Najčešće korišteni UML dijagrami u programskom inžinjerstvu

Đaić, Katarina

Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko- komunikacijske tehnologije

KATARINA ĐAIĆ

NAJČEŠĆE KORIŠTENI UML DIJAGRAMI U PROGRAMSKOM INŽENJERSTVU

Završni rad

Pula, rujan 2017.

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko- komunikacijske tehnologije

KATARINA ĐAIĆ

NAJČEŠĆE KORIŠTENI UML DIJAGRAMI U PROGRAMSKOM INŽENJERSTVU

Završni rad

JMBAG: 0303054563, redoviti student
Studijski smjer: Sveučilišni preddiplomski studij informatika
Kolegij: Programsko inženjerstvo
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc.dr.sc. Tihomir Orehovački

Pula, rujan 2017.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana, Katarina Đaić, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, 2017.

IZJAVA

o korištenju autorskog djela

Ja, Katarina Đaić, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Najčešće korišteni UML dijagrami u programskom inženjerstvu“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____2017.

Potpis

_____

Sažetak

U ovom završnom radu, pod naslovom “Najčešće korišteni UML dijagrami u programskom inženjerstvu” je objašnjeno modeliranje sustava. Ono predstavlja proces razvoja modela sustava pomoću grafičkog jezika UML. On obuhvaća 14 dijagrama od kojih je jedan dijagram pomoćni. Oni se dijele u tri vrste, a to su dijagrami ponašanja, strukturni dijagrami i dijagrami međudjelovanja. Ti se dijagrami prvenstveno razlikuju po prethodno navedenoj podjeli, ali i također po elementima pomoću kojih se crtaju. U radu će biti objašnjeni i nacrtani u alatu “Lucidchart” vlastiti primjeri od sljedećih dijagrama, a to su dijagram aktivnosti, dijagram korištenja, sekvencijski dijagram, klasni dijagram i dijagram stanja. Oni se najčešće koriste u programskom inženjerstvu.

Ključne riječi: modeliranje sustava, UML, dijagrami ponašanja, strukturni dijagrami, dijagrami međudjelovanja, Lucidchart, dijagram aktivnosti, dijagram korištenja, sekvencijski dijagram, klasni dijagram, dijagram stanja

Abstract

In this bachelor's thesis "The most commonly used UML diagrams in software engineering", system modeling was explained. It presents the system development process by means of UML graphical language. It includes 14 diagrams, from which one is supplementary. They are gruped into three types, which are behavior diagrams, structural diagrams and interaction diagrams. Primary difference of these diagrams is mentioned in text above but also they are divided by the elements they are drawn with. In this thesis the author has drawn, with the help of "Lucidchart" tool, her own examples of following five diagrams: activity diagram, use case diagram, sequence diagram, class diagram and state machine diagram. They are most commonly used in software engineering.

Keywords: system modeling, UML, behavior diagrams, structural diagrams, interaction diagrams, Lucidchart, activity diagram, use case diagram, sequence diagram, class diagram, state machine diagram

Sadržaj

Uvod

Danas se modeliranje gotovo koristi u svim ljudskim djelatnostima. Ono predstavlja razvoj modela sustava, odnosno njegovu izgradnju koja se prikazuje u obliku dijagrama.

Modeliranje se smatra široko prihvaćenom inženjerskom disciplinom. Posebice, samo modeliranje se često koristi jer olakšava posao pojedinim osobama. Primjerice, to može biti osoba koja je po zanimanju arhitekt te mu sama izrada dijagrama uvelike pomaže kod projektiranja neke zgrade ili kuće. Pritom se koriste razni matematički modeli kako bi se primjerice, analizirali utjecaji vjetra ili potresa. Naravno, modeliranje nije vezano samo za građevinsku industriju, te da nema njega, bilo bi gotovo nemoguće proizvesti automobil ili avion.

Modeliranje se ne koristi samo kod velikih sustava, već se koristi i kod malih sustava (imaju koristi od njega). U slučaju da se koristi kod velikih sustava, onda je bitno da oni budu što veći i kompleksniji jer će tada biti veća važnost samog modeliranja.

Postoje 4 faze modeliranja kod procesa obrade, a to su odabir tehnike modeliranja, definiranje uzorka (za testiranje), proces izrade modela i ocjena kvalitete. Problem odabira tehnike modeliranja se nazire kod definiranja problema. No, ako postoji dobro definiran skup podataka, onda se može odabrati i neka druga, odnosno prikladnija tehnika od one koju smo imali definiranu na početku.

Sami modeli se koriste u raznim fazama razvoja softvera. Prvo su se pojavili unutar utvrđivanja zahtjeva (kod analize zahtjeva). Ti modeli se onda dalje sređuju kod oblikovanja, a potom se implementiraju kada se pretvaraju u opis sustava. Poslije implementacije, modeli se koriste u svrhu same građe te načina rada sustava.

Modeli sadržavaju vrlo detaljne nacrte koji daju prikaz samo većih cjelina. Dobrim modelom se smatra onaj model koji uključuje elemente koji imaju veliki utjecaj, dok zanemaruje one manje elemente koji su manje bitni. Isto tako, on sam prepoznaje koji su bitni elementi nekog sustava, te se stoga dolazi do pojma vizualnog modeliranja koji služi za prepoznavanje poslovnih objekata, analiziranje i oblikovanje programske podrške i rješavanje kompleksnosti.

U prošlosti su se koristile razne notacije, ali danas je prihvaćen UML.

Do UML- a je došlo zbog toga što su korisnici i poslovni analitičari govorili jednim programskim jezikom, dok su primjerice, programeri govorili nekim drugim jezikom. Na taj način je svaka grupa ljudi pričala drugim jezikom te su oni u svakoj razvojnoj fazi životnog ciklusa softverskog proizvoda koristili svoj način opisivanja nekog procesa. Dosta se vremena gubilo na prijenosu samih informacija te je tako onda svaka grupa preuzimala rezultate od prethodne grupe. Po dospijeću rezultata, morali su ih „prevesti“ kako bi ih uopće mogli razumjeti (primjerice, složeni projekti). Zbog toga je UML onda objedinio sve te jezike u jedan jezik (često korišten).

UML se može koristiti za modeliranje sustava kod poslovnih informacijskih sustava te kod distribuiranih mrežnih aplikacija. U slučaju da se koristi kod distribuiranih mrežnih aplikacija, onda ti sustavi rade u realnom vremenu.

UML je jezik koji se bavi svim pogledima na sustav koji pritom služi za pokretanje i za razvoj, što zapravo znači da predstavlja samo dio metode razvoja programske podrške. Pod programskom podrškom se misli na područje primjene, a to su primjerice, poslovni informacijski sustavi, financije i bankarstvo, telekomunikacije, znanost, prodaja, transport te distribuirane mrežne usluge. Međutim, UML nije ograničen samo na modeliranje programske podrške (mada se tamo najčešće upotrebljava), već se koristi i za ostale sustave (primjerice, tokovi podataka, zdravstvo te dizajn sklopovlja).

Ovaj grafički jezik ne ovisi o procesu, ali bi se trebao koristiti u procesima koji se temelje na slučajevima korištenja (iterativno i inkrementalno).

U ovom završnom radu, pisat će se o jednom dijelu izgradnje samog softvera (modeliranje sustava) te o najčešće korištenoj notaciji (UML). Važno je tako naglasiti da se UML koristi za crtanje dijagrama (obuhvaća ih 14) te za skup koncepata. On služi za konstrukciju (povezivanje s programskim jezicima), vizualizaciju (obuhvaća grafičke modele), specifikaciju (modeli su precizni, nedvosmisleni i spremni u donošenju odluka, primjerice, faze analize, dizajna i implementacije) i dokumentiranje sustava (dijagrami sadrže sve informacije koje je skupio tim pritom međusobno razgovarajući te dijeleći prikupljeno znanje). UML pritom pruža planiranje i funkcije sustava te konkretne stvari (primjerice, klase pisane u određenom programskom jeziku), sheme baze podataka i programske komponente (mogu se ponovno

upotrebljavati). Tako on ne pripisuje nikakav određeni pristup kod rješavanja problema, nego se prilagođava svakom pristupu. Kao što svaki jezik ima definiranu odgovarajuću sintaksu, tako je ima i UML. Njegova sintaksa svodi se na grafičku notaciju, niz pravila vezanih za dijagrame i semantiku. Sama semantika zapravo zahtjeva najviše napora (kod postojećih i novih koncepata). Nakon modeliranja, detaljno će biti opisano 14 dijagrama te nacrtani primjeri od 5 dijagrama.

U prvom poglavlju će biti opisano modeliranje pomoću grafičkog jezika UML-a. U njegovim podpoglavljima će biti navedena povijest UML- a, cilj UML- a te koncepti.

Drugo poglavlje je bazirano na vrstama modela (modeli postojećeg sustava te modeli za razvijanje novog sustava), načinima njihova korištenja i podjeli UML- a (podjela dijagrama).

U trećem poglavlju su opisani dijagrami ponašanja. Prvo je objašnjen dijagram korištenja, njegovi elementi te je nacrtan primjer. Potom je objašnjen dijagram aktivnosti, njegove karakteristike, elementi te primjer. Treće je naveden dijagram stanja, njegove karakteristike, elementi te primjer.

U četvrtom poglavlju će biti objašnjeni strukturni dijagrami. Prvo će biti opisan dijagram klasa ili razreda, njegove karakteristike, odnosi između razreda, postupak pridruživanja, agregacija i kompozicija, atributi, operacije, nasljeđivanje, ovisnost, sučelje i realizacija, tipovi podataka i enumeracija, komentari i vlastiti primjer. Potom su objašnjeni dijagrami objekata, tj. njegove karakteristike, definiranje i veze između objekata. Kod dijagrama komponenata i dijagrama složene strukture su navedene karakteristike i elementi. Kod paketnog dijagrama su definirane karakteristike paketa, vidljivost i ugnježđenje paketa, veze paketa te stereotipovi. Kod dijagrama rasporeda su navedene karakteristike i odnosi između razreda, dok su kod dijagrama profila opisane karakteristike i elementi.

Posljednja vrsta dijagrama, koja je navedena u radu, dijagrami su međudjelovanja. Kod dijagrama komunikacije, vremenskog dijagrama i dijagrama pregleda međudjelovanja su objašnjene njihove karakteristike i elementi, dok je kod dijagrama slijeda, osim karakteristika i elemenata, još nacrtan i primjer.

1. Modeliranje sustava- UML

Modeliranje je proces razvoja apstraktnih modela sustava. (Manger, 2013., str.41). Ono se smatra glavnim dijelom svih aktivnosti koji vodi do dobre programske podrške, produkcije ili isporuke. Nakon što završi postupak produkcije, vrši se vizualizacija i kontroliranje arhitekture sustava te dolazi do izgrađivanja sustava. Stoga, modeliranje se koristi iz dva razloga, a to su pojednostavljenje sustava i ponovna iskoristivost dijelova istog sustava.

Modeliranje spada pod inženjerskom disciplinom (primjerice, arhitekt). On koristi modeliranje i projektiranje neke zgrade da sebi olakša posao. Isto tako, modeliranje je vezano i za računalne modele, fizičke modele i za izradu prototipova.

Modeliranje se odvija iterativno u dva smjera, a to su:

1. Konceptualni ili dizajn sustava (što će sustav raditi?)
2. Tehnički dizajn (kako će sustav izgledati?)

Dizajn sustava se odnosi na funkcije sustava (pisan je jezikom klijenta), ne sadrži tehničke izraze, ne ovisi o implementaciji, povezan je s dokumentima specifikacije zahtjeva i objašnjava vidljive (vanjske) karakteristike sustava. Tehnički dizajn se odnosi na oblik ili karakteristike sustava, služi za opis građe hardvera, potrebe softvera i sučelja, ulaze i izlaze iz sustava, mrežnu arhitekturu te strukturu i tokove podataka.

Modeli koji predstavljaju sustav s različitih pogleda su:

1. Vanjska perspektiva
2. Unutarnja perspektiva
3. Perspektiva ponašanja
4. Strukturna perspektiva

Pod vanjskom perspektivom se podrazumijeva prikazivanje modela konteksta, sustava i okoline sustava. Unutarnja perspektiva obuhvaća objašnjavanje prikazivanja procesa u nekom sustavu. Perspektiva ponašanja služi za opisivanje modela ponašanja sustava. Strukturna perspektiva definira prikazivanje modela arhitekture sustava ili strukturu samih podataka.

Definiranje modela služi za nacrte koji će kasnije služiti za izgradnju nekog sustava. No, sustavi se opisuju koristeći više modela i postižu semantički zatvorenu apstrakciju sustava.

Modeli se koriste za razne faze razvoja softvera, a pojavili su se kod:

1. Postupka utvrđivanja zahtjeva
2. Oblikovanja
3. Građe sustava
4. Načina rada sustava
5. Dokumentacije

Postupak utvrđivanja zahtjeva obuhvaća analizu zahtjeva. Oblikovanje služi za pretvaranje u opis sustava gdje se koristi postupak implementacije.

Modeliranje se koristi i za velike i za male sustave (imaju koristi od njega). Pritom se zahtjeva veća važnost modeliranja u slučaju da se ono koristi za veći sustav.

Modeliranje obuhvaća 4 cilja s aspekta modela, a to su:

1. Pomaganje kod vizualizacije sustava
2. Omogućavanje specifikacije ili ponašanja sustava
3. Pružanje predložaka
4. Dokumentiranje odluka

Pomaganje kod vizualizacije sustava obuhvaća postupke promatranja i izgrađivanja. Pružanje predložaka služi za izgradnju sustava s pogleda vodiča. Dokumentiranje odluka specificira odluke koje su donijete.

Dva najčešća načina pristupa modelu su:

1. Pristup sa strane algoritama
2. Objektno- orijentirani pristup

U slučaju kada se koristi pristup sa strane algoritama, koriste se procedure, odnosno funkcije. On je najviše fokusiran na kontrolu i dekompoziciju većih dijelova u manje, te je kod njega uočen veliki nedostatak kod promjene zahtjeva na sustav. Upravo iz tog razloga, dolazi do problema održavanja.

Modernim pristupom se smatra objektno- orijentirani pristup. On sadrži objekte i klase gdje klasa služi za opis više kolekcija objekata, te ona u tom kontekstu, ima ista te zajednička svojstva. Za razliku od klase, svaki objekt se pritom imenuje i razlikuje od ostalih objekata, sadrži stanje (pridruženi podaci) i ponašanje (izvršavanje operacija nad tim i drugim objektima). Objektno- orijentirani pristup obuhvaća sve razine kompleksnosti. Stoga, većina se modernih jezika, operativnih sustava i alata smatra objektno orijentiranim.

Razvoj tehnika modeliranja se prikazuje:

1. U obliku strukturalnih tehnika
2. U obliku strukturalne analize procesa

Oblik strukturalne analize procesa obuhvaća funkcionalne i objektno- orijentirane metode.

Nakon razvijanja tehnika modeliranja, dolazi do CASE (engl. Computer- Aided System ili Software Engineering) alata i do UML-a (engl. Unified Modeling Language). CASE alati služe za potrebe programiranja softvera, dok UML ujedinjuje sve postojeće notacije koristeći modeliranje softverskih elemenata (sredstvo komunikacije).

UML je grafički jezik koji služi za:

1. Vizualizaciju
2. Specifikaciju
3. Konstruiranje
4. Dokumentiranje sustava programske podrške

Pod vizualizacijom se podrazumijeva obuhvaćanje grafičkih modela s preciznom semantikom. Specifikacija definira sadržavanje modela koji su precizni, nedvosmisleni i spremni u predstavljanju značajnih odluka (primjerice, postupci, analize, dizajna i implementacije). Konstruiranje obuhvaća povezivanje modela s programskim jezicima direktno, te omogućava direktno i reverzno inženjerstvo. Dokumentiranje sustava programske podrške služi za predstavljanje dijagrama sa svim informacijama, koje je skupio razvojni tim, pritom donoseći komunikaciju i dijeljenje prikupljenog znanja.

UML se smatra i objedinjenim vizualnim jezikom gdje služi za poslovno i softversko modeliranje u svim fazama razvoja, za sve tipove sustava i za opće modeliranje (bilo koje konstrukcije) u definiranju statičke strukture i dinamičkog ponašanja.

UML obuhvaća tri osnovna modela, a to su:

1. Funkcionalni model
2. Objektni model
3. Dinamički model

Kod funkcionalnog modela se prikazuje funkcionalnost sustava sa strane korisnika (korištenje dijagrama korištenja). Objektni model služi za prikaz strukture ili njegove podstrukture sustava (primjerice, objektni i klasni dijagrami, atributi, operacije, veze, itd.). Dinamički model prikazuje ponašanje sustava, odnosno dijagrame tijeka i aktivnosti.

Postavlja se pitanje: „Zašto uopće koristiti standardni jezik za modeliranje“? Dakle, iz razloga što softverski sustavi postaju sve veći i veći. U slučaju da se to dogodi, onda programeri mogu stvoriti loš dizajn, te im je potrebno veće vrijeme za izradu samog softvera, pa je većina sustava dokumentirana dijagramima korištenja. Oni pružaju poglede na strukturu i funkcionalnost koju bi bilo teško shvatiti gledajući samo kodove ili tekstualne opise. Drugim riječima, dijagrami pružaju apstrakciju te je potrebno koristiti standardni jezik za modeliranje, odnosno UML. (Lethbridge i Langaniere, 2005., str. 170-171).

Prema Lethbridgeu i Langanieru (2015.) upotreba UML- a, dobro definiranog standarda za modeliranje, dodaje dodatne prednosti (Lethbridge i Langaniere, 2005., str. 171):

1. Standardna notacija
2. Postojanje širokog raspona alata za izradu modela UML-a

Standardna notacija definira svakog tko radi na modelu na način da ga svi mogu interpretirati na isti način. Širok raspon alata za izradu modela UML-a omogućuje simulaciju, animaciju i/ili generiranje koda za sve sustave ili samo za neke dijelove.

UML omogućuje određivanje radnje kod pojedinih dijelova sustava. Usprkos njegovom nazivu, UML nije jedinstveni jezik iz razloga što on koristi više vrsta dijagrama koji služe

za predstavljanje različitih dijelova sustava. Neki od njih predstavljaju programske stavke (primjerice, klase), dok drugi predstavljaju ponašanje (primjerice, objekti koji međusobno djeluju i način na koji podaci „protječu“ kroz sustav). (Stephens, 2015., str. 77).

Najveći UML-ov nedostatak je njegova kompleksnost, a razlog tomu su kategorije dijagrama. Kategorije dijagrama su podijeljene u više od desetak specifičnih tipova, svaki sa svojim kompleksnim skupom pravila. Određivanje složenih zahtjeva kod UML-a je korisno u slučaju kada ga korisnik razumije u potpunosti. Nažalost, veći broj korisnika ga ne želi naučiti. (Stephens, 2015., str. 77).

UML je povezan s UP- om (engl. Unified Process) i oni se skladno nadopunjuju. UP u ovom kontekstu, služi za određivanje procesa razvoja softvera, dok UML omogućuje da se rezultati tog razvoja softvera dokumentiraju. UML i UP su nastali zajedno (glede istog projekta). Tako je UML dio projekta koji se odnosi na vizualizaciju, dok se UP odnosi na proces. Razlika između njih se krije u tome što je UML prihvaćen kao svjetski standard, dok UP nije. (Manger, 2015a, str. 5).

UML ima semantička pravila koja su vezana za:

1. Imena
2. Doseg
3. Vidljivost
4. Integritet
5. Izvođenje

Kod imena se imenuju stvari, veze i dijagrami. Doseg daje značenje imenu. Vidljivost omogućuje prikaz načina vidljivosti imena drugih korisnika. Integritet služi za povezivanje stvari na pravi način. Izvođenje definira pokretanje ili simuliranje modela.

Uključuju se različite vrste modeliranja, a to su primjerice:

1. Modeli koji olakšavaju razumijevanje poslovnih procesa
2. Odvijanje tokova događaja
3. Sekvence upita
4. Aplikacije
5. Baze podataka

UML koristi i kategorije korisnika, kao što su to:

1. Sistem-analitičari i krajnji korisnici
2. Arhitekti sustava
3. Razvojni inženjeri
4. Kontrolori kvalitete
5. Rukovodioci projekta ili menadžeri

Sistem- analitičari i krajnji korisnici vrše zahtijevanje strukture i ponašanje sustava. Arhitekti sustava zadovoljavaju zahtjevima sustava. Razvojni inženjeri transformiraju arhitekturu u izvršni kod. Kontrolori kvalitete provjeravaju strukturu i ponašanje sustava. Rukovodioci projekta (menadžeri) vode i usmjeravaju kadrove i resurse.

1.1 Povijest UML-a

Dolazi do pojave objektno- orijentiranog programiranja sredinom i krajem 1980-ih godina. Razlog tomu je nova vrsta objektno- orijentiranog programskog jezika i povećana složenost aplikacija. Zbog toga dolazi do početka eksperimentiranja s pristupima, odnosno do analize i dizajna.

Od 1989. do 1994. godine dolazi do izrazitog povećanja objektno- orijentiranih metoda (s manje od 10 na nešto više od 50 metoda). Ovdje se javlja problem kod korisnika u korištenju tih metoda (primjerice, u traženju prikladnog jezika za modeliranje).

Zbog prethodno navedenih razloga nastaju nove metode, a jedna od najpoznatijih je OOSE (engl. Object- Oriented Software Engineering). Tu metodu su stvorili Booch i Jacobson, te je ona predstavljala jezik ili metodologiju koja je služila za modeliranje. OOSE se smatrala dosta utjecajnom i koristila se za dizajn, fazu izrade samog projekta, te je obuhvaćala dijagrame slučaja korištenja (postavljanje zahtjeva na sustav), analizu i dizajn visoke razine.

Druga poznata metoda je OMT (engl. Object Modeling Technique) koju je stvorio Rumbaugh. Ona je spadala među modele objektnog modeliranja, koja je osim za modeliranje, služila i za projektiranje. Verzija OMT-2 se koristila za analizu informacijskih sustava kod velikih zahtjeva za obradu podataka.

Prethodno navedene metode su bile napredne, a postojale su još neke kompletne metode (koje su manje popularne), a to su Fusion, Shlaer- Mellor i Coad- Yourdon.

Sredinom 1990-ih godina, došlo je do formiranja ideja Ivara Jacobsona (tvrtka Objectory), Grady Boocha (tvrtka Rational Software Corporation) i Jamesa Rambaugh-a (tvrtka General Electric) koji su međusobno počeli surađivati te usvajati ideje. Tako se od njihovih ideja htjela napraviti daljnja stabilizacija objektno- orijentiranog tržišta, te se htjelo omogućiti pri izradi projekta korištenje jednog programskog jezika za modeliranje (stvaranje nove i naprednije metode).

1994. godine u listopadu, došlo je do razvijanja UML-a. Te godine je Rumbaugh počeo surađivati s Boochem u tvrtci Rational Software Corporation, te se u to vrijeme UML sastojao od njihove dvije metode. Tako je došlo do razvijanja prve verzije UML-a koja se zvala 0.8, te je izašla godinu dana nakon što su oni počeli surađivati. Nedugo nakon toga, dolazi do proširenja UML-a (Jacobson im se pridružio), pa je tada UML obuhvaćao i OOSE. Stoga, 1996. godine u lipnju je izašla nova verzija 0.9., pa je većina organizacija smatrala UML korisnim za svoje poslovanje. Nakon toga je osnovan UML konzorcij koji je obuhvaćao nekoliko organizacija. Zatim je došlo do ulaganja vlastitih resursa da bi došlo do kompletnijeg i stabilnijeg UML-a.

Tvrtke koje su pridonijele razvoju UML verzije 1.0 su Oracle, Hewlett- Packard, I- Logix, Intellicorp, Icon Computing, IBM, Texas Instruments, Microsoft, DEC, Rational, MCI i Unisys. UML je tada postao jak, formalan, utjecajan te primjenjiv za rješavanje velikih problema. Tako je on, 1997. godine u siječnju, ponuđen za standardizaciju OMG-a (engl. Object Management Group) koji služi za stvaranje komercijalnog konsenzusa kroz standarde i osiguravanje tržišta u obuhvaćanju standarda.

OMG se smatra i međunarodnom neprofitnom organizacijom koja definira standarde modeliranja uključujući UML. (Stephens, 2015., str.105).

Odgovor na standardizaciju OMG-a je ponuđen, pa je došlo do prijedloga standardnog jezika za modeliranje. Zatim je u rasponu od siječnja do srpnja 1997. godine, došlo do proširenja i obuhvaćanja svih koji su poslali prijedloge OMG-u (primjerice, Ericsson, Softeam, Platinum Techology, itd.). Nakon toga je došlo do osnivanja grupe čiji je vođa bio Cris Kobryn koja je služila za stvaranje UML specifikacije i integracije uključujući druge standarde.

Došlo je do razvijanja nove verzije 1.1 koja je predana OMG-u 1997. godine u srpnju. Tako su OMG ADTS (engl. Object Management Group Analysis Design Task Force) prihvatili prethodno navedenu verziju 1997. godine u rujnu. OMG ADTS služi za razvijanje standarda za integraciju poduzeća i obuhvaćanje širokog raspona tehnologija i industrija. Nakon što je završilo glasovanje, verzija 1.1 je prihvaćena od strane OMG-a 1997. godine u prosincu.

Dolazilo je do razvijanja novih verzija UML-a, pa je aktualna verzija 2.5 koja je nastala 2015. godine. (Manger, 2015a, str. 7).

1.2 Cilj UML-a

Najvažniji cilj UML-a se krije u tome, da se on smatra općim jezikom u svrhu modeliranja kojeg mogu svi koristiti.

Neke karakteristike UML-a (vezane za cilj) su da:

1. On nije ničije vlasništvo.
2. Zasnovan je na principima dogovora računalske zajednice.
3. Uključuje elemente svih važnijih metodologija.
4. Koristi svoj jezik za modeliranje.
5. Njegov dizajn se zasniva na najlogičnijoj i najjasnijom notacijom.
6. Podržava jednostavno izražavanje svih bitnih i dobrih koncepata (primjerice, enkapsulaciju, prikazivanje namjere vezane za neku konstrukciju, razdvajanje odgovornosti, itd.).

UML se smatra i distribuiranim softverom i konkurentnim softverom. Iz tih razloga, on se koristi kao potpora velikom broju postojećih metodologija.

1.3 Koncepti UML-a

Postoje tri koncepta UML modela, a to su:

1. Stvari (engl. things), tj. apstrakcije
2. Relacije ili veze (engl. relationships)
3. Dijagrami (engl. diagrams)

Pod stvarima se podrazumijevaju elementi modela (primjerice, klase, sučelja, komponente, računalni čvorovi, itd.). Relacije služe za povezivanje stvari i određivanje kako da se stvari semantički odnose jedne prema drugima. Dijagrami omogućuju grupiranje skupina povezanih stvari i definiranje pogleda na model. Na njima su nacrtane stvari (kao kućice) i veze (u obliku spojnica). Stoga, oni se koriste za vizualizaciju strukture ili ponašanje sustava. Tako dijagram nije isto što i model, makar se stvara samim crtanjem dijagrama.

Stvari se dijele u 4 skupine, a to su:

1. Stvari strukture
   Stvari strukture služe za prikazivanje statičkih dijelova modela i konceptualnih, odnosno fizičkih elemenata, tj. imenica.

   Ova vrsta stvari obuhvaća klase (primjerice, dijagram klasa) i aktivne klase.
   Klase služe za opisivanje skupa objekata i dijeljenje zajedničkih karakteristika kao što su to atributi, operacije, ograničenja i semantika. Aktivna klasa pritom obuhvaća objekte koji imaju vlastitu kontrolu te tako mogu započeti neku upravljačku aktivnost.

   Svaka metodologija mora definirati ključne elemente vezane za softver koji se izrađuje.

   Podaci se kod stvari strukture modeliraju atributima, dok se ponašanje klasa određuje operacijama. Pod operacijama se podrazumijevaju objekti koji služe za izvršavanje i za implementaciju. Ovdje klase mogu dijeliti unutrašnju strukturu putem generalizacije gdje onda svaka klasa (niže) u hijerarhiji može

dodavati nove informacije te operacije (inkrementalni način dodavanja). One potom nasljeđuju postojeće klase koje su smještene više u hijerarhiji. Štoviše, objekti klasa mogu biti i povezani u toku izvršavanja (primjerice, škola bez učenika, tzv. asocijacija).

2. Stvari ponašanja

   Stvari ponašanja prikazuju dinamičke dijelove modela, odnosno ponašanje kroz prostor i vrijeme (glagoli).

   Postoje dvije dimenzije, a to su povijest objekta i pravilnost u komunikaciji glede drugih objekata. Pod povijesti objekta se podrazumijeva to da ona, bez komunikacije s drugim objektima, predstavlja se stanjima putem kojih objekt prolazi. Ovdje se vrše radnje te se mijenjaju stanja u skladu s radnjama. Kod pravilnosti u komunikaciji glede drugih objekata se prikazuju pojedini objekti, odnosno njihova suradnja u razvijanju većeg posla. Tu se dakle vidi koji objekti su koliko prisutni (primjerice, na početku ili na kraju posla) te kakve poruke se razmjenjuju između njih.

3. Stvari grupiranja

   Stvari grupiranja definiraju organizacijske dijelove modela, tj. kutije ili pakete kamo model može biti ukomponiran. Na taj način, timovi moraju raditi na različitim dijelovima sustava te upravljati međuzavisnostima paketa.

4. Stvari anotacije

   Stvari anotacije objašnjavaju dijelove modela te služe za prikazivanje komentara koji se mogu primjenjivati na bilo koji element.

2. Vrste modela i načini njihova korištenja

Postoje dvije vrste modela koje služe za modeliranje postojećeg sustava ili za razvijanje novog sustava, a to su (Manger, 2013.,str. 42):

1. Modeli postojećeg sustava
2. Modeli za razvijanje novog sustava

Modeli postojećeg sustava se koriste prilikom utvrđivanja zahtjeva i služe za diskusiju o samom postojećem sustavu (primjerice, njegove prednosti i mane). Na taj način se onda dolazi do zahtjeva na novi sustav. Modeli za razvijanje novog sustava se koriste isto za utvrđivanje zahtjeva, ali u svrhu novog sustava, odnosno u razlikama u pogledu postojećeg sustava. Ove modele koriste softverski inženjeri za diskutiranje i dokumentiranje varijanti koje će se implementirati.

Postoji 13, odnosno 14 dijagrama među kojima je jedan pomoćni. Oni služe za grafičko predstavljanje skupa elemenata na način da se crtaju gledano s različitih stajališta te predstavljaju poboljšani prikaz elemenata.

UML 2.4 ima sljedeću podjelu dijagrama:

a) Dijagrami ponašanja
   - ✓ Dijagrami korištenja
   - ✓ Dijagrami aktivnosti
   - ✓ Dijagrami stanja

b) Strukturni dijagrami
   - ✓ Dijagrami klasa ili razreda
   - ✓ Dijagrami objekata
   - ✓ Dijagrami komponenata
   - ✓ Dijagrami složene strukture
   - ✓ Paketni dijagrami
   - ✓ Dijagrami rasporeda
   - ✓ Dijagrami profila, tzv. pomoćni (verzija UML-a 2.4)

c) Dijagrami međudjelovanja/ interakcije
   - ✓ Dijagrami komunikacije
   - ✓ Dijagrami slijeda
   - ✓ Vremenski dijagrami
   - ✓ Dijagrami pregleda međudjelovanja

Strukturni dijagrami opisuju stvari koje će se nalaziti u sustavu koji korisnici planiraju. Primjerice, klasni dijagram prikazuje odnose između klasa koji će predstavljati objekte u sustavu. Za razliku od strukturnog dijagrama, dijagrami interakcije se smatraju podskupom dijagrama ponašanja.(Stephens, 2015., str. 107 i 113).

Dijagrami se još dijele na statičke i dinamičke. U statičke dijagrame spadaju dijagram klasa, dijagram objekata, dijagram komponenti, dijagram složene strukture, dijagram razmještaja i dijagram paketa. Dinamički dijagrami su dijagram aktivnosti, dijagram slučaja korištenja, dijagram stanja, slijedni dijagram, dijagram komunikacija, dijagram pregleda međudjelovanja i vremenski dijagram.

Od prethodno navedenih dijagrama, najčešće se koristi sljedećih pet, a to su dijagram aktivnosti, dijagram slučaja uporabe, dijagram slijeda, klasni dijagram i dijagram stanja.

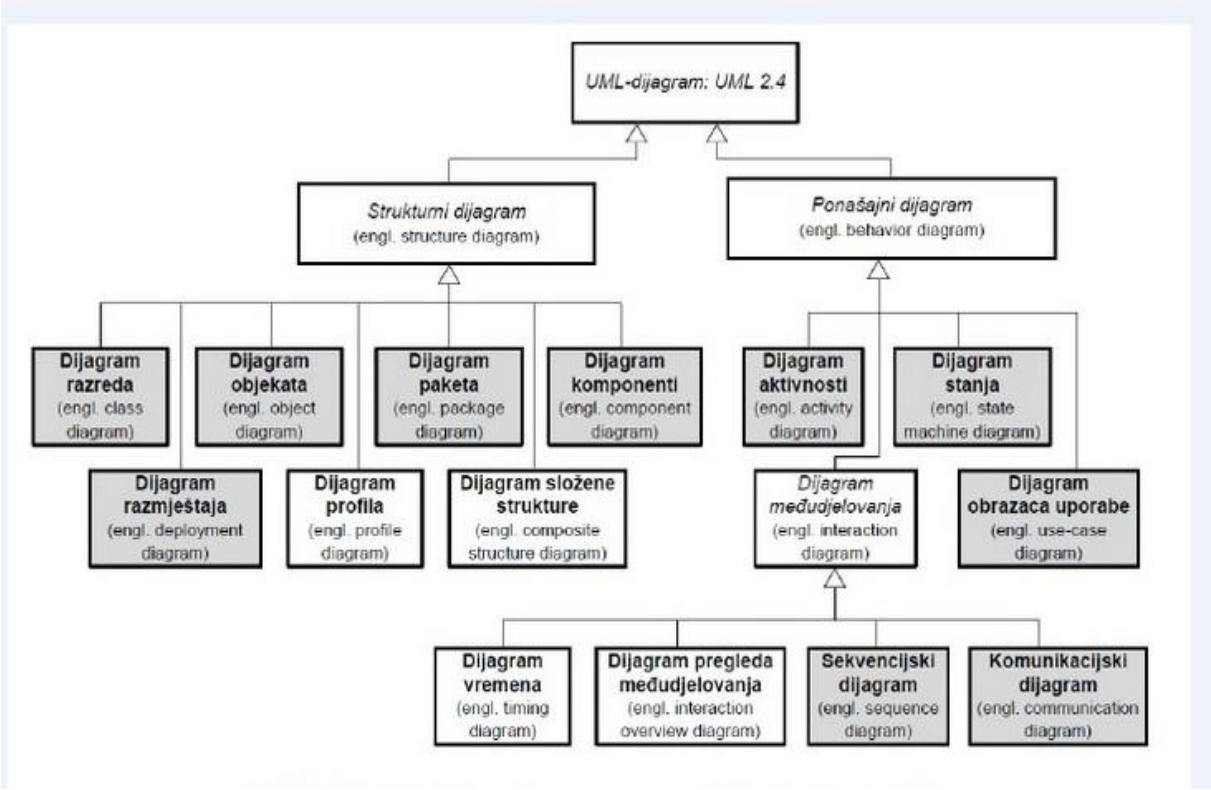
Prilikom razvijanja modela sustava često korisnici mogu biti fleksibilni kod korištenja samog grafičkog zapisa, a to znači da se ne moraju uvijek stručno držati detalja zapisa. Detalji i strogost samog modela ovise o tome kako se namjerava koristiti pojedini model. Stoga, postoje 3 načina kako se dijele modeli po načinu ili svrsi primjene, a to su:

1. Model korisnicima služi kao pomagalo u olakšavanju postojećeg ili novog sustava
2. Model se koristi kao dokumentacija
3. Model se koristi za automatsku implementaciju sustava

Model korisnicima služi kao pomagalo u olakšavanju postojećeg ili novog sustava gdje se omogućuje diskusija među korisnicima. Model se koristi kao dokumentacija koja sadrži modele postojećeg sustava. Model se koristi za automatsku implementaciju sustava koja služi za detaljno opisivanje sustava.

U prvom slučaju, svrha modela je potaknuti raspravu među softverskim inženjerima uključenih u razvoj sustava. Modeli ovdje mogu biti nepotpuni te neformalno mogu koristiti oblikovanje zapisa (tzv. agilni način rada). U drugom slučaju, modeli ne moraju biti potpuni jer se samo žele razviti modeli za neke dijelove sustava. Međutim, ovi modeli moraju biti točni, odnosno trebaju se ispravno upotrebljavati i sadržavati točan opis sustava. U trećem slučaju, modeli sustava moraju biti potpuni i ispravni. Razlog ispravnosti i potpunosti je taj da bi se trebali koristiti kao osnova za generiranje izvornog koda sustava. Stoga, mora se biti vrlo oprezan kod samih elemenata. (Sommerwille, 2011., str. 120-121).

Slika 1.: Pregled UML dijagrama prema verziji 2.4 [OMG 2011.godine]



Izvor: Jović i sur., 2013., str. 5

3. Dijagrami ponašanja

3.1 Dijagrami korištenja

3.1.1 Karakteristike dijagrama

Dijagrami korištenja (engl. use case diagram) spadaju u statičke UML dijagrame i pripadaju grupi ponašajnih dijagrama, što znači da oni služe za modeliranje mogućeg ponašanja korisnika sustava. Stoga, samo statičko ponašanje nije dovoljno za modeliranje sustava, a dinamičko ponašanje se smatra važnijim od statičkog ponašanja.

Ovaj dijagram služi kao „ugovor“ između klijenata, korisnika i programera u provjeravanju sustava kako bi on postao ono što su oni očekivali. (Radovanović, 2011., str. 1).

Dijagram korištenja prikazuje:

1. Ponašanje sustava
2. Dijelove sustava
3. Konkretne klase

Ponašanje sustava se prikazuje pomoću ciljeva i sudionika te se postiže apstrakcija korisnika sustava. Konkretne klase su vidljive samom korisniku sustava.

Ovi dijagrami služe za opisivanje pogleda na ponašanje sustava glede korisnikove strane, tj. grubog prikaza što sustav radi. Isto tako, kod ove vrste dijagrama je bitno naglasiti, da oni ne služe za opisivanje funkcionalnosti koja je izvedena unutar sustava, odnosno funkcioniranje sustava.

Ovaj dijagram se sastoji od aktera i od slučaja korištenja. Svaki slučaj korištenja se detaljno opisuje te se prikazuje korak po korak kakva je interakcija sustava. Tako ovdje ne postoji nikakva korelacija, odnosno veza između slučaja korištenja i klasa unutar sustava. Razlog tomu, sami su slučajevi korištenja koji igraju ulogu vanjskog pogleda s korisnikove strane. (Radovanović, 2011., str. 1).

Štoviše, on omogućava i krajnjim korisnicima razumijevanje sustava.

Razvoj dijagrama korištenja se definira sljedećim nizom aktivnosti (Radovanović, 2011., str. 3):

1. Definiranjem osoba koje sudjeluju u sustavu.
2. Definiranjem slučajeva korištenja, odnosno upotrebe.
3. Definiranjem tipova veza između osoba (sudjelovanje u sustavu) i slučaja korištenja.
4. Izradom dijagrama slučaja korištenja.

Prednosti dijagrama slučaja korištenja su:

1. Komunikacija s krajnjim korisnicima i ekspertima gdje se osigurava obostrano razumijevanje zahtjeva i odobravanje projekta u ranim fazama razvoja sustava (najčešće kao prvi dijagram).
2. Identificiranje korisnika sustava i onoga što sustav treba raditi (identificiranje zahtjeva sučelja sustava).
3. Verifikacija svih prikupljenih zahtjeva i osiguravanje razvojnom timu razumijevanje zahtjeva.

3.1.2 Elementi dijagrama

Elementi dijagrama slučaja korištenja su:

1. Sudionici (engl. actors)
   Sudionici predstavljaju vanjske entitete i izravno su povezani sa sustavom. Isto tako, oni su pokretač svih akcija te pripadaju vanjskom dijelu sustava kao i njihove odgovornosti (dodjeljivanje imena). Također, poželjno je da imena budu različita u svakom sustavu, tj. da ih ne bude istih i da ne budu povezana s organizacijama poduzeća ili sa samim korisnikom.

   Element akter predstavlja živo ili neživo biće.

   Postoje dva tipa aktera, a to su primarni i sekundarni. Primarni akter pokreće sam obrazac uporabe (smjer strelice je prikazan od sudionika do obrasca

uporabe). U tom kontekstu, sekundarni akter predstavlja obrnutu situaciju, tj. definiran je obrazac uporabe nakon što je on pokrenut (smjer strelice je prikazan od obrasca uporabe prema akteru).

Postoje sljedeća pitanja za prepoznavanje aktera, a to su:

a) Tko vodi, odnosno koristi određeni dio sustava?
b) Tko će održavati, servisirati te administrirati sustav?
c) Da li sustav koristi vanjske resurse?
d) Da li sustav koristi druge sustave?

Na dijagramima, akteri su prikazani u obliku čovječuljaka, odnosno jednostavnim prikazom čovjeka. Prikaz aktera je na sljedećoj slici:

2. Obrasci uporabe (ili slučajevi korištenja)

   Obrasci uporabe su skup scenarija koji su povezani putem jednog cilja korisnika. Pod pojmom scenarija se podrazumijeva slijed koraka koji opisuje interakciju između korisnika i sustava. On predstavlja radnju ili slijed operacija koju izvode akteri (njemu je od neke koristi). Stoga, izvršavanje svakog pojedinog obrasca uporabe je neovisno o izvođenju nekog drugog obrasca uporabe.

   Obrasci uporabe služe za prikazivanje funkcija koje sustav nudi i za definiranje komunikacije između aktera i sustava.

   Smatra se dobrom praksom koristiti 10-tak obrazaca uporabe po dijagramu. Zatim, treba paziti da obrazac uporabe definira radnju, a ne stanje ili objekt.

Ispravno je tako pisati unos narudžbe, kontrola zaliha, evidentiranje zaposlenika, dok je neispravno pisati narudžba, zalihe, zaposlenik.

Neka pitanja za određivanje obrasca uporabe su:

a) Koji su zadaci svakog od aktera s aspekta sustava?
b) Da li neki akter upisuje, čita ili mijenja podatke iz sustava?
c) Da li akter treba obavijestiti sustav o nekim promjenama izvana?
d) Mogu li se svi zahtjevi sustava obaviti pomoću definiranih slučajeva?

Na dijagramu se označava u obliku ovala. Prikaz obrasca uporabe je na sljedećoj slici:

Veza je linija koja služi za spajanje aktera i obrasca uporabe.

Ona se između aktera i obrasca uporabe prikazuje ravnom crtom bez strelice, a može biti između:

1. Sudionika i obrasca uporabe
2. Dva ili više sudionika
3. Dva ili više obrasca uporabe

Tipovi veza kod dijagrama obrasca uporabe su:

1. Pridruživanje ili asocijacija

   Neke od karakteristika ove veze su služenje za povezivanje sudionika s obrascima uporabe, mogućnost definiranja njene višestrukosti (određivanje koliko je puta neki sudionik sudjelovao u određenom obrascu uporabe) i definiranje broja sudionika (određivanje koliko se može izvesti određenih obrazaca uporabe).

Ona se na dijagramu označava s crnom crtom bez strelice. Prikaz asocijacije je na sljedećoj slici:

2. Generalizacija ili poopćenje (poznato kao nasljeđivanje)
   Nasljeđivanje je tip veze gdje se povezuju dva sudionika ili dva obrasca uporabe. U slučaju da se povezuju dva sudionika, specifičniji sudionik preuzima sve uloge apstraktnijeg uz dodatak novih uloga ili osobina (ako ih ima). Međutim, ako se povežu dva obrasca uporabe, specifičniji proširuje, tj. mijenja funkcionalnost apstraktnijeg.

   Ovdje se više sličnih obrazaca uporabe poopćuju u zajednički slučaj. Isto tako, služi za prikazivanje veze između roditelja i djeteta (dijete nasljeđuje osobine roditelja).

   Na dijagramu se označava s trokutastom strelicom. Prikaz nasljeđivanja je na sljedećoj slici:

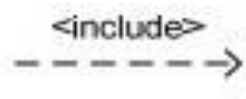
3. Uključivanje (engl. include)
   Uključivanje je tip veze gdje se povezuju dva obrasca uporabe tako da obrazac koji je u tijeku svog izvođenja, u potpunosti izvede uključeni obrazac uporabe. Na prethodno navedeni način, smatra se da prvi use case uključuje (engl. includes) drugi.

   Ova veza se koristi za kompleksne korake zbog preglednosti glavnog scenarija te za korake koji se ponavljaju glede nekoliko obrazaca uporabe.

Tako ovdje osnovni obrazac uporabe ne može postojati bez uključenog.

Ta veza se koristi kada više obrazaca uporabe mogu dijeliti zajedničku funkcionalnost koja se onda smješta u zaseban slučaj.

Na dijagramu se označava s isprekidanom strelicom i s nazivom <<include>> iznad strelice. Prikaz te veze je na sljedećoj slici:



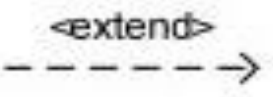
4. Proširenje (engl. extend)
   Proširenje je tip veze gdje se povezuju dva obrasca uporabe, a služi da jedan obrazac uporabe proširuje funkcionalnost drugog (kao dodatna opcija).

   Proširenje će se ostvariti ukoliko je zadovoljen određeni uvjet koji je definiran u točki proširenja. U tom slučaju, obrazac uporabe koji se proširuje nastavlja ponašanje glavnog obrasca uporabe, te je tako jedan obrazac uporabe osnovni, dok ga drugi proširuje.

   Tip takve veze se uvijek definira od proširenog prema osnovnom. Osnovni obrazac uporabe može postojati bez proširenog.

   Koristi se u svrsi, kada se neki obrazac uvjetno proširuje s drugim.

   Proširenje se (kao i uključivanje) označava s isprekidanom strelicom, ali s nazivom iznad strelice <<extend>>. Prikaz te veze je na sljedećoj slici:

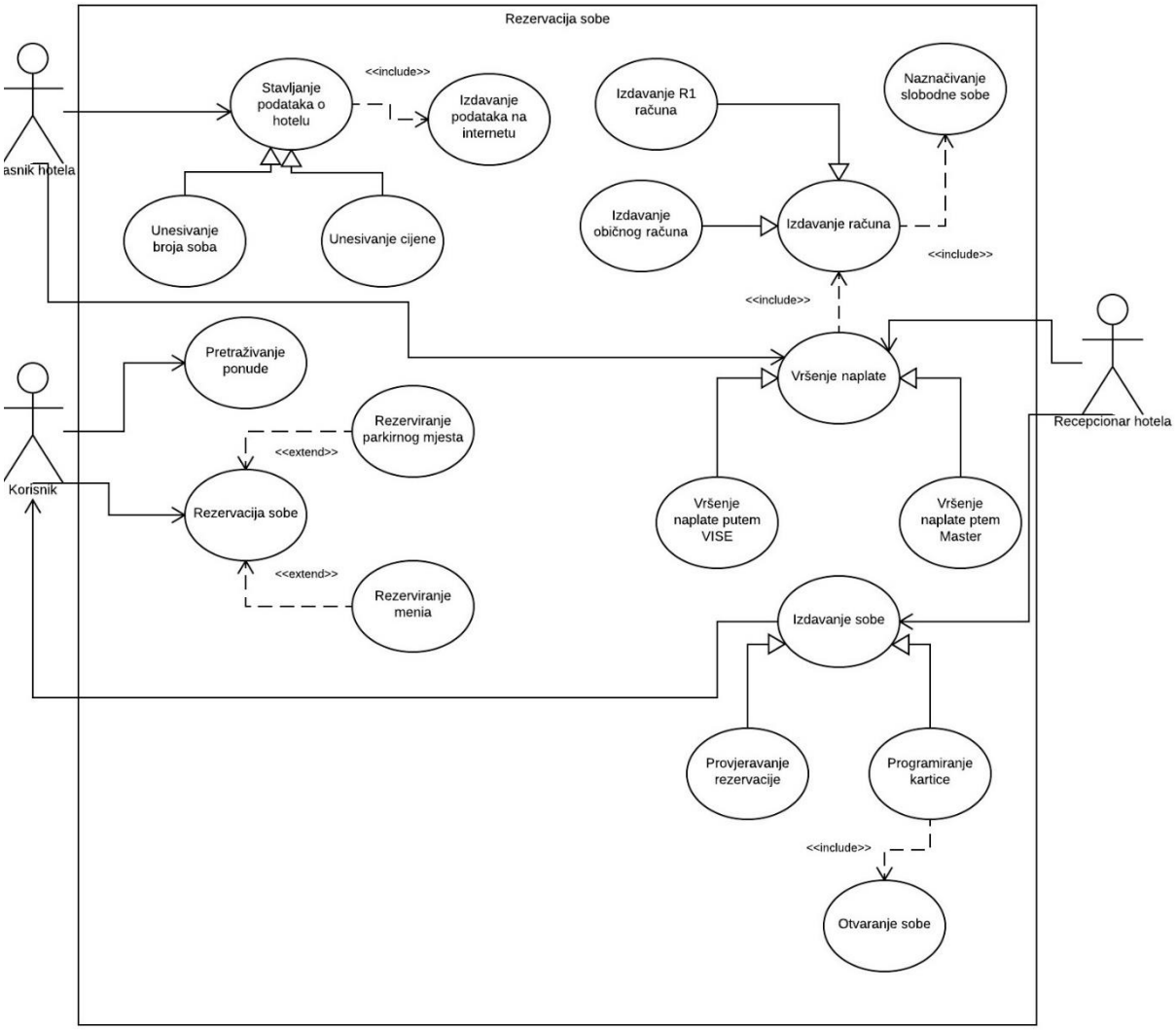


3.1.3 Vlastiti primjer

Tekst zadatka:

Vlasnik hotela stavlja na Internet sljedeće podatke, a to su broj soba hotela, parkirna mjesta i cijene. Tako korisnici putem interneta pretražuju ponudu te prave rezervacije sobe. Također, nudi im se i dodatna mogućnost rezerviranja parkirnog mjesta te menija. Recepcionar hotela potom izdaje sobu korisniku i vrši naplatu. U slučaju izdavanja sobe, recepcionar provjerava rezervaciju i programira karticu za otvaranje sobe. Prilikom naplate, koja se može vršiti putem VISE ili MASTER CARD-a, izdaje se račun R1 ili običan i naznačuje se da soba postaje slobodna. Naravno, vlasnik hotela ima uvid u vršenje naplate.

Slika 2.: Rezervacija sobe iz hotela putem interneta



Izvor: Vlastiti rad

Opis slike:

Na slici je prikazana granica sustava i tri aktera, a to su vlasnik hotela, korisnik, recepcionar hotela (na dijagramu označeni s čovječuljcima koji su povezani s obrascima uporabe). Vlasnik hotela potom vrši radnju stavljanja podataka o hotelu (na dijagramu označeno s ovalom). Potom on uključuje izdavanje tih podataka na internet (prikazano s vezom <<include>> putem iscrtane strelice). Obrasci uporabe (unos broja soba i cijene) nasljeđuju radnju stavljanja podataka o hotelu (na slici prikazano s trokutastim strelicama). Akter korisnik pretražuje ponude te rezervira sobu. Radnja rezervacije sobe ima dodatne opcije, a to su da korisnik može uz rezervaciju sobe još rezervirati meni ili parkirno mjesto (označeno putem veze <<extend>> s iscrtanom

strelicom). Recepcionar hotela izdaje sobu korisniku, a obrasci uporabe (programiranje kartice i provjeravanje rezervacije) nasljeđuju izdavanje sobe. Programiranje kartice podrazumijeva još uključivanje otvaranja sobe gdje recepcionar hotela još vrši radnju naplate, u čiji uvid još ima i vlasnik hotela. Ono se može vršiti putem Master Carda ili putem Visa kartice i to je na dijagramu prikazano kao nasljeđivanje. Vršenje naplate još uključuje izdavanje računa, koji može biti u obliku izdavanja R1 računa ili običnog računa (također, prikazano kao nasljeđivanje). Potom još samo izdavanje računa uključuje naznaku slobodne sobe.

3.2 Dijagrami aktivnosti

3.2.1 Karakteristike dijagrama

Dijagrami aktivnosti (engl. activity diagram) su jednostavni jer se operacije odvijaju slijedno, odnosno jedna za drugom. Štoviše, oni pripadaju dinamičkim dijagramima jer opisuju ponašanje sustava, tj. aktivnosti i prijelaze između tih stanja. (Anonymous_2, n.d., str. 1).

Ovi dijagrami su dosta nalik staroj tehnici dijagrama toka te podržavaju modeliranje paralelnog izvršenja aktivnosti.

U prijašnjim verzijama ovi dijagrami su se zasnivali na dijagramima prijelaza stanja, tzv. teoriji konačnih automata. Danas (od verzije UML 2.0) se zasnivaju na konceptima Petrijevih mreža. Konačni automat predstavlja matematički model koji se sastoji od konačnog broja stanja, prijelaza između tih stanja i akcija koja obavljaju ta stanja. Ovdje token predstavlja koncept putem kojeg se prati samo izvršenje dijagrama aktivnosti. Pod Petrijevim mrežama se podrazumijeva prikazivanje i modeliranje dinamičkih sustava (dobivanje njihove svrhe u analiziranju ponašanja u različitim okolnostima).

Dijagram aktivnosti se koristi za različite nivoe projektiranja softvera. Ovaj dijagram pritom služi za opisivanje poslovnih procesa, radnog toka aktivnosti (koji se obavljaju korak po korak), složenih use- case tokova, proceduralnih logika i algoritama.

Ovi dijagrami se promatraju kao sustavi koji imaju svoja stanja te služe za obavljanje aktivnosti (pod tim se smatra prijelaz iz jednog u drugo stanje i prouzrokovanje nekog

događaja ili tranzicije). Dakle, prikazuju se sekvencijalni tokovi aktivnosti koji se sastoje od stanja, akcija ili radnji i prijelaza. Pod stanjem se podrazumijeva:

1. Akcija
2. Aktivnost
3. Pseudostanje
4. Stanje toka objekata

Akcija traje neko vrijeme te se ne prekida. Aktivnost, za razliku od akcije, ima trajanje i može se prekidati zbog događaja. Pseudostanje obuhvaća stanje prijelaza.

Proces razvoja ovih dijagrama sadrži definiranje plivačkih staza, stanja dijagrama aktivnosti i tranzicije.

3.2.2 Elementi dijagrama

Dijagram aktivnosti ima sljedeće elemente, a to su:

1. Početno i završno stanje

   Početno stanje se na dijagramu označava s punim krugom, dok se završno stanje označava s punim krugom unutar drugog kruga.

   Prikaz simbola početnog i završnog stanja je na sljedećim slikama:

2. Aktivnost

   Postoje sljedeće vrste aktivnosti, a to su paralelne (nije bitan redoslijed odvijanja aktivnosti) i slijedne. Paralelne aktivnosti se trebaju „sinkronizirati“ (primjerice, nije moguće zaključiti narudžbu prije nego što se ne izvrši to da roba ne bude dostavljena ili plaćena).

   Aktivnost se na dijagramu označava sa zaobljenim pravokutnikom. Prikaz simbola aktivnosti je na sljedećoj slici:

3. Prijelaz ili tok

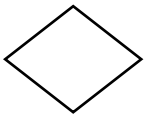
   On se koristi između aktivnosti.

   Na dijagramu se označava s običnom strelicom. Prikaz simbola toka je na sljedećoj slici:

4. Odluka (poznatija kao if)

   Odluku koristimo kada postoji neki uvjet.

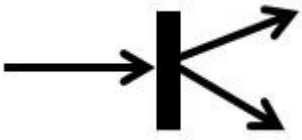
   Na dijagramu se označava s praznim dijamantnim rombom. Prikaz simbola odluke je na sljedećoj slici:

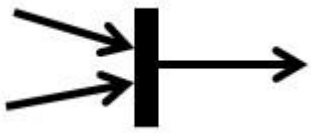


5. Račvanje i skupljanje

   Skupljanje (engl. join) se koristi za sinkronizaciju paralelnih aktivnosti. Ovdje se potom može odlazni tok tek pokrenuti kada su svi dolazni tokovi došli do skupljanja. Tako skupljanje ima više ulaznih tokova i samo jedan izlazni tok (označava kraj uvjetnog ponašanja koje je bilo započelo s odlukom), dok račvanje ima jedan ulazni tok i više izlaznih tokova.

   I račvanje i skupljanje se označava s crnom podebljanom strelicom. Prikaz račvanja i skupljanja je na sljedećim slikama:





6. Signal

   On služi za pokazivanje neke aktivnosti (stalno „sluša“ signale) gdje pritom djeluju neki događaji koji su nastali tijekom vanjskog procesa. Dijagrami u tom pogledu služe za definiranje kako će aktivnost reagirati.

   Stoga, postoje šaljući signali (prikaz na prvoj slici), primajući signali (prikaz na drugoj slici) i vremenski signali (prikaz na trećoj slici):

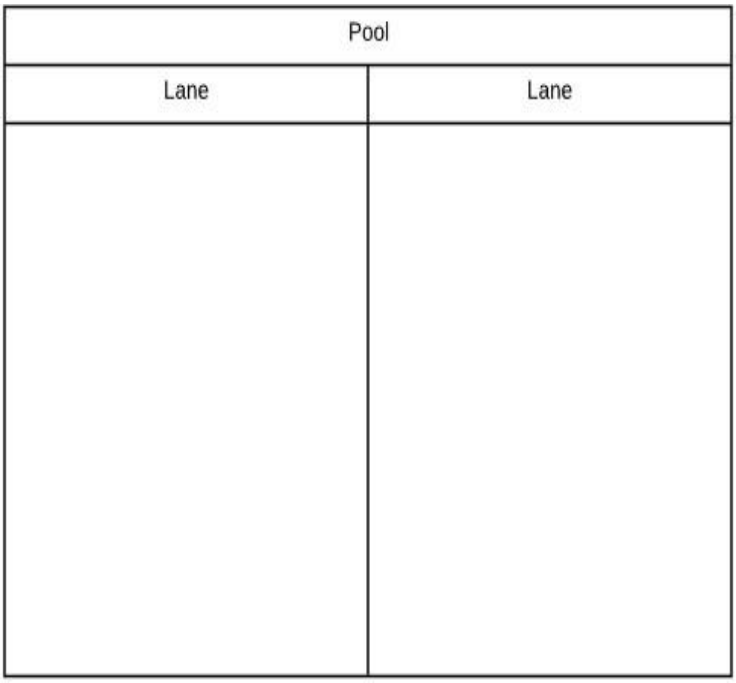
7. Plivaće staze (horizontalne ili vertikalne)

   Plivaće staze su razdvojene linijama u odgovarajuće logičke cjeline gdje se kod svake staze navode sudionici i stanja koja pripadaju stazama.

   Dosta su bitne kada se želi istaknuti koji od dijelova sustava što radi, te na taj način, tokovi podataka mogu prelaziti iz jedne staze u drugu stazu.

   Plivaće staze nisu obavezne, što znači da se može crtati dijagram i bez njih.

   Prikaz plivaće staze je na sljedećoj slici:

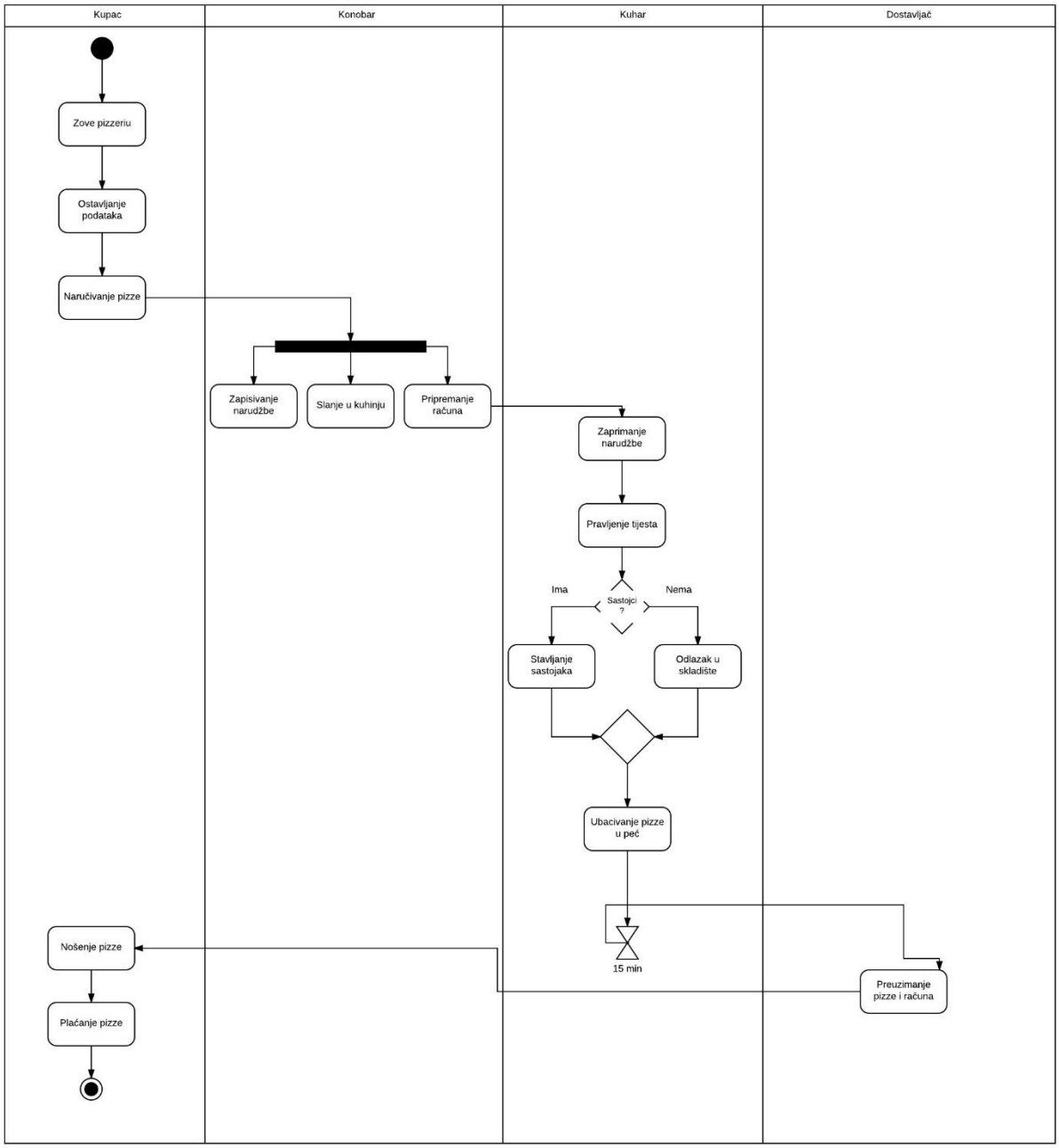


3.2.3 Vlastiti primjer

Tekst zadatka:

Kupac zove pizzeriu i naručuje pizzu te dostavlja svoje podatke. Konobar potom zapisuje narudžbu od kupca. U isto vrijeme, narudžbu šalje u kuhinju i priprema račun. Kuhar zaprima narudžbu i počinje s pravljenjem tijesta te na tijesto stavlja sve potrebne sastojke. Ukoliko nema sve sastojke, odlazi u skladište po ono što mu nedostaje. Na kraju pizzu ubacuje u peć. Nakon 15-tak minuta, pizza je pečena. Dostavljač preuzima pizzu i račun te ju nosi kupcu koji ju potom plaća.

Slika 3.: Narudžba pizze



Izvor: Vlastiti rad

Opis slike:

Na dijagramu su naznačene plivaće staze (koriste se kada postoji više aktera), a to su kupac, konobar, kuhar te dostavljač. Radnje na dijagramu su označene putem zaobljenog pravokutnika. Početak dijagrama je označen s punim krugom te on započinje kod aktera kupac. Tako kupac vrši radnje zvanja pizzerie, ostavljanja svojih podataka i naručivanja pizze. Potom se prelazi na drugu plivaću stazu kod konobara koji zapisuje narudžbu od kupca. On u isto vrijeme narudžbu šalje u kuhinju i priprema račun (prikaz s vodoravnom crtom). Tako kuhar zaprima narudžbu i pravi tijesto te provjerava je li ima sve sastojke koji su mu potrebni (odluka se označava s praznim rombom). Ako ima sve sastojke, onda ih stavlja na pizzu, a ako ih nema, onda ide u skladište. Potom je zatvorena odluka sa spajanjem (ista oznaka kao i grananje). Kada su stavljeni svi sastojci na pizzu, ona se ubacuje u peć. Nakon 15 min (označeno s pješčanim satom), dostavljač preuzima pizzu i račun, te ju nosi kupcu koji ju potom plaća. Dijagram završava s punim krugom unutar drugog kruga te na plivaćoj stazi od kupca kamo je i počeo.

3.3 Dijagrami stanja

3.3.1 Karakteristike dijagrama

Dijagrami stanja (engl. state machine diagram) predstavljaju nadkategoriju dijagrama aktivnosti gdje služe za modeliranje od stanja do stanja. (Manger, 2015f, str.4).

Postoje sličnosti između dijagrama stanja i aktivnosti, a to su (Manger, 2015f, str. 4):

1. Oba dijagrama služe za opisivanje dinamike sustava.
2. Oba dijagrama su zasnovana na “grafovskom” matematičkom modelu (sastojanje od čvorova i od lukova).

Postoje i razlike između prethodno navedenih dijagrama, a to su (Manger, 2015f, str. 4):

1. Dijagrami aktivnosti služe za modeliranje jednog poslovnog procesa gdje sudjeluje više objekata (oslanjanje na Petrijeve mreže).

2. Dijagrami stanja služe za modeliranje ponašanja, također, jednog poslovnog procesa u raznim procesima. Oni se oslanjaju na matematiku gdje se nazivaju konačnim automatom.

Dijagrami stanja pokazuju kako objekt prelazi iz jednog stanja u drugo stanje, odnosno kako se prikazuju različita stanja.

Ovaj dijagram pokazuje pravila ili događaje gdje se upravlja tom promjenom. Isto tako, on služi za to kako jedan objekt prolazi tijekom svog životnog ciklusa.

Dijagram stanja se ne koristi za sve objekte, već se koristi za dodatno definiranje složenih objekata. Tako se onda na taj način dobiva pojednostavljenje dizajna algoritma.

Dijagram stanja predstavlja:

1. Stanja u kojima se objekt može naći.

2. Događaje (engl. events) koji utječu na ponašanje objekata.

Događaj se događa u određenom trenutku i mijenja vrijednosti, a služi za opisivanje objekata. Zauzvrat mijenja stanje ili ponašanje samog objekta.

Događaj na UML dijagramu može biti prikazan na dva načina, a to su izvana na strelici (koja predstavlja prijelaz) i iznutra unutar stanja (unutrašnji događaji).

Postoje četiri vrste događaja, od kojih svaki ima otprilike drukčiju semantiku:

1. Događaj- poziv (engl. call event)
2. Događaj- signal (engl. signal event)
3. Događaj- promjena (engl. change event)
4. Vremenski događaj (engl. time event)

Događaj- poziv predstavlja poziv u izvršavanju nekog skupa akcija. Ovdje se većinom radi o pozivu operacije iz pripadne klase i on se smatra najčešćom vrstom događaja. Događaj- signal predstavlja prijem signala, odnosno predstavlja asinkrone poruke od nekog drugog objekta (signal predstavlja informaciju). Modelira se u obliku klase bez operacija sa stereotipom <<signal>> te se informacija koja se prenosi nalazi u atributima te klase. Događaj- promjena se događa kad se neki booleovski izraz promjeni iz istine u laž. U tom se booleovom izrazu mogu pojavljivati konstante,

globalne varijable, atributi ili operacije neke klase. Ovaj događaj se prikazuje na dijagramu tako da na mjesto gdje se inače upisuje događaj (iznad strelice ili unutar stanja) upiše odgovarajući booleov izraz. Da bi se događaj- promjena ponovno dogodio, vrijednost booleovog izraza se mora iz istine vratiti u laž, i potom opet u istinu. Vremenski događaj se događa nakon što protekne zadani vremenski interval ili u slučaju kada dođe zadani trenutak.

3. Prijelaze između tih stanja (predstavljanje reakcije na događaje).

4. Pokazivanje krajnje te početne točke kada se vrši promjena stanja ili kada se događaju akcije ili aktivnosti.

3.3.2 Elementi dijagrama

Osnovni elementi dijagrama stanja su:

1. Početno stanje (engl. initial state)

Predstavlja trenutak u kojem objekt započinje svoje postojanje. Na dijagramu mora postojati jedno početno stanje.

Početno stanje se prikazuje isto kao i kod dijagrama aktivnosti, odnosno punim crnim krugom. Prikaz početnog stanja je na sljedećoj slici:

2. Završno (krajnje) stanje (engl. final state)

Predstavlja završetak stanja, te na dijagramu mora postojati bar jedno završno stanje. On se prikazuje isto kao i kod dijagrama aktivnosti, odnosno punim krugom unutar drugog kruga. Prikaz tog elementa je na sljedećoj slici:

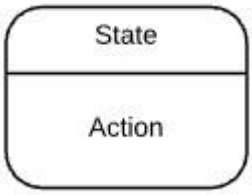
3. Stanje (engl. state)

Predstavlja situacije tokom trajanja života objekta. Može se reći da je on i skup vrijednosti gdje se omogućuje opis objekata u nekom određenom trenutku i sam trenutak u životu objekta kamo se zadovoljavaju neki određeni uvjeti.

Služi za obavljanje određenih radnji i čekanje da se nešto dogodi (primjerice, neki događaj). Mijenja se tokom vremena, no u jednom trenutku ono je određeno s vrijednostima atributa tog objekta, poveznicama (vezama) prema drugim objektima i aktivnostima (radnjama) koje objekt obavlja.

Kod modeliranja ponašanja objekata, savjetuje se da se ne pretjera s brojem stanja. Isto tako, treba uočiti samo stanja koja zaista ima smisla razlikovati glede aplikacija. U tom slučaju, takva stanja se onda nazivaju “semantički značajna” za ponašanje objekata.

Na slici je prikazan u obliku zaobljenog pravokutnika koji je rastavljen na dva dijela po polovici. Na gornjoj polovici piše ime stanje (primjerice, “potpisan”, “ovjeren”, “naplaćen”, itd.), dok donji dio ostaje prazan. Prikaz tog elementa je na sljedećoj slici:



4. Tranzicija (engl. transition)

Tranzicija predstavlja reakciju na događaj (oznaka na strelici) i akciju (radnju) koja mijenja stanje. Služi i za prikazivanje kretnje objekta gdje se prelazi iz jednog u drugo stanje. Objekt se često kreće iz jednog stanja u drugo na temelju rezultata radnje na koje je utjecao, odnosno koje je potaknuo neki događaj. U slučaju da se nalazi u određenom stanju, objekt može izvršavati određene akcije, odnosno aktivnosti (unutarnji prijelaz). Stoga, predstavlja se reakcija objekta na neki događaj (nije važan kod prijelaza u novo stanje).

Između akcija i aktivnosti postoje sljedeće navedene razlike, a to su:

1. Akcija je dio posla
2. Aktivnost je dio posla

Akcija se trenutno izvršava i ne može se prekinuti. Aktivnost troši neko određeno vrijeme i može se prekinuti.

Postoje dvije posebne akcije, a to su ulazna akcija i izlazna akcija. Ulazna akcija se izvršava u trenutku ulaska u stanje (događaj entry), dok se izlazna izvršava u trenutku izlaska iz stanja (događaj exit).

Tranzicija se na dijagramu prikazuje u obliku strelice gdje su zapisani sljedeći elementi:

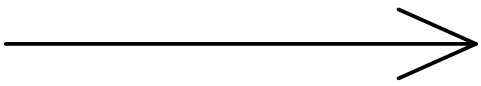
1. Nula
2. Jedan ili više događaja
3. Nula i jedan ili više uvjeta- stražara
4. Nula i jedna ili više akcija

Jedan ili više događaj/a predstavlja/predstavljaju događaj/e od kojih svaki pobuđuje prijelaz. Nula i jedan ili više uvjeta- stražara podrazumijeva Booleove izraze gdje će svi na kraju biti istiniti kako bi prijelaz bio postignut te se onda zapisuju iza događaja. Nula i jedna ili više akcija obuhvaća dijelove posla kamo se svi izvršavaju u trenutku prijelaza. Isti ti elementi se mogu zadati i kod unutarnjih prijelaza, ali tada su oni upisani unutar ikone stanja i nema strelice.

U slučaju da kod prijelaza nije zadan nikakav događaj, tada se radi o automatskom prijelazu. On se aktivira sam od sebe čim njegov stražar- uvjet postane istinit.

Prijelazi na dijagramu mogu biti povezani preko spojnog pseudo- stanja (engl. junction pseudo- state) koje predstavlja mjesto gdje se prijelazi sažimlju ili granaju. Crta se isto kao i početno stanje, odnosno kao puni krug. Spojno stanje se može sastojati od više izlaznih prijelaza uz uvjet da ti prijelazi moraju imati međusobno isključujuće stražare- uvijete.

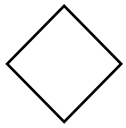
Prikazi tranzicije i spojnog pseudo- stanja su na sljedećim slikama:



5. Grananje prijelaza bez spajanja

Ovdje se koristi pseudo- stanje za grananje (engl. choice pseudo- state).

On se na dijagramu prikazuje kao prazni romb, a prikaz njega je na sljedećoj slici:

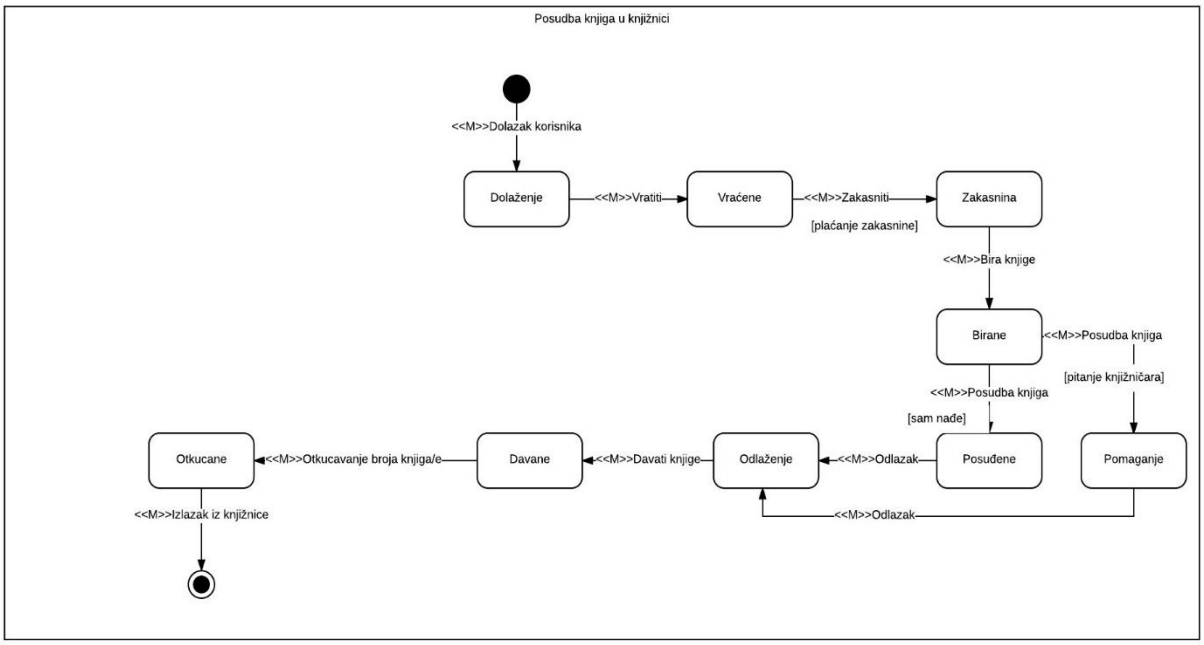


3.3.3 Vlastiti primjer

Tekst zadatka:

Korisnik dolazi u knjižnicu vratiti knjige. U slučaju da ima zakasninu plaća ju. Potom bira knjige koje želi posuditi te u slučaju da ih nađe sam, odlazi do knjižničara te mu ih daje. Međutim, ako ne nađe željenu knjigu, pita knjižničara da mu nađe. Ako su sve knjige odabrane, knjižničar ih otkucava (otkucavanje njihovog broja) te korisnik odlazi iz knjižnice.

Slika 4.: Posudba knjiga u knjižnici



Izvor: Vlastiti rad

Opis slike:

Postupak posudbe knjiga u knjižnici započinje s punim krugom (dolazak korisnika, označeno s <<M>> koje označava radnju) i stanjem dolaska (označeno sa zaobljenim pravokutnikom). Potom postoji radnja vratiti (pretvaranje u stanje vraćene) te zakasniti (pretvaranje u zakasninu). Radnja zakasniti se odvija uz uvjet plaćanja (što je označeno u uglatoj zagradi). U slučaju da korisnik plati zakasninu, bira knjige koje potom prelaze u stanje birane. Tako ako želi posuditi knjige, a ne može ih naći, onda pita knjižničara da mu pomogne naći. Potom, kada dobije željene knjige od knjižničara ili ako ih sam nađe, odlazi do njega te mu daje knjige. Zatim knjižničar otkucava brojeve s knjiga te one postaju otkucane. Sam postupak završava s odlaskom korisnika, odnosno sa simbolom punog kruga unutar drugog kruga.

4. Strukturni dijagrami

4.1 Dijagrami klasa/ razreda

4.1.1 Karakteristike dijagrama

Ovaj dijagram pripada statičkim dijagramima i strukturnoj skupini UML dijagrama. Međutim, on ne uzima u obzir vremensku komponentu sustava, nego uzima dio, odnosno cijeli sustav kakav postoji u nekom trenutku. (Jović i sur., 2013., str. 35).

Dijagrami klasa (engl. class diagrams) prikazuju sustav pomoću klasa ili razreda te pomoću relacija između klasa. (Jović i sur., 2013., str. 35).

Razred ili klasa je osnovni element ovog dijagrama. Prije pojašnjenja značenja riječi razred, mora se odrediti značenje objekta. Tako je objekt pojam koji predstavlja entitet, koncept ili apstrakciju stvarnog svijeta. On mora imati jasno definirane granice te isto tako, mora predstavljati određeni smisao u sustavu. Razred, odnosno klasa s obzirom na objekte, služi za opisivanje njih samih gdje se sadrže slična svojstva. Svaki pojedinačni objekt pritom predstavlja instancu jedne klase.

Svaka klasa mora sadržavati naziv, ali ne mora nužno imati popis atributa i operacije. Iako ih nije potrebno definirati, bez njih sama klasa nema smisla, odnosno nema implementaciju. Kod samih atributa je stoga potrebno navesti naziv i tip podatka. Za

operacije je potrebno navesti definiciju koja će sadržavati naziv operacije te ulazne i izlazne parametre.

4.1.1. Odnosi između razreda

Dva osnovna tipa odnosa između razreda se nazivaju pridruživanje, veza ili asocijacija te podtip (primjerice, klasa koja predstavlja Studenta koji pritom pristupa klasi Ispit, klasa Zaposlenik koja je pridružena klasi Asistent).

4.1.2. Pridruživanje

Pridruživanje, odnosno veza (engl. association) služi za opisivanje statičnih odnosa između dva pojedinca, tj. instance ili razreda.

Veze mogu biti:

1. Jednosmjerne
2. Dvosmjerne
3. Agregacijske
4. Refleksivne

Refleksivne veze predstavljaju objekte iz dotične klase koje mogu imati poveznice prema drugim objektima iz iste klase. Agregacijske veze uključuju kompoziciju.

U slučaju da je vrh neke asocijacije označen sa strelicom, tada je definiran njezin smjer (engl. navigability). Ovisno o smjeru pridruživanja, veze mogu biti:

1. Undirekcionalne
2. Bidirekcionalne

Undirekcionalne (jednosmjerne) veze sadrže smjer koji se nalazi u samo jednom vrhu. Za razliku od undirekcionalnih veza, bidirekcionalne (dvosmjerne) veze, sadrže smjer koji se nalazi na oba vrha.

U slučaju da smjer strelice nije jasno definiran, smatra se da je tada veza nepoznata, odnosno nedefinirana (bidirekcionalna). Međutim, ako se naiđe na takvu vezu, onda

će njezini smjerovi biti invertni. Tako je onda definiran računalni kod koji će služiti za implementiranje obje klase (imati će suprotnu funkciju).

Veza će uvijek imati dva vrha u slučaju pridruživanja razredu te će se definirati klase koje su u vezi. Nedozvoljenom vezom se smatra veza koja spaja više razreda. Vrhovi još imaju imena poput uloge (imenice, glagoli) ili role (engl. association role). Pritom se podrazumijeva da svaki vrh može imati naziv, tj. ime (engl. role name), višestrukost, vidljivost i neka druga svojstva. Obično je u programskom kodu naziv uloge identičan nazivu varijable koja predstavlja vezu između razreda. Osim naziva uloge i veza može imati vlastiti naziv.(Jović i sur., 2013., str. 37-38).

U praksi se vrlo često koriste imenovane veze od imenovanih vrhova. Samo imenovanje veza nije uvijek potrebno, već samo onda kada će to doprinijeti razumijevanju pojedinog dijagrama. Pojam veze se označava još s nazivima glagoli ili (ne baš često) imenicama. Pritom one mogu opisivati radnju koju će jedan razred ili klasa vršiti nad drugim razredom ili klasom (primjerice, veza između klasa Profesor i Student definirana kao “sluša_predmet”) te odnosom između dvaju razreda ili klase (primjerice, odnos između klasa Profesor i Student putem veze “mentor”). Veza “mentor” se tako može imenovati i u glagolskom obliku, a to je “je_mentor_od”. Većinom se za takvo imenovanje koristi 3. lice jednine prezenta, a kod imenica se koristi nominativ u jednini. Ime pojedine veze se izabire sukladnosti smjeru pridruživanja.

Bitno je napomenuti da je na dijagramu nekada potrebno označiti samo jedno ime vrha jer pri tome dobivamo vidljivost (označena svojstva imena veze). U tom kontekstu, sva svojstva imena vrha se prenose na ime veze. Višestrukost pridruživanja, odnosno veze (engl. multiplicity) se odnose na svaki vrh veze. Ona pri tome služi za određivanje broja pojedinaca ili samih objekata koji mogu sudjelovati u odnosu između dvije klase.

Dozvoljene vrijednosti na bilo kojoj strani pridruživanja (Jović i sur., 2013., str. 39):

1 = točno jedan pojedinac

n1 = bilo koji točno određen broj, npr. 0,1,3,5,15

n1..n2 = između n1 i n2, npr. 5...8-> 5,6,7,8

n1..n = između n1 i više pojedinaca

n1...n2, n3 = kombinacija, npr. 4...7,9-> 4,5,6,7 ili 9

n..* = n ili više pojedinaca, neograničeno

0..* ili * ili n = više pojedinaca, neograničeno

Veza jedan na više kaže da primjerice, ako neka tvrtka ima više zaposlenika, taj će onda zaposlenik moći raditi samo za tu tvrtku. U slučaju da je naznačena veza više na više, onda primjerice, administrativni asistent može funkcionirati za više upravitelja, a isto tako, upravitelj može imati puno administrativnih asistenata. Veza jedan na jedan, kaže da za svaku tvrtku postoji upravo jedan upravni odbor. (Lethbridge i Langaniere, 2005., str. 176-177).

U slučaju da višestrukost pridruživanja nije označena (primjerice, “točno 1 pojedinac”), tada se smatra da je vrijednost jedan. Kod implementacije, odnosno u programskom kodu višestruko pridruživanje se vrši putem dinamičke strukture podataka (primjerice, std::vector u C++- u koji sadrži pokazivače na drugu klasu).

Refleksivno pridruživanje (agregacija) je tip veze gdje više pojedinaca iste klase mogu međusobno komunicirati. U slučaju kada je potrebno definirati vezu, koriste se nazivi uloga (vrhova), a ne nazivi veza, te je potrebno odrediti njegovu višestrukost. Ovo pridruživanje se može primijeniti na jednosmjerne i dvosmjerne veze, ovisnost ili na bilo koju vrstu pridruživanja dijagrama klase.

4.1.3 Agregacija i kompozicija

Veza agregacije (engl. aggregation relationship) predstavlja vrstu asocijacije kod koje jedna strana igra ulogu cjeline, dok druga strana igra ulogu dijela (engl. whole/ part relationship). Cjelina sadrži dijelove (engl. “has-a” relacija), a najčešće označava grupiranje dijelova te ne govori ništa o međusobnom odnosu životnih vjekova cjeline i njihovih dijelova. Stoga, označava se zajednički dio više cjelina.

Štoviše, agregacija pokazuje i da jedna klasa sadrži druge klase. U tom slučaju se smatra da je klasa agregirana, odnosno sadržana unutar druge klase. To je zapravo oblik odnosa nadskup- podskup (veći skup- manji skup). Agregacija se koristi i u slučajevima kada se modeliraju složene, odnosno miješane relacije između samih klasa.

Prikaz tog elementa je na sljedećoj slici:

Kompozicija (engl. composition) je isto (kao i agregacija) asocijacija kod koje postoji odnos između dijela i cjeline. Međutim, ovdje je cjelina odgovorna za životni vijek dijela. Dio može nastati u toku života cjeline te može biti uništen (engl. termination) prije samog uništenja cjeline. Ovdje se objekti razreda ili pojedinaca uvijek uništavaju kao i pojedinci razreda koji su dio početnog objekta, te on može biti dio samo jedne cjeline. Stoga, sustav kada oslobađa zauzetu radnu memoriju i briše pojedince, osim njih će obrisati i sve pojedince koji su s njime povezani putem kompozicije. Slična je agregaciji, ali se prethodno opisan slučaj neće dogoditi. To znači da će i drugi pojedinci, koji su agregirani s početnim objektom, ostati sačuvani u radnoj memoriji sustava.

Simboli agregacije te kompozicije uvijek će dodirivati razred nadskup, a prazna linija razreda će dodirivati razred podskup. Agregacija i kompozicija su uvijek usmjerene od nadskupa prema podskupu.

Prikaz tog elementa je na sljedećoj slici:

4.1.4 Atributi

Atributi (engl. attributes) se definiraju kao imenovana svojstva klase i kao članske varijable razreda. Može se reći da su oni strukturne, odnosno opisne karakteristike (engl. features) klasa. Njihova svrha svodi se na opisivanje opsega vrijednosti i pojava tog svojstva koje mogu oni sadržavati.

Postoje i druga imena za atribute, a to su da su oni članovi podataka (kod jezika C++) ili da su oni polja (kod Jave).

Notacija predstavlja ime (engl. name) koje je u sklopu tipa ili vrste (engl. type) i podrazumijevanu ili početnu vrijednost (engl. initial value) te stupanj vidljivosti (engl. visibility). Za njih je dozvoljeno definirati promjenjivost (engl. changebility) te modifikator (engl. modifier).

Glede svih atributa potrebno je definirati svojstva vidljivosti, vrstu ili tip te različita svojstva promjenjivosti atributa. Stupanj vidljivosti atributa služi za određivanje načina kod pojedinaca (mogućnost pristupanja atributima razreda). Tako postoje četiri vrste stupnja vidljivosti atributa, a to su:

1.Javni (engl. public)
2.Privatni (engl. private)
3.Zaštićeni (engl. protected)
4.Paketni (engl. package)

Javni pristup kaže da je atribut dostupan svim klasama te paketima. Privatni opisuje da je atribut dostupan samo unutar iste klase. Zaštićeni se upotrebljava kada je atribut dostupan unutar istog razreda ili izvedenih klasa. Paketni podrazumijeva to da je atribut dostupan svim klasama istog paketa.

Za javni, privatni i zaštićeni pristup postoje znakovi koji se upotrebljavaju, a to su:

1. Znak “+” (za javni pristup)
2. Znak “-” (za privatni pristup)
3. Znak “#” (za zaštićeni pristup)

Kod svojstava vrste atributa postoje razne oznake tipova, a to su:

1. UML- ovi tipovi (primjerice, boolean, integer, string, itd.)
2. Java tipovi (primjerice, byte, char, double, float, int, itd.)
3. Java razredi iz paketa java.until (primjerice, collection, date, list, set, time, vector, itd.)
4. Java.lang, java.lang.math i java.net (URL) paketi

Dozvoljeno je korištenje i vlastitih tipova klasa te podataka koji su onda definirani na istom dijagramu. Stoga, kod nekih uređivača UML dijagrama je moguće definirati i dodatna svojstva promjenjivosti atributa, a to su primjerice:

1. AddOnly
2. Changeable
3. Frozen
4. Static
5. Read- only

Kod addOnly vrijednost atributa se može samo povećavati. Changeable podrazumijeva vrijednost atributa koja se može nesmetano mijenjati (podrazumijevano svojstvo). Frozen obuhvaća vrijednost atributa ili asocijacije koja se ne smije promijeniti tokom života objekta. Static označava modifikator, a vrijednost atributa je konstanta ili se ne mijenja tokom života objekta i ne ovisi o njemu. Read- only definira vrijednost atributa koja se ne može mijenjati izvan objekta kojem pripada, ali ga može mijenjati unutar objekta kojem pripada (za razliku od frozena, jedino ga alat ArgoUML ne podržava).

4.1.5 Operacije

Operacije (engl. operations) predstavljaju servise (procese) gdje se takve operacije pritom mogu zahtijevati od bilo kojeg objekta klase. Njih izvršava razred, odnosno vlastite metode i funkcije razreda.

Notacija služi za potpis koji sadrži listu argumenata s tipovima i podrazumijevanim vrijednostima (ona ih može sadržavati, primjerice, tip rezultata).

Najvažnijim svojstvima operacije se smatraju vidljivost (javna, privatna, zaštićena i paketna) te ulazni i izlazni parametri ili argumenti (svojstveni za operacije). Pritom parametri služe za određivanje ulaznih vrijednosti (koje operacija dobiva) te izlaznih vrijednosti gdje se one (izlazne vrijednosti) vraćaju pozivajućim funkcijama.

Na dijagramu se operacije označavaju unutar klase, odnosno kako je klasa podijeljena na tri dijela, one se pišu na najdonjem dijelu.

4.1.6 Nasljeđivanje

Nasljeđivanje (engl. inheritance) spada u temeljne koncepte objektno- orijentiranog programiranja.

Ova vrsta operacije se smatra oblikom odnosa između klasa (nasljedna veza) gdje je objekt koji se nasljeđuje proširen objektom koji ga nasljeđuje.

Nasljeđivanje služi za povezivanje općih stvari (superklasa, nadklasa, osnovna klasa ili roditelj) sa specijaliziranom stvari (subklasa, podklasa, izvedena klasa ili dijete).

Postoje dvije vrste odnosa između razreda, a to su:

1. Generalizacija
2. Specijalizacija

U ovom kontekstu, generalizacija označava pojavljivanje objekta djece gdje god se očekuje pojavljivanje objekta roditelj gdje djeca nasljeđuju osobine svojih roditelja, atribute, strukturu i operacije. Tako onda operacija djeteta redefinira operaciju roditelja na način da omogućava njeno polimorfno ponašanje.

Klasa koja ima samo jednog roditelja koristi jednostruko nasljeđivanje, dok klasa koja ima više roditelja koristi višestruko nasljeđivanje.

Generalizacija omogućuje i stvaranje nadrazreda (uključuje strukturu) te ponašanja (zajedničko za sve klase). Specijalizacija služi za stvaranje podrazreda koji dodaje nove elemente. Generalizacija i specijalizacija su obrnute radnje. Konkretno, generalizacija djeluje od podrazreda prema nadrazredu, a specijalizacija od nadrazreda prema podrazredu.

Na dijagramu, nasljeđivanje se uvijek prikazuje od podrazreda prema nadrazredu, odnosno u smjeru generalizacije. Nekad je korisnije crtati obrnuto jer to pridonosi jednostavnijem i bržem razumijevanju dijagrama klase.

Kod nasljeđivanja uvijek vrijede sljedeća pravila (Jović i sur., 2013., str. 45):

1. Podrazred uvijek ima više ili jednak broj svojstava u odnosu na nadrazred.

2. Podrazred nasljeđuje od nadrazreda atribute, relacije i operacije.

3. Podrazred može biti proširen atributima, operacijama i relacijama.

4. Podrazred može imati svoju implementaciju operacija koje je naslijedila klasa

Vezu nasljeđivanja je potrebno koristiti kada se neka zajednička svojstva dijele, odnosno rastavljaju od specifičnih svojstava.

Važno je još napomenuti da nasljeđivanje nema višestrukost.

4.1.7 Ovisnost

Ovisnost (engl. dependency) povezuje stvari (predstavlja relaciju) kod kojih izmjena nezavisne stvari, odnosno entiteta utječe na ponašanje neke zavisne stvari (korištenje nezavisne stvari). Nezavisni entitet predstavlja isporučitelja, a ovisni klijenta. Ovisnost se dosta često koristi kada je jedna klasa (B) tip parametara operacije neke druge klase (A).

Ova veza predstavlja i odnos gdje jedna klasa ovisi o drugoj klasi. Međutim, ona može predstavljati i jedan paket koji pritom ovisi o drugom paketu (ili komponentu koja ovisi o drugoj komponenti). Značenje ovisnosti se krije u jednosmjernoj vezi (ili undirekcionalnoj) gdje se ta veza čita kao da “B ovisi o A”.

U programiranju, ovisnost se može prikazati na više načina. Primjerice, s funkcijama koje pritom osluškuju događaje, što znači da klasa B ima navedenu funkciju (ovisna o klasi A koja generira događaj).

4.1.8 Sučelje i realizacija

Realizacija je semantička relacija između klasifikatora gdje jedan dio služi za specificiranje ugovora, dok drugi garantira njegovu implementaciju. Može se reći da je ona i usko povezana sa sučeljem, tj. označava ostvarenje sučelja.

Jedan ili više razreda realiziraju ili ostvaruju sučelje, odnosno koriste sučelje kako bi implementirali operacije definirane u sučelju. Odnos realizacije je sličan nasljeđivanju, ali se razlika krije u tome da se kod realizacije “nasljeđuju” samo operacije s parametrima bez implementacije. (Jović i sur., 2013., str. 48).

Veza realizacije se označava pomoću strelice i uvijek je usmjerena od razreda prema sučelju.

Višestruko nasljeđivanje je zabranjeno u nekim objektno- orijentiranim programskim jezicima (primjerice, kod Jave), dok je višestruka realizacija uvijek dozvoljena (omogućuje višestruko nasljeđivanje).

Grafička notacija se dijeli u dvije forme:

1. Kanonička forma

2. Skraćena forma

Kanonička forma predstavlja vezu između klase i sučelja. Ostvaruje se putem veze ovisnosti te se na dijagramu onda označava (kao i ovisnost) s isprekidanom strelicom. Skraćena forma služi za klasu koja je povezana putem asocijacije sa sučeljem (na dijagramu se označava s običnom linijom).

Korištenje sučelja (engl. interface) predstavlja relaciju zavisnosti gdje isto postoje dvije forme, a to su kanonička i skraćena. Sučelje je skup operacija koje sačinjavaju usluge neke klase. Ono definira skup potpisa od operacijskih specifikacija, ali nikada ne definira implementaciju. Može se još reći da je sučelje vrsta klase bez atributa gdje operacije imaju definiciju bez tijela funkcije i implementacije.

Izgled UML simbola sučelja se prikazuje kao klasa s jedinom razlikom, a to je da nema prostora za atribute.

4.1.9 Tipovi podataka i enumeracija

Tipovi podataka (engl. data type) predstavljaju tipove čije vrijednosti, odnosno podaci nemaju identitet. Oni su slični klasama, a razlikuju se od njih po tome što se pojedinci identificiraju samo po vrijednosti.

Na dijagramu se ne povezuju s klasama, ali se crtaju tako što se iznad naziva enumeracije nalazi oznaka <<enumeration>>.

Postoje dvije vrste tipova podataka, a to su:

1. Primitivni tipovi (engl. primitive type)

2. Tipovi nabrajanja (engl. enumeration type)

Kod primitivnih tipova postoje postojeći prosti tipovi podataka u samoj implementaciji (jednostavni tipovi). Među njih se ubrajaju cjelobrojni tipovi, brojevi s pokretnim zarezom, znakovni i logički. Tipovi nabrajanja uzimaju vrijednosti iz definiranog skupa simboličkih imena (primjerice, boja očiju i kose, spol, itd.). Enumeracija sadrži uređene

parove imenovanih identifikatora te njihovih pridruženih vrijednosti (nazivaju se još i obrojčani literali). Oni se koriste za praktičniji i razumljiviji opis raznih diskretnih vrijednosti sustava. Također, oni nisu povezani s klasama (kao i tipovi podataka).

4.1.10 Komentari

Komentari služe za učinkovitije razumijevanje opisanog sustava. Stoga, oni se ne upotrebljavaju često, ali i ne trebaju uvijek, odnosno na svakom dijagramu. Konkretnije, njihova svrha se krije u dodatnom opisivanju:

1. Razreda (nekih)
2. Atributa
3. Veza
4. Operacija

Komentari moraju biti jasni i točno izraženi kako bi obuhvatili bitne aspekte elemenata koji se opisuju.

Postoji mogućnost povezivanja s nekim dijelovima dijagrama, kao što su to primjerice, pridruživanje, klase, itd. Posebice, oni se mogu povezivati s nekim dijelovima dijagrama, kao što mogu biti navedeni i za cijeli dijagram. Komentari vezani za određeni element se označuju s neoznačenom vezom, dok se komentari definirani za cijeli dijagram nalaze na njegovom rubu ili u jednom od njegovih uglova.

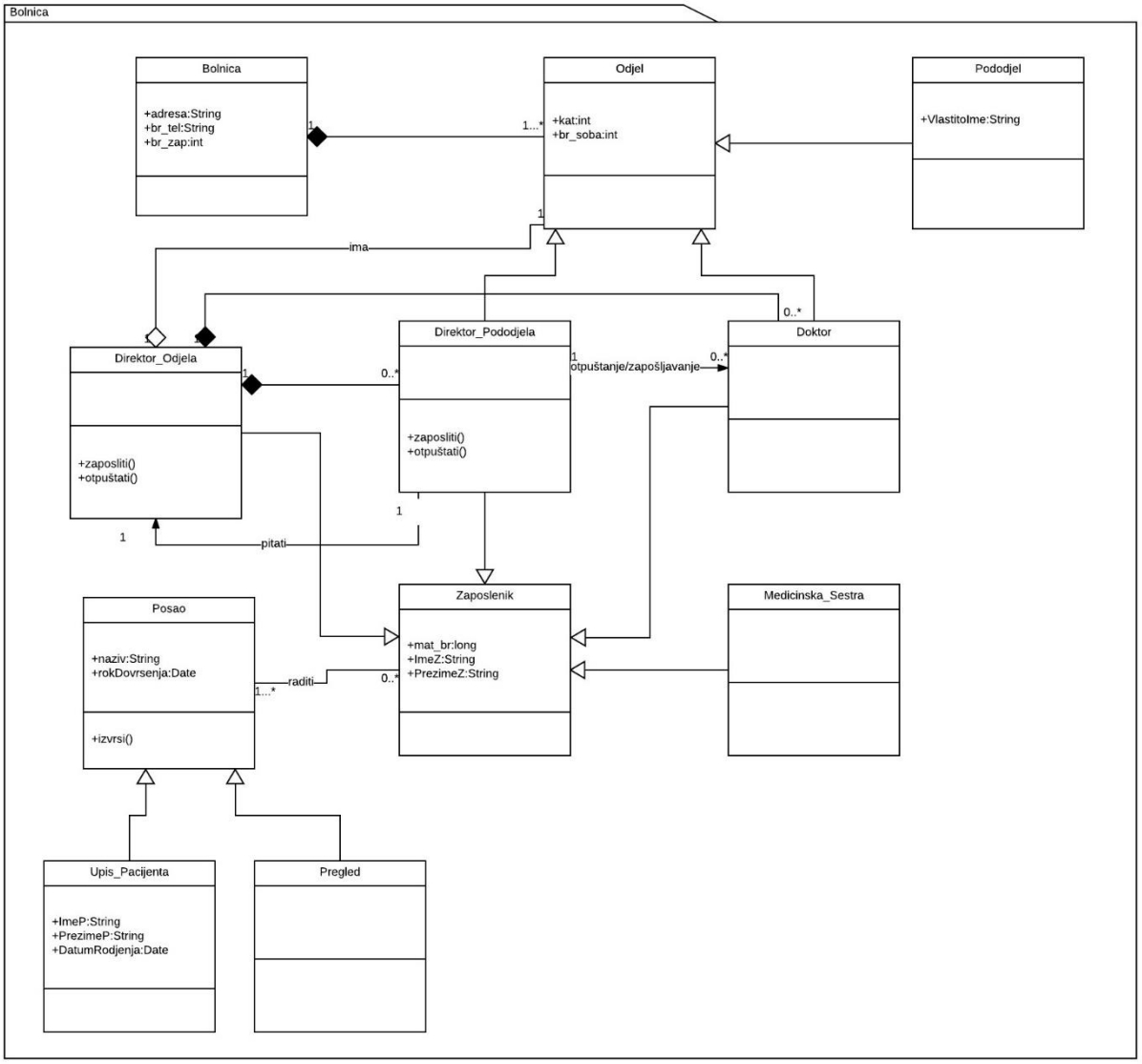
4.1.11 Vlastiti primjer

Tekst zadatka:

Bolnica se sastoji od jednog ili više odjela, a na čelu svakog odjela je direktor odjela. Svaki odjel može imati više pododjela gdje svaki pododjel ima svog direktora pododjela i radnike (doktore i medicinske sestre). Direktori mogu biti direktori odjela ili pododjela. Svaki zaposlenik može raditi više poslova, a svaki posao može obavljati nijedan ili više zaposlenika. Posao zaposlenika može biti vezan za upis pacijenta ili pregled. Svaki posao sadrži naziv (String) i rok dovršenja (Date). On ima i operaciju izvrši() koja je vidljiva svima. Ako je posao vezan za upis pacijenta, on sadrži još i ime (String), prezime (String) te datum rođenja (Date). Direktor odjela može zaposliti ili otpuštati sve

direktore pododjela i sve radnike u pododjelima (pritom su oni ovisni o njemu). Direktor pododjela ne može otpuštati niti zapošljavati radnika, ali može pitati direktora odjela da li da otpusti ili zaposli radnika. Bolnica ima svoju adresu (String), broj telefona (String) te broj zaposlenih (int). Svaki odjel ima svoj kat (int) i broj soba (int), a svaki pododjel osim naslijeđenog kata i broja soba, ima i vlastito ime (String). Svaki zaposlenik ima svoj matični broj u bolnici te ime i prezime (String).

Slika 5.: Bolnica



Izvor: Vlastiti rad

Opis slike:

Klasa bolnica je povezana s klasom odjel pomoću kompozicije (na slici označeno s punim rombom). To znači da ako se obriše bolnica, obrisat će se i odjel. Ta bolnica se sastoji od jednog ili više odjela, dok taj pojedini odjel pripada samo jednoj bolnici. Kod odjela se vrši veza nasljeđivanja, odnosno klase pododjel, direktor_ pododjela i doktor ga nasljeđuju (na slici označeno s trokutastom strelicom). U tom kontekstu, prethodno navedene klase nasljeđuju sve atribute i operacije od odjela. Klasa direktor_ odjela je povezana s klasom odjel pomoću agregacije (na slici označeno s praznim rombom). Veza je potom slabija od kompozicije te ona označava da samo klasa odjel sadrži klasu direktor_ odjel, tj. cjelina sadrži dio. U ovom slučaju, odjel ima jednog direktora odjela kao što i direktor odjela može raditi na samo jednom odjelu. Nadalje, direktor_ odjel je još povezan sa sljedećim klasama doktor i direktor_ pododjela putem veze kompozicije (ako se briše klasa direktor_ odjel, brišu se i klase doktor i direktor_ pododjela). Tako se vidi da direktor odjela ne mora, ali i može, otpustiti ili zaposliti doktora, dok doktor može biti otpušten ili zaposlen od samo jednog direktora odjela. Direktor odjela ne mora otpustiti, ni jednog ili više, te isto tako i zaposliti ni jednog ili više direktora pododjela. Direktor pododjela je spojen s klasom direktor_ odjel putem asocijacije (na slici označeno s običnom strelicom). To označava da direktor pododjela mora pitati direktora odjela prije nego što želi nekog zaposliti ili otpustiti. Direktor odjela je nadređen samo jednom direktoru pododjela (vrijedi i obrnuto). Direktor pododjela je povezan s klasom doktor putem asocijacije. Vidi se da direktor pododjela može otpustiti ili zaposliti ni jednog ili više doktora, dok taj doktor može biti otpušten ili zaposlen od strane jednog doktora. Klasu zaposlenik nasljeđuju sljedeće klase, a to su doktor, medicinska_sestra, direktor_pododjel i direktor_odjel (sve klase nasljeđuju njegove atribute). Klasa zaposlenik je povezana s klasom posao pomoću asocijacije. Konkretnije, zaposlenik može raditi jedan ili više poslova, dok taj posao ne mora obavljati nijedan ili više zaposlenika. Klase upis_ pacijenta i pregled nasljeđuju klasu posao, tj. njenu operaciju te atribute. Posao može biti u obliku upisa pacijenta ili pregleda. Na ovom primjeru su svi atributi te operacije vidljivi (označeno sa znakom „+“- public).

4.2 Dijagrami objekata

Dijagrami objekata (engl. object diagram) opisuju objekt kao fizičku stvar (nalazi se u samoj memoriji) te statički aspekt modela (prikazuju se samo veze, a ne interakcije preko tih veza).

Ovi dijagrami prikazuju primjere (objekte), apstrakcije (klase) i njihove veze. Veze kod ovog dijagrama služe za komuniciranje među objektima. (Anonymous_4, 2012., str. 2).

Ti dijagrami predstavljaju "snimak" pogleda na sustav u jednom trenutku. Pod jednim trenutkom se podrazumijeva prikazivanje objekata s njihovim trenutnim stanjem i njihovom trenutnom vezom. (Anonymous_4, 2012., str. 2).

Elementi dijagrama objekta su stvari (objekti i paketi) i relacije (veze između objekata i paketa). Dijagram sadrži i strukturu grafa gdje objekti predstavljaju čvorove, a veze grane.

4.2.1 Karakteristike objekata

Dijagram objekta služi za prikaz složene strukture. On se sastoji od više objekata koji služe za prikaz ponašanja kroz vrijeme preko niza povezanih objekata. Objekti predstavljaju samo primjer, a ne specifikaciju, odnosno definiciju modela. Naravno, on ima i dokumentacijsku svrhu te služi u pomaganju kod razumijevanja modela.

4.2.2 Definiranje objekata imenima

Objekt se definira na sljedeća dva načina:

1. Postojanje u memoriji.
2. Sadržavanje stanja, ponašanja i identiteta (primjer apstrakcije, tj. tipa i klase).

UML notacija se sastoji od:

1. Pravokutnika
2. Podvučenog imena
3. Objekta

Pravokutnik sadrži odjeljke s vrijednostima atributa. Pod podvučenim imenom se podrazumijeva objekt koji treba imati točno naznačeno ime. Objekt može biti konkretan ili prototipski te imenovan ili anoniman.

4.2.3 Veze između objekata

Veze se definiraju kao komunikacijski putevi između objekata. One su primjeri, odnosno instance asocijacija gdje jedna asocijacija, koja je između dvije klase, predstavlja skup veza definiranih između objekata tih klasa.

Na vezama se mogu nalaziti svi ukrasi osim multiplikativnosti. Nije potrebno da se ona nalazi jer uvijek iznosi jedan. Primjerice, opis hijerarhije objekata uzorka kompozicije gdje je sklop element koji sadrži druge elemente, dok je struktura te hijerarhije rekurzivna, a tipa je stabla.

Drugi primjer se odnosi na opis relacije “pohađa” između studenata i smjerova. U ovom kontekstu, asocijacija označava veze koje se mijenjaju u vremenu. Tako su u ovom primjeru, asocijacije različite u svakoj školskoj godini.

4.3 Dijagrami komponenata

4.3.1 Karakteristike dijagrama

Dijagrami komponenata (engl. component diagram) se koriste pri modeliranju, odnosno za izvorni kod, izdanje za isporuku, izvršna izdanja te njihovu isporuku i fizičke baze podataka.

Dio koji se koristi u dijagramu je komponenta. Svrha komponente se krije u zamjenjivanju dijela sustava gdje se pritom realizira skup sučelja. On prikazuje organizaciju te same zavisnosti između komponenti. Konkretnije, on služi za prikazivanje organizacije softverskog sustava i statičkog pogleda na implementaciju sustava.

UML definira standardne stereotipove za komponente, a to su:

1. Executable
2. Library
3. File
4. Document (dokument)
5. Script (skripta)
6. Source
7. Table

Executable predstavlja komponentu koja se izvršava na čvoru. Library definira statičku ili dinamičku objektnu biblioteku. File obuhvaća datoteku ili sadržaj. Source sadrži datoteku gdje se nalazi izvorni kod. Table specificira tablice iz baze podataka (verzija UML 1).

Paketi kod komponentnog dijagrama sadrže druge pakete i komponente. Ti paketi se koriste za predstavljanje fizičkog grupiranja komponenti i tipično prikazivanje datoteke u sustavu datoteka. (Anonymous_5, 2014., str. 11).

Logički paketi koji dolaze od klasnog dijagrama se preslikavaju u paketima kod komponentnog dijagrama.

Paketi ili komponente se povezuju relacijom zavisnosti koja prikazuje (Anonymous_5, 2014., str. 11):

1. Zavisnost u vrijeme prevođenja kada se radi o datotekama izvornog koda.

2. Zavisnost u vrijeme povezivanja kada se radi o bibliotečnim i objektnim datotekama.

3. Zavisnost u vrijeme izvršenja kada se radi o izvršnim datotekama.

4.3.2 Elementi dijagrama

Postoje sljedeći elementi komponentnog dijagrama, a to su:

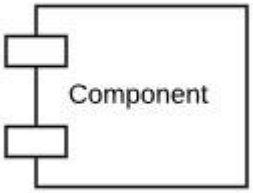
1. Komponenta

U ovom dijagramu, ona je modularni dio sustava. Pritom, zamjenjiva je u okruženju, odnosno zamjenjiva je s drugom komponentom. Uvjet zamjene je da ona mora podržavati isto sučelje. Stoga, kapsulira neki sadržaj koji je vidljiv samo kroz sučelje. Služi i za definiranje svog ponašanja kroz ponuđena i zahtijevana sučelja gdje je ona veća od klase i manja od cijelog sustava. Ponašanje komponente ovisi o ponuđenom i traženom sučelju.

U komponentnom dijagramu, ona predstavlja i tip, odnosno apstrakciju stvari. Komponenta (kada predstavlja apstrakciju stvari) mora obuhvaćati statičku i dinamičku semantiku. Postoji puno primjera komponenata od kojih su najpoznatije sljedeće Java Bean, EJB, Corba, COM+ i .NET assembly. Razvoj komponente se gradi i strukturira i oblikuje sučelje od već postojećih komponenti. Njegova bitna osobina je reupotreba komponenti koje su razvijane ranije i podržavaju zadana sučelja.

Postoje sličnosti sa samom klasom. Primjerice, u slučaju kada komponenta (koja sadrži samo operacije) predstavlja pakiranje klasa (sadrži atribute i operacije) u implementaciji. Isto tako postoje sličnosti i sa sučeljem gdje sučelje predstavlja skup operacija. Operacije potom specificiraju samu klasu ili komponentu (ovdje se realizira jedan ili više sučelja).

Komponenta se na dijagramu crta kao pravokutnik sa stereotipom <<component>> gdje se na rubu pravokutnika nalaze utikač i utičnica za ponuđeno ili traženo sučelje. Putem njih (utikača i utičnice) se crta sama veza između komponenti. Prikaz komponente je na sljedećoj slici:



2. Artefakt

Artefakt predstavlja fizičku informaciju (on ju sam koristi ili proizvodi kao razvojni proces ili izvršenje) te manifestaciju komponente.

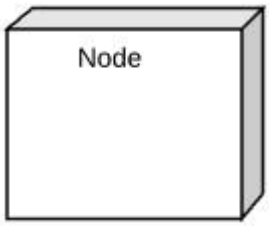
Neki od primjera artefakta su modeli, arhiva, tablice baza podataka te dokumenti.

Postoje tri vrste artefakta, a to su:

1. Oni koji pripadaju razvojnom procesu (primjerice, modeli, izvorni kod, skripte, resursi, itd.).
2. Oni koji služe za isporuku (primjerice, exe, dll, jar, dokumenti i tablice).
3. Oni koji su izvršni (primjerice, COM objekt kreiran iz DLL-a).

Izvršni artefakti su kreirani kao posljedica izvršenja.

Prikaz artefakta je na sljedećoj slici:



3. Sučelje (engl. interface)

Njegova svrha je navedena u predstavljanju relacije u obliku kanoničke ili skraćene forme. Ono je i imenovani skup svojstva te se pridružuje klasi, komponenti, podsustavu ili bilo kojem drugom klasifikatoru. Posebice, on definira usluge koje taj klasifikator pruža ili traži. (Manger, 2015d, str.3).

Klasifikator služi za razdvajanje samog sučelja od implementacije i funkcija. Smatra se da ako klasifikator realizira javno okruženje onda će i sama komponenta ili podsustav realizirati to isto sučelje. On se sastoji od definicija operacija i atributa.

Naravno, postoje i sljedeći standardi koji koriste sučelje, a to su COM, Corba i EJB (engl. enterprise java beans).

Sučelje se uspoređuje s nasljeđivanjem jer omogućuje polimorfizam.

Prednost pronalaženja sučelja se krije u većoj fleksibilnosti gdje se pritom ispituje svaka asocijacija. Poruke koje se onda šalju između objekata se izdvajaju te se onda grupe operacija i atributa mogu ponovno upotrijebiti. Također, bitno je izdvajati grupe operacija i atributa, tražiti klase (koje su različite ali imaju istu ulogu kod sustava) te razmišljati o mogućnostima proširenja našeg sustava u skoroj budućnosti.

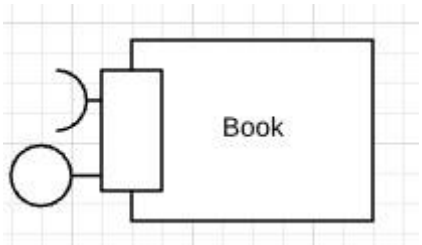
Prikaz sučelja je na sljedećim slikama:

4. Port

Kod ovog dijagrama, prikazuje se točka interakcije između klasifikatora i okruženja. Sučelja su pridružena portu na način da karakteriziraju zahtjeve koje prati klasifikator u pogledu okruženja, odnosno za pružanje usluga (tzv. zahtijevano sučelje). Štoviše, oni služe i za karakteriziranje zahtjeva koje ponuđeno sučelje kod okruženja pravi klasifikatoru, odnosno koje treba za svoj rad sučelja.

Komponenta se tu koristi putem portova gdje je ona semantički povezani skup ponuđenih i traženih sučelja. Ovdje port može imati i multiplikativnost koja se pritom piše iza imena u uglatim zagradama.

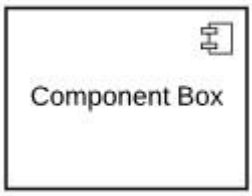
Na dijagramu se prikazuje u obliku pravokutnika iz kojeg izlazi utikač i utičnica. Crta se na rubu ikone koja predstavlja klasifikator. Njihova prednost korištenja se krije u strukturiranju ponuđenih i traženih sučelja te u jednostavnosti prikazivanja kod dijagrama. Prikaz porta je na sljedećoj slici:



5. Podsustavi

Podsustavi predstavljaju stereotip paketa (kod verzije UML 1), dok sada (kod verzije UML 2) oni predstavljaju stereotip komponente. To je komponenta na najvišoj razini koja se pojavljuje kod početnog dijeljenja velikog sustava u manje dijelove. Konkretnije, ona je relativno samostalna cjelina koja se pritom može izdvojiti iz sustava te se uključiti u neki drugi sustav. Promatra se kao crna kutija čije je ponašanje određeno s ponuđenim i traženim sučeljem.

Podsustav se na dijagramu crta kao komponenta sa stereotipom <<subsystem>>. Prikaz podsustava je na sljedećoj slici:



4.4 Dijagrami složene strukture

4.4.1 Karakteristike dijagrama

Dijagrami složene strukture (engl. composite structure diagram) služe za hijerarhijsku dekompoziciju pritom razdvajajući dijelove njegove unutarnje strukture i korištenje događaja suradnje u samoj suradnji.

Struktura ovog dijagrama se svodi na kompoziciju povezanih elemenata gdje pritom oni predstavljaju pojave. Te pojave surađuju preko veza kako bi postigli zajedničke ciljeve.

Postoji unutarnja struktura koja se nalazi unutar pojava klasifikatora ili suradnje. Unutarnju strukturu imaju:

1. Klase
2. Komponente
3. Suradnja

U ovom kontekstu, klase sadrže dijelove i portove povezane konektorima. Komponente imaju istu svrhu kao i klase. Suradnja definira uloge koje su pritom povezane konektorima te samim događajima suradnje.

Kao što postoji unutarnja struktura, tako postoji i port koji predstavlja točku interakcije klasifikatora s okruženjem. Kod ovog dijagrama se upotrebljava i strukturirana klasa, koja sadrži portove i unutarnju strukturu.

4.4.2 Elementi dijagrama

Postoje sljedeći elementi dijagrama složene strukture, a to su:

1. Portovi

Ovdje se oni povezuju sa zahtijevanim sučeljem, stvarnim sučeljem te unutarnjim dijelom. Ti dijelovi moraju imati na dijagramu naznačeno ime, tip i multiplikativnost.

Kod portova postoje i veze, a to su kompozicija i agregacija. Kompozicija se na dijagramu crta u obliku pravokutnika s punom linijom, dok se agregacija crta također u obliku pravokutnika, ali s isprekidanom linijom.

2. Konektori

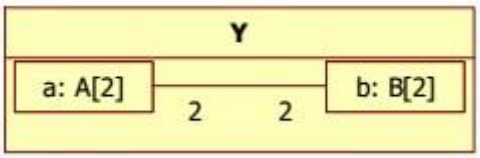
Ovaj element povezuje međusobno dijelove i dijelove s portovima (delegirajući konektor). Predstavlja zavisnost od porta prema dijelu za stvarno sučelje i od dijela prema portu za zahtijevano sučelje gdje on povezuje dijelove putem direktne veze. Ta direktna veza označava komunikaciju između dijelova (asocijacija) i označava vezu koja povezuje zahtijevano i stvarno sučelje.

Prikaz konektora je na sljedećoj slici:

3. Multiplikativnost

Ona se označava putem uglatih zagrada ili u gornjem desnom uglu. Na krajevima konektora se isto označava multiplikativnost veze. Konkretnije, ona predstavlja i broj objekata. Taj broj objekata, kada se gleda sa strane konektora, u vezi je s jednim objektom i nalazi se na drugoj strani veze (unutar jedne pojave klasifikatora).

Slika 6.: Prikaz multiplikativnosti



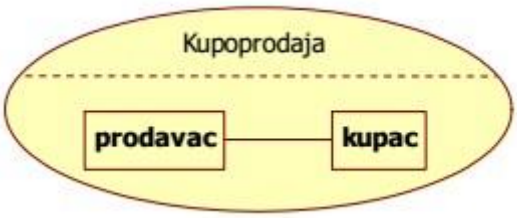
Izvor: Anonymous_6, 2014., str. 6

4. Suradnja

Suradnja služi za opis strukture samih elemenata koji imaju ulogu obavljanja funkcije u cilju postizanja funkcionalnosti. Stoga, između uloga se nalaze konektori.

Suradnja se na dijagramu pretežito označava na dva načina, a to su gornji dio elipse i ispod linije (nekad se označava i putem stereotipa <<occurrence>>).

Slika 7.: Prikaz suradnje



Izvor: Anonymous_6, 2014., str. 7

4.5 Paketni dijagrami

4.5.1 Karakteristike paketa

On pripada statičkim i strukturnim UML dijagramima.

Paket (engl. package) je UML- ov mehanizam za grupiranje “stvari”, a može služiti za (Manger, 2015b, str. 3):

1. Grupiranje semantički srodnih elemenata.

2. Definiranje “semantičke granice” u modelu.

3. Stvaranje učahurenog prostora imena (engl. encapsulated namespace) unutar kojeg sva imena moraju biti jedinstvena.

4. Odvajanje dijelova na kojima se može paralelno raditi tijekom oblikovanja.

Paket kod paketnog dijagrama služi za logičko grupiranje. Međutim, to ne znači da će sve stvari unutar nekog paketa biti i fizički implementirane u obliku cjeline. Kod fizičkog grupiranja koristimo komponente gdje će svaka stvar pripadati jednom paketu. U tom slučaju se događa sljedeće:

1. Paketi će nekad činiti hijerarhiju tako da će biti ugniježđeni jedan unutar drugog.

2. Paket će na vrhu biti označen putem stereotipa <<topLevel>>.

3. Po pravilu se svi elementi stavljaju u stereotip <<topLevel>>, ako nije drukčije navedeno.

4. Stvara se hijerarhija koja će sadržavati poseban prostor za imena kod ugniježđenih paketa.

Konkretno, kada se vrši analiza paketi će sadržavati slučajeve uporabe, klase (koje će se nalaziti na razini same analize) i realizaciju samih slučajeva uporabe.

Učahurenje prostora imena služi za definiranje granice gdje će sva imena biti jedinstvena. Stoga, u slučaju kada postoje jedinstvena imena, onda to znači da ne smiju postojati dva elementa koja će se isto zvati. Ako primjerice, postoji više prostora imena, onda mogu postojati elementi koji će se isto zvati. Isto tako, ako neki element hoće pristupiti elementu koji pripada tom istom prostoru, onda će on morati samo

definirati naziv tog elementa. Postoji i mogućnost da neki element hoće pristupiti elementu koji pritom nije iz njegovog prostora. U prethodno navedenom slučaju, definirat će se sama navigacija, odnosno put koji je potreban (kvalificirano ime ili ime puta elementa).

Kvalificirano ime se gradi na sljedeći način, a to je da se ispred imena elementa stavljaju prefiksi imena ugniježđenih paketa u kojima se taj element nalazi. Zatim se imena razdvajaju dvostrukom dvotočkom gdje se vanjski paketi imenuju prije unutrašnjih, odnosno u redoslijedu ugniježđenih. (Manger, 2015b, str. 10).

Javnim elementima koji pripadaju vanjskom paketu u pogledu nekvalificiranih imena pristupa se iz unutarnjeg paketa. U obrnutom slučaju, odnosno kada se želi pristupiti iz vanjskog paketa, onda se moraju koristiti kvalificirana imena.

U slučaju da se pronađe paket, radi se sljedeće:

1. Promatraju se klase na razni analize (primjerice, koherentnost grupa, generalizacijska hijerarhija klasa i klase koje su u bliskom odnosu putem ovisnosti ili asocijacija).
2. Promatraju se slučajevi uporabe.
3. Podešava se model paketa u svrhu maksimiziranja paketa i minimiziranja ovisnosti (primjerice, dodavanje i uklanjanje paketa, prebacivanje klasa, itd.).

U slučaju kada se promatraju slučajevi uporabe, definiraju se grupe koje podržavaju određeni poslovni proces ili aktera.

4.5.2 Vidljivost i ugnježđenje paketa

Ugniježđeni paketi se na dijagramu prikazuju na dva načina, a to su:

1. Jedan paket se nacrta unutar drugog paketa.

2. Crtaju se odvojeno, a potom ih se povezuje putem veze ugnježđenja.

Vidljivost (engl. visibility) se koristi kada postoji neki element koji je pritom smješten unutar nekog drugog paketa, ali je istodobno i vidljiv izvan tog paketa. Putem vidljivosti dolazi do smanjenja te kontroliranja povezanosti između paketa.

Postoje dvije vrste vidljivosti koje se na dijagramu označavaju sa + i - , a to su (Manger, 2015b, str. 8):

1. +...Javni

2. -...Privatni

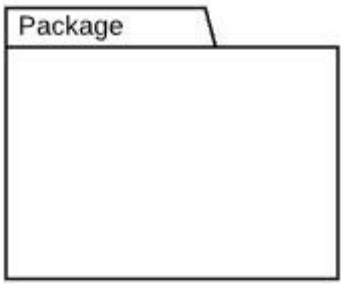
Kod javnog pristupa, element je vidljiv unutar svakog paketa te i iz drugih paketa. Privatni definira element koji je vidljiv unutar paketa, ali je sakriven za druge pakete.

4.5.3 Veze paketa

Paketni dijagram (engl. package diagram) definira pakete te veze između njih.

Paket se na dijagramu prikazuje putem ikone foldera, odnosno mape. No, ime paketa se može prikazivati i u samom tijelu ili u njegovom "uhu" (s riječi UpperCamelCase). Stoga, sve ono što sadrži paket ucrtava se u tijelo.

Prikaz paketa je na sljedećoj slici:



Veze između paketa su najčešće, a to su (Manger, 2015b, str. 5):

1. Ovisnosti sa stereotipima (isprekidane strelice)

2. Generalizacije (crtaju se kao kod klasa)

3. Ugniježđenost

Generalizacija je tip veze koji je analogan istoimenoj vezi kod klasnog dijagrama. Tako ona predstavlja vezu između onog općenitog paketa (ili osnovnog) te onog posebnog. Poseban paket služi za nasljeđivanje javnih elemenata koji se nalaze u općenitom paketu. On ima sljedeće karakteristike:

1. Dodavanje novih elemenata
2. Redefiniranje nekih elemenata iz općenitog paketa

Dodavanje novih elemenata se vrši kod općenitog paketa. Redefiniranje nekih elemenata iz općenitog paketa služi za definiranje elemenata koji imaju isti naziv.

Postoji još i tranzitivna te netranzitivna veza.

Tranzitivna veza ima sljedeće svojstvo, a to je ako je primjerice, A u vezi sa B, B je u vezi sa C, tada je i A u vezi sa C (stereotip <<import>>).

Netranzitivna veza ima sljedeća svojstva:

1. Javni elementi koji pripadaju C postaju privatni elementi u B.
2. Samo javni elementi iz B (bez importiranih iz C) postaju privatni u A.
3. Elementi iz A, upravo zbog prethodnog razloga, ne mogu pristupiti elementima iz C (stereotip <<access>>).

4.5.4 Stereotipovi paketa

Najvažnije vrste ovisnosti između paketa sa stereotipom su sljedeće:

1. <<use>>
2. <<import>>
3. <<access>>

Stereotip <<use>> se definira kada se u paketu- klijenta koristi javni element koji pripada paketu- dobavljaču. Stereotip <<import>> specificira dobavljačev prostor imena kamo su dodani javni elementi koji se nalaze u klijentovom prostoru. Upravo zbog tog razloga, elementi koji se nalaze u klijentu mogu pristupiti javnim elementima u dobavljaču koristeći nekvalificirana imena. Stereotip <<access>> omogućuje javnim elementima koji se nalaze u dobavljačevom prostoru, dodavanje u obliku privatnog elementa u klijentov prostor (razlog isti kao i kod stereotipa <<import>>).

4.6 Dijagrami rasporeda

4.6.1 Karakteristike rasporeda

On pripada statičkim i strukturnim UML dijagramima i dopunjuje dijagram komponenti.

Dijagrami rasporeda (engl. deployment diagram) (negdje se može naći i naziv dijagram razmještaja) opisuju sklopovlje (hardver) i programsku potporu (softver). Hardver i softver se koriste u samom razvoju sustava, odnosno u njegovom radnom i produkcijskom okruženju. Konkretno, ovaj dijagram se sastoji od čvorova, komponenti i njihovih veza.

Ti dijagrami služe i za prikazivanje računalnih resursa koji su neophodni za funkcioniranje sustava i njihovih odnosa.

U ovaj dijagram spadaju:

1. Stvarni i virtualni uređaji (primjerice, poslužitelji, osobna računala, radne stanice, itd.)
2. Komponente (koje se nad njima izvršavaju)
3. Veze između resursa

Između čvorova i komponenti postoje dvije velike razlike, a to su (Jović i sur., 2013., str. 78):

1. Komponente sudjeluju u izvršavanju sustava, a čvorovi izvršavaju komponente. U ovom slučaju, čvorovi koji izvršavaju komponente predstavljaju računalne resurse koji omogućuju pokretanje i izvršavanje komponenti.

2. Čvorovi predstavljaju fizički aspekt sustava, a komponente logički.

Čvorovi i veze predstavljaju samo sklopovlje računala (primjerice, poslužitelji), dok komponente mogu biti izvršne datoteke (.exe), stolne, mobilne i mrežne aplikacije. Ovdje jedan čvor može uključivati više računala, što onda predstavlja grozd poslužitelja. Čvorovi se prikazuju kao kocke, a komponente svojim posebnim simbolom. Svaki čvor na dijagramu mora imati svoje jedinstveno ime, tj. ime čvora. Ime puta predstavlja ime paketa kojemu taj čvor pripada (primjerice, server::backup) i prethodi imenu čvora. Oblik koji predstavlja ime čvora skupa s nazivom puta je zapravo

proširenі naziv. Čvorovi koji su zajedno s drugim paketom se zovu prošireni čvorovi, a oni koji imaju jednostavan naziv zovu se jednostavnim čvorom.

U praksi se smatra da je najbolje primjerice, poslužitelje (a tako i ostale stvari) nazvati prema stvarnom imenu (kako bi odmah znali o kakvom je primjerice, poslužitelju riječ).

Kod dijagrama razreda postoje i stereotipovi koji proširuju čvorove (<<processor>> i <<device>>). Za razliku od stereotipova, komponente se ne mogu proširivati te one mogu označavati pojedince čvorova i komponente (sličan objektnom dijagramu). Njih se prepoznaje po podcrtanim nazivima. U svakom dijagramu razreda je moguće prikazati ili samo pojedince ili samo definicije.

4.6.2 Odnosi između rasporeda

Postoje veze čvorova koje se definiraju jednosmjernom (ne tako često) ili dvosmjernom vezom gdje one označavaju put prijenosa podataka između čvorova (primjerice, TCP/IP, HTTP, .NET Remoting, itd.).

Čvorovi i komponente su jedne o drugima ovisne.

Smjer je definiran kao usmjerenje od elementa (koji sadrži informaciju ili poruku te isporučitelja) do klijenta koji ju koristi. Naravno, kod odnosa između rasporeda, postoje i komponente koje su ovisne jedna o drugoj te su onda te veze povezane putem usmjerene veze ovisnosti. U slučaju da komponenta ima aplikativno sučelje, ona će dodirivati simbol sučelja komponente.

4.7 Dijagrami profila

4.7.1 Karakteristike dijagrama

Dijagrami profila (engl. profile diagram) spadaju u pomoćne UML dijagrame. Taj dijagram nije postojao kod UML 1 verzije, već od verzije UML 2.

Dijagram profila omogućuje definiranje:

1. Novih i posebnih stereotipova (kao klase)
2. Njihovih oznaka i ograničenja
3. Tehnologije
4. Primjene
5. Metode

Ovaj dijagram djeluje na razini metamodela. Pojam metamodel predstavlja model modela, dok metamodeliranje predstavlja proces stvaranja takvih modela. Pri tom se misli na analizu, konstrukciju i razvoj, pravila, ograničenja modela i teoriju (korisni za modeliranje neke unaprijed definirane klase).

4.7.2 Elementi dijagrama

Elementi dijagrama profila su sljedeći:

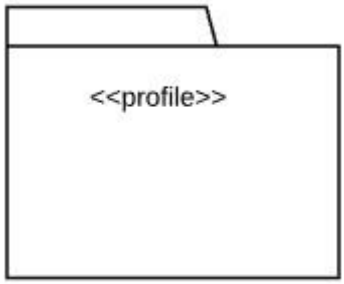
1. Profili

Ovaj element omogućuje da se izvorni model prilagodi različitim platformama za rad (primjerice, Java i .NET okruženje) i domenama (primjerice, modeliranje poslovnih procesa).

Profil definira:

1. Razrede
2. Stereotipove
3. Primitivne tipove i tipove podataka
4. Nabrajanje

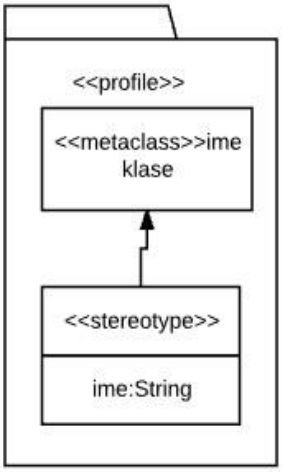
Na dijagramu se označava kao i paket. Prikaz oznake profila je na sljedećoj slici:



2. Stereotip <<metaclass>>

On se proširuje s posebnim (jednim ili više) stereotipa.

Prikaz stereotipa je na sljedećoj slici:



3. Veze

One se na dijagramu označavaju putem strelice (iscrtane ili obične).

5. Dijagrami međudjelovanja

5.1 Dijagrami komunikacije

5.1.1 Karakteristike dijagrama

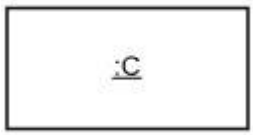
Dijagrami komunikacije (engl. communication diagram) ili suradnje te kolaboracije (stariji naziv) pripadaju dijagramima međudjelovanja te prikazuju dinamiku ponašanja, odnosno usredotočeni su na samo slanje poruka. Druga svrha je ta, da oni tako služe i za definiranje “tko kome šalje” poruke. To znači da on ne prikazuje “kada” se šalje poruka, već je to uloga sekvencijskog dijagrama (dijagrama slijeda). Ovi dijagrami su usmjereni na strukturne odnose te na djelovanje pojedinih instanci.

5.1.2 Elementi dijagrama

Postoje sljedeći elementi dijagrama komunikacije, a to su:

1. Objekt ili razred

Taj element se na dijagramu označava sa slovom :c (predstavlja klijenta) koji je podcrtan. Prikaz objekta je na sljedećoj slici:

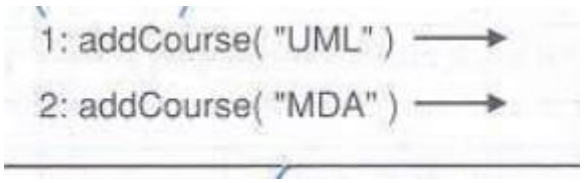


2. Veza

Ovaj element povezuje objekt s akterom.

Ona se na dijagramu označava kao obična linija i pokraj nje se nalaze strelice koje označavaju njezin smjer. Iznad tih strelica je naznačena radnja koja se treba izvršiti.

Slika 8.: Prikaz veze

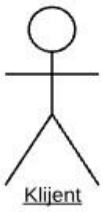


Izvor: Manger, 2015c, str.34

3. Akter

U ovom dijagramu, on može predstavljati primjerice, klijenta ili putnika, itd.

Oni se (kao i kod dijagrama slučaja uporabe i sekvencijskog) označavaju kao jednostavni čovječuljak. Međutim, postoji razlika kod ovog dijagrama, a to je da su oni podcrtani (točno definirani). Prikaz aktera je na sljedećoj slici:



5.2 Dijagrami slijeda

5.2.1 Karakteristike dijagrama

Dijagrami slijeda ili sekvencijski dijagrami (engl. sequence diagram) pripadaju skupini dijagrama ponašanja (kao i dijagrami stanja, slučaja korištenja i aktivnosti) i dinamičkim UML dijagramima. Konkretno, on je realizacija dijagrama korištenja ako su oni prethodno definirani. Takvi dijagrami su onda nužni u modeliranju, odnosno za prikaz rada sustava u realnom vremenu.

Kod ovog dijagrama se definiraju:

1. Objekti
2. Veze
3. Poruke

Dijagram slijeda daje naglasak na vrijeme (vertikalna dimenzija), odnosno on daje naglasak na redoslijed gdje se odvija međudjelovanje sudionika u nekom sustavu. Vrijeme uvijek na dijagramu ide od vrha prema dnu. Stoga, sudionici (korisnici) i sustavi predstavljaju vremenske pravce. Također, postoji i horizontalna dimenzija koja određuje objekte.

Ovaj dijagram prikazuje i poruke koje potom razmjenjuju sami sudionici ili objekti (kako bi se izvršila određena operacija) te služi za kreiranje i uništavanje života objekata.

Dijagram slijeda pokazuje i način na koji objekti surađuju u nekom određenom scenariju gdje suradnju oni prikazuju upravo putem poruka. (Stephens, 2015., str. 113).

Bitno je napomenuti da ako oznaka uključuje i ime i klasu, one moraju biti odvojene dvotočkom. (Stephens, 2015., str. 113).

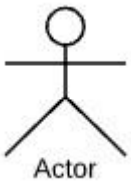
5.2.2 Elementi dijagrama

Postoje sljedeći elementi dijagrama slijeda, a to su:

1. Sudionici

U ovom dijagramu oni se modeliraju (objekti), te se kod ovog elementa prikazuje komunikacija između instanci klasa.

Oni se na dijagramu prikazuju u obliku pravokutnika koji onda sadrži naziv sudionika koji je podcrtan. Prikaz sudionika je na sljedećoj slici:

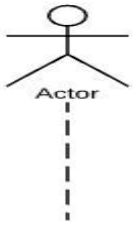


2. Životna linija sudionika

Redoslijed tokova bira se tako da se smanji presijecanje crta. Svaki tok ima crtkani vertikalni rep koji označava njegovo trajanje. Aktivacija je označena duguljastim vertikalnim pravokutnikom preko repa te se na njemu (repu) mogu označiti i promjene stanja dotičnog klasifikatora (promjene su obično uzrokovane prijemom poruke). Novo stanje crta se kao zaobljeni pravokutnik. (Manger, 2015c, str. 9).

Životna linija služi i za boravak sudionika unutar sustava koji se modelira. Postoji i mogućnost da životna linija, odnosno tok šalje poruku samom sebi, te se to onda naziva samodelegacija („razgovor sa samim sobom“). Pomoću delegacije nastaje ugniježđena aktivacija te je ona poredana s lijeva na desno.

Ona se na dijagramu crta kao isprekidana ili puna linija ispod sudionika. Prikaz životne linije je prikazan na sljedećoj slici:



3. Poruke

One se razmjenjuju u određenom vremenu između sudionika (predstavlja iteraciju). Ako su one na dijagramu označene sa zvjezdicom, znači da će se poruka slati dok god je neki uvjet ispunjen. U slučaju da nema zvjezdice, poruka se onda šalje samo jednom ako je uvjet ispunjen.

Ovdje je bitno reći da se one na dijagramu prikazuju u obliku vodoravnih strelica. Te vodoravne strelice potom spajaju repove životnih tokova te su isto tako poredane u određenom vremenskom redoslijedu. Izgled poruke je na sljedećoj slici:

Tipovi poruka koji se odvijaju na sekvencijskom dijagramu su:

A) “Query event”

Ova poruka služi za zahtjev ili prikaz informacija (na dijagramu se označava s <<Q>>).

B) “Mutation event”

Ona označava promjenu podataka u sustavu (na dijagramu se označava s <<M>>).

U slučaju postojanja nekog složenijeg dijagrama, potrebno je osigurati uvjetno i iterativno slanje poruka. Takvo slanje poruka se radi pomoću operatora ili elemenata poruke gdje se oni smatraju analognim naredbama koje se povezuju s grananjem ili petljama iz programskih jezika.

Nadalje, ovdje se javlja pojam operand koji služi za definiranje poruke na koju se operator odnosi. Oni predstavljaju sve poruke koje su se našle unutar okvira. Primjerice, može postojati zadan neki logički uvjet koji vraća istinu ili laž (stražar). Prethodno definirani elementi, na dijagramu se prikazuju unutar posebnog okvira koji se naziva kombinirani fragment. Operator i stražar se na dijagramu prikazuju u gornjem lijevom kutu, dok se stražar pojedinog operanda prikazuje uz strelicu. Kod ovog dijagrama, postoje i stražari koji se označavaju u pseudokodu pomoću uglatih zagrada.

Elementi poruka su:

A) Iteracija

Na dijagramu se označava s riječi “LOOP” u pravokutniku u gornjem lijevom kutu, a ispod nje (unutar pravokutnika) se nalazi petlja. Ona je kao for koji se koristi u nekim programskim jezicima (“dok god ima nečega”).

B) Uvjetno izvođenje

Na dijagramu se označava na isti način kao i iteracija. Razlika je samo u tome što u gornjem kutu piše “Alternative” te se nakon toga definiraju uvjeti (primjerice, zaliha>0 (kao prvi uvjet) te (kao drugi uvjet) ako je zaliha<=0, unutar uglatih zagrada).

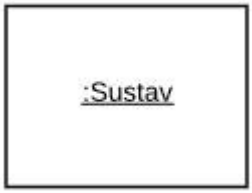
C) Opcionalno izvođenje

Opcionalno izvođenje se u gornjem lijevom kutu označava s riječi “Optional”. Koristi se ako se primjerice, želi dodati neki dodatni komentar ili napomena.

4. Sustav

Ovaj element se na dijagramu označava kao pravokutnik i unutar njega piše se riječ “Sustav” koji je podcrtan. Crta se u ravnini sudionika.

Prikaz sustava je na sljedećoj slici:



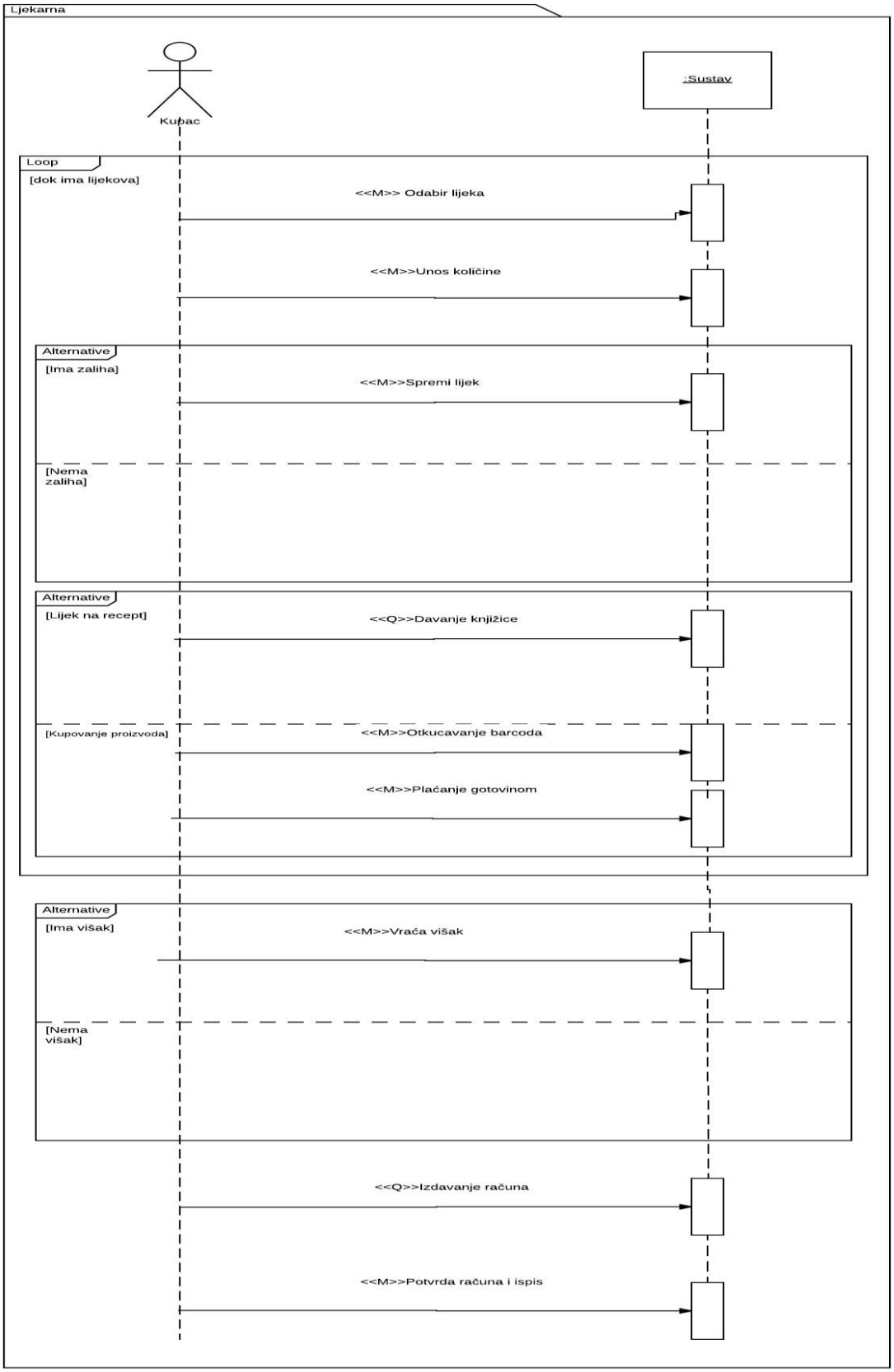
Na lijevom rubu dijagrama se nalaze objašnjenja. Uz njih se mogu pojaviti još i oznake te ograničenja (na dijagramu se označavaju putem vitičastih zagrada).

5.2.3 Vlastiti primjer

Tekst zadatka:

Proces započinje odabirom lijeka u ljekarni te željene količine. Pritom se izdaje račun koji se ponavlja za svaki lijek. Kod odabira svakog lijeka; ako ih nema dovoljno u zalihi, ne sprema se lijek. Ako kupac uzima lijek na recept, treba dati svoju zdravstvenu knjižicu, a ako kupuje neki proizvod (primjerice, neki lijek koji nije na recept) onda treba platiti gotovinom (u slučaju viška vraća mu se ostatak). Blagajnik na kasi otkucava barcode. Posljednje, vrši se potvrda računa te započinje ispis.

Slika 9.: Ljekarna



Izvor: Vlastiti rad

Opis slike:

Na dijagramu je prikazan kupac te sustav s kojim je u interakciji. Tako proces započinje odabirom lijeka u ljekarni (pomoću petlje loop te oznakom <<M>>) i unosom količine. Potom je naznačen alternate (kao uvjet) koji ako ima zaliha spremi lijek, a ako nema, ne događa se ništa. On se nalazi unutar petlje loop, kao i sljedeći alternate. Kod njega se provjerava je li lijek na recept, ako je, kupac daje knjižicu (prikazano pomoću <<Q>>). Ako kupac odluči kupiti neki proizvod, onda se otkucava barcode te se plaća gotovinom. Izvan loop-a postoji još jedan alternate, koji ako ima viška novca, vraća višak, a ako ga nema, ne događa se ništa. Posljednje navedeno se nalazi izvan alternate-a. Tako je definirano postojanje i izdavanje računa te potvrda računa i ispis s čime završava dijagram.

5.3 Vremenski dijagrami

5.3.1 Karakteristike dijagrama

Vremenski dijagrami (engl. timing diagram) spadaju u dijagrame međudjelovanja, a pojavili su se prvi puta u 2. UML- ovoj verziji.

Vremenski dijagrami se nikad ne koriste u analizi, a rijetko u oblikovanju. U slučaju da se koriste za oblikovanje, onda se odnose za real- time ili embeded sustave. Pod *ugrađenim sustavima* (embeded) se podrazumijevaju računalni sustavi s namjenskom funkcijom unutar većeg sustava često s ograničenjima računanja u realnom vremenu. Njihova zadaća je da oni mogu omogućavati opisivanje preciznih vremenskih ograničenja. Kod njih postoji i horizontalni oblik vremenskog dijagrama gdje su stanja prikazana horizontalno.

Ovi dijagrami služe za prikazivanje jednog ili više objekta/objekata u stanju nekog vremena. Na prethodno navedeni način, vremenski dijagram podsjeća na sekvencijski dijagram, što znači da oni mogu biti korisni u odlučivanju koliko će trajati različiti dijelovi nekog scenarija. Tako složenije verzije dijagrama pokazuju više sudionika. Konkretno, kada se pokazuje više sudionika, onda ovaj dijagram služi za prikazivanje poruka između njih samih. (Stephens, 2015., str. 115).

Vremenski dijagrami se mogu još koristiti i za prikazivanje vremenskih ograničenja u interakciji između nekoliko životnih tokova. Ako ti životni tokovi pripadaju aktivnim objektima, tada takav dijagram predstavlja još jedan način prikazivanja istovremenosti. Dijagram s više životnih tokova podijeljen je u više vodoravnih odjeljaka svaki za pojedini životni tok. Istovremeni događaji u različitim životnim tokovima nalaze se okomito jedan iznad drugog ili jedan ispod drugog. (Manger, 2015e, str. 26).

5.3.2 Elementi dijagrama

Postoje sljedeći elementi vremenskog dijagrama, a to su:

1. Horizontalna os

2. Vertikalna os

3. Promjena stanja životnih objekata

Horizontalna os na ovom dijagramu predstavlja vrijeme. Vertikalna os specificira prikazivanje životnih tokova te njihovih stanja. Promjena stanja životnih objekata se na dijagramu označuje s izlomljenom crtom.

5.4 Dijagrami pregleda međudjelovanja

5.4.1 Karakteristike dijagrama

Dijagram pregleda međudjelovanja ili dijagram interakcije (engl. interaction overview diagram) se rijetko koristi. Pod pojmom interakcija se podrazumijeva skup uređenih podataka gdje onda svaki podatak specificira jednu komunikaciju (primjerice, slanje signala ili operacija koje treba pozvati). Koristeći ga, on služi za prikazivanje odnosa između samih interakcija te u tu svrhu on, na jednom mjestu, prikazuje sve opcije vezane za međudjelovanje. Takav dijagram se koristi i za modeliranje sustava cijelog toka kontrole.

On je sličan dijagramu aktivnosti i dijagramu interakcija (često se koristi i taj naziv). Razlika je jedino u tome što su glavni čvorovi zamijenjeni fragmentima međudjelovanja

ili ponekad dijelovima sekvencijskog dijagrama. Fragmenti moraju biti točno raspoređeni po redoslijedu gdje čvorovi predstavljaju dijagram interakcije.

Čvorovi mogu biti okviri koji će sadržavati neke druge vrste dijagrama (ne samo dijagram aktivnosti i sekvencijski dijagram). U tom slučaju, on služi za prikaz više detalja glede čvorova koji pritom služe za komplicirane zadatke. (Stephens, 2015., str. 115).

Kao što postoje sličnosti, tako postoji i razlika u odnosu na dijagram interakcije. Tako je ta razlika definirana u tome, što dijagram interakcije ima dva nova elementa, a to su točke interakcije i elementi interakcije.

5.4.2 Elementi dijagrama

Dijagram pregleda međudjelovanja ima sljedeće elemente, a to su:

1. Čvor početka
2. Čvor završetka
3. Čvor spajanja
4. Čvor odlučivanja
5. Čvor priključivanja

Zaključak

Tema ovog završnog rada je bila korištenje UML dijagrama u programskom inženjerstvu s naglaskom na samo modeliranje, ali i izradu dijagrama. Tako je bitno zaključiti, da se modeliranje koristi pri rješavanju složenih problema na način da ih pojednostavljuje i da ih sami korisnici bolje razumiju. Isto tako, izrada modela služi da bi dobili cjelovitu sliku nekog sustava koji se promatra iz različitih perspektiva. Pojavljuju se unutar utvrđivanja zahtjeva koji obuhvaćaju aktivnosti (primjerice, otkrivanje i analiziranje zahtjeva).

Kod modeliranja konteksta, odnosno interakcije, trebaju se odrediti same granice sustava (vrši se na početku). Samo modeliranje konteksta smanjuje troškove te vrijeme izgradnje sustava, dok modeliranje interakcije naglašava problem u komunikaciji do kojeg može doći. Naravno, interakcije ne mogu biti do kraja razrađene jer ne ovise o implementaciji.

Ovdje pripadaju sljedeći dijagrami (navedeni najvažniji), a to su klasni dijagrami, dijagrami obrazaca uporabe te sekvencijski dijagrami.

Kod dijagrama obrazaca uporabe, bitno je naglasiti da oni služe isključivo za interakciju između aktera i samog sustava. Akter u ovom dijagramu predstavlja čovječuljka (živ ili neživ) te taj element stupa u vezu s našim sustavom. Obrazac uporabe predstavlja samu radnju koju obavlja akter. Bitno je reći da ovaj dijagram imenuje veze, dok ne služi za opisivanje kako se zapravo te veze obavljaju.

Sekvencijski dijagram se razlikuje od dijagrama obrazaca uporabe upravo po tome što on detaljnije opisuje vezu između aktera izvan sustava, ali i isto tako unutar sustava.

Pod modeliranjem strukture se misli na organizaciju ili građu sustava (dizajniranje). Tako postoje dvije vrste modela, a to su statički (ranije faze razvoja softvera) i dinamički (kasnije faze razvoja softvera).

Ova vrsta modeliranja se svodi na klasni dijagram. On se sastoji od pravokutnika (predstavljaju klase) i spojnica (predstavljaju veze) te se na tim vezama prikazuju brojevi. Ti brojevi služe za prikaz koliko objekata jedne klase je povezano s drugom klasom.

Klasni dijagram se često povezuje s Chenovim dijagramima iz razloga što izgledaju dosta slično. Štoviše, same klase se mogu i detaljnije prikazati dodajući još i popis atributa te popis operacija koje ta klasa izvršava. Nadalje, detaljna definicija klase služi i za analizu zahtjeva.

Posljednja vrsta modela su modeli ponašanja. Ova vrsta modela služi za opisivanje promjena u nekom sustavu za vrijeme obavljanja rada.

Ovdje spadaju dijagrami toka podataka (1.oblik u softverskom inženjerstvu) gdje se oni rijetko (ili više i ne) koriste, pa su ih zamijenili dijagrami aktivnosti. Kod tih dijagrama, ovali predstavljaju aktivnosti, pravokutnici služe za podatke, a strelica predstavlja tok podataka. Primjerice, kada se želi prikazati neka aktivnost ili neka radnja koju obavljaju korisnici nekog sustava onda će se za to upotrijebiti dijagram aktivnosti. Njega će korisnici upotrijebiti jer on služi za opisivanje samih aktivnosti, odnosno za promjene u nekom sustavu za vrijeme njegovog rada.

U ovom modelu, spadaju još sekvencijski dijagrami te dijagrami stanja.

Dijagrami stanja ovdje služe za prikazivanje stanja sustava u kojima se objekt može naći (crtaju se u obliku ovala). Strelice u ovom kontekstu, služe za prijelaze iz jednog stanja u drugo stanje. Naglasak je kod ovog dijagrama na postojanje problema koji se uopće mogu javiti. Prethodno navedeni problem je taj da stanja brzo rastu u slučaju kada sustav postaje složeniji. Taj problem se onda rješava na način da se uvode stanja koja pritom izgledaju kao jedno stanje.

Kod modela ponašanja, može doći do reagiranja sustava na sljedeća dva načina, a to su putem podataka (neki podatak se pojavi te ga sustav mora obraditi) ili putem događaja (mogućnost slanja nekih podataka kada se dogodi neki događaj na koji sustav mora reagirati). Međutim, postoji i razlika između samog podatka i događaja. Kod pojma podatka, sustav sam odlučuje kada će ga obraditi, dok na događaj mora odmah reagirati (mora prekinuti ono što je u tom trenutku radio).

Rad nekog sustava u budućnosti dijeli se na slučajeve korištenja koji predstavljaju skup inačica. Te inačice predstavljaju niz akcija koje sustav izvršava te se njima postiže rezultat koji je namijenjen određenom sudioniku. Kod modeliranja, ono nam služi za modeliranje statike (primjerice, klasni dijagram) i dinamike (obuhvaća skup više dijagrama).

U ovom završnom radu nije pisano o softverskom inženjerstvu u cijelosti. Stoga, softversko inženjerstvo je inženjerska grana koja je povezana s razvojem programskog proizvoda pritom koristeći dobro definirane znanstvene principe, metode i postupke. Ishod softverskog inženjerstva je tako učinkoviti i pouzdani softverski proizvod. Upravljanje softverskim projektom obuhvaća veći opseg od procesa softverskog inženjerstva jer uključuje komunikaciju, podršku prije i nakon isporuke, itd. Naime, ovdje je opisan samo jedan dio koji predstavlja građu samog softvera, odnosno sustava. Ukratko, ovaj rad je fokusiran na modeliranje, vrste modela i načine njihova korištenja te naposljetku i vrste samih dijagrama. Tako je UML namijenjen objektno-orijentiranim metodama analize i dizajna sustava. Druga svrha se krije u tome da je on grafički jezik. To u principu znači da on veći dio sustava opisuje slikom. Svaki dijagram onda sadrži svoje elemente putem kojih se crta. Isto tako, u ovom radu je naglasak na lakšem razumijevanju i razlikovanju dijagrama kako bi znali koji dijagram kada upotrijebiti jer je svaki dijagram namijenjen za nešto drugo.

S obzirom na stalnu prisutnost modeliranja te korištenje dijagrama, potrebno je shvatiti korisnike kojima je potrebno korištenje modeliranja u poslu. Potrebno je iz razloga jer softver, ali i modeliranje, treba biti što bolje i kvalitetnije u smislu da pomogne samim korisnicima.

Literatura

1) Anonymous_1 (n.d.) *Klasni dijagram*. Dostupno na: https://www.scribd.com/document/85835300/06-Klasni-dijagram, pristup: 21.2.2017
2) Anonymous_2 (n.d.) *Oblikovanje implementacije- Dijagram aktivnosti*, dostupno na: www.efos.unios.hr/oblikovanje-implementacija-is/wp/DijagramAktivnosti.docx, pristup: 18.4.2017.
3) Anonymous_3 (n.d.) *Projektiranje softvera- Dijagrami klasa*, dostupno na: http://rti.etf.bg.ac.rs/rti/ir4ps/predavanja/UML/03%20Dijagrami%20Klasa.pdf, pristup: 21.2.2017.
4) Anonymous_4 (2012.) *Projektiranje softvera- Dijagrami objekata*. Dostupno na: rti.etf.bg.ac.rs/rti/ir4ps/predavanja/UML/05%20Dijagrami%20Objekata.pdf, pristup: 15.3.2017.
5) Anonymous_5 (2014.) *Projektiranje softvera- Dijagrami komponenata*. Dostupno na: http://rti.etf.bg.ac.rs/rti/ir4ps/predavanja/UML/11%20Dijagrami%20Komponenata.pdf, pristup: 15.3.2017.
6) Anonymous_6 (2014.) *Projektiranje softvera- Dijagrami složene strukture*. Dostupno na: rti.etf.bg.ac.rs/rti/ir4ps/predavanja/UML/20%20Dijagrami%20Slozene%20Strukture.pdf, pristup: 15.3. 2017.
7) Jović, A., Horvat, M. i Grudenić, I. (2013.) *Priručnik UML dijagrama*. Zagreb: Sveučilište u Zagrebu; dostupno na: https://www.scribd.com/document/252441750/PrirucnikUML-dijagrami, pristup: 13.3.2017.
8) Lethbridge C. T. i Langaniere R. (2005.) *Object- Oriented Software Engineering; Practical Software Development using UML and Java*. Second edition. McGraw- Hill Education:British
9) Manger, R. (2013.) *Softversko inženjerstvo*. Skripta, nadopunjeno drugo izdanje. Zagreb: Sveučilište u Zagrebu
10) Manger, R. (2015a) *Softversko inženjerstvo; Vježbe 1: Uvod u UML i UP.* Zagreb: Sveučilište u Zagrebu; dostupno na:

web.studenti.math.pmf.unizg.hr/~manger/si/SI-vjezbe-01.pdf, pristup: 18.4.2017.

11) Manger, R. (2015b) *Softversko inženjerstvo; Vježbe 6: Paketi i njihovo korištenje u analizi.* Zagreb: Sveučilište u Zagrebu; dostupno na stranici: web.studenti.math.pmf.unizg.hr/~manger/si/SI-vjezbe-06.pdf, pristup: 20.4.2017.
12) Manger, R. (2015c) *Softversko inženjerstvo; Vježbe 7:Realizacija use caseova.* Zagreb: Sveučilište u Zagrebu; dostupno na: web.studenti.math.pmf.unizg.hr/~manger/si/SI-vjezbe-07.pdf, pristup: 20.4.2017.
13) Manger, R. (2015d) *Softversko inženjerstvo; Vježbe 11: Sučelja, komponente, podsustavi.* Zagreb: Sveučilište u Zagrebu; dostupno na: web.studenti.math.pmf.unizg.hr/~manger/si/SI-vjezbe-11.pdf, pristup: 30.4.2017.
14) Manger, R. (2015e) *Softversko inženjerstvo; Vježbe 12: Realizacija use caseova na razini oblikovanja.* Zagreb: Sveučilište u Zagrebu; dostupno na stranici: web.studenti.math.pmf.unizg.hr/~manger/si/SI-vjezbe-12.pdf, pristup: 30.4.2017.
15) Manger, R. (2015f) *Softversko inženjerstvo; Vježbe 13: State machine dijagrami.* Zagreb: Sveučilište u Zagrebu; dostupno na: web.studenti.math.pmf.unizg.hr/~manger/si/SI-vjezbe-13.pdf, pristup: 10.5.2017.
16) Požgaj, Ž. (n.d.) *Tehnike oblikovanja*. Dostupno na: http://web.efzg.hr/dok/inf/pozgaj/sazeci/05%20Tehnike%20oblikovanja.pdf, pristup: 21.2.2017.
17) Požgaj, Ž. i Bosilj Vukšić, V. (n.d.) *Use Case Dijagram*. Dostupno na: web.efzg.hr/dok/INF/pozgaj/vje%C5%BEbe/01-USE%20CASE%20DIJAGRAM-za-studente.pdf, pristup: 15.3.2017.
18) Radovanović, D. (2011.) *Projektiranje informacijskih sustava- Use Case Diagram.* Dostupno na: predmet.singidunum.ac.rs/pluginfile.php/387/mod_folder/content/0/Predavanja%202011-2012/02%20-%20Use%20case%20dijagram/02-PRIS-UseCase_Teorija.pdf?forcedownload=1, pristup: 10.5.2017.

19) Sommerwille, I. (2011.) *Software Engineering*, 9th edition, Addison- Wesley, Harlow, England
20) Stephens, R. (2015.) *Beginning Software Engineering*, John Wiley i Sons, Inc., Canada

Popis slika