KREIRANJE DINAMIČKIH INTERFEJSA ZASNOVANIH NA META-ŠEMAMA
CREATION OF DYNAMIC INTERFACES BASED ON META-SCHEMES

Vladimir Vujović, Elektrotehnički fakultet Istočno Sarajevo
Branko Perišić, Fakultet Tehničkih Nauka, Novi Sad
Budimir Kovačević, Elektrotehnički fakultet Istočno Sarajevo
Snježana Milinković, Elektrotehnički fakultet Istočno Sarajevo

Sadržaj - *Kreiranje dinamičkih interfejsa, generatora interfejsa ili česte promjene interfejsa bazirane na strukturama i podacima unutar baze podataka, predstavljaju samo neke od slučajeva sa kojima se developer susreće u procesu proizvodnje softvera. U ovom radu je prikazano kako se na osnovu meta-šema baze podataka proces izrade interfejsa može ubrzati i napraviti dinamičnijim. Srž rada je u prepoznavanju struktura podataka unutar tabela, kreiranju autonomnih objekata ili modula za prikaz određene strukture, te logici koja omogučava kreiranje dinamčkih interfejsa. Jednostavnim primjerima i diskusijama prikazuje se mogući način realizacije ovoga rješenja u programskom jeziku JAVA.*

Abstract - *The creation of dynamic interfaces, interface generators or often changes of interface based on structures and data within a database, present only some of the cases that the developer faces with in the software production process. This paper presents a way in which, on the basis of database meta-scheme, the interface making process can be accelerated and made more dynamic. The essence of the paper is in the recognition of data structures within the tables, creation of autonomous objects or modules for displaying a certain structure and the logic that enables the creation of dynamic interfaces. By simple examples and discussions, a possible way of the realization and application of this technology at creating an interface in JAVA is presented.*

1. UVOD

Kreiranje korisničkog interfejsa je posao sa kojim se svakodnevno susreće veliki broj programera. Pri tome, pored samog dizajna interfejsa, problemi na koje se nailazi vezani su i za tehničke detalje. Ovo posebno dolazi do izražaja kada se radi o složenim poslovnim aplikacijama koje u pozadini imaju rad sa nekim od sistema za digitalno čuvanje i obradu podataka kao što su baze podataka, XML fajlovi i dr. Korisnički interfejsi za aplikacije koje rade sa ovakvim izvorima podataka moraju biti u stanju da prikazuju te podatke na neki od poznatih načina, tj. tabelom, poljima za unos i dr. Unificiranjem načina prikazivanja podataka može se riješiti problem prikazivanja istih u svim srodnim sistemima. Na takav način se može postići da jedan interfejs za rad sa bazom podataka, ako je dovoljno fleksibilan i urađen tako da bi korisnik na najlakši način mogao doći do podataka, ima visoki stepen iskorištenja.

Međutim, iako su usvajanje standardizovanog načina prikaza podataka i standardizacija interfejsa od velike pomoći, za sisteme koji se mijenjaju dinamički i dalje ostaje primarni problem a to je veliki broj izmjena u kodu pri projektovanju ovakvih softvera. Da bi se process izmjene koda maksimalno ubrzao i doveo do zadovoljavajućih i optimalnih rješenja (što po korisnika, što po projektanta, odnosno, programera) moraju se uvesti složene analize koje obuhvataju podatke, tipove podataka i njihove specifične karakteristike. Upravo u taj proces se uključuju meta-šeme podataka na osnovu kojih se može dobiti većina, ako ne i svi potrebni podaci kako bi taj zadatak bio ispunjen.

U ovom radu je pokazano kako se na jednostavan način mogu kreirati dinamički interfejsi koji će moći u što kraćem roku opslužiti promjene na strani baza podataka. Uvođenjem dinamičkog kreiranja interfejsa otvaraju se vrata za brzo prototipovanje i brza prilagođavanja samog softvera krajnjem korisniku.

2. KREIRANJE INTERFEJSA

Sa razvojem računarske tehnologije dolazi i do razvoja aplikacija, a samim tim i do povečanog broja korisnika koji postaju sve zahtjevniji, jer računare ne koriste samo u strogo poslovne svrhe, već i za zabavu. Međutim, iako se računari koriste i za zabavu, njihova primarna namjena ostaje obrada podataka, odnosno primjena u poslovnim aplikacijama.

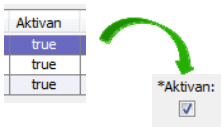
Poslovne aplikacije karakteriše potreba za pouzdanošću i brzom obradom podataka, ali se pored toga zahtjeva i jednostavnost njihovog korišćenja. Pred programere se postavlja ne tako jednostavan zadatak koji glasi: "kako napraviti nešto brzo i efikasno, a da pri tome bude jednostavno i pouzdano?". Upravo pri kreiranju poslovnih aplikacija dolaze do izražaja prednosti koje se postižu korišćenjem unificiranog izgleda korisničkog interfejsa. Unificiranjem interfejsa za svaku od aplikacija može se definisati izgled forme i taj isti izgled primjeniti onoliko puta koliko bude potrebno.

Ako se razvija korisnički interfejs na klasičan način onda se mora krenuti od analize problema koja će dati specifikaciju

rješenja. Dobijanjem detalja o svim entitetima mogu se definisati svi elementi forme. Nedostatak ove metode je u činjenici da programer mora za svaki entitet odlučiti kako će izgledati i kako će se ponašati komponenta koju on predstavlja.

Ako se za čuvanje podataka koristi baza podataka kreiranje interfejsa mora započeti analizom tipova podataka unutar baze, njihovim dozvoljenim ili nedozvoljenim vrijednostaima, primarnim ključevima, spoljnjim ključevima i svim ostalim podacima koji će biti ključni za održavanje referencijalnog integriteta same baze.

Nakon analize podataka iz baze vrši se razvoj formi za prikazivanje, unos i pretragu, odnosno, vrši se "slaganje" komponenti interfejsa. Funkcionalnost takvih sistema ispituje se analizom zahtjeva i postavljanjem SQL upita prema sistemu za upravljanje bazama podataka - DBMSu (eng. ***D**ata**B**ase **M**enagement **S**ystem*). To znači, da bi se dobili podaci iz tabele moraju se znati ime tabele, imena njenih kolona i svi ostali podaci koji su vezani za pojedine kolone (tipovi podataka, ograničenja, itd.). Kao što se može naslutiti, za sisteme koji rade sa velikim brojem tabela unutar baze, morat će se definisati interfejs i upiti za većinu, a u prosjeku je to za 50%, svih tabela. Očigledno je da to za programere predstavlja mukotrpan posao koji se ponavlja iznova u svakom projektu po nekoliko puta. Pored toga, programer mora poznavati SQL sintaksu i rad sa bazom podataka zbog toga što je u tijesnoj sprezi sa DBMSom i što mu od podataka u bazi zavisi sam dizajn interfejsa,slika 1.



Slika 1

3. META-ŠEME BAZA PODATAKA

Metapodaci se mogu definisati kao "podaci o podacima". Međutim, ako se ovaj izraz primjeni na baze podataka onda je preciznije definisati da su metapodaci zapravo "podaci o podacima unutar baze podataka". To znači da nije riječ o običnim podacima, već o podacima kojima opisujemo podatke koji su smješteni unutar baze. U jednoj bazi podataka metapodatke dijelimo na metapodatke za opis:

1. šema, kataloga, tabela, pogleda, kolona i tipova podataka smještenih unutar kolona,
2. baza, korisnika, drajvera, ugnježđenih procedura i funkcija,
3. limita baza podataka (maksimum, minimun te granice baze)
4. podržanih podataka i funkcija, te onih koje nisu podržane.

Metapodaci unutar DBMSa se smiještaju u određene strukture koje su za to namjenjene. Te strukture se nazivaju meta-šeme i one mogu biti smještene:

- interno – nalaze se na istom mjestu kao i podaci,
- eksterno – nalaze se u odvojenom dijelu.

Obje metode imaju svojih prednosti i svojih mana. Ako su smještene interno olakšan je pristup njihovom čitanju i mijenjanju, ali ova metoda dovodi do velike redudantnosti. U drugom slučaju, tj. kada su metapodaci smješteni eksterno, njihovo pretraživanje postaje mnogo efikasnije, nema redudantnosti, ali ekstrakcija ovih podataka postaje tehnički složenija.

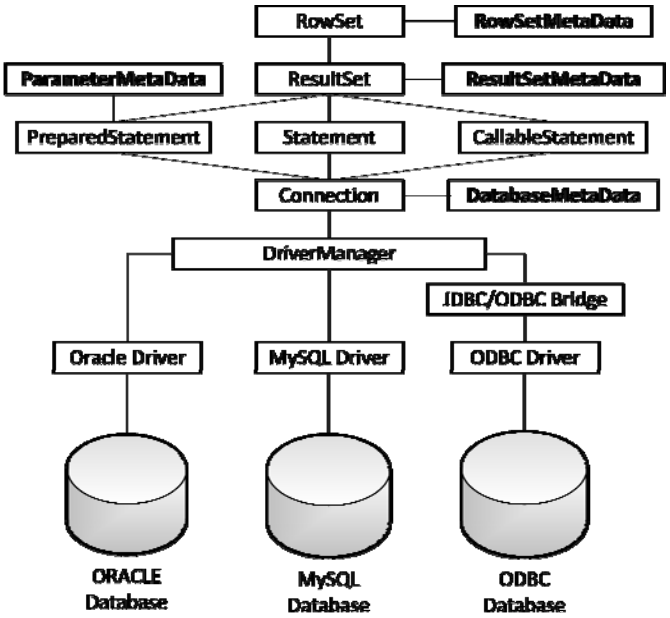
Svi metapodaci se smještaju u riječnik podataka ili sistemski katalog kojem se pristupa preko DBMSa. DBMS započinje sve pretrage i relacije unutar tih kataloga, a njihovo održavanje postaje dosta jednostavnije jer se sve promjene vode na jednom mjestu. Na slici 2 je prikazan način smještanja metapodataka u MySQL bazi.

COLUMN_NAME	COLUMN_TYPE		DATA_TYPE	CHARACTER_M	NUMERIC_PR
zipcode	varchar(30)	11B	varchar	30	(NULL)
zaposleni	varchar(51)	11B	varchar	51	(NULL)
zaposlen	char(1)	7B	char	1	(NULL)
zanr	longtext	8B	longtext	4294967295	(NULL)
XA	varchar(3)	10B	varchar	3	(NULL)
x509_subject	blob	4B	blob	65535	(NULL)
x509_issuer	blob	4B	blob	65535	(NULL)
Wrapper	char(64)	8B	char	64	(NULL)
vrijeme	timestamp	9B	timestamp	(NULL)	(NULL)
vrijednost	varchar(20)	11B	varchar	20	(NULL)
vojska	char(1)	7B	char	1	(NULL)
vlasnik	varchar(51)	11B	varchar	51	(NULL)

Slika 2

Većina današnjih proizvođača DBMSa nudi već ugrađene interfejse (metode) za rad sa metapodacima. U slučaju da takav interfejs ne postoji, korisnici moraju sami da "izvlače" metapodatke iz baze, tj. iz prethodno pomenutih tabela.

Za potrebe ovog rada korišćeni su DBMS MySQL v.5.1.42 i programski jezik JAVA.



Slika 3

Da bi se pristupilo metapodacima, nakon konekcije na bazu podataka mora se kreirati objekat *DatabaseMetaData* koji će sadržavati sve metapodatke baze, kao što su nazivi tabela, nazivi kolona unutar tabela, itd. iz kojeg će se po potrebi moći doći do željenih informacija. Na slici 3 može se vidjeti kompletna šemu klasa i interfejsa za povezivanje JAVE i DBMSa [2].

Sa slike 3 se može primjetiti da *DatabaseMetaData* nije jedini objekat preko kojeg se dolazi do metapodataka. Svaka od klasa *RowSetMetaData*, *ResultSetMetaData*, *Parameter-MetaData* i naravno, *DatabaseMetaData* ima ulogu da korisniku pribavlja podatke iz meta-šema i vraća ih kao niz zapisa, odnosno, kao *ResultSet*.

4. TIPOVI PODATAKA I OGRANIČENJA

Tipovi podataka koji se žele prikazati pri kreiranju korisničkog interfejsa igraju veliku ulogu zbog toga što komponente kojima treba prikazati tekst, logičku vrijednost, broj ili neki drugi oblik podataka, a koji se čuvaju unutar baze podataka nemaju isti izgled. Kod standardnog kreiranja korisničkih interfejsa programer preuzima na sebe obradu ovih komponenti: on sam odlučuje o načinu na koji će se prikazati podaci u samom procesu kreiranja aplikacije. Za programere je to veoma težak posao jer u tom slučaju oni pored dizajna moraju pratiti i tipove podataka u samoj bazi, a potom tim tipovima dodijeljivati određenu komponentu. Osim toga, dodatne komplikacije nastaju ako se promijeni tip podataka unutar baze ili ako je iz bilo kojeg razloga potrebno izvršiti redizajn interfejsa (npr. ako bi se željelo svim komponentama za prikaz numeričkih vrijednosti uvesti ograničenje zabrane unosa karaktera).

Da bi se napravilo elegantno rješenje mogu se koristiti meta-šeme unutar kojih se može pronaći sve što je potrebno za dinamičko kreiranje interfejsa. Izvršavanjem upita nad bazom dobija se objekat tipa *ResultSet*, a nakon toga koristeći naredni segment koda

```
ResultSetMetaData rsmd = rs.getMetaData();
```

dobijaju se metapodaci. Objekat *ResultSetMetaData* obezbjeđuje interfejs za dobijanje podataka koji opisuju tabelu iz koje se traže podaci. Neke od metoda koje su sadržane u ovom objektu su:

- *getColumnClassName()*
- *getColumnType()*
- *getColumnTypeName()*
- *getPrecision()*
- *getTableName()*
- *isReadOnly()*
- *isWritable()*
- ...

Svaka od ovih metoda vraća niz sa informacijama kojima se pristupa preko indeksa kolone.

Najčešće korišćene metode ovog objekta su *getColumnName* i *getColumnType* jer se preko njih dobijaju vrijednosti koje su neophodne da bi se napravio osnovni dinamički interfejs. *getColumnType* metoda vraća informaciju o tipu podataka koji su smješteni unutar kolone. Ova informacija predstavlja SQL-ov tip, odnosno tip podataka koji se koristi unutar DBMSa. Zbog činjenice da tipovi podataka nisu u potpunosti jednaki za sve programske jezike i baze podataka, mora se napraviti sistem koji mapira (preslikava) tipove podataka DBMSa u tipove ciljanog programskog jezika.

U tabeli 1 može se vidjeti koje su razlike između tipova za MySQL i programski jezik JAVA [1].

MySQL	JAVA
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Tiimestamp

Tabela 1

Podaci koji su, pored toga, potrebni za odlučivanje o ponašanju interfejsa su veličina podataka koji mogu biti unijeti u bazu i preciznost. Obje ove veličine dobijaju se na potpuno isti način, koristeći metode namjenjene za to, *getPrecision* i *getScale*, koje vraćaju numeričke vrijednosti maksimalne dužine sadržaja kolone i dužinu frakcije.

5. MODULARNI ATOMIČNI ELEMENTI I NJIHOVE POSTAVKE

Modularni atomični elementi predstavljaju najmanju gradivnu jedinicu sistema. Kao što je ranije spomenuto, za prikaz podataka poželjno je imati elemente koji trebaju biti nezavisni, prilagodljivi i sa mogućnošću postavljanja velikog broja parametara koji će diktirati njihov izgled, prikazivanje i ponašanje.

Svaki od tipova podataka trebao bi imati poseban atomični element kojim će se prikazivati, odnosno treba se obezbjediti unificiran način unošenja i prikazivanja podataka. Tako na primjer, idealno rješenje za komponente koje prikazuju numeričke vrijednosti bi bilo da se u edit modu onemogući unos znakova koji nisu numerički. Također, bilo bi poželjno da komponente za rad sa datumom, vremenom i oznakom vremena imaju predefinisan način unošenja i prikazivanja vrijednosti podataka radi samog očuvanja konzistentnosti sistema, odnosno podataka u bazi.

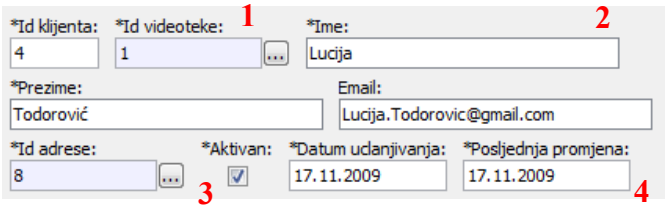
U ovom radu je za prikazivanje podataka iskorišćena grafička biblioteka SWING. Unutar ove biblioteke postoje sve gradivne komponente pomoću kojih se može kreirati bogat korisnički interfejs. Međutim, ove komponente radi složenosti problema ne predstavljaju najbolji izbor za kreiranje dinamičkih korisničkih interfejsa. Jedno od rešenja je kreiranje kompozitnih struktura koje unutar sebe sadrže elemente SWING biblioteke. Svaka od ovih kompozitnih struktura mora biti, kako je već spomenuto, veoma fleksibilna i mora sadržavati sve metode za promjenu vrijednosti atributa koje korisnik ima pravo proizvoljno postaviti unutar same baze, a na osnovu kojih će kasnije biti generisan interfejs.

Korišćenjem objektno orijentisane paradigme, odnosno apstrakcije, interfejsa i nasljeđivanja kreiraju se elementi koji se mogu jednako tretirati. Kao osnovna komponenta pravi se apstraktna klasa unutar koje se definišu sve metode koje će biti zajedničke za te elemente. U ovdje opisanom konkretnom primjeru neke od zajedničkih metoda su:

- *setCapacity* – postavlja kapacitet (broj znakova) koje element može prikazati,
- *setEnable* – postavlja ili uklanja mogućnost editovanja sadržaja komponente,
- *setIsAutoincrement* – postavlja zabranu na editovanje *AutoIncrement* kolona,
- *setData* – postavlja vrijednost komponente,
- ...

Pri kreiranju konkretnih komponenti, odnosno nakon nasljeđivanja ove apstraktne klase, treba predefinisati neke od metoda kako bi se prikaz podataka učinio najoptimalnijim za rad sa određenim tipom.

Na slici 4 mogu se vidjeti neke od komponenti koje su iskorišćene za demonstraciju kreiranja dinamičkog interfejsa.



Slika 4

Svaka od prikazanih komponenti je namjenjena za određeni tip podataka, dok su neke, pored toga, proširene i svojstvom za održavanje referencijalnog integriteta. U skladu s tim, na slici 4 se može uočiti:

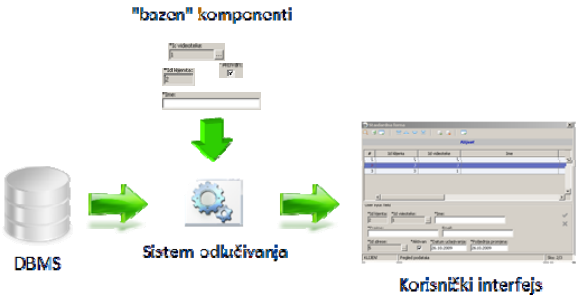
1. komponenta za prikazivanje brojnih vrijednosti sa proširenjem u cilju očuvanja referencijalnog integriteta (dugme koje je vezano za tabelu iz koje je preuzet spoljnji ključ),
2. polje za prikazivanje teksta - u slučaju da dužina tekstualnog zapisa u bazi podataka prelazi predefinisanu dužinu komponente za prikazivanje onda ista postaje komponenta sa višelinijskim unosom,
3. komponenta za prikazivanje logičke istine ili neistine, odnosno komponenta za prikaz *Boolean* tipa podataka i
4. komponenta za prikazivanje datuma.

Predstavljene komponente su samo jedan podskup svih komponenti koje su korišćene i koje mogu biti korišćene. Radi modularne strukture koja je zasnovana na objektno orjentisanim konceptima, veoma brzo se može kreirati bilo koja nova komponenta za prikazivanje podataka.

6. KREIRENJE DINAMIČKIH INTERFEJSA

Nakon što se izvrši prepoznavanje tipova podataka na osnovu *ResultSetMetaData* i nakon što se kreiraju modularne atomične komponente, može se pristupiti izradi dinamičkih korisničkih interfejsa. Upravo te dvije cjeline su neophodne da bi se moglo započeti generisanje interfejsa: uz pomoć prve dobijaju se opisi onoga što se želi prikazati, a sa drugom se to i postiže.

Nakon što se dobiju rezultati, odnosno, vrijednosti iz baze podataka, potrebno je odrediti kojeg tipa su ti podaci, koja su njihova ograničenja i za šta su ta ograničenja bitna. Odgovori na sva ova pitanja dobijaju se "izvlačenjem" podatke iz meta-šema. Princip odluke za odabir elementa koji treba biti prikazan dat je na slici 5.



Slika 5

Kao što se može vidjeti, iz "bazena" komponenti se vrši odabir na osnovu tipa klase podatka koji treba biti prikazan. Bazen komponenti predstavlja skup komponenti koje su unaprijed definisane, a predstavljaju modularno atomične elemente.

Prilikom dodavanja komponenti u vektor, za svaku komponentu se postavljaju parametri koji određuju njeno ponašanje. Zbog objektno orjentisane paradigme koja je iskorišćena pri kreiranju interfejsa, veoma je jednostavno uticati na sve komponente. Tako na primjer, da bi se onemogućile sve komponente dovoljno je proći kroz sve elemente vektora i postaviti njihovo stanje *setEnable* na *false*.

Slaganje elemenata i njihov raspored na formi predstavlja dosta složeniji proces. U te svrhe može se iskoristiti pogodnost jezika JAVA koji u sebi ima ugrađene *Layout* menadžere. Ovi menadžeri raspoređuju komponente na korisničkom interfejsu. Na slici 4 može se vidjeti raspored elemenata nakon završenog procesa dinamičkog kreiranja.

7. ZAKLJUČAK

Kreiranje korisničkog interfejsa nije jednostavan posao, a kreiranje dinamičkog korisničkog interfejsa predstavlja spoj vještina i znanja iz više oblasti računarstva i informatike, pa samim tim i proizilazi njegova težina.

Kreiranjem korisničkog interfejsa na standardan način postoji mogućnost da se za male projekte i projekte koji svoj konceptualni model baze neće često mijenjati veoma jednostavno i brzo dobije zadovoljavajuće rješenje. Međutim, ako se pogledaju poslovni sistemi današnjice može se

zaključiti da oni kao tako mali projekti skoro da više i ne postoje i da je za svaki ozbiljniji posao potrebno mnogo više. Baze podataka postaju mnogo složenije, kompleksnije, a samim tim i teže za održavanje. Programeri, pred kojima je zadatak da naprave interfejs za ovakve baze, imaju ogroman i mukotrpan posao izdvajanja podataka i odlučivanja o tome kako će isti biti prikazani.

Kreiranjem dinamičkih interfejsa koji svoj prikaz temelje na metapodacima unutar baze, odnosno šemama koje opisuju tabele, kolone i njihove tipove podataka, programerima se veoma mnogo olakšava posao dizajniranja i kreiranja interfejsa. Dodatne beneficije su i jednoznačno određeno ponašanje aplikacije tako da kada korisnik jednom nauči princip rada i "gdje mu šta stoji" može lako obavljati posao bez straha da će neka od komponenti da se počne ponašati drugačije.

Potrebno je napomenuti još i to da testiranje ovakvih aplikacija postaje dosta lakše. Svaki modul prije nego se počne koristiti mora biti istestiran i provjeren. To znači da moduli, kao takvi, tvore gradivne elemente koji ne smiju imati grešku i dovesti aplikaciju do otkaza.

U budućnosti se može očekivati sve veći broj kreiranja aplikacija koje će biti u mogućnosti da zadovolje sve veće potrebe programera i njihovih klijenata.

LITERATURA

[1] http://www.java2s.com/Tutorial/Java/0340__Database/MySQLDatatypetoJavaDatatypeConversionTable.htm

[2] Apress, JDBC Metadata MySQL and Oracle Recipes A Problem Solution Approach, Mahmoud Parsian,Mar 2006