Univerzitet u Nišu
Elektronski fakultet

Petar Rajković

Unapređenje procesa razvoja i održavanja informacionih sistema primenom domenskih modela podataka

Doktorska disertacija

Niš 2015.

University of Niš
Faculty of Electronic Engineering

Petar Rajković

Using Domain Models for Improving Information System Development and Maintenance

Doctoral Dissertation

Niš 2015.

AUTOR	
Ime i prezime	Petar Rajković
Datum rođenja	5.4.1978.

DOKTORSKA DISERTACIJA	
Naziv	Unapređenje procesa razvoja i održavanja informacionih sistema primenom domenskih modela podataka
Broj strana	169 + xiv
Broj slika	123
Broj tabela	10
Broj bibliografskih jedinica	147
Ustanova gde je rađena disertacija	Univerzitet u Nišu, Elektronski fakultet
Mentor	dr Dragan Janković, redovni profesor
Datum prijave teme doktorske disertacije	17.12.2013.
Broj odluke i datum prihvatanja teme doktorske disertacije	Elektronski fakultet: 07/03-061/14-001 od 18.12.2014. Univerzitet u Nišu: 8/20-01-001/15-009 od 19.1.2015.
Komisija za ocenu podobnosti teme	1. Prof. dr Dragan Janković, redovni profesor, Univerzitet u Nišu, Elektronski fakultet u Nišu, mentor 2. Prof. dr Milena Stanković, redovni profesor, Univerzitet u Nišu, Elektronski fakultet u Nišu 3. Prof. dr Milorad Tošić, redovni profesor, Univerzitet u Nišu, Elektronski fakultet u Nišu 4. Prof. dr Dejan Rančić, redovni profesor, Univerzitet u Nišu, Elektronski fakultet u Nišu 5. Doc.dr. Dejan Aleksić, Univerzitet u Nišu, Prirodno-matematički fakultet
Komisija za odbranu disertacije	1. Prof. dr Dragan Janković, redovni profesor, Univerzitet u Nišu, Elektronski fakultet u Nišu, mentor 2. Prof. dr Milena Stanković, redovni profesor, Univerzitet u Nišu, Elektronski fakultet u Nišu 3. Prof. dr Milorad Tošić, redovni profesor, Univerzitet u Nišu, Elektronski fakultet u Nišu 4. Prof. dr Dejan Rančić, vanredni profesor, Univerzitet u Nišu, Elektronski fakultet u Nišu 5. Doc.dr. Dejan Aleksić, Univerzitet u Nišu, Prirodno-matematički fakultet
UDK	004.41; 004.414.32; 004.416.3
Datum odbrane	

AUTHOR	
Name and surname	Petar Rajković
Date of birth	5.4.1978.

DOCTORAL DISSERTATION	
Title	Using Domain Models for Improving Information System Development and Maintenance
Number of pages	169 + xiv
Number of figures	123
Number of tables	10
Number of references	147
Institution	University of Niš, Faculty of Electronic Engineering
Mentor	dr Dragan Janković, full professor
Date of doctoral dissertation submit	17.12.2013.
Number of decision and the date of acceptance of the doctoral dissertation theme	Faculty of Electronic Engineering: 07/03-061/14-001,18.12.2014. University of Niš: 8/20-01-001/15-009, 19.1.2015.
Commission for the assessment of the doctoral dissertation theme	1. Prof. dr Dragan Janković, full professor, University of Niš, Faculty of Electronic Engineering, mentor 2. Prof. dr Milena Stanković, full professor, University of Niš, Faculty of Electronic Engineering 3. Prof. dr Milorad Tošić, full professor, University of Niš, Faculty of Electronic Engineering 4. Prof. dr Dejan Rančić, full professor, University of Niš, Faculty of Electronic Engineering 5. Doc.dr. Dejan Aleksić, University of Niš, Faculty of Sciences and Mathematics
Commission for the dissertation defence	1. Prof. dr Dragan Janković, full professor, University of Niš, Faculty of Electronic Engineering, mentor 2. Prof. dr Milena Stanković, full professor, University of Niš, Faculty of Electronic Engineering 3. Prof. dr Milorad Tošić, full professor, University of Niš, Faculty of Electronic Engineering 4. Prof. dr Dejan Rančić, associate professor, University of Niš, Faculty of Electronic Engineering 5. Doc.dr. Dejan Aleksić, University of Niš, Faculty of Sciences and Mathematics
UDC	004.41; 004.414.32; 004.416.3
Date of defence	

Naučni doprinos doktorske disertacije

1. Kreiranje softverskog alata za generisanje specifičnih domenskih modela podataka. Alat odlikuje visoki nivo mogućnosti konfigurisanja i na jednostavan način može menjati osnovni meta model i izgled.
2. Definisanje domenskog modela podataka za upotrebu u medicinskim informacionim sistemima pomoću kreiranog alata a na bazi proširenog OpenEHR meta-modela.
3. Kreiranje pravila i procedura za identifikovanje tačaka proširenja u informacionim sistemima. Kao podrška ovom procesu implementiran je alat za generisanje inicijalne administratorske aplikacije koja služi za ispitivanje potencijalnih tačaka proširenja.
4. Kreiranje procedure za izdvajanje, testiranje i verifikovanje šablonskih komponenti.
5. Kreiranje softverskog alata za generisanje softverskih komponenti na osnovu modela podataka i kreirane šablonske komponente.
6. Kreiranje procedura za generisanje dinamičkih (runtime) komponenti koje interpretiraju modele podataka.
7. Definisanje posebnog pristupa za izbor adekvatne metode razvoja u zavisnosti od segmenta životnog ciklusa informacionog sistema i vrste softverskog modula koji se razvija.
8. Definisanje kombinovane metodologije, bazirane delom na tradicionalnim a delom na agilnim metodama razvoja, za primenu u procesu dogradnje informacionih sistema.
9. Definisanje mesta metode razvoja bazirane na modelu (Model Driven Engineering) u procesima razvoja, implementacije i dogradnje informacionih sistema.

Scientific Contributions of the Doctoral Dissertation

1. Defining highly customized data modeling tool used for domain model creation. The tool is highly adaptable and able to easily switch both underlying meta model and a view.
2. Defining specific domain model intended for use within medical information systems using mentioned tool and based on extended OpenEHR model.
3. Defining rules for detection of extension points in existing information systems. As a test bed specific database administering application is used. Database administering application is generated using specifically developed application.
4. Defining procedures for extracting template components and their testing and verifying.
5. Developing software tool for software component generation based on a domain model and template component.
6. Defining procedure for generating runtime software components that interpret domain models.
7. Defining specific approach for choosing a development method on the base of information system's lifecycle stage and a type of a developing software module
8. Defining combined methodology, partly based on traditional and partly on agile methodologies, for use in a process of information system upgrade process.
9. Positioning model driven engineering methodology in processes of development, deployment and upgrade of information systems.

Sažetak

Procesi razvoja, dogradnje i održavanja informacionih sistema uključuju mnoge podoblasti računarstva i u isto vreme zahteva specifično domensko znanje koje je potrebno ugraditi kako bi sistem bio upotrebljiv krajnjem korisniku. Uzevši sve nivoe složenosti u obzir, vreme potrebno za kreiranje informacionog sistema, u mnogim slučajevima, je duže nego što bi potencijalni korisnici to želeli. Sa druge strane, vrlo često, informacioni sistemi obiluju komponentama koje dele gotovo isti skup funkcionalnosti, a koje su bazirane na drugačijim skupovima podataka. Implementacija i testiranje takvih funkcionalnosti odnosi mnogo više vremena nego što bi realno bilo neophodno.

U ovoj disertaciji će biti predložen način tj. proces za rešavanje pomenutih problema, koji u osnovi koristi domenske modele podataka i razvoj baziran na modelima. Kao podrška navedenoj paradigmi razvijen je skup specifično dizajniranih softverskih alata i predložen način upotrebe hibridnih metodologija razvoja u cilju unapređenja svih bitnih segmenata u životnom ciklusu informacionih sistema.

Kao početni deo procesa prikazano je definisanje meta i domenskih modela podataka na osnovu postojećih standardnih modela. Kao osnova razvoja uzeti su openEHR i ISA-95 meta modeli. OpenEHR je značajni predstavnik kategorije meta modela okrenutih ka proširenju strukture, a ISA-95 je fokusiran na proširenje kroz modelovanje akcija. Uz proces modelovanja, predstavljen je i softverski alat za modelovanje kreiran tako da se jednostavno može prilagoditi drugim meta modelima. Dat je i primer adaptacije prikazanog alata za efikasniju primenu u PIMS sistemima baziranim na ISA-95 i B2MML-u.

Alati za modelovanje su u osnovi jednosmerne komponente koje služe da kreiraju domenski model i omoguće njegovo preslikavanje na ostatak sistema, pa kao takvi nisu potpuno pogodni za primenu u celokupnom životnom ciklusu informacionih sistema. Zbog toga ih treba proširiti dodatnim alatima koji će omogućiti njihovu širu primenu, kojima je posvećeno posebno poglavlje. U okviru ovog poglavlja prikazan je modul za reverzni inženjering koji služi da na osnovu postojećih struktura podataka ažurira model. Takođe, uz ovaj modul prikazana je i biblioteka za analizu strukture baze podataka kao i alat za generisanje aplikacija za administraciju baze. Biblioteka za analizu strukture baze ima primenu kod detektovanja određenih anomalija u bazi, kao i pri izdvajanju tačaka proširenja. Proces izdvajanja tačaka proširenja je prikazan na primeru već postojeće standardne B2MML baze. Na kraju poglavlja o dodatnim alatima je dat opis šablonskih komponenata i prikazan proces njihovog izdvajanja.

Na osnovu domenskih modela podataka generisanih kroz alat za modelovanje i njegovih pratećih komponenti, alat za generisanje softverskih komponenti generiše nove elemente sistema. Alat za generisanje je baziran na šablonskim komponentama i predstavlja još jedan doprinos ove disertacije. Dizajniran je tako da učita šablonsku komponentu, model podataka, generatorsku klasu i na osnovu njih kreira realnu komponentu. Kako su svi delovi ovog sistema zamenljivi nezavisno jedan od drugog, praktično je moguće generisati bilo kakve komponente u bilo kojoj tehnologiji, ali je sistem ipak optimizovan za .NET. U disertaciji je dat opis postupka generisanja nekoliko različitih klasa komponenti – Windows formi, komponenti za selektovanje vrednosti, resursa koji sadrže prevode kao i konfiguracija sa listama privilegija. Takođe, dat je opis procesa validacije generisanih komponenti kao i procesa proširenja aplikacije generisanim komponentama.

Sem pristupa generisanja koda na osnovu modela, u disertaciji je analizirana i mogućnost primene pristupa interpretacije modela u toku izvršenja programa. Za to je razvijen poseban skup Web komponenti koji je inicijalno prilagođen generisanju različitih izveštaja. S obzirom da je i ovde primenjen pristup sa nezavisnim interpretatorskim klasama (slično kao sa nezavisnim generatorskim klasama kod alata za generisanje) moguće je relativno jednostavno proširenje sistema kako bi podržao i druge klase komponenti.

Poslednje poglavlje u disertaciji se odnosi na procedure i smernice za razvoj i održavanje informacionih sistema. Dat je osvrt na različite metodologije razvoja sistema, kao i skup preporuka kako ih kombinovati u različitim životnim fazama informacionog sistema. Posebno su izdvojene preporuke za fazu razvoja arhitekture i centralnog dela sistema i paralelno sa tim smernice za rad na korisničkom interfejsu. Uz ceo ovaj proces dat je i primer definisanja arhitekture novog sistema na osnovu korisničkih zahteva kao i opšteg okruženja u kome sistem treba da radi. Smernice vezane za proces razvoja prati i skup preporuka za instaliranje i dogradnju informacionih sistema. Iako se ovi procesi čine jednostavnim, na osnovu iskustva iz prakse biće prikazani glavni problemi kao i načini njihovog prevazilaženja koji su dali dobre rezultate.

Svi prikazani softverski alati i smernice za razvoj predstavljaju sublimaciju dvanaestogodišnjeg iskustva u radu na razvoju više različitih tipova informacionih sistema. Kao važni elementi za ceo proces jasno su izdvojeni i glavni uslovi prihvatanja razvijenih sistema uz potenciranje uloge krajnjeg korisnika u celokupnom životnom ciklusu softvera. Svi elementi opisani u ovoj disertaciji čine jedan okvir za razvoj i održavanje informacionih sistema baziran na modelu podataka koji ima glavni cilj da olakša posao tehničkom osoblju kroz sve faze u životu softvera, i u isto vreme doprinese kreiranju sistema koji će krajnji korisnici mnogo lakše prihvatiti.

Summary

All major parts of information systems life cycle depend on a knowledge that come from many areas of computer sciences. In the same time, it requires domain specific knowledge that should be incorporated in order to make develop software useful to its end users. Considering all levels of complexity, time needed for a system development is, pretty often, much longer than it is really necessary. Also, this period can be unacceptable for potential. On the other hand, information system contains many components that share the same set of basic functionalities, but display different data. Processes of implementation and testing in mentioned cases can last much longer than expected.

This dissertation is focused on a process of solving mentioned problem having domain driven development as a basic paradigm. As a main contribution the dissertation presents a set of specially designed software tools as well as a hybrid methodology tending to define best practice guidelines for domain driven engineering. The final goal is to create a framework that should improve processes of development, deployment and maintenance of information systems.

Defining meta and domain models is identified as an initial part of the model driven engineering. New models are created on the base of industrial standards such are openEHR and ISA-95 meta models. OpenEHR come from medical informatics (OpenEHR stands for Open Electronic Health Record) and it is an example of the model which structure can be easily extended. On the other side ISA-95 is a standard intended to be used for manufacturing process modeling, and it mainly supports model extension through action modeling. Modeling process is supported by a general purpose and easy-adaptable modeling tool presented in this dissertation.

Modeling tool is just a base piece of software used for developing and maintaining different software components. To complete the set of needed developing components reverse engineering tool is next tool that is presented. Its main aim is to analyze existing data structures and automatically expand the model. Along with reverse engineering tool library able to examine database and detect different anomalies is presented. At the end of the chapter describing additional software tools the process of identifying extension points along with extracting template components is explained. As the base for the demonstration, standard B2MML database is used.

Domain model, extension points and template components are used as the input for the next important step – automated software components generation. For this purpose specific highly customized generation tool is developed and presented in this thesis. Generation tool loads domain model, template component, generator class and using them creates new software component that can be included in a software project or compiled and immediately used as a library. Beside it is based on general approach, the system is optimized for Microsoft .NET platform. The general purpose of generation tool is displayed through processes of automatic creation of several different classes of components – Windows forms, value selection components, translation resources and access privilege lists. Together with generation, the process of component validation is described. Generated components are then used for the extension of existing applications.

Model based component generation is not the only way of domain model usage described in this dissertation. The separate chapter is dedicated to runtime model interpretation. For this purpose, special set of Web components is developed. The model interpretation library presented in this

dissertation is tuned up for reporting tools. Nevertheless, interpreting engine is based on independent runtime generation classes and could be easily extended for the other classes of components.

The last segment of dissertation defines guidelines for information system development and maintenance. Here is presented the overview on existing methodologies as well as a set of recommendations how to effectively combine them in different phases of information system life cycle. All subsystems are categorized and thus, the proper methodology is chosen. The recommendations for developing phase, as well as for deploying and maintaining are separately presented.

All presented software tools, methodologies and sets of recommendations are result of twelve years of effective working and research experience in the field of information systems. All of them are developed and implemented having in mind end users and its role in every aspect of information system life cycle. The framework is highly user focused and helps in developed system adoption process. All of the described tools and methodologies make a joint domain model based framework used to ease the software engineers' task from the modeling through development to maintenance phase of information system.

Sadržaj

1 Uvod ... 1

1.1 Pozicioniranje disertacije ... 5

1.2 Struktura disertacije ... 8

1.3 Pregled literature ... 9

2 Razvoj baziran na modelu (MDE) ... 12

2.1 Varijante u korišćenju MDE pristupa ... 15

2.2 Definisanje meta i domenskih modela ... 18

2.3 OpenEHR – meta model fokusiran na proširenje strukture sistema ... 18

2.4 ISA – 95 i B2MML – meta modeli fokusirani na modelovanje akcija ... 23

2.5 Definisanje specijalizovanog meta modela za zdravstvo ... 25

2.6 Alat za kreiranje modela podataka za informacione sisteme ... 30

2.7 Primer adaptacije alata za modelovanje ... 37

3 Dodatni alati za podršku razvoju baziranom na modelima ... 41

3.1 Reverzni inženjering – generisanje modela na osnovu baze podataka ... 42

3.2 Biblioteka za analizu strukture podataka ... 45

3.3 Alat za generisanje administratorskih aplikacija ... 51

3.4 Tačke proširenja ... 57

3.5 Šablonske komponente ... 66

4 Alat za generisanje softverskih komponenata ... 73

4.1 Generisanje formi ... 77

4.2 Generisanje drugih vrsta komponenti ... 85

4.3 Generisanje komponenti za selektovanje vrednosti ... 85

4.4 Generisanje prevoda ... 87

4.5 Konfiguracije za liste privilegija ... 88

4.6 Validacija generisanih komponenti ... 92

4.7 Razvoj aplikacije pomoću generisanih komponenata ... 101

4.8 Efekat generisanih komponenti ... 107

5 Interpretirane komponente ... 110

5.1 Prezentacioni sloj ... 116

5.2 Poređenje sa prethodno korišćenim rešenjima ... 118

6 Procedure i smernice za razvoj i održavanje informacionih sistema ... 120

6.1 Uslovi prihvatanja informacionih sistema ... 125
6.2 Definisanje arhitekture sistema ... 127
6.3 Uloga korisnika u modelovanju sistema ... 132
6.4 Smernice za efikasniji razvoj informacionih sistema ... 136
6.5 Smernice za instaliranje i implementiranje informacionih sistema ... 142
6.6 Problemi vezani za GUI i smernice za njihovo rešavanje ... 144
6.7 Smernice za dogradnju informacionih sistema ... 147
6.8 Primer strukturnog unapređenja sistema kroz CQRS pristup ... 149
7 Zaključak ... 157
8 Literatura ... 162

Spisak slika

Slika 1 Glavni činioci u razvoju informacionog sistema 2
Slika 2 Konceptualni prikaz tradicionalnih modela razvoja softvera 3
Slika 3 Elementi agilnih metoda 3
Slika 4 Evolucija u nivoima apstrakcije u programiranju 4
Slika 5 Osnovni koraci u razvoju baziranom na modelu 6
Slika 6 MDE - minimalistički pristup 15
Slika 7 MDE - scenario idealnog slučaja 16
Slika 8 MDE - pristup sa tačkama proširenja 17
Slika 9 Osnovni openEHR meta model 18
Slika 10 Primer enumeracije sastavljene od stringovskih vrednosti 19
Slika 11 Pozicija openEHR-a na dijagramu standarda u medicinskoj informatici [60] 20
Slika 12 Podaci i događaji vezani za merenje krvnog pritiska (dijagram je preuzet iz aplikacije openEHR Clinical Knowledge Manager [63]) 21
Slika 13 Primer upotrebe artefakta u kreiranju većeg medicinskog dokumenta. Arhetip za krvni pritisak (označen crveno) učestvuje u pregledu nakon srčanog udara (dijagram je preuzet iz aplikacije openEHR Clinical Knowledge Manager) 21
Slika 14 Šema za definisanje vrsta i pojedinih instanci opreme u okviru proizvodnje (preuzeta sa MESA sajta [64]) 23
Slika 15 Primer generisanog modela proizvodnje kroz alat ABB ECS Process Definition Tool (PDT) [65] 24
Slika 16 Meta model za Medis.NET 26
Slika 17 Proširenje meta modela za složene preglede 28
Slika 18 Deo baze podataka u kome se čuvaju EHR meta podaci 29
Slika 19 Skup uređenih koraka u okviru jednog pregleda 30
Slika 20 Primer koraka u složenom pregledu kod moždanog udara (primer preuzet sa http://www.openclinical.org/gmm_guide.html) 31
Slika 21 Primer konfiguracije alata za modelovanje medicinskih informacionih sistema 32
Slika 22 Primer definisanih pravila za proširenje 33
Slika 23 Primer konfiguracionog dijaloga alata za modelovanje 33
Slika 24 Konfiguracija sa definisanim nazivom kolekcije entiteta 34
Slika 25 EHR alat za modelovanje – početna strana 34
Slika 26 Pogled na listu definisanih pregleda kroz alat za modelovanje 35
Slika 27 Definisanje jednog pregleda (laboratorijska analiza na slici levo) i njegovih polja (desno) ... 35
Slika 28 Primer modela jednog zdravstvenog kartona 36
Slika 29 Primer modela jednog atributa pregleda sa definisanim nabrojivim opsegom 36
Slika 30 Meta model za definisanje zahteva za proizvodnju 37
Slika 31 Izgled osnovne forme nakon promene vizuelne komponente koja prikazuje elemente modela 38
Slika 32 Deo atributa definisanih u okviru klase ProductionOrder 39
Slika 33 Primer klase koja služi kao atribut za validaciju 40
Slika 34 Konfiguracija za reverzni inženjering 44
Slika 35 Factory projektni obrazac upotrebljen za kontrolu procesa instancijacije klasa 45
Slika 36 Ekstenzija OLE DB klasa 47
Slika 37 Primer meta podataka ekstrahovanih pomoću DatabaseStructure biblioteke 48

Slika 38 Primer tabele (HumanResources.Employee) koja učestvuje u skraćenom skupu tabela 48
Slika 39 Prikaz određenih cirkularnih relacija 49
Slika 40 Primer cirkularnih relacija reda 1 u standardnoj B2MML šemi 50
Slika 41 Odnos između komponenata generatorskog alata 52
Slika 42 Dijagram klasa alata za generisanje administratorske aplikacije 52
Slika 43 Proces generisanja aplikacije 53
Slika 44 Generisani projekat u razvojnom okruženju Visual Studio 54
Slika 45 Editovanje podataka iz odabrane tabele kroz automatski generisanu formu 55
Slika 46 Dijagram klasa generisane aplikacije 56
Slika 47 Proces izdvajanja tačaka proširenja 58
Slika 48 Primer NHibernate ICriteria upita 59
Slika 49 Upit koji vraća sve primarne ključeve 59
Slika 50 Upit koji vraća spoljne ključeve sa brojem referenci 60
Slika 51 Polja u kreiranim pogledima koja će jednoznačno biti preslikana u objektni model 60
Slika 52 Upit koji vraća sve tabele koje su spoljni ključ 60
Slika 53 Lista sa deset tabela iz B2MML baze koje su najčešće bile roditeljski entitet u "spoljni ključ" vezi 61
Slika 54 Prvih deset tabela po broju zapisa u njihovim tabelama-deci 61
Slika 55 Sve potencijalne tačke proširenja u B2MML bazi nakon proširenja skupa pravila 62
Slika 56 Deo B2MML modela koji se odnosi na zapisivanje podataka o proizvodnji (ProductionPerformance šema u okviru B2MML-a) 62
Slika 57 Veza između ProductionOrder i ProductionPerformance delova B2MML modela 64
Slika 58 Proces definisanja šablonskih komponenti 67
Slika 59 Primer potencijalne šablonske komponente 67
Slika 60 Izdvojeni kod koji se odnosi na definiciju specifičnih vizuelnih komponenata 68
Slika 61 Izdvojeni specifični kod u funkcijama za učitavanje i snimanje entiteta tipa dekurzus 69
Slika 62 Specifični kod koji se odnosi na postavljanje adekvatnih vrednosti u dizajneru forme 69
Slika 63 Implementacija metoda iz dodatih intertfejsa 70
Slika 64 Posebne komponente zamenjene komentarima koji će biti tačke proširenja 70
Slika 65 Kod u opštim metodama koji će biti zamenjen u procesu generisanja 71
Slika 66 Način funkcionisanja generatorskog alata 74
Slika 67 Primer konfiguracije alata za generisanje 75
Slika 68 Primeri definisanja dodatnih atributa uz tag generationType 76
Slika 69 Primer inicijalne deklaracije CodeDom objekata kroz konstruktor 77
Slika 70 Primer CodeDom koda za generisanje propertija 77
Slika 71 Potpis alata za generisanje na početku generisanog fajla 78
Slika 72 Primer koda za promenu potpisa metoda za generisanje 78
Slika 73 Properti sa proverom kod postavljanja vrednosti 79
Slika 74 Generisani kod u delu za deklarisanje promenljivih 80
Slika 75 Primer generisanog koda u delu gde se vrši instanciranje vizuelnih elemenata na formi 81
Slika 76 Deo generisanog koda u InitializeComponent metodi 82
Slika 77 Primer generisanog koda u InitializeComponent metodi koji se odnosi na listu sa više izbora predstavljenu ComboBox komponentom 82
Slika 78 Generisani kod u metodama za snimanje i učitavanje 83
Slika 79 Izgled generisane forme 84
Slika 80 Primer još jedne generisane forme u runtime okruženju, sa drugačije konfigurisanom kontrolom za prikaz podataka o pacijentu i sa prikazom standardnih vrednosti u okviru posebne labele 84

Slika 81 Proces modelovanja i generisanja uz prikazan skup najznačajnijih generisanih komponenti 85
Slika 82 Šablonizovana funkcija za filtriranje entiteta po odgovarajućem polju 86
Slika 83 Generisane metode za pretragu 86
Slika 84 Komponenta za selektovanje leka iz liste.. 87
Slika 85 Mesto za unos prevoda - kolekcija Translations u modelu 87
Slika 86 Primer generisanih parametara konfiguracionog profila... 89
Slika 87 Lista privilegija za formu koja prikazuje karton za predškolsku decu.................................. 90
Slika 88 Generisani dodatni parametri za formu koja prikazuje karton za predškolsku decu.............. 90
Slika 89 Metoda za primenu konfiguracije 91
Slika 90 Proširenje konstruktora novim parametrima 92
Slika 91 Generalni prikaz procesa testiranja komponente... 93
Slika 92 Vrste test projekata podržanih kroz Visual Studio 2013... 93
Slika 93 Primer metode za testiranje LoadValues funkcionalnosti 94
Slika 94 Test metoda koja podržava rad sa više test vektora... 95
Slika 95 Primer generalne klase za serijalizaciju i deserijalizaciju 96
Slika 96 Primer niza serijalizovanih test vektora 99
Slika 97 Primer metode za integraciono testiranje sa označenim generisanim parametrom.............. 100
Slika 98 Primer inicijalne konfiguracije aplikacije za primarno zdravstvo....................................... 101
Slika 99 Dodavanje konfiguracije za podršku pregledima na neurološkoj klinici 102
Slika 100 Konfiguracija za listu pregleda 103
Slika 101 Primer forme koja tabelarno prikazuje podržane preglede. Pregledi su podeljeni u grupe, gde svaka grupa ima svoju karticu 104
Slika 102 Uzorci sa procesora pre dodavanja modula "neuro" 104
Slika 103 Uzorci sa procesora nakon dodavanja modula "neuro".. 105
Slika 104 Procesi (niti) aktivni pre dodavanja novog modula.. 106
Slika 105 Niti aktivne u programu nakon dodavanja novog modula 106
Slika 106 Proces kreiranja i konfigurisanja novog izveštaja..113
Slika 107 Slojevita ("onion") arhitektura [86][87] 115
Slika 108 Kreiranje sadržaja vidžeta 117
Slika 109 Opšta struktura Sistema.. 130
Slika 110 Standardni skup slučajeva upotrebe 131
Slika 111 Izdvajanje glavnih grupa potprojekata i odabir adekvatnih metodologija......................... 137
Slika 112 Razvoj baziran na funkcionalnostima (Feature Driven Development) 138
Slika 113 Kanban tabela.. 139
Slika 114 Scrum tabela.. 140
Slika 115 Promena u rasporedu zadataka 140
Slika 116 Preraspodela rada kako bi se uravnotežilo opterećenje 140
Slika 117 Opšti prikaz procesa dogradnje informacionog sistema... 148
Slika 118 Modifikovani lean model za strukturnu dogradnju informacionih sistema........................ 149
Slika 119 Primer realizacije CQRS pristupa 150
Slika 120 Odnos glavnih nivoa u SOA sistemu bez (A) i sa (B) primenom CQRS arhitekture.......... 152
Slika 121 Forma za prijem pacijenata (levo) i njen prototip sa izdvojenim funkoionalnim celinama (desno)... 154
Slika 122 Forma koja prikazuje početnu stranu zdravstvenog kartona pacijenta – 1: demografski podaci, 2: medicinska upozorenja, 3: podaci vezani za osiguranje, 4: podaci o porodici 154
Slika 123 Inicijalna šema RD 155

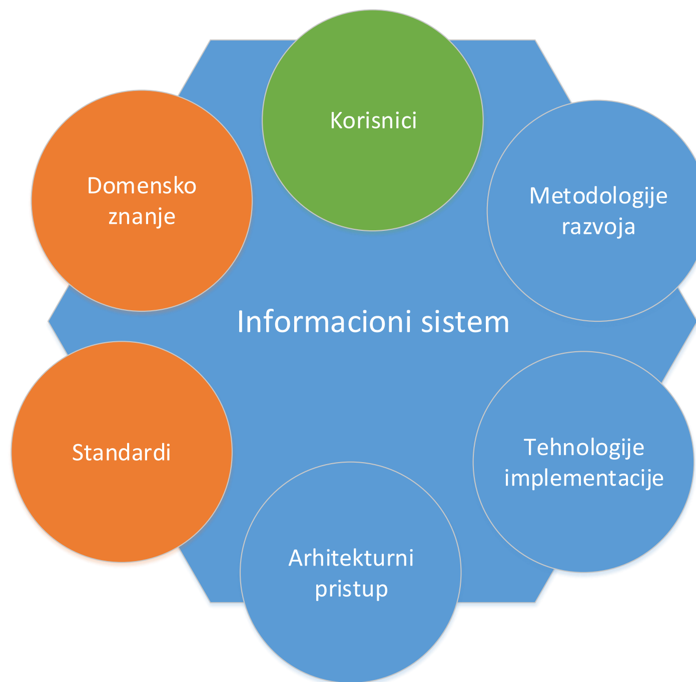
Spisak tabela

Tabela 1. Primeri redukcije ranga matrice incidencije za takozvane "test-baze" 51
Tabela 2 Broj izveštaja u toku godine 108
Tabela 3 Broj izveštaja u toku godine 112
Tabela 4 Broj formi za sakupljanje podataka 112
Tabela 5 Efekat korišćenja interpretiranih komponenata 119
Tabela 6 Nivo interakcije potencijalnih korisnika MIS-a (za projekat koji je počeo 2008)................ 126
Tabela 7 Efekti korišćenja inkrementalne strategije u procesu implementacije MIS.......................... 144
Tabela 8 Statistika tabela koje se najčešće koriste u „select“ upitima.. 153
Tabela 9 Efekat korišćenja read baze na formi za prijem pacijenata – 1: ReadPatientData, 2: ReadScheduledTerm.. 156
Tabela 10 Efekat korišćenja read baze na formi za zdravstveni karton – 1: ReadPatientData, 2: ReadElectronicPatientRecord, 3: ReadInsurance, 4: ReadEmployment 156

1 Uvod

Proces razvoja informacionih sistema uključuje mnoge oblasti računarstva i u isto vreme zahteva specifično domensko znanje koje je potrebno ugraditi kako bi sistem bio upotrebljiv krajnjem korisniku (Slika 1). Informacioni sistem se najčešće razvija po narudžbini, za specifičnog korisnika ili za primenu u određenoj vrsti institucija ili grani industrije [1][2]. U isto vreme, informacioni sistem predstavlja softver velikog obima, sa puno različitih funkcija, koji će konkurentno upotrebljavati veliki broj korisnika. Takođe, sama arhitektura sistema mora biti takva da omogućuje jednostavno proširenje sistema. Još jedan ključan zahtev koji se postavlja pred informacione sisteme je i skalabilnost – potreba da njihove performanse ne padaju proporcionalno povećanju broja korisnika i transakcija, već mnogo sporije [3].

Sem što treba da bude dobro projektovan i da efikasno i u realnom vremenu izvršava sve ugrađene funkcije, informacioni sistem treba da bude i jednostavan za korišćenje [4] i da postoji poslovno opravdanje za njegovo uvođenje [5]. Potreba da se rad učini boljim i efikasnijim je zapravo glavni pokretač svih projekata razvoja informacionih sistema. Iz želje da se unapredi posao, dolazi se do skupa zahteva na osnovu kojih se informacioni sistem projektuje, zatim razvija i na kraju instalira i implementira kod krajnjeg korisnika koji instalirani sistem i koristi za unapređenje svog poslovanja [6].

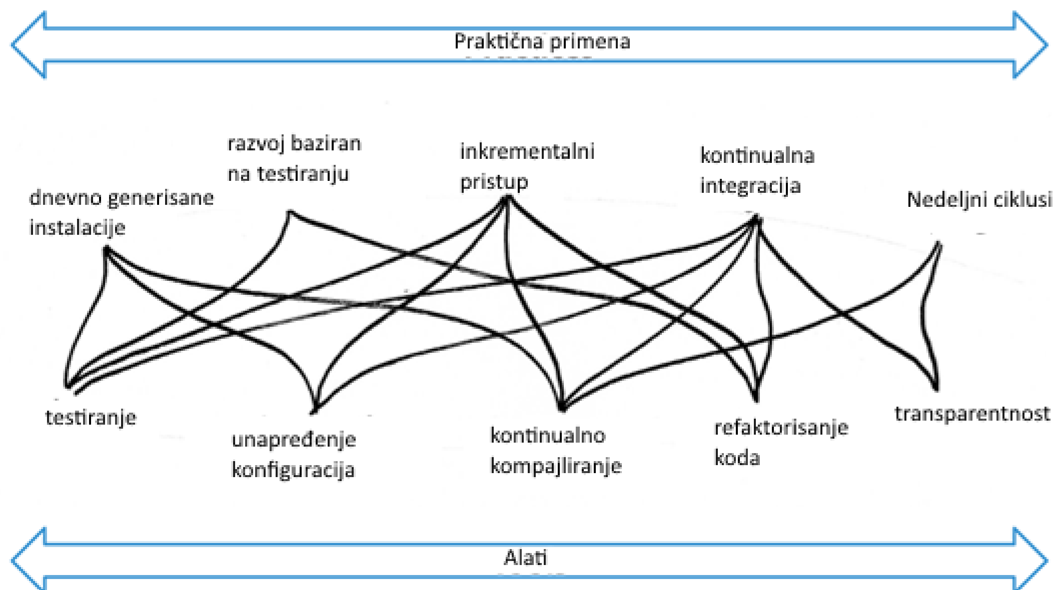


Slika 1 Glavni činioci u razvoju informacionog Sistema

Uzevši sve nivoe složenosti u obzir, vreme potrebno za kreiranje informacionog sistema, u mnogim slučajevima, je duže nego što bi zaista bilo neophodno, ili što bi potencijalni korisnici to želeli. Sa druge strane, vrlo često, informacioni sistemi obiluju komponentama koje dele sličan skup funkcionalnosti, a koje su bazirane na drugačijim skupovima podataka. Implementacija i testiranje takvih funkcionalnosti odnosi mnogo više vremena nego što bi realno bilo neophodno. Ovaj problem se u izvesnoj meri rešava standardnim alatima za automatsko generisanje koda, ali su oni najčešće dizajnirani tako da podržavaju isključivo jednu softversku platformu [7], ili su u stanju da generišu samo jednu kategoriju komponenti [8], tako da im je upotrebljivost ograničena.

Nadalje, u toku celog životnog ciklusa informacionog sistema, vrlo često od korisnika stižu zahtevi za proširenje i doradu postojećih funkcionalnosti [9][1[9]]. Korisnici ponekad zahtevaju i kreiranje novih ili čak i promenu nekog od osnovnih slojeva aplikacije – baze, objektnog modela ili skupa komponenti za prikaz. Takvi zahtevi stižu posle nekoliko godina korišćenja sistema, pa vreme koje se u ovakvim slučajevima utroši na identifikovanje pogodnih tačaka za proširenje može da prevaziđe čak i inicijalnu procenu vremena za celokupni razvoj.

Slika 2 Konceptualni prikaz tradicionalnih modela razvoja softvera

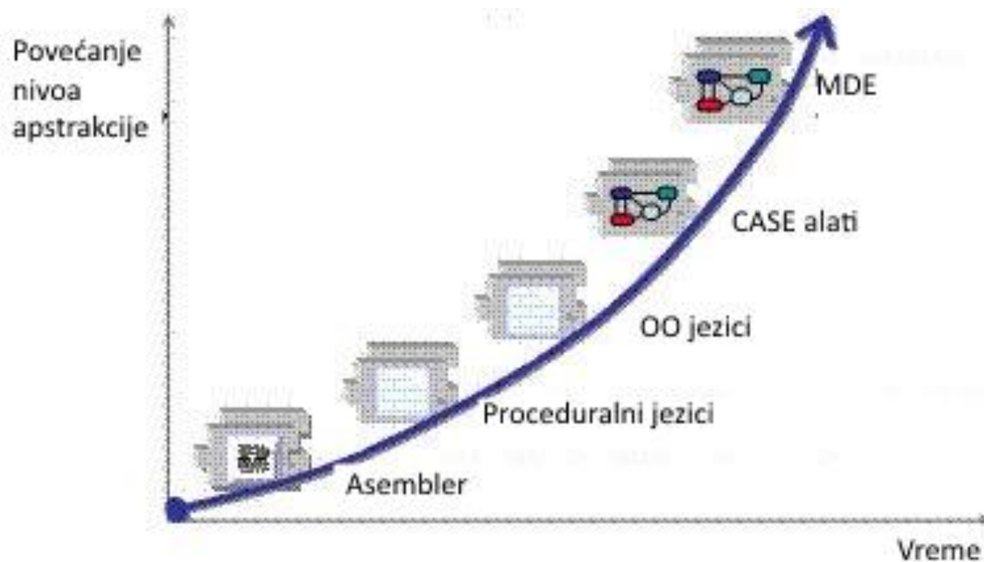


Slika 3 Elementi agilnih metoda

Za proces razvoja informacionih sistema postoji razvijen veliki broj metodologija koje se najčešće svrstavaju u dve grupe – tradicionalne i agilne metode razvoja [11][12][13][15]. U tradicionalne metode spadaju, na primer, model vodopada [16], RUP (rational unified process) [17] i V-model [18]. Njihova glavna prednost je to što su potpuno standardizovane i programeri imaju iskustva u radu sa njima. Nizak nivo interakcije programera i budućih korisnika tokom prilično dugog perioda razvoja je veliki nedostatak klasičnih pristupa. Nakon što se napravi inicijalna specifikacija, programeri razvijaju

komponente sistema, i sve dok se ne dođe do faze testiranja korisnici se uglavnom jako malo konsultuju [11].

Sa druge strane agilne metode poput SCRUM metodologije [19] ili razvoja baziranog na funkcionalnostima [20] (feature driven development) donose veću interakciju sa korisnicima što na kraju dovodi do veće verovatnoće prihvatanja krajnjeg rešenja. Problem koji mnogi programeri vide ovde je to što su pod konstantnim pritiskom zato što iteracije u agilnim metodama traju znatno kraće nego kod tradicionalnih. Takođe, agilni pristup razvoju i intenzivnija interakcija sa korisnicima zahteva od svih učesnika u razvoju informacionog sistema i izvesne društvene veštine, koje dobar programera ne mora da poseduje u opštem slučaju [11].



Slika 4 Evolucija u nivoima apstrakcije u programiranju

Glavni integrativni faktor za pomenute metode razvoja biće potražen u oblasti softverskog inženjerstva koja se bavi razvojem softvera baziranog na modelima podataka (Model Driven Engineering – MDE) [21][22]. Biće pokazano kako je u ovakvim slučajevima veoma korisno imati već gotove domenske modele podataka, kao i alate koji bi inicijalno generisali nove softverske komponente koje bi programeru dale početnu tačku od koje bi mogao da krene u razvoj informacionog sistema. Dodatno, ovde bi modeli podataka mogli da se iskoriste i za razvoj alata koji ispituju strukturu baze i pronalaze najzgodnije tačke za proširenje sistema, kao i za alate koji generišu testove i testiraju generisane komponente.

Ova doktorska disertacija nudi pogled na proces razvoja informacionih sistema koji će kombinovanjem tradicionalnih i agilnih metoda probati da iskoristi sve prednosti obe filozofije [23][24][25][26]. Kao ilustracija valjanosti predloženog pristupa, prvenstveno su prikazani rezultati dobijeni u toku razvoja medicinskih informacionih sistema (MIS). Pojedini koncepti su provereni i na procesu razvoja informacionih sistema za podršku proizvodnji (production information management system – PIMS). MIS kao mnogo osetljivija kategorija softvera, sa tačke gledišta prihvatanja od strane krajnjeg korisnika, su detaljnije razmatrani [27]. Medicinski informacioni sistemi, postoje jednako dugo kao i PIMS i informacioni sistemi za banke i osiguravajuća društva, ali je do skora njihov procenat uspešne implementacije bio neuporedivo manji. Zbog toga su u disertaciji formulisane i preporuke za efikasniji razvoj informacionih sistema, uz osvrt na najbitnije razloge odbacivanja MIS

sistema u prošlosti [28][29]. Detaljno je elaboriran uticaj svakog od glavnih faktora prihvatanja MIS sistema na kreirani skup preporuka.

U skladu sa tim, glavni cilj predložene doktorske disertacije je definisanje skupa procedura i softverskih alata koji bi se koristili u procesima razvoja, implementacije, dogradnje i održavanja informacionih sistema. Tehnički gledano, glavna primena pomenutih softverskih alata je vezana za razvoj skupova komponenti baziranih na sličnim baznim funkcionalnim celinama, a koji se direktno mogu integrisati u informacioni sistem. U osnovi predloženih procedura bi bio MDE pristup, baziran na domenskim modelima podataka, koji bi uključio i efikasnije generisanje raznovrsnih softverskih komponenti. Pomenute komponente bi bile generisane na osnovu već gotovih, testiranih i verifikovanih šablonskih komponenti i mogle bi odmah biti uključene u informacioni sistem. Sem toga, dodatni ciljevi su i definisanje pravila i procedura za kreiranje i verifikaciju šablonskih komponenti, kao i za identifikovanje tačaka proširenja u informacionom sistemu.

1.1 Pozicioniranje disertacije

Ova disertacija može da se klasifikuje u oblast razvoja softvera baziranog na modelu – MDE. MDE ima za cilj da poveća nivo apstrakcije u procesu specifikacije softvera i poveća stepen automatizacije u procesu njegovog razvoja. Evolutivno gledano, alati za kreiranje i interpretaciju modela su nastavak grane koja je započela sa programskim jezicima i nastavila se sa CASE (computer aided software engineering) alatima.

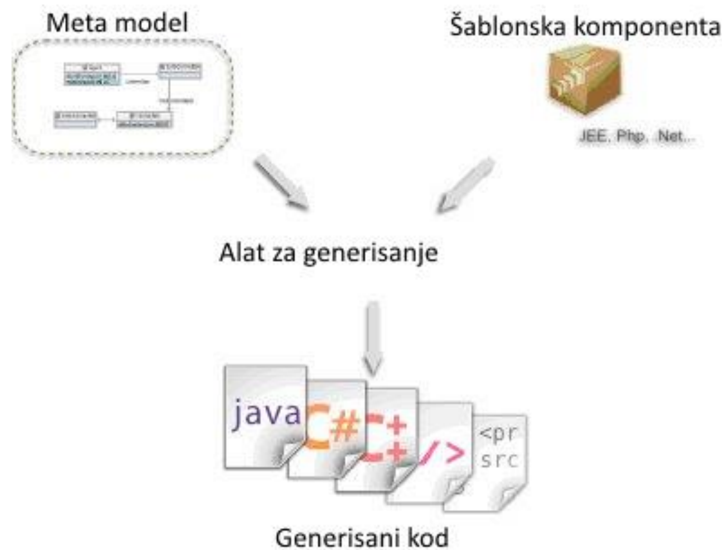
Ideja koju MDE pristup promoviše je da se koriste modeli na raznim nivoima apstrakcije za opisivanje sistema koji se razvijaju, a da se primena modela u gotovom softveru bazira na dva pristupa – generisanju koda i interpretaciji modela. Jedna od prednost koju MDE donosi je to što čini softver bližim krajnjem korisniku. Alati za modelovanje su zamišljeni tako da može da ih koristi bilo ko, i oni se u toku razvoja daju budućim korisnicima da kroz njih definišu svoje strukture podataka, što u krajnjoj liniji vodi ka većem procentu prihvatljivosti generisanih rešenja.

Korišćenje domenskog modela podataka ostavlja dovoljno mesta za razvoj programskog okvira za generisanje različitih softverskih komponenti koje bi se mogle uklopiti u postojeće projekte i razvojna okruženja. Ovakav razvojni okvir, zajedno sa dodatnim alatima za analizu strukture baze podataka i generisanje komponenti za sinhronizaciju i migraciju podataka značajno bi pomogao programerima u toku razvoja, a krajnji korisnici bi dobili lakše proširiv sistem. Razvoj softvera baziran na modelu, omogućuje uključivanje korisnika sa specifičnim domenskim znanjem u gotovo sve faze razvoja.

Krajnji korisnik sa specifičnim domenskim znanjem je, po predloženom pristupu, uključen u generisanje modela strukture i toka podataka, dok bi programeri mogli da se skoncentrišu na kritične delove sistema koji razvijaju. Dodatno, komponente generisane na osnovu modela bi krajnjem sistemu dale fleksibilnost u pogledu izbora tehnologije na svim nivoima – od baze podataka do tehnologije za razvoj prezentacionog sloja. Takođe, kroz alate za generisanje mogli bi biti direktno podržani različiti pristupi u kreiranju i održavanju baze podataka kao što je na primer pristup podeljenih baza za čitanje i upis. Poseban izazov u istraživanju biće kreiranje osnovnih struktura podataka i arhitekturna rešenja u alatima za modelovanje i generisanje softvera koja bi u krajnjoj instanci omogućila generisanje komponenti za više različitih softverskih platformi.

Postoji brojna literatura vezana za oblast razvoja softvera na osnovu modela podataka [30][31][32]. U mnogim radovima se objašnjavaju prednosti MDE pristupa [33][34][35], ali u isto vreme postoje i radovi koji ga značajno osporavaju [36][37]. Uzrok osporavanju ovog koncepta je to da su u poslednjoj deceniji neki autori pokušavali da na osnovu opštih modela generišu kompletne

informacione sisteme i, nažalost, nisu uspeli. U poslednjih par godina je preovladalo mišljenje da MDE ipak ne može da bude sveti gral programiranja, ali i da je mnogo više nego generisanje prostih formi na osnovu tabela iz baze. MDE je koristan koncept, ali mu je potrebno naći pravo mesto. Ova disertacija treba da bude prilog tvrđenju da su dogradnja informacionih sistema i generisanje podsistema predvidivog stepena složenosti pravo mesto gde treba koristiti MDE pristup [38][39][40][41]. Kada se MDE ovako pozicionira, definisani modeli podataka mogu da budu dovoljno opšti kako bi se obezbedila željena fleksibilnost. Sa druge strane, skup softverskih komponenti koje budu generisane na kraju može da bude dovoljno brojan i raznovrstan kako bi zaista dao vidljive efekte kroz skraćenje vremena potrebnog za sve korake u životnom ciklusu informacionog sistema.



Slika 5 Osnovni koraci u razvoju baziranom na modelu

S obzirom da je ceo proces generisanja koda na osnovu modela složen i uključuje više koraka (definisanje meta modela, definisanje specifičnih jezika za opis modela, kreiranje alata za modelovanje, generisanje koda, kreiranje alata za interpretaciju modela, transformisanje modela, testiranje modela i dobijenih komponenti, uključivanje dobijenih komponenti u odredišni informacioni sistem) postoji mnogo radova iz ove oblasti koji se bave samo nekim od njih [42][43][44].

Rezultati su najčešće rešenja koja se fokusiraju samo na neke od pomenutih koraka imajući za rezultat softverski proizvod koji je stabilan ali nema željeni stepen konfigurabilnosti [45]. Sa druge strane, mnogi autori čine napore da iskoriste postojeće jezike kao jezike modela a da se u većoj meri skoncentrišu na podršku za platformsku nezavisnost [46]. Još jedan cilj predložene teme je da nađe balans između pomenuta dva pristupa i da kroz ostvarene rezultate da doprinos metodologiji razvoja, kao što je MDE, koja u svakom trenutku uzima u obzir i programera i krajnjeg korisnika.

Istraživanja, predstavljena u ovoj doktorskoj disertaciji, će biti primarno fokusirana na kreiranje skupa procedura koje se mogu efikasno koristiti u procesu razvoja softverskih komponenti koje dele isti skup osnovnih funkcionalnosti. Iste procedure bi onda bile primenjene i u procesima vezanim za dogradnju informacionih sistema. Razvoj skupa procedura bi bio praćen razvojem softverskog okvira za proširenje informacionih sistema baziranog na modelu podataka. Osnovni koncept koji će biti podržan je pristup po kome se nakon definisanja modela, automatski generišu softverske komponente koje se odmah mogu uključiti u informacioni sistem. Na ovaj način se smanjuje vreme potrebno za razvoj,

povećava se uniformnost rešenja, skraćuje vreme potrebno za testiranje i izbegava dodavanje nepotrebne složenosti u sistem.

U osnovi predloženi pristup u razvoju i održavanju IS-a čini sledeći niz koraka: početno kreiranje prototipa pojedinih segmenata informacionih sistema prebacuje se iz samog informacionog sistema u okvir za razvoj baziran na modelu podataka (u daljem tekstu ORBM) gde bi se testirao i verifikovao; nakon definisanja modela, prototip se kopira u odgovarajuću komponentu koja se uključuje u informacioni sistem [9]. Ovde će biti podržani i pristup generisanja koda i pristup interpretacije modela, pošto u određenim slučajevima svaki od njih ima svoje prednosti. Takođe, kroz ovu disertaciju biće prikazani i softverski alati za generisanje komponenti definisani kroz proces razvoja MIS sistema.

Još jedan značajan element u razvoju predloženog pristupa je definisanje skupa pravila za verifikaciju i testiranje potencijalnih šablonskih komponenti, kao i pravila za njihovu transformaciju u šablonske komponente. Nakon toga su definisani kriterijumi za izdvajanje tačaka proširenja iz baze podataka na osnovu njene strukture.

Sledeći korak je definisanje skupa struktura podataka (meta modela) na osnovu koga će se generisati specifični domenski modeli podataka kroz alate za modelovanje. Meta model podataka će biti razvijen po ugledu na postojeće standarde koji se koriste u razvoju različitih kategorija informacionih sistema, kao što su na prvi pogled raznorodni ISA-95 [47] i openEHR [48], i težiće generalizaciji njihovih osnovnih koncepata. Cilj je stvoriti manje složen, ali šire primenljiv skup osnovnih entiteta koji će u krajnjoj liniji biti upotrebljen za generisanje domenskih modela. Kreirani domenski modeli biće korišćeni kao ulazna struktura za različite alate za automatsko generisanje softverskih komponenti koji treba da podrže pomenuti pristup, a gde će razvoj nekih od njih biti sastavni deo predložene doktorske teze.

S obzirom da promena modela koja vodi promeni informacionog sistema nije jedini smer u kome može da se dešavaju promene, dodatni aspekt ovog istraživanja biće razvoj komponenti za reverzni inženjering čija je uloga da na osnovu detektovanih promena i verifikacije strukture baze podataka ažuriraju model [40]. Zajedno sa njima, biće razvijene i komponente koje se mogu uključiti u informacioni sistem a koje će omogućiti rad sa novim strukturama sve dok se informacioni sistem ne ažurira kroz model.

Evaluacija predloženih rešenja biće obavljena kroz kreiranje domenskog modela za MIS sisteme, proširenje postojećeg ISA-95 modela i generisanje većeg broja softverskih komponenti koje će biti uključene u različite informacione sisteme. Kao primarna platforma za testiranje biće upotrebljen medicinski informacioni sistem Medis.NET razvijan na Elektronskom fakultetu u Nišu koji zbog svoje složenosti i dinamike razvoja predstavlja odličan primer složenog informacionog sistema u okviru koga se često generišu novi moduli koji dele osnovne funkcionalnosti. Što se tiče evaluacije komponenti koje interpretiraju model, one će biti uključene u demo aplikaciju koja podržava industrijski ISA-95 standard i B2MML (Business to Manufacturing Markup Language – B2MML) strukture podataka. Dodatno, u disertaciji će biti dat i uporedni prikaz rezultata dobijenih različitim pristupima u razvoju pojedinih delova sistema, kao i poređenje funkcionalnosti komponenti dobijenih generisanjem koda i komponenti koje interpretiraju model.

1.2 Struktura disertacije

Na kraju ovog poglavlja biće dat kratak pregled literature, nakon čega sledi opis karakterističnih pristupa razvoju baziranom na modelu. Biće opisani scenario najboljeg slučaja, minimalistički pristup i scenario baziran na tačkama proširenja.

Poglavlje posvećeno definisanju meta i domenskih modela podataka daje prikaz osnovnih openEHR i ISA-95 meta modela. OpenEHR je značajni predstavnik kategorije meta modela okrenutih ka proširenju strukture, a ISA-95 je fokusiran na proširenju kroz modelovanje akcija. U ovom poglavlju dat je i prikaz razvijenog meta modela za primenu u MIS sistemima koji treba da podrže poslovne procese iz zdravstvenog sistema Republike Srbije. Zajedno sa prikazanim meta-modelom dat je i opis razvijenog generalnog alata za modelovanje. Uz to, data je i primer adaptacije prikazanog alata za efikasniju primenu u PIMS sistemima baziranim na ISA-95 i B2MML-u.

Alati za modelovanje su u osnovi jednosmerne komponente koje služe da kreiraju domenski model i omoguće njegovo preslikavanje na ostatak sistema, pa kao takvi nisu potpuno pogodni za primenu u celokupnom životnom ciklusu informacionih sistema. Zbog toga ih treba proširiti dodatnim alatima koji će omogućiti njihovu širu primenu, kojima je posvećeno posebno poglavlje. U okviru ovog poglavlja prikazan je modul za reverzni inženjering koji služi da na osnovu postojećih struktura podataka ažurira model. Takođe, uz ovaj modul prikazana je i biblioteka za analizu strukture baze podataka kao i alat za generisanje aplikacija za administraciju baze. Biblioteka za analizu strukture baze ima primenu kod detektovanja određenih anomalija u bazi, kao i pri izdvajanju tačaka proširenja. Proces izdvajanja tačaka proširenja je prikazan na primeru već postojeće standardne B2MML baze. Na kraju poglavlja o dodatnim alatima je dat opis šablonskih komponenata i prikazan proces njihovog izdvajanja.

Na osnovu domenskih modela podataka generisanih kroz alat za modelovanje i njegovih pratećih komponenti, alat za generisanje softverskih komponenti generiše nove elemente sistema. Alat za generisanje je baziran na šablonskim komponentama i predstavlja još jedan doprinos ove disertacije. Dizajniran je tako da učita šablonsku komponentu, model podataka, generatorsku klasu i na osnovu njih kreira realnu komponentu. Kako su svi delovi ovog sistema zamenjivi nezavisno jedan od drugog, praktično je moguće generisati bilo kakve komponente u bilo kojoj tehnologiji, ali je sistem ipak optimizovan za .NET. U disertaciji je dat opis postupka generisanja nekoliko različitih klasa komponenti – Windows formi, komponenti za selektovanje vrednosti, resursa koji sadrže prevode kao i konfiguracija sa listama privilegija. Takođe, dat je opis procesa validacije generisanih komponenti kao i procesa proširenja aplikacije generisanim komponentama.

Sem pristupa generisanja koda na osnovu modela, u disertaciji je analizirana i mogućnost primene pristupa interpretacije modela u toku izvršenja programa. Za to je razvijen poseban skup Web komponenti koji je inicijalno prilagođen generisanju različitih izveštaja. S obzirom da je i ovde primenjen pristup sa nezavisnim interpretatorskim klasama (slično kao sa nezavisnim generatorskim klasama kod alata za generisanje) moguće je relativno jednostavno proširenje sistema kako bi podržao i druge klase komponenti.

Poslednje poglavlje u disertaciji se odnosi na procedure i smernice za razvoj i održavanje medicinskih informacionih sistema. Dat je osvrt na različite metodologije razvoja sistema, kao i skup preporuka kako ih kombinovati u različitim životnim fazama informacionog sistema. Posebno su izdvojene preporuke za fazu razvoja arhitekture i centralnog dela sistema i paralelno sa tim smernice za rad na korisničkom interfejsu. Uz ceo ovaj proces dat je i primer definisanja arhitekture novog sistema na osnovu korisničkih zahteva kao i opšteg okruženja u kome sistem treba da radi. Smernice vezane za

proces razvoja prati i skup preporuka za instaliranje i dogradnju informacionih sistema. Iako se ovi procesi čine jednostavnim, na osnovu iskustva iz prakse biće prikazani glavni problemi kao i načini njihovog prevazilaženja koji su dali dobre rezultate [49].

Svi prikazani softverski alati i smernice za razvoj predstavljaju sublimaciju dvanaestogodišnjeg iskustva u radu na razvoju više različitih tipova informacionih sistema. Kao važni elementi za ceo proces jasno su izdvojeni i glavni uslovi prihvatanja razvijenih sistema uz potenciranje uloge krajnjeg korisnika u celokupnom životnom ciklusu softvera. Svi elementi opisani u ovoj disertaciji čine jedan okvir za razvoj i održavanje informacionih sistema baziran na modelu podataka koji ima glavni cilj da olakša posao tehničkom osoblju kroz sve faze u životu softvera, i u isto vreme doprinese kreiranju sistema koji će krajnji korisnici mnogo lakše prihvatiti.

1.3 Pregled literature

Pošto se cela disertacija bavi procesima razvoja, održavanja i unapređenja polaznu tačku u istraživanju čini postojeće paradigme vezane za informacione sisteme. S obzirom da je ovo široka oblast, kao značajna literatura u širem smislu, a koja je direktno uticala na ovu disertaciju, predstavljaju knjige [1], [2] i [3].

One, ne samo što su značajni univerzitetski udžbenici, već i na veoma slikovit način objašnjavaju različite pristupe, metode razvoja, kao i uglove gledanja i razne tehničke i netehničke faktore koji utiču na ceo životni ciklus informacionih sistema. Kao što je u [1] navedeno, najveći izazov u razvoju informacionog sistema nije samo u rešavanju tehničkih problema, već konstantno držati sve njegove segmente u vezi sa stvarnim potrebama korisnika.

Sa druge strane, [2] daje dublju analizu o položaju informacionih sistema u okviru složenih poslovnih okruženja, sa posebnim osvrtom na to kako poslovni korisnici zapravo žele da koriste informacione sisteme. Posebno se naglašava da je svaki informacioni sistem tesno povezan za domen za koji se implementira i da korisnici iz različitih oblasti imaju drugačije prioritete. Takođe, daje i smernice o etičkim i društvenim aspektima svih prikupljenih podataka, kao i tome kako ih treba organizovati i štititi.

Kada se govori o vezi poslovnog okruženja i informacionih sistema, [3] je značajan izvor zato što je prvenstveno namenjen ljudima čija je osnovna struka ekonomija ili pravo. Sadašnja generacija menadžera je tehnološki pismena i ima mnogo realnija očekivanja od tehnologije. Kroz ovu knjigu se mogu detaljnije sagledati potrebe budućih korisnika, ali i bolje razumeti način njihovog razmišljanja, kao i želja da budu više uključeni u razvoj sistema koji će kasnije koristiti. Ovde se kroz nekoliko primera vezanih za sisteme za planiranje resursa i korisničke servise, budućim menadžerima približava koncept razvoja softvera baziranog na modelu, koji je glavni teoretski postulat ove disertacije.

Koncept razvoja baziranog na modelima podataka je deo i aktuelnih inicijativa za proširenje fokusa u različitim oblastima primene, kao što su na primer medicinski informacioni sistemi. U [4] su opisani izazovi proširenja aktuelnih MIS sistema kako u budućnosti ne bi bili okrenuti jedino ka zdravstvenim radnicima, veći i prema pacijentima. Gledano sa te strane, za takve projekte proširenja biće neophodno definisati mnoge nove komponente, a razvoj baziran na modelu bi mogao biti od velike pomoći. Smernice za ovakve projekte proširenja su date u poglavlju koje se bavi metodologijama razvoja.

Cela priča vezana za procese dogradnje informacionih sistema se mora pogledati i sa ekonomskog aspekta. U knjizi [5] date su detaljne smernice za više različitih pristupa analizi kako sistema koji se inicijalno razvija, tako i za procenu isplativosti različitih proširenja, od jednostavnih pa sve do integracija sa drugim sistemima. Dodatak tome je i rad prikazan u [6] gde je prikazana poslovna

filozofija koja stoji iza odluke da se neki softverski sistem uvede u upotrebu, proširi ili integriše sa drugim sistemom.

Posebno osetljivo pitanje za buduće korisnike je razvoj korisničkog interfejsa. Ova disertacija ide u prilog korišćenja razvoja baziranog na modelu. Rešenje dato u [7] je primer korišćenja modela za razvoj interfejsa, ali je usko specijalizovan samo za jednu vrstu modela koji se koriste za primenu u hemiji. Sa druge strane, mnogo generalnije rešenje je dati kroz patent [8]. Ovaj patent se opisuje sistem koji je kreiran za efikasni razvoj korisničkog interfejsa i kao takav je direktno primenjiv u praksi.

Problem nastaje onoga trenutka kada počinje da bude neophodno generisati nove klase komponenata na osnovu istog modela. Zbog toga će u ovoj disertaciji biti prikazano generalizovano rešenje čije komponente su slabije povezane, i čija upotrebljivost će biti prikazana na dva različita sistema MIS i PIMS.

Kao što je već pomenuto, upotreba domenskih modela nije striktno vezana za proces razvoja. U [9] je prikazana evaluacija mogućeg održavanja medicinskog, softvera baziranog na openEHR standardu i meta modelu. Dati su odgovarajući slučajevi održavanja i proširenja sistema i na osnovu njih urađena analiza potrebnih resursa. Kao što je u radu primećeno, izuzetno je značajno kako je projektovan osnovni model i kako ga je lako proširiti za potrebe dogradnje sistema.

Upravo imajući tu preporuku na umu, zajedno sa višegodišnjim iskustvom u razvoju MIS sistema, razvijen je poseban meta model za zdravstvene informacione sisteme. Baziran je tako da se lako može obogatiti novim tačkama proširenja, a u isto vreme je baziran na openEHR standardu.

Kada se govori o razvoju meta modela rad koji je iskorišćen kao zbog prikazanih konkretnih smernica je [21]. U njemu su autori prikazali značajne zahteve koji se predstavljaju pred meta modele koji se razvijaju. Šest glavnih zahteva koje meta model treba da ispuni, predstavljenih na isti način kao i u [21], su:

1. Model mora definisati sve neophodne osnovne koncepte, kao i pravila koja opisuju njihovo korišćenje
2. U okviru modela treba da se koristi uniformna notacija
3. Mora biti precizno definisano kako elementi modela odslikavaju entitete iz stvarnog sveta
4. Model mora biti definisan tako da ga je kasnije moguće lako proširiti novim konceptima
5. Model mora da podrži razmenu elemenata između više modela, uključujući i kasnije izvedene modele
6. Model mora da omogući naknadno uvedena specifična proširenja

Opšte postavke metodologije razvoja baziranog na modelu date su [22] i [33]. Dok je prvi od dva citirana rada opis svih neophodnih teoretskih postavki, knjiga [33] daje veliki broj praktičnih primera. Sve što je opisano podržano je kroz odgovarajuće softverske alate definisane kroz Java/Eclipse okruženje. Ono što bilo od posebne koristi su bili detaljni opisi transformacije modela u model, odnosno u tekst, što je predstavljalo osnovu za razvoj posebnih alata za modelovanje prikazanih u ovoj disertaciji. Sledeći značajan izvor je [34]. Ovde je uz adekvatan teoretski uvod opisan proces testiranja modela kao i komponenti dobijenih na osnovu njega.

Kako je sam proces modelovanja jednosmeran, da bi se dobio sistem koji može da omogući i ažuriranje modela na osnovu postojećih sistema, neophodno je uključiti i pristup reverznog inženjeringa u proces modelovanja. Dakle, nije neophodno samo omogućiti proširenje postojećih sistema na osnovu modela, već i proširenje modela na osnovu komponenti postojećih sistema. Takav pristup je detaljno primenjen u [40].

Tu je predložen MDE pristup koji putem automatskog reverznog inženjeringa generiše GUI model i automatski kreira odgovarajuće forme i kontrole. Tu je predložen MDE pristup koji putem automatskog reverznog inženjeringa generiše GUI model i automatski kreira odgovarajuće forme i kontrole. Za razliku od pomenutog sistema, sistem reverznog inženjeringa opisan u ovoj disertaciji generiše samo model, ali model opšteg tipa, a kasnije taj model koristi za generisanje raznih vrsta komponenti, ne samo za GUI forme.

Drugi deo disertacije se odnosi na smernice za razvoj, implementaciju, održavanje i proširenje informacionih sistema. S obzirom da su svi ovi procesi dosta složeni, predložen je hibridni pristup u kome se koriste i standardne i agilne metodologije razvoja. Na osnovu ličnog iskustva kao i preporuka datih u [23][24][25][26][27], može se pokazati da sve od navedenih metoda imaju svoje mesto u složenim procesima vezanih za životni ciklus informacionih sistema, i da je zapravo veoma važno podeliti ceo projekat informacionog sistema na više potprojekata i onda kod svakog od njih primeniti najbolju metodologiju, ili kombinaciju nekoliko njih. Ovome je posvećeno celo poglavlje, i date su posebne smernice i za razvoj i za implementaciju i za dogradnju informacionih sistema.

2 Razvoj baziran na modelu (MDE)

Kada su razvijani prvi programski jezici, glavni cilj je bio da se programeri na izvestan način zaštite od mašinskih jezika i da se proces kodiranja podigne na viši, apstraktniji nivo. Pisanje koda koristeći direktne instrukcije procesora i tačne memorijske adrese bio bi izuzetno mukotrpan posao i zahtevao bi previse vremena čak i za najjednostavnije programe. Programski jezici su doneli unapređenje nivoa apstrakcije [50], ali se to unapređenje odnosilo samo na računarske tehnologije. Programerima je olakšan posao, ali i je ljudima koji poseduju znanje o specifičnom domenu (zdravstvo, industrija, osiguranje, bankarstvo, biohemija) a koji žele da svoj svakodnevni posao olakšaju korišćenjem softvera proces razvoja informacionih sistema ostao crna kutija. Ovo je imalo za posledicu da se u mnogim domenima, kao što je na primer zdravstvo, mnoga dobra rešenja (sa tehničke tačke gledišta) jednostavno ne prihvate. Nažalost, korisnici su očekivali da detaljno razumeju kako softver funkcioniše pre nego ga prihvate. Nekome ko spasava ljudske živote je izuzetno bitno da može da se osloni na tehnologiju, a u slučaju kada razumevanje tehnologije izostane, ona se jednostavno ostavi po strani.

Kod napisan u nekom programskom nije razumljiv svima i izuzetno je teško objasniti radi sve neophodne činjenice krajnjem korisniku koji insistira da vidi kako program zapravo radi. Želja da se proces razvoja informacionih sistema približi krajnjim korisnicima datira još iz vremena razvoja prvih efikasnih informacionih sistema, tj. još iz sedamdesetih godina dvadesetog veka. Značajnim probojem na ovom polju može se smatrati nastanak prvih CASE (CASE – computer aided software development) alata [51]. Oni su omogućili korisnicima da procese opišu putem grafičkih reprezentacija programskih struktura (naredbe dodele, grananja, petlje) povezane u dijagrame toka podataka. Takođe, napredniji alati su omogućili kreiranje dijagrama stanja kao i dijagrame klasa u objektno orijentisanim jezicima.

Glavni ciljevi koji su imali tvorci CASE alata bio je da se na putem grafičke reprezentacije približe krajnjem korisniku, a da se u isto vreme omogući efikasno generisanje koda na osnovu nje. Na ovaj način bi se u isto vreme povećao procenat prihvatanja softverskih rešenja i u isto vreme redukovalo vreme potrebno za razvoj i testiranje. I sa današnje tačke gledišta ovo izgleda prilično obećavajuće, a u vreme svog nastanka izazvalo je veliko interesovanje među naučnicima. Međutim, tehnološka ograničenja računarskih sistema iz osamdesetih i devedesetih godina su uticala da ovaj koncept vrlo malo zaživi u stvarnosti. Mnogim računarskim sistemima u to vreme je bio pravi izazov da prikažu dijagram sa hiljadu čvorova, što je za bilo kakav industrijski proces sasvim uobičajen broj. CASE alati su najveću primenu, na kraju, našli među programerima koji su ih koristili za dokumentovanje svojih algoritama [52].

Sa druge strane, dalji razvoj programskih jezika doneo je i dalji razvoj apstrakcije ali je ona dostupna samo programerima. Krajnjem korisniku je kod razvijen na jezicima kao što su C#, Java ili C++ jednako težak za čitanje i razumevanje kao i kod napisan u Pascal-u. Korišćenje savremenih razvojnih platformi današnjim programerima daje velike kolekcije gotovih klasa koje oni mogu direktno da koriste u svojim programima što bi trebalo da omogući brži i efikasniji razvoj koda [53]. Međutim, to u praksi i nije baš uvek tako. Vremenom su platforme za razvoj postale toliko složene i opširne da je mnogim programerima potrebno značajno vreme da nauče pravilno da koriste samo njihove delove. Takođe, korišćenje određenih klasa iz razvojne platforme na neadekvatan način u jednom delu softvera može da proizvede neželjene bočne efekte u drugim delovima. Zbog toga, programeri često moraju da utroše mnogo više vremena na samo programiranje umesto da budu više fokusirani na arhitekturni nivo što dovodi u mnogim slučajevima do narušavanja stabilnosti sistema, degradiranja njegovih performansi, pisanja duplog koda i prolongiranja procesa testiranja [54]. Kasnije, kada dođe do održavanja i dogradnje sistema koji već radi, problemi se multiplikuju i nekada ispravljanje minornih bagova zahteva dugo testiranje, koje je teško objasniti krajnjem korisniku [55][56].

Sledeći korak u nastojanju da se poveća apstraktnost opisa sistema, olakša posao programerima i obezbedi bolja razumljivost gotovog softverskog proizvoda je razvoj baziran na modelu (Model Driven Engineering – MDE) [22]. On je nastao iz nemogućnosti programskih jezika treće generacije da smanje ukupnu kompleksnost koda i da na razumljiv način predstave domenske koncepte. Osnovu MDE pristupa čine modeli podataka bazirani na meta-modelima koji su karakteristični sa specifičnu oblast. Uz meta modele podataka drugi značajan element su jezici za kreiranje modela (Domain Specific Modeling Language - DSML) koji služe da se dodatno opiše struktura i ponašanaje aplikacije u okviru pojedinačnih domena – medicinskih informacionih sistema, proizvodnih sistema, finansijskih servisa, sistema skladišta, itd. Meta modeli se sastoje od kolekcije osnovnih koncepata u domenu i relacija među njima. Takođe, meta modeli definišu ključnu semantiku i uvode neophodna ograničenja nad relacijama.

Model koji se kreira na osnovu domenskog meta modela i DSML alata se dalje koristi kako bi se sintetisali različiti artefakti – počev od koda u odgovarajućem programskom jeziku, preko konfiguracionih fajlova i dokumentacije, pa sve do modela alterantivnih modela. Za sintetisanje pomenutih artefakata zadužene su razni transformacioni (Transformation engines) i generatorski (code generators) alati koji imaju i dodatnu ulogu da proveravaju sam model. Sem za generisanje komponenata, model može biti ulaz i u takozvanu interpretatorsku komponentu. Njena uloga je da, u toku izvršenja programa, učita model i odgovarajuće podatke, procesira ih i prikaže rezultat. Prednost generisanih komponenti je brzina rada, a interpretiranih fleksibilnost.

Unifikovani proces generisanja softverskih komponenti na osnovu modela doprinosi većoj stabilnosti sistema kroz veću konzistentnost između polaznih podataka i krajnjeg softverskog proizvoda. Dalje, kako modeli mogu da uoče greške i da ne dozvole generisanje kode pre njihovog ispravljanja, automatizovani proces generisanja kode se često naziva i „correct by construction“ [57]. Ovo je potpuno suprotno standardnom procesu programiranja koji se bazira na debagovanju i ispravljanju grešaka i koji se onda može nazvati „construct by correction“.

Ovo zapravo predstavlja najveći plus na strani MDE pristupa i čini ga izuzetno pogodnim izborom za procese dogradnje i unapređenja informacionih sistema. U idealnom slučaju, dogradnja se svodi na ažuriranje modela i jednostavno ponovno generisanje ažuriranih komponenti. Nažalost, u realnosti nije baš toliko jednostavno, ali MDE pristup svakako daje značajne rezultate i doprinosi većoj efikasnosti celog procesa.

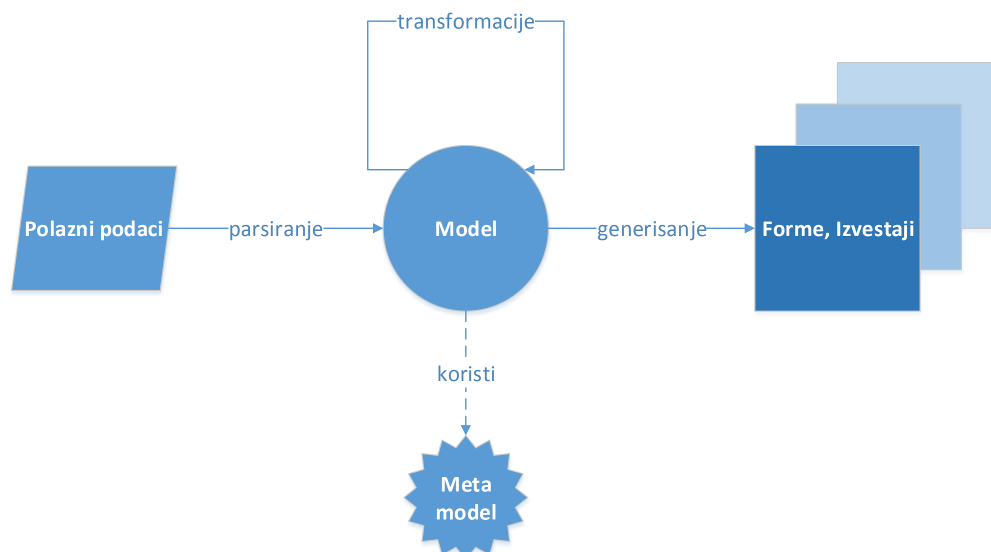
MDE se trenutno ne predstavlja kao apsolutno najbolji pristup ikada u programiranju, i istraživanja nisu okrenuta ka traženju najopštijih mogućih meta modela. MDE meta modeli se definišu za odgovarajuće domene, i za svaki od njih se kreiraju odgovarajući DSML alati koji generišu odgovarajuće komponente. Sa CASE alatima je svojevremeno pokušano da se kreira nešto što bi se moglo nazvati opštim meta modelom, i cela priča nije dala očekivane rezultate. Iako se vidi izvestan potencijal da se MDE može koristiti za generisanje celih aplikacija, mnogo efikasnijim se čini njegova primena za razvoj pojedinih segmenata složenih sistema. U ovoj disertaciji MDE će biti korišćen kao osnova za razvoj funkcionalnosti koje dele zajedničku osnovu, kao i jedan od glavnih koncepata kako u procesu razvoja, tako i u procesu dogradnje informacionih sistema.

2.1 Varijante u korišćenju MDE pristupa

Kada se pominje MDE, može se definisati do kog nivoa će se aplikacija razvijati na osnovu modela, a gde će biti primenjeni standardni metodi razvoja. U skladu sa tim, mogu se definisati tri osnovna pristupa korišćenju MDE principa:

- Minimalistički pristup
- Scenario idealnog slučaja
- Scenario sa tačkama proširenja

Minimalistički pristup, u okviru MDE metodologije, bi bio korišćenje fiksnih meta modela za generisanje predefinisanih softverskih komponenti kao što su forme i izveštaji. Na osnovu polaznih podataka, i postojećeg meta modela kreirao bi se model kao privremeni entitet. Model bi podržavao minimalni skup transformacija, a generatorski alat bi pri tome generisao forme i izveštaje.



Slika 6 MDE - minimalistički pristup

Ovakav pristup je prisutan kod mnogih komercijalnih alata za generisanje formi i izveštaja na osnovu već postojećih tabela u bazi. Primer za to su različiti form wizard alati za Visual Studio kao i Microsoft Reporting Service za Microsoft SQL Server bazu podataka.

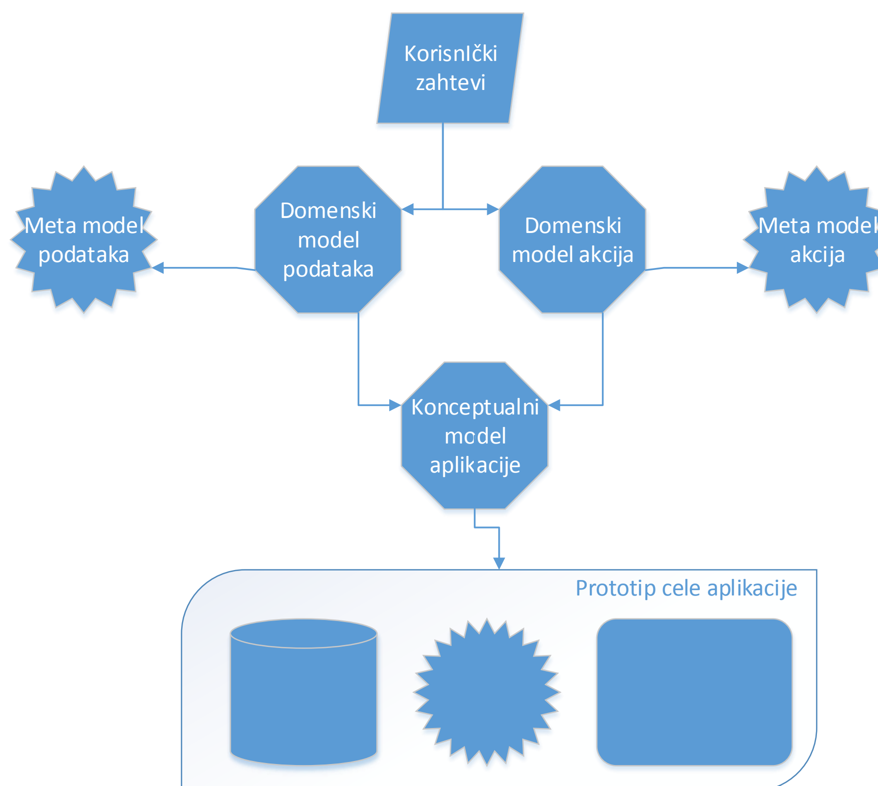
Kod pomenutih alata, meta model je deo njih samih i opisuje strukture podataka neophodne za generisanje krajnjih komponenti. Ulazni podaci za ove sisteme predstavlja struktura odgovarajuće tabele baze podataka ili SQL upit. Na osnovu njih i meta modela kreira se model podataka. Nad modelom su najčešće su dozvoljene izvesne transformacije kao što su definisanje sortiranja, grupisanja i agregacijskih funkcija. Dodatno, korisnik može da definiše izgled dobijene softverske komponente i da pokrene proces generisanja.

Glavna prednost u ovakvom pristupu automatizaciji procesa dogradnje informacionih sistema je to što se generisanje novih komponenata bazira na testiranom i verifikovanom meta modelu, i što nije

potrebno uložiti vreme i napor za razvoj osnovnih šablonskih komponenata. Generisane komponente će biti unifikovane i programer neće gubiti vreme na kreiranje formi i njihovo povezivanje za bazom.

Nedostatak kod ovog pristupa je fleksibilnost. Korisnici pomenutih alata ne mogu da utiču na meta model i na predefinisani skup akcija. Takođe, skup predefinisanih komponenti je često neproširiv i vrlo često je neophodna dodatna intervencija nad generisanom komponentom pre nego što se uklopi u sistem. Na kraju, ovakve komponente su najčešće izgrađene oko jedne vrste modela podataka i u slučaju kada je potrebno menjati tehnologiju modela podataka, postojeće komponente se ne mogu ponovo generisati istim alatom.

U idealnom slučaju, sa tačke gledišta primene MDE pristupa, generatorski alat bi trebao da izgeneriše kompletnu aplikaciju. Razvoj ovakvih alata, za sada, predstavlja samo koncept i sa tačke gledišta koliko bi napora bilo potrebno uložiti u njihov razvoj u odnosu na krajnji dobitak čine se prilično neisplativim. Ova tvrdnja posebno dobija na značaju kada se uzmu u obzir moderni razvojni alati koji programerima u mnogome olakšavaju posao i imaju integrisane razne opcije za ubrzanje razvoja.

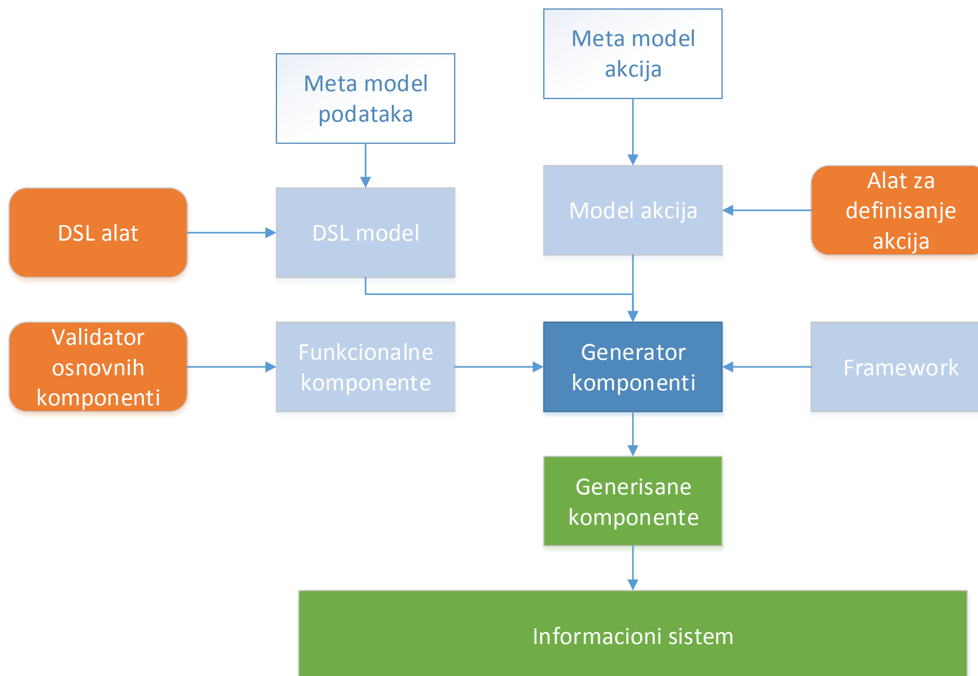


Slika 7 MDE - scenario idealnog slučaja

Ipak, u idealnom MDE slučaju, alat za modelovanje bi bio sposoban da na osnovu korisničkih zahteva, meta modela podataka i meta modela akcija generiše domenske modele podataka i akcija. Dalje, generatorski alat bi definisao konceptualni model aplikacija, iz koga bi kasnije proizašao ceo informacioni sistem – od baze podataka, preko modela podataka, poslovne logike do korisničkog interfejsa. Prednost ovakvog pristupa je evidentna, ali nivo složenosti generatorskih i alata za modelovanje ga ipak drži dalje od relanosti.

Primena MDE pristupa za proces dogradnje, pa i sam proces razvoja informacionih sistema, koja se bazira na tačkama proširenja sistema predstavlja, može se reći, pravu meru korišćenja MDE pristupa

za razvoj aplikacija. Ideja kod ovog pristupa je da se osnova informacionog sistema razvije korišćenjem standardnih razvojnih alata, ali da se nad aplikacijom definišu tačke proširenja.



Slika 8 MDE - pristup sa tačkama proširenja

Tačke proširenja su mesta u samoj aplikaciji koja omogućuju dodavanje novih komponenti u sistem, a koja su projektovana tako da podržavaju skalabilnost – povećanje obima podataka uz minimalan gubitak performansi. Primer za tako nešto je dodavanje novih tipova pregleda i zdravstvenih kartona u medicinski informacioni sistem.

U ovom pristupu glavna komponenta je alat za modelovanje podataka koji je definisan za specifični domenski jezik (domain specific language – DSL). Na osnovu unetih podataka i meta modela podataka on generiše DSL model koji će kasnije biti korišćen u generatorskom alatu. Sa druge strane, na osnovu meta modela akcija, kroz alat za definisanje akcija definiše se model akcija nad podacima iz DSL modela, spajaju se u jedan model i prosleđuju alatu za generisanje komponenata.

Generator komponenti učitava šablonske komponente i pripremljeni model podataka i na osnovu njih kreira softversku komponentu u okviru odgovarajućeg razvojnog okruženja. Takva komponenta se može zatim direktno uključiti u projekat informacionog sistema.

Kod ovakvog pristupa prednost je veća fleksibilnost u odnosu na minimalistički pristup. Može se definisati veći skup šablonskih komponenti koje imaju isti osnovni skup deljenih funkcionalnosti i na osnovu njih kasnije generisati komponente koje se uključuju u sam sistem. Nedostatak je to što programer mora sam da razvije, testira i validira šablonske komponente pre nego što ih uključi u sistem.

Ipak, početni korak u primeni bilo koga od navedenih varijanti MDE pristupa je definisanje meta modela podataka na osnovu koga će se razvijati domenski modeli podataka. Prateća komponenta procesa razvoja je alat za modelovanje, koji treba da obezbedi jednostavan korisnički interfejs kako za programere tako i za krajnje korisnike.

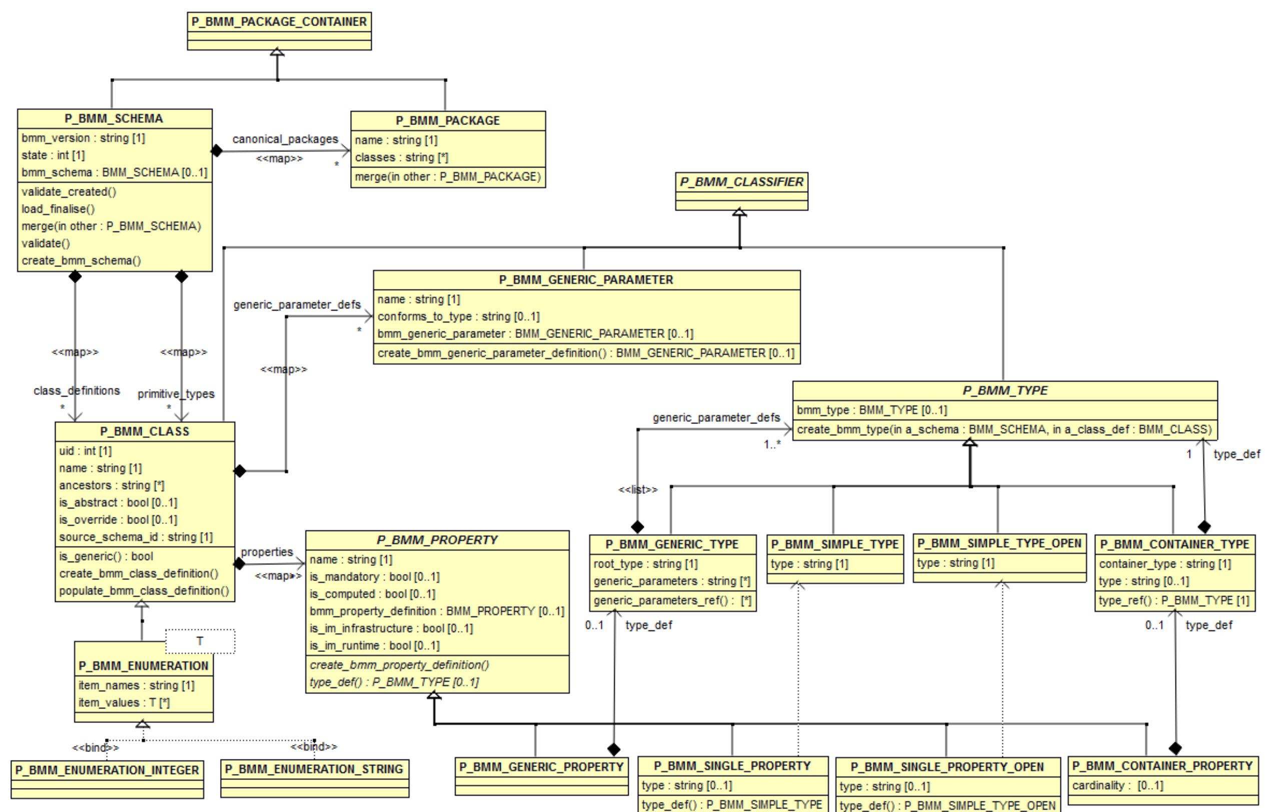
2.2 Definisanje meta i domenskih modela

Jedan od glavnih ciljeva istraživanja predstavljenog u ovoj disertaciji je i definisanje meta modela podataka koji bi se koristio u modelovanju medicinskih informacionih sistema za zdravstvene ustanove u Republici Srbiji. Definisanje meta modela je osnovni korak u razvoju baziranom na modelu i meta model u suštini predstavlja model po kome će biti kreirani domenski modeli podataka.

Za kreiranje meta modela za određenu oblast vrlo je bitno pronaći pravu meri između opšteg i posebnog i kreirati meta model tako da se iz njega lako mogu definisati domenski modeli koji će moći, bez kasnijih intervencija na meta modelu, da pokriju mnoge segmente domenskog znanja [58]. U ovom poglavlju biće opisana struktura domenskog modela podataka za primenu u MIS sistemima.

2.3 OpenEHR – meta model fokusiran na proširenje strukture sistema

Kao polazna tačka u kreiranju meta modela za MIS sisteme koji se mogu koristiti u zdravstvenim organizacijama u Republici Srbiji, uzet je standardizovani osnovni openEHR meta model (Slika 9). Osnovni meta model iz openEHR-a se sastoji iz više različitih šema, od kojih je najznačajnija ona koja definiše odnose između osnovnih tipova [59].



Slika 9 Osnovni openEHR meta model

Osnovni meta model (Base Meta Model – BMM) daje strukturu za definisanje skupa tipova podataka koje će biti podržane iz modela. Glavni deo jednog modela je šema (P_BMM_Schema). Šema je uz paket (P_BMM_Package) vrsta kontejnerskog entiteta. Odnos između njih je takav, da u okviru jedne šeme, može da postoji više paketa koji su prosti elementi za grupisanje. Inače, obe klase nasleđuju P_BMM_Package_Container, i ne obezbeđuju mogućnost kompozicije (projektni obrazac Composite nije podržan [60]). Posledica toga je da šema može da sadrži samo niz paketa, dok jedan paket u sebi sadrži samo listu klasa, a ne i druge pakete. Ova osobina se može smatrati i nedostatkom, zato što je

onemogućena dublja hijerarhijska struktura, ali i prednošću sa tačke gledišta razumevanja strukture sistema.

Svi kontejnerski entiteti, i paket i šema, sastoje se od niza klasa (P_BMM_Class) od kojih neke modeluju osnovne tipove podataka a neke klase sa više atributa. Klase predstavljaju glavni strukturni entitet. Njima se opisuje struktura svih vrsta objekata koji će biti uključeni u domenski model. Kao klase se modeluje sve od jednostavnih tipova, preko prostih struktura, pa sve do klasa koje opisuju složene specijalističke preglede sa par stotina stavki.

Kao specijalan slučaj klase, ovde su definisane enumeracije, odnosno nabrojivi tipovi podataka, koji su, zbog specifičnih primena u zdravstvu podeljeni na numeričke i stringovske nabrojive tipove. Razlog za definisanje stringovskih enumeracija treba tražiti u mogućnosti izbora iz liste vrednosti kao najstandardnijeg zahteva za mnoge preglede. Na primer, vrednost atributa pozicija_srca može da uzme vrednost levo ili desno. Takođe, postoje mnoge složene enumeracije sa više izbora, a primer jedne takve, definisane u domenskom modelu za kardiološke pregleda prikazuje Slika 10.

```
- <MedicalRecordItemElement>
    <Name>AV valvula zajednicka po tipu inkompl AV kanala</Name>
    <Type>shorttext</Type>
    <Description/>
  - <Ranges>
    - <MedicalRecordItemElementRange>
        <RangeName>ASD I</RangeName>
      </MedicalRecordItemElementRange>
    - <MedicalRecordItemElementRange>
        <RangeName>Inlet VSD</RangeName>
      </MedicalRecordItemElementRange>
    - <MedicalRecordItemElementRange>
        <RangeName>Rascep TV</RangeName>
      </MedicalRecordItemElementRange>
    - <MedicalRecordItemElementRange>
        <RangeName>Rascep MV</RangeName>
      </MedicalRecordItemElementRange>
    </Ranges>
    <MeasurementUnit/>
  </MedicalRecordItemElement>
```

Slika 10 Primer enumeracije sastavljene od stringovskih vrednosti

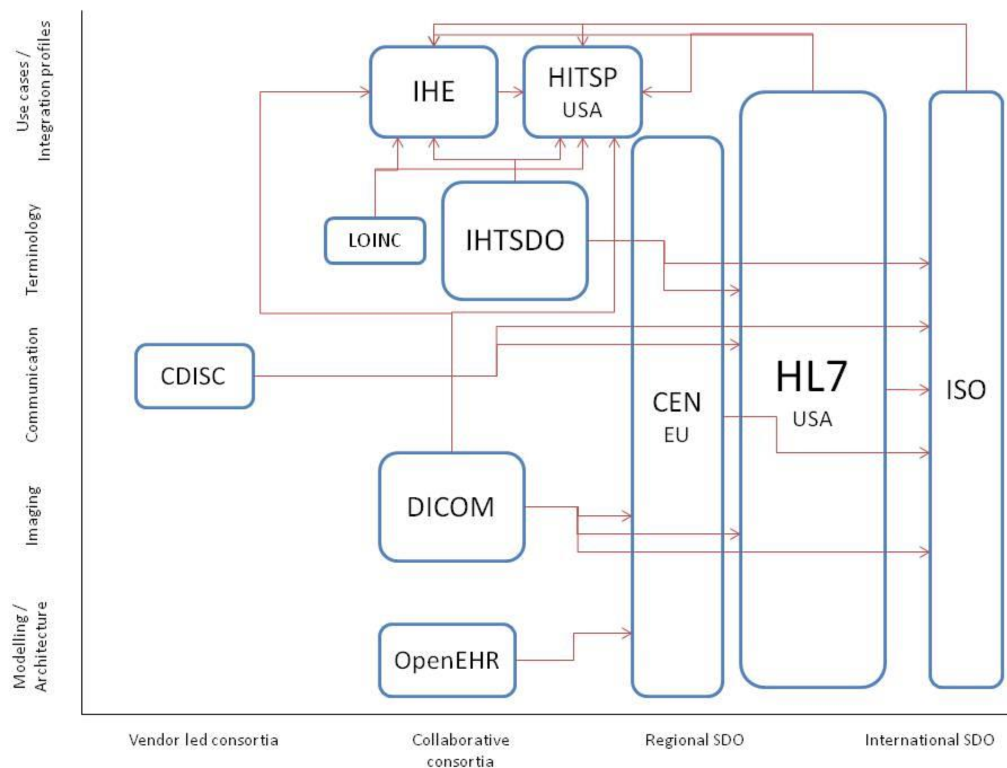
Klasa se sastoji od niza atributa, i svi atributi modelovani preko klase P_BMM_Property. Ova klasa opisuje opšte karakteristike atributa kao što su ime, da li je atribut obavezan ili ne, i da li se vrednost atributa računa ili ne itd. Takođe, ostavljena je mogućnosti i da se pojedini atributi pojavljuju samo u toku izvršenja programa, a da se ne serijalizuju u repozitorijum za čuvanje podataka (u bazu ili u strukturisani fajl).

Drugi deo stabla služi da definiše tipove podataka za atribute klasa. Osnovna klasa u ovom delu stabla je klasifikator (P_BMM_Classifier). Klasifikator je osnovna klasa za modelovanje tipova podataka, i nju sa jedne strane nasleđuju klase, a sa druge generički parametri klase i tipovi podataka atributa klasa (P_BMM_Type). Na ovaj način je omogućeno da atributi klase, mogu da budu apsolutno bilo kog tipa definisanog u okviru meta modela.

Kao što je pomenuto, osnovna klasa koja opisuje tipove atributa je P_BMM_Type. Ona definiše jedino naziv tipa i nju nasleđuju klase koje opisuju realne tipove: P_BMM_Simple_Type, P_BMM_Simple_Type_Open i P_BMM_Container_Type, kao i klasa P_BMM_Generic_Type koja opisuje generičke atribute. Ovde je izvršena gruba podela na proste i složene tipove i to omogućuje kasnije jednostavniju klasifikaciju atributa u modelu.

Svi ovi tipovi služe da bi se definisali stvarni atributi klasa. Oni se definišu posebnim klasama koje nasleđuju P_BMM_Property i referenciraju odgovarajući tip atributa. Tako na primer, klasa P_BMM_Single_Property nasleđuje P_BMM_Property i referencira P_BMM_Simple_Type.

Meta model podataka za medicinske informacione sisteme definisan je na osnovu specifičnih domenskih zahteva. Svaki domen implementacije informacionih sistema karakteriše se određenim specifičnostima vezanim za definisanje poslovnih procesa. Sem toga, tokovi i tipovi podataka, kao i pravila za njihovu agregaciju i prezentaciju mogu biti prilično domenski specifični. Kao osnova za razvoj meta modela koji se može koristiti u razvoju aplikacija za zdravstveni sistem u Republici Srbiji, korišćen je domenski model openEHR inicijative kao i dodatnih zahteva koji su došli iz našeg zdravstva kroz proces razvoja medicinskih informacionih sistema.



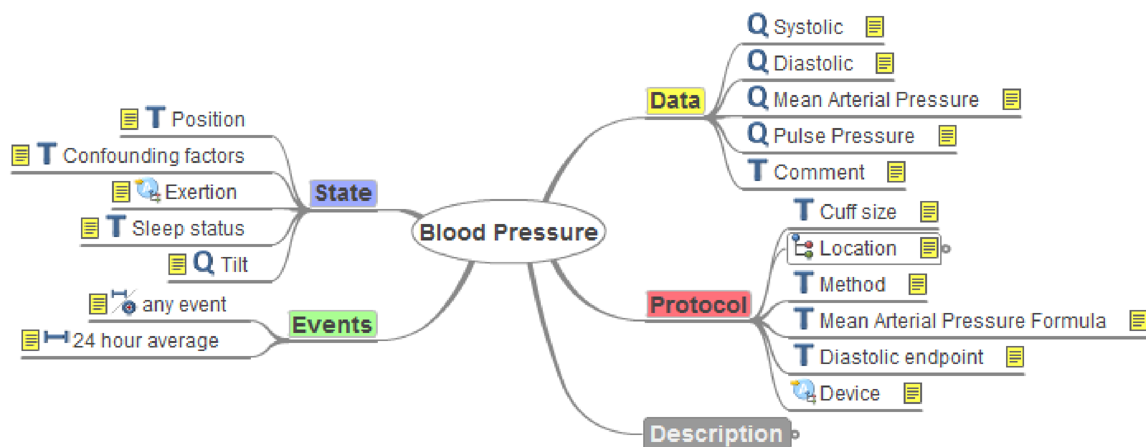
Slika 11 Pozicija openEHR-a na dijagramu standarda u medicinskoj informatici [61]

Slika 11, preuzeta sa [61], pokazuje poziciju openEHR standarda u okviru opšte standardizacione mape medicinske informatike. OpenEHR predstavlja otvoreni standard definisan od strane konzorcijuma koji čine više univerziteta i firmi iz celog sveta i većim delom je posvećen modelovanju i arhitekturnom projektovanju sistema. Sem toga, opisuje i procedure za upravljanje, skladištenje, prikupljanje i razmenu elektronskih zdravstvenih kartona (Electronic Health Record – EHR). Što se tiče organizacije podataka, openEHR je baziran na jedinstvenom kartonu definisanom po pacijentu. Za

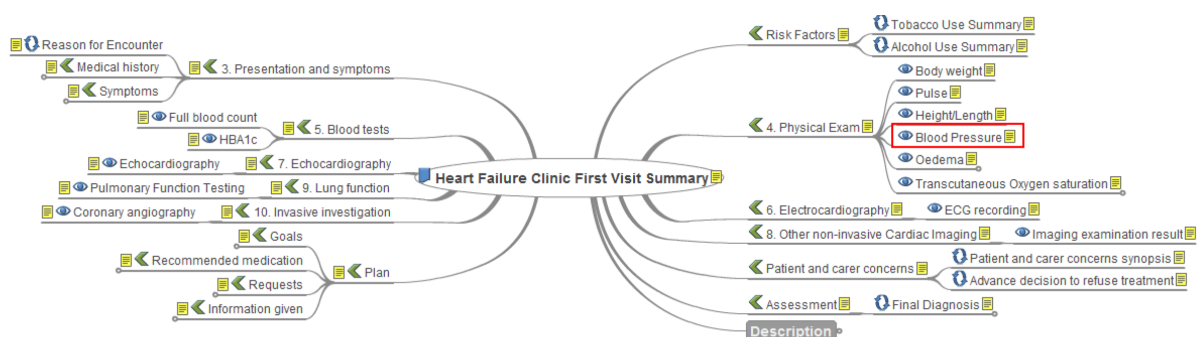
razliku od starijih i rasprostranjenijih EN 13606 [62] i HL7 [63] standarda koji su okrenuti razmeni podataka, openEHR je primarno fokusiran na strukturni model podataka.

Specificiranje openEHR-a je plod rada ljudi iz celog sveta, ali prvenstveno je baziran na iskustvima iz Evrope i Australije. Specifikacija meta modela openEHR standarda se bazira na pet glavnih segmenta u koje su grupisani artefakti modela. Pomenuti glavni segmenti su EHR, demografski podaci, klinički putevi, arhetipi i šabloni. Svi ovi segmenti su dizajnirani tako da zadovolje sve neophodne medicinske i pravne standardne koji se zahtevaju od jednog distribuiranog medicinskog informacionog sistema. Takođe, meta model je kreiran na dovoljno opštim principima da se jednostavno može prilagoditi i proširiti specifičnim zahtevima jednog zdravstvenog sistema.

EHR predstavlja elektronski karton pacijenta definisan tako da bude jedan jedinstveni karton koji će pratiti pacijenta celog života. EHR je povezan sa demografskim podacima, i neophodno je preslikati podatke o pacijentu na karton kako bi se uspostavila veza. Ovaj koncept omogućava jednostavnu inpersonalizaciju zdravstvenih kartona za potrebe statističkih analiza i istraživanja.



Slika 12 Podaci i događaji vezani za merenje krvnog pritiska (dijagram je preuzet iz aplikacije openEHR Clinical Knowledge Manager [64])



Slika 13 Primer upotrebe artefakta u kreiranju većeg medicinskog dokumenta. Arhetip za krvni pritisak (označen crveno) učestvuje u pregledu nakon srčanog udara (dijagram je preuzet iz aplikacije openEHR Clinical Knowledge Manager)

Sadržaj medicinskih procedura je definisan kroz koncept arhetipa koji su zamišljeni kao mesto gde će biti definisana komponenta koja će se kasnije koristiti na mnogo mesta. Arhetipi predstavljaju gradivne elemente većih artefakata i moguće ih je kombinovati kroz model. Na primer, arhetip koji se

često navodi kao primer je "systemic arterial blood pressure measurement". On definiše podatke koji se prikupljaju prilikom merenja krvnog pritiska (Slika 12). Ovaj arhetip se kasnije koristi za kreiranje modela mnogih različitih pregleda, kao što su na primer pregled nakon srčanog udara (Slika 13), alergološki pregledi ili analiza funkcije jetre. Arhetipi su uglavnom jednostavni, sa par različitih podataka. Glavno pravilo koje se primenjuje na arhetipe je to da oni treba da budu uzajamno isključivi. U zavisnosti od pravila za definisanje modela, ta isključivost može da se definiše na dva nivoa:

- Ne postoje dva arhetipa sa potpuno istim skupom elemenata, ili
- Svi arhetipi imaju potpuno disjuktne skupove elemenata

Svi verifikovani arhetipovi su javno dostupni kroz aplikaciju openEHR Clinical Knowledge Manager i čine takozvanu osnovnu biblioteku. Prilikom projektovanja pojedinačnih modela, mora se ipak paziti da se dodati arhetipi uklope u skupove postojećih, bez obzira koje je pravilo odabrano. Ovo se inače u praksi pokazalo kao veći problem za članove medicinskog osoblja i na izvesni način umanjilo opštu upotrebljivost alata za modelovanje sistema.

Druga vrsta artefakta za opis medicinskih podataka je nazvana šablon (template) i koristi se da opiše jedan specifični skup podataka za određeni medicinski slučaj upotrebe – pregled, analizu, terapeutski tretman. Šablon se generiše tako što referencira sve elemente iz pridruženih arhetipova. Opciono, prilikom uključivanja jednog arhetipa, mogu se isključiti neki njegovi elementi i pri tome se ne narušava generisana šema.

Preslikano na jezik objektno orijentisanog programiranja, šablon predstavlja klasu a rezultat svakog od pregleda instancu te klase. Pristup baziran na arhetipima omogućava kasnije lakšu pretragu po podacima po ugledu na refleksiju tipova iz objektnih jezika. OpenEHR za tu svrhu nudi i posebnu sintaksu za pretraživanje, koja se naziva AQL (Archetype Querying Language).

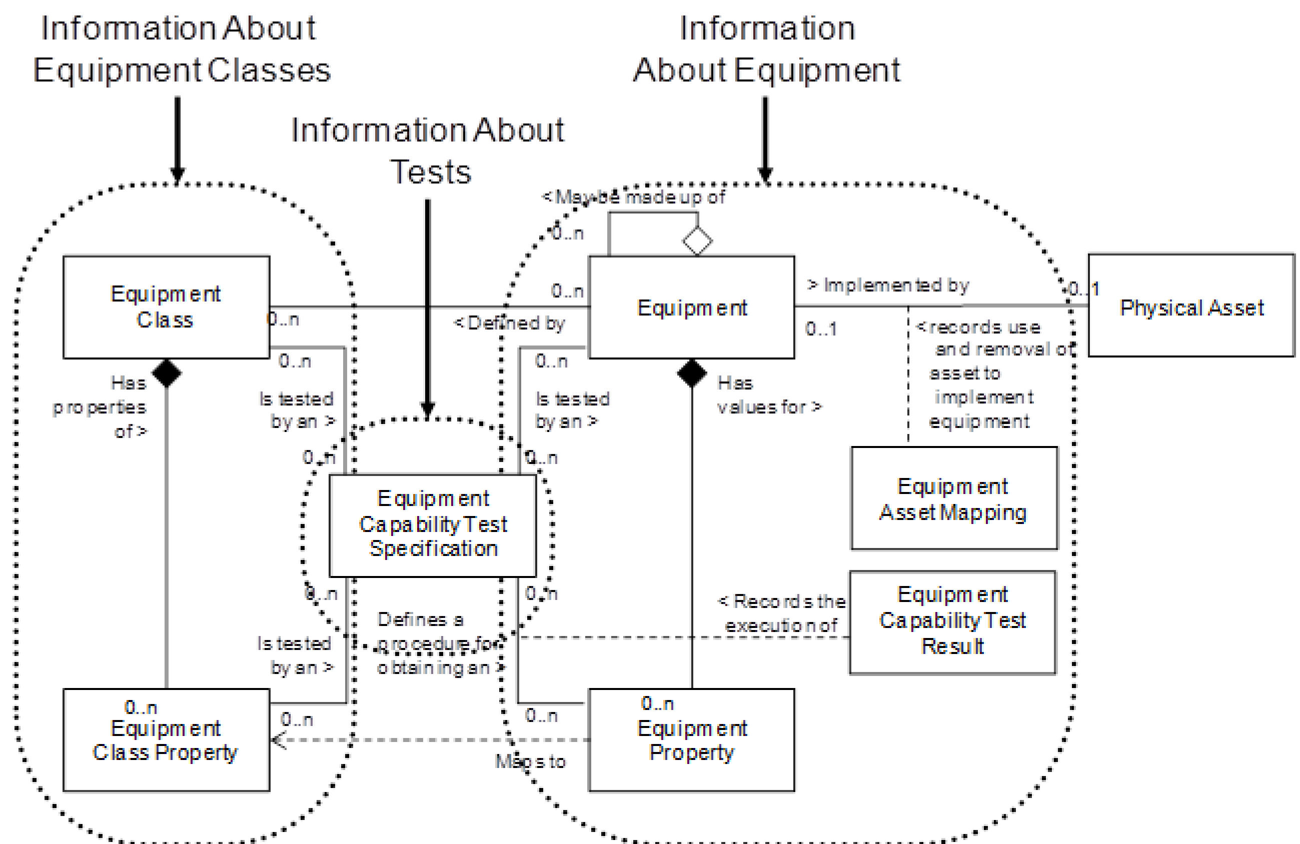
OpenEHR skup modela je u skladu sa novim standardom za EHR (EN 13606) i za sada je iskorišćen u punoj meri kao osnova za informacione sisteme u državnim institucijama u nekoliko zemalja (Velika Britanija, Švedska, Danska, Slovačka, Čile i Brazil). Takođe, komercijalni programi u velikoj meri koriste koncepte openEHR-a u mnogo više zemalja.

Na osnovu ovog meta modela se vidi da je prvenstveno zamišljen da podrži jednostavno definisanje modela podataka koji će sadržati veliki broj različitih tipova podataka i klasa. Ovo je posledica prirode samog domena za koji je ovaj meta model definisan. Generalno, meta modeli mogu da budu fokusirani tako da omogućuju jednostavnu proširivost strukture, ili da omoguće jednostavnije definisanje akcija. Domen medicinskih informacionih sistema je karakterističan po tome da zahteva podršku za veliki broj različitih vrsta dokumenata – od medicinskih, preko laboratorijskih i onih vezanih za osiguranje , pa sve do administrativnih. Druga vrsta meta modela može da ide u pravcu definisanja striktnije strukture podataka, ali zato da ima akcenat na definisanju domenskih akcija kroz proces razvoja modela. Takav vid modela je karakterističan za modele proizvodnih postrojenja koja se razvijaju oko ISA-95 standarda [47] i podržana su posebnim skupom XML šema za modelovanje koje su deo B2MML standarda [65].

2.4 ISA – 95 i B2MML – meta modeli fokusirani na modelovanje akcija

Druga kategorija meta modela su oni usmereni na definisanje akcija, dok je struktura mnogo čvršće definisana i glavni elementi sistema su u mnogo većoj meri definisani. Kod ovog meta modela akcenat je na definisanju tokova podataka i procesa (workflow) i definisanju akcija (events).

Slika 14 predstavlja deo meta modela za proizvodne sisteme koji se odnosi na opis opreme koja se koristi u proizvodnji. Entitet najvišeg nivoa apstrakcije je EquipmentClass. On opisuje vrstu opreme (npr. pumpa) i prepoznaje dve kategorije opreme – fizičke i logičke. Kada se govori o vrstama opreme, u fizičku opremu spadaju sve mašine i njihovi sklopovi, koji se mogu tretirati nezavisno u okviru proizvodnog sistema i za koje se mogu definisati i određeni atributi. Logička oprema su svi oni entiteti koji se dodatno uvode u cilju obrade podataka, npr. zahtevi i odgovori, ili upiti nad bazom.

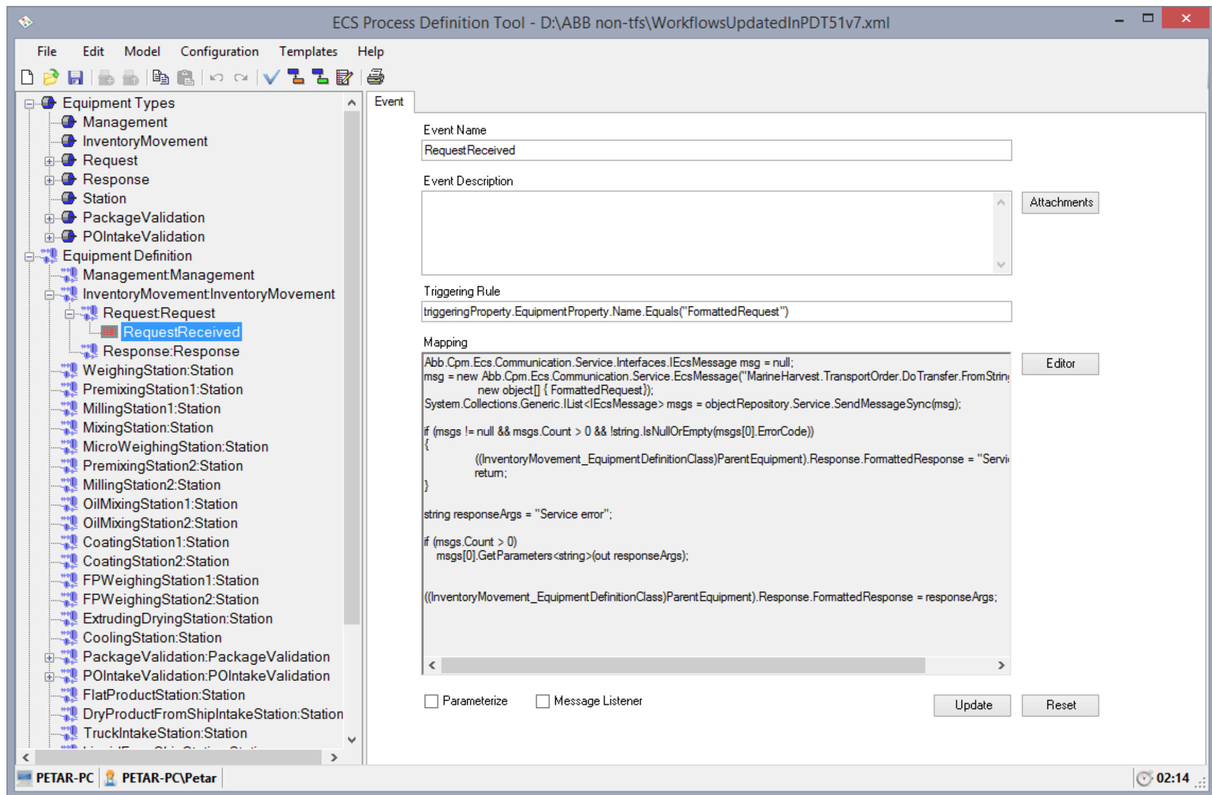


Slika 14 Šema za definisanje vrsta i pojedinih instanci opreme u okviru proizvodnje (preuzeta sa MESA sajta [65])

Kao i kod openEHR modela, i ovde se opis jedne klase definiše skupom njenih atributa. I tu se ceo opis osnovne strukture završava. Ovde nemamo kao kod openEHR-a složene tipove i razgranato stablo nasleđivanja. Što se tiče podržanih tipova podataka za atribute klase (EquipmentClassProperty) tu se nalaze standardni tipovi (celi brojevi, razlomljeni brojevi, stringovi, logički podaci, datum/vreme) i selekcija opštih tipova koji su, u opštem slučaju, dovoljni za opis atributa. Tu spadaju:

- Dataset – skup podataka koje vraća select upit nad bazom podataka
- Object – objekat opšteg tipa, analogon tipu System.Object iz .NET okruženja
- XML – strukturisani podataka

Na osnovu klasa opreme definišu se konkretne vrste opreme (EquipmentDefinition). One, uslovno rečeno, nasleđuju tipove opreme i preuzimaju od njih sve definisane atribute. Na osnovu jednog tipa opreme može se izvesti više definicija. U primeru (Slika 15) na osnovu tipa Station koji modeluje radnu stanicu za registrovanje utrošenog materijala, definisano je nekoliko definicija opreme: WeighingStation, MillingStation, CoolingStation itd. Za razliku od klasa, jedna definicija opreme može da sadrži niz poddefinicija.



Slika 15 Primer generisanog modela proizvodnje kroz alat ABB ECS Process Definition Tool (PDT) [66]

Po ISA standardu, sve akcije se definišu nad definicijama opreme. Tako na primer (Slika 15) definicija InventoryManagement sadrži poddefiniciju Request. U okviru te definicije Request definisana je akcija, event po ISA terminologiji, nazvana RequestReceived. Ovde se specificira kako će ta akcija da izgleda (polje Mapping, Slika 15). Kao najopštije rešenje, preporučuje se korišćenje specifičnih jezika za generisanje akcija, ali ovde je primenjen malo drugačiji pristup – kod koji opisuje akciju se odmah piše u odredišnom programskom jeziku.

Na početku kreiranja modela definiše se programski jezik u kome će sve akcije biti definisane. Ujedno to će biti i jezik po kome će na osnovu modela na kraju biti generisan kod. Po ISA standardu, generisani kod treba da bude asembli koji se može tretirati kao COM komponenta i na taj način se može uključiti u bilo koji programski jezik koji podržava COM princip.

Prednost ovakvog pristupa je u tome što programeri ne treba da uče novi jezik, i što je rešenje u velikoj meri opšte zato što je standard po kome se generiše asembli široko prihvaćen. Kod razvoja proizvodnih sistema, kao i kod razvoja MIS sistema vreme isporuke je jedan od kritičnih uslova, tako da je smanjena fleksibilnost cena kojom se plaća brži razvoj, ovde, itekako prihvatljiva.

2.5 Definisanje specijalizovanog meta modela za zdravstvo

Meta model za primenu u zdravstvu Republike Srbije, nastao je kroz razvoj MIS sistema Medis.NET. Kao polazna osnova uzet je openEHR model, što znači da je fokus u kreiranju meta modela bio na lakšem strukturnom proširenju. Deo modelovanja koji se odnosi na definisanje akcija pomeren je u fazu generisanja koda. Ovde je iskorišćeno iskustvo iz razvoja proizvodnih sistema i preuzet je koncept iz ISA standarda da se same akcije definišu u odredišnom programskom jeziku. U okviru meta modela koji se koriste za razvoj MIS sistema postoji mnogo različitih jezika za definisanje akcija, ali za njih ipak ne postoji opšti standard.

Pomeranjem definisanja akcija u kasniju fazu, proces modelovanja se svodi samo na razvoj strukturnog modela, što za posledicu ima kreiranje jednostavnijeg alata za modelovanje koji je u mnogome pristupačniji krajnjim korisnicima. Koncept sa definisanjem akcija u okviru modela, kao kod ISA alata, je izvor najvećih problema u komunikaciji sa krajnjim korisnicama. Posebno kada žele da im se objasni kako kod radi, a pri tome nemaju ni osnovnog programerskog znanja.

U konkretnom slučaju, upotreba posebnog jezika za definisanje akcija u meta modelu daje veću fleksibilnost celokupnom sistemu, ali po cenu kreiranja pravila za preslikavanje, odnosno prevođenje akcija iz jezika za modelovanje u programski jezik u kome se razvija MIS. U zavisnosti od vrste odredišnog jezika i same strukture akcija to može da predstavlja suviše dugotrajan proces koji rezultuje usporavanjem celog procesa razvoja.

Početna tačka u razvoju meta modela za Medis.Net je, ipak openEHR model. Može se reći da je struktura openEHR modela proširena i specijalizovana da bi pratila naše zdravstvo, ali uz tendenciju da svi novouvedeni koncepti ne naruše postignuti stepen generalnosti.

Sve klase u ovom meta modelu (Slika 16) su izvedene iz P_BMM_Class, dok su kroz posebno preslikavanje tipovi podataka iz razvojnog okruženja .NET povezani u model. Oni su povezani na P_BMM_Type. Ovo je urađeno kako bi kreirani model bio openEHR kompatibilan i kako bi mogao da se primeni u drugim zdravstvenim sistemima.

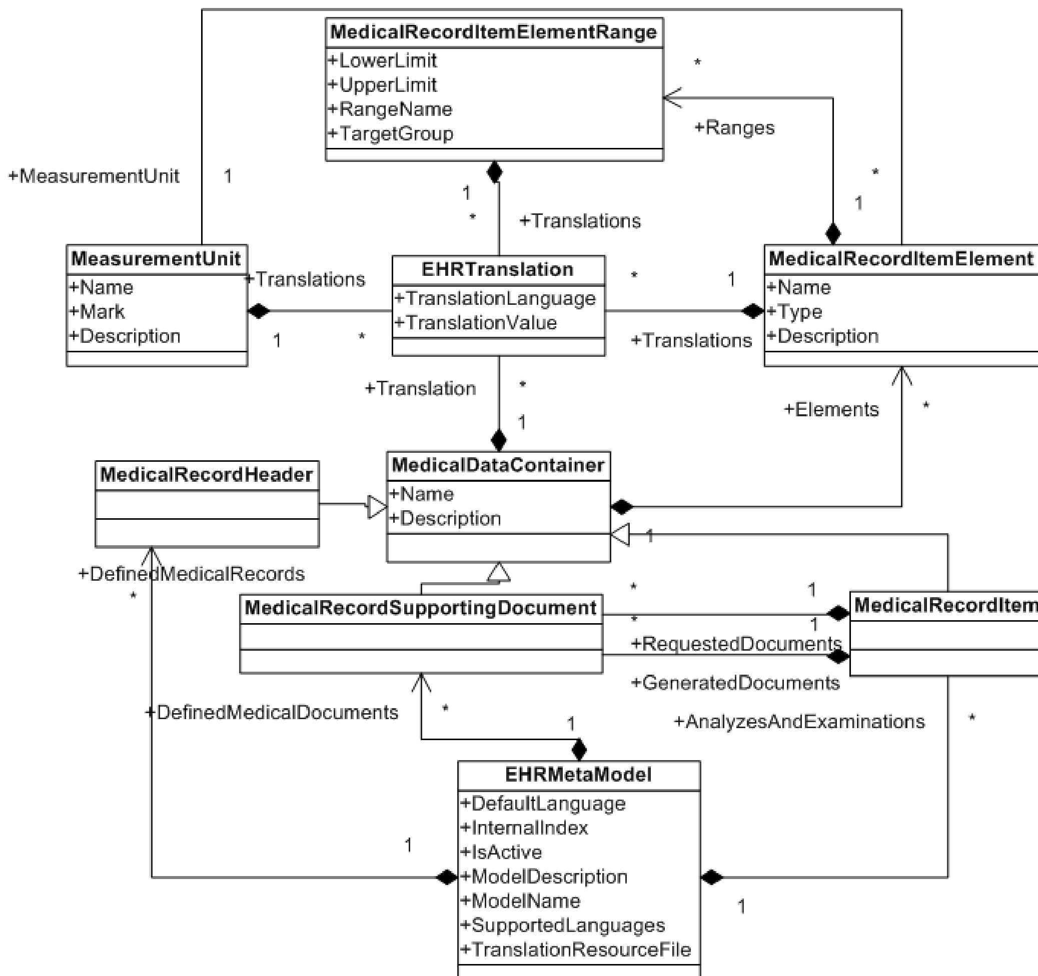
U ovom modelu je uveden i novi, vršni, entitet koji je nazvan EHRMetaModel. On predstavlja kontejner koji sadrži sve tipove definisane u okviru modela. On takođe, sadrži i posebno definisane specijalizovane nizove najkarakterističnijih grupa entiteta koji se definišu u sistemu. To su nizovi medicinskih dokumenata, pregleda i kartona. Klasa EHRMetaModel sadrži sve neophodne opšte atribute koji su neophodni za njegovo bliže definisanje. To su ime modela, opis, i polja potrebna za višejezičnu podršku. Ovo predstavlja proširenje u odnosu na standard definisan openEHR-om. Sam model sadrži listu podržanih jezika, referencu na osnovni jezik model i putanju do fajla u kome će se na kraju čuvati generisani prevodi. Kada se definiše osnovni jezik, svi nazivi svih elemenata modela (atribut Name u svakoj od njihovih klasa) se podrazumevaju da su u osnovnom jeziku. Način na koji se prevodi definišu biće dat u narednim poglavljima.

Kako je meta model osnova za razvoj alata za modelovanje, njegova struktura odslikava kako zahteve krajnjih korisnika tako i zahteve programera za odgovarajućim sadržajem. EHRMetaModel sadrži listu entiteta tipa MedicalDataContainer, i to je glavna kolekcija elemenata koja se nalazi u okviru modela.

Klasa MedicalDataContainer je osnovni entitet koji služi da opiše bilo kakav tip podataka koji će biti definisan kroz meta model. Sve klase koje opisuju posebne zdravstvene kartone, posebne preglede i medicinske dokumente nasleđuju ovu klasu. Ovim entitetima se može pristupiti kao posebnoj listi koja je atribut klase EHRMetaModel, ali i kroz tri posebna atributa koji čuvaju reference zdravstvene kartone, preglede i dokumente. Ovo je obezbeđeno kako bi se ceo koncept modelovanja približio

krajnjim korisnicima i kako bi mogli da unose elemente modela na način kako vide svoj poslovni proces.

Klasa MedicalDataContainer, kao svoj glavni deo, ima listu objekata tipa MedicalRecordItemElement. Ovom klasom se opisuju pojedinačni atributi svakog od elemenata modela. Objektima ove klase definišu se ime atributa, opis i tip podataka. Ova klasa se preslikava direktno na P_BMM_Property. MedicalRecordItemElement je modelovan tako da sadrži liste standardnih vrednosti. Na ovaj način se jednostavno uključuju sve moguće enumeracije u sistem i obezbeđuje se da sadrži i listu standardnih vrednosti za atribut (lista atributa tipa MedicalRecordItemElementRange).



Slika 16 Meta model za Medis.NET

MedicalRecordItemElementRange je zamišljena da može da se koristi da opiše i nabrojive tipove i opsege. Za definisanje opsega tu su atributi LowerLimit i UpperLimit koji služe za definisanje donje i gornje granice opsega. Polje RangeName, kome se može pristupiti i preko atributa Name, kako bi se obezbedio prevod, sadrži naziv opsega koji se definiše. Atribut TargetGroup služi da dodatno opiše opseg tako što mu dodeljuje opis grupe pacijenata za koje se opseg primenjuje.

Kada se koristi za nabrojive tipove, postoje dve moguće primene. Prva je da se tip objekta podesi kao ENUM, a drugi je da se bez definicije posebnog tipa, polje koje pamti podatke o ciljnoj grupi popuni

vrednošću ENUM. Na ovaj način se izbegava potreba za korišćenjem refleksije nad tipovima podataka u slučaju kada skup vrednosti iz nabrojivog tipa nije vezan ni za jednu ciljnu grupu.

Posledica ovoga je to što će i u generisanom modelu biti podržane iste dve različite vrste standardnih vrednosti – nabrojive (ENUM) i definisane opsegom. Standardne vrednosti definisane opsegom imaće dva zapisa koji se odnose na najnižu i najvišu regularnu vrednost i određenu ciljnu grupu. Nabrojive standardne vrednosti se koriste za definisanje kataloga koji čine skup dozvoljenih vrednosti za neko polje. Takođe, ova klasa sadrži i listu dodatnih elemenata koji se modeluju kroz objekte tipa MedicalRecordItemElementProperty. Sve pomenute klase nasleđuju MedicalRecordItemBase koja ima referencu na niz objekata tipa EHRTranslation koja obezbeđuje višejezičku podršku za svako ime i opis koji se nalaze u okviru različitih objekata.

Još jedan dodatak na osnovni model je i to da klasa MedicalDataContainer, za razliku od definicije klase iz openEHR-a podržava i rekurzivno definisanje složenih objekata. Na ovaj način se jednostavno modeluju složeni pregledi kod kojih nije bitno uređenje podelemenata. Na taj način se eliminiše u većini slučajeva potreba da se definiše specifičan proces (workflow). Atribut SubContainers je ovde zadužen da čuva niz podelemenata istog tipa (MedicalDataContainer).

Specijalizacija nad klasom MedicalDataContainer je inicijalno implementirana kroz tri klase koje opisuju tri glavne grupe entiteta sa tačke gledišta krajnjih korisnika sistema:

- MedicalRecordItem koja je osnova za sve preglede, analize i terapije
- MedicalRecordDocument koja modeluje različite dokumente i izveštaje
- i MedicalRecordHeader kojom se modeluju zdravstveni kartoni.

Za dalje primene moguće je dodatno proširiti ovo stablo nasleđivanja kako bi se meta model dodatno približio domenu korisnika. Posebno je bitno proširenje modela koje se odnosi na definisanje složenih procesa (Slika 17). To proširenje se zasniva na definiciji međusobnog odnosa podelemenata u okviru roditeljskog elementa. Glavni entitet je Workflow koji definiše osnovne metode i atribute za definiciju složenog pregleda. U okviru njega se definiše za koju je vrstu pregleda, a sastoji se od niza koraka (WorkflowStep).

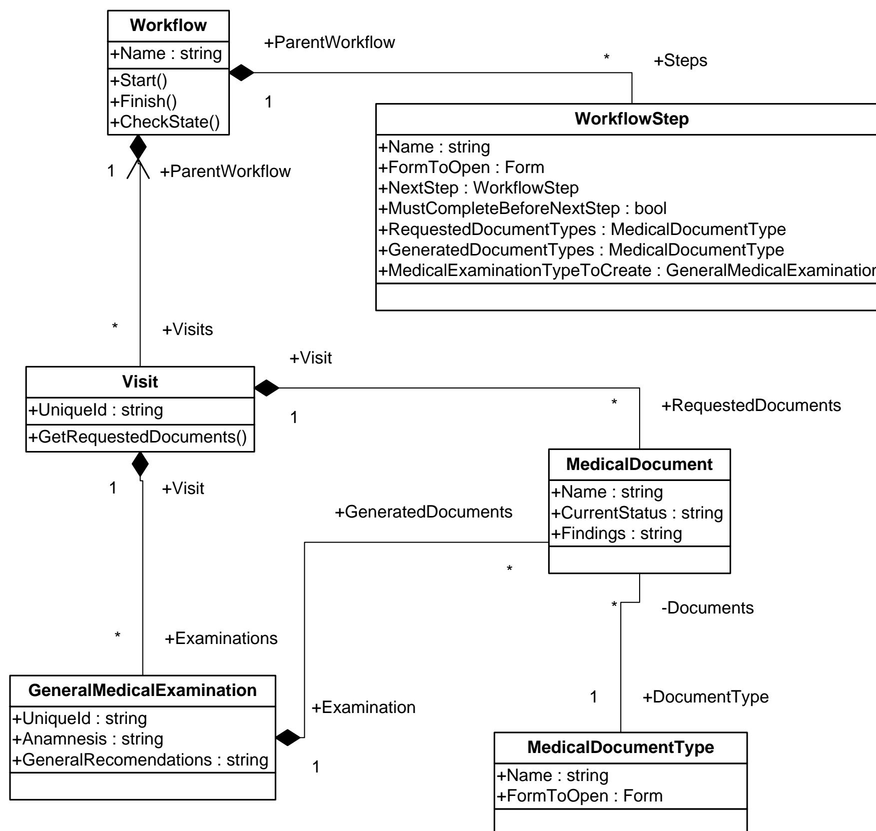
Svaki od koraka ima reference na prethodne i sledeće korake, kao i liste medicinskih dokumenta koji su neophodni da bi pregled počeo i liste dokumenata koji se generišu na kraju pregleda. Opciono, ove liste mogu da budu prazne.

Pravila za definisanje tokova pregleda su identična pravilima za generisanje flowchart dijagrama iz UML-a [67]. Svaki workflow mora da ima:

- jednu početnu tačku
- makar jednu krajnju tačku
- svaka putanja kroz graf mora da ima mogućnost izlaza do neke od krajnjih tačaka
- moguće je kreirati petlje, ali iz svake od njih mora da postoje precizno definisani uslovi za izlaz.

Svaki proces se vezuje za pojedinačnu posetu. Entitet koji modeluje posetu (Visit) izveden je iz opšteg MedicalDataContainer tipa i obogaćen odgovarajućim atributima u kojima se pamte neophodni administrativni podaci.

Tok procesa (Workfow) se definiše globalno za svaki od tipova pregleda, a onda se kopira uz konkretni pregled. Na taj način se za svaki složeni pregled pojedinačno može jednoznačno utvrditi kroz koje grane se prošlo i kojim redom, što omogućuje kasniju optimizaciju procesa.



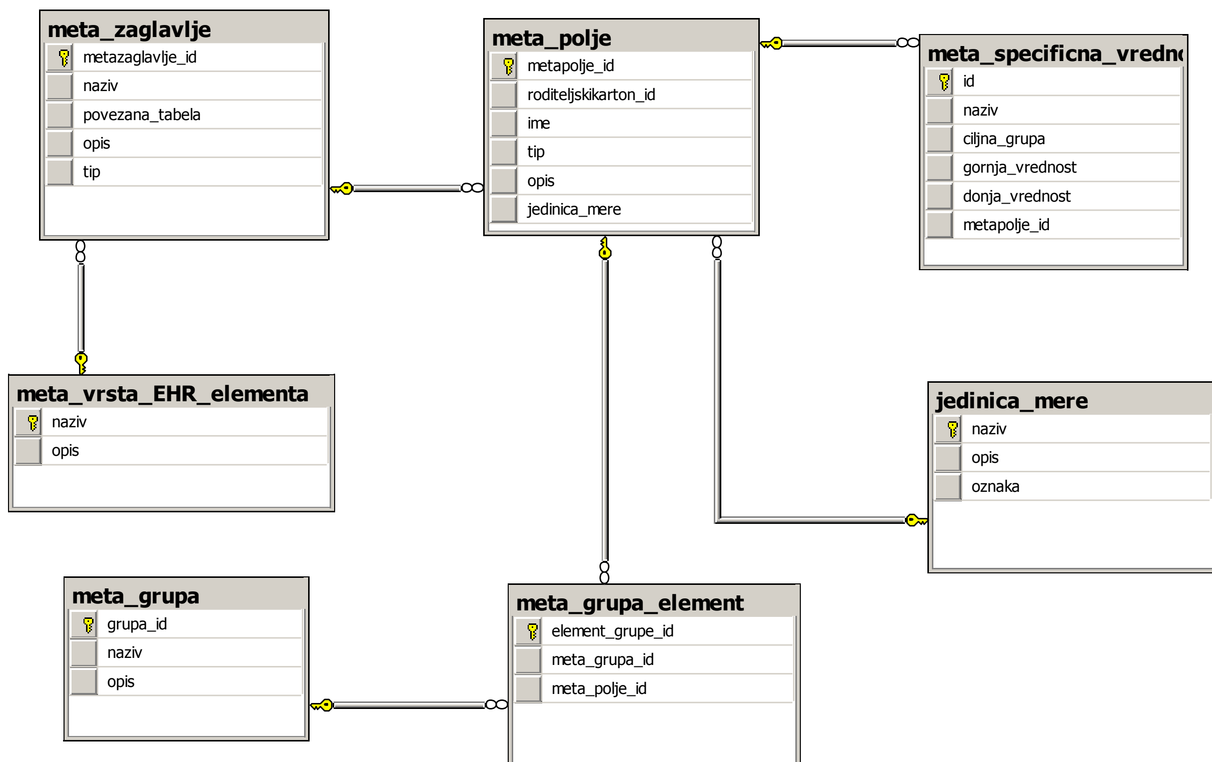
Slika 17 Proširenje meta modela za složene preglede

Definisani meta model ima jednu glavnu svrhu, a to je razvoj posebnih domenskih modela. Razvoj posebnih modela se bazira na elektronskom zdravstvenom kartonu razvijenom za Medis.NET i internoj organizaciji zdravstva. Najpre se definišu različiti tipovi zdravstvenih kartona sa pripadajućim atributima. Kod nas jedan pacijent može da ima više zdravstvenih kartona, a kartoni se vode po različitim odeljenjima i službama u okviru zdravstvenih institucija. U pomenutom EHR-u, svi oni su vezani za jedan karton koji se proglašava za „glavni". Glavni karton je najčešće onaj koji se čuva kod pacijentovog izabranog lekara opšte prakse i kroz njega je dostupna puna istorija bolesti pacijenta. Ostali kartoni mogu, u zavisnosti od podešenih privilegija, pristupati samo podacima smeštenim u okviru njih ili i podacima iz drugih kartona. Sistem takođe podržava i scenario kada jedan pacijent ima samo jedan karton i kada više pacijenata deli jedan karton. Drugi pomenuti scenario je za takozvane porodične kartone koji su podržani u mnogim zdravstvenim sistemima.

U modelima kreiranim na osnovu predstavljenog meta modela, klasa za definisanje kartona MedicalRecordHeader se preslikava u runtime klasu BaseMedicalRecord i svi definisani kartoni se u modelu predstavljaju kao klase izvedene iz BaseMedicalRecord. BaseMedicalRecord sadrži niz referenci na objekte čije klase nasleđuju tip BaseMedicalRecordItem. BaseMedicalRecordItem je osnovna klasa za opis bilo kakvog pregleda, terapije ili analize u generisanom modelu i mapira se na MedicalRecordItem iz modela. Svaki od pomenutih elemenata se slika u posebnu klasu koja nasleđuje BaseMedicalRecordItem.

Ako se pogleda veza sa osnovnim entitetima iz openEHR-a, MedicalRecordItem predstavlja vrstu šablona i samim tim i jedno od glavnih mesta proširenja sistema. Što se tiče arhetipa, oni u modelu za Medis.NET ne igraju značajnu strukturnu ulogu. U predstavljenom modelu jedan šablon se sastoji od niza direktnih elemenata u svom opisu, i samim tim ne postoji dodatni nivo između njih. Na ovaj način izbegnuta je konfuzija koju arhetipi proizvode za medicinske radnike koji treba da učestvuju u modelovanju sistema.

Arhetipi su uvedeni samo kao logičke organizacione grupe, i posebnom evaluacijom se nakon instancijacije modela kreiraju kroz posebnu strukturu u bazi (Slika 18). Na ovaj način se omogućuje da sistem ostane u saglasnosti sa openEHR-om i da AQL upiti mogu da se koriste.



Slika 18 Deo baze podataka u kome se čuvaju EHR meta podaci

U lokalizovanom modelu, arhetipovima odgovaraju entiteti koji se zovu meta_grupa. Meta podaci o pojedinačnim elementima koji ulaze u različite preglede smešteni su u tabeli meta_polje. S obzirom da se jedno polje može naći u više arhetipa, veza između te dve tabele je data kroz tabelu spoja meta_grupa_element. Dodatnom logikom je obezbeđeno da ne postoje dva potpuno ista arhetipa, odnosno grupe elemenata. Veza definisanih polja sa glavnim delom sistema ostvarena je kroz tabelu meta_zaglavlje. Tabela meta_zaglavlje služi da čuva opšte podatke o osnovnim elementima koji ulaze u skup arhetipa i predstavljaju polja u tabelama koje opisuju preglede.. Ona čuva jedinstveni identifikator definicije, naziv definisanog elementa, njegov opis, i referencu ka povezanoj tabeli, kao i podvrstu elementa. Podvrste elemenata su definisane u tabeli meta_vrsta_EHR elementa. Tu je sačuvan naziv tipa i njegov opis.

2.6 Alat za kreiranje modela podataka za informacione sisteme

Definisani meta modeli služe kao osnova za kreiranje domenskih modela podataka. Kroz kreiranje domenskih modela, primarno se kreiraju se novi tipovi podataka napravljenih po ugledu na osnovne klase definisane u meta-modelu. Slika 15 prikazuje alat za modelovanje opreme koja čini proizvodni sistem. U okviru meta modela definisani su opisi za tipove i instance opreme, a u okviru alata za modelovanje kreiraju se opisi stvarnih delova opreme sa odgovarajućim atributima i akcijama.

Ovo je inače glavna ideja koja stoji iza alata za kreiranje domenskih modela. Alati za kreiranje domenskih modela ili alati za modelovanje, kako se još često nazivaju, su aplikacije koje kao ulaze imaju definisane meta modele i kolekcije podataka koje korisnici unesu. Rezultat aplikacije je domenski model koji se može čuvati u obliku strukturisanog XML fajla ili držati u odgovarajućoj strukturi u okviru baze podataka.

U ovom odeljku biće prikazana aplikacija za kreiranje domenskih modela baziranih na meta modelu za MIS sisteme. Ono što je glavni rezultat ovih domenskih modela su nizovi instanci tipa MedicalDataContainer koji predstavljaju okvir za opis svih pregleda, laboratorijskih analiza i ostalih medicinskih dokumenata. S obzirom, da meta model podržava kompozitnu rekurzivnost nad objektima tipa MedicalDataContainer, i da je proširen i strukturom za definisanje toka podataka (workflow), kroz alat za modelovanje se mogu definisati kako prosti tako i složeni entiteti.

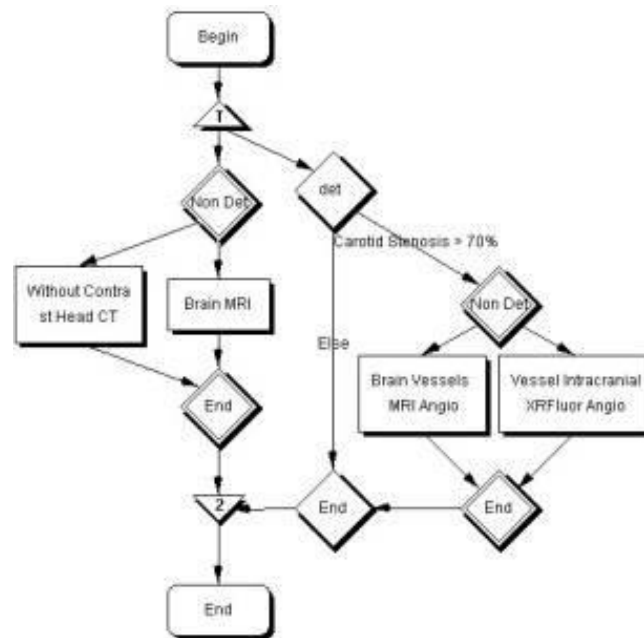
Prost entitet je MedicalDataContainer koji ima definisan samo skup atributa, i nema definisane podelemente. Podelementi su takođe tipa MedicalDataContainer, ili nekog od izvedenih tipova. Složeni entiteti se mogu definisati na dva načina – sa nizom podelemenata koji može biti uređen ili neuređen, ili sa podelementima strukturiranim kao graf (workflow).

Slika 19 Skup uređenih koraka u okviru jednog pregleda

Primer složenog entiteta sa nestrukturisanim podelementima je standardni sistematski pregled. On se najčešće sastoji od nekoliko manjih pregleda kod lekara različitih specijalnosti gde svi oni moraju da se obave, ali nebitno u kom redosledu. Slika 19 prikazuje skup uređenih podelemenata u okviru jednog

pregleda. Ovde su elementi standardnog pregleda kod lekara opšte prakse definisani kao sukcesivni koraci, gde sledeći korak postaje dostupan tek kada se prethodni obavi. Tako se ovde najpre unose podaci vezani za anamnezu, pa zatim opšti medicinski podaci, radne dijagnoze, terapije, uputi, podaci o datim uslugama, propisanim pomagalima i završno sa administrativnim podacima.

Složeni pregledi se mogu definisati i specifičnim tokom pregleda. Slika 20 prikazuje korake kod pregleda za pacijente koji su preživali moždani udar. Dijagram je preuzet sa sajta Open clicnical gde su dostupni mnogi medicinski resursi kao open source dokumenti.



Slika 20 Primer koraka u složenom pregledu kod moždanog udara (primer preuzet sa http://www.openclinical.org/gmm_guide.html)

Glavna uloga alata za modelovanje je kreiranje novih MedicalRecordContainer tipova. Kao što je već pomenuto oni su grupisani u logičke grupe što omogućava korisnicima alata da ih lakše kategorizuju. Kategorije se mogu generički definisati a standardnim meta modelom su predviđene tri – kartoni, pregledi i dokumenti. Sve te kategorije se, za alat za modelovanje, mogu definisati kroz konfiguraciju alata na način kao što to prikazuje Slika 21.

Ovde se dodatno prati i koncept iz openEHR modela o tačkama proširenja sistema. Svaka kategorija entiteta se vezuje za jednu tačku proširenja. Te tačke se u konkretnom slučaju definišu kao tabele u bazi koje će imati ulogu baznog entiteta za sve definisane MedicalDataContainer-e. Najčešći slučaj je da jedna klasa ima jednu tačku proširenja, ali je moguće definisati sistem tako da više kategorija bude vezano na istu tačku proširenja.

Kroz konfiguraciju se zapravo definiše mapiranje između tabela koje predstavljaju tačke proširenja u bazi i odgovarajućih objekata iz meta modela. Kasnije, kroz korisnički interfejs, ta mapiranja se mogu promeniti. Slika 21 prikazuje primer konfiguracije alata za modelovanje koja sadrži dva glavna bloka definiciju parametara za povezivanje na bazu i definiciju tačaka proširenja kroz opis različitih kategorija MedicalDataContainer dokumenata.

Prvi deo konfiguracije, sekcija DestinationDBSettings čuva podatke neophodne za povezivanje na bazu podataka – ime servera, baze i podatke neophodne za logovanje. Veza sa bazom nije neophodna

za sam proces definisanja domenskih modela, ali se koristi kod reverznog inženjeringa modela i kao ulaz za alat generisanje komponenti koji može da deli konfiguraciju sa alatom za modelovanje. O ovome će biti reči u narednim poglavljima.

```
  <?xml version="1.0" encoding="UTF-16"?>
- <ModelingToolSettings xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <DestinationDBSettings>
      <ServerName>m1server</ServerName>
      <DatabaseName>zdravstvo</DatabaseName>
      <FailoverPartner/>
      <IntegratedSecurity>true</IntegratedSecurity>
      <UserName/>
      <Password/>
    </DestinationDBSettings>
  - <BaseMedicalDataContainers>
    - <MedicalDataContainer>
        <Name>BaseEHRRecordTableName</Name>
        <Description>Medical Record Base Table</Description>
        <ExtensionPoint>karton</ExtensionPoint>
        <ExtensionRules/>
      </MedicalDataContainer>
    - <MedicalDataContainer>
        <Name>BaseEHRItemTableName</Name>
        <Description>Medical Record Item Base Table</Description>
        <ExtensionPoint>data_usluga</ExtensionPoint>
        <ExtensionRules/>
      </MedicalDataContainer>
    </BaseMedicalDataContainers>
  </ModelingToolSettings>
```

Slika 21 Primer konfiguracije alata za modelovanje medicinskih informacionih sistema

Ono što je ovde značajno za sam proces modelovanja je sekcija BaseMedicalDataContainers. Ona definiše kategorije entiteta koji se modeluju i dodatno definiše preslikavanje tabela koje predstavljaju tačke proširenja za model. U primeru koji prikazuje Slika 21 definisane su dve tačke proširenja – jedna za zdravstvene kartone i druga za preglede. Tačka proširenja za medicinske dokumente nije definisana. Ove tri tačke proširenja su inicijalno podržane, ali korisnik može definisati koliko kod njih hoće. U pomenutom primeru jedna tačka proširenja (sekcija MedicalDataContainer) opisana je pomoću:

- naziva (Name),
- opisa (Description) koji će se pojaviti kasnije na konfiguracionoj formi (Slika 23)
- naziva tabele koja će biti tačka proširenja (ExtensionPoint), odnosno osnova za sve novokreirane entitete
- skupa pravila (ExtensionRules) koja se dodaju svim entitetima koji proizađu iz tačke proširenja.

Za samo modelovanje bitno je navesti naziv nove kategorije, dok su podaci o pravilima za proširenje i osnovnim tabelama u bazi bitne alatu za generisanje koda koji na osnovu modela kasnije generiše različite softverske komponente.

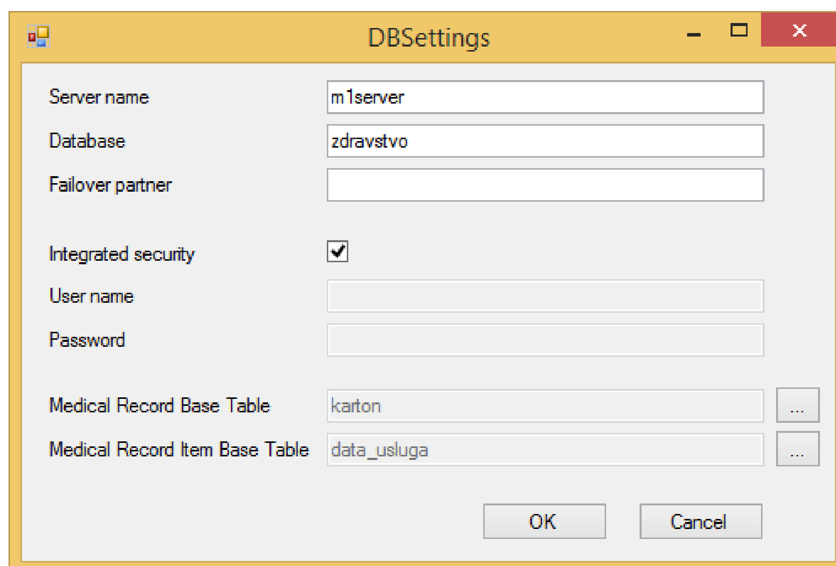
Pravila proširenja su dodatak konfiguraciji koji koristi komponenta za reverzni inženjering, kao i (ExtensionRules) definišu se kao polja i relacije koje će biti dodata u svaku tabelu koja bude generisana na osnovu modela. Svaki novogenerisani entitet naslediće entitet definisan kao tačka proširenja i dodati sva definisana pravila.

Sintaksa za definisanje dodatnih polja pri proširenju preuzeta je iz NHibernate biblioteke koja predstavlja standard za kreiranje modela podataka. Slika 22 prikazuje primer definisanih pravila proširenja.

```
- <MedicalDataContainer>
    <Name>BaseEHRItemTableName</Name>
    <Description>Medical Record Item Base Table</Description>
    <ExtensionPoint>data_usluga</ExtensionPoint>
  - <ExtensionRules>
      <property not-null="true" type="int" name="Priority"/>
      <property type="DateTime" name="ScheduledStart"/>
      <property type="DateTime" name="ScheduledEnd"/>
      <many-to-one name="VerifiedBy" cascade="save-update" column="lekarId" class="lekar"/>
    - <bag name="MaterialItems" cascade="all-delete-orphan" lazy="true" batch-size="5" inverse="true">
        <key column="ParentId"/>
        <one-to-many class="material"/>
      </bag>
    </ExtensionRules>
  </MedicalDataContainer>
```

Slika 22 Primer definisanih pravila za proširenje

Tako je ovde dodato jedno polje tipa int koje se zove Priority i koje mora imati dodeljenu vrednost. Takođe, dodati su prosti atributi za predviđeno vreme početka i kraja pregleda. Sem njih, dodata je referenca na roditeljski entitet tipa lekar (many-to-one tag) i veza ka kolekciji materijala neophodnih za pregled (tag koji se zove bag). Ovi atributi će se iskopirati u generisani model i uključiti u svaku od tabela koju alat za generisanje bude kreirao.



Slika 23 Primer konfiguracionog dijaloga alata za modelovanje

Svi definisani MedicalDataContainer objekti pojaviće se u konfiguracionom dijalogu (Slika 23) i korisnik može da promeni njihovo preslikavanje. Na početku dijaloga je skup fiksnih polja koja se odnose na parametre za povezivanje sa bazom (narandžasti okvir) dok su u donjem delu forme (zeleni okvir) polja koja se pojavljuju na osnovu konfiguracije.

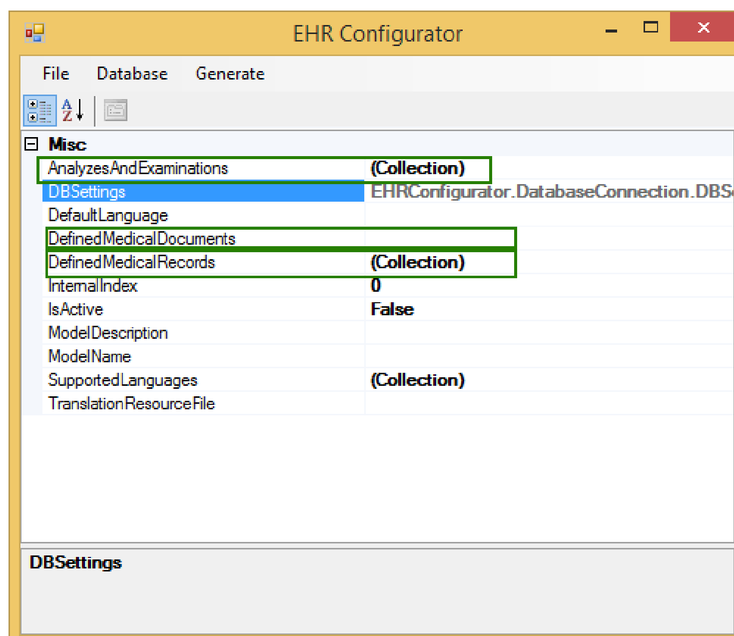
Za svaku kategoriju definisanu u modelu pojaviće se po jedan red koji na početku ima natpis koji odgovara opisu unetom u konfiguraciju, jedan neaktivni tekst boks koji prikazuje naziv osnovne tabele i jedno dugme (na kome stoje tri tačke). Klikom na pomenutom dugme otvara se forma iz koje se definiše tabela koja ima ulogu tačke proširenja kao i pravila proširenja (ExtensionRules).

Sve definisane kategorije MedicalRecordContainer-a pojaviće se u alatu za modelovanje kao posebna kolekcija. Naziv kolekcije (atribut CollectionName, Slika 24) će se pojaviti u okviru alata za

modelovanje (Slika 25). U okviru svake od kolekcija korisnici će definisati entitete po strukturi definisanoj u meta modelu.

```
- <BaseMedicalDataContainers>
  - <MedicalDataContainer>
      <Name>BaseEHRRecordTableName</Name>
      <Description>Medical Record Base Table</Description>
      <ExtensionPoint>karton</ExtensionPoint>
      <CollectionName>DefinedMedicalRecords</CollectionName>
      <ExtensionRules/>
    </MedicalDataContainer>
  - <MedicalDataContainer>
      <Name>BaseEHRDocumentTableName</Name>
      <Description>Medical Document Base Table</Description>
      <ExtensionPoint>dokument</ExtensionPoint>
      <CollectionName>DefinedMedicalDocuments</CollectionName>
      <ExtensionRules/>
    </MedicalDataContainer>
  - <MedicalDataContainer>
      <Name>BaseEHRItemTableName</Name>
      <Description>Medical Record Item Base Table</Description>
      <ExtensionPoint>data_usluga</ExtensionPoint>
      <CollectionName>AnalyzesAndExaminations</CollectionName>
    - <ExtensionRules>
        <property not-null="true" type="int" name="Priority"/>
        <property type="DateTime" name="ScheduledStart"/>
        <property type="DateTime" name="ScheduledEnd"/>
        <many-to-one name="VerifiedBy" cascade="save-update" column="lekarId" class="lekar"/>
      - <bag name="MaterialItems" cascade="all-delete-orphan" lazy="true" batch-size="5" inverse="true">
          <key column="ParentId"/>
          <one-to-many class="material"/>
        </bag>
      </ExtensionRules>
    </MedicalDataContainer>
  </BaseMedicalDataContainers>
```

Slika 24 Konfiguracija sa definisanim nazivom kolekcije entiteta

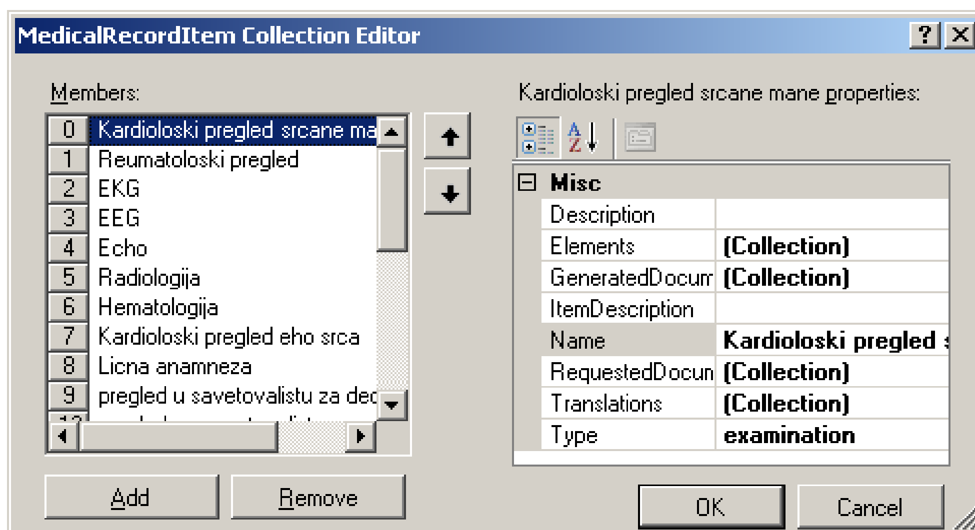


Slika 25 EHR alat za modelovanje – početna strana

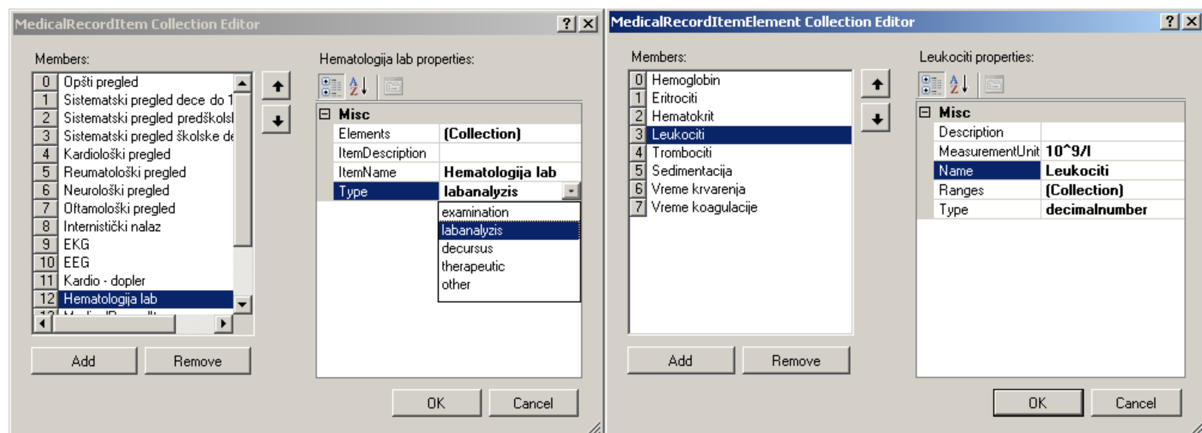
Nakon završetka konfigurisanja, alat se može dati budućim korisnicima da zajedno sa tehničkim osobljem rade na razvoju modela (Slika 26). Ovde je prikazana najjednostavnija verzija

konfiguratorskog alata koja je bazirana na komponenti koja se zove PropertyGrid i koja dolazi iz standardnog skupa vizuelnih komponenti .NET razvojnog okruženja. Ovoj komponenti se dodeljuje objekat čiji će atributi biti prikazani i čiji se sadržaj može menjati. U ovom slučaju to je objekat klase EHRDomainModel koja se dinamički kreira za odgovarajući domenski model na osnovu konfiguracije. Klasa EHRDomainModel nasleđuje vršnu klasu iz meta modela (EHRMetaModel), i uz nasleđene atribute dodaje kolekcije MedicalDataContainer-a definisanih u konfiguraciji. Uslov da se neki objekat može dodeliti kao objekat za editovanje u PropertyGrid-u je da je definisan kao serijabilna klasa.

Sam proces modelovanja se odvija tako što korisnici u okviru svake od konfigurisanih grupa entiteta, dodaju nove entitete sa pripadajućim elementima, opsezima standarnih vrednosti i prevodima. Nakon zatvaranja svakog od dijaloga za definisanje elemenata modela vrši se evaluacija unetih vrednosti, a dodatnu evaluaciju kasnije sprovodi i alat za generisanje koda.



Slika 26 Pogled na listu definisanih pregleda kroz alat za modelovanje



Slika 27 Definisanje jednog pregleda (laboratorijska analiza na slici levo) i njegovih polja (desno)

Kroz model se moraju uneti svi podaci koji su kasnije neophodni za generisanje softverskih komponenti. Jedino što se iz modela može izostaviti su prevodi. Ovde je dovoljno da korisnik na inicijalnoj formi (Slika 25) u kolekciji SupportedLanguages unese sve potrebne jezike i da definiše

osnovni (default) jezik kao i putanju do fajla sa resursima za prevođenje. Ukoliko osnovni jezik ne bude definisan, osnovnim će se smatrati prvi iz kolekcije unetih jezika.

```
- <MedicalRecordHeader>
  - <Elements>
    - <MedicalRecordItemElement>
        <Name>broj porodjaja</Name>
        <Type>integer</Type>
        <Description/>
        <Ranges/>
        <MeasurementUnit/>
      </MedicalRecordItemElement>
    - <MedicalRecordItemElement>
        <Name>broj trudnoca</Name>
        <Type>integer</Type>
        <Description/>
        <Ranges/>
        <MeasurementUnit/>
      </MedicalRecordItemElement>
    - <MedicalRecordItemElement>
        <Name>masa prvorodjenog deteta</Name>
        <Type>decimalnumber</Type>
        <Description/>
        <Ranges/>
        <MeasurementUnit/>
      </MedicalRecordItemElement>
    </Elements>
    <Name>Ginekoloski karton</Name>
    <Description/>
    <RecordDescription/>
  </MedicalRecordHeader>
```

Slika 28 Primer modela jednog zdravstvenog kartona

```
- <MedicalRecordItemElement>
    <Name>AV valvula zajednicka po tipu inkompl AV kanala</Name>
    <Type>shorttext</Type>
    <Description/>
  - <Ranges>
    - <MedicalRecordItemElementRange>
        <RangeName>ASD I</RangeName>
        <TargetGroup>ENUM</TargetGroup>
        <LowerLimit>0.000000</LowerLimit>
        <UpperLimit>0.000000</UpperLimit>
      </MedicalRecordItemElementRange>
    - <MedicalRecordItemElementRange>
        <RangeName>Inlet VSD</RangeName>
        <TargetGroup>ENUM</TargetGroup>
        <LowerLimit>0.000000</LowerLimit>
        <UpperLimit>0.000000</UpperLimit>
      </MedicalRecordItemElementRange>
    - <MedicalRecordItemElementRange>
        <RangeName>Rascep TV</RangeName>
        <TargetGroup>ENUM</TargetGroup>
        <LowerLimit>0.000000</LowerLimit>
        <UpperLimit>0.000000</UpperLimit>
      </MedicalRecordItemElementRange>
    - <MedicalRecordItemElementRange>
        <RangeName>Rascep MV</RangeName>
        <TargetGroup>ENUM</TargetGroup>
        <LowerLimit>0.000000</LowerLimit>
        <UpperLimit>0.000000</UpperLimit>
      </MedicalRecordItemElementRange>
    </Ranges>
    <MeasurementUnit/>
  </MedicalRecordItemElement>
```

Slika 29 Primer modela jednog atributa pregleda sa definisanim nabrojivim opsegom

Kada završi unos podataka, korisnik jednostavno samo treba da snimi model. Primeri elemenata iz snimljenog modela dati su na prethodnim slikama (Slika 28 i Slika 29). Model se može snimiti u spoljni XML fajl ili u bazu podataka. Takođe, kada se snimi model, generiše se i resursni fajl u koji kasnije mogu da se unose prevodi. Sam sistem za generisanje prevoda je generisan kasnije u poglavlju Generisanje prevoda.

Korišćenje komponente kao PropertyGrid u prilično olakšava posao razvoja, zato što se u okviru njega automatski generišu forme za unos vrednosti. Dodatna pogodnost korišćenja ovakve jedne klase je i u tome što podržava efikasno pisanje specijalizovanih editora za kolekcije baziranih na atributima klasa iz meta modela.

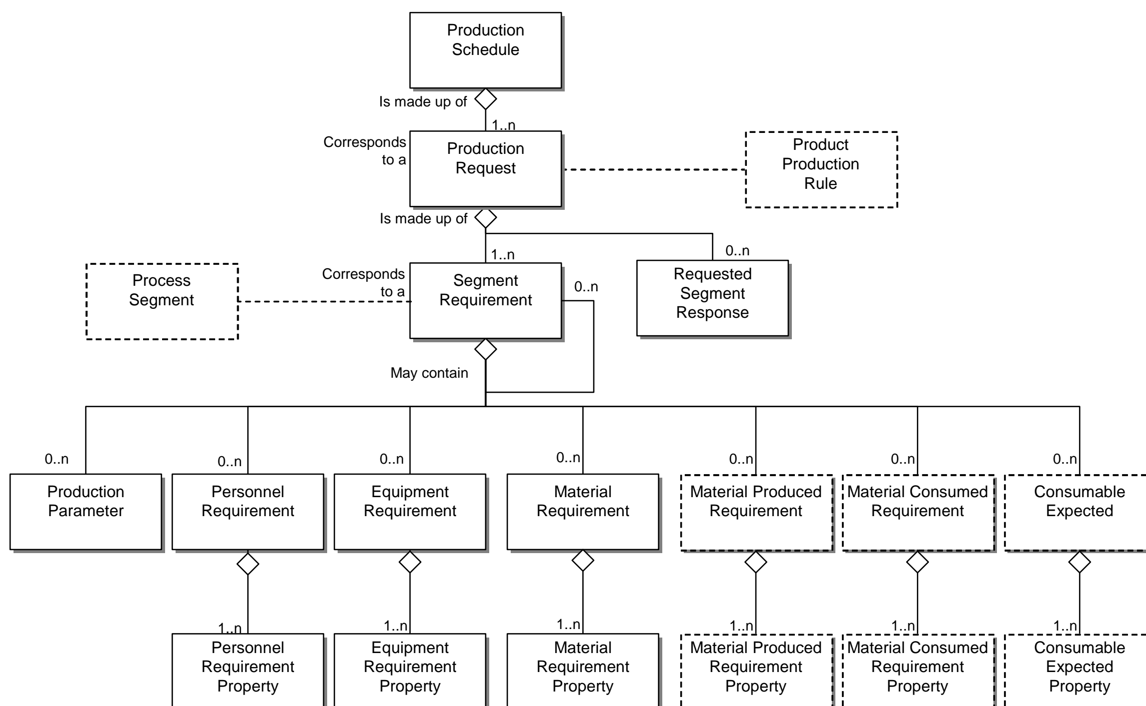
Sledeća pogodnost je i to što postoje mnoge biblioteke vizualnih komponenti koje u sebi imaju vizuelno atraktivnije gridove a koji inicijalno nasleđuju osnovni property grid. Na taj način jednostavnim uključivanjem druge biblioteke dobiće se grid potpuno drugačijeg izgleda. Tako da je jednostavno promeniti vizuelni identitet alata za modelovanje.

Takođe, alat za modelovanje može da učita i drugačiji meta model i da podrži modelovanje za entitete iz potpuno drugog domena.

2.7 Primer adaptacije alata za modelovanje

Prikazani pristup u realizaciji alata za modelovanje je dovoljno opšti da se može primeniti i za meta modele iz drugih domena. Ovde će biti prikazan proces adaptacije alata za modelovanje za primenu u modelu proizvodnog procesa u delu koji se bavi modelovanjem zahteva za proizvodnju.

Slika 30 prikazuje meta model zahteva za proizvodnjom. Vršni entitet se u opštem slučaju naziva ProductionSchedule, ali se u zavisnosti od industrije naziva i Production order i Workorder i Manufacturing order.



Slika 30 Meta model za definisanje zahteva za proizvodnju

U primeru koji će biti prikazan koristi se naziv Production order. On uz svoje osnovne atribute ima i listu entiteta tipa ProductionRequest koji odgovaraju pojedinačnim jedinicama proizvodnje. U primeru koji će biti prikazan. pojedinačne jedinice proizvodnje nazivaju se proizvodi, i termin ProductionRequest biće zamenjen terminom Product. Kolekcija SegmentRequirements je u ovom modelu samo kontejnerski objekat tako da će se u alatu za modelovanje pojaviti samo kolekcije njenih zavisnih objekata – PersonelRequirement, EquipmentRequirement itd. Dakle, to će biti meta model koji će alat za modelovanje učitati umesto meta modela za MIS sisteme.

Sledeća promena je zamena standardnog PropertyGrid-a, komponentom koja dolazi iz Telerik biblioteke. Da bi se ovo postiglo dovoljno je samo promeniti poziv konstruktora u metodi u kojoj se kreiraju vizuelne komponente. Umesto konstruktora osnovne klase, pozvaće se konstruktor klase RadGridView iz Telerik biblioteke. Slika 31 prikazuje dobijeni efekat i preko nje će biti modelovani entiteti tipa ProductionOrder. Ovde je napravljena još jedna promena, a to je da su osnovne akcije dostupne kroz dugmad, a ne kroz meni.

Create Orders

New Production Order

ExternalId !	
Description	
BatchSize	0
DeliveryDateSet	☐
DeliveryDate	
Priority	0
ProductType !	
LineItem	
ScheduledStartSet	☐
ScheduledStart	
ScheduledEndSet	☐
ScheduledEnd	
Status	Planned
⊟ CustomerOrder	CustomerOrder
ExternalId !	
CustomerName	
Properties	(Collection)
ProductionOrderConfigurationItems	(Collection)
Products	(Collection)

Save Order to file | Load Order from file | OK | Cancel

Slika 31 Izgled osnovne forme nakon promene vizuelne komponente koja prikazuje elemente modela

Kao što je bilo pomenuto, komponente kao što je property grid pozivaju validaciju unetih vrednosti u svako od polja nakon što se fokusom izađe iz njega. Validacija se okida preko atributske klase definisane uz polje u modelu. U primeru na prethodnoj slici validacija nije uspela za polja ExternalId i ProductType. Slika 32 prikazuje nekoliko atributa iz klase ProductionOrder gde se može videti način na koji je uključena validacija za atribute ExternalId i ProductType. Neposredno iznad naziva navedenih atributa postavljene su takozvane atributske identifikacije:

- za ExternalId su to [ProductionOrderExternalIdValidation] i [StringNotEmptyValidation],
- a za ProductType je to [StandardValuesNotEmptyValidationAttribute]

Za polja kao što su Description, BatchSize i Priority nije postavljen nikakav atribut tako da će ona biti validirana samo po tipu podataka. Validacija se vrši tako da se za svaki od atributa zove predefinisana metoda Validate. Uslov da sve ovo radi kako treba je da validaciona klasa nasledi klasu ValidationAttribute i da predefiniše metodu Validate, kako bi mogla da se uklopi u skup poziva koje podržava PropertyGrid (Slika 33).

```
#region Properties

[StringNotEmptyValidation]
[ProductionOrderExternalIdValidation]
5 references
public string ExternalId...

2 references
public string Description...

2 references
public int BatchSize...

[System.ComponentModel.RefreshProperties(RefreshProperties.All)]
1 reference
public bool DeliveryDateSet...

[ReadOnly(true)]
2 references
public DateTime DeliveryDate...

2 references
public int Priority...

[TypeConverter(typeof(ProductTypeConverter))]
[StandardValuesNotEmptyValidationAttribute]
4 references
public string ProductType...
```

Slika 32 Deo atributa definisanih u okviru klase ProductionOrder

```
20 references
public class StringNotEmptyValidationAttribute : ValidationAttribute
{
    13 references
    public override bool Validate(object value, string displayName, string propertyName, AttributeCollection attributes, 
    {
        errorMessage = null;
        string stringValue = value as String;
        if (String.IsNullOrEmpty(stringValue))
        {
            errorMessage = String.Format(translation.TranslateLine("[#cannotBeEmpty]", TranslationPrefix), displayName);
            return false;
        }
        return true;
    }
}
```

Slika 33 Primer klase koja služi kao atribut za validaciju

Ovde je bitno napomenuti da je moguće da se klase koje opisuju entitete iz meta modela uvoze iz spoljnih biblioteka. U tom slučaju, nije moguće direktno dekorisati njihove atribute klasama za validaciju, već ih je potrebno ili najpre naslediti, pa u modelu raditi sa nasleđenim klasama ili ih adaptirati na neki lokalni meta model.

Na kraju sve funkcionalnosti koje su bile dostupne kod modelovanja EHR domenskih modela ostaju dostupne i ovde. Primena standardnih komponenti ovde se pokazala kao izuzetno zgodna praksa zato što se onda na jednostavan način ceo alat za modelovanje veoma brzo može adaptirati za primenu u potpuno drugačijem domenu. Takođe, moguće je izuzetno brzo adaptirati izgled samih formi za unos polja kako bi se bolje uklopile u projekte.

Iako jednostavno dizajniran, alat za modelovanje je jedna od najznačajnijih komponenti ovde i modeli koji se definišu kroz njega koriste se kasnije kao ulaz u veliki broj različitih komponenti.

3 Dodatni alati za podršku razvoju baziranom na modelima

Alat za modelovanje podataka, odnosno za razvoj domenskih modela na osnovu meta modela je glavna komponenta za podršku razvoju informacionih sistema. Međutim, sam po sebi, ovakav alat omogućava jednosmernu komunikaciju od modela ka softveru, ali ne pruža mogućnosti da se omogući komunikacija od razvijanog softvera ka modelu. Odnosno, ne pruža mogućnost ažuriranja modela na osnovu već razvijenih delova softvera.

Dodatni alati koji prate alat za modelovanje se tehnički mogu realizovati i kao nezavisne aplikacije, ali i kao biblioteke koje omogućuju proširenje. Razvoj ovakvih alata je od velike važnosti u slučajevima kada treba unaprediti postojeći informacioni sistem ili kada brzo treba razviti skup novih, a međusobno sličnih, funkcionalnosti.

U ovom poglavlju biće predstavljeni alati za reverzni inženjering modela, za ispitivanje i validaciju strukture baze podataka i za automatsko generisanje proširive aplikacije za administraciju baze koja zapravo služi kao okvir za razvoj korisničkog interfejsa a i za validaciju strukture baze. Biblioteka za reverzni inženjering modela je zamišljena da u okolini definisanih tačaka proširenja pronađe tabele koje sadrže definisane interfejse i preslika ih u odgovarajuće klase koje će biti uključene u domenski model.

Kod biblioteke za ispitivanje strukture baze podataka, glavna ideja je bila napraviti alat koji će se vezati za bazu, ispitati njenu konzistentnost i normalizovanost. Dalje alat može da učita odgovarajuću konfiguraciju i na osnovu nje traži potencijalne tačke proširenja u postojećim bazama.

Nakon što struktura baze bude verifikovana, alat za generisanje administratorske aplikacije napraviće odgovarajući projekat u .NET razvojnom okruženju i u njega uključiti sve potrebne elemente kao i model podataka. Nakon kompajliranja, rezultat će biti osnova aplikacije za administriranje baze. Posle toga, parametrizacijom, i uključivanjem dodatnih softverskih komponenti završio bi se proces razvoja demonstrativne aplikacije koja korisnicima treba da približi funkcionisanje budućeg informacionog sistema.

Proširenje samog alata za modelovanje ovakvim biblioteka stvara se konzistentna celina pomoću koje se u mnogome može olakšati razvoj softvera. Primena svih ovih alata biće detaljno elaborirana u sledećim poglavljima. Na ovaj način se ubrzava proces realizacije i implementacije informacionih sistema i stvara se pogodno tle da se kroz primenu adekvatnih metodologija razvoja, ceo životni ciklus jednog softvera učini mnogo produktivnijim.

Korišćenjem prvenstveno alata za modelovanje i generisane osnovne aplikacije za administraciju, omogućuje se i osoblju koje će koristiti sistem, a koje nije programerski edukovano, da direktno utiče na razvoj sistema koji će koristiti. Ovakvim pristupom bi se olakšala realizacija specifičnosti koje poseduje odgovarajuća zdravstvena ustanova i olakšava kasnije prihvatanje informacionog sistema od strane krajnjih korisnika.

3.1 Reverzni inženjering – generisanje modela na osnovu baze podataka

Dodatna funkcionalnost koja je poželjna kod ovakvih alata je i mogućnost takozvanog reverznog inženjeringa. U konkretnom slučaju to znači ažuriranje modela na osnovu promena u bazi podataka. Alat analizira bazu i na osnovu promena koje uoči u okolini tačaka proširenja, predloži korisniku promenu modela. Takođe, alat može da izgradi ceo model na osnovu baze i definisanih pravila za traženje tačaka proširenja.

Postupak reverznog inženjeringa počinje tako što alat učita konfiguraciju opisanu u prethodnom poglavlju. Za svaki definisani entitet tipa MedicalDataContainer sistem potraži tabelu definisanu

atributom ExtensionPoint. Dalje, alat izdvoji sve tabele iz baze kojima je tabela definisana u ExtensionPoint referencirana kao roditeljska tabela. Sve pomenute tabele predstavljaju potencijalne elemente modela. Sledeći korak je provera tabela po dodatnim poljima definisanim u ExtensionRules sekciji. Ukoliko izdvojena tabela zadovoljava sve uslove iz kolekcije ExtensionRules automatski će biti dodata u model. U suprotnom će ili korisnik ili alat (ukoliko je konfigurisan) da odluči o njenom daljem statusu. Alat može da bude konfigurisan da automatski odbaci takve tabele ili da ih automatski proširi svim nedostajućim poljima iz modela.

Alat takođe može da ažurira i tabele čiji su povezani entiteti već u modelu, a koje su promenjene mimo alata za modelovanje. U tom slučaju, alat će izdvojiti listu polja koja se razlikuju u modelu i u bazi. O razlikama može da odluči ili korisnik ili alat automatski. Alat, u zavisnosti od konfiguracije, može da odluči da odbacuje promene u modelu, da odbacuje promene u bazi ili da promene sa obe strane uvrsti u model.

Tipovi promena koje se ovde identifikuju su sledeće:

- Promene tipa A – novi atribut dodat u model (u tabeli ne postoji polje sa istim imenom)
- Promene tipa B – novo polje dodato u bazu (u modelu ne postoji atribut sa istim imenom)
- Promene tipa C – postoji polje u bazi sa istim nazivom kao i atribut u modelu ali se razlikuju po tipu podataka – tipovi su iz iste kategorije tipova (float i decimal, int i bigint, varchar i nvarchar itd)
- Promene tipa D – postoji polje u bazi sa istim nazivom kao i atribut u modelu ali se razlikuju po tipu podataka – tipovi su iz različitih kategorija podataka (float i varchar, decimal i int, itd)

Način ažuriranja modela se definiše u okviru sekcije ReverseEngineeringSettings (Slika 34). Ovde se definiše kako će biti realizovan način rešavanja konflikata prilikom spajanja učitanog modela i modela rekonstruisanog na osnovu strukture baze. Atribut DefaultConflictResolutionStrategy definiše opšti način razrešavanja konflikata za svaku od prethodno definisanih MedicalDataContainer sekcija. Taj način razrešavanja konflikata se može redefinisati za svaki od posebno definisanih MedicalDataContainer-a.

Podržani načini razrešavanja konflikata su sledeći:

- Manual – korisnik sam razrešava konflikte. Nakon analize učitanog modela i modela dobijenog nakon čitanja baze, korisnik dobije listu promena i odlučuje o tome kako će biti razrešavani. Korisnik može prihvatiti promenu iz modela, promenu iz baze ili ručno podesiti naziv polja i tip podataka
- AcceptModelChangesOnly – sve promene koje dođu iz baze biće ignorisane
- AcceptDatabaseChangesOnly – model se ažurira na osnovu tabele i odbacuju se sve promene koje su u međuvremenu dodate u model
- Merge – sve promene će se uvrstiti u model, a korisnik kasnije može da interveniše

Pravila za način rešavanja konflikata Merge su definisana tako da se redukuje mogućnost gubitka podatka. Tako na primer, promene tipa A i B (nova polja dodata u model ili bazu) će biti uvrštena u rezultantni model bez ikakvih promena.

Kod promena tipa C rezultantni model će imati polje sa odgovarajućim nazivom, a tip podatka će se ažurirati po onome koji je šireg opsega. Na primer, ako je int (32-bitni ceo broj) tip podataka na jednoj a bigint (64-bitni ceo broj) na drugoj strani, konflikt će biti rešen tako što će tip podataka biti bigint. U

slučaju da je na jednoj strani decimal(15,6) a na drugoj decimal(19,3) konflikt će biti rešen tako što će rezultantni slog imati decimal(19,6). To znači ako je na jednoj strani bio decimalni broj sa 6 decimala i ukupno 15 cifara, a na drugoj decimalni broj sa 5 decimala i ukupno 19 cifara rezultat će biti decimalni broj sa 19 mesta od kojih je 6 rezervisano za decimale.

Promene tipa D u automatskom modu razrešavanja kao rezultat daju dva atributa – jedan sa originalnim imenom i jedan na čije je ime konkateniran jedinstveni broj u modelu kako bi se obezbedilo jedinstveno ime svakog od atributa. U ovom slučaju je na kraju ipak potrebna manuelna intervencija pošto je kod ovakvih primera najčešće slučaj da je tekstualno polje zamenjeno referencijalnim integritetom.

```
  <?xml version="1.0"?>
- <ReverseEngineeringSettings>
    <DefaultConflictResolutionStrategy>Merge</DefaultConflictResolutionStrategy>
  - <MedicalDataContainerMergeStrategy>
      <MedicalDataContainerName>BaseEHRItemTableName</MedicalDataContainerName>
      <ConflictResolutionStrategy>AcceptModelChangesOnly</ConflictResolutionStrategy>
    </MedicalDataContainerMergeStrategy>
  - <MedicalDataContainerMergeStrategy>
      <MedicalDataContainerName>BaseAdministrativeTableName</MedicalDataContainerName>
      <ConflictResolutionStrategy>AcceptDatabaseChangesOnly</ConflictResolutionStrategy>
    </MedicalDataContainerMergeStrategy>
  - <MedicalDataContainerMergeStrategy>
      <MedicalDataContainerName>BaseEHRDocumentTableName</MedicalDataContainerName>
      <ConflictResolutionStrategy>Manual</ConflictResolutionStrategy>
    </MedicalDataContainerMergeStrategy>
  </ReverseEngineeringSettings>
```

Slika 34 Konfiguracija za reverzni inženjering

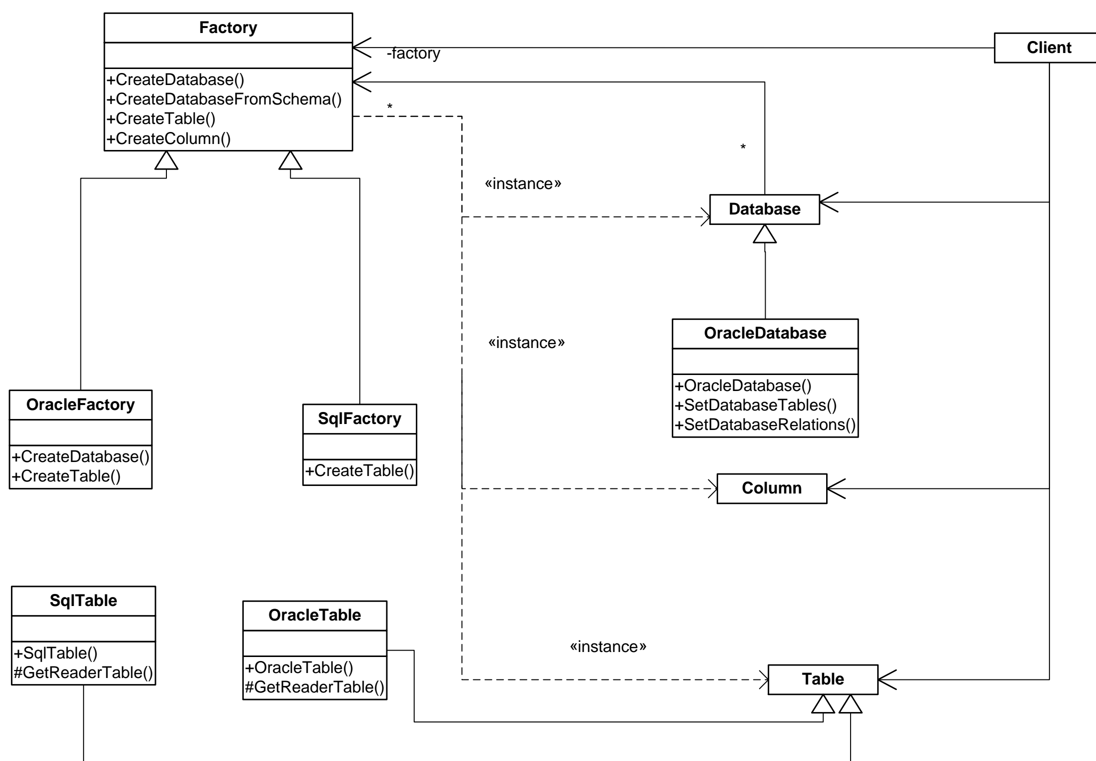
Ovaj pristup u kreiranju modela se pokazao izuzetno korisnim kada treba uvesti u model celu novu klasu entiteta koji su vezani za neku od tačaka proširenja. Na ovaj način je moguće efikasno uvesti u model velike delove postojećih sistema i smanjiti vreme potrebno za integraciju. U slučaju kada treba integrisati potpuno novi sistem ili kada treba na potpuno novu tehnologiju (npr. sa Delphi-ja na .NET) značajno se redukuje potreba za ponovnim kodiranjem.

3.2 Biblioteka za analizu strukture podataka

Biblioteka za ispitivanje strukture baze je inicijalno bila zamišljena da bude deo alata za generisanje administratorske aplikacije, ali je vremenom proširena i tako da može da se koristi za traženje tačaka proširenja po postojećim bazama. Ideja koja stoji iza alata za generisanje administratorske aplikacije je, kreiranje aplikacije koja će služiti kao prototip u toku razvoja sistema, a kasnije kao alat za efikasniju validaciju i verifikaciju podataka u bazi.

Kada se kreira administratorska aplikacija, korisnik startuje alat za generisanje, on se konektuje na specifikovanu bazu, verifikuje njenu strukturu i generiše novi .NET projekat i izvršni (exe) fajl, koristeći CodeDom biblioteku. Generisani projekat se može dalje razvijati kao posebna aplikacija.

Biblioteka za analiziranje šeme baze je nazvana DatabaseStructure. Prilikom komunikacije sa bazom ona koristi osnovne .NET klase gde god je moguće. Na taj način se omogućuje da aplikacija bude upotrebljena, teoretski, nad svim bazama koje OLE DB (.NET-ov deo za komunikaciju sa bazama) podržava. Gde god to nije bilo u potpunosti dovoljno, može se iskoristiti obrazac „Factory“ kako bi se kreirale specijalizovane klase za one DBMS sisteme kod kojih funkcionalnosti koje dolaze iz osnovnog OLE DB-a nisu dovoljne. Na primer, za Microsoft-ov SQL server, koristi se klasa SqlTable i umesto standardne OLE DB klase Table zato što klasa Table ne podržava neke od specifičnih osobina tabela iz Microsoft SQL servera, kao na primer „auto increment“ za polja koja su označena kao ključevi.



Slika 35 Factory projektni obrazac upotrebljen za kontrolu procesa instancijacije klasa

Obrazac „Factory“ je upotrebljen i da bi kontrolisao proces instancijacije klasa. Klasa Database enkapsulira koncept baze podataka i sadrži tabele i relacije. U slučaju SQL Servera ova klasa mapira entitet Catalog, dok u slučaju Oracle baze podataka Database modeluje šemu asociranu aktuelnom korisniku. Klasa Database donosi zapravo proširenje osnovne OLE DB funkcionalnosti implementirano kroz „Factory“ projektni obrazac (Slika 35).

Osnovni strukturalni deo biblioteke DatabaseStructure je takozvani „database wrapper“. On sadrži klase koje služe za čitanje meta podataka određene baze – definicije tabela, kolona i relacija. Sve ove klase su bazirane na osnovnim klasama OLE DB tehnologije i, po Microsoftovoj specifikaciji, bi trebalo da rade sa svim bazama koje podržavaju OLE DB komunikacioni interfejs. Pomenuti „database wrapper“ je dodatno podešavan i testiran za SQL Server i Oracle. Dakle, „database wrapper“ je implementiran kroz sledeće strukture: osnovna OLE DB implementacija i implementacija specifičnih klasa za podršku SQL Server i Oracle bazama (Slika 36).

Slika 35 prikazuje kako je modelovan skup klasa koje opisuju meta podatke. Osnovna klasa ovde je klasa *Database* jer su u objektu ove klase ugnežđeni nizovi objekata svih ostalih klasa iz modela. Ona definiše procedure za pristup bazi, preuzimanje podataka iz baze, poziva validacione metode i kreira kompletnu logičku reprezentaciju baze. Osnovno ponašanje je: baza se oslanja na OLE DB klase koje omogućuju direktan pristup bazi i njenim podacima. Klasa *OracleDatabase* implementira neke razlike koje su specifične za Oracle baze podataka, a odnose se prvenstveno na nepostojanja koncepta „baza“, već insistiranje na korišćenju koncepta „šema“.

Na slici Slika 36 prikazana je detaljna struktura klasa. Može se uočiti da klasa *Database* ima privatna polja za definisanje svih podataka neophodnih za povezivanje na bazu – vrsta baze (*provider*), ime servera baze podataka (*dbServer*), naziv baze (*dbName*) i kredencijale za pristup bazi (korisničko ime, lozinka, naziv domena).

Klasa *Table* opisuje tabelu baze podataka koja se jedinstveno identifikuje na osnovu svog imena, roditeljske baze i pripadajuće šeme. Ona sadrži niz svih kolona, podatke o primarnom ključu, podatke o stranim ključevima, kao i podatke o svim relacijama u kojima tabela učestvuje. Takođe, ova klasa implementira i metod za poređenje objekata klase table, koja služi da generiše upozorenja ako je struktura tabele promenjena nakon što je model baze kreiran. Ova klasa još pruža korisniku i metode pomoću kojih može da dodaje i briše relacije između tabela u cilju normalizacije baze podataka. Klase *SqlTable* i *OracleTable* nasleđuju klasu *Table* i implementiraju podršku za specifične osobine tabela, a koje nisu deo standardnog OLE DB interfejsa, u prvom redu metode za kreiranje tabela zbog različite SQL sintakse i metode za preuzimanje sadržaja tabele primenom takozvanih *Reader* objekata.

Klasa *Relation* opisuje relaciju između dve tabele. Ona sadrži tabelu koja sadrži tabelu sa primarnim ključem i tabelu u kojoj je pomenuti primarni ključ spoljni (foreign key table). Uz to ovde se čuva i niz parova kolona koje opisuju mapiranje primarnih i spoljnih ključeva iz navedenih tabela. Ovde se vodilo računa da se u odgovarajućoj bazi koristi odgovarajuća terminologija, pa se tako tabela sa primarnim ključem u slučaju SQL baze zove „lookup table“, a ona sa spoljnim ključem „master table“. Klasa Relation, takođe, implementira metode koje omogućuju manipulisanje pravilima za ažuriranje i brisanje podataka (update rule & delete rule). Ostale klase koje modeluju bazu podataka su:

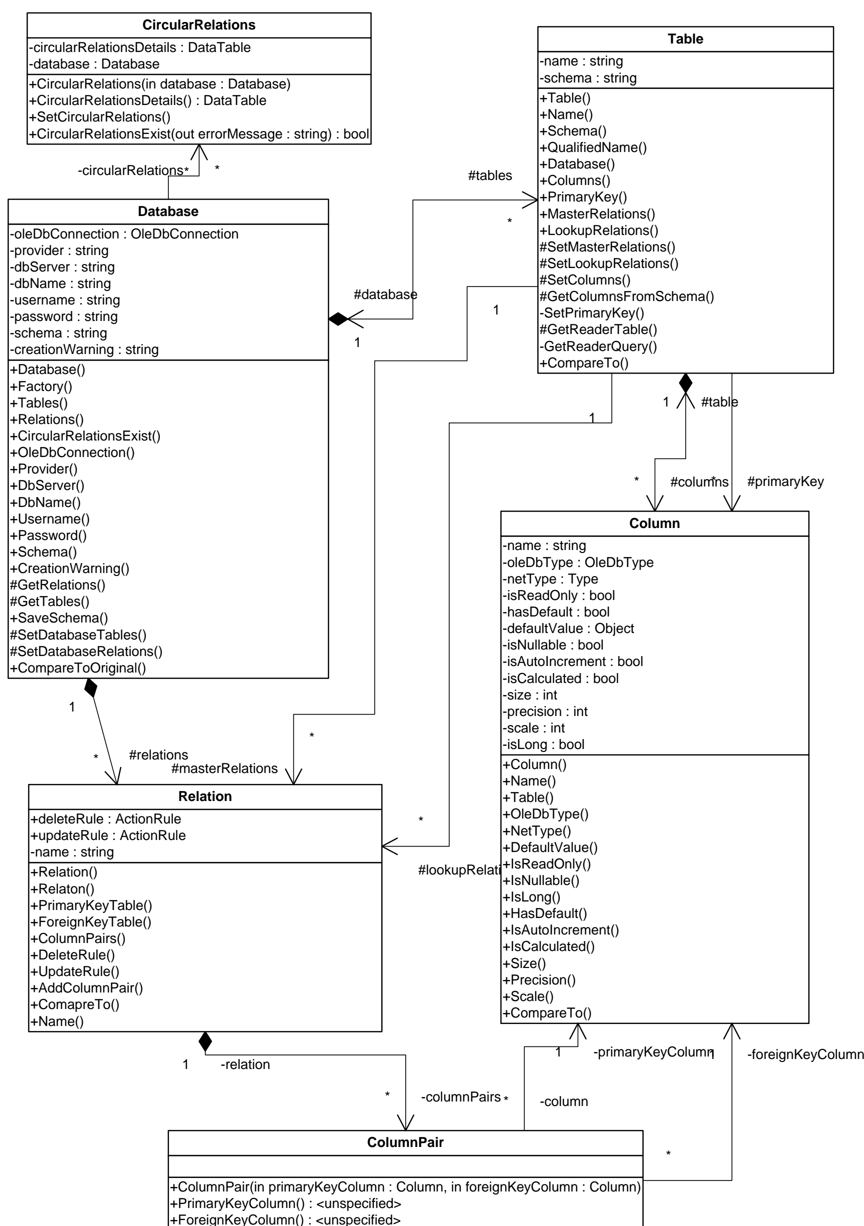
- *Column* - opisuje kolonu tabele iz baze podataka. Ovde su podržana polja koja opisuju u potpunosti kolonu tabele podataka – tip podatka po OLEDB-u, opšti tip podatka, podrazumevana vrednost, da li može da bude nedefinisana (null), da li je zaključana za nove vrednosti, da li je izračunavano polje, itd.
- *ColumnPair* predstavlja par kolona koji uspostavljaju vezu između dve tabele.
- *TableArrayList* nasleđuje System.Collections.ArrayList i deklariše indekser nad poljem *QualifiedName* tabele kako bi se pristupalo tabelama direktno po njihovim imenima, a ne samo po indeksu (rednom broju u nizu).

- *ColumnArrayList* takođe nasleđuje System.Collections.ArrayList i deklariše indekser nad poljem Name koje predstavlja ime kolone.

- *RelationArrayList* takođe nasleđuje System.Collections.ArrayList i deklariše indekser nad poljem Name koje predstavlja ime relacije.

- *ColumnPairArrayList* nasleđuje System.Collections.ArrayList i specijalizovana je da radi sa nizom objekata klase ColumnPairs.

- Klasa *CircularRelations* (Slika 36) sadrži metode koje služe da lociraju cirkularne relacije u bazi. Cirkularne relacije su ozbiljna anomalija u dizajnu baze, i onemogućuju transformaciju šeme baze u strukturu stabla.



Slika 36 Ekstenzija OLE DB klasa

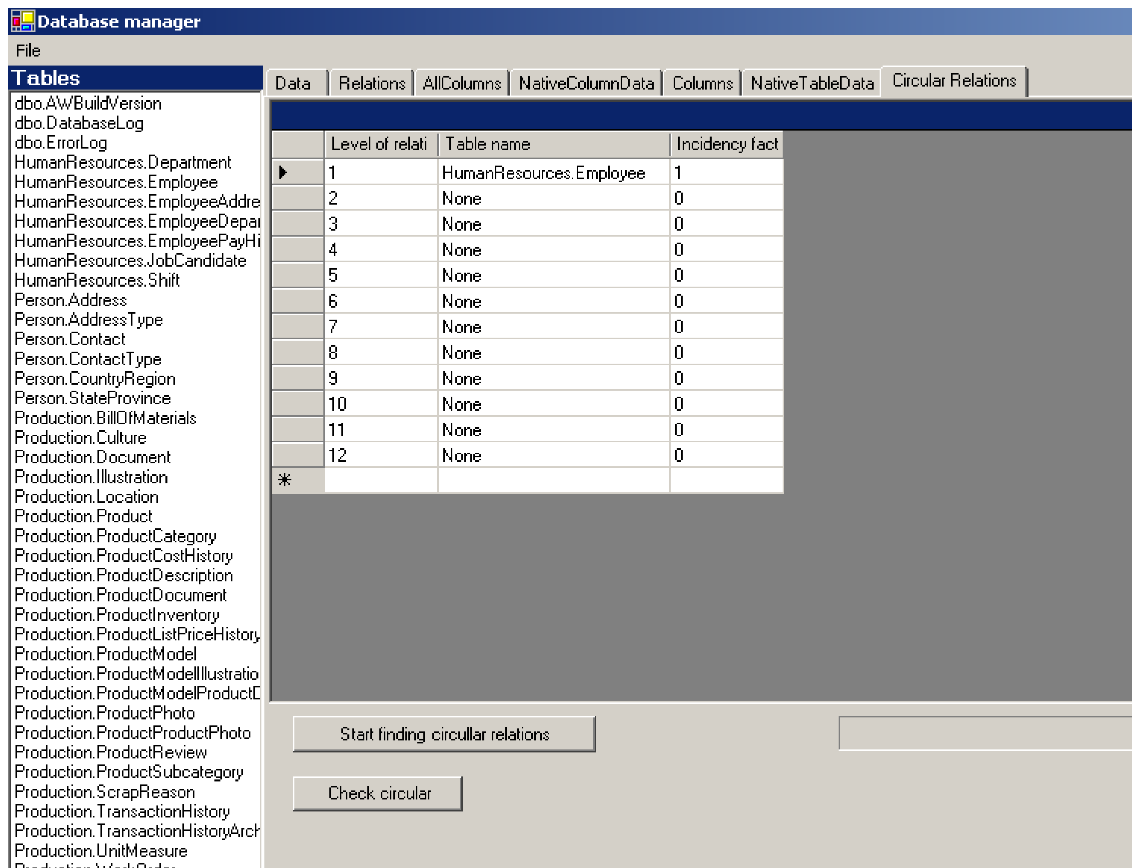
Klijentska aplikacija koristi biblioteku *DatabaseStructure* i direktno pristupa klasama koje su na najvišem hijerarhijskom nivou (*Database*, *Table* i *Column*). Ove klase nose sa sobom implementaciju

OLE DB interfejsa kao i implementaciju osnovnih funkcionalnosti za analizu strukture baze. Kada se zahteva nova funkcionalnost, vezana za neku novu bazu koja nije u potpunosti podržana kroz OLE DB, mogu se jednostavno kreirati nove klase i definisati novi „factory method“ koji služi da kreira objekte tih novih klasa. Na ovaj način, primenom Abstract Factory obrasca, nove klase će moći da budu jednostavno dodate bez potrebe da klijentska aplikacija menja način na koji koristi funkcionalnosti biblioteke.

Slika 37 Primer meta podataka ekstrahovanih pomoću DatabaseStructure biblioteke (tabela HumanResources.Employee iz baze AdventureWorks)

Slika 38 Primer tabele (HumanResources.Employee) koja učestvuje u skraćenom skupu tabela

Cirkularne relacije se smatraju izuzetno opasnom anomalijom pri projektovanju baza podataka. One mogu dovesti do beskonačnog niza transakcija koji nastaje nakon najobičnijeg pokušaja ažuriranja podataka. Za tri ili više tabela se kaže da su u cirkularnoj relaciji, ako je tabela A master tabela za tabelu B, tabela B master tabela za tabelu C i tabela C master tabela za tabelu A.



Slika 39 Prikaz određenih cirkularnih relacija

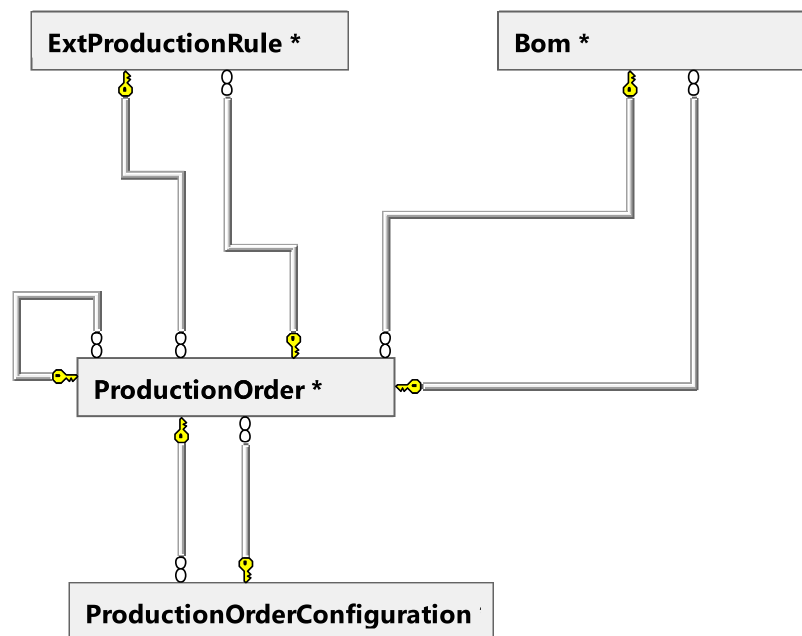
Postojanje ovakve vrste veza može da dovede do mogućeg trajnog zaključavanja podataka među sobom (deadlock), a takođe je, u slučaju postojanja cirkularne relacije, nemoguće predstaviti tabele iz baze u vidu stabla. U tom slučaju bi cirkularna relacija uticala na to da grane stabla koje sadrže tabele iz cirkularne relacije budu sačinjene od beskonačno mnogo elemenata.

Relacije „više–na–više“, odnosno relacije „M:N“ tipa, se, u zavisnosti od autora, nekada tretiraju kao cirkularne, a nekada ne. Takođe, podržani su u nekim bazama, a u nekima ne. Biblioteka DatabaseStructure podržava samo veze jedan–na–jedan i jedan–na–više. U slučaju veza „M:N“ biblioteka će uočenu vezu konvertovati u dve 1:N veze uz kreiranje privremene tabele spoja, a prema specifikaciji treće normalne forme za dizajniranje baza. Nakon što se biblioteka poveže na bazu i preuzme njene meta podatke, odnosno podatke o tabelama i relacijama, sledi verifikacija strukture baze i traženje cirkularnih relacija kao njen važan segment.

Tabele i relacije iz baze se najčešće predstavljaju strukturom orijentisanog grafa, gde su tabele čvorovi, a relacije potezi. Traženje cirkularne relacije se, u tom slučaju, svodi na traženje puta određene dužine kroz graf gde su početna i krajnja tačka identične. Postoji više algoritama za traženje puteva kroz grafove, a biblioteka DatabaseStructures koristi metodu stepenovanja kvadratnih matrica, pošto koristi metodu matrice incidencije za predstavljanje grafa. Matrica incidencije je kvadratna

matrica koja ima onoliko vrsta i kolona koliko graf ima čvorova. U preseku odgovarajuće vrste i kolone se upisuje broj koji pokazuje koliko ima potega koji polaze iz jednog čvora, a završavaju se u drugom. Polazni čvorovi se prikazuju sa strane vrsta, a odredišni sa strane kolona.

Prvi korak je kreiranje prazne matrice incidencije reda n, gde je n broj tabela u posmatranoj bazi. Prazna matrica znači da su vrednosti svih njenih polja jednake nuli. Nakon toga se prođe kroz niz relacija i matrica incidencije se popuni odgovarajućim vrednostima. Svaka relacija se predstavi kao uređeni par indeksa tabela na koje se odnosi. Prva vrednost u uređenom paru je indeks „master" tabele po Microsoft terminologiji (foreign key tabele), a druga vrednost je indeks tabele koja čuva primarni ključ za master tabelu (lookup tabela po Microsoft terminologiji). Nakon što se odrede indeksi tabela, uveća se u matrici incidencije vrednost koja se odnosi na konkretnu relaciju za 1. Nakon završetka iteracije kroz niz relacija, matrica incidencije će biti inicijalizovana. Cirkularne relacije se inače, u matrici incidencije lako uočavaju jer su predstavljene vrednostima različitim od nule po glavnoj dijagonali. Ispitivanje da li postoje cirkularne relacije određenog reda se svodi na stepenovanje matrice incidencije na odgovarajući stepen i proveravanje vrednosti na glavnoj dijagonali. Ako su sve vrednosti na glavnoj dijagonali nule, onda nema cirkularnih zavisnosti. Ovde treba napomenuti da se cirkularne relacije reda 1 ne tretiraju kao anomalija u skladu sa trećom normalnom formom za projektovanje baza.



Slika 40 Primer cirkularnih relacija reda 1 u standardnoj B2MML šemi

Slika 40 prikazuje primer korišćenja cirkularnih relacija reda 1 u projektovanju šeme baze za standardni B2MML model. Centralni entitet prikazan ovde je ProductionOrder. Jednom entitetu ovog tipa potrebni su entiteti tipa BOM (koji modeluje recept, odnosno opisuje potrebne sirovine za proizvodnju), ProductionOrderConfiguration (skup vrednosti određenih signala koji se prosleđuju u proizvodni system) i ExtProductionRule (skup komandi koje se direktno kopiraju iz sistema za planiranje). Tako, jedan entitet tipa ProductionOrder ima sva tri navedena entiteta kao spoljne ključeve. U isto vreme, navedeni dodatni entiteti se definišu na nivou jednog ProductionOrder-a i ne mogu da postoje nezavisno od njega, tako da im je on potreban kao spoljni ključ. Na ovaj način se obezbeđuje da ProductionOrder ima uvek tačno jedan aktivan BOM (ili ExtProductionRule ili

ProductionOrderConfiguration), a da u toku svog postojanja može da promeni više njih i da se sačuva cela istorija promena.

Stepenovanje matrica incidencije ima smisla do reda koji je jednak broju tabela u bazi. Međutim treba imati u vidu da je množenje matrica, pa i retko posednutih kakve su matrice incidencije relativno dugotrajan proces. Zbog toga se ovde pristupa redukciji matrica incidencije na niži stepen. Naime, matrica incidencije, koja predstavlja bazu podataka, je, zbog same prirode baza, retko posednuta matrica. Pošto nama matrice incidencije trebaju kako bi tražili cirkularne relacije, od interesa su nam jedino tabele koje u isto vreme imaju i „primary“ i „foreign“ ključne veze. Odnosno samo one koje su nekim tabelama roditelji i u isto vreme nekim tabelama deca. Redukovana matrica incidencije se sada formira za podgraf koji je sačinjen samo od tabela koje zadovoljavaju prethodno definisani uslov i od veza među tim tabelama. Na ovaj način se matrica incidencije može redukovati nekoliko puta. Tabela 1 prikazuje rangove nekoliko test baza pre i posle navedene redukcije.

Nakon završene redukcije matrice incidencije može se početi sa traženjem cirkularnih relacija tako što se redukovana matrice incidencije diže na svaki stepen počev od drugog pa zaključno sa stepenom koji odgovara njenom rangu.

Tabela 1. Primeri redukcije ranga matrice incidencije za takozvane "test-baze"

Ime baze	Inicijalni rang matrice	Rang redukovane matrice
AdventureWorks	70	12
ReportServer	27	2
SharePointServices	161	13
SharePoint_Config	24	4

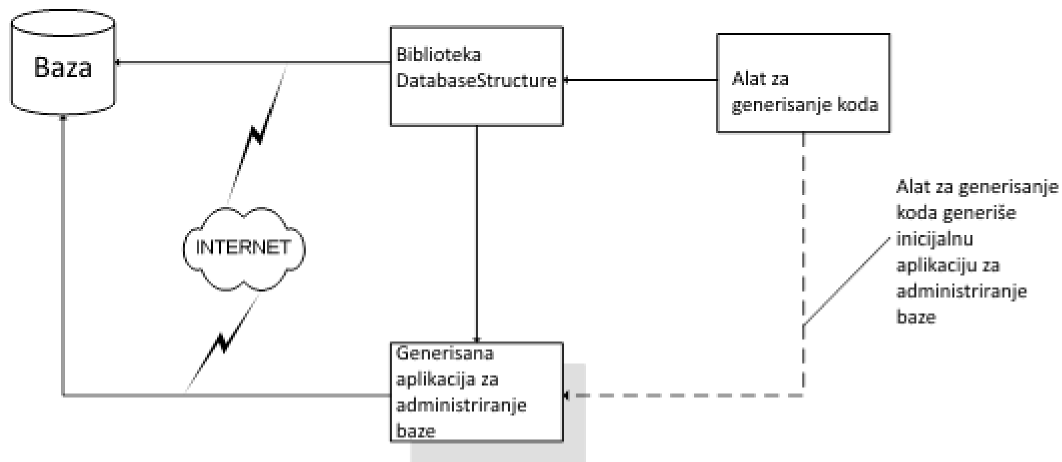
Ovakav metod za traženje cirkularnih relacija je izabran zato što je najjednostavniji za implementaciju, a kako je traženje cirkularnih relacija operacija koja se retko koristi, obično samo jednom, kada se prvi put analizira neka baza ili na eksplicitni zahtev korisnika, faktor brzine algoritma nije bio ključan. Inače, redukcija matrice incidencije grafa koji prikazuje bazu podataka, zahvaljujući samoj prirodi baza podataka, daje izuzetno dobre rezultate, pa je npr. za baze od 200 tabela jako redak slučaj da postoji 20 tabela koje mogu da učestvuju u cirkularnim relacijama reda dva ili više.

3.3 Alat za generisanje administratorskih aplikacija

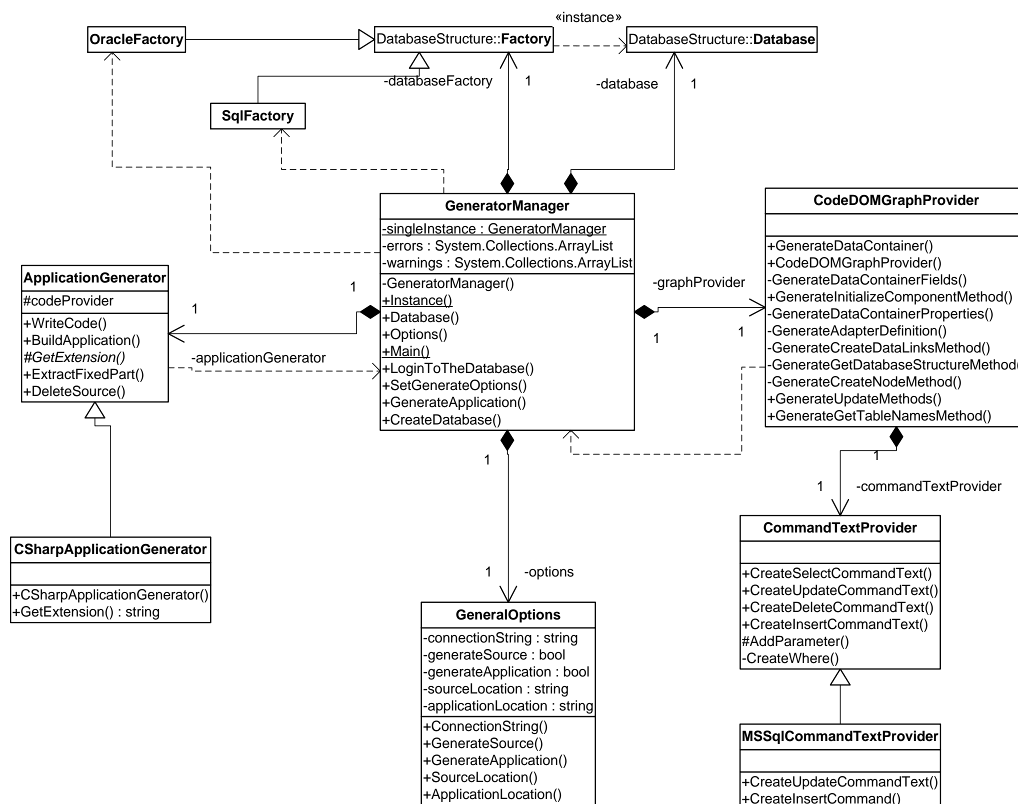
Biblioteka DatabaseStructure je iskorišćena kao osnova softverskog alata za generisanje aplikacija za manipulisanje sa podacima iz baze podataka, a na osnovu šeme baze – takozvane administratorske aplikacije. Softverski alat je u stanju da generiše i izvršni kod aplikacije i .NET projekat koji se dalje može dorađivati. Zapravo, generatorski alat, generisane aplikacije i biblioteka DatabaseStructure se mogu posmatrati i kao delovi jedinstvenog sistema. Slika 41 prikazuje njihov međusobni odnos.

Slika 42 prikazuje dijagram osnovnih klasa generatorskog alata. Osnovu čini GeneratorManager, klasa čije su metode odgovorne za sve podržane procese. Ona najpre treba da obezbedi vezu sa bazom podataka. Ta veza je ostvarena kroz agregaciju sa klasom DatabaseFactory koja dolazi iz DatabaseStructure biblioteke i koja je odgovorna za uspostavljanje konekcije i provere strukture baze.

Sve greške i upozorenja koja biblioteka DatabaseStructure generiše u toku validacije baze smeštaju se u posebne liste nazvane *errors* i *warnings*.



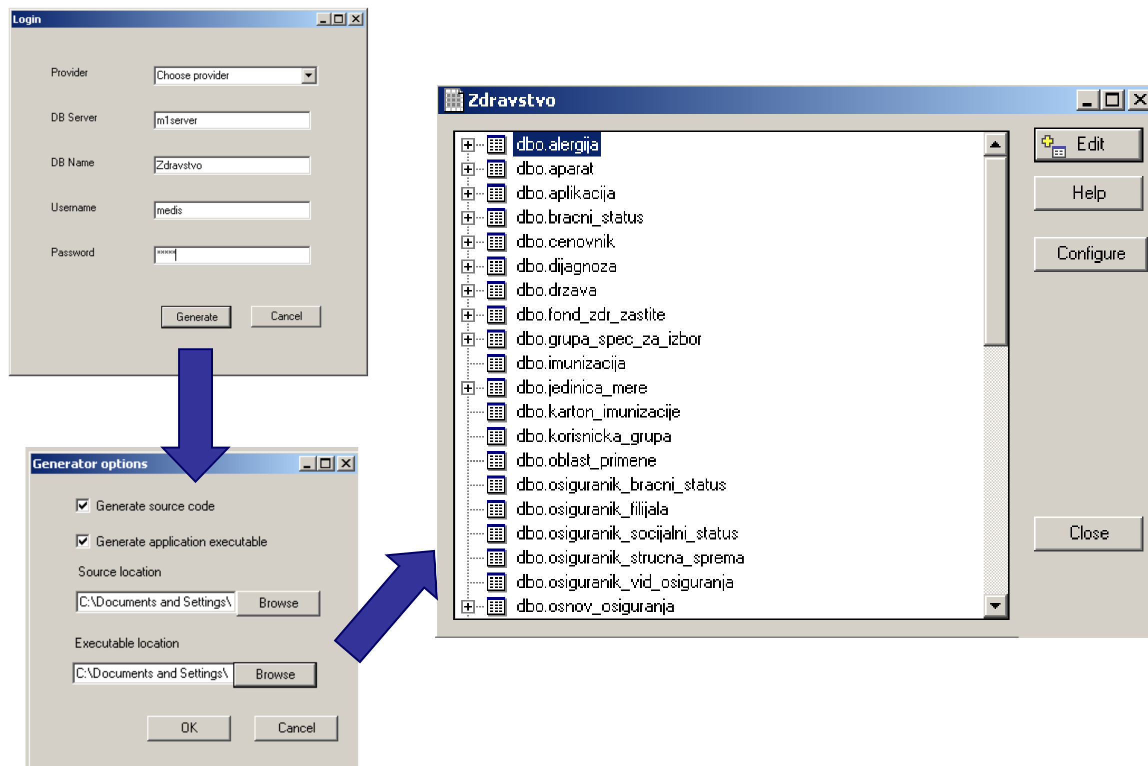
Slika 41 Odnos između komponenata generatorskog alata



Slika 42 Dijagram klasa alata za generisanje administratorske aplikacije

Sve što je potrebno za generisanje koda dolazi kroz klasu CodeDOMgraphProvider i tu su definisane metode koje treba da generišu kod za pojedine komponente buduće administratorske aplikacije. Ukoliko je potrebno pojedine inicijalne implementacije se mogu zameniti specifičnim. U primeru (Slika 42) je prikazan osnovni dijagram proširen klasom koja treba da generiše SQL skriptove za Microsoft Sql Server DBMS. Inicijalne metode za generisanje SQL komandi definisane su u opštem

SQl dijalektu. Kako se Micorsoft Sql Server razlikuje u pojedinim detaljima, razvijena je klasa MSSqlCommandTextProvider koja nasleđuje osnovnu klasu CommandTextProvider, ali donosi novu implementaciju za sve virtuelne metode osnovne klase.



Slika 43 Proces generisanja aplikacije

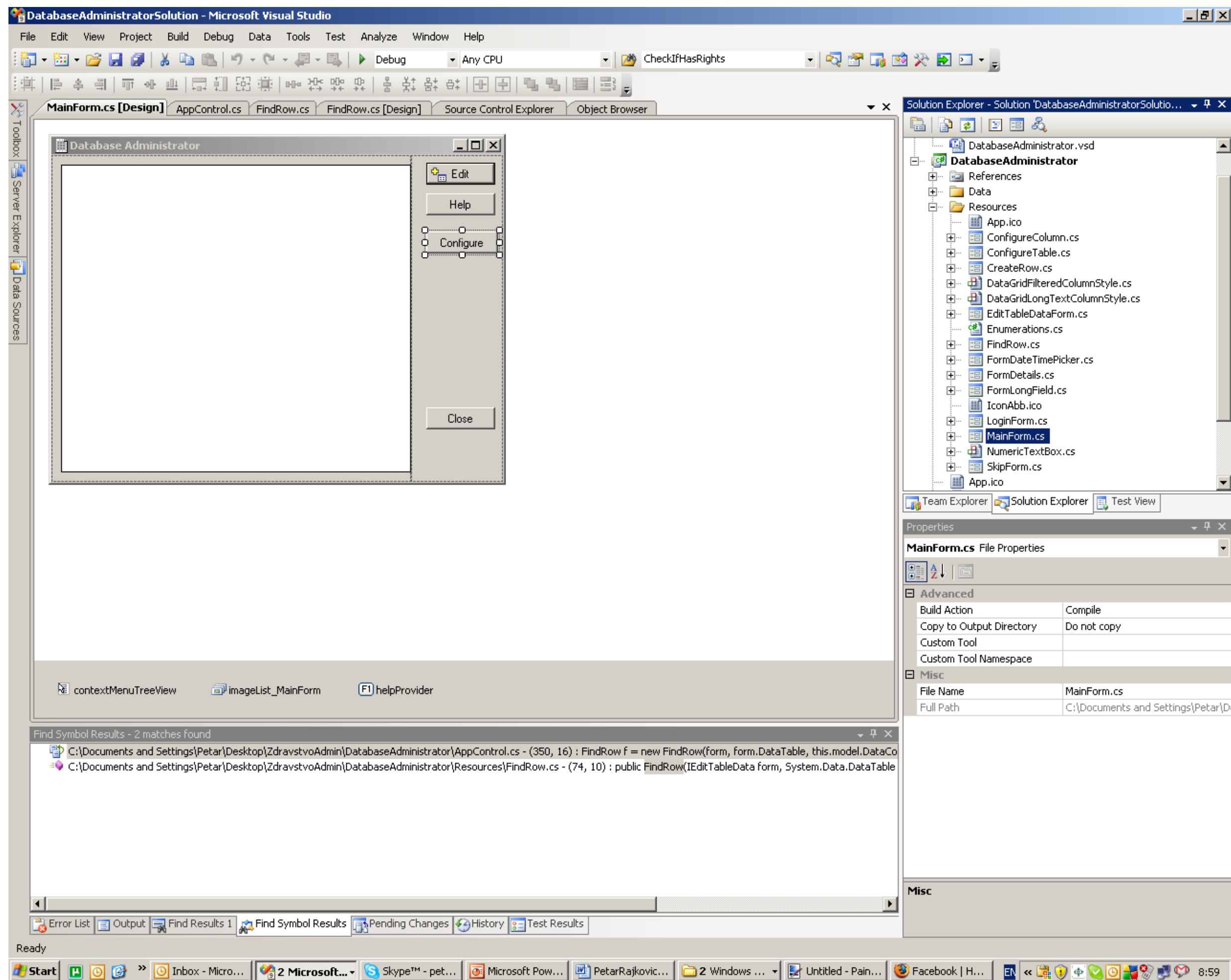
Sa druge strane je klasa koja enkapsulira pozive ka kompajlerima (ApplicationGenerator) i koja povezuje šablonski i generisani kod u jedinstveni projekat. U okviru nje su definisane metode za povezivanje fiksnog i generisanog dela koda, za kreiranje fajlova koji sadrže kod i za kompajliranje. Tu je takođe i funkcija za uklanjanje privremeno generisanih fajlova. Ukoliko je potrebno, za podršku specifičnostima u određenim programskim jezicima, definisaće se nova klasa koja nasleđuje klasu ApplicationGenerator i u njoj će se predefinisati potrebne metode. Na dijagramu alata za generisanje administratorske aplikacije (Slika 42) dodata je klasa CSharpApplicationGenerator u okviru koje je predefinisana metoda GetExtensions koja služi da pribavi sva naknadno dodata proširenja u sistem. Ona je ključno mesto koje će u kasnijem razvoju omogućiti da se u aplikaciju uključuju forme generisane od strane alata za generisanje softverskih komponenti.

Kao što je već navedeno, generatorski alat omogućuje automatsko generisanje aplikacije za rad sa određenom bazom podataka. On kreira odgovarajuće datoteke sa izvornim kodom kao i izvršni fajl. Generisanje aplikacije počinje od logovanja na server baze podataka i tada korisnik definiše parametre za proces generisanje koda (Slika 43). Sledeći korak je analiza odabrane baze i kreiranje njenog objektnog modela. Na osnovu objektnog modela i strukture baze, kreira se objekat DataContainer klase, koji će biti „proxy“ u komunikaciji sa bazom (Slika 46). Takođe, svi generisani fajlovi se mogu grupisati u projekat i na njima se može dalje raditi kroz razvojno okruženje (Slika 44).

U svom glavnom prozoru administratorska aplikacija prikazuje listu svih tabela u bazi. Tabele mogu biti prikazane u top-down ili u bottom-up modu. U top-down modu, uz tabelu najvećeg prioriteta biće prikazan znak za ekspanziju čvora u stablu i nakon otvaranja podstabla biće prikazane sve zavisne

tabele. Rekurzivno, kliktanjem na znak za otvaranje dela stabla doći će se do tabela koje nisu roditeljski entitet ni jednoj drugoj tabeli. U delu gde su prikazane sve tabele, uz takve ne stoji znak za proširenje dela stabla.

Slika 43 prikazuje tabele iz MIS Medis.NET prikazane u bottom-up modu. To znači da uz tabele najvišeg prioriteta (kataloške tabele) ne stoji mogućnost proširenja stabla, već samo uz tabele koje imaju roditeljske entitete. Kada se klikne za proširenje stabla prikazaće se lista svih roditeljskih entiteta. Rekurzivno, prikazaće se na kraju i kataloške tabele.

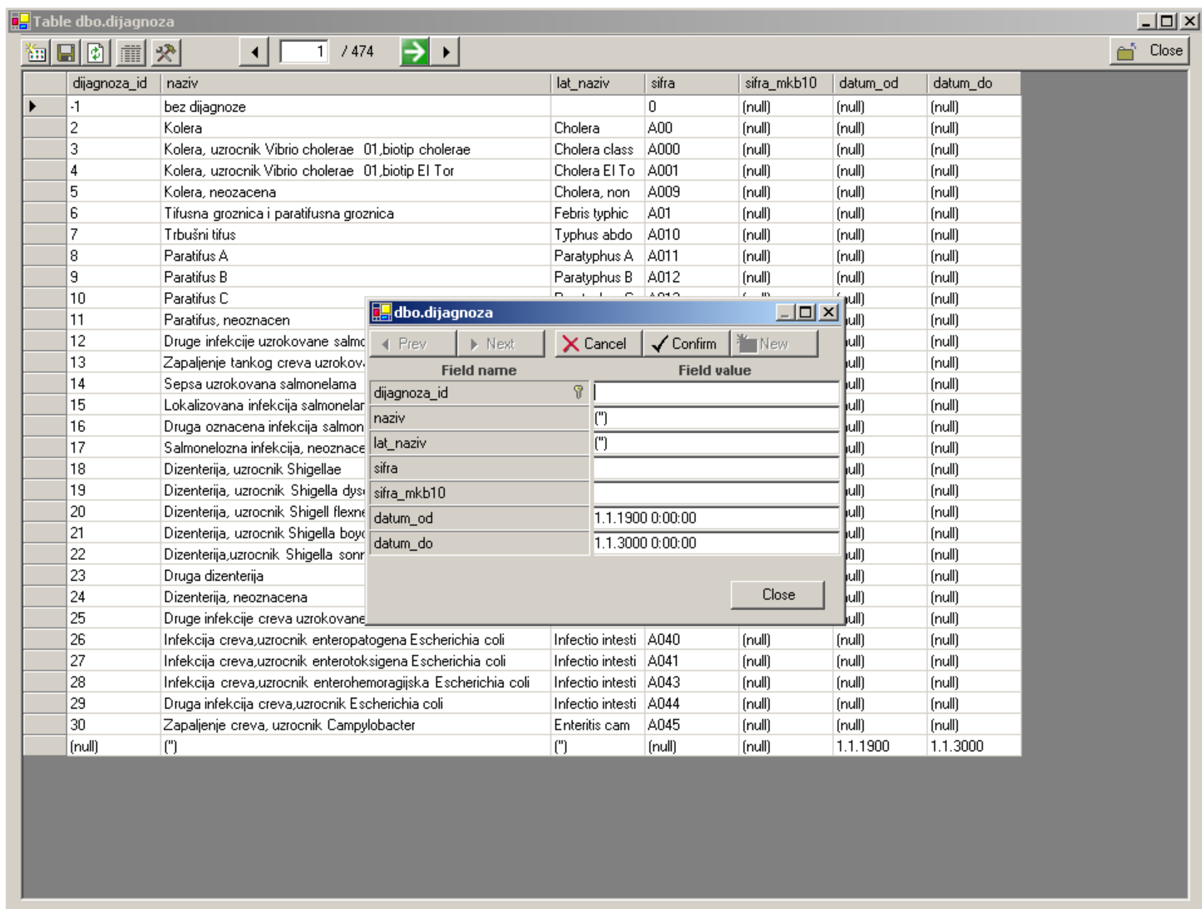


Slika 44 Generisani projekat u razvojnom okruženju Visual Studio

Ako se pogleda statička struktura generisane aplikacije, centralno mesto u njoj zauzima klasa DataContainer (Slika 46). Klasa DataContainer sadrži sve podatke i metode neophodne za manipulaciju podacima iz jedne izabrane tabele iz baze. U kreiranju objekta ove klase najpre se kreira njen CodeDom graf. Nakon toga se generiše kod na osnovu grafa i zajedno sa fiksnim delom za sve generisane aplikacije, snima se projekat nove generisane aplikacije.

Iako je navedeno grupisanje tabela u podstabla malo neuobičajen pristup za administratorsku aplikaciju (uglavnom se prikazuje samo lista tabela), način editovanja je standardan. Korisnik odabere jednu tabelu sa glavne forma (Slika 43) i odabere opciju Edit. Nakon toga se otvori forma koja sadrži izvestan broj redova iz tabele. Sadržaj tabele se prikazuje uz upotrebu straničenja, dok je veličina stranice podesive dužine.

Dupli klik na jedan red otvara formu za unos, odnosno editovanje selektovanog reda. Ovde se u vidu PropertyGrid-a, istog onog koji je korišćen kod alata za modelovanje, prikazuju vrednosti svih polja iz selektovanog reda. Pomenuti PropertyGrid je deo forme FormDetails koja predstavlja GUI formu za ažuriranje vrednosti u izabranom redu. Za obradu podataka koji će biti prikazani koristi se klasa DataTableAbstractor, koja je referencirana kroz formu FormDetails. Ona inicira učitavanje podataka i iterira kroz niz kolona koje sadrži odabrana tabela i za svaku od njih definiše odgovarajuću instancu DataColumnAbstractor-a u zavisnosti od tipa podataka. Kroz ovu klasu uvode se ograničenja specifična za svaki posebni tip podataka i na taj način se smanjuje margina za loš unos. Sve klase koje nasleđuju DataColumnAbstractor specijalizovane su za prikaz odgovarajućeg tipa podataka.



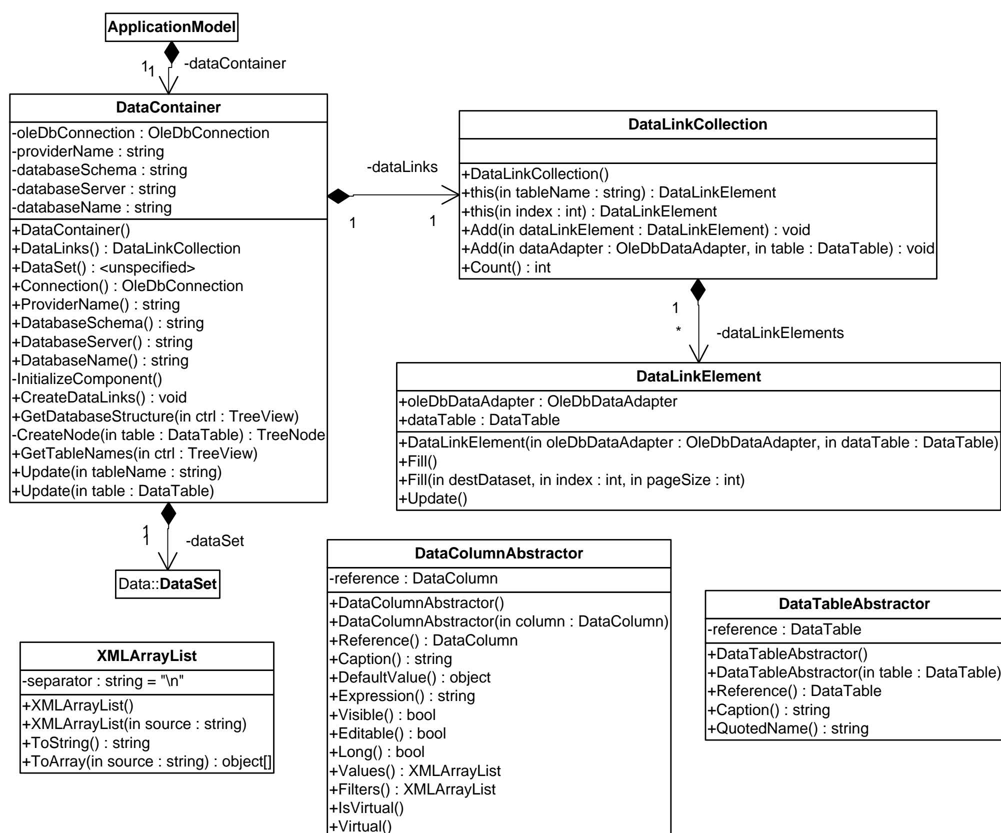
Slika 45 Editovanje podataka iz odabrane tabele kroz automatski generisanu formu

Klasa DataTableAbstractor je mesto gde se u generisanu aplikaciju dodaju veze ka formama kreiranim u drugim bibliotekama. Da bi se druga forma otvorila ona se mora adaptirati u sistem klasa administratorske aplikacije. To znači da je potrebna jedna klasa koja će referencirati novu formu, naslediti klasu FormDetails i predefinisati sve značajne metode u njoj. Zahvaljujući tome, omogućeno je jednostavno proširenje generisane aplikacije i njena kasnija primena kao probnog stola za generisane grafičke komponente.

Administratorska aplikacija je rezultat procesa generisanja aplikacije kojom upravlja generatorski alat i služi da upravlja podacima baze za koju je generisan. Jedna administratorska aplikacija je specifična i podešena za samo jednu bazu, i za svaku novu bazu se mora generisati nova aplikacija.

Sve tri komponente softvera (Slika 41) su implementirane u jeziku C#. Generatorski alat generiše kod za aplikaciju Database Administrator takođe u jeziku C#, ali se može jednostavno podesiti da generiše kod u bilo kom .NET jeziku (Visual Basic .NET, C++ ili J#). Skup svih funkcionalnosti paketa DatabaseStructure je dostupan administratorskoj aplikaciji, pošto i ona uključuje referencu na pomenutu biblioteku.

Svaki put kada se Database Administrator aplikacija pokrene i nakon što se uspešno aplikacija poveže na bazu, inicira se proces verifikacije baze. Proces verifikacije treba da ispita da li se struktura date baze poklapa sa strukturom baze koja je korišćena za generisanje Database Administrator aplikacije (originalne baze). Ako je struktura ostala nepromenjena administratorska aplikacija će prikazati stablo sačinjeno od tabela iz izabrane baze. Ako se strukture baza ne poklapaju, generisaće se poruka o odgovarajućoj grešci. Biće prikazana greška ili upozorenje u zavisnosti od toga koliko je uočena greška opasna. U zavisnosti od toga, program će nastaviti svoj rad ili će se završiti.



Slika 46 Dijagram klasa generisane aplikacije

Na primer, ako je nova tabela dodata u bazu biće izbačeno samo upozorenje i korisnik će nastaviti sa radom, s tim što novu tabelu neće moći da vidi u strukturi stabla u administratorskoj aplikaciji. U ovom slučaju, aplikacija na dalje radi normalno kao i pre.

U slučaju brisanja tabele, ili oduzimanja privilegija korisniku nad nekom tabelom, program nastavlja rad nakon obaveštenja o grešci, ali u granama u kojima je bila pomenuta tabela onemogućena je bilo kakva interakcija sa podacima u bazi. U slučaju neadekvatnih privilegija korisnika nad bazom, aplikacija će se sama ugasiti. U sledećim slučajevima DatabaseAdministrator aplikacija generiše poruke o greškama i indikovane promenama u strukturi baze:

- Promenjena imena tabelama – nastavak rada uz gašenje grana gde su se tabele javljale.

- Obrisana tabela koja ne učestvuje ni u jednoj relaciji – nastavak rada, i obrisana tabela se neće pojaviti u stablu.

- Imena ili tipovi nekih kolona promenjeni – nastavak rada; tabelama kojima se to desilo biće onemogućeno menjanje podataka.

- Nove kolone dodate u tabelu – nastavak rada; tabelama kojima se to desilo biće onemogućeno menjanje podataka.

- Kolone koje nisu deo primarnog ili spoljnog ključa su obrisane – nastavak rada; tabelama kojima se to desilo biće onemogućeno menjanje podataka.

- Obrisana je relacija između tabela – nastavak rada uz gašenje grana gde su se tabele javljale.

- Dodata nova relacija između tabela – nastavak rada uz gašenje grana gde su se tabele javljale.

- Dodata nova kolona postojećem primarnom ključu – nastavak rada; tabelama kojima se to desilo biće onemogućeno menjanje podataka.

Inicijalno definisana administratorska aplikacija ima generičke forme za rad sa svim tabelama u bazi. Na početku postoji samo jedna komponenta koja učitava strukturu tabele i u toku izvršenja generiše i prikazuje formu za ažuriranje podataka. Kada se kroz proces razvoja sistema kreiraju posebne forme za editovanje podataka, administratorska aplikacija se može proširiti tako da koristi njih.

Administratorska aplikacija je dizajnirana da radi sa imenovanim dataset objektima. To znači da podržava koncepte koje podržavaju i objektni modeli podataka, a to znači po jedan atribut po polju iz tabele. Zahvaljujući tome, moguće je definisati automatska preslikavanja iz objekataka koji donose generisane forme a koji dolaze iz objektnih modela kao što su NHibernate ili EntityFramework.

Ono što je u tom slučaju potrebno je u konfiguraciji aplikacije definisati preslikavanje koje sa jedne strane ima tabelu, a sa druge formu koja treba da edituje određeni slog. Da bi se ovo uspešno izvršilo, neophodno je dodati još jednu adaptersku klasu između generičke i konkretne forme. Ona treba da nasledi generičku formu, i da referencira konkretnu. Ove adapterske klase najčešće dolaze zajedno sa formom za editovanje podataka i generiše ih alat za generisanje softverskih komponenata. Ovom alatu je posvećeno celo naredno poglavlje.

3.4 Tačke proširenja

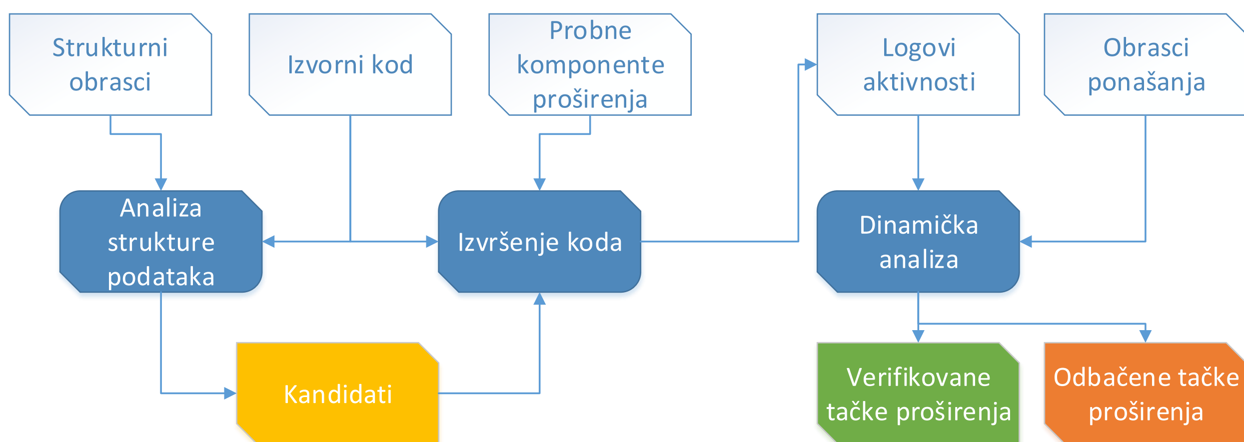
Tačke proširenja sistema predstavljaju mesta kod kojih je moguće, na prihvatljivo složen način, dodavati nove funkcionalnosti u sistem bez narušavanja njegove opšte stabilnosti. Postoje dve glavne vrste tačaka proširenja – tačke proširenja predviđene arhitekturom sistema i naknadno izdvojene tačke proširenja.

Tačke proširenja predviđene arhitekturom su značajno povoljnija situacija sa stanovišta razvoja softvera, pošto je za njih kroz arhitekturu sistema uvedena podrška u vidu različitih interfejsa čije metode treba implementirati u komponentama kojima se sistem proširuje. U narednim poglavljima biće prikazana arhitektura medicinskog informacionog sistema Medis.NET u čijoj strukturi su uključene tačke proširenja u skladu sa implementiranim modelom podataka. Sledeći nomenklaturu iz openEHR-a šabloni (templates) su funkcionalnosti koji imaju isti bazni skup podfunkcionalnosti. Taj bazni skup zajedno sa predefinisanim interfejsom predstavlja tačke proširenja. Za svaku od tačaka proširenja se zatim definiše šablonska komponenta, i na osnovu nje kasnije alat za generisanje koda

kreira finalne softverske komponente i uključuje ih u odgovarajuće biblioteke. Ovaj proces je detaljno opisan u sledećem poglavlju.

Sa druge strane proces izdvajanja tačaka proširenja je složeniji i uključuje kako ispitivanje statičke strukture sistema, tako i analize izvršenja. Proces traženja i verifikacije tačaka proširenja može da bude izuzetno složen proces (Slika 47), ali za uzvrat se pronalaze mesta u sistemu od kojih je moguće jednostavnije dodavati nove funkcionalnosti koje dele osnovni skup akcija. Ovaj proces je karakterističan za informacione sisteme koji se koriste dugi niz godina, i za koje korisnici zahtevaju izvesnu dogradnju.

U ovom procesu je nekada nemoguće koristiti kod stare aplikacije, tako da se o direktnoj dogradnji vrlo često ne može govoriti. Međutim, ono što je moguće je analizirati strukturu baze podataka, kao i strukturu zapisa u njoj, i na osnovu toga doneti izvesne zaključke koji će pomoći u definisanju kriterijuma za tačke proširenja. U slučaju kada se ne može direktno dograđivati stara aplikacija, projekat dogradnje se mora tritirati kao posebni projekat i kao početna tačke se onda može koristiti administratorska aplikacija.



Slika 47 Proces izdvajanja tačaka proširenja

U traženju tačaka proširenja za informacioni sistem čiji je kod nedostupan, početni korak je analiza strukture podataka. Generalno, ispituje se struktura baze i pronalaze se tabele sa određenim karakteristikama. Zatim se pomoću posebne test aplikacije i njenih predefinisanih komponenata generiše aplikacija koja se može koristiti za administriranje baze a koja će kroz probne komponente proširenja i izvršenje koda generisati logove aktivnosti na osnovu kojih se kasnijom dinamičkom analizom mogu izdvojiti najpogodnije tačke proširenja.

Ideja je da se najpre analizom veza u bazi podataka izdvoje tabele koje po određenim kriterijumima imaju potencijal za definisanje sličnih ili entiteta-dece. S obzirom da baze mogu da imaju više stotina tabela, detaljna analiza za svaku od njih bi trajala predugo.

Kada se izdvoje potencijalne tačke proširenja, pristupa se kreiranju probnih komponenti proširenja. One se mogu razvijati nezavisno, ili se mogu automatski generisati kroz alat za generisanje komponenata (ovaj alat je detaljno predstavljen u sledećem poglavlju). Za sva izvršenja koda generišu se logovi aktivnosti kako bi se videlo koliko nove komponente utiču na opterećenje sistema – koliko doprinose utrošku memorije a koliko zauzetosti procesora. Takođe, ispituje se i koliku količinu ostalih podataka učitavaju tokom svog rada. Primer generisanih logova aktivnosti dat je na kraju poglavlja Razvoj aplikacije pomoću generisanih komponenata.

Na osnovu logova aktivnosti i definisanih obrazaca za testiranje, dobija se detaljnija slika o definisanim komponentama i izdvojenim tačkama proširenja. Na osnovu kompletnih rezultata testiranja, donosi se odluka da li je izdvojena tačka proširenja pogodna ili ne.

Alati za analizu strukture baze i za generisanje administratorske aplikacije, prikazani u prethodnim sekcijama predstavljaju osnovu za traženje potencijalnih tačaka proširenja. U nastavku će biti prikazan primer traženja tačaka proširenja nad sistemom definisanim po industrijskom ISA 95 standardu, čiji je jezik za modelovanje B2MML (Business To Manufacturing Markup Language). B2MML predstavlja standard za definisanje modela proizvodnih sistema i kao takav se koristi u mnogim granama proizvodnje. Standardna baza kreirana na osnovu B2MML modela je reda veličine 250 do 300 tabela i sadrži oko 700 referencijalnih integriteta. Takođe, ovakve baze najčešće imaju relativno visok priraštaj podataka reda veličine 100MB dnevno.

Prvi korak u traženju tačaka proširenja je definisanje strukturnih obrazaca koje treba tražiti u strukturi baze podataka. Biblioteka za analizu baze može da učita strukturu i da prateći definisane strukturne obrasce pronađe tabele u bazi koje sadrže definisani obrazac. Biblioteka trenutno podržava pretraživanje po nekoliko glavnih kriterijuma koji se mogu povezati logičkim operatorima, a podržana je i mogućnost proširenja skupa kriterijuma. Definisanje kriterijuma je bazirano na principima za pisanje upita primenjenih u NHibernate ICriteria sistemu. Ideja je bila iskoristiti poznatu sintaksu kako bi programeri lakše pisali svoje upite (Slika 48).

```
IList cats = session.CreateCriteria(typeof(Cat))
    .Add( Expression.Like("name", "Iz%") )
    .Add( Expression.Gt( "weight", minWeight ) )
    .AddOrder( Order.Asc("age") )
    .List();
```

Slika 48 Primer NHibernate ICriteria upita

U sintaksi za definisanje upita za traženje tačaka proširenja rezultati upita su donekle drugačiji. Umesto niza zapisa iz neke tabele u bazi, upiti koji treba da ispitaju strukturu baze treba da vrate imena tabela i vrednosti odgovarajućih parametara. Ovo je ostvareno kreiranjem posebnih pogleda na sistemske tabele u okviru DBMS-a, i kreiranjem posebnog objektnog modela koji će tretirati njih kao posebne klase. Izvršenje upita nad tim klasama daće kao rezultat podatke za strukturnu analizu.

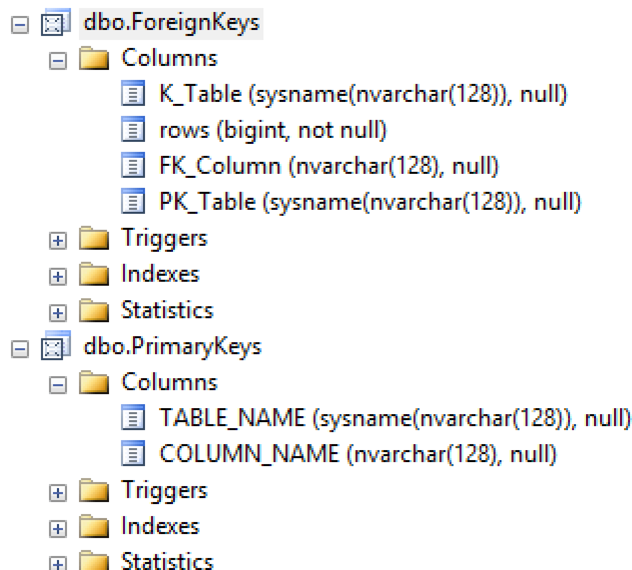
Najjednostavniji primer strukturnog obrasca je definisanje obrasca koji bi imao samo jedan element, a najkarakterističnija osobina je da li tabela ima referencijalni integritet prema drugim tabelama. Kao što je pomenuto ranije, najpre je potrebno definisati poglede koji bi prikazali odgovarajuće skupove tabela. Za ovu svrhu trebaju nam svi pogledi koje će objektni model tretirati kao klase – pogled na sve primarne(Slika 49). i pogled na sve spoljne ključeve (Slika 50). Slika 51 prikazuje kreirane poglede zajedno sa svojim poljima koji će biti preslikani u atribute klase objektnog modela.

```
SELECT   i1.TABLE_NAME, i2.COLUMN_NAME
FROM     INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS i1 INNER JOIN
           INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS i2 ON i1.CONSTRAINT_NAME = i2.CONSTRAINT_NAME
WHERE    (i1.CONSTRAINT_TYPE = 'PRIMARY KEY')
```

Slika 49 Upit koji vraća sve primarne ključeve

```sql
SELECT DISTINCT FK.TABLE_NAME AS K_Table, p.rows, CU.COLUMN_NAME AS FK_Column, PK.TABLE_NAME AS PK_Table
FROM        INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS AS C INNER JOIN
             INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS FK ON C.CONSTRAINT_NAME = FK.CONSTRAINT_NAME INNER JOIN
             INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS PK ON C.UNIQUE_CONSTRAINT_NAME = PK.CONSTRAINT_NAME INNER JOIN
             INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS CU ON C.CONSTRAINT_NAME = CU.CONSTRAINT_NAME CROSS JOIN
             sys.tables AS t INNER JOIN
             sys.indexes AS i ON t.object_id = i.object_id INNER JOIN
             sys.partitions AS p ON i.object_id = p.object_id AND i.index_id = p.index_id
```

Slika 50 Upit koji vraća spoljne ključeve sa brojem referenci



Slika 51 Polja u kreiranim pogledima koja će jednoznačno biti preslikana u objektni model

```csharp
var criteria = CurrentSession.CreateCriteria(typeof(ForeignKeys))
            .CreateAlias("PrimaryKeys", "s", JoinType.InnerJoin)
            .Add(Restrictions.Eq("s.TableBame", ForeignKeys.K_Table));
            .AddOrder( Order.Desc(ForeignKeys.rows) )
```

Slika 52 Upit koji vraća sve tabele koje su spoljni ključ

Nakon kreiranih pogleda, jednostavno je kreirati upit koji će vratiti listu sa svim tabelama koje su roditeljski entitet drugim tabelama. Slika 53 prikazuje naziv tabele i broj tabela nad kojima je uspostavljena veza.

Ovakav upit, iako je jednostavan i vraća sve tabele iz baze, vratiće ih uređene na način da će tabele koje imaju najviše zavisnih tabela biti pri vrhu. U tu kategoriju spadaju tabele koje zaista učestvuju u tačkama proširenja i tabele koje su jednostavno kataloške tabele koje se referenciraju na mnogo mesta, kao što je na primer tabela za jedinice mere (mesto br 10, Slika 53). Koristeći podatke iz ova dva pogleda i navedenu sintaksu moguće je kreirati i složenije upite koji mogu na primer da vrate sve tabele koje imaju odgovarajući skup roditeljskih entiteta i u isto vreme su one same nekome roditelj.

U skladu sa tabelom koju prikazuje Slika 53, svi rezultati pretrage bi mogli da budu tačke proširenja. Ovde je prikazano samo prvih 10, ali zapravo cela lista ima 181 element. Ovako definisano strukturno pravilo daje širok skup rezultata i gledano na taj način ne predstavlja najbolji izbor. Ovde treba izabrati pravu meru složenosti pravila, pošto suviše složeno pravilo se izvršava previše dugo, a suviše jednostavno daje previše rezultata.

Sledeće pravilo koje se može primeniti je da se tabele koje daju spoljni ključ urede po ukupnom broju redova u zavisnim tabelama (Slika 54). Na ovaj način se vidi kolika količina podataka je zavisna od odgovarajuće tabele. U ovom slučaju javlja se gotovo upola manje kandidata – 98, ali i to bi bilo previše za analizu.

	PrimaryKeyTable	RefCount
1	Person	51
2	EquipmentInstance	37
3	MaterialItem	37
4	ProductSegment	35
5	Product	23
6	SegmentResponse	19
7	ProductType	18
8	ProductionOrder	14
9	WfNodes	14
10	MaterialUOM	14

Slika 53 Lista sa deset tabela iz B2MML baze koje su najčešće bile roditeljski entitet u "spoljni ključ" vezi

	PrimaryKeyTable	TotalRows
1	MaterialMove	30700466
2	MaterialItem	23928304
3	SegmentResponse	15607508
4	Person	10532024
5	EquipmentInstance	10114826
6	Addresses	9750366
7	ProductSegment	1621254
8	MHTransportOrder	1165596
9	ProductionResponse	815286
10	BomItem	740446

Slika 54 Prvih deset tabela po broju zapisa u njihovim tabelama-deci

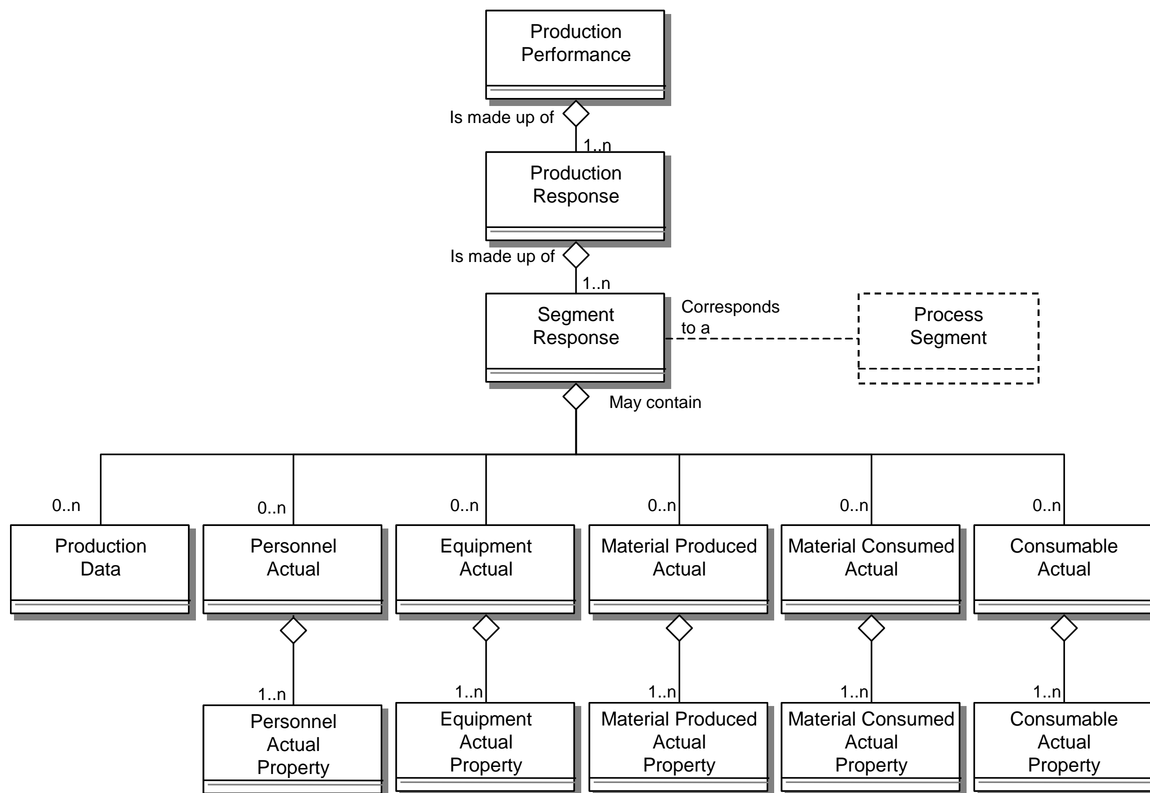
U ovom primeru biće prikazana analiza nad skupom jednostavnih pravila koja brzo mogu da daju dobar rezultat. Za kreiranje boljih pravila ipak je neophodno poznavanje domena za koji je baza definisana, ali samo ova pravila, uz par dodatnih restrikcija daju prilično dobre rezultate u opštem slučaju. Dakle, ovde su upotrebljena samo dva opšta pravila, a sledeće proširenje bi bilo uvođenje restrikcija. Rezultat toga bilo uvođenje heurističkih pravila za izbacivanje tabela, za koje se, u opštem slučaju, zna da ne mogu da budu tačke proširenja – to su na primer:

- Kataloške tabele. Tabele koje nemaju roditeljske entitete, ali se javljaju kao spoljni ključ u mnogo različitih tabela. Primer takvog kataloga je tabela sa jedinicama mere
- Pseudokatalozi. Tabele koje imaju roditeljski entitet u kome se definiše tip, a koje opet daju referencijalni integritet brojnim drugim tabelama. Takva je na primer tabela ProductSegment.

- Tabele sa puno zapisa koje od zavisnih entiteta imaju samo slabe entitete ili evaluirane slabe entitete.

	Ime tabele
1	SegmentResponse
2	Person
3	EquipmentInstance
4	Addresses
5	ProductionResponse
6	ProductionOrder
7	Product
8	GoodsReceivedOrderLine
9	ProductionPerformance
10	WarehousePackage
11	WarehouseElement
12	GoodsReceivedOrder
13	CustomerOrder
14	ExtProductionRule
15	Document
16	Interruption

Slika 55 Sve potencijalne tačke proširenja u B2MML bazi nakon proširenja skupa pravila



Slika 56 Deo B2MML modela koji se odnosi na zapisivanje podataka o proizvodnji (ProductionPerformance šema u okviru B2MML-a)

Kada se samo ova tri dodatna pravila primene dobije se skup od 16 tabela, od kojih neke čine grane u istim stablima, pa se uslovno mogu tretirati kao tačke proširenja. Tako na primer ProductionResponse i ProductionPerformance čine istu granu sa SegmentResponse tabelom. Tabela GoodsReceivedOrder je u istoj grani kao i GoodsReceivedOrderItem, a isto je i sa CustomerOrder i ProductionOrder.

Tabele Person, koja pamti podatke o korisniku, i EquipmentInstance, koja pamti podatke o konkretnim mašinama, se mogu posmatrati zajedno kao takozvane tabele potpisa. Oko njih se gotovo nikad ne gradi zvezda nasleđivanja, ali su isto vreme spoljni ključevi mnogim tabelama koje prikupljaju podatke o proizvodnji.

Preostale tabele su značajne kao tačke proširenja. Kada se strukturnom analizom generišu kandidati za tačke proširenja, prelazi se na njihovo testiranje i verifikaciju. Za tu svrhu se koriste probne generisane komponente i administratorska aplikacija kao okruženje za testiranje. Kako bi se testiranje sprovelo kreira se probni domenski model koji ima po nekoliko entiteta za testiranje svake od tačaka proširenja. Testiranje probnih generisanih komponenti daje rezultate i logove aktivnosti koji se dalje evaluiraju pa se na kraju odlučuje koje će tabele biti proglašene tačkama proširenja a koje ne.

Prvi kandidat za tačku proširenja je tabela SegmentResponse. Ona je u B2MML modelu glavna klasa koja pamti informacije na odgovarajućem segmentu proizvodnje (Slika 56). Kao što se vidi na dijagramu, već u osnovnom modelu, šest različitih entiteta koristi SegmentReponse za svoj bazni entitet. U zavisnosti od vrste proizvodnje, može se po potrebi definisati i niz dodatnih entiteta koji će dodatno specijalizovati SegmentResponse za neku određenu primenu.

Daljom analizom izvršavanja može se videti da tabela SegmentResponse ima veliki broj zapisa i da se skoro svaki put snimaju podaci u sve zavisne tabele. Ovo potencijalno može da bude opasnost i da prilikom dodavanja novih entiteta kombinacija složenosti strukture i velikog broja zapisa otežа izvršavanje upita.

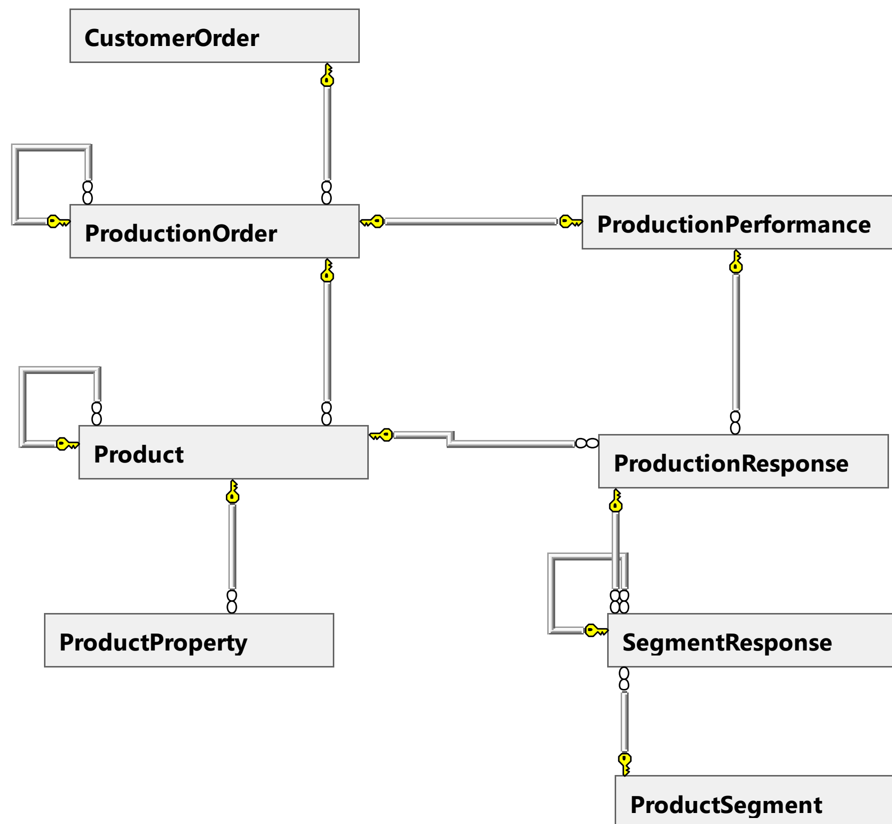
Zbog ovoga, sama tabele SegmentResponse nakon dinamičke analize može biti odbačena kao inicijalna tačka proširenja, a da se kao potencijalne nove pojave tabele koje su njoj zavisne. Takav primer je MaterialConsumedActual. Ova tabela može da ima različite skupove atributa u zavisnosti od primene. Kada ih nema mnogo, taj problem se rešava tabelom MaterialConsumedActualProperty, ali u slučaju kada se po svakom segmentu pamti po stotinak dodatnih atributa, izvedeni entiteti iz MaterialConsumedActual su bolje rešenje.

Ovo je dobar primer koji pokazuje da se ne može potpuno osloniti na strukturnu analizu, i da je analiza ponašanja sistema jednako značajna. U konkretnom primeru se pokazuje da je prava tačka proširenja ne entitet koji je dobijen statičkom analizom, već njegov podentitet.

Dalje, sledeći kandidati za tačke proširenja su Person i EquipmentInstance. Kao što je već pomenuto, oni su entiteti koji služe da „potpišu“ razne podatke, i samim tim u konkretnom domenu ne predstavljaju entitete oko kojih se sistem proširuje novim entitetima. Tabela EquipmentInstance ima određeni potencijal da, po istom principu kao i MaterialConsumedActual, bude tačka proširenja. Međutim, u praksi je to veoma redak slučaj pošto se elementi opreme u B2MML bazi definišu najjednostavnije moguće, pošto ovakvi sistemi najčešće rade zajedno sa kontrolerima koji upravljaju mašinama i koji čuvaju sve podatke o njima.

Sledeći kandidat za tačku proširenja je entitet koji se zove Addresses. On služi za pamćenje kontakt podataka različitih učesnika u sistemu. Ovaj entitet se našao ovako visoko na listi zato što učestvuje kao spoljni ključ u više različitih tabela, a u isto vreme sve nove vrste komunikacije se modeluju kao

entiteti koji nasleđuju njega. U tom svetlu, entitet Addresses predstavlja pravu tačku proširenja sistema.



Slika 57 Veza između ProductionOrder i ProductionPerformance delova B2MML modela

Entiteti ProductionOrder i Product su još dva značajna kandidata. Oni su bazni entiteti za definisanje naloga za proizvodnju i posebnih proizvoda (Slika 57). Vezani su sa ProductionPerformance stablom i predstavljaju validne tačke proširenja. U velikom broju proizvodnih sistema postoji potreba da se definišu posebni tipovi proizvodnih naloga (ProductionOrder) i proizvoda (Product) koji imaju potpuno drugačiji skup parametara. U zavisnosti od broja i raznovrsnosti dodatnih atributa ovih entiteta oni će biti korišćeni kao tačke proširenja. Konkretno u proizvodnim procesima gde se proizvodi širok skup raznih proizvoda postoji veća verovatnoća da je potrebno proširivati sistem kod njih.

WarehouseElement i WarehousePackage su entiteti koji dolaze iz dela B2MML modela koji opisuje materijale i skladišta. WarehousePackage je entitet koji je kreirao evolucijom i integracijom LOT i sub-LOT enitieta. On opisuje bilo kakav fizički kontejner koji direktno može da sadrži materijal u sebi. WarehouseElement opisuje elemente fizičkih skladišta. Kako tih elemenata, kao i paketa koji direktno sadrže materijal ima mnogo vrsta, tabele u kojima se čuvaju ovi entiteti predstavljaju validne tačke proširenja.

ExtProductionRule, Document i Interruption su takođe značajni kandidati za tačke proširenja. ExtProductionRule čuva podatke o pravilima za proizvodnju, koja se preuzimaju sa spoljnih sistema za upravljanje resursima. Prilično se razlikuju od jednog do drugog takvog sistema, ali u okviru jedne implementacije informacionog sistema, najčešće budu definisana na isti način. Tako, da potencijal ovog entiteta zavisi od konkretne realizacije sistema.

Document je bazna klasa za sve dokumente i predstavlja pravi prototip tačke proširenja, na isti način kao i rezultati pregleda u MIS sistemima. Dokumenti po strukturi su najčešće toliko različiti da osim opisa, imena i jedinstvene identifikacije nemaju gotovo ništa zajedničko među sobom.

Na kraju, entitet Interruption ne predstavlja mesto proširenja, pošto se u ovoj tabeli našao kao nulabilni spoljni ključ mnogih entiteta koji opisuju rezultate proizvodnje. Uglavnom se definišu preko tipova i vrlo retko se iz njih izvode podentiteti.

Ono što je karakteristično za tačke proširenja je to da su one mesto od koga se funkcionalnosti granaju i za njih se može reći da topološki predstavljaju središte zvezde. Izuzetno je značajno uočiti statičkom analizom sve potencijalne kandidate, a onda dodatnim analizama izvršenja koda na osnovu probnih domenskih modela aplikacije verifikovati prave i odbaciti one koje nisu toliko značajne.

3.5 Šablonske komponente

Šablonske komponente predstavljaju osnovu za kreiranje generisanih komponenti. Šablonska komponenta je zapravo standardna softverska komponenta koja sem standardnog koda u nekom programskom jeziku sadrži i posebne komentare koji će u procesu generisanja koda biti zamenjeni generisanim kodom. Izgled komentara ne mora nužno biti jedinstven u okviru celog sistema, pošto svaka klasa za generisanje specifičnih komponenti može biti razvijana nezavisno. U primerima koji će biti pokazani u okviru ove sekcije, specifični komentari sadrže literal oivičen specijalnom kombinacijom karaktera – za početak <# i za kraj #>.

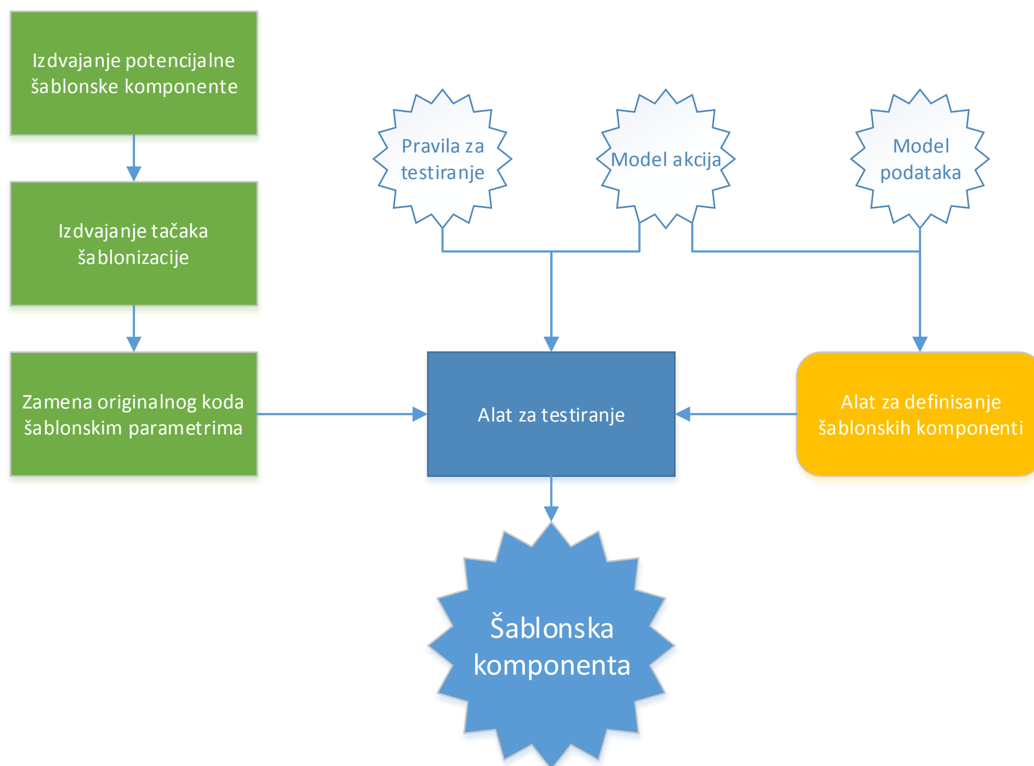
Šablonska komponenta može da bude čak i potpuno funkcionalna komponenta koja sadrži delove koda koji su zajednički svim budućim generisanim komponentama. Kod minimalističkog MDE pristupa taj zajednički skup funkcionalnosti čine najčešće operacije za učitavanje, ažuriranje i snimanje novih entiteta, kao i za navigaciju kroz master-detail prikaz. U slučaju korišćenja šablonskih komponenti, bilo koja komponenta koja je korektno napisana i testirana i koja ima specifične komentare koji će poslužiti kao njene validno definisane tačke proširenja može biti korišćena za generisanje novih komponenti.

Postoje, generalno, dva načina za definisanje šablonskih komponenti. Jedan je kreiranje šablonske komponente u nekom eksternom alatu koji je baziran na meta modelu podataka, modelu akcija i skupu pravila za testiranje. Ovo je mnogo opštiji slučaj i daje rešenja koja bi teoretski mogla da budu uključena bilo gde. Problem sa ovim pristupom je to što je generisanje skupa pravila za testiranje i sam proces testiranja može da traje nepredvidivo dugo iz prostog razloga što šablonska komponenta može biti bilo šta.

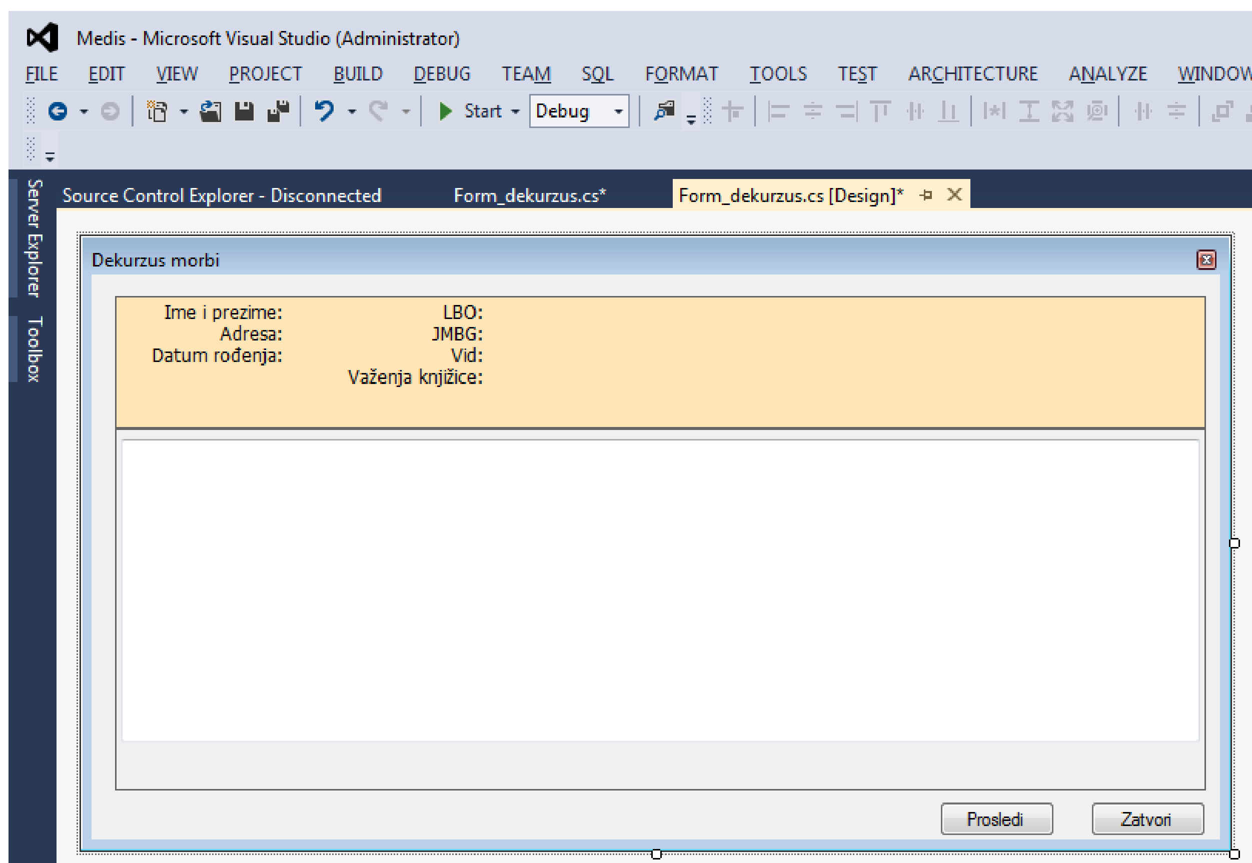
Efikasniji, ali manje generalni pristup, je kreiranje šablonske komponente na osnovu već postojećih i aktivnih komponenti razvijenih kroz proces razvoja informacionog sistema. Ovakve komponente su već testirane kroz standardni proces razvoja informacionog sistema i svaki dalji rad na njima je u velikoj meri olakšan tom činjenicom. Takođe, programeri će na njima raditi u poznatom okruženju i poznatoj tehnologiji što će dodatno doprineti efikasnijem razvoju.

Prvi korak u ovom procesu je izdvajanje potencijalne šablonske komponente. Potencijalna šablonska komponenta je tesno povezana sa tačkama proširenja sistema, i glavna ideja je da se kroz analizu koda pronađe komponenta koja u sebi ima implementirane deljene funkcionalnosti vezane za tačku proširenja. Nakon toga se, svi specifični entiteti menjaju opštim, a specifičan kod se menja posebnim komentarima koji će u procesu generisanja koda biti zamenjeni generisanim kodom.

Ovo je pristup kada se radi sa sistemima gde je celokupan kod dostupan. U slučaju da treba razviti dodatak sistemu kad postojeći kod nije vidljiv, inicijalno se kreira jedna komponenta koja će podržati rad sa jednim entitetom koji je direktno vezan za tačku proširenja. Kada se razvijena komponenta testira i verifikuje, moći će da se započne sa njenom transformacijom u šablonsku komponentu koja će biti dalje korišćena za generisanje novih komponenti.



Slika 58 Proces definisanja šablonskih komponenti



Slika 59 Primer potencijalne šablonske komponente

U nastavku ove sekcije biće prikazan proces kreiranja šablonske komponente na osnovu jedne koja je definisana oko tačke proširenja sistema i koja je kroz razvoj i korišćenje testirana i verifikovana. Tačka proširenja za koju će biti vezana buduća šablonska komponenta je entitet data_usluga iz MIS Medis.NET.

Kao potencijalna forma za transformaciju u šablonsku izabrana je forma za unos dekurzusa (Slika 59). Ova forma je potencijalna šablonska forma zato što ima funkcionalnosti koje su po dizajnu sistema zajedničke za sve vrste pregleda, a od posebnih komponenti ima samo polje za unos teksta. Forma ima zaglavlje gde se prikazuju podaci o pacijentu i podržava kreiranje novih dekurzusa kao i ažuriranje postojećih. Ovo je kvalifikuje za idealnog kandidata za šablonsku komponentu. Poseban deo je malog obima i jasno diferenciran, dok je opšti potpun i u okviru njega ne nedostaje ni jedna od baznih funkcionalnosti.

Forma za dekurzuse je razvijana kao Windows forma u .NET okruženju, tako da je celokupna implementacija klase već podeljena u dva fajla – u jednom su vizuelne komponente koje se javljaju na formi, njihovo instanciranje i definisanje vizuelnih atributa, dok su u drugom fajlu smeštene funkcije koje služe za obradu podataka. Slika 60 prikazuje deo koda koji je identifikovan kao specifični deo u formi za dekurzuse. Tu su deklaracije promenljivih koje predstavljaju specifične vizuelne komponente kao i referenca na objekat tipa dekurzus čiji će se atributi editovati ovom formom. Sem toga, ovde je potreban i jedan bidirekcioni properti koji će enkapsulirati referencirani entitet i omogućiti njegovo postavljanje i čitanje sa forme. Pomenuta forma radi sa entitetima tipa dekurzus i za učitavanje i snimanje atributa tih entiteta potrebne su posebne metode. Slika 61 prikazuje kod koji se koristi za tu svrhu.

```
25    public class Form_dekurzus : System.Windows.Forms.Form, IDocumentProperties
26    {
27
28        private System.Windows.Forms.Panel pnlData;
29
30        private System.ComponentModel.IContainer components = null;
31
32        private dekurzus pregled;
33
34        private Medis.Controls.CtrlOsiguranikInfo ctrlOsiguranikInfo1;
35
36        private Medis.Controls.CtrlLekarSelektor ctrlLekarSelektor1;
37
38        private System.Windows.Forms.Button btnSnimi;
39
40        private System.Windows.Forms.Button btnNazad;
41
42        private System.Windows.Forms.Label lbldekurzus_morbi;
43
44        private System.Windows.Forms.TextBox tbxdekurzus_morbi;
45
46        private System.Windows.Forms.Label lbldekurzus_morbiMU;
47
48        public Form_dekurzus(long patientId)
49        {
50            InitializeComponent();
51            this.ctrlOsiguranikInfo1.SelectOsiguranik(patientId);
52        }
```

Slika 60 Izdvojeni kod koji se odnosi na definiciju specifičnih vizuelnih komponenata

```
202        private void btnSnimi_Click(object sender, EventArgs e)
203        {
204            if (this.pregled == null)
205            {
206                this.pregled = new dekurzus();
207            }
208            this.pregled.dekurzus_morbi = tbxdekurzus_morbi.Text;
209        }
210
211        public void LoadValues()
212        {
213            this.tbxdekurzus_morbi.Text = this.pregled.dekurzus_morbi;
214        }
```

Slika 61 Izdvojeni specifični kod u funkcijama za učitavanje i snimanje entiteta tipa dekurzus

```
 96    this.pnlData.Size = new System.Drawing.Size(708, 235);
 97    this.pnlData.TabIndex = 0;
 98    //
 99    // lbldekurzus_morbi
100    //
101    this.lbldekurzus_morbi.AutoSize = true;
102    this.lbldekurzus_morbi.Font = new System.Drawing.Font("Tahoma", 9F);
103    this.lbldekurzus_morbi.Location = new System.Drawing.Point(50, 30);
104    this.lbldekurzus_morbi.Name = "lbldekurzus_morbi";
105    this.lbldekurzus_morbi.Size = new System.Drawing.Size(0, 14);
106    this.lbldekurzus_morbi.TabIndex = 0;
107    //
108    // tbxdekurzus_morbi
109    //
110    this.tbxdekurzus_morbi.Font = new System.Drawing.Font("Tahoma", 9F);
111    this.tbxdekurzus_morbi.Location = new System.Drawing.Point(3, 6);
112    this.tbxdekurzus_morbi.Multiline = true;
113    this.tbxdekurzus_morbi.Name = "tbxdekurzus_morbi";
114    this.tbxdekurzus_morbi.Size = new System.Drawing.Size(700, 197);
115    this.tbxdekurzus_morbi.TabIndex = 1;
116    //
117    // lbldekurzus_morbiMU
118    //
119    this.lbldekurzus_morbiMU.AutoSize = true;
120    this.lbldekurzus_morbiMU.Font = new System.Drawing.Font("Tahoma", 9F);
121    this.lbldekurzus_morbiMU.Location = new System.Drawing.Point(550, 30);
122    this.lbldekurzus_morbiMU.Name = "lbldekurzus_morbiMU";
123    this.lbldekurzus_morbiMU.Size = new System.Drawing.Size(0, 14);
124    this.lbldekurzus_morbiMU.TabIndex = 2;
125    //
126    // ctrlOsiguranikInfo1
127    //
128    this.ctrlOsiguranikInfo1.BackColor = System.Drawing.SystemColors.Control;
```

Slika 62 Specifični kod koji se odnosi na postavljanje adekvatnih vrednosti u dizajneru forme

Slika 62 je deo koda iz metode InitializeComponent. U .NET win forms aplikacijama ova metoda je sastavni deo svake klase koja predstavlja formu i u njoj je smešten kod koji detaljno opisuje sve vizuelne komponente i postavlja sve njihove atribute na inicijalne vrednosti. Ovde se mogu videti kako su definisani atributi najpre za jedan natpis (blbdekurzus_morbi) uključujući font, poziciju na ekranu i

veličinu. Za njim je definicija jednog polja za prikaz teskta (tbxdekurzus_morbi), koji uz pomenute atribute ima i atribut koji kaže da se u njega može uneti tekst u više redova (Multiline = true).

Nakon izdvajanja posebnih od opštih delova koda, počinje proces transformacije odabrane forme u šablonsku komponentu. Najpre se dodaju neophodni interfejsi i implementacije njihovih metoda kako bi se efikasno i uniformno mogle uklopiti u celu aplikaciju (Slika 63) Ovde je dodat jedan interfejs koji se naziva IDocumentProperties koji sadrži dve metode SetPatient i GetPregled.

```
216        #region IDocumentProperties Members
217
218        public void SetPatient(long patientId)
219        {
220            this.ctrlOsiguranikInfo1.SelectOsiguranik(patientId);
221        }
222
223        public IMedicalDocument GetPregled()
224        {
225            return this.pregled;
226        }
227
228        #endregion
```

Slika 63 Implementacija metoda iz dodatih intertfejsa

```
25     public class Form_sablon : System.Windows.Forms.Form, IDocumentProperties
26     {
27
28         private System.Windows.Forms.Panel pnlData;
29
30         private System.ComponentModel.IContainer components = null;
31
32         private Medis.Controls.CtrlOsiguranikInfo ctrlOsiguranikInfo1;
33
34         private Medis.Controls.CtrlLekarSelektor ctrlLekarSelektor1;
35
36         private System.Windows.Forms.Button btnSnimi;
37
38         private System.Windows.Forms.Button btnNazad;
39
40         ///<#DefaultFieldDefinition#>
41
42         ///<#DefaultPropertyDefinition#>
43
44         ///<#ControlsFromModelDefinition#>
45
46         public Form_sablon(long patientId)
47         {
48             InitializeComponent();
49             this.ctrlOsiguranikInfo1.SelectOsiguranik(patientId);
50         }
51
```

Slika 64 Posebne komponente zamenjene komentarima koji će biti tačke proširenja

Sledeći korak je zamena funkcionalnih celina koda posebnim komentarima koji će biti menjani u procesu generisanja novih komponenti.

U delu gde se definišu atributi forme izdvojene su tri celine (Slika 64):

- deklaracija privatnog atributa koji čuva referencu na odgovarajući objekat (DefaultFieldDefinition komentar)
- deklaracija bidirekcionog propertija koji služi da pomenuti atribut učini vidljivim van klase (komentar DefaultPropertyDefinition)
- deklaracija liste specifičnih kontrola (komentar ControlsForModelDefinition)

U ostatku koda izdvojeni su delovi koji će biti ugrađeni u odgovarajuće metode na formi (Slika 65). U metodi InitializeCompoent to su:

- instancijacija kontrola (komentar InstantiateControls)
- podešavanje atributa generisanih kontrola (komentar ControlsFromModelImplementation)
- dodavanje kreiranih kontrola u listu kontrola koje pripadaju formi (komentar AddControls)

Sem toga, kod se generiše i za metode za snimanje i učitavanje entiteta.

```
58   #region initialize and dispose components
59   private void InitializeComponent()
60   {
61       ///<#InstantiateControls#>
62
63       #region default components instantiation
86
87       ///<#ControlsFromModelImplementation#>
88
89       #region default components
137
138
139      ///<#AddControls#>
140
141      #region define form and default components
157
158  }
159
160
161  protected override void Dispose(bool disposing)
162  {
169  #endregion
170
171  private void btnSnimi_Click(object sender, EventArgs e)
172  {
173      ///<#CodeForSave#>
174  }
175
176  public void LoadValues()
177  {
178      ///<#CodeForLoad#>
179  }
180
```

Slika 65 Kod u opštim metodama koji će biti zamenjen u procesu generisanja

Nakon završenog procesa dobija se šablonska komponenta koja se može koristiti u procesu generisanja novih komponenti. Nakon ovoga treba definisati specifičnu klasu za generisanje koda čije će metode učitati odgovarajući entitet iz objektnog modela i u šablonskoj komponenti zameniti specifične komentare odgovarajućim kodom.

Prednost ovog pristupa je u tome što se ista šablonska komponenta može koristiti zajedno sa više različitih generatorskih klasa, a jedna generatorska klasa može da koristi više šablonskih komponenata. Na ovaj način se kasnije jednostavno može menjati izgled aplikacije, a za pojedine podgrupe komponenata se mogu defnisati potpuno drugačije komponente.

4 Alat za generisanje softverskih komponenata

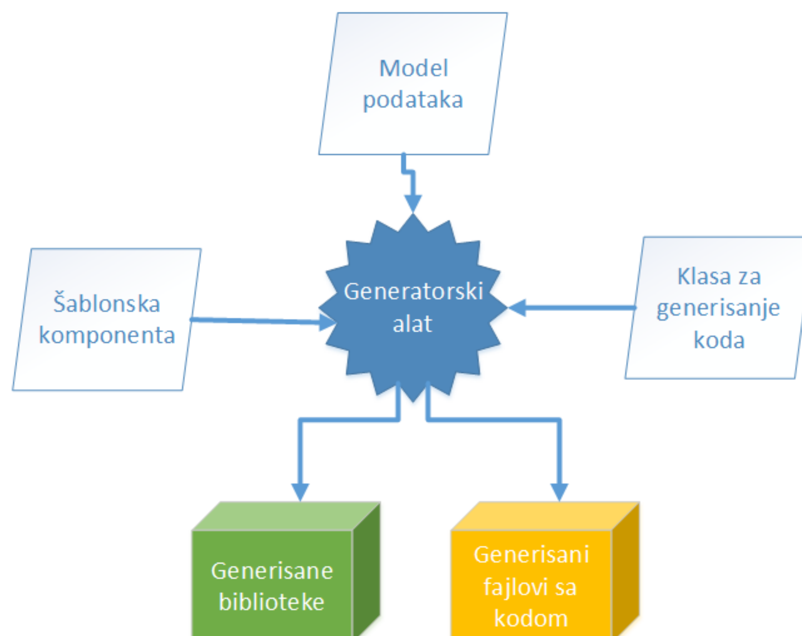
Kao što je već pomenuto, alat za modelovanje kreira model aplikacije, koji predstavlja ulaz za alat za generisanje komponenata. Alat za generisanje, inicijalno generiše tabele u bazi podataka, i na osnovu njih ažurira povezani objektni model. Takođe, alat podržava i ažuriranje baze i modela podataka na osnovu promenjenog objektnog modela.

Sem baze i objektnog modela, generatorski alat može da generiše i druge komponente koje se kasnije mogu uključiti u projekat u okviru razvojnog okruženja. U krajnjoj instanci, generatorski alat je u stanju da generiše odgovarajuće biblioteke i da kroz inverziju kontrole (IOC – inversion of control) ažurira odredišnu aplikaciju.

Na ovaj način se značajno ubrzava razvoj kao i procesi adaptacije i dogradnje informacionih sistema. Alat za generisanje se bazira na CodeDom biblioteci koja je standardni deo Microsoft-ovog .NET razvojnog okvira i podržava generisanje koda u svim .NET programskim jezicima. Nažalost, ne podržava generisanje koda za druge tehnologije. Međutim, ovo ograničenje važi samo za ugrađene funkcije, dok sve dodatne komponente mogu biti generisane pomoću bilo koje druge generatorske biblioteke.

Generatorski alat se povezuje na bazu koristeći ili OLE DB ili DatabaseStructure biblioteku koja ima predefinisanu podršku za MS SQL Server, Oracle i Postgree. Teoretski, moguće je koristiti bilo koju bazu koju podržava OLE DB, ali je onda potrebno uključiti odgovarajući spoljni alat za generisanje njihovog objektnog modela podataka.

Kada se radi o objektnim modelima, generatorski alat inicijalno podržava rad sa Entity Framework i NHibernate bibliotekama pošto su oni najrašireniji u praksi. Generalno, moguće je dodati bilo kakav novi objektni model, samo bi u tom slučaju korisnik morao sam da implementira svoju generatorsku klasu.



Slika 66 Način funkcionisanja generatorskog alata

Generatorski alat učitava model iz alata za modelovanje u kome su definisani entiteti za koje dalje treba generisati komponente (Slika 66). Za svaku vrstu komponenata, alat treba da učita šablonsku

komponentu sa odgovarajućim komentarima kao i uputstvo za generisanje (Slika 67). Pomenuto uputstvo može biti dato kao niz CodeDom naredbi, ili se može učitati odgovarajuća biblioteka koja sadrži metode za generisanje koda.

```
<CodeGenerationSettings>
    <ComponentToGenerate>
        <ComponentName>WinForm</ComponentName>
        <GenerationType>library</GenerationType>
        <DestinationFile />
        <DestinationNamespace />
        <TemplateComponent>Medis.TemplateComponents.Form_sablon</TemplateComponent>
        <CodeGenerationDefinition>
            <GenerationClass>Medis.TemplateComponents.FormCreator,Medis.TemplateComponents</GenerationClass>
            <CodeGenerationRules/>
        </CodeGenerationDefinition>
    </ComponentToGenerate>
    <ComponentToGenerate>
        <ComponentName>WinFormNeuro</ComponentName>
        <GenerationType>code</GenerationType>
        <DestinationFile />
        <DestinationNamespace />
        <TemplateComponent>Medis.TemplateComponents.Form_sablon</TemplateComponent>
        <CodeGenerationDefinition>
            <GenerationClass>Medis.Neuro.TemplateComponents.FormCreator,Medis.Neuro.Templates</GenerationClass>
            <CodeGenerationRules>
                <GenerationDefinition>
                    <MappingTag>DefaultPropertyDefinition</MappingTag>
                    <GenerationMapping>
                        <GenerationMethod>GenerateDefaultPropertyWithCheckOnSave</GenerationMethod>
                        <GeneratorObject />
                        <GenerationInstructions />
                    </GenerationMapping>
                </GenerationDefinition>
                <GenerationDefinition>
                    <MappingTag>AdditionalImports</MappingTag>
                    <GenerationMapping>
                        <GenerationMethod />
                        <GeneratorObject>samples</GeneratorObject>
                        <GenerationInstructions>
                            samples.Imports.Add(new CodeNamespaceImport("Medis.Neuro.Common"));
                            samples.Imports.Add(new CodeNamespaceImport("Medis.Neuro.Constants"));
                            samples.Imports.Add(new CodeNamespaceImport("Medis.Neuro.Reports"));
                        </GenerationInstructions>
                    </GenerationMapping>
                </GenerationDefinition>
            </CodeGenerationRules>
        </CodeGenerationDefinition>
    </ComponentToGenerate>
</CodeGenerationSettings>
```

Slika 67 Primer konfiguracije alata za generisanje

Slika 67 prikazuje primer konfiguracije koja definiše različite vrste uputstava za generisanje. Svi podaci vezani za jednu komponentu dati su u okviru atributa ComponentToGenerate. Tu je specificiran naziv komponente, vrsta generisanja, naziv fajla u koji se generiše kao i namespace u koji će biti uključen generisani fajl. Rezultat generisanja može da bude smešten i u više od jednog fajla, u zavisnosti od toga kako je šablonska komponenta definisana. Na ovaj način su podržani standardni načini za pisanje klasa u jezicima kao što su C++ (h i cpp fajl) i C# (partial class).

Naziv komponente (ComponentName) mora biti jedinstven u okviru konfiguracije. Komponente se po nazivu jedinstveno identifikuju u okviru skupa dostupnih opcija za generisanje. Za dodatne komentare se može koristiti polje ComponentDescription gde se može specifikovati sve što je dodatno neophodno za komponentu.

Vrsta generisanja je definisana poljem GenerationType i tu može biti upisana jedna od vrednosti code i library. Vrednost code označava generisanje fajlova sa izvornim kodom, a library generisanje biblioteka. Podrazumevana vrednost za ovo polje je code.

Ukoliko se kao vrsta generisanja specifikuje library, inicijalno podržana vrsta biblioteke je dinamički povezana biblioteka, ili dll. Kako generationType nasledjuje klasu sa kompajlerskim opcijama iz CodeDom okruženja, kroz dodatne atribute taga GenerationType mogu se uključiti različite kompajlerske opcije. Slika 68 prikazuje nekoliko primera definisanja dodatnih atributa.

```
<!-- generisanje izvršnog fajla-->
<GenerationType GenerateExecutable = "true">library</GenerationType>

   <!-- generisanje debug informacija-->
<GenerationType IncludeDebugInformation = "true">library</GenerationType>

   <!-- definisanje nivoa upozorenja i grešaka-->
<GenerationType WarningLevel = "3">library</GenerationType>

   <!-- definicija da li tretirati upozorenja kao greške-->
<GenerationType TreatWarningsAsErrors = "true">library</GenerationType>

   <!-- definisanje kompajlerskih opcija kao što je npr optimizacija -->
<GenerationType CompilerOptions = "/optimize">library</GenerationType>
   <GenerationType CompilerOptions = "/optimize+ /platform:x86 /target:winexe /unsafe">library</GenerationType>

   <!-- definisanje klase u kojoj je "main" metod-->
<GenerationType MainClass = "Samples.Class1">library</GenerationType>

<!-- uključivanje spoljnih resursa-->
<GenerationType EmbeddedResources = "Resources\\Default.resources">library</GenerationType>
```

Slika 68 Primeri definisanja dodatnih atributa uz tag generationType

DestinationFile i DestinationNamespace polja služe za definisanje imena fajlova i prostora imena u kojima će pripadati generisani elementi. Ukoliko se izostave (kao što je slučaj u datom primeru), generatorski alat će tražiti od korisnika da ih unese pre početka generisanja koda.

Sledeće što je potrebno generatorskom alatu je specifikacija šablonske komponente za šta se koristi atribut TemplateComponent. On čuva putanju do komponente koja sadrži komentare koji će biti zamenjeni u procesu generisanja koda (videti sledeću sekciju - Slika 64 i Slika 65). U sekciji CodeGenerationDefinition definiše se putanja do klase koja će generisati kod i zameniti svaki od komentara u kodu. Ukoliko je element CodeGenerationRule prazan, onda se podrazumeva da u specificiranoj generatorskoj klasi postoje metode koje se zovu isto kao i tagovi u okviru komentara za generisanje koda.

Dodatno, u okviru pomenutog CodeGenerationRules sekcije može se definisati koja metoda će generisati kod za koju vrstu komentara (polje GenerationMethod), ili se kroz tagove GenerationObject i GenerationInstructions može dati CodeDom kod koji će generisati odgovarajuće elemente.

U definiciji za WinFormNeuro prikazano je kako je predefinisana metoda za generisanje koda za DefaultPropertyDefintion. Generatorski alat će, kada bude generisao kod, za DefaultPropertyDefinition izvršiti kod iz metode označene u konfiguraciji – GenerateDefaultPropertyWithCheckOnSave.

Za komentare tipa AdditionalImports data je specifikacija koda kroz parametre GenerationObject i GenerationInstructions. GenerationObject definiše naziv osnovnog CodeDom objekta za generisanje koda. On je podrazumevanog tipa CodeNamespace i služi da čuva instrukcije za generisanje koda.

Kroz GenerationInstructions dodeljuje mu se lista naredbi. Ovaj objekat se uključuje u CodeCompileUnit koji je odgovoran za generisanje koda.

Pošto se kod generiše za neki entitet iz definisanog modela, sve klase i funkcije za generisanje se moraju definisati da rade za prosleđeni objekat tipa MedicalDataContainer.

4.1 Generisanje formi

Kako Windows forme predstavljaju glavni deo korisničkog interfejsa MIS sistema, proces njihovog generisanja doprinosi u velikoj meri bržem razvoju celokupnog sistema. Posebno se ovo ogleda u fazi kreiranja prototipa aplikacije, kada je posebno važno da put od kreiranja modela do generisanja prototipa bude što je moguće kraći.

U ovoj sekciji biće opisan proces generisanja formi koji je baziran na modelu podataka i standardnoj .NET CodeDom biblioteci. Iako alat za generisanje može da učita bilo koju klasu koja generiše kod, ovde je odabran CodeDom kako zbog kompatibilnosti sa ostatkom sistema tako i zbog velikog broja opcija za podešavanje i optimizaciju kako generisanog koda, tako i kompajliranih biblioteka.

Slika 69 prikazuje primer kako se može definisati jedan CodeDom objekat za generisanje koda kroz konstruktor generatorske klase. Kao što je već bilo pomenuto, generatorske klase se zapravo učitavaju kroz konfiguraciju korišćenjem obrazaca inverzija kontrole i „ubrizgavanje" zavisnosti (dependency control). Dalje u okviru metoda za definisanje koda mogu se upisivati CodeDom naredbe, ili se jednostavno može učitati kod iz spoljnih resursa. Slika 70 prikazuje niz naredbi koje služe za generisanje koda za properti koji je povezan na osnovni atribut forme. Prikazan je inače najjednostavniji način korišćenja, dok sama biblioteka pruža mnogo šire mogućnosti.

```
public FormCreator(MedicalDataContainer item, GenerationConfiguration conf)
{
    this.selectedMDC = item;
    targetUnit = new CodeCompileUnit();
    samples = new CodeNamespace(conf.DestinationNamespace);
    AddNamespaces();
    targetClass = new CodeTypeDeclaration("Form_" + DBController.ConvertName(item.Name));
    targetClass.BaseTypes.Add(typeof(Form));
    targetClass.IsClass = true;
    targetClass.TypeAttributes =
        TypeAttributes.Public;
    samples.Types.Add(targetClass);
    targetUnit.Namespaces.Add(samples);
}
```

Slika 69 Primer inicijalne deklaracije CodeDom objekata kroz konstruktor

```
CodeMemberProperty cmp = new CodeMemberProperty();
cmp.Name = "Pregled";
cmp.Attributes = MemberAttributes.Public;
cmp.Type = new CodeTypeReference(typeName);
cmp.GetStatements.Add(new CodeSnippetStatement("\t\t\t\treturn this.pregled;"));
cmp.SetStatements.Add(new CodeSnippetStatement("\t\t\t\tthis.pregled = value;"));
targetClass.Members.Add(cmp);
```

Slika 70 Primer CodeDom koda za generisanje propertija

Otvoren pristup u uključivanju novih biblioteka za generisanje koda omogućava jednostavno proširenje skupa komponenti koje je moguće generisati. Glavni posao ovde je kreiranje klasa koje će biti odgovorne za generisane koda, ali je koncept u velikoj meri opšti i može se primeniti za veći broj različitih tipova komponenata. Kada se one definišu i verifikuju, onda je njihovo korišćenje od velike pomoći. Nedostatak ovog konkretnog rešenja je u tome što koristi biblioteku koja je tesno vezana za jednu tehnologiju (.NET) i što programer mora da uloži određeni napor da razume način rada pomenute biblioteke. Prednost je što biblioteka dolazi kroz okruženje za koje postoji dokumentacija velikog obima i što nema nikakvih nestandardnih sintaksnih konstrukcija.

Dakle, nakon što se učitaju domenski model, generatorska klasa i šablonska komponenta, alat za generisanje će pozvati izvršenje odgovarajućih metoda iz generatorske klase i kod koji se dobije kao rezultat izvršenja će umetnuti na mesto generičkih komentara u šablonskoj komponenti.

Na sledećim slikama je prikazan rezultat generisanja koda za Windows forme. Slika 71 predstavlja automatski generisano zaglavlje fajla. U njemu su sadržani podaci o verziji CodeDom biblioteke kojom je fajl generisan. On se nalazi u okviru auto-generated komentarisanog taga. Ovaj potpis je moguće proširiti, tako što će se u procesu generisanja koda deo iznad ili ispod njega dopuniti odgovarajućim komentarima (Slika 72).

```
//------------------------------------------------------------------------------
// Medis.NET code generator
// Medis.Neuro.TemplateComponents.FormCreator
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:2.0.50727.3053
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------
```

Slika 71 Potpis alata za generisanje na početku generisanog fajla

```
CSharpCodeProvider provider = new CSharpCodeProvider();
var tw = new IndentedTextWriter(new StreamWriter(filename, false), "    ");

tw.WriteLine("//------------------------------------------------------------------------------");
tw.WriteLine("// Medis.NET code generator");
tw.WriteLine("// Medis.Neuro.TemplateComponents.FormCreator");

provider.GenerateCodeFromCompileUnit(compileUnit, tw, new CodeGeneratorOptions());
```

Slika 72 Primer koda za promenu potpisa metoda za generisanje

Generisanje bilo kakvih dodatnih komentara, kao što je pomenuti potpis komponente je standardna funkcionalnost kao i bilo koje drugo generisanje komponenata. Slika 74 prikazuje deo generisanog koda koji se odnosi na definisanje novih članova generisane klase – atributa i propertija. Oni su ovde podeljeni u tri glavne grupe:

- regularni atributi klase

- propertiji koji enkapsuliraju atribute
- članovi klase koji se odnose na generisane vizuelne komponente

Regularni atributi klase su svi članovi koje klasa za generisanje kreira, a koji nisu vezani za vizuelne kontrole. Njihov broj nije ograničen i mogu biti bilo kog tipa koji je vidljiv iz generisane forme. U većini generisanih formi koje slede interfejs korišćen u ovom primeru, postoji samo jedna referenca na neki spoljni objekat. Ti objekti se zovu pregled i njihov tip je određen odgovarajućim tipom koji je izabran iz objektnog modela prilikom generisanja koda. Slika 74 prikazuje primer u kome je to klasa angiografija koja dolazi iz domenskog modela podataka.

Za svaki od regularnih atributa moguće je definisati bilo jednosmerni ili dvosmerni javni properti koji će omogućiti pristup atributu van klase. U konkretnom primeru get i set metode propertija samo čitaju, odnosno postavljaju novu vrednost. Generalno, u okviru oba metoda jednog enkapsulirajućeg propertija moguće je generisati bilo kakav kod koji će vršiti dodatnu obradu. Na osnovu dosadašnjeg iskustva, uočeno je da se najčešće vrše dve vrste provere:

- kod get metode: provera da li je enkapsulirani objekat jednak NULL, i ako jeste kreiranje i vraćanje podrazumevanog objekta
- kod set metode: provera da li je prosleđena vrednost jednaka enkapsuliranom objektu, i ako nije prosleđivanje promene komponentama koje osluškuju promenu (Slika 73).

Kao što je pomenuto, u primeru sa sledeće slike, properti Pregled u metodama get i set vrši jedino prostu dodelu.

```
public string FirstName
{
    get { return firstName; }
    set
    {
        if (firstName == value)
            return;
        firstName = value;
        RaisePropertyChanged("FirstName");
    }
}
```

Slika 73 Properti sa proverom kod postavljanja vrednosti

Nakon toga sledi lista deklaracija promenljivih koje odgovaraju generisanim vizuelnim komponentama. U zavisnosti od planiranog izgleda komponente proces generisanja se može veoma razlikovati od klase do klase. Odnosno, može se reći da je generisanje vizuelnih komponentama zavisno od generatorske klase i njene implementacije.

U konkretnom slučaju, generatorska klasa iterira kroz niz atributa klase preuzete iz domenskog modela, i za svaki od njih generiše vizuelne komponente koje će prikazati jedan atribut. U konkretnom slučaju za svaki od pomenutih atributa generišu se tri komponente – jedna je labela koja čuva naziv komponente, jedna je komponenta koja omogućuje prikaz i promenu vrednosti (text box, check box, numeric text box, combo box) i još jedna labela u kojoj se prikazuje jedinica mere. Niz standardnih vrednosti se u konkretnom primeru prikazuje kao on-hover-hint. Opciono, niz standardnih vrednosti se

može dodati uz labelu za jedinice mere, ako ukupna dužina prikazanog teksta ne pređe definisani maksimalni broj karaktera.

```
34          private Medis.Controls.CtrlLekarSelektor ctrlLekarSelektor1;
35          private System.Windows.Forms.Button btnSnimi;
36          private System.Windows.Forms.Button btnNazad;
37
38          private angiografija pregled;
39          public virtual angiografija Pregled
40          {
41              get{ return this.pregled;}
42              set{ this.pregled = value;}
43          }
44          private System.Windows.Forms.Label lblpatoloski_nalaz;
45          private System.Windows.Forms.CheckBox cbxpatoloski_nalaz;
46          private System.Windows.Forms.Label lblpatoloski_nalazMU;
47          private System.Windows.Forms.Label lblarterija;
48          private System.Windows.Forms.ComboBox comxarterija;
49          private System.Windows.Forms.Label lblarterijaMU;
50          private System.Windows.Forms.Label lblstrana;
51          private System.Windows.Forms.ComboBox comxstrana;
52          private System.Windows.Forms.Label lblstranaMU;
53          private System.Windows.Forms.Label lblnalaz;
54          private System.Windows.Forms.ComboBox comxnalaz;
55          private System.Windows.Forms.Label lblnalazMU;
56          private System.Windows.Forms.Label lblopis_nalaza;
57          private System.Windows.Forms.TextBox tbxopis_nalaza;
58          private System.Windows.Forms.Label lblopis_nalazaMU;
59          private System.Windows.Forms.Label lblnapomena;
60          private System.Windows.Forms.TextBox tbxnapomena;
61          private System.Windows.Forms.Label lblnapomenaMU;
62
63          public Form_angiografija(long patientId)
64          {
65              InitializeComponent();
66              this.ctrlOsiguranikInfo1.SelectOsiguranik(patientId);
67          }
```

Slika 74 Generisani kod u delu za deklarisanje promenljivih

Nakon generisanja koda zaduženog za definisanje novih članova klase, generiše se kod zadužen za njihovo instanciranje. Instanciranje deklarisanih vizuelnih komponenti i postavljanje vrednosti vrši se u okviru metode InitializeComponent (Slika 75). Ova metoda dolazi kroz .NET okvir i standardni je deo svake vizuelne Windows forme. U njoj se definiše izgled svih vizuelnih elemenata. Ovu metodu može da izvrši u design time-u visual editor iz razvojnog okruženja i omogući podešavanje izgleda forme kroz vizuelni alat. Ovo je veoma bitno zato što se onda nad generisanom formom može dalje raditi kako bi se dodatno podesio izgled ili kako bi se kasnije programirale posebne funkcionalnosti.

Kod generisanja komponenti vezanih za jedan atribut klase za koju se generiše forma, glavni element je komponenta za prikaz i promenu vrednosti. Ona se odabira na osnovu tipa odgovarajućeg atributa klase:

- za tip bool se generiše checkbox komponenta, i uz check box ne ide labela za jedinicu mere. Checkbox ima tri stanja – označeno, neoznačeno i nedefinisano, ali kod generisanja u konkretnom primeru koriste se samo označeno i neoznačeno kako bi korelirale sa stanjima koje donosi tip podataka bool

- za tip text generiše se text box koji sadrži samo jednu liniju i ograničenje dužine na broj karaktera koji je jednak definisanoj dužini u modelu (Slika 76). Maksimalni broj karaktera se definiše na nivou modela, ali se kasnije može predefinisati.

- za tip description generiše se text box sa više linija uz uključeno ograničenje broja karaktera. Kao i kod tipa text i ovde se uvodi maksimalno ograničenje na nivou modela koje se kasnije može predefinisati za konkretne slučajeve

- za brojčane tipove generiše se numeric text box koji omogućava samo unos brojčanih vrednosti ili textbox u koji mogu da se ukucaju samo cifre i decimalni znak. U zavisnosti od vrste brojčanog tipa uključuje se ograničenje na broj cifara i na mogućnost unosa decimalnog znaka. Ograničenja na maksimalni i minimalni broj se retko definišu. Granice numeričkih vrednosti se najčešće definišu granicama samog tipa podatka.

- za izbor između više vrednosti generiše se combo box (Slika 77). Combobox sa mogućnošću pretrage prilikom kucanja je glavni izbor za komponentu koja treba da prikaže izbor iz više mogućnosti. Dodatna pogodnost je ta što combobox može da prikaže i dataset i na taj način uključi i složene objekte u listu za izbor. U primeru za generisanje koda za combobox postavljen je niz vrednosti tipa string, ali ako se kod pogleda malo bolje videće se da je memorija zauzeta za niz objekata najopštijeg tipa (new object[] ...).

```
76      private void InitializeComponent()
77      {
78          this.pnlData = new System.Windows.Forms.Panel();
79          this.lblpatoloski_nalaz = new System.Windows.Forms.Label();
80          this.cbxpatoloski_nalaz = new System.Windows.Forms.CheckBox();
81          this.lblpatoloski_nalazMU = new System.Windows.Forms.Label();
82          this.lblarterija = new System.Windows.Forms.Label();
83          this.comxarterija = new System.Windows.Forms.ComboBox();
84          this.lblarterijaMU = new System.Windows.Forms.Label();
85          this.lblstrana = new System.Windows.Forms.Label();
86          this.comxstrana = new System.Windows.Forms.ComboBox();
87          this.lblstranaMU = new System.Windows.Forms.Label();
88          this.lblnalaz = new System.Windows.Forms.Label();
89          this.comxnalaz = new System.Windows.Forms.ComboBox();
90          this.lblnalazMU = new System.Windows.Forms.Label();
91          this.lblopis_nalaza = new System.Windows.Forms.Label();
92          this.tbxopis_nalaza = new System.Windows.Forms.TextBox();
93          this.lblopis_nalazaMU = new System.Windows.Forms.Label();
94          this.lblnapomena = new System.Windows.Forms.Label();
95          this.tbxnapomena = new System.Windows.Forms.TextBox();
96          this.lblnapomenaMU = new System.Windows.Forms.Label();
97          this.ctrlOsiguranikInfo1 = new Medis.Controls.CtrlOsiguranikInfo();
98          this.ctrlLekarSelektor1 = new Medis.Controls.CtrlLekarSelektor();
99          this.btnSnimi = new System.Windows.Forms.Button();
100         this.btnNazad = new System.Windows.Forms.Button();
101         this.pnlData.SuspendLayout();
102         this.SuspendLayout();
```

Slika 75 Primer generisanog koda u delu gde se vrši instanciranje vizuelnih elemenata na formi

Teoretski moguće je koristiti bilo koje komponente za prikaz atributa koji može da uzme jednu vrednost iz predefinisanog skupa – listbox, listview, treecontrol itd. Prednost comboboxa je u tome što zauzima najmanje mesta na formi i stoga ne stvara utisak kod korisnika da se u njemu čuva neki

izuzetno bitan podatak. Prikaz komponenti sa većom površinom od ostalih na jednoj formi utiče na korisnika da poveruje da je veća kontrola zapravo mesto gde se čuva značajniji podatak.

```
326  //
327  // lblopis_nalaza
328  //
329  this.lblopis_nalaza.AutoSize = true;
330  this.lblopis_nalaza.Font = new System.Drawing.Font("Tahoma", 9F);
331  this.lblopis_nalaza.Location = new System.Drawing.Point(50, 130);
332  this.lblopis_nalaza.Name = "lblopis_nalaza";
333  this.lblopis_nalaza.Size = new System.Drawing.Size(64, 14);
334  this.lblopis_nalaza.TabIndex = 12;
335  this.lblopis_nalaza.Text = "opis nalaza";
336  //
337  // tbxopis_nalaza
338  //
339  this.tbxopis_nalaza.Font = new System.Drawing.Font("Tahoma", 9F);
340  this.tbxopis_nalaza.Location = new System.Drawing.Point(350, 130);
341  this.tbxopis_nalaza.Multiline = true;
342  this.tbxopis_nalaza.Name = "tbxopis_nalaza";
343  this.tbxopis_nalaza.Size = new System.Drawing.Size(350, 73);
344  this.tbxopis_nalaza.TabIndex = 13;
```

Slika 76 Deo generisanog koda u InitializeComponent metodi

```
261  //
262  // comxnalaz
263  //
264  this.comxnalaz.Font = new System.Drawing.Font("Tahoma", 9F);
265  this.comxnalaz.Items.AddRange(new object[] {
266  "normalan nalaz",
267  "spazam",
268  "okluzija",
269  "stenoza",
270  "ulcerozni plak",
271  "disekcija",
272  "fibromuskularna displazija",
273  "hipoplazija",
274  "aplazija",
275  "dolikoektazija",
276  "infundibularna dilatacija",
277  "neodredjena dilatacija",
278  "bazalna dislokacija",
279  "elevirana dislokacija",
280  "medijalna dislokacija",
281  "lateralna dislokacija",
282  "ipsilaterlana dislokacija",
283  "kontalateralna dislokacija",
284  "aneurizma",
285  "AVM",
286  "duralne fistule"});
287  this.comxnalaz.Location = new System.Drawing.Point(350, 105);
288  this.comxnalaz.MaxLength = 100;
289  this.comxnalaz.Name = "comxnalaz";
290  this.comxnalaz.Size = new System.Drawing.Size(350, 22);
291  this.comxnalaz.TabIndex = 10;
```

Slika 77 Primer generisanog koda u InitializeComponent metodi koji se odnosi na listu sa više izbora predstavljenu ComboBox komponentom

Slika 78 prikazuje kod za metode koje služe za učitavanje i snimanje vrednosti. U zavisnosti od odgovarajućeg tipa podataka i činjenice da li polje može prihvati NULL vrednosti, mogu se uključiti i dodatne inicijalne provere. Provere na dužinu podataka i opseg vrednosti su izmeštene u GUI, ali je moguće, za svaki slučaj uključiti ih i ovde.

Ovo se pokazalo kao posebno dobra praksa kada programeri, nakon generisanja, zamene komponentu nekom iz neke druge biblioteke i na taj način ukinu sve generisane provere. Zbog toga je dobro uključiti sve neophodne provere u metodi za snimanje podataka.

```
432         private void btnSnimi_Click(object sender, EventArgs e)
433         {
434             if (this.pregled == null)
435             {
436                 this.pregled = new angiografija();
437             }
438             this.pregled.patoloski_nalaz = cbxpatoloski_nalaz.Checked;
439             this.pregled.arterija = comxarterija.Text;
440             this.pregled.strana = comxstrana.Text;
441             this.pregled.nalaz = comxnalaz.Text;
442             this.pregled.opis_nalaza = tbxopis_nalaza.Text;
443             this.pregled.napomena = tbxnapomena.Text;
444         }
445
446         public void LoadValues()
447         {
448             this.cbxpatoloski_nalaz.Checked = this.pregled.patoloski_nalaz;
449             this.comxarterija.Text = this.pregled.arterija;
450             this.comxstrana.Text = this.pregled.strana;
451             this.comxnalaz.Text = this.pregled.nalaz;
452             this.tbxopis_nalaza.Text = this.pregled.opis_nalaza;
453             this.tbxnapomena.Text = this.pregled.napomena;
454         }
```

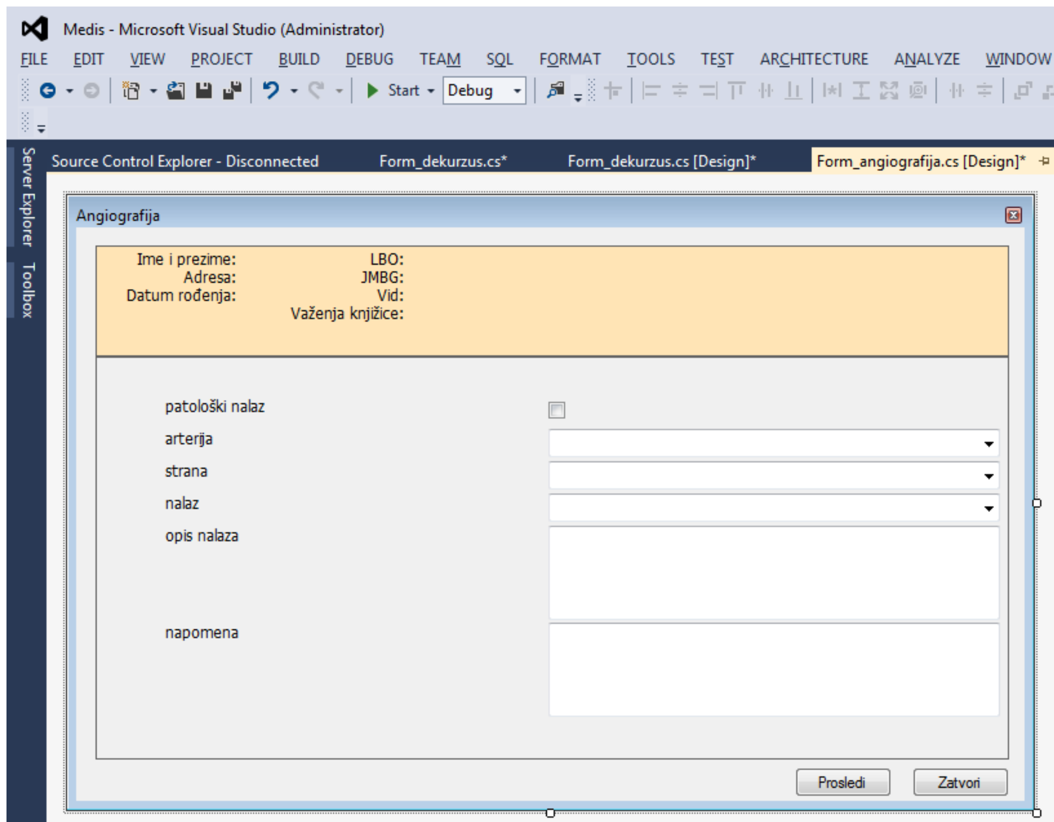
Slika 78 Generisani kod u metodama za snimanje i učitavanje

Kada se izvrše sve metode iz generatorske klase i zamene svi komentari u šablonskoj komponenti dobija se generisana forma koja može da se koristi ili za dalji razvoj ili da se uključi u odgovarajuću biblioteku. Slika 79 prikazuje konačan izgled generisane forme za objekte tipa angiografija.

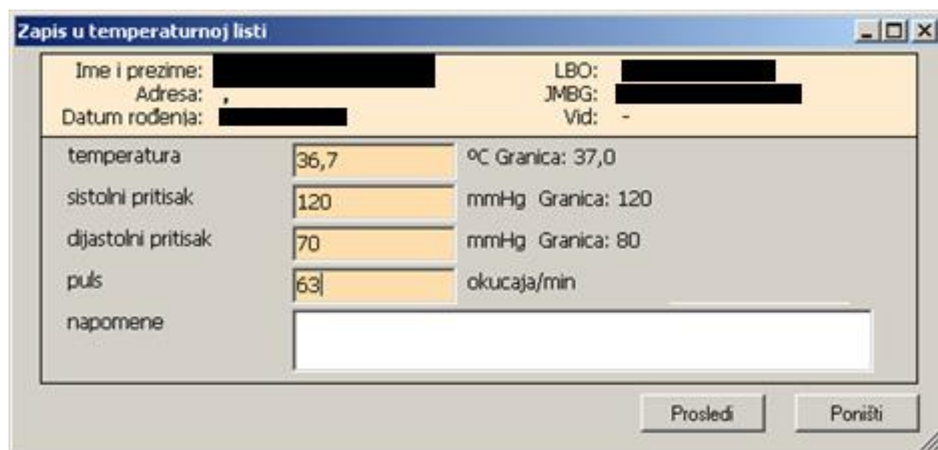
Generisana forma ima tri komponente za izbor iz skupa predefinisanih vrednosti, dva velika tekstualna polja i jedan checkbox za logički atribut koji definiše da li je nalaz pregleda patološki ili ne. U modelu ovog pregleda nisu definisane jedinice mere niti opsezi standardnih vrednosti, pošto ni jedno od polja nije odgovarajućeg tipa. Na generisanoj formi se zato vidi po jedna labela sa natpisom za svaki od atributa klase, kao i po jedna komponenta za editovanje. Uz to generisana forma ima komponentu za prikaz podataka o pacijentu kao i dugmad Prosledi i Zatvori. Ova forma je prikazana iz razvojnog okruženja, dok je sledeća generisana forma prikazana iz runtime-a.

Slika 80 prikazuje još jednu generisanu formu koja treba da prikaže zapis iz temperaturne liste. Za razliku od forme za angiografski pregled, ovde je forma sačinjena od četiri numerička atributa i jednim atributom koji služi da se unesu napomene (longtext). Ova forma je karakteristična po tome da su za prikaz numeričkih podataka odabrane obične textbox komponente u koje ne može da se unese ništa sem cifara i decimalnog znaka. Takođe, ovde su prikazane i standardne vrednosti zajedno sa labelom

koja prikazuje jedinice mere. To je bilo moguće zato što ukupna dužina teksta nije premašila prostor koji je bio raspoloživ za prikaz.



Slika 79 Izgled generisane forme

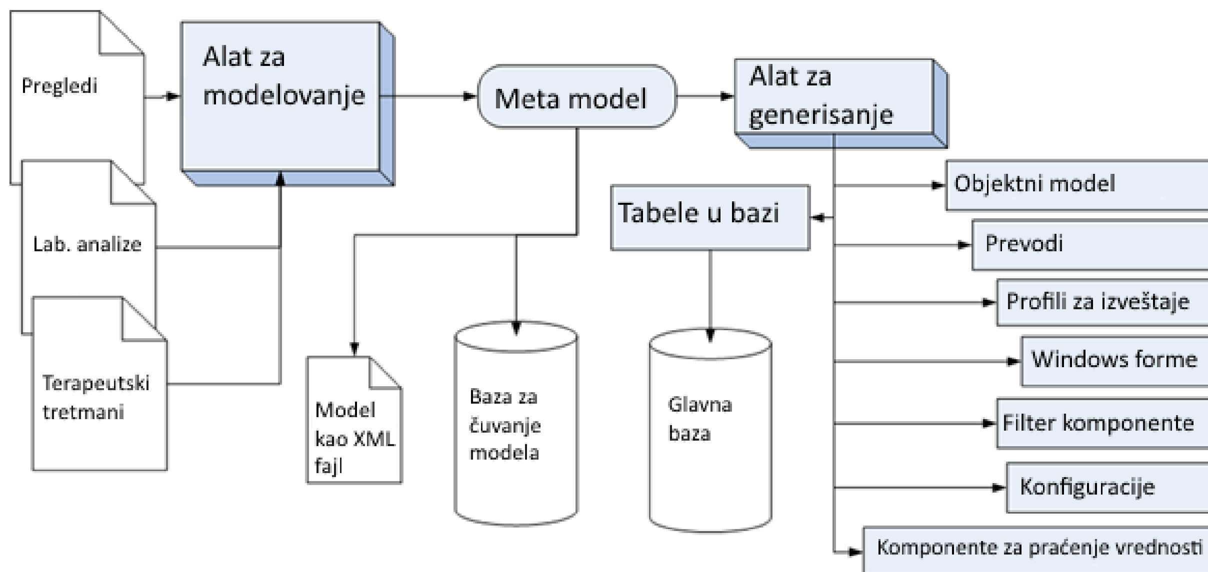


Slika 80 Primer još jedne generisane forme u runtime okruženju, sa drugačije konfigurisanom kontrolom za prikaz podataka o pacijentu i sa prikazom standardnih vrednosti u okviru posebne labele

Generisane Windows forme su samo inicijalne komponente oko kojih je razvijan sistem za generisanje koda. Nakon njih, ukazala se potreba za više različitih klasa komponenti koje bi bilo potrebno generisati. U sledećim sekcijama ovog poglavlja biće dat opis generisanja najkarakterističnijih komponenata.

4.2 Generisanje drugih vrsta komponenti

Slika 81 prikazuje celokupan proces modelovanja i generisanja koda. Ovde su prikazane klase komponenti koje su nam bile potrebne u toku realizacije i dogradnje medicinskih informacionih sistema. Sem pomenutih klasa za objektni model i windows formi, tu su još i komponente za selektovanje vrednosti, resursi za prevod, profili za izveštaje, komponenta za praćenje sačuvanih vrednosti i konfiguracioni profili. U nastavku teksta, prikazane su najznačajnije klase generisanih komponenti.



Slika 81 Proces modelovanja i generisanja uz prikazan skup najznačajnijih generisanih komponenti

4.3 Generisanje komponenti za selektovanje vrednosti

Za svaki od entiteta definisanih kroz model, kao i inače za bilo koju tabelu iz baze, moguće je definisati komponentu za pretragu. Komponenta za pretragu i odabir vrednosti je zamišljena tako da podrži pretragu vrednosti u okviru jedne tabele u bazi kroz nekoliko predefinisanih polja. Broj polja po kojima se obavlja pretraga varira od jedan do tri, u zavisnosti od toga koja vrsta entiteta se pretražuje.

Komponenta radi tako što korisnik ukucava karaktere u jedno od polja za pretragu, i nakon unetih nekoliko karaktera (broj unetih karaktera se može definisati), inicira se dopremanje podataka iz memorije i njihovo prikazivanje u okviru liste. Može se definisati kroz konfiguraciju i ukupan broj redova koji će biti prikazani.

Slika 84 prikazuje generisanu komponentu za traženje leka. Ovde je podržava pretraga po dva polja – po šifri i po nazivu. Ona je podešena da prikazuje listu izdvojenih vrednosti nakon tri uneta karaktera i da lista prikazanih lekova za izbor ne prelazi dvadeset elemenata.

Iako se mogu definisati za bilo koju tabelu, ovakve komponente su izuzetno korisne kod velikih kataloga čije su šifre često u upotrebi i koje lekari delimično znaju napamet. Klasa za generisanje ove vrste komponenti ima u sebi implementiranu drugačiju dinamiku nego što je imala klasa za generisanje formi. Ona učitava prosleđeni opšti šablon za komponente, ali sa sobom donosi i predefinisane resurse za složenije funkcije. Slika 82 prikazuje šablon za funkciju za pretraživanje kroz jednu tabelu definisanu tagom <#entitet#> po atributu definisanom sa <#polje#>. Lista filtriranih entiteta je obeležena sa <#listaFiltriranihEntiteta#> i kroz konfiguraciju se podešava njena maksimalna

dužina. Kroz konfiguraciju koja ide uz ovu klasu za generisanje specifikuju se još i minimalna broj unetih karaktera, kao i polja po kojima se može pretraživati.

```
private void FilterBy<#polje#>(string <#polje#>)
{
  <#polje#> = <#polje#>.ToLower();

  if (_lastFilterSource == FilterSource.<#polje#> &&
      <#listaFiltriranihEntiteta#> != null &&
      <#polje#>.Contains(_filterText.ToLower()))
  {
    <#listaFiltriranihEntiteta#> = <#listaFiltriranihEntiteta#>
    .Where(x => x.<#polje#>.StartsWith(<#polje#>) && x.datum_do == Common.DATUM_MAX);
    _filterText = <#polje#>;
  }
  else
  {
      <#listaFiltriranihEntiteta#> = _context.<#entitet#>
      .Where(x => x.<#polje#>.ToLower().StartsWith(<#polje#>) && x.datum_do == Common.DATUM_MAX);
      _filterText = <#polje#>;
      _lastFilterSource = FilterSource.<#polje#>;
  }
}
```

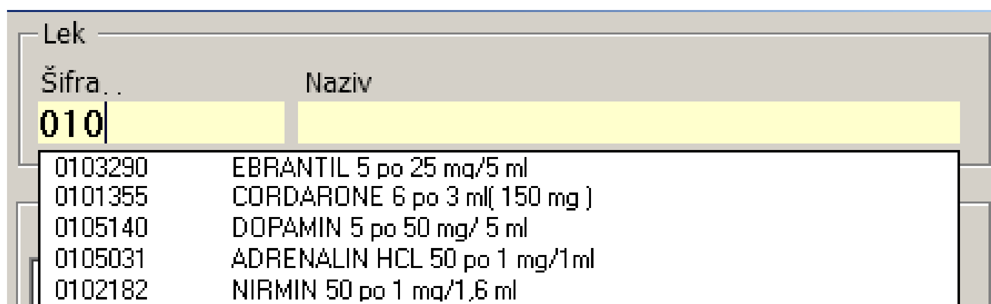
Slika 82 Šablonizovana funkcija za filtriranje entiteta po odgovarajućem polju

```
private void FilterBySifra(string sifra)
{
  sifra = sifra.ToLower();

  if (_lastFilterSource == FilterSource.Sifra &&
      _filteredLekovi != null &&
      sifra.Contains(_filterText.ToLower()))
  {
    _filteredLekovi = _filteredLekovi.Where(x => x.sifra.StartsWith(sifra) && x.datum_do == Common.DATUM_MAX);
    _filterText = sifra;
  }
  else
  {
      _filteredLekovi = _context.lek.Where(x => x.sifra.ToLower().StartsWith(sifra) && x.datum_do == Common.DATUM_MAX);
      _filterText = sifra;
      _lastFilterSource = FilterSource.Sifra;
  }
}

private void FilterByNaziv(string naziv)
{
  naziv = naziv.ToLower();

  if (_lastFilterSource == FilterSource.Naziv &&
      _filteredLekovi != null &&
      naziv.Contains(_filterText.ToLower()))
  {
      _filteredLekovi = _filteredLekovi.Where(x => x.naziv.ToLower().StartsWith(naziv) && x.datum_do == Common.DATUM_MAX);
      _filterText = naziv;
  }
  else
  {
      _filteredLekovi = _context.lek.Where(x => x.naziv.ToLower().StartsWith(naziv) && x.datum_do == Common.DATUM_MAX);
      _filterText = naziv;
      _lastFilterSource = FilterSource.Naziv;
  }
}
```

Slika 83 Generisane metode za pretragu

Slika 83 prikazuje izgled generisanih funkcija koje pribavljaju podatke iz baze. Za svako od polja se generiše zasebna funkcija, a uz njih se generišu i odgovarajući vizuelni elementi. Svako od polja za pretragu dobija u kontroli tekst boks odgovarajuće dužine.

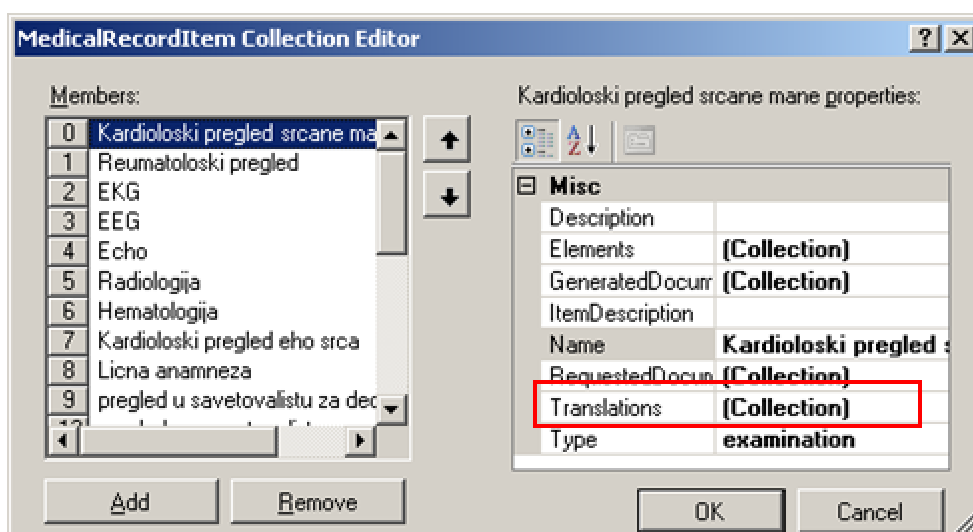
Relativni odnos veličine polja za pretragu na formi se generiše na osnovu maksimalne veličine podataka koju oni prikazuju. Dodatak u ovoj logici je definisanje minimalne dužine ovih tekst boksova pošto u slučaju kada je jedno polje mnogo veće od drugog, ne bi imalo smisla generisati vizuelnu komponentu izuzetno male širine.



Slika 84 Komponenta za selektovanje leka iz liste

4.4 Generisanje prevoda

Podrška za više jezika je jedan od, može se reći, standardnih zahteva koji se postavljaju pred informacione sisteme. Kao što je navedeno u poglavlju koje se bavi modelom za MIS sisteme, višejezička podrška je sastavni deo modela i uključena je u sve entitete koji su definisani kroz meta model, i kasnije kroz model.



Slika 85 Mesto za unos prevoda - kolekcija Translations u modelu

Predviđena su dva načina rada sa prevodima – prevod koji je ugrađen u model i prevod kroz nezavisni spoljni fajl. Prvi je unos vrednosti u kolekciju Translations (Slika 85, polje obeleženo crvenim okvirom). Klikom na ovo polje, korisnik dobija listu sa identifikatorima različitih jezika koju kasnije treba popuni sa vrednostima koje predstavljaju adekvatne prevode za naziv odabranog entiteta. Kako je ime entiteta u modelu, uvek čuvano u polju Name, uneti prevodi se odnose na to polje.

U slučaju da se odabere prevod kroz spoljni fajl, alat za modelovanje generisaće Excel dokument u kome će u okviru prve kolone uneti sve vrednosti koje su unete kroz atribut Name, svih modelovanim entiteta i atributa, i uz to će generisati po jednu kolonu za svaki od definisanih jezika. Korisnik onda treba da popuni taj Excel dokument i da ga učita u alat za generisanje, koji će generisati resurse sa

prevodima. Generisani resursi sa prevodima su obliku XML fajla, tako da je lako procesirati ih, a takođe su i razumljivi čoveku.

Generisani resursi sa prevodima se mogu koristiti dalje na dva načina. Najpre, u procesu generisanja formi, može se definisati jezik za koji se forma generiše i onda će se umesto vrednosti iz polja Name upisati vrednost odgovarajućeg prevoda. Ako nije definisan prevod, alat za generisanje će uzeti vrednost iz polja Name. Na ovaj način se dobijaju forme čiji su natpisi fiksirani i koje su prilagođene samo jednom jeziku, što nije pravi put za podršku radu sa više jezika.

Alternativa tome je da se forma generiše ne sa pravim natpisima, već vrednostima koje će biti zamenjene pravim literalima u momentu startovanja forme. Na primer, ako imamo definisan natpis za pregled čije je ime na srpskom Eho, i prevod na engleski Echo. U procesu generisanja forme, natpis će biti zamenjen generičkim stringom [#Eho]. Ako se odabere srpski, na formi će se natpis zameniti vrednošću asociranom za osnovni jezik, odnosno Eho. Ako se izabere engleski predstaviće se prevod kao Echo.

Kod pristupa generisanju formi sa fiksiranim prevodom, prednost je to što je startovanje takvih formi neznatno brže zato što tada nema potrebe da se prevode natpisi. Ako se izabere pristup sa generisanim resursima za prevod, dobija se na fleksibilnosti i jednostavnijoj podršci za više jezika.

4.5 Konfiguracije za liste privilegija

Uz veliki broj generisanih formi je potrebno definisati i listu privilegija za pristup, kao i definicije da li će neko polje biti vidljivo ili ne. Ovo nije toliko izražen zahtev kod formi sa pregledima, ali se koristi kod formi koje prikazuju različite zdravstvene kartone. Na nivou forme definišu se četiri osnovne akcije koje se mogu dozvoliti ili zabraniti korisnicima – pregled, dodavanje, ažuriranje i brisanje. Na nivou pojedinačnog polja definiše se da li je polje vidljivo, i ako jeste može se definisati da li je dozvoljeno menjanje vrednosti.

Za polja koja čuvaju kolekcije različitih objekata, definiše se konfiguracioni parametar za svaku od standardnih akcija dodavanja, brisanja i ažuriranja. Kada se izabere jedna klasa iz objektnog modela i odabere za generisanje konfiguracije, odigra se sledeći skup koraka:

- Kreira se klasa sa konfiguracionim parametrima
- Konfiguraciona klasa se poveže sa svim neophodnim, a već definisanim formama
- Definisane forme se ažuriraju da podrže konfiguraciju

Klasa sa konfiguracionim parametrima se formira u tri koraka. Najpre se generišu atributi koji se odnose na akcije na nivou celog povezanog entiteta. Svi konfiguracioni atributi su logičke vrednosti, a njihova imena se dobijaju konkatenacijom reči „dozvoli“, naziva akcije i naziva entiteta.

U primeru koji će dalje biti predstavljen, biće korišćena konfiguracija za karton za predškolsku decu. Tako, inicijalno deklarisani atributi po opisanom metodu su:

- bool DozvoliPregledKartonZaPredskolskuDecu
- bool DozvoliDodavanjeKartonZaPredskolskuDecu
- bool DozvoliPromenaKartonZaPredskolskuDecu

- bool DozvoliBrisanjeKartonZaPredskolskuDecu

```
bool dozvoliDodavanjeClanoviPorodice;
public bool DozvoliDodavanjeClanoviPorodice
{
    get { return dozvoliDodavanjeClanoviPorodice; }
    set { dozvoliDodavanjeClanoviPorodice = value; }
}

bool dozvoliPromenuClanoviPorodice;
public bool DozvoliPromenuClanoviPorodice
{
    get { return dozvoliPromenuClanoviPorodice; }
    set { dozvoliPromenuClanoviPorodice = value; }
}

bool dozvoliBrisanjeClanoviPorodice;
public bool DozvoliBrisanjeClanoviPorodice
{
    get { return dozvoliBrisanjeClanoviPorodice; }
    set { dozvoliBrisanjeClanoviPorodice = value; }
}
```

```
bool dozvoliPromenuPodaciORodjenju;
public bool DozvoliPromenuPodaciORodjenju
{
    get { return dozvoliPromenuPodaciORodjenju; }
    set { dozvoliPromenuPodaciORodjenju = value; }
}
```

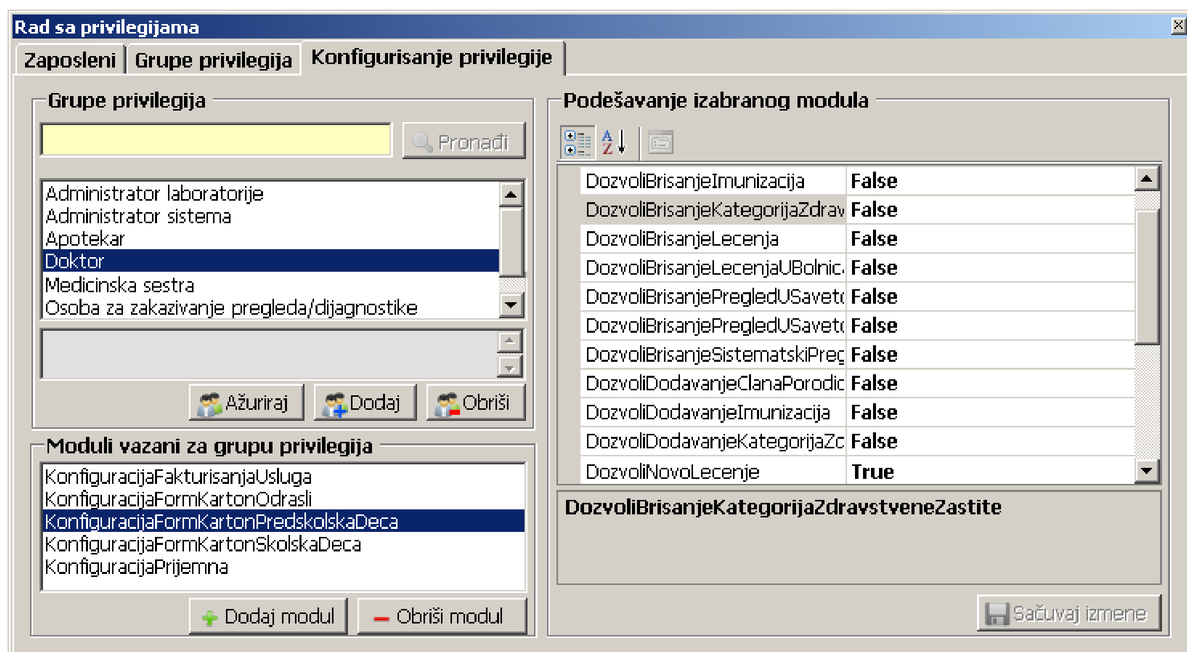
Slika 86 Primer generisanih parametara konfiguracionog profila

Dalje se generišu atributi konfiguracije za svaki element iz klase odabrane iz modela podataka. Ukoliko element modeluje niz entiteta, za njega se definišu konfiguracioni atributi za dodavanje, brisanje i promenu (Slika 86, deo koda uokviren plavom). Za proste elemente se definiše samo atribut za promenu (Slika 86, deo koda uokviren zelenom). Opciono se može uključiti dodavanje konfiguracionog parametra za prikazivanje nekog polja. U konkretnom primeru on je izostavljen, što znači da će sva polja biti vidljiva na prikazanim formama. Konfiguracioni profil, učitan kroz alat za konfigurisanje izgleda kao što prikazuje Slika 87.

Sledeći korak je povezivanje generisane konfiguracije sa formom. Ovaj korak se svodi na tri podkoraka – generisanje dodatnih logičkih atributa u odredišnu klasu, generisanje metode za primenu

konfiguracije i proširenje konstruktora parametrom koji je referenca na objekat koji nosi opis konfiguracije.

U prikazanom primeru, kod je generisan tako da se podaci sa forme ne menjaju direktno kucanjem vrednosti u kontrolu gde se prikazuju, već se klikom na dugme koje je nastavak kontrole otvara posebna forma za editovanje. Ovo je u konkretnom slučaju urađeno na ovaj način kako bi korisnici imali dodatni utisak o značaju podataka koje direktno menjaju na kartonu.



Slika 87 Lista privilegija za formu koja prikazuje karton za predškolsku decu

Dodatne logičke komponente se definišu za svaku od akcija koja nema direktno izdvojenu kontrolu na formi. U ovom slučaju, imamo formu koja koristi kontrolu sa karticama kako bi grupisala podatke (na osnovu definicije iz modela). Inicijalno nije bilo predviđeno da se izgled forme konfiguriše, tako da su sve kartice inicijalno prikazane. Uvođenjem konfiguracije za prikazivanje pojedinih grupa elemenata iz modela, koji se u formi reprezentuju jednom karticom na kontroli, u formu za prikaz zdravstvenog kartona se moraju dodati atributi koji određuju koja će kartica biti prikazana a koja ne. Ovi atributi se jednostavno kopiraju iz konfiguracije u kod forme.

```
bool prikaziTabOpstiPodaci = true;
bool prikaziTabAktivnaLecenja = true;
bool prikaziTabZnacajniMedicinskiPodaci = true;
bool prikaziTabSistematskiPregledi = true;
bool prikaziTabZavrsenaLecenja = true;
bool prikaziTabSavetovalisteZaOdojcad = true;
bool prikaziTabSavetovalisteZaMaluDecu = true;
```

Slika 88 Generisani dodatni parametri za formu koja prikazuje karton za predškolsku decu

Dalje, sledeći korak je generisanje metode za primenu konfiguracije na elemente forme (Slika 89). Klasa koja generiše kod ovde učitava model, konfiguracionu klasu i formu i povezuje komponentu za editovanje vrednosti sa odgovarajućim atributima iz konfiguracione klase. S obzirom da se i forma i konfiguraciona klasa generišu iz istog modela, sve akcije dostupne u formi mogu se konfigurisati preko konfiguracione klase.

```
public void ApplyConfiguration(KonfiguracijaFormKartonPredskolskaDeca konfiguracija)
{
    if (konfiguracija == null) return;
    this.revizija = konfiguracija.Revizija;
    prikaziTabOpstiPodaci = konfiguracija.PrikaziTabOpstiPodaci;
    prikaziTabAktivnaLecenja = konfiguracija.PrikaziTabAktivnaLecenja;
    prikaziTabZnacajniMedicinskiPodaci = konfiguracija.PrikaziTabZnacajniMedicinskiPodaci;
    prikaziTabSistematskiPregledi = konfiguracija.PrikaziTabSistematskiPregledi;
    prikaziTabZavrsenaLecenja = konfiguracija.PrikaziTabZavrsenaLecenja;
    prikaziTabSavetovalisteZaMaluDecu = konfiguracija.PrikaziTabSavetovalisteZaMaluDecu;
    prikaziTabSavetovalisteZaOdojcad = konfiguracija.PrikaziTabSavetovalisteZaOdojcad;

    #region tab opsti podaci

    #region aktivne posete

    #region tab znacajni medicinski podaci

    if (prikaziTabZnacajniMedicinskiPodaci)
    {
        btnAlergijeEdit.Visible = konfiguracija.DozvoliPromenuAlergije;
        btnVakcineEdit.Visible = konfiguracija.DozvoliPromenuVakcineSerumiIDatumi;

        btnImunizacijeAdd.Visible = konfiguracija.DozvoliDodavanjeImunizacija;
        btnImunizacijeEdit.Visible = konfiguracija.DozvoliPromenuImunizacija;
        btnImunizacijeDelete.Visible = konfiguracija.DozvoliBrisanjeImunizacija;

        btnHronicneBolestiEdit.Visible = konfiguracija.DozvoliPromenuHronicneBolesti;

        btnLecenjeUBolnicamaAdd.Visible = konfiguracija.DozvoliDodavanjeLecenjaUBolnicama;
        btnLecenjeUBolnicamaEdit.Visible = konfiguracija.DozvoliPromenuLecenjaUBolnicama;
        btnLecenjeUBolnicamaDelete.Visible = konfiguracija.DozvoliBrisanjeLecenjaUBolnicama;

        btnZnacajnaOboljenjaEdit.Visible = konfiguracija.DozvoliPromenuUtvrdjenaZnacajnaOboljenja;
        btnPoremecajiPsihFizStatEdit.Visible = konfiguracija.DozvoliPromenuPoremecajiPsihofizickogStatusa;
    }
    else
    {
        tabControl.TabPages.Remove(tpageZnacajniMedPodaci);
    }

    #endregion

    #region sistematski pregledi
}
```

Slika 89 Metoda za primenu konfiguracije

Tako na primer podaci o alergijama se mogu samo editovati, tako da je generisano preslikavanje u metodi za primenu konfiguracije:

btnAlergijeEdit.Visible = konfiguracija.DozvoliPromenuAlergije;

što znači da će dugme koje omogućuje editovanje podataka o alergijama biti vidljivo ako je atribut DozvoliPromenuAlergije iz konfiguracije postavljen na true. U slučaju kada su dozvoljene i akcije dodavanja i brisanja, kao što je slučaj sa listom imunizacija, generisani deo koda izgledaće:

btnImunizacijeAdd.Visible = konfiguracija.DozvoliDodavanjeImunizacija;

btnImunizacijeEdit.Visible = konfiguracija.DozvoliPromenuImunizacija;

btnImunizacijeDelete.Visible = konfiguracija.DozvoliBrisanjeImunizacija;

Generisanje novog konstruktora je poslednji korak u ažuriranju forme. Novi konstruktor se generiše na osnovu postojećih, tako što se doda nova promenljiva u zaglavlju konstruktora, a poziv metode za primenu konfiguracije se umetne u kod neposredno pre poziva metode PopulateForm koja je zadužena da prikaže učitani karton.

```
public FormKartonPredskolskaDeca(EntityKey kartonKey,
        EntityKey lekarKey,
        EntityKey zakKey,
        KonfiguracijaFormKartonPredskolskaDeca konfiguracija)
{
    InitializeComponent();
    this.ShowInTaskbar = false;

    try
    {
        if (!IsDesignerHosted)
            _context = new ZdravstvoEntities();
    }
    catch (Exception e)
    {
        MessageBox.Show("Problem sa bazom: " + e.Message);
    }
    InitKartonLekar(kartonKey, lekarKey, zakKey);
    ApplyConfiguration(konfiguracija);
    PopulateForm();
}
```

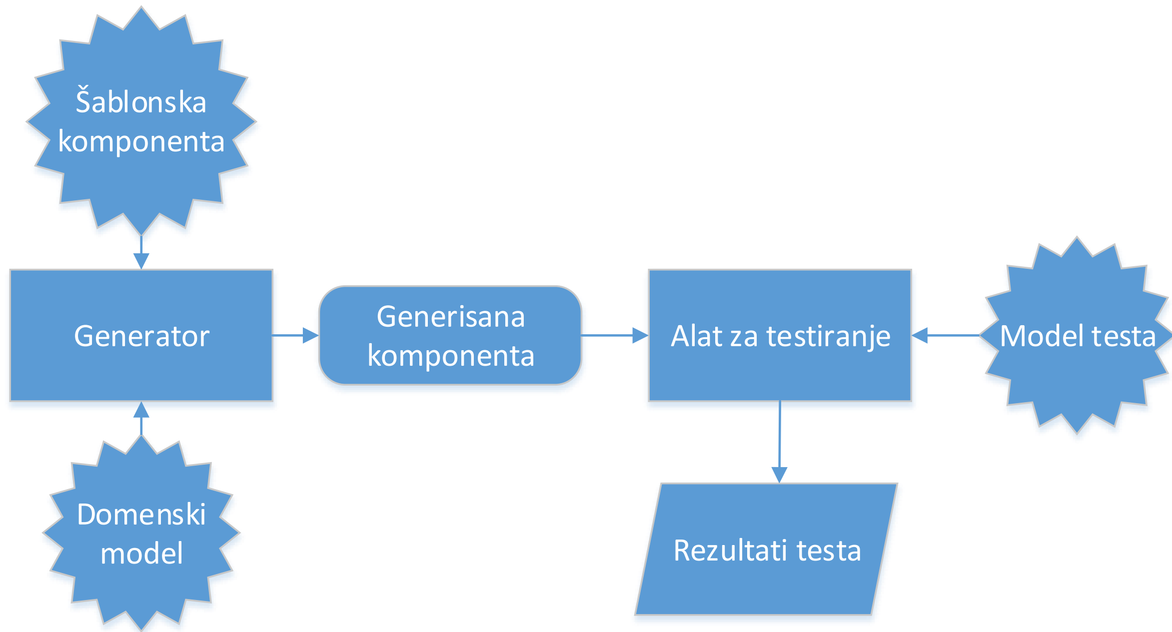
Slika 90 Proširenje konstruktora novim parametrima

4.6 Validacija generisanih komponenti

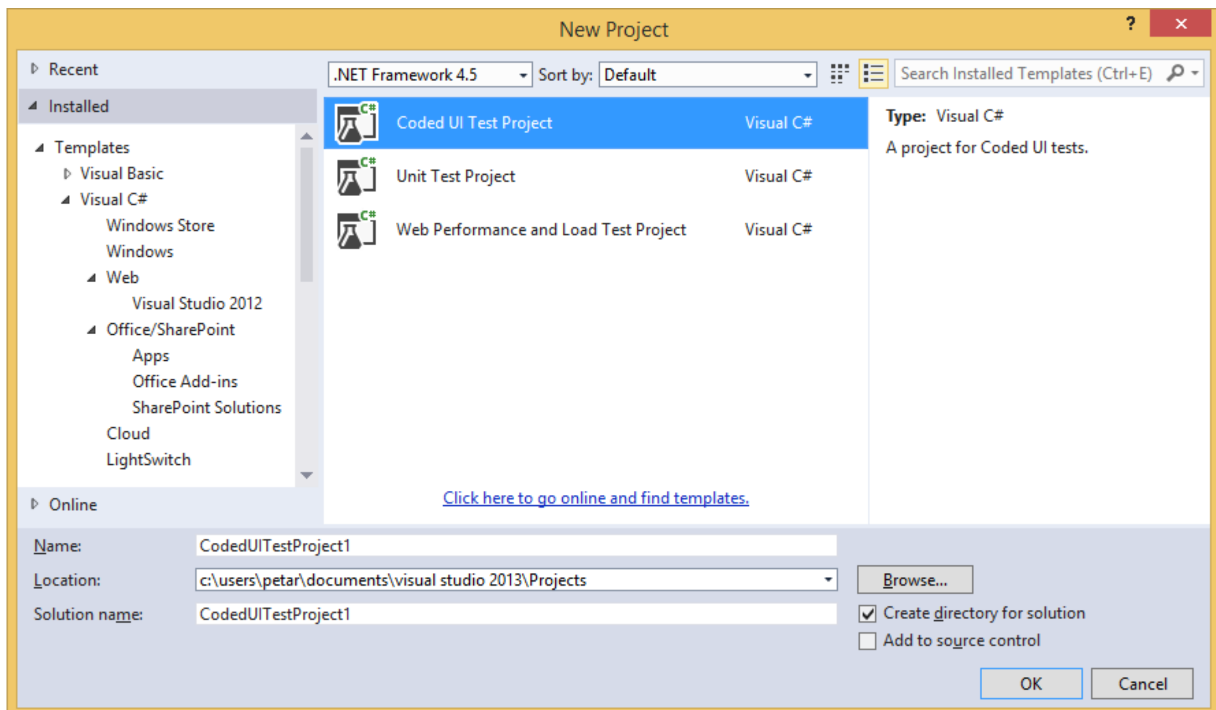
Za evaluaciju generisanih kompoenti korišćeni su standardni metodi testiranja kao što su metode crne kutije (black box testing) i metode bele kutije (white box testing). Generisane komponente se ili uključuju u projekat ili se kao dll biblioteka povezuju sa ostatkom softvera. U prvom slučaju proći će jedan nivo testiranja više pošto će programer koji ih doda u projekat izvršiti inicijalno testiranje.

Inicijalno, prvih nekoliko generisanih komponenti se detaljno testira kako bi se uočile eventualne greške u klasama koje generišu kod. Nakon nekoliko iteracija testiranja dobija se stabilna verzija klase za generisanje specifičnog koda i na osnovu nje i modela podataka može se pristupiti generisanju test vektora. Generatorska klasa može opciono generisati i delove koda koji će omogućiti logovanje vrednosti i omogućiti kasniju analizu rezulatata. Za generisanje test-vektora se takođe može kreirati nova generatorska klasa koja će u odgovarajućem šablonu menjati pojedine specifične komentare. Na taj način će alat za generisanje dobiti još jednu dodatnu komponentu.

Glavna mera efikasnosti alata za modelovanje i generisanje koda je ubrzanje procesa dogradnje i proširenja informacionih sistema. Vreme za razvoj celokupnog sistema se na ovaj način smanjuje čak za trećinu vremena. Ako se gledaju samo moduli gde se primenjuje integracija generisanih komponenti, ukupno vreme (računajući i vreme potrebno za testiranje) je manje od 40% inače potrebnog vremena. Vreme potrebno da se dobije inicijalni funkcionalni model koji se može predstaviti krajnjem korisniku je manje od 10% inicijalno potrebnog vremena (više detalja je prikazano u poglavlju koje se zove Efekat generisanih komponenti).



Slika 91 Generalni prikaz procesa testiranja komponente



Slika 92 Vrste test projekata podržanih kroz Visual Studio 2013

Tamo gde se ne može značajno dobiti na vremenu primenom ovog pristupa je finalno podešavanje izgleda generisanih komponenti na osnovu zahteva krajnjeg korisnika. U toku generisanja se može izgled forme uklopiti u odgovarajući vizuelni šablon (layout template). Takođe, polja se mogu generisati u odgovarajućem redosledu (kako god bilo implementirano u generatorskoj klasi). Ipak, finalno podešavanje se mora raditi ručno kako bi se izašlo u susret konkretnom korisničkom zahtevu.

Moguće je kreirati klase za definisanje različitih vrsta testova, ali glavna ideja je minimizovati potrebu za testiranjem i bazirati novokreirane komponente na postojećim validnim komponentama. U zavisnosti od vrste kreiranih komponenti pristupa se odgovarajućem testiranju.

Gledajući sa strane testiranja pomoću metode bele kutije, potrebno je generisati odgovarajuće testove pojedinih jedinica koda (unit tests), testove integracije (integration tests) i testove regresije (regression tests). Sve te testove izvršava alat za testiranje na osnovu generisanog skupa test vektora, odnosno modela testa (Slika 91). Izlaz iz alata za testiranje su rezulati testa koji sadrže informaciju o tome da li je i na koji način neki test prošao ili nije. Opciono, može se uključiti i dodatno logovanje iz testiranih komponenti i na taj način se kasnije mogu analizirati tokovi podataka.

```
15  /// <summary>
16  /// Summary description for CodedUITestForm_glazgov_koma_skala
17  /// </summary>
18  [CodedUITest]
    0 references
19  public class CodedUITestForm_glazgov_koma_skala
20  {
21      [TestMethod]
        0 references
22      public void ValidateLoadValues()
23      {
24          // arrange
25          Form_glazgov_koma_skala formToTest = new Form_glazgov_koma_skala();
26          glazgov_koma_skala pregled = new glazgov_koma_skala();
27          pregled.oci_otvorene = "na verbalnu komandu - 3";
28          pregled.motorni_odgovor_na_verbalnukomandu = "fleksija - povlačenje - 4";
29          pregled.verbalni_odgovor = "nerazumljivi zvuci - 2";
30          pregled.ukupno = 10;
31
32          formToTest.Pregled = pregled;
33
34          // act
35          try
36          {
37              formToTest.LoadValues();
38          }
39          catch (Exception ex)
40          {
41              // assert
42              Assert.Fail(ex.Message);
43          }
44
45          // additional assert
46          int actual = formToTest.Suma; ;
47          Assert.AreEqual(formToTest.Suma, pregled.ukupno, 0.000, "Suma nije korektno izračunata");
48
49      }
50
```

Slika 93 Primer metode za testiranje LoadValues funkcionalnosti

Za zadatke testiranja korišćen je standardni deo .NET okruženja namenjen testiranju softvera (Slika 92). U okviru jednog projekta, generišu se test klase sa odgovarajućim test metodama koje moraju biti napisane po određenom uputstvu. Svaka test metoda mora imati deo inicijalizacije podataka, deo poziva odgovarajućeg koda za izvršenje i deo gde se rezultati validiraju.

Slika 93 predstavlja primer jedne test metode. Ova metoda služi za testiranje LoadValues funkcionalnosti na formi koja prikazuje elemente takozvane Glazgov koma skale. LoadValues funkcija učitava vrednosti iz prosleđenog objekta i upisuje ih u odgovarajuće vizuelne komponente. Inicijalno postavljanje vrednosti za testiranje obavlja se u delu koda ispod komentara arrange. Tu se kreira nova instanca forme, nova instanca objekta iz objektnog modela podataka i postavljaju joj se vrednosti. U delu ispod komentara act izvršava se kod koji želimo da testiramo. Ispod komentara assert ispituju se uslovi za prijavljivanje greške u testiranoj metodi. U konkretnom slučaju, greška će se uvek prijaviti kada tog programa dođe do naredbe Asser.Fail. Pomenuta naredba se nalazi u okviru catch bloka, tako da će se aktivirati kad god se desi izuzetak u metodi LoadValues.

Dodatna provera je napisana ispod try catch strukture u delu kome prethodi komentar additional assert. Tu se poziva metoda za računanje vrednosti i proverava se da li je prosleđena vrednost jednaka izračunatoj.

```
18    [CodedUITest]
      1 reference
19    public class CodedUITestForm_glazgov_koma_skala
20    {
          2 references
21        public List<glazgov_koma_skala> TestVektori
22        {
23            get;
24            set;
25        }
26
          0 references
27        public CodedUITestForm_glazgov_koma_skala()
28        {
29            TestVektori = DeseriailizeFromFile(typeof(glazgov_koma_skala), "Form_glazgov_koma_skalaTestVektori.xml");
30        }
31
32        [TestMethod]
          0 references
33        public void ValidateLoadValues()
34        {
35            // arrange
36            Form_glazgov_koma_skala formToTest = new Form_glazgov_koma_skala();
37
38             foreach(glazgov_koma_skala vektor in TestVektori)
39             {
40                 formToTest.Pregled = vektor;
41
42                 // act
43                try
44                {
45                    formToTest.LoadValues();
46                }
47                catch (Exception ex)
48                {
49                    // assert
50                    Assert.Fail(ex.Message);
51                }
52
53                // additional assert
54                int actual = formToTest.Suma; ;
55                Assert.AreEqual(formToTest.Suma, vektor.ukupno, 0.000, "Suma nije korektno izračunata");
56             }
57
58        }
```

Slika 94 Test metoda koja podržava rad sa više test vektora

Problem sa ovakvom metodom je to što obrađuje samo jedan test vektor, i da bi se dobili smisleni rezultati testa mora joj se proslediti, na neki način, niz test vektora koji su smešteni na mestu na kome su dostupni test metodi. Takođe, sve klase za testiranje se mogu šablonizovati i iskoristiti generatorski alat da generiše sve što je potrebno za testiranje.

Za tu svrhu, ovakva test klasa zajedno sa metodom se mora malo izmeniti kako bi podržala testiranje nad više vektora odjednom. Slika 94 prikazuje primer tako izmenjen metode i pratećih atributa test klase. U odnosu na prethodni primer dodat je properti TestVektori koji čuva listu učitanih objekata koji će da budu test vektori. Ti objekti su učitani kroz metodu DeserializeFromFile kojoj se prosleđuje tip podataka koji treba izdvojiti i naziv xml fajla iz koga treba učitati objekte. Ova metoda se poziva kroz konstruktor test klase i obezbeđuje da u toku izvršenja test metode svi potrebni test vektori budu učitani.

Alat za generisanje ovde učitava model test klase koja sadrži Kroz proces generisanja koda u šablonskoj formi se menjaju komentari koji se odnose na naziv klase koja se testira (označeni zelenim okvirom na prethodnoj slici) i komentari koji se odnose na naziv forme (uokvireno narandžastim). Opciono se mogu dodati i provere specifične za svaku posebnu formu – deo uokviren plavom bojom.

```
internal static class Serializer<T>
{
    public static void Serialize(T o, string path)
    {
        using (StringWriter sww = new StringWriter())
        {
            XmlSerializer xsSubmit = new XmlSerializer(typeof(T));
            var xmlSettings = new XmlWriterSettings()
            {
                NewLineOnAttributes = true,
                NewLineHandling=NewLineHandling.Entitize,
                Indent=true,
                IndentChars="\t"
            };
            XmlWriter writer = XmlWriter.Create(sww, xmlSettings);
            xsSubmit.Serialize(writer, o);
            System.IO.File.WriteAllText(path, sww.ToString());
        }
    }

    internal static T Deserialize(string filePath)
    {
        T settings;
        using (StreamReader reader = new StreamReader(filePath))
        {
            XmlSerializer xsSubmit = new XmlSerializer(typeof(T));
            settings = xsSubmit.Deserialize(reader) as T;
        }
        return settings;
    }
}
```

Slika 95 Primer generalne klase za serijalizaciju i deserijalizaciju

Pošto je ovo metoda za testiranje forme koja u osnovi ima deljeni skup funkcionalnosti, većina koda je uvek ista. Programeri mogu kasnije, nakon generisanja ovih klasa, da dodaju još koda i tako promenjenu klasu koriste dalje za testiranje.

Sledeći bitan korak je generisanje modela testa na osnovu podataka iz modela sistema, odnosno kreiranje fajla koji sadrži test vektore. Generisanje pomenutih fajlova bazira se na mehanizmu

serijalizacije koji omogućuje da se sve klase koje su kreirane tako da imaju default konstruktor i bidirekcione propertije mogu jednostavno zapisati u XML fajl koristeći klase za serijalizaciju koje dolaze iz :NET okruženja. Srećom, svi značajni objektni modeli zahtevaju da klase koje modeluju objekte iz baze budu kreirane kao serijabilne klase i da omogućuju i snimanje u XML fajlove, ne samo u bazu. Slika 95 prikazuje primer klase koja služi za serijalizaciju i deserijalizaciju klasa iz modela podataka.

Za kreiranje test vektora mogu se definisati različiti skupovi pravila u zavisnosti od konkretne primene sistema. U definiciji testova za medicinske informacione sisteme korišćena su pravila definisana u nastavku teksta.

Logički podaci se testiraju za true, false i NULL vrednosti. Dok se za brojčane podatke definiše sledeći skup inicijalnih pravila:

- Test vrednost 0 (NUM1)
- Pozitivan slučajan broj (NUM2)
- Negativan slučajan broj (NUM3)
- Decimalan pozitivan broj sa više decimala nego što je u opsegu tipa (NUM4)
- Decimalan negativan broj sa više decimala nego što je u opsegu tipa (NUM5)

Ukoliko kod brojčanog tipa postoji definisan opseg vrednosti, uzeće se i sledeći slučajevi:

- jedan broj iz opsega validnih vrednost (RNG1)
- broj sa donje granice (RNG2)
- broj sa gornje granice (RNG3)
- broj do 10% manji od dodje granice (RNG4)
- broj za red veličine manji od donje granice (RNG5)
- broj za 10% veći od gornje granice (RNG6)
- broj za red veličine veći od gornji granice (RNG7)

Oni služe da se ispitaju specifične funkcionalnosti (ako ih ima) vezane za obradu podataka van standardnog opsega. Takođe, NULL i DEFAULT NOT NULL vrednosti se testiraju za svaki tip podataka.

Kod tekstualnih podataka, početni skup sadrži sledeća pravila za testiranje:

- prazan string (STR1)
- string sa puno blanko znakova (STR2)
- string sastavljen od jedne reči (STR3)
- string sastavljen od više reči (STR4)
- string sa specijalnim znacima (STR5)

- ćirilični tekst (STR6)
- hebrejski tekst (STR7)

Svi slučajevi se testiraju kada string ima manje ili jednako karaktera u odnosu na definisanu maksimalnu dužinu. Takođe, svi slučajevi se testiraju i kada je test string duži od maksimalne dozvoljene dužine podatka u bazi. Ukoliko postoji definisan niz dozvoljenih vrednosti, testira se za svaku od njih kao i za po jedan podatak iz svake od osnovnih grupa za testiranje.

Pošto je definisanje generatorskih klasa na programeru, predstavljeni skup pravila za testiranje nije niti konačan niti jedini mogući, već samo jedan koji se prikazao prikladnim za testiranje pojedinih jedinica koda, generisanih windows formi konkretno.

Klasa u prethodnom primeru ima ukupno 4 značajna polja za testiranje – tri od njih su stringovi sa predefinisanim opsezima vrednosti a jedna je ceo broj. U inicijalnom skupu test vektora, stringovski atributi bi se testirali sa po 14 vrednosti (NULL, prazan string i ostalih 6 pravila sa po dva slučaja – kada je dužina stringa u okviru dozvoljene dužine i kada je prosleđen duži string). Celobrojni atribut nema definisan opseg, tako da će se testirati sa ukupno 6 različitih slučajeva – NULL i 5 inicijalnih pravila za testiranje brojeva. Ovome treba dodati i testiranje sa najvećim i najmanjim brojem iz opsega vrednosti, što ukupan broj test slučajeva dovodi do broja 8.

Maksimalan broj generisanih vektora za testiranje bi onda bio 21952, što je 14x14x14x8. U zavisnosti od broja polja ovaj broj bi lako mogao da „eksplodira“ i u slučaju da se generiše forma koja ima npr 23 stringa i 16 brojeva, kao što je npr. forma za pregled oka, potpun skup testova bi imao više od 10^{19} vektora. Za forme koje podržavaju složene bolničke preglede, kao na primer pregled posle moždanih udara, koji sadrže više od 150 polja bespredmetno je i računati ukupan broj kombinacija.

Zbog toga se potpun skup testova generiše samo za forme koje se inicijalno testiraju i koje se izaberu kao reprezentativne. Reprezentativne su one forme koje imaju relativno mali broj polja, gde su polja drugačijih tipova. Ovakve forme se inače koriste i za validaciju i testiranje generatorskih klasa.

Zbog potencijalno velikog broja testova, za testiranje formi se uvode uprošćenja u generisanju skupa vektora koja služe da se inicijalno izbaci veći broj testova za koje se sa velikom dozom sigurnosti može tvrditi da će dati očekivani rezultat.

Prvi nivo smanjenja broja testova je redukovanje test pravila za odgovarajući tip podataka. Na ovom nivou primenjuje izolacija takozvanih type guard uslova – odnosno onih koji će se sa tačke gledišta korišćene tehnologije svesti na isto.

Tako na primer u slučaju stringova NULL zajedno sa pravilima STR1 i STR2 komprimuju u jedno. Ovo je moguće ako se aplikacija razvija u .NET-u zato što je u samom tipu string uvedena evaluacija koja se zove IsNullOrWhiteSpace. Ovo je provera da li je string NULL, prazan ili se sastoji od niza blanko znakova. Označimo ovo pravilo sa CBSTR1, što bi bilo skraćeno od combined string rule 1.

Dalje pravila STR3 i STR4 ostaju nepromenjiva, dok se pravila STR5, STR6 i STR7 mogu spojiti u jedno pravilo CBSTR2 koje će prosleđivati string koji sadrži različite unicode karaktere. Na ovaj način se broj pravila smanjuje na 4, i svako se testira dva puta – jednom za validnu dužinu stringa i jednom za nevalidnu. Na ovaj način se ukupan broj testova svede na 8.

Kod testiranja celih brojeva, izbacuju se pravila NUM4 i NUM5 zato što će sigurno biti odbačena zbog konverzije tipova. Takođe, pravilo NUM2 kao i testiranje sa najmanjim brojem iz opsega se

može odbaciti zato što su celobrojni atributi pregleda najčešće prirodni brojevi koji pokazuju koliko nečega ima u nekom uzorku. Na ovaj način se ukupan broj test slučajeva za brojeve smanjuje sa 8 na 4. Pa je u prethodno navedenim primerima broj testova smanjen na 2048 sa 21592 za formu sa 3 stringa i jednim celobrojnim atributom. Kod forme sa 23 stringovska i 16 celobrojnih polja ukupan broj testova se smanji sa $4*10^{27}$ na $5*10^{20}$ što je takođe irelevantno i bespredmetno. Ovaj prvi nivo smanjenja se koristi za finalno testiranje generatorskih klasa i na formama koje imaju najviše do 10 polja.

Drugi nivo smanjenja predstavlja generisanje testova kod kojih se za promenljivu jednog tipa generiše maksimalan broj testova, a sve ostale promenljive istog tipa se testiraju za po jednu ispravnu i jednu neispravnu vrednost. Na ovaj način se za prvu formu sa tri stringa i jednim celim brojem dobija ukupno 448 testova (14x2x2 za stringove što množi 8 za celobrojnu promenljivu). Kod forme sa 23 stringa i 16 celih brojeva imamo $1.5*10^{13}$ testova ($14*2^{22}*8*2^{15}$). Ovo je takođe generisanje testova u domenu testiranja inicijalno generisanih komponenti i koristi se forme do petnaestak komponenti.

Sva ova smanjenja broja test vektora se koriste u fazi razvoja i gotovo su neprimenjiva za forme koje imaju više od dvadesetak različitih elemenata. Zato se pribegava generisanju takozvanih ukrojenih (tailored) test vektora. Cilj je da se na određenom fiksnom broju test vektora izaberu reprezentativne vrednosti tako da svi slučajevi budu pokriveni u globalu, dok ukupan broj testove ne pređe neku, uslovno govoreći, razumnu meru od manje od 100. Slika 96 prikazuje jedan primer niza serijalizovanih testova.


```xml
<TestVektori>
    <glazgov_koma_skala>
        <oci_otvorene>NULL</oci_otvorene>
        <motorni_odgovor_na_verbalnukomandu>NULL</motorni_odgovor_na_verbalnukomandu>
        <verbalni_odgovor>NULL</verbalni_odgovor>
        <ukupno>NULL</ukupno>
    </glazgov_koma_skala>
    <glazgov_koma_skala>
        <oci_otvorene></oci_otvorene>
        <motorni_odgovor_na_verbalnukomandu></motorni_odgovor_na_verbalnukomandu>
        <verbalni_odgovor></verbalni_odgovor>
        <ukupno>0</ukupno>
    </glazgov_koma_skala>
    <glazgov_koma_skala>
        <oci_otvorene>Lorem</oci_otvorene>
        <motorni_odgovor_na_verbalnukomandu>Ipsum</motorni_odgovor_na_verbalnukomandu>
        <verbalni_odgovor>Neque</verbalni_odgovor>
        <ukupno>11</ukupno>
    </glazgov_koma_skala>
    <glazgov_koma_skala>
        <oci_otvorene>bez odgovora - 1</oci_otvorene>
        <motorni_odgovor_na_verbalnukomandu>ekstenzija (decerebraciona rigidnost) - 2</motorni_odgovor_na_verbalnukomandu>
        <verbalni_odgovor>
        Lorem ipsum је био стандард за модел текста још од 1500. године,
        када је непознати штампар узео кутију са словима и сложио их како би направио узорак књиге
        </verbalni_odgovor>
        <ukupno>-1731</ukupno>
    </glazgov_koma_skala>
    <glazgov_koma_skala>
        <oci_otvorene>spontano - 4</oci_otvorene>
        <motorni_odgovor_na_verbalnukomandu>ekstenzija (decerebraciona rigidnost) - 2</motorni_odgovor_na_verbalnukomandu>
        <verbalni_odgovor>dezorijentisan i razgovara - 4</verbalni_odgovor>
        <ukupno>10</ukupno>
    </glazgov_koma_skala>
</TestVektori>
```


Slika 96 Primer niza serijalizovanih test vektora

Tako na primer, definiše se jedan test vektor gde su sve vrednosti NULL. Jedan gde su sve vrednosti default vrednost za odgovarajući tip podataka i jedan gde su sva polja sa graničnim vrednostima. Dalje, dva vektora gde su sva polja ili enormno velike ili enormno male vrednosti. Sledeće je nekoliko vektora sa ispravnim vrednostima. I na kraju izvestan broj vektora gde se za svako polje nasumice bira

vrsta test vrednosti i generiše slučajna vrednost. Da bi testovi dobili na realističnosti, određenim vrstama test vrednosti se mogu dodeliti različite važnosti tako da se one značajnije češće generišu.

Na kraju, za konačnu validaciju komponenti koriste skupovi koji imaju par desetina vrednosti i koji obezbeđuju, uz sve prethodno testiranje u toku razvoja da generisana komponenta bude dobro istestirana.

Pošto se ovde prvenstveno radi o komponentama koje dele isti osnovni skup funkcionalnosti, glavni akcenat je na testiranju samih komponenti. Drugi nivo testiranja – integraciono testiranje se u ovakvom slučaju može značajno uprostiti i svesti na unapred definisan skup testova kod kojih će se menjati samo entitet koji se odnosi na generisanu formu i klasu koja modeluje njen prateći objekat.

```
public void TestAddNewMedicalRecordItem()
{
        string formToOpenTypeName = "Form_" + typeToCreate.Name;
        Type formToOpen = Type.GetType("Medis.Neuro." + formToOpenTypeName);
        object form = Activator.CreateInstance(formToOpen); Form createdForm = form as Form;
        IDocumentProperties dokument = form as IDocumentProperties;
        dokument.SetPatient(_pacijentId);

        foreach (glazgov_koma_skala vektor in TestVektori)
        {
            try
            {
                _context.data_usluga.MergeOption = MergeOption.NoTracking;
                data_usluga du = new data_usluga();
                _context.AddTodata_usluga(du); dokument.Pregled = vektor;

                PropertyInfo propPregled = du.GetType().GetProperty(vektor.GetType().Name);
                object referenceToPregledCollection = propPregled.GetValue(du, null);
                MethodInfo mi = referenceToPregledCollection.GetType().GetMethod("Add");

                mi.Invoke(referenceToPregledCollection, new object[] {
                                                vektor.GetType().GetMethod("GetConvertedObject",
                                                BindingFlags.Public | BindingFlags.Static)
                                                .Invoke(null, new object[] {vektor})
                                            });

                du.data_usluga_dijagnoza = null; du.poseta = prijemnaPoseta;
                du.usluga = _context.usluga.Where(x => x.usluga_id == -1).First();
                du.ImenikLekarKojiJePregledao = du.ImenikUneo = du.ImenikZadnjiMenjao = OrdinirajuciLekar;

                if (zakazivanje != null) zakazivanje.osiguranikReference.LoadIfNeeded();
                else zakazivanje = _context.zakazivanje.Where(x => x.zakazivanje_id == -1).First();
                IstorijaBolesti.karton.imenikReference.LoadIfNeeded();
                karton kt = _context.karton.Where(x => x.karton_id == IstorijaBolesti.karton.karton_id).First();

                du.karton = kt; du.zakazivanje = zakazivanje;
                osiguranik pacijent = _context.osiguranik.Where(x => x.osiguranik_id == PacijentId).First();
                _context.Attach(pacijent);  du.osiguranik = pacijent;
                du.orgj = _context.orgj.Where(x=>x.orgj_id == LogovaniKorisnik.Instance.Orgj.orgj_id).First();

                du.datum_vreme = Common.GetServerDate();
                _context.SaveChanges();
            }
            catch (Exception ex)
            {
                Assert.Fail(ex.Message);
            }
        }
    }
}
```

Slika 97 Primer metode za integraciono testiranje sa označenim generisanim parametrom

Slika 97 prikazuje primer generisane metode za integraciono testiranje. To je metoda koja treba da testira dodavanje novog entiteta tipa glazgov_koma_skala i njegovo povezivanje sa istorijom bolesti. Ovde je maksimalno iskorišćeno to što generisane klase implementiraju odgovarajuće interfejse pa se

u kodu koriste maksimalno metode iz osnovnih klasa i interfejsa. Tamo gde to nije bilo moguće, ili nije postojalo adekvatno stablo nasleđivanja iskorišćena je refleksija tipova da se uzmu i izvrše odgovarajuće metode. Kao i kod testiranja pojedinih jedinica koda i ovde se koriste iste kolekcije test vektora.

U krajnjoj instanci, treći nivo testiranja, regresivno testiranje, ne zahteva kreiranje specijalnih dodatnih testova pošto je cela ideja proširenja na osnovu modela zapravo dogradnja sistema oko tačaka proširenja, pa se stoga celokupno testiranje fokusira na samu tačku proširenja i njenu okolinu.

4.7 Razvoj aplikacije pomoću generisanih komponenata

Proces proširenja aplikacije biće predstavljen na primeru dela informacionog sistema koji se koristi za podršku procesima u bolnici. Kao što je bilo navedeno u opisu arhitekture sistema, MIS je kreiran kao servisno orijentisani sistem u čijem središtu je WCF servis. Takođe, proširenje sistema na nivou modula ostvareno je inverzijom kontrola kroz spring biblioteku [106].

Slika 98 prikazuje konfiguraciju sistema koja se koristi u primarnom zdravstvu – u ambulantama. Kao što se vidi, ovde je definisan objekat koji predstavlja komunikacioni servis kao i objekat koji referencira osnovni modul (MedisRuntimeModul). Sem toga, konfigurisano je nekoliko administrativnih modula (Kadrovska, izbor lekara, rad sa zdravstvenim knjižicama i modul za definisanje radnog vremena) kao i moduli za vođenje opštih kartona i specijalističke preglede. Sve to je podržano i laboratorijskim delom kroz module za centralnu i mobilnu laboratoriju.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- modules declarations -->
<objects xmlns="http://www.springframework.net">
    <object type="Spring.Objects.Factory.Attributes.RequiredAttributeObjectPostProcessor, Spring.Core"/>
        <!-- Komunikacioni servis -->
    <object type="Medis.Communication.Service, Medis.Communication" singleton="true" id="MedisCommunicationService"/>
        <!-- Osnovni modul-->
    <object type="Medis.Core.Runtime.CoreRuntimeMsgListener, Medis.Core.Runtime" singleton="true" id="MedisRuntimeModule"/>
        <!-- Administracija -->
    <object type="Medis.Common.Kadrovska, Medis.Common" singleton="true" id="Kadrovska"/>
    <object type="Medis.Common.IzborLekara, Medis.Common" singleton="true" id="IzborLekara"/>
    <object type="Medis.Common.ZdravstvenaKnjizica, Medis.Common" singleton="true" id="ZdravstvenaKnjizica"/>
    <object type="Medis.Common.RadnoVreme, Medis.Common" singleton="true" id="RadnoVreme"/>
        <!-- Ambulanta-->
    <object type="Medis.Primar.Ambulanta, Medis.Primar" singleton="true" id="AmbulantaPrimar"/>
    <object type="Medis.Primar.Specijalisticki, Medis.Primar" singleton="true" id="AmbulantaSpecijalisticki"/>
    <object type="Medis.Dental.Modul, Medis.Dental" singleton="true" id="Dental"/>
        <!-- Laboratorija-->
    <object type="Medis.Laboratorija.Modul, Medis.Laboratorija" singleton="true" id="CentralnaLaboratorija"/>
    <object type="Medis.Laboratorija.Mobilna, Medis.Laboratorija" singleton="true" id="MobilnaLaboratorija"/>
</objects>
```

Slika 98 Primer inicijalne konfiguracije aplikacije za primarno zdravstvo

Dodavanje novih modula vrši se tako što se u konfiguracioni fajl doda opis odgovarajućeg modula tako što se navede naziv klase koja omogućuje komunikaciju sa servisom i naziv biblioteke u kojoj je ta klasa implementirana. Komunikaciona klasa mora da ispoštuje odgovarajući interfejs kako bi bila vidljiva servisu nakon učitavanja.

Menjanje ove konfiguracije, generalno, zahteva promenu konfiguracionog fajla i restartovanje sistema. Vreme potrebno za restartovanje sistema zavisi od broja konfigurisanih modula, kao i od složenosti koda koji će se izvršiti prilikom startovanja svakog od modula.

Na ovaj način se jednostavno mogu uključiti novi, i isključiti moduli koji više nisu neophodni. Slika 99 prikazuje primer gde je sistem proširen dodavanjem modula koji su neophodni za funkcionisanje neurološke klinike. Dodate su klase Stacionar i IstorijaBolesti koje nose funkcionalnosti vezane za osnovno funkcionisanje bolnice. Ove dve fasadne klase su, u osnovi, dovoljne da podrže minimum zahteva za funkcionisanje stacionarnih ustanova – prijem pacijenta, temperaturna lista, utrošak lekova, otpust itd.

Klasa Neuro donosi reference ka pregledima čiji su objekti za pristup podacima i forme kreirani kroz model i kroz alat za generisanje. Ovde je moguće konfigurisati proizvoljan broj takvih fasadnih pregleda, a one će kroz svoju inicijalizaciju definisati sve potrebne tipove.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- modules declarations -->
<objects xmlns="http://www.springframework.net">
    <object type="Spring.Objects.Factory.Attributes.RequiredAttributeObjectPostProcessor, Spring.Core"/>
        <!-- Komunikacioni servis -->
    <object type="Medis.Communication.Service, Medis.Communication" singleton="true" id="MedisCommunicationService"/>
        <!-- Osnovni modul-->
    <object type="Medis.Core.Runtime.CoreRuntimeMsgListener, Medis.Core.Runtime" singleton="true" id="MedisRuntimeModule"/>
        <!-- Administracija -->
    <object type="Medis.Common.Kadrovska, Medis.Common" singleton="true" id="Kadrovska"/>
    <object type="Medis.Common.IzborLekara, Medis.Common" singleton="true" id="IzborLekara"/>
    <object type="Medis.Common.ZdravstvenaKnjizica, Medis.Common" singleton="true" id="ZdravstvenaKnjizica"/>
    <object type="Medis.Common.RadnoVreme, Medis.Common" singleton="true" id="RadnoVreme"/>
        <!-- Ambulanta-->
    <object type="Medis.Primar.Ambulanta, Medis.Primar" singleton="true" id="AmbulantaPrimar"/>
    <object type="Medis.Primar.Specijalisticki, Medis.Primar" singleton="true" id="AmbulantaSpecijalisticki"/>
    <object type="Medis.Dental.Modul, Medis.Dental" singleton="true" id="Dental"/>
        <!-- Laboratorija-->
    <object type="Medis.Laboratorija.Modul, Medis.Laboratorija" singleton="true" id="CentralnaLaboratorija"/>
    <object type="Medis.Laboratorija.Mobilna, Medis.Laboratorija" singleton="true" id="MobilnaLaboratorija"/>
        <!-- Stacionar-->
    <object type="Medis.Common.Stacionar, Medis.Common" singleton="true" id="Stacionar"/>
    <object type="Medis.Bolnica.IstorijaBolesti, Medis.Bolnica" singleton="true" id="IstorijaBolesti"/>
    <object type="Medis.Neuro.Pregledi, Medis.Neuro" singleton="true" id="Neuro"/>
</objects>
```

Slika 99 Dodavanje konfiguracije za podršku pregledima na neurološkoj klinici

Lista podržanih pregleda se može konfigurisati na strani servisa i proslediti klijentima, ili se može predefinisati lokalno na svakom klijentu. Slika 100 pokazuje jednu takvu konfiguraciju dok Slika 101 prikazuje primer forme koja služi za administriranje pregleda – dodavanje novih, ažuriranje i brisanje. Lista pregleda sadrži naziv i parametre koji omogućuju da se uključe ili isključe prijem pacijenata, temperaturna lista i otpust pacijenata kao predefinisane funkcionalnosti.

Svi ostali pregledi koji su definisani i generisani kroz model definišu se kroz grupe pregleda. Svaka grupa ima svoj naziv i listu elemenata prikazanih kroz puno kvalifikovano ime generisanih formi. Na formi koja služi za administraciju pregleda (Slika 101) svaka grupa će biti predstavljena kroz posebnu karticu. U okviru jedne kartice, svaki od pregleda je prikazan kroz posebnu kolonu. Svaki pojedinačni pregled prikazan je onim što vraća njegova predefinisana ToString metoda. U konkretnom slučaju, to je za sve preglede vreme i datum kada su registrovani.

Desni klik na zaglavlje tabele sa pregledima otvara kontekstni meni u kome su opcije za dodavanje novog pregleda kao i za otvaranje komponente koja služi da poredi vrednosti iz svih registrovanih pregleda jedne grupe. Ukoliko imaju generisanu grafičku komponentu ona će biti otvorena, u suprotnom otvoriće se osnovna komponenta za tabelarni prikaz vrednosti. Temperaturna lista je jedan od primera takve komponente. Ona je, kao što je pomenuto u prethodnom poglavlju, upotrebljena da bude osnova za izvođenje šablonske komponente za grafički prikaz vrednosti.

Aplikacija za klinike ima fiksiran centralni deo, dok je deo sa pregledima i dokumentima fleksibilan i njegovo ažuriranje se bazira na učitavanju novih biblioteka sa formama koje su generisane na osnovu domenskog modela. Svaka od pomenutih formi ima u osnovi skup osnovnih funkcionalnosti koje obezbeđuju vezu na odgovarajuće roditeljske objekte i realizuju osnovne funkcionalnosti za svaki od generisanih entiteta – dodavanje, učitavanje, ažuriranje i brisanje.

Centralni deo aplikacije obezbeđuje kreiranje i učitavanje objekata i njihovo prosleđivanje formi. Kroz formu se ostvaruju neophodne izmene i promenjeni objekat se vraća centralnom delu aplikacije koji obezbeđuje interakciju sa bazom i snimanje ili brisanje specifičnog objekta.

```
- <ListaPregleda>
    <Naziv>Neuro</Naziv>
    <PrijemPacijenta>true</PrijemPacijenta>
    <TemperaturnaLista>true</TemperaturnaLista>
  - <Grupe>
    - <Grupa>
        <Naziv>Pregledi</Naziv>
      - <Elementi>
          <Element>Medis.Neuro.Pregledi.neurologija_glavne_tegobe</Element>
          <Element>Medis.Neuro.Pregledi.neurologija_licna_anamneza</Element>
          <Element>Medis.Neuro.Pregledi.neurologija_porodicna_anamneza</Element>
          <Element>Medis.Neuro.Pregledi.neurologija_psihicki_status</Element>
          <Element>Medis.Neuro.Pregledi.neurologija_sadasnja_bolest</Element>
          <Element>Medis.Neuro.Pregledi.neurologija_status_neurologicus</Element>
          <Element>Medis.Neuro.Pregledi.neurologija_status_praesens</Element>
          <Element>Medis.Neuro.Pregledi.dekurzus</Element>
        </Elementi>
      </Grupa>
    - <Grupa>
        <Naziv>CTM</Naziv>
      - <Elementi>
          <Element>Medis.Neuro.Pregledi.ctm_haemorrhagio_intracerebrallis</Element>
          <Element>Medis.Neuro.Pregledi.ctm_haemorrhagio_subarachnoidalis</Element>
          <Element>Medis.Neuro.Pregledi.ctm_infarctus_cerebri</Element>
          <Element>Medis.Neuro.Pregledi.ctm_reduktivne_promene</Element>
        </Elementi>
      </Grupa>
    - <Grupa>
        <Naziv>Lab.analize</Naziv>
      - <Elementi>
          <Element>Medis.Neuro.Pregledi.biohemija</Element>
          <Element>Medis.Neuro.Pregledi.hemostatski_status</Element>
          <Element>Medis.Neuro.Pregledi.acidobazni_status</Element>
          <Element>Medis.Neuro.Pregledi.likvor</Element>
        </Elementi>
      </Grupa>
    - <Grupa>
        <Naziv>Skale za procenu bolestika</Naziv>
      - <Elementi>
          <Element>Medis.Neuro.Pregledi.bartel_indeks</Element>
          <Element>Medis.Neuro.Pregledi.glazgov_koma_skala</Element>
        </Elementi>
      </Grupa>
    </Grupe>
    <OtpustPacijenta>true</OtpustPacijenta>
  </ListaPregleda>
```

Slika 100 Konfiguracija za listu pregleda

S obzirom da je dodavanje novih modula u aplikaciju izuzetno jednostavno, ljudima koji rade na razvoju može veoma lako ispasti iz vida činjenica da svaki novi modul doprinosi povećanju obima obrade podataka, kao i ukupnom zauzeću ostalih resursa računara. Zbog toga je veoma važno izvršiti analizu performansi sistema pre i nakon dodavanja nekog modula kako bi se proverilo u kom procentu sam modul utiče na ceo sistem.

Za potrebe te analize korišćeni su alati za analizu koda koji standardno dolaze uz Visual Studio. Slika 99 prikazuje način na koji je u aplikaciju dodat moduo za podršku neurološkim pregledima. Nakon toga neophodno je, uz standardno testiranje, izvršiti i analizu zauzeća resursa. U navedenoj konfiguraciji moduo za neurologiju, iako samo jedan, trebao bi da utiče u znatnoj meri na povećanje zahteva sistema za resursima. On donosi podmoduo koji će potpuno da predefiniše početni deo aplikacije i uz to doneće 146 novih pregleda i medicinskih dokumenata. Na ovaj način gotove će duplirati broj podržanih dokumenata – sa 160 na 306.

ĆIRIĆ MILAN ALEKSANDRA (4)

Ime i prezime:
Adresa:
Datum rođenja: 05.05.
LBO:
JMBG:
Vid: -

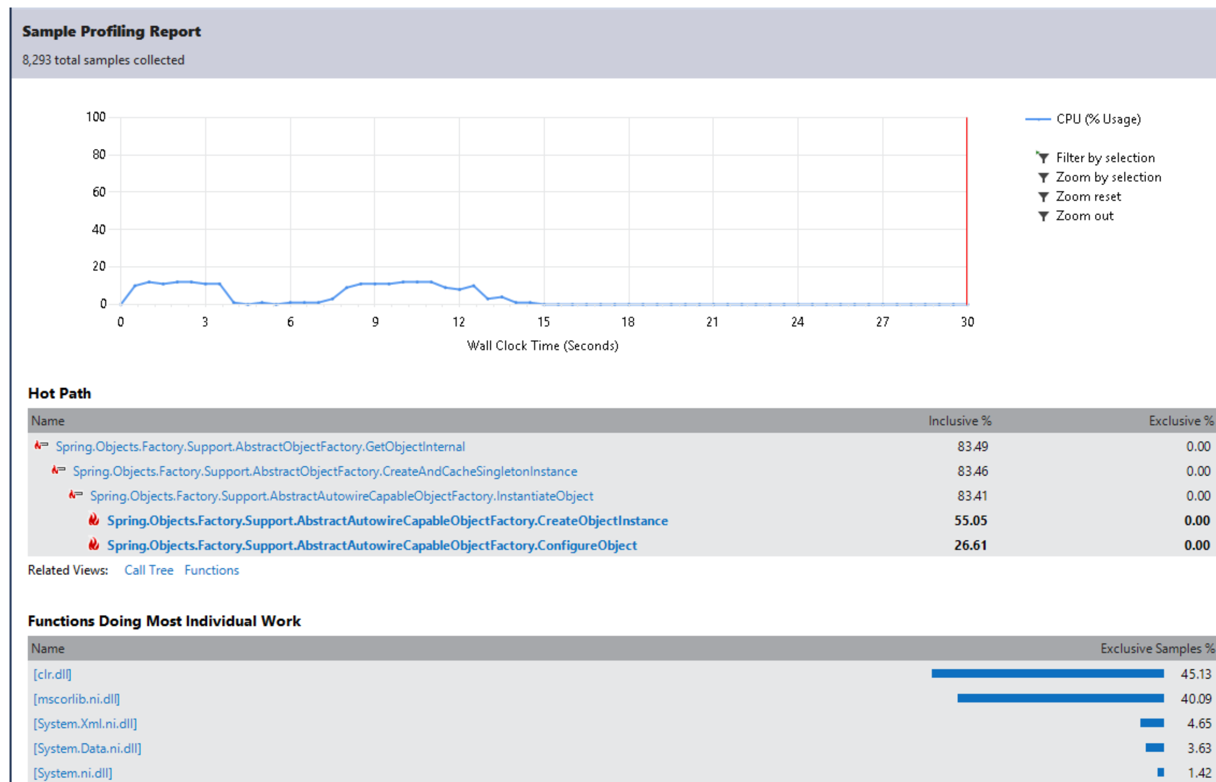
Prijem | Temperaturna lista | Pregledi | CTM | Lab. analize | Skale za procenu bolesnika | Otpust

glavne tegobe	licna anamneza	porodicna anamneza	psihicki status	sadasnja bolest	ne
9.2.2010 12:04:44	11.2.2010 16:29:22	11.2.2010 16:29:15	11.2.2010 16:29:18		
	9.2.2010 15:08:00				

Lekar koji je otvorio istoriju bolesti

Prosledi Poništi

Slika 101 Primer forme koja tabelarno prikazuje podržane preglede. Pregledi su podeljeni u grupe, gde svaka grupa ima svoju karticu



Name	Inclusive %	Exclusive %
Spring.Objects.Factory.Support.AbstractObjectFactory.GetObjectInternal	83.49	0.00
Spring.Objects.Factory.Support.AbstractObjectFactory.CreateAndCacheSingletonInstance	83.46	0.00
Spring.Objects.Factory.Support.AbstractAutowireCapableObjectFactory.InstantiateObject	83.41	0.00
Spring.Objects.Factory.Support.AbstractAutowireCapableObjectFactory.CreateObjectInstance	**55.05**	**0.00**
Spring.Objects.Factory.Support.AbstractAutowireCapableObjectFactory.ConfigureObject	**26.61**	**0.00**

Name	Exclusive Samples %
[clr.dll]	45.13
[mscorlib.ni.dll]	40.09
[System.Xml.ni.dll]	4.65
[System.Data.ni.dll]	3.63
[System.ni.dll]	1.42

Slika 102 Uzorci sa procesora pre dodavanja modula "neuro"

Slika 102 i Slika 103 prikazuju aktivnost procesora tokom izvršenja. Za ovaj primer analiziran je samo proces startovanja servisa i njihova inicijalizacija. Ovde se može uočiti da dodavanje novog modula ne doprinosi u značajnoj meri maksimalnom zauzeću procesora. Ni u jednom slučaju, maksimalno zauzeće procesora ne stiže ni do 20%. Međutim, sa novim modulom ceo proces startovanja traje duže. Period kada zauzeće dođe na nivo blizu nule je manje od 15 sekundi bez modula Neuro, a skoro 17,5 sa njim.

Ako se pogleda na šta je vreme najviše potrošeno, to je kreiranje objekata i njihovo konfigurisanje. U prvom slučaju te vrednosti su 55,05% i 26,61% od 30 sekundi koliko je test trajao, a u drugom slučaju (nakon dodavanja modula) 53,85% i 25,35% od 40 sekundi. S, obzirom da su procenti slični, a vreme trajanja testa je u drugom slučaju za jednu trećinu duže, upravo toliko iznosi i prosečno povećanje korišćenja procesora.

Ako se pogleda koje funkcije individualno troše najviše vremena, to su funkcije iz osnove .NET okvira koje su odgovorne za pozivanje konstruktora, kreiranje i konfigurisanje objekata (clr i mscorlib). Nakon toga su funkcije za rad XML-om, koje se aktiviraju kroz čitanje raznih konfiguracionih fajlova. Sledeće na listi su funkcije iz biblioteke System.Data koje su zadužene za pristupanje bazama podataka.

Vrednosti koje se dobiju za System.Data, u principu mogu da se prilično razlikuju od jednog do drugog test okruženja i zavise od toga koliko je konekcija na bazu dobra. Što se tiče svih ostalih parametara, njihov međusobni odnos je manje više isti, posebno ako konfiguracioni fajlovi stoje na istom računaru kao i izvršni program.

Slika 103 Uzorci sa procesora nakon dodavanja modula "neuro"

Slika 104 i Slika 105 pokazuju aktivne niti i vreme njihovog izvršenja. Ovde se vidi da je novi moduo doneo ukupno pet novih aktivnih niti nakon starta sistema. Ako se pogleda sadržaj tabela na

navedenim slikama, vidi se da se pri startu inicira u oba slučaja 7 različitih niti (to su one čiji je parametar begin time jednak nuli) koje pokreću osnovne delove sistema.

Unique ID	ID	Name	Begin Time	End Time	Life Time
0	6704	Unknown	0.00	30,662.37	30,662.37
1	1904	Unknown	0.00	30,662.37	30,662.37
2	5876	Unknown	0.00	30,662.37	30,662.37
3	5372	Unknown	0.00	28,861.86	28,861.86
4	2240	Unknown	0.00	578.01	578.01
5	5200	Unknown	0.00	28,885.79	28,885.78
6	7284	Unknown	0.00	30,662.37	30,662.36
7	7208	Unknown	0.00	30,662.37	30,662.36
8	3832	Unknown	898.77	30,662.37	29,763.60
9	7996	Unknown	1,293.17	30,662.37	29,369.20
10	3500	Unknown	1,299.26	30,662.37	29,363.11
11	1196	Unknown	1,299.52	30,662.37	29,362.85
12	1200	Unknown	1,299.63	30,662.37	29,362.73
13	8144	Unknown	1,299.90	30,662.37	29,362.47
14	7892	Unknown	12,435.16	30,662.37	18,227.21
15	5904	Unknown	12,435.19	30,662.37	18,227.18
16	6956	Unknown	12,437.23	30,662.37	18,225.14
17	2384	Unknown	13,153.46	30,662.37	17,508.91
18	2216	Unknown	13,861.50	28,865.80	15,004.30
19	6904	Unknown	13,861.51	28,862.83	15,001.32
20	5660	Unknown	14,175.53	14,376.85	201.32
21	7744	Unknown	23,224.28	24,185.38	961.10
22	5968	Unknown	24,240.39	24,245.26	4.87

Slika 104 Procesi (niti) aktivni pre dodavanja novog modula

Unique ID	ID	Name	Begin Time	End Time	Life Time
0	224	Unknown	0.00	41,442.25	41,442.25
1	8284	Unknown	0.00	41,442.25	41,442.25
2	8280	Unknown	0.00	41,442.25	41,442.25
3	5264	Unknown	0.00	32,403.41	32,403.41
4	2500	Unknown	0.00	1,358.79	1,358.79
5	516	Unknown	0.00	14,859.27	14,859.27
6	992	Unknown	0.00	14,859.24	14,859.24
7	1420	Unknown	0.00	41,442.25	41,442.25
8	9024	Unknown	2,120.28	41,442.25	39,321.97
9	9112	Unknown	2,126.46	41,442.25	39,315.79
10	9012	Unknown	2,126.62	41,442.25	39,315.63
11	9028	Unknown	2,126.75	41,442.25	39,315.50
12	8984	Unknown	2,127.06	41,442.25	39,315.19
13	9080	Unknown	2,816.17	37,436.09	34,619.92
14	7300	Unknown	17,290.98	41,442.25	24,151.27
15	9008	Unknown	17,291.00	41,442.25	24,151.25
16	8980	Unknown	17,388.42	32,392.31	15,003.90
17	8636	Unknown	17,388.47	32,388.95	15,000.48
18	4464	Unknown	17,388.59	32,388.92	15,000.32
19	8588	Unknown	17,388.62	32,388.90	15,000.28
20	7344	Unknown	17,388.65	32,388.85	15,000.20
21	8976	Unknown	17,388.74	32,392.36	15,003.62
22	7184	Unknown	17,388.76	41,442.25	24,053.49
23	8608	Unknown	17,940.16	41,442.25	23,502.09
24	4992	Unknown	17,940.26	41,442.25	23,501.99
25	3144	Unknown	26,009.31	26,052.62	43.31
26	8632	Unknown	28,022.52	28,040.44	17.92
27	6920	Unknown	36,102.86	36,108.52	5.66

Slika 105 Niti aktivne u programu nakon dodavanja novog modula

Nakon što se oni startuju inicira se povezivanje sa bazama i drugim spoljnim servisima, za koje su zadužene niti sa rednim brojevima 8 do 13. Nakon toga kreće startovanje modula i pre dodavanja modula za neurološke preglede bilo je startovano 6 niti. Nakon dodavanja novog modula startovano je ukupno 11. Na kraju, poslednje 3 niti su sinhronizacioni procesi sa spoljnim servisima koji pristupaju servisima fonda za zdravstvenu zaštitu i proveravaju da li treba ažurirati neki od kataloga. S obzirom, na ukupno povećanje obima sistema, povećanje zauzeća resursa je u granicama očekivanog.

4.8 Efekat generisanih komponenti

Generisane komponente doprinose ukupnom smanjenju vremena potrebnom za razvoj aplikacije, ali egzaktan procenat nije moguće tačno utvrditi, tako da će kvantitativna analiza efekata biti samo približno tačna.

Najpre treba identifikovati gde se sve, u standardnom procesu razvoja komponenti primenjuju generisane komponente. Tako, na primer mogu da se izdvoje sledeći koraci:

1. kreiranje tabela u bazi
2. kreiranje klase u objektnom modelu
3. kreiranje vizuelne komponente (najčešće forme) koja sadrži polja koja treba povezati sa poljima u bazi
4. implementiranje metoda za validaciju vrednosti polja na forma
5. implementiranje specifične logike za konkretnu formu
6. implementiranje konfiguracije i liste privilegija
7. implementiranje komponente za pretragu po zapisima
8. inicijalno testiranje

Primenom modela mnogi od ovih koraka se značajno ubrzavaju tako da na primer kreiranje modela i tabela u bazi se mogu stopiti u jedan korak. Na osnovu ličnog, i iskustva članova tima koji su radili na razvoju medicinskih informacionih sistema približno možemo da definišemo kreiranje tabele u bazi kao osnovnu operaciju i da definišemo njeno trajanje sa T. Svi ostali procesi su normirani na ovo vreme i prikazani u tabeli 2.

Otprilike isto toliko vremena je potrebno i da se kreira klasa koja ima ista polja kao i baza. Kada se koristi objektni model iz nekog standardnog paketa (kao što je NHibernate) onda se može smatrati da se koraci 1 i 2 mogu spojiti u jedan. Naime, svi standardni paketi za objektne modele nude mogućnost da se na osnovu kreirane klase automatski generiše tabela. Isti slučaj je i kada se radi sa modelom podataka. Na osnovu modela entiteta biće generisani i klasa u objektnom modelu i tabela u bazi. Vreme potrebno da se definiše jedan entitet u modelu je takođe približno jednak vremenu da se kreira tabela.

Kreiranje same forme bez pomoći alata za automatsko generisanje traje znatno duže nego kreiranje tabele. Naime, potrebno je dodati sve vizuelne elemente forme i povezati svako polje u formi sa poljem u bazi. S obzirom da za svako polje treba kreirati makar natpis i jednu komponentu za ažuriranje vrednosti, pozicionirati nove komponente u okviru forme i povezati sa bazom, ceo proces traje makar 3 puta duže, odnosno 3T. U slučaju automatskog generisanja korišćenjem standardnih form wizard komponenti, neophodno je odabrati tabelu i osnovne opcije i pokrenuti proces

generisanja. Nakon toga generisana forma će biti dodata u aplikaciju. U slučaju korišćenja procesa baziranog na modelu, forma se automatski generiše kad i sve ostale komponente tako da je samo dovoljno uključiti generisane fajlove u projekat i eventualno promeniti namespace. Vreme koje se utroši na ovom koraku je mnogo manje nego kada se forma manuelno pravi. Subjektivno je procenjeno na 5 do 10% vremena da se kreira tabela u bazi.

Sledeći element koji se analizira je kreiranje metoda za validaciju ulaza. Form wizard alati daju najčešće validaciju po tipu, ali se svaka specifična validacija i dalje mora implementirati. Kod pristupa baziranog na modelu, generisana klasa već sadrži sve potrebne metode, i eventualno je potrebna neka mala adaptacija za poneko polje. Subjektivna procena je da je za ovo potrebno jednako vremena kao i za osnovno kreiranje forme kada se radi bez automatizacije. Sa form wizardom, posao se smanjuje na otprilike trećinu. Primenom modela i svih alata za generisanje, ovde gotovo da nije potrebna naknadna intervencija.

Tabela 2 Broj izveštaja u toku godine

Korak	Bez ikakve optimizacije procesa	Korišćenjem standardnih komponenti	Primenom razvoja baziranom na modelu
kreiranje tabela u bazi	T	0	0
kreiranje klase u objektnom modelu	T	T	T
kreiranje forme	3T	0.1T	0.05T
implementiranje metoda za validaciju	3T	T	0.1T
implementiranje specifične logike za konkretnu formu	2T	2T	2T
implementiranje konfiguracije	2T	2T	0.1T
implementiranje komponente za pretragu po zapisima	2T	0.5T	0.1T
Inicijalno testiranje	6T	3T	0.5T
Ukupno	22T	9.6T	3.9T

Implementacija specifične logike za konkretnu formu je deo koda koji se ne može nikako bitno ubrzati, pošto je to skup procedura koje su tipične za svaki konkretan slučaj. Subjektivna procena je da je to vreme, u proseku, otprilike dva puta duže nego vreme potrebno da se definiše tabela.

Implementacija konfiguracije i liste privilegija je poseban proces koji traje makar dva puta duže nego kreiranje same tabele, i kod njega form wizard ne može da pomogne. U pristupu sa alatima za generisanje ove komponente se kreiraju kad i forma, tako da je kasnije neophodna neka mala izmena. Sa druge strane, form wizardi kreiraju najčešće i komponentu za pretragu, ali kao sastavni elemenat forme, tako da se mora izdvajati sa forme ako je potrebna kao posebna komponenta koja će da se uključi u druge projekte.

Kod korišćenja bilo kog pristupa sa automatskim generisanjem koda, vreme za testiranje i inicijalno ispravljanje grešaka je mnogo kraće. Ukupno vreme potrebno za razvoj jedne komponente se otprilike smanjuje na petinu inicijalne potrebe kada se ne koristi nikakva automatizacija. U poređenju sa form wizard alatima vreme razvoja je redukovano na 40% inicijalno potrebnog.

Ako se pogleda efekat na ukupno ubrzanje procesa razvoja, može se primetiti da alati za automatsko generisanje daju efekta tamo gde postoji mogućnost za njihovu primenu. Kada treba razvijati delove sistema koji su jedinstveni, onda celokupno ubrzanje zavisi od konkretnog projekta. Na osnovu nekog našeg iskustva, poređeno sa vremenom potrebnim kada se ne koristi automatizacija, razvoj celokupnog sistema može da se svede na jednu do dve trećine ukupno potrebnog vremena, što je prilično širok interval, ali je ova stavka ipak direktno zavisna od projekta koji se razvija.

Međutim, glavni dobitak je kada treba kreirati demo aplikacije. Onda se ceo proces svodi samo na inicijalno testiranje generisane demo aplikacije što je u principu najviše 10% vremena potrebnog za njen razvoj.

5 Interpretirane komponente

Uz generisane komponente, druga kategorija koja omogućava brži i efikasniji razvoj su interpretirane, odnosno komponente koje se generišu dinamički, u toku izvršenja programa. Kod informacionih sistema, one su veoma često prvi izbor kada je u pitanju deo sistema koji je odgovoran za izveštaje. U ovom odeljku biće predstavljeno Web rešenje za generisane komponente, prvenstveno namenjene kreiranju izveštaja.

Informacioni sistemi sakupljaju, prikazuju i analiziraju podatke koji dolaze iz različitih izvora. Za svaku specifičnu primenu potrebno je da se prilagode elementi korisničkog interfejsa: izveštaji, šabloni i logičke grupe vizuelnih komponenata – vidžeti. Vidžeti su glavna osnova za izgradnju korisničkog interfejsa i oblikuju se tako da budu jednostavni za korišćenje u alatima za editovanje. U isto vreme, treba da budu dovoljno opšti za kreiranje složenog GUI – ja. Ovo se postiže njihovom dvojnom prezentacijom. Vidžeti u okviru izveštaja poseduju model – view – controller (MVC) arhitekturu i prave se kombinovanjem konfigurabilnih šablona i unetih podataka. U okviru alata za modelovanje i generisanje koda, svaki vidžet se zamenjuje sa jednim HTML elementom obogaćenim standardnim atributima podataka. Korišćenjem HTML5 jezika omogućeno je izvršavanje koda i na klijentu i na serveru. Prednosti predloženog rešenje su visoki nivo konfiguracije i mogućnost dinamičkog kreiranja korisničkog interfejs.

U opštem slučaju, MIS ili PIMS može biti veliki distribuirani sistem koji sakuplja podatke iz više raznorodnih izvora podataka. Ti izvori mogu biti relacione baze podataka, realtime baze podataka, XML fajlovi, i strimovi podataka sakupljeni sa uređaja. Primer takvog standarda za PIMS sisteme može se videti u [68].

Glavna ideja koja stoji iza konkretnog pristupa je da se proširiva platforma zasnovana na modelu može dodatno specijalizovati za konkretnu oblast primene. Proširenje oblasti primene podrazumeva proširenje oblasti modela kod poslovne logike [69]. Nakon ovoga obično sledi i proširenje korisničkog interfejsa: novi šabloni izveštaji i novi UI elementi. Posledica toga je da korisnički interfejs MIS-a treba da ima visok nivo plastičnosti i u idealnom slučaju da bude generisan u toku izvršenja.

Postoji konstantna potreba da se poboljšava korisnički interfejs u MIS-u, i u delu sakupljanja podataka i u delu izveštaja [70]. Tabela 3 prikazuje broj pruženih medicinskih usluga (pregleda, analiza, tretmana) kao i izveštaja koji se koriste u domovima zdravlja. Neki izveštaji se generišu za ministarstvo zdravlja, neki za osiguranje, dok se jedan broj izveštaja smatra internim i koristi ih uprava ustanove. Još jedna kategorija značajnih izveštaja su stručni statistički izveštaji. Istovremeno, istraživači, sa medicinskog fakulteta, koji imaju pristup bazi podataka, zahtevaju drugačije naučne izveštaje. Pošto se veliki broj ovih izveštaja često kreira, menja i briše njihov broj nije precizan, već je procena autora.

Sledeća tabela prikazuje podatke vezane za Dom zdravlja Niš. Dom zdravlja Niš je uveo informacioni sistem najpre u službu opšte prakse 2011-te godine. U 2012-toj sistem se proširio na pedijatriju i odeljenje stomatologije. Od januara 2013, sistem se koristi na svih 45 odeljenja sakupljajući podatke za više od 13000 medicinskih pregleda dnevno. Sa porastom broja korisnika raste i broj internih i naučnih izveštaja.

Sa proširenjem sistema, došlo je i do povećanja zahteva za novim formama za sakupljanje podataka, zbog toga što je sistem stalno proširivan. Za svaku novu medicinsku uslugu dodatu u model, korisnicima će biti na raspolaganju osnovna forma. Tabela 4 prikazuje povećanje broja formi za kolekciju podataka u korelaciji sa ukupnim brojem različitih medicinskih usluga koje pruža Dom

zdravlja u Nišu i broj medicinskih usluga koje su definisane kao osnovne od strane Ministarstva zdravlja.

Podaci prikazani u sledeće dve tabele (Tabela 3 i Tabela 4) idu u prilog realizaciji sistema za dinamičko generisanje korisničkog interfejsa kako bi se izašlo u susret potrebama krajnjih korisnika. U ovom odeljku fokus će biti na rešavanju problema dinamički kreiranog korisničkog interfejsa za deo sistema koji je zadužen za izveštaje u MIS-u. Predloženo rešenje je zasnovano na iskustvu sa prethodnom verzijom MIS-a, kao i informacionim sistemima korišćenim u farmaceutskoj i industriji hrane.

Tabela 3 Broj izveštaja u toku godine

Godina	Medicinske usluge (dnevno)	Izveštaji za ministarstvo zdravlja	Izveštakji za osiguranje	Interni izveštaji	Naučni izveštaji
2011	2990	14	3	5	2
2012	7322	17	10	19	21
2013	13599	23	14	28	45
2014	13696	29	14	36	103

Tabela 4 Broj formi za sakupljanje podataka

Godina	Forme za unos podataka	Broj podržanih tipova pregleda	Broj obaveznih tipova pregleda
2011	14	210	170
2012	63	206	194
2013	74	255	231
2014	91	302	245

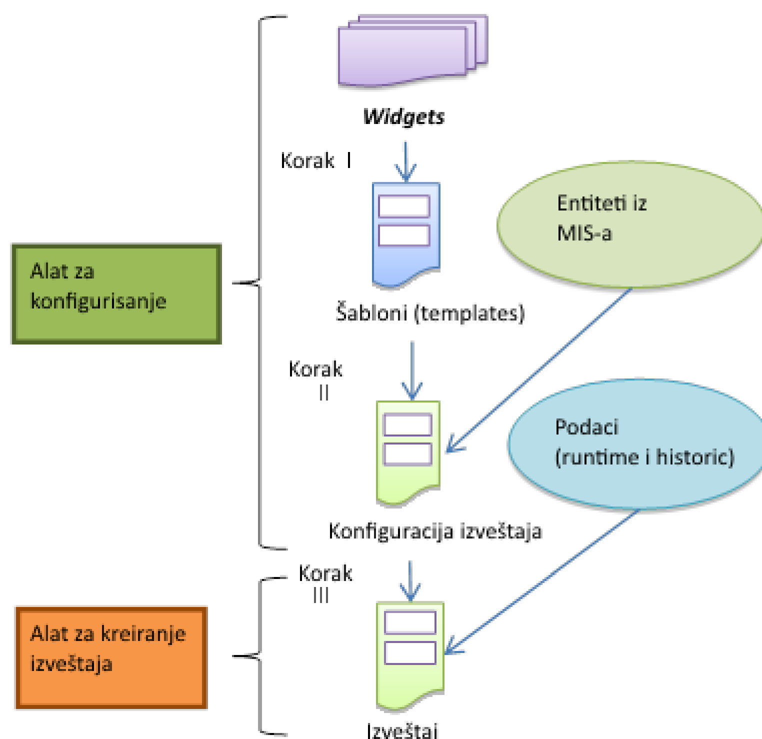
Predloženo rešenje se sastoji od dva različita softverska alata u svom segmentu za izveštaje – konfiguracioni alat i alat za generisanje izveštaja. Njegov korisnički interfejs karakterišu kompleksni UI elementi kao što su mreže podataka, filteri i grafikoni. Oni predstavljaju osnovne blokove za izgradnju šablona i izveštaja, i nazivaju se, kao što je već pomenuto, posebnim imenom vidžeti. Šabloni su HTML stranice kreirane kombinovanjem vidžeta. Konačno. izveštaji se zasnivaju na šablonima, sa vidžetima povezanim na različite izvore podataka. Tabelarni izveštaji se uglavnom koriste za prikazivanje različitih poslovnih procesa ili opisnih statičkih podataka, i mogu biti uključeni u kompleksnije izveštaje.

Kako bi postigli zadovoljavajući nivo prilagođenosti i konfigurabilnosti, sistem treba da podržava šablone i vidžete koji se mogu menjati u toku izvršenja. Vidžeti mogu biti prilično kompleksni,

sastavljeni od potencijalno velikog broja HTML elemenata, ali istovremeno i jednostavni za menjanje u editoru – korisnici bi morali da mogu da selektuju, menjaju veličinu, pomeraju i konfigurišu ih kao celine. To uključuje i sprečavanje korisnika da selektuje pojedinačni HTML element u okviru vidžeta, kako mu ne bi narušio strukturu i osnovne funkcionalnosti.

Web deo prethodno razvijanih verzija MIS sistema bio je dizajniran tako da vidžete koristi kao serverske stranice smeštene u okviru IFRAME elemenata. Do trenutka kada je stari sistem razvijan, većina današnjih Web tehnologija nije postojala. Jedna od tih tehnologija je Ajax. Jedini način da se asinhrono učita stranicu bio je korišćenjem IFRAME elemenata. Ovo je bio razlog njihovog korišćenja kao okvira za vidžete. Ovi elementi se danas smatraju lošim sa stanovište funkcionisanja sistema.

Svaki IFRAME poseduje svoj prozor i celokupnu DOM hijerarhiju, kao i skup dodatnih fajlova sa kodom koji su mu neophodni za izvršavanje. To su fajlovi koji sadrže opis stilova, spoljne resurse i java script kod. U takvoj strukturi se za svaki vidžet morao generisati posebni HTTP zahtev. Odgovor dobijen od strane servera je kompletna Web stranica. Dešavalo se da pojedine stranice sa kompleksnijim izveštajima imaju i po par stotina HTTP zahteva. Ovo je rezultiralo velikim utroškom memorije, kao i zahtevom za velikim zauzećem procesora u toku obrade prispeli odgovora.



Slika 106 Proces kreiranja i konfigurisanja novog izveštaja

Rešenje ovog problema leži u drugačijem pristupu definisanja vidžeta. Šablonske strane, sastavljene od skupa vidžeta, mogu se smatrati i planom izvršenja HTTP zahteva, detaljnije određen pozicioniranjem i razmeštanjem vidžeta. Ako se prihvati činjenica da je sadržaj vidžeta bitan u trenutku izvršenja njegovog aktivnog koda, a ne trenutku kada se kreira šablon, vidžet u okviru šablona može zameniti jednim DIV elementom.

Predloženo rešenje koristi HTML5 jezik za definisanje vidžeta. Kako će se vidžeti ponašati i izgledati je određeno u njihovom definicionom fajlu u toku izvršenja aktivnog koda na serveru. Rešenje obezbeđuje proširivu arhitekturu, i u isto vreme dozvoljava dodavanje novih definicija vidžeta u toku izvršenja. Kako su bazirani na MVC paternima, svaka definicija sadrži listu mogućih opcija (modela), šablon za pregled (view) i metode koje obezbeđuju funkcionalnost (kontroler). Šablon za pregled obezbeđuje jasno odvajanje između izgleda i ponašanja vidžeta. On prikazuje osobine modela u izlaznom HTML-u.

Različita klijentska okruženja opisana u [71] imaju konceptualno sličan pristup korisničkim interfejsima. Pristup razvoju baziran na modelu usvojen je od najvažnijih koncepata korišćenih za implementaciju portabilnih korisničkih interfejsa. Na osnovu ovoga, možemo uzeti Model-View-Controller u distribuiranim podešavanjima kao početne tačke našeg istraživanja. MVC može takođe da se smatra kao uzorak za aplikacije sa spoljašnjim elementima korisničkog interfejsa, kao što je opisano u [72].

U dobro poznatom [72], objavljenom 1993, autori su postavili arhitekturu za adaptivne sisteme zasnovane na interaktivnom modelu koji obuhvata korisnički model i domenski model. U [73], izdatom posle skoro 20 godina, originalna ideja je proširena kako bi se pokrio i slučaj generisanja prilagodljivog korisničkog interfejsa kada se domenski modeli menjaju u toku izvršenja. Iz [99] koristili smo pristup da dizajneri "ne specificiraju predefinisanu prilagodljivost, već koncepte prilagođavanja i okvire".

Prateći pomenuti pristup, koncept kreiranja dinamičkog korisničkog interfejsa na dalje se razvija na takav način da omogući dodavanje i konfigurisanje izveštaja u toku izvršenja. Dinamičko kreiranje UI je kao i u ASP.NET MVC 4 [74], zasnovan na korišćenju šablonskih pogleda i primenom specifičnih stilova. Sličan pristup možemo videti i u Xplain editoru koji dozvoljava konfiguraciju UI toka u toku izvršenja [75].

Standardno MVC radno okruženje, kao ASP.NET MVC 4 [76] i JavaServer Faces (JSF), inicijalno podržavaju kreiranje pogleda u vreme kompajliranja. Nasuprot tome, MIS GUI mora da se kreira u toku izvršenja programa. Kako bi podržali taj zahtev, upoznavanje sa vidžetima kao standardizovanim elementima bili su neophodni. Deo predložene strukture vidžeta je ASP.NET pogled, koji se izvodi u toku izvršenja [77] pomoću mašine za dinamičke šablone [78].

CRUISe [79] podržava sličan koncept vidžeta i njihovog kreiranja. Postavljanje Web UI- zasnovanog na servisu obezbeđuje konfigurabilne komponente koje mogu ponovo da se koriste što predstavlja dinamički kreirani korisnički interfejs. Naše rešenje daje dodatnu proširljivost obezbeđujući registrovanje vidžeta trećeg lica u toku izvršenja.

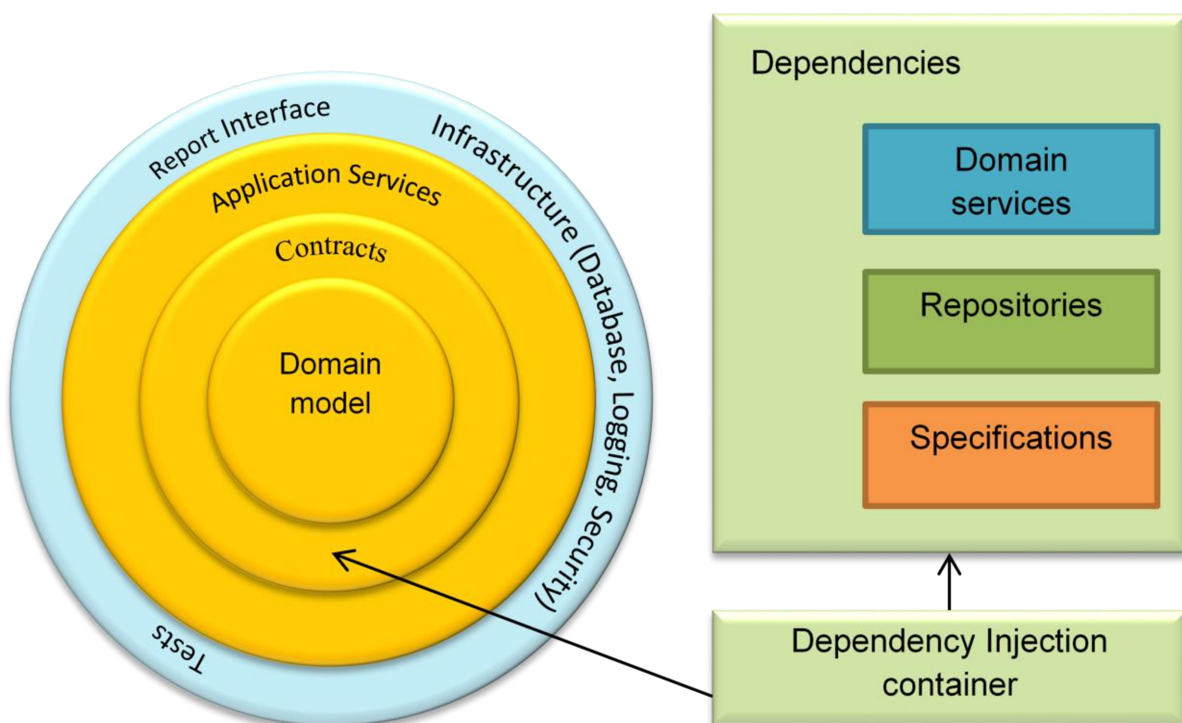
Drugi primer prilagođavanja korisničkog interfejsa zasnovanog na modelu možemo naći u literaturi [80]. On opisuje proces prilagođenja GUI zasnovanog na modelu korišćenjem mešovitih UI komponenti, pravila mapiranja i gradivnih blokova, i generičku prilagodljivu arhitekturu zasnovanu na komponentama. Za razliku od drugih rešenja, predloženo rešenje za UI se prilagođava i u vreme kreiranja (ručno kreiranje vidžeta) i u toku izvršenja (kreiranje šablona i izveštaja).

Predloženo rešenje ima iste osnovne koncepte za predstavljanje korisničkih kontrola kao i Windows Presentation Foundation (WPF). WPF tehnologija zasnovana na XAML je u širokoj upotrebi, to je deklarativni jezik za označavanje, za kreiranje UI [81][82]. Na primer INT-MANUS, sistem koji se koristi u proizvodnim okruženjima, koristi XML i XSL transformacije da opiše i prikaže objekte za predstavljanje [147]. Ovo je sličan pristup kao i u predloženom rešenju, gde su vidžeti unutar šablona

opisani korišćenjem jezika za označavanje. Razlika je u tome da pored vidžeta , koristimo i postojeće HTML5 elemente.

Pristup zasnovan na modelu i generisanje prilagodljivog UI nisu jedini delovi sistema koji su realno potrebni. Jedan od važnih zahteva je mogućnost editovanja šablona kroz vizuelne alate. To znači da vidžeti treba da budu kompaktni, kako bi mogli u okviru alata lako da se selektuju, pomeraju ili da im se menja veličina. U sistemu za upravljanje Web sadržajem, kao Wordpress, korisnici samo programski mogu da menjaju šablone. Dodavanje vidžeta je ograničeno na određivanje dela stanice koja će čuvati vidžet [83].

U predloženom pristupu, konfiguracioni alat predstavlja vidžet kao pojedinačni DIV element. Svi vidžeti se kreiraju na jednoj jedinoj stranici, kreirajući samo jednu web stranicu za ceo šablon, a ne za jedan vidžet. Dok neki imaju koristi od IFRAME-ovog odvajanja skript logike, naše mišljenje je da sa dobrim dizajnom, nema potrebe za ovakvim razdvajanjem. Moderne Javascript biblioteke, kao JQuery, mogu u svakom slučaju da obezbede to razdvajanje . Neke od koristi predloženog pristupa su te da se svi spoljni izvori učitavaju sa jednog mesta, ne u okviru svakog IFRAME-a , komunikacija između vidžeta je jednostavna, različiti GUI efekti kao prevlačenje i puštanje su podržani (čak i obrađivanje preko IFRAME-ova je moguće ali teško). Isti pristup je korišćen pomoću JQuery vidžeta [84][85], koji dopunjuju mnoge moderne klijentske web aplikacije.



Slika 107 Slojevita ("onion") arhitektura [86][87]

U standardnom JQuery pristupu, vidžeti se kreiraju na već prisutnom podskupu HTML elemenata. Na primer, postojeća lista (UL i polje LI elemenata) može se koristiti za JQuery vidžet sačinjen od radio dugmadi. Ovo nije odgovarajuće za editovanje u vizuelnim alatima, gde korisnici mogu da selektuju, pomeraju ili menjaju veličinu vidžeta kao celine. Druga razlika je da su JQuery vidžeti komponente sa klijentske strane, dok predloženi vidžeti sadrže i serversku stranu.

Predloženo rešenje koristi pristup dizajna definisanog za specifični domen [69]. On podstiče korišćenje SOA arhitekture [87] [88] pre nego standardne n-slojne arhitekture. Domenski model je u srži sistema, u potpunosti nesvestan ostalih delova, nezavistan je i lako se testira. Svi ostali su izgrađeni oko njega, i svaki sloj prepoznaje samo slojeve koji su unutar njega. Prezentacioni sloj je u istom nivou kao infrastrukturni nivo. Oni obuhvataju aplikacione servise, sloj sa takozvanim servisnim ugovorima (service contract layer) i domen model. Klijenti šalju zahteve prezentacionom sloju što ima za rezultat komunikaciju sa servisima domena preko prethodno definisanih ugovora.Na isti način, domenski servisi se preko svojih ugovora povezuju sa skladištima podataka. Sve je zajedno spojeno korišćenjem dependency injection mehanizma [89] [90].

5.1 Prezentacioni sloj

Prezentacioni sloj prikazuje izveštaje i preko njega klijenti komuniciraju sa sistemom. Kao što je prethodno naglašeno, postoje dva tipa klijenata u MIS-u: konfigurator i alat za izveštaje. Prvi se koristi da postavi parametre za šablone na osnovu kojih se prave izveštaji. Aplikacioni inženjer kreira nove vidžete, postavlja ih na šablonsku stranu i podešava njihove mesta na strani. Kada je šablon završen, na osnovu njega može da se generišu izveštaji. Inženjer zatim povezuje različite parametre obaveznih podataka za vidžet i tako konfiguriše kako će izveštaj da se ponaša i koje podatke da prikaže. Konfiguracija izveštaja ostaje u bazi podataka ili se snima u poseban XML fajl. Posle toga, krajnji korisnici mogu da otvore izveštaj u drugom klijentu – alatu za izveštaje. Sem toga, pošto vidžeti podržavaju dinamičku promenu sadržaja, izveštaj koji je jednom otvoren osvežavaće svojih izgled u definisanom periodu i omogućiti korisnicima da vide promenu parametara u vremenu bliskom realnom.

Kao i kod razvoja celog sistema, i ovde je dilema bila da li alat za izveštaje da bude desktop ili Web aplikacija. Ovde smo se odlučili za Web aplikaciju baziranu na HTML5 i ASP MVCu. Sa rastom HTML5, postoji nada da jedna tehnologija može da radi na svim klijentskim uređajima. Ideja je jednostavna – umesto da razvijamo GUI korišćenjem različitih tehnologija za svaki uređaj, koristimo samo jednu tehnologiju koja svima odgovara. Zvuči nestvarno, ali da li je HTML5 dorastao tom zadatku? U ovom trenutku je prerano da damo odgovor na ovo pitanje. Sa jedne strane vidimo da se u nekim industrijama odustaje od ovog opšteg pristupa [91], ali sa druge strane, fokus GUI razvoja je sigurno pomeren sa desktopa ka Web-u [92].

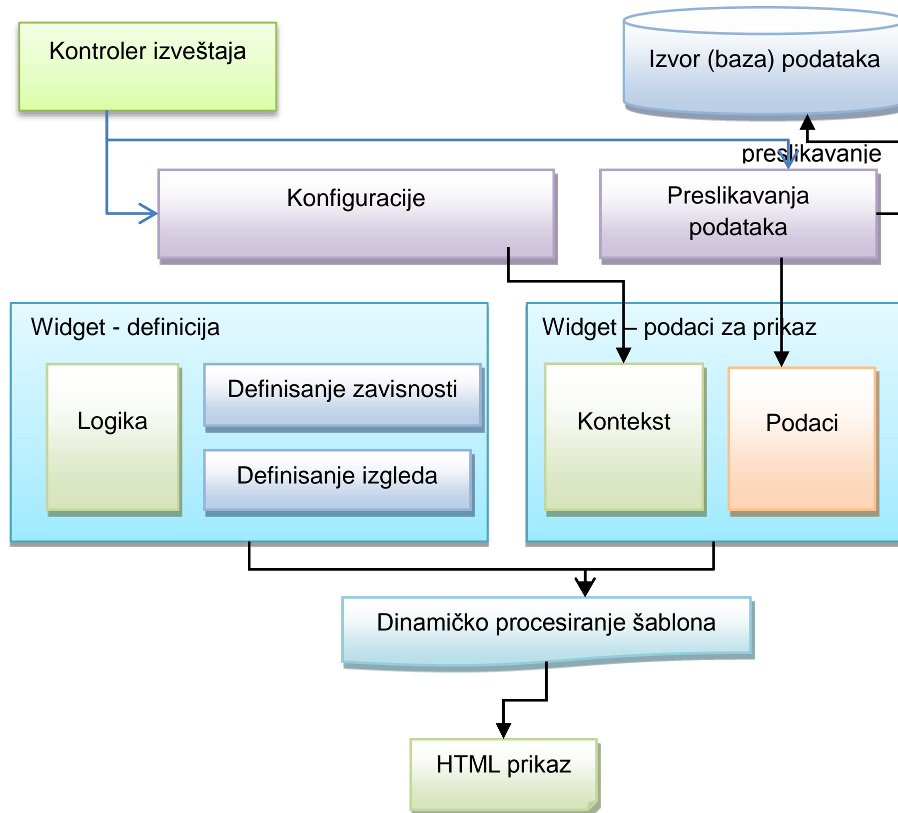
Uzimajući ovo u obzir, odlučeno je da alat za izveštaje treba da bude Web orijentisan, i usaglašen sa HTML5. Glavni principi koje i konfiguracioni i klijenti za izveštavanje moraju da zadovolje jeste to da moraju da zadrže minimum, od poslovne logike. Ovo će poboljšati učiniti sistem relativno imunim na promene u poslovnoj logici. Kada se promene sadržaji pojedinih pregleda, inženjeri iz računskog centra doma zdravlja biće u stanju da jednostavno prilagode sistem tako što će pisati sopstvene skriptove i proširiti UI kreiranjem novih vidžeta. Konačno kreiranjem novih šablona i izveštaja zasnovanih na ovim vidžetima, sami će doprineti razvoju komponente za izveštavanje.

MIS UI mora biti prilagodljiv specifičnim domenima. Na primer, vidžeti, izveštaji i šabloni u bolnicama razlikuju se od onih u domovima zdravlja. Stoga, pristup koji podržava sledeće principe bi trebalo da bude osnova celog sistema:

- jednostavno dodavanje i registrovanje novih tipova vidžeta u toku izvršenja,
- modifikacija vidžeta, izveštaja i konfiguracija šablona u toku izvršenja,
- jednostavno povezivanje podataka sa vizuelnim komponentama u vidžetima,

- optimizovana struktura vidžeta koja dopušta editovanje u vizuelnim alatima,
- HTML5 jezik za vidžete i konfigurisanje šablona
- Vidžeti predstavljaju osnovu za proširenje korisničkog interfejsa.

Koncept serverske kontrole, koja se koristi u Webforms [93][94] i Java JSP Custom Tags, je najsličniji predloženom mehanizmu vidžeta. U konceptu serverske kontrole, vidžeti moraju da se opišu serverskim tagovima i propuštaju se do servera kako bi se tamo izvršavali. Sa druge strane, predloženi pristup koristi obične HTML5 tagove, koje mogu obraditi i web pretraživači. Ovo omogućava se u vizuelnim alatima jednostavno razvijaju klijentske Web aplikacije.



Slika 108 Kreiranje sadržaja vidžeta

Arhitektura vidžeta je zasnovana na MVC. Svaki vidžet se sastoji od kontrolera, modela i pogleda. Kontroler je glavna klasa koja sadrži većinu logike. Model sadrži žive podatke i konfiguraciju, ili opisuje kontekst. Kontekst može da se opiše korišćenjem standardnih HTML atributa podataka. Pogled se predstavlja kao poseban dokument – šablon za pogled. Komponenta za interpretiranje kompajlira šablone za pregled, vezuje kontekstni model i podatke i generiše izveštaj.

Za razliku od našeg prethodne verzije sistema, predloženo rešenje ima šablone predstavljene kao običan HTML5, bez elemenata sa serverske strane. Ovo je razumniji pristup od korišćenja XMLa zato što i mi imamo HTML5 elemente na klijentskim stranama. Nedostatak elemenata sa serverske strane eliminiše potrebu za obradom šablona na serveru, zato što jednostavno mogu biti prikazani i menjani u konfiguratoru.

Za razliku od nekih rešenja sa klijentske strane koja se zasnivaju na celokupnoj obradi u okviru Web pretraživača [95], prethodno pomenuti vidžeti sadrže logiku i na klijentu i na serveru. Unutar editora za šablone svaki vidžet je predstavljen jednim HTML elementom (DIV) opisan specifičnim atributima standardnih podataka [96]. Sadržaji stvarnog vidžeta se ne kreiraju se u tom trenutku, pa nema potrebe za delom sa serverske strane. Početna konfiguracija vidžeta, definisana u šablonu, može biti promenjena i smeštena u konfiguracionom fajlu.

Obrada podataka i pripremanje izveštaja kada se dobije odgovarajući zahtev. Nakon toga, se učitava odgovarajuća konfiguracija i odgovarajući šablon. Zatim se oni povezuju sa podacima iz baze ili drugih spoljnih izvora (Slika 108). Kod nekih vidžeta postoji još jedan korak, a to je izvršavanje odgovarajućeg klijentskog skripta. U tom slučaju, server će izvršiti inicijalnu obradu, poslati podatke klijentu, a klijentski skript će ih dalje obraditi. Klijentski skript uključuje i bootstrap funkciju koja će se izvršiti na klijentskoj strani i koja će generisati JQuery vidžet.

5.2 Poređenje sa prethodno korišćenim rešenjima

Kako ciljani MIS postoji u trenutnoj formi poslednje četiri godine, a cela priča sa razvojem MIS-a je počela 2002 godine, imali smo priliku da koristimo mnoga različita rešenja za generisanje dinamičkog korisničkog okruženja sa raznim prednostima i manama.

Za izveštaje su bili korišćeni standardni alati za izveštaje kao što su Microsoft Reporting Services ili Pentaho. Njihovim korišćenjem, bili smo u stanju da napravimo validne izveštaje u prihvatljivim vremenskim okvirima. Ovde je glavni problem bio što krajnji korisnici MIS-a ne pokazuju veliku volju da učestvuju u kreiranju izveštaja, i pomenuti alati im se čine previše složeni za korišćenje.

Sledeći korak je predstavljalo korišćenje alat za generisanje, koji je bio u stanju da generiše inicijalne skupove atributa po kojima je bilo moguće kreirati izveštaje [99]. Ovaj alat je lako je proširiv i omogućuje jednostavno dodavanje novih tipova izveštaja koji se mogu generisati na osnovu kreiranog domenskog modela. Sa ovakvim tipom alata uspeli smo da se približimo našim potencijalnim klijentima, kao i da smanjimo ukupno vreme potrebno za razvijanje novih formi i izveštaja. Naše celokupno iskustvo u ovom procesu opisano je u [49].

Pošto je tendencija ” sve ide na web” sve više i više dominantna, pristup opisan u ovom odeljku izgleda kao logičan nastavak prethodnog rada. Osnova za opisani sistem su već definisani modeli podataka. Što se tiče reakcije budućih korisnika, koji su već koristili alat za modelovanje, one su u velikoj meri pozitivne. Medicinsko osoblje je sa jedne strane zadovoljno zato što sada ima bogatije web klijente dostupne na većem broju uređaja. Takođe, mogu učestvovati u definiciji domena modela na isti način kao što su to činili do sada. Postavljanjem akcenta na web tehnologije za dizajniranje interfejsa, sada se možemo više osloniti na IT osoblje iz medicinskih ustanova.

Tabela 5 prikazuje kvantitativnu analizu sva tri pristupa koja imamo do sada. Na početku, pri radu sa standardnim alatima imali smo značajno lošije rezultate. Regularne sesije za prikupljanje zahteva, inicijalno su bile zasnovane na stakeholder dokumentima i gubljeno je dosta vremena pre nego što smo došli do tačke opšteg razumevanja. Takođe, budući korisnici nisu bili oduševljeni pisanjem i kreiranjem Word dokumenata, imali su običaj da kažu “Proza”. Istovremeno, IT osoblje nije pokazivalo veliko interesovanje da počne rad sa novim alatima i da povećaju obim dnevnih obaveza.

Predstavljanjem glavnog modela podataka, i odgovarajućeg alata za modelovanje [99] počeli su da se osećaju važnim u kompletnom procesu razvoja softvera i dali su značajan trud da definišu sva polja zajedno sa opsezima standardnih vrednosti. Takođe, sami lekari su uspevali da specifikuju mnogo detaljnije izveštaje.

Pomeranjem cele priče ka web aplikacijama povećao se broj uređaja koji mogu biti korišćeni kao klijenti za MIS aplikacije. Doktori sada imaju širi pristup svom poslu, a zadržavaju da utiču na ažuriranje sistema kroz alate za modelovanje podataka. Sa druge strane, IT osoblje pokazuje veće interesovanje da učestvuje u kreiranju novih komponenti. Ispravnost izveštaja koje je generisalo IT osoblje je značajno više nego ranije i konačno počinju da ljude koji rade na razvoju MISa prihvataju kao kolege, a ne kao "neprijatelje" koji su im doneli više obaveza.

Sa tehničke strane, pomeranje ka web interfejsu smanjuje vreme potrebno za ažuriranje sistema. Takođe smanjuje i vreme potrebno za održavanje. Sa druge strane, glavna mana, kao i kod svake druge web aplikacije je – sporiji odziv sistema i pitanja sigurnosti.

Tabela 5 Efekat korišćenja interpretiranih komponenata

Kategorija	Generatori formi i standardne komponente za izveštaje	Primena MDE alata za generisanje	Interpretiranje modela kroz Web aplikaciju
Broj sesija za prikupljanje korisničkih zahteva (po modulu)	4.33	2.16	1.68
Broj iteracija do inicijalnog rešenja	6.66	2.75	2.5
Brzina inicijalnog odgovora nakon zahteva za proširenje sistema	4 dana	1 dan	1 dan
Izveštaji o bagovima nakon unapređenja sistema (po modulu)	9.65	3.74	3.
Trening za medicinsko osoblje	5 dana	3 dana	2 dana
Trening za IT osoblje	8 dana	4 dana	5 dana
Validnost modela kreiranog od strane korisnika	NA	72%	84%
Validnost komponenti kreiranih od strane IT osoblja u ZU	19%	44%	68%

6 Procedure i smernice za razvoj i održavanje informacionih sistema

Životni ciklus softvera ili (Software development life cycle – SDLC) predstavlja glavni skup osnovnih koncepata vezanih primenjeno softversko inženjerstvo [97]. U okviru njega se izdvajaju i definišu glavne faze i svi bitni koraci vezani za razvoj, implementaciju, instaliranje i kasnije održavanje i dogradnju softvera.

Što se tiče faze prikupljanja podataka i primarnog razvoja i instalacije, literatura obiluje različitim metodama koje sve imaju svoje mesto i koje su po svojim karakteristikama dobre za određene faze razvoja. Trenutno je glavna podela svih metodologija na tradicionalne i agilne i postavlja se pitanje gde koja od njih može da bude efikasnije korišćena. Proces razvoja je inače veoma lepo dokumentovan, ali što se tiče procesa dogradnje postojećih sistema tu se ne izdvaja nikakva posebna metodologija ili skup preporuka koji bi značajnije mogao programerima da da smernice za razvoj. U ovom poglavlju biće dato poređenje različitih metoda i predložena kombinacija koja bi bila izuzetno pogodna za proces dogradnje i održavanja informacionih sistema.

Informacioni sistem se razvija kako bi pomogao ljudima da lakše obavljaju svoj posao i najčešće je tesno vezan za određeni domen ljudske delatnosti. Krajnji korisnici su ljudi koji izuzetno dobro poznaju pomenuti domen tako da sva izračunavanja i generisanja rezultata na osnovu ulaznih podataka moraju biti dobro testirana i verifikovana pošto će na kraju ti podaci značiti nešto nekom čoveku.

Proces samog razvoja informacionih sistema je takođe veoma zahtevan i sa stanovišta organizacije i upravljanja. Izuzetno je bitno da osoba zadužena za upravljanje razvojem bude neko ko može realistično da sagleda sve izazove i mogućnosti svog tima i na taj način najbolje dovede posao do kraja.

Što se tiče različitih pristupa SDLC-u trenutno su dominante dve grupe metoda čiji zagovornici se inače trude da prikažu kako su jedne bolje od drugih – to su tradicionalne i agilne metode. Kao i u mnogim drugim oblastima, vreme će pokazati da je istina najverovatnije negde na sredini i da su jedne metode bolje za jedan skup faza razvoja, a druge su efikasnije pod drugačijim okolnostima.

Pod tradicionalnim metodama se smatraju model vodopada, V-model i RUP (Rational Unified Process). Sve one se baziraju četiri glavna podprocesa, odnosno faze. To su prikupljanje podataka, razvoj softvera, testiranje i instalacija.

U fazi prikupljanja zahteva potrebno je shvatiti kakvu vrstu sistema korisnik zapravo želi, odrediti adekvatni nivo automatizacije procesa i predvideti moguće načine korišćenja. Takođe, treba predvideti najizglednija mesta za buduća proširenja i napraviti plan razvoja. U tom planu treba predvideti vreme i budžet potrebno za završenje projekta.

Nakon toga kreće se sa razvojem samog softvera. Počevši od određivanja tehnologije razvoja i tehničke infrastrukture preko definisanja arhitekture sistema do samog procesa pisanja koda. Sam proces razvoja se najčešće podeli na nekoliko potprocesa koji treba da obezbede ostvarenje pojedinačnih ciljeva u okviru projekta.

Dok faza razvoja još uvek traje, počinje se sa testiranjem. Glavna ideja je da se otkriju potencijalni problemi pre nego što razvoj ode predaleko. Što se neki problem uoči ranije, to će njegovo rešavanje zahtevati mnogo više vremena i napora. Zbog toga se menadžeri projekata trude da što ranije krenu sa inicijalnim testovima i na vreme otkriju greške u arhitekturi, kao i potencijalne sigurnosne propuste. Ovo treba da obezbedi sigurniji proizvod na kraju i da se krajnjem korisniku da na inicijalno testiranje aplikacija koja je u velikoj meri gotova i gde je testiranje došlo do otprilike polovine. U idealnom

slučaju, korisničko preliminarno testiranje se završava kad i faza testiranja razvojnog tima, a onda se pristupa instalaciji sistema. Tada korisnici nastavljaju testiranje i nakon njihove faze testiranja, završava se razvoj softvera i počinje održavanje.

Nažalost, projekat može da ode u potpuno drugom smeru. U slučaju da je korisnik nezadovoljan bilo čime što zahteva značajnu izmenu, ceo proces razvoja dospeva u veoma teško stanje. Vreme za razvoj je već u mnogome potrošeno, a značajna promena mora da bude urađena. Ovo može da rezultuje značajnim kašnjenjem ili kodom lošeg kvaliteta koji su napisali programeri koji su bili pod velikim pritiskom.

Generalno, za tradicionalne metode razvoja se može reći da obezbeđuju mehanizme da se od loše prikupljenih zahteva dođe do softvera na najefikasniji mogući način. Kod tradicionalnih metoda postoji ozbiljan problem sa zahtevima i njihovim sadržajem. Najčešće se zahtevi prikupljaju tako što programeri i menadžeri razgovaraju sa potencijalnim korisnicima i kreiraju dokument koji nakon pregovaranja potpišu obe strane. Nakon toga, počinje razvoj i nakon npr. šest meseci razvoja korisnik dobije inicijalnu verziju softvera. Za to vreme se na strani korisnika promeni mnogo stvari. Na primer:

- ljudi koji su radili vreme prikupljanja zahteva ne rade više
- neko od potencijalnih korisnika nije intervjuisan i nije učestvovao u pisanju inicijalne dokumentacije a zna vrlo bitne informacije potrebne za definisanje pojedinih delova sistema
- procesi značajni za neki deo sistema su jednostavno izostavljeni usled administrativne odluke
- došlo je do previda u toku prikupljanja korisničkih zahteva
- i na kraju, ljudi su zaboravili zašto je neko od rešenja predloženo

Bez obzira na to što se ljudi koji prikupljaju zahteve trude da detaljno opišu svaki deo budućeg softvera i da uključe sve realne korisničke zahteve, glavni nedostatak tradicionalnih metoda je nedostatak tešnje komunikacije sa korisnikom kada proces pređe u fazu razvoja. Nakon završene faze prikupljanja podataka, pristupiće se programiranju, pa onda i testiranju i tek će nakon toga korisnik dobiti prvu verziju softvera. Ako tada dođe do značajnih promena u zahtevima, ceo proces razvoja će se najverovatnije značajno prolongirati. Još gora stvar se onda može desiti sa zahtevima za naknadni razvoj dodatnih funkcionalnosti (change requests). Oni mogu dodatno negativno uticati na softver. Iz ovoga mogu proizaći razni problemi vezani za kompatibilnost i mogućnost integrisanja novih delova sistema. Što je sistem složeniji to je mogućnost nastajanja problema veća. Na kraju, sa tačke gledišta poslovnih procesa, standardne metode se ne mogu smatrati najboljim mogućim.

Sa druge strane su takozvane agilne metode razvoja. One se baziraju na inkrementalnom i iterativnom razvoju u kome se faze u razvojnom procesu ponavljaju za svaku od funkcionalnosti ponaosob. Najbitnija stavka ovde je to što je krajnji korisnik uključen u ceo proces mnogo ranije. Dok je u klasičnim metodama razvoja korisnik dolazio na scenu tek kada je veći deo posla završen, ovde se korisnik uključuje od faze testiranja prve od funkcionalnosti. Glavna razlika između standardnih i agilnih metoda je to što je razvoj u standardnim metodama zapravo jedan veliki složeni proces, dok je kod agilnih metoda ceo proces podeljen na puno manjih potprojekata gde svaki od njih ima sve faze razvoja kao i veliki proces kod standardnih metoda. Kao što je i napomenuto u manifestu agilnog pristupa razvoju, sledeća četiri faktora se nazivaju glavnim faktorima agilnom razvoja:

- Rano uključivanje korisnika u projekat

- Iterativni razvoj
- Timovi treba da se sami organizuju
- Adaptacija na promene

Trenutno postoji šest generalnih metoda koje se svrstavaju u grupu agilnih metoda, a to su:

- Crystal metodologije
- Metod dinamičkog razvoja softvera
- Razvoj baziran na funkcionalnostima (feature-driven development)
- Razvoj baziran na eliminaciji nepotrebnog (lean software development)
- Scrum
- Ekstremno programiranje (Extreme programming)

Agilne metode insistiraju na stalnom kontaktu sa krajnjim korisnikom. Takođe, veći akcenat je na timskom radu i kolaboraciji kao i na iterativnom razvoju gde na kraju svake iteracije korisnik dobija prototip sa tačno preciziranim funkcionalnostima. Sa druge strane, konvencionalne metode insistiraju na ugovorima, planovima, procesima, dokumentima i alatima. Agilne metode smatraju da je povratak uloženog (Return of Investment – ROI) najbitnija osobina koju softverski projekat treba da pokaže krajnjem korisniku.

Za razliku od tradicionalnih metoda, agilne izbegavaju inicijalne dugotrajne sesije prikupljanja podataka. Na samom početku se zato dešava da početne verzije korisničkih zahteva nisu dovoljno detaljne. Ovo je često posledica činjenice da korisnici često ne mogu da se na početku odluče kako treba da izgledaju pojedine funkcionalnosti i šta treba a šta ne treba da bude deo sistema. Zbog toga, agilne metode insistiraju na čestoj demonstraciji koja će pomoći korisnicima da iskristališu svoje mišljenje. Interaktivni pristup omogućava korisnicima da u pojedinim situacijama odlože donošenje odluke sve dok ne budu imali dovoljno informacija o načinu rada budućeg sistema i postanu bliskiji sa tehnologijom.

U ovom delu životnog ciklusa softvera, prednost agilnih metoda je u tome što razvoj može da počne i pre nego što su svi zahtevi formulisani. Zbog mnogo češće interakcije, produkt agilnih metodologija ima mnogo veću šansu da bude bolje prihvaćen od strane budućeg korisnika. Na kraju svake iteracije korisnicima se prikazuje aplikacija koja ima zaokružen određen skup funkcionalnosti. Te funkcionalnosti su najčešće grupisane i module koji se integrišu u ceo sistem. Integracija celog sistema se kod ovih metodologija zapravo vrši onda kada je skup korisničkih zahteva finalizovan. U slučaju velikih promena, modularni sistem se lakše održava i ažurira. Takođe, moduli omogućavaju mnogo lakši pristup kada treba popravljati bagove kao i kada kasnije treba održavati i proširivati sistem.

Pošto se skup korisničkih zahteva razvija zajedno sa sistemom, korisnici imaju dovoljno vremena da ponovo analiziraju sve aspekte svog poslovanja i donesu pravovremeno adekvatne odluke vezane za skup funkcionalnosti koje sistem treba da im pruži. Još jedna dobra strana kod ovakvog pristupa je i to što za svaku promenu zahteva, korisnik može da dobije detaljno objašnjenje od tehničkog osoblja o potencijalnim tehničkim rizicima.

Iako deluje da su agilne metode neuporedivo bolje od tradicionalnih, ipak u određenim fazama imaju značajne nedostatke zbog kojih mnoge organizacije ne žele da ih slede. Prvi na listi problema je to što kod projekata koji su razvijeni agilnim metodologijama ima mnogo manje dokumentacije. U agilnim metodologijama se smatra da je kod sam po sebi vrsta dokumentacije. Zbog ovoga, programeri moraju da stavljaju mnogo više komentara u kod nego kod razvoja standardnim metodama. Prva posledica toga je da novim članovima razvojnog tima treba više vremena da se uklope i da razumeju kod na kome treba da rade. Sledeća posledica je da onda iskusniji programeri moraju mnogo više vremena da provedu objašnjavajući novim kolegama. Tradicionalne metode zahtevaju kreiranje velikog broja arhitekturnih dokumenata i uputstava za razvoj što doprinosi efikasnijem uvođenju novih programera u posao.

Zbog insistiranja na kolaboraciji, članovi razvojnog tima imaju mnogo češće sastanke nego kad se radi po tradicionalnim metodologijama. Sastanci se održavaju makar jednom sedmično, ali je mnogo češći slučaj da se održavaju dnevno. Za mnoge ljude to je suviše naporno i često se može čuti negodovanje od strane programera zbog takve prakse. Takođe, sastanci sa klijentima se održavaju relativno često, što od programera zahteva razvijene veštine pregovaranja i prezentacije. Češći sastanci i više interakcije među ljudima mogu da dovedu i do razvoja neželjenih međuljudskih odnosa. U slučaju kada neko nema dovoljno razvijene veštine komunikacije, to može da se odrazi na ceo tim i ceo proces, zato što njegova objašnjenja mogu biti nerazumljiva svima. To za posledicu može da ima da se od njega često traže dodatna objašnjenja i revizije, što rezultira dodatnim neželjenim gubitkom vremena.

Sledeća loša strana agilnih metodologija je mnogo veći i konstantni vremenski pritisak na programere. Iteracije vrlo često traju svega par nedelja pa su ljudi pod konstantnim pritiskom rokova za završenje nekog posla. Kada dođe do situacije da se sa nekom iteracijom kasni, zato što je npr. bilo potrebno implementirati složene algoritme i detaljno ih testirati, to dovodi do narušavanja komunikacije među timovima. Takođe, kašnjenja u iteracijama mogu da naruše i odnos sa klijentom. Ovaj problem nije toliko izražen kod tradicionalnih metoda, zato što se tamo ne komunicira intenzivno sa klijentom sve dok proces ne dodje do većeg stepena završenosti. Kod tradicionalnih pristupa, programeri su pod manjim pritiskom i mogu da napišu kod boljeg kvaliteta.

Pošto agilni pristup podržava menjanje zahteva po iteracijama, to može da ima dve negativne posledice – jedna se u literaturi naziva rigidnost (rigidity), a druga mobilnost (mobility). Rigidnost se odnosi na promene u sistemu koje izazivaju kaskadne promene u raznim modulima. Mobilnost se odnosi na nemogućnost sistema da enkapsulira komponente koje bi se kasnije ponovo koristile pošto je to vrlo često isuviše rizično i zahteva puno dodatnog rada.

Agilne metode razvoja softvera su definisane kako bi krajnji korisnici bili zadovoljniji razvijenim softverom. Sledeće na listi je bilo skraćenje vremena za razvoj i smanjenje broja bagova. Na kraju, još jedna ideja vezana za njih je bila omogućiti korisnicima da mogu da menjaju specifikaciju i toku procesa razvoja. Iako se čine veoma korisnim sa tačke gledišta savremenog razvoja softvera, još uvek nisu usavršene i moraće da unaprede još par svojih elemenata pre nego što u potpunosti budu korišćene u praksi.

Prvenstveno, kad je reč o dokumentaciji, tu je najveća prednost tradicionalnih metoda u odnosu na agilne. Sam kod se, za sada, ne može smatrati dokumentacijom. Kod može biti razumljiv tehničkom osoblju, ali svima ostalima je težak za tumačenje. Dalje, rigidnost i mobilnost se smatraju velikim problemima pošto mogu značajno da redukuju kvalitet koda. Potencijalno rešenje ova dva problema je

razvoj baziran na modelima i automatizovano generisanje koda kada je reč o komponentama koje dele zajednički skup funkcionalnosti.

Sledeće što se može primeniti je automatska analiza koda koja se danas može definisati kroz veliki broj razvojnih alata. Ovo se može raditi i pre samog testiranja tako što će se kod testirati na predefinisani skup pravila. Primenom ove rutine mogu se izbeći razni defekti u kodu i na taj način kasnije uštedeti vreme i novac. Automatska analiza koda je zajedno sa refaktorisanjem odličan način da se razdvoje delovi koda koji treba da završe u različitim modulima.

Kao što je navedeno i agilne i tradicionalne metode imaju svoje prednosti i nedostatke. Kako trenutno stoje stvari agilne metode se čine idealnim za male projekte, dok su standardne metode, za sada, bolje kod velikih projekata. Zbog toga je za svaki tim bitno koju će metodologiju da koristi kod određenog projekta, kako bi imali najbolje moguće rezultate.

6.1 Uslovi prihvatanja informacionih sistema

Različiti pristupi u definisanju arhitekture, razvoju, i implementaciji informacionih sistema se konstantno razvijaju. Danas već imamo dovoljno metodologija da neke od njih možemo da proglasimo tradicionalnim ili klasičnim. Uspešno se koriste u različitim oblastima, od bankarstva pa do proizvodnih sistema. Medicina bi, tako gledano, trebala da bude još jedno polje gde su informacioni sistemi uspešno primenjeni, ali, nažalost, ta misija se nije pokazala tako jednostavno.

Proces prihvatanja medicinskih informacionih sistema, je posao koji je, u mnogo slučajeva, potrošio mnogo više vremena od očekivanog. Takođe, često se dešavalo i da već urađen i završen softver jednostavno ne bude prihvaćen. Za ovo postoje različita objašnjenja. Najpre, ponekad je previše složena arhitektura sistema dovodila do toga da sistem ima spor odziv, zbog čega su krajnji korisnici gubili strpljenje i vremenom odustajali od korišćenja.

Dalje, često uzrok odbijanja MIS sistema je ležao u nekorektnom korisničkom interfejsu. Interfejs koji nije bio u skladu sa potrebama korisnika, jednostavno je postajao faktor odbijanja. Takođe, nezainteresovanost medicinskog osoblja za korišćenje računara i njihovo nepribližavanje informacionim tehnologijama stvaralo je idealno tle za razvoj straha od korišćenja informacionih sistema, koji je dodatno bio podgrejan i jednostavnim nepoverenjem u softver. Doktori su znali da se „tamo neki Windows ruši" i govorili su da ni softver koji radi na njemu ne može biti bolji.

Međutim, ako se dublje pogleda uzrok pomenutih razloga odbijanja, oni delimično leže u korišćenju klasičnih metoda za projektovanje softvera, kao što je model vodopada, u razvoju medicinskih informacionih sistema. Slaba interakcija sa korisnicima je zapravo bila glavni razlog za odbijanje.

Sledeća grupa faktora koja je vodila odbijanju medicinskih informacionih sistema je bila nedostatak potrebne informatičke infrastrukture. Kada smo prvi put učestvovali u razvoju medicinskog informacionog sistema, sada već davne 2002, glavni problem je bio nedostatak mrežne infrastrukture. Često se dešavalo da je mreža razvedena do prostorija u kojima nisu više računari, a da lekarske sobe nisu imale pristup mreži. Takođe, u to vreme je veoma mali broj lekara uopšte bio voljan da koristi računare.

Ipak, glavno pozitivno iskustvo iz tog vremena je vezano za lekare sa Klinike za neurologiju Kliničkog centra Niš, koji ne samo da su koristili naš softver, već su svojim sugestijama aktivno učestvovali u zadnjim fazama razvoja i čak testirali naš informacioni sistem. Na osnovu njihovih sugestija i zahteva, bio je kreiran informacioni sistem document management tipa koji je podržavao ukupno 135 različitih pregleda i potpregleda. Nažalost, na drugim klinikama, osim par izuzetaka,

nismo imali ni približno tako dobar odziv. Saradnja sa drugim klinikama se svodila na to da dobijemo dokumente u papirnoj formi i da kasnije retko ko proveri kako softver izgleda.

Srećom, deceniju kasnije, medicinski radnici su mnogo voljniji da prihvate novine u radu i da vođenje medicinske dokumentacije povere informacionom sistemu. Kada smo 2002. intervjuisali medicinske radnike, od ukupno 95 njih samo 19 je htelo da učestvuju u testiranju medicinskog informacionog sistema, dok je otprilike jedna trećina (ukupno 31) smatrala da im je MIS uopšte potreban. Kada smo šest godina kasnije počinjali novi značajni projekat, ovog puta usmeren na primarno zdravstvo, podrška je bila oko dve trećine. Na treningu za korišćenje softvera imali smo ukupno 630 ljudi, od kojih 451 smatrao da predstavljeni MIS može da im pomogne u svakodnevnom radu.

Za novi MIS, koji je razvijan od 2008. imali smo drugačiji pristup koji se prikazao pogodnijim nego model vodopada koji smo koristili 2002. Upotrebili smo, u procesu razvoja kombinaciju klasičnog modela vodopada i agilne metode redukcije nepotrebnog (lean software development) kao i razvoja baziranog na modelu (model driven development/engineering). U toku implementacije i dogradnje koristili smo kombinaciju inkrementalnog metoda kao i razvoja baziranog na funkcionalnostima (feature driven development) i modelu (model driven development/engineering). Na osnovu iskustava iz razvoja pomenutog projekta biće dat skup preporuka za razvoj, implementaciju kao i kasniju dogradnju i održavanje sistema.

U razvoj novog MIS sistema uključili smo sve ljude koje smo mogli da uključimo sa strane krajnjeg korisnika – lekare, medicinske tehničare, administrativne radnike kao i tehničko i IT osoblje. Ovo se ispostavilo kao dobar potez, pošto smo osnovno testiranje uspeli da delegiramo IT osoblju iz domova zdravlja. Takođe, izuzetno smo bili zadovoljni brojem i stepenom otkrivenih bagova od strane njihovog IT osoblja. U toku celokupnog testiranja, ukupno četiri IT inženjera prijavila su nam ukupno 276 različitih stvari, dok je 13 lekara koji su učestvovali u testiranju prijavilo 132 stavke.

Tabela 6 Nivo interakcije potencijalnih korisnika MIS-a (za projekat koji je počeo 2008)

Kategorija	IT osoblje	Medicinsko osoblje
A – broj testiranih vrsta dokumenata	41	41
B – broj eskplicitno zahtevanih posebnih vrsta dokumenata	74	59
C – broj izveštaja o greškama na opštim dokumentima	155	64
D – broj izveštaja o greškama na posebnim dokumentima	121	68
E – broj uključenih ljudi	4	13

Novi sistem, koji se razvijao, od 2008 pokazao se mnogo uspešnijim pošto je, nakon inicijalnog pilot projekta u Domu zdravlja Niš, instaliran u više od 25 ambulantnih centara u Republici Srbiji. Samo u niškom Domu zdravlja se dnevno registruje preko 12000 pregleda, laboratorijskih analiza i terapija. Ukupno je aktivan 471 korisnik. Zapravo, glavna lekcija koju smo naučili 2001/2002 je to da krajnje korisnike treba što ranije uključiti u razvoj sistema i konstantno održavati kontakt sa njima, kako već agilne metodologije nalažu. Na ovaj način se dolazi do softvera koji će lakše biti na kraju prihvaćen od strane korisnika i gde će vrlo rano biti uočeni problemi, posebno u korisničkom interfejsu.

Danas je još jedan faktor odbijanja eliminisan. Zdravstvene ustanove danas imaju dovoljan broj kompjutera povezanih na mrežu, kao i ostale informatičke opreme. Takođe, danas gotovo sve

zdravstvene ustanove u Srbiji koriste neki vid MIS sistema. Svi oni nude standardni paket funkcionalnosti definisan od strane Ministarstva zdravlja 2008. Te osnovne funkcionalnosti se odnose na vođenje medicinske dokumentacije, vođenje evidencije o utrošku materijala i generisanju izveštaja koji se šalju Zavodu za osiguranje i Ministarstvu zdravlja.

Kao rezultat svega iskustva vezanog za razvoj i implementaciju medicinskih informacionih sistema, možemo definisati skup procedura i metodologija čija primena daje odlične rezultate. U nastavku će biti opisani posebni pristupi razvoju, implementaciji i kasnijoj dogradnji sistema.

6.2 Definisanje arhitekture sistema

Dok su jedni članovi razvojnog tima mogli da se posvete razvoju korisničkog interfejsa, i da mnogo češće komuniciraju sa klijentima prateći principe agilne metode razvoja na osnovu funkcionalnosti, drugi deo tima je razvijao bazični deo sistema i rešavao arhitekturne probleme primenom modela vodopada i RUP-a.

Najpre je, razvoj MIS sistema koji će zadovoljiti potrebe srpskog zdravstva i koji će biti bazirana na paradigmama "evidence-based medicine" i "patient-centered-medicine" je postavljen za glavni cilj projekta. Upravo to je diktiralo da se posveti jednaka pažnja i korisničkom interfejsu i razvoju centralnog dela sistema. Pomenute paradigme su inicijalno definisane u domenu pružanja zdravstvene nege, ali su u zadnjoj deceniji postale glavne smernice za razvoj MIS sistema. Iako ih lekari nekada tumače kao potpuno suprotne, i mnogi procesi definisani po jednom skupu preporuka se tretiraju kao problematični sa druge tačke gledišta, sa tehnološkog aspekta predstavljaju dve strane iste medalje. Današnji MIS sistemi inače teže da popune prazninu na način predložen u [111].

Evidence-based medicine je noviji koncept i kao osnovni postulat uzima činjenicu da doktor treba da odabere najbolji mogući tretman za pacijenta na osnovu predstavljenih medicinskih činjenica. Kod ovog pristupa pacijentovo mišljenje, lične potrebe i zahtevi se uglavnom zanemaruju ako nisu u potpunoj saglasnosti sa mišljenjem lekara.

Patient-centered medicine je pristup gde svaka medicinska odluka uzima u obzir pacijentove zahteve i konsultuje ga kod svake odluke. MIS treba, kao što je naglašeno u [111] da približi ova dva stanovišta. Dve glavne preporuke za ovo su:

- Dodavanje pacijentovih zapažanja u zdravstveni karton, bez obzira da li su uvažena ili ne.
- Definisanje baznog skupa bitnih medicinskih podataka koji će biti dostupnim svim lekarima uključeni u proces pružanja zdravstvene nege.

Jedna bitna stavka iz openEHR preporuke koja nije mogla da se direktno sprovede u delo je definisanje samo jednog opšteg zdravstvenog kartona pacijenta. Pošto je razvijani MIS morao da prati strukturu i organizaciju primarnog zdravstva u Srbiji, morali su da budu podržani parcijalni zdravstveni kartoni koje pojedina odeljenja vode. Ovaj zahtev je realizovan tako što je ceo skup podataka organizovan oko jednog osnovnog kartona, dok kartoni pojedinih odeljenja predstavljaju samo odgovarajuće poglede na celokupan skup pruženih zdravstvenih usluga.

Domovi zdravlja su najčešće organizovani oko nekoliko odeljenja. Najčešće, tu su opšta praksa, pedijatrija, ginekologija, stomatologija, internističko odeljenje. U zavisnosti od veličine doma zdravlja može biti i nekoliko desetina odeljenja, kao što je to slučaj u Domu zdravlja Niš. Uz ova odeljenja, najčešće idu i laboratorija i osnovna dijagnostika – rendgen, ultra zvuk i EKG.

Tipičan dom zdravlja se sastoji od centralnog dispanzera i više manjih ambulanti koje su rasute po okolnim selima ili prigradskim naseljima. U okviru jedne manje ambulante najčešće se nalazi jedna ordinacija za lekara opšte prakse i još jedna za specijalistu. Najčešće se posete specijalista organizuju tako, da u toku jednog dana bude prisutan jedan specijalista. Uz ovakvu organizacionu topologiju mora se definisati takva arhitektura sistema, da omogućava povezivanje više udaljenih čvorišta uz ograničenje vezano za sporiji internet.

Što se tiče zahtevane fleksibilnosti sistema, na nivou primarnog zdravstva često se menjaju izveštaji koje sistem generiše i šalje Ministarstvu i RFZO-u, dok je broj i oblik podržanih pregleda gotovo nepromenjiv. Sa druge strane, u ustanovama sekundarnog i tercijarnog zdravstva, promena broja i strukture podržanih pregleda je mnogo dinamičnija. Zbog toga, sistem mora da bude definisan kao modularan i lako proširiv. Moduli se definišu tako da podrže osnovne funkcionalnosti u okviru jednog odeljenja ili službe. Na osnovu pomenutih zahteva, razmatrana su dva različita pristupa razvoju:

- informacioni sistem baziran na Web tehnologijama
- razvoj servisno orijentisanog (Service Oriented Architecture – SOA) informacionog sistema definisanog kao skup distribuiranih aplikacija

Prvi izbor je bio razvoj informacionog sistema baziranog na Web tehnologijama. U prilog tom izboru išle su sledeće činjenice:

- Postoji nekoliko efikasnih Web aplikacija sa pratećim razvojnim alatima
- Korisnici ne bi morali da instaliraju nikakav dodatni softver, već bi aplikacijama pristupali kroz Web pretraživač
- MIS sistemi bazirani na Web tehnologijama smatraju se prihvatljivim rešenjima

MIS bazirani na Web tehnologijama se koriste kako kao informacioni sistemi opšte namene, tako i u mnogim posebnim granama medicine kao specijalizovane aplikacije. Na tržištu, spadaju u drugu najuspešniju grupu, posle standardnih troslojnih aplikacija. Razvijale su ih i velike firme (Siemens Soarian [112]) i open source konzorcijumi kao što je OpenEPR [113]. Takođe, najzastupljeniji MIS u primarnom zdravstvu u Srbiji sa udelom na tržištu od oko 50% je MIS baziran na Webu - Heliant [114].

Najveći problem sa Web rešenjima je bio nedostatak telekomunikacione infrastrukture i loša internet konekcija u određenom broju izdvojenih zdravstvenih stanica. Pošto je zahtev bio da sistem može da funkcioniše na terenu, distribuirana SOA aplikacija koja podržava replikaciju i spajanje podataka je bila bolje rešenje. Kao što je prikazano u [115] odziv Web MIS sistema u odnosu na SOA sisteme je za red veličine sporije. Prosečan odziv Web baziranog sistema je oko 0.8s, dok MIS baziran na SOA odgovara na zahteve za prosečno 0.04s, što je svega 5% od vremena koje troši Web bazirani MIS.

Pre početka našeg projekta, kolege iz Računskog centra niškog Doma zdravlja probale su sa uvođenjem pilot Web MIS-a, ali efekti su bili prilično daleko od očekivanog. Probni Web MIS je radio dobro tamo gde su veze bile dobre, ali generalno nije bio od koristi udaljenim ispostavama i terenskoj službi. Takođe, kada bi se funkcionisanje mreže usporilo problemi su se odmah uočavali u Web stranicama koje su imale puno vizuelnih elemenata, odziv sistema je bio na granici prihvatljivosti. Sa tačke gledišta lekara, ažuriranje klijenata je mnogo manji problem nego spori odziv ili nemogućnost korišćenja Web baziranih sistema.

U svakom slučaju, analizirano je mnogo različitih i zanimljivih Web MIS rešenja, iz kojih su preuzete određene ideje i smernice za rešavanje mnogih praktičnih problema. U [116] je predstavljen Web MIS za hitne službe. Pošto je njihov posao skopčan sa potrebom da se reaguje najbrže moguće, autori su odabrali Web sisteme sa pažljivo dizajniranim korisničkim interfejsom. Iz ovog rada su preuzete osnovne preporuke za razvoj korisničkog interfejsa, kao i filozofija da se njegovom razvoju mora ipak posvetiti veća pažnja.

Dobar primer primene domenskih modela podataka za razvoj MIS sistema predstavljen je u [117]. Ovde je prikazan MIS koji se bavi jednim konkretnim zadatkom – negom pacijenata obolelim od hemofilije. U ovom radu je predstavljen pogled na proces definisanja slučajeva upotrebe kao i primer izvođenja domenskih modela iz postojećeg standarda. Ovaj pristup je upotrebljen prvenstveno za generisanje novih formi za specijalističke i subspecijalističke preglede. Na osnovu uputstva prikazanih u razvijan je specifični domenski model predstavljen u ovom radu.

Na kraju, izabrano je da se sistem gradi kao SOA sa takozvanim smart klijentima [121], umesto Web MIS-a. Ipak, Web aplikacije nisu odbačene u potpunosti, pošto je sistem za generisanje izveštaja realizovan kao Web aplikacija sa interpretiranim komponentama. Takođe, svi servisi za pacijente biće Web aplikacije. Ipak, za aplikacije baznog MIS-a, smart klijenti se čine kao bolji izbor.

SOA rešenja se primenjuju u informacionim sistemima koji nadgledaju proizvodnju u fabrikama (Manufacturing Execution Systems – MES), gde je rad sa velikom količinom podataka i velikim brojem povezanih klijenata uobičajen. Sa te strane, SOA garantuje bolji odziv, veću skalabilnost i veću stabilnost sistema. U odnosu na troslojne aplikacije i Web bazirane sisteme, SOA je noviji pristup, pa samim tim ima i najbržu stopu rastu u tržišnom udelu. Takvi sistemi se mogu naći u našem zdravstvu kao što je na primer ZipSoft [122], a i među vodećim svetskim kompanijama kao što su GE Healthcare [123] i Allscripts [124].

SOA arhitekture su dobar izbor i sa stanovišta ne samo razvoja softvera, već se čine i efikasnijim u fazi održavanja. U [115] i [125] autori su obrazložili da novorazvijeni SOA bazirani informacioni sistemi pružaju lakši pristup kod održavanja i dogradnje sistema, dok u toku razvoja nemaju značajnih nedostataka u odnosu na druge pristupe. Mnogo radova, kao što su [115] i [126], se bavi opisom uspešnih dizajna MIS sistema baziranih na SOA.

Teg koji je prevagao da se razvija SOA sistem je analiza predstavljena u [127]. Ovde su predstavljeni razni izazovi koji se predstavljaju pred različite tehnologije, kao i različiti pristupi agregaciji podataka. Pomenuta analiza, kao i zahtev da sistem radi u okruženju koje nije povezano na mrežu, podstakli su nas da razvijemo i uprošćenu verziju aplikacije, koja bi radila na malom broju kompjutera (jedan do 5) i kasnije sinhronizovala svoje podatke sa centralnom bazom. Da bi ovaj koncept uspešno radio, neophodno je bilo implementirati i replikaciju podataka [128] [129] kako bi se osigurala konzistentnost podataka.

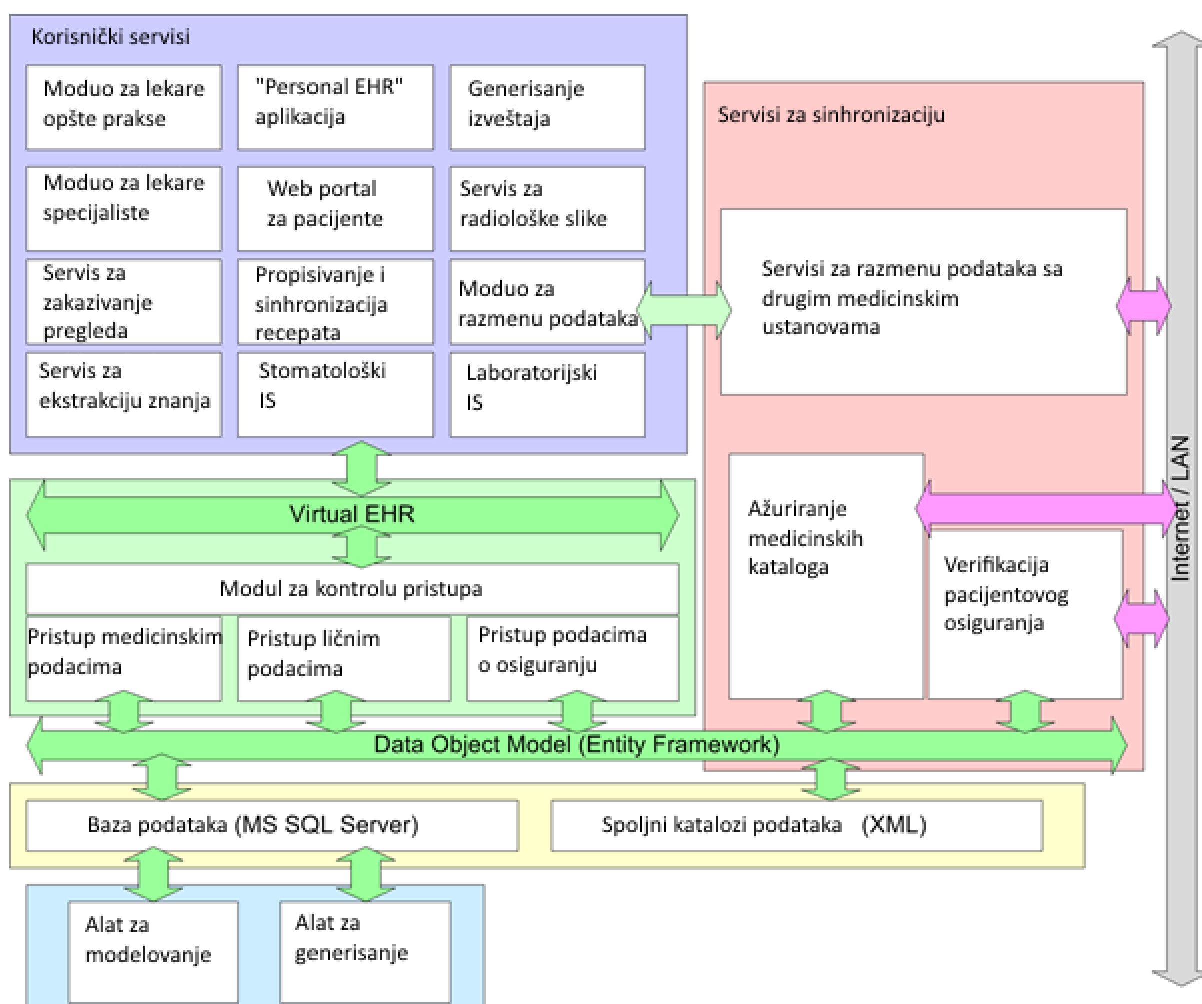
Arhitektura SOA baziranog MIS sistema koji je poslužio za verifikaciju posebnog pristupa razvoju sastoji se od nekoliko nivoa i servisa čije su funkcionalnosti podeljene u module (Slika 109). Sistem je podeljen na nekoliko većih celina i to su:

- Korisnički moduli i servisi
- Sinhronizacioni servisi
- EHR sistem sa modulom za kontrolu pristupa podacima

- Objektni modeli za pristup repozitorijumima podataka (baze i eksterni XML katalozi)
- Alati za modelovanje i generisanje koda

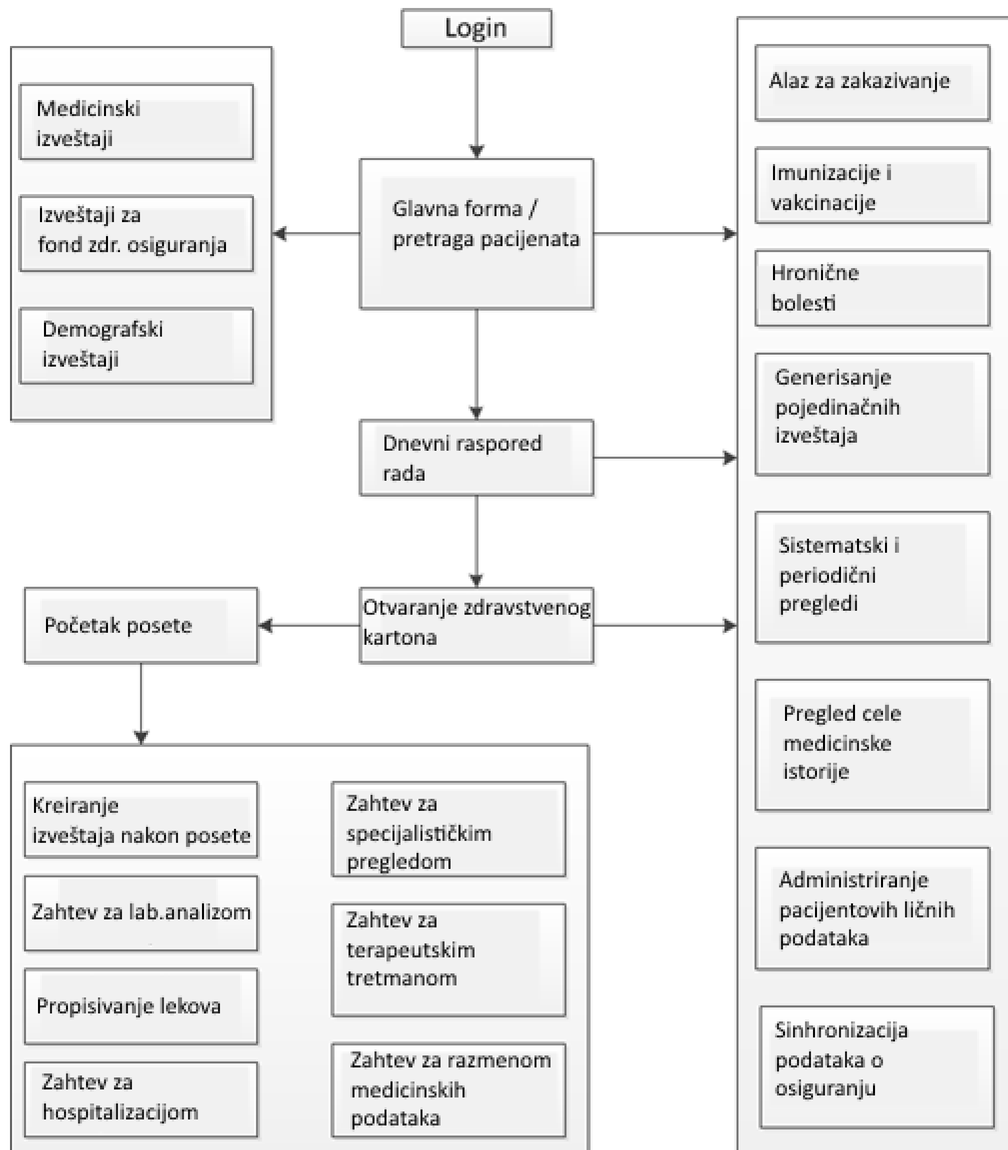
Korisnički moduli i servisi su skup različitih modula koji rade na strani servisa, kao i klijentskih aplikacija koje su razvijene kroz različite tehnologije i dostupne su širokom skupu korisnika. Svi moduli namenjeni za podršku radu lekara unutar institucije, realizovani su kao smart klijenti koji komuniciraju sa servisom kroz odgovarajući modul. Sa druge strane, moduli namenjeni pacijentu imaju interfejs razvijen kao Web aplikacija. Laboratorijski i radiološki informacioni sistemi se oslanjaju na istu bazu, objektni model i EHR, ali su implementirani kao standardne troslojne aplikacije. Generator izveštaja je Web aplikacija nove generacije (Web 2.0) koja podržava konfigurisanje i generisanje izveštaja kroz mehanizam interpretiranja modela i kreiranja runtime komponenti na bazi šablona.

Sistem je projektovan na osnovu openEHR standarda, tako da su tačke proširenja odmah izdvojene i najpre je bio definisan skup specifičnih interfejsa oko njih. Ovo je značajno za sve komponente koje su kasnije razvijane. Sve komponente su mogle da se razvijaju nezavisno oko tačaka proširenja i da svaka donese svoje posebne funkcionalnosti, a da u isto vreme mogu da dele te funckionalnosti među sobom.



Slika 109 Opšta struktura Sistema

Najbolji primer za to su forme koje su generisane da podrže odgovarajuće preglede. One su modelovane kroz alat za modelovanje, generisane kroz generatorsku aplikaciju a inicijalno su bile uključene u aplikaciju za administriranje baze kako bi se kroz interakciju sa korisnicima njihov izgled podesio, a one mogle da budu testirane u relativno realnom okruženju. Nakon toga, uključene su u projekat informacionog sistema i završile su, u najvećoj meri, kao delovi modula za specijalističke preglede.



Slika 110 Standardni skup slučajeva upotrebe

Što se tiče sinhronizacije sistema sa spoljnim resursima, MIS razmenjuje podatke sa dva velika spoljna izvora – sa bazom Ministrastva zdravlja i bazom RZZO-a. Određeni statistički izveštaji se šalju i drugim medicinskim institutima. Sistem preuzima podatke vezane za kataloge lekova, dijagnoza, uslova i spiskova osiguranika. S obzirom da se struktura dobijenih podataka relativno često menja, više puta je bio pravi izazov menjati preslikavanje na podatke koji se čuvaju u lokalnoj bazi. Ovde je još jednom upotrebljen model podataka kao rešenje. Za sve podatke iz spoljnih izvora i podatke koji se čuvaju lokalno, definiše se model preslikavanja. Poseban dodatak na alat za generisanje, koji se zove

XML2SQL, služi da se preslikavanje podešava. Nakon promene u XMLu koji donosi podatke iz spoljnog izvora, ažurira se preslikavanje, a dalje i tabela u bazi i objektni model aplikacije.

Sledeći openEHR preporuke, EHR je razvijen kao nezavistan servis koji integriše i modul za kontrolu pristupa. Na ovaj način unifikovan je pristup podacima i sistem privilegija je rešen jedinstveno za sve module koji učestvuju u sistemu. Na ovaj način je jednostavno kreirati konfiguraciju informacionog sistema praveći samo specifikaciju uključenih modula. Trenutno, u okviru sistema postoji ukupno 32 različita modula, dok je još nekih desetak u fazi razvoja.

Slika 110 prikazuje standardni skup slučajeva korišćenja u okviru MIS sistema. Ovde su prikazane najčešće funkcionalnosti koje su na raspolaganju lekaru opšte prakse, i koje su najkorišćeniji deo sistema. Osnovne funkcionalnosti su login (sa verifikacijom naloga), servisi za pretragu, EHR, rad sa podacima vezanim za konkretnu posetu, definisanje medicinskih dokumenata, kreiranje odgovarajućih izveštaja i štampanje. Sve ostale funkcionalnosti predstavljaju derivate navedenih. Na primer, sve što je vezano sa „start visit“ nasleđuje osnovne akcije vezane za medicinske dokumente. Dalje, svi procesi levo od „search for the patient“ su izvedeni iz „create a specific report“ itd.

Kada su definisane medicinske procedure, definisane su na način da mogu da postoje i nezavisno i kao deo složenih procedura. Tako na primer, osnovni oftamološki pregled može da postoji i kao poseban pregled, i kao deo sistematskog pregleda za vozače.

Poredivši prezentovani sistem sa najznačajnijim akterima na svetskom tržištu, može se reći da sistem podržava sve standardne funkcionalnosti – od prijema do otpusta kliničkim jezikom rečeno. Ono što u pomenutom sistemu nedostaje, a sastavni je deo ponude svih velikih proizvođača [112] [113][122][123][124], su takozvani „single-sign-on“ i „e-prescription“ servis. Prva od pomenutih funkcionalnosti omogućuje korisniku da poveže svoj Windows nalog sa nalogom aplikacije i na taj način omogući da je njegovo logovanje na operativni sistem u isto vreme i logovanje na aplikaciju. Druga opcija koja nedostaje je „e-prescribing“. Ovakva opcija omogućuje lekaru da svoje recepte automatski prosledi u bazu apotekarskih ustanova. Za sada, ovo još uvek ne može biti podržano pošto ne postoje definisane procedure komunikacije zdravstvenih i apotekarskih ustanova u Srbiji.

6.3 Uloga korisnika u modelovanju sistema

Pre početka rada na MIS sistemu za domove zdravlja, intervjuisali smo ukupno 28 lekara – 12 iz domova zdravlja i 16 koji rade na klinikama. Iako je primarni cilj projekta razvoj informacionih sistema za primarnu zdravstvenu zaštitu, uključili smo u inicijalni razvoj više lekara sa klinika pošto su oni već imali iskustva u korišćenju informacionog sistema razvijanog 2002. godine. Takođe, oni su planirali da unaprede svoje postojeće aplikacije, pa je to bio dodatni razlog da oni budu uključeni u razvoj sistema od samog početka. Dalje, lekari sa klinika su se pokazali kao mnogo zainteresovaniji za učešće u ovakvom projektu, a još jedan plus na njihovoj strani je i to što su mnogi od njih vodeći stručnjaci u svojim oblastima.

Prvi intervju se odnosio na strukturu i obim podataka koju bi lekari želeli da imaju u okviru svojih MIS sistema. Najveći broj odgovora (24 od 28) je bio – celu istoriju bolesti. Ovo je po njima značilo sve podatke o svim bolestima od kojih je pacijent bolovao, zajedno sa zapisima o hroničnim bolestima, izdvojenim bitnim dijagnozama, vakcinacijama, pregledima, rezultatima analiza itd.

Prvi problem koji se ovde javio je bio pravne prirode. Prikazati potpunu istoriju bolesti svih pacijenata svim lekarima je neprihvatljivo sa tačke gledišta zakona o privatnosti podataka. Najveći broj lekara (njih 15) veruje da treba da imaju neograničen pristup svim podacima svojih pacijenata, bez obzira na to u kojoj su ustanovi prikupljeni. Inače, svega šestoro iz ove grupe je smatralo da je neophodno da se

lekarima omogući neometan pristup svim podacima svih registrovanih osiguranika. Sledeća grupa od 8 lekara je imala mišljenje da lekari koji prikupljaju podatke treba da određene preglede označe na određeni način, što bi lekarima sa drugih klinika dalo automatski pristup. Na kraju, grupa od pet lekara je smatrala da podaci sa drugih klinika treba da budu dostupni samo na osnovu upita poslatog drugoj ustanovi.

Što se tiče pogleda na potrebu verifikacije pristupa podacima od strane pacijenta, gotovo svi lekari (25 od 28) je mišljenja da to nije neophodno. Smatraju, da ako je pacijent došao da se leči, da onemogućavanje lekara da ima potpun uvid u pacijentovu istoriju bolesti može da bude jedino kontraproduktivno.

Rezultat prve sesije razgovora je bilo kreiranje inicijalnog modela podataka koji će se koristiti kod MIS sistema namenjenog za domove zdravlja. Definisane vrste zdravstvenih kartona koje su neophodne u primarnom zdravstvu, kao i sadržaj velikog broja neophodnih pregleda i rezultata analiza. Takođe, definisan je skup podataka vezan za pristup podacima pacijenata. Uz uvažavanje mišljenja lekara, rešenje je moralo da prati zakonske smernice i preporuke Ministarstva zdravlja. Konsultovani su i međunarodni standardi za razmenu podataka, kao što su evropski EN13606 i američki HIPAA (Health Insurance Portability and Accountability Act). Ceo sistem je baziran na standardnom modelu kontrole pristupa baziranog na ulogama. Za tu namenu razvijen je poseban administratorski alat koji omogućuje definisanje različitih korisničkih modela i definisanje privilegija na nivou svake pojedinačne akcije. Naravno, podržani su i predefinisani skupovi privilegija, kako bi se administratorima pojednostavio posao.

Što se tiče privilegija, pacijent mora da izabere jednog lekara opšte prakse koji će imati uvid u sve medicinske podatke prikupljene u okviru ustanove gde je karton registrovan. Odabrani lekar je pacijentov glavni lekar koji će imati pun uvid u apsolutno sve podatke o pacijentu, pa i one prikupljene u drugim ustanovama i razmenjene na propisan način. Kasnije, pacijent može da izabere još nekoliko lekara specijalista kojima će dozvoliti pun pristup svojim podacima. Ovome najšešće primenjuju hronični bolesnici, koji moraju u određenim vremenskim intervalima da dolaze na preglede i uzimaju recepte. Ovde se mora napomenuti da dodela potpunog pristupa nekom lekaru, po zakonu, mora biti pokrivena i potpisanim papirnim dokumentom.

Kada pacijent posećuje drugog lekara u okviru iste institucije (u slučaju kada je njegov izabrani lekar na odmoru), lekar će imati uvid u trenutno aktivna lečenja. Od celokupne istorije, imaće na raspolaganju samo listu bitnih dijagnoza. Ukoliko je potrebno, pacijent može da dozvoli drugom doktoru pristup svojim podacima za jednu konkretnu posetu tako što će uneti pin kod u odgovarajuću formu. Pacijenti dobijaju pin kod u zatvorenoj koverti onda kada im se prvi put kreira karton u okviru jedne zdravstvene ustanove.

Što se tiče razmene podataka sa drugim ustanovama, to je moguće jedino po zahtevu spoljne institucije. Kada se primi zahtev spoljne institucije, bez posebne verifikacije pacijenta mogu se razmeniti podaci označeni kao bitni. To su najčešće podaci o aktivnim lečenjima, zapisi o hroničnim bolestima, kao i liste poslednjih propisanih lekova. U hitnim slučajevima može se razmeniti i veći obim podataka, ali to je pokriveno protokolom Ministarstva zdravlja.

Drugi vid komunikacije između ustanova je kada ustanova primarne zaštite šalje pacijenta na pregled u drugu ustanovu. Tada se uz zahtev za pregled prosleđuju i svi podaci sa pregleda i rezultati analiza koje lekar koji kreira uput smatra značajnim.

Sledeći skup sesija se odnosio na zahteve pri izgradnji korisničkog interfejsa. Pošto je ovaj set razgovora došao mnogo pre nego što su mnoge osnovne funkcionalnosti bile gotove, za prezentaciju je korišćena generisana administratorska aplikacija, u koju su uključivane generisane forme kako bi korisnici mogli da testiraju interfejs.

Za veliku većinu lekara (19 od 28), glavni zahtev je bio da forme liče na postojeće papirne dokumente. Interesantno je da su na ovome najviše insistirali ginekolozi i lekari opšte prakse, odnosno lekari koji popunjavaju najviše različitih dokumenata. Za istraživače ovaj zahtev uopšte nije bio značajan. Samo dvoje od osam lekara koji se uglavnom bave istraživanjima smatralo je da je značajno imati interfejs koji liči na dokumente. Opšti zaključak na kraju je bio da treba slediti izgled postojećih dokumenata kad god je to moguće, i da treba korisnički interfejs maksimalno uprostiti. Ovo je izuzetno značajno za prihvatanje sistema na kraju procesa razvoja. Interfejs koji više liči na dokumente olakšava kasniji trening i olakšava prihvatanje sistema.

Kako bi se testirale reakcije korisnika na softver, korišćen je period obuke. Ukupno je na obuci bilo 630 lekara iz različitih domova zdravlja. Najpre smo ih pitali da procene svoje poznavanje informacionih tehnologija. Ukupno 108 njih je tvrdilo da ranije nisu koristili računare, 326 su koristili kompjuter kako bi pristupali internetu, 149 se izjasnilo da koristi i druge aplikacije, Microsoft Office uglavnom, dok je 47 njih imalo već iskustva sa medicinskim informacionim sistemima.

Nakon treninga 82% od ukupno 630 polaznika je izjavilo da je zadovoljno interfejsom i odzivom sistema. Posebno su im se dopali, kako oni kažu, „digitalni" dokumenti. Interesantno je napomenuti da postoji i veliki broj dokumenata koji nemaju striktno definisani izgled. Takvi su određeni interni uputi i izveštaji. Za njih smo dobili sugestije o tome kako treba da izgledaju od čak 235 polaznika treninga. Ovo nam je dalo ideju, da u pojedine module uključimo i dinamičko kreiranje formi, odnosno interpretiranje modela. Uz dodatni konfiguracioni alat i sistem prikazan u poglavlju o interpretiranim komponentama, omogućen je još jedan stepen slobode u kreiranju interfejsa za MIS sistem.

Sledeći zahtev je bio da se maksimalno redukuje unos slobodnog teksta kako bi se onemogućio pogrešan unos podataka. Jedan aspekt rešenja je bio minimizovati potrebu za unosom demografskih i podataka o osiguranju. Kada se koriste papirni dokumenti, u toku jedne posete, medicinski radnici prepišu samo ime pacijenta najmanje četiri puta. Ponavljanje teksta koji treba pisati povećava verovatnoću nastanka greške. U nekoliko domova zdravlja, proverio sam svega nekih petnaestak kartona u kojima je bilo 476 različitih medicinskih dokumenata. Usled brzog pisanja u mnogim slučajevima nije lako identifikovati ime i prezime. S obzirom da je to u velikoj meri subjektivna procena, ja nisam mogao da dešifrujem ime u 56 od 476 pomenutih dokumenata.

Sledeći značajan zahtev je bio vezan za štampanje dokumenata. Kako u Srbiji ne postoji zakonska mogućnost čuvanja jedino digitalnih dokumenata, opcija štampanja mora biti dostupna na svakom dokumentu. Ovde je podržano štampanje u slobodnom formatu, kao i štampanje kroz odabrani šablon.

Naš predlog koji se pokazao kao izuzetno efikasan je bila izrada komponenti za pretragu i selektovanje vrednosti. One su opisane u prethodnom poglavlju i omogućuju korisniku da odabere vrednost iz kataloga na osnovu par unetih karaktera. Na ovaj način se izbegavaju greške kod unosa naziva dijagnoza, lekova, pomagala i sl. Ovakve komponente doprinose povećanju opšteg pozitivnog utiska o sistemu zato što se mnogim korisnicima čini da ih sistem prati i zapravo im pomaže.

Bez obzira na to koliko se pregleda definiše kroz alat za modelovanje, u toku eksploatacije sistema neprekidno se javlja potreba za novim pregledima, kao i za generisanjem složenih pregleda koji se sastoje od nekoliko osnovnih. Složene preglede na osnovu postojećih može da definiše i sam

administrator sistema kroz definisanje nove vrste složenog pregleda i crtanja dijagrama koji povezuje njegove potpreglede. U slučaju dodavanja novih pregleda, sistem treba da učita biblioteku u kojoj je novi pregled definisan.

U toku razvoja, javlja se i situacija da korisnici hoće da forme za njihove preglede izgledaju potpuno drugačije od onoga što sad imaju kao dokument. Tokom godina rada, uvidelo se šta može biti unapređeno, koje nove podatke treba dodati, a koje izostaviti, ili bar skloniti iz prvog plana. Najbolji primer za to je skup zahteva koje su stomatolozi imali za pregled zuba.

Najviše zahteva za kreiranje specifičnih dokumenata je došao od lekara specijalista. Velikoj većini njih nije bilo dovoljno da imaju samo jednostavnu formu o izveštaju lekara specijaliste, već su želeli strukturisane dokumente sa tačno definisanim poljima. Ovde je do izražaja došao naš alat za modelovanje. U toku razvoja, programer bi zajedno sa lekarom specijalistom dodao opis novog pregleda u model i generisao formu, koju bi onda uključio u administratorsku aplikaciju. Za nekoliko minuta, krajnji korisnik bi imao inicijalnu verziju forme za preglede i mogao bi dalje da definiše potrebne parametre. Automatsko generisanje formi je značajno ubrzalo i ispravljanje grešaka u njima. Pošto je osnovni deo koda došao iz šablonske komponente, najveći broj bagova se dešavao u specifičnim funkcijama zahtevanim za pojedinačne forme. Ovo je omogućilo da se inicijalno vreme ispravljanja bagova u interfejsu smanji sa 90 na 15 minuta.

Ipak, jedan od najvećih izazova je bila obuka starijeg medicinskog osoblja za korišćenje računara. Bez te obuke, korišćenje MIS sistema bi bilo nemoguće i ne bi donelo nikakvu korist ustanovi koja bi instalirala sistem. Srećom, Ministarstvo zdravlja je organizovalo osnovnu obuku za rad na računarima tako da na kraju mi nismo imali prevelike probleme. Imajući sve to u vidu, akcenat je bio da se nikako ne menjaju standardizovane procedure naših korisnika, već da se samo adaptiraju u okruženje MIS sistema.

6.4 Smernice za efikasniji razvoj informacionih sistema

Kao što je u [143] prikazano, uprkos tome što se ulaže sve više napora da se unapredi proces razvoja softvera, vreme potrebno da se dođe do validnog krajnjeg proizvoda se ne smanjuje. Procenat neuspešnih softverskih projekata se nije značajno smanjio tokom zadnjih deset godina, a uzroci toga su vezani za raznorodne rizike koji prate proces razvoja [144][145][146]. Upravljanje procesom razvoja softvera, a posebno informacionih sistema, često je povezano sa neadekvatnim planiranjem, nepoznavanjem tehnologije, i posebno, kod primene tradicionalnih metoda razvoja, gubitkom stvarnog kontakta sa budućim korisnikom.

Sa druge strane, primena agilnih metoda redukuje taj faktor, ali zato donosi mnogo veći pritisak za programere i kvalitet napisanog koda je u znatnoj meri lošiji. Takođe, u isto vreme, psihološki pritisak koje mnogi od programera trpe zbog čestih susreta sa korisnicima ne ide u prilog bržem i efikasnijem razvoju.

Informacioni sistemi, u velikom broju slučajeva, predstavljaju klasu softvera koji se neprekidno razvijaju i kod kojih nije lako povući granicu između različitih faza životnog ciklusa. Takođe, informacioni sistem je veoma retko softver koji korisnik preuzme, instalira i koristi. Informacioni sistem se razvija za određenog korisnika (ili određenu kategoriju korisniku), po specifičnim zahtevima u specifičnom okruženju i veoma je redak slučaj da se instalacija informacionog sistema jednog korisnika može direktno prekopirati kod drugog. Čak kada je i to moguće, svaka instanca informacionog sistema najčešće ima svoju evolucionu priču potpuno nezavisnu od svih ostalih. Svaki korisnik, nakon određenog vremena, dolazi kod proizvođača softvera sa listom zahteva za novim funkcionalnostima i tu počinje proces dogradnje i proširenja postojećih informacionih sistema koji može trajati duži vremenski period i koji, uz manje ili više stresa, vodi do stvaranja novih verzija informacionog sistema.

Zahtevi za proširenje i dogradnju mogu doći u bilo kom trenutku razvoja sistema i u zavisnosti od svoje složenosti mogu implicirati manje ili veće promene u dizajnu aplikacije. Pošto je tema ove disertacije unapređenja procesa dogradnje i proširenja informacionih sistema, primarni cilj će biti razrešenje problema koji dolaze nakon što je inicijalna verzija informacionog sistema razvijena, instalirana i prihvaćena od strane kupca.

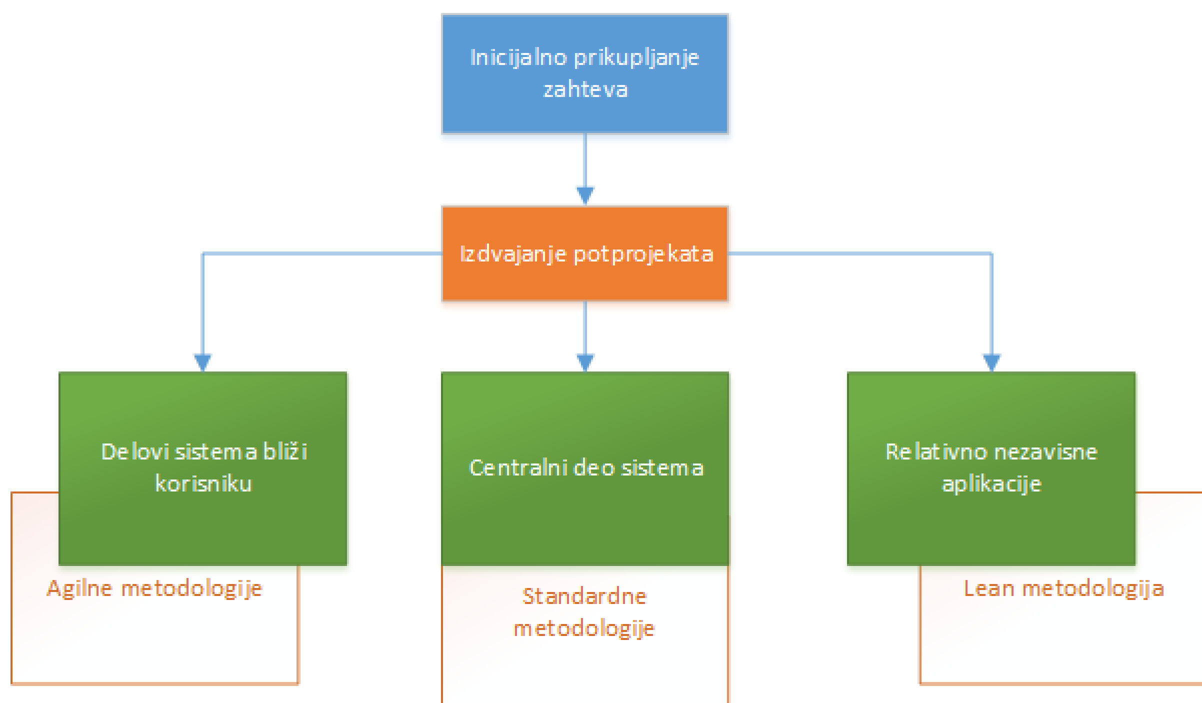
Ono što je bitno predvideti u osnovnoj arhitekturi informacionog sistema su tačke proširenja. Mora se uvek poći od pretpostavke da informacioni sistem neće zauvek ostati isti i ostaviti mogućnost jednostavnog dodavanja novih modula i kao i proširenje postojećih. Dalje, ukoliko je moguće, treba definisati i potencijalne nove tačke proširenja i definisati procedure za njihovo identifikovanje i „otvaranje“ u budućnosti.

Što se primene metodologije razvoja tiče, primenjen je princip kompromisa. S obzirom na to da je razvijan složen sistem sa velikim brojem elemenata koji nisu slični, identifikovano je gde je pogodnije koristiti agilne metode a gde tradicionalne. Generalno, identifikovane su sledeće glavne smernice razvoja:

- Celom procesu prethodi intenzivna faza skupljanja zahteva i gradnje modela sistema
- Integrativni deo celog procesa čini razvoj baziran na modelu podataka. Tačke proširenja, po ugledu na openEHR, treba izdvojiti što ranije i definisati interfejse koji će ih podržati

- Delovi sistema bliže korisniku razvijaće se primenom agilne metodologije razvoja baziranog na funkcionalnostima (feature driven engineering, FDE)
- Generisaće se softverske komponente bazirane na modelu kako bi se skratilo inicijalno vreme razvoja
- Glavni delovi sistema – EHR, Workflow, servisi za pretragu, replikaciju podataka i sinhronizaciju razvijaće se primenom standardnih metoda razvoja Waterfall-RUP
- Delovi sistema koji se razvijaju na standardnom skupu zahteva, koji će se kasnije identifikovati kao nezavisni moduli (radiološlki IS i laboratorijski IS), i gde nije potrebna velika komunikacija sa klijentima biće razvijena primenom lean software development metodologije
- Voditi računa ne samo o tehničkim, već i o društvenim veštinama članova tima, kada se odlučuje na kom će potprojektu da budu angažovani
- Ne kretati u razvoj značajnijih delova sistema sa manje poznatim tehnologijama

Ceo proces razvoja počinje kao kod klasičnih metoda intenzivnim prikupljanjem korisničkih zahteva. U isto vreme počinje i razvoj modela podataka. Bitno je da se već u ovoj fazi, klijentima prezentuje alat za modelovanje i predstavi kao izuzetno značajno sredstvo preko koga oni učestvuju u razvoju.



Slika 111 Izdvajanje glavnih grupa potprojekata i odabir adekvatnih metodologija

Nakon završenog prikupljanja podataka i formiranja neophodne dokumentacije pristupa se razvoju inicijalne verzije baze i objektnog modela podataka. Ovo je jedan od najznačajnijih početnih koraka i ovde se postavljaju osnove sistema i definišu inicijalne tačke proširenja. Kada se završi definisanje inicijalnog objektnog modela, dalji proces razvoja može da krene paralelno u tri pravca (Slika 111):

- razvoj korisničkog interfejsa primenom FDE,

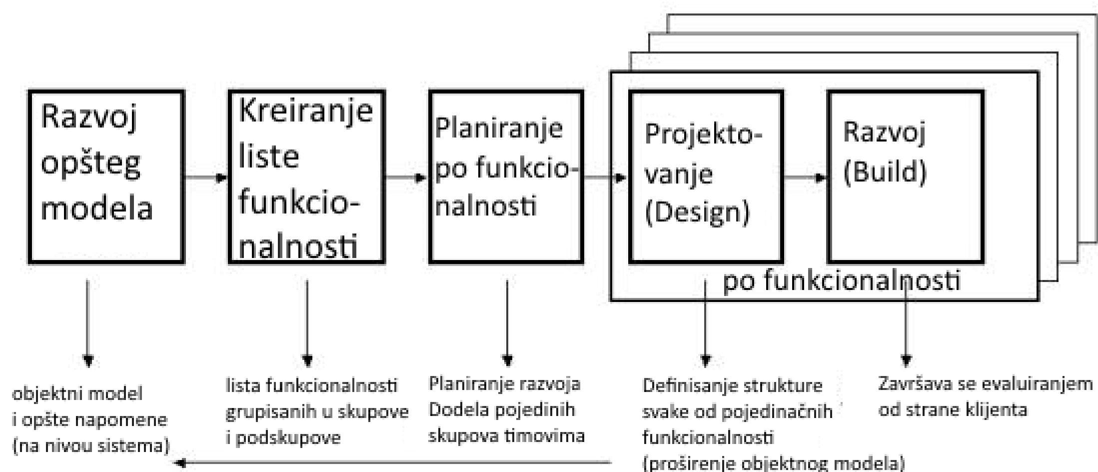
- razvoj osnove sistema primenom standardnih metodologija
- razvoj relativno nezavisnih aplikacija primenom lean metodologije

Tim koji se bavi razvojem korisničkog interfejsa je u najčešćem kontaktu sa korisnicima. Oni počinju svoj posao generisanjem aplikacije za administraciju baze koja predstavlja zapravo probni sto za proveru funkcionisanja strukture objektnog modela i baze, i u isto vreme je okvir u koji će se ugnezditi kreirane forme i omogućiti nezavistan razvoj od poslovne logike na strani servisa.

Dalje nastavljaju sa razvojem modela iz koga će na kraju generisati inicijalne verzije svojih komponenti. Na osnovu kreiranog modela, identifikuju se glavne funkcionalnosti i pravi se plan za razvoj svake od njih. Nakon toga, definiše se dizajn po funkcionalnosti, pristupa se kodiranju i rezultat je po jedan kompajlirani asembli po funkcionalnosti. Pomenuti asembli se nakon toga jednostavno može uključiti u bilo koji projekat primenom dependency injection mehanizma.

U konkretnom slučaju razvoja, sve forme koje su zahtevane za preglede u okviru jednog odeljenja ili službe definišu jednu posebnu funkcionalnost (feature). Korišćenjem alata za modelovanje i alata za generisanje dobijaju se inicijalne komponente, koje se dalje kroz razvojno okruženje finalizuju. Nakon toga, razvijeni moduli sa komponentama koje čine interfejs se integrišu sa centralnim delom sistema, kada njegov odgovarajući segment bude završen.

S obzirom da je GUI deo sistema otvoren ka korisniku, posebno je važno učiniti sve da budući korisnici budu zadovoljni, s obzirom da je loš GUI jedan od glavnih razloga ne prihvatanja sistema.



Slika 112 Razvoj baziran na funkcionalnostima (Feature Driven Development)

Centralni deo sistema, od koga zavisi pouzdanost i stabilnost razvija se primenom standardnih metodologija razvoja i u tehnologiji koja je poznata ljudima koji razvijaju projekat. Nakon prikupljanja podataka, ide razvoj arhitekture, planiranje, programiranje i testiranje. Problem gubitka kontakta sa korisnikom koji se javlja kod standardnih metoda razvoja, u ovakvom kombinovanom pristupu je manje izražen. Deo tima koji radi na GUI delu je u stalnom kontaktu sa klijentima tako da se sve eventualne promene u dizajnu blagovremeno prosleđuju svima. Ono što je u centralnom delu razvoja uključeno iz lean metode razvoja je Kanban tabela, o čemu će biti reči kasnije.

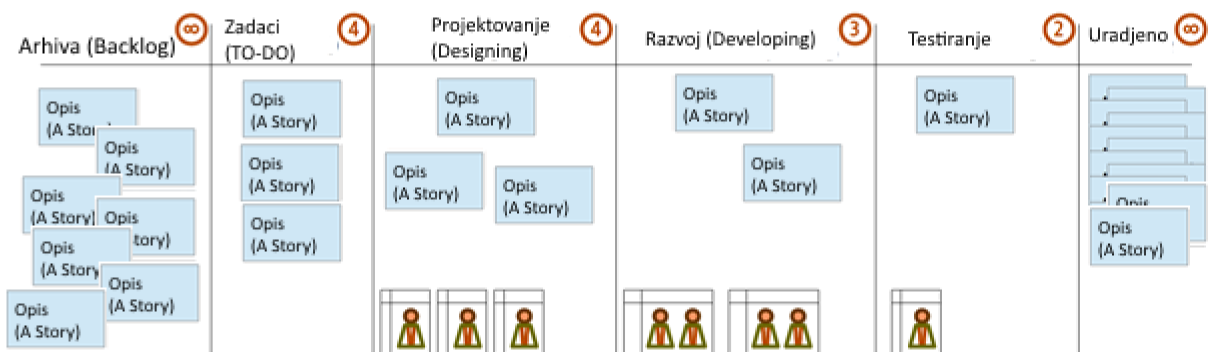
Treći deo razvoja čini razvoj delimično nezavisnih aplikacija kao što su radiološki (RIS) i laboratorijski informacioni sistem (LIS). U ovu grupu spadaju i aplikacije koje nemaju veliki broj

funkcionalnosti koje se baziraju na istoj osnovi, pa je korišćenje modela znatno manje. Kako se one mogu definisati i kao projekti manjeg obima u odnosu na ceo projekat, za njih je prikladna neka od agilnih metoda razvoja.

Kada se birala pogodna metodologija za razvoj ovih sistema glavni konkurenti su bili scrum i lean pristup. Scrum pristup je manje rizičan kod manjih projekata od modela vodopada. Ovde je fokus na kontinuiranoj isporuci potpuno testiranih, nezavisnih, manjih funkcionalnih celina. Na ovaj način se ukupan rizik deli među podkomponentama – ako nešto krene loše u jednom segmentu, ostali segmenti bi trebalo da budu bezbedni. Rad se kod scruma planira u manjim iteracijama, koje se zovu sprintovi, i koje na svom kraju treba da isporučenu završenu komponentu.

Lean i scrum su veoma slični u smislu da je focus razvoja na disjunktnin skupovima funkcionalnosti. Razlika je u tome što lean pokušava da „eliminiše otpad“, odnosno kod lean pristupa najpre se planira, razvija i testira jedna funkcionalnost u svom osnovnom obliku, potpuno se završi i tek se onda pređe na sledeću. Na ovaj način se dublje izoluje rizik na nivo pojedinačne funkcionalnosti. Još jedna prednost koju lean ima je filozofija donošenja odluka, kod lean razvoja je pravilo da se odluka donese što kasnije, odnosno tek kada budu poznate sve relevantne činjenice. S obzirom da su pomenuti LIS i RIS komponente duboko zavisne od spoljne tehnologije, ovakav pristup omogućuje da se eksperimentiše sa odgovarajućim spoljnim uređajima sve dok se stekne adekvatno znanje o njihovom funkcionisanju. Onda se donosi odluka i pristupa se razvoju funkcionalnosti.

Dodatna pogodnost u lean pristupu je korišćenje kanban metode za upravljanje znanjem sa naglaskom na završetku projekta tačno na vreme, vodeći računa o ravnomernom opterećenju članova tima. Kanban je vizuelni alat u kome se prikazuju trenutni zadaci – od definisanja pa do isporuke korisniku. Kanban tabela liči na scrum tabelu za planiranje (Slika 113 Kanban tabela, Slika 114 Scrum tabela), ali je malo obogaćena kako bi se pratio lean proces.

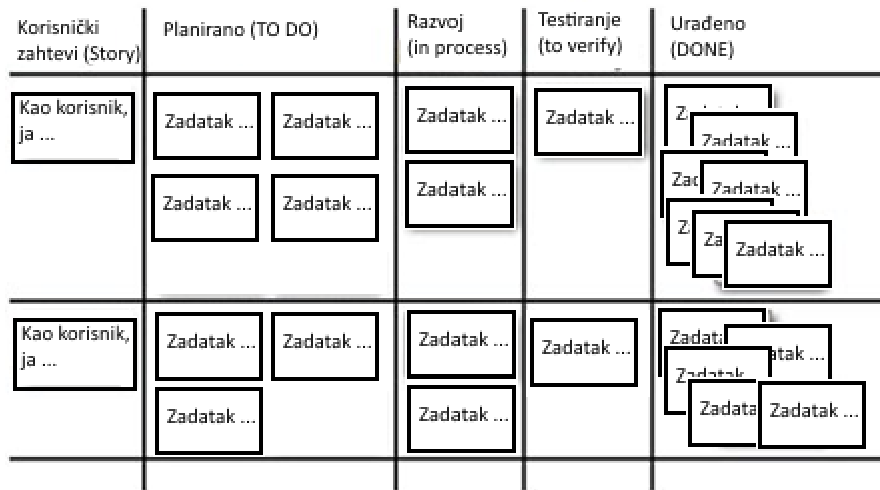


Slika 113 Kanban tabela

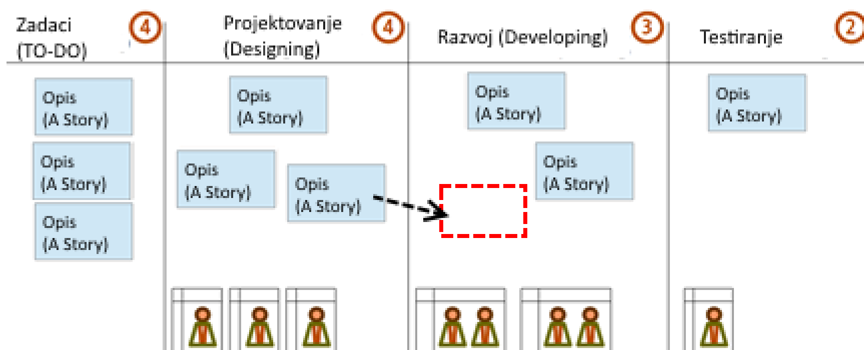
Prva kolona sa leve strane predstavlja listu svih zadataka koji se moraju završiti u nekom trenutku. Na suprotnom kraju stoje zadaci koji su završeni, ali još nisu isporučeni kupcu. Kolona *To-Do* sadrži listu zadataka koji moraju da budu završeni. Sledeća je kolona *Design*, što predstavlja modelovanje i projektovanje rešenja. Kolona *Development*, je zapravo proces pisanja koda. Na kraju je kolona *Testing* gde su završeni zadaci koje drugi developer treba da testira i verifikuje.

Prva razlika koja se uočava između ove dve tabele su brojevi u zaglavlju kanban tabele. Ti brojevi predstavljaju maksimalan broj zadataka po koloni. Svaki tim je zadužen za definisanje maksimalnog broja zadataka, a primer na slici 108 je samo ilustrativan.

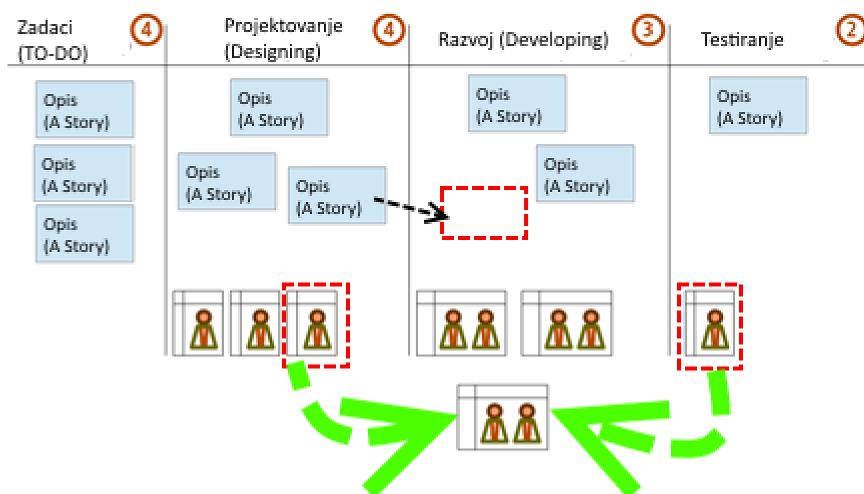
Sledeća razlika je to što Kanban tabela ima na kraju svake kolone kartice sa imenima angažovanih članova tima. Lean i Kanban stavljaju veliki akcenat na kolaboraciju, i tabela mora uvek da bude ažurirana kako bi se videlo ko šta trenutno radi. Još jedno pravilo je da članovi tima ne treba da se preopterete poslom.



Slika 114 Scrum tabela



Slika 115 Promena u rasporedu zadataka



Slika 116 Preraspodela rada kako bi se uravnotežilo opterećenje

Uzmimo na primer da je deo tima koji radi dizajn aplikacije, završio dva zadatka i želi da ih prebaci u fazu kodiranja. Po lean metodologijim to neće biti moguće direktno raditi, pošto ni se na taj način narušio maksimum broja zadataka u koloni. Ovakvi slučajevi se tretiraju kao izuzetni i rešavaju se preraspodelom ljudi. Ovde su jedan od dizajnera i tester krenuli da programiraju nakon što tester više nije imao aktivnih zadataka. U protivnom, zadatak ne bi mogao da pređe u fazu razvoja zato što nema dovoljno resursa.

U prikazanom pristupu razvoju sistema ceo proces je podeljen na različite podsegmente koji pripadaju različitim kategorijama projekata i za svaki od njih je odabrana metodologija koja daje najbolje rezultate. Centralni deo aplikacije se radi standardnom metodom, i ljudi koji rade na njemu rade u svom uobičajenom okruženju i sa tehnologijom koju poznaju. Jedino malo odstupanje koje imaju od klasičnog vodopada je korišćenje Kanban tabele.

Ljudi koji rade na projektu koji zahteva intenzivnu komunikaciju sa korisnikom koriste razvoj baziran na funkcionalnostima, dok oni koji razvijaju relativno nezavisne aplikacije koriste lean metodu razvoja kako bi dobili čisto rešenje sa razdeljenim rizikom. Na kraju celog procesa razvoja, timovi koji su radili na nezavisnim aplikacijama i GUI delu imaju alocirano vreme za pisanje dokumentacije i testiranje celog sistema.

Kombinovanjem metoda razvoja iz različitih kategorija, postiže se da se ponište negativni uticaji različitih metoda i da krajnji rezultat razvoja bude softver sa boljim izgledima da bude prihvaćen.

6.5 Smernice za instaliranje i implementiranje informacionih sistema

Dok razvoj informacionih sistema predstavlja jedno iskustvo, njihova implementacija u odredišnoj instituciji je nešto potpuno drugačije. Počev od situacije gde je MIS prvi softver koji se instalira, pa do situacije kada ustanova već koristi različite vrste softvera, od administrativnog do medicinskog, izazovi su prilično drugačiji. Međutim, u svemu tome se može izdvojiti zajednička nit i mogu se formulisati smernice koje treba da pomognu u uspešnijem i bržem završetku posla, uz sve objektivne i subjektivne probleme.

Prvi pokušaji implementacije MIS sistema, pre otprilike desetak godina, bili su bazirani na principu da odjednom instaliramo softver na svim odeljenjima i svim računarima i krenemo sa testovima i treningom za sve članove osoblja. Ovakav pristup je iziskivao veći broj ljudi koji bi bili prisutni na odredišnoj klinici i koji bi instalirali računare i držali obuku budućim korisnicima. Još jedna negativna strana ovog pristupa je i to da kad puno ljudi krene da koristi potpuno novi sistem, mnogi aspekti korišćenja im i dalje nisu jasni i oni zovu proizvođača softvera da pitaju sve što im se učini nejasnim. Na taj način, u početku, imamo veliki broj poziva u pomoć i potrebno nam je više ljudi da ih usluži. Još jedan razlog velikom broju poziva u ovom slučaju je i to što kad se sve odjednom instalira, korisnici nisu sigurni da li imaju adekvatnu podršku u svojoj ustanovi. Upravo zbog toga smo odlučili da u toku implementacije iskoristimo inkrementalni pristup i na taj način kroz par iteracija, u toku malo dužeg vremenskog perioda, postignemo bolje rezultate sa manje ljudi angažovanih na treningu i instalaciji.

Poseban izazov je u velikom broju slučajeva činila potrebna povezivanja sa drugim sistemima koji su ranije već korišćeni u odredišnoj ustanovi a koje je naravno neophodno bilo integrisati sa našim sistemom. Ne treba posebno napominjati, da o postojanju takvih sistema nije bilo reči sve dok nije krenula instalacija našeg softvera. Ovaj problem je rešavan tako što su inače predviđene tačke proširenja korišćenje kao mesta gde će se nove funkcionalnosti povezati. Za ovo je upotrebljen uprošćeni sistem razvoja baziran na funkcionalnostima (feature driven development) i modelu (model driven development/engineering).

Za svaki poseban slučaj implementacije softvera u pojedinačnoj instituciji najpre smo kreirali mapu prethodno korišćenog softvera sa svim međuzavisnostima i implikacijama na naš sistem. Nakon toga bismo kreirali plan implementacije, koji je uključivao odgovarajuće korake i uporedo bi instalirali odeljenje po odeljenje, obučavali ljude i razvijali komponente za povezivanje sa drugim vrstama softvera.

U slučaju kada je potrebna integracija sa drugim softverskim rešenjima od velike pomoći nam je bio IT personal zaposlen u medicinskim ustanovama. Međutim, osim Doma zdravlja Niš, samo su još dve ustanove imale svoj računski centar. Velika većina ostalih institucija imala je samo jednog zaposlenog IT profesionalca dok svaka peta ustanova nije imala nikog ko je plaćen da vodi računa o računarima, mreži i softveru. Prva značajna smernica u implementaciji MIS sistema je obavezno obučiti nekoga od osoblja ko može da reši najveći broj jednostavnih problema sa kojima se korisnici mogu sresti. Ovo je izuzetno značajno u slučajevima kada niko od našeg osoblja nije prisutan u odredišnoj ustanovi.

Interesantno je da u okviru iste institucije imamo situaciju da neki lekari ne koriste nikakav softver, neki koriste neki drugi medicinski softver da vode evidencije o pregledima, dok neki rade sa složenim elektronskim sistemima kao što su ultra zvučni uređaji, rendgen, EEG ili laboratorijski analizatori. Svi ti različiti programi dolaze od različitih proizvođača i imaju potpuno drugačije korisničke interfejse. Problem koji se javlja kod ovih korisnika je to što su se već navikli na jednu vrstu interfejsa, a sada

treba da rade sa nečim potpuno drugačijim. Zbog toga njima posvećujemo posebnu pažnju kako bi sa što manje problema krenuli da koriste naš softver.

Instaliranje našeg softvera počinje od onih odeljenja gde su već koristili neko softversko rešenje pre našeg. Ovo je izuzetno bitno iz dva razloga – najpre velika je verovatnoća da njima ne moramo da objašnjavamo osnovno korišćenje računara i kroz instaliranje našeg softvera prikupićemo zahteve za integracijom sa postojećim sistemima (ukoliko je potrebno). Odmah nakon instaliranja softvera, počinjemo sa obukom budućih korisnika i u najkraćem mogućem roku dobijamo prve korisnike kojima se potpuno posvećujemo kako bi postigli inicijalni uspeh u instaliranju.

Dok obučavamo početni skup korisnika u korišćenju našeg softvera, za korisnike koji ranije nisu koristili računar organizujemo osnovnu obuku. Nakon toga, instaliramo naš softver u jednom po jednom odeljenju i obučavamo buduće korisnike. U našem inkrementalnom pristupu, implementacija u početnim odeljenjima traje duže, ali se tim korisnicima posvećuje malo veća pažnja, kako bi u perspektivi oni mogli da pomognu ostalim korisnicima.

Na ovaj način, u isto vreme imamo tri grupe korisnika – jedne koji već koriste sistem, druge koji prolaze obuku za korišćenje našeg softvera i treću koju čine korisnici na osnovnoj računarkoj obuci. To rezultuje posledicom da ljudi koji pohađaju trening imaju u svojoj ustanovi ljude koji već koriste i sistem i sa kojima mogu da razmene iskustva. Takođe, na ovaj način obezbeđujemo u malo dužem periodu prisustvo našeg osoblja u instituciji, iako u manjem obimu, što kod korisnika stvara dodatnu sigurnost.

Kada dođe do situacije da treba integrisani naš informacioni sistem sa drugim sistemima, onda je svaki put to priča za sebe. Sa strane našeg sistema, definišu se ili nove tačke proširenja ili se koriste postojeće. Zatim se kroz alata za modelovanje definišu neophodni entiteti za razmenu podataka, a potom se generišu potrebne softverske komponente. Nakon toga se implementiraju metode za preslikavanje podataka iz jednog domena u drugi.

Tabela 7 prikazuje efekte inkrementalnog pristupa u implementaciji i dogradnji informacionog sistema na primeru tri zdravstvena centra približne veličine. U centru 1 (u mestu udaljenom 75 km jugoistočno od Niša) primenjena je inkrementalna strategija, dok su u centru 2 (50 km istočno od Niša) instalirani odjednom svi računari na svim radnim stanicama u svim odeljenjima. Prva dva centra su bila među prvima gde je naš softver bio instaliran, tako da je broj prijavljenih bagova bio veći nego kasnije. U centru 3 (110 km istočno od Niša) primenjena je inkrementalna strategija, al je sistem instaliran tri meseca nakon instalacije u centrima 1 i 2. Implementacija i instalacija je bila brža i efikasnija u centrima gde je korišćena inkrementalna strategija (Tabela 7).

Sva tri centra su slične veličine što se tiče broja aktivnih korisnika, broja obavljenih pregleda i generisanih medicinskih dokumenata dnevno, kao i broja stanovnika koji žive na teritoriji koju ovi centri pokrivaju. (kategorije A, B i C, Tabela 7). Ono što se razlikuje je dužina perioda koji je bio potreban da se sistem instalira. U centrima 1 i 3, celokupni period implementacije sistema je bio oko 30 dana, dok je u centru 2 ukupan period za koji je sistem postao potpuno operativan. prešao cifru od dva meseca (63 dana).

U toku perioda implementacije, instaliraju se svi potrebni serveri, sve klijentske mašine, vrši se trening medicinskog osoblja i administratora. Takođe, nadgleda se inicijalnih 7 dana upotrebe sistema. Za vreme pomenutog perioda, broj poziva je u mnogome veći nego kasnije, a najveći broj njih se odnosi na potvrđivanje određenih procedura. Pozivi koje dobijamo u toku tog perioda su i od strane tehničkog i od strane medicinskog osoblja i njihov broj je otprilike podjednak. Iako se najveći broj poziva može

svrstati u kategoriju „podrška krajnjem korisniku“, dobijali smo i izveštaje o bagovima u programu, kao i sugestije za unapređenje sistema (kategorije G i H, Tabela 7).

Još jedna lekcija koja je ovde naučena je to da inkrementalni pristup instalaciji sistema daje, u slučaju medicinskih informacionih sistema, značajno bolje rezultate u velikom broju slučajeva. Pokazalo se da što je klijentska ustanova složenija i sa više korisnika i odeljenja, inkrementalni pristup daje bolje rezultate. U slučajevima kada je klijentska ustanova mala, sa manje od 10 korisnika, inkrementalni pristup gotovo da ne donosi nikakvu prednost. Broj korisnika je faktor koji poništava sve pozitivne strane inkrementalnog pristupa.

Međutim, u domovima zdravlja, čiji se broj korisnika kreće od oko 25 pa do otprilike 500, inkrementalni pristup se pokazao kao znatno bolji.

Tabela 7 Efekti korišćenja inkrementalne strategije u procesu implementacije MIS

Kategorija	Centar 1	Centar 2	Centar 3
A – broj stanovnika na teritoriji zdravstvenog centra	12259	12051	10056
B – prosečan broj generisanih medicinskih dokumenata dnevno	1350	1490	1050
C – broj medicinskog osoblja koji koriste MIS	54	57	49
D – ceo period implementacije i instalacije	32	63	28
E – ukupan broj zahteva za pomoć u toku instaliranja sistema	306	1116	227
F – broj zahteva za pomoć u prvih 30 dana korišćenja	51	173	35
G – broj prijavljenih bagova	17	19	2
H – broj sugestija za unapređenje sistema	14	21	6

6.6 Problemi vezani za GUI i smernice za njihovo rešavanje

Glavni korisnici medicinskih informacionih sistema su lekari i medicinske sestre. MIS treba da bude tako organizovan da je njihov posao okrenut radu sa pacijentima i da minimizuje sve kolateralne administrativne aktivnosti. Takođe, GUI treba da bude jednostavan za korišćenje kako bi lekari više vremena posvetili pacijentu nego računaru.

Intervjuišući veliki broj lekara i medicinskih sestara i tehničara, uspeli smo da klasifikujemo njihove glavne zahteve i probleme koje su imali u radu sa drugim aplikacijama. U toku razvoja sistema, akcenat je bio da se svi ti glavni problemi eliminišu ili makar minimizuju. Kada se razvija GUI za MIS sisteme, programeri moraju da budu izuzetno pažljivi, kako bi budući korisnici na kraju pristali da koriste MIS. Ovde se mora imati u vidu da su mnogi medicinari godinama radili sa dokumentima u papirnoj formi i prelazak na informacioni sistem im neće biti jednostavan.

Glavni problem koji je najveći broj lekara istakao je preveliko skrolovanje po windows i web formama. Veliki broj njih doživljava skrolovanje kao veliki problem i smatraju da određeni skup informacija mora biti prikazan u isto vreme na ekranu. U situacijama kada treba popuniti podatke o

pregledima koji imaju puno stavki, kao što su npr. sistematski pregledi, pokazalo se da je korišćenje vizarda najprihvatljivije rešenje. Opcijama sa kontrolom sa karticama je prihvatljiva samo ako sve kartice mogu da se smeste u jedan red. Ovo je inače, veoma problematično kada treba voditi računa o rezoluciji ekrana, i kada u sistemu postoje radne stanice sa manjom rezolucijom od predviđene. Za značajan broj lekara je čak prihvatljivo da otvaraju i po nekoliko povezanih manjih formi.

Sledeći problem koji je često uočen u drugim programima je nedostatak podrške za opsege standardnih vrednosti. Mi smo, shvativši značaj ovih informacija za lekare, uključili opsege u domenski model. Zajedno sa problemom nedostatka opsega standardnih vrednosti, javlja se i problem sa nedostajućim ili lošim prevodom. Ovo je posebno izraženo kod latinskih naziva. Zbog toga je bilo potrebno predvideti i mogućnosti prevoda za sve natpise i kataloge koji su prisutni u sistemu. Iz istog razloga, i prevodi su uključeni u domenski model. Zahvaljujući tome i generisanim prevodima, natpisi se mogu menjati i nakon instalacije softvera. Takođe, u toku samog rada sistema, može se menjati i skup podržanih jezika.

U mnogim procesima, korisnici su se izražavali da su se osećali „izgubljeni u aplikaciji“. Kada pojedini proces zahteva mnogo unosa ili višestruki izbor, i u isto vreme stoji otvoreno više od jednog prozora, korisnik dolazi u situaciju da mora da pritisne na „Ok“ ili „Sačuvaj“ više od jednom. Ovakav koncept je onda potpuna noćna mora za korisnike MIS sistema. Većina njih kada jednom klikne na dugme da potvrdi nešto, ne prihvata nikakvo objašnjenje zašto bi morali da ponovo kliknu da potvrde nešto drugo. Takođe, ispostavilo se da je veliki problem kada forme izgledaju raznorodno i kada su značajno drugačije od dokumentacije na koju su navikli. Upravo ovde se vidi značaj alata za modelovanje i alata za generisanje koda, pošto oni obezbeđuju uniformnost u funkcionalnostima i izgledu. Kada smo kreirali forme koje treba da podrže procese u domovima zdravlja, vodili smo se principom da forme što je moguće više liče na dokumente, i da se minimizuje potreba da se sa jedne forme otvara druga. Ukoliko je to neophodno, sistem je dizajniran da najviše jedna dodatna forma može da bude otvorena.

Problem sličan prethodnom je i situacija kada lekar kaže da ne može da pronađe pacijenta u sistemu. Ako u toku rada, program premešta pacijenta iz jedne liste u drugu, jedan pogrešan klik mišem prebaciće pacijenta na mesto na kome ga lekar ne očekuje. Zahvaljujući tome, lekar ne zna više šta da radi i to rezultuje gubljenjem vremena i strpljenja. Zbog toga smo u modelovanju samog sistema predvideli opcije kojima lekar može u svakom trenutku da vidi listu gde se njegov pacijent nalazi. Ovo je posebno važno kod složenih pregleda koji imaju složeni skup puteva u grafu koji ih opisuje. Podrška za složene preglede, zajedno sa njihovim putanjama, je zbog toga sastavni deo domenskog modela.

Još jedan značajan problem koji je uočen kod mnogih različitih informacionih sistema je problem preduge pretrage. Nekada kataloški skupovi podataka traju predugo. Još jedna posledica je da onda kontrole za izbor podataka imaju previše redova i korisnik gubi mnogo vremena da ih pronađe. Iz kombo boksa koji sadrži i samo hiljadu podataka, nije jednostavno izabrati jedan. Da bi ovo izbegli, u modelovanju kataloških tabela predviđena je upotreba heš algoritama i kreiranje stabala traženja kako bi se jednostavnije došlo do manjeg skupa podataka.

Zbog toga je u inicijalni skup generisanih komponenata na osnovu modela uvršteno kreiranje komponenti za pretraživanje. One rade tako da korisniku omoguće da krene da unosi vrednosti u neko filter polje i nakon određenog broja unetih karaktera (najčešće 3) prikaže korisniku listu sa filtriranim komponentama. Na ovaj način se značajno smanjuje saobraćaj sa bazom zato što na npr tri karaktera, prosečno se dobija manje od jednog promila vrednosti iz tabele. Dodatno lista može da prikazuje samo

prvih n rezulatata ili da omogući i straničenje. Filtrirane vrednosti su prikazane u listi ispod tekst boksa u koji se unose inicijalni karakteri, i korisnik ih može izabrati korišćenjem kursorskih strelica ili miša. Ova vrsta komponenata je posebno pogodna za kataloge koje korisnici stalno koriste i gde znaju približno mnoge šifre iz šafranika. To su na primer liste dijagnoza i lekova.

Dalje, previše zapisa se izgubi u mnogim programima zato što korisnici ne dobiju pitanje „da li ste sigurni" kada krenu nešto da obrišu ili odbace. Klik na pogrešno dugme odbaciće promene. Zbog toga se u našem sistemu sve akcije ažuriranja i brisanja potvrđuju. Kod ažuriranja se zapravo ne menja postojeći zapis, već se starome postavlja odgovarajući status, a kreira se novi sa novim vrednostima svojih atributa. Isti slučaj je i kod brisanja, pa čak i kod ažuriranja kada se klikne na „poništi".

Pomeranje sadržaja gore-dole ili levo-desno (Scrolling up/down and scrolling left/right) su široko rasprostranjeni po raznim aplikacijama. Mnogi lekari to smatraju izuzetno iritantnim. U svim procesima koji su podržani trudili smo se da izbegnemo skrol barove. Sledeće o čemu treba voditi računa je da vizuelne kontrole koje prikazuju brojeve nekako obaveste korisnika kada je prikazani broj van regularnog opsega. Ista pravila moraju da budu korišćena i kod operacija unosa novih podataka. Ako na primer korisnik unese 40 za temperaturu tela, vrednost će promeniti boju u crveno, ili ako korisnik unese 405 umesto 40.5, sistem reaguje i odbacuje vrednost.

Medicinski informacioni sistemi vrlo često mogu da imaju stotine pa i hiljade korisnika, i svaki vid ubrzanja odziva sistema je značajan. Zbog toga se svaki pristup koji će ubrzati ukupan odziv izuzetno ceni. U ovoj tezi je prikazan i CQRS pristup koji treba da omogući ubrzanje pretraživanja i učitavanja kod onih podataka koji se često koriste a retko menjanju. Ipak, zdravstvo je karakteristično po tome što ipak korisnici rade sa živim ljudima, i sam informacioni sistem nije najbitniji činilac celokupnog zdravstvenog sistema. Kada je odziv sistema kritičan, posebno kod hitnih slučajeva, korišćenje MIS-a umesto papirne dokumentacije je velika promena za nekog ko već 20 godina radi sa papirnim obrascima. U ovim slučajevima, MIS može neobučenom korisniku da prilično uspori posao umesto da ga učini efikasnijim.

Zahvaljujući interfejsu koji „izgleda poznato" uspeli smo da smanjimo osnovni trening za lekare opšte prakse sa inicijalno planiranih 16 dana na nekih 9 časova. Standard za trening je inače tri do pet dana. Generalno, trening za medicinsko osoblje je kraći za 30 do 50% inicijalno planiranog vremena, dok je trening za administrativno osoblje u nivou sa standardima – tri do pet dana.

Ukratko, preporuke iz ovog poglavlja se mogu podeliti na tri grupe. To su preporuke koje se odnose na procese projektovanja, razvoja i instalacije i održavanja treninga za korisnike.

U procesu projektovanja, bitno je odabrati optimalnu arhitekturu koja će garantovati optimalno funkcionisanje sistema i njegovo kasnije širenje. Takođe, mora se imati u vidu kako će sistem reagovati kada postane opterećen velikom količinom podataka. U skladu sa tim treba voditi računa o arhiviranju i brisanju starih podataka. Takođe, treba predvideti redundantne servise i omogućiti rad u okruženju kada je narušena veza između delova sistema. Takođe, treba obezbediti da sistem može da radi sa makar dve različite baze podataka.

Kao što je napomenuto kod opisa agilnih metoda, korisnike treba što ranije uključiti u proces razvoja i dati im da testiraju i evaluiraju što je moguće više komponenti. GUI treba da izgleda slično formama koje korisnici već koriste i izgled formi treba da je konzistentan. Treba pojednostaviti procese koliko god je moguće i smanjiti potreban broj klikova kako bi se neka akcija uradila.

Kada se sagledaju problemi vezani za GUI i problemi koji dolaze iz procesa instalacije i implementacije, može se doći do sledećeg skupa opštih napomena. Odluke koje se vezuju za proces implementacije i instalacije informacionog sistema su takođe značajne i loše odluke u ovom segmentu mogu značajno usporiti ceo proces. Glavne preporuke u ovom segmentu bi mogle da se podvedu pod sledeće:

- Obezbediti da rukovodstvo ustanove podržava ceo projekat uvođenja informacionog sistema
- Obezbediti da makar jedna osoba u odredišnoj ustanovi ima dovoljan nivo znanja
- Obučiti IT osoblje pre ostalih i posvetiti im više pažnje nego ostalima
- Obučavati buduće korisnike grupu za grupom, a ne sve od jednom
- Eksploataciju sistema početi po odeljenjima – jedno za drugim
- Ohrabriti buduće korisnike da razmenjuju znanje i dodatno testiraju sistem

Bez saradnje sa budućim korisnicima ne može se ni zamisliti da će realizovan MIS biti dobro prihvaćen. Pogledi programera i lekara na to kako softver treba da izgleda se u velikoj meri razlikuju i neophodno je da se kroz razgovor dođe do najboljeg rešenja za krajnjeg korisnika. Na ovo ne treba gledati kao na izgubljeno vreme. Više vremena provedenog u fazi specificiranja i inicijalnog testiranja, smanjiće kasnije lutanje i potrebu da se veliki delovi sistema menjanju. Takođe, korisnici koji su više uključeni u sistem, kasnije će ga lakše prihvatiti.

U toku naših projekata, tri činjenice su se pokazale kao značajne za projektante MIS sistema:

- Neutralizovati strah i nepoverenje kod korisnika kada je u pitanju inovacija u njihovom poslovnom procesu
- Neka korišćenje sistema bude što je moguće bliže njihovoj dnevnoj rutini
- Kreirati jednostavan i intuitivan korisnički interfejs

Konzistentan korisnički interfejs je takođe neophodan. Kako bi se smanjio broj korisničkih grešaka ne treba dozvoliti slobodan unos gde god to nije neophodno. Kad god se koriste komponente za izbor vrednosti, treba se osloniti na činjenicu da je već 20 stavki previše da bi se iz njih brzo izabralo. Na kraju, redosled polja treba da sledi poslovni proces a ne obrnuto. Zato je bitno da kad korisnik pogleda formu, ona na prvi pogled treba da izgleda poznato.

6.7 Smernice za dogradnju informacionih sistema

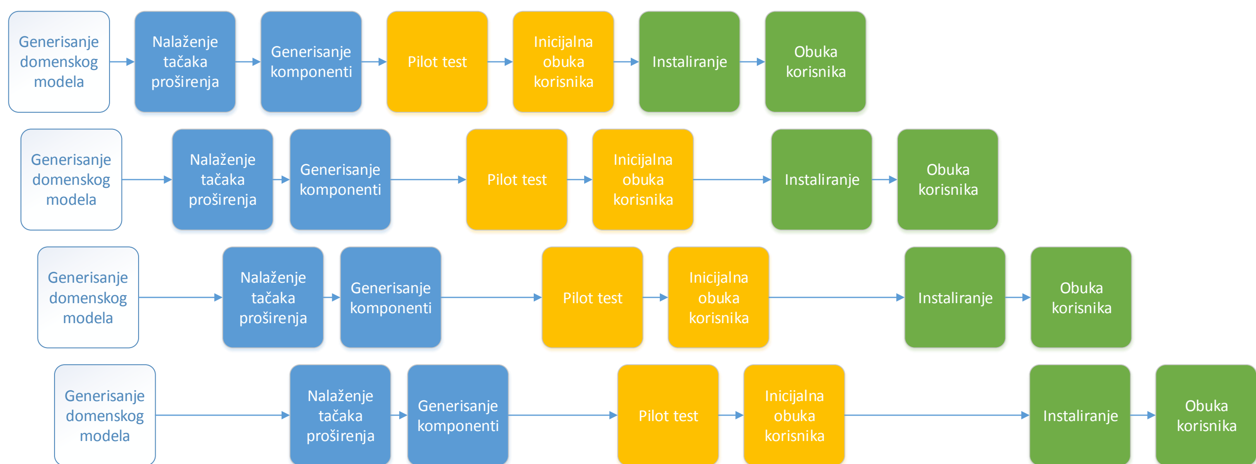
U procesu razvoja korišćena je kombinacija klasičnog modela vodopada i agilne metode redukcije nepotrebnog (lean software development) kao i razvoja baziranog na modelu (model driven development/engineering). U toku implementacije i fokus je bio na kombinaciji inkrementalnog metoda i razvoja baziranog na modelu.

Što se tiče dogradnje informacionog sistema, nakon više od deset godina iskustva, može se izvesti zaključak da se svaki proces dogradnje može tretirati kao nezavisni projekat, manjeg obima nego što je osnovni projekat. Osnovni princip koji se koristi bi bio inkrementalni, a glavni koraci su kao i u svakoj od standardnih metodologija (Slika 117). U zavisnosti od vrste promena koja se zahteva, koristi se metodologija koja je najprikladnija. Kako se svi projekti dogradnje baziraju na već postojećim funkcionalnostima, neophodno je imati stalno u vidu širu sliku celog projekta i težiti razvoju koji će

efikasno i brzo dati rezultate a pri tome ne narušiti opštu stabilnost sistema. Projekti dogradnje i održavanja se na osnovu zahteva mogu podeliti u sledeće grupe:

- Izmena i/ili dogradnja postojećeg modula
- Dodavanje novog modula sa novim funkcionalnostima vezanim za jednu tačku proširenja
- Dodavanje novog modula koji se na više mesta povezuje za glavni sistem
- Integracija sa drugim sistemima

U slučaju kada se zahteva izmena ili dogradnja postojećih modula, lean pristup se pokazao kao najbolje rešenje. U procesu dogradnje se može raditi i na optimizaciji postojećih funkcionalnosti i eliminaciji nepotrebnih delova koda, a u isto vreme se mora imati pogled na ceo sistem. Kako treba voditi računa i o reakciji i verifikaciji od strane korisnika, svi potrebni elementi se slažu da se lean koristi kao vodeća metodologija.



Slika 117 Opšti prikaz procesa dogradnje informacionog sistema

Kada se radi o dodavanju novog modula sa novim funkcionalnostima vezanim za jednu tačku proširenja, razvoj baziran na modelu je logičan izbor. S obzirom da se zahtevaju nove funkcionalnosti u okolini jedne tačke proširenja, proces počinje definisanjem modela i generisanjem potrebnih komponenti. Komponente se uključuju u novi projekat, koji generiše novi asembli koji se zatim uključuje u projekat informacionog sistema.

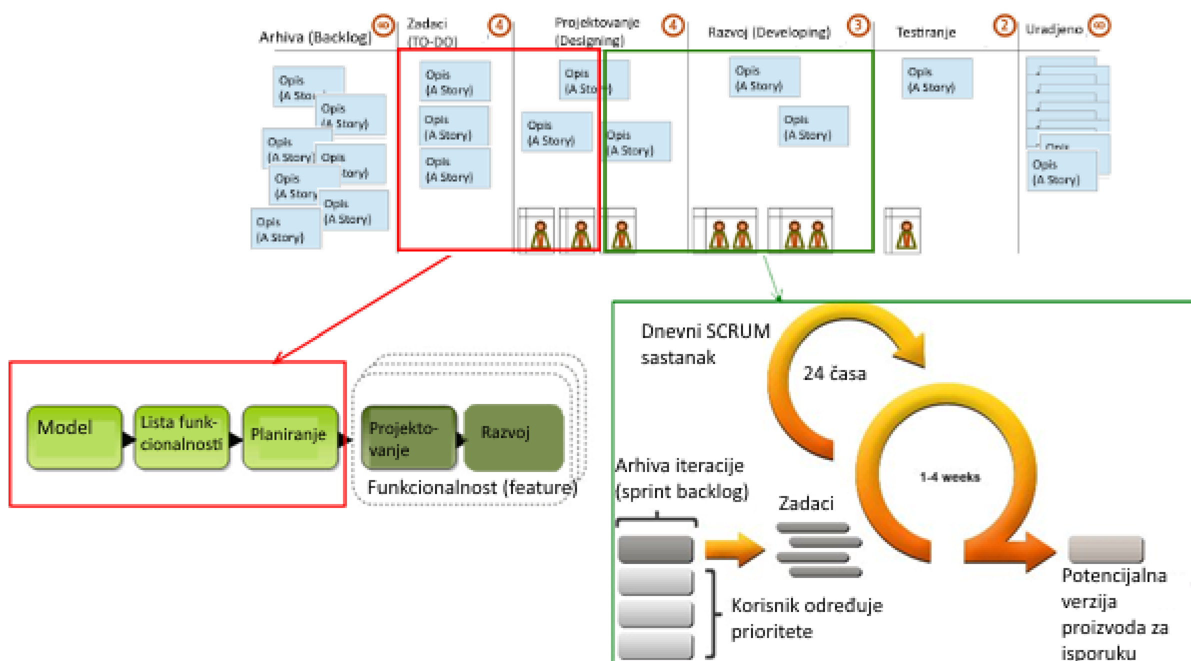
Dodavanje novog modula koji se sa postojećim sistemom integriše na više mesta je najsloženiji vid proširenja sistema. U istoj ravni je i integracija sa drugim sistemima, koja u isto vreme zahteva i rešavanje problema razmene podataka sa eksternim sistemom. On može izuzetno da utiče na ceo sistem i stoga je potrebno obratiti posebnu pažnju na testiranje i verifikaciju razvijenih komponenti. Osnova za ovaj pristup je lean metod, ali sa izvesnim modifikacijama (Slika 118).

Ideja je da se faza arhitekturnog razvoja podeli u dva dela, od kojih bi se opštiji deo pridodao procesu identifikacije skupova funkcionalnosti koje treba razviti. Deo bliži implementaciji pridodao bi se delu razvoja koda, odnosno samom procesu programiranja. Taj deo bi prvenstveno imao definisanja detalja koje nisu u potpunosti projektovani tokom faze definisanja arhitekture.

Nakon prikupljanja zahteva i formiranja backlog liste, elementi iz te liste se grupišu u smislene celine i koristeći se pravilima iz FDD i MDD definišu se skupovi funkcionalnosti koje treba razviti. Takođe, ovde se definišu i prateći modeli podataka, arhitekturni projekat na visokom nivou i kreiraju se funkcionalni zahtevi za fazu razvoja.

Nastavak projektovanja se provodi na nižem nivou koristeći pomenute elemente kao ulaze. Ovaj deo faze projektovanja se zajedno sa fazom razvoja tretira kao skup malih nezavisnih projekata koji će biti razvijeni scrum metodologijom. U okviru faze razvoja odvijaće se i inicijalno testiranje razvijenih komponenti i njihova integracija u sistem.

Dalje, integraciono testiranje vrši se u regularnoj fazi testiranja okvirnog projekta koji se vodi lean metodologijom. Ovako definisane faze su potpuno u skladu sa lean filozofijom. Na kraju svake podfaze dobija se rezultat koji je smisleni ulaz za sledeću fazu. Svaka sledeća faza dodaje novi kvalitet na prethodnu, a integraciono testiranje na kraju služi da verifikuje integraciju novorazvijenih komponenti u ceo sistem.



Slika 118 Modifikovani lean model za strukturnu dogradnju informacionih sistema

6.8 Primer strukturnog unapređenja sistema kroz CQRS pristup

U svim informacionim sistemima se pre ili kasnije javi potreba za optimizacijama. Tokom eksploatacije, uočavaju se mesta gde se može smanjiti protok podataka, optimizovati vreme izvršenja ili implementirati strukturna promena koja će uključiti i jedno i drugo.

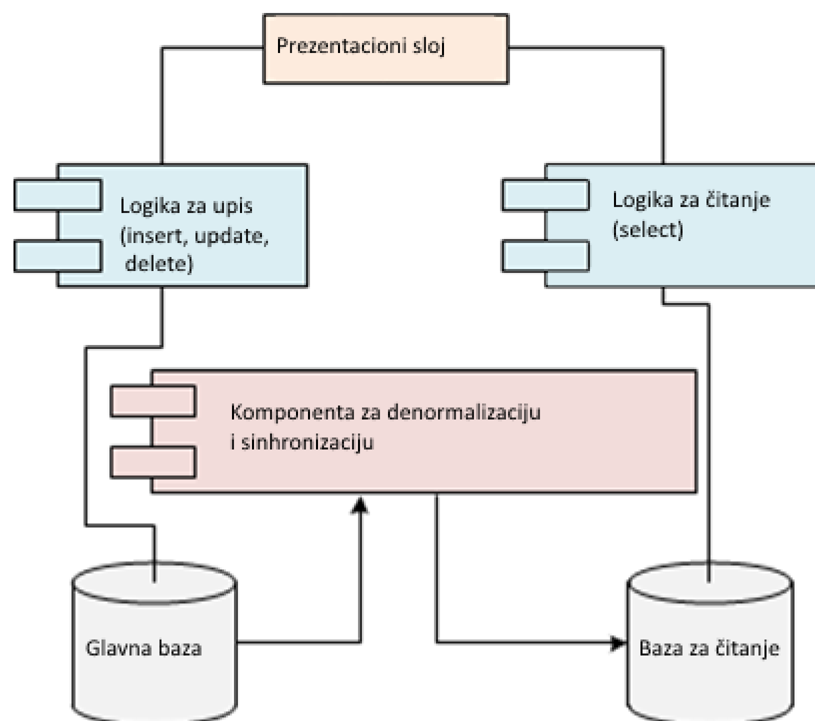
U ovom odeljku, biće opisan jedan česti problem koji se javlja kod informacionih sistema a koji rezultira povećanom obimu saobraćaja i većem protoku informacija. U informacionim sistemima, često postoji nekoliko upita koji se vrlo često izvršavaju a koji sa sobom prikazuju podatke o osnovnim entitetima u sistemu. Ovakvi podaci se često refrešuju kako bi ažurirali GUI.

Kod MIS sistema to su podaci o zakazivanjima pacijenata, podaci iz zdravstvenih kartona kao i podaci vezani za protekla lečenja. Za ovo postoje dva generalna rešenja:

- Definicija klasa sa redukovanim skupom atributa koje služe za prenos podataka ka korisničkom interfejsu, takozvane DTO klase (Data Transfer Object – DTO)
- Kreiranje posebne baze koja služi samo za čitanje podataka, a koja se sastoji od manjeg broja denormalizovanih tabela. U tom slučaju objekti kreirani na osnovu tih tabela se mogu smatrati za DTO objekte

Drugi pristup je pogodan kada se radi o podacima kojih ima relativno mnogo, a relativno retko se ažuriraju. U MIS ssitemima takvi su demografski podaci, podaci vezani za kartone, kao i rezultati pregleda. Dalje u poglavlju biće prikazana statistika iz Medis.NET MIS-a instaliranog u Domu zdravlja Niš. Ovakve baze, koje služe samo za čitanje podataka, se kasnije mogu upotrebiti i kao izvor za razne personal health Web portale.

Glavni koncept koji je ovde primenjen je CQRS – command and query responsibility segregation, odnosno podela odgovornosti između command (add, update, delete) i query (select) upita nad bazom. Osnovna zamisao je da se baza podeli na dva dela, gde će se nad jednom izvršavati primarno command, a nad drugom query operacije. Veza između dve baze je sinhronizaciona komponenta koja treba da osigura konsistenciju podataka. Ta sinhronizaciona komponenta je jedina koja ima privilegiju da upisuje podatke u bazu koja služi za query operacije. Ta baza se još naziva i baza za čitanje – read database (RD). Baza iz koje se podaci prebacuju u RD, naziva se još i glavna baza – main database (MD).



Slika 119 Primer realizacije CQRS pristupa

Za razliku od standardne višeslojne arhitekture, koja zahteva striktnu podelu između nivoa, CQRS uvodi podelu u okviru komponenti iz istog nivoa [99]. Slika 119 prikazuje jedan mogući pristup realizaciji CQRS koncepta. Prezentacioni sloj ostaje jedinstven, dok se nivo komandne logike deli na logiku za izvršenje command i logiku za izvršenje query upita. Sloj skladišta za podatke se sastoji od dve baze – GD i RD. Veza između baza je komponenta za denormalizaciju podataka i sinhronizaciju.

CQRS pristup je pogodan generalno za sisteme gde je broj query upita mnogo veći od broja command upita. Najpoznatiji primeri takvih sistema su online prodavnice, ali i informacioni sistemi imaju delove gde je primena ovog koncepta pogodna.

Zahtev za ovakvom dogradnjom je došao nakon više od dve godine aktivnog korišćenja sistema, i ovakva promena će uticati na mnogo nivoa i modula u sistemu. Zbog toga je odlučeno da se sistem razvija inkrementalno, i da za svaki korak u tom procesu se definiše novi mali projekat koji će se raditi po kombinovanoj metodologiji prezentovanoj u prethodnoj sekciji (Slika 118). Na kraju, svi read zahtevi treba da odu u read bazu, a svi command u postojeću. Do tada, MD će služiti i za command i za deo query zahteva.

U toku inicijalnog dizajniranja sistema, odlučeno je da ceo MIS treba da podrži rad i sa i bez RD. Odnosno rad u standardnom i CQRS modu. Pošto sve operacije pribavljanja podataka treba da prođu kroz isti interfejs, podržan je takozvani hot-swap mehanizam za modove rada. Ovo je vrlo pogodno kada se sistem ažurira, zato što je tada moguće otkačiti RD, instalirati unapređenu verziju, sinhronizovati podatke i ponovo omogućiti RD da prihvata upite.

Što se tiče vizuelnih komponenata, koje su deo prezentacionog nivoa MISa, one komuniciraju sa standardnim skupom interfejsa za svoje operacije, tako da je promena logike u nivou ispod, transparentna za GUI. GUI sloj uvek šalje isti strukturisani zahtev, jedino se razlikuje komponenta koja vraća odgovor.

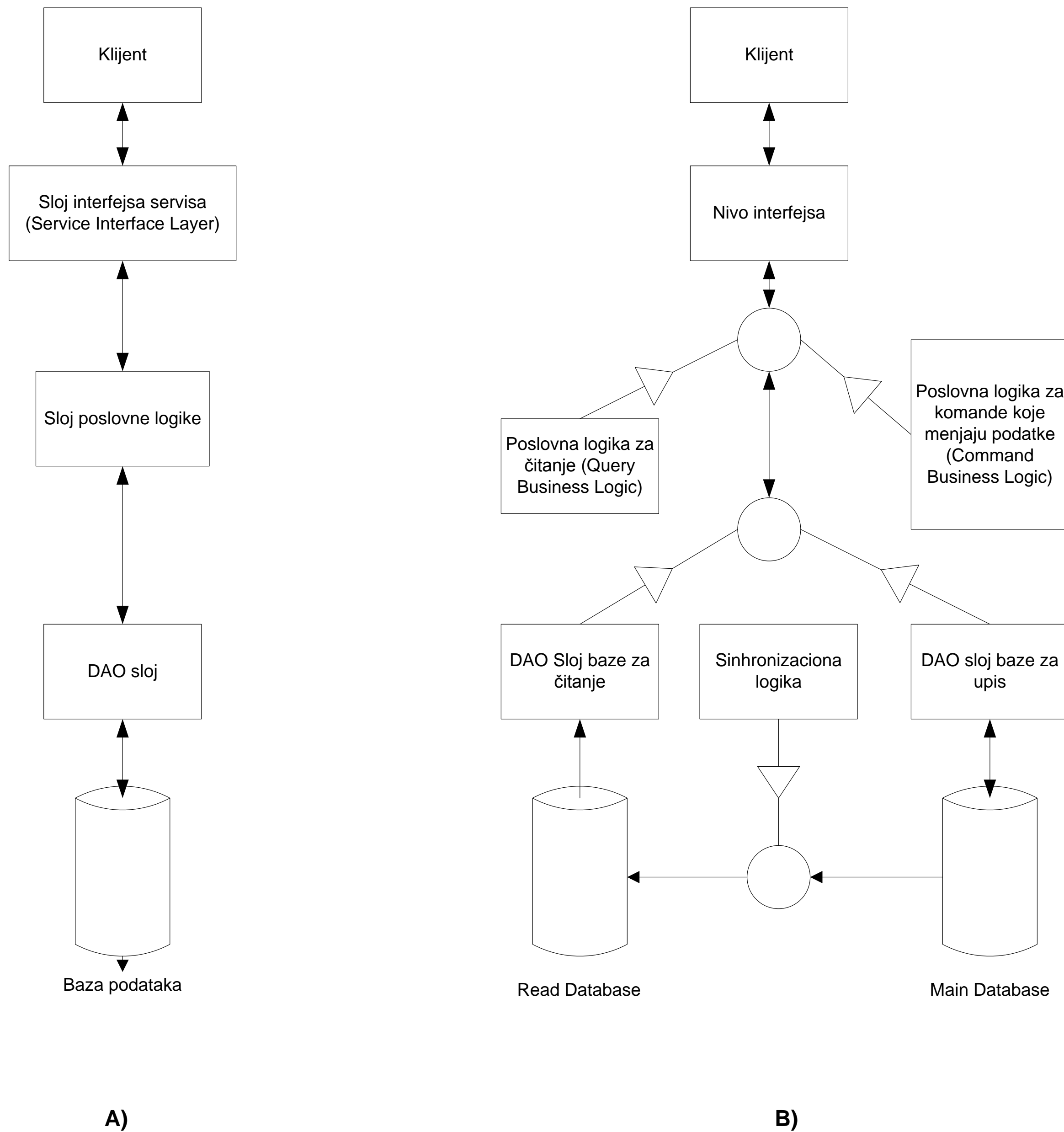
Kod ovakve vrste promene, neophodno je izvesti strukturnu promenu i u sloju poslovne logike i u sloju objektnog modela podataka (Data Access Object – DAO). Najpre potrebno je definisati interfejse ka dao klasama. Jedan skup interfejsa treba da pokrije command, a drugi query operacije. Sloj poslovne logike će prosleđivati zahteve za pribavljanje podataka samo query interfejsu. U inicijalnoj poslovnoj logici nalaze se implementacije za sve potrebne akcije i ovde je jedino potrebno definisati eksplicitno implementiranje novih interfejsa, kako bi se stare klase uklopile u novu arhitekturu.

U zavisnosti od konfiguracije sistema, biće aktivna implementacija interfejsa vezanog za glavnu, ili vezanog za read bazu. Na ovaj način, sistem ostaje potpuno operativan i kada read baza nije prisutna u sistemu. Ovo je važno podržati na univerzalni način pošto je odnos klijenata koji žele izdvojenu read bazu otprilike jednak onima kojima ona nije potrebna. Kada je jednom definišu, pomenuti interfejsi predstavljaju nove tačke proširenja u sistemu i omogućuju jednostavno menjanje poslovne logike i biblioteka na DAO nivou.

Logika za query operacije koja treba da obezbedi čitanje iz read baze treba da implementira samo skup interfejsa za čitanje podataka. S obzirom da postoji veliki broj operacija za čitanje iz baze, read skup interfejsa ima veliki broj članova. U toku razvoja, oni mogu biti implementirani jedan po jedan u okviru query poslovne logike i na taj način se read sistem može inkrementalno povećavati sve dok to ima smisla, i dok se ne dođe do optimalnog rešenja za podelu read operacija. Ako se pogleda sa tačke planiranja projekta, izdvojeni zadaci, koji bi se ovde definisali, bili bi:

- Definisanje skupa interfejsa komandne logike
- Ažuriranje standardnog sloja poslovne logike da implementira interfejse komandne logike
- Definisanje interfejsa za DAO nivo logike
- Ažuriranje DAO klasa tako da implementiraju DAO interfejse

- Proširenje Command logike operacijama za sinhronizaciju podataka sa read bazom
- Razvoj Query poslovne logike
- Razvoj Query DAO klasa



Slika 120 Odnos glavnih nivoa u SOA sistemu bez (A) i sa (B) primenom CQRS arhitekture

Nakon ažuriranja statičke strukture aplikacije, sledeći korak je definisanje read baze (RD). Za definisanje ove baze koristi se proširenje alata za modelovanje koji služi da mapira polja iz tabela osnovnog modela u tabele read baze. Takođe, generatorska klasa koja ide uz njega služi da izgeneriše kod koji će biti potreban za sinhronizaciju podataka i MD u RD. Za RD se može definisati i posebni model podataka, na osnovu koga se dalje mogu generisati različite softverske komponente. Bitno je

napomenuti i da se tabele u read bazi ne mogu definisati bilo kako, već u saglasnosti sa read interfejsom.

Pre nego počne sa definisanjem preslikavanja polja iz MD, korisnik odabere jedan od read interfejsa. Uz svaki od njih dolazi lista funkcija, i klasa čiji objekti se vraćaju kroz definisane funkcije. Svaka od tih klasa dolazi sa skupom definisanih atributa, i za te atribute treba definisati polja u RD kao i definiciju preslikavanja između polja iz MD u RD.

Definisana polja za RD se zatim koriste za generisanje tabela i objektnog modela za RD, a definisana preslikavanja za sinhronizaciono-denormalizacionu komponentu (Synchronization-Denormalization Component – SDC). Na ovaj način se olakšava inkrementalno razvijanje celog CQRS sistema, a kao ulaz alata za generisanje se mogu definisati nove generatorske klase koje će olakšati kreiranje koda za nove read interfejse.

SDC ima dva glavna zadatka – sinhronizaciju podataka i ažuriranje nakon akcije (update-upon-action, UUD). Oba procesa koriste preslikavanje između MD i RD kao skup pravila za kopiranje podataka. Sinhronizacija je proces kada SDC proverava razliku između MD i RD i ažurira RD. Sinhronizacija je bazirana na postojećoj platformi za replikaciju razvijanoj u okviru Medis.NET sistema [129]. Sinhronizacija ima tri režima rada: invalidate-insert, delete-insert i clean-up. Sinhronizacija je, zapravo mod rada koji služi za održavanje sistema i njegovo usaglašavanje sa glavnom bazom.

UUD je glavni mod rada za SDC. U ovom modu je SDC konstantno aktivan, prati promene podataka u MD, preuzima podatke, strukturiše ih i upisuje u RD. Praćenje promena podataka u glavnoj bazi se bazira na takozvanom database triggering mehanizmu i može se definisati za bilo koji tip entiteta iz glavne baze.

Bez obzira u kom modu SDC radi, on je na kraju, jedini deo sistema koji može da upisuje nove podatke u RD. Sve druge komponente mogu samo da čitaju iz nje.

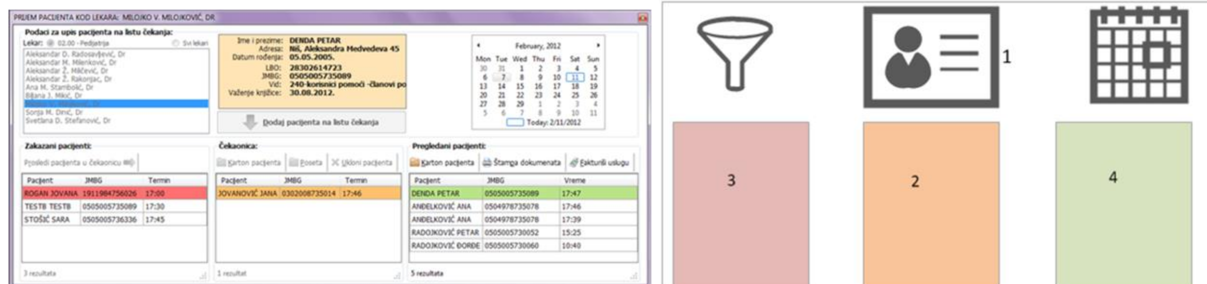
Glavna baza i Medis.NET MIS sistemu instaliranom u Domu zdravlja Niš sadrži 348 različitih tabela u koje se tokom godine prikupi oko 5.5 GB podataka. Tu je uključeno 2.25 miliona poseta lekaru sa ukupno 3.4 miliona različitih pregleda i oko 10 miliona generisanih dokumenata. Ovo je sasvim dovoljno podataka za analizu kako bi se utvrdilo gde su najpogodnija mesta za realizaciju CQRS pristupa.

Tabela 8 Statistika tabela koje se najčešće koriste u „select“ upitima

Tabela	Broj redova	Broj redova koji su dodati, promenjeni ili obrisani
Patient	431567	17335 (4%)
MedicalRecord	360234	10170 (2.8%)
ScheduledVisit	1885030	78434 (4.1%)
Insurance	460333	55635 (12%)

Tokom jedne posete, lekar opšte prakse prođe kroz manje više isti niz formi, gde se prikazuju podaci iz nekoliko osnovnih tabela. Tabela 8 prikazuje broj zapisa iz pomenutih osnovnih tabela, kao i broj i procenat redova koji su promenjeni, ažurirani ili obrisani u zadnjih godinu dana. Pošto se ovi podaci veoma retko menjaju, to ih čini idealnim kandidatima za prebacivanje u RD.

Forma koja je najčešće pred lekarom, kada ne radi preglede je forma za prijem pacijenata (Slika 121). Na levoj strani slike je njen aktuelni izgled, a na desnoj je njen prototip na kome su izdvojene karakteristične zone sa podacima. U zoni 1 se prikazuju demografski podaci odabranog pacijenta. Ovi podaci su rasuti u sedam različitih tabela i veoma se retko menjaju. Stopa promene je oko 4% godišnje. Ovo kvalifikuje demografske podatke da budu prebačeni u prvu tabelu koja će biti definisana u read bazi. Ta tabela je nazvana ReadPatientData i sadrži 13 kolona. Ove kolone dolaze iz 7 različitih tabela koje su povezane sa 9 relacija referencijalnih integriteta i sadrže ukupno 72 kolone.



Slika 121 Forma za prijem pacijenata (levo) i njen prototip sa izdvojenim funkoionalnim celinama (desno)

Slika 122 Forma koja prikazuje početnu stranu zdravstvenog kartona pacijenta – 1: demografski podaci, 2: medicinska upozorenja, 3: podaci vezani za osiguranje, 4: podaci o porodici

Deo poslovne logike koji čita ove podatke, prosleđuje ih kontroli za prikaz osnovnih podataka o pacijentu, koja se javlja kao deo više stotina formi i kontrola. Na ovaj način će se ubrzati učitavanje podataka na svakom od tih mesta i smanjiti ukupan saobraćaj sa bazom.

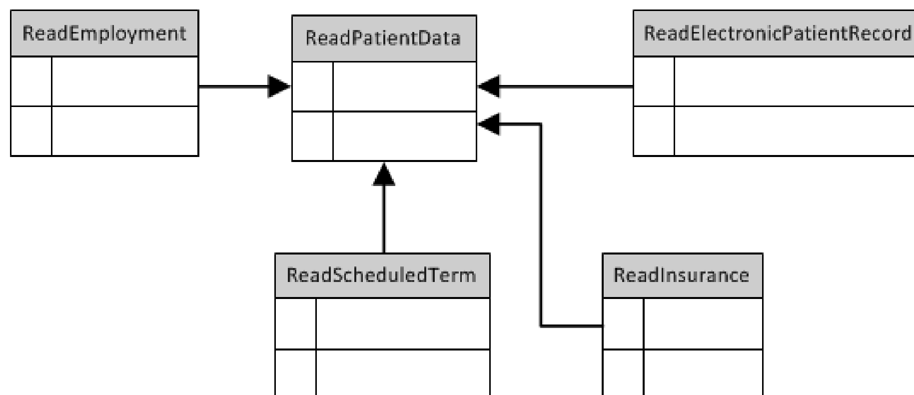
Sledeći kandidat za prebacivanje u RD je struktura koja opisuje podatke vezane za zakazivanje pregleda. U MD ova struktura ima 16 polja, ali 8 od njih su reference na druge tabele. Na formi za prijem to su podaci prikazani u zonama 2,3,4 (Slika 121). Za prikaz je potrebno pribaviti samo status,

vreme, i reference na lekara, odeljenje i pacijenta. Od ukupno 16 polja, pet je potrebno za prikaz na formi za prijem. Dakle, broj polja je smanjen sa 16 na 5 i nova tabela za RD je nazvana ReadScheduledTerm.

Sledeće mesto gde se može primeniti princip redukcije količine učitanih podataka je u formi koja prikazuje osnovu stranu zdravstvenog kartona. Slika 122 prikazuje pomenutu stranu. Na njoj su obeležene zone podataka koje dolaze iz različitih izvora a koje su praćene složenim strukturama u pozadini. Zone obeležene brojem 1 prikazuju podatke koji se čitaju iz ReadPatientData. Opšta medicinska upozorenja, broj kartona kao i specifični podaci dolaze iz zdravstvenog kartona i ti podaci se nalaze u zonama obeleženim brojem 2. Podaci o osiguranju su u zoni 3, a podaci o članovima porodice u zoni 4. Većina doktora, pre pregleda pogleda ovu formu, bez obzira što sa forme za prijem mogu odmah da startuju novu posetu (samo 7% intervjuisanih doktora to radi). Takođe, sa forme za prijem 29% lekara najpre otvara listu aktivnih lečenja, ali po navici, čak 64% lekara otvara karton najpre.

Medicinska upozorenja su deo osnovne tabele za zdravstveni karton. I ovo je deo gde je izbačeno najmanje podataka za preslikavanje u RD. Od inicijalnih 19 kolona, 17 je kopiranu u ReadElectronicPatientRecord tabelu.

Podaci o osiguranju dolaze iz 3 različite tabele sa ukupno 28 kolona. Njih zamenjuje denormalizovana tabela ReadInsurance sa ukupno 6 kolona. Slično je i sa podacima o porodici i zaposlenju. U glavnoj bazi, rasuti su u 6 tabela sa 43 kolone, dok tabela ReadEmployment ima svega njih 11.



Slika 123 Inicijalna šema RD

Na navedeni način kreirana je inicijalna prosta RD sa ukupno 5 tabela koje imaju 54 kolone i 4 referentne veze. Originalni deo iz MD se sastojao od 17 tabela sa ukupno 178 kolona. Ovo je donekle novi pristup, pošto se ne prebacuje cela baza odjednom u RD, već se RD postepeno povećava. Prikazana baza sadrži mali broj tabela, ali se odnosi na najčešće akcije i stoga doprinosi boljem odzivu sistema i smanjuje ukupan saobraćaj, što je prikazuju Tabela 9 i Tabela 10.

CQRS nije novi pristup, ali inkrementalni pristup njegovom uvođenju, podržan hot-swap mehanizmom je donekle novina. Standardni pristup je da se razvije celokupna read baza, i da se sve operacije prebace na nju. U predstavljenom rešenju akcenat je na tome da se u RD prebace najkorišćeniji podaci kako bi se ubrzao odziv sistema. Kreiranje RD koja je kopija osnovne baze je prilično zahtevan proces i sinhronizacija podataka bi onda bila nepotrebno velika.

Osnovu za CQRS je definisao još Bertrand Meyers u svojoj knjizi iz 1988 godine [135]. Sa ove distance, se može reći da je CQRS često korišćen kao arhitekturni obrazac i da je mnogo više podržan kroz tehničku nego kroz akademsku literaturu. Međutim, dugo godina nije obraćana preterana pažnja

na ovaj pristup. Tek nedavno ga je Microsoft uključio u svoju biblioteku obrazaca [136], i to tek nakon projekta koji je trebao da sagleda nove pristupe za razvoj skalabilnih aplikacija. Kako je rečeno u [137], “the project was focused on building highly scalable, highly available, and maintainable applications with the Command & Query Responsibility Segregation and the Event Sourcing patterns”.

Tabela 9 Efekat korišćenja read baze na formi za prijem pacijenata – 1: ReadPatientData, 2: ReadScheduledTerm

Korišćene tabele	Prosečna brzina odgovora (s)	Prosečna količina učitanih podataka iz baze (KB)
Nijedna	1.243	32.296
1	0.961	21.553
1, 2	0.774	11.658

Tabela 10 Efekat korišćenja read baze na formi za zdravstveni karton – 1: ReadPatientData, 2: ReadElectronicPatientRecord, 3: ReadInsurance, 4: ReadEmployment

Korišćene tabele	Prosečna brzina odgovora (s)	Prosečna količina učitanih podataka iz baze (KB)
Nijedna	1.927	12.858
1	1.675	11.026
1, 2	1.674	11.027
1,2,3	1.509	7.182
1,2,3,4	1.416	4.971

Kao glavni izvor o CQRS obrascu smatraju se članci koje su napisali Greg Young [134], Udi Dahan [138] i Martin Fowler [139], i oni se uz [136] i [137] smatraju glavnim izvorima za CQRS. Ovde su poslužili kao inicijalna tačka za istraživanje. Takođe, teza [141] koja se bavi generisanjem koda kroz CQRS obrazac i članak [142] koji pokazuje sve dobre i loše strane korišćenja CQRSa punog obima u MIS sistemima.

Glavna negativna strana upotrebe CQRS pristupa je redundansa podataka. Rezultat toga je da se u bazama čuva više podataka nego što je potrebno. Inkrementalnim pristupom u kreiranju CQRS podsistema, dobija se na kreiranju minimalne RD koja će najviše uticati na popravljanje performansi celog sistema.

Uvođenjem CQRSa, u delovima sistema gde je primenjen, vreme odziva se u proseku smanjilo za 40%, dok je količina prenetih podataka pala na trećinu. U isto vreme, količina uskladištenih podataka je povećana za manje od 3%.

7 Zaključak

Celokupni životni ciklus jednog softverskog proizvoda najviše zavisi od stava njegovih korisnika. Bez obzira na to koliko je genijalno smišljen, projektovan, realizovan i bez obzira na to koliko ima intuitivni interfejs (sa tačke gledišta programera i kognitivnih psihologa) softver neće biti prihvaćen, ako ne bude onakav kakvim ga njegovi korisnici žele.

Ovo je još izraženije kod informacionih sistema koji se razvijaju za određenog kupca po njegovoj specifikaciji. U ranijem periodu dobro su poznati slučajevi da su izuzetno dobri projekti, sa tehničke tačke gledišta, bili odbacivani zato što korisnik nije bio u potpunosti zadovoljan. U mnogim slučajevima, odsudni faktor je bio nedostatak komunikacije sa korisnicima.

Glavno istraživačko pitanje na koje ova disertacija treba da da odgovor je kako definisati pogodno okruženje za razvoj, implementaciju, održavanje i dogradnju informacionih sistema koje bi doprinelo povećanju verovatnoće prihvatanja sistema i skraćenju vremena razvoja sistema. Kao jedan od značajnih procesa na koje se troši mnogo vremena identifikovan je razvoj komponenti koje dele isti osnovni skup funkcionalnosti.

Rešenje ovog problema potraženo je u oblasti razvoja softvera baziranog na modelima podataka. Ideja je bila razviti odgovarajuće softverske alate za modelovanje i generisanje koda koji bi mogli biti široko primenjeni – za više razvojnih platformi i za više kategorija informacionih sistema. U isto vreme, prateći cilj je bio definisati skup smernica za korišćenje postojećih metodologija za razvoj softvera na najbolji mogući način u svakoj od faza životnog ciklusa informacionog sistema.

U poslednjih dvadeset godina većina softvera iz kategorije informacionih sistema razvijana je standardnim metodama razvoja, kod kojih, u najvećem broju slučajeva, postoji veliki vremenski period kada nema dovoljno kontakta sa klijentima. Nakon faze prikupljanja zahteva, kada se održavaju intenzivni sastanci i prave predlozi i prototipovi, kreće proces samog pisanja koda u kome je nivo komunikacije sa klijentima minimalan. Komunikacija se intenzivira ponovo kada dođe do testiranja, ali tada je već prošlo mnogo vremena i pitanje je šta budući korisnik zapravo očekuje. Sa druge strane, postoji veliki broj agilnih metoda razvoja koje inače podržavaju učestaliju komunikaciju sa klijentima. Međutim, njihova primena izaziva mnogo više stresa programerima [148] i vodi ka softveru čiji je kod u velikom broju slučajeva manje kvalitetan.

Deo rešenja koje se bavi primenom razvojnih metodologija ne odbacuje ni jedan ni drugi pristup – ni tradicionalni ni agilni. Naprotiv, ideja je uzeti najbolje od svih i iskombinovati ih kako bi zajedno dali što bolji rezultat. Metod predložen u ovoj disertaciji bazira se na podeli celokupnog projekta na manje segmente koji se tretiraju kao potprojekti. U isto vreme se definišu tačke proširenja i spoja za ceo sistem, kako bi se relativno nezavisni potprojekti kasnije jednostavnije uklopili. Za svaki od potprojekata se onda izabere metodologija koja je najpogodnija za njih. Izbor se bazira na tipu projekta, tako što se osnovni, takozvani „core“ projekti razvijaju standardnim, projekti koji razvijaju komponente okrenute korisnicima agilnim, a projekti koji se tretiraju kao relativno nezavisne aplikacije „lean“ pristupom.

Tako se, na primer, korisnički interfejs razvija primenom FDE pristupa, čiji je osnovni član generisanje softverskih komponenti bazirano na modelu podataka. Korisnički interfejs se razvija u okviru generisane administratorske aplikacije, koja se generiše na osnovu domenskih modela podataka i modela baze. Osnovni deo sistema se razvija primenom standardnih metodologija (model vodopada/RUP) i kasnije se integriše sa projektima koji su zaduženi za korisnički interfejs. Integracija

između GUI biblioteka i centralnog dela aplikacije moguća je kroz prethodno definisane tačke proširenja sistema, oko kojih se ujedno grade i domenski modeli.

Pomenute relativno nezavisne aplikacije, kao što su u okviru medicinskog informacionog sistema bili laboratorijski i radiološki informacioni sistemi, predstavljaju aplikacije srednjeg nivoa složenosti. Manje nego informacioni sistemi, a ipak znatno više nego skupovi generisanih komponenata. Za ovakav vid projekata korišćena je lean metodologija zajedno sa Kanban pristupom, i dala je izuzetno dobre rezultate. Uklapanje u ceo sistem je olakšano korišćenjem istih domenskih modela i u centralnom MISu i u pomenutim aplikacijama.

Što se tiče procesa implementacije i instalacije razvijenih informacionih sistema, inkrementalni pristup je dao najbolje rezultate. Iako inkrementalni metod inicijalno zahteva duži vremenski period prisustva tehničkog osoblja na mestu instalacije, u veoma kratkom roku pozitivni efekti poništavaju taj veći početni napor. Na ovaj način se veoma smanjuje broj poziva za podrškom, kao i zahteva za ponovnim objašnjavanjem funkcionalnosti, a i sami korisnici iskazuju veće početno zadovoljstvo sistemom.

Kao što je pomenuto, kod procesa implementacije i instalacije informacionih sistema korišćen je inkrementalni pristup, koji je za hitno dodavanje podrške za nove vrste dokumenata bio podržan alatima za modelovanje i generisanje koda. Pokazalo se da što je klijentska ustanova složenija i sa više korisnika i odeljenja, inkrementalni pristup daje bolje rezultate. U slučajevima kada je klijentska ustanova mala, sa manje od 10 korisnika, inkrementalni pristup gotovo da ne donosi nikakvu prednost.

Što se tiče razvoja korisničkog interfejsa, pogotovo kod MIS sistema, najbitnije je ne narušiti dnevnu rutinu budućih korisnika. A dodatna pogodnost koja je uočena je da oni veoma vole da vide interfejs koji izgleda kao njihovi dokumenti. Priča koja prati razvoj korisničkog interfejsa je i definisanje natpisa koji će se pojaviti na dijalozima. I ovde, ponovo, značajnu ulogu ima domenski model. S obzirom da je podrška za prevode ugrađena u osnovni deo modela podataka, alat za generisanje može jednostavno generisati resursne fajlove sa prevodima koje korisnici mogu sami da popune i na taj način se ne gubi vreme na ispravljanje prevoda od strane programera i eliminišu se moguće greške iz te kategorije. Prevodi se mogu definisati za natpise na svakom entitetu iz modela, tako da će sve generisane komponente imati uniformi način za prikaz labela i natpisa.

Nakon instalacije počinje održavanje sistema uz povremene zahteve za dogradnjom. S obzirom da se zahtevi za dogradnjom tretiraju kao posebni mali projekti, preporučeni pristup je primena neke od agilnih metodologija razvoja. U disertaciji je opisan pristup koji je dobijen adaptacijom lean metodologije. Lean je uzet kao osnova, dok su u okviru faze arhitekturnog rešenja i samog razvoja iskombinovane metode razvoja baziranog na funkcionalnostima i scrum metodologije. Originalna faza testiranja je ovde upotrebljena za integraciono testiranje, kao poseban bitan proces u okviru dogradnje postojećih sistema.

Mesta koja su najpogodnija da budu početne tačke oko kojih će se domenski modeli graditi se nazivaju tačke proširenja. Idealna situacija za primenu domenskih modela je kada je sistem projektovan tako da su tačke proširenja predviđene, međutim to nije uvek slučaj. Zbog toga je razvijen alat za specifičnu analizu strukture baze podataka, koji uz dodatno definisanje pravila može veoma brzo da identifikuje potencijalne tačke proširenja. Kada se u nekom postojećem sistemu takve tačke otkriju, onda je proces njegove dogradnje i adaptacije prilično olakšan. Onda u celu priču može da se uključe alati koji podržavaju razvoj baziran na modelu i da se ceo proces ubrza.

Ceo proces dogradnje se u predstavljenom pristupu tretira kao projekat dovoljne veličine da se na njega primeni neka od agilnih metodologija. Izbor je pao na kombinaciju nekoliko metodologija, tako

što je za osnovu ipak odabran lean-Kanban pristup, a onda su pojedine faze zamenjene drugačijim pristupima. Tako su na primer, svi pozitivni elementi SCRUM metodologije uključeni u samu fazu kodiranja, što je ostavilo mesta da proces testiranja dobije više na značaju.

Nedostatak ovakvog opšteg pristupa je prvenstveno u tome, da je neophodno imati u timu nekoga ko je u stanju da precizno raščlani početne zahteve i na osnovu njih definiše potprojekte, kao i tačke integracije i proširenja. U timovima koji se dugo bave razvojem sistema i koji se bave jednim određenim domenom ljudske delatnosti ovaj proces će biti mnogo lakše sproveden nego kod manje iskusnog tima koji treba da krene u razvoj potpuno novog sistema.

Iza bilo kog od navedenih pristupa razvoju stoji razvoj baziran na modelu podataka. Razvoj baziran na modelu nije ovde upotrebljen da bi bio opšte rešenje svih problema u programiranju i projektovanju, već element koji se može pridodati gotovo svakoj od metodologija razvoja softvera i učiniti je efikasnijom. Takođe, ovde je bitno napomenuti da je značajno pronaći pravu meru za primenu MDE pristupa. MDE iako nije opšte rešenje, on je mnogo više od pukog generatora formi koji ima ista polja kao i tabela koju generisana forma treba da edituje.

Prvi korak u primeni MDE pristupa je kreiranje (ili izbor iz skupa postojećih) meta modela podataka. Meta modeli podataka se najčešće kreiraju na jedinstvenim opštim principa, ali ipak moraju da podrže specifičnosti oblasti za koju se generišu. Tako na primer, meta modeli za PIMS i MIS sisteme ne mogu da budu isti. U zavisnosti od primene, neki meta modeli mogu da budu kreirani tako da su više okrenuti modelovanju kroz proširenje strukture a neki više okrenuti razvoju akcija. U okviru disertacije dat je opis meta modela specijalno razvijenog na osnovu zahteva koji su došli iz zdravstva u Republici Srbiji.

Generisanje struktura podataka, kao rezultat pretvaranja domenskog znanja u znanje potrebno za razvoj informacionog sistema je mesto gde meta modeli podataka i alati za modelovanje imaju glavnu ulogu. Ono gde treba paziti, i ne otići predaleko, je proces definisanja akcija u modelu. Treba se zadržati na jednostavnim akcijama koje može da definiše neko i bez velikog programerskog znanja, dok je mesto složenim procedurama, još uvek, u razvojnim okruženjima odgovarajućih programskih jezika. U disertaciji je prikazan i alat za modelovanje koji je dizajniran tako da može da razvija modele ne samo na osnovu različitih meta modela, već i da koristi različite biblioteke vizuelnih komponenti kako bi se unapredio njegov izgled. Njegovo funkcionisanje je prikazano na primeru razvijenog meta modela za zdravstvo, kao i meta modela koji dolazi kroz B2MML, a koristi se u PIMS sistemima.

Sam alat za modelovanje je komponenta koja služi da kreira model, ali kod sistema koji već rade, a koje treba dograđivati neophodno je dodati funkcionalnost koja će omogućiti izgradnju modela na osnovu postojećih struktura podataka. U tu svrhu alat za modelovanje je proširen bibliotekama za analizu strukture aplikacije, identifikovanje tačaka proširenja na osnovu definisanog skupa pravila i reverzno ažuriranje modela na osnovu promena u analiziranim strukturama podataka. Ovde je osim same analize baze moguće definisati i dodatne skupove pravila za identifikovanje tačaka proširenja. Kako je skup proširiv, korisnici mogu definisati nova pravila što doprinosi opštosti rešenja.

Sledeći značajan segment MDE baziranog razvojnog okvira su alati za generisanje softverskih komponenti. U okviru disertacije prikazani su alati za generisanje aplikacije za administriranje baze i alat za generisanje softverskih komponenti baziran na domenskim modelima podataka. Aplikacija za administriranje baze je pomoćna aplikacija i služi kao osnova za testiranje različitih formi korisničkog interfejsa. Takođe se može koristiti kod procesa dogradnje za bolje razumevanje veza između podataka u bazi.

Alat za generisanje komponenti na osnovu modela radi na principu zamene specijalnih komentara u okviru šablonskih komponenti generisanim kodom. Na ovaj način generisane komponente su prilično nezavisne od tehnologije pošto generatorske klase mogu generisati kod u bilo kom programskom jeziku. Dalje, moguće je definisati raznorodne šablonske komponente i klase za generisanje koda i na taj način uticati na više različitih segmentata sistema. Na kraju, još jedan element MDE skupa aplikacija je sistem za interpretaciju modela i generisanje izveštaja u toku izvršenja. Ove komponente koje se bave interpretacijom modela su za razliku od elemenata standardnog MIS-a Web komponente.

Nedostaci MDE pristupa su poznati i iako ih je lako identifikovati, nije ih uvek jednostavno zaobići ili premostiti [37]. Kada se kreiraju alati za modelovanje njihov glavni cilj je brzo generisanje što većeg broja artefakata kako bi se što pre došlo do vidljivog rezultata. Glavni nedostatak alata za modelovanje je to što se kroz njih ne mogu uočiti potencijalno beskorisni artefakti. Ovde se može identifikovati i pravac daljeg rada na temi koju je ova disertacija započela, a to je kreiranje odgovarajućeg pristupa za identifikovanje artefakata koji se ne koriste, kao i za procenu njihovog potencijalnog životnog veka. Zajedno sa tim još jedan pravac budućeg rada će biti i razvoj alata za modelovanje sa unapređenim korisničkim interfejsom.

Predstavljeni alat za modelovanje, ima najznačajniji konceptualni nedostatak u tome što nije platformski nezavistan, iako generisani modeli jesu. Alat je razvijen u :NET tehnologiji i kao takav efikasno može da radi samo na Microsoft Windows platformama. S obzirom na oblasti u kojima se primenjuje, za sada, ovo ne predstavlja značajnu prepreku njegovom korišćenju, ali u perspektivi bi mogao da bude zamenjen odgovarajućom platformski nezavisnom verzijom. Takođe, alat u sadašnjoj formi ne podržava kolaborativni rad i to je još jedan zadatak za budući rad.

Sledeći nedostatak MDE pristupa je pristup problemu ponovnog generisanja komponenti nakon promena u modelu. U trenutnom stanju sistema ovo se rešava u zavisnosti od obima promena tako što se ili ponovo generiše komponenta ili se dodaju ili uklone odgovarajuća polja. Ovde je glavni problem kako izbeći gubitak promena napravljenih na komponenti nakon prethodnog generisanja, i njegovo rešavanje je još jedan od budućih zadataka u razvoju okruženja za razvoj baziranog na modelu podataka.

Kada su razvijani alati za modelovanje i generisanje koda težilo se, iz praktičnih razloga, da se izbegne korišćenje meta jezika i specifičnih jezika za modelovanje. Iz tog razloga, za definisanje različitih konfiguracija iskorišćena je NHibernate sintaksa, dok se kod akcija modeluje u nekom od jezika koji dolaze kroz .NET – C++, Visual Basic ili C#. Na ovaj način je žrtvovana opštost rešenja zarad dobijanja alata koji će veći broj programera moći da koristi.

Kroz razvoj različitih alata za podršku MDE pristupu, dat je doprinos na polju razvoja odgovarajućih alata za modelovanje i generisanje koda. Kroz proces verifikacije i testiranja šablonskih komponenti je uspešno izbegnuta zamka korišćenja netestiranih elemenata u modelu i vraćanje na fazu modelovanja je smanjeno u značajnoj meri.

Razvoj baziran na domenskim modelima podataka prožima sve procese i sve životne faze informacionog sistema. Kroz predstavljeni skup softverskih alata i procedura MDE pristup je iskorišćen za definisanje razvojnog okvira koji može u mnogome da unapredi svaku fazu života informacionih sistema. Kroz skupove smernica i opise procesa razvoja, implementacije i dogradnje informacionih sistema, definisano je mesto i širina uticaja MDE pristupu na način da daju najveći doprinos projektima čiji rezultat treba da budu efikasni i upotrebljivi informacioni sistemi

8 Literatura

[1] Valacich, Joseph S., Christoph Schneider, and Leonard M. Jessup. *Information systems today: managing in the digital world*. Pearson, 2014.
[2] Laudon, Kenneth C., and Jane Price Laudon. *Essentials of management information systems*. Upper Saddle River: Pearson, 2011.
[3] Stair, Ralph, and George Reynolds. *Principles of information systems*. Cengage Learning, 2011.
[4] Krist, Alex H., and Steven H. Woolf. "A vision for patient-centered health information systems." *Jama* 305.3 (2011): 300-301.
[5] Irani, Zahir, and Peter Love, eds. *Evaluating Information Systems*. Routledge, 2013.
[6] Davenport, Thomas H. *Process innovation: reengineering work through information technology*. Harvard Business Press, 2013.
[7] Jo, Sunhwan, et al. "CHARMM-GUI: a web-based graphical user interface for CHARMM." *Journal of computational chemistry* 29.11 (2008): 1859-1865.
[8] Molina-Moreno, Pedro Juan, et al. "Method and apparatus for automatic generation of information system user interfaces." U.S. Patent No. 7,941,438. 10 May 2011.
[9] Burke, Rory. *Project management: planning and control techniques*. 2013.
[10] Atalag, Koray, et al. "Evaluation of software maintainability with< i> open</i> EHR–a comparison of architectures." *International journal of medical informatics* (2014).
[11] Leau, Yu Beng, et al. "Software Development Life Cycle AGILE vs Traditional Approaches." *International Conference on Information and Network Technology*. Vol. 37. No. 1. 2012.
[12] Doherty, Michael J. "Examining Project Manager Insights of Agile and Traditional Success Factors for Information Technology Projects: A Q-Methodology Study." *PhD, Marian University, Doctoral Dissertation* (2012).
[13] Dingsøyr, Torgeir, et al. "A decade of agile methodologies: Towards explaining agile software development." *Journal of Systems and Software* 85.6 (2012): 1213-1221.
[14] Black, Sue, et al. "Formal versus agile: Survival of the fittest." *Computer* 42.9 (2009): 37-45.
[15] Iivari, Juhani, and Netta Iivari. "The relationship between organizational culture and the deployment of agile methods." *Information and Software Technology* 53.5 (2011): 509-520.
[16] Boehm, Barry W. "A spiral model of software development and enhancement." *Computer* 21.5 (1988): 61-72.
[17] Kruchten, Philippe. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.
[18] V-model, opis, http://www.softwaretestingclass.com/v-model/.
[19] Schwaber, Ken, and Mike Beedle. "Agilè Software Development with Scrum." (2002).
[20] Hunt, John. "Feature-Driven Development." *Agile Software Construction* (2006): 161-182.
[21] Atkinson, Colin, and Thomas Kuhne. "Model-driven development: a metamodeling foundation." *Software, IEEE* 20.5 (2003): 36-41.
[22] Schmidt, Douglas C. "Model-driven engineering." *COMPUTER-IEEE COMPUTER SOCIETY-* 39.2 (2006): 25.
[23] Karlstrom, Daniel, and Per Runeson. "Combining agile methods with stage-gate project management." *IEEE software* 22.3 (2005): 43-49.

[24] Lindvall, Mikael, et al. "Agile software development in large organizations." *Computer* 37.12 (2004): 26-34.

[25] Korkala, Mikko, Minna Pikkarainen, and Kieran Conboy. "Combining agile and traditional: Customer communication in distributed environment." *Agility Across Time and Space*. Springer Berlin Heidelberg, 2010. 201-216.

[26] Virani, Shamsnaz, and Lauren Stolzar. "A Hybrid System Engineering Approach for Engineered Resilient Systems: Combining Traditional and Agile Techniques to Support Future System Growth." *Procedia Computer Science* 28 (2014): 363-369.

[27] Garcia-Smith, Dianna, and Judith A. Effken. "Development and initial evaluation of the clinical information systems success model (CISSM)." *International journal of medical informatics* 82.6 (2013): 539-552.

[28] Petter, Stacie, William DeLone, and Ephraim R. McLean. "Information systems success: The quest for the independent variables." *Journal of Management Information Systems* 29.4 (2013): 7-62.

[29] Khalifa, Mohamed. "Barriers to Health Information Systems and Electronic Medical Records Implementation. A Field Study of Saudi Arabian Hospitals." *Procedia Computer Science* 21 (2013): 335-342.

[30] Torchiano, Marco, et al. "Relevance, benefits, and problems of software modelling and model driven techniques—A survey in the Italian industry." *Journal of Systems and Software* 86.8 (2013): 2110-2126.

[31] Loniewski, Grzegorz, Emilio Insfran, and Silvia Abrahão. "A systematic review of the use of requirements engineering techniques in model-driven development." *Model driven engineering languages and systems*. Springer Berlin Heidelberg, 2010. 213-227.

[32] Cerovsek, Tomo. "A review and outlook for a 'Building Information Model'(BIM): A multi-standpoint framework for technological development." *Advanced engineering informatics* 25.2 (2011): 224-244.

[33] Brambilla, Marco, Jordi Cabot, and Manuel Wimmer. "Model-driven software engineering in practice." *Synthesis Lectures on Software Engineering* 1.1 (2012): 1-182.

[34] Völter, Markus, et al. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[35] Blanco, Carlos, et al. "Showing the Benefits of Applying a Model Driven Architecture for Developing Secure OLAP Applications." *J. UCS* 20.2 (2014): 79-106.

[36] Sneed, Harry M. "The drawbacks of model-driven software evolution." *IEEE CSMR* 7 (2007).

[37] http://www.infoq.com/articles/8-reasons-why-MDE-fails.

[38] Almonaies, A., James R. Cordy, and Thomas R. Dean. "Legacy system evolution towards service-oriented architecture." *International Workshop on SOA Migration and Evolution*. 2010.

[39] Selic, Bran. "The pragmatics of model-driven development." *IEEE software* 20.5 (2003): 19-25.

[40] Ramón, Óscar Sánchez, Jesús Sánchez Cuadrado, and Jesús García Molina. "Model-driven reverse engineering of legacy graphical user interfaces." *Automated Software Engineering* 21.2 (2014): 147-186.

[41] Mohagheghi, Parastoo, et al. "An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases." *Empirical Software Engineering* 18.1 (2013): 89-116.

[42] Sánchez-Cuadrado, Jesús, Juan De Lara, and Esther Guerra. *Bottom-up meta-modelling: An interactive approach*. Springer Berlin Heidelberg, 2012.

[43] Jézéquel, Jean-Marc, Olivier Barais, and Franck Fleurey. "Model driven language engineering with kermeta." *Generative and Transformational Techniques in Software Engineering III*. Springer Berlin Heidelberg, 2011. 201-221.
[44] Cunha, Jácome, et al. "MDSheet: A framework for model-driven spreadsheet engineering." *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 2012.
[45] ADL WorkBench, http://www.openehr.org/downloads/ADLworkbench/home
[46] LinkEHR normalization tool, http://pangea.upv.es/linkehr/.
[47] http://www.isa-95.com/.
[48] http://www.openehr.org/.
[49] Rajković P, Janković D, and Milenković A. "Developing and deploying medical information systems for Serbian public healthcare: Challenges, lessons learned and guidelines." *Computer Science and Information Systems* 10.3 (2013): 1429-1454.
[50] Schach, Stephen R. *Object-oriented and classical software engineering*. Vol. 6. New York: McGraw-Hill, 2002.
[51] Batini, Carlo, Stefano Ceri, and S. Navathe. *Entity Relationship Approach*. Elsevier Science Publishers BV (North Holland), 1989.
[52] Fichman, Robert G., and Chris F. Kemerer. "Adoption of software engineering process innovations: The case of object-orientation." *Sloan management review* 34.2 (2012).
[53] Dwarampudi, Venkatreddy, et al. "Comparative study of the Pros and Cons of Programming languages Java, Scala, C++, Haskell, VB. NET, AspectJ, Perl, Ruby, PHP & Scheme-a Team 11 COMP6411-S10 Term Report." *arXiv preprint arXiv:1008.3431* (2010).
[54] Linberg, Kurt R. "Software developer perceptions about software project failure: a case study." *Journal of Systems and Software* 49.2 (1999): 177-192.
[55] Agarwal, Nitin, and Urvashi Rathod. "Defining 'success' for software projects: An exploratory revelation." *International journal of project management* 24.4 (2006): 358-370.
[56] McLeod, Laurie, and Stephen G. MacDonell. "Factors that affect software systems development project outcomes: A survey of research." *ACM Computing Surveys (CSUR)* 43.4 (2011): 24.
[57] Poernomo, Iman, and Jeffrey Terrell. "Correct-by-construction model transformations from partially ordered specifications in coq." *Formal Methods and Software Engineering*. Springer Berlin Heidelberg, 2010. 56-73.
[58] Cerovsek, Tomo. "A review and outlook for a Building Information Model(BIM): A multi-standpoint framework for technological development." *Advanced engineering informatics* 25.2 (2011): 224-244.
[59] OpenEHR meta model, http://www.openehr.org/wiki/display/spec/openEHR+1.0.2+UML+resources.
[60] Composite Design Pattern, http://sourcemaking.com/design_patterns/composite.
[61] OpenEHR case study, http://www.talkstandards.com/openehr-case-study/.
[62] EN13606 standard, http://www.en13606.org/.
[63] HL7 standard, http://www.hl7.org/.
[64] OpenEHR Clinical Knowledge Manager, http://www.openehr.org/ckm/
[65] B2MML standard, http://www.mesa.org/en/B2MML.asp
[66] ABB Enterprise Connectivity Solution System Overview, http://www05.abb.com/global/scot/scot313.nsf/veritydisplay/bbb952d67fd5187ec12572c200734224/$file/3bus094415_a_en_ecs_enterprise_connectivity_solution.pdf.

[67] Dumas, Marlon, and Arthur HM Ter Hofstede. "UML activity diagrams as a workflow specification language." *≪UML≫ 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools*. Springer Berlin Heidelberg, 2001. 76-90.
[68] Nikiforova, O., Nikulsins, V., Sukovskis, U.: Integration of MDA Framework into the Model of Traditional Software Development.In: Frontiers in Artificial Intelligence and Applications, Databases and Information Systems V, vol. 187, pp. 229–239. IOS Press, Amsterdam, 2009.
[69] O. Foundation, "The Interoperability Standard for Industrial Automation," [Online]. Available: http://www.opcfoundation.org/.
[70] E. Evans, Domain-driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley Professional, 2004.
[71] J. Vanderdonckt, J. Coutaz, G. Calvary and A. Stanciulescu, "Multimodality for Plastic User Interfaces: Models, Methods, and Principles," *Lecture Notes in Electrical Engineering, Springer-Verlag,* p. 61–84, 2007.
[72] A. Lorenz, "Architectural patterns for applications with external user interface elements," *Pervasive and Mobile Computing, Elsevier,* vol. 9, p. 269–280, 2013.
[73] D. Benyon and D. Murray, "Adaptive systems: from intelligent tutoring to autonomous agents," *Knowledge-Based Systems, Elsevier,* vol. 6, no. 4, p. 197–219, 1993.
[74] D. Roscher, G. Lehmann, V. Schwartze, M. Blumendorf and S. Albayrak, "Dynamic distribution and layouting of model-based user interfaces in smart environments," *Studies in Computational Intelligence,* vol. 340, pp. 171-197, 2011.
[75] M. Blumendorf, G. Lehmann and S. Albayrak, "Bridging models and systems at runtime to build adaptive user interfaces," in *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, New York, 2010.
[76] A. G. Frey, G. Calvary and S. Dupuy-Chessa, "Xplain: an editor for building self-explanatory user interfaces by model-driven engineering," in *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, New York, 2010.
[77] J. Galloway, P. Haack, B. Wilson, K. S. Allen and S. Hanselman, Professional ASP.NET MVC 4, Wrox, 2012.
[78] "ASP.NET MVC 3 Razor," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/vs2010trainingcourse_aspnetmvc3razor.aspx.
[79] M. Abbott, "Dynamic templating Razor engine," RazorEngine, 2013. [Online]. Available: https://github.com/Antaris/RazorEngine.
[80] S. Pietschmann, M. Voigt, A. Rümpel and K. Meißner, "CRUISe: Composition of Rich User Interface Services," *Web Engineering, Lecture Notes in Computer Science,* vol. 5648, pp. 473-476, 2009.
[81] E. G. Nilsson, J. Floch, S. Hallsteinsen and E. Stav, "Model-based user interface adaptation," *Computers & Graphics, Elsevier,* vol. 30, p. 692–701, 2006.
[82] Microsoft, "XAML Overview (WPF)," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms752059.aspx.
[83] L. A. MacVittie, XAML in a Nutshell, O'Reilly, 2006.
[84] J. Team, "JQuery Code Organization Concepts," 2013. [Online]. Available: http://learn.jquery.com/code-organization/concepts/.
[85] J. Foundation, "JQuery Widgets," [Online]. Available: http://jqueryui.com/widget/. [Accessed 2013].
[86] Microsoft, "jQuery UI Widgets," [Online]. Available: http://msdn.microsoft.com/en-us/library/hh404085.aspx.
[87] Onion Architecture, http://jeffreypalermo.com/blog/the-onion-architecture-part-1/
[88] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2003.
[89] Microsoft, "Service Oriented Architecture," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb833022.aspx.
[90] M. Fowler, "Inversion of control containers and the dependency injection patternl," 2004. [Online]. Available: www.martinfowler.com/articles/injection.html.

[91] MSDN, "The Repository Pattern," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/ff649690.aspx. [Accessed 2013].
[92] J. Colao, "Facebook's HTML5 Dilemma, Explained," 2012. [Online]. Available: http://www.forbes.com/sites/jjcolao/2012/09/19/facebooks-html5-dilemma-explained/.
[93] H. Shen, Z. Yang and C. Sun, "Collaborative Web Computing: From Desktops to Webtops," IEEE Distributed Systems Online, vol. 8, no. 4, p. 3, 2007.
[94] Microsoft, "ASP.NET Web Server Controls," [Online]. Available: http://msdn.microsoft.com/en-us/library/bb386416(v=vs.100).aspx. [Accessed 2013].
[95] D. Esposito, "Comparing Web Forms And ASP.NET MVC," 2009. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/dd942833.aspx. [Accessed 2013].
[96] J.Maras, M.Stula and J.Carlson, "Extracting client-side web user interface controls," Proceedings of the ICWE, Springer Verlag, p. 502–505, 2010.
[97] W3C, "Custom data attributes, HTML5.1 specification," 2013. [Online]. Available: http://www.w3.org/TR/html51/dom.html#custom-data-attribute.
[98] Jacobson, Ivar, et al. *The unified software development process*. Vol. 1. Reading: Addison-Wesley, 1999.
[99] P. Rajkovic, D. Jankovic, T. Stankovic and V. Tosic, "Software Tools for Rapid Development and Customization of Medical Information Systems," 12th IEEE International Conference on e-Health Networking, Applications and Services (HealthCom 2010), Lyon, France, Vol. 1, Nr. 1, pp. 119-126, ISSN: ISBN 978-1-4244-6375-6/10; IEEE Publishing, 2010.
[100] L. Ren, FengTian, X. Zhang and LinZhang, "DaisyViz: A model-based user interface toolkit for interactive information visualization systems," Journal of Visual Languages and Computing 21, Elsevier, p. 209–229, 2010.
[101] P. Barclay, T. Griffiths, J. McKirdy, J. Kennedy, R. Cooper, N. Paton and P. Gray, "Teallach — a flexible user-interface development environment for object database applications," Journal of Visual Languages & Computing, Elsevier, vol. 14, no. 1, p. 47–77, 2003.
[102] G. Meixner , F. Paterno and J. Vanderdonckt , "Past, Present, and Future of Model-Based User Interface Development," i-com, vol. 3, 2011.
[103] T. Schlegel, "An Interactive Process Meta Model for Runtime User Interface Generation and Adaptation," in Fifth International Workshop on Model Driven Development of Advanced User Interfaces , Atlanta, 2010.
[104] P. Sukaviriya, "From user interface design to the support of intelligent and adaptive interfaces: an overhaul of user interface software infrastracture," Knowledge-Based Systems, Elsevier, vol. 6, no. 4, p. 220–229, 1993.
[105] T. Höllerer, S. Feiner, D. Hallaway, B. Bell, M. Lanzagorta, D. Brown, S. Julier, Y. Baillot and L. Rosenblum, "User interface management techniques for collaborative mobile augmented reality," Computers & Graphics, Elsevier, vol. 25, no. 5, p. 799–810, 2001.
[106] S. Mohorovicic, "Implementing responsive web design for enhanced web presence," in 36th International Convention on Information & Communication Technology Electronics & Microelectronics (MIPRO), Opatija, 2013.
[107] Spring Framework, http://projects.spring.io/spring-framework/.
[108] Simonazzi, A.: Care regimes and national employment models, Cambridge Journal of Economics, Vol. 33, 211-232. (2009).
[109] Health care system and spending in Serbia, Report, on current status www.healthsystems2020.org/.../2285_file_NHA_health_expenditures.doc.
[110] Gajic-Stevanovic, M.: Healthcare System and Spending in Serbia, report for 2003-2006, taken from http://www.healthsystems2020.org/ content/impact/detail/2285/.
[111] Jovanović, V., Milošević, B., Potter, B.: Improving quality of primary healthcare in Serbia, Int'l Journal on Total QualityManagement & Excellence, Belgrade, Serbia, Vol. 35. No.1 - 2, 347-352. 2007.
[112] Bensing, J.: Bridging the gap. The separate worlds of evidence-based medicine and patient-centered medicine. Patient Education and Counseling 39 17–25. 2000.
[113] Siemens Soarian website, http://www.siemenssoarian.com/.

[114] OpenERP website, http://www.openerp.com/.
[115] Heliant website, http://www.heliant.rs/health.
[116] Tzu-Hsiang Yang, Yeali S Sun, Feipei Lai: A scalable healthcare information system based on a service-oriented architecture, Journal of Medical Systems (impact factor: 1.13). 06/2011; 35(3):391-407. DOI:10.1007/s10916-009-9375-5.
[117] Amouh, T., Gemo, M., Macq, B., Vanderdonckt, J., Gariani, A.W.E., Reynaert, M.S., Stamatakis, L., Thys, F.: Versatile clinical information system design for emergency departments, Information Technology in Biomedicine, IEEE Transactions on, Volume: 9 , Issue: 2 , 174 – 183. (2005).
[118] Teixeira, L., Ferreira, C., Santos, B.S., Martins, N.: Modeling a Web-based Information System for Managing Clinical Information in Hemophilia Care, Engineering in Medicine and Biology Society, EMBS '06. 28th Annual International Conference, 2610 – 2613. (2006).
[119] Rajković, P., Janković, D., Tošić, V.: A Software Solution for Ambulatory Health Facilities in the Republic of Serbia, HEALTHCOM 2009 - 11th International Conference on e-Health Networking, Application and Services, Sydney, Australia ISBN: 978-1-4244-5013-8, 161-168, (2009).
[120] Rajkovic, P., Jankovic, D., Stankovic, T.: An e-Health Solution for Ambulantory Facilities, ITAB 2009, Vol. 1, Nr. 1, Fr. 1.5.1 1-4, Larnaca, Cyprus. (2009).
[121] Hwang, S.C., Lee, M. H.: A web-based telePACS using an asymmetric satellite system, IEEE Trans. Inf. Technol. Biomed., vol. 4, no. 3, 212–215. (2000).
[122] Smart client architecture, http://msdn.microsoft.com/en-us/library/ff647359.aspx.
[123] ZipSoft website, http://www.zipsoft.rs.
[124] GE Centricity website on GE Healthcare, http://www3.gehealthcare.com/en.
[125] Allscripts, http://www.allscripts.com/en/solutions/ambulatory-solutions/ehr.html.
[126] Sung-Huai Hsieh, Sheau-Ling Hsieh, Po-Hsun Cheng, Feipei Lai: Telemedicine and e-Health, April 2012, 18(3): 205-212. (2012).
[127] Zhang Xiao-guang; Li Jing-song; Zhou Tian-shu; Yang Yi-bing; Chen Yun-qi; Xue Wan-guo; Zhao Jun-ping; Design and implementation of Interoperable Medical Information System based on SOA, IT in Medicine & Education, 2009, Volume: 1 Digital Object Identifier: 10.1109/ITIME.2009.5236236,1074 – 1078. (2009).
[128] Tsiknakis, M., Katehakis, D. G., Orphanoudakis, S. C.: An open, component-based information infrastructure for integrated health information networks. Int. J. Med. Inf. [Online] 68(1–3), 3–26. (2002).
[129] Stankovic, T., Jankovic, D., Pesic, S.: Public Health Care Distributed DBMS with Resolving Database Replication Conflicts in the Health Care Information System Project Early Phase, 9th TELSIKS, Vol. 2, 487-490.(2009).
[130] Stankovic, T., Pesic, S., Jankovic, D.: Platform Independent Database Replication Solution Applied to Medical Information System, LNCS, Springer-Verlag, Berlin Heidelberg, Vol. 6295, 587-590. (2010).
[131] Stanković, T., Rajković, P., Milenković, A., Janković, D.: User Interface in Medical Information Systems – Common Problems and Sustainable Solutions, Electronics, vol. 14, No. 2, 59-64. (2010).
[132] Stanković, T., Rajković, P., Milenković, A., Janković, D., From optional talk to medical information system's user interface, INFOTEH, Jahorina, Vol. 9, E1-1, 894-898. (2010).
[133] Rita Kukafka, et al,: Redesigning electronic health record systems to support public health, Journal of Biomedical Informatics Volume 40, Issue 4, 398-409. (2007).
[134] Jankovic D., Stankovic T., Rajkovic P.: Comprehensive data reporting approach in health care information systems, International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC), Valencia, Spain, 20-23 January, 456-460. (2010) .
[135] Greg Young, CQRS Documents by Greg Young, http://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf.
[136] Bertrand Meyer, Object-oriented Software Construction, Prentice Hall 1988, ISBN 0-13-629049-3.

[137] Dominic Betts, Julián Domínguez, Grigori Melnik, Fernando Simonazzi, Mani Subramanian, Foreword by Greg Young: Exploring CQRS and Event Sourcing, http://www.microsoft.com/en-us/download/details.aspx?id=34774.
[138] CQRS Journey, http://msdn.microsoft.com/en-us/library/jj554200.aspx.
[139] Udi Dahan, Clarified CQRS. Dec. 2009. url: http://www.udidahan.com/2009/12/09/clarified-cqrs/.
[140] Martin Fowler. CQRS, http://martinfowler.com/bliki/CQRS.html.
[141] Hendrikse, Z. W., and K. Molkenboer, A radically different approach to enterprise web application development, 2012, http://www.codeboys.nl/white-paper.pdf.
[142] Fitzgerald, Seán, A pattern for state machine persistence using Event Sourcing, CQRS and a Visual Workbench, 2012.
[143] Arunava Chatterjee, Healthy Architectures - Using CQRS and Event Sourcing for Electronic Medical Records, http://www.infoq.com/articles/healthcare-emr-ehr;jsessionid=2E1EE38911CC1632B2012A572455AC2C.
[144] Di Tullio, Dany, and Bouchaïb Bahli. "The impact of Software Process Maturity on Software Project Performance: The Contingent Role of Software Development Risk." *Systèmes d'information & management* 18.3 (2014): 85-116.
[145] Wallace, Linda, Mark Keil, and Arun Rai. "Understanding software project risk: a cluster analysis." *Information & Management* 42.1 (2004): 115-125.
[146] El-Masri, Mazen, and Suzanne Rivard. "Towards a Design Theory for Software Project Risk Management Systems." (2012).
[147] X. Li, T. Schlegela, M. Rotard and T. Ertl, "A Model-Based Graphical User-Interface for Process Control Systems in Manufacturing," in *2nd I*PROMS Virtual International Conference 3–14 July 2006*, 2006.
[148] Nerur, Sridhar, RadhaKanta Mahapatra, and George Mangalaraj. "Challenges of migrating to agile methodologies." *Communications of the ACM* 48.5 (2005): 72-78.

Biografija

Petar Rajković je rođen 5. aprila 1978. godine u Nišu. Osnovnu školu i gimnaziju je završio u Beloj Palanci, a Elektronski fakultet je upisao 1. oktobra 1997. godine. Diplomirao je 12. decembra 2002. godine na Elektronskom fakultetu u Nišu na smeru za Računarsku tehniku i informatiku. Nakon toga upisao je poslediplomske studije na Elektronskom fakultetu u Nišu, a magistarsku tezu „Aspekti realizacije i implementacije medicinskih informacionih sistema“ odbranio je 27. aprila 2009.

Tokom školovanja i studiranja bio je više puta nagradjivan za svoj rad – nosilac je Vukove diplome za odličan uspeh i u osnovnoj i u srednjoj školi, kao i nagrade za najbolji diplomski rad na Elektronskom fakultetu u toku 2012. godine. Takođe, na republičkim takmičenjima mladih matematičara osvojio je dva puta prvo mesto i prvu nagradu (1993. i 1994. godine).

Zaposlen je na Elektronskom fakultetu u Nišu kao asistent na Katedri za Računarstvo od. Prethodno je biran u zvanje asistent-pripravnik (24.12.2004) i istraživač-stipendista (1.3.2003). Angažovan je na izvođenju nastave iz više različitih predmeta sa Katedre za Računarstvo – Algoritama i programiranja, Objektno orijentisanog programiranja, Objektno orijentisanog projektovanja, Zaštite informacija, Multimedijalnih sistema itd.

Što se tiče profesionalnog i naučnog angažovanja, bio je učesnik različitih projekata koji su imali za cilj razvoj i unapređenje kako pojedinih koncepata i prototipova, tako i kompletnih rešenja informacionih sistema za primenu u medicini i industriji. Projekti vezani za medicinske informacione sisteme rađeni su u saradnji sa Domom zdravlja i Kliničkim centrom u Nišu. Projekti razvoja industrijskih informacionih sistema ostvaruju se u saradnji sa ABB Switzerland AG. U sklopu pomenutih projekata, počev od marta 2007. godine više puta je bio na studijskom boravku u Švajcarskoj i Poljskoj.

U skladu sa tim, fokus njegovog naučnog rada je na definisanju efikasnijih strategija za razvoj, implementaciju i održavanje informacionih sistema baziranih na domenskim modelima podataka. Uporedo sa tim, implementacija softverskih alata za generisanje gotovih komponenti i njihova upotreba u razvoju informacionih sistema predstavljaju njegov glavni profesionalni izazov

IZJAVA O AUTORSTVU

Izjavljujem da je doktorska disertacija, pod naslovom:

Unapređenje procesa razvoja i održavanja informacionih sistema primenom domenskih modela podataka

- rezultat sopstvenog istraživačkog rada,
- da predložena disertacija, ni u celini, ni u delovima, nije bila predložena za dobijanje bilo koje diplome, prema studijskim programima drugih visokoškolskih ustanova,
- da su rezultati korektno navedeni i
- da nisam kršio/la autorska prava, niti zloupotrebio/la intelektualnu svojinu drugih lica.

U Nišu, 2.2.2015.god.

Autor disertacije: Petar Rajković

Potpis doktoranda:

____

IZJAVA O ISTOVETNOSTI ŠTAMPANE I ELEKTRONSKE VERZIJE DOKTORSKE DISERTACIJE

Ime i prezime autora: Petar Rajković

Studijski program: Računarstvo i informatika

Naslov rada: Unapređenje procesa razvoja i održavanja informacionih sistema primenom domenskih modela podataka

Mentor: dr Dragan Janković, redovni profesor.

Izjavljujem da je štampana verzija moje doktorske disertacije istovetna elektronskoj verziji, koju sam predao/la za unošenje u **Digitalni repozitorijum Univerziteta u Nišu**.

Dozvoljavam da se objave moji lični podaci, koji su u vezi sa dobijanjem akademskog zvanja doktora nauka, kao što su ime i prezime, godina i mesto rođenja i datum odbrane rada, i to u katalogu Biblioteke, Digitalnom repozitorijumu Univerziteta u Nišu, kao i publikacijama Univerziteta u Nišu.

U Nišu, 2.2.2015.god.

Autor disertacije: Petar Rajković

Potpis doktoranda:

____

IZJAVA O KORIŠĆENJU

Ovlašćujem Univerzitetsku biblioteku “Nikola Tesla” da, u Digitalni repozitorijum Univerziteta u Nišu, unese moju doktorsku disertaciju, pod naslovom:

Unapređenje procesa razvoja i održavanja informacionih sistema primenom domenskih modela podataka koja je moje autorsko delo.

Disertaciju sa svim prilozima predao/la sam u elektronskom formatu, pogodnom za trajno arhiviranje.

Moju doktorsku disertaciju, unetu u Digitalni repozitorijum Univerziteta u Nišu, mogu koristiti svi koji poštuju odredbe sadržane u odabranom tipu licence Kreativne zajednice (Creative Commons), za koju sam se odlučio/la.

1. Autorstvo
2. Autorstvo – nekomercijalno
3. Autorstvo – nekomercijalno – bez prerade
4. Autorstvo – nekomercijalno – deliti pod istim uslovima
5. Autorstvo – bez prerade
6. Autorstvo – deliti pod istim uslovima

U Nišu, 2.2.2015.god.

Autor disertacije: Petar Rajković

Potpis doktoranda:

____