# პროგრამული დანართის არქიტექტურა და RIA-აპლიკაციების დაპროექტება

ნინო კივილაძე
საქართველოს ტექნიკური უნივერსიტეტი

## რეზიუმე

განიხილება RIA-აპლიკაციების არქიტექტურა, მისი კომპონენტები და აგების ფუნდამენტური კონცეფციები. გამოვლენილია არქიტექტურის დაგეგმვის ზოგადი საფეხურები და რეკომენდაციები. წარმოდგენილია პროგრამული დანართის ლოგიკურ დონეებად დაყოფის მიდგომის უპირატესობები და RIA არქიტექტურის სპეციფიკა. მოცემულია არქიტექტურული სტანდარტების ანალიზი Ms Silverlight აპლიკაციებთან მიმართებაში. რეალიზებულია ტრადიციულ და RIA-არქიტექტურათა შედარება.

**საკვანძო სიტყვები:** RIA-აპლიკაცია. შრეების არქიტექტურა. მონაცემთა წვდომის დონე. ბიზნეს-ლოგიკის დონე. პრეზენტაციის დონე. MVVM.. Command-ობიექტი. ასინქრონული გამოძახება.

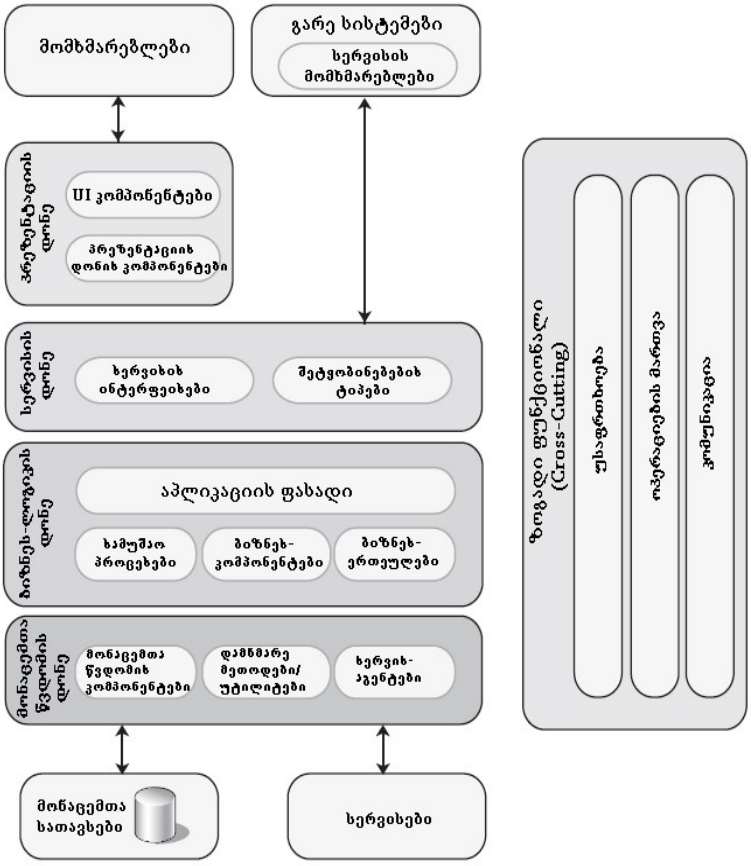
## 1. შესავალი

პროგრამული აპლიკაციის არქიტექტურა არის სტრუქტურული გადაწყვეტილების პროცესით განსაზღვრული პროგრამული სისტემის ორგანიზაციების შედეგი, რომელიც აკმაყოფილებს აპლიკაციის მიმართ წაყენებულ ყველა ტექნიკურ და ფუნქციურ მოთხოვნილებას [1]. ამასთან ერთად მიღწეულია მაღალი ხარისხობრივი მახასიათებლები, როგორიცაა უსაფრთხოება, მართვადობა და სისწრაფე. ყოველ მიღებულ გადაწყვეტილებას განაპირობებს მრავალი გარე ფაქტორი და თითოეულ მათგანს ზეგავლენა აქვს პროგრამული დანართის წარმატებაზე. სხვა ნებისმიერი კომპლექსური სტრუქტურის მსგავსად, პროგრამული დანართი უნდა აიგოს მყარ საფუძველზე. ნაშრომში განიხილება RIA-აპლიკაციების არქიტექტურა და მისი ფუნდამენტური კონცეფიები, ყურადღება გამახვილებულია აპლიკაციაში ძირითადი ელემენტებისა და კომპონენტების გამოყენებასა და მათ ურთიერთკავშირზე. პროგრამული დანართის არქიტექტურის პრინციპების გათვალისწინებით გამოვლენილია RIA-არქიტექტურის შემუშავების მეთოდოლოგია და მოდელები. ბოლოს კი მოყვანილია Command-არქიტექტურის რეალიზაცია Silverlight-აპლიკაციაში.

## 2. პროგრამული დანართის არქიტექტურა

პროგრამული დანართის არქიტექტურას ხშირად განმარტავენ, როგორც სისტემის ორგანიზების პროცესს, სადაც სისტემა სპეციფიური ფუნქციის შემცველი კომპონენტების კოლექციაა. სხვაგვარად რომ ვთქვათ, არქიტექტურა ფოკუსირდება კომპონენტების ორგანიზებაზე, რათა მათ იმპლემენტაცია გაუკეთოს სპეციფიკურ ლოგიკურ ფუნქციას [2]. ფუნქციის ორგანიზებას ხშირად უწოდებენ კომპონენტების დაჯგუფებას ინტერესის არეალში (Areas of Concern). 1-ელ ნახაზზე ნაჩვენებია აპლიკაციის ზოგადი არქიტექტურა, სადაც კომპონენტები გადანაწილებულია სხვადასხვა ლოგიკურ არეალში. გარდა კომპონენტების დაჯგუფებისა, გამოყოფილია ,,ზოგადი ფუნქციონალის არეალი", რომელიც ფოკუსირდება კომპონენტების ურთიერთკავშირზე და ერთობლივ მუშაობაზე. პროგრამული დანართის ლოგიკურ დონეებზე გადანაწილება გულისხმობს ერთმანეთთან დაკავშირებული ფუნქციის ცალკეულ ლოგიკურ შრეებში (Layer) გაერთიანებას. ეს შრეები ერთიმეორეზე ვერტიკალურად არის დალაგებული. თითოეული შრის ფუნქციონალი ატარებს რაიმე ერთ კონკრეტულ დანიშნულებას და შრეებს შორის მიმდინარეობს ინფორმაციის მიმოცვლა, კომუნიკაცია. პროგრამული დანართის გადანაწილება ლოგიკურ შრეებში თავიდან გვარიდებს კონცეფციების მჭიდრო გადაჯაჭვულობას და შედარებით იოლი სამართავია.

ლოგიკური შრეების არქიტექტურა მუშაობის პრინციპით წააგავს ამობრუნებულ პირამიდას: ყოველი შრე უკავშირდება და იყენებს მის უშუალოდ ერთი დონით ქვემოთ მდგომი შრის ფუნქციონალს.




**ნახ.1. ერთგვერდიანი აპლიკაცის (Single Page Application - SPA) მუშაობის სქემა**

აპლიკაციის ლოგიკური შრეები შეიძლება იმყოფებოდეს ერთსა და იმავე ფიზიკურ კომპიუტერზე ან გადანაწილებული იყოს რამდენიმე კომპიუტერზე (N-Tier). N-Tier განაწილების შემთხვევაში შრეებს შორის კომუნიკაცია ხორციელდება მკაფიოდ განსაზღვრული ინტერფეისით. ტიპიური ვებ-აპლიკაციის არქიტექტურული დიზაინი შედგება: პრეზენტაციის, ბიზნესლოგიკის და მონაცემთა სათავსოსთან წვდომის (Data Layer) დონეებისგან [1].

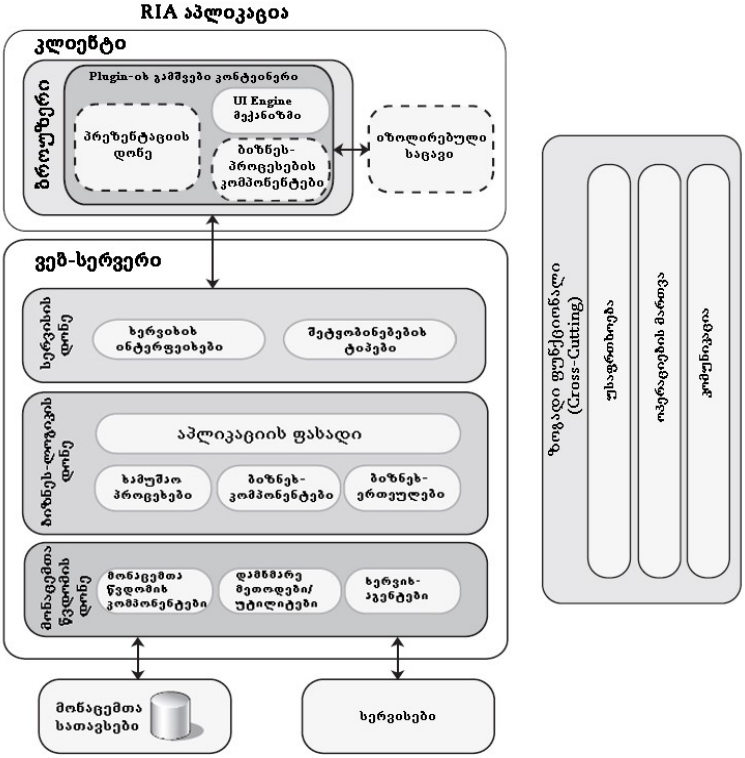
- **პრეზენტაციის დონე** (Presentation Layer). ეს დონე შედგება მომხმარებელზე ორიენტირებული ფუნქციისგან, რომელიც პასუხისმგებელია მომხმარებლსა და სისტემას შორის ურთიერთქმედებაზე. პრეზენტაციის დონე ხიდის როლს თამაშობს და მომხმარებელს აწვდის ბიზნეს-ლოგიკის დონეზე აღწერილ ძირითად ფუნქციონალს.
- **ბიზნეს-ლოგიკის დონე** (Business Layer). ამ დონეზე იმპლემენტირებულია სისტემის ძირითადი ფუნქცია და ინკაფსულირებულია შესაბამისი ლოგიკა. ზოგადად, ბიზნეს ლოგიკის დონე შედგება კომპონენტებისგან, რომლებიც იძლევა სერვისის ინტერფეისებს, რათა სხვა გამომძახებელმა სისტემებმა შეძლოს მათი გამოყენება.
- **მონაცემთა წვდომის დონე** (Data Layer). ამ დონეზე უზრუნველყოფილია სისტემის ფარგლებში დაჰოსტილ მონაცმთა სათავსოებზე და სხვა სისტემების მიერ მოწოდებულ მონაცემებზე წვდომა (შესაძლოა ეს წვდომა სერვისების გავლით ხდებოდეს). მონაცემთა წვდომის დონე იძლევა Generic ტიპის ინტერფეისებს, რომლებსაც მოიხმარს ბიზნესლოგიკის დონის კომპონენტები.

### 3. RIA აპლიკაციების დაპროექტება

ტერმინი RIA-აპლიკაცია ითარგმნება როგორც „მდიდარი" ინტერნეტ აპლიკაცია ("Rich Internet Application"). განვიხილოთ RIA-აპლიკაციების ძირითადი დანიშნულება, გავაანალიზოთ მისი კომპონენტები და RIA არქიტექტურის შემუშავების ძირითადი რეკომენდაციები [4].

RIA აპლიკაცია უზრუნველყოფს მდიდარ გრაფიკულ სამომხმარებლო ინტერფეისს და Streaming Media სერვისებს, ამავდროულად, გააჩნია ვებ-აპლიკააცისთვის დამახასიათებელი თვისებები: ვებ-სერვერზე განთავსების სიიოლე და შემდგომი მხარდაჭერა. RIA-აპლიკაციები ეშვება ბრაუზერის Plug-In გარემოში, რითაც განსხვავდება სხვა ვებ-ტექნოლოგიებისგან, რომლებიც იყენებს ბრაუზერის კოდს (მაგალითად, AJAX ტექნოლოგია). ტერმინი Plug-In აღწერს კომპიუტერულ პროგრამას, რომელიც ხელს უწყობს სხვა პროგრამული დანართების მუშაობის გაუმჯობესებას ან ფუნქციის გამდიდრებას. ტიპური RIA-იმპლემენტაცია ვებ-ინფრასტრუქტურას იყენებს კლიენტის-მხარის აპლიკაციასთან ერთობლიობაში, რომელიც პრეზენტაციის დონეს უზრუნველყოფს. ბრაუზერის Plug-In დანამატი

იძლევა მდიდარ გრაფიკულ ინტერფეისთან სამუშაო ბიბლიოთეკებს და კონტეინერის როლს თამაშობს, რათა უსაფრთხოების დაცვის მიზნით, ლოკალურ რესურსებზე შეზღუდოს წვდომა. RIA-აპლიკაციებს ახასიათებს უფრო კომპლექსური და მდიდარი კლიენტი მხარის (Client-Side) კოდის მხარდაჭერა, ვიდრე შესაძლებელი იყო ტრადიციული ვებ-აპლიკაციებისთვის, რის შედეგადაც ვებ-სერვერზე დატვირთვა მსუბუქდება და ნაწილდება კლიენტის მანქანაზე. მე-2 ნახაზზე ნაჩვენებია ტიპიური RIA-იმპლემენტაციის სტრუქტურა.




ნახ.2. RIA აპლიკაციის არქიტექტურა

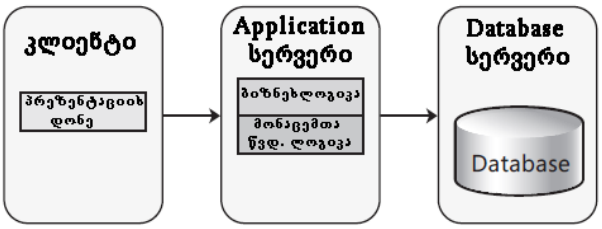
ტიპური RIA-აპლიკაცია დანაწილებულია 3 ლოგიკურ დონეზე: პრეზენტაციის, ბიზნეს-ლოგიკის და მონაცემთა წვდომის. პრეზენტაციის დონე შეიცავს სამომხმარებლო ინტერფეისს (UI) და პრეზენტაციის ლოგიკის კომპონენტებს; ბიზნეს-ლოგიკის დონე შეიცავს ბიზნეს-ლოგიკას, სამუშაო პროცესებს და ბიზნეს-ერთეულებს; მონაცემთა წვდომის დონე შეიცავს მონაცემთა წვდომის და სერვის-აგენტ კომპონენტებს.

RIA-აპლიკაციის დაპროექტების დროს გასათვალისწინებელია ზოგადი მიდგომები, რომელიც შემუშავებულია RIA კომპონენტებთან მიმართებაში.

განვიხილოთ არქიტექტურული რეკომენდაციები თითოეული მნიშვნელოვანი არეალისთვის [2].

- **ბიზნესლოგიკის დონე.** უმრავლეს სცენარებში RIA-აპლიკაციები უკავშირდება მონეცემებს. მონაცემთა სათავსოები აპლიკაციის გარეთაა. მიუხედავად იმისა, რომ ინფორმაციის ნატურა განსხვავდება, მისი მიღება ხშირად ბიზნესსისტემიდან ხორციელდება. უსაფრთხოების მიზნით არ უნდა დავუშვათ სენსიტური ბიზნესლოგიკის დაუშიფრავად განთავსება კლიენტის მხარეს. XAP ფაილები გადმოწერადია და კოდის დეკომპილაციის პროცესი საკმაოდ იოლია. ამიტომ, რეკომენდებულია ასეთი ინფორმაციის იმპლემენტირება სერვერის მხარეს და მასზე წვდომის დაშვება ვებ-სერვისების გავლით;
- **კომუნიკაცია.** RIA იმპლემენტაციებში რეკომენდებულია სერვისებზე ასინქრონული გამოძახების მოდელის გამოყენება, რათა თავიდან ავიცილოთ ბრაუზერის პროცესებზე ბლოკის დადება. კროს-დომეინ, პროტოკოლისა და სერვისების ეფექტურად მუშაობის საკითხები არქიტექტურის ნაწილი და განსახილველი საკითხია. თუკი RIA-იმპლემენტაცია ამის საშუალებას იძლევა, რეკომენდებულია სხვადასხვა ოპერაციისთვის განსხვავებული Thread გარემოს გამოყენება;
- **პრეზენტაციის დონე.** RIA აპლიკაციები ეშვება ბრაუზერში და ამრიგად, შემოიფარგლება ბრაუზერის შესაძლებლობებით. ამიტომ მათი მუშაობა ოპტიმალურია ერთიანი, საერთო ცენტრალური ინტერფეისის არსებობის პირობებში. მრავლი გვერდის შემცველი პროგრამული დანართები დამატებითი სამუშაოების ჩატარებას საჭიროებს, რათა დაიგეგმოს კავშირი გვერდებს შორის;
- **ვალიდაცია** - შესაძლებელია კლიენტის მხარის კოდის ან სერვერზე განთავსებული სერვისების საშუალებით. თუ კლიენტის მხარეს ტრივიალურ ვალიდაციებზე მეტი ფუნქციონალი გვესაჭიროება, რეკომენდებულია ვალიდაციის ლოგიკის იზოლირება დამოუკიდებელ, ჩამოტვირთვად კრებულში (Assembly). ეს აიოლებს ბიზნესწესების მართვას.

- **პროგრამული დანართის სერვერზე განთავსება** (Deployment). რადგან RIA-აპლიკაციებს გადააქვს პრეზენტაციის ლოგიკის ასლი კლიენტის მანქანაზე, პროგრამული უზრუნველყოფის განთავსების დროს ყველაზე მისაღები სცენარია განაწილებული არქიტექტურა. განაწილებულ სერვერებზე განთავსების დროს პრეზენტაციის ლოგიკა კლიენტის მანქანაზე იმყოფება; ბიზნეს-ლოგიკა შესაძლოა იმყოფებოდეს კლიენტის მანქანაზე, სერვერზე ან გაზიარებული იყოს სერვერზეც და კლიენტზეც. მონაცემთა წვდომის დონე იმყოფება ვებ-სერვერზე ან Application-სერვერზე. ზოგადად, ბიზნესლოგიკის ნაწილი გადაგვაქვს კლიენტის მხარეს, რითაც პროგრამული დანართის წარმადობა და სისწრაფე იზრდება. ამ შემთხვევაში ბიზნესლოგიკისა და მონაცემთა წვდომის დონეები გავრცობილია კლიენტის მანქანასა და Application-სერვერზე, როგორც მე-3 ნახაზზეა ილუსტრირებული.




**ნახ.3. RIA აპლიკაციის განთავსება განაწილებულ სერვერებზე**

## 4. RIA აპლიკაციების შესაბამისი არქიტექტურული სტანდარტები

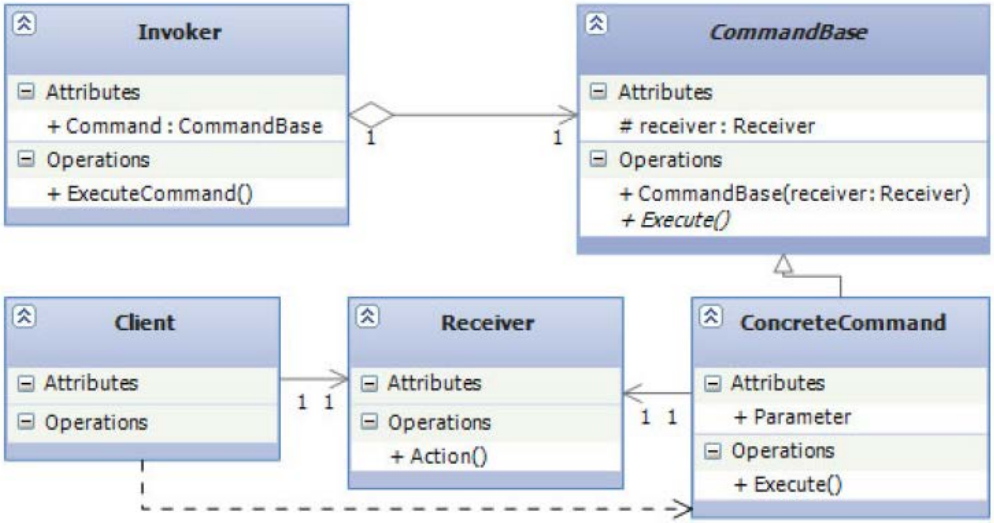
ძირითადი სტანდარტები (Design Pattern) ორგანიზებულია კატეგორიებად სექციებში: შრეები (Layers), კომუნიკაცია (Communication), კომპოზიცია (Composition), პრეზენტაცია (Prezentation). ამ სტანდარტების გათვალისწინებით ვხელმძღვანელობთ არქიტექტურული გადაწყვეტილებების არჩევისას (1-ელი ცხრილი).

**RIA-აპლიკაციებში გამოყენებული არქიტექტურული ტანდარტები** **ცხრ.1**

| კატეგორია | შესაბამისი სტანდარტი (Design Pattern) |
|---|---|
| შრეები (Layers) | **Service Layer.** არქიტექტურული სტანდარტი, რომელშიც სერვისის ინტერფეისი და იმპლემენტაცია გაერთიანებულია ერთ ლოგიკურ შრეზე(Layer). სერვისი იმპლემენტირებულია WCF ტექნოლოგიით. |
| კომუნიკაცია (Communication) | **Asynchronous Callback.** გრძელვადიანი სამუშაოების განცალკევებულ Background Thread ფონზე გაშვება და Callback ფუნქციით შესრულებული სამუშაოს რეზულტატების ამოკითხვა.<br>**Command.** მოთხოვნის დამუშავების ლოგიკის ინკაფსულაცია Command ობიექტში, რომელიც იძლევა Common Execution Interface გარემოს. |
| კომპოზიცია (Composition) | **Composite View.** ინდივიდუალური View გვერდების გაეთიანება კომპოზიტურ View-ში.<br>**Inversion of Control**. ობიექტების სხვა ობიექტებზე/კომპონენტებზე დამოკიდებულებების აღწერა, რომელთა იმპლემენტაცია სავალდებულოა, რათა მოცემული ობიექტის გამოყენება შესაძლებელი გახდეს პროგრამულ დანართში. |
| პრეზენტაცია (Prezentation) | **Application Controller.** ობიექტი, რომელიც შეიცავს სრული სამუშაო პროცესის ლოგიკას და გამოიყენება სხვა კომპონენტების მიერ, რომლებიც მუშაობენ Model ობიექტთან და ასახავენ შესაბამის View პრეზენტაციას.<br>**Supervising Presenter**. პრეზენტაციის დონის სტრუქტურის გადანაწილება სამ დამოუკიდებელ როლში: View პასუხისმგებელია მომხმარებლთან ურთიერთ-კავშირზე (User Input); მონაცემთა ბმის მექანიზმით გამოაქვს Model კომპონენტის ინფორმაცია, ხოლო Model კომპონენტში ინკაფსულირებულია ბიზნეს-ლოგიკა; Presenter ობიექტში იმპლემენტირებულია პრეზენტაციის ლოგიკა.<br>**Presentation Model.** წარმოადგენს Model-View-Presenter(MVP) არქიტექტურული სტანდარტის ვარიაციას და შექმნილია თანამედროვე სამომხმარებლო ინტერფეისების დეველოპმენტ-პლატფორმებისთვის, სადაც View გრაფიკული დიზაინერის დანიშნულებას უფრო ატარებს, ვიდრე პროგრამული ლოგიკის. |

## 5. არქიტექტურული სტანდარტი - Command

Command - არქიტექტურული სტანდარტი შემომავალ მოთხოვნებზე საპასუხო ლოგიკას აერთიანებს ერთ ობიექტში. Command ობიექტის გამოძახება მოთხოვნის შემოსვლისთანავე ხდება. ამ არქიტექტურული მიდგომის მთავარი დამახასიათებელია ამა თუ იმ პროგრამული ლოგიკის შესასრულებლად საჭირო ინფორმაციის თავმოყრა ერთ ობიექტში. თავად ობიექტი არ ასრულებს რაიმე ოპერაციას, იგი მხოლოდ ინფორმაციას შეიცავს [5]. მე-4 ნახაზზე მოყვანილი დიაგრამა აღწერს Command სტანდარტის რეალიზებას. დიაგრამა შედგება ხუთი კლასისგან:




**ნახ.4. Command არიტექტურული სტანდარტის იმპლემენტაცია**

- **კლიენტი** (Client) კლასი – Command არქიტექტურის მომხმარებელია. იგი ქმნის Command ობიექტს და აკავშირებს მიმღებთან (Receiver);
- **მიმღები** (Receiver) კლასი – იცის თუ როგორ უნდა განახორციელოს შემომავალ მოთხოვნასთან დაკავშირებული ოპერაციები;
- **CommandBase** აბსტრაქტული კლასი – (ან ინტერფეისი) ყველა Command ობიექტისთვის. იგი შეიცავს ინფორმაციას თუ რომელი მიმღები (Receiver) არის პასუხისმგებელი Command ობიექტში ინკაპსულირებული ოპერაციების შესრულებაზე;
- **კონკრეტული იმპლემენტაცია** (ConcreteCommand) – არის CommandBase აბსტრაქტული კლასის (ან ინტერფეისის) კონკრეტული იმპლემენტაცია.
- **გამომძახებელი** (Invoker) – ობიექტია, რომელიც წყვეტს თუ როდის უნდა გაეშვას Command ობიექტი.

## 6. დასკვნა

RIA-არქიტექტურა ინტერაქტიული გრაფიკული სამომხმარებლო ინტერფეისის მქონე Web-აპლიკაციების შესამუშავებელად საჭირო პრინციპების ნაკრებია. იგი შემუშავებულია ვებ-საიტებისა და დესკტოპ-აპლიკაციების საუკეთესო პრაქტიკების საფუძველზე. RIA-აპლიკაციებში პროცესები სრულდება ასინქრონულად და მომხმარებელს არ უწევს ლოდინი ვებ-გვერდთან მუშაობისას. გარდა ამისა, ბიზნესლოგიკის ნაწილი სრულდება კლიენტის მხარეს, რაც ამსუბუქებს სერვერზე დატვირთვას და ზრდის პროგრამული დანართის შესრულების სისწრაფეს. RIA არქიტექტურის გამოყენება კარგ შედეგს იძლევა 1-ელ ცხრილში მოცემული არქიტექტურული სტანდარტების საფუძველზე. Command არქიტექტურული სტანდარტის გამოყენებით მოთხოვნებზე საპასუხო ლოგიკის ინკაფსულირება ხდება ერთ დამოუკიდებელ ობიექტში.

**ლიტერატურა –References – Литература:**

1. Esposito D., Saltarello A. (2014). Architecting Applications for the Enterprise. ISBN: 978-0-7356-8535-2. Copyright © 2014.
2. Gary McLean Hall. (2010). Pro WPF and Silverlight MVVM, Effective Application Development with Model-View-ViewModel. ISBN-13 (pbk): 978-1-4302-3162-2 Copyright © 2010.
3. Freeman E., Bates B., Sierra K., Robson E. (2004). Head First Design Patterns. ISBN: 0596007124 Publisher: O'Reilly Media, Inc. Copyright © 2004.
4. Lair R. (2010). Beginning Silverlight 4 in C# ISBN-13: 978-1-4302-2988-9, Copyright © 2010.
5. http://www.go4expert.com/articles/design-pattern-simple-examples-t5127/ გადამოწმ. 17.04.16
6. http://www.dofactory.com/net/design-patterns. გადამოწმ. 17.04.16.

## WEB APPLICATION ARCHITECTURE. DESIGNING RIA APPLICATIONS

Nino Kiviladze

Georgian Technical University

**Summary**

This article gives an information about designing the RIA implementations and its fundamental concepts. It provides the general stages of software architecture planning and recommendations for each layer. This paper makes comparison between standard web application architecture and RIA application implementations. We discuss benefits of designing RIA-applications in combination with software design patterns, such as: Command object, Asynchronous Callback, Service Layer, Composite View, Model-View-Presenter Model. At the end of the article, we provide an UML diagram, showing an implementation of command design pattern used in RIA-applications.

## АРХИТЕКТУРА ПРОГРАММНОГО ПРИЛОЖЕНИЯ И ПРОЕКТИРОВАНИЕ RIA-АППЛИКАЦИЙ

Кивиладзе Нино

Грузинский Технический Университет

**Резюме**

Рассматриваются архитектура RIA аппликаций и ее фундаментальные концепции. Выявлены общие шаги планирования и рекомендации. Представлены преимущества подхода деления на логические уровни программного приложения и специфика RIA архитектуры. Предлагаются результаты анализа архитектурных стандартов по отношению к аппликациям Silverlightи и приводится сравнение между традиционными и RIA архитектурами.