიპების დაპროექტების პროცესის უნიფიკაციის საკითხებმა ასახვა პოვა ჯერ SQL:1999 გაფართოებაში, შემდეგ კი ერთიან საერთაშორისო ISO/IEC 13249 SQL/MM სტანდარტში [65].

სტანდარტი ხუთ პაკეტად დაიყო: ფრეიმვორკი (ან ჩარჩო, სტრუქტურა), სრული ტექსტი, ვექტორული გრაფიკა, რასტრული სურათები და მონაცემთა ინტელექტუალური ანალიზი (Data Mining).

ამასთანავე, SQL:1999 სინტაქსის ტერმინების გამოყენების თვალსაზრისით, კლასების ცნებისთვის გამოყენებულ იქნება „მომხამარებელის-მიერ განსაზღვრული ტიპები“ (UDT – User-Defined Types) ხოლო ოპერაციებისთვის - „მომხმარებლის-მიერ განსაზღვრული ფუნქციები“ (UDF – User-defined Functions).

3.1.3.1. SQL/MM სრული ტექსტი

ეს პაკეტი განსაზღვრავს UDT FullText-ს ტექსტური მონაცემებისთვის და სხვა UDT FT_Pattern-ს საძებნი შაბლონებისთვის (ან მოდელებისთვის). FullText მიუთითებს ოთხ საძებნ მეთოდზე, რომელთაგან ორი განსხვავდება მხოლოდ მათი პარამეტრების ტიპებით. კერძოდ, ესაა ან სიმბოლოების ჯაჭვი (სტრიქონი) ან FT_Pattern-ის ტიპის შაბლონი. მეთოდების სახელები გადატვირთვადია (overloaded).

ორი Contains-მეთოდი ახორციელებს ლოგიკურ ძებნას. შედეგის სახით მიიღება მხოლოდ „დიახ" ან „არა". ხოლო ორი Rank-მეთოდი კი, მეორე მხრივ, განსაზღვრავს რიცხვს მცოცავი წერტილით, რომელიც გამოიყენება პოზიცირებისათვის (ranking).

ამას გარდა FullText-ს აქვს ორი კონსტრუქტორი. ერთი აწარმოებს ტექსტის ობიექტს სტრიქონიდან, მეორე კი დამატებით არეგისტრირებს ტესქტის ენას. დასასრულ, არსებობს კიდევ ერთი ფუნქცია FullText_to_Character სტრიქონების მისაღებად, ტექსტის გამოტანის მიზნით.

SQL:1999 სინტაქსით UDT-განსაზღვრება სტანდარტულად ჩაიწერება შემდეგი სახით:

```
create type FullText as (
    Contents character varying (FT_MaxTextLength),
    Language character varying (FT_MaxLanguageLength),
. . .
)
method Contains (pattern FT_Pattern)
      returns integer
method Contains (pattern character varying (FT_MaxTextLength))
      returns integer
method Rank (pattern FT_Pattern)
```

```
        returns double precision
    method Rank (pattern character varying (FT_MaxTextLength))
        returns double precision
    method FullText (String character varying (FT_MaxTextLength))
        returns FullText
    method FullText (
        String . . . ,
        Language character  varying (FT_MaxTextLength)
    )
        returns FullText;
    create cast (FullText as
        character varying (FT_MaxTextLength)
        with FullText_to_Character);
    create type FT_Pattern as
        character varying (FT_MaxPatternLength);
```

FT_Pattern მნიშვნელობები უნდა იყოს გამოსახულებები, აღწერილი ბეკუს-ნაურის ფორმის ენის საშუალებით. შეფასება აღიწერება ენის სიმბოლოების მიხედვით წესების მეშვეობით. განვიხილოთ მაგალითები ამ საკითხებზე.

ტექსტური ობიექტი აქ მოიცემა aText სახით და შინაარსით: „ეს სექცია წარმოადგენს SQL/MM სტანდარტს. ამ სტანდარტით განსაზღვრულია მულტიმედიური ობიექტების ტიპები და ქვეპროგრამები".

უმარტივესი ძებნის შაბლონი არის ერთი სიტყვა:

aText.Contains(' "სექცია" ') = 1

ეს გამოსახულება ასახავს შემთხვევას, როცა შედეგი „ჭეშმარიტია" („true" ანუ 1).

შესაძლებელია ძებნის განხორციელება სიტყვების სიმრავლითაც, სადაც გამოყენებულ იქნება სიტყვებსშორისი შემავსებლები (placeholder, wildcards), მაგალითად „ქვედა ტირე“.

aText.Contains(‘ “სექცია_“ ‘) = 0.

შედეგი მიღებულა ამჯერად „მცდარი“ (false ანუ 0) მნიშვნელობით.

ძებნის შაბლონებში შესაძლებელია აგრეთვე თეზაურუსის გამოყენებაც, მასზე მიმართვით. იგი გაფართოებული შაბლონით აფორმირებს ახალ სიტყვას, მაგალითად, მსგავსი სიტყვები, განზოგადებული სიტყვები, სპეციალური სიტყვები, სინონიმები ან წარმოებული:

```
aText.Contains ( ‘
      thesaurus “Informatik”
      expand synonym term of “Norm”
  ‘ ) = 1
```

აქ იგულისხმება, რომ „ინფორმატიკის“ თეზაურუსი სიტყვისათვის „Norm“ პოულობს სინონიმს „Standard“ (გერმანული ენა). სიტყვათა მიმდევრობა და დაცილება შესაძლებელია განისაზღვროს კონტექსტური შაბლონის ფორმით:

```
aText.Contains ( ‘
      (“მოხსენება“)  near “სტანდარტი”
      within 0 sentences in order
  ‘ ) = 1
```

დასასრულ, შესაძლებელია ასევე ე.წ. კონცეფციური შაბლონის (concept pattern) განხილვა ტექსტის მნიშვნელობაზე მისამართად:

```
aText.Contains ( ‘
      is about “Internationaler Standart
      yur Volltextsuche”
  ‘ ) = 1
```

ამ დროს ღიადაა დატოვებული, თუ is about როგორ იქნება რეალიზებული.

ყველა ზემოგანხილული მაგალითი იყო ცალკეული ძებნის-ფრაზები. შესაძლებელია მათი კომბინაციური გამოყენებაც კონიუნქციების და სხვა ლოგიკური ოპერაციების გაერთიანების საფუძველზე.

შეიძლება განვიხილოთ უმარტივესი შაბლონი, რომელიც მოთხოვნის მაგალითის კონტექსტში იქნება ნაჩვენები:

```
select * from myDocs
where Doc.Rank(‘ “ Standard” ‘) > 0.8.
```

ამგვარად, სისტემა უნდა აკონტროლებდეს (შეფასების თავალსაზრისით), რომ მოხდა ინფორმაციის სწორი ამოცნობა, ანუ სიტყვის, წინადადების, აბზაცის მიხედვით (კონტექსტური შაბლონებით). გაფართოებული შაბლონებით კი მიიღწევა სწორი (ანალოგიური, მსგავსი) სიტყვების პოვნა.

3.1.3.2. SQL/MM სივრცითი ტიპი (Spatial)

მონაცემთა სივრცითი ტიპი Spatial გამოიყენება “graphic” ტიპის ობიექტებისთვის.

განიხილება UDT-ები 2D-მონაცემებისა (წერტილი, წრფეები, სიბრტყეები) და მათი კოლექციებისთვის. მათ შეიცავს პროგრამები სივრცითი მონაცემების მანიპულაციის, ძებნის და შედარებისთვის, აგრეთვე კონვერტირებისთვის UDT და სიმბოლოებს ან ოროობით წარმოდგენებს შორის.

ცალკეული ტიპები ორგანიზებულია განზოგადებული იერარქიით. ფესვის ტიპს ეწოდება ST_Geometry. ყოველ გეომეტრიულ ობიექტს მიეკუთვნება ერთი SRID (Spatial reference system identifier), რომელიც ახასიათებს სივრცით ათვლის სისტემას. იგი ეფუძნება ცნობილ ათვლის სისტემებს:

გეოგრაფიული საკოორდინატო სისტემა (განედი და გრძედი), საპროექციო საკოორდინატო სისტემა (X-ით, Y-ით და Z-ით). ისინი აღიწერება, მაგალითად ასეთი სახის BNF-ით:

<geographic system> ::= <projected cs> | <geocentric cs>
<geographic cs> ::= GEOGCS <left delimiter>
<double quote> <name> <double quote> <comma>
<datum> <comma>
<prime meridian> <comma>
<angular unit>
<right delimiter>

ST_Geometry ტიპის კოლექციის ელემენტებისთვის და ST_Geometry ველის შიგნით უნდა იქნას შერჩეული ერთი და იგივე ათვლის სისტემა.

დეტალურად რომ დავახასიათოთ, არსებობს შემდეგი ტიპები:

- „ნულგანზომილებიანი“ - განიხილება წერტილი: ST_Point;
- ერთგანზომილებიანი ობიექტები არის ზოგადად ST_Curve, რომელშიც შესაძლებელია ქვეტიპების შექმნა, რომლებიც განსაზღვრავს ინტერპოლაციას ცალკეულ წერტილებს შორის: ST_LineString – წრფივი ინტერპოლაციაა; ST_CircularString – წრიული ფორმის და ST_CompoundString – შერეული ფორმისაა;
- ორგანზომილებიანი ობიექტები - ST_Surface, სადაც ST_CurvePolygon განისაზღვრება გარე და შიგა ST_CompoundString საზღვრებით, მაშინ, როცა ST_CurvePolygon იყენებს მხოლოდ ST_LineString საზღვრებს.

დამატებით არსებობს აგეთვე კოლექციის ობიექტები, რომლებიც აკეთებს იმას, რაც სხვა სისტემებისგან ცნობილია „დაჯგუფების“ (grouping) სახით. ისინი მოითხოვს, როგორც უკვე

აღინიშნა, ერთნაირ ათვლის სისტემას ყველა ელემენტისთვის. არსებობს ST_MultiPoint, ST_MultiCurve, ST_MultiLineString, ST_MultiSurface და ST_MultiPolygon, რომელთა სახელები ცხადად მითითებს თუ ელემენტის რომელი სახისთვის არის თითოეული დანიშნული.

ST_Geometry-სა და ქვეტიპებზე განსაზღვრულია შემდეგი მეთოდები: საშუალო (წერტილთა სიმრავლის საშუალო), სხვაობა და გაერთიანება, მანძილის გაანგარიშება, ტესტირება (contains, overlaps, touches, croses,…), და ათვლის სისტემის განსაზღვრა.

ამასათანავე ხდება აგრეთვე რამდენიმე სხვა მეთოდის დამატება ქვეტიპებისთვის, როგორებიცაა, მაგალითად, length ST_Curve-თვის, area და perimeter ST_Surface-თვის.

3.1.3.3. SQL/MM სტილი სურათი/ფოტო (Styl Image)

განვიხილოთ სურათის (ფოტოს) სტილის სტანდარტის გამოყენების საკითხი. თანამედროვე სტანდარტების შემოთავაზებით, მაგალითად, UDT-ს SI_StillImage გამოიყენება სურათის (ფოტოს) მონაცემებისთვის, SI_Feature – სურათების თვისებებისთვის და SI_FeatureList ასეთი თვისებების სიისთვის. SI_StillImage-ს შემთხვევაში, საინტერესოა, რომ ატრიბუტების სია, ანუ შინაგანი წარმოდგენა ღიაა და შესაძლებელია მასზე მიმართვა:

```
create type SI_StillImage  as (
    SI_content binary large object (SI_MaxContLength),
    SI_contentLength integer,
    SI_format character varying (8),
    SI_height integer,
    SI_width integer,
    . . .
)
```

ატრიბუტი SI_content შეიცავს სათაურს, ფერის ცხრილებს და სხვა მსგავს მონაცემებს, ანუ სარეგისტრაციო მონაცემებს და შესაძლოა ასევე აღწერის მონაცემებსაც. „Container“-ის სახით განიხილება მთლიანი სურათი.

ატრიბუტი SI_format ის ფორმატია, რომლითაც სურათი ინახება კონტეინერში, მონაცემთა მასივის სუფიქსის მსგავსად. განასხვავებენ „მხარდამჭერ“ და „მომხმარებლის_მიერ_ განსაზღვრულ“ ფორმატებს. მხარდამჭერი ფორმატები იკითხება თვით მბმს-ის მიერ ისე, რომ ხდება BLOB-ის შინაარსის ინტერპრეტაცია. მაგალითად, მას შეუძლია სურათის თვისებების ამოღება. მომხმარებლის მიერ განსაზღვრული ფორმატები კი, თავის მხრივ, იმართება მბმს-ით მხოლოდ როგორც სახელები. ინტერპრეტაციის საკითხი რჩება აპლიკაციის გადასაწყვეტი.

ცხდად ჩანს, რომ მონაცემთა დამოუკიდებლობა აქ არ თამაშობს განსაკუთრებულ როლს. ანუ ის მოთხოვნები, რომლებიც ზემოთ იქნა განსაზღვრული სხვა ტიპებისთვის, ნაწილობრივ ხორციელდება. როგორც ჩანს, იგი უფრო ორიენტირებულია ფაილური სისტემების (მასივების) მოდელებზე, ვიდრე მონაცემთა ბაზებზე ტიპებით.

ეს საკითხი შემდგომ აისახება ასევე ოპერატორებზეც. SI_StillImage-ს აქვს ორი კონსტრუქტორი, რომელთაგან ერთი შესასვლელზე ელოდება უბრალოდ BLOB-ს, ხოლო მეორე, პირიქით – BLOB-ის გარდა სხვა ფორმატის მონაცემებს.

ამიტომ არსებობს ცვლილებათა ორი მეთოდი: პირველი ჩაანაცვლებს შიგთავსს (Content-ს) ახალი BLOB-ით და შეიძლება იმედი ვიქონიოთ, რომ სხვა ატრიბუტებიც მთლიანობას მოერგება. მეორე მეთოდი ცვლის ფორმატს და ამან უნდა მოახდინოს Content-ის (შიგთავსის) შინაარსის კონვერტირება.

არსებობს აგრეთვე წაკითხვის ორი მეთოდი მინიატიურული სურათის (Thumbnails) საწარმოებლად:

method SI_StillImage (
content binary large object (SI_MaxContLength)
) returns SI_StillImage

method SI_StillImage (
content binary large object (SI_MaxContLength)
format character varying (. . .)
) returns SI_StillImage

method SI_setContent (
content binary large object (SI_MaxContLength)
) returns SI_StillImage

method SI_changeFormat (
targetFormat character varying (. . .)
) returns SI_StillImage

ფორმატის ცვლილების მეთოდები გამოიყენება მხოლოდ არსებული (მხარდაჭერილი) ფორმატებისთვის.

აღწერილი მონაცემებისთვის შესაძლებელია სურათის მახასიათებლების მიღება და შენახვა ცალკე სვეტებში სურათის გვერდით. SI_Feature საბაზო ტიპს აქვს შემდეგი ქვეტიპები:

- SI_AverageColor – ასახავს მთლიან სურათს ერთ ფერში, უბრალოდ საშუალო ფერით;
- SI_ColorHistogram – მართავს ფერთა ჯგუფების სიხშირეებს [32];

- SI_PositionalColor – მიიღება სურათის დაშლით მართკუთხედებად, რომლითაც განისაზღვრება შემდეგ საშუალო ფერი;
- SI_Texture – შეიცავს განმეორებადი ელემენტების სიდიდეებს, სიკაშკაშის ვარიაციებს და დომინირებად მიმართულებებს.

ყველა მახასიათებელი ხელმისაწვდომია SI_Score მეთოდით, რომელიც ითვლის სურათის დისტანციას მახასიათებლებში და იძლევა შედეგს 0 -:-1 ინტერვალში.

SI_Feature-ს ყველა ქვეტიპს აქვს მახასიათებლების ამოღების ფუნქცია (იმავე სახელით, როგორიც აქვს ქვეტიპს), რომელიც სურათს გამოიყენებს არგუმენტის სახით და იძლევა ქვეტიპის ობიექტს შედეგის სახით.

SI_AverageColor და SI_ColorHistogram ქვეტიპთა ობიექტებს შეუძლია ამასთანავე უშუალოდ იქნას აგებული კონსტანტების ჩაწერის საშუალებით. საილუსტრაციო მაგალითი მოცემულია საშუალო ფერის სინტაქსის აღწერით.

```
create type SI_Feature
method SI_Score (image SI_StillImage)
     returns double precision

create type SI_AverageColor under SI_Feature as
        (SI_ AverageColorSpec SU_Color)
method SI_ AverageColor (
        ReadValue integer,
        GreenValue integer,
        BlueValue integer
)  returns SI_AverageColor

create function SI_AveageColor (Image SI_StillImage)
     returns SI_AverageColor
```

სურათი შეიძლება აღიწეროს სხვადასხვა მახასიათებლით (თვისებებით), და მისი საშუალებით შეუძლებელია შედარების თვითშეფასების განხორციელება, ამიტომ შემოთავაზებულია შესაძლებლობა, რომ ისინი გაერთიანდეს ერთიან თვისება-მნიშვნელობის წყვილების სიაში. იგი ქმნის ტიპს SI_FeatureList. მას აქვს აგრეთვე SI_Score მეთოდი, რომელიც შეწონილ საშუალო მნიშვნელობას იძლევა ცალკეული Score შეფასებისთვის:

```
self.SI_Features[1].SI_Score(img) * self.SI_Weights[1]
+ self.SI_Features[2].SI_Score(img) * self.SI_Weights[2] + . . .
/ (self.SI_Weights[1] + self.SI_Weights[2] + . . .)
```

კონსტრუქტორის მიერ ხდება ზუსტად ერთი თვისების და ერთი წონის სიის ინიციალიზაცია, რომელიც SI_Append მეთოდით ამატებს მომდევნო თვისებას თავის წონასთან ერთად.

```
create type SI_FeatureList as (
    SI_Features SI.Feature array[SI_MaxFeatureNumber],
    SI_Weights double precision array[SI_MaxFeatureNumber]
)
method SI_FeatureList (firstFeature SI_Feature, weight double precision)
    returns SI_FeatureList
method SI_Append (feature SI_Feature, weight double precision)
    returns SI_FeatureList
```

განვიხილოთ ერთი მაგალითი, რომელშიც შეჯამებულია აღწერილი შესაძლებლობები SQL-ის კონტექსტში. წარმოდგენილია Logos რელაცია, რომელიც სურათის ერთი Logo ატრიბუტით უნდა იქნას გამოკვლეული. შესადარებელი ობიექტის სახით გამოიყენება bspLogo, რომლიდანაც მიიღება ტექსტურა და ფერთა ჰისტოგრამა. ამ მახასიათებლებიდან აიგება თვისებათა სია, რომელშიც ჯერ შეიტანება ტექსტურა წონით 0.8 (კონსტრუქტორი) და შემდეგ

ფერთა ჰისტოგრამა წონით 0.2 (SI_Append მეთოდი). შედგენილი თვისებების სიიდან გამოიძახება SI_Score და, კერძოდ, ველით Logo პარამეტრის სახით. მსგავსებისთვის შეწონილი საშუალო მნიშვნელობა უნდა იყოს 0.7-ზე მეტი.

```
select * from Logos
wher  SI_FeatureList (
        SI_Texture (bspLogo), 0.8
   ).SI_Append (
      SI_ColorHistogram (bspLogo, 0.2)
    ).SI_Score (Logo) > 0.7
```

დასასრულ, შეიძლება მოკლედ შევაჯამოთ SQL/MM სტანდარტის წინადადებები.

Spatial და Still Image-ს გამოყენების დროს გვხვდება ყველგან ST ან SI პრეფიქსები, რომლებიც Full Text შემთხვევაში არ არსებობდა. Full Text -ს აქვს ფუნქცია Rank, რომელიც Still Image-სთვის გვხვდება Score-ს სახით. ვინაიდან სპეციალიზირება ხდება ტიპებზე, შეიძლებოდა ასევე MM_Object-თვის განზოგადოება ან მსგავსი ხერხის შემოთავაზება, რომლის დროსაც ერთი (ვირტუალური) MM_Score მეთოდი თავის ადგილს იპოვნიდა.

3.2. ობიექტრელაციური მულტიმედიურ მონაცემთა ბაზის სისტემები

წინა პარაგრაფში შემოტანილ იქნა მონაცემთა ახალი ტიპები მულტიმედიური ობიექტებისთვის და იყო მცდელობა, შეძლებისდაგვარად, განხორციელებულიყო ჯერ არარსებული კავშირების ასახვა მონაცემთა მოდელზე. ამჟამად ჯერ კიდევ არ არსებობს მულტიმედიურ მონაცემთა (მაგალითად, მულტი-მედიური დოკუმენტები აგრეგაციული სტრუქტურებით) ობიექტების მოდელირების საშუალებები [32].

აქ შემოთავაზებულია მონაცემთა მოდელების იმ შესაძლებლობათა გამოყენება, რომლებშიც დამოკიდებულებები და აგრეგაციები (სხვადასხვა ფორმებით) ხელმისაწვდომია. ამას მივყავართ კითხვამდე, თუ როგორ იქნება ახალი მონაცემთა ტიპები ჩაშენებული ამ მონაცემთა მოდელების კონტექსტში და როგორი სახე ექნება შემდგომ მათ გამოყენებას.

ეს საკითხი, პირველ რიგში, გამოკვლეულ იქნება რელაციური ბაზების სისტემის მაგალითზე. დღეს ბევრს საუბრობენ ობიექტრელაციურ სისტემებზე, ამასთანავე მათი გაფართოებები არის ძალზე სასარგებლო მულტიმედიური ობიექტებისთვის. ასეთი სისტემების საფუძველი ყოველთვის რელაციური მოდელებია. მათ უდავოდ აქვს უდიდესი პრაქტიკული მნიშვნელობა.

ჩვენ ქვემოთ დავახასიათებთ თავდაპირველად მათ იმ განსაკუთრებულ თვისებებს, რომლებიც მულტიმედიისთვისაა მთავარი, შემდეგ ნაჩვენები იქნება თუ როგორ პროექტდება სქემები, რომლებიც მულტიმედიური ობიექტების ჩადგმულ მონაცემთა ტიპებს გამოიყენებს. დაბოლოს, ნაჩვენები იქნება, თუ როგორ შეიძლება SQL-მოთხოვნების ენით, რომელიც ასეთი სისტემების მნიშვნელოვანი ენაა, განხორციელდეს მონაცემებთან მიმართვა ობიექტრელაციურ მონაცემთა ბაზებში.

3.2.1. მოკლე დახასიათება

ობიექტრელაციური მონაცემთა ბაზების სისტემა (ორმბს) წარმოადგენს მცდელობას, რელაციური მბს ისე განვითარდეს და ახალი კონცეფციებით გამდიდრდეს, რომ მან შეძლოს, მინიმუმ, ფუნქციონალთა ერთი ნაწილის შემოთავაზება, რომელიც აქამდე ცნობილი იყო ობიექტორიენტირებული მბს-თვის. ესენია კერძოდ, ობიექტთა იდენტურობა, მომხმარებელთა მიერ განსაზღვრებადი მონაცემთა ტიპები და ტიპთა იერარქიები. თუმცა ძირითადი

სტრუქტურა ყველა მონაცემთა ორგანიზაციის და მონაცემთა შენახვისა რჩება ისევ ცხრილები. ასე რომ, ყველა ეს ახალი გაფართოება უნდა დაექვემდებაროს, როგორც ადრე, ამ (წინა) კონცეფციას.

მულტიმედიური მონაცემებისთვის თანამედროვე სტანდარტებით განიხილება დიდი ობიექტები (large objects) ორი ფორმატით: სიმბოლოების ობიექტი (character large object - CLOB) და ბინარული ობიექტები (binary large object - BLOB).

არსებობს გარკვეული შეზღუდვები ამ ტიპის ატრიბუტებისთვის: ისინი არ შეიძლება იყოს განსაზღვრული როგორც პირველადი გასაღებური ატრიბუტები, UNIQUE ან მეორეული გასაღებური ატრიბუტები. აგრეთვე არ შეიძლება მათი გამოყენება GROUP BY ან ORDER BY კონსტრუქციებში.

განსაკუთრებით მნიშვნელოვანია სტრუქტურირებული, მომხმარებელთა მიერ განსაზღვრული ტიპები (UDT's). ეს ტიპები შეიცავს ატრიბუტებს და მეთოდებს. ასევე ძველ ფუნქციებს და პროცედურებს, რომლებიც 1996 წლიდან SQL92-ის სტანდარტის გაფართოებით იქნა განსაზღვრული, შეეძლოთ ასეთი ტიპის პარამეტრებით სარგებლობა [70].

პროცედურები განსაზღვრული იყო ისე, რომ ისინი შედეგის მნიშვნელობას უკან არ აბრუნებდა, მაგრამ შემავალი და გამომავალი პარამეტრები შეიძლება ჰქონოდა.

ფუნქციები, პირიქით, აბრუნებს უკან მიღებულ შედეგებს და აქვს მხოლოდ შემავალი პარამეტრები.

ტიპები შეიძლება იყოს ორგანიზებული იერარქიებად. ამავე დროს, ტიპები მიეკუთვნება ეგზემპლარულს (instantiable), თუ მათ შეუძლია უშუალოდ შედეგის მნიშვნელობების მოცემა. წინააღმდეგ შემთხვევაში ტიპები აბსტრაქტული ან ვირტუალურია, რაც ნიშნავს, რომ მათ შეუძლია მხოლოდ ქვეტიპების მნიშვნელობებზე მითითება. ქვეტიპების განსაზღვრა

ამით თავიდან აცილებულია, რადგანაც ტიპი თვითონ ასრულებს ამ ფუნქციას.

იერარქიულობის ეს პრინციპი არის ურთიერთშენაცვლების: სადაც ტიპის მნიშვნელობა შეიძლება მოთავსდეს, იქ შეიძლება ასევე ქვეტიპების მნიშვნელობათა არსებობა. იმისათვის, რომ ეს შესრულდეს, ტიპის ატრიბუტები და მეთოდები ასევე წვდომადი უნდა იყოს მისი ყველა ქვეტიპისთვის, ანუ მემკვიდრეობითობის თვისება უნდა იყოს რეალიზებული.

განვიხილოთ მაგალითი ტიპისა და ქვეტიპებისათვის.

```
create type Student
    under Person
     as (
         MatrNr  integer,
         Studienfach varchar(30),
    )
    instance method Durchschnittsnote()
          return real
          language SQL
          deterministic
          contains SQL
    . . . ;
```

მეთოდები მხოლოდ ცხადდება ტიპების აღწერაში, ანუ განისაზღვრება მათი სიგნატურები. რეალიზაცია შეიძლება მოგვიანებით განხორციელდეს (ბრძანება create method). ჩვეულებისამებრ, განასხვავებენ ეგზემპლარულ მეთოდებს (instance method) კლასთა მეთოდებს (static method).

დამატებითი ინფორმაცია მეთოდისთვის შემდეგია:

- returns – განსაზღვრავს უკან დასაბრუნებელი მნიშვნელობის ტიპს. იგი აღწერილია create-method ბრძანებაში;
- language SQL – მიუთითებს, რომ მეთოდის ტანის რეალიზაცია მთლიანად ხორციელდება SQL -ში;

- deterministic – მიუთითებს, რომ უშუალოდ ერთმანეთის მომდევნო გამოძახებები ერთი და იმავე პარამეტრების მნიშვნელობებით ყოველთვის ერთსა და იმავე შედეგებს იძლევა;
- contains SQL – მიუთითებს, რომ მეთოდის რეალიზაცია შეიცავს SQL-ბრძანებას, რომელიც არ ეხება ეგზემპლარის ან კლასის მონაცემებს, არამედ ეხება მხოლოდ სხვა მონაცემებს. ალტერნატიული ინფორმაციაა no SQL, როცა არაა მოცემული SQL-ბრძანება; reads SQL data, როცა ეგზემოლარის ატრიბუტები მხოლოდ წაკითხულ უნდა იქნას; modifies SQL data, როცა ატრიბუტები ასევე უნდა შეიცვალოს;
- returns null on null input – მიუთითებს, რომ შედეგი არ ბრუნდება უკან, როცა შემავალი პარამეტრია Nullwert (zero values);

ერთი ტიპის ყოველი ატრიბუტისთვის ორი მეთოდი ავტომატურად გენერირდება:

დამკვირვებელი (Observer) პასუხისმგებელია წაკითხვის წვდომისთვის.

```
instance method MatrNr ()
    returns integer
    language SQL
    deterministic
    contains SQL
```

ვინაიდან მეთოდის გამოძახებისას პარამეტრების გარეშე ყოველთვის ხდება ფრჩხილების გამოტოვება, ამ მეთოდის ვიზუალური გამოძახება არ განსხვავდება ატრიბუტებთან ნორმალური წვდომისგან: Student1.MatrNr.

მუტატორი ახდენს ატრიბუტის მნიშვნელობის ცვლილებას (ჩანაცვლებას):

```
instance method MatrNr (new value)
     returns Student
```

```
        self as result
        language SQL
        deterministic
        contains SQL
        returns null on null input
```

შემდეგი მაგალითი უფრო გამოკვეთილად შეესაბამება მულტიმედიურ შემთხვევას:

```
create type ImageType
        under <Supertyp>
        as  (< Attributliste> )
        static method countImages ()
        returns integer
        language SQL
        deterministic
        contains SQL
instance method height ()
        returns integer
        language SQL
        deterministic
       rads SQL data
```

კიდევ ერთხელ, საბოლოოდ უნდა გაესვას ხაზი იმას, რომ კორტეჟები და ცხრილები შეუცვლელად თამაშობს მნიშვნელოვან როლს: იგი უშუალოდ დაკავშირებულია კორტეჟების შეტანასთან ცხრილებში, სხვა ობიექტები ან მნიშვნელობები არ იქმნება.

ასევე ყოველთვის შესაძლებელია მოთხოვნები ცხრილებისადმი, რომლებიც გამოსასვლელზე იძლევა ისევ ცხრილებს.

3.2.2. ობიექტრელაციური ბაზის სტრუქტურები მულტიმედია ობიექტისთვის

ობიექტრელაციურ ბაზის სქემაში დასაშვებია მონაცემთა ახალი ტიპების გამოყენება მულტიმედიური ობიექტებისთვის, როგორც მნიშვნელობათა არეები (domains). ატრიბუტები შეიძლება იყოს ტიპებით text, image და სხვა. მაგალითად, ყოველი ამომრჩევლისთვის კორტეჟში უშუალოდ ჩაისმება პირადი ნომერი, გვარი, დაბადების თარიღი, და ა.შ., ასევე ფოტოსურათი, თითების ანაბეჭდი, ხმა და მოხდება მისი შენახვა:

```
Voter ( Pers_N integer,
        PersName varchar (50),
        BirthsDate date,
        Adress varchar (100),
            . . .
        PersFoto image,
      PersFingerprints image,
      PersVoice sound)
```

დავარქვათ ამ სტრუქტურას, პირობითად, რელაციური სქემა (ტიპი_1), რათა შემდგომში შემოტანილი სხვა ტიპებისგან განვასხვავოთ.

ნორმალურ ფორმათა თეორიის საფუძველზე, სქემათა ოპტიმალური სტრუქტურის მისაღებად, ხშირად საჭიროა ერთი რელაციის დეკომპოზიცია ორ ან მეტ რელაციად, რომლებშიც მოხდება შესაბამისი კორტეჟების გადანაწილება, პირველადი და მეორეული გასაღებური ატრიბუტების განსაზღვრა და ა.შ.

ჩვენი მაგალითისთვის შესაძლებელია ამომრჩევლის პირადი ტექსტური მონაცემების ერთ რელაციაში შენახვა, ხოლო მულტიმედიური მონაცემებისა – მეორეში. ქვემოთ ნაჩვენებია ეს შემთხვევა:

```
Person ( Pers_N integer,
        PersLastName varchar (50),
        PersFirstName varchar (30),
        BirthsDate date,
        Adress varchar (100),
            . . .
        )
 Voters_MM_Daten( Pers_N integer,
          PersFoto image,
        PersFingerprints image,
        PersVoice sound)
```

მიღებული სტრუქტურა არის რელაციური სქემა (ტიპი_2). ამ შემთხვევვაში ამომრჩევლის ვინაობა (გვარი, სახელი,...) უკავშირდება მულტიმედიური მონაცემების რელაციას პირველადი გასაღებური ატრიბუტით Pres_N. ამ ორი რელაციის საფუძველზე მოთხოვნების დასამუშავებლად საჭირო იქნება „Join” გაერთიანების ოპერაციის გამოყენება.

არსებობს შემთხვევები, როდესაც რელაციებს შორის არსებობს M:N დამოკიდებულება. ამ დროს საჭიროა რელაციებსშორისი კავშირების რეალიზაცია დამატებითი ინდექსების საფუძველზე, მათ შორის პირველადი და მეორეული გასაღებებითაც.

განვიხილოთ მაგალითი, რომელიც ასახავს მაჟორიტარი დეპუტატის (Majority_Deputy) განაწილებას რეგიონის (Region) მხარის (Area) საარჩევნო უბნებზე (Polling_Districts).

ერთი დეპუტატი კენჭს იყრის რამდენიმე საარჩევნო უბანზე, ასევე ერთ საარჩევნო უბანზე რამდენიმე კანდიდატია, ანუ საქმე გვაქვს M:N დამოიკდებულებასთან:

```
Majority_Deputy ( majority_deputyID integer,
        Deputy_Name varchar (50),
        . . .
        Deputy_Foto image
       )
Polling_District ( polling_districtID integer,
        Polling_DistrictNr varchar (20),
        polling_district_AddressName varchar (100),
        areaID integer,
        RegionID integer
      )
Results_List (polling_districtID integer,
        majority_deputyID integer,
        number_votes integer,
        Percentage double,
        Position integer
      )
```

მივიღეთ რელაცირი სქემა (ტიპი_3). ამგვარად, შეიძლება შევაჯამოთ შედეგები შემდეგი სახით:

- სამივე ტიპის რელაციური სქემა მულტიმედიურ ობიექტებსა და არსებს შორის შეიძლება აისახოს 1: 1, 1 : n, m : n დამოკიდებულებათა სახით, სპეციალური სემენტიკის გარეშე;
- მულტიმედიური ობიექტები შეიძლება გამოვლინდეს როგორც ატრიბუტები ან როგორც არსები;

- მიმართვა ზოგიერთ რელაციურ სქემაზე შეიძლება იყოს მოუხერხებელი და მოითხოვდეს რამდენიმე გაერთიანების ოპერაციის (Join) გამოყენებას.

3.2.3. მოთხოვნების დამუშავება ობიექტრელაციურ მონაცემთა ბაზებში

რელაციურ მონაცემთა ბაზების შესახებ, რომლებშიც ტრადიციული, ტექსტური ტიპის ატრიბუტების გარდა არის აგრეთვე მულტიმედიური ტიპებიც, როგორიცაა image, graphics და ა.შ., აქამდე ზედაპირულად იყო დახასიათებული. აქვე ნახსენები იყო მომხმარებლისთვის მოუხერხებელი გამოყენების საშუალება გაერთიანების ოპერაციის სახით.

იმისათის რომ აქ შედარებით ზუსტი სურათის გადმოცემა მოხერხდეს, საჭიროა წარმოდგენილ იქნას ახალ ტიპებზე განსაზღვრული ოპერაციების სავარაუდო ჩანერგვა (embedding) რელაციური მოთხოვნის ენაში. ამას გვთავაზობს SQL ენა, რომელიც სტანდარტის სახით დამკვიდრდა მოთხოვნათა ენებისთვის.

განვიხილოთ მარტივი, ორატრიბუტიანი რელაციის მაგალითი:

```
Aerial_image ( Nr integer,
               Picture image )
```

კორტეჟების შესატანად ბაზაში საჭიროა SQL-ენის insert-ოპერატორის გამოყენება. ამ დროს ცალკეული ატრიბუტების მნიშვნელობები გადაეცემა როგორც კონსტანტები ან პროგრამის ცვლადები.

მაგალითად,

```
insert into  Aerial_image
values (:nr, image(:pr, :cm ));
```

სადაც pr-ში ინახება სურათი, ხოლო cm-ში ფერთა ცხრილები.

გამოსახულებაში ორი წერტილის შემდეგ დგას პროგრამული ცვლადები, რითაც ისინი სინტაქსურად განსხვავდება რელაციების და ატრიბუტების სახელებისგან.

value წინადადების პირველი ჩანაწერი ეკუთვნის ატრიბუტის ნომერს და უნდა იყოს integer ტიპის. მეორე ჩანაწერი ეკუთვნის სურათს და ტიპი image. კომპილატორი ცნობს image ტიპის ოპერაციების პარამეტრების და შედეგის ტიპებს და შეუძლია შესაბამისი შემოწმების ჩატარება. ეს პრინციპი ძალაშია შემდგომ განხილული ოპერატორებისთვისაც.

თუ კორტეჟი სურათის ატრიბუტებით ერთხელ უკვე შენახულია მონაცემთა ბაზაში, მაშინ შესაძლებელია მათი SQL-ის update ბრძანებით ცვლილება.

```
update  Aerial_image
set  Picture = Picture.replaceColormap(:yuv, 4096, 24, : cm )
where Nr = 1286;
```

შინაარსობრივი მონაცემების მისაღებად შესაძლებელია შემდეგი სახის ბრძანების ჩაწერა:

```
update Aerial_image
set Picture = Picture.newDescr (
      „მოედანს, რომლის ცენტრში ძეგლი და გარშემო\n
       ყვავილებია, უერთდება ხუთი ქუჩა“
 where Nr  = 1234;
```

განსაზღვრული კორტეჟების მოსაძებნად (განსაზღვრული სურათებისთვის) შესაძლებელია ჩვეულებრივი SQL გამოსახულებების გამოყენება. ატრიბუტთა მნიშვნელობების შედარება კონსტანტებთან, რომელიც ამ დროს მთავარ როლს თამაშობს, დასაშვებია, უპირველეს ყოვლისა, მხოლოდ

სტანდარტული ტიპებისთვის. მულტიმედიური ტიპებისთვის კი არსებობს სპეციალური შედარების ოპერაციები.

პროგრამაზე გადაცემის დროს image ტიპის ატრიბუტებს არ შეუძლია უშუალოდ პროგრამის ცვლადებზე მინიჭება, რადგან ცვლადთა ტიპები სხვადასხვა დანართში შეიძლება სხვადასხვა იყოს. პირიქით, კომპონენტების ამორჩევა და მასთან დაკავშირებული ტიპების შერჩევა (ადაპტაცია) ხდება ცხადად შესაბამისი image - ოპერაციებით:

```
select Picture.getPixrect(), Picture.getColormap()
into :pr,  :cm
from Aerial_image
where Nr = :k;
```

ამ მაგალითში k-ცვლადში მოცემულია სურათის ნომერი, რომელშიც სურათი (Picture) იქნება გამოძახებული მონაცემთა ბაზიდან Pixrect ფორმატში. შესაძლებელია ასევე სურათის ატრიბუტების თვისებათა შერჩევა, თუ იგი წინასწარ image-ოპერაციების გამოყენებით მონაცემთა ტიპებისთვის შეიქმნა, რომლებზეც განსაზღრულია შედარების ოპერაციები:

```
select Picture.height(),  Picture.width()
into :hoehe,  :breite
from  Aerial_image
where Picture.pixelcount( :dunkelbraun) < 1000
and Picture.noOfColors() > 4095;
```

შეიძლება ალტერნატიული ვარიანტის განხილვაც image - მონაცეთა ტიპის შედარების ოპერაციისთვის:

```
select Picture.description(), . . .
from Aerial_image
where Picture.contains(“ცენტრში ძეგლი”);
```

ეს, უპირველეს ყოვლისა, უჩვენებს მომხმარებლის ინტერფეისის სიმარტივეს. ზემოთ აღწერილი select-ბრძანება იძლევა სურათს ნომრით Nr1234, ვინაიდან ამ აღწერაში არის სიტყვები „ცენტრში ძეგლი“ ნახსენები და იგი როგორც საძებნი გამოსახულება „ცენტრში ძეგლი“, ისე მოიაზრება.

შემდეგ შენახვის და მოთხოვნის შემთხვევებში შეიძლება გამოიცეს შეცდომის შეტყობინება, როგორიცაა, მაგალითად, „ამ სიტყვას არ ვიცნობ“ ან „ეს ტექსტი არ მესმის“ და სხვა. ესაა ის ღირებულება, რომლითაც მარტივი ტექსტების შედარებისგან განსხვავებით მომხმარებლის ინტერფეისი საძებნი პროცედურის უკეთესი ხარისხით გამოირჩევა.

instert – ბრძანებაში დაუშვებელია კომბინაცია values-წინადადებისა (კონსტანტების ან პროგრამის ცვლადების მონაცემები) და ქვეარჩევის (subselect) (ატრიბუტების ახალი მნიშვნელობების შეკრება მონაცემთა ბაზისთან მიმართვის შესახებ). ასეთი პროცედურა უფრო მაშინაა საჭირო, როდესაც სურათის ნაწილი, ამოღებული მონაცემთა ბაზიდან, უნდა იქნეს შენახული სხვა კორტეჟში.

ამ ახალი კორტეჟის ფორმატირებული ატრიბუტები მიიღება არა მონაცემთა ბაზიდან, არამედ პროგრამებიდან. მიზანშეწონილია კონსტანტების მიწოდება select-წინადადებაში, რაც თუმცა არც ძალიან გასაგებად გამოიყურება:

```
insert into Aerial_image
        select  :neueNr, Bild window (50,50,100,100)
        from Aerial_image
        where Nr = :alteNr;
```

update – ბრძანების სემანტიკა, სპეციალური set-წინადადება, არის მნიშვნელობის ჩანაცვლება (აღდგენა) და არა მნიშვნელობის მოდიფიკაცია. მნიშვნელობის მინიჭების კონსტრუქციის მარჯვენა

ნაწილში შეიძლება ნებისმიერი სირთულის გამოსახულება იყოს, რომელსაც უნდა ჰქონდეს სწორად შერჩეული შედეგის ტიპი. მაგალითად:

```
update  Aerial_image
apply Picture.replaceColormap ( :cm);
where Nr = 1234;
```

SQL – ბრძანებები პროგრამული ენების სინტაქსის მსგავსად პროცედურების გამოძახებას ჰგავს და არა ფუნქციურ გამოსახულებას, რომელსაც მნიშვნელობა გააჩნია. ამიტომაც ბრძანებები, როგორც, მაგალითად, ქვემოთაა ნაჩვენები, ძალზე მგრძნობიარეა მონაცემთა ახალი ტიპების მიმართ, რათა თავიდან იქნას აცილებული შრომატევადი დუბლირების პროცესები:

```
var := select . . . from . . . where . . . ;
writeScreen(select Picture.pixelmatrix ( ) );
```

მიზანშეწონილია, რომ მეთოდები აღიწეროს დეტალურად. ეს იმას ნიშნავს, რომ მოხდეს განსხვავება წასაკითხსა და ჩასაწერს შორის და, მულტიმედიური კონტექსტის თვალსაზრისით, უპირველეს ყოვლისა, განასხვავონ დროზე დამოკიდებული და დამოუკიდებელი პროცესები.

რელაციური მონაცემთა ბაზები, მოთხოვნების დამუშავების თვალსაზრისით, წლების განმავლობაში ასრულებს განსაკუთრებულ როლს, სხვა ახალი მონაცემთა ბაზებისგან განსხვავებით.

მათი გაფართოება ახალი ტიპის, მულტიმედიური მონაცემებით ძალზე ეფექტურია და პრაქტიკულად ღირებული.

მომხმარებელს, მათი გამოყენების მიზნით, სჭირდება ამ მიმართულებით დამატებითი ცოდნის მიღება.

3.3. ობიექტორიენტირებული მულტიმედიური მონაცემთა ბაზის სისტემები

პირველი ნაშრომები მულტიმედიურ მონაცემთა ბაზების შესახებ გამოჩნდა 80-ან წლებში, როდესაც მონაცემთა რელაციური მოდელების თემატიკა, როგორც ახალი მეცნიერული მიმართულება, ძალზე აქტუალური და დომინირებადი იყო [67, 71-73]. გასაოცარი არაა, რომ ამ დროს მულტიმედიურ მონაცემთა ბაზების სისტემების პირველი პროექტები მოიაზრებოდა როგორც აუცილებლად ობიექტორიენტირებული სისტემები [74,75]. მაგრამ ამ პერიოდში ვერ მოხერხდა ასეთი სისტემების პრაქტიკული რეალიზაცია.

ამ პრობლემებზე მუშაობა და მნიშვნელოვანი მიღწევები მიღებულ იქნა 90-ანი წლებიდან, როდესაც ODMG (Object Data Management Group) ჯგუფმა წარმოადგინა რელაციური ბაზებისთვის SQL-ის მსგავსი ეფექტური კონცეფცია [76].

3.3.1. მონაცემთა მულტიმედიური ტიპების ჩაშენება

ობიექტორიენტირებულ სისტემებში მულტიმედიური მონაცემთა ტიპების ასახვა ხდება უშუალოდ როგორც კლასები. ეგზემპლარები უნდა იყოს ინკაფსულირებული ისე, რომ წვდომის განხორციელება შეიძლებოდეს მხოლოდ კლასის მეთოდებზე.

როგორც ობიექტრელაციურ მოდელში, ეს კლასები შეიძლება იყოს ორგანიზებული განზოგადებული (გენერალიზებული) იერარქიით. ისინი გამორიცხავს საერთო მეთოდების განმეორებად განსაზღვრებებს და ნებას იძლევა მონაცემთა სპეციალური ტიპების შესატანად, რომლებისთვისაც დამატებითი მეთოდებია განსაზღვრული.

მონაცემთა ტიპები Text, Image, ასევე Graphics, Audio და Video განიხილება როგორც MediaObject-ის ქვეკლასები, და მემკვიდრეობით ღებულობს მის მეთოდებს. ისინი აფართოებენ ამ

მეთოდებს საკუთარი ოპერაციებით, რომლებიც მორგებულია სპეციალურ მულტიმედიურ მონაცემთა ტიპებზე, მაგალითად, image-ს დროს height, ან Text-ის დროს length.

ORION სისტემა ან MIM (Multimedia Information Manager)-ით აღნიშნული კლასების კოლექცია წარმოადგენს კიდევ ერთ შრეს MediaObject-სა და Text ან Image ტიპებს შორის, რომლებიც შეიცავს კლასებს „წრფივი მედიაობიექტები“ და „სივრცითი მედიაობიექტები“. პირველი მოიცავს ტექსტებს და აუდიოს, ხოლო მეორე – სურათებს, გრაფიკას და ვიდეოს. მართლაც, ამ მეთოდებს, როგორიცაა length და height, შეუძლია ფაქტობრივად განსაზღვრულ იქნეს ამ კლასებზე და შემდეგ მემკვიდრეობით იქნას გამოყენებული. სხვა მხრივ უფრო მნიშვნელოვნად ჩანს კლასიფიკაცია როგორც „დროზე დამოკიდებული“ და „დროზე დამოუკიდებელი“. ამიტომ გადაწყვეტა ასეთი შუალედური შრის შესახებ ჯერჯერობით არ განიხილება.

კლასების იერარქიის გენერალიზაცია და აგება საშუალებას იძლევა შევძლოთ არა მხოლოდ მულტიმედიური ობიექტების სასარგებლო ჩანერგვა, არამედ აგრეთვე ობიექტებისაც (Entities), რომლებსაც ისინი ასახავს ან აღწერს.

წინა პარაგრაფში ობიექტრელაციური მოდელების დისკუსიამ გვიჩვენა, რომ გარკვეულ შემთხვევებში სხვადასხვა is.presented_in დამოკიდებულება ამოდელირებს და ეს ყველაფერი ძებნის დროს ცხადად უნდა იქნას გათვალისწინებული.

გენერალიზაცია, პირიქით, ნებას იძლევა, რომ ამომრჩეველი, დეპუტატობის_კანდიდატი და საარჩევნო_უბნის_თანამშრომელი ზემნიშვნელობით (Superclass) გავაერთიანოთ, როგორიცაა „პიროვნება“, და მათი ეგზემპლარები სურათებით დავაკავშიროთ.

რომელი ზე-მნიშვნელობა მიესადაგება დასმულ მიზანს დამოკიდებულია იმაზე, თუ რომელი ობიექტებია სურათებზე, ტექსტებში, გრაფიკებზე და ა.შ. უფრო რელევანტური (შესაბამისი)

გამოსაყენებლად. ყველაზე ზოგად შემთხვევაში, შეძლებისდაგვარად, თუ მოცემულია პრესიდან სურათი (ფოტო), საჭიროა ისევ „Object“ კლასზე მიმართვა.

ობიექტორიენტირებულ მოდელს შეუძლია მე-3 ტიპის სქემის (იხ. წინა პარაგრაფი) უკეთესად წარმოდგენა, ვიდრე რელაციურ მოდელს. როგორი მდგომარეობაა სქემის სხვა ტიპებისთვის? კლასის ეგზემპლარები გამოისახება კორტეჟების სახით ატრიბუტების მნიშვნელობებზე, რომელთა განსაზღვრის არეები დადგენილია კლასების აღწერაში.

კლასიკური რელაციური მოდელისგან განსხვავებით მნიშვნელობათა ეს არეები არ უნდა იყოს არჩეული ერთი წინასწარგანსაზღვრული სიმრავლიდან, არამედ შესაძლებელია ასევე იყოს ნებისმიერი თვითგანსაზღვრებადი კლასი.

ასე მაგალითად, employee კლასს შეუძლია თავისი ეგზემპლარებისთვის განსაზღვროს ატრიბუტი (ეგზემპლარის ცვლადი) Portrait, რომელთა მნიშვნელობები შეიძლება იყოს GreyscaleImage კლასის ეგზემპლარები. ეს შეესაბამება სქემის 1-ელ ტიპს რელაციური ბაზისთვის.

ობიექტორიენტირებული სისტემები არ მოითხოვს ნორმალიზაციას თავისი კლასებისა და ეგზემპლარებისთვის, ასე რომ ნებადართულია ატრიბუტები რამდენიმე მნიშვნელობით. აუცილებლობის შემთხვევაში საჭიროა ტიპების კონსტრუქტორის, როგორიცაა list ან set, გამოყენება. შესაბამისად იგი მე-2 ტიპის სქემასთან შედარებით ოდნავ ჭარბია. Portrait ატრიბუტს შეუძლია აგრეთვე მარტივად ჰქონდეს რამდენიმე სურათი.

მრავალი ობიექტორიენტირებული სისტემა არ განსხვავდება სუფთად ობიექტის ატრიბუტებით და კომპონენტებით. ხშირად ატრიბუტთა დახმარებით აგრეგაცია გამოისახება მარტივად. ODMG-მოდელი მკაფიოდ გამოყოფს ატრიბუტებს დამოკიდებულებებისგან და თუ ატრიბუტებს შეუძლია მხოლოდ

მნიშვნელობების მიღება, ობიექტები უკავშირდება ერთმანეთს დამოკიდებულებებით.

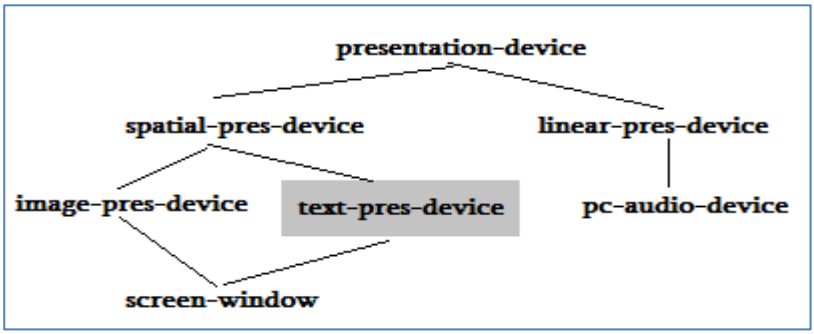
ობიექტორიენტირებული სისტემების სუსტი ადგილია სიმრავლეებზე ორიენტაციის არარსებობა და მასთან დაკავშირებული ძებნა. ეს გასაგები იყო, ამიტომაც ცდილობდნენ ობიექტორიენტირებული ბაზების მართვის სისტემებისთვის მოთხოვნების ენის განსაზღვრას [77].

OMDG-მოდელში არსებობს ამისთვის OQL-ენა (Object Query Language), რომელიც ორიენტირებულია SQL-ზე და ზოგიერთ შემთხვევაში სრულად თავსებადიცაა. მას მოაქვს თან ყველა სასურველი თვისება, მაგრამ არ შეესაბამება ობიექტებზე პირველად წვდომას, რისთვისაც იყო შექმნილი ოო-ბაზების სისტემები: იგულისხმება ირიბი მიმართვა. ვინაიდან OQL ენა ძალზე რთულია, მისი გამოყენება ნაკლებად გვხვდება პროგრამულ პროდუქტებში.

3.3.2. მულტიმედიური ინფორმაციული მენეჯერი (MIM)

ობიექტორიენტირებული მულტიმედიურ მონაცემთა ბაზის კონკრეტული მაგალითი განვიხილოთ ORION სისტემის მაგალითზე, რომელიც ასევე ცნობილია მულტიმედიური ინფორმაციული მენეჯერის (MIM – Multimedia Information Manager) სახელით [77-79]. ესაა ობიექტ-ორიენტირებული კლასების ბიბლიოთეკა. ასეთი ობიექტორიენტირებული მონაცემთა ბაზების სისტემა მომხმარებლისთვის აადვილებს მულტიმედიური ობიექტების სუბკლასებამდე დაყვანას და შემდეგ MIM ბიბლიოთეკის ოპერაციებით მემკვიდრეობითობის მექანიზმის გამოყენებით სარგებლობას [77].

MIM-ის განსაკუთრებული თავისებურება მდგომარეობს იმაში, რომ ყველა მოწყობილობა წარმოიდგინება როგორც ობიექტები ORION-ში, კერძოდ, როგორც შეტანა/გამოტანის, ასევე მეხსიერების მოწყობილობები. გამოტანის მოწყობილობისთვის მომზადებულია კლასების იერარქია, რომელიც 3.2 ნახაზზეა მოცემული.



ნახ.3.2. MIM-ში კლასთა იერარქია გამოსატანი მოწყობილობებისთვის

გრაფის წიბოები ასახავს მემკვიდრეობითობის კავშირს, სადაც სუბკლასები მოთავსებულია კლასების ქვეშ. მონიშნული კლასები text-pres-device არის მაგალითი გაფართოებისა, რომელიც თვით მომხმარებელმა გააკეთა. ასეთი კლასი არ ეკუთვნის MIM-ბიბლიოთეკას.

კლასთა ეგზემპლარები ამ იერარქიაში არაა განაწილებული ცალსახად გამომტანი მოწყობილობებისთვის, არამედ ისინი სპეციფიცირებულია:

- სადაა მოწყობილობაზე ასახული (მაგალითად, ეკრანის რომელ ნაწილში);
- მედიალური ობიექტის რომელი ნაწილი (ამონაჭერი) არის ასახული.

შედეგად შესაძლებელია რამდენიმე ეგზემპლარის მიცემა, რომლებიც ერთი და იმავე ფიზიკურ მოწყობილობას ასახავს. შეიძლება ასევე მათი ინტერპრეტაცია შეზღუდვებით, როგორც „გამოცემის ფორმატი“ მედიური ობიექტებისთვის. მაგალითად, კლასი spatial-pres-device ასე განმარტავს მისი ეგზემპლარის შემდეგ ატრიბუტებს:

upper-left-x,
upper-left-y,
width,
height.

ამ ატრიბუტების მნიშვნელობები შეესაბამება მედიური ობიექტის ამონაჭერს (მაგალითად, რასტრულ სურათს), რომელიც უნდა გამოიცეს სივრცით გამომტან მოწყობილობაზე.

ქვეკლასში screen-window სპეციფიცირდება ატრიბუტები:

win-upper-left,
win-upper-right,
win-width,
win-height.

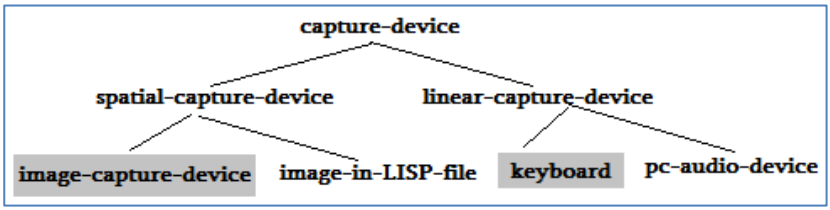
დამატებით ეკრანის ნაწილზე, რომელიც უნდა იქნას გამოყენებული ასახვისთვის.

screen-window -სთვის განსაზღვრულია ასევე მეთოდები, კერძოდ:

present,
capture,
present-pres,

რომლებიც გამოიყენება განსაზღვრული მულტიმედიური ობიექტების გამოსატანად და, აუცილებლობის შემთხვევაში, გამოიძახება წაკითხვის აღსადგენადაც.

ანალოგიურად არის MIM-ში განსაზღვრული კლასთა იერარქია შესატანი მოწყობილობებისთვის (ნახ.3.3). აქ მონიშნული კლასები, image-capture-device და keyboard აღწერს მომხმარებელთა გაფართოებებს.



ნახ.3.3. MIM-ში კლასთა იერარქია შესატანი მოწყობილობებისთვის

აქაც, როგორც წინა შემთხვევაში, კლასის ეგზემპლარები მეტია, ვიდრე მხოლოდ სპეციფიცირებული მოწყობილობები. ისინი მიუთითებს ასევე, თუ მულტიმედიური ობიექტის რომელი ნაწილი განიხილება და როგორაა მოწყობილობა დაკომპლექტებული. შედეგად შესაძლებელია ერთი ფიზიკური მოწყობილობისთვის კვლავ capture-device ტიპის რამდენიმე ეგზემპლარის მიცემა.

spatial-capture-device სუბკლასის ეგზემპლარები მიუთითებს შემდეგ ატრიბუტებს:

upper-left-x,

upper-left-y,

width,

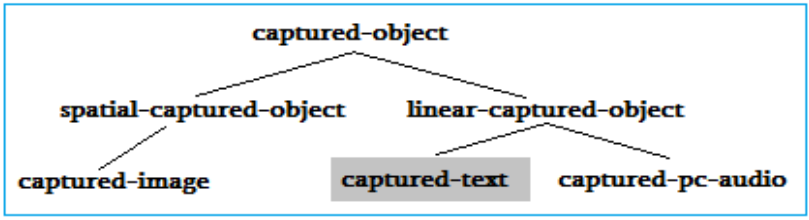
height.

იგი სპეციფიცირებას უკეთებს სივრცითი მულტიმედიური ობიექტის ამონაჭერს, რომელიც მიიღება შემტანი მოწყობილობით ან ჩანაცვლებით. მომხმარებლის მიერ განსაზღვრულ image-capture-device კლასს შეუძლია:

cam-width,
cam-height,
bits-per-pixel

ატრიბუტების და capture მეთოდის წარმოდგენა.

დამახსოვრებული მულტიმედიური ობიექტები ორგანიზებულია ასევე კლასთა იერარქიის სახით. ამ დროს კვლავ განასხვავებენ სივრცით და წრფივ მულტიმედიურ ობიექტებს. რასტრული სურათები და გრაფიკები მიეკუთვნება სივრცითს, ხოლო ტექსტი და აუდიო კი – წრფივს. ქვეკლასები ასახულია 3.4 ნახაზზე.



ნახ.3.4. MIM-ში დამახსოვრებული მედიური ობიექტების კლასთა იერარქია

captured-object კლასებში ყველა ქვეკლასისთვის განსაზღვრულია შემდეგი ატრიბუტები:

storage-object – მიუთითებს storage-device კლასის ერთ ეგზემპლარზე, რომელიც ქვემოთ შემოიტანება.

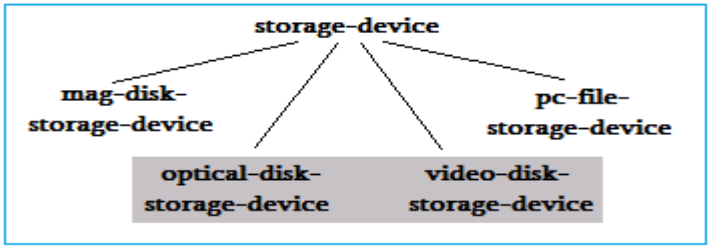
logical-measure – განსაზღვრავს საწყისი მონაცემების ელემენტარულ ერთეულებს მომხმარებლის თვალსაზრისით, რომლებიც არ უნდა ემთხვეოდეს სისტემოტექნიკურ ერთეულებს [ბიტი, ბაიტი ან მეხსიერების სიტყვა (memory word)]. ასეთი ლოგიკური ზომის ერთეულებია, მაგალითად, წამები აუდიოს დროს ან კადრები - ვიდეოს დროს.

დამოკიდებულება ფიზიკურ და ლოგიკურ ზომის ერთეულებს შორის აისახება phys-logic-ratio ატრიბუტში. ეს იქნება მაშინ ბაიტი/წამში აუდიოსთვის ან ბაიტი/კადრი ვიდეოსთვის.

spatial-captured-object სუბკლასი იძლევა ატრიბუტებს width, height და row-major. უკანასკნელი გვიჩვენებს, რომ დამახსოვრება მოხდა სტრიქონულად ან სვეტურად.

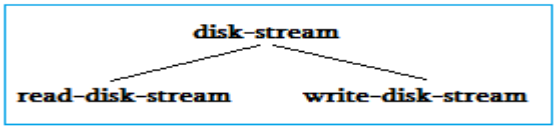
დაბოლოს, არსებობს ატრიბუტი bits-per-pixel, რომელიც ცხადყოფს, რომ საქმე ეხება ტიპურ სარეგისტრაციო მონაცემებს.

დამახსოვრებისთვის არსებობს მოწყობილობები, რომლებიც შესატან და გამოსატან მოწყობილობათა მსგავსად ნაწილობრივ გამოიყენება და storage-device კლასის ეგზემპლარების საშუალებით წარმოიდგინება (ნახ.3.5).



ნახ.3.5. MIM-ში შენახვის მოწყობილობების კლასთა იერარქია

დაბოლოს, კლასებში არსებობს კიდევ ორგანიზებული მონაცემთა ობიექტები, რომლებიც ასახავს ცალკეულ წაკითხვის ან ჩაწერის პროცესებს (ნახ.3.6).



ნახ.3.6. MIM-ში წაკითხვის და ჩაწერის პროცესების კლასთა იერარქია

მათი წარმოება ხდება დინამიკურად და შეტანის ან გამოტანის დამთავრების შემდეგ ისევ იშლება. ამავდროულად, disk-stream შეიცავს ორივე შემთხვევისთვის საჭირო storage-object ატრიბუტს, რომელიც მიმართავს storage-device კლასის წასაკითხ ან ხელახლაჩასაწერ ეგზემპლარს. read-disk-stream-ში არსებობს დამატებთ read-block-list ატრიბუტი, რომელიც აღწერს პოზიცირების მარკირებას და აიდენტიფიცირებს მულტიმედიური ობიექტის მომდევნო წასაკითხ ბლოკს. მსგავს როლს ასრულებს write-block-list ატრიბუტი write-disk-stream -ში.

ამჯერად ილუსტრირებული გვაქვს აღნიშნული კლასების ურთიერთმოქმედება და მეთოდების ჩადგმული გამოძახება რასტრული სურათის გამოცემის მაგალითზე.

დავუშვათ, რომ არსებობს car კლასი (ავტომობილი), რომელიც თავისი ეგზემპლარებისთვის განსაზღვრავს ატრიბუტებს image (სურათი) ტიპით captured-image და output_ device (გამოსატანი მოწყობილობა) ტიპით image-pres-device.

ავტომანქანის აღწერას ეკუთვნის აგრეთვე გამოსატანი მოწყობილობა, რომელზეც მანქანის სურათი კარგად უნდა აისახოს. ამის გარდა ეს კლასი შეიცავს ასევე მეთოდს show_image (სურათის ჩვენება), რომელიც ახორციელებს დამახსოვრებული სურათის გამოცემას წინასწარგანსაზღვრულ გამოსატან მოწყობილობაზე.

ამის მისაღწევად, საჭიროა show-image მეთოდმა გააგზავნოს present შეტყობინება ობიექტისკენ, რომელიც წარმოადგენს გამოსატან მოწყობილობას, და მას ერთ პარამეტრში დაუსახელოს გამოსატანი სურათი.

თუ გავითვალისწინებთ, რომ ასეთი გადაცემის ან გამოძახების ბრძანებები LISP-ენის სახის მეთოდოლოგიის სინტაქსით აიგება, მიმღები ობიექტი და პარამეტრი

სპეციფიცირდება, მაშინ show-image-ის შესაბამისი ბრძანება შეიძლება ასე ჩაიწეროს:

(present output-device Image)

output-device ატრიბუტით იდენტიფიცირებული image-pres-device კლასის ეგზემპლარი ასრულებს ამის შემდეგ მის present მეთოდს.

მისი ატრიბუტები upper-lef-x, upper-lef-y, width და height უთითებს, თუ სურათის რომელი ნაწილი (ამონაჭერი) იხილება. ეს მართკუთხა ამონაჭერი გაითვლება წრფივ კოორდინატებში. ეს შესაძლებელია მხოლოდ captured-image ეგზემპლარზე წვდომის საშუალებით, რომელიც იდენტიფიცირებულია image პარამეტრით, რადგან აქ row-major ატრიბუტი მხოლოდ გვეუბნება, რომ რეგისტრაციის მონაცემები ხელმისაწვდომია.

ამჯერად შენახული სურათი წასაკითხად იხსნება:

(open-for-read Image [start-offset])

Image-ს საშუალებით დასახელებული capture-image ეგზემპლარი ასრულებს open-for-read მეთოდს. ამ დროს იგი აწარმოებს ახალ read-disk-stream ეგზემპლარს, რომლის სახელს (ობიექტის იდენტიფიკატორს) იგი image-pres-device-ს უკან უგზავნის. თუ შესაძლებელია, სპეციფიკაცია start-offset ემსახურება read-block-list -ის ინიციალიზაციას. გამოსატანი მოწყობილობა აგზავნის ამჯერად (present-ის შესრულების გაგრძელება) ამ ნაკადისთვის (straem) შეტყობინებას წაკითხვის შესახებ:

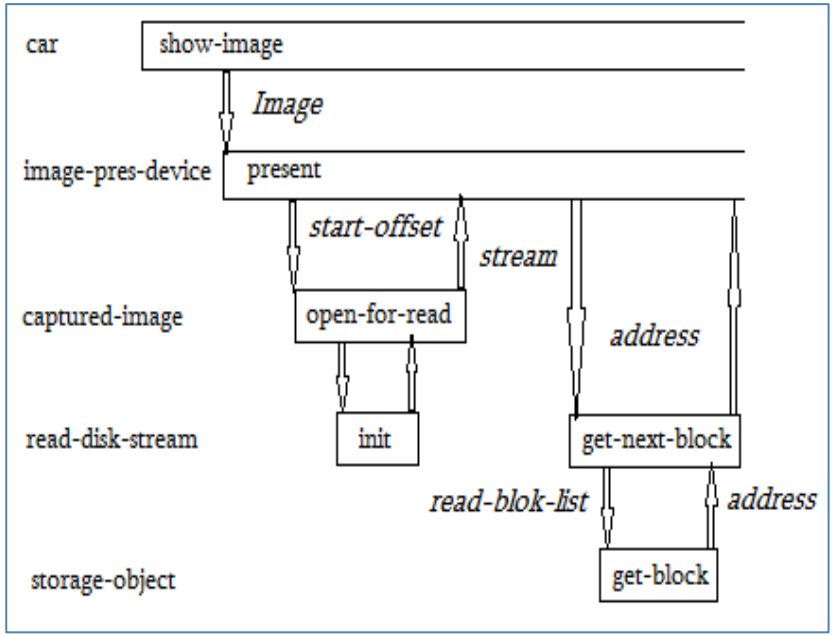
(get-next-block read-disk-stream)

შემდეგ ნაკადი (straem) თვითონ აგზავნის ისევ:

(get-block storage-object read-block-list)

ამ გამოძახებით მიიღება ბუფერის მისამართი, სადაც ინახება სასურველი ბლოკი. შემდეგ ნაკადი ზრდის მაჩვენებელს და აწვდის

image-pres-device–ს უკან ბუფერის მისამართს, როგორც get-next-device მეთოდის შედეგს. 3.7 ნახაზი ასახავს ზემოაღწერილ პროცესს გრაფიკულად.



ნახ.3.7. MIM-მეთოდების ჩადგმული გამოძახება სურათის გამოცემის დროს

მარცხენა მხარეს შემოტანილია კლასის სახელები, რომელთაგან თითოეული ზუსტად ერთ ეგზემპლარში მონაწილეობს. მართკუთხედები აღნიშნავს მეთოდის შესრულებას. ისინი დახრილი (italic) სიმბოლოებით აღწერილი პარამეტრებით გამოიძახება და აბრუნებს ასევე კურსივით აღწერილ მნიშვნელობებს.

შემდეგი მსვლელობისას გამომტანი მოწყობილობა გადასცემს წაკითხული ბლოკის შინაარსს ტექნიკურ

უზრუნველყოფას და აცნობებს ნაკადს, რომ ბუფერის სახელები კვლავ ხელმისაწვდომია:

(free-block read-disk-stream).

სურათის ყველა ბლოკის წაკითხვის და გათავისუფლების შემდეგ (close-read read-disk-stream)-ით დაიხურება წაკითხვის პროცესი. ნაკადი შეიძლება კვლავ წაიშალოს.

captured-object კლასი უზრუნველყოფს თავისი ეგზემპლარებისთვის კიდევ სხვა მეთოდებს. make-captured-object-version-ის საშუალებით იწარმოება შენახული მულტიმედია ობიექტის ახალი ვერსია და, ირიბად კი – ასევე დაკავებული მეხსიერების მოწყობილობის ახალი ვერსია (storage-device). ძველი და ახალი ვერსიები თავიდან იკავებს დისკზე ერთი და იმავე ბლოკებს.

დასასრულ, არსებებოს კიდევ delete-captured-object და delete-part-of-captured-object მეთოდები. უკანასკნელი ელოდება პარამეტრებს start-offset და delete-count, რომლებიც მიუთითებს, თუ რომელი ბაიტ-პოზიციიდან და რამდენი ბაიტი უნდა იქნას წაშლილი.

ეს ოპერაცია მოითხოვს სიფრთხილეს, ვინაიდან იგი იმავდროულად არ სრულდება სარეგისტრაციო მონაცემების ცვლილებით. ასეთი ოპერაცია არაა გათვლილი სისტემის საბოლოო მომხმარებელზე.

ამგვარად, ORION სისტემა MIM-თან ერთად ასახავს ყველაზე სრულყოფილ წინადადებას დღემდე არსებულ მულტიმედიურ მონაცემთა ბაზების მართვის სისტემებს შორის. იგი მომხმარებელს სთავაზობს დიდ მოქნილობას, რათა არსებული კლასების იერარქია საკუთარი სუბკლასების სპეციალიზებით გაფართოვდეს, რაც დამატებითი დამუშავების და წვდომის ოპერაციების გამოყენების შესაძლებლობას იძლევა.

ამგვარად, ობიექტრელაციური და ობიექტორიენტირებული მონაცემთა ბაზების მართვის სისტემების შედარება გვიჩვენებს, რომ ზოგადად არსებობს ერთიანობა მათში ახალი ტიპების განსაზღვრისა და მართვისათვის ან ობიექტორიენტირებული სისტემის კონტექსტში (მაგალითად, ORION), ან რელაციურ სისტემაში (მაფალითად, SQL / MM).

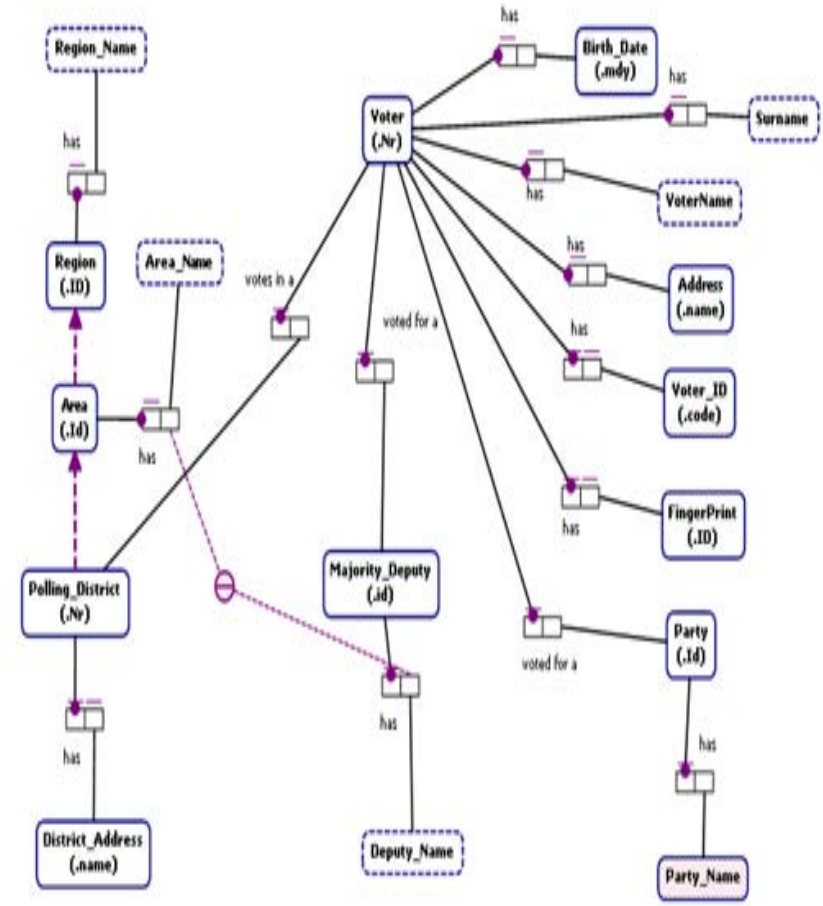
3.4. მულტიმედიური რელაციური ბაზის ოპტიმალური სტრუქტურის დაპროექტება

ელექტრონული საარჩევნო სისტემის მონაცემთა რელაციური ბაზის კონცეპტუალური სქემის დასაპროექტებლად ჩვენ გამოვიყენეთ ობიექტროლური მოდელირების მეთოდი და ინსტრუმენტული საშუალება (იხ. თავი 2).

ავტომატიზებული პროცესი მომხმარებლის მიერ სემანტიკური ფაქტების აღწერით იწყებოდა, რომლის საფუძველზე ფორმირდებოდა კონცეპტუალური სქემა (1-ელი დონე) ORM დიაგრამის სახით (ნახ.3.8) ობიექტებით, პრედიკატებით (კავშირები ობიექტებს შორის), ატრიბუტებით და მათი მნიშვნელობებით.

ვინაიდან საპრობლემო სფეროს აღწერა სისტემური ანალიტიკოსის ან საბოლოო მომხმარებლის მიერ ხდება (ან ორივეს თანამშრომლობით), სემანტიკური ფაქტების სიმრავლე, შემდეგ ORM დიაგრამა და ბოლოს, ERM სქემა შეიძლება იყოს განსხვავებული.

მიიღება ეკვივალენტური კონცეპტუალური სქემები. რომელია მათ შორის ოპტიმალური ან უკეთესი სისტემის მონაცემთა ბაზის საბოლოო რეალიზაციისათვის?



ნახ.3.8. ORM მოდელი

შესაძლებელია თუ არა დაპროექტების წინა სტადიებზე განისაზღვროს უკეთესი ვარიანტი და გამოირიცხოს არაპერსპექტიული ვარიანტები? როგორ შევაფასოთ მოსალოდნელი შედეგი წინასწარ?

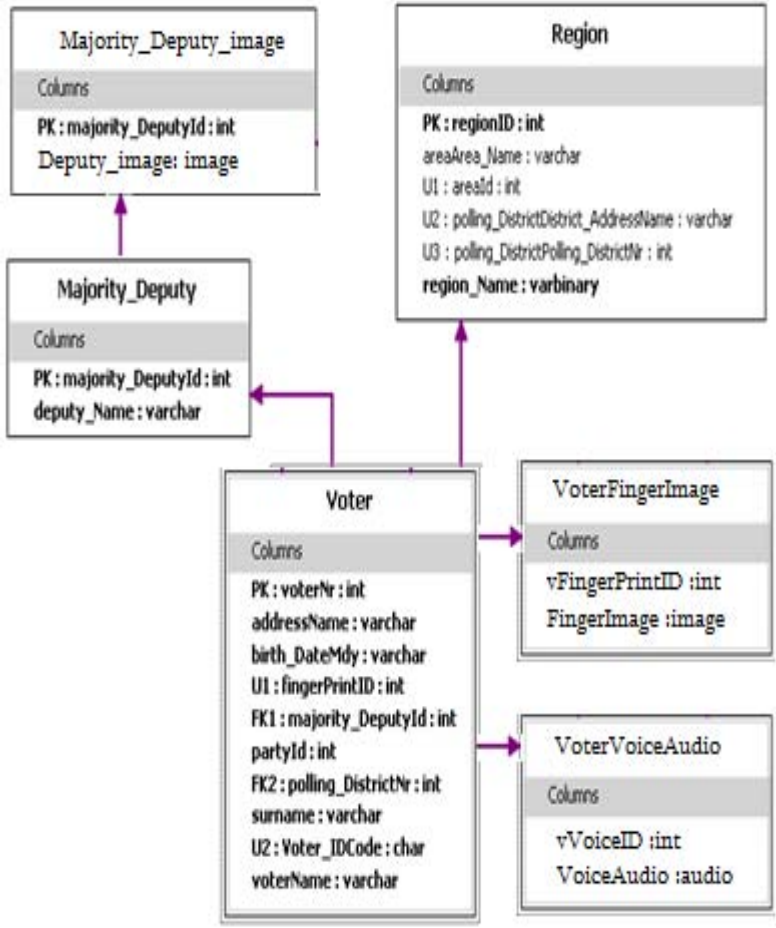
ამგვარად, წინამდებარე პარაგრაფში გვინდა წარმოვადგინოთ ამ საკითხის გადაჭრის გზა, მიდგომა და ფორმალიზებული ალგორითმული სქემები.

3.9 და 3.10 ნახაზებზე ნაჩვენებია ორი ეკვივალენტური კონცეპტუალური (მე-2 დონე) ERM სქემა. ისინი განსხვავდება ცხრილების (Tables) რაოდენობით, რომლებიც ავტომატიზებული პროცედურების ბოლოს, მაგალითად, SQL Sever-ის DDL-ფაილებად გარდაიქმნება ფიზიკური რეალიზაციის მიზნით.

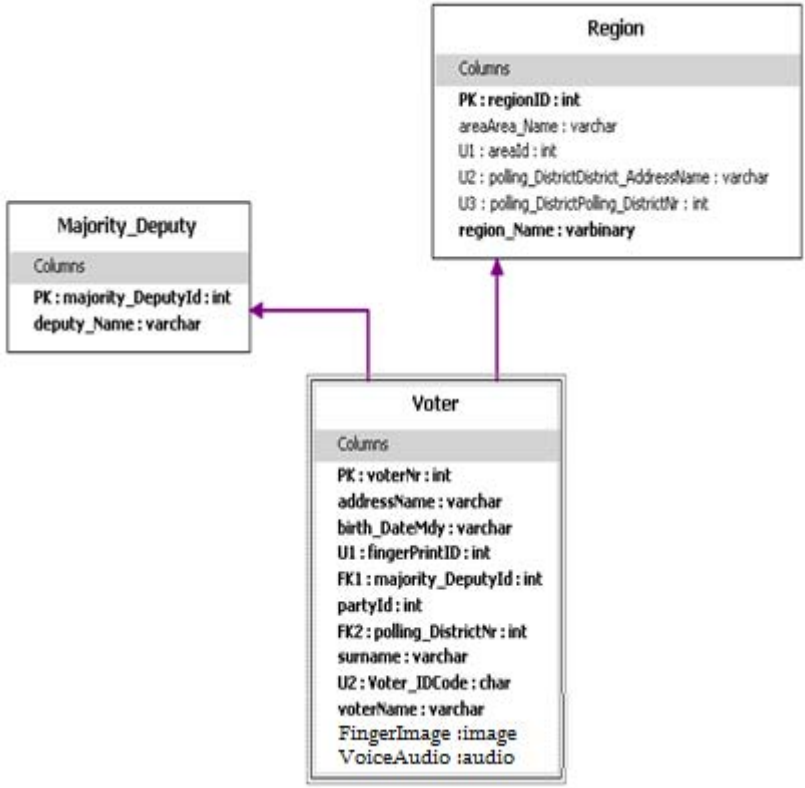
ER-დიაგრამების შედარება გვიჩვენებს, რომ ORM-მოდელის შეცვლამ გაამარტივა ER-მოდელი, კერძოდ, ექვსი ცხრილის ნაცვლად მივიღეთ სამი. ე.ი. ოპტიმიზაცია, ამ შემთხვევაში, არის ობიექტის ER მოდელის აგების პროცესში ცხრილების რაოდენობის მინიმიზაცია, როცა სემანტიკური მთლიანობა არ ირღვევა (!), ანუ აღმოიფხვრება გარკვეული ინფორმაციული სიჭარბე [73].

ER-მოდელის ცხრილების ოპტიმალური შემადგენლობის განსაზღვრის ამოცანა კავშირშია რელაციური ბაზის სტრუქტურის ნორმალიზაციის საკითხთან [69]. ამ ამოცანის გადაწყვეტა და მისი ალგორითმის შემუშავება მოცემულია ქვემოთ.

დავუშვათ, მოცემულია საპრობლემო სფეროს აღწერის ატრიბუტთა $U = \bigcup_i U_i$ სიმრავლე. მონაცემთა ბაზის $F^0(\overline{A})$ საწყისი სქემა შეიძლება გამოვსახოთ უნივერსალური დამოკიდებულების სახით: $\overline{S}^0 = \{\overline{R} = < U, P >\}$, სადაც $\overline{R}$ დამოკიდებულებათა უნივერსუმია, ხოლო P სემანტიკურ შეზღუდვათა კლასები ან ფუნქციონალურ დამოკიდებულებათა (ფდ) ერთობლიობა [59, 66-68].



ნახ.3.9. ER-მოდელის 1-ელი ვარიანტი
ექვსი ცხრილით



ნახ.3.10. ER-მოდელის მე-2 ვარიანტი
სამი ცხრილით

ამოცანის მიზანია $\overline{S}^0$ სქემის ეკვივალენტური $\overline{S}$ სქემის კონსტრუირება $\overline{S} = \{R_i = < U_i, P_i >\}$, სადაც R_i არის $\overline{R}$ -ის პროექცია, ხოლო P_i ეთანადება ფდ-ის განსაზღვრულ კლასს, მაგალითად, ფდ, სრული-ფდ (სფდ), ტრანზიტული-ფდ (ტფდ), ფსევდოტრანზიტული-ფდ (ფტფდ), მრავალსახა (მსდ), ზოგადი არაფუნქციონალური (ზად).

ნორმალურ ფორმათა (ნფ) თეორიის გამოყენებით ხორციელდება უნივერსუმის მიმდევრობითი დეკომპოზიცია შემდეგი სქემით:

ანფ -> 1ნფ -> 2ნფ -> 3ნფ -> 4ნფ -> 5ნფ -> . ? . -> ბნფ ,

სადაც **ანფ** – არანორმალიზებული ფორმაა, **1ნფ** – პირველი ნფ, . . . , **ბნფ** – ბინარული ნფ.

ძირითადი სქემის განშტოებაში შეიძლება განვიხილოთ ბოის-კოდის ნფ (3ნფ-დან), პირველი რიგის იერარქიული დეკომპოზიცია (4ნფ-დან), ურთიერთდამოკიდებულებანი (5ნფ-დან) და ა.შ. ნიშანი „?" მიუთითებს იმაზე, რომ დამოკიდებულებათა თვისებების კვლევა გრძელდება, მათი შემდგომი ოპტიმიზაციის მიზნით.

დღეისათვის მრავლად არსებობს სხვა სახის **ნფ**-ებიც, მაგრამ ნორმალიზაციის კლასიკურ თეორიაში ისინი ნაკლებადაა ასახული [59,66].

დამოკიდებულებათა დეკომპოზიციის დროს ბნფ-ები მიიღება სქემის დაპროექტების ცალკეულ ეტაპზე.

არადეკომპონირებადი დამოკიდებულებისთვის კი საჭიროა ხელოვნურად ფიქტიური ატრიბუტის (ნატურალურ რიცხვთა სასრული სიმრავლე) შემოტანა.

ინფორმაციის სემანტიკური მთლიანობის უზრუნველყოფა დეკომპონირებულ დამოკიდებულებათა შორის ხორციელდება ინდექსური კავშირების დუბლირების საშუალებით, ე.ი. შემოტანილია გარკვეული სიჭარბე.

რაც მაღალია ნფ-ის რიგი, მით ნაკლებია ინფორმაციული და მეტია ინდექსური სიჭარბე:

1nf->2nf->. . . ->bnf, 1nf -> 2nf ->. . . ->bnf,
V1-inf>=V2-inf>=...>=Vb_inf, V1-ind<=V2-ind<=...<=Vb-ind.

აქედან გამომდინარე, დაისვა ოპტიმალური სიჭარბის განსაზღვრის კომპრომისული ამოცანა განახლების დროის მინიმიზაციის მოთხოვნით.

ამოცანის გადაწყვეტის შედეგად შესაძლებელი გახდა მონაცემთა ბაზის სქემის დამოკიდებულებათა ოპტიმალური ნფ-ების დადგენა, რის შემდეგაც ავტომატიზებულად დაპროექტდება ნორმალურ ფორმათა სტრუქტურები.

დავუშვათ, რომ მოცემულია სემანტიკურად თავსებადი ფუნქციონალურ დამოკიდებულებათა სიმრავლე (ფდ):

$$\begin{cases} R_1(k_1, k_2, \ldots k_{n1}, A_1, A_2, \ldots, A_{a1}) \\ R_2(k_1, k_2, \ldots k_{n2}, B_1, B_2, \ldots, B_{a2}) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ R_l(k_1, k_2, \ldots k_{nl}, Z_1, Z_2, \ldots, Z_{al}) \end{cases} \quad (3.1)$$

სადაც k ატრიბუტების გასაღებური, ხოლო A-:-Z არაგასაღებური ნაწილებია.

სემანტიკური მთლიანობა შედეგია იმ ფაქტისა, რომ სიმრავლე მიღებულია ერთი უნივერსუმის დეკომპოზიციით. თუ ჩავთვლით, რომ

$$k_1, k_2, \ldots k_{n1} \supseteq k_1, k_2, \ldots k_{n2} \supseteq k_1, k_2, \ldots k_{nl},$$

(1) სისტემის კომპოზიციით, მაშინ შესაძლებელია ერთი შედარებით დაბალი ნფ-ის მიღება:

$$R(k_1 \ldots k_{n1}, A_1 \ldots A_{a1}, B_1 \ldots B_{a2}, \ldots Z_1 \ldots Z_{al}) \quad (3.2)$$

დავუშვათ აგრეთვე, რომ წინასწარ ცნობილია R_i-ის ცვლილების რაოდენობა μ_i დროის განსაზღვრულ ინტერვალში და მართებულია შემდეგი მოწესრიგება:

$$\mu_1 \geq \mu_2 \geq \ldots \geq \mu_l$$

(1) და (2) გამოსახულებებისთვის განახლებათა მოცულობები შესაბამისად გამოითვლება შემდეგნაირად:

$$Q_{dec} = \sum_{i=1}^{l} \mu_i * (n_i + a_i) \text{ da } Q_{com} = \mu_1 * (n_1 + \sum_{j=1}^{l} (a_j - r))$$

სადაც r ატრიბუტების ის რაოდენობაა, რომლითაც სრულდება შეერთების („Join“) ოპერაცია. შემდგომში შეიძლება მისი იგნორირება.

თუ დავუშვებთ, რომ (1) და (2) ნფ-ებს შორის არსებობს შუალედური ნფ, მაშინ მისთვის განახლებათა მოცულობა შეადგენს:

$$Q = \sum_{j=1}^{s} \mu_j * (n_j + \sum_{k=1}^{l} a_k)$$

სადაც S ფდ-ების რაოდენობაა შუალედურ ნფ-ში. მართებულია შემდეგი უტოლობა:

$$\mu_1 * (n_1 + \sum_{k=1}^{l} a_k) \geq \ldots \geq \sum_{j=1}^{s} \mu_j * (n_j + \sum_{k=1}^{l} a_k) \geq \ldots \geq \sum_{i=1}^{s} \mu_i * (n_i + a_i) \quad ,(3.3)$$

სადაც უტოლობის მარცხენა მხარე ეთანადება (i-1)ნფ-ს, ნაპირა მარჯვენა მხარე – (i+1)ნფ-ს, ხოლო ცენტრალური – i ნფ-ს, სადაც $i >= 4$.

გავაანალიზოთ დეტალურად ორი მოსაზღვრე ნფ, მაგალითად, i და i+1. (3)_დან შეიძლება მივიღოთ:

$$\sum_{j=1}^{s} \mu_j n_j + \sum_{j=1}^{s} \sum_{k=1}^{l} \mu_j a_k \geq \sum_{i=1}^{l} \mu_i n_i + \sum_{i=1}^{l} \mu_i a_i \quad , \qquad (3.4)$$

აქედან მართებულია შემდეგი გამოსახულებანი:

$$\sum_{i=1}^{l} \mu_i n_i - \sum_{j=1}^{s} \mu_j n_j = \sum_{i=s+1}^{l} \mu_i n_i \quad , \qquad (3.5)$$

$$\sum_{j=1}^{s}\sum_{k=1}^{l}\mu_j a_k - \sum_{i=1}^{l}\mu_i a_i = \sum_{j=1}^{s}\sum_{k=1,\, j\neq k}^{l}\mu_j a_k - \sum_{i=s+1}^{l}\mu_i a_i \ , \qquad (3.6)$$

თუ (5) და (6) ტოლობათა მარჯვენა ნაწილებს ჩავსვამთ (4)-ში, მივიღებთ:

$$\sum_{j=1}^{s}\sum_{k=1,\, j\neq k}^{l}\mu_j a_k \geq \sum_{i=s+1}^{l}\mu_i n_i + \sum_{i=s+1}^{l}\mu_i a_i \ , \qquad (3.7)$$

უტოლობის ორივე მხარე გავყოთ $\sum_{i=s+1}^{l}\mu_i a_i$ - ზე, გვექნება:

$$\frac{\sum_{j=1,k=1}^{s}\sum_{j\neq k}^{l}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i} \geq \frac{\sum_{i=s+1}^{l}\mu_i n_i}{\sum_{i=s+1}^{l}\mu_i a_i} + \frac{\sum_{i=s+1}^{l}\mu_i a_i}{\sum_{i=s+1}^{l}\mu_i a_i} \ . \qquad (3.8)$$

ვინაიდან $[1:l]=[1:s]\cup[s+1:l]$, ამიტომ :

$$\frac{\sum_{j=1}^{s}\sum_{k=1,\, j\neq k}^{l}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i} = \frac{\sum_{j=1}^{s}\sum_{k=1,\, j\neq k}^{s}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i} + \frac{\sum_{j=1}^{s}\mu_j \sum_{k=s+1}^{l} a_k}{\sum_{i=s+1}^{l}\mu_i \sum_{i=s+1}^{l} a_i}$$

ამგვარად, (8)-დან მივიღებთ ვედეკინდ-სურგულაძის მოდელს [69]:

$$\frac{\sum_{j=1}^{s}\sum_{k=1,\, j\neq k}^{l}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i} + \frac{\sum_{j=1}^{s}\mu_j}{\sum_{i=s+1}^{l}\mu_i} \geq \sum_{i=s+1}^{l}\frac{n_i}{a_i} + 1 \ , \qquad (3.9)$$

სადაც $l \geq 2, s \geq 1$ და $l > s$.

პრაქტიკაში ხშირად გამოიყენება შემთხვევა, როცა *l*=2 და *s*=1, მაშინ (9) ღებულობს შემდეგ სახეს:

$$\frac{\mu_1}{\mu_2} \geq \frac{n_2}{a_2} + 1 \qquad (3.10)$$

რაც, როგორც ცნობილია, არის ვონგ-ვედეკინდის მოდელი [88].

იგი არის (3.9) გამოსახულების კერძო შემთხვევა. (3.10)-ის გამოყენების დიაპაზონია მე-3ნფ-მდე, ხოლო (3.9)-ისა, მთელი დიაპაზონი ნფ-ებისა, ამგვარად იგი უნივერსალურია.

ახლა გამოვიკვლიოთ შემთხვევა, როდესაც კორტეჟის არაგასაღებური ატრიბუტების მნიშვნელობათა ცვლილების სიხშირე მაღალია, ხოლო გასაღებურისა _ დაბალი. დავუშვათ, *l*=2, *s*=1 და მოცემულია რელაციათა სქემები:

$$\begin{cases} R_1(k_1, k_2, \ldots k_{n1}, A_1, A_2, \ldots, A_{a1}) \\ R_2(k_1, k_2, \ldots k_{n2}, B_1, B_2, \ldots, B_{a2}) \\ \\ R_{12}(k_1, k_2, \ldots k_{n1}, A_1, A_2, \ldots, A_{a1}, B_1, B_2, \ldots, B_{a2}) \end{cases},$$

რომლებშიც მართებულია შემდეგი პირობები:

$$k_1, \ldots k_n \supseteq k_1, \ldots k_{n2} \text{ da } \mu_1 > \mu_2 \ . \qquad (3.11)$$

(3.9)-დან გამომდინარე, მოცემული R_1, R_2 და R_{12} სქემებისთვის, გასაღებურ ატრიბუტთა მნიშვნელობების ცვლილების მაღალი სიხშირის დროს მიზანშეწონილია R_1 da R_2 დამოკიდებულებათა კომპოზიცია R_{12}-ში, თუ სრულდება პირობა:

$$\mu_1(n_1 + a_1) + \mu_2(n_2 + a_2) > \mu_1(n_1 + a_1 + a_2).$$

აქედან გამომდინარეობს, რომ:

$$\frac{n_2}{a_2} > \frac{\mu_1}{\mu_2} - 1.$$

თუ განიხილება არაგასაღებური ატრიბუტების მნიშვნელობათა ცვლილება გასაღებური ატრიბუტების ცვლილების გარეშე, მაშინ მართებულია შემდეგი გამოსახულება:

$$\mu_1 a_1 + \mu_2 a_2 > \mu_1 (a_1 + a_2).$$

აქედან გამომდინარეობს, რომ:

$$\mu_2 > \mu_1.$$

რაც ეწინააღმდეგება (3.11)-ს.

ამგვარად, დამოკიდებულებათა სქემები, რომლებისთვისაც დომინირებადია არაგასაღებურ ატრიბუტთა ნაწილის ცვლილება, მიზანშეწონილია გამოისახოს მაღალი რიგის ნფ-ებით.

- სქემის გარდასახვა კონცეპტუალურ დონეზე შეიძლება გამოყენებულ იყოს იმისათვის, რომ უფრო ნათელი გახდეს კონცეპტუალური მოდელი ან გაუმჯობესდეს მონაცემთა ბაზის აპლიკაციის ხარისხი;

- მონაცემთა ბაზის დამოკიდებულებანი უნდა წარმოდგენილი იქნეს სხვადასხვა რიგის ნორმალური ფორმებით (3ნფ-:-ბნფ), განახლების სიხშირესა და კავშირების ტიპებზე დამოკიდებულებით მოცემულ კონტექსტში;

- მოცემული μ-თვის შეიძლება (3.9) გამოსახულებით დადგინდეს მოსახერხებელი (ოპტიმალური) ნფ-ები;

- თუ დამოკიდებულებათა გასაღებური ატრიბუტების ცვლილებების სიხშირე მაღალია, მაშინ მათთვის სასურველია დაბალი რიგის ნფ-ების გამოყენება, ხოლო თუ არაგასაღებურ ატრიბუტთა ცვლილების სიხშირე დომინირებადია, მაშინ – შედარებით მაღალი რიგის ნფ-ებისა.

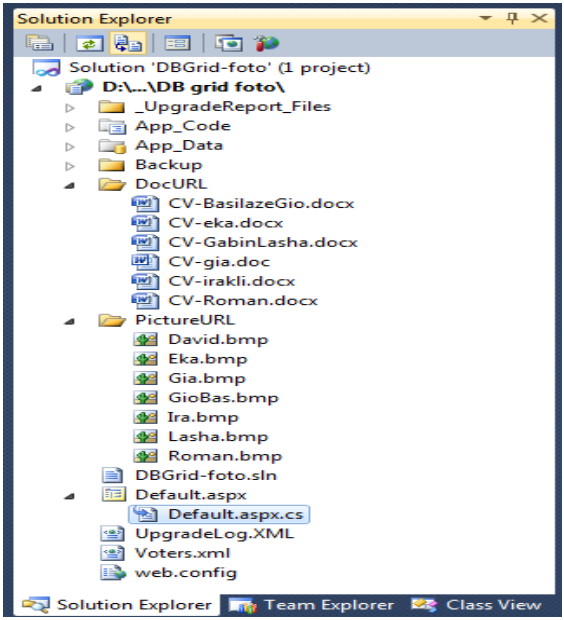
IV თავი

სისტემის პროგრამული რეალიზაცია და ექსპლუატაციის მხარდაჭერა

4.1. ელექტრონული საარჩევნო სისტემის კომპონენტების დამუშავება Visual Studio.NET Framework 4.0 გარემოში

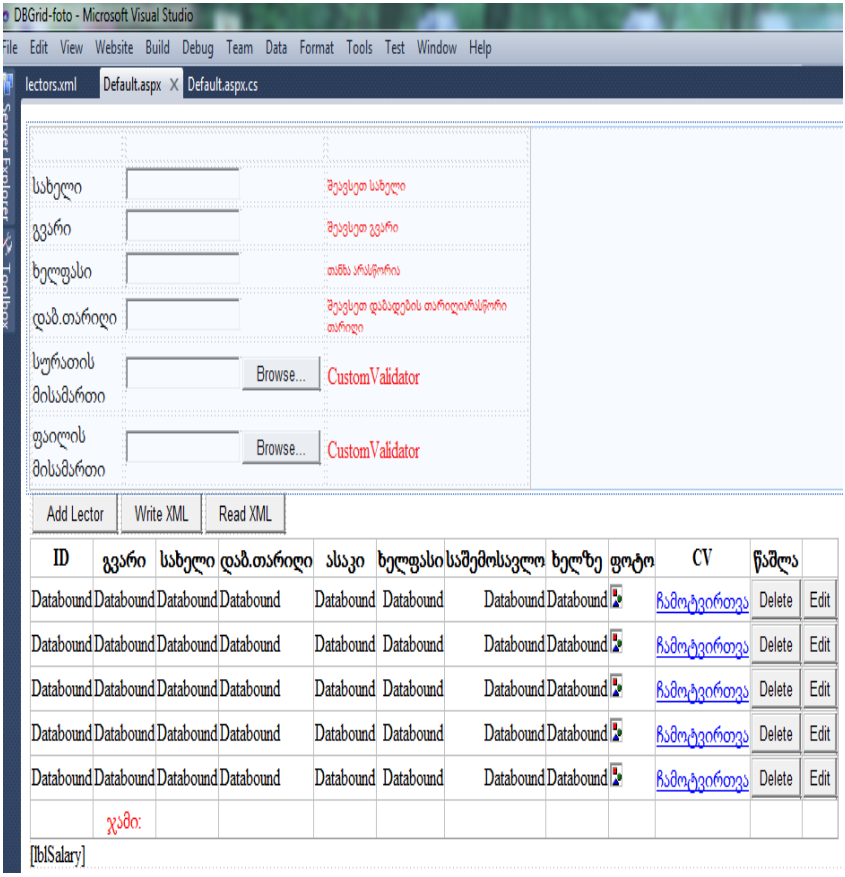
4.1.1.DBGrid-foto აპლიკაცია

სადემონსტრაციო პროგრამული პაკეტის ფრაგმენტი მოტანილია 4.1 ნახაზზე, რომელიც Ms Visual Studio .NET Framework 4.0 ვერსიისთვისაა შესრულებული. Solution Explorer-ში ჩანს პროგრამული პროექტის სტრუქტურა, რომელიც შედგება სისტემური, ტექსტური, გრაფიკული და სხვა ტიპის ფაილებისგან (C#, HTML, ASP, XML, XSS, MDF და სხვ.).



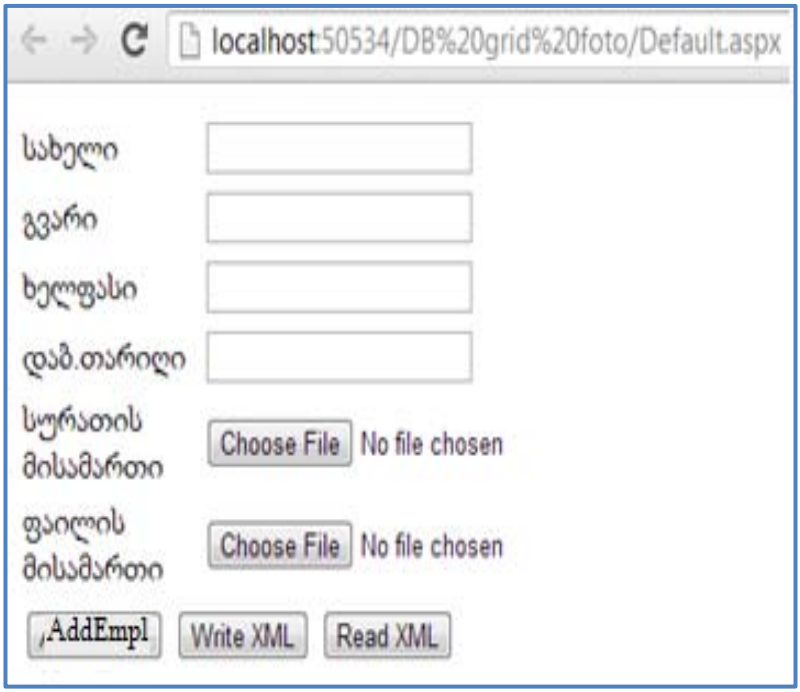
ნახ.4.1. პროგრამული პროექტის სტრუქტურა

პროგრამული სისტემის დიზაინი (ნახ.4.2) შესრულებულია .aspx-ტიპის ფაილით და html პროგრამით აღიწერება (პარაგრაფი 4.1.2-ის ლისტინგი 4.2).

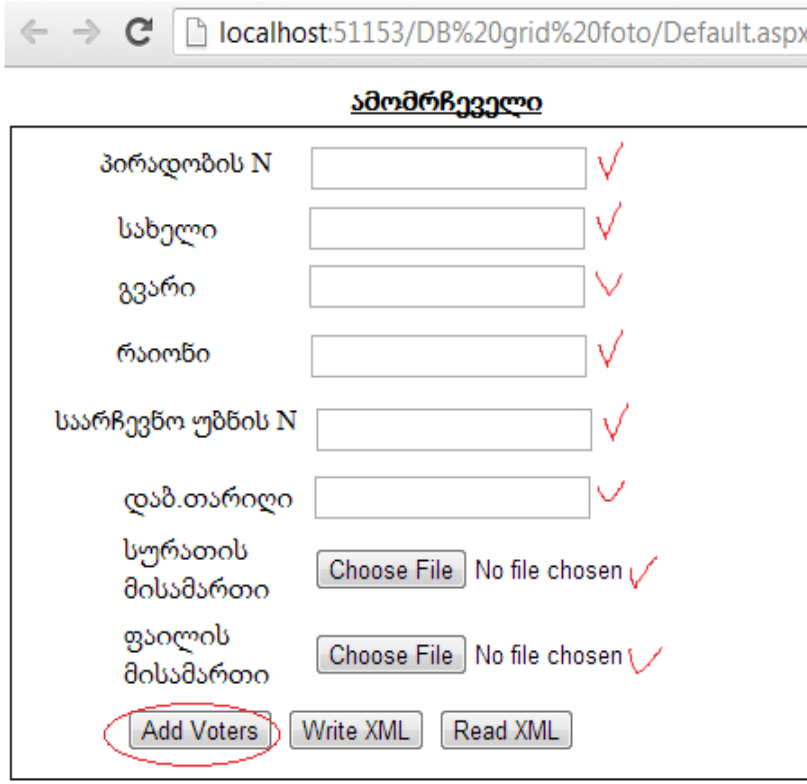


ნახ.4.2. სისტემის სამომხმარებლო ინტერფეისი

4.3–4.4 ნახაზებზე ნაჩვენებია ინტერნეტ ბრაუზერში მომუშავე სისტემის ინტერფეისი – თანამშრომელთა რეგისტრაცია, ამომრჩეველთა რეგისტრაცია.

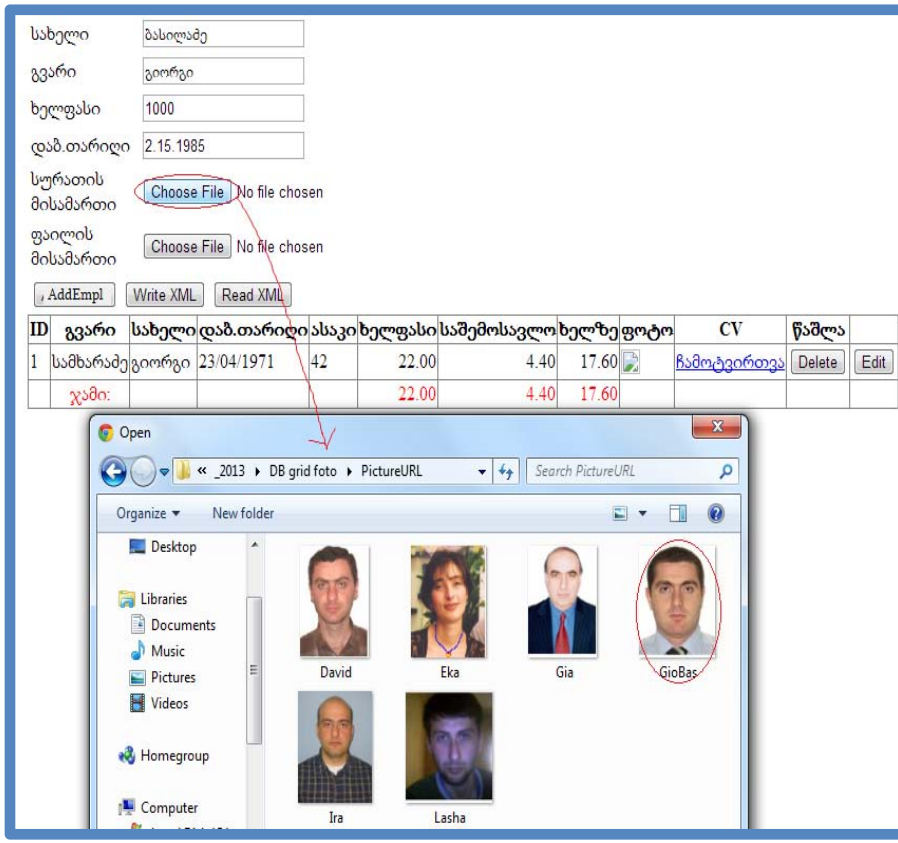


ნახ.4.3. თანამშრომელთა (Employees) რეგისტრაცია

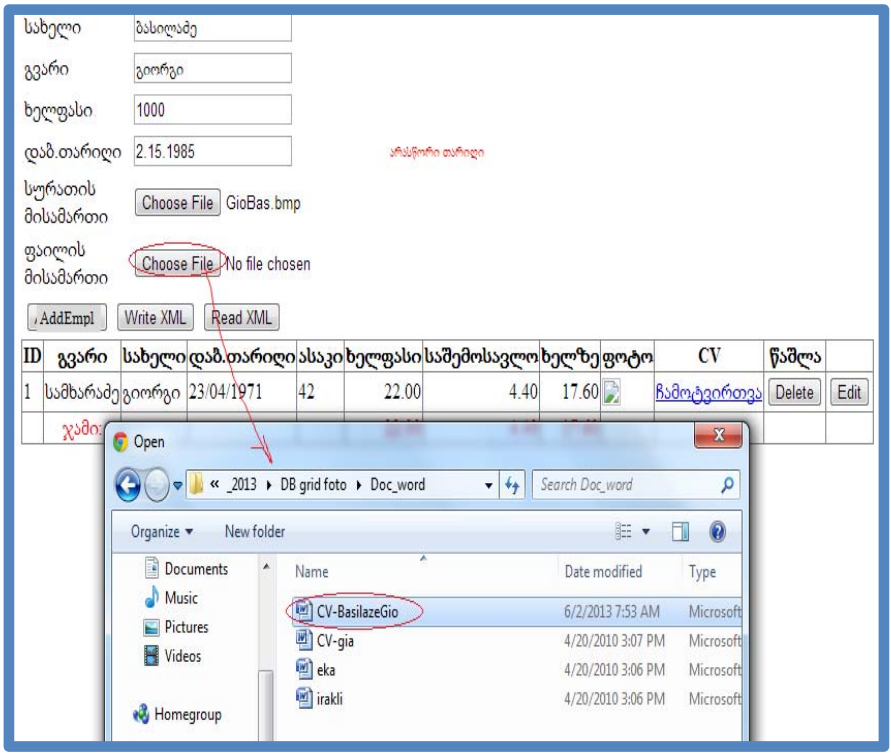


ნახ.4.4. ამომრჩეველთა (Voters) რეგისტრაცია

გრაფიკული და ტექსტური ინფორმაციის შესატანად გამოყენებულია შესაბამისი დიალოგური პროცედურები (ნახ. 4.5, 4.6).

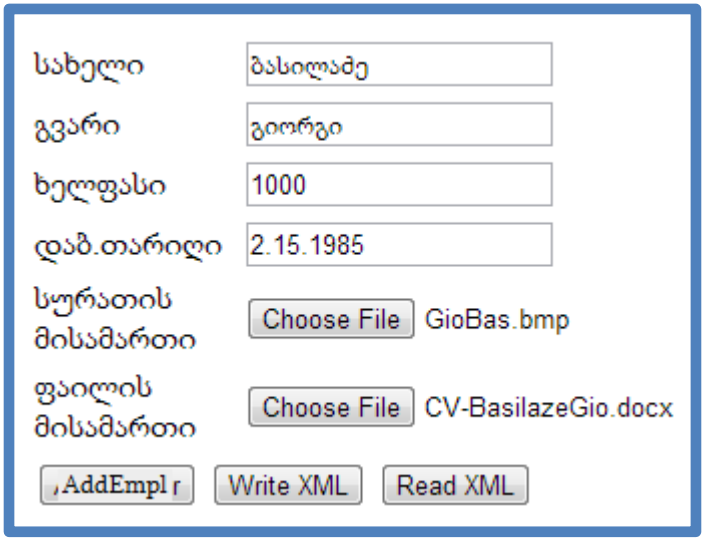


ნახ.4.5. ფოტო ინფორმაციის მიერთება



ნახ.4.6. ტექსტური ინფორმაციის მიერთება

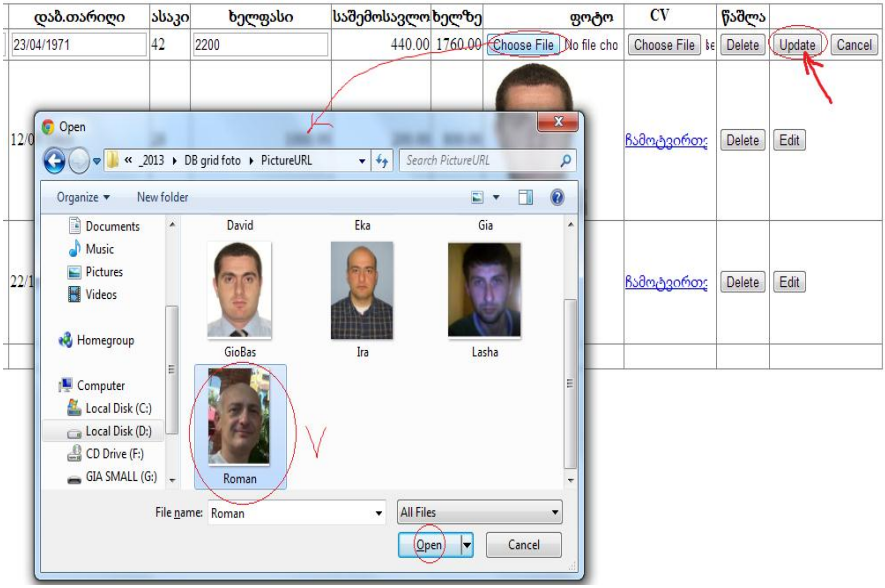
ახალი მოქალაქის შესახებ ინფორმაციის შეტანის, გრაფიკული და ტექსტური ფაილების არჩევის შემდეგ სარეგისტრაციო ფანჯარა გამოიყურება ნახაზის სახით.



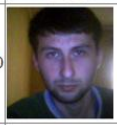
ნახ.4.7. ინფორმაცია მზადაა ბაზაში შესატანად

თუ Grid-ცხრილის შეტანილ სტრიქონში საჭიროა კორექტირების ჩატარება, მაშინ სტრიქონის მარჯვენა ბოლოში ვირჩევთ edit-ღილაკს და მივიღებთ 4.8 ნახაზზე ნაჩვენებ მდგომარეობას. Choos File-ს არჩევით გამოდის დიალოგური ფანჯარა და ვირჩევთ შესაბამის კატალოგს და ფაილს.

კორექტირების შემდეგ ვირჩევთ Update-ღილაკს და ცხრილის პირველ სტრიქონში ჩაემატება ფოტოსურათი. განახლებული ცხრილი ნაჩვენებია 4.9 ნახაზზე.

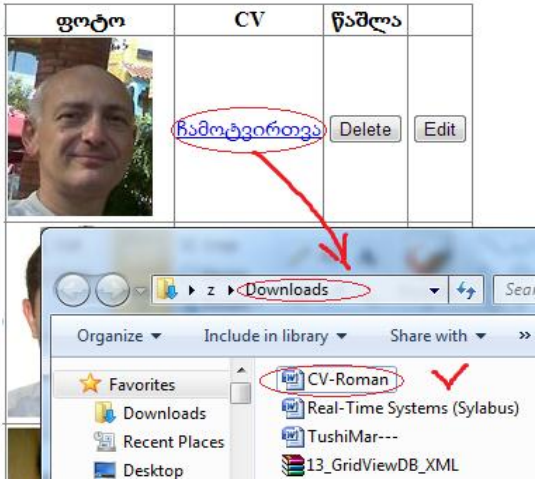


ნახ.4.8. უნდა ჩაემატოს ფოტოსურათი

ID	გვარი	სახელი	ხელფასი	საშემოსავლო	ხელზე	ფოტო	CV	წაშლა	
1	სამხარაძე	გიორგი	2200.00	440.00	1760.00		ჩამოტვირთვა	Delete	Edit
31	გიორგი	ბასილაძე	1000.00	200.00	800.00		ჩამოტვირთვა	Delete	Edit
41	ლამა	გაბინაშვილი	500.00	100.00	400.00		ჩამოტვირთვა	Delete	Edit
	ჯამი:		3700.00	740.00	2960.00				

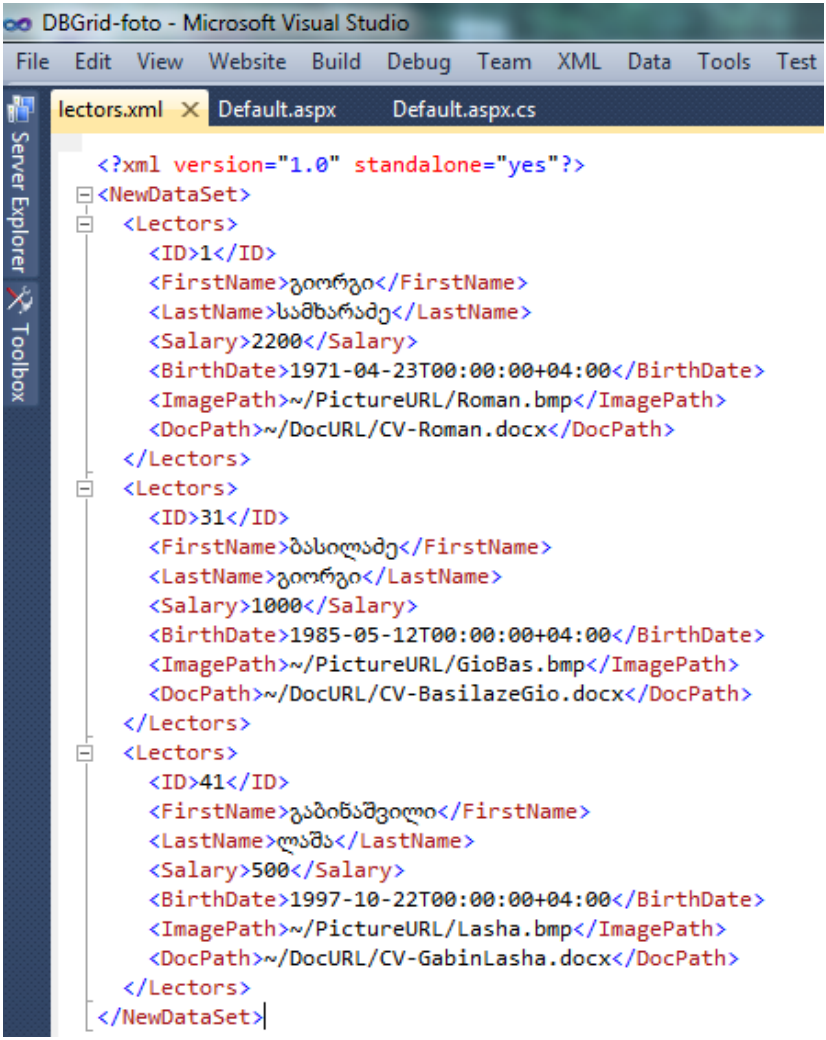
ნახ.4.9. შედეგები კორექტირების შემდეგ

ტექსტური ინფორმაცია, მაგალითად CV უნდა დაემატოს სტრიქონს „**ჩამოტვირთვა**“ ღილაკის დახმარებით, რის შემდეგაც გამოჩნდება ახალი დიალოგური ფანჯარა (ნახ.4.10).



ნახ.4.10. ტესქტ-ფაილის მიბმა ცხრილზე

Write XML ღილაკით განახლებული ცხრილი ჩაიწერება XML-ფაილში (მონაცეთა ბაზაში). Read XML ღილაკი უზრუნველყოფს ამ ბაზის ხელახალ წაკითხვას. 4.11 ნახაზზე ნაჩვენებია XML ფაილის ლისტინგი:



ნახ.4.11. XML ფაილის შიგთავსის სტრუქტურა

4.1.2. DBGrid-foto აპლიკაციის დიზაინის Default.aspx კოდი

ქვემოთ მოცემულია პროგრამული სისტემის დიზაინის ნაწილის ლისტინგი 4.2, რომელიც მომხმარებლის ინტერფეისის რეალიზაციას ემსახურება.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
                    Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head runat="server">
   <title>Untitled Page</title>
  </head>
<body>
  <form id="form1" runat="server">
    <div>
     <table style="position: relative">
       <tr>
         <td style="width: 100px">
         </td>
         <td style="width: 100px">
         </td>
         <td style="width: 223px">
          </td>   </tr>
         <tr>
           <td style="width: 100px">
            <asp:Label ID="Label1" runat="server" Style="position: relative"
Text="სახელი"></asp:Label></td>
           <td style="width: 100px">
```

```
            <asp:TextBox ID="txtFirstName" runat="server" Style="position:
relative"></asp:TextBox></td>
          <td style="width: 223px">
                <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
                  runat="server" ControlToValidate="txtFirstName"
          Display="Dynamic" ErrorMessage="შეავსეთ სახელი"
             Font-Size="8pt" Style="position:
             relative"></asp:RequiredFieldValidator></td>
        </tr>
        <tr>
          <td style="width: 100px; height: 26px">
           <asp:Label ID="Label2" runat="server" Style="position: relative"
Text="გვარი"></asp:Label></td>
          <td style="width: 100px; height: 26px">
           <asp:TextBox ID="txtLastName" runat="server" Style="position:
relative"></asp:TextBox></td>
          <td style="width: 223px; height: 26px">
           <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
                  runat="server" ControlToValidate="txtLastName"
         ErrorMessage="შეავსეთ გვარი" Font-Size="8pt" Style="position:
relative"></asp:RequiredFieldValidator>
          </td>
        </tr>
        <tr>
          <td style="width: 100px; height: 26px">
          <asp:Label ID="Label3" runat="server" Style="position: relative"
Text="ხელფასი"></asp:Label></td>
          <td style="width: 100px; height: 26px">
           <asp:TextBox ID="txtSalary" runat="server" Style="position:
relative"></asp:TextBox></td>
    <td style="width: 223px; height: 26px; text-align: left;">
          <asp:RangeValidator ID="RangeValidator1" runat="server"
ControlToValidate="txtSalary"
```

```
          Display="Dynamic" ErrorMessage="თანხა არასწორია"
                      MaximumValue="10000" MinimumValue="0"
          Style="left: -78px; position: relative; top: 30px"
                       Type="Currency" Font-Italic="False"
             Font-Size="X-Small"></asp:RangeValidator>
         </td>
        </tr>
        <tr>
          <td style="width: 100px; height: 26px">
            <asp:Label ID="Label4" runat="server" Style="position: relative"
Text="დაბ.თარიღი"></asp:Label></td>
          <td style="width: 100px; height: 26px">
            <asp:TextBox ID="txtBirthDate" runat="server" Style="position:
relative"></asp:TextBox></td>
          <td style="width: 223px; height: 26px">
            <asp:RequiredFieldValidator ID="RequiredFieldValidator3"
           runat="server" ControlToValidate="txtBirthDate"
           Display="Dynamic" ErrorMessage="შეავსეთ დაბადების თარიღი" Font-
           Size="8pt" Style="position: relative"></asp:RequiredFieldValidator>
        <asp:RangeValidator ID="RangeValidator2" runat="server"
ControlToValidate="txtBirthDate"
     Display="Dynamic" ErrorMessage="არასწორი თარიღი" Font-Size="X-Small"
MaximumValue="01/01/2079"
        MinimumValue="01/01/1900" Style="position: relative"
Type="Date"></asp:RangeValidator></td>
        </tr>
        <tr>
          <td style="width: 100px; height: 26px">
            <asp:Label ID="Label5" runat="server" Style="position: relative"
Text="სურათის მისამართი"></asp:Label></td>
        <td style="width: 100px; height: 26px">
          <nobr><asp:FileUpload id="FileUpload1"
runat="server"></asp:FileUpload> </nobr>
```

```
      </td>
       <td style="width: 223px; height: 26px">
        <asp:CustomValidator ID="CustomValidator1" runat="server"
ErrorMessage="CustomValidator"> </asp:CustomValidator></td>
      </tr>
      <tr>
        <td style="width: 100px; height: 26px">
       <asp:Label ID="Label6" runat="server" Style="position: relative"
Text="ფაილის მისამართი"></asp:Label></td>
         <td style="width: 100px; height: 26px">
          <nobr><asp:FileUpload id="FileUpload2"
runat="server"></asp:FileUpload> </nobr>
      </td>
       <td style="width: 223px; height: 26px">
       <asp:CustomValidator ID="CustomValidator2" runat="server"
ErrorMessage="CustomValidator"> </asp:CustomValidator></td>
     </tr>
   </table>
  </div>
  <table>
   <tr>
     <td>
     <asp:Button ID="Button1" runat="server" Text="Add Lector"
OnClick="Button1_Click" />
   </td>
    <td>
     <asp:Button ID="Button2" runat="server" OnClick="Button2_Click"
Text="Write XML"
                 CausesValidation="False" />
   </td>
    <td>
     <asp:Button ID="Button3" runat="server" OnClick="Button3_Click"
Text="Read XML" CausesValidation="False" />
```

```
    </td>
   </tr>
  </table>
     <asp:GridView ID="GridView1" runat="server" Style="position: relative; top:
0px;
       left: 0px;" AutoGenerateColumns="False" ShowFooter="True" FooterStyle-
ForeColor="red"
       OnRowDataBound="GridView1_RowDataBound"
OnRowDeleting="GridView1_RowDeleting"
      OnRowEditing="GridView1_RowEditing"
           OnRowUpdating="GridView1_RowUpdating"
           OnRowCancelingEdit="GridView1_RowCancelingEdit">
      <Columns>
       <asp:BoundField DataField="ID" HeaderText="ID"
                  ReadOnly="True" />
        <asp:BoundField DataField="LastName"
                  HeaderText="გვარი" FooterText="ჯამი:">
          <FooterStyle HorizontalAlign="Center" />
       </asp:BoundField>
       <asp:BoundField DataField="FirstName"
                                          HeaderText="სახელი" />
       <asp:BoundField DataField="BirthDate"
                  HeaderText="დაბ.თარიღი" HtmlEncode="False"
         DataFormatString="{0:dd/MM/yyyy}"
                              ApplyFormatInEditMode="True" />
      <asp:BoundField DataField="Age" HeaderText="ასაკი" HtmlEncode="False"
ReadOnly="True" />
       <asp:BoundField DataField="Salary"
                  HeaderText="ხელფასი" HtmlEncode="False"
                    DataFormatString="{0:F2}">
          <ItemStyle HorizontalAlign="Right" />
          <FooterStyle HorizontalAlign="Right" />
       </asp:BoundField>
```

```
      <asp:BoundField DataField="Tax"
                  HeaderText="საშემოსავლო" HtmlEncode="False"
                  DataFormatString="{0:F2}" ReadOnly="True">
          <ItemStyle HorizontalAlign="Right" />
          <FooterStyle HorizontalAlign="Right" />
      </asp:BoundField>
      <asp:BoundField DataField="NettoSalary"
                  HeaderText="ხელზე" HtmlEncode="False"
                  DataFormatString="{0:F2}" ReadOnly="True">
        <ItemStyle HorizontalAlign="Right" />
       <FooterStyle HorizontalAlign="Right" />
      </asp:BoundField>
       <asp:TemplateField HeaderText="ფოტო">
        <ItemTemplate>
         <asp:Image runat="server" ID="imgLector"
ImageUrl='<%#Eval("ImagePath") %>' />
      </ItemTemplate>
      <EditItemTemplate>
      <asp:FileUpload id="FileUpload1"
                    runat="server"></asp:FileUpload>
      </EditItemTemplate>
      </asp:TemplateField>
       <asp:TemplateField HeaderText="CV">
        <ItemTemplate>
        <asp:HyperLink runat="server" ID="lnkDocCV"
NavigateUrl='<%#Eval("DocPath") %>' Text="ჩამოტვირთვა"></asp:HyperLink>
       </ItemTemplate>
       <EditItemTemplate>
      <asp:FileUpload id="FileUpload2" runat="server"></asp:FileUpload>
       </EditItemTemplate>
      </asp:TemplateField>
      <asp:ButtonField ButtonType="Button" CommandName="Delete"
Text="Delete" HeaderText="წაშლა" />
```

```
    <asp:CommandField ButtonType="Button" CausesValidation="False"
ShowEditButton="True" />
     </Columns>
     <FooterStyle ForeColor="Red" />
    </asp:GridView>
        <asp:Label runat="server" ID="lblSalary">
         </asp:Label> 
  </form>
  </body>
</html>
```

4.1.3. DBGrid-foto აპლიკაციის ლოგიკის Default.aspx.sc კოდი

პროგრამა დაწერილია Visual C#.NET ენაზე.

```
// ------- ლისტინგი 3.3 -------------
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.IO;
using System.Globalization;

public partial class _Default : System.Web.UI.Page
{
 protected void Page_Load(object sender, EventArgs e)
  {
```

```
   CultureInfo culture = new CultureInfo("en-GB");
System.Threading.Thread.CurrentThread.CurrentCulture = culture;
  if (!IsPostBack)
    {
      DataSet lectors = GetDataSet();
      Session["MyDataSet"] = lectors;
 DataSet dsLectors = Session["MyDataSet"] as DataSet;
      GridView1.DataSource = dsLectors;
      GridView1.DataBind();
    }
  }

private void BindData()
  {
DataSet dsLectors = Session["MyDataSet"] as DataSet;
    GridView1.DataSource = dsLectors;
    GridView1.DataBind();
  }

private DataSet GetDataSet()
  {
   DataTable lectors = new DataTable("Lectors");
     //Add the DataColumn using all properties
     DataColumn id = new DataColumn("ID");
      id.DataType = typeof(int);
      id.Unique = true;
      id.AutoIncrement = true;
      id.AutoIncrementSeed = 1;
      id.AutoIncrementStep = 10;
      id.AllowDBNull = false;
      id.Caption = "ID";
      lectors.Columns.Add(id);
```

```
        //Add the DataColumn using defaults
     DataColumn firstName = new DataColumn("FirstName");
        firstName.DataType = typeof(string);
        firstName.MaxLength = 35;
        firstName.AllowDBNull = false;
        lectors.Columns.Add(firstName);

     DataColumn lastName = new DataColumn("LastName");
        lastName.DataType = typeof(string);
        lastName.MaxLength = 50;
        lastName.AllowDBNull = false;
        lectors.Columns.Add(lastName);

    DataColumn salary = new DataColumn("Salary", typeof(decimal));
        salary.DefaultValue = 0.00m;
        lectors.Columns.Add(salary);

    DataColumn birthDate = new DataColumn("BirthDate",
                                                typeof(DateTime));
        //birthDate.DefaultValue = DateTime.Now;
        birthDate.AllowDBNull = true;
        lectors.Columns.Add(birthDate);

    DataColumn imagePath = new DataColumn("ImagePath",
                                                typeof(string));
        imagePath.AllowDBNull = true;
        lectors.Columns.Add(imagePath);

    DataColumn docPath = new DataColumn("DocPath",
                                                typeof(string));
        docPath.AllowDBNull = true;
        lectors.Columns.Add(docPath);
```

```
DataColumn age = new DataColumn("Age",
                                                typeof(DateTime));
      age.ColumnMapping = MappingType.Hidden;
      age.Expression = "BirthDate";
      lectors.Columns.Add(age);

DataColumn tax = new DataColumn("Tax", typeof(decimal));
      tax.ColumnMapping = MappingType.Hidden;
      tax.DataType = typeof(decimal);
      tax.Expression = "salary*0.2";
      lectors.Columns.Add(tax);

DataColumn netto = new DataColumn("NettoSalary",
                                                typeof(decimal));
      netto.ColumnMapping = MappingType.Hidden;
      netto.DataType = typeof(decimal);
      netto.Expression = "salary - salary*0.2";
      lectors.Columns.Add(netto);

      DataSet ds = new DataSet();
      ds.Tables.Add(lectors);
      return ds;

   }
protected void Button1_Click(object sender, EventArgs e)
 {
  DataSet dsLectors = Session["MyDataSet"] as DataSet;
  DataTable dtLectors = dsLectors.Tables["Lectors"];

  DataRow newlector = dtLectors.NewRow();
    //newlector["ID"] = "1";
  newlector["FirstName"] = txtFirstName.Text;
  newlector["LastName"] = txtLastName.Text;
```

```
if (!String.IsNullOrEmpty(txtSalary.Text))
 newlector["Salary"] = Decimal.Parse(txtSalary.Text);

  newlector["BirthDate"]=DateTime.Parse(txtBirthDate.Text);

 //failis atvirtvis kodi
 if (FileUpload1.HasFile)
  {
   try
   {
        string filename =
                Path.GetFileName(FileUpload1.FileName);

FileUpload1.SaveAs(Server.MapPath("~/PictureURL/") +
        filename);
newlector["ImagePath"] = "~/PictureURL/" + filename;
    }
  catch (Exception ex)
    {
     CustomValidator1.IsValid = false;
    CustomValidator1.ErrorMessage = "Upload status: The file could not be
uploaded. The following error occured: " + ex.Message;
    }
 }
  // დოკუმენტის შენახვის კოდი
 if (FileUpload2.HasFile)
  {
   try
    {
         string filename =
                Path.GetFileName(FileUpload2.FileName);
          FileUpload2.SaveAs(Server.MapPath("~/DocURL/") +
        filename);
```

```
newlector["DocPath"] = "~/DocURL/" + filename;
      }
   catch (Exception ex)
    {
      CustomValidator2.IsValid = false;
     CustomValidator2.ErrorMessage = "Upload status: The file could not be
uploaded. The following error occured: " + ex.Message;
     }
  }

      dtLectors.Rows.Add(newlector);

object sumSalary = dtLectors.Compute("SUM(Salary)", "");
object sumTax = dtLectors.Compute("SUM(Tax)", "");
object sumNettoSalary = dtLectors.Compute("SUM(NettoSalary)", "");
      //lblSalary.Text = sumSalary.ToString();
 GridView1.Columns[5].FooterText =
                  String.Format("{0:F2}", sumSalary);
 GridView1.Columns[6].FooterText =
                  String.Format("{0:F2}",sumTax);
 GridView1.Columns[7].FooterText =
                  String.Format("{0:F2}", sumNettoSalary);
    Session["MyDataSet"] = dsLectors;
     GridView1.DataSource = dsLectors;
    GridView1.DataBind();
   }

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

```
protected void Button3_Click(object sender, EventArgs e)
 {
  if (Session["MyDataSet"] != null)
   {
    DataSet ds = Session["MyDataSet"] as DataSet;

    DataTable dtLectors = ds.Tables["Lectors"];
    dtLectors.Rows.Clear();

     ds.ReadXml(Request.PhysicalApplicationPath + "\\lectors.xml");
   dtLectors = ds.Tables["Lectors"];
   object sumSalary = dtLectors.Compute("SUM(Salary)", "");
   object sumTax = dtLectors.Compute("SUM(Tax)", "");
   object sumNettoSalary = dtLectors.Compute("SUM(NettoSalary)", "");
   //lblSalary.Text = sumSalary.ToString();
   GridView1.Columns[5].FooterText =
            String.Format("{0:F2}", sumSalary);
    GridView1.Columns[6].FooterText =
                      String.Format("{0:F2}", sumTax);
    GridView1.Columns[7].FooterText =
                  String.Format("{0:F2}", sumNettoSalary);

       GridView1.DataSource = ds;
       GridView1.DataBind();
     }
  }
protected void GridView1_RowDataBound(object sender,
                  GridViewRowEventArgs e)
 {
  if (e.Row.RowType == DataControlRowType.DataRow)
   {
    DateTime dt = (DateTime)DataBinder.Eval(e.Row.DataItem, "BirthDate");
```

```
    int years = DateTime.Now.Year - dt.Year;
    e.Row.Cells[4].Text = years.ToString();

     if (years>65) e.Row.BackColor =
                          System.Drawing.Color.Gray;
   }
     //e.Row.Cells[4].Text = ((DateTime)e.Row.DataItem).ToShortDateString();
 }

protected void GridView1_RowDeleting(object sender,
          GridViewDeleteEventArgs e)
{
  DataSet dsLectors = Session["MyDataSet"] as DataSet;
  DataTable dtLectors = dsLectors.Tables["Lectors"];
  dtLectors.Rows[e.RowIndex].Delete();
  BindData();
}
protected void GridView1_RowEditing(object sender,
              GridViewEditEventArgs e)
 {
   GridView1.EditIndex = e.NewEditIndex;
   BindData();
 }
protected void GridView1_RowUpdating(object sender,
        GridViewUpdateEventArgs e)
 {
   //Retrieve the table from the session object.
  DataSet dsLectors = Session["MyDataSet"] as DataSet;
  DataTable dt = dsLectors.Tables["Lectors"];

  //Update the values.
  GridViewRow row = GridView1.Rows[e.RowIndex];
```

```
dt.Rows[row.DataItemIndex]["LastName"] =
        ((TextBox)(row.Cells[1].Controls[0])).Text;
 dt.Rows[row.DataItemIndex]["FirstName"] =
       ((TextBox)(row.Cells[2].Controls[0])).Text;
dt.Rows[row.DataItemIndex]["BirthDate"] =
DateTime.Parse(((TextBox)(row.Cells[3].Controls[0])).Text,
System.Globalization.CultureInfo.CurrentCulture);

dt.Rows[row.DataItemIndex]["Salary"] =
      ((TextBox)(row.Cells[5].Controls[0])).Text;

   //failis atvirtvis kodi

 FileUpload fileUpload = ((row.Cells[8].Controls[1]))
          as FileUpload;
 if (fileUpload.HasFile)
  {
   try
    {
     string filename =
           Path.GetFileName(fileUpload.FileName);
          fileUpload.SaveAs(Server.MapPath("~/PictureURL/") +
         filename);
  dt.Rows[row.DataItemIndex]["ImagePath"] =
                    "~/PictureURL/" + filename;
    }
  catch (Exception ex)
    {
   }
}

 fileUpload = ((row.Cells[9].Controls[1])) as
         FileUpload;
```

```
 if (fileUpload.HasFile)
  {
   try
    {
     string filename =
                Path.GetFileName(fileUpload.FileName);
              fileUpload.SaveAs(Server.MapPath("~/DocURL/") +
             filename);
dt.Rows[row.DataItemIndex]["DocPath"] = "~/DocURL/" +
             filename;
    }
   catch (Exception ex)
     {
     }
   }
     //Reset the edit index.
  GridView1.EditIndex = -1;
    //Bind data to the GridView control.
   BindData();
 }
protected void GridView1_RowUpdated(object sender,
             GridViewUpdatedEventArgs e)
  {

  }
protected void GridView1_RowCancelingEdit(object
          sender, GridViewCancelEditEventArgs e)
 {
   GridView1.EditIndex = -1;
   BindData();
 }
}
```

4.2. მომხმარებელთა ინტერფეისების დამუშავება

დაპროექტებული და აგებულია ელექტრონული საარჩევნო სისტემის მხარდამჭერი პროგრამული უზრუნველყოფა, რომლის ჩაწერაც მოხდება საარჩევნო უბნებზე არსებულ რეგისტრატორთათვის განკუთვნილ კომპიუტერებზე.

ელექტრონული რეგისტრაციის ფორმა შექმნილია თანამედროვე ტექნოლოგიების გამოყენებით, ისეთების როგორებიცაა Windows Presentation Foundation (WPF) და Metro Style App, რომელიც დღეისათვის ინოვაციას წარმოადგენს და გამოიყენება Microsoft Windows 8-ში.

ჩვენ შევეცადეთ რომ პროგრამა ყოფილიყო ძალიან მარტივი და ადვილად სამართავი, რათა ნებისმიერ საარჩევნო უბნის რეგისტრატორს გაადვილებოდა მასთან მუშაობა და მომხდარიყო დროის მაქსიმალურად გამოყენება. ასევე გათვალისწინებულია მომხმარებელთა ინტერფეისების დამუშავება საქართველოს სომეხი, აზერბაიჯანელი, ოსი და აფხაზი ეროვნების წარმომადგენელთათვის.

აღნიშნული პროგრამული უზრუნველყოფის გამოყენება, გვაძლევს გარანტიას სრულად გამოვრიცხოთ ისეთი ტერმინები და მისგან გამომდინარე შექმნილი უხერხულობები, როგორიც არის *საარჩევნო სიების გაყალბება*, *მკვდარი სულები საარჩევნო სიებში*, *გამორჩენები საარჩევნო სიებში* და არჩევნების მიმდინარეობის პროცესში – „*კარუსელები*“ (რაც გამოიხატება ერთი ამომრჩევლის ან ამომრჩეველთა ჯგუფის მიერ რამდენიმე საარჩევნო უბანზე საკუთარი ხმის დაფიქსირებით).

პროგრამა არ იძლევა უფლებას, რომ ერთმა ადამიანმა ერთზე მეტ საარჩევნო უბანზე გაიაროს რეგისტრაცია და მისცეს ხმა. იმ შემთხვევაში თუ ამომრჩეველს უკვე გავლილი აქვს რეგისტრაცია

და შესაბამისად მისი კონსტიტუციური უფლება – მისცეს ხმა სასურველ კანდიდატს უკვე აღსრულებულია, ხელმეორედ ნებისმიერ საარჩევნო უბანზე მისვლას პროგრამა აღმოაჩენს. საუბნო საარჩევნო კომისიის რეგისტრატორს ეცნობება იმის შესახებ, თუ რა დროს და რომელ უბანზე გაიარა უკვე რეგისტრაცია ამა თუ იმ ამომრჩეველმა.

კომისიის წევრი ვალდებულია აღკვეთოს ხელმეორედ ხმის მიცემა და აცნობოს სამართალდამცავ ორგანოებს ზემოაღნიშნული ფაქტი შესაბამისი რეაგირებისათვის.

პროგრამული უზრუნველყოფისათვის გამოვიყენეთ უახლესი ტექნიკა და თანამედროვე ტექნოლოგიები: თითის ანაბეჭდის სკანერი, ელექტრონული ხელმოწერის პანელი, ხმის ჩამწერი, შემდეგ მისი ამომცნობი სისტემები, ფოტოაპარატი და ბიომეტრული სურათების შედარების სისტემები.

ამ ყველაფრის გათვალისწინებით თუ მონაცემები არ იქნება გატარებული მონაცემთა ბაზაში, თქვენ არ მოგეცემათ არჩევნებში მონაწილეობის უფლება. პროგრამა ითვალისწინებს, გარდა პირადი ნომრით ამომრჩევლის იდენტიფიცირებისა, ისეთი დამატებითი იდენტიფიკატორების შემოტანას და მათ რეალიზაციას, როგორებიცაა ამომრჩევლის თითის ანაბეჭდი, ხმა, ხელმოწერა და ბიომეტრული სურათი.

ახლა დეტალურად განვიხილოთ პროგრამის მუშაობის ძირითადი პრინციპი:

ცენტრალურ საარჩევნო კომისიას ექნება წინასწარ ფორმირებული მონაცემთა ბაზა, თუ რომელ საარჩევნო უბანზე რომელი კომისიის წევრი არის მიმაგრებული და რა ფუნქცია-მოვალეობები აქვს ნაკისრი.

საარჩევნო უბნის გახსნისას ამომრჩევლთა მიღების დაწყებამდე, როდესაც რეგისტრატორი გახსნის პროგრამას, პირველ რიგში თვითონ გაივლის იდენტიფიკაციას და როგორც კი მისი

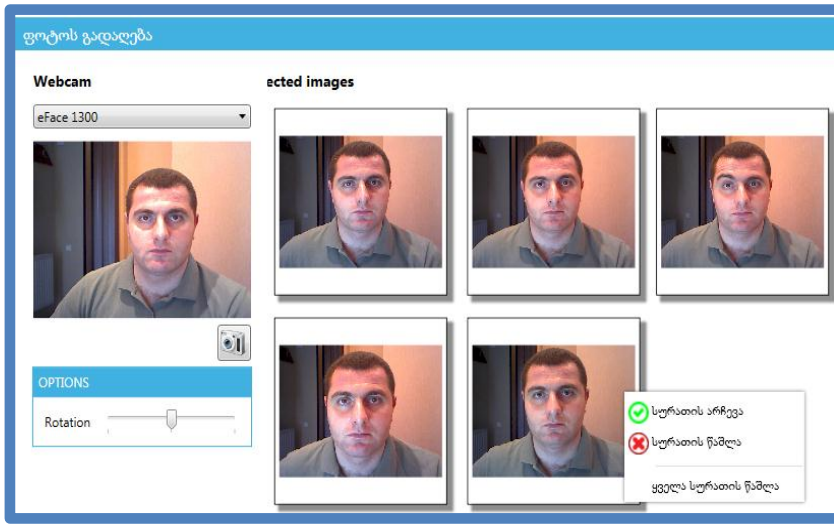
აუთენტურობა დადგინდება, როგორც რეგისტრატორის, შემდეგ მოხდება მოთხოვნა Access Code(AC)-ის, რომელიც იქნება უნიკალური და წინასწარ დალუქული კონვერტით ექნება მიღებული საუბნო კომისიის თავჯდომარეს.

აღნიშნული კონვერტი გაიხსნება და შედგება ოქმი, რომელზეც კომისიის თავმჯდომარესთან ერთად ხელმომწერები იქნებიან თავჯდომარის მოადგილე და საუბნო კომისიის წევრები. აღნიშნული AC კოდის შეტანის შემდეგ რეგისტრატორებს მიეცემათ უფლება დაიწყონ ამომრჩეველთა მიღება და რეგისტრაცია. ყოველივე ეს აძლიერებს და ამყარებს უსაფრთოების ხარისხს და მეტ დამაჯერებლობას სძენს ელეტრონულ საარჩევნო სისტემას.

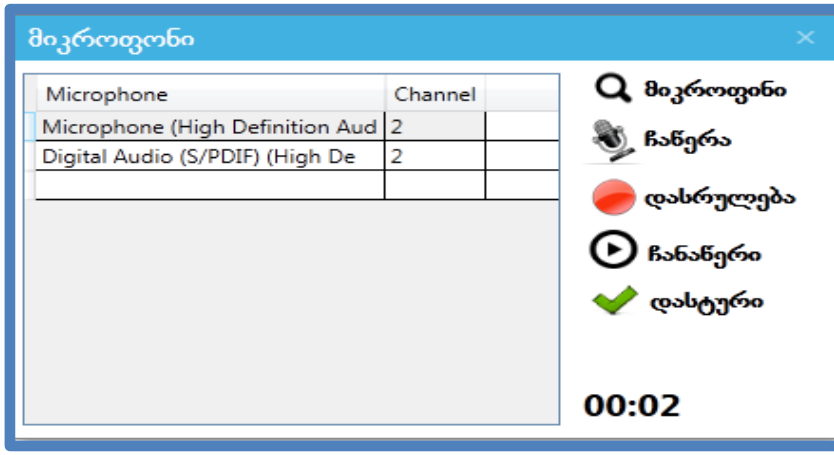
უბანზე მისულ ამომრჩეველს უღებენ ბიომეტრულ სურათს ვებ-კამერის მეშვებით (რეგისტრატორს შეუძლია ერთი ან რამდენიმე სურათის გადაღება და არჩევანის გაკეთება).

ვებ-კამერა იმართება პროგრამიდან და აკეთებს დროებით ჩანაწერებს კომპიუტერში, სადაც ინახება მოქალაქის სურათი თავისივე უნიკალური დასახელებით, შერჩეული სურათის დაშლა ხდება ბიტებად და მონაცემთა ბაზაში ჩაწერა, ხოლო დანარჩენი სურათები ავტომატურად იშლება მყარი დისკიდან. კამერის სამართავად პროგრამული უზრუნველყოფა იყენებს კომპიუტერში არსებულ დრაივერს, რომელსაც პოულობს და ავტომატურად აყენებს (ნახ.4.12).

ხმის ჩასაწერად და მიკროფონზე წვდომის განსახორციელებლად გამოვიყენეთ ყველასთვის კარგად ცნობილი და გამოცდილი ბიბლიოთეკა, როგორიცაა Naudio. მისი მეშვეობით ხდება წვდომა მიკროფონზე და იწყება ძიება კომპიუტერზე მიერთებული ყველა არსებული მიკროფონის (ნახ.4.13).

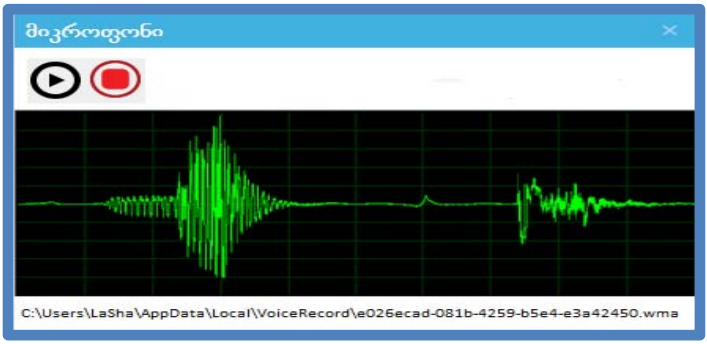


ნახ.4.12.



ნახ.4.13.

რეგისტრატორმა უნდა აირჩიოს მიკროფონი და ჩაიწეროს ხმა, რომელიც ასევე დროებით ჩაიწერება კომპიუტერის მეხსიერებაში. ჩაწერის შემდეგ შესაძლებელია ჩაწერილი ხმის მოსმენა და მისი გრაფიკული გამოსახულების ნახვა. (ნახ.4.14).



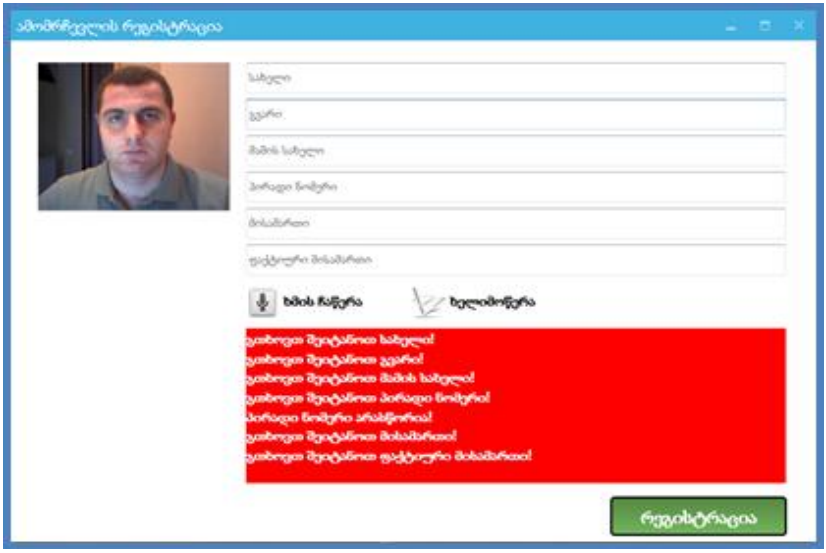
ნახ.4.14.

პროგრამული უზრუნველყოფა შეიცავს აგრეთვე მთელ რიგ ვალიდაციებს, რომელიც არ მისცემს რეგისტრატორს რაიმე სახის შეცდომის დაშვების უფლებას ამომრჩეველთა რეგისტრაციის დროს. კერძოდ, სახელის, გვარის, მამის სახელის, მისამართის და ფაქტობრივი მისამართის შეტანა და რეგისტრაცია მოხდება წინასწარ განსაზღვრული ფონტით-Sylfaen. რაც შეეხება პირად ნომერს, იგი აუცილებლად იქნება 11- ნიშნა რიცხვითი მონაცემი.

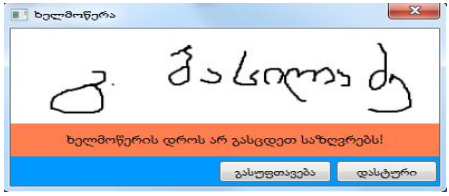
აღნიშნული შეცდომების დაშვების დროს მივიღებთ შემდეგი სახის გამაფრთხილებელ დიალოგურ ფანჯარას (ნახ.4.15).

პროგრამა გამორიცხავს ამომრჩევლის შესახებ არასრული ინფორმაციის შემთხვევაში ბაზაში ჩანაწერის გაკეთებას, რაც თავის მხრივ, ხელს უშლის არჩევნებზე რეგისტრირებული ამომრჩეველთა რიცხვის ხელოვნურ ზრდას.

პროგრამული უზრუნველყოფა ითვალისწინებს აგრეთვე ელექტრონული ხელმოწერის შეტანა-შენახვის მოდულს (ნახ.4.16) და თითის ანაბეჭდის შეტანა-შენახვა-იდენტიფიცირებას.



ნახ.4.15



ნახ.4.16.

ავტორების მიერ ხდება ასევე სხვა დეტალების და ბლოკების შესწავლა და კვლევა, რაც უახლოეს მომავალში იქნება გასაჯაროებული სხვადასხვა სამეცნიერო სტატიებისა და ნაშრომების სახით. ბოლოს გვინდა კიდევ ერთხელ აღვნიშნოთ, რომ შემოთავაზებული პროგრამული უზრუნველყოფის გამოყენებით ელექტრონული არჩევნების ჩატარება იქნება გარანტი მოსახლეობაში არჩევნებისადმი ნდობის აღდგენის და მისადმი არა ნიჰილისტური განწყობის დასაბამი.

4.3. მომხმარებელთა ინსტრუქციები

წინამდებარე პარაგრაფში შემოთავაზებულია ელექტრონული საარჩევნო სისტემის მომხმარებელთა ჩამონათვალი და თითოეულის უფლება-მოვალეობანი. შედგენილია პროგრამული სისტემის ფუნქციონირების ინსტრუქციები და წესები მომხმარებელთათვის. მათი იერარქიული სტრუქტურა ასეთია:

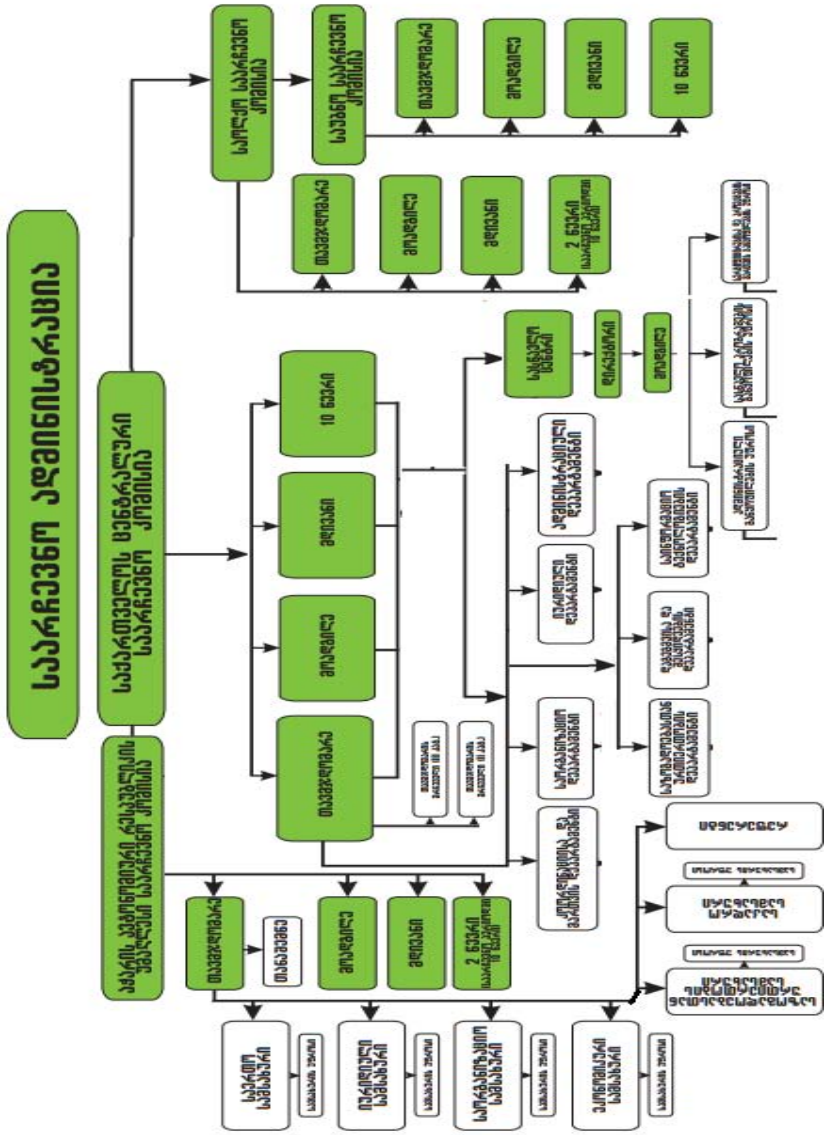
1. მომხმარებლები ცენტრალურ საარჩევნო კომისიაში;
2. მომხმარებლები საოლქო საარჩევნო კომისიაში;
3. მომხმარებლები საარჩევნო უბნებზე.

ვიდრე განვიხილავდეთ თითოეულს დეტალურად, წარმოგიდგენთ ცენტრალური საარჩევნო კომისიის არსებულ სტრუქტურას: http://www.cec.gov.ge/ge/home (ნახ.4.17).

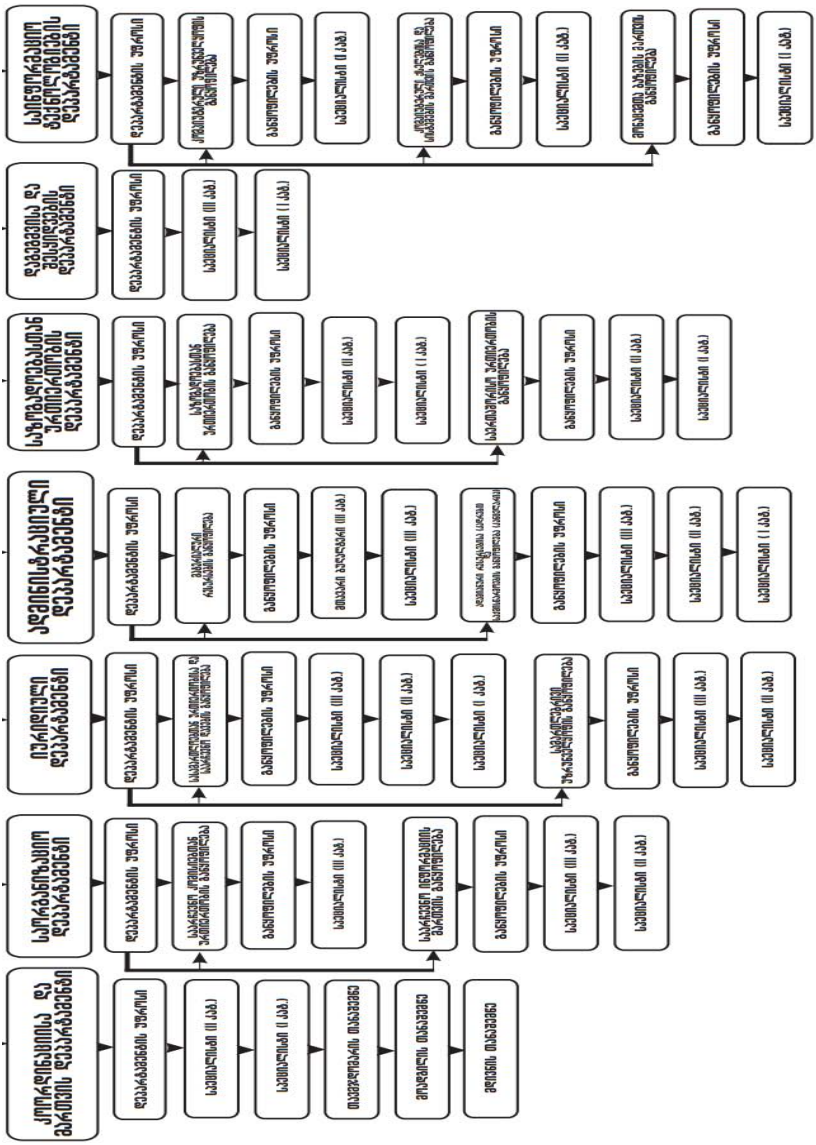
მიგვაჩნია, რომ მომხმარებელთა განლაგება და მათი როლები ცენტრალურ საარჩევნო კომისიაში სრულად ემთხვევა ჩვენ მიერ წარმოდგენილ სტრუქტურას, ამიტომ ცვლილებები არაა გათვალისწინებული.

რაც შეეხება საოლქო საარჩევნო კომისიას, ვფიქრობთ, რომ უნდა მოხდეს საინფორმაციო ტექნოლოგიების დეპარტამენტის დამატება, რომელშიც სტრუქტურულად გადანაწილებული იქნება ფუნქციები ქსელის ადმინისტრატორებს, მონაცემთა ბაზების ადმინისტრატორებს და სისტემურ ადმინისტრატორებს შორის.

აქვე უნდა დავამატოთ, რომ ყველა ზემოჩამოთვლილი სამუშაო ადგილისათვის თანამშრომლების აყვანა მოხდება მხოლოდ არჩევნების პერიოდში, ისევე როგორც ეს რეგულირდება საოლქო საარჩევნო კომისიის წევრების შემთხვევაში, სადაც არასაარჩევნო პერიოდში დასაქმებულია კომისიის 2 წევრი, ნაცვლად 10-ისა. ყველა სხვა თანამდებობა რჩება უცვლელი.



ნახ.4.17-1. ცესკო-ს სტრუქტურა



ნახ.4.17-2 (გაგრძელება)

რაც შეეხება საუბნო საარჩევნო კომისიებს, აქაც აუცილებელია დაემატოს ინფორმაციული ტექნოლოგიების დეპარტამენტი. დეპარტამენტში შევლენ ერთი სისტემური ადმინისტრატორი და ერთი ქსელის ადმინისტრატორი. ეს სპეციალისტებიც მხოლოდ საარჩევნო პერიოდში იქნებიან დასაქმებულნი.

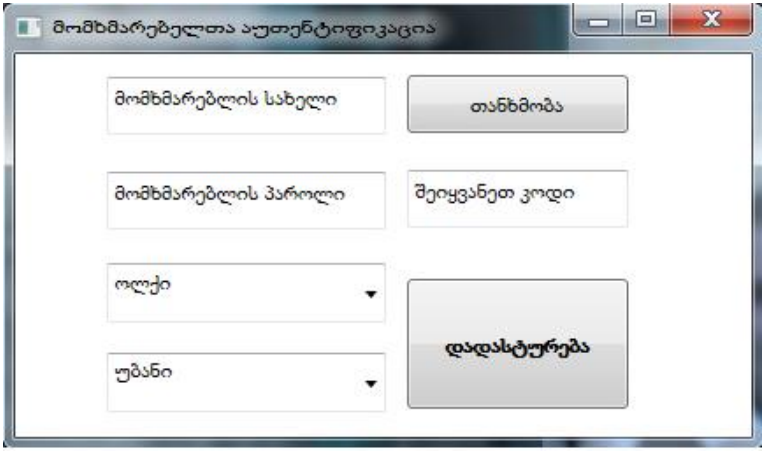
აუცილებელია გამოვყოთ და განვიხილოთ ყველაზე შრომატევადი, პირველი საფეხურის მომხმარებლები. ესაა უშუალოდ საარჩევნო უბნებზე მომუშავე რეგისტრატორთა ჯგუფები, რომელთა თავდადების და კეთილსინდისიერი შრომით მიიღწევა საუკეთესო შედეგები.

ქვემოთ შემოთავაზებული გვაქვს სცენარი, თუ დეტალურად როგორ წარიმართება არჩევნების დღე საარჩევნო უბნებზე ელექტრონული სისტემით არჩევნების ჩატარებისას:

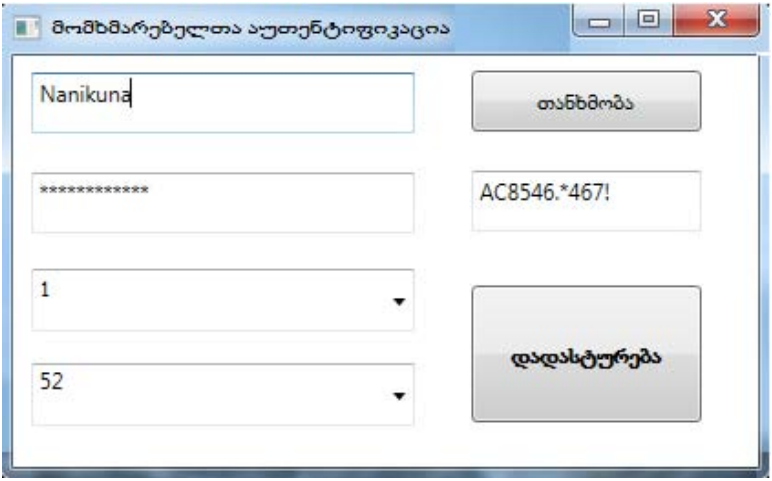
1. საარჩევნო უბნის გახსნისას რეგისტრატორი აამოქმედებს პროგრამას (ნახ.4.18.) და გაივლის აუთენტიფიკაციას. შეიტანს სახელს, გვარს, ოლქს, უბანს და დააჭერს ღილაკს „თანხმობა“. ამის შემდეგ შეიტანს Access Code (AC)-ს, რომელიც იქნება უნიკალური და წინასწარ დალუქული კონვერტით მიღებული საუბნო კომისიის თავჯდომარეს. შემდეგ დააჭერს ღილაკს „დადასტურება“ (ნახ.4.19);

2. გაიხსნება უშუალოდ სარეგისტრაციო ფორმა (ნახ.4.20.), სადაც მოხდება სრული ინფორმაციის რეგისტრაცია ამომრჩეველთა შესახებ;

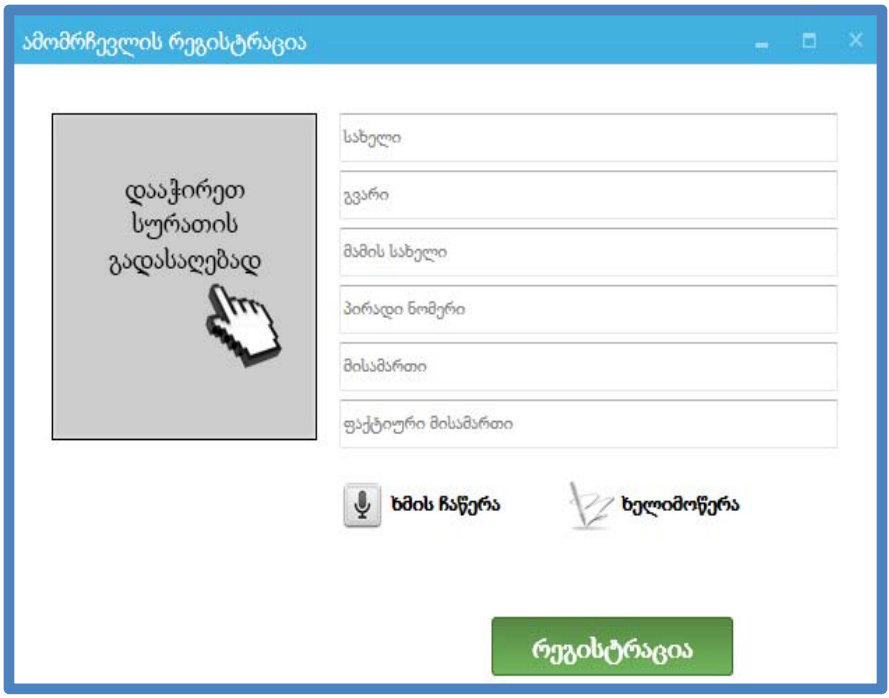
3. რეგისტრატორს შეაქვს მოქალაქის სახელი, გვარი, მამის სახელი, პირადი ნომერი, მისამართი და ფაქტობრივი მისამართი. პირადი ნომრის შემთხვევაში მოქმედებს შეზღუდვა და იგი აუცილებლად მოითხოვს, რომ შეტანილი ინფორმაცია იყოს ციფრი და აუცილებლად 11 სიმბოლო;



ნახ.4.18



ნახ.4.19



ნახ.4.20

4. **„დააჭირეთ სურათის გადასაღებად“** ღილაკის ამოქმედებით გამოჩნდება ახალი ფანჯარა, სადაც შეგვიძლია ფოტოაპარატის გამოსახულების არჩევით გადავიღოთ რამდენიმე სურათი და შემდეგ, ამ სურათთაგან ერთ-ერთზე, მაუსის მარჯვენა ღილაკით გამოსულ დიალოგურ ფანჯარაში **„სურათის არჩევა“** შევარჩიოთ სასურველი ფოტო (ნახ.4.12);

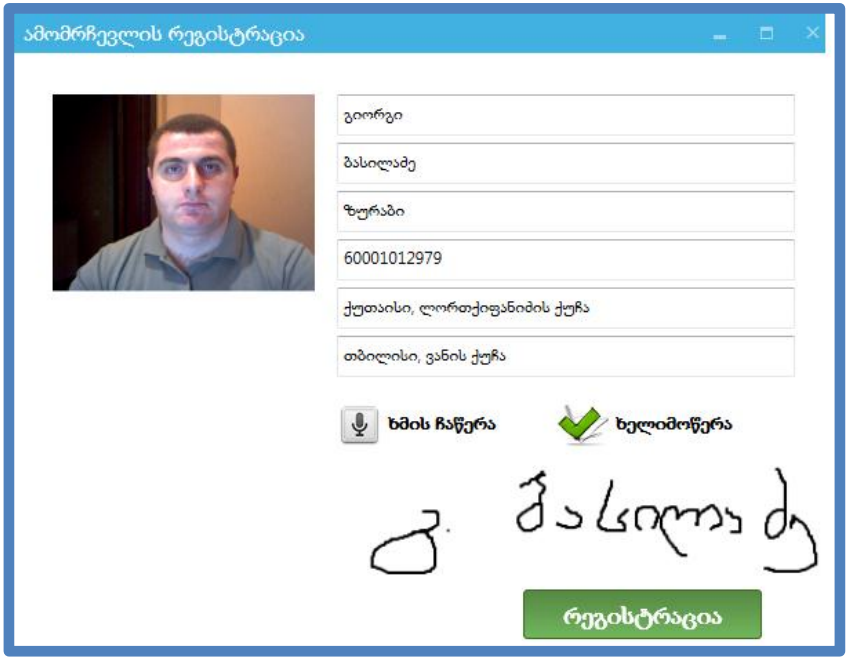
5. ღილაკზე **„ხმის ჩაწერა“** დაწკაპუნებით, გამოდის დიალოგური ფანჯარა, სადაც რეგისტრატორი ერთხელ აირჩევს მიკროფონს და აჭერს ღილაკზე **„ჩაწერა“**. მოქალაქე თავის სახელსა

და გვარს ამბობს მიკროფონში. ამის შემდეგ ხდება მისი დამახსოვრება ღილაკზე „**დასტური**“ დაჭერით;

6. „**ხელისმოწერა**“ღილაკის ამოქმედებით გამოდის ახალი დიალოგური ფანჯარა, სადაც მითითებულია, რომ მოქალაქე არ უნდა გასცდეს მითითებულ ველებს და უნდა მოაწეროს ხელი. ამის შემდეგ ხდება მისი დამახსოვრება.

7. ღილაკზე „**თითის ანაბეჭდი**“ დაწკაპუნებით გამოიტანება დიალოგური ფანჯარა, რომელზეც აისახება მოქალაქის მიერ თითის ანაბეჭდის სკანერზე თითის დადებით ანაბეჭდი. ამის შემდგომ ხდება მისი დამახსოვრება.

4.21 ნახაზზი გვიჩვენებს რეგისტრაციის ფორმის შევსებულ, საბოლოო ვარიანტს.



ნახ.4.21.

4.4. შედეგების ანალიზი და შემდგომი სინქრონიზაცია

ჩვენ განვიხილეთ ელექტრონული წესით არჩევნების ჩატარების დადებითი მხარეები და ვისაუბრეთ სისტემის ეკონომიკურ თუ სოციალურ ეფექტებზე. გვინდა ასევე ყურადღება გავამახვილოთ კიდევ ერთ უპირობო მიღწევაზე, რაც ელექტრონულ საარჩევნო სისტემის დანერგვას შეუძლია მოუტანოს ქვეყანას და დიდი როლი ითამაშოს მის დემოკრატიულ განვითარებაში.

ცნობილი და აღიარებული ფაქტია ის, რომ ინფორმაცია არის უდიდესი მნიშვნელობის მატარებელი, განსაკუთრებით თუ ის საჭირო ინფორმაციაა. ინფორმაციის ფლობა განაპირობებს მსოფლიოს გულისცემას. საჭირო ინფორმაციის ანალიზის საფუძველზე მიიღება სწორი და ადეკვატური გადაწყვეტილებები, რაც, თავის მხრივ, განაპირობებს ქვეყნებისა და ბიზნესების წარმატებას. მწირი ინფორმაციის შემთხვევაში ძალიან რთულია სწორი გადაწყვეტილებების მიღება და ძირითადად ხდება გამოცდილებისა და რეალური ნაბიჯების გადადგმის ხარჯზე გამოტანილი დასკვნებით, რაც ხშირ შემთხვევაში სრული კრახით მთავრდება ხოლმე ქვეყნისთვისაც და ბიზნესისთვისაც.

ელექტრონული საარჩევნო სისტემა სახელმწიფოსათვის ისეთ უმნიშვნელოვანეს ინფორმაციას უყრის თავს, რომელიც მხოლოდ ერთეულ შემთხვევებში და ერთეულ პიროვნებებზე თუ მოეპოვება ქვეყანას. ეს ინფორმაცია სწორედ ის მულტიმედიური მონაცემთა ბაზებია, რომლის ანალოგიც ჯერ არ შექმნილა და არც განიხილება მისი შექმნა მხოლოდ ერთი მიზეზით, მისი სირთულიდან გამომდინარე, რაც მოქალაქეთა სამოქალაქო რეესტრში იძულების წესით მიყვანას და შესაბამისი ინფორმაციების მოძიებას გულისხმობს. თუ ამას იძულების სახე არ

ექნება და ცხადია, რომ ვერც ექნება, ეს პროცესი შესაძლოა გაიწელოს წლობით და სასურველი შედეგიც მიუღწეველი დარჩება. ჩვენს შემთხვევაში კი მოქალაქე თავისი ნებით მიდის არჩევნებზე და თავისი სურვილით აძლევს სახელმწიფოს მისთვის საჭირო ინფორმაციას და მხოლოდ ამის შემდეგ ახორციელებს თავის კონსტიტუციურ უფლებას, რაც არჩევნებში ხმის მიცემას გულისხმობს.

ამ ტიპის ინფორმაციას განეკუთვნება მოქალაქეთა ბიომეტრული სურათები, ელექტრონული ხელმწერები, ხმის აუდიო ჩანაწერები და თითის ანაბეჭდები. ეჭვგარეშეა, რომ ეს არის უაღრესად დიდი საქმე საქართველოს იუსტიციის სამინისტროსათვის, კერძოდ კი – სამოქალაქო რეესტრისათვის, რადგან საუკეთესო შემთხვევაში აქვთ მხოლოდ მოქალაქეთა ბიომეტრული სურათები, რაც პირად ნომერზეა მიბმული.

ჩვენ განვიხილავთ, რომ განხორციელდება ელექტრონული არჩევნების პროცესში მოგროვებული ინფორმაციის ანალიზი და მხოლოდ ამის შემდგომ მოხდება სამოქალაქო რეესტრის მონაცემთა საცავებში აღნიშნული ინფორმაციის გადატანა.

მოხდება პირადი ნომრების და გვარის იდენტიფიკატორების შემოტანა და ამის საფუძველზე ორივე, არსებული მონაცემებისა და ახალი, ელექტრონული სისტემის შედეგად მოპოვებული მონაცემების სრული სინქრონიზაცია.

შედეგად ჩვენ მივიღებთ მონაცემთა საცავს, სადაც მოსახლეობის შესახებ გვექნება შემდეგი სახის მონაცემები: სახელი, გვარი, მამის სახელი, პირადი ნომერი, მისამართი, ფაქტობრივი მისამართი, ბიომეტრული ფოტოსურატი, თითის ანაბეჭდი, ხმის აუდიო ჩანაწერი და ელექტრონული ხელმოწერა.

აღნიშნული მონაცემთა ბაზები, ჩვენი ღრმა რწმენით, გახდება სამართალდამცავი სტრუქტურებისათვის მიმზიდველი და მათი საქმიანობის ეფექტურობას აამაღლებს. იგი გახდება

გზამკვლევი ჩადენილ დანაშაულებათა გახსნის საქმეში. ყოველივე ზემოხსენებული აუცილებელია დემოკრატიული სახელმწიფოს მშენებლობისათვის, სადაც კანონი იქნება უზენაესი, ყველა დამნაშავე იქნება იდენტიფიცირებული და დასჯილი კანონის სრული სიმკაცრით.

ვიმედოვნებთ, რომ სახელმწიფო სტრუქტურებიდან იქნება მზაობა და სურვილი, მოხდეს აღნიშნული იდეის დანერგვა და ხორცშესხმა. ჩვენ კი ჩვენის მხრივ ვაცხადებთ სრულ მზადყოფნას, აღმოვუჩინოთ დახმარება სახელმწიფოს აღნიშნული პროექტის განხორციელებაში.

4.5. სისტემის განვითარების კონცეფცია

საბოლოო სახით ელექტრონული საარჩევნო სისტემა მოიცავს შემდეგ პროცედურებს:

1. რეგისტრაციის გავლა ნებისმიერ რეგისტრატორთან წინასწარ დადგენილი წესების დაცვით. ტექსტური მონაცემების მონაცემთა ბაზაში შეყვანის შემდეგ უნდა მოხდეს თითის სკანირება და ანაბეჭდის ბაზაში გადატანა; ელექტრონული წესით ხელმოწერის განორციელება და მისი მულტიმედიურ მონაცემთა ბაზაში გადატანა; ბიომეტრული ფოტოსურათის გადაღება და მონაცემთა ბაზის შესაბამის ველში მისი დამახსოვრება; აუდიო ფაილის, რომელიც ამომრჩევლის ხმას ჩაიწერს, მულტიმედიურ ბაზის შესაბამის ველში ჩადება;

2. მას შემდეგ, რაც მოქალაქე წარმატებით გაივლის იდენტიფიცირებას, მიიღებს დაშვებას საარჩევნო კაბინაში და გააქტიურდება სენსორული ეკრანი, რომელზეც გამოტანილი იქნება კანდიდატების ჩამონათვალი;

3. ხმის მიცემა – სასურველი კანდიდატის გასწვრივ ნომერი შემოიხაზება სპეციალური კალმით;

4. საარჩევნო უბნის დატოვება.

ამრიგად, ჩვენ კონცეფციაში აღწერილი საარჩევნო სისტემა აღარ საჭიროებს საარჩევნო სიაში თავის გადამოწმებას და უბნებზე მარკირებას, რომლის წინააღმდეგია მოქალაქეთა ნაწილი, ბიულეტენს და შესაბამისად არც საარჩევნო ურნას. იგი უზრუნველყოფს მაქსიმალურ უსაფრთხოებას და სანდოობას. ამ ფაქტორების გათვალისწინებით შეგვიძლია ვისაუბროთ სისტემის უდიდეს უპირატესობაზე.

ელექტრონული საარჩევნო სისტემის დანერგვა და არჩევნების ჩატარება შეგვიძლია განვიხილოთ როგორც კონცეფცია ეკონომიკური ეფექტურობის თვალსაზრისით.

თუ გავითვალისწინებთ იმას, რომ ძირითადი ხარჯი იქნება ერთჯერადი, რაც მოიცავს ტექნიკისა და პროგრამული უზრუნველყოფის სისტემის შექმნას, ხოლო შემდგომ მოხდება მხოლოდ და მხოლოდ მონტაჟისა და დემონტაჟის თანხების გამოყოფა სახელმწიფო ბიუჯეტიდან, შეგვიძლია ვთქვათ, რომ ელექტრონული საარჩევნო სისტემით არჩევნების ჩატარება გაცილებით იაფი დაუჯდება სახელმწიფოს, ვიდრე არსებული-ტრადიციული სისტემით. სახელმწიფო აღარ დაბეჭდავს ბიულეტინებს, რაც კოლოსალური დანაზოგი იქნება სახელმწიფო ბიუჯეტისთვის.

აღსანიშნავია სოციალური ეფექტი, რაც მოსახლეობაში არჩევნებისადმი ნდობის გაჩენას განაპირობებს. ეს უკანასკნელი კი წინაპირობაა იმისა, რომ მოქალაქე იმედით განმსჭვალული, რომ მისი ხმა არ დაიკარგება, მივა არჩევნებზე თავისი კონსტიტუციური უფლების განსახორციელებლად და ხმას მისცემს მისთვის სასურველ კანდიდატს.

V თავი
კლიენტ-სერვერ და სერვისორიენტირებული არქიტექტურების პროგრამული რეალიზაცია

5.1. კომუნიკაცია: WCF - ტექნოლოგია

ბიზნესპროცესების შესრულებისას ერთ-ერთი მნიშვნელოვანი ასპექტია ურთიერთობა (კომუნიკაცია) აპლიკაციებს შორის, კლიენტებსა და სერვერებს შორის, აგრეთვე მუშაპროცესებსა და ჰოსტ-დანართებს შორის. ამ ნაწილში გავეცნობით, თუ როგორ შეიძლება ბიზნესპროცესების გამოყენებით გამარტივდეს და კოორდინაცია გაეწიოს კომუნიკაციათა სხვადასხვა სცენარებს.

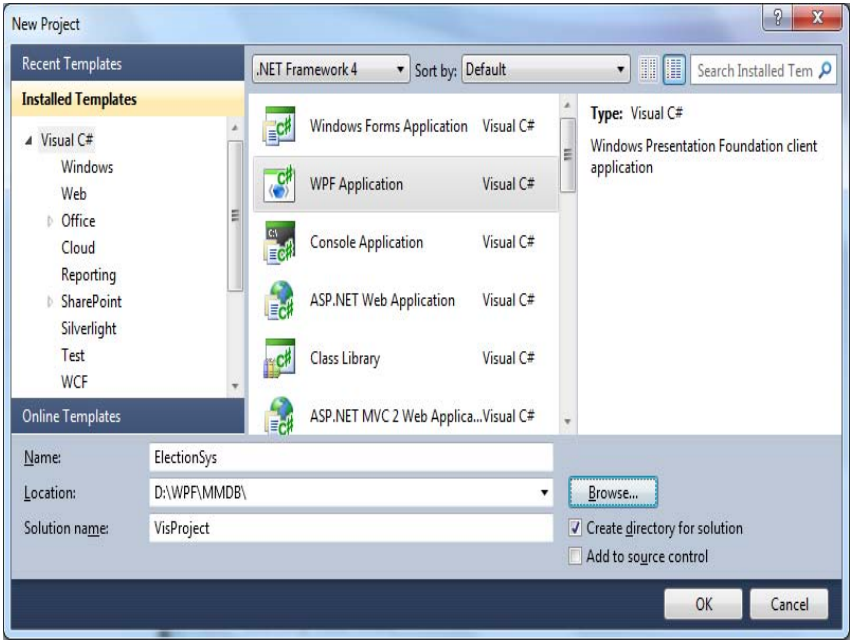
აპლიკაციის მაგალითის სახით განიხილება პროექტის აგება საარჩევნო სისტემისათვის, რომელშიც მოთხოვნილი ინფორმაცია (მაგალითად, ამომრჩევლის შესახებ) გადმოეცემა სხვა ფილიალიდან. იმავე აპლიკაციას შეუძლია მოთხოვნის გაგზავნა სხვა ფილიალში და ასევე პასუხის გაცემა სხვა ფილიალების მოთხოვნებზე.

მთავარი ქმედებები, რომლებიც კომუნიკაციისთვის გამოიყენება, არის Send და Receive ქმედებები (და მათი ვარიაციები: SendReply და ReceiveReply). ეს ქმედებები გამოიყენებს Windows Communication Foundation (WCF) ტექნოლოგიას შეტყობინებათა გადასაცემად და სამეთვალყურეოდ [86, 87].

ამ თავში ავაგებთ მარტივ WPF-აპლიკაციას (Windows Presentation Foundation), რომელიც გამოიყენებს კომუნიკაციას სხვადასხვა აპლიკაციათა ბიზნესპროცესებს შორის. მაგალითად, ეს შეიძლება იყოს ცენტრალური საარჩევნო კომისიის ოფისი (ცენტრი) და რეგიონალური ან უბნის საარჩევნო ოფისები (ფილიალები).

5.1.1. WPF პროექტის შექმნა

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



ნახ.5.1. WPF აპლიკაციის შექმნა

2. Solution Explorer-ში ElectionSys პროექტზე მაუსის მარჯვენა ღილაკით ავირჩიოთ Add Reference და .NET ცხრილიდან დავამატოთ შემდეგი კავშირები:

- System.Activities
- System.Configuration
- System.ServiceModel
- System.ServiceModel.Activities

3. Solution Explorer-ში გენერირდება ვინდოუსის ფაილი სახელით Window1.xaml, რომელიც შევცვალოთ Election.xaml - ით.

App.xaml ფაილი განსაზღვრავს ვინდოუსის startup-ს. ესაა ახლა Window1.xaml ფაილი, რომელიც უნდა შევცვალოთ ასე:

StartupUri=" Election.xaml"

5.1.2. ახალი config-ფაილის და Class-ების შექმნა

რამდენიმე აპლიკაციის კონფიგურირების მიზნით (საარჩევნო ფილიალების რაოდენობის შესაბამისად) შეიქმნება App.config ფაილების ასლები, რომლებშიც მოთავსებულ იქნება შესაბამისი ფილიალის ფიზიკური მონაცემები.

ElectionSys-პროექტის Solution Explorer-დან ვირჩევთ Add New Item ➤. დიალოგურ ფანჯარაში General ჯგუფში ვირჩევთ Application Configuration File. ფაილის სახელი ავტომატურად არის App.config(). კონფიგურაციის ფაილში შევიტანთ საჭირო მონაცემებს 5.1 ლისტინგის მსგავსად.

```
<!— ლისტინგი_.5.1 ———— –>
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Branch Name" value="Regioni GURIA"/>
    <add key="ID" value="{43E6DADD-4751-4056-8BB7-
                                    7459B5C361AB}"/>
    <add key="Address" value="8000"/>
    <add key="Request Address" value="8730"/>
  </appSettings>
</configuration>
```

AppSettings სექციას აქვს მნიშვნელობები ფილიალის სახელი, ID (უნიკალური იდენტიფიკატორი) და მისამართი (პორტის ნომერი, რომელსაც აპლიკაცია იყენებს). მოთხოვნის მისამართი განსაზღვრავს პორტის ნომერს, საითაც იქნება მოთხოვნები

გაგზავნილი. შესაძლებელია სხვა პორტების გამოყენებაც, თუ ეს აუცილებელი იქნება.

შემდეგ საჭიროა შეიქმნას კლასი, რომელიც განსაზღვრავს შეტყობინებებს აპლიკაციებს შორის. Solution Explorer-იდან Add -> Class და ჩავწეროთ კლასის სახელი Election.cs. ამ ფაილში დავამატოთ სახელსივრცეები (namespaces):

```
using System.Runtime.Serialization;
using System.ServiceModel;
```

Election.cs ფაილში განვსაზღვროთ სამი კლასი:

- Branch: განსაზღვრავს მონაცემებს საარჩევნო სისტემის ფილიალის მდებარეობის შესახებ;
- ElectionRequest: განსაზღვრავს ფილიალის მოთხოვნას;
- ElectionResponse: განსაზღვრავს პასუხს მოთხოვნის შესაბამის ფილიალისთვის.

5.2 ლისტინგში მოცემულია შესაბამისი კოდი:

```
//--- ლისტინგი 5.2 --- Election.cs ------
using System;
using System.Runtime.Serialization;
using System.ServiceModel;

namespace ElectionSys
{    /*********************************************/
    // ფილიალის მონაცემთა სტრუქტურის განსაზღვრა
    /*********************************************/
    public class Branch
    {
        public String BranchName { get; set; }
        public String Address { get; set; }
        public Guid BranchID { get; set; }

        #region Constructors
        public Branch() { }
        public Branch(String name, String address)
        {
            BranchName = name;
```

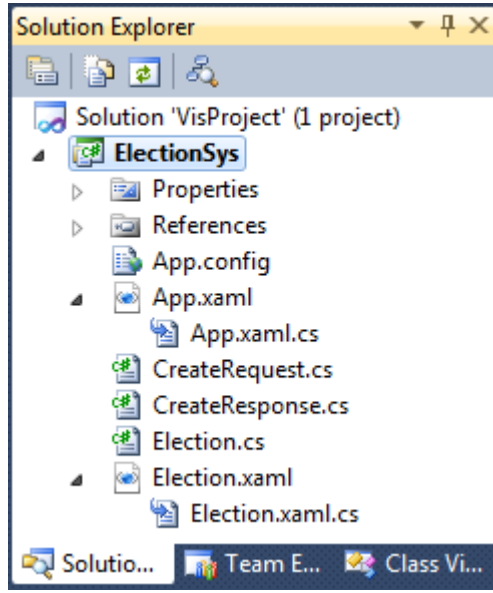
```
            Address = address;
            BranchID = Guid.NewGuid();
        }
        public Branch(String name, String address, Guid id)
        {
            BranchName = name;
            Address = address;
            BranchID = id;
        }
       public Branch(String name, String address, String id)
        {
            BranchName = name;
            Address = address;
            BranchID = new Guid(id);
        }
        #endregion Constructors
    }
    /*****************************************************/
    // მოთხოვნის შეტყობინების განსაზღვრა, ElectionRequest
    /*****************************************************/
    [MessageContract(IsWrapped = false)]
    public class ElectionRequest
    {
        private String _Region;
        private String _ArealName;
        private String _MajorDeputy;
        private Guid _RequestID;
        private Branch _Requester;
        private Guid _InstanceID;

        #region Constructors
        public ElectionRequest() { }
public ElectionRequest(String arealname, String majordeputy,
     String region, Branch requester)
        {
            _ArealName = arealname;
            _MajorDeputy = majordeputy;
            _Region = region;
            _Requester = requester;
            _RequestID = Guid.NewGuid();
        }
public ElectionRequest(String arealname, String majordeputy,
     String region, Branch requester, Guid id)
```

```
{
    _ArealName = arealname;
    _MajorDeputy = majordeputy;
    _Region = region;
    _Requester = requester;
    _RequestID = id;
}
#endregion Constructors
#region Public Properties
[MessageBodyMember]
public String Title
 {  get { return _ArealName }
    set { _ArealName = value; }
 }
[MessageBodyMember]
public String ISBN
 {
    get { return _Region; }
    set { _Region = value; }
 }
[MessageBodyMember]
public String Author
 {
    get { return _MajorDeputy; }
    set { _MajorDeputy = value; }
 }
[MessageBodyMember]
public Guid RequestID
 {
    get { return _RequestID; }
    set { _RequestID = value; }
 }
[MessageBodyMember]
public Branch Requester
 {
    get { return _Requester; }
    set { _Requester = value; }
 }
[MessageBodyMember]
public Guid InstanceID
 {
    get { return _InstanceID; }
    set { _InstanceID = value; }
```

```
        }
        #endregion Public Properties
    }
  /*****************************************************/
 // მოთხოვნის შეტყობინების განსაზღვრა:ElectionResponse
  /*****************************************************/
    [MessageContract(IsWrapped = false)]
    public class ElectionResponse
    {
        private bool _Reserved;
        private Branch _Provider;
        private Guid _RequestID;

        #region Constructors
        public ElectionResponse() {  }
public ElectionResponse(ElectionRequest request, bool
      reserved,  Branch provider)
         {
            _RequestID = request.RequestID;
            _Reserved = reserved;
            _Provider = provider;
         }
        #endregion Constructors

        #region Public Properties
       [MessageBodyMember]
       public bool Reserved
         {
            get { return _Reserved; }
            set { _Reserved = value; }
         }

       [MessageBodyMember]
       public Branch Provider
         {
            get { return _Provider; }
            set { _Provider = value; }
         }
       [MessageBodyMember]
       public Guid RequestID
         {
            get { return _RequestID; }
            set { _RequestID = value; }
```

```
        }
        #endregion Public Properties
    }
    /*****************************************************/
    // სერვისის კონტრაქტის განსაზღვრა, IElectionSys
    // რომელიც ორი მეთოდისგან შედგება: RequestElinfo() და
    // RespondToRequest()
    /*****************************************************/
    [ServiceContract]
    public interface IElectionSys
     {
        [OperationContract(IsOneWay = true)]
        void RequestElinfo(ElectionRequest request);

        [OperationContract(IsOneWay = true)]
        void RespondToRequest(ElectionResponse response);
     }
}
```

შედეგად, Solution Explorer-ს ექნება ასეთი სახე (ნახ.5.2).



ნახ.5.2

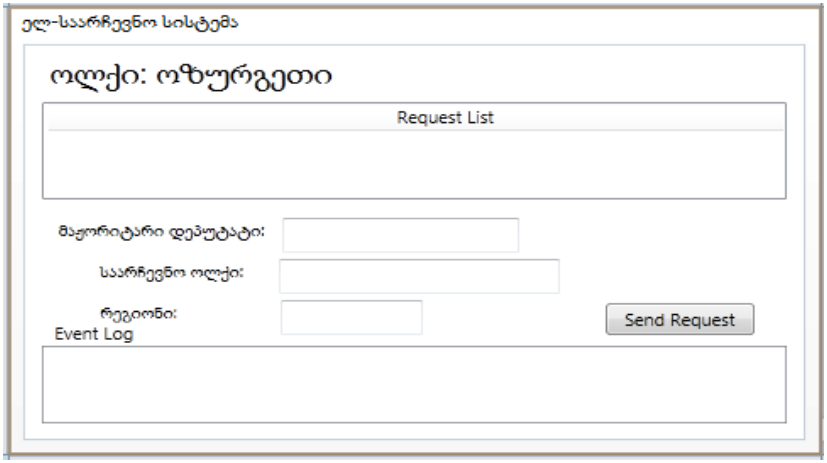
5.1.3. WindowForm-ის განსაზღვრა

გავხსნათ Election.xaml ფაილი და ავირჩიოთ XAML tab. ჩავანაცვლოთ კოდი შემდეგი 5.3 ლისტინგის ტექსტით:

```
<!-- ლისტინგი_5.3 --- ->
<Window x:Class="ElectionSys.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml
        /presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="ელ-საარჩევნო სისტემა" Height="480" Width="650"
    Loaded="Window_Loaded" Unloaded="Window_Unloaded">
 <Grid>
    <Label Height="40" HorizontalAlignment="Left"
       Margin="12,0,0,0" Name="lblBranch" FontSize="22"
       VerticalAlignment="Top" Width="276"
       FontStretch="Expanded">ოლქი: ოზურგეთი</Label>
    <ListView x:Name="requestList" Margin="12,42,12,5"
       Height="150" VerticalAlignment="Top"
       ItemsSource="{Binding}"
       SelectionChanged="requestList_SelectionChanged">
    <ListView.View>
     <GridView>
       <GridViewColumn Header="Request List" Width="610">
         <GridViewColumn.CellTemplate>
           <DataTemplate>
             <StackPanel Orientation="Horizontal">
               <TextBlock Text="{Binding
                     Requester.BranchName}" Width="100"/>
          <TextBlock Text="{Binding MajorDeputy}" Width="95"/>
          <TextBlock Text="{Binding ArealName}" Width="180"/>
               <TextBlock Text="{Binding Region}" Width="90"/>
               <Button Content="Reserve" Tag="{Binding
                 InstanceID}" Click="Reserve" Width="65"/>
               <Button Content="Cancel" Tag="{Binding
                 InstanceID}" Click="Cancel" Width="60"/>
              </StackPanel>
             </DataTemplate>
           </GridViewColumn.CellTemplate>
         </GridViewColumn>
        </GridView>
       </ListView.View>
```

```
        </ListView>
       <Label Height="30" Margin="15,0,0,210" Name="label5"
              VerticalAlignment="Bottom"
              HorizontalAlignment="Left" Width="148"
               HorizontalContentAlignment=
                   "Right">მაჟორიტარი დეპუტატი:</Label>
    <Label Height="30" Margin="34,0,479,180" Name="label2"
             VerticalAlignment="Bottom"
             HorizontalContentAlignment=
                 "Right">საარჩევნო ოლქი:</Label>
    <Label Height="30" Margin="45,25,0,150" Name="label3"
              VerticalAlignment="Bottom"
              HorizontalAlignment="Left" Width="60"
              HorizontalContentAlignment=
                  "Right">რეგიონი:</Label>
    <TextBox Height="25" Margin="169,0,0,210"
              Name="txtMajorDeputy"
              VerticalAlignment="Bottom"
              HorizontalAlignment="Left" Width="200" />
    <TextBox Height="25" Margin="0,0,161,180"
              Name="txtArealName"
              VerticalAlignment="Bottom"
              HorizontalAlignment="Right" Width="300" />
    <TextBox Height="25" Margin="168,0,0,150"
              Name="txtRegion"
              VerticalAlignment="Bottom"
              HorizontalAlignment="Left" Width="100" />
    <Button Height="23" Margin="497,0,0,150"
              Name="btnRequest"
              VerticalAlignment="Bottom"
              HorizontalAlignment="Left" Width="98"
           Click="btnRequest_Click">Send Request</Button>
    <Label Height="27" HorizontalAlignment="Left"
           Margin="15,0,0,137" Name="label4"
           VerticalAlignment="Bottom"
           Width="76">Event Log</Label>
    <ListBox Margin="12,0,12,12" Name="lstEvents"
           Height="130" VerticalAlignment="Bottom"
           FontStretch="Condensed" FontSize="10" />
    </Grid>
</Window>
```

შემდეგ ავირჩიოთ Design Tab. ფორმას უნდა ჰქონდეს შემდეგი სახე (ნახ.5.3).



ნახ.5.3. „საარჩევნო უბანი“

ფორმის ზედა ნაწილში მოთხოვნების სია (RequestList) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებს. მოთხოვნის გასაგზავნად რეგიონალურ ცენტრში გამოიყენება ველები ფორმის შუაში. აქ მიეთითება მაგალითად, მაჟორიტარი დეპუტატის გვარი ს., საარჩევნო ოლქი და რეგიონი, შემდეგ გაგზავნის ღილაკი „მოთხოვნის გაგზავნა“ (Send Request).

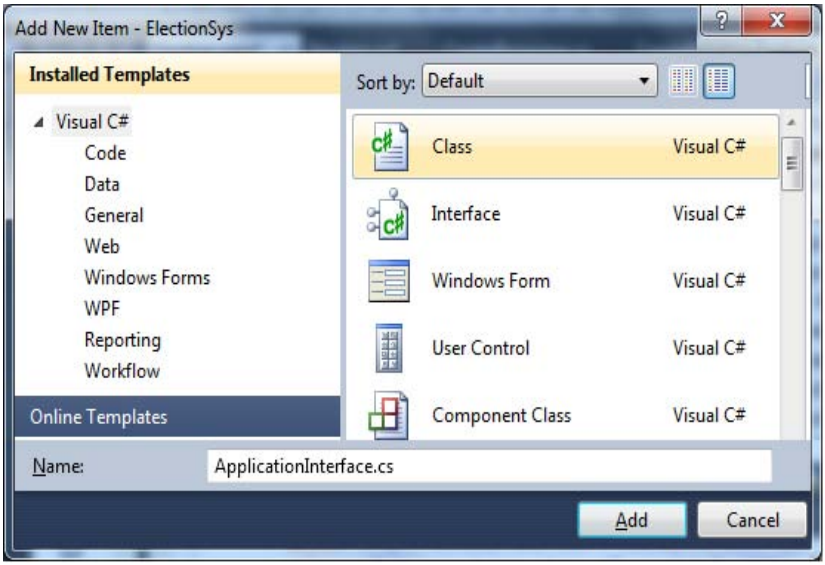
„მოვლენათა რეგისტრაცია“ (Event Log) ქვედა მარცხენა კუთხეში ასახავს ბიზნესპროცესების შეტყობინებებს.

5.1.4. ტექსტის ჩამწერის რეალიზაცია (TextWriter)

WriteLine ქმედებისთვის, რომელიც გამოიყენება ტექსტის გამოსატანად ეკრანზე, არ დაგვიყენებია თვისება TextWriter. კონსოლის რეჟიმში ტექსტი ავტომატურად გამოიტანება, ფორმაზე გამოსატანად კი უნდა გავეცნოთ TextWriter კლასს.

5.1.5. აპლიკაციის სტატიკური მიმთითებლის უზრუნველყოფა

თავიდან უნდა **შეიქმნას სტატიკური კლასი,** რომელიც უზრუნველყოფს აპლიკაციის ფანჯარასთან წვდომას. Solution Explorer-ში ElectionSys–ზე მაუსის მარჯვენა ღილაკით ვირჩევთ Add -> Class (ნახ.5.4). კლასის სახელია ApplicationInterface.cs, რომლის პროგრამული რეალიზაცია მოცემულია 5.4 ლისტინგში.



ნახ.5.4.

```
// – – ლისტინგი_5.4 – – ApplicationInterface.cs  – –
using System;
using System.Windows.Controls;
using System.Activities;
namespace ElectionSys
{
    public static class ApplicationInterface
    {
        public static MainWindow _app { get; set; }
        public static void AddEvent(String status)
```

```
        {
            if (_app != null)
            {
                new
ListBoxTextWriter(_app.GetEventListBox()).WriteLine(status);
            }
        }
  public static void RequestElinfo(ElectionRequest request)
     {
        if (_app != null)
            _app.RequestElinfo(request);
     }
public static void RespondToRequest(ElectionResponse response)
     {
        if (_app != null)
            _app.RespondToRequest(response);
     }
 public static void NewRequest(ElectionRequest request)
     {
        if (_app != null)
            _app.AddNewRequest(request);
     }
   }
}
```

ApplicationInterface კლასს აქვს სტატიკური მიმთითებელი (_app) აპლიკაციის ფანჯარაზე (MainWindow კლასი). სტატიკური AddEvent() მეთოდი ქმნის ListBoxTextWriter კლასის ეგზემპლარს, რომელიც შემდგომში იქნება რეალიზებული და იძახებს მის WriteLine() მეთოდს.

გავხსნათ Election.xaml.cs ფაილი და დავამატოთ შემდეგი სახელსივრცეები:

```
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
using System.ServiceModel.Description;
using System.ServiceModel.Channels;
using System.Activities;
```

```
using System.Xml.Linq;
using System.Configuration;
```

დავამატოთ `MainWindow()`-ში კონსტრუქტორის შემდეგი კოდი:

```
ApplicationInterface._app = this;
```

აქ this აინიციალებს _app მიმართვას ApplicationInterface კლასში. ვინაიდან იგი სტატიკური კლასია, მასში იქნება მხოლოდ ერთი ეგზემპლარი, რომელსაც ექნება მიმართვა MainWindow კლასზე. დავამატოთ შემდეგი მეთოდები Election.xaml.cs ფაილში:

```
public ListBox GetEventListBox()
  {
    return this.lstEvents;
  }
private void AddEvent(string szText)
  {
     lstEvents.Items.Add(szText);
  }
```

GetEventListBox() მეთოდი აბრუნებს უკან მიმთითებელს ListBox–ის ფაქტობრივი კონტროლისთვის, რომელმაც უნდა ასახოს ეს მოვლენები. ამ მეთოდს იყენებს ApplicationInterface კლასი. AddEvent () მეთოდს იყენებს აპლიკაცია მაშინ, როცა მას სჭირდება მოვლენის დამატება.

5.1.6. ListBoxTextWriter–ის რეალიზაცია

დავამატოთ პროექტს კლასი ListBoxTextWriter.cs, რომლის რეალიზაცია 5.5 ლისტინგშია მოცემული.

```
// – – ლისტინგი_5.5 --- ListBoxTextWriter.cs ---
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.IO;
using System.Windows.Controls;
namespace ElectionSys
{
  public class ListBoxTextWriter : TextWriter
   {
     const string textClosed = "This TextWriter must be opened
                               before use";
     private Encoding _encoding;
     private bool _isOpen = false;
     private ListBox _listBox;

     public ListBoxTextWriter()
      {
        // Get the static list box
       _listBox = ApplicationInterface._app.GetEventListBox();
        if (_listBox != null)
             _isOpen = true;
      }
    public ListBoxTextWriter(ListBox listBox)
      {
         this._listBox = listBox;
         this._isOpen = true;
      }
    public override Encoding Encoding
      {
        get
         {
           if (_encoding == null)
             {
              _encoding = new UnicodeEncoding(false, false);
             }
           return _encoding;
         }
      }
    public override void Close()
      {
         this.Dispose(true);
      }
    protected override void Dispose(bool disposing)
      {
```

```
            this._isOpen = false;
            base.Dispose(disposing);
        }

        public override void Write(char value)
        {
            if (!this._isOpen)
                throw new ApplicationException(textClosed); ;

            this._listBox.Dispatcher.BeginInvoke
                 (new Action(() =>
                    this._listBox.Items.Add(value.ToString())));
        }
    public override void Write(string value)
        {
          if (!this._isOpen)
              throw new ApplicationException(textClosed);
                                    ;
          if (value != null)
            this._listBox.Dispatcher.BeginInvoke
                  (new Action(() =>
                      this._listBox.Items.Add(value)));
        }
    public override void Write(char[] buffer, int index,
                                                    int count)
        {
          String toAdd = "";
          if (!this._isOpen)
             throw new ApplicationException(textClosed); ;

          if (buffer == null || index < 0 || count < 0)
             throw new ArgumentOutOfRangeException("buffer");

        if ((buffer.Length - index) < count)
          throw new ArgumentException("The buffer is too small");

        for (int i = 0; i < count; i++)
               toAdd += buffer[i];
        this._listBox.Dispatcher.BeginInvoke
           (new Action(() => this._listBox.Items.Add(toAdd)));
        }
    }
}
```

ListBoxTextWriter კლასი არის მიღებული აბსტრაქტული TextWriter კლასიდან და უზრუნველყოფს Write() მეთოდის განხორციელებას, რომელიც ამატებს სტრიქონს ListBox-ში (თუ გსურთ განახორციელოთ Write() მეთოდის სამი გადატვირთვა, იმისთვის რომ იყოს მიღებული, როგორც char ან string ან char [] მასივი).

არსებული კონსტრუქტორი იყენებს სტატიკურ ApplicationInterface კლასს MainWindow-ის lstEvents კონტროლის მისაღებად. იგი ასევე უზრუნველყოფს კონსტრუქტორს, რომელშიც ListBox შეიძლება შესრულდეს. ეს კონსტრუქტორი გამოიყენება ApplicationInterface კლასის AddEvent () მეთოდით.

ListBox-ის Add() მეთოდი ხორციელდება აპლიკაციის შესრულების ნაკადის (thread) შემდეგაც. იგი აკეთებს ამას Dispatcher-ის BeginInvoke() მეთოდის გამოყენებით, რომელიც ასოცირდება lstEvents მართვის ელემენტთან. ეს საშუალებას აძლევს მეთოდს იმუშაოს სხვადასხვა ნაკადებიდან გამოძახების დროსაც.

იმის გამო, რომ ListBoxTextWriter კლასი არის მიღებული TextWriter-დან, შეიძლება მისი მითითება, როგორც TextWriter თვისებისა ნებისმიერი WriteLine ქმედებისთვის. და სტატიკური ApplicationInterface კლასის გამო, ListBoxTextWriter კლასს შეუძლია წვდომა lstEvents ელემენტზე აპლიკაციის გარედანაც კი.

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

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

5.1.7. მუშა პროცესების რეალიზაცია

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

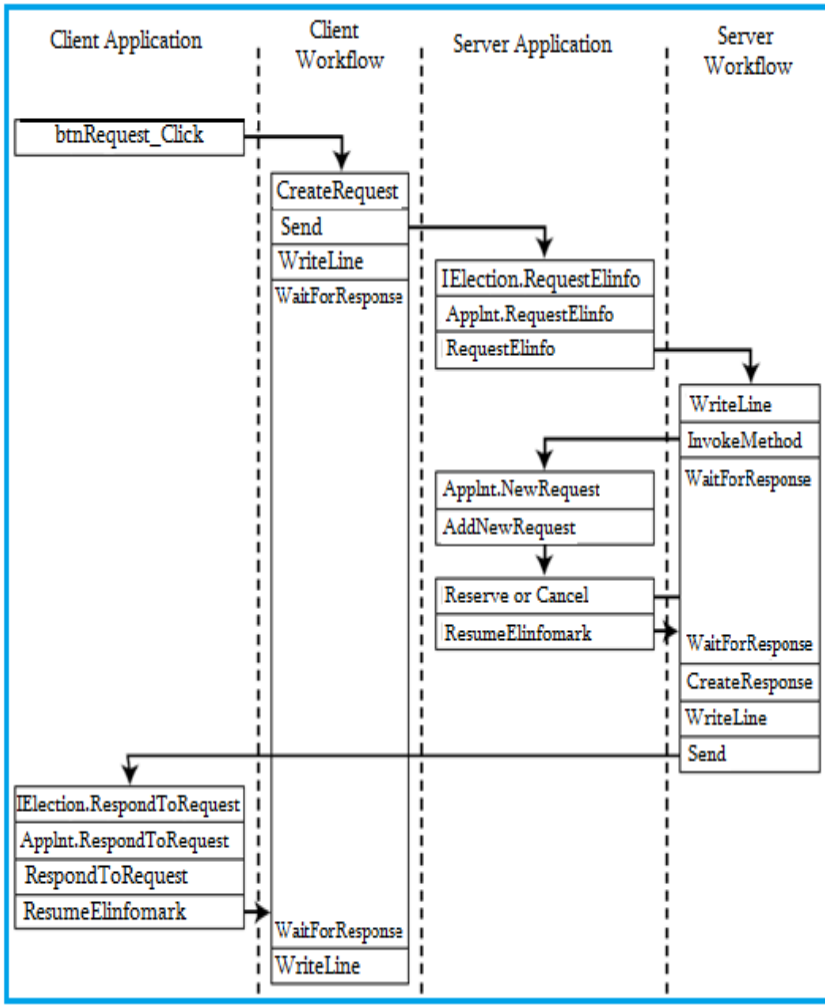
არ არსებობს არავითარი Receive ქმედება. მის მაგივრად აპლიკაცია მიიღებს შემავალ შეტყობინებებს, შემდეგ კი გამოიძახებს (ან აღადგენს) მუშა პროცესს.

5.1.8. შეტყობინებათა მიღება

5.5 ნახაზზე სერვერული აპლიკაცია იღებს შეტყობინებას და მასთან დაკავშირებული ელემენტი დიაგრამეზე არის ლებელი IElection.RequestElinfo. გარდა ამისა, კლიენტის აპლიკაცია იღებს შეტყობინებას და ელემენტი IElection.RespondToRequest-ით არის აღნიშნული. ესაა სერვისული კონტრაქტის მეთოდები.

გავხსნათ Election.cs ფაილი, რომელშიც გამოჩნდება ინტერფეისის შემდეგი განსაზღვრება:

```
[ServiceContract]
public interface IElectionSys
{
    [OperationContract]
    void Elinfo(ElectionRequest request);

    [OperationContract]
    void RespondToRequest(ElectionResponse response);
}
```


Client Application
Client Workflow
Server Application
Server Workflow
btnRequest_Click
CreateRequest
Send
WriteLine
WaitForResponse
IElection.RequestElinfo
AppInt.RequestElinfo
RequestElinfo
WriteLine
InvokeMethod
WaitForResponse
AppInt.NewRequest
AddNewRequest
Reserve or Cancel
ResumeElinfomark
WaitForResponse
CreateResponse
WriteLine
Send
IElection.RespondToRequest
AppInt.RespondToRequest
RespondToRequest
ResumeElinfomark
WaitForResponse
WriteLine

ნახ.5.5

საჭიროა მცირე ცვლილების ჩატარება. კერძოდ, OperationContract-ს უნდა დაემატოს (IsOneWay = true). ქვემოთ ნაჩვენებია ეს:

```
[ServiceContract]
public interface IElectionSys
 {
   [OperationContract(IsOneWay = true)]
   void RequestElinfo(ElectionRequest request);

   [OperationContract(IsOneWay = true)]
   void RespondToRequest(ElectionResponse response);
 }
```

შეტყობინება იგზავნება ბიზნესპროცესთან ერთად, მაგრამ პასუხი მიიღება ServiceHost-ით აპლიკაციის შიგნით. ასე, რომ ეს არაა ტქნიკური ორმხრივი საუბარი. არსებობს შეტყობინებები ორივე მიმართულებით. რადგან გაგზავნის და მიღების საბოლოო წერტილები სხვადასხვაა, WCF ამას აფიქსირებს როგორც ცალკე ერთმიმართულებიანი შეტყობინებები.

5.1.9. სერვისის კონტრაქტის რეალიზაცია

სერვისის კონტრაქტი განსაზღვრავს მხოლოდ ხელმისაწვდომ მეთოდებს, იგი არ უზრუნველყოფს მათ იმპლემენტაციას (რეალიზაციას).

ჩვენი პროექტისთვის აუცილებელია ამ საკითხის გადაწყვეტა. ამიტომაც, Solution Explorer-ში ElectionSys -ზე მარჯვენა ღილაკით ვირჩევთ Add Class. მივუთითებთ კლასის სახელს ClientService.cs. მისი კოდის რეალიზაცია ნაჩვენებია 5.6 ლისტინგში.

```
// ===== ლისტინგი 5.6 ==== ClientService.cs =====
using System;
using System.ServiceModel;

namespace ElectionSys
{
```

```
public class ClientService : IElectionSys
    {
   public void RequestElinfo(ElectionRequest request)
       {
          ApplicationInterface.RequestElinfo(request);
       }
   public void RespondToRequest(ElectionResponse response)
       {
          ApplicationInterface.RespondToRequest(response);
       }
    }
}
```

ეს რეალიზაცია იყენებს ApplicationInterface სტატიკურ კლასს, რომელიც უკვე შექმნილია ჩვენ მიერ. ყოველი მეთოდი უბრალოდ იძახებს ApplicationInterface კლასის შესაბამის მეთოდს. გავხსნათ ApplicationInterface.cs ფაილი და დავამატოთ შემდეგი მეთოდები:

```
public static void RequestElinfo(ElectionRequest request)
  {
    if (_app != null)
        _app.RequestElinfo(request);
  }

public static void  RespondToRequest(ElectionResponse
                                                    response)
  {
     if (_app != null)
         _app.RespondToRequest(response);
   }
```

ეს მეთოდები, თავის მხრივ, იძახებს შესაბამის მეთოდებს აპლიკაციაში სტატიკური მიმთითებლის გამოყენებით. საჭირო ინება ამ მეთოდების რეალიზება Election.xaml.cs ფაილში, რასაც მოგვიანებით დავუბრუნდებით.

5.1.10. ServiceHost-ის რეალიზაცია

აპლიკაციისთვის აუცილებელია ServiceHost-ის რეალიზაცია შემავალი შეტყობინებების მისაღებად (მოსასმენად). გავხსნათ Election.xaml.cs ფაილი და დავამატოთ შემდეგი კლასის წევრები:

```
private ServiceHost _sh;
```

იგი უნდა მოთავსდეს კონსტრუქტორის წინ. ასე:

```
public partial class MainWindow : Window
  {
      private ServiceHost _sh;

      public MainWindow()
      {
          InitializeComponent();
          ApplicationInterface._app = this;
      }
```

ServiceHost იწყება მაშინ, როცა ფანჯარა ჩატვირთულია, და იხურება, როცა ფანჯარა ამოტვირთულია. მეთოდების დამატება ნაჩვენებია 5.7 ლისტინგში MainWindow კლასისთვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

```
// -- ლისტინგი 5.7 --- The Loaded and Unloaded Event Handlers ----
private void Window_Loaded(object sender, RoutedEventArgs e)
{
  // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და
  //  მისი ქსელური მისამართი
Configuration config =   ConfigurationManager.OpenExeConfiguration(
                                ConfigurationUserLevel.None);
AppSettingsSection app =
        (AppSettingsSection)config.GetSection("appSettings");
  string adr = app.Settings["Address"].Value;
```

```
// ფილიალის სახელის გამოტანა ფორმაზე
lblBranch.Content = app.Settings["Branch Name"].Value;
  // ServiceHost-ის შექმნა
 _sh = new ServiceHost(typeof(ClientService));
  // დასასრულის წერტილის (Endpoint) დამატება
  string szAddress = "http://localhost:" + adr + "/ClientService";
 System.ServiceModel.Channels.Binding bBinding =
                                                  new BasicHttpBinding();
 _sh.AddServiceEndpoint(typeof(ILibraryReservation),bBinding, szAddress);
// ServiceHost-ის გახსნა შეტყობინებების მისაღებად (listen)
 _sh.Open();
// ListBoxTextWriter -ის ტესტირება
//ListBoxTextWriter lbtw = new ListBoxTextWriter();
//lbtw.Write("ეს არის ტესტი - This is a test");
}
private void Window_Unloaded(object sender, RoutedEventArgs e)
 {
   // service host-ის დატოვება
    _sh.Close();
 }
```

მოვლენის დამმუშავებელი Loaded ხსნის კონფიგურაციის ფაილს და ათავსებს საარჩევნო უბნის (ფილიალის) სახელს lblBranch - მართვის ელემენტში, ამიტომაც ფორმა ასახავს ლოკალური ფილიალის სახელს.

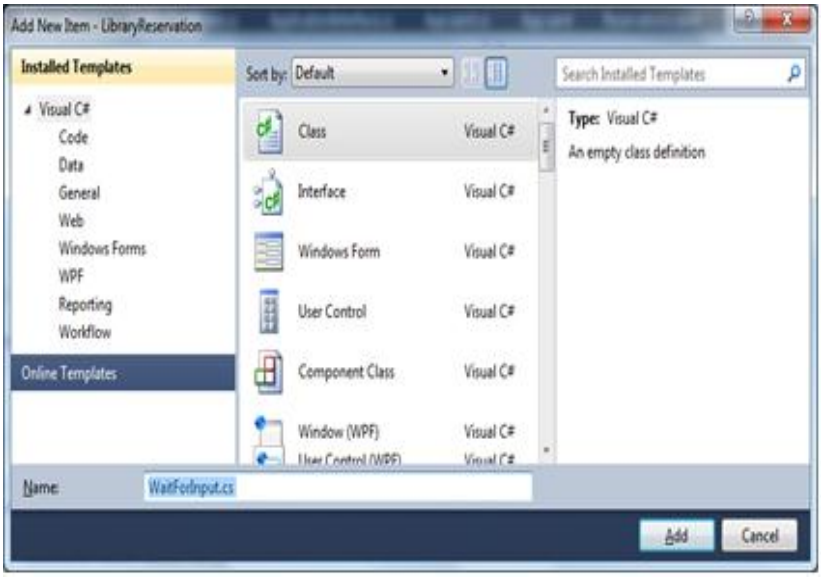
შემდეგ იქმნება ServiceHost თანამგზავრი (passing) ClientService კლასისა, რომელიც ახლახანს შევქმენით როგორც მისი რეალიზაცია. შემდეგ იგი აკონფიგურირებს დასასრულის წერტილს ServiceHost-თვის, იყენებს რა ცნობილი მისამართის, მიბმის და კონტრაქტის სამეულს.

Unloaded მოვლენის დამმუშავებელი უბრალოდ ხურავს ServiceHost-ს, ასე რომ მეტი შეტყობინება აღარ მიიღება.

5.1.11. სანიშნეები (Bookmarks)

სანიშნეები იძლევა საშუალებას შეაჩეროს მუშა პროცესი და შეინახოს მარკერი ისე, რომ შემდგომ შეიძლებოდეს მისი იმავე წერტილიდან განახლება. ისინი დამუშავებულია მონაცემთა მისაღებად შემდგომი აღდგენის მიზნით. ამ პროექტში, მაგალითად, როცა მოთხოვნა მიღებულია, აპლიკაციას გამოაქვს ეკრანზე ეს მოთხოვნა და ელოდება მომხმარებლისგან პასუხის (კი/არა) შეტანას. შემდეგ მუშა პროცესი გრძელდება ამ პასუხის გავლით.

სანიშნეები იქმნება მომხმარებელთა აქტიურობით. აქ ჩვენ შევქმნით ზოგად ქმედებას, რომელიც შემდგომში გამოყენებადი იქნება ყველგან, სადაც სანიშნე მოითხოვს. LibraryReservation-ზე Solution Explorer-ში, მარჯვენა ღილაკით დავამატოთ WaitForInput.cs კლასი, რომლის 5.6 ლისტინგი მოცემულია ქვემოთ.



ნახ.5.6. WaitForInput.cs კლასის შექმნა

```
// ---- ლისტინგი 5.8  ----- WaitForInput.cs ---------------
using System;
using System.Activities;

namespace ElectionSys
{
  public sealed class WaitForInput<T> : NativeActivity<T>
   {
     public WaitForInput() : base()
      {
      }

      public string ElinfomarkName { get; set; }
      public OutArgument<T> Input { get; set; }

  protected override void Execute(NativeActivityContext
                context)
      {
        context.CreateBookmark(ElinfomarkName,
             new BookmarkCallback(this.Continue));
      }
     void Continue(NativeActivityContext context,
           Bookmark Elinfomark, object obj)
        {
          Input.Set(context, (T)obj);
        }
     protected override bool CanInduceIdle { get {
                     return true; } }
    }
}
```

მომხმარებლის ეს ქმედება გამოიყენებს NativeActivity საბაზო კლასს (CodeActivity-ის ნაცვლად), იმიტომ რომ იგი აძლევს მას NativeActivityContext-თან მიმართვის საშუალებას, რომელიც აუცილებელია სანიშნის შესაქმნელად. იგი ასევე იყენებს შაბლონის ვერსიას (კლასში <T> აღნიშვნაა). შემავალი არგუმენტი ასახავს მონაცემებს, რომლებიც გადაეცემა ბიზნესპროცესს, როცა ის განახლდება. შაბლონური ვერსიის დახმარებით ეს ქმედება

შესაძლებელია გამოყენებულ იქნას განმეორებით მონაცემთა ნებისმიერ ტიპთან.

Execute() მეთოდი იძახებს NativeActivityContext-ის CreateBookmark() მეთოდს, მიუთითებს რა სანიშნის სახელს და მიმთითებელს უკუგამოძახების მეთოდზე, სახელით Continue(). როდესაც მუშა პროცესი აღდგენილია, მაშინ ეს უკუგამოძახების მეთოდი სრულდება. ყურადსაღებია, რომ უკუგამოძახების მეთოდი ობიექტს ღებულობს მესამე პარამეტრის სახით. ესაა მონაცემები, წარმოდგენილი აპლიკაციით. იგი ინახება შემავალ არგუმენტში, რაც ხელმისაწვდომია ბიზნესპროცესისთვის.

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

5.1.12. SendRequest მუშა პროცესის რეალიზაცია

ახლა შევასრულოთ მუშა პროცესების რეალიზაცია. Solution Explorer-ის ElectionSys-ზე მარჯვენა ღილაკით ავირჩიოთ Add -> Class. სახელი ElectionWF.cs. კოდის რეალიზაცია მოცემულია 5.9 ლისტინგში.

```
// --- ლისტინგი 5.9 -- ElectionWF.cs ----
using System;
using System.Activities;
using System.Activities.Statements;
using System.ServiceModel.Activities;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Runtime.Serialization;
using System.Xml.Linq;
using System.IO;
```

```
namespace ElectionSys
{
  // ეს ფაილი შეიცავს ორი workflow-ის განსაზღვრას:
  // SendRequest - აინიციალიზირებს ახალ მოთხოვნებს,
  // ProcessRequest - ამუსავებს შემოსულ მოთხოვნებს
  public sealed class SendRequest : Activity
    {
     // შემავალი და გამომავალი არგუმენტების განსაზღვრა----
      public InArgument<string> ArealName { get; set; }
      public InArgument<string> MajorDeputy { get; set; }
      public InArgument<string> Region { get; set; }
      public InArgument<TextWriter> Writer { get; set; }
public OutArgument<ElectionResponse> Response {get; set;}
    public SendRequest()
    {
     // ცვლადების განსზღვრა ამ workflow -სთვის ---
     Variable<ElectionRequest> request =
       new Variable<ElectionRequest> { Name = "request" };
     Variable<string> requestAddress =
       new Variable<string> { Name = "RequestAddress" };
     Variable<bool> reserved = new Variable<bool> { Name =
         "Reserved" };

     // SendRequest workflow -ის განსაზღვრა ---
     this.Implementation = () => new Sequence
      {
        DisplayName = "SendRequest",
        Variables = { request, requestAddress, reserved },
        Activities =
          {
           new CreateRequest
            {
              ArealName = new InArgument<string>(env =>
                                 ArealName.Get(env)),
              MajorDeputy = new InArgument<string>(env =>
                                 MajorDeputy.Get(env)),
              Region = new InArgument<string>(env =>
                                 Region.Get(env)),
              Request = new OutArgument<ElectionRequest>
```

```
                (env => request.Get(env)),
            RequestAddress = new OutArgument<string>
                (env => requestAddress.Get(env))
        },
       new Send
        {
         OperationName = "RequestElinfo",
         ServiceContractName = "IElectionSys",
         Content = SendContent.Create
          (new InArgument<ElectionRequest>(request)),
            EndpointAddress = new InArgument<Uri>
              (env => new Uri("http://localhost:" +
        requestAddress.Get(env) + "/ClientService")),
          Endpoint = new Endpoint
           {
              Binding = new BasicHttpBinding()
            },
          },
    new WriteLine
     {
       Text = new InArgument<string>
        (env => "Request sent; waiting for response"),
       TextWriter = new InArgument<TextWriter> (env =>
            Writer.Get(env))
     },
   new WaitForInput<ElectionResponse>
     {
       ElinfomarkName = "GetResponse",
        Input = new OutArgument<ElectionResponse>
                        (env => Response.Get(env))
      },
   new WriteLine
     {
        Text = new InArgument<string>
       (env => "Response received from " +
       Response.Get(env).Provider.BranchName + " [" +
       Response.Get(env).Reserved.ToString() + "]"),
       TextWriter = new InArgument<TextWriter> (env =>
                Writer.Get(env))
      },
    }
```

```
        };
      }
    }
  // აქ უნდა დაემატოს წიგნში“ ლისტინგი 5.10  დასასრული“
}
```

უნდა აღვნიშნოთ, რომ ყოველ WriteLine ქმედებას აქვს დამატებითი თვისება:

TextWriter = new ListBoxTextWriter()

ის მიუთითებს იმაზე, რომ ახალი კლასი ListBoxTextWriter, რომელიც იქნა რეალიზებული, უნდა იქნას გამოყენებული ამ ტექსტის დისპლეიზე გამოსატანად. ეს გამოიწვევს ტექსტის ასახვას lstEvents მართვის ელემენტში.

სხვა განსხვავება იმაშია, რომ მომხმარებლის ქმედება WaitForInput გამოიყენება Receive ქმედების ნაცვლად. აპლიკაცია მიიღებს საპასუხო შეტყობინებას უშუალოდ (პირდაპირ). როცა მიიღება პასუხი, მაშინ აპლიკაცია აღადგენს მუშა პროცესს, რომელიც მიმდინარეობს ElectionResponse კლასში. ყურადსაღებია, რომ მომხმარებლის ქმედება განისაზღვრება როგორც WaitForInput <ElectionResponse>, მიუთითებს რა, რომ გადასაცემი მონაცემები იქნება ElectionResponse კლასის.

5.1.13. ProcessRequest მუშა პროცესის რეალიზაცია

ProcessRequest მუშა პროცესი განსაზღვრება მოცემულია 5.8 ლისტინგში. ჩავამატოთ ეს კოდი ElectionWF.cs ფაილში.

```
// ლისტინგი 5.10 =========
public sealed class ProcessRequest : Activity
 {
  public InArgument<ElectionRequest> request {get; set;}
  public InArgument<TextWriter> Writer { get; set; }
  public ProcessRequest()
   {
```

```
// ცვლადების განსზღვრა ამ  workflow -სთვის ---
Variable<ElectionResponse> response =
 new Variable<ElectionResponse> { Name = "response" };
Variable<bool> reserved = new Variable<bool> { Name =
                 "Reserved" };
Variable<string> address = new Variable<string> { Name
                = "Address" };
// ProcessRequest workflow-ს განსაზღვრა ---
this.Implementation = () => new Sequence
  {
    DisplayName = "ProcessRequest",
    Variables = { response, reserved, address },
    Activities =
     {
       new WriteLine
       {
         Text = new InArgument<string>(env => "Got
           request from: " +
           request.Get(env).Requester.BranchName),
          TextWriter = new InArgument<TextWriter> (env
          => Writer.Get(env))
       },
       new InvokeMethod
       {
        TargetType = typeof(ApplicationInterface),
           MethodName = "NewRequest",
             Parameters =
              {
                new InArgument<ElectionRequest>(env =>
                      request.Get(env))
              }
       },
       new WaitForInput<bool>
       {
         ElinfomarkName = "GetResponse",
          Input = new OutArgument<bool>(env =>
               reserved.Get(env))
       },
       new CreateResponse
       {
```

```
                Request = new InArgument<ElectionRequest>
                    (env => request.Get(env)),
                    Reserved = new InArgument<bool>(env =>
                        reserved.Get(env)),
                Response = new OutArgument<ElectionResponse>
                                (env => response.Get(env))
              },
            new WriteLine
              {
               Text = new InArgument<string>(env =>
                      "Sending response to: " +
                    request.Get(env).Requester.BranchName),
                    TextWriter = new InArgument<TextWriter>
                        (env => Writer.Get(env))
            },
          new Send
            {
             OperationName = "RespondToRequest",
               ServiceContractName = "ILibraryReservation",
             EndpointAddress = new InArgument<Uri>(
                   env => new Uri("http://localhost:" +
             request.Get(env).Requester.Address +
                 "/ClientService")),
                Endpoint = new Endpoint
                 {
                   Binding = new BasicHttpBinding()
                 },
               Content = SendContent.Create (new
                InArgument<ElectionResponse>(response))
              }
            }
          };
        }
    }
}
```

იმის მაგივრად, რომ დაწყება იყოს Receive ქმედებით, რათა მიღებულ იქნას შემავალი მოთხოვნა, ElectionRequest გადასცემს მუშა პროცესს შემავალი არგუმენტის გამოყენებით. WriteLine ქმედება, რომელიც მოსდევს მას, ცნობს შემავალ მოთხოვნას.

InvokeMethod ქმედება გამოვიყენოთ მონაცემთა გადასაცემად აპლიკაციაში. ApplicationInterface კლასი მოხერხებულადაა შესრულებული ამ მიზნით. იგი უზრუნველყოფს მუშა პროცესს, რათა განხორციელდეს გამოძახება აპლიკაციაში. InvokeMethod ქმედება იძახებს ApplicationInterface კლასის NewRequest() მეთოდს ElectionRequest კლასში გადასაცემად.

გავხსნათ ApplicationInterface.cs ფაილი და დავამატოთ მეთოდი, რომელიც უბრალოდ იძახებს AddNewRequest ()-ს აპლიკაციაში:

```
public static void NewRequest(ElectionRequest request)
    {   if (_app != null)
            _app.AddNewRequest(request);
    }
```

შემდეგი აქტიურობაა მომხმარებლის WaitForInput ქმედება, რომელიც გამოიყენებოდა SendRequest მუშა პროცესში.

ამჯერად იგი ელოდება Bool-შესატან მითითებას, იყო თუ არა დაჯავშნული დასახელება (სათაური). CreateResponse და WriteLine ქმედებები იგივეა. აქ გამოიყენებოდა SendReply ქმედება, ვინაიდან იგი იყო დაკავშირებული საწყის Receive ქმედებასთან.

ამ პროექტში, ვინაიდან არაა არავითარი Receive ქმედება, ჩვენ გამოვიყენებთ Send ქმედებას. საყურადღებოა, რომ EndpointAddress აწყობილია მისამართის გამოყენებით (პორტის ნომერი), რომელიც გათვალისწინებულია შესატან მოთხოვნაში.

5.1.14. აპლიკაციის რეალიზაცია

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

5.1.15. მუშა პროცესების ეგზემპლარების მხარდაჭერა

აპლიკაცია თვალყურს უნდა ადევნებდეს ბიზნესპროცესის ეგზემპლარებს, ამიტომაც მას შეუძლია განაახლოს სწორი ეგზემპლარი. ამის შესრულება შესაძლებელია მარტივად ობიექტის ლექსიკონით.

გახსენით Election.xaml.cs ფაილი და დაამატეთ კლასის წევრები მომხმარებლის ServiceHost `_sh` სტრიქონის ქვემოთ:

```
private IDictionary<Guid, WorkflowApplication> _incomingRequests;
private IDictionary<Guid, WorkflowApplication> _outgoingRequests;
```

ისინი იყენებს ბიზნეპროცესის ეგზემპლარის იდენტიფიკატორს, როგორც ლექსიკონის გასაღებს და WorkflowApplication ობიექტს, როგორც მნიშვნელობას. ვინაიდან აპლიკაცია ამუშავებს ორივე მუშა პროცესს SendRequest და ProcessRequest, ამიტომაც საჭირო იქნება ლექსიკონის ორი ობიექტი. დავამატოთ კონსტრუქტორში კოდი ამ ობიექტების ინიციალიზებისთვის:

```
_incomingRequests = new Dictionary<Guid, WorkflowApplication>();
_outgoingRequests = new Dictionary<Guid, WorkflowApplication>();
```

საჭიროა კიდევ ერთი მცირე ცვლილება მომხმარებლის CreateRequest ქმედებაში. ბიზნესპროცესის ეგზემპლარის ID გამოყენებულ უნდა იქნას როგორც ElectionRequest კლასის RequestID ველი. აპლიკაცია მას გამოიყენებს პროცესის განახლების დროს. გავხსნათ CreateRequest.cs ფაილი და შევცვალოთ გამოძახება, რომელიც ქმნის ElectionRequest კლასს, ალტერნატიული კონსტრუქტორის გამოსაყენებლად, რომელიც ღებულობს მეხუთე პარამეტრს RequestID -თვის. დავამატოთ მუქი სტრიქონი კოდის შემდეგ ტექსტში:

```
// -- ElectionRequest კლასის შექმნა და მისი შევსება შესატანი
// არგუმენტებით
  ElectionRequest r = new ElectionRequest
    (  ArealName.Get(context),
       MajorDeputy.Get(context),
       Region.Get(context),
       new Branch
         {
     BranchName = app.Settings["Branch Name"].Value,
     BranchID = new Guid(app.Settings["ID"].Value),
     Address = app.Settings["Address"].Value
          },
        context.WorkflowInstanceId  // ეს დაემატა!!!
     );
```

5.1.16. მოვლენათა დამმუშავებელი (Event Handlers)

ახალი მოთხოვნის შესაქმნელად მომხმარებელი შეავსებს *მაჟორ_დეპუტატის_გვარის, ოლქის_დასახელების, რეგიონის_სახელის* ველებს და აამოქმედებს Send Request ღილაკს. ამ მოვლენის ღილაკის რეალიზება მოცემულია 5.11 ლისტინგში, Election.xaml.cs ფაილში.

// --- ლისტინგი 5.11 -----Click Event -ის რეალიზება -----

```
private void btnRequest_Click(object sender,RoutedEventArgs e)
 {
   // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად ---
    Dictionary<string, object> parameters = new
                          Dictionary<string, object>();
    parameters.Add("MajorDeputy", txtMajorDeputy.Text);
    parameters.Add("ArealName", txtArealName.Text);
    parameters.Add("Regioni", txtRegioni.Text);
    parameters.Add("Writer", new
               ListBoxTextWriter(lstEvents));
    WorkflowApplication i =
       new WorkflowApplication(new SendRequest(), parameters);

       _outgoingRequests.Add(i.Id, i);
```

```
            i.Run();
        }
```

ამ მეთოდის პირველი ნაწილი ჩვენთვის ნაცნობია. იგი იყენებს ობიექტის ლექსიკონს შემავალი არგუმენტების შესანახად, რომლებიც უნდა გადაეცეს ბიზნესპროცესს. შემდეგ იგი ქმნის WorkflowApplication-ს, რომლის კონსტრუქტორსაც გადაეცემა პარამეტრები:

მუშა პროცესების დეფინიცია
ობიექტის ლექსიკონი, რომელიც შეიცავს შემავალ არგუმენტებს

WorkflowApplication-ი შემდეგ ემატება _outgoingRequests კოლექციას. ბოლოს, ეგზემპლარი გაიშვება Run () მეთოდით.

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

5.12 ლისტინგი აღწერს ამ ღილაკებისთვის მოვლენათა დამმუშავებლების რეალიზაციას. დავამატოთ ეს მეთოდები Election.xaml.cs კლასში.

// -- ლისტინგი 5.12 --Reserve და Cancel ღილაკების რეალიზაცია --

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

```
            ResumeBookmark(id, false);
          }
        private void ResumeBookmark(Guid id, bool bReserved)
         {
           WorkflowApplication i = _incomingRequests[id];
             try
               {
                i.ResumeBookmark("GetResponse", bReserved);
               }
              catch (Exception e)
               {
                  AddEvent(e.Message);
               }
           }
```

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

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

5.1.17. ApplicationInterface მეთოდები

ჩვენ განვსაზღვრეთ ApplicationInterface კლასის სამი მეთოდი. ახლა უნდა უზრუნველვყოთ მათი რეალიზაცია MainWindow კლასში. გავხსნათ Election.xaml.cs ფაილი და ჩავამატოთ ამ მეთოდების რეალიზაცია 5.13 ლისტინგის მიხედვით.

```
//--ლისტინგი 5.13--ApplicationInterface კლასის მეთოდების რეალიზაცია--
public void RequestElinfo(ElectionRequest request)
    {
```

```
        // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად ---
        Dictionary<string, object> parameters = new
                Dictionary<string, object>();
        parameters.Add("request", request);
        parameters.Add("Writer", new
                  ListBoxTextWriter(lstEvents));
        WorkflowApplication i = new WorkflowApplication(new
                   ProcessRequest(), parameters);

         request.InstanceID = i.Id;
         _incomingRequests.Add(i.Id, i);
          i.Run();
        }
      public void RespondToRequest(ElectionResponse response)
        {
          Guid id = response.RequestID;
          WorkflowApplication i = _outgoingRequests[id];
            try
             {
               i.ResumeBookmark("GetResponse", response);
             }
           catch (Exception e2)
             {
               AddEvent(e2.Message);
             }
          }
       public void AddNewRequest(ElectionRequest request)
         {
            this.requestList.Dispatcher.BeginInvoke
                    (new Action(() =>
             this.requestList.Items.Add(request)));
          }
```

RequestBook () მეთოდი ანალოგიურია btnRequest_Click () მეთოდის. იგი გამოიძახება მაშინ, როცა შემავალი შეტყობინება მიღებულია ServiceHost-დან და სერვის კონტრაქტის RequestBook მეთოდი მითითებულია. ის აგებს ობიექტის ლექსიკონს ერთი არგუმენტის შესანახად, ქმნის WorkflowApplication-ს, ამატებს მას

_incomingRequests კოლექციაში, ხოლო შემდეგ აამოქმედებს მუშა პროცესს.

RespondToRequest () მეთოდი ასევე გამოიძახება ServiceHost-დან მიღებული შეტყობინებით. იგი გამოიძახება მაშინ, როცა RespondToRequest მეთოდია მითითებული. ეს ხდება მაშინ, როცა სხვა ფილიალები აგზავნის უკან პასუხს შემოსულ მოთხოვნაზე. იგი ღებულობს WorkflowApplication-ს _outgoingRequests კოლექციიდან და აღადგენს სანიშნეს, გამავალს ElectionResponse კლასში.

AddNewRequest() გამოიძახება ProcessRequest მუშა პროცესით, როცა მიიღება ახალი შეტყობინება. ეს ხდება InvokeMethod ქმედების დახმარებით. იგი უბრალოდ დაამატებს ჩანაწერს ListView-კონტროლის RequestList-ელემენტში. ვინაიდან ის გამოძახებულ უნდა იქნეს ბიზნესპროცესის შესრულებად ნაკადში, Dispatcher კლასი გამოიყენებს შესასრულებლად Add () მეთოდს main window-ის შესრულებადი ნაკადით. Election.xaml.cs-ის სრული რეალიზაცია მოცემულია 5.14 ლისტინგში.

```
// -- ლისტინგი 5.14 -- Election.xaml.cs საბოლოო სახე -----
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

```
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.ServiceModel.Activities.Description;
using System.ServiceModel.Description;
using System.ServiceModel.Channels;
using System.Activities;
using System.Xml.Linq;
using System.Configuration;

namespace ElectionSys
{
  public partial class MainWindow : Window
   {
    private ServiceHost _sh;
    private IDictionary<Guid, WorkflowApplication>
          _incomingRequests;
   private IDictionary<Guid, WorkflowApplication>
          _outgoingRequests;

   public MainWindow()
     {
        InitializeComponent();
        ApplicationInterface._app = this;
        _incomingRequests = new Dictionary<Guid,
                  WorkflowApplication>();
        _outgoingRequests = new Dictionary<Guid,
                  WorkflowApplication>();
     }
private void Window_Loaded(object sender, RoutedEventArgs e)
  {
  // გაიხსნას config ფაილი და მიეცეს ფილიალის სახელი და მისი
  // ქსელური მისამართი
    Configuration config = ConfigurationManager
        .OpenExeConfiguration(ConfigurationUserLevel.None);
      AppSettingsSection app =
(AppSettingsSection)config.GetSection("appSettings");
       string adr = app.Settings["Address"].Value;

    // ფილიალის სახელის გამოტანა ფორმაზე
```

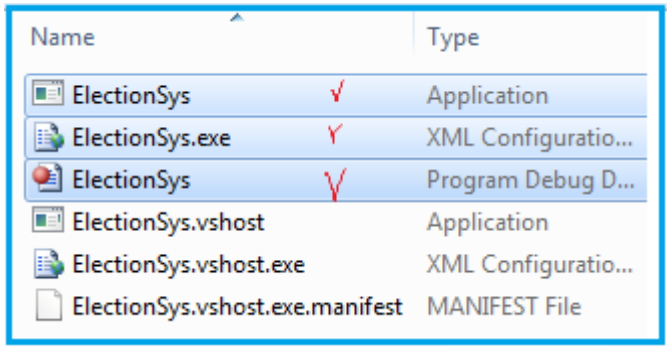
```
    lblBranch.Content = app.Settings["Branch Name"].Value;

     // ServiceHost-ის შექმნა
       _sh = new ServiceHost(typeof(ClientService));

     // დასასრულის წერტილის (Endpoint) დამატება
       string szAddress = "http://localhost:" + adr +
          "/ClientService";
      System.ServiceModel.Channels.Binding bBinding = new
                BasicHttpBinding();
     _sh.AddServiceEndpoint(typeof(IElectionSys), bBinding,
               szAddress);

     // ServiceHost-ის გახსნა შეტყობინებების მისაღებად (listen)
       _sh.Open();
       // ListBoxTextWriter -ის ტესტირება
       //ListBoxTextWriter lbtw = new ListBoxTextWriter();
       //lbtw.Write("ეს არის ტესტი - This is a test");
      }
private void Window_Unloaded(object sender, RoutedEventArgs e)
    {
      // service host-ის დატოვება
        _sh.Close();
    }
 // ----- მოვლენათა დამმუშავებელი -----Event Handler-----
private void btnRequest_Click(object sender,RoutedEventArgs e)
    {
     // Setup a dictionary object for passing parameters
     Dictionary<string, object> parameters = new
              Dictionary<string, object>();
     parameters.Add("MajorDeputy", txtMajorDeputy.Text);
     parameters.Add("ArealName", txtArealName.Text);
     parameters.Add("Regioni", txtRegioni.Text);
     parameters.Add("Writer", new
                 ListBoxTextWriter(lstEvents));

            WorkflowApplication i =
     new WorkflowApplication(new SendRequest(), parameters);

          _outgoingRequests.Add(i.Id, i);
          i.Run();
       }
```

```
// -- Reserve ღილაკის მოვლენის დამმუშავებელი ---
  private void Reserve(object sender, RoutedEventArgs e)
   {
    // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ----
     FrameworkElement fe = (FrameworkElement)sender;
     Guid id = (Guid)fe.Tag;
     ResumeBookmark(id, true);
   }
  // -- Cancel ღილაკის მოვლენის დამმუშავებელი ---
    private void Cancel(object sender, RoutedEventArgs e)
     {
       // ეგზემპლარის ID -ს მიღება Tag-ის თვისებიდან ---
       FrameworkElement fe = (FrameworkElement)sender;
       Guid id = (Guid)fe.Tag;
       ResumeBookmark(id, false);
     }
   private void ResumeBookmark(Guid id, bool bReserved)
     {
       WorkflowApplication i = _incomingRequests[id];
       try
         {
          i.ResumeBookmark("GetResponse", bReserved);
         }
       catch (Exception e)
         {
           AddEvent(e.Message);
         }
     }
 public void RequestElinfo(ElectionRequest request)
     {
  // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად ---
    Dictionary<string, object> parameters = new
            Dictionary<string, object>();
    parameters.Add("request", request);
    parameters.Add("Writer", new
              ListBoxTextWriter(lstEvents));

  WorkflowApplication i = new WorkflowApplication(new
            ProcessRequest(), parameters);

      request.InstanceID = i.Id;
     _incomingRequests.Add(i.Id, i);
```

```
            i.Run();
          }
      public void RespondToRequest(ElectionResponse response)
        {
          Guid id = response.RequestID;
          WorkflowApplication i = _outgoingRequests[id];
            try
             {
               i.ResumeBookmark("GetResponse", response);
              }
            catch (Exception e2)
             {
               AddEvent(e2.Message);
             }
          }
       public void AddNewRequest(ElectionRequest request)
          {
            this.requestList.Dispatcher.BeginInvoke
        (new Action(() => this.requestList.Items.Add(request)));
          }
         public ListBox GetEventListBox()
          {
            return this.lstEvents;
          }
          private void AddEvent(string szText)
          {
              lstEvents.Items.Add(szText);
          }
      }
}
```

5.1.18. აპლიკაციის ამუშავება

პროგრამული სისტემის ასამუშავებლად საჭიროა აპლიკაციის რამდენიმე ასლის (კოპიოს) ერთად გაშვება, თითოეული თავისი კონფიგურაციის ფაილის ვერსიით. თავიდან საჭიროა F6 კლავიშის ამოქმედება solution-ის (გადაწყვეტის) აღსადგენად და კომპილატორის შენიშვნების აღმოსაფხვრელად. შევქმნათ ახალი ფოლდერი ElectionSys-ფოლდერის ქვეშ, რომელიც იძახებს ფილიალებს. შემდეგ დავაკოპიროთ ფილიალის ფოლდერში ფაილები, რომლებიც 5.7 ნახაზზეა ნაჩვენები.



ნახ.5.7.

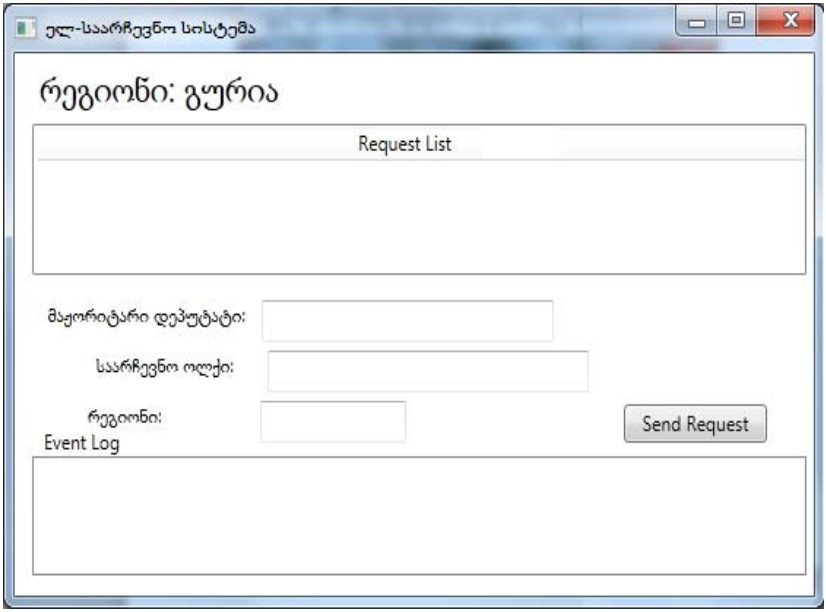
გავხსნათ ElectionSys.exe.config ფაილი (ფილიალის ქვეფოლდერში) და შევასწოროთ შემდეგნაირად:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Branch Name" value="Olqi Ozurgeti"/>
    <add key="ID" value="{43E6DADD-4751-4056-8BB7-
                                        7459B5C361AB}"/>
    <add key="Address" value="8730"/>
    <add key="Request Address" value="8000"/>
  </appSettings>
</configuration>
```

შენიშვნა: თუ შედეგი მიღებულია შეცდომით, უნდა ვცადოთ აპლიკაციის გაშვება ადმინისტრატორის უფლებებით (!).

ფილიალის ფოლდერში ElectionSys.exe ფაილზე ორჯერ დავაწკაპუნოთ. აპლიკაცია ასე გამოიყურება (ნახ.5.8).

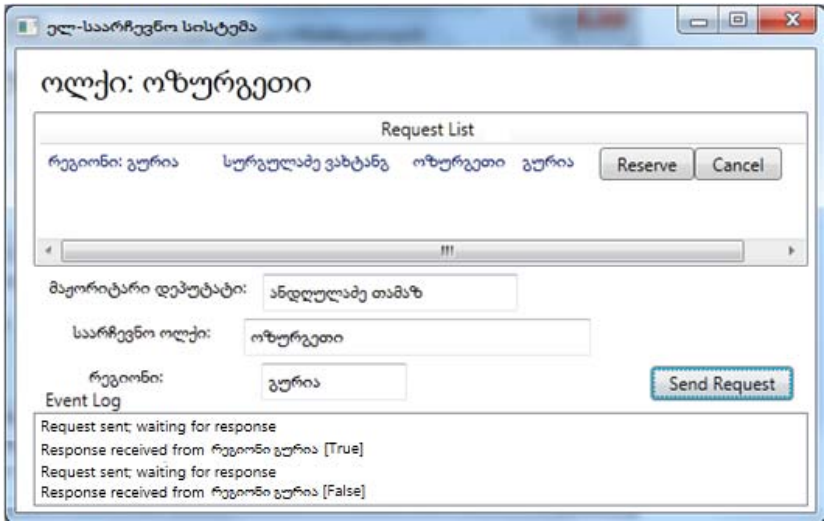
Visual Studio-ში F5-ით გავმართოთ აპლიკაცია. ანალოგიური ფანჯარა უნდა მივიღოთ, ოღონდ სათაურით – „საარჩევნო ოლქი“ (ან „რეგიონალური საარჩევნო კომისია“ ან „ცენტრალური საარჩევნო კომისია“ და ა.შ.).



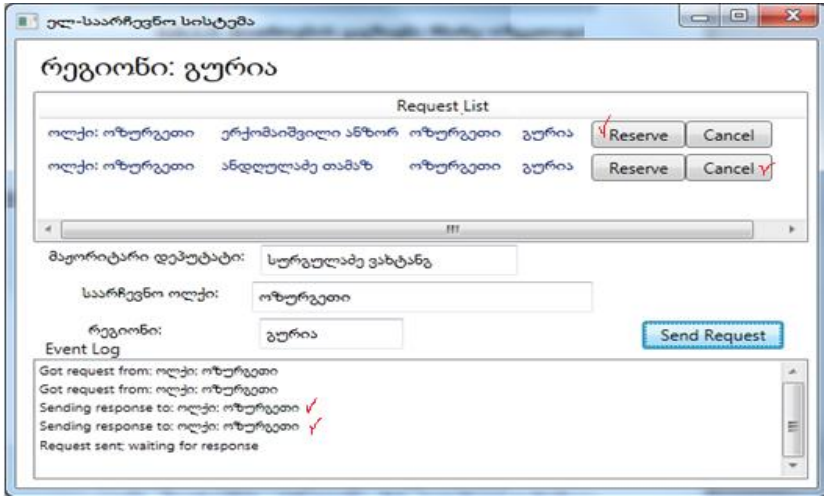
ნახ.5.8.

ერთ-ერთ აპლიკაციაში შევიტანოთ მაჟორიტარი დეპუტატის გვარი, მხარე (ოლქი), რეგიონი და ავამოქმედოთ ღილაკი „მოთხოვნის გაგზავნა“ (ნახ.5.9). მოთხოვნა უნდა გამოჩნდეს მეორე ფანჯრის მოთხოვნების სიაში. დააჭირეთ Reserve ღილაკს მეორე აპლიკაციაში (ნახ.5.10). გამოჩნდება შეტყობინება პირველი ფანჯრის მოვლენების ჟურნალში, რომ პასუხი მიღებულია.

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



ნახ.5.9. მოთხოვნის გაგზავნა ოლქიდან „ოზურგეთი“



ნახ.5.10. მოთხოვნის დამუშავება რეგიონში „გურია“

VI თავი
სერვისორიენტირებული არქიტექტურის პროგრამული რეალიზაცია

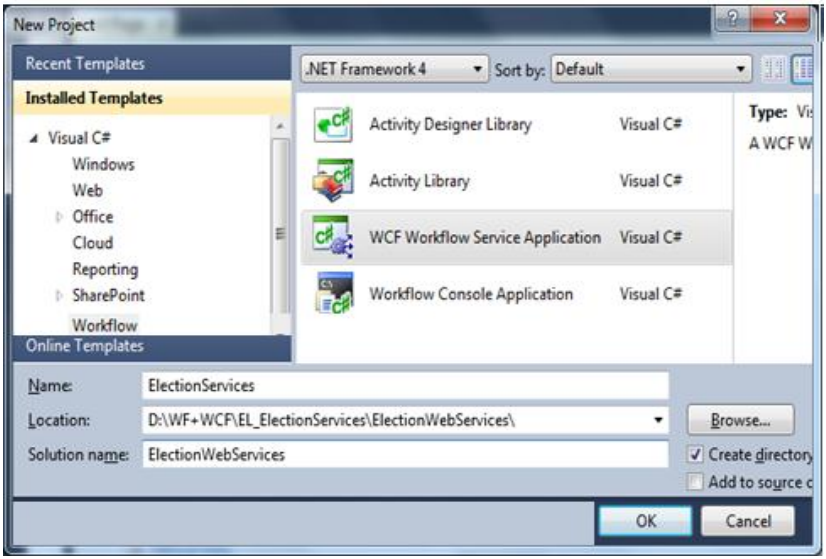
ბიზნესპროცესები შეიძლება განთავსდეს Web-სერვისში, რომელიც უზრუნველყოფს იდეალურ საშუალებას მუშა პროცესის გადაწყვეტილების მისაწოდებლად არამუშა პროცესის კლიენტებისთვის, როგორიცაა ვებაპლიკაციები. Web-სერვისი იღებს მოთხოვნას, ასრულებს მის სათანადო გადამუშავებას და აბრუნებს პასუხს. ეს, ბუნებრივია, სრულდება Receive და Send ქმედებებით, რომლებიც წინა თავში განვიხილეთ. აქტიურობები ინტეგრირებულია Windows Communication Foundation (WCF) -თან, ჩვენ შეგვიძლია ადვილად შევქმნათ WCF სერვისები.

6.1. ბიზნესპროცესის სერვისის შექმნა

ავამუშავოთ Visual Studio 2010 და შევქმნათ ახალი პროექტი WCF Workflow Service Application template გამოყენებთ. შევიტანოთ პროექტის სახელი BookInventory და solution-ის სახელი WebServices (ნახ.6.1). შეიქმნება საინიციალიზაციო workflow Sequence ბლოკი, რომელიც შეიცავს Receive და SendReply ქმედებებს, როგორც 6.2 ნახაზზეა ნაჩვენები.

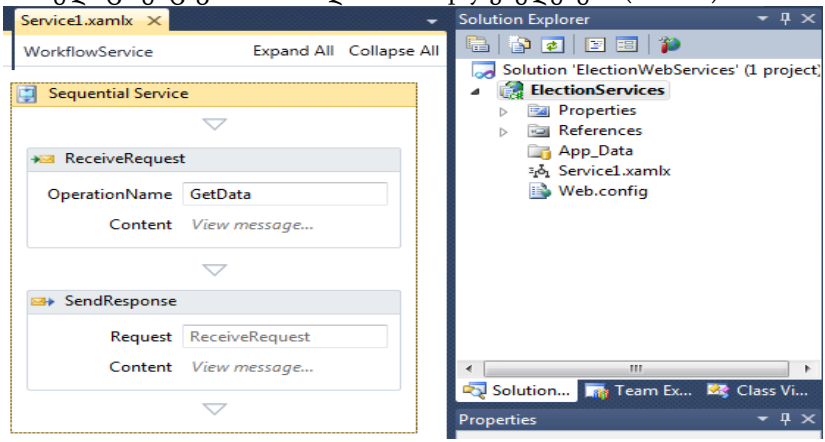
თავიდან საჭიროა ამ ქმედებების კონფიგურაცია სერვისის კონტრაქტის განსაზღვრის მიზნით, რომელსაც ისინი დააკმაყოფილებს. შემდეგ დავამატოთ დამუშავების მუშა პროცესი, რომელიც განხორციელდება Receive და SendReply ქმედებებს შორის.

შაბლონი შექმნის საწყის მუშა პროცესს ფაილში სახელით Service1.xamlx. შევცვალოთ Solution Explorer-ში ეს სახელი MajorDeputyInfo.xamlx -ით. ***„სერვისი, რომლის შექმნაც გვინდა, ძებნის მითითებულ დეპუტატს და აბრუნებს უკან ყოველი ასლის მდგომარეობას, რომელიც საარჩევნო უბანს ეკუთვნის”.***



ნახ.6.1. ბიზნესპროცესების სერვისების WCF აპლიკაციის შექმნა

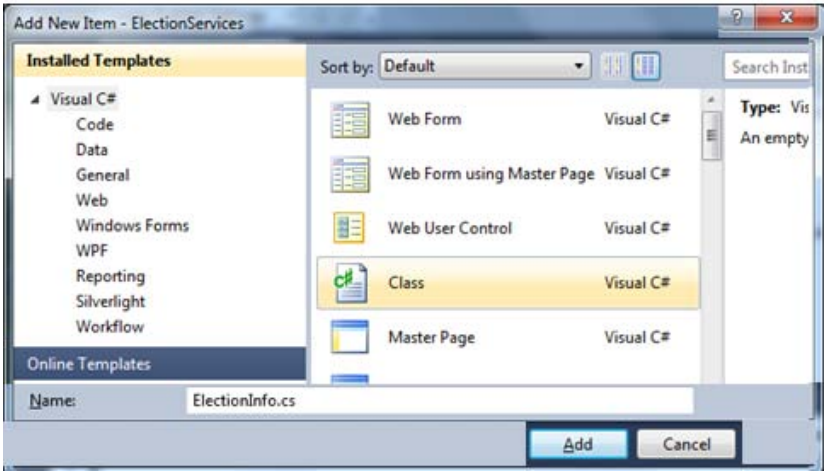
შეიქმნება საინიციალიზაციო workflow Sequence ბლოკი, რომელიც შეიცავს Receive და SendReply ქმედებებს (ნახ.6.2).



ნახ.6.2. workflow sequence საინიციალიზაციო ბლოკი

6.2. სერვისის კონტრაქტის განსაზღვრა

Solution Explorer-ში, მარჯვენა ღილაკით MajorDeputyInfo პროექტზე ავირჩიოთ: Add->Class სახელით ElectionInfo.cs, რომლის ტექსტი მოცემულია 6.1 ლისტინგში (ნახ.6.3).



ნახ.6.3. სერვისის კონტრაქტი

```
// ----- ლისტინგი 6.1 ---ElectionInfo.cs --
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;

namespace ElectionServices
{
   /*************************************************/
   // Define the service contract, IElectionServices
   // which consists of a single method, LookupElection()
   /*************************************************/
   [ServiceContract]
   public interface IElectionServices
```

```
{
    [OperationContract]
    ElectionInfoList LookupElection(ElectionSearch request);
}
/*************************************************/
// Define the request message, ElectionSearch
/*************************************************/
[MessageContract(IsWrapped = false)]
public class ElectionSearch
{
    private String _Regioni;
    private String _Mxare;
    private String _MajorDeputy;
    public ElectionSearch() {  }
    public ElectionSearch(String regioni, String mxare, String majorDeputy)
    {
        _Regioni = regioni;
        _Mxare = mxare;
        _MajorDeputy = majorDeputy;
    }
    #region Public Properties
    [MessageBodyMember]
    public String Regioni
    {
        get { return _Regioni; }
        set { _Regioni = value; }
    }
    [MessageBodyMember]
    public String Mxare
    {
        get { return _Mxare; }
        set { _Mxare = value; }
    }
```

```
    [MessageBodyMember]
    public String MajorDeputy
    {
        get { return _MajorDeputy; }
        set { _MajorDeputy = value; }
    }
  #endregion Public Properties
}
/***********************************************/
// Define the ElectionInfo class
/***********************************************/
[MessageContract(IsWrapped = false)]
public class ElectionInfo
 {
    private Guid _InventoryID;   // არ შეცვალე
    private String _Regioni;
    private String _Mxare;
    private String _MajorDeputy;
    private String _Status;     // არ შეცვალე

    public ElectionInfo ( )  {  }
    public ElectionInfo (String regioni, String mxare, String majorDeputy,
            String status)
    {
        _Regioni = regioni;
        _Mxare = mxare;
        _MajorDeputy = majorDeputy;
        _Status = status;
        _InventoryID = Guid.NewGuid();
    }
    #region Public Properties
    [MessageBodyMember]
    public Guid InventoryID
```

```
    {
        get { return _InventoryID; }
        set { _InventoryID = value; }
    }
    [MessageBodyMember]
    public String Regioni
    {
        get { return _Regioni; }
        set { _Regioni = value; }
    }
    [MessageBodyMember]
    public String Mxare
    {
        get { return _Mxare; }
        set { _Mxare = value; }
    }
    [MessageBodyMember]
    public String MajorDeputy
    {
        get { return _MajorDeputy; }
        set { _MajorDeputy = value; }
    }
   [MessageBodyMember]
    public String status
    {
        get { return _Status; }
        set { _Status = value; }
    }
    #endregion Public Properties
}
/*************************************************/
// Define the response message, ElectionInfoList, which
// is a list of ElectionInfo classes
```

```
    /*************************************************/
    [MessageContract(IsWrapped = false)]
    public class ElectionInfoList
    {
        private List<ElectionInfo> _ElectionList;

        public ElectionInfoList()
        {
            _ElectionList = new List<ElectionInfo>();
        }
        [MessageBodyMember]
        public List<ElectionInfo> ElectionList
        {
            get { return _ElectionList; }
        }
    }
}
```

სერვისის კონტრაქტი IElectionServices შეიცავს ერთადერთ მეთოდს LookupElection(). იგი მონაცემებს გადასცემს ElectionSearch კლასს, რომელსაც აქვს სხვადასხვა თვისებები, საჭირო მონაცემების მოსაძებნად, მაგალითად რეგიონი და მაჟორიტარი დეპუტატი. ის აბრუნებს უკან ElectionInfoList კლასს, რომელიც შეიცავს ElectionInfo კლასების კოლექციას.

F6 ამოქმედებით აიგება განახლებული solution.

შენიშვნა*) MessageContract ატრიბუტი მიუთითებს, რომ ეს კლასი ჩართული იქნება SOAP ბარათში. SOAP–ის გამოყენების დროს შეტყობინებები გადაიცემა XML-ის მსგავსი ფორმატირებადი ენით. ეს უზრუნველყოფს კლიენტებსა და სერვერს შორის მაღალი ხარისხის ურთიერთქმედების პლატფორმას. SOAP არის სტანდარტული პროტოკოლი, რომლის მხარდაჭერაც აქვს WCF -ს.

არსებობს აგრეთვე MessageBodyMember ატრიბუტი მის ყოველ პუბლიკ-თვისებაზე. ეს აუცილებელია WCF-ფენისთვის, რათა სწორად დაფორმატდეს SOAP შეტყობინება.

WCF–ის ბოლო წერტილის განსაზღვრისთვის არსებობს ინფორმაციის სამი პორცია, რომლებზეც მითითებულ უნდა იქნას: მიერთება (binding), მისამართი და კონტრაქტი.

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

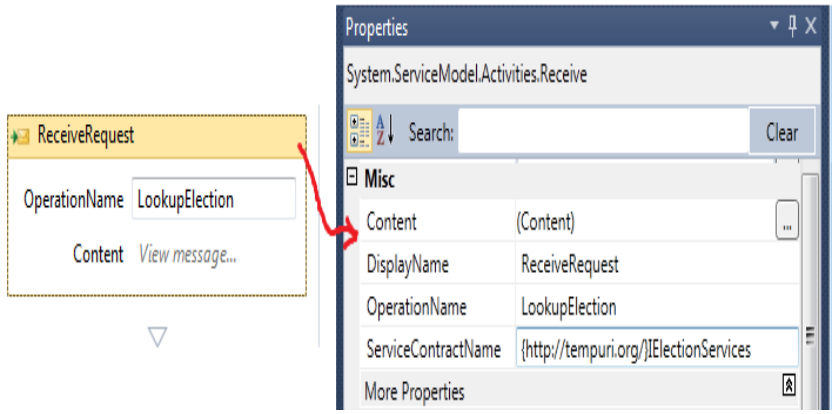
მისამართი მიუთითებს თუ სად უნდა ვიპოვოთ ბოლო წერტილი, და მისამართის ტიპს, რომლის გამოყენება დამოკიდებულია მიერთებაზე. მაგალითად, HTTP–მიერთებისას უნდა მიეთითოს URL, ხოლო TCP-თვის მისამართი იქნება სერვერის სახელი ან IP–მისამართი.

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

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

6.3. Receive და SendReply კონფიგურირება

გავხსნათ ElectionServices.xamlx ფაილი და ავირჩიოთ “ReceiveRequest” ქმედება. თვისებათა ფანჯარაში ServiceContract თვისებას აქვს default-მნიშვნელობა {http://tempuri.org/}IService. შევცვალოთ Iservice სტრიქონი **IElectionServices**-ით. შევიტანოთ OperationName როგორც **LookupElection** (ნახ.6.4).

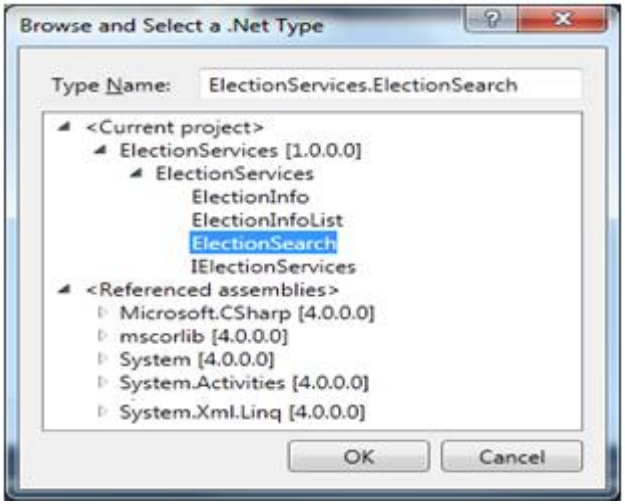


ნახ.6.4. კონტრაქტის სახელის შეცვლა

ეკრანზე WorkflowService-დიზაინერში ქვედა მარცხენა კუთხეში დავაწკაპუნოთ Variables ღილაკზე. გამოჩნდება შაბლონი ორი ცვლადის შესაქმნელად. გადასამუშავებელი ცვლადი (handle variable) გამოიყენება პასუხის კორელაციისთვის იმ ეგზემპლართან, რომელმაც გააგზავნა მოთხოვნა.

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

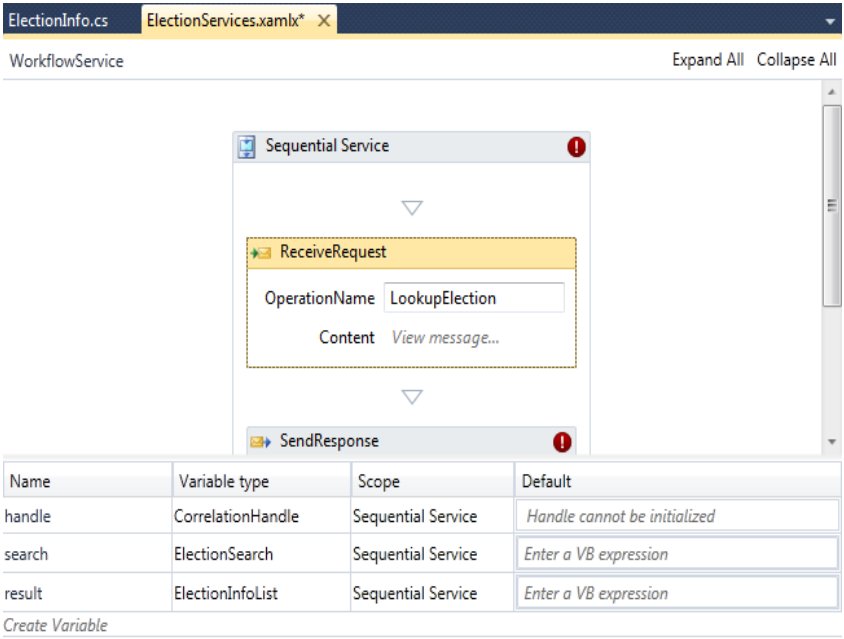
პირველისთვის ავირჩიოთ სახელი Name და ტიპი – Browse for Typs. ახალ დიალოგურ ფანჯარაში BookInventory ნაკრები გავაფართოვოთ BookSearch-ით (ნახ.6.5).



ნახ.6.5.

მეორე ცვლადისთვის შეიტანეთ Name: result. ტიპი შეირჩევა Browse-დან, ElectionInfoList კლასით. მიიღება 6.6 ნახაზზე მოცემული შემთხვევა.

ბიზნესპროცესების დიზაინერში „ReceiveRequest“ (მოთხოვნების მიღების) ქმედებას აქვს view message (შეტყობინების ნახვის) ლინკი შინაარსის თვისებისთვის.

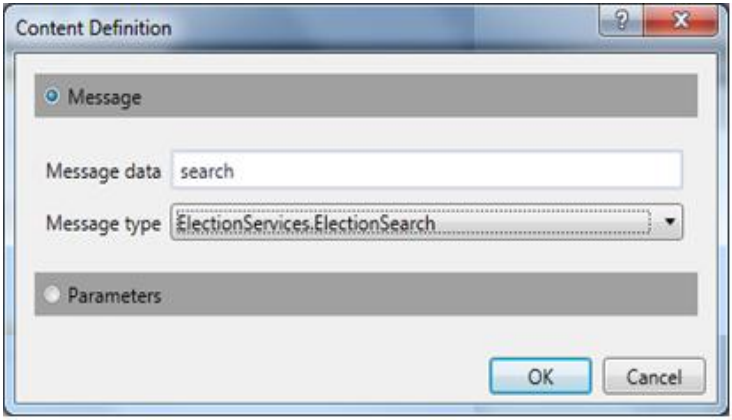


ნახ.6.6. ცვლადები განისაზღვრა ბიზნესპროცესისთვის

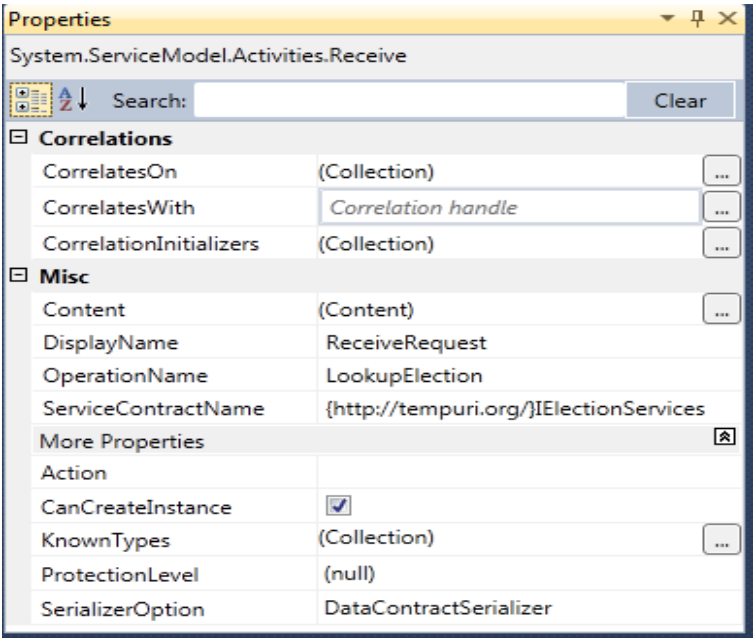
მისი ამოქმედებით იხსნება დიალოგური ფანჯარა შემავალი შეტყობინების დასადგენად (შეიძლება ასევე სამწერტილიანი ღილაკის გამოყენებაც, თვისების გვერდით). შესასვლელი განისაზღვრება ორი ხერხით: შეტყობინებით ან პარამეტრების ერთობლიობით. ამ თავში მოგვიანებით ჩვენ განვიხილავთ მეორე ხერხსაც. ახლა დავაკვირდეთ, რომ რადიობუტონის გადამრთველი შეტყობინებისთვის სწორადაა არჩეული.

Message data თვისებისთვის შევიტანოთ **search.** ის მიუთითებს, რომ შემომავალი შეტყობინება უნდა ინახებოდეს search ცვლადში. Message ტიპისთვის ვირჩევთ ElectionServices.ElectionSearch. დიალოგური ფანჯარა მოცემულია 6.7 ნახაზზე.

თვისებების ფანჯარა მოცემულია 6.8 ნახაზზე.

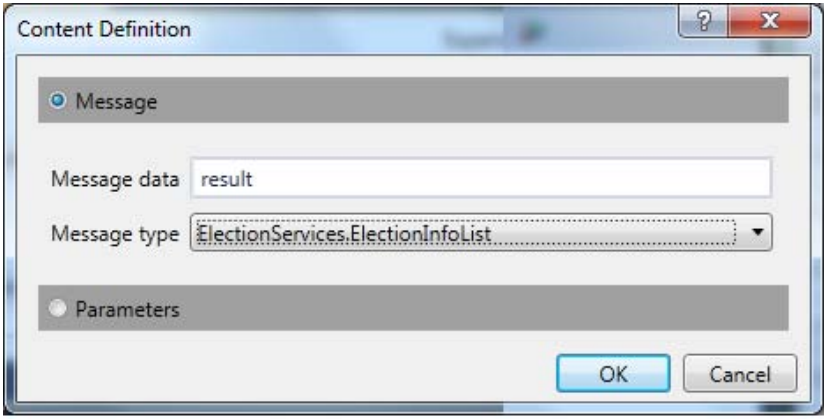


ნახ.6.7. შემავალი შეტყობინების განსაზღვრა



ნახ.6.8. Receive ქმედების თვისებათა ფანჯარა

ვირჩევთ ქმედებას „SendResponse“ და ავამოქმედებთ view message ლინკს. კვლავ შევამოწმოთ, რომ არჩეულია შეტყობინების გადამრთველი. Message data თვისებისთვის შევიტანოთ result, ხოლო Message type თვისებისთვის ვირჩევთ ElectionInfoList კლასს (ნახ.6.9).



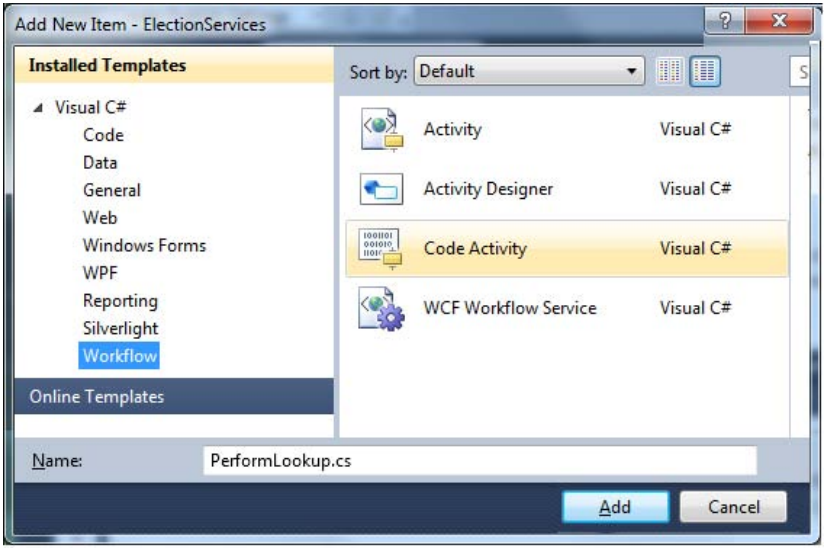
ნახ.6.9. result გამომავალი ცვლადის განსაზღვრა

6.4. PerformLookup აქტიურობის შექმნა

ამ პროექტისთვის შევქმნით მომხმარებლის ქმედებას (აქტიურობას) “lookup“-ის (ძებნის) შესასრულებლად. ფაქტობრივად, მარტივი იქნება ხისტად-კოდირებული მონაცემების დაბრუნება. რეალურ სიტუაციაში მან, ალბათ, უნდა შეასრულოს მონაცემთა ბაზისადმი მოთხოვნა საჭირო მონაცემების მისაღებად.

Solution Explorer-ში BookInventory პროექტზე მარჯვენა ღილაკით ვირჩევთ Add ➤ New Item და დიალოგში Workflow კატეგორიისთვის ვირჩევთ Code Activity-ს. Name-ში შევიტანთ PerformLookup.cs სახელს (ნახ.6.10).

შევიტანოთ PerformLookup ქმედების რეალიზაციისთვის 6.2 ლისტინგში მოცემული კოდი.



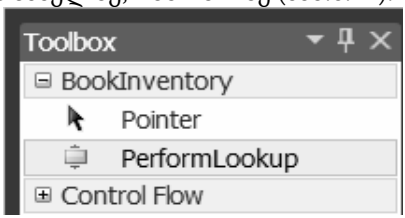
ნახ.6.10. მომხმარებლის ქმედების შექმნა

```
// ლისტინგი 6.2 ----------------------------------
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Activities;

namespace ElectionServices
{
    /*****************************************************/
    // This custom activity creates a ElectionInfoList class
    // which is a collection of ElectionInfo classes.  It uses
    // the input parameters (ElectionSearch class) to "lookup"
    // the matching items. The ElectionInfoList class is
    // returned in the output parameter.
    /*****************************************************/

    public sealed class PerformLookup : CodeActivity
    {
```

```
        // Define an activity input argument of type string
        public InArgument<ElectionSearch> Search { get; set; }
        public OutArgument<ElectionInfoList> ElectionList {
get; set; }

 protected override void Execute(CodeActivityContext context)
  {
    string regioni = Search.Get(context).Regioni;
    string mxare = Search.Get(context).Mxare;
    string majorDeputy = Search.Get(context).MajorDeputy;

    ElectionInfoList l = new ElectionInfoList();

    l.ElectionList.Add(new ElectionInfo(regioni, mxare,
majorDeputy, "Available"));
    l.ElectionList.Add(new ElectionInfo(regioni, mxare,
majorDeputy, "CheckedOut"));
    l.ElectionList.Add(new ElectionInfo(regioni, mxare,
majorDeputy, "Missing"));
    l.ElectionList.Add(new ElectionInfo(regioni, mxare,
majorDeputy, "Available"));
    ElectionList.Set(context, l);
  }
 }
}
```

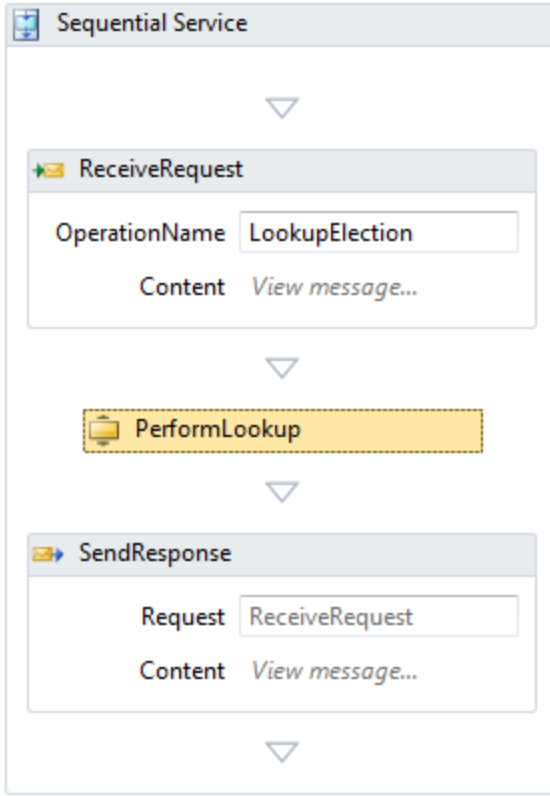
F6-ით განვახორციელოთ აპლიკაციის აღდგენა.

გავხსნათ ElectionServices.xamlx ფაილი. გასათვალისწინებელია, რომ მომხმარებლის PerformLookup ქმედება არის ინსტრუმენტების პანელზე, ToolBox-ზე (ნახ.6.11).



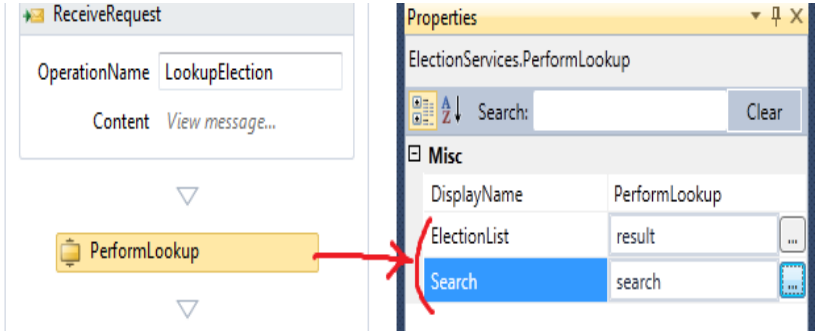
ნახ.6.11.

გადმოვიტანოთ PerformLookup ქმედება „ReceiveRequest” და „SendResponse” ქმედებებს შორის, როგორც ეს 6.12 ნახაზზეა ნაჩვენები.



ნახ.6.12

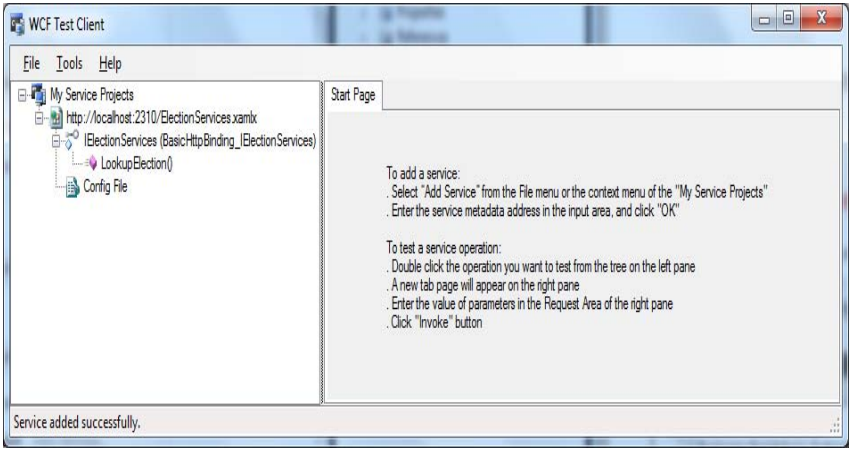
ავირჩიოთ PerformLookup ქმედება. Properties-ის ფანჯარაში, ElectionkList თვისებისთვის შევიტანოთ result; Search თვისებისთვის კი – search.



ნახ.6.13.

6.5. სერვისის ტესტირება

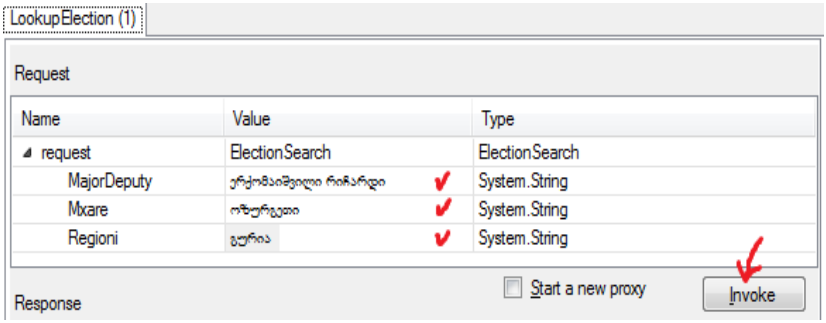
F5-ით ჩავატაროთ სერვისის გამართვა (debug). ვინაიდან ეს Web-სერვისია, Visual Studio ავტომატურად აამუშავებს WCF Test Client-ს. ეს მეტად მოსახერხებელი უტილიტაა. იგი ჩატვირთავს Web-სერვისებს და აღმოაჩენს გათვალისწინებულ მეთოდებს. ისინი ჩანს 6.14 ნახაზის მარცხენა პანელზე.



ნახ.6.14. WCF Test Client ინიციალიზაციის ფანჯარა

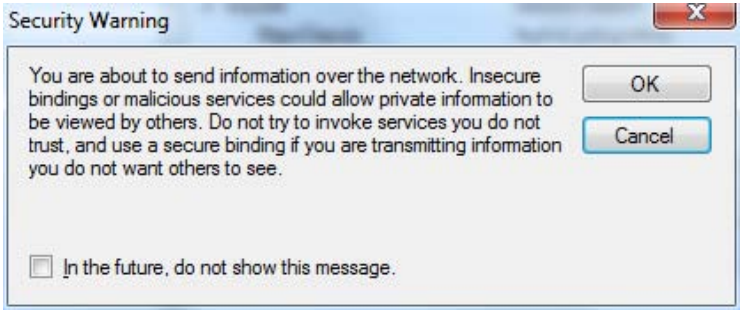
LookupElection() მეთოდზე მაუსის 2-ჯერ დაჭერით მარჯვენა პანელის ზედა ნაწილში გამოიყოფა ადგილი შემოსული შეტყობინების განსათავსებლად. იგი მზადაა რთული შეტყობინებებისთვისაც, რომლებიც შეიცავს კლასების და თვისებების კოლექციებს.

შევიტანოთ Regioni, Mxare და MajorDeputy მნიშვნელობები (ნახ.6.15).



ნახ.6.15.

შემდეგ ავამოქმედოთ Invoke ღილაკი. თავდაპირველად გამოჩნდება გაფრთხილება (ნახ.6.16), შემდეგ კი შედეგები, რომლებიც ანალოგიურია 6.17 ნახაზზე ნაჩვენები შედეგების.



ნახ.6.16.

Name	Value	Type
◢ (return)		ElectionInfoList
◢ ElectionList	length=4	ElectionServices.ElectionInfo[]
◢ [0]		ElectionServices.ElectionInfo
InventoryID	7bcde832-5272-4c63-9ee2-c21ad5a	System.Guid
MajorDeputy	"ერქომაიშვილი რიჩარდი"	System.String
Mxare	"ოზურგეთი"	System.String
Regioni	"გურია"	System.String
status	"Available"	System.String
◢ [1]		ElectionServices.ElectionInfo
InventoryID	fbad0d21-febe-4a35-8f2a-a366332f9	System.Guid
MajorDeputy	"ერქომაიშვილი რიჩარდი"	System.String
Mxare	"ოზურგეთი"	System.String
Regioni	"გურია"	System.String
status	"CheckedOut"	System.String
◢ [2]		ElectionServices.ElectionInfo
InventoryID	7583ba31-50d2-4e9d-83b6-9d87544	System.Guid
MajorDeputy	"ერქომაიშვილი რიჩარდი"	System.String
Mxare	"ოზურგეთი"	System.String
Regioni	"გურია"	System.String
status	"Missing"	System.String
◢ [3]		ElectionServices.ElectionInfo
InventoryID	5dca683e-4f4e-46f4-b834-eec7d305	System.Guid
MajorDeputy	"ერქომაიშვილი რიჩარდი"	System.String
Mxare	"ოზურგეთი"	System.String
Regioni	"გურია"	System.String
status	"Available"	System.String

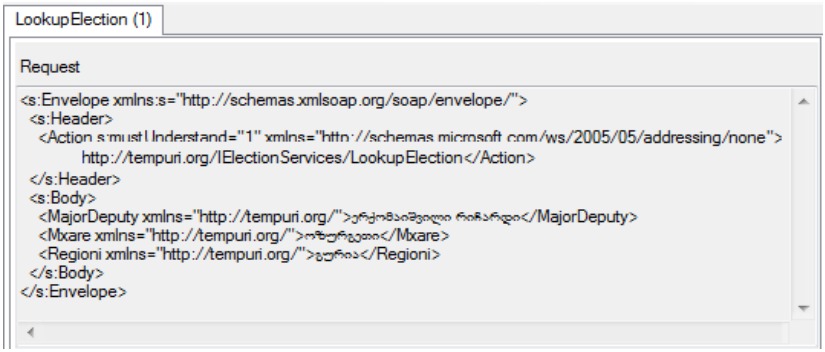
Formatted XML

ნახ.6.17. WCF Client ტესტის სერვისის შედეგები

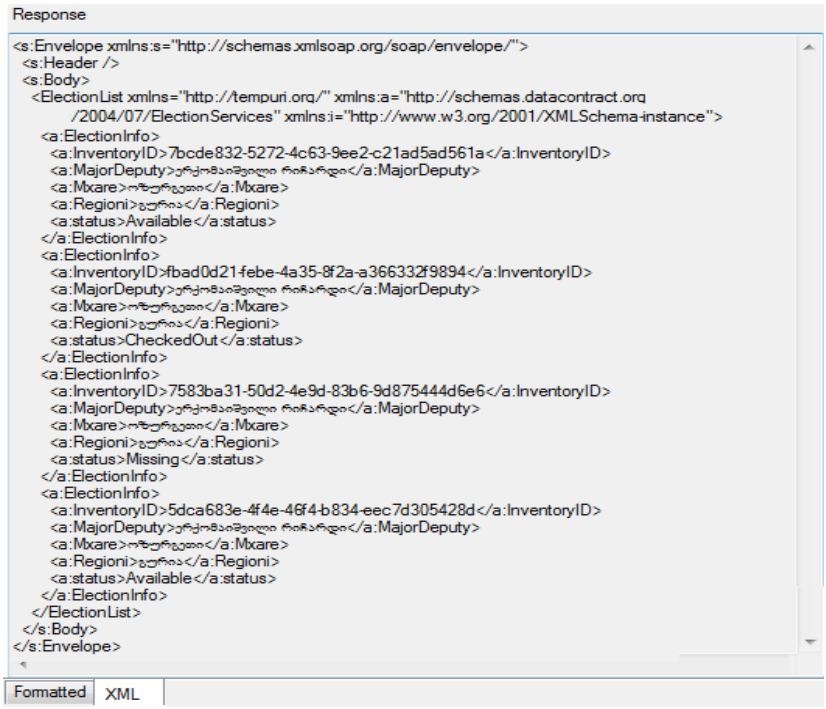
სერვისი აბრუნებს ElectionInfo-ს ოთხ კლასს. ამ ნახაზზე მესამე ჩანაწერი გაფართოებულია, რათა დავინახოთ დაბრუნებული მონაცემების მაგალითი. მოცემულ კონკრეტულ ელემენტს აქვს სტატუსი Missing (დაკარგული, ამოვარდნილი).

შენიშვნა: თუ .xamlx ფაილი არის მიმდინარე ფაილი Visual Studio-ში, როცა F5-ს ვაჭერთ, მაშინ WCF Test Client გაიშვება, როგორც აქაა ნაჩვენები. თუ სხვა ფაილია აქტიური, მაშინ გამოჩნდება შესაბამისი კატალოგი და შედეგები. ის უნდა დაიხუროს და გააქტიურდეს ჩვენთვის საჭირო .xamlx ფაილი.

6.18 ნახაზზე მოცემულია Request და Response შესაბამისი XML კოდები.



ნახ.6.18-ა. Request-ის XML ფაილი



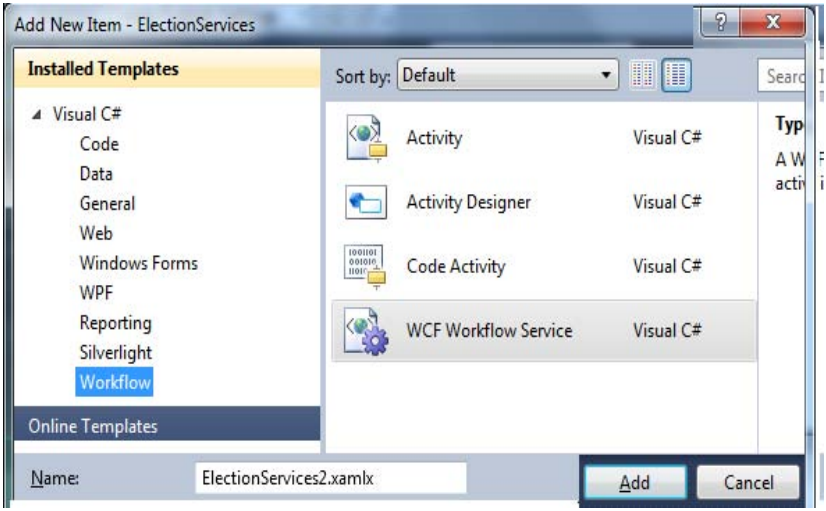
ნახ.6.18-ბ. Response-ს XML ფაილი

6.6. პარამეტრების გამოყენება

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

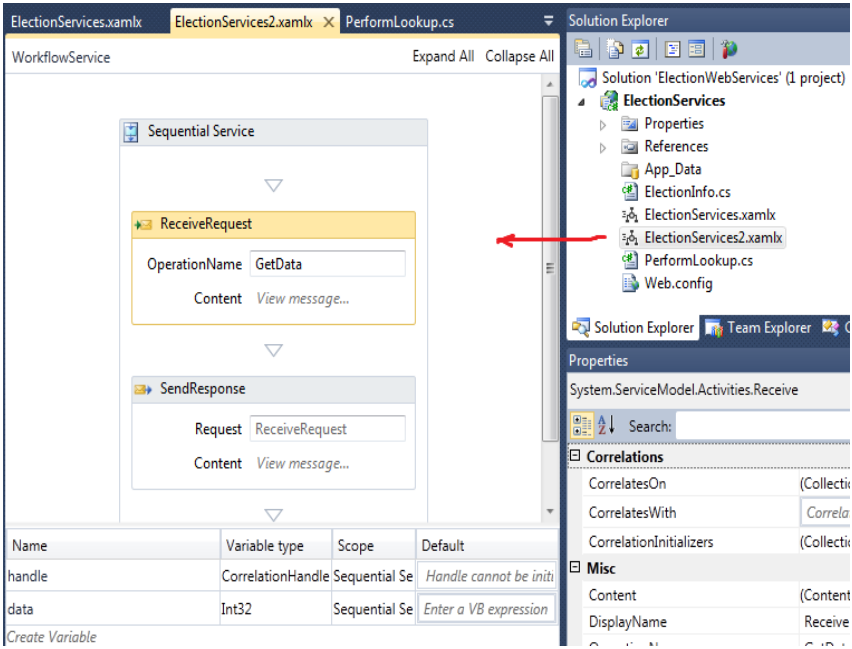
6.7. მეორე სერვისის შექმნა

Solution Explorer-ში მარჯვენა ღილაკს ვაჭერთ BookInventory პროექტზე და ვირჩევთ Add ➤ New Item. ამ დიალოგში ვირჩევთ WCF Workflow Service შაბლონს, რომელიც Workflow კატეგორიაშია. ეს ნაჩვენებია 6.19 ნახაზზე, სახელით Name: **ElectionServices2.xamlx**.



ნახ.6.19. WCF ბიზნესპროცესის სერვისის შექმნა

სამუშაო გარემოს ექნება ასეთი სახე (ნახ.6.20).

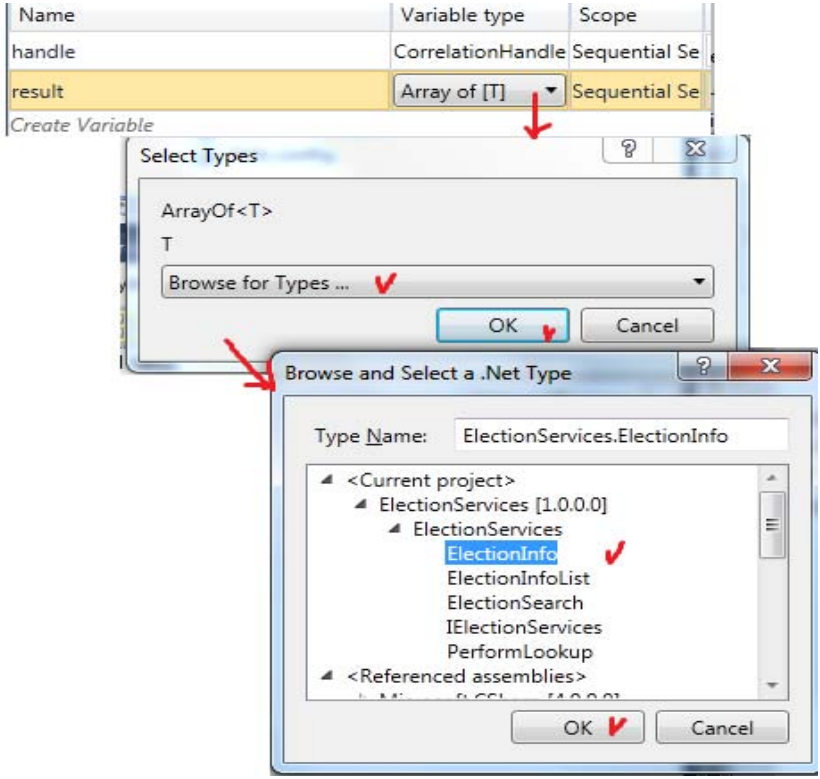


ნახ.6.20.

ბიზნესპროცესის დიზაინერში ქვემოთ, მარცხნივ ავამოქმედოთ ცვლადების ღილაკი Variables. რამდენიმე ცვლადი უკვე შექმნილია პირველი სერვისის შექმნის დროს.

წავშალოთ data ცვლადები და შევქმნათ ახალი ცვლადი, სახელით result. ცვლადის ტიპისთვის (type) ავირჩიოთ ArrayOf<T>. გამოჩნდება დიალოგური ფანჯარა <T> ტიპის ასარჩევად (ნახ.6.21). ვირჩევთ Browse-ს და ElectionServices კრებულიდან ვირჩევთ ElectionInfo კლასს.

კიდევ დავამატოთ სამი String ტიპის ცვლადი სახელებით: რეგიონი, მხარე და მაჟორიტარი დეპუტეტი. 6.22 ნახაზზე ნაჩვენებია ცვლადების სია.



ნახ.6.21.

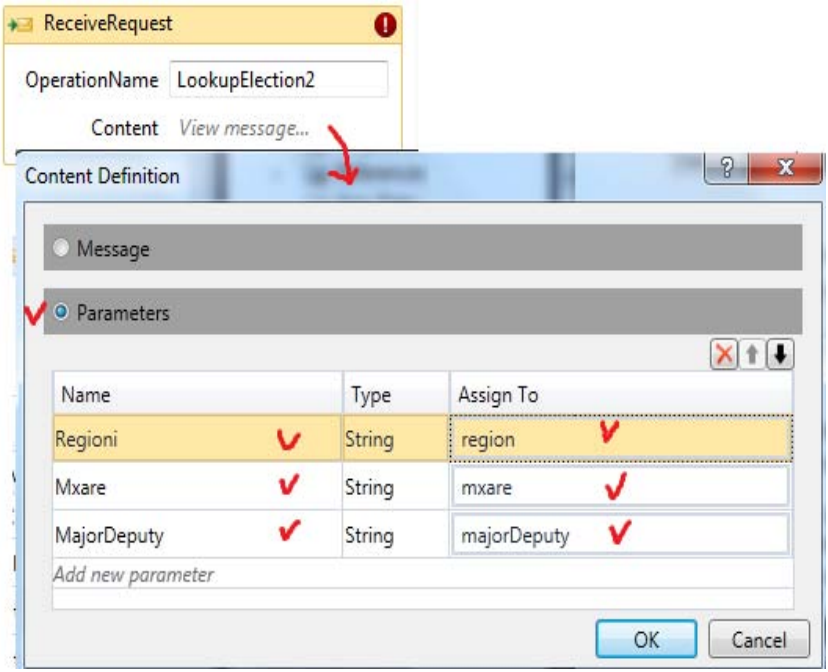
Name	Variable type	Scope	Default
handle	CorrelationHandle	Sequential Servic	*Handle cannot be initialized*
result	ElectionInfo[]	Sequential Servic	*Enter a VB expression*
region	String	Sequential Servic	*Enter a VB expression*
mxare	String	Sequential Servic	*Enter a VB expression*
majorDeputy	String	Sequential Servic	*Enter a VB expression*
Create Variable			

ნახ.6.22. ცვლადების სია

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

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

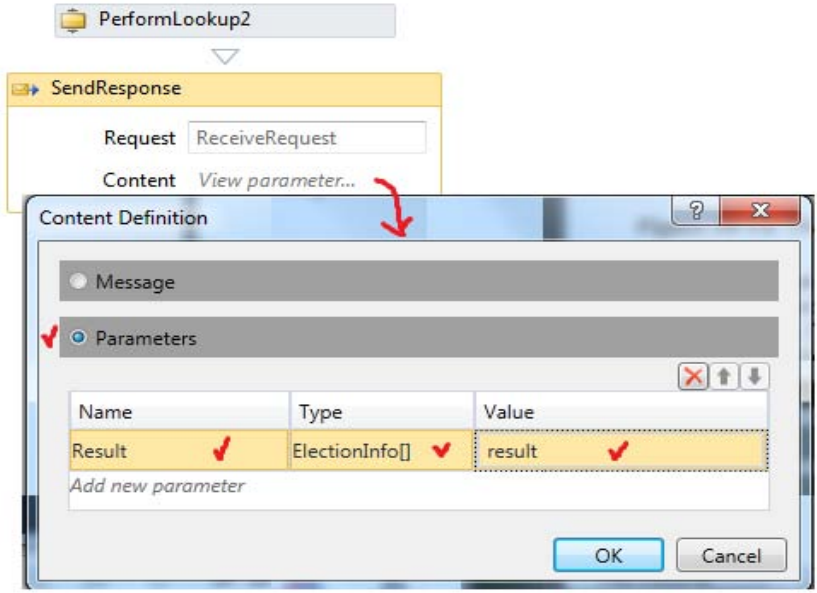
ავირჩიოთ „ReceiveRequest” ქმედება (მოთხოვნის მიღება) და ავამოქმედოთ Content ლინკი. ამჯერად დავაყენოთ გადამრთველი პოზიციაში „პარამეტრები“ (ნახ.6.23).



ნახ.6.23. ReceiveRequest პარამეტრების სია

პარამეტრები ყენდება ცვლადების და არგუმენტების ანალოგიურად. ავამოქმედოთ ლინკი „Add new parameter “. შევიტანოთ Name როგორც რეგიონი და დავაყენოთ Assign რეგიონის თვისებით. დავამატოთ კიდევ ერთი პარამეტრი, სახელით მხარე და Assign მხარეზე. დავამატოთ მესამე პარამეტრი მაჟორიტარი დეპუტატი და Assign მასზე. შევსებულ გვერდს ექნება ასეთი სახე (ნახ.6.23).

დავაჭიროთ „SendResponse” ქმედების Content ლინკს. ავირჩიოთ გადამრთველი პარამეტრები და დავაჭიროთ ლინკს „Add new parameter“. შევიტანოთ Name როგორც Result; ტიპისთვის ავირჩიოთ BookInventory.BookInfo[] ჩამოსაშლელი სიიდან. დიალოგურ ფანჯარას აქვს 6.24 ნახაზზე ნაჩვენები სახე.

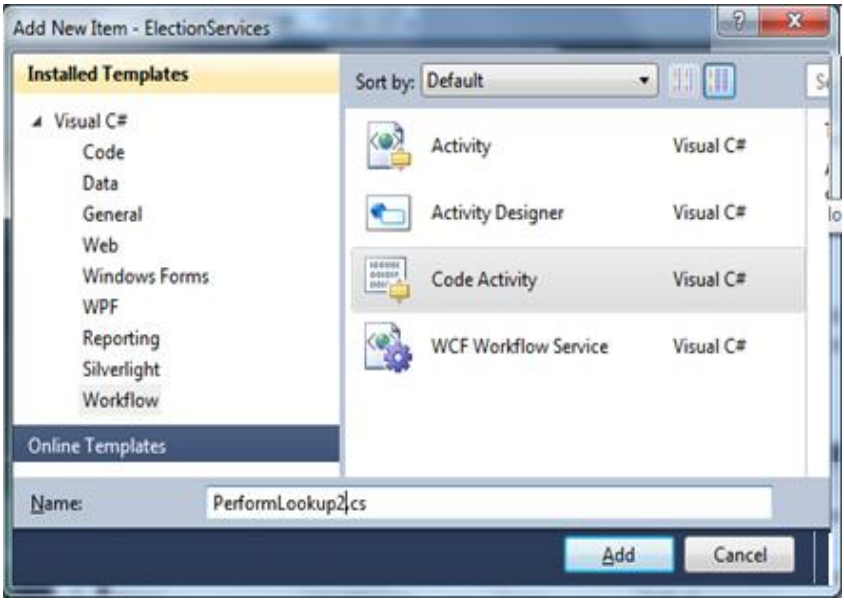


ნახ.6.24. SendResponse პარამეტრების სია

6.8. მოდიფიცირებული PerformLookup ქმედების შექმნა

მომხმარებლის PerformLookup ქმედება, რომელიც ჩვენ შევქმენით პირველი სერვისისთვის, იყენებს ElectionSearch კლასს როგორც შესასვლელ არგუმენტს და აბრუნებს უკან ElectionInfoList კლასს. ახლა ჩვენ უნდა შევქმნათ სხვა სამომხმარებლო ქმედება, რომელიც იყენებს ცალკეულ პარამეტრებს.

Solution Explorer-ში მაუსის მარჯვენა ღილაკით ვაჭერთ ElectionServices პროექტზე და ვირჩევთ Add ➤ New Item. ვირჩევთ შაბლონიდან Code Activity-ს და სახელისთვის Name: PerformLookup2.cs (ნახ.2.25). ამ ქმედების რეალიზაცია ნაჩვენებია 6.3 ლისტინგში.



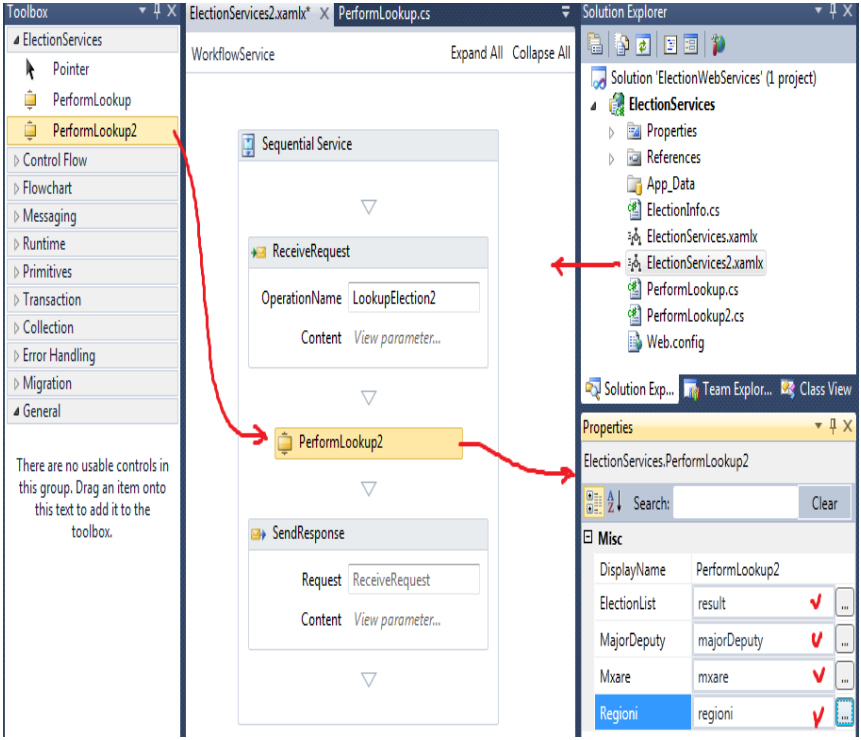
ნახ.6.25-1

```
// ლისტინგი -- 6.3 ----PerformLookup2.cs. რეალიზაცია----
using System;
using System.Collections.Generic;
using System.Activities;

namespace ElectionServices
{
   /**************************************************/
   // This custom activity creates a ElectionInfo array and
   // uses the input parameters to "lookup" the matching
   // items. The ElectionInfo array is returned in the output
   // parameter.
   /**************************************************/
  public sealed class PerformLookup2 : CodeActivity
   {
      // Define an activity input argument of type string
      public InArgument<String> Regioni { get; set; }
      public InArgument<String> Mxare { get; set; }
      public InArgument<String> MajorDeputy { get; set; }
      public OutArgument<ElectionInfo[]> ElectionList { get; set; }

      // If your activity returns a value, derive from CodeActivity<TResult>
      // and return the value from the Execute method.
      protected override void Execute(CodeActivityContext context)
      {
         // Obtain the runtime value of the Text input argument
         string regioni = Regioni.Get(context);
         string mxare = Mxare.Get(context);
         string majorDeputy = MajorDeputy.Get(context);
         ElectionInfo[] l = new ElectionInfo[4];
         l[0] = new ElectionInfo(regioni, mxare, majorDeputy, "Available");
         l[1] = new ElectionInfo(regioni, mxare, majorDeputy, "CheckedOut");
         l[2] = new ElectionInfo(regioni, mxare, majorDeputy, "Missing");
         l[3] = new ElectionInfo(regioni, mxare, majorDeputy, "Available");
         ElectionList.Set(context, l);
      }
   }
}
```

ეს კოდი მუშაობს ისევე, როგორც პირველი, იმისგან განსხვავებით, რომ შემავალი არგუმენტები მიეწოდება ცალ-ცალკე, და შედეგები ბრუნდება მასივად და არა კლასში.

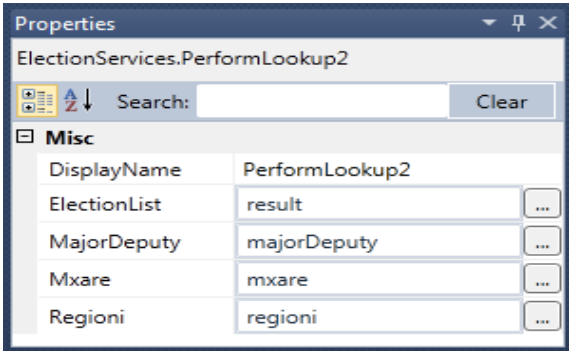
F6-ის დაჭერით განვაახლოთ solution.

ავირჩიოთ BookInventory2.xamlx ფაილი და გადმოვიტანოთ PerformLookup2 ქმედება ინსტრუმენტების პანელიდან „ReceiveRequest” და „SendResponse” ქმედებებს შორის (ნახ.6.25-2).



ნახ.6.25-2.

თვისებების ფანჯარაში შევიტანოთ შესაბამისი მნიშვნელობები, როგორც 6.26 ნახაზზეა ნაჩვენები.

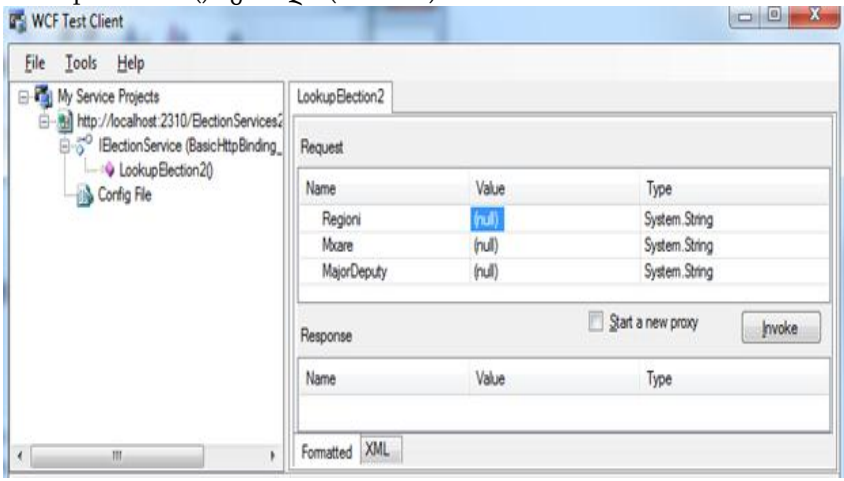


ნახ.6.26. PerformLookup2 ქმედების Properties ფანჯარა

სერვისის ტესტირება

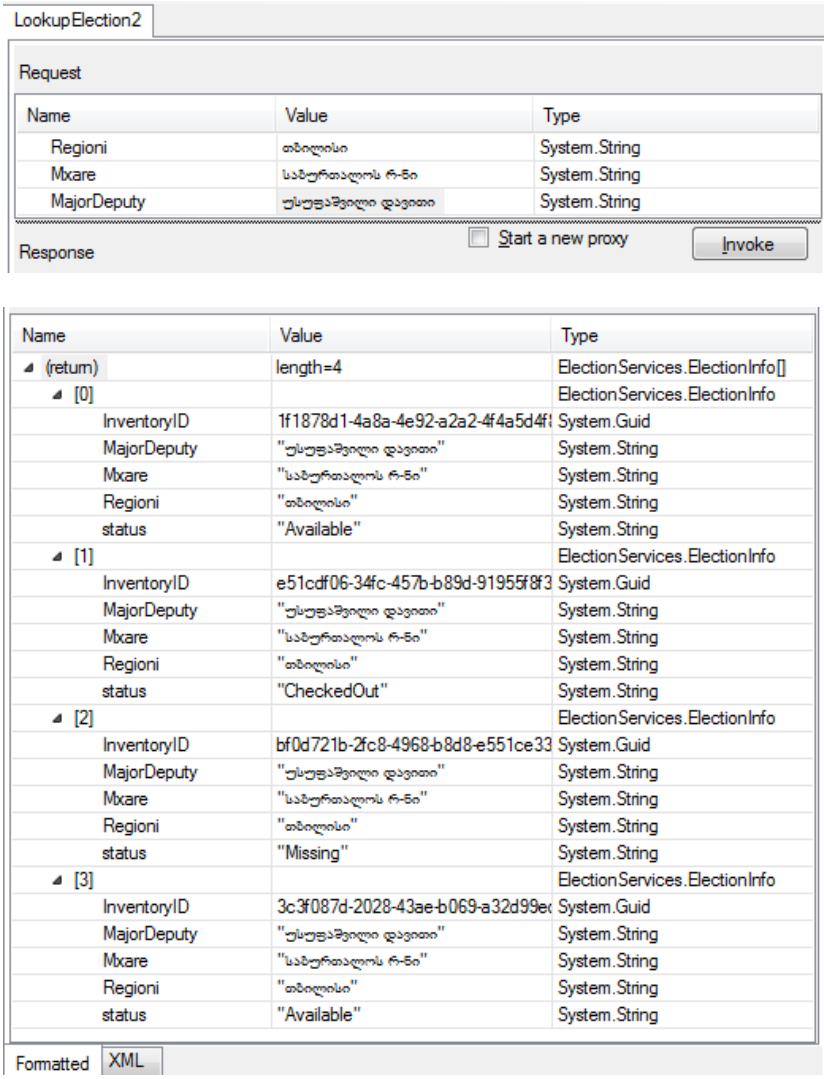
დავრწმუნდეთ, რომ ElectionServices2.xamlx ფაილი არის აქტიური Visual Studio-ში და F5-ით გავუშვათ debug-ის პროცესი კოდის გასამართად.

WCF Test Client უტილიტა უნდა ამოქმედდეს ისე, როგორც პირველი სერვისის შემთხვევაში. 2-ჯერ დავაწკაპუნოთ LookupElection2() მეთოდი (ნახ.6.27).



ნახ.6.27.

შევიტანოთ მოთხოვნის მონაცემები, და დავაჭიროთ Invoke ღილაკს. შედეგებს ექნება 6.28 ნახაზზე ნაჩვენები სახე.



ნახ.6.28. WCF Test Client

შესაბამისი XML ფაილები მოცემულია 6.29 ნახაზზე.

LookupElection2

Request

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1" xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">
                http://tempuri.org/IElectionService/LookupElection2</Action>
  </s:Header>
  <s:Body>
    <LookupElection2 xmlns="http://tempuri.org/">
      <Regioni>თბილისი</Regioni>
      <Mxare>საბურთალოს რ-ნი</Mxare>
      <MajorDeputy>უსუფაშვილი დავითი</MajorDeputy>
    </LookupElection2>
  </s:Body>
</s:Envelope>
```

Response

```
  <LookupElection2Response xmlns="http://tempuri.org/">
    <Result xmlns:a="http://schemas.datacontract.org/2004/07/ElectionServices
                    " xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
      <a:ElectionInfo>
        <a:InventoryID>1f1878d1-4a8a-4e92-a2a2-4f4a5d4f84c0</a:InventoryID>
        <a:MajorDeputy>უსუფაშვილი დავითი</a:MajorDeputy>
        <a:Mxare>საბურთალოს რ-ნი</a:Mxare>
        <a:Regioni>თბილისი</a:Regioni>
        <a:status>Available</a:status>
      </a:ElectionInfo>
      <a:ElectionInfo>
        <a:InventoryID>e51cdf06-34fc-457b-b89d-91955f8f34c2</a:InventoryID>
        <a:MajorDeputy>უსუფაშვილი დავითი</a:MajorDeputy>
        <a:Mxare>საბურთალოს რ-ნი</a:Mxare>
        <a:Regioni>თბილისი</a:Regioni>
        <a:status>CheckedOut</a:status>
      </a:ElectionInfo>
      <a:ElectionInfo>
        <a:InventoryID>bf0d721b-2fc8-4968-b8d8-e551ce336d81</a:InventoryID>
        <a:MajorDeputy>უსუფაშვილი დავითი</a:MajorDeputy>
        <a:Mxare>საბურთალოს რ-ნი</a:Mxare>
        <a:Regioni>თბილისი</a:Regioni>
        <a:status>Missing</a:status>
      </a:ElectionInfo>
      <a:ElectionInfo>
        <a:InventoryID>3c3f087d-2028-43ae-b069-a32d99ed07ba</a:InventoryID>
        <a:MajorDeputy>უსუფაშვილი დავითი</a:MajorDeputy>
        <a:Mxare>საბურთალოს რ-ნი</a:Mxare>
        <a:Regioni>თბილისი</a:Regioni>
        <a:status>Available</a:status>
      </a:ElectionInfo>
    </Result>
  </LookupElection2Response>
</s:Body>
</s:Envelope>
```

Formatted XML

ნახ.6.29.

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

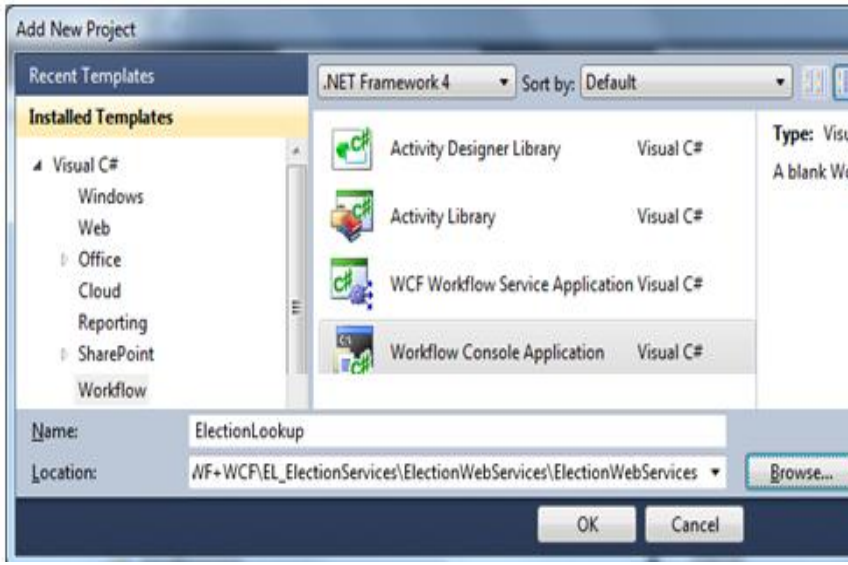
მეორე სერვისისთვის არ შეგვიქმნია კონტრაქტი მომსახურებისთვის. ჩვენ მხოლოდ განვსაზღვრეთ პარამეტრები, რომლებიც გადასცა და უკან დაიბრუნა სერვისმა. კონტრაქტი სერვისის მიწოდებისთვის იქმნება ავტომატურად.

შენიშვნა: Receive/SendReply წყვილის განსაზღვრისას, გვეძლევა არჩევანის უფლება: შეტყობინებები ან პარამეტრები. ამ ორი ვარიანტის შერევა დაუშვებელია. თუ ვიყენებთ პარამეტრებს ქმედების მისაღებად, მაშინ არ შეიძლება შეტყობინების გამოყენება SendReply ქმედებისთვის. ამავდროულად, პარამეტრების გამოყენებისას ტიპებს არ უნდა ჰქონდეს ატრიბუტი MessageContract. წინააღმდეგ შემთხვევაში ფიქსირდება საკმაოდ ხანგრძლივი განსაკუთრებული შემთხვევა შესრულების პროცესში.

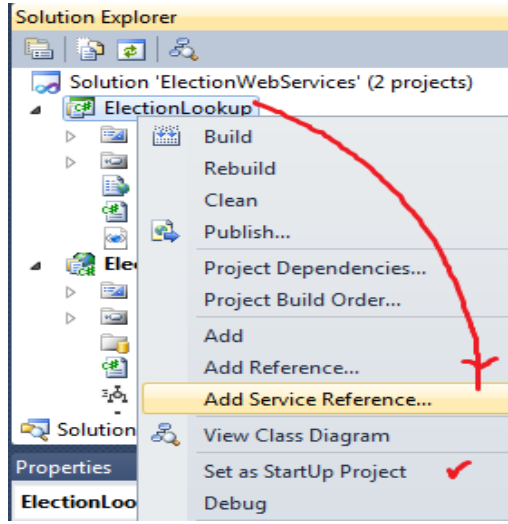
6.9, კლიენტის ბიზნესპროცესის შექმნა

ახლა უნდა შევქმნათ Client Workflow, რომელიც გამოიძახებს Web-სერვისებს. Solution Explorer-ში მარჯვენა ღილაკით ვირჩევთ Add ➤ New Project. შემდეგ Workflow Console Application-ს და შეგვაქვს პროექტის სახელი: ElectionLookup (ნახ.6.30).

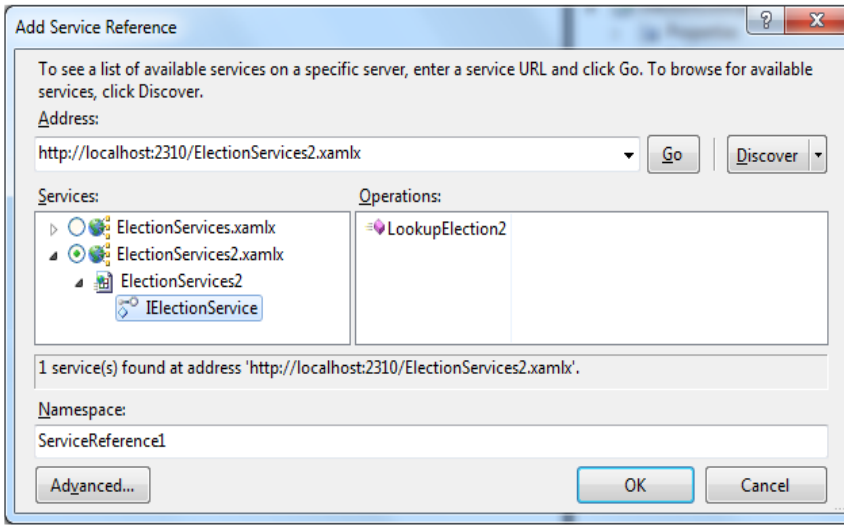
Solution Explorer-ში ElectionLookup project-ზე მარჯვენა ღილაკის დაჭერით და Add Service Reference არჩევით (ნახ.6.31), მივიღებთ 6.32 ნახაზს.



ნახ.6.30. console application პროცესის დამატება



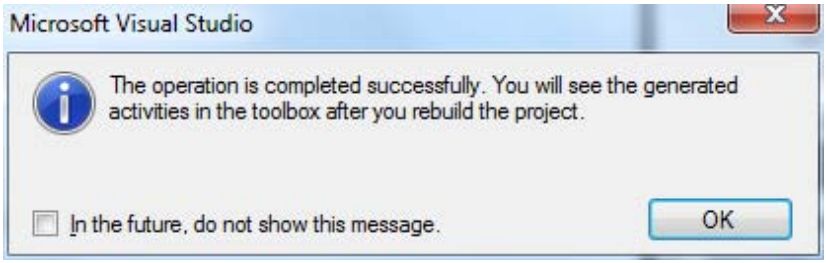
ნახ.6.31.



ნახ.6.32. არსებული სერვისების ძებნა

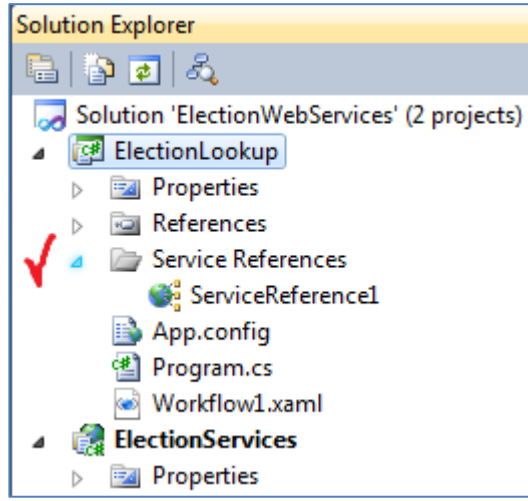
დავაჭიროთ Discover ლინკის ღილაკს და ავიჩიოთ სერვისები Solution-ში. ნახაზზე დიალოგის სიაში ჩანს ორი ჩვენ მიერ შექმნილი სერვისი ElectionServices პროექტში.

შესაძლებელია ამ სერვისების გაფართოება, რათა შესაძლებელი გახდეს თითოეულში გათვალისწინებული მეთოდის დანახვა. ავირჩიოთ მეორე, BookInventory2.xamlx და OK. რამდენიმე წამის შემდეგ უნდა გამოჩნდეს დიალოგური ფანჯარა (ნახ.6.33).



ნახ.6.33. დასრულებული დიალოგის პროცესი

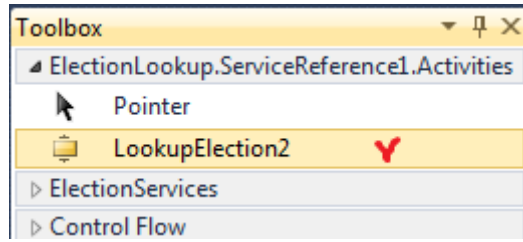
ეს გვაგებინებს, რომ მიმართვა სერვისზე დამატებულია პროექტში (ნახ.6.34).



ნახ.6.34.

F6-ით განახლდება solution.

უნდა აისახოს ფაილი Workflow1.xaml. წინააღმდეგ შემთხვევაში გავხსნათ იგი. ინსტრუმენტების პანელის ზედა ნაწილში უნდა მივიღოთ 6.35 ნახაზის შესაბამისი ინსტრუმენტების პანელი.



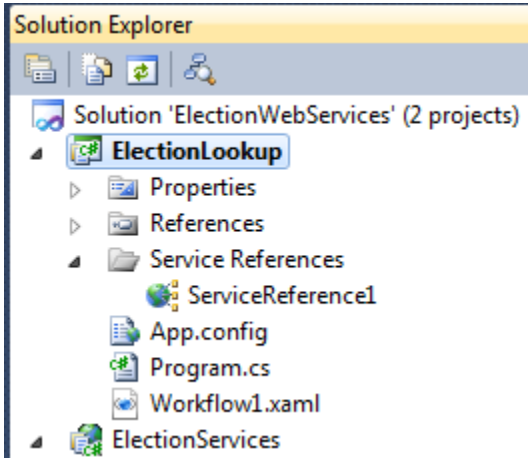
ნახ.6.35. განახლებული Toolbox სერვისების გარსით

ინსტრუმენტების პანელზე სახელსივრცე

ElectionLookup.ServiceReference1.Activities

შეიცავს მომხმარებლის ქმედებას ყოველი მეთოდისთვის სერვისში. ჩვენ შემთხვევაში არის მხოლოდ ერთი: LookupBook2.

Solution Explorer-ში მაუსის მარჯვენა ღილაკით ElectionLookup პროექტზე დავაჭიროთ და ავირჩიოთ Set as Startup Project. მივიღებთ გააქტიურებულ ElectionLookup სტრიქონს (ნახ.6.36).



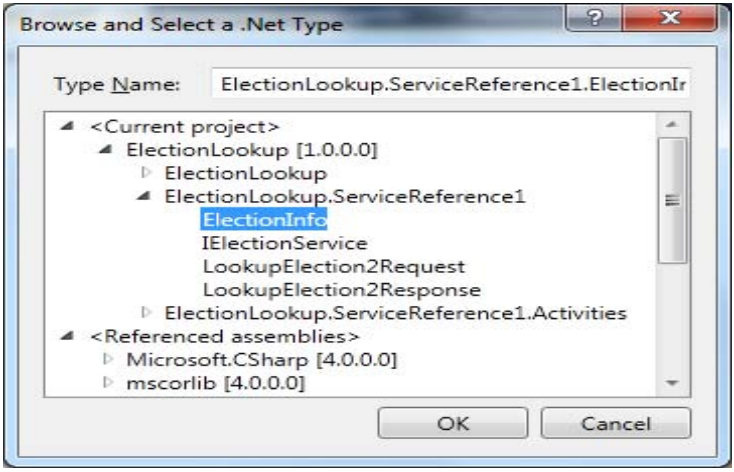
ნახ.6.36.

6.10. ბიზნესპროცესის განსაზღვრა

გადავიტანოთ LookupBook2 ქმედება მუშა პროცესზე. შემდეგ დავაყენოთ არგუმენტები, რათა შესაძლებელი იყოს საძებნი კრიტერიუმების გადაცემა მუშა პროცესში და შედეგების დაბრუნება. დააჭირეთ Arguments მართვის ელემენტს.

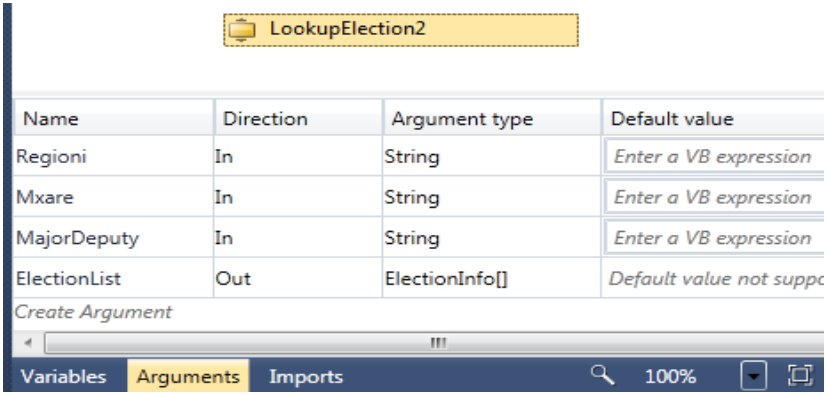
დავამატოთ სამი String შემავალი არგუმენტი სახელით Regioni, Mxare და MajorDeputy. დავამატოთ გამომავალი არგუმენტი, სახელით ElectionList.

არგუმენტის ტიპისთვის ავირჩიოთ Array Of[T]. გამოსულ დიალოგურ ფანჯარაში ვირჩევთ Browse for Types და შემდეგ ElectionInfo კლასს ElectionLookup.ServiceReference1.ElectionServices ნაკრებიდან (ნახ.6.37).



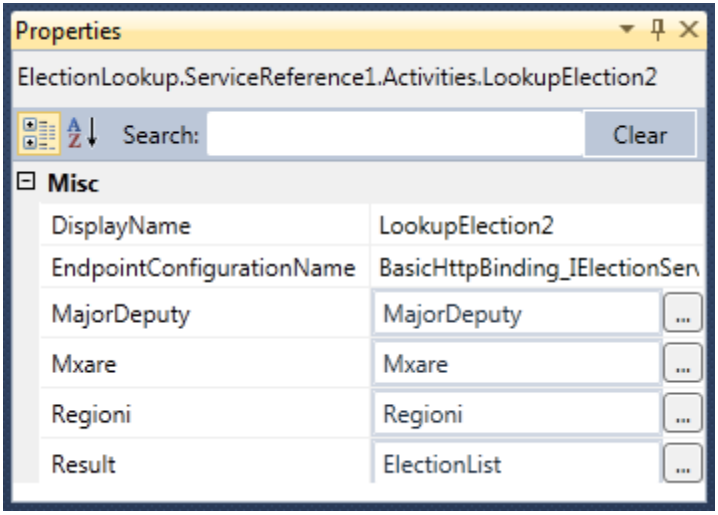
ნახ.6.37.

არგუმენტების სია მოცემულია 6.38 ნახაზზე.



ნახ.6.38. ბიზნეს პროცესის არგუმენტები

ავირჩიოთ “LookupBook2” ქმედება; Properties ფანჯარაში შევიტანოთ თვისებების values , როგორც 6.39 ნახაზზეა ნაჩვენები.



ნახ.6.39. LookupBook2 თვისებები

6.11. ჰოსტ-აპლიკაციის რეალიზაცია და ამუშავება

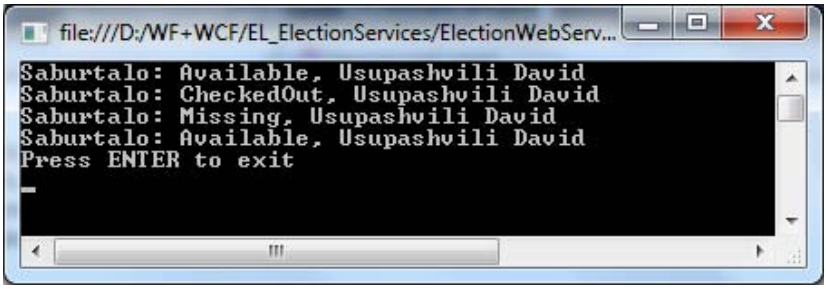
გავხსნათ Program.cs ფაილი LookupBook პროექტში. ამ ფაილის რეალიზაცია ნაჩვენებია 10.4 ლისტინგში.

```
//- ლისტინგი 6.4 ---   Program.cs ფაილი ------------
using System;
using System.Linq;
using System.Activities;
using System.Activities.Statements;
using System.Collections.Generic;
using ElectionLookup.ServiceReference1;

namespace ElectionLookup
{
```

```
class Program
{
    static void Main(string[] args)
    {
        // create dictionary with input arguments for the workflow
        IDictionary<string, object> input = new Dictionary<string, object>
        {
            { "MajorDeputy" , "Usupashvili David" },
            { "Mxare" , "Saburtalo" },
            { "Regioni" , "Tbilisi" }
        };

        // execute the workflow
        IDictionary<string, object> output =
            WorkflowInvoker.Invoke(new Workflow1(), input);

        ElectionInfo[] l = output["ElectionList"] as ElectionInfo[];
        if (l != null)
        {
            foreach (ElectionInfo i in l)
            {
                Console.WriteLine("{0}: {1}, {2}", i.Mxare, i.status, i.MajorDeputy);
                //Console.WriteLine("{0}: {1}, {2}", i.Mxare, i.status, i.InventoryID);
            }
        }
        else
            Console.WriteLine("No items were found");

        Console.WriteLine("Press ENTER to exit");
        Console.ReadLine();
    }
}
}
```

ეს კოდი გადასცემს MajorDeputy-, Mxare- და status-ში არგუმენტებს ობიექტის ლექსიკონის საშუალებით. ბიზნეს-პროცესი აბრუნებს უკან ElectionInfo ობიექტების მასივს. ამ კოდს გამოაქვს ეკრანზე ამ მასივის შინაარსი.

F5-ით გაიშვება პროგრამა. შედეგებს უნდა ჰქონდეს ასეთი სახე (ნახ.6.40).



ნახ.6.40. აპლიკაციის მუშაობის შედეგი

6.12. Pick-ის გამოყენება

WF 4.0 უზრუნველყოფს ქმედებას, სახელით Pick (არჩევა), რომელსაც აქვს რამდენიმე შტო (PickBranch). ყოველი შტო შეიცავს ტრიგერის თვისებას და აქციის თვისებას. თითოეული მათგანი ასრულებს ქმედებას (ან ქმედებათა მიმდევრობას). როდესაც Pick ქმედება შესრულდა, ტრიგერის ყველა ქმედება დაწყებულია. როგორც კი ამათგან ერთი ქმედება დასრულდება, მისი შესაბამისი აქციები შესრულებულია და ყველა სხვა შტო გაუქმებულია.

ეს სასარგებლოა შესაბამისი აქციების განსაზღვრისათვის რომელიმე მოვლენის საფუძველზე. მაგალითად, შეიძლება მონაცემთა მიღების (Receive) ქმედების გამოყენება ტრიგერისთვის.

ყოველ შტოს შეიძლება ჰქონდეს Receive ქმედება, რომელიც ელოდება სხვა შეტყობინებას, რომლის საფუძველზეც შესრულებულია შეტყობინების შესაბამისი აქცია.

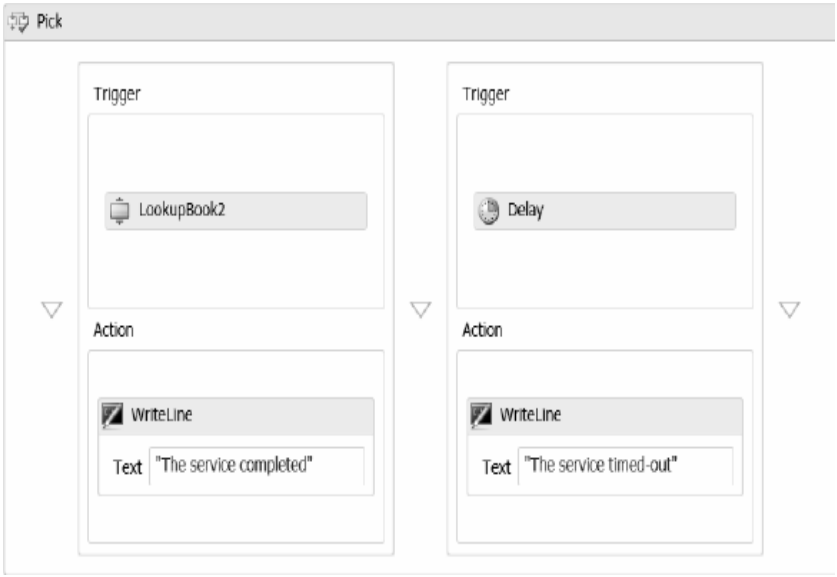
ამ პროექტისთვის გამოიყენება Pick ქმედება, რათა უზრუნველყოფილ იქნას ტაიმ-აუტის ფუნქცია ბიზნეს-პროცესისთვის.

რჩევა. თუ თქვენ იცნობთ მუშა პროცესის წინა ვერსიებს, ეს ეკვივალენტურია, მაგალითად, Listen ქმედების.

გავხსნათ Window1.xaml ფაილი, მაუსის მარჯვენა ღილაკით „LookupBook2” ქმედებაზე დავაჭიროთ და ავირჩიოთ Cut (ამოჭრა). გადავიტანოთ Pick ქმედება მუშა პროცესზე. მაუსის მარჯვენა ღილაკით დავაჭიროთ პირველი განყოფილების Trigger-ს და ავირჩიოთ ბრძანება Paste (ჩასმა).

გადავიტანოთ დაყოვნების ქმედება (Delay) მეორე შტოს Triggeg-ის განყოფილებაში. დააყენეთ მისი ხანგრძლივობის (Duration) თვისება TimeSpan.FromSeconds(5).

გადავიტანოთ WriteLine ქმედება ყოველი აქციის სექციაში და მივცეთ Text-თვისება „სრული სერვისი“ და „სერვისი ტაიმ-აუტით“. ბიზნესპროცესი უნდა გამოიყურებოდეს 6.41 ნახაზზე მოცემული სახით.



ნახ.6.41. ბიზნესპროცესი timeout ლოგიკით

F5-ით ავამუშავოთ აპლიკაცია. შედეგები უნდა იყოს პროგრამის ბოლო გაშვების იდენტური გამორიცხვის დამატებული შეტყობინებით („სრული სერვისი“). ტაიმ-აუტის ფუნქციის შესამოწმებლად გავხსნათ ElectionServices2.xamlx ფაილი.

გადმოვიტანოთ დაყოვნების Delay ქმედება SendResponse ქმედების დაწყების წინ და დავაყენოთ მისი ხანგრძლივობის Duration თვისება TimeSpan.FromSeconds(7). F5-ით ავამუშავოთ აპლიკაცია. 5 წამის დაყოვნების შემდეგ შედეგებს ექნება ასეთი სახე:

```
The service timed-out
No items were found

Press ENTER to exit
```

6.13. რეზიუმე

ბიზნესპროცესები ხშირად განაწილებულია სხვადასხვა აპლიკაციებში და სხვადსხვა სერვერებზეც კი, ამიტომაც კომუნიკაცია არის მნიშვნელოვანი ნაწილი დიზაინის ბიზნეს-პროცესის. მაგალითად, პროექტში „ელექტრონული საარჩევნო სისტემა“ მისი სხვადასხვა ფილიალები [ცენტრალური საარჩევნო კომისია, რეგიონი, მხარე (ოლქი), საარჩევნო უბანი და სხვ.] ურთიერთქმედებენ ერთმანეთთან, რათა მოითხოვონ ელემენტები, რომლებიც ექვემდებარება გადაცემას.

Send და Receive ქმედებები (და მათი „კოლეგები“ ReceiveReply და SendReply) უზრუნველყოფს შეტყობინებათა მოსახერხებელი ფორმით გადაცემას და მიღებას. ეს ქმედებები ეყრდნობა WCF-ს შეტყობინებების გადასაცემად და შეუძლია გამოიყენოს რიგი პროტოკოლებისა, როგორიცაა HTTP ან TCP. მიუხედავად ამისა, ჰოსტ-აპლიკაციას შეუძლია ასევე უშუალოდ მიიღოს WCF-შეტყობინება.

მიუხედავად იმისა, რომ ბიზნესპროცესის ქმედებებს არ აქვს მომხმარებლის ინტერფეისი, ისინი ხშირად საჭიროებს ურთიერთობას ჰოსტ აპლიკაციასთან ან აპლიკაციის განახლებასთან, ან მოითხოვოს მონაცემებს, რომლებიც შეიტანება მომხმარებელთა მიერ. აქ ვიყენებდით სანიშნეს (Bookmark), რათა შეჩერებულიყო ბიზნესპროცესი მომხმარებელთა მონაცემების შეტანამდე. აპლიკაციას ადვილად შეუძლია ბიზნესპროცესის განახლება, სადაც იგი შეწყდა.

გამოიყენებოდა ასევე WriteLine ქმედება, კონსოლზე ტექსტის გამოსატანად. ნაჩვენები იყო ამ ქმედების გამოყენება აპლიკაციის სიაში (listbox).

Web-სერვისების გამოყენება დაპროექტების სულ უფრო პოპულარული მიდგომა ხდება. საქმე შეიძლება დაიწყოს კონტრაქტიდან მომსახურების მიღებაზე ან უბრალოდ განისაზღვროს შემავალი და გამომავალი პარამეტრები, მუშა პროცესების კონსტრუქტორის გამოყენებით.

ბიზნესპროცესის გამოყენება შეიძლება როგორც სერვისის შესაქმნელად, ისე მათ გამოსაყენებლად. მეთოდები, რომლებიც უზრუნველყოფილია Web-სერვისებით, ხდება ტრადიციული, სტანდარტული ქმედებები, რომელთა გამოყენება შესაძლებელია ბიზნესპროცესებში.

დასკვნა

ამრიგად, ჩატარებული სამუშაოს შედეგად შეიძლება შემდეგი სახის დასკვნების ჩამოყალიბება:

1. ელექტრონული საარჩევნო სისტემა, როგორც კორპორაციული მართვის ობიექტი ელექტრონული მთავრობის შემადგენლობაში, მიეკუთვნება რთული და დიდი სისტემების კლასს. მისმა კომპლექსურმა ანალიზმა გვიჩვენა, რომ ასეთი სისტემების დასაპროექტებლად აუცილებელია ობიექტ-ორიენტირებული, პროცესორიენტირებული და სერვის-ორიენტირებული მოდელირების მეთოდების გამოყენება;

2. საქართველოში არსებული ტრადიციული საარჩევნო სისტემის დიაგნოსტიკური გამოკვლევის საფუძველზე, აგრეთვე საზღვარგარეთული დემოკრატიული ინსტიტუტების გამოცდილების გათვალისწინებით ამ სფეროში, განსაზღვრულ იქნა ძირითადი პრობლემები და ამოცანები, შემუშავდა მათი გადაწყვეტის გზები და ერთიანი ელექტრონული საარჩევნო სისტემის აგების კონცეფცია;

3. ელექტრონული საარჩევნო სისტემის დაპროექტება, აპარატურული და პროგრამული რეალიზაცია უნდა განხორციელდეს მულტიმედიური საშუალებების, განაწილებული, რელაციური ბაზების, კლიენტ-სერვერ არქიტექტურის და უსაფრთხო, საიმედო ქსელების ბაზაზე;

4. ელექტრონული საარჩევნო სისტემის მულტიმედიური მონაცემთა რელაციური ბაზების ეფექტური დაპროექტებისთვის და ასაგებად მიზანშეწონილია თანამედროვე CASE-ტექნოლოგიების გამოყენება, მოდელირების კატეგორიული თეორიის და დაპროექტების ობიექტორიენტირებული მეთოდების საფუძველზე;

5. კლიენტ-სერვერ არქიტექტურის ლოგიკურად ერთიანი და ფიზიკურად განაწილებული მონაცემთა რელაციური ბაზების სისტემის დამუშავება შესაძლებელი გახდა ობიექტროლური მოდელირების პრინციპების საფუძველზე და შესაბამისი გრაფო–ანალიზური ინსტრუმენტების გამოყენებით;

6. აგებული სისტემის ბიზნესპროცესების ფუნქციურობის შეფასების მიზნით გამოყენებულია იმიტაციური მოდელირების და ანალიზის მექანიზმები. კერძოდ, შემოთავაზებულია ფერადი პეტრის ქსელების (CPN) გამოყენება ელექტრონული საარჩევნო სისტემის იმიტაციური მოდელის ასაგებად და მისი ფუნქციონირების დროითი მახასიათებლების გამოსაკვლევად, როგორც მასობრივი მომსახურების სისტემა;

7. აუცილებელია ელექტრონული საარჩევნო სისტემის ქსელის უსაფრთხო და საიმედო მუშაობის პრინციპების რეალიზაცია თანამედროვე კომუნიკაციური საშუალებების ბაზაზე. ამ მიზნით შემოთავაზებულია დაცული სახელმწიფო ქსელის აგების კონცეფცია და მისი არქიტექტურა VPN ტექნოლოგიის საფუძველზე;

8. კორპორაციული განაწილებული მულტიმედიური ელექტრონული საარჩევნო სისტემის ექსპერიმენტული საპილოტო ვერსიის პროგრამული რეალიზაცია განხორციელდა ახალი ინფორმაციული ტექნოლოგიების ბაზაზე, კერძოდ, Visual Studio.Net, WPF, Workflow, WCF, SQL Server, ORM/ERM, CPN, VPN ტექნოლოგიების და პროგრამული პაკეტების გამოყენებით;

9. ელექტრონული საარჩევნო სისტემის საპილოტო პროექტის ვერსია პროგრამულად რეალიზებულ იქნა კლიენტ-სერვერ არქიტექტურის და მონაცემთა განაწილებული ბაზების მართვის სისტემის პრინციპების საფუძველზე. გამოყენებულია Windows Presentation Foundation (WPF) და Windows Communication Foundation (WCF) ტექნოლოგიები;

10. დამუშავებულ იქნა სერვისორიენტირებული არქიტექტურის (SOA) პროგრამული რეალიზაცია ელექტრონული საარჩევნო სისტემისთვის ინფორმაციის უსაფრთხო გადაცემის გათვალისწინებით. გამოყენებულ იქნა Workflov და Windows Communication Foundation (WCF) ტექნოლოგიები;

11. შემუშავებულია აგებული სისტემის მომხმარებელთა ინტერფეისები, ინსტრუქციები, დანერგვის და ექსპლუატაციის პროცესების ორგანიზაციული, ტექნიკური და იურიდიული ასპექტები. განსაზღვრულია სისტემისათვის საჭირო ტექნიკური საშუალებები და მათი შესაძლებლობები;

12. შედგენილია სავარაუდო ბიუჯეტი და ხარჯთა კალკულაცია მულტიმედიური ელექტრონული საარჩევნო სისტემის დასაპროექტებლად, ასაგებად და დასანერგად.

ლიტერატურა:

1. Surguladze G., Turkia E., Topuria N., Basiladze G. Automation of Business-Processes of an Election System. VI Intern. Conf. (AICT 2012). Application of Information and Communic. Technologies. ISBN 978-1-4673-1740-5. Tb., Georgia, 2012. pp. 308-312.

2. Basiladze G. Multimedia Databases and IT-Infrastructure of an Electronic Election System. Intern. Sc.Conf.: “Automated Control Systems & new IT” . 2011. გვ. 161–162.

3. Turkia E., Topuria N., Basiladze G. Construction of Multi-dimensional Analysis Packet of Commercial Objects with Decision Cube Components. Application of information and Communication Technologies (PCI’ 2012) Tbilisi, Georgia, 17-19.10.2012 p. 16-19.

4. სურგულაძე გ., ბასილაძე გ., ოხანაშვილი მ. პროდუქციის მიწოდების პროცესის იმიტაციური მოდელირება ფერადი პეტრის ქსელებით. სტუ-ის შრ. კრ. „მართვის ავტომატიზებული სისტემები“, N1(6), თბილისი, 2009, გვ. 62-69.

5. პეტრიაშვილი ლ., ოხანაშვილი მ., აბრამიშვილი ნ., ბასილაძე გ. მულტიმედიურ მონაცემთა ბაზაში რელატიური დოკუმენტის იდენტიფიკაცია. სტუ-ის შრ. კრ. „მართვის ავტომატიზებული სისტემები“, N2(9), თბილისი, 2010, გვ. 82-86.

6. პეტრიაშვილი ლ., ვაჭარაძე ი., ბასილაძე გ. გადაწყვეტილების მიღების მხარდამჭერ საინფორმაციო სისტემებში OLAP კონცეფციის ერთი რეალიზაციის შესახებ. სტუ-ის შრ. კრ. „მართვის ავტომატიზებული სისტემები“, N1(4), თბილისი, 2008, გვ. 103-107.

7. სურგულაძე გ., ბასილაძე გ., გაბინაშვილი ლ. მულტიმედიური ელექტრონული საარჩევნო სისტემის პროგრამული უზრუნველყოფის დამუშავება. სტუ-ის შრ. კრ.

„მართვის ავტომატიზებული სისტემები“, N1(14), თბილისი, 2013, გვ. 234-239.

8. პეტრიაშვილი ლ., ბასილაძე გ., კორპორაციულ სისტემებში მონაცემთა მონიტორინგი და ანალიზი OLAP ModelKit ტექნოლოგიის გამოყენებით. სტუ-ის შრ. კრ. „მართვის ავტომატიზებული სისტემები“, N1(12), თბილისი, 2012, გვ. 104-108.

9. თოფურია ნ., ბასილაძე გ. ბიზნეს-ანალიზის ინსტრუმენტები კორპორატიული ორგანიზაციებისათვის. სტუ-ის შრ. კრ. „მართვის ავტომატიზებული სისტემები“, N2(13), თბილისი, 2012, გვ. 95-99.

10. სურგულაძე გ., პეტრიაშვილი ლ. მონაცემთა საცავის აგების ტექნოლოგია ინტერნეტული ბიზნესის სისტემებისათვის. სტუ. თბ., 2005.

11. ჩაჩანიძე გ., ნანობაშვილი ქ., ბასილაძე გ. ქალაქის ბიუჯეტის ავტომატიზებული მართვის ეკონომიკურ-სამართლებრივი საფუძვლების ინფორმაციული ბაზის კონცეპტუალური მოდელი. ჟურნალი „ინტელექტი“, N1(24). თბილისი, 2006. გვ. 45–49.

12. ჩაჩანიძე გ., ნანობაშვილი ქ., ბასილაძე გ. ქალაქის ბიუჯეტის ავტომატიზებული მართვის ეკონომიკური საფუძვლების მათემატიკური მოდელი. ჟურნალი „ინტელექტი“, N2(25). თბილისი, 2006. გვ. 100-103.

13. ბულია ი., სურგულაძე გ., თურქია ე. ვებ-აპლიკაციების დამუშავება მონაცემთა ბაზების საფუძველზე (ADO.NET, ASP.NET, C#). ISBN 978-9941-14-289-5. სტუ, თბ., 2009. 172 გვ.

14. Krafzig D., Banke K., Slama D. Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall. 2004.

15. Anonymous Digital Identity in e-Government. Dissertation, Der Universität Zürich 2004.

16. თურქია ე. ბიზნესპროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. სტუ. თბ., 2010.

17. თურქია ე., გიუტაშვილი მ. კორპორაციულ სისტემებში ინტელექტუალური რესურსების მენეჯმენტი. სტუ. თბ., 2008.

18. Clarke R. Electronic Data Interchange (EDI): An Introduction. http://www.roger clarke.com/EC/EDIIntro.html.

19. Index Structures for Similarity Search in Multimedia Databases Dissertation Universität Konstanz, Fachbereich Informatik 2006.

20. Freund J. Zwölf Fragen zum Business-Process-Management. http://wiki. computerwoche.de/doku.php/soa_bpm/bpm_faq.

21. Seeley R. Business-side often drives BPM initiatives today, http://searchsoa.techtarget.com/news/1323855/Business-side-often-drives-BPM-initiatives-today.

22. ცენტრალური საარჩევნო კომისიის ვებ-გვერდი http://cec.gov.ge/index.php?lang_id=GEO&sec_id=181.

23. Magal S.R., Word J. Essentials of Business Processes and Information Systems. Wiley.

24. Erl T. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall.

25. Chappell D. Enterprise Service Bus: Theory in Practice. O'Reilly Media.

26. 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.

27. Goel A. Enterprise Integration EAI vs. SOA vs. ESB, http://ggatz.com/images/Enterprise_20Integration_20-_20SOA_20vs_20 EAI_20vs_20ESB.pdf.

28. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley.

29. Ruh A. W., Maginnis X. F., Brown J. W. Enterprise Application Integration: A Wiley Tech Brief. Wiley.

30. Bussler Ch. B2B Integration: Concepts and Architecture. Springer.

31. Oracle Enterprise Application Integration Services. http://www.oracle.com/ us/products/consulting/resource-library/enter-prise-application-integration-069324.pdf.

32. Meyer-Wegener K. Multimediale Datenbanken- B. G. Teubner Stuttgart· Leipzig· Wiesbaden 2003.

33. Буч Г., Рамбо Дж., Джакобсон А. Язык UML. Руководство пользователя. Пер.с англ., Питер, 2004.

34. Jablonski, S., Petrov, I., Meiler, C., Mayer, U. Guide to Web Application and Platform Architectures, Germany, Springer-Verlag Berlin And Heidelberg Gmbh, 2004.

35. Burad A. Multimedia Databases. Indian Institute of Technology, April 6. 2006.

36. https://www.cs.ualberta.ca/research/research-areas/computer-vision-and-multimedia-communications გადამოწმდა 15.02.2013.

37. ვედეკინდი ჰ., სურგულაძე გ., თოფურია ნ. განაწილებული ოფის-სისტემების მონაცემთა ბაზების დაპროექტება და რეალიზაცია UML-ტექნოლოგიით. მონოგრ., სტუ, თბ., 2006.

38. სურგულაძე გ., შონია ო., ყვავაძე ლ. მონაცემთა განაწილებული ბაზების მართვის სიტემები. სტუ, თბ., 2004.

39. სურგულაძე გ., დაპროგრამების ვიზუალური მეთოდები და ინსტრუმენტები (UML, MsVisio,C++Builder). სტუ, თბ., 2005.

40. ბოტჰე კ., სურგულაძე გ., დოლიძე თ. და სხვ.. თანამედროვე პროგრამული პლატფორმები და ენები (WindowsNT, Unix, Linux, C++, Java, XML). სტუ. თბ.,2003.

41. სურგულაძე გ., გულუა დ. განაწილებული სისტემების ობიექტ_ორიენტირებული მოდელირება უნიფიცირებული პეტრის ქსელებით. თბილისი: ტექნიკური უნივერსიტეტი, 2005, გვ. 210.

42. http://aceproject.org/ გადამოწმებულია 3.04.2013.

43. რესიგი ვ., სურგულაძე გ., გულუა დ, „ვიზუალური ობიექტ–ორიენტირებული დაპროგრამების მეთოდები“. სტუ, თბილისი, 2002.

44. Jablonski S. Projekt-ForFlow. http://www.ai4.uni-bayreuth.de /en/research/projects/010_forflow/index.html. გადამოწმ. 01.05.14.

45. http://www.ida.liu.se/divisions/adit/dwis/publications.shtml გადამოწმდა 5.02.2013.

46. Service-Oriented Architecture: A Primer. Michael S. Pallos. http://www.bijonline.com/PDF/SOAPallos.pdf. გადამოწმდა–25.04.2012.

47. თურქია ე. ელექტრონული ბიზნესისა და ელექტრონული კომერციის სისტემების მოდელირება და დაპროექტება. სტუ, თბილისი, 2009.

48. Open Management Group, Business Process Management Initiative: Business Process Modeling Notation (BPMN), http://www.bpmi.org, 2006.

49. 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.

50. სურგულაძე გ., თოფურია ნ., ყვავაძე ლ. განაწილებული მონაცემთა ბაზების აგების ავტომატიზაცია .NET გარემოში. სტუ შრ. კრ.”მას” N1(2), 2007. გვ.105-108.

51. http://www.vvk.ee/voting-methods-in-estonia/engindex/.

52. Halpin T., ORM 2 Graphical Notation, Neumont University, 2005. http://www.orm.net/ pdf/ORM2_TechReport1.pdf.

53. Paul A. Blythe Sr.-Biomertic Authentification System for Secure Digital Cameras. State University of New York 2005-Dissertation.

54. http://www.cisco.com/en/US/docs/routers/csbr/rvs4000/administration/guide/RVS4000_AG_OL-22605.pdf.

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. 24].

57. A. Heinrich-Information Retrieval, Otto-Friedrich-Universität Bamberg 2001-2008.

58.http://www.cisco.com/en/US/docs/routers/access/2800/software/configuration/guide/2800_guide.pdf .

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., გულუა დ. განაწილებული სისტემების ობიექტ-ორიენტირებული მოდელირება უნიფიცირებული პეტრის ქსელებით. მონოგრაფია, სტუ, თბილისი. 2005.

63. NetMan 3.0 Handbuch, 2 Bände, Göttingen. 2005.

64. სურგულაძე გ., თოფურია ნ., ბასილაძე გ., ურუშაძე ბ., ლომიძე მ., გაბინაშვილი ლ. პროგრამული სისტემების მენეჯმენტი მულტიმედიალური აპლიკაციების დასაპროექტებლად და ასაგებად. VI საერთ. სამეცნ.პრაქტიკული კონფ., „ინტერნეტი და საზოგადოება“. აკ. წერეთლის სახ. უნივერსიტეტი. ქუთაისი, სექტემბერი 2013, გვ. 66-70.

65. Melton J., Eisenberg A. SQL Multimedia and Application Packages (SQL/MM). ACM SIGMOD Record 30 (Dec. 2001) 4 97-1-2.

66. Codd E.F. Further normalization of the database relational model. In Data Base Systems, Courant Computer Science Symposia 6. Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 65-98.

67. Fagin R.. A Normal Form for Relational Databases That Is Based on Domains and Keys. IBM Research Laboratory. ACM Transactions on Database Systems, Vol. 6, No. 3, September. 1981, pp. 387-415.

68. H. Wedekind, G. Surguladze. Technology of Designing of Distributed Systems on the Basis of Objectoriented Programming. ISSN 021-7164, GTU, Tbilissi. 1996. pp.96-100.

69. Сургуладзе Г., Качибая В., Кортуа Т. О выборе приемлемых нормальных форм логической схемы реляционных БД. Сб.научн. тр. ГПИ., “Техническая кибернетика”, N 10(267),Тб., 1983. ст. 47-51.

70. Eisenberg A. New Standard for Stored Procedures in SQL. ACM SIGMOD Record 25. 12, 1996.

71. Christodoulakis S. Multimedia Data Base Management: Application and Problems –A Position Paper. In S. Navathe (Hrsg.), Proc. ACM SIGMOD-85. Conf. on Management of Data (Austin, TX, May 1985). Bd14. No4, pp. 304-305.

72. Сургуладзе Г. Построение структур реляционных баз данных в АСУ. Автореф. канд. диссерт. Эл-Технический Унив., Сн-т Петербург. 1980. 18 ст.

73. Чоговадзе Г., Сургуладзе Г., Качибая В. Теория реляционных зависимостей и проектирование логической схемы баз данных. Тб. Госуд. Унив., Тбилиси. 230 ст.

74. Woelk D., Kim W. Multimedia Information Management in an Object-Oriented Database System. In: Stocker P.M., Kent W. (Hrsg.), Proc.13th Int.Conf. on VLDB (Brighton, England, Sept. 1987). Los Altos, CA: Morgan Kaufmann Publ., 1987. pp.319-329.

75. Mattos N., Meyer-Wegener K., Mitschang B. A Grand Tour of Concepts for Object-oriented from a Database Point of View. Data & Knowledge Engineering 9 (1992/93). pp. 321-352.

76. Cattell R.G., Barry D.K. The Object Database Standard: ODMG 3.0. San Francisco. Morgan Kaufmann Publ. 2000.

77. Bancilhon F. Query Language for Object-Oriented Database Systems: Analysis and a Proposal. In: Härder (Hrsg.), Datenbanksysteme für Büro,Technik und Wissenschaft, Proc.GI/SI-Fachtagung (Zürich, März 1989), InformatikßFachberichte, Nr.204, Berlin, Springer Verlag, 1989, pp. 1-18.

78. Kim W., Garza J.F., Ballou N., Woelk D. Architecture of the ORION Next-Generation Database System. 1990. http://dl.acm.org /citation.cfm?id=627402 .

79. Rakow T.C., Neuhold E.J., Löhr M. Multimedia Database Systems. The Notions and the Issues. 1995. 31p. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.45.4279&rep=rep1&type=pdf .

80. ბასილაძე გ. ელექტრონული საარჩევნო სისტემის მხარდამჭერი IT-ინფრასტრუქტურის დამუშავება. სტუ-ის შრ. კრ.

„მართვის ავტომატიზებული სისტემები“, N1(8), თბილისი, 2010, გვ. 223–226 .

81. მურჯიკნელი გ. კორპორაციულ ქსელში სისტემურ აპლიკაციათა ინტეგრაცია და უსაფრთხო კავშირის უზრუნველყოფა. სამაგისტრო ნაშრომი. სტუ. თბილისი. 2012.

82. Bridging-Transmitting Non-IP Traffic or Merging Two Networks. ftp://ftp.hp.com/pub/networking/software/SR7000dl-Basic-C10-Bridging-Nov2006.pdf. გადამოწმ. 15.06.14.

83. Configuring Network Security with ACLs (Cisco). http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst2960/software/release/12-2_55_se/configuration/guide/scg_2960/swacl.html. გადამოწმ. 15.06.14.

84. სურგულაძე გ., ბიტარაშვილი მ., ქრისტესიაშვილი ხ. პროგრამული აპლიკაციების დეველოპმენტის საფუძვლები (C#, MsAccess, ADO & ASP.NET). ISBN 978-9941-20-251-3. სტუ, თბ., 2013.

85. Wang C., Wedekind H. Segment Synthesis in Logocal Data Base Design. IBM J. RSD 19, N1, January, 1975. pp.71-77.

86. Collins M.J. Beginning WF: Windows Workflow in .NET 4.0. ISBN-13 (pbk): 978-1-4302-2485-3 Copyright © 2010. USA. http://www.ebooks-it.net/ebook/beginning-wf.

87. 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. Вильямс", 2009.

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

89. Windows. https://ru.wikipedia.org/wiki/Windows. გადამოწმ. 15.06.14.

90. Microsoft Windows 1.0. https://ru.wikipedia.org/wiki/Windows_1.0x. გადამოწმ. 15.06.14.

91. Windows7. https://ru.wikipedia.org/wiki/Windows_7. გადამოწმ. 15.06.14.

92. Windows_Server_2008. https://ru.wikipedia.org/wiki/Windows_Server_2008. გადამოწმ. 15.06.14.

93. WindowsVista. https://ru.wikipedia.org/wiki/Windows_Vista. გადამოწმ. 15.06.14.

94. Windows Small Business Server. https://ru.wikipedia.org/wiki/Windows_Small_Business_Server. გადამოწმ. 15.06.14.

95. Windows NT (New Technology). https://ru.wikipedia.org/wiki/Windows_NT. გადამოწმ. 15.06.14

96. Active Directory. https://en.wikipedia.org/wiki/Active_Directory. გადამოწმ. 15.06.14.

97. Windows_domain. https://en.wikipedia.org/wiki/Windows_domain. გადამოწმ. 15.06.14.

* * *

რედაქტორი ნ. ქაფიანიძე

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“,
თბილისი, მ. კოსტავას 77