UDC: 004.414 INFO M: str. 22-29

TRANSFORMACIONI PRISTUP RAZVOJU SOFTVERA KORIŠĆENJEM UML PROFILA
TRANSFORMATIONAL APPROACH TO SOFTWARE DEVELOPMENT USING UML PROFILES

Slađan Babarogić

REZIME: Osnovni cilj ovoga rada je da se pokaže da korišćenjem MDA pristupa može automatizovati razvoj softvera transformacijom specifikacije date preko koncepta složenog objekta u odgovarajuće koncepte .NET okruženja. Poštujući principe MDA definisan je UML profil baziran na složenom objektu koji služi za projektovanje platformski nezavisnih UML modela. Pored jedinstvenog UML profila za PIM predložen je i UML profil za .NET koji sadrži koncepte koji direktno podržavaju ideju složenog objekta. Takođe, definisana su pravila za preslikavanje koncepata PIM profila u koncepte PSM.NET profila čime je podržan transformacioni razvoj programskih sistema transformacijom PIM UML modela u PSM.NET UML model.
KLJUČNE REČI: Arhitektura bazirana na modelima (MDA), Složeni objekat, Platformski nezavisan model (PIM), Platformski specifičan model (PSM), UML profil, Microsoft .NET.
ABSTRACT: The main goal of this paper is to show that is possible to automate software development by transformation of software specification based on the concept of complex entity to corresponding concepts in .NET environment using Model Driven Architecture (MDA) approach. According to MDA principles, for development of platform independent models (PIM) an UML profile based on complex entity is defined. In addition to the introduced PIM profile, an UML profile for development of platform specific models (PSM) in Microsoft .NET environment is also proposed. For the proposed UML profiles, the appropriate PIM to PSM transformation rules are defined.
KEY WORDS: Model Driven Architecture (MDA), Complex Entity, Platform Independent Model (PIM), Platform Specific Model (PSM), UML profile, Microsoft .NET.

1. UVOD

Stalnom promenom kako softverskih arhitektura i metoda razvoja, tako i korišćenih informacionih tehnologija teško je u jednom kontinuiranom vremenu iole precizno izmeriti progres u razvoju softvera. Još uvek se u razvoju softvera suočavamo sa od ranije poznatim problemima produktivnosti, portabilnosti, interoperabilnosti i održavanja i dokumentovanja. Izgradnja softverskih paketa još uvek zahteva veoma veliko angažovanje ljudskog rada. Sa svakom novom tehnologijom većinu informacionih sistema je potrebno izgraditi iznova. Ovom problemu treba dodati i činjenicu da se veliki broj informacionih sistema razvija korišćenjem više informacionih tehnologija a uz to od svakog sistema se očekuje da obezbedi podršku za komunikaciju sa drugim sistemom [9].

Arhitektura bazirana na modelima (Model Driven Architecture - MDA) predstavlja framework za razvoj informacionih sistema, koji je definisan od strane Object Management Group (OMG). Ključni element MDA predstavljaju modeli. Razvoj informacionih sistema korišćenjem MDA pristupa vođen je aktivnostima modeliranja sistema.

MDA definiše pristup specifikaciji informacionog sistema koji razdvaja specifikaciju funkcionalnosti sistema od specifikacije implementacije za specifičnu tehnološku platformu [13]. U ovom pristupu definišu se dve vrste modela: Model nezavisan od platforme (Platform Independent Model - PIM) i Model sa specifičnostima platforme (Platform Specific Model - PSM). Preporuka je da se oba modela rade u UML notaciji. PIM obezbeđuje formalnu specifikaciju strukture i funkcija sistema i apstrahuje tehničke detalje. PSM uključuje tehničke detalje izabranog ciljnog okruženja, što znači da je baziran na implementacionoj arhitekturi. Ovakav pristup razvoju softvera podrazumeva transformaciju PIM u PSM, odnosno automatsko generisanje srednjeg sloja aplikacija za različite middleware platforme (J2EE/EJB, CORBA, Microsoft DNA, Microsoft .NET).

Još jedan ključni element MDA je transformacija koja se izvršava pomoću softverskih alata. Mnogi CASE alati za objektno-orijentisani razvoj, koji su danas zastupljeni na tržištu, podržavaju automatsku transformaciju iz PSM modela u programski kod. Ono što trenutno nedostaje je komponenta za transformaciju PIM modela u PSM model.

Transformaciju PIM u PSM se bazira na tzv. UML "profilima" (UML profile). UML je dizajniran da bude proširiv tako što obezbeđuje standardne mehanizme za proširenje kojima se definišu novi koncepti sa svojom semantikom. U saglasnosti sa UML specifikacijom, UML profil specijalizuje i profinjuje UML za specifične namene korišćenjem standardnih UML mehanizama za proširenje. Do sada je definisan veći broj UML pro-

fila za različite namene, a najpoznatiji i najkorišćeniji među njima su: UML Profile for CORBA specification, UML Profile for Enterprise Application Integration, UML Profile for Enterprise Distributed Object Computing, UML profil za EJB, UML Profile for Data Modeling, UML Extension for Web Application, UML Profile for Web Ontology Language, UML Profile for Aspect Oriented Modeling.

2. UML PROFILI

UML se široko koristi za specifikaciju, vizualizaciju i dokumentovanje svih tipova programskih sistema bez obzira na implementaciono okruženje. Rasprostranjenost implementacionih okruženja kao što su J2EE/EJB, CORBA i DNA/COM+, u kojima je implementirana većina novih programskih sistema, dovela je do potrebe da se UML modeli koji ih opisuju obogate specifičnim konceptima posmatranog okruženja.

UML je dizajniran da bude proširiv tako što obezbeđuje standardne mehanizme za proširenje kojima se definišu novi koncepti sa svojom semantikom. Takvi mehanizmi mogu biti korišćeni za definisanje koncepata specifične implementacione arhitekture. Mnoge softverske kompanije su prethodnih godina uložile dosta napora da definišu ova proširenja, koja su često bila nekompatibilna jedna sa drugim. Standardizacijom ovakve specifikacije postiže se interoperabilnost alata i implementacionih kostura (frameworks) različitih proizvođača, kao i portabilnost aplikacija koje podržavaju.

UML profil specifikacija definiše skup UML proširenja (ekstenzija) koja opisuju i obuhvataju strukturu i semantiku koncepata specifične implementacione arhitekture. UML definiše nekoliko standardnih mehanizama proširenja u koje spadaju stereotipovi (stereotypes), označene vrednosti (tagged values), ograničenja (constraints) i ikonice za grafičku reprezentaciju (icons). Ovi mehanizmi mogu biti korišćeni za specijalizaciju i finu prečišćenu specifikaciju (refine) u UML-u za specifične namene. Kombinovani, ovi mehanizmi omogućavaju kreiranje novih tipova gradivnih elemenata koje koristimo za modeliranje sistema.

U saglasnosti sa UML specifikacijom, UML profil specijalizuje i profinjuje UML za specifične namene korišćenjem standardnih UML mehanizama za proširenje. Profil ne dodaje ni jedan bazičan koncept. Umesto toga, on specijalizuje postojeće koncepte i definiše konvencije za njihovu primenu. Novi OMG standardi u izradi dodaju prethodno navedenoj definiciji zahtev da UML profil specifikacija uključuje i specifikaciju podskupa UML metamodela koji može biti korišćen za definisanje modela baziranih na posmatranom profilu. Čitav UML metamodel može biti uključen u ovaj podskup.

Kao dodatak, UML profil može uključiti i:

- Standardne ekstenzije kao što su stereotipovi, ograničenja i označene vrednosti
- Semantiku definisanu govornim jezikom
- Dobro formulisana pravila izražena kao ograničenja napisana u OCL-u

OMG je usvojio nekoliko UML profila, kao što su UML Profile for CORBA i UML Profile for Enterprise Application Integration. Radne grupe rade na razvoju UML profila za Web servise i .NET okruženje. Van OMG, Java Community Process je razvila UML profil za EJB.

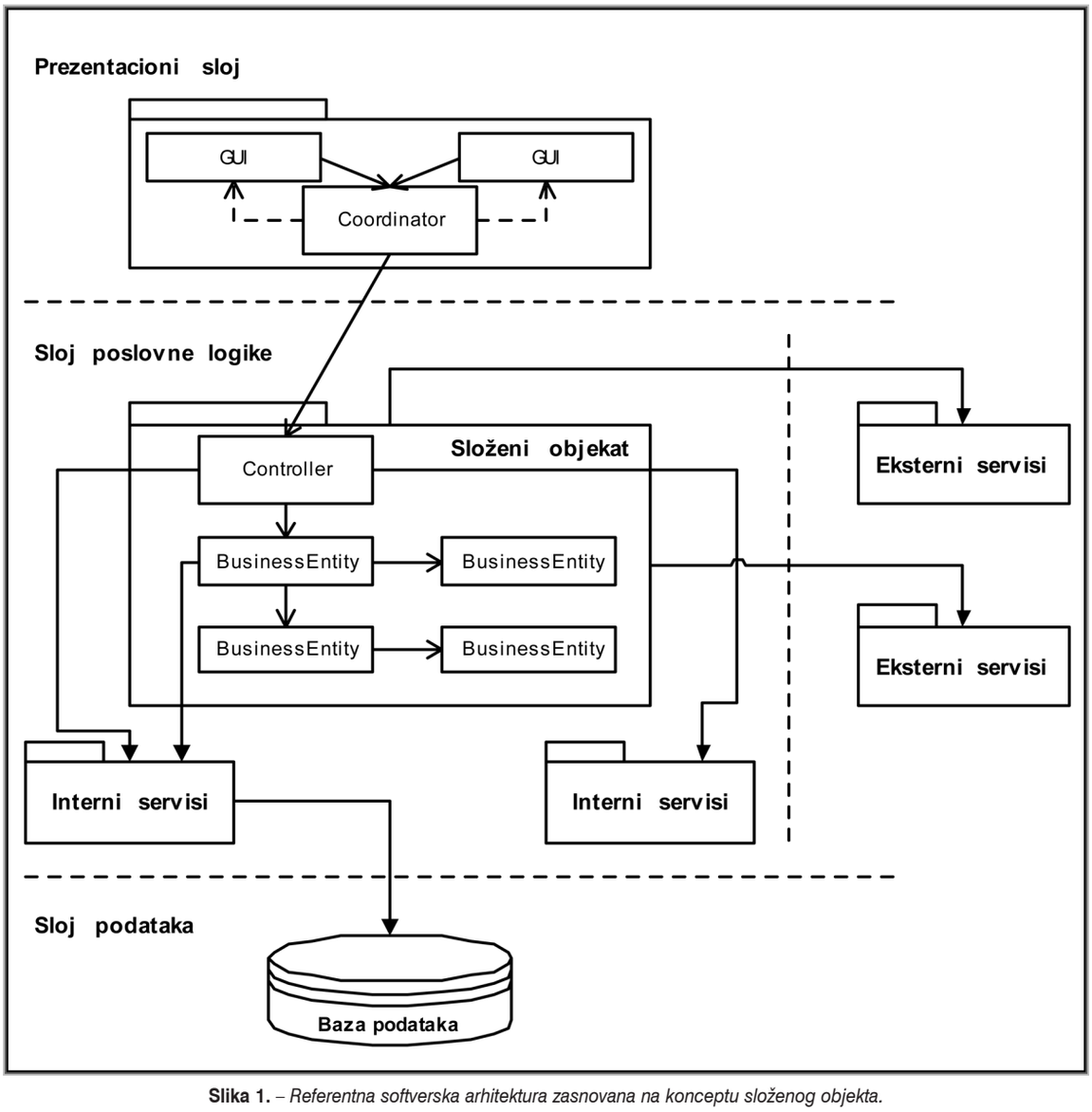
3. PIM UML PROFIL ZASNOVAN NA SLOŽENOM OBJEKTU

Postupak identifikacije složenog objekta detaljno je opisan u [3], tako da je u ovom radu naglasak na definisanim UML profilima koji podržavaju modeliranje sistema pomoću koncepta složenog objekta.

Kao što je na Slici 1. prikazano, jasno se vidi položaj identifikovanog složenog objekta (“sistemski” objekat) u referentnoj troslojnoj logičkoj arhitekturi. U ovom UML profilu složeni objekat predstavljen je kao paket (*package*) koji je stereotipizovan sa <<Complex Entity>>. Složeni objekat mora da poseduje skup stanja kako bi se u njemu mogle formirati sve složene strukture koje se koriste ili generišu u posmatranom slučaju korišćenja. Specifikovane strukture složenog objekta se dalje prikazuju kao povezane klase, formirajući tako dijagram klasa za posmatrani slučaj korišćenja. Svaka od tih klasa predstavlja aplikacionu perzistentnu klasu stereotipa <<Business Entity>>. Dalje se može tvrditi da tako identifikovane klase koje odražavaju strukturu složenog objekta predstavljaju i klase domenskog modela. Tako bi projektovanjem svih slučajeva korišćenja IS na osnovu razlaganja složenih struktura odgovarajućih složenih objekata mogli doći i do kompletnog konceptualnog (domenskog) modela IS.

Sastavni deo paketa kompleksnog objekta je i vrlo značajna klasa sa stereotipom <<Controller>> koja u potpunosti drži logiku jednog slučaja korišćenja (potvrdu postojanja ovog stereotipa možemo naći kod [10]). Pod njegovu odgovornost spadaju iniciranje instanciranja (materijalizacije) odgovarajućeg objektnog modela, prijem i obrada zahteva korisničkog interfejsa, razna izračunavanja, upravljanje transakcijama, pozivi internih i eksternih servisa.

Što se tiče modeliranja prezentacionog sloja u usvojenoj troslojnoj logičkoj arhitekturi, koriste se stereotipovi <<GUI>> i <<Coordinator>>. Stereotipom <<GUI>>označavaju se klase koje su zadužene za direktnu interakciju sa korisnikom sistema, kako za unos podataka tako i za prikaz podataka posmatranog slučaja korišćenja. Treba napomenuti da klase sa stereotipom <<GUI>> prikupljaju podatke potrebne za



Slika 1. – *Referentna softverska arhitektura zasnovana na konceptu složenog objekta.*

ažuriranje složenog aplikacionog objekta, a s druge strane sam složeni aplikacioni objekat obezbeđuje podatke koji će biti prikazani korisniku preko klase sa stereotipom <<GUI>>.

Klasa sa stereotipom <<Coordinator>> odgovorna je za koordinaciju <<GUI>> klasa. Koordinator instancira i otvara ekranske forme u skladu sa definisanim tokom slučaja korišćenja, prikuplja podatke sa ekranskih formi i brine o privremenom skladištenju preuzetih podataka (*State Management*). Koordinator reaguje na događaje koje korisnik generiše na ekranskim formama tako što poziva odgovarajuće operacije klase Kontrolora i kao parametre prosleđuje potrebne podatke unete preko korisničkog interfejsa. Takođe prosleđuje ekranskim formama podatke dobijene od objekata srednjeg sloja. Fowler u [8] detaljno opisuje uzore projektovanja i jedan od njih je *Application Controler* čija odgovornost u potpunosti odgovara ovde predloženom Koordinatoru.

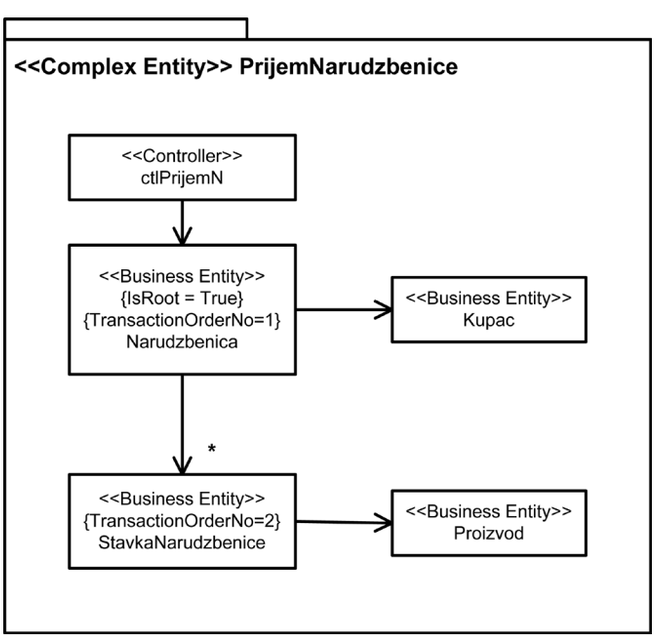
Na sloju poslovne logike identifikuju se i servisi. Servisi su predstavljeni paketima a obezbeđuju usluge objektima klasa srednjeg sloja. Servis je u krajnjoj liniji izdvojena i jasno zaokružena funkcionalnost koja je već ranije specificirana, gde su definisani interfejsi preko kojih se taj servis i koristi. Na ovom nivou apstrakcije nije potrebno govoriti kako su realizovani ovi servisi, već šta oni pružaju od funkcionalnosti. Pre svega tu su infrastrukturni servisi preko kojih se obezbeđuje konekcija sa bazom podataka, pristup direktorijumima, pristup sistemu asinhronih poruka kao i perzistentni servisi koji omogućavaju instanciranje i pamćenje objekata u bazu podataka. U ovom profilu predlažu se dva stereotipa za servise. Prvi stereotip <<Internal Service>> koristi se za označavanje internih paketa koji su sastavni deo programskog sistema koji se razvija. Drugi stereotip <<External Service>> koristi se za označavanje eksternih paketa, udaljene izdvojene funkcionalnosti drugog programskog sistema kojoj posmatrana aplikacija može pristupiti.

Definisani stereotipovi su proistekli iz definisane referentne arhitekture i usvojenog arhitekturnog uzora.

Naziv stereotipa	Primenljivo na	Opis
<<Complex Entity>>	Package	Složeni objekat.
<<Controller>>	Class	Klasa koja drži logiku aplikacije (slučaja korišćenja) u okviru definisanog složenog objekta.
<<Business Entity>>	Class	Perzistentna klasa aplikacije u okviru paketa složenog objekta.
<<GUI>>	Class	Grafički korisnički interfejs (Ekranska forma).
<<Coordinator>>	Class	Koordinator grafičkog korisničkog interfejsa.
<<Internal Service>>	Package	Izdvojena funkcionalnost koja predstavlja servis u okviru posmatranog programskog sistema.
<<External Service>>	Package	Udaljeni servis drugog programskog sistema koji se može koristiti.

Definisane označene vrednosti

Naziv označene vrednosti	Opis
IsRoot	Označena vrednost koja je primenljiva kod stereotipa <<Business Entity>>. Moguće vrednosti su True i False. Označava da li je posmatrani objekat koren u okviru složenog objekta. Složeni objekat može imati više korenih objekata.
TransactionOrderNo	Označena vrednost koja je primenljiva kod stereotipa <<Business Entity>>. Redni broj koji označava redosled spuštanja objekta u bazu podataka u okviru transakcije.



Slika 2. – *Dijagram klasa za složeni objekat PrijemNarudzbenice formiran korišćenjem UML profila za složeni objekat*

Jedan slučaj korišćenja radi nad jednim složenim objektom. U okviru složenog objekta može postojati više korenih (root) objekata koji su obeleženi sa označenom vrednošću *IsRoot*, koja može imati vrednosti True ili False. Kontrolor uglavnom komunicira sa korenim objektima, ali posredno ili neposredno može pozvati operacije i svih ostalih objekata koji su sastavni deo paketa složenog objekta. Označena vrednost *TransactionOrderNo* služi za označavanje redosleda spuštanja objekata (vrednosti atributa objekata) u bazu podataka. To je važno zato što klasa sa stereotipom Kontrolor na osnovu tih označenih vrednosti zna redosled pri izvršavanju transakcije nad bazom podataka. Ograničenja u UML Profilu trebala bi da se definišu korišćenjem *Object Constraint Language* (OCL), gde se pre svega može obogatiti semantika profila – koji koncept se s kojim može povezati.

Na Slici 2. predstavljen je dijagram klasa za složeni objekat PrijemNarudzbenice kreiran korišćenjem stereotipova i označenih vrednosti UML profila za složeni objekat.

4. PSM.NET UML PROFIL BAZIRAN NA KONCEPTU SLOŽENOG OBJEKTA

Umesto da se detaljno i u potpunosti razvija kompletan UML profil za .NET, predložen je UML profil koji sadrži stereotipove i označene vrednosti koji uključuju tehničke detalje .NET okruženja ali samo sa aspekta složenog objekta.

U ovom poglavlju detaljno se opisuje UML profil za .NET okruženje. PSM.NET UML profil sadrži većinu opštih stereotipova koji su ranije definisani u platformski nezavisnom UML profilu za složeni objekat. Osnovni razlog postojanja ovih stereotipova i u ovom profilu je što imaju svoje značenje i u ovom specifičnom .NET UML profilu i lako se mogu transformisati u programski kod nekog od jezika .NET okruženja.

S druge strane gledano, polazeći od ciljnog .NET okruženja, potrebno je identifikovati moguće vrste aplikacija i ugrađene tehnologije u okviru .NET framework-a. Na osnovu te analize moguće je doći do novog skupa UML proširenja koja bi obogatila modele IS sa tehnološkim detaljima .NET okruženje, što bi omogućilo kvalitetniju i precizniju transformaciju u programski kod.

Kao što je u četvrtom poglavlju prikazano, ADO.NET klase u okviru biblioteka .NET framework-a predstavljaju važnije elemente kompletnog okruženja. Klase DataSet i DataTable predstavljaju ključne elemente koje mogu služiti kako za čuvanje proizvoljno složenih struktura podataka tako i za obezbeđivanje perzistentnog servisa u objektno-orijentisanim aplikacijama.

Stereotip <<DataSet>> se koristi da označi UML paket koji u sebi sadrži klase koje su stereotipizovane sa <<DataTable>>.

Kompletan UML paket sa stereotipom <<DataSet>> predstavljaju jednu moguću realizaciju složenog aplikacionog objekta za jedan slučaj korišćenja. Podrazumeva se postojanje klase (klasa Kontrolora) koja bi kontrolisala

Naziv stereotipa	Primenljivo na	Opis
<<DataSet>>	Package	Stereotip paketa koji obuhvata sve perzistentne klase posmatrane aplikacije. Sastavni elementi ovog paketa su klase sa stereotipom <<DataTable>>.
<<DataTable>>	Class	Stereotip klase koja je sastavni deo paketa sa stereotipom <<Dataset>>.
<<DataRelation>>	Association	Stereotip koji povezuje dve klase stereotipa <<DataTable>>, pri čemu se definišu polja iz povezanih tabela po kojima se vrši povezivanje.
<<Win Form>>	Class	Windows grafički korisnički interfejs.
<<Web Form>>	Class	Web grafički korisnički interfejs.
<<Local DLL>>	Package	Izdvojena funkcionalnost koja predstavlja servis u okviru posmatranog programskog sistema (lokalna softverska komponenta). Ovaj stereotip treba da predstavlja *namespace* korenih klasa sa izloženim interfejsom (public operacijama).
<<.NET Remoting Service>>	Package	Udaljeni servis drugog programskog sistema koji se može koristiti.Ovaj stereotip treba da predstavlja namespace korenih klasa sa izloženim interfejsom (public operacijama), s tim što se objekti ove komponente pozivaju preko .NET remoting-a.
<<Web Service>>	Package	Udaljeni servis koji obezbeđuje određenu funkcionalnost i koji je moguće pozivati korišćenjem standardnih Internet tehnologija.

Definisane označene vrednosti

Naziv označene vrednosti	Opis
IsRoot	Označena vrednost koja je primenljiva kod stereotipa <<DataTable>>. Moguće vrednosti su True i False. Označava da li je posmatrani objekat koren u okviru DataSet-a. DataSet može imati više korenih tabela (objekata).
TransactionOrderNo	Označena vrednost koja je primenljiva kod stereotipa <<DataTable>>. Redni broj koji označava redosled spuštanja pojavljivanja tabele u bazu podataka u okviru transakcije.
Namespace	*Namespace* predstavlja logički naziv paketa u kojem se nalaze klase koje se mogu instancirati. Ova označena vrednost je primenljiva kod stereotipova <<Local DLL>> i <<.NET Remoting Service>>.
RelPK	Označena vrednost koja je primenljiva kod stereotipa <<DataRelation>>. Naziv jednog ili više atributa po kojima se povezuje i pripadaju jednoj od klasa u vezi gde predstavljaju identifikator.
RelFK	Označena vrednost koja je primenljiva kod stereotipa <<DataRelation>>. Naziv jednog ili više atributa po kojima se povezuje i pripadaju drugoj klasi u vezi gde predstavljaju spoljni identifikator.

i upravljala specificiranim DataSetom. Klasa Kontrolora na sebe preuzima sve operacije vezane za manipulaciju sa DataSet-om, dok DataSet predstavlja memorijski keš složenih struktura podataka i tako čuva stanje sistema, odnosno posmatrane aplikacije.

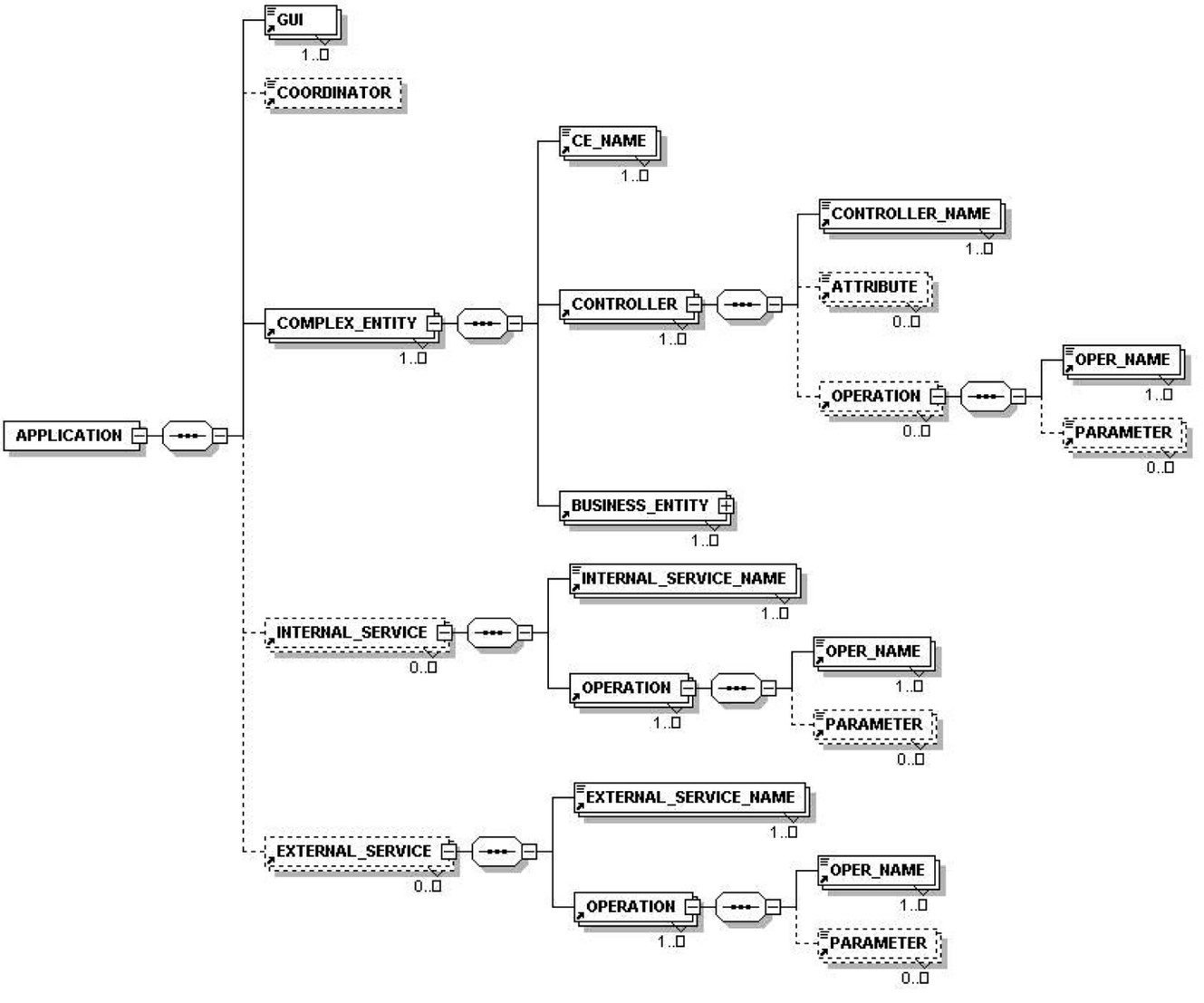
Stereotip <<DataTable>> označava klase koje su sastavni deo paketa sa stereotipom <<DataSet>> i predstavlja jednu moguću alternativu predstavljanja domenskih klasa koje se koriste u određenom slučaju korišćenja.

DataRelation predstavlja stereotip koji povezuje dve klase sa stereotipom DataTable, pri čemu se definišu polja iz povezanih tabela po kojima se vrši povezivanje.

5. RAZVOJ SOFTVERA TRANSFORMACIJOM PIM U PSM

Jedna od značajnih karakteristika životnog ciklusa razvoja IS predloženog u radu [3] je i transformacioni pristup razvoja višeslojnih distribuiranih aplikacija. Prvi korak u usvojenom "Sistemsko-teorijskom životnom ciklusu" je identifikacija gde se koristi koncept složenog objekta koji definiše strukturu ulaza i izlaza i način transformacije ulaza u izlaz. Nakon toga, za fazu realizacije koristi se definisani PIM UML profil za složeni objekat. Na kraju, u fazi implementacije koristi se PSM. NET UML profil. Kao što je u delu opisa MDA rečeno, okosnicu ovog pristupa čine alati (moduli) za transformaciju nezavisnog PIM modela u PSM model.

Na osnovu analize .NET okruženja, lako se može zaključiti da preovlađuju tri osnovna tipa aplikacija. Prvi tip predstavljaju klasične Windows desktop aplikacije, opšte prihvaćeni tip aplikacija sa bogatim grafičkim korisničkim interfejsom za PC računare. U drugi tip spadaju Web aplikacije koje se baziraju na Internet tehnologijama i koje polako preuzimaju primat. Treći tip predstavljaju Web servisi čiji je razvoj takođe zasnovan na Internet tehnologijama, ali je specifičan u tome što ne sadrži grafički interfejs već predstavlja homogen skup operacija koje se izlažu i time obezbeđuju određenu funkcionalnost za krajnjeg korisnika. Samo okruženje dozvoljava kombinacije ovih tipova aplikacija čime je obezbeđena podrška za izgradnju složenih višeslojnih distribuiranih aplikacija. Tako, i



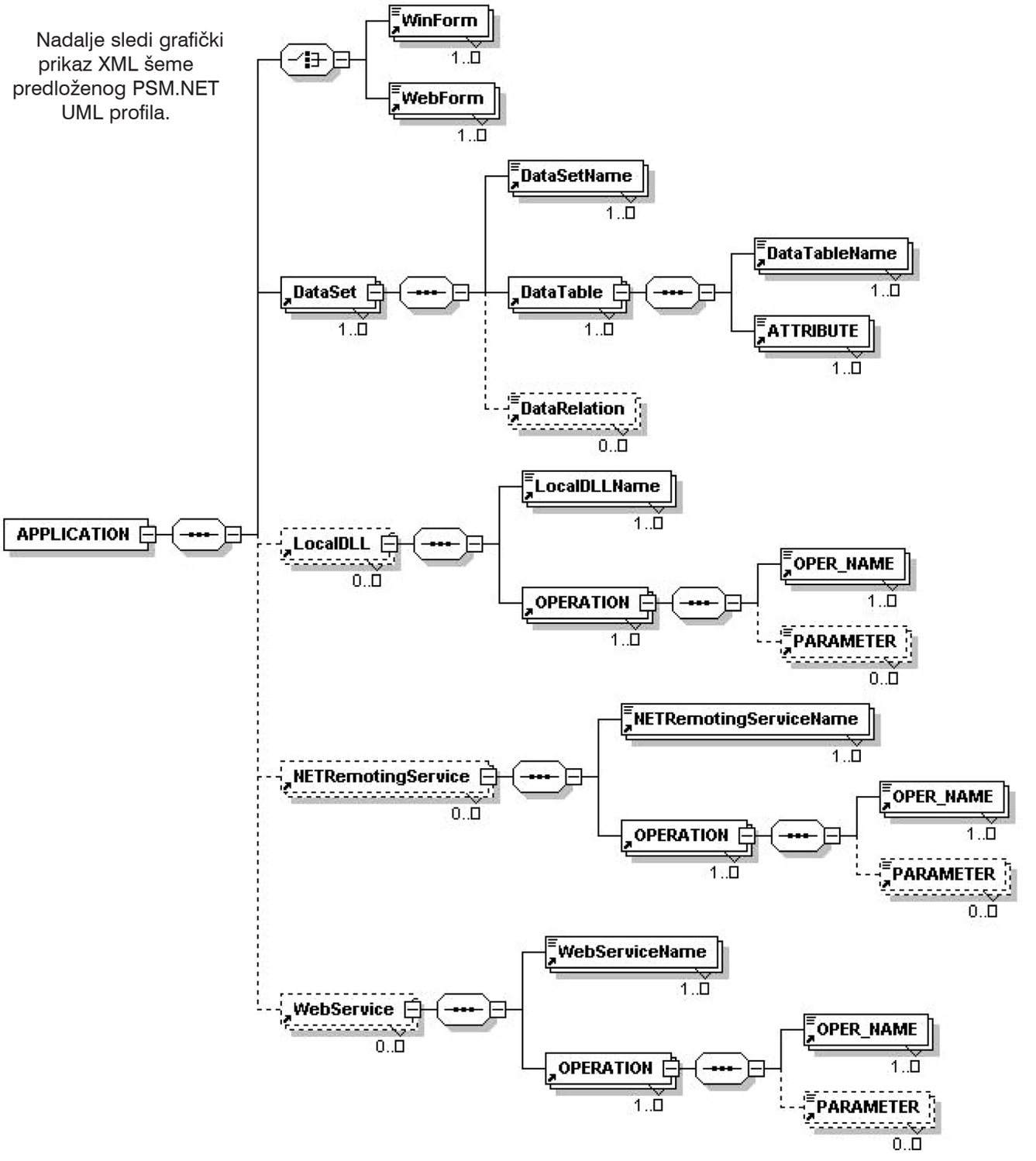
Windows desktop aplikacija i Web aplikacija mogu pozivati operacije udaljenog Web servisa kako bi realizovale neku specificiranu funkcionalnost, a sa druge strane to može biti i .NET Remoting poziv metode nekog udaljenog objekta. Imajući sve ovo u vidu mogu se izvesti moguće transformacije koncepata PIM u koncepte PSM.NET profila (Tabela 1).

Postavlja se pitanje kako je najlakše izvršiti transformaciiu PIM UML modela u PSM UML model. Teško

PIM stereotip		PSM.NET stereotip
<<GUI>>	⇨	<<WinForm>> <<WebForm>>
<<Complex Entity>>	⇨	<<DataSet>>
<<Business Entity>>	⇨	<<DataTable>>
<<Internal Service>>	⇨	<<Local DLL>>
<<External Service>>	⇨	<<.NET Remoting Service>> <<Web Service>>

Tabela 1. – *Tabela transformacije PIM stereotipova u PSM.NET stereotipove*

Nadalje sledi grafički prikaz XML šeme predloženog PSM.NET UML profila.



bi bilo osloniti se na aplikaciju koja bi prepoznavala grafičke elemente modela, ujedno znala njihovu semantiku i zatim prebacivala u grafičke elemente nekog drugog UML profila.

Kao rešenje ovog problema nameće se ideja postojanja zajedničkog jezika u kojem bi oba modela mogla da budu zapisana. Naravno da prva pomisao pada na danas standardni jezik za označavanje – XML.

Sličan problem su imali i proizvođači CASE alata za projektovanje objektno-orijentisanih sistema korišćenjem UML. Model zapisan u internom formatu jednog CASE alata nije bilo moguće učitati pomoću drugog CASE alata, što je otežavalo rad projektanata koji su radili na velikim softverskim projektima a koristili su različite CASE alate. Kao rešenje problema nastao je već standardizovani XMI (OMG XML Metadata Interchange) [19]. Standard definiše set XML tagova kojima se opisuju koncepti UML modela. Tako standardizovani zapis UML modela korišćenjem XML tagova, moguće je učitati u bilo koji CASE alat koji podržava navedeni standard.

U ovom radu nije razmatran XMI već su kreirane specifične XML šeme za svaki od predloženih UML profila. Nadalje se daje grafički prikaz XML šeme predloženog PIM UML profila zasnovanog na složenom objektu.

Identifikovana su dva osnovna načina realizacije transformacije. Prvi mogući način realizacije transformacije modela zapisanog u XML formatu je putem specifičnog programa korišćenjem DOM (Document Object Model) biblioteke klasa. Drugi način realizacije transformacije PIM UML modela u PSM.NET UML model podrazumeva korišćenje XSLT (eXtensible Stylesheet Language Transformations), koji je detaljno i razmatran u ovom radu.

XSLT opisuje transformaciju XML dokumenta u neki drugi format zapisa. Ciljni formati u koje se može transformisati izvorni XML dokument su HTML, običan tekst, vrednosti odvojene zapetama (Comma Separated Values) i drugi standardni formati, čak i formati koje sam programer definiše. Vrlo često se izvorni XML dokument transformiše u drugi XML dokument.

XSLT može se posmatrati kao jedan pravi mali jezik, koji obezbeđuje kompletnu funkcionalnost za jedan ograničen skup zadataka. Za razliku od viših programskih jezika opšte namene (Pascal, C, C++, Java, C# i drugi), ovaj jezik je projektovan za specifičnu familiju problema. Osim za transformaciju iz jednog u drugi XML dokument, XSLT se može koristiti i za generisanje sumarne statistike o XML dokumentima, za prebacivanje podataka iz XML datoteke u bazu podataka, kao i za transformisanje podataka koji idu ka i od mobilnog uređaja.

U sledećoj tabeli prikazane su definisane XSL transformacije. Definisane XSL transformacije bile bi jezgro CASE alata za transformaciju PIM UML modela u PSM. NET UML model.

T 1 <GUI> ⇨ <WinForm>

```
<xsl:for-each select="GUI">
  <WinForm> <xsl:value-of select="GUI"> </WinForm>
</xsl:for-each>
```

T 2 <GUI> ⇨ <WebForm>

```
<xsl:for-each select="GUI">
  <WebForm> <xsl:value-of select="GUI"> </WebForm>
</xsl:for-each>
```

T 3 <Complex Entity> ⇨ <DataSet>

```
<xsl:for-each select="COMPLEX_ENTITY">
  <DataSet>
    <DataSetName>
      <xsl:value-of select="CE_NAME">
    </DataSetName>
    ... // Deo transformacije tagova BUSINESS_ENTITY – pravilo T4
  </DataSet>
</xsl:for-each>
```

T 4	<Business Entity> ⇨ <DataTable> Business Entity.IsRoot ⇨ DataTable.IsRoot Business Entity.TransactionOrderNo ⇨ DataTable.TransactionOrderNo

```
<xsl:for-each select="BUSINESS_ENTITY">
  <DataTable IsRoot="<xsl:value-of select="IsRoot">"
      TransactionOrderNo="<xsl:value-of select="TransactionOrderNo">">
    <DataTableName>
      <xsl:value-of select="BUSINESS_ENTITY_NAME">
    </DataTableName>
  </DataTable>
</xsl:for-each>
```

T 5	<Internal Service> ⇨ <Local DLL>

```
<xsl:for-each select="INTERNAL_SERVICE">
  <LocalDLL>
    <LocalDLLName>
      <xsl:value-of select="INTERNAL_SERVICE_NAME">
    </LocalDLLName>
  </LocalDLL>
</xsl:for-each>
```

Tabela 2. – *XSL transformacije XML tagova PIM profila u XML tagove PSM.NET profila*

6. ZAKLJUČAK

U ovom radu su navedeni i ukratko pojašnjeni značajniji problemi koji se javljaju u razvoju softvera. Kao jedno od mogućih rešenja prezentirana je arhitektura bazirana na modelima (Model Driven Architecture - MDA) koja predstavlja framework za razvoj informacionih sistema, koji je definisan od strane Object Management Group (OMG). Poštujući principe MDA definisan je UML profil baziran na složenom objektu koji služi za projektovanje platformski nezavisnih UML modela. Pored jedinstvenog UML profila za PIM predložen je i UML profil za .NET koji sadrži koncepte koji direktno podržavaju ideju složenog objekta. Takođe, definisana su pravila za preslikavanje koncepata PIM profila u koncepte PSM.NET profila čime je podržan transformacioni razvoj programskih sistema transformacijom PIM UML modela u PSM.NET UML model.

LITERATURA

[1] Rumbaugh J., I. Jacobson, G. Booch : *The Unified Modeling Language Reference Manual*, Addison-Wesley Longman Inc., 1998.

[2] Booch G., J. Rumbaugh, I. Jacobson : *The Unified Modeling Language User Guide*, Addison-Wesley Longman Inc., 1999.

[3] Babarogić S. : *Primena arhitekture bazirane na modelima na razvoj programskih sistema u .NET okruženju*, Magistarska teza, Fakultet organizacionih nauka, Beograd, 2004.

[4] Bennet S., S. McRobb, R. Farmer : *Object-Oriented Systems Analysis and Design using UML*, McGraw-Hill International, 1999.

[5] Bezivin J. : *From Object Composition to Model Transformation with the MDA*, Proceedings of TOOLS'USA, Volume IEEE TOOLS-39, Santa Barbara, 2001.

[6] Chappel D. : *Understanding .NET – A Tutorial and Analysis*, Independent Technology Guides, Addison-Wesley, 2002.

[7] Flater D. : *Impact of Model Driven Standards*, National Institute of Standards and Technology, USA, 2001.

[8] Fowler M. : *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.

[9] Kleppe A., J. Warmer, W. Bast : *MDA Explained: The Model Driven Architecture--Practice and Promise*, Addison-Wesley Inc., 2003.

[10] Larman C. : *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall PTR, 1998.

[11] Lazarević B., Nešković S. : *Objektno-orijentisana specifikacija aplikacija*, materijal za kurseve, Beograd, 1996.

[12] Lazarević B., Nešković S. : *Sistemsko-teorijska kritika objektno-orijentisanog pristupa razvoju softvera*, Zbornik radova Info-Teh '97, Vrnjačka Banja, 1997.

[13] *Model Driven Architecture – A Technical Perspective*, OMG Document ormsc/01-07-01, Architecture Board, Available at: http://www.omg.org/

[14] Nordmoen B. : *Beyond CORBA – Model Driven Development*, Oslo Technology Center, Norway, 2001.

[15] *White Paper on the Profile mechanism*, Analysis and Design Platform Task Force, OMG, 1999.

[16] *Requirements for UML Profile*, Analysis and Design Platform Task Force, OMG, 1999.

[17] *Unified Modeling Language (UML) Specification ver. 1.5*, OMG Document formal/03-03-01, OMG, Available at: http://www.omg.org/

[18] *Unified Modeling Language (UML) Specification ver. 2.0*, OMG Document ptc/03-09-15, Available at: http://www.omg.org/

[19] *XML Metadata Interchange (XMI) Specification ver. 2.0,* OMG Document formal/03-05-02, Available at: http://www.omg.org/

[20] Sceppa D. : *Microsoft ADO.NET – core reference*, Microsoft Press, Redmond, Washington, 2002.

[21] *XML Schema*, World Wide Web Consortium Recommendation, Available at: http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

Slađan Babarogić

Fakultet organizacionih nauka Univerziteta u Beogradu