MDA pristup u realizaciji izveštajnog podsistema informacionih sistema

implementacijom MOF baziranog metamodela

Igor Zečević, Petar Bjeljac, Igor Kekeljević, Ines Perišić
Katedra za primenjene računarske nauke
Fakultet tehničkih nauka
Novi Sad, Republika Srbija
igor.zecevic@uns.ac.rs, pbjeljac@uns.ac.rs, igor.kekeljevic@gmail.com, ines.perisic@gmail.com

Sadržaj—**Produktivnost koju je u izradi softverskih proizvoda donela upotreba modela, pokazala je važnost korišćenja odgovarajućih jezika za modelovanje već u ranim fazama implementacije softvera. I pored toga, izrada izveštajnog podsistema informacionih sistema je oblast u kojoj modelom upravljana arhitektura (MDA) gotovo i da nije korišćena. Cilj ovog rada jeste otklanjanje uočenih problema u tradicionalnom pristupu realizacije izveštajnih podsistema uvođenjem MDA pristupa u proces njihove izrade.**

Ključne riječi - Model-driven architecture; Domain specific language; UML; MOF; Metamodeling; Report;

I. UVOD

MDA pristup [1] modele zajedno sa odgovarajućim transformacijama između njih postavlja kao subjekte od primarnog značaja u svim fazama životnog ciklusa softvera. Programski kod softverskog rešenja u MDA pristupu dobija se kao rezultat automatizovanog lanca transformacija modela. Ovaj pristup baziran je na ideji da prvi korak u efikasnom rešavanju problema bude kreiranje novog ili prilagođavanje postojećeg jezika koji će omogućiti adekvatan opis posmatranog sistema. Produktivnost koju je u izradi softverskih rešenja donela upotreba modela pokazala je važnost korišćenja odgovarajućih jezika za modelovanje već u ranim fazama implementacije softvera.

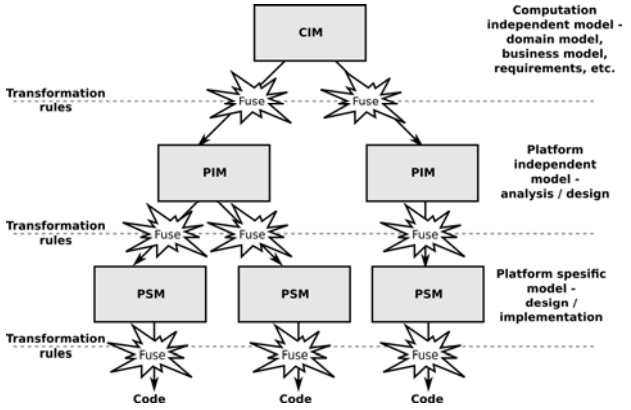
I pored toga, izrada izveštajnog podsistema informacionih sistema je oblast u kojoj MDA pristup gotovo i da nije korišćen. Izveštaj se definiše kao nepromenjivi izlaz informacija dobijen iz aplikativnog sistema na zahtev korisnika [2]. Slično razvoju ostalih delova aplikativnih sistema i izrada izveštaja bazirana je na zahtevima. Korisnici izveštaja zajedno sa aplikativnim programerima i/ili projektantima razvijaju izveštaje. Tradicionalni pristup u izradi izveštaja sastoji se iz sledećih faza:

- Definisanje zahteva - korisnici izveštaja svoje zahteve iznose u vidu postojećih elektronskih i/ili štampanih dokumenata, skica na papiru ili u vidu usmene komunikacije.
- Analiza i specifikacija zahteva - analizom prikupljenih zahteva, projektanti formiraju specifikaciju zahteva za izradu izveštaja. Specifikacija može biti definisana i kroz neki jezik opšte namene kao što je UML [3].
- Implementacija izveštaja - u radnom okruženju za izradu izveštaja, na osnovu specifikacije zahteva, programeri izrađuju izveštaj.
- Testiranje izveštaja - realizovani izveštaji prezentuju se korisnicima izveštaja. U zavisnosti od zadovoljavanja potreba korisnika, postupak se iterativno ponavlja.
- Uključivanje izveštaja u rad - nakon verifikacije od strane korisnika, kreirani izveštaj se uključuje u informacioni sistem.

Uočeni nedostaci u navedenom pristupu su:

- Nedovoljno formalna specifikacija - korišćenje neformalnih oblika definisanja zahteva kao i neusklađena terminologija korisnika sa projektantima može dovesti do pogrešne interpretacije zahteva. Jezici za modelovanje opšte namene koriste direktne programerske koncepte čime njihova upotreba nije odgovarajuća na nivou specifikacije zahteva.
- Nemogućnost validacije prikupljenih zahteva - bez formalne specifikacije u ranoj fazi izrade izveštaja zahteva nije moguće proveriti da li specifikacija odgovara onome što korisnik želi. Eventualne greške u fazi prikupljanja i analize zahteva najčešće se uočavaju tek nakon implementacije izveštaja.
- Nizak nivo automatizacije – i pored velikog broja naprednih radnih okruženja za izradu izveštaja, najveći deo posla prilikom njihove implementacije obavlja se ručno. Ponavljanje procesa ručne izrade izveštaja povećava mogućnost greške.
- Promene u strukturi podataka - izmene u strukturama podataka koje koriste izveštaji povlače sa sobom i izmenu izveštaja. Nepostojanje direktne veze između elemenata struktura podataka i izveštaja koji ih koriste značajno otežava njihovo održavanje.

Cilj rada jeste otklanjanje navedenih problema uvođenjem MDA pristupa u proces izrade izveštajnog podsistema u proces izrade izveštajnog podsistema informacionih sistema. Prednosti koje ovaj pristup donosi u izradi izveštaja su sledeće:



Slika 1. Modelom upravljana arhitektura (MDA)

1) Standardni način specifikacije zahteva - implementacija jezika za modelovanje izveštaja sa vizuelnom sintaksom i grafičkim editorom pruža mogućnost korisnicima izveštaja da svoje zahteve iznose u njima prirodnom okruženju. Zahtevi su opisani konačnim, kontrolisanim brojem elemenata i korisniku bliskom sementikom.

2) Razdvajanje procesa izrade dizajna izveštaja od implementacije njegove funkcionalnosti - jezik za opis izveštaja omogućava korisnicima da kroz modelovanje svojih zahteva u isto vreme projektuju i željeni dizajn izveštaja. Kako je ovaj proces nezavistan od strukture podataka koje izveštajni podsistem koristi, korisnicima su dovoljna samo znanja iz domena njihovog posla.

3) Prenosivost - izmeštanje dizajna izveštaja iz radnog okruženja za njihovu izradu u modele omogućava ponovnu upotrebu već postojećih specifikacija. Na ovaj način "pamet" ne ostaje zarobljena u specifičnom okruženju za izradu izveštaja već u dobro poznatoj i iskoristivoj strukturi modela.

4) Nezavisnost od platforme - izveštaji implementirani u radnom okruženju jednog proizvođača, neupotrebljivi su u radnom okruženju drugog. Potreba za promenom radnog okruženja za izradu izveštaja zahteva njihovu implementaciju od početka. Ideja ustanovljena u modelom upravljanoj arhitekturi (Sl. 1), razdvaja specifikaciju operacija sistema od načina njihove implementacije [1] čime se omogućava nezavisnost od platforme.

5) Automatizacija procesa - primena MDA pristupa pruža podlogu za uvođenje generatora izveštaja u proces njihove implementacije. Generatori kao ulaz koriste modele sa opisima izveštaja i struktura podataka. Rezultat njihovog rada je izvorni kod izveštaja prilagođen ciljanoj platformi (radnom okruženju za izradu izveštaja). Automatizacija u procesu izrade izveštaja pored podizanja kvaliteta izveštaja, smanjenja mogućnosti pojave greške, omogućava i brži odziv na zahteve korisnika.

II. RADOVI NA SLIČNU TEMU I PREGLED RADNIH OKRUŽENJA ZA IZRADU IZVEŠTAJA

MDA pristup, pored toga što je tema velikog broja teorijskih radova [4]-[7], našao je konkretnu primenu i u industriji [8]-[10]. Iskustva uvođenja MDA i rezultati njegove primene kod velikih učesnika iz industrije u projektu MODELPLEX [11] prezentovani su u [12]. Projekat je imao za cilj razvoj tehnika i alata za primenu MDA pristupa u velikim i kompleksnim softverskim sistemima. Veliki napredak u produktivnosti, brži odziv na zahteve tržišta i značajno smanjenje broja grešaka u odnosu na tradicionalni pristup, osnovne su odlike primene MDA koncepta u sistemima kao što su NOKIA, Lucent, USAF, NASA [13], [14]. OMG [15] katalog specifikacija [16] UML profila nudi pregled implementiranih MDA rešenja koja su primenjiva u različitim aplikativnim domenima.

Veliki uticaj na trendove u razvoju izveštajnih podsistema imaju i proizvođači radnih okruženja za njihovu implementaciju. Radna okruženja za izradu izveštaja predstavljaju alate koji omogućavaju brz razvoj izveštaja i njihovu integraciju u aplikativne sisteme. Reprezentativna radna okruženja za izradu izveštaja imaju određeni broj zajedničkih karakteristika:

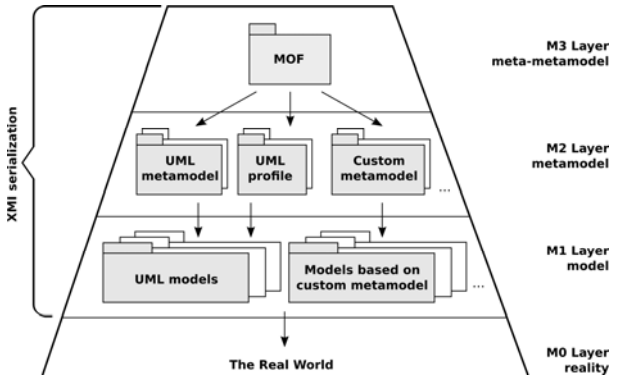
1) razumljiv i korisniku-prilagođen proces izrade dizajna izveštaja;
2) izbor izvora podataka (relacione baze podataka svih standardnih proizvođača, XML datoteke, JDBC API, itd.) putem kojih izveštaji dolaze do svog sadržaja;
3) postojanje predefinisanih šablona i stilova rasporeda elemenata dizajna koje korisnik po potrebi može dodatno prilagođavati;
4) WYSIWYG editor dizajna izveštaja;
5) podrška standardnim izlaznim formatima izveštaja: HTML, HTMLCSS, XML, PDF, RTF, XLS, DOC, CSV;
6) generisanje izvornog koda izveštaja koji će se izvršavati na određenoj platformi.

I pored preklapanja dela navedenih karakteristika sa osnovnim konceptima MDA pristupa, radna okruženja za izradu izveštaja imaju značajne nedostatke u odnosu na navedeni pristup. Prvi nedostatak jeste zavisnost izbora radnog okruženja za izradu izveštaja od platforme u kojoj je realizovan aplikativni sistem u koji će se izveštaj integrisati. Nemogućnost prenosivosti izvornog koda izveštaja između radnih okruženja različitih proizvođača predstavlja drugi značajan nedostatak.

III. METODOLOGIJA RAZVOJA METAMODELA

OMG definiše četvoronivovsku arhitekturu koja razdvaja različite konceptualne nivoe u izradi modela (Sl. 2):

- Nivo M0: Instance - elementi ovog nivoa predstavljaju objekte (instance) koji postoje u posmatranom realnom sistemu.
- Nivo M1: Model sistema - elementi M1 nivoa predstavljaju modele. Elementi nivoa M1 su klasifikatori elemenata nivoa M0.
- Nivo M2: Model modela (metamodel) - elementi ovog nivoa definišu koncepte koji se koriste za modelovanje elemenata u nivou M1. Nivo M2 predstavlja jezik modela. Slično odnosu između nivoa M0 i M1, postoji i relacija između nivoa M1 i M2. Svaki element nivoa M1 mora biti instanca jednog od elemenata nivoa M2.



Slika 2. OMG četvoronivovska arhitektura

- Nivo M3: model M2 nivoa (meta-metamodel) - Nivo M3 određuje koncepte koji se koriste u definisanju metamodela.

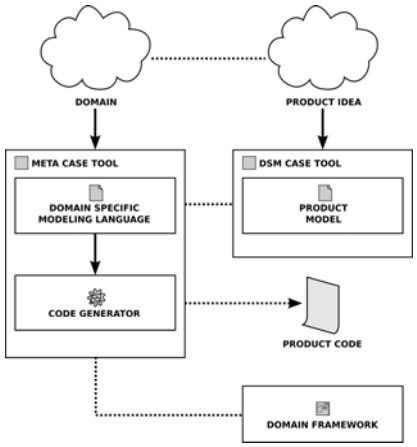
Specifikacija jezika za modelovanje koji omogućava opis specifičnog domena problema na konceptualnom nivou zahteva izradu jezika na M2 nivou (metamodela). OMG definiše dva pristupa u definisanju metamodela [17], [18]. Prvi pristup zasnovan je na specijalizaciji UML jezika. UML pruža skup mehanizama proširenja koji omogućavaju specijalizaciju elemenata modela, prilagođavajući UML za određene domene. Skup ovih proširenja zajedno sa pravilima njihove primene naziva se UML Profili [3].

Drugi pristup u definisanju metamodela baziran je na kreiranju potpuno novog jezika prema pravilima OMG-a za definisanje objektno-baziranih vizuelnih jezika. Novi jezici se definišu korišćenjem Meta-Object Facility (MOF) jezika [19]. Metamodeli na M2 nivou kao što su UML i UML Profili kreirani su prema konceptima MOF jezika. Elementi MOF baziranih metamodela reprezentuju direktne koncepte koji se nalaze u kontekstu posmatranog domena problema. Modelovanje u ovakvom okruženju pruža osećaj direktnog rada sa konceptima domena. Veće podudaranje jezika za modelovanje i domena problema donosi prednosti višeg nivoa apstrakcije: skrivanje kompleksnosti i poklanjanje pažnje bitnim stvarima. Kao takav, MOF baziran metamodel pogodniji je korisnicima koji se bave domenom problema. Zbog toga je za potrebe uvođenja MDA pristupa u izradi izveštajnog podsistema izabran MOF bazirani metamodel.

Cena za izbor MOF baziranog metamodela je razvijanje kompletnog domen-specifičnog okruženja, koje u slučaju UML Profila već postoji. Prateći smernice za kreiranje domen - specifičnih jezika u akademskim člancima [20], [21] i knjigama [22]-[24] moguće je izdvojiti uopšteni šablon na kome se ovo okruženje zasniva (Sl. 3). Specijalizacija opisanog šablona primenjenog na domen izveštajnog podsistema informacionih sistema obuhvata:

1) analizu domena izveštajnog podsistema koja je predstavljena u poglavlju 4.

2) metamodel za opis izveštajnog podsistema (Report metamodel) koji je realizovan korišćenjem Eclipse Modeling Framework-a (EMF) [25]. EMF je otvoreno radno okruženje implementirano u programskom jeziku JAVA. EMF omogućava kreiranje metamodela i njihovu jednostavnu



Slika 3. Domen specifično okruženje

transformaciju u promenjivi JAVA kod. Nastao je kao implementacija MOF specifikacije. Kao takav, EMF se može posmatrati kao implementacija Essential MOF-a (EMOF) [19]. Centralna komponenta EMF radnog okruženja predstavlja ECORE meta-metajezik [25], koji služi za definisanje metamodela u EMF radnom okruženju.

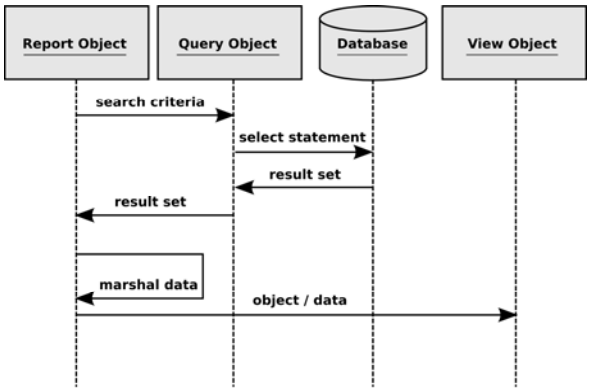
3) realizovani grafički editor za potrebe izveštajnog podsistema, koji podržava rad sa realizovanim metamodelom, razvijen je uz oslonac na Eclipse Graphical Modeling Framework (GMF) [26]. GMF je radno okruženje koje služi za automatsko generisanje grafičkih editora na osnovu ECORE metamodela. GMF u toku procesa izrade grafičkih editora omogućava i korišćenje Object Constraint Language-a (OCL) [27] nad defisanom strukturom metamodela. U ovom jeziku strukturalna ograničenja se iskazuju u formi "invarijanti" dodeljenih MOF metaklasama [28].

4) za domensko radno okruženje koristi se JasperReport [29] radno okruženje za kreiranje izveštaja. Generisani izveštaji prilagođeni su ovoj platformi.

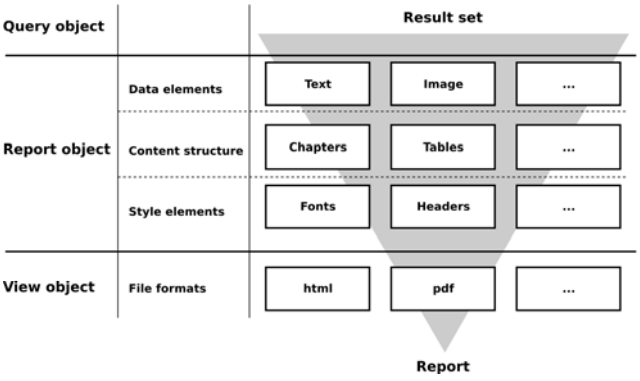
5) modele koji se razvijaju prema definisanom metamodelu, implementirani grafički editor ima mogućnost da serijalizuje u XML. Implementacija generatora koda svodi se na transformaciju XML opisa modela izveštaja nastalog prema Report metamodelu u XML opis izveštaja koji odgovara JasperReport XML šemi. Transformacija je realizovana korišćenjem XLST [30] jezika za transformaciju XML-a.

IV. DOMEN IZVEŠTAJNOG PODSISTEMA

Za sva reprezentativna radna okruženja koja se koriste u izradi izveštaja, može se odrediti zajednička strategija [31] ponašanja prema kojoj se kreiraju izveštaji. Ova strategija opisana je „Report Object" šablonom ponašanja (Sl. 4). Kroz prikazani šablon, jasno su definisane granice opsega domena izveštajnog podsistema. Objekat izveštaja koristeći objekat upita dobija podatke iz relacione baze podataka, koje uz moguću obradu i dodatnu organizaciju podataka šalje objektu prikaza, koji prikazuje izveštaj u željenom formatu. Objekat upita moguće je implementirati kao instancu relacionog jezika upita, bilo kroz ANSI SQL6 ili neku od njegovih specifičnih implementacija, zavisno od konkretnog sistema za rukovanje bazom podataka. Objekat upita može biti enkapsuliran na



Slika 4. Šablon ponašanja izveštajnog podsistema



Slika 5. Fundamentalni atributi izveštaja

strani baze podataka u vidu uskladištene procedure ili prikaza ili na strani objekta izveštaja kao select upit.

Bez obzira na način implementacije objekta izveštaja, postoje zajednički, fundamentalni atributi svih instanci izveštaja (Sl. 5), koje je prema [32] moguće klasifikovati na elemente podataka, strukturu sadržaja i elemente stila.

A. Elementi podataka (data elements)

Elementi podataka su gradivni blokovi, odnosno atomički elementi sadržaja izveštaja koji se dobijaju iz rezultata objekta upita. Elementi podataka mogu biti:

1) polja izveštaja (fields) – predstavljaju elemente podataka jednoznačno mapirana na odgovarajuće kolone iz rezultata objekta upita;
2) promenjive (variables) predstavljaju definisane kalkulacije nad poljima izveštaja;
3) izrazi (expression) predstavljaju matematičke i logičke operacije nad poljima i promenjivim izveštaja.

B. Struktura sadržaja (content structure)

Struktura sadržaja predstavlja uređenu kompoziciju elemenata podataka u vidu njihovog grupisanja i/ili struktuiranja prema predviđenom dizajnu izveštaja. Dizajn izveštaja je šablon po kome se podaci dobijeni iz objekta upita organizuju u stranično orijentisane dokumente radi slanje objektu prikaza. Struktura sadržaja bazirana je na sekcijama. Sekcije su delovi izveštaja koji imaju zadate dimenzije i mogu sadržati elemente podataka. Kompletna struktura izveštaja može sadržati sledeće sekcije:

1) title - naslov izveštaja koji se u dokumentu pojavljuje samo jednom, na samom početku izveštaja;
2) page header - zaglavlje izveštaja koje se pojavljuje na početku svake stranice izveštaja;
3) column header - naslov svake kolone koja se pojavljuje u detalju;
4) group header - zaglavlje grupe u strukturi izveštaja. Grupa u strukturi izveštaja predstavlja grupisanja uzastopnih slogova dobijenih iz objekta upita sa mogućnosti deljenja zajedničkih karakteristika slogova;
5) detail - glavna sekcija izveštaja koja se ponavlja za svaki slog koji se dobije iz objekta upita;
6) group footer - podnožje grupe u strukturi izveštaja;
7) column footer - podnožje svake kolone koja se pojavljuje u detalju;
8) page footer - podnožje izveštaja koje se pojavljuje na dnu svake stranice.begin{figure}

C. Elementi stila (style elements)

Elementi stila predstavljaju skup vizuelnih transformacija primenjenih na elemente podataka radi izmene načina prezentacije sadržaja izveštaja. U elemente stila spadaju:

1) font;
2) background i foreground color;
3) borderWidth;
4) horizontal alignment i vertical alignment.

V. METAMODEL ZA OPIS IZVEŠTAJNOG PODSISTEMA (REPORT METAMODEL)

Glavni koncepti realizovanog metamodela (Sl. 6) predsta - vljeni su sledećim klasama:

A. ReportElement

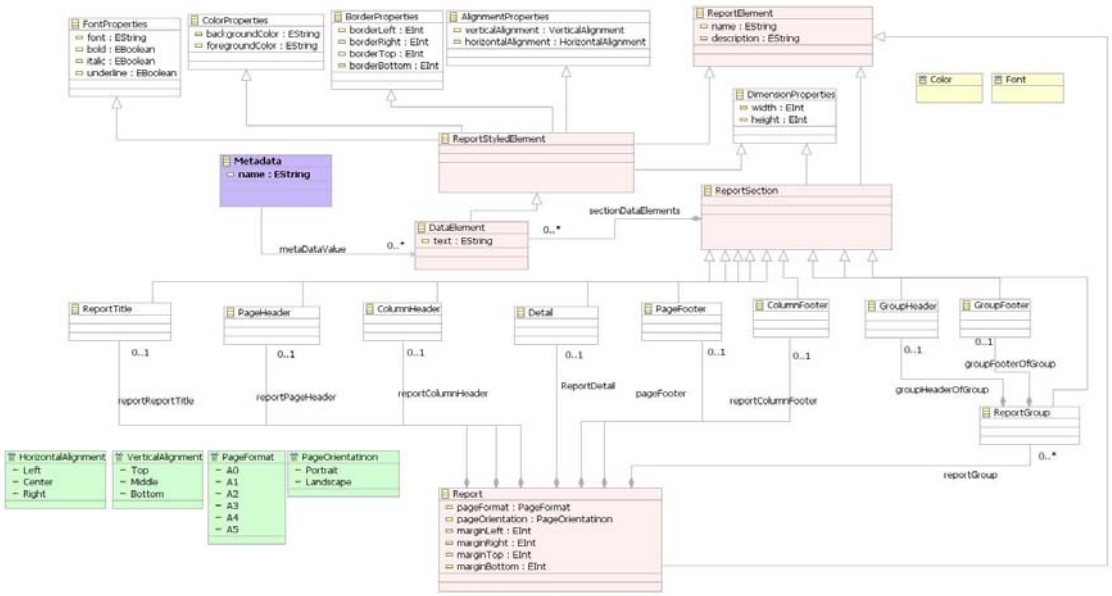
Apstraktna klasa koja čini koreni element (nadklasu) svih ostalih klasa u modelu. Opisuje apstraktni koncept čiji je zadatak predstavljanje zajedničkih osobina svih klasifikatora u metamodelu. Sadrži dva tekstualna atributa:

1) name - ima obaveznu i jedinstvenu vrednost na nivou modela. Svaki objekat modela mora imati jedinstveni naziv u okviru izveštaja kome pripada;
2) description - atribut sa neobaveznom vrednošću koja predstavlja proizvoljan opis elementa modela.

B. ReportStyledElement

Opisuje apstraktni koncept kompozicije vizuelnih transformacija primenjenih na pojedinačne elemente izveštaja. Apstraktna klasa koja nasleđuje ukupno pet klasa od kojih svaka za cilj ima opis određenog segmenta stila, odnosno opis pojedinačne vizuelne trasformacije.

1) FontProperties - sadrži četiri atributa čije vrednosti definišu izgled fonta teksta koji se pojavljuje u pojedinačnim elementima. Atribut font određuje vrstu i veličinu slova. Atributi bold, italic i underline predstavljaju indikatore postojanja dodatnih izmena stila kojim se prikazuje tekst u izveštaju.



Slika 6. Metamodel izveštaja

2) ColorProperties - klasa koja sa vrednostima dva atributa: backgroundColor i foregroundColor, definiše boju teksta i boju pozadine pojedinačnih elemenata.

3) BorderProperties - ova klasa sa četiri atributa: borderLeft, borderRight, borderTop i borderBottom definiše postojanje okvira u elementima izveštaja.

4) AlignmentProperties - poravnanja svih tekstualnih elemenata izveštaja u potpunosti su određena sa dva atributa ove klase. Atribut verticalAlignment definiše vertikalnu poziciju teksta u okviru nekog elementa izveštaja. Atribut horizontalAlignment definiše horizontalnu poziciju teksta u okviru nekog elementa izveštaja.

5) DimensionProperties - svaki pojedinačni element u modelu poseduje kvantitativne vrednosti koje određuju njegove dimenzije. Celobrojni atributi height i width ove klase imaju za cilj da odrede visinu i širinu elementa u milimetrima.

Sve navedene klase koje čine pojedine vizuelne trasformacije su apstraktne i imaju mogućnost instanciranja tek na nivou konkretnog naslednika koji predstavlja pojedinačni element izveštaja. Dekompozicija vizuelnih transformacija koje su predstavljene sa 5 razdvojenih klasa, omogućava primenu pojedinačnih elemenata stila na proizvoljne elemente izveštaja.

C. *DataElement*

Klasa koja reprezentuje elemente podataka, odnosno atomičke elemente sadržaja izveštaja. Kako realizovani Report metamodel opisuje izveštaj na konceptualnom nivou, objekat klase DataElement u sebi ne nosi informaciju o poreklu podatka koji predstavlja (polje baze podataka, statični tekst, promenjiva, kalkulisana vrednost...). Obavezni atribut "text" deskriptivno određuje sadržaj elementa podataka.

D. *ReportSection*

Zajedničke osobine svih sekcija strukture izveštaja apstrahovane su u ovu apstraktnu klasu. Pojedinačne sekcije strukture izveštaja predstavljene su konkretnim klasama koje direktno nasleđuju ReportSection klasu (ReportTitle, PageHeader, ColumnHeader, Detail, ColumnFooter, PageFooter, ReportGroup, GroupHeader, GroupFooter). Iako nijedna od navedenih klasa ne poseduje sopstvene atribute koji bi ih činile specifičnim, svaka sekcija predstavljena je zasebnim naslednikom ReportSection klase. Na ovaj način postoji mogućnost uvođenja kontrole odnosa između pojedinih sekcija na nivou grafičkog editora. ReportSection sadrži jednu relaciju kompozicije koja okuplja proizvoljan broj objekata klase DataElement oko jedne instance konkretnog naslednika klase ReportElement. Ovo znači da svaka sekcija izveštaja može da sadrži neograničen broj elemenata podataka.

E. *Report*

Predstavlja centralni koncept prikazanog metamodela. U procesu modelovanja instanca klase Report reprezentuje jedan izveštaj. Kroz vrednosti atributa ove klase predstavljane su osobine modelovanog izveštaja:

1) pageFormat - obavezan atribut čija je vrednost predstavljena PageFormat enumeracijom. Izborom vrednosti ovog atributa određuju se dimenzije modelovanog izveštaja.

2) pageOrientation - obavezan atribut čija je vrednost predstavljena PageOrientation enumeracijom. Izborom vrednosti pageOrientation atributa korisnik određuje orijentaciju izveštaja u konačnom prikazu.

3) marginLeft, marginRight, marginTop i marginBottom su atributi sa celobrojnom, obaveznom vrednošću. Predstavljaju margine modelovanog izveštaja u milimetrima.

Struktura modelovanog izveštaja predstavlja se kroz relacije između Report klase i klasa koje opisuju elemente strukture izveštaja (naslednici ReportSection klase). Report klasa poseduje nekoliko relacija, od kojih svaka određuje vezu sa izgledom i sadržajem tačno jedne sekcije modelovanog izveštaja. Iako svi koncepti koji reprezentuju pojedine sekcije nasleđuju klasu ReportSection u metamodelu nije

uspostavljena direktna relacija između Report i ReportSection klase. Razlog za ovo je mogućnost kontrolisanja broja instanci pojedinih sekcija kroz višestrukost pojedinih relacija.

VI. ZAKLJUČAK I PRAVCI DALJEG ISTRAŽIVANJA

Prezentovani Report metamodel zajedno sa implementiranim grafičkim editorom služi kao podrška u razvoju modela koji opisuju izveštajni podsistem informacionih sistema. Cilj rada koji je bio uvođenje MDA pristupa u izradu izveštajnog podsistema nije u potpunosti ispunjen.

Naime, realizovani metamodel omogućava standardnu specifikaciju izveštaja od strane korisnika, potpuno razdvajanje dizajna izveštaja od implementacije njegove funkcionalnosti, prenosivost koja se dobija izvozom strukture modela u XML, potpunu nezavisnost od konkretne platforme na kojoj će izveštaj biti izvršavan, ali samo delimično uspostavlja punu automatizaciju procesa izrade izveštaja. Razdvajanje dizajna izveštaja od implementacije njegove funkcionalnosti stvorilo je prazninu između elemenata sadržaja modelovanih izveštaja i struktura podataka koje će izveštaj koristiti. Za potpunu automatizaciju neophodno je asociranje struktura podataka sa elementima izveštaja. Ovo je moguće postići razvijanjem modela koji opisuju izveštaj iz različitih perspektiva (dizajn izveštaja i perspektiva podataka koje izveštaj korisiti) i njihovim međusobnim kombinovanjem.

LITERATURA

[1] Object Management Group, MDA Guide Version 1.0.1, http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

[2] S.W. Ambler, M. Lines, “Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise”, IBM Press, 2012.

[3] Object Management Group, Unified Modeling Language (OMG UML), Superstructure, V2.1.2, http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF

[4] B. Selic, “The pragmatics of model-driven development”, IEEE Software, vol. 20, pp. 19-25, 2003.

[5] B. Perišić, G. Milosavljević, I. Dejanović, B. Milosavljević, "UML Profile for Specifying User Interfaces of Business Applications”, Computer Science and Information Systems (ComSIS), vol. 8, no. 2, pp. 405-426, 2011.

[6] A.W. Brown, "Model-Driven Architecture: Principles and Practice", Journal of Systems and Software Modeling, vol. 3, no. 4, pp. 314-327, Springer Verlag, December 2004.

[7] I. Dejanović, G. Milosavljević, M. Tumbas Živanov, B. Perišić, "A Domain-Specific Language for Defining Static Structure of Database Applications", Computer Science and Information Systems (ComSIS), vol. 7, no. 3, pp. 409-440, 2010.

[8] J. P. Tolvanen, "Making model-based code generation work - Practical examples (Part 2)", Embedded Systems Europe, vol. 9, no 64, pp. 38-41, 2005.

[9] S. Anonsen, “Experiences in Modeling for a Domain Specific Language”, Satellite Activities at the Unified Modeling Language, 7th International Conference (UML 2004), LNCS, vol. 3297, pp. 187-197, Springer, 2004.

[10] G. Milosavljević, I. Dejanović, B. Perišić, "Ready for the industry: A practical approach to teaching mde", 7th Educators Symposium@MODELS 2011: Software Modeling in Education, pp. 31-40, Wellington, New Zealand, www.se.uni-oldenburg.de/documents/olnse-2-2011-EduSymp.pdf

[11] MODELPLEX (MODelling solution for comPLEX software systems), http://cordis.europa.eu/fetch?CALLER=IST_UNIFIEDSRCH&ACTION=D&DOC=5&CAT=PROJ&QUERY=0123e6a38da8:d296:0dc7894d&RCN=79760

[12] P. Mohagheghi, W. Gilani, A. Stefanescu, M.A. Fernandez, B. Nordmoen, M. Fritzsche, “Where does Model-Driven Engineering Help? Experiences from Three Industrial Cases”, Software and Systems Modeling (SoSyM), 2013. In press.

[13] R. Kieburtz, et al “A Software Engineering Experiment in Software Component Generation”, Proceedings of 18th International Conference on Software Engineering, IEEE Computer Society Press, 1996.

[14] Nokia Case Study, http://www.metacase.com/

[15] OMG - Object Management Group, http://www.omg.org

[16] Object Management Group, Catalog of UML Profile Specifications, http://www.omg.org/technology/documents/profile_catalog.htm

[17] L. Fuentes-Fernandez, A. Vallecillo-Moreno, “An introduction to UML profiles”, The European Journal for the Informatics Professional, vol. 5. no. 2, pp. 6–13, 2004.

[18] G. Giachetti, B. Marín, O. Pastor, "Using UML as a Domain-Specific Modeling Language: A Proposal for Automatic Generation of UML Profiles", 21st International Conference on Advanced Information Systems (CAiSE 2009), 2009.

[19] Object Management Group, Meta Object Facility (MOF) Version 2.4.1, http://www.omg.org/spec/MOF/2.4.1

[20] D. Roberts, R. Johnson, "Evolve frameworks into domain-specific languages", Proceedings of the 3rd International Conference on Pattern Languages for Programming, Sept. 4-6, 1996.

[21] G. Milosavljević, B. Perišić, "Really Rapid Prototyping of Large-Scale Business Information Systems", IEEE Workshop on Rapid Systems Prototyping San Diego, 2003.

[22] S. Kelly, J.P. Tolvanen, "Domain-Specific Modeling: Enabling full code generation", Wiley-IEEE Computer Society Press, 2008.

[23] J. Greenfield, K. Short, “Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools”, John Wiley & Sons, Chicester, 2004.

[24] D. Weiss, C.T.R. Lai, “Software Product-Line Engineering”, Addison Wesley Longman, 1999.

[25] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf

[26] Graphical Modeling Project, http://www.eclipse.org/modeling/gmp/

[27] Object Management Group, OCL SpecificationVersion 2.0, http://www.omg.org/docs/ptc/05-06-06.pdf

[28] D. S. Kolovos, R. F. Paige, F. A. C. Polack, “On the Evolution of OCL for Capturing Structural Constraints in Modelling Languages”, Rigorous Methods for Software Construction and Analysis, vol. 5115, pp. 204-218, Springer, 2009.

[29] JasperForge, http://jasperforge.org/projects/jasperreports

[30] World Web Wide Consortium (W3C), The Extensible Stylesheet Language Family Transformations (XSLT), http://www.w3.org/TR/xslt

[31] D. Leffingwell, “Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise (Agile Software Development Series)”, Addison-Wesley Professional, 2011.

[32] S. Mahapatra, “Automatic Report Generation in Model-Based Design”, SAE 2010 Commercial Vehicle Engineering Congress, Chicago, IL, USA, 2010.

ABSTRACT

The productivity in software development brought by the use of models has shown the importance of using appropriate modeling languages in early software implementation fazes. Nevertheless, the implementation of a information system reporting subsystem is an area where the MDA approach has almost never been used. The goal of this paper is to eliminate problems noticed in the traditional approach of implementation of reporting subsystems by introducing the MDA approach into the implementation.

MDA approach in the implementation of an information system reporting subsystem

Igor Zečević, Petar Bjeljac, Igor Kekeljević, Ines Perišić