NAČRTOVANJE RELACIJSKIH PODATKOVNIH BAZ Z UML

Sebastian Lahajnar
PRIS Inženiring, Ljubljana, Slovenija
sebastian.lahajnar@pris-inz.si

Izvleček

Objektna analiza in načrtovanje informacijskih sistemov z UML prinašata tudi nove diagramske tehnike za načrtovanje podatkovnih baz. Opisna moč konceptov jezika UML, predvsem diagrama razredov, je večja od primerljivega modela ER, ki ga danes večinoma uporabljamo za modeliranje podatkovnih struktur. Zaradi objektne narave novih tehnik je njihova uporaba idealna v kombinaciji z objektnimi ali objektno relacijskimi podatkovnimi bazami. Po drugi strani pa imajo relacijski podatkovni strežniki danes še vedno vodilno vlogo na tem področju industrije programske opreme. Namen članka je predstavitev treh osnovnih korakov za kreiranje relacijske podatkovne baze na podlagi objektnega pristopa. Koraki zajemajo zbiranje zahtev uporabnikov z diagrami primerov uporabe, modeliranje podatkovne strukture z diagrami razredov in proces preslikave razredov jezika UML v tabele relacijske podatkovne baze.

Abstract

Information systems object oriented analysis and design with UML provides also new methods and diagramming techniques for database design. UML concepts expressive power is much stronger than that of Entity-Relationship Diagrams we commonly use for that purpose. Due to their object nature these techniques are perfect for designing object and object relational databases. On the other hand, relational database servers still have the leading place on the market today. The purpose of the article is to describe three basic steps needed to create relational databases based on the object approach, including system requirements acquisition with use cases, modeling data structure with class diagrams and the process of transforming UML classes into relational tables.

■■■

Uvod

Pojav objektnih metodologij za analizo in načrtovanje informacijskih sistemov v devetdesetih letih je prinesel tudi nove pristope h gradnji podatkovnih baz. Glavna razloga, zakaj se objektni pristopi vse do danes niso širše uveljavili kot alternativa obstoječim, predvsem modelu ER (entiteta-razmerje), je moč iskati v pomanjkanju enotnega standarda na začetku njihove razvojne poti in izraziti prevladi relacijskih sistemov za upravljanje podatkovnih baz nad objektnimi.

Z nastankom in uveljavitvijo jezika UML kot nespornega standarda na področju objektnega modeliranja je bil prvi razlog odstranjen, po drugi strani pa nič ne kaže, da bi v bližnji prihodnosti prišlo do bistvenih sprememb na tržišču podatkovnih strežnikov. Ti še vedno temeljijo na relacijski tehnologiji, čeprav je vanje vključenih vse več objektnih konceptov. Ob tem se zastavlja vprašanje, ali ne bi bilo smiselno uporabiti diagrame jezika UML tudi za načrtovanje relacijskih podatkovnih baz. Odgovor je vsekakor pritrdilen, saj opisna moč jezika UML zagotavlja verodostojno preslikavo realnega sveta v konceptualni podatkovni model. UML vsebuje številne diagrame za celovito modeliranje informacijskih sistemov z več možnih vidikov. Za načrtovanje podatkovnih baz sta pomembna dva izmed njih: diagram primerov uporabe in diagram razredov. Prvi je namenjen zbiranju zahtev in predstavitvi sistema na najvišjem nivoju, drugi pa modeliranju statične strukture in ga potemtakem lahko uporabimo namesto modela ER. Seveda je potrebno po končanem modeliranju izvesti še preslikavo gradnikov diagrama razredov v tabele relacijske podatkovne baze.

Definiranje zahtev z diagrami primerov uporabe

Načrtovanje podatkovne baze in informacijskega sistema nasploh se prične z zbiranjem zahtev uporabnikov. Posamezni tipi uporabnikov (končni, sistemski itd.) imajo različne zahteve, ki jih je potrebno z uporabo temu namenjenih metod (intervjuji, vprašalniki itd.) najprej pridobiti, prečistiti in nenazadnje predstaviti v neki primerni obliki. UML v ta namen uporablja diagram primerov uporabe. Njegovi osnovni gradniki so (Muller, 1999, str.75):

- Primer uporabe: predstavlja atomarno transakcijo skozi sistem, ki jo sproži neki akter. Posamezna transakcija je sestavljena iz večjega števila operacij, zaporedje katerih pripelje do nekega merljivega rezultata. Atomarna transakcija je tista, ki se mora

izvršiti v celoti. Posamezen primer uporabe opisuje vse možne scenarije interakcije.

- Akter: predstavlja uporabnika sistema (človek, sistem, stroj itd.), ki na različne načine komunicira s sistemom (posreduje podatke, uporablja rezultate obdelav). Akterji pomagajo pri določitvi mej sistema.
- Povezava *komunicira*: predstavlja medsebojni odnos med akterjem in posameznim primerom uporabe.

Diagram primerov uporabe prikazuje medsebojne odvisnosti med akterji in posameznimi primeri uporabe. Primer uporabe je v diagramu prikazan z ovalnimi liki, akter kot možic, povezava pa kot polna črta med njima. V začetni fazi izgradnje diagrama določimo različne akterje, ki bodo uporabljali sistem, in jih po potrebi organiziramo v hierarhično strukturo. Nato določimo vse temeljne transakcije, ki jih bodo sprožili evidentirani akterji in jih z njimi tudi povežemo. V zaključku poiščemo še morebitne preostale povezave med akterji in transakcijami in jih primerno zabeležimo. Na ta način začrtamo meje sistema, kar je osnova za nadaljnjo analizo in načrtovanje.

Posamezen primer uporabe lahko tekom nadaljnje analize razčlenimo in dobimo nabor primerov uporabe, ki podrobneje opisuje temeljno transakcijo. Z vidika transakcij tako ločimo dva tipa primerov uporabe: atomarne, ki so neposredno povezani z akterji in podatomarne, ki razširjajo preostale oziroma so vsebovani v drugih primerih uporabe. V ta namen se uporabljata dva tipa povezav, definirana z ustreznima stereotipoma[1] (Muller,1999, str. 83-86):

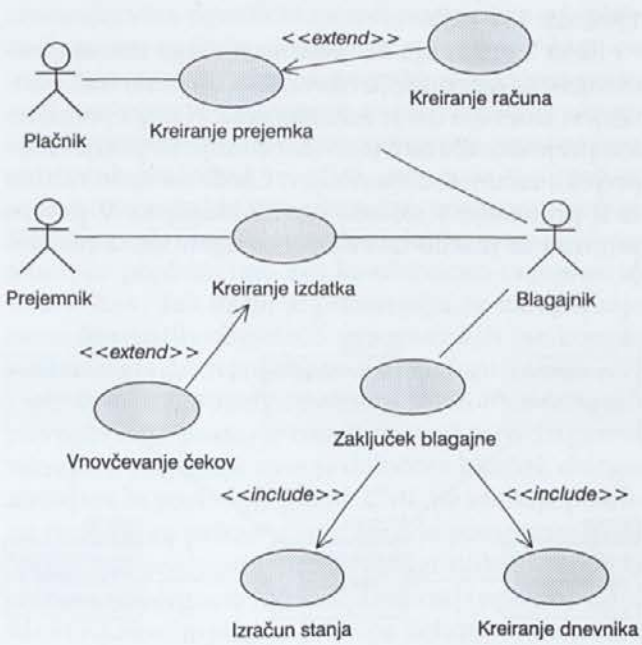
- *Include* (vsebuje): povezava ponazarja primer, ko en primer uporabe vsebuje drugega (povezava je prikazana v obliki puščice od osnovnega do vsebovanega primera uporabe). Z uporabo te povezave lahko določimo nabor primerov uporabe, ki pripadajo različnim transakcijam in predstavljajo komponente za skupno rabo.
- *Extend* (razširja): povezava ponazarja primer, ko se en primer uporabe pogojno izvaja v drugem (povezava je prikazana v obliki puščice od primera uporabe, ki predstavlja razširitev do osnovnega). Z izvzetjem posebnosti poenostavimo posamezen primer uporabe tako, da se lahko osredotočimo na njegovo primarno aktivnost.

Slika 1 prikazuje diagram primerov uporabe za blagajniško poslovanje. Temeljne transakcije so kreiranje izdatka in prejemka ter zaključek blagajne ob koncu dneva, osnovni akterji pa plačnik, prejemnik in blagajnik. Iz diagrama je nadalje razvidno, da se transakcija zaključka blagajne podrobneje deli na izračun stanja in kreiranje dnevnika. Prisotna sta tudi dva primera uporabe, ki razširjata osnovne in sicer kreiranje računa, če kreiramo prejemek na podlagi prejetega računa in vnovčevanje čekov, če gre za izdatek namenjen vnovčevanju prevzetih čekov.

Poleg izdelave ustreznega diagrama primerov uporabe zahteva faza definiranja zahtev tudi podrobnejši opis posameznega primera uporabe in različnih možnih scenarijev. Ta zajema kratek povzetek njegovih funkcionalnih značilnosti, opis vseh temeljnih operacij scenarijev ter definiranje osnovnih podatkovnih elementov in poslovnih pravil. V ta namen lahko uporabimo različne tehnike kot so diagrami aktivnosti, jezika SQL (Structured Query Language) in OQL (Object Query Language) za relacijske ali objektne podatkovne baze; lahko pa vse skupaj kar opišemo z ustrezno strukturiranim besedilom.

Definiranje podatkovne strukture z diagrami razredov

Diagrami razredov prikazujejo statično strukturo sistema, ki zajema predvsem stvari, ki obstajajo v svetu, njihovo notranjo zgradbo in medsebojne odvisnosti. Z vidika modeliranja gre za nabor statičnih gradnikov modela, kot so razredi, vmesniki in povezave. Temeljni gradnik diagrama je razred, v UML



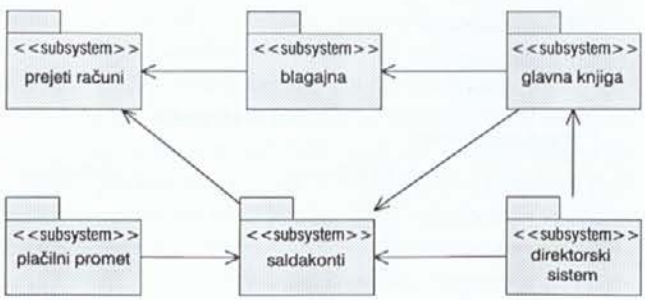
Slika 1: Diagram primerov uporabe blagajniškega poslovanja

1 *Stereotip predstavlja podrobnejšo specifikacijo gradnika v UML (razreda, paketa, atributa itd.). Prikazan je kot beseda v dvojnih srednjih narekovajih, praviloma pridana imenu gradnika, ki ga opisujemo. Nekatere stereotipe UML definira vnaprej, druge določamo sami glede na svoje potrebe.*

definiran kot opis množice objektov, ki si delijo iste atribute, metode, povezave ter vsebino. Razred lahko uporablja nabor vmesnikov, ki določajo množico operacij, ki jih nudi v uporabo svoji okolici. Razredi se kreirajo v diagramu razredov, uporabljajo pa se tudi v večini drugih diagramov. Ime razreda mora biti enolično v okviru paketa, katerega predstavnik je (posamezen razred naslavljamo s sintakso Ime paketa::Ime razreda) (Muller, 1999, str 128-130).

Paket je v UML definiran kot skupina gradnikov modela, ki jih združujemo z nekim določenim namenom. Paket ne nosi nekega globljega semantičnega pomena, gre bolj za imensko področje, ki omogoča prikaz razdelitve sistema na podsisteme in njihove medsebojne odvisnosti. Pakete prikazujemo v diagramu paketov (različici diagrama razredov) v obliki mape z enoznačnim imenom in stereotipom (Subsystem, Framework, Stub itd.). Diagram paketov prikazuje strukturo sistema na najvišji ravni, pri čemer lahko vsak paket predstavlja neki podsistem, evidentiran v fazi zbiranja zahtev. Medsebojno odvisnost dveh paketov (tudi dveh razredov v diagramu razredov) prikažemo s prekinjeno puščico (paket A odvisen od paketa B), uporabimo pa lahko tudi gradnik za prikaz generalizacije (polna puščica s prazno glavo). S podrobnejšo analizo sistema pridobivamo vse več podatkov o notranji strukturi paketov (razredi, povezave, vmesniki), paketi postajajo vse bolj neodvisni drug od drugega, komunikacija med njimi pa poteka z uporabo dobro definiranih, ponovno uporabljivih vmesnikov. Vsak paket se tako v naslednjih fazah analize in načrtovanja razdeli v nabor diagramov razredov, ki so osnova za implementacijo namenske rešitve (Muller, 1999, str. 130-136).

Slika 2 prikazuje del informacijskega sistema, namenjenega spremljanju obveznosti do poslovnih partnerjev, katerega del je tudi blagajna. Naloga podsistema prejetih računov je evidentiranje in potrjevanje prejetih računov dobaviteljev. Glede na vrsto računa se ti prenesejo v saldakonte ali blagajno. V prvem primeru se plačilo izvede preko agencije za plačilni promet ali banke, v drugem pa neposredno v gotovinski obliki. Vsi poslovni dogodki se stekajo v glavno knjigo, od koder direktorski podsistem črpa prečiščene in združene podatke za izvajanje poslovnih analiz.



Slika 2: Diagram paketov spremljanja obveznosti do poslovnih partnerjev

Notacija UML zahteva prikaz razreda v obliki pravokotnika s tremi razdelki, ločenimi z vodoravnimi črtami. Zgornji predel vsebuje ime razreda in njegove osnovne lastnosti (skupaj s stereotipom), osrednji del vsebuje seznam atributov, spodnji pa seznam operacij. Za primere predstavitve podatkovnih struktur z razredi je posebej uporaben stereotip <<persistent>>, ki predstavlja poseben tip razreda za katerega velja, da sistem ohrani stanja njegovih primerkov tudi po prenehanju njihovega obstoja. Atribut UML definira kot poimenovano režo znotraj razreda, ki opisuje nabor vrednosti, ki jih posamezen primerek razreda lahko zavzame (Muller, 1999, str 128). Ta definicija je blizu definiciji atributa v relacijski teoriji, ki govori o atributu kot o preslikavi množice objektov v domeno (Mohorič, 1992, str. 110). Sintaksa za prikaz atributa je:

stereotip vidljivost [številnost]ime:
tip=privzeta vrednost [lastnost]

Standardna notacija UML ne definira nobenega stereotipa za atribut, lahko pa ga po potrebi določimo sami. Vidljivost atributa je lahko javna (katerikoli razred lahko pregleduje in spreminja vrednosti atributa), zaščitena (pregledovanje in spreminjanje vrednosti atributa je dovoljeno metodam razreda in njegovih podrazredov) ali zasebna (pregledovanje in spreminjanje vrednosti atributa je dovoljeno zgolj metodam nekega razreda ne pa tudi njegovim podrazredom). Številnost določa število pojavitev atributa znotraj posameznega razreda, ime je identifikator vrednosti, podatkovni tip je lahko elementaren ali sestavljen in je odvisen od implementacije, privzeta vrednost pa je vrednost, ki jo atributu sistem samodejno dodeli ob kreiranju novega primerka razreda. Posebni vrsti atributa sta razredni atribut, za katerega je značilno, da je njegova vrednost enaka za vse primerke razreda (gre za neke vrste globalno spremenljivko v objektnem okolju) in izpeljani atribut, katerega vrednost izračunamo iz drugih podatkov.

Operacija je storitev, ki jo razred nudi svoji okolici (Muller, 1999, str. 128). Opišemo jo z naslednjo sintakso:

stereotip vidljivost ime (seznam parametrov):
tip [lastnost]

UML definira tri stereotipe za operacije: destroy (uniči primerk), create (kreiraj primerek) in signal (prožilec). Vidljivost operacije je analogna vidljivosti

atributov, ime je identifikator operacije, seznam parametrov določa formalni nabor spremenljivk, ki se posredujejo operaciji (sintaksa parametra: ime: tip=privzeta vrednost), tip pa podatkovni tip rezultata operacije (pogojen z implementacijo). Posebna vrsta operacije je razredna operacija, za katero je značilno, da deluje nad razrednimi atributi oziroma na nivoju celotnega razreda (npr. konstruktor).

Pri obravnavi operacij ne moremo mimo gradnika, imenovanega vmesnik. UML definira vmesnik kot deklaracijo nabora navzven vidnih operacij razreda (Muller, 1999, str.129). Posamezen razred ima lahko enega ali več vmesnikov, prav tako pa lahko vmesnik združuje operacije različnih razredov. Vmesnik ponavadi označujemo s krogcem, lahko pa ga prikazujemo tudi kot razred brez atributov in s pripadajočim stereotipom <<interface>>. V tem primeru odnos med razredom in vmesnikom ponazorimo s prekinjeno puščico (razred realizira vmesnik). Za razliko od razredov pri vmesniku ne moremo govoriti o primerkih vmesnika, temveč govorimo o primerkih razreda, ki vmesnik implementirajo.

Implementacijo operacije imenujemo metoda. Metoda definira konkreten algoritem oziroma postopek, ki določa rezultat operacije. Relacijske podatkovne baze za razliko od objektnih ne vsebujejo metod, saj v osnovi niso namenjene implementaciji dinamičnega dela informacijskega sistema, kar je naloga namenskih rešitev. Navkljub temu številni sistemi za upravljanje relacijskih podatkovnih baz vsebujejo različne tehnike za izvajanje programske logike tudi na strani podatkovnega strežnika (prožilce, shranjene postopke itd.).

Tretji bistveni gradnik diagrama razredov je povezava, ki določa odnos med dvema ali več razredi. UML definira več vrst povezav med katerimi sta najpomembnejši:

- asociacija: vsebinska povezava med razredi, ki določa njihove medsebojne relacije in
- generalizacija: relacija med splošnim in specifičnim elementom.

Asociacije so v diagramu razredov prikazane kot polne črte, ki povezujejo dva ali več razredov. Posamezno stran asociacije imenujemo vloga, ki dejansko ponazarja vlogo pripadajočega razreda v asociaciji. Vsaki vlogi dodelimo številnost, lahko pa tudi ime in vidljivost (analogno z atributi in operacijami). Za boljše razumevanje diagrama lahko asociacije tudi poimenujemo. Številnost je vsekakor najpomembnejša lastnost vloge, saj določa poslovno pravilo, to je omejitev števila objektov, ki sodelujejo v relaciji. Ponavadi prikažemo številnost s parom celih števil ali kombinacijo celega števila z zvezdico (neskončnost), lahko pa uporabimo tudi oznake za naštevanje, interval itd. (primeri številnosti: 0..1,1..*, 2..6 itd.). Določitev številnosti je ključna za podatkovno modeliranje, saj je osnova za kasnejšo določitev zunanjih ključev v primeru preslikave razredov v tabele relacijske podatkovne baze.

Poleg standardne oblike asociacije definira UML tudi več posebnih tipov, ki dodatno specificirajo odnos med dvema razredoma. Agregacija (prikazana kot prazen romb) in kompozicija (prikazana kot poln romb) predstavljata asociacijo med dvema razredoma, ko en razred poseduje drugega oziroma je drugi del prvega. Razlika med njima se kaže v moči lastništva, ki je v primeru kompozicije izrazitejša (en razred ekskluzivno poseduje drugega za razliko od agregacije, kjer ima lahko razred več lastnikov). Zanimiv tip povezave je tudi *povezava z določilom*, posebnim atributom (atribut Vrsta_plačila na sliki 3), katerega vrednosti služijo za razdelitev množice objektov enega razreda, povezanih z nekim določenim objektom drugega razreda. Določilo zmanjšuje efektivno številnost relacije. Večkratne asociacije (asociacije med tremi ali več razredi) ne smejo vsebovati določila, agregacije ali kompozicije. Nekatere asociacije poleg informacij o razredih, ki jih povezujejo, nosijo tudi dodatne informacije, informacije o njih samih. Tovrstne asociacije imajo lastnosti razreda in jih tako tudi modeliramo (asociaciji dodamo razred, v katerem navedemo ime asociacije in nabor atributov). Asociacije modelirane kot razred se ujemajo s konceptom atributov razmerja v modelu ER (Fowler, 1997, str. 75-101).

Drugi tip povezave med razredi je *generalizacija*, ki jo prikažemo s puščico od specifičnega razreda k splošnejšemu. V primeru večnivojske generalizacije z večjim številom specifičnih razredov dobi diagram obliko drevesa. Specifični razredi podedujejo od splošnejšega vse atribute, metode in povezave, hkrati pa vsebujejo še svoje lastne. Z *generalizacijo* so tesno povezani predvsem abstraktnimi razredi (razredi, ki nimajo lastnih primerkov), saj ti nimajo pravega pomena brez ustreznih naslednikov. UML omogoča tudi prikaz večkratnega dedovanja (razred je povezan z večjim številom splošnejših razredov), lahko pa navedeno situacijo predstavimo kot kombinacijo razredov in vmesnikov. Tak način je primernejši za modeliranje razredov tistih objektnih programskih jezikov, ki večkratnega delovanja ne podpirajo (npr. java).

Z določitvijo razredov in njihovih medsebojnih povezav smo postavili osnovno strukturo diagrama razredov. Preostane nam še določitev ključev, domen atributov in poslovnih pravil. UML ne vsebuje posebne notacije za prikaz identifikatorja posameznih objektov, saj predpostavlja, da je identifikator objekta ena izmed njegovih bazičnih lastnosti (pravimo da je identifikator impliciten). Če pa želimo v diagramu identifikator tudi eksplicitno prikazati, moramo notacijo UML za atribute razširiti z oznako 'OID' (Object

identificator) oziroma 'alternate OID=n'. Omenjeni oznaki sta ekvivalentni oznakama 'ključ' in 'n-ti kandidat za ključ' v relacijski teoriji. Za kateri pristop (implicitni ali eksplicitni) se bomo odločili, je povsem v naših rokah. Problem prvega pristopa je, da nam manjka vsebinski podatek o tem, kaj objekt enolično določa, medtem ko pri uporabi drugega pristopa naletimo na probleme pri velikih podatkovnih bazah (nepregledni, obsežni sestavljeni ključi). Zagrizeni zagovorniki objektnih tehnologij se vsekakor ne bodo dali prepričati v smiselnost eksplicitnih identifikatorjev, medtem ko je nam, vajenim načrtovanja z modelom ER, implicitni pristop nekako tuj.

Naslednje vprašanje, ki se zastavlja, je vprašanje predstavitve domen oziroma podatkovnih tipov atributov in kompleksnih poslovnih pravil. UML ne narekuje nekega vnaprej definiranega modela domen, model preprosto pogojuje z implementacijo. Ustrezni podatkovni tipi se tako pripišejo posameznemu atributu v obliki niza za elementarne tipe ali kot objektni tip za sestavljene. Za opis kompleksnih poslovnih pravil pa lahko uporabimo katerikoli naravni ali programski jezik, če že ne kar v UML definiranega jezika za opis omejitev OCL (Object Constraint Language). V vseh primerih je besedilo zapisano v obliki opombe povezane z ustreznim objektom.

Kreiranje relacijskega podatkovnega modela

Z diagramom razredov opišemo podatkovno strukturo sistema na konceptualnem nivoju, čaka pa nas še preslikava v izbran površinski podatkovni model, pri čemer se bomo omejili na danes najpogosteje uporabljen, relacijski. Preslikava diagrama razredov v relacijski podatkovni model je podobna preslikavi modela ER v relacijski podatkovni model. V obeh primerih je najpomembnejši končni rezultat postopka nabor relacij in tabel z definiranimi medsebojnimi razmerji in omejitvami. Za model ER obstajajo natančna navodila v obliki korakov, ki se jih moramo držati pri pretvorbi njegovih osnovnih konceptov v tabele relacijske podatkovne baze. Celoten postopek je tako natančno definiran, da ga danes uporabljajo že vsa pomembnejša orodja CASE. V primeru diagrama razredov prav tako obstaja neki priporočljiv vrstni red korakov, ki nas pripeljejo do normaliziranega relacijskega podatkovnega modela.Vendar postopek vseeno ni tako trivialen, kot bi si morda mislili na prvi pogled. Diagram razredov je namreč opisno mnogo močnejši od modela ER, kar posredno zahteva več raznovrstnih preslikav gradnikov diagrama v tabele. Sam relacijski podatkovni model je z vidika konceptov izredno šibak in preprost, saj vsebuje praktično en sam temeljni koncept: relacijo (tabelo) kot podmnožico kartezičnega produkta izbranih domen. Že preslikava modela ER v relacijski podatkovni model je s tega vidika lahko v posameznih primerih problematična, še bolj pa je negativne strani preprostosti relacijskega podatkovnega modela opaziti pri preslikavi kompleksnih objektov diagrama razredov.

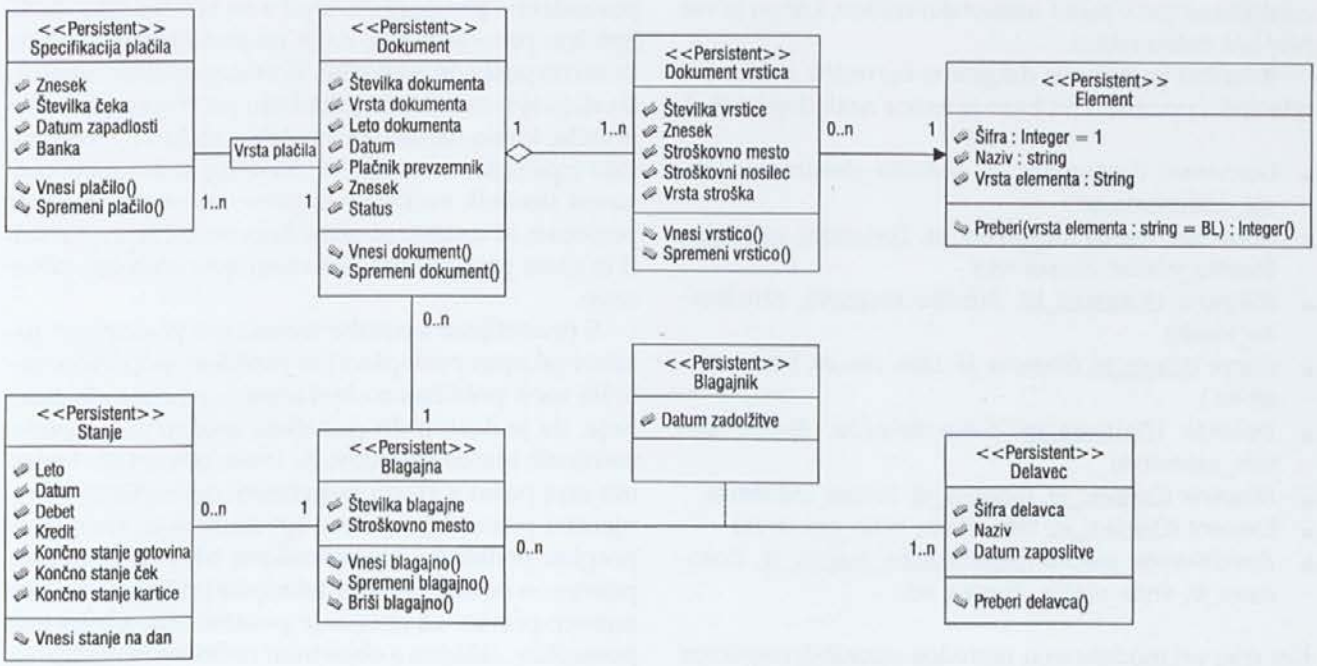
Kreiranja relacijskega podatkovnega modela na podlagi diagrama razredov se lotimo z uporabo pristopa z vrha navzdol. V primeru objektnega modeliranja podatkov imamo na najvišjem nivoju nabor paketov, ki celotno statično strukturo sistema razdelijo na smiselne, medsebojno bolj ali manj neodvisne podsisteme. Paketu jezika UML še najbolj ustreza koncept imenskega prostora oziroma sheme standarda ANSI SQL-92. Težava, ki se pri tem pojavi, je, da danes noben sistem za upravljanje relacijskih podatkovnih baz ne uporablja koncepta sheme skladno s postavljenim standardom. Različni proizvajalci so problem imenskega prostora rešili na različne načine, pri čemer so pri IBM, Oraclu in Informixu koncept sheme poenotili s konceptom uporabnika, pri Microsoftu pa s konceptom podatkovne baze. Iz navedenega sledi, da je preslikava paketov v ustrezne koncepte relacijskega podatkovnega modela tesno povezana z izbranim sistemom za upravljanje podatkovnih baz, kar vsekakor ne pripomore k večji transparentnosti celotnega postopka.

Ko imamo kreirane temeljne podsisteme bodisi v obliki nabora uporabnikov ali nabora podatkovnih baz, se šele prične prava preslikava gradnikov diagrama razredov v tabele. Najpreprosteje je začeti s preslikavo temeljnega gradnika, razreda. Vsak razred iz diagrama razredov se preslika v tabelo relacijske podatkovne baze in vsak atribut razreda se preslika v atribut ali kolono tabele. Atributi v diagramu razredov imajo lahko poleg naziva še vrsto dodatnih lastnosti, ki jih moramo ustrezno obravnavati pri preslikavi. Kot sem že omenil, sam UML vnaprej ne definira nobenega stereotipa za atribute in če jih tudi sami ne, se nam s stereotipi pri preslikavi ni potrebno ukvarjati. Prav tako ni težav z lastnostjo vidljivosti, ki mora biti v primeru relacijskih podatkovnih baz vedno javna, sicer ne bi imeli dostopa do vrednosti atributov s poizvedovalnim jezikom SQL. Pri preslikavi primitivnih podatkovnih tipov imamo dve možnosti: lahko se odločimo za standardizirane ANSI podatkovne tipe ali specifične podatkovne tipe izbranega sistema za upravljanje podatkovnih baz. Prva možnost prinaša večjo prenosljivost in prilagodljivost, druga pa optimizacijo. Več pozornosti je potrebno nameniti kompleksnejšim podatkovnim tipom, ki v nekaterih primerih (naštevni tipi) zahtevajo kreiranje posebnih tabel podatkovnih tipov, v drugih pa vsaj ustrezne tabele preslikav. Nasploh se je pri preslikavi kompleksih atributov smotrno vprašati ali ni morda bolj smiselno tovrstne

podatke modelirati kot samostojne razrede, kar olajša tako razumevanje modela kot tudi samo preslikavo. Podobno velja za lastnost številnosti, ki omogoča prikaz večvrednostnih atributov v okviru razreda. Sam menim, da je večvrednostne atribute bolje odpraviti že med samim modeliranjem diagrama razredov in ne šele ob preslikavi v relacijski podatkovni model. Pri preslikavi razreda v tabelo moramo rešiti še vprašanje ključev, tako glavnega kot morebitnih nadomestnih. Če ima razred eksplicitno definiran glavni in nadomestne ključe (oznaki OID in alternate OID), potem ekvivalentni atributi tvorijo tudi glavni in nadomestne ključe tabele. Po drugi strani pa implicitni pristop zahteva dodajanje novega atributa v tabelo, ki prevzame vlogo glavnega ključa.

Preslikavi razredov sledi preslikava asociacij. Najpreprostejše med njimi, binarne (vzpostavljajo povezavo med natanko dvema razredoma), ustrezajo konceptu zunanjega ključa v relacijski teoriji. Vsaka binarna asociacija se tako preslika v ustrezen zunanji ključ, ki bo vsebovan v eni od dveh tabel, nastalih s preslikavo z asociacijo povezanih razredov. Del katere tabele bodo atributi zunanjega ključa, zavisi od številnosti vloge povezane z razredom. Postopek določanja zunanjega ključa v grobem poteka tako, da v binarnih asociacijah poiščemo vloge z največjo številnostjo 1, te vloge ustrezajo zunanjemu ključu, ki pa ga nato priredimo tabeli nastali iz razreda, povezanega s sosedno vlogo (druga stran asociacije). Če imata obe vlogi v binarni asociaciji številnost večjo od ena, koncept zunanjega ključa na zadostuje več. Asociacija se preslika v vmesno tabelo, ki vsebuje zgolj atribute glavnih ključev obeh povezanih tabel. Enako pravilo velja za večkratne asociacije (asociacije med tremi in več razredi) s tem, da vmesna tabela vsebuje glavne ključe vseh z asociacijo povezanih tabel. Če ima asociacija pripet še asociacijski razred, potem njegove atribute dodamo atributom vmesne tabele. Diagram razredov pozna še več tipov asociacij, ki obogatijo modeliranje podatkovne strukture. Preslikava agregacije se ne razlikuje mnogo od preslikav drugih binarnih asociacij s tem, da postane glavni ključ nadrejene tabele tudi sestavni del glavnega ključa podrejene tabele in ne zgolj zunanji ključ. Kompozicijo, ki za razliko od agregacije zahteva ekskluzivno posedovanje, prav tako preslikamo z uporabo glavnega in zunanjega ključa, ob tem pa dodamo še lastnost kaskadnega brisanja in spreminjanja. Tu je še asociacija z določilom, ki jo izvedemo s kombinacijo zunanjega ključa in atributa določila v podrejeni tabeli.

Tretji bistveni gradnik diagrama razredov, generalizacijo, lahko preslikamo v tabele relacijske baze na dva načina. V primeru direktne preslikave za vsak, tudi abstrakten razred v hierarhiji kreiramo tabelo, ki vsebuje poleg svojih lastnih, specifičnih atributov, še glavni ključ korenskega razreda ali sorodne tabele. Če imamo v korenu hierarhije abstrakten razred, moramo dodatno zagotoviti, da se ob dodajanju in ažuriranju zapisov ustrezne korenske tabele izvedejo tudi spremembe vseh podrejenih tabel, kar v relacijskih podatkovnih



Slika 3: Diagram razredov blagajniškega poslovanja

bazah zagotovimo z uporabo programske logike v obliki prožilcev. Drugi način, imenovan širjenje, narekuje, da vsebujejo podrejene tabele poleg glavnega ključa tudi vse preostale atribute nadrejenih tabel. Prednosti širjenja se kažejo v tem, da imamo vse podedovane atribute v sami tabeli in se nam torej ni potrebno sklicevati na nadrejene tabele. Po drugi strani pa preslikava generalizacije s širjenjem pripelje do nenormaliziranega podatkovnega modela z vsemi znanimi slabostmi (podvajanje, problem konsistentnosti itd.).

Po preslikavi generalizacije v tabele relacijske podatkovne baze se moramo spoprijeti še s poslovnimi pravili in omejitvami. Osnovne omejitve izvirajo iz samega diagrama razredov in njegove preslikave v relacijski podatkovni model in so posledica definiranja glavnih, nadomestnih in zunanjih ključev ter podrobne specifikacije atributov glede omejitve domen in možnosti uporabe ničelnih vrednosti. Preslikavo zahtevnejših poslovnih pravil (v diagramu razredov prikazanih v obliki opomb) izvedemo z uporabo različnih prijemov v odvisnosti od ravni omejitve (raven atributa, tabele ali sistema) in kompleksnosti same logike. Ko obravnavamo omejitve povezane s posameznim atributom ali z razredom z ne prezahtevno logiko, poskusimo zadevo rešiti z ukazom *check constraint* jezika SQL, ki na podlagi poizvedbe preveri izpolnjevanje pogoja. Če je po drugi strani logika za preverjanje pravilnosti zapletena, nam ne preostane drugega kot uporaba ustreznih prožilcev ali celo shranjenih postopkov. Slednji so tudi glavno orodje za realizacijo omejitev na nivoju sistema, pri čemer so najkompleksnejša poslovna pravila pogosto zakodirana kar v samo namensko rešitev, kar pa je vse prej kot dobra izbira.

Rezultat preslikava diagrama razredov s slike 3 v relacijsko podatkovno bazo je nabor naslednjih tabel:

- Dokument (Dokument_id, Številka_dokumenta, Vrsta_dokumenta itd.)
- Dokument vrstica (Dokument_id, *Dokument_vrstica_id*, Številka_vrstice, Znesek itd.)
- Blagajna (Blagajna_id, Številka blagajne, Stroškovno_mesto)
- Stanje (Stanje_id, *Blagajna_id*, Leto, Datum, Debet, Kredit itd.)
- Delavec (Delavec_id, Šifra_delavca, Naziv, Datum_zaposlitve)
- Blagajnik (Delavec_id, Blagajna_id, Datum_zadolžitve)
- Element (Element_id, Šifra, Naziv, Vrsta_elementa)
- Specifikacija_plačila (Specifikacija_plačila_id, *Dokument_id*, Vrsta_plačila, Znesek, itd.)

Ker smo pri modeliranju razredov uporabili implicitni pristop, se glavni ključi (podčrtani atributi) tvorijo šele ob preslikavi v tabele z uporabo sekvenc. Agregacija med razredoma Dokument in Dokument_vrstica narekuje vključitev glavnega ključa tabele Dokument tudi v glavni ključ tabele Dokument_vrstica. Razred Blagajnik se preslika v vmesno tabelo, določilo Vrsta_plačila pa v atribut tabele Specifikacija_plačila, medtem ko so preostale preslikave trivialne.

Kreiranje operacij

Dosedanji postopek preslikave diagrama razredov zelo spominja na preslikavo modela ER v relacijski podatkovni model. V predhodnem poglavju smo namenoma izpustili najpomembnejšo razliko med objektnim in klasičnim pristopom operacije. Na tem mestu se zastavlja ključno vprašanje, kako preslikati operacije razredov in vmesnikov v relacijsko podatkovno bazo. Relacijska teorija se namreč osredotoča zgolj na opis statične strukture sistema, pri čemer dinamični vidik povsem zanemarja. Na srečo imajo vsi današnji sistemi za upravljanje relacijskih podatkovnih baz vgrajene različne možnosti izvajanja programske kode, med katerimi prevladujejo shranjeni postopki in prožilci. Druga možnost je, da operacije realiziramo v obliki namenskih programov, ki se izvajajo na aplikacijskem strežniku v primeru večnivojske arhitekture ali odjemalcu v primeru arhitekture odjemalec-strežnik in ne na strani podatkovnega strežnika. Izbira ravni implementacije posamezne operacije ali vmesnika ima velik vpliv na zmogljivost sistema, zato je vredna temeljitejšega premisleka. V literaturi lahko najdemo kar nekaj vodil in priporočil, kaj je potrebno upoštevati in česa se je potrebno zavedati pri porazdelitvi programske logike na posamezne ravni. Splošno prepričanje je, da je na podatkovni strežnik smotrno postaviti postopke, ki vračajo nabore zapisov, izvajajo spremembe na podatkih, preverjajo poslovna pravila, lovijo napake povezane s podatki, izračunavajo izpeljane vrednosti itd. Po drugi strani pa podatkovni strežnik vsekakor ni pravo mesto za vse tiste postopke, ki izvajajo obsežne komunikacije z odjemalci in s tem prekomerno obremenjujejo omrežne povezave.

S premišljeno uporabo shranjenih postopkov, paketov (skupin postopkov) in prožilcev se lahko v dobršni meri približamo objektnemu pristopu, ki narekuje, da je dostop do podatkov možen samo preko ustreznih metod objektov. Za vsako tabelo tako kreiramo svoj paket z vsemi potrebnimi operacijami (shranjenimi postopki) za vnos, spreminjanje, brisanje in pregled podatkov. Uporabnikom nato ne dodelimo pravice za dostop do tabel relacijske podatkovne baze, temveč pravice za izvajanje posameznih shranjenih postopkov, skladno z objektnim načinom razmišljanja. S tem povsem zadostimo načelu ograjevanja, saj noben namenski program ali uporabnik nima neposrednega

dostopa do podatkov. Ima pa simuliranje objektnega pristopa tudi svojo negativno stran, ki se kaže v nezmožnosti neposredne uporabe jezika SQL. Prav SQL znatno poenostavlja izvedbo kompleksnih poizvedb in drugih operacij nad relacijsko podatkovno bazo in posredno povečuje učinkovitost dela. To pa je tudi tista ključna omejitev, ki v veliki večini primerov nagne tehtnico na stran klasičnega pristopa z uporabo vseh prednosti, ki jih ponujajo relacijski sistemi za upravljanje podatkovnih baz.

Sklep

Prihodnost načrtovanja in razvoja informacijskih sistemov bo tesno povezana z objektnimi tehnologijami. Tako že dandanes večji del razvoja programske opreme poteka v objektnih programskih jezikih (C++, java itd.) ali vsaj objektno usmerjenih razvojnih okoljih (Visual Basic, Oracle Forms Developer, PowerBuilder itd.). Čeprav na področju podatkovnih strežnikov še vedno nesporno prevladuje relacijska tehnologija, pa je z vidika razvoja informacijskega sistema kot celote smiselno razmisliti o objektnem pristopu k analizi in načrtovanju. V ta namen jezik UML nudi standardizirane opisne mehanizme za predstavitev statičnega in dinamičnega vidika poslovnega sistema skladno z temeljnimi načeli objektne tehnologije. Na prvi pogled je sicer videti, da med objektno in relacijsko tehnologijo obstaja globok prepad zaradi razlik pri obravnavi dinamične komponente, na koncu pa se vendarle izkaže, da prehod le ni tako težak in je zatorej objektni pristop primerna alternativa modelu ER tudi pri načrtovanju relacijskih podatkovnih baz. Na končni izbor vedno vpliva še vrsta drugih dejavnikov kot so razpoložljiva orodja CASE, razvojno okolje, tip podatkovnega strežnika in nenazadnje pridobljene izkušnje preteklega dela. Prav slednje pa so ponavadi tisti ključen dejavnik, ki pretehta na stran uporabe preizkušenih metod in ne eksperimentiranja z neznanim.

Literatura

1. Dorsey Paul, Hudicka R. Joseph:
   Oracle 8 Design Using UML Object Modeling. Berkeley: Osborne/MCGraw-Hill, 1999. 496 str.
2. Fowler Martin:
   UML Distilled. Reading: Addison-Wesley, 1997. 183.str.
3. Jurič B. Matjaž, Domajnko Tomaž, Heričko Marijan:
   Objektno modeliranje z uporabo UML, seminarsko gradivo. Ljubljana: SRC, 1998. 139 str.
4. Korth F. Henry, Silberschatz Abraham:
   Database System Concepts. New York: McGraw-Hill, 1991. 694 str.
5. Mohorič Tomaž:
   Uvod v podatkovne baze. Ljubljana: BI-TIM, 1995. 266 str.
6. Mohorič Tomaž:
   Načrtovanje relacijskih podatkovnih baz. Ljubljana: BI-TIM, 1997, 206 str.
7. Muller J. Robert:
   Database Design for Smarties. San Francisco: Morgan Kaufmann Publishers, 1999. 442 str.

◆

Mag. Sebastian Lahajnar je diplomiral leta 1997 na Fakulteti za računalništvo in informatiko v Ljubljani. Po diplomi je vpisal podiplomski študij na Ekonomski fakulteti, smer Informacijsko upravljalske vede, in leta 1999 zagovarjal magistrsko delo z mentorjem prof. dr. Borko Jerman Blažič. Zaposlen je kot razvijalec v podjetju PRIS Inženiring, kjer se ukvarja z razvojem poslovnih informacijskih sistemov.

◆