UDK 681.3.06:91:007
Originalni znanstveni članak

RAZVOJ GIS-ORIJENTIRANIH APLIKACIJA U 4GL PROGRAMSKOM OKOLIŠU BAZA PODATAKA OBJEKTNI PRISTUP

Zdravko GALIĆ — Zagreb*

SAŽETAK: U radu je predočen jedan pristup razvoju GIS-orijentiranih aplikacija raspoloživom tehnologijom baza podataka i njihovih 4GL razvojnih okoliša, primjenom adekvatnih objektno-orijentiranih koncepata iz oblasti programskih jezika, baza podataka i softverskog inženjerstva. Za definiranje i ugradbu objekata definiranih od strane korisnika unutar sustava baze podataka, razmotren je koncept apstrakcije podataka, a sami objekti su implementacija apstraktnih tipova podataka. Pokazan je i način definiranja i implementiranja aplikacijski specifičnih operacija (funkcija) nad takvim objektima, definiranje korisničkih klasa i razvoj aplikacije u kojoj su sadržani svi razmatrani i predloženi koncepti.

1. UVOD

Korištenje računalske tehnologije za potrebe obrade i manipuliranja podacima prikupljenim geodetskim mjerenjima u gotovo svim organizacijama koje se bave takvim poslom, redovito uzrokuje probleme u izboru odgovarajućega softverskog proizvoda. Važan utjecaj na takvo stanje ima široka rasprostranjenost te tehnologije i spoznaja da je uvođenje novih informatičkih tehnologija jednostavno nužnost. Trenutačni se pristup najbolje očituje u tomu da se za takve potrebe uglavnom koriste softverski proizvodi namijenjeni za računalski podržano projektiranje (CAD), koji nisu niti dizajnirani imajući na umu njihovu isključivu uporabu u geodeziji, odnosno kartografiji. Kako svi takvi proizvodi raspolažu skupom predefiniranih operacija, korisnici često ne mogu riješiti ili teško rješavaju neke svoje specifične probleme. Uz to, proizvodi tog tipa ne posjeduju, odnosno veoma slabo podržavaju bilo kakvo sučelje prema bazama podataka, a posljedica je nedjelotvorno i neuniformno manipuliranje velikim skupovima podataka, prikupljenih geodetskim mjerenjima ili na neki drugi način.

Kada je to tehnički i ekonomski opravdano, koriste se geografski informacijski sustavi (GIS), čije je šire korištenje, uz visoke cijene na tržištu,

* Dr. Zdravko Galić, Geodetski fakultet, Kačićeva 26, Zagreb.

ograničeno i njihovim tehnološkim nedostacima. Naime, ti su se sustavi razvili iz ranih pokušaja upravljanja prostornim analizama koristeći kartografske podatke. Stoga je i većina GIS utemeljena na modelima naslijeđenima iz područja kartografije, CAD, itd., i može se ustvrditi da su uglavnom utemeljeni na paradigmi »karte-kao-grafika« a ne »karte-kao-baza-podataka« (Cook, 1989). Također, i pored pojave objektno-orijentiranih sustava baza podataka, kao što su GemStone (Servio, 1989), Ontos (Ontologic, 1900), Iris (Fishman, 1988), itd., korištenje relacijskih baza podataka u komercijalno raspoloživim GIS proizvodima je prividno univerzalno. Kako trenutačno dominira relacijska tehnologija, tako je i strukturirani upitni jezik (SQL) dominantan jezik za interakciju s bazama podataka. Svega nekoliko GIS proizvođača razvija prostorno-orijentirana proširenja SQL-a (npr. Computervision, 1990). Nažalost, temeljni upitni jezik obično je nekompletna implementacija ANSI/SQL standarda, a proširenja su minimalna i elementarna. Intenzivno se radi na SQL2 i SQL3 proširenjima koja sadrže određen broj objektno-orijentiranih koncepta, uključujući podršku apstraktnim tipovima podataka, metoda, operatora i funkcija. Takav pristup podršci apstrakciji podataka može omogućiti okvir u kojemu razvoj prostornih tipova podataka, prostornih operatora i funkcija ostaje potpuno unutar SQL standarda (ISO-ANSI, 1989).

Uz to što su skupi, većina današnjih GIS proizvoda je prije svega i teška za učenje i korištenje, složena i teška za prilagođivanje konkretnom aplikacijskom okolišu, a znanje stečeno u korištenju jednoga GIS proizvoda nije potpuno iskoristivo u primjeni nekoga drugoga. Glavni razlog su nestandardni modeli podataka, sredstva i alati za razvoj aplikacija. S druge strane, postaje jasno da se zbog potencijalne raznolikosti korištenja GIS tehnologije u raznovrsnim aplikacijama, mnogi zadaci tradicionalno prepušteni proizvođaču prebacuju u područje korisnika. To praktički znači da proizvođači ne moraju uložiti napor, u ponudu gotovih »rješenja«, nego ga preusmjeriti u razvoj sredstava koji korisnicima omogućuju rješavanje njihovih specifičnih problema. Za učinkovitiju primjenu GIS-a, korisnicima su potrebni i modeli podataka za učinkovito modeliranje i razvojni okoliši, koji proširuju i funkcionalnost sustava i načine na koji se primjenjuju (Dutton, 1991).

Svojedobno je zapaženo da je kôd većine aplikacijskih programa u području baza podataka komponiran na sljedeći način:

— manipuliranje ekranom	60%
— kontrolne strukture i računanje	15%
— interakcija s bazom podataka	25%

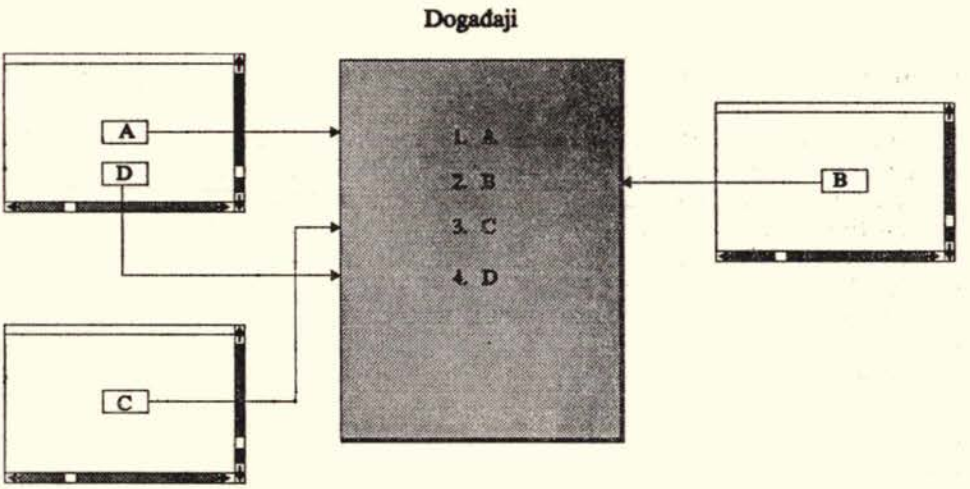
Bilo je jasno da za djelotvoran razvoj aplikacija korisnik mora raspolagati programskim okolišem, kojim se relativno lako i brzo razvijaju aplikacije. Rezultat istraživanja u tom smjeru jest pojava tzv. jezika četvrte generacije (4GL — 4th Generation Languages) (Stonebraker, 1986). Sve veća mogućnost razvoja grafičkih apikacija, raspoloživost grafičkih knjižnica (npr. OpenLook i OSF/Motif za Unix, DECWindows za VMS operacijski sustav, itd.) i njihovo korištenje iz programskih jezika 3. ili 4. generacije, omogućuje razvoj aplikacija za manipuliranje i prikaz podataka geodetskih mjerenja u

atraktivnom grafičkom obliku. Većinu aplikacija tog tipa moguće je realizirati i nekim od GIS proizvoda (Arc/Info, GeoSQL, System 9, itd.). Međutim, ako su to relativno jednostavnije aplikacije, dvojbena je ekonomska opravdanost investiranja u GIS.

Stoga se u ovom radu razmatra mogućnost razvoja takvih aplikacija (otuda i naziv GIS-orijentirane) korištenjem raspoloživih sustava za upravljanje bazama podataka i njihovih razvojnih okoliša, izborom i primjenom suvremenih, objektno-orijentiranih koncepata iz područja programiranja i programskih jezika, baza podataka i softverskog inženjerstva.

2. Windows4GL RAZVOJNI OKOLIŠ — OSNOVNI KONCEPTI

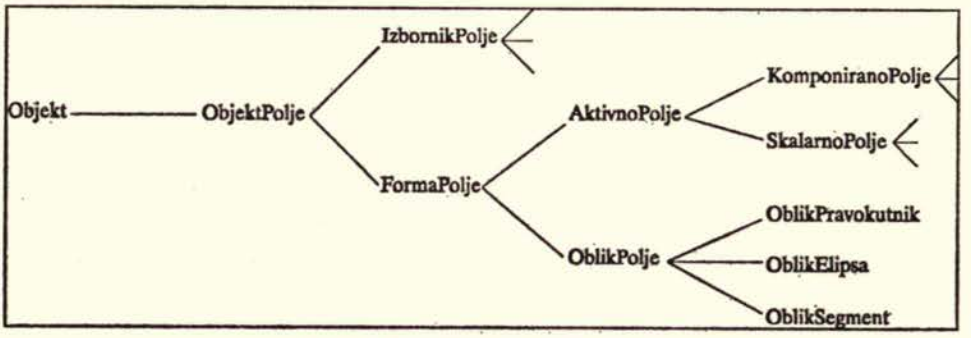
Aplikacije utemeljene na korištenju tzv. prozora i jezika 4. generacije (Windows4GL) grafičke su aplikacije, čiji su osnovni elementi okvir (frame), događaj (event) i objekt (object). **Okvir** se sastoji od oblika pokazanog u prozoru. Svakom okviru je pridružen programski kôd u jeziku 4. generacije, koji definira akcije u okviru. **Događaj** je akcija koju je pokrenuo korisnik, a svakom događaju je pridružen programski kôd, koji se izvršava kada ga korisnik inicira. Zapravo, u takvih aplikacija, kontrola odvijanja toka programa ne definira se internim strukturiranjem kôda, nego je vođena događajima. Svi događaji koji su inicirani smještaju se u tzv. rep (queue) događaja, čije je funkcioniranje utemeljeno na FIFO (First In First Out) konceptu (sl. 1).



Slika 1. Procesiranje događaja

Objekt je složena struktura podataka koja sadrži vrijednosti kojima možemo upravljati. Svaka komponenta aplikacije tog tipa je također objekt, i svaki takav objekt pripada sistemskoj klasi. Sistemska klasa objekta definira atribute objekta, kao i metode kojima je moguće manipulirati objektom.

Atributi sistematske klase kontroliraju **izgled** pojedine komponente i njeno *ponašanje. Metoda* je specijalna funkcija kojom se manipulira objektom dane klase. Sistematske klase su hijerarhijski organizirane i omogućuju korištenje *nasljeđivanja* (Stroustrup, 1986), (Egenhofer, 1989) važnoga koncepta objektno-orijentiranog programiranja. U toj hijerarhiji, klasa koja se nalazi »ispod« određene klase je *potklasa*, a ona koja je »iznad« određene klase je njena *superklasa*. Svaka klasa nasljeđuje atribute i metode svih njenih superklasa. Uz sistemske klase za komponente aplikacije, ta hijerarhija klasa uključuje i opće sistemske klase, koje služe isključivo za klasifikaciju i koje odgovaraju *apstraktnim klasama* u objektno-orijentiranom programiranju (npr. AktivnoPolje i ObjektPolje na slici 2) (Ingres, 1991).



Slika 2. Hijerarhija klasa (dio)

3. OBJEKTI — IMPLEMENTACIJA APSTRAKTNIH TIPOVA PODATAKA

U osnovi, apstraktni tip podataka (ATP) je *učahurenje* strukture podataka zajedno s kolekcijom odgovarajućih operacija (metoda) na toj učahurenoj strukturi, tako da implementacijski detalji nisu vidljivi za korisnike takvog tipa podataka (Cardelli, 1985), (Rowe, 1979). Koncept ATP intenzivno je korišten u sklopu semantičkog modeliranja podataka i implementiranja sustava baza podataka (Lockemann, 1979). U kontekstu relacijskih (Alagić, 1986) i objektnih baza podataka (Alagić, 1988), relacija je ATP, čiji su implementacijski detalji nevidljivi s razine aplikacijskog softvera. Odgovarajuće operacije su definirane u procedurama u programskom jeziku koji podržava i interakciju s bazom podataka i koncept ATP. Apstrakciju podataka u sklopu programskih jezika moguće je realizirati konceptom ATP (Liskov, 1977), (Meyer, 1988) ili modulima (Alagić, 1988), (Reiser, 1992). Razlika između ATP i modula je u tomu što moduli ne definiraju jedan tip. Modul omogućuje da se različiti tipovi, promjenljive i procedure definiraju i deklariraju zajedno, te koriste u aplikacijskom programu kao cjelina. Stoga se moduli koriste prije svega za specificiranje programskog sučelja prema bazama podataka (Rowe, 1979). Koncept ATP je pogodniji za probleme koji zahtijevaju višestruke primjerke jednog tipa, dok su moduli pogodniji za probleme koji zahtijevaju jednu kolekciju tipova, promjenljivih i procedura.

Radi proširenja semantičke moći sustava baza podataka (Stonebraker, 1986) u ovom radu ćemo pokazati pristup korištenju ATP, kao domene atributa objekata definiranih od strane korisnika. Pristup ćemo ispitati i ilustrirati na primjeru razvoja aplikacije kojom se obrađuju podaci geodetskih mjerenja dobivenih elektroničkim tahimetrom. Rezultati te obrade (koordinate detaljnih točaka) pohranjuju se u bazu podataka, uz generiranje vektorskoga grafičkog prikaza snimljenih objekata i mogućnošću manipuliranja tim objektima.

Za svaki novi objekt (ATP) definira se njegovo ime, potreban memorijski prostor za njegovu reprezentaciju, jedinstveni identifikator i pokazivači na C funkcije, kojima se manipulira tim novim tipom. Funkcije su sastavni dio definicije tipa podataka, a za potrebe integracije svakoga novog objekta unutar sustava baza podataka, implementiraju se obvezne funkcije, koje se mogu podijeliti u četiri skupine:

— *zajedničke funkcije*

— compare usporedba dva ATP
— length_check provjera duljine
— getempty kreiranje »praznog« ATP
— value_check provjera važećih vrijednosti
— minmaxdv najmanja i najveća vrijednost

— *funkcije za indeksiranje*

— keybuild izgradnja ključa za ISAM, BTree i HASH indekse
— hashprep priprema za HASH ključ

— *ulazno/izlazne funkcije*

— convert pretvorba
— dbtoev određivanje eksternog tipa za konverziju
— tmcvt pretvorba u format za prikazivanje
— tmlen duljina tipa za SQL terminalski monitor

— *funkcije za optimiziranje upita*

— helem kreiranje vrijednosti za histogram
— hmin vrijednost histograma za minimalnu vrijednost
— hmax vrijednost histograma za maksimalnu vrijednost
— dhmin predefinirana minimalna vrijednost za histogram
— dhmax predefinirana maksimalna vrijednost za histogram
— hg_dtln tip i duljina podataka za histogram

Osim tih, za svaki ATP, odnosno objekt, moguće je definirati funkcije i operatore i pridružiti im nazive, odnosno simbole koji su već definirani za neke objekte. Tako npr., ako korisnik definira objekt *poligon*, može definirati operacije unije, presjeka i razlike poligona i pridružiti im operatore »+«, »/« i »—«. Na taj način je u potpunosti moguće koristiti *preopterećenje* operatora, odnosno *polimorfizam*, još jednu od temeljnih osobina objektnog programiranja (Cardelli, 1985), (Stroustrup, 1986). Tako definirane funk-

cije i operatore, moguće je koristiti u sklopu SQL-a, na svim onim mjestima gdje njegova sintaksa dopušta korištenje standardnih funkcija i operatora. Ako za ATP *poligon* definiramo operator »# ||«, koji kao rezultat vraća istinitu vrijednost ako su dva operanda (poligona) susjedni, tada je rezultat upita (2) nad relacijom (1):

PARCELA (Parcela_Id, Poligon, Povrsina) (1)

SELECT Parcela_Id
FROM PARCELA (2)
WHERE Poligon # || *parcela_id.poligon*

skup brojeva svih parcela koje su susjedne nekoj parceli.

Implementacija ATP je ostvarena pisanjem knjižnice u programskom jeziku C. Polja strukture ATP_DT_DFN sadrže osnovne informacije neophodne za definiranje ATP za sustav baze podataka.

```
/*                                                              */
/* ATP.H   Zaglavlje definicije ATP definirih od korisnika      */
/*                                                              */
. . .
typedef struct _DT_NAZIV {
#define MAX_NAZIV 24
 char naziv[MAX_NAZIV]
} DT_NAZIV

typedef struct _ATP_DT_DFN {
 int dtd_object_type;
#define O_DATATYPE 0x0210
 DT_NAZIV dtd_name;                  /* naziv ATP            */
 DT_ID    dtd_id;                    /* identifikator ATP    */
 int      (*dtd_lenchk_addr)();      /* pokaz. na funkciju   */
 int      (*dtd_compare_addr)();
 int      (*dtd_keybuild_addr)();
 . . .
} ATP_DT_DFN
 . .
/* informacije o primjercima funkcija: */

typedef struct _ATP_F_DFN {
 int      fod_object_type;
#define   O_OPERATION 0x0211
 DT_NAZIV fod_name;             /* naziv funkcije/operatora */
 OP_ID    fod_id;               /* identifikator funkcije   */
 int      fod_type;             /* funkcija ili operator ?  */
#define   NORMAL ((int) 4)
} ATP_F_DFN
. . .
```

ATP, odnosno njegovi atributi, obvezne funkcije, kao i funkcije i operacije koje se odnose samo na taj ATP, definiraju se u C knjižnici na sljedeći način:

```
/*                                                    */
/* TOCKA.C          implementacija ATP TOCKA */
/*                                                    */
#include <ATP.H>
#include <math.h>
#include <stdio.h>
. . .
typedef struct _TOCKA
{
 double y;
 double x;
} TOCKA;
. . .
/* obavezne funkcije za integraciju ATP TOCKA unutar baze */
int t_compare (scb, op1, op2, rezultat)
. . .
int t_length_check (scb, user_specified, dv, dv_rezultat)
. . .
int t_keybuild (scb, key_block)
. . .
/* funkcije i operatori samo za ovaj ATP */

int t_duljina (scb, dv_1, dv_2, dv_rezultat)
SCB *scb;
DATA_VALUE *dv_1, *dv_2, *dv_rezultat;
{
 int rezultat = ERROR;
 double duljina;
 double x1, y1, x2, y2;
 TOCKA *op_1, *op_2;
 if ((dv_1->db_datatype == op_datatype_id) &&
     (dv_2->db_datatype == op_datatype_id)&&
     (dv_rezultat->db_datatype == FLOAT) &&
     (dv_rezultat->db_data))
 {
  op_1 = (TOCKA *) dv_1->db_data;
  op_2 = (TOCKA *) dv_2->db_data;
  y1 = op_1->y;  x1 = op_1->x;
  y2 = op_2->y;  x2 = op_2->x;
  duljina = sqrt (
                  pow((double)(y2 - y1), (double) 2.0) +
                  pow((double)(x2 - x1), (double) 2.0)
                 );
  *dv_rezultat->db_data = duljina;
  rezultat = OK;
 }
 return (rezultat);
}
. . .
int t_smjerni_kut (scb, dv_1, dv-2, dv_rezultat)
. . .
int t_y (scb, dv_1, dv_rezultat)
. . .
int t_x (scb, dv_1, dv_rezultat)
. . .
}
```

Radi razvoja konceptualnog modela podataka, klasificirat ćemo relevatne objekte u kolekcije tipova objekata. Tip objekta je skup objekata koji imaju iste osobine, odnosno atribute, i čija je važnost definirana namjenom modela, odnosno aplikacije. Klasifikacija se primjenjuje specificiranjem atributa skupa objekata, koji pripadaju istom tipu, tj. klasi. Zapravo, tip objekta je definiran kao agregacija njegovih atributa (Galić, 1991), koju u apstraktnoj formi i C notaciji možemo napisati kao:

```
typedef struct {
  Objekt1 atribut1;
  Objekt2 atribut2;                    (3)
  ...
  Objektn atributn;
} OBJEKT
```

Formalno, skup OBJEKT može se definirati na sljedeći način:

$$\text{OBJEKT} = \text{Objekt}_1 \times \text{Objekt}_2 \times \ldots \times \text{Objekt}_n \qquad (4)$$

tj. kao Kartezijev umnožak skupova Objekt_1, Objekt_2, ..., Objekt_n.

Primjerak tipa Objekt je *n-torka* $(a_1, a_2, \ldots, a_n)$ vrijednosti atributa tog primjerka:

$$\text{Objekt} = \{(a_1, a_2, \ldots, a_n) \parallel ai \in \text{Objekt}_i,\ i = 1, 2, \ldots, n\}$$

U sklopu korištenog modela podataka, različiti objekti istog tipa moraju biti reprezentirani različitim n-torkama vrijednosti njihovih atributa. Da bi taj uvjet bio zadovoljen, dostatno je da n-torke imaju različite najmanje vrijednosti jednog atributa. Takav atribut, ili skup atributa, zapravo je identifikator, odnosno ključ objekta. Relevantni objekti u razmatranoj aplikaciji su detaljne točke i parcele:

```
CREATE TABLE DTocka (DTocka_ID Integer, Pozicija Tocka)
CREATE TABLE Parcela (Parcela_ID Char(10), Povrsina Float8)        (5)
```

Svaka parcela je definirana s *N* detaljnih točaka, a svaka detaljna točka može pripadati ne samo jednoj, nego u općem slučaju *M* parcela. Očevidno, među objektima *Parcela* i *DTocka* postoji *M : N* odnos. Taj tip odnosa reprezentovaćemo agregacijom, uvođenjem asocijativnog tipa objekta *Parcela_DTocka:*

```
CREATE TABLE Parcela_DTocka
            (Parcela_ID Char(10), RedBr Integer, DTocka_ID Integer)   (6)
```

Uočimo da objekt *DTocka* ima atribut *Pozicija*, čija je domena prethodno definirani ATP *Tocka*. Za objekt *DTocka* moguće je kreirati i indeks nad atributom čija je domena ATP:

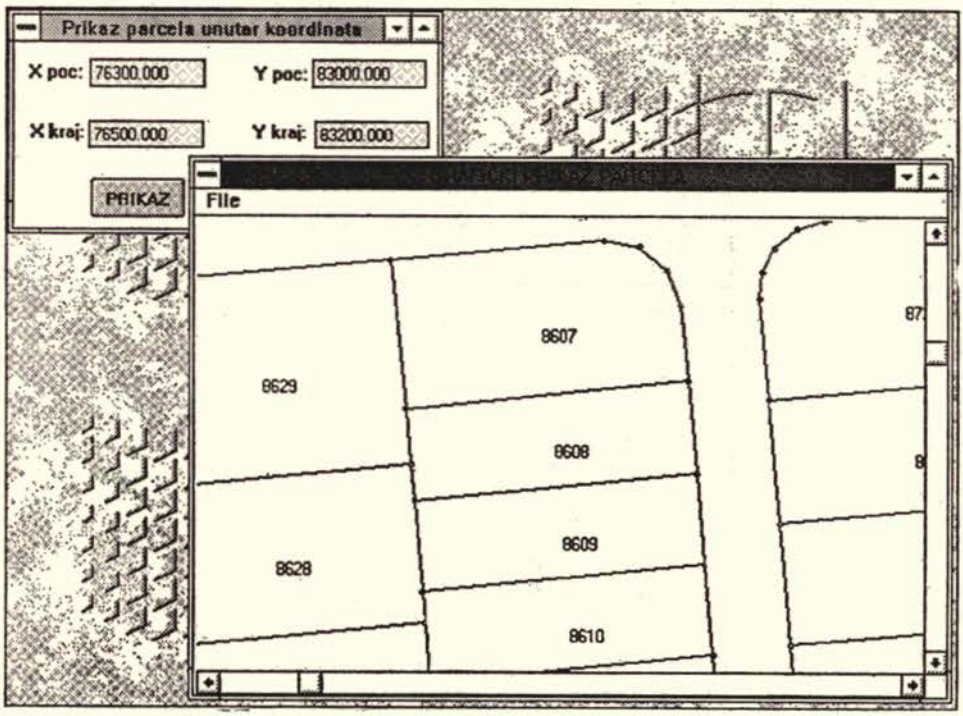
```
CREATE INDEX Ind_DTocka ON DTocka (Pozicija)
WITH STRUCTURE = BTree                                             (7)
```

U objektno-orijentiranom sustavu baza podataka, svi objekti pripadaju nekoj od klasa, a na vrhu hijerarhije klasa se definira tzv. *meta* klasa *Object* (Servio, 1989), (Stonebraker, 1987). Objekti mogu nasljeđivati atribute i metode od drugih klasa, i osnovna razlika (kao i u programskih jezika) jest u tomu da li je nasljeđivanje *jednostavno* (objekt ima samo jednu superklasu) ili je *višestruko* (objekt nasljeđuje osobine i metode više klasa). U našem je slučaju moguće da u sklopu aplikacije korisnik definira objekt koji nasljeđuje atribute i od ATP i od predefiniranih objekata od sustava:

```
USER CLASS
{
 int            Id;
 int            Red_Br;
 Tocka          Koord;     /* ATP */
 EllipseShape   Oznaka;    /* sistemska klasa */
} Det_Tocka
```

7. PRIMJER RAZVOJA APLIKACIJE

Integracija svih razmotrenih koncepata realizirana je za potrebe opisane aplikacije, kojom se predočuju vektorski grafički objekti. Slika 3. je rezultat izvršenja 4GL programskoga kôda na osobnom računalu s MS Windows



Slika 3. Vektorski grafički prikaz

grafičkim sučeljem, povezanim DECnet mrežom, s računalom MicroVAX 3100 na kojemu se nalazi baza podataka.

```
INITIALIZE (parcela_id=c10 not null,
            t1=Tocka, t2=Tocka,
            pret_t=Tocka, prva_t=Tocka,
            tekst=EntryField, tekst_t=Tocka,
            DT=Det_Tocka, linija=SegmentShape,
            l=integer not null) =
BEGIN
     DT = Det_Tocka.Create();  tekst_t = Tocka.Create()
     pret_t = Tocka.Create(); prva_t = Tocka.Create();

     FIELD(Prozor.Slika).Height = M * (t2.y-t1.y);
     FIELD(Prozor.Slika).Width  = M * (t2.x-t1.x);

     SELECT :parcela_id=p_dt.parcela_id,
          :tekst_t.x=(max(dtoc.x)+min(dtoc.x))/2-:t1.x,
          :tekst_t.y=(max(dtoc.y)+min(dtoc.y))/2-:t1.y
     FROM parcela_dtocka p_dt, dtocka dtoc
     WHERE dtoc.x > :t1.x AND toc.x < :t2.x
     AND dtoc.y > :t1.y AND toc.y < :t2.y
     AND dtoc.dtocka_id = p_dt.dtocka_id
     GROUP BY p_dt.parcela_id
     BEGIN
     l = 0;
     REPEATED
     SELECT :DT.id=dt.dtocka_id, DT.red_br=redbr,
          :DT.Koord.x=x-:t1.x, :DT.Koord.y=y-:t1.y
     FROM dtocka dt, parcela_dtocka pdt
     WHERE dt.dtocka_id = pdt.dtocka_id
     AND pdt.parcela_id = :parcela_id
     ORDER BY red_br
     BEGIN
          DT.Oznaka.SetAttribute (
               AbsXLeft = M * T.Koord.x - 15,
               AbsYTop = M * T.Koord.y - 15,
               Height = 30,
               Width = 30,
               CurBias = FB_CLICKPOINT;
               ParentField = FIELD(Prozor.Slika));
          l = l + 1;
          IF l = 1 THEN
               prva_t = DT.Koord.Duplicate();
          ELSE
               linija = SegmentShape.Create();
               linija.SetAttribute (
                    Point1X = M * pret_t.x,
                    Point1Y = M * pret_t.y,
                    Point2X = M * T.Koord.x,
                    Point2Y = M * T.Koord.y,
                    ParentField =
                         FIELD(Prozor.Slika));
          ENDIF;
          pret_t = DT.Koord.Duplicate();
          DT = Det_Tocka.Create();
     END;
     IF l > 2 THEN
          linija = SegmentShape.Create();
```

```
            linija.SetAttribute (
                  Point1X = pret_t.x,
                  Point1Y = pret_t.y,
                  Point2X = prva_t.x,
                  Point2Y = prva_t.y,
                  ParentField = FIELD(Prozor.Slika));
      ENDIF;
      tekst = EntryField.Create();
      tekst.SetAttribute (
            XLeft = M * tekst_t.x,
            YTop  = M * tekst_t.y,
            Height = 70,
            Width = 300,
            FormatString = 'c'+text(length(:parcela_id)),
            OutlineWidth = LW_NOLINE,
            BgColor = FIELD(Prozor.Slika).BgColor,
            CurBias = FB_LANDABLE,
            TextValue = :parcela_id,
            ParentField = FIELD(Prozor.Slika));
      END;
      COMMIT;
END
```

5. ZAKLJUČAK

Ako pažljivo analiziramo programski kôd, uočavamo da su njegova složenost i veličina drastično smanjene u odnosu na uobičajenu složenost i veličinu kôda napisanog u nekom od programskih jezika 3. generacije (Pascal, C, C++, ...). Na taj način, omogućen je jednostavan i djelotvoran razvoj i održavanje tako razvijenih aplikacija. Raspoloživost SQL-a, kao podskupa unutar 4GL-a, omogućuje učinkovitu, eksplicitnu, i što je veoma važno (pogotovu u odnosu na uporabu CAD proizvoda), *standardiziranu* interakciju s bazom podataka. Razmotreni objektni pristup (prije svega implementacija ATP, nasljeđivanje i polimorfizam), uz pružanje mogućnosti korisniku da u cijelosti razvije aplikaciju po svojoj potrebi, omogućuje djelotvorniji i lakši razvoj novih aplikacija koje koriste iste klase objekata. Time je podržan, danas veoma važan, zahtjev softverskog inženjerstva za ponovno korištenje već razvijenog softvera za razvoj novih aplikacija *(reusability).*

Uz te jasne prednosti takvog pristupa u odnosu na korištenje jezika 3. generacije, CAD, a u stanovitoj mjeri i *makro* jezika nekih GIS proizvoda, očevidno je da impelementacija objekata, tj. ATP nije trivijalna. Potencijalni korisnik mora, uvjetno rečeno, potpuno poznavati tri tehnologije: programski jezik 3. generacije, baze podataka i programski jezik 4. generacije. U svakom slučaju, izloženi pristup potvrđuje da pažljivo izabrani i adekvatno primijenjeni objektni koncepti iz područja programskih jezika, baza podataka i softverskog inženjerstva, pružaju mogućnost djelotvornog projektiranja i razvoja GIS-orijentiranih aplikacija, korištenjem već raspoložive tehnologije baza podataka i njihovih razvojnih okoliša.

6. ZAHVALA

Zahvaljujem Ministarstvu znanosti Republike Hrvatske na financijskoj potpori ovih istraživanja u sklopu projekta »Osnovni geodetski radovi prostornog informacijskog sustava Republike Hrvatske«, posebice kolegama s Geodetskog fakulteta Sveučilišta u Zagrebu: prof. dr. A. Bilajbegoviću, prof. dr M. Solariću i prof. dr. N. Solariću, koji su, svaki na svoj način, inicirali i omogućili istraživanja predočena u ovom radu, Jasminu Nakiću, dipl. inž. el. na testiranju kôda i tvrtki »EuroComputer Systems« iz Zagreba, ovlaštenom distributeru »Digital Equipment Corporation«, koja je stavila na raspolaganje sve potrebne računalske resurse.

LITERATURA

Alagić, S. (1986): Relational Database Technology, Springer-Verlag, New York.

Alagić, S. (1988): Object-Oriented Database Programming, Springer-Verlag, New York.

Beech, D. (1988): A Foundation for Evolution from Relational to Object Databases, Advances in Database Technology — EDBT, Proceedings of the International Conference, Venice.

Cardelli, L., Wegner, P. (1985): On Underştanding Type, Data Abstraction, and Polymorphism, ACM Computing Surveys, 17 (4), 471—552.

Computervision (1990): System 9 Query Language Manual, Zurich.

Cooke, D. (1989): TIGER and the »Post-GIS« Era, GIS World, Vol. 2, №. 4.

Dutton, G. (1991): Improving Spatial Analysis in GIS Environments, Technical Papers, ACSM—ASPRS Annual Convention, Vol. 6, Auto-Carto 10, Baltimore, 168—185.

Egenhofer, M., Frank, A. (1989): Object-Oriented Modeling in GIS: Inheritance and Propagation, Proceedings Auto-Carto 9, 9th International Conference on Computer-Assisted Cartography, Baltimore, 588—598.

Fishman, D. L., et al. (1988): Overview of the Iris DBMS, Hewlett-Packard Laboratories, Palo Alto.

Galić, Z. (1991): Prilog efikasnom izravnanju geodetskih mreža u objektno-orijentisanoj programskoj okolini baza podataka, Doktorska disertacija, Građevinski fakultet, Sarajevo.

Ingres Corporation (1991): Object Management User's Guide for the UNIX and VMS Operating Systems, Alameda, California.

ISO-ANSI (1989): Database Language SQL2 and SQL3 (ISO-ANSI working draft) X3H2-89-252 and ISO DBL FIR-3.

Liskov B., et al. (1977): Abstraction Mechanisms in CLU, Communications of the ACM, Vol. 20, №. 8, 564-576.

Lockmann P., et al. (1979): Data Abstraction for Data Base Systems, ACM Transactions on Database Systems, Vol. 4, №. 1.

Meyer, B. (1988): Object-Oriented Software Construction, Prentice-Hall.

Ontologic, Inc. (1990): Ontos, Product Reference, Burlington, Massachusetts.

Reiser M., Wirth, N. (1992): Programming in Oberon, Addison-Wesley Publishing Company, Reading, Massachusetts.

Rowe L., Shoens K. (1979): Data Abstractions, Views and Updates in RIGEL, Proceedings of the ACM-SIGMOD Conference on the Management of Data.

Servio Logic Development (1989): GemStone, Product Reference, Portland, Oregon.

Stonebraker M. (1986): The INGRES Papers — Anatomy of a Relational Database System, Addison-Wesley Publishing Company, Reading, Massachusetts.

Stonebraker M., Rowe L. (1987): The POSTGRES Data Model, The POSTGRES Papers, Memorandum №. UCB/ERL M86/85, University of California, Berkeley.

DEVELOPMENT OF GIS-ORIENTED APPLICATIONS IN 4GL DATABASE PROGRAMMING ENVIRONMENT OBJECT APPROACH

This paper presents an approach to design and implementation of GIS--oriented applications using currently available database technology and 4GL developming environment, exploring the modern concepts of object--oriented paradigm (programming languages, databases, and software engineering). Abstract data types are used for the purpose of user-defined object implementation. Definition and specification of the specific operations for user-defined types, classes, and application development, based on these conceptual apstractions is also shown.

Primljeno: 1993-05-21