MODEL OBJEKTI — VEZE KAO IZVRŠNA SPECIFIKACIJA

U radu se razmatra model objekti — veze (MOV) kao izvršna specifikacija u razvoju informacionih sistema (IS). U uvodu je istaknut osnovni problem projektovanja informacionih sistema u fazi konceptualnog modeliranja, primenom klasičnih modela podataka. Zatim se obrazlažu motivi za postojanje izvršnih specifikacija koji bi podržavali semantičke modele podataka. Sledi razrada osnovne teme ovog rada kroz iznošenje motiva za izbor MOV kao osnove za razvoj IS i njegovu dalju operacionalizaciju kroz jezik za specifikaciju. U sledećem poglavlju se neformalno definiše jezik za opis i rukovanje podacima u okviru MOV. Na kraju se govori o jednom načinu implementacije ovog jezika. U radu se koncepti MOV ne opisuju detaljno, već se daju samo njihove osnovne karakteristike.

Uvod

Osnovna teškoća u projektovanju informacionih sistema je semantički jaz između koncepata realnog sistema koji informacioni sistem modeluje i koncepta implementacionog sistema na koje se koncepti realnog sistema moraju sveći prilikom implementacije [8].

Semantički nivo apstrakcije, koje podržavaju klasični modeli podataka koji su realizovani na računaru (hijerarhijski, mrežni i relacioni), jesu, na žalost, bliži konceptima računara nego konceptima čoveka. Zbog toga je potreban izuzetan napor i umeće projektanta da bi uspešno isprojektovali sistem. Da bi se ovaj napor umanjio predloženi su različiti tzv. semantički modeli podataka čiji koncepti su na višem semantičkom nivou. Njihova uloga je, zapravo, da posluže kao intelektualni alat za predstavljanje opisa, specifikacije realnog sistema na osnovu kojeg se radi implementacija, tj. ručno prevođenje sa semantičkog modela na neki implementarni model (model za koji postoji DBMS — Data Base Management System). Osnovna teškoća u vezi sa tim jeste da se isuviše kasno, tek kad se utroši dosta ljudskog rada i ostalih resursa, otkrivaju greške i neadekvatno shvaćeni koncepti realnog sistema. Dodatni problem je još i taj što se održavanje, tj. ispravljanje grešaka, iz-

mene i dopune strukture i procesa informacionog sistema rade direktno na implementacionom modelu, što je proces koji je podložan greškama i zahteva dosta resursa. Posledica toga da je životni ciklus razvoja informacionih sistema izuzetno neproduktivan i skup.

Izvršne specifikacije u modeliranju podataka

Uzrok svemu tome što je specifikacija, opis realnog sistema, nedovoljno precizna i jasna, odnosno, modeli podataka koji služe za specifikaciju nisu dovoljno formalni. Pogotovo što za operaciju modela nisu formalno date sintaksa i semantika. Upotreba formalnih specifikacija pruža niz prednosti u odnosu na informalne. Ona služi kao komunikacioni medijum između korisnika i projektanta. Osim toga, kompletnost i konzistentnost specifikacije, kao i korektnost implementacije u odnosu na specifikaciju može se formalno proveriti i dokazati.

Dakle, specifikacija je važna tehnika softverskog inženjerstva koja se koristi za opis ponašanja željenog sistema. U konvencionalnom razvoju softvera postoji vrlo dugo vreme između specifikacija softverskih zahteva i pojave nečega što može biti izvršeno i testirano. Jedno korisno proširenje teh-

nologije specifikacija je upotreba izvršnih specifikacija kao pristupa za vreme razvoja softvera. Cilj je preliminarne implementacije sistema koji može biti brzo generisan i lako modifikovan. Pošto se modeliranje podataka može posmatrati kao deo softverskog inženjerstva, onda se ista tehnologija može i primeniti.

Cilj ovog rada je davanje doprinosa formalnoj definiciji jednog od najpopularnijih semantičkih modela podataka — (modela objekti-veze) i jezika za definiciju i rukovanje podacima u okviru ovog modela podataka. Kada se jezik formalno definišu sintaksa i semantika, onda se može napraviti njegov interpreter. Na taj način, specifikacija postaje prototip. U daljem tekstu se objašnjava motivacija za izbor modela objekti-veze kao modela za operacionalizaciju.

ER pristup — istorijska perspektiva i dalji pravci

Model objekti-veze (izvorno Entity-Relationship Model — ERM) koji je prvi put objavljen u Chenovom članku [1] jedan je od prvih i najpopularnijih semantičkih modela podataka. Ovaj model se kraće naziva MOV ili ERM. Upotreba MOV dovela je do tzv. ER pristupa u projektovanju informacionih sistema i softverskom inženjerstvu. Početne motivacije za razvoj ER pristupa su [2]:

(1) potreba za jedinstvenim modelom podataka;

(2) potreba za metodologijom logičkog projektovanja baze podataka (BP),

(3) potreba za prevođenjem podataka između različitih DBMS-a.

Glavni razlozi za dalju široku primenu i popularnost ER pristupa su:

(1) koncepti MOV (objekti, veze, atributi) su jednostavni, laki za razumevanje, ali prirodni i snažni koncepti za modeliranje i percepciju realnog sveta,

(2) on pokazuje da tri konvencionalna (hijerarhijski, drežni i relacioni) modela podataka mogu biti izvedeni iz semantički bogatijeg — ER modela, i

(3) koncepti ER modela su formalizovani i predstavljaju osnovu za dalje istraživanje [2].

Učesnici 5. Internacionalne konferencije o ER pristupu, u Dijonu, novembra 1986, istakli su i sledeće prednosti ER pristupa [3]:

- konstrukti ER pristupa su pogodniji za konceptualno modeliranje od drugih modela podataka. Tako, relacioni model usmerava pažnju ljudi sa realnog problema na »stvari koje nisu važne — kao što je normalizacija«. ER pristup, takođe, eksplicitno podržava strukturu posmatranih objekata ne samo u terminima entiteta već, takođe, atributa i veza (koji su često odsutni kod drugih modela podataka). ER pristup omogućuje i strukturu atributa, što je korisno, jer često »atribut« na prvi pogled nije atomski;
- ER pristup dopušta top-down pristup sistem-analizi. On omogućuje prvo uočavanje najvažnijih objekata, koji se tada obogaćuju sa atributima i kasnije normalizovanim strukturama podataka;
- ER pristup je vrlo korisna tehnika za planiranje razvoja informacionih sistema. On pomaže odgovoru na pitanje koju fazu (segment) prvo implementirati;
- jedna značajna prednost ER pristupa u odnosu na relacioni model je izražajna snaga ER dijagrama;
- ER pristup čini, takođe, lakšim analizu događaja u sistemu. Analitičari, na primer, mogu videti kako poslovni događaji utiču na poslovne objekte u terminima operacija kreiranja, ažuriranja i brisanja;
- popularnost ER pristupa kod softverskih projekata je u porastu i on je geografski univerzalan. Zbog toga je relativno sigurna komercijalna investicija.

Sve to je, svakako, uticalo na izuzetnu popularnost i interesovanje za MOV i ER pristup uopšte, kako u akademskim, tako i u poslovnim krugovima. To se, naravno, odrazilo na obim i intenzitet istraživanja na ovom polju. Na ovom mestu ćemo nešto više reći o naporima koji su učinjeni na polju razvoja i implementacije jezika i DBMS-a zasnovanih na MOV. To je usko povezano sa temom ovog rada. Može se reći da je u proteklih desetak godina veća pažnja poklanjana korišćenju MOV kao alatke za projektovanje baze podataka već postojećih hijerarhijskih, mrežnih ili relacionih DBMS-a, nego implementaciji ER DBMS-a.

Da bi jezici i DBMS, zasnovani na ER modelu, postali stvarni izazov, na primer relacionim DBMS, ER pristup mora, verovatno, potpuno iskoristiti prednosti vrlo bogatih struktura podataka sadržanih u MOV i njegovim proširenjima. Primeri takvih struktura podataka su: višeznačni atributi, kompleksni atributi, koncept uloge, koncept generalizacije, n -arne ($n > 2$) veze. Realizacija ovih ciljeva znatno bi doprinela smanjenju semantičkog jaza pri modeliranju realnog sistema, o čemu je bilo reći na početku. To bi doprinelo, takođe, adekvatnijim sistemima za upravljanje bazama podataka i činili lakšim generisanje prototipa aplikacije kroz izvršne specifikacije. Može se reći da većina postojećih upitnih jezika, zasnovanih na MOV, ne sadrže konstrukte za eksplicitno rukovanje ER strukturama podataka. Ipak, neki sistemi kao ECRIN '86 [4] i NETUL [5] sadrže mehanizme koji omogućuju rukovanje ER strukturama podataka.

Ne upuštajući se u dublju analizu predloženih jezika i DBMS-a, koji bi trebalo da podrže MOV, možemo navesti sledeće nedostake:

- mada postoji niz radova, nije usaglašeno šta je osnova za manipulaciju, odnosno koji tipovi koncepata mogu biti operandi;

- većina radova ne daje potpuni skup operatora nad strukturom MOV;
- nema formalne specifikacije jezika;
- implementacioni aspekti često nisu razmatrani;
- ograničenja ER modela kojeg podržavaju (model dopušta samo binarne veze, samo su atomski atributi dopušteni, nisu podržani koncepti specijalizacije i generalizacije, i dr.);
- predložena algebra je vrlo blizu relacije i ne razmatra specifičnosti ER pristupa;
- operacijama ažuriranja i održavanja integriteta poklanjana je mala pažnja.

Sve to nameće potrebu razvoja sopstvenog mehanizma formalizacije i implementacije MOV, kojim bi se izbegle neke od navedenih slabosti. Dobro razrađeni mehanizam omogućio bi relativno laku modifikaciju i poboljšanje samog modela podataka i jezika za njegovu specifikaciju. To je veoma važno, jer se u poslednje vreme nameće potreba za novim proširenjima MOV koja bi omogućila njegovu adekvatnu primenu u oblastima kao što su: modeliranje dinamičkih aspekata informacionih sistema, modeliranje kompleksnih objekata (CAD/CAM aplikacije, automatizacija ureda, projektovanje VLSI, i dr.) kombinacija ER pristupa sa lingvističkim pristupom, itd.

Osnovni koncepti modela objekti-veze

Model objekti — veze je dobro obrađen u literaturi (npr. u [9, 10]) i u ovom radu nisu detaljno obrađeni koncepti ovog modela podataka. Daju se samo osnovne karakteristike koncepata MOV zbog kompletnosti. Struktura modela objekti — veze predstavlja se *dijagramima objekti — veze (DOV)*. Na sl. 1 se vidi DOV za konkretni primer koji ćemo koristiti u daljem tekstu i na njemu su prikazani osnovni koncepti ovog modela podataka.

U MOV sistem se opisuje kao skup objekata i njihovih veza. Pojedinačni objekti (entitet) u sistemu se klasifikuju (apstrakcije, klasifikacije-tipizacije) u tipove objekata. Tip objekta je opšti predstavnik neke klase, a svaki pojedinačni objekat predstavlja jedno pojavljivanje (primerak) datog tipa. Tipovi objekata u našem primeru (sl. 1) jesu, na primer, jedinica, slušalac, nastavnik, i dr.

Veze u modelu opisuju način povezivanja objekata. Svaki tip veze između dva objekta definiše dva preslikavanja, preslikavanja sa skupa pojavljivanja prvog objekta u skup pojavljivanja drugog objekta i (inverzno) preslikavanje sa skupa drugog objekta u skup prvog objekta. Svakom preslikavanju u vezi mogu se dodeliti imena. Npr. veza Sl-Pr ima preslikavanja

Sluša: SLUŠALAC → PREDMET, (za svakog slušaoca daje predmete koje sluša).

Slušao: PREDMET → SLUŠAOC (za svaki predmet daje slušaoce koji ga slušaju).

Ovde se može uočiti da postoji mogućnost uspostavljanja više *tipova veza* između istih tipova objekata (veze Sl-Pr i POLOŽIO između objekata SLUŠALAC i PREDMET). Jedna od bitnih karakteristika veza između objekata je *kardinalnost* preslikavanja koja je čine, npr. kardinalnost preslikavanja definiše se parom (DG, GG), gde DG (donja granica) daje najmanji mogući, a GG (gornja granica) najveći mogući broj pojavljivanja tipa objekta PREDMET za jedno pojavljivanje tipa objekta SLUŠALAC. Ovde mora biti zadovoljen uslov $DG \leq GG$.

Svaka binarna veza je definisana sa dva preslikavanja. Nazivi ovih preslikavanja značajno opterećuju sam MOV, pa se mogu izostaviti, a u specifikaciji operacija nad preslikavanjem uloga imena preslikavanja mogu preuzeti imena njihovih kodomena. Zbog toga se u MOV unose obično samo nazivi veza i kardinalnost preslikavanja.

Atribut i domen. Objekti u sistemu se opisuju preko svojih ATRIBUTA (svojstava). Na primer, atributi objekta SLUŠALAC su IME, PREZIME, BR IND. Atributi uzimaju vrednost iz skupa mogućih vrednosti. Ovi skupovi se nazivaju domenima. Na primer, atribut *BrInd* uzima vrednosti iz skupa mogućih brojeva indeksa. Uz određene pretpostavke imena atributa i domena se mogu izjednačiti, što je i učinjeno na sl. broj 1. Formalno se atribut može predstaviti kao preslikavanje iz skupa objekata datog tipa u skup vrednosti (domen). U zavisnosti od kardinalnosti preslikavanja postoje, različite vrste atributa:

a) jednoznačni atribut objekta ($DG = 1$ i $GG = 1$);

b) identifikator objekta (i kod inverznog preslikavanja $DG = 1$ i $GG = 1$) — na slici je pored njih stavljena '*';

c) višeznačni atribut ($DG = 1$ i $GG = M$).

Uslov za postojanje atributa je $DG > 0$. Ovde se može uočiti razlika između koncepta veze i koncepta atribut. *Veza* definiše dva preslikavanja, direktno i inverzno, između dva tipa objekta, a *atribut* preslikavanje između tipa objekta i odgovarajućeg domena.

Pored navedenih konceptata postoje drugi koncepti MOV koji služe za semantički bogatiji opis realnog sistema:

Slabi objekat. Slab objekat je onaj koji zavisi od nekog drugog objekta i u sistemu ne može da se identifikuje nezavisno od tog objekta (*identifikaciona zavisnost*) [9]. Primer slabog tipa objekta je predmet (sl. 1). Na primer, ne radi se o istom pojavljivanju »matematike« za redovno i vanredno školovanje (pošto se radi o dva različita nastavna plana i programa). Ovo ima za posledicu da je tip objekta Predmet identifikaciono zavistan od tipa objekta NPP [10].

Generalizacija i specijalizacija (podtipovi). Generalizacija je apstrakcija u kojoj se skup sličnih tipova objekata predstavlja opštijim generičkim

Sl. 1 — Dijagram objekti-veze (DOV)

tipom (nadtipom) [9]. Ovaj vid apstrakcije u MOV ima eksplicitnu podršku. Na primer, specijalizacijom tipa objekta Slušalac dobili smo podtip Vanredni Slušalac (VANR SLUŠ). To znači da je svaki vanredni slušalac i slušalac (veza S) i da nasleđuje sve attribute svoga nadtipa (slušalac), kao i da može imati dodatne (vojna pošta). Naravno, u skladu sa ovim tipom apstrakcije podtip nasleđuje i sve veze i preslikavanja svog nadtipa. Definisanjem podtipova može se razrešiti problem tzv. opcionih atributa ($DG=0$) kao i u pojedinim slučajevima višeznačnih atributa ($GG>1$) [9].

Agregirani ili mešoviti tip objekat — veza. Ovaj tip podržava koncept agregacije u MOV. Ovaj koncept je implicitno uveden u MOV preko koncepta atributa (objekat je agregacija njegovih atributa). U ovoj verziji MOV problem višestrukih veza i atributa veza se razrešava upotrebom ovog koncepta (ova verzija modela dozvoljava samo binarne veze i nema koncept atributa veze). U našem primeru agregacija Položio (sl. 1) nastao je kao posledica postojanja atributa odnosa (veze) između Slušalaca i Predmeta.

Jezik za specifikaciju MOV (ERSL — Entity-Relationship Specification Language)

Osnovni cilj ovog jezika je da predstavlja izražajno sredstvo za definiciju i rukovanje podacima u okviru MOV. To znači da se kroz sintaksne konstrukcije jezika mogu staviti na raspolaganje koncepti ovog modela. Naravno, koncepti modela mogu biti skriveni kroz sintaksne konstrukcije jezika (u zavisnosti od toga šta se želi). Semantika jezika će odrediti preslikavanje sintaksnih konstrukcija jezika na koncepte modela. Dakle, pri razvoju jezika polazi se od samog modela, tj. od njegovih koncepata, struktura, ograničenja i operacija.

Kroz samu strukturu MOV izražavaju se i određena ograničenja, kao što je, na primer, kardinalnost preslikavanja. Zato se ova ograničenja moraju ugraditi kroz deo jezika za opis strukture (DDL — Data Definition Language). Održavanje, tj. zadovoljenje ovih ograničenja realizuje se kroz semantiku jezika za rukovanje (manipulaciju) podacima (DML — Data Manipulation Language). Jezičkim konstrukcijama za rukovanje podacima realizuju se, u stvari, operacije modela. Sada ćemo nešto više reći o operacijama MOV.

Operacije kojima se generišu tipovi objekata (sa svojim atributima) i tipovi veza su statičke prirode — njima se definiše struktura modela. Dinamičkim operacijama vrši se »oživljavanje« modela. Njima se tipovi objekata i veza tretiraju kao ekstenzije — skupovi objekata i veza. Dakle, ove operacije MOV definišu se nad objektima i vezama kao osnovnim konceptima ovog modela podataka. To su standardne operacije:

— *ubacivanje (kreiranje) objekta.* Ovom operacijom kreira se objekat koji pripada određenom tipu objekta. Argument za ovu operaciju je skup imenovanih vrednosti za odgovarajuće ključeve (jedan za regularni, a dva ili više za slabi ili agregirani objekat) i skup imenovanih vrednosti odgovarajućih atributa;

— *izbacivanje (uništavanje) objekta.* Ovom operacijom izbacuje se objekat, određenog tipa objekta, iz baze. Neophodan argument za ovu operaciju je jedinstveni identifikator (ključ) objekta;

— *ažuriranje objekta.* Ovom operacijom ažuriraju se vrednosti atributa objekta. Argument za ovu operaciju je identifikator objekta i skup imenovanih vrednosti koje predstavljaju vrednosti ažuriranih atributa;

— *selekcija objekata.* Ovom operacijom izdvajaju se objekti koji zadovoljavaju neki uslov. Dakle, ova operacija, kao argument, prihvata funkciju čija je funkcionalnost objekat $\rightarrow$ boo-

lean i na osnovu nje izdvaja objekte koji se evaluacijom ove funkcije preslikavaju u istinitosnu vrednost;

— *projekcija objekta*. Ova operacija je posebno pogodna za generisanje izveštaja. Možemo reći da ona izdvaja podskup atributa od tipa objekta koji je u ulozi operanda. Ako uzmemo da se interno formira nov tip objekta sa ovim podskupom atributa, onda se ovom operacijom vrši svojevrsna generalizacija tipa objekta. Kao argument, ova operacija prihvata listu atributa i specifikaciju objekta nad kojim se izvršava;

— *povezivanje dva objekta (kreiranje veze)*. Ova operacija, kao argument, prihvata specifikacije objekata koje treba povezati i naziv uloge jednog od specificiranih objekata u toj vezi;

— *razvezivanje dva objekta (izbacivanje veze)*. Ova operacija prihvata iste argumente kao i prethodna i vrši razvezivanje specificiranih objekata.

Pored ovih standardnih operacija MOV, uvodi se i operacija *nađi Referencu*, kojom se može pristupiti određenom atributu objekta (poseban slučaj operacije projekcije) ili kroz veze objekata, svim objektima koji se nalaze u zatvaraču veza datog objekta.

U jeziku kojim se želi zapisati model (konstruisati specifikacija sistema) moraju postojati sintaksne konstrukcije kojima će se izražavati napred rečeno, tj. koncepti, struktura, ograničenja i operacije modela. Pored toga, u jezik se često uvode i drugi, širi koncepti, kao što su npr. kontrolne strukture za upravljanje tokom izvršenja operacija, proceduralne apstrakcije, i dr. U daljem tekstu prikazuju se sintaksni elementi jezika za specifikaciju MOV (ERSL) i neformalno govori o njihovoj semantici.

Opis podataka (definicija strukture)

Tip objekta se deklariše koristeći sintaksnu konstrukciju.

entity ime = opis_tipa attributes:
lista_atributa

Njom se definiše tip objekta pod datim imenom. Definicija *opis_tipa* predstavlja sintaksnu konstrukciju za generisanje tipa objekta određene vrste. Sa konstrukcijom *lista_atributa* definišu se njegovi atributi.

U konstrukciji *opis_tipa* definišu se, tipovi objekata određene vrste sa svojim ključevima, koristeći odgovarajuću sintaksu. Tako:

— sa *regular with key*: A se generiše regularni (jaki) tip objekta sa ključem A. Na primer, sa:

entity Slušalac = *regular with key*:
Brind of number;
attributes:...

definiše se regularni tip objekta slušalac ključem Brind (tipa *number*).

— sa *subtype of ID* se generiše podtip od tipa objekta ID. Na primer, sa:

entity Vanr_Sluš = *subtype of*
Slušalac;
attributes:...

definiše se tip objekta Vanr_Sluš kao podtip od Slušalac sa dodatnim atributima.

— sa *weak of ID with key*: A generiše se slab objekat, identifikaciono zavisan od ID sa dodatnim ključem A koji ga identifikuje na nivou jednog pojavljivanja »nadređenog« objekta. Na primer, sa:

entity Zadatak = *weak of* Projekat
with key:
ŠifZad of number;
attributes:...

definiše se slab objekat Zadatak identifikaciono zavisan od Projekat sa dodatnim ključem ŠifZad.

— sa *aggregation of* (objekat₁ : uloga₁
(n₁, n₂);

objekat₂ : uloga₂
 (n₁, n₂);
 ...
 objekat_n : uloga_n
 (n₁, n₂);
)

generiše se agregirani tip objekta od agregirajućih objekata objekat₁ do objekat_n. Na primer sa:

entity Položio = *aggregation of*
 (Slušalac : Polaže(0,M);
 Predmet : PolaganOd(0,M)
)
attributes:...

definiše se agregacija Položio od objekata Slušalac i Predmet. Navođenje uloge u agregaciji je neophodno, jer je moguće da objekti istog tipa učestvuju u agregaciji, pa je jedini način njihovog razlikovanja preko uloge.

Dakle, konstrukcijom *opis-tipa* definiše se vrsta objekta i njegov ključ (prost ili složen — dodeljen od nadređenih ili agregirajućih objekata).

U svim ovim primerima, kao što je naznačeno, moguće je tipovima objekata dodati opise njihovih atributa.

Tip veze se deklariše koristeći sintaksnu konstrukciju:

relationship ime_objekta₁ : uloga₁ (n₁n₂)
invers ime_objekta₂ : uloga₂ (n₁n₂)

Njome se generiše tip veze između dva tipa objekta. Navođenjem uloga objekata u vezi omogućuje se uspostavljanje više različitih tipova veza između dva tipa objekta (ne obavezno različitih). Pri manipulaciji sa objektima, pomoću uloga se vrši i pristup pojedinim objektima koji su u vezi (npr. sa Slušalac. Sluša pristupa se predmetu — predmetima, koje Slušalac sluša). Na primer, sa:

relationship Slušalac:Sluša (1,M) *invers*
 Predmet:SlušanOd (0,M)

generiše se veza između objekata Slušalac (sa ulogom Sluša) i objekta pred-

met (sa ulogom SlušanOd). U definiciji veze navode se i kardinalnosti ova dva inverzna preslikavanja.

Rukovanje podacima (definisanje procesa i operacija)

Dinamika sistema (procesi) modeluje se sa operacijama nad modelom podataka. Već smo rekli da pojedine sintaksne konstrukcije jezika odgovaraju operacijama nad konceptima MOV. Kako je korisnik svestan koncepta objekata, veza, atributa objekata i njihovih pojavljivanja, treba mu omogućiti da kroz sintaksne konstrukcije jezika koristi odgovarajuće operacije modela. Operacije za generisanje tipova objekata i tipova veza već smo upoznali. Dinamičke operacije nad objektima i vezama koriste se pomoću sintaksnih konstrukcija koje slede.

Kreiranje objekta (pojavljivanje tipa objekta) izvršava se sledećom sintaksnom konstrukcijom:

create ID *with* (ključ₁ = vred₁
 ključ₂ := vred₂
 ...
 ključ_k := vred_k
 such that atribut₁ := vred₁;
 atribut₂ := vred₂;
 ...
 atribut_n := vred_n.

Tako, na primer, sa

create Slušalac *with* (Brind := 10190);
 such that Ime := »Marko«;
 Prezime := »Marković«;

kreira se Slušalac sa datim brojem indeksa (10190) kao sopstvenim identifikatorom (jak objekat) i ostalim navedenim atributima. Slabi objekti imaju složen ključ sastavljen od sopstvenog identifikatora i identifikatora nadređenog objekta. Tako, na primer,

create Zadatak *with* (ŠifProj := 77,
 ŠifZad := 88);

such that NazZad := »PBIS«;

OpisZad := »Projektovanje
bibliotečnog informacionog
sistema...«

kreira se slab objekat Zadatak (sa datom šifrom zadatka — 88) identifikaciono zavisan od objekta Projekat (sa šifrom projekta — 77) i datim atributima NazZad i OpisZad.

Naredbom *create* može se obezbediti da sistem sam generiše ekransku formu za unos vrednosti ključeva i ostalih atributa objekta u trenutku kada se naredba *create* izvršava. Ako korisnik u fazi pisanja specifikacije zadaje sve vrednosti atributa, kreiranje objekta će se obaviti bez ponovnog učešća korisnika. U suprotnom, sama operacija *create* će, ako za neki atribut nije zadata vrednost, zahtevati njen unos — interaktivno. Ako korisnik nije u stanju da obezbedi vrednosti za ove attribute, efekat operacije *create* na stanje baze se poništava.

Sledeća dinamička operacija MOV je operacija *izbaci objekat*, čija odgovarajuća sintaksna konstrukcija izgleda:

delete S

S (selektor) je skup objekata koji su izabrani po određenom kriterijumu (tj. koji zadovoljavaju neki uslov). Selektor je izvedeni sintaktički domen jezika za specifikaciju MOV. Njegovo značenje je slično onom u [6]. Može se posmatrati kao varijabla koja se koristi u komandama. U poređenju sa tipičnim programskim jezicima, glavna razlika je u tome što selektori mogu biti višeznačni, tj. imati više vrednosti (pojavljivanja objekata). Ova karakteristika čini jezik za specifikaciju snažnijim, ali, u drugu ruku, neki jezički konstrukti mogu biti neprirodni, nečitljivi ili rizični za upotrebu.

Dakle, semantika ove konstrukcije zavisi od vrednosti S. Ako je to prazan skup (null), onda se očigledno radi o grešci. U slučaju jednog objekta, zna-

čenje konstrukcije je operacija *Izbaci objekat*, koja datu objekat izbacuje iz baze. Ako se radi o grupi objekata (ova grupa objekata može se posmatrati kao izvedeni tip objekta čiji su oni pojavljivanja), tada se nad njom iterira operacija *Izbaci objekat*, koja će sve objekte iz grupe izbaciti iz baze. Naravno da se pri izbacivanju svakog pojedinog objekta mora voditi računa o ograničenjima (uslovima integriteta) i obezbediti da ne dođe do njihovog narušavanja.

Na primer, sa:

delete (Slušalac *where* Brind = 12190)

izbacuje se slušalac koji ima datu broj indeksa.

Ako izbacujemo objekat (objekte) iz nekog podtipa, onda se oni automatski izbacuju i iz svih odgovarajućih nadtipova i obrnuto. Na primer, sa:

delete (Vanr_Sluš *where* Voj_Poš = 7788)

izbacuju se svi vanredni slušaoci koji rade u dotičnoj vojnoj pošti. U ovom primeru se koristi operator »tačka« za zapis operacije *NadReferencu* kojom se postiže prirodna, logička navigacija u mrežnim bazama podataka. Za to se koristi sintaksna konstrukcija ID₁.ID₂, gde ID predstavlja neki identifikator (ime objekta, ili njegova uloga u vezi). Dakle, preko ID₁ se pristupa objektu ID₂.

Ažuriranje vrednosti atributa objekata vrši se sa:

update S

such that atr₁ := vred₁;

atr₂ := vred₂;

...

atr_n := vred_n.

Ova operacija prvo formira selektor S (jedan ili više objekata), a zatim se nad izdvojenim objektima iterira operacija ažuriranja objekta.

Tako, na primer, sa:

update (Slušalac *where* Brind = 10190)
such that Status = »V«)

se slušalac, sa brojem indeksa 10190, prebacuje se u status vanrednog slušaoca.

Operaciju MOV selekcija objekata smo, u stvari, već sreli u prethodnim operacijama. Ona se realizuje kroz formiranje selektora — S. Sintaksna konstrukcija ove operacije izgleda ovako:

ID *where* u

Semantika ove konstrukcije je da se od uslova *u* formira funkcija koja objekat preslikava u Bool vrednost. Ova funkcija se daje kao argument operaciji selekcije koja formira selektor (skup objekata) čiji objekti evaluiraju *true* vrednost, tj. ispunjavaju uslov *u*.

Na primer:

Nastavnik *where* (ImeNas = »Marko«
and
Učestvuje. NazProj =
= »PBIS«)

formira selektor kao skup nastavnika sa imenom »Marko« i učestvuju u projektu sa datim nazivom. Izdvojeni objekti mogu se prikazati sa:

print S.

Efekat ove komande je da se vrednost selektora S izračuna i prikaže korisniku. Tako će se npr. sa:

print Slušalac

prikazati sva pojavljivanja tipa entiteta Slušalac.

Sledeća dinamička operacija MOV je operacija projekcija objekta čija odgovarajuća sintaksna konstrukcija izgleda ovako:

select P *from* S

gde je P lista atributa objekata selektovanih sa S. Semantika ove operacije je interno formiranje tipa objekta sa atributima iz P čija pojavljivanja određuje selektor S. Prikazivanje ovih pojavljivanja korisniku vrši se komandom *print*. Tako se npr. sa:

print (*select* Brind, Ime, Prezime
from (Slušalac *where* Status = »R«))

prikazuju vrednosti navedenih atributa, svih redovnih slušalaca. Kada se žele prikazati svi atributi nekog objekta, umesto navođenja svakog atributa pojedinačno, moguće je koristiti (slično kao npr. u SQL-a) »*« sa istim dejstvom.

select * *from* S.

Pored operacija nad objektima, u MOV postoje operacije i nad vezama (mogu se shvatiti i kao operacije nad dva objekta). Prva od njih je *kreiranje veze* (povezivanje dva objekta). Odgovarajuća sintaksna konstrukcija je

connect ID *from* S₁ *to* S₂

ID je naziv uloge objekta/objekata specificiranih sa S₁ u vezi sa objektom/objektima specificiranih sa S₂. Jedna od restrikcija ove komande, a koja se ne vidi direktno iz ove sintaksne konstrukcije, jeste da ne sme i S₁ i S₂ u isto vreme specificirati više od jednog objekta. Naravno, povezivati se mogu samo objekti kojima je eksplicitno opisom strukture to dozvoljeno. Pri svakom uspostavljanju veza mora se voditi računa o kardinalnosti preslikavanja između objekata specificiranih sa S₁ i S₂. Sa

connect RadiU *from* (Radnik *where* šifra Radnika = 101) *to*
(Organizacija *where* šifOrg = 201)

se uspostavlja veza između radnika i organizacije sa datim šiframa.

Sa

connect RadiU *from* (Radnik *where* zanimanje = »programer« *to*
(Organizacija *where* naziv = »Energodata«)

se uspostavlja veza između svih radnika koji imaju zanimanje »programer« i organizacije sa datim nazivom (pretpostavka je da postoji samo jedna organizacija sa nazivom »Energodata«). Struktura modela dozvoljava veze između istih tipova objekata.

Tako se npr. sa:

```
connect SastavljenOd from (Proizvod
  where ŠifProiz = 333) to
  (Proizvod where ŠifProiz = 444)
```

uspostavlja veza između proizvoda sa datim šiframa. Pri kreiranju veze dovoljno je navesti ulogu samo jednog objekta u toj vezi jer se pretpostavlja ograničenje imenovanja da jedan objekat sa jednom ulogom ne može učestvovati u više od jedne veze sa drugim objektom (ovo ograničenje se proširuje na veze sa bilo kojim objektom — što pojednostavljuje upotrebu i implementaciju operacije *nađiReferencu*).

Druga operacija nad vezama je *izbacivanje veze* (razvezivanje objekata). Odgovarajuća sintaksna konstrukcija je:

```
disconnect ID from S1 to S2
```

Semantika ove konstrukcije je razvezivanje objekata specificiranih sa S₁ i S₂. Restrikcije iz prethodne operacije važe i ovde. Kod primene ove operacije mora se voditi računa o donjoj granici kardinalnosti. Gde dođe do njenog narušavanja traži se novi objekat i operacijom povezivanja objekata obezbeđuje se integritet. Ako se novi objekat ne obezbedi, efekat komande *disconnect...* se poništava (operacija *rollback*). Npr. sa

```
disconnect RadiU from (Radnik where
  zanimanje = »Programer«)
```

```
to (Organizacija where naziv =
  = »Energodata«)
```

brišu se veze uspostavljene u primeru iznad.

Rečeno je da je, pored operacija modela podataka, moguće uvesti i dodatne, kojima se proširuje model, a koje se, u suštini, mogu svesti na osnovne operacije modela. To je u skladu sa tendencijama da zajedno sa povećanjem aplikacija baza podataka dolazi do standardizacije programskih jezika orijentisanih ka bazama podataka [6]. Osnovne operacije modela i ove dodatne operacije mogu se zajednički nazvati komande (naredbe, instrukcije). Skup komandi koje se sekvencijalno izvršavaju mogu se grupisati u sekvencu i imenovati. Tako dolazimo do proceduralnih apstrakcija — procedura.

Procedura se deklariše sa: *proc* = *cs*, gde je *cs* niz komandi.

Sintaksa komande za poziv na izvršenje prethodno definisane procedure je:

```
exec ID,
```

gde je ID identifikator procedure.

Nezaobilazne naredbe u klasičnim programskim jezicima su i naredbe selekcije (grananja). One omogućuju kontrolu toka izvršenja komandi. U ERSL-u odgovarajuća sintaksna konstrukcija je:

```
if u then cs1 else cs2 end
```

ili

```
if u then cs1 end
```

Vrednost izraza *u* je tipa *boolean*. Ako je njegova vrednost logička istina, onda se izvršava niz komandi *cs₁*, inače *cs₂* u prvom slučaju, a u drugom slučaju se prelazi na izvršenje prve komande posle *if-then*.

U upitnim jezicima često se uvode *pomoćna imena* za određene podatke. Razlog za to je što se u nekom nizu komandi često pozivamo na određenu grupu podataka. Pomoćna imena mogu se, uslovno govoreći, tretirati kao varijable u programiranju. U ERSL-u ćemo koristiti tzv. *letblok* čija sintaksa je sledeća:

```
let ID = S in cs end
```

Za identifikator ID vezuje se vrednost dobijena evolucijom selektora S, tako da se na tu vrednost u nizu komandi *cs* možemo pozivati koristeći identifikator ID. Sledeći primer pokazuje njegovu upotrebu:

```
let X = (Radnik where Radnik.Plata >
>10000) in
  print (select Ime, Prezime, Plata
from X);
  update (X such that Plata := Plata-500)
end.
```

Sintaksu prikazanog jezika svakako treba proširiti sa konstruktima za definisanje (izvođenje) složenih objekata (pogleda) po uzoru na [11, 12].

Problem implementacije jezika za specifikaciju

Da bi se neki jezik implementirao mora, pre svega, biti definisana njegova sintaksa. U prethodnom poglavlju dat je opis jezika na osnovu koga se lako može definisati njegova sintaksa. Veći problem je, svakako, definisati formalno njegovu semantiku, koja se samo intuitivno naslućuje iz prethodnog opisa. Formalna definicija semantike ovakvog jezika je van domena ovog članka i u nastavku se samo obrazlaže predlog izbora denotacionog formalizma za njenu definiciju.

Generalno gledajući, semantika je relacija između konstrukata jezika i nekih drugih objekata. Za različite namene i pristupe ovi objekti mogu biti različiti: npr. oni mogu biti izrazi drugih jezika, akcije neke mašine ili apstraktni matematički objekti. Odatle slede i tri vrste semantike programskih jezika: aksiomska, operaciona i denotaciona. Za definiciju semantike jednog ovakvog jezika predlaže se denotaciona semantika kao možda najuspešnija teorijska alatka na polju programskih jezika, koja ima sledeće prednosti:

— značenja programa se predstavljaju preko semantičkih algebri. Zahvaljujući semantičkim algebrama struktura denotacione semantike je modularna. To omogućuje projektantima i korisnicima da proučavaju, dopunjuju i menjaju pojedine delove definicije bez razmatranja ostalih delova;

— pošto koristi matematičke objekte, funkcije i algebre, denotaciona semantika je dovoljno apstraktna, pa bi se jezik mogao proučavati nezavisno od načina implementacije ili mašine;

— formalna specifikacija može se neposredno implementirati jednostavnim preslikavanjem formalnih objekata u mašinske objekte (tj. objekte programskog jezika na kojem se implementacija vrši);

— rekurzivne definicije mogu biti jednostavno izražene.

Ovo su glavni razlozi zašto se denotaciona semantika sve više koristi za definisanje semantike programskih jezika, kao i koncepata baze podataka [5, 6, 7 i dr.]

Izbor programskog jezika u kojem će biti implementiran jezik za specifikaciju je važan, jer od njega zavisi koliko je implementacija teška. U tom smislu se za implementaciju predlaže neki od objektno-orijentisanih jezika (na primer, C++, Smalltalk, Actor i slični) koji će omogućiti prednosti primene objektnog programiranja i apstraktnih tipova podataka. Kasnija nadgradnja MOV i jezika za njegovu specifikaciju biće, svakako, olakšana upotrebom jednog jezika ovako visokog nivoa.

Problem parsiranja jezika može biti rešen upotrebom nekog generatora parsera (npr. YACC - Yet Another Compiler-Compiler) koji će generisati program koji vrši sintaksnu analizu. Dakle, umesto pisanja samog parsera, piše se specifikacija za njega, koja je, u stvari, opis gramatike jezika. Kasnije izmene i dopune sintakse jezika unose se u specifikaciju parsera na osnovu kojeg se generiše novi parser.

Predloženim putem implementacije mogao bi se realizovati interpreter jezika koji bi predstavljao alat za prototipski razvoj IS, tj. on bi na ulazu prihvatao specifikaciju IS, a na izlazu generisao prototip aplikacije.

Zaključak

U radu je prikazan pristup modelu objekt — veze kao izvršnoj specifikaciji. Istaknut je značaj ovog modela podataka u razvoju IS, a posebno potrebe njegove pune operacionalizacije kroz izvršnu specifikaciju. U vezi s tim, opisan je jezik za definiciju i rukovanje

podacima u okviru modela objekt — veze (jezik za specifikaciju). Na kraju je predložen i način implementacije ovog jezika. Implementaciju ovakvog ili sličnog jezika svakako bi trebalo uraditi, ne težeći, u prvo vreme, ka dobijanju posebno efikasnog sistema, već, pre svega, alata za generisanje brzog prototipa na kome bi se izvodilo testiranje i validacija. Tek kada bi prototip zadovoljio, prešlo bi se na stvarnu implementaciju (koja bi se mogla automatizovati konstrukcijom prevodioca, koji bi specifikaciju prevodio u neki efikasniji implementacioni model). Ovakav pristup vodio bi, svakako, ka većoj produktivnosti u razvoju IS.

Literatura:

- [1] P. P. Chen, »The Entity — Relationship Model — Towards a Unified View of Data«, ACM TODS, VOL. 1, No. 1, 1976.
- [2] P. P. Chen (Editor), »Entity — Relationship Approach to Software Engineering«, Anaheim, North-Holland, Amsterdam, the Netherlands, 1983.
- [3] S. Spaccapietra (Editor), »Entity — Relationship Approach : Ten Years of Experience in Information Modeling«, North-Holland, Amsterdam, the Netherlands, 1987.
- [4] M. Junet, »Design and Implementation of an Extended Entity — Relationship Data Base Management System ECRINS/86«, u /2/.
- [5] K. Subieta, M. Missala, »Semantics of Query Languages for The Entity — Relationship Model«, u /2/.
- [6] K. Subieta, »Semantics of Query Languages for Network Databases«, ACM TODS, VOL. 10, No. 3, September 1985.
- [7] K. Subieta, »Denotational Semantics of Query Languages«, Information Systems, Vol. 12, No. 1. pp. 69—82, 1987.
- [8] S. Nešković, »Semantički model podataka kao izvršna specifikacija«, magistarski rad, FON, Beograd, 1989.
- [9] B. Lazarević, »Prošireni model objekti-veze«, Interni materijal, FON, Beograd, 1990.
- [10] M. Majstorović, R. Terzić, »Jedan pristup projektovanju informacionih sistema na primeru segmenta IS Univerziteta Vojske Jugoslavije«, Vojnotehnički glasnik 1/93.
- [11] B. Lazarević, S. Nešković, »Opis složenih ulazno-izlaznih objekata«, interni materijal, FON, Beograd, 1992.
- [12] B. Lazarević, S. Nešković, »Objektivno-orientisani transformacioni razvoj jednog CASE sistema«, interni materijal, FON, Beograd, 1993.