Matematički fakultet

Univerzitet u Beogradu

Razvoj web aplikacije za planiranje medijskog prikazivanja korišćenjem Spring okvira i tehnologije MyBATIS

Mentor:

Prof. Dušan Tošić

Student:

Miroslav Mijajlović

Beograd, septembar 2013.

Rezime

U ovom radu je prikazan razvoj aplikacije za medijsko planiranje korišćenjem Spring okvira i MyBatis objektno-relacionog preslikavanja. Spring predstavlja relativno novu vrstu softverskog alata koji omogućava drugačiji način razmisljanja u razvoju aplikacija. Svojim, donekle strogim pravilima, ukida nečitljivost koda i praktično zabranjuje pravljenje klasa za čije tumačenje je potrebno izdvojiti dosta vremena. Modularnost i raslojavanje aplikacije, po jasno definisanim granicama kakve nam predlaže Spring okvir, predstavljaju jako veliku prednost i podstiču jedan nov način razmišljanja prilikom razvijanja aplikacije. Cilj ovog rada je predstavljanje migracije aplikacije sa jednog načina rada, koji se oslanjao na JDBC tehnologije, na Spring aplikaciju koja komunikaciju sa bazom vrši uz pomoć objektno-relacionog preslikavanja MyBatis tehnologije.

MyBatis je okvir za objektno-relaciono preslikavanje koji SQL upite, naredbe, procedure i funkcije, povezuje sa Java komponentama preko xml deskriptora ili anotacija. Sa MyBatis-om je moguće koristiti sve funkcionalnosti koje nudi SQL, a njegova najveća prednost u odnosu na druge tehnologije objektno-relacionog preslikavanja, je jednostavnost.

Abstract

This paper presents the development of application for media planning by using Spring framework and MyBatis technology for object-relational mapping. Spring is a relatively new type of software tool that allows a different way of thinking in the development of applications. Its rigid rules does not tolerate unreadable code and effectively bans making classes that require a lot of time for interpretation. Spring framework proposes modularity and layering applications with clearly defined borders that can be a very big advantage and encourage for new way of developing applications. The aim of this paper is to present the migration of application from one development mode that relied on the JDBC technology to a Spring application that communicate with the database using the object-relational mapping technology provided by MyBatis.

MyBatis is a framework for object-relational mapping of SQL queries, commands, procedures and functions associated with Java components via XML descriptor or annotations. It is possible to use all the functionalities offered by SQL in MyBatis, and its biggest advantage over other technologies of object-relational mapping is simplicity.

Sadržaj:

1 Predgovor

U današnje vreme je za uspeh u poslovanju, pored dobrog proizvoda, jako bitna i dobra reklama. Neki će reći da je reklama čak i bitnija. Reklame vidimo na svakom koraku: bilbordi, televizijski reklamni spotovi, dodaci u štampanim medijima itd. Ljudi su odavno shvatili da u reklamu treba ulagati jer dobra reklama može biti osnov uspeha. Tako su nastale kompanije čija je osnovna delatnost reklamiranje.

U ovom radu se opisuje aplikacija rađena za firmu čija je osnovna delatnost upravo reklamiranje. Aplikacija je rađena po uzoru na ERP (*eng. Enterprise Resource Planing*) sistem asw:dominus. Zbog svoje specifičnosti aplikacije je realizovana kao zaseban softverski paket. U pitaju je veb aplikacija koja je podignuta na Apache-ovom Tomcat serveru, napisana na Java programskom jeziku, a baza je podignuta na Oracle-ovom serveru.

Prva verzija aplikacije je vršila povezivanje sa bazom isključivo tehnologijom JDBC i rađena je po J2EE standardima. U cilju praćenja novih tehnologija i usavršavanja aplikacije, prelazi se na okvir Spring i povezivanje sa bazom podataka korišćenjem tehnologije MyBATIS.

Obradom i analizom prikupljenih podataka od strane konsultantskog tima u ASW Inženjering-u, rukovodioci konsultantskog i razvojnog tima su usaglasili strategiju i plan razvoja aplikacije. Nakon dobijenih informacija i utvrđenog obima projekta autor ovog teksta je dobio zadatak da realizuje ceo projekat. Nakon isporučenog proizvoda autor je takođe vršio dorade na aplikaciji po primljenim zahtevima za nove funkcionalnosti.

U cilju usavršavanja kako proizvoda tako i stručnog kadra koji radi na njemu, u ERP timu je započet proces prelaska sa starih tehnologija na nove, opšte prihvaćene tehnologije u razvoju Java poslovnih i Web aplikacija. Za početak je rešeno da se aplikacija Mediaplan prebaci na novu tehnologiju jer je dosta manjeg obima od *asw:dominus* aplikacije, a po strukturi su jako slične pa da se na osnovu iskustava iz Mediaplana krene na prebacivanje celog *asw:dominus-a* na novu tehnologiju.

Prebacivanje na novu tehnologiju nije bio zahtev klijenata, već interna potreba tima. Kao osoba koja je razvijala prvu verziju aplikacije, autor ovog teksta je dobio zadatak da Mediaplan prevede na novu tehnologiju.

Tema ovog rada je prebacivanje aplikacije na tehnologije Spring i MyBatis. U radu će biti opisane korišćene tehnologije, kao i projekat i njegova realizacija.

2 O projektu Mediaplan

Cilj realizacije dogovora dve firme je izrada strategije razvoja i primene informacione tehnologije u poslovnim procesima kompanije Multikom group i svim njenim članicama.

Strategija razvoja projekta bazirana je na informacijama dobijenim različitim tehnikama prikupljanja informacija: analizom trenutnog stanja, aktuelnoj informatičkoj podršci poslovnim procesima, upitnicima i intervjuima vezanim za predloge poboljšanja funkcija i velikom iskustvu u razvoju i implementaciji integralnog informacionog sistema konsultantskog tima ASW Inženjering-a.

Razvijeno softversko rešenje generiše aktuelne informacije neophodne za upravljanje sistemom i obezbeđuje centralizaciju podataka i distribuiranost funkcija kao osnov za ažurnost i relevantnost informacija.

2.1 Razvoj prvobitne verzije aplikacije

Aplikacija Mediaplan je, po svom aplikativnom i prezentacionom delu, izrađena po uzoru na ERP proizvod *asw:dominus* koji se razvijao već više od jedne decenije. Mediaplan je od *dominus* projekta nasledio konvenciju pisanja klasa, definisanja naziva klasa, paketa, način izrade *jsp* strana, kao i sam izgled koji je praktično isti.

Projekat je realizovan u troslojnoj arhitekturi.

NIVO I
Prvi nivo je DB Server, aktuelan je Oracle RDBMS.
Poslovna logika je implementirana delom PLSQL procedurama, a većim delom u Java klasama srednjeg sloja (NIVO II).
Za razvoj uskladištenih procedura se koristi proceduralni jezik koji odgovara serveru baze, npr. Oracle PL/SQL i sl.

NIVO II
Drugi nivo je aplikativni/Veb Server. Programsko rešenje koristi Tomcat Application Server, Apache Web Server i Jasper Reports.
Srednji nivo i izveštaji su konstruisani tako da su portabilni na različite servere baza (do sada su bili zastupljeni Oracle, Informix).

NIVO III
Treći nivo je radna stanica (Client) uz pretraživač kao izvršno programsko okruženje.
Grafičko korisničko okruženje po uzoru na operativni sistem Windows (Graphical User Interface) je realizovano u HTML-u i JavaScriptu. Opreativni sistem na klijentu je Windovs, a pretraživači su Internet Explorer 7 i noviji.

Za razvoj klijentskog dela aplikacije koriste se JAVA, HTML, JavaScript. Za vezu sa bazom podataka koristi se JDBC-ov driver.

Razvoj ove verzije aplikacije se odvijao početkom 2012. godine. Korisnik je tu verziju koristio od aprila 2012. do februara 2013. godine.

2.2 Motivi za uvođenje novih tehnologija u razvoj projekata

Sa starom aplikacijom je postojao problem proširivosti funkcionalnosti, dodavanja novog koda, mogućnost ponovnog iskorišćavanja rezultata nekog upita predstavljenog klasom *PreparedStatment* u JDBC-u. Želja je bila da se stvori modularan sistem, gde će funkcionalnosti biti jasno razdvojene od njihove implementacije što će dodavanje novih funkcionalnosti, pa čak i promenu tehnologije, na primer promenu sistema za upravljanje bazama podataka, učiniti bezbolnijim. U tom smislu Spring MVC (Model View Controller – model, izgled, kontroler) se nametnuo kao rešenje.

3 Opis tehnologija i alata korišćenih u izradi

3.1 Java platforma

Opšte je poznato da je prednost Java jezika nezavisnost od operativnog sistema na kojem se izvršava, što obezbeđuje prenosivost aplikacije. Aplikacija se može izvršiti na bilo kom operativnom sistemu pod uslovom da za taj sistem postoji JRE (Java Runtime Environment), tj. JVM virtuelna mašina koja interpretira Java bajtkod preslikavajući ga u mašinski jezik konkretnog računarskog sistema. Takođe, Java je nezavisna i u odnosu na sisteme za upravljanje bazama podataka (DBMS – Database Management System).

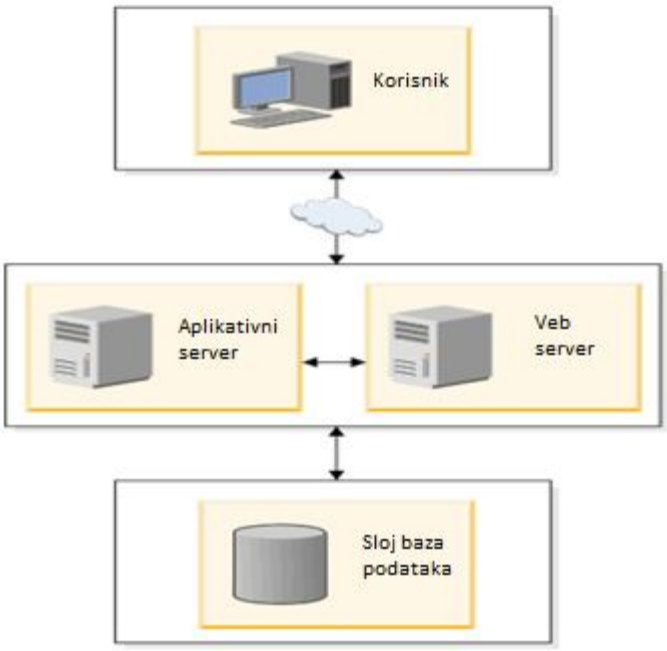
Postoje 3 izdanja Jave [5]:

- Java standardno izdanje (J2SE Java 2 Standard Edition) jezgro za ostala izdanja
- Java poslovno izdanje (J2EE Java 2 Enterprise Edition) obezbeđuje razvoj poslovnih aplikacija
- Java Micro Edition (J2ME) okruženje za razvoj aplikacija na mobilnim uređajima i ostalim uređajima koji imaju manjak resursa da podrže J2EE.

3.1.1 J2EE

Platforma *J2EE* koristi distribuirani model koji se sastoji od više nivoa (obično su u pitanju 3 nivoa). *J2EE* aplikacija sastoji se od *J2EE* komponenti koje su obično instalirane na

različitim računarima. Komponente, odnosno računari na kojima se izvršavaju komponente, su dodeljene različitim nivoima *J2EE* aplikacije.



Slika 1: Skica troslojne strukture

- Klijentski nivo – komponente koje se izvršavaju na klijentskim računarima.
- Nivo aplikacione logike – komponente koje se izvršavaju na J2EE (aplikacionom) serveru. Ovaj nivo se deli na Veb nivo i poslovni nivo.
- Treći nivo čine komponente koje se izvršavaju na serveru baze podataka. Obično se ovaj nivo naziva EIS nivo (*eng. Enterprise Information System*).

3.1.1.1 J2EE komponente

J2EE aplikacije su sastavljene od J2EE komponenti. J2EE komponente su samostalne funkcionalne softverske jedinice J2EE aplikacije u kojima su grupisane klase i fajlovi, a koje mogu komunicirati sa drugim komponentama. J2EE specifikacija definiše sledeće komponente:

· Komponente koje se izvršavaju na klijentu – Veb klijenti, aplikacioni klijenti i apleti,

· Veb komponente koje se izvršavaju na serveru - Java Servleti i JSP,

· Komponente poslovne logike koje se izvršavaju na serveru - EJB (Enterprise Java Bean) komponente.

J2EE komponente su napisane i kompajlirane u programskom jeziku Java.

3.1.1.2 Klijentske komponente

Što se tiče klijentskih komponenti, one mogu biti Veb klijenti, aplikacioni klijenti ili apleti.

Veb klijenti se sastoje iz dva dela: dinamičkih Veb strana napravljenih korišćenjem različitih tipova jezika za označavanje (HTML, XML, itd.). One generišu Veb komponente koje se prave na veb sloju i prikazuju u okviru Veb pretraživača, koji prikazuje stranice primljene od servera. Veb klijenti se često nazivaju tankim klijentima (*eng. thin clients*). Tanki klijenti obično ne rade stvari kao što su upiti u baze, izvršavanje vezano za softver koji se odnosi na poslovnu logiku itd. već to prepuštaju EJB komponentama koje se izvršavaju na J2EE serveru.

Veb stranica primljena od Veb sloja može sadržati i aplet. Aplet je mala klijentska aplikacija napisana u programskom jeziku Java koja se izvršava na Java virtualnoj mašini (JVM) instaliranoj u pretraživaču. Takođe Veb komponente omogućavaju čistiji i modularniji dizajn aplikacije jer obezbeđuju odvajanje programiranja aplikacije od dizajna Veb strana. Personal koji učestvuje u dizajnu Veb strana ne mora poznavati Java programski jezik.

J2EE aplikacioni ("debeli") klijenti izvršavaju se na klijentskoj mašini i imaju bogatiji grafički korisnički interfejs (Graphical user interface - GUI), koji sadrži Swing i AWT komponente, mada je moguć i komandni interfejs. Aplikacioni klijenti direktno pristupaju java zrnima (*eng. Bean*) na poslovnom nivou. Takođe, aplikacioni klijent može otvoriti HTTP konekciju da bi uspostavio komunikaciju sa servletom koji se izvršava na Veb serveru.

3.1.1.3 Serverske komponente

J2EE Veb komponente mogu biti servleti i JSP strane. Servleti su Java klase koje dinamički procesiraju zahteve i konstruišu odgovore. JSP strane su tekst-bazirani dokumenti koji se izvršavaju kao servleti ali dozvoljavaju prirodniji pristup stvaranju statičkog sadržaja.

Statičke HTML strane i apleti se povezuju sa Veb komponentama tokom kreiranja aplikacije, ali ne spadaju u Veb komponente prema J2EE specifikaciji.

Veb sloj može sadržati i komponente Java zrna koje obrađuju korisničke zahteve i šalju upite na procesiranje EJB-ovima koji se izvršavaju na sloju poslovne logike.

3.1.1.4 Komponente poslovne logike

Enterprise Java Beans (EJB) su J2EE komponete koje su bazirane na EJB tehnologiji. To je specifikacija koju je definisao *Sun Microsystems* i ona predstavlja definiciju serverskih komponenti za razvoj višeslojnih arhitektura sa distribuiranim objektima. Ona definiše okruženje u kom se izvršavaju EJB komponente. One se izvršavaju u EJB kontejneru, izvršnom okruženju unutar Sun Java System platforme, mada su vidljive za aplikacionog programera. EJB kontejner obezbeđuje sistemske servise kao što su transakcije i sigurnost za

svoja zrna. Ovi servisi omogućavaju brz razvoj i razmeštaj zrna koja čine srž transakcionih J2EE aplikacija.

EJB-ovi, koji se izvršavaju na nivou poslovne logike, rukuju poslovnom logikom pomoću koje se rešava ili zadovoljava potreba polja rada kao što je bankarstvo, maloprodaja ili finansije. Komponente poslovne logike implementiraju se preko EJB-ova i izvršavaju se na aplikacionom serveru. Komponente poslovne logike posreduju između Veb slojeva i baze podataka. EJB prima podatke od klijentskih programa, procesira ih, ako je to potrebno, i šalje ih u (enterprice information system) EIS sloj na čuvanje. EJB takođe obavljaju i obrnut proces - uzimaju podatke iz baze, procesiraju ih i šalju klijentskom programu.

Postoje 3 vrste EJB-ova [5]:

1) Sesijski (session bean)
2) Entitetski (entity bean)
3) Porukama vođen (message-driven bean)

Sesijska zrna služe za implementaciju poslovnih pravila i realizaciju procesa poslovne logike. Ona upravljaju informacijama koje se odnose na konverzaciju između klijenta i servera. Izvršavaju zadatak ili proces za klijenta. Kada klijent završi rad sa njim, zrno i podaci su izgubljeni.

Entitetska zrna služe za memorisanje poslovnih podataka u jednoj vrsti tabele baze podataka, koji su deo poslovne logike. Trajna su jer manipulišu trajnim domenskim podacima aplikacija. Po završetku rada stanje je sačuvano u bazi.

Porukama vođena zrna kombinuju obe gore navedene tehnike dozvoljavajući asinhrono primanje poruka.

3.1.1.5 J2EE kontejner

J2EE server pruža servise u okviru kontejnera, specifične za pojedine tipove komponenti. Servisi se izvršavaju u pozadini poslovne logike o kojoj jedino programer vodi računa. Pre izvršenja zrna, ono se mora uključiti u J2EE aplikaciju i rasporediti (*eng. deploy*) u pripadajući kontejner. Kontejneri takođe upravljaju servisima koji se ne mogu podešavati (životni ciklus EJB i Servleta, konekcija prema bazi, perzistencija).

Proces smeštanja uključuje podešavanja kontejnera svih komponenti J2EE aplikacije i podešavanja za samu J2EE aplikaciju. Podešavanja kontejnera obuhvataju podršku niskog nivoa od J2EE servera koji uključuje servise kao što su sigurnosni servisi, upravljanje transakcijama, JNDI(*eng. Java Naming and Directory Interface*) i daljinsko povezivanje.

Proces povezivanja komponente sa aplikacijom uključuje podešavanje kontejnera za svaku komponentu J2EE aplikacije, kao i za samu aplikaciju, uz obezbeđenje sledećih servisa [5]:

- J2EE sigurnosni model omogućava podešavanje Veb komponente ili EJB radi zabrane pristupa neautorizovanim korisnicima,
- J2EE transakcioni model,
- JNDI (Java Naming and Directory Interface) pretraživački servisi(*eng. lookup service*)
- J2EE model daljinskog povezivanja.

J2EE sigurnosni model dozvoljava konfigurisanje Veb komponente ili EJB-a tako da sistemskim resursima mogu pristupiti samo autorizovani korisnici, dok je neautorizovanim pristup zabranjen.

J2EE transakcioni model omogućava specifikaciju veza između metoda koje čine jednu transakciju tako da se sve metode u jednoj transakciji tretiraju kao jedna jedinica, tj. J2EE omogućava deklarativno postavljanje granica transakcija i upravljanje transakcijama kontejnerom.

JNDI pretraživački servisi obezbeđuju jedinstveni interfejs za servise imenovanja, koji pronalaze resurse ili objekte kojima J2EE komponente pristupaju.

J2EE model daljinskog povezivanja (*eng. Remote Connectivity model*) upravlja komunikacijom niskog nivoa između EJB i klijenta. Nakon što je EJB napravljen, klijent proziva metode kao da su na istoj virtuelnoj mašini.

J2EE server obezbeđuje EJB i Veb kontejnere koji upravljaju izvršavanjem EJB-ova, kao i JSP stranica i Servleta, respektivno. Oni se izvršavaju na J2EE serveru. Pored pomenutih kontejnera, postoje i kontejner aplikacionog klijenta i aplet kontejner, koji upravljaju izvršavanjem komponenti aplikacionog klienta i apleta, respektivno. Oni se izvršavaju na klijentu.

3.2 Okvir Spring

Od samog početka razvoja J2EE aplikacija od 1999. godine pa do danas, nije došlo do nekog velikog uspeha te tehnologije u praksi. Stvari su praktično ostale iste. Ipak, postavljeni su mnogi standardi u smislu upravljanja transakcijama, razvoja višeslojnih aplikacija kao i razvoja i proširivanja mahom srednjeg sloja. Osnovni problem kod J2EE aplikacija je njihova složenost. Zahtevaju jako veliki trud u njihovom razvoju, a nije učinjeno mnogo na poboljšavanju performansi. Okvir Spring značajno umanjuje složenost aplikacija, smanjuje EJB nedostatke kod „teških“ komponenti i pokazao se kao veoma prilagodljiv. Naime, aplikacija ne mora u potpunosti da primenjuje Spring, što prevođenje aplikacije na tu

tehnologiju čini bezbolnijim. Modularan pristup omogućava da se uz pomoć Spring-a mogu razviti jako složene poslovne aplikacije, ali i jednostavnije aplikacije, na nivou Java apleta. Spring je primenljiv u velikom spektru okruženja, a ne samo u serverskom J2EE okruženju. Spring nije zamena za J2EE, već njegova nadogradnja.

3.2.1 Problemi u tradicionalnom pristupu J2EE

Pokazalo se da J2EE aplikacije vremenom počnu da sadrže sve više „krpljenog" koda. Daljim razvojem i doradama tako nastane i dobar deo koda koji apsolutno ništa ne radi ali crpi dosta resursa, umesto da bude skoncentrisan na poslovnu logiku. Takav primer su gomila try/catch blokova za oslobađanje ili dohvatanje JDBC objekata. Takođe, mnoge J2EE aplikacije koriste distribuirani model tamo gde on nije neophodan, što duplira dosta koda koji obezbeđuje tu distribuiranost koja za aplikaciju nije od suštinskog značaja.

Jedan od najvećih problema je i što su J2EE aplikacije jako teške za testiranje njenih jedinica. To je zato što je veći deo logike J2EE definisan pre nego što je pokret agilnog razvoja softvera uzeo maha. Testovi jedinica su ključni u otkrivanju mogućih grešaka.

3.2.2 Arhitektura Spring-a

Okvir Spring je Java tehnologija koja omogućava sveobuhvatnu podršku za razvoj Java aplikacija.

U Springu se može napisati metoda koja izvršava transakciju na nivou baza podataka, bez ulaženja u strukturu transakcionog okruženja (*eng. transaction API*). Takođe, možemo napraviti metodu pomoću koje se izvršava neki proces na udaljenoj lokaciji bez znanja o okruženju na toj lokaciji.

Prvu verziju napisao je Rod Džonson, koji je pustio izdanje zajedno sa publikacijom njegove knjige „Expert One-on-one J2EE Design and Development" u oktobru 2002. Okruženje je radilo pod Apache 2.0 licencom u junu 2003. Nakon toga, rađene su mnoge nadogradnje i tako su se izdavale nove verzije. Verzija Spring 2.0 izdata je u oktobru 2006, Spring 2.5 u novembru 2007. Spring 3.0 u decembru 2009 i Spring 3.1 u decembru 2011. Trenutna verzija je Spring 3.2.4 [7].

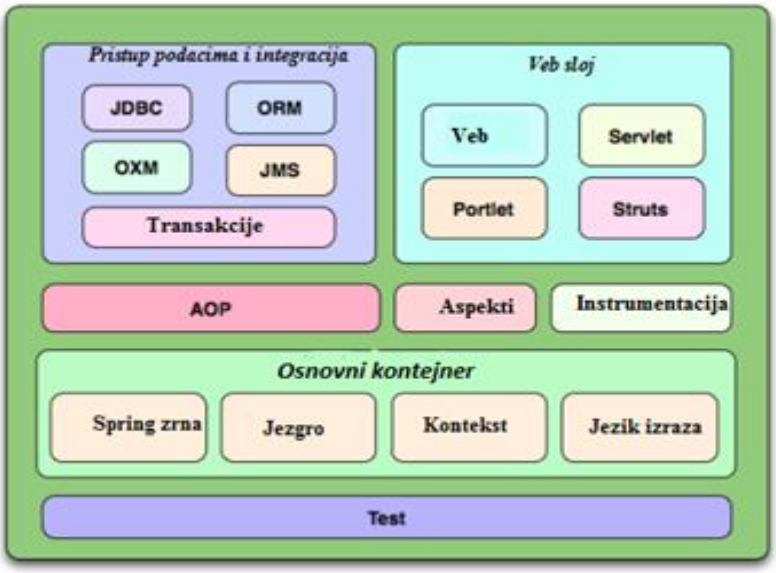
Spring ima slojevitu arhitekturu, što znači da mi možemo izabrati samo neki od slojeva i njega koristiti potpuno nezavisno od celine. To nam ostavlja dosta prostora za

pojednostavljenje u radu. Na primer, možemo koristiti Spring samo radi lakšeg korišćenja JDBC-a, a isto tako ga možemo kristiti za upravljenje svim poslovnim objektima. Još jedna njegova bitna osobina je ta što se lako i postepeno uvodi u postojeće projekte.

Pored toga, Spring je kreiran "odozdo na gore" kako bi olakšao pisanje koda koji je lak za testiranje. Spring je odlično okruženje za razvoj projekata vođenih testiranjem(*eng. test driven projects*) [1].

Spring postaje sve značajnija integraciona tehnologija čije prednosti prepoznaje nekoliko kompanija važnih u softverskom svetu.

Moduli:



Slika 2: Spring slojevi: podela i raslojavanje (izvor http://static.springsource.org*)*

Na slici su prikazani moduli Spring-a [1]. O njima će detaljnije biti izloženo u narednim poglavljima.

3.2.2.1 Osnovni kontejner

Modul jezgro i moduli zrna

Ovi moduli čine osnovu okvira Spring. Oni su nosioci mogućnosti koje nudi Inverzija kontrole (*eng. inversion of control*) i umetanje zavisnosti (*eng. dependency injection*).

Suština ovog modula je da obezbeđuje slabo povezivanje (loose coupling).

Kontekst modul

Ovaj modul nasleđuje mogućnosti iz modula zrna (eng. Beans modul) i dodaje podršku za internacionalizaciju, širenje događaja, učitavanje resursa. Interfejs *ApplicationContext* i njegove implementacije su centralni deo ovog modula.

Modul opisnog jezika (Expression Language)

Modul opisnog jezika daje mogućnost manipulisanja objektima. Taj jezik podržava mogućnost uzimanja i postavljanja vrednosti, poziva metoda, dohvatanje objekata po imenu iz kontejnera inverzije kontrole. Kontejner inverzije kontrole je predstavljen xml konfiguracionim fajlom koji sadrži Spring zrna i njihovu specifikaciju.

3.2.2.2 Pristup podacima (Data Access/Integration)

Ovaj sloj se sastoji od JDBC, ORM, OXM, JMS i transakcionog modula.

JDBC (*Java Database Connectivity*) modul otklanja potrebu za prevelikim starim JDBC programiranjem koje je prilagođeno specifičnom sistemu za upravljanje bazama podataka (Oracle, mySQL, IBM DB2, ...). Rad sa JDBC-om, zahteva pisanje koda koji uzima konekciju, pravi upit klasom *PreparedStatement*, obrađuje rezultat i na kraju prekida vezu sa bazom. Pomoću Spring-ovog JDBC i Data Access Objects (DAO) modula, nije neophodno ponavljanje koda i njihovim korišćenjem se dobija mnogo čistiji, pregledniji i kod daleko lakši za održavanje i izbegavaju se problemi koji nastaju pisanjem koda sa JDBC-om.

ORM (Objektno-relaciono preslikavanje) omogućuje integraciju sa popularnim okruženjima (*API*) za objektno relaciono preslikavanje, kao što su Hibernate, JPA (*Java Persistence API*), JDO (*Java Data Objects*), iBatis (MyBatis). Koristeći ovaj modul moguće je pored izvršavanja integracije sa pomenutim ORM alatima koristiti i druge mogućnosti Spring-a, kao što je na primer deklarativno upravljanje transakcijama. O ovom modulu će više reči biti kasnije.

OXM (Objekntno/XML preslikavanje) je apstraktni sloj koji omogućuje implementaciju ove vrste preslikavanja i rad sa xml bazama podataka.

JMS (*Java Messaging Service*) je modul koji omogućuje rad sa porukama između 2 ili više klijenata.

Transakcioni modul se bavi upravljanjem transakcijama koje se odvijaju u samim klasama i reguliše rad nad operacijama sa bazom podataka.

3.2.2.3 Veb

Veb sloj sadrži Web, Web-Servlet, Web-Struts i Web-Portlet module.

Springov Web modul omogućava standardne mehanizme kao što su otpremanje (upload) fajla iz više delova. Takođe, ovom modulu pripada i Spring MVC implementacija koja se može koristiti za razvoj web aplikacija.

Web-Servlet modul sadrži Spring MVC (model-view-controller) implementaciju za veb aplikacije. MVC zasnovane aplikacije su one aplikacije kod kojih su jasno razgraničeni slojevi (Model služi za dovlačenje podataka u onom obliku u kojem su smešteni u bazi podataka, pogled služi isključivo za prikazivanje tih podataka u, recimo, Veb pretraživaču kod Klijenta, Kontroler predstavlja spregu između ova dva sloja jer pokupi podatke iz modela i obradi ih tako da budu pogodni za prikazivanje.) Spring MVC okruženje obezbeđuje čisto razgraničenje veb formi i logičkog sloja i potom ih integriše sa ostalim rutinama okruženja Spring.

Web-Struts modul sadrži klase koje služe kao podrška integraciji klasičnog Struts-a (čiji je cilj razdvajanje logičnog dela od pogleda i od kontrolera kod MVC) u Spring aplikaciji.

Web-Portlet predstavlja korisnički interfejs koji je ugradiv na nekom Veb portalu, a implementira funkcionalnost koja je nezavisna od tog portala. Na primer, na Veb portalu nekih časopisa, portlet bi predstavljala vremenska prognoza ili u okviru nečijeg sajta mogućnost slanja e-mail.

3.2.2.4 AOP i instrumentacija

Springov AOP (*Aspect-Oriented Programming*) modul obezbeđuje mehanizme za korišćenje Aspektno orijentisanog programiranja koji omogućava korisniku da na jednostavan način razdvoji komponente aplikacije implementirajući funkcionalnosti koje su logički odvojene. Ovaj modul poseduje podršku za aspektno-orijentisano programiranje koje obezbeđuje razvoj odvajajući poslovnu logiku od sistemskih usluga (kao što je npr. upravljanje transakcijama). Objekti rade samo ono što treba da rade – izvršavaju poslovnu logiku i ništa više. Objekti nisu odgovorni (ili čak svesni) sistemskih koncepata poput logovanja ili podrške transakcijama. Trenutno najkorišćeniji jezik koji implementira AOP metodologiju je AspectJ.

3.2.2.5 Test modul

Test modul podržava testiranje komponenti Springa korišćenjem JUnit-a ili TestNG-a. On omogućava konstantno punjenje Spring aplikacionog konteksta i prikupljanje komponenti definisanih u kontekstu. Takođe, omogućava test objekte koji mogu biti korišćeni za testiranje izolovanog koda.

3.2.2.6 Inverzija kontrole

U svetu Java aplikacija je bilo raznih kontejnera koji pomažu da se skupe komponente iz različitih projekata u jednu kohezivnu aplikaciju. Način povezivanja, koji je takoreći pobedio, je inverzija kontrole. On se može u neku ruku shvatiti kao projektni obrazac (Design pattern). On radi po tzv. „Holivudskom principu“ – „Nemoj me ti zvati, ja ću pozvati tebe“. To je odgovor koji holivudski menadžeri često daju glumcima amaterima. Drugim rečima, upotreba toga je u sledećem. Ako jedna klasa, kao privatni atribut ima drugu klasu i u nekoj metodi ili konstruktoru ona instancira tu klasu, klasa koja je atribut kaže: „Nemoj me ti kreirati, ja ću se kreirati preko nekog drugog!“

U osnovi okvira Spring je umetanje zavisnosti. Spring kontejner omogućuje umetanje objekata u druge objekte. Umetanje zavisnosti u Spring-u se definiše na 2 moguća načina: umetanje konstruktora ili umetanje preko set-metoda.

Java klase u okviru Spring najčešće nazivamo Spring zrnima (*Spring beans*) ili samo zrnima. Spring-ov kontejner održava konfiguraciju, najviše zasnovanu na XML konfiguracionim fajlovima (application context fajlovima) i anotacijama. On takođe upravlja Java klasama koristeći *BeanFactory*. Kontejner koristi tu tzv. fabriku zrna za pravljenje novih objekata. Novi objekti se prave kao unikati u smislu da se pravi samo jedna instanca ako nije navedeno drugačije.

Interfejs *org.springframevork.beans.factory.BeanFactory* je ustvari predstavnik Springovog IoC kontejnera (*Inversion of Control)* i odgovorna je za upravljanje gore pomenutim zrnima. Interfejs *BeanFactory* je zadužen za instanciranje objekata, kao i za skupljanje zavisnosti među tim objektima.

Spring kontejner koristi meta-podatke (koji su definisani ili u xml fajlovima ili delom u anotacijama) da bi znao kako da instancira, konfiguriše i okupi objekte u aplikaciji. Najčešće se ti podaci čitaju iz xml fajla. Konfiguracija se sastoji iz najmanje jedne definicije zrna kojim kontejner mora upravljati. Zrna odgovaraju stvarnim objektima koji postoje u jednoj aplikaciji. Uobičajeno je imati definicije zrna za servisne objekte i za DAO objekte, objekte koji služe za pristup podacima, najčešće iz baze podataka (DAO dolazi od Data Access Object).

Instanciranje IoC kontejnera se vrši na sledeći način:

```
ApplicationContext context = nev ClassPathXmlApplicationContext(
        new String[] {"services.xml", "daos.xml"});
BeanFactory factory = context;
```

Primer 1: IoC kontejner – instanciranje ApplicationContext-a

ApplicationContext nasleđuje *BeanFactory*.

Definicija zrna u okviru konfiguracionog fajla se sastoji od punog imena klase, elemenata koji definišu ponašanje zrna (domen, životni ciklus itd.), zrna od kojih ono zavisi, tj.

takozvanim kolaboratorima i drugih podešavanja od kojih se neka mogu odnositi na broj konekcija ka bazi kojima zrno upravlja.

Svako zrno ima jedan ili više identifikatora (ili ime). To ime mora biti jedinstveno u okviru jednog kontejnera. Uobičajena je "camel-case" konvencija. U slobodnom prevodu slova pišemo kao kamilje grbe. Počinjemo malim slovima pa sledeću reč odvajamo tako što je započnemo velikim slovom, kao npr. *ioTextTemplate*. Nije obavezno definisati ime zrna. Ako se to ne uradi, kontejner će sam dodeliti jedinstveno ime.

Primer definicije zrna u konfiguracionom fajlu:

```
<bean  id="grupisanjeaService" parent="basicService">
       <property name="dao" ref="grupisanjeaDAO"/>
       <property name="dataBeanClass" value="asw.iis.media.model.Grupisanjea"/>
</bean>
```

Primer 2: Definicija zrna – xml konfiguracioni fajl

U ovom primeru se definiše zrno *grupisanjeaService*, čiji je otac zrno *basicService*, a kolaboratori (zrna koja određuju zavisnosti zrnu grupisanjeaService) su svojstva sa imenima *dao* i *dataBeanClass*. Ključna reč *ref* označava da je u pitanju zrno koje je već definisano u istom konfiguracionom fajlu, a *value* označava klasu koja implementira to zrno.

Definicija zrna se može posmatrati kao recept kako se definiše jedan objekat tog tipa. Postoje 2 vrste instanciranja zrna: preko konstruktora i preko statičke fabričke (factory) metode.

```
public class Medij {
       private Medijtip medijtip;

       public Medij () {
              medijtip = new Medijtip ();
       }
}
```

Primer 3: Instanciranje zrna bez inverzije kontrole

```
public class Medij {
       private Medijtip medijtip;

       public void setMedijtip(Medijtip medijtip) {
              this.medijtip = medijtip;
       }

}
```

Primer 4: Instanciranje zrna sa inverzijom kontrole [8]

U prvom primeru, ako iz bilo kog razloga nije moguće instancirati objekat tipa Medijtip, cela Medij klasa nailazi na problem još u konstruktorskoj inicijalizaciji. Zato se Medij klasa pravi tako što će se pustiti da neko treći razmišlja o pravljenju instanci klasa koje su joj atributi.

Ključni principi inverzije kontrole [8]:

- Klase koje se pozivaju na druge klase (agregiraju ih) ne treba da zavise od direktne implementacije agregiranih klasa. Obe vrste klasa treba da zavise od apstrakcije. Tako u primeru: Medij klasa ne treba da zavisi od implementacije klase Medijtip, već i jedna i druga treba da zavise od apstrakcije koja je definisana bilo kroz interfejs, bilo kroz apstraktnu klasu.

- Apstrakcija ne sme da zavisi od detalja. Detalji treba da zavise od apstrakcije.

- Objekti napravljeni uz pomoć ovog kontejnera se često nazivaju „upravljani objekti“ ili zrna. Obično, kontejner se konfiguriše tako što se učitavaju XML fajlovi koji sadrže Bean definicije koje sadrže neophodne informacije za pravljenje objekata.

- Objekti se mogu dobiti metodama traženja zavisnosti ili umetanja zavisnosti. Pretraživanje zavisnosti (*Dependency lookup*) je uzorak gde „pozivač“ traži od kontejnera objekat sa specifičnim imenom ili specifičnog tipa. Umetanje zavisnosti (*Dependency injection*) je uzorak gde kontejner predaje objekat sa određenom karakteristikom drugom objektu preko konstruktora ili neke metode za pravljenje objekta.

Inverzija kontrole (IoC) u Spring-u je primena uzorka *Umetanja zavisnosti (Dependency Injection - DI)*.

3.2.2.7 Umetanje zavisnosti

Svaka netrivijana aplikacija sastoji se od više klasa koje međusobno sarađuju kako bi implementirale neku poslovnu logiku. Uobičajeno je da svaki objekat treba sam da pokupi potrebne reference drugih objekata, što može dovesti do čvrstih veza između objekata i zavisnosti različitih komponenata sistema. Što je veća zavisnost, znači da će prilagođavanje sistema eventualnim promenama biti teže, promene na jednoj komponenti najčešće će zahtevati promene i na drugim zavisnim komponentama, a i testiranje će samim tim biti komplikovanije. Kada se primeni uzorak ubacivanja zavisnosti, objektima se prilikom njihovog kreiranja dodeljuju reference ka drugim objektima od strane nekog eksternog entiteta. Najčešće je u pitanju konfiguraciona datoteka u kojoj su te zavisnosti definisane. Ona se čita prilikom pokretanja aplikacije kada se prave instance svih tih klasa. Najveća korist od ubacivanja zavisnosti je oslabljena veza između objekata (*loose coupling*) koja je postignuta time što se te veze definišu na jednom mestu i spolja, a ne u samim klasama.

Ovakvim načinom rada u jednoj aplikaciji postižemo da komponente, odnosno klase, budu što je manje moguće zavisne jedna od druge. To povećava mogućnost ponovnog korišćenja tih klasa i olakšava mogućnost nezavisnog testiranja. Da bismo razdvojili jednu Java komponentu od druge i zavisnost jedne klase od druge, treba reći toj klasi da zavisi od te druge klase umesto da u njoj pravimo instancu te klase. Tako klasa A zavisi od klase B ako koristi klasu B kao atribut, odnosno svojstvo. Drugim rečima dozvoljeno je da u klasi imamo definiciju:

`private B instancaB;` a ne da je u metodama klase A pravimo:

```
instancaB = new B();
```

Postoje 3 načina umetanja zavisnosti u Spring-u [8]. Klasa B se ubacuje u klasu A

- preko njenog konstruktora - to se zove konstruktorsko umetanje
- preko setter metode i to se zove Setersko umetanje
- umetanje metoda (Method Injection) .

Prilikom korišćenja umetanja metoda kontejner je zadužen da u vreme izvršavanja aplikacije implementira metode. Na primer, objekat može da sadrži metodu koja je deklarisana kao `protected abstract` metoda, pri čemu će kontejner u vreme izvršenja da je realizuje. Ova forma umetanja zavisnosti se najređe koristi.

Izbor između preostala 2 načina je deo jednog opštijeg pitanja u objektno-orijentisanom programiranju, a to je da li da se vrednosti polja postave u konstruktoru ili preko set metoda. Konstruktori preko parametara daju jasnu i očiglednu odrednicu šta se treba napraviti. Ukoliko ima više načina da se napravi instanca nekog objekta, samo je potrebno napraviti više konstruktora sa različitim brojem parametara. Međutim, ukoliko zaista ima više načina da se napravi instanca nekog objekta, teže je opisati to konstruktorima jer oni mogu samo varirati u broju i tipovima parametara. U slučaju da klase nasleđuju jedna drugu i da imaju veći broj konstruktora struktura postaje još složenija. Zato je umetanje seter metodama najpopularniji vid umetanja zavisnosti u Spring-u.

```
<bean  id="podmedijController" parent="basicController">
       <property name="service" ref="podmedijService"/>
       <property name="dataBeanClass" value="asw.iis.media.model.Podmedij"/>
</bean>
```

Primer 5: Umetanje zavisnosti seterima – zavisnosti su ubačene u zrno

Pored ovih načina koji se koriste u okviru Spring postoje i ubacivanja korišćenjem interfejsa. Metoda je deklarisana u interfejsu a zavisna klasa je implementira. Zavisnosti se prosleđuju preko parametara metode. Taj način nije popularan, jer je zaživelo mišljenje da klasa ne treba da konfiguriše samu sebe, već treba biti konfigurisana spolja.

3.2.2.8 Područje definisanosti zrna

Pravljenjem definicije zrna, pravi se recept po kojem će se napraviti instance definisane na taj način. Pored kontrolisanja zavisnosti, može se kontrolisati i područje definisanosti (scope) objekata koji ovako nastaju. Ovaj pristup je jako fleksibilan jer je moguće definisati ciklus i definisanost objekta prilikom njegove konfiguracije umesto da se to radi kroz Java klase. Spring podržava 5 različitih područja definisanosti, ali se može kreirati i ciklus po želji kao kombinacija svojstava ovih 5 [1].

Tabela 1: Područje definisanosti zrna

Singlton	Ovako kreirana instanca objekta će biti jedina u celom kontejneru. Ovo je podrazumevana konfiguracija.
Prototip	Ista definicija zrna se koristi za bilo koji broj instanci.
Zahtev (request)	Svaki HTTP zahtev ima jednu jedinu instancu ovog zrna.
Sesija	Svaka HTTP sesija ima jednu jedinu instancu ovog zrna.
Globalna sesija	Postoji jedna instanca ovog zrna na nivou globalne HTTP sesije. Koristi se samo u kontekstu portleta.

Za projekat kojim se bavimo su nam važne samo prve 2 vrste: Singlton i Prototip.

3.2.2.8.1 Singlton područje definisanosti

Ovo područje definisanosti je podrazumevano ako se ništa drugo ne navede.

Po njemu samo jednom deljenom instancom ovako definisanog zrna se upravlja i svi zahtevi sa zrnom čiji se id poklapa sa njim (singlton zrnom) rezultiraju tom istom instancom zrna.

Drugim rečima, definisanjem singlton zrna, Spring IoC kontejneru kažemo da kreira tačno jednu instancu objekta navedenu u konfiguracionoj datoteci. Ta jedna jedina instanca se čuva u kešu, tako da svi zahtevi za tim zrnom povlače taj objekat iz keša.

3.2.2.8.2 Prototip područja definisanosti

Nova instanca prototip-zrna se kreira svaki put kada se uputi zahtev za tim zrnom. Svaki put se nova instanca tog zrna ubaci u objekat koji ga traži. Obično ovakva zrna nose neko stanje koje se naziva "konverzacijsko" stanje. Tako da se pri definisanju zrna mora reći sa kojim posebnim stanjem želimo da kreiramo instancu tog zrna. Primer bi bio prototip neke datoteke u koju moramo pisati neke informacije; ta datoteka bi se sastojala od 3 dela: zaglavlje, glavni sadržaj i podnožje (*footer*) i za sva tri se različito upisuju informacije. U tom slučaju ćemo instancirati 3 posebna objekta za svaki deo datoteke.

Treba naglasiti da, za raliku od drugih područja definisanosti, Spring ne vodi računa o daljem životnom ciklusu prototip-zrna. Programer mora voditi računa o tome da ih na kraju počisti.

3.2.2.8.3 Singlton zrna koji zavise od prototip-zrna

U većini slučajeva zrna su uglavnom definisana kao singlton, što je za dohvatanje objekata iz baze jedina dobra praksa. Međutim, dešava se i da singlton zrno u nekom trenutku treba da "sarađuje" (*collaborate*) sa nekim prototip-zrnom. U slučaju da singlton zrno treba da zavisi od nekog prototip-zrna, stvari mogu izgledati malo komplikovanije jer im se ne preklapaju područja definisanosti. Treba nekako reći singltonu koji mu tačno prototip treba, a njih može biti instanciran jako veliki broj. U tom slučaju se koristi takozvano "umetanje metoda" (*Method Injection*) [1].

3.2.2.9 Uvod u ORM (objektno relaciono preslikavanje)

Tehnologija Spring podržava integraciju sa alatima za upravljanje resursima kao što su Hibernate, Java Persistence API (JPA), Java Data Objects (JDO) i iBATIS SQL Maps. Spring nosi značajna poboljšanja u ORM sloju kada se kreiraju aplikacije za pristup podacima. ORM podrška se može koristiti i kao biblioteka, bez obzira na tehnologiju, jer je sve dizajnirano da se upotrebi kao ponovo koristivo java zrno. ORM u Springovom IoC kontejneru olakšava konfigurisanje i razvoj. Zbog toga će u ovom delu rada biti prikazani neki primeri konfigurisanja unutar Springovog kontejnera.

Prednosti koje dobijamo korišćenjem Springovog okruženja za kreiranje ORM pristupa podacima (DAO - data access objects) su [1]:

- Lakše testiranje: Springov IoC pristup olakšava menjanje lokacije implementacija i konfiguracija Hibernate SessionFactory objekata, JDBC DataSource objekata, upravljanje transakcijama i implementacije mapiranih objekata ukoliko je potrebno. Ova mogućnost znatno olakšava testiranje.

- Uobičajeni izuzeci pri pristupu podacima: Spring može da upakuje izuzetke iz ORM alata i transformiše ih u standardnu *DataAccessException* hijerarhiju. Ova rutina omogućava rukovanje najčešćim izuzecima bez korišćenja opštenamenskih deklaracija o izuzecima (*catch, throw*). I dalje je moguće standardno rukovanje izuzecima ako je potrebno.

- Sveobuhvatno upravljanje resursima: Spring aplikacija konteksta može da upravlja lokacijom i konfiguracijom *Hibernate SessionFactory* objekata, *JPA*

EntityMenagerFactory objektima, *JDBC DataSource* objekata, *iBATIS SQL Maps* konfigurisanim objektima i drugim povezanim resursima. Ovo omogućava lakše menjanje i upravljanje vrednostima. Spring nudi efikasno, lako upravljanje trajnim resursima.

- Integracija upravljanja transakcijama.

Glavni cilj ORM integracije je: odvajanje slojeva bez pristupa podacima. Nema više zavisnosti servisa od pristupa podacima ili tehnologije koja obavlja transakcije. Ovo je jedan jednostavan i dosledan pristup za pisanje koda odnosno kreiranje aplikacije. Takvim pristupom pravimo objekte koji mogu biti ponovno iskoristivi i potpuno nezavisni od kontejnera zavisnosti.

3.2.2.9.1 iBatis SQL preslikavanje

iBatis podrška u okruženju Spring u mnogome liči na podršku JDBC-a. Obe tehnologije podržavaju isti šablonski stil programiranja i rad sa hijerarhijom izuzetaka.

Upravljanje transakcijama se vrši Springovim standardnim postrojenjima. Kod iBATIS-a dodatnih strategija transakcija iz razloga što nikakvi transakcioni resursi nisu korišćeni, osim *JDBC Connection*. Stoga su Springovi JDBC *DataSourceTransactionMenager* ili *JtaTransactionManager* sasvim dovoljni.

Kada koristimo *iBATIS SQL* preslikavanje, prvo što moramo uraditi jeste da napravimo *SQLMap* konfiguraciju koja sadrži naredbe i rezultujuću mapu. Spring se onda o njima brine koristeći *SqlMapClientFactoryBean*. Za pravljenje primera prvo koristimo klasu Skale:

```
public class Skale {

    private Integer id;

    private String medijtip;

    private Integer klijent;

    private Integer grupisanje;

    private BigDecimal minvrednost;

    private BigDecimal maxvrednost;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getMedijtip() {
        return medijtip;
    }
```

```
    public void setMedijtip(String medijtip) {
        this.medijtip = medijtip;
    }

    public Integer getKlijent() {
        return klijent;
    }

    public void setKlijent(Integer klijent) {
        this.klijent = klijent;
    }

    public Integer getGrupisanje() {
        return grupisanje;
    }

    public void setGrupisanje(Integer grupisanje) {
        this.grupisanje = grupisanje;
    }


    public BigDecimal getMinvrednost() {
        return minvrednost;
    }

    public void setMinvrednost(BigDecimal minvrednost) {
        this.minvrednost = minvrednost;
    }

    public BigDecimal getMaxvrednost() {
        return maxvrednost;
    }

    public void setMaxvrednost(BigDecimal maxvrednost) {
        this.maxvrednost = maxvrednost;
    }

}
```

Primer 6a: Klasa koju koristimo za preslikavanje

Da bismo mapirali Skale klasu sa iBATIS-om treba napraviti sledeću SQL mapu Account.xml

```
<mapper namespace="asw.iis.media.dao.SkaleDAO">
       <select id="selectOne" parameterType="int" resultType="Skale">
              select *
              from skale
              where id = #{id}
       </select>
       <insert id="insert" parameterType="Skale">
              <selectKey keyProperty="id" resultType="int" order="BEFORE">
                     select skale_id_seq.nextval from dual
              </selectKey>
              insert into skale
              (
              id,
              medijtip,
              klijent,
              grupisanjea,
              grupisanjeb,
              grupisanjec,
              minvrednost,
              maxvrednost
              )
              values
              (
              #{id},
              #{medijtip},
              #{klijent},
```

```
            #{grupisanjea},
            #{grupisanjeb},
            #{grupisanjec},
            #{minvrednost},
            #{maxvrednost}
            )
        </insert>
</sqlMap>
```

Konfiguracioni fajl za iBATIS treba da izgleda:

```
<sqlMapConfig>
        <sqlMap resource="asw.iis.media.mybatis.Skale.xml"/>
</sqlMapConfig>
```

Primer 6b: Primer preslikavanja

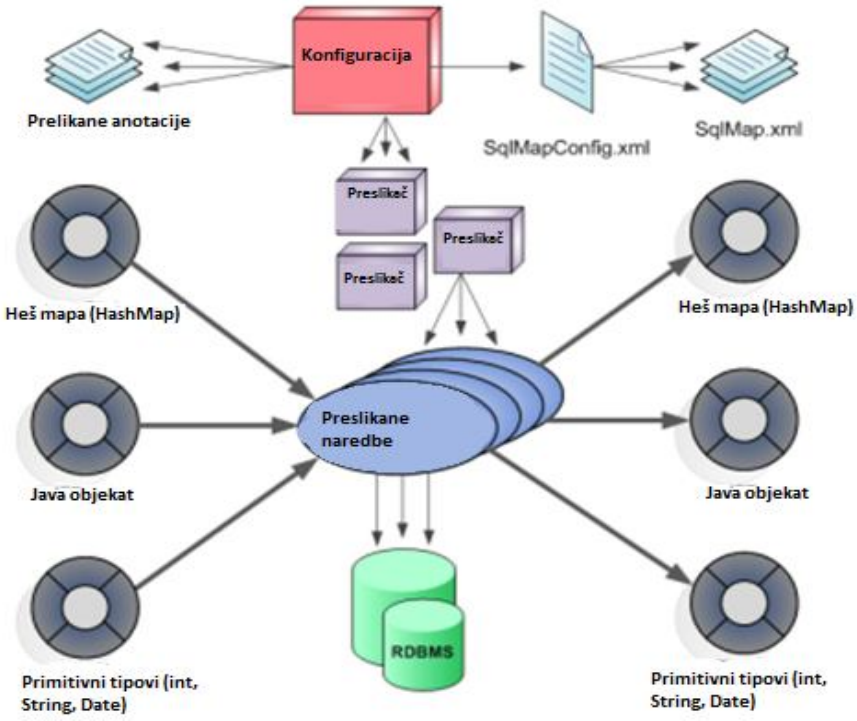
Pri ovome, treba imati u vidu da iBATIS puni resurse koristeći putanju klase, pa se mora dodati Skale.xml fajl u putanju klase.

3.3 MyBatis

MyBatis je naslednik iBatis-a. To je takođe okvir za objektno-relaciono preslikavanje koji SQL upite, naredbe, procedure i funkcije, povezuje sa Java komponentama preko xml deskriptora ili anotacija. Za razliku od drugih alata za objektno-relaciono preslikavanje, on ne preslikava Java objekte na tabelu u bazi podataka, već preslikava metodu na SQL naredbu [7]. Sa MyBatis-om je moguće koristiti sve funkcionalnosti koje nudi SQL. Dobar je za korišćenje u slučajevima gde je potrebna potpuna kontrola nad time šta SQL izvršava.

MyBatis se integriše lako sa okvirom Spring kao i sa okvirom *Google Guice*.

Najveća prednost MyBatis-a u odnosu na druge objektno-relacione okvire je jednostavnost.



Slika 3: Objektno- relaciono preslikavanje: preslikavanje naredbi i iskaza [2]

Svaka MyBatis aplikacija se bazira na instanci SqlSessionFactory. Ona može biti napravljena korišćenjem SqlSessionFactoryBuilder. SqlSessionFactoryBuilder može napraviti SqlSessionFactory instancu iz XML konfiguracionog fajla ili od instance iz Configuration klase.

3.3.1 XML konfiguracioni fajl za preslikavanje

MyBatis XML konfiguracioni fajl sadrži podešavanja i karakteristike koje određuju ponašanje MyBatis-a. Struktura fajla prati sledeće [3]:

1) konfiguraciju
   a) karakteristike
   b) podešavanja
   c) alijasi tipova - typeAliases
   d) vezivači tipova - typeHandlers
   e) fabrika objekata - objectFactory
   f) dodaci - plugins
   g) okruženja
      (1) upravljač transakcijama - transactionManager
      (2) izvor podataka - dataSource
   h) preslikači - mappers

3.3.1.1 Karakteristike

Ovde je reč o spoljno podesivim, promenljivim karakteristikama koje mogu biti konfigurisane u Java Properties fajlu ili prosleđene kao podelementi elementa osobina.

```
<properties resource="classpath*: config.properties">
          <property name="username" value="adm"/>
          <property name="password" value="F2Fa3!33TYyg"/>
</properties>
```

Primer 7: Definisanje karakteristika

U ovom primeru se karakteristike čitaju iz fajla *config.properties,* a pored njih su dodate još karakteristike *username* i *password*.

Osobine u konfiguracionoj datoteci mogu biti postavljene i dinamički. Na primer:

```
<dataSource type="POOLED">
      <property name="driver" value="${driver}"/>
```

```
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
</dataSource>
```

Primer 8: Dinamička konfiguracija karakteristika

Username i password karakteristike će biti zamenjene vrednostima postavljenim u elementu osobina. Driver i url karakteristike će biti zamenjene vrednostima koje se nalaze u config.properties fajlu.

3.3.1.2 Podešavanja (settings)

Ovo stavka modifikuje ponašanje MyBatis-a u vreme izvršavanja. Ta ponašanja se definišu stavkama kao što su: cacheEnabled (omogućavanje keširanja), lazyLoadingEnabled (omogućava ili onemogućanje lenjo punjenje), multipleResultSetEnabled (da li jedan upit može da vrati više skupova rezultata) itd.

```
<settings>
        <setting name="cacheEnabled" value="false" />
        <setting name="jdbcTypeForNull" value="NULL" />
</settings>
```

Primer 9: Definisanje podešavanja

3.3.1.3 Alijasi tipova (type aliases)

Alijasi tipova su samo kraće ime za tip Java objekta. Bitni su samo za XML konfigurisanje i postoje samo da bi smanjilo često kucanje celih imena klasa. Dovoljno je navesti paket čije klase ćemo koristiti preko svojih alijasa. Primer:

```
<typeAliases>
        <package name="asw.iis.common.model" />
        <package name="asw.iis.commonapp.model" />
        <package name="asw.iis.adm.model" />
        <package name="asw.iis.media.model" />
</typeAliases>
```

Primer 10: Definisanje alijasa tipova

Sa ovakvom konfiguracijom klasa iz paketa asw.iis.media.model može biti korišćena svuda svojim imenom, a da ne mora da se navodi cela putanja. Postoji mnogo ugrađenih alijasa tipova za obične Java tipove.

3.3.1.4 Rukovaoci tipovima (typeHandlers)

Kadgod MyBatis podesi neki parametar u *PreparedStatement*-u ili povuče vrednost iz *ResultSet*-a, TypeHandler povlači vrednost tako da ona bude odgovarajućeg Java tipa.

Možemo predefinisati rukovanje tipom da bismo stvorili odgovarajuću vezu sa nepodržanim ili nestandardnim tipovima. Da bi se to uradilo, potrebno je jednostavno implementirati *TypeHandler* interfejs i mapirati novu *TypeHandler* klasu sa Java tipom i opcionalno *JDBC* tipom. Primeri:

```
public class ColorTypeHandler implements TypeHandlerCallback {
	public Object getResult(ResultGetter getter) throws SQLException {
		int value = getter.getInt();
		if (getter.wasNull()) {
			return null;
		}
		Color color = Color.getInstance(value);

		return color;
	}

	public void setParameter(ParameterSetter setter, Object parameter)
			throws SQLException {
		if (parameter == null) {
			setter.setNull(Types.INTEGER);
		} else {
			Color color = (Color) parameter;
			setter.setInt(color.getId());
		}
	}

	public Object valueOf(String s) {
		return s;
	}
}
```

Primer 11: Klasa **ColorTypeHandler.java**

```
<typeHandlers>
	<typeHandler javaType="asw.example.Color"
		handler="asw.mybatis.example.ColorTypeHandler " />
</typeHandlers>
```

Primer 12: XML konfiguracija

3.3.1.5 Fabrika objekata (objectFactory)

Svaki put kada MyBatis pravi novu instancu, on koristi *ObjectFactory*. *ObjectFactory* instancira ciljnu klasu podrazumevanim konstruktorom, ili konstruktorom sa parametrima ako parametar postoji u mapiranju.

3.3.1.6 Okruženje (environment)

MyBatis može biti konfigurisan korišćenjem više okruženja. Ovo nam omogućava da primenimo SQL Maps na više baza podataka iz raznih razloga. Na primer, možemo imati različito konfigurisana okruženja za razvoj, test i produkciju. Ili, možemo imati više rezultujućih baza koje imaju istu šemu pa želimo koristiti isto SQL mapiranje za njih.

Međutim, iako možemo imati konfigurisano više okruženja, možemo izabrati samo jedno za SqlSessionFactory objekat. Stoga, ako želimo da se povežemo na više baza, treba da napravimo onoliko instanci *SqlSessionFactory* klase koliko baza imamo.

Da bismo naveli koje okruženje želimo, možemo samo da to prosledimo kao opcionalni parametar *SqlSessionFactoryBuilder* metodi. Postoje dva potpisa metode build koji prihvataju okruženje.

U okviru Spring objekat *SqlSessionFactory* se instancira kao Spring zrno.

```
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader,
                environment);
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader,
                environment, properties);
```

Primer 13a: Potpisi metoda za izgradnju okruženja

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
	<property name="dataSource" ref="dataSource" />
	<property name="configLocation" value="classpath:properties/mybatis-
config.xml" />
	<property name="mapperLocations">
		<list>
			<value>classpath*:asw/iis/adm/mybatis/*.xml</value>
			<value>classpath*:asw/iis/commonapp/mybatis/*.xml</value>
			<value>classpath*:asw/iis/common/mybatis/*.xml</value>
			<value>classpath*:asw/iis/media/mybatis/*.xml</value>
		</list>
	</property>
</bean>
```

Primer 13b: Spring zrno sqlSessionFactory

Element okruženja (*environments*) definiše konfiguraciju okruženja:

```
<environments default="development">
	<environment id="development">
		<transactionManager type="JDBC">
			<property name="..." value="..." />
		</transactionManager>
		<dataSource type="POOLED">
			<property name="driver" value="${driver}" />
			<property name="url" value="${url}" />
			<property name="username" value="${username}" />
			<property name="passvord" value="${password}" />
		</dataSource>
	</environment>
</environments>
```

Primer 14: Konfiguracija okruženja

3.3.1.7 Upravljač transakcijama (transactionManager)

Postoje dva tipa upravljača transakcijama koja su uključena u MyBatis:

- JDBC-ova konfiguracija koristi direktno JDBC-ove komande potvrđivanja transakcije (*commit*) i poništavanja transakcije (*rollback*).
- MANAGED konfiguracija ne radi skoro ništa. Nikad ne potvrđuje ili poništava vezu. Umesto toga, ona dozvoljava kontejneru da u potpunosti upravlja životnim ciklusom transakcije. Ona podrazumevano zatvara vezu sa bazom.

3.3.1.8 Izvor podataka (dataSource)

Element *dataSource* predstavlja izvorišni objekat. On opisuje tip *JDBC* konekcije na bazu.

Postoje tri ugrađena dataSource tipa: UNPOOLED, POOLED i JNDI implementacije.

Implementacija *unpooled* tipa samo otvara i zatvara konekciju svaki put kada je ona potrebna. Iako je malo sporija, ova varijanta je dobar izbor za prostije aplikacije koje ne zahtevaju odmah dostupne veze. *Unpooled dataSource* se konfiguriše preko pet osobina:

- driver
- url
- username
- password
- defaulTransactionIsolationLevel

Opciono, osobine se mogu proslediti drajveru baze podataka. Da bi se to uradilo, treba dodati prefiks “driver.” na osobinu, na primer:

```
driver.encoding=UTF8
```

Ovo će proslediti osobinu kodiranja sa vrednošću “UTF8” drajveru baze podataka preko DriverManager.getConnection(url, driverProperties) metoda.

Implementacija *pooled* tipa povlači JDBC objekte veze tako da izbegnu inicijalno povezivanje sa bazom i vreme autentikacije pri pravljenju novog objekta veze. Ovo je popularan način za dobijanje brzog odgovora kod konkurentnih veb aplikacija.

Implementacija *JNDI* tipa se koristri sa kontejnerima Springa koji mogu da konfigurišu izvor podataka (*DataSource*).

3.3.2 XML datoteke za preslikavanje SQL upita i komandi

Srž MyBatis-ove moći je u njegovim preslikanim naredbama. Ovo je najbitniji deo MyBatis-a. I pored toga što je ta moć velika, XML fajlovi za SQL preslikavanje su relativno jednostavni. Ako se oni uporede sa ekvivalentnim JDBC kodom koji u Java klasama implementira iste stvari, može se odmah primetiti velika ušteda u pisanju koda.

Java ne obezbeđuje nikakve pogodnosti za automatsko otkrivanje pozicije fajlova za preslikavanje, pa moramo uputiti MzBatis gde da nađe ove fajlove. Pri ovome možemo koristiti relativne reference resursa, ili potpuno odgovarajuće url reference. Na primer:

```
<property name="mapperLocations">
	<list>
		<value>classpath*:asw/iis/adm/mybatis/*.xml</value>
		<value>classpath*:asw/iis/commonapp/mybatis/*.xml</value>
		<value>classpath*:asw/iis/common/mybatis/*.xml</value>
		<value>classpath*:asw/iis/media/mybatis/*.xml</value>
	</list>
</property>
```

Primer 15: Definicije preslikača

Ove naredbe samo kažu MyBatis-u kuda da ide odavde. Ostali detalji se nalaze u SQL mapirajućim fajlovima.

XMl fajl za preslikavanje ima nekoliko elemenata [3]:

- cache - konfiguracija keša za određeno područje imena (*namespace*)
- cache-ref - referenca na konfiguraciju keša iz drugog područja imena
- resultMap - najkomplikovaniji i najmoćniji element koji opisuje kako da se pune objekti iz skupa rezultata iz baze podataka
- sql - ponovno iskoristivo SQL kod na koji mogu se referenciraju druge naredbe
- insert - preslikana insert naredba
- update - preslikana update naredba

- delete - preslikana delete naredba

- select- preslikana select naredba

Možda je najveća mana MyBatis-a što ne rezultat ne može nikako biti generički tip u Javi. Uvek se kao *resultType* mora postaviti neko zrno takvo da kolone navedene u SQL *select* delu u tom zrnu imaju odgovarajuće *set* metode. Ukoliko ne možemo takvo zrno postaviti kao *resultType* onda moramo staviti `resultType="hashmap"` i onda dalje obrađivati dobijeni rezultat.

4 Realizacija projekta

Projekat Mediaplan je nastao po uzoru na veću aplikaciju **asw:dominus** koja je glavni proizvod tima za razvoj ERP[1] rešenja.

Osnovni cilj projektnog zadatka je da se, nakon realizacije jedinstvenog šifarskog sistema i uvoznika, automatskog sistema za uvoz u produkcionu bazu za izveštavanje, dobiju svi podaci neophodni za dobijanje mesečnih izveštaja u kompaniji Multikom.

4.1 Zadaci

- **Šifarnici** - Kreiranje jedinstvenog šifarskog sistema. Osnovna uloga šifarskog sistema je da obezbedi konzistenciju podataka iz različitih izvora koji se koriste u izveštavanju.
- **ETL[2] - Import** - Mehanizam za prikupljanje podataka iz različitih izvora koje firma dobija spolja i proveru podataka sa šifarskim sistemom.

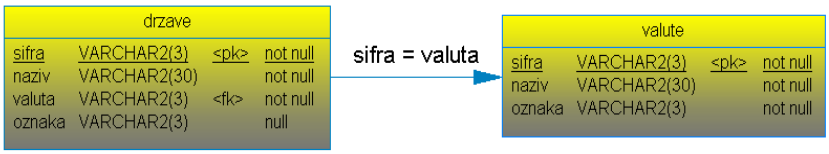
[1] ERP (Enterprise resource planning) su sistemi koji objedinjuju sve poslovne procese jedne firme u jedan sistem

[2] ETL je skraćenica od Extract, Transform and Load (dovuci, obradi i učitaj). Odnosi se na način skladištenja podataka. Podaci se dovlače spolja, vrši se njihova obrada i na kraju skladištenje.

4.1.1 Šifarnici

Analizom sistema, zaključilo se da su potrebni šifarnici koji bi sadržali šifre za sledeće pojmove:

- Države
- Organizacione jedinice
- Korisnici
- Valute, vrsta kursa, kursna lista
- Komitenti, Tipovi komitenata
- Vrsta klijenata
- Posrednici i klijenti
- Grupisanje komitenata
- Naručilac
- Naručioci i klijenti
- Tipovi medija
- Podmediji
- Grupisanje podmedija
- Tip prodaje
- Tip usluge
- Skale
- **Države** - Ovaj šifarnik sadrži nazive država sa kojima se posluje i odgovarajuće šifre.
- **Organizacione jedinice** - Osnovna namena šifarnika organizacionih jedinica je da se podrži postojeća organizaciona šema u firmama po službama i organizacionim jedinicama. U procesu realizacije ovog projekta ovo je neophodna tabela da bi se u njoj evidentirale sve firme čiji elementi ulaze u izveštavanja. Tu ulaze sve firme koje posluju u okviru holdinga Multikom, kao i sve firme saradnici.
- **Valute, vrsta kursa, kursna lista** - U ovom šifarniku nalaze se valute sa vrednostima kursa i hronologijom promena kursa.

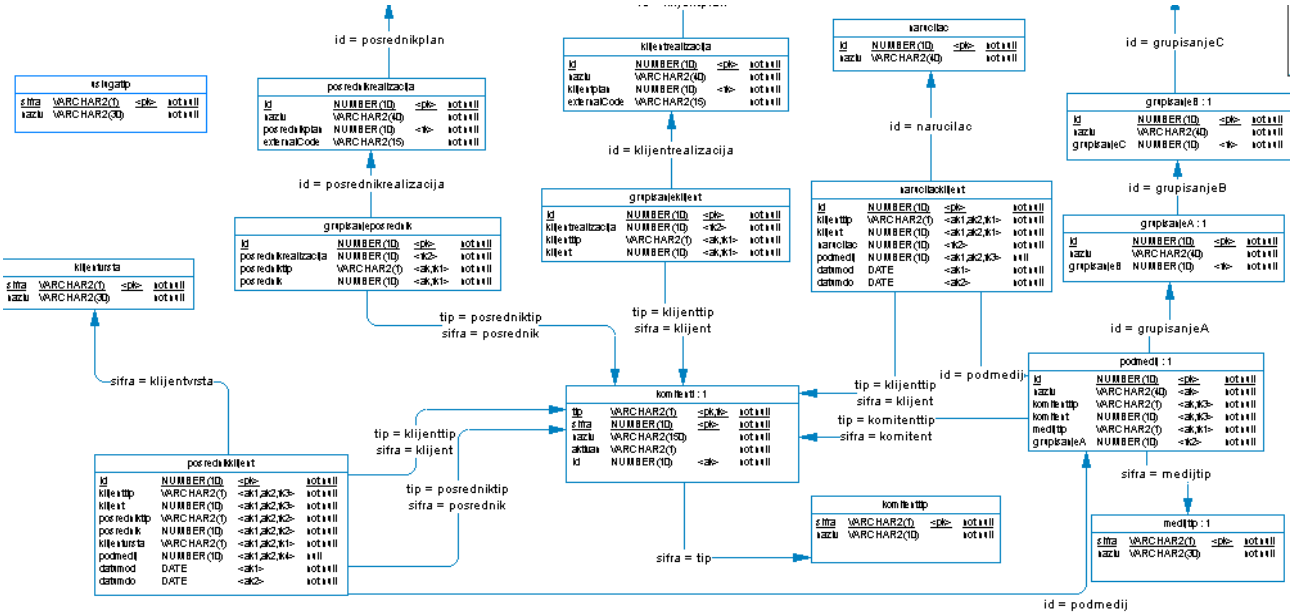


Slika4: Šifarnici država i valuta

- **Korisnici** - Osnovna namena šifarnika je da se obezbede podaci o planeru kao krajnjem korisniku sistema. Ti podaci sadrže njegovo ime, korisničko ime i lozinku.
- **Tipovi komitenata** - Osnovna namena ovog šifarnika je da se komitenti podele po tipovima (Posrednici, Klijenti i Dobavljači).
- **Komitenti** - Ovaj šifarnik služi za evidenciju svih komitenata Klijenata, Posrednika i Dobavljača.
- **Vrsta klijenta** - Ovaj šifarnik služi da se za klijenta unese podatak koji klasifikuje istog kao direktnog klijenta ili klijenta agencije.
- **Posrednici i klijenti** – Vezivna tabela za evidenciju odnosa između klijenta i posrednika, kao delova jedinstvenog šifarnika komitenata i vrstu veze u periodu.
- **Grupisanje komitenata -** Osnovna namena ovih šifarnika je kreiranje grupa klijenata i zakupaca i povezivanje sa komitentima na nivou plana i realizacije.
- **Naručilac** – Evidencija grupe klijenata sa istim godišnjim uslovima.
- **Naručioci i klijenti** – Vezivna tabela za evidenciju odnosa između klijenata i naručioca u periodu.
- **Tipovi medija** – Tipovi medija mogu biti tv, internet, štampani mediji, radio, ...
- **Podmediji -** Osnovna namena šifarnika je evidencija medijskih kuća(npr. RTS1), emitera i njihovu vezu sa nadređenim Medijem (npr. RTS) i tipovima medija (tv, internet, radio...).
- **Grupisanje podmedija** - Osnovna namena ovih šifarnika je kreiranje grupa podmedija. Zbog 3 vrste grupisanja uvedena su 3 šifarnika: Grupisanje A, Grupisanje B i Grupisanje C.
- **Tip prodaje -** Ovaj šifarnik služi da se definiše tip prodajnog resursa (redovna kampanja, sponzorstvo)
- **Tip usluge -** Ovaj šifarnik služi da se definiše tip usluge (ŠTAMPA ili OOH[3]).

[3] OOH je skraćenica za Out-of-home advertising. Odnosi se na tipove reklamiranja kao što su bilbordi, viseće reklame na zgradama, tržnim centrima,...

- **Skale** – Šifarnik skala za različite tipove medija. Služi da definiše najmanje i najveće vrednosti realizacije klijenata na podmedijima.



Slika 5: Šifarnici sistema za medijsko planiranje slika iz programskog paketa Sybase Power Designer u kojem je napravljena šema baze

4.1.2 ETL - uvoznici

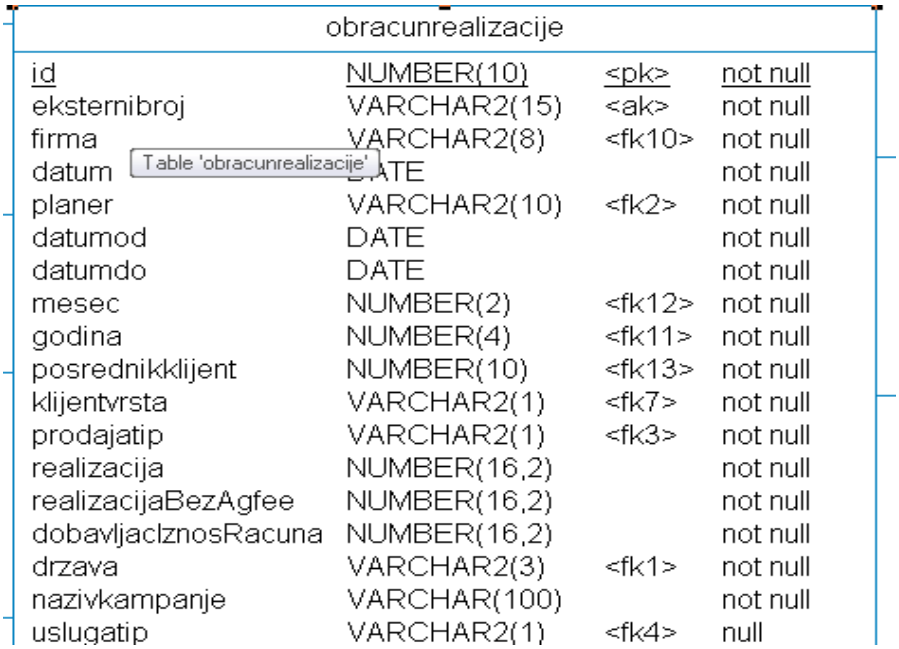
Uvoz podataka se vrši sakupljanjem informacija iz izvora podataka koji su, u ovom slučaju, određeni dokumenti koje klijenti koriste u svom radu.

Izvori podataka su grupisani u sledeće celine:

- Obračun realizacije
- Godišnji plan
- Nuđenje
- Plan media menadžera
- Arhiva

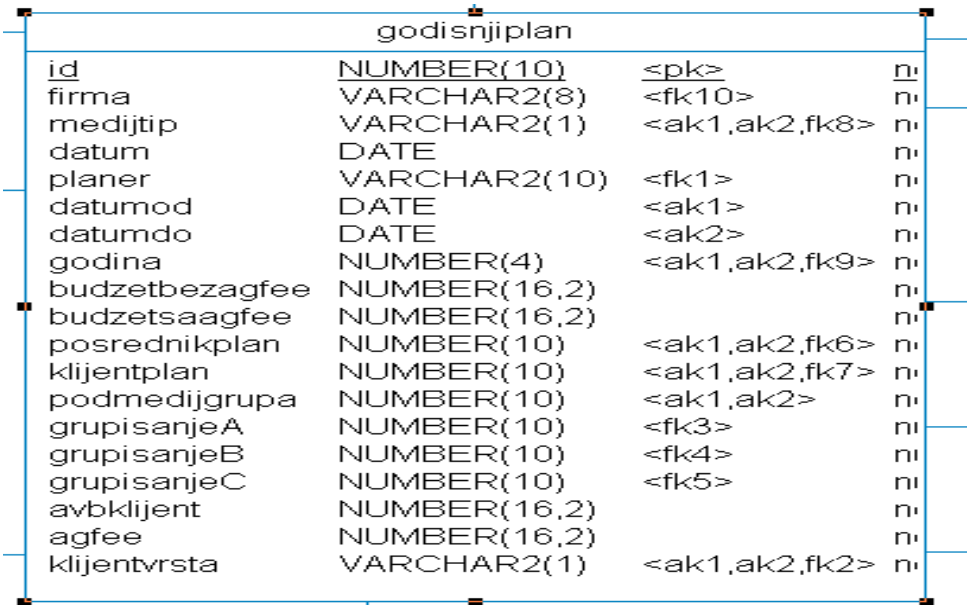
Uvoz se vrši obradom tih podataka i direktnim upisom u bazu u tabele koje se zovu isto kao i nabrojani izvori podataka.

Preduslov za uvoz podataka je da korisnici dostave csv fajlove sa kolonama čiji je redosled određen konfiguracijom sistema za uvoz podataka.



Slika 5: Tabela obračun realizacije

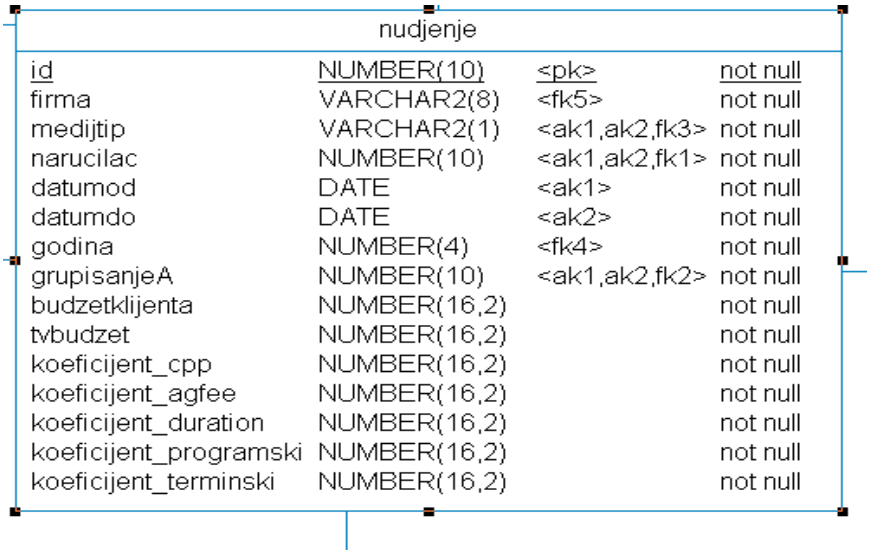
Godišnji plan je realizovan kao tabela koja služi za povezivanje sa postojećim softverskim rešenjem **Plan Real**[4] u koji se unose godišnji budžeti za grupe klijenata.



Slika 6: Tabela godišnji plan

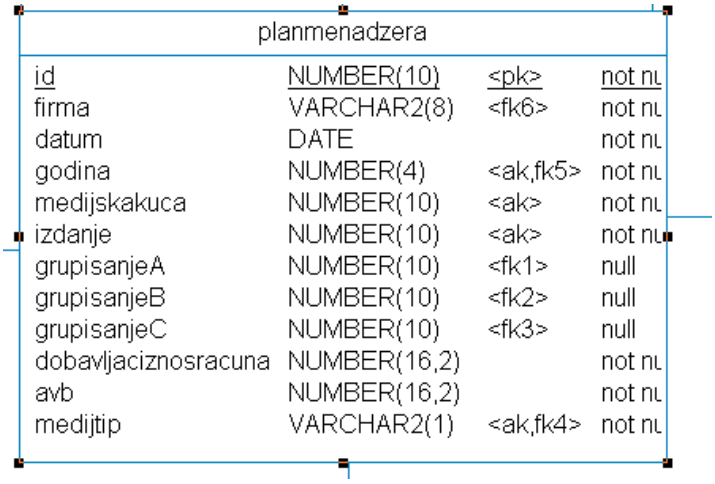
Nuđenje je tabela za Excel sa podacima iz procesa ugovaranja i definisanja uslova sa klijentima i dobavljačima na tv mediju.

[4] Inpress i Plan Real su softverska rešenja drugih firmi koje reklamni stručnjaci iz Multikoma koriste svakodnevno u svom radu.



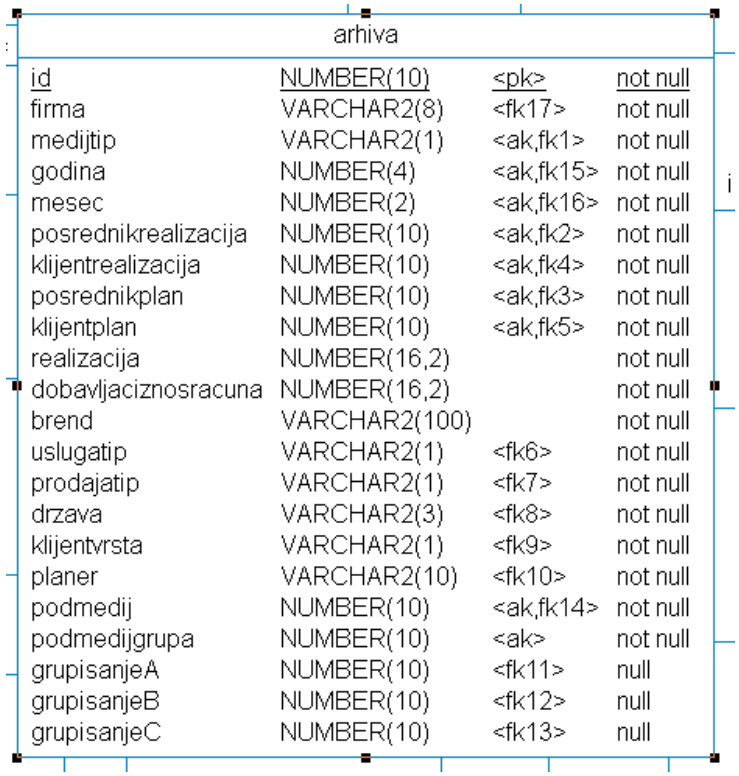
Slika 7: Tabela nuđenje

Plan media menadžera je tabela za Excel sa procenama media menadžera.



Slika 8: Tabela plan menadžera

Arhiva je tabela za Excel sa podacima iz ranijih godina, počevši od 2010. godine.



Slika 9: Tabela arhiva

4.2 Stari način rada

Centralni deo stare verzije projekta Mediaplan je bila Data klasa (Data.java). Sve ostale klase koje predstavljaju entitete iz baze su nasleđivale Data klasu, koja je apstraktna. Ona je, čitajući meta podatke o tabelama, skupljala polja i punila atribut ASWDataFieldsCollection. Prilikom instanciranja objekta koji nasleđuje Data klasu, punila se ta kolekcija odgovarajućim vrednostima i na osnovu tih vrednosti vršio se upis u bazu.

Data klasa je osnovni deo sloja II koji čine Java klase.

Povezivanje sa bazom, radi izvršavanja osnovnih operacija kao što su insert, update ili delete, ili izvršavanje nekih složenijih select upita, se vršilo uz pomoć JDBC-ovog PreparedStatement-a.

```
PreparedStatement stmt;
       stmt = conn.prepareStatement
                    ("select count(*) from narucilacklijent"
                                 + "    where klijenttip = ? and klijent = ? "
                                        + " and not (? < datumod or ? > datumdo)
                                        and podmedij is null  ");
       stmt.setString(1, fields.getStringValue("klijenttip"));
       stmt.setInt(2, fields.getIntegerValue("klijent"));
```

```
        stmt.setDate(3, fields.getDateValue("datumdo"));
        stmt.setDate(4, fields.getDateValue("datumod"));
        ResultSet rts = stmt.executeQuery();
        stmt.close();
```
Primer 16: Rad sa PreparedStatement klasom iz JDBC-a

Sloj III čine JSP stranice. U ovoj aplikaciji postoje 2 vrste jsp strana. Stranice koje predstavljaju listu i one koje prestavljaju detalj. Na primer, obracunrealizacijeLista.jsp je strana koja prikazuje spisak svih dokumenata obračuna realizacije, dok detalj strana (obracunrealizacije.jsp) daje informacije o jednom konkretnom objektu.

Najvažnija šifarnička tabela je tabela komitenata. U njoj su sadržani najosnovniji podaci o kompanijama sa kojima se sarađuje. Postoje 3 tipa komitenata: klijent - komitent koji kupuje neke usluge od firme, posrednik - komitent posredstvom koga se postiže ugovoren posao sa klijentom (preko koga se vrši reklamiranje klijenta i dobavljač – komitent koji služi za nabavku osnovnih sredstava potrebnih za rad firme. Komitent može imati stanje aktivan i neaktivan. Neaktivni komitenti se čuvaju da bi se sačuvali podaci o ranijoj saradnji i postoji mogućnost ponovnog aktiviranja ukoliko se saradnja obnovi.

Prvobitni način rada, koji se oslanjao na tehnologiju JDBC, je bio takav da su se svi entiteti predstavljani svojim Data klasama (npr. NarucilacData.java, PodmedijData.java…) i odgovarajućim jsp stranama. Na jsp stranama su se instancirali objekti Data klasa kojima su se vršile osnovne operacije (unos novog, izmena, brisanje). Data klase su sadržale svu logiku. Pomoću njih se vršio upis u bazu, u njima su definisani upiti za složeniju obradu podataka iz baze i validacija podataka pre samog upisa u bazu.

4.3 Novi način rada (Spring)

Prevođenje aplikacije iz starog stanja u Spring MVC aplikaciju zahteva 3 koraka:

- biranje perzistentnog okruženja za preslikavanje sql upita
- prevođenje klasa u Spring, tj. pravljenje Java zrna po uzoru na stare klase, ali uz potpunu podršku Spring načina rada, korišćenjem ApplicationContext klase i konfiguracionog xml fajla applicationContext.xml
- izbacivanje jsp strana i uvođenje nove tehnologije koja će biti kompatibilna sa Spring-om

Treći korak nije tema ovog rada i on je još uvek u pripremnoj fazi. Deo sa jsp stranama je ostao isti i prilagođen je slojevima koji se nalaze ispod prezentacionog tako što koristi klasu WebApplicationContext za čitanje Spring zrna definisanih u konfiguracionom xml fajlu.

Među rešenjima koja treba da zamene dosadašnji način rada sa JDBC-om, vršilo se istraživanje šta je najpouzdanije, najpopularnije i najkorišćenije u poznatim svetskim kompanijama. Svakako se kao najpopularnija tehnologija izdvojio Hibernate ali većina aplikacija u firmi ima deo poslovne logike implementiran u uskladištenim procedurama, što nije moguće realizovati u Hibernate-u. Sledeće rešenje koje se nudilo, a koje je lako prilagodljivo postojećem stanju je MyBatis. Prvi deo prelaska se odvijao tako što su svi upiti koji su bili definisani u java klasama, korišćenjem *PreparedStatement* klase izbačeni iz njih i prebačeni u Mybatis xml fajlove i iz njih čitani. Nakon toga se krenulo korak dalje. Sve klase, koje su nasleđivale već pominjanu Data klasu, su reorganizovane.

4.3.1 Strukturna organizacija projekta

Prvobitni način rada se oslanjao na tehnologiju JDBC i bio je takav da su se svi entiteti predstavljali svojim Data klasama (npr. *NarucilacData.java*, *PodmedijData.java*…) i odgovarajućim jsp stranama. Na jsp stranama su se instancirali objekti Data klasa kojima su se vršile osnovne operacije (unos novog, izmena, brisanje). Data klase su sadržale svu logiku. U njima se vršio upis u bazu, u njima su definisani upiti za složeniju obradu podataka iz baze i validacija podataka pre samog upisa u bazu.

Prelaskom na tehnologije Spring i MyBatis došlo je do raslojavanja i sledeće podele:

- Zrno predstavlja model. U njemu se čuvaju podaci o kolonama tabele koja predstavlja entitet. Svako model-zrno nasleđuje klasu BasicModel. BasicModel ima tri apstraktne metode: getTableName (implementacija ove metode vraća string koji je naziv tabele predstavljene zrnom), getPrimaryKey (ova metoda nam vraća atribut koji odgovara primarnom ključu te tabele). Svako model-zrno, pored navedene tri metode, sadrži još samo get i set metode i nijednu drugu metodu čiji bi cilj bio dalja obrada podataka tog entiteta.
- DAO interfejsi. Svaki DAO interfejs nasleđuje BasicDAO interfejs. BasicDAO interfejs sadrži potpis metoda: *insert(BasicModel dataBean)*, *update(BasicModel dataBean)*, *delete(BasicModel dataBean)*. Interfejsi koji nasleđuju DAO već sadrže ta tri potpisa, a pored njih se u tom interfejsu mogu pobrojati i druge metode koje će vršiti neku SQL select operaciju nad bazom podataka. Na primer, metod list<BasicModel> selectAll (vraća listu svih entiteta koji su u bazi i njoj odgovara SQL naredba SELECT * FROM *ENTITET*;).
- XML datoteke MyBatis-a. Koristeći opisani mehanizam MyBatis-ovog preslikavanja ove datoteke sadrže definiciju Java Objekta čije će metode preslikati (u našem

slučaju DAO interfejse) i same SQL upite i naredbe koje odgovaraju metodama DAO interfejsa (insert, update, delete i razne složenije upite koji na neki način obrđuju podatke).

- Servisni interfejsi. Svaki servisni interfejs nasleđuje BasicService interfejs. BasicService i njegova implemetaciona klasa BasicServiceImpl opisuju osnovni rad sa nekim od entiteta. U osnovi su opet insert, update i delete metode, jer nam insert iz DAO interfejsa nije dovoljan. Pre nego što se insert izvrši, rade se sledeće operacije: validate() i track(). Validate vrši proveru ispravnosti entiteta koji će biti upisan, tj. proverava da li će unos tog entiteta narušiti stanje u bazi. Track metoda služi za interno praćenje korisničkih operacija. Naime, postoje tabele u bazi koje se zovu tracktabele i track istorija koje sadrže podatke o tome kad je unet neki podatak i sa koje IP adrese. Tek kada se ove operacije izvrše, radi se operacija nad tom tabelom (insert, update, delete), drugim rečima servisni interfejsi i klase koje ih implementiraju su "oblanda" nad DAO operacijama.
- Implementacione klase. Njihov cilj je da implenetiraju gore navedene interfejse.
- Validatori. Validatori su klase čije metode proveravaju da li će unos novog, odnosno izmena postojećeg entiteta narušiti logiku rada aplikacije. Svaka validator klasa implemetira BasicValidator interfejs. On sadrži samo jednu metodu validateInsertUpdate(). U opisu servisnih klasa je objašnjeno da se validacija radi pre samog unosa u bazu. Validacija je ništa drugo nego poziv pomenute metode validateInsertUpdate iz odgovarajuće klase. Kao primere validacije možemo navesti: provera da li postoje neka preklapanja datuma takva da recimo jedan klijent sa nekom medijskom kućom ne može imati dva ugovora u istom vremenskom periodu, ili da se ne sme dodeliti nov ugovor sa klijentom koji ima stanje "neaktivan" i sl.
- Uvozne klase. Za entitete koji predstavljaju dokumente medijske agencije (obračun realizacije, godišnji plan, arhiva, nuđenje i plan menadžera) se unos ne radi kroz aplikaciju, već na osnovu Microsoft Exel dokumenata, koji su izgenerisani nekom drugom aplikacijom. Za te dokumente su napravljene import klase koje iz CSV fajla (koji se lako generisše iz Exel dokumenta) radi upis u bazu. Te klase implementiraju postojeće interfejse u okviru ASW sistema koji od jednog reda u CSV fajlu instanciraju objekat tog entiteta i vrše upis u bazu. Pored tih standardnih metoda, vrše se provere da li je neka kolona u CSV fajlu izostavljena. Na primer, u tabeli godišnji plan postoji kolona "mesec" koja se ne šalje CSV fajlom, već se za mesec podrazumeva onaj mesec kojeg je napravljen taj dokument.

Za povezivanje podataka, koji se unose na jsp strani sa podacima o odgovarajućem entitetu (model zrnu), služi zrno BasicController. Ono uz pomoć klase ASWFormDataAccessor prolazi kroz jsp stranu čiji elementi moraju da se zovu isto kao i elementi model zrna i korišćenjem metode set iz te klase postavljaju vrednosti atributa.

Upravljanje transakcijama se vršilo anotacijama na osnovnim metodama za unos, izmenu i brisanje podataka. Sve ovakve metode imaju anotaciju:

```
@Transactional(readOnly = false, propagation = Propagation.REQUIRED, isolation = Isolation.READ_COMMITTED).
```

Ovo znači da ta metoda menja podatke. Deo `propagation = Propagation.REQUIRED` označava da će pozivom te metode biti započeta transakcija nad bazom, a ukoliko je transakcija započeta u nekoj metodi koja poziva ovu metodu onda će metoda o kojoj je reč ući u tu već postojeću transakciju. Deo `isolation = Isolation.READ_COMMITTED` označava nivo izolovanosti. Ovim nivoom se brani čitanje podataka nad kojim postoje nezavršene odnosno ne potvrđene promene.

Korišćenje ovih anotacija je omogućeno dodavanjem opcije `<tx:annotation-driven/>` u xml konfiguracioni fajl.

U narednom primeru se može videti predstavljanje jednog entiteta po navedenim slojevima.

```
public class Narucilac implements Serializable, BasicModel{

    private static final long serialVersionUID = -3411119425289527193L;

    private Integer id;

    private String naziv;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNaziv() {
        return naziv;
    }

    public void setNaziv(String naziv) {
        this.naziv = naziv;
    }

    public Object getPrimaryKey() {
       return getId();
    }

    public String getPrimaryKeyAsString() {
       if (getPrimaryKey()==null) {
              return null;
       }
       return getPrimaryKey().toString();
    }

    public String getTableName() {
       return "narucilac";
    }
}
```

Primer 17: Primer model klase

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
       "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="asw.iis.media.dao.NarucilacDAO">
       <select id="selectOne" parameterType="int" resultType="Narucilac">
              select *
              from narucilac
              where id = #{id}
       </select>
       <insert id="insert" parameterType="Narucilac">
              <selectKey keyProperty="id" resultType="int" order="BEFORE">
                     select nvl(max(id), 0)+1 from narucilac
              </selectKey>
              insert into narucilac
              (
              id,
              naziv
              )
              values
              (
              #{id},
              #{naziv}
              )
       </insert>
       <update id="update" parameterType="Narucilac">
              update narucilac
              set
              naziv = #{naziv}
              where id = #{id}
       </update>
       <delete id="delete" parameterType="int">
              delete from narucilac where id
              = #{id}
       </delete>
</mapper>
```
Primer 18: XML fajl za preslikavanje

```java
public interface NarucilacklijentDAO extends BasicDAO {
       public int selectPreklapanjePeriodaPodmedijNull(Narucilacklijent nk);

       public int selectPreklapanjePeriodaPodmedijNotNull(Narucilacklijent nk);

}
```
Primer 19: DAO interfejs

```java
package asw.iis.media.service;

import asw.iis.common.model.BasicModel;
import asw.iis.common.service.BasicService;
import asw.iis.media.model.Klijentplan;

public interface KlijentplanService extends BasicService<BasicModel> {
       public Klijentplan selectByExternalCode(String externalCode);
}
```
Primer 20: Servisni interfejs

```java
package asw.iis.media.service.impl;

import asw.iis.common.service.impl.BasicServiceImpl;
import asw.iis.media.dao.KlijentplanDAO;
import asw.iis.media.model.Klijentplan;
```

```
import asw.iis.media.service.KlijentplanService;

public class KlijentplanServiceImpl extends BasicServiceImpl implements
		KlijentplanService {

	public Klijentplan selectByExternalCode(String externalCode) {
		Klijentplan klijentPlan = null;
		if (externalCode != null) {
			klijentPlan = ((KlijentplanDAO) getDao())
					.selectByExternalCode(externalCode);
		}
		return klijentPlan;
	}

}
```
Primer 21: Implementacija servisnog interfejsa

```
package asw.iis.media.validator;


public class PosrednikklijentValidator extends BasicValidator {

	@Autowired
	private PosrednikklijentDAO posrednikklijentDAO;
	public boolean supports(Class clazz) {
		return Posrednikklijent.class.isAssignableFrom(clazz);
	}
	public void validate(Object target, Errors errors) {
		super.validate(target, errors);
		int counter = 0;
		Posrednikklijent posrednikKlijent = (Posrednikklijent)target;
		if (posrednikKlijent.getPodmedij() == null) {
			counter =
posrednikklijentDAO.selectPreklapanjePeriodaPodmedijNull(posrednikKlijent);
			if (counter != 0) {
				errors.reject("preklapanjePerioda");
			}
		} else {
			counter =
posrednikklijentDAO.selectPreklapanjePeriodaPodmedijNotNull(posrednikKlijent);
			if (counter != 0) {
				errors.reject("preklapanjePerioda");
			}
		}
	}
}
```
Primer 22: Klasa za validaciju

Navedeni java objekti su definisani kao spring zrna u *applicationContext.xml* fajlu po sledećem principu:

```
<!-- POSREDNIKKLIJENT -->
<bean id="posrednikklijentDAO" parent="baseMapper">
	<property name="mapperInterface"
value="asw.iis.media.dao.PosrednikklijentDAO" />
</bean>

<bean id="posrednikklijentValidator"
	class="asw.iis.media.validator.PosrednikklijentValidator">
</bean>
```

```
<bean id="posrednikklijentService" parent="basicService"
       class="asw.iis.media.service.impl.PosrednikklijentServiceImpl">
       <property name="dao" ref="posrednikklijentDAO" />
       <property name="validator" ref="posrednikklijentValidator" />
       <property name="dataBeanClass"
       value="asw.iis.media.model.Posrednikklijent"/>
</bean>

<bean id="posrednikklijentController" parent="basicController">
       <property name="service" ref="posrednikklijentService" />
       <property name="dataBeanClass" value="asw.iis.media.model.Posrednikklijent"
/>
</bean>
<!-- END POSREDNIKKLIJENT -->
```
Primer 23: Primer definicije zrna koji predstavljaju jednu tabelu iz baze podataka

4.3.1.1 Prednosti ovakve strukture

Ovakvom strukturom je jasno razdvojena implementacija i način rada java klasa od sistema za upravljanjem bazama podataka. Promena sistema bi uticala samo na promenu xml fajlova iz paketa mybatis, dok bi ostala struktura ostala netaknuta.

Ukoliko dođe do promene perzistentnog okruženja i Mybatis se u budućnosti zameni nekim drugim okruženjem, npr. JPA, Hibernate, takođe bi došlo samo do promene paketa Mybatis. U nekom drugom paketu bi se razvilo povezivanje objekata sa bazom podataka i to po receptu definisanom u DAO interfejsima.

Još jedna od mogućih promena je nadgradnja Jave, ili čak i Spring-a. Ukoliko nekad dođe do potrebe da se radi nekom drugom tehnologijom razvijena je univerzalna i prilagodljiva slojevitost tako da će se menjati samo klase koje implementiraju servisne interfejse iz paketa *service*, dakle samo paket *impl*.

Rad na ovom prevođenju se pokazao korisnim za dalju strategiju razvoja proizvoda u firmi, jer je nakon uspešnog prelaska na ovoj, po obimu manjoj aplikaciji, došlo do rada na prevođenju veće aplikacije asw:dominus na tehnologije Spring i MyBatis.

Ovakva struktura aplikacije je dobra osnova za dalji rad na izboru adekvatne tehnologije za predstavljanje korisničkog interfejsa, jer je to deo koji je najveća slabost ove aplikacije.

5 Korišćenje aplikacije

Aplikativno rešenje zahteva osnovni nivo znanja u Windows okruženju. Sve forme su ekranske.

Pozicioniranjem na bilo koje polje forme ili ikonicu, otvara se, takozvani ***Tool tip*** u kojem se nalazi informacija o atributima u polju ili o akciji koju je potrebno uraditi. Ista ta informacija ispisuje se u levom, donjem uglu svake forme i naziva se ***Status bar***.

Osnovne napomene za rad:

- Pod selekcijom se podrazumeva mesto na kojem je miš fokusiran.
- Kretanje po poljima moguće je korišćenjem miša (pomeranje gore i dole) ili tastature (tipke TAB).
- Levi klik miša (u nastavku samo klik miša) može se ostvariti preko tastature istovremenim pritiskom tipke Alt i podvučenog slova.
- Ikonice se pokreću duplim klikom miša (dvoklik miša) ili istovremenim pritiskom tipke Alt i podvučenog slova.

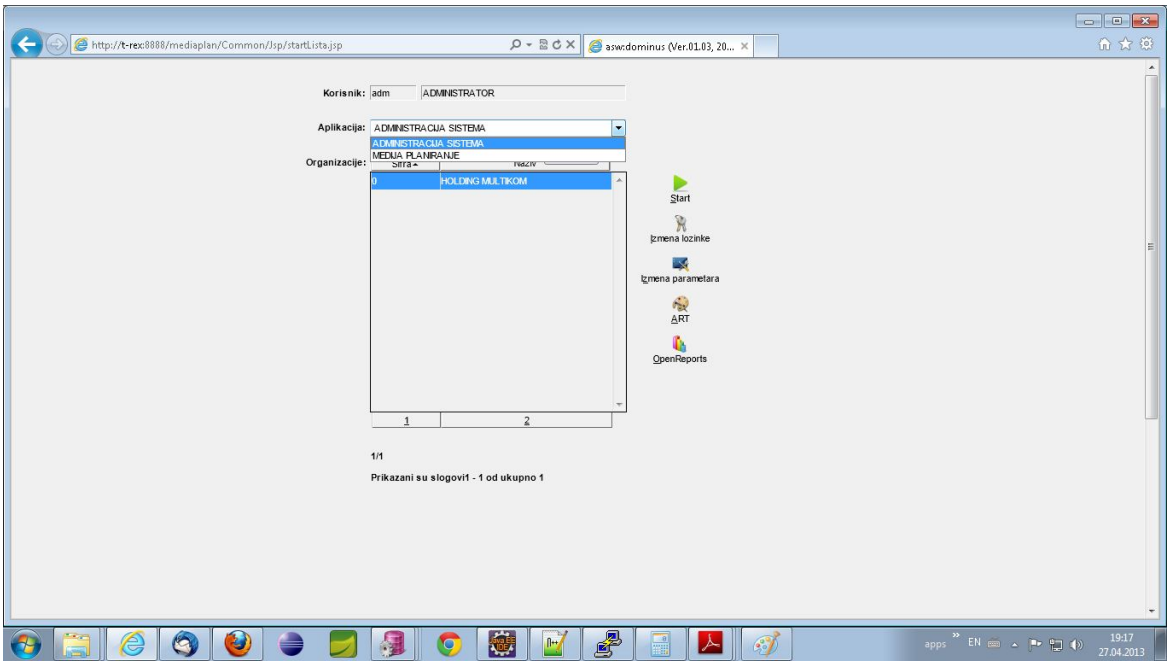
ENTER na tastaturi pokreće naredbu koja se ostvaruje klikom miša.

5.1 Pokretanje aplikacije

Pri ulasku u aplikaciju se pojavljuje standardna forma za login.

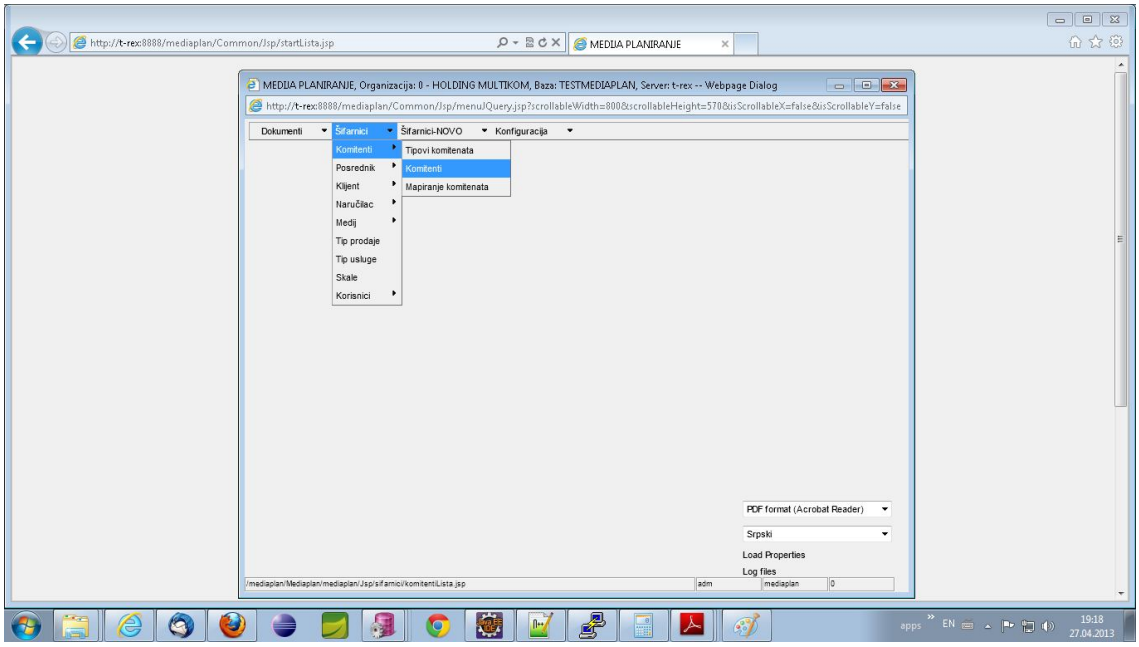
Slika 10: Pokretanje aplikacije

Nakon uspešnog pristupa, korisniku se nudi izbor aplikacije (podsistema) koji će koristiti. Postoje 2 podsistema - podsistem za administraciju, koji krajnji korisnik (marketinški agent) neće koristiti i podsistem za medijsko planiranje koji je tema ovog rada.



Slika 11: Izbor podsistema

Nakon ulaska u željeni podsistem, korisniku se prikazuje meni koji sadrži forme za unos, izmenu i brisanje podataka.



Slika 12: Meni aplikacije

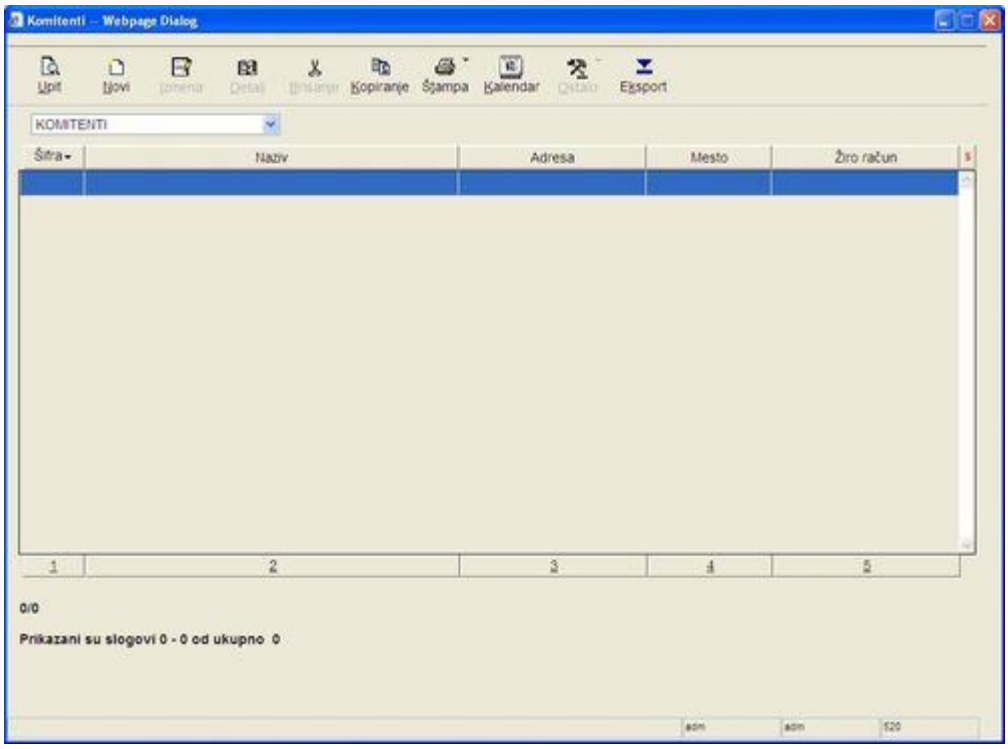
Izborom opcije iz menija otvara se lista koja odgovara podacima iz odgovarajuće tabele.

5.2 Lista

Lista predstavlja grafičku formu za prikaz slogova jedne tabele.

Osnovne komponente liste su:

- Meni liste
- Lista (tabela za prikaz podataka)



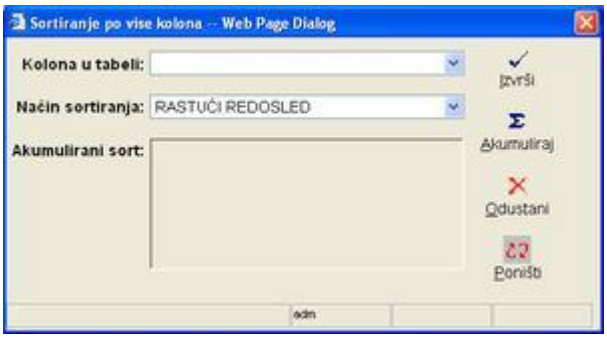
Slika 13: Lista podataka

Meni liste se otvara ako sa desnim klikom miša pozicionira bilo gde na listi.

5.3 Sortiranje

Lista se može sortirati klikom miša na naziv kolone. Na primer, klikom na kolonu Šifra, u prikazanom primeru, lista će biti sortirana po šifri komitenta; klikom na kolonu Naziv lista će biti sortirana po nazivu komitenta. Može se sortirati po rastućem ili opadajućem redosledu.

Sortiranje se vrši po atributu na čiji naziv se fokusira miš (tamo gde se pojavi ▼). Na samoj listi moguće je postaviti sortiranje po jednom ili više atributa klikom na kolonu S , koja omogućava startovanje Sortiranja po više kolona.



Slika 14: Sortiranje liste

Ukoliko se želi postaviti sortiranje po više atributa, npr. šifri i nazivu, potrebno je nakon izbora kolone kliknuti na opciju ***Akumuliraj***, a nakon unosa svih željenih kolona po kojima se želi sortiranje, kliknuti na opciju ***Izvrši***.

5.4 Brojač

Pri dnu svake liste, u levom uglu, za selektovani podatak na listi postoji brojač koji nosi informacije o poziciji dokumenta i ukupnom broju elemenata na listi.

Slika 15: Brojač u listi

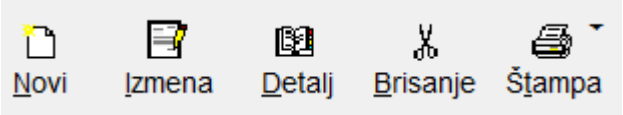
Ukoliko ima više od 50 slogova na listi, na dnu liste sa desne strane se pojavi komponenta koja omogućava prelazak na sledeću stranu liste.

Slika 16: Dugmići za kretanje kroz listu

5.5 Meni liste

Lista predstavlja osnovu za rad u aplikaciji. Pomoću ove forme pokreću se akcije za unos, izmenu, brisanje, pretraživanje i štampu podataka.

Osnovne opcije za rad sa šifarnicima prikazane su na slici



Slika 17: Opcije u listi

Novi – unos novog podatka u šifarnik

Izmena – izmena postojećih podataka

Detalj – pregled unetih podataka

Brisanje – brisanje podataka iz šifarnika

Štampa – štampa podataka iz šifarnika

6 Završna reč

Cilj ovog rada je bio da se predstavi evolucija višeslojnih aplikacija i osavremenjivanje jednog načina rada koji je davao jako dobre rezultate, ali je bio previše složen. Modularnost, i raslojavanje aplikacije po jasno definisanim granicama, predstavljaju jako veliku prednost i podstiču jedan nov način razmišljanja prilikom razvijanja aplikacije. To daje jednu vrstu uzora koji treba pratiti i za razvoj budućih aplikacija, nezavisno od tehnologija koje se koriste.

Sam Spring predstavlja novu vrstu softverskog alata koji omogućava drugačiji način razmisljanja u razvoju aplikacija. Svojim, donekle strogim pravilima, ukida nečitljivost koda i praktično zabranjuje pravljenje klasa za čije tumačenje je potrebno izdvojiti dosta vremena. Svojom strukturom je možda uveo nešto što mnogi programeri ne vole, a to je utapanje u šablone, ali se ne može osporiti uvođenje reda pobrojavanjem šta sve jedna klasa treba da uradi i odvajanjem tog "recepta" od same implementacije, koja jednoga dana može nadmašiti i sam Spring. Kao takav Spring je postao nezaobilazna tehnologija za razvoj Java aplikacija novijeg doba.

7 Literatura

[1] Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Arendsen, A., Risberg, T., Davison, D., Kopylenko, D., Pollack, M., Templier, T., Vervaet, E., Tung, P., Hale, B., Colyer, A., Levis, J. "Spring Java Application Framevork," http://www.springsource.org/ 30/08/2012,

[2] MyBatis Community, "MyBatis-Spring 1.0.0 - Reference Documentation," 01/09/2012,

[3] MyBatis Community, "MyBatis 3 – User Guide," 04/09/2012,

[4] Valls, C., Breidenbach, R.,: "Spring in Action, Manning Publications, 2005,"

[5] Johnson, R.,: "Expert One-on-One J2EE Design and Development, Viley Publishing, 2002,"

[6] Wikipedia, MyBatis "http://en.wikipedia.org/wiki/MyBatis," 17/08/2012,

[7] Wikipedia, Spring "http://en.wikipedia.org/wiki/Spring," 16/08/2012,

[8] Vogel, L. "Dependency injection vith Spring framework – Tutorial, " 25/08/2012