საინფორმაციო სისტემების ინტეგრაციის პროცესების მენეჯმენტი სერვის-ორიენტირებული არქიტექტურით და UML/Agile ბაზირებული მეთოდებით

გია სურგულაძე, ირაკლი ბულია, ბექა ურუშაძე

რეზიუმე

განხილულია კორპორაციათა მართვის საინფორმაციო სისტემების ინტეგრაციის პრობლემები, მათი ობიექტ-, პროცეს- და სერვის-ორიენტირებული ანალიზის, დაპროექტების და რეალიზაციის მიზნით. პროგრამული სისტემის სასიცოცხლო ციკლის მენეჯმენტისთვის გამოყენებულია UML/Agile ბაზირებული მეთოდები, განსაკუთრებით ექსტრემალური დაპროგრამების კონცეფცია. ექსპერიმენტული ნაწილი განხორციელდა ფინანსური ოპერაციების მონიტორინგის ამოცანისთვის კომერციული ბანკებისა და შემოსავლების სამსახურის სერვის-პროცესების ინტეგრაციის მიზნით.

საკვანძო სიტყვები: მართვის საინფორმაციო სისტემები. პროგრამული სისტემების მენეჯმენტი. ინტეგრაცია. სერვის-ორიენტირებული არქიტექტურა. პროგრამული უზრუნველყოფის სასიციცხლო ციკლი. UML. ექსტრემალური დაპროგრამება.

1. შესავალი

კორპორაციათა მართვის საინფორმაციო სისტემების ინტეგრაციის (Enterprise Application Integration) ამოცანა მდგომარეობს ამ კომპანიებში უკვე არსებული და ახლად შესაქმნელი აპლიკაციების ურთიერთდაკავშირებაში, მათ სინქრონიზაციასა და სერვისების მართვის გამარტივებაში, რაც მნიშვნელოვნად უწყობს ხელს განსახილველ სფეროში ბიზნეს-პროცესების ავტომატიზაციას [1]. ამ კავშირების უზრუნველყოფის მიზნით იქმნება სხვადასხვა მედიატორული პროგრამები, რომლებიც ასრულებს დამაკავშირებელ, შუალედურ რგოლს ამ სისტემებს შორის. მსგავსი ბროკერული აპლიკაციების რაოდენობა იზრდება, რაც უფრო იზრდება ორგანიზაციაში სისტემების და ასევე გარე ორგანიზაციებთან არსებული კავშირები. საჭირო ხდება ამ ბროკერული აპლიკაციების შექმნა, იზრდება მათი ადმინისტრირების, მონიტორინგის, პროგრამული უზრუნველყოფის ცვლილებების მართვის სამუშაოები, რაც იწვევს ინფორმაციული ტექნოლოგიების რესურსების დიდი ოდენობით მოხმარებას. საჭირო ხდება როგორც ადამიანური რესურსების, ასევე ინფრასტრუქტურული, პროგრამული უზრუნველყოფების მუდმივი განახლება. თითოეული ბროკერული აპლიკაციის შექმნა არის ხანგრძლივი და შრომატევადი პროცესი.

სხვადასხვა სისტემები განსხვავებული ფორმატის მონაცემებს იძლევა გარე სისტემებთან დასაკავშირებლად. შესაძლებელია ეს იყოს ტექსტური ან ორობითი ფაილი, მონაცემები XML ფორმატში, იძლეოდეს მონაცემებზე წვდომის საშუალებას ვებ-სერვისებით, ან ბაზის სხვადასხვა პროცედურების და ფუნქციების საშუალებით, ამასთანავე მონაცემთა გადაცემა ხდებოდეს TCP, HTTP პროტოკოლების ან სხვა საშუალებების გამოყენებით. საჭიროა ისეთი სახის ინტეგრაციული არქიტექტურის შექმნა, რომელიც კომპანიაში მარტივად უზრუნველყოფს ინტეგრაციის ამოცანების გადაწყვეტას ნაკლები რესურსების გამოყენებით [2].

ამოცანა მდგომარეობს ისეთი პროგრამული აპლიკაციის შექმნაში, რომელიც უზრუნველყოფს არსებული სისტემების ინტეგრაციას როგორც სხვადასხვა ორაგნიზაციებს შორის, ასევე ორგანიზაციის შიგნით, რითაც გარანტირებულ იქნება განსხვავებული ფორმატის მონაცემების მიღება-გადაცემა-დამუშავება, მაღალ წარმადობა და საიმედოობა [3].

პროგრამული აპლიკაციების ინტეგრაციას ძირითადად აქვს სამი დანიშნულება:

- მონაცემთა ინტეგრაცია, რაც უზრუნველყოფს მონაცემების იდენტურობას სისტემებს შორის;
- აპლიკაციათა მიმწოდებლებზე დამოუკიდებლობა. უზრუნველყოფს, რომ აპლიკაციის ცვლილების შემთხვევაში, ბიზნეს-პროცესი და ბიზნეს წესების ხელახალი შექმნა არ იყოს საჭირო;
- ინტერფეისის შექმნა, რაც უზრუნველყოფს აპლიკაციებთან ურთიერთობის ერთიანი სტანდარტის შექმნას, რომელიც საშუალებას იძლევა აპლიკაციებთან კომუნიკაცია შესრულდეს მათი შიგა სტრუქტურების შესწავლის გარეშე.

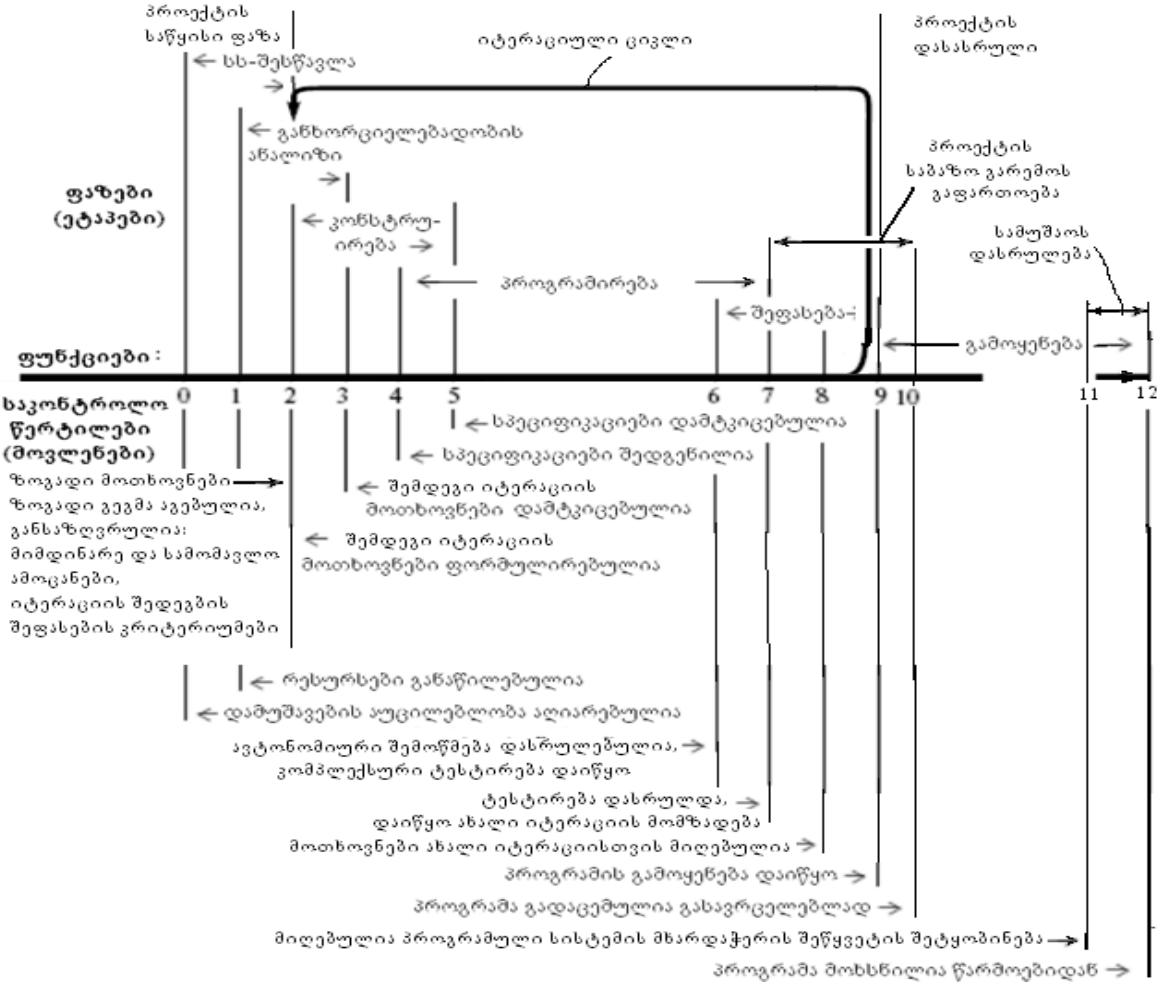
ამ ამოცანის რეალიზაცისათვის საჭიროა გამოყენებულ იყოს ინტეგრაციის აპრობირებული მეთოდები, პროგრამული უზრუნველყოფის არქიტექტურის მოდელები, სტანდარტები, რომლებიც უზრუნველყოფს მოქნილობას, მასშტაბირებას, არაერთგვაროვანი და კომპლექსური სისტემების ურთიერთკავშირს.

2. ძირითადი ნაწილი

სრულყოფილი და საიმედო, მოქნილი პროგრამული უზრუნველყოფის (Software Engineering) სწრაფად დაპროექტება, რეალიზაცია, დანერგვა და შემდგომი თანხლება სისტემის დამკვეთ ორგანიზაციაში მეტად მნიშვნელოვანი ამოცანაა და მისი ეფექტურად გადაწყვეტა ბევრადაა დამოკიდებული როგორც საპროექტო-დეველოპმენტის გუნდის შემადგენლობასა და გამოცდილებაზე, ასევე IT-ინფრასტრუქტურასა და CASE-ინსტრუმენტებზე.

ხშირად შეუძლებელია სრულყოფილი და საიმედო სისტემების აგება „სწრაფად“ (მოქნილად - Agile). ობიექტ-ორიენტირებული დაპროგრამების მეთოდი, რომელიც უნიფიცირებული მოდელირების ენის (UML) საშუალებით დამკვიდრდა, უნივერსალურია და მისი გამოყენებით პროგრამის სასიცოცხლო ციკლი მოითხოვს მისი აუცილებელი ეტაპების იტერაციულ განვითარებას [4,5].

1-ელ ნახაზზე ნაჩვენებია ეს ეტაპები განტერის მოდელის საფუძველზე იტერაციული ბიჯებით [6].



ნახ.1. განტერის სასიცოცხლო ციკლის მოდელი იტერაციით ობიექტ-ორიენტირებული დაპროგრამების მეთოდისთვის

პროგრამული სისტემის მენეჯმენტის საკონტროლო წერტილებში (0-12), ეტაპების მიხედვით ხორციელდება იტერაციული სამუშაოები (დაბრუნება უკანა წერტილებში განმეორებითი პროცედურების ჩასატარებლად), სისტემის ფუნქციონალობის სისრულის დაზუსტების ან გაფართოების მიზნით. ექსტრემალური პროგრამირების მეთოდის სასიცოცხლო ციკლის მოდელში ძირითადი ყურადღება მახვილდება საპრობლემო ამოცანის სწორად ჩამოყალიბებაში დამკვეთის მიერ ბიზნეს-ანალიტიკოსთან ერთად, ნაკლებად იხარჯება დრო უნივრსალური დიაგრამების აგებასა და საანგარიშო დოკუმენტაციის გაფორმებაზე, და რა თქმა უნდა, ხდება ძირითადი ეტაპების (კონსტრუირება-დაპროგრამება) ფაზათა შერწყმა [7].

აქედან გამომდინარე, პროგრამული სისტემის მენეჯერი, კონკრეტული პროექტის ამოცანებისა და მოთხოვნების შესაბამისად, უნდა განსაზღვრავდეს როგორც პროგრამირების მეთოდის, ეტაპთა ფაზების და იტერაციათა მოთხოვნების შერჩევა-ფორმირებას, ასევე მუშა გუნდის შემადგენლობას. ამ პროცესში მონაწილე როლებია: დამკვეთი, პროექტის მენეჯერი, ბიზნეს-პროცესების სპეციალისტი (ბიზნეს-ანალიტიკოსი), სისტემის არქიტექტორი, დეველოპერი-პროგრამისტი, ტესტირების სპეციალისტი და სხვ. მე-2 ნახაზზე მოცემულია აქტიურობათა დიაგრამის ფრაგმენტი „საპროექტო გუნდის ფორმირების“ პროცედურიდან „წინასაპროექტო კვლევის რეპორტის შედგენის“ პროცედურის ჩათვლით [6]. პროგრამული სისტემის პროექტის მენეჯერი ახორციელებს ყველა საკონტროლო წერტილის მონიტორინგს.

დიდი პროექტებისათვის, რომელშიც რესურსები და დროითი ფაქტორები, შედარებით კრიტიკული არაა, ხდება ობიექტ-ორიენტირებული მიდგომის ყველა ეტაპის და ფაზის გამოყენება შესაბამისი საკონტროლო წერტილების აუცილებელი მონიტორინგით და რეპორტებით. ამ დროს სრული მოცულობით ხორციელდება უნიფიცირებული მოდელირების ენის (UML/2) და შესაბამისი ინსტრუმესნტული საშუალების, მაგალითად, Enterprise Architect პაკეტის გამოყენება [8].

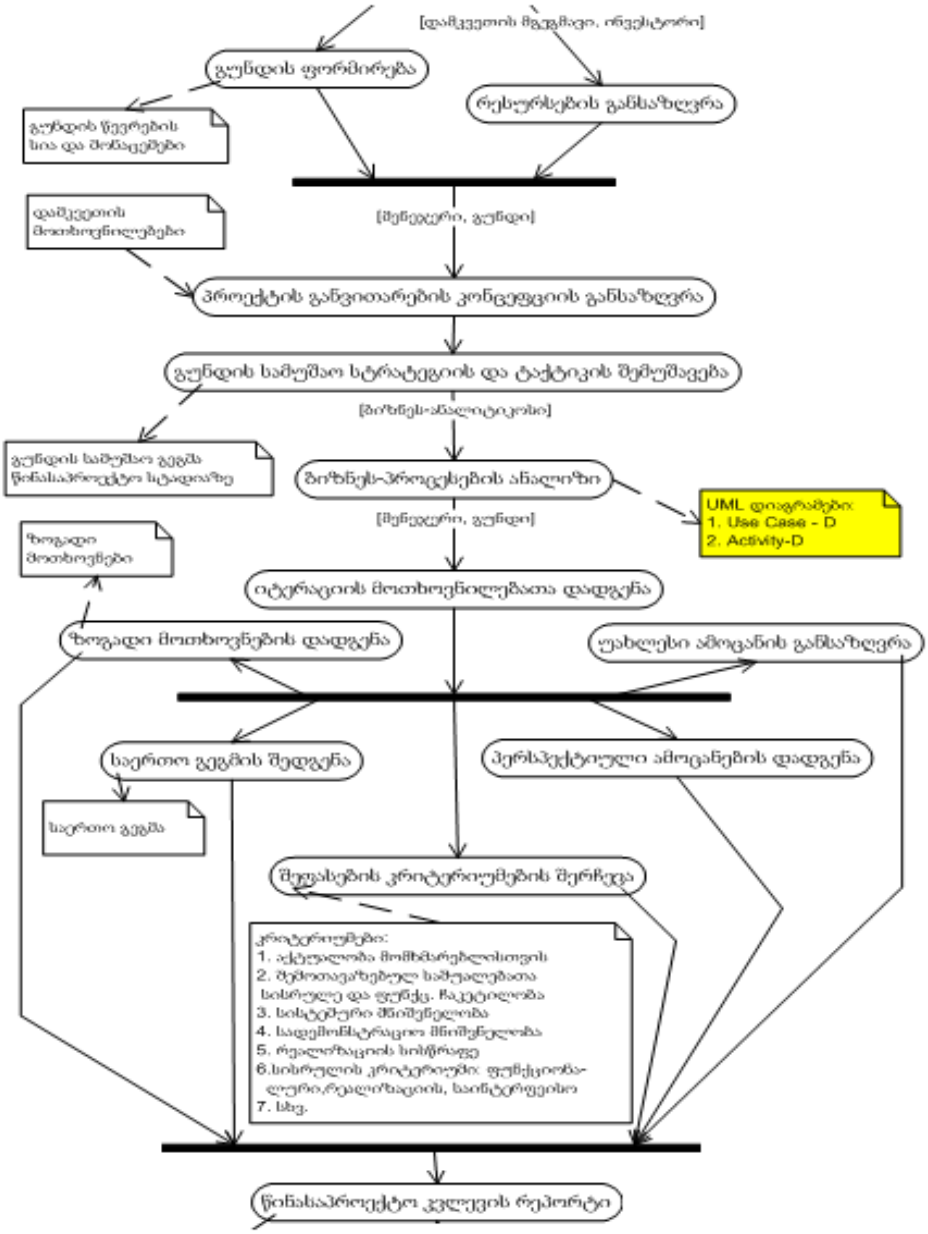
ექსტრემალური მიდგომის მეთოდის გამოყენებისას, მცირე ან საშუალო პროექტებში, სადაც სისტემების რეალიზაციის დრო კრიტიკულია, ძირითადი ყურადღება ამოცანის სწორად ჩამოყალიბების, დეველოპინგის და ტესტირების ეტაპებზე გადადის. კოდის გენერაციის, ტესტირების შემთხვევათა მოდელირების და რეფაქტორინგის პროცესების კომბინაციით მიიღწევა მნიშვნელოვანი სინერგიული ეფექტები, რომელთაც განსაკუთრებული მნიშვნელობა აქვს პროგრამული პაკეტების ხარისხის მართვის ამოცანების გადასაწყვეტად [9].

3. ინტერკორპორაციული სისტემის ბიზნეს-პროცესების აღწერის BPMN დიაგრამები ვებ-სერვისული მეთოდებით

ინტერკორპორაციული სისტემის მაგალითისათვის განვიხილავთ ფინანსთა სამინისტროს (MOF) შემოსავლების სამსახურისა და საფინანსო ბანკების ერთიან ელექტრონულ სისტემის პროექტს [10].

ბანკსა და შემოსავლების სამსახურს შორის ინფორმაციის გაცვლა ხდება პაკეტებით. ეს პაკეტები აგებულია XML-ფაილების საფუძველზე, რაც მნიშვნელოვნად ეფექტურს ხდის ინტერნეტის გამოყენებას ამ სფეროში. მაგალითად, შემოსავლების სამსახური ბანკში აგზავნის გარკვეული დანიშნულების საინფორმაციო შეტყობინების XML-ფაილს. ბანკის მხარეს არის რეალიზებული ვებ-სერვისი AcceptMessageFromMOF. ეს XML შეიცავს ინფორმაციას ინკასოების დადების/ ცვლილების/გაწვევის, ყადაღების დადების/გაწვევის, კლიენტების რეგისტრაციაზე ან დახურვაზე დასტურის შეტყობინებებს და ა.შ. ბანკიც თავის მხრივ აგზავნის ერთ XML-ს, რომელიც შეიცავს ინფორმაციას ახალი კლიენტების რეგისტრაციის, კლიენტების დახურვის დასტურის, ინკასოს თანხების, კლიენტების ტიპების ცვლილების შესახებ. MOF სერვისის

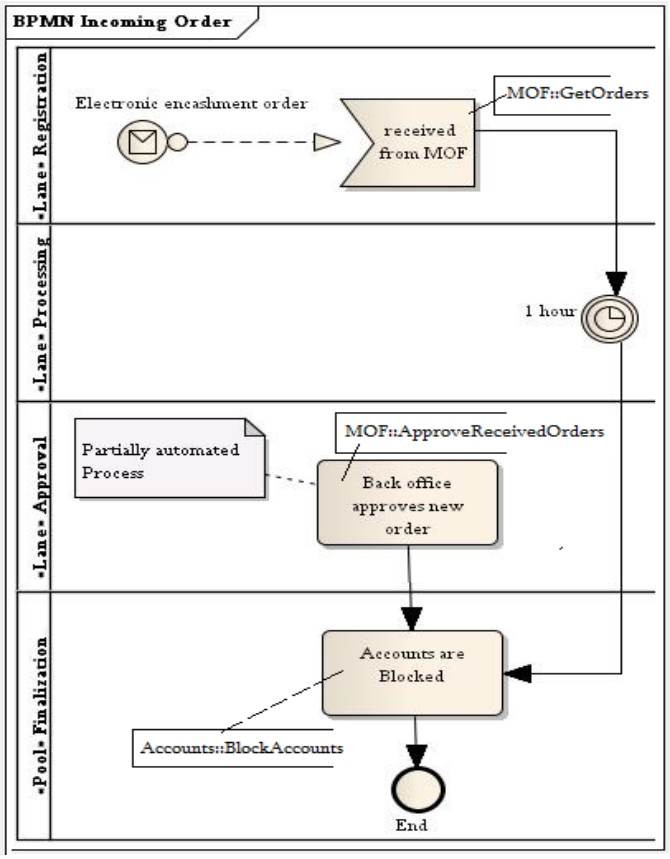
მეთოდების გამოძახებით ბანკი აგროვებს ინფორმაციას დროებით ცხრილებში და შემდგომ იძახებს შემოსავლების სამსახურის AcceptMessageFromBank ვებ-სერვისს, რომელსაც პარამეტრად გადაეცემა ბანკიდან გაგზავნილი ინფორმაცია.



ნახ.2. აქტიურობათა დიაგრამის ფრაგმენტი

ახლა განვიხილოთ BPMN ინსტრუმენტით აგებული ჩვენი კონკრეტული საპრობლემო სფეროს ბიზნესპროცესების მოდელები, რომელთა საფუძველზეც დამუშავდა შესაბამისი პროგრამული უზრუნველყოფა. კერძოდ შევარჩიეთ საფინანსო ბანკებსა და ფინანსთა სამინისტროში „ინკასო-ოპერაციების“ ავტომატიზაციის პროცედურები სერვის-ორიენტირებული არქიტექტურით (ნახ.3-5).

მე-3 ნახაზზე ასახულია კლიენტის ანგარიშებზე ინკასოს წარმოდგენის მოთხოვნის სერვისული ბიზნესპროცესის სქემა.



ნახ.3. Incoming Order(): სერვისებით (GetOrderRecalls, ReceiveApprovedAmount, BlockAccounts)

GetOrderRecalls- იძახებს შემოსავლების სამსახურიდან მიღებული ინკასოების გაწვევების ან ცვლილებების შესახებ მონაცემებს (კლიენტის იდენტიფიკატორებს და ინკასოს რედაქტირებულ თანხას, რა თანხის გადარიცხვაც უნდა მოხდეს კლიენტის ანგარიშებიდან. შესაძლოა ვალდებულების თანხა განულდეს, რაც ნიშნავს, რომ კლიენტს მოეხსნა ინკასო);

ReceiveApprovedAmount - შემოსავლების სამსახურიდან მიღებული ინფორმაციიდან (XML), რომელიც დაბრუნდა GetXML მეთოდით, კლიენტის თანხების დამუშავება;

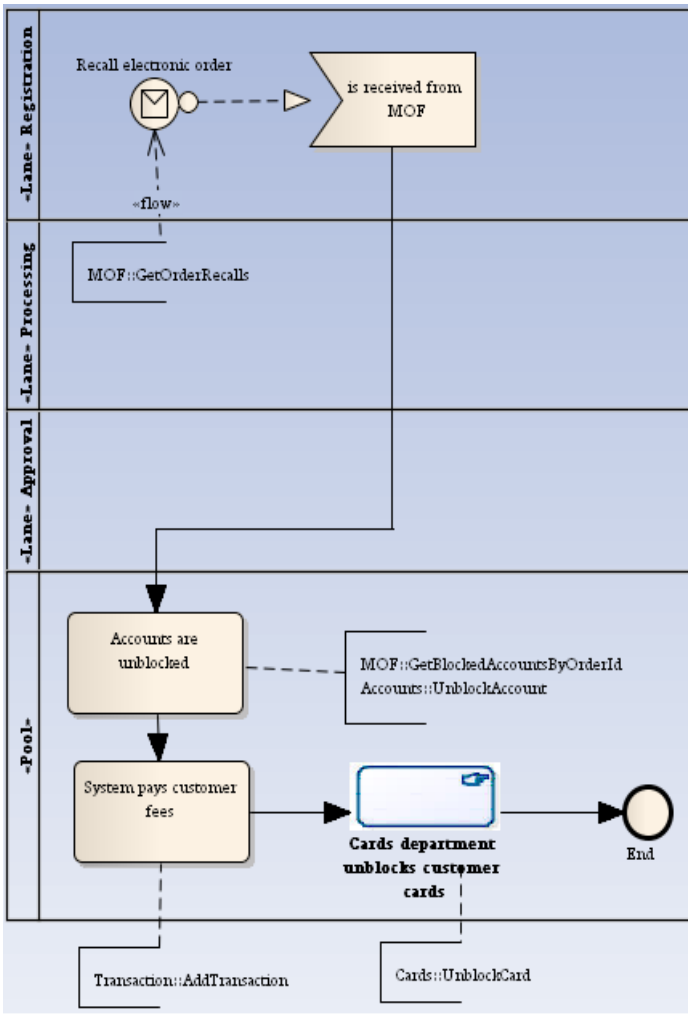
BlockAccounts - გადაეცემა კლიენტის იდენტიფიკატორი და სისტემა ასრულებს ამ კლიენტის ანგარიშების სტატუსის გადაყვანას „მხოლოდ ბიუჯეტურში“, რის შემდეგაც კლიენტი ვეღარ გამოიყენებს ამ ანგარიშებს, გარდა თანხის ბიუჯეტში გადასარიცხად.

ფუნქციონალობის აღწერა: კლიენტის ანგარიშებზე ინკასოს წარმოგენისთვის შემოსავლების სამსახური აფორმირებს XML ფორმატის შეტყობინებას, რომელიც შეიცავს კლიენტის მონაცემებს და თანხის ოდენობას.

ეს შეტყობინება იგზავნება ბანკის ვებ-სერვისს, რომელიც ცხრილში ჩაწერს ამ მონაცემებს. ოპერატორი ინკასოების მართვის მოდულში ნახულობს შემოსული ინკასოების სიას და ადასტურებს. დადასტურებულ ინკასოებზე Windows სერვისი კლიენტის ანგარიშებს ანიჭებს სტატუსს „მხოლოდ ბიუჯეტური“.

მე-4 ნახაზზე ასახულია ინკასოების გაწვევის მოთხოვნის სერვისული ბიზნეს–პრცესის სქემა.

ფუნქციონალობის აღწერა: ინკასოების გაწვევა ბანკს გადმოეცემა XML-ის სახით AcceptMessage პროცედურა.



ნახ.4. Incoming Order Recall(): სერვისებით

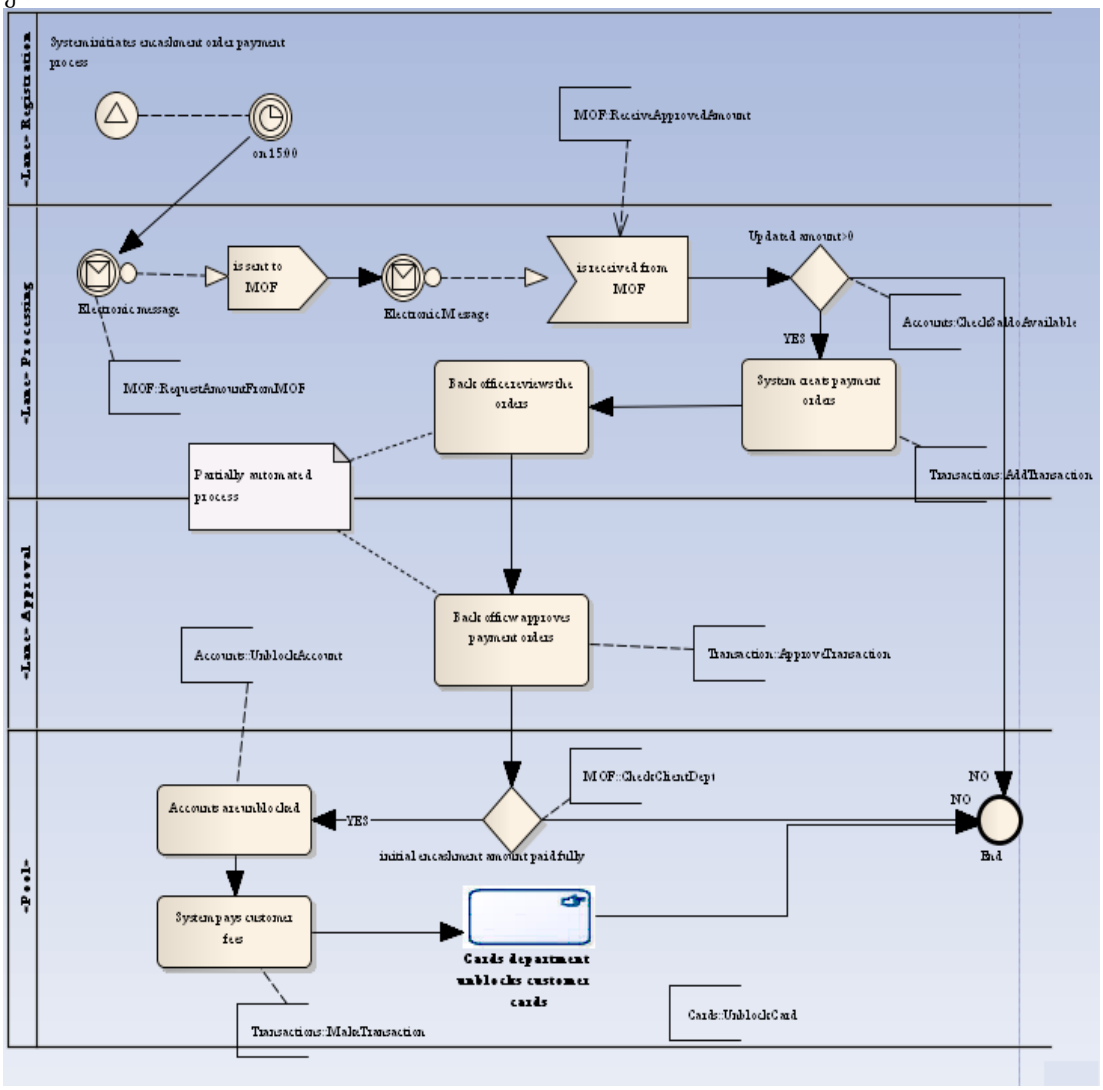
მიღებული XML-ის მონაცემები იწერება ცხრილში და შემდგომ გამოიძახება GetOrderRecalls ვებ-სერვისი, ამ სიის მიხედვით ხდება უკვე ინკასოდადებულ ანგარიშების მოძებნა, GetBlockedAccouonts-ByOrderID მეთოდი აბრუნებს თითოეული ინკასოსთვის დაბლოკილ ანგარიშის მონაცემებს, ამ ანგარიშების სიის მიხედვით ხდება UnblockAccounts მეთოდის გამოძახება და ანგარიშების სტატუსის შეცვლა „აქტიურზე“.

თუ ანგარიში არის საბარათე, მაშინ გამოიძახება UnblockCard ვებსერვისის მეთოდი, რომელიც განბლოკავს ბარათებს.

მე-5 ნახაზზე ასახულია ბანკში მიღებული ინკასოების დამუშავების მოთხოვნის სერვისული ბიზნეს–პრცესის სქემა.

ფუნქციონალობის აღწერა: ბანკში მიღებული ინკასოების დამუშავება იწყება წინასწარ განსაზღვრულ დროს. Windows სერვისი ბაზის ცხრილებში ნახულობს დასამუშავებელი ინკასოების სიას, იძახებს GetActiveOrder-ს მეთოდს, მიღებული ინკასოების ნომრები გადაეცემა RequestAmountFromMof მეთოდს, რომელიც აფორმირებს XML შეტყობინებას და იძახებს შემოსავლების სამსახურის ვებ-სერვისს. შემოსავლების სამსახური XML-ის სახის შეტყობინებას უბრუნებს ბანკის ვებ-სერვისს, რომელიც შეიცავს ინკასოების ნომრებს და შესაბამის თანხას. ეს ინფორმაცია იწერება ბაზაში. ბანკის თანამშრომელი პროგრამის საშუალებით ნახულობს ინკასოების სიას, გამოიძახება ვებ-სერვისის მეთოდი Get, ამტკიცებს მას, რის შემდეგაც

გამოიძახება ბაზის პროცედურები, რომლებიც ატარებს ტრანზაქციებს, იხდიან ანგარიშებიდან თანხებს შემოსავლების სამსახურის ანგარიშზე. თუ თანხის დაფარვა მოხდება სრულად, შეიცვლება ანგარიშის სტატუსი „აქტიურზე“ და მოხდება ბარათების განბლოკვა თანამშრომლის მიერ.



ნახ.5. Incoming Order Fulfilment (): სერვისებით

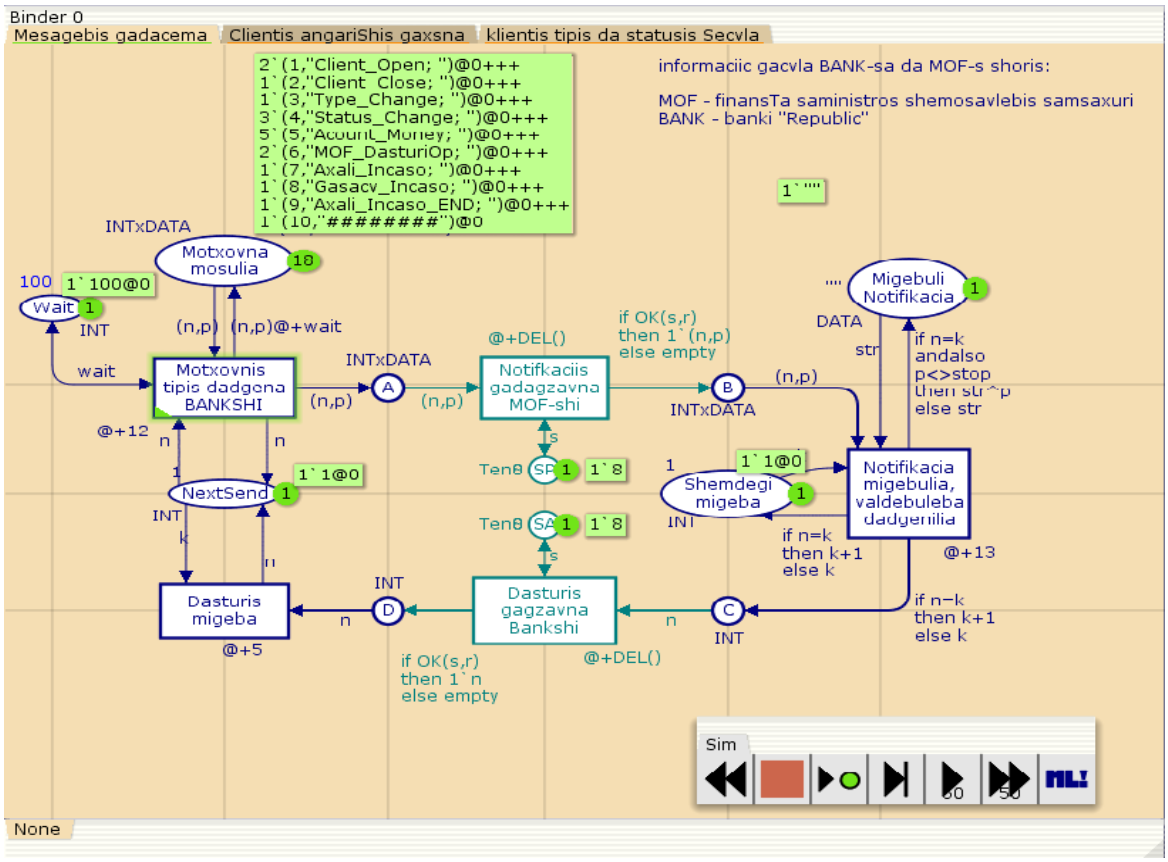
სერვის-ორიენტირებული არქიტექტურა აფართოებს ობიექტ-ორიენტირებული ტექნოლოგიის გამოყენების საზღვრებს და ინტეგრაციის შესაძლებლობას ქმნის. მთავარი განსხვავება ამ ორ ტექნოლოგიას შორის მეთოდების განზოგადებისა და წვდომის ასპექტში იკვეთება. პრინციპული განსხვავება ობიექტ-ორიენტირებულ და სერვის-ორიენტირებულ მიდგომებს შორის არის ორიენტირება ბიზნეს-პროცესზე და არა ობიექტზე. ამ პრინციპით, ვებ-სერვისში ინკაფსულირებულია ბიზნეს-ფუნქცია აბსტრაქციის მაღალი დონით, რაც უნიფიცირებული პროცესის შაბლონად შექმნის შესაძლებლობას ქმნის. სერვის-ორიენტირებული არქიტექტურის ერთ-ერთი პრინციპია ტიპიზირებული ვებ-სერვისების ბიბლიოთეკის არსებობა, მასში ამ შაბლონების შენახვა ვებ-სერვისის სახით და მისი მრავალჯერადად გამოყენება [11]. ეს არის ობიექტ-ორიენტირებული ტექნოლოგიის ობიექტების ბიბლიოთეკის ანალოგია, სადაც ობიექტი წარმოდგენილია პროცესის სახით და გატანილია სერვერზე. ამავდროულად, სერვის-ორიენტირებულ არქიტექტურის პრაქტიკული რეალიზაციაა არსებული ვებ-სერვისების ნაკრებისგან

(უმეტესად ობიექტ-ორიენტირებულ სისტემებში შექმნილი) კონკრეტული ბიზნეს-სცენარის აწყობა, რაც ორკესტრირებისა და ქორეოგრაფიის კონცეფციით ხორციელდება [10].

4. ინფორმაციის გაცვლის იმიტაციური მოდელირება სერვის-ორიენტირებული პროცესებისთვის

საფინანსო ბანკებსა და შემოსავლების სამსახურს შორის ინფორმაციის გაცვლა სერვისული ბიზნეს-პროცესების საფუძველზე ხორციელდება შეტყობინებათა და მონაცემთა პაკეტების გაცვლის სერვისების ხშირი გამოყენებით (ყოველდღიურად ბანკმა შეიძლება მიიღოს (ან გადასცეს) 1000-ზე მეტი მოთხოვნა კლიენტების შესახებ). ასეთი ინფორმაციის მენეჯმენტი მოითხოვს საიმედო აღრიცხვისა და რისკების გამორიცხვის პროცედურების გათვალისწინებას. შეტყობინებათა ერთობლიობა, რომელიც მუდმივად გადაიცემა ქსელის საშუალებით, არ უნდა დაიკარგოს და ყოველი მათგანი უნდა ექვემდებარებოდეს მკაცრ კონტროლს, უნდა შეიძლებოდეს აღდგენა, ანუ განმეორებითი პროცედურის შესრულება. ასეთი სერვისული მოთხოვნების დამუშავების მართვის პროცესის მოდელირება განხორციელებულია CPN ინსტრუმენტით (Colored Petri Net) [12,13].

მე-6 ნახაზზე მოცემულია ასეთი ქსელის ფრაგმენტი ჩვენი სისტემისათვის. აქ გადასასვლელ ბლოკებში ნაჩვენებია, მაგალითად, მოთხოვნის ტიპის დადგენა ბანკში, ნოტიფიკაციის გადაგზავნა MOF-შემოსავლების სამსახურში, ნოტიფიკაცია მიღებულია და ვალდებულება დადგენილია, MOF-დან დასტურის გადაგზავნა ბანკში, დასტურის მიღება ბანკში. თითოეული მათგანი უნდა გაიშალოს დამოუკიდებელი პეტრის ქსელით და მოხდეს მათი ანალიზი, ამასთანავე შეიქმნება ერთიანი იერარქიული სისტემა ჩადგმული პეტრის ქვექსელებით.

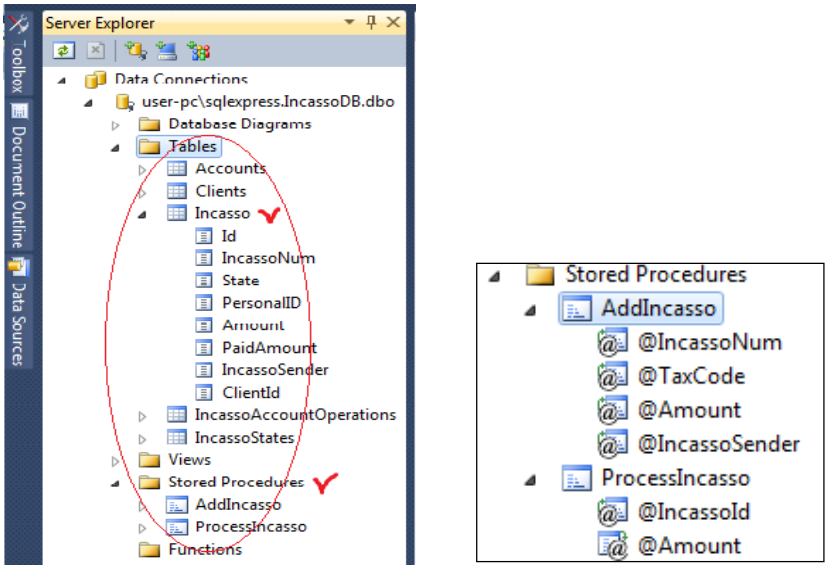


ნახ.6. MOF-BANK კორპორაციული კავშირების პროცესების იმიტაციური მოდელი CPN-ის გარემოში: „მოთხოვნების დამუშავება“

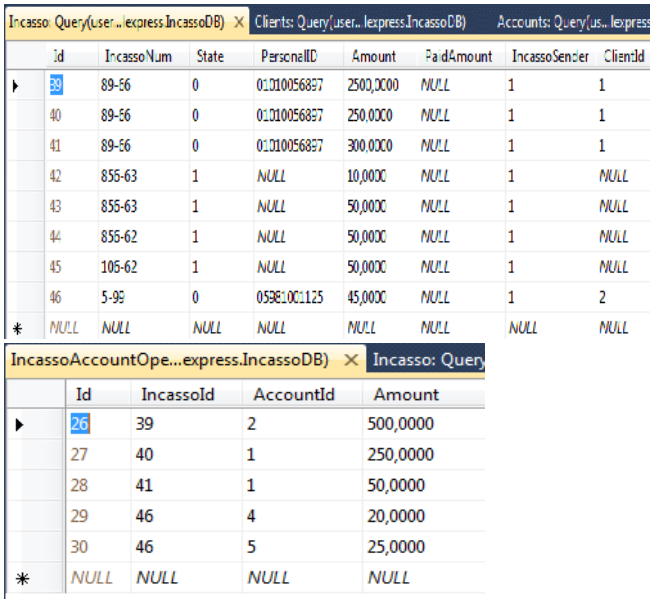
5. აპლიკაციის რეალიზაცია IncassoDB ბაზით და სერვის–ორიენტირებული არქიტექტურით

პროგრამული აპლიკაციის დეველოპინგი ნაშრომის ექსპერიმენტული ნაწილისთვის შესრულდა MsSQL Server მონაცემთა ბაზის და MsVisual Studio.NET Framework 4.0 ინტეგრირებული პაკეტის გარემოში, WPF-ტექნოლოგიის, C# და XAML ენების საფუძველზე.

მე-6 ნახაზზე ნაჩვენებია Incasso.DB მონაცემთა ბაზის სტრუქტურა Accounts, Clients, Incasso, IncassoAccountOperatio და IncassoStates ცხრილებით (იხ. ფრაგმენტი ნახ.7).



ნახ.6. IncassoDB ბაზა და შენახვადი პროცედურები



Id	IncassoNum	State	PersonalID	Amount	PaidAmount	IncassoSender	ClientId
39	89-56	0	01010056857	2500,0000	NULL	1	1
40	89-56	0	01010056857	250,0000	NULL	1	1
41	89-56	0	01010056857	300,0000	NULL	1	1
42	855-63	1	NULL	10,0000	NULL	1	NULL
43	855-63	1	NULL	50,0000	NULL	1	NULL
44	855-62	1	NULL	50,0000	NULL	1	NULL
45	105-62	1	NULL	50,0000	NULL	1	NULL
46	5-99	0	05981001125	45,0000	NULL	1	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

IncassoAccountOpe...express.IncassoDB) | Incasso: Query

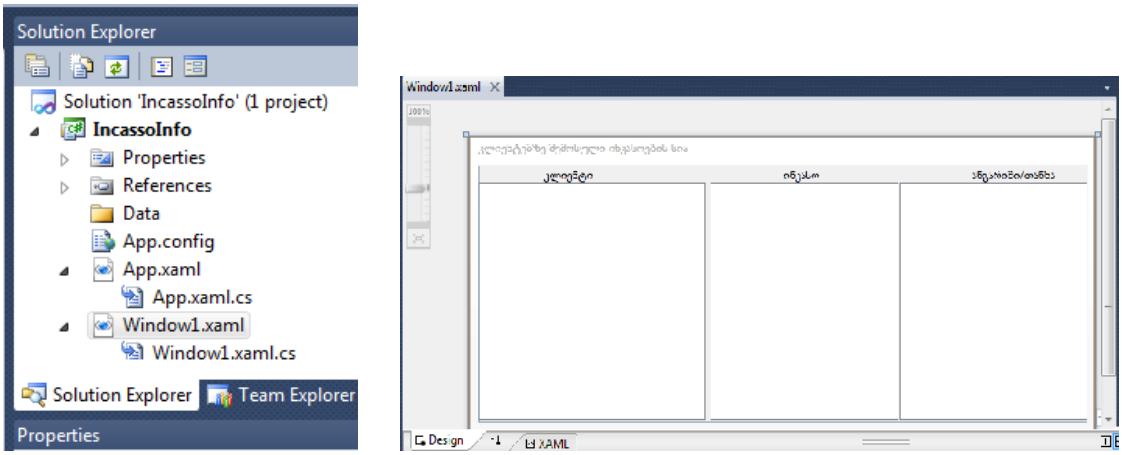
Id	IncassoId	AccountId	Amount
26	39	2	500,0000
27	40	1	250,0000
28	41	1	50,0000
29	46	4	20,0000
30	46	5	25,0000
NULL	NULL	NULL	NULL

ნახ.7.

მონაცემთა ბაზის შენახვად პროცედურებში (Stored Procedures: AddIncasso, ProcessIncasso) რეალიზებულია სერვის-ფუნქციები. ქვემოთ, 1-ელ ლისტინგში ნაჩვენებია ამ პროცედურის ტექსტი:

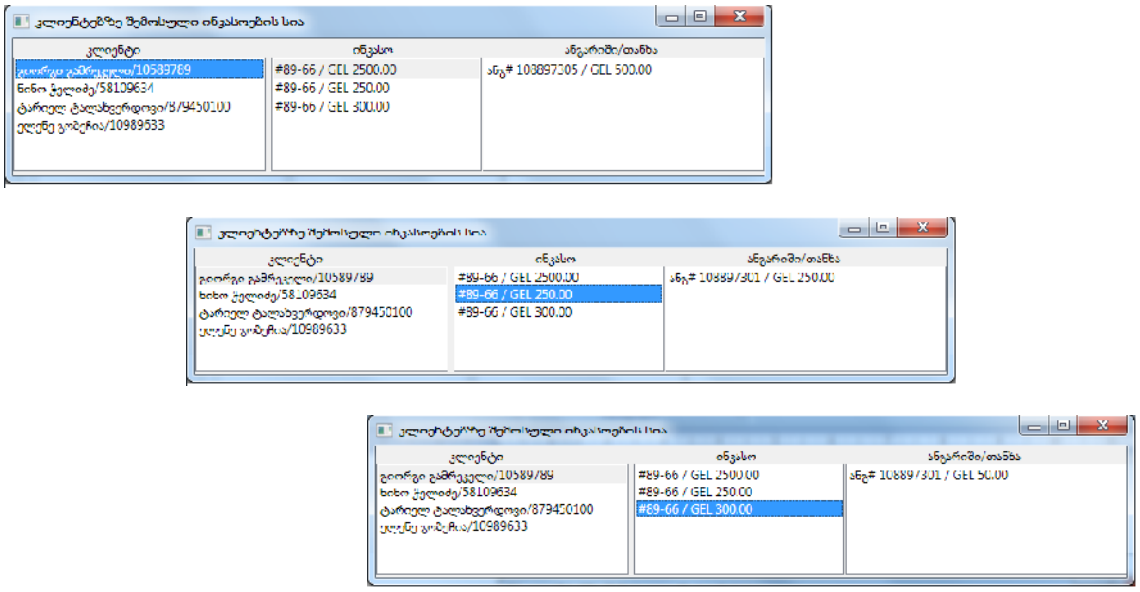
```
-- ========== AddIncasso  ====================== ლისტინგი-1========
-- Description:       ახალი ინკასოს შემოსვლა
-- ============================================
ALTER PROCEDURE [dbo].[AddIncasso]
    @IncassoNum varchar(15),
    @TaxCode varchar(15),
    @Amount money,
    @IncassoSender tinyint
AS
BEGIN
-- SET NOCOUNT ON added to prevent extra result sets from
-- interfering with SELECT statements.
        SET NOCOUNT ON;
        DECLARE @State  tinyint = 0 -- გადაუხდელი
        DECLARE @ClientId int
        DECLARE @PaidAmount money
        Declare @incasoId int
        DECLARE @PersonalID varchar(12)
        --კლიენტის მოძებნა პირადი ნომერის მიხედვით
SELECT @ClientId = Id, @PersonalID = PersonalID FROM dbo.Clients where TaxCode  = @TaxCode

        IF ISNULL(@ClientId,0)=0
        SET  @State = 1--არ არის ბანკის კლიენტი/არ არის გადასახდელი
    -- Insert statements for procedure here
        INSERT INTO [dbo].[Incasso]
            ([IncassoNum]
            ,[State]
            ,[PersonalID]
            ,[Amount]
            ,[IncassoSender]
            ,[ClientId])
     VALUES
            (@IncassoNum
            ,@State
            ,@PersonalID
            ,@Amount
            ,@IncassoSender
            ,@ClientId)
        if @@ROWCOUNT = 1 RETURN SCOPE_IDENTITY()
        RETURN -1
END
```

მე-8 ნახაზზე წარმოდგენილია VisualStudio.NET გარემოში აგებული IncassoInfo-აპლიკაცის სტრუქტურა (მარცხნივ) და სისტემის მომხმარებლის ერთ-ერთი ინტერფეისის ფრაგმენტი.



ნახ.8

მე-9 ნახაზზე ილუსტრირებულია აპლიკაციის ამუშავების შემდეგ მიღებული შედეგები ბანკში კლიენტების მიხედვით ინკასოს მდგომარეობა.



ნახ.9. მაგალითი ერთი კლიენტის 3 ინკასოს შემთხვევით

პროგრამული აპლიკაციის ინტერფეისი ჰიბრიდული WPF-ტექნოლოგიის საფუძველზე რეალიზებულია XAML-ენის გამოყენებით (ტექსტი მოცემულია მე-2 ლისტინგში).

<-- **ლისტინგი-2:** Window1.xaml ფაილი ინტერფეისის ასაგებად ------- -->

```
<Window x:Class="DataBindingRelation.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Height="406" Width="726"
    Title="კლიენტებზე შემოსული ინკასოების სია"
    MinHeight="300" MinWidth="300" WindowStartupLocation="CenterScreen"
    xmlns:local="clr-namespace:DataBindingRelation"
    >
  <Window.Resources>
      <SolidColorBrush x:Key="ControlColorBrush"
            Color="{x:Static SystemColors.ControlColor}" />
  </Window.Resources>
  <Grid Background="{StaticResource ResourceKey = ControlColorBrush}">
       <Grid Name="ClientIncasso">
           <Grid.RowDefinitions>
              <RowDefinition />
                 <RowDefinition Height="Auto" />
           </Grid.RowDefinitions>
         <Grid>
           <Grid.RowDefinitions>
              <RowDefinition Height="Auto" />
              <RowDefinition />
           </Grid.RowDefinitions>
           <Grid.ColumnDefinitions>
              <ColumnDefinition Width="238" />
              <ColumnDefinition Width="204" />
              <ColumnDefinition Width="252*" />
```

```
          </Grid.ColumnDefinitions>
          <TextBlock HorizontalAlignment = "Center" Margin="69,0,87,0"
                          Width="82">კლიენტი</TextBlock>
          <ListBox Grid.Row="1" Margin="0,0,0,3"
                ScrollViewer.VerticalScrollBarVisibility="Auto"
                ItemsSource="{Binding}"
                DisplayMemberPath="Name"
      IsSynchronizedWithCurrentItem="True" DataContext="{Binding}" Width="238" />
       <TextBlock Grid.Column="1" HorizontalAlignment="Center" Margin="82,0,76,0"
                                  Width="46">ინკასო</TextBlock>
          <ListBox Grid.Row="1" Margin="0,0,1,3"
                ScrollViewer.VerticalScrollBarVisibility="Auto"
                ItemsSource="{Binding Path=ClientIncasso}"
                DisplayMemberPath="IncassoAmount"
      IsSynchronizedWithCurrentItem="True" DataContext="{Binding}" Width="198"
                                   HorizontalAlignment="Right" Grid.Column="1" />
       <TextBlock Grid.Column="2" HorizontalAlignment="Center" Margin="76,0,87,0"
                                    Width="99">ანგარიში/თანხა</TextBlock>
         <ListBox Grid.Row="1" Grid.Column="2"
                Margin="0,0,0,3"
                ScrollViewer.VerticalScrollBarVisibility="Auto"
               ItemsSource="{Binding Path=ClientIncasso/Incasso_Incasso_Details}"
                DisplayMemberPath="AccAmount"
               />
          </Grid>
       </Grid>
    </Grid>
</Window>
```

აპლიკაციის ლოგიკური ნაწილი რეალიზებულია C#-პროგრამის ტექსტში, რომელიც მე-3 ლისტინგშია ასახული.

//--- **ლისტინგი–3:** Window1.xaml.cs

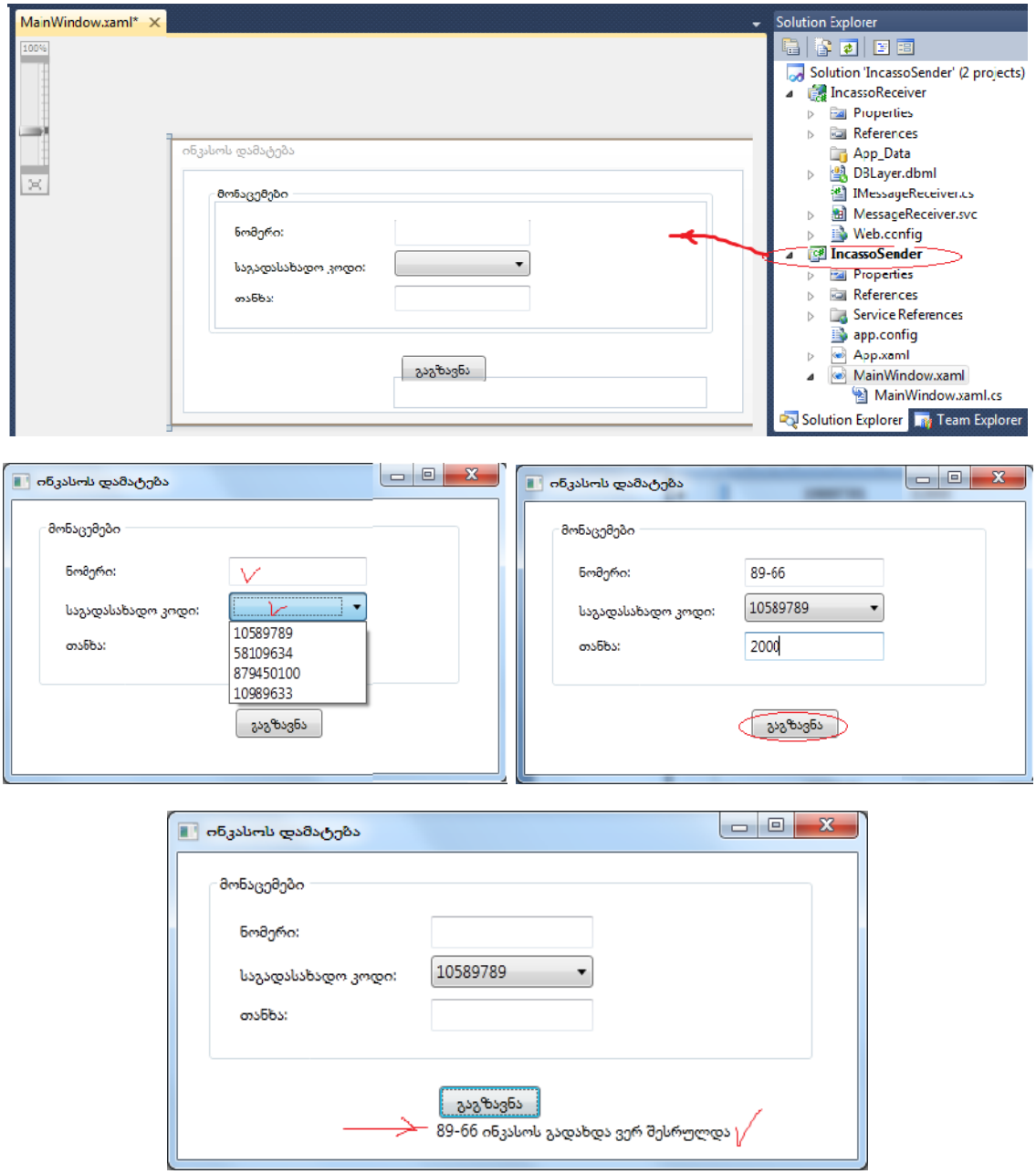
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Data;
using System.Data.SqlClient;

namespace DataBindingRelation
{
  public partial class Window1 : Window
    {
     public Window1() {InitializeComponent(); FillClientIncassoData(); }
        void FillClientIncassoData()
        { // App.config ფაილიდან ამოიღება ბაზასთან მიბმის სტრიქონი ----
          String connectionString = System.Configuration.

ConfigurationManager.ConnectionStrings["Incasso"].ConnectionString;
```

```
    DataSet dataSet = new DataSet(); // გამოყოფილ მონაცემთა შესანახი ადგილის შექმნა
       // გამოყოფილი ადგილის შევსება მონაცემთა ბაზიდან -----
  using (SqlConnection conn = new SqlConnection(connectionString))
    {
      SqlCommand selectCommand = conn.CreateCommand();
      SqlDataAdapter adapter = new SqlDataAdapter(selectCommand);
        // მონაცემები ჩაიტვირთება Clients ცხრილიდან ----
    selectCommand.CommandText = "SELECT FirstName +  ' ' + LastName + '/' +
                                          TaxCode as [Name],  * FROM Clients";
    adapter.Fill(dataSet, "Clients");
        // მონაცემები ჩაიტვირთება Incasso ცხრილიდან ----
    selectCommand.CommandText = "SELECT '#' + IncassoNum + ' / GEL '+
                Cast(Amount as varchar(15)) as IncassoAmount, * FROM Incasso";
    adapter.Fill(dataSet, "Incasso");

      // მონაცემთა ასახვა sql-მოთხოვნით -----
    selectCommand.CommandText = @"select  N'ანგ# '+ A.AccountNum  + ' / GEL '  +
           Cast(IO.Amount as nvarchar(15)) AccAmount, IO.IncassoId FROM
IncassoAccountOperations IO left Join Accounts A on A.AccountId = IO.AccountId";

      adapter.Fill(dataSet, "Incasso_Details");
    }
   // ClientIncasso დამოკიდებულების შექმნა Clients და Incasso ცხრილებს შორის ----
dataSet.Relations.Add("ClientIncasso", dataSet.Tables["Clients"].Columns["Id"],
               dataSet.Tables["Incasso"].Columns["ClientId"]);

 // დამოკიდებულების შექმნა Incasso და Incasso_Details ცხრილებს შორის განსხვავებული ხერხით ----
  DataColumn parentColumn = dataSet.Tables["Incasso"].Columns["Id"];
  DataColumn childColumn = dataSet.Tables["Incasso_Details"].Columns["IncassoId"];
  DataRelation relation = new DataRelation("Incasso_Incasso_Details",
                                                    parentColumn, childColumn);
  dataSet.Relations.Add(relation);
   // Clients-ის მონაცემთა მიმაგრება ინტერფეისთან ----
     ClientIncasso.DataContext = dataSet.Tables["Clients"];
    }
  }
}
```

აპლიკაციის კონფიგურაციის ფაილის ტექსტი ნაჩვენებია მე-4 ლისტინგში:

```
<-- ლისტინგი-4:    App.config ფაილი  ------- -->
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="Incasso" connectionString="Data Source=USER-PC\SQLEXPRESS;
                           Initial Catalog=IncassoDB;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

მე-10 ნახაზზე ნაჩვენებია კლიენტისათვის ახალი ინკასოს დამატების სერვისით შესასრულებელი ინტერფეისის ფრაგმენტები. IncassoSender პროექტში MainWindow.xaml ფაილი ასახავს ფორმის დიზაინს. ინკასოს ნომრის და თანხის მოცულობის შეტანის და საგადასახადო კოდის არჩევის შემდეგ „გაგზავნა“ ღილაკით ინფორმაცია გადაეცემა ბანკს. თუ შესაბამისი კლიენტის ანგარიშზე არაა საკმარისი თანხა, მაშინ ბრუნდება შეტყობინება: „ინკასოს გადახდა ვერ შესრულდა“.



ნახ.10. ინტერფეისი „ინკასოს დამატება“

3. დასკვნა

ობიექტ-ორიენტირებული აპლიკაციების ვებ-სერვისებით ფორმირება განსაკუთრებით მოქნილი, მოსახერხებელი და გაცილებით საიმედოა, როგორც ახალი პროგრამული პროდუქტების შექმნისას, ისე არსებულ სისტემაში ცვლილებების გატარებისას. ვებ-სერვისი წარმოადგენს დამაკავშირებელ რგოლს ობიექტ-ორიენტირებულ და პროცეს-ორიენტირებულ ტექნოლოგიებს შორის, რაც კომპანიათაშორისი და კომპანიის მსხვილ სტრუქტურათაშორისი საქმიანი პროცესების ინტეგრაციასა და მრავალაპლიკაციურ მართვას უზრუნველყოფს.

თანამედროვე პროგრამული ტექნოლოგიების, კერძოდ UML, Agile და Case-ინსტრუმენტების გამოყენებით კომპიუტერული სისტემების მენეჯმენტის და ინტერკორპორაციული ვებ-აპლიკაციების დეველოპმენტის პროცესი მნიშვნელოვნად უმჯობესდება, როგორც ხარისხობრივად, ასევე დაპროექტების, მისი იმპლემენტაციის და რეინჟინერინგის ვადების შემცირებით.

ლიტერატურა:

1. სურგულაძე გ., პეტრიაშვილი ლ. მონაცემთა საცავის აგების ტექნოლოგია ინტერნეტული ბიზნესის სისტემებისათვის. მონოგრ., სტუ. თბ., 2005

2. ბულია ი. თანამედროვე სისტემებში ინტეგრაციის, მონაცემთა გადაცემის და დამუშავების ტექნოლოგიები. სტუ-ს შრ.კრ. „მას“-№2(11), თბილისი, 2011, გვ.139–144

3. Clarke R. Electronic Data Interchange (EDI): An Introduction. http://www.roger clarke.com/EC/EDIIntro.html

4. Booch G., Jacobson I., Rumbaugh J. Unified Modeling Language for Object-Oriented Development. Rational Software Corporation, Santa Clara, 2006

5. სურგულაძე გ., გულიტაშვილი მ., კაკულია ი., ჩერქეზიშვილი გ., ჯავახიშვილი ი. პროგრამული სისტემების სასიცოცხლო ციკლის პროცესის მოდელირება უნივერსალური და ექსტრემალური პროგრამირების პრინციპების კომპრომისული გადაწყვეტით. სტუ-ს შრ.კრ. მას-№1(8), 2008. გვ.63-70

6. Скопин И.Н. Основы менеджмента программных проектов. www.intuit.ru/ department/se/msd. 2004

7. Бек К. Шаблоны реализации корпоративных приложений. Экстремальное программирование: Пер. с англ. М.: Вильяме, 2008

8. თურქია ე. ბიზნეს–პროექტების მართვის ტექნოლოგიური პროცესების ავტომატიზაცია. სტუ. თბ., 2010

9. Rumpe B. Agile Modellierung mit UML. Berlin, „Springer“. 2-te Auflage. 2012

10. თურქია ე., ბულია ი., გიუტაშვილი მ. ინტერკორპორაციული აპლიკაციების ჰორიზონტალური და ვერტიკალური ინტეგრაციის მართვა სერვის-ორიენტირებული არქიტექტურის ბაზაზე. სტუ-ს შრ.კრ. „მას“-№ 1(12). 2012. გვ.57–62.

11. Surguladze G., Turkia E., Topuria N., Lominadze T., Giutashvili M. Towards an Integration of Process-Modeling: from Business Method: from Business-Content to the Software Implementation. IV Intern. Conf. “Problems of Cybernetics and Informatics” (PCI’ 2012). Baku, Azerbaijan, 2012

12. სურგულაძე გ., ბულია ი., ოხანაშვილი მ., ქრისტესიაშვილი ხ. კორპორაციული მენეჯმენტის ბიზნეს-პროცესების მოდელირება და კვლევა ფერადი პეტრის ქსელებით. სტუ-ს შრ.კრ. „მას“-№1(12), თბილისი, 2012, გვ.73–82

13. Jensen K., Kristensen M.L., Wells L. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. University of Aarhus. Denmark. 2007.

MANAGEMENT OF THE INTEGRATION OF INFORMATION SYSTEMS WITH A SERVICE-ORIENTED ARCHITECTURE AND UML/AGILE BASED METHODS

Surguladze Gia, Bulia Irakli, Urushadze Beka
Georgian Technical University

Summary

Problems of integration of corporate information management systems, for their object-, process- and service-oriented analysis, design and implementation are considered. For the management of the life cycle of a software system using methods and tools, based on the UML / Agile, especially on the concept of extreme programming. The experimental part of the research was carried out to solve problems of monitoring the financial transactions of commercial banks and Ministry of Finance for integration their service processes.

МЕНЕДЖМЕНТ ПРОЦЕССОВ ИНТЕГРАЦИИ ИНФОРМАЦИОННЫХ СИСТЕМ С СЕРВИС-ОРИЕНТИРОВАННОЙ АРХИТЕКТУРОЙ И UML/AGILE БАЗИРОВАННЫМИ МЕТОДАМЫ

Сургуладзе Г., Булия И., Урушадзе Б.
Грузинский Технический Университет

Резюме

Рассматриваются проблемы интеграции корпоративных информационных систем управления с целью их объектно-, процесс- и сервис-ориентированного анализа, проектирования и реализации. Для менеджмента жизненного цикла программной системы используются методы, базируемые на UML/Agile, в особенности, концепция экстремального программирования. Экспериментальная часть исследований проводилась для решения задач мониторинга финансовых операций с целью интеграции сервис-процессов коммерческих банков и министерства финансов.