ISSN 1512-1232

უაკ: 004.5

ელექტრონული საარჩევნო სისტემა მულტიმედიური მონაცემთა ბაზებით და კლიენტ-სერვერ არქიტექტურით

გია სურგულაძე, ნინო თოფურია, გიორგი ბასილაძე,
ნინო კივილაძე, მაია ნეფარიძე
საქართველოს ტექნიკური უნივერსიტეტი

რეზიუმე

განიხილება ელექტრონული საარჩევნო სისტემის, როგორც რთული და დიდი სისტემის მოდელირების, ობიექტ–ორიენტირებული დაპროექტების და შემდგომი პროგრამული რეალიზაციის საკითხები. შემოთავაზებულია დაცული სახელმწიფო ქსელის აგების კონცეფცია და მისი არქიტექტურა. დაპროექტებულია ელექტრონული საარჩევნო სისტემისათვის საჭირო და აუცილებელი მულტიმედიალური მონაცემთა რელაციური ბაზები. ამ თვალსაზრისით განიხილება მოდელირების კატეგორიალური მეთოდების და დაპროექტების ობიექტ–ორიენტირებული, CASE-ტექნოლოგიების გამოყენება. ანალიზის შედეგად შერჩეულია ელექტრონული საარჩევნო სისტემისათვის აუთენტიფიკაციის სხვადასხვა ტიპების კომბინირება და მათი ინტეგრირება მულტიმედიალურ მონაცემთა ბაზებში. ესენია, თითის ანაბეჭდის სკანირების კომპონენტი, ხმის აუდიო ჩანაწერი, ბიომეტრული ფოტოსურათი და ელექტრონული ხელმოწერა. ზემოთხსენებული მოდულების ინტეგრაციით იზრდება უსაფრთხოება და ამავდროულად პირდაპირპროპორციულად მატულობს საიმედოობისა და ნდობის ხარისხი.

დამუშავებულია კლიენტ–სერვერ არქიტექტურის ლოგიკურად ერთიანი და ფიზიკურად განაწილებული მონაცემთა რელაციური ბაზების სისტემა ობიექტ–როლური მოდელირების პრინციპების საფუძველზე და შესაბამისი გრაფო–ანალიზური ინსტრუმენტების გამოყენებით.

კორპორაციული განაწილებული მულტიმედიალური ელექტრონული საარჩევნო სისტემის ექსპერიმენტული საპილოტო ვერსიის პროგრამული რეალიზაცია განხორციელდა ახალი ინფორმაციული ტექნოლოგიების ბაზაზე, კერძოდ Visual Studio.Net, SQL Server, ORM/ERM, VPN ტექნოლოგიების და პროგრამული პაკეტებით. გამოყენებულია Windows Presentation Foundation (WPF) და Windows Communication Foundation (WCF) ტექნოლოგიები, აგრეთვე Metro Style App.

შემუშავებულია აგებული სისტემის მომხმარებელთა ინტერფეისები, ინსტრუქციები, დანერგვის და ექსპლუატაციის პროცესების ორგანიზაციული, ტექნიკური და იურიდიული ასპექტები.

საკვანძო სიტყვები: *ელექტრონული საარჩევნო სისტემა, მულტიმედიური მონაცემთა ბაზები, ობიექტ-როლური მოდელირება, კლიენტ-სერვერ არქიტექტურა.*

ISSN 1512-1232

1. შესავალი - ამოცანის დასმა

ნაშრომის მიზანია ელექტრონული საარჩევნო სისტემის კონცეფციის შემუშავება და მისი პროგრამული რეალიზაცია საპილოტო ვერსიის სახით. იგი უზრუნველყოფს ტრადიციული საარჩევნო სისტემის ნაწილობრივ სახეცვლილებას (პირველ ეტაპზე) და მოამზადებს ნიადაგს მისი სრული ჩანაცვლების მიზნით ელექტრონული ანალოგით.

ახალი ელექტრონული საარჩევნო სისტემის ძირითადი დანიშნულებაა [1,2]:

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

ნაშრომში ხორციელდება ელექტრონული საარჩევნო სისტემის ბიზნეს-პროცესების მართვის დაპროექტება და რეალიზაცია სერვის-ორიენტირებული არქიტექტურით.

მიზნის მისაღწევად ნაშრომში განიხილება შემდეგი ძირითადი ამოცანები:

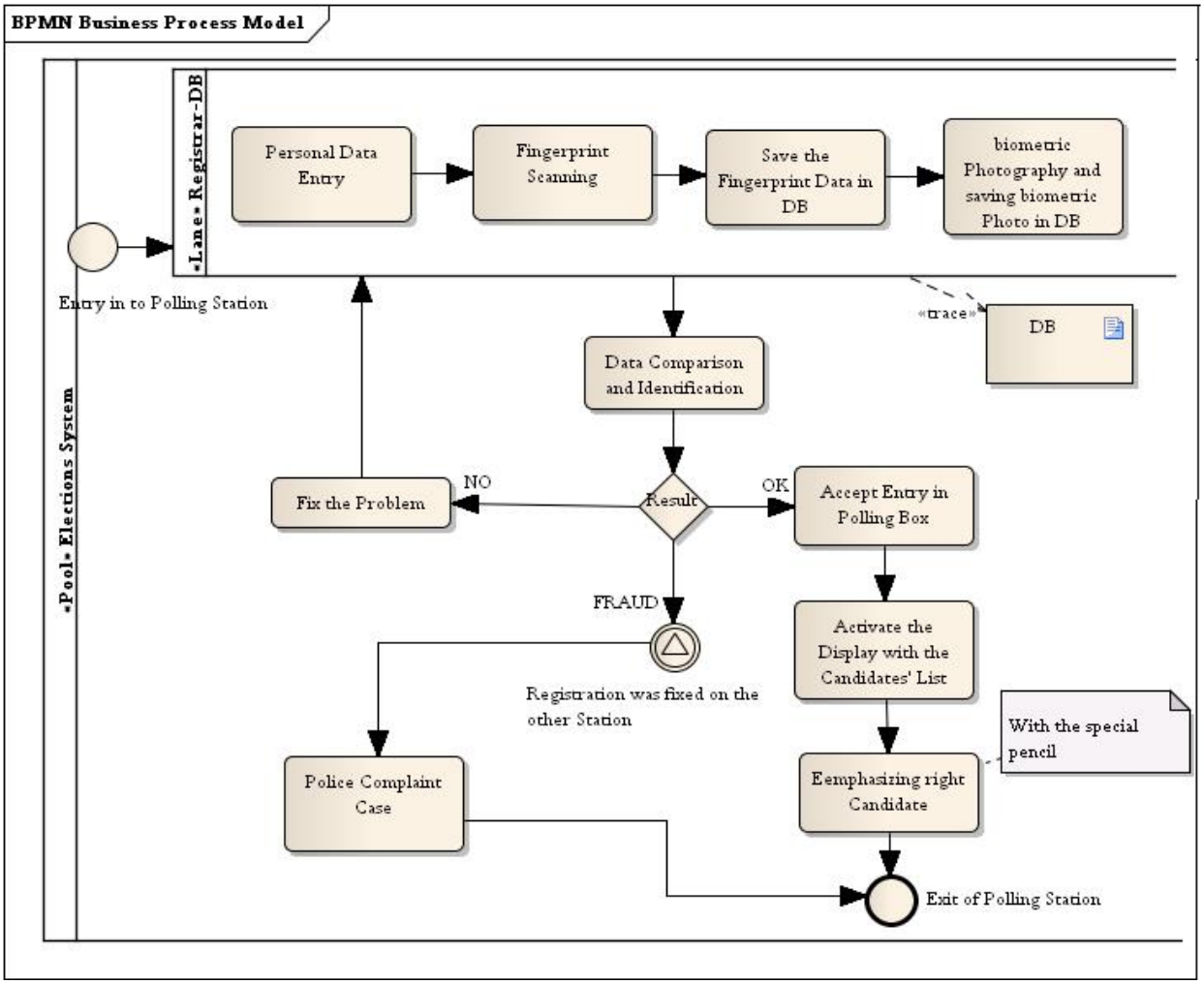
- არსებული თანამედროვე ელექტრონული საარჩევნო სისტემების ანალიზი და შესაბამისი ინფორმაციული ტექნოლოგიების კლასიფიკაცია, ობიექტ-, პროცეს- და სერვის-ორიენტირებული დაპროექტების პრინციპებით;
- საარჩევნო სისტემის ბიზნეს-პროცესების ტრადიციული და სერვის–ორიენტირებული მოდელების აგება სისტემური ანალიზის საფუძველზე, BPMN და UML ტექნოლოგიების ბაზაზე;
- მულტიმედიალური მონაცემთა ბაზების დაპროექტება და აგება კლიენტ-სერვერული ტექნოლოგიის გამოყენებით SQL Server-ის ბაზაზე;
- სერვის-ორიენტირებული მონაცემთა განაწილებული ბაზების სტრუქტურების დასაპროექტებლად ობიექტ-როლური მოდელების (ORM) აგება და კვლევა რევერსიული CASE ტექნოლოგიების გამოყენებით;
- პროექტის შედეგების საფუძველზე ესპერიმენტული პროგრამული სისტემის რეალიზაცია .NET პლატფორმაზე, C#.NET, Natural ORM Architect და SQL Server პროგრამული პაკეტების გამოყენებით.

2. ელექტრონული საარჩევნო სისტემების ბიზნეს-პროცესების აღწერა BPMN ენაზე

ელექტრონული საარჩევნო სისტემის ბიზნეს-პროცესები შედგება რიგი სავალდებულო პროცედურებისგან, რომელთა განხორციელება ევალება ყოველ ამომრჩეველს წინასწარი რეგისტრაციის სახით [3-5]. 1-ელი ნახაზი ასახავს ელექტრონულ საარჩევნო სისტემას, რომელიც საბოლოო პროდუქტია და მოიცავს შემდეგ პროცედურებს: დარეგისტრირება ნებისმიერ რეგისტრატორთან შემდეგი წესის დაცვით: ტექსტური ინფორმაციის მონაცემთა ბაზაში შეყვანის შემდეგ უნდა მოხდეს თითის სკანირება და ანაბეჭდის ბაზაში გადატანა; ბიომეტრული ფოტო-სურათის გადაღება და მონაცემთა ბაზის შესაბამის ველში მისი დამახსოვრება; ხმის ჩაწერა და 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

ISSN 1512-1232

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 უბანზე ნამყოფია და დარეგისტრირებულია, იგი მხილებული იქნება და აღნიშნული კეისის გარკვევაში ჩაერთვება პოლიცია და სამართალდამცავი ორგანოები.

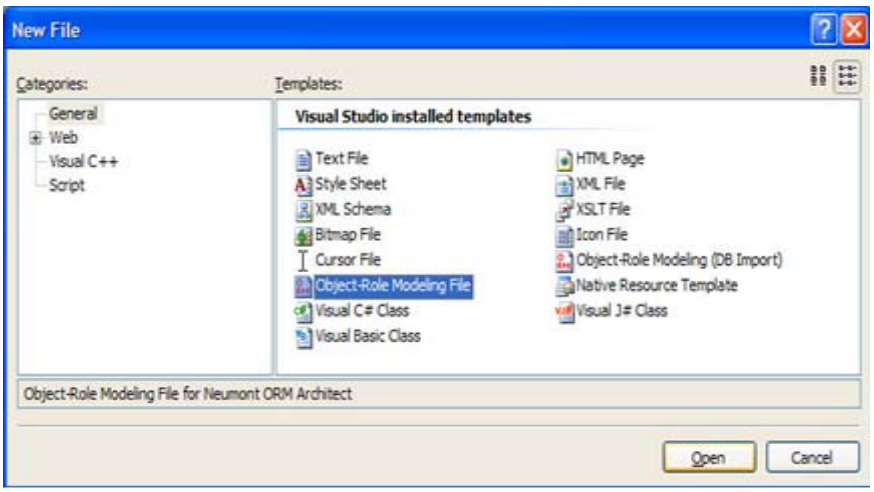


ნახ.1. ელექტრონული საარჩევნო სისტემის ბიზნეს-პროცესების სქემა

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

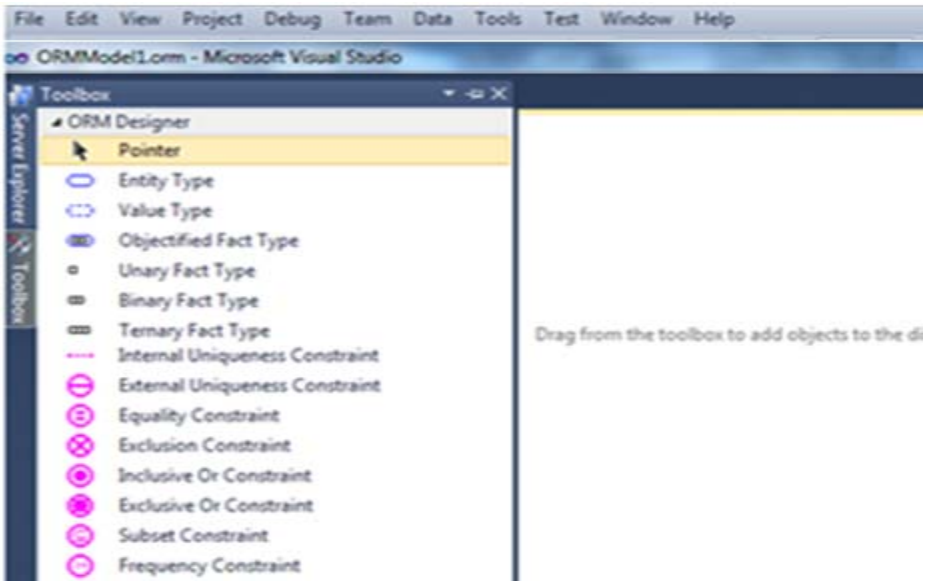
მონაცემთა ბაზების ავტოამტიზებულ რეჟიმში დაპროექტება შესაძლებელია ობიექტ-როლური ORM (Object-Role Modeling) მოდელირების ტექნოლოგიის საშუალებით. Natural ORM Architect (NORMA) ინსტრუმენტი წარმოადგენს Microsoft Visual Studio.NET Framework-ის plugin-ს [3,5,8]. ORM-დიაგრამის შესაქმნელად საჭიროა General კატეგორიიდან Object-Role Modeling File template-ის ამორჩევა (ნახ.2).

ISSN 1512-1232



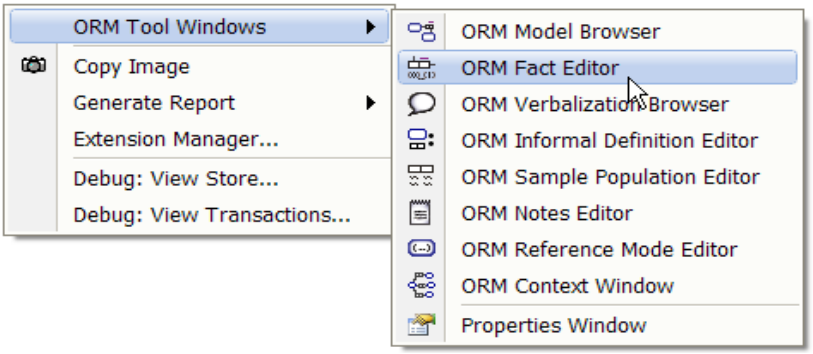
ნახ.2. ORM-ფაილის გახსნა

ORM-დიაგრამის აგება ხდება Document Window ფანჯარაში, სადაც სახეზეა დიაგრამის ასაგებად საჭირო ყველა ინსტრუმენტი: არსი (Entity), კავშირი (Connector) და შეზღუდვები (Constraints), იხ.ნახ.3.



ნახ.3. ORM-დოკუმენტის ფანჯარა

ფაქტების შესატანად საჭიროა FACT EDITOR-ის გააქტიურება (ნახ.4).



ნახ.4. ORM Fact Editor-ის ამორჩევა

ISSN 1512-1232

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

ფაქტი არის კატეგორიალური 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. 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.

ზემომოყვანილი ფაქტების შეტანის შემდეგ ORM-დიაგრამა მიიღებს მე-5 ნახაზზე ნაჩვენებ სახეს.

აგებული დიაგრამიდან ავტომატიზებულ რეჟიმში ვახდენთ არსთა-დამოკიდებულების მოდელის ER (Entities-Relationship Model) კონსტრუირებას Extension manager-ის საშუალებით და მიიღება ER-მოდელის დიაგრამა, რომელიც ნაჩვენებია მე-6 ნახაზზე.

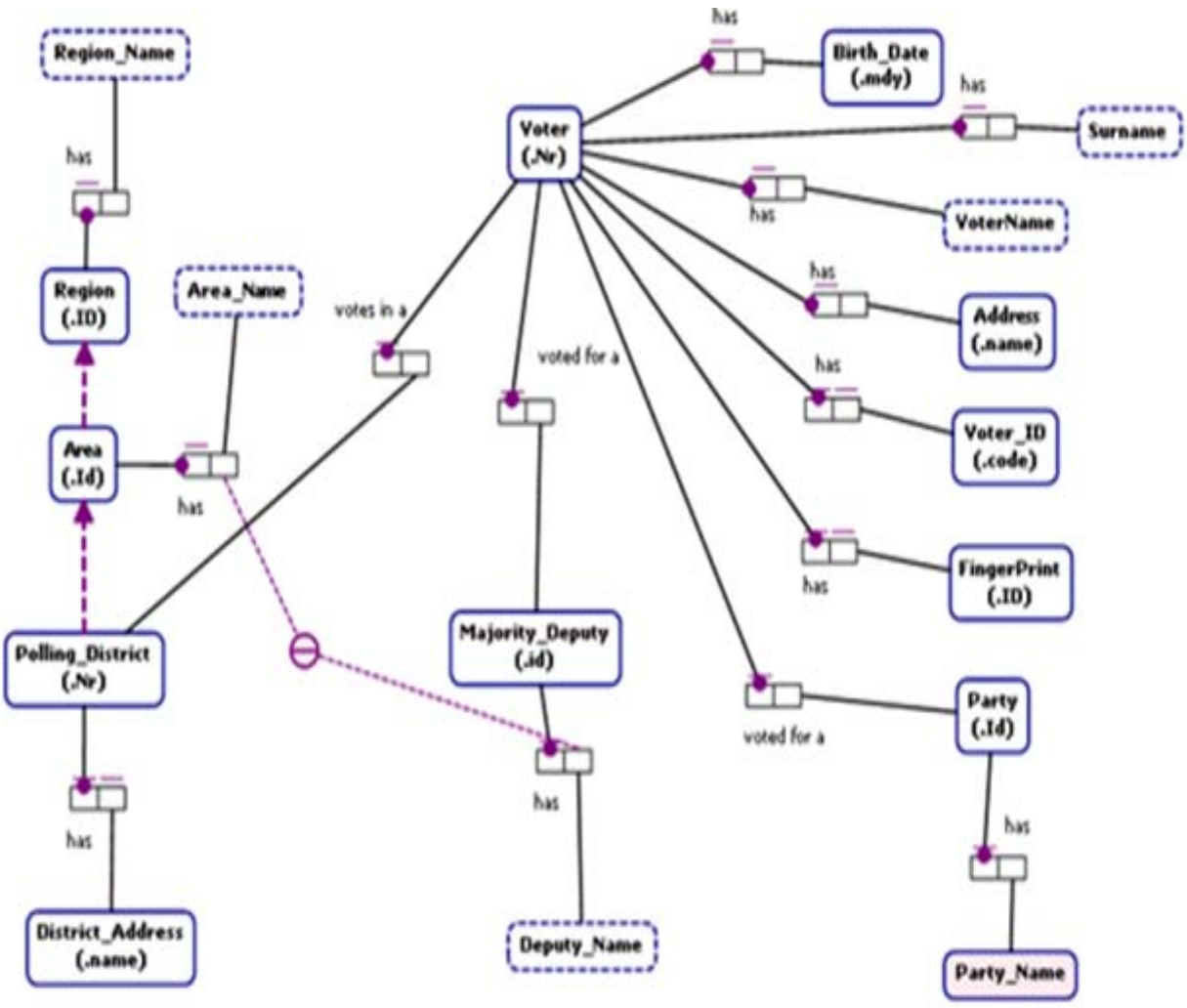
დიაგრამაზე გამოსახულია სამი არსი (Entities) და ორი ლოგიკური რელაციური კავშირი, რომელთა პროგრამული რეალიზაცია ხორციელდება ცხრილების (Tables) და ცხრილთაშორისი კავშირებით პირველადი (Primary key) და მეორეული (Foreign key) გასაღებების გამოყენებით.

ჩვენ მაგალითში ამ მიზნით გამოყენებულია: PK: majority_DeputyId, PK: regionId, PK: voterNr, FK1: majority_DeputyId, FK2:pooling_DistrictNr, U1:fingerPrintID, U2:voter_IDCode.

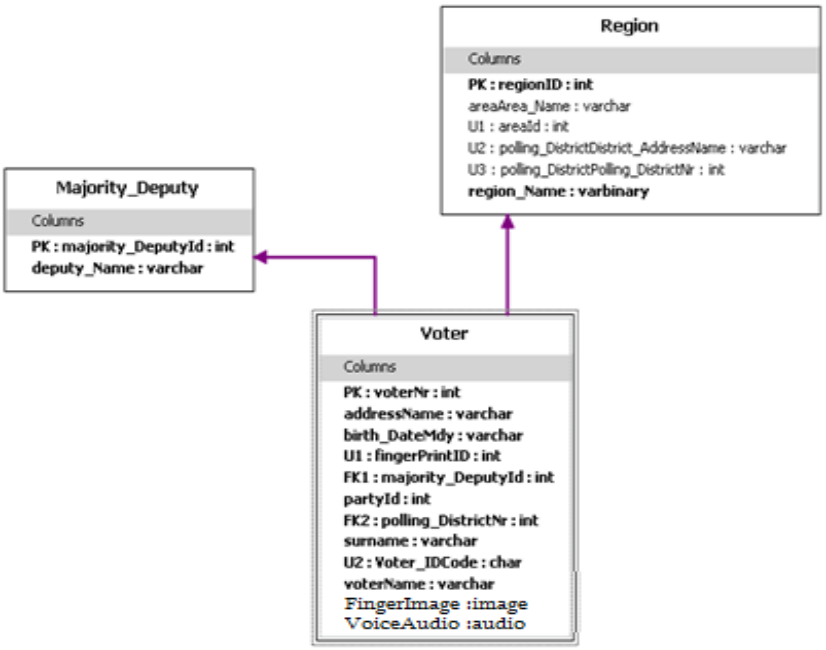
Natural ORM Architect პაკეტის საშუალებით, ასევე ავტომატიზებულ რეჟიმში, ვახდენთ მონაცემთა აღწერის ენის შესაბამის DDL-კოდის გენერაციას.

Visual Studio.NET პაკეტის Solution Explorer-ის საშუალებით შესაძლებელია SQL Server მონაცემთა ბაზისთვის ჩვენ მიერ გენერირებული კოდის ნახვა.

ISSN 1512-1232



ნახ.5. ORM-დიაგრამის ფრაგმენტი



ნახ.6. ავტომატიზებულ რეჟიმში მიღებული
ER-მოდელის ფრაგმენტი 3 ცხრილით

ISSN 1512-1232

მონაცემთა ბაზის ეს DDL-ფაილის ფრაგმენტი შემდეგნაირად გამოიყურება:

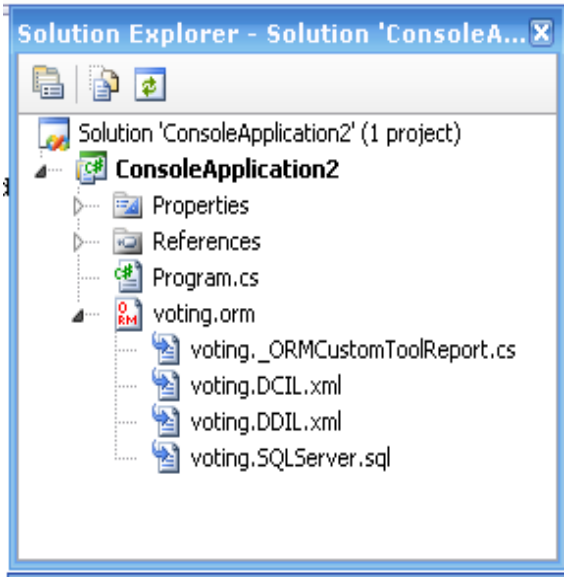
```
CREATE SCHEMA ORMModel1
GO
CREATE TABLE ORMModel1.Voter
( voterNr INTEGER NOT NULL,
  fingerPrintID INTEGER IDENTITY (1, 1) NOT NULL,
  Voter_IDCode NATIONAL CHARACTER(4000) NOT NULL,
  voterName NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  surname NATIONAL CHARACTER VARYING(MAX) NOT NULL,
  addressName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

ISSN 1512-1232

```
    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
```

4. მონაცემთა ბაზის აგების პროგრამული რეალიზაციის ავტომატიზებული პროცედურა

Natural ORM Architect პროგრამული პაკეტის დახმარებით ავტომატიზებულ რეჟიმში მოვახდინეთ DDL კოდის გენერირება, რათა აგვეგო დასაპროექტებელი სისტემის რელაციური სქემა. Solution Explorer-ში ნაჩვენებია კოდი, რომელიც დაგენერირდა SQL Server-ისათვის (ნახ.7). ნახაზზე ილუსტრირებულია ConsoleApplication2 პროექტის სტრუქტურა, რომლის voting.orm ფაილი რამდენიმე კომპონენტისგან (.cs, .xml და .sql ფაილები) შედგება.



ნახ.7. VisualStudio.NET გარემოში პროექტი ConsoleApplication2

ISSN 1512-1232

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

ცხრილი 1. ელექტრონული საარჩევნო სისტემის დამახასიათებელი ფაქტები

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.
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 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**
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**

ISSN 1512-1232

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**.
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.

Birth_Date.
Region **is an entity type**.
Reference Scheme: Region has Region_ID.
Reference Mode: .ID.

Fact Types:
Region has Region_ID.
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: **C** 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.

5. მულტიმედიური რელაციური ბაზის ოპტიმალური სტრუქტურის დაპროექტება

ელექტრონული საარჩევნო სისტემის მონაცემთა რელაციური ბაზის კონცეპტუალური სქემის დასაპროექტებლად გამოვიყენეთ ობიექტ-როლური მოდელირების მეთოდი და CASE-ინსტრუმენტული საშუალება (NORMA). ავტომატიზებული პროცესი მომხმარებლის მიერ სემანტიკური ფაქტების აღწერით იწყებოდა, რომლის საფუძველზე ფორმირდებოდა კონცეპტუალური სქემა (1-ელი დონე) ORM დიაგრამის (ნახ.5) სახით, ობიექტებით და 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. ანუ მიიღება ექვივალენტური კონცეპტუალური საქემები. რომელია მათ შორის ოპტიმალური, ან უკეთესი სისტემის მონაცემთა ბაზის საბოლოო

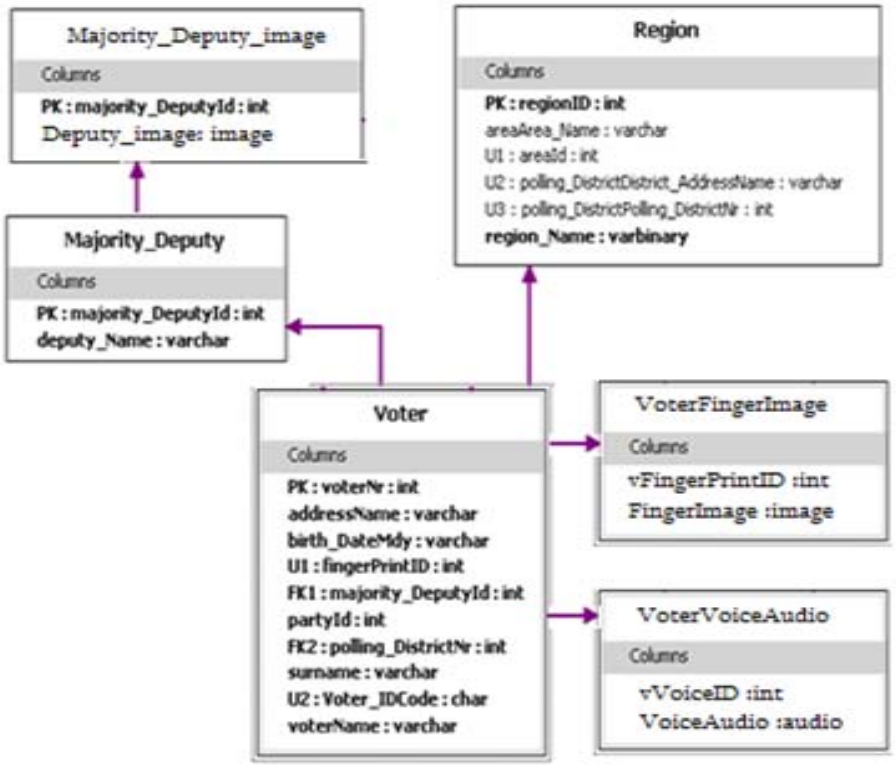
ISSN 1512-1232

რეალიზაციის მიზნით ? 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.

მე-8 ნახაზზე მოცემულია მე-6 ნახაზის დიაგრამის ექვივალენტური კონცეპტუალური (მე-2 დონე) ERM სქემა. მათ შორის განსხვავება ცხრილების (Tables) რაოდენობაშია, რომლებიც ავტომატიზებული პროცედურის 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 პროცესი ცხრილების რაოდენობის მინიმიზაცია, როცა სემანტიკური მთლიანობა არ ირღვევა (!), ანუ ეს გარკვეული ინფორმაციული სიჭარბის აღმოფხვრაა [9-12].



ნახ.8. მე-6 დიაგრამის ექვივალენტური ERM სქემა 6 ცხრილით

ER-მოდელის ცხრილების ოპტიმალური შემადგენლობის განსაზღვრის ამოცანა კავშირშია რელაციური ბაზის სტრუქტურის ნორმალიზაციის საკითხთან [13-15]. 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].

ISSN 1512-1232

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 [15].

დამოკიდებულებათა დეკომპოზიციის დროს ბნფ-ები მიიღება სქემის დაპროექტების ცალკეულ ეტაპზე. არადეკომპონირებადი დამოკიდებულებისთვის კი საჭიროა ხელოვნურად ფიქტიური ატრიბუტის (ნატურალურ რიცხვთა სასრული სიმრავლე) შემოტანა [13]. 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 ფდ-ების სიმრავლე:

$$\begin{cases} R_1(k_1, k_2, \ldots k_{n1}, A_1, A_2, \ldots, A_{a1}) \\ R_2(k_1, k_2, \ldots k_{n2}, B_1, B_2, \ldots, B_{a2}) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ R_l(k_1, k_2, \ldots k_{nl}, Z_1, Z_2, \ldots, Z_{al}) \end{cases} \quad (1)$$

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:

ISSN 1512-1232

$$R(k_1 \dots k_{n1}, A_1 \dots A_{a1}, B_1 \dots B_{a2}, \dots Z_1 \dots Z_{al}) \quad (2)$$

დავუშვათ აგრეთვე, რომ წინასწარ ცნობილია R_i-ის ცვლილების რაოდენობა $\square_i$ დროის განსაზღვრულ ინტერვალში და მართებულია შემდეგი მოწესრიგება.

$$\mu_1 \geq \mu_2 \geq \dots \geq \mu_l$$

(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 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:

$$\mu_1 * (n_1 + \sum_{k=1}^{l} a_k) \geq \dots \geq \sum_{j=1}^{s} \mu_j * (n_j + \sum_{k=1}^{l} a_k) \geq \dots \geq \sum_{i=1}^{s} \mu_i * (n_i + a_i) \quad (3)$$

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:

$$\sum_{j=1}^{s} \mu_j n_j + \sum_{j=1}^{s} \sum_{k=1}^{l} \mu_j a_k \geq \sum_{i=1}^{l} \mu_i n_i + \sum_{i=1}^{l} \mu_i a_i \quad (4)$$

აქედან მართებულია შემდეგი გამოსახულებანი:

$$\sum_{i=1}^{l} \mu_i n_i - \sum_{j=1}^{s} \mu_j n_j = \sum_{i=s+1}^{l} \mu_i n_i \quad (5)$$

$$\sum_{j=1}^{s} \sum_{k=1}^{l} \mu_j a_k - \sum_{i=1}^{l} \mu_i a_i = \sum_{j=1}^{s} \sum_{k=1, j \neq k}^{l} \mu_j a_k - \sum_{i=s+1}^{l} \mu_i a_i \quad (6)$$

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

$$\sum_{j=1}^{s} \sum_{k=1, j \neq k}^{l} \mu_j a_k \geq \sum_{i=s+1}^{l} \mu_i n_i + \sum_{i=s+1}^{l} \mu_i a_i \quad (7)$$

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

$$\frac{\sum_{j=1, k=1}^{s} \sum_{j \neq k}^{l} \mu_j a_k}{\sum_{i=s+1}^{l} \mu_i a_i} \geq \frac{\sum_{i=s+1}^{l} \mu_i n_i}{\sum_{i=s+1}^{l} \mu_i a_i} + \frac{\sum_{i=s+1}^{l} \mu_i a_i}{\sum_{i=s+1}^{l} \mu_i a_i} \quad (8)$$

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

ISSN 1512-1232

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

ამგვარად, (8)-დან მივიღებთ ვედეკინდ-სურგულაძის მოდელს [13]:

$$\frac{\sum_{j=1}^{s}\sum_{k=1,j\neq k}^{l}\mu_j a_k}{\sum_{i=s+1}^{l}\mu_i a_i}+\frac{\sum_{j=1}^{s}\mu_j}{\sum_{i=s+1}^{l}\mu_i}\geq\sum_{i=s+1}^{l}\frac{n_i}{a_i}+1 \qquad (9)$$

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:

$$\frac{\mu_1}{\mu_2}\geq\frac{n_2}{a_2}+1 \qquad (10)$$

რაც, როგორც ცნობილია, არის ვონგ-ვედეკინდის მოდელი [16].

იგი არის (9) გამოსახულების კერძო შემთხვევა. (10)-ის გამოყენების დიაპაზონია მე-3ნფ-მდე, ხოლო (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:

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

რომლებშიც მართებულია შემდეგი პირობები:

$$k_1,\dots k_n\supseteq k_1,\dots k_{n2}\ \text{ da }\ \mu_1>\mu_2 \qquad (11)$$

(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 (11)-ს.

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

ISSN 1512-1232

- 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;

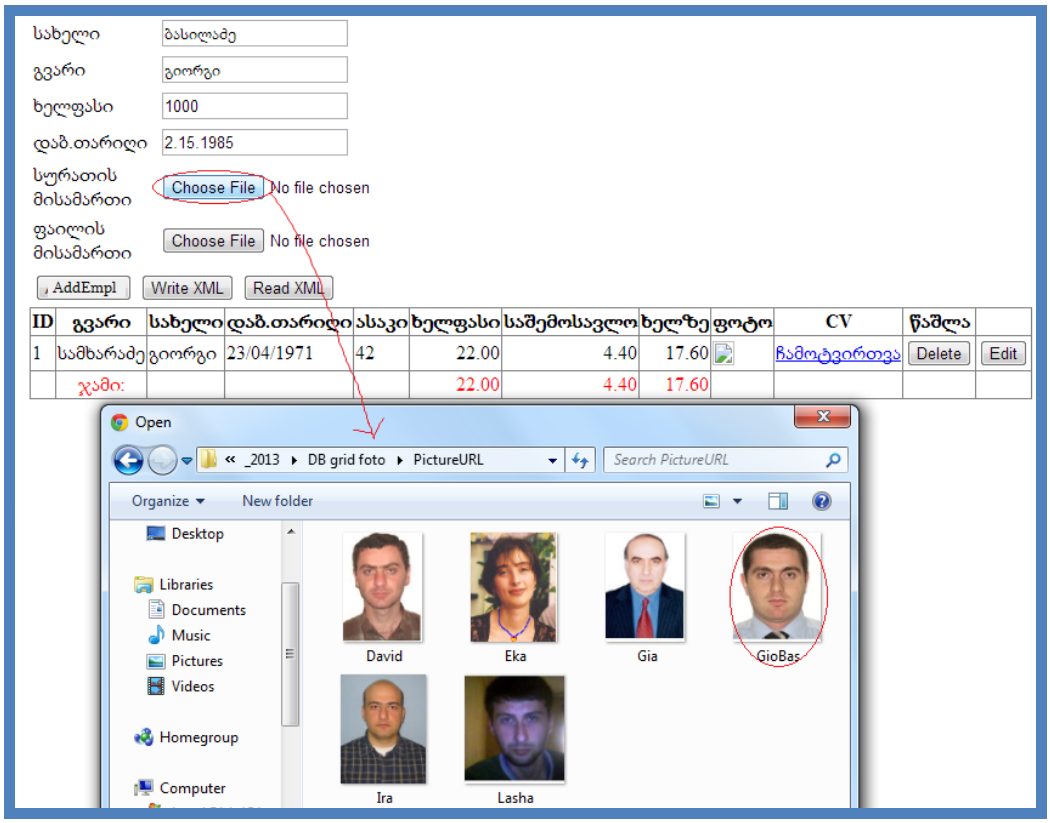
- მოცემული □--თვის შეიძლება (9) გამოსახულებით დადგინდეს მოსახერხებელი (ოპტიმალური) ნფ-ები;

- 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.

6. სისტემის პროგრამული რეალიზაცია და ექსპლუატაციის მხარდაჭერა

6.1. ელექტრონული საარჩევნო სისტემის კომპონენტების დამუშავება Visual Studio.NET Framework გარემოში

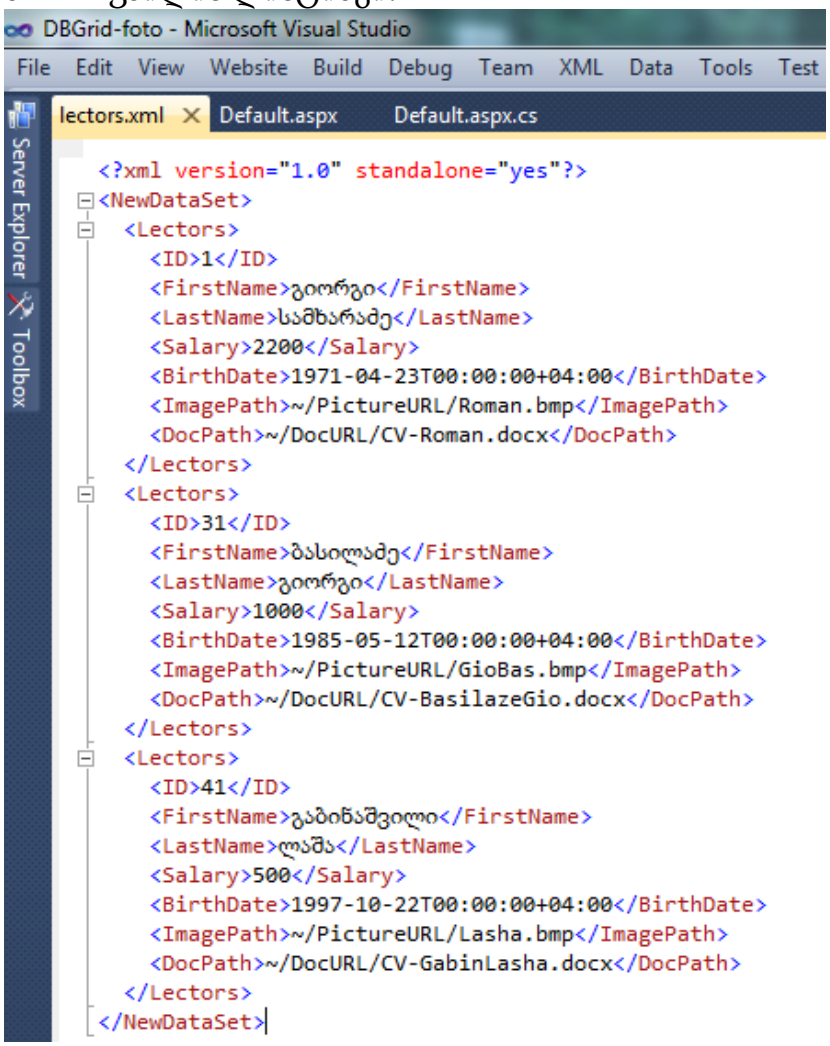
მე-9 ნახაზზე ნაჩვენებია ინტერნეტ ბრაუზერში მომუშავე სისტემის ინტერფეისი - ამომრჩეველთა რეგისტრაცია. გრაფიკული და ტექსტური ინფორმაციის შესატანად გამოყენებულია შესაბამისი დიალოგური პროცედურები.



ნახ.9. ინტერფეისის ფრაგმენტი ფოტო ინფორმაციის მიერთებით

ISSN 1512-1232

Write XML ღილაკით განახლებული ცხრილი ჩაიწერება XML-ფაილში (მონაცეთა ბაზაში). Read XML ღილაკი უზრუნველყოფს ამ ბაზის ხელახლად წაკითხვას. მე-10 ნახაზზე ნაჩვენებია XML ფაილის ლისტინგი:



ნახ.10. XML ფაილის შიგთავსის სტრუქტურა

6.2. 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 შედარებით 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 [7,17].

ISSN 1512-1232

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 დაფიქსირებაში).

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 მისვლა აღმოჩენილი იქნება პროგრამის მიერ და ეცნობება საუბნო საარჩევნო კომისიის რეგისტრანტ წევრს იმის შესაებ თუ რა დროს და რომელ უბანზე გაიარა უკვე რეგისტრაცია ამა თუ იმ ამომრჩეველმა.

კომისიის წევრი ვალდებულია აღკვეთოს ხელმეორედ ხმის მიცემის ფაქტი და აცნობოს სამართალდამცავ ორგანოებს ზემოაღნიშნული, შესაბამისი რეაგირებისათვის.

პროგრამული უზრუნველყოფისათვის გამოვიყენეთ უახლესი ტექნიკა და თანამედროვე ტექნოლოგიები: თითის ანაბეჭდის სკანერი, ელექტრონული ხელმოწერის პანელი, ხმის ჩამწერი და შემდეგ მისი ამომცნობი სისტემები და ფოტოაპარატი, აგრეთვე ბიომეტრული სურათების შედარების სისტემები.

ამ ყველაფრის ხარჯზე, 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, როგორიცაა ამომრჩევლის თითის ანაბეჭდი, ამომრჩევლის ხმა, ამომრჩევლის ხელმოწერა და ამომრჩევლის 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 რეგისტრატორი, ამის შემდეგ მოხდება მოთხოვნა Access Code(AC)-ის, რომელიც იქნება უნიკალური და წინასწარ დალუქული კონვერტით ექნება მიღებული საუბნო კომისიის თავჯდომარეს.

უნდა შედგეს აღნიშნული კონვერტის გახსნის ოქმი, რომელზეც კომისიის თავმჯდომარესთან ერთად ხელმომწერები იქნებიან თავჯდომარის მოადგილე და საუბნო კომისიის წევრები. აღნიშნული AC კოდის შეტანის შემდეგ მიეცემათ უფლება დაიწყონ ამომრჩეველთა მიღება და რეგისტრაცია. ყოველივე ეს აძლიერებს და ამყარებს უსაფრთოების ხარისხს და მეტ დამაჯერებლობას სძენს ელეტრონულ საარჩევნო სისტემას.

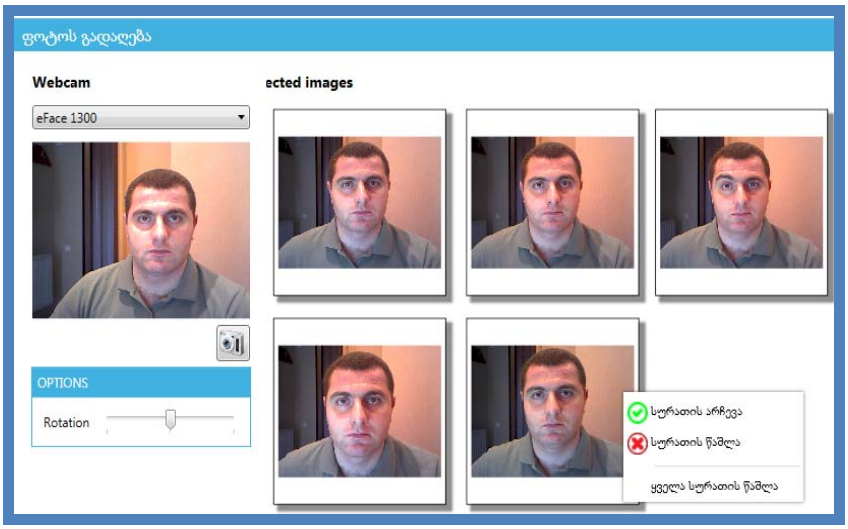
ამომრჩევლის უბანზე მისვლისას ხდება მისთვის ბიომეტრული სურათის გადაღება ვებ-კამერის მეშვებით, რეგისტრატორს შეუძლია ერთი ან რამდენიმე სურათის გადაღება და არჩევანის გაკეთება.

ვებ-კამერა იმართება პროგრამიდან და აკეთებს დროებით ჩანაწერებს კომპიუტერში, სადაც ინახება მოქალაქის სურათი თავისივე უნიკალური დასახელებით,

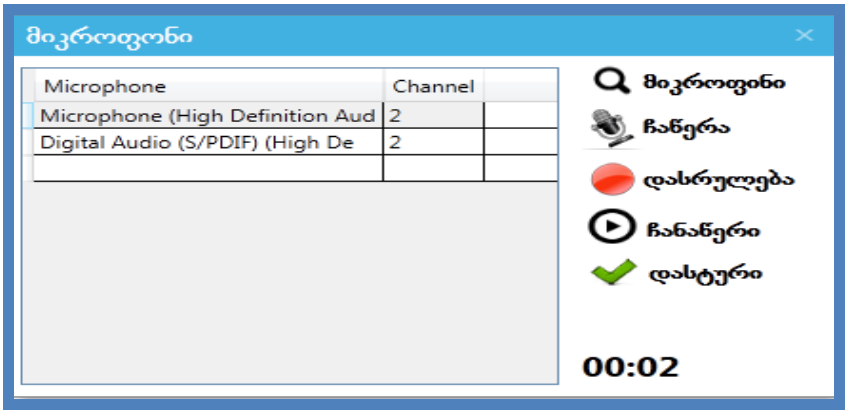
ISSN 1512-1232

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.11).

ხმის ჩასაწერად და მიკროფონზე წვდომის განსახორციელებლად გამოვიყენეთ ყველასთვის კარგად ცნობილი და გამოცდილი ბიბლიოთეკა, როგორიცაა Naudio. მისი მეშვეობით ხდება წვდომა მიკროფონზე და ძიება ყველა არსებული მიკროფონის, რომელიც მიერთებულია კომპიუტერზე (ნახ.12).



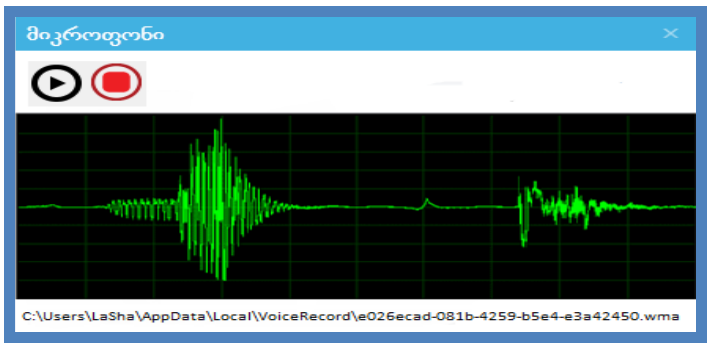
ნახ.11. მოქალაქის სურათი თავისივე უნიკალური დასახელებით



ნახ.12. კომპიუტერთან მიერთებული მიკროფონების ნუსხა

რეგისტრატორმა უნდა აირჩიოს მიკროფონი და ჩაიწეროს ხმა, რომელიც ასევე დროებით ჩაწერას აკეთებს კომპიუტერის მეხსიერებაში. ჩაწერის შემდეგ შესაძლებელია ჩაწერილი ხმის მოსმენა და მისი გრაფიკულად ნახვა. (ნახ.13).

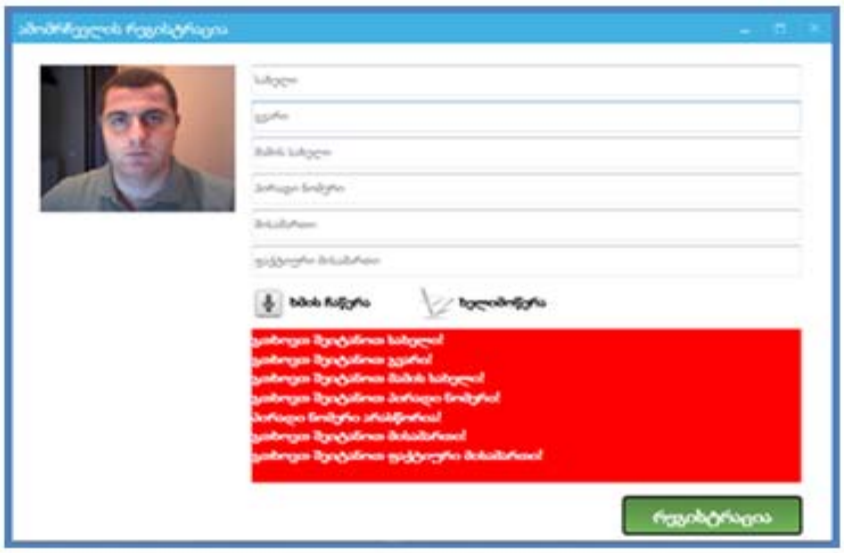
ISSN 1512-1232



ნახ.13. ხმის გრაფიკული გამოსახულება

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 ფაქტიური მისამართის შეტანა და რეგისტრაცია მოხდება წინასწარ განსაზღვრული ფონტით-Sylfaen. რაც შეეხება პირად ნომერს, იგი აუცილებლად იქნება 11 ნიშნა რიცხვითი მონაცემი.

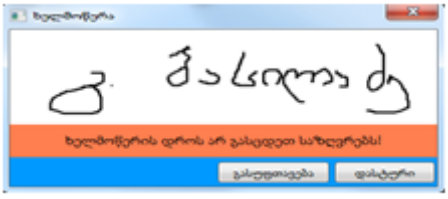
აღნიშნული შეცდომების დაშვების დროს მივიღებთ შემდეგი სახის გამაფრთხილებელ დიალოგურ ფანჯარას (ნახ.14).



ნახ.14. გამაფრთხილებელი ფანჯარა

პროგრამა გამორიცხავს ამომრჩევლის შესახებ არასრული ინფორმაციის შემთხვევაში ბაზაში ჩანაწერის გაკეთებას, რაც თავის მხრივ ხელს უშლის არჩევნებზე დარეგისტრირებული ამომრჩეველთა რიცხვის ხელოვნურ ზრდას.

პროგრამული უზრუნველყოფა ითვალისწინებს აგრეთვე ელექტრონული ხელმოწერის შეტანა-შენახვის მოდულს (ნახ.15) და თითის ანაბეჭდის შეტანა-შენახვა-იდენტიფიცირებას.

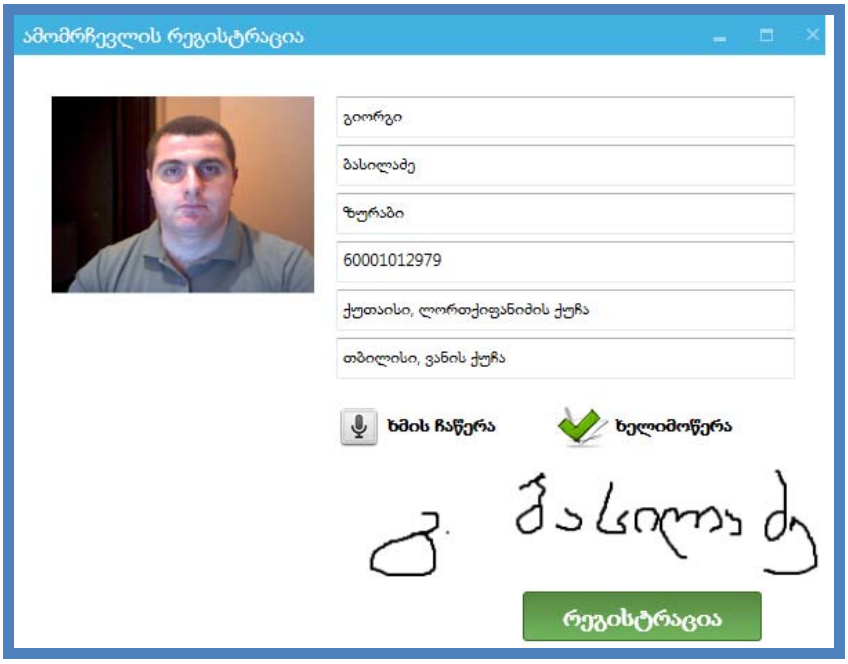


ISSN 1512-1232

ნახ.15. ელექტრონული ხელმოწერის შეტანა

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.

პროექტის ფარგლებში შემუშავებულია მომხმარებელთა ინსტრუქციებიც [1]. მაგალითად, მე-16 ნახაზი გვიჩვენებს რეგისტრაციის ფორმის შევსებულ, საბოლოო ვარიანტს.



ნახ.16. რეგისტრაციის ფორმა

6.3. კომუნიკაცია: WCF - ტექნოლოგია

ბიზნეს–პროცესების შესრულებისას ერთ–ერთი მნიშვნელოვანი ასპექტია ურთიერთობა (კომუნიკაცია) აპლიკაციებს შორის, კლიენტებსა და სერვერებს შორის, აგრეთვე მუშა–პროცესებსა და ჰოსტ–დანართებს შორის.

აპლიკაციის მაგალითის სახით განიხილება პროექტის აგება საარჩევნო სისტემისათვის, რომელშიც მოთხოვნილი ინფორმაცია (მაგალითად, ამომრჩევლის ან დეპუტატის შესახებ) გადაიცემა ფილიალებს (საარჩევნო სისტემის იერარქიული რგოლები) შორის.

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 მონიტორინგისთვის.

ავაგოთ მარტივი WPF-აპლიკაცია (Windows Presentation Foundation), რომელიც გამოიყენებს კომუნიკაციას სხვადასხვა აპლიკაციათა ბიზნეს–პროცესებს შორის.

ISSN 1512-1232

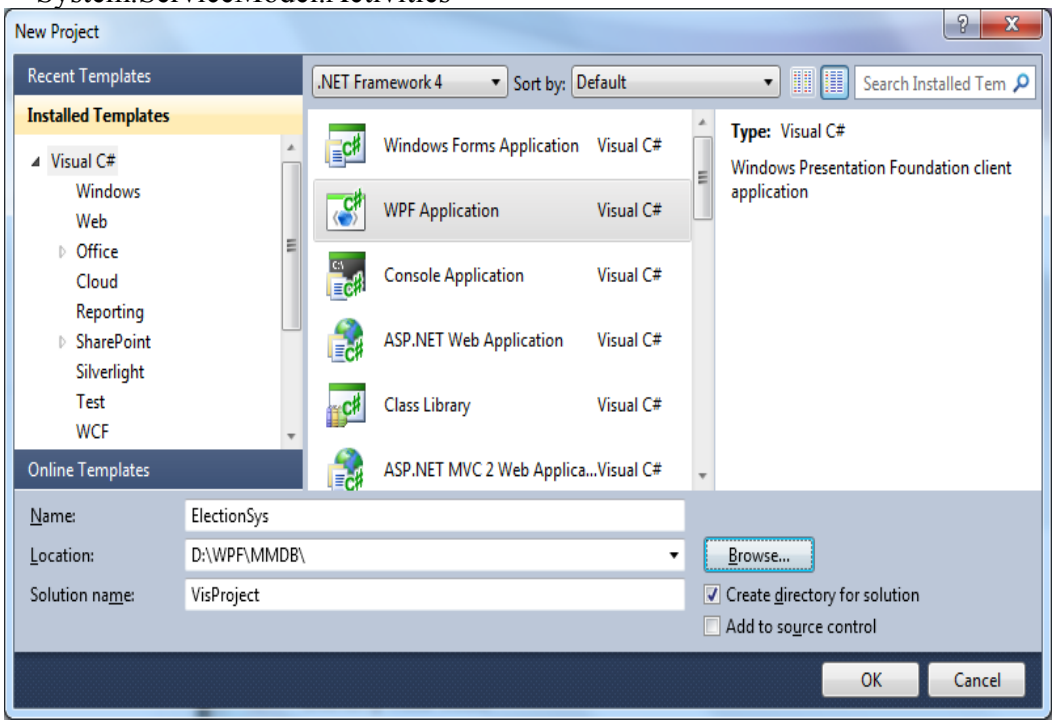
მაგალითად, ეს შეიძლება იყოს ცენტრალური საარჩევნო კომისიის ოფისი (ცენტრი), რეგიონალური, ოლქის (მხარე) ან უბნის საარჩევნო ოფისები (ფილიალები).

6.3.1. WPF პროექტის შექმნა

1. ახალი პროექტის შექმნა WPF აპლიკაციის სახით იწყება პროექტის სახელის ElectionSys და Solution-ის დასახელების VisProject შერჩევით (ნახ.17).

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



ნახ.17. WPF აპლიკაციის შექმნა

3. Solution Explorer-ში გენერირდება ვინდოუსის ფაილი სახელით Window1.xaml, რომელსაც ვცვალოთ Election.xaml - ით.

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

StartupUri=" Election.xaml"

6.3.2. ახალი config-ფაილის და Class-ების შექმნა

რამდენიმე აპლიკაციის კონფიგურირების მიზნით (საარჩევნო ფილიალების რაოდენობის შესაბამისად) შეიქმნება App.config ფაილების დუბლები, 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

ISSN 1512-1232

მონაცემებს
1-ელი ლისტინგის მსგავსად.

```
<!— ლისტინგი_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.

მე-2 ლისტინგში მოცემულია შესაბამისი კოდი:

```
//--- ლისტინგი 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)
      {
```

ISSN 1512-1232

```
            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; }
```

ISSN 1512-1232

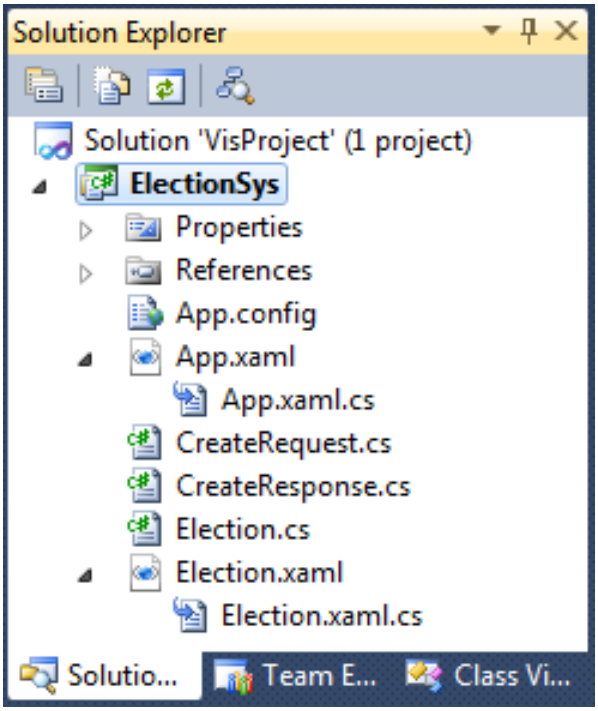
```
        }
      [MessageBodyMember]
      public Guid RequestID
        {
         get { return _RequestID; }
         set { _RequestID = value; }
        }
      [MessageBodyMember]
      public Branch Requester
        {
         get { return _Requester; }
         set { _Requester = value; }
        }
      [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
  }
  /*****************************************************/
```

ISSN 1512-1232

```
    // სერვისის კონტრექტის 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.18).



ნახ.18. Solution Explorer-ის ფანჯარა

6.3.3. Window Form –ის განსაზღვრა

გავხსნათ Election.xaml ფაილი და ავირჩიოთ XAML tab. ჩავანაცვლოთ კოდი შემდეგი
მე-3 ლისტინგის ტექსტით:

```
    <!– – ლისტინგი_.3 – – – Election.xaml ---- –>
<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"
```

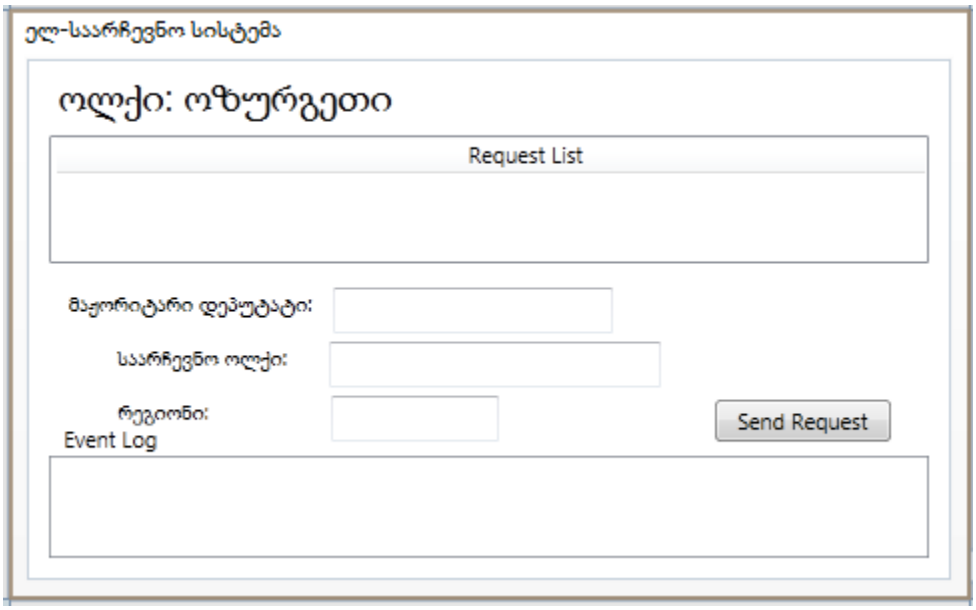
ISSN 1512-1232

```
   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"
```

ISSN 1512-1232

```
            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-ს. ფორმას უნდა ჰქონდეს შემდეგი სახე (ნახ.19). ფორმის ზედა ნაწილში მოთხოვნების აია (RequestList) ასახავს შემოსულ მოთხოვნებს, რომლებიც მოქმედებაშია. მოთხოვნის გასაგზავნად რეგიონალურ ცენტრში გამოიყენება ველები ფორმის შუაში.



ნახ.19. ინტერფეისი „საარჩევნო ოლქი“

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.

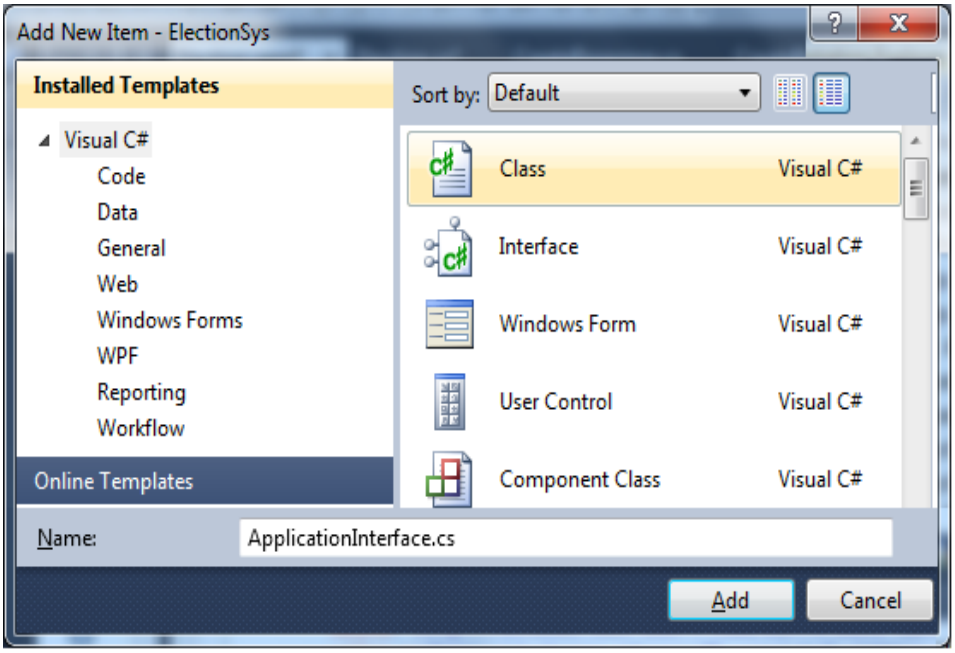
6.3.4. ტექსტის ჩამწერის რეალიზაცია

ISSN 1512-1232

WriteLine ქმედებისთვის, რომელიც გამოყენება ტექსტის გამოსატანად ეკრანზე, არ დაგვიყენებია თვისება TextWriter. კონსოლის რეჟიმში ტექსტი ავტომატურად გამოიტანება, ფორმაზე გამოსატანად კი უნდა გავეცნოთ TextWriter კლასს.

6.3.5. 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.20). კლასის სახელია ApplicationInterface.cs, რომლის პროგრამული რეალიზაცია მოცემულია მე-4 ლისტინგში.



ნახ.20. Class-ის არჩევა

// – – ლისტინგი_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)
    {
```

ISSN 1512-1232

```
            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 აინიციალიზებს _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 ფაქტიური კონტროლისთვის, რომელმაც უნდა ასახოს ეს მოვლენები. ამ მეთოდს იყენებს ApplicationInterface კლასი. AddEvent () მეთოდს იყენებს აპლიკაცია მაშინ, როცა მას სჭირდება მოვლენის დამატება.

6.3.6. ListBoxTextWriter–ის რეალიზაია

დავამატოთ პროექტს კლასი ListBoxTextWriter.cs, რომლის რეალიზაცია მე-5 ლისტინგშია მოცემული.

ISSN 1512-1232

```
// – – ლისტინგი_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);
```

ISSN 1512-1232

```
                    ;
      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.

6.3.7. მუშა პროცესების რეალიზაცია

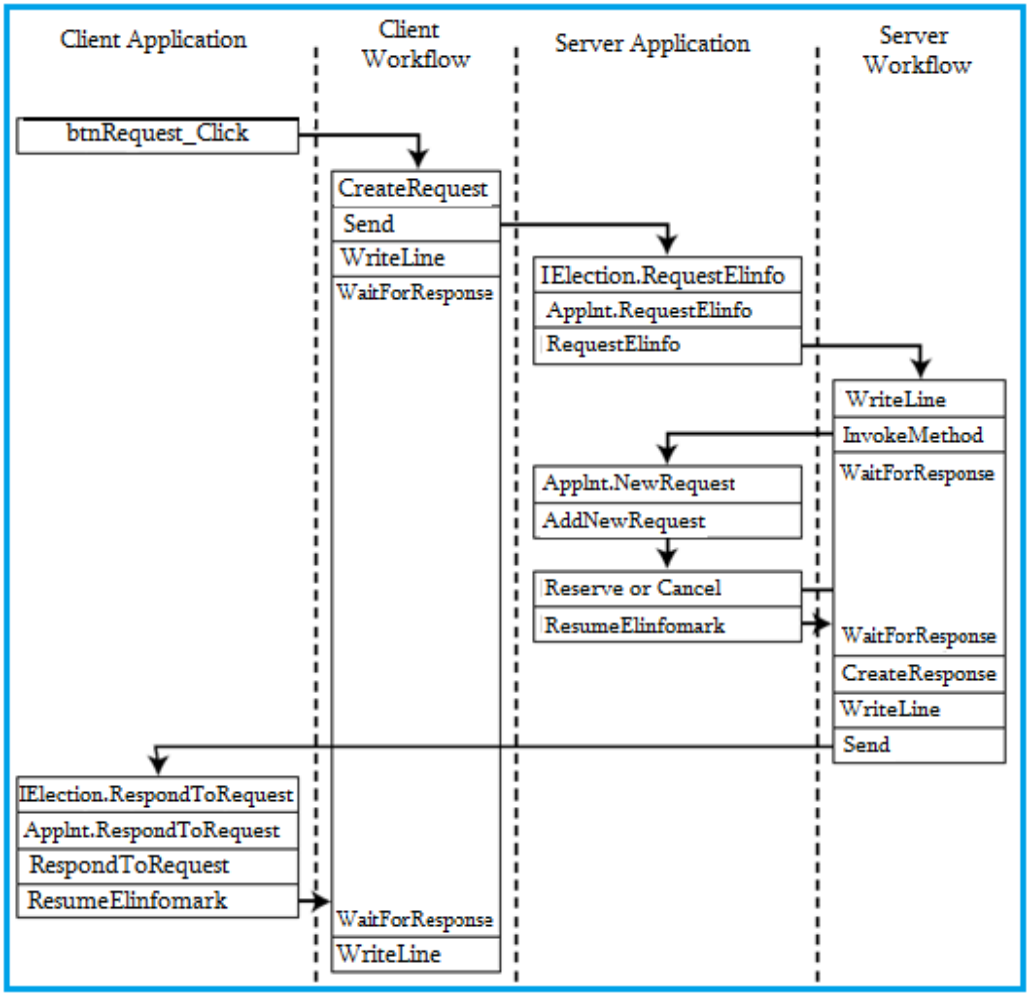
მე-20 ნახაზზე ნაჩვენებია ზოგადი ლოგიკა და შეტყობინებათა ნაკადი. ამ სქემის ელემენტები მოგვიანებით იქნება ახსნილი, ამჯერად კი განხილულ იქნება ძირითადი

ISSN 1512-1232

კონცეფციები. არ არსებობს არავითარი Receive ქმედება. მის მაგივრად აპლიკაცია მიიღებს შემავალ შეტყობინებებს, შემდეგ კი გამოიძახებს (ან აღადგენს) მუშა პროცესს.

6.3.8. შეტყობინებათა მიღება

მე-21 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.



ნახ.21. მოდულების მუშაობის ლოგიკა და შეტყობინებათა ნაკადი

გავხსნათ Election.cs ფაილი, რომელშიც გამოჩნდება ინტერფეისის შემდეგი განსაზღვრება:

```
[ServiceContract]
public interface IElectionSys
{
   [OperationContract]
   void Elinfo(ElectionRequest request);

   [OperationContract]
```

ISSN 1512-1232

```
    void RespondToRequest(ElectionResponse response);
  }
```

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.

6.3.9. 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 მე-6 ლისტინგში.

```
// ===== ლისტინგი 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 ჩვენს 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)
```

ISSN 1512-1232

```
        _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.

6.3.10. 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 მე-7 ლისტინგში MainWindow კლასისთვის ჩატვირთვის და ამოტვირთვის მოვლენათა დამმუშავებლების სარეალიზაციოდ.

```
// -- ლისტინგი 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
```

ISSN 1512-1232

```
  //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 აღარ მოხდება შეტყობინებების მიღება.

6.3.11. 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 მე-8 ლისტინგში.

```
// --- ლისტინგი 8 --- 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 },
```

ISSN 1512-1232

```
            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 მე-9 ლისტინგი -  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 მიღებულ იქნება პასუხი, მაშინ აპლიკაცია აღადგენს მუშა პროცესს, რომელიც მიმდინარეობს ElectionResponse კლასში.

ISSN 1512-1232

ყურადსაღებია, რომ მომხმარებლის ქმედება განისაზღვრება როგორც WaitForInput <ElectionResponse>, მიუთითებს რა, რომ გადასაცემი მონაცემები იქნება ElectionResponse კლასის.

6.3.12. ProcessRequest ბიზნესპროცესის რეალიზაცია

ProcessRequest ბიზნესპროცესი განსაზღვრება მოცემულია მე-9 ლისტინგში. ჩავამატოთ ეს კოდი ElectionWF.cs ფაილში.

```
// ---- ლისტინგი 9 ---------ElectionWF.cs დამატება ------
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
```

ISSN 1512-1232

```
                {
                    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.

6.3.13. აპლიკაციის რეალიზაცია

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

6.3.14. მუშა პროცესების ეგზემპლარების მხარდაჭერა

ISSN 1512-1232

აპლიკაცია ახდენს ბიზნესპროცესის ეგზემპლარების მონიტორინგს, ამიტომაც მას შეუძლია გნაახლოს სწორი ეგზემპლარი. ამის შესრულება შესაძლებელია მარტივად ობიექტის ლექსიკონით.

გავხსნათ Election.xaml.cs ფაილი და დავამატოთ კლასის წევრები მომხმარებლის ServiceHost _sh სტრიქონის ქვემოთ:

```
private IDictionary<Guid, WorkflowApplication> _incomingRequests;
private IDictionary<Guid, WorkflowApplication> _outgoingRequests;
```

ისინი იყენებს ბიზნესპროცესის 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!!!

  )
```

6.3.15. 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 მე-10 ლისტინგში, Election.xaml.cs ფაილში.

```
// --- ლისტინგი 10 -----Click Event -ის რეალიზება -----
private void btnRequest_Click(object sender,RoutedEventArgs e)
 {
   // ობიექტის ლექსიკონის Setup პარამეტრების მისაწოდებლად ---
```

ISSN 1512-1232

```
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.

მე-11 ლისტინგი აღწერს ამ ღილაკებისთვის მოვლენათა დამმუშავებლების რეალიზაციას. დავამატოთ ეს მეთოდები Election.xaml.cs კლასში.

// -- ლისტინგი 11 --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);
      }
  }
```

ISSN 1512-1232

მოვლენის დამმუშავებლები იღებს ბიზნესპროცესის 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 განახლედება (resumed).

6.3.16. 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 მე-12 ლისტინგის მიხედვით.

//--ლისტინგი 12--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.

ISSN 1512-1232

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 იქნას ბიზნესპროცესის შესრულებად ნაკადში, Dispatcher კლასი გამოიყენებს შესასრულებლად Add () მეთოდს main window-ის შესრულებადი ნაკადით. Election.xaml.cs-ის სრული რეალიზაცია მოცემულია მე-13 ლისტინგში.

// -- ლისტინგი 13 -- 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();
```

ISSN 1512-1232

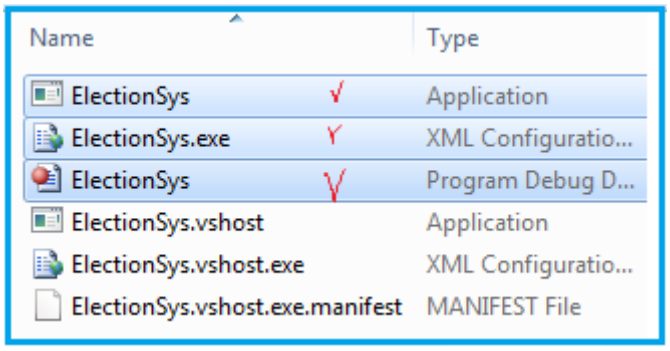
```
    _sh.AddServiceEndpoint(typeof(IElectionSys), bBinding,  szAddress);
    // ServiceHost-ის გახსნა შეტყობინებების მისაღებად (listen) ---
      _sh.Open();
   }
 private void Window_Unloaded(object sender, RoutedEventArgs e)
    {
     // service host-ის დატოვება
       _sh.Close();
   }
 // ----- 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();
       }
```

ISSN 1512-1232

```
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);
    }
  }
}
```

6.3.17. 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 22-ე ნახაზზეა ნაჩვენები.



ნახ.22. ფილიალის ფოლდერში ფაილების დაკოპირება

გავხსნათ ElectionSys.exe.config ფაილი (ფილიალის ქვეფოლდერში) და შევასწოროთ შემდეგნაირად:

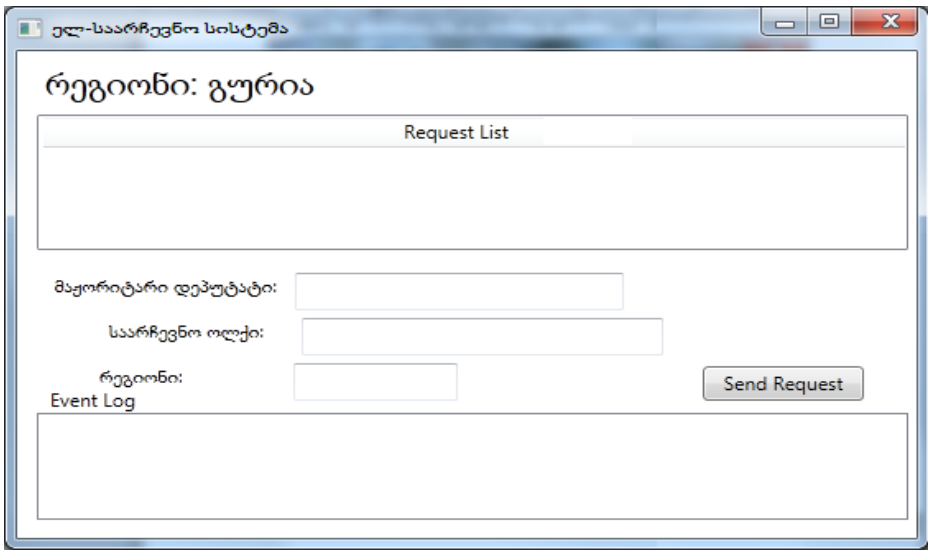
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Branch Name" value="Olqi Ozurgeti"/>
    <add key="ID" value="{43E6DADD-4751-4056-8BB7-7459B5C361AB}"/>
```

ISSN 1512-1232

```
    <add key="Address" value="8730"/>
    <add key="Request Address" value="8000"/>
  </appSettings>
</configuration>
```

შენიშვნა: თუ შედეგი მიღებულია შეცდომით, უნდა ვცადოთ აპლიკაციის გაშვება ადმინისტრატორის უფლებებით (!).

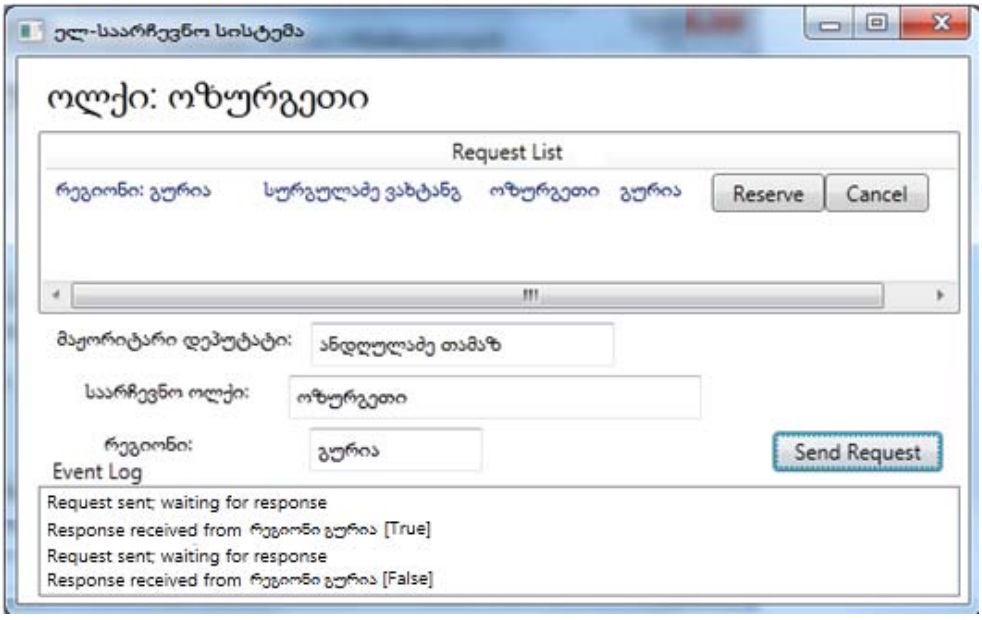
ფილიალის ფოლდერში ElectionSys.exe ფაილი ორჯერ დავკლიკოთ. აპლიკაცია ასე გამოიყურება (ნახ.23).



ნახ.23. ფილიალის ფორმა

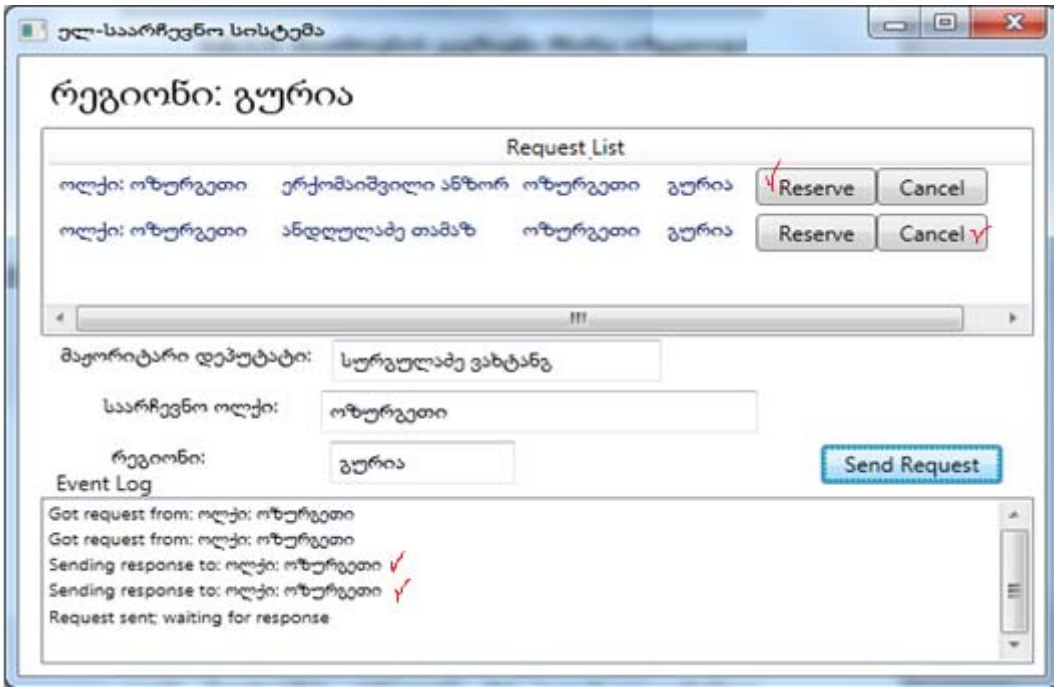
Visual Studio-ში F5-ით გავმართოთ აპლიკაცია. ანალოგიური ფანჯარა უნდა მივიღოთ, ოღონდ სათაურით - „საარჩევნო ოლქი“ (ან „რეგიონალური საარჩევნო კომისია“ ან „ცენტრალური საარჩევნო კომისია“ და ა.შ.).

ერთ-ერთ აპლიკაციაში შევიტანოთ მაჟორიტარი დეპუტატის გვარი, მხარე (ოლქი), რეგიონი და ავამოქმედოთ ღილაკი „მოთხოვნის გაგზავნა“ (ნახ.24). მოთხოვნა უნდა გამოჩნდეს მეორე ფანჯრის მოთხოვნების სიაში. დააჭირეთ Reserve ღილაკს მეორე აპლიკაციაში (ნახ.25). გამოჩნდება შეტყობინება პირველი ფანჯრის მოვლენების ჟურნალში, რომ პასუხი მიღებულია.



ISSN 1512-1232

ნახ.24. მოთხოვნის გაგზავნა ოლქიდან „ოზურგეთი“



ნახ.25. მოთხოვნის დამუშავება რეგიონში „გურია“

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

7. დასკვნა

ამრიგად, ჩატარებული სამუშაოების შედეგად შეიძლება შემდეგი სახის დასკვნების ჩამოყალიბება:

- 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, პროცეს-ორიენტირებული და სერვის-ორიენტირებული მოდელირების მეთოდების გამოყენება;
- 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 ელქტრონული საარჩევნო სისტემის აგების კონცეფცია;
- ელექტრონული საარჩევნო სისტემის დაპროექტება, აპარატურული და პროგრამული რეალიზაცია უნდა განხორციელდეს მულტიმედიალური საშუალებების, განაწილებული, რელაციური ბაზების, კლიენტ-სერვერ არქიტექტურის და უსაფრთხო, საიმედო ქსელების ბაზაზე;
- ელექტრონული საარჩევნო სისტემის მულტიმედიალური მონაცემთა რელაციური ბაზების ეფექტურად დასაპროექტებლად და ასაგებად მიზანშეწონილია

ISSN 1512-1232

თანამედროვე CASE-ტექნოლოგიების გამოყენება, მოდელირების კატეგორიალური თეორიის და დაპროექტების ობიექტ–ორიენტირებული მეთოდების საფუძველზე;

- კლიენტ–სერვერ არქიტექტურის ლოგიკურად ერთიანი და ფიზიკურად განაწილებული მონაცემთა რელაციური ბაზების სისტემის დამუშავება შესაძლებელი გახდა ობიექტ–როლური მოდელირების პრინციპების საფუძველზე და შესაბამისი გრაფო–ანალიზური ინსტრუმენტების გამოყენებით;

- შემუშავებულია აგებული სისტემის მომხმარებელთა ინტერფეისები, ინსტრუქციები, დანერგვის და ექსპლუატაციის პროცესების ორგანიზაციული, ტექნიკური და იურიდიული ასპექტები. განსაზღვრულია სისტემისათვის საჭირო ტექნიკური საშუალებები და მათი შესაძლებლობები.

-

ლიტერატურა:

1. მეიერ-ვეგენერი კ., სურგულაძე გ., ბასილაძე გ. საინფორმაციო სისტემების აგება მულტიმედიური მონაცემთა ბაზებით. მონოგრაფია. სტუ. თბილისი. 2014.
2. ბასილაძე გ. მულტიმედიალური მონაცემთა საცავის დაპროექტება და რეალიზაცია კლიენტ-სერვერ არქიტექტურით. დისერტაცია. აკად.დოქტ. სტუ, თბილისი, სტუ, 2013.
3. სურგულაძე გ., თოფურია ნ., ბასილაძე გ., ურუშაძე ბ., ლომიძე მ., გაბინაშვილი ლ. პროგრამული სისტემების მენეჯმენტი მულტიმედიალური აპლიკაციების დასაპროექტებლად და ასაგებად. VI საერთ. სამეცნ.პრაქტ. კონფ. „ინტერნეტი და საზოგადოება“. აკ.წერეთლის სახ.უნივ. ქუთაისი, 2013. გვ. 66-70
4. ბასილაძე გ. ელექტრონული საარჩევნო სისტემის მხარდამჭერი IT-ინფრასტრუქტურის დამუშავება. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(8), თბილისი, 2010, გვ. 223–226.
5. 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. Tbilisi, 2012, pp. 308-312.
6. Basiladze G.Multimedia Databases and IT-Infrastructure of an Electronic Election System. Intern. Sc.Conf.: “Automated Control Systems & new IT”, Tbilisi, 2011, pp. 161–162.
7. ბასილაძე გ., სურგულაძე გ., გაბინაშვილი ლ. მულტიმედიალური ელექტრონული საარჩევნო სისტემის პროგრამული უზრუნველყოფის დამუშავება. სტუ-ს შრ.კრ. „მართვის ავტომატიზებული სისტემები“. N1(14), თბილისი, 2013, გვ. 234-239.
8. Halpin T. ORM-2 Graphical Notation. Neumont Univer., http://www.orm.net/ pdf/ORM2_TechReport1.pdf , 2005.
9. Codd E.F. (1972). 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.
10. Fagin R.. A Normal Form for Relational Databases That Is Based on Domains and Keys. IBM Research Laboratory. ACM Transactions on Database Systems, 1981,Vol. 6, No. 3, pp. 387-415.
11. Чоговадзе Г., Сургуладзе Г., Качибая В. Теория реляционных зависимостей и проектирование логической схемы баз данных. Тб. Госуд. Унив., Тбилиси: “Ганатлеба”, 1988.
12. ჩოგოვაძე გ., გოგიჩაიშვილი გ., სურგულაძე გ., შეროზია თ., შონია ო. მართვის ავტომატიზებული სისტემების დაპროექტება და აგება. სტუ, თბილისი, 2001.

ISSN 1512-1232

13. Сургуладзе Г., Качибая В., Кортуа Т. О выборе приемлемых нормальных форм логической схемы реляционных БД. Сб.научн. тр. ГПИ., "Техническая кибернетика", Тбилиси, 1983, N 10(267), ст. 47-51.
14. Wedekind H., Surguladze G. Technology of Designing of Distributed Systems on the Basis of Objectoriented Programming. ISSN 021-7164, GTU, Tbilisi,1996, pp.96-100.
15. ჩოგოვაძე გ., სურგულაძე გ., შონია ო. მონაცემთა და ცოდნის ბაზების აგების საფუძვლები. სტუ, თბილისი, 1996.
16. Wang C., Wedekind H. Segment Synthesis in Logocal Data Base Design. IBM J. RSD 19, 1975, N1, pp.71-77.
17. Petzold Ch. Applications=Code+Markup. A Guide to the MicroSoft Windows Presentation Foundation. St-Petersburg, 2008.
18. Collins M.J. Beginning WF: Windows Workflow in .NET 4.0. USA. http://www.ebooks-it.net/ebook/beginning-wf, 2010.
19. Anurag S. WCF: From a Beginner's perspective & a Tutorial. http://www.codeproject.com/Articles/566691/WCF-From-a-Beginners-perspective-a-Tutorial, V, 4 2013.

Article received: 2014-03-25