Metodologija naučnog i stručnog rada

UML-motivacija, vrste i primeri

Ognjen Kocić[1], Sanja Mijalković[2] i Predrag Vujić[3]

[1] Matematički fakultet u Beogradu,
11000 Beograd, Srbija
zbetmen1@gmail.com

[2] Matematički fakultet u Beogradu,
11000 Beograd, Srbija
sanja.mijalkovic@yahoo.com

[3] Matematički fakultet u Beogradu,
11000 Beograd, Srbija
wujic88@gmail.com

Apstrakt. UML dijagramska tehnika za vizualizaciju, modeliranje, specifikaciju i dokumentovanje softverskih rešenja je danas jedina opšteprisutna tehnika ove vrste koja se primenjuje intenzivno u svim fazama i aspektima razvoja softvera. Bilo kakav proizvod ili alat koji ima široku bazu korisnika ima veliku važnost za tu zajednicu, pa je tako i UML ključan pri pisanju visoko kvalitetnog i složenog softvera. Poslednjih godina UML je od prestižnih softverskih kompanija stigao i u studentske klupe svih ozbiljnih univerziteta na kojima se izučava softversko inženjerstvo ili slična računarska nauka. Uprkos velikim naporima nastavnog osoblja prava važnost UML-a ne može biti dočarana na malim veštačkim primerima na kojima se prezentuje. U našem radu mi ćemo predstaviti sve bitnije UML tehnike, dati predloge vezane za dobre komercijalne i alate otvorenog koda, pri čemu će akcenat biti na najvažnijoj osobini UML-a, a to je prevazilaženje komunikacionih problema jedinstvenim zapisom ideja.
Ključne reči: dizajn i projektovanje softvera, objektno-orijentisan razvoj, grafički jezik, komunikacija u računarstvu

Sadržaj:

1. Uvod

Objedinjeni jezik za modeliranje je opšti grafički jezik za koji se koristi za specifikaciju, vizuelizaciju, konstrukciju i dokumentovanje detalja jednog softverskog sistema. On čuva informacije o odlukama i pruža razumevanje o sistemu koji treba napraviti. U osnovi je dizajniran tako da može jednako dobro da se koristi sa svim razvojnim metodologijama, životnim ciklusima razvoja i oblasti primene pravljenog softverskog sistema. Nastao je kao pozitivna unija svih prethodnih iskustava vezanih za modelovanje softverskih sistema i kao takav je postao jedini standard. Sadrži semantičku komponentu, jednistvenu notaciju i razne vodiče za primenu. Dizajniran je da podrži interaktivne alate koji imaju mogućnost generisanja dokumentacionih izveštaja i programskog koda.

UML čuva dve vrste informacija, prva je o statičkoj strukturi, a druga je o dinamičkom ponašanju sistema. Sistem se modeluje kao zatvorena celina koja se sastoji od jasno razdvojenih komponenti čijim se zajedničkim delovanjem proizvodi rezultat koji ima značaj za spoljašnjeg korisnika. Statička struktura definiše kakvoću objekata koji su važni za sistem i njegovu implementaciju, kao i postojane veze između ovih objekata (hijerarhije klasa i sl.). Dinamička komponenta UML-a ima za cilj da jasno opiše kakve se promene dešavaju objektima u određenom vremenskom intervalu dok sistem radi, kao i koji objekti za to vreme međusobno komuniciraju i kakve su im poruke. Prikazivanje sistema iz različitih uglova omogućava razumevanje njegove primene u različite svrhe.

Softverski projekti danas imaju čak i milione linija programskog koda. Postalo je nemoguće razmatrati projekte u celini i sa velikim nivou detalja. UML pruža organizacione konstrukte koji mogu da drže ovu složenost pod kontrolom. On omogućava organizaciju manjih modela u veće pakete, koji dalje omogućavaju softverskim timovima da podele velike sisteme na delove koji se pojedinačno mogu efikasno obrađivati (za razliku od celog sistema). UML dodatno omogućava prikazivanje međusobnih zavisnosti između ovih paketa kao i organizaciju elemenata u fazi izvršavanja (Eng. *run-time*) u komponente.

Na kraju, bitno je napomenuti šta sve UML nije i na taj način razbiti uobičajene predrasude koje postoje u računarskom svetu. UML nije programski jezik. Tačno je da postoje alati za automatsko generisanje koda iz UML dijagrama, kao i alati za generisanje UML dijagrama iz postojećeg programskog koda, međutim niti postoji kompajler, niti postoji interpreter za UML, niti je ikad zamišljeno da ovaj grafički jezik za modelovanje preraste u programski jezik. UML nije totalno formalan jezik, na primer moguće su manje varijacije u sintaksi od alata do alata. Diskretan je pa samim tim nije

pogodan za modelovanje kontinualnih sistema kakvi se na primer mogu naći u fizici i hemiji. UML je opšt pa samim tim za specijalizovane poslove kao što je VLSI dizajn, raspored GUI komponenti (Eng. *GUI layout*) sigurno postoje bolje prilagođeni alati.

2. Motivacija iza UML-a

U ovoj sekciji biće predstavljeno koja je motivacija vezana za UML, kako je tekao razvoj UML i šta je rukovodilo tim razvojem, osnovni cilj UML-a i na samom kraju biće reči o konceptima UML-a.

2.1. Kratak istorijat

Bez obzira ne svoju opštost UML je realno izuzetno vezan za ideju objektno-orijentisanog programiranja. Razvojni metodi za stare proceduralne programske jezike kao što su Fortran i Cobol razvili su se još u 70-im godina, a bili su u širokoj upotrebi već 80-ih. Uglavnom su se koristili Strukturna analiza i Strukturni dizajn [Yourdon-79] i varijante kao što je Strukturni dizajn u realnom vremenu [Ward-85] i sl. Kako su se u to vreme sistemi razvijali za potrebe vojske morali su biti razvijani sa visokim nivoom organizacije i možda nepotrebno detaljnim nivoom dokumentovanja sistema i njegovog razvoja. Rezultati nisu bili baš sjajni, veoma često se dešavalo da napravljeni sistemi bude tek nešto korisniji od sistema za generisanje izveštaja. Privatni preduzetnici nisu bili baš srećni sa ovim čeličnim načinom razvoja softvera, već je kružilo uvreženo mišljenje da su biznis aplikacije mnogo jednostavnije pa im ne treba nadzor spoljašnjih specijalizovanih kompanija. To je rezultovalo time što su one razvijale softver interno, svaka koristeći neku svoju metodologiju.

Prvi objektno-orijentisani jezik je zvanično Simula iz 1967. ali nije bio baš korišćen. Pravi razvoj objektno-orijentisanih jezika dešava se početkom 80-ih ka 90-im godinama sa pojavom jezika kao što su Objective-C, C++, Eiffel i CLOS. Kako je ova paradigma zadobila pažnju softverskih inženjera i softverskih arhitekti tako je u, relativno gledano, kratkom vremenskom periodu od 83-e do 92-e izdato jako puno knjiga koje su opisivale razvoj zasnovan na toj paradigmi. Za ove knjige važe da su bili prepune dobrih ideja sugestija i postupaka koji bi mogli postati dobra praksa, međutim postojala je velika raznolikost u samoj terminologiji, definicijama i notaciji. Zbog svega ovoga čitaocima je bilo jako teško da se naviknu na izlagani materijal.

U jednom trenutku svima je postalo jasno da se mora doći do nekg sporazuma vezano za terminologiju i tehnike kojima bi se objektno-orijentisane ideje predstavile. Bilo je

nekih ranih pokušaja koji nisu uspeli, od kojih je najznačnija tzv. Fuzija u kojoj su učestvovali Coleman, Rumbaugh, Booch i Wirfs-Brock. Međutim krajnji proizvod je bio više novi metod nego unifikacija postojećih. Uspeh dolazi malo kasnije, 94-te godine, kada se Rumbaugh priključuje Booch-u u firmi Rational Software Corporation. Oni zajedno kombinuju OMT i Booch-ove metode što već sledeće godine rezultira prvim predlogom razvojne metode. Te iste godine im se priključuje i Jackobson i njih troica zajedno nastavljaju da rade na novoj metodi. Kako su oni bili tri najveća imena u objektno-orijentisanim metodologijama u to vreme, ovo je ujedno bio i veliki podstrek drugim razvijaocima metodologija da se potrude da ujedine znanje, ideje i ciljeve. Godinu dana kasnije organizacija *Object Managment Group* (u daljem tekstu OMG) otvoreno traži da se napiše predlog za standardizaciju pristupa razvoju sa objektno-orijentisanim metodologijama. Drugi predlozi su otpali i 1997. godine rad Rumbaugh-a, Booch-a i Jackobson-a poznatiji kao UML biva prihvaćen za neku vrstu standarda. Treba pomenuti da je iz njihovog rada nastala i razvojna metodologija poznatija kao *Unified Approach*. Prvi UML standard je postao zvaničan u novembru 1997.

2.2. Cilj UML-a

Prvi i najvažniji cilj UML-a je da je to opšti jezik za modelovanje koji svi mogu koristiti. Ne nalazi se ni u čijem vlasništvu i zasnovan je principima koji su postignuti dogovorom velikog dela računarske zajednice. Planiran je tako da uključi elemente svih važnijih metodologija kako bi ga one mogle koristiti kao svoj jezik za modelovanje. Dizajniran je tako da bude prirodan za upotrebu i sa najlogičnijom i najjasnijom notacijom. Pravljen je da podržava jednostavno izražavanje dobrih koncepata kao što su enkapsulacija, razdvajanje odgovornosti i prikazivanje namere vezane za neku softversku konstrukciju. Obiman softver, distribuiran softver, konkurentan softver, timski rad, na svaku od ovih stavki se mislilo pri dizajniranju UML-a. Na kraju, treba shvatiti da UML nije zamišljen da se koristi kao metodologija razvoja, mada originalni autori predlažu jednu, već kao potpora za veliki broj postojećih metodologija.

2.3. UML koncepti

UML koncepti se mogu grupisati u nekoliko grupa.

Statičke strukture. Svaka metodologija mora definisati određene ključne elemente strukture vezane za softver koji se pravi. Taj pogled na skelet sistema je statički pogled na aplikaciju. Osnovni koncepti softverskog sistema se modeluju klasama koje nisu ništa više nego skupovi objekata sa zajedničkim ponašanjem i/ili podacima. Podaci se modeliraju atributima, dok se ponašanje klasa opisuje operacijama koje objekti mogu da

izvrše (implementiraju se kao metodi). Nekoliko klasa može deliti unutrašnju strukturu kroz koncept generaizacije pri čemu svaka klasa niže u hijerarhiji može inkrementalno dodavati nove informacije i/ili operacije, dok nasleđuje neke postojeće iz viših klasa u hijerarhiji. Objekti klasa mogu biti povezani na neki način u toku samog izvršavanja (na primer škola ne postoji bez učenika). Ovakve povezanosti se opisuju asocijacijama. Osim ovih osnovnih pojmova postoje i interfejsi, signali, slučajevi upotrebe, tipovi podataka i sl., ali će oni biti predstavljeni kasnije u tekstu. Statička struktura se predstavlja *dijagramom klasa.*

Dinamičko ponašanje. Postoje dve dimenzije kod modeliranja ponašanja objekta. Jedna se odnosi na njegovu istoriju, a druga na pravilnosti u komunikaciji sa drugim objektima (na primer slanje "otkucaja kod" mrežnih aplikacija). Istorija objekta bez komunikacije sa drugim objektima se može jednostavno predstaviti stanjima kroz koje objekat prolazi. On pravi neke akcije i menja stanja u skladu sa svojim akcijama i spoljašnjim odgovorima na te akcije. Druga dimenzija prikazuje kako neki objekti zajedno sarađuju da bi ostvarili neki veći posao. Tu treba da se vidi koji objekti su koliko prisutni (od početka posla do kraja ili samo na početku itd.) i kakve poruke oni međusobno razmenjuju.

Implementacioni konstrukti. Takođe, osim logičke podrške UML pruža i podršku samoj implementaciji rešenja. Definiše komponentu koja mora imati određeni javni interfejs. Osnovna ideja komponente je da bude lako zamenljiva drugom koja pruža isti intefejs (naravno nekom modernijom i boljom implementacijom tog interfejsa). Često u aplikacijama imamo udaljene resurse koji se koriste (na primer server sa bazom podataka nije isti kao i aplikativni server). Ovi resursi se nazivaju čvorovi. Postoji tzv. produkcioni pogled na aplikaciju (Eng. *deployment view*) koji prikazuje organizaciju ovih komponenti po čvorovima, topologiju čvorova, moduće migracije komponenti i sl.

Organizacija modela. Malo nelogičan naziv koji opisuje jednu jako prirodnu potrebu – timovi moraju da mogu da rade na različitim delovima sistema konkurentno. UML pruža podršku podeli posla na pakete, upravljanju njihovim međuzavisnostima i sl.

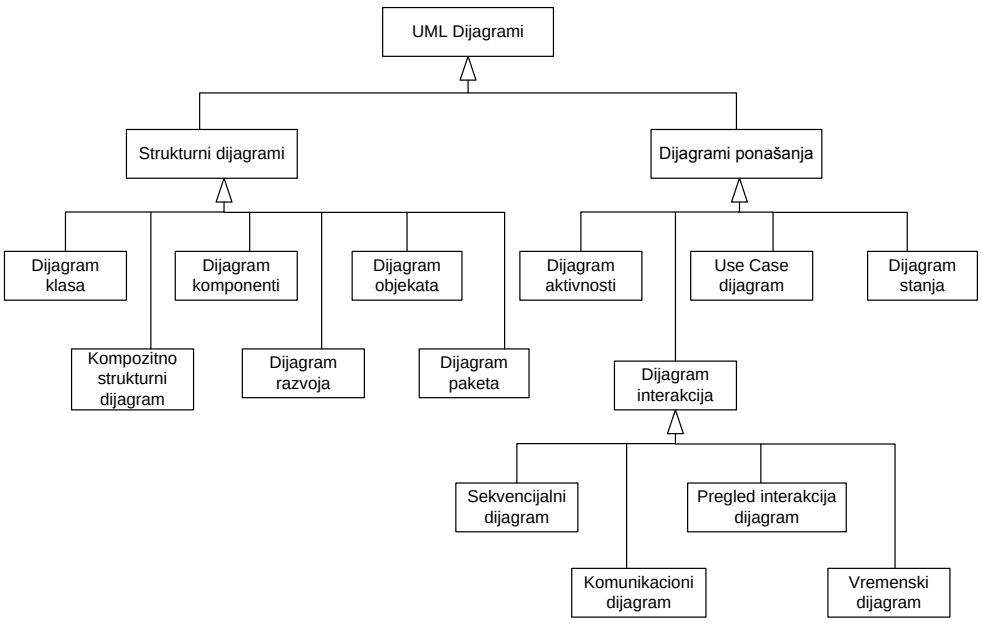
Mehanizam proširenja. Autori UML-a su bili svesni dinamičkog razvoja računarstva koji se dešavao do tada neviđenom brzinom i znali su da će jednog dana doći do potrebe za proširenjem UML-a. Zbog toga su obezbedili određene mehanizme kojima se to može izvesti bez opasnosti od dolaska do nekih radikalnijih promena (međutim ipak je moguće da dođe do različitih dijalekata UML-a). Osnova za ovo je jezik OCL koji služi da se njime zapišu ograničenja. Pojednostavljeno, proširenje se vrši tako što se prave nove vrste dijagrama primenom novih ograničenja na već postojeće dijagrame.

3. O vrstama UML dijagrama

U procesu razvoja softvera pojavljuju se razne uloge koje koriste različite tehnike formiranja dijagrama, za rešavanje različitih problema. To u stvari znači da brojni učesnici u procesu razvoja „razgovaraju“ istim jezikom. Da ne bi došlo do zabune, potrebno je napomenuti da se proces razvoja ne može obaviti samo UML-om. Medjutim, standardizacijom i opštim prihvatanjem UML-a stvoreni su preduslovi da sve aktivnosti koje se ne mogu realizovati pomoću UML-a budu usaglašene sa aktivnostima koje se realizuju pomoću njega. Trenutno aktivna verzija UML-a jeste 2.0. Osnovna podela UML dijagrama je na sledeće tri grupe:

1. Strukturni dijagrami
2. Dijagrami ponasanja
3. Dijagrami interakcije

Hijerarhija UML dijagrama je predstavljena na slici Slika 1.



Slika 1. Hijerarhija UML dijagrama

3.1. Strukturni dijagrami

Strukturni dijagrami ističu koje činioce sistema je potrebno modelovati. Shodno tome, dele se na sledeće grupe:

- Dijagram klasa (Eng. *Class diagram*)
- Dijagram komponenti (Eng. *Component diagram*)
- Dijagram složene strukture (Eng. *Composite structure diagram*)
- Dijagram raspoređivanja (Eng. *Deployment diagram*)
- Dijagram objekata (Eng. *Object diagram*)
- Dijagram paketa (Eng. *Package diagram*)

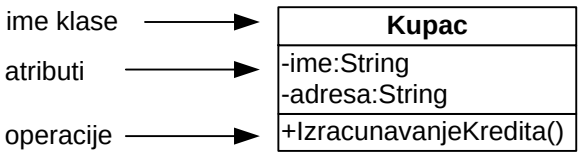
U nastavku je posebno predstavljene neke značajnije vrste strukturnih dijagrama.

Dijagram klasa. Dijagrami klasa predstavljaju ključne dijagrame za opisivanje strukture sistema. Klasa predstavlja osnovni pojam u objektnom razvoju, te kao takva predstavlja i osnovnu komponentu objektno razvijanog sistema. Ovi dijagrami opisuju klase sistema i uz to attribute koji pripadaju toj klasi, metode (operacije) i odnose između klasa.

- Atributi predstavljaju obeležja koja opisuju svojstva klase. Navode se u okviru klase, a svaki atribut može da poseduje sledeća svojstva: vidljivost, ime, tip, multiplicitet, podrazumevanu vrednost, kao i opis nekih dodatnih svojstava atributa (npr. čitljivost).
- Metode klase opisuju poslove koje će objekat, kao instance konkretne klase, znati da izvrši. Kada se od objekta zahteva da obavi neki posao, to se može zahtevati samo preko metode koju on poseduje, odnosno preko interfejsa koji odgovarajuća klasa pruža. Metod takođe poseduje odgovarajuća svojstva: vidljivost, ime, listu parametara, tip rezultata operacije, itd.
- Relacije služe da bi se prikazao međusobni odnos klasa. Između klasa mogu da postoje sledeće vrste relacija
  - asocijacija
  - agregacija
  - kompozicija
  - specijalizacija

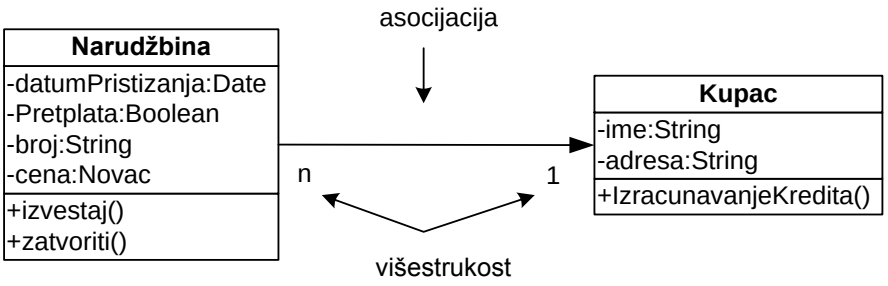
- generalizacija.

U dijagramu klasa UML-a klasa se predstavlja pravougaonikom koji je najčešće podeljen u tri dela. U prvom delu se nalazi ime klase, drugi deo sadrži attribute klase dok treći deo predstavlja metode klase, kao sto je predstavljeno u seledećem primeru:



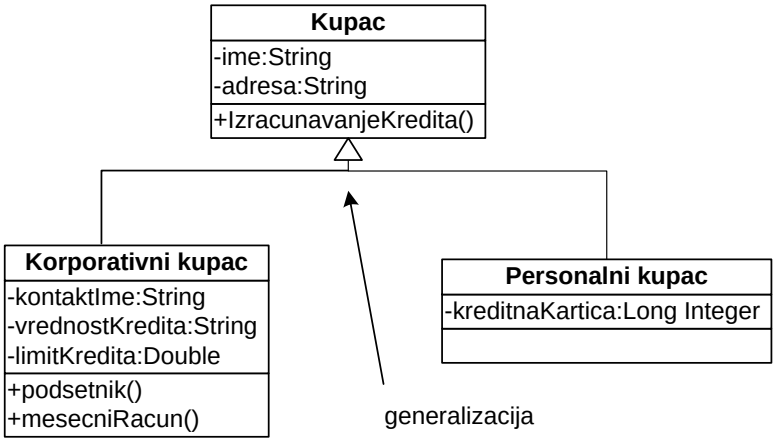
Slika 2. Primer klase u dijagramu klasa

Sam dijagram se sastoji od ovakvih dijagrama i odnosa medju njima. Na slikama Slika 3 i Slika 4 se može videti kako se predstavljaju pojedini odnosi medju klasama.



Slika 3. Primer dijagrama klasa sa asocijacijom

Veza asocijativnost je veza koja se najviše koristi kod ovakvih dijagrama. Ova veza prikazuje vezu između instanca klasa. Ovde je klasa Narudžbina asocirana klasom Kupac. Višestrukost asocijacije naznačava broj objekata koji učestvuje u vezi. U ovom primeru višestrukost iynačava da jedna narudžbina može da odgovara samo jednom kupcu, ali kupac može biti povezan sa većim brojem porudžbina.



Slika 4. Primer dijagrama klasa sa generalizacijom

Kada su dve klase slične sa pojedinim različitostima možemo ih generalizovati kao sto je prikazano na slici.

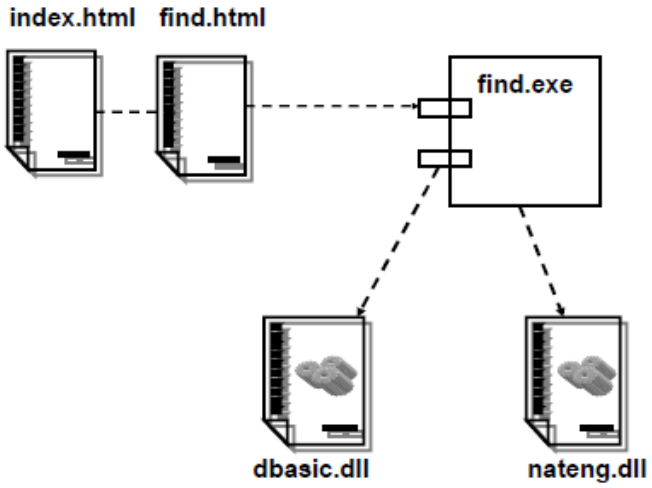
Dijagram komponenti. Dijagrami komponenti služe da bi se prikazale komponente sistema. Pod komponentama se podrazumevaju takvi delovi sistema koji se mogu samostalno isporučivati krajnjim korisnicima. Sve komponente moraju biti tako međusobno usaglašene da dodavanje nove komponente u sistem ne može da izazove poremećaje kompletnog sistema.

Uobičajeno svaka komponenta se sastoji od jedne ili više klasa i predstavlja nezavisnu celinu, koja je povezana sa ostatkom sistema pomoću njenog spoljašnjeg interfejsa. Na ovakav način se postiže da promene u jednoj komponenti ne deluju destruktivno na ostatak sistema, jer interfejs i dalje čuva integritet komponente i ostatka sistema. Na ovaj način se prikazuje konstrukcija izvršnih softverskih sistema.

Dijagram sadrži

- komponente
- njihove međusobne odnose
- javne interfijse

Primer se može videti na slici Slika 5.



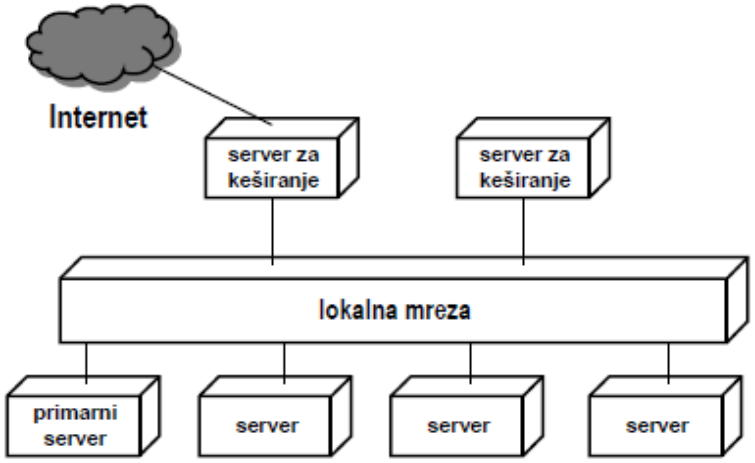
Slika 5. Primer dijagrama komponenti

Dijagrami složene strukture. Dijagrami složene strukture prikazuju saradnju klasa, interfejsa ili komponenti u cilju opisa strukture zadužene za izvršavanje posmatrane funkcionalnosti. Ovi dijagrami su slični dijagramima klasa. Razlika je u tome što dijagrami klasa prikazuju statičku strukturu sistema, kroz prikaz klasa sa njihovim atributima i operacijama, a dijagrami složene strukture prikazuju izvršnu arhitekturu, relacije izmenu njenih gradivnih elemenata i odnos posmatrane arhitekture sa okruženjem, u cilju prikazivanja informacija koji se ne mogu prikazati pomoću statičkih dijagrama. Dijagrami složene strukture su takone novina u verziji UML-a 2.0.

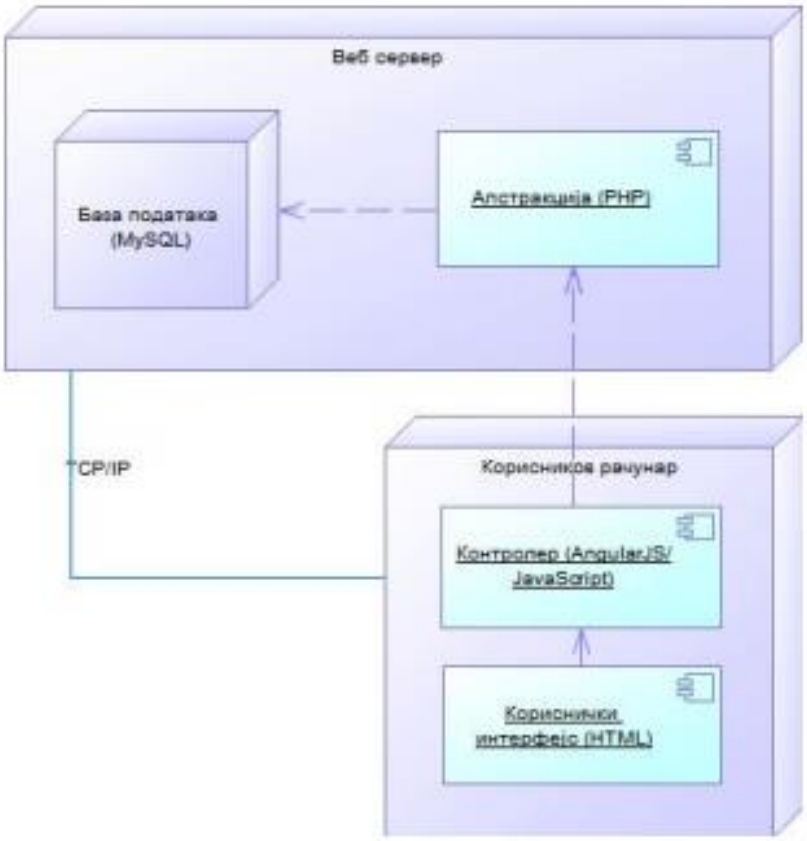
Dijagram sadrži:

- složene komponente i njihove elemente
- tačke interakcije sa drugim elementima Sistema

Dijagrami raspoređivanja. Dijagrami raspoređivanja služe za predstavljanje hardverske arhitekture sistema. Oni prikazuju delove sistema raspoređene po fizičkim lokacijama. Ovi dijagrami se mogu posmatrati i kao prikaz arhitekturalnog rešenja celokupnog sistema. Pomoću njih se prikazuje konfiguracija procesnih čvorova u toku izvršavanja i komponenti koje se nalaze u njima. Predstavljaju posebnu vrstu dijagrama klasa namenjenih statičkom prikazu rasporenenosti hardvera sistema. Sastoje se iz čvorova, koji se prikazuju u obliku kvadra, kao i relacija izmenu ovih čvorova. Primeri se mogu videti na slikama Slika 6 i Slika 7.



Slika 6. Primer dijagrama raspoređivanja

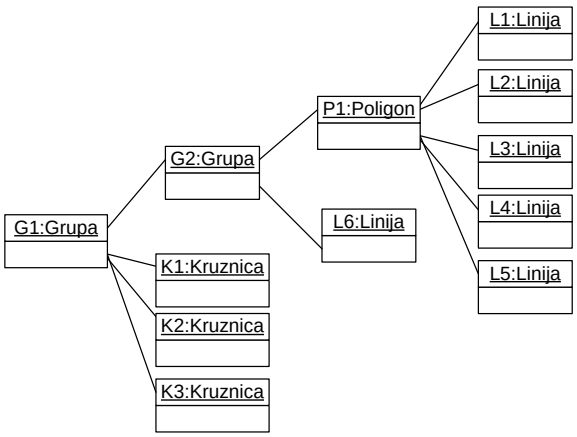


Slika 7. Primer dijagrama raspoređivanja

Dijagrami objekata. Dijagrami objekata su postojali i u ranijim verzijama UML-a, ali kao neformalni dijagrami. Od verzije 2.0 zvanično postaju UML dijagrami. Služe da prikažu objekte posmatranog dela sistema u posmatranom trenutku. Koriste se da bi se dodatno opisala struktura sistema, u situacijama kada dijagrami klasa ne daju dovoljno kvalitetan opis iste. Oni objašnjavaju i ilustruju skup dijagrama klasa i predstavljaju slike objekata i njihovih veza u specificnom trenutku vremena. Znači, dijagrami objekata sadrže:

- nazive klasa
- nazive objekata
- imena i vrednosti atributa

- odnose između objekata.



Slika 8. Primer dijagrama objekata

Imena objekata, koja su podvučena, su sastavljena od imena samog objekta praćenog dvotačkom i imenom klase, npr. G1:Grupa gde G1 predstavlja ime objekta a Grupa predstavlja klasu čija je istanca objekat G1.

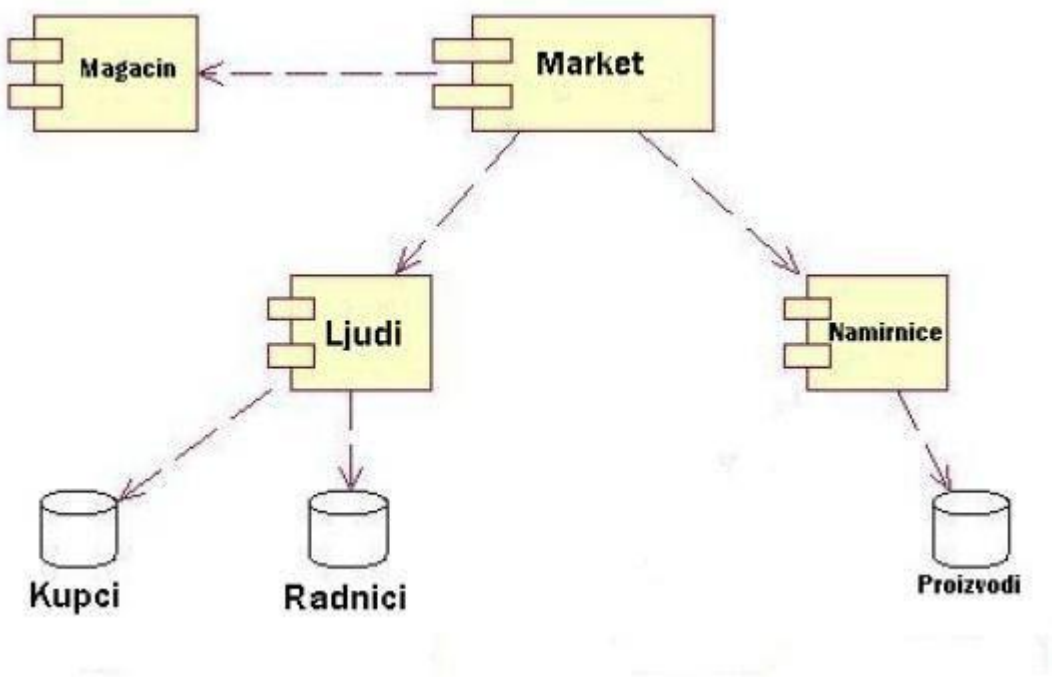
Dijagrami paketa. Paketi predstavlju mehanizme za grupisanje UML elemenata. Iako se najčešće koriste za grupisanje klasa, mogu se koristiti i za grupisanje drugih elemenata, npr. za grupisanje slučajeva upotrebe, za grupisanje entiteta ili za grupisanje komponenti sistema. Predstavljaju se pomoću pravougaonika sa jezičkom u levom gornjem uglu, na kojem je ispisan naziv paketa. Izmenu paketa se povlače relacije zavisnosti, koje govore da se neki od elemenata smeštenih u pakete između kojih postoji zavisnost nalaze u međusobnom odnosu.

Ukoliko govorimo o paketima klasa, tada bi do njihovog kreiranja moglo doći, npr. radi grupisanja hijerarhijske strukture klasa ili grupisanja klasa čije su instance u menusobnoj zavisnosti predstavljenoj na dijagramima sekvenci ili dijagramima saradnje. Moguće je takone praviti pakete na osnovu stereotipova klasa. Dakle, ukoliko se klase nalaze u logičkoj ili fizičkoj povezanosti moguće ih je smestiti u paket klasa i predstaviti više takvih paketa na dijagramu paketa. Ovi su dijagrami, iako ranije nezvanično korišćeni, novina u UML verziji 2.0.

Dijagrami paketa sadrže:

- nazive i granice paketa
- klase u paketima
- međusobne odnose klasa
- međusobne zavisnosti paketa
- može se koristiti i u domenu slučajeva upotrebe

Na slici Slika 9 prikazano je kako su elementi logičkog modela organizovani u pakete, kao i međuzavisnosti paketa. Dijagram sa slike opisuje organizaciju marketa, prikazane su komponente organzacije i medjusobna povezanost. Paketi predstavljaju logicke celine: market, magacin, ljudi i namirnice.



Slika 9. Primer dijagrama paketa

3.2. Dijagrami ponašanja

Dijagrami ponašanja ističu šta se dešava u sistemu koji se modeluje:

- Dijagram aktivnosti (Eng. *Activity diagram*)
- Dijagram stanja (Eng. *State diagram*)
- Use Case dijagram (Eng. *Use case diagram*)

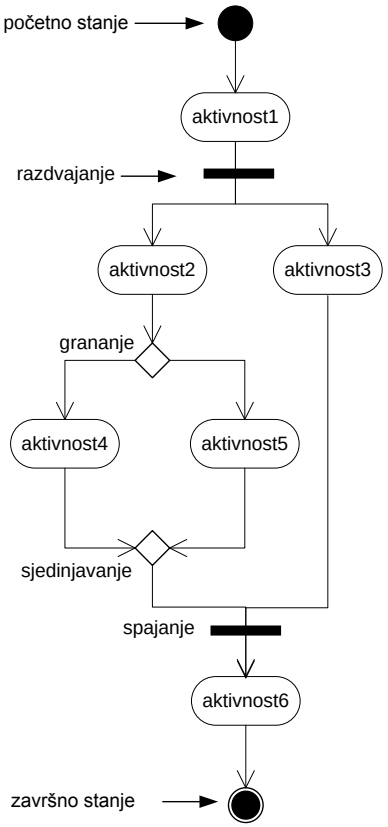
Dijagrami aktivnosti. Dijagrami aktivnosti služe za istraživanje i opisivanje logike procedura, poslovnih postupaka i radnog toka. Koriste se i za prikazivanje akcija operacija u klasi, slično kao i tradicionalni dijagrami toka podataka. Tokovi funkcionalnosti predstavljeni dijagramima slučajeva upotrebe se opisuju dijagramima aktivnosti, koji prikazuju sve aktivnosti koje se odvijaju u okviru posmatrane funkcionalnosti. Pomoću jednog dijagrama aktivnosti moguće je prikazati više potencijalnih scenarija koji se mogu desiti pri izvršavanju neke funkcionalnosti. Ukoliko se na dijagramima slučajeva upotrebe slučaj upotrebe posmatra kao „crna kutija“, na dijagramima aktivnosti se prikazuje redosled izvršavanja aktivnosti u okviru te „crne kutije“.

Dijagrami aktivnosti sadrže:

- stanja aktivnosti
- tokove podataka
- stanja akcije
- razdvajanja
- grananja
- spajanja
- može sadržati i linije autora
- početnu tačku (označava se crnim krugom)
- krajnju tačku (označava se crnim krugom u belom krugu).

Tokovi podataka povezuju početnu tačku, akcije i krajnju tačku i predstavljaju se usmerenim linijama. Stanja aktivnosti i akcija navode se u zaobljenim pravougaonicima. Imaju grananje za opisivanje uslova i tačka u kojoj se vrši grananje prikazuje se rombom iz koga postoje 2 ili vise izlaznih tokova podataka. Razdvajanja služe za opisivanje paralelnih aktivnosti, odnostno koristi se kada se javi veci broj aktivnosti u isto vreme. Predstavlja se ravnom zadebljanom linijom. Linije autora se (opciono) koriste ukoliko želimo da prikažemo koji učesnik izvršava koju aktivnost. Dijagrami se čitaju odozgo na dole (s leva na desno).

Na slici Slika 10 nakon aktivnosti aktivnost1 sledi razdvajanje, što znači da se aktivnosti aktivnost2 i aktivnost3 odvijaju paralelno u istom vremenu. Nakon aktivnosti aktivnost2 javlja se grananje, koje se prikazuje simbolom odluke (prazan romb). Grananje opisuje koje aktivnosti će da se odigraju nakon odgovarajućeg uslova. Sva grananja na kraju se završavaju i istoj tački koja predstavlja sjedinjavanje i označava kraj grananja. Nakon sjedinjavanja sve paralelne aktivnosti se kombinuju i spajaju pre završnog stanja.



Slika 10. Grafički prikaz dijagrama aktivnosti

Dijagrami stanja. Ovi dijagrami služe za prikazivanje ponašanja dela sistema, odnosno ponašanje objekta kao instance posmatrane klase. Na njima se predstavljaju stanja posmatranog objekta, tranzicije izmenu stanja i događaji koji uzrokuju prelazak objekta iz jednog u drugo stanje. Crtanje dijagrama stanja se ne preporučuje za sve klase sistema. Najčešće se crtaju za najznačajnije klase sistema ili se uopšte ne crtaju ukoliko razvojni tim informacije koje se dobijaju ovim dijagramima dobije na drugi način. Predstavljaju uopštenje dijagrama aktivnosti i najčešće kombinuju sa njima i dijagramima interakcije. Ovim dijagramima su najčešće obuhvaćena sva stanja u kojima se objekat može naći tokom vremena. Stanje je izraženo preko vrednosti atributa i veza sa drugim objektima. Na primer, stanje objekta je:

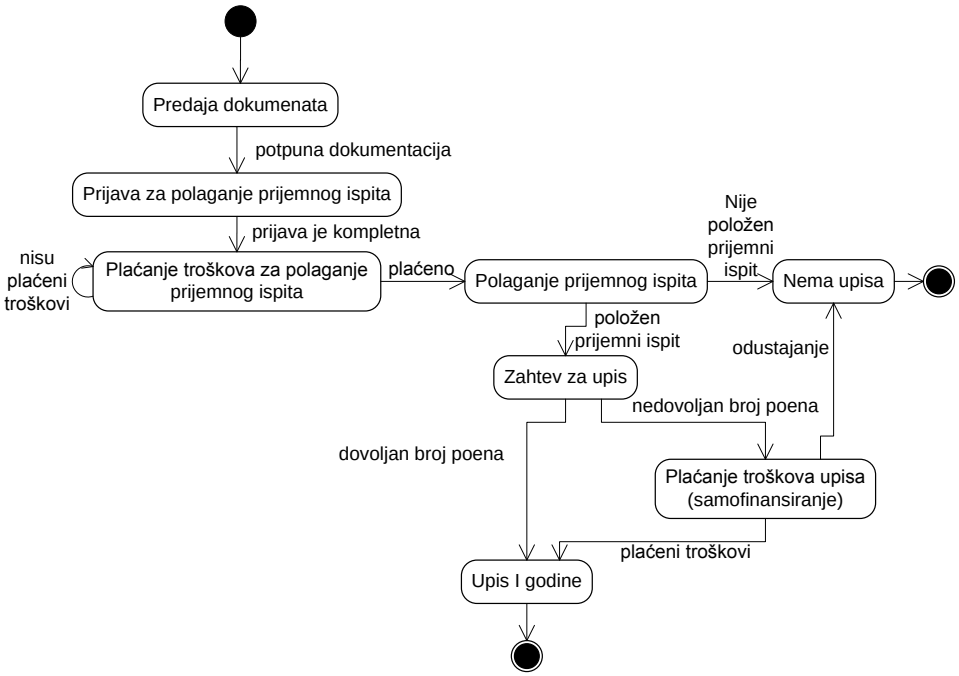
Račun (**objekat**) je plaćen (**stanje**).

Pomoću dijagrama stanja se modeluju dinamički aspekti sistema koji se projektuje. Koriste se za modelovanje životnog veka objekta, a najčešće se modeluju stanja objekata koji reaguju. Na dijagramu se, u obliku konačnog automata, prikazuje odvijanje upravljanja od stanja do stanja. Ovo se prikazuje tako što se stanje prikazuje pravougaonikom sa zaobljenim ivicama, dok je prelaz relacija izmedju dva stanja i prikazuje se usmerenom strelicom od jednog stanja ka drugom, a stanje od kojeg počinje kreiranje ovog konačnog automata se prikazuje crnim kružićem. Promena stanja imenovana je svojim razlogom – događajem koji je izazvao prelazak iz jednog u drugo stanje. Kada se taj događaj dogodi, ispuni se promena iz jednog stanja u drugo. Početna tacka, ili početno stanje, predstavljena je punim krugom, krajnja tačka ili krajnje stanje je predstavljena malim punim krugom okruženog velikim praznim krugom.

Dijagrami stanja sadrže :

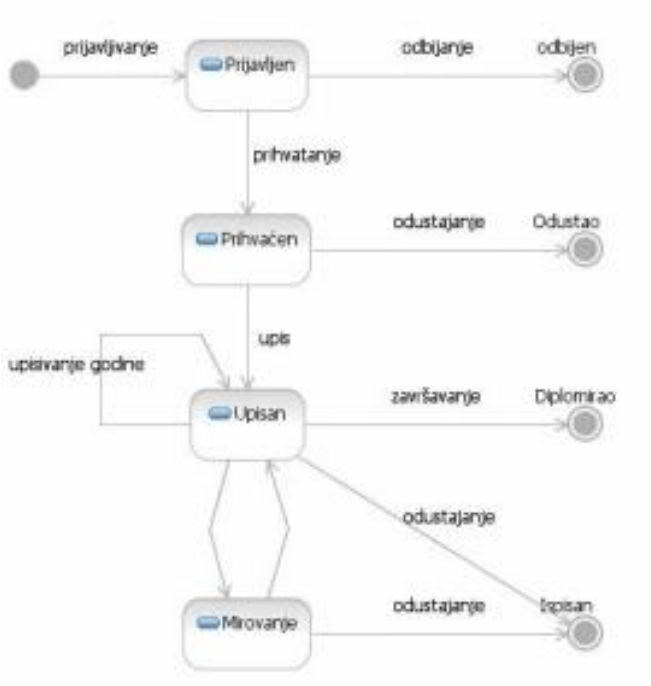
- sva moguća stanja objekta
- posebno naglašeno početno stanje
- posebno naglašeno završno stanje – može biti i više završnih stanja
- prelaske između stanja
    - strelica od prethodnoh prema narednom stanju
    - nazivi događaja koji menjaju stanje objekta
    - odgovarajuća objašnjenja

Na slici Slika 11. prikazan je primer procedure upisa na fakultet upotrebom dijagrama stanja. Treba napomenuti da na dijagramu postoje dva završna stanja, kada je fakultet upisan i kada se odustalo zbog neplaćanja troškova oko upisa za samofinansirajuće studente ili ako se prijemni ispit nije položio.



Slika 11. Primer dijagrama stanja: Upis na fakultet

Na slici Slika 12 prikazan je primer dijagrama stanja studenta u procesu upisivanja na fakultet. Student prolazi kroz 4 stanja : prijavljen, prihvaćen, upisan i mirovanje , a moguća su 4 završna stanja: odbijen, odustao, diplomirao i ispisan.



Slika 12. Primer dijagrama stanja

Dijagrami slučajeva upotrebe. Definicija slučaja upotrebe je : "Slučaj upotrebe (Eng. *Use case*) je specifikacija skupa akcija koje vrši sistem, koje proizvode vidljiv rezultat koji je, po pravilu, od vrednosti za jednog ili više učesnika u sistemu."

Dijagrami slučajeva upotrebe služe za grub opis funkcionalnosti posmatranog sistema ili posmatranog dela organizacije. U principu se može konstatovati da postoje dve vrste ovih dijagrama: dijagrami slučajeva upotrebe (Eng. *Use Case Diagrams*) i dijagrami slučajeva upotrebe poslovnog procesa (Eng. *Business Use Case Diagrams*). Dijagrami slučajeva upotrebe treba da daju odgovor na pitanje „**Šta sistem radi?**", dok dijagrami slučajeva upotrebe poslovnog procesa treba da daju odgovor na pitanje „**Šta organizacija radi?**". Pomoću dijagrama slučajeva upotrebe predstavljaju se funkcije sistema koje će biti automatizovane, a pomoću dijagrama slučajeva upotrebe poslovnog procesa i automatizovane i manuelne funkcionalnosti.

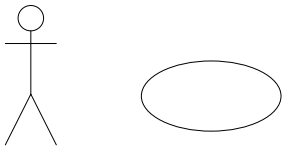
Slučajevi upotreve daju odgovor na pitanje kako učesnici interaguju sa sistemom i opisuje akcije koje sistem izvodi.

Sastavni delovi dijagrama Slučajeva upotrebe su:

- scenariji (Eng. *use cases*) - opisuju karakteristične sekvence akcija u tipičnim situacijama korišćenja sistema, odnosno predstavljaju funkcije koje sistem obavlja;
- nosioci uloga, učesnici (Eng. *actors*) - osobe ili veštački entiteti koji učestvuju u scenariju;
- interakcije - vrste aktivnosti koje se vrše u međusobnoj komunikaciji nosioca uloga;

Učesnici predstavljaju nekoga ili nešto što se nalazi izvan sistema ili organizacije (u zavisnosti od vrste dijagrama), a u interakciji je sa njim. Uloge su dodatni elementi i koriste se samo prilikom izrade dijagrama slučajeva upotrebe poslovnog procesa i predstavljaju nekoga ili nešto što se nalazi unutar organizacije i u interakciji je sa funkcionalnostima posmatranog dela organizacije. Slučajevi upotrebe i slučajevi upotrebe poslovnog procesa služe da bi se prikazale konkretne funkcionalnosti sistema, odnosno organizacije. Ovi dijagrami predstavljaju vodilju za kompletan proces razvoja softvera, pa se često za razvoj zasnovan na UML-u kaže da je usmeravan slučajevima upotrebe.

Akteri i slucajevi upotrebe se predstavljaju na sledeći način:



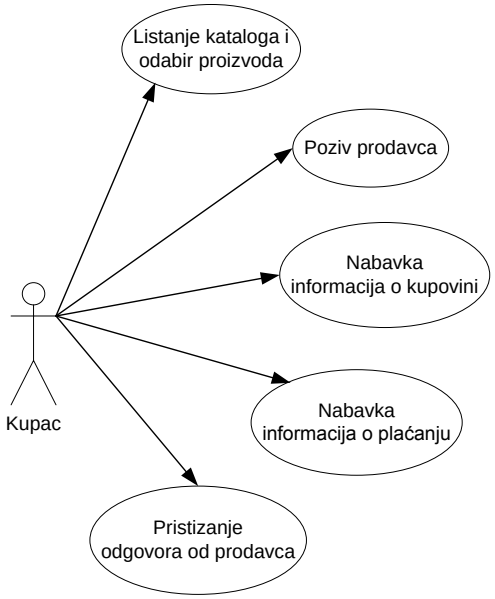
Svakog učesnika je potrebno imenotavi , ime se navodi odmah ispod njegovog grafičkog simbola. Takođe je potrebno imenovati slučajeve upotrebe, ime se navodi unutar elipse.

Use Case dijagrami su jednostavni za crtanje. Na slici Slika 13 dat je jednostavan dijagram slučaja upotrebe na primeru izabira nekog proizvoda od strane kupca. Kreće se od kreiranja spiska koraka koje kupac treba da izvrši pri narudžbi nekog proizvoda, na primer:

1. listanje kataloga i izabir proizvoda
2. poziv prodavca
3. nabavka informacija o kupovini
4. nabavka informacija o plaćanju

5. pristizanje odgovora od prodavca

Ovi koraci zapravo generišu jednostavan Use Case dijagrama prikazan na slici Slika 13. Ovaj primer prikazuje kupca kao učesnika sistema. Dijagram prikazuje korake (gore navedene) kao akcije koje treba da preduzme kupac. Takođe, i prodavac može da se uključi kao učesnik sistema, jer i on vrši interakciju sa sistemom narudžbine. Na osnovu ovog jednostavnog primera moguće je izvesti zahteve sistema za narudžbinu. Sistem treba da je u mogućnosti da izvede sve akcije predstavljene use case-ovima.



Slika 13. Primer dijagrama slučajeva upotrebe

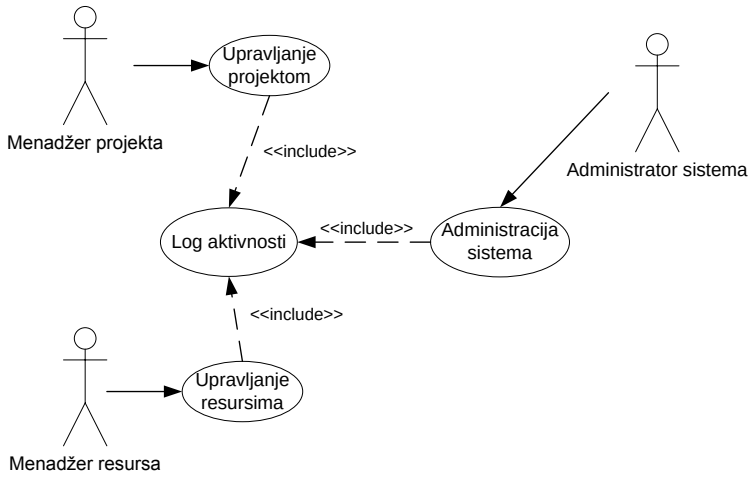
Elipsa se može povezivati sledećim vezama, odnosno zavisnostima:

- Uključivanje (Eng. *Include*) - ova zavisnost povlači se od jednog slučaja upotrebe (koji se naziva osnovni) ka drugom (koji se naziva uključeni) i označava da osnovni slucaj upotrebe uključuje ili poziva uključeni. Jedan može da pozove veći broj slučajeva upotrebe.

- Proširivanje (Eng. *Extend*) – ovakva veza između dva slučaja upotrebe znači da je osnovni slučaj upotrebe proširen osnovnim ponašanjem proširujućeg slučaja upotrebe.

- Generalizacija - Generalizacija između slučajeva upotrebe je isto što i generalizacija između klasa. Jedan slučaj upotrebe može biti specijalizovan na više različitih koji nasleđuju ili dodaju osobine. Prikazuje se isto kao generalizacija kod klasa (linija sa trouglom na kraju).

Primer uključivanje zavisnosti prikazana je na slici Slika 14. Log aktivnost slučaj upotrebe je glavni za upravljanje projektom, upravljanje resursima i administracija sistema i uključen je od strane tih use slučajeva upotrebe.



Slika 14. Primer dijagrama sa vezom uključivanja

3.3. Dijagrami interakcije

Dijagrami interakcija, kao podskup dijagrama ponašanja, prikazuju tok kontrola i podataka u sistemu, modeluju ponašanje slučajeva upotrebe opisujući način kako grupa objekata interaguje u cilju završavanja nekog zadatka. U ovu grupu spadaju:

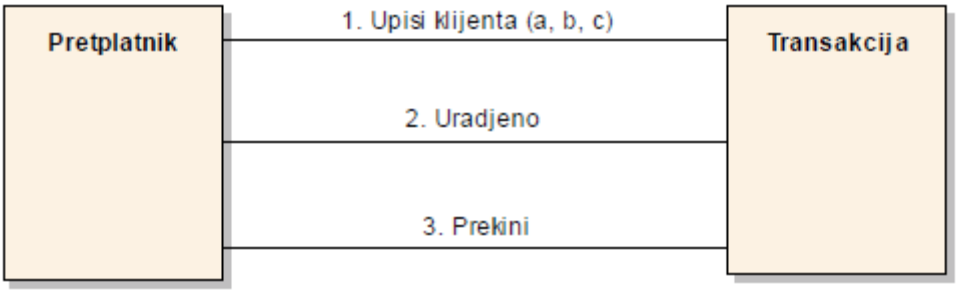
- Dijagram saradnje ili Komunikacioni dijagram (Eng. *Collaboration /Communication diagram*)
- Pregled interakcija dijagram (Eng. *Interaction overview diagram*)
- Sekvencijalni dijagram (Eng. *Sequence diagram*)

- Vremenski dijagram (Eng. *UML Timing Diagram*)

Dijagrami saradnje. Dijagrami saradnje su semantički ekvivalentni dijagramima sekvence. Pomoću njih se ističe strukturna organizacija objekata koji učestvuju u interakciji. Za razliku od dijagrama sekvence, dijagrami saradnje ne sadrže liniju života objekta niti fokus upravljanja, već sadrže objekte iz interakcije sa vezama koje sadrže poruke izmenu njih.

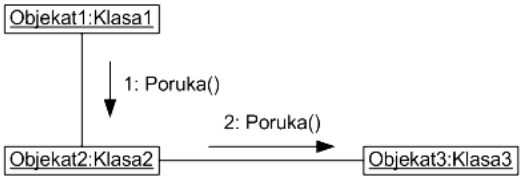
Objekti se prikazuju kao čvorovi grafa, a poruke izmenu ovih objekata prikazuju se kao grane grafa. Poruke mogu imati svoje redne brojeve (sa prefiksom), a koji označavaju redosled razmene poruka. Ovaj redosled formira putanju poruka, koja se takone označava i usmerenim strelicama na granama grafa.

Na slici Slika 15 može se videti primer jednog ovakvog dijagrama. Objekti su transakcija i pretplatnik dok cu poruke koje razmenjuju naznacene na linijama, kao i njihov odgovarajući redosled.



Slika 15. Primer dijagrama saradnje

Na slici Slika 16 prikazani su osnovni elementi: objekti i poruke koje se prosleđuju između njih.



Slika 16. Primer dijagrama saradnje

Dijagrami pregleda interakcija. Dijagrami pregleda interakcija su jedna od novina verzije 2.0. Predstavljaju kombinaciju dijagrama aktivnosti i dijagrama sekvenci. Mogu se posmatrati kao dijagrami aktivnosti u kojima su aktivnosti zamenjene sa dijagramima sekvenci.

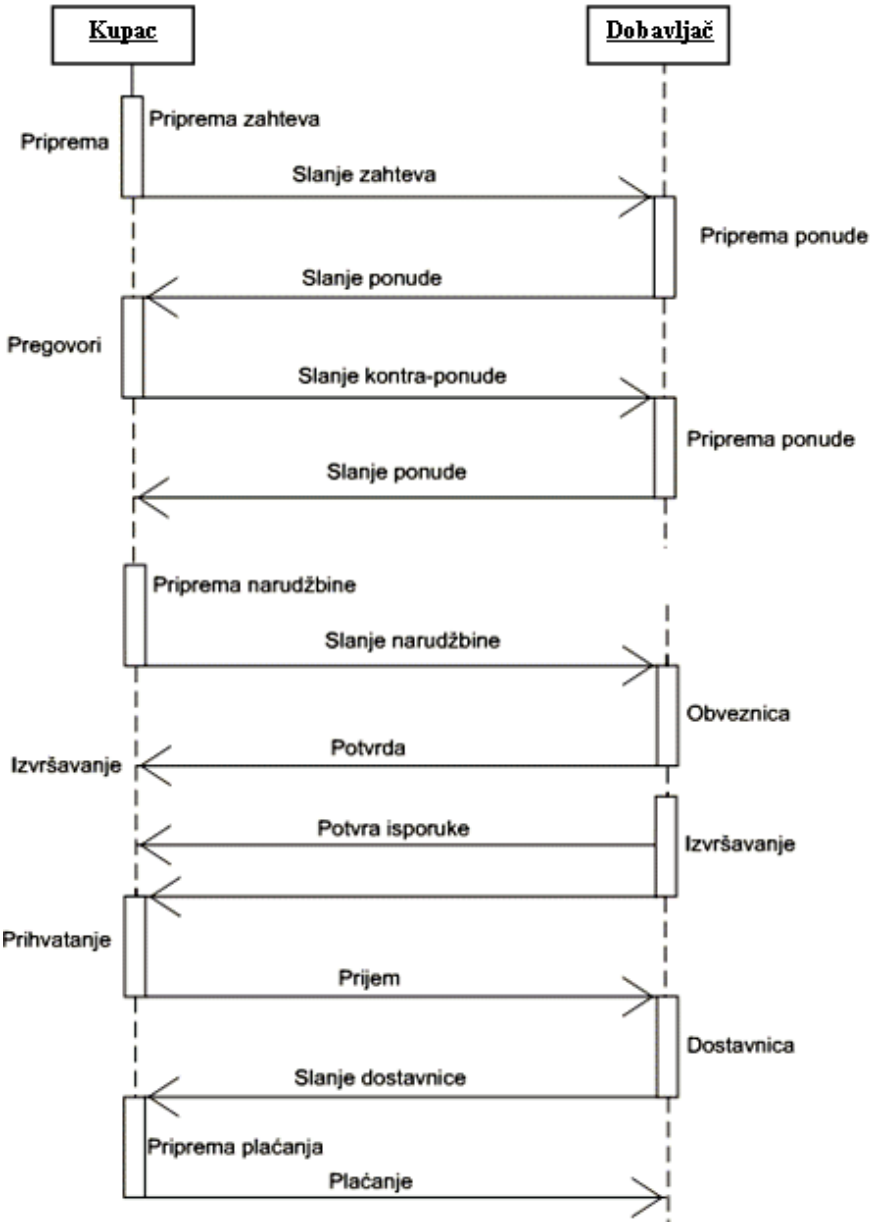
Vremenski dijagrami. Vremenski dijagrami su takođe novina u verziji 2.0. Iako se odavno koriste pri rešavanju elektrotehničkih problema, tek su u verziji 2.0 uvršteni u UML. Ova vrsta dijagrama je slična dijagramima stanja, sa tom razlikom što se vreme pojavljuje kao inicijator promene stanja objekta. Pored mogućnosti praćenja promena stanja jednog objekata moguće je pratiti i uporenivati promenu stanja više objekata. Dakle, ovi dijagrami prikazuju stanja u koja objekti dolaze nakon unapred predefinisanog vremenskog intervala.

Dijagram sekvence. Dijagram sekvence predstavlja specijalni slučaj dijagrama interakcije, a njime se prikazuje skup poruka razmenjenih između objekata koji sarađuju da bi se realizovala određena operacija ili dobio neki rezultat. Ovaj dijagram se prikazuje u dve dimenzije: vertikalna dimenzija prikazuje vreme, a horizontalna određene objekte. Na dijagramu se prikazuju poruke koje se razmenjuju u definisanom vremenskom redosledu između uočenih objekata. Među objektima se uspostavljaju veze, a dijagramom se prikazuje linija života svakog prikazanog objekta. Linije života pokazuju u kojim periodima je taj objekat aktivan. Isto tako ovi dijagrami sadrže fokus upravljanja kojim se prikazuje vremenski period u kojem objekat vodi neku akciju.

Elementi dijagrama sekvenci se predstavljaju na sledeći način:

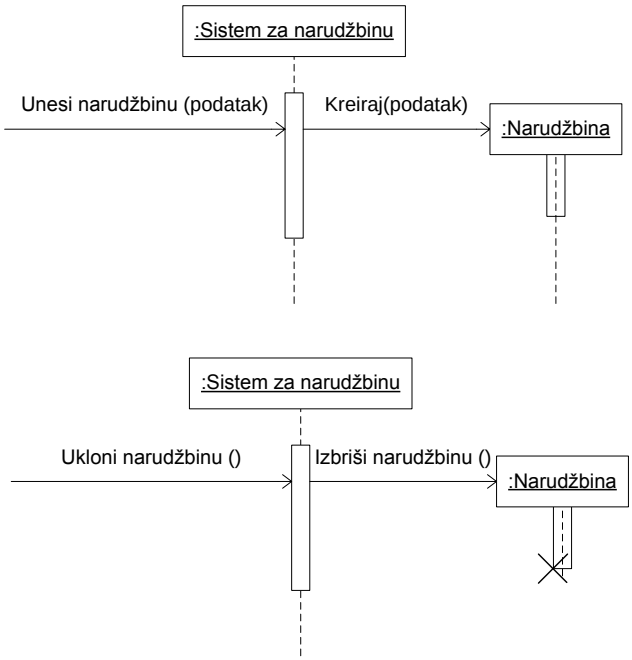
- objekti se prikazuju na ranije opisani način,
- veze izmenu objekata su poruke, koje se prikazuju položenim usmerenim linijama,
- linija života objekta je vertikalna isprekidana linija,
- fokus upravljanja se predstavlja uzanim, vertikalno postavljenim pravougaonikom.

Na slici Slika 18 vidimo da su objekti smešteni na vrhu modela a da su linije nacrtane vertikalno od objekta. One pokazuju kada su objekti napravljeni i uništeni – tzv. linije života.



Slika 17. Interakcija između kupca i dobavljača

Slika 18. pokazuje kako su objekti napravljeni i uništeni za vreme izvršenja sekvence. Poruke u sekvencijalnom dijagramu mogu da imaju parametre kao što je prikazano na slici Slika 18.



Slika 18. Kreiranje i uništavanje objekta u sekvencijalnom dijagramu

3.4. Kratak pregled upotrebe po fazama razvoja

Opisani UML dijagrami mogu se koristiti u različitim fazama razvoja softvera. U fazi analize najčešće se koriste: dijagrami slučajeva upotrebe, dijagrami aktivnosti i

dijagrami stanja mašine, dok se u fazi projektovanja uobičajeno koriste: dijagrami klasa, dijagrami sekvenci i saradnje, dijagrami paketa, dijagrami rasporenivanja.

4. Zaključak

U ovom radu je sažeto, a opet kompletno predstavljen jedan od najkorišćenijih jezika u svetu razvoja softvera – UML. Data je kratka istorija UML-a, navedeni su osnovni ciljevi kojima su se vodili kreatori ovog jezika i ono što je najvažnije, urađen je pregled svih dijagrama, a ne samo onih najčešćih kao što je često slučaj, jer smatramo da su svi dijagrami bitni – inače bi ih sami autori isključili. Sledi par napomena vezanih za upotrebu ovog rada. Najbolje i najefektivnije bi bilo pročitati ga u celini, međutim svako od podpoglavlja, čak i većina pasusa, je pisano kao mala celina za sebe što omogućava da se ovaj rad koristi i kao omanja referenca i podsetnik pri svakodnevnom radu softverskih inženjera i arhitekti. Na samom kraju daćemo i smernice za nastavak istraživanja na ovu temu. Pravac kojim treba ići je pravac izučavanja razvojnih procesa i toga kako efektivno i minimalno proširiti UML tako da se dobije bolji alat za predstavljanje samih poslovnih procesa koji su ipak još uvek jako bitni i to pogotovu na samom početku razvoja novih sistema.

Literatura

1. Grady Booch , James Rumbaugh, Ivar Jacobson: UML vodič za korisnike (2000. god.)

2. James Rumbaugh, Ivar Jackobson, Grady Booch: The Unified Modeling Language Reference Manual 2nd edition, Addison-Wesley Professional ((July 29, 2004)

3. Profesor Saša Malkov : Materijali sa predavanja (Oktobar 2013), http://poincare.matf.bg.ac.rs/~smalkov/download.html?dpth=1&cap=Informacioni+sistemi&bp=is.r271.2013%2Fpublic&rp=%2Fpredavanja

4. Asistent Staša Vujicić Stankovic: Materijali sa vežbi (Oktobar 2013) , http://poincare.matf.bg.ac.rs/~stasa/IS.html

5. Vikipedija enciklopedijski projekat slobodnog sadržaja na internet koji razvijaju dobrovoljci uz pomoć viki softvera. [Online]. Dostupno na: http://en.wikipedia.org/wiki/Unified_Modeling_Language (Maj 2014)