Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

UML-dijagrami:
Zbirka primjera i riješenih zadataka

Manualia Universitatis studiorum Zagrabiensis
Sveučilišni priručnik

Autori:

Dr. sc. Alan Jović, dipl. ing.
Mr. sc. Marko Horvat, dipl. ing.
Dr. sc. Igor Grudenić, dipl. ing.

Zagreb, listopad 2013.

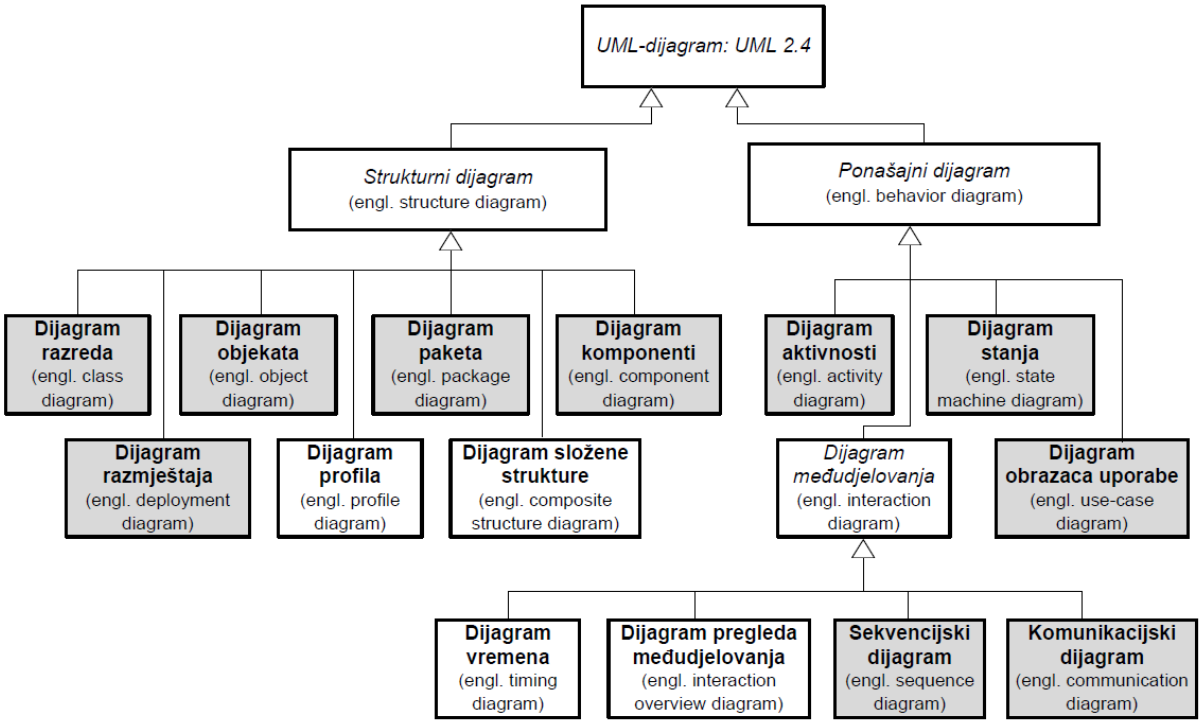
Sadržaj

1. Uvod

Ujedinjeni jezik za modeliranje (engl. *Unified Modelling Language*, kraće: UML) je normirani jezik opće namjene koji se koristi za modeliranje računalnih sustava temeljenih na objektno-orijentiranoj paradigmi. Jezik je normirala Grupa za upravljanje objektima (engl. *Object Management Group*, kraće: OMG) s prvom normom iz 1997., a s trenutačnom inačicom UML 2.4 iz 2011. godine. UML uključuje skup tehnika koje ostvaruju grafički prikaz objektno-orijentiranih računalnih sustava. Računalni sustav modelira se raznovrsnim dijagramima, od kojih se svaki koristi za prikaz sustava iz ponešto drugačije perspektive.

Dijagrami unutar UML-a mogu se podijeliti s obzirom na dinamičnost na statičke i dinamičke dijagrame. Statički dijagrami ne razmatraju vremensku komponentu sustava, već daju sliku dijelova ili cijelog sustava kakav postoji u nekom trenutku. Težnja dinamičkih dijagrama je da uključe međudjelovanje sudionika i vremensku komponentu u opis sustava kako bi se modelirali slijedovi događaja unutar sustava. Nešto suvremenija podjela dijagrama je ona koja dijeli UML-dijagrame s obzirom na to da li modeliraju strukturu sustava (engl. *structure diagram*) ili ponašanje sustava (engl. *behavior diagram*). Ovakva podjela se okvirno podudara s podjelom na statičke (strukturni) i dinamičke (ponašajni) dijagrami, s iznimkom dijagrama obrazaca uporabe koji, iako modelira ponašanje, ne modelira vremensku komponentu sustava. Podjela UML-dijagrama prema normi UML 2.4 prikazana je na slici 1.1. U okviru ove podjele postoji veći broj pojedinačnih dijagrama, od čega se u okviru ove zbirke obrađuju oni dijagrami koji su označeni sivo na slici 1.1.

Cilj ove zbirke je predočiti što više normiranih UML-dijagrama kako bi se omogućila kvalitetna komunikacija na grafičkoj razini između inženjera koji su uključeni u razvoj računalnog sustava. Pritom je potrebno ovladati znanjem o komponentama svakog opisanog dijagrama i razumjeti za što se taj dijagram koristi. U zbirci je dan naglasak na tri dijagramima koji se najčešće koriste u praksi:



Slika 1.1. Pregled UML-dijagrama, norma UML 2.4 [OMG 2011]

dijagramu obrazaca uporabe, dijagramu razreda i sekvencijskom dijagramu. Čitatelji se upoznaju s dijagramima kroz nekoliko primjera i riješenih zadataka za svaku vrstu dijagrama. Rješavanjem zadanih zadataka čitatelj može usporediti svoja rješenja s onima koja su dana u ovoj zbirci i na taj način naučiti ispravan način crtanja UML-dijagrama.

Zbirka je namijenjena poglavito studentima 3. godine preddiplomskog studija računarstva, na kolegiju Oblikovanje programske potpore (OPP), u sklopu Zavoda za elektroniku, mikroelektroniku, računalne i inteligentne sustave, Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu (FER). Namjena zbirke je da se koristi kao dodatni nastavni materijal koji će olakšati studentima savladavanje gradiva iz modeliranja računalnog sustava korištenjem jezika UML u okviru projekta iz kolegija OPP, što pokriva približno trećinu gradiva kolegija. Također, vježbanjem zadataka iz ove zbirke studenti se mogu uspješno pripremiti za ispite iz ovog kolegija. Zbirka je također namijenjena i svim ostalim zainteresiranim čitateljima koji bi željeli naučiti modelirati sustave korištenjem jezika UML.

Primjeri i zadaci unutar ove zbirke crtani su korištenjem dvaju alata dostupnih studentima FER-a, a to su Microsoft Visio 2007 i 2010 (putem licence MSDNAA) i ArgoUML v.0.32 (slobodan programski proizvod [Wulp 2011]). Izbor alata za crtanje UML-dijagrama ostavljen je na volju čitatelja budući da kod samih alata postoji velika raznolikost po pitanju licenciranja, podržanih dijagrama i mogućnosti crtanja njihovih komponenata.

Zbirka je ustrojena na sljedeći način. U poglavlju 2 opisuju se dijagrami obrazaca uporabe: njihovo značenje iz perspektive krajnjeg korisnika sustava i opis bitnih komponenti dijagrama. Poglavlje 3 posvećeno je dijagramima međudjelovanja, što znači sekvencijskim i komunikacijskim dijagramima. Opisuju se karakteristike tih dijagrama, njihove razlike i daju se razrađeni primjeri da bi se što lakše razumjelo kako modelirati međudjelovanje dijelova sustava. Dijagrami razreda detaljno su razrađeni u opsežnom poglavlju 4, zajedno sa svim entitetima koji se na tim dijagramima prikazuju. Poglavlje 5 opisuje dijagrame paketa i objekata, koji služe kao nadopuna dijagramima razreda. Poglavlje 6 posvećeno je dijagramima stanja i dijagramima aktivnosti koji modeliraju dinamičko ponašanje dijelova sustava. Poglavlje 7 uključuje dijagrame komponenti, a poglavlje 8 dijagrame razmještaja. Obje vrste dijagrama na sličan način modeliraju statičku sliku sustava iz perspektive povezanosti dijelova sustava, samo to čine na različitoj razini. U poglavlju 9 dana su rješenja zadataka iz svih ostalih poglavlja. Studentima se savjetuje da konzultiraju rješenja tek kad su sami izradili svoje rješenje zadatka na temelju primjera i uputa danih u ovoj zbirci. Literatura je navedena na kraju zbirke.

2. Dijagrami obrazaca uporabe

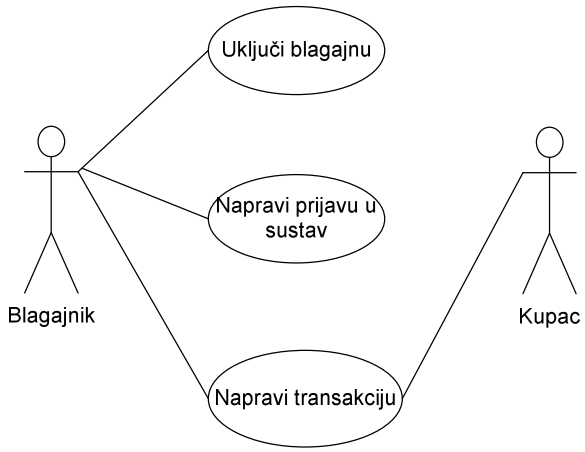
2.1. Karakteristike dijagrama

Dijagrami obrazaca uporabe (engl. *use-case diagrams*) koriste se da bi se prikazalo ponašanje sustava, dijelova sustava ili konkretnog razreda na način vidljiv korisniku sustava. Obrasci uporabe također služe tome da se razvojni tim i korisnici usaglase po pitanju ponašanja korisnika pri komunikaciji sa sustavom [Rational 2001]. Ponašanje sustava opisano je pomoću ciljeva (engl. *use-cases* – obrasci uporabe) i aktora (engl. *actors*) koji predstavljaju apstrakciju korisnika sustava, odnosno općenitije nekog od dionika (engl. stakeholder) sustava. Aktori mogu predstavljati i druge vanjske sustave koji sudjeluju u radu sustava koji se modelira. Jedan obrazac uporabe uključuje komunikaciju aktora s modeliranim sustavom koji se ostvaruje kao niz poruka među sudionicima obrasca uporabe, a koji doprinosi ostvarenju nekog jedinstvenog cilja. Na dijagramima obrazaca uporabe dodatno se definiraju odnosi između pojedinih obrazaca uporabe kao i odnosi između aktora.

Dijagrami obrazaca uporabe su statički UML-dijagrami (kao i dijagrami razreda, objekata, paketa i razmještaja), a također pripadaju i skupini ponašajnih dijagrama budući da modeliraju moguće ponašanje korisnika sustava.

Primjer 2.1.1. Blagajna u trgovačkom centru

Prikažite u dijagramu obrazaca uporabe funkcionalnost blagajne u trgovačkom centru.



Slika 2.1. Rješenje primjera 2.1.1.

U primjeru navedenom na slici 2.1 aktori u sustavu su Blagajnik i Kupac, označeni pojednostavljenim prikazom čovjeka (engl. *stickman*). Blagajnik može uključiti blagajnu, napraviti prijavu u sustav te napraviti transakciju. Te temeljne aktivnosti čine obrasce uporabe (engl. *use-case*) koji se prikazuju elipsom. Obrazac uporabe *Napravi transakciju* odnosi se na očitavanje svih artikala, ispis računa, odabir načina plaćanja i plaćanje robe. U tom obrascu uporabe sudjeluje i kupac. Obrazac uporabe *Napravi transakciju* moguće je raščlaniti na nekoliko povezanih jednostavnijih obrazaca uporabe. Razina apstrakcije na kojoj će se sustav modelirati obrascima uporabe ovisi o iteraciji procesa programskog inženjerstva, odnosno potrebnoj razini detalja u određenoj fazi izrade programske potpore.

2.2. Elementi dijagrama

Prema UML-specifikaciji dijagrami obrazaca uporabe sastoje se od aktora, obrazaca uporabe te veza i odnosa među aktorima i obrascima uporabe. Dodatno se na dijagramu obrazaca uporabe može istaknuti granica sustava koji se izgrađuje.

2.2.1. Aktori

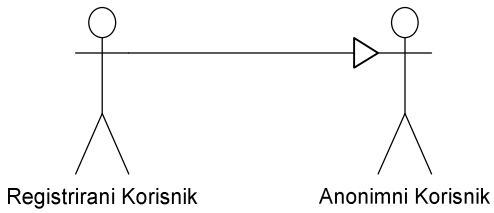
Aktori u dijagramima obrazaca uporabe predstavljaju jednu od idealiziranih uloga korisnika u sustavu. Na dijagramima obrazaca uporabe aktori su prikazani pojednostavljenim prikazom čovjeka. Jedan fizički korisnik može ostvariti više uloga u sustavu, a jednu ulogu u sustavu može preuzeti više korisnika. Aktori su sudionici pojedinih obrazaca uporabe pri čemu jedan aktor može sudjelovati u više obrazaca uporabe, a u jednom obrascu uporabe može sudjelovati više aktora. Uz to što predstavljaju uloge korisnika u sustavu, aktori mogu predstavljati određene funkcionalnosti eksternih sustava s kojima sustav koji se modelira komunicira.

Aktori komuniciraju putem niza poruka s obrascima uporabe u kojima sudjeluju. Niz poruka između aktora i obrazaca uporabe ne prikazuje se na dijagramima obrazaca uporabe, ali se može prikazati na drugim UML-dijagramima poput sekvencijskog ili komunikacijskog dijagrama. Ponekad se dinamika određene komunikacije korisnika sa sustavom opisuje neformalnim načinima poput proznog teksta.

Kod modeliranja sustava obrascima uporabe najvažnije je identificirati obrasce koji su ključni za ispravno funkcioniranje sustava. Nakon toga treba odrediti ljudske čimbenike, a dodatno i vanjske sustave koje treba prikazati u svrhu boljeg razumijevanja funkcionalnosti modeliranog sustava. Aktori međusobno mogu biti povezani vezom generalizacije pri čemu je definicija apstraktnog aktora dopunjena određenijim ulogama izvedenih aktora.

Primjer 2.2.1.1. Odredi aktore čija uloga je korištenje internet trgovine.

Kod ostvarenja većine internet trgovina korisnici se dijele u dvije veće skupine: anonimni i registrirani korisnici. Anonimnim korisnicima dozvoljene su aktivnosti poput pregledavanja kataloga proizvoda, odabira proizvoda koje planiraju kupiti, pregled odabranih proizvoda i slične funkcionalnosti koje ne zahtjevaju da identitet korisnika bude poznat. Registrirani korisnici mogu izvoditi sve aktivnosti (obrasce uporabe) kao i anonimni korisnici uz dodatak izmjene osobnih podataka, te provedbu plaćanja robe. Budući da je skup obrazaca uporabe koje može provoditi anonimni korisnik pravi podskup skupa obrazaca uporabe koje provodi registrirani korisnik moguće je definirati da je registrirani korisnik posebna vrsta anonimnog korisnika koji ostvaruje i dodatne uloge u sustavu. Rješenje primjera 2.2.1.1 prikazano je na slici 2.2.



Slika 2.2. Rješenje primjera 2.2.1.1.

Uvođenje generalizacije među aktorima u dijagramima obrazaca uporabe je korisno zato što se smanjuje broj veza između aktora i obrazaca uporabe te se dijagram čini čitljivijim.

2.2.2. Obrasci uporabe

Obrazac uporabe predstavlja jedinstvenu funkcionalnost koju prema vanjskom svijetu pruža modelirani sustav. Obrazac uporabe sastoji se od glavnog tijeka izvođenja, a može biti proširen varijacijama tog tijeka kao i posebnim slučajevima koji se mogu dogoditi za vrijeme njegovog ostvarenja. Tijek izvođenja obrasca uporabe opisan je nizom poruka koje aktori izmjenjuju s modeliranim sustavom. Spomenuti niz poruka se ne prikazuje na dijagramu obrazaca uporabe.

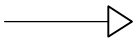
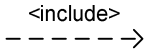
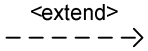
Izvršavanje svakog obrasca uporabe je neovisno od drugih obrazaca uporabe. Jedina ovisnost između obrazaca uporabe ostvaruje se preko dijeljenih podataka.

Obrasci uporabe su od velike važnosti u iterativnom procesu programskog inženjerstva. Kod iterativnog pristupa u svakoj od iteracija se odabire podskup obrazaca uporabe koje je potrebno ostvariti.

2.2.3. Veze na dijagramima obrazaca uporabe

U dijagramima obrazaca uporabe postoji nekoliko vrsta veza koje se mogu ostvariti između aktora i obrazaca uporabe. Pregled i kratki opis svih vrsta veza prikazan je u tablici 2.1.

Tablica 2.1. Grafički prikaz i opis veza na dijagramima obrazaca uporabe.

Tip veze	Grafički prikaz	Opis
asocijacija		Asocijacijom se povezuju aktori s obrascima uporabe u kojima sudjeluju. Međusobno povezivanje aktora ili međusobno povezivanje obrazaca uporabe ovom vrstom veze nije dozvoljeno. Kod asocijacije moguće je definirati njenu višestrukost (engl. *multiplicity*) čije značenje je definirano u nastavku teksta.
generalizacija		Generalizacijom se povezuju dva aktora ili dva obrasca uporabe. U slučaju da se generalizacijom povezuju dva aktora, specifičniji aktor preuzima sve uloge apstraktnijeg aktora uz dodatak novih uloga. Kod korištenja generalizacije između dva obrasca uporabe specifičniji obrazac proširuje odnosno mijenja funkcionalnosti apstraktnijeg obrasca uporabe.
uključivanje		Vezom uključivanja se povezuju dva obrasca uporabe na način da jedan obrazac u tijeku svog izvođenja u potpunosti izvede uključeni obrazac uporabe.
proširenje		Vezom proširenja se povezuju dva obrasca uporabe pri čemu jedan proširuje funkcionalnost drugog. Proširenje se ostvaruje ako je zadovoljen određeni uvjet definiran u točki proširenja.

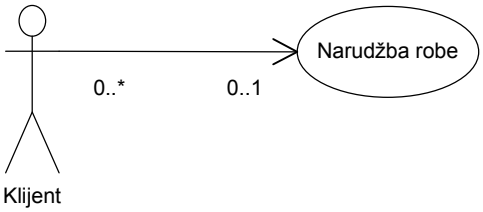
2.2.4. Asocijacija (engl. *association*)

Asocijacijom se povezuju aktori s obrascima uporabe u kojima sudjeluju. U slučaju da se želi naglasiti koji aktor inicira određeni obrazac uporabe (inicijator) to se može napraviti dodatkom strelice na vezu asocijacije, kako je to prikazano na slici 2.3.

Slika 2.3. Asocijacija između aktora i obrasca uporabe.

Naznačeno je da klijent inicira izdavanje potvrde, a službenik sudjeluje u izdavanju potvrde. Ako neki aktor samo sudjeluje u obrascu uporabe, ali sam ne inicira akciju (sudionik), tada se veza između obrasca uporabe i sudionika može označiti strelicom koja ide u sudionika. Tipičan primjer bila bi baza podataka ili datotečni sustav.

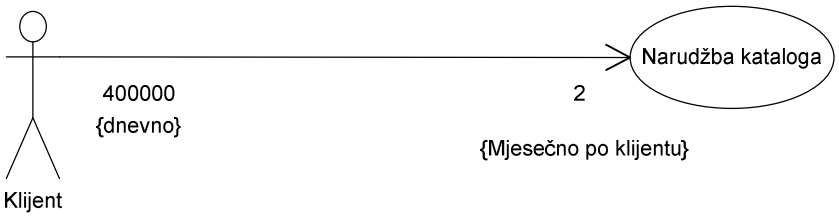
Kod asocijacije je moguće koristiti višestrukost pri čemu se određuje koliko puta neki aktor sudjeluje u određenom obrascu uporabe te koliko aktora može izvesti određeni obrazac uporabe. Na slici 2.4 prikazan je primjer narudžbe robe u internet trgovini.



Slika 2.4. Primjer višestrukosti.

Pri tome u sustavu internetske trgovine u nekom vremenskom trenutku klijent može izraditi najviše jednu narudžbu robe (0..1), dok se u istom trenutku može odvijati nula ili više (0..*) narudžbi robe različitih klijenata.

Višestrukost na vezi asocijacije se promatra u nekom vremenskom periodu. U navedenom primjeru višestrukost se promatra u jednom beskonačno kratkom trenutku vremena, ali vremenski interval koji se promatra može biti duži (primjerice tjedan dana) ili može biti vezan uz neko svojstvo sustava poput trajanje sjednice za vrijeme korištenja web sjedišta. U svakom slučaju je korisno na dijagramu napomenuti na koji se vremenski interval višestrukosti odnose, primjerice na način kako je to prikazano na slici 2.5.



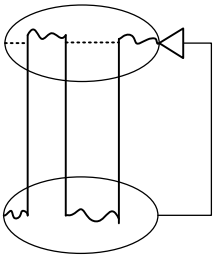
Slika 2.5. Vremenski interval pri određivanju višestrukosti.

Ovime je naznačeno da klijent može izraditi narudžbu kataloga dva puta mjesečno, a da se dnevno provede 400 000 narudžbi kataloga.

2.2.5. Generalizacija (engl. *generalization*)

Generalizacijom se povezuju dva aktora ili dva obrasca uporabe. U slučaju da se generalizacijom povezuju dva aktora, specifičniji aktor preuzima sve uloge apstraktnijeg aktora uz dodatak novih uloga. Kod korištenja generalizacije između dva obrasca uporabe, specifičniji obrazac proširuje, odnosno mijenja funkcionalnosti apstraktnijeg obrasca uporabe.

Primjer generalizacije obrazaca uporabe i tijeka njihovog izvođenja prikazan je na slici 2.6.



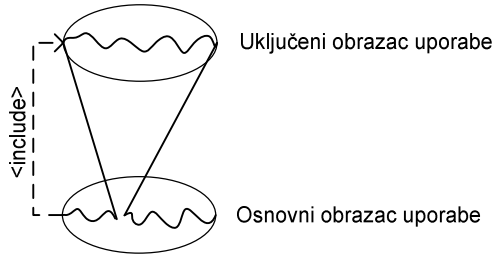
Slika 2.6. Primjer generalizacije.

Valovitom linijom označen je tijek izvođenja obrasca uporabe. Kod generalizacije specifičniji obrazac uporabe dodaje nove ili mijenja dijelove postojeće funkcionalnosti apstraktnijeg obrasca uporabe.

2.2.6. Uključivanje (engl. *include*)

Vezom uključivanja se povezuju dva obrasca uporabe na način da jedan obrazac u tijeku svog izvođenja u potpunosti izvede uključeni obrazac uporabe pri čemu trenutak izvođenja uključenog obrasca nije specificiran dijagramom.

Grafički prikaz tijeka izvođenja obrazaca povezanih vezom uključivanja prikazan je na slici 2.7.



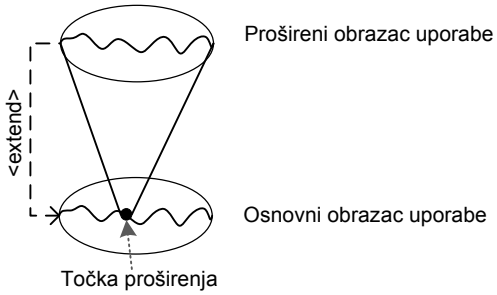
Slika 2.7. Primjer uključivanja.

U slučaju da osnovni obrazac uporabe dođe u točku izvođenja u kojoj je potrebno izvesti uključeni obrazac uporabe tada se uključeni obrazac uporabe izvodi u cijelosti. Moguće je da u tijeku izvođenja nisu zadovoljeni uvjeti pozivanja uključenog obrasca uporabe.

2.2.7. Proširenje (engl. *extend*)

Vezom proširenja se povezuju dva obrasca uporabe pri čemu jedan proširuje funkcionalnost drugog. Proširenje se ostvaruje ako je zadovoljen određeni uvjet definiran u točki proširenja. Proširenja se obično koriste kada je potrebno naznačiti da je neka funkcionalnost opcionalna u nekom obrascu uporabe. Dodatno se koriste kada je potrebno naznačiti da se neka posebna funkcionalnost ostvaruje samo kada su ostvareni određeni uvjeti, pri čemu su ti uvjeti dovoljno značajni da bi ih se naglasilo na dijagramu obrazaca uporabe.

Primjer toka izvođenja osnovnog i proširenog obrasca uporabe prikazan je slikom 2.8.

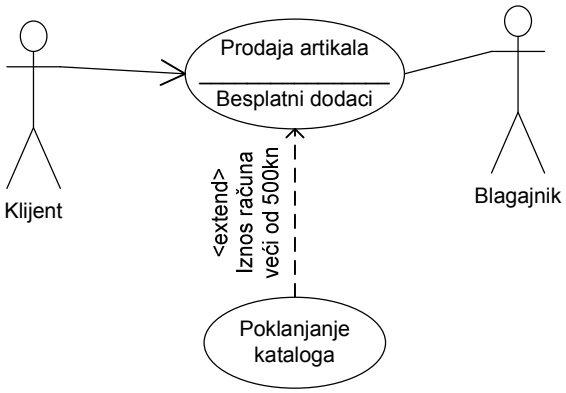


Slika 2.8. Primjer proširenja.

U točki proširenja se funkcionalnost osnovnog obrasca uporabe proširuje proširenim obrascem uporabe ako su zadovoljeni uvjeti točke proširenja. U slučaju da postoji više točaka proširenja i proširenih obrazaca uporabe redoslijed njihovog izvođenja ovisi o komunikaciji aktora s osnovnim obrascem uporabe.

Primjer korištenja veze proširenja u dijagramima obrazaca dan je kroz postupak dobivanja besplatnog kataloga uz kupnju u trgovini čiji iznos premašuje 500 kn. Pri tome se u točki proširenja *Besplatni dodaci* vrši provjera je li iznos računa dovoljan da bi se ostvarilo pravo na besplatni katalog.

Iako veze proširenja i uključivanja obje uključuju potpuno izvođenje dodatnog obrasca uporabe, pri čemu to izvođenje može biti uvjetno, one su semantički različite. Kod veze uključivanja osnovni obrazac uporabe ne može postojati bez uključenog obrasca uporabe, dok je kod veze proširenja moguće ukloniti prošireni obrazac uporabe. Dodatno se kod veze proširenja posebno naglašava točka proširenja i odgovarajući uvjeti nužni za izvođenje proširenog obrasca uporabe. Primjer korištenja veze proširenja u dijagramima obrazaca prikazan je na slici 2.9.



Slika 2.9. Primjer korištenja veze proširenja u dijagramima obrazaca.

2.2.8. Riješeni složeni primjer

Primjer 2.2.8.1. Izraditi dijagram obrazaca uporabe za projektni zadatak vezan uz trgovinu nekretninama.

Trgovci nekretninama najčešće su zatvorene, male privatne tvrtke koje posluju bez informacijske infrastrukture i bez kvalitetnog vizualnog predstavljanja klijentima. Od klijenata se očekuje da iskažu svoje zahtjeve uživo i tada im trgovac pronađe pošto-poto ono što su klijenti tražili, bez transparentnosti procesa ili utjecaja klijenata. Zadatak ovog programskog proizvoda je u povećanju transparentnosti tržišta nekretninama čime se olakšava posao i trgovcima nekretninima i klijentima.

Programskom proizvodu mogu pristupiti bilo registrirani klijenti bilo registrirani trgovci nekretninama. Registraciju provodi administrator programa na temelju osnovnih podataka korisnika (ime i prezime, OIB, adresa, telefon, e-mail). Administrator ima ovlast promjene podataka svih korisnika. Klijenti i trgovci imaju različit pogled na program. Svaki klijent ima pridruženog jednog trgovca nekretninama. Jedan trgovac nekretninama može imati više pridruženih klijenata.

Klijent na početku u programu bira trgovca nekretninama iz liste aktivnih trgovaca na temelju preporuka (od 1 do 10) koje su dali drugi klijenti i iznosa naknade koju trgovac zaračunava. Klijent može promijeniti osobne podatke o sebi, kao i dodati opis o tome kakva ga točno nekretnina zanima. Klijent ima pravo pretraživati bazu dostupnih nekretnina koju su sastavili trgovci nekretninama. Svaka nekretnina opisana je svojim podacima: nazivom, nazivom grada, nazivom kvarta, adresom, vrstom nekretnine (stan, obiteljska kuća, vila), tipom (garsonijera, 1-,2-,3-,višesobni stan; prizemnica, jedno-, dvo- tro-, višekatnica), starosti, brojem kvadrata, priključcima (struja, telefon, plin, voda), bazenom, brojem parkirnih mjesta, tekstualnim komentarom i cijenom. Svaka nekretnina ima pridruženu jednu ili više fotografija. Bazu se može pretraživati po redu kako su nekretnine dodavane, ili na temelju parametara pretrage (grad, kvart, vrsta, tip, starost, broj kvadrata), pri čemu se dobiva lista odgovarajućih nekretnina.

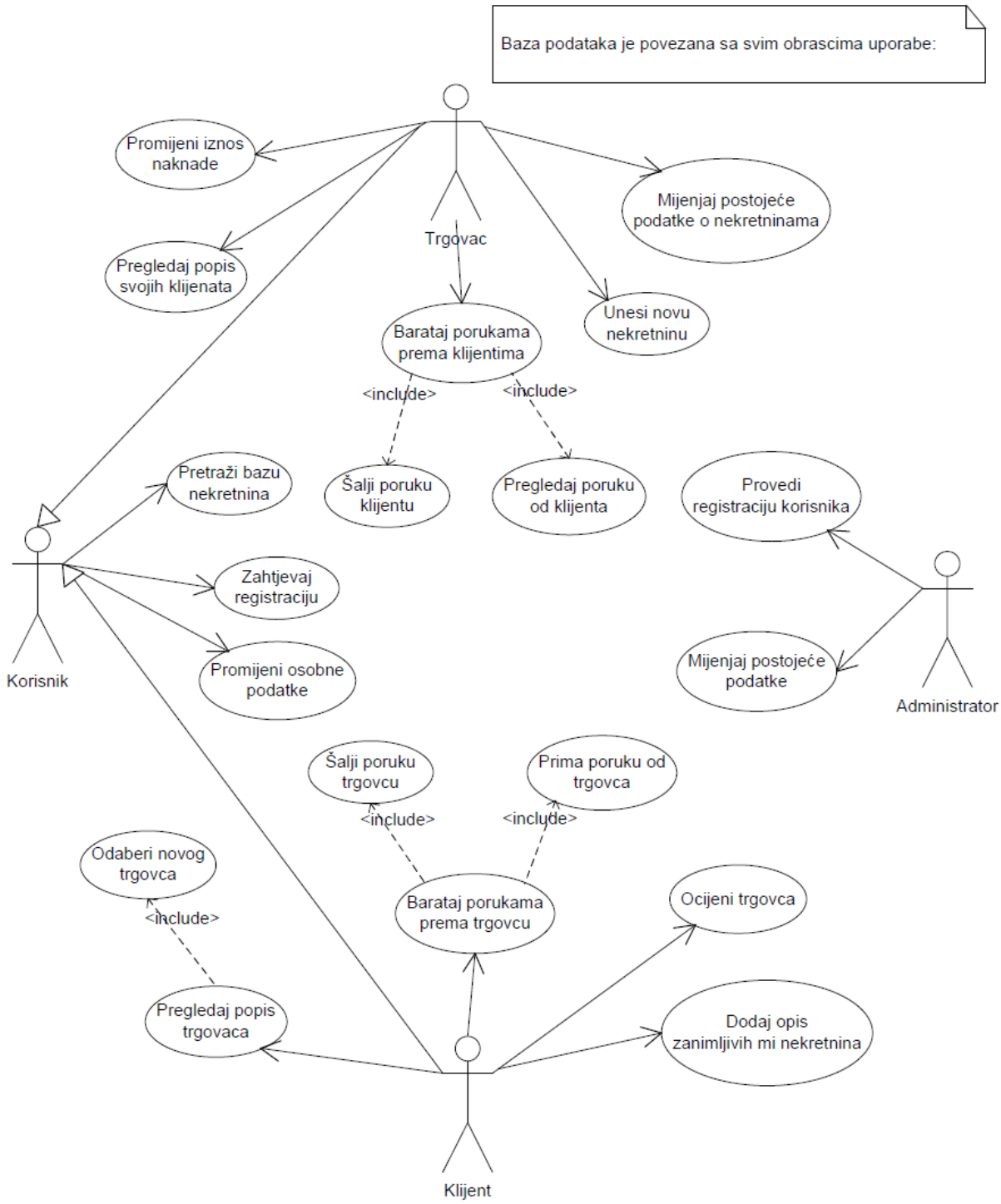
Nakon što se odluči obići nekretninu, klijent može poslati poruku sebi pridruženom trgovcu za dogovor oko vremena i mjesta sastanka. Poruka se šalje kroz program i klijent dobiva odgovor kroz program o tome odgovara li termin trgovcu ili je trgovac zauzet. U svakom trenutku, klijent ima pravo odabrati drugog pridruženog trgovca nekretninama. O svakoj promjeni statusa pridruženosti treba biti obaviješten trgovac porukom unutar programa. Također, klijent mora moći pregledati sve poruke koje je poslao trgovcima ili primio od trgovaca. Klijent može ocijeniti sebi pridruženog trgovca nekretninama ocjenom od 1 do 10 u svakom trenutku u programu, pri čemu se revidira prosječna preporuka trgovca.

Trgovac nekretninama ima sljedeće mogućnosti rada u programu: 1) Pregledati popis klijenata koji su mu pridruženi, zajedno s njihovim osnovnim podacima i opisom zahjeva; 2) Dodati novu nekretninu u bazu, pri čemu treba unijeti sve potrebne informacije o nekretnini, uključujući i fotografije. Pretpostavlja se da je podatke dobavio na terenu od prodavača. 3) Izmijeniti postojeće podatke o nekoj nekretnini; 4) Poslati poruku klijentu, bilo kao odgovor na već pristiglu poruku, bilo kao preporuku klijentu koja bi bila dobra nekretnina za njega; 5) Pregledati bazu nekretnina, slično kao i klijenti; 6) Izmijeniti iznos naknade (u postotcima od iznosa nekretnine) i 7) izmijeniti osobne podatke na svojoj početnoj stranici profila.

1. Prvi korak u rješavanju zadatka je pronalazak aktora. Aktore je moguće iščitati iz teksta zadatka. U ovom slučaju, aktori su: klijent, trgovac, administrator i baza podataka. Osim toga, može se izdvojiti i zaseban aktor, korisnik, koji je generalizacija klijenta i trgovca.

2. Da bi se lakše izradio dijagram obrazaca uporabe, potrebno je na temelju teksta zadatka pregledno pobrojati aktivnosti koje pojedini aktori koriste i koja je njihova uloga (da li su inicijatori akcije ili samo sudionici). Ovo je rezultat:
   - Korisnik, inicijator:
     - Može zahtijevati registraciju
     - Ima pristup i mogućnost promjene osobnih podataka nakon registracije
     - Pretražuje i pregledava bazu nekretnina
   - Klijent, inicijator:
     - Pregledava popis trgovaca
     - Odabire novog trgovca
     - Ocjenjuje trgovca
     - Dodaje opis kakva ga nekretnina zanima
     - Baratanje porukama prema trgovcu (primanje i slanje)
   - Trgovac, inicijator:
     - Pregledava popis svojih klijenata
     - Mijenja postojeće podatke o nekretninama
     - Unosi nove nekretnine u bazu
     - Baratanje porukama prema klijentima (primanje i slanje)
     - Mijenja iznos naknade
   - Administrator, inicijator:
     - Provodi registraciju korisnika
     - Ima pristup svim podacima i može ih mijenjati
   - Baza podataka, sudionik:
     - Sadrži arhivu podataka o nekretninama sa svim karakteristikama
     - Pohranjuje sve podatke o korisnicima sustava
     - Pohranjuje poruke korisnika

3. Na kraju se crta dijagram obrazaca uporabe koji će obuhvatiti sve navedene aktivnosti korisnika. Prema potrebi, na dijagram se ucrtavaju odnosi generalizacije, uključivanja i proširenja. Bazu podataka u ovom slučaju nije potrebno crtati budući da većina obrazaca uporabe komunicira s njom, ali je potrebno navesti da baza podataka postoji u komentaru dijagrama. Rezultat je prikazan na slici 2.10.



Slika 2.10. Dijagram obrazaca uporabe za primjer 2.2.8.1.

2.3. Zadaci za vježbu

Zadatak 2.1. Modeliranje upravljanja hotelom

Potrebno je izraditi use-case dijagram dijela sustava namijenjenog upravljanju hotelom. Administrator sustava može konfigurirati sustav što uključuje upis svih raspoloživih soba, parkirnih mjesta i cjenika. Korisnici preko interneta mogu pretražiti hotelsku ponudu i napraviti rezervaciju

sobe. Prilikom rezervacije sobe korisnici mogu opcionalno rezervirati parkirno mjesto, te dokupiti doručak, polupansion ili puni pansion.

Recepcionar hotela može izdati sobu i napraviti naplatu. Kod izdavanja sobe recepcionar obavezno provjerava rezervaciju i programira ključ (karticu) za otvaranje sobe. Prilikom naplate recepcionar izdaje račun i naznačuje da soba postaje slobodna.

Zadatak 2.2. Modeliranje prodaje autobusnih karata

Potrebno je izraditi dijagram obrazaca uporabe za prodaju karata na autobusnom kolodvoru. Sustav koriste blagajnici i putnici. Blagajnik prodaje kartu putniku. Prodaja karte uključuje odabir relacije i naplatu. Naplata se može izvršiti u gotovini ili putem kartice. U slučaju da se naplata vrši putem kartice u naplati sudjeluje putnik utipkavanjem PIN-a. Opcionalno kod prodaje karte blagajnik može izdati račun R1.

Uz prodaju karata blagajnik može napraviti i rezervaciju karte. Rezervaciju karte može napraviti i putnik samostalno preko telefonskog automata.

Zadatak 2.3. Modeliranje korištenja grozda računala

Grozd računala mogu koristiti administratori i korisnici. Administratori sustava zaslužni su za dodavanje novih korisnika, promjenu podataka korisnika, brisanje postojećih korisnika i konfiguraciju sustava. Konfiguracija sustava podrazumijeva podešavanje postavki raspoređivača poslova i izrade sigurnosne kopije podataka. Izrada sigurnosne kopije podataka je posebna vrsta kopiranja podataka. Korisnici mogu dodavati nove poslove na obradu, mogu brisati postojeće poslove, te mogu mijenjati parametre postojećih poslova. Izmjena parametra postojećih poslova uključuje brisanje postojećeg posla i stvaranje novog posla. Dodatno, korisnici mogu slati podatke na grozd računala, kopirati podatke i učitavati podatke s grozda računala.

Zadatak 2.4. Modeliranje rada *rent-a-car* kompanije

Potrebno je izraditi dijagram obrazaca uporabe sustava koji služi kao podrška radu *rent-a-car* kompanije, a u žarištu sustava je kontrola flote rent-a-car vozila i rad s kupcima. Sustav treba pratiti nabavke i rashodovanja vozila te pratiti njihovo korištenje kroz vrijeme. Korisnici koji iznajmljuju automobile mogu preko interneta napraviti i otkazati rezervaciju vozila. Kod stvaranja rezervacije korisnici moraju unijeti broj kreditne kartice. Korisnici mogu napraviti rezervaciju vozila samo u slučaju da u sustavu za zatražen vremenski period postoje slobodna vozila. Kod stvaranja rezervacije korisnici mogu prema želji i raspoloživim resursima napraviti i rezervaciju za GPS uređaj, dječju sjedalicu, krovne nosače za bicikle ili skije i prikolicu.

Kada korisnici dođu u poslovnicu *rent-a-car* kompanije u kojoj trebaju preuzeti vozilo zaposlenik provjerava postoji li rezervacija vozila za određenog korisnika. Ako rezervacija ne postoji zaposlenik *rent-a-car* kompanije izrađuje rezervaciju za korisnika.

Nakon što je rezervacija napravljena (ili je postojala od ranije) zaposlenik *rent-a-car* kompanije ispisuje obrazac o preuzimanju vozila koji korisnik potpisuje. Potpisani obrazac se skenira i unosi u računalo. Nakon što je obrazac skeniran zaposlenik *rent-a-car* kompanije odvodi klijenta do automobila i predaje mu ključeve, prometnu dozvolu i obrazac o preuzimanju vozila.

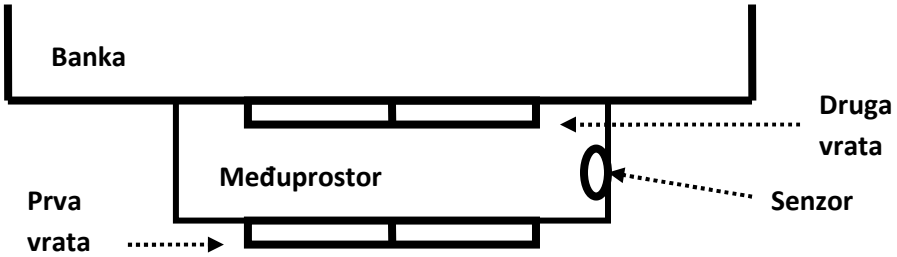
Kod povratka automobila korisnik dolazi u poslovnicu *rent-a-car* kompanije gdje predaje obrazac o preuzimanju vozila, prometnu dozvolu i ključeve automobila. Zaposlenik odlazi s korisnikom do automobila, provjerava stanje automobila i preostalu količinu goriva. Nakon povratka u poslovnicu zaposlenik generira račun i odobrava terećenje kreditne kartice korisnika.

Zadatak 2.5. Modeliranje sigurnosnog sustava banke

Neka banka ima sigurnosni sustav koji se sastoji od nadzornih kamera i dvostrukih kliznih vrata sa senzorom na ulazu. Klijent banke može ući u banku tako da mu se otvore prva vrata i on prođe kroz njih u međuprostor. Nakon toga, senzor na drugim vratima detektira klijenta i nakon što prođe 5 sekundi, senzor pokreće ulazak klijenta u banku. Time se otvaraju druga vrata i zatvaraju prva vrata. Klijent zatim prođe kroz druga vrata u banku. Klijent u banci može obaviti svoje poslovanje i potom izaći iz banke. Prilikom izlaska, otvaraju mu se najprije druga vrata, zatim klijent opet čeka 5 sekundi u međuprostoru i potom senzor pokreće izlazak klijenta. Time se otvaraju prva vrata i zatvaraju druga vrata nakon čega klijent može izaći. Senzor ne pokreće otvaranje prvih ili drugih vrata drugim klijentima dok su neki klijenti već u međuprostoru i dok traje zadanih 5 sekundi čekanja.

Službenik sigurnosti banke može kroz sigurnosni sustav u svakom trenutku zaključati prva vrata i/ili druga vrata. Ako službenik kroz sustav zaključa bilo koja vrata, to se evidentira u bazi podataka banke. Jednako tako, službenik sigurnosti može otključati bilo koja vrata, što se također evidentira u bazi. Službenik sigurnosti može provjeriti zapise nadzornih kamera iz baze podataka banke. Tijekom provjere zapisa, službenik sigurnosti može pobrisati neki zapis nadzorne kamere iz baze. U specijalnom slučaju, kada postoji neki evidentirani incident, brisanje zapisa nadzorne kamere iz baze povlači to da sustav o pokušaju brisanja obavještava policiju. Nadzorne kamere imaju samo zadatak snimanja protoka klijenata u banci i zapisivanje toga u bazu podataka banke.

Crtež uz zadatak 2.5:



2.4. Napomene

Česta pogreška prilikom modeliranja sustava dijagramima obrazaca uporabe odnosi se na stvaranje obrasca uporabe kojeg svi drugi obrasci uključuju. Tipičan primjer takve pogreške je uključivanje obrasca uporabe, *Prijava na sustav* u sve druge obrasce uporabe, zato jer je nužno ostvariti prijavu na sustav prije nego se izvode ostali obrasci uporabe. Takav pristup nije dobar budući da podrazumijeva da će se obrazac uporabe *Prijava na sustav* izvesti u sklopu svakog pojedinačnog obrasca uporabe koji ga uključuje, odnosno da će se korisnik stalno morati prijavljivati na sustav u sklopu svake aktivnosti.

Takva pogreška rezultat je krive percepcije izražajne vrijednosti dijagrama obrazaca uporabe. Dijagrami obrazaca uporabe ne koriste se da bi se ostvario sustav preduvjeta izvođenja određenih obrazaca uporabe (uz iznimku točaka proširenja). Napomene o preduvjetima ostvarenja pojedinog obrasca uporabe treba navesti u detaljnom opisu tog obrasca u formalnoj ili neformalnoj formi koja nije dio dijagrama obrazaca uporabe.

Često se pri modeliranju griješi i sa pretjeranom generalizacijom obrazaca uporabe. Kod takvog pristupa može se dogoditi da se napravi jedan generalni obrazac uporabe tipa *Transakcija* i onda

puno specifičnih obrazaca koji bi trebali predstavljati posebne vrste te transakcije. Prije korištenja generalizacije potrebno je identificirati koja ponašanja općenitog obrasca uporabe je nužno promijeniti ili dodati u specifičnim obrascima uporabe.

Neki detalji ostvarenja sustava koji se modelira su suvišni na dijagramu obrazaca uporabe. Iako aktori mogu biti vanjski sustavi ili čak dijelovi sustava koji se modelira, njihovo korištenje treba biti ograničeno. Primjerice, ako sustav koji se modelira sadrži bazu podataka i svi ili većina obrazaca korištenja komunicira s tom bazom podataka, tada se baza podataka može ukloniti s dijagrama obrazaca, jer u suprotnom ona djeluje kao centar i kao najvažnija stavka na dijagramu obrazaca korištenja.

3. Sekvencijski i komunikacijski dijagrami

3.1. Karakteristike dijagrama

Sekvencijski dijagrami (engl. *sequence diagram*) i komunikacijski dijagrami (engl. *communication diagram*) pripadaju široj skupini UML-dijagrama međudjelovanja (engl. *interaction diagram*). Obje vrste dijagrama spadaju u nadskupinu ponašajnih dijagrama (engl. *behavioral diagram*), zajedno s dijagramima stanja, aktivnosti i obrazaca uporabe. Stari naziv za komunikacijski dijagram je kolaboracijski dijagram.

Za razliku od dijagrama obrazaca uporabe, kod kojih vremenska komponenta nije važna, sekvencijski i komunikacijski dijagrami daju naglasak na vremenskom redoslijedu kojim se odvija međudjelovanje sudionika u sustavu, tako da ih se svrstava i u dinamičke UML-dijagrame.

Sudionici koji se modeliraju na sekvencijskim i komunikacijskim dijagramima mogu biti aktori (predočavanjem komunikacije aktora s dijagrama obrazaca uporabe) ili objekti, pri čemu se prikazuje komunikacija instanci razreda prikazanih na dijagramu razreda. Dok se prilikom komunikacije aktora govori o porukama, kod objekata to su pozivi postupaka u objektima drugih razreda.

Razlika između sekvencijskog i komunikacijskog dijagrama s informacijskog stajališta nema. Bilo koji od ova dva dijagrama može se koristiti za prikaz međudjelovanja sudionika i jednog je moguće jednoznačno pretvoriti u drugi [Quatrani 2002]. Ipak, manje razlike između te dvije vrste dijagrama postoje u samom prikazu i one su sažete u tablici 3.1.

Tablica 3.1. Razlike između sekvencijskog i komunikacijskog dijagrama.

Sekvencijski	**Komunikacijski**
Veći naglasak na vremenskoj uređenosti scenarija	Veći naglasak na pregledu scenarija, odnosno na međusobnoj komunikaciji
Redoslijed poruka sudionika odozgora lijevo prema dolje desno	Redoslijed poruka sudionika određen je brojčanim oznakama koje se stavljaju na poruke
Koristi se u ranim fazama specifikacije i analize projekta	Koriste se u fazi dizajniranja implementacije odnosa
Raširenији i čitljiviji	Sažetiji i teži za čitati
Teže izmjene	Lakše izmjene

3.2. Sekvencijski dijagrami

3.2.1. Riješeni primjeri

Primjer 3.2.1.1. Otvaranje računa u banci

Potrebno je modelirati otvaranje računa klijenta u banci sekvencijskim dijagramom. Klijent najprije zatraži otvaranje novog računa. Osobnom bankaru je za akciju otvaranja računa potreban identifikacijski dokument klijenta. Također, za tu akciju treba naplatiti iznos od 50 kuna. Osobni bankar zato najprije zatraži od klijenta identifikacijski dokument i 50 kuna. U slučaju da klijent nema dokument ili 50 kuna, međudjelovanje se obustavlja i klijent odlazi. Inače, klijent predaje dokument i novac, a osobni bankar otvara račun u bazi podataka. Čim je račun otvoren, osobni bankar to potvrđuje klijentu i time je međudjelovanje završeno. Pretpostavite da će otvaranje računa proći bez grešaka.

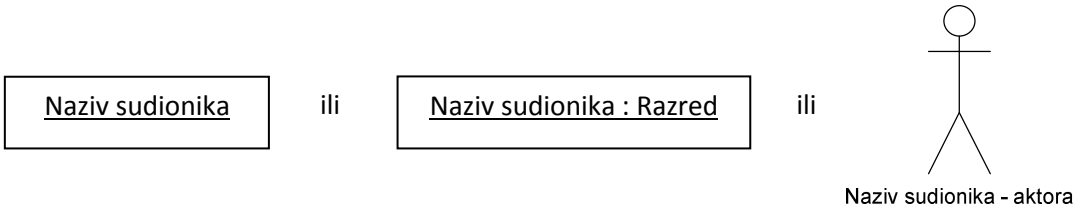
Postupak rješavanja

1. Pronađite sudionike u međudjelovanju:

Modelirati otvaranje računa klijenta u banci sekvencijskim dijagramom. Klijent najprije zatraži otvaranje novog računa. Osobnom bankaru je za akciju otvaranja računa potreban identifikacijski dokument klijenta. Također za tu akciju treba naplatiti iznos od 50 kuna. Osobni bankar zato najprije zatraži od klijenta identifikacijski dokument i 50 kuna. U slučaju da klijent nema dokument ili 50 kuna, međudjelovanje se obustavlja i klijent odlazi. Inače, klijent predaje dokument i novac, osobni bankar otvara račun u bazi podataka i potvrđuje klijentu da je račun otvoren. Pretpostavite da će otvaranje računa proći bez grešaka i da klijent nema već otvoren račun u banci.

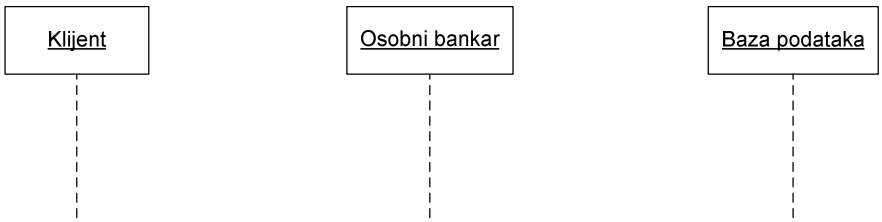
U ovom slučaju, aktori su: klijent, osobni bankar i baza podataka. Pritom su osobni bankar i klijent aktivni aktori, a baza podataka pasivan aktor (vidjeti dijagrame obrazaca uporabe).

2. Nacrtajte sudionike u redoslijedu od lijeva prema desno kako se pojavljuju u tekstu zadatka. Za crtanje sudionika koristite pravokutnik s nazivom sudionika unutar njega. Ako postoji nejasnoća po pitanju pripadnosti razredu, može se navesti i naziv razreda objekta, odvojen od objekta znakom ":". U posebnom slučaju, ako su sudionici aktori, mogu se crtati i u obliku čovječuljka (engl. *stickman*) s nazivom ispod ili iznad. Primjer sudionika prikazan je na slici 3.1.



Slika 3.1. Primjeri sudionika.

Naziv sudionika obično je u računalnim alatima potcrtan ili je ispred njega stavljen znak "/" ili ":". Pretpostavlja se da je naziv sudionika jednak nazivu razreda čiji objekt je taj sudionik. Linija koja se proteže okomito ispod sudionika naziva se životna linija (engl. *lifeline*) sudionika i može biti puna ili iscrtkana. Životna linija označava boravak sudionika u međudjelovanju unutar sustava koji se modelira. Sudionici sa životnim linijama iz primjera 3.2.1.1 prikazani su na slici 3.2.



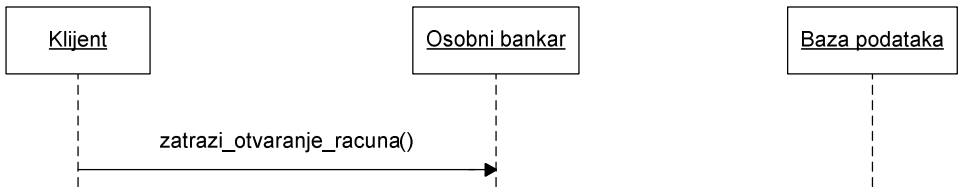
Slika 3.2. Sudionici sa životnim linijama iz primjera 3.2.1.1.

3. Analizirajte rečenice po redu, uvijek tražeći prvu sljedeću poruku koja se pojavljuje između neka dva sudionika.

„Klijent najprije zatraži otvaranje novog računa.“

Ovdje se pojavljuje poruka između klijenta i osobnog bankara (iako nije eksplicitno naveden, ali se podrazumijeva), kojim klijent traži otvaranje novog računa. Sa životne linije klijenta šalje se poruka prema osobnom bankaru koja sadrži samo naziv i praznu listu argumenata: zatrazi_otvaranje_racuna().

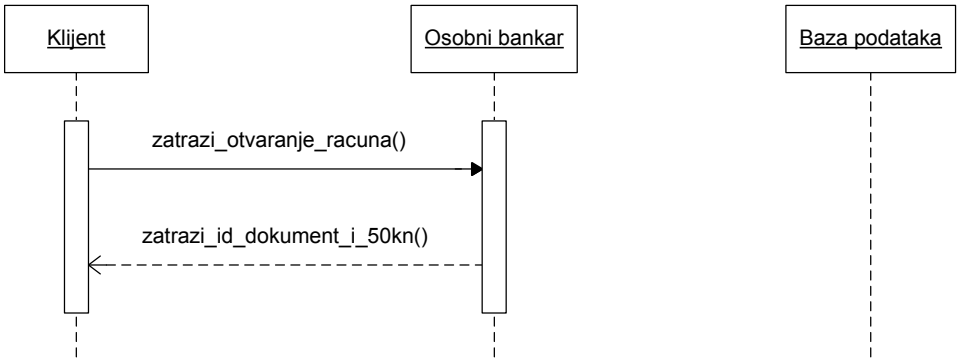
Poruka (ili poziv) između klijenta i bankara je sinkrona (engl. *synchronous message* ili engl. *synchronous call*). To znači da će klijent čekati na odgovor osobnog bankara i da za to vrijeme neće ništa drugo raditi unutar sustava. Iako takvo ponašanje klijenta nije eksplicitno navedeno u tekstu zadatka, sinkrona poruka se pretpostavlja. Gdje god nije jasno navedeno koji tip poruke se očekuje, pretpostavlja se sinkrona poruka. Sinkrona poruka označava se s punom strelicom na vrhu u svim verzijama UML-a. Traženje otvaranja računa prikazano je na slici 3.3.



Slika 3.3. Traženje otvaranja računa.

„Osobnom bankaru je za akciju otvaranja računa potreban identifikacijski dokument klijenta. Također za tu akciju treba naplatiti iznos od 50 kuna. Osobni bankar zato najprije zatraži od klijenta identifikacijski dokument i 50 kuna."

Osobni bankar treba na zahtjev klijenta odgovoriti s upitom za identifikacijskim dokumentom i uplatom u iznosu 50 kuna.

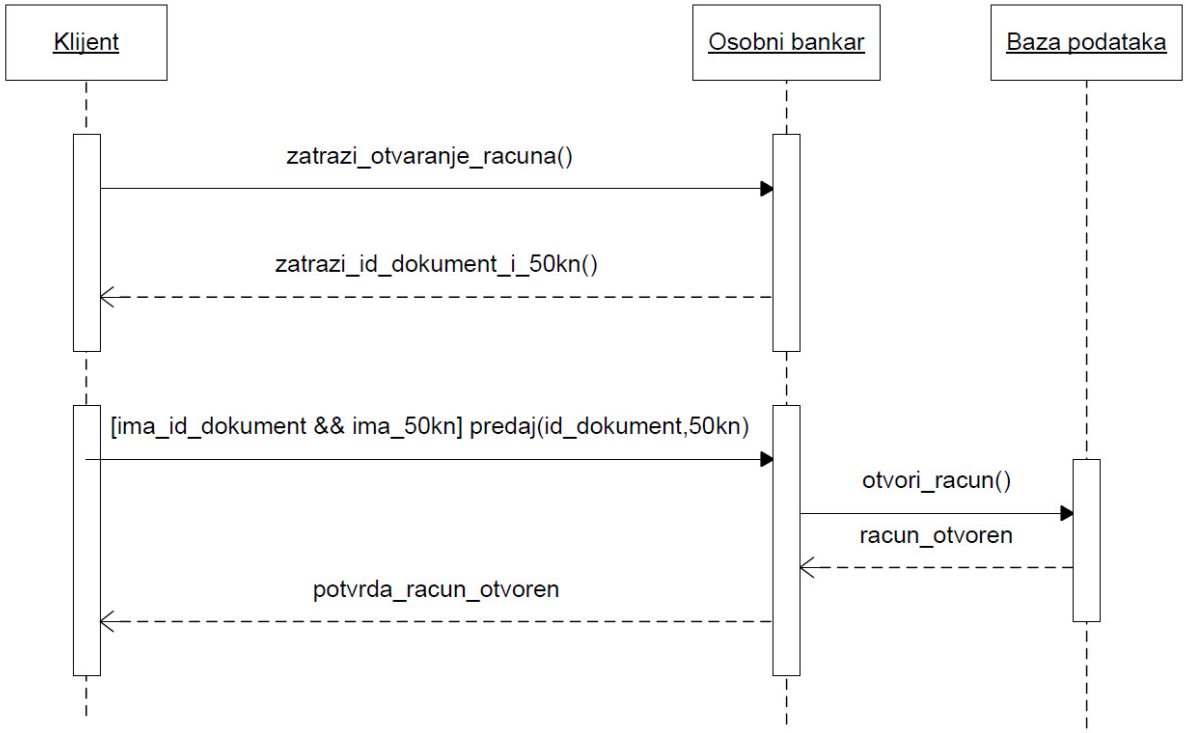


Slika 3.4. Povratna poruka osobnog bankara.

Zato se šalje povratna poruka (engl. *return message* ili *return call*). Povratna poruka sadrži naziv i može, ali i ne mora imati navedenu listu argumenata. Povratna poruka je također poruka i može pokrenuti reakciju kod aktora koji ju je primio ovisno o svojem sadržaju. Povratna poruka crta se iscrtkanom linijom s običnom strelicom na vrhu. Primijetite da sad životne linije klijenta i osobnog bankara sadrže i okomiti pravokutnik koji se naziva aktivacija (engl. *activation*), a koristi se za bolji vizualni prikaz komunikacije između procedura dvaju sudionika. Aktivacije se mogu, ali i ne moraju crtati na sekvencijskim dijagramima. Posebno su korisne ako se želi prikazati komunikacija između više razina procedura (poziv procedura unutar procedura) kao i za prikaz rekurzije. Povratna poruka osobnog bankara prikazan je na slici 3.4.

„U slučaju da klijent nema dokument ili 50 kuna, međudjelovanje se obustavlja i klijent odlazi. Inače, klijent predaje dokument i novac, a osobni bankar otvara račun u bazi podataka. Čim je račun otvoren osobni bankar to potvrđuje klijentu i time je međudjelovanje završeno.“

Ovdje je navedeno da postoje uvjeti na izvođenje daljnje komunikacije sudionika. Uvjeti su da klijent ima identifikacijski dokument i da ima 50 kuna. U tom slučaju osobni bankar mu smije otvoriti račun u bazi podataka banke i potvrđuje mu da je operacija uspješno provedena.



Slika 3.5. Konačno rješenje primjera 3.2.1.1.

Dva uvjeta navode se unutar uglatih zagrada prije naziva poruke i povezana su s „&&“ što označava logički „i“. Primijetite da su argumenti poruke identifikacijski dokument i 50 kuna. Konačno rješenje primjera 3.2.1.1 prikazano je na slici 3.5.

Primjer 3.2.1.2. Sustav za prešanje piljevine

Modelirajte rad sustava za prešanje piljevine sekvencijskim dijagramom. Zadaća plavog robota je prenošenje kutije od skladišta do crvenog robota. Crveni robot ih najprije raspakira i potom sadržaj (piljevinu) prazni u kontejner koji služi za skupljanje piljevine. Kad je kontejner preko 80% pun, na njemu se pali signalna lampica koja signalizira nadzorniku da pokrene prešu kojom će dobiti niskokvalitetnu ivericu. Nadzornik najprije privremeno zaustavlja rad plavog i crvenog robota dok kontejner ne bude spreman za novo punjenje. Pretpostavite da dok se zaustavlja plavi robot da se može pokrenuti zaustavljanje crvenog. Nakon što se oba zaustave, nadzornik pokreće prešu. Nakon što preša završi s poslom, nadzornik pokreće logiku u kontejneru kojom se aktivira interni postupak pražnjenja i čišćenja kontejnera. Kad se taj postupak završi, nadzornik pokreće crveni i plavi robot.

Pretpostavite da je za punjenje kontejnera do 80% svaki put potreban neodređen broj kutija iz skladišta kao i da uvijek ima kutija na skladištu. Pretpostavka je također da crveni robot brže

raspakira i isprazni piljevinu nego što plavom robotu treba da mu donese novu kutiju tako da je crveni robot uvijek spreman. Nije potrebno modelirati moguće kvarove u sustavu.

Postupak rješavanja

1. Pronađite sudionike u međudjelovanju:

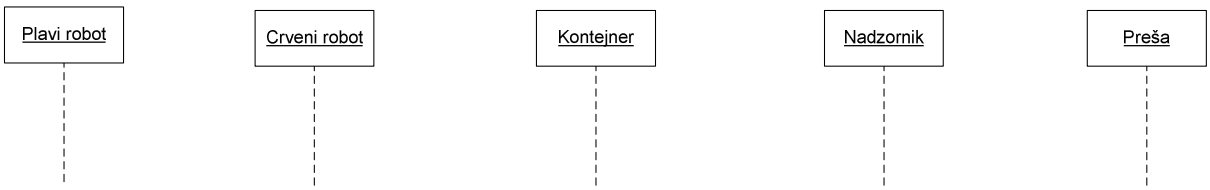
Modelirajte rad sustava za prešanje piljevine sekvencijskim dijagramom. Zadaća plavog robota je prenošenje kutije od skladišta do crvenog robota. Crveni robot ih najprije raspakira i potom sadržaj (piljevinu) prazni u kontejner koji služi za skupljanje piljevine. Kad je kontejner preko 80% pun, na njemu se pali signalna lampica koja signalizira nadzorniku da pokrene prešu kojom će dobiti niskokvalitetnu ivericu. Nadzornik najprije privremeno zaustavlja rad plavog i crvenog robota dok kontejner ne bude spreman za novo punjenje. Pretpostavite da dok se zaustavlja plavi robot da se može pokrenuti zaustavljanje crvenog. Nakon što se oba zaustave, nadzornik pokreće prešu. Nakon što preša završi s poslom, nadzornik pokreće logiku u kontejneru kojom se aktivira interni postupak pražnjenja i čišćenja kontejnera. Kad se taj postupak završi, nadzornik pokreće crveni i plavi robot.

Pretpostavite da je za punjenje kontejnera do 80% svaki put potreban neodređen broj kutija iz skladišta kao i to da uvijek ima kutija na skladištu. Pretpostavka je također da crveni robot brže raspakira i isprazni piljevinu nego što plavom robotu treba da mu donese novu kutiju tako da je crveni robot uvijek spreman. Nije potrebno modelirati moguće kvarove u sustavu.

Zašto skladište nije zasebni sudionik? Moglo bi biti, međutim jedino gdje se u tekstu pojavljuje to je na mjestu gdje plavi robot iz njega vadi kutije i nosi crvenom robotu. Time će prenašanje kutije iz skladišta biti modelirano kao poruka.

Zašto signalna lampica nije zasebni sudionik? Zato što se signalna lampica nalazi na kontejneru i služi samo za signalizaciju. Zato će se modelirati kao poruka.

2. Nacrtajte sudionike u redoslijedu s lijeva na desno kako se pojavljuju u tekstu zadatka. Sudionici iz primjera 3.2.1.2 prikazani su na slici 3.6.

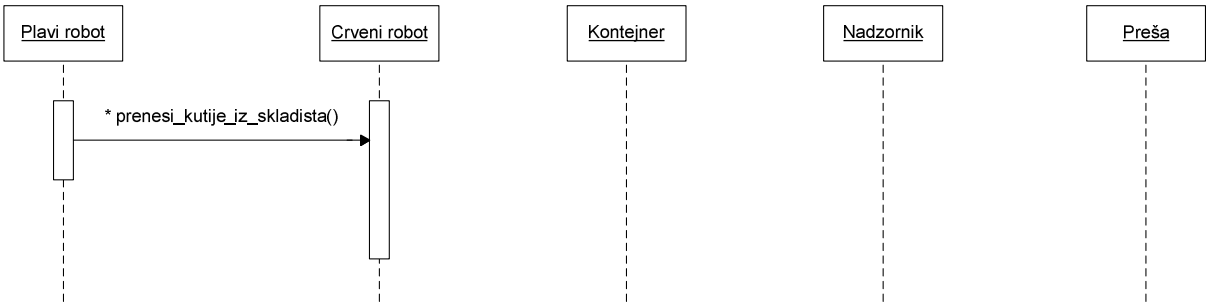


Slika 3.6. Sudionici iz primjera 3.2.1.2.

3. Analizirajte rečenice po redu, uvijek tražeći prvu sljedeću poruku koja se pojavljuje između neka dva sudionika.

„Zadaća plavog robota je da prenosi kutije od skladišta do crvenog robota.“

Očito, kutije se prenose u petlji bez nekog ograničenja, jedna ili više odjednom. Prenosi ih plavi robot iz skladišta do crvenog robota, kako je to prikazano na slici 3.7. Ne spominje se da plavi robot radi išta drugo za to vrijeme, tako da je poruka sinkrona.

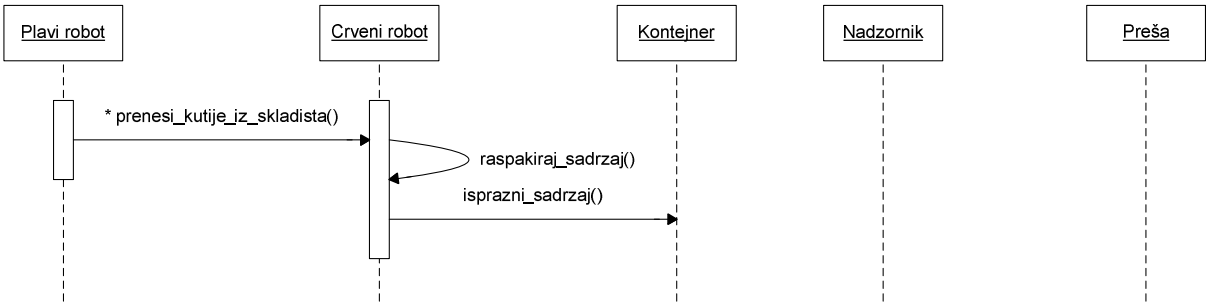


Slika 3.7. Plavi robot prenosi kutije iz skladišta crvenom robotu – primjer petlje.

Petlja se označava znakom „*“. Uvjet na petlju navodi se unutar uglatih zagrada, kao što je već prikazano u primjeru 3.2.1.1. za slučaj poruke koja se nije izvodila u petlji. Povratna poruka u ovom je slučaju implicitna, budući da crveni robot automatski dojavi plavom da je sve u redu čim primi kutije. Tada se plavi robot vraća u skladište po nove.

„Crveni robot ih najprije raspakira i potom sadržaj (piljevinu) prazni u kontejner koji služi za skupljanje piljevine.“

Raspakiravanje sadržaja je interni postupak (ili interna procedura) koji izvodi crveni robot prije operacije pražnjenja sadržaja u kontejner. Svaki interni postupak označava se tako da se poruka šalje samom sebi pri čemu se samo navede poziv poruke. Pritom se mogu koristiti dodatne aktivacije za proceduru, ali i ne moraju. Poziv internog postupka za raspakiravanje sadržaja kutija prikazan je na slici 3.8.

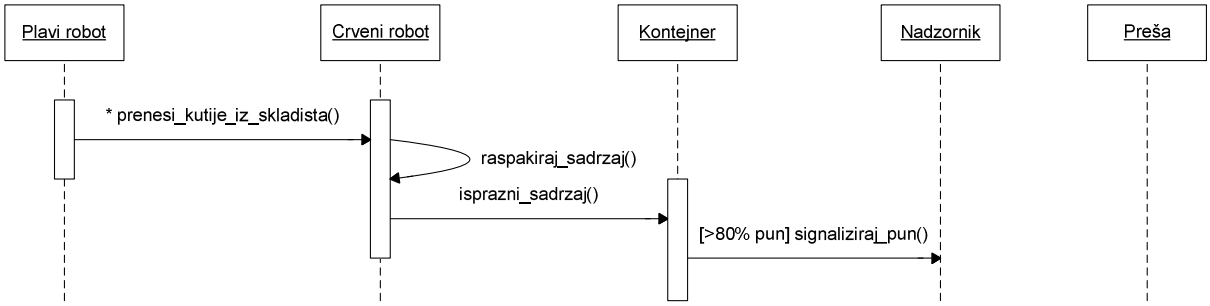


Slika 3.8. Poziv internog postupka za raspakiravanje sadržaja kutija.

Poruka isprazni_sadrzaj() je sinkrona zato što na kraju primjera piše da crveni robot uvijek spremno čeka (tj. ništa drugo ne radi) dok ponovno ne dođe plavi robot. Povratna poruka je također implicitna.

„Kad je kontejner preko 80% pun, na njemu se pali signalna lampica koja signalizira nadzorniku da pokrene prešu kojom će dobiti niskokvalitetnu ivericu.“

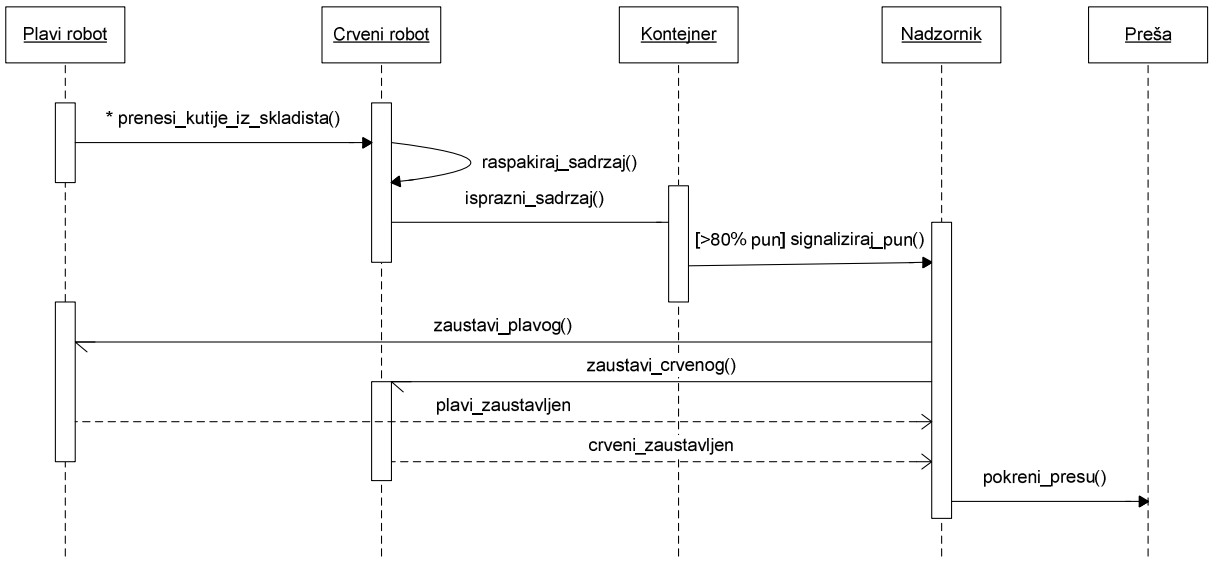
Postavlja se uvjet na sadržaj kontejnera. Kad je preko 80% pun, signalizirat će nadzorniku da je sve spremno za prešanje. Na slici 3.9 prikazan je uvjet na popunjenje kontejnera i signalna poruka nadzorniku.



Slika 3.9. Uvjet na popunjenje kontejnera i signalna poruka nadzorniku.

„Nadzornik najprije privremeno zaustavlja rad plavog i crvenog robota dok kontejner ne bude spreman za novo punjenje. Pretpostavite da dok se zaustavlja plavi robot da se može pokrenuti zaustavljanje crvenog. Nakon što se oba zaustave, nadzornik pokreće prešu."

Nadzornik izvodi privremeno zaustavljanje radne snage. To se zbog uvjeta zadatka ostvaruje asinkronim porukama (engl. *asynchronous message* ili *asynchronous call*). Kad je vidio da su oba robota zaustavljena pokreće prešu koja treba sprešati piljevinu.

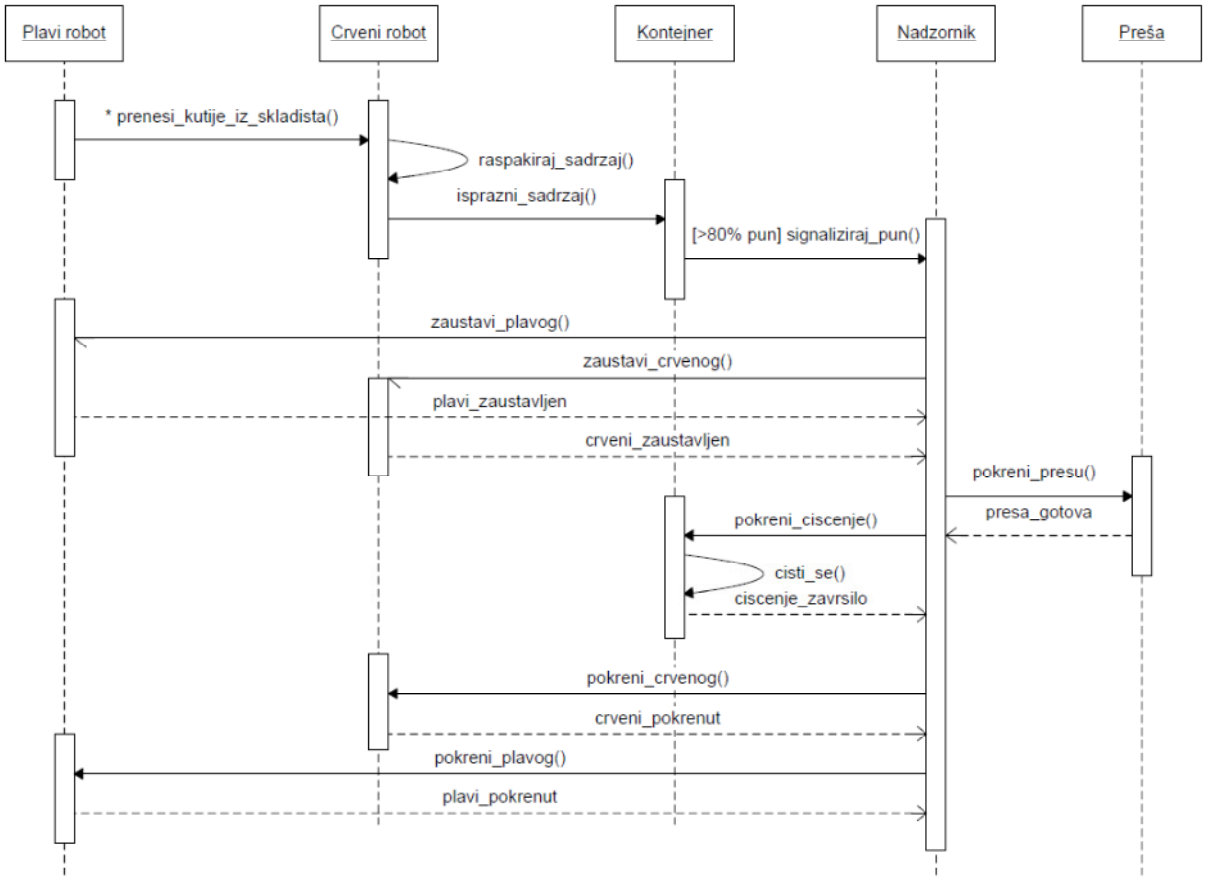


Slika 3.10. Primjer asinkronih poruka koje se koriste za zaustavljanje rada robota.

Asinkrone poruke označavaju se sa strelicom samo na jednoj strani poruke. Drugi načini prikaza su s običnom strelicom (pri čemu nije jasno radi li se o bilo kojoj poruci ili baš o asinkronoj) ili alternativno, sa strelicom u obliku trokuta, samo praznom iznutra (ista strelica kao i generalizacija kod dijagrama razreda). Primjer asinkronih poruka koja se koriste za zaustavljanje rada robota prikazan je na slici 3.10.

„Nakon što preša završi s poslom, nadzornik pokreće logiku u kontejneru kojom se aktivira interni postupak pražnjenja i čišćenja kontejnera. Kad se taj postupak završi, nadzornik pokreće crveni i plavi robot."

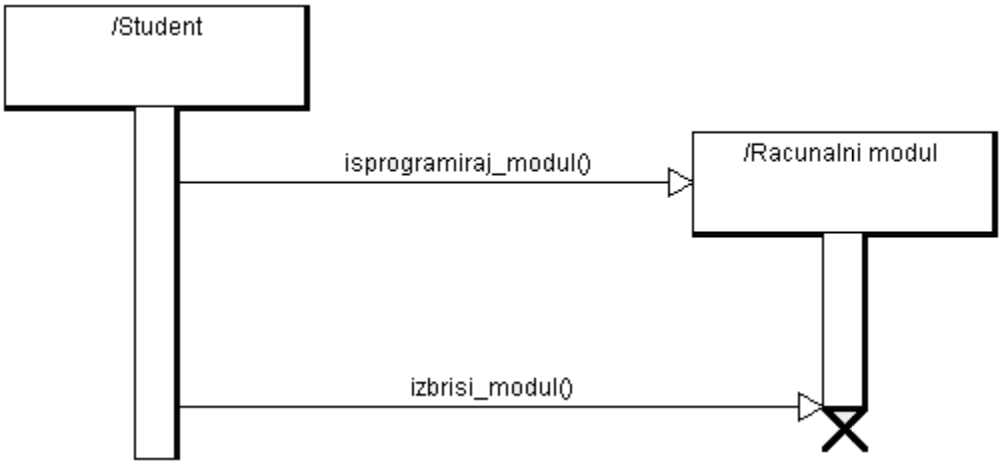
Nadzornik pokreće interni postupak čišćenja kontejnera tek nakon što preša završi s radom. Kad se taj postupak završi, nadzornik pokreće robote (budući da nije navedeno ništa drugo, pretpostavlja se sinkrono pokretanje). Konačno rješenje primjera 3.2.1.2 prikazano je na slici 3.11.



Slika 3.11. Konačno rješenje primjera 3.2.1.2.

3.2.2. Napomene u vezi sekvencijskih dijagrama

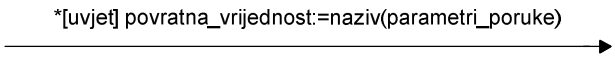
1. Tijekom života nekog sudionika moguće je stvoriti ili uništiti nekog drugog sudionika u sustavu ako se tako traži u zadatku. Stvaranje se ostvaruje jednostavnom porukom usmjerenom prema novom sudioniku (pravokutniku ili čovječuljku). To znači da se taj sudionik ne crta odmah na početku, već ga se dodaje u dijagram kasnije. Uništavanje se provodi također porukom pri čemu se na kraju životne



Slika 3.12. Primjer stvaranja i brisanja sudionika u sustavu.

linije uništavanog sudionika nacrta znak „X“ čime je označeno da se linija prekida i da sudionik prestaje biti modeliran u sustavu. Primjer stvaranja i brisanja sudionika u sustavu prikazan je na slici 3.12.

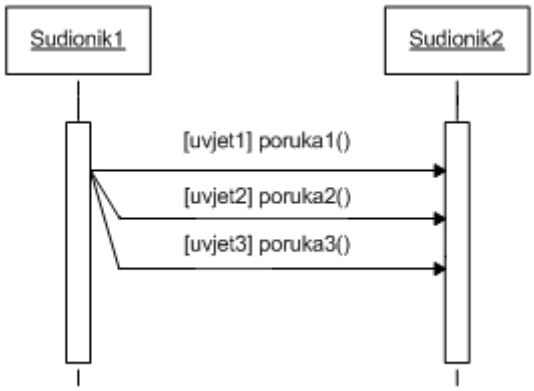
2. Opći oblik poruke (sinkrone ili asinkrone) u sekvencijskom dijagramu prikazan je na slici 3.13.



Slika 3.13. Opći oblik poruke u sekvencijskom dijagramu.

Pritom je jedino naziv poruke obavezan, dok su svi ostali elementi poruke opcionalni. Parametri poruke odvajaju se zarezom, a mogu sadržavati ili samo naziv varijable, ili oblik „varijabla : tip“, kao npr. naziv_korisnika : String.

3. Uvjetno grananje (if - else if - else) moguće je prikazati kao na slici 3.14.



Slika 3.14. Uvjetno grananje podržano u sekvencijskom dijagramu.

Neki alati za crtanje sekvencijskih dijagrama ne podržavaju ovakav način prikaza grananja. U tom slučaju poruke se crtaju jedna ispod druge bez zajedničke izvorišne točke.

4. Mogući su i drugačiji načini prikaza uvjetnih grananja i petlji. Zainteresiranog čitatelja upućujemo da pogleda web stranicu [Bell 2004] na kojoj je detaljno objašnjeno kako je moguće na drugačiji način prikazati ove složenije elemente u sekvencijskim dijagramima.

3.2.3. Zadaci za vježbu

Zadatak 3.1. Implementacija sustava internetske knjižare

Modelirajte sustav internetske knjižare sekvencijskim dijagramom. Kod implementacije sustava internetske knjižare postoje razredi cKorisnik, cAutorizacija, cKatalog, cKosarica, cSkladiste, cBanka. Prilikom kupovine, objekt razreda cKorisnik poziva postupak autoriziraj(k_ime,zaporka) razreda cAutorizacija i pri tome šalje svoje korisničko ime i zaporku. Ako je autorizacija dobro prošla, korisnik može pretraživati katalog postupkom pretrazi(parametri_pretrage) razreda cKatalog kojem šalje parametre pretrage, a kao rezultat dobiva listu knjiga koje je moguće kupiti. Da bi kupio knjigu

korisnik je prvo mora dodati u košaricu pozivom postupka dodaj(id_knjige) razreda cKosarica kojem šalje identifikator knjige koju želi kupiti. Razred cKosarica unutar postupka dodaj() postupkom provjeri_stanje(id_knjige) razreda cSkladiste, kojoj se šalje identifikator knjige, provjerava raspoloživost knjige na skladištu. Ako je knjiga raspoloživa, dodaje se u košaricu. Ciklus pretraživanja kataloga može se ponoviti više puta, koliko god korisnik želi.

Kupnja se završava korisničkim pozivom postupka kupi(br_kartice) razreda cKosarica čiji parametar je broj kreditne kartice korisnika. Nakon poziva tog postupka, razred cKosarica za svaku knjigu zove postupak azuriraj(id_knjige) razreda cSkladiste kojem šalje identifikator knjige, te na kraju poziva postupak naplati(br_kartice,iznos) razreda cBanka, kojem šalje broj kreditne kartice i ukupni iznos koji je potrebno naplatiti.

Zadatak 3.2. Komunikacija klijent-poslužitelj

Modelirajte sekvencijskim dijagramom jednostavnu komunikaciju između klijenta i poslužitelja. Poslužitelj najprije pokreće interni postupak pokreni(), kojim inicijalizira svoje resurse. Poslužitelj zatim čeka na dolazak zahtjeva od klijenta. Klijent najprije šalje poslužitelju zahtjev za odobravanjem sjednice zahtjev_za_sjednicom(). Pretpostavite da je zahtjev sinkron.

Ako poslužitelj ima dostupnih resursa, vraća odgovor klijentu da je sjednica odobrena i vraća mu identifikacijski broj sjednice (id), inače vraća odgovor da sjednica nije dostupna. U slučaju da je sjednica odobrena, klijent šalje više datoteka na obradu. Slanje datoteka treba se odvijati u petlji dokle god ima datoteka. Svaka datoteka sastoji se od naziva (String), sadrzaja (Object) i imena_ korisnika (String).

U slučaju da je bilo koji sastavni dio poruke jednak *null*, tada poslužitelj šalje asinkronu poruku o grešci. Poslužitelj prima ostale datoteke i dalje bez obzira da li je bilo slanja poruke o grešci. Nakon što je primio sve datoteke, poslužitelj asinkrono javlja klijentu da je u tijeku obrada datoteka te pokreće interni postupak obrade pristiglih datoteka. Interni postupak odvija se onoliko puta koliko ima primljenih datoteka bez grešaka.

Na kraju, poslužitelj šalje asinkronu poruku da su sve datoteke obrađene. Klijent tada prekida svoju sjednicu slanjem sinkrone poruke prekini_sjednicu(id) na što mu poslužitelj odgovara s porukom o potvrdi prekida sjednice.

Zadatak 3.3. Slanje članka na recenziju

Modelirati sekvencijskim dijagramom postupak slanja članka na recenziju u znanstveni časopis. Prvi autor odlučio je poslati članak u znanstveni časopis. Najprije autor pošalje članak uredništvu i čeka potvrdu od uredništva o ispravnosti. Uredništvo obavi internu kontrolu pri čemu se provjerava da li je oblik i tema članka u skladu s pravilima časopisa. Ako je članak ispravan, šalje se povratni odgovor da je članak ispravan i da se kreće s recenzijom. Ako članak nije ispravan, daljnji proces recenzije se obustavlja i članak se odbija. Zatim, uredništvo šalje jednu kopiju članka prvom recenzentu. Ne čekajući na potvrdu, druga kopija članka šalje se drugom recenzentu. Ne čekajući na potvrdu, treća kopija rada šalje se trećem recenzentu.

Recenzenti odgovore u roku od 14 dana mogu li recenzirati članak. Ako manje od dvoje recenzenata može recenzirati, tada uredništvo šalje primjerke članka novim recenzentima (i dalje označenima kao prvi, drugi i treći recenzent), što se ponavlja sve dok se ne nađu najmanje dvoje onih koji su voljni recenzirati. Recenzenti koji su odgovorili potvrdno provode recenziju, što se smatra internim postupkom koji radi recenzent. Za recenziju imaju 30 dana. Nakon što završi s recenzijom, recenzent šalje svoju recenziju uredništvu.

Recenzija treba sadržavati opisnik s: 1) poljem int [], u kojem pišu ocjene za svaku komponentu članka, 2) String s opisom rada i mišljenjem. Uredništvo zaprimi sve recenzije. Zatim iterativno provjerava recenzije, tako da za svaku recenziju i za svaku komponentu recenzije provjeri da li je u nekom slučaju jednaka 1, što znači da članak ne zadovoljava. Ako postoji neka komponenta članka koja ne zadovoljava, uredništvo šalje poruku prvom autoru da se članak odbija. Ako nema komponente koja ne zadovoljava, tada uredništvo šalje poruku prvom autoru da se članak prihvaća.

Zadatak 3.4. Postupak deblokade tvrtke

Modelirajte sekvencijskim dijagramom postupak deblokade i plaćanja dugova tvrtke. Tvrtka je zbog dugovanja prema banci u blokadi, što znači da ne može slobodno raspolagati imovinom i da ne može isplaćivati novac sa svog bankovnog računa (ali ga smije primati). Tvrtka duguje novac i drugim tvrtkama, a ne samo banci. Da bi se tvrtku deblokiralo, ona najprije mora vratiti sva dugovanja banci.

Pretpostavite da u deblokadi izravno sudjeluju troje subjekata: šef tvrtke, računovodstvo tvrtke i banka kod koje tvrtka ima podignut kredit i otvoren račun. Neka tvrtka povremeno prima novac od klijenata potreban za isplatu svog duga. Najprije računovodstvo tvrtke javi šefu da je stigao novac u iznosu od *N* kuna (float) i da tvrtka ukupno raspolaže s *K* kuna (float). Ako je ukupna količina novca kojom tvrtka raspolaže veća od iznosa potrebnog za isplatu svih dugova (iznos koji šef zna) tada šef odluči isplatiti sve dugove. Inače, ako novca još nema dovoljno, šef čeka dok se ne sakupi dovoljno novca.

Kad je odlučio isplatiti sve dugove, šef najprije nazove banku i najavi isplatu duga banci. Zatim šef šalje nalog računovodstvu da isplati dugove. Računovodstvo isplati banci dug. Banka po isplati deblokira račun tvrtke (interni postupak banke). Po završetku deblokade, šef odlučuje ostatkom novca isplatiti ostale dugove. Za svaku tvrtku kojoj još duguje novac, šef šalje nalog računovodstvu da joj se isplati dug. Računovodstvo putem deblokiranog bankovnog računa isplaćuje dugovanja dotičnim tvrtkama (te transakcije se provode preko banke).

Pretpostavite da se sve transakcije provode putem jednog računa tvrtke. Na kraju računovodstvo obavještava šefa da je sve plaćeno.

Zadatak 3.5. Sigurnosni sustav banke

Modelirajte kretanje klijenta u banci pomoću sekvencijskog dijagrama. Klijent najprije prolazi kroz prva vrata u međuprostor prilikom čega aktivira senzor. Nakon što prođe 5 sekundi kako je klijent u međuprostoru, senzor šalje istovremeno dvije poruke sigurnosnom sustavu. Prvu da se zatvore prva vrata i drugu da se otvore druga vrata. U slučaju da klijent ostane u međuprostoru (predomislio se i želi izaći), senzor opet čeka 5 sekundi i šalje istovremeno dvije poruke sigurnosnom sustavu. Prvu, da se zatvore druga vrata i drugu, da se otvore prva vrata. Ciklus provjere senzora da li je klijent u međuprostoru i slanja dviju poruka se ponavlja sve dok klijent ne izađe iz banke ili prođe kroz druga vrata u banku. Ako klijent izađe iz banke, završava se sekvencijski dijagram, a ako uđe u banku, tada klijent komunicira s nekim od bankovnih službenika.

Klijent može prema potrebi komunicirati s bankovnim službenikom na pultu ili s osobnim bankarom. Nakon što je završio komunikaciju, klijent izlazi iz banke prolazeći najprije kroz druga vrata u međuprostor. Nakon što prođe 5 sekundi, senzor šalje istovremeno dvije poruke sigurnosnom sustavu. Prvu, da se zatvore druga vrata i drugu, da se otvore prva vrata. Nakon toga klijent može izaći iz banke ili se vratiti nazad u banku ponovnim čekanjem u međuprostoru. Napomena: uz ovaj zadatak ide isti crtež kao i uz zadatak 2.5.

3.3. Komunikacijski dijagrami

3.3.1. Riješeni primjeri

Primjer 3.3.1.1. Kupovina zrakoplovnih karata putem Interneta

Modelirajte međudjelovanje klijenta i sustava pri kupovini zrakoplovnih karata putem Interneta komunikacijskim dijagramom. Klijent može rezervirati zrakoplovnu kartu na internetskom portalu zrakoplovnog prijevoznika. Pritom klijent najprije poziva podsustav za odabir parametara leta. Klijent određuje odredište, datum polaska i broj karata. Podsustav mu dojavljuje ima li dostupnih karata. Ako ima, klijent se prosljeđuje podsustavu za detaljan odabir leta. Klijent određuje vrstu karte (ekonomska ili poslovna) i željeni zrakoplov na dan polaska. Pretpostavite da klijent rezervira kartu samo u jednom smjeru tako da ne određuje povratni termin. Nakon odabira tih parametara, zahtjev klijenta se prosljeđuje podsustavu za naplatu. Klijent treba podsustavu za naplatu definirati ime i prezime, broj putovnice, e-mail i način plaćanja. Ako definira da je naplata putem bankovnog računa, klijent treba upisati broj bankovne kartice. Podsustav za naplatu treba na kraju zatražiti od klijenta potvrdu kupovine karte. Ako klijent potvrdi naplatu i ako je upisao broj bankovne kartice, podsustav za naplatu treba u internom postupku skinuti novac s bankovnog računa klijenta. Na kraju podsustav za naplatu šalje e-mail klijentu sa zrakoplovnom kartom.

Postupak rješavanja

1. Pronađite sudionike u međudjelovanju:

Modelirajte međudjelovanje pri kupovini zrakoplovnih karata putem Interneta komunikacijskim dijagramom. Klijent može rezervirati zrakoplovnu kartu na internetskom portalu zrakoplovnog prijevoznika. Pritom klijent najprije poziva podsustav za odabir parametara leta. Klijent određuje odredište, datum polaska i broj karata. Podsustav mu dojavljuje ima li dostupnih karata. Ako ima, klijent se prosljeđuje podsustavu za detalje leta. Klijent određuje vrstu karte (ekonomska ili poslovna) i željeni zrakoplov na dan polaska. Pretpostavite da klijent rezervira kartu samo u jednom smjeru tako da ne određuje povratni termin. Nakon odabira tih parametara, zahtjev klijenta se prosljeđuje podsustavu za naplatu. Klijent treba podsustavu za naplatu definirati ime i prezime, broj putovnice, e-mail i broj bankovnog računa. Podsustav za naplatu treba zatražiti od klijenta potvrdu kupovine karte. Ako klijent potvrdi naplatu i ako je upisao broj bankovne kartice, podsustav za naplatu treba u internom postupku skinuti novac s bankovnog računa klijenta. Ako je sve uspješno provedeno, na kraju podsustav za naplatu šalje e-mail klijentu sa zrakoplovnom kartom.

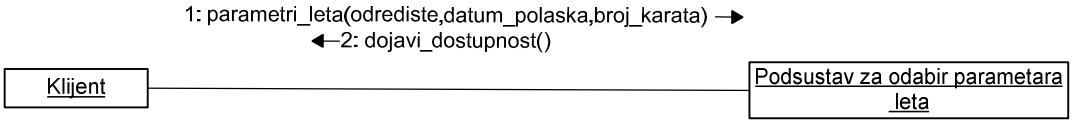
2. Za razliku od sekvencijskih dijagrama, kod komunikacijskih dijagrama nije jednostavno unaprijed posložiti sudionike da bi prikaz bio pregledan. Zato se treba pristupiti postepenom crtanju, počevši od komunikacije prva dva sudionika u opisu zadatka i postepeno dodavati sljedeće kako bi se očuvala preglednost dijagrama.

Ako su sudionici aktori s dijagrama obrazaca uporabe, za prikaz se može koristiti čovječuljak kao i kod sekvencijskog dijagrama. Ipak, najčešće se koristi prikaz sudionika unutar pravokutnika.

3. Analizirajte rečenice po redu.

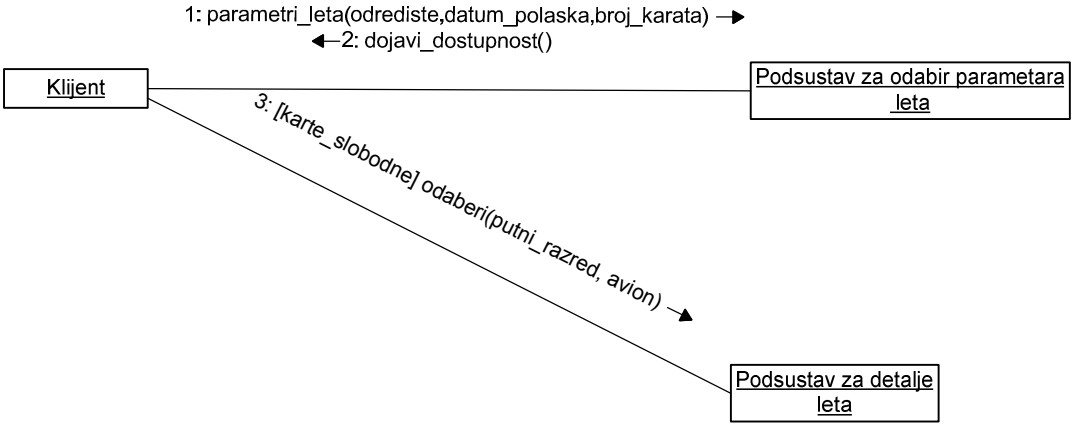
„Klijent može rezervirati zrakoplovnu kartu na internetskom portalu zrakoplovnog prijevoznika. Pritom klijent najprije poziva podsustav za odabir parametara leta. Klijent određuje odredište, datum polaska i broj karata. Podsustav mu dojavljuje ima li dostupnih karata.“

Prva komunikacija u ovom primjeru je ona između klijenta i podsustava za odabir parametara leta. Klijent djeluje tako da u sustav unese parametre odredišta, datuma polaska i broj zrakoplovnih karata. Na to mu sustav odgovara s informacijom ima li dostupnih karata. Međudjelovanje klijenta i podsustava za odabir parametara leta prikazan je na slici 3.15.



Slika 3.15. Međudjelovanje klijenta i podsustava za odabir parametara leta.

„Ako ima karata, klijent se prosljeđuje podsustavu za detalje leta. Klijent određuje vrstu karte (ekonomska ili poslovna) i željeni zrakoplov na dan polaska. Pretpostavite da klijent rezervira kartu samo u jednom smjeru tako da ne određuje povratni termin.“

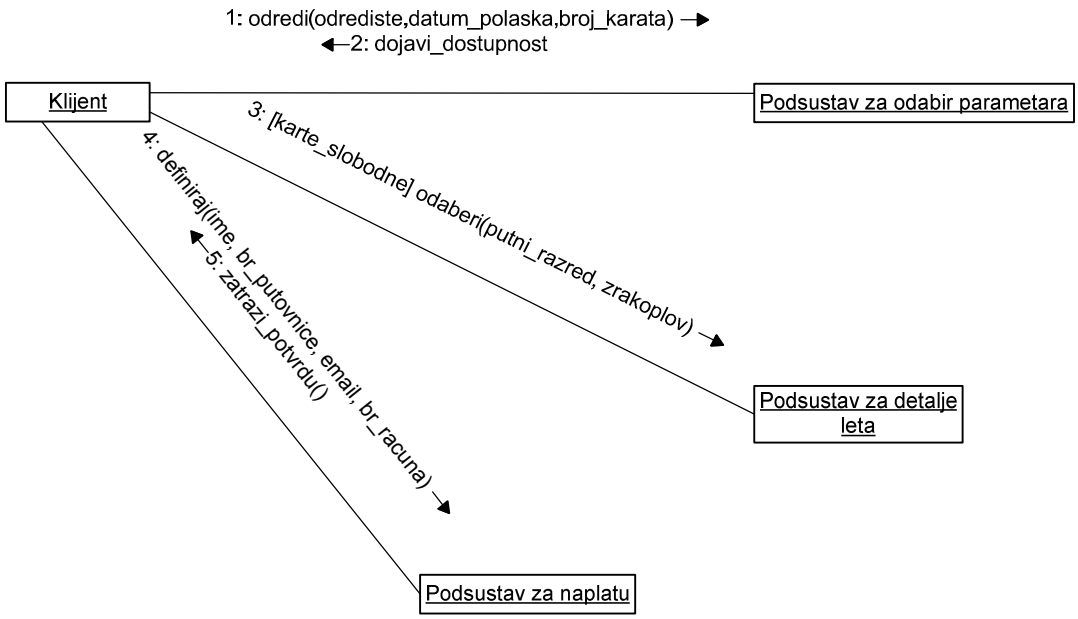


Slika 3.16. Komunikacija klijenta s podsustavom za detalje leta.

Dijagram se širi kao što je prikazano na slici 3.16. Potrebno je primijetiti da se komunikacija odvija samo u slučaju kad postoji dovoljan broj slobodnih karata. Sintaksa za ograničenje je ista kao i kod sekvencijskog dijagrama.

„Nakon odabira tih parametara, zahtjev klijenta se prosljeđuje podsustavu za naplatu. Klijent treba podsustavu za naplatu definirati ime i prezime, broj putovnice, e-mail i broj bankovnog računa. Podsustav za naplatu treba zatražiti od klijenta potvrdu kupovine karte. “

Na slici 3.17. prikazana je komunikacija između klijenta i podsustava za naplatu. Pritom klijent najprije definira sve potrebne informacije sustavu, a sustav od njega zatraži da potvrdi kupnju zrakoplovne karte.



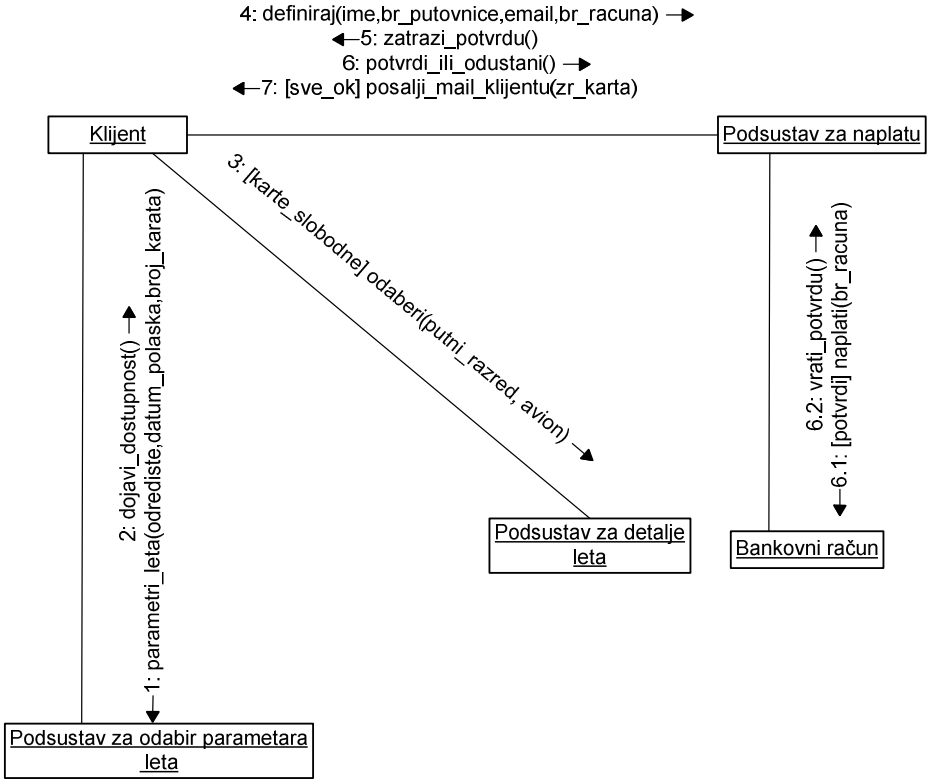
Slika 3.17. Komunikacija između klijenta i sustava za naplatu.

„Ako klijent potvrdi naplatu i ako je upisao broj bankovne kartice, podsustav za naplatu treba u internom postupku skinuti novac s bankovnog računa klijenta. Ako je sve uspješno provedeno, na kraju podsustav za naplatu šalje e-mail klijentu sa zrakoplovnom kartom. “

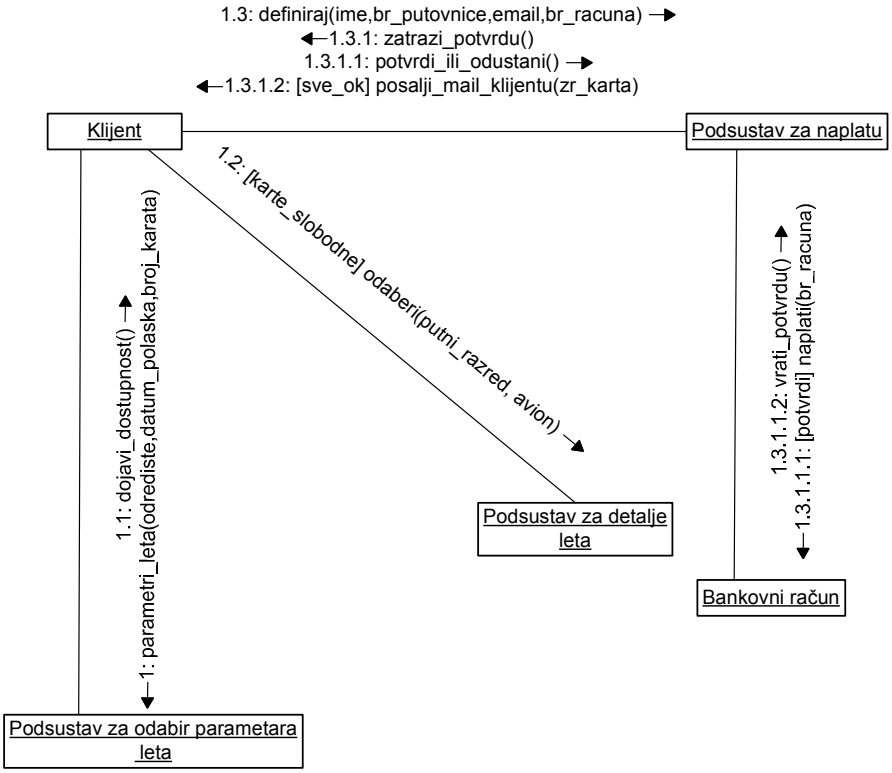
Ovdje je prisutan interni postupak unutar podsustava za naplatu koji se aktivira u slučaju ako klijent potvrdi kupnju karata. U ovakvim slučajevima najbolje je koristiti ulančano obrojčavanje poruke.

Ulančano obrojčavanje se koristi zato da se istakne da je neki postupak unutarnji dio nekog vanjskog postupka. Ulančano obrojčavanje prikazano je u konačnom rješenju zadatka, na slici 3.18.

Neki autori više vole koristiti ulančano obrojčavanje na čitavom komunikacijskom dijagramu, pri čemu se sve daljnje poruke koje se događaju nakon prve mogu promatrati kao unutarnji postupci aktivirani prvom porukom. Primjer kako bi izgledao komunikacijski dijagram primjera 3.3.1.1 u tom slučaju, prikazan je na slici 3.19. Studentima se preporuča da ulančano obrojčavanje koriste samo u slučajevima jasne aktivacije nekog postupka unutar nekog drugog postupka da bi se ublažio stupanj ulančavanja, odnosno da bi se spriječilo da poruke počinju sa zbunjujućim brojem kao što je npr. 1.3.1.1.2.



Slika 3.18. Konačno rješenje primjera 3.3.1.1.



Slika 3.19. Konačno rješenje primjera 3.3.1.1, uz jaču primjenu ulančanog obrojčavanja.

3.3.2. Napomene u vezi komunikacijskih dijagrama

Za stvaranje i uništavanje sudionika u komunikacijskim dijagramima mogu se koristiti sljedeće ključne riječi: <<create>> i <<destroy>> kao nazivi poruka.

3.3.3. Zadaci za vježbu

Zadatak 3.6. Plaćanje u dućanu

Prikazati postupak plaćanja proizvoda u dućanu komunikacijskim dijagramom. Kupac dolazi na blagajnu dućana s kupljenom robom. Kupac predaje proizvod po proizvod prodavaču koji očitava cijenu proizvoda što se pohranjuje u računalo. Računalo svaki puta potvrdi učitanje proizvoda. Postupak traje sve dok ima proizvoda u košarici. Nakon što je sva roba očitana, prodavač pita kupca ima li karticu za popust od 10% pri kupnji. Ako kupac ima karticu, tada predaje tu karticu na uvid prodavaču, koji upisuje podatke s kartice u računalo i time ostvaruje popust kupcu. Zatim prodavač pita kupca želi li plaćati gotovinom ili kreditnom karticom. U slučaju plaćanja gotovinom, kupac ostvaruje daljni popust od 5% što prodavač zapisuje u računalo.

Računalo pokazuje na ekranu kupcu kolika je konačna cijena. U slučaju da je to sve, kupac potvrđuje kupnju prodavaču koji pokreće naplatu na računalu. Računalo zatim ispisuje račun prodavaču i pohranjuje podatke o kupnji u zapis na lokalnom poslužitelju. Podaci o kupnji uključuju naziv kupca (*String*, ako postoji), listu kupljenih artikala (*List*) i konačnu cijenu (*float*). Na kraju, prodavač uručuje račun kupcu.

Zadatak 3.7. Sustav za prešanje piljevine

Riješiti primjer 3.2.1.2 (Sustav za prešanje piljevine) komunikacijskim dijagramom.

4. Dijagrami razreda

4.1. Karakteristike razreda

Dijagrami razreda (engl. *class diagrams*) opisuju razrede i njihove međusobne veze. Jednako tako, dijagrami razreda opisuju vrste objekata unutar nekog sustava i njihove međusobne statične odnose.

Dijagrami razreda pripadaju strukturnoj skupini UML-dijagrama (engl. *structure diagram*). Oni ne opisuju događaje, stanja, aktivnosti ili bilo kakvu vremenski promjenjivu karakteristiku sustava koji se modelira. Naprotiv, dijagrami razreda su statični s obzirom na vremensku komponentu.

Razred ili klasa (engl. *class*) je osnovni tvorbeni element UML-dijagrama razreda. Stoga su drugi nazivi za dijagrame razreda dijagram klasa, ili *class dijagram*. Za ispravno razumijevanje definicije razreda važno je prvo odrediti značenje objekta. Objekt predstavlja entitet iz stvarnog svijeta ili neki koncept, odnosno apstrakciju nečega što ima dobro definirane granice i smisao u sustavu. Stoga je razred opis grupe objekata sa sličnim svojstvima, a svaki objekt je obvezno pojedinac (instanca) jedne klase.

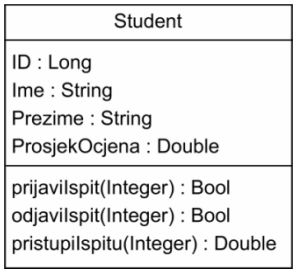
Za svaki razred nužno je definirati naziv, a moguće je odrediti popis atributa i operacija. Makar atribute i operacije nije nužno definirati, bez njih razred nema implementacijsku svrhu. Za atribute nužno je odrediti njihov naziv i tip podataka, a za operacije njihovu definiciju koja uključuje naziv operacije, te sve ulazne i izlazne parametre.

Primjer 4.1.1. Definirati razred Student

Svaki student ima svoj identifikacijski broj (ID, podatkovnog tipa Long), ime (String), prezime (String) i prosjek ocjena (Double). Student može prijaviti ispit, odjaviti ispit i pristupiti ispitu. Za prijavu i odjavu ispita potrebna je šifra predmeta (Integer), a operacije vraćaju logičke (boolean) vrijednosti da li je izvršenje uspjelo ili ne. Pristupanje ispitu je definirano drugačije: ulazni argument operacije je šifra predmeta (Integer), a povratna vrijednost (Double) je dobivena ocjena na ispitu. Ako je manja ili jednaka 1 smatra se da je student pao na ispitu, ili -1 ako je došlo do greške u radu sustava.

Rješenje:

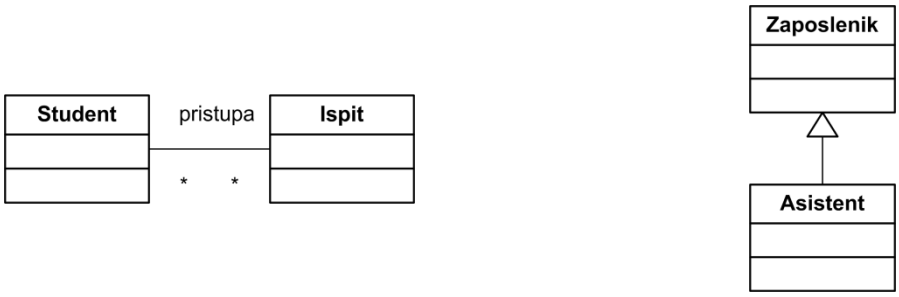
Tekst zadatka jasno definira koji atributi se traže u definiciji razreda i koji su njihovi tipovi. Nazivi atributa i operacija moraju biti jasno razumljivi i sažeti. Dodatno, u odabiru naziva operacija dobro je slijediti konvencije. Potrebno je definirati tri operacije koje imaju jedan ulazni parametar (cjelobrojna vrijednost). Prve dvije operacije (prijaviIspit i odjaviIspit) vraćaju vrijednost tipa *Bool*, dok treća operacija (pristupiIspitu) vraća vrijednost tipa *Double*. Rješenje primjera 4.1.1 prikazano je na slici 4.1.



Slika 4.1. Rješenje primjera 4.1.1.

4.2. Odnosi između razreda

Dva osnovna tipa odnosa između razreda su pridruživanje (veza ili asocijacija) i podtip. Odnosi između razreda: pridruživanje i podtip prikazani su na slici 4.2.



Slika 4.2. Odnosi između razreda: pridruživanje i podtip.

„Student pristupa ispitu“ je jedan primjer veze, dok je „Asistent je zaposlenik fakulteta“ primjer podtipa. Pridruživanje može biti jednostruko, višestruko i refleksivno. Agregacija i kompozicija su vrste pridruživanja. Odnos podtip određen je mehanizmom nasljeđivanja u međusobno suprotnim smjerovima generalizacije i specijalizacije. Osim navedenog, UML-dijagrami razreda mogu prikazati atribute i operacije razreda, njihova svojstva i ograničenja nad njima, pakete, ovisnost, tipove podataka, obrojčavanje (enumeraciju), komentare i više drugih strukturnih svojstava sustava koji su opisani u sljedećim potpoglavljima.

4.3. Pridruživanje

Pridruživanje ili veza (engl. *association*) opisuje statične odnose između dva pojedinca, odnosno instance, razreda. Veze dijelimo na jednosmjerne, dvosmjerne, agregacije i refleksivne. Tip agregacije uključuje i kompoziciju.

Ako je vrh neke asocijacije označen strelicom, onda je definiran njezin smjer (engl. *navigability*). Veze ovisno o njihovom smjeru pridruživanja dijele se na unidirekcionalne i bidirekcionalne. Unidirekcionalne veze (jednosmjerne) imaju smjer definiran samo na jednom vrhu, dok bidirekcionalne (dvosmjerne) imaju smjer definiran na oba vrha. Ako smjer nije eksplicitno definiran, smatra se da je veza nepoznata (nedefinirana) ili bidirekcionalna. Ako je veza bidirekcionalna, onda njezini smjerovi moraju biti međusobno inverzni, odnosno računalni kod koji implementira bidirekcionalnu vezu u obje klase mora imati suprotnu funkcionalnost.

Primjer 4.3.1. Dvosmjerno pridruživanje

Student upisuje predmet po vlastitom izboru. Izraditi UML-dijagram razreda i pripadni kod u programskom jeziku C++.

Rješenje:

Na dijagramu se mora nalaziti dvosmjerna veza između razreda Student i Predmet koju se interpretira kao „student upisuje predmet“. Ovdje dvosmjerna veza znači da studenti „vide“ svoj predmet, a predmet „ima informaciju“ koji studenti su ga odabrali. Programski kod u jeziku C++ je

jednostavan. Razred Student ima u svojoj definiciji pokazivač na pojedinca razreda Predmet i obrnuto, Predmet ima vlastiti pokazivač na pojedinca razreda Student.

Osim ručne izrade programskog koda može se iskoristiti funkcionalnost uređivača ArgoUML koji iz dijagrama automatski generira kôd u nekoliko programskih jezika. Ako programski jezik ne podržava pokazivače, implementacija dvosmjerne veze ostvaruje se pomoću obične varijable. Rješenje primjera 4.3.1 prikazano je na slici 4.3.


```
class Student {

 public:

    Predmet *myPredmet;


class Predmet {

 public:

    Student *myStudent;
};
```

Slika 4.3. Rješenje primjera 4.3.1.

Primjer 4.3.2. Jednosmjerno i dvosmjerno pridruživanje

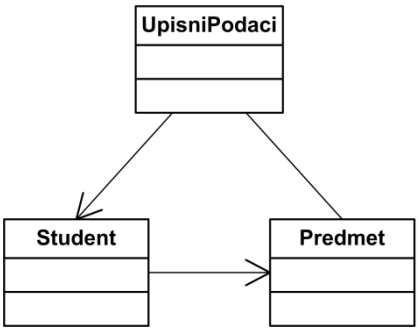
Student upisuje predmet po vlastitom izboru. Informacija o upisima studenata i njihovom odabiru predmeta pohranjena je u upisnim podacima. Zbog sigurnosnih razloga studenti nemaju uvid u sve upisne podatke. Također, osobni podaci o studentima ne smiju biti dostupni kroz podatke o upisanim predmetima. Implementirati dizajnirani sustav u programskom jeziku C++.

Rješenje:

U rješenju ovog primjera moraju se koristiti jednosmjerne i dvosmjerne veze. Zbog jednosmjerne veze od Student prema Predmet, Predmet nema referencu na pojedince Student i ne može doći do njihovih podataka. Zato se uvodi novi razred UpisniPodaci koji je povezan jednosmjernom vezom sa Student i dvosmjernom s Predmet. Na taj način Predmet može preko dvosmjerne veze s UpisniPodaci doći do pokazivača tog razreda na pojedinca tipa Student i tako dobiti informacije o studentima koji su upisali predmet. Rješenje primjera 4.3.2 prikazano je na slici 4.4.

4.4. Vrhovi i nazivi veza

Veza uvijek ima dva vrha, a svaki vrh obvezno dodiruje jedan od dva razreda u vezi. Pri tome se kaže da su vrhovi pridruženi razredima. Veza koja spaja više od dva razreda nije dozvoljena. Vrhovi se još nazivaju uloge ili role (engl. *association role*). Svaki vrh može, ali i ne mora imati naziv ili ime (engl.




```
class Student {

 public:

    Predmet *myPredmet;


class Predmet {

 public:

    UpisniPodaci *myUpisniPodaci;
};


class UpisniPodaci {

 public:

    Student *myStudent;
    Predmet *myPredmet;
};
```

Slika 4.4. Rješenje primjera 4.3.2.

role name), višestrukost, vidljivost i druga svojstva. Nazivi uloga su imenice ili glagoli. Obično u programskom kodu naziv uloge je identičan nazivu varijable koja predstavlja vezu između razreda. U posljednjem primjeru pokazivač na Predmet eksplicitno je nazvan „upisuje“ dok u primjeru koji prethodi tome ima generički naziv automatski dodijeljen od uređivača UML-dijagrama.

Osim naziva uloga i veza može imati vlastiti naziv. U praksi češće se susreću imenovane veze od vrhova. Nije potrebno imenovati svaku vezu već samo onda kada to doprinosi razumijevanju dijagrama. Veze se nazivaju glagolima ili rjeđe imenicama, odnosno mogu opisivati akciju koju jedan razred vrši nad drugim (npr. veza „sluša_predmet“ između razreda Student i Predmet), ili odnos između dva razreda (npr. „mentor“ između razreda Profesor i Student). Primjerice, vezu „mentor“ može se jednako tako imenovati glagolskim oblikom „je_mentor_od“. Ako se veza opisuje glagolom obično se koristi treće lice jednine prezenta, a za imenice, nominativ jednine. Naziv veze uvijek se odabire tako da je sukladan smjeru pridruživanja.

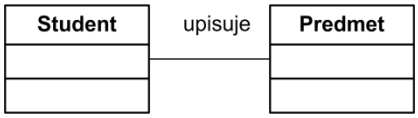
Naposljetku, važno je istaknuti da je ponekad na dijagramu dovoljno naznačiti samo jedan naziv vrha. Tada se naziv veze poistovjećuje s nazivom imenovanog vrha, pri čemu onda i naziv veze može imati označena svojstva poput vidljivosti. U tom slučaju, sva svojstva naziva vrha prenose se na naziv veze.

Primjer 4.4.1. Naziv veze

Student može upisati predmet po vlastitom izboru. Označiti pridruživanje između razreda. Izraditi UML-dijagram razreda i pripadni kod u programskom jeziku C++.

Rješenje:

Rješenje je gotovo identično prvom primjeru iz prethodnog podpoglavlja, osim definiranog naziva pridruživanja koji se u programskom kodu preslikao na naziv pokazivača na pojedinca drugog razreda u vezi. Rješenje primjera 4.4.1 prikazano je na slici 4.5.




```
class Student {

 public:

    Predmet *upisuje;
};


class Predmet {

 public:

    Student *upisuje;

};
```

Slika 4.5. Rješenje primjera 4.4.1.

4.5. Višestrukost pridruživanja

Za svaki vrh veze moguće je definirati višestrukost veze (engl. *multiplicity*) koja određuje koliko pojedinaca, ili objekata, može sudjelovati u odnosu između dva razreda.

Dozvoljene vrijednosti na bilo kojoj strani pridruživanja:

1	= točno 1 pojedinac
n_1	= bilo koji točno određen broj, npr. 0, 1, 3, 5, 15
n_1.. n_2	= između n_1 i n_2, npr. 5…8 → 5, 6, 7 ili 8
n_1..n	= između n_1 i više pojedinaca
n_1…n_2, n_3	= kombinacija, npr. 4…7, 9 → 4, 5, 6, 7 ili 9
n..*	= n ili više pojedinaca, neograničeno
0..* ili * ili n	= više pojedinaca, neograničeno

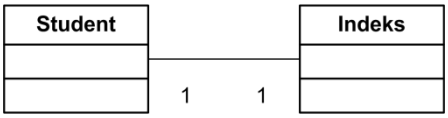
Ako višestrukost pridruživanja nije naznačena podrazumijeva se vrijednost 1 („točno 1 pojedinac“).

U programskom kodu višestrukost se implementira dinamičkom strukturom podataka, npr. std::vector u C++, koja sadržava pokazivače na drugi razred.

Primjer 4.5.1. Višestrukost veze između razreda Student i Indeks

Izradite UML-dijagram razreda gdje svakom studentu pripada točno jedan indeks.

Rješenje primjera 4.5.1 prikazano je na slici 4.6.

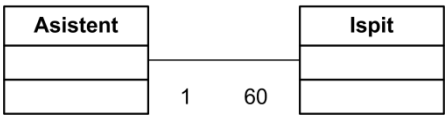


Slika 4.6. Rješenje primjera 4.5.1.

Primjer 4.5.2. Višestrukost veze između razreda Asistent i Ispit

Nakon završetka ispitnog roka asistent svaki put ispravlja točno 60 pismenih ispita.

Rješenje primjera 4.5.2 prikazano je na slici 4.7.

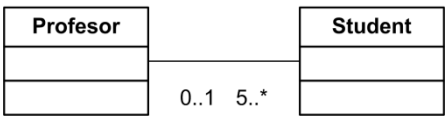


Slika 4.7. Rješenje primjera 4.5.2.

Primjer 4.5.3. Višestrukost veze između razreda Profesor i Student

Na nekom projektu profesor mora voditi barem pet studenata. Studenti mogu biti samo na jednom projektu. Ako to žele, studenti ne moraju prijaviti sudjelovanje na projektu.

Rješenje primjera 4.5.3 prikazano je na slici 4.8.



Slika 4.8. Rješenje primjera 4.5.3.

4.6. Refleksivno pridruživanje

Više pojedinaca istog razreda ponekad moraju međusobno komunicirati. Ova vrsta veze u dijagramu razreda prikazuje se pomoću refleksivnog pridruživanja ili refleksivne agregacije. Ako je potrebno imenovati vezu koriste se nazivi uloga (vrhova), a ne nazivi veza. Kao i kod svakog pridruživanja potrebno je odrediti njegovu višestrukost.

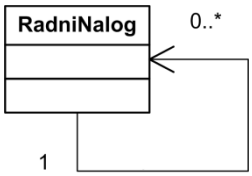
Refleksivno pridruživanje može se primijeniti na bilo koju vrstu pridruživanja dijagrama razreda. To jest, i na jednosmjerne veze, dvosmjerne, ovisnost, i tako dalje.

Primjer 4.6.1. Refleksivna jednosmjerna veza

U nekom auto-servisu otvaraju se radni nalozi nakon zaprimanja automobila klijenata. Ponekad je potrebno nakon otvaranja jednog radnog naloga otvoriti novi koji je pridružen prethodnom. Broj radnih naloga je neograničen.

Rješenje:

Nužno je na razred RadniNalog primijeniti jednosmjernu refleksivnu vezu. Novi radni nalog je uvijek pridružen točno jednom prethodnom radnog nalogu. Budući da neki radni nalozi nemaju niti jedan drugi pridruženi nalog, a njihov krajnji broj je neograničen višestrukost veze je 1 i 0..*. Rješenje primjera 4.6.1 prikazano je na slici 4.9.



Slika 4.9. Rješenje primjera 4.6.1.

4.7. Agregacija i kompozicija

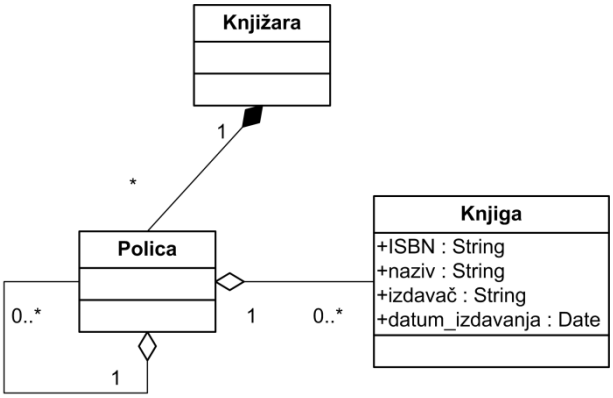
Agregacija (engl. *aggregation*) je vrsta pridruživanja koja pokazuje da jedan razred sadrži druge, tj. da je dio drugog razreda. Kaže se da je razred agregiran (sadržan) u drugom razredu. To je oblik odnosa nadskup-podskup, odnosno veći skup-manji skup. Agregacija se još naziva i odnos cjelina-dio (engl. *whole-part relationship*).

Kompozicija (engl. *composition*) je vrsta pridruživanja slična agregaciji, ali bitna razlika jest da se prilikom uništavanja (engl. *termination*) objekta razreda (tj. pojedinca) uvijek uništavaju i pojedinci razreda koji su dio početnog objekta. Dakle, ako sustav oslobađa zauzetu radnu memoriju sustava i briše pojedince, osim njih bit će izbrisani i svi pojedinci koji su s njim povezani kompozicijom. U slučaju pridruživanja agregacijom to se neće dogoditi i drugi pojedinci (agregirani s početnim objektom) ostati će u radnoj memoriji sustava.

Simbol agregacije i kompozicije uvijek dodiruje razred nadskup, a prazna linija razred podskup. Veze agregacija i kompozicija usmjerene su od nadskupa prema podskupu.

Primjer 4.7.1. Knjižara i police

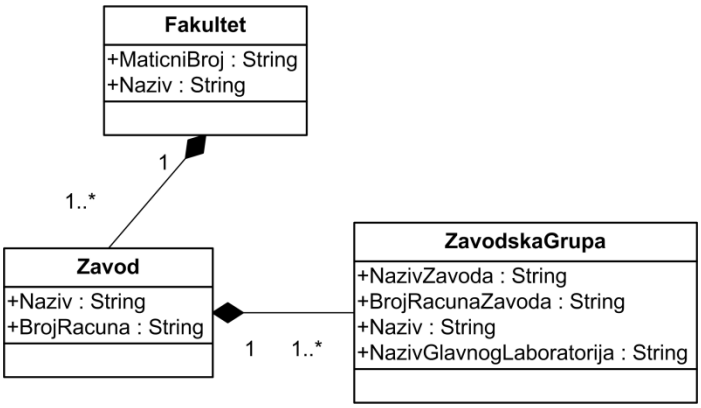
U nekoj knjižari nalazi se više polica koje su organizirane tematski, tako da se unutar polica mogu nalaziti police, unutar njih druge police, i tako dalje. Svakoj polici pripadaju neke knjige. U slučaju reorganizacije knjižare, knjige se raspoređuju na druge police. Svaka knjiga ima svoj ISBN (String), naziv (String), izdavača (String) i datum izdanja (Date). Rješenje primjera 4.7.1 prikazano je na slici 4.10.



Slika 4.10. Rješenje primjera 4.7.1.

Primjer 4.7.2. Organizacija fakulteta

Neki fakultet sastoji se od jednog ili više zavoda, a svaki zavod od jedne ili više zavodskih grupa. Fakultet ima svoj matični broj (String) i naziv (String). Zavod ima svoj naziv (String) i broj računa (String). Naziv i broj računa zavoda nasljeđuju i zavodske grupe, s tim da one osim toga imaju i svoj naziv grupe te dodatno, naziv glavnog laboratorija (String). Rješenje primjera 4.7.2 prikazano je na slici 4.11.



Slika 4.11. Rješenje primjera 4.7.2.

4.8. Pridruživanje, agregacija ili kompozicija?

Važno pitanje za ispravnu izradu dijagrama razreda jest: "Kada se koristi koja vrsta pridruživanja?" Odnosno, kada je potrebno koristiti obične jednosmjerne ili dvosmjerne veze, a kada agregaciju i kompoziciju?

Ako se primijeti da su dva razreda u svezi i da jedan obuhvaća ili sadržava drugi, odnosno, da su povezana odnosom cjelina-dio, tada se obično radi o agregaciji. Ova odluka može ovisiti i o kontekstu kako se promatra sustav. Primjerice, za auto-servis razred „Guma“ može biti povezan agregacijom s razredom Vozilo jer su gume bitan i nedjeljiv dio automobila. Međutim, za trgovinu auto-gumama razredi Vozilo i Guma mogu biti povezani običnim pridruživanjem jer su u kontekstu njihovog sustava

gume, kao proizvod koji prodaju, puno važnije od vozila te se na njih gleda odvojeno, a ne kao na jedinstvenu cjelinu nadskup-podskup.

Nakon prve odluke o vrsti veze potrebno je uočiti da li pojedinci razreda podskupa ostaju u sustavu nakon brisanja objekta razreda nadskup. U slučaju da to nije eksplicitno naglašeno, ispravno je pretpostaviti da pojedince razreda podskup nije potrebno izbrisati iz radne memorije. Naravno, ako se povezani razredi ne odnose jedan prema drugom kao cjelina-dio, onda se radi o običnoj jednosmjernoj ili dvosmjernoj vezi. Naposljetku, ovisno o opisu sustava bira se jedna od ove dvije veze, s tim da se bidirekcionalna podrazumijeva unaprijed ako nisu zadana nikakva ograničenja na odnos pridruženih razreda.

Ukratko, postupak odabira ispravnog pridruživanja je jednostavan: prvo je potrebno odlučiti da li je veza obična ili agregacija. Ako je agregacija odlučuje se da li je možda ipak kompozicija. U suprotnom, ako je veza ipak obična, prosuđuje se da li je jednosmjerna, ali ako nije onda je zasigurno dvosmjerna.

4.9. Atributi

Atributi (engl. *attributes*) su članske varijable razreda. Atributi imaju sljedeća svojstva: stupanj vidljivosti (engl. *visibility*), naziv (engl. *name*), vrsta ili tip (engl. *type*), početna vrijednost (engl. *initial value*). Također za atribute je dozvoljeno, ali nije nužno, definirati promjenjivost (engl. *changeability*) i modifikator (engl. *modifier*).

Za sve atribute razreda potrebno je definirati svojstva vidljivost (engl. *attribute visibility*) i vrsta ili tip (engl. *attribute type*), a moguće je definirati i više različitih svojstva promjenjivosti atributa (engl. *attribute changebility*). Stupanj vidljivosti atributa određuje koji pojedinci mogu pristupiti atributima razreda. Moguće su četiri vrijednosti: javno (*public*; atribut je dostupan svim razredima i paketima), privatno (*private*; atribut je dostupan samo unutar istog razreda), zaštićeno (*protected*; atribut je dostupan unutar istog razreda i izvedenih razreda), a moguće je definirati i stupanj vidljivosti paket (*package*; atribut je dostupan svim razredima istog paketa).

Oznake *public*, *private* i *protected* mogu se zapisati skraćeno simbolima +, - i #. U definiranju svojstva vrste atributa dozvoljene su razne oznake tipova. To su UML-tipovi (Boolean, Integer, String, UnlimitedInteger), Java tipovi podataka (byte, char, double, float, int), Java razredi iz paketa java.util (Collection, Date, List, Set, SortedSet, Time, Vector, …), java.lang, java.lang.math i java.net (URL) paketa. Također je dopušteno korištenje vlastitih tipova razreda i podataka koji su definirani u istom dijagramu. Korištenje razreda iz drugih dijagrama neće dopustiti mnogi uređivači UML-dijagrama, a prilikom ručne izrade dijagrama potrebno ih je naznačiti i objasniti UML-komentarima.

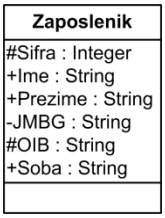
U nekima uređivačima UML-dijagrama moguće je definirati dodatna svojstva promjenjivosti atributa. Tako su moguća i svojstva: *addOnly* (vrijednost atributa može se samo povećavati), *changeable* (vrijednost atributa može se nesmetano mijenjati), *frozen* (vrijednost atributa ili asocijacije ne smije se promijeniti tijekom života (engl. *lifetime*) pripadajućeg objekta), *static* (modifikator, vrijednost atributa je konstanta, ne mijenja se i ne ovisi o životu objekta), *read-only* (vrijednost atributa ne može se mijenjati izvan objekta kojemu pripada, uređivač ArgoUML ne podržava ovo svojstvo). Svojstvo *read-only* nije isto kao i *frozen*, jer je kod *read-only* moguće mijenjati atribut unutar objekta kojemu pripada. Podrazumijevano svojstvo promjenjivosti atributa je *changeable*.

Primjer 4.9.1. Bolnički informacijski sustav i stupanj vidljivosti atributa

U nekom bolničkom informacijskom sustavu nalaze se podaci o zaposlenicima. Svaki zaposlenik ima vlastitu šifru (Integer), ime (String), prezime (String), JMBG (String), OIB (String) i oznaku sobe u kojoj radi (String). Šifra i OIB su zaštićeni podaci, JMBG privatni, a ime i prezime te oznaka sobe su javno dostupni.

Rješenje:

U rješenju ovog kratkog zadatka, dovoljno je definirati samo jedan razred, ali nužno je označiti stupnjeve vidljivosti atributa. Koriste se simboli + („javno dostupno“), - („privatni podatak“) i # („zaštićeni podatak“). Rješenje primjera 4.9.1 prikazano je na slici 4.12.



Slika 4.12. Rješenje primjera 4.9.1.

4.10. Operacije

Operacije (engl. *operations*) su procesi koje razred može izvršiti. Drugim riječima, to su vlastite metode i funkcije razreda. Najvažnija svojstva operacija su vidljivost, te ulazni i izlazni parametri. Svojstvo vidljivosti je identično kao kod atributa i moguće vrijednosti su *public*, *package*, *protected* i *private.* Parametri ili argumenti su svojstveni samo za operacije. Oni određuju ulazne vrijednosti koje operacija može dobivati i izlazne vrijednosti koje vraća pozivajućim funkcijama.

Osim vidljivosti i argumenata u uređivaču ArgoUML za operacije mogu se dodatno odrediti svojstva: modifikator (*static*, *abstract*, *leaf*, *root*, *query*) i istodobnost (*sequential*, *guarded*, *concurrent*).

4.11. Nasljeđivanje

Nasljeđivanje (engl. *inheritance*) je temeljni koncept objektno-orijentiranog programiranja koji se može uspješno modelirati UML-dijagramima. Nasljeđivanje je oblik odnosa između razreda, odnosno nasljedna veza između razreda. Jedan razred je roditelj (nadrazred) jednom ili više drugih razreda (djeca ili podrazredi).

Kaže se da je objekt koji se nasljeđuje proširen u objektu koji ga nasljeđuje. Stoga su definirane dvije vrste odnosa između razreda: generalizacija i specijalizacija. Generalizacija omogućuje stvaranje nadrazreda koja objedinjuje strukturu i ponašanje zajedničko za nekoliko razreda, a specijalizacija omogućuje stvaranje podrazreda koja predstavlja dodavanje novih elemenata.

Odnosi generalizacija i specijalizacija usmjereni su u međusobno suprotnim smjerovima. Generalizacija je usmjerena od podrazreda prema nadrazredu, a specijalizacija od nadrazreda prema podrazredu. Na dijagramima razreda veza nasljeđivanja uvijek je usmjerena od podrazreda prema

nadrazredu, tj. u smjeru generalizacije. Ponekad je korisno nasljeđivanje crtati tako da je nadrazred smješten iznad podrazreda jer to pridonosi jednostavnijem i bržem razumijevanju dijagrama razreda.

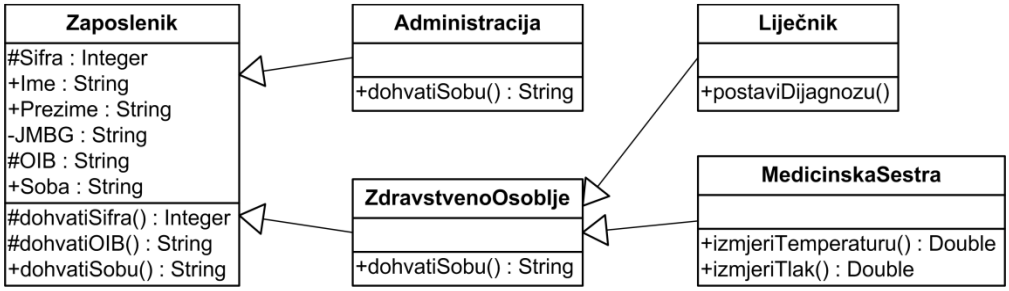
Kod nasljeđivanja uvijek vrijede sljedeća pravila:

- Podrazred uvijek ima više ili jednak broj svojstava u odnosu na nadrazred
- Podrazred nasljeđuje od nadrazreda atribute, relacije i operacije
- Podrazred može biti proširena atributima, operacijama ili relacijama
- Podrazred može imati svoju implementaciju operacija koje je naslijedila

Napomena: Nasljeđivanje nema višestrukost. Ovo svojstvo nasljeđivanja je očito i samorazumljivo, ali korisno ga je izričito naglasiti da bi se izbjegle greške studenata u rješenjima ispita.

Primjer 4.11.1. Bolnički informacijski sustav i nasljeđivanje razreda

Bolnički informacijski sustav iz primjera 4.9.1. je potrebno proširiti. Medicinske sestre i liječnici su zdravstveno osoblje. Zdravstveno osoblje i administracija su različite vrste zaposlenika bolnice. Zaposlenici imaju zaštićenu operaciju s kojima se dohvaća njihova šifra i OIB, te javnu operaciju za dohvaćanje radne sobe. Zdravstveno osoblje može imati radnu obvezu u nekoliko soba istodobno, te imaju vlastitu implementaciju operacije dohvaćanja radne sobe. Liječnici mogu postavljati dijagnozu. Medicinske sestre mogu izmjeriti temperaturu i tlak bolesnika. Rješenje primjera 4.11.1 prikazano je na slici 4.13.



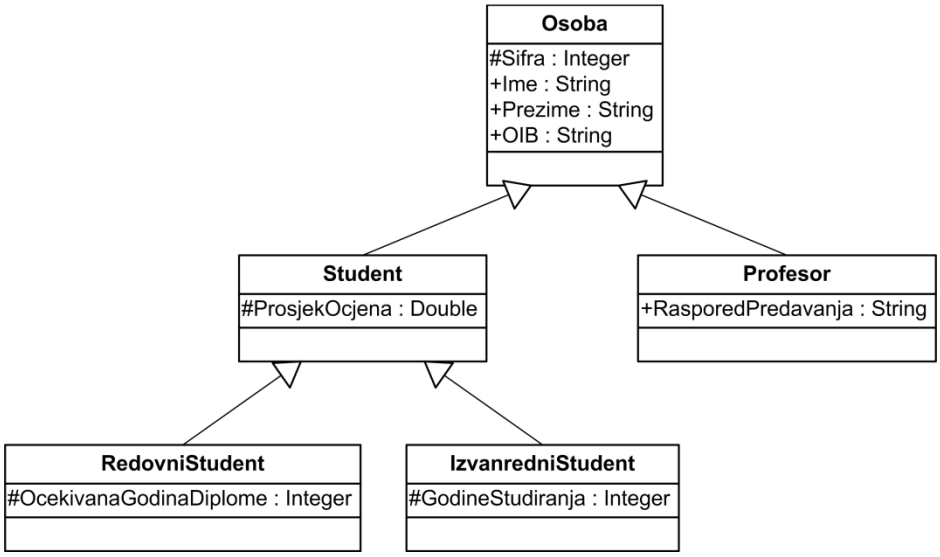
Slika 4.13. Rješenje primjera 4.11.1.

4.12. Nasljeđivanje ili agregacija?

Zadan je sljedeći problem: potrebno je definirati razrede računalnog sustava koji prati studente, upisane predmete i profesore kao nositelje tih predmeta. Pri tome se sustav mora moći jednostavno proširiti i nadograditi tako da kasnije izmjene zahtijevaju minimalne preinake programskog koda. Često zadaci ovog tipa eksplicitno zahtijevaju sve korisnike sustava proširiti u jedan nadrazred koji sadržava njihove zajedničke podatke kao što je ime, prezime, OIB, i tako dalje. Ali čak ako se to izričito ne zahtijeva, dobra je praksa iskoristiti nasljeđivanje za obuhvaćanje zajedničkih svojstava dvaju ili više razreda. U ovom slučaju razredi Student i Profesor mogu naslijediti razred Osoba u kojemu se nalaze atributi Šifra, Ime, Prezime i OIB. Pri tome Profesor i Studenti imaju vlastite specifične atribute kao što su, recimo, RasporedPredavanja za profesora i ProsjekOcjena za studenta.

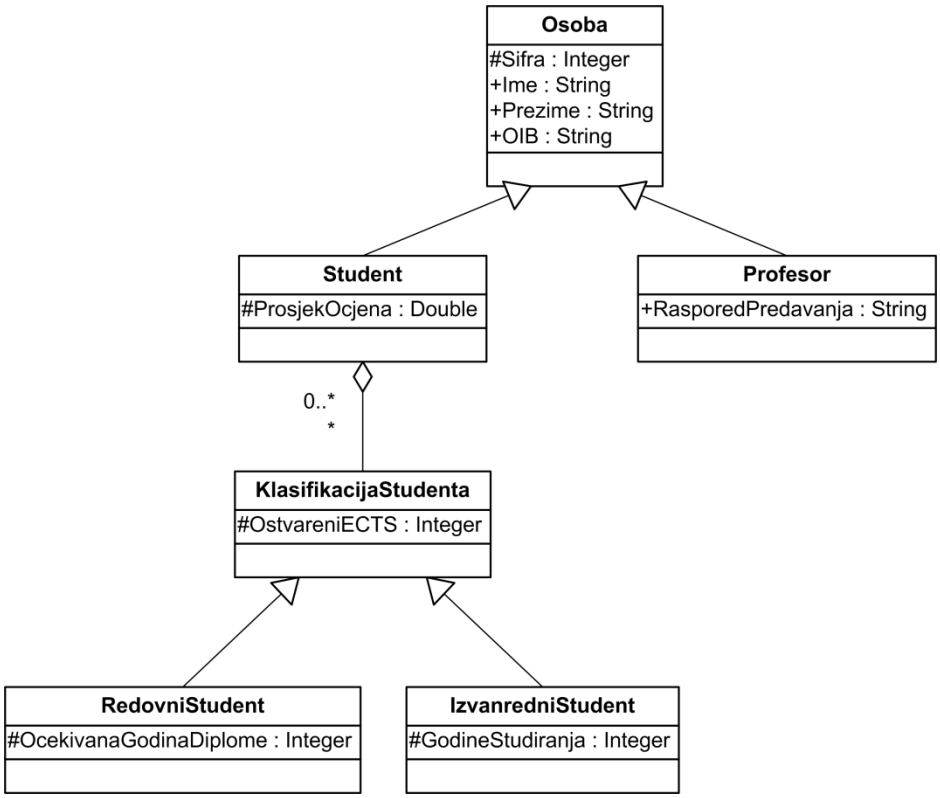
Sada neka se pretpostavi da je u sustav potrebno dodati redovne i izvanredne studente, a svaki od njih ima svoje specifične podatke ili operacije. Ako se dodaju novi razredi RedovniStudent, IzvanredniStudent i izvede ih se iz razreda Student napravit će se greška u dizajnu. Makar je takvo rješenje ispravno, dugoročno će se takav sustav morati puno više izmjenjivati nego da se koristila

agregacija i nasljeđivanje zajedno. Primjerice, što će se dogoditi ako neki student promijeni status iz izvanrednog u redoviti? U tom slučaju morat će pojedinac od razreda IzvanredniStudent promijeniti razred u RedovniStudent. Što ako se doda novo ograničenje? Naprimjer, student ima stipendiju i student nema stipendiju? Samo pomoću nasljeđivanja morale bi se raditi dva nova podrazreda i bilo bi nužno koristiti višestruko nasljeđivanje da se pokriju sve mogućnosti koje se mogu dogoditi u sustavu – redovni student sa stipendijom, redovni student bez stipendije, i tako dalje. Ispravno i nefleksibilno rješenje prikazano je slikom 4.14.



Slika 4.14. Ispravno i nefleksibilno rješenje.

Stoga bi jedino skalabilno rješenje, odnosno fleksibilni dizajn koji dugoročno zahtijeva minimalne preinake, bilo korištenje agregacije za novi razred KlasifikacijaStudenta iz kojeg se izvode podrazredi RedovniStudent i IzvanredniStudent, kao što je prikazano na slici 4.15.

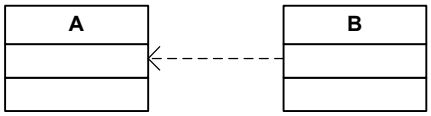


Slika 4.15. Ispravno, skalabilno i fleksibilno rješenje.

Nasljeđivanje je potrebno koristiti samo kada se zajednička svojstva dijele ili rastavljaju od specifičnih, dok se agregacija koristi za modeliranje složenih ili miješanih relacija između razreda. Često je potrebno nasljeđivanje i agregaciju koristiti zajedno.

4.13. Ovisnost

Ovisnost (engl. *dependency*) je odnos koje pokazuje da jedan razred ovisi o drugome razredu, ili jedan paket o drugome paketu, ili jedna komponenta o drugoj, i tako dalje. Može se općenito reći da je ovisnost relacija između dva entiteta u kojoj promjena u jednoj (neovisnom) entitetu može utjecati na drugi (ovisni) entitet. Pri tome se neovisni entitet naziva isporučitelj (engl. supplier), a ovisni klijent (engl. client). Ovisnost je jednosmjerna (unidirekcionalna) veza i u smjeru simbola veze između dva entiteta u dijagramu čitamo je kao: "B ovisi o A". Primjerice, ako postoje dva razreda A i B, te ako se zna da B ovisi o promjenama stanja razreda A, tada će se nacrtati UML veza ovisnosti, usmjerena od B prema A. U ovom primjeru razred A je isporučitelj i B klijent. Ovisnost između razreda („B ovisi o A") prikazana je na slici 4.16.



Slika 4.16. Ovisnost između razreda („B ovisi o A").

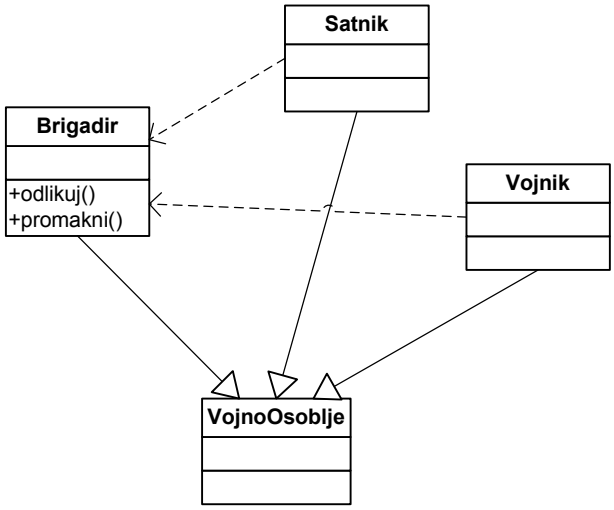
U programskom kodu ovisnost možemo implementirati na više načina, primjerice s funkcijama koje osluškuju događaje (engl. *event handlers*). U tom slučaju razred B koji ima navedenu funkciju je

ovisan o razredu A koji generira događaj. Uređivač ArgoUML ne preslikava svojstvo ovisnosti u programski kod.

Zadatak 4.13.1. Vojno osoblje
Brigadir i satnik su vojno osoblje, kao i vojnik. Brigadir smije odlikovati i promicati sve članove vojnog osoblja (osim samog sebe).

Rješenje:

Čitanjem zadatka odmah se uočava jedan nadrazred (vojno osoblje) i tri podrazreda koji ga nasljeđuju (brigadir, satnik i vojnik). Razred Brigadir ima dvije operacije odlikuj() i promakni() koje izvršava nad pojedincima razreda VojnoOsoblje. Na taj način vojno osoblje ovisi o brigadiru (veza ovisnosti), ali on ne ovisi sam o sebi. Brigadir ne može sam sebe promaknuti i odlikovati, odnosno razred Brigadir ne može izvršiti svoje operacije nad sobom i zato se zna da nema potrebe za refleksivnim pridruživanjem. Zahtjevi nad razinama vidljivosti u zadatku nisu eksplicitno navedeni, stoga se pretpostavlja da su sve operacije i atributi javni (*public*). Operacije nemaju izričito navedene ulazne i izlazne parametre pa ih se ne mora modelirati. Rješenje primjera 4.13.1 prikazano je na slici 4.17.



Slika 4.17. Rješenje primjera 4.13.1.

4.14. Sučelje i realizacija

Sučelje (engl. *interface*) je skup operacija koje specificiraju usluge nekog razreda. Sučelje definira skup operacijskih specifikacija (tj. njihovih potpisa), ali nikada skup njihovih implementacija. Drugim riječima, sučelje je vrsta razreda ali bez atributa, dok sve operacije imaju samo definiciju, bez tijela funkcije i implementacije programskog koda. Ispravan izgled UML-simbola sučelja je sličan razredu ali bez prostora za atribute.

Sučelje je usko povezano s odnosom realizacije (engl. *realization*) koje 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, samo što se kod realizacije „nasljeđuju“ samo operacije s parametrima, bez implementacije. Veza realizacije (strelica) uvijek je

usmjerena od razreda prema sučelju. U uređivaču ArgoUML realizacija ima dva svojstva: sučelje (*supplier*) i razred koji ostvaruje sučelje (*client).*

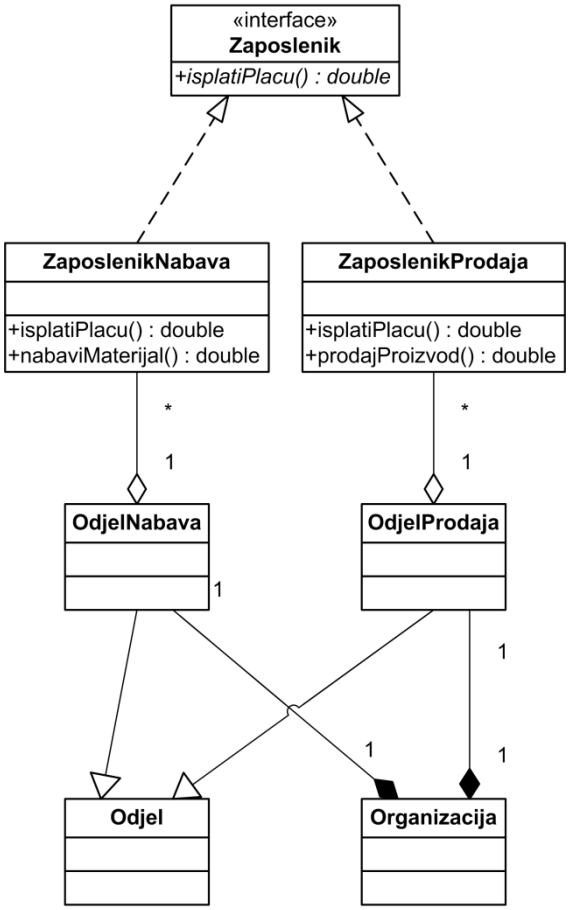
Za razliku od višestrukog nasljeđivanja (engl. *multiple inheritance*) koje je zabranjeno u nekim objektno-orijentiranim programskim jezicima (npr. Java), višestruka realizacija je uvijek dopuštena. Višestruka realizacija omogućuje tzv. opće višestruko nasljeđivanje (engl. *general multiple inheritance*). U Javi je tako moguće implementirati više razreda bez mijenjanja njihove definicije, što je u konačnici slično učinku višestrukog nasljeđivanja.

Primjer 4.14.1. Definirati sučelje Zaposlenik

U nekoj organizaciji postoje odjeli Nabave i Prodaje. Odjeli imaju vlastite zaposlenike, koji rade samo u tom odjelu. Svi zaposlenici dobivaju isplate mjesečnih plaća, ali zaposlenici nabave i prodaje imaju drugačiji algoritam izračuna iznosa plaće. Zaposlenici prodaje mogu prodati proizvode organizacije, a zaposlenici Nabave kupiti ulazni materijal potreban za proizvodnju. Operacije nabave i prodaje vraćaju double vrijednosti. Zaposlenike je potrebno definirati koristeći zajedničko sučelje.

Rješenje:

Iz teksta zadatka razvidna je nužnost korištenja sučelja u definiranju dvije vrste zaposlenika – ZaposlenikNabava i ZaposlenikProdaja. Oba razreda imaju vlastitu implementaciju operacije isplatiPlacu(). Pretpostavljamo da operacija vraća vrijednost tipa *double* (iznos mjesečne plaće). Dodatno ZaposlenikNabava ima vlastitu operaciju nabaviMaterijal(), a ZaposlenikProdaja prodajProizvod(). Oba razreda realiziraju sučelje Zaposlenik koje mora imati definiciju operacije isplatiPlacu(). Vidljivosti svih operacija su *public* jer drugačije nije traženo, niti eksplicitno niti implicitno. Zaposlenici rade u dva različita odjela (Nabava i Prodaja) koji nasljeđuju zajednički razred Odjel. Odjeli pripadaju Organizaciji. U određivanju višestrukosti uočava se da postoji samo jedan odjel Nabave i jedan Prodaje, i samo jedna Organizacija. Dakle, višestrukost oba pridruživanja je 1 – 1. Nestankom (brisanjem) organizacije odjeli moraju biti izbrisani iz sustava, stoga su pridruženi Organizaciji pomoću kompozicije. U pravilu, može se reći da, ako u tekstu zadatka nije eksplicitno navedeno da se pojedinci brišu iz sustava nakon brisanja razreda koji ih sadržava, tada se koristi agregacija.



Slika 4.18. Rješenje primjera 4.14.1.

4.15. Tipovi podataka i obrojčavanja (enumeracije)

Tipovi podataka (engl. *data types*) su slični razredima, ali za razliku od razreda pojedinci se identificiraju samo po vrijednosti. U dijagramu nisu povezani s razredima. U dijagramu razreda iznad naziva obrojčavanja obavezno se nalazi oznaka <<enumeration>>.

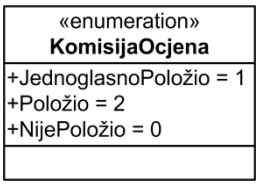
Obrojčavanje (engl. *enumeration*) je oblik tipa podataka koji sadržava uređene parove imenovanih identifikatora i njima pridruženih vrijednosti. Te vrijednosti nazivaju se obrojčani literali. Obrojčavanja se koriste za praktičniji i razumljiviji opis različitih diskretnih vrijednosti sustava. Kao i tipovi podataka, oni nisu povezani s razredima.

Primjer 4.15.1. Ocjene komisije

Na nekom specijalističkom studiju nakon polaganja završnog ispita komisija odlučuje o uspjehu pristupnika. Moguće ocjene su – jednoglasno položio, položio s većinom glasova, nije položio.

Rješenje:

Ocjene komisije moguće je predstaviti s tri diskretne vrijednosti ili kategorije. Pogodno je koristiti obrojčavanje za opis ocjena komisije. Unose se kategorije i pridružuju im se brojčane vrijednosti: JednoglasnoPoložio = 1, Položio = 2, i NijePoložio = 0. Budući da u tekstu nema ograničenja pristupa sve vrijednosti moraju biti javne (*public*). Rješenje primjera 4.15.1 prikazano je na slici 4.19.



Slika 4.19. Rješenje primjera 4.15.1.

4.16. Komentari

Unatoč širokoj formalnoj izražajnosti dijagrama razreda, ponekad je potrebno koristiti i komentare za učinkovitije razumijevanje opisanog sustava. Komentare ne treba upotrebljavati uvijek i u svakoj prilici. Oni se koriste za dodatni opis svrhe nekog razreda, atributa, veza, operacija i drugih elemenata dijagrama. U komentarima je poželjno biti jasan i sažet, te obuhvatiti sve bitne aspekte UML-elementa koji se opisuje.

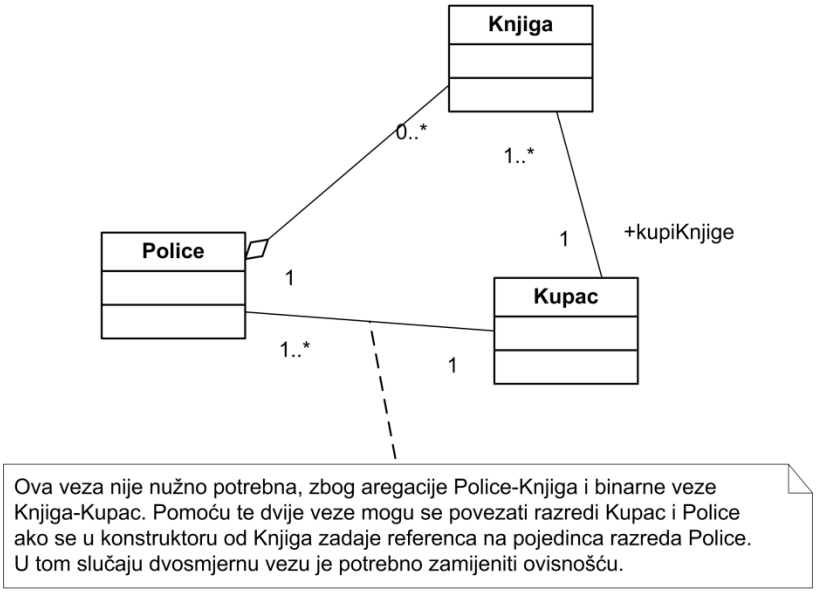
Komentari mogu biti povezani s konkretnim elementom dijagrama (pridruživanje, razred, …), ili se mogu odnositi na cijeli dijagram. Specifični komentari povezani su s elementom neoznačenom vezom, a komentari o cijelom dijagramu nemaju veze i nalaze se na rubu dijagrama, obično u jednom od uglova. U radu s alatom ArgoUML osim komentara za detaljniji opis elementa dijagrama moguće je koristiti i karticu “Documentation” na dnu ekrana.

Primjer 4.16.1. Komentar pridruživanja

U knjižari se nalaze police na kojima su knjige. Dolaze kupci i uzimaju knjige s polica. Ako se neke police bace, ne bacaju se i knjige nego se preraspodjele na druge police. Neke police možda nemaju knjige. Kupci mogu kupiti knjige.

Rješenje:

Dijagram iz ovog primjera je jednostavan, ali uočavaju se barem dva moguća rješenja problema. Budući da iz teksta zadatka nije sigurno koje se rješenje zahtijeva, koriste se komentari kako bi se objasnilo vlastito razmišljanje. Rješenje primjera 4.16.1 prikazano je na slici 4.20.



Slika 4.20. Rješenje primjera 4.16.1.

4.17. Zadaci za vježbu

Zadatak 4.1. Organizacija poduzeća

Neko poduzeće sastoji se od jednog ili više odjela, a na čelu svakog odjela je direktor odjela. Svaki odjel može imati više pododjela. Svaki pododjel ima svog direktora pododjela i radnike. Postoje dva tipa zaposlenika: direktori i radnici. Nadalje, direktori mogu biti direktori odjela ili direktori pododjela. Svaki zaposlenik može raditi na više poslova, a svaki posao može raditi nijedan ili više zaposlenika. Konkretni posao može biti administrativni ili razvojni. Svaki posao ima svoj naziv (String) i rok dovršenja (Date). Ako je posao razvojni, onda on sadrži i matični broj nekog drugog poduzeća (String) za koje se takav posao obavlja. Direktor odjela može zaposliti ili otpuštati sve direktore pododjela i sve radnike u pododjelima (oni ovise o njemu). Direktor pododjela ne može otpuštati niti zapošljavati radnika, ali ima opciju da upita direktora odjela ako se ukaže potreba za otpuštanjem ili zapošljavanjem radnika. Poduzeće ima svoj matični broj (String), broj računa (String) i ukupan broj zaposlenika (int). Svaki odjel ima svoj naziv i adresu, a svaki pododjel, osim naslijeđenog naziva i adrese odjela, ima i vlastiti naziv. Svaki zaposlenik ima svoj matični broj u poduzeću i ime i prezime.

Zadatak 4.2. Organizacija knjižare

Neka se knjižara sastoji od dvije organizacijske jedinice: prodaje i nabave. Knjižara ima dvije vrste zaposlenika: šef knjižare i radnici, s tim da uvijek postoji točno jedan šef knjižare i barem jedan radnik. Šef knjižare vodi obje organizacijske jedinice. Jedinice prodaje i nabave imaju vlastite vrste radnika. Radnici koji rade u prodaji ne rade u nabavi, i obrnuto. Ako se prodaja ili nabava zatvore, radnici neće zato biti otpušteni iz knjižare. Svaki zaposlenik ima svoj JMBG (String), ime (String) i prezime (String). Glavni artikal s kojim barata i prodaja i nabava je knjiga. Svaka knjiga ima svoj ISBN (String), naziv (String), izdavača (String) i datum izdanja (Date). U jedinici prodaje nalazi se jedna ili više polica na kojima se nalazi više knjiga. U nekom trenutku jedna knjiga može se nalaziti samo na jednoj polici. Ako se neku policu ukloni ili premjesti, pripadajuće knjige prebacuju se na drugu policu (knjige ne nestaju zajedno s policom). Isto tako, ako se jedinica prodaje ugasi, s njome ne nestaju i police. Jedinica nabave odjednom dobavlja grupe od po 100 knjiga. U knjižaru dolaze kupci. Kupci mogu: uzeti knjige s polica, vratiti knjige na police, konzultirati se s jednim radnikom koji radi u prodaji, stati u red, izaći iz reda i platiti knjige. Radnik može savjetovati kupca i naplatiti knjige ako radi u prodaji. Radnik u prodaji ili nabavi u svakom trenutku se može konzultirati sa šefom.. Obje organizacijske jedinice knjižare imaju istu funkciju izracunajCijenuKnjige() koju implementiraju na različite načine i njihove razrede potrebno je definirati koristeći zajedničko sučelje. U dijagramu označite međusobne ovisnosti i razrede koji se nasljeđuju.

Zadatak 4.3. Veterinarska ambulanta

Dijagramom razreda modelirajte veterinarsku ambulantu. Veterinarska ambulanta „Umiljato stvorenje“ sastoji se od barem jedne ambulantne jedinice. Svakoj jedinici pripada barem jedan liječnik. Liječnik može biti dio samo jedne jedinice. U slučaju zatvaranja neke jedinice pripadajući liječnici pridružuju se preostalim jedinicama. Ambulantom upravlja direktor. Svaku jedinicu vodi jedan voditelj. Voditelj ne može voditi druge jedinice osim svoje.

Liječnik, voditelj i direktor su zaposlenici. Zaposlenik ima atribute ID (tipa Integer, vidljivost *protected*), Ime (String, *public*) i Prezime (String, *public*), i operaciju primiPlacu() vidljivosti *public*. Direktor drugačije implementira operaciju primiPlacu() od ostalih zaposlenika. Direktor određuje bonus na plaću voditelja, dok voditelj određuje bonus liječnika. Zbog toga voditelj i direktor

implementiraju sučelje IVoditelj s definicijom operacije odrediBonus() koja je vidljiva svim razredima. Operacija odrediBonus() ima jedan ulazni argument tipa Integer i vraća vrijednost tipa Double. Liječnik izvršava nula ili više postupaka. Jedan postupak mora izvršiti barem jedan liječnik. Postupci ponekad mogu sadržavati druge postupke. Postupak se vrši ili izvodi nad životinjom. Životinja ima atribute Vrsta (String, *public*) i Masa (Integer, *public*). Sustav raspoznaje dvije vrste životinja: mala životinja i velika životinja. Jedinica ima atribute ID (Integer, *private*) i Naziv (String, *public*). Postupak ima atribut Opis (String, *public*).

U izradi dijagrama navedite nazive rola gdje je to potrebno, te označite vidljivosti svih atributa i operacija pomoću simbola. Dijagram izradite koristeći najmanji potreban broj razreda i njihovih međusobnih odnosa.

Zadatak 4.4. Tvrtka za popravak informatičke opreme

U tvrtci za popravak informatičke opreme zaposleno je više servisera i jedan voditelj. Oni su djelatnici tvrtke čiji se podaci brišu njezinim zatvaranjem. Svaki djelatnik ima svoje Ime (String, public), Prezime (String, public) i OIB (String, private).

Voditelj otvara radne naloge, dok jedan serviser izvršava barem jedan radni nalog. Jednom radnom nalogu pridružena je jedna stranka u čije ime je radni nalog otvoren. Svaka stranka posjeduje Ime (String, public), Prezime (String, public) i Broj telefona (String, public). Ponekad se jedan radni nalog sastoji od jednog ili više dodatnih radnih naloga zbog složenosti popravaka.

Status radnog naloga je javna varijabla s diskretnim vrijednostima: „zaprimljen“, „u obradi“, „završen – uspješno“ i „završen – neuspješno“. Svaki radni nalog sadržava barem jednu servisnu akciju. Akcije se dijele na: dijagnostiku, podešavanje i ugradnju. Ugradnja sadržava barem jednu računalnu komponentu koja se ugrađuje tijekom akcije. Svaka komponenta ima svoju cijenu (Double, public). Svaki radni nalog obavlja se na informatičkoj opremi koja se dijeli na: periferne jedinice, stolna računala i prijenosna računala. Nakon otvaranja radnog naloga voditelj inicijalno procjenuje trajanje radnog naloga i upisuje taj podatak u radni nalog.

Tvrtka omogućava korisnicima uvid u promjene radnih naloga putem Web aplikacije, stoga se može reći da Web aplikacija ovisi o radnom nalogu. Nakon što je radni nalog završen, bilo uspješno ili neuspješno, serviser zove stranku na telefon i informira je o promjeni statusa radnog naloga. Servisna akcija posjeduje atribute UtrošeniČovjekSati (Double, protected) i CijenaČovjekSata (Double, protected). U radnom nalogu nalazi se javno dostupna operacija IzračunajCijenu() s izlaznom vrijednosti tipa Double.

Nacrtajte dijagram razreda te navedite nazive uloga gdje je to potrebno, te označite vidljivosti svih atributa i operacija pomoću simbola.

5. Dijagrami objekata i paketa

5.1. Dijagrami objekata

5.1.1. Karakteristike objekata

Dijagrami objekata (engl. *object diagrams*) kao i dijagrami razreda pripadaju strukturnim UML-dijagramima, ali njihova uloga je različita: dijagrami objekata prikazuju strukturu sustava u nekom trenutku. Prikaz može biti djelomičan ili cjelovit, odnosno nije nužno prikazati objekte svih razreda sustava. U proizvoljno odabranim trenucima objekti sustava imaju različite vrijednosti atributa i dijagrami objekata prikazuju upravo te vrijednosti, zajedno s objektima i njihovim međusobnim vezama. Najčešće je dovoljno prikazati samo jedan trenutak u radu sustava, ali ponekad ako je stanje sustava izrazito dinamično potrebno je izraditi više dijagrama koji opisuju rad sustava u nekoliko trenutaka s karakterističnim stanjima objekata.

5.1.2. Definiranje objekata

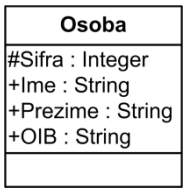
Simbol objekta je pravokutnik s dva pretinca – jedan za naziv objekta i drugi za vrijednosti atributa. Objekti se razlikuju od razreda jer nemaju definicije atributa i procedura, ali imaju jasno označeno stanje važnih atributa. Prikazuju se atributi po kojima se pojedinci unutar dijagrama međusobno razlikuju, odnosno po kojima su oni specifični.

Primjer 5.1.2.1. Objekti razreda Osoba

Razred Osoba ima sljedeće atribute: jedinstvena šifra (Integer), ime (String), prezime (String) i OIB (String). Šifra je zaštićene vidljivosti, a ime, prezime i OIB javno dostupni podaci. Izradite tri pojedinca navedenog razreda s proizvoljnim vrijednostima atributa.

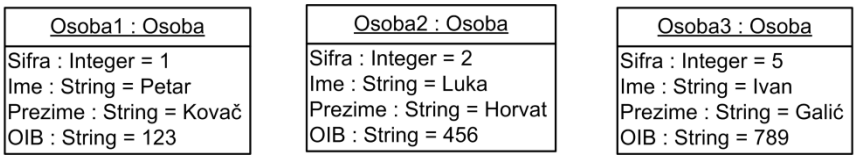
Rješenje:

Prvo treba definirati razred Osoba. U tekstu zadatka zadan je naziv razreda i njegovi atributi, bez operacija. Operacije razreda nisu bitne za objektni dijagram jer objekti nemaju operacije. Stoga, čak i da su operacije zadane u tekstu zadatka, mogli bismo ih ignorirati. Atributi Ime, Prezime i OIB su *public*, a Šifra je *protected*. Prvi dio rješenja prikazan je na slici 5.1.



Slika 5.1. Prvi dio rješenja.

Pojedinci se međusobno razlikuju barem po vrijednosti jednog atributa. Dva pojedinca sa istim atributima nisu moguća u istom dijagramu. U nazivu svakog pojedinca nalazi se njegovo jedinstveno ime i razred kojemu pripada. Primjerice, Osoba1 : Osoba jer pojedinac imena Osoba1 koje je jedinstveno u dijagramu i izveden je iz razreda Osoba. Unosimo vrijednosti atributa pojedinaca razreda Osoba: (Šifra, Ime i prezime, OIB) = { (1, Petar Kovač, 123); (2, Luka Horvat, 456); (3, Ivan Galić, 789). Vrijednosti su proizvoljno odabrane. Konačno rješenje primjera 5.1.2.1 prikazano je na slici 5.2.



Slika 5.2. Konačno rješenje primjera 5.1.2.1.

5.1.3. Veze između objekata

Dijagrami objekata sastoje se od objekata i veza. Za pridruživanja objekata najčešće se koristi dvosmjerna veza, a dozvoljeno je korištenje i kompozicije. Ako su dva razreda povezana nasljeđivanjem, veza između njihovih objekata biti će dvosmjerna, a može dodatno biti označena i s oznakom „parent“ koja se nalazi pokraj točke gdje veza dodiruje razred roditelj.

U izradu dijagrama objekata prvo odaberemo razrede čije objekte ćemo prikazati. Najčešće to su najvažniji razredi za ispravno funkcioniranje sustava, odnosno oni razredi bez kojih sustav ne može ispuniti svoje temeljne funkcionalnosti. Nakon toga unosimo proizvoljne vrijednosti u objekte, ali pazimo da one ispravno demonstriraju vrijednosti koje će atributi imati tijekom rada sustava. Na kraju povezujemo objekte, označavamo veze gdje je to potrebno i završavamo izradu dijagrama.

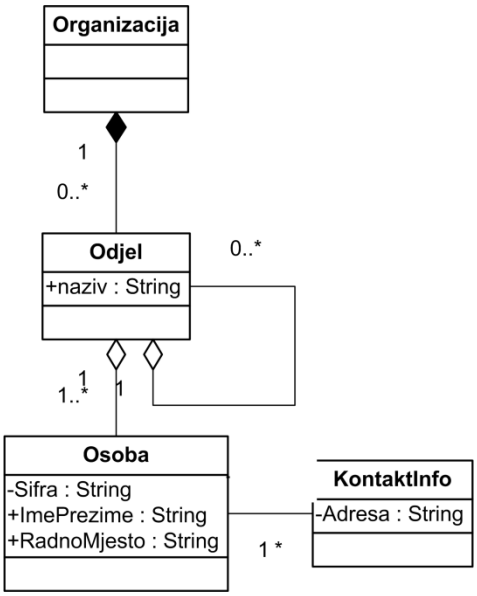
U pravilu se za neki složeni sustav radi nekoliko dijagrama objekata koji uvijek odgovaraju njegovim obrascima uporabe. Tijekom rada i uporabe sustava stvorit će se različiti pojedinci ili će oni imati razna stanja. Dijagrami objekata moraju prikazati stanje sustava za svaki obrazac uporabe prethodno definiran u istoimenim dijagramima.

Primjer 5.1.3.1. Veze između objekata

Neka organizacija ima više odjela. Odjeli se mogu dodavati i brisati. Svaki odjel ima svoj naziv (String). Odjeli se mogu sastojati od pododjela. Svakom odjelu može pripadati zaposlenik. Zaposlenici imaju šifru (Integer), ime i prezime (String) i naziv radnog mjesta (String). Šifra je zaštićen podatak. Nekim zaposlenicima mogu biti pridružene vlastite kontakt informacije.

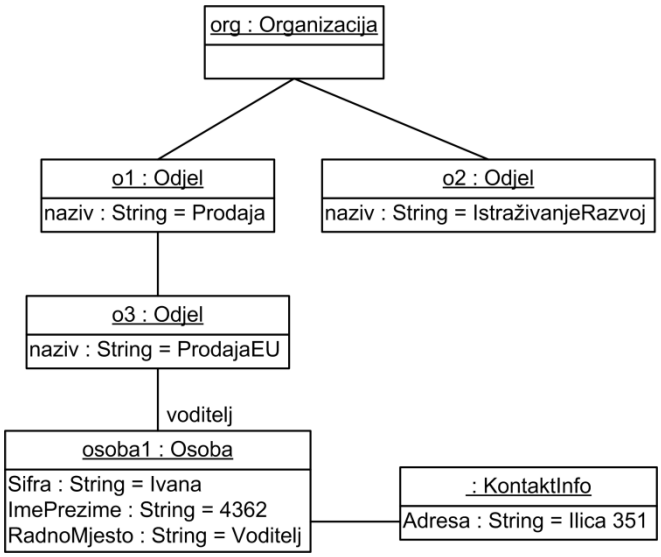
Rješenje:

Čitajući zadatak uočavamo četiri razreda: organizacija, odjel, osoba i kontakt informacije. Razred Organizacija sadržava razred Odjel koji je u reflektivnoj vezi sam sa sobom. Jednom odjelu može pripadati od 0 do neograničeno mnogo drugih odjela. Osoba pripada samo jednom odjelu, te u svakom odjelu mora raditi barem jedna osoba. Kontakt informacije povezane su s osobom. Prvi dio rješenja primjera 5.1.3.1 prikazan je na slici 5.3.



Slika 5.3. Prvi dio rješenja primjera 5.1.3.1.

Postoji samo jedna organizacija i nekoliko odjela, stoga će se u rješenju nalaziti samo jedan pojedinac razreda Organizacija i tri pojedinca razreda Odjel – dva pojedinca koji su neposredno pridruženi organizaciji i jedan pojedinac koji je pridružen drugom odjelu. Jednom od odjela bit će pridružen pojedinac zaposlenik osoba1. Veza pojedinca osoba1 s odjelom o3 je nazvana „voditelj“. To je rola (uloga) pojedinca osoba1 u odjelu o3. Istom zaposleniku je pridružen anonimni pojedinac kontakt podataka Time će se predstaviti sve zahtijevane karakteristike sustava. Konačno rješenje primjera 5.1.3.1 prikazano je na slici 5.4.



Slika 5.4. Konačno rješenje primjera 5.1.3.1.

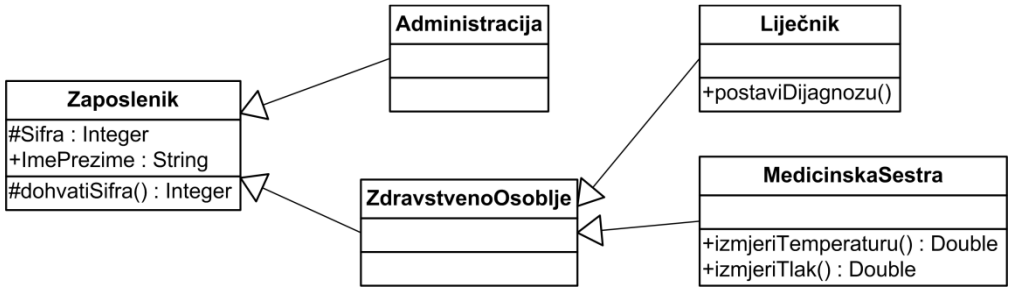
Primjer 5.1.3.2. Nasljeđivanje objekata

U bolnici rade zaposlenici. Medicinske sestre i liječnici su zdravstveno osoblje. Zdravstveno osoblje i administracija su različite vrste zaposlenika bolnice. Prikazati u objektnom dijagramu jednog pojedinca razreda liječnik i administracija. Svaki zaposlenik ima šifru (integer) i ime i prezime (string).

Šifra je zaštićeni podatak. Liječnik može postaviti dijagnozu, dok medicinska sestra može izmjeriti temperaturu i tlak.

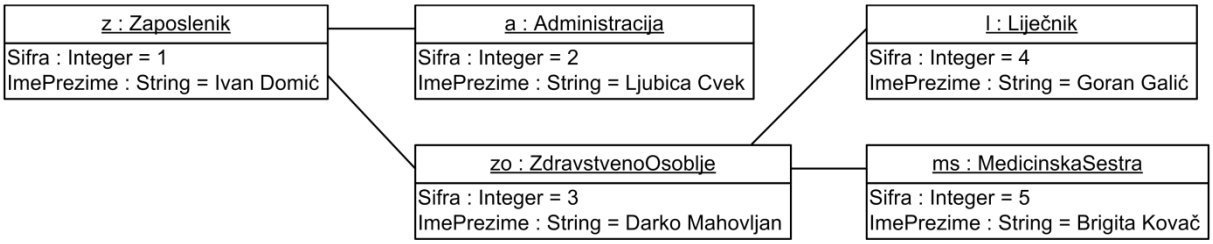
Rješenje:

U primjeru bolničkog informacijskog sustava iz poglavlja o dijagramima razreda definirani su razredi: zaposlenik, administracija, zdravstveno osoblje, liječnik i medicinska sestra, te njihovi međusobni odnosi. Prvi dio rješenja prikazan je na slici 5.5.



Slika 5.5. Prvi dio rješenja primjera 5.1.3.2.

Pojedinca razreda koji nasljeđuju druge razrede automatski nasljeđuju sve atribute roditelja. Stoga je za pojedince Liječnik i MedicinskaSestra potrebno definirati karakteristične vrijednosti koje su definirane u razredima Zaposlenik i MedicinskoOsoblje. Pojedinci razreda Administracija nasljeđuju atribute razreda Zaposlenik. Konačno rješenje primjera 5.1.3.2.



Slika 5.6. Konačno rješenje primjera 5.1.3.2.

5.1.4. Zadaci za vježbu

Zadatak 5.1. Objektni dijagram autonomnog robota

Potrebno je modelirati dio računalnog sustava autonomnog robota. Robot ima neko stanje r, npr. stoji ili kreće se. Robot sadrži razred za interni prikaz okoline, tj. svijeta. Model svijeta ili okoline sastoji se od nekoliko elemenata i površina. Svaka površina može sadržavati zidove i vrata. Zidovi i vrata imaju svojstvo Širina koje je izraženo u elementima slike (Integer). Svi elementi koji tvore površinu su međusobno ovisni.

5.2. Dijagrami paketa

5.2.1. Karakteristike paketa

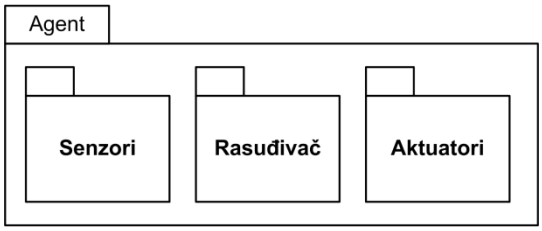
Dijagrami paketa (engl. *package diagram*) prikazuju kako je sustav podijeljen u logične cjeline ili skupove pokazujući ovisnost između tih cjelina. Korisni su za uspostavu hijerarhijskog odnosa između razreda i njihovo grupiranje u smislene i funkcionalne cjeline. Dijagrami paketa također pripadaju strukturnoj skupini UML-dijagrama i prikazuju vremenski statička svojstva sustava.

Paket (engl. *package*) je skup različitih UML-objekata. U programskom kodu paketi se interpretiraju kao struktura *namespace* u programskom jeziku C++ ili *package* u Javi, ali paketi mogu sadržavati druge pakete, objekte, razrede, komponente, pa čak i dijagrame. Pomoću paketa moguće je objediniti obrasce uporabe kako bi se strukturalno objasnila funkcija modeliranog sustava. U višeslojnim sustavima paketi se mogu koristiti za grupiranje pojedinačnih slojeva programske podrške kako bi se mogao objasniti tok podataka između različitih slojeva.

Primjer 5.2.1.1. Dijagram paketa inteligentnog agenta

Definirajte zajednički paket nekog inteligentnog agenta koji se sastoji od tri manja paketa: senzora, rasuđivača i aktuatora.

Rješenje primjera 5.2.1.1 prikazano je na slici 5.7.



Slika 5.7. Rješenje primjera 5.2.1.1.

5.2.2. Vidljivost i ugniježđenje paketa

Slično kao razredi, i paketi imaju mogućnost definiranja vidljivosti objekata koje sadržavaju. Oznake imaju jednako značenje: + *public*, - *private*, # *protected*. Javno vidljivi objekti (*public*) nazivaju se eksporti paketa (engl. *package exports*), jer njih mogu koristiti objekti u drugim importirajućim paketima.

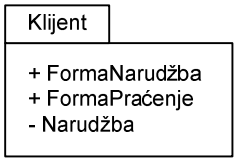
U prikazu paketa može se koristiti tekstualno ugniježđenje (engl. *textual nesting*) ili grafičko ugniježđenje (engl. *graphical nesting*). To su dva različita i jednako vrijedna pristupa označavanju objekata unutar dijagrama.

Primjer 5.2.2.1. Ugniježđenje paketa

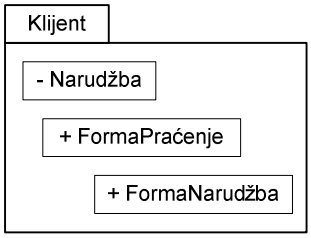
Klijent neke raspodijeljene aplikacije sastoji se od sljedećih formi ili prozora: forme za narudžbu, forme za praćenje narudžbe i forme za pregled izvršene narudžbe. Forme za narudžbu i praćenje su javno dostupne svim korisnicima aplikacije, dok formi za pregled izvršene narudžbe može pristupiti samo administrator aplikacije. Prikazati sve komponente klijenta s tekstualnim i grafičkim ugniježđenjem.

Rješenje:

Klijentska aplikacija ima tri forme koje imaju različite vidljivosti. Forme za narudžbu i praćenje su *public*, dok je forma za pregled izvršene narudžbe *private*. Koristimo simbole + i – za označavanje razina vidljivosti komponenti *public* i *private*. U tekstualnom ugniježđenju prikazujemo komponente u istom paketu jednu ispod druge, a u grafičkom kao zasebne pakete unutar većeg paketa Klijent. Prvi dio rješenja (tekstualno ugniježđenje paketa) prikazan je na slici 5.8, a drugi dio rješenja (grafičko ugniježđenje paketa) prikazan je na slici 5.9.



Slika 5.8. Prvi dio rješenja (tekstualno ugniježđenje paketa).



Slika 5.9. Drugi dio rješenja (grafičko ugniježđenje paketa).

5.2.3. Veze paketa

Paketi mogu biti povezani s tri vrste veza: importiranje (engl. *import stereotype*), nasljeđivanje (engl. *inheritance*) i ovisnost (engl. *dependency*). Importiranje se označava vlastitim stereotipom <<import>>. Importiranje je usmjerena veza i omogućuje jednom razredu da „vidi“ objekte drugog.

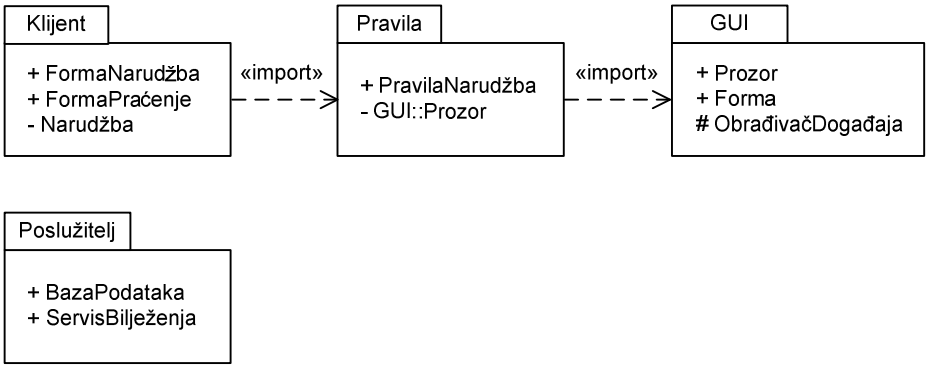
Primjer 5.2.3.1. Importiranje i eksportiranje paketa

Raspodijeljena aplikacija iz prethodnog primjera proširena je s paketima Posluživač, Pravila i GUI. Server ima objekte BazaPodataka i ServisBilježenja (engl. *LoggingService*). Svi su javno dostupni. Paket Pravila ima javno dostupni objekt PravilaNarudžba i zaštićeni objekt Prozor iz paketa GUI. Paket GUI sadržava objekte Prozor, Forma i zaštićeni ObrađivačDogađaja (engl. *EventHandler*).

Rješenje:

Paket GUI eksportira dva razreda: Prozor i Forma. ObrađivačDogađaja nije eksportiran jer je označen kao zaštićen dio paketa. Objekti koje paket eksportira vidljivi su samo onom paketu koji ga importira. U ovom primjeru Pravila eksplicitno importira paket GUI. Pa su tako GUI: Prozor i GUI:Form vidljivi sadržaju paketa Pravila. Ali GUI:ObrađivačDogađaja nije vidljiv jer je zaštićen (*protected*). Poslužitelj ne importira paket GUI, te stoga sadržaji tog paketa nemaju pravo pristupa nijednom objektu iz paketa GUI. Isto tako, objekti iz paketa GUI ne mogu pristupiti sadržaju paketa Poslužitelj. Zavisnosti importiranja i pristupa nisu tranzitivne. U ovom primjeru Klijent importira Pravila i Pravila importiraju GUI, ali Klijent ne importira GUI i ne može pristupiti njegovim objektima. Zato sadržaj paketa Klijent nema pravo pristupa eksportima paketa GUI, ali ima pristup eksportima paketa Pravila.

Da bi Klijent dobio pristup morao bi biti neposredno ili eksplicitno povezan s GUI. Rješenje primjera 5.2.3.1 prikazano je na slici 5.10.



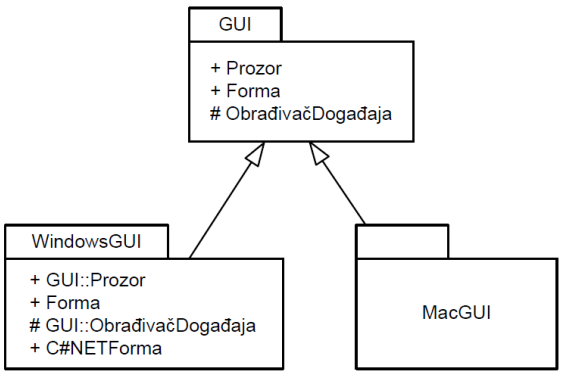
Slika 5.10. Rješenje primjera 5.2.3.1.

Primjer 5.2.3.2. Nasljeđivanje paketa

U raspodijeljenoj aplikaciji iz prethodnog primjera potrebno je napraviti odvojena sučelja za operacijske sustave Windows i MacOS, ali nužno je maksimalno iskoristiti postojeći paket GUI.

Rješenje:

Nasljeđivanje između paketa je vrlo slično nasljeđivanju između razreda. Na sljedećoj slici prikazan je paket WindowsGUI i MacOSGUI koji nasljeđuju zajedničku strukturu iz generičkog paketa GUI koji eksportira dva razreda (Prozor i Forma) i ima jednu zaštićeni razred (ObrađivačDogađaja). Dva specijalizirana paketa (WindowsGUI i MacOSGUI) nasljeđuju javne i zaštićene elemente od općenitijeg paketa (GUI). Isto kao i kod nasljeđivanja razreda, paketi mogu zamijeniti elemente općenitog razreda sa svojima. Na taj način WindowsGUI nasljeđuje iz GUI i uključuje razrede GUI::Prozor i GUI::ObrađivačDogađaja, nadjačava (engl. *override*) razred Forma i dodatno definira vlastiti novi razred C#NETForma. Rješenje primjera 5.2.3.2 prikazano je na slici 5.11.



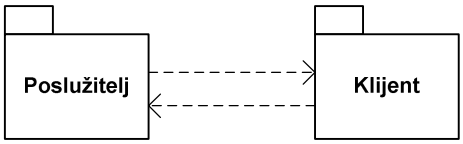
Slika 5.11. Rješenje primjera 5.2.3.2.

Primjer 5.2.3.3. Ovisnost između paketa

Dvoslojni informatički sustav sastoji se od komponenti Klijent i Poslužitelj. One su međusobno ovisne, jer izvršavanje poslužiteljskih funkcija utječu na izgled i ponašanje klijenta, dok akcije korisnika upravljaju ponašanjem poslužitelja.

Rješenje:

Dijagram paketa u ovom primjeru je vrlo jednostavan. Klijent i Poslužitelj su međusobno ovisni, a veza ovisnosti je jednosmjerna i usmjerena. Stoga je potrebno spojiti Klijent i Poslužitelj s dvije veze ovisnosti - jedna je usmjerena od Poslužitelja prema Klijentu, a druga obrnuto. Rješenje primjera 5.2.3.3 prikazano je na slici 5.12.



Slika 5.12. Rješenje primjera 5.2.3.3.

5.2.4. Stereotipovi paketa

Stereotipovi su uobičajen način proširenja UML-a. U dijagramima paketa, stereotipovi se koriste za preciznije definiranje vrste ili tipa paketa. Stereotipovi nemaju posebne simbole, već se u postojeću oznaku paketa ispod njegovog naziva umeće oznaka stereotipa. Korištenje stereotipova podataka nije nužno.

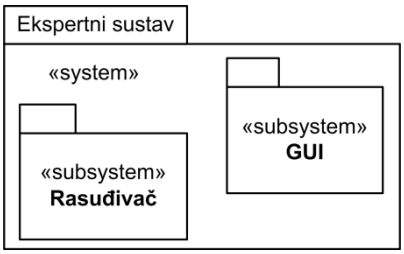
U dijagramima paketa mogu se koristiti sljedeći stereotipovi:

1. «facade» Paket definira samo pogled (engl. *view*) na neki drugi paket
2. «framework» Paket se sastoji većinom od programskih okvira i obrazaca
3. «stub» Paket je samo posrednik (engl. *proxy*) za javno dostupne komponente drugih paketa
4. «subsystem» Paket predstavlja nezavisni dio cijelog sustava koji se modelira
5. «system» Paket predstavlja cijeli sustav koji se modelira

Primjer 5.2.4.1. Sistemska komponenta ekspertnog sustava

Rasuđivač je najvažnija sistemska komponenta nekog ekspertnog sustava. Druga neophodna komponenta je grafičko korisničko sučelje. Prikazati sve komponente u paketnom dijagramu.

Rješenje primjera 5.2.4.1 prikazano je na slici 5.13.



Slika 5.13. Rješenje primjera 5.2.4.1.

5.2.5. Zadaci za vježbu

Zadatak 5.2. Sustav kućnog alarma ima dva paketa. Paket Senzor generira poruke koje sluša paket Nadzornik. Paket Nadzornik sastoji se od tri razreda: Upravljanje, Podaci i Sučelje. Izraditi dijagram paketa sustava koristeći grafičko ugniježđenje.

Zadatak 5.3. Najvažniji paketi računalnog sustava za prodaju preko Interneta su Narudžbe i Kupci koji pripadaju većem paketu Prodaja. Razredi u Kupci ovise o paketu Narudžbe. Time cijeli paket Prodaja generira poruke koje sluša abstraktni paket SučeljeBazePodataka. Ovaj paket obuhvaća specijalizirane pakete za pristup pojedinačnim bazama podataka: SučeljeOracle i SučeljeMSSQL. Cijela aplikacija je izrađena u Javi i za implementaciju korisničkog sučelja korišten je AWT Toolkit. Najvažniji dijelovi korisničkog sučelja su razredi u paketima NarudžbeUlaz i ObavijestiUlaz. Paket AplikacijaNarudžbe šalje poruke u paket Prodaja, a dobiva poruke iz NarudžbeUlaz. Na sličan način operacije u AplikacijaObavijesti pozivaju se temeljem akcije korisnika u ObavijestiUlaz i nakon obrade prosljeđuju direktno u paket Kupci. U svim paketima koriste se globalne operacije i konstante koje su definirane u razredima Količina, Novac i RasponPodataka. Ovi razredi nalaze se u istom paketu Općenito. Modelirati opisani dijagram paketa.

6. Dijagrami stanja i aktivnosti

6.1. Karakteristike dijagrama

Dijagrami stanja (engl. *statechart*, *UML state machine diagram*) i aktivnosti (engl. *activity diagram*) pripadaju ponašajnim dijagramima (engl. behavioral diagram) u UML-u, zajedno s dijagramom obrazaca uporabe. Za razliku od dijagrama obrazaca uporabe, dijagrami stanja i aktivnosti prikazuju funkcionalnost softverskog sustava iz perspektive unutrašnjosti sustava. Pritom ovi dijagrami ne prikazuju niti aktore niti vanjsko sučelje prema krajnjim korisnicima. Budući da razrađuju ponašanje sustava u smislu aktivnosti i prijelaza između stanja, svrstavaju se u dinamičke dijagrame.

Iako slični, ove dvije vrste dijagrama imaju i određene razlike koje se mogu uočiti. Prva razlika odnosi se na njihovu svrhu. Dijagram stanja služi za opis diskretnih stanja sustava i prijelaza između tih stanja. Težište mu je na unutarnjem djelovanju dijelova sustava i često prikazuje prijelaze između stanja u sustavu koji su poticani događajima. Dijagram aktivnosti prikazuje radni tok (ili kontrolni tok) aktivnosti koje se obavljaju u sustavu korak po korak. Stoga je kod dijagrama aktivnosti naglasak na jednostavnosti i poslovnim operacijama koje se uvijek odvijaju slijedno, jedna za drugom.

Druga razlika je u tome što dijagrami stanja na prijelazima između stanja sadrže naziv događaja koji je aktivirao prijelaz i često poziv izlazne metode ili operacije koja se događa, dok kod dijagrama aktivnosti to nije slučaj.

6.2. Dijagram stanja

6.2.1. Elementi dijagrama

Dijagram stanja sastoji se od sljedećih (najčešćih) gradivnih elemenata:

1. Početno stanje i konačno stanje. Početno stanje označeno je crnim krugom, a konačno stanje označeno je zaokruženim crnim krugom, slika 6.1. Dijagram stanja može imati jedno početno i više konačnih stanja.

Slika 6.1. Oznake početnog i konačnog stanja, prikazanih zajedno s prijelazima.

2. Stanje, označeno zaobljenim pravokutnikom. Pritom gornji dio pravokutnika sadrži naziv stanja, a doljnji dio, odvojen horizontalnom crtom, sadrži aktivnosti koje se događaju u tom stanju, slika 6.2. Stanje ne mora sadržavati donji dio, ono mora sadržavati samo naziv. Uobičajene oznake za trenutke kada se aktivnosti odvijaju su: pri ulasku u stanje (engl. *entry*) pri izlasku iz stanja (engl. *exit*) i nedefinirani trenutak izvođenja (engl. *do*).

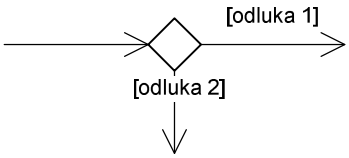
Slika 6.2. Oznaka stanja.

3. Prijelaz između stanja, označen strelicom. Opći oblik prijelaza između stanja prikazan je na slici 6.3. Prijelaz između stanja može sadržavati:
   3.1. Događaj (engl. *event*) koji je izazvao prijelaz (ako takav postoji, inače se prijelaz uvijek događa), prikazan svojim nazivom (engl. *eventName*).
   3.2. Ograničenje ili izraz koje treba zadovoljiti događaj da bi se prijelaz ostvario (engl. *guardExpression*), prikazan u uglatim zagradama.
   3.3. Akciju koja se događa prilikom prijelaza ukoliko je ograničenje zadovoljeno. Akcija može biti bilo koje vrste, npr. poziv vanjske procedure, pridruživanje vrijednosti varijabli, itd. Akcija je od događaja odvojena znakom "/".
   3.4. Tip događaja (engl. *type*). Događaji imaju najčešće predefinirane tipove kao što su: poziv (engl. *call*), signal (engl. *signal*), promjena (engl. *change*), vrijeme (engl. *time*).

   Sve komponente prijelaza mogu se izostaviti. Opći oblik prijelaza između stanja prikazan je na slici 6.3.

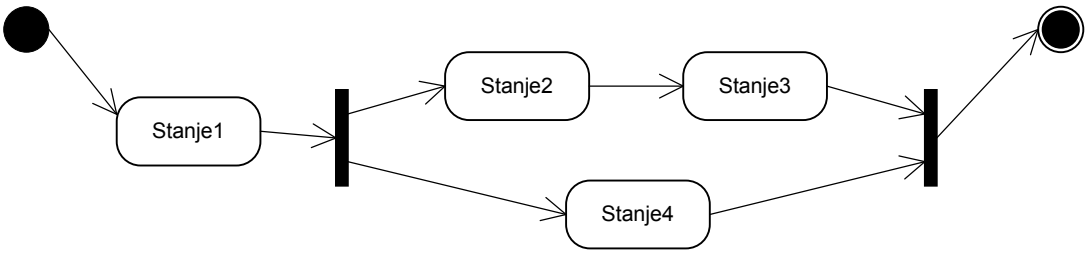
Slika 6.3. Opći oblik prijelaza između stanja.

4. Odluka ili uvjetno grananje (engl. *decision*), prikazana bijelim, dijamantnim oblikom, slika 6.4. Uz odluku, moguće je i spajanje (engl. *merge*), koje se prikazuje s istim bijelim dijamantnim oblikom, samo s dvije strelice koje idu u čvor, a jednom koja izlazi.



Slika 6.4. Oznaka odluke.

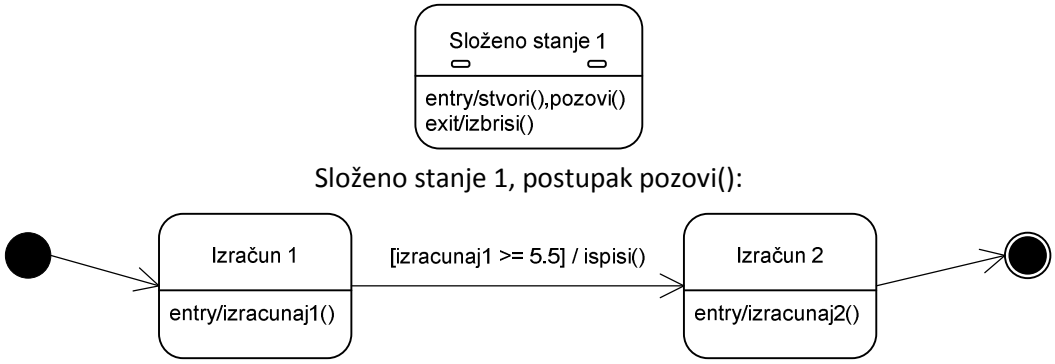
5. Račvanje i skupljanje (engl. *fork and join*) koje se najčešće koristi pri paralelnom odvijanju nekih aktivnosti. Pritom se dijagram stanja račva od nekog stanja u niz drugih stanja da bi se nakon prolaska kroz ta paralelna stanja ponovno skupio, slika 6.5. Sastanak (engl. *rendezvous*) ima više prijelaza koji ulaze i izlaze. Jednom kad su svi ulazni prijelazi aktivirani pokreću se svi izlazni prijelazi, svaki u zasebnoj dretvi [Lethbridge 2005].



Slika 6.5. Oznaka račvanja i skupljanja.

6. Složeno stanje, koje se sastoji od hijerarhije stanja i prijelaza, prikazano zaobljenim pravokutnikom unutar čega se nalazi novi dijagram stanja. Budući da većina alata za crtanje UML-

dijagrama ima problema s prikazom takvih složenih stanja, uobičajeno je hijerarhiju stanja crtati odvojeno, kao na slici 6.6.



Slika 6.6. Primjer hijerarhije stanja.

6.2.2. Zadaci za vježbu

Zadatak 6.1. Izmjena podataka u ćeliji

Modelirati dijagramom stanja izmjenu podataka u ćeliji unutar tablice tabličnog kalkulatora. Unos počinje klikom miša ili pozicioniranjem putem tipkovnice, prilikom čega se na ekranu podeblja ćelija u tablici i omogući upis teksta. Pretpostavite da unos teksta traje dok se pritišće bilo koji znak osim znakova "Enter", "Tab" ili "Esc". Prilikom unosa, pritisnuti znak se ispisuje na ekran, ali ne briše se dosadašnji sadržaj ćelije.

U slučaju uspješnog završetka unosa (znakovi "Enter" ili "Tab"), dolazi do izmjene podataka u ćeliji. Najprije se dosadašnji sadržaj ćelije briše iz memorije. Potom se trenutni podatak pohrani u memoriju i ispiše u ćeliju. U slučaju obustave unosa (znak "Esc"), podatak u ćeliji ostaje netaknut. U svakom slučaju nakon završetka unosa ćelija se defokusira, odnosno postaje normalne debljine.

Zadatak 6.2. Paralelizacija izvođenja procesa

Upotrebom dijagrama stanja potrebno je modelirati paralelizaciju poslova pri ulasku u procesor koji sadrži dvije jezgre. Poslovi se nalaze u priručnoj memoriji. Pretpostavite da svaki posao ima zabilježenu količinu priručne memorije koju je zauzeo i približnu količinu vremena potrebnu za izvođenje. Pretpostavite da jedna jezgra izvodi samo jedan posao u nekom trenutku. Ako su obje jezgre pune, posao ostaje u priručnoj memoriji.

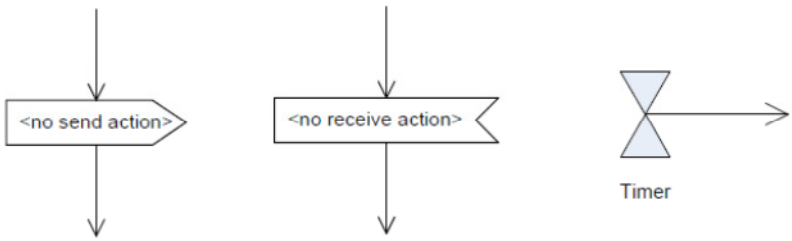
Posao će se nalaziti u priručnoj memoriji sve dok se neka jezgra ne oslobodi. Zatim posao prelazi u stanje razmatranja. U tom stanju, onaj posao koji zauzima najviše priručne memorije bit će odabran za izvršavanje. Svi ostali poslovi će se vratiti u prethodno stanje. Svaki posao pri izvršavanju u procesoru mora proći kroz tri stanja: 1) predobradu, u kojoj se posao optimira, 2) izvođenje, na kraju čega se rezultati zapisuju u pričuvnu memoriju, i 3) brisanje, pri čemu se briše stanje u jezgri koja je posao izvršavala i postavlja se signalna zastavica da je jezgra prazna.

6.3. Dijagram aktivnosti

6.3.1. Elementi dijagrama

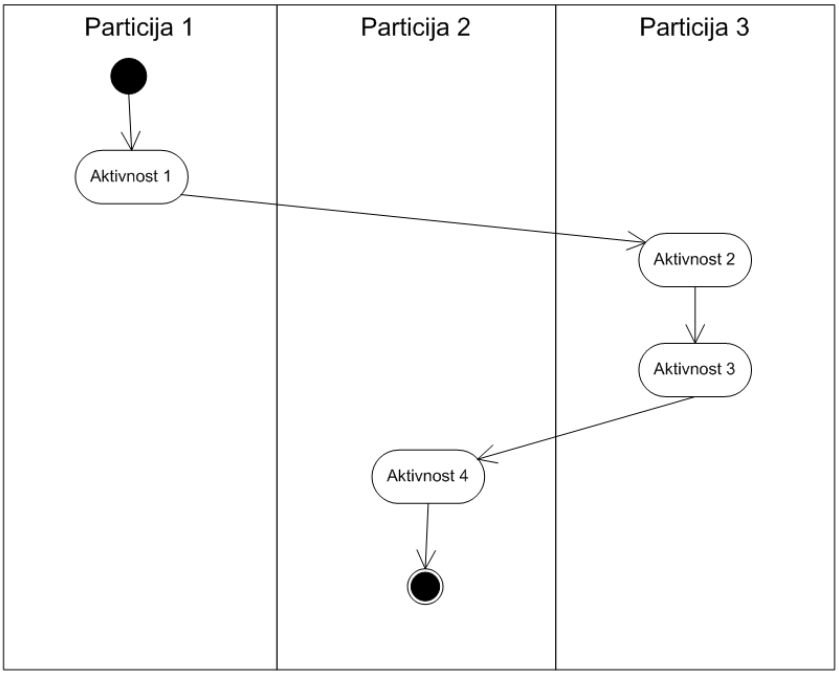
Dijagram aktivnosti sastoji se od sljedećih temeljnih elemenata:

1. Početnog i konačnog stanja, koje se obilježava jednako kao i kod dijagrama stanja.
2. Aktivnosti, koja se obilježava zaobljenim pravokutnikom, jednako kao i stanje kod dijagrama stanja. Jedina razlika je u tome što se unutar neke aktivnosti odvija samo ta aktivnost - nema više aktivnosti unutar nekog stanja kao što je to moguće kod hijerarhije stanja.
3. Prijelaz između aktivnosti, označen strelicom. Jedine dvije komponente prijelaza koje su dozvoljene kod dijagrama aktivnosti su naziv prijelaza i ograničenje.
4. Odluka ili uvjetno grananje, označena bijelim, dijamantnim oblikom isto kao i kod dijagrama stanja.
5. Račvanje i skupljanje, koje se označava isto kao i kod dijagrama stanja.
6. Signal (događaj). Signali se koriste na dijagramu aktivnosti da bi nadomjestili postojanje događaja kod dijagrama stanja. Postoje tri tipa signala: 1) Šaljući signal (engl. *sending signal* ili *generating signal*), 2) Primajući signal (engl. *receiving signal* ili *accepting signal*) i vremenski signal (engl. *timer signal*). Sva tri primjera signala prikazana su na primjeru na slici 6.7.



Slika 6.7. Signali kao elementi dijagrama aktivnosti.

7. Plivačke staze (engl. *swimlanes*), kod kojih su aktivnosti raspoređene u vertikalne i/ili horizontalne particije koje su razgraničene linijama. Same particije ne donose dodatnu semantiku u dijagram osim organizacijske, kada se želi istaknuti koji od dijelova sustava što izvodi [Fowler 2004]. Primjer plivačkih staza prikazan je na slici 6.8. Aktivnost 1 izvodi se u okviru particije broj 1. Zatim se druga i treća aktivnost izvode u okviru particije 3, da bi aktivnost 4 bila posljednja unutar particije 2.



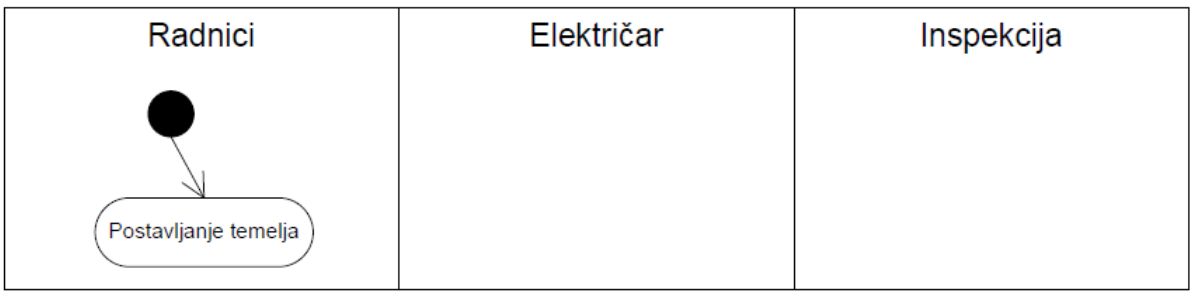
Slika 6.8. Plivačke staze.

6.3.2. Riješeni primjer

Primjer 6.3.2.1.

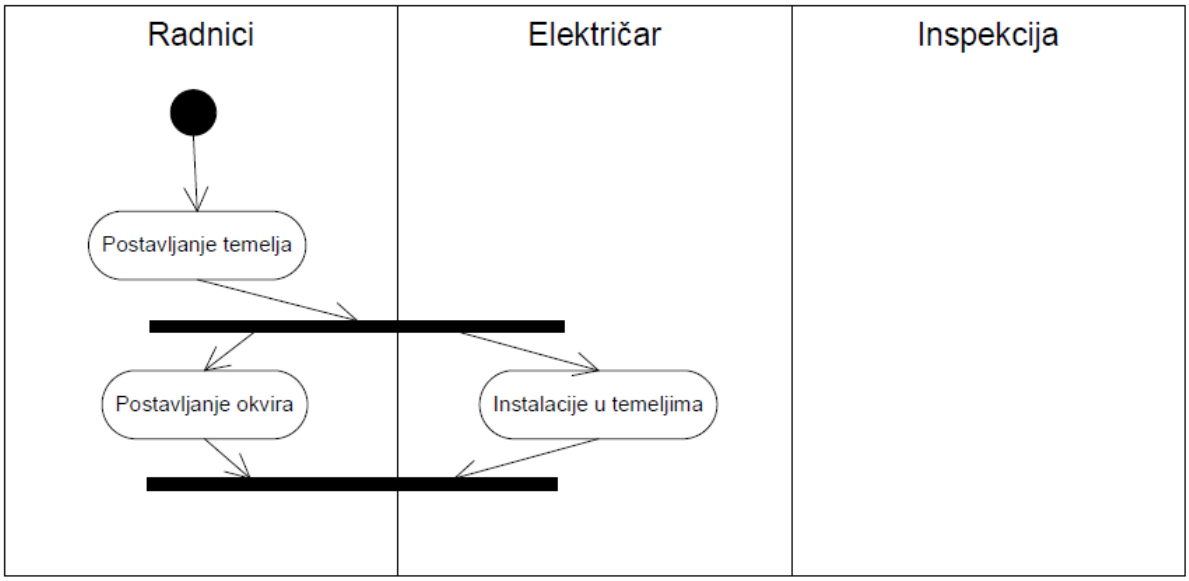
Modelirajte dijagramom aktivnosti izgradnju kuće. Najprije radnici postavljaju temelje kuće. Zatim u paraleli radnici grade okvir (katovi i nosivi zidovi) dok električar postavlja električne instalacije u temeljima. Nakon toga, električar postavlja električne instalacije u nosivim zidovima. Dok jedni radnici postavljaju krov, drugi grade pregradne zidove. Nakon toga električar postavlja ostale električne instalacije kroz cijelu kuću. Zatim radnici provode vanjsko i unutarnje žbukanje. Na kraju radnici postavljaju vrata i prozore. Inspekcija po završetku pregledava stanje kuće.

Rješenje:
Podijelit će se aktivnosti u tri plivačke staze. Jedna je za radnike, druga za električara, a treća za inspekciju. Prva aktivnost koju radnici provode je postavljanje temelja kuće, slika 6.9.



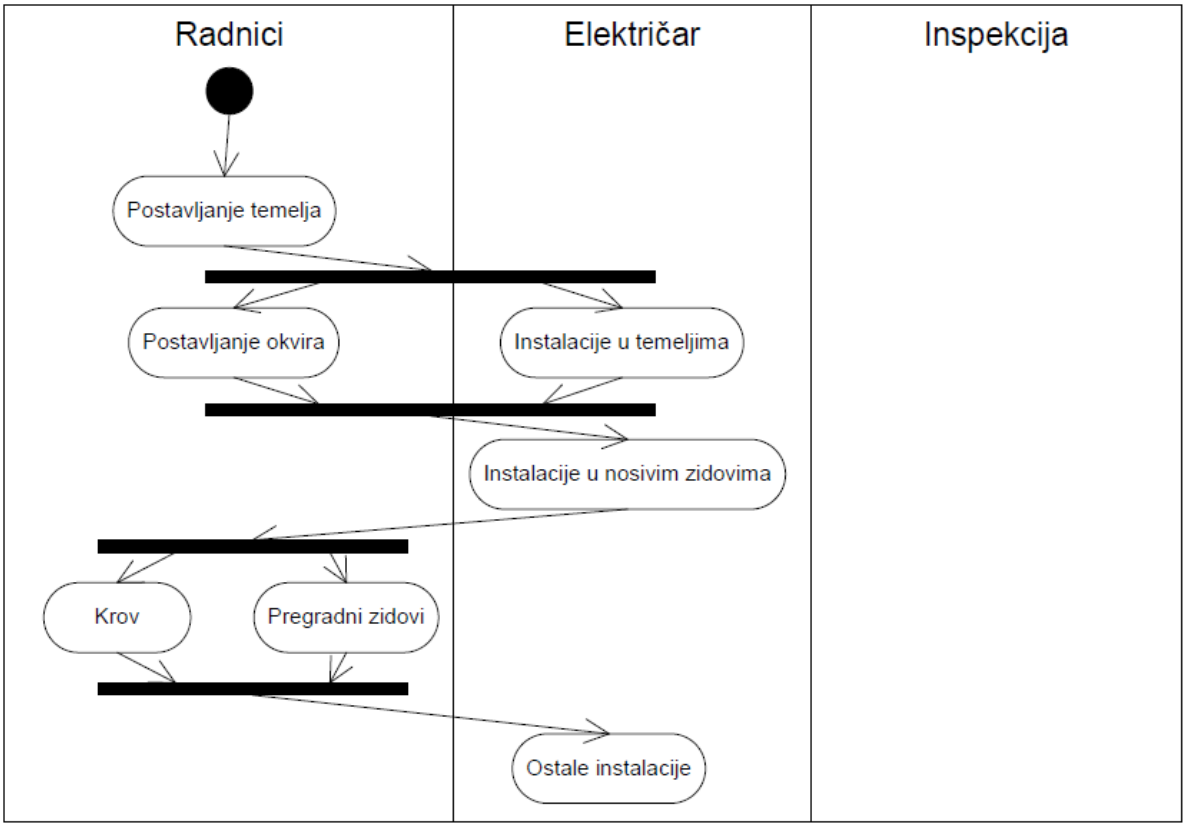
Slika 6.9. Početak dijagrama aktivnosti iz primjera 6.3.2.1.

Nakon toga radnici i električar rade u paraleli na postavljanju okvira i električnim instalacijama u temeljima. Ovdje se koristi račvanje i skupljanje, slika 6.10.



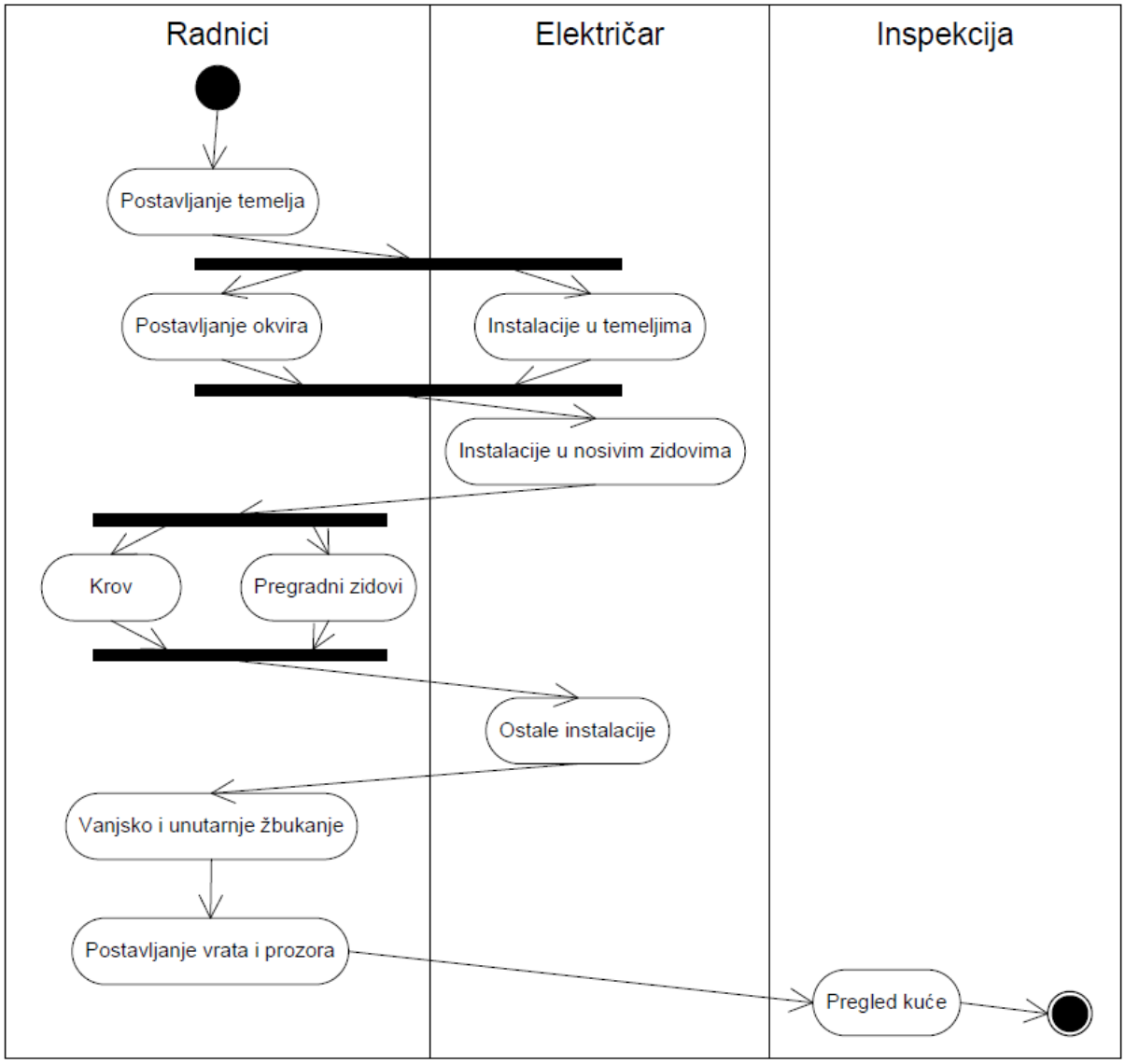
Slika 6.10. Nastavak dijagrama aktivnosti iz primjera 6.3.2.1.

Zatim neko vrijeme električar sam radi svoj posao postavljanjem elektrike u nosivim zidovima. Nakon toga radnici rade u paraleli i postavljaju krov i pregradne zidove, da bi potom električar provukao ostale instalacije kroz cijelu kuću, slika 6.11.



Slika 6.11. Nastavak dijagrama aktivnosti iz primjera 6.3.2.1.

Potom radnici provedu vanjsko i unutarnje žbukanje i postavljanje vrata i prozore. Na kraju inspekcija to sve pregleda i time je dijagram završen, slika 6.12.



Slika 6.11. Završetak dijagrama aktivnosti iz primjera 6.3.2.1.

6.3.3. Zadaci za vježbu

Zadatak 6.3. Kuhanje kave

Modelirajte postupak kuhanja kave u nekom poduzeću dijagramom aktivnosti. Pretpostavite da u dotičnom poduzeću tajnica svako jutra kuha šefu kavu. Slijed aktivnosti je sljedeći. Po dolasku, šef pozdravi tajnicu i zamoli ju za kavu. Nadalje, šef sjedne i čita pristiglu poštu. Za to vrijeme, tajnica najprije provjeri ima li dovoljno kave i šećera u uredu. Ako ima dovoljno kave i šećera, tada stavi vodu da zakuha. U slučaju da nema dovoljno kave ili šećera, tajnica ode do službe nabave poduzeća. Ako služba nabave ima kavu ili šećer, tada joj to daju i ona se vrati nazad u ured i zakuha vodu za kavu. U slučaju da služba nabave nema kavu ili šećer, tada oni naruče da im se ista što prije dostavi. Budući da tajnica zna da šef ne može bez jutarnje kave, ona ostaje kod službe nabave sve dok ne dobije kavu. Čim kavu dobije, tajnica se vraća nazad u ured i zakuha kavu. Ako u međuvremenu prođe više od 45

minuta otkako je šef naručio kavu, šef prestaje čitati poštu i izlazi ljutit iz ureda. Tajnica u svakom slučaju čim ima kavu i šećer i kad voda zakuha pripremi kavu, čak i u slučaju da je šef već izašao. Ako je šef još tamo, šef pije kavu.

Zadatak 6.4. Pohađanje vještine „Osnove programskog jezika Java“

Modelirajte pohađanje vještine „Osnove programskog jezika Java“ dijagramom aktivnosti. Student želi upisati i proći tečaj Jave. Prvi korak je proći prijemni test. Prijemni test ocjenjuje predavač. U slučaju da je student uspio proći prijemni test, student dolazi na sat Jave. Satovi Jave oblikovani su na način da najprije predavač priča, a studenti ga slušaju. Na kraju sata predavač daje studentima domaću zadaću. Student tijekom sljedećeg tjedna napiše domaću zadaću i preda ju predavaču. Zatim u paraleli predavač ocjenjuje domaću zadaću, a student razmišlja je li dovoljno dobro napisao domaću zadaću za prolaz (grize nokte).

Predavač donosi odluku o tome je li zadaća bila dovoljno dobra. Ako je zadovoljio na domaćoj zadaći, postupak se ponavlja i student ponovno dolazi na sat Jave. Ako je student na taj način uspio zadovoljiti na 10 domaćih zadaća, tada je prošao vještinu. U svakom drugom slučaju, smatra se da je student odustao. Napomena: ovaj zadatak je pojednostavljen, u stvarnosti student ima pravo izostati s najviše jednog predavanja i ne napisati najviše jednu zadaću, bez obzira na opravdanja. ☺

7. Dijagrami komponenti

7.1. Karakteristike dijagrama

Dijagrami komponenti prikazuju komponente (strukturne cjeline) sustava i njihove međusobne odnose [Fowler 2000]. Komponenta je zasebna cjelina programske potpore s vlastitim sučeljem. Komponenta predstavlja fizičku i stvarnu implementaciju logičkih elemenata sustava kao što su razredi, sučelja i pridruživanja [Booch 1999]. Komponentni dijagrami pomažu u modeliranju fizičkih cjelina sustava kao što su izvršne datoteke, programske biblioteke, tablice, datoteke i svi drugi dokumenti. Često se kaže da su u UML-u sve fizičke „stvari“ modelirane kao komponente [Booch 1999].

U izradi programske podrške mnogi operacijski sustavi i računalni jezici podržavaju koncept komponente: objektne biblioteke, izvršne datoteke, DCOM i COM+ komponente i Enterprise Java Beans su primjeri komponenata koje se mogu direktno predstaviti u UML-u korištenjem komponenti [Booch 1999]. Jedan razred može biti predstavljen s više komponenata, ali samo s jednim paketom. Na primjer, razred Java *String* je smješten u paket *java.lang*, ali istodobno sastoji se od mnogo komponenata [Fowler 2000].

Dijagrami komponenti su strukturni UML-dijagrami. Prikazuju vremenski nepromjenjiva (statička) svojstva sustava s fizičkog aspekta implementacije. Stoga se uz dijagrame razmještaja još nazivaju fizički dijagrami te se često prikazuju zajedno u jednom UML-dijagramu komponenti i razmještaja.

Primjer 7.1.1. Sistemska DLL datoteka

Prikazati sistemsku datoteku 'gdi32.dll' operacijskog sustava Windows. Datoteka eksportira *Graphics Device Interface* (GDI) programsko sučelje. Zanemariti ostala sučelja komponente.

Rješenje primjera 7.1.1 prikazano jer na slici 7.1.

Slika 7.1. Rješenje primjera 7.1.1.

7.2. Svojstva komponenti

Svaka komponenta mora imati vlastito ime koje je razlikuje od ostalih komponenti u dijagramu [Booch 1999]. Jednostavan naziv (engl. *simple name*) je niz znakova. Naziv puta (engl. *path name*) je ime komponente s prefiksom imena paketa kojemu komponenta pripada. Jednostavne komponente (engl. *simple components*) imaju jednostavne nazive, a proširene komponente (engl. *extended components*) naziv puta ispred jednostavnog imena. U praksi, jednostavni nazivi komponenata uključuju i ekstenziju njihovih datoteka (npr. .java ili .dll).

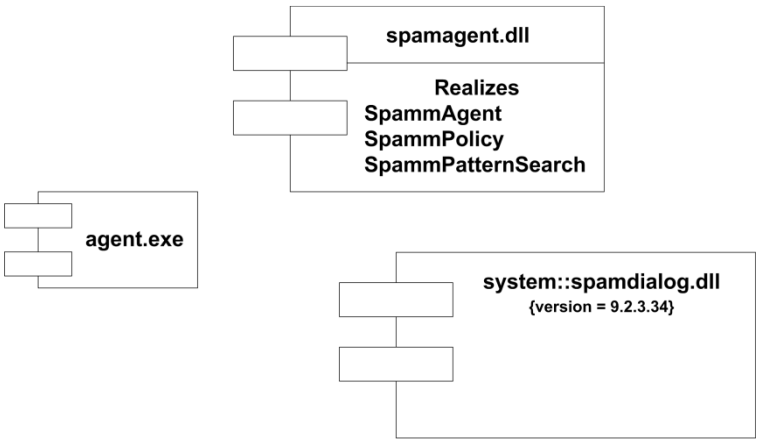
Značajne razlike između komponente i razreda su:

1. Razredi predstavljaju logičku apstrakciju sustava, dok su komponente fizički (stvarni i opipljivi) artefakti.
2. Komponente i razredi pripadaju različitim pogledima na sustav s drugačijim razinama apstrakcije.
3. Razredi imaju vlastite atribute i operacije koje mogu neposredno izvršavati. Komponente imaju pristup samo onim operacijama koje se mogu dosegnuti kroz sučelja.

Primjer 7.2.1. Jednostavne i proširene komponente

Neki informatički sustav za detekciju neželjene pošte (engl. *spam*) sadržava komponente 'agent.exe, 'spamagent.dll' i 'agentdialog.dll'. Datoteka 'spamagent.dll' realizira razrede SpamAgent, SpamPolicy i SpamPatternSearch. Datoteka 'agentdialog.dll' nalazi se u paketu system i u dijagramu je prikazana inačica datoteke 9.2.3.34. Potrebno je prikazati komponente 'agent.exe' i 'spamagent.dll' s jednostavnim nazivom, i 'agentdialog.dll' s proširenim.

Rješenje primjera 7.2.1 prikazano je na slici 7.2.

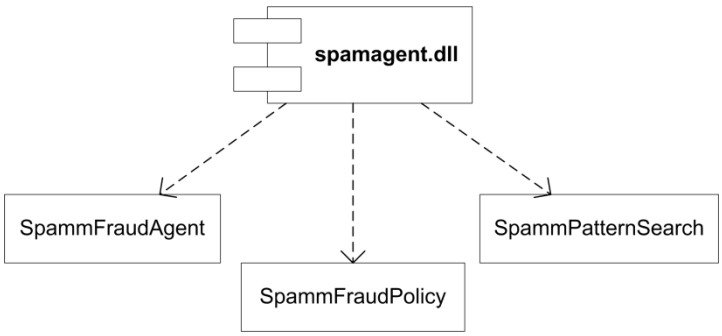


Slika 7.2. Rješenje primjera 7.2.1.

Primjer 7.2.2. Komponente i razredi

U sustavu iz prethodnog primjera ključna DLL-datoteka 'spamagent.dll' sastoji se od tri razreda: SpamAgent, SpamPolicy i SpamPatternSearch. DLL-datoteku implementiraju dodatni razredi, ali oni su manje važni za njezinu funkcionalnost te ih stoga nije potrebno prikazati u dijagramu.

Rješenje primjera 7.2.2 prikazano je na slici 7.3.



Slika 7.3. Rješenje primjera 7.2.2.

7.3. Sučelja komponenti

Sučelje je kolekcija operacija za specificiranje usluge razreda ili komponente. Veza između komponenti i sučelja je vrlo važna. Svi moderni operacijski sustavi koji podržavaju komponente kao što su COM+, CORBA, Enterprise Java Beans koriste sučelja za povezivanje komponenata. Sučelja su „ljepilo“ ili „poveznica“ između odvojenih komponenata unutar istog ili između različitih paketa i čvorova [Booch 1999]. Drugim riječima, sučelja premošćuju logičke i fizičke granice unutar sustava.

Sučelje koje neka komponenta realizira je usluga za druge komponente i naziva se eksportirano sučelje (engl. *export interface*). Jedna komponenta može realizirati više sučelja. Sučelje koje neka komponenta koristi naziva se importirano sučelje (engl. *import interface*). Komponenta može koristiti neograničeni broj importiranih sučelja.

Kod dijagrama komponenti u slučaju norme UML 1.x sučelja mogu biti prikazana na dva načina:

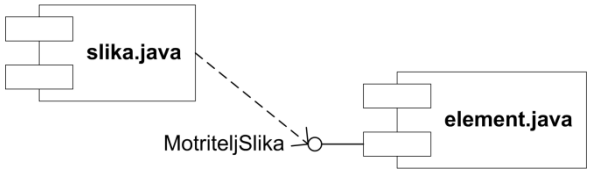
1. Ikonizirani oblik (engl. *iconic form*). Najčešći način prikaza sučelja. Komponenta koja realizira sučelje povezana je sa sučeljem s izostavljenom vezom realizacije (engl. *elided realization relationship*).
2. Prošireni oblik (engl. *expanded form*). Prikaz sučelja pomoću veze realizacije. Izgled ove veza identičan je vezi realizacije u dijagramima razreda, vidjeti poglavlje 4.14. Komponenta koja realizira sučelje povezana je sa sučeljem pomoću veze realizacije tako da strelica dodiruje simbol sučelja iz dijagrama razreda, a drugi dio veze dodiruje komponentu. U ovom obliku atributi i operacije sučelja moraju biti prikazani u dijagramu, dok u ikoniziranom obliku ne moraju.

U oba slučaja komponenta koja importira sučelje povezana je s njim pomoću veze zavisnosti.

Primjer 7.3.1. Sučelje razreda *ImageObserver*

Neki sustav sastoji se od komponente 'slika.java' i 'element.java'. Komponenta 'element.java' ima sučelje MotriteljSlika (engl. *ImageObserver*) s tri operacije prekini(), greska() i osvjeziSliku(). Prve dvije operacije vraćaju cjelobrojnu (Integer) vrijednost, a treća operacija Boolean. Prikazati sustav pomoću ikoniziranog i proširenog oblika dijagrama sučelja.

Prvi dio rješenja primjera 7.3.1 (ikonizirani oblik) prikazan je na slici 7.4, a drugi dio rješenja (prošireni oblik) prikazan je na slici 7.5.



Slika 7.4. Prvi dio rješenja primjera 7.3.1 (ikonizirani oblik).

Slika 7.5. Drugi dio rješenja primjera 7.3.1 (prošireni oblik).

7.4. Vrste komponenti

UML definira tri vrste komponenti:

1. Komponente razmještaja (engl. *deployment components*) su nužne i dovoljne za rad neke računalne aplikacije ili sustava, kao što su dinamički povezane biblioteke (engl. *dynamic linked libraries*, DLL) i izvršne datoteke (engl. *executables*, EXE). UML-definicija komponente razmještaja uključuje i širok raspon drugih objekata poput COM+, CORBA, Enterprise Java Bean, dinamičkih mrežnih stranica, tablica baza podataka i drugih izvršnih datoteka.
2. Komponente radnog proizvoda (engl. *work product component*) su datoteke nastale tijekom razvojnog procesa poput datoteka izvornog koda (engl. *source code files*), datoteka s podacima (engl. *data files*), itd. Komponente radnog proizvoda koriste se za stvaranje drugih komponenti.
3. Komponente izvođenja (engl. *execution components*) nastaju kao posljedica izvođenja i rada sustava. Primjerice, COM+ objekt je takva komponenta jer nastaje izvođenjem pripadne DLL-datoteke.

Komponente se organiziraju i grupiraju u pakete slično kao što se organiziraju razredi. U UML 1.x komponente se mogu organizirati i pomoću veze ovisnosti. Druge vrste pridruživanja (generalizacija, agregacija, proširenja dvosmjernih vezama prototipovima,…) dozvoljene su samo u kasnijim verzijama UML-a [Booch 1999].

7.5. Stereotipovi komponenti

Dijagrami komponenti definiraju pet standardnih stereotipova za komponente:

1.	«executable»	Komponenta može biti izvršena u čvoru (engl. *node*)
2.	«library»	Komponenta je statički ili dinamički objekt biblioteke
3.	«table»	Komponenta je tablica baze podataka
4.	«file»	Komponenta je dokument s izvornim kodom ili podacima
5.	«document»	Komponenta je dokument općeg značenja ili sadržaja

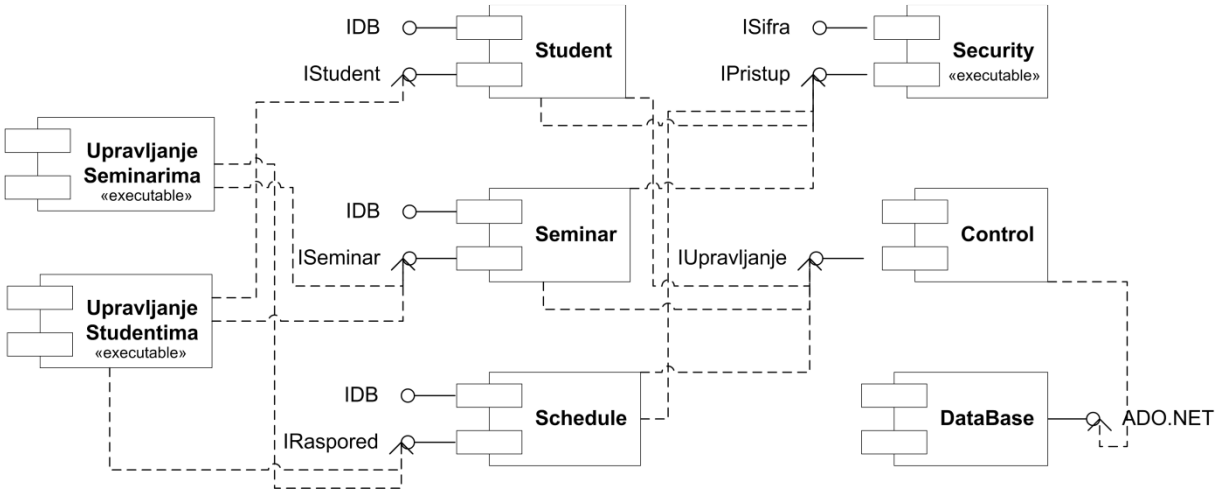
UML ne definira specifične ikone za stereotipove, ali potiče se korištenje u praksi standardiziranih slikovnih simbola za izvršne datoteke, DLL datoteke, tablice baze podataka, itd.

Primjer 7.5.1. Modeliranje sveučilišnog informacijskog sustava

Višeslojni sveučilišni informacijski sustav sastoji se od tri velika podsustava: UpravljanjeSeminarima, UpravljanjeStudentima i Sigurnost, koji su u modelu ekvivalentni izvršnim datotekama. U najvišem sloju sustava nalaze se komponente UpravljanjeSeminarima i UpravljanjeStudentima, u drugom sloju Student, Seminar i Raspored, te u trećem Sigurnost i Upravljanje. U drugom sloju komponenta Student ima sučelja IDB i IStudent, komponenta Seminar sučelja IDB i ISeminar, te komponenta Raspored sučelja IDB i IRaspored. U trećem sloju komponenta Sigurnost ima sučelja ISifra i IPristup, a komponenta Upravljanje sučelje IUpravljanje. Komponenta UpravljanjeSeminarima importira sučelja IRaspored i ISeminar, dok UpravljanjeStudentima importira

ista sučelja i dodatno IStudent. Komponente Student, Seminar i Raspored koriste sučelja IPristup i IUpravljanje. Komponenta Upravljanje pristupa bazi podataka koristeći radni okvir ADO.NET. Prikazati pripadni dijagram komponenti. Rješenje primjera 7.5.1 prikazano je na slici 7.6.

Rješenje:



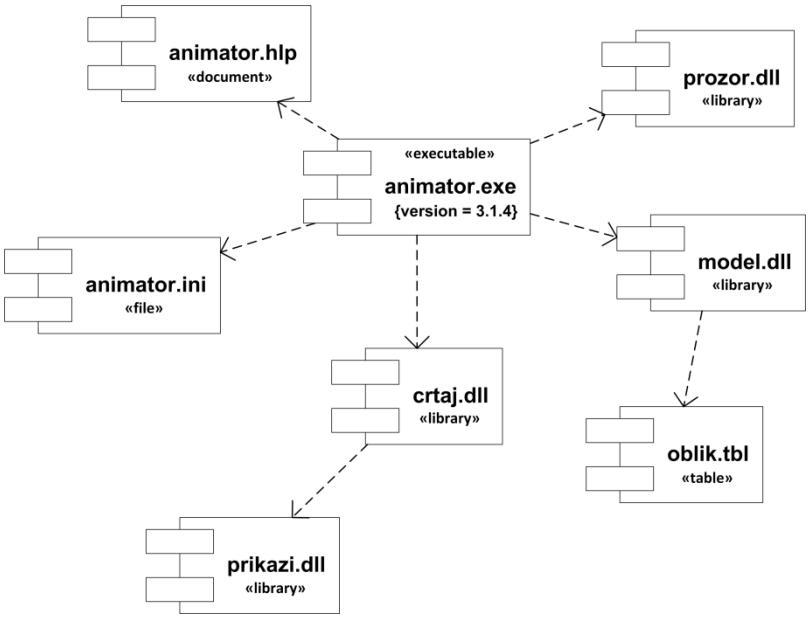
Slika 7.6. Rješenje primjera 7.5.1.

Primjer 7.5.2. Modeliranje tablica, datoteka i dokumenata

Nakon instalacije, Windows aplikacija za računalnu animaciju „Animator“ sastoji se od izvršne datoteke 'animator.exe' koja koristi DLL datoteke 'prozor.dll', 'model.dll', 'crtaj.dll' i 'prikazi.dll'. Konfiguracija aplikacije nalazi se u formatiranoj datoteci 'animator.ini', a sustav pomoći u binarnoj datoteci 'animator.hlp'. Izvršna datoteka 'animator.exe' je u inačici 3.1.4. Biblioteka 'model.dll' pohranjuje podatke u tablicu baze podataka pod nazivom 'Oblik', koja se sastoji od datoteke 'oblik.tbl'. Biblioteka 'crtaj.dll' realizira 'prikazi.dll'. Prikažite navedene komponente u dijagramu.

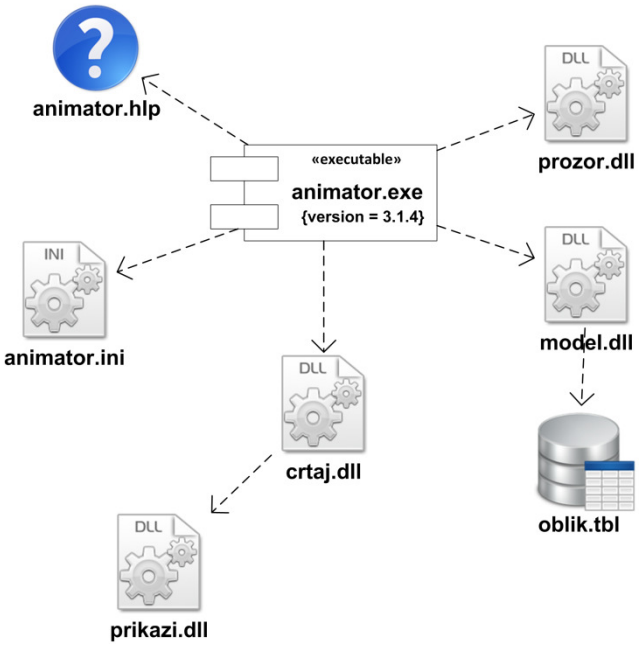
Rješenje:

Koristit će se stereotipovi komponenti i standardni simboli za prikaz svih navedenih datoteka. Jedina veza u dijagramu je ovisnost. Rješenje primjera 7.5.2 prikazano je na slici 7.7.



Slika 7.7. Rješenje primjera 7.5.2.

Zbog jasnoće dijagrama često se koriste normirane ikone umjesto simbola UML-komponenti. Ako je dijagram složen, s puno komponenti i veza, upotreba ikona pridonosi jednostavnijem i bržem razumijevanju dijagrama, pogotovo kod osoba koje nisu stručne u UML-u. Moguće rješenje primjera 7.5.2 korištenjem normiranih ikona prikazano je na slici 7.8.



Slika 7.8. Moguće rješenje primjera 7.5.2 korištenjem normiranih ikona.

7.6. Zadaci za vježbu

Zadatak 7.1.

Neki projekt računalne grafike u programskom jeziku C++ sastoji se od nekoliko datoteka izvornog koda i zaglavlja: 'crtaj.h' (inačica 1.9), 'crtaj.cpp' (inačica 2.6.9), 'jezgra.h' (inačica 1.8), 'polinom.h' (inačica 4.5) i 'boja.h' (inačica 7.3). Datoteka 'crtaj.h' ovisi o svim ostalim datotekama. Prikazati pripadni dijagram komponenti.

Zadatak 7.2.

Aplikacija za animaciju ima vlastiti API - radni okvir koji definira četiri sučelja: ISkripte, ICrtanje, IModeli i IAplikacija. Izvršna datoteka aplikacije je 'animator.exe' u inačici 9.2.8. Prikazati dijagram komponenti.

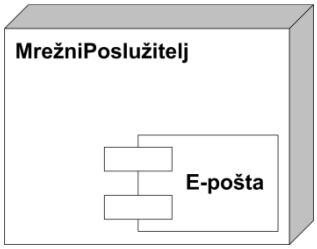
8. Dijagrami razmještaja

8.1. Karakteristike dijagrama

Dijagrami razmještaja (engl. *deployment diagrams*) opisuju topologiju sklopovlja i programsku potporu koja se koristi u implementaciji sustava u njegovom radnom i produkcijskom okruženju. Drugim riječima, dijagrami razmještaja prikazuju računalne resurse koji su neophodni za ispravno funkcioniranje sustava i njihove međusobne odnose: stvarne uređaje (poslužitelje, radne stanice, korisnička računala, itd.), komponente programske podrške koje se na njima izvršavaju i veze između prikazanih resursa. Dijagrami razmještaja, kao i dijagrami paketa i razreda, su statički i strukturni UML-dijagrami.

Primjer 8.1.1. Mrežni poslužitelj s komponentom

Prikažite u dijagramu razmještaja mrežni poslužitelj s komponentom za slanje elektroničke pošte. Rješenje primjera 8.1.1 prikazano je na slici 8.1.



Slika 8.1. Rješenje primjera 8.1.1.

8.2. Elementi dijagrama

Po specifikaciji UML 1.1 dijagrami razmještaja sastoje se od čvorova (engl. *nodes*), komponenti (engl. *components*) i njihovih međusobnih veza.

Između čvorova i komponenti postoje dvije velike razlike:

1. Komponente sudjeluju u izvršavanju sustava, a čvorovi izvršavaju komponente. Čvorovi su računalni resursi koji omogućuju pokretanje i izvršavanje komponenti.
2. Čvorovi predstavljaju fizički aspekt sustava, a komponente logički.

Čvorovi predstavljaju sklopovlje sustava kao što su razni poslužitelji, dok su komponente izvršne datoteke, stolne, mobilne i mrežne aplikacije koje čine opisani sustav. Jedan čvor može simbolizirati više računala kao što je grozd poslužitelja. Čvorovi su predstavljeni u obliku kocki, a komponente vlastitim posebnim simbolom. Zbog veće informativnosti dijagrama unutar komponenti mogu biti naznačeni objekti koji se pomoću njih realiziraju.

Svaki čvor mora imati jedinstveno ime koje ga razlikuje od ostalih čvorova u istom dijagramu razmještaja. Jednostavan naziv (engl. *simple name*) je samo ime čvora, a naziv putanje (engl. *path name*) je naziv paketa kojemu taj čvor pripada (npr. „server::backup“). Naziv putanje uvijek prethodi (prefiks) imenu čvora. Oblik imena čvora s nazivom putanje je prošireni naziv (engl. *extended name*),

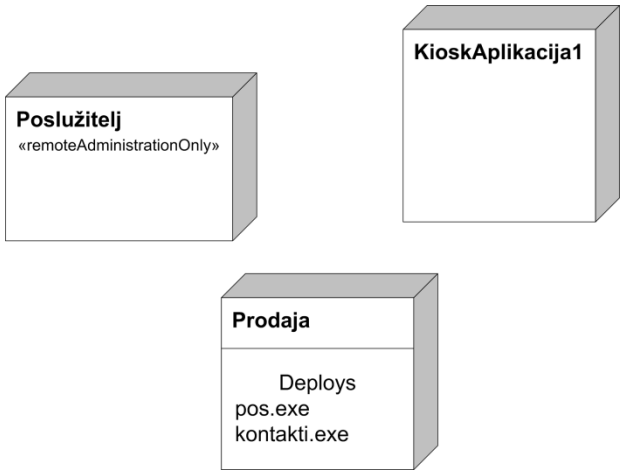
a takvi čvorovi – koji pripadaju drugom paketu – nazivaju se prošireni čvorovi (engl. *extended nodes*). Čvorovi s jednostavnim nazivom nazivaju se jednostavni čvorovi (engl. *simple nodes*). Dobra je praksa nazvati čvorove nazivom poslužitelja ili računala (npr. „IBM zEnterprise 196“, „PRIMERGY RX200 S6“, „Stolno računalo“, „Pametni mobilni telefon Android“), generičkim nazivom koji opisuje njegovu funkciju (npr. „Poslužitelj baze podataka“, „Mrežni poslužitelj“, „Mobilni klijent“), ili po operacijskom sustavu koji se na računalu izvršava (npr. poslužitelj „FreeBSD“).

Primjer 8.2.1. Jednostavni i prošireni čvorovi

Čvorovi nekog sustava za naplatu su: kiosk aplikacija, prodaja i pohrana podataka (engl. *backup*). Podsustav pohrane podataka izvršava se na poslužitelju iz drugog paketa i s njim se može upravljati samo putem udaljenog pristupa (engl. *remote administration*), a ne neposredno iz paketa s čvorovima kiosk i prodaja. Prikazati čvorove s jednostavnim i proširenim imenima.

Rješenje:

KioskAplikacija1 je pojedinac (instanca) kiosk aplikacije. Prodaja predstavlja istoimeni podsustav koji razmješta dvije komponente: 'pos.exe' i 'kontakti.exe'. Osim tekstualno, komponente su se mogle predstaviti i njihovim simbolima. Poslužitelju se može pristupiti samo udaljeno, pa se dijagram proširuje s posebnom oznakom «remoteAdministrationOnly» kako bi se to ograničenje jasno naznačilo. Rješenje primjera 8.2.1 prikazano je na slici 8.2.



Slika 8.2. Rješenje primjera 8.2.1.

8.3. Veze čvorova

Čvorovi su međusobno povezani jednosmjernom ili puno češće dvosmjernom vezom koja označava put prijenosa podataka između čvorova. Dobra praksa je označiti vezu komunikacijskim protokolom ili tehnologijom pomoću koje se odvija prijenos informacija, npr. „TCP/IP“, „HTTP“, „RMI“, „.NET Remoting“, itd.

Čvorovi i komponente su često međusobno ovisni. Pri tome komponenta može biti ovisna o čvoru, ali i obrnuto. Smjer veze ovisnosti uvijek je usmjerena od elementa koji generira informaciju ili poruku, isporučitelja (engl. *supplier*) prema klijentu (engl. *client*) koji je konzumira, odnosno ovisi o

toj poruci. Komponente koje međusobno ovise jedna o drugoj, unutar istog čvora ili između više različitih čvorova, povezane su usmjerenom vezom ovisnosti (engl. *dependency*).

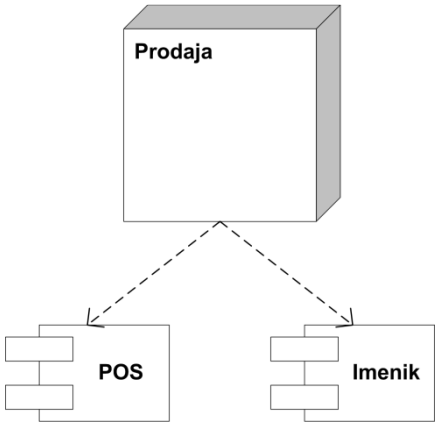
Ako komponenta ima definirano aplikativno sučelje (npr. koristeći port TCP/IP ili programsku fasadu) veza dodiruje simbol sučelja komponente (engl. *component interface*).

Primjer 8.3.1. Ovisnosti komponenti podsustava za naplatu

Neki podsustav za naplatu sastoji se od dvije komponente: mjesto prodaje (POS) i imenik. Podsustav ovisi o komponentama. Prikazati navedene ovisnosti u dijagramu razmještaja.

Rješenje:

Ovisnost komponenti o čvoru kojemu pripadaju se podrazumijeva, ali svejedno može se dodatno naglasiti u dijagramu. U rješenju primjera Prodaja je klijent, a komponente POS i Imenik su isporučitelji. Rješenje primjera 8.3.1 prikazano je na slici 8.3.



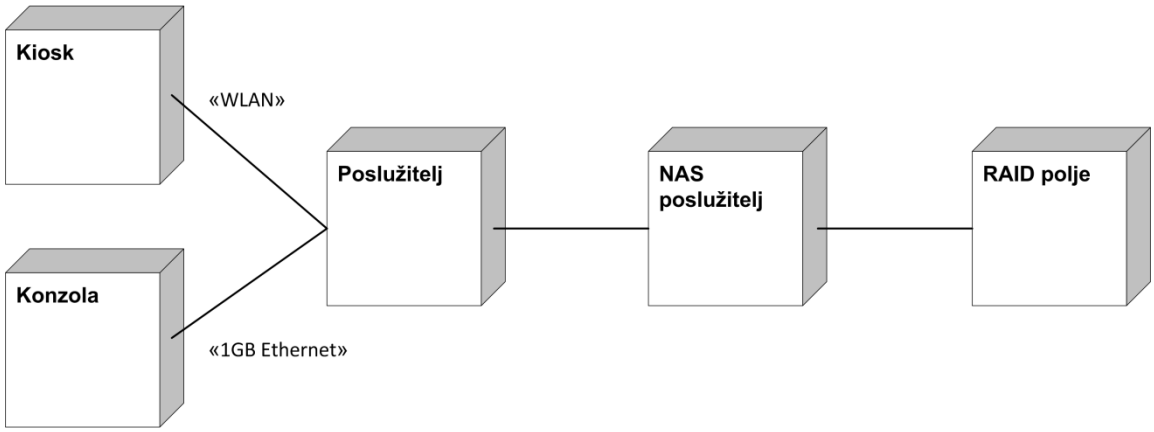
Slika 8.3. Rješenje primjera 8.3.1.

Primjer 8.3.2. Veze između čvorova

Neki sustav sastoji se od nekoliko čvorova: kiosk-aplikacija, konzola, poslužitelj, poslužitelj za sigurnosno spremanje podataka (engl. *NAS storage*) i RAID-polje tvrdih diskova. Kiosk je bežično povezan s poslužiteljem putem WLAN-a, a konzola je povezana pomoću 1 GB Etherneta. Poslužitelj pohranjuje podatke u polje čvrstih diskova pomoću NAS poslužitelja.

Rješenje:

Potrebno je nacrtati zadane čvorove i povezati ih na opisani način. Zbog veće informativnosti dijagrama korisno je svaku vezu dodatno opisati. Rješenje primjera 8.3.2 prikazano je na slici 8.4.



Slika 8.4. Rješenje primjera 8.3.2.

Primjer 8.3.3. Sučelje komponente

Neki poslužitelj sastoji se od više komponenti. Jedna komponenta je poslužiteljska aplikacija koja ima javno izloženo sučelje za konfiguraciju poslužitelja.

Rješenje primjera 8.3.3 prikazano je na slici 8.5.

Slika 8.5. Rješenje primjera 8.3.3.

8.4. Stereotipovi

UML-dijagrame moguće je proširiti korištenjem stereotipova. U dijagramima razmještaja prošrivi su samo čvorovi (komponente se ne mogu proširivati). To svojstvo može se iskoristiti za označavanje namjene čvorova. Standardni stereotipovi čvorova su:

1. «processor» Čvor koji ima obrađuje podatke i izvršava komponente
2. «device» Čvor koji ne može obrađivati podatke, ali predstavlja sklopovlje, neki uređaj ili sučelje prema fizičkom svijetu.

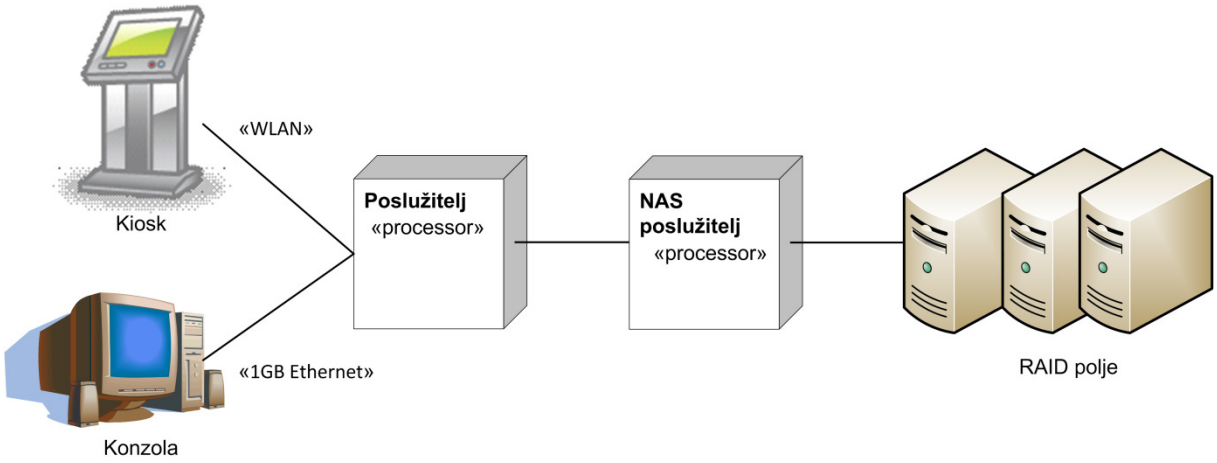
Također, u dijagramima razmještaja umjesto standardnog simbola čvora dozvoljeno je koristiti različite slike ili ikone karakteristične za čvor određene namjene. Slikovni znakovi pružaju učinkovitije razumijevanje namjene čvorova, pogotovo u opširnim dijagramima s velikim brojem čvorova. Učestalo korištenje istog simbola za procesore i uređaje modeliranog sustava različite namjene doprinosi nejasnoći dijagrama.

Primjer 8.4.1. Modeliranje topologije sustava

Prikazati topologiju sustava iz prethodnog primjera korištenjem uobičajene ikonografije za čvorove kiosk, konzola i farme diskova.

Rješenje:

Odabiremo pogodne ikone za naznačene čvorove koje će doprinijeti izražajnosti dijagrama razmještaja. Rješenje primjera 8.4.1 prikazano je na slici 8.6.



Slika 8.6. Rješenje primjera 8.4.1.

8.5. Pojedinci čvorova i komponenti

Dijagram razmještaja može prikazivati pojedince čvorova i komponenti. U tom slučaju dijagram razmještaja postaje sličan dijagramu objekata. Pojedince raspoznajemo po podcrtanim nazivima.

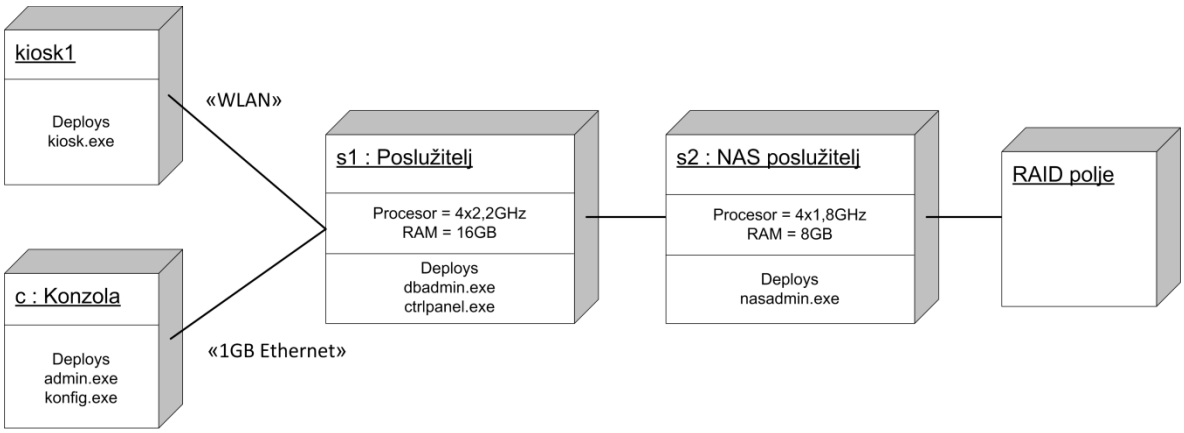
U jednom dijagramu razmještaja nije dozvoljeno miješati definicije i pojedince čvorova ili komponenti, već je moguće prikazati samo definicije, ili samo pojedince.

Pojedince čvorova je korisno označiti najvažnijim podacima iz specifikacije računala i radne okoline, primjerice „Brzina procesora=3GHz, Radna memorija=16GB, Čvrsti disk=2TB, Verzija=3.1.4“.

Primjer 8.5.1. Modeliranje topologije sustava pomoću pojedinaca čvorova i komponenti

Sustav iz prethodnog primjera potrebno je konkretizirati sa specifičnim sklopovljem. Poslužitelj ima procesor s četiri jezgre brzine 2.2 GHz i 16 GB RAM-a. Poslužitelj razmješta dvije komponente: 'dbadmin.exe' i 'ctrlpanel.exe' koje čine aplikaciju za administratora baze podataka, odnosno aplikaciju za upravljanje postavkama operacijskog sustava poslužitelja Na NAS poslužitelju će se nalaziti upravljačka aplikacija 'nasadmin.exe'. Na kiosku se razmješta aplikacija 'kiosk.exe'. Na konzoli će se nalaziti komponente 'admin.exe' za upravljanje i 'konfig.exe' za promjenu postavki konzole.

Rješenje primjera 8.5.1 prikazano je na slici 8.7.



Slika 8.7. Rješenje primjera 8.5.1.

8.6. Zadaci za vježbu

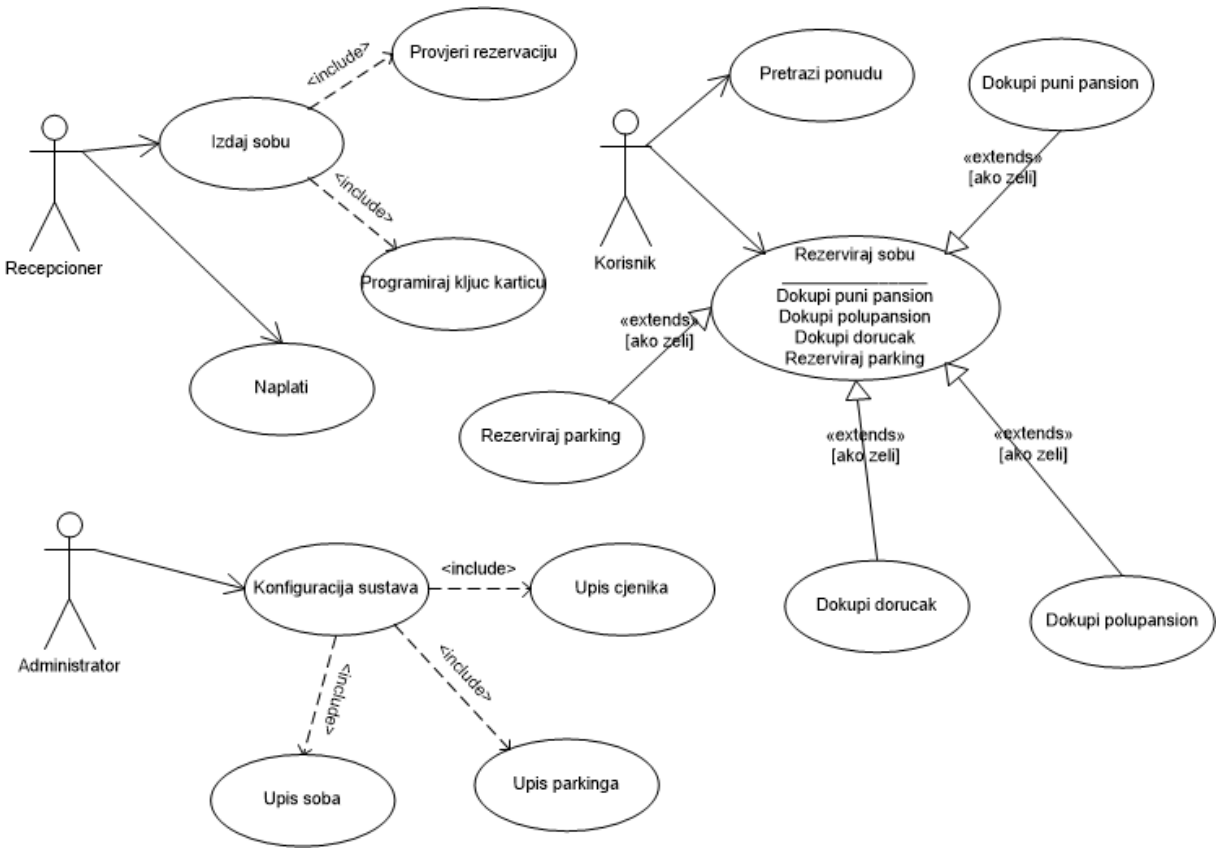
Zadatak 8.1.

Neki raspodijeljeni dvoslojni informatički sustav sastoji se od korisničke aplikacije koja se preko Interneta povezuje na serversku aplikaciju. Korisnička aplikacija izvršava se na korisničkom računalu i spaja se na Internet preko ADSL-modema. Poslužiteljska aplikacija izvršava se na grozdu tri poslužitelja kojima upravlja poslužitelj privremene memorije (engl. *caching server*). Modelirati razmještaj opisanog sustava.

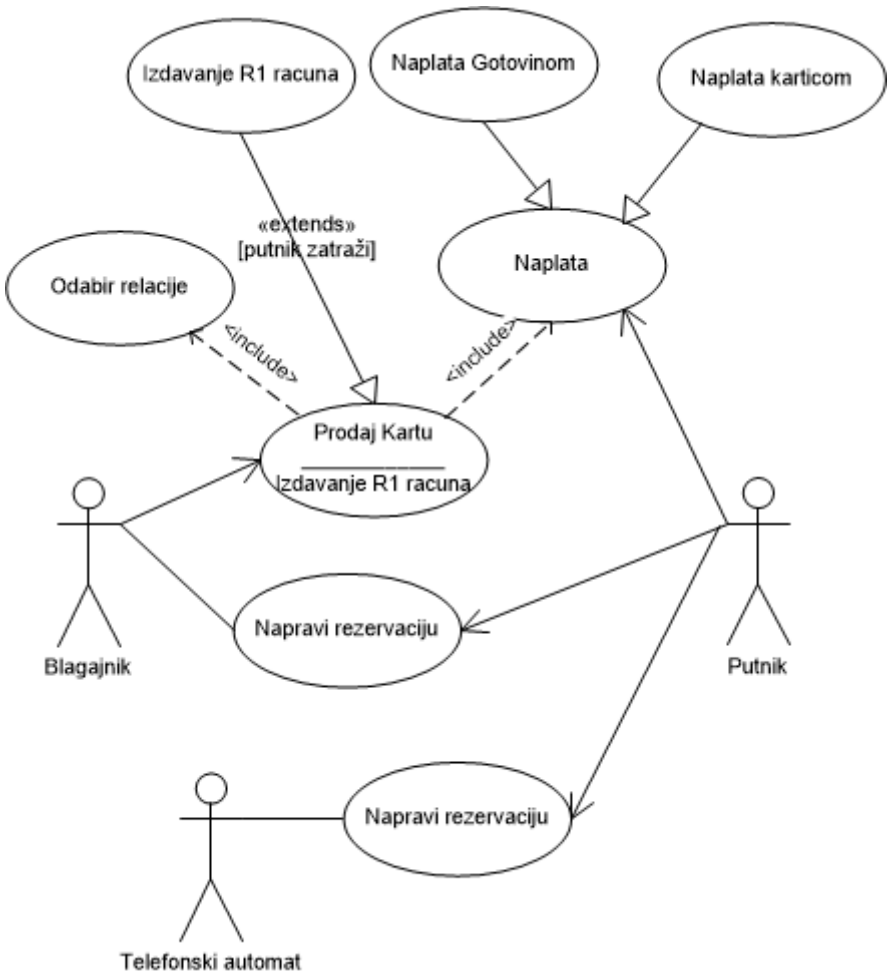
9. Rješenja zadataka

9.1. Dijagrami obrazaca uporabe

Zadatak 2.1.



Zadatak 2.2.


Izdavanje R1 racuna
Naplata Gotovinom
Naplata karticom
«extends»
[putnik zatraži]
Naplata
Odabir relacije
<include>
<include>
Prodaj Kartu
Izdavanje R1 racuna
Napravi rezervaciju
Blagajnik
Putnik
Napravi rezervaciju
Telefonski automat

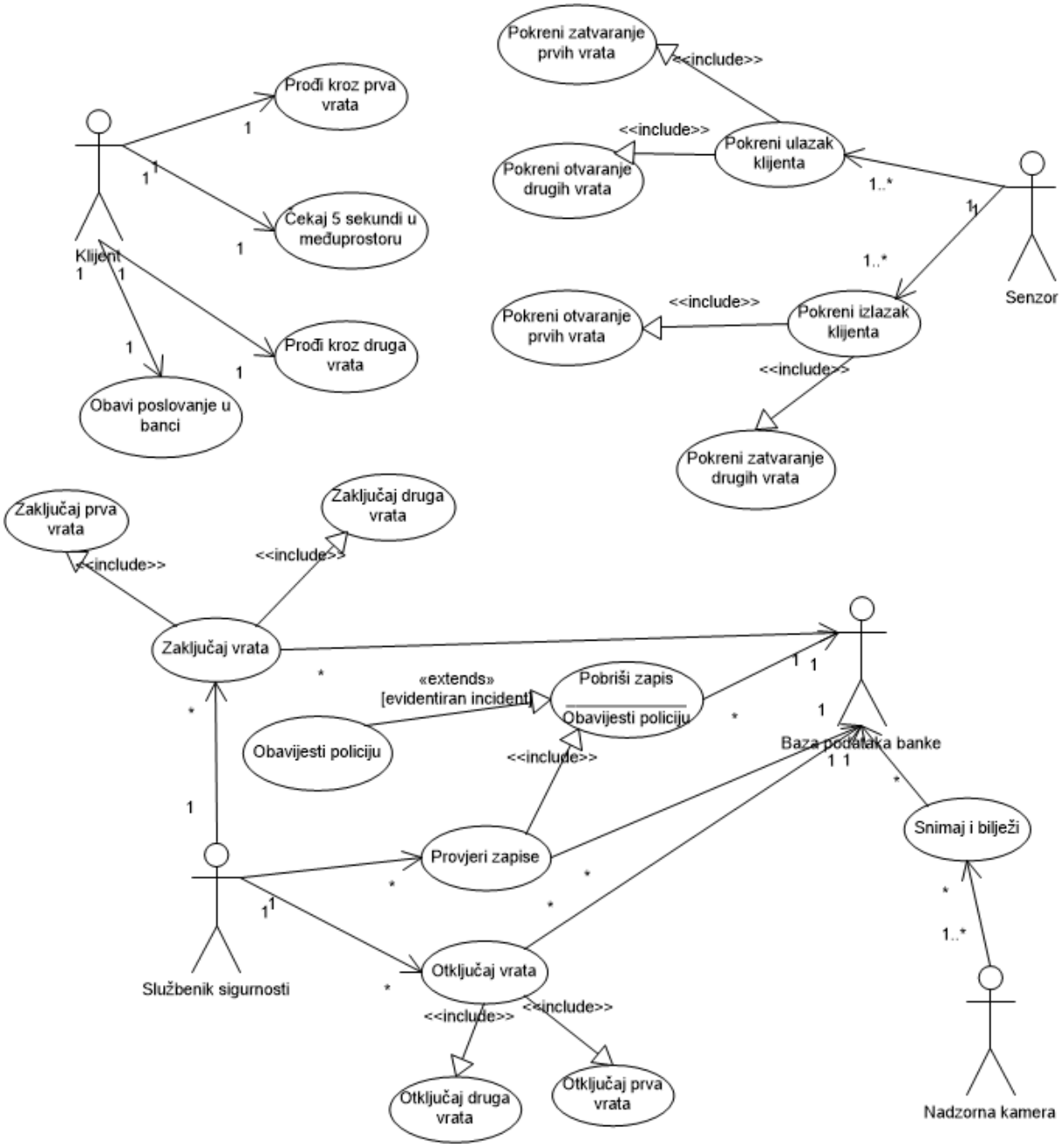
Zadatak 2.3.


Brisanje posla
Slanje podataka na grozd
Učitavanje podataka s grozda
<include>
Dodavanje korisnika
Promjena parametara posla
Brisanje korisnika
<include>
Korisnik
Dodavanje novog posla
Promjena korisničkih podataka
Administrator
<include>
Konfiguracija sustava
Podešavanje postavki raspoređivača
<include>
Izrada sigurnosne kopije
Kopiranje podataka

Zadatak 2.4.

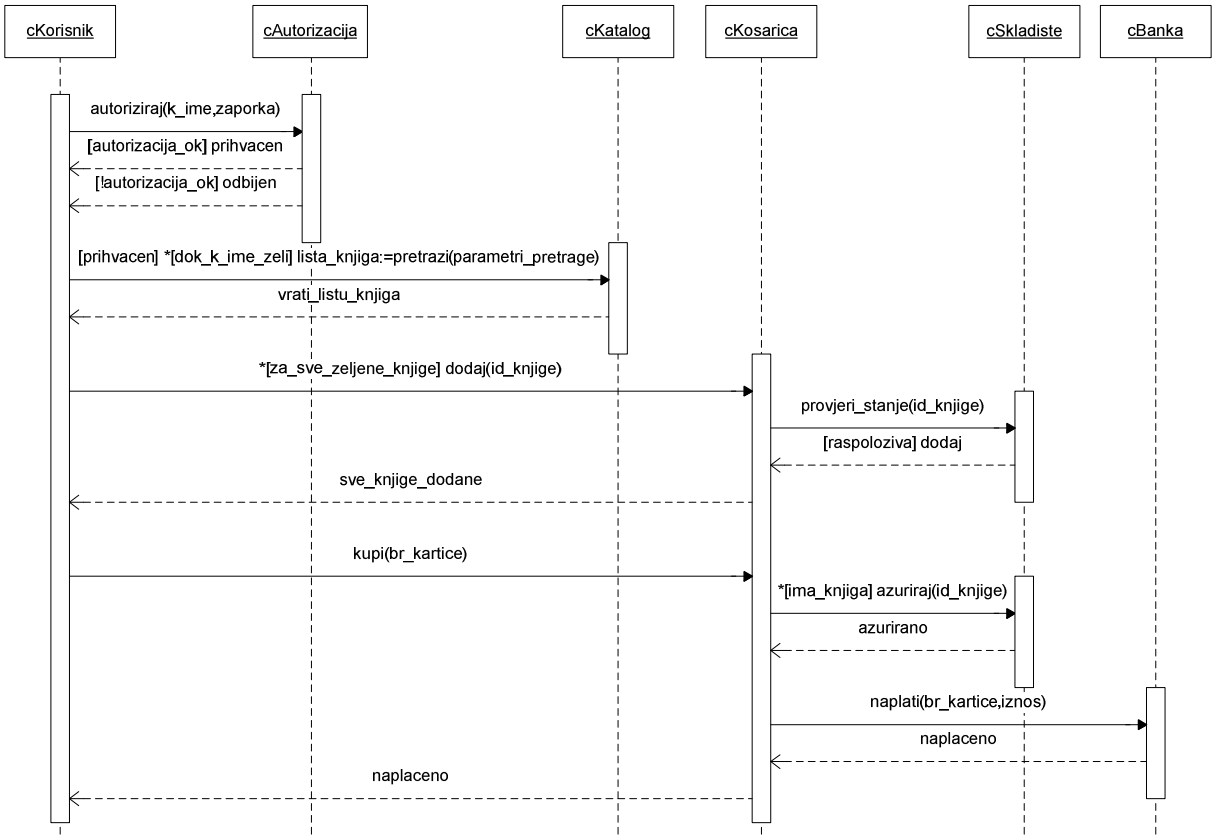

Vrati dokumente/ključeve
Provjeri automobil/gorivo
Napisati
<include>
<include>
<include>
Povrat automobila
Otkazivanje rezervacije
Unesi br. kred. kartice
<include>
Dodaj automobil
Napravi rezervaciju
Rezerviraj dodatak
Rashoduj automobil
Zaposlenik
Korisnik
«extends»
[raspoloživi resursi]
Izdaj automobil
<include>
Predaj dokumente/ključeve
<include>
<include>
Skeniraj obrazac
Rezerviraj dodatak
<include>
Ispiši obrazac
<include>
Potpiši obrazac
Odlazak do automobila

Zadatak 2.5.

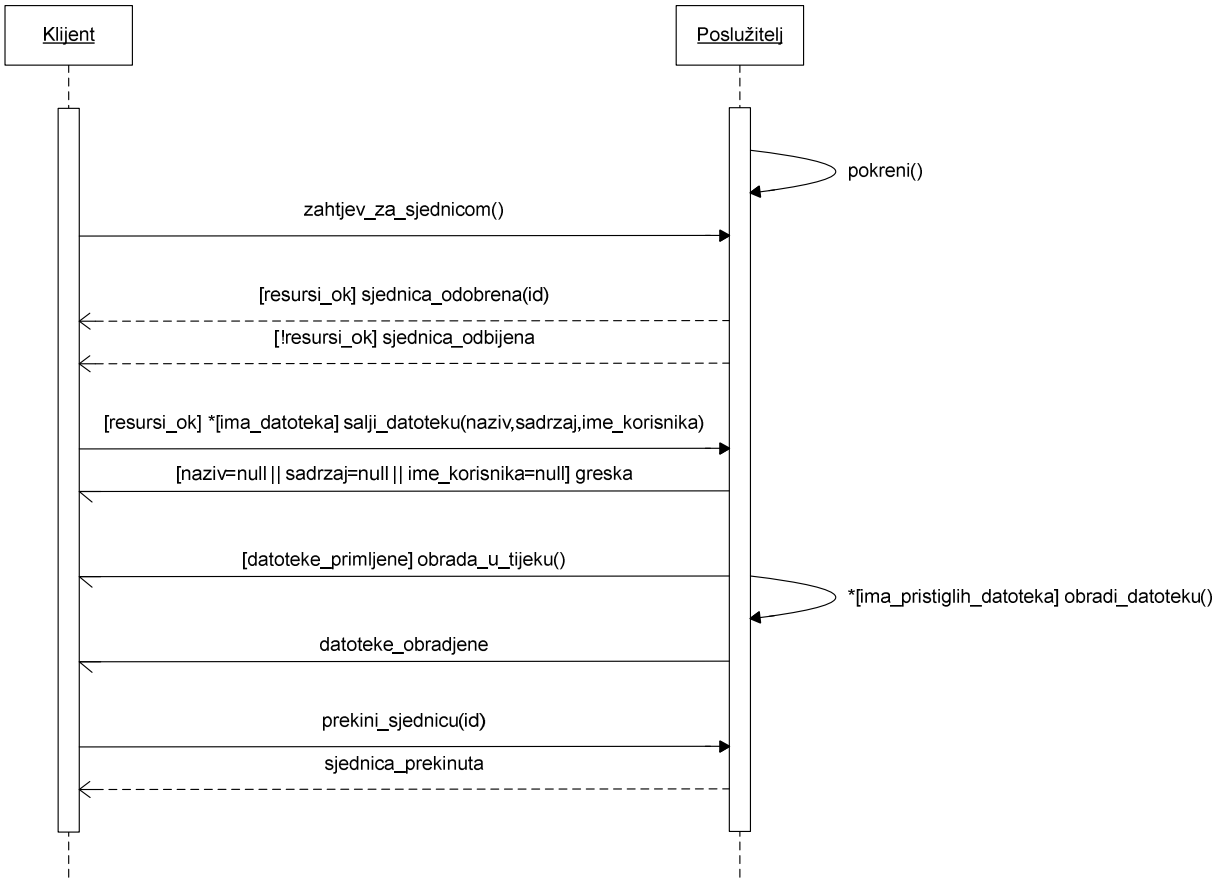

Pokreni zatvaranje prvih vrata
<<include>>
Prođi kroz prva vrata
<<include>>
Pokreni ulazak klijenta
Pokreni otvaranje drugih vrata
1..*
1
1
Senzor
Čekaj 5 sekundi u međuprostoru
1..*
Klijent
<<include>>
Pokreni izlazak klijenta
Pokreni otvaranje prvih vrata
Prođi kroz druga vrata
<<include>>
Obavi poslovanje u banci
Pokreni zatvaranje drugih vrata
Zaključaj prva vrata
Zaključaj druga vrata
<<include>>
<<include>>
Zaključaj vrata
«extends»
[evidentiran incident]
Pobriši zapis
Obavijesti policiju
Obavijesti policiju
<<include>>
Baza podataka banke
Snimaj i bilježi
Provjeri zapise
Službenik sigurnosti
Otključaj vrata
<<include>>
<<include>>
Otključaj druga vrata
Otključaj prva vrata
Nadzorna kamera

9.2. Sekvencijski i komunikacijski dijagrami

Zadatak 3.1.

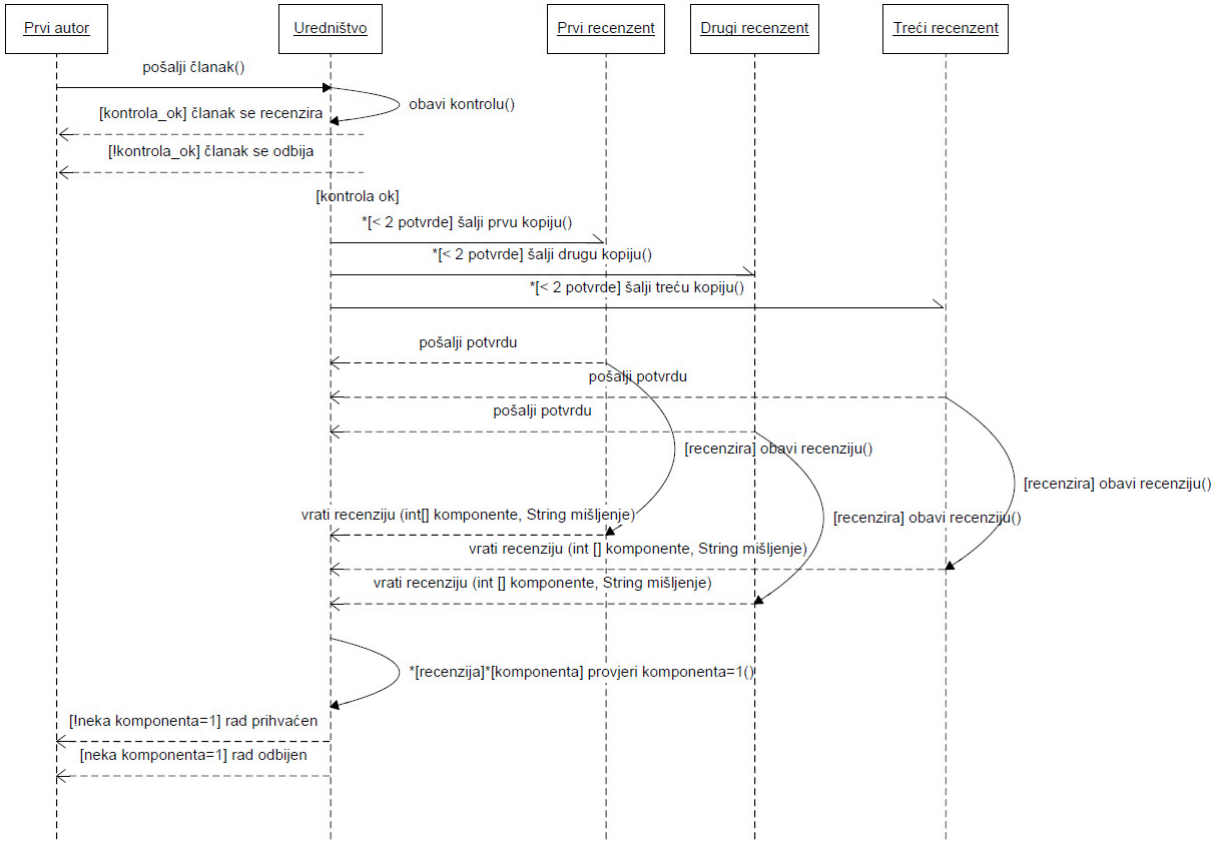


Zadatak 3.2.

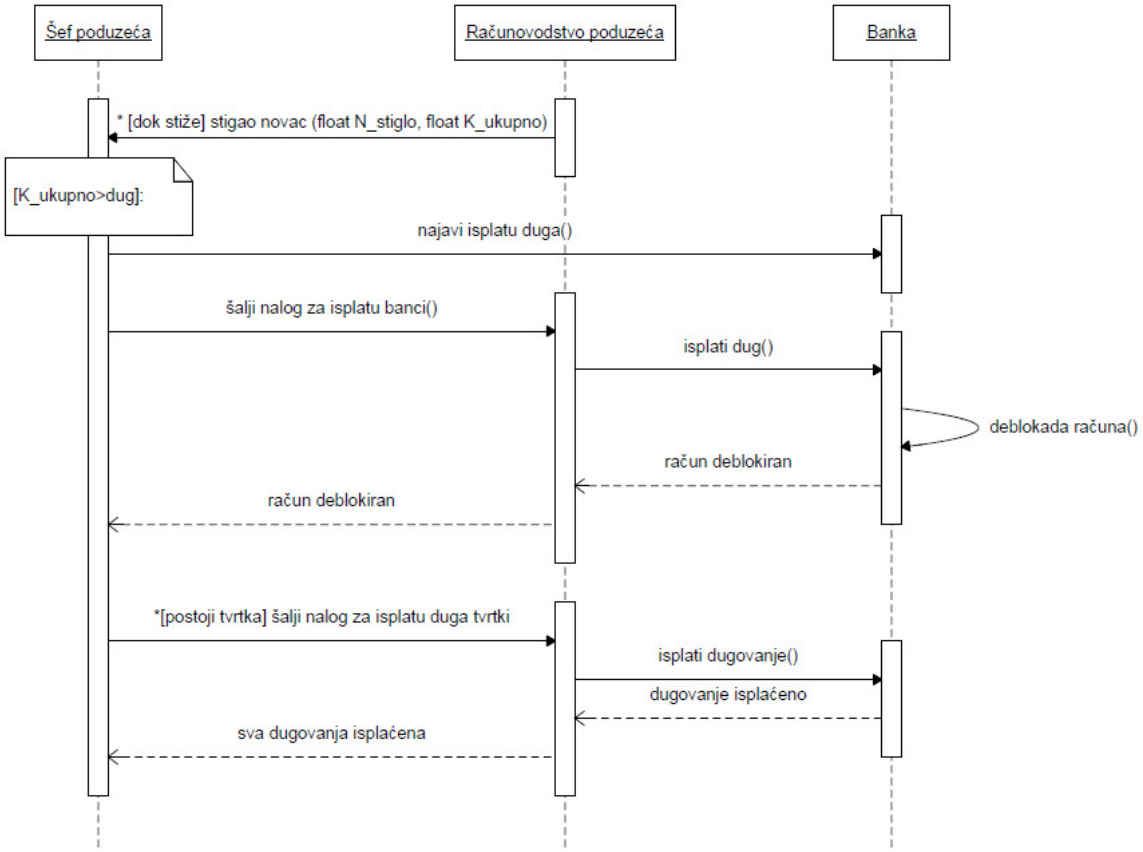

Klijent
Poslužitelj
pokreni()
zahtjev_za_sjednicom()
[resursi_ok] sjednica_odobrena(id)
[!resursi_ok] sjednica_odbijena
[resursi_ok] *[ima_datoteka] salji_datoteku(naziv,sadrzaj,ime_korisnika)
[naziv=null || sadrzaj=null || ime_korisnika=null] greska
[datoteke_primljene] obrada_u_tijeku()
*[ima_pristiglih_datoteka] obradi_datoteku()
datoteke_obradjene
prekini_sjednicu(id)
sjednica_prekinuta

.

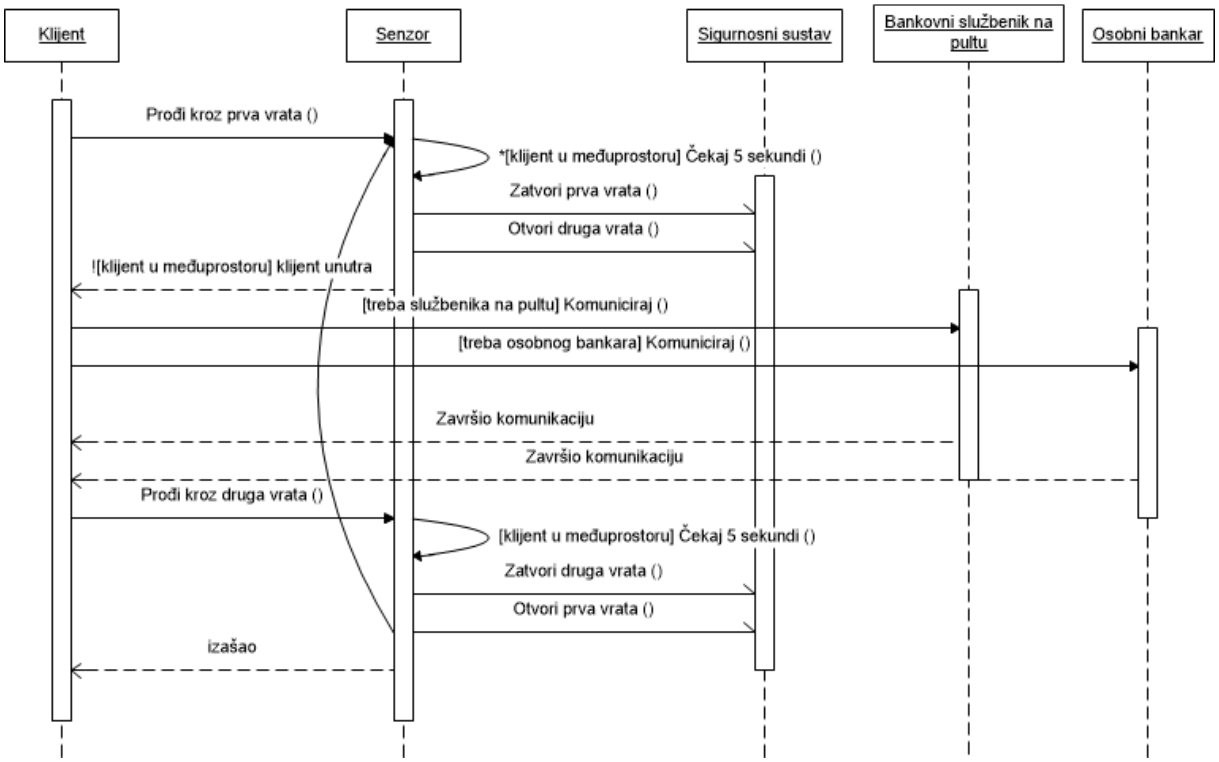
Zadatak 3.3.


Prvi autor
Uredništvo
Prvi recenzent
Drugi recenzent
Treći recenzent
pošalji članak()
obavi kontrolu()
[kontrola_ok] članak se recenzira
[!kontrola_ok] članak se odbija
[kontrola ok]
*[< 2 potvrde] šalji prvu kopiju()
*[< 2 potvrde] šalji drugu kopiju()
*[< 2 potvrde] šalji treću kopiju()
pošalji potvrdu
pošalji potvrdu
pošalji potvrdu
[recenzira] obavi recenziju()
[recenzira] obavi recenziju()
vrati recenziju (int[] komponente, String mišljenje)
[recenzira] obavi recenziju()
vrati recenziju (int [] komponente, String mišljenje)
vrati recenziju (int [] komponente, String mišljenje)
[recenzija][komponenta] provjeri komponenta=1()
[!neka komponenta=1] rad prihvaćen
[neka komponenta=1] rad odbijen

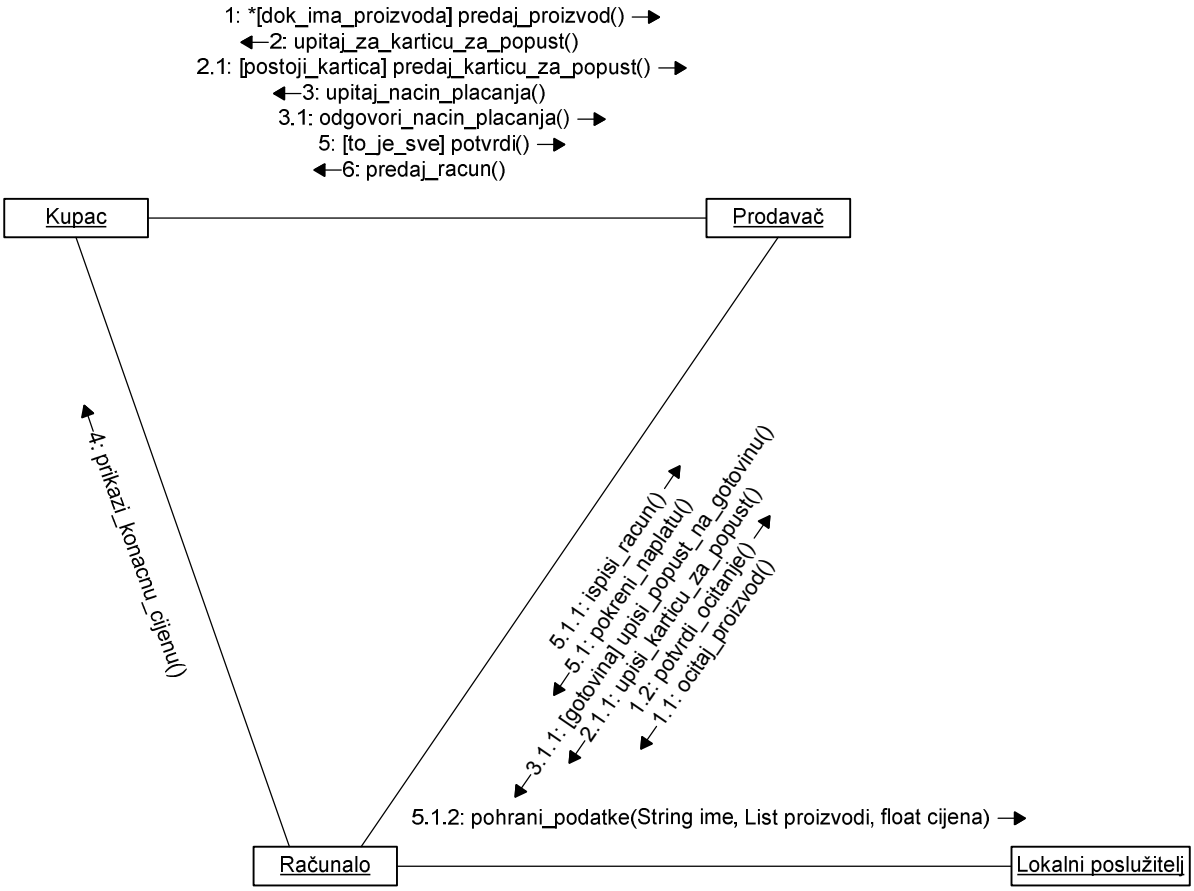
Zadatak 3.4.


Šef poduzeća
Računovodstvo poduzeća
Banka
* [dok stiže] stigao novac (float N_stiglo, float K_ukupno)
[K_ukupno>dug]:
najavi isplatu duga()
šalji nalog za isplatu banci()
isplati dug()
deblokada računa()
račun deblokiran
račun deblokiran
*[postoji tvrtka] šalji nalog za isplatu duga tvrtki
isplati dugovanje()
dugovanje isplaćeno
sva dugovanja isplaćena

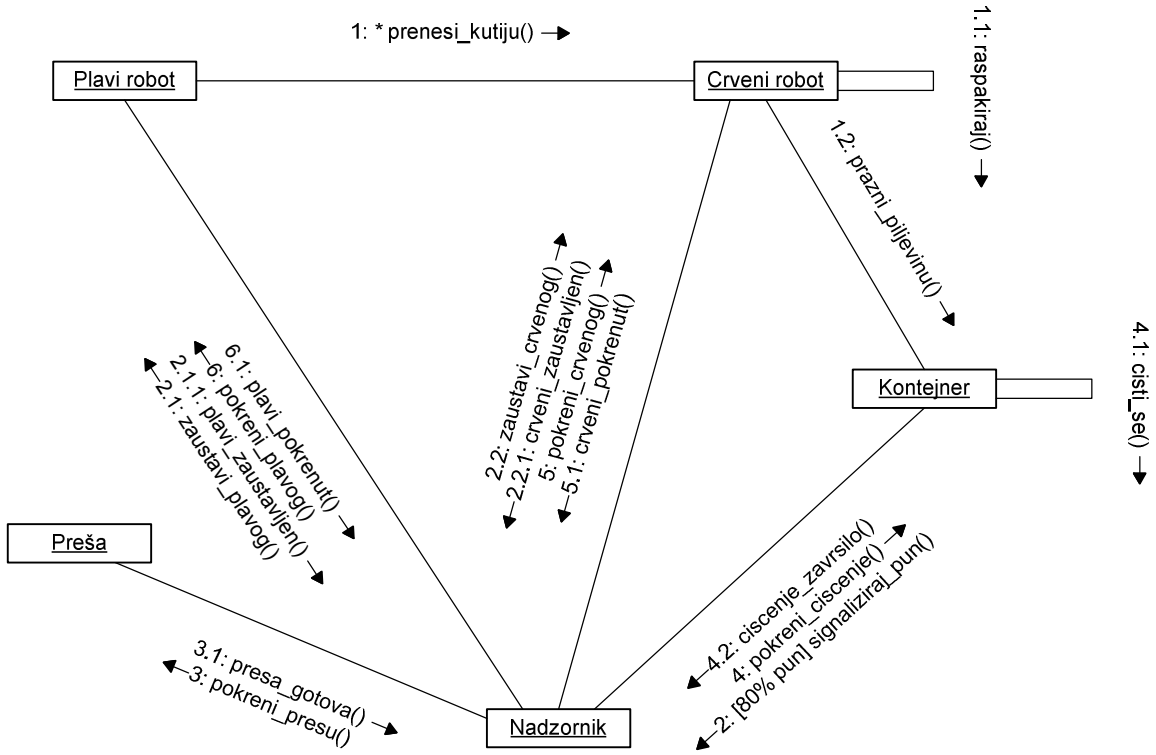
Zadatak 3.5.


Klijent
Senzor
Sigurnosni sustav
Bankovni službenik na pultu
Osobni bankar
Proði kroz prva vrata ()
*[klijent u međuprostoru] Čekaj 5 sekundi ()
Zatvori prva vrata ()
Otvori druga vrata ()
![klijent u međuprostoru] klijent unutra
[treba službenika na pultu] Komuniciraj ()
[treba osobnog bankara] Komuniciraj ()
Završio komunikaciju
Završio komunikaciju
Proði kroz druga vrata ()
[klijent u međuprostoru] Čekaj 5 sekundi ()
Zatvori druga vrata ()
Otvori prva vrata ()
izašao

Zadatak 3.6.


1: *[dok_ima_proizvoda] predaj_proizvod()
2: upitaj_za_karticu_za_popust()
2.1: [postoji_kartica] predaj_karticu_za_popust()
3: upitaj_nacin_placanja()
3.1: odgovori_nacin_placanja()
5: [to_je_sve] potvrdi()
6: predaj_racun()
Kupac
Prodavač
4: prikazi_konacnu_cijenu()
5.1.1: ispisi_racun()
5.1: pokreni_naplatu()
3.1.1: [gotovina] upisi_popust_na_gotovinu()
2.1.1: upisi_karticu_za_popust()
1.2: potvrdi_ocitanje()
1.1: ocitaj_proizvod()
5.1.2: pohrani_podatke(String ime, List proizvodi, float cijena)
Računalo
Lokalni poslužitelj

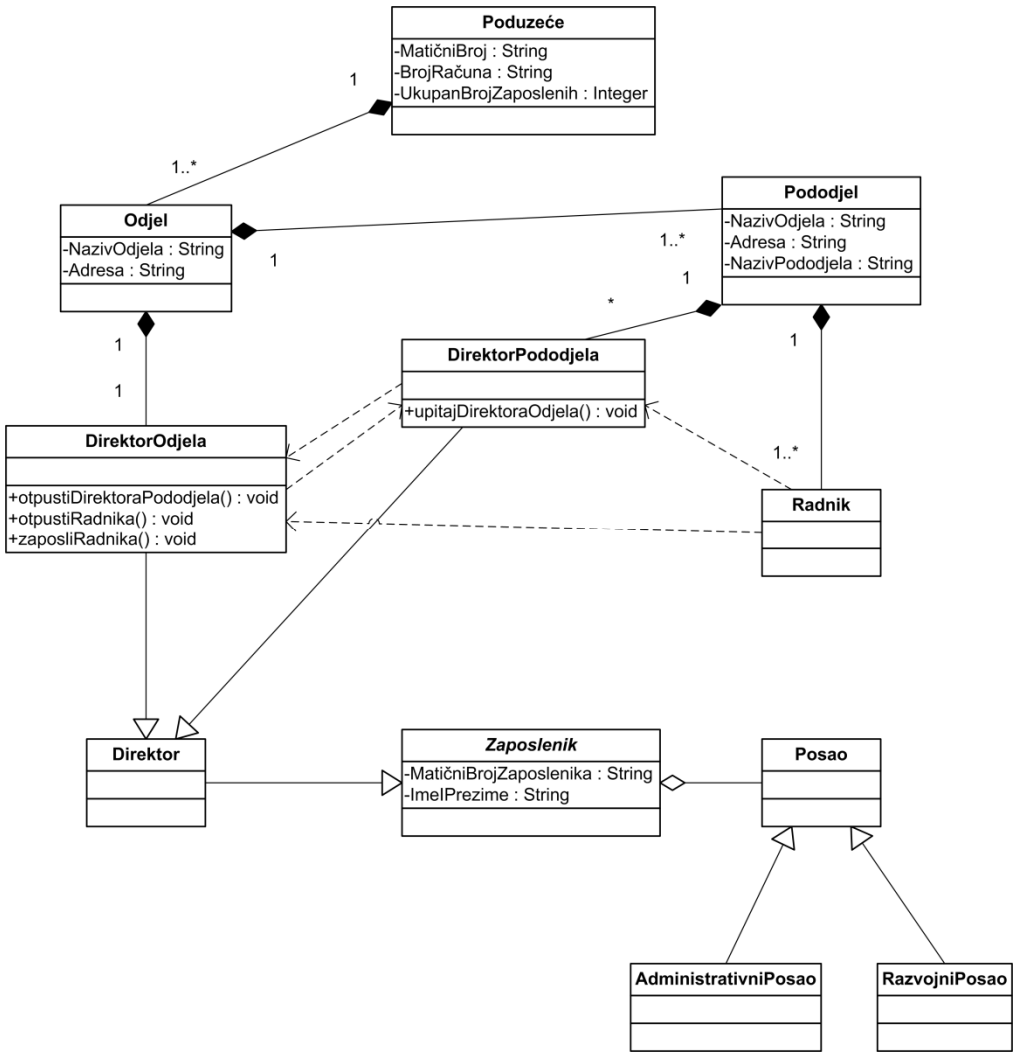
Zadatak 3.7.



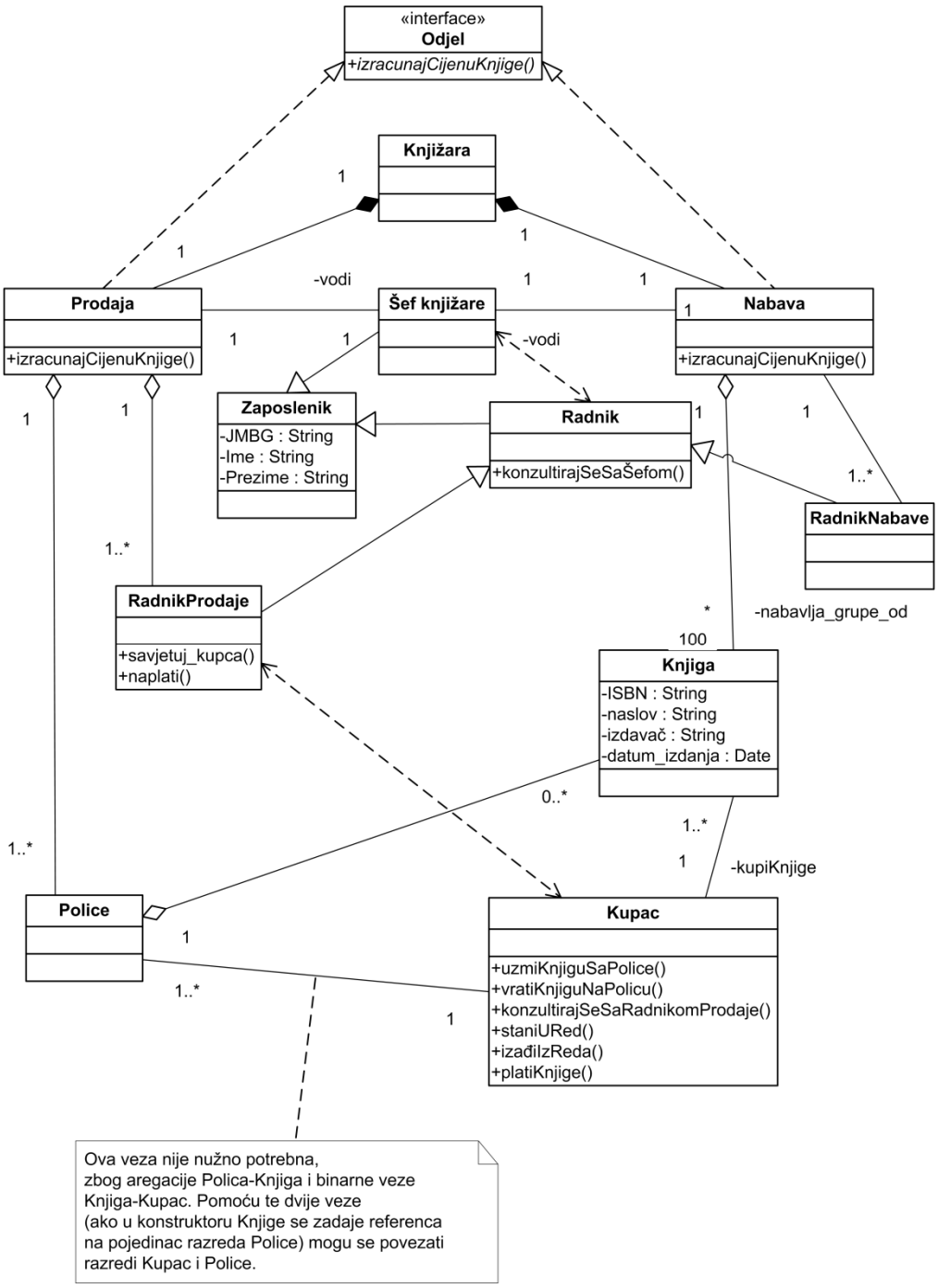
Napomena: asinkronost poruka (u ovom slučaju pri zaustavljanju robota) nije moguće modelirati komunikacijskim dijagramom, tako da se najprije zaustavlja plavi, a potom crveni robot.

9.3. Dijagrami razreda

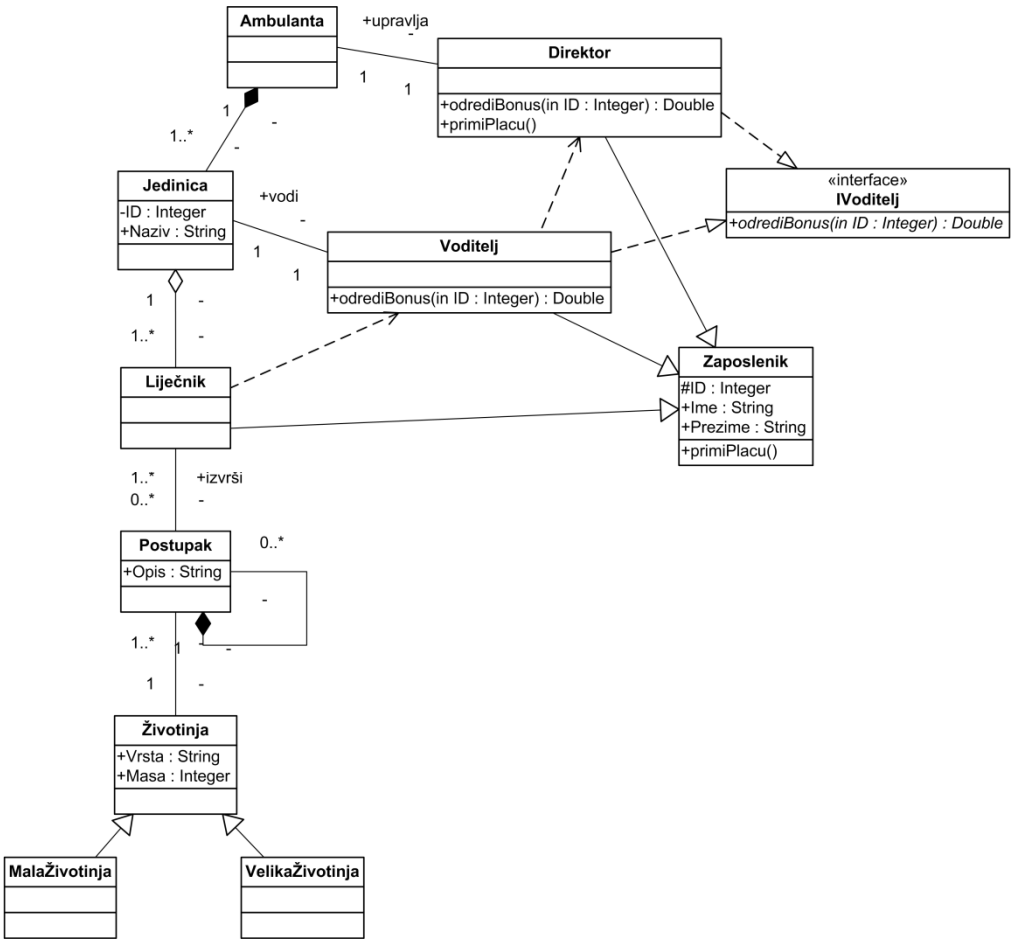
Zadatak 4.1.



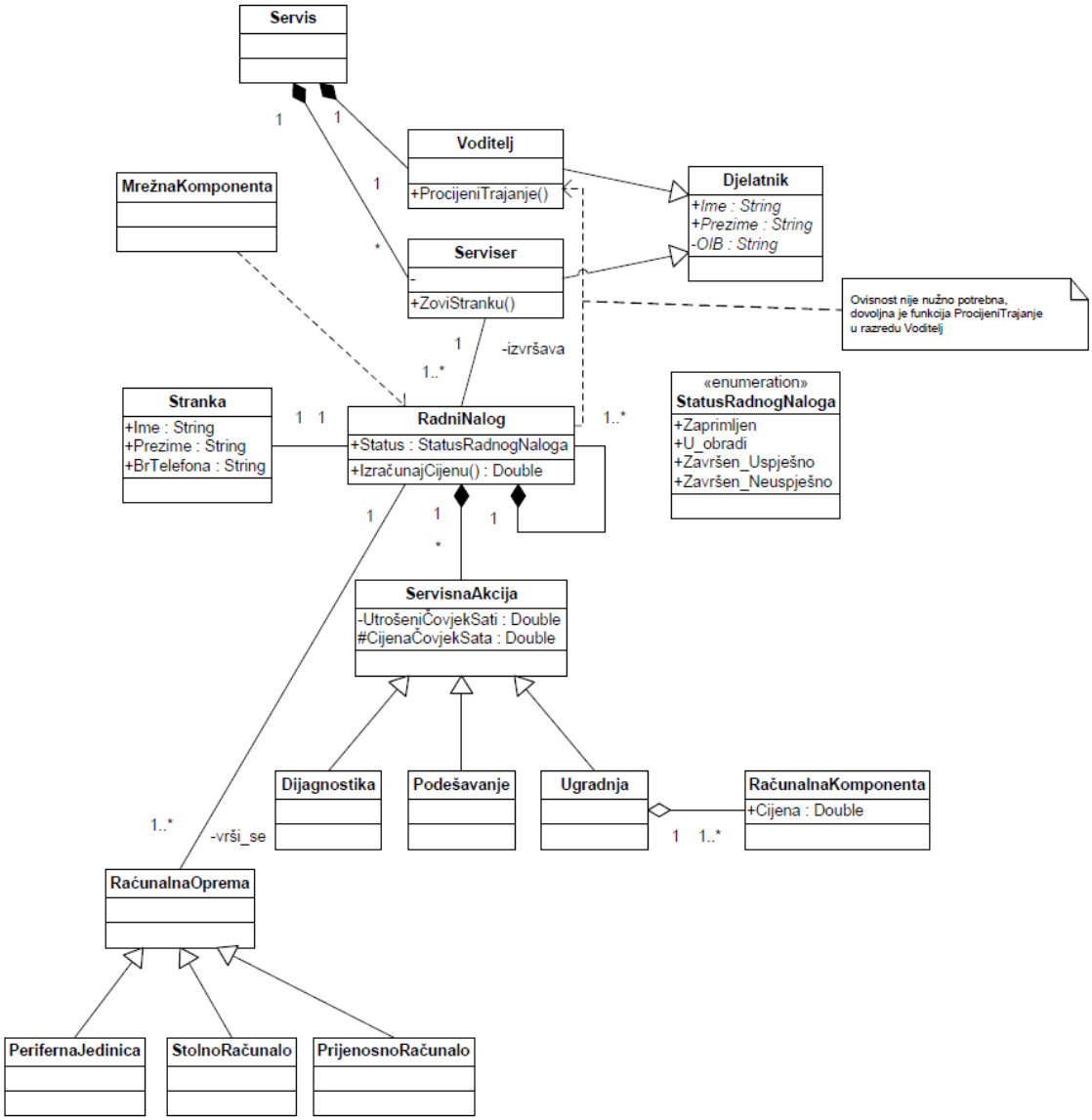
Zadatak 4.2.


«interface»
Odjel
+izracunajCijenuKnjige()
Knjižara
Prodaja
+izracunajCijenuKnjige()
Šef knjižare
Nabava
+izracunajCijenuKnjige()
-vodi
-vodi
Zaposlenik
-JMBG : String
-Ime : String
-Prezime : String
Radnik
+konzultirajSeSaŠefom()
RadnikNabave
RadnikProdaje
+savjetuj_kupca()
+naplati()
-nabavlja_grupe_od
Knjiga
-ISBN : String
-naslov : String
-izdavač : String
-datum_izdanja : Date
-kupiKnjige
Police
Kupac
+uzmiKnjiguSaPolice()
+vratiKnjiguNaPolicu()
+konzultirajSeSaRadnikomProdaje()
+staniURed()
+izađiIzReda()
+platiKnjige()
Ova veza nije nužno potrebna,
zbog aregacije Polica-Knjiga i binarne veze
Knjiga-Kupac. Pomoću te dvije veze
(ako u konstruktoru Knjige se zadaje referenca
na pojedinac razreda Police) mogu se povezati
razredi Kupac i Police.

Zadatak 4.3.

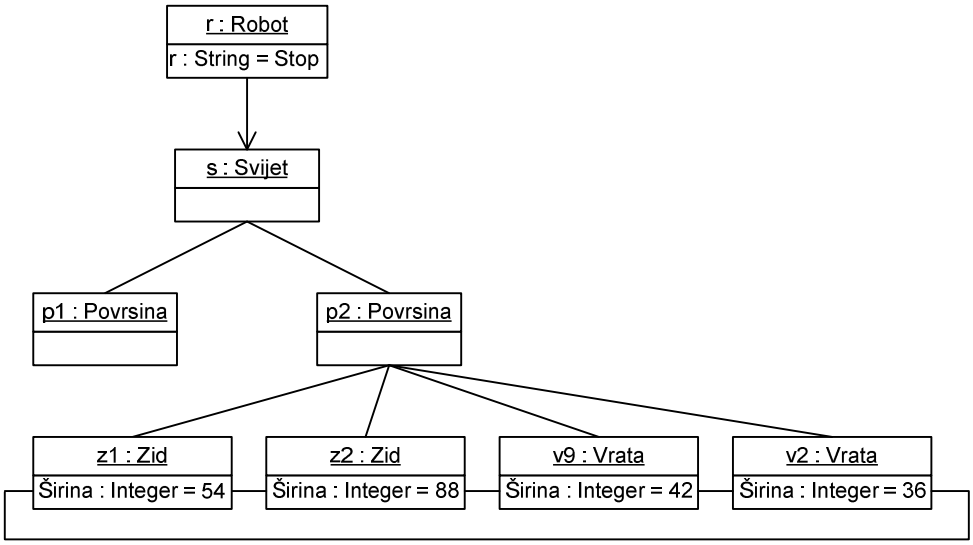

Ambulanta
+upravlja
Direktor
+odrediBonus(in ID : Integer) : Double
+primiPlacu()
«interface»
IVoditelj
+odrediBonus(in ID : Integer) : Double
Jedinica
-ID : Integer
+Naziv : String
+vodi
Voditelj
+odrediBonus(in ID : Integer) : Double
Zaposlenik
#ID : Integer
+Ime : String
+Prezime : String
+primiPlacu()
Liječnik
+izvrši
Postupak
+Opis : String
Životinja
+Vrsta : String
+Masa : Integer
MalaŽivotinja
VelikaŽivotinja

Zadatak 4.4.

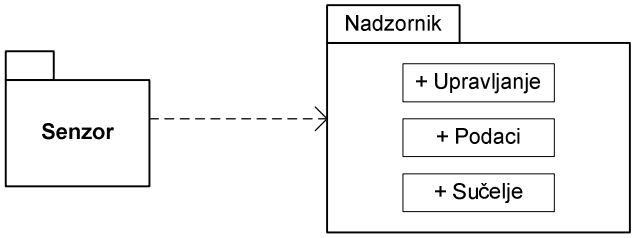

Servis
Voditelj
+ProcijeniTrajanje()
Djelatnik
+Ime : String
+Prezime : String
-OIB : String
MrežnaKomponenta
Serviser
+ZoviStranku()
Ovisnost nije nužno potrebna, dovoljna je funkcija ProcijeniTrajanje u razredu Voditelj
-izvršava
Stranka
+Ime : String
+Prezime : String
+BrTelefona : String
RadniNalog
+Status : StatusRadnogNaloga
+IzračunajCijenu() : Double
«enumeration»
StatusRadnogNaloga
+Zaprimljen
+U_obradi
+Završen_Uspješno
+Završen_Neuspješno
ServisnaAkcija
-UtrošeniČovjekSati : Double
#CijenaČovjekSata : Double
Dijagnostika
Podešavanje
Ugradnja
RačunalnaKomponenta
+Cijena : Double
-vrši_se
RaćunalnaOprema
PerifernaJedinica
StolnoRačunalo
PrijenosnoRačunalo

9.4. Dijagrami objekata i paketa

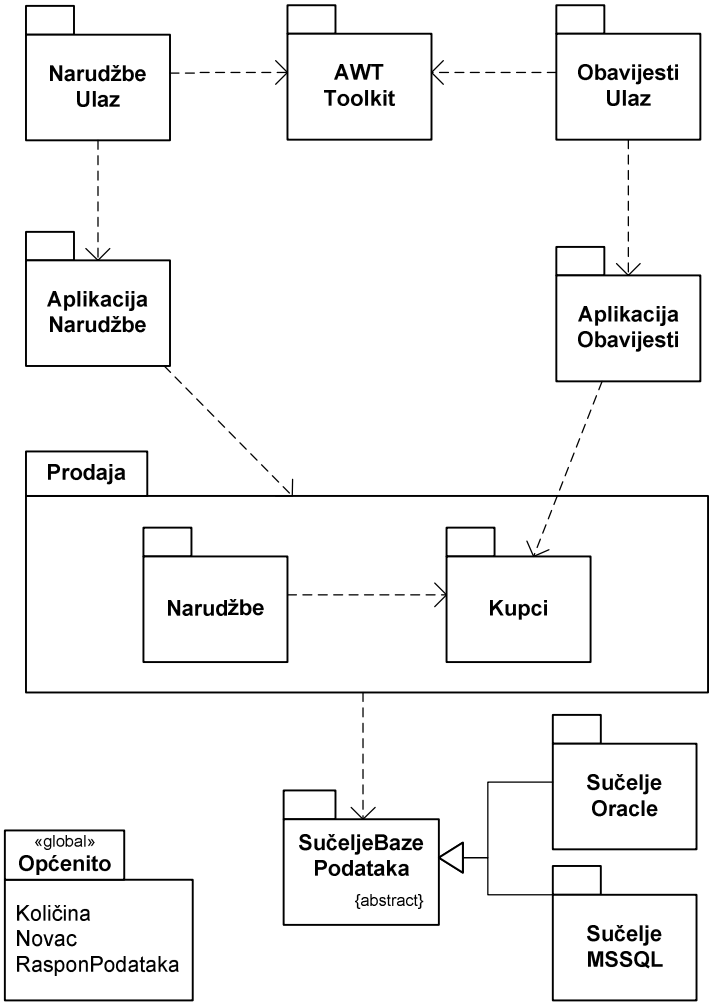
Zadatak 5.1.



Zadatak 5.2.

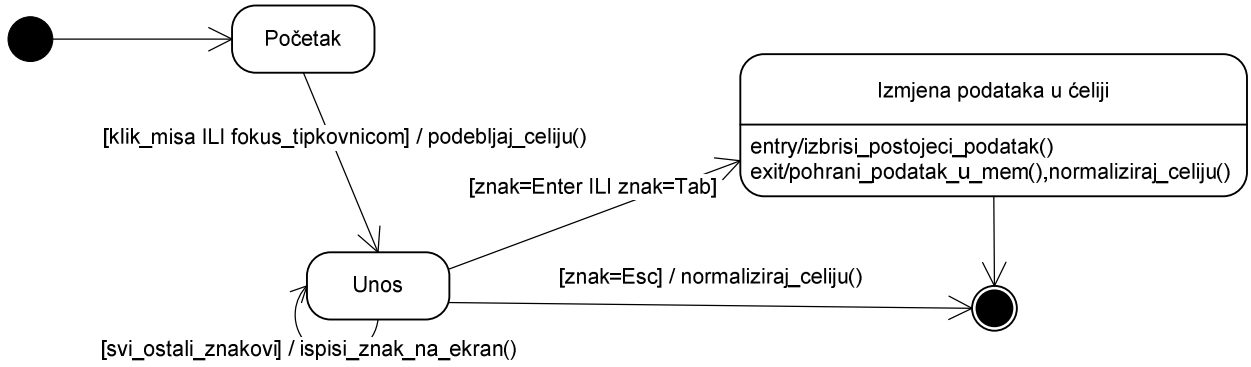


Zadatak 5.3.

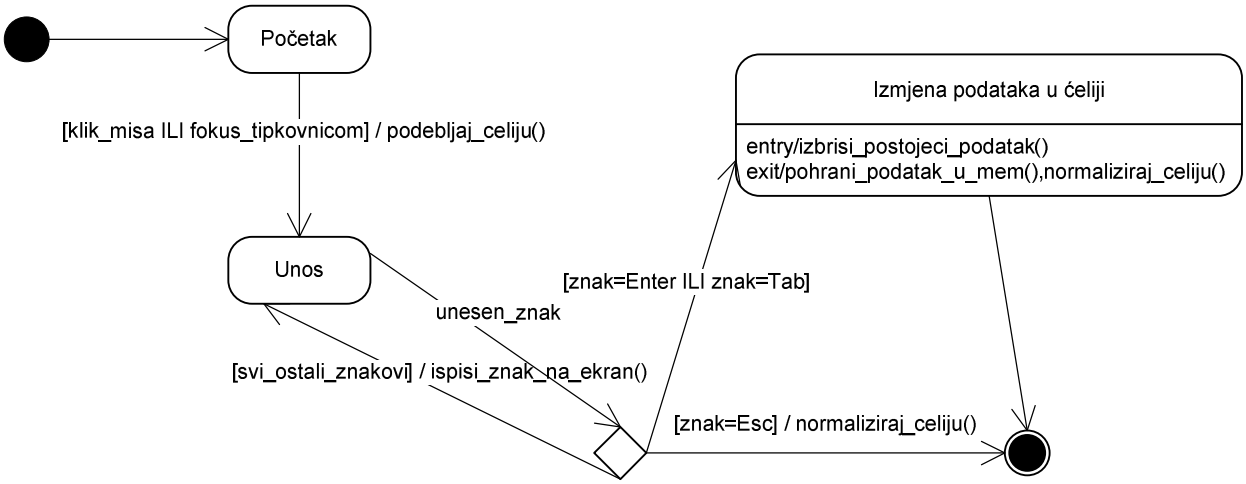

Narudžbe
Ulaz
AWT
Toolkit
Obavijesti
Ulaz
Aplikacija
Narudžbe
Aplikacija
Obavijesti
Prodaja
Narudžbe
Kupci
«global»
Općenito
Količina
Novac
RasponPodataka
SučeljeBaze
Podataka
{abstract}
Sučelje
Oracle
Sučelje
MSSQL

9.5. Dijagrami stanja i aktivnosti

Zadatak 6.1.

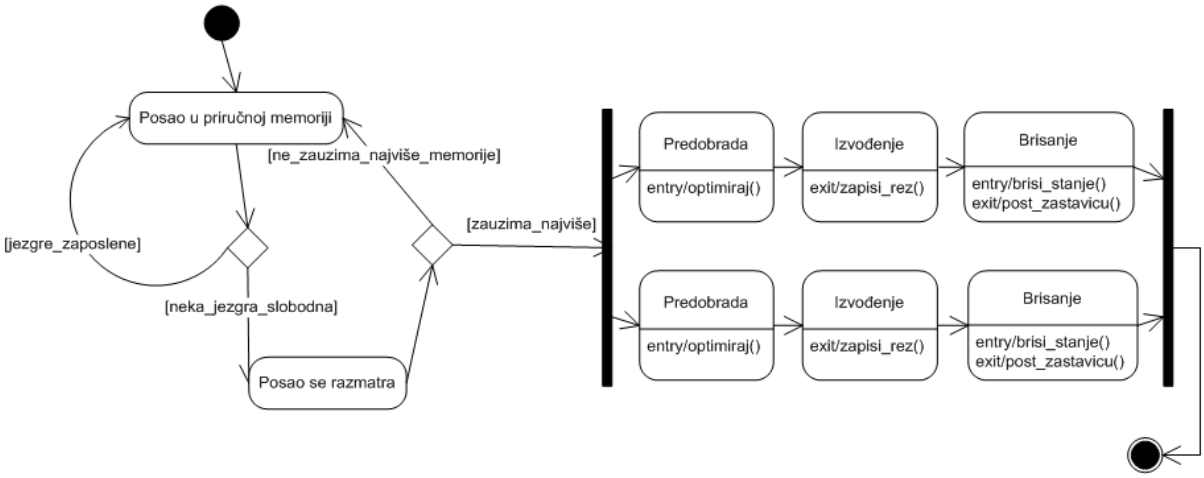


Napomena 1: zadatak je moguće riješiti u uvođenjem čvora odluke odnosno uvjetnog grananja, ovako:

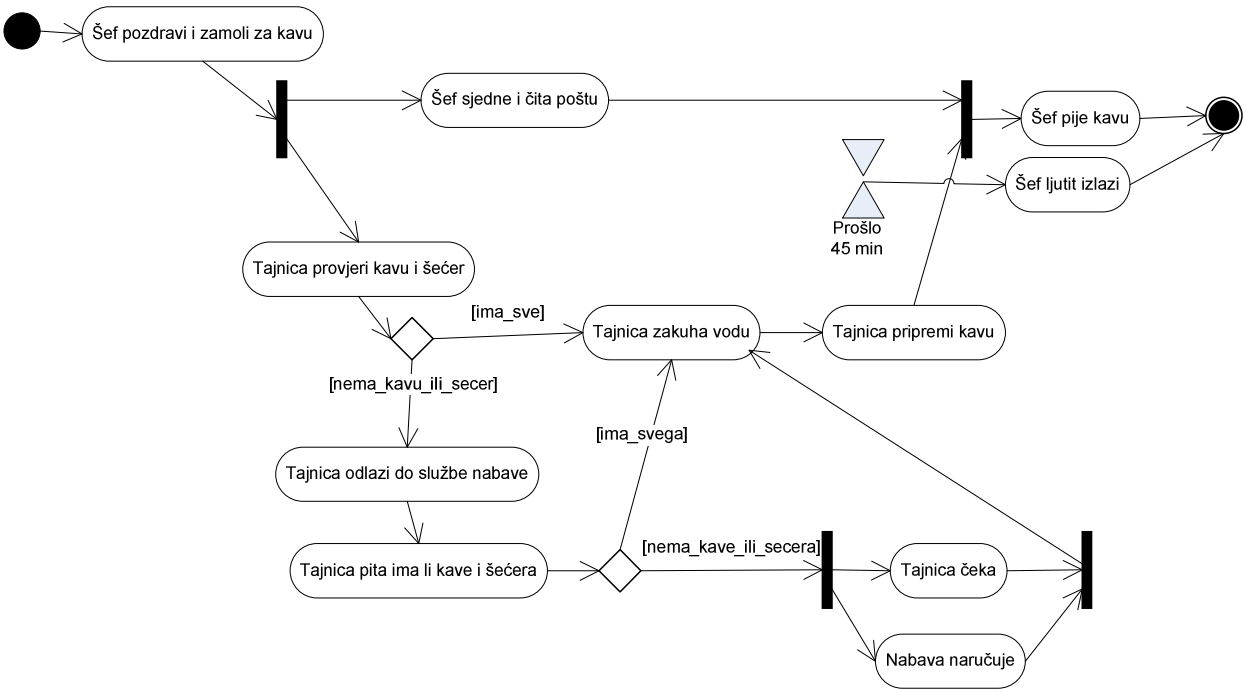


Napomena 2: Nazivi događaja "entry" i "exit" kod stanja "Izmjena podataka u ćeliji" generirani su automatski u programu Visio 2007. Moguć je i bilo koji drugi naziv za događaje u tom stanju.

Zadatak 6.2.

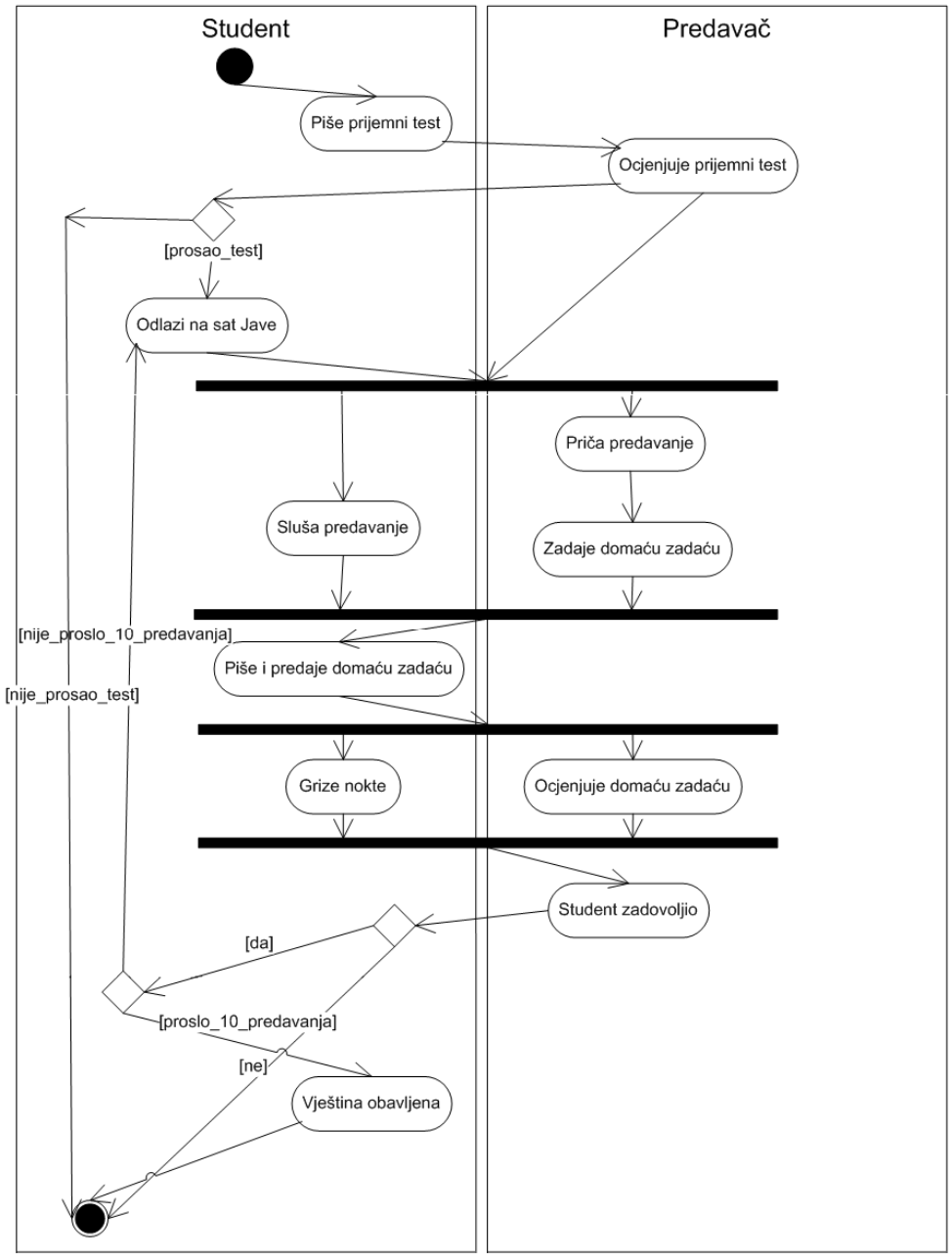


Zadatak 6.3.



Napomena: U dijagram je moguće uključiti i plivačke staze šefa, tajnice i nabave.

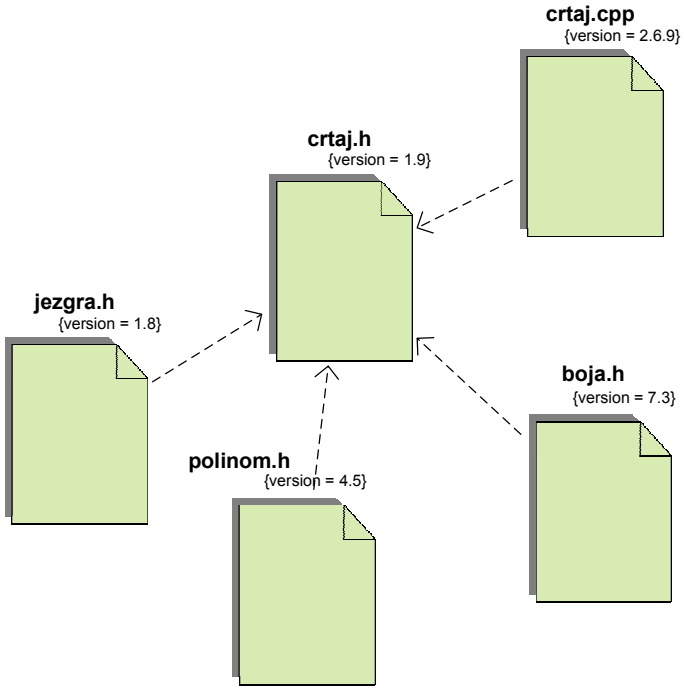
Zadatak 6.4.


Student
Predavač
Piše prijemni test
Ocjenjuje prijemni test
[prosao_test]
Odlazi na sat Jave
Priča predavanje
Sluša predavanje
Zadaje domaću zadaću
[nije_proslo_10_predavanja]
Piše i predaje domaću zadaću
[nije_prosao_test]
Grize nokte
Ocjenjuje domaću zadaću
Student zadovoljio
[da]
[proslo_10_predavanja]
[ne]
Vještina obavljena

9.6. Dijagrami komponenti

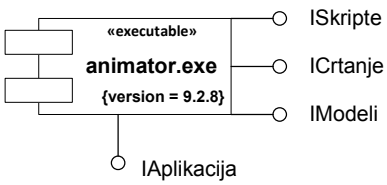
Zadatak 7.1.

U izradi dijagrama koristit će se ikonizirani oblik i općenita slika datoteke izvornog koda.



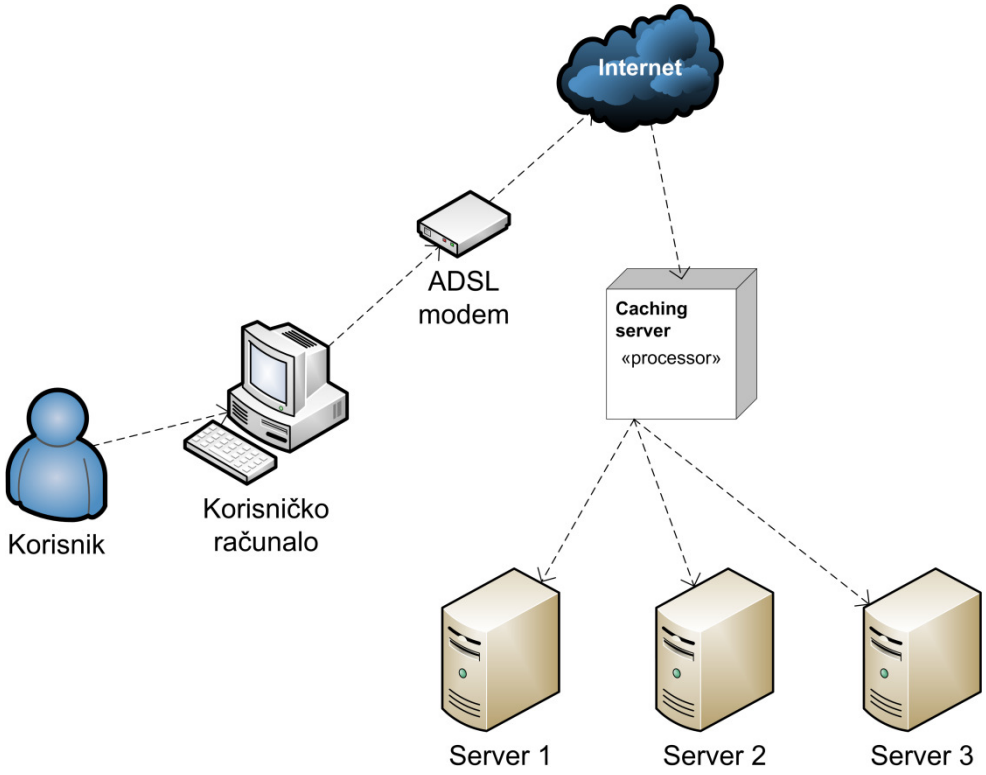
Zadatak 7.2.

Broj sučelja komponente je neograničen. U ovom zadatku definirana su četiri sučelja. Nisu definirane operacije sučelja pa se stoga koristi ikonizirani oblik prikaza.



9.7. Dijagrami razmještaja

Zadatak 8.1.



Literatura

[Bell 2004] D. Bell, UML basics: The sequence diagram, 2004, dostupno na: http://www.ibm.com/developerworks/rational/library/3101.html, [pristupljeno dana 10. 10. 2012.]

[Booch 1999] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[Fowler 2000] M. Fowler, K. Scott, UML Distilled, 2nd ed., Addison-Wesley, 2000.

[Fowler 2004] M. Fowler, UML Distilled: a brief guide to the Standard object modeling language, 3rd ed., Addison-Wesley, 2004.

[Holub 2012] A. I. Holub, Allen Holub's UML Quick Reference (UML 2.0), version 2.1.3, dostupno na: http://www.holub.com/goodies/uml/, [pristupljeno dana 10. 10. 2012.]

[Lethbridge 2005] T. C. Lethbridge, R. Laganière, Object-Oriented Software Engineering, McGraw-Hill Education, 2005.

[OMG 2011] Object Management Group, OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1, 2011, dostupno na: http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/, [pristupljeno dana 10. 10. 2012.]

[Quatrani 2002] T. Quatrani, Visual Modeling with Rational Rose 2002 and UML, 3rd ed., Addison-Wesley, 2002.

[Rational 2001] Rational Software Corporation, Artifact: Use-Case Model, dostupno na: http://www.ts.mah.se/RUP/RationalUnifiedProcess/process/artifact/ar_ucmod.htm, [pristupljeno dana 10. 10. 2012.]

[Wulp 2011] M. van der Wulp, et al., ArgoUML User Manual, v.0.32, 2011, dostupno na: http://argouml-downloads.tigris.org/argouml-0.32/, [pristupljeno dana 10. 10. 2012.]