UNIVERZITET U NOVOM SADU
PRIRODNO MATEMATIČKI FAKULTET
INSTITUT ZA MATEMATIKU

mr Mirjana Ivanović

PRILOG RAZVOJU PROGRAMSKIH JEZIKA KORIŠĆENJEM OBJEKTNO ORIJENTISANE METODOLOGIJE

- DOKTORSKA DISERTACIJA -

Novi Sad, 1992.

Sadržaj

Slike

Predgovor

Razvojem tehnike i tehnologije računarske nauke postaju jedna od najznačajnijih oblasti ljudskog stvaralaštva i delatnosti. One su jedinstven spoj teorijskih rezultata i praktičnih proizvoda i teško je razgraničiti da li teorijski rezultati uslovljavaju razvoj i implementaciju praktičnih proizvoda ili praktični rezultati uslovljavaju uspostavljanje novih i dopunjavanje postojećih teorija. U svakom slučaju bliska interakcija teorije i prakse vodi sve većem i bržem razvoju računarskih nauka.

Računarske nauke pokrivaju čitav niz različitih pravaca i disciplina kako u oblasti hardvera tako i u oblasti softvera.

Najznačajniji pravci u oblasti softvera su:

- razvijanje i implementacija programskih jezika,
- razvijanje integrisanih sistema za reprezentaciju informacija i znanja,
- razvijanje metoda veštačke inteligencije.

Najnovija faza u oblasti obrade informacija i podataka na računaru karakteriše se kao faza formalizacije (reprezentacije) i korišćenja (prezentacije) informacija i znanja.

Razvijanje različitih metoda i tehnika u ovoj oblasti uslovljava pojavu mnoštva integrisanih sistema koji omogućavaju reprezentaciju i prezentaciju informacija.

U poslednje vreme sve je češće uvodjenje inteligentnih funkcija u ovakve sisteme. Medjutim, veštačka inteligencija još uvek nije dostigla dovoljan stepen razvoja da bi mogla u potpunosti da podrži opšte inteligentne sisteme za reprezentaciju i prezentaciju informacija, a pre svega znanja. Za sada postoje samo skromni rezultati u razvijanju inteligentnih sistema, krajnje specijalizovane namene. Ako je sistem pogodan za primenu u nekoj uskoj oblasti to se više može učiniti inteligentnim, i obrnuto, ako je sistem opštije prirode, to je teže učiniti ga inteligentnim.

Druga tendencija u ovoj oblasti je primena i uključivanje multimedijalnih elemenata, tj. razvoj sistema koji integrišu računarski memorisane informacije sa zvukom, videom, grafikom, animacijom i slično.

Bez obzira da li su multimedijalni ili ne, sistemi za reprezentaciju i prezentaciju informacija se mogu podeliti u nekoliko karakterističnih grupa [Ivanović, 1991b].

1) **HyperText sistemi** predstavljaju softverska orudja za prikupljanje, uskladištavanje, pretraživanje i prezentaciju informacija sa referencama. Ovi sistemi su u osnovi opšte namene i mogu se primenjivati u različitim oblastima. Informacije koje treba reprezentovati se dele u manje celine. Ove celine se medjusobno vezuju referencama i proces korišćenja HyperText-a predstavlja kretanje iz jedne celine u drugu.

2) **Autorski sistemi** su sistemi namenjeni procesima učenja. Znanje i informacije koje treba da budu prezentovane dele se u manje medjusobno povezane celine koje formiraju bazu čvorova. U procesu korišćenja sistema vrši se kretanje po različitim putevima u sistemu od jednog čvora da drugog, često, uz kontrolisanje savladanog gradiva.

3) **Inteligentni tutorski sistemi (ITS)** su računarski programi koji koriste tehnike veštačke inteligencije da bi što uspešnije i kvalitetnije podržali proces učenja u nekoj specijalizovanoj oblasti.

4) **Inteligentni interfejsi** su sistemi koji koriste metode veštačke inteligencije u obezbedjivanju inteligentne komunikacije čovek - računar. Oni prihvataju informacije od aplikativnog programa, prevode ih u pogodan oblik (tekst, grafika, zvuk) i prezentiraju ih korisniku. Sa druge stane prihvataju akcije od korisnika, prilagodjavaju ih i prosledjuju aplikativnom programu.

Analizom karakteristika pomenutih sistema uočena su osnovna svojstva a zatim je definisano karakteristično jezgro koje leži u njihovoj osnovi.

Uočene karakteristike sistema su iskorišćene za razvijanje formalizma tj. pogodnih struktura kako za reprezentaciju tako i za prezentaciju informacija, korišćenjem objektno orijentisane metodologije. Predložene strukture omogućavaju memorisanje različitih pojavnih oblika informacija (tekst, slika, animacija, zvuk), komponovanje struktura i njihovo medjusobno vezivanje u multidigraf nekog domena reprezentacije. Uočeno je da takav multidigraf leži u osnovi svih pomenutih sistema.

Predloženi formalizam je poslužio kao osnova i polazni element za kreiranje odgovarajućeg programskog jezika *Less*. Programski jezik se sastoji od niza direktiva kojima se kreira i popunjava svaki pojedinačni element multidigrafa, a takodje odredjuju i uspostavljaju veze medju njima.

Osnovne strukture i odgovarajući programski jezik, koji podržavaju ključne principe objektno orijentisane metodologije i programiranja kao što su klase, objekti i nasledjivanje, predstavljaju originalne rezultate.

Objektno orijentisana metodologija i programiranje je jedan od trenutno najinteresantnijih i najznačajnijih pravaca u oblasti razvijanja i primene programskih jezika. Objektno orijentisano programiranje uobličava postojeće elemente programiranja, a uvodi i niz novih koncepata koji ga odvajaju od klasičnih programskih jezika. Objektno orijentisano programiranje je stil programiranja u kojem je ceo sistem skup objekata koji medjusobno dejstvuju.

Ova svojstva objektno orijentisane metodologije su iskorišćena tako da se svaki element multidigrafa može posmatrati kao jedan objekat (pojava neke odgovarajuće klase), a programski jezik omogućava rukovanje kako ovim objektima tako i celokupnim multidigrafom.

Predložene ugradjene strukture (klase), mehanizam njihovog komponovanja i izgradnje novih klasa u okviru programski jezik *Less* omogućavaju generisanje integrisanih sistema za reprezentaciju i prezentaciju informacija tipa HyperText-a, autorskog sistema, a uz nešto više napora i jednostavnijih inteligentnih tutorskih sistema.

Osnovni doprinosi ove disertacije se mogu prikazati kroz nekoliko aspekata:

- analiza postojećih sistema za reprezentaciju i prezentaciju informacija i uočavanje osnovnih zajedničkih elemenata,

- kreiranje osnovnog koncepta tj. ugradjenih struktura i mehanizama komponovanja složenijih struktura,

- kreiranje objektno orijentisanog programskog jezika prilagodjenog predloženim strukturama, koji omogućava definisanje multidigrafa domena reprezentacije, tj. realizaciju sistema za reprezentaciju i prezentaciju informacija. Predloženi jezik predstavlja:

  - osnovu za formalizaciju sistema za reprezentaciju i prezentaciju informacija,
  - apstraktni mašinski jezik za kreiranje sistema za reprezentaciju i prezentaciju informacija, koji omogućava lakšu prenostvost multimedijalnih aplikacija na druge računare i operativne sisteme,
  - orudje za lakšu i jednostavniju implementaciju konkretnih aplikacija.

Osim ovih osnovnih teoretskih rezultata, otvara se i niz novih teoretskih i praktičnih mogućnosti za dalji rad:

- implementacija programskog jezika *Less*,
- implementacija jezgra HyperText i autorskih sistema,
- uključivanje metoda veštačke inteligencije u sisteme iz prethodne tačke,
- obogaćivanje multimedijalnog aspekta osnovnog koncepta.

Disertacija se sastoji od šest poglavlja:

1) Sistemi za (re)prezentaciju informacija i znanja,
2) Objektno orijentisana metodologija,
3) Struktura za reprezentaciju i prezentaciju informacija,

4) Osnove jezika *Less*,
5) Konkterni primeri generisani nad opštom strukturom i jezikom *Less*,
6) Zaključak.

U prvom poglavlju je dat pregled i izdvojeni su osnovni elementi postojećih sistema za reprezentaciju i prezentaciju informacija tipa HyperText, autorski sistemi, inteligentni tutotski sistemi, inteligentni interfejsi i sistemi za automatsku prezentaciju. Prikazani su osnovni elementi multimedijalnih sistema. Originalni doprinos dat u ovom delu je predlog nekih kriterijuma za podelu sistema za reprezentaciju i prezentaciju informacija.

U drugom poglavlju je dat kratak prikaz elemenata i pojmova koji čine objektno orijentisanu metodologiju i programiranje. Pored toga, naznačene su oblasti primene objektno orijentisanog programiranja kao i veze objektno orijentisane metodologije i baza podataka.

Treće i četvrto poglavlje su ključna poglavlja disertacije i u njima su izloženi originalni rezultati tj. formalizacija jezgra sistema za reprezentaciju i prezentaciju informacija, a kao orudje za tu formalizaciju uvedena je objektno orijentisana struktura podataka i odgovarajući programski jezik za njihovo upravljanje.

U trećem poglavlju je postavljena osnovna koncepcija tj. opisane su strukture pogodne za reprezentaciju i prezentaciju informacija. Predložene su osnovne (ugradjene) klase, tipovi podataka, kao i mehanizmi formiranja novih klasa. Klase se sastoje od vezanih promenljivih koje sadrže konkretne vrednosti ili procedure, tj. metoda ili neke druge klase. Kao osnovni način za uključivanje postojećih klasa u novodefinisane, predlažu se nasledjivanje i umetanje. Klase predvidjaju reprezentaciju informacija u sledećim oblicima: tekst, grafički prikazi, animacije i zvuk.

U četvrtom poglavlju je izložen programski jezik *Less*. Prikazani su osnovni tipovi podataka i struktura jezika, način gradjenja složenih struktura, osnovni elementi svakog *Less* programa i sve direktive za popunjavanje definisane strukture multidigrafa i komande za pisanje procedura. Procedure iniciraju, vode kompletan proces prezentacije i završavaju prezentaciju.

U poslednjem delu ovog poglavlja ukratko se prikazani i predloženi neki mogući načini za reprezentaciju teksta, grafičkih prikaza, animacija i zvuka.

U petom poglavlju su dati konkretni primeri koji ilustruju primenu rezultata iznetih u trećem i četvrtom poglavlju. Definisane su opšte strukture HyperText-a i autorskog sistema, a zatim su napisani odgovarajući *Less* programi za popunjavanje konkretnog primera jednog HyperText i jednog autorskog sistema.

Kao treći primer date su smernice i načini formiranja inteligentnih tutorskih sistema takodje korišćenjem predložene strukture i jezika.

U zaključku je ukazano na značaj predložene strukture i *Less* jezika. Osim toga navedene su neke mogućnosti njihove primene i korišćenja.

U literaturi su navedene knjige, časopisi i radovi koji su korišćeni pri izradi ove disertacije.

Zahvaljujem se dr Djuri Pauniću docentu Prirodno-matematičkog fakulteta, Instituta za matematiku u Novom Sadu na pomoći u izboru teme doktorske disertacije, na korisnim savetima, sugestijama i komentarima pri njenoj izradi. Takodje se zahvaljujem ostalim članovima komisije na ispoljenom razumevanju i sugestijama pri čitanju rukopisa.

Zahvaljujem se suprugu mr Zoranu Budimcu, na korisnim stručnim savetima i opštim smernicama pri izradi disertacije.

Posebno se zahvaljujem dr Dušanu Tošiću docentu Matematičkog fakulteta u Beogradu, suprugu i roditeljima na moralnoj podršci, bodrenju i razumevanju u toku izrade doktorske disertacije.

Neizmernu zahvalnost dugujem i dr Dušanu Surli, redovnom profesoru Prirodno-matematičkog fakulteta, Instituta za matematiku u Novom Sadu na ukazanoj pomoći oko organizacije i praćenja čitavog procesa od prijave teme do odbrane disertacije.

U Novom Sadu
13. marta 1992.

Kandidat
mr Mirjana Ivanović

1 Sistemi za (re)prezentaciju informacija i znanja

Velika ekspanzija i razvoj metoda i tehnika u različitim oblastima računarskih nauka, uslovljava pojavu integrisanih sistema koji pokušavaju da razreše probleme reprezentacije i prezentacije informacija i znanja. Jedan od problema koji se javlja je, kako ogromnu količinu znanja i informacija na najpogodniji način reprezentovati da bi se efikasno i relativno jednostavno koristila. Za ovaj problem je još uvek teško pronaći adekvatno rešenje koje bi pomirilo, često, kontradiktorne zahteve.

U procesu kreiranja sistema možemo izabrati usko specijalizovane i jednonamenske strukture podataka i maksimalno ih prilagoditi odredjenoj oblasti, ili pak, univerzalne i višestruko primenljive. Takodje, možemo pokušati da reprezentovano znanje iskoristimo za razna zaključivanja i eventualno multipliciranje. Možemo se odlučiti da u sistem uključimo interakciju čovek-mašina, na bazi povratne sprege. Na raspolaganju nam stoji i čitav niz drugih mogućnosti.

U poslednje vreme se istraživanja kreću u pravcu razvoja sistema koji bi objedinili sledeće tri komponente:

- **Univerzalna struktura podataka** - pogodna za prikaz svih pojavnih oblika znanja i informacija (tekst, slika, animacija, zvuk, ...).

- **Univarzalni mehanizami** - pogodni za prezentaciju raspoloživih znanja i informacija. Takvi mehanizmi bi trebalo da, na osnovu znanja i informacija sačuvanih u postojećoj strukturi podataka, izdvoje potrebne informacije i odluče na koji način će se one prezentirati.

- **Opšta namena** - mogućnost primene u različitim oblastima gde je potrebno reprezentovati neke informacije koje će se potom koristiti kroz eksplicitnu prezentaciju ili na druge načine.

U poslednje vreme na tržištu softverskih proizvoda mogu se uočiti različite koncepcije i pravci razvoja sistema za reprezentaciju i prezentaciju znanja i informacija, medjutim svi oni, manje ili više, sadrže sledeće elemente [Wahlster,1990]:

1) **Raspoloživo znanje, informacije**. Problemi, oblasti, orudja, pojmovi, ideje i sl., koje treba opisati i reprezentovati, predstavljaju niz znanja i informacija. Da bi se ona koristila potrebno ih je klasifikovati i podeliti u manje semantičke celine.

2) **Dodatne informacije**. Da bi se odredjeni pojmovi i informacije lakše shvatili potrebno je ponekad dodatno ih razjasniti, ili u kontekstu u kom se koriste, ili nekim informacijama opšteg karaktera.

3) **Formalizam reprezentacija**. Osnovne i dodatne informacije je potrebno na pogodan način formalizovati da bi se mogle reprezentovati u računaru, u svrhu daljeg korišćenja.

4) **Mehanizmi prezentacije**. Na osnovu reprezentovanog znanja i informacija mehanizmi prezentacije imaju zadatak da izvrše adekvatnu prezentaciju. Zavisno od domena primene u toku prezentacije mogu se očekivati efekti reakcije korisnika kroz razne vidove povratne sprege.

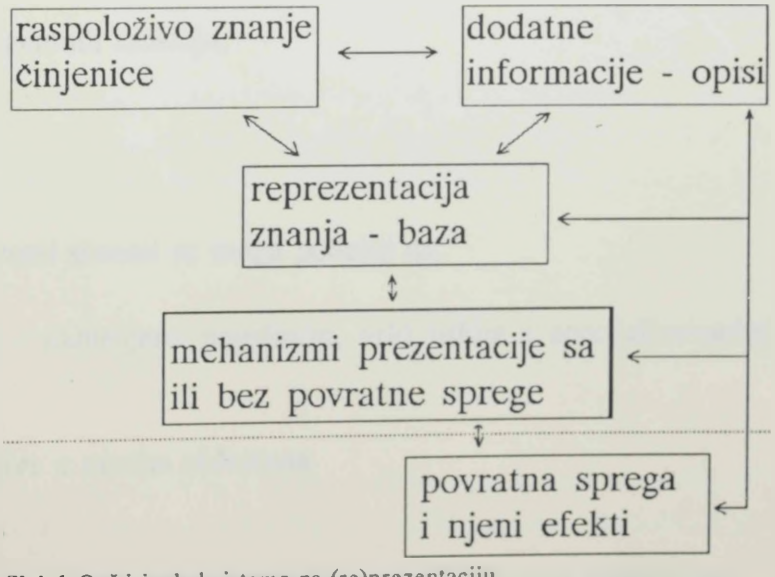
5) **Povratna sprega.** Ukoliko se u procesu prezentacije od korisnika informacija očekuju reakcije, povratna sprega bi trebalo da prihvati informacije od korisnika, pretvori ih u adekvatne akcije i prosledi sistemu na (eventualnu) dalju obradu.

Jedan opšti sistem za (re)prezentaciju informacija i znanja sa navedenim elementima je prikazan na slici 1.1.

Trenutno u svetu, dosta multidisciplinarnih timova projektuje i izradjuje različite sisteme i okruženja pogodne za primenu u sledećim sferama:

- **Procesi učenja i obučavanja** - tj. nastavni procesi na svim obrazovnim nivoima, različiti kursevi,...

- **Prezentacije u užem smislu** - tj. prezentacije softverskih proizvoda, projekata, ideja,...



Sl 1.1 Opšti izgled sistema za (re)prezentaciju

- **Korisnički interfejsi** - sofisticirana i raznim efektnim orudjima i sredstvima realizovana veza koristik - program (računar).

Kompleksnost znanja i informacija koje treba (re)pezentovati pomoću računara, zahteva i različite načine njihovog medijskog predstavljanja: tekst, slika, simulacija, zvuk... Uključivanje različitih aspekata (re)prezentovanja znanja i informacija, karakteriše navedene sisteme kao **multimedijske sisteme.**

Takodje razvoj metoda veštačke inteligencije uslovljava njihovu primenu i kombinovanje sa klasičnim metodama u razvijanju i korišćenju raznih programskih proizvoda. Ovo spajanje metodologija daje nove i kvalitetnije softverske proizvode tipa: sistemi za reprezentaciju i prezentaciju informacija (znanja), ekspertni sistemi, inteligentni sistemi za obuku, inteligentni interfejsi i sl.

Upotreba metoda veštačke inteligencije u raspoloživim sistemima za (re)prezentaciju znanja i informacija, za sada ne daje zadovoljavajuće rezultate. Medjutim, i dalje se istraživanja usmeravaju kako na razvijanje i poboljšavanje postojećih metoda veštačke inteligencije, tako i na njihovu primenu i kombinovanje sa klasičnim metodama u većini sistema za reprezentaciju i prezentaciju informacija i znanja.

1.1 Podela sistema za (re)prezentaciju

Sistemi za (re)prezentaciju znanja i informacija mogu se deliti prema kriterijumima:

- oblasti primene,
- posedovanja inteligentnih funkcija,
- interaktivnosti.

1) **Prema kriterijumu primenljivosti** sistemi se mogu podeliti na:

a) specijalizovane sisteme - namenjene pojedinim, vrlo uskim i specijalizovanim oblastima,

b) opšte sisteme - primenljive u raznim oblastima.

2) **Prema kriterijumu posedovanja inteligentnih funkcija** sistemi se mogu podeliti na:

a) inteligentne sisteme - koji u sebe uključuju različite metode veštačke inteligencije,

b) neinteligentne sisteme - koji ne koriste metode veštačke inteligencije, te stoga ne poseduju nikakve inteligentne funkcije.

3) **Prema kriterijumu interaktivnosti** sistemi se mogu podeliti na:

a) sisteme sa povratnom spregom - koji od korisnika očekuju akcije koje će uticati na ponašanje sistema,

b) sisteme bez povratne sprege - kada je korisnik pasivni element procesa prezentacije i od njega se ne očekuju nikakve akcije.

U opštem slučaju, što je sistem uže specijalizovan to postoji veća i realnija mogućnost efikasnog uključivanja u njega metoda veštačke inteligencije i obrnuto, što je sistem opštiji to je teže učiniti ga inteligentnim.

Medjutim, uprkos svim postojećim problemima, krajnji cilj većine istraživanja su inteligentni sistemi opšte namene.

Trenutno se razlikuje nekoliko osnovnih grupa sistema za (re)prezentaciju informacija i znanja koji još uvek imaju niz zajedničkih svojstava i dodirnih tačaka. Medjutim, niz svojstava jasno razlikuje i odvaja ove grupe, pa se može u budućim istraživanjima očekivati i njihovo oštrije razgraničavanje. U osnovi se razlikuju sledeće četiri grupe sistema:

- HyperText sistemi,
- autorski sistemi,
- inteligentni tutorski sistemi,
- interfejsi.

HELP sistemi i sistemi za automatsku prezentaciju su takodje sistemi za reprezentaciju i prezentaciju informacija. Pošto ovi sistemi još uvek nemaju ključne karakteristike koje ih jasno izdvajaju od ostalih sistema, mogu se javiti i kao posebne grupe, a mogu se podvesti i pod HyperText sisteme.

1.2 Multimedijalnost

Multimedijalnost je termin koji se nedavno pojavio u oblasti računarskih nauka, ali je brzo zauzeo jednu od najznačajnijih pozicija u sadašnjem trenutku, sa tendencijom da još dugo ostane na tom mestu.

Multimedijalnost je istovremeno nova naučna discplina, nova tehnologija i nova filozofija.

Bez obzira da li se upotrebljavaju pojmovi: multimedijalni sistemi, interaktivni multimedijalni sistemi, hipermedijalni sistemi i sl., misli se na sisteme koji integrišu tekstualne informacije memorisan u računaru, sa grafikom, animacijom, stereo zvukom, pa čak i videom, sa namenom prezentacije znanja i informacija [Veljkov, 1990].

Kada se govori o multimedijalnoj osobenosti softverskih proizvoda, onda se misli na softverske proizvode koji integrišu

- kreiranje jednostavne animacije,
- transformacije grafičkih prikaza na ekranu,
- snimanje i reprodukcija digitalizovanog govora i muzike,
- memorisanje čitave sekcije rada, da bi se kasnije iskoristila u različite svrhe.

U takvim sistemima, sa stanovišta učešća periferija, postoji:

1. Višeperiferijska - rasuta prezentacija. U proces prezentacije je uključeno više periferija: monitor, video, CD plejer, ... koji podržavaju proces prezentiranja informacija.

2. Jedoekranska prezentacija. U proces prezentacije je uključen samo monitor računara i na njemu mogu istovremeno da figurišu različiti oblici prezentiranja informacija: tekst, grafika, animacija, zvuk...

Sa stanovišta koordiniranosti informacija razlikuju se dva načina prezentacije.

1. Koordiniranost medija. Celovita informacija se dobija kao unija neredundantnih pojedinačnih pojavnih oblika informacija.

2. Nekoordiniranost medija. Pojedini delovi informacija mogu biti multiplicirani, tj. izraženi više puta kroz različite pojavne oblike.

Razvijanje multimedijalnih aplikacija je još uvek mlada oblast, čiji se burni razvoj i standardizacija postupaka očekuju u budućnosti.

1.3 HyperText

Da bi se na jednostavan, intuitivan način shvatio HyperText sistem opišimo sledeću situaciju [Fiderio, 1988].

Čovek zainteresovan da što više sazna o Mocartu dolazi u biblioteku i nalazi knjigu o Mocartu. Iz nje saznaje da je Mocart bio čuveni Austrijski kompozitor 18. veka.

On počinje da se pita šta se dešavalo u Austriji u to vreme? U katalogu istorijskih knjiga pronalazi knjigu o istoriji Austrije, u kojoj uočava grad Salcburg i poželi da vidi sliku grada tog vremena. U drugom katalogu pronalazi knjigu sa slikama iz tog doba. Potom se ponovo vraća knjizi o Mocartu i čita o njegovoj kompoziciji koju nikada do tada nije čuo. U delu sa snimljenim kompozicijama, pronalazi željenu i sluša je.

Ovaj proces završava kada je zadovoljan znanjem koje je stekao ili, kada se umori, ili zbog prestanka rada biblioteke, ili ....

Zamislimo da čovek kod kuće sedi za računarom, startuje (HyperText) program, o muzici i sve gore navedeno pronalazi na pogodan način povezano u tom okruženju.

U užem smislu HyperText se može definisati kao struktura podataka za reprezentaciju informacija koja uključuje i skup raznih orudja za njihovo rukovanje i prezentaciju.

U širem smislu HyperText je softversko orudje za prikupljanje, uskladištavanje, pretraživanje i prezentaciju informacija sa referencama (asocijativnim vezama).

HyperText oponaša sposobnost ljudskog mozga da uskladišti i pretražuje informacije sa referencama, na brz i intuitivan način. Na osnovnom nivou to je sistem za upravljanje bazom podataka (DBMS), koji pruža mogućnost vezivanja informacija na ekranu i u bazi korišćenjem asocijativnih veza.

HyperText sistemi su primenljivi u različitom oblastima učenja, prezentacija informacija, prezentacija u užem smislu, korisničkog interfejsa i slično.

Prema podelama sistema za (re)prezentaciju informacija iz poglavlja 1.1., HyperText sistemi se mogu svrstati u grupu neinteligentnih sistema, opšte namene bez povratne sprege.

Struktura znanja i uskladištenih informacija, kao i kvalitet sistema, zavise od autora HyperText sistema. Zadatak je autora da podeli raspoložive informacije na semantički zaokružene celine i izdvoji pojedinačne značajne i ključne pojmove, uspostavi veze izmedju njih i obezbedi dodatno znanje za pretraživanje.

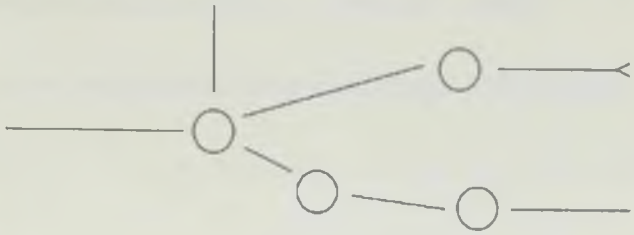
1.3.1 Svojstva HyperText-a

Suštinu svakog HyperText-a čine sledeći elementi [Frisse, 1988]:

a) **Baza podataka** sa čvorovima u kojima će se zapisivati informacije i metod za zapisivanje i pristup informacijama u njoj.

b) **Šema veze elemenata (čvorova) baze podataka**, koji čine semantičku mrežu. Iz svakog čvora izvire bar jedna veza prema ostalim čvorovima baze (Slika 1.2).

c) **Interfejs** koji omogućava vizuelnu prezentaciju elemenata baze podataka, i jednostavan način kretanja iz jednog elementa u elemente koji su sa njim povezani.

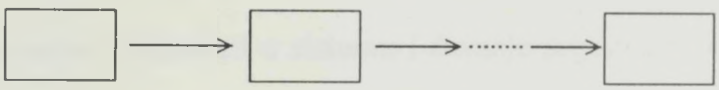


Sl 1.2 Izgled jednog dela semantičke mreže HyperTexta-a

Baza podataka HYPETEXT-a se (obično sastoji) od čvorova, čiji je radni prostor za memorisanje informacija veličine jednog ekrana. Oni se mogu puniti tekstualnim, grafičkim, audio i video informacijama.

Postoji nekoliko načina vezivanja elemenata - čvorova baze podataka HyperText-a, ali su najinteresantniji i ujedno najprimenljiviji sledeći.

1. **Linearna veza čvorova baze podataka** (Slika 1.3),



Sl 1.3 Linearna veza čvorova HyperText-a

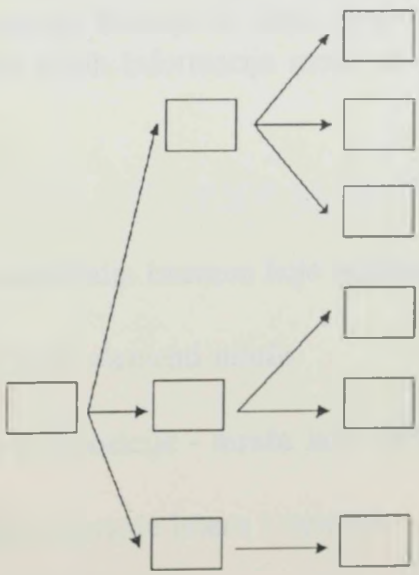
2. **Hijerarhijska veza čvorova baze podataka** (Slika 1.4).

Funkcije HyperText-a su u značajanoj meri podredjene multimedijalnoj prezentaciji. One dozvoljavaju dinamičko vezivanje elemenata multimedijalne aplikacije (tekst, grafika, animacija, video,...) za dodatne informacije.

Tipičan HyperText uključuje u sebe:

- tekst editor,
- grafički editor,

Osim ovih HyperText, u opštem sličaju, ima i čitav niz drugih orudja.



Sl 1.4 Hijerarhijska veza čvorova HyperText-a

1. Neka od orudja HyperText-a su: miševi, prozori, ikone i 'padajući' meniji.

2. HyperText može da pruži niz indeksnih mogućnosti koje olakšavaju i ubrazavaju rad kao što su:

- invertovane datoteke reči,
- hijerarhijski indeksi,
- kontekstno zasnovani indeksi.

3. Postoji mogućnost povezivanja HyperText sistema (aplikacije) sa nekim drugim eksternim programima.

U poslednje vreme je primetna tendencija mešanja tehnika HyperText-a i veštačke inteligencije u realizaciji inteligentnih sistema [Carando, 1989]. Pored navedenih elemenata koji čine suštinu HyperText-a ((a), b), c)), takvi sistemi poseduju i neke nove elemente:

d) **Šemu prikupljanja znanja** koja omogućava automatsko dodavanje informacija i veza medju njima,

e) **'Posmatranje' aktivnosti u sistemu** i davanje sugestija o promeni informacija u nekim čvorovima.

1.3.2 Korišenje HyperText-a

Kada se HyperText koristi da bi se kreirao neki (najčešće) multimedijalni proizvod potrebno je informacije (koje treba memorisati), prvo podeliti u jednostavne semantičke celine - čvorove. Čvorovi sadrže jednostavan koncept ili ideju, koja se može prikazati na jednom ekranu [Frisse, 1988]. Za slučaj dužih informacija može se uključiti mehanizam skroliranja.

Čvorovi mogu biti:

- **tipizirani (imenovani)** - sa asocijativnim imenom koje odslikava sadržaj čvora,

- **netipizirani (bezimeni)** - 'anonimni' elementi mreže.

Pojedinačni čvorovi se vezuju u kompozicije - mreže koje treba da omoguće brzo prebacivanje iz jednog čvora u drugi.
Neki HyperText-ovi omogućavaju zadavanje imena i vezama.

Kada se koristi HyperText sistem korisnik može, ukoliko je baza podataka (čvorova) velika, da zaboravi kako i zašto je došao u tačku u kojoj je trenutno. U takvim slučajevima,

mnogi sistemi omogućavaju grafički prikaz - strukturni dijagram mreže čvorova sa oznakom trenutnog čvora. Taj dijagram bi, osim prikaza, trebalo da omogući prebacivanje u bilo koju drugu tačku - čvor mreže.

Neki HyperText sistemi dozvoljavaju zadavanje podrazumevajućeg (eng. *default*) puta kroz bazu, koja vodi ili usmerava korisnika kroz uredjenu listu čvorova.

U zavisnosti od oblasti primene i korisnika, HyperText sistemi variraju značajno, medjutim, u suštini se razlikuju četiri tipa.

1. sistemi za rešavanje problema (uvežbavanje) i obučavanje,

2. sistemi za on-line pretraživanje (on-line priručnici i dokumentacija),

3. bibliotečki sistemi (nerealni zbog ogromne količine informacija) i javni informacioni sistemi,

4. višenamenski (opšti) sistemi.

HyperText sistemi nisu veštačko-inteligentni u svojoj osnovi. Oni samo pomažu da se razjasne i ubrzaju misli, ali čovek je u osnovi kreator i kontrolor čitavog procesa.

Kreator HyperText aplikacije odlučuje koje će se informacije uključiti u bazu podataka, koje veze će se kreirati, kako će se organizovati čvorovi, i na koji način će se koristiti.

1.3.3 Nedostaci HyperText-a

HyperText je mlada tehnologija koja još nema izgradjene standarde i pati od niza nerazrešenih problema:

1) Kreiranje pogodnog modela kojim se može relativno lako upravljati?

2) Programi za realizaciju HyperText sistema (koji mogu da budu veoma dugi) se teško prepravljaju u prilagodjavaju nekim novim zahtevima.

3) Nepostojanje tačno utvrdjenog puta kojim se prolazi kroz čvorove HyperText sistema i mogućnost da korisnik sam bira čvorove u koje će 'otići', može da dovede do zbrke i gubljenja u mreži čvorova.

4) Teškoće da se raspoloživi skup informacija podeli u čvorove, a takodje teškoće da se prvobitna podela koriguje.

5) Nedostatak zadovoljavajuće primene metoda veštačke inteligencije u procesu popunjavanja HyperText sistema i odredjivanja veza medju čvorovima. Osnovni sadržaj HyperText čvorova čini velika količina informacija i podataka koje bi trebalo vezati asocijativnim vezama. Tehnike obrade i komprimovanja teksta sa semantičkog stanovišta, još nisu razvijene tako da bi se mogle primeniti na automatsko formiranje HyperText-a. Automatsko formiranje bi zahtevalo da se, na primer, na osnovu skupa ključnih reči napravi raspodela informacija po čvorovima i uspostave logične veze medju njima.

1.4 Autorski sistemi - AS

Autorski sistemi (eng. *Authoring Systems*) su softverska orudja koja uspevaju da integrišu računar i periferije, kao što su CD-ROM-ovi (laserski diskovi sa ugradjenim tekstom) i laserski videodiskovi [Veljkov, 1990].

AS poseduju povratnu spregu pa su poznati i pod nazivom CAI (eng. *Computer Aided Instruction*) sistemi.

Predstavljaju značajnu grupu sistema koji su namenjeni oblasti učenja i prezentacija u užem smislu.

Prema podelama iz poglavlja 1.1, mogu se svrstati u grupu neinteligentnih sistema, sa povratnom spregom.

Široki spektar ljudi: nastavnici, menadžeri, prodavci, kreatori raznih nastavnih procesa, koristi AS da kreira svoje sopstvene interaktivne multimedijalne programe (sisteme).

Autorski sistemi prve generacije su bili nezavisno razvijeni softverski proizvodi bez sposobnosti kontrolisanja eksternih uredjaja. Medjutim, tekući trendovi su razvijanje AS koji će moći da kontrolišu i sinhronizuju veliki broj eksternih uredjaja te stoga postaju sve moćniji i sve se više primenjuju u oblasti interaktivnih multimedijalnih sistema.

AS se dele prema nameni i može se reći da su u većini slučajeva specijalizovani za pojedine oblasti.

Znanje i informacije koje treba da budu reprezentovane u sistemu su podeljene u manje celine - teme (eng. *topics*) koji su medjusobno povezani tako da čine semantičku celinu. Teme i veze medju njima formiraju graf. U procesu korišćenja sistema (prezentacije reprezentovanih informacija) vrši se kretanje po različitim putevima u grafu sistema. Kretanje po temama najčešće nije slobodno već je odredjeno odgovorima koje korisnik sistema daje na upite koji mu se postavljaju u procesu prezentacije.

1.4.1 Svojstva koja treba da poseduje AS

Autorski sistemi su kompleksni sistemi i treba da poseduju niz svojstava koja će ih učiniti boljim, kvalitetnijim i efikasnijim. Jasno je da svi autorski sistemi ne poseduju sva navedena svojstva.

1. AS daje korisniku mogućnost da dizajnira i kreira sopstvenu aplikaciju. Za kreiranje multimedijalne aplikacije AS ne zahteva znanje i korišenje nekog programskog jezika. Medjutim, postojanje nekog jezika kao opcije može biti vrlo korisno. To znači da AS treba da ima svoj sopstveni programski jezik ili da nudi direktan pristup nekom takvom jeziku.

Do sada je obično programski jezik autorskih sistema bio proceduralnog tipa. Medjutim, pojava raznih paradigmi programskih jezika (opisanih i prikazanih u poglavlju 2.) kao što su objektna, funkcionalna, logička, može da igra značajnu ulogu u dizajniranju i implementaciji autorskih jezika nove generacije.

2. AS treba da omogući jednostavnu integraciju izlaza iz nekih drugih aplikacija, unutar same reprezentacije i prezentacije. Treba da omogući korišćenje nekih već postojećih tekstualnih datoteka, kao i uključivanje grafike, kreirane nekim nezavisnim, grafičkim aplikacijama (van AS).

3. AS mora da omogući kreiranja datoteka. Tekst editor može biti ugradjen u AS ili može da podržava neke postojeće. Ako AS ima sopstveni tekst editor, onda on treba da poseduje većinu svojstava osnovnih tekst editora.

4. AS može da poseduje svoj sopstveni grafički editor. Medjutim, daleko su moćniji samostalni grafički paketi koji nude daleko kompleksnija rešenja i veći grafički kvalitet od grafičkih editora ugradjenih u AS.

5. AS treba da omogući kontrolisanje toka prezentacije i kretanje iz jednog dela sistema u drugi kao i jednostavnu 'šetnju' kroz multimedijalne aspekte.

6. AS treba da poseduje jedinstven interfejs, jednostavan za učenje i korišćenje, koji će omogućiti unificirano korišenje svih orudja raspoloživih na računaru. Svaki put kada se kreira novi multimedijalni element (doda novi uredjaj), postojeći interfejs treba da ga podrži, ili se to može postići njegovom jednostavnom dogradnjom i proširenjem.

7. AS treba da omogući potpunu integraciju teksta, grafike, animacije, zvuka i videa i pristup tim elementima treba da je ugradjen u standardni intefejs.

8. AS treba da poseduje orudja za obradu digitalizovanog zvuka, kao i mogućnost animacije i videa, direktno ili preko lako dostupnih eksternih programa. Prikazivanje video pokreta i kompjuterskih informacija na istom ekranu naziva se jednoekranski interaktivni video. (Postoje oprečna mišljenja da li interaktivni multimedijalni proces treba da koristi jedan ili dva ekrana. U svakom slučaju bilo bi korisno ako bi AS podržavao obe mogućnosti.)

9. Sistemom treba da budu obuhvaćene dve osnovne funkcije:

- grananje - koje dopušta odlazak u naznačenu temu iz koje se može vršiti povratak u prethodno stanje ili kretanje kroz neki drugi put u grafu sistema.

- startovanje nekih drugih aplikacija iz AS i automatski povratak iz startovane aplikacije.

10. AS treba da nudi kontekstno-osteljiv help sistem.

1.4.2 Primena AS u procesima učenja

AS se može koristiti u obrazovnim procesima tipa kreiranje lekcija, kreiranje kurseva i slično. U tom slučaju u AS moraju da postoje specijalni elementi za proveru stečenih znanja.

Dakle, AS bi mogao da poseduje neke mehanizme testiranja učesnika obrazovnog procesa, vodjenje statistike prolaznosti, formiranje ocena i/ili sličnih elemenata na osnovu kojih bi se mogli izvoditi zaključci o težini obrazovnog procesa i kvalitetu samog autorskog sistema.

Jedno od pogodnih rešenja bi bilo kreiranje posebnih slogova za svakog učesnika, u kojim bi se čuvale informacije tipa:

1) broj tačnih odgovora,

2) dužina odziva na postavljeni zadatak i izdavanje odgovora,

3) broj prolazaka kroz pojedine teme.

Takodje se može, na osnovu procenta tačnih odgovora u jednoj sekciji, dopustiti ili zabraniti prelazak na neki viši nivo, ili na sledeću temu (zabrana grananja i kretanja nekim putevima).

Primena AS u obrazovnim procesima trebala bi da omogući rukovanje različitim tipovima testova uz korišćenje pitanja-odgovora:

1) istina-laž,
2) višestruki izbor,
3) otvoreni odgovor i sl..

U procesu testiranja, pri postojanju pitanja tipa otvoreni odgovor, AS bi trebalo da podrži višerečne odgovore uključujući rukovanje pravopisnim greškama. Ovaj aspekt otvara mogućnost primene metoda veštačke inteligencije iz oblasti prepoznavanja i razumevanja prirodnog jezika i govora.

AS podržavaju značajan deo orudja potrebnih za kreiranje sofisticiranih i moćnih interaktivnih multimedijalnih aplikacija. Medjutim, on je onoliko dobar koliko je kreativan i maštovit njegov autor.

1.5 Inteligentni sistemi za obuku - ITS

ITS (eng. *Intelligent Tutoring Systems*) su sistemi namenjeni isključivo procesima učenja. Oni predstavljaju presek raznih oblasti: obrazovanja, psihologije, veštačke inteligencije, kognitivne nauke i dr.. To su najčešće usko specijalizovani sistemi koji zbog takve orijentacije, uglavnom, uspešno koriste metode veštačke inteligencije.

ITS su računarski programi koji koriste tehnike veštačke inteligencije, a pomažu ljudima u procesu učenja. Kako su u osnovi namenjeni procesu učenja to su takodje sistemi sa povratnom spregom.

Arhitektura i struktura varira od sistema do sistema, ali se uglavnom razlikuju četiri elementa svakog ITS [Wood, 1990].

1) **Ekspertni (nastavnički) modul**. To je baza podataka koja čuva informacije iz oblasti u kojoj se ITS koristi.

2) **Učenički modul**. Ovaj modul služi za modeliranje individualnih znanja pojedinih učenika o oblasti u kojoj se ITS koristi. Sadržaj modula se konstantno menja kako učenik napreduje u učenju i stiče nova znanja.

3) **Obučavajući (tutorski) modul**. Definiše kako materijal o oblasti koja treba da se prezentira, tako i način i vremensku dinamiku prezentacije.

4) **Dijagnostički modul**. Kontinuirano menja učenički modul, u skladu sa odgovorima koje učenik daje na postavljene upite u procesu učenja. Ovaj modul je takodje vezan sa ekspertnim modulom.

Cilj u ovim sistemima je da se komunikacija izmedju računara i učenika odvija na prirodnom jeziku. Medjutim, obzirom na stanje u oblasti razvoja interfejsa na prirodnom jeziku, za sada se komunikacija odvija ili na nekom meta-jeziku bliskom prirodnom jeziku, ili na nekom uskom podskupu prirodnog jezika.

Komunikacija se ostvaruje tako što se 'prirodni' jezik na ulasku u sistem prevodi u računarski kod, a na izlazu iz sistema računarski kod se prevodi u 'prirodni' jezik. Komunikacija je uglavnom tekstualne prirode ili poseduje rudimentirane grafičke mogućnosti.

1.6 Inteligentni multimedijski interfejsi

Sve češće se javlja potreba da se komunukacija čovek-računar standardizuje i približi ljudskim potrebama. Korišćenje metoda veštačke inteligencije dovodi do razvijanja inteligentnih korisničkih interfejsa.

Ovakav interfejs pre svega treba da prihvata formalizovano reprezentovane informacije od aplikativnog programa i da ih prezentira grafikom, animacijom, zvukom ili prirodnim jezikom (po mogućnosti koordiniranim medijima).

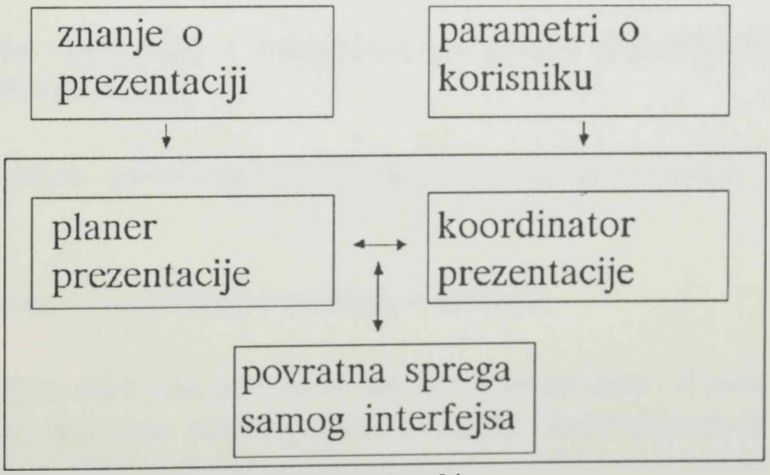
Takodje, treba da omogući prihvatanje informacija od korisnika, prirodnim jezikom, pokazivanjem, izborom iz menija i slično, i da ih pretvara u formalizam i kod razumljiv aplikaciji.

Da bi, inteligentni interfejs, ostvario ovakvu komunikaciju potrebno je da prethodno poseduje dodatne informacije o:

- aplikaciji,
- korisniku i njegovom predznanju vezanom za aplikaciju,
- cilju koji treba da se postigne prezentacijom.

Zahtevi, koje treba da zadovolji takav interfejs, nisu ni malo jednostavni i lako razrešivi. U prilog tome govori činjenica da trenutno postoji samo par projekata [Wahlster, 1990], koji se približavalju postavljenim ciljevima, ali ih još uvek nisu dostigli. Projekti su usko specijalizovani, i primenljivi samo na jednu klasu problema.

Formalno se može odrediti opšta struktura inteligentnog interfejsa (Slika 1.5).



Sl 1.5 Opšta struktura inteligentnog interfejsa

1) Planer prezentacije. Uzima formalizovano znanje i iz njega generiše (po mogućnosti koordinirano) tekst, sliku, animaciju, zvuk. Tako formiranu prezentaciju šalje dalje na obradu koordinatoru prezentacije.

2) Koordinator prezentacije. Dobijene elemente prezentacije pokušava da smisleno ukomponuje na medijima koji mu stoje na raspolaganju. Ukoliko ne uspe u tom pokušaju, on vraća prosledjene elemente sa zahtevom korekcije. Kada se planer i kordinator prezentacije usklade, ukomponovana prezentacija se šalje dalje na obradu povratnoj sprezi.

3) Povratna sprega prezentacije. Simulira korisnika - čoveka. Njen zadatak je da ustanovi da li je postavljeni cilj postignut datom prezentacijom. Ukoliko nije zadovoljna razumljivošću i načinom prezentacije povrtana sprega vraća sve elemente koordinatoru prezentacije i zahteva korekcije. Onog trenutka kada je povratna sprega, zadovoljna formiranom prezentacijom prosledjuje je krajnjem korisniku - čoveku.

Kako bi čitav proces trebalo da se odvija u realnom vremenu, jasno je da komunikacija izmedju pojedinih elemenata interfejsa treba da je svedena na razumnu vremensku meru.

1.7 Sistemi za automatsku prezentaciju

Jednu specijalnu grupu sistema za upravljanje informacijama različitog tipa čine takozvani sistemi za automatsku prezentaciju. Ovi sistemi ne donose ništa novo u odnosu na pomenute sisteme, već, poseduju znatno suženi skup mogućnosti.

U suštini sistemi za automatsku prezentaciju sadrže sledeće elemente:

- **Grafički editor** - koji omogućava kreiranje različitih šema, blok dijagrama i sličnih grafičkih prikaza.

- **Tekst editor** - sa specijalnim mogućnostima ispisa teksta kao što su: različiti skup fontova, različita veličina slova, specijalni efekti (istaknut ispis, trepćuća slova, podvučeni ispis, italik ispis i sl.).

- **Generator jednostavnijih animacija** - omogućava generisanje jednostavnih animacija na osnovu kreiranih grafičkih prikaza.

- **Skup direktiva** - omogućava povezivanje pojedinačnih elemenata u celinu i odredjivanje dinamike prezentacije.

Korišćenje sistema za automatsku prezentaciju se odvija u dve faze.

Prva faza predstavlja pripremu elemenata koji će se koristiti u prezentaciji. U ovoj fazi se uz pomoć editora i generatora formiraju tekstovi, slike i animacije. Svaki element je takav da može da stane na jedan ekran. Kada su elementi formirani vrši se njihovo vezivanje u determinisanu sekvencu (elementi su vezani linearno onim redom kojim će se prezentirati) i odredjuje dinamika prezentacije. I ovde postoje dve mogućnosti.

Svaki element može na ekranu da bude fiksiran u dužini od nekoliko vremenskih jedinica, nakon čega se automatski prikazuje naredni element, i tako redom do kraja prezentacije.

U drugom slučaju se element može zadržati na ekranu sve dok korisnik sistema ne preuzme neku akciju (pritisne dirku na tastaturi, mišu, ili slično).

Druga faza predstavlja korišćenje pripremljenih elemenata. Prezentacija je tada strogo determinisani prikaz niza elemenata (tekstova, slika, animacija) bez ikakvog uticaja korisnika na redosled prezentiranja pojedinih elemenata. Elementi se prezentiraju uvek u istom redosledu, jedan za drugim. Nikakva komunikacija sa sistemom nije moguća.

Sistemi za automatsku prezentaciju su namenjeni, uglavnom, za prezentiranje nekih proizvoda i ideja u reklamne ili informativne svrhe.

2 Objektno orijentisana metodologija

Viši programski jezici, od njihove pojave, neprestano doživljavaju evoluciju. Od asemblerskih jezika u 50-tim godinama, proceduralnih u 60-tim, strukturnog programiranja i apstrakcije podataka u 70-tim, programski jezici su evoluirali do objektno orijentisanih, distribuiranih, funkcionalnih i relacionih u 80-tim godinama. Ove paradigme se i danas dalje razvijaju i obogaćuju tako da će još dugo zauzimati značajno mesto u računarskim naukama.

Evolucija programskih jezika

Programski jezici visokog nivoa su evoluirali na sledeći način [Wegner, 1990]:

1) **Period 1954 - 1958** (nazivaju se jezicima prve generacije) - karakterišu jezici FORTRAN I, Algol 58, Flowmatic, ... koji uvode osnovne koncepte programskih jezika i njihove implementacije.

2) **Period 1959 - 1961** (nazivaju se jezicima druge generacije) - karakterišu jezici FORTRAN II, Algol 60, COBOL, LISP. Oni konkretizuju i doradjuju koncepte uvede u programskim jezicima prve generacije.

3) **Period 1962 - 1969** (nazivaju se jezicima treće generacije) - karakterišu jezici PL/I, Algol 68, Pascal, Simula.

4) **Period 1970 - 1979** (jezici ovog perioda predstavljaju generacijski jaz) - karakterišu jezici CLU, Ada, Smalltalk koji su vodili od izražajne moći jezika ka strukturama i softverskom inženjerstvu.

5) **Period 1980 - do danas** - uvodi paradigme programskih jezika kao što su objektno orijentisana, distribuirana, funkcionalna, relaciona i druge. One dalje dogradjuju i razradjuju postojeće koncepte programskih jezika, ali uvode i karakteristične strukture, karakterističnu implementaciju i način mišljenja. Oni oblikuju jezike četvrte generacije.

Veliki broj osnovnih koncepata postavljaju programski jezici prve generacije uključujući aritmetičke izraze, naredbe, nizove, liste, stekove i podprograme.

Ovi koncepti su čvršće ustanovljeni i razradjeni u jezicima druge generacije. Na primer, jezik Algol 60 nije doživeo širu upotrebu, ali je imao izuzetno značajan uticaj na jezike narednog perioda kao što su Pascal, Ada, Simula.

Pokušaj da jezici iz treće generacije zauzmu mesto jezika iz druge generacije je bio neuspešan (osim jezika Pascal). T. Hoare je istakao da je Algol 60 bolji jezik od većine njegovih direktnih potomaka.

Period od 70-tih godina pa nadalje je period intenzivnih israživanja i preispitivanja ciljeva kreiranja programskih jezika. Usledile su promene u dizajniranju programskih jezika od izražejne moći jezika ka struktuiranosti programa. Na mikro nivou struktuirana **while** naredba zamenjuje nestruktuirane **goto** naredbe. Na makro nivou insistira se na modularnosti, prvo na funkcijama i procedurama a kasnije na objektima i apstrakciji podataka. Ideju o skrivanju i apstrakciji podataka uveo je Hoare, a nešto kasnije i mehanizam konkurentnog programiranja.

Ada je prvi programski jezik koji donosi bogat skup navedenih mogućnosti, medjutim zbog velike opštosti i glomaznosti za sada ne doživljava šire primene. Sličan ovom jeziku je jezik Modula-2 koji je razvio Wirth 80-tih godina. Pored toga što podržava mogućnosti jezika niskog nivoa i jezika visokog nivoa on omogućava značajan nivo modularnosti i apstrakcije podataka [Helman, 1988; Sinkovec, 1986].

U periodu od 80-tih godima do danas vrše se i intenzivna istraživanja u oblasti funkcionalnih, logičkih programskih jezika, jezika baza podataka, a pojam objektna orijentisanost postaje sve značajniji. Istraživanja u oblasti programskih jezika se danas kreću od izučavanja pojedinih jezika ka izučavanju paradigmi vezanih za pojedine klase jezika.

Apstrakcija podataka

Razvoj struktura podataka uslovljen je problemima koji se rešavaju računarom i razvojem i projektovanjem novih programskih jezika.

Počev od pojave prvih programskih jezika pa sve do 80-tih godina, strukture podataka i tipovi su se razvijali od jednostavnijih kao što su nizovi, skupovi preko pokazivača i slogova, do raznih kompleksnih struktura podataka realizovanih mehanizmom apstrakcije.

Apstraktni tipovi podataka su parovi (V,O) gde je V skup vrednosti, a O skup operacija definisanih nad tim vrednostima.

Osim apstrakcije podataka, mnogi programski jezici podržavaju i druge apstrakcije kao što su apstrakcija funkcija, procesa, problema itd.

U nekim programskim jezicima se uvode moduli koji predstavljaju sintaksne jedinice koje pružaju mogućnost da se realizuju apstrakcije podataka, funkcija, procesa što predstavlja značajnu mogućnost u modernom softverskom inženjerstvu [Hewitt,1989].

Moduli omogućavaju da se softverski sistemi podele u fizičke i logičke jedinice koje imaju jasno definisane kanale za komunikaciju - interfejs, i omogućavaju skrivanje načina realizacije i funkcionisanja. To je značajno svojstvo odvajanja koncepcije od realizacije.

Apstrakcija podataka omogućava grupisanje logički vezanih softverskih komponenti zajedno.

Druga značajna mogućnost u programskim jezicima je realizacija **opštih** apstraktnih tipova podataka. Oni omogućavaju da elementi nekog apstraktnog tipa podataka mogu biti različitih tipova.

Korišćenje mehanizama apstrakcije podataka i opštih tipova podataka omogućava uvodjenje objektne orijentisanosti u konvencionalne programske jezike.

Softverski sistem je skup mehanizama kojima se izvršavaju neke akcije nad nekim podacima. Ako se zanemari arhitektura sistema ostaje fundamentalno pitanje da li funkcionisanje sistema treba da se zasniva na akcijama ili podacima? U odgovoru na ovo pitanje se krije razlika izmedju tradicionalnih metoda i objektno orijentisanog pristupa i projektovanja.

Objektno orijentisano projektovanje je koncept koji proizvodi arhitekturu zasnovanu na podacima - objektima sa kojima svaki sistem ili podsistem manipuliše. Projektanti mogu dovoljno dugo da ostave po strani definisanje osnovnih funkcija sistema, a da se skoncentrišu na analizu klasa objekata koji se javljaju u sistemu. Projektovanje kompletnog sistema će stoga biti rezultat njihovog konstantnog poboljšanja u razumevanju objekata tj. struktura podataka sistema. Ova koncepcija, obrnuta od tradicionalne, omogućava ključna svojstva pojedinih modula sistema: ponovno korišćenje (eng. *reusability*) i proširivanje (eng. *extendibility*).

Definicija 2.1. Objektno orijentisano projektovanje je formiranje softverskih sistema u obliku struktuiranih modula. Svaki pojedinačan modul predstavlja implementaciju nekog apstraktnog tipa podataka.

Ako se u modulu nalazi neka apstrakcija podataka, njegov interfejs služi za prikaz funkcija nad podacima. Moduli objektno orijentisanih sistema se zovu **klase**. Znači klase su implementacija apstraktnog tipa podataka a ne sami apstraktni tipovi podataka.

Objektno orijentisano programiranje

Objektno orijentisano programiranje uobličava i dogradjuje postojeće elemente programiranja, a takodje uvodi i nove koncepte koji ga odvajaju od klasičnih programskih jezika.

Nekoliko osnovnih pravila i zahteva odredjuje objektnu orijentisanost [Meyer, 1988]:

1) **Objektno zasnovana modularna struktura**. Sistem se deli na module na osnovu njegovih struktura podataka.

2) **Apstrakcija podataka**. Objekti se opisuju kao implementacija apstraktnih tipova podataka.

3) **Automatsko upravljanje memorijom.** Nekorisne objekte treba da ukloni sistem bez intervenisanja programera.

4) **Klase.** Svaki složeni tip podataka je moduo i svaki moduo visokog nivoa je tip podataka. Klasa opredeljuje skup sličnih objekata i dobija potpuni i pravi smisao kada se vezuje i komponuje sa drugim klasama kroz mehanizme nasledjivanja (eng. *inheritence*) i umetanja (eng. *embedding*).

5) **Nasledjivanje.** Klase se mogu definisati kao proširenja ili suženja nekih drugih klasa. Na taj način jedna klasa nasledjuje i strukturu i ponašanje neke druge klase.

6) **Polimorfizam i dinamičko vezivanje.** Elementi programa treba da ukazuju na elemente različitih klasa i operacije mogu da budu različito realizovane u različitim klasama. Na taj način je moguće jednom elementu programa dodeliti elemente različitih tipova u toku izvršavanja programa.

7) **Višestruko i ponavljajuće nasledjivanje.** Kada se definiše neka nova klasa, ona može da nasledi strukturu i ponašanje više drugih različitih klasa. Osim toga, u njoj može da se ponavlja više puta jedna ista klasa.

Osnovni elementi koji karakterišu objektno orijentisano programiranje su:

- objekti,
- klase,
- nasledjivanje,
- polimorfizam.

1. Objekat je integralna celina podataka (tj. stanje) i procedura (tj. operacija) za rad, upravljanje i obradjivanje tih podataka. Mehanizam učaurivanja (eng. *encapsulation*) omogućava skrivanje unutrašnjih podataka objekta od spoljašnjeg pristupa, što daje mogućnost za postojanje objekata u kojima su ujedinjeni i podaci i kod.

Operacije odredjuju poruke (pozive) na koje objekat može da odgovori.

Stanje tj. vezane promenljive u objektu su skrivene od spoljašnjosti, i dostupne su samo za operacije iz tog objekta.

Promenljive koje predstavljaju unutrašnje stanje objekta zovu se **vezane promenljive**, a operacije nad njima nazivaju se **metodima**. Skup metoda objekta odredjuje njegov interfejs tj. njegovo ponašanje. Na taj način je oformljen jedan apstraktni tip podataka.

Primer 2.1. Struktura jednog opšteg objekta.

ime: objekat
    lokalne vezane promenljive
    opercije ili metodi (na koje objekat može da odgovori)

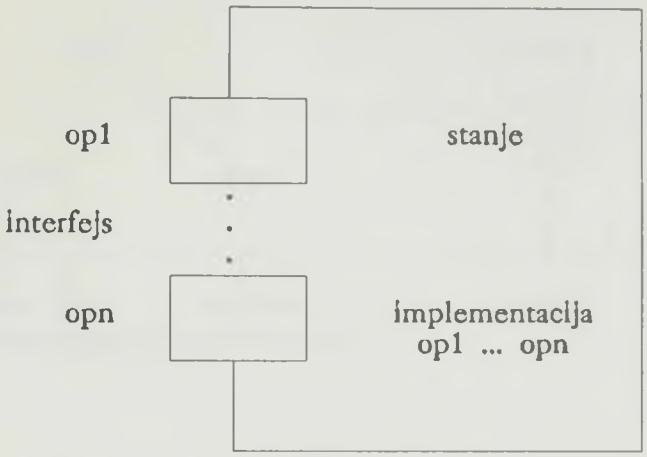
Primer 2.2. Objekt tačka, sa vezanim promenljivim x,y i metodima za njihovo čitanje i menjanje.

tačka: objekat
x := 0; y := 0;
čitaj-x : ^x; - vraća vrednost za x
čitaj-y : ^y; - vraća vrednost za y
izmeni-x(dx) : x := x + dx;
izmeni-y(dy) : y := y + dy;

Objekt zabranjuje svaki proizvoljan pristup promenljivim x i y, osim preko poruka - operacijama za čitanje i izmenu.

Interfejs kojeg čine operacije objekta može se prikazati kao slog $(op_1, op_2, \ldots, op_n)$.

Ako operacija op_i ima tip T_i, interfejs objekta je tada tipizirani slog - signatura, oblika $(op_1 : T_1, \ldots, op_n : T_n)$.



Sl 2.1 Grafički prikaz jednog opšteg objekta

Objekat **tačka** se može definisati kao tipizirani slog na sledeći način:

tačka-interfejs = (čitaj-x : Real,
čitaj-y : Real,
izmeni-x : Real -> Real,
izmeni-y : Real -> Real).

2. Klasa je skup objekata sa zajedničkim svojstvima. Klasa se takodje posmatra i tretira kao fiktivni uzorak (eng. *template*) na osnovu koga se mogu kreirati stvarni objekti.

Klasa ima iste vezane promenljive i operacije kao i objekat ali se oni različito interpretiraju: vezane promenljive u objektu predstavljaju aktuelene promenljive, a vezane promenljive klase predstavljaju potencijalne promenljive, koje se vezuju samo kada se kreira objekat te klase.

Takodje, klase odredjuju zajedničko ponašanje svih objekata koji pripadaju toj klasi. Vezane promenljive odredjuju strukturu podataka za realizaciju ponašanja, a operacije klase odredjuju njeno ponašanje.

3. Nasledjivanje je mehanizam za formiranje novih klasa od već postojećih. Nasledjivanje dozvoljava korišćenje ponašanja jedne klase u definisanju nove klase. Podklase neke klase nasledjuju vezane promenljive i operacije klase roditelja, ali se mogu proširiti dodavanjem i novih vezanih promenljivih i operacija.

Primer 2.3. Struktura klase sisara, koja se nalazi na vrhu tj. u korenu hijerarhije, i hijerarhija odgovarajućih podklasa (Slika 2.2).

U navedenom primeru klasa **ljudi** ima klasu **sisari** kao svoju nadklasu (superclass) i klase **studenti** i **žene** kao svoje podklase (subclass). Svaka od pojava tj. objekata: **ivan, mara, jova, milana, dambo** imaju jedinstvenu osnovnu klasu.

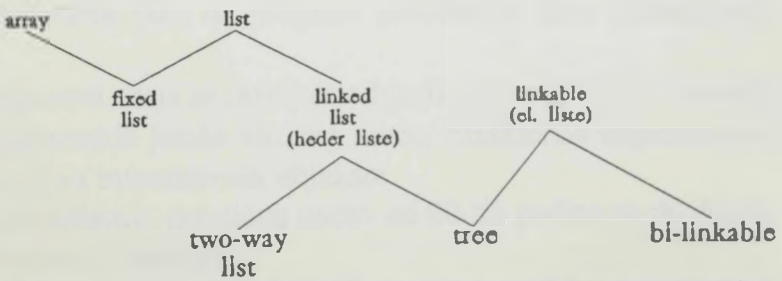
Sl 2.2 Grafički prikaz hijerarhije klase **sisari**

Nasledjivanje se ubraja u grupu orudja super apstrakcije (upravljanje objektima, izmena ponašanja) komplementarnu orudjima apstrakcije podataka.

Nasledjivanje pruža mogućnost da se izraze relacije medju ponašanjima kao što su: **klasifikacija, specijalizacija, generalizacija, aproksimacija** i **evolucija.**

U primeru 2.3. sisari se klasifikuju kao ljudi i slonovi. Slonovi specijaliziju svojstva sisara, dok sisari generalizuju svojstva slonova. Svojstva sisara aproksimiraju svojstva slonova, dok slonovi evoluiraju od sisara. Slične relacije se mogu opredeliti i za ostale klase iz prikazane hijerarhije.

Nasledjivanje klasifikuje klase slično kao što klase klasifikuju vrednosti.



Sl 2.3 Grafički prikaz hijerarhije nekih klasa u biblioteci programskog jezika Eiffel.

3. Višestruko nasledjivanje. U predhodnom primeru svaka klasa naslednik je imala samo jednu klasu roditelja. Takvo nasledjivanje se još naziva i **jednostruko nasledjivanje.** Medjutim, može se desiti da jedna klasa nasledjuje svojstva od dve i više klasa roditelja. Takvo nasledjivanje se naziva **višetruko nasledjivanje.**

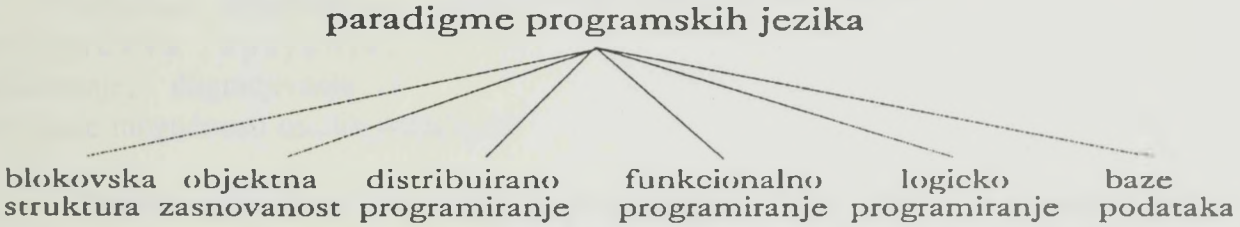
Primer 2.4. Nasledjivanja nekih klasa u biblioteci objektno orijentisanog programskog jezika Eiffel. U datoj hijerarhiji klasa FIXED-LIST, nasledjuje ponašanje dve klase i to ARRAY i LIST (Slika 2.3).

4. **Polimorfizam** je svojstvo promenljive da može da predstavlja podatke različitog tipa.

On omogućava da različite vrste objekata odgovaraju na istu poruku na različite načine jer je zasnovan na opštim operacijama u okviru odgovarajućih klasa i dinamičkom vezivanju metoda. Adresa metoda koji će se izvršiti odredjuje se u vreme izvršavanja tj. u trenutku kada objekat primi poruku.

2.1 Paradigme programskih jezika

Različiti stilovi i metodologije programiranja uslovljavaju podelu programskih jezika na različite klase tj. paradigme [Wegner, 1990].



Sl 2.4 Različite paradigme programskih jezika

Paradigmu blokovskih struktura, dominirajuću u 60-tim i 70-tim godinama, karakteriše orijentisanost ka procedurama tako da program predstavlja skup ugnježdjenih blokova i procedura.

Paradigmu objektne orijentisanosti, koja se razvijala od jezika Simula 67 do današnjih dana kada je podražava mnogo programskih jezika visokog nivoa, karakteriše orijentisanost ka objektima. Program predstavlja skup interaktivnih objekata.

Konkurentnu distribuiranu paradigmu, razvijanu počev od 60-tih godina pa do danas, karakteriše sinhronizacija, konkurentnost, monitori.

Funkcionalnu paradigmu, razvijanu takodje od 60-tih pa do danas, karakteriše jaka matematička osnova zasnovana na lambda računu. Osnovno svojstvo ove grupe jezika je odsustvo sporednih efekata, zadržano izračunavanje, ravnopravnost funkcija sa ostalim tipovima podataka.

Logičku paradigmu, razvijanu od 70-tih godina, karakterišu relacije, logičke promenljive, mehinizam rezolucije i unifikacije.

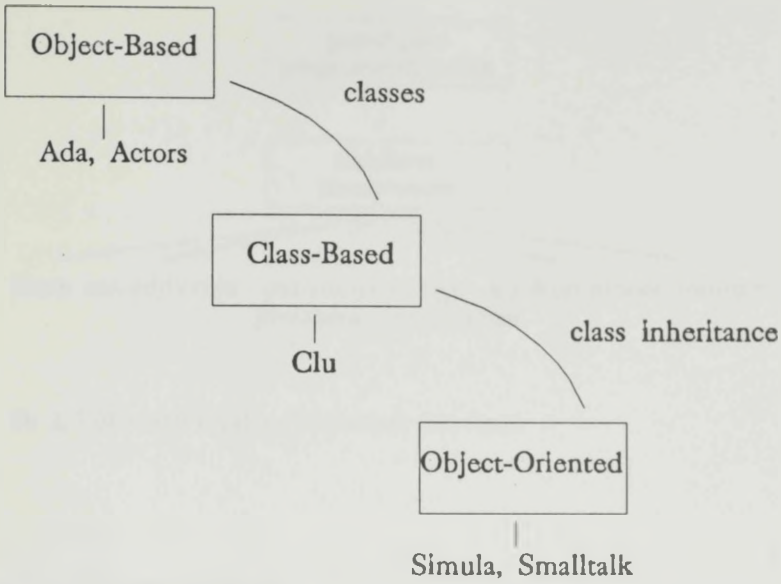
Paradigma baze podataka, je noviji pravac u razvoju programskih jezika. Karakterišu je trajnost podataka, fleksibilnost u odnosu na promenu podataka i konkurentna kontrola. Sa

stanovišta organizacije i modela podatka razvoj se kretao od hijerarhijske preko mrežne i relacione do objektno orijentisane struktuiranosti.

Navedene paradigme ne dele jezike isključivo tj. jedan jezik može istovremeno da pripada različitim paradigmama. Tako na primer sistemi Encore i O2 kombinuju paradigme objektna orijentisanost i baze podataka. Programski jezika ADA kombinuje strukturnu, konkurentnu i delimično objektnu paradigmu.

Sistemi koji podržavaju programske stilove više paradigmi tretiraju se kao **multiparadigmatski** sistemi.

Objektna orijentisanost omogućava spajanje, udruživanje, dogradjivanje i razvijanje mogućnosti ostalih paradigmi.



Sl 2.5 Paradigme objektnih programskih jezika

Objektno orijentisano programiranje je stil programiranja u kojem je ceo sistem skup objekata koji medjusobno dejstvuju. Medjitim, zavisno od elemenata koje konkretan jezik podržava, razlikuju se tri paradigme objektno orijentisanih programskih jezika [Wegner, 1990]:

1) Objektno zasnovani programski jezici - su oni jezici koji podržavaju definisanje objekata, njihovu funkcionalnost ali ne i njihovo upravljanje.

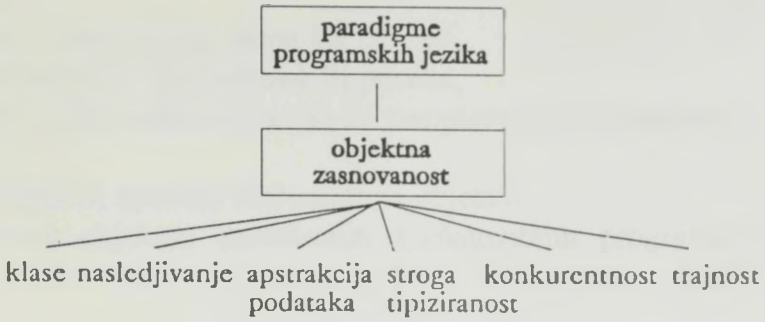
2) Klasno zasnovani jezici - su oni jezici koji podržavaju klasifikaciju svih objekata, tj. svi objekti moraju pripadati nekoj klasi. Oni podržavaju upravljanje objektima, ali ne i upravljanje klasama.

3) Objektno orijentisani jezici - su oni jezici u kojima je podržano nasledjivanje medju klasama. Oni podržavaju funkcionalnost objekata, upravljanje objektima pomoću klasa, upravljanje klasama pomoću nasledjivanja.

Stilovi rešavanja problema ove tri grupe objektno orijentisanih jezika programiranja su dovoljno različiti da garantuju tri različite paradigme.

2.2 Objektno projektovanje

Objektno orijentisana metodologija zauzima sve značajnije mesto u raznim oblastima i disciplinama. Jedan od opštih pravaca je objektno zasnovano projektovanje [Loomis, 1987].



Sl. 2.6 Elementi objektno-zasnovane paradigme

Objektno zasnovano projektovanje treba da obuhvati sledeće dimenzije:

- objekte,
- klase i tipove,
- nasledjivanje i delegaciju (delegation),
- apstrakciju podataka,
- strogu tipizirnost,
- konkurentnost,
- trajnost.

2.2.1 Objekti

Razlikuju se tri kategorije (vrste) objekata:

- funkcionalni (kao vrednosti),
- imperativni (kao promenljive),
- aktivni (kao procesi).

Funkcionalni objekti - imaju objektni interfejs, ali neizmenljivo stanje. Njihove operacije se pozivaju funkcijama u izrazima, čije izračunavanje je oslobodjeno sporednih efekata. Degenerisani objekti, koji se sastoje od skupa funkcija bez stanja, su funkcionalni objekti i oni su zastupljeni u funkcionalnim programskim jezicima.

Imperativni objekti - imaju ime, skup metoda koji se aktiviraju nakon prijema poruka od drugih objekata i vezane promenljive koje dele metodi tog objekta, a ne može im se pristupiti iz drugih objekata. U opštem slučaju, ako se objekti ili objektno orijentisani programski jezici ne odrede (tj. klasifikuju), podrazumeva se da su objekti u njima imperativni. Tradicionalni programski jezici imaju imperativne objekte.

Aktivni objekti - Imperativni objekti su pasivni dok ih ne aktivira neka poruka. Medjutim, aktivni objekti se mogu već izvršavati kada poruka stigne. Prispela poruka mora

tada da sačeka u redu. Operacije koje se izvršavaju moraju biti suspendovane nekim podzadatkom ili prioritetnom porukom.

Aktivni objekti se mogu nalaziti u tri različita režima rada:

- spreman (prikriven) - kada objekt nema šta da radi,
- aktivan - kada već izvršava neki zadatak ili poruku,
- blokiran (čekajući) - kada čeka na resurse ili kompletiranje podzadataka.

Prispele poruke se sinhronizuju sa tekućim aktivnostima objekta.

Aktivni objekti se javljaju u objektno zasnovanim konkurentnim programskim jezicima.

Kako objekti ne mogu biti jedinstveno identifikovani na osnovu njihovog ponašanja, potrebno je da postoji jedinstven identifikator objekta, koji ga razlikuje od ostalih - **identitet objekta.** Identitet objekta se razlikuje od njegove vrednosti i imena koje mu zadaje korisnik.

Jedinstveni identifikator objekta treba da je podržan na sistemskom nivou, koji će omogućiti identifikaciju nezavisnu od okruženja u kom je objekat kreiran i u kom se koristi. On takodje mora biti sakriven od korisnika.

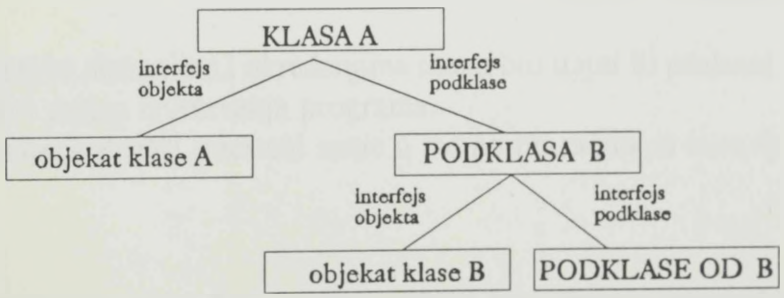
2.2.2 Tipovi, klase, nasledjivanje

Razlike izmedju tipova i klasa odgovaraju, u osnovi, razlikama izmedju strukture i ponašanja objekta.

Osnovna namena uspostavljanja (odredjivanja) tipova je:

1) odredjivanje strukture izraza za kontrolu tipa,

2) odredjivanje ponašanja klasa za razvijanje, povećavanje i izvršavanje programa.



Sl. 2.7 Jedan primer interfejsa klasa i objekata

Svaka klasa je tip, definisan tvrdjenjima koja odredjuju njeni uzorci (eng. *templates*). Medjutim, nije svaki tip klasa.

U objektno orijentisanim jezicima klase imaju dve vrste elemenata:

1) objekte - koji pristupaju operacijama klase kojoj pripadaju,

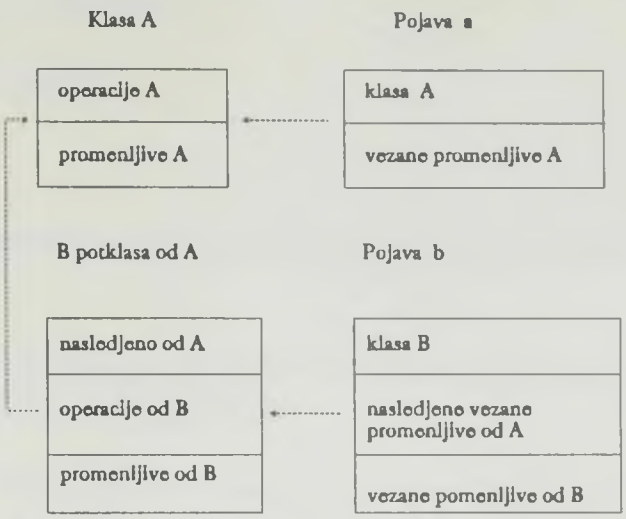
2) podklase - koje nasledjuju klase.

Primer 2.5. Nasledjivanje je mehanizam koji omogućava deljenje koda i ponašanja. Posmatrajmo klasu **A** i jednu njenu pojavu **a**, i njenu podklasu **B** i jednu njenu pojavu **b**.

Kada objekat **b** primi poruku da izvrši metod, on prvo pretražuje metode klase **B**. Ako pronadje odgovarajući metod izvršava ga sa vezanim promenljivim pojave **b**.

U suprotnom on prati pokazivač na njegovu nadklasu **A**. Ako pronadje odgovarajući metod u **A**, izvršava ga na podacima iz **b**.

U suprotnom pretražuje se nadklasa od **A**, ako je ima i tako redom. Ako **A** nema nadklasu i metod nije pronadjen javlja se greška.



Sl. 2.8 Grafički prikaz klasa i njihovih pojava

Medjutim, pitanje koje često izaziva dileme je da li nasledjivanje treba da bude jednostruko ili višestruko?

Realni svet obično nudi višestruke veze nasledjivanja, ali su one daleko komplikovanije za realizaciju i brže i češće vode u grešku. Stoga se pri dizajniranju i realizaciji objektno orijentisanih jezika mora voditi računa o svim prednostima i poteškoćama i najčešće se mora težiti kompromisnim rešenjima.

2.2.3 Trajnost

Objekti u objektno orijentisanim sistemima i okruženjima mogu biti trajni ili prolazni.

Trajni objekti ostaju u bazi i nakon izvršavanja programa.

Prolazni objekti su novokreirani objekti smešteni samo u memoriju računara tako da nestaju kada se program završi.

2.3 Oblasti primene objektno orijentisanog programiranja

Objektno orijentisana metodologija zauzima značajno mesto u projektovanju i izradi softvera, medjutim izdvajaju se oblasti u kojima se ona lakše primenjuje i efikasnije koristi.

Prirodne oblasti primene objektno orijentisanog programiranja su:

- reprezentacija znanja u veštačkoj inteligenciji, sa posebnim akcentom na:
  - simulaciju diskretnih dogadjaja,
  - inženjerstvo projektovanja,

- softversko inženjerstvo,

- konceptualno modeliranje baza podataka,

- objektno orijentisane baze podataka.

2.4 Veza objektno orijentisane metodologije i baza podataka

U dosadašnjem razvoju računarskih nauka baze podataka kao ključni elementi velikog broja softverskih proizvoda, se mogu prikazati kroz tri generacije.

- Prva generacija baza podataka iz 70-tih godina odlikovala se hijerarhijskom ili mrežnom strukturom i modelom podataka.

- Druga generacija iz 80-tih godina podržavala je relacioni model podataka.

- Treća generacija iz 90-tih godina predstavlja objektno orijentisane baze podataka.

Stoga je u poslednje vreme, jedan od značajnih trendova u računarskim naukama proširivanje objektno orijentisanih jezika tako da podržavaju baza podataka i obrnuto proširivanje baza podataka objektno orijentisanim konceptima [Blaha, 1988], [Bloom, 1987], [Rumbaugh, 1987].

Objektno orijentisani jezici, sa jedne strane, i baze podataka, sa druge strane, predstavljaju komplementarne oblasti: dok prva podržava kompleksne strukture i lokalne podatke, druga podržava deklarativniji pristup, deljenje podataka izvan domena aplikacije i veliku količinu podataka. Prirodno je stoga očekivati da će sjedinjavanje objektno orijentisanih jezika i baza podataka umnogome pojednostaviti programiranje nekih aplikacija.

Pojam klase ima različito značenje i upotrebu u bazama podataka i programskim jezicima. U programskim jezicima pojam klase se koristi da označi tip podataka (definiciju ponašanja i strukture skupa objekata). U bazi podataka klasa označava skup samih objekata. Medjutim, neki jezici baza podataka uključuju i tipove i klase da bi razlikovali skup objekata od njihovih definicija.

U toku poslednjih nekoliko godina, primena objektno orijentisane metodologije postaje značajan deo istraživanja u velikom broju disciplina u računarskim naukama, kao što su baze podataka, programski jezici, reprezentacija znanja.

I kao što je pojava frejmova, kao šema za reprezentaciju znanja, uslovila pojavu jezika za reprezentaciju znanja zasnovanih na frejmovima, tako je u oblasti baza podataka, izučavanje semantičkog modela podataka vodilo u objektno orijentisane koncepte.

Medjutim, objektna orijentisanost ne odbacuje niti negira prethodno uspostavljene koncepte, već ih uopštava i dalje dogradjuje. Objektne baze podataka uopštavaju relacione baze koristeći strukturno programiranje kad razvijanja metoda.

2.4.1 Objektno orijentisane baze podataka

Objektno orijentisane baze podataka [Premerlani, 1990], [Smith, 1987], [Won, 1990] su zadržale niz značajnih svojstava druge generacije, kao što su:

- neproceduralni pristup,
- nezavisnost podataka i
- komunikaciju sa raznim softverskim proizvodima

ali su obogaćene i nekim novim konceptima kao što su:

- multimedijalni karakter i
- podržavaje ugneždjene strukture objekata.

Multimedijalni karakter elemenata baze podataka, koji omogućava razne vrste čvorova u bazi: tekst, slika, zvuk, mešani mediji, nameće kao prirodno svojstvo, da se svaki od navedenih elemenata izrazi kao klasa u objektno orijentisanoj reprezentaciji.

Objektno orijentisane baze podataka podržavaju neke elemente upravljanja klasama, objektima i memorijom.

1. **Proširenje klase** U relacionoj bazi podataka postoji samo jedan parametrizujući tip - relacija. Operacije nad relacijama su ograničene na postavljanje i prikaz vrednosti polja (get i say).
U objektno orijentisanom pristupu svaki objekat je vezan sa klasom. Korisnički definisani tipovi, ili klase, na istom su semantičkom nivou kao ugradjeni tipovi. Interfejs objekta je prilagodjen objektu. Korisnik može da kreira nove klase od postojećih pridružujući definiciju i implementaciju te definicije.

2. **Apstrakcija podataka** U objektno orijentisanim bazama podataka, ponašanje objekta odredjeno je definicijom klase. Svaka klasa je definisana korišćenjem mehanizma apstrakcije. Ubacivanjem apstrakcije na nivo baza podataka, moguće je vršiti izmene jedne klase, bez obzira na druge klase.

3. **Mogućnost čuvanja aktivnih objekata.**

4. **Nema potreba za kopiranjem podataka u virtuelnu memoriju** Pošto je moguće slanje poruke u bazu i poziv metoda objekata, nema potreba da se neki elementi baze kopiraju u virtuelnu memoriju.

5. Automatska kontrola tipova u vreme korišćenja Kako se metode izvršavaju lokalno, objektno orijentisane baze podataka mogu da proveravaju tip njihovih argumenata u toku poziva.

2.4.2 Osnovni koncepti objektno orijentisanih baza podataka

U ovom poglavlju ukratko su prikazani osnovni koncepti koji odredjuju objektno orijentisane baze podataka.

1. Model objektno orijentisane baze podataka

Model podataka je logička organizacija entiteta realnog sveta, i relacija medju njima.

Jezik baze podataka je konkretizacija sintakse modela podataka. Baza podataka implementira model podataka. Objektno orijentisana baza podataka podržava objektno orijentisani model podataka.

U poslednjoj deceniji baze podataka su obogaćene novim dimenzijama kao što su:

- jezik upita,
- celovitost,
- sekundarno indeksiranje i klasterizacija,
- kontrola konkurentnosti i vešekorisničko okruženje.

2. Svojstva objektno orijentisanog modela podataka

- Svaki objekat ima jedinstven identifikator.

- Svaki objekat ima stanje (skup vrednosti za atribute objekta) i ponašanje (skup metoda koji operišu sa stanjem objekta).

- Klasa predstavlja grupisanje objekata koji dele isti skup atributa i metoda, i čini apstraktan tip podataka.

3. Hijerarhija klasa i nasledjivanje

Objektno orijentisana baza podataka treba da podrži jednostruko i višestruko nalsedjivanje.

4. Perspektive koncepta jezgra

Koncept klase je jedna od najvažnijih veza izmedju objektno-orijentisanih sistema i baza podataka. Klasa obuhvata bitan koncept semantičkog modela podataka a takodje služi kao osnova na kojoj se može formulisati upit.

Kako se svaka klasa može posmatrati kao objekat neke druge klase, dolazi se do pojma metaklase. Na isti način se od metaklase može doći do nove metaklase i tako redom.

Da bi se izbeglo ovo uopštavanje u neke jezike i objektno orijentisane sisteme se uvodi sistemski definisana klasa, koja je klasa svih ostalih klasa i koren hijerarhije klasa.

5. **Hijerarhijske i mrežne baze podataka**

Postoji izvesna sličnost izmedju hijerarhijskih i mrežnih i objektno orijentisanih baza podataka:

- ugnježdjena struktura objekata u objektno orijentisanim bazama podataka odgovara ugnježdjenoj strukturi slogova hijerarhijskih i mrežnih baza podataka,
- sličnost identifikatora objekta i pokazivača sloga u hijerarhijskoj bazi podataka.

Jedna od razlika je u tome što objektno orijentisane baze podataka podržavaju koncepte hijerarhija klasa, nasledjivanje i metod, dok mrežne i hijerarhijske baze podataka ne uključuju te koncepte.

6. **Semantičke baze podataka**

Relacije generalizacije i specifikacije izmedju nadklasa i podklasa, relacije argegacije izmedju klasa i njihovih atributa, i relacija izmedju pojave i klase (i nadklase i klase) su takodje uključeni u semantički model. Medjutim, u semantičkom modelu nema metoda.

7. **Relacione baze podataka**

- Objektno orijentisane baze podataka imaju (omogućavaju) hijerarhiju klasa, ugnježdjene objekte i metode, dok relacione baze podataka nemaju ni jedan od ovih koncepata.

- Objektno orijentisani model podataka nije zasnovan na elegantnoj matematičkoj teoriji, kao što je slučaj sa relacionim modelom. Ipak, veliki broj istraživača pokušava da definiše model upita i objektno orijentisanu algebru koja bi odgovarala relacionoj algebri.

2.5 Trendovi u objektno orijentisanoj metodologiji

Objektno orijentisana metodologija kako u oblasti baza podataka, tako i u ostalim oblastima, predstavlja značajan izvor daljih istraživanja, i pokriva različite koncepte.

2.5.1 Integracija baza podataka i objektno orijentisanih programskih jezika

Značajan pravac daljeg razvoja je integracija objektno orijentisanih programskih jezika sa bazama podataka, koji će podržati postojeće programiranje. Medjutim razlika izmedju

aplikacija implementiranih u objektno orijentisanim programskim jezicima i objektno orijentisanim bazama podataka, naglašenija je nego razlika izmedju objektno orijentisanih aplikacija i ne objektno orijentisanih baza podataka.

2.5.2 Arhitektura baze podataka

Objektno orijentisani koncept baze podataka zahteva ili potpuno nove tehnike ili značajna proširenja konvencionalnih tehnika za uspostavljanje arhitekture baze podataka. Uspostavljanje arhitekture i implementiranje objektno orijentisanih baza podataka, zahteva donošenje odluka koje uključuju strukturu memorije za objekte, kontrolu konkurentnosti, obradu upita, šemu definisanja i modifikacije.

Identifikator objekta - se može konstruisati na više načina, što utiče na aktivnosti u bazi vezane za upravljanje objektima.

1) Identifikator objekta je par koji obuhvata identifikator klase i identifikator pojave.

2) Identifikator objekta se sastoji samo od identifikatora pojave (u tom slučaju identifikator klase mora biti nekako registrovan).

Hijerarhija klasa i njihova kompozicija - uslovljava da je objektno orijentisani model sa elementima semantičkog modela bogatiji i sveobuhvatniji od relacionog modela.

1) Evaluacija šeme. Kod relacionih baza podataka jednostavno se izvode manipulacije sa relacijama.
Šema objektno orijentisane baze podataka ima dve nove dimenzije:

- hijerarhija klase koja obuhvata postojanje relacije generalizacije izmedju klasa i podklasa,

- hijerarhija kompozicije klasa obuhvata relaciju agregacije izmedju klasa i njenih atributa i domena atributa, tj. svaka klasa u objektno orijentisanoj bazi podataka pripada negde u hijerarhiji kompozicije klasa.

2) Upiti. Teorija ugnježdjenih relacija (kod relacionog modela) nije odgovarajuća za model upita za objektno orijentisane baze podataka jer,

- hijerarhija klasa može da formira orijentisan cikličan graf, dok kod relacionog modela važi striktna hijerarhija relacija i

- teorija ugnježdenih relacija ne podržava neke objektno-orijentisane koncepte.

Objektno orijentisani sistemi modeliraju svaki entitet realnog sveta kao objekat sa jedinstvenim identifikatorom, objekti pripadaju klasi, a klasa je smeštena negde u hijerarhiji

klasa. Ovi elementi predstavljaju osnovne predpostavke prilikom kreiranja objektno orijentisanog jezika upita.

Uprkos svim razlikama u modelu podataka, objektno orijentisani upiti se mogu izvršavati u maniru sličnom relacionim upitima.

3) Indeksiranje. Prilikom indeksiranja mora se voditi računa o hijerarhiji klasa i nasledjivanju atributa.

4) Takodje se postavlja problem dodefinisanja i prestruktuiranja autorizacije pristupa i izmena kako u objektima tako i u hijerarhiji klasa i kontroli konkurentnih pristupa pojedinim elementima strukture objektno orijentisanih baza podataka.

2.5.3 Još neki pravci razvoja

U sferi objektno orijentisanog programiranja i metodologije postoji niz problema koji otvaraju nove oblasti rada a mogu se grupisati prema sličnosti.

Formalizacija. Standardizacija objektno orijentisanih koncepata, teorijsko razvijanje i dogradjivanje notacije nasledjivanja i upita, razvoj objektno orijentisanog jezika upita koji bi bili kompatibilni sa objektno orijentisanim konceptima.

Orudja projektovanja baza podataka. Objektno orijentisani model omogućava korisniku da lakše modelira svoju aplikaciju, dok sa druge strane kompleksnost objektno orijentisane šeme ističe probleme logičkog i fizičkog projektovanja baze podataka. Stoga se javlja potreba za pomoćnim orudjima za logičko i fizičko projektovanje baza podataka.

Optimizacija često korišćenih operacija. Optimizacija performansi često korišćenih operacija.

Jezgro nezavisno od jezika. Postavlja se pitanje da li različiti sistemi baza podataka treba da se prave za svaki različit objektno orijentisani model podataka, ili ako neke aplikacije zasnovane na sličnom modelu podataka, mogu da koriste zajedničku bazu podataka, da se on iskoristi i samo dopuni nekim posebnim zahtevima.

Semantičko modeliranje. Izražena je tendencija za ujedinjavanjem snage modeliranja podataka zasnovanog na objektno orijentisanim konceptima sa konceptima semantičkog modeliranja podataka.

Dodatna svojstva baza podataka. Značajno je korišenje objektno orijentisanog pristupa u projektovanju i implementaciji informacionih sistema, što omogućava upravljanje integrisanim bazama podataka.

3 Struktura za reprezentaciju i prezentaciju informacija

HyperText metodologija i sistemi koji podržavaju 'specijalizovane' baze podataka, sa jedne strane, i objektno orijentisana metodologija, sa druge strane, otvaraju velike mogućnosti za nove pravce razvoja i povezivanje sa postojećim oblastima i istraživanjima.

Iskorišćavanje nekih dobrih svojstava i koncepata ove dve metodologije dovodi do sistema pogodnih za reprezentaciju informacija različitog tipa.

Domen reprezentacije je deo realnog sveta, koji treba modelirati, prezentirati ili rešiti uz pomoć računara. Medjutim, pre bilo kakvog rešavanja problema, modeliranja i/ili implementiranja na računaru, potrebno je domen reprezentacije opisati na adekvatan način da bi se kasnije lakše i jednostavnije obradjivao računarom. Trenutno su za takve potrebe na raspolaganju najrazličitiji mediji i uredjaji koji mogu da podrže tekstualne, grafičke, simulacione, zvučne i razne druge načine iskazivanja i opisivanja elemenata domena.

3.1 Osnovna struktura za reprezentaciju informacija

Mnoge oblasti rada obiluju informacijama i saznanjima, koja je potrebno formalizovati i sačuvati da bi se kasnije koristila za različite svrhe. Različiti problemi se opet mogu razložiti na sitnije smislene i semantički nezavisne celine. Svaka od njih može da predstavlja nezavisan i celovit domen reprezentacije.

Znanja, saznanja i informacije treba dopunjavati novim saznanjima, obogaćivati i prenositi dalje.

Ova problematika je rešavana u nizu istraživanja, ali još uvek ostavlja otvorene mogućnosti za nova, kvalitetnija rešenja.

U svakom protoku informacija razikuju se dve faze.

- **I faza** : reprezentacija informacija - tj. njihov adekvatan način zapisivanja i memorisanja.

- **II faza** : korišćenje informacija - tj. adekvatna prezentacija reprezentovanih informacija.

Primer 3.1. Posmatrajmo jedan jednostavan primer.

Proizvoljan udžbenik sadrži niz informacija i znanja o odredjenoj oblasti za koju je pisan.

Proces pisanja i štampanja udžbenika se može posmatrati kao faza reprezentacije informacija.

Kako je udžbenik namenjen procesu učenja, sa druge strane, svako ko čita taj udžbenik, učesnik je druge faze - korišćenja reprezentovanih informacija tj. učesnik je procesa prezentacije.

Primer 3.2. Sledeći primer takodje ilustruje navedene faze.

Mapa jednog grada predstavlja grafičko-tekstualni način reprezentacije tj. zapisivanja informacija i saznanja o arhitektonsko-urbanističkoj strukturi grada. Njeno kreiranje i štampanje predstavlja proces formalizacije i reprezentacije informacija.

Sa druge strane, svaki korisnik mape grada učesnik je procesa korišćenja tj. prezentacije informacija o rasporedu, nazivima i povezanosti ulica, mostova i ostalih objekata grada.

Obe faze protoka informacija - reprezentacija i prezentacija, treba da podrže i omoguće što jednostavnije, jasnije i sveobuhvatnije formalizme i načine da se izrazi neka informacija, činjenica, saznanje. Računari su jedno od orudja koja omogućavaju da se na pogodan način dodje do željene informacije, kada je ona prethodno memorisana i reprezentovana na odgovarajućim medijima i u odgovarajućem obliku.

U ovom poglavlju je predložen opšti formalizam za **reprezentaciju** informacija koji je pogodan za memorisanje različitih tipova informacija, a takodje se može jednostavno upotrebiti za različite procese **prezentacije**.

Svaki **domen reprezentacije** se može razdvojiti na sitnije semantičke celine - **teme** (eng. *topics*) koje su medjusobno povezane u celinu. Najpogodnija struktura za prikaz medjusobnih veza tema u domenu je **multidigraf**.

Definicija 3.1. Digraf (orijentisani graf) G je uredjen par (V,E), gde je V neprazan skup, a $E = \{(x,y) \mid x,y \in V\}$. Kada skup E sadrži iste elemente takav graf se naziva **multidigraf.**

Elementi skupa V su čvorovi, a elementi skupa E su grane.

Multidigraf se intuitivno opisuje kao graf koji ima samo orijentisane grane pri čemu je dozvoljeno postojanje više grana izmedju istih čvorova.

Kada se semantičke celine domena - teme opredele čvorovima multidigrafa, a veze medju temama, granama multidigrafa, tada se formira **mreža domena** - okvir za reprezentaciju informacija iz domena.

Veze izmedju tema odredjuje kreator mreže domena vodeći računa o eksplicitnoj, implicitnoj, semantičkoj ili nekoj drugoj realnoj vezi medju temama. Veze u mreži domena će takodje zavisiti i od načina njenog korišćenja u procesu prezentacije.

Razlikuju se dva osnovna načina prezentacije reprezentovanih informacija:

a) **linearna prezentacija** - koju odredjuje kreator mreže,

b) **slobodna prezentacija** - koju odredjuje korisnik mreže.

Linearna prezentacija. Kreator mreže odredjuje neke teme kao početne i iz njih se započinje proces prezentacije. Nakon toga, u zavisnosti od izbora početne teme, korisnik je dalje 'vodjen' kroz ostale teme tj. strogo je odredjen put (niz tema) kroz mrežu domena. U svakoj temi korisniku se prezentiraju reprezentovane informacije i bez ikakvog njegovog uticaja se uzima naredna grana tj. tema iz koje se nastavlja put kroz mrežu.

Neke teme se odredjuju kao završne, i kada se na putu kroz mrežu dostigne neka od njih završava se proces prezentacije.

Slobodna prezentacija. Učesniku prezentacije (korisniku) se omogućava da u svakoj temi u koju je dospeo, može da odredi temu iz koje će nastaviti prezentaciju.

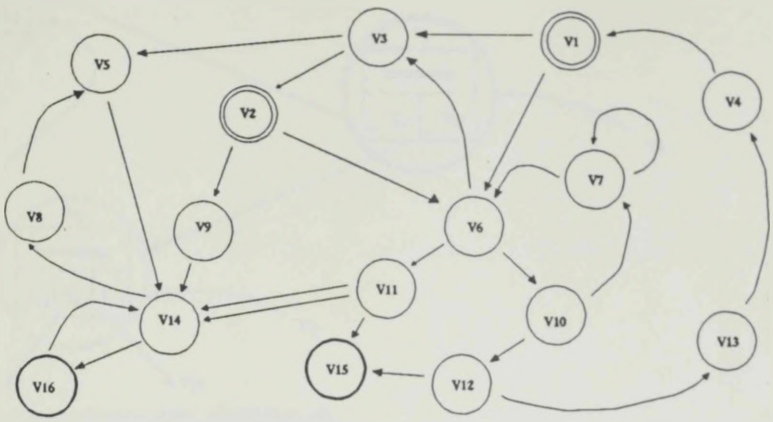
Proces prezentacije, takodje započinje u nekoj od početnih tema, a završava se dostizanjem neke od završnih tema.

Slobodne prezentacije se dele na dve osnovne grupe.

- **Slobodna prezentacija sa slobodnim izborom naredne teme** - u svakoj temi učesnik ima mogućnost da izabere narednu temu (od nekih ponudjenih i mogućih) iz koje će se dalje nastaviti započeta prezentacija.

U ovom slučaju, predloženi okvir za reprezentaciju informacija, predstavlja jezgro HyperText sistema opsanih u prvom poglavlju.

- Slobodna prezentacija sa determinisanim izborom naredne teme - u svakoj temi se učesniku procesa prezentacije postavlja zadatak ili pitanje čije rešenje odredjuje dalji tok prezentacije.

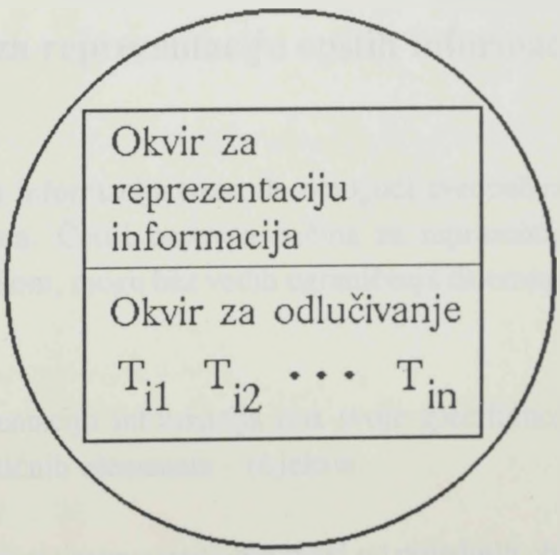


Sl 3.1. Multidigraf - globalna struktura domena reprezentacije

U oba slučaja postoji niz različitih puteva kroz mrežu, tj. niz različitih prezentacija reprezentovanih informacija, koji zavise od izbora, odluka i sposobnosti učesnika u prezentaciji.

Linearna prezentacija se može realizovati istim mehanizmima kao i slobodna prezentacija, pa se nadalje neće posebno razmatrati.

Opšta struktura nekog domena reprezentacije [Ivanović, 1991b] bi se mogla prikazati multidigrafom kao na slici 3.1.



Sl. 3.2 Osnovne celine teme

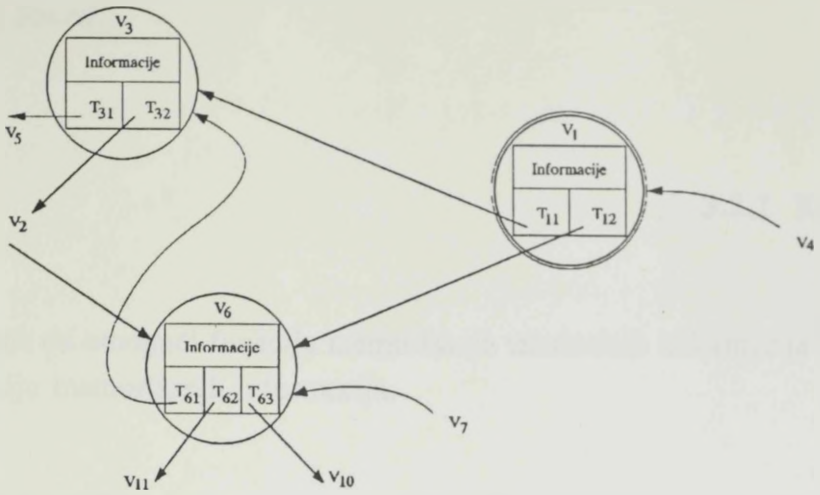
Teme v_1 i v_2 predstavljaju moguće početke prezentacije, a v_{15} i v_{16} predstavljaju teme u kojima će se prezentacija završiti. Putevi izmedju početnih i završnih tema predstavljaju različite puteve prezentacije nad istim multidigrafom. Oni su uslovljeni kako načinom odlučivanja, tako i željama i sposobnostima učesnika prezentacije.

U najopštijem slučaju svaka tema se sastoji od dva dela:

- od okvira za reprezentaciju informacija,

- od okvira za odlučivanje o daljem toku procesa prezentacije.

Detaljnija struktura opšte teme prikazana je na slici 3.2.



Sl. 3.3 Detaljnija struktura dela multidigrafa

Teme T_{i1}, T_{i2},..., T_{in} predstavljaju potencijalne teme u kojima se može nastaviti proces prezentacije, što je uslovljeno načinom odlučivanja, koji je prihvaćen u konkretnoj strukturi tj. implementaciji.

Na osnovu ovako uvedene detaljnije strukture teme, pogledajmo deteljniju šemu dela mreže domena prikazanog na slici 3.1, koja obuhvata teme v_1, v_3, i v_6 (slika 3.3).

3.2 Osnovne klase za reprezentaciju opštih informacija

Osnovne klase za reprezentaciju opštih informacija treba da omogući sveobuhvatnu reprezentaciju informacija iz različitih domena. Četiri osnovna načina za reprezentaciju informacija tekstom, slikom, animacijom i zvukom, mogu bez većih ograničenja da omoguće reprezentaciju širokog spektra domena.

Svaki od ovih načina za opis i reprezentaciju informacija ima svoje specifičnosti i svaki od njih opredeljuje i karakteriše klasu sličnih elemenata - objekata.

Koristeći se terminologijom objektno orijentisane paradigme svaki od pojedinih oblika za reprezentaciju informacija se može predstaviti kao posebna **klasa** .

Na taj način okvir za reprezentaciju informacija bi u svom osnovnom obliku (jezgro) obuhvatao:

- klasu *Text*,
- klasu *Picture*

- klasu *Animation* i
- klasu *Sound*.

3.2.1 Klasa *Text*

Klasa ***Text*** treba da omogući funkciju memorisanja tekstualnih informacija i funkciju za pogodnu prezentaciju memorisanih informacija.

Text = CLASS {

texts : promenljiva strukturnog tipa za reprezentaciju tekstualnih informacija;

interte : metod tj. operacija koja odredjuje način prezentacije reprezentovanog teksta;

}

3.2.2 Klasa *Picture*

Slično kao i klasa *Text*, klasa ***Picture*** treba da omogući formalizam za memorisanje niza slika kojima se reprezentuje domen i adekvatnu funkciju za njihovu prezentaciju.

Picture = CLASS {

pictures : promenljiva strukturnog tipa za reprezentaciju grafičkih prikaza elemenata domena;

interpic : metod koji odredjuje način prezentacije memorisanih grafičkih prikaza;

}

3.2.3 Klasa *Animation*

Klasa ***Animation*** je ekvivalentna prethodnim dvema klasama, ali je predvidjena za memorisanje trećeg oblika informacija - animacija.

Animation = CLASS {

anims : promenljiva strukturnog tipa za reprezentaciju animacija, tj. simulacija kojima se opisuje domen;

interan : metod koji odredjuje način prezentacije memorisanih animacija;

}

3.2.4 Klasa *Sound*

Klasa ***Sound*** treba da omogući memorisanje i način prezentacije zvučnih informacija.

Sound = CLASS {

sounds : promenljiva strukturnog tipa za reprezentaciju zvučnih informacija;

interso : metod koji odredjuje način prezentacije memorisanih zvučnih informacija;

}

Navedene klase su, prikazane uopšteno i globalno, bez zadržavanja na konkretnim mehanizmima za reprezentaciju informacija i metoda za njihovo prikazivanje. One predstavljaju samo okvirne elemente nekog realnog sistema za (re)prezentaciju informacija proizvoljnog domena.

Domen, kao kompleksni deo realnog sveta, teško se može prikazati samo jednim aspektom tj. ili tekstom ili slikom ili animacijom ili zvukom. Kompletan opis i sagledavanje domena zahteva i kompleksan opis njegovih elemenata, koji najčešće obuhvata sva četiri navedena aspekta.

Stoga se prirodno nameće potreba za uvodjenjem nove opšte klase koja bi uključila sva četiri aspekta i načina za reprezentaciju informacija, tj. uključila ponašanje gore definisanih klasa.

3.2.5 Klasa *Info*

Klasa ***Info*** omogućava memorisanja različitih pojavnih oblika informacija umetanjem u njenu strukturu, klasa *Text*, *Picture*, *Animation* i *Sound*.

Osim toga, klasa sadrži i dodatni metod, koji odredjuje redosled tj. raspored prezentacije teksta, slika, animacija i zvuka na ekranu: redosledno, prema prioritetu, prema specifičnim zahtevima domena, ili nekim drugim specifičnim kriterijumima.

Info = CLASS {

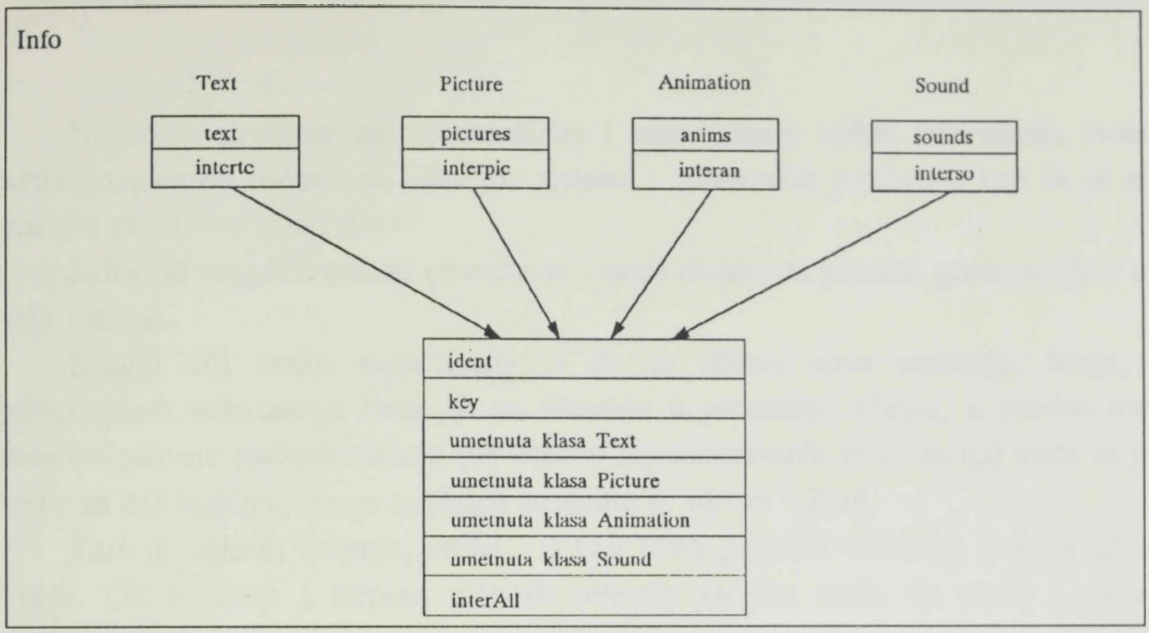
ident :	identifikator za jedinstveno odredjivanje objekta date klase;
key :	lista ključnih reči koje karakterišu reprezentovane informacije;
tex :	CLASS *Text*;
pic :	CLASS *Picture*;
ani :	CLASS *Animation*;
sou :	CLASS *Sound*;
interAll :	metod za način prezentacije svih reprezentovanih informacija;

}

Klasa *Info* sadrži sledeće promenljive.

- **ident** - vrednost promenljive će u konkretnom objektu predstavljati jedinstveni identifikator po kom će se konkretan objekat prepoznavati ukoliko je to potrebno. Ako promenljivoj nije eksplicitno dodeljena vrednost objekat će ostati neimenovan.

- **key** - je lista ključnih reči koje asociraju na sadržaje reprezentovane u konkretnom objektu. Ona pruža mogućnost odredjivanja daljeg toka prezentacije na osnovu izbora neke od raspoloživih ključnih reči.



Sl. 3.4 Grafički prikaz strukture klase Info

- **tex, pic, ani, sou** - su promenljive koje čuvaju reprezentovane informacije koje su karakteristične za konkretan objekat. Oni su tipa osnovnih klasa za reprezentaciju različitih oblika informacija. Ako nema potrebe za prikazom informacija u sva četiri oblika promenljiva se eksplicitno ili implicitno postavlja na **EMPTY** (što znači da je bez sadržaja).

Osnovne klase za reprezentaciju su umetnute u klasu *Info* i njihovim elementima se pristupa preko vezanih promenljivih iz klase *Info*.

- **interAll** - reprezentvane informacije se mogu prezentirati na različite načine, u zavisnosti od samih informacija, učesnika procesa prezentacije i niza drugih činilaca, stoga se ovom procedurom odredjuje način, dinamika i respored prezentacije pojedinih celina reprezentovanih informacija.

Struktura klase *Info* je grafički prikazana na Slici 3.4.

Klasa *Info* predstavlja osnovni element opšteg sistema za (re)prezentaciju informacija. Nekoliko mogućih sistema, zasnovanih u celini na ovoj predloženoj strukturi, će biti opisano u petom poglavlju.

3.3 Osnovne klase za odlučivanje - upotreba reprezentovanih informacija

Navedena struktura za reprezentaciju i memorisanje opštih informacija može da predstavlja polaznu osnovu za čitav niz sistema i softverskih proizvoda koji bi se mogli koristiti u različitim oblastima.

Jedna od mogućih oblasti primene su i razni obrazovni procesi, prezentacije u užem smislu i slično.

Krajnji cilj svake prezentacije je da se steknu nova saznanja. Stoga, ako reprezentovane informacije treba da se iskoriste u procesima učenja, a shodno tome i procesima provere stečenih znanja (na osnovu reprezentovanih informacija) onda se javlja potreba za definisanjem druge značajne strukture tj. okvira - *Task*.

Task je zadatak (pitanje, vežba ...) koji treba postaviti učesniku procesa učenja - provere. Od kvaliteta i stepena tačnosti rešenog zadatka može da zavisi i dalji tok prezentacije.

Kako elementi domena u procesima učenja mogu biti rezentovani u četiri pojavna oblika, to opredeljuje da i zadatak koji se postavlja, može da sadrži iste načine reprezentacije.

Osim toga, proces provere predstavlja najčešće niz zadataka na koje učesnik treba da odgovori. U zavisnosti od strukture i načina provere, zadatak treba da bude snabdeven i nizom drugih informacija.

Opšta struktura klase *Task* bi se mogla prikazati uz pomoć jednostavnijih, dosad navedenih i opisanih, klasa.

3.3.1 Klasa *BaseTask*

Klasa ***BaseTask*** treba da omogući reprezentaciju osnovnih informacija vezanih za svaki zadatak.

BaseTask = CLASS { CLASS *Info* EXCL ident,key;

type : promenljiva koja sadrži informaciju o tipu zadatka (moguće je definisati veliki broj različitih tipova zadataka);

intert : metod koji odredjuje i obezbedjuje način prezentacije zadatka i prihvatanje rešenja učesnika procesa učenja - provere;

}

Klase *BaseTask* nasledjuje klasu *Info*. Ovo nasledjivanje, medjutim, ne obuhvata celu klasu, već klasu iz koje su isključeni **ident** i **key** vezane promenljive. Promenljive iz klase *Info* su direktno dostupne samo navodjenjem njihovog imena.

U zavisnosti od tipa zadatka, javljali bi se i dodatni elementi strukture tj. klase. Tačnije za svaki tip zadatka formirala bi se nova klasa, koja uključuje ponašanje klase *BaseTask* ali i dobija nove promenljive i metode.

U poglavlju 5.2. biće prikazan jedan autorski sistem zasnovan na ovoj strukturi u kom će biti predloženo nekoliko osnovnih tipova zadataka. Ovi zadaci bi se prilikom implementacije i konkretizacije predložene strukture mogli, takodje, tretirati kao ugradjene klase.

3.3.2 Klasa *Task*

Opšti oblik klase *Task* bi se mogao prikazati na sledeći način.

```
Task = CLASS {
            ident :     STRING;

            CLASS BaseTask;

            spec  :     nove promenljive i metodi koji karakterišu konkretan tip
                        klase Task;
       }
```

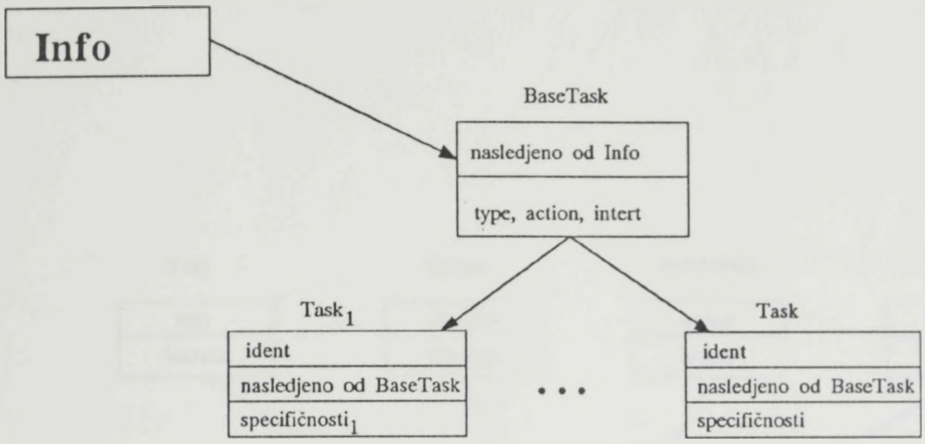
Klasa ***Task*** sadrži sledeće elemente:

- **ident** - je promenljiva u kojoj se čuva jedinstveni identifikator konkretnog objekta ove klase.

- Osnovne informacije vezane za svaki objekat tipa *Task*, se čuvaju u promenljivim koje su nasledjene od klase *BaseTask*.

- **spec** - predstavlja proizvoljan skup novih promenljivih koje detaljnije karakterišu svaki pojedinačni tip zadatka.

Struktura klase *Task* je grafički prikazana na Slici 3.5.

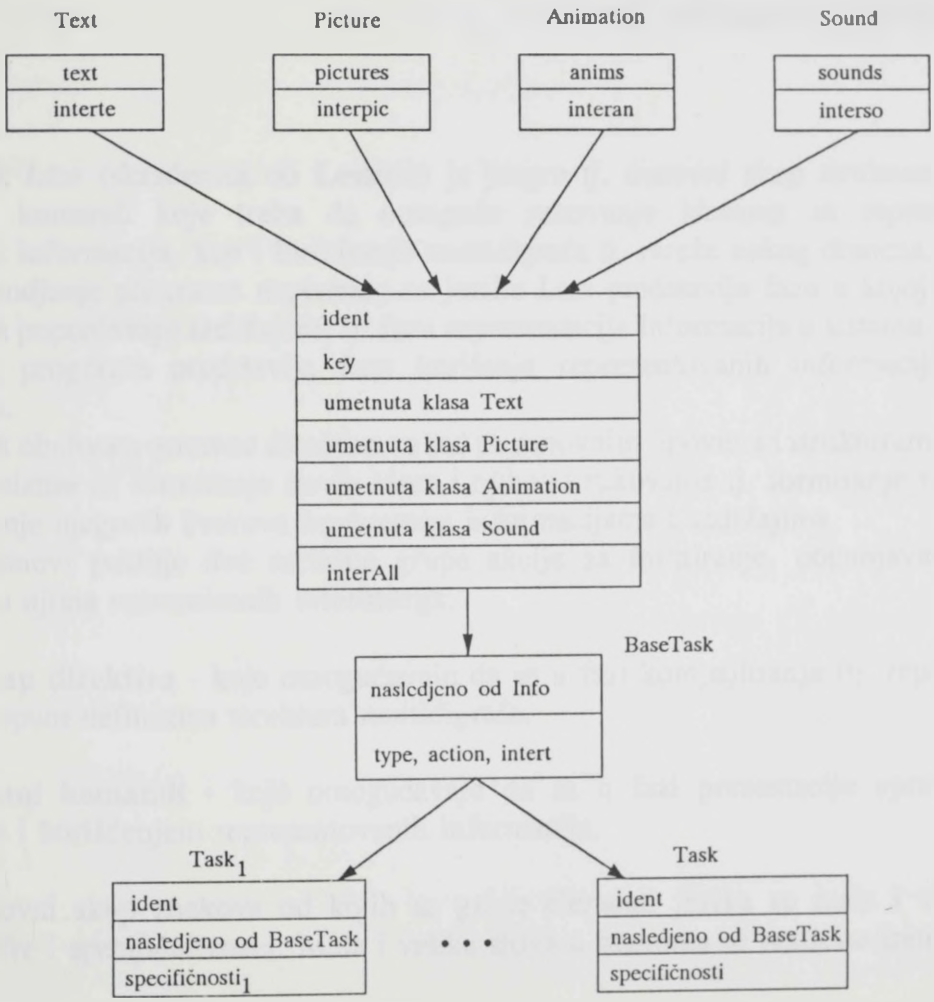


Sl 3.5 Grafički prikaz klase $Task_i$

3.4 Konačna struktura za (re)prezentaciju informacija

Navedene strukture za reprezentaciju informacija u procesima učenja i provere stečenih znanja, predstavljaju osnovu na kojoj se bazira konkretna struktura i sistem za reprezentaciju i prezentaciju informacija. Takodje, takva struktura čini osnovnu strukturu jezika za kreiranje sistema za učenje i proveru stečenih znanja.

Struktura navedenih klasa za reprezentaciju informacija grafički je prikazana na Sl. 3.6.



Sl. 3.6 Konačna struktura za (re)prezentaciju informacija

4 Osnove jezika Less

Jezik *Less* (skraćenica od Lessons) je jezgro tj. osnovni skup struktura podataka, direktiva i komandi koje treba da omoguće rukovanje klasama za reprezentaciju i prezentaciju informacija, kao i formiranje multidigrafa tj. mreže nekog domena.

Prevodjenje programa napisanog na jeziku *Less* predstavlja fazu u kojoj se čvorovi multidigrafa popunjavaju sadržajem, tj. fazu reprezentacije informacija u sistemu. Izvodjenje prevedenog programa predstavlja fazu korišenja reprezentovanih informacija, tj. fazu prezentacije.

Jezik obuhvata osnovne direktive za rad sa osnovnim tipovima i strukturama podataka, kao i mehanizme za formiranje novih klasa i njihovo rukovanje tj. formiranje multidigrafa i popunjavanje njegovih čvorova konkretnim informacijama i sadržajima.

U osnovi postoje dve različite grupe akcija za formiranje, popunjavanje klasa i korišćenje u njima memorisanih informacija.

- **skup direktiva** - koje omogućavaju da se u fazi kompajliranja (tj. reprezentacije) formira i popuni definisana struktura multidigrafa.

- **skup komandi** - koje omogućavaju da se u fazi prezentacije upravlja tokom prezentacije i korišćenjem reprezentovanih informacija.

Osnovni skup znakova od kojih se grade elementi jezika su mala i velika slova abecede, cifre i specijalni znaci. Mala i velika slova u imenima se različito tretiraju.

4.1 Osnovni tipovi podataka i struktura

Jezik *Less* poseduje niz ugradjenih primitivnih tipova i primitivnih klasa kao i načine i mehanizme njihovog gradjenja i definisanja. U nekoliko narednih poglavlja biće navedeni ti osnovni tipovi, osnovne klase i načini komponovanja tipova i definisanja novih tj. korisničkih klasa.

4.1.1 Ugradjeni tipovi podataka

Osnovni element jezika je čvor multidigrafa koji se sastoji od osnovnih klasa za reprezentaciju informacija i za odlučivanje, tj. upotrebu tih informacija.

Klasa je struktura podataka pogodna za predstavljanje znanja i informacija, a sastoji se od skupa imenovanih svojstava - **vezanih promenljivih** koje sadrže neke konkretne vrednosti, procedure ili neke druge klase.

Vrednosti promenljivih mogu da budu različitih tipova: tekstualni, grafički, zvučni itd.

Za vrednosti promenljivih dozvoljeni su sledeći ugradjeni tipovi:

- **STRING** - niz znakova ograničene dužine,

- **NUMBER** - proizvoljna numerička vrednost (celobrojna ili realna),

- **LOGICAL** - istinitosna vrednost,

- **TEXT** - formatizovani niz znakova proizvoljne dužine,

- **PICTURE** - grafički prikaz,

- **ANIMATION** - grafička animacija,

- **SOUND** - zvučna informacija,

- **IDENT TipKlase** - odredjuje jedinstveno ime objekta tipa **TipKlase**.

Za odredjivanje metoda tj. operacija u klasi predvidjeni su tipovi procedura.

- **PUBLIC PROCEDURE** - procedura tj. metod koji je zajednički za sve pojave tj. objekte iste klase.

Ako je promenljiva tipa **PUBLIC PROCEDURE**, mora se obezbediti i identifikator odgovarajuće procedure koja se vezuje za tu promenljivu. Identifikator procedure se tada navodi prilikom formiranja prvog objekta date klase.

Svaki objekat, koji sadrži promenljivu tog tipa, nasledjuje javnu proceduru i vezuje je za odgovarajuću promenljivu.

- **PRIVATE PROCEDURE** - procedura tj. metod koji je jedinstven za svaku pojavu tj. objekte iste klase.

Privatne procedure se, takodje, vezuju za promenljive. Svaki objekat, koji sadrži promenljivu tipa **PRIVATE PROCEDURE**, mora eksplicitno da veže odgovarajuću proceduru.

- **PROCEDURE** - će se koristiti u specifikaciji tipova promenljivih u definisanju klasa, kada nije fiksno i unapred odredjeno da li će nekoj promenljivoj biti dodeljena javna ili privatna procedura.

Navedeni tipovi podataka se mogu koristiti i za definisanje tipova vezanih promenljivih u klasama, i za definisanje ostalih promenljivih u programu.

4.1.2 Gradjenje podataka

Pored navedenih ugradjenih tipova postoji mogućnost njihovog vezivanja i kombinovanja.

- **LIST OF Tipovi** - lista sa proizvoljnim brojem elemenata istog tipa.

Uvodjenje ovakve strukture zahteva i nekoliko osnovnih operacija za manipulisanje elementima liste i kretanje kroz listu. Tako u jeziku treba da budu podržane neke od operacija za rad sa listama (a biće navedene u delovima koji slede).

- **ARRAY n OF Tipovi** - niz od n elemenata istog tipa. Najčešće su elementi niza ugradjeni tipovi.
Svakom elementu niza se pristupa standardno pomoću indeksa.

- **T(ime$_1$, ... , ime$_n$)** - niz elemenata koji odredjuju n-torku. **ime$_1$**, ... , **ime$_n$** su jedinstveni identifikatori nekih konkretnih objekata, koji čine n-torku.

- **(ime$_1$, ... , ime$_n$)** - niz imena koja odredjuju korisnični tip (kao nabrojivi tipovi u jeziku Modula-2).
Ovaj oblik je sastavni deo nekih direktiva gde se u zagradama navode konkretne vrednosti - imena tipova. Stoga ovaj tip neće biti dalje zasebno razmatran, već samo u okviru drugih direktiva (vidi primer 4.3.).

4.1.3 Ugradjene klase

Jezik poseduje odredjeni skup ugradjenih klasa, koje su raspoložive kao i standardni, ugradjeni tipovi podataka.
Ugradjene klase jezika *Less* čine sve klase navedene i opisane u poglavljima 3.2., 3.3. i 3.4. i to:

- Klasa *Text*,
- Klasa *Picture*,
- Klasa *Animation*,
- Klasa *Sound*,
- Klasa *Info*,
- Klasa *BaseTask*.

1. Klasa *Text*

Klasa ***Text*** omogućava memorisanje proizvoljnog broja tekstualnih informacija. Ona sadrži dve vezane promenljive od kojih je jedna lista tekstova, a druga je metod za prezentaciju reprezentovanih tektualnih informacija. Tip procedure za prezentaciju zavisiće od konkretne realizacije i sistema, a može biti javna ili privatna procedura. Za uspostavljanje konkretnog tipa procedure na raspolaganju je odgovarajuća direktiva navedena u poglavlju 4.3.4.3 (preporuka je da to bude javna procedura, jedinstvena za upravljanje prikazom tekstualnih informacija u svim objektima).

```
Text = CLASS {
                texts :      LIST OF TEXT;

                interte :    PROCEDURE;
            }
```

2. Klasa *Picture*

Klasa ***Picture*** omogućava memorisanje proizvoljnog broja grafičkih prikaza. Ona sadrži dve promenljive od kojih je jedna lista grafičkih prikaza, a druga je metod za njihovu prezentaciju. Tip procedure takodje će zavisiti od konkretne realizacije i sistema.

```
Picture = CLASS {
                pictures :    LIST OF PICTURE;

                interpic :    PROCEDURE;
            }
```

3. Klasa *Animation*

Klasa ***Animation*** omogućava memorisanje proizvoljnog broja grafičkih animacija. Ona sadrži dve promenljive od kojih je jedna lista grafičkih animacija, a druga je metod za njihovu prezentaciju. Za tip procedure važe iste napomene kao kod prethodnih klasa.

```
Animation = CLASS {
                anims :     LIST OF ANIMATION;

                interan :   PROCEDURE;
            }
```

4. Klasa *Sound*

Klasa ***Sound*** omogućava memorisanje proizvoljnog broja zvučnih informacija. Ona sadrži dve vezane promenljive od kojih je jedna lista zvučnih informacija, a druga je metod za njihovu prezentaciju. Za tip procedure važe iste napomene kao kod prethodnih klasa.

```
Sound = CLASS {
            sounds  :   LIST OF SOUND;

            interso :   PROCEDURE;
        }
```

5. Klasa *Info*

Klasa ***Info*** je ključni element i struktura jezika Less. Ona omogućava način reprezentacije i mehanizme za prezentaciju različitih pojavnih oblika informacija. Osim toga sadrži jedinstven identifikator za svaki objekt, kao i listu ključnih reči koje karakterišu informacije memorisane u konkretnom objektu. InterAll procedura opredeljuje način prezentacije i koordinirani prikaz svih informacija sačuvanih u objektu. Procedura je javna i podržava prezentaciju informacija iz svakog objekta ovog tipa.

```
Info = CLASS {
            ident   :   STRING;

            key     :   LIST OF STRING;

            tex     :   CLASS Text;

            pic     :   CLASS Picture;

            ani     :   CLASS Animation;

            sou     :   CLASS Sound;

            interAll :  PUBLIC PROCEDURE;
        }
```

Napomena : Listama LIST OF TEXT, LIST OF PICTURE, LIST OF ANIMATION, LIST OF SOUND i LIST OF STRING u ugradjenim klasama se rukuje kao nerazdvojnim celinama. Ako se radi o listi u nekim drugim klasama, onda se elementi liste razmatraju i obradjuju kao pojedinačni elementi.

6. Klasa *BaseTask*

Osnovu okvira za odlučivanje čini klasa *BaseTask*. Ona sadrži osnovne promenljive potrebe u svakom zadatku.

Klasa *BaseTask* nesledjuje klasu *Info* za reprezentaciju različitih pojavnih oblika informacija, iz koje su isključene promenljive ident i key, jer je suvišno njihovo postojanje. Ta aktivnost je omogućena navodjenjem ključne reči **EXCL** iza koje se navodi lista promenljivih, koje se isključuju iz nasledjene klase, razdvojenih zarezima.

Vezana promenljiva **type** specificira kog tipa su zadaci u tom objektu.

Vezana promenljiva **intert** ostvaruje komunikaciju sa učesnikom prezentacije i prihvata njegov odgovor na postavljeni zadatak.

```
BaseTask = CLASS {
                    CLASS Info EXCL ident,key;

                    type   :      Type_Name;

                    intert :      PROCEDURE;
           }
```

Odnosi i relacije medju ovim klasame se zadržavaju i ostaju iste onakve kako je to opisano u poglavlju 3.

4.1.4 Složene strukture jezika - gradjenje

Uvedeni ugradjeni tipovi podataka i ugradjene klase predstavljaju polazne strukture od kojih se u jeziku dalje mogu komponovati i graditi nove strukture i klase.

Kreiranje klasa. Nove klase jezika se dobijaju definisanjem njihovih vezanih promenljivih, koje nisu tipa neke klase.

Nasledjivanje. Nove klase mogu da se kreiraju potpuno nezavisno od postojećih ugradjenih klasa. Medjutim, korišćenjem mehanizma nasledjivanja, nove klase nasledjuju ponašanje ugradjenih klasa, kao i korisnički definisanih klasa. Na ovaj način vezane promenljive nasledjenih klasa se direktno uključuju u novu klasu.

Umetanje. To je drugi način da se u klasu koja se definiše uključi neka već postojeća klasa. U ovom slučaju u novodefinisanoj klasi se odredjuje nova vezana promenljiva koja je tipa neke od postojećih klasa. Vezanim promenljivim uključenih klasa se pristupa preko te uvedene vezane promenljive.

Nova klasa se gradi tako što joj se pridružuje jedinstveno ime. Iza imena klase navodi se znak = i ključna reč **CLASS**, iza koje se u zagradama {, } navode imena vezanih

promenljivih i njihovi tipovi. Klasa može da sadrži fiksni i/ili varijabilni deo, slično kao u tipu podataka RECORD jezika Modula-2.

Varijabilni deo klase započinje ključnom rečju **CASE** iz koje sledi varijabilno (težinsko) polje i njegov tip. Iza ključne reči **OF** za svaku od vrednosti težinskog polja u {, } zagradama navode se svojstva koja odgovaraju toj vrednosti težinskog polja.

Ako klasa koja se definiše nasledjuje neku već poznatu (ugradjenu ili korisnički definisanu) klasu, tada se u novodefinisanoj klasi navodi ključa reč **CLASS**, a iza nje ime željene klase. Ako se iza toga ne navodi nikakva opcija, onda se klasi koja se definiše dodeljuje poznata klasa u potpunosti.

Ako se iz klase (koja se nasledjuje) žele eksplicitno i trajno isključiti neke promenljive, tada se iza imena klase navodi ključna reč **EXCL** iza koje se navodi lista imena promenljivih koje se isključuju iz klase.

Ako je broj promenljivih koje se isključuju veći od broja promenljivih koje ostaju u klasi, tada se može koristiti druga opcija. Iza ključne reči **INCL** navodi se lista imena promenljivih koje se uključuju u klasu, razdvojenih zarezima. Preostale nenavedene promenljive se trajno isključuju iz klase.

Ako se u klasu koja se definiše umeće neka već poznata klasa, tada se navodi vezana promenljiva i iza nje ključna reč **CLASS** sa imenom klase i eventualnim opcijama.

U ostalim slučajevima iza imena promenljive se navodi njen tip.

U svim slučajevima delimiter izmedju vezane promenljive i njenog tipa je znak :.

Primer 4.1.

a) Neka je u nekoj klasi koja se definiše navedena vezana promenljiva **prikaz** na sledeći način

```
...
prikaz : CLASS Info;
...
```

U ovom slučaju u novodefinisanu klasu u potpunosti je umetnuta klasa *Info*.

b) Neka je u istoj toj klasi vezana promenljiva **prikaz** zadata na sledeći način

```
...
prikaz : CLASS Info EXCL ident, sou;
...
```

U ovom slučaju vezana promenljiva **prikaz** je tipa klase *Info* iz koje su isključene promenljive **ident** i **sou**. Sve ostale promenljive iz klase *Info* su umetnute u novodefinisanu klasu.

c) Neka je u istoj toj klasi vezana promenljiva **prikaz** zadata na sledeći način

```
...
prikaz : CLASS Info INCL key, tex, interAll;
...
```

U ovom slučaju vezana promenljiva **prikaz** je tipa klase *Info* u kojoj se zadržavaju samo vezane promenljive **key, tex** i **interAll.**

Ilustrujmo nadalje postupak kreiranja novih klasa na primeru klasa koje odredjuju temu i neke tipove zadataka.

Primer 4.2. Tema (odnosno *Topic*) je klasa koja se sastoji od dve vezane promenljive i koja nasledjuje klasu *Info*.

Vezana promenljiva **ident** je jedinstveni identifikator koji opredeljuje objekat tipa ove klase.

Vezana promenljiva **query** predstavlja listu tj. proizvoljan broj zadataka koji se vezuju za temu da bi determinisali dalji tok prezentacije.

Osim toga, klasa nasledjuje klasu *Info* iz koje je isključena jedino njena vezana promenljiva **ident**.

```
Topic = CLASS {
            ident :      STRING;
            CLASS Info EXCL ident;
            query :      LIST OF CLASS Task;
        }
```

Primer 4.3. Klasa *Action* se sastoji od vezane promenljive **ident** i od varijabilnog dela koji odredjuje tri moguća toka dalje prezentacije: kraj, prelazak na novi zadatak, prelazak na novu temu. U okviru varijabilnog dela koristi se nabrojivi tip koji definiše tri stanja prezentacije.

```
Action = CLASS {
            ident : STRING;
```

```
                CASE Type  : ( EndOfLearn, GoToTopic, GoToTask ) OF

                EndOfLearn : { END };

                GoToTopic  : { IDENT Topic1 };

                GoToTask   : { IDENT Task2 };
        }
```

END - je konstanta koja označava kraj prezentacije. *Topic*1 i *Task*2 su identifikatori koji jedinstveno odredjuju objekte tipa *Topic* i *Task*, na kojima treba da se nastavi prezentacija.

Primer 4.4. Klasa ***TaskPrompt*** odredjuje jedan specijalan tip zadatka koji učesnik prezentacije rešava ukucavanjem odgovora na prirodnom jeziku.

```
TaskPrompt = CLASS {
                        ident    :    STRING;

                        CLASS Info EXCL ident,key;

                        displayT :    PUBLIC PROCEDURE;

                        correct  :    STRING;

                        answer   :    STRING;

                        isCorr   :    PRIVATE PROCEDURE;

                        ifCorr   :    CLASS Action;

                        ifNot    :    CLASS Action;
                }
```

Klasa *TaskPrompt* se sastoji od sledećih vezanih promenljivih.

- **ident** - je promenljiva koja na jedinstven način opredeljuje svaki pojedinačni objekat te klase.

- **displayT** - je procedura koja podržava rukovanje zadatkom, prikaz informacija, prihvatanje odgovora kao i preduzimanje dalje akcije. Ona je jedinstvena za sve objekte ove klase.

- **correct** - je promenljiva čija je vrednost ispravan odgovor, koji se očekuje na postavljeni zadatak.

- **answer** - je promenljiva u koju će biti prihvaćen odgovor na zadatak koji daje učesnik prezentacije.

- **isCorr** - je privatna procedura (i može biti različita za svaki zadatak) koja utvrdjuje ispravnost odgovora koji je na postavljeni zadatak, dao učesnik prezentacije.

- **ifCorr, IfNot** - su promenljive kojima se odredjuje dalji tok prezentacije u slučaju da je ili ne, tačan odgovor koji je dao učesnik prezentacije.

- Osim toga, klasa *TaskPrompt* nasledjuje klasu *Info* bez vezanih promenljivih ident i key. Ona omogućava reprezentaciju osnovnih informacija koje čine zadatak.

4.1.5 Jedinstvena identifikacija objekata

Svaka nova klasa koja se definiše (trebala bi obavezno) da ima promenljivu **ident** tipa STRING kojom će se na jedinstven način identifikovati pojedini objekti te klase.

Ako se u klasu koja se definiše umeću neke druge klase koje takodje sadrže svoju promenljivu **ident** tada postoje dve mogućnosti da se upravlja ovim promenljivim.

1) Sve promenljive **ident** iz klasa koje se umeću se isključuju iz tih klasa (opcijom **EXCL**). Tada u klasi koja se definiše ostaje samo jedna **ident** promenljiva koja jedinstveno identifikuje objekte te klase.

2) U (nekim ili svim) klasama koje se umeću, ostaje **ident** promenljiva. Tada objekat novodefinisane klase ima svoj identifikator, a svaki od podobjekata (umetnutih klasa) ima svoj jedinstven identifikator. Tako se svakom podobjektu može pristupati i nezavisno od opšteg, u kom su sadržani.
Ovakav način umetanja, može da ukomplikuje rukovanje objektima i nije preporučljiv za uobičajeno korišćenje.

Ako klasa koja se definiše, nasledjuje neke druge klase koje takodje sadrže svoju vezanu promenljivu ident, tada se ta vezana promenljiva mora eksplicitno i obavezno isključiti iz nasledjene klase.

4.2 Gramatika tipova podataka jezika Less

Na osnovu dosadašnjih razmatranja o tipovima podataka i klasa, može se definisati odgovarajuća gramatika korišćenjem proširene Bekus-Naur forme (EBNF).

```
<Tipovi> = <Osnove_Jezika> |
           <Složene_Strukture>
```

```
<Osnove_Jezika> = <Ugradjeni_Tipovi> ¦
                  <Ugradjene_Klase>  ¦
                  <Kombinacije>

<Ugradjeni_Tipovi> = STRING ¦ NUMBER ¦ TEXT ¦ LOGICAL ¦
                     PICTURE ¦ ANIMATION ¦ SOUND ¦
                     PRIVATE PROCEDURE ¦ PUBLIC PROCEDURE ¦
                     PROCEDURE ¦ IDENT <Tip_Klase>

<Tip_Klase> = <Ime>

<Ugradjene_Klase> = Text ¦ Picture ¦ Animation ¦
                    Sound ¦ Info ¦ BaseTask

<Kombinacije> = <Liste_Nizovi> ¦
                T( <Tipovi> { , <Tipovi> } ) ¦
                ( <Ime> { , <Ime> } )

<Liste_Nizovi> = LIST OF <Tipovi> ¦
                 ARRAY <Ceo_Broj> OF <Tipovi>

<Složene_Strukture> = <Ime> = CLASS
                      { { <Fiksni_Deo> ; }
                         [ <Var_Deo> [ ; ] ] }

<Fiksni_Deo> = <Nasledjivanje> ¦
               <Umetanje> ¦
               <Ime> : <Osnove_Jezika>

<Nasledjivanje> = CLASS <Ime>
                  [ ( EXCL ¦ INCL ) <Lista_Imena> ]

<Umetanje> = Ime : <Nasledjivanje>

<Lista_Imena> = <Ime> { , <Ime> }

<Var_Deo> = CASE <IME> :
                        ( <Ugradjeni_Tipovi> ¦ <Kombinacije> ) OF
            { <Vrednost> { <Fiksni_deo> } }

<Vrednost> = <Ime>
```

4.3 Struktura programa u jeziku Less

Koristeći predloženu strukturu za reprezentaciju informacija može se konstruisati relativno jednostavan i pogodan za korišćenje, skup direktiva i komandi koje rukuju tom strukturom.

Program napisan na jeziku *Less* definiše način kreiranja i popunjavanja multidigrafa kojim se modelira odredjeni domen reprezentacije.

U toku prevodjenja programa izvršavaju se direktive pomoću kojih se popunjava definisani multidigraf.

U toku korišćenja multidigrafa procedure koje su uključene u multidigraf podržavaju i u potpunosti vode prezentaciju.

Svaki program na jeziku *Less* može da se sastoji od četiri osnovne celine - bloka:

1) blok definisanja novih klasa,

2) blok definisanja strukture multidigrafa domena,

3) blok definisanja - deklarisanja procedura - metoda,

4) blok popunjavanja multidigrafa.

Svaki blok započinje ključnim rečima koje ga jedinstveno identifikuju.

Primer 4.5. Globalna struktura programa koji sadrži sva četiri bloka.

```
BLOCK CLASSES

     definicija_klase_1;
     ...
     definicija _klase_n;

BLOCK MGRAPH

     definicija_multidigrafa_odredjenog_domena;

BLOCK PROCEDURES

     definicija_ili_deklaracija_procedure_1;
     ...
     definicija_ili_deklaracija_procedure_n;

START

     popunjavanje_multidigrafa_domena;

STOP.
```

Korišćenjem EBNF može se zapisati gramatika strukture programa u *Less* jeziku.

```
<Program> = <Block_Klasa>
            <Block_MDGrafa>
            <Block_Procedure>
            <Block_Popunjavanja>

<Block_Klasa> = [ BLOCK CLASSES ] { <Složene_Strukture> ; }

(* <Složene_Strukture> su definisane u delu 4.2. *)

<Block_MDGrafa> = [ BLOCK MGRAPH ] <Definicija_MDGrafa>
```

```
<Definicija_MDGrafa> = <Složene_Strukture> ¦
                       <Liste_Nizovi>

(* <Liste_Nizovi> su definisane u delu 4.2. *)

(* <Block_Procedure> je definisano u delu 4.3.3. *)

<Block_Popunjavanja> = START <Popunjavanje_MDGrafa> STOP

(* <Popunjavanje_MDGrafa> je definisano u delu 4.6. *)
```

4.3.1 Blok definisanja novih klasa

Blok definisanja novih klasa omogućava da se definišu i uvedu sve nove klase koje će se koristiti u strukturi multidigrafa. Nove klase se definišu na način kako je to izloženo u poglavlju 4.1.3.

Identifikaciju početka bloka predstavljaju ključne reči **BLOCK CLASSES** iza kojih se sukcesivno navode sve nove klase koje će se koristiti u ostatku programa.

```
<Blok_Klasa> = [ BLOCK CLASSES { <Složene_Strukture> ; } ] ¦
               <Promena_tipa>

<Promena_Tipa> = Retype( [ <Tip_Klase> ] [ , ]
                         [ <Spec_Prom> ] [ , ]
                         <Osnove_Jezika>

<Spec_Prom> = <Ime_Prom> ¦ & <Ime_Prom>

<Ime_Prom> = <Ime>

(* <Osnove_Jezika> je definisan u delu 4.2. *)
```

4.3.2 Blok definisanja strukture multidigrafa domena

U ovom bloku se vrši odredjivanje globalne strukture multidigrafa, kao i svih njegovih komponenti. Odredjivanje globalne strukture i imenovanje pojedinih njegovih značajnih elemenata omogućava jednostavniju identifikaciju i pristup u fazi popunjavanja.

Multidigraf se najčešće definiše kao lista (niz) objekata neke pogodne klase za reprezentaciju informacija ili kao nova klasa u kojoj postoji bar jedna promenljiva koja se definiše kao lista (niz) objekata neke druge klase pogodne za reprezentaciju i prezentaciju informacija.

4.3.3 Blok definisanja metoda - procedura

U ovom bloku se definišu ili identifikuju procedure koje će biti korišćene ili kao javne ili kao privatne u klasama koje čine konkretan multidigraf.

Definisanje tj. identifikacija procedura se sastoji ili u navodjenju imena datoteke u kojoj se nalazi procedura ili navodjenjem same procedure.

Obraćanje vezanim promenljivim klasa

Pojedinačnim promenljivim iz klase se pristupa na sledeći način:

<Ident_Objekta> { . <Pod_Objekat> } . <Prom>

<Ident_Objekta> jedinstveni identifikator objekta ili promenljiva čiji je sadržaj jedinstveni identifikator objekta, čijoj promenljivoj se pristupa. **<Pod_Objekat>** je oznaka podobjekta čijoj promenljivoj se pristupa. **<Prom>** je oznaka promenljive kojoj se pristupa. Tačka je delimiter u ovom navodjenju puta kroz koji treba da se prodje da bi se došlo do željene vezane promenljive tj. njene vrednosti.

Ako se navodi ime objekta tj. njegov jedinstveni identifikator, on za razliku od promenljive započinje znakom **$**.

Primer 4.6. Posmatrajmo klasu *Topic*, ali sa izmenjenom strukturom, tj. umesto nasledjivanja klase *Info* neka se vrši umetanje tj. klasa *Info* se vezuje za promenljivu inf1.

```
Topic = CLASS {
            ident :     STRING;

            inf1  :     CLASS Info EXCL ident;

            query :     LIST OF CLASS Task;
        }
```

Neka je formiran objekat čiji je jedinstveni identifikator **ProgJezici**, tipa klase *Topic*. Ako se želi pristup vezanoj promenljivoj **texts** (koja se nalazi u klasi *Text*, koja je umetnuta u klasu *Info*), onda se mora navesti put do nje i to tako što se navodi:

- jedinstveni identifikator objekta,
- ime promenljive za koju se vezuje podobjekat tipa *Info*,
- ime promenljive u objektu tipa *Info*, za koju se vezuje podobjekat tipa *Text*,
- i konačno ime željene promenljive.

U ovom konkretnom primeru put glasi:

$ProgJezici.inf1.tex.texts

Vrednost promenljive **texts** se može pridružiti nekoj promenljivoj ili se sa njom može direktno manipulisati u procedurama.

Ako je promenljiva tipa niza nekih elemenata tada se pojedinačnom elementu pristupa po indeksu.

Primer 4.7. Neka je klasa *Topic* definisana na sledeći način.

```
Topic = CLASS {
            ident :     STRING;
            inf1  :     CLASS Info EXCL ident;
            query :     ARRAY 5 OF CLASS TaskPrompt;
        }
```

Ako se želi pristup trećem zadatku objekta ProgJezici tipa *Topic*, onda se put navodi na sledeći način

$ProgJezici.query[3]

Pošto je query[3] objekat tipa *TaskPrompt*, onda bi se njegovoj vezanoj promenljivoj **correct** pristupalo na sledeći način

$ProgJezici.query[3].correct

Ako je promenljiva tipa, lista nekih elemenata, tada se željenom elementu pristupa ili po poziciji (ako je ona poznata), ili po jedinstvenom identifikatoru objekta (tj. elementa).

Primer 4.8. Neka je klasa *Topic* definisana na sledeći način.

```
Topic = CLASS {
            ident :     STRING;
            inf1  :     CLASS Info EXCL ident;
            query :     LIST OF CLASS TaskPrompt;
        }
```

Neka je query lista sa 5 elemenata i neka su konkretni objekti na jedinstven način identifikovani imenima Upit1, ... , Upit5.

Tada se trećem objektu može pristupiti:

1) ako se zna da je on treći u listi kao

$ProgJezici.query[3]

2) ako se ne zna njegova pozicija u listi kao

$ProgJezici.query[$Upit3].

Znakom & koji stoji ispred promenljive se označava vrednost sadržaja te promenljive. Korišenje ovog znaka omogućava da se ime ili niz imena, kao string, može čuvati u drugim promenljivim.

Tako se, na primer, put

$ProgJezici.inf1.tex.texts

može napisati na nekoliko različitih načina.

1) Neka je promenljiva pp = 'texts', tada je put oblika

$ProgJezici.inf1.tex.&pp

2) Neka je promenljiva pp = 'inf1.tex.texts', tada je put oblika

$ProgJezici.&pp

3) Neka je promenljiva pp = '$ProgeJezici.inf1.tex.texts', tada je put oblika

&pp

U narednim poglavljima biće uvedene specijalizovane direktive za rukovanje vezanim promenljivim odgovarajućih tipova. Te direktive će u najvećem broju slučajeva eliminisati potrebu za eksplicitnim navodjenjem čitavog puta do promenljivih.

<u>Deo imenovanja procedura</u>

Deo imenovanja procedura je sledećeg oblika.

Procedure(ProcName, FileName)

ili

Procedure DefProcedure;

U prvom slučaju se navodi ime procedure i ime datoteke u kojoj se nalazi kod procedure koja će se koristiti u multidigrafu.

U drugom slučaju navodi se definicija same procedure. Svaka procedura okarakterisana je ključnom rečju **Procedure** iza koje se navodi ime procedure, njeni formalni parametri i telo.

Da bi se napisala procedura u *Less* jeziku je odredjen specifičan skup komandi i tipova podataka koji mogu u njoj da se koriste. Veći deo naredbi je preuzet iz Pascal jezika [Rees, 1988], ali se može uzeti i iz nekog sličnog konvencionalnog programskog jezika.

Gramatika dela imenovanja procedura

Originalna pravila Pascal jezika su preuzeta iz [Ress, 1988] i data su na engleskom jeziku. Niz dopuna i izmena pravila u skladu sa potrebama *Less* jezika su data na srpskom jeziku.

```
<BLOCK_PROCEDURE> = { <PROCEDURES> ; }

<PROCEDURES> = <Ident_Procedure> ¦
               <PROCEDURE_DEF>

<Ident_Procedure> = Procedure ( <Ime_Proc> , <Ime_Dat> )

<Ime_Proc> = <NAME>

<Ime_Dat> = <NAME>

<PROCEDURE_DEF> = Procedure <NAME>
                  <FORMAL_PARAMETER_PART> ;
                  <BLOCK> ;

<FORMAL_PARAMETER_PART> = <EMPTY> ¦
                          <FORMAL_PARAMETER_LIST>

<FORMAL_PARAMETER_LIST> = ( <PARAMETER_PART>
                          { ; <PARAMETER_PART> } )

<PARAMETER_PART> = <PARAMETER_LIST> ¦
                   <VAR_PARAMETER_LIST>

<VAR_PARAMETER_LIST> = Var <PARAMETER_LIST>

<PARAMETER_LIST> = <IDENTIFIER_LIST> : <TYPE_IDENTIFIER_PROC>

<IDENTIFIER_LIST> = <NAME> { , <NAME> }

<TYPE_IDENTIFIER_PROC> = <NAME> ¦ IDENT [ <NAME> ]

<BLOCK> = <CONST_DECLARATION>
          <TYPE_DECLARATION>
          <VAR_DECLARATION>
          { ( <PROC_DECLARATION> ¦ <FUNC_DECLARATION> ) }
```

```
        Begin <STATEMENT_SEQUENCE> End

<CONST_DECLARATION> = <EMPTY> ¦
                      <CONST_PART>

<CONST_PART> = Const <CONST_LIST> { <CONST_LIST> }

<CONST_LIST> = <NAME> = <CONSTANT> ;

<TYPE_DECLARATION> = <EMPTY> ¦
                     <TYPE_PART>

<TYPE_PART> = Type <TYPE_LIST> { <TYPE_LIST> }

<TYPE_LIST> = <NAME> = <TYPE> ;

<TYPE> = <TYPE_IDENTIFIER> ¦
         <ARRAY_TYPE> ¦
         <RECORD_TYPE> ¦
         <Korisničke_Klase>

<TYPE_IDENTIFIER> = <INTEGER> ¦
                    <REAL> ¦
                    <BOOLEAN> ¦
                    <CHAR> ¦
                    <Ugradjeni_Tipovi> ¦
                    <Ugradjene_Klase>

(* <Ugradjeni_Tipovi> i <Ugradjene_Klase> su definisani u delu
4.2. *)

<ARRAY_TYPE> = Array [ <INDEX_TYPE> ] Of <TYPE>

<RECORD_TYPE> = Record <FIELD_PART> End

<INDEX_TYPE> = <CONSTANT> .. <CONSTANT>

<FIELD_PART> = { <IDENTIFIER_LIST> : <TYPE> }

<Korisničke_Klase> = CLASS <Ime_Klase>

<Ime_Klase> = <NAME>

<VAR_DECLARATION> = <EMPTY> ¦ <VAR_PART>

<VAR_PART> = Var <VAR_LIST> { <VAR_LIST> }

<VAR_LIST> = <IDENTIFIER_LIST> : <TYPE> ;

<FUNC_DECLARATION> = Function <NAME>
                     <FORMAL_PARAMETER_PART>
                     : <TYPE_IDENTIFIER> ; <BLOCK> ;

<STATEMENT_SEQUENCE> = <STATEMENT> { ; <STATEMENT> }
```

```
<STATEMENT> = <ASSIGNMENT_STATEMENT> ¦
              <PROCEDURE_CALL> ¦
              <COMPOUND_STATEMENT> ¦
              <IF_STATEMENT> ¦
              <CASE_STATEMENT> ¦
              <WHILE_STATEMENT> ¦
              <REPEAT_STATEMENT> ¦
              <FOR_STATEMENT> ¦
              <EMPTY>

<ASSIGNMENT_STATEMENT> = <ASSIGN_DEST> := <EXPRESSION>

<ASSIGN_DEST> = <VARIABLE> ¦
                <FUNC_IDENTIFIER>

<VARIABLE> = <VARIABLE_IDENTIFIER>
             { ( <ARRAY_ACCES> ¦ <RECORD_ACCESS> ) } ¦
             <CLASS_Access>

<ARRAY_ACCESS> = [ <INDEX_PART> ]

<RECORD_ACCESS> = . <FIELD_IDENTIFIER>

<INDEX_PART> = <EXPRESSION> { , <EXPRESSION> }

<FIELD_IDENTIFIER> = <NAME>

<CLASS_Access> = <Ident_Objekta> { . <Ident_Prom> }

<Ident_Objekta> = $ <Ime_Objekta> ¦
                  & <Prom>

<Ime_Objekta> = <NAME>

<Prom> = <NAME>

<Ident_Prom> = <Ime_prom>
               [ [ (<Ident_Objekta>¦<UNSIGNED_INTEGER>) ] ] ¦
               & <Prom>

<Ime_Prom> = <NAME>

<PROCEDURE_CALL> = <PROC_IDENTIFIER>
                   <ACTUAL_PARAMETER_PART>

<PROC_IDENTIFIER> = <NAME>

<ACTUAL_PARAMETER_PART> = <EMPTY> ¦
                          <ACTUAL_PARAMETER_LIST>

<ACTUAL_PARAMETER_LIST> = ( <EXPRESSION>
                          { , <EXPRESSION> } )

<FUNC_IDENTIFIER> = <NAME>
```

```
<COMPOUND_STATEMENT> = Begin <STATEMENT_SEQUENCE> End

<IF_STATEMENT> = <IF_THEN_STATEMENT> ¦
                 <IF_THEN_ELSE_STATEMENT>

<IF_THEN_STATEMENT> = If <EXPRESSION> Then <STATEMENT>

<IF_THEN_ELSE_STATEMENT> = If <EXPRESSION>
                           Then <STATEMENT>
                           Else <STATEMENT>

<CASE_STATEMENT> = Case <EXPRESSION> Of <CASE_PART> End

<CASE_PART> = <EMPTY> ¦
              <CASE_LIST>

<CASE_LIST> = <CASE_ARM> { ; <CASE_ARM> }

<CASE_ARM> = <CASE_LABEL_LIST> : <STATEMENT>

<CASE_LABEL_LIST> = <CONSTANT> { , <CONSTANT> }

<WHILE_STATEMENT> = While <EXPRESSION> Do <STATEMENT>

<REPEAT_STATEMENT> = Repeat
                     <STATEMENT_SEQUENCE>
                     Until <EXPRESSION>

<FOR_STATEMENT> = For <VARIABLE_IDENTIFIER> :=
                  <EXPRESSION> <TO_OR_DOWNTO>
                  <EXPRESSION> Do <STATEMENT>

<TO_OR_DOWNTO> = To ¦ Downto

<VARIABLE_IDENTIFIER> = <NAME>

<EXPRESSION> = <SIMPLE_EXPRESSION> ¦ <RELATIONAL_EXP>

<RELATIONAL_EXP> = <SIMPLE_EXPRESSION> <RELATIONAL_OP>
                   <SIMPLE_EXPRESSION>

<RELATIONAL_OP> = = ¦ <> ¦ >= ¦ > ¦ <= ¦ <

<SIMPLE_EXPRESSION> = <TERM> ¦
                      <SIGNED_TERM> ¦
                      <ADDING_EXP>

<SIGNED_TERM> = <SIGN> <TERM>

<ADDING_EXP> = <TERM> { <ADDING_OP> <TERM> }

<SIGN> = + ¦ -

<ADDING_OP> = + ¦ - ¦ or
```

```
<TERM> = <FACTOR> { <MULTIPLYING_OP> <FACTOR> }

<MULTIPLYING_OP> = * ¦ / ¦ div ¦ mod ¦ and

<FACTOR> = <UNSIGNED_CONSTANT> ¦
           <VARIABLE> ¦
           <FUNC_CALL> ¦
           <BRACK_EXP> ¦
           <NOT_FACTOR>

<FUNC_CALL> = <FUNC_IDENTIFIER> <ACTUAL_PARAMETER_PART>

<BRACK_EXP> = ( <EXPRESSION> )

<NOT_FACTOR> = Not <FACTOR>

<CONSTANT> = <CONSTANT_PART> ¦
             <CHAR_CONSTANT>

<CONSTANT_PART> = <CONSTANT_NUM_OR_ID> ¦
                  <SIGNED_CONST_NUM_OR_ID>

<SIGNED_CONST_NUM_OR_ID> = <SIGN> <CONSTANT_NUM_OR_ID>

<CONSTANT_NUM_OR_ID> = <UNSIGNED_NUMBER> ¦
                       <CONSTANT_IDENTIFIER>

<CONSTANT_IDENTIFIER> = <NAME>

<CHAR_CONSTANT> = ' <CHARACTER> '

<UNSIGNED_CONSTANT> = <CONSTANT_IDENTIFIER> ¦
                      <UNSIGNED_NUMBER> ¦
                      <CHAR_CONSTANT>

<UNSIGNED_NUMBER> = <UNSIGNED_INTEGER> <REAL_PART>

(* tip podataka NUMBER u jeziku Less je ekvivalentan tipu
podataka <UNSIGNED_NUMBER> *)

<REAL_PART> = <EMPTY> ¦
              <DECIMAL_PART> ¦
              <E_SIGNED_PART> ¦
              <E_UNSIGNED_PART>

<DECIMAL_PART> = . <UNSIGNED_INTEGER>

<E_SIGNED_PART> = E <SIGN> <UNSIGNED_INTEGER>

<E_UNSIGNED_PART> = E <UNSIGNED_INTEGER>

<UNSIGNED_INTEGER> = <DIGIT> { <DIGIT> }
```

Pored navedenih komandi i operacija dozvoljena je i upotreba svih direktiva Less jezika koje manipulišu sa multidigrafom i objektima. Deo definisanja procedura predstavlja

niz rutina koje će se aktivirati u procesu prezentacije. Tačnije, one startuju proces prezentacije i u potpunosti ga podržavaju.

4.3.4 Blok popunjavanja multidigrafa

Deo popunjavanja multidigrafa je poslednja celina svakog *Less* programa. Imenovani multidigraf se puni odgovarajućim sadržajima, pomoću niza direktiva koje se izvršavaju u toku prevodjenja *Less* programa.

Rezultat ove faze je multidigraf popunjen željenim sadržajem. U suštini razlikuje se nekoliko načina i mogućnosti popunjavanja strukture multidigrafa.

Globalna struktura dela popunjavanja je:

```
START
   FILL_IN_ALL_OBJECTS
STOP.
```

4.3.4.1 Popunjavanje ugradjenih klasa

U prethodnim poglavljima su uvedeni osnovni tipovi i klase, kao i način kreiranja tj. gradjenja novih klasa.

U nastavku je naveden načine popunjavanja tj. dodele vrednosti promenljivim primitivnih tipova i objektima primitivnih klasa.

1. Formiranje objekata zadatog tipa

Da bi se konkretan objekata ugradio u strukturu potrebno je da prethodno bude formiran. Direktiva kojom se ta akcija ostvaruje je:

- *AssignOb*(Tip, Ident)

- **Tip** - je argument kojim se specificira tip tj. ime klase, kojoj pripada objekat.

- **Ident** - je jedinstveno ime koje se pridružuje formiranom objektu.

Ako se neki objekat direktno vezuje za vezanu promenljivu nekog drugog objekta, njemu se ne mora dodeliti ime tj. on može biti anoniman. U tom slučaju objekat se ne mora eksplicitno formirati ovom direktivom.

Pri formiranju objekta sve vezane promenljive se automatski postavljaju na praznu vrednost tj. **EMPTY**. Ako je u pitanju anonimni objekat onda se mora svakoj vezanoj promenljivoj (koja nema vrednost) eksplicitno dodeliti prazna vrednost.

2. Ugradjena klasa *Text*

Kako tekst može biti proizvoljne dužine, oblika i specijalno formatizovan, on se može zadati na sledeće načine.

1) ***AssignTPAS***(SpecObjekta,SpecProm, **Text** (* konkretan tekst *) **EndText**

...

Text (* konkretan tekst *) **EndText**)

Direktiva za dodelu vrednosti tekstualnoj promenljivoj je *AssignTPAS*. Ona može da ima tri argumenta od kojih je samo poslednji obavezan.

- **SpecObjekta** - je argument kojim se omogućava jedinstvena identifikacija objekta za koji se popunjava tekstualna informacija. To može biti samo ime objekta, ili promenljiva čija je vrednost ime objekta.

- **SpecProm** - je ime vezane promenljive u okviru zadatog objekta, kojoj se dodeljuje vrednost. Ona takodje može biti samo ime vezane promenljive ili neka programska promenljiva čija vrednost je ime vezane promenljive.

- Tekstualne informacije, koje se dodeljuju promenljivoj, započinju ključnom rečju **Text**. Iza nje se navodi tekst u proizvoljnom obliku. Završetak teksta se identifikuje nailaskom na ključnu reč **EndText**. Niz tekstova koji se navode vezuju se u listu.

2) ***AssignTPAS***(SpecObjekta,SpecProm,ImeDat)

U ovom obliku direktive prva dva argumenta su ista kao i kod prvog oblika. Medjutim, niz tekstova je zadat u datoteci čije ime se navodi kao treći argument. Tekst je u datoteci zadat u istom formatu kao što je to navedeno u prvoj direktivi.

Tekst se navodi tako, da se u datom trenutku ne mora voditi računa o njegovom formatizovanju prema obliku, veličini i izgledu ekrana. Medjutim, neki delovi teksta se mogu specijalno označiti da bi se oni u fazi prezentacije prikazali u željenom obliku. Za detaljnija objašnjenja pogledati poglavlje 4.5.1.

Dodela vrednosti tj. metoda, promenljivoj tipa procedure se vrši na sledeći način:

1) ***Interpret***(SpecObjekta, SpecProm, ProcName)

Direktiva *Interpret* može da ima tri argumenta od kojih je samo poslednji obavezan.

- **SpecObjekta** - je argument kojim se omogućava jedinstvena identifikacija objekta za čiju promenljivu se vezuje procedura.

- **SpecProm** - je ime promenljive u okviru zadatog objekta, kojoj se dodeljuje konkretna procedura.

- **ProcName** - je ime procedure ili datoteke u kojoj se procedura nalazi.

Direktivom *Interpret* se vrši vezivanje konkretne procedure za zadatu promenljivu u objektu.

3. Ugradjene klase *Picture, Animation, Sound*

Vrednosti za polja u primitivnim klasama *Picture*, *Animation* i *Sound*, se zadaju slično kao u klasi *Text* na sledeće načine.

1) ***AssignTPAS***(SpecObjekta,SpecProm,KEY (* konkretna reprezentacija *) **EndKEY**

...

KEY (* konkretna reprezentacija *) **EndKEY**)

Prva dva argumenta imaju ista svojstva i ulogu kao i kod klase *Text*.
Konkretne informacije koje se dodeljuju promenljivoj započinju ključnom rečju **KEY** (tj. za objekat tipa *Picture* to je reč **Pic**, za objekat tipa *Animation* to je **Ani**, a za objekat tipa *Sound* to je **Sou**) iza koje se u odredjenom formalizmu ili na odredjen način navodi informacija (grafički prikaz, animacija, zvučna informacija). Završetak odgovarajuće informacije se identifikuje nailaskom na ključnu reč **EndKEY** (tj. za *Picture* **EndPic**, za *Animation* **EndAni**, a za *Sound* **EndSou**).

2) ***AssignTPAS***(SpecObjekta,SpecProm,ImeDat)

U ovom obliku direktive prva dva argumenta su ista kao i kod prvog oblika. Niz odgovarajućih informacija je zadat u datoteci čije ime se navodi kao treći argument. Informacije su u datoteci zadate u istom formatu kao što je to navedeno u prvom obliku direktive.

3) *AssignPAS*(SpecObjekta,SpecProm,FileName$_1$, ... , FileName$_n$)

Direktiva kojom se omogućuje dodela vrednosti promenljivoj koja predstavlja niz slika, animacija ili zvučnih informacija sačuvanih u posebnim datotekama, je *AssignPAS*.

Od svih argumenata obavezna su samo imena datoteka.

Ako iz konteksta nije jasno koji su argumenti izostavljeni potrebno je navesti zareze da bi se mogli identifikovati izostavljeni i prisutni argumenti.

- **FileName$_1$, ... , FileName$_n$** - je niz imena datoteka u kojima su zapisani pojedinačni elementi liste kojom se reprezentuje jedan oblik informacija.

U slučaju da se radi o objektu tipa *Picture*, tada se u navedenim datotekama nalaze reprezentovani grafički prikazi.

U slučaju da se radi o objektu tipa *Animation*, tada se u specificiranim datotekama nalaze reprezentovane animacije.

U slučaju da se radi o objektu tipa *Sound*, tada se u specificiranim datotekama nalaze reprezentovane zvučne informacije.

4) *AssignPAS*(SpecObjekta,SpecProm,FileName$_1$ - FileName$_n$)

Sve napomene za argumente direktive važe kao i u prethodnim oblicima.

Medjutim, kao što se vidi iz poslednja dva oblika direktiva niz imena datoteka se može zadati na dva načina.

1) Eksplicitnim navodjenjem svih imena datoteka (kao u obliku 3)).

2) Ako se imena datoteka razlikuju samo u jednom znaku, onda se navodi ime prve datoteke, potom znak '-', i iza njega ime poslednje datoteke. Imena medjudatoteka se generišu automatski (kao u obliku 4).

Direktiva za dodelu vrednosti promenljivoj tipa PROCEDURE vrši se na isti način kao kod klase *Text*.

Primer 4.9. Direktivama

```
AssignOb(Picture, 'Jezici');          (* formiranje objekta Jezici tipa
                                      Picture *)
Interpret($Jezici,pictures,slikaa - slikad),
                                      (* dodeljivanje promenljivoj
                                      pictures imenima datoteka u
                                      kojima se nalaze slike *)
```

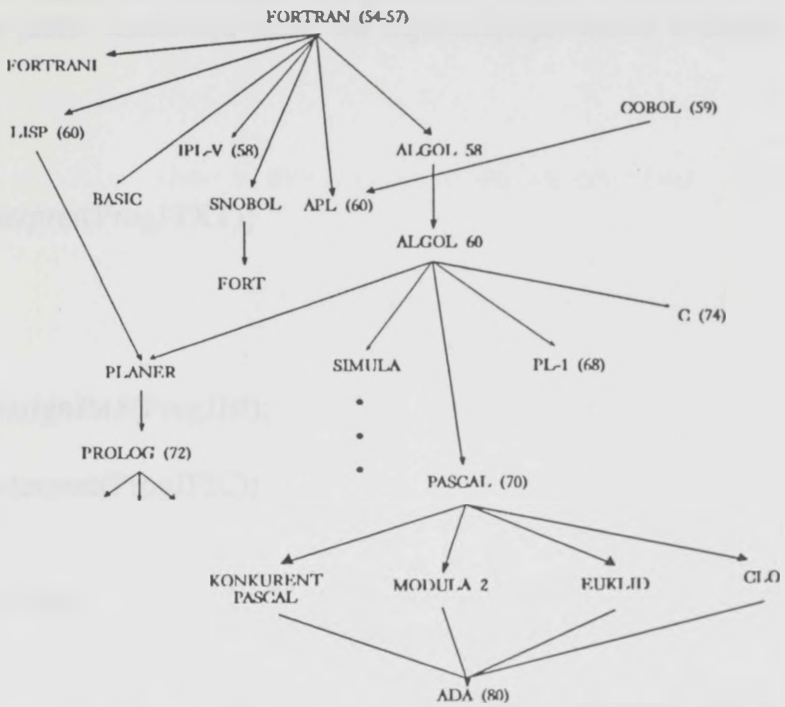
se promenljivoj pictures pridružuju imena četiri datoteke: slikaa, slikab, slikac, slikad, pomoću kojih su reprezentovane slike u objektu Jezici tipa *Picture*.

Ako se dodela vrednosti vrši direktno u objektu, tada se u komandama mogu izostaviti prva dva argumenta (pogledati primer popunjavanja klase *Info* u narednom delu).

4. Ugradjena klasa *Info* - primer popunjavanja

Koristeći uvedene direktive i organizaciju klasa mogući su različiti načini popunjavanja vezanih promenljivih neke klase željenim vrednostima. Ilustrujmo popunjavanje na dva načina, na primeru klase *Info*.

Neka je potrebno popuniti jedan objekat tipa *Info*. Neka je objekat sa jedinstvenim identifikatorom **FJezici**, definisan:



Sl. 4.1 Istorija razvoja značajnijih prog. jezika

tekstom:

Funkcionalni programski jezici su jedna od paradigmi programskih jezika. Ubrajaju se u jezike veštačke inteligencije. Ovi jezici zauzimaju jedno od najznačajnijih mesta u oblasti programskih jezika.

i slikom

na kojoj je prikazano stablo razvoja programskih jezika (slika se nalazi u datoteci ProgJIst - slika 4.1).

Neka se procedura za interpretaciju teksta nalazi u datoteci ProgJTXT, za interpretaciju slika u datoteci ProgJPIC, i procedura za globalnu interpretaciju informacija memorisanih u datoteci ProgJGLB.

Prvi način popunjavanja objekta FJezici tipa *Info*

AssignOb(*Info*,'FJezici');
(* direktivom *AssignOb* se automatski promenljivoj ident u formiranom objektu, pridružuje zadata vrednost *)

```
{
key : AssignMul(STRING,kljucne,('Funkcionalni jezici','Istorijat razvoja'));

(* pogledati direktive iz poglavlja 4.3.4.3. Opšte direktive *)

tex :
        {
                texts : AssignTPAS(Text
```

Funkcionalni programski jezici su jedna od paradigmi programskih jezika. Ubrajaju se u jezike veštačke inteligencije. Ovi jezici zauzimaju jedno od najznačajnijih mesta u oblasti programskih jezika.

```
                EndText);

                interte : Interpret(ProgJTXT);
        };

pic :
        {
                pictures : AssignPAS(ProgJIst);

                interpic : Interpret(ProgJPIC);
        };

interAll : Interpret(ProgJGLB);
};
```

U direktivama za dodelu vrednosti navode se samo obavezni argumenti jer su identifikator objekta (FJezici) i imena vezanih promenljivih poznati i nema potrebe da se navode.

Drugi način popunjavanja objekta FJezici tipa *Info*

```
AssignOb(Info,'FJezici');
AssignMul($FJezici,key,STRING,kljucne,('Funkcionalni jezici','Istorijat razvoja'));
AssignTPAS($Fjezici,tex.texts,Text
```

Funkcionalni programski jezici su jedna od paradigmi programskih jezika. Ubrajaju se u jezike veštačke inteligencije. Ovi jezici zauzimaju jedno od najznačajnijih mesta u oblasti programskih jezika.

```
EndText);

Interpret($FJezici,tex.interte,ProgJTXT);
AssignPAS($FJezici,pic.pictures,ProgJIst);
Interpret($FJezici,pic.interpic,ProgJPIC);
```

Interpret($FJezici,interAll,ProgJGLB);

Ako je klasa umetnuta u novodefinisanu klasu kao što je to slučaj sa klasama *Text*, *Picture*, *Animation* i *Sound* u klasi *Info*, tada se za svaki vezanu promenljivu podklasa mora navesti kompletan put pristupa kao što je dato u primeru.

Na primer, nije ispravna direktiva

AssignPAS($FJezici,pictures,ProgJIst)

već se ona mora navesti kao

AssignPAS($FJezici,pic.pictures,ProgJIst)

jer se vezanoj promenljivoj pictures u klasi *Pictures* pristupa preko vezane promenljive pic u klasi *Info*.

4.3.4.2 Popunjavanje ugradjenih tipova

1. **STRING** - dodela vrednosti promenljivoj tipa STRING u nekom objektu vrši se direktivom:

- *AssStr*(SpecObjekat,SpecProm,'niz znakova')

Prva dva argumenta imaju istu ulogu kao i u dosad obradjivanim direktivama i oni nisu obavezni deo direktive. Vrednost koja se dodeljuje direktivom se navodi kao treći argument. To je niz znakova ogradjen navodnicima.

2. **NUMBER** - dodela vrednosti promenljivoj tipa NUMBER u nekom objektu vrši se direktivom:

- *AssNo*(SpecObjekat,SpecProm,broj)

Slično kao i u prethodnoj direktivi prva dva argumenta nisu obavezna, a treći argument je numerička vrednost koja se dodeljuje vezanoj promenljivoj objekta.

3. **LOGICAL** - dodela vrednosti promenljivoj tipa LOGICAL u nekom objektu vrši se direktivom:

- *AssLog*(SpecObjekat,SpecProm,logička_konstanta)

Logičke konstante su **TRUE** i **FALSE**.

Globalne napomene koje su date za ostale direktive tipa Assign važe i u ovim slučajevima.

a) Direktive za rad sa listama i nizovima

Uvodjenje u jezik tipova **LIST OF** Tipovi i **ARRAY n OF** Tipovi, nameće potrebu za definisanjem niza pomoćnih direktiva za upravljanje i rukovanje takvim elementima. Predložen je minimalni skup takvih pomoćnih direktiva.

- *AssignMul*(SpecObjekta, SpecProm, SpecUgradjenogTipa, IdentListe, (ListaVrednosti)) - omogućava dodelu vrednosti elementima liste čiji su tipovi neki od ugradjenih tipova podataka.

SpecObjekta je jedinstveni identifikator objekta za koji se vrši popunjavanje vrednostima neke njegove vezane promenljive.

SpecProm je vezana promenljiva u navedenom objektu koja se popunjava.

SpecUgradjenogTipa odredjuje tip elemenata koji čine listu.

IdentListe je jedinstveni identifikator liste koja može biti formirana (pogledati sledeću direktivu).

ListaVrednosti je lista konkretnih vrednosti razdvojenih zarezima. Prva vrednost će se dodeliti prvom elementu liste, druga drugom i tako redom, poslednja poslednjem elementu lite. Broj vrednosti nije i ne mora biti odredjen unapred.

Nije obavezno navodjenje prva dva argumenta. Ako je iz konteksta jasno na koji objekat i promenljivu se direktiva odnosi, onda se ovi argumenti ne navode.
U suprotnom potrebno je navesti odgovarajući broj zareza na odgovarajućim pozicijama.

- *AssignList*(SpecTipa,IdentListe) - formira identifikator koji jedinstveno opredeljuje listu čiji su elementi zadatog tipa.

SpecTipa je tip elemenata liste koja se odredjuje i formira.

IdentListe je identifikator koji jedinstveno opredeljuje listu.

Ova direktiva nije obavezna kod formiranja liste. Navodi se samo ako je lista globalna (nije u okviru nekog objekta) tj. čini multidigraf, ili ako je lista u okviru nekog objekta ali se želi pristup njenim elementima nezavisno od objekta u kom se nalazi.

- *AssignArray*(SpecTipa,IdentNiza,BrojElemenata) - formira identifikator koji jedinstveno opredeljuje niz od BrojElemenata, čiji su elementi zadatog tipa.

SpecTipa je tip elemenata niza koji se formira.

IdentNiza je identifikator koji jedinstveno opredeljuje niz. Svaki pojedinačan element automatski poprima jedinstven identifikator koji se sastoji od identifikatora niza i indeksa koji opredeljije njegovu poziciju u nizu.

BrojElemenata je broj elementa u nizu, tj. njegova dužina.

Ova direktiva slično kao i direktiva za formiranje liste nije obavezna.

- *First*(SpecListe, SpecProm) - izdvaja prvi element liste SpecLista i pridružuje ga promenljivoj SpecProm.

- *Last*(SpecListe, SpecProm) - izdvaja poslednji element liste SpecLista i pridružuje ga promenljivoj SpecProm.

- *Next*(SpecListe, SpecProm) - izdvaja sledeći element liste SpecLista i pridružuje ga promenljivoj SpecProm.

- *FirstN*(SpecListe, SpecProm, n) - izdvaja prvih n elemenata liste SpecListe i takvu novu listu pridružuje promenljivoj SpecProm. SpecProm tada postaje jedinstven identifikator izdvojene podliste.

- *RestN*(SpecListe, SpecProm, n) - izdvaja elemente liste SpecListe od (n+1)-vog elementa do kraja i tu podlistu pridružuje promenljivoj SpecProm. SpecProm tada postaje jedinstven identifikator izdvojene podliste.

- *EndOfList*(SpecListe) - vraća logičku vrednost TRUE, kada je dostignut kraj liste SpecListe.

- *AddFirst*(SpecListe, SpecObjekta) - dodaje popunjeni objekat odgovarajućeg tipa čiji je jedinstveni identifikator SpecObjekta na početak liste SpecListe. Lista mora biti prethodno formirana (direktivom *AssignList*), a SpecObjekta je ili ime promenljive kojoj je dodeljen anoniman objekat, ili je to jedinstveni identifikator objekta ili vrednost nekog ugradjenog tipa.

- *AddLast*(SpecListe, SpecObjekta) - dodaje popunjeni objekat odgovarajućeg tipa čiji je jedinstveni identifikator SpecObjekta na kraj liste SpecListe. Lista mora biti prethodno formirana (direktivom *AssignList*), a SpecObjekta je ili ime promenljive kojoj je dodeljen anoniman objekat odgovarajućeg tipa, ili je to jedinstveni identifikator objekta ili vrednost nekog ugradjenog tipa.

b) Direktive za rad sa varijabilnim delom klase

Varijabilni deo klase sadrži kao odrednicu težinsko polje nekog korisničkog tipa. Zavisno od vrednosti težinskog polja odredjuje se konkretna specifikacija tog varijabilnog dela. Direktiva kojom se specificira vrednost težinskog polja je:

- *Tag*(SpecObjekta,SpecProm,Vrednost)

Prva dva argumenta specificiraju objekat i težinsko polje za koje se pridružuje vrednost. Ako se direktiva nalazi u samom objektu prva dva argumenta se ne moraju navoditi.

Poslednji argument je sama vrednost tj. neka vrednost korisničkog tipa kojim je okarakterisano to težinsko polje.

c) Direktive za rad sa datotekama

Ako se u programu na *Less* jeziku koristi datoteka ona se može tretirati na dva načina.

Prvi način je da se datoteka ne otvara specijalno i tada je ona namenjena za jednokratnu upotrebu tj. otvara se skriveno, iz nje se preuzima tačno odredjena informacija i datoteka se zatvara.

Na primer u slučaju direktive *AssignTPAS*(SpecObjekta,SpecProm,ImeDat) u datoteci ImeDat je zadat niz tekstova koji se iz nje učitava u jednom pristupu i nakon toga ona postaje nedostupna.

Drugi način je da se iz jednom otvorene datoteke mogu u više navrata čitati željene informacije. Svako novo obraćanje datoteci započinje od pozicije na kojoj se zaustavila u prethodnom korišćenju. Za takve potrebe datoteka se otvara direktivom:

- *OpenFile*(SpecDat,SpecTipa,Delim)

Prvi argument opredeljuje ime datoteke. Drugi argument opredeljuje tip objekata koji su u njoj memorisani. Treći argument je delimiter kojim su razdvojeni pojedinačni elementi u datoteci.

Potrebne informacije iz ove datoteke se dobijaju pomoću direktive *File*.

- *File*(SpecDat,SpecProm) - omogućava čitanje narednog elementa iz datoteke koja je specificirana argumentom **SpecDat**, i vrši njegovo pridruživanje promenljivoj koja je specificirana argumentom **SpecProm**. SpecProm može biti ili neka programska promenljiva ili vezana promenljiva iz objekta koja je specificirana po izloženim pravilima za pristup promenljivim objekata. Ako je iz konteksta jasno o kojoj promenljivoj je reč ne mora se navoditi ovaj argument.

d) Direktive za ispis

Za ispisivanje podataka na ekranu na raspoloaganju su sledeće direktive:

- *WriteSTR*(String) - ispisivanje stringa koji je zadat kao argument.

- *WriteNO*(Broj) - ispisivanje broja koji je zadat kao argument.

- *WriteLn* - prelazak u novi red.

- *WriteTXT*(PromTXT) - ispisivanje vrednosti pridruženih tekstualnoj promenljivoj.

- *WritePAS*(ArgTipaKlase) - prikazivanje informacije reprezentovane u nekom objektu tipa neke ugradjene klase: *Picture*, *Animation*, *Sound*.

e) Direktive za učitavanje

Za učitavanje akcije sa ekrana na raspolaganju je direktiva *ReadAns*.

- *ReadAns*(Prom) - omogućava da se učitana akcija smešta u specificiranu promenljivu Prom.

f) Direktive za promenu tipa

Ukoliko se želi promena tipa neke vezane promenljive u ugradjenim klasama onda se to može uraditi u delu definisanja klasa direktivom *Retype*.

- *Retype*(SpecKlase, SpecProm, NoviTip) - omogućva da se u zadatoj klasi SpecKlase tip promenljive SpecProm postavi na novi tip NoviTip.

4.3.4.4 Dodela vrednosti promenljivim

U *Less* programima se mogu koristiti i pomoćne promenljive za privremenu dodelu vrednosti. Tip takve promenljive, kao i sama promenljiva se ne uvode unapred. Tip kao i egzistecija promenljive se utvrdjuju dodelom.

Dodela vrednosti promenljivim se vrši kao u većini programskih jezika pomoću znaka =. Sa leve strane znaka = navodi se ime promenljive, a sa desne strane vrednost koja joj se dodeljuje: vrednost nekog ugradjenog tipa, vrednost neke ugradjene klase, ili vrednost neke korisnički definisane klase.

Dodela konstante tipa STRING

Neka je pomStr programska promenljiva kojoj treba dodeliti string 'programski jezici, funkcionalni programski jezici'. Ta dodela se vrši na sledeći način.

```
pomStr = 'programski jezici, funkcionalni programski jezici'
```

Dodela konstante tipa NUMBER

Neka je pomNo programska promenljiva kojoj treba dodeliti brojnu vrednost 135. Ta dodela se vrši na sledeći način.

```
pomNo = 135
```

Dodela konstante tipa LOGICAL

Neka je pomLog programska promenljiva kojoj treba dodeliti istinitosnu vrednost TRUE. Ta dodela se vrši na sledeći način.

```
pomLog = TRUE
```

Dodela konstante tipa TEXT

Neka je pomTex promenljiva kojoj treba dodeliti nekakvu tekstualnu vrednost. Ta dodela se vrši na sledeće načine.

1. Ako se odmah iz promenljive navodi željeni tekst.

pomTex = **Text** (* konkretan tekst *) **EndText**

2. Ako je željeni tekst sačuvan u nekoj datoteci.

pomTex = *AssignTPAS*(TxtDat)

Dodela konstante tipa PICTURE

Neka je pomPic programska promenljiva kojoj treba dodeliti nekakav grafički prikaz. Ta dodela se vrši na sledeće načine.

1. Ako se odmah iz promenljive navodi konkretan grafički prikaz.

pomPic = **Pic** (* konkretan grafički prikaz *) **EndPic**

2. Ako je željeni grafički prikaz sačuvan u nekoj datoteci.

pomPic = *AssignPAS*(PicDat)

Dodela konstante tipa ANIMATION

Neka je pomAni programska promenljiva kojoj treba dodeliti nekakvu animaciju. Ta dodela se vrši na sledeće načine.

1. Ako se odmah iz promenljive navodi konkretna animacija.

pomAni = **Ani** (* konkretna animacija *) *EndAni*

2. Ako je željena animacija sačuvana u nekoj datoteci.

pomAni = *AssignPAS*(AniDat)

Dodela konstante tipa SOUND

Neka je pomSou programska promenljiva kojoj treba dodeliti nekakvu zvučnu informaciju. Ta dodela se vrši na sledeće načine.

1. Ako se odmah iz promenljive navodi konkretna zvučna informacija.

pomSou = **Sou** (* konkretna animacija *) **EndSou**

2. Ako je željena zvučna informacija sačuvana u nekoj datoteci.

pomSou = *AssignPAS*(SouDat)

Dodela konstante tipa neke klase

Dodela vrednosti programskoj promenljivoj se tada vrši na sličan način kao i u prethodnim slučajevima. Sa leve strane znaka = se navodi promenljiva, a sa desne konkretan objekat, ili jedinstven identifikator nekog objekta.

Neka je pomClPic promenljiva kojoj treba dodeliti konkretan objekat. Ilustrovaćemo načine dodele za slučaj objekta tipa *Picture*. Slična dodela se vrši i za objekte brugih tipova.

1. Ako se vrši dodela objekta, tada taj objekat može biti anoniman, tj. ne mora mu se pridružiti jedinstven identifikator.

```
pomClPic = {
             pictures : AssignPas(PicDat);

             interPic : Interpret(PicProc);
           }
```

2. Ako se promenljivoj vrši dodela vrednosti korišćenjem jedinstvenog identifikatora objekta, tada taj objekat mora biti prethodno formiran i popunjen konkretnim vrednostima.

pomClPic = **IDENT** SlikaFPJ

gde je SlikaFPJ jedinstveni identifikator objekta tipa *Picture*.

4.3.4.5 Popunjavanje definisanih klasa

Dosad uvedenim direktivama su obuhvaćeni svi ugradjeni tipovi i klase i oni predstavljaju bazni skup direktiva dovoljnih za popunjavanje vrednosti korisnički definisanih klasa.

Način popunjavanja definisanih klasa je sličan kao kod popunjavanja klase *Info*.

Kod prvog načina, popunjavanje započinje formiranjem objekta odredjene klase. Iza toga se u {, } zagradama popunjavaju vrednosti za pojedine vezane promenljive. Delovi programa ogradjeni znacima (* i *) predstavljaju komentare.

Ako je vezana promenljiva nekog ugradjenog tipa, onda se njena vrednost navodi onako kako je to odredjeno za ugradjene tipove.

Ako je promenljiva tipa klasa, onda se iza njenog imena popunjava konkretan objekat te klase ili ako je već ranije popunjen konkretan objekat te klase, navodi se samo njegov jedinstven identifikator.

Varijabilni deo klase se popunjava tako što se specificira vrednost težinskog polja, a onda odgovarajuća struktura tj. akcije za konkretnu vrednost težinskog polja.

Primer 4.10. Tako bi se, na primer, klasa *Action* popunila na sledeći način:

AssignOb(Action,'SkokNaT');

{
    Tag(GoToTopic);

    { **IDENT** $Topic_1$ };
}

$Topic_1$ je jedinstveni identifikator nekog konkretnog objekta. U ovom primeru se predpostavlja da je taj objekat kreiran i da su mu dodeljene odgovarajuće vrednosti.

Primer 4.11. Klasa *TaskPrompt*, u opštem slučaju, bi se punila na sledeći način.

AssignOb(*TaskPrompt*, 'Upit');

{

    key : *AssignMul*(**STRING**,upitk,('$reč_1$', ... , '$reč_n$'));

    tex : {
        texts : *AssignTPAS*(**Text** ... **EndText**);
        interte : *Interpret*(...);
        };

    pic : {
        pictures : *AssignPAS*(...);
        interpic : *Interpret*(...);
        };

    ani : {
        anims : *AssignPAS*(...);
        interan : *Interpret*(...);
        };

    sou : {
        sounds : *AssignPAS*(...);
        interso : *Interpret*(...);
        };

    interAll : *Interpret*(...);

DisplayT : *Interpret*(...);

Correct : *AssStr*('ispravan odgovor');

(* Nema potrebe za postavljanjem

Answer : EMPTY;

jer su prilikom formiranja objekta sve vezane promenljive postavljene na prazno tj. na **EMPTY**. Ova promenljiva će se popunjavati i koristiti u toku prezentacije *)

IsCorr : *Interpret*(...);

IfCorr : **IDENT** SkokNaT;

IfNot : {
Tag(GoToTask);
{ **IDENT** $Zadatak_1$ };
};
}

Primer 4.12. Posmatrajmo popunjavanje objekta tipa *TaskPrompt*, u kom je memorisana samo tekstualna informacija, a izostavljena je lista ključnih reči i informacije tipa: *Picture*, *Animation* i *Sound*.

AssignOb(*TaskPrompt*, 'FJeziciUpit');

{
tex : {
texts : *AssignTPAS*(**Text**
U koju grupu jezika se ubrajaju funkcionalni programski jezici?
EndText);
interte : *Interpret*(PrikazUpita);
};

interAll : *Interpret*(PrikazSvega);

DisplayT : *Interpret*(KoordinirajUpit);

Correct : *AssStr*('u jezike veštačke inteligencije');

(* Answer : **EMPTY**; bez vrednosti dok se ne unese odgovor *)

IsCorr : *Interpret*(IspitajTačnost);

```
        IfCorr : IDENT OstaliJezici;        (* odlazak na neki novi čvor u
                                               multigrafu *)

        IfNot :                             (* odlazak na isti čvor u kom se nalazi i
                                               ovaj upit *)
                {
                    Tag(GoToTopic);         (* da bi se ponovila prezentacija *)
                    {IDENT FJezici};
                };
}
```

Primer 4.13. Posmatrajmo popunjavanje istog objekta kao u primeru 4.12., samo sada korišćenjem nekih pomoćnih programskih promenljivih različitih tipova.

```
AssignOb(TaskPrompt, FJeziciUpit);

pomText = Text U koju grupu jezika se ubrajaju funkcionalni programski jezici? EndText;
pomPrikaz = Interpret(PrikazUpita);

pomTex = {
            texts : pomText;
            interte : pomPrikaz;
         };

pomOdg = AssStr('u jezike veštačke inteligencije'};

pomTop = {
            Tag(GoToTopic);
            {IDENT FJezici};
         };

{
    tex : pomTex;

    interAll : Interpret(PrikazSvega);

    DisplayT : Interpret(KoordinirajUpit);

    Correct : pomOdg;

    IsCorr : Interpret(IspitajTačnost);

    IfCorr : IDENT OstaliJezici;

    IfNot : pomTop;
}
```

U prethodnim poglavljima je rečeno da gradjeni tipovi nastaju kao kompozicije nekih drugih tipova. Takodje, od pomenuta četiri načina za formiranje kompozicija najznačajniji su (i na njima ćemo se uglavnom zadržavati) liste i nizovi.

Mogu se razlikovati dva osnovna načina za njihovo popunjavanje i to:

- sekvencijalni i

- ciklični.

Sekvencijalno popunjavanje gradjenih tipova

U ovom slučaju, bilo da se radi o gradjenom tipu **LIST** bilo o **ARRAY**, svaki pojedinačni element se popunjava posebno, u sekvenci.

Za tip LIST OF Tipovi

AssignList(...); (* ustanovljavanje konkretne liste *)

(* Formiranje i/ili popunjavanje prvog elementa liste; *)
(* Dodavanje elementa na početak ili na kraj liste; *)
...
(* Formiranje i/ili popunjavanje poslednjeg elementa liste; *)
(* Dodavanje elementa na početak ili kraj liste; *)

Za tip ARRAY n OF Tipovi

AssignArray(...); (* ustanovljavanje konkretnog niza *)

(* Formiranje i/ili popunjavanje prvog elementa niza; *)
...
(* Formiranje i/ili popunjavanje poslednjeg elementa niza; *)

U ovom slučaju potpuno je nebitno da li se elementi popunjavaju baš redom prvi, drugi, treći, ... , n-ti, jer im je položaj u nizu odredjen indeksom.

Primer 4.14. Multidigraf koji čini osnovu sistema za reprezentaciju informacija, može se definisati kao lista tema (*Topic*-a). Globalna struktura programa je data u nastavku.

BLOCK CLASSES

(* definisana je klasa *Topic* *)

Topic = CLASS { ... };

BLOCK MGRAPH

(* multidigraf se definiše kao lista tema *)

LIST OF *Topic*;

BLOCK PROCEDURE

(* definisanje potrebnih procedura *)

START

AssignList(*Topic*,'MGH'); (* imenovanje liste za multidigraf MGH *)

AssignOb(*Topic*, 'FJezici'); (* formiranje objekta FJezici *)

{ (* popunjavanje objekta FJezici *) }

AddFirst($MGH,$FJezici); (* dodavalje popunjenog objekta FJezici u listu koja čini multidigraf MGH, na prvu pziciju *)

.... (* popunjavanje ostalih *Topic*-a i smeštanje u listu *)

AssignOb(*Topic*, 'AIJezici'); (* formiranje poslednjeg objekta AIJezici *)

{ (* popunjavanje poslednjeg objekta multidigrafa - AIJezici *) }

AddFirst($MGH,$AIJezici);

STOP

Ciklično popunjavanje gradjenih tipova

Korišćenjem mehanizma dodele vrednosti pomoćnim promenljivim kao i direktive za višestruko čitanje datoteke moguće je jednostavnije i kraće popunjavanje vrednosti elemenata niza ili liste.

Za tip LIST OF Tipovi

AssignList(...); (* ustanovljavanje konkretne liste *)
OpenFile(F_1); (* otvaranje potrebnih datoteka odgovarajućih tipova, za popunjavanje elemenata liste *)

OpenFile(F_m);

LoopFile
(* Popunjavanje pomoćnih programskih promenljivih konkretnim vrednostima; *)
(* Formiranje i/ili popunjavanje narednog elementa liste pomoću popunjenih programskih promenljivih; *)
(* Dodavanje elementa na početak ili na kraj liste; *)
EndLoop

Ciklus se izvršava sve dok se ne naidje na kraj datoteke, tj. kada u njoj nema više podataka.

Za tip ARRAY n OF Tipovi

AssignArray(...); (* ustanovljavanje konkretnog niza *)
OpenFile(F_1) (* otvaranje datoteke sa elementima odgovarajućeg tipa *)
...
OpenFile(F_m)

LoopFile ARRAY n
(* Popunjavanje pomoćnih programskih promenljivih konkretnim vrednostima; *)
(* Formiranje i/ili popunjavanje narednog elementa niza pomoću popunjenih programskih promenljivih; *)
EndLoop

Ciklus se izvršava n puta, a elementi niza se popunjavaju u redosledu od prvog do poslednjeg. Da bi se ciklus pravilno izvršavao, zahteva se da u datoteci ima dovoljan broj elemenata. U suprotnom doći će do pojave greške.

Primer 4.15. Posmatrajmo multidigraf kao što je to definisano u primeru 4.14. Koristeći mehanizam cikličnog popunjavanja on bi se popunio kao što je to dato u nastavku.

U datotekama $DatTip_1$, ... , $DatTip_n$ se nalaze konkretne vrednosti za pojedine vezane promenljive klase *Topic*. Odgovarajuće vrednosti nekog objekta se nalaze na logički 'istim pozicijama' u svim datotekama tj. prve vrednosti u datotekama su predodredjene za prvi objekat, sve druge za drugi objekat i tako redom.

(* prva tri poglavlja programa ostaju nepromenjena *)

START
AssignList(*Topic*,MGH);

OpenFile($DatTip_1$);
...
OpenFile($DatTip_n$);

```
    LoopFile
        (* čitanje vrednosti iz datoteka i dodela programskim promenljivim *)
        AssignOb(Topic,PomIme);   (* PomIme je promenljiva u kojoj se nalazi ime
                                     trenutnog objekta *)
        (* popunjavanje objekta učitanim vrednostima *)
        AddFirst($MGH,&PomIme);
    EndLoop;
STOP
```

4.4 Konstante jezika

Ugradjene konstante jezika su:

- **TRUE** i **FALSE** - kao standardne logičke konstante u programskim jezicima,

- **END** - konstanta koja označava kraj prezentacije,

- **EMPTY** - konstanta koja označava odustvo bilo kakvih vrednosti tj. prazno.

4.5 O načinima reprezentacije nekih primitivnih tipova

Primitivni tipovi TEXT, PICTURE, ANIMATION i SOUND se tretiraju kao imena datoteka u kojima se nalazi informacija u odgovarajućem obliku. U konkretnim slučajevima to može biti bilo koja pogodna reprezentacija i/ili formalizam.

4.5.1 Načini reprezentacije teksta

Tekstualne informacije mogu biti reprezentovane na mnogo različitih načina. Medjutim, ovde će biti predloženi samo neki načini za zapisivanje tekstualnih informacija.

a) Slobodan format teksta

Tekst će biti prezentiran u onoj formi u kojoj se unosi samo poravnat prema širini ekrana na kom se vrši prezentacija. Tekst je bez ikakvih dodatnih i naglašenih vizuelnih

efekata kao što su velika slova, indeksiranje, podvlačenje teksta i sl. To je ujedno i najjednostavniji način za reprezentaciju tekstualnih informacija.

b) Specijalni efekti

U tekst se radi lepšeg i izražajnijeg prikaza i naglašavanja pojedinih bitnih informacija, mogu unositi i odredjeni specijalni znaci. Ti specijalni znaci će se u procesu prezentacije tekstualne informacije interpretirati i na ekranu će se dobiti lepo formatiran i doteran tekst. Na primer, mogu se koristiti sledeći specijalni znaci.

1) /**B** tekst /**B** - tekst ograničen kontrolnim simbolima /B, će u toku prezentacije biti prikazan pojačano (boldovano).

2) /**I** tekst /**I** - tekst ograničen kontrolnim simbolima /I, će u toku prezentacije biti prikazan u italik obliku.

3) /**F** tekst /**F** - tekst ograničen kontrolnim simbolima /F, će u toku prezentacije biti prikazan blinkovano.

4) /**L** tekst /**L** - tekst ograničen kontrolnim simbolima /L, će u toku prezentacije biti prikazan uvećanim slovima.

5) /**X** tekst /**X** - tekst ograničen kontrolnim simbolima /X, će u toku prezentacije biti prikazan jako uvećanim slovima.

6) /**S** tekst /**S** - tekst ograničen kontrolnim simbolima /S, će u toku prezentacije biti prikazan umanjenim slovima.

7) /**U** tekst /**U** - tekst ograničen kontrolnim simbolima /U, će u toku prezentacije biti prikazan kao gornji indeks.

8) /**D** tekst /**D** - tekst ograničen kontrolnim simbolima /D, će u toku prezentacije biti prikazan kao donji indeks.

c) Format tekst procesora

Takodje, jedna od realnih mogućnosti za specijalno formatiranje teksta i postizanje specijalnih efekata, je korišćenje nekog od postojećih tekst-procesora. Na primer, može se koristiti sistem za obradu dokumenata TEX koji poseduje izuzetno bogat skup mogućnosti kako za unos običnog teksta, tako i za ispisivanje najrazličitijih matematičkih simbola i formula, formiranje tabela i čitav niz drugih mogućnosti.

4.5.2 Načini reprezentacije grafičkih prikaza

a) Grafički editori

Najjednostavniji način za formiranje i korišćenje grafičkih prikaza je uz pomoć gotovih grafičkih editora [Putnik, 1990].

Uz pomoć grafičkog editora nacrta se željena slika koja se potom sačuva u datoteku. Tako sačuvana slika se potom može jednostavno direktivama jezika uključiti u strukturu multidigrafa, a u procesu prezentacije uz pomoć odgovarajućih procedura i prezentirati [Ivanović, 1991a].

b) Formalizacija

Drugi način za formiranje grafičkih prikaza je mogućnost opisa slike korišćenjem nekih formalizama za reprezentaciju grafike [Turk, 1990], [Faconi, 1990]. Trenutno u oblasti kompjuterske grafike i geometrije postoji niz zanimljivih formalizama koji bi se mogli iskoristiti za ove potrebe.

Takodje, je trenutno značajan proces standardizacije postojećih grafičkih sistema [Klajn, 1990]. Stoga bi uključivanje i korišćenje standardnog grafičkog sistema olakšalo, kako zadavanje tj. reprezentaciju slike, tako i pisanje i korišćenje procedura za rukovanje slikama tj. njihovu prezentaciju.

4.5.3 Načini reprezentacije animacija

a) Programiranje animacija

Jedan od načina za formiranje animacija je da se uz pomoć nekih programskih jezika (opštih ili specijalizovanih) napiše program koji prilikom izvršavanja proizvodi animaciju [Budimac, 1990]. Program se može prevesti i u multidigraf vezati kao izvršna datoteka čijim startovanjem se u procesu prezentacije aktivira animacija.

b) Grafički editori

Drugi način je da se uz pomoć grafičkih editora nacrta statička slika. Pojedini delovi slike se mogu 'uvući' i uključiti u program napisan na nekom programskom jeziku. Program koristeći 'uvučeni' grafički prikaz može da vrši njegovo kretanje i na taj način se od polazne statičke slike dobija animacija. Ovaj program se takodje može prevesti i vezati u strukturu kao izvršna datoteka, koja se potom može jednostavno aktivirati u procesu prezentacije.

c) Formalizacija

Slično kao kod grafičkih prikaza može se iskoristiti neki formalizam za reprezentaciju animacija.

4.5.4 Načini reprezentacije zvučnih informacija

Zvučne informacije se takodje mogu iskoristiti u ovom sistemu uz pomoć različitih načina reprezentacije. Zavisno od hardvera koji stoji na raspolaganju za implementaciju takvih sistema, te varijante se mogu kretati do vrhunske opreme za korišćenje zvuka na računaru [Thompson, 1990].

Medjutim, bez obzira na sve mogu se uočiti uglavnom tri osnovna pravca.

a) Programiranje zvuka

Korišćenjem specijalnih formalizama i jezika moguće je pisanje programa tokom čijeg izvršavanja se javlja zvučni efekat. Tako se u poslednje vreme pojavila specijalna grupa 'muzičkih programskih jezika' (little languages) koji omogućavaju komponovanje programiranjem [Langston, 1990].

b) Memorisanje zvuka

Memorisanje zvučnih informacija u specijalne izvršne datoteke čijim aktiviranjem se realizuje memorisani zvuk.

c) Metode veštačke inteligencije

Korišćenjem savremenih hardversko-softverskih rešenja i metoda veštačke inteligencije koji omogućavaju memorisanje, korišćenje i manipulaciju govorom.

4.6 Gramatika dela popunjavanja multidigrafa

Do sada izložene strukture, tipovi podakata i direktive za rad sa njima predstavljaju osnovu jezika *Less* koji omogućuje da se na jednostavan način formira baza tj. multidigraf nekog domena reprezentacije, da se ona popuni odgovarajućim sadržajem i da se prati i podržava čitav tok prezentacije iz tako dobijene baze.

U ovom delu je korišćenjem EBNF prikazana gramatika dela popunjavanja multidigrafa.

```
<Blok_Popunjavanja> = START <Popunjavanje_MGrafa> STOP

<Popunjavanje_MGrafa> = { <Direktiva> ; }

<Direktiva> = <Formiranje_Ob> ¦
              <Ugradjene_Klase> ¦
              <Opšte_Direktive> ¦
              <Dodele> ¦
```

```
                <Definisane_Klase> ¦
                <Gradjeni_Tipovi>

<Formiranje_Ob> = AssignOb( <Tip_Klase> , <Ident_Objekta> )

<Tip_Klase> = <Ime> ¦
              <Prom_Vrednost>

<Prom_vrednost> = & <Ime>

<Ident_Objekta> = <Ident_Vrednost> ¦
                  <Prom_Vrednost>

<Ident_Vrednost> = $ <Ime>

<Ugradjene_Klase> = <AssignTPAS> ¦
                    <AssignPAS> ¦
                    <Interpret>

<AssignTPAS> = AssignTPAS( <Ident_Ob_Put> <TPAS_Vrednost> )

<Ident_Ob_Put> = [ <Ident_Objekta> ] [ , ]
                 [ <Ident_Prom> { . <Ident_Prom> } ] [ , ]

<Ident_Prom> = <Ime> ¦
               & <Ime>

<TPAS_Vrednost> = <Ime_Dat> ¦
                  <Key> <Reprezentacija> <End_Key>
                  { <Key> <Reprezentacija> <End_Key> }

<Key> = Text ¦ Pic ¦ Ani ¦ Sou

<End_Key> = EndText ¦ EndPic ¦ EndAni ¦ EndSou

<Reprezentacija> = <Odgovarajuća_Rep_Teksta> ¦
                   <Odgovarajuća_Rep_Slike> ¦
                   <Odgovarajuća_Rep_Animacije> ¦
                   <Odgovarajuća_Rep_Zvuka>

<Assign_PAS> = AssignPAS( <Ident_Ob_Put> <Specifikacija_Dat> )

<Specifikacija_Dat> = <Ime_Dat> { , <Ime_Dta> } ¦
                      <Ime_Dta> - <Ime_Dat>

<Ime_Dat> = <Ime>

<Interpret> = Interpret( <Ident_Ob_Put> <Proc_Dat_Ime> )

<Proc_Dat_Ime> = <Ime>

<Opšte_Direktive> = <Dir_Niz_Lista> ¦
                    <Dir_Datoteka> ¦
                    <Dir_Ispisa> ¦
                    <Dir_Čitanja> ¦
```

```
                    <Dir_Tipa>

<Dir_Niz_Lista> = <AssignMul> ¦
                  <AssignList> ¦
                  <AssignArray> ¦
                  <First> ¦
                  <Last> ¦
                  <Next> ¦
                  <FirstN> ¦
                  <RestN> ¦
                  <EndOfList> ¦
                  <AddFirst> ¦
                  <AddLast>

<AssignMul> = AssignMul( <Ident_Ob_Put>
                         [ <Ident_Ugradjenog_Tipa> ] [ , ]
                         [ <Ident_Liste> ] [ , ]
                         ( <Lista_Vrednosti> ) )

<Ident_Liste> = <Ime>

<Ident_Ugradjenog_Tipa> = <Osnove_Jezika>

<Lista_Vrednosti> = <Vrednost> { , <Vrednost> }

<Vrednost> = <Vrednosti_Ugradjenih_Tipova>

<AssignList> = AssignList( <Spec_Tipa> , <Ident_Liste> )

<Spec_Tipa> = <Prom_Vrednost> ¦
              <Ime>

<AssignArray> = AssignArray( <Spec_Tipa> ,
                             <Ident_Liste> ,
                             <Broj_Elemenata> )

<Broj_Elemenata> = <Broj>

<First> = First( <Spec_Liste> , <Spec_Prom> )

<Spec_Liste> = <Prom_Vrednost> ¦
               <Ime>

<Spec_Prom> = <Ime>

<Last> = Last( <Spec_Liste> , <Spec_Prom> )

<Next> = Next( <Spec_Liste> , <Spec_Prom> )

<FirstN> = FirstN( <Spec_Liste> ,
                   <Spec_Prom> ,
                   <Broj_Elemenata> )

<RestN> = RestN( <Spec_Liste> ,
                 <Spec_Prom> ,
```

```
                <Broj_Elemenata> )

<EndOfList> =  EndOfList( <Spec_Liste> )

<AddFirst> =  AddFirst( <Spec_Liste> , <Ident_Objekta> )

<AddLast> =  AddLast( <Spec_Liste> , <Ident_Objekta> )

<Dir_Datoteka> = <Otvaranje_Dat> ¦
                 <Čitanje_Dat>

<Otvaranje_Dat> = OpenFile( <Ime_Dat> ,
                            <Spec_Tipa> ,
                            <Delimiter> )

<Delimiter> = <Ime>

<Čitanje_Dat> = File( <Ime_Dat> [ , ] [ <Spec_Prom> ] )

<Dir_Ispisa> = <WriteSTR> ¦
               <WriteNO> ¦
               <WriteLn> ¦
               <WriteTXT> ¦
               <WritePAS>

<WriteSTR> = WriteSTR( <String_Vrednost> )

<String_Vrednost> = ' <Niz_Proizvoljnih_Znakova> '

<WriteNO> = WriteNO( <Broj> )

<WriteLn> = WriteLn

<WriteTXT> = WriteTXT( <Spec_Ispis_TXT> )

<Spec_Ispis_TXT> = <Spec_Prom> ¦
                   <Odgovarajuća_Rep_Teksta>

<WritePAS> = WritePAS( <Spec_Ispis_PAS> )

<Spec_Ispis_TXT> = <Spec_Prom> ¦
                   <Odgovarajuća_Rep_Slike> ¦
                   <Odgovarajuća_Rep_Animacije> ¦
                   <Odgovarajuća_Rep_Zvuka>

<Dir_Čitanja> = ReadAns( <Spec_Prom> )

<Dir_Tipa> = Retype ( [ <Spec_Klase> ] [ , ]
                      [ <Spec_Prom> ] [ , ]
                      <Tipovi> ] )

<Dodele> = <Dodela_STR_NO_LOG> ¦
           <Dodela_Ug_Def_Klasa>

<Spec_Klase> = <Prom_Vrednost> ¦
```

```
                <Ime>

<Dodela_STR_NO_LOG> = <Spec_Prom> = <Vrednost_STR_NO_LOG>

<Vrednost_STR_NO_LOG> = <Spec_Prom> ¦
                        <String_Vrednost> ¦
                        <Broj> ¦
                        <Log_Vrednost>

<Log_Vrednost> = TRUE ¦ FALSE

<Dodela_Ug_Def_Klasa> = <Spec_Prom> = <Vrednost_Ug_Def_Klasa>

<Vrednost_Ug_Def_Klasa> = <Vrednost_Ug_Klasa> ¦
                          <Vrednost_Def_Klasa>

<Vrednost_Ug_Klasa> = <AssignTPAS> ¦
                      <AssignPAS> ¦
                      <Key> <Reprezentacija> <End_Key>

<Vrednost_Def_Klasa> = [ IDENT ] <Ident_Objekta> ¦
                       { { <Dodela_Fiks_Var_Dela> } }

<Dodela_Fiks_Var_Dela> = <Dodela_Var_Dela> ¦
                         <Dodela_Fiks_Dela>

<Dodela_Fiks_Dela> = <Ime_Prom> :
                     <Dodela_Vr_Vez_Prom> ;

<Dodela_Var_Dela> = <Tag> { ( IDENT <Ident_Objekta> ¦
                            { <Dodela_Fiks_Var_Dela> } ) }

<Tag> = Tag( [ <Ident_Objekta> ] [ , ]
             [ <Put_Prom> ] [ , ]
             <Ime_Vrednosti_Tipa> )

<Put_Prom> = <Ident_Prom> { . <Ident_Prom> }

<Ime_Vrednosti_Tipa> = <Ime>

<Ime_Prom> = <Ime>

<Dodela_Vr_Vez_Prom> = <Ugradjene_Klase> ¦
                       <AssignMul> ¦
                       <Čitanje_Dat> ¦
                       <Dir_Čitanja> ¦
                       <Vrednost_Def_Klasa> ¦
                       <Vrednost_STR_NO_LOG>

<Definisane_Klase> = <Vrednost_Def_Klasa>

<Gradjeni_Tipovi> = LoopFile [ ARRAY <Broj> ]
                    <Popunjavanje_MGrafa>
                    EndLoop
```

5 Konkretni primeri generisani nad opštom strukturom i jezikom Less

Opšta struktura podataka za reprezentaciju (multidigrafa), opisana u trećoj glavi, može da posluži kao dobra osnova za razvoj niza konkretnih sistema.

Takodje, opšti jezik *Less* kao polazno jezgro, može da posluži za razvijanje niza biblioteka definicija struktura za reprezentaciju i procedura za prezentaciju, uz čiju pomoć bi se mogli popunjavati i koristiti razni sistemi za reprezentaciju i prezentaciju informacija.

U narednim poglavljima biće prikazana tri konkretna sistema, u potpunosti realizovana korišenjem struktura opisanih u poglavlju 3 i programskog jezika opisanog u poglavlju 4. Ovi sistemi se svrstavaju u tri različite grupe sistema za reprezentaciju i prezentaciju informacija koji su opisani u poglavlju 1.

1) HyperText sistem - na jednom konkretnom primeru biće prikazan način kreiranja i korišćenja HyperText sistema.

2) Autorski sistem - jedan konkretan autorski sistem je kreiran i on pokazuje način na koji se može kreirati niz sličnih sistema sa zajedničkom polaznom osnovom.

3) (Kvazi-)Inteligentni tutorski sistem - kao treći primer grubo je prikazana mogućnost i date su globalne smernice, kako da se izloženi formalizam iskoristi za formiranje inteligentnih sistema za reprezentaciju i prezentaciju znanja.

5.1 Jedan HyperText sistem

Svaki HyperText sistem ima tri ključne celine:

1) Bazu podataka - u kojoj se uskladištavaju informacije po logičnim celinama.

2) Šemu veza - izmedju pojedinih elemenata baze.

3) Pogodan mehanizam za vizuelnu prezentaciju informacija - reprezentovanih u bazi i realizaciju kretanja iz jednog u drugi element baze.

Korišćenjem predložene opšte strukture podataka i jezika za njeno popunjavanje moguće je podržati i realizovati sva tri elementa tj. celine HyperText-a sa četiri tipa informacija (tekstualna, grafička, animaciona i zvučna).

5.1.1 Osnovna struktura podataka i šema veza HyperText-a

Domen reprezentacije koji se obradjuje HyperText-om se sastoji od velike količine informacija. Da bi reprezentacija bila lakša i jednostavnija najpogodnije je kompletnu količinu informacija razdeliti u manje semantičke celine - Teme.

Globalna struktura HyperText-a

Globalna struktura HyperText-a je predloženi multidigraf. Čvorovi multidigrafa su teme domena, a veze izmedju čvorova su putevi kroz koje se vrši kretanje iz čvora u čvor u toku prezentacije tj. korišćenja HyperText-a.

Svi čvorovi HyperText-a se mogu podeliti u tri osnovne grupe.

- **početni čvorovi** - su oni iz kojih započinje korišćenja HyperText-a.

- **završni čvorovi** - su oni čvorovi u kojima se završava prezentacija, tj. dolazak u završni čvor indicira da je predjen jedan 'smisleni' put kroz multidigraf i da je završena jedna 'smislena' prezentacija.

- **unutrašnji čvorovi** - su čvorovi kroz koje se prolazi na relaciji od početnog do završnog čvora.

Postoji najčešće više različitih načina i puteva da se dodje iz jednog uočenog početnog čvora do odredjenog završnog čvora. Konkretan put prezentacije zavisi od korisnika HyperText-a, njegovih odluka i izbora daljeg toka prezentacije u svakom čvoru u koji je dospeo.

Pogledajmo na primeru jedan multidigraf koji predstavlja bazu jednog HyperText-a. Svi čvorovi tog multidigrafa su potencijalno završni čvorovi, ali postoji nekoliko čvorova u koje kada se dospe ne može se vršiti nikakvo kretanje u neke druge čvorove. Oni su obavezno završni (u primeru su to čvorovi 6, 7, 11, 13) jer se u njima vrši 'nasilni' prekid prezentacije.

Par čvorova je izabrano za početne (u primeru su to čvorovi 1, 2 i 5). Prezentacija tj. korišćenje HyperText-a se mora obavezno započeti u nekom od njih.

Takodje, ako korisnik želi da se prepusti sistemu, tj. da ne odlučuje o toku prezentacije, nego da bude vodjen sistemom, postoje dozvoljeni putevi koji zavise (u najjednostavnijem slučaju) od izbora početnog čvora.

Inače, mogla bi se ugraditi komponenta slučajnosti korišćenjem, recimo, generatora slučajnih brojeva, tako da dozvoljeni putevi ne budu uvek isti, već da u jednom čvoru postoji nekoliko različitih susednih čvorova u kojima je moguće nastaviti prezentaciju.

Medjutim, predpostavimo da iz jednog čvora postoji tačno jedan susedni čvor u kom se može nastaviti prezentacija.

Dozvoljeni putevi u ovom slučaju bi mogli biti:

1) **Za početni čvor 1:**

1 (računarstvo i programiranje) ->
3 (podela softvera) ->
5 (programski jezici) ->
4 (klasični programski jezici) ->
15 (jezici veštačke inteligencije) ->
11 (objektno orijentisani programski jezici).

2) **Za početni čvor 5:**

5 (programski jezici) ->
8 (apstraktni tipovi podataka) ->
4 (klasični programski jezici) ->
10 (funkcionalni programski jezici) ->
15 (jezici veštačke inteligencije) ->
9 (logički programski jezici) ->
11 (objektno orijentisani programski jezici).

3) **Za početni čvor 2:**

2 (hardver softver) ->
14 (delovi hardvera) ->
12 (diskovi) ->
13 (monitori).

Struktura čvora HyperText-a

Struktura čvora HyperText-a je klasa koja nasledjuje osnovnu klasu za reprezentaciju informacija tj. klasu *Info*. Svaki čvor *HTTopic* je ili početni ili obavezno završni ili unutrašnji.

Medjutim, svaki od njih je i potencijalno završni jer u skupu ključnih reči sadrži kao mogućnost za izbor i konstantu **END** koja označava kraj prezentacije.

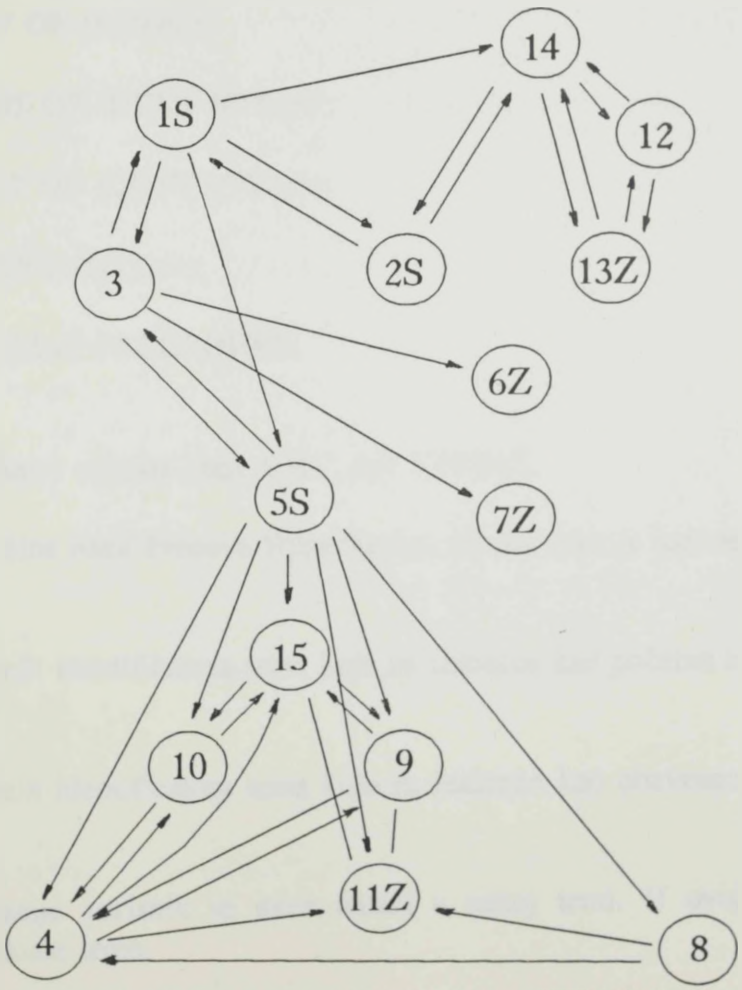
```
HTTopic = CLASS {
                    ident :     STRING;
                    CLASS Info EXCL ident;
                    defaHT :    NUMBER;
                    paHT :      LIST OF IDENT HTTopic;
                    ansHT :     PUBLIC PROCEDURE;
                    typeHT :    (START, OBLEND, INNER);
                }
```

ident - je vezana promenljiva u kojoj se nalazi jedinstveno ime objekta tog tipa.

defaHT - odredjuje redni broj ključne reči koja predstavlja jedinstveni identifikator čvora u kom se nastavlja prezentacija u slučaju dozvoljenih puteva tj. u slučaju kada učesnik prezentacije nije odabrao naredni čvor u kom će se nastaviti prezentacija.

paHT - lista jedinstvenih identifikatora tema. Svakoj ključnoj reči reprezentovanoj listom, odgovara po jedna tema na kojoj će se nastaviti prezentacija ako je izabrana ta ključna reč. Svakoj ključnoj reči iz liste ključnih reči odgovara ime teme koje je, u listi imena tema, na istoj poziciji kao i ključna reč.

ansHT - je globalna procedura koja ima zadatak da prihvati odgovor učesnika prezentacije, tj.



Sl 5.1. Globalna struktura multidigrafa jednog HYPERTEXT-a o računarima i programiranju.

da identifikuje objekat koji odgovara ključnoj reči koju je izabrao, ili da opredeli dozvoljenu ključnu reč.

typeHT - opredeljuje tip teme, početna (tipa **START**), obavezno završna (tipa **OBLEND**) i unutrašnja (tipa **INNER**).

Struktura HyperText-a

HyperText se sastoji od liste tema oblika *HTTopic*, ali mora da poseduje i niz drugih specifičnih elemenata. Struktura HyperText-a je klasa sledećeg oblika.

```
GHT = CLASS {
                ident : STRING;

                topGHT : LIST OF HTTopic;

                startGHT : LIST OF IDENT HTTopic;

                stopGHT : LIST OF IDENT HTTopic;

                currGHT : IDENT HTTopic;

                interGHT : PUBLIC PROCEDURE;
            }
```

ident - je jedinstveni identifikator objekta klase GHT, tipa STRING.

topGHT - je lista tema koje čine bazu čvorova HyperText-a. Svaka tema je jednog istog tipa, tipa klase *HTTopic*.

startGHT - je lista jedinstvenih identifikatora tema koje su izabrane kao početne u procesu prezentacije.

stopGHT - je lista jedinstvenih identifikatora tema koje su izabrane kao obavezno završne u procesu prezentacije.

currGHT - u toku prezentacije korisnik se uvek nalazi u nekoj temi. U ovoj promenljivoj se čuva identifikator tekuće teme.

interGHT - je procedura koja vodi i komlpetno podržava početak, čitav tok i završetak prezentacije. Ova procedura se aktivira na početku korišćenja HyperText-a, nudi korisniku izbor početnog čvora, pokreće proceduru za prikaz svih informacija u čvoru, i

proceduru za prihvatanje odgovora tj. izbor korisnika, prelazak u novi čvor i tako dalje, do završetka prezentacije.

Veze medju čvorovima HyperText-a.

Čvorovi HyperText-a reprezentovani klasom *HTTopic* su tipiziranu čvorovi jer poseduju jedinstveno ime tj. identifikator koji na jednisntven nači opredeljuje svaki čvor.

Veze medju čvorovima HyperText-a ostvarene su pomoću liste paHT, koja za svaku ključnu reč pridruženu toj temi, odredjuje temu u kojoj će se nastaviti prezentacija, ako je izabrana ta ključna reč.

Skup svih tema sa njihovim listama paHT opredeljuje čitav multidigraf tj. čvorove HyperText-a i sve veze medju njima.

5.1.2 Upravljanje i tok prezentacije u HyperText-u

Kada je struktura HyperText-a definisana i svi čvorovi popunjeni potrebnim sadržajima i uspostavljene veze medju njima, tada je multidigraf na raspolaganju za proces prezentacije tj. korišćenja.

Proces prezentacije započinje izborom nekog od početnih čvorova i dalje se nastavlja u zavisnosti od izbora novog pojma (tj. ključne reči) ili prepuštanja vodjenju kroz HyperText dopustivim putem.

Procedura interGHT, ima zadatak da prikaže sve početne teme HyperText-a, da prepusti korisniku izbor jedne kao početne, i da inicira proces prezentacije pozivajući ostale procedure. Grubi algoritam ove javne procedure, koja podržava celokupan rad i korišćenje HyperText-a je dat u nastavku.

```
(* Neka je HyperText, formirani objekat čiji je jedinstveni identifikator HTRP, i taj
identifikator se prosledjuje proceduri kroz argument HyTx *)

PROCEDURE interGHT(HyTx : GHT);

BEGIN
    (* prikaz svih početnih tema *)
    (* u programsku promenljivu memst se postavlja identifikator početne teme *)
    First(&HyTx.startGHT, memst);

    (* sve dok nije kraj liste identifikatora početnih tema *)
    WHILE NOT EndOfList(&HyTx.startGHT) DO
        WriteStr(memst);                    (* prikaži identifikator tekuće teme *)
        WriteLn;
```

```
        Next($HyTx.startGHT,memst);      (* uzmi sledeći identifikator teme *)
END;

ReadAns(&HyTx.currGHT);                  (* izabranu početnu temu učitaj u vezanu
                                            promenljivu currGHT *)
Choice = EMPTY;
(* započinjanje prezentacije *)

(* sve dok se nije došlo u obavezno završnu temu i nije izabran kraj prezentacije *)
WHILE &HyTx.currGHT.typeHT < > OBLEND AND Choice < > END DO

        &HyTx.currGHT.interAll;          (* prikaz svih informacija sačuvanih u
                                            toj temi *)
        &HyTx.currGHT.ansHT;             (* očekivanje i učitavanje odgovora u
                                            promenljivu Choice *)
        &HyTx.currGHT = Choice;          (* odgovor tj. ime sledeće teme će se
                                            učitati u promenljivu Choice *)
END;

(* kraj prezentacije *)
END.
```

5.1.3 Primena realizovanog HyperText-a

Slika 5.1. iz poglavlja 5.1.1. poslužiće kao osnova tj. baza čvorova u multidigrafu jednog konkretnog HyperText sistema. To je primer jednog jednostavnog HyperText-a koji sadrži osnovne informacije iz oblasti računarstva, nekih hardverskih elemenata i nekih podela softvera.

Globalna struktura HyperText-a je kao na slici 2. Znači baza čvorova se sastoji od 15 čvorova. U nastavku su dati brojevi čvorova, njihov tip i lista ključnih reči.

- 1 početni čvor : **računarstvo i programiranje**;

ključne reči : hardver softver, podela softvera, delovi hardvera, programski jezici, END.

- 2 početni čvor : **hardver i softver**;

ključne reči : delovi hardvera, podela softvera, računarstvo i programiranje, END.

- 3 unutrašnji čvor : **podela softvera**;

ključne reči : programski jezici, sistemi za unakrsno izračunavanje, tekst editori, računarstvo i programiranje, END.

- 4 unutrašnji čvor : **klasični programski jezici;**

ključne reči : programski jezici, objektno orijentisani programski jezici, jezici veštačke inteligencije, funkcionalni programski jezici, logički programski jezici, END.

- 5 početni čvor : **programski jezici;**

ključne reči : podela softvera, klasični programski jezici, funkcionalni programski jezici, logički programski jezici, objektno orijentisani programski jezici, jezici veštačke inteligencije, apstraktni tipovi podataka, END.

- 6 završni čvor : **sistemi za unakrsno izračunavanje;**

ključne reči : .

- 7 završni čvor : **tekst editori;**

ključne reči : .

- 8 unutrašnji čvor : **apstraktni tipovi podataka;**

ključne reči : klasični programski jezici, objektno orijentisani programski jezici, END.

- 9 unutrašnji čvor : **logički programski jezici;**

ključne reči : jezici veštačke inteligencije, klasični programski jezici, objektno orijentisani programski jezici, END.

- 10 unutrašnji čvor : **funkcionalni programski jezici;**

ključne reči : jezici veštačke inteligencije, klasični programski jezici, objektno orijentisani jezici, END.

- 11 završni čvor : **objektno orijentisani jezici;**

ključne reči : .

- 12 unutrašnju čvor : **diskovi;**

ključne reči : delovi hardvera, monitori, END.

- 13 završni čvor : **monotori;**

ključne reči : delovi hardvera, diskovi, END.

- 14 unutrašnji čvor : **delovi hardvera;**

ključne reči : diskovi, monitori, hardver softver, END.

- 15 unutrašnji čvor : **jezici veštačke inteligencije;**

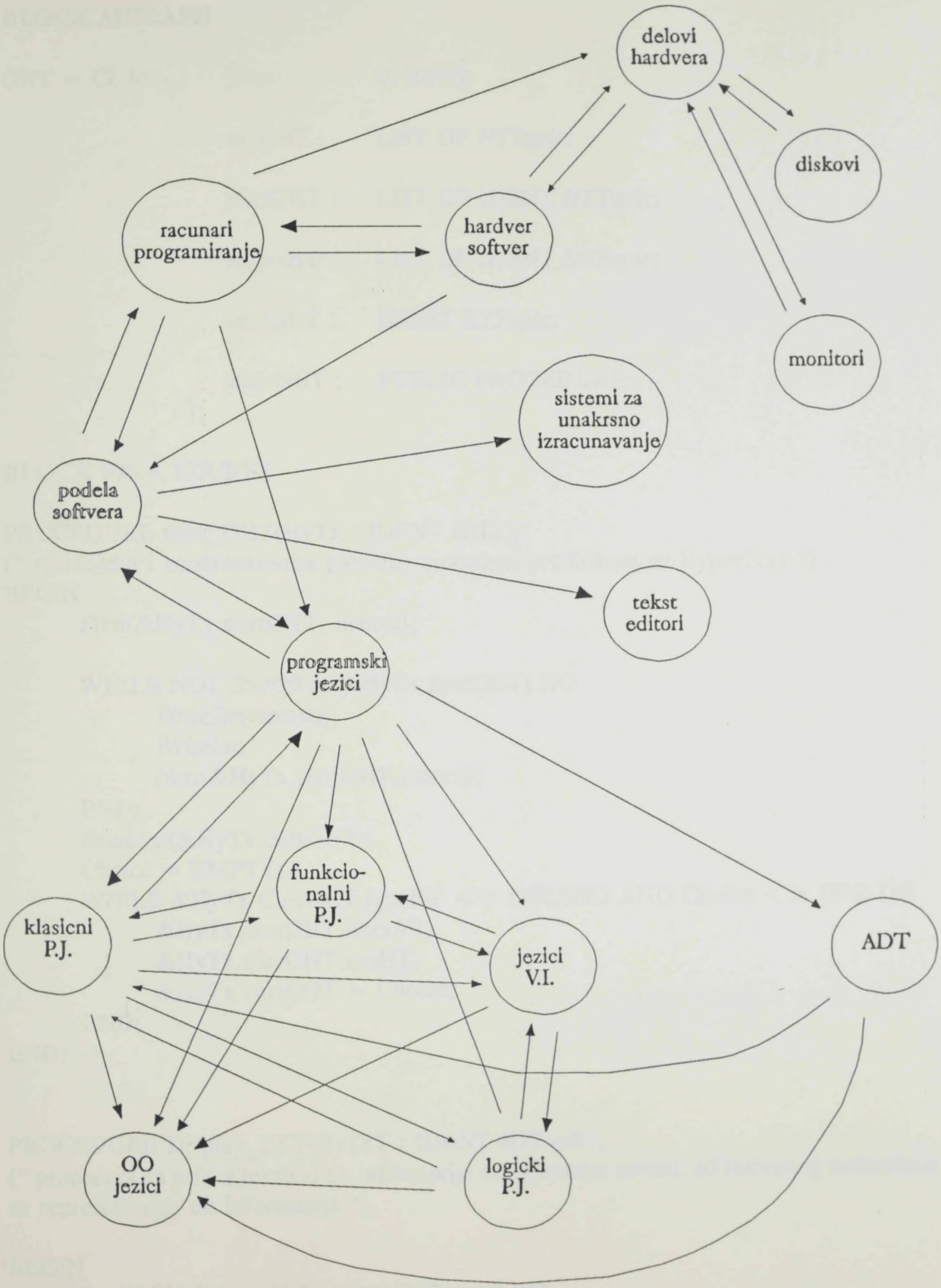
ključne reči : funkcionalni programski jezici, logički programski jezici, objektno orijentisani programski jezici, END.

Da bi se smanjio obim primera uzećemo da u čvorovima pretežno figurišu tekstualne informacije, sa sporadičnim grafičkim prikazima. Medjutim, ova pretpostavka ne umanjuje njegovu opštost, jer bi se i ostali oblici informacija uključili u teme na sličan način.

Program u jeziku **Less** za formiranje i popunjavanje strukture ovog HyperText-a je dat u nastavku.

Sekvencijalni način popunjavanja HyperText-a

```
BLOCK CLASSES

Retype(Text,interte,PUBLIC PROCEDURE);
Retype(Picture,interpic,PUBLIC PROCEDURE);
Retype(Animation,interan,PUBLIC PROCEDURE);
Retype(Sound,interso,PUBLIC PROCEDURE);

HTTopic = CLASS {
                    ident :     STRING;

                    CLASS Info EXCL ident;

                    defaHT :    NUMBER;

                    paHT :      LIST OF IDENT HTTopic;

                    ansHT :     PUBLIC PROCEDURE;

                    typeHT :    (START, OBLEND, INNER);
                };
```



Sl 5.2 Struktura HYPERTEXT-a o računarstvu i programiranju

BLOCK MGRAPH

```
GHT = CLASS {   ident :      STRING;
                topGHT :     LIST OF HTTopic;
                startGHT :   LIST OF IDENT HTTopic;
                stopGHT :    LIST OF IDENT HTTopic;
                currGHT :    IDENT HTTopic;
                interGHT :   PUBLIC PROCEDURE;
              };
```

BLOCK PROCEDURES

```
PROCEDURE inter_GHT(HyTx : IDENT GHT);
(* razradjena i konkretizovana globalna procedura predložena za HyperText *)
BEGIN
        First(&HyTx.startGHT, memst);

        WHILE NOT EndOfList(&HyTx.startGHT) DO
                WriteStr(memst);
                WriteLn;
                Next(&HyTx.startGHT,memst);
        END;
        ReadAns(&HyTx.currGHT);
        Choice = EMPTY;
        WHILE &HyTx.CurrGHT.typeHT < > OBLEND AND Choice < > END DO
                &HyTx.currGHT.interAll;
                &HyTx.currGHT.ansHT;
                &HyTx.currGHT = Choice;
        END;
END;


PROCEDURE Display_TXT(HyTxT : IDENT HTTopic);
(* procedura za prikaz tekstualnih informacija će uglavnom zavisiti od izabranog mehanizma
za reprezentaciju tih informacija *)

BEGIN
        WHILE NOT EndOfList(&HyTxT.texts) DO
                (* uzmi naredni text iz liste *)
                (* formatizuj ga za prikaz *)
```

```
            (* prikaži tekst *)
            (* zadrži prikaz neko vreme na ekranu da bi ga korisnik analizirao *)
        END;
END;

PROCEDURE Display_ALL(HyTx : IDENT GHT);
(* procedura za sinhronizovani prikaz svih informacija iz čvora *)
BEGIN
        (* prikazuje tekstualne informacije koje su memorisane u tekućem čvoru tj. temi (čiji
        je jedinstveni identifikator vrednost promenljive currGHT) tekućeg HyperText-a (čiji
        je jedinstveni identifikator vrednost promenljive HyTx) *)

        Display_TXT(&HyTx.currGHT);

        (* pristupanje grafičkim informacijama tekućeg čvora HyperText-a ako ih ima i
        njihova prezentacija *)

        First(&HyTx.currGHT.pic,P);

        WHILE NOT EndOfList(P) DO
            Display_PIC(P);                 (* prezentacija grafičkih prikaza na
                                               ekranu *)
            (* zadržavanje grafičkog prikaza neko vreme na ekranu *)
            Next(&HyTx.currGHT.pic,P);      (* uzima sledeći grafički prikaz *)
        END;

        (* pristupanje simulacijama tekućeg čvora HyperText-a ako ih ima i njihova
        prezentacija *)

        First(&HyTx.currGHT.ani,A);

        WHILE NOT EndOfList(A) DO
            Display_ANI(A);                 (* prezentacija animacije na ekranu *)
            (* zadržavanje animacije neko vreme na ekranu *)
            Next(&HyTx.currGHT.Ani,A)       (* uzima sledeću animaciju *)
        END;

        (* pristupanje zvučnim informacijama tekućeg čvora HyperText-a ako ih ima i
        njihova prezentacija *)

        First(&HyTx.currGHT.sou,S);

        WHILE NOT EndOfList(S) DO
            Display_SOU(S);                 (* prezentacija zvučnih informacija *)
            (* vrši eventualno ponavljanje prezentirane informacije *)
```

```
            Next(&HyTx.currGHT.Sou,S)        (* uzima sledeću zvučnu informaciju *)
      END;
END;

PROCEDURE Accept_Ans;
(* procedura za prihvatanje odgovora tj. izbora ključne reči *)
BEGIN
      WHILE True DO BEGIN
            ReadAns(Choice);
            IF učitan_string
            THEN
                  BEGIN
                        (* vrši eventualno kontrolu ispravnosti učitanog string i
                        proverava da li postoji takav objekat u listi ključnih reči *)
                        rbr = redni_broj_ključne_reči;
                        Choice = paHT[rbr];
                        IF Ispravan_odgovor
                              RETURN
                        END;
                  END

            ELSE
                  BEGIN
                        (* uzima se dopustiva ključna reč *)
                        Choice = paHT[defaHT];
                  END
      END;
END;

START

(* formiranje HyperText-a sa jedinstvenim identifikatorom OKR *)
AssignOb(GHT, 'OKR');    (* automatski se vezanoj promenljivoj ident u formiranom
                         objektu dodeljuje ovaj jedinstveni identifikator *)

(* popunjavanje HyperText-a odgovarajućim sadržajima *)
{
      topGHT :    AssignList(TopicsOKR);

      startGHT :  AssignMul(IDENT HTTopic, startOKR,('računarstvo i
                  programiranje','hardver softver','programski jezici'));

      stopGHT :   AssignMul(IDENT HTTopic, stopOKR, ('sistemi za unakrsno
                  izračunavanje','tekst editori','objektno orijentisani programski
                  jezici','monitori'));
```

```
        currGHT :    EMPTY;

        interGHT :   Interpret(inter_GHT);
};

(* formiranje prvog čvora tj. objekta HyperText-a *)
AssignOb(HTTopic,'računarstvo i programiranje')

(* popunjavanje tog čvora odgovarajućim sadržajima *)
{
        key :        AssignMul(STRING,,('hardver softver','podela softvera','delovi
                     softvera','programski jezici','END'));
        tex : {
                texts :      AssignTPAS(Text ... EndText);
                interte :    Interpret(Display_TXT);
                };

        interAll :   Interpret(Display_ALL);

        defaHT :     AssNo(2);

        paHT :       AssignMul(IDENT HTTopic,,('hardver softver','podela
                     softvera','delovi softvera','programski jezici','END'));

        ansHT :      Interpret(Accept_ans);

        typeHT :     START;
};

(* dodavanje prvog popunjenog objekta u mrežu multidigrafa tj. u listu topika na poslednju
poziciju *)
AddLast($TopicsOKR,'računarstvo i programiranje');

(* formiranje i popunjavanje drugog objekta *)
AssignOb(HTTopic,'hardver i softver');
(* popunjavanje objekta *)

(* dodavanje drugog popunjenog objekta u mrežu multidigrafa tj. u listu topika na poslednju
poziciju *)
AddLast($TopicsOKR,'hardver i softver');

...
(* formiranje i popunjavanje ostalih objekata tj. čvorova *)
...
```

STOP

Ciklični način popunjavanja HyperText-a

Nadalje će biti ilustrovan način popunjavanja navedenog HyperText-a korišćenjem datoteka i ciklusa raspoloživih za gradjene tipove.

(* prva tri dela programa su ista *)

START

(* formiranje HyperText-a sa jedinstvenim identifikatorom OKR *)
AssignOb(GHT, 'OKR');

(* popunjavanje HyperText-a odgovarajućim sadržajima *)
{
topGHT : *AssignList*(TopicsOKR);

startGHT : *AssignMul*(IDENT *HTTopic*, startOKR,('računarstvo i programiranje',hardver softver','programski jezici'));

stopGHT : *AssignMul*(IDENT *HTTopic*, stopOKR, ('sistemi za unakrsno izračunavanje','tekst editori','objektno orijentisani programski jezici','monitori'));

currGHT : EMPTY;

interGHT : *Interpret*(inter_GHT);
};

(* otvaranje datoteka u kojima se nalaze odgovarajuće vrednosti za objekte tipa *HTTopic* *)

(* otvaranje datoteke u kojoj se nalaze jedinstveni identifikatori objekata *)
OpenFile(DatIdent,STRING,$$);

(* otvaranje datoteke u kojoj se nalaze ključne reči koje odgovaraju objektima *)
OpenFile(DatKey,LIST OF STRING,$$);

(* otvaranje datoteke u kojoj se nalaze reprezentovane tekstualne informacije objekata *)
OpenFile(DatText,LIST OF TEXT,$$);

(* otvaranje datoteke u kojoj se nalaze imena procedura za prezentaciju tekstualnih informacija memorisanih u objektima *)

OpenFile(DatPTex,STRING,$$);

(* slične datoteke se otvaraju za informacije tipa PICTURE, ANIMATION i SOUND a njihova imena su DatPic, DatAni, DatSou za reprezentovane informacije i DatPPic, DatPAni, DatPSou za imena procedura za interpretaciju *)

(* otvaranje datoteke u kojoj se nalaze imena procedura za prezentaciju svih pojavnih oblika informacija *)
OpenFile(DatPPAll,STRING,$$);

(* otvaranje datoteke u kojoj se nalaze brojevi koji označavaju redni broj ključne reči koja čini dozvoljeni put u objektima *)
OpenFile(DatDefa,NUMBER,$$);

(* otvaranje datoteke u kojoj se nalazi lista identifikatora objekata koji odgovaraju ključnim rečima *)
OpenFile(DatPat,LIST OF STRING,$$);

(* otvaranje datoteke u kojoj se nalaze imena procedura koje prihvataju odgovor u procesu prezentacije *)
OpenFile(DatAns,STRING,$$);

(* otvaranje datoteke u kojoj se nalaze tipovi objekata *)
OpenFile(DatTyp,STRING,$$);

LoopFile

```
(* formiranje prvog čvora tj. objekta HyperText-a *)
AssignOb(HTTopic,File(DatIdent,PomIdent));

(* popunjavanje tog čvora odgovarajućim sadržajima *)
{
    key :       File(DatKey);

    tex : {
        texts :     File(DatText);
        interte :   File(DatPTex);
        };

    interAll :  File(DatPAll);

    defaHT :    File(DatDefa);

    paHT :      File(DatPat)
```

```
        ansHT :     File(DatAns);

        typeHT :    File(DatTyp);
    };

    (* dodavanje popunjenog objekta u mrežu multidigrafa tj. u listu topika na poslednju
    poziciju *)
    AddLast($TopicsOKR,&PomIdent);
EndLoop;
STOP
```

U izloženom primeru prikazan je globalni program za popunjavanje strukture multidigrafa prikazanog na slici 5.2. Da bi se bolje shvatio rad programa potrebno je prikazati i sadržaj datoteka sa kojima se u primeru rukuje.

DatIdent - jedinstveni identifikatori objekata

```
'računarstvo i programiranje'   $$   (* identifikator prvog objekta *)
'hardver siftver'               $$   (* identifikator drugog objekta *)
...                             $$   (* identifikatori preostalih objektat *)
```

DatKey - ključne reči objekata

```
'hardver softver'                    (* ključne reči prvog objekta *)
'podela siftvera'
'delovi hardvera'
'programski jezici'
'END'                           $$

'delovi hardvera'                    (* ključne reči drugog objekta *)
'podela softvera'
'računarstvo i programiranje'
'END'                           $$
...                             $$   (* ključne reči ostalih objekata *)
```

DatText - tekstovi objekata

```
Text ... EndText                     (* tekstovi prvog objekta *)
...
Text ... EndText                $$

Text ... EndText                     (* tekstovi drugog objekta *)
```

...
Text ... EndText $$

... $$ (* tekstovi ostalih objekata *)

Ako su tekstovi reprezentovani u nekim drugim datotekama (pojedinačno) tada se u ovoj datoteci navode imena tih datoteka, razdvojena delimiterima. Taj slučaj se jednostavno prepoznaje usled odsustva ključnih reči **Text** i **EndText**. Ista napomena važi za datoteke u kojima su predstavljeni ostali oblici informacija.

DatPTex - procedure za prezentaciju teksta

PrikazTXT (* U slučaju da se ime javne procedure za više objekata čita iz datoteke onda se podrazumeva da je u njoj samo jedno ime. Ime javne datoteke se pridružuje samo prvoformiranom objektu i važi za sve kasnije objekte. Inače se ime procedure može dodeliti i nekoj programskoj promenljivoj i onda nema potrebe da se čita iz datoteke. *)

(* slična je i konfiguracija datoteka za prikaz slika, animacija i zvuka *)

DatPPAll - procedure za prezentaciju svih informacija

PrikazALL (* ista napomena važi kao i za ostale javne procedure *)

DatDefa - dozvoljeni putevi u objektima

2 $$ (* pozicija ključne reči koja opredeljuje dozvoljeni put za prvi objekat - to je ključna reč 'podela softvera' *)

1 $$ (* pozicija ključne reči koja opredeljuje dozvoljeni put za drugi objekat - to je ključna reč 'delovi hardvera' *)

... $$ (* specifikacija dozvoljenih puteva za ostale objekte *)

DatPat - identifikatori dozvoljenih puteva

```
'hardver_softver'                    (* identifikatori koji za svaku ključnu reč prvog
                                        objekta opredeljuju odgovarajući objekat *)
'podela_softvera'
'delovi_hardvera'
'programski_jezici'          $$

'delovi_hardvera'                    (* odgovarajući identifikatori za drugi objekat *)
'podela_softvera'
'računarstvo_i_programiranje'  $$
...                            $$    (* odgovarajući identifikatori za ostale objekte
                                        *)
```

Kako identifikator treba da bude kompaktno ime, praznine u ključnim rečima se zamenjuju donjom crtom.

U ovom primeru identifikatori objekata su identični ključnim rečima mada u opštem slučaju ne moraju biti. Naročiti, ako su ključne reči duže, zbog lakšeg manipulisanja identifikatori odgovarajućih objekata mogu imati druge oznake.

DatAns - procedure za prihvatanje odgovora

```
AcceptAns                    (* ista napomena važi kao i za ostale javne procedure *)
```

DatTyp - tipovi čvorova

```
START                  $$    (* tip prvog čvora - početni *)
START                  $$    (* tip drugog čvora - početni *)
INNER                  $$    (* tip trećeg čvora - unutrašnji *)
...                    $$    (* tipovi ostalih čvorova *)
```

U ovom primeru vršeno je direktno dodeljivanje vrednosti čitanjem iz datoteka, bez medjukoraka tj. dodele vrednosti programskim promenljivim, a tek onda promenljivim u objektima. Obzirom da se programske promenljive ne bi višestruko koristile (osim PomIdent) nije bilo potrebe za njihovim uvodjenjem.

5.2 Reprezentacija i korišćenje informacija u obrazovanju - autorski sistem

Najnovija faza u razvoju sistema za obradu podataka i računarskih nauka, karakteriše se kao faza formalizacije (reprezentacije) i primene znanja. U literaturi, naročito u poslednjih

desetak godina, baze znanja su jedan od najpopularnijih, ali u isto vreme, i najnedefinisanijih pojmova [Bobrow, 1976], [Feigenbaum, 1977], [Freudlich, 1990], [Rubaškin, 1989]. Kao posledica najnovijih istraživanja, u toj oblasti, javlja se nova disciplina - inženjering znanja [Osuga, 1989, 1990].

U isto vreme razvoj sve moćnijih računara omogućava implementaciju principa veštačke inteligencije, kao nove tehnologije u obradi podataka. Primena tih principa uslovljava razvoj različitih vrsta sistema za formalan opis znanja, njegovo prikupljanje i korišćenje [Carando, 1989], [Kary, 1986].

Medjutim, postoji još mnogo problema koje treba razrešiti. Najznačajniji su sledeći.

- Pronalaženje adekvatnih metoda i struktura podataka za reprezentaciju opštih znanja - frejmovi, semantičke mreže, predikatska logika, objekti, ili neki sasvim novi formalizam.

- Omogućavanje obrade znanja, sačuvanog u bazi kojom se opisuje neki problem. Raspoložive tehnike variraju od čisto manualnih do raznih pokušaja komprimovanja teksta i izdvajanja ključnih znanja.

- Razvijanje pogodnih mehanizama zaključivanja. Ovi mehanizmi treba da omoguće izvodjenje inteligentnih i preciznih zaključaka, iz raspoložive baze znanja i nekih dodatnih informacija.

- Razrešenje problema generisanja novih znanja iz postojeće baze, i njihovog uključivanja u primarnu bazu.

Rešenja navedenih problema treba da uključuju inteligentne funkcije, treba da omoguće obradu znanja sa semantičkog aspekta i pri tome treba da dostignu zadovoljavajući stepen pouzdanosti u radu i zaključivanju. Medjutim, rešavanje tih problema nije jednostavno.

U poslednjih desetak godina, javio se veliki broj radova, koji se bave ovom problematikom. Niz autora je pokušavao da implementira kompletne i kompleksne sisteme za reprezentaciju, prikupljanje i korišćenje znanja [Grossman, 1987], [Maurer, 1987], [Wahlster, 1990]. Rešenja su se obično kretala izmedju kompleksnih, elegantnih i ugodnih za rad, softverskih proizvoda sa malo ili nimalo inteligentnih funkcija, i poluinteligentnih sistema.

Medjutim, kako razvijanje inteligentnih baza znanja i sistema za njihovo korišćenje predstavlja još uvek posao budućnosti, prve i polazne aktivnosti su razvijanje sistema za reprezentaciju i prezentaciju informacija.

Jedna od mogućih primena predloženog okvir za reprezentaciju informacija i odgovarajućeg jezika za formiranje baze tema, su i procesi obrazovanja.

5.2.1 Lekcija kao multidigraf

Svaki obrazovni proces predstavlja skup smislenih celina tj. **lekcija - nastavnih sekvenci**, koje treba preneti većem broju ljudi.

U osnovi, proces obrazovanja (učenja) uključuje dva tipa učesnika:

- **kreatore nastavne sekvence - profesore,**

- **korisnik nastavne sekvence - učenike.**

Kreatori nastavne sekvence moraju da obradjuju svoje (i/ili tudje) znanja o nekom objektu, pojavi i sl., i da uspostave odredjene veštine i znanja koja učenici treba da usvoje u procesu obrazovanja. Pri tome jedan od ključnih elemenata je izbor najpogodnijeg načina reprezentacije i prezentacije pripremljenih informacija.

Konačan rezultat aktivnosti profesora je nastavna sekvenca (lekcija) koja treba da bude prezentirana učenicima.

Učenici moraju (treba) da prihvate neka znanja i veštine, koristeći lekcije koje je pripremio profesor.

U opštem slučaju, svaka lekcija se posmatra kao kompleksan skup članaka.

Članci su manji, relativno nezavisni delovi lekcije, koji zajedno grade semantičku celinu. Članak sadrži optimalnu količinu informacija (znanja) koju učenik treba da prihvati u nekom predvidjenom periodu vremena.

Proces učenja predstavlja prelazak iz jednog članka u drugi članak nastavne sekvence na neki determinisani način i prihvatanje informacija (znanja) reprezentovanih u svakom članku. Učeniku se u svakom članku prezentiraju memorisane informacije i on treba da ih prihvati. Da bi se utvrdio stepen prihvaćenih znanja i savladanosti lekcije u svakoj fazi procesa učenja, učeniku mogu biti postavljeni zadaci i/ili pitanja koje treba da reši i/ili na koja treba da odgovori.

Dalji tok učenja zavisi od rešenja tj. odgovora učenika. Odluke o daljem toku učenja obično donosi profesor, mada o njima može da odlučuje i sam učenik.

Ako prihvatimo opšti proces učenja, kako je to napred navedeno, tada je najpogodnija struktura za prikaz kompletne lekcije multidigraf, onakav kakav je uveden u poglavlju 3.

Članci su čvorovi lekcije, a putevi kroz lekciju grane grafa.

U svakoj lekciji postoji tri tipa čvorova: početni, završni i unutrašnji čvorovi.

Svaka lekcija sardži najmanje jedan početni i najmanje jedan završni čvor.

Učenik, proces učenja, započinje u nekom od početnih, a završava u nekom od završnih čvorova lekcije. Na putu od početnog do završnog čvora lekcije, učenik prolazi kroz unutrašnje čvorove, u kojima su na pogodan način reprezentovane informacije iz lekcije.

Različiti putevi kroz graf (lekcije) omogućavaju različite načine prihvatanja znanja: sa manje ili više ponavljanja, sa ili bez dodatnih objašnjenja, sa ili bez naprednih članaka itd.

Tako, na primer, u lekciji prikazanoj na Slici 3.1., v_1 i v_2 su početni čvorovi, a v_{15} i v_{16} su završni čvorovi. Izmedju njih su svi mogući putevi kroz lekciju, gde grane prikazane pravim linijama vode cilju tj., nekom od završnih čvorova lekcije, a grane prikazane krivim linijama vode u dodatna ili napredna objašnjenja.

Ako učenik poseduje predznanja o lekciji, on može da preskoči početna objašnjenja (iz čvora v_1) i da započne učenje iz čvora v_2. U zavisnosti od njegovih odgovora na postavljene zadatke, on se iz čvora v_2 može dalje kretati u čvor v_6 ili v_9, a iz njih dalje u zavisnosti od odgovora tj. od savladanih informacija iz čvora.

Struktura klasa opisana u poglavlju 3. i jeseik *Less*, predstavljaju dobru osnovu za korišćenje računara u nastavnom procesu, tj. za realizaciju sistema za reprezentaciju i prezentaciju informacija u obrazovanju.

U procesu prezentacije, nakon što se učeniku informacije iz članka prezentiraju, postavljaju mu se zadaci koje treba da reši. U zavisnosti od kvaliteta i ispravnosti rešenja postoje sledeće (logične) mogućnosti da se nastavi proces učenja u skladu sa prihvaćenom koncepcijom.

- Ako je učenik tačno odgovorio na postavljeni zadatak, a u tom članku ima još nerešavanih zadataka, učenik se može uputiti na sledeći zadatak istog članka.

- Ako je učenik tačno odgovorio na postavljeni zadatak, a to je poslednji zadatak vezan za taj članak, učenik se može uputiti na novi članak.

- Ako učenik nije tačno odgovorio na postavljeni zadatak, on se može uputiti na neko dodatno objašnjenje ili na neki već predjeni članak radi ponovne prezentacije informacija iz tog članka.

- Ako je učenik tačno odgovorio na postavljeni zadatak, a to je poslednji zadatak vezan za poslednji članak, može se prekinuti prezentacija tj. učenje.

Ovo su samo neka od prirodnih i uobičajenih varijanti za kretanje kroz lekciju, medjutim, u opštem slučaju kretanje zavisi od samog profesora i njegove kreativnosti i želja. Isto tako se može opredeliti da učenik, koji je suviše dobro odgovorio na postavljeni zadatak i pokazao veće znanje, 'preskoči' neke članke i brže završi učenje. Putevi kroz lekciju mogu biti različiti, neki delovi se mogu višestruko ponavljati, što zavisi od sposobnosti učenika i njegovih rešenja zadataka.

Čitavu strukturu, podelu lekcije u članke izbor i broj zadataka, veze medju čvorovima u multidigrafu odredjuje kreator lekcije tj. profesor.

Kada su čvorovi popunjeni sadržajem i uspostavljene veze medju njima lekcija je spremna za korišćenje, tj. mogu je koristiti učenici u procesu učenja.

Svaki članak multidigrafa može da čuva različite informacije, koje treba da budu prezentirane učesniku prezentacije.

On takodje, treba da uključi zadatke pomoću kojih se odlučuje o daljem toku prezentacije.

Dalji tok prezentacije može biti u nekoj relaciji sa rešenjem postavljenog zadatka.

Za reprezentaciju članka, kao osnovnog elementa mreže, koristi se klasa koja omogućava predstavljanje različitih tipova: tekstualni, grafički itd.

5.2.2.1 Struktura lekcije

Koristeći navedene konvencije, multidigraf lekcija se može prikazati kao klasa *Lesson*.

```
Lesson = CLASS {
                ident :      STRING;
                lsn :        LIST OF LTopic;
                start :      LIST OF IDENT LTopic;
                stop :       LIST OF IDENT LTopic;
                currTpc :    IDENT LTopic;
                currTsk :    IDENT LTask;
                interpr :    PUBLIC PROCEDURE;
                stud :       LIST OF Student;
        }
```

- **ident** - je kratko, asocijativno ime nastavne sekvence koje suštinski reprezentuje informacije sačuvane u njoj.

- **lsn** - je lista članaka (tj. *LTopic* objekata), koji čine nastavnu sekvencu. Čvorovi su medjusobno vezani u graf.

- **start** - je lista jedinstvenih identifikatora početnih članaka nastavne sekvence.

- **stop** - je lista jedinstvenih identifikatora završnih članaka nastavne sekvence.

- **currTpc** i **currTsk** - su pomoćne vezane promenljive nastavne sekvence koje pokazuju pozicije do kojih je učenik, u posmatranom trenutku, došao u procesu učenja. Oni uvek pokazuju na, respektivno, tekući članak (tj. *LTopic*) i tekući zadatak (tj. *LTask*) tekućeg članka, u procesu učenja.

- **interpr** - je procedura koja inicira i sve vreme podržava proces učenja tj. interpretaciju lekcije.

- **stud** - je lista objekata od kojih se svaki pridružuje jednom učeniku u procesu učenja. Može da se sastoji od dve grupe informacija:

1) Opšte informacije o učeniku.

2) Informacije statističke prirode o procesu učenja.

5.2.2.2 Struktura klase *Student*

Nastavnu sekvencu koju pripremi profesor, učenik koristi u procesu učenja.

Put kroz graf nastavne sekvence zavisi od predznanja učenika, opštih ili vezanih za samu nastavnu sekvencu. Ukoliko je predznanje učenika veće, i ukoliko je on sposobniji, to će relativno jednostavno, bez vraćanja i ponavljanja nekih delova puta kroz mrežu, bez dodatnih objašnjenja i slično, da savlada zadatu nastavnu sekvencu.

Ukoliko su mu predznanja manja, ako je manje sposoban i motivisan tada može da se desi da se po nekoliko puta vraća u isti čvor i ponavlja delove puta kroz nastavnu sekvencu. Naravno, sve to uzrokuje različitu uspešnost i dužinu savladavanja gradiva.

Različite statističke informacije, ako bi se memorisale, mogle bi kasnije da pomognu profesorima kod korigovanja nastavne sekvence, za naredne generacije učenika.

Stoga, da bi profesor stekao uvid u kvalitet kreirane nastavne sekvence, potrebno je uz svaku nastavnu sekvencu vezati informacije o toku uspešnosti učenja svakog učenika koji je koristio lekciju.

Jedna moguća struktura klase *Student* je data u nastavku. Predložena struktura je vrlo jednostavna jer ne čuva informacije o prethodnim saznanjima učenika. Medjutim, iz nje se mogu izvući različite statističke informacije koje mogu biti od pomoći da se revidira i popravi lekciju za naredna korišćenja.

Pitanja na koja većina učenika nije tačno odgovorila, mogu se smatrati preteškim, i obrnuto, pitanja na koja je većina učenika tačno odgovorila mogu se smatrati lakim. U skladu sa tim mogu se vršiti uskladjivanja pitanja i nastavne sekvence.

```
Student = CLASS {
                ident :      STRING;

                success :    LIST OF STopic;
```

```
}
```

- **ident** - je ime i prezime učenika ili neka njegova jedinstvena identifikacija.

- **success** - predstavlja listu objekata u kojima se nalaze identifikatori članaka i pitanja vezanih za te članke, a takodje sadrži i informacije o procesu učenja i njegovoj uspešnosti.

STopic je nova klasa koja omogućava praćenje toka učenja.

```
STopic = CLASS {
                idTopic :     STRING;

                solAns :      LIST OF T(NUMBER,LOGICAL);
              }
```

- **idTopic** - je jedinstveni identifikator objekta tj. članka kroz koji je učenik prošao u toku učenja.

- **solAns** - je lista parova. Svaki element liste je par čiji prvi element odgovara rednom broju zadatka vezanom za taj članak. Drugi element je logička vrednost koja kazuje da li je zadatak uspešno rešen ili ne. Ako je učenik na zadovoljavajući način rešio postavljeni zadatak onda se u odgovarajući element liste postavlja logička vrednost TRUE, inače se postavlja logička vrednost FALSE.

Objekti ovog tipa se popunjavaju u toku prezentacije. Kada učenik dospe u novi članak lekcije, formira se novi element liste success sa jedinstvenim identifikatorom članka. Kada se učeniku postavljaju zadaci iz tog članka, formira se za svaki zadatak odgovarajući par u listi **solAns**.

5.2.2.3 Struktura osnovne klase za reprezentaciju informacija *LInfo*

Klasa ***Info*** pruža mogućnost reprezentacije različitih vrsta informacija, koje treba da poseduje svaki članak lekcije. Većina klasa, koje se formiraju za podržavanje strukture lekcije, će uključivati strukturu klase ***Info*** i njen način za reprezentaciju različitih pojavnih oblika informacija.

Za slučaj autorskog sistema tj. lekcije iz klase *Info* zadržavaju se tri karakteristična i najčešće korišćena načina za reprezentaciju informacija tekst, slika i animacija,
i smeštaju se u strukturu klase *LInfo* koja služi za reprezentaciju osnovnih pojavnih oblika informacija u lekciji.

```
LInfo = CLASS {
                    CLASS Info EXCL key, sou ;

                }
```

- **ident** - je asocijativno ime za informacije koje se čuvaju u konkretnom objektu.

- **tex** - je vezana promenljiva tipa ugradjene klase *Text* i omogućava memorisanje tekstualnih informacija. Tekst se unosi u relativno jednostavnom formatu i može se u ovom slučaju pretpostaviti da je to reprezentacija koja uključuje specijalne simbole za formatiranje teksta: mala slova, velika slova, pojačani ispis, blinkovani ispis, ... (kao što je to opisano udelu 4.5.1.).

- **pic** - je lista grafičkih prikaza. Možemo predpostaviti da je izabrana reprezentacija uz pomoć grafičkih editora (opisana u delu 4.5.2.), tj. svaki grafički prikaz se nalazi u nekoj izvršnoj datoteci. Lista grafičkih prikaza je lista imena izvršnih datoteka u kojima se 'nalaze' ti prikazi.

- **ani** - je lista animacija (simulacija). Možemo predpostaviti da je izabrana reprezentacija uz pomoć grafičkih editora (opisana u delu 4.5.3), tj. svaka animacija se nalazi u nekoj izvršnoj datoteci. Lista animacija je lista imena izvršnih datoteka u kojima se 'nalaze' animacije.

- **interAll** - je javna procedura koja se koristi za prikaz svih informacija sadržanih u objektu ***LInfo***. Za svaki objekat ovog tipa ta proceduru je zajednička.

5.2.2.4 Struktura članka

Klasa *LTopic* treba da omogući reprezentaciju pojedinačnih semantičkih celina lekcije.
Cela struktura klase ***LTopic*** (koji služi za reprezentaciju informacija u članku lekcije) je prikazana na Slici 2. Struktura klase ***LTopic*** je data u nastavku.

```
LTopic = CLASS {
                    ident :         STRING;

                    CLASS LInfo EXCL ident;

                    query :         LIST OF LTask;
          }
```

LTopic nasledjuje klasu ***LInfo*** bez vezane promenljive ident. Osim toga sadrži još neke vezane promenljive.

- **ident** - je jedinstveni identifikator objekta.

- **query** - je lista objekata tipa *LTask*. Objekt *LTask* služi za reprezentaciju pitanja, zadataka ili vežbi koji se pridružuju članku. Vezane *LTask*-ove učenik treba da reši u procesu učenja, posle prikaza svih informacija reprezentovanih u tom članku.

(U daljem tekstu će se pojam **zadatak** odnositi na sve moguće tipove *LTask*-ova: pitanje, zadatak ili vežba)

Ako članak ne sadrži vezanu promenljivu **query** (tj. ona je postavljena na praznu vrednost), radi se o članku koji predstavlja dodatno objašnjenje. Dodatno objašnjenje može pomoći da se objasni ili skrene pažnja na neke druge opšte informacije koje mogu da pomognu učeniku da bolje shvati pojedine elemente lekcije.

5.2.2.5 Struktura zadatka

Za svaki članak nastavne sekvence vezuje se proizvoljan broj zadataka različitog tipa koje učenik treba da reši. Od načina i kvaliteta rešenja zavisi dalji tok prezentacije tj. procesa učenja.

Medjutim, sve odluke u sistemu donosi kreator nastavne sekvence tj. profesor u procesu kreiranja lekcije tj. u procesu njenog popunjavanja uz pomoću Less programa. Stoga se ovaj sistem može ubrojiti u sisteme sa slobodnom prezentacijom sa determinisanim izborom teme iz koje će se nastaviti prezentacija (ukratko izloženog u delu 3.1).

U ovom sistemu se opredeljujemo za strukturu zadataka koja je data klasom *LTask*.

```
LTask = CLASS {
                ident :        STRING;

                CLASS LInfo EXCL ident;

                aldSlvd :      LOGICAL;

                disp :         LOGICAL;

                dispLT :       PUBLIC PROCEDURE;

                {
                    CASE type : (Test, Prompt, Pair, Bool, Exec) OF

                    Prompt :
                    {
                        correct :      STRING;
                        answer :       STRING;
                        isCorr :       PRIVATE PROCEDURE;
```

```
            ifCorr :    CLASS Action;
            ifNot :     CLASS Action;
        };

        Test :
        {
            answer :    LIST OF CLASS Alter;
        };

        Pair :
        {
            answer1 :   LIST OF TEXT;
            answer2 :   LIST OF TEXT;
            true :      LIST OF NUMBER;
            isCorr :    PRIVATE PROCEDURE;
            ifCorr :    CLASS Action;
            ifNot :     CLASS Action;
        };

        Bool :
        {
            correct :   LOGICAL;
            answer :    LOGICAL;
            isCorr :    PRIVATE PROCEDURE;
            ifCorr :    CLASS Action;
            ifNot :     CLASS Action;
        }

        Exec :
        {
            exeFil :    STRING;
            next :      CLASS Action;
        }


    };
}
```

Klasa ***LTask*** nasledjuje klasu ***LInfo*** za reprezentaciju informacija o zadatku koji treba rešiti. Osim toga, sadrži još niz različitih fiksnih vezanih promenljivih.

- **ident** - je identifikator koji jedinstveno predeljuje svaki zadatak u lekciji.

- **aldSlvd** - sadrži istinitosnu vrednost **TRUE**, ako je zadatak već rešavan u nekom prethodnom prolasku kroz odgovarajući članak (kom pripada zadatak). U suprotnom, sadrži

istinitosnu vrednost **FALSE**. Ova informacija je korisna jer omogućava da se učeniku, kada se ponovo nadje u nekom već obradjivanom članku, ne postavljaju ona pitanja na koja je u prethodnom prolazu tačno odgovorio.

- **disp** - odredjuje da li će se nakon pogrešnog rešenje zadatka, prikazati tačan odgovor. Ako je promenljivoj pridružena logička vrednost TRUE, kada učenik reši zadatak pogrešno biće mu prikazano tačno rešenje. U suprotnom, tačno rešenje neće biti prikazano.

- **dispLT** - je javna procedura koja ima zadatak da prikaže sve informacije vezane za zadatak (objašnjenja zadatka i postavljanje upita) i da prihvati učenikov odgovor. Sve pojave objekta *LTask* koriste istu proceduru.

Do sada razmatrane promenljive predstavljaju fiksni deo zadatka. Medjutim, postoji niz različitih tipova zadataka koji se mogu postavljati učenicima u procesima učenja. U ovom autorskom sistemu predloženo je pet tipova zadataka koji pokrivaju značajan skup upita i zadataka koji se mogu postavljati učenicima. Ostali tipovi zadataka bi se mogli definisati na sličan način ili kao sasvim nove klase.

Uvode se sledeći tipovi zadataka.

- **Prompt** - je takav tip zadatka gde se od učenika očekuje da uz pomoć tastature ukuca svoje rešenje tj. odgovor.

- **Test** - je takav tip zadatka gde se od učenika očekuje da izabere ispravan odgovor iz niza ponudjenih alternativnih odgovora.

- **Pair** - je takav tip zadatka gde se učeniku nude dve grupe rečenica koje on treba da upari u tačne odgovore. Svakoj rečenici iz prve grupe odgovara tačno jedna rečenica iz druge grupe.

- **Bool** - je takav tip zadatka gde se od učenika očekuje da na postavljeni zadatak odgovori samo sa da ili ne. U osnovi, ovaj tip je sličan tipu zadatka Prompt, čak poseduje i istu strukturu, ali može se posmatrati i odvojeno.

- **Exec** - je takav tip zadatka - vežbe u kom se poziva neka izvršna datoteka uz čije korišćenje i pomoć učenik treba nešto da uvežba. Na primer, poziv Prolog interpretatora da bi proverio ili napisao neki program; WordStar tekst procesora, da bi proverio neku izloženu instrukciju itd.

Tip zadatka Prompt

Ako je *LTask* tipa **Prompt**, tada klasa ima još pet dodatnih vezanih promenljivih.

- **correct** - je vezana promenljiva koja sardži tačno rešenje zadatka. Rešenje zadatka je zadato u obliku teksta tipa STRING.

- **answer** - je vezana promenljiva koja se u procesu učenja popunjava učenikovim rešenjem na postavljeni zadatak, takodje tipa STRING.

- **isCorr** - je privatna procedura koja je u većini slučajeva različita za svaki objekat i prilagodjena konkretnom objektu. Ona treba da uporedi tačno rešenje sa učenikovim, i da vrati vrednost **TRUE** ako je učenikovo rešenje zadovoljavajuće. U suprotnom vraća vrednost **FALSE**. Ova procedura može koristiti neke poznate algoritme za poredjenje stringova ili može sadržati i metode za semantičko prepoznavanje ispravnosti odgovora.

- **ifCorr** - sardži akciju koju treba preduzeti (tj. poziciju na kojoj se nastavlja proces učenja) ako je učenik ispravno rešio zadatak.

- **ifNot** - sardži akciju koju treba preduzeti (tj. poziciju na kojoj se nastavlja proces učenja) ako učenik nije ispravno rešio zadatak.

Naredna pozicija može biti neki članak lekcije, ili neki novi zadatak tekućeg članka ili kraj prezentacije tj. procesa učenja.

Tip zadatka Test

Ako je *LTask* tipa **Test**, tada klasa ima još jednu dodatnu vezanu promenljivu.

- **answer** - sadrži listu svih alternativnih odgovora pridruženih odgovarajućem zadatku. U procesu učenja učeniku se nude te alternative kao potencijalni odgovori i on izabira tačno rešenje. Učenik izabira samo jedno od ponudjenih rešenja za koje smatra da je tačno. U zavisnosti od izabranog rešenja, nastavlja se dalje proces prezentacije. Tj. svaka alternativa opredeljuje svoj sopstveni tok prezentacije.

Tip zadatka Pair

Ako je *LTask* tipa **Pair** tada klasa ima još šest dodatnih vezanih promenljivih.

- **answer1** - je lista parcijalnih rešenja postavljenog zadatka i treba je dopuniti, tj. vezati sa parcijalnim rešenjima iz liste **answer2**.

- **answer2** - je lista parcijalih rešenja koja se uparuju sa rešenjima iz liste **answer1**.

Liste answer1 i answer2 imaju isti broj elemenata i svaki od njih je zadat u obliku teksta tipa TEXT.

- **true** - je lista brojeva. Prvi element liste true odredjuje sa kojim rešenjem iz druge liste (answer2) je povezan prvi element prve liste (answer1). Drugi element liste true odredjuje sa kojim rešenjem iz druge liste (answer2) je povezan drugi element prve liste (answer1) i tako redom. Brojevi iz liste true predstavljaju redne brojeve odgovarajućih elemenata rešenja u listi answer2.

Moglo bi se, radi veće opštosti, pretpostaviti da pozicija u listi true ne mora da odgovara poziciji alternative u listi answer1. Tj. uparivanje se ne mora prikazivati redom po alternativama iz prve liste. U tom slučaju elementi liste true bi mogli biti parovi. Parovi bi se mogli prikazati ugradjenim tipom podataka n-torka (opisanim u delu 4.1.5)

T(NUMBER, NUMBER)

Prvi element para je pozicija alternative iz prve liste, a drugi element para je pozicija odgovarajuće alternative iz druge liste.

- **isCorr** - je privatna procedura koja treba da utvrdi kada se učenikov odgovor tretira kao tačan. Može da se zahteva potpuna i apsolutna uparenost odgovora ili se može odrediti neki procenat tačnog uparivanja da bi se odgovor prihvatio kao korektan.

- **ifCorr, ifNot** - imaju isto značenje kao u tipu zadatka **Prompt**.

Tip zadatka Bool

Ako je *LTask* tipa **Bool** tada klasa ima još pet dodatnih vezanih promenljivih.

- **correct** - sadrži informaciju o tome da li na postavljeno pitanje treba odgovoriti sa da ili sa ne. Ako je sadržaj polja **TRUE** na postavljeno pitanje treba odgovoriti sa da, a u suprotnom sa ne.

- **Answer** - je vezana promenljiva koja se u procesu učenja popunjava učenikovim odgovorom.

- **isCorr, ifCorr, ifNot** - imaju isto značenje kao u tipu zadatka **Pair**.

Tip zadatka Exec

Ako je *LTask* tipa **Exec** tada klasa ima još dve dodatne vezane promenljive.

- **exeDat** - je vezana promenljiva koja sadrži ime neke izvršne datoteke koju treba aktivirati da bi učenik obavio vežbu ili zadatak.

- **next** - sadrži akciju koju treba preduzeti (tj. poziciju na kojoj se nastavlja proces učenja) kada se učenik 'vrati' iz aktiviranog izvršnog programa.

5.2.2.6 Struktura alternativa

Alternativa služi da bi se učeniku u procesu učenja nakon postavljenog zadatka ponudio niz odgovora od kojih on treba, po svom mišljenju, da odabere tačan.

Svaka alternativa se može prikazati kao sledeća klasa.

```
Alter = CLASS {
                ident : STRING;

                CLASS Info EXCL ident, key, sou;

                act :       CLASS Action;

                isCorr :    LOGICAL;
        }
```

Klasa ***Alter*** nasledjuje klasu ***LInfo*** za reprezentaciju informacija za svaku pojedinu alternativu. Osim toga, klasa sadrži još i dve vezane promenljive.

- **act** - je vezana promenljiva koja sadrži akciju koju treba preduzeti (tj. poziciju na kojoj treba nastaviti proces učenja) ako je učenik odabrao tu alternativu kao rešenje za postavljeni zadatak.

- **isCorr** - je vezana promenljiva koja sadrži informaciju o tome da li je ta alternativa tačno rešenje zadatka (za koji je vezana). Ova vrednost je potrebna kao informacija, ako se za zadatak zahteva prikaz tačnog odgovora. (tj. *LTask*.disp = TRUE).

5.2.2.7 Struktura akcije

Action je poslednja klasa koja je uključena u reprezentaciju lekcije. U njoj je sadržana informacija o daljem toku procesa učenja.

Učenje može biti usmereno na kraj, na neki članak lekcije, ili na neki zadatak istog članka. Klasa ***Action*** se definiše na sledeći način.

```
Action = CLASS {

                CASE Type : ( EndOfLearn, GoToTopic, GoToTask ) OF

                EndOfLearn
```

```
                { en : END };

        GoToTopic

                { Top : IDENT LTopic };

        GoToTask

                { Tsk : IDENT LTask }
}
```

- Ako je vrednost težinske promenljive **Type** EndOfLearn, proces učenja tj. prezentacije se završava. Članak koji sadrži *Action* objekt tog tipa, predstavlja završni čvor u multidigrafu lekcije.

- Ako je vrednost težinske promenljive **Type** GoToTopic, tada je vrednost ime tj. jedinstveni identifikator članka od koga treba da se nastavi proces učenja.

- Ako je vrednost težinske promenljive **Type** GoToTask, tada je vrednost jedinstveni identifikator zadatka od koga treba da se nastavi proces učenja. Podrazumevaćemo da ***LTask*** nije vidljiv izvan odgovarajućeg objekta ***LTopic*** i stoga su dostupni samo zadaci unutar tekućeg članka. Medjutim, u opštem slučaju nema nikakvih problema, pošto svaki objekat tipa ***LTask*** ima svoj jedinstveni identifikator, da se proces učenja nastavi i na bilo kom proizvoljnom zadatku nekog drugog članka. Zadaci su u tom slučaju vidljivi iz proizvoljne pozicije lekcije.

Klasa *Action* se može učiniti suvišnom u ovom sistemu i strukturi. Ta činjenica je tačna. Umesto klase *Action* u strukturama klasa mogla bi se koristiti specifikacija **IDENT** koja označava da se radi o jedinstvenom identifikatoru nekih objekata. Ali, upotreba ove klase ilustruje mogućnost korišćenja varijabilnih klasa, a mogla bi da posluži za slučaj da se tok učenja usmerava na osnovu slobodnog izbora učenika. Ako bi se u svakom trenutku procesa učenja, učeniku dozvolilo da bez obzira na rešenja zadatka može da izabere poziciju na kojoj se nastavlja učenje (zadatak ili članak) onda bi se tip akcije odredjivao u samoj prezentaciji (dinamički) i onda bi klasa morala da bude varijabilna.

5.2.3 Popunjavanje lekcije

Program napisan na *Less* jeziku predstavlja ujedno i proces kreiranja lekcije i program za reprezentaciju informacija koje će se uz pomoć odgovarajućih procedura koristiti u procesu učenja tj. prezentacije lekcije.

Jezik *Less* je pomoćno orudje za formiranje nastavne sekvence i popunjavanje strukture odgovarajućeg multidigrafa.

Jezik poseduje niz jednostavnih direktiva i komandi koje manipulišu pojedinim objektima kao nezavisnim celinama, i omogućavaju jednostavan i pogodana način za formiranje lekcija iz različitih oblasti.

Svakim programom *Less* jezika se formira multidigraf lekcije koji, kasnije, svaki učenik može jednostavno da koristi u procesu učenja tj. prezentacije.

Definisanje potrebnih struktura

Struktura lekcije se opredeljuje u prva dva bloka svakog programa. U njima se definišu sve komponente nastavne sekvence - multidigraf, članci, zadaci, alternative, kako bi se mogle jednostavnije identifikovati i pristupati im se u fazi popunjavanja.

Blok definisanja procedura

Za ustanovljenu strukturu podataka za reprezentaciju nastavne sekvence, može se realizovati skup svih javnih procedura, a i dati smernice za formiranje privatnih procedura. One bi se eksplicitno pridružile odgovarajućim vezanim promenljivim objekata nastavne sekvence i profesor ne bi mogao da utiče na njih. Tačnije, ne bi mogao niti da ih menja i prilagodjava nekim novim zahtevima, niti da umesto njih koristi nove procedure. Na ovaj način se od profesora ne zahteva nikakvo programersko znanje, a postupak kreiranja lekcije se ne bi puno razlikovao od njegovih beleški za pripremu časa. Prihvaćene procedure bi u potpunosti vodile i podržavale proces učenja.

Takodje, ako bi profesor imao ideje, znanja i volje mogao bi na sličan način da definiše nove tipove zadataka, a ako ima nekakvo programersko znanje mogao bi doradjivati postojeće procedure ili čak pisati potpuno nove.

Blok popunjavanja lekcije

Deo popunjavanja nastavne sekvence je poslednja celina svakog *Less* programa i jedina koju profesor mora da popuni. U ovom delu programa profesor ima zadatak da definisanu nastavnu sekvencu popuni odgovarajućim sadržajima, pridruži svakom članku odgovarajuće zadatke i odredi akcije, kao posledice odgovora na postavljene zadatke.

Rezultat ove faze će (pri prevodjenju programa) biti popunjeni multidigraf nastavne sekvence. U suštini razlikuje se nekoliko načini i mogućnosti popunjavanja nastavne sekvence što uglavnom zavisi od opredeljenja profesora.

5.2.4 Proces učenja

Interpretacija lekcije tj. proces učenja započinje u nekom od početnih članaka i dalje se nastavlja u zavisnosti od učenikovog rešenja postavljenog zadatka, i akcija priloženih uz svako rešenje. Ovaj proces podržava i vodi niz javnih (i privatnih) procedura ugradjenih u strukturu lekcije.

Opšta forma interpretacije lekcije, će biti opisana javnim procedurama **Int**, **DisplayQuery** i **DisplayAllInfo** koje iniciraju i podržavaju kompletan proces učenja.

Procedura za prezentaciju lekcije - Int

Procedura **Int** (koja je vezana u klasi *Lesson* za promenljivu interpr) ima zadatak da prikaže sve početne članke lekcije, da prepusti učeniku izbor jednog početnog i da inicira proces učenja pozivajući ostale procedure prema potrebi. U nastavku je dat grubi algoritam ove procedure.

```
PROCEDURE Int(LES : IDENT Lesson);
BEGIN
    (* formira objekat tipa Student i popunjava njegov jedinstveni identifikator *)

    (* prikazuje sve početne članke nastavne sekvence *)

    First(&LES.start,MemLes);          (* uzima prvi identifikator početnih
                                          čvorova *)

    WHILE NOT EndOfList(&LES.start) DO
    BEGIN
        (* odredjuje poziciju za naredni ispis identifikatora čvorova *)
        WriteStr(MemLes);              (* ispisuje identifikator čvora *)
        Next(&LES.start,MemLes);       (* pozicionira se na sledeći identifikator
                                          *)
    END;

    (* učesnik procesa učenja izabira jedan čvor kao početni *)
    Read(&LES.currTpc);

    (* započinje procesa učenja *)

    WHILE &LES.currTpc < > END DO
    BEGIN
        (* formira element za tekući članak u objektu Student *)
```

```
            interAll(&LES.currTpc);                 (* prikazuje sve informacije koje
                                                       su reprezentovane u tekućem
                                                       čvoru*)
            First(&LES.CurrTpc.query,PomLis);       (* uzima prvi zadatak *)
            &LES.currTsk = &PomLis.ident;           (* postavi tekući zadatak na
                                                       odgovarajuću vrednost *)

            (* prikazuje zadatak i prihvata odgovor *)
            WHILE &PomLis < > END DO
            BEGIN
                  (* formira objekat za tekući zadatak i vezuje ga za tekući objekat
                  članka u objektu Student *)
                  &PomLis.dispLT(&LES,&LES.currTsk); (* prikazuje zadatak i
                                                        podržava prezentaciju do
                                                        prihvatanja rešenja *)
                  PomLis = &LES.currTsk;
            END;
      END;
END;
```

Procedura za prikaz zadatka - DisplayQuery

Procedura koja podržava rukovanje objektima tipa *LTask* vezana je za promenljivu dispLT.

Pošto je procedura DisplayQuery javna, ona je automatski vezana za svaku pojavu objekta *LTask*. Ona će prikazati informacije o zadatku, prihvatiti odgovor učenika i dalje usloviti prezentaciju u zavisnosti od pridružene akcije. U proceduri se mora voditi računa o svim tipovima zadataka koji se dopuštaju i mora se sa njima rukovati na adekvatan način.

PROCEDURE **DisplayQuery**(LES : IDENT *Lesson*; TAS : IDENT *LTask*);

FUNCTION **FindCorrectIn**(ITSK: IDENT *LTask*) : IDENT *Alter*;

(* nalazi ispravnu alternativu u listi alternativa zadatka čiji je jedinstveni identifikator zadat argumentom ITSK i vraća kao rezultat jedinstveni identifikator odgovarajuće alternative *)

```
BEGIN
      .....
END;
```

PROCEDURE **ProceedWith**(Act: IDENT *Action*);

(* u zavisnosti od tipa akcije postavlja vezane promenljive currTpc i currTsk u objektu tipa *Lesson* čiji jedinstveni identifikator se nalazi u argumentu LES *)

```
BEGIN
      CASE Act.Type OF

            EndOfLearn:
                  &LES.currTpc := END;
                  &LES.currTsk := END;

            GoToTopic:
                  &LES.currTsk := END;
                  &LES.currTpc := &act.Top;

            GoToTask:
                  &LES.currTsk := &act.Tsk;
      END;
END;

BEGIN       (* DisplayQuery *)
      IF NOT &TAS.aldSlvd THEN      (* ako zadatak nije već rešavan u prethodnim
                                       prolascima kroz taj članak treba ga postaviti
                                       učeniku *)
      BEGIN
            interAll(&TAS);         (* objašnjava zadatak tj. prikazuje sve
                                       informacije koje su u njemu reprezentovane *)
            CASE &TAS.type OF

            Prompt:
                  ReadAns(&TAS.answer);     (* prihvata odgovor *)

                  (* poziva privatnu proceduru isCorr koja treba da utvrdi ispravnost
                  datog rešenja u odnosu na tačno rešenje*)

                  IF &TAS.isCorr(&TAS.answer, &TAS.correct) THEN
                  BEGIN
                        &TAS.aldSlvd := TRUE;
                        ProceedWith(&TAS.ifCorr)
                  END
                  ELSE
                  BEGIN
                        IF &TAS.disp THEN
                        BEGIN
                              (* prikazuje tačan odgovor *)
                              WriteLn('Correct answer is ', &TAS.correct)
                        END;
                        ProceedWith(&TAS.ifNot)
                  END;
```

```
Test:
        First(&TAS.answer,A);

        (* prikazuje sve alternative *)
        WHILE NOT EndOfList(&TAS.answer) DO
        BEGIN
              interAll(A);                (* prikazuje sve informacije
                                          memorisane u alternativi *)
              Next(&TAS.answer,A)         (* prelazi na sledeću alternativu
                                          *)
        END;

        ReadAns(AnsA);                    (* prihvata odgovor *)
        FindA(AnsAlt,AnsA);               (* smešta u promenljivu AnsAlt
                                          jedinstveni identifikator izabrane
                                          alternative *)
        IF &AnsAlt.isCorr THEN            (* ispravan odgovor *)
              &TAS.aldSlvd := TRUE        (* ako je izabrana alternativa
                                          tačno rešenje zadatka u narednim
                                          eventualnim prolazima kroz isti
                                          taj članak, ne treba postaljati taj
                                          zadatak ponovi *)
        ELSE
        BEGIN                             (* neispravan odgovor *)
              IF &TAS.disp THEN
              BEGIN
                    (* prikazije ispravnu alternativu *)
                    TacAlt = FindCorrectIn(&TAS.answer);
                    interAll(&TacAlt);
              END;
        END;
        ProceedWith(&AnsAlt);

Pair :
        First(&TAS.answer1,FirAns);

        (* prikazuje sve prve alternative *)
        WHILE NOT EndOfList(&TAS.answer1) DO
        BEGIN
              Disply_TXT(FirAns);         (* prikazuje tekstualne
                                          informacije memorisane
                                          kao prve alternative *)
              Next(&TAS.answer1,PrviAns)  (* prelazi na sledeću
                                          alternativu *)
        END;
```

```
        First(&TAS.answer2,SecAns);

        (* prikazuje sve druge alternative *)
        WHILE NOT EndOfList(&TAS.answer2) DO
        BEGIN
              Disply_TXT(SecAns);          (* prikazuje tekstualne
                                              informacije memorisane
                                              kao druge alternative *)
              Next(&TAS.answer2,SecAns)    (* prelazi na sledeću
                                              alternativu *)
        END;
        ReadArray(HelTrue);          (* učitava listu brojeva koja označava
                                        koji elementi iz druge liste alternativa se
                                        vezuju za (redom prvi, drugi, itd.)
                                        elemente prve liste alternativa *)

        IF &TAS.isCorr(&TAS.true,HelTrue) THEN
              (* Ispituje da li se i koliko podudaraju dati odgovori sa
              očekivanim tj. tačnim. Ovde profesor može da odluči koji
              procenat ispravnih uparivanja je prihvatljiv. *)
        BEGIN
              &TAS.aldSlvd = TRUE;
              ProceedWith(&TAS.ifCorr.ident);
        END
        ELSE
        BEGIN
              IF &TAS.disp THEN
              BEGIN
                    (* prikazuje listu ispravnih odgovora *)
                    First(&TAS.true,HelTrue);

                    i = 0;
                    WHILE NOT EndOfList(&TAS.true) DO
                    BEGIN
                          i = i + 1;
                          WriteStr(STR(i)+'   '+STR(HelTrue));
                          Next(&TAS.true,HelTrue);
                    END;
              END;
              ProceedWith(&TAS.ifNot.ident);
        END;
Bool:
        ReadAns(&TAS.answer);     (* prihvati odgovor *)
```

```
                (* poziva privatnu proceduru isCorr koja treba da utvrdi da li je
                odgovor ispravan tj. da li je on da ili ne *)

                IF &TAS.isCorr(&TAS.answer, &TAS.correct) THEN
                BEGIN
                        &TAS.aldSlvd := TRUE;
                        ProceedWith(&TAS.ifCorr)
                END
                ELSE
                BEGIN
                        IF &TAS.disp THEN
                        BEGIN
                                (* prikazuje tačan odgovor *)
                                WriteLn('Correct answer is ', &TAS.correct)
                        END;
                        ProceedWith(&TAS.ifNot)
                END;

        Exec :
                (* poziva i aktivira izvršnu datoteku *)
                &TAS.exeDat;
                ProceedWith(&TAS.next)          (* preuzimanje akcije nakon
                                                vraćanja iz izvršne datoteke *)

        END; (* CASE *)
    END; (* If NOT &TAS.aldSlvd ... *)
END; (*DisplayQuery *)
```

Procedura za prikaz memorisanih informacija - DisplayAll

Procedura **DisplayAll** je vezana za svaku pojavu objekta Info. Ona upravlja i reguliše prikaz svih informacija smeštenih u objektu tipa *LInfo*.

```
PROCEDURE DisplayAll(IDT : STRING);

(* IDT je put do objekta iz koga treba prikazati reprezentovane informacije *)

(* procedura vrši sinhronizovani prikaz svih informacija iz odgovarajućeg objekta *)
BEGIN
        (* prikazivanje tekstualnih informacija koje su memorisane u objektu koji je dostupan
        pomoću puta koji je zadat u argumentu IDT *)

        Display_TXT(&IDT);
```

```
        (* pristup grafičkim informacijama zadatog objekta ako ih ima i njihova adekvatna
        prezentacija *)

        First(&IDT.pic,P);

        WHILE NOT EndOfList(&IDT.pic) DO
                Display_PIC(P);                 (* prezentacija grafičkog prikaza *)
                (* zadržavanje slike neko vreme na ekranu *)
                Next(&IDT.pic,P);               (* uzimanje sledeće slike *)
        END;

        (* pristupanje simulacijama zadatog objekta ako ih ima i njihova prezentacija *)

        First(&IDT.ani,A);

        WHILE NOT EndOfList(&IDT.ani) DO
                Display_ANI(A);                 (* prezentacija animacije *)
                (* zadržavanje animacije neko vreme na ekranu *)
                Next(&IDT.ani,A)                (* uzimanje sledeće animacije *)
        END;
END;
```

5.2.5 Primena realizovanog autorskog sistema

Izložena struktura podataka za reprezentaciju informacija u obrazovanju i procedure za podržavanje procesa korišćenja tj. prezentacije lekcije, predstavljaju osnovni koncept autorskog sistema. Uz manje ili više modifikacija može se dobiti niz različitiha autorskih sistema.

Da bi se ilustrovao način korišćenja autorskog sistema i njegova primena, u nastavku je dat jedan konkretan primer lekcije.

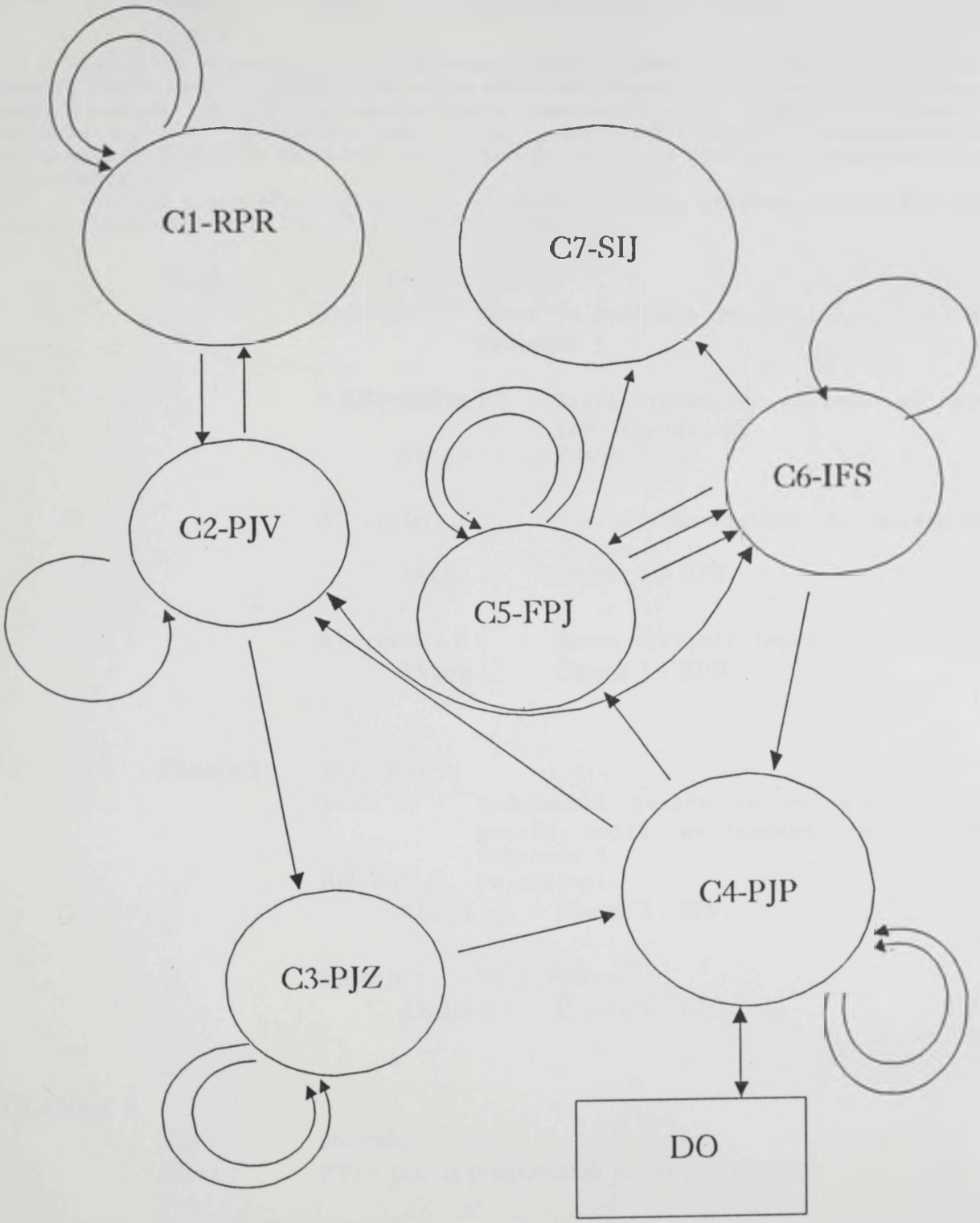
NAPOMENA: Pretpostavlja se da su prva tri znaka iz ident vezane promenljive u objektima dovoljno da bi se identifikovao objekat. Ova 'olakšavajuća okolnost' se uzima da se ne bi rukovalo dugačkim identifikatorima objekata.

U opštem slučaju taj problem se može razrešiti na dva načina:

1) uvodjenjem nove vezane promenljive koja bi opisivala sadržaj iz objekta,

2) ugradnjom, prilikom implementacije strukture klasa i *Less* jezika, činjenice da je samo prvih n znakova ident vezane promenljive dovoljno da se jedinstveno odredi objekat i da mu se pristupa na taj način.

Na slici 5.3. prikazan je multidigraf jedne konkretne lekcije o istoriji programskih jezika i osnovnim svojstvima funkcionalnih programskih jezika [Budimac, 1991b]. Baza multidigrafa lekcije se sastoji od sedam članaka. U primeru, znak *, ispred alternative označava da je ta alternativa ispravno rešenje zadatka.



Sl 5.3 Struktura multidigrafa lekcije.

Struktura multidigrafa lekcije

ČLANAK 1.

Tip : početni

Ident : RPR - rešavanje problema pomoću računara

Text 1 :

Jedan od pravaca razvoja računara bilo je uvodjenje programskih jezika pomoću kojih se na sistematičan način opisuje proces izračunavanja računarom. Proces rešavanja zadataka zasniva se na rasčlanjavanju složenog (polaznog) zadatka na niz jednostavnijih, i to predstavlja proces programiranja. Programski jezici su veštački jezici pomoću kojih se realizuje programiranje na računaru.

Postoje razne klasifikacije programskih jezika, zavisno od toga šta se koristi kao glavni kriterijum klasifikacije.

Pitanje 1: **Tip** : Test

Tekst 1 : Kakav je postupak rešavanja zadataka na računaru ?

*** Alternativa 1 :** Rasčlanjavanjem zadatka na niz jednostavnijih.

Akcija : Pitanje 2.

Alternativa 2 : Tekstualnim opisom na prirodnom jeziku.

Akcija : Članak 1 - RPR.

Alternativa 3 : Nizom strujnih impulsa.

Akcija : Članak 1 - RPR.

Pitanje 2: **Tip** : Bool

Tekst 1 : Programski jezici su veštački jezici pomoću kojih se rešava zadatak na računaru ?

Rešenje : Da (tačno).

Akcija : Članak 2 - PJV.

Rešenje : Ne (netačno).

Akcija : Članak 1 - RPR.

ČLANAK 2.

Tip : unutrašnji

Ident : PVJ - podela programskih jezika po vremenu

Text 1 :

Ako se kao glavni kriterijum za podelu programskih jezika razmatra vreme nastanka programskog jezika, razvrstavanje se može izvršiti po generacijama.

1. 1954. - 1958. - jezici prve generecije: FORTRAN, ALGOL 58, Flowmatic, ...

2. 1959. - 1969. - jezici druge generacije: FORTRAN II, ALGOL 60, COBOL 61, LISP,

3. 1962. - 1969. - jezici treće generacije: PL/I, ALGOL 68, Pascal, Simula,

4. 1970. - 1979. - predstavlja generacijski jaz u razvoju programskih jezika, ali su se razvili zanimljivi jezici: CLU, CSP, Ada, Smalltalk,

5. 1980. - do današnjih dana - razvijaju se različite paradigme programskih jezika.

Pitanje 1: **Tip** : Prompt
Tekst 1 : Koji jezici su jezici treće generacije?

Rešenje : PL/I, algol68, Pascal, Simula.
Akcija za tačno rešenje: Pitanje 2.
Akcija za netačno rešenje: Članak 2 - PVJ.

Pitanje 2: **Tip** : Prompt
Tekst 1 : Koliko ima generacija programskih jezika?
Rešenje : 4
Akcija za tačno rešenje: Članak 3 - PJZ.
Akcija za netačno rešenje: Članak 1 - RPR.

ČLANAK 3.

Tip : unutrašnji
Ident : PJZ - podela programskih jezika po zavisnosti od računara
Text 1 :

Kada se kao glavni kriterijum podele programskih jezika koristi stepen zavisnosti od računara, svi programski jezici se mogu razvrstati u dve velike klase:

1. mašinski zavisni,
2. mašinski nezavisni (viši programski jezici).

Ova klasifikacija je potpuna jer obuhvata sve programske jezike, medjutim danas je od posebnog značaja razvrstavanje viših programskih jezika.

Pitanje 1: **Tip** : Test
Tekst 1 : Kako se dele programski jezici po zavisnosti u odnosu na računar ?

Alternativa 1 : Na mašinski zavisne i asemblerske.
Akcija : Članak 3 - PJZ.

Alternativa 2 : Na mašinski nezavisne i simboličke.
Akcija : Članak 3 - PJZ.

*** Alternativa 3** : Na mašinski zavisne i mašinski nezavisne.
Akcija : Članak 4 - PJP.

ČLANAK 4.

Tip : početni

Ident : PJP - podela po primeni programskih jezika

Text 1 :

Ako je glavni kriterijum podele programskih jezika oblast njegove najčešće primene, onda se može uočiti više klasa:

1. Jezici za primenu u matematici i tehnici (FORTARN, ALGOL, ...)
2. Jezici za poslovnu primenu (COBOL, RPG, ...)
3. Jezici za obuku programiranja (Pascal, BASIC, ...)
4. Jezici za primenu u veštačkoj inteligenciji (LISP, PROLOG, ...)
5. Univerazlni jezici (PL/I, Ada, ...)

....

Ukoliko je glavni kriterijum za podelu programskih jezika način rešavanja problema pomoću nekog programskog jezika, svi viši programski jezici mogu se podeliti u dve klase:

1. Proceduralne (imperativne) i
2. Deklarativne (deskriptivne).

Pitanje 1: **Tip** : Test

Tekst 1 : Koliko ima podela programskih jezika po kriterijumu primene ?

*** Alternativa 1 :** Mnogo.
Akcija : Pitanje 2.

Alternativa 2 : Pet.
Akcija : Članak 2 - PJV.

Alternativa 3 : Prema oblasti primene.
Akcija : Pitanje 2.

Pitanje 2: **Tip** : Pair

Tekst 1 : Koji jezici pripadaju kojoj grupi?

Alternativa 11 : Oblast matematike i tehnike. -> Alternativa 23.

Alternativa 12 : Oblast poslovne primene. -> Alternativa 22.

Alternativa 13 : Obuka u programiranju. -> Alternativa 24.

Alternativa 14 : Oblast veštačke inteligencije. -> Alternativa 21.

Alternativa 21 : LISP, PROLOG, ...
Alternativa 22 : COBOL, RPG, ...
Alternativa 23 : FORTH, ALGOL, ...
Alternativa 24 : PASCAL, BASIC, ...

Akcija za tačno rešenje: Pitanje 3.
Akcija za netačno rešenje: Članak 4 - PJP.

Pitanje 3: **Tip** : Prompt

Tekst 1 : Kakve jezike razlikujemo u odnosu na način rešavanja problema?

Rešenje : procedurelne, deklarativne

Akcija za tačno rešenje: Članak 5 - FPJ.

Akcija za netačno rešenje: Dodatno objašnjenje.

ČLANAK 5.

Tip : unutrašnji

Ident : FPJ - funkcionalni programski jezici

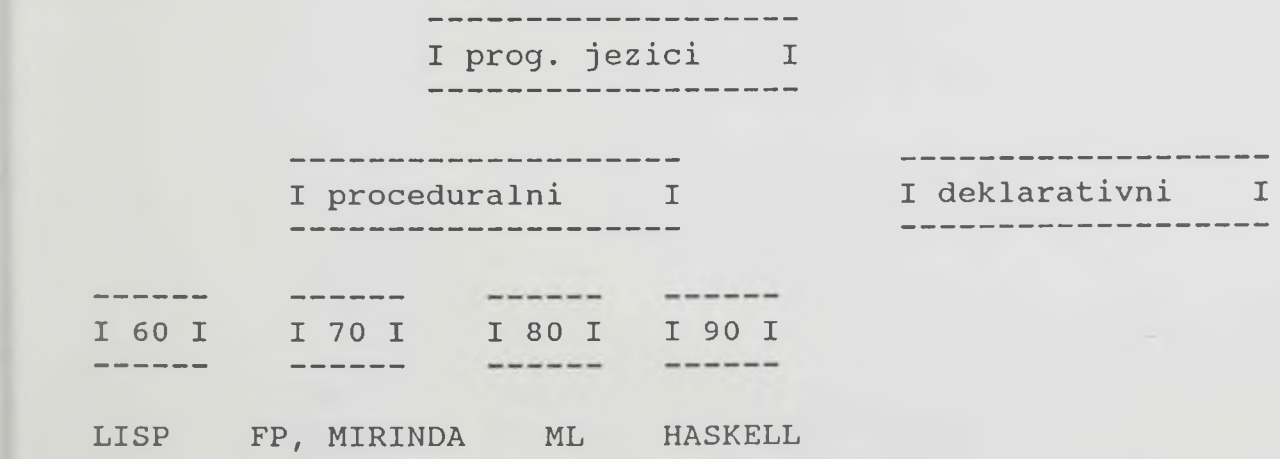
Text 1 :

Jedna od značajnih kategorija programskih jezika su funkcionalni programski jezici. Funkcionalni jezici su evoluirali od prvog funkcionalnog jezika LISP (koji se pojavio šezdesetih godina), preko FP, Miranda jezika (u 70-tim godinama) i ML (u 80-tim godinama) do jezika Haskell čija je definicija prvi put objavljena početkom 1990. godine.

U funkcionalnim programima postoje samo sledeće mogućnosti:

- definisanje funkcije, tj. pridruživanje imenu funkcije, izraza kojim se izračunava vrednost funkcije,
- poziv funkcije, tj. aktiviranje ranije definisane funkcije sa njenim stvarnim argumentima,
- kompozicija funkcija kao jedina operacija.

Slika 1 : Godine nastanka nekih značajnih funkcionalnih programskih jezika.



Slika 1. Godišta nastanka fukcionalnih programskih jezika.

Slika 2 : Istorija razvoja značajnih programskih jezika. (* slika je data u primeru HyperText-a *)

Pitanje 1: **Tip** : Prompt

Tekst 1 : Nabroj nekoliko značajnih funkcionalnih programskih jezika ?

Rešenje : LISP, FP, MIRINDA, HASKELL.
Akcija za tačno rešenje: Pitanje 2.
Akcija za netačno rešenje: Članak 5 - FPJ.

Pitanje 2: Tip : Test
Tekst 1 : Prolog je evoluirao od LISP-a ?

* Alternativa 1 : Da.
Akcija : Članak 6 - IFS.

Alternativa 2 : Ne.
Akcija : Članak 2 - PJV.

Alternativa 3 : Možda.
Akcija : Pitanje 3.

Pitanje 3: Tip : Test
Tekst 1 : Program u funkcionalnom programskom jeziku se sastoji od ?

Alternativa 1 : Definicija funkcija, poziva funkcija i niza komandi.
Akcija : Članak 6 - IFS.

* Alternativa 2 : Definicija, poziva i kompozicija funkcija.
Akcija : Članak 7 - SIJ.

Alternativa 3 : Poziva funkcija i definicija funkcija.
Akcija : Članak 5 - FPJ.

ČLANAK 6.

Tip : unutrašnji
Ident : IFS - imperativni - funkcionalni stil
Text 1 :

Funkcionalni stil programiranje je superiorniji od tradicionalnog, imperativnog stila. Razlike izmedju ova dva stila postoje i uglavnom se ogledaju u sledećem:

1. Program u imperativnom jeziku je, uobičajeno lista komandi, koje treba da se izvrše u odredjenom redosledu.
2. Program u funkcionalnom jeziku se koristi da definiše izraz koji je rešenje skupa problema.

Text 2 :

Jedna velika klasa funkcionalnih jezika - čisto funkcionalni jezici nemaju kontrolne strukture. Zbog odsustva kontrolnih struktura proističu dve prednosti u korišćenju funkcionalnih programskih jezika:

1. ne mora se voditi računa o sekvenci komandi,
2. ne moraju se traži greške nastale kao posledica eksplicitne sekvence komandi.

Pitanje 1: **Tip** : Test
Tekst 1 : Da li funkcionalni programski jezici imaju kontrolne strukture ?

Alternativa 1 : Svi funkcionalni programski jezici imaju kontrolne strukture.
Akcija : Članak 5 - FPJ.

*** Alternativa 2 :** Neki funkcionalni programski jezici imaju kontrolne strukture.
Funkcionalni programski jezici nemaju kontrolne strukture.
Akcija : Članak 4 - PJP.

Pitanje 2: **Tip** : Pair
Tekst 1 : Povezati odgovarajuće činjenice?
Alternativa 11 : Program ne vodi računa o sekvenci komandi -> Alternativa 21.
Alternativa 12 : Program je sekvenca komandi. -> Alternativa 22.

Alternativa 21 : U programu nema grešaka zbog sekvence komandi.
Alternativa 22 : U programu postoje sporedni efekti.

Akcija za tačno rešenje: Članak 7 - SIJ.
Akcija za netačno rešenje: Članak 6 - IFS.

ČLANAK 7.

Tip : završni
Ident : SIJ - slabost imperativnih programskih jezika
Text 1 :

Jedna od značajnih slabosti imperativnih programskih jezika potiče od činjenice da procedure i funkcije u tim jezicima imaju sporedne efekte.

Primer 1. Pogledajmo globalnu deklaraciju u PASCAL programu.

```
...
VAR
      x,y,glob : INTEGER;
      b1,b2,b3,b4 : BOOLEAN;
FUNCTION sala(i : INTEGER) : BOOLEAN;
BEGIN
      glob := glob + 1;
      sala := i = glob;
END;
```

Razmotrimo efekat sledeće komponovane naredbe.

```
BEGIN
      glob := 0;
      x := 1;
      y := 1;
      b1 := x = y;
      b2 := sala(y);
      b3 := sala(x);
      b4 := sala(y);
END;
```

Posle ovog izračunavanja b1 i b2 su tačni ali su b3 i b4 netačni. Ovo svojstvo imperativnih programskih jezika je nešto što je potpuno strano u jeziku matematike. U matematici funkcije uvek vraćaju istu vrednost kada se pozovu sa istim argumentima.

Pitanje 1: **Tip** : Exec
Tekst 1 : Kompletirati predloženi zadatak i proveriti njegov rad Pascal jeziku ?

Akcija : END.

DODATNO OBJAŠNJENJE.

Tip : unutrašnji
Ident : dodatno objašnjenje

Text 1 :

Ilustrujmo dva pristupa programiranju (imperativni i deklarativni) na primeru specifikacije barake.

Prvi, imperativni pristup predstavlja opis načina izgradnje barake:

- postaviti temelje,
- sazidati zidove,
- postaviti pod,
- postaviti krov.

Drugi, funkcionalnio pristup predstavlja opis strukture barake, uz pomoć njenih sastavnih delova:

- zidovi se oslanjaju na temelj,
- pod se oslanja na temelj,
- krov se oslanja na zidove.

Imperativni (komandni) opis prvog pristupa može se jasno suprotstaviti statičkom, definicionom opisu drugog pristupa. Eksplicitni redosled akcija koje se zahtevaju u prvom opisu može se zameniti implicitnim redosledom uslovljenim odnosima izmedju objekata u drugom opisu.

Pitanje 1: **Tip** : Test
Tekst 1 : Povratak na prethodni članak ?

Alternativa 1 : Da.
Alternativa 2 : Ne.

Akcija : Članak 4 - PJP.

Popunjavanje strukture lekcije

U datom primeru lekcije u člancima se prikazuju tekstualne i grafičke informacije. Kako u primeru nije bilo prostora za uključivanje animacija, one se ne pojavljuju u strukturi.

Program u jeziku *Less* za formiranje i popunjavnje strukture autorskog sistema za zadatu lekciju sledi.

Zbog dužine primera dat je samo sekvencijalni način popunjavanja lekcije. Ciklični način popunjavanja bi se realizovao slično kao kod primera HyperText-a.

BLOCK CLASSES

```
Student = CLASS {
                ident :      STRING;
                success :    LIST OF STopic;
               };
LInfo = CLASS {
                CLASS Info EXCL key, sou ;
               };
LTopic = CLASS {
                ident :      STRING;
                CLASS LInfo EXCL ident;
                query :      LIST OF LTask;
               };
LTask = CLASS {
                ident :      STRING;
                CLASS LInfo EXCL ident;
                aldSlvd :    LOGICAL;
                disp :       LOGICAL;
                dispLT :     PUBLIC PROCEDURE;
                {
                    CASE type : (Test, Prompt, Pair, Bool, Exec) OF
                    Prompt :
                    {
                        correct :    STRING;
                        answer :     STRING;
                        isCorr :     PRIVATE PROCEDURE;
                        ifCorr :     CLASS Action;
                        ifNot :      CLASS Action;
                    };
                    Test :
                    {
                        answer :     LIST OF CLASS Alter;
                    };
```

```
                    Pair :
                    {
                        answer1 :    LIST OF TEXT;
                        answer2 :    LIST OF TEXT;
                        true :       LIST OF NUMBER;
                        isCorr :     PRIVATE PROCEDURE;
                        ifCorr :     CLASS Action;
                        ifNot :      CLASS Action;
                    };
                    Bool :
                    {
                        correct :    LOGICAL;
                        answer :     LOGICAL;
                        isCorr :     PRIVATE PROCEDURE;
                        ifCorr :     CLASS Action;
                        ifNot :      CLASS Action;
                    }
                    Exec :
                    {
                        exeFil :     STRING;
                        next :       CLASS Action;
                    }
                };
            };
Alter = CLASS {
                ident : STRING;
                CLASS Info EXCL ident, key, sou;
                Act :        CLASS Action;
                IsCorr :     LOGICAL;
        };
Action = CLASS {
                CASE Type : ( EndOfLearn, GoToTopic, GoToTask ) OF
                EndOfLearn :
                    { en : END };
                GoToTopic :
                    { top : IDENT LTopic };
                GoToTask :
                    { tsk : IDENT LTask }
            };
```

BLOCK MGRAPH

```
Lesson = CLASS {
                name :       STRING;
                lsn :        LIST OF LTopic;
```

```
        start :     LIST OF IDENT LTopic;
        stop :      LIST OF IDENT LTopic;
        currTpc :   IDENT LTopic;
        currTsk :   IDENT LTask;
        interpr :   PUBLIC PROCEDURE;
        stud :      LIST OF Student;
    };
```

BLOCK PROCEDURES

(* U ovom delu su uključene sve javne i privatne procedure koje će se koristiti u lekciji. Sve dosad izložene javne procedure se mogu uključiti u ovaj deo a privatne procedure zbog svoje raznovrsnosti mogu da budu jedna sasvim zasebna tema i ovde nije od većeg interesa zadržavanje na njima. *)

START

(* formiranje lekcije sa jedinstvenim identifikatorom FPJ - funkcionalni programski jezici *)
AssignOb(*Lesson*, 'FPJ - funkcionalni programski jezici);

(* popunjavanje multidigrafa osnovnim podacima *)

```
{
    lsn :       AssignList(ČvorLek);
    start :     AssignMul(IDENT LTopic,startL,('RPR', 'PJP'));
    stopL :     AssignMul(IDENT LTopic,stopL,('SIJ'));
    (* currTpc, currTsk i stud vezanim promenljivim se na početku kod formiranja
    dodeljuje prazna vrednost i oni se koriste u procesu prezentacije *)
    interpr :   Interpret(Int);

}
```

(* popunjavanje lekcije odgovarajućim sadržajem *)
(* formiranje prvog objekta *LTopic*, sa identifikatorom RPR - rešavanje problema pomoću računara *)
AssignOb(*LTopic*,'RPR -rešavanje problema pomoću računara');
(* pupunjavanje tog čvora odgovarajućim sadržajem *)

```
{
    tex : {
            texts :     AssignTPAS(Text (* tekst prvog članka *) EndText);
            interte :   Interpret(Display_TXT);
        };

    (* pic i ani vezane promenljive su automatski postavljene na praznu vrednost *)
```

```
        interAll :      Interpret(DisplayAll);

        query : AssignList(Zadaci1);
}
(* popunjavanje prvog zadataka za prvi članak *)
AssignOb(LTask,'č1Pit1');
{
        tex : {
                        texts :         AssignTPAS(Text Kakav je postupak ... ? EndText);

                        (* za vezanu promenljivu interte je dodeljena javna procedura kod
                        prvog objekta tog tipa i ona ne treba da se dodeljuje za ostale objekte -
                        - ova napomena važe za sve naredne objekte *);
                };

        (* za vezanu promenljivu interAll je dodeljena javna procedura kod prvog objekta tog
        tipa i ona ne treba da se dodeljuje za ostale objekte - ova napomena važi za sve
        naredne objekte *);

        aldSlvd :       FALSE;
        disp :          TRUE;                   (* zahteva se prikaz tačnog odgovora *)
        dispLT :        Interpret(DisplayQuery);
        type :          Tag(Test);
        answer :        AssignList(č1Alt1);
};
(* dodavanje prve alternative za prvi zadatak *)
AssignOb(Alter,'jed_zad');
{
        tex : {
                        texts :         AssignTPAS(Text Rasčlanjavanjem ... EndText);
                };

        act : {
                        Type : Tag(GoToTask)
                        { tsk : č1Pit2 };
                };
        isCorr : TRUE;
};
(* vezivanje prve alternative za odgovarajuću listu prvog zadatka *)
AddLast($č1Alt1,$jed_zad);
(* dodavanje druge alternative za prvi zadatak *)
AssignOb(Alter,'pri_jez');

{
        tex : {
```

```
                texts :     AssignTPAS(Text tekstualni opis na ... EndText);
            };
        act : {
                Type : Tag(GoToTopic)
                { top : $RPR };
            };
        isCorr : FALSE;
};
(* vezivanje druge alternative za odgovarajuću listu prvog zadatka *)
AddLast($č1Alt1,$pri_jez);
(* dodavanje treće alternative za prvi zadatak *)
AssignOb(Alter,'stru_imp');
{
        tex : {
                texts :     AssignTPAS(Text ... EndText);
            };
        act : {
                Type : Tag(GoToTopic)
                { top : $RPR };
            };
        isCorr : FALSE;
};
(* vezivanje druge alternative za odgovarajuću listu prvog zadatka *)
AddLast($č1Alt1,$stru_imp);
(* dodavanje prvog zadatka u listu zadataka prvog članka *)
AddLast($Zadaci1,$č1Pit1);
(* popunjavanje drugog zadataka za prvi članak *)
AssignOb(LTask,'č1Pit2');
{
        tex : {
                texts :     AssignTPAS(Text Programski jezici su ... ? EndText);
            };

        aldSlvd :   FALSE;
        disp :      FALSE;              (* ne vrši se prikaz tačnog odgovora *)
        (* vezanoj promenljivoj dispLT je kod prvog objekta ovog tipa dodeljena javna
        procedura koja ostaje važeća i za sve ostale objekte nadalje *)
        type :      Tag(Bool);
        correct :   TRUE;               (* na zadatak se odgovara sa Da *)
        (* vezana promenljiva answer je prazna i ona će se popunjavati u fazi prezentacije
        *)
        isCorr :    Interpret(Tačno_č1P2);
        ifCorr : {
                Type : Tag(GoToTopic)
                { top : $PJV };
```

```
                };
        ifNot : {
                    Type : Tag(GoToTopic)
                    { top : $RPR };
                };
};
(* dodavanje drugog zadatka *)
AddLast($Zadaci1,$č1Pit2);
(* dodavalje članka u odgovarajuću listu lekcije *)
AddLast($čvorLek,$RPR);
(* pošto treći i četvrti čvor imaju iste tipove zadataka koji su do sada obradjivani neće se
navoditi njihov način popunjavanja jer je sličan prethodnim *)
(* popunjavanje četvrtog članka *)
AssignOb(LTopic,'PJP - podela po primeni programskih jezika');

(* pupunjavanje tog čvora odgovarajućim sadržajem *)

{
        tex : {
                    texts :        AssignTPAS(Text (*tekst prvog članka *) EndText);
              };
        query : AssignList(Zadaci4);
}

(* popunjavanje prvog zadataka četvrtog članka *)

AssignOb(LTask,'č4Pit1');
{
        tex : {
                    texts :        AssignTPAS(Text ... ? EndText);
              };

        aldSlvd :    FALSE;
        disp :       TRUE;                    (* zahteva se prikaz tačnog odgovora *)
        type :       Tag(Test);
        answer :     AssignList('č4Alt1');
};

(* dodavanje prve alternative za prvi zadatak *)
AssignOb(Alter,'mnogo');

{
        tex : {
                    texts :        AssignTPAS(Text ... EndText);
              };
```

```
        act : {
                        Type : Tag(GoToTask)
                        { tsk : $č4Pit2 };
                };
        isCorr : TRUE;
};
(* vezivanje prve alternative za odgovarajuću listu prvog zadatka *)
AddLast($č4Alt1,$mnogo);
(* dodavanje druge alternative za prvi zadatak *)
AssignOb(Alter,'pet');

{
        tex : {
                        texts :         AssignTPAS(Text ... EndText);
                };
        act : {
                        Type : Tag(GoToTopic)
                        { top : $PJV };
                };
        isCorr : FALSE;
};
(* vezivanje druge alternative za odgovarajuću listu prvog zadatka *)
AddLast($č4Alt1,$pet);
(* dodavanje treće alternative za prvi zadatak *)
AssignOb(Alter,'prim');

{
        tex : {
                        texts :         AssignTPAS(Text ... EndText);
                };
        act : {
                        Type : Tag(GoToTask)
                        { tsk : $č4Pit2 };
                };
        isCorr : FALSE;
};
(* vezivanje treće alternative za odgovarajuću listu prvog zadatka *)
AddLast($č4Alt1,$prim);
(* dodavanje prvog pitanja *)
AddLast($Zadaci4,$č4Pit1);
(* popunjavanje drugog zadataka četvrtog članka *)
AssignOb(LTask,'č4Pit2');
{
        tex : {
                        texts :         AssignTPAS(Text ... ? EndText);
```

```
        };

    aldSlvd :    FALSE;
    disp :       TRUE;                    (* zahteva se prikaz tačnog odgovora *)
    type :       Tag(Pair);
    answer1 :    AssignMul(TEXT,(  'oblast matematike i tehnike', 'oblast poslovne
                                   primene', obuka u programiranju', oblast
                                   veštačke inteligencije'));
    answer2 :    AssignMul(TEXT,(  'LISP, PROLOG, ...', 'COBOL, RPG,...',
                                   'FORTH, ALGOL, ...', 'PASCAL, BASIC,
                                   ...'));
    true :       AssignMul(NUMBER,(3,2,4,1));
    ifCorr : {
                 Type : Tag(GoToTask)
                 { tsk : $č4Pit3 };
             };
    ifNot : {
                 Type : Tag(GoToTopic)
                 { top : $PJP };
        };
};
(* dodavanje drugog zadatka *)
AddLast($Zadaci4,$č4Pit2);
(* popunjavanje trećeg zadataka četvrtog članka *)
AssignOb(LTask,'č4Pit3');
{
    tex : {
                 texts :      AssignTPAS(Text ... ? EndText);
        };

    aldSlvd :    FALSE;
    disp :       TRUE;                    (* zahteva se prikaz tačnog odgovora *)
    type :       Tag(Prompt);
    correct :    AssStr('proceduralne, deklarativne');
    isCorr :     Interpret(promptCorr4);
    ifCorr : {
                 Type : Tag(GoToTopic)
                 { top : $FPJ };
             };
    ifNot : {
                 Type : Tag(GoToTopic)
                 { top : $DOD };
        };
};
(* dodavanje trećeg zadatka *)
```

```
AddLast($Zadaci4,$č4Pit3);
(* dodavanje članka *)
AddLast($čvorLek,$PJP);
(* popunjavanje petog članka *)
AssignOb(LTopic,'FPJ - funkcionalni programski jezici');
(* pupunjavanje tog čvora odgovarajućim sadržajem *)

{
      tex : {
                  texts :      AssignTPAS(Text (*tekst prvog članka *) EndText);
            };
      pic : {
                  pictures :   AssignPAS(Slika1, Slika2);
                  (* imena datoteka u kojima se nalaze grafički prikazi vezani za ovaj
                  članak *)
                  interpic :   Interpret(Display_PIC);
            };

      ...
};

(* popunjavanje ostalih objekata vezanih za ovaj članak. Takodje, pošto su svi tipovi do sada
bili obradjivani nema potrebe za detaljnijim prikazom *)
(* dodavanje petog članka u lekciju *)
AddLast($čvorLek,$FPJ);
(* dodavanje sedmog članka *)
Assign(LTopic,'SIJ - slabost imperativnih programskih jezika');

{
      tex : {
                  texts :      AssignTPAS(Text (* tekst sedmog članka *) EndText);
            };
      query : AssignList('Zadaci7');
}

(* popunjavanje prvog zadataka za sedmog članak *)

AssignOb(LTask,č7Pit1);
{
      (* tex je kao i ostale vezane promenljive postavljen na prazno *)
      aldSlvd :    FALSE;
      disp :       FALSE;
      type :       Tag(Exec);
      exeFil :     AssStr('TURBO');          (* ime datoteke u kojoj se nalazi turbo
                                             pascal *)
```

```
        next : {
                        Type : Tag(EndOfLearn)
                        { en : $END };
                };
};

(* dodavanje prvog zadatka sedmog članka *)
AddFirst($Zadaci7,$č7Pit1);
(* dodavanje sedmog članka u lekciju *)
AddLast($čvorLek, $SIJ);
...
STOP.
```

5.3 Mogućnosti razvoja inteligentnih sistema za obuku

Kao što je već napomenuto u prethodnim poglavljima, metode veštačke inteligencije se sve više koriste u različitim programskim sistemima. Medjutim, rezultati tih aktivnosti još nisu zadovoljavajući.

Takodje, bez obzira na neveliki uspeh, i dalje se i sve više nastavlja sa procesima integracije metoda veštačke inteligencije i ostalih klasičnih metoda u različitim sistemima.

Predložena struktura podataka za formiranje lekcija (deo 5.2.) i jezik za upravljanje tom strukturom predstavljaju dobru osnovi za razvoj (kvazi)inteligentnih sistema za obuku.

Ugradjivanje inteligentnih funkcija u sisteme za obuku može da se realizuje u nekoliko osnovnih pravaca.

1. Automatsko formiranje lekcije tj. odgovarajućeg multidigrafa i njegovo popunjavanje.
2. 'Intelektualizacija' samog procesa učenja tj. komunikacije učenik - računar (odnosno učenik - lekcija).
3. Razvijanje i obogaćivanje učeničkog modula.

5.3.1 Automatsko formiranje lekcija

Jedna od mogućnosti uključivanja tehnika veštačke inteligencije u sisteme za obuku je i automatsko formiranje multidigrafa lekcije. U tom slučaju morao bi u jednom modulu da postoji celokupan tekst lekcije, kao i niz odgovarajućih grafičkih prikaza.

Posao profesora bi bio da pojedine članke lekcije okarakteriše samo nekim ključnim rečima na osnovu kojih bi se 'automatski' vršila podela celokupnog teksta na pojedinačne članke. Dakle, ključni element ovog sistema bi bio automatski konstruktor članaka lekcije.

Takodje, na osnovu ključnih reči i njihovog prioriteta sistem bi mogao da ponudi i logičan tok prezentacije tj. redosled obrade i prezentacije pojedinih članaka da bi se dobila smisleno ukomponovana lekcija.

Niz grafičkih prikaza bi mogao biti okarakterisan i opisan kratkim tekstom, pa bi se na osnovu tih opisa vršila podela grafičkih prikaza po člancima.

Tako pripremljena lekcija bi se potom mogla obradjivati na sledeća dva načina:

1) Profesor može preuzeti ponudjene članke, preurediti ih po sopstvenim željama (ako nije u potpunosti zadovoljan automatskom podelom) i dalje nastaviti sa njihovom doradom. Drugim rečima, profesor za pojedinačne članke formira, pridružije zadatke i odredjuje akcije i veze medju člancima.

2) U nekoj naprednijoj varijanti, moglo bi se vršiti i 'automatsko' formiranje zadataka i odredjivanje akcija koje opredeljuju veze medju člancima. Ova aktivnost bi se mogla obaviti slično kao i za članke zadavanjem ključnih reči na osnovu kojih treba formirati zadatke.

Da bi se obavile ove aktivnosti, potrebno je da se razvije jezgro odnosno modul za semantičku obradu i eventualno komprimovanje teksta. Što bi moduo bio opštiji to bi mogao da ima veću primenu. Medjutim, s obzirom na stanje u ovoj oblasti, realizacija opšteg modula je posao budućnosti. Realnijom se čini mogućnost realizacije prve faze tj. podela celokupnog teksta na pojedinačne članke na osnovu zadatog skupa reči. Osim toga, čini se realnijom i mogućnost razvoja takvog automatskog modula ali za pojedinačne usko specijalizovane oblasti kao što su matematika, fizika ili neke druge egzaktne oblasti.

5.3.2 Inteligentna komunikacija sa autorskim sistemom

Druga mogućnost za uključivanje metoda veštačke inteligencije u predloženu strukturu autorskog sistema je intelektualizacija procesa komunikacije učenik - autorski sistem.

Ako se u globalu prihvati struktura autorskog sistema kako je to predloženo u delu 5.2, onda mogućnost uključivanja metoda veštačke inteligencije, leži u obogaćivanju i 'intelektualizaciji' privatnih procedura vezanih za svaki pojedinačni zadatak.

Tip zadatka Prompt u predloženoj strukturi se javlja u vrlo rudimentiranom obliku tako da se dozvoljavaju samo jednostavni odgovori. Svaka privatna procedura vezana za zadatak treba da omogući prihvatanje odgovora učenika i na osnovu odgovora i tačnog rešenja preuzme odgovarajuću akciju. U ove procedure bi se mogle ugraditi daleko komplikovanije i kompleksnije metode veštačke inteligencije koje bi mogle da podrže sintaksno-semantičku analizu odgovora i sa semantičkog stanovišta uporedjuju dati odgovor i tačno rešenje. Na taj način bi se čak mogla omogućiti komunikacija na prirodnom jeziku. Ostvarivanjem ove komunikacije ostali tipovi zadataka bi se mogli skoro u potpunosti isključiti iz autorskog sistema.

Sledeća faza u ovom istom smeru bi mogla da bude pružanje mogućnosti učeniku, da umesto, ili i pored odgovora, postavlja pitanja u vezi sa informacijama i znanjima u člancima lekcije.

5.3.3 Statističke funkcije u autorskom sistemu

Sledeća mogućnost 'intelektualizacije' autorskog sistema je razvijanje i obogaćivanje statističke funkcije u sistemu. Struktura predložena u delu 5.2 uključuje u sebe rudimentiranu i krajnje suženu statističku funkciju preko klase *Student*.

U predloženoj strukturi vodi se minimalna evidencija o načinu i kvalitetu savladavanja lekcije. Medjutim, ako bi se htela kompleksnija i kvalitetnija analiza procesa učenja trebalo bi ovu klasu proširiti i dopuniti raznim drugim mogućnostima. Razlikuju se uglavnom dva načina i pravca obogaćivanja ove funkcije:

1) formiranjem učeničkog modula,
2) formiranjem modula za evidentiranje slabosti i propusta u samoj lekciji.

Formiranje učeničkog modula

Osnovna funkcija ovog modula bi bila da se prati napredovanje i savladavanje niza lekcija iz odredjene oblasti. Tačnije, za svakog učenika bi se prilikom korišćenja sistema za prvu lekciju iz neke oblasti, formirao odgovarajući moduo. Za svaku narednu lekciju iz iste oblasti koristio bi se već formiran moduo, obogaćivao i menjao u slkladu sa napredovanjem učenika i savladavanjem novih lekcija. Na osnovu ovakvih modula profesor bi mogao da analizira i popravlja lekcije za nove generacije. Takodje u nekoj naprednijoj fazi, autorski sistem bi mogao na osnovu informacija iz učeničkih modula da ukazuje profesoru na slabosti u lekcijama.

Drugi pozitivni efekat ovog modula bi bio automatsko ocenjivanje. Sistem bi mogao na osnovu svih učeničkih modula da izvrši rangiranje učenika i predloži ocene.

Formiranje modula lekcije

Drugi način za dobijanje statističkih informacija je formiranje modula za samu lekciju. U takvom modulu bi se mogle pratiti i beležiti informacije o samoj lekciji kao posledica različitih procesa prezentacije tj. na osnovu višestrukog korišćenja (od strane više učenika). Na osnovu ovih informacija profesoru bi se mogle predočiti slabosti i eventualna pozitivna svojstva lekcije i na taj način bi se mogle formirati kvalitetnije i pogodnije lekcije.

Jasno je da su ukratko razmotrene mogućnosti uključivanja veštačke inteligencije u sisteme za obučavanje, još uvek previše komplikovane i kompleksne da bi se u potpunosti mogle implementirati i iskoristiti u postojećem sistemu. Medjutim, nekakvi početni koraci i manji 'zalogaji' nisu nerealni i nemogući, mada su dovoljno komplikovani i pre svega tematski se ne uklapaju u ovaj rad.

Svaka od navedenih oblasti (5.3.1, 5.3.2 i 5.3.3) predstavlja zasebne celine koje se nezavisno mogu dalje razvijati, konkretizovati i implementirati. Osim toga, njihovom integracijom bi se mogli dobiti kompletni inteligentni tutorski sistemi. Medjutim, orudja i metode veštačke inteligencije još uvek ne mogu u potpunosti da odgovore ovim zahtevima.

6 Zaključak

U prethodnim poglavljima je prikazan predlog jednog opšteg sistema za reprezentaciju i prezentaciju informacija, i odredjeno njegovo mesto i uloga u oblasti programskih jezika i sličnih sistema za reprezentaciju i prezentaciju informacija.

Predloženi opšti sistem obuhvata niz ugradjenih klasa i mogućnosti definisanja novih i kompleksnijih struktura i klasa. Predložene strukture omogućavaju reprezentaciju različitih pojavnih oblika informacija i to tekst, grafički prikazi, animacije i zvučne informacije kao i nekih standardnih tipova podataka (LOGICAL, NUMBER,...). Osim toga, strukture dozvoljavaju definisanje i ugradjivanje procedura tj. metoda za manipulisanje kako pojedinačnim elementima struktura tako i celim strukturama. Fleksibilni način definisanja novih i kompleksnijih struktura pruža mogućnost korišćenja ovog sistema u raznim oblastim gde se zahteva reprezentacija i prezentacija informacija.

Struktura klasa i način njihovog komponovanja i nasledjivanja omogućava primenu i realizaciju principa objektno orijentisane paradigme.

Polazeći od predloženog koncepta za reprezentaciju i prezentaciju informacija, razvijen je i odgovarajući jezik (*Less*) koji omogućava definisanje i popunjavanje kompleksnih baza podataka. Elementi baze podataka su objekti klasa pogodnih za reprezentaciju i prezentaciju informacija različitih pojavnih oblika, medjusobno povezanih tako da čine multidigraf nekog domena reprezentacije.

Predloženi jezik predstavlja:

- osnovu za formalizaciju sistema za reprezentaciju i prezentaciju informacija,
- apstraktni mašinski jezik za kreiranje sistema za reprezentaciju i prezentaciju informacija, koji omogućava lakšu prenostvost multimedijalnih aplikacija na druge računare i operativne sisteme,
- orudje za lakšu i jednostavniju implementaciju konkretnih aplikacija.

Predloženi opšti koncept i jezik *Less* pružaju mogućnosti široke primene:

- za razvoj HyperText sistema za različite oblasti,
- za razvoj opštih autorskih sistema koji se mogu koristiti u različitim oblastima koje podražavaju proces reprezentacije i prezentacije informacija počev od raznih kurseva pa sve do primene i korišćenja lekcija u obrazovnim procesima na svim nivoima,

- za razvoj jednostavnijih inteligentnih sistema za obuku,
- za razvoj HELP i opštih sistema za prezentaciju.

Osim mogućnosti primene predložena struktura pruža razne mogućnosti za dalji rad.

1) Pre svega predloženi sistem je potrebno konkretizovati i implementirati. Što se tiče same implementacije nameću se dve mogućnosti.

Prva mogućnost je implementacija objektno orijentisanog programskog jezika koji podržava i upravlja predloženom strukturom i obuhvata predložene direktive i komande, na nekom programskom jeziku niskog nivoa.

Druga mogućnost je realizacija predprocesora koji bi programe pisane na Less jeziku prevodio u ekvivalentne programe ne nekom višem programskom jeziku. Kako jezik Less i njegove strukture podržavaju osnovne principe objektno orijentisanog programiranja i metodologije, a takodje uključuju i procedure na paskalolikom jeziku pogodan ciljni jezik mogao bi biti MODULA-2 jezik u kom su ugradjeni osnovni principi objektno orijentisanog stila programiranja [Blaschek, 1989].

2) Ukoliko bi se sistem implementirao i realizovao otvaraju se nove mogućnosti daljeg rada.

- Razvijanje jezgra niza HyperText sistema opšte prirode u potpunosti zasnovanog na sistemu predloženom u delu 5.1.
- Razvijanje jezgra opšteg autorskog sistema za različite obrazovne procese, takodje u potpunosti zasnovanog na sistemu predloženom u delu 5.2.

3) Ukoliko su realizovane prethodne dve tačke javljaju se ponovo nove mogućnosti za dalji rad.

Jedan od pravaca je razvijanje biblioteka različitih procedura koje podržavaju proces prezentacije. Biblioteke bi obuhvatale, kako javne procedure, tako i niz pomoćnih procedura čijom kombinacijom bi se mogle dobiti razne privatne procedure. Osim toga mogle bi se realizovati i kompletne privatne procedure. Te biblioteke bi koristili profesori u procesu kreiranja lekcija.

Drugi pravac je razvijanje i ugradnja raznih metoda veštačke inteligencije i semantičke analize teksta u privatne procedure kojima se obezbedjuje prihvatanje odgovora učenika i njegova analiza.

Treći pravac može obuhvatati uvodjenje inteligentnih funkcija i u ostale delove sistema.

4) Svojstvo sistema i strukture da memoriše različite pojavne oblike informacija (tekst, grafika, animacija, zvuk) otvara mogućnosti za jednostavno uključivanje i ostalih vidova manipulacije informacijama kao što su video, CD-plejeri i slično. Na ovaj način bi se mogli formirati i u potpunosti multimedijalni sistemi za reprezentaciju i prezentaciju informacija.

Svaka od navedenih mogućnosti daljeg rada je dovoljno kompleksna da može biti tema nekog magistarskog ili doktorskog rada.

Ugradjivanje u sistem elemenata najsavremenijih i najnovijih pravaca razvoja u računarskim naukama kao što su multimedijalnost, objektno orijentisana paradigma, prezentacija informacija, čini ga zanimljivim i pogodnim za dalje razvijanje, dogradjivanje i korišćenje.

Literatura

1. M. R. Blaha, W. J. Premerlani, J. E. Rumbaugh, [1988], *Relational Database Design Using an Object Oriented Methodology*, CACM, Vol. 31., No. 4., April 1988., pp. 414-427.

2. G. Blaschek, *Implementation of Objects in Modula-2*, [1989], Structured Programming Vol. 10., No3., 1989., pp 147-155.

3. T. Bloom and S.B. Zdonik, [1987], *Issues in the Design of Object-Oriented Database Programming Languages*, OOPSLA '87 Proceedings, October 4-8, 1987. pp. 441-451.

4. D. G. Bobrow and T. Winograd, [1976], *An overview of KRL, A knowledge representation language*, Stanford Artificial Intelligence Laboratory, Memo AIM-293, Report No. STAN-CS-76-581, November 1976.

5. Z. Budimac, Lj. Jerinić, **M. Ivanović**, and Đ. Paunić, [1988a], *Educational Package from Novi Sad*, Svet kompjutera, january (1988), 20, (in Serbian).

6. Z. Budimac, Lj. Jerinić, **M. Ivanović** and Đ. Paunić, [1988b], *OSOF for students*, Svet kompjutera, february (1988), 21 - 22, (in Serbian).

7. Z. Budimac, and **M. Ivanović**, [1989a], *New Data Type in Pascal*, Proceedings of the DECUS Europe Symposium (The Hague, Holland), 1989, 193 - 199.

8. Z. Budimac, Đ. Paunić, **M. Ivanović**, and Lj. Jerinić, [1989b], *The Structure of Educational Software in Mathematics*, Proceedings of XIII Information Technologies Conference "Sarajevo-Jahorina 89" (Sarajevo, Yugoslavia), 1989, 250-1 - 250-4, (in Serbian).

9. Z. Budimac, Z. Putnik, **M. Ivanović**, Đ. Paunić and Lj. Jerinić, [1990], *General Purpose Educational Software with "Simulation" Ability - OSOF*, Proceedings of XII International Symposium "Computer at the University" (Cavtat, Yugoslavia), 1990, S1.5.1 - S1.5.6.

10. Z. Budimac, **M. Ivanović**, Z. Putnik, D. Tošić, [1991], *Lisp kroz primere*, Institute of mathematics, Novi Sad, 1991.

11. Z. Budimac and **M. Ivanović**, [1992], *On Specialized Language for Informatin (Re)presentation*, Proceedings of DECSYM '92, Side-Antalya, Turkye, 1992, 175-187.

12. P. Carando, [1989], *SHADOW - Fusing Hypertext with AI*, IEEE EXPERT, WINTER 1989., pp. 65-78.

13. G.P. Faconi and F. Paterno, [1990], *A Graphical syntax of ECSP construct to describe human-computer interaction*, Automatika 31(1990) 1-2, pp. A.357-A.367.

14. E. A. Feigenbaum, [1977], *The art og artificial intelligence: I Themes and case studies of knowledge engineering*, Stanford Heuristic Programming Project, Memo HPP-77-25, Report No. STAN-CS-77-621.

15. J. Fiderio, [1988], *A Grand Vision*, BYTE, October 1988., pp. 237-244.

16. Y. Freudlich and D. General, [1990], *Knowledge Basis and Databases-Converging Technologies, Diverging Interests*, IEEE COMPUTER, November 1990, pp 51-57.

17. M. Frisse, [1988], *From Text to Hypertext*, BYTE, October 1988., pp. 247-253.

18. M. Grossman and R.K. Ege, [1987], *Logical Composition of Object Oriented Interfaces*, OOPSLA '87 Proceedings, October 4-8, 1987, pp 295-306.

19. P. Helman and R. Veroff, [1988], *Walls and mirrors, Intermediate problem solving and data structures, Modula-2 edition*, The Benjamin/Cummings Publishing Company, 1988.

20. J. A. Hewitt and R. J. Frank, [1989], *Software Engineering in Modula-2, An Object Oriented Approach* , MacMillan Education LTD, 1989.

21. **M. Ivanović** and Z. Budimac, [1986], *The Grammar of Typed Pascal Expressions*, Informatika 20 (1986) 3, 153 - 160, (in Serbian).

22. **M. Ivanović**, [1988a], *An Implementation of Parser Based on Simple Precedence Grammars*, Master thesis, University of Novi Sad, march 1988, 125 pages, (in Serbian).

23. **M. Ivanović** and Z. Budimac, [1988b], *Automatic Translation of EBNF Grammar Rules into BNF ones*, Informatica 12 (1988) 4, 49 - 54, (in Serbian).

24. **M. Ivanović** and Z. Budimac, [1990a], *Involving Coroutines in Interaction between Functional and Conventional Language*, ACM SIGPLAN Notices 25 (1990) 11, 65 - 74.

25. **M. Ivanović**, [1990b], *Complete Syntax Definitions of Pascal Statements and Calls of Standard Procedures READ and WRITE*, Informatica 14 (1990) 1, 70 - 73.

26. **M. Ivanović**, Đ. Paunić and Z. Budimac, [1991a], *A Tool for General CAI Lesson Creation*, Proceedings of XVI International Summer School "Programming '91" (Sofia, Bulgaria), 1991, 107-110.

27. **M. Ivanović**, Đ. Paunić i Z. Budimac, [1991b], *(Re)prezentacija informacija u obrazovnom procesu*, Zbornik radova simpozija "Informatika u obrazovanju i nove obrazovne tehnologije", Novi Sad, 1991, 38-1 - 38-9.

28. Lj. Jerinić, Đ. Paunić, Z. Budimac and **M. Ivanović**, [1988a], *A Universal Programming Package for use of Computers in Learning and Testing*, Proceedings of X International Symposium "Computer at the University" (Cavtat, Yugoslavia), 1988, 1.13.1 - 1.13.4, (in Serbian).

29. Lj. Jerinić, Z. Budimac, **M. Ivanović** and Đ. Paunić, [1988b], *Artificial Intelligence in School*, YU 21, 8(1988), 46 - 48, (in Serbian).

30. Lj. Jerinić, Đ. Paunić, Z. Budimac and **M. Ivanović**, [1989a], *The Development and Use Educational Software in Autonomous Province Vojvodina*, Proceedings of XIII Information Technologies Conference "Sarajevo-Jahorina 89" (Sarajevo, Yugoslavia), 1989, 230-1 - 230-5, (in Serbian).

31. Lj. Jerinić, Z. Budimac, Đ. Paunić and **M. Ivanović**, [1989b], *An Application of Knowledge Engineering Methods in Computer Aided Instruction*, Informatica 13 (1989) 4, 69 - 71, (in Serbian).

32. D. D. Kary and P. L. Juell, [1986], *TRC : An Expert System Compiler*, SIGPLAN Notices, Vol. 21., No. 5., May 1986., pp 64-68.

33. E. Klajn, [1990], *An advanced framework for graphic systems standardization*, Automatika 31(1990) 1-2, pp. A.353-A.362.

34. P.S. Langston, [1990], *Little languages for music*, Computing Systems USENIX, Vol. 3. No. 2, spring 1990, pp.193-288.

35. M. E. S. Loomis, A.V. Shah and J. E. Rumbaugh, [1987], *An object modeling technique for conceptual Design*, Proceedings of European Conference on Object Oriented Programming, Paris, June, 1987.

36. H. Maurer and R. Stubenrauch, [1987], *A General Lesson Specification System*, Institute fur Information-Sverarbeitung, Report 241, September 1987, Graz.

37. B. Meyer, [1988], *Object-Oriented Sofrware Construction*, Prentice Hall, 1988.

38. J. Nielsen, [1990], *The Art of Navigation through Hypertext*, Communications of the ACM, March 1990., pp. 297-310.

39. S. Osuga, [1989], *Obrabotka znanij*, MIR, Moskva, 1989.

40. S. Osuga and Ju. Saeki, [1990], *Preobretenie znanij*, MIR, Moskva 1990.

41. Đ. Paunić, Lj. Jerinić, Z. Budimac and **M. Ivanović**, [1988], *A Universal Programming Package for Use of Computers in Education*, Proceedings of X International Symposium "Computer at the University" (Cavtat, Yugoslavia), 1988, 1.12.1 - 1.12.4, (in Serbian).

42. W.J. Premerlani, M.R. Blaha, J.E. Rumbaugh and T.A. Vatwig, [1990], *An Object-Oriented Relational Database*, CACM, Vol 333, No. 11, November 1990, pp 99-109.

43. Z. Putnik, Z. Budimac, **M. Ivanović**, Đ. Paunić and Lj. Jerinić, [1990], *Using Pictures in General Purpose Educational Software - OSOF*, Proceedings of XII International Symposium "Computer at the University" (Cavtat, Yugoslavia), 1990, 1.3.1 - 1.3.6.

44. M. Rees and D. Robson, [1988], *Practical Compiling with Pascal-S*, Addison-Wesley, 1988.

45. V. Š. Rubaškin, [1989], *Predstavlenie i analiz smisla v intellektualjnih informacionnih sistemah*, Nauka, Moskva 1989.

46. J. Rumbaugh, [1987], *Relations as semantic constructs in an object oriented language*, OOPSLA '87 Proceedings, October 4-8, 1987.

47. A. V. Shah, J. Rumbaugh et all, [1989], *DSM : An Object Relationship Modeling Language*, OOPSLA '89 Proceedings, New Orleans, October 1989.

48. R. F. Sincovec and R. S. Wiener, [1986], *Data structures using Modula-2*, John Wiley & Sons, 1986.

49. K.E. Smith and S.B. Zdonik, [1987], *Intermedia: A Case Study of the Differences Between Relational and Object-Oriented Database Systems*, OOPSLA '87 Proceedings, October 4-8, 1987. pp. 452-465.

50. T. Thompson, [1990], *Keynote - A Language and Exstensible Graphic Editor for Music*, Computing Systems USENIX, Vol. 3. No. 2, spring 1990, pp.289 - 330.

51. Ž. Turk, [1990], *Object Oriented Graphics*, Automatika 31(1990) 1-2, pp. A.239-A.246.

52. H. Ueno and M. Isidzuka, [1989], *Predstavlenie i ispolzovanie znanij*, MIR, Moskva 1989.

53. D. Ungar and R.B. Smith, [1987], *Self: The Power of Simplicity*, OOPSLA '87 Proceedings, October 4-8, 1987. pp. 227-242.

54. M.D. Veljkov, [1990], *Managing Multimedia*, BYTE August 1990., pp. 227-232.

55. W. Wahlster, [1990], *Knowlwdge-Based Information Presentation*, CAS '90, Dubrovnik, September 1990.

56. P. Wegner, [1990], *Concepts and Paradigms of Object Oriented Programming*, OOPS MESSENGER, ACM Press, Vol. 1., No. 1., August 1990., pp. 7-87.

57. T. Winograd, [1976], *Towares a procedural understanding of semantics*, Stanford Artificial Intelligence Laboratory, Memo AIM-292, Report No. STAN-CS-76-580, November 1976.

58. Kim Won, [1990], *Object-Oriented Databases: Definition and Research Directions*, IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 3, September 1990., pp. 327-341.

59. P.H. Wood, [1990], *Intelligent Tutoring Systems: An Annotated Bibliography*, SIGART BILLETIN,Vol. 1., No. 1.,January/April 1990., pp. 21-41.