5 - UML

„Ti, koji me čitaš,
jesi li siguran da razumeš moj jezik?“
Horhe Luis Borhes, Vavilonska biblioteka

5.1 Objedinjeni jezik za modeliranje

Objedinjeni jezik za modeliranje (engl. *Unified Modeling Language – UML*) predstavlja jednu od najvažnijih savremenih razvojnih tehnologija. Kao što je već naglašeno u poglavlju *3.3 – Objektno-orijentisane metodologije*, *UML* je nastao uglavnom u periodu od 1995. do 2000. godine, nakon čega se neprekidno dalje unapređuje. Uticaj koji je *UML* ostvario na industriju razvoja softvera može da se meri sa uticajem strukturnog programiranja, objektno-orijentisanog programiranja ili relacionih baza podataka – sve ove tehnologije su danas praktično svuda prisutne i bez njih je nezamisliv razvoj softvera.

Održavanjem standarda *UML*-a se bavi *OMG*. Na njihovoj veb lokaciji može da se pronađe zvanična dokumentacija o standardu, kao i mnogo drugih izvora informacija o *UML*-u [*OMG*] [*UML*]. Zanimljivo je da se kao jezik za predstavljanje apstraktne sintakse i strukture *UML*-a koristi upravo *UML*.

Iscrpno predstavljanje *UML*-a bi zahtevalo mnogo više prostora nego što nam je ovde na raspolaganju. Zbog toga ćemo da napravimo osvrt na osnovne elemente jezika, dok ćemo se temeljnije posvetiti samo nekim vrstama dijagrama. Na srpski jezik je prevedeno nekoliko knjiga o *UML*-u. Iako one često nisu najsvežije (uglavnom su prevođene knjige sa početka 2000-ih godina, koje se bave prvim

verzijama jezika), sasvim su dobre za početno upoznavanje *UML*-a i toplo ih preporučujemo [*Booch 1999*][19] [*Fowler 2003a*].

UML je prvenstveno grafički jezik. Najveći deo elemenata jezika se odnosi na različite dijagramske tehnike, dok se tekstom samo detaljnije opisuju ili komentarišu određeni elementi. Samo pojedini delovi jezika se više oslanjaju na tekstualne nego na grafičke opise elemenata modela.

Svi dijagrami *UML*-a mogu da se podele na:

- strukturne dijagrame i
- dijagrame ponašanja.

U nastavku ovog poglavlja ćemo da ukratko predstavimo vrste dijagrama i da malo temeljnije opišemo one koje se koriste u ovoj knjizi.

Strukturni dijagrami

Strukturni dijagrami opisuju strukturu softverskog sistema i njegovih delova. Različiti dijagrami mogu da imaju različit nivo apstraktnosti i mogu da odgovaraju različitim fazama modeliranja ili implementacije softvera. Elementi strukturnih dijagrama nemaju dodira sa konceptom protoka vremena ili povezanim konceptima, kao što su tok komunikacije među objektima ili promene stanja objekata.

U strukturne dijagrame spadaju:

- dijagram klasa (engl. *class diagram*);
- dijagram objekata (engl. *object diagram*);
- dijagram paketa (engl. *package diagram*);
- dijagram složene strukture (engl. *composite structure diagram*);
- dijagram komponenti (engl. *component diagram*);
- dijagram raspoređivanja (engl. *deployment diagram*, naziva se i *dijagram isporučivanja*) i
- dijagram profila (engl. *profile diagram*).

Dijagram klasa je jedan od najvažnijih strukturnih dijagrama. On opisuje elemente statičkog modela softvera, a pre svega klase, njihovu strukturu i

[19] Postoji novije izdanje [*Booch 2005*] koje pokriva *UML* 2.0, ali ono nije prevedeno na srpski jezik do trenutka pisanja ove knjige.

međusobne odnose. Dijagram klasa predstavlja klase kao osnovne statičke elemente sistema.

Dijagram objekata nam pomaže da bolje razumemo dinamičku prirodu odnosa među objektima. Koristi se kao dopuna dijagrama klasa i komunikacije, za opisivanje dinamičkih sistema. Dijagram objekata predstavlja izabrani skup objekata, koji svojim sadržajem i odnosima ilustruje primer jedne konkretne slike stanja sistema u nekom izabranom trenutku. Sadrži nazive objekata i njihovih klasa, imena i vrednosti atributa i odnose među objektima.

Dijagram paketa opisuje kako su elementi logičkog modela organizovani u pakete, kao i međuzavisnosti paketa. U savremenim OO programskim jezicima, *paket* je često sinonim za *prostor imena*. Paket okuplja elemente (klase, komponente, slučajeve upotrebe i sl.) koji su semantički povezani i očekuje se da se zajedno menjaju. Dijagram sadrži nazive i granice paketa i međusobne zavisnosti paketa. Može da sadrži i klase u paketima, kao i njihove međusobne odnose, predstavljene na isti način kao što se predstavljaju u dijagramima klasa.

Dijagram složene strukture ima za cilj da bliže objasni unutrašnju strukturu elemenata neke strukture (komponente ili glavne klase), predstavljanjem njenih sastavnih delova i njihovih međusobnih odnosa.

Dijagram komponenti opisuje komponente koje čine aplikaciju, podsistem ili veću komponentu. Za razliku od dijagrama paketa, koji predstavlja logičku dekompoziciju softverskog sistema, dijagram komponenti predstavlja funkcionalnu dekompoziciju sistema. Sadrži nazive komponenti, njihove javne interfejse i međusobne odnose. Svaka komponenta bi trebalo da ima jasno prepoznatu i oblikovanu funkciju, koju druge komponente koriste putem njenog interfejsa.

Dijagram raspoređivanja predstavlja elemente fizičke arhitekture softverskog sistema i način raspoređivanja softverskih komponenti na te fizičke elemente. Sadrži čvorove (servere, klijente i druge uređaje), softverske i hardverske podsisteme ili komponente, linije komunikacije među uređajima i međusobne veze podsistema. Na svakom od čvorova se predstavljaju softverske komponente koje će njima raditi, kao i osnovne linije komunikacije (odnosi) među njima.

Dijagram profila predstavlja osnovu za prilagođavanje *UML* dijagrama specifičnim domenima problema, dodavanjem novih vrsta elemenata, pravljenjem novih svojstava i određivanjem nove specifične semantike. Počiva na primeni tri mehanizma za proširivanje: stereotipovima, označenim vrednostima i uslovnim ograničenjima. Dijagramima profila se definišu nove vrste stereotipova, metaklasa i slično.

Dijagrami ponašanja

Dijagrami ponašanja se bave dinamičkom prirodom softverskog sistema. Oni opisuju ponašanje pojedinačnih objekata ili podsistema. Ponašanje se predstavlja u

kontekstu toka vremena, u obliku razmene poruka, toka komunikacije ili toka promene stanja.

Dijagrami interakcije su posebna podgrupa dijagrama ponašanja, ali se ponekad razmatraju posebno. Razlikuju se od ostalih dijagrama ponašanja po tome što je na njima vremenski tok opisan eksplicitnije nego na ostalim dijagramima.

U dijagrame ponašanja spadaju:

- dijagram slučajeva upotrebe (engl. *use case diagram*);
- dijagram aktivnosti (engl. *activity diagram*);
- dijagram stanja (engl. *state machine diagram*) i
- dijagrami interakcije:
  - dijagram sekvence (engl. *sequence diagram*);
  - dijagram komunikacije (engl. *communication diagram*, do verzije 2 je bio u upotrebi naziv *dijagram saradnje*, engl. *collaboration diagram*);
  - dijagram vremena (engl. *timing diagram*) i
  - pregledni dijagram interakcija (engl. *interaction overview diagram*).

Dijagram slučajeva upotrebe predstavlja slučajeve upotrebe softverskog sistema, aktere (učesnike) koji u njima učestvuju i njihove međusobne odnose. Slučajevi upotrebe mogu da budu organizovani u pakete ili podsisteme, pa ovi dijagrami mogu da sadrže i njihove međusobne odnose. Za razliku od ostalih dijagrama, svaki slučaj upotrebe mora da bude propraćen opsežnom tekstualnom dokumentacijom. Dijagram je samo okvirna ilustracija čiji cilj je da predstavi ključne elemente i odnose među njima, dok se sve važne informacije o pojedinačnim slučajevima upotrebe opisuju tekstualno.

Dijagram aktivnosti opisuje poslovne procese višeg nivoa i tokove podataka a može da predstavlja i složene logičke elemente sistema. Sadrži procese, tokove podataka između procesa, grananja i čvorišta, uslovne tačke i početne i završne tačke. Može da bude grupisan po trakama aktera, tako da se vidi ko je od subjekata zadužen za koju aktivnost. Ima sličnosti sa klasičnim dijagramskim tehnikama za opisivanje algoritama.

Dijagram stanja opisuje kako se stanje jednog objekta menja u zavisnosti od interakcija u koje objekat ulazi tokom svog života. Dijagram stanja sadrži sva stanja u kojima posmatrani objekat može da bude (pri čemu su posebno označena početno i završno stanje), zatim sve moguće prelaske iz jednog u drugo stanje, kao i nazive događaja koji menjaju stanje objekata.

Dijagram sekvence služi za preciznije opisivanje toka odvijanja slučajeva upotrebe, kao i za jasno prepoznavanje odgovornosti subjekata i objekata za

pojedinačne korake. Sadrži slučajeve upotrebe, aktere, pakete, podsisteme, poruke i međusobne odnose. Dijagram sekvence često može da se neposredno prevede u implementaciju metoda ili skupa metoda.

Dijagram komunikacije ilustruje objekte, njihove međusobne odnose i poruke koje razmenjuju. Posvećuje se posebna pažnja strukturnoj organizaciji objekata koji učestvuju u razmeni poruka. Sadrži objekte (među kojima mogu da se nađu i subjekti), poruke koje oni razmenjuju, komentare i napomene.

Dijagram vremena predstavlja promene stanja objekata tokom vremena. koristi se za opisivanje zavisnosti promena stanja od spoljašnjih događaja. Sadrži protok vremena, spoljašnje događaje i promene stanja objekata.

Pregledni dijagram interakcija je varijanta dijagrama aktivnosti u kojoj je akcenat na upravljanju procesima ili sistemom. Svaki čvor/aktivnost u dijagramu može da predstavlja neki drugi dijagram interakcija ili aktivnosti. Sadrži objekte, manje dijagrame aktivnosti ili interakcija, slučajeve upotrebe, tok odvijanja procesa (protoka podataka), grananja i spajanja, početak i kraj.

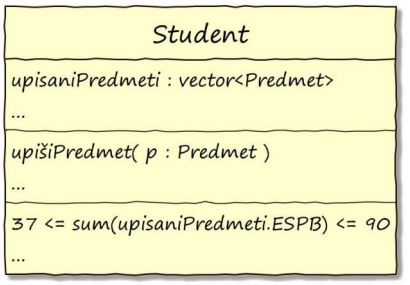
5.2 Dijagram klasa

Dijagram klasa je jedan od najčešće korišćenih dijagrama *UML*-a. Njegova osnovna namena je da opiše strukturu statičkog modela softvera – strukturu klasa i njihove međusobne odnose. Dijagram klasa je postao osnovna tehnika kojom se dokumentuju poslovi koje treba implementirati ali i ostvareni rezultati rada.

Slika 2 – Primer predstavljanja klase na dijagramu klasa

Osnovni elementi dijagrama su *klase*, koje se predstavljaju pravougaonicima. Klasa može da bude podeljena horizontalnim linijama na odeljke – prvi odeljak sadrži naziv klase, drugi sadrži spisak atributa, a treći sadrži spisak metoda klase. Vidljivost atributa i metoda se može predstavljati na različite načine, ali je uobičajeno da se privatni članovi označavaju znakom „– “, javni znakom „+“, a zaštićeni znakom „#“. Dopušteno je da umesto toga postoji neka jasna grafička notacija. U primeru (*Slika 2*) je vidljivost opisana bojama i oblicima – crveni kvadrat označava

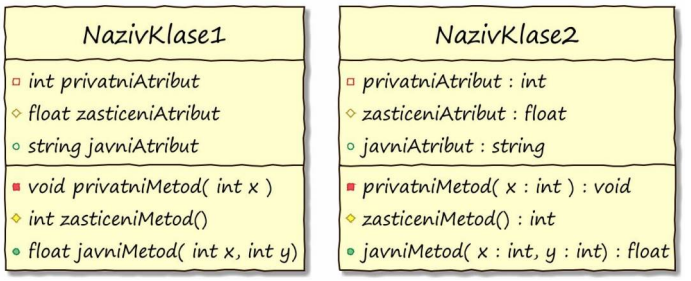
privatne, zeleni krug javne, a žuti romb zaštićene elemente. Pritom su atributi označeni praznim a metodi popunjenim oznakama.



Slika 3 – Primer predstavljanja ograničenja na dijagramu klasa

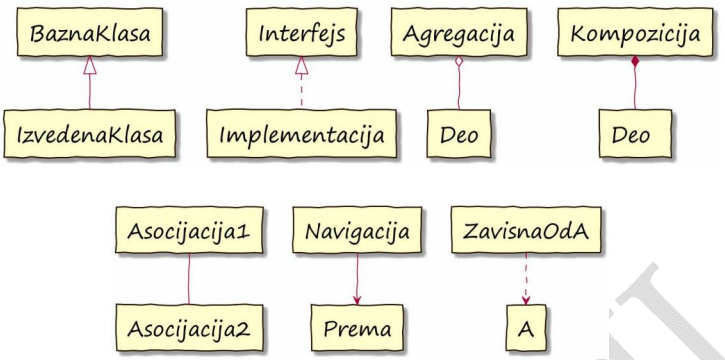
U zavisnosti od vrste modela koji se predstavlja dijagramom klasa, neki elementi strukture klasa mogu da se izostavljaju. Tako se, na primer, u dokumentaciji koja služi korisnicima nekog dela softvera, u klasama navode samo elementi interfejsa, tj. javni metodi, dok se sve ostalo sakriva da ne bi skretalo pažnju korisnika. Ako je potrebno da stanje klase ili način upotrebe klase poštuju neka posebna pravila ili ograničenja (engl. *constraints*) onda može da se doda i četvrti deo opisa klase, u kome se navode ta ograničenja (*Slika 3*). Ograničenja i dodatni uslovi mogu da se zapisuju rečima ili formulama.

Tipovi atributa i metoda mogu da se navode, ali ne moraju. Ako se navode, onda se obično navode u skladu sa sintaksom izabranog programskog jezika ili tako da se tipovi navode iza imena, odvojeni simbolom „:“ (*Slika 4*).



Slika 4 – Dva načina predstavljanja tipova atributa i metoda na dijagramu klasa

Navođenjem tipova atributa i metoda se dijagram klasa dodatno opterećuje informacijama koje mogu da se vide i u tekstualnom opisu, a mogu da otežavaju razumevanje dijagrama. Zbog toga se, u slučaju kada na dijagramima imamo mnogo klasa i želimo da pre svega istaknemo njihove međusobne odnose, često odlučujemo da ne predstavljamo tipove, a ponekad čak ni interfejse, već da opise klasa svedemo samo na nazive (*Slika 5*).



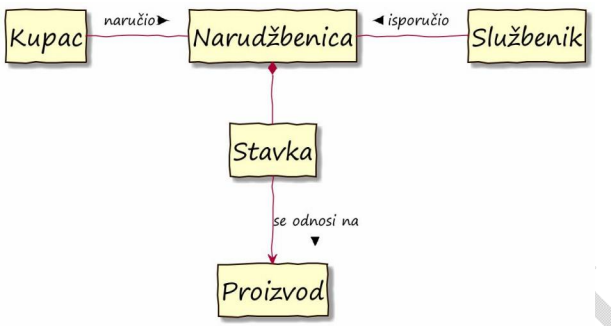
Slika 5 – Predstavljanje odnosa među klasama

Odnosi među klasama se predstavljaju linijama, koje na krajevima mogu da imaju određene simbole:

- nasleđivanje se predstavlja linijom koja se na strani bazne klase (tj. na strani *generalizacije*) završava strelicom u obliku praznog trougla;
- implementacija interfejsa se predstavlja slično nasleđivanju, ali sa isprekidanom linijom;
- agregacija se predstavlja linijom koja se na strani klase koja sadrži delove završava praznim rombom;
- kompozicija se predstavlja linijom koja se na strani klase koja sadrži delove završava popunjenim rombom;
- asocijacija nema nikakve simbole na krajevima i
- zavisnost se predstavlja isprekidanom linijom.

Agregacija predstavlja odnos dveju klasa u kome objekat jedne klase (tzv. *agregacija* ili *složeni objekat*) sadrži referencu na objekat druge klase (tzv. *deo*), ali je odnos takve prirode da delovi mogu da postoje i nezavisno od objekta koji ih okuplja – ako se uništi složeni objekat to ne znači da će biti uništeni i njegovi sastavni delovi. Zato se kaže da agregacija predstavlja *slabu integraciju* delova u celinu. Primer agregacije je povezivanje automobila i točkova – točkovi jesu delovi automobila, ali mogu da postoje i bez njega.

Nasuprot tome, u slučaju kompozicije, podrazumeva se da se radi o tzv. *jakoj integraciji* – ako se uništi složeni objekat, automatski će se uništiti i njegovi delovi. Štaviše, objekti koji predstavljaju delove kompozicije ne mogu ni da se naprave a da već tom prilikom ne bude jasno određeno kom složenom objektu pripadaju. Primer kompozicije je odnos narudžbenice i stavki narudžbenice – stavke narudžbenice ne mogu da postoje bez narudžbenice kojoj pripadaju.



Slika 6 – Dijagram klasa za rad sa narudžbenicama

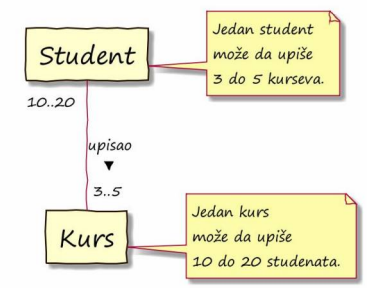
Najopštiji odnosi, kod kojih postoji neposredna veza između objekata, nazivaju se asocijacijama. Asocijacije znače da su neki konkretni objekti međusobno povezani, ali da ta veza može biti ostvarena na više načina. Ako jedan od objekata sadrži referencu na drugi objekat i može da se praćenjem te reference dođe do drugog objekta, ali obrnuto ne važi, onda je u pitanju jednosmerna asocijacija, koja se označava strelicom u smeru u kome se može ostvariti pristupanje objektu, tzv. *navigacija*. Ako svaki od dva asocirana objekta sadrži referencu na drugi objekat (primetimo da je to redundantno i da se ne preporučuje, ali se ipak dešava), ili postoji neki spoljašnji mehanizam za pronalaženje povezanih objekata (na primer katalog parova i sl.), onda kažemo da je asocijacija dvosmerna i ne označavamo je na poseban način.

Ako se među objektima klasa ne uspostavlja nikakva trajna povezanost, ali jedna klasa mora da zna za drugu klasu, na primer zato što se ona koristi u implementaciji njenih metoda, onda je to najslabiji oblik povezanosti, koji se naziva *zavisnost* klasa. Zavisnost klasa se označava isprekidanom linijom, a u slučaju jednosmerne zavisnosti i strelicom na kraju linije.

Svaki odnos može da se označi imenom. Ime odnosa se navodi što bliže sredini linije. Pored imena može da se navede i koju ulogu u odnosu ima koji od učesnika, što se označava navođenjem kratkog naziva uz liniju bliže klasi na koju se odnosi. Da se dijagram ne bi suviše vizualno opterećivao, alternativa je da se navodi samo predikat koji opisuje odnos subjekta i objekta, pa se onda navodi i strelica u obliku trougla, kojom se označava smer čitanja od subjekta prema objektu (*Slika 6*).

Važan element opisa odnosa je i kardinalnost, tj. broj objekata koji mogu da učestvuju u odnosu. Broj koji se navodi bliže jednoj klasi označava koliko objekata te klase može da učestvuje u datom odnosu sa jednim objektom druge klase (*Slika 7*).

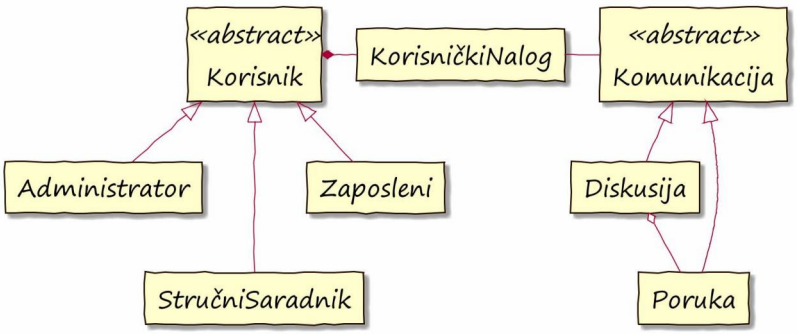
Dodatne zabeleške (poput komentara i detaljnijih opisa) mogu da se navode u pravougaonicima koji se povezuju sa elementima na koje se odnose. Komentar se povezuje isprekidanom linijom sa elementom na koji se odnose (najčešće je to klasa, ali može da bude u pitanju i neki njen deo ili odnos između dve klase), ali se često koriste i alternativne vizualne reprezentacije (*Slika 7*).



Slika 7 – Opisivanje kardinalnosti odnosa *upisanKurs*

Za bliže označavanje prirode ili namene neke klase uobičajena je upotreba tzv. *stereotipova* (*Slika 8*, *Slika 11*). Stereotip se navodi u obliku „<< *naziv stereotipa* >>“ ispred naziva klase. Neki od uobičajenih stereotipova su:

- *abstract* – klasa je apstraktna;
- *auxiliary* – u pitanju je pomoćna klasa, koju korisnici obično ne vide;
- *entity* – klasa predstavlja trajni podatak, obično zapisan u bazi podataka;
- *enumeration* – ne radi se o pravoj klasi već o novom tipu čije se dopuštene vrednosti navode (ili opisuju) u delu namenjenom za atribute;
- *focus* – u pitanju je najvažnija klasa na dijagramu; ako dijagram predstavlja implementaciju neke komponente, ovako je obično označena klasa koja implementira interfejs ili poslovnu logiku komponente;
- *interface* – u pitanju je specifikacija interfejsa koji neke druge klase implementiraju.

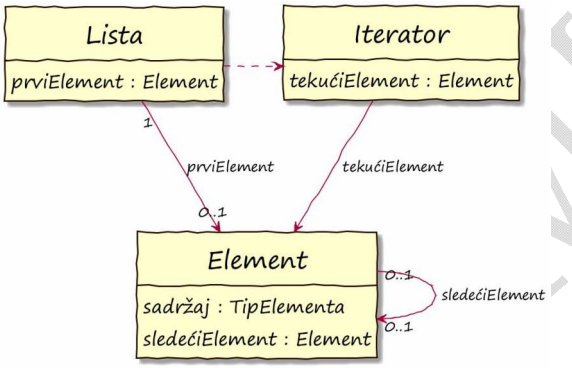


Slika 8 – Dijagram klasa podsistema za komunikaciju zaposlenih

Vrste dijagrama klasa

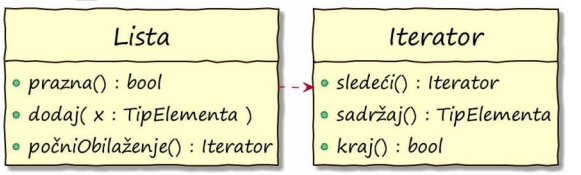
Struktura jednog podsistema može da bude veoma složena, pa predstavljanje svih relevantnih elemenata podsistema na jednom dijagramu može da ima za rezultat dijagram koji je pretrpan informacijama i prilično težak za razumevanje.

Zato je svaki konkretan dijagram klasa obično fokusiran na samo neke od aspekata posmatranog dela sistema. Relativno retko se na jednom dijagramu klasa predstavljaju svi atributi, metodi, ograničenja i odnosi svih klasa, već je uobičajeno da se predstavljaju samo oni elementi koji su značajni u nekom kontekstu. Pri tome mora uvek da bude potpuno jasno naznačeno ako neki elementi nisu predstavljeni, da ne bi neko mogao da pomisli da oni uopšte ne postoje.



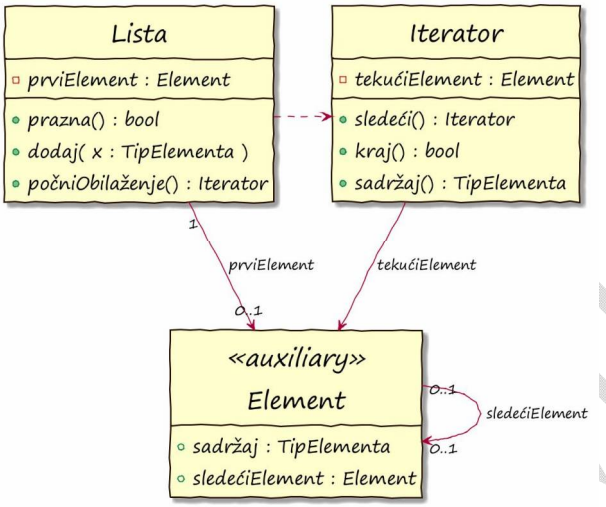
Slika 9 – Dijagram modela domena *Lista*

Dijagram klasa koji se prvenstveno bavi strukturom objekata, tj. informacijama koje su karakteristične za sadržaj objekata i njihove odnose, a zapostavlja njihovo ponašanje tako što uopšte ne sadrži metode, obično se naziva *dijagram klasa domena* ili *dijagram:modela domena* (*Slika 9*). Namena dijagrama klasa domena je da predstavi koje su sve informacije ili podstrukture sadržane u posmatranom domenu ili delu domena, kao i kakav je statički odnos između tih informacija. Takav dijagram može da se koristi i u fazi projektovanja baza podataka, kada se često naziva i *dijagram klasa podataka*.



Slika 10 – Dijagram interfejsa *Lista*

Ako se dijagram klasa bavi prvenstveno ponašanjem, onda može da sadrži metode a da zanemari atribute klasa. Ako sadrži samo javne metode onda se naziva *dijagram interfejsa klasa* (*Slika 10*), a ako sadrži neki izabran skup metoda onda se imenuje tako da bude jasno šta sadrži. Dijagram koji sadrži sve metode, bez atributa, često se naziva *dijagram implementacije*, mada se isti naziv koristi i za kompletne dijagrame, sa svim metodima i atributima.

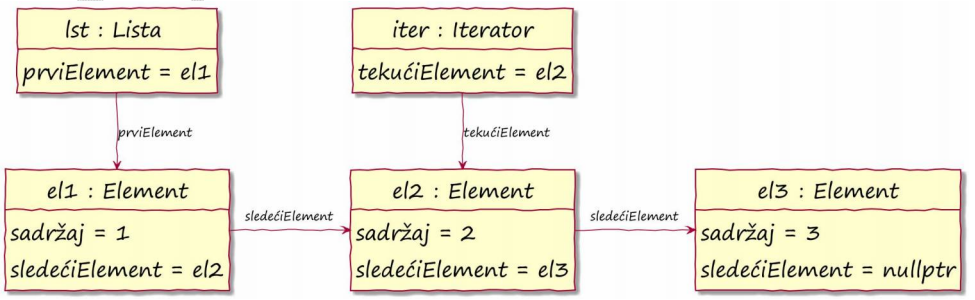


Slika 11 – Dijagram klasa *Lista*

Radi ilustracije, *Slika 9* predstavlja primer dijagrama modela domena, *Slika 10* predstavlja primer dijagrama interfejsa klasa, a *Slika 11* predstavlja kompletan dijagram klasa. Svi ovi dijagrami opisuju kako se implementira dinamička struktura podataka *Lista*.

5.3 Dijagram objekata

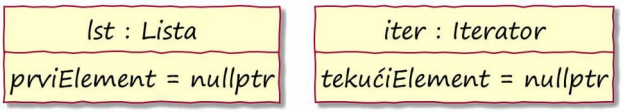
Dijagram objekata nam pomaže da bolje razumemo dinamičku prirodu odnosa među objektima. Koristi se kao dopuna dijagrama klasa i dijagrama komunikacije za opisivanje dinamičkih sistema. Sadrži nazive klasa, nazive objekata, imena i vrednosti atributa i odnose među objektima. On predstavlja objekte i njihove odnose u jednom trenutku vremena, pa je zato za ilustraciju složenije dinamičke prirode nekih klasa često potrebno da se napravi više odgovarajućih dijagrama objekata.



Slika 12 – Dijagram objekata koji ilustruje implementaciju liste

Na dijagramu se predstavlja skup objekata koji postoje u nekom izabranom trenutku vremena. Objekti se predstavljaju pravougaonicima koji u prvom delu sadrže naziv objekta i njegovu klasu (tip), a u drugom delu imena i vrednosti atributa. Dijagram ne mora da sadrži vrednosti svih atributa, već je dovoljno da se predstave oni koji su nam od značaja za konkretan dijagram.

Na primer, *Slika 11* sadrži dijagram klasa koji opisuje odnose klasa koje implementiraju dinamičku strukturu liste. Međutim, iz tih opisa može da ne bude sasvim jasno kako izgleda lista u memoriji. Dodavanjem dijagrama objekata (*Slika 12*), koji opisuje stanje jedne izabrane liste u nekom trenutku, možemo da pomognemo da se lakše razume kako su elementi interno povezani. Dodatno možemo da predstavimo i dijagram objekata koji opisuje praznu listu i završni iterator (*Slika 13*). Ako bi bile dopuštene i kružne liste, to bi takođe moglo da se predstavi posebnim dijagramom objekata.



Slika 13 – Dijagram objekata koji ilustruje implementaciju prazne liste

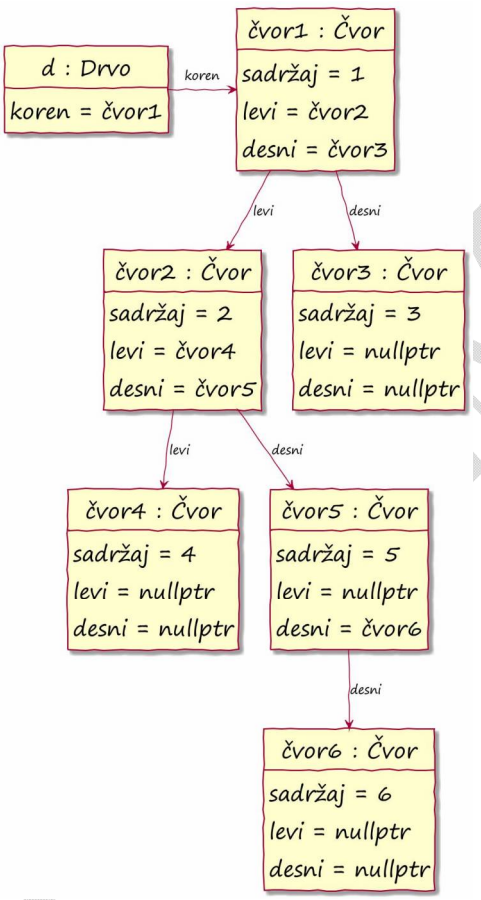
Slično prethodnom primeru, ako napravimo dijagram klasa koje opisuju strukturu binarnog drveta (*Slika 14*), onda odnosi između čvorova drveta mogu da budu još teže razumljivi nego u slučaju liste. Zato bi i u ovom slučaju mogao da bude od pomoći dodatni dijagram objekata (*Slika 15*).

Slika 14 – Dijagram modela domena *Drvo*, koji predstavlja binarno stablo

5.4 Dijagram komponenti

Komponenta predstavlja funkcionalnu ili fizičku celinu softvera, koja ima prepoznatu i zaokruženu ulogu ili funkciju. Za komponente važe veoma slična pravila projektovanja kao za klase – komponenta bi trebalo da ima jednu odgovornost i da tu odgovornost obavlja samostalno, koliko je god to moguće. Na komponente se takođe primenjuju pravila enkapsulacije i izdvajanja interfejsa. Komponente bi trebalo da se međusobno povezuju putem interfejsa. Jedna

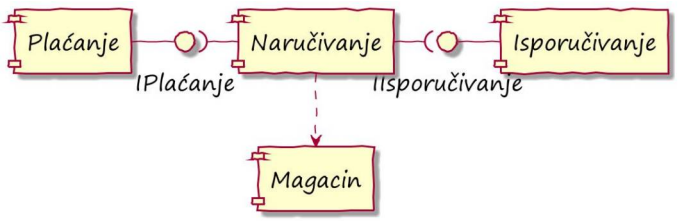
komponenta može da ima više interfejsa, na primer za različite protokole komunikacije ili za različite vrste korisnika.



Slika 15 – Dijagram objekata koji ilustruje implementaciju binarnog drveta

Komponenta se predstavlja pravougaonikom sa dodatnim oznakama koje ukazuju na to da se radi o komponenti. U verziji *UML* 1 se kao oznaka koristi par ispusta sa kojima bi komponenta trebalo da liči na slagalicu. U verziji *UML* 2 se umesto toga koristi pravougaonik koji u desnom gornjem uglu ima simbol slagalice kao oznaku stereotipa. Obe verzije se ravnopravno koriste.

Javni interfejs se predstavlja malim krugom koji može da ima ime i koji je povezan pravom linijom sa odgovarajućom komponentom. Upotreba interfejsa se označava linijom sa polukrugom ili strelicom prema interfejsu (*Slika 16*). Ako jedna komponenta zavisi od druge, ali ne želimo da ističemo interfejse, onda taj odnos može da se predstavi isprekidanom strelicom.



Slika 16 – Primer dijagrama komponenti

Interfejsi moraju da se predstavljaju ako u projektu potencijalno imamo više komponenti koje pružaju isti interfejs, ili ako je fokus upravo na interfejsima. U ostalim slučajevima, a posebno ako na dijagramu ima mnogo komponenti, onda je nekada korisno da se interfejsi sakriju i da se ostave samo neposredne zavisnosti.

5.5 Dijagram slučajeva upotrebe

Dijagram slučajeva upotrebe ilustruje na koji način neke grupe korisnika mogu da stupaju u interakciju sa softverom ili delom softvera. Osnovni elementi ovog dijagrama su *slučajevi upotrebe*, *akteri* koji u njima učestvuju i njihovi međusobni odnosi. Svaka celovita funkcionalna interakcija korisnika se predstavlja kao jedan zaokružen *slučaj upotrebe*. Različite vrste korisnika se predstavljaju kao različite vrste aktera. Pored toga, dijagram može da sadrži i pakete i podsisteme.

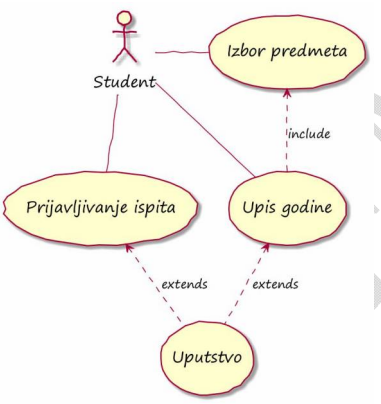
Za razliku od ostalih dijagrama *UML*-a, ova vrsta dijagrama služi samo da nam pruži površan osnovni uvid u elemente i odnose među njima. Glavni deo specifikacije odgovarajućih funkcionalnih celina čine tekstualni opisi slučajeva upotrebe. Svaki slučaj upotrebe mora da bude propraćen opsežnom tekstualnom dokumentacijom, koja ga opisuje do pojednosti.

U zavisnosti od nivoa dekompozicije sistema, slučajevi upotrebe mogu da budu na višem ili nižem nivou sptrahovanja. Pri predstavljanju konceptualnog nivoa softvera teži se da se sa što manje slučajeva upotrebe opiše ceo sistem. Tada su slučajevi upotrebe vrlo apstraktni i opisuju čitav model upotrebe sistema od strane jedne vrste korisnika. Takvi slučajevi upotrebe se obično nazivaju *poslovni slučajevi upotrebe* i uobičajeno se teži da jedan korisnik učestvuje u što manje poslovnih slučajeva, ili čak u samo po jednom. Na primer, *Student studira*, *Nastavnik podučava* i sl.

Na nižim nivoima apstrahovanja, teži se da jedan slučaj upotrebe odgovara jednoj celovitoj aktivnosti korisnika, koja je istovremeno dovoljno velika da predstavlja značajnu celinu, ali i dovoljno mala da može relativno brzo da se implementira. Kao donja granica dekompozicije se obično zahteva da slučaj upotrebe ima *prepoznatljiv rezultat*, tj. da nakon što akter upotrebi sistem na opisan način, kao rezultat ostaje neki trajan trag, koji predstavlja ili završni rezultat tog slučaja upotrebe ili osnovu za

započinjanje ili obavljanje nekog drugog slučaja upotrebe. Tako oblikovani slučajevi upotrebe kasnije predstavljaju osnovne elemente planiranja i implementiranja softvera.

Akteri se predstavljaju jednostavnim simbolima koji označavaju čoveka, ali mogu da se koriste i drugačiji simboli. Naziv aktera se navodi ispod simbola. Slučajevi upotrebe se predstavljaju elipsama u kojima su navedeni nazivi. Akteri i slučajevi upotrebe u kojima oni učestvuju se povezuju (obično pravim) linijama bez strelica na krajevima (*Slika 17*).



Slika 17 – Dijagrama slučajeva upotrebe – Studiranje

Odnosi između slučajeva upotrebe se označavaju strelicama sa isprekidanim linijama (*Slika 17*). Svaki odnos je označen stereotipom koji opisuje prirodu odnosa. Uobičajeni odnosi su:

- *include* – slučaj obuhvata čitav slučaj prema kome ide strelica; obično se koristi za opisivanje delova posla koji se često ponavljaju u drugim slučajevima;
- *extends* – slučaj od koga ide strelica predstavlja *moguće* tj. *opcionalno* proširenje drugog slučaja, u vidu dodatka, ali ne i njegov obavezan deo.

Tekstualna dokumentacija slučajeva upotrebe mora da bude dovoljna da na osnovu nje može da se pristupi planiranju i izradi implementacije. U ranim fazama projektovanja se obično ne navode sve pojedinosti koje su potrebne za implementaciju, ali je potrebno da se navede dovoljno informacija da mogu da se okvirno procene potreban obim i trajanje poslova na implementaciji, kao i eventualne međuzavisnosti sa drugim slučajevima upotrebe. Kasnije, pri detaljnom planiranju implementacije, dokumentacija mora da se dopuni tako da omogući da slučaj upotrebe može da se implementira bez dodatnih zahteva za informacijama.

Opis slučaja upotrebe može da sadrži različite vrste informacija, ali po pravilu mora da sadrži bar sledeće elemente:

- naziv – mora da dobro ilustruje slučaj upotrebe, ali je dobro da bude što kraći;
- aktere – učesnici u slučaju upotrebe; ako različiti subjekti imaju različite uloge, to je potrebno da se objasni;
- kratak opis – suština slučaja upotrebe se objašnjava ukratko, u svega nekoliko rečenica, često i samo jednom rečenicom;
- preduslove – koji preduslovi moraju da budu ispunjeni da bi akteri mogli da započnu slučaj upotrebe;
- postuslove – koji uslovi moraju da budu ispunjeni posle odvijanja slučaja upotrebe;
- opis toka slučaja upotrebe – detaljan opis koraka koji se izvode tokom odvijanja slučaja upotrebe, uključujući i njihov redosled i sve moguće osnovne tokove koraka;
- opis posebnih slučajeva – ako postoje posebni slučajevi, potrebno je da se opiše kada nastupaju i po čemu se odvijanje postupka razlikuje od uobičajenog (npr. greške pri unosu i sl.);
- dijagrame koji tačnije opisuju slučaj upotrebe – često se slučaj upotrebe najbolje opisuje dijagramima aktivnosti ili sekvence, ali mogu da budu od koristi i svi drugi dijagrami *UML*-a;

Dodatni neobavezni elementi obuhvataju sve one informacije koje mogu da budu od značaja pri planiranju ili implementiranju slučaja upotrebe. Na primer, mogu da obuhvataju:

- klasifikaciju – bilo po vrsti slučaja upotrebe ili logičkoj ili poslovnoj celini kojoj pripadaju;
- podaci – ako slučaj upotrebe obrađuje ili proizvodi veću količinu podataka, onda je dobro da se ti podaci opišu i izdvojeno, a ne samo u okviru opisa toka;
- dodatne nefunkcionalne zahteve – na primer, ciljne performanse;
- bezbednosne karakteristike – nivo poverljivosti podataka ili postupka, način pristupanja poverljivim podacima, način kodiranja ili proveravanja ispravnosti podataka i sve drugo što može da ima veze sa bezbednošću podataka;

- pregled najvažnijih rizika – spisak rizika za koje je prepoznato da mogu da nastupe, eventualno alternativne karakteristike ili moguće očekivane izmene u slučaju upotrebe, moguća potencijalna unapređenja i slično;
- jasno objašnjen cilj slučaja upotrebe – ako iz naziva, kratkog opisa i opisa toka izvođenja možda nije sasvim jasno koja je svrha konkretnog slučaja upotrebe ili koju ulogu on ima u okviru softvera kao celine, onda je to potrebno da se objasni;
- prilozi – različiti prilozi koji omogućavaju da se bolje razumeju svrha i tok slučaja upotrebe: primeri formulara ili dokumenata, uzorci realnih podataka i slično.

Radi ilustracije, navešćemo jedan primer tekstualnog opisa slučaja upotrebe (*Slika 18*). Detaljnost odgovara ranim fazama planiranja.

Naziv: Prijavljivanje na konkurs

Akteri: Kandidat koji se prijavljuje na konkurs za upis na fakultet, službenik koji radi na evidentiranju kandidata.

Kratak opis: Kandidat podnosi neophodna dokumenta i formulare na konkurs za upis na fakultet.

Tok događaja:

Osnovni tok:

1. Kandidat donosi rukom popunjen obrazac P1 kao i sledeća dokumenta:
   - fotokopiju izvoda iz matične knjige rođenih (original na uvid),
   - fotokopije svedočanstva sva četiri razreda srednje škole (originali na uvid),
   - fotokopiju svedočanstva o završenoj srednjoj školi (original na uvid),
   - potvrdu o uplati odgovarajućeg iznosa za prijavu.
2. Službenik proverava dokumenta i unosi podatke u sistem
3. Službenik štampa popunjene formulare
4. Kandidat potpisuje i predaje formulare
5. Službenik štampa i potpisuje obrazac P2 i uručuje kandidatu.

Alternativni tokovi:

1a. Za strane državljane (osim za državljane BiH koji su završili školu u RS), potrebna su i nostrifikovana školska dokumenta ili potvrda o započetoj nostrifikaciji

1b. Ako je kandidat položio prijemni ispit na drugom fakultetu, potrebna je i potvrda o položenom prijemnom ispitu na drugom fakultetu

1c. Ako je kandidat nosilac nagrada sa takmičenja koje mogu da doprinesu oslobađanju od prijemnog ispita, onda popunjava i formular P3.

...

Preduslovi: Kandidat je završio srednju školu i dobio odgovarajuća dokumenta.

Postuslovi: Kandidat je prijavljen i dobio je potvrdu koja sadrži redni broj prijave.

Prilozi: sadržaj i predložen izgled formulara P1, P2 i P3

P1. Prijavni list sa ličnim podacima kandidata, podacima o mestu rođenja, državljanstvu, roditeljima, prebivalištu, prethodno završenoj instituciji i studijskim programima koje kandidat želi da upiše.

P2. Identifikacioni list sa imenom, prezimenom i rednim brojem prijave.

P3. Formular za diplome koje kandidat prijavljuje i na osnovu kojih želi da bude oslobođen od prijemnog ispita

Slika 18 – Primer tekstualne dokumentacije slučaja upotrebe

5.6 Dijagram sekvence

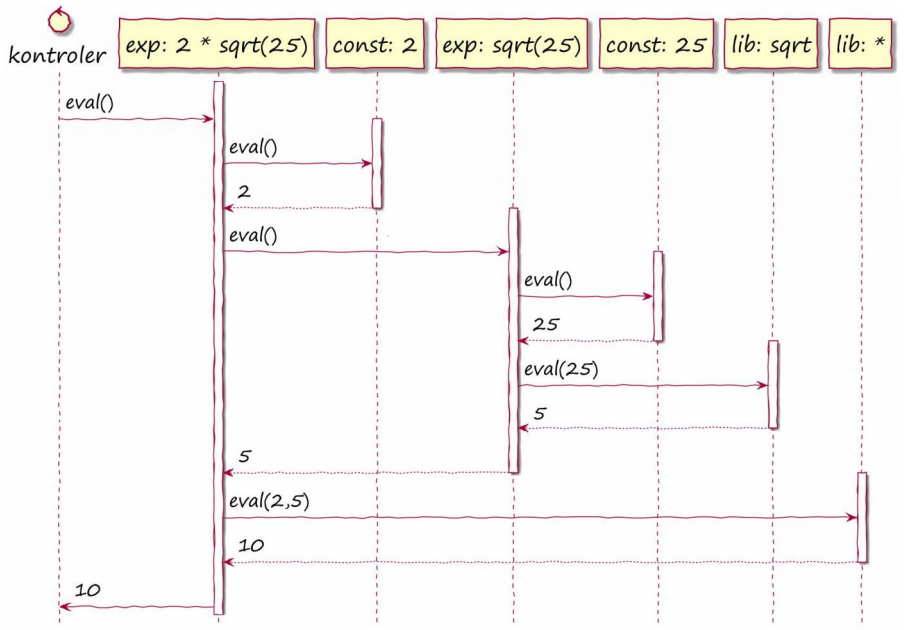
Dijagram sekvence služi za opisivanje toka odvijanja slučaja upotrebe. On omogućava da se predstavi koji subjekat ili objekat je zadužen za koji korak slučaja upotrebe, kao i kojim redom se odvijaju koraci i prenose odgovornosti sa jednog na drugi subjekat ili objekat. Na najnižem nivou apstrakcije, dijagram sekvence predstavlja način implementiranja nekog postupka, sve do nivoa pozivanja konkretnih metoda, odnosno slanja poruka između objekata.

Vertikalna dimenzija predstavlja tok vremena, od vrha prema dnu dijagrama. Horizontalno se dijagram deli na prostore koji odgovaraju pojedinačnim objektima. U svakoj koloni je u vrhu identifikacija objekta, a zatim se ispod crta isprekidana vertikalna linija koja predstavlja životni vek objekta.

Na toj liniji se crtaju uski pravougaonici koji predstavljaju okvire aktivnosti. Okvir aktivnosti objekta započinje prijemom poruke, a završava se kada se primljena poruka obradi. Ako se objekat pravi u toku sekvence, onda je prva poruka koja mu se šalje označena sa *napravi* (ili *create* ili slično). Ako se objekat u nekom trenutku uništava, onda se na dnu odgovarajućeg okvira aktivnosti crta „*X*“.

Sekvenca se predstavlja nizom poruka koje se razmenjuju. Poruka kojom se vraća rezultat se predstavlja strelicom u suprotnom smeru. Poruka sa rezultatom ne mora da se navodi. Ipak, ako je komunikacija sinhrona, tako da objekat koji je uputio poruku ne radi ništa dok ne dobije odgovor, onda je uobičajeno da se i ona navodi. Takođe, ako je rezultat koji se vraća značajan za kontekst dijagrama, dobro je da se navede poruka sa tim rezultatom.

Dijagram sekvence može da se predstavlja na različitim nivoima apstrakcije. U principu, svaka poruka odgovara fizičkom slanju poruka između objekata, tj. pozivanju metoda. Na najnižem nivou apstrahovanja se predstavljaju doslovno sve poruke i onda dijagram sekvence praktično ilustruje kako je potrebno da se implementiraju odgovarajući metodi. Na višim nivoima se ne predstavljaju sve poruke, tj. ne opisuju se svi metodi.



Slika 19 – Dijagram sekvence koji opisuje korake izračunavanja grafa izraza

U primeru (*Slika 19*) je predstavljen dijagram sekvence kojim se opisuje kako teče izračunavanje izraza koji je predstavljen grafom, u kome postoje konstante, aplikacije operacija i bibliotečke operacije.

5.7 Umesto zaključka

Ono što savremenom društvu nije uspelo sa *esperantom*, u domenu razvoja softvera je praktično uspelo sa *UML*-om – dobili smo jezik koji poznaju (ili bi bar trebalo da poznaju) praktično svi učesnici u razvoju softvera. Iscrpno predstavljanje *UML*-a bi zahtevalo mnogo više prostora nego što nam kontekst dopušta, pa smo se zato ograničili samo na predstavljanje osnovnih vrsta dijagrama, da bi čitaoci mogli da počnu da koriste *UML*, ali i da ih njegovo delimično upoznavanje motiviše da se dodatno angažuju i da ga bolje upoznaju.

Na srpskom jeziku postoji solidna literatura o *UML*-u. Iako prevedene knjige nisu baš najažurnije, zato što se radi o prevodima knjiga sa početka 2000-ih godina, koje

se bave prvim verzijama jezika, one su ipak sasvim dobre za početak. Istakli bismo [*Booch 1999*] i [*Fowler 2003a*].