PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET METROPOLITAN BEOGRAD

KI206 - PROCES I METODOLOGIJE RAZVOJA SOFTVERA

Modelovanje sistema u UML

Lekcija 05

PRIRUČNIK ZA STUDENTE

KI206 - PROCES I METODOLOGIJE RAZVOJA SOFTVERA

Lekcija 05

MODELOVANJE SISTEMA U UML

- Modelovanje sistema u UML
- Poglavlje 1: Uvod u UML
- Poglavlje 2: Uvod u Power Designer
- Poglavlje 3: UML Strukturni modeli
- Poglavlje 4: UML Modeli ponašanja
- Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

Uvod

UVOD

UML je jezik opisivanja softverskog sistema.

Cilj ove lekcije da vas upozna sa grafičkim jezikom za modeliranje sofverskih sistema poznatim pod nazivom UML (Unified Modelling Language) razumete kako se grafički modeli mogu koristiti za predstavljanje softverskih sistema. Kao budući Java programeri, moraćete, yavisno od praks eu IT firmi u kojoj radite, da često pravite Java program na osnovu dokumentacije pripremljene uy pomoć UML, tj. grafičkog jeyika ya prikayivanje funkcija softverskog sistema ili njegove strukture. Kao što mašinski i građevinski inženjeri korist etehničke crteže mašina, odn. ygrada, d abi ynali šta treba d aproizvedu ili da iygrade, tako i programeri, treba da ynaju da čitaju i razumeju UML dijagrame d ali ynali šta treba d aprogramiraju. YBog toga, iako nećete projektovati softver primeom UML.a, treba da ga rayumete, kako bi mogli da programirate na osnovu UML opisa softvera koji vam pripreme projektanti softvera.

Na osnovu ove nastavne jedinice, , razumećete sve važnije UML dijagrame, kao što su:

- **Dijagrami strukture**: Dijagram klasa, Dijagram objekata, Dijagram komponenti, Dijagram rasopreda opreme i instalacije softvera
- **Dijagrami ponašanja (interakcija):** Dijagram slučajeva korišćenja sistema, Dijagram sekvenci, Dijagram stanja, Dijagram aktivnosti

Ove dijagrame ćete kreirati koristeći neki od softverskih alata, kao što je Power Designer. On u sebi sadržijedinsven UML model softverskog sistema, a svaki od dijagrama je samo jedan po+gled na taj model, tako da korisnik vidi samo onaj deo UML modelakoji ga u tom momentu interesuje.

Posle ove lekcije, bićete osposobljeni da samostalno:

- razumete najvažnije UML dijagrame, tj. one koji se najčešće koriste,
- kreirate sve navedene UML dijagrame
- definišete druge podatke koji su od značaja za kreiranje jedinstveng modela softverskog sistema

Detaljniji i potpun opis UML-a se može naći na sajtu OMG organizacije: http://www.omg.org/spec/UML/2.5/

Poglavlje 1

Uvod u UML

UNIFIED MODELING LANGUAGE (UML)

Objašnjenje šta je UML i čemu služi

Inženjeri mašinstva, građevinarstva, eletrotehnik ei dr. koriste svoje specifične jezuike za opis objekta koji projektuju. To mogu biti tehnički crteži, dšematski dijagrami, prostorni CAD modeli idr. Primenom odgovarajućig softverskih alata oni opisuju ono što projektuju, da bi se međusovno razumeli i da bih razumeli ostali, koji treba da proizvedu ono šo su oni projektovali. Koji jeyik koriste softver inženjeri, odnosno inženjeri softvera? Mogu koristiti programske jezike, ali taj način je isuviše detaljan i nejasan pri opisu vežih softverskih sistema. Očigledno, neophodan je jedan grafički orijentisan jezik, jer čovek mnogo brže prihavta i razume informacije koje su grafički prikayane, nego one koji su prikayane samo tekstom.

Kao odgovor na ovo pitanje nameće se standardna definicija UML-a. UML ili Unified Modeling Languageje grafički jezik za vizualizaciju, specifikaciju, konstruisanje i dokumentovanje sistema programske podrške koji je postavljen kao standard.

Ako malo fleksibilnije želimo to da objasnimo moramo naglasiti da UML nije programski već grafički jezik. Njime, moguće je pratiti razvoj neke građevine, izraditi plan uzgajanja voća u voćnjaku itd. UML predstavlja kolekciju najuspešnijih inženjerskih metoda kojima je iskustveno dokazano, pojednostavljeno modelovanje velikih i složenih sistema.

Ako napravimo poređenje između građevinara i programera, možemo reći da je UML standardizovani jezik za izradu nacrta programa.

Definisano je sedam ciljeva kojima UML kao jezik teži:

1. Pružiti korisniku brz jezik za vizuelno modelovanje kojim će moći u relativno kratkom vremenu napraviti i razmenjivati modele sa određenim značenjem,
2. Pružiti korisniku mogućnost proširenja i stvaranja specijalizovanih delova,
3. Biti nezavisan od programskih jezika i razvojnih procesa,
4. Pružiti formalne osnove za razumevanje jezika za modelovanje,
5. Podsticanje rasta i razvoja objektno orijentisanih programskih jezika,
6. Podrška visoko pozicioniranih razvojnih pojmova kao što su saradnja, okvirni rad, uzorci i komponente,
7. Integrisanje i nadopunjavanje praktičnim iskustvom.

PET POGLEDA NA UNIFIED MODELING LANGUAGE (UML)

Struktura UML-a

U cilju lakšeg razumevanja šta je UML potrebno je poći od činjenice od čega se on sastoji.UML se sastoji od niza pogleda na arhitekturu (Architectural Views) koji zavise od problema i rešenja, a dele se na:

1. Pogled korišćenja (use case view)
2. Logički pogled (logical view)
3. Pogled paralelnog rada (concurrency view)
4. Pogled na komponente (component view)
5. Pogled postavljanja (deployment view)

Pogled korišćenja - opisuje problem i rešenje onako kako ga vide oni koji postavljaju problem.

Logički pogled – opisuje strukturnu dimenziju problema i rešenja.

Pogled paralelnog rada – opisuje dimenziju ponašanja problema i rešenja, a naziva se još i dinamički pogled.

Pogled na komponente – opisuje strukturu i ponašanje realizacije rešenja, a naziva se još i razvojni pogled.

Pogled postavljanja – opisuje strukturu i ponašanje domena u kome je rešenje ostvareno, a naziva se još i fizički pogled ili pogled na razmeštaj.

Svaki od ovih pogleda opisan je pomoću devet UML dijagrama, a svaki pogled je opisan sa jednim ili više dijagrama. UML dijagram je grafička prezentacija skupa elemenata. Dijagrami se crtaju kako bi se vizuelno prikazao sistem iz različitih perspektiva. Za gotovo sve sisteme, osim onih vrlo jednostavnih, dijagrami predstavljaju poboljšani prikaz elemenata koji čine sistem. Isti elementi mogu se pojaviti u svim dijagramima. Teoretski, dijagram može sadržavati bilo koju kombinaciju stvari i relacija u modelu. U praksi, međutim, samo se mali broj kombinacija pojavljuje i one su konzistentne sa pet najkorisnijih pogleda koje smo prethodno nabrojali.

UML DIJAGRAMI

Upotreba i svrha UML dijagrama

Da bi smo razumeli ulogu ovih dijagrama, moramo da naglasimo da se UML sastoji od dva tipa gradivnih blokova i samih dijagrama, elemenata i relacija.

- **Dijagram slučajeva korišćenja** (use case diagram) prikazuje skup slučajeva korišćenja, aktera (specijalne vrste klasa) i njihovih relacija. Use case dijagram se koristi za modeliranje funkcionalnosti i specifikaciju sistema.

- **Dijagram klasa** (class diagram) prikazuje logičku strukturu apstrakcija (skup klasa, interfejsa, kolaboracija i njihovih relacija).
- **Dijagram objekata** (object diagram) prikazuje logičku strukturu instanci (skup objekata (instanci klasa) i njihovih veza).
- **Dijagram sekvenci,** tj. redosleda poruka među objektima-klasama (sequence diagram) je dijagram interakcije koji naglašava vremenski redosled poruka.
- **Dijagram saradnje** (collaboration diagram) definiše zajednicu i interakciju aktera i drugih elemenata koji dejstvuju zajedno da ostvare kooperativno ponašanje koje je kvalitativno novo u odnosu na prostu sumu ponašanja elemenata.
- **Dijagram stanja** (statechart diagram) prikazuje konačni automat koji obuhvata stanja, tranzicije, događaje i aktivnosti.
- **Dijagram aktivnosti** (activity diagram) prikazuje tok od jedne do druge aktivnosti u sistemu.
- **Dijagram komponenti** (component diagram) prikazuje fizičku organizaciju i zavisnosti između skupa komponenata.
- **Dijagram instalcije** (deployment diagram) prikazuje konfiguraciju čvorova obrade i komponenata koje postoje na njima.

Bez obzira na čitav niz teorija i pristupa još uvek ne postoji tačno uputstvo kakav proces primeniti za razvoj pojedinih vrsta softvera.

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji obuhvataju UML i poglede na UML.

1. Zadatak: Opisati pogled korišćenja i logički pogled. Koje su razlike u navedenim pogledima?

2. Zadatak: Kada je potrebno koristiti dijagram komponenti? Dati konkretan primer.

Poglavlje 2

Uvod u Power Designer

UVOD

Power Designer je grafičko okruženje za modelovanje aplikacija, podataka, poslovnih procesa

Power Designer je grafičko okruženje za modelovanje aplikacija, podataka, poslovnih procesa, koje omogućava i kreiranje izveštaja, generisanje koda, reverse engineering i brojne druge funkcionalnosti. Sadrži više tipova modela:

- model zahteva (requirements model, RQM)
- konceptualni model podataka (conceptual data model, CDM)
- logički model podataka (logical data model, LDM)
- fizički model podataka (physical data model, PDM)
- model likvidnosti informacija (information liquidity model, ILM)
- objektno orijentisani model (object-oriented model, OOM)
- model arhitekture organizacije (enterprise architecture model, EAM)
- model poslovnih procesa (business process model, BPM)
- XML model (XSM)
- “slobodni” model (free model, FEM).

INSTALACIJA POWER DERSIGNER ALATA

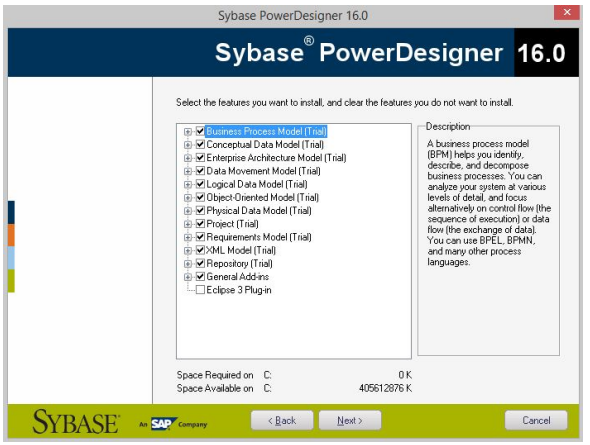
Kratke informacije koje se odnose na instalaciju Power Designer alata

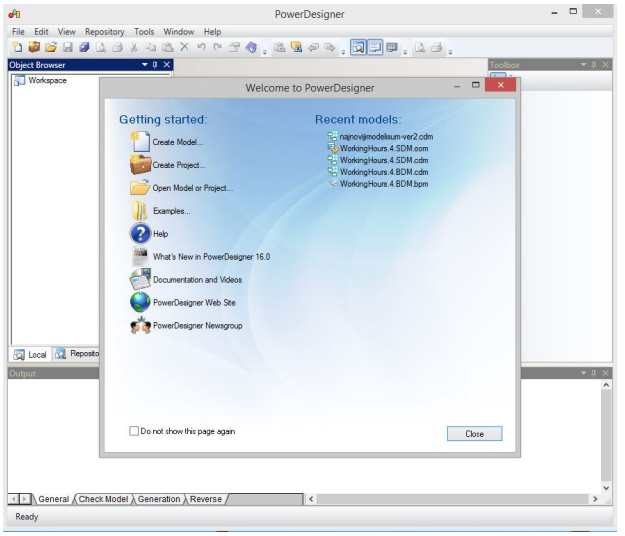
Power Designer razvojni alat možete preuzeti sa zvaničnog sajta (http://www.sybase.com/products/modelingdevelopment/powerdesigner). Sa sajta je moguće preuzeti TRAIL tip licence. Obzirom da naš Univerzitet ima licence za korišćenje ovog alata možete ih preuzeti kontaktiranjem IT službe Univerziteta.

Ukoliko niste u mogućnosti da preuzmete sa sajta program, možete isti tražiti takođe od ajti službe ili asistenta.

Potrebno je pokrenuti instalaciju i prilikom instalacije odabrati sve tipove dijagrama kao što je prikazano na slici 1a . Ukoliko ste uspešno instalirali Power Designer razvojni alat možete ga pokrenuti i dobićete prikaz kao na slici 1b .

Ovim se smatra da ste prvi deo vežbi završili i možete nesmetano da pratite nastavak ovih vežbi.

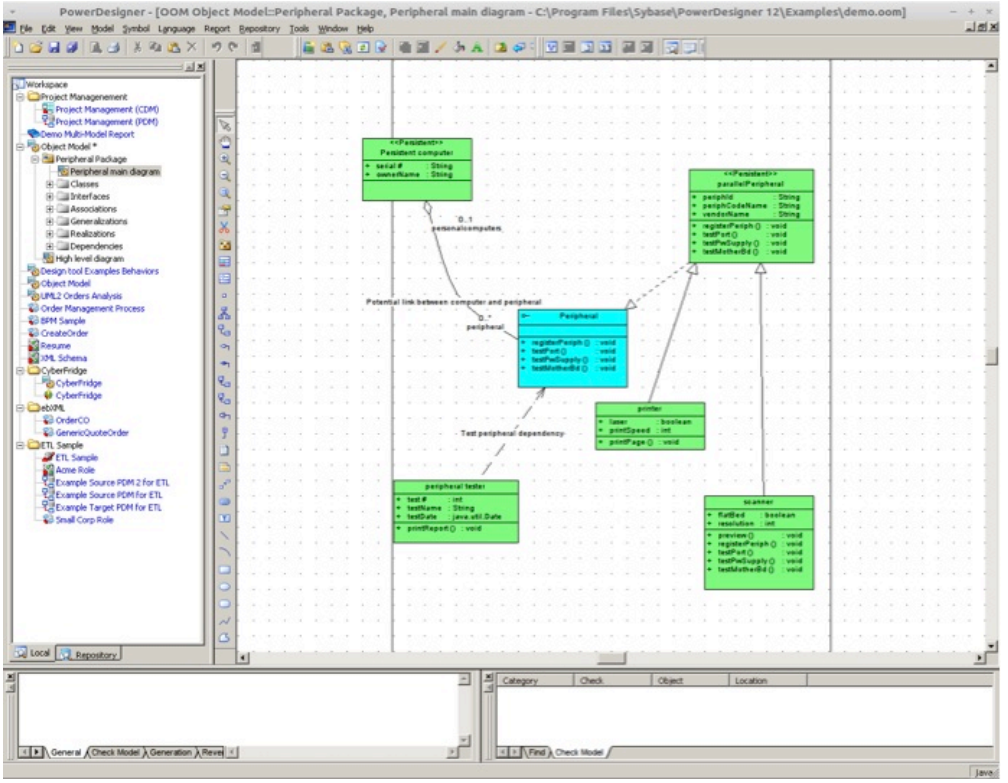




GLAVNI PROZOR

Radni prostor prikazuje trenutno izabrani dijagram

- Browser prikazuje model i njegove elemente u obliku stabla, te omogućava brzu navigaciju.
- Radni prostor prikazuje trenutno izabrani dijagram.
- Paleta alatki sadrži alatke za kreiranje elemenata tekućeg dijagrama, a njen sadržaj zavisi od tipa prikazanog dijagrama.
- Pomoćni prozori prikazuju informacije o napretku trenutne operacije (Output), te o rezultatima izvršene operacije (Result List), a prikazuju se automatski kada se u njima pojave nove informacije

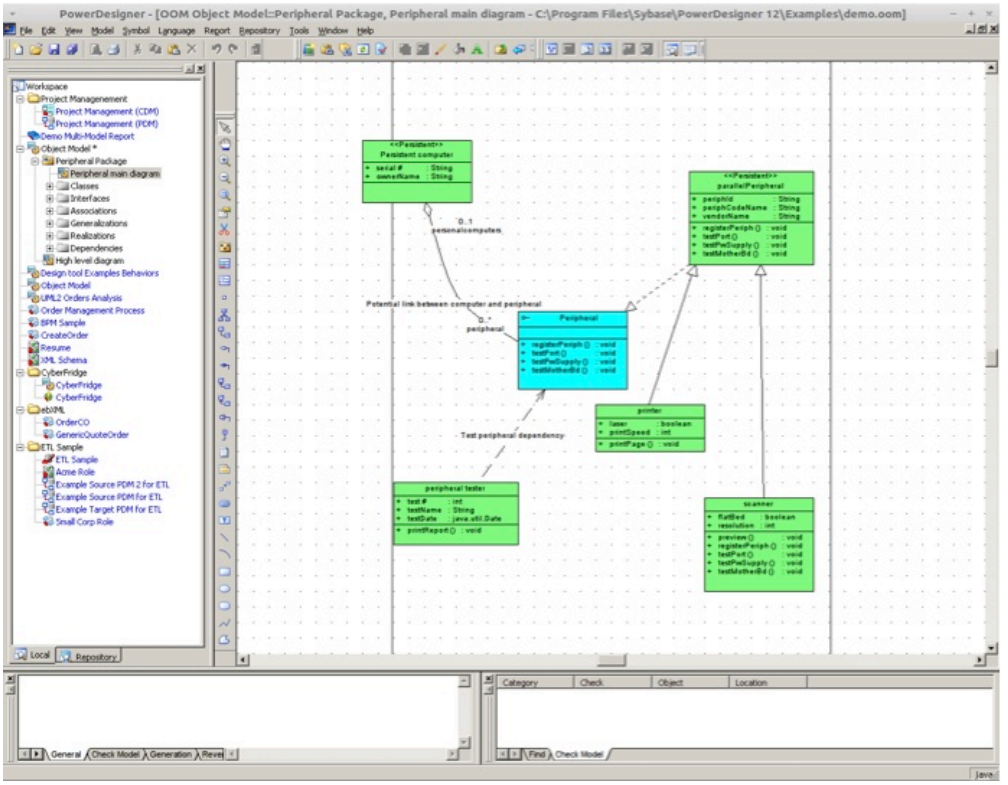


HIJERARHIJA MODELA U POWER POINT-U

Radni prostor obezbeđuje innformaciju o projektu i njegovim modelima.

U korenu browser-a se uvek nalazi neki workspace, koji sadrži informacije o projektima i modelima sa kojima se trenutno radi, te omogućava snimanje i učitavanje čitavog tog radnog okruženja. Dodatno se projekti i
modeli unutar jednog workspace-a mogu organizovati u foldere. Workspace se čuva u obliku datoteke sa ekstenzijom sws, koja sadrži informacije o organizaciji njegovih elemenata i linkove ka datotekama sa modelima i projektima. U jednom trenutku može i mora da bude otvoren samo jedan workspace. Otvaranje
workspace-a (File → Open Workspace) podrazumeva zatvaranje trenutno otvorenog workspace-a.

Zatvaranjem workspace-a (File → Close Workspace) automatski se kreira novi prazan workspace.



Slika 2.1 Hijerarhija Power Designer modela

MODEL SISTEMA

Model je osnovna jedinica dizajna, a svaki model može da sadrži više dijagrama

Projekat je sledeći mogući nivo u hijerarhiji. Ukoliko postoji, on služi za grupisanje srodnih modela. Na
primer, jedan projekat može da sadrži sve modele (model zahteva, objektni, konceptualni...) koji opisuju neki
softverski projekat, te se učitavanjem date projektne datoteke učitavaju svi modeli relevantni za taj softverski
projekat. Sem modela, koji mogu da budu organizovani u foldere, projekat sadrži dijagram koji opisuje na koji
su način njegovi modeli i drugi resursi međusobno povezani (veze se kreiraju automatski).

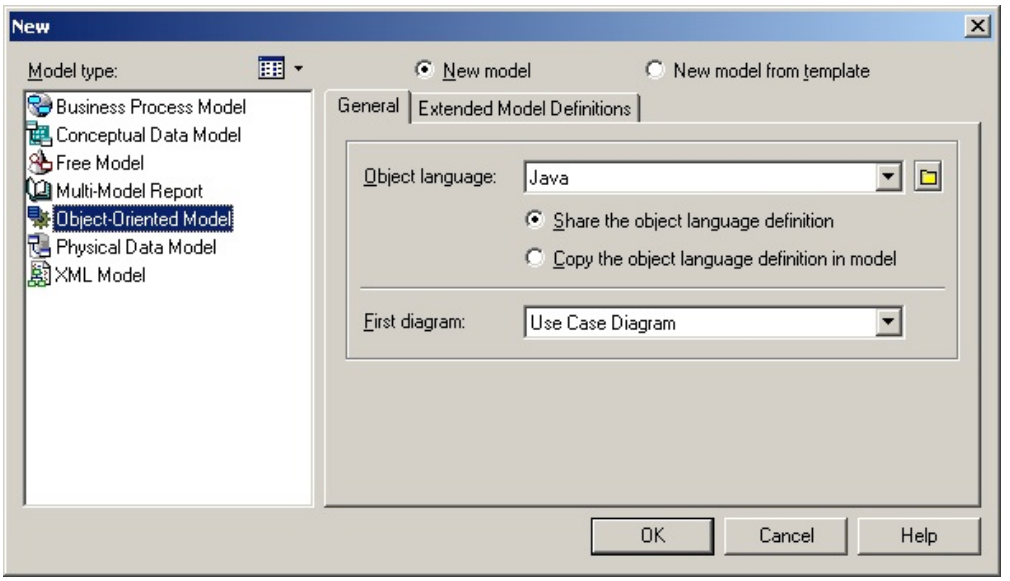
Model je osnovna jedinica dizajna, a svaki model može da sadrži više dijagrama. Dijagrami se u okviru
modela mogu organizovati u pakete. Model se (zajedno sa svim svojim dijagramima) čuva u datoteci čija
ekstenzija odgovara tipu modela (na primer, objektni modeli se čuvaju sa ekstenzijom oom). Prilikom čuvanja
modela, kreira se uvek i dodatna, backup datoteka, koja ima ekstenziju koja se završava na “b” (na primer,
za objektne modele oob).

KREIRANJE MODELA

Novi model se kreira odabirom opcije File → New iz glavnog menija

Novi model se kreira odabirom opcije File → New iz glavnog menija. U dijalogu se odabira tip modela sa leve strane, te njegove specifične osobine sa desne.

Svi UML dijagrami se nalaze u okviru Object-Oriented modela, a sa desne strane se za njega odabira programski jezik pomoću kojeg će sistem biti implementiran, te tip UML dijagrama koji će prvi biti kreiran.



Slika 2.2 Kreiranje modela

PALETA ALATKI

Paleta alatki je toolbar omogućava izbor standardnih alata, alatke za kreiranje objekata i grafičkih apalata

Paleta alatki je toolbar koji može da stoji u sopstvenom prozoru ili se može "dokovati" na neku od četiri strane radnog prostora. Ukoliko se ne vidi, može se prikazati odabirom: Tools → Customize Toolbars, pa čekirati odgovarajući check-box.
Alatke koje se nalaze na paleti grupisane su u tri dela:

- standardni alati (prvih sedam ikonica): strelica (selekcioni režim), ručica (panning), alatke za
  zumiranje, otvaranje poddijagrama, properties, brisanje alatke specifične za prikazani tip dijagrama
- alatke za kreiranje objekata zajedničkih za sve tipove dijagrama: beleška (note), uopštena veza
  (extended dependency), eksterna datoteka, okvir sa informacijama o modelu (title box), tekstualno
  polje (text)

- grafičke alatke za crtanje osnovnih matematičkih oblika (linija, pravougaonik, elipsa...).

Slika 2.3 Paleta alatki

Beleška (note) je element koji se koristi za pojašnjavanje drugih elemenata dijagrama. Iako u Properties dijalozima elemenata postoji prostor za komentare, prednost beleške je u tome što je informacija vidljiva na samom dijagramu. Ona se uvek povezuje sa elementom koji objašnjava vezom Extended Dependency.

PROPERTIES DIJALOG

Properties dijalog sadrži sve informacije o izabranom objektu raspoređene u kartice. Inicijalno se ne prikazuju sve kartice.

Svaki objekat u PowerDesigner-u (model, dijagram, elementi dijagrama, čak i veze) ima pridružen Properties dijalog, do kojeg se dolazi dvoklikom na objekat ili izborom Properties stavke iz kontekstnog menija. Ovaj dijalog sadrži sve informacije o izabranom objektu raspoređene u kartice. Inicijalno se ne prikazuju sve kartice, te se one mogu prikazati klikom na dugme “More >>” u donjem levom uglu dijaloga.

Svi objekti imaju sledeće kartice:

- General, na kojoj se mogu specificirati: Name (naziv objekta), Code (kodni naziv objekta), Comment
  (komentar)
- Notes, na kojoj se mogu nalaziti dodatni, opširniji komentari (sadrži dve potkartice: Description i
  Annotation)
- Version Info, sa informacijama o autoru i datumu kreiranja i poslednje izmene
- Rules, sa spiskom poslovnih pravila vezanih za dati objekat
- Requirements, sa spiskom zahteva vezanih za dati objekat
- Dependencies, sa spiskom svih zavisnih elemenata (generiše se automatski na osnovu veza sa
  drugim elementima i sadrži potkartice za svaki tip veze)
- Extended Dependencies, sa spiskom svih objekata sa kojima postoji korisnički definisana zavisnost
  (pomoću Extended dependency veze sa palete) - ove veze ne podležu nikakvoj proveri niti
  interpretaciji, te služe samo u dokumentacione svrhe.

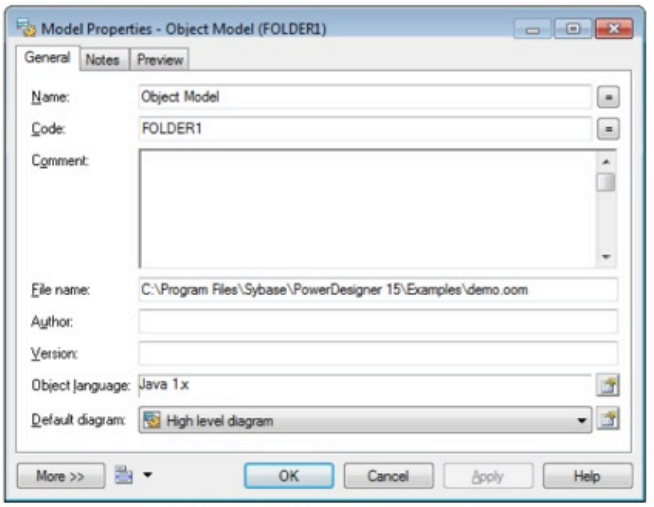
Ostale kartice i polja na Properties dijalogu zavise od tipa izabranog objekta i opisane su u uputstvima za odgovarajuće dijagrame.

DEFINISANJE OSOBINA MODELA

Properties dijalog modela omogućava definisanje osnovnih osobina modela

Properties dijalog modela omogućava definisanje osnovnih osobina modela. Pristupa mu se odabirom Model → Model Properties ili desni klik na model u browser-u → Properties:

Tools → Model options ili desni klik na praznu površinu dijagrama → Model options: otvara dijalog u kojem se mogu podešavati konvencije za dodelu imena za sve tipove objekata u modelu.



Slika 5. Properties dijalog modela

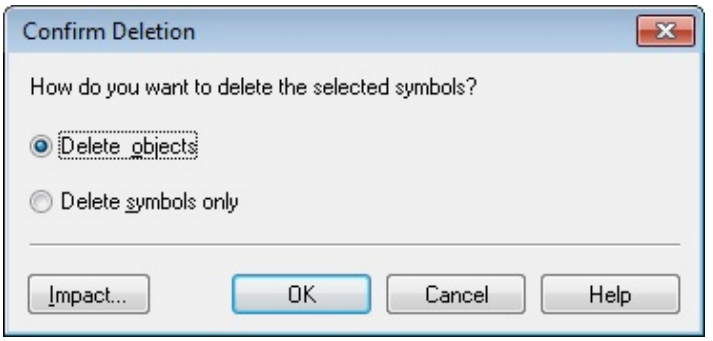
Name	Deskriptivan naziv modela
Code	Kodni naziv modela
Comment	Kratak opis modela
File name	Datoteka u kojoj se model čuva
Author	Autor modela
Version	Verzija
Object language	Programski jezik odabran pri kreiranju modela (read-only), koji će se koristiti kao ciljni jezik prilikom generisanja koda
Default diagram	Dijagram koji se prvi otvara kada se otvori model. Podrazumevano je to dijagram koji je prvi kreiran.

Slika 2.4 Properties dijalog modela

DIJALOG ZA POTVRDU BRISANJA

Omogućava potvrdu namere da se obriše neku objekat.

Kada se pokrene brisanje elemenata i veza, prikazuje se sledeći dijalog;



Slika 2.5 Dijalog za potvrdu brisanja

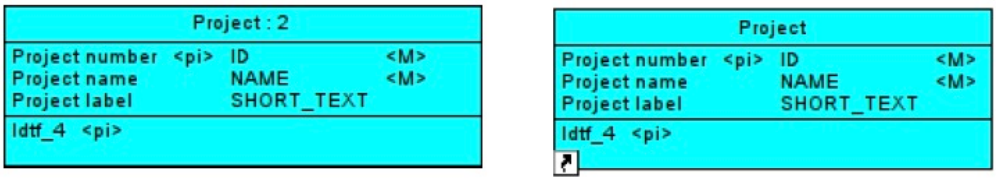
Naime, elementi i veze u PowerDesigner-u mogu, a ne moraju da imaju svoju grafičku predstavu, tj. da budu prikazani na nekom dijaramu. Opcija “Delete objects” briše objekat kompletno iz modela, dok opcija “Delete symbols only” briše samo njegovu grafičku predstavu sa dijagrama Ovakav element se i dalje može nalaziti na drugim dijagramima, a ako ga nema ni na jednom dijagramu, može mu se pristupiti iz browser-a ili različitih listi objekata. Simbol elementa se može dobiti tako što se element iz browser-a prevuče na dijagram.

SINONIMI I PREČICE

Sinonimi se koriste za povećanje preglednosti velikih dijagrama. Prečice (shortcut) se koriste za povezivanje elemenata koji se nalaze na različitim dijagramima

Sinonimi se koriste za povećanje preglednosti velikih dijagrama. Moguće je kreirati proizvoljno mnogo sinonima za element i rasporediti ih na različita mesta u okviru jednog dijagrama. Svaki sinonim je označen svojim rednim brojem (prikazan uz naziv elementa posle “:”). Svaka izmena na bilo kojem od sinonima jednog elementa se propagira i na sve ostale.

Sinonim elementa se kreira odabirom opcije Edit → Create Graphical Synonym iz njegovog kontekstnog menija, ili pomoću prečice Ctrl+M. Kada se neki od sinonima obriše, ne prikazuje se dijalog za brisanje, a sve njegove veze se prevezuju na druge sinonime.



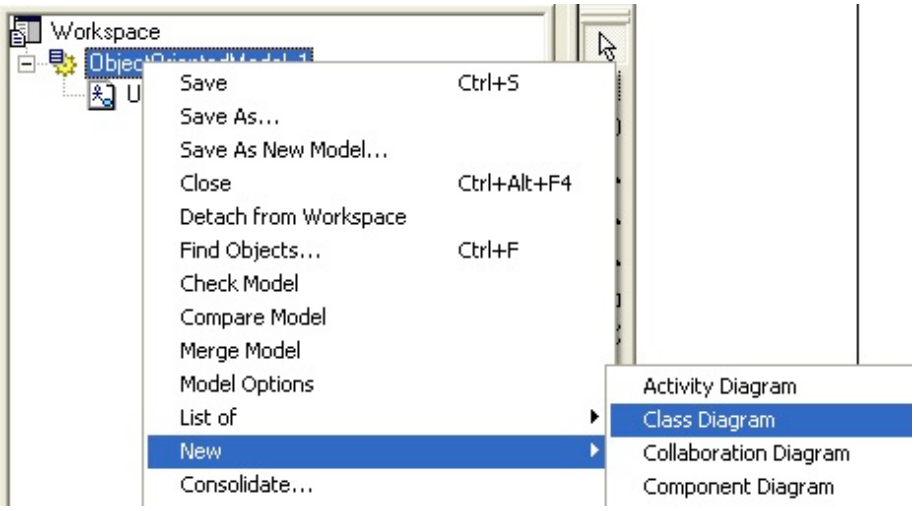
Slika 2.6 Sinonim i prečuca

Prečice (shortcut) se koriste za povezivanje elemenata koji se nalaze na različitim dijagramima. Prečica se kreira običnim kopiranjem elementa, a na ciljnom dijagramu se odabira desni klik na prazno → Edit → Paste as Shortcut. (Običan Paste kreira kopiju elementa koja nije vezana za original, tj. kreira novi element.)

KREIRANJE KLASA

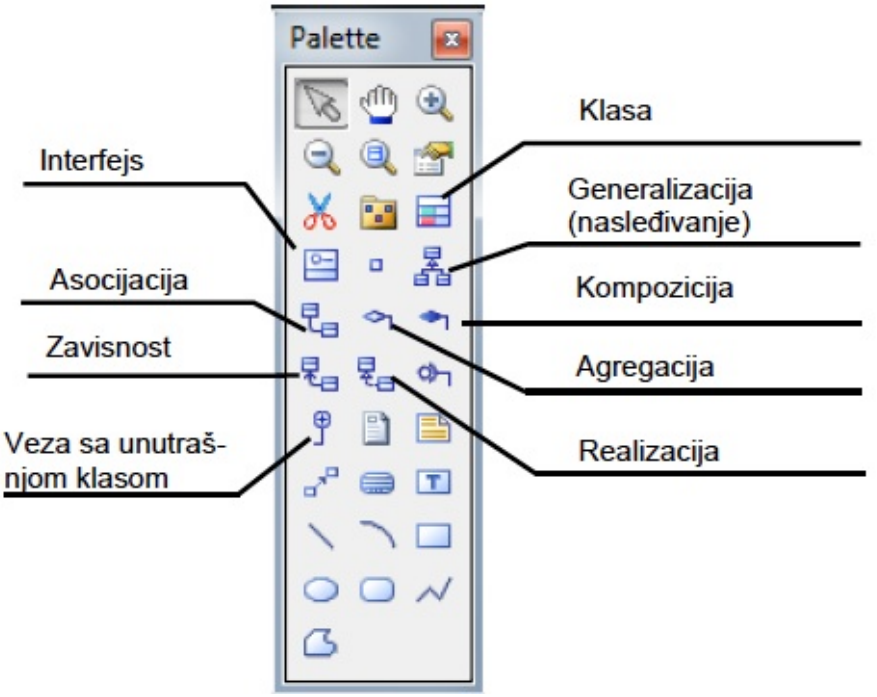
Vrši izbor tipa dijagrama (Class Diagram) i biraju se potrebni alati uz pomoć paleta alata.

Kreiranje novog dijagrama klasa



Slika 2.7 0 -izbor za Dijagram klasa

Korišćenje palete

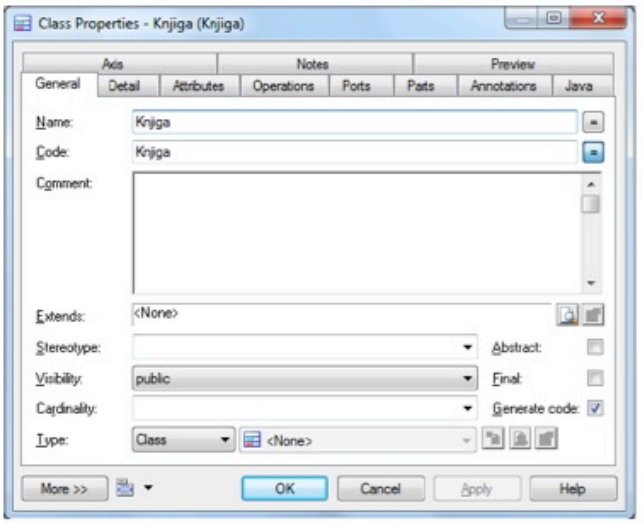


Slika 2.8 Korišćenje palete

DIJALOZI ZA PODEŠAVANJE OSOBINA KLASE I INTERFEJSA

Karica General omogućava unos podataka za svojstva klasa i interfejsa

U ovom odeljku opisan je način definisanja osobina klase, njenih atributai operacija. U velikoj meri se ovajopis odnosi i na interfejse, koji sadrže podskup ovde opisanih osobina klase.



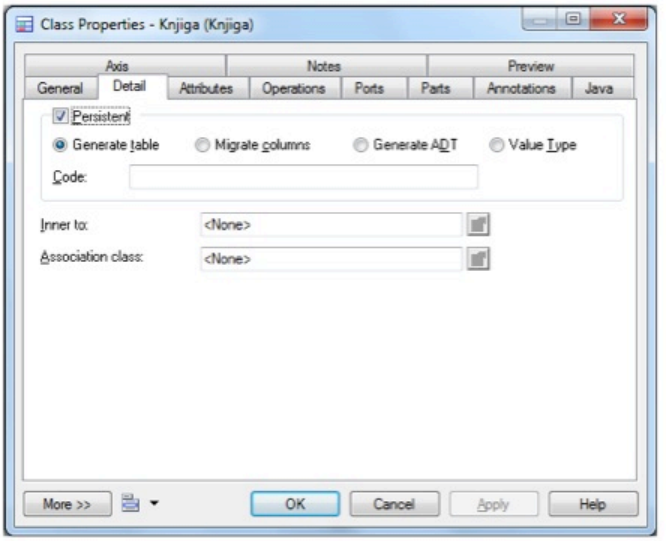
Najvažniji parametri:

Name	Deskriptivan naziv klase
Code	Naziv klase koji se prenosi u programski kod
Comment	Opis klase
Extends	Predak date klase
Visibility	Vidljivost klase, označava kako je vide drugi objekti
Cardinality	Predviđeni broj instanci klase: 0..1, 1..1, 1..*, *
Type	Class – obična klasa Generic – generička klasa (v. odeljak 3.5 Generičke klase) Bound – konkretizacija generičke klase
Abstract	Oznaka da je klasa apstraktna
Final	Oznaka da je klasa poslednja u lancu nasleđivanja, odnosno da ne može imati naslednike
Generate code	Oznaka da je klasa predviđena za generisanje koda na izabranom programskom jeziku (uključuje se među objekte koji se generišu iz modela kada se pokrene proces generisanja)

KARTICA DETAIL

Kartica Detail omogućava unos podataka klase: Pesistant, Inner to i Association Class.

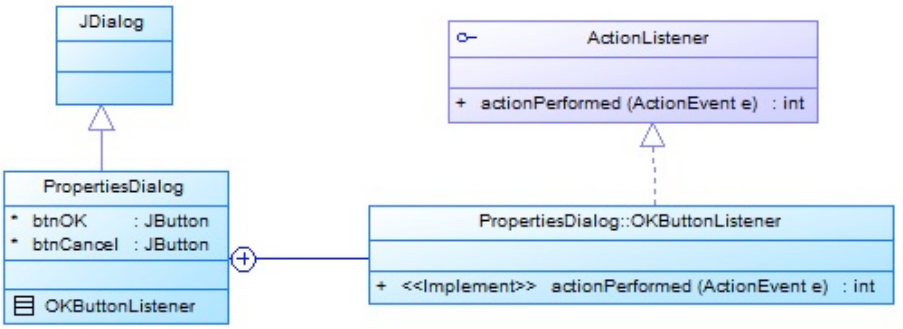
Kartica Detail:



Persistent	Oznaka da je klasa perzistentna. Perzistentne klase su klase koje poseduju metode za smeštanje svojih atributa na disk i učitavanje sa diska, pri čemu "skladište" za smeštanje/učitavanje može biti datoteka ili baza podataka. U okviru *PowerDesigner*-a, ukoliko je klasa označena kao perzistentna, njeni podaci će biti preneti u fizički model baze podataka. Način na koji će ovaj proces biti obavljen definiše se podopcijama u okviru polja Persistent.
Inner to	Ime klase kojoj data klasa pripada, ukoliko se radi o unutrašnjoj klasi neke druge klase
Association class	Oznaka da klasa učestvuje u okviru veze asocijacije između druge dve klase (pogledati odeljak 4.1.3 Klasa asocijacije)

Slika 2.9 Kartica Detail

Unutrašnje klase se na dijagramu povezuju vezom “Inner Link”. Na primer, na sledećoj slici je prikazan Java dijalog sa dva dugmeta i listenerom za dugme OK koji je implementiran kao unutrašnja klasa dijaloga



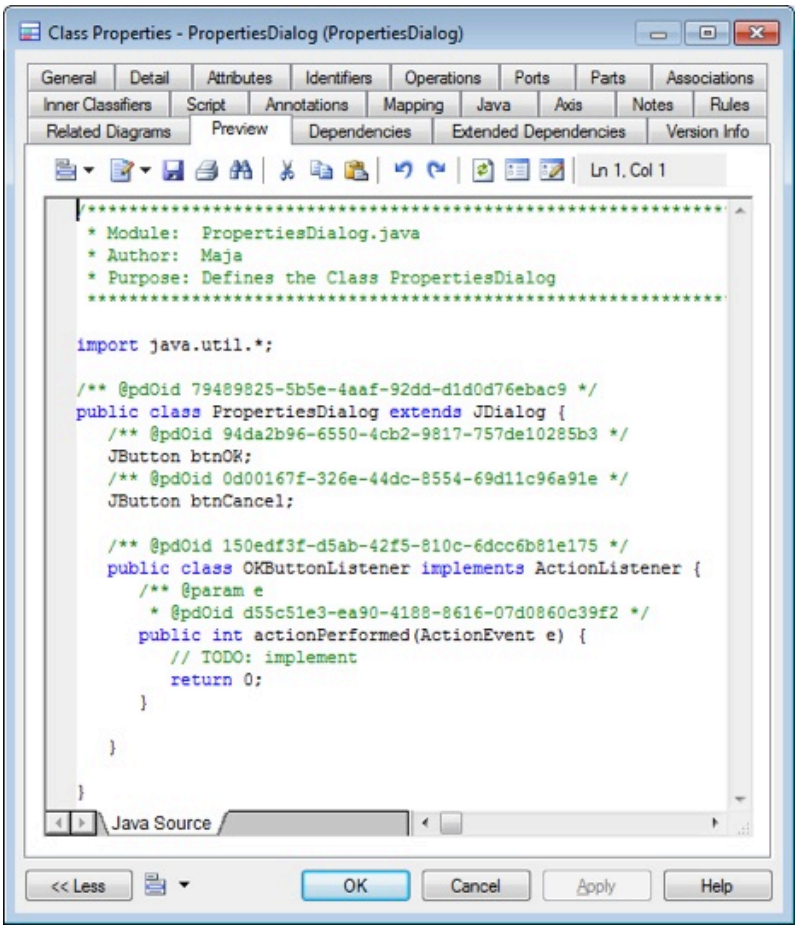
Slika 2.10 Primer

KARTICA PREVIEW

Preview pokazuje automatski generisan programski kod klase.

Služi za uvid u predloženi programski kod klase

Generisani kod je proširen komentarima koji služe za Reverse engineering – proces kojim se tekuća verzija aplikacije usaglašava sa dijagramom klasa..

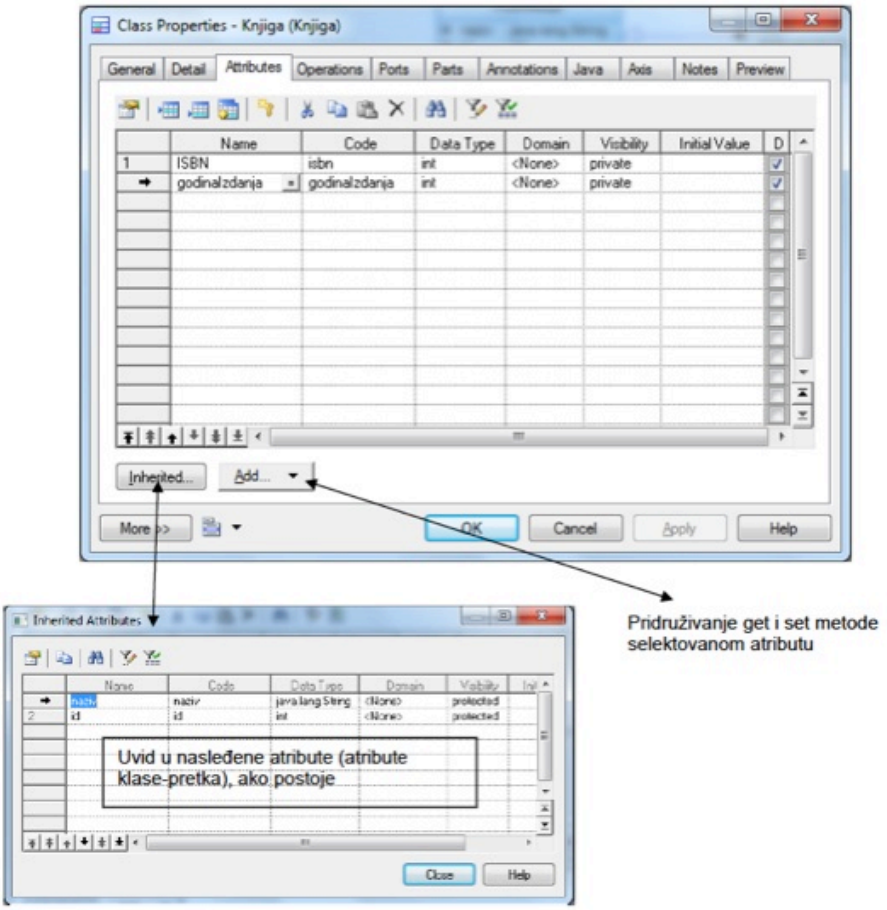


Slika 2.11 Kartica Perview - pokazuje generisano kod klase

ATRIBUTI KLASE

Kartica (tab) Atributes oogućava unos atributa klase

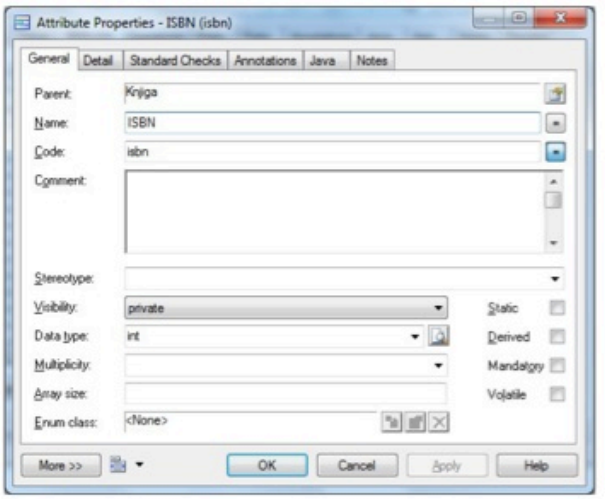
Kolona “D” u spisku atributa označava da li je atribut vidljiv u okviru dijagrama. Radi povećanja preglednosti dijagrama, moguće je ostaviti da budu vidljivi samo najvažniji atributi. Vidljivost atributa se tiče samo izgleda dijagrama, ne i generisanja koda ili prava pristupa atributu!



Slika 2.12 Kartica (tab) Atributi

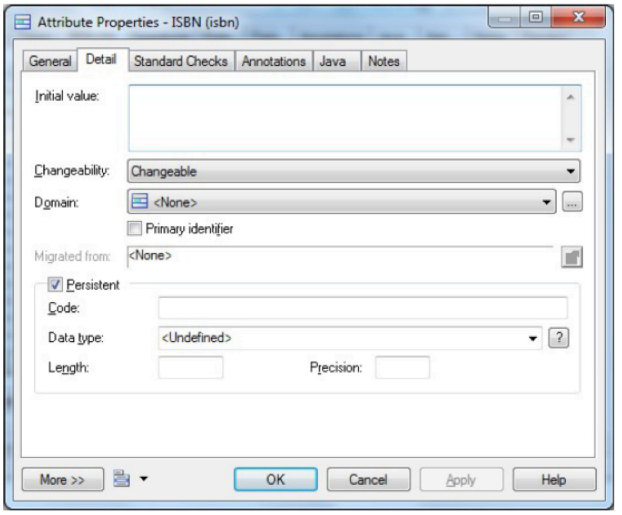
KARTICE GENERAL I DETAIL

Kartice General i Deatil omogućavaju unos podataka o atributima



Name	Deskriptivan naziv atributa
Code	Naziv atributa koji se prenosi u programski kod
Visibility	Vidljivost atributa (public, protected, private, package). Podrazumevana vidljivost može se definisati u dijalogu Tools → Model Options.
Data Type	Tip atributa. Lista raspoloživih tipova zavisi od izabranog programskog jezika. Podrazumevani tip može se definisati u dijalogu Tools → Model Options.
Mulitplicity	Višestrukost atributa: opseg broja vrednosti koje objekat posmatrane klase može da ima za dati atribut (npr. 0..*)
Array size	Služi za specificiranje višestrukosti atributa kao slobodnog teksta. Ako ponuđene opcije u Multiplicity polju nisu dovoljne, ovde se može uneti opisni izraz ili izraz na ciljnom programskom jeziku (ne utiče na generisani kod).
Static	Oznaka da je atribut static (vezan za klasu, a ne za instancu klase)
Derived	Vrednost atributa može da se izvede iz vrednosti ostalih atributa
Mandatory	Obavezan atribut (donja granica za Multiplicity veća od nule)

Slika 2.13 Kartica General

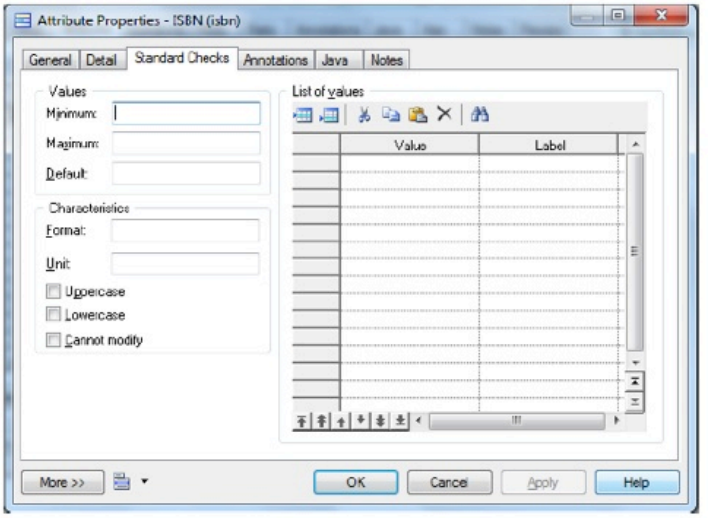


Initial value	Inicijalna vrednost atributa. Ukoliko je uneta, kôd za dodelu inicijalne vrednosti formira se u okviru konstruktora klase (C++), odnosno u odeljku za definisanje atributa (Java).
Changeability	Izmenljivost atributa nakon kreiranja instance klase: - Changeable : može da se menja (podrazumevano) - Read-only: može se samo očitavati (zabranjena set metoda) - Frozen: jednom dodeljena vrednost atributu ne može se menjati (konstanta) - Add-only (samo ako je višestrukost veća od 1): nove vrednosti se mogu dodavati, ali se ne mogu vršiti druge izmene kolekcije
Domain	Odabir domena, kojim se može zameniti tip atributa (v. odeljak 3.4.4 Domeni)
Primary identifier	Oznaka da se atribut pretvara u deo primarnog ključa u relacionom modelu
Persistent	Oznaka da je atribut perzistentan, tj. da predstavlja obeležje u tabeli koja odgovara posmatranoj klasi, sa sledećim osobinama: - Code: naziv obeležja - Data type: tip podatka obeležja

Slika 2.14 Kartica Detail

KARTICE STANDARD CHECKS

Domen predstavlja tip podataka sa pridruženim informacijama o ograničenjima i perzistentnosti.



Values	Opseg dozvoljenih vrednosti (minimum i maksimum) i podrazumevana (default) vrednost atributa
Characteristics	Ostale osobine: - Format (npr. dd.mm.yy) - Unit: jedinica mere - Uppercase/Lowercase: dozvoljena samo velika/mala slova - Cannot modify: vrednost atributa se ne može menjati
List of values	Tabela u kojoj se mogu navesti dozvoljene vrednosti atributa. Value je dozvoljena vrednost, a Label opis.

Slika 2.15 Kartica Standard Checks

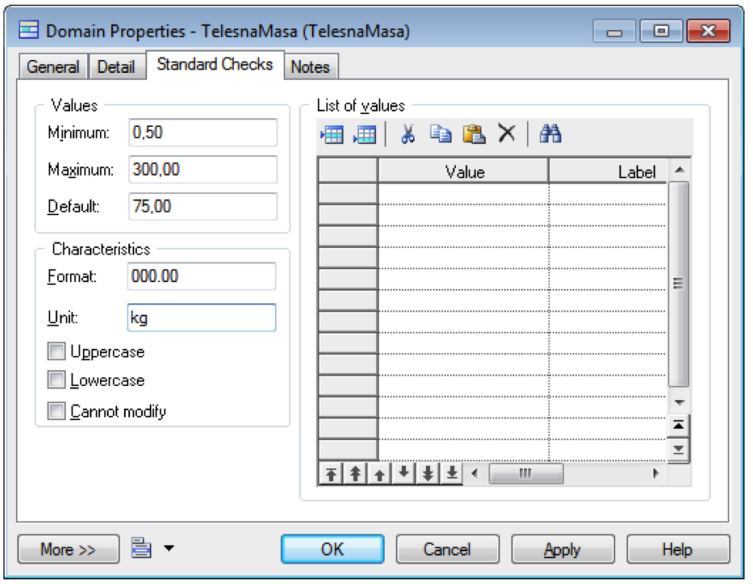
Izmene na ovoj kartici ne utiču na generisani kod, već na generisanje konceptualnog/fizičkog modela.

Domeni se u PowerDesigner-u koriste za definisanje templejta za tipove podataka. Domen predstavlja tip

podataka sa pridruženim informacijama o ograničenjima i perzistentnosti.

Dodavanje domena:

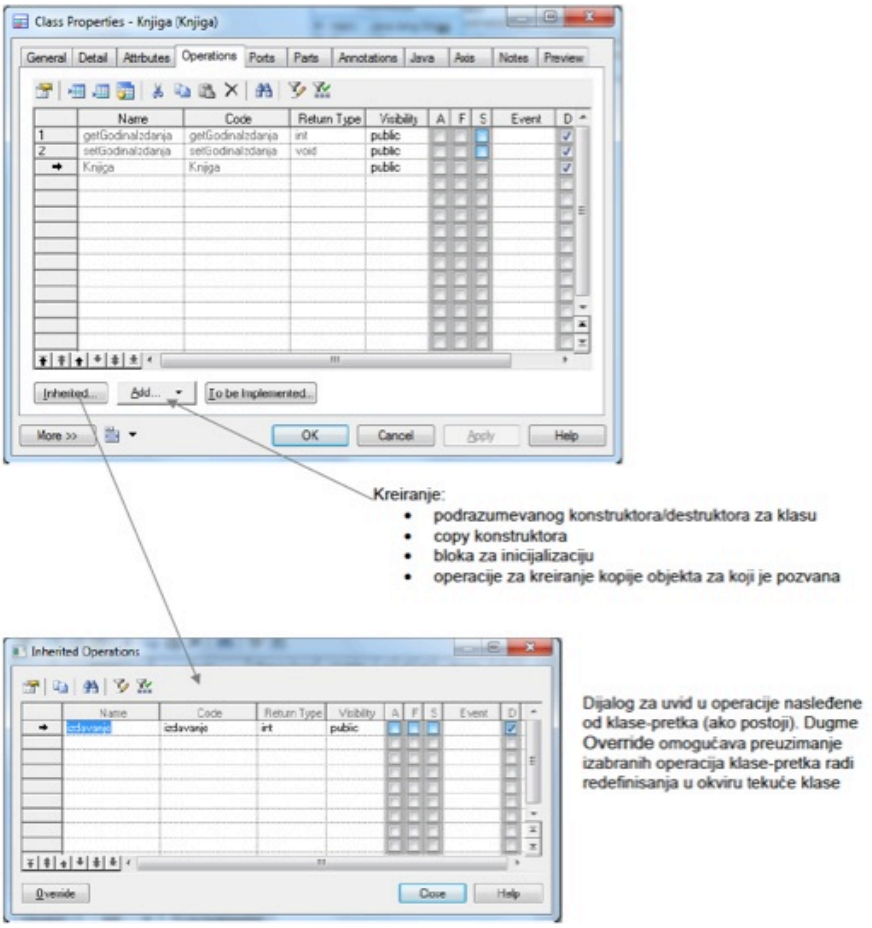
1. iz Browser-a: desni klik na model → New → Domain
2. iz liste svih domena, koja može da se otvori iz glavnog menija (Model → Domains) ili sa Details kartice bilo kojeg atributa (v. odeljak 3.4.2 Kartica Detail).



Slika 2.16 0 Definisanje domena

DEFINISANJE OPERACIJA KLASE - KARTICA OPERATIONS

Kartica "Operations" daje uvid u operacije

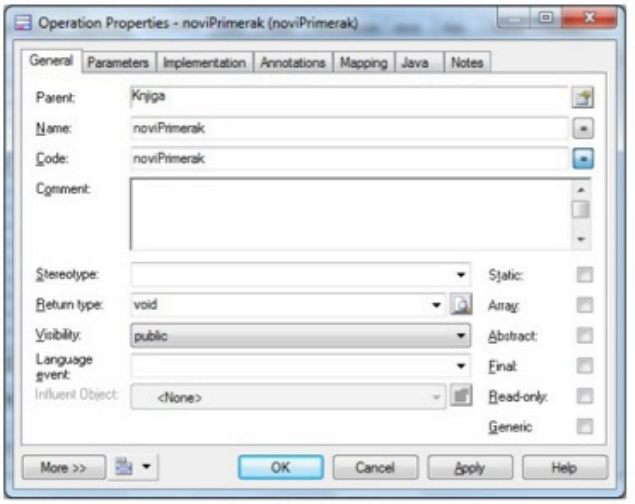


Slika 2.17 Unos operacije preko kartice Operations

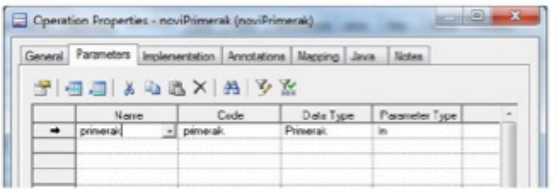
Dugme To be Implemented... omogućava uvid u operacije "nasleđene" od interfejsa (ako klasa realizuje neki interfejs), sa mogućnošću da se te operacije implementiraju. Dijalog je identičan dijalogu za uvid u
operacije nasleđene od klase-pretka, samo se umesto dugmeta Override pojavljuje dugme Implement.

KARTICE GENERAL I PARAMETERS- UNOS PODATAKA O OPERACIJI

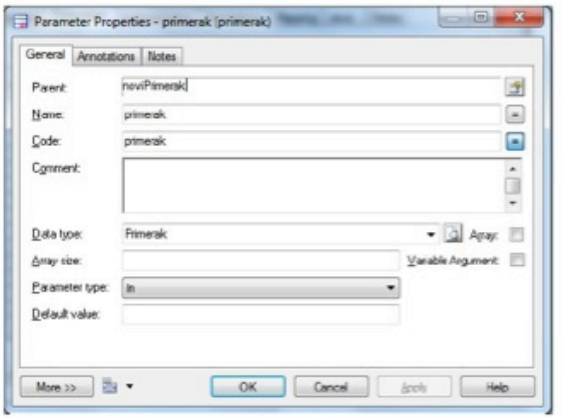
Omogućava se unos parametara operacije



Name, Code	Naziv operacije (opisni i naziv koji se prenosi u generisani kod)
Return Type	Tip rezultata operacije. Ako operacija vraća niz, ovde se navodi tip elementa niza, a treba selektovati indikator Array. Podrazumevani tip može se definisati u dijalogu Tools → Model Options.
Visibility	Vidljivost operacije: public, protected, private i package
Language event	Naziv događaja koji aktivira datu operaciju (ukoliko se operacija definiše kao reakcija na neki događaj)
Influent Object	Ukoliko je operacija redefinisana ili implementira operaciju interfejsa, ovde je navedena originalna operacija klase-pretka odnosno interfejsa.
Static	Oznaka da je operacija statička
Array	Operacija vraća niz (tip elementa niza se definiše u polju Return Type)
Abstract	Oznaka da je operacija apstraktna (apstraktne operacije ne mogu da imaju implementaciju).
Final	Oznaka da nije dozvoljeno redefinisanje operacije u okviru klasa-naslednica (ako postoje)
Read-only	Operacija ne menja objekat za koji je pozvana
Generic	Operacija sa generičkim parametrima (definišu se na kartici Generic)



Dijalog za podešavanje osobina parametra:

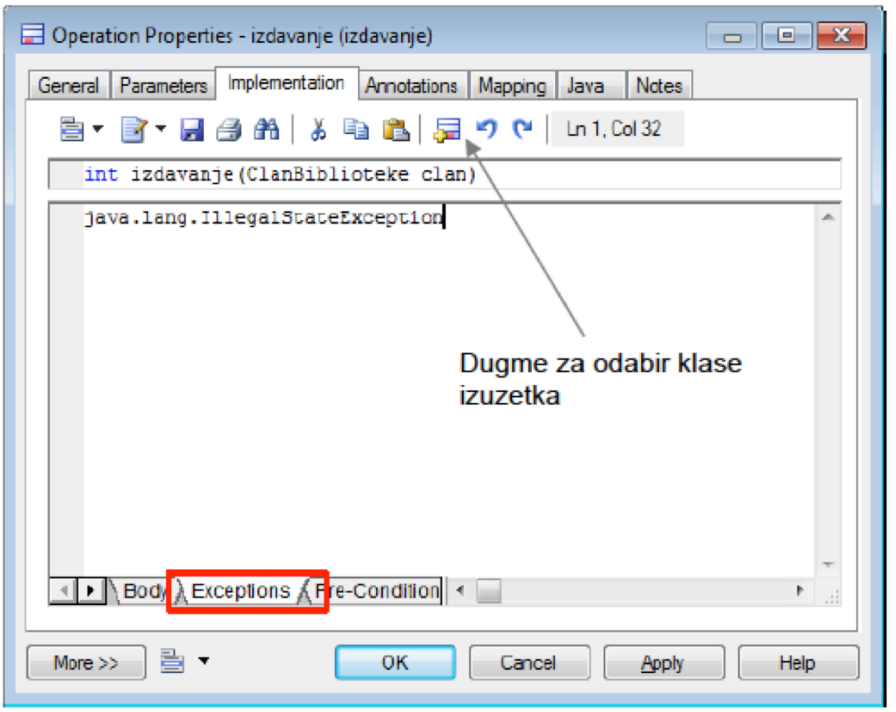


Name, Code	Naziv parametra (opisni i naziv koji se prenosi u generisani kod)
Data Type	Tip parametra. Ako je u pitanju niz, treba selektovati indikator Array. U polju Array size može se navesti tačan broj elemenata niza.
Parameter type	Vrsta parametra: ulazni, izlazni, ulazno-izlazni
Default value	Podrazumevana vrednost parametra
Variable Argument	Selektovanjem ove opcije se definiše da operacija može da primi promenljiv broj argumenata. Opcija se selektuje samo za jedan parametar metode i on mora da bude poslednji u listi parametara.

Slika 2.18 Unos parametara

KARTICA IMPLEMENTATION

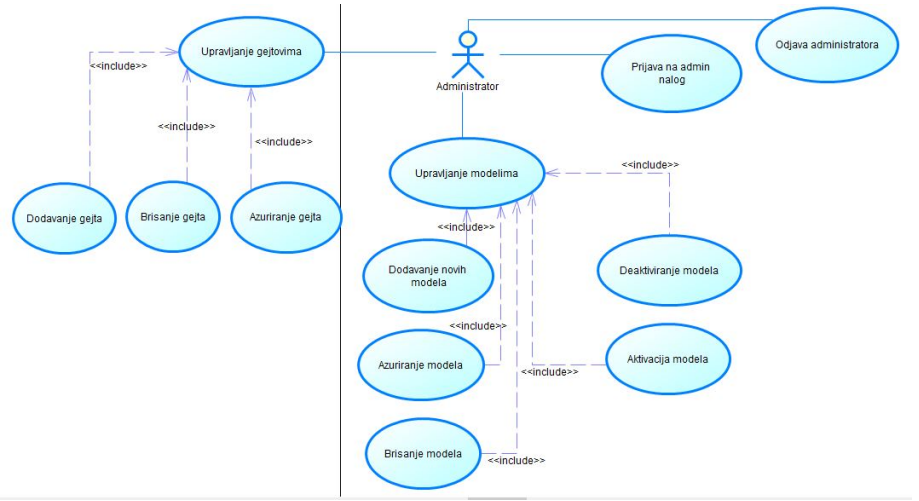
Opisuje se upis izuzetka u operacije.



Slika 2.19 Unos dodatnih podataka o operaciji

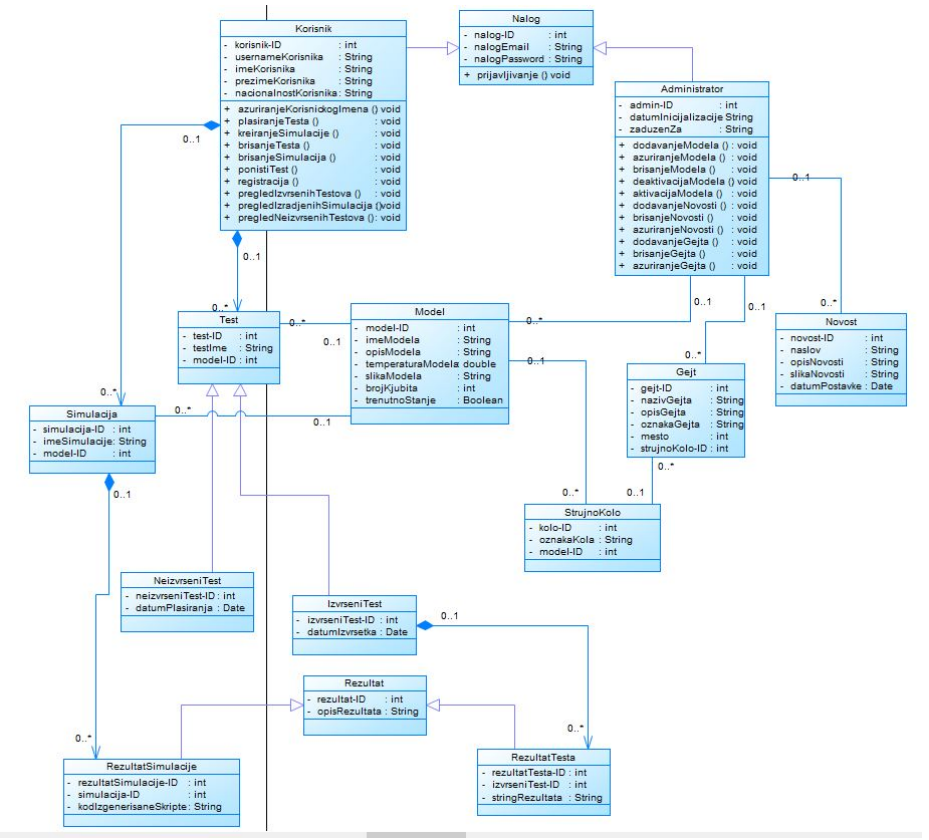
PRIMER MODELOVANIH DIJAGRAMA U POWER DESIGNER ALATU

Dati su primeri modelovanih dijagrama u Power Designer alatu.



Slika 2.20 Primer modelovanog dijagrama slučajeva korišćenja

Na slici 25 dat je primer modelovanog dijagrama slučajeva korišćenja kroz Power Designer razvojno okruženje. Na slici se može videti administrator kao korisnik sistema, slučajevi korišćenja administratora u okviru sistema i veze između korisnika i slučajeva korišćenja.



Slika 2.21 Primer modelovanog klasnog dijagrama

Na slici 26 dat je primer klasnog dijagrama modelovanog u Power Designer razvojnog okruženja. Predstavljene su klase i veze između njih.

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji se odnose na modelovanje dijagrama u Power Designer alatu.

1. Zadatak: Kreirati klasu unutar klasnog dijagrama za sistem za naplatu parkinga u Power Designer alatu. Definisati atribute klase. Definisati operacije.

2. Zadatak: Uneti podatke za definisane operacije kroz dijalog.

Poglavlje 3

UML Strukturni modeli

ULOGA STRUKTURNIH MODELA

Strukturni modeli pokazuju organizaciju softverskog sistema u obliku komponenata koje čine sistem i njihove veze.

Strukturni modeli pokazuju organizaciju softverskog sistema u obliku komponenata koje čine sistem i njihove veze. Strukturni modeli mogu da budu

1. *statički,* koji pokazuju strukturu sistema*, i*
2. *dinamički,* koji pokazuje strukturu sistema u fazi izvršenja svojih funkcija, tj. u fazi operativnog rada.

Strukturni modeli se kreiraju radi diskusije o sistemu i kada se projektuje arhitektura sistema. Pri projektovanu sistema, koriste se i drugi strukturni UML dijagrami, kao što su komponentni dijagram, dijagram pakovanja, i dijagram instalisanja sistema. Ovde će se izložiti primena dijagrama klasa u razvoju softverskog sistema.

3.1 Dijagram klasa

POJAM KLASE

Uvod i upotreba UML klasnog dijagrama, objekata, klasa i veza

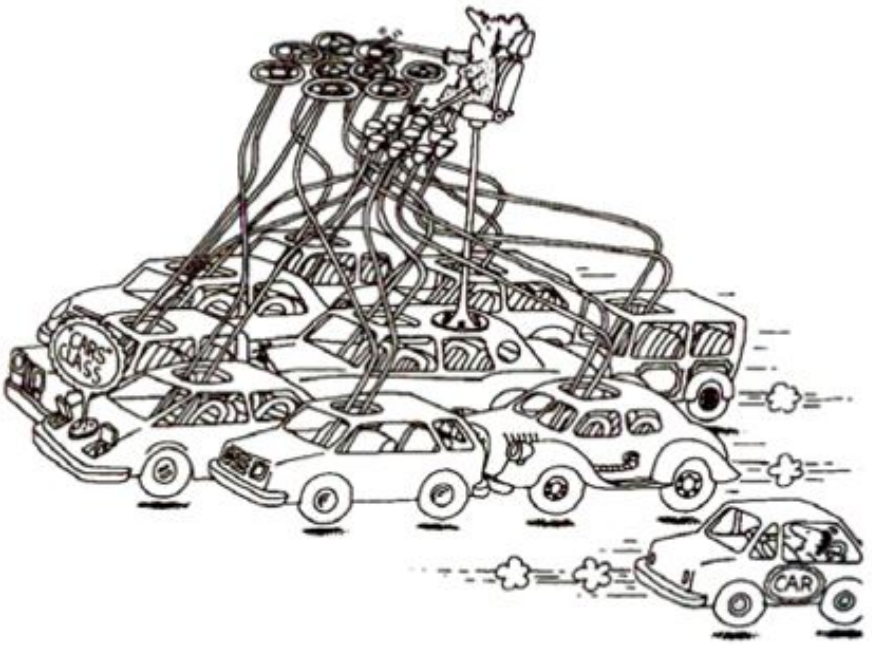
Koncept objekta i klase se međusobno stalno prepliću, tako da ne možemo govoriti o nekom objektu, a da ne pomenemo njegovu klasu. Između ova dva termina ipak, postoje bitne razlike. Dok objekat predstavlja konkretan entitet, koji postoji u vremenu i prostoru, klasa je samo apstrakcija, suština objekta. **Grady Booch** daje sledeću definiciju klase: ''Klasa je skup objekata koji imaju zajedničku strukturu i ponašanje''

Na slici 1 prikazani su automobili i njihova zajednička svojstva:

1. Kreću se na četiri točka
2. Njima vozač upravlja preko volana, i papučicama za gas, za kvačilo i za kočnicu;
3. Svi imaju menjač i brzine

Te zajedničke karakteristike svih automobila (kao objekata) svrstavaju ih u isti tip objekata, tj. u klasu ''automobil''. Jedan objekat predstavlja primerak neke klase. Pogledajte primerak

neke knjige; taj primerak ima svoje osobine: može biti sasvim nov ili pohaban; možda ste na prvoj stranici napisali svoje ime; možda pripada biblioteci, pa ima svoju jedinstvenu šifru. Sa druge strane, osnovni elementi knjige, kao što su: naslov, izdavač, autor i sadržaj su određeni opisom koji je primenljiv na svaki zaseban primerak: naslov knjige je taj i taj; izdavač je taj i taj; itd. Taj skup osobina ne definiše objekat, već klasu objekata (ponekad se koristi i termin *tip objekata*).



Slika 3.1.1 Zajednička svojstva sviha automobila određuju svojstva klase

UML SIMBOL KLASA

Prikaz simbola klasa i njegova značenja

Da bismo lakše opisali klase i objekte, kao i njihove veze koji svi zajedno čine OO modele, koristićemo njihove grafičke prikaze u skladu sa tzv. UML jezikom za OO modeliranje (UML - Unified Programming Language).

UML klasni dijagram ili dijagram klasa prikazuje skup klasa, interfejsa i njihovih relacija odnosno opisuje strukturu sistema. Klase predstavljaju abstrakciju koja određuje zajedničku strukturu i ponašanje niza objekata. Objekti su instance klasa, koje se stvore, modifikuju svoje stanje i uništavaju tokom izvođenja sistema. Svaki objekat ima stanje koje obuhvata vrednost atributa i njenih veza sa drugim objektima.

Svaka klasa sastoji se od imena, atributa i metoda(operacija). Na slici 2 prikazan je primer klase korišćenjem PowerDesigner razvojnog alata.

Kao što se može primetiti na slici jedan klasa kupac ima tri celine. Prva celina predstavlja naziv same klase(u primeru datom na slici 2 naziv klase je Kupac), drugi deo predstavlja mesto za deklarisanje svih promenljivih odnosno atributa klase i njihovih tipova podataka. U primeru datom na slici klasa Kupac ima tri atributa (ime, adresa, datumRodjenja) i oni

pre početka naziva atributa stoje odgovarajući simboli koji definišu vidljivost(pristup) samog atributa. Treća sekcija predstavlja definisanje metoda odnosno operacija (funkcija). Pre naziva metode i dalje stoje simboli za enkapsulaciju koji određuju vidljivost metode, a nakon imena same metode posle dve tačke stoji tip podatka koji ta metoda vraća. U primeru datom na slici 2 klasa Kupac ima jednu metodu izracunavanjeKredita() koja je public i koja vraća double tip podataka.

Slika 3.1.2 Prikaz UML klase kupac

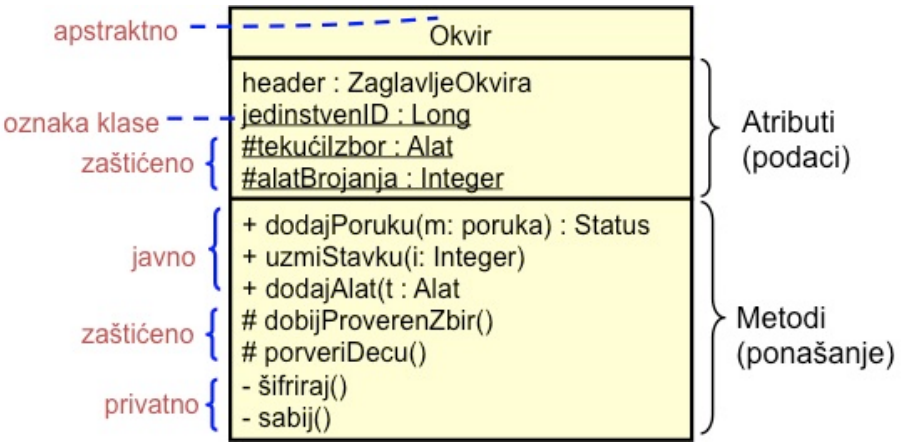
Da bi se ograničilo pravo pristupa nekom atributu ili metodu nekog objekta, koriste se tri stepena zaštićenosti, tj. kontrole pristupa:

public (+) – javan, dozvoljen pristup sa drugih objekata;

private (-) – privatan, dozvoljen pristup samo iz te klase;

protected (#) – zaštićen, dozvoljen pristup i iz podklasa ove klase

Kontrola pristupa definiše vidljivost klase, promenljive, ili metoda – od strane drugih objekata.



Slika 3.1.3 UML označavanje jedne klase

Da bi se ograničilo pravo pristupa nekom atributu ili metodu nekog objekta, koriste se tri stepena zaštićenosti, tj. kontrole pristupa:

- ***public (+)*** – javan, dozvoljen pristup sa drugih objekata;
- ***private (-)*** – privatan, dozvoljen pristup samo iz te klase;
- ***protected (#)*** – zaštićen, dozvoljen pristup i iz podklasa ove klase

Kontrola pristupa definiše vidljivost klase, promenljive, ili metoda – od strane drugih objekata.

ULOGA DIJAGRAMA KLASA

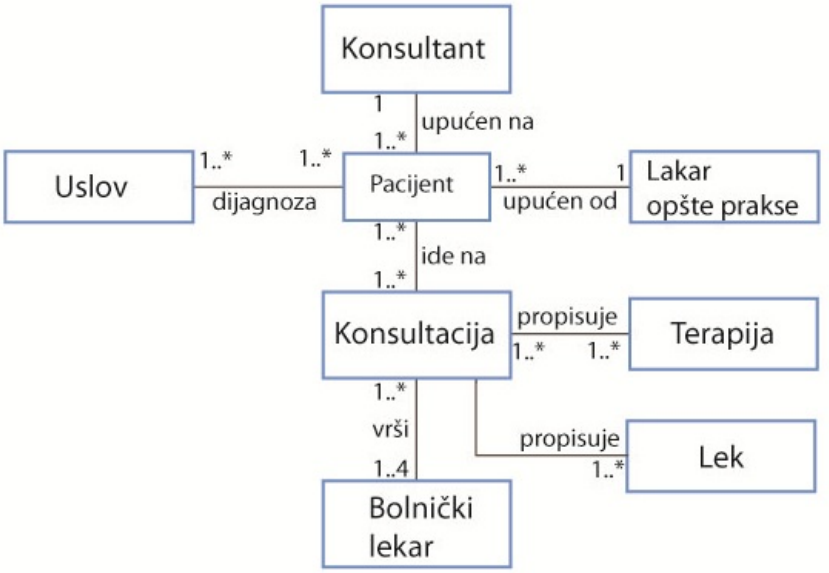
Dijagrami klasa se koriste pri razvoju modela objektno-orijentisanog softverskog sistema da bi se pokazale klase sistema i asocijacije između ovih klasa

Dijagrami klasa se koriste pri razvoju modela objektno-orijentisanog softverskog sistema da bi se pokazale klase sistema i asocijacije između ovih klasa. Zbog toga, svaka klasa ima neko saznanje o klasi sa kojom je u asocijaciji.

U početnim fazama projektovanja softvera, objekti predstavljaju nešto što postoji u realnom svetu (npr. pacijent, recept, doktor i dr.). Sa daljim razvojem softvera, tj. njegovom implementacijom (kodiranjem), moraju se definisati najčešće dodatni objekti koji obezbeđuju dodatnu funkcionalnost.

Dijagrami klasa se mogu kreirati sa različitim nivoima detaljnosti. U početku se navode samo stvarni objekti, sa svojim klasama, bez mnogih detalja koji se kasnije dodavaju, sa napretkom projektovanja sistema. Slika 1 prikazuje jedan jednostavni dijagram klasa, sa samo dve klase: Pacijent i Dosije pacijenta. Dijagram pokazuje i koliko objekta može biti povezano asocijacijom. U prikazanom dijagramu, to je asocijacija tipa 1:1, tj. samo jedan objekt može da bude u asocijaciji sa drugim. To znači da pacijent može da ima samo jedan zdravstveni dosije. Ako se u dijagram stavi (, to označava neograničen broj objekata može biti u asocijaciji na toj strani asocijacije, što je i slučaj na primeru dijagrama klasa prikazanog na slici 2 .

Slika 3.1.4 UML klase i asocijacija



Slika 3.1.5 Klase i asocijacije u MHC-PMS sistemu

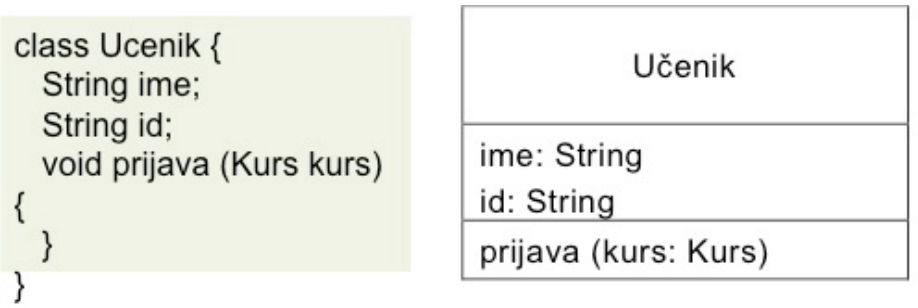
MODELOVANJE KLASE

Deo na vrhu označava ime klase, a u sredini se navode atributi, a u donjem delu – operacije

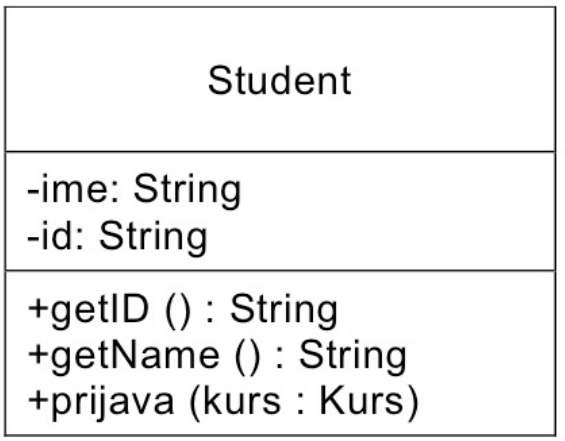
Kao što se vidi, klasa Pacijent je u vezi sa drugim klasama. Dijagram daje informaciju o tipovima asocijacije, kao i o ulogama koje klase, povezane asocijacijama, imaju. Takođe, na dijagramu se vide i nazivi asocijacija, što olakšava razumevanju svrhe postojanja navedenih asocijacija.

UML dijagram klasa se može koristiti i za modelovanje podataka u bazi podataka (tzv. semantički model baze podataka), jer je semantički model ustvari pojednostavljen dijagram klasa objekata (bez operacija/metoda), a sa atributima u vidu atributa klasa i relacijama određenim imenovanim asocijacijama između klasa objekata.

Pri modelovanju klasa, ako je to potrebno, mogu se prikazivati i dodatne informacije o klasama, kao što su atributi i metoda (operacija) klasa (slika 3). Broj atributa i operacija se tokom projektovanja povećava, sa povećanjem funkcionalnosti sistema i vezama između klasa objekata. Klasa je predstavljena pravougaonikom sa tri dela. Deo na vrhu označava ime klase, a u sredini se navode atributi, a u donjem delu – operacije (koje se nazivaju metodima u Javi, C*++ i drugim programskim jezicima).



Slika 3.1.6 Klasa Učenik u Javi i u UML modelu



Slika 3.1.7 Klasa Student

VEZE IZMEĐU OBJEKATA(KLASA)

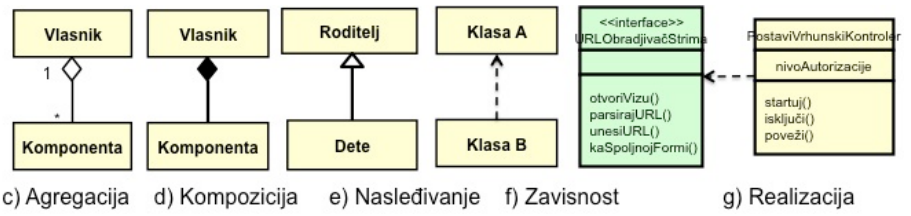
Tipovi veza na klasnom dijagramu

Objektno-orijentisani model se definiše objektima i vezama koje postoje između objekata. Kao što se može primetiti sa prethodnog primera korišćeni su neki od tipova veza (generalizacija i asocijacija). U nstavku teksta govoriće se detaljnije o vezama objekata.

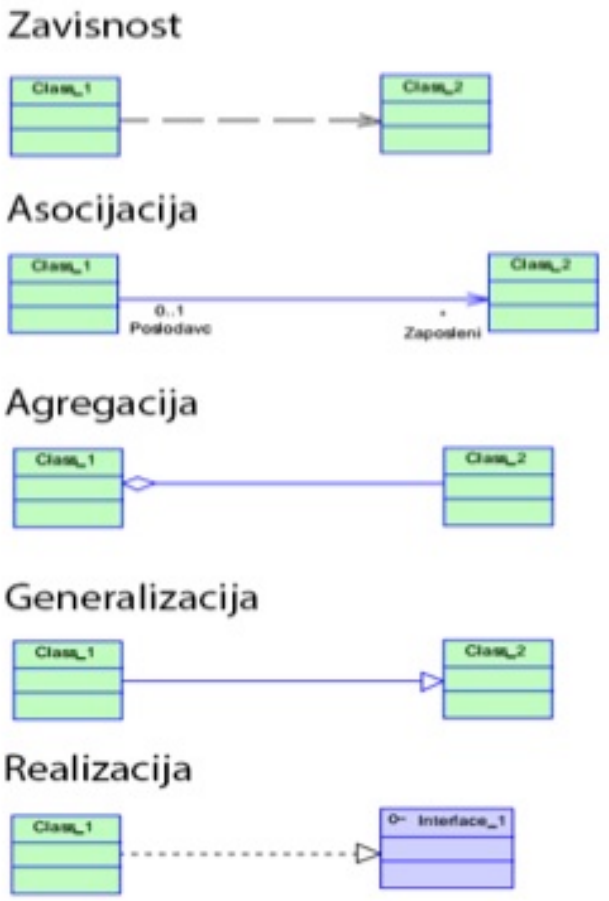
Postoji nekoliko elementarnih tipova veza (ili relacija) između objekata:

1. Zavisnost - najlabavija veza između dve klase
2. Asocijacija – jedan objekat upotrebljava servise drugog;
3. Agregacija – jedan objekat sadrži u sebi drugi objekat;
4. Kompozicija – vlasnik je odgovoran za kreiranje ili uništavanje delova objekata;
5. Generalizacija – podklasa ima sve karakteristike klase;
6. Realizacija – veza između klase i njenog interfejsa.

Svako ud ovih veza ćemo posebno objasniti.



Slika 3.1.8 UML oynačavanje različitih tipova veza između objekata



Slika 3.1.9 Generalni prikaz veza među klasama

GENERALIZACIJA

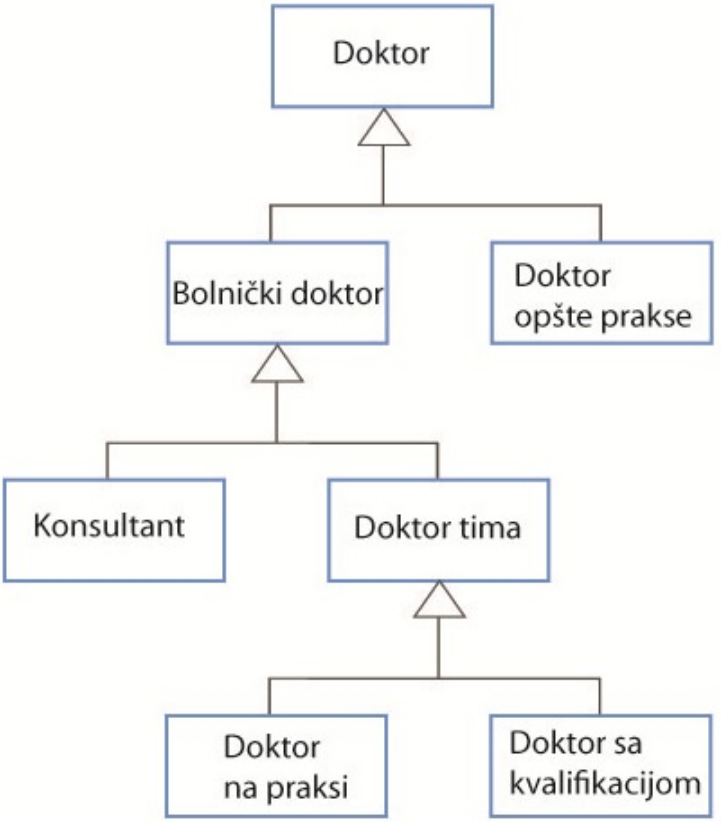
Generalizacija omogućava da se zajedničke informacije (atributi i metode) definišu i održavaju samo na jednom mestu (klasi roditelju).

Generalizacija je tehnika koja se koristi da bi se na jednostavan način rešavale složene situacije. Uopštene klase sadrže atribute i operacije (metode) koje imaju druge klase, koje su onda potklase uopštene klase. Potklase nasleđuju atribute i metode svojih klasa-roditelja (nad klasa) te se na taj način pojednostavljuje opis klasa sistema, jer su česti slučajevi kada više klasa imaju iste atribute i metode (operacije).

Generalizacija omogućava da se zajedničke informacije (atributi i metode) definišu i održavaju samo na jednom mestu (klasi roditelju). Kada se treba izvršiti neka promena, onda

se promena obavi samo na jednom mestu, umesto da se vrši u raznim klasama. U Javi (i drugim OO jezicima), generalizacija se primenjuje mehanizmom nasleđivanja koji je deo jezika.

Na slici 8 prikazana je grupa klasa povezana znacima generalizacije (bela strelica) koja ukazuje na uopštenije (generalizovane) klase. Klase na vrhu imaju zajednička svojstva (atribute i metode), a klase na dnu imaju specifična svojstva, kao i svojstva koja nasleđuju od generalizovanih klasa (roditelja). Znači, atribute i metode koje imaju generalizovane klase (pokazane sa belim strelicama) imaju i sve klase povezane ispod (sa belim strelicama) generalizovanih klasa



Slika 3.1.10 0 Hijerarhija generalizacije klasa

GENERALIZACIJA I NASLEĐIVANJE

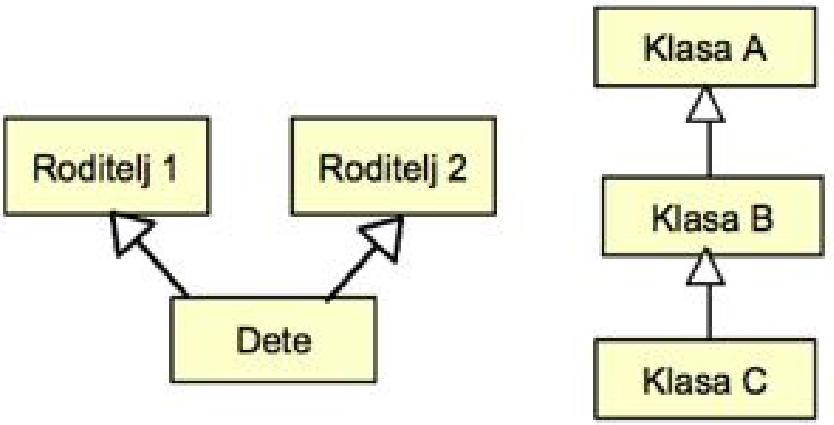
Šta je generalizacija, čemu služi i kako se primenjuje

Dve klase su povezane tako što je jedna klasa-roditelj ili **nadklasa**, a druga je klasa-dete ili **podlklasa**. Klasa-roditelj ima sva svojstva svojih klasa- dece, tj. klasa-dete nasleđuje sva svojstva (atribute i metode) klase- roditelja. Klasa-dete može da sadrži dodatne atribute i ponašanja (metode). U UML-u, ovakva veza se označava linijom sa trouglom na mestu strelice, kojim se označava klasa-roditelj kao što je prikazano na slici.

Nasleđivanje je ono što razlikuje objektno-orijentisane jezike od jezika koji su samo “bazirani na objektima”. Koristi se radi modeliranja odnosa koji postoje između različitih, ali vrlo sličnih klasa, koji na ovaj način postaju delovi iste familije. Nasleđivanje je veza tipa “kao što je”.

Klasa (ili neko njeno svojstvo) je vidljiva, kada su njena svojstva pristupačna nekoj spoljnoj klasi. Nasleđivanje kod klasa otvara još jednu mogućnost kontrole pristupa. Pristup se dozvoljava unutar klase i svim podklasama, ali ne i spoljnim klasama. To je slučaj tzv. zaštičen pristup (#, tj. protected).

Postoji i mogućnost tzv. višestrukog nasleđivanja (slika 9). To je slučaj kada jedna klasa ima više klase-roditelja (nadklasa). Tada ona nasleđuje svojstva svojih klasa-roditelj, te sa njima deli svojstva. Ponovljeno nasleđivanje je slučaj kada jedna klasa ima više klasa-roditelj, ali oni su u istoj liniji. Java ne podržava koncept višestrugog nassleđivanja.

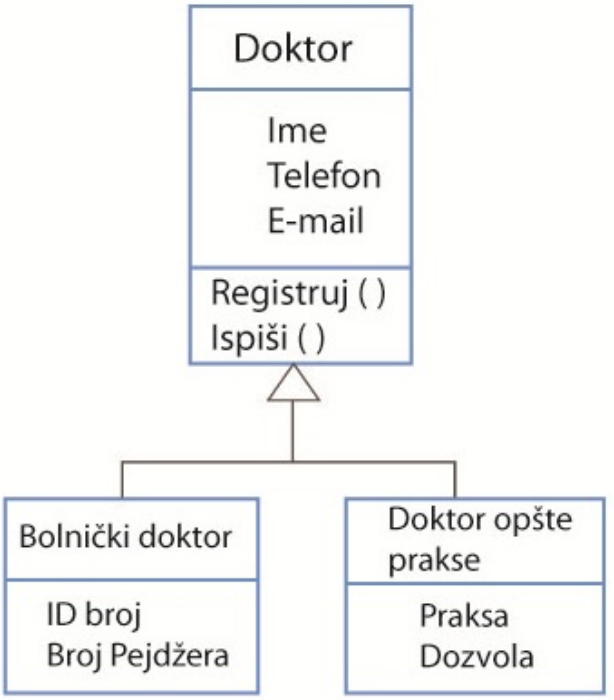


Slika 3.1.11 Primeri jednostrukog i višestrugog nasleđivanja

NASLEĐIVANJE SVOJSTAVA SUPERKLASE

Podklase nasleđuju sve atribute i metode svoje superklase.

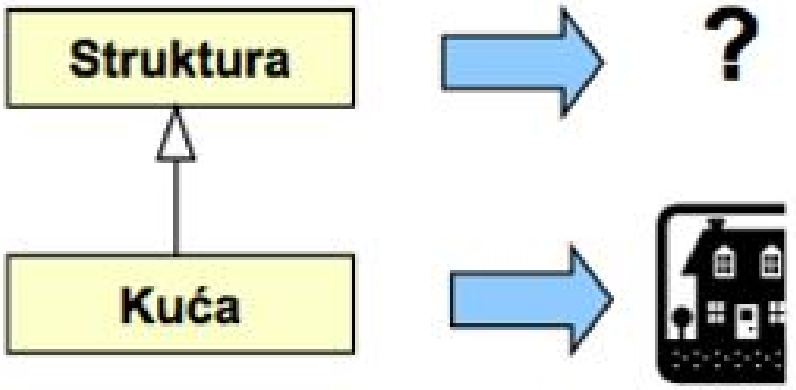
Na slici 10 je prikazana hijerarhija generalizacije sa više detalja, tj. sa prikazanim atributima i metodima. Na primer, sa metodom **registruj(),** mogu da se registruju u bazi i bolnički lekari (objekti podklase *Bolnički doktor)* i lekar opšte prakse (objekti podklase Lekar opšte prakse). To je moguće jer je metod registruj() metod definisan u generalisanoj klasi, čije atribute i metode nasleđuju i klasa *Bolnički lekar* i klasa *Lekar opšte prakse.*



APSTRAKTNE KLASE

Apstraktna klasa ne proizvodi objekte i služi za definisanje strukture (klasifikaciju) nekog koncepta

Apstraktne klase definišu vrlo široku lepezu klasa entiteta. Tako postavljen opšti tip klase se koristi kao superklasa koja sadrži svojstva svih svojih podklasa, koje imaju i svoja specifična svojstva – karakteristična za objekte koje definišu. Zbog svoje velike uopštenosti (generalisanja) apstraktna klasa nema svoje primerke u svetu objekata, tj. nema svoju instancu, objekat. Apstraktna klasa predstavlja neki koncept, a ne objekte. Na primer, mnogi različiti objekti mogu imati strukturu (velike i male zgrade, antene, statue i drugo). Struktura je koncept i ne može se predstaviti nekim objektom (slika 11).
Drugi primer apstraktne klase prikazan je na slici 8 Geometrijske figure sa nazivima trougla, pravougaonika i kruga imaju izvesna zajednička svojstva. Da se ne bi stalno prepisivala, ona se u takvim slučajevima, definišu u jednoj superklasi koju smo najavali – Oblik, tako da ih mehanizmom nasleđivanja dobijaju sva tri ova elementa. Međutim, postavlja se pitanje, kako izgleda taj geometrijski oblik koji predstavlja klasa Oblik? Odgovor je – ne izgleda nikako! Takav opšti oblik u stvarnosti ne postoji, te je očigledno da ta superklasa ne može da ima svoju instancu, tj. iz nje se ne može dobiti oblik. Zato je ona apstraktna klasa. Ona nema mogućnost da ima svoje instance (objekte). Ona samo služi za deljenje zajedničkih svojstava između njenih podklasa, i sama za sebe – ne predstavlja nikakav realan objekat.



Slika 3.1.12 Prikaz apstrakcije

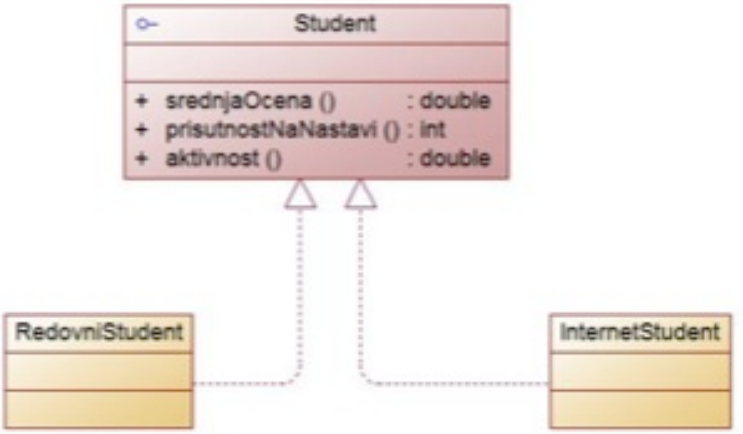
Slika 3.1.13 Primer apstraktne klase

INTERFEJS

Način kreiranja, svrha i upotreba interfejsa

Već smo pomenuli pojam interfejsa. Rekli smo da se **interfejs** nekog objekta prezentuje kako se može njime manipulisati ili kako se sa njim može komunicirati. To je sredstvo pomoću kojeg drugi objekti mogu da utiču na stanje objekta, ili da saznaju njegovo stanje. Interfejs nekog softverskog objekta se definiše programima (metodima) klase kojoj objekat pripada. To znači da interfejs sadrži popis metoda neke klase koji su javni, tj. dostupni drugim objektima. Na taj način, ove metode se mogu pozvati iz metoda koje su definisane drugim klasama.

Interfejs daje i formu koja se mora koristiti da bi se neki metod mogao da pozove. To su tzv. signature koje sadrže naziv metoda i sve parametre koje metod koristi, a koji se definišu između otvorene i zatvorene male zagrade. To je onda potpuna informacija drugim klasama i njihovim metodima o tome kako mogu da aktiviraju određeni metod klase koja koristi ovaj interfejs. Sam metod napisan u nekom programskom jeziku nalazi se u klasi (a ne u interfejsu) koja je povezana sa interfejsom. I interfejs, i veza između klase i interfejsa, su stereotipi.



Slika 3.1.14 Interfejs Student povezuje klase RedovniStudent i InternetStudent

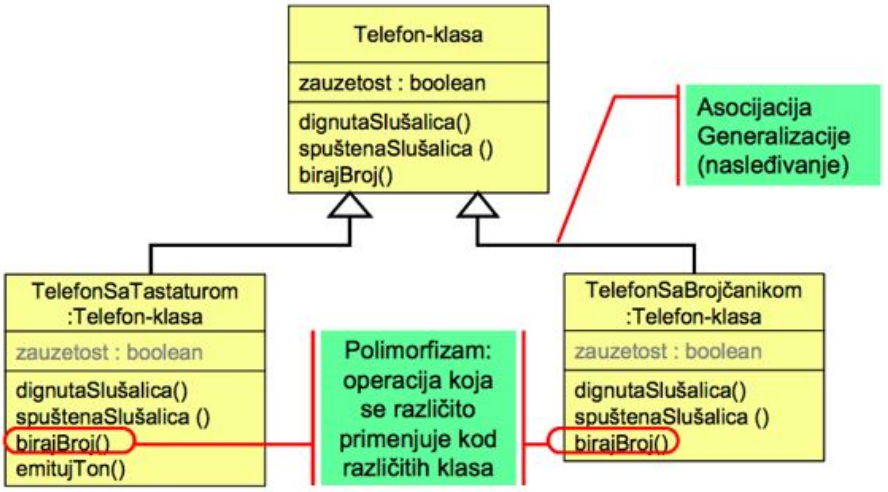
POLIMORFIZAM

Razjašnjenje pojma polimorfizma i njegovog nastanka

Polimorfizam (grčka reč) označava “mnoge forme”. U OO modeliranju označava tehniku korišćenja različitih klasa iz iste hijerarhije klasa (familije) koju može da koristi bilo koji član familije klasa. Poliformizam dozvoljava programeru da koristi bilo koju podklasu na mestu u kojem se zahteva korišćenje superklase. Suprotno, (da superklasa zameni podklasu) nije mogućno.

Na slici 14 dat je primer polimorfizma. Dve klase: TelefonSaBrojčanikom i TelefonSaTastaturom imaju zajedničku superklasu koja sadrži njihova zajednička svojstva. Da bi bilo jasnije, nasleđeni metodi super klase koja se naziva Telefon-klasa, su navedeni i u podklasama, što nije bilo neophodno, jer se njihovo prisustvo podrazumeva. Metod birajBroj() po ovom mehanizmu nasleđivanja nalazi se u obe podklase. Medjutim, implementacija tog metoda je različita. On podržava unos brojeva preko tatsature kod telefona sa tastaturom, a kod telefona sa brojčanikom – podržava unos brojeva preko brojčanika koji se okreće. Iako imaju isto ime, ova dva metoda se razlikuju u programskoj implementaciji. Iako se aktivraju na isti način, pozivom metoda birajBroj(), ova dva objekta različito reaguju.

U jednom slučaju se aktivira program koji radi sa tastaturom, a u drugom – sa okretnim brojčanikom. To je primer polimorfizma. Neki metod se poziva na identičan način, a on izaziva različta ponašanja u različitim objektima. Slično je i sa metodom print() koja se primenjuje kod štampača. Postoje različiti štampači. Ista komanda, zbog poliformizma, koji dozvoljava objektno- oriujentisan model, aktivira štampanje dokumenata na različitim računarima. Za svaki od njih, naravno, prethodno je napravljena drugačija implementacija metoda print().



Slika 3.1.15 Prikaz polimorfizma

ASOCIJACIJA

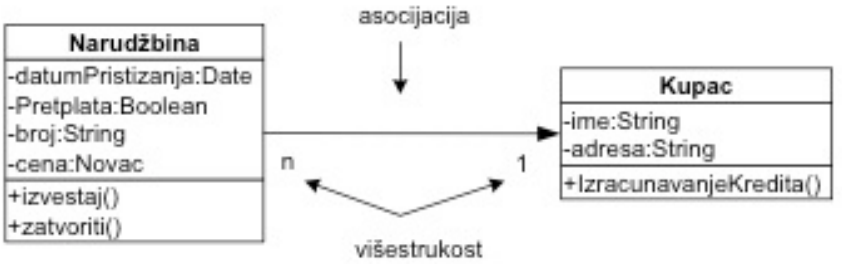
Asocijacija označava da dve klase mogu da razmenjuju poruke, odnosno, da raynenjuju servise, ako je asocijacija dvosmerna.

Kod veze tipa asocijacije jedan objekat upotrebljava servise drugog, ali ga ne poseduje. Asocijacije su relacije koje se manifestuju u vreme izvršavanja (realizacije) time što dozvoljavaju razmenu poruka među objektima. Asocijacija je veza tipa klijent-server. Upotrebljavaju se kada:
Jedan objekat upotrebljava servise drugog, ali on nije sadržan u njemu
Životni ciklus upotrebljene klase ne zavisi od klase koja je koristi.

- Upotrebljen objekat se deli podjednako sa puno drugih objekata.
- Objekti šalju informaciju drugim objektima slanjem parametara.
- Asocijacije označavaju veze objekata koji razmenjuju poruke.

Kada se šalje parametar iz objekta A, poziva se metod objekta B koji prima parametar. Definicija klase treba da odredi da takva veza postoji i koji tip informacije se može preneti (slika 15).
Na slici 16 prikazana je asocijacija dva objekta pomoću UML jezika. Asocijacija ima svoj naziv koji se upisuje na sredini linije, a uz objekte se upisuju dva podatka: (1)uloga koju objekat igra u odnosu na objekat sa kojim je povezan (2) multiplikator (ili kardinalost) koji pokazuje broj mogućih objekata koji mogu biti u vezi.

Slika 3.1.16 Asocijacija dve klase

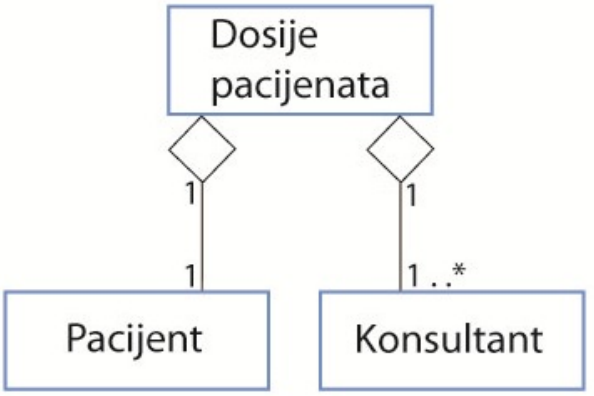


Slika 3.1.17 Primer asocijacije izmežu klasa Kupac i Narudžbenica

AGREGACIJA

Agregacija je asocijacija između klasa koja označava pripadnost jedne klase drugoj, tj. da jedan objekat jedne klase sadrži objekte drugih klasa

Objekti u realnom svetu, često sadrže različite delove (na primer, automobil). UML obezbeđuje poseban tip asocijacije između klasa koja označava pripadnost jedne klase drugoj, tj. da jedan objekat jedne klase sadrži objekte drugih klasa. Ta asocijacija se označava zarotiranim jednakostraničnim rombom (slika 17). Slika pokazuje da dosije pacijenta sadrži podatke objekta Pacijent, i puno podataka (objekata) klase Konsultacija



Slika 3.1.18 Asocijacija agregacije

AGREGACIJA I KOMPOZICIJA

Primena veze agregacije i kompoyicije na UML klasnom dijagramu

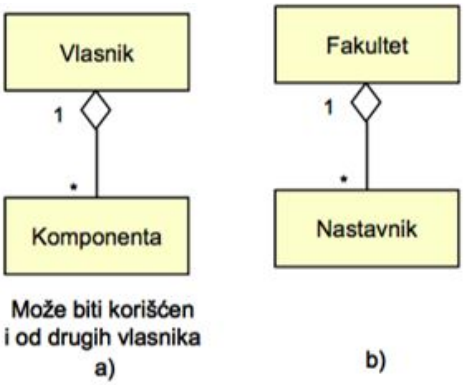
Agregacija

Agregacija se primenjuje kada jedan objekat fizički ili koncepcijski sadrži drugi.

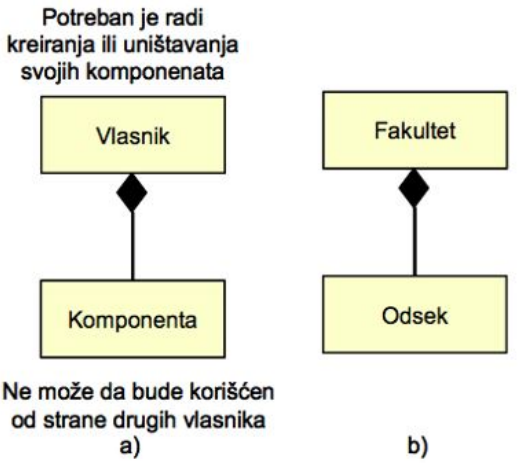
Na slici 18a prikazan je UML model veze tipa agregacija između dva objekta. Objekat-vlasnik sadrži objekat-komponentu, ali objekat- komponentu mogu istovremeno da koriste i druge klase, tj. objekat- komponenta može da bude i u vlasnitvu vise različitik klasa- vlasnika. Na slici 18b prikazan je primer ovakve veze. Na fakultetetu radi više nastavnika. Kako veza tipa agregacije dozvoljava da objekat – komponenta ima odnose i sa drugim klasama, to se jasno vidi iz ovog dijagrama da nastavnik može da radi i na nekom drugom fakultetu, tj. da deli svoje radno vreme na dva ili više fakulteta, odnosno, da ima više poslodavaca.

Kompozicija

Kompozicija je jak oblik agregacije. Upotrebljava se za aktivne objekte – objekte koji su izvori upravljanja. Ona kreira proces upravljanja, i njene komponente izvršavaju taj proces. Veza tipe agregacije se u UML obeležava izokrenutim rombom, koji je popunjen crnom bojom (slika 8). Opet, kao primer, navodimo fakultet. Fakultet može da ima više odseka, ali za razliku od nastavnika, ti odseci ne mogu da budu i delovi drugih fakulteta ili organizacija. Zato je ovde korišćen jači oblik akregacije, a to je kompozicija. Ona ne dozvoljava komponenti da bude deo nekog drugog objekta ili da ima veze tipa agregacije sa drugim objektima.



Slika 3.1.19 0 Primer agregacije

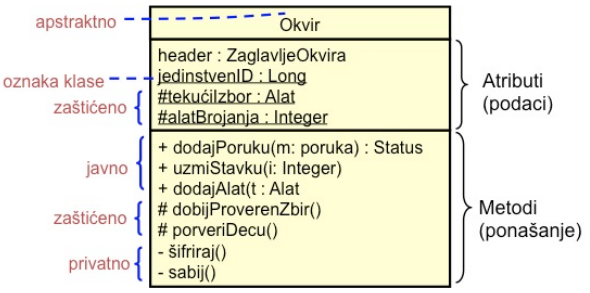


Slika 3.1.20 Primer kompozicije

REKAPITULACIJA

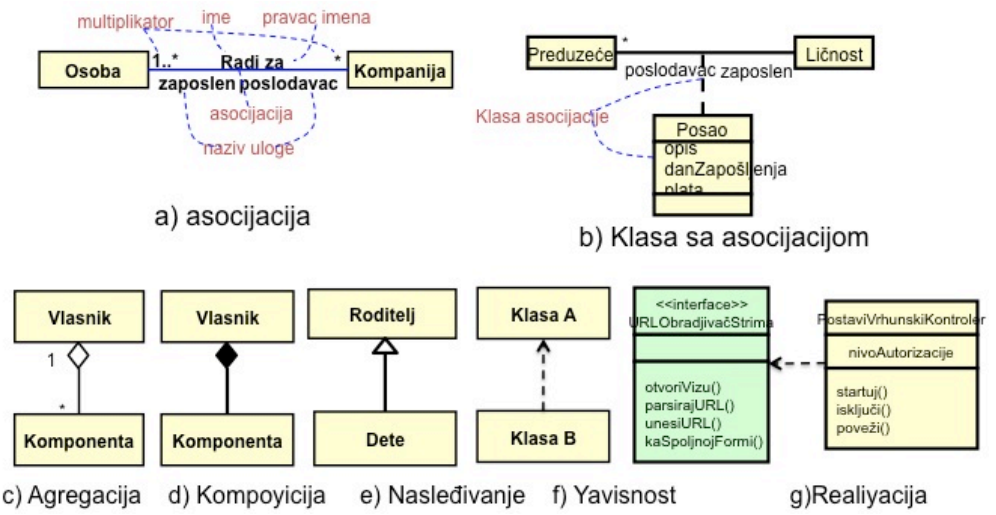
Dijagram klasa daje strukturu OO modela sistema koja pokazuje veze između klasa.

UML model klase dat je na slici 20 . On ima tri dela. Prvi pokazuje naziv klase, drugi sadrži atribute, a treći – operacije.



Ispred naziva atributa i metoda se mogu staviti simboli koji označavaju stepen zaštićenosti pristupa atributu, odnosno, metodu: slobodan, javi pristup (+), ograničen pristup (#) i privatan pristup (-). Kao što je rečeno na prethodnom predavanju, javni pristup dozvoljava pristup iz svih ostalih klasa, dok ograničeni pristup dozvoljava pristup samo iz ove klase i iz njenih podklasa. Pristup označen kao privatni, dozvoljava pristup samo iz same klase.

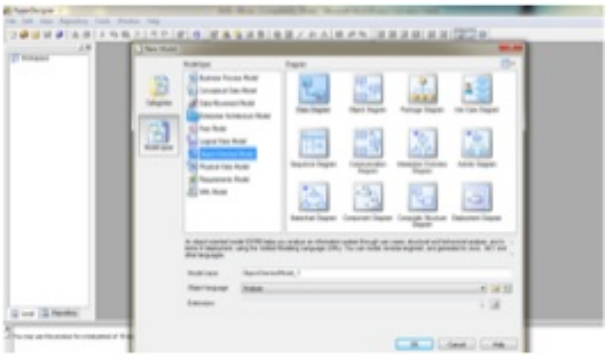
Veze između klasa (o kojima je bilo reči na prethodnom predavanju) označavaju se u UML jeziku na način prikazan na slici 21 .



POKAZNI PRIMER 3

Izbor ranije formiranih modela klase

OO modelovanje u PowerDesigner-u se vrši preko kreiranja novog OO modela kao što je prikažano na slici 3. Kao što se vidi na slici nakon odabira OO modela, prikazuje se lista svih mogućih UML dijagrama koje pripadaju OO modelima. U ovom slučaju mi biramo klasni dijagram

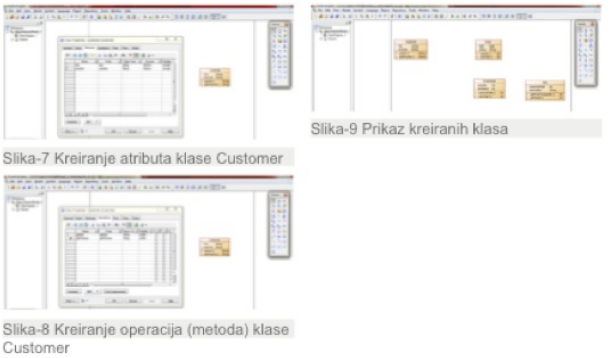


Slika 3.1.21 Primer izbora OO modela u Power Designer alatu

Slika 3.1.22 Prikaz dijagrama klasa u okviru objkeltno-orijentisanog modela

Sledeći korak je kreiranje nove klase Customer koja ima nekoliko atributa i metoda. Prilikom kreiranja klase sledi se jedan od principa OO modeliranja – Sakrivanje informacija. Generalno, atributi su private (sakriveni) i

pristupa im se spolja preko odgovarajucih metoda koji su u glavnom vidjivi van klase (public). Na sličan način dodajemo jos 3 klase: Order, OrderDetail I Item

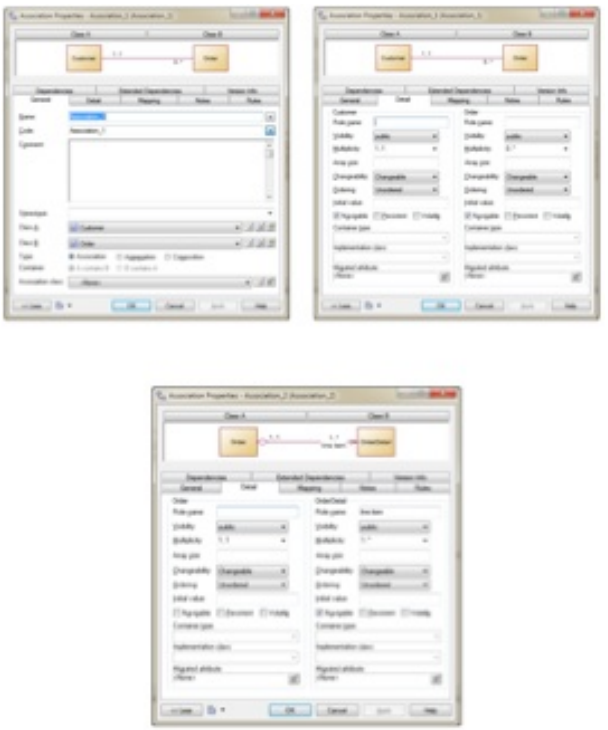


Slika 3.1.23 Kreiranje klase Customer

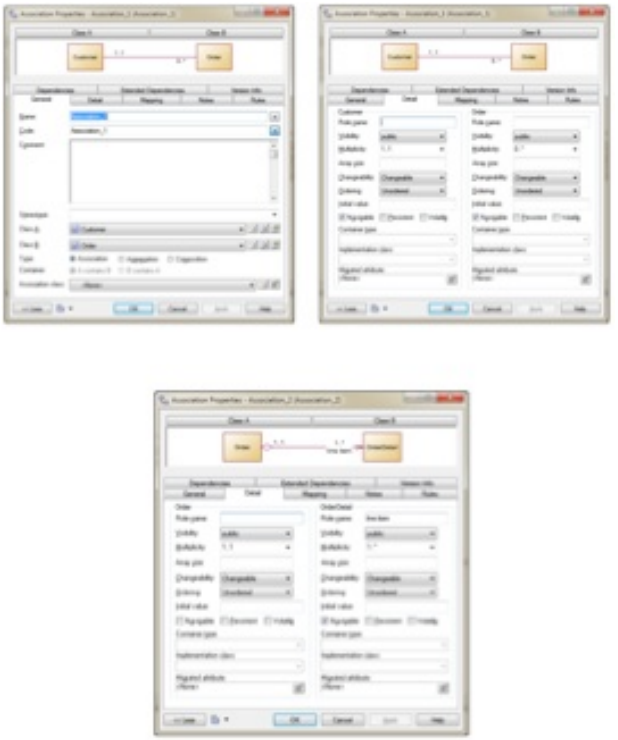
POKAZNI PRIMER 3 - NASTAVAK

Dodavanje relacija između klasa

Sad dodajemo relacije između klasa. Relacija između Customer I Order je asocijacija (slika levo), multiplicity je one-to-many I navigacija je sa obe strane (slika 6).
Order i OrderDetail su povezani agregacijom (Order sadrži OrderDetail), multiplicity je 1 -1..* i OrderDetail ima ulogu u ovoj relaciji koje se zove line item
Trenutno stanje dijagrama prikazano je na slici 7. Sad bi hteli da modelujemo način plaćanja tako što bi dodali tri nove klase: CreditCard, Cash i Check – za svaki način plaćanja.
S obzirom da sve tri klase predstavljaju tip plaćanja, tu možemo primeniti dva druga principa OO modeliranja apstrakciju i generalizaciju, tako što bi uvelu novu apstraktnu klasu Payment koja sadrži apstraktnu metodu pay() I generalizovali bi ove tri klase tj. tri tipa plaćanja.



Slika 3.1.24 Dodavanje relacija između klasa



Slika 3.1.25 Dodavanje novoh klasa CreditCard, Cash i Check

DIJAGRAMI KLASA - ZADACI ZA VEŽBU

Zadaci izrade dijagrama klasa

1. Kreirati klasu Učenik sa atributima (ime, prezime, razred, prosečna ocena). Ova klasa treba da sadrži metode promeniRazred() i promeniOcenu().
2. Kreirati roditelj klasu predavač. Pored te klase treba kreirati klasu Profesor i Asistent. Klasa predavač ima atribute (ime tipa tekst, prezime tipa tekst, Predmet tipa Predmet). Klasa predavač ima metodu predaje(Predmet p). Ova metoda prima parametar Predmet i menja stanje klase. Klasa Asistent ima sledeće atribute (doktorant tipa boolean) i metode upisaoDoktorske() – postavlja stanje promenljive na true i metodu doktorirao() – menja stanje promenljive na false. Klasa profesor ima atribut (k poeni ceo broj)
3. Kreirati interfejs student koji ima definisane sledeće metode (srednjaOcena(), prisutnostNaNastavi(), aktivnost()). Interfejs student implementiraju dve klase RedovniStudent i InternetStudent. Klasa redovni sutdent ima svoju metodu prisutnost() i atribut prisutan tipa boolean, a klasa InternetStudent ima metodu zalaganje koja vraća ukupan broj zalaganja.
4. Napraviti dijagram UML klasa za slučaj rezervacije hotela. Napomena: Studentu se ostavlja da sam definiše strukturu i veličinu ovog dijagrama. Dijagram mora imati minimum 4 klase.
5. Napraviti UML klasni dijagram za sistem magacina. Postoje ulazne kalkulacije artikli i izlazne fakture. Razraditi sistem po želji. Minimum 3 klase.

Za 4 i 5 zadatak podeliti klase po paketima. Studentu je na raspolaganju da definiše raspored po paketima. Cilj je da student savlada i kreiranje paketa u okviru PowerDesigner-a.

3.2 Dijagram objekata

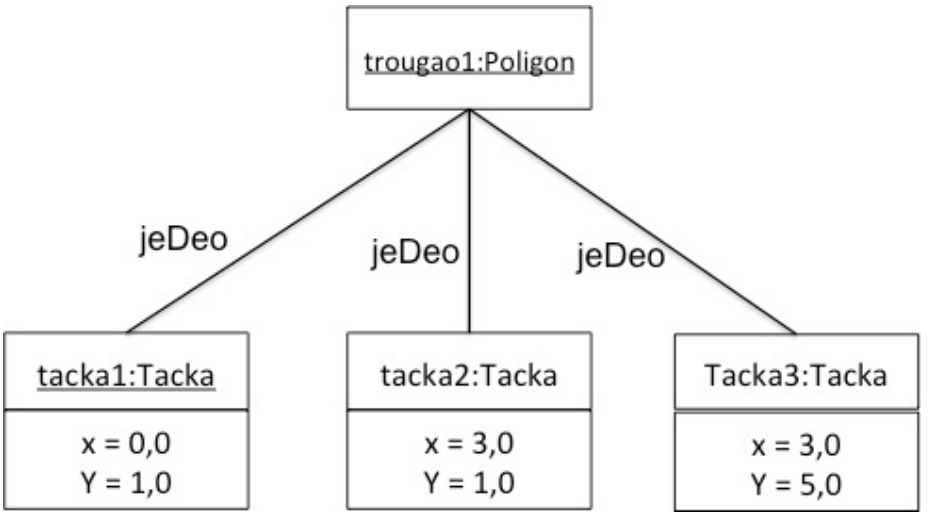
DIJAGRAM OBJEKATA SISTEMA

Dijagram objeka prikazuje objekte sistema.

Dijagram objekata daje sliku sistema u jednom vremenskom trenutku. Dijagram objekata sadrži objekte. Koristi se za prikazivanje komplikovanih struktura podataka, ili za prikaz ponašanja sistema priyaivanjem njegovih uzastopnih dijagrama objekata, kao trenutnog prikaya sistema.

Prikaz sistema nije model sistema, već samo primer izgleda sistema u pojedinim trenucima. To je statički pogled na sistem, koji pokazuje objekte, kao primene (instance) klasa.

Slika 1 pokazuje dojagram objekata sa četiri objekta. Tri definišu tačke, a četvrti-konkretan trougao.
Dijagrami objekata (engl. Object Diagram) objašnjavaju i ilustruju skup dijagrama klasa i predstavljaju slike objekata i njihovih veza u specificnom trenutku vremena. Znači, dijagrami objekata se sastoje od objekata i njihovih veza. Veze predstavljaju načine na koji su objekti povezivani i udruživani.



Slika 3.2.1 Primer objekta trougao1 klase Poligon povezan sa tro tačke, kao posebnih objekata

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji obuhvataju dijagrame objekata.

1. Zadatak: Modelovati dijagram objekata za sistem za elektronsko bankarstvo. Identifikovati minimum četiri objekta i prikazati veze između njih.

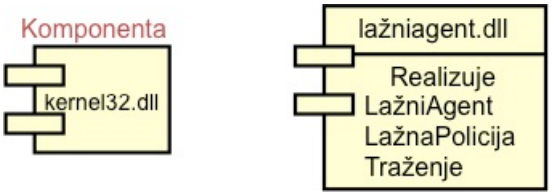
2. Zadatak: Modelovati dijagram objekata za aplikaciju za prepoznavanje kuhinjskog posuđa: Aplikacija bi radila na mobilnim uređajima sa Android operativnim sistemom i davala bi rezultate na osnovu uslikane fotografije. Analizom fotografije obrađivali bi se rezultati i korisniku slala informacija o prepoznatom kuhinjskom posuđu. Za svakog korisnika bi se u bazi čuvale informacije o količini posuđa koje poseduje, tj. o njegovoj virtuelnoj kuhinji. Nakon uslikavanja novog posudja korisnik ima opciju da sačuva nov element u svoju virtuelnu kuhinju ili da odustane. Identifikovati sve objekte koji se nalaze unutar aplikacije.

3.3 Dijagram komponenata

KOMPONENTA

Komponenta je fizički i zamenljivi deo sistema koji sadrži i realizuje skup interfejsa

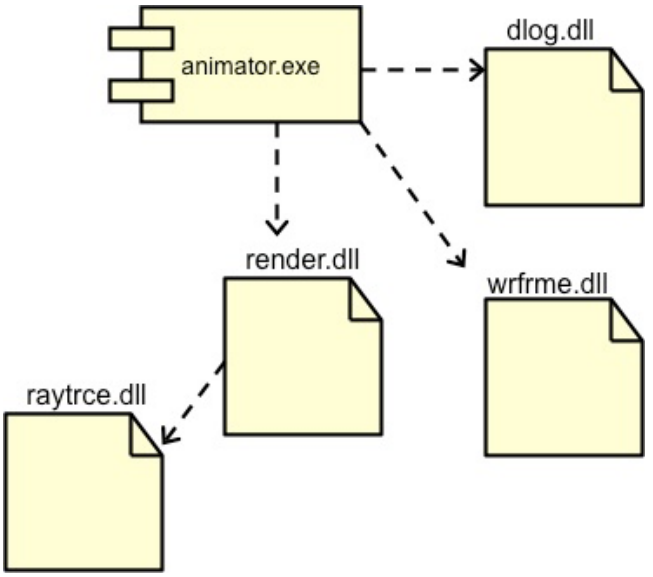
Komponenta je fizički i zamenljivi deo sistema koji sadrži i realizuje skup interfejsa (slika 1). Dok klase predstavljaju logičke apstrakcije, komponente predstavljaju fizičke stvari koje se nalaze u računarima. Komponente predstavljaju fizičke pakete logičkih komponenata sa različitim nivoima apstrakcije. Komponente imaju samo operacije kojima se prilazi preko njihovih interfejsa.



Slika 3.3.1 UML model komponente

Obično jedna komponenta predstavlja neki izvršni program koji je sastavljen od više elementa, i njen sadržaj se može prikazati preko dijagrama, kao što je dijagram na slici 2 . Dijagram pokazuje da se izvršni program pod nazivom animator.exe sastoji od četri softverska elementa u izvršnom obliku: dlog.dll, render.dll, wrfmre.dll, i ravtrce.dll. Na ovaj način se u procesu razvoja softvera definiše konfiguracija nekog izvršnog programa

.

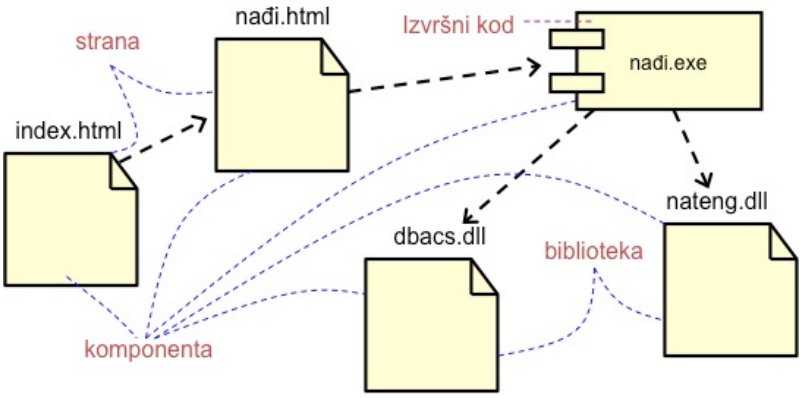


Slika 3.3.2 Konfiguracija komponente animator.exe

STRUKTURA JEDNE KOMPONENTE

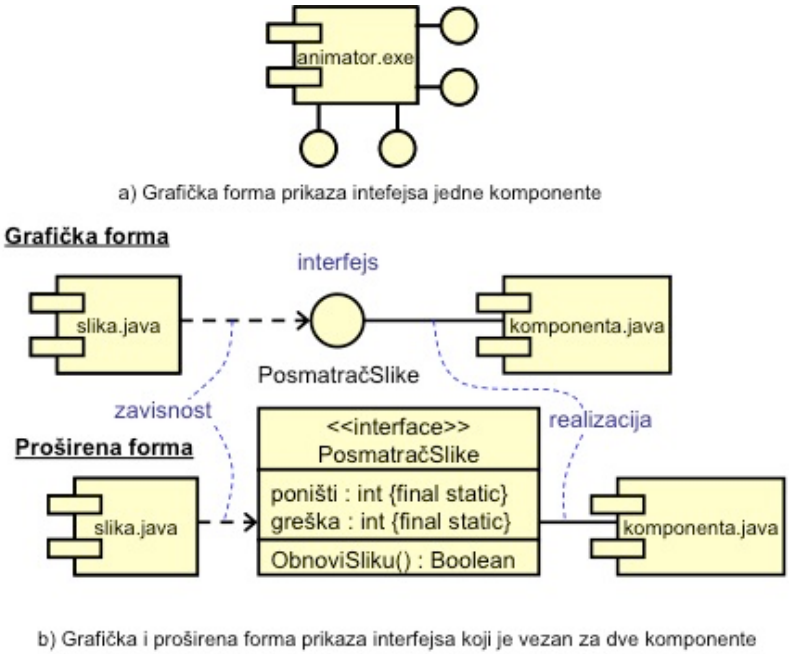
Komponenta sistema pokazuje organizaciju i zavisnost unutar jednog skupa komponenata

UML komponente se upotrebljavaju za modeliranje fizičkih aspekata objektno-orijentisanih sistema. Komponenta sistema pokazuje organizaciju i zavisnost unutar jednog skupa komponenata (slika 3). Isprekidana linija sa strelicom pokazuje vezu zavisnosti između UML komponente i njenih podkomponenti (elemenata). Te podkomponente, kao što se vidi, mogu biti bibliotekarski izvršni programi (.dll), mogu biti web stranice u HTML formatu (.html) ili nešto drugo



Slika 3.3.3 Organizacija i veze elemenata koji čine jednu UML komponenta

Komponente komuniciraju sa svojim okruženjem preko interfejsa. Interfejs je kolekcija operacija (tj. usluga) koje se upotrebljavaju radi specifikacije usluga klase ili komponente. Komponente realizuju interfejse (slika 4)



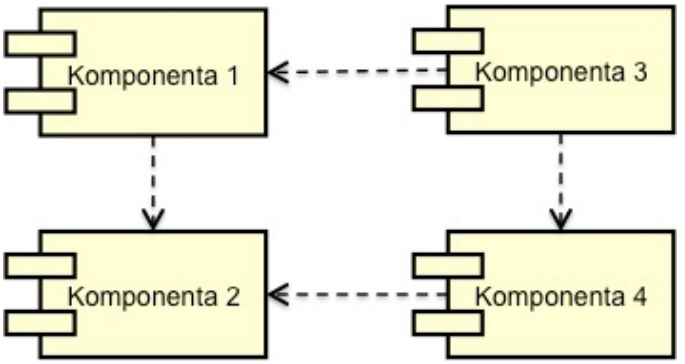
Slika 3.3.4 UML komponente realizuju interfejse

UML DIJAGRAMI KOMPONENATA

Dijagram komponenata predstavlja jedan fizički pogled na sistem.

Dijagram komponenata predstavlja jedan fizički pogled na sistem. Njegova svrha je da pokaže zavisnost između softvera i drugih softverskih komponenata (npr., softverskih biblioteka) u sistemu. Dijagaram komponenti se može prikazati na vrlo visokom nivou, na kojem ga čine samo najveće komponente, ali se može prikazati i na nižem nivou, na nivou paketa komponenti.

Kao što je već rečeno u predavanju broj 2, interfejs je kolekcija operacija (tj. usluga) koje se upotrebljavaju radi specifikacije usluga klase ili komponente. Komponente realizuju interfejse (slika 4).



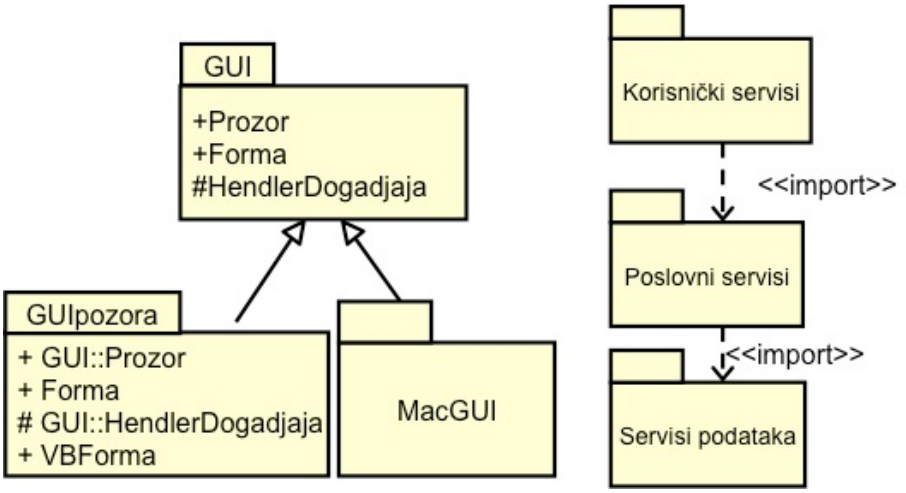
Slika 3.3.5 Dijagram komponenti

UML PAKETI

Paket je grupisanje elemenata od kojih neki sadrže specifikaciju ponašanja, te na taj način utiču na ponašanje celog paketa

Paket je grupisanje elemenata od kojih neki sadrže specifikaciju ponašanja, te na taj način utiču na ponašanje celog paketa. Paketi predstavljaju jedan opšti mehanizam za organizovanje elemenata u grupe. Na slici 6 .a prikazana su tri pakteta. Paket GUI, koji sadrži komponente koje se nalaze u nekom grafičkom korisničkom interfejsu koji je nezavisan od operativnog sistema. Druga dva paketa, WndowsGUI i i MacGUI sadrže softverske komponente koje su zavisne od operativnog sistema. U prvom paketu WindowsGUI se nalaze komponente koje se isporučuju ako sistem treba da radi korišćenjem MS Windows operativnog sistema; MacGUI je paket koji sadrži softverske komponente koje su neophodne pri korišćenju Macintosh operativnog sistema. Kao što se vidi, ova dva paketa nasleđuju one komponente koje ne zavise od operativnog sistema, a koje su sadržane u paketu GUI.

Na slici 6 .b prikazan je drugi primer korišćenja paketa u kojem su tri paketa međusobno povezana vezama zavisnosti tipa <<import>>. Prema tome, paketi su kontejneri softverskih komponenti, slično kao što su folderi ili direktorijumi kod operativnih sistema računara



Slika 3.3.6 Korišćenje UML paketa za prikaz organizacije grupisanja komponenti

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji obuhvataju dijagrame komponenti.

1. Zadatak: Napraviti dijagram komponenti za sistem za upravljanje vremenom i dnevnicima zaposlenih u jednoj firmi. Sistem ima za cilj da omoguća zakazivanje sastanaka grupe zaposlenih. Kada je potrebno organizovati sastanak određene grupe zaposlenih, sistem analizira njihove elektronske rokovnike, i nalazi vremenski slot u kome su svi oni slobodni i zakazuje sastanak, obaveštavajući sve članove grupe. Ukoliko ne može da nađe slobodan vremenski slot, sistem pregovara sa korisnicima tako da promenama u svojim rokovnicima, omoguće zajednički vremenski slot za sastanak.

2. Zadatak: Napraviti dijagram komponenti za sistem za podršku autobuskog prevoza putnika. Administrator sistema predstavlja i samog direktora firme koji ima mogućnost upravljanja svim podacima i izmena istih. Radnici imaju uvid u svoj raspored vožnji, raspored vozila koja koriste, imaju mogućnost štikliranja obavljenih poslova i imaju mogućnost nekih manjih izmena. Dok korisnici mogu da se informišu o vremenu polaska, rutama, vremenu trajanja, ceni i ostalo što je bitno jednom korisniku da zna za svoje putovanje.

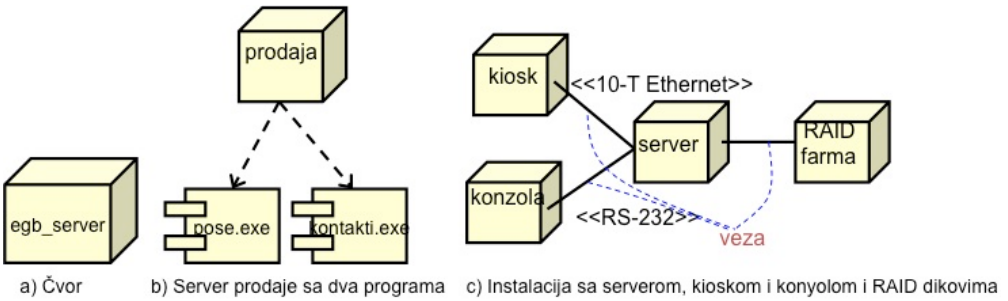
3.4 Dijagram instalacije

DIJAGRAM INSTALACIJE-KONFIGURACIJA RAČUNARSKOG SISTEMA

Dijagram instalacije predstavlja fizički instalaciju, tj. konfiguraciju nekog računarskog sistema. Čvor predstavlja fizičku instalaciju komponenata.

Dijagram instalacije (engl., the deployment diagram) predstavlja fizički instalaciju, tj. konfiguraciju nekog računarskog sistema. On pokazuje gde treba da se instaliraju softverske komponente, tj. na kojim hardverskim komponentama računarskog sistema treba da se instaliraju, kako bi onda mogle da rade i da međusobno komuniciraju. Kako su ovi dijagrami modela računarskog sistema (tj. njegovog softvera i hardvera) u fazi njegovog izvršavanja (realizacije), to je on od posebnog interesa za ljude koji stavljaju sistem u pogon i koji ga kasnije koriste u svakodnevnom radu. (slika 1).

Čvor (engl., node) je fizički element koji postoji u vreme realizacije programa i predstavlja računarski resurs; on u načelu ima neku vrstu memorije, a često i svojstva obrade podataka. Čvor predstavlja fizičku instalaciju komponenata.



Slika 3.4.1 Primeri UML dijagrama instalacije

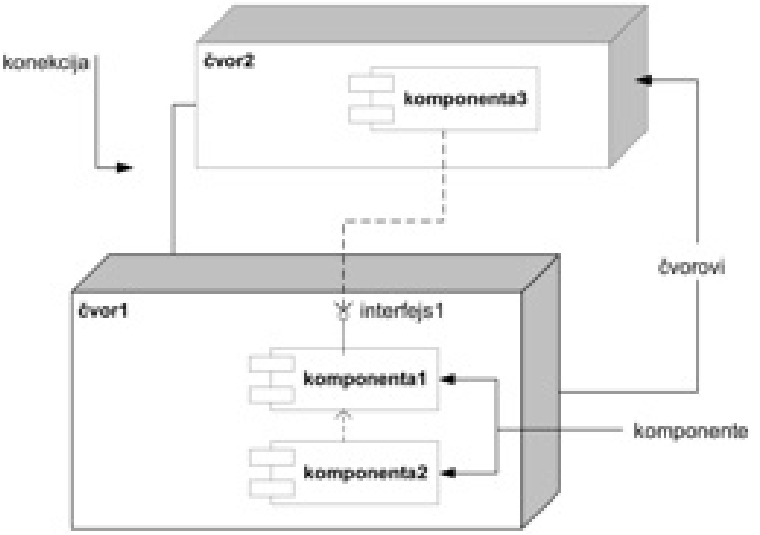
Oznake u dijagramu instalacije obuhvataju simbole koji se koriste i kod dijagrama komponenata, uz nekoliko razlika koje se pre svega odnose na prikazivanje čvora, ili čvorišta sistema. Čvor predstavlja ili fizičku mašinu (npr. server računar) ili neku virtualnu mašinu. UML model čvora predstavlja jedan trodimenzionalni paralelopiped koji sadrži ime čvora na svom vrhu, pri čemu se koristi sledeći način označavanja: Ime instande : tip instance. Na slici 1 .a prikazan je grafički simbol nekog čvora, a na slikama 1 .b i 1 .c prikazana su dva primera dijagrama instalacije: u prvom vidimo da se u čvoru (tj. serveru) pod nazivom „prodaja“

nalaze dva izvršna programa (pose.exe i kontakti.exe); u drugom dijagramu vidimo jednu hardversku konfiguraciju računarskog sistema.

PRIMER DIJAGRAMA INSTALACIJE

Dijagram instalacije daje raspored hardverskih i komponenti računarskog sistema. Svaki čvor sadrži jednu ili više komponenti softverskog sistema.

Na slici 2 prikazan je primer jednog dijagrama instalacije koji sadrći dva čvora. Čvor 1 sadrži dve komponente, a čvor 3 - trežu. Ovaj dijagram instalacije je konbinovan sa dijah+gramom komponenti, tako da pokayuje položaj komponenata po čvorovima.



Slika 3.4.2 Dijagram instalacije (rasporeda) kombinovan sa dijagramom kmponenata

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji obuhvataju dijagrame instalacije.

1. Zadatak: Osmisliti i napraviti dijagram instalacije za sistem za podršku rada hotela. Sistem će recepcioneru omogućiti da se prijavi sa svojim korisničkim imenom i lozinkom kako bi pristupio bazi podataka hotela. U bazi su smešteni podaci o gostima.
Moći će da unosi u sistem nove goste i da uklanja goste kojima se boravak u hotelu završio. Dodavanje gosta će biti jednostavno. Recepcioner izabere datum dolaska i datum odlaska gosta, zatim upisuje njegove osnovne podatke i u dogovoru sa gostom rezerviše sobu. Sobe mogu biti sa klimom i bez klime. Takođe sobe se mogu razlikovati po broju kreveta i tipu kreveta. Cene za jednu noć provedenu u hotelu se razlikuju u zavisnosti od izabrane sobe. Za VIP goste moći će da se naruči prevoz, tj. personalni vozač za određeni vremenski period.
2. Zadatak: Osmisliti i napraviti dijagram instalacije za aplikaciju za zakazivanje termina. Aplikacija ima funkciju da automatizuje zakazivanje termina korisnicima kod svog omiljenog frizera. Korisnik ima pregled termina kod svog omiljenog frizera gde vidi pregled slobodnih i zakazanih termina kod frizera. Korisnik dalje odabira jedan od slobodnih termina koji mu odgovara i šalje zahtev za zakazivanje termina. Frizer dobija obaveštenje o zahtevu korisnika

i može da odgovori na njega – da prihvati ili da odbije pružanje usluge. U slučaju prihvatanja pružanja usluge termin prelazi iz stanja slobodan u zakazan, a korisnik dobija poruku o uspešno zakazanom terminu. U slučaju odbijanja zahteva za pružanje usluge, korisnik dobija poruku o nemogućnosti zakazivanja sa opcionim dodatnim tekstom o obrazloženju.

Poglavlje 4

UML Modeli ponašanja

ŠTA SU MODELI PONAŠANJA?

Modeli ponašanja ili interakcija su modeli koji modeluju dinamičko ponašanje sistema kada radi. Oni opisuju interakcije objekata sa okruženjem i sa objektima unutar sistema.

Modeli ponašanja su modeli koji modeluju dinamičko ponašanje sistema kada radi. Oni pokazuju kako se sistem ponaša (reaguje) na neki podsticaj iz svog okruženja. Postoje dva tipa podsticaja:

1. ***Podaci:*** *Dolazak podataka koje sistem treba da obradi.*
2. ***Događaji****: Neki događaj se desi i on aktivira obradu sistema. Događaji mogu da imaju pridodate podatke, ali to nije uvek slučaj.*

Mnogi poslovni sistemi su vođeni podacima. Pri unosu podataka, oni počinju njihovu obradu, što podrazumeva izvršavanje definisanih akcija po rednom redosledu. (npr. priprema računa za telefonske pretplatnike). S druge strane, sistemi koji rade u realnom vremenu, su vođeni događajima, jer reaguju na razne događaje koji dolaze iz njihovog okruženja.
Svi sistemi koriste neki vid interakcije sa svojim okruženjem. To mogu biti interakcije sa korisnikom sistema, interakcije sa drugim sistemima ili interakcije između komponenata sistema. **Modelovanje interakcija** pomaže u utvrđivanju zahteva korisnika. Interakcija sa drugim sistemima može da ukaže na probleme komuniciranja sa drugim sistemima. Interakcije među komponentama sistema pomaže razumevanju rada sistema, i to u skladu sa zahtevanim performansama sistema i zavisnosti

Za potrebe interakcije, najčešće se koriste sledeći **modeli interakcije**:

1. **Modeli slučajeva korišćenja**, koji se najviše koriste za interakcije sa spoljnim akterima sistema,
2. **Sekvencijalni dijagrami** koji se koriste za modelovanje interakcija među komponentama sistema, mada se mogu uključiti i spoljni akteri
3. **Dijagrami stanja** koji opisuju stanja nekog objekta i događaje kojiutiču na ta stanja.
4. **Dijagrami aktivnosti** . koji opisuju tok aktivnosti koje realizuje softverski sistem.

Ovi modeli modeluju sistem na različitim nivoima detaljnosti. Detaljne interakcije se modeluju sekvencijalnim dijagramima, a uopštenije i apstraktnije interakcije – dijagramima slučajeva korišćenja.

4.1 Slučajevi korišćenja sistema

SLUČAJEVI KORIŠĆENJA

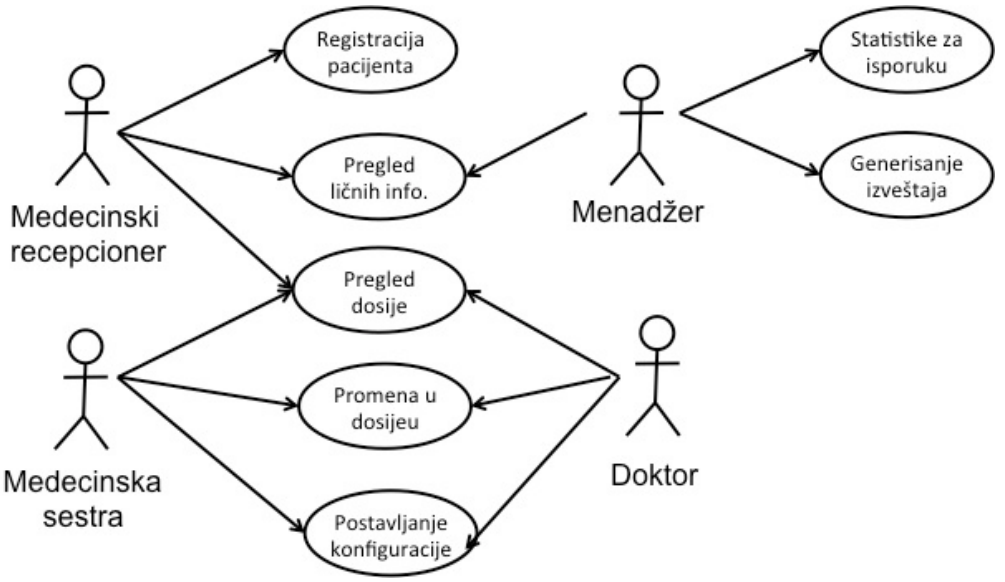
Slučaj korišćenja utvrđuje učesnike (koji se u terminologiji UML nazivaju akterima) koji učestvuju u interakciji sa softverskim sistemom i ima svoj naziv

Slučaj korišćenja utvrđuje učesnike (koji se u terminologiji UML nazivaju akterima) koji učestvuju u interakciji sa softverskim sistemom i ima svoj naziv. Slučajevi korišćenja su postali osnovna osobina UML označavanja za opisivanje objektno-orijentisanih modela sistema. Skup slučajeva korišćenja treba da opiše sve moguće interakcije sa sistemom. Akteri i slučajevi korišćenja u kojima učestvuju su povezani linijom sa strelicom na kraju.

Na slici 1 je prikazan dijagram korišćenja softverskog sistema MHC-PMS. Akteri u procesu predstavljeni su figuricama čoveka i svaka interakcije predstavljena je kao elipsa.

Za svaki slučaj korišćenja neophodno je definisati osnovni (primarni) scenario i pomoćne (sekundarne) scenarije koji u tekstualnoj formi opisuju interakciju aktera i sistema. Osnovni scenario opisuje osnovnu funkcionalnost sistema koju slučaj korišćenja opisuje.

Pomoćni (sekundarni)scenari definišu sve poremećaje koji se mogu desiti u realizaciji osnovnog scenarija interakcije. .Sve te sekundarne scenarije treba definisati da bi se kasnije predvidelo kako da sistem razreši takve situacije.



Slika 4.1.1 Slučajevi korišćenja sistema MHC-PMS

ČEMU SLUŽI DIJAGRAM SLUČAJEVA KORIŠĆENJA?

Uvod u dijagrame slučajeva korišćenja, opis i način njihove upotrebe

Dijagram slučajeva korišćenja (UseCase) prikazuje spoljne učesnike, odnosno korisnike sistema i njihove veze sa korisničkim funkcijama koje sistem omogućava. Model slučajeva korišćenja obično se sastoji od više dijagrama slučajeva korišćenja.

Osnovne komponente su: korisničke funkcije, učesnik (korisnik ili akter) i sistem koji se modelira, kao i različite veze: asocijacija, generalizacija i zavisnost između elemenata.

Korisnička funkcija je opis pojedine funkcije koju sistem obezbjeđuje, viđen iz perspektive korisnika, bez opisa unutrašnje funkcionalnosti. Detaljnije se opisuje tekstom ili dijagramom aktivnosti. Učesnik (aktor) je bilo koji spoljni entitet, čovek, drugi sistem ili neki hardverski uređaj, koji treba da komunicira sa sistemom koji se modelira. Sam proces kreiranja modela čini: definisanje sistema, uočavanje učesnika i korisničkih funkcija, opisivanje korisničkih funkcija, definisanje veza između funkcija i na kraju validacija modela.

Neke od najbitnijih stvari koje opisuju dijagram slučlajeva korišćenja su:

1. Prikazuju ponašanja ili funkcionalnost sistema (grubi prikaz šta sistem ili podsistem radi, a ne kako sistem radi)
2. Kreiraju se u ranoj fazi da bi se razumeli zahtevi, a ne kako će se oni realizovati
3. Slučajevi korišćenja su načini prikupljanja funkcionalnih zahteva sistema, opisuju interakcije korisnika i sistema

PRIMER DIJAGRAM SLUČAJEVA KORIŠĆENJA KOD BANKOMATA

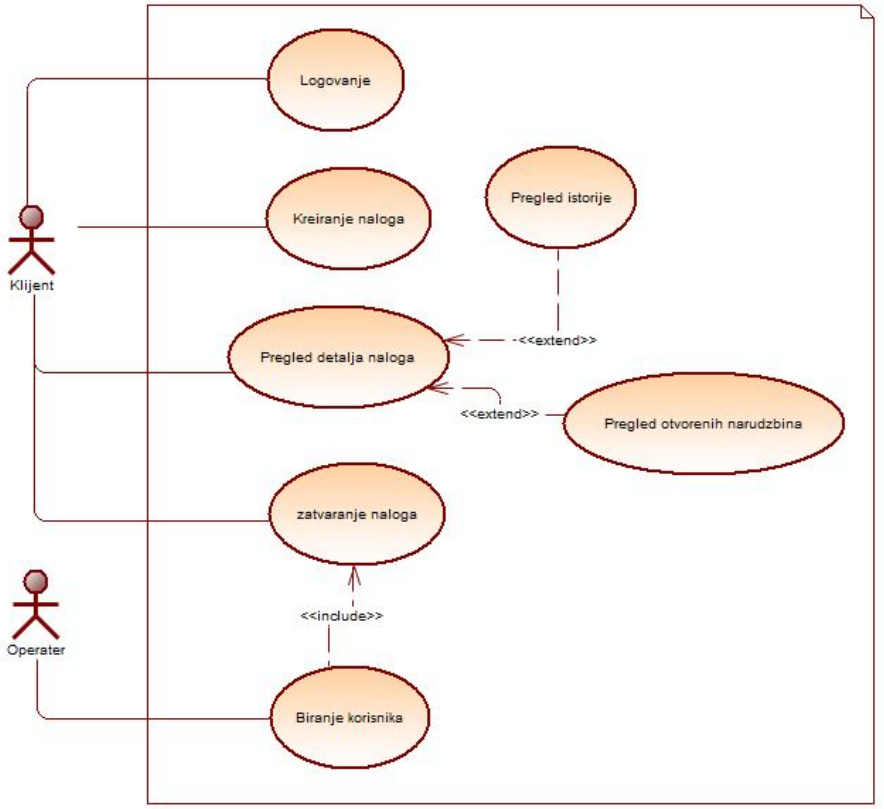
Osnovni koncepti dijagrama slučajeva korišćenja

Nijedan sistem ne postoji izolovano bez interakcije sa ljudskim ili automatizovanim izvođačima, koji taj sistem koriste iz određenih razloga. Izvođači koji koriste određeni sistem očekuju od tog sistema da se ponaša na predvidljive načine.

Slučajevi korišćenja se primenjuju da se prikaže željeno ponašanje sistema koji se razvija, a da se pri tome ne mora definisati kako se to ponašanje realizuje. Korisničke funkcije obezbeđuju način da se projektanti sporazumevaju sa krajnjim korisnicima sistema i ekspertima iz posmatrane oblasti.

U UML-u se dijagrami slučajeva korišćenja koristi se za vizualizaciju sistema, tako da korisnici mogu shvatiti kako da koriste sistem, odnosno da projektanti mogu realizovati posmatrani sistem. Dobro struktuirani slučajevi korišćenja prikazuju samo bitne pojedinosti u ponašanju sistema i nisu ni preterano uopštene ni preterano detaljne.

Na primer, da bi se odredilo ponašanje bankomata navodeći u korisničkim funkcijama kakva uzajamna dejstva sa korisnicima treba da postoje, nije potrebno ništa znati o unutrašnjosti sistema. Korisničke funkcije definišu željeno ponašanje, a ne diktiraju kako to ponašanje treba realizovati. Na taj način omogućena je komunikacija između krajnjih korisnika i eksperata iz raznih domena sa projektantima sistema (koji treba da naprave sistem koji ispunjava zahteve bez preteranog ulaženja u detalje).



SADRŽAJ DIJAGRAMA SLUČAJEVA KORIŠĆENJA (DSK)

Korisničke funkcije, kteri, relacije, komentari, paketi, i granice upotrebe korišćenja

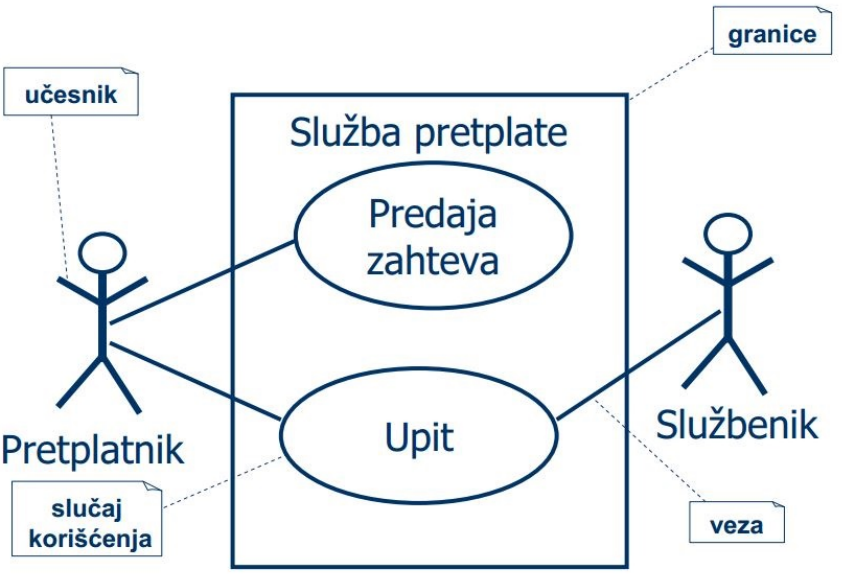
Dijagrami slučajeva korišćenja najčešće sadrže:

- Korisničke funkcije (Use case)
- Uloge, izvođače (Actor)
- Relacije generalizacije , asocijacije i zavisnosti
- Komentare (Note)
- Pakete (Package)
- Granice Sistema

Korisničke funkcije (Use case)

Korisnička funkcija predstavlja funkcionalan zahtev posmatranog sistema. Korisnička funkcija izvršava određenu, sagledivu količinu posla. Iz perspektive datog izvođača, korisnička funkcija radi nešto što je izvođaču korisno. **Korisnička funkcija opisuje šta sistem radi ali ne definiše kako to radi**. Korisnička funkcija se u UML-u predstavlja kao oblačić (elipsa) sa svojim imenom koje je jedinstveno. Korisnička funkcija može da ima varijatete. Jedna korisnička funkcija može biti specijalizovana verzija neke druge korisničke funkcije (ostvaruje se relacijom generalizacije), može biti sadržana u nekim drugim korisničkim funkcijama (ostvaruje se relacijom obuhvatanja), ili može proširivati ponašanje nekih drugih korisničkih

funkcija (ostvaruju se relacijom proširivanja). Korisnička funkcija predstavlja određenu funkcionalnost sistema iz perspektive korisnika tog sistema. Korisnička funkcija mora imati ime, koje obično predstavlja glagol.



Slika 4.1.2 Generalni primer dijagrama slučajeva korišćenja sa svim komponentama

Slika 4.1.3 Primer simbola slučaja korišćenja

AKTER U DIJAGRAMIMA KORIŠĆENJA

Uloge, akter (Actor), njihova upotreba i odnos na dijagramu slučajeva korišćenja

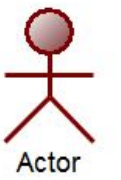
Akter (izvođač) predstavlja ulogu koju korisnici korisničkih funkcija izvode kada su u interakciji sa tim korisničkim funkcijama. Uobičajeno je da izvođač predstavlja ulogu koju čovek, hardverski uređaj ili čak neki drugi sistem igra sa posmatranim sistemom. Uloga se u UML-u predstavlja kao ljudska fugurica. Izvođači mogu biti povezani sa korisničkom funkcijom samo pomoću asocijacije. Dakle, može se zaključiti da je učesnik sadrži jednu ili više uloga jednog ili više entiteta van sistema koji komuniciraju sa sistemom.

Učesnik ne pripada sistemu i njegova se struktura ne opisuje.

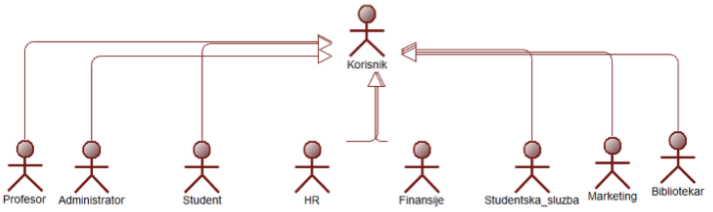
Na dijagramu slučajeva korišćenja (DSK) učesnik (aktor/stakeholder) predstavlja se kao klasa sa sereotipom <<actor>> ili specijalnim simbolom prikazanim na slici 5 .

Izvršavanje slučajeva korišćenja počinje na zahtev jednog učesnika. U zvršavanju slučajeva korišćenja može učestvovati više učesnika i oni sa sistemom takođe razmenjuju poruke i podatke.

Svaki učesnik mora biti povezan sa bar jednim slučajem korišćenja vezom asocijacije.



Slika 4.1.4 Simbol aktora dijagrama SK



Slika 4.1.5 Primer generalizacije aktora sistema

VEZE SLUČAJEVA KORIŠĆENJA

Relacije generalizacije i asocijacije u dijagramu slučajeva korišćenja

Korisničke funkcije i uloge mogu se organizovati definišući relacije generalizacije, zavisnosti (obuhvatanja/uključenja – include i proširivanje - extend) i asocijacije između njih.

Generalizacija između uloga znači da potomak neke uloge nasleđuje ponašanja i smisao roditelja uloge. Naslednik (dete) može da dopuni ili nadjača ponašanje roditelja. Isto važi kao i na vezi generalizacije klasa. Primer generalizacije aktora prikazano je na slici 5 . Bitno je napomenuti da svi slučajevi korišćenja za koje je roditelj vezan asocijacijom postaju dostupne i nasledniku (iako nije direktno vezan asocijacijom sa korisničkim funkcijama).

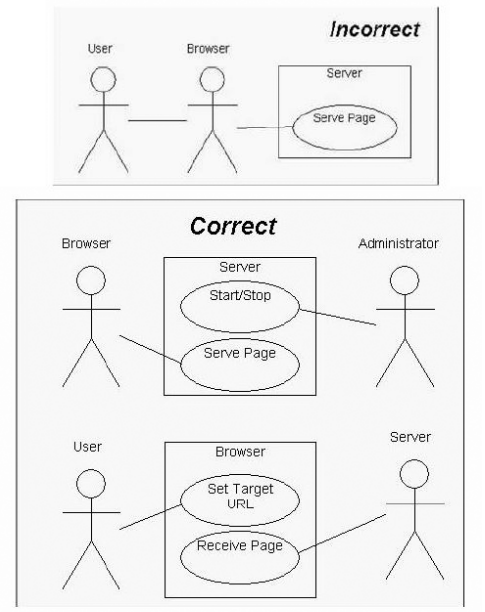
Generalizacija između slučajeva korišćenja označava da naslednih nekog SK nasleđuje ponašanja i smisao roditeljske SK. Primer generalizacije SK dat je na slici 1 . (transakcija / depozit / transfer).

Asocijacija ne postoji između slučajeva korišćenja, jer bi to označavalo mogućnost istovremenog izvršavanja njihovih akcija. Slučajevi korišćenja mogu sadržati druge slučajeve

korišćenja kao deo svojih funkcija. Veze slučajeva korišćenja formiaraju se isključivo pri modelovanju istog sistema.

Relacija asocijacije označava interakciju uloge (izvođača) sa korisničkom funkcijom. Ova relacija predstavlja jedni način povezivanja uloge i korisničke funkcije.

Ukoliko bi postojala interakcija između uloga u posmatranom sistemu, nju ne bismo mogli predstaviti na istom dijagramu. Ono što bi u tom slučaju trebali napraviti jeste kreiranje još jednog Use Case dijagrama, tretirajući jednu ulogu kao poseban sistem, a originalni sistem (sa preostalom ulogom u njemu) kao jednu ulogu u novokreiranom sistemu. Prikaz primera dat je na slici.



Slika 4.1.6 Primer dobre i loše relacije asocijacije

NAČIN CRTANJA VEZA U DIJAGRAMIMA SLUČAJEVA KORIŠĆENJA

Relacije zavisnosti (include, extend) na DSK i njihova primena

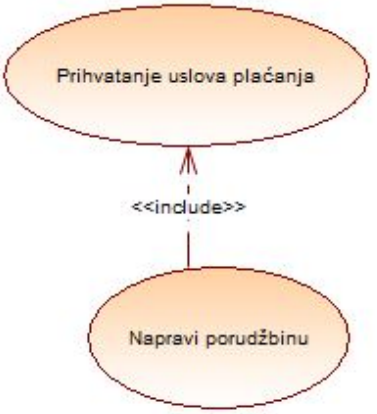
Relacija uključivanja/obuhvatanja (include) može se primeniti samo između korisničkih funkcija. Relacija uklučivanja između korisničkih funkcija znači da osnovna korisnička funkcija eksplicitno obuhvata ponašanje druge korisničke funkcije. Obuhvaćena korisnička funkcija nikad ne može da stoji sama za sebe, već se javlja kao deo neke veće korisničke funkcije koja je obuhvata.

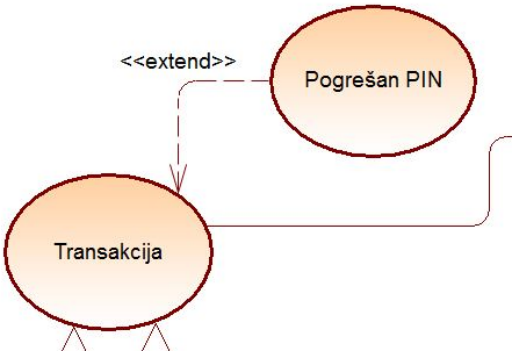
Relaciju uključivanja treba koristiti da bi se izbeglo da se isti tok događaja opisuje više puta, stavljajući zajedničko ponašanje u posebnu korisničku funkciju. Relacija uključivanja označava se kao zavisnost koristeći stereotip include.

Relacija proširivanja može se primeniti samo između korisničkih funkcija.

Relacija proširivanja znači da osnovna korisnička funkcija indirektno ugrađuje ponašanje druge korisničke funkcije na mestu koje je indirektno definisano proširujućom korisničkom funkcijom. Osnovna korisnička funkcija može stajati sama, ali pod određenim okolnostima,

njeno ponašanje može biti prošireno ponašanjem druge korisničke funkcije. Osnovna korisnička funkcija može biti proširena samo na određenim mestima koja se nazivaju tačke proširenja. Proširivanje se može shvatiti kao da proširujuća korisnička funkcija ubacuje ponašanje u osnovnu korisničku funkciju ukuliko su zadovoljeni određeni uslovi. Relacija proširivanja koristi se za modelovanje dela korisničke funkcije koji korisnik može videti kao opciono ponašanje. Na taj način razdvaja se opciono od obaveznog ponašanja. Relacija proširivanja označava se kao zavisnost koristeći stereotip extend.

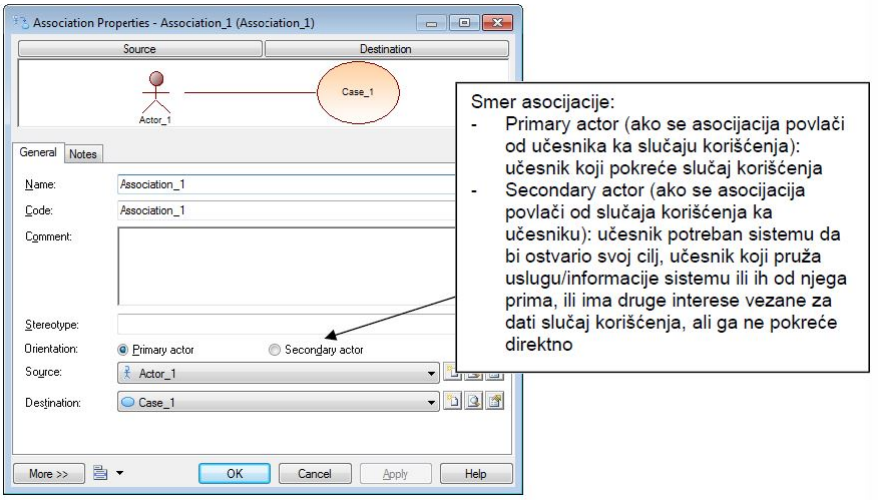




ZAŠTO SU POTREBNE VEZE SLUČAJEVA KORIŠĆENJA?

Izrada veza na DSK u okviru Power Designer-a

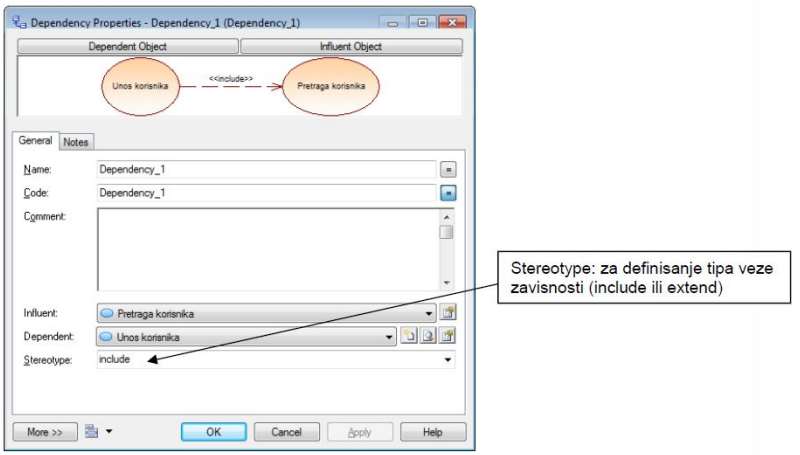
Asocijacija se na dijagramu slučajeva korišćenja koristi za povezivanje učesnika sa slučajem korišćenja u kojem učestvuje. Ona nema strelicu, ali ima smer, koji se može videti i izmeniti u njenom Properties dijalogu kao na slici



Veza generalizacije se na dijagramu slučajeva korišćenja koristi da predstavi nasleđivanje učesnika i nasleđivanje slučajeva korišćenja. Nema posebnih osobina.

Veza zavisnosti je uopštena UML veza kojom se mogu povezivati svi objekti bez ograničenja. Može i ne mora da ima definisan stereotip koji označava oblik zavisnosti.

Na dijagramu slučajeva korišćenja koristi se za međusobno povezivanje slučajeva korišćenja za predstavljanje veze uključivanja (include) i veze proširivanja (extend).

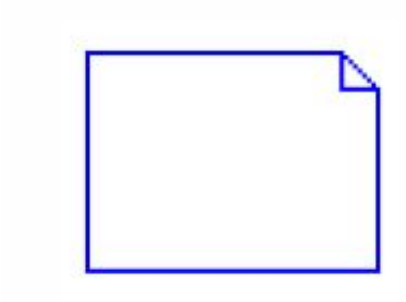


Slika 4.1.7 Dijalog za podešavanje veze zavisnosti između slučajeva korišćenja

KOMENTARI, PAKETI I GRANICE SISTEMA U DIJAGRAMIMA SLUČAJEVA KORIŠĆENJA

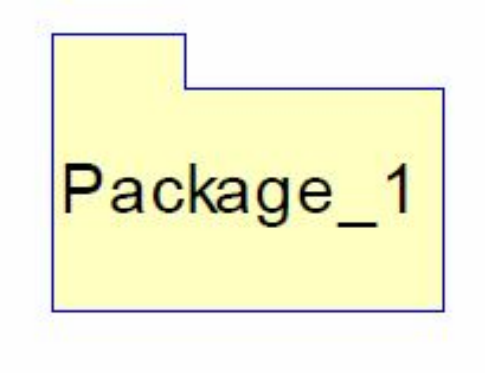
Komentar služi za unos napomene, paket za grupisanje elemenata dijagrama u grupe, a pravougaoni okvir dijagrama označava ospeg funkcionisanja sistema, tj.njegove granice dejstva.

- Komentar predstavlja osnovno anotaciono sredstvo koje se može uvrstiti u UML dijagrame. Komentari se obično koriste za dopunjavanje dijagrama sa ograničenjima ili napomenama koji se najbolje izražavaju neformalnim ili formalnim tekstom.



Slika 4.1.8 Simbol komentara (note)

- Elementi na dijagramima korisničkih funkcija mogu se organizovati grupisanjem u pakete, na isti način na koji se mogu organizovati klase. Paket je mehanizam opšte namene za organizovanje elemenata u grupe.



Slika 4.1.9 Simbol paketa

- Pravougaonik oko korisničkih funkcija označava opseg sistema - korisničke funkcije unutar pravougaonika predstavljaju funkcionalnost koju sistem implementira.

Slika 4.1.10 granice sistema

MODELOVANJE DSK

Na šta treba obratiti pažnju prilikom modelovanja ponašanja posmatranog sistema

a) Identifikujte uloge (izvođače) koji su u interakciji sa posmatranim sistemom. Kandidati za uloge su grupe koje zahtevaju određeno ponašanje radi izvođenja svojih zadataka ili koji su neophodni direktno ili indirektno radi izvršavanja funkcija tog sistema

b) Za svakog izvođača razmotriti ponašanje koje on očekuje ili zahteva da sistem obezbedi i ta ponašanja potrebno je prikazati kao korisničke funkcije

c) Organizujte uloge identifikujući opšte i specijalne uloge (primena generalizacije kod uloga)

d) Za svaku ulogu ponaosob odrediti načine na koje on stupa u interakciju s sistemom koje menjaju stanje sistema ili njegove okoline ili koje obuhvataju slanje odgovora na neki događaj.

e) Organizujte korisničke funkcije primenjujući relacije proširivanja i obuhvatanja radi razlaganja na zajedničke delove ponašanja i razlikovanje izuzetnih ponašanja.

f) Sve korisničke funkcije, uloge, relacije generalizacija, asocijacija i zavisnosti treba da imaju imena koja govore o njihovj nameni

g) Rasporediti elemente tako da se broj linija koje se seku svedu na najmanju moguću meru

h) Ukoliko nije moguće prostorno organizovati elemente dijagrama tako da ponašanje i uloge koje su semantički bliske budu i fizički blisko raspoređeni, potrebno je uvesti pakete u dijagram

SPECIFIKACIJA SLUČAJA KORIŠĆENJA

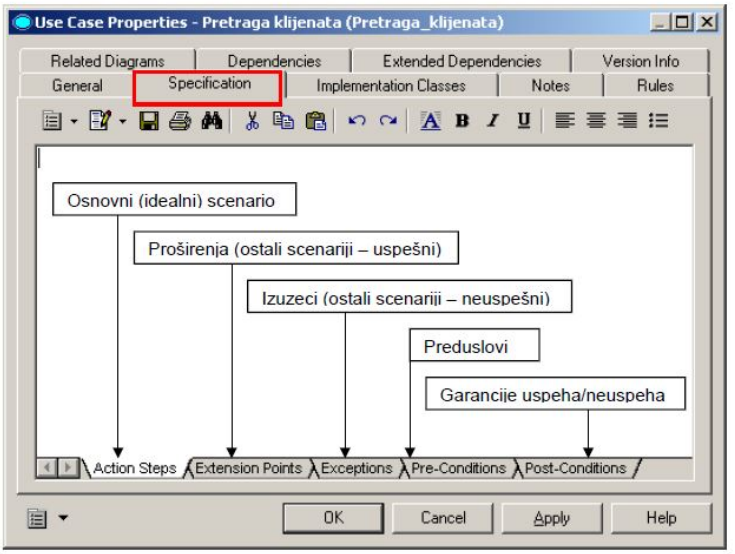
Specificiranje slučaja korišćenja korišćenjem Power Designer razvojnog alata

Korisnička funkcija opisuje šta sistem radi, ali ne definiše kako to radi. Moguće je specifikovati ponašanje korisničke funkcije tekstualno opisujući tok događaja, dovoljno jasno da to lako razume i neko ko je neupućen. Kada se piše tok događaja, potrebno je naznačiti kada i kako korisnička funkcija započinje i završava, kada korisnička funkcija stupa u interakciju sa izvođačima, koji se objekti razmenjuju, potrebno je opisati osnovni i alternativne tokove ponašanje.

Specifikaciju korisničke funkcije moguće je odraditi u kartici „Specification“ na korisničkoj funkciji.

1. **Action steps** - opisuje se osnovni (uspešni) tok događaja. Pod osnovnim tokom događaja navodi se kada korisnička funkcija počinje svoje izvršavanje i ko inicira njeno izvršavanje. Ukoliko korisnička funkcija uključuje neke druge funkcije (postoji include zavisnost), navodi se kada se uključena korisnička funkcija izvršava.

2. **Extension Points** - alternativni tok događaja, ukoliko korisnička funkcija ima proširenja ona se ovde navode, uz potrebne uslove izvršavanja.
3. **Exceptions** - neuspešni tok događaja
4. **Pre - Conditions** - navođenje potrebnih preduslova da bi se korisnička funkcija uopšte izvršila
5. **Post - Conditions** - rezultati uspešnog i/ili neuspešnog izvršavanja korisničke funkcije



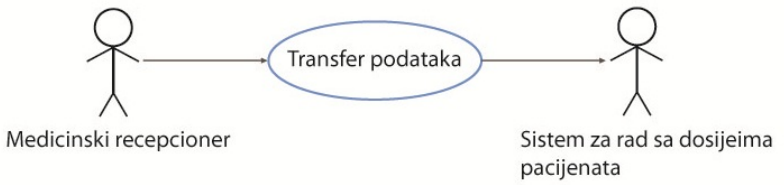
Slika 4.1.11 Kartica za opis izvršavanja slučajeva korišćenja

PRIMER SISTEMA MHC-PMS: SLUČAJ KORIŠĆENJA "TRANSFER PODATAKA"

Slučaj korišćenja se detaljnije opisuje najčešće tabelom, koja sadrži i scenario (opis) interakcije aktera i softverskog sisetma.

Modelovanje slučajevima korišćenja se najčešće koristi kod definisanja zahteva i radi utvrđivanja zahteva. Slučaj korišćenja je jedan jednostavni scenario koji opisuje šta korisnik može da očekuje od nekog sistema. Svaki slučaj korišćenja predstavlja jedan poseban zadatak koji sadrži spoljnu interakciju sa sistemom. U najprostijoj verziji, slučaj korišćenja se predstavlja jednom elipsom sa akterom u vidu ljudske figurice. Slika 15 pokazuje jedan slučaj korišćenja sistema MHC-PMS koji predstavlja zadatak ubacivanja podataka iz MHC-PMS u sistem za upravljanje zdravstvenim informacijama pacijenata.

Vidi se da ljudska ikonica, akter, može ,pored čoveka, da predstavlja i neki drugi sistem. Ovde se strelica neformalno koristi da bi se ukazalo ko inicira transfer i gde se završava transfer. UML, inače, ne koristi strelice u dijagramima slučaja korišćenja.



Slika 4.1.12 Slučaj korišćenja - Transfer podataka

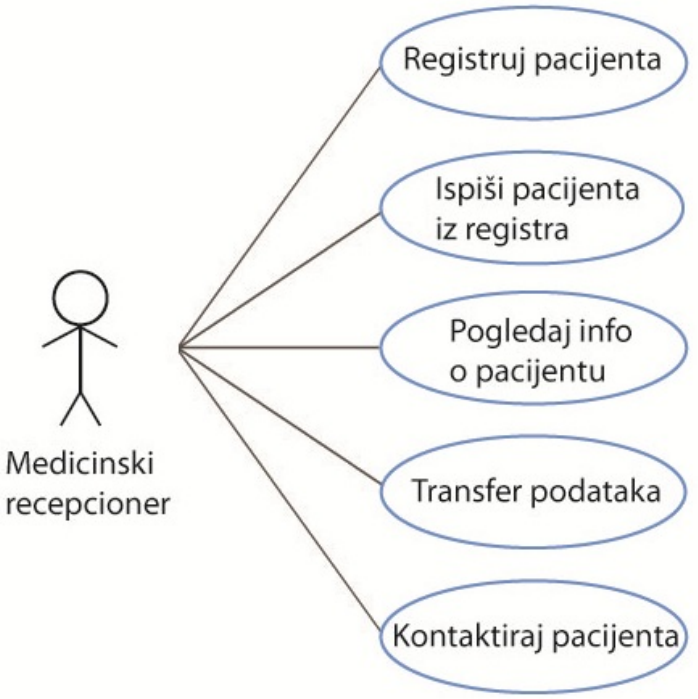
Kao dopuna dijagramima korišćenja, upotrebljava se i tekstualna informacija, koja opisuje scenario interakcije. Pored toga, mogu se koristiti i tabele, ili sekvencijalni dijagrami, kao dopuna koja daje više detalja o interakciji sistema sa akterima i okruženjem. U Tabeli 1 prikazana je tabela sa opisom slučaja korišćenja Transfer podataka.

MHC-PMS: Transfer podataka	
Akteri	Medicinski recepcioner, Sistem za upravljanje informacijama o pacijentima (PRS)
Opis	Recepcioner može da izvrši transfer podataka iz MHC-PMS sistema u bazu podataka sistema za upravljanje zdravstvenim informacijama pacijenata, koje održava državni organ iz oblasti zdravstva. Informacije koje se prebacuju su ili promenjene lične informacije (adresa, broj telefona i dr.) ili informacije o dijagnozama i terapijama pacijenata.
Podaci	Lične informacije o pacijentu, informacije o lečenju
Stimulans	Komanda korisnika urađena od strane medicinskog recepcionera
Odgovor	Potvrda da je PRS osvežio svoju bazu podataka uspešnim transferom podataka.
Komentari	Recepcioner mora da ima odgovarajuće sigurnsone dozvole pristupa informacijama o pacijentima i PRS sistemu.

PRIMENA SCENARIJA

Scenario opisuje interakciju slučaja korišćenja sistema, tj. sekvencijalno navedene akcije aktera i reakcije softverskog sistema. Postoje osnovni (primarni) i pomoćni (sekundarni) scenariji.

Na slici 17 prikazuje isti slučaj, ali pokazuje i ostale slučajeve korišćenja, tj. sve interakcije aktera „Medicinski recepcioner“



Za svaku interakciju se može dati tekstualni opis, kao što je dat na slici 2 za slučaj transfera podataka. Pored takvog opisa, moguće je, a i poželjni, da se da i tekstualni opis scenarija interakcije. Na primer, za interakciju "Registruj pacijenta" može se u više tačaka navesti šta treba da uradi akte (actor) Medicinski recepcioner, a šta treba sistem da na to reaguje. To je tzv. osnovni scenario.

Svaki osnovni ili primarni scenario prati i najčešće, više tzv. pomoćnih ili sekundarnih scenarija. Oni opisuju situacije u interakciji u kojoj se javljaju neželjeni poremećaji ili događaji koji mogu da ometu željenu interakciju aktera i sistema, tj.dolazi do odstupanja osnovnog scenarija. Na primer, šta će se desiti ako dođe do nestanka struje prilikom registracije pacijenta. Ili, kako sistem da reaguje, ako je pacijent već ranije registrovan, a medicinskom recepcioneru to nije bilo poznato. Ako se takve situacije ne predvide, kasnije razvijen softver u takvim situacijama može ući u bag, tj. u blokadu rada, jer nije obučen šta da radi u tim situacijama.

Čovek je element neizvesnosti i izvor mogućih događaja koji remete normalnu interakciju. Profesionalni softver mora da bude projektovan da pravilno radi i u takvim situacijama.

POKAZNI PRIMER 1

Slučaj koriišćenja onlajn CD kataloga

Zadatak 1

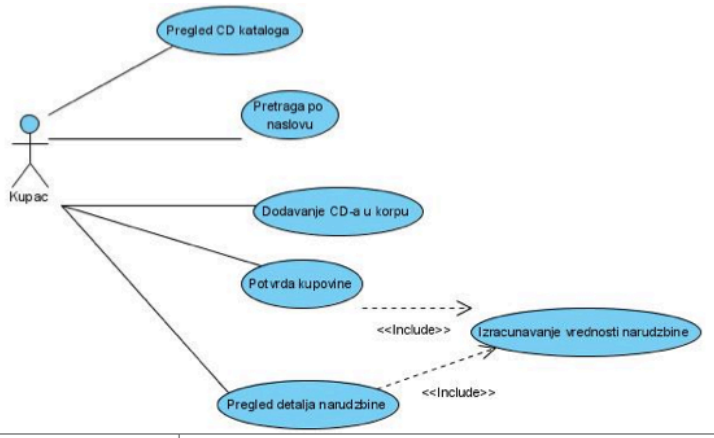
Kreirati dijagram slučajeva korišćenja za on-line CD katalog. Sistem ima jedinstvenog učesnika: on-line kupca. Kupac može da pregleda katalog, pretražuje po naslovu, dodaje CD

u narudžbinu, pregleda detaljnije narudžbine i potvrdi kupovinu. Dati specifikaciju najbitnijeg slučaja korišćenja upotrebe sa odgovarajućim opisima.

Objašnjenje:

Na slici 1 prikazan je dijagram osnovnih slučajeva korišćenja: pregled CD kataloga, pretraga po naslovu, dodavanje CD-a u korpu, potvrda kupovine i pregled detalja narudžbine. Ono što je takođe prikazano i jako bitno jeste da dva slučaja: potvrda kupovine i pregled detalja kupovine, uključuju i izračunavanje vrednosti narudžbine.

Na osnovu slike, dati su odgovarajući opisi, preduslovi, posledice, scenario i alternativni tokovi.



Ime	Potvrda kupovine
Učesnici	Kupac
Okidač	Pokreće se izborom opcije za potvrdu kupovine sa korisničkog interfejsa
Preduslovi	Kupac ima dovoljno novca na svom računu za plaćanje ukupnog računa, izabrani proizvod se nalazi na stanju u magacinu…
Posledice	Vrši se plaćanje troškova porudžbine sa računa korisnika I isporuka proizvoda korisniku
Uspešni scenario	Korisnik vrši izbor opcije za potvrđivanje kupovine. System vrši proveru validnosti korisnikove kartice. Sistem izračunava vrednost narudžbine I proverava da li korisnik ima dovoljno novca na računu. Sistem upućuje zahtev bankarskom sistemu da izvrši novčanu transakciju I prebaci novčana sredstva sa korisnikovog računa. Nakon potvrde uspešne transakcije narudžbenica se prosleđuje odeljenju za isporuku robe.
Alternativni tokovi	A)Bankarski system prijavljuje da je korisnikova kartica istekla. - korisniku se šalje poruka da nije moguće izvršiti kupovinu. B)Bankarski system prijavljuje da korisnik nema dovoljno novca na računu - system za online kupovinu šalje poruku korisniku o tome …

Slika 4.1.13 Dijagram osnovnih slučajeva korišćenja

POKAZNI PRIMER 2

Slučaj korišćenja softverskog sistema za lansiranje raketa

Zadatak 2

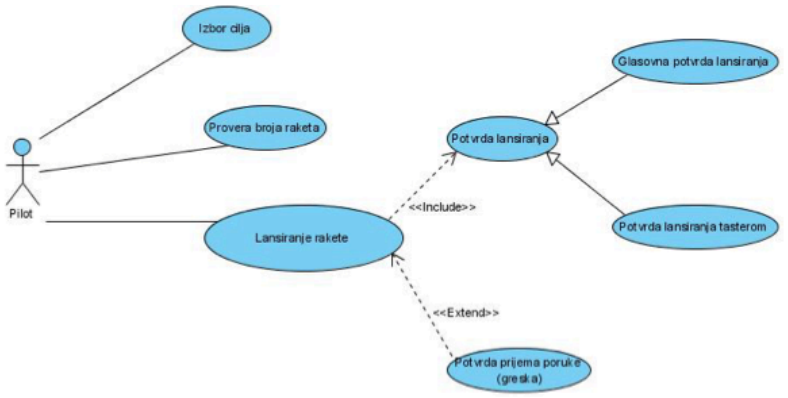
Realizuje se softverski sistem za kontrolu lansiranja raketa. Pilot može izvršiti izbor cilja, lansiranje raketa, proveru broja raketa. Svako lasniranje rakete potrebno je da potvrdi pri čemu se potvrda može obaviti odgovarajućom glasnom komandom ili pritiskom odgovarajućeg dugmeta. Ukoliko se u toku lansiranja desi kvar na sistemu za lansiranje pilot

će biti obavešten o ovom događaju i potrebno je da potvrdi prijem poruke. Dati specifikaciju najbitnijeg slučaja korišćenja sa opisom.

Objašnjenje:

Na slici 2 prikazan je dijagram na osnovu opisa zadatka. Pored izbora cilja i provere broja raketa, postoji i složenija akcija koja se odnosi na lansiranje raketa. Ova akcija pre svega mora biti potvrđena, ali to može biti na dva načina: glasovnom porukom ili tasterom, što je i prikazano vezom generalizacije na slici 2.

Pored toga, alternativa, odnosno proširenje sistema je da se na sistemu desi kvar i da se porukom o grešci sistem obavesti o tome.



Ime	Lansiranje rakete
Učesnici	Pilot
Okidač	Pritisak na dugme za lansiranje
Preduslovi	Izvrsena potvrda lansiranja rakete Na lanseru rakete je postavljena raketa, Avion se nalazi na visini od preko 1000 m, Izabrani cilj se nalazi u dometu rakete,
Posledice	Ispaljuje se raketa sa lansera u pravcu izabranog cilja.
Uspešni scenario	Pilot vrsi izbor tipa rakete koju zeli da ispali na izabrani cilj. Pilot pritiska dugme za ispaljivanje rakete. Sistem trazi potvrdu lansiranja. Pilot potvrdjuje (glasom ili tasterom) lansiranje rakete. Raketa se ispaljuje
Alternativni tokovi	A) Sistem prijavljuje da izabrana raketa ne postoji B) Sistem prijavljuje kvar na lanseru rakete C) Izabrani cilj se nalazi van dometa. a. Pilot dobija poruku da je cilj izasao iz dometa, b. Pilot potvrdjuje prijem poruke I obustavlja lansiranje.

Slika 4.1.14 0 Primer mogućeg rešenja postavljenog zadatka.

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji se odnose na modelovanje dijagrama slučajeva korišćenja.

1. Zadatak: Opis sistema: Sistem za prodaju PC komponenti. Sistem je namenjen firmama koje žele da naveliko kupuju PC komponente za svoje radnje. Napravljen je tako da olakša proces naručivanja i isporuke svih porudžbina. Podaci i informacije o svim poručivanjima čuvaće se u bazi podataka i time smanjiti broj potrebne dokumentacije za sve narudžbine.
Glavne komponente od kojih će sistem da se sastoji su komponenta za naručivanje i isporuku

PC delova, komponenta za upravljanje proizvodima, komponenta za informisanje klijenata i komponenta za upravljanje nalogom.

Modelovati dijagram slučajeva korišćenja za korisnika. Izvršiti specifikaciju svakog identifikovanog slučaja korišćenja.
2. Zadatak: Opis sistema: Chatspace sistem simulira osnovne funkcionalnosti jedne chat aplikacije. Sistem omogućava korisniku da pristupi sistemu sa svojim nalogom na osnovu koga dobija sve privilegije korisnika sistema. Korisnik dobija mogućnost da razmenjuje poruke sa drugim korisnicima sistema. Poruke mogu biti u okviru teksta, priče, pesme, i ostalih tekstualnih objava. Korisnik ima mogućnost da poruke šalje drugim korisnicima. Takođe, korisnik ima mogućnost da čita poruke drugih korisnika kao i uvid u listu ostalih korisnika koji koriste sistem.

Modelovati dijagram slučajeva korišćenja za korisnika. Izvršiti specifikaciju svakog identifikovanog slučaja korišćenja.

4.2 Dijagrami interakcija

DIJAGRAMI INTERAKCIJE

Dijagram interakcije prikazuje interakciju koja se sastoji od skupa objekata i njihovih relacija, uključujući poruke koje mogu da se razmenjuju između njih

Dijagrami sekvenci i **dijagrami saradnje** - koji se zajedno nazivaju **dijagramima interakcija** - dva su od pet dijagrama u UML-u koji se koriste za modelovanje dinamičkih aspekata sistema. Dijagram interakcije prikazuje interakciju koja se sastoji od skupa objekata i njihovih relacija, uključujući poruke koje mogu da se razmenjuju između njih. Dijagram sekvence je dijagram interakcije u kome se ističe vremenski redosled poruka; dijagram saradnje je dijagram interakcije u kome se ističe strukturna organizacija objekata koji šalju i primaju poruke.
Dijagrami interakcija koriste se za modelovanje dinamičkih aspekata sistema. To najčešće uključuje modelovanje konkretnih ili prototipskih konkretnih primeraka klasa, interfejsa, komponenti i čvorova, zajedno sa porukama koje se razmenjuju između njih, sve u kontekstu scenarija koji ilustruje neko ponašanje. Dijagrami interakcije mogu se koristiti nezavisno, radi vizualizovanja, specifikovanja, konstruisanja i dokumentovanja dinamike određenog društva objekata, ili se mogu koristiti za modelovanje jednog određenog odvijanja upravljanja neke korisničke funkcije.
Dijagrami interakcije nisu važni samo za modelovanje dinamičkih aspekata sistema, nego i za konstruisanje izvršivih sistema pomoću direktnog i reverznog inženjeringa.

Dijagram interakcije prikazuje interakciju, koja se sastoji od skupa objekata i njihovih relacija, uključujući poruke koje se mogu razmenjivati između njih. Dijagram sekvence je dijagram interakcije koji ističe vremenski redosled poruka. Grafički, dijagram sekvence je tabela koja prikazuje objekte uređene po X osi i poruke, poređane po vremenskom redosledu, po Y osi.

Dijagram interakcije je jedna posebna vrsta dijagrama i ima iste zajedničke osobine kao i ostaii dijagrami - ime i grafičke sadržaje koji su neka projekcija u model. Dijagram interakcije razlikuje se od ostalih dijagrama po vrsti svog sadržaja. Dijagram interakcije obično sadrži:

- Objekte
- Veze
- Poruke

KOLABORACIONI I SEKVENCIJALNI DIJAGRAM

Dijagrami interakcije i njihova upotreba

Dijagrami interakcija pokazuju interakciju (međusobnu aktivnost), koja sadrži skup objekata i njihove relacije, uključujući i poruke koje oni mogu da razmenjuju. Upotrebljavaju se radi modeliranja dinamičkih aspekata sistema, a naročito kod projektovanja izvršnih sistema.

Postoje dve vrste dijagrama interakcija:

1. **Kolaboracioni dijagrami:** Naglasak je na strukturnoj organizaciji objekata. Koriste se za modeliranje upravljačkog toka u sistemu koji se modelira.

2. **Sekvencijalni dijagrami:** Naglasak je na vremenu slanja poruka. Služe za modeliranje upravljačkog toka - tokom vremena.

Sekvencijalni dijagram je deo Objektno-orijentisanog modela. Predstavlja dinamički pogled na objekte (instance klasa) i poruke koje objekti međusobno razmenjuju. Sekvencijalni dijagram može sadržati učesnike ili aktore koji su u interakciji sa sistemom. Interakcija se predstavlja u dve dimanzije. Vertikalna dimenzija predstavlja vreme, a hroizontalna predstavlja objekte.

SEKVENCIJALNI DIJAGRAM

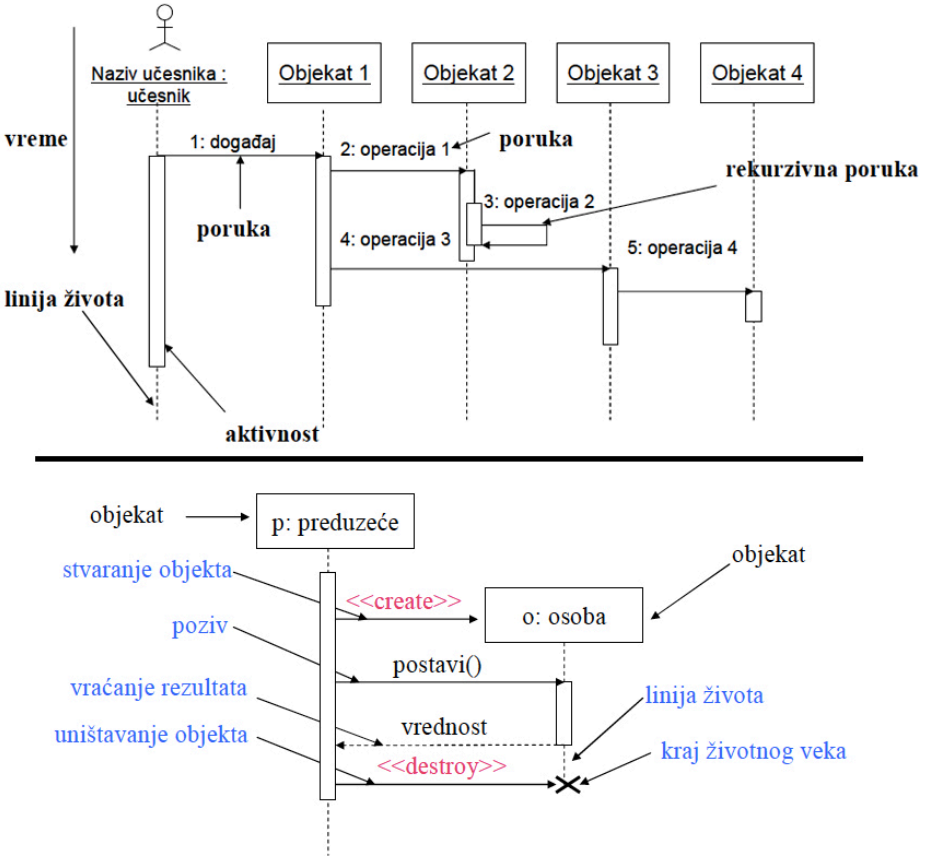
Sekvencijalni dijagram predstavlja dinamički pogled na objekte (instance klasa) i poruke koje objekti međusobno razmenjuju

Ranije u ovim vežbama dat je primer sistem sekvencijalnog dijagrama i njegov opis i upotreba. Svi elementi koji su korišćeni prilikom njegove izrade primenjuju se i prilikom izrade sekvencijalnog dijagrama. Glavna razlika je u tome što objekat koji je predstavljao najčešće sistem („crna kutija“) u okviru sistemskog sekvencijalnog dijagrama gde nismo znali koji interno objekti komuniciraju sada oni postaju predmet ovog dijagrama. Na sekvencijalnom dijagramu (SD) ne postoji sistem kao objekat već prave instance klasa odnosno objekte.

Da bi se izdizajnirao sekvencijalni dijagram neophodno je prvo izraditi klasni dijagram sa svim pratećim atributima i metodama (operacijama) tih klasa kako bi se moglo prepoznati koji su to objekti značajni i u upotrebi prilikom kreiranja sekvencijalnog dijagrama.

Kao što je i ranije napomenuto jedan Sekvencijalni dijagram odnosi se na jedan slučaj korišćenja, ali i jedan slučaj korišćenja može imati jedan ili više dijagrama i to za glavni tok, ali i za alternativne tokove u okviru dijagrama slučajeva kroišćenja.

Na slici 1 prikazana je grafička notacija sekvencijalnog dijagrama.



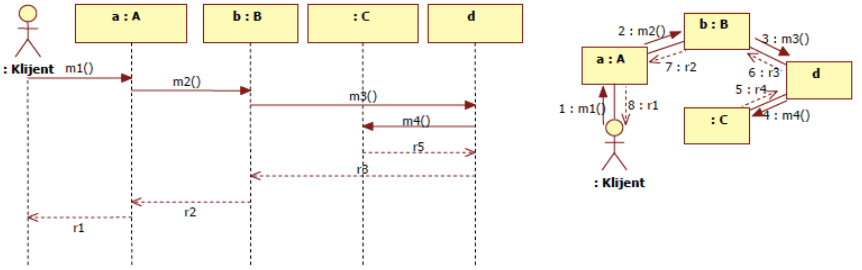
Slika 4.2.1 Grafička notacija sekvencijalnog dijagrama

KOLABORACIONI DIJAGRAMI

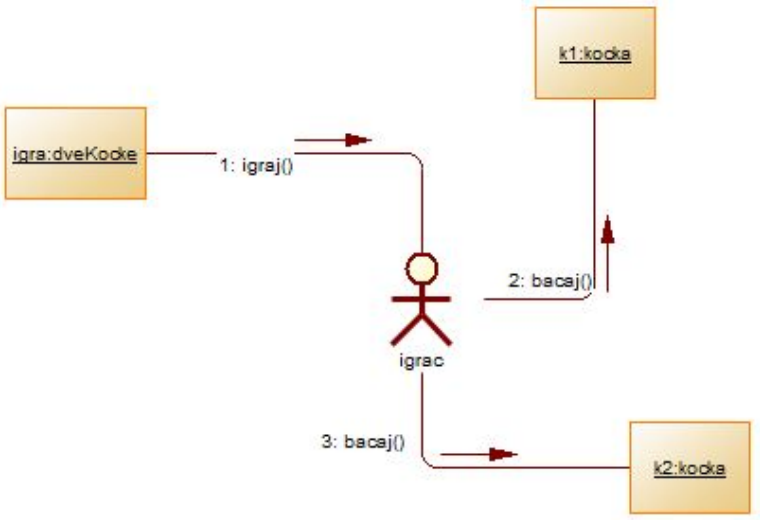
Kolaboracioni (komunikacioni) dijagrami predstavljaju jednu vrstu interakcionih dijagrama sa naglaskom na povezanosti podataka između raznih učesnika u interakciji

Kao što je već opisano ranije dijagram sekvenci fokusira se na prikaz poruka u vremenu. Dijagram komunikacije (saradnje) je u nekim aspektima sličan dijagramu sekvenci ali se on fokusira na saradnju objekata. On predstavlja interakciju između objekata kao niz sekvenci poruka. Kolaboracioni dijagrami opisuju statičku strukturu i dinamičko ponašanje sistema. Ponašanje objekata opisano je ulogama klase. Kada se koristi UML simbol za objekat, radi ilustracije uloga klase, ne prikazuju se atributi objekata. Za razliku od sekvencijalnog, ovi dijagrami nemaju eksplicitnog načina označavanja vremena nego numerišu poruke po redu izvršavanja.

Na slici je prikazan osnovni dijagram kolaboracije. Takođe, dat je primer iste aktivnosti na dijagramu kolaboracije i dijagramu sekvenci kako bi se uočile sličnosti i razlike.



Slika 4.2.2 Poređenje dijagrama sekvenci i kolaboracije

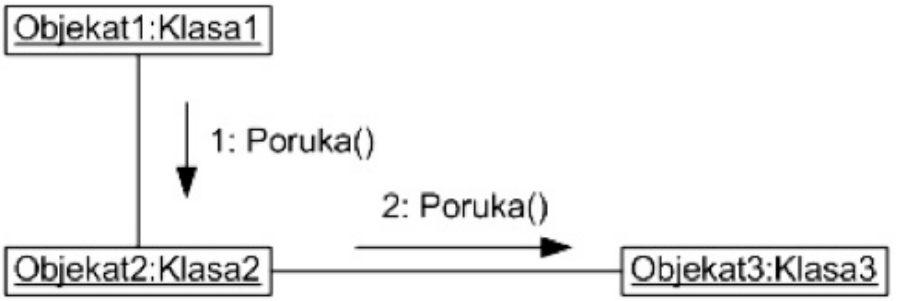


Slika 4.2.3 Primer dijagrama kolaboracije

MODELIRANJE SARADNJE (KOMUNIKACIJE) OBJEKATA

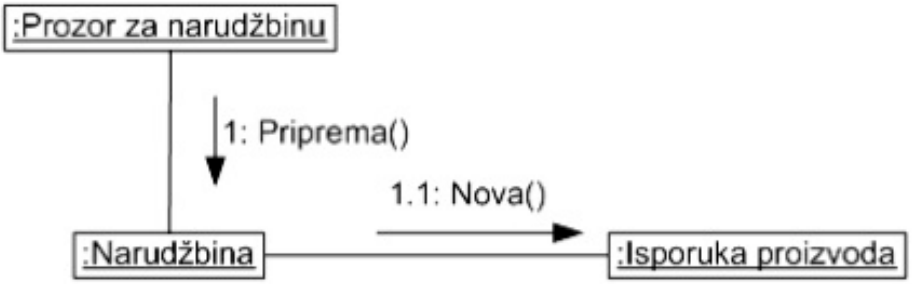
Dijagram saradnje pokazuje vezu između objekata i redosled poruka koje se prosleđuju između njih

Dijagram saradnje pokazuje vezu između objekata i redosled poruka koje se prosleđuju između njih. Ovi dijagrami pomažu da se lakše prikažu složenije interakcije i da se pokažu veze između objekata koji saradjuju.
Strelice između objekata prikazuju poruke koje se prosleđuju između njih. Brojevi pored poruka predstavljaju sekvencu brojeva. Kako naziv predlaže, brojevi prikazuju sekvencu poruka koja se prosleđuje između objekata. Za jednostavnije primere može da se koristi sekvenca brojeva 1.,2.,3. itd., dok za složenije se prelazi na 1,1.1,1.2,1.2.1,...itd. Na slici 1 prikazani su osnovni elementi: objekti i poruke koje se prosleđuju između njih.



Slika 4.2.4 Dijagram saradnje

Na slici 2 prikazan je jednostavan dijagram saradnje na primeru narudžbine. U ovom primeru naziv objekata je smešten iza dve tačke

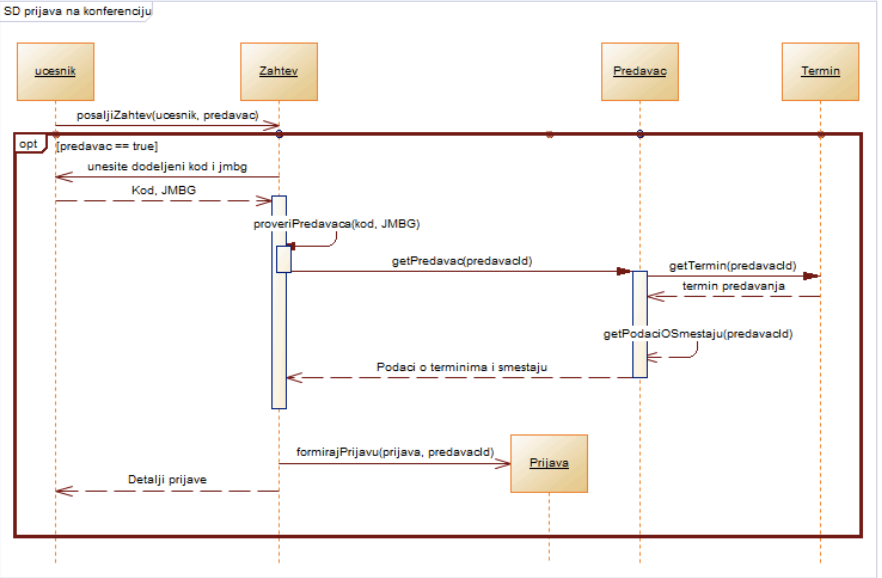


Slika 4.2.5 Primer primene dijagrama saradnje

POKAZNI PRIMER

Izrada dijagrama interakcije korišćenjem alata Power Designer za dati primer

Korišćenjem sekvencijalnog dijagrama modelovati scenario prijave učesnika na konferenciju. Pretpostavka je da je učesnik već registrovan i prijavljen na sistem. Ukoliko se učesnik pri slanju zahteva prijavi kao predavač, potrebno je proveriti da li se on nalazi na definisanoj listi predavača konferencije. U tom procesu sistem od učesnika očekuje unos matičnog broja i dodeljenog koda kako bi se potvrdio njegov identitet. Uspešnom potvrdom predavača prosleđuje se raspored termina dogovorenih predavanja i podaci o smeštaju za vreme konferencije. Ukoliko se učesnik prijavljuje kao slušalac na konferenciji, potrebno je da se putem sistema izvrši naplata kotizacije. U tom procesu učesnik može odabrati i opciju “uključen smeštaj” pri čemu sistem proverava da li ima slobodnog mesta. Ukoliko ima smeštaja cena se uvećava na kotizaciju i učesniku se prosleđuju informacije. Sve prijave učesnika je neophodno evidentirati u sistem. Koristiti objekte sledećih klasa: Učesnik, Predavač, Zahtev, Termin, Prijava.



Slika 4.2.6 Primer rešenja zadatka

Napomena: Objekat :Zahtev koristi se za inicijalnu komunikaciju i zadužen je za osnovne provere definisane u zadatku, dok se objekat :Prijava kreira nakon što se kompletan postupak uspešno izvrši.

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji se odnose na dijagrame interakcije.

Zadatak 1: Opis aplikacije: Ova aplikacija namenjena je za zaposlene u privatnoj klinici (medicinska sestra, doktor, direktor klinike). Svaki korisnik se prijavljuje i na osnovu prijave ima različite opcije korišćenja sistema. Opcije su sledeće:

1. Prijavljivanje na sistem klinike
2. Otvaranje kartona pacijenta
3. Pristup kartonu pacijenta
4. Slanje pacijentovih podataka iz kartona lekaru
5. Prepisivanje terapije
6. Izdavanje recepta za leka
7. Davanje uputa za dalje lečenje
8. Pretraživanje zaposlenih
9. Potpisivanje naloga, zapisnika i pravilnika klinike
10. Prikaz pravilnika klinike

Modelovati sekvencijalni dijagram i kolaboracioni dijagram za opisanu aplikaciju.

Zadatak 2: Opis aplikacije: Aplikacija treba da omogući međusobnu interakciju korisnika putem društvene mreže na kojoj korisnici mogu da dele video sadržaje, četuju međusobno i prate sadržaje koji drugi korisnici dele na društvenoj mreži. Korisnik da bi postao član

društvene mreže prvo mora da se registruje, proces registracije jeste jako jednostavan i ne zahteva previše podataka od korisnika već samo one osnovne. Kada se registruje korisnik će moći da deli svoj video sadržaj sa drugim korisnicima, prati sadržaje koji su drugi korisnici podelili, lajkuje i komentariše te iste sadržaje kao i mogućnost četovanja sa drugim registrovanim korisnicima. Takođe, registrovani korisnik će imati opciju da prati druge korisnike koji izbacuju video sadržaj na društvenoj mreži.

Modelovati sekvencijalni dijagram i kolaboracioni dijagram za opisanu aplikaciju.

4.3 Sekvencijalni dijagrami

ULOGA SEKVENCIJALNIH DIJAGRAMA

Sekvencijalni dijagram prikazuje redosled odvijanja interakcija (slanje poruka između objekata) za određeni slučaj korišćenja ili primer (instancu) slučaja korišćenja

Svaki slučaj korišćenja sistema (dat u modelu slučajeva korišćenja) se opisuje osnovnim i pomoćnim scenarijima. Oni u tekstualnom obliku opisuju redosled (sekvencu) interakcija aktera i sistema. Međutim, pored opisa slučaja korišćenja u vidu scenarija u tekstualnoj formi, isti scenario se može izraziti i grafičkim putem u vidu tzv. *sekvencijalnog dijagrama*.

UML sekvencijalni dijagrami se koriste za modelovanje interakcije između aktera i objekata u sistemu, ili samih objekata.

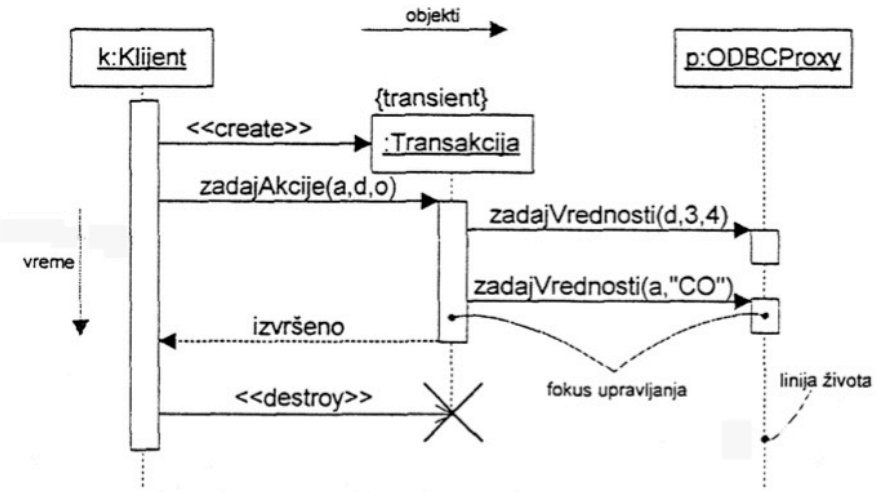
Sekvencijalni dijagram prikazuje redosled odvijanja interakcija (slanje poruka između objekata) za određeni slučaj korišćenja ili primer (instancu) slučaja korišćenja. Slika 1 pokazuje dijagram sa osnovnim oznakama. On pokazuje pregled pacijentovih informacija od strane medicinskog recepcionera. Akteri i objekti su na vrhu dijagrama sa isprekidanim vertikalnim linijama, a strelice pokazuju interakcije između objekata. Pravougaonici na isprekidanim linijama označavaju životni ciklus objekta (na vrhu linije), tj. vreme trajanja objekta u proračunima (proces obrade poruka). Dijagram se čita odozgo do dole, sleva na desno. Oznake iznad strelica pokazuju poruke objektima, njihove parametre, i povratne vrednosti. Mogu se na dijagrami navoditi i sulovi za razmenu pruka između objekata u interakciji

Dijagrami interakcija koriste se za modelovanje dinamičkih aspekata sistema. To najčešće uključuje modelovanje konkretnih ili prototipskih konkretnih primeraka klasa, interfejsa, komponenti i čvorova, zajedno sa porukama koje se razmenjuju između njih, sve u kontekstu scenarija koji ilustruje neko ponašanje. Dijagrami interakcije mogu se koristiti nezavisno, radi vizualizovanja, specifikovanja, konstruisanja i dokumentovanja dinamike određenog društva objekata, ili se mogu koristiti za modelovanje jednog određenog odvijanja upravljanja neke korisničke funkcije.
Dijagrami interakcije nisu važni samo za modelovanje dinamičkih aspekata sistema, nego i za konstruisanje izvršivih sistema pomoću direktnog i reverznog inženjeringa.

DIJAGRAM SEKVENCI

Sekvencijalni dijagram predstavlja objekte koji učestvuju u jednoj interakciji i koji su horizontalno poređani. Po vertikali se navode inetrakcije poređane po vremenu aktiviranja.

Dijagram sekvence ističe vremenski redosled poruka. Kao što se vidi na slici 1, dijagram sekvence formira se postavljanjem objekata koji učestvuju u interakciji na vrh dijagrama, po X osi. Obično, objekat koji započinje interakciju stavljate na levu stranu, a one koji se kasnije uključuju, redom ka desnoj strani. Zatim unosite po Y osi poruke koje ti objekti šalju i primaju, od vrha ka dnu, kako vreme teče. To čitaocu daje jasnu vizuelnu predstavu o tome kako se odvija upravljanje tokom vremena.



Slika 4.3.1 Primer djagrama sekvenci

U dijagramu sekvence postoji linija života objekta. Linija života objekta je vertikalna crtkasta linija koja predstavlja da objekat postoji u nekom vremenskom periodu. Većina objekata koji se pojavljuju u nekom dijagramu interakcije živi tokom celog trajanja interakcije, i svi takvi objekti poravnavaju se na vrhu dijagrama, a njihova linija života crta se od vrha do dna dijagrama. Objekti mogu nastajati za vreme interakcije. Njihove linije života započinju po prijemu poruke u vidu stereotipa create. Objekti mogu biti uništeni za vreme interakcije. Njihove linije života završavaju se po prijemu poruke u vidu stereotipa destroy (i vizuelno se kraj njihovog života obeležava oznakom X).

Takođe u dijagramu sekvenci postoji fokus upravljanja. Fokus upravljanja je uzak, dugačak pravougaonik koji pokazuje period vremena u kome objekat izvodi neku akciju, bilo direktno ili pomoću podređene procedure. Vrh pravougaonika poklapa se sa početkom akcije; dno pravougaonika poklapa se sa završetkom akcije (i može se označiti povratnom porukom).

MODELOVANJE SEKVENCI U VREMENU

Dijagram sekvenci pokayuje jedno odvijanje upravljanja, tj. redosled slanja poruka iymeđu objekata koji učestvuju u interakciji.

Sekvencijalni dijagrami (dijagrami sekevnci) se koriste kada se želi da prikaže protok poruka prema vremenu njihovog pojavljivanja.

Da biste modelovali odvijanje upravljanja po vremenskom redosledu:

- Ustanovite kontekst za interakciju, bez obzira da li je u pitanju sistem, podsistem, operacija ili klasa, ili jedan scenario korisničke funkcije ili društva saradnika.
- Ustanovite scenu za interakciju identifikujući objekte koji imaju ulogu u toj interakciji. Rasporedite ih na dijagramu sekvence sleva nadesno, stavljajući važnije objekte na levu stranu, a njihove susede na desnu.
- Ustanovite liniju života za svaki objekat. Najčešće objekti žive tokom celog trajanja interakcije. Za one objekte koji su nastali i uništeni za vreme interakcije, ustanovite odgovarajuće linije života i eksplicitno naznačite njihovo rođenje i uništenje porukama u vidu stereotipa.
- Započinjući porukom koja inicira posmatranu interakciju, postavite svaku narednu poruku idući od vrha ka dnu između linija života, naznačavajući osobine poruka (kao što su njihovi parametri), u meri potrebnoj za razumevanje semantike interakcije.

- Ako je potrebno da vizualizujete gneždenje poruka ili momente u kojima se odvija stvamo izračunavanje, dodatno obeležite fokus kontrole na liniji života svakog objekta.
- Ako treba da specifikujete vremenska ili prostorna ograničenja, dodatno obeležite svaku poruku vremenskom oznakom i pogodnim vremenskim ili prostomim ograničenjima.
- Ako ovo odvijanje upravljanja treba formalnije da specifikujete, dodajte svakoj poruci početne i krajnje uslove.

Na jednom dijagramu sekvence može da se prikaže samo jedno odvijanje upravljanja (mada jednostavne varijacije možete prikazati pomoću UML-ove notacije za grananja i iteracije). Obično ćete imati više dijagrama interakcije, od kojih su neki osnovni, a drugi prikazuju altemativne putanje ili izuzetne uslove. Za organizovanje ovih kolekcija dijagrama sekvenci možete da koristite pakete, dajući svakom dijagramu pogodno ime po kome ćete ga razlikovati od srodnika.

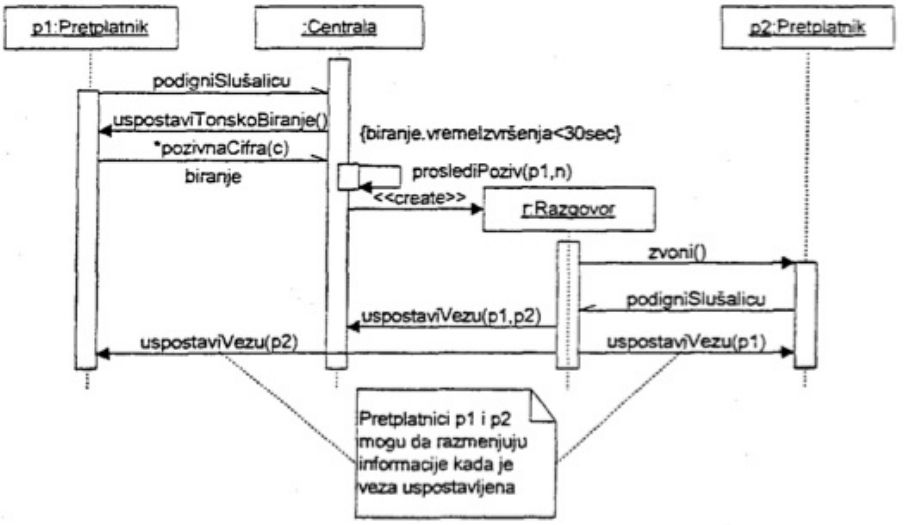
PRIMER - USPOSTAVLJANJE DVOSTRUKE TELEFONSKE VEZE

Dijagram sekvenci može prikazati odvijanje upravljanja u vezi sa uspostavljanjem jednostavne, dvostrane telefonske veze

Na slici 2 prikazan je dijagram sekvence koji specifikuje odvijanje upravljanja u vezi sa uspostavljanjem jednostavne, dvostrane telefonske veze. Na tom nivou apstrakcije, postoje četiri obuhvaćena objekta: dva Pretplatnik (p1 i p2), jedna bezimena telefonska Centrala, i r, Razgovor između dva pretplatnika. Sekvenca započinje kada Pretplatnik(p1) dostavi signal (podigniSlušalicu) objektu Centrala. Sa svoje strane, Centrala dostavlja poziv uspostaviTonskoBiranje tome objektu Pretplatnik, koji potom iterativno šalje poruke pozivnaCifra. Zapazite da ta poruka ima vremensku oznaku (biranje) koja je primenjena kao vremensko ograničenje (njeno vremelzvršenja mora da bude manje od 30 sekundi). Na tom dijagramu nije prikazano šte se događa ako se to vreme prekorači. Za to biste mogli da dodate novu granu ili potpuno odvojen dijagram sekvence. Objekt Centrala zatim poziva samog sebe porukom proslediPoziv. Potom on stvara objekt Razgovor (r), komeprepušta

ostatakposla. Iako u ovoj interakciji nije prikazano, r ima dopunsku odgovomost da učestvuje u mehanizmu naplaćivanja u centrali (što bi se moglo iskazati u drugom dijagramu interakcije). Objekt Razgovor (r) zvoni kod Pretplatnik (p2), koji asinhrono šalje poruku podigniSlušalicu. Objekt Razgovor tada saopštava da Centrala treba da uspostaviVezu i saopštava i jednom i drugom objektu Pretplatnik da uspostaviVezu, posle čega oni mogu da razmene informacije, kao što je naznačeno u pridruženoj napomeni.

Može se desiti da na dijagramima sekvenci treba da modelujete menjanje stanja, uloge ili vrednosti atributa objekta. To možete ostvariti na dva načina. Prvo, taj objekat bi mogao da se pojavi više puta na dijagramu, prikazujući svaki put drugačije stanje, ulogu ili vrednosti atributa, uz upotrebu tranzicije u vidu stereotipa become, koja ukazuje na tu promenu. Drugo, za promenu stanja, možete postaviti ikonu stanja direktno na liniju života tog objekta.



Slika 4.3.2 Primer dijagrama sekvenci za slučaj dvostrane telefonske veze

MODELOVANJE USLOVNIH INTERAKCIJA

Sekvencijalni dijagrami mogu opisivati i uslovljenje sekvencije sa alternativama.

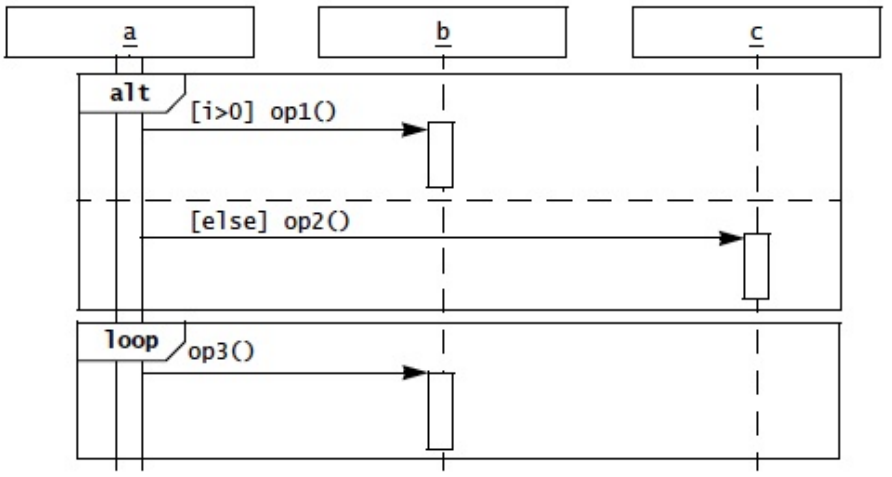
Sekvencijalni dijagrami se mogu koristiti za opis bilo neke apstrakte sekvence (tj. sa svim mogućim interakcijama) bili za opis konkretnih sekvenci (kada se prikayuje samo jedna moguća interakcija.

Kada se pokazuju sve moguće interakcije, skencijalni dijagrami obeybeđuju mogućnost unošenje napomena (notations) u vezi sa iteracijama i uslovima za njehovo aktiviranje.

Iteracija je kombinacija obeleženog fragmenta (dela interkacije) i sa operatorom petlje (slika 3).

Jedna alternativa prikazuje fragment koji sadrži deo za svaku alternativu. Alternative se biraju prema uslovu ispunjenja koji se definiše u prvoj poruci dela (i>0) i sa else, kao što je pokayano na slici 3.

Ako je mogućno, prvo se izvršava alternativa na vrhu sa svim kombinovanim fragmentima i onda se šalje poruka op1(). Ukoliko ne+ije ispunjen neophoddan uslova za ovu realizaciju, iyvršava se donja alternativa i šalje se poruka ot2().



Slika 4.3.3 Primer sa uslovima i iteratorima u skevencijlnim dijagramima

ULOGA SEKVENCIJLNOG DIJAGRAMA U DEFINISANJU KLASA

Sekvencijalni dijagram pokazuje svojim porukama, koje operacije (metode) primajući objekti (klase) moraju da poseduju, jer treba da realiyuju zahteve koje poruke donose.

Sekvencijalni dijagram se može koristiti:

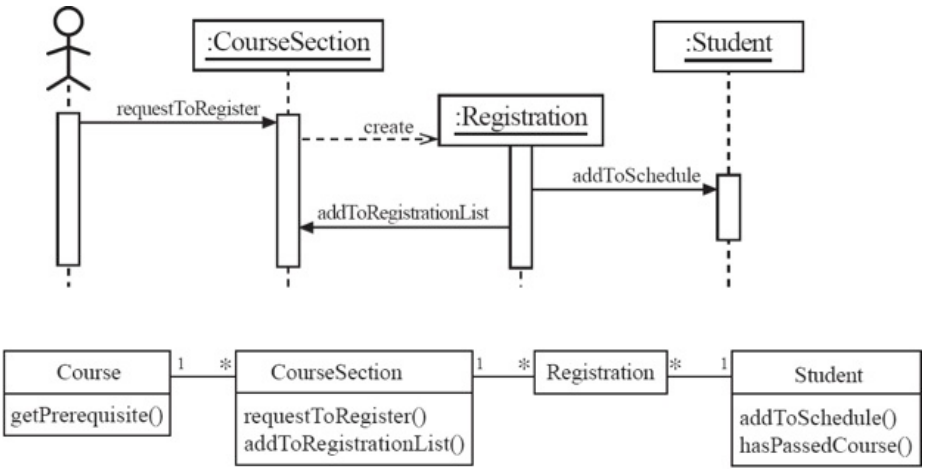
- u prikazu interakcije aktora i sistema u **okviru opisa slučaja korišćenja** softverskog sistema
- u prikazu interakcija **pri konceptualnog projektovaju sistema**, kada se utvrđuju osnovni entiteti (objekti, klase) koji učestvuju u interakcijama
- pri **detaljnom projektovanju interkacija** između klasa, a radi utvrđivanja potrebnih metoda i stributa u klasama.

Ustvari, ako posmatramo proces projektovajja softverskog sistema, kao procesa u više faya, onda u svakoj od njih sekvencijalni dijagram igra važnu ulogu, jer objašnjava razmenu poruka između aktera i objekata sistema, kao i između nih samih. Počev od vrlo apstrakting interakcija, u svakoj sledećoj fazi, interakcije postaju konkretnije, te i objekti koji učetsvuju u toj razmeni prouka, postaju detaljnije definisani.

Projektanot sistema, uz korišćenje odgovarajućeg softverskih alata (kao na primer, Power Designer), kreiraju sve detaljnije sekvencijalne dijagrame, u saglasnosti sa njihovim napredovanjem u sagledavanju načina rada budućeg softverskog sistema, kao i njegovih elemenata. Konačan cilj je da se vrlo detaljno definisu svi potrebni objekti i sve poruke koje

oni treba da razmene, kako bi se na taj način, definisali svi metodi (ili operacije u UML terminologiji) i svi atributi, koje treba da sadrže klase koje definišu ove objekte.

Na slici 4 prikazan je primer sekvencijalnog dijagrama lpki opisuje interkaciju koja se odvija u slučaju izbora izbornog predmeta nekog programa od strane studenta (ref, Lethbridge/ Laganière 2005).,



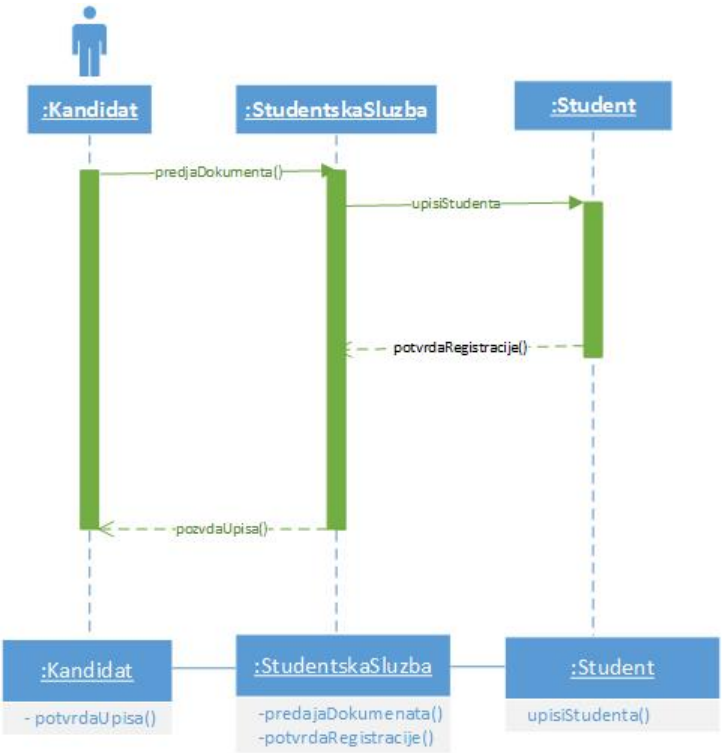
Slika 4.3.4 Poruke između objekata definišu operacije (metode) tih objekata

Poruku koju objekat prima određuje operaciju (metod) koju objekat treba da poseduje, jer se očekuje da on realiyuje zahtev koji poruka donosi. Nayivi poruka na slici 4 su istovremeno i nazivi operacije (metoda) kojeprimajući objekti moraju da poseduju. U prvim fazama projektovanja, to ne mora tako da bude. Međutim, u konačnoj fazi, fazi detaljnog projektovanja, to tako mora da bude.

PRIMER: UPIS STUDENTA NA STUDIJE - PRVA FAZA

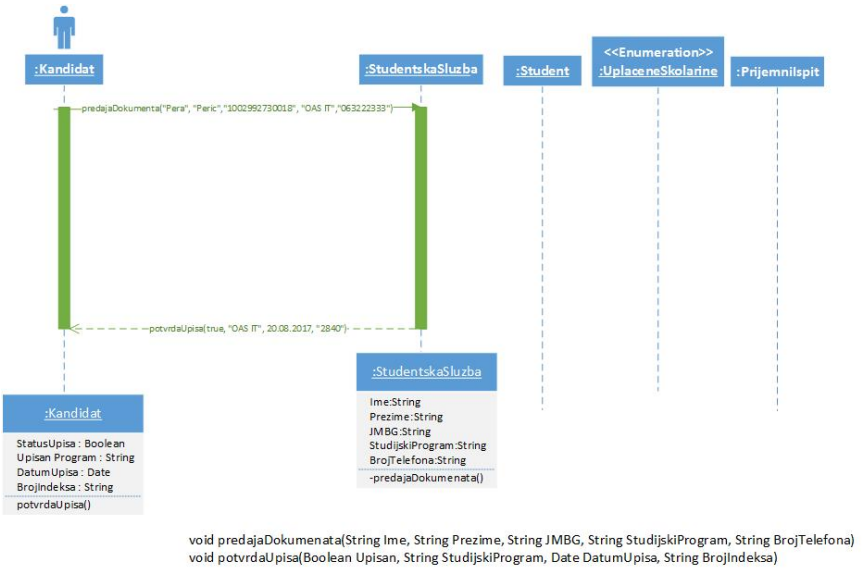
Naziv poruka, određuju nazive operacija (metoda) primajućih klasa, a njihovi argumenti, definišu atribute primajućih klasa.

Slika 5 prikazuje sekvencijalni dijagram upisa jednog studenta na studije, i to u početnoj, vrlo apstraktnom obliku.



Slika 4.3.5 U početnoj fazi,projetant je definisao tri klase objekata

U sledećoj fazi, projektan uviđa koje argumente treva da sadrži poruka *predajDokumentaciju().* Isto raku, uvrđuje i argumente poruke **potvrdaUpisa()** mora da sadrži (slika 6).



Slika 4.3.6 Argumenti u porukama određuju atribute primajućeg objekta

Na opisan način određeni su atributi i operacije klsa **:Kandidat** i **:StudentskaSluzba.** Na slici su posebno prikazane kreirane operacije (metodi) u ovim klasama.

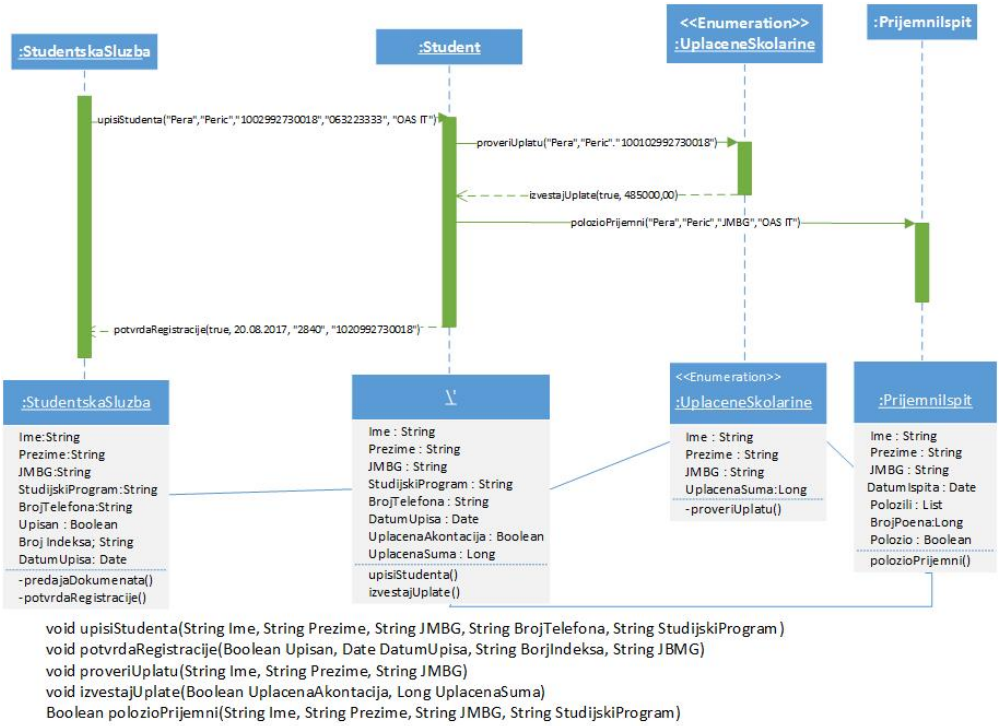
PRIMER: UPIS STUDENTA NA STUDIJE - DRUGA FAZA

Poruke u sekvencijalnim dijagramima određuju nazive operacija (metoda) objekata koji ih primaju, a argumeti ovih poruka, definišu atribute ovih primajućih objekata

U toku projektovanja softverskog sistema, projektant je primetio da pored tri klase objekata, prikazanih na slici 5, tj. klasa Kandidat, StudentskaSluzba i Student, da je potrebno da u ovom slučaju korišćenja sistema (upis studenta na studije), dodati još dve klase objekata UplaćeneSkolerine i Prijemni Ispit, jer se student može upisati samo ako je uplatio akontaciju školarine i ako je položio prijemni ispit. Slika 7 prikazuje pve dodatne klase u sekvencijalnom dijagramu, kao i poruke koje one razmenjuju sa ostalim klasama.

Prikzane poruke određuje nazive operacija (metoda) objekata koji ih primaju, a argumeti ovih poruka, definišu atribute ovih primajućih objekata. Ove operacije su posebno prikazane u donjem delu slike.

Na prikazan načun, primenjujući sekvencijalne dijagrame ya svaki slučaj korišćenja sistema



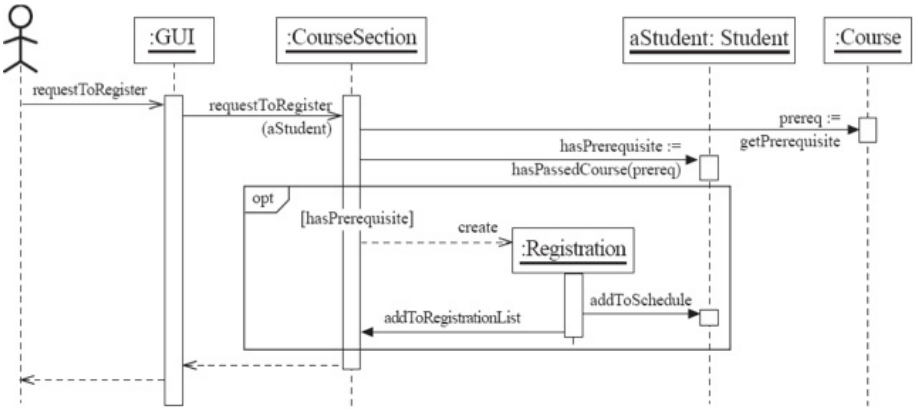
Slika 4.3.7 Definisanje još dve klase i svih prouka koje raymenjuju prikazane četiri klase

SLOŽENIJI SEKVENCIJALNI DIJAGRAMI

U sekvencijalne dijagrame mogu dodati nove interakcije ili petlje sa iterativnim ponavljanjem poruka, au vidu pravougaonika u koji je ucrtana interakcija,

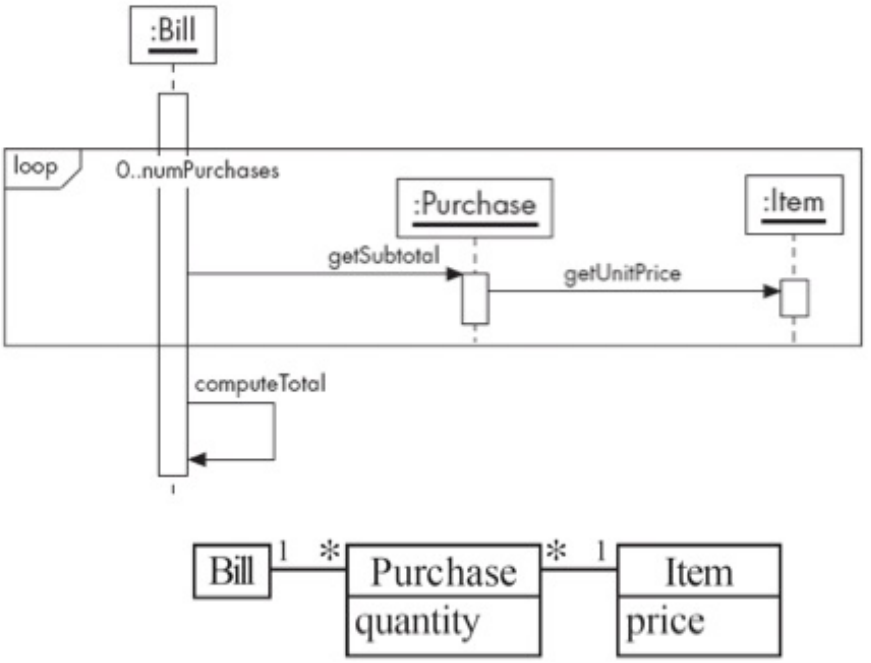
Sekvencijalni dijagrami poprimaju složenij oblik, ako sadrže i opcione varijante interakcija ili alternativne interakcije.

Na slici 8 prikazan je detaljniji sekvencijalni dijagram sa slike 4 koji prikazuje dodatnu opcionu interakciju vezanu za klasu **CourseSection (**Lethbridge/Laganière 2005). Ta dodatna interakcija proverava da li su prilikom izbora kursa zadovoljeni definisani preduslovi za izbor kursa.



Slika 4.3.8 Dijagram iterakcije sa slike 4, ali sa dodatom interakciom

Ukoliko se neka poruka iterativno ponavlja, onda se pre njenog naziva stavlja znal zvezide (*). Na slici 9 prikazana je petlja u kojoj se poruka upita za kupovinu iterativno ponavlja.



Slika 4.3.9 Sekvencijalni dijagram sa petljom za iterativno slanje poruke

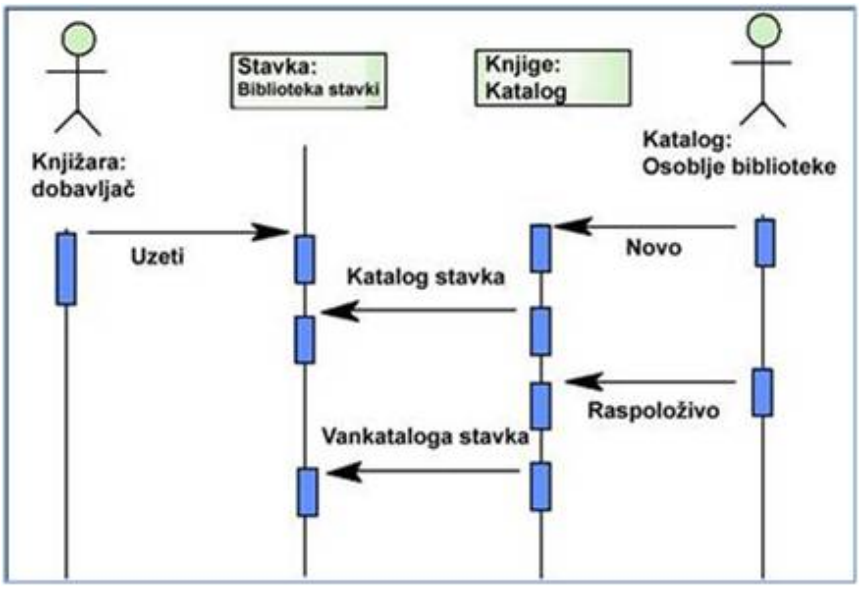
PRIMENA UML SEKVENCIJALNIH DIJAGRAMA U SLUČAJEVIMA KORIŠĆENJA

Sekvencijalni dijagram prikazuje redosled interakcija aktera sa osnovnim objektima sistema, ali prikazuje i interakciju između ovih objekata.

Pored tekstualnog opisa scenarija, koji opisuju svaki slučaj korišćenja, moguće je ya njihov i grafički opis, koristiti tzv. sekvencijalne dijagrame UML sistema, Unutar UML-a sekvencijalni dijagrami mogu biti korišćeni da dodaju informacije u slučaj korišćenja. Ovi sekvencijalni dijagrami pokazuju kako su akteri uključeni u interakciju, objekte unutar sistema sa kojima su akteri u interakciji. Sekvencijalni dijagrami koji se koriste radi opisa scenarija su prilagođeni svrsi, te ne sadrže detalje koji koriste sekvencijalni dijagrami kada se koriste za projektovanje klasa. Sekvencijalni dijagrama koji se koriste za opis scenarija, koji se odnose na slučajeve korišćenja, sadrže apstraktno definisane entitete ili objekte koji realizuju definisan scenario. Linije sa strelicama prikazuju operacije koje se realizuju u vezi sa prikazanim objektima.

Na slici 18 je prikazana interakcija obuhvaćena uzimanjem i katalogiziranjem knjiga za biblioteku. Slika ilustruje da postoje dva objekta klasa Biblioteka stavki i Katalog obuhvaćeni slučajem korišćenja Upravljanje katalogom knjiga. Sekvenca akcija je od vrha naniže i oznake na strelicama između aktera i objekata ukazuju na ime operacije. Zbog toga, kada je knjiga kupljena, operacija Novo je izvedena na Katalogu i operacija Uzeti na Stavki. Jednom kada je knjiga dostupna, Katalog stavka je izvedena na Stavki.

.

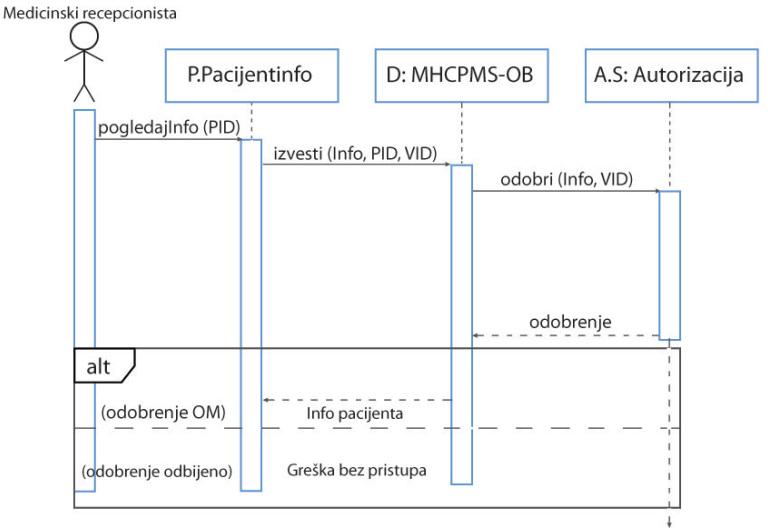


Slika 4.3.10 0 Primene UML sekvencijalnog dijagrama za opsiscenarija slučaja korišćenja

PRIMER: PREGLED INFORMACIJA O PACIJENTU

Primer pokazuje relizaciju osnovnog (primarnog) i alternativnog (sekundarnog) scenarija

Sekvencijalni dijagram na slici 10 na grafički način opisuje interakciju slučaja korišćenja "Pregled informacija pacijenta". U gornjem delu je dat osnovni (primarni) scenario, a u donjem je dat alternativni, tj. pomoćni ili sekundarni scenario. On opisuje vanrednu situaciju do koje dolazi zbog prekida pristupa objektu A.S. U kome se vrši autorizacija.
Pored sekvencijalnog dijagrama, isti scenario se može opisati i tekstualno, navođenjem redosleda akcije aktera i sistema.



Iz dijagrama na slici 1 čita se sledeći scenario interakcije:

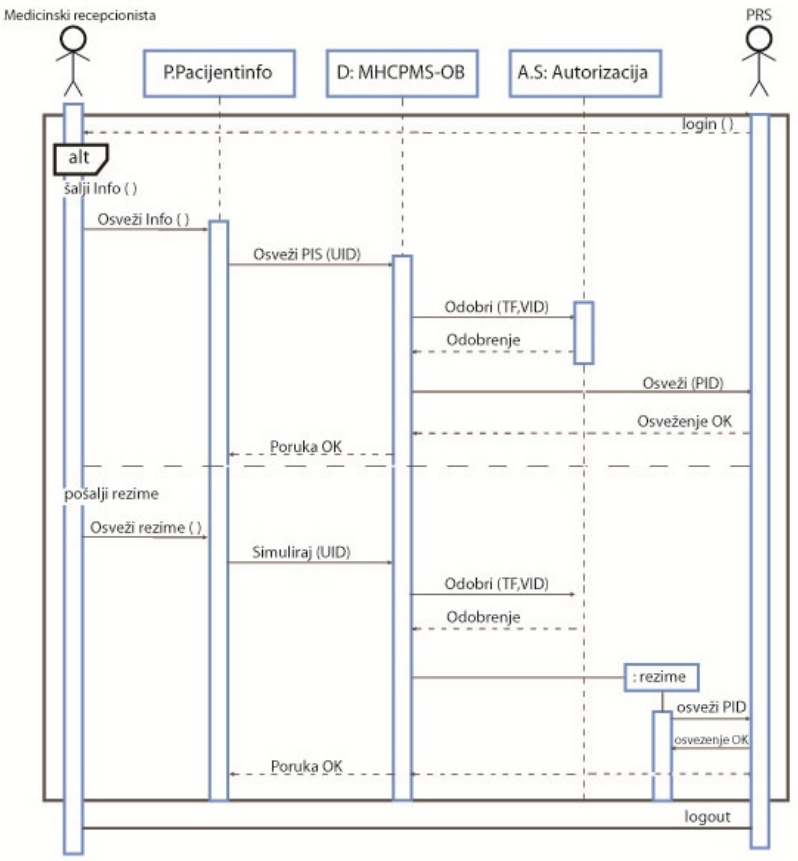
1. Medicinski recepcioner pokreće ViewInfo metod u objektu O klase PatientInfo, ubacujući identifikator pacijenta PID, P je objekt za interakciju sa korisnikom, prikazan u vidu forme sa podacima o pacijentu.
2. Objekt P poziva bazu podataka da vrati traženu informaciju, unoseći identifikacioni broj recepcionera radi sigurnosne provere.
3. Baza podataka proverava sa sistemom za proveru ovlašćenja da li je korisnik (recepcioner) autorizovan za ovu akciju, tj. da li ima dozvolu za ovu akciju.
4. Lična informacija se šalje direktno sa objekta korisničkog interfejsa u PRS sistem. Alternativno, može se formirati slog iz baze podataka i onda se taj slog šalje.
5. Po završetku transfera, PRS šalje poruku o statusu i korisnik se isključuje. Ako se dijagram interakcije ne koristi za generisanje izvornog koda ili detaljnu dokumentaciju, on ne mora da sadrži svaku interakciju. Često se detalji o interakcijama ostavljaju za fazu implementacije (kodiranja) te dijagrami interakcije u fazi

PRIMER SCENARIJA INTERAKCIJE

Primer za bolje razumevanje razvijanja scenarija

Ako se dijagram interakcije ne koristi za generisanje izvornog koda ili detaljnu dokumentaciju, on ne mora da sadrži svaku interakciju. Često se detalji o interakcijama ostavljaju za fazu implementacije (kodiranja) te dijagrami interakcije u fazi definisanja zahteva ili koncepcijskog projektovanja ne sadrže mnoge poruke i detalje.To je ostavljeno za fazu implementacije. Na primer, odluka o tome kako dobiti identifikator za dobijanje ovlašćenja (autorizaciju) korisnika, slučaja na slici 11 , je odložena. U fazi implementacije ovo je uključeno u interakciju sa

objektom Korisnik, ali to nije važno u ovoj fazi i ne mora da bude uključeno u sekvencijalni dijagram



ZADATAK ZA SAMOSTALNI RAD - DIJAGRAM INTERAKCIJA

Ovaj zadatak ima za cilj da studenti obnove znanje UML i njegove primene

Za potrebe video kluba, potrebno je definisati zahteve koje treba opisati pomoći dijagrama zahteva. Nakon toga potrebno je dizajnirati slučajeve kroišćenja. Kada se završi sa osnovnim modelovanjem zahteva, potrebno je kreirati dijagram klasa sa odgovarajućim atributima i metodama. Za kreirane dijagrame slučajeva korišćenja treba kreirati odgovarajuće SSD dijagrame za glavne scenarije. Nakon toga potrebno je definisati detaljne sekvencijalne dijagrame koji će biti izrađeni na osnovu scenarija koje student bude zadao. Iz sekvencijalnog dijagrama potrebno je izgenerisati dijagram kolaboracije.

Ovaj zadatak treba da se odradi korišćenjem Power Designer alata. Studentu se ostavlja na raspolaganje da razvije ideju i zahteve prema svojoj mašti. Potrebno je da zahtevi budu realni i obuhvatju osnovne scenarije rada video kluba kao što je poznato.

4.4 Sistemski dijagram sekvenci

UVOD

Pokazni primeri daju primere promene Power Designer alata u UML moldeliranju i dobijanju različitih dijagarama koje UML omogućava.

- Cilj vežbe je da se se student upozna sa predmetom i osnovnim pravilima predmeta. U ovim vežbama student će se upoznati sa Power Designer softverskim alatom koji će biti korišćen na ovom predmetu. Na ovom času ćemo prikazati kako se koristi UML u okviru softverskog alata PowerDesigner-a. Biće prikazano kako se kreira projekat i uvod u klasne dijagrame, odnosno kako se kreiraju klase i interfejsi. Kako se postavljaju atributi klasa i operacije klasa. Zatim ćemo pokazati kako se uspostavljaju veze između elemenata i to upotrebom generalizacije, implementacije, asocijacije i agrtegacije.
- Korisnički alat je neophodan korak u procesu modelovanja aplikacije. Potrebno je da dobro poznajemo UML. Crtanje UML-a bez alata je prilično teško tako da je bitno da naučimo da koristimo neki alat za kreiranje UML dijagrama. Posebna pogodnost je ako koristimo isti alat za UML i za programiranje, time dobijamo mogućnost da kod bude automatski pretvoren u UML dijagram i obrnuto. Naravno, ovaj postupak može da se izvrši i sa drugim alatima samo zahteva više koraka.
- Najbitniji deo razvoja softverskog inženjerstva jeste ispuniti zahteve korisnika i udovoljiti njegovim željama i potrebama. Odatle je i nastala izreka “Ukoliko korisnik nije zadovoljan, niko nije zadovoljan”. Svaki uspešan deo softvera nastao je kao korisnikova velika ideja. Vaš posao kao profesionalnog softverskog inženjera jeste da njegove ideje sprovedete u delo i jednostavno ih oživite. Uzimajući nejasnu ideju i sporvesti je u rad – zadovoljivši klijenta / nije tako lako. U ovim vežbama biće demonstriran način kreiranja zahteva korišćenjem Power Designer modela zahteva i njegovo povezivanje sa drugim delovima sistema.
- Takođe, biće objašnjen način modelovanja dijagrama slučajeva korišćenja (DSK) i njegova primena korišćenjem PowerDesigner alata.

SISTEMSKI DIJAGRAM SEKVENCI

Sistemski dijagram sekvenci (SSD) prikazuje interakciju spoljnih aktera sa sistemom i kreira se na osnovu opisa dijagrama slučajeva korišćenja

U prethodnim vežbama data je analiza i upotreba dijagrama slučajeva korišćenja. Kao što je već rečeno, dijagrami slučajeva korišćenja opisuju kako akteri koriste sistem. Tokom tog korišćenja sistema akteri izvršavaju (generišu) sistemske događaje, obično inicirajući određenu sistemsku operaciju koja će obraditi njihov zahtev. Upravo u svrhu prikazivanja događaja koje akter generiše u okviru slučaja korišćenja koriste se sistemski dijagrami.

Sistemski dijagram sekvenci (SSD) prikazuje interakciju spoljnih aktera sa sistemom. Kreira se na osnovu opisa dijagrama slučaja korišćenja. SSD prikazuje određeni događaj (sekvencu) u okviru jednog slučaja korišćenja i aktere koji direktno koriste tu funkcionalnosti sistema.

Dakle, jedan SSD dijagram odgovara jednom slučaju upotrebe. Prilikom kreiranja SSD dijagrama opisuje se samo glavni tok, odnosno primarni scenarijo slučaja korišćenja.

Važno je napomenuti da se u ovom dijagramu sistem prikazuje kao „crna kutija“ odnosno ne ulazi se u detalje objekata unutar sistema i njihove komunikacije, nego samo prepoznavanje spoljnih aktera i komunikaciju odnosno događaje koje generišu akteri i rezultate koje daje sistem određenom akteru.

Sistemski dijagram sekvenci pored interakcija spoljnih aktera sa sistemom može se koristiti i za prikazivanje interakcije između različitih sistema. Na primer, sistem studentske službe može da šalje zahteve finansijskoj službi univerziteta kako bi proverili finansijsko zaduženje studenta (radi prijave ispita, ukoliko se želi zabraniti studentu izlazak na ispit jer nije izvršio sve obaveze plaćanja).

Sistemski dijagram sekvenci predstavlja apstraktni nivo prikaza, od samog sekvencijalnog dijagrama. U praksi se najčešće koristi isti model prilikom izrade obe vrste dijagrama, a razlikom u nivou detalja.

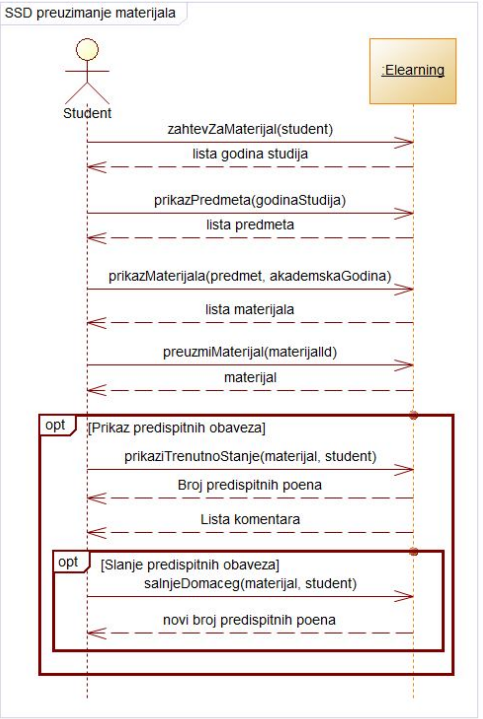
SSD MODELIRANJE I EKSTERNI SISTEMI

Grafički prikaz SSD dijagrama kroz primer i objašnjenje komponenti dijagrama.

Na slici 1 . prikazan je primer sistemskog dijagrama sekvenci za preuzimanje materijala preko Eleaarning sistema (obuhvaćen je primerni scenario). Akter (student) se prikazuje istim simbolom kao i na dijagramu slučajeva korišćenja, s tom razlikom što sada ima isprekidanu liniju „liniju života“. Sistem je grubo prikazan kao Elearning u okviru pravougaonika takođe sa linijom života.

Na početku student inicira komunikaciju, zahteva materijal od sistema pri čemu se prosleđuju podaci o tom studentu, zato što sistem vraća godinu studija u zavisnosti na kojoj se student nalazi. Vrlo je važno prepoznati parametre koje akter treba da prosledi sistemu da bi on dao korektan rezultat. Metode napisane u zahtevu (puna linija) kasnije postaju metode klasnog dijagrama, pa zbog toga treba voditi računa kako se definišu.

Isprekidanom linijom obeležavaju se povratne poruke gde sistem vraća određene rezultate akteru. U okviru drugog dela dijagrama prikazanog na slici 1 može se primetiti uokvireni pravougaonici. Oni omogućavaju opcije, uslove ili petlje koje treba da se evidentiraju u okviru SSD dijagrama.
U ovom slučaju ovo je opcioni deo gde student bira da li želi prikaz predispitnih obaveza. Ukoliko želi, prosleđuje koji student i za koji materijal želi uvid i sistem mu vraća dva odgovora pa zbog toga ima i dve povratne strelice. Ovo podrazumeva da sistem ažurira npr. dva različita polja na ekranu ili vuče podatke iz dve različite tabele. Takođe ukoliko se student opredeli za prikaz predispitnih poena može opciono da izabere i slanje novih domaćeg zadatka. U okviru toga poziva metodu slanje domaćeg sa parametrima, a sistem vraća nakon pregledanja novi broj predispitnih poena. Ovaj primer služi samo za demonstraciju dijagrama i nije realan.



Slika 4.4.1 Prikaz sistem sekvencijalnog dijagrama korišćenjem Power Designer-a

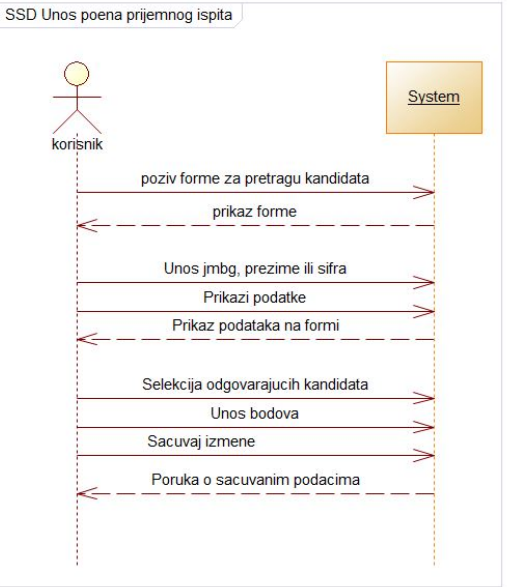
POKAZNI PRIMERI

Pokazni primeri upotrebe sistemskog dijagrama sekvenci na osnovu glavnog scenarija slučaja korišćenja koji su prikazani tabelarno

- Na slici 2 i 3 prikazane su tabele komunikacije spoljnog aktora i sistema koje su izvučene iz glavnog scenarija slučaja korišćenja. Za Ova dva primera data su i dva sistemska dijagrama sekvenci korišćenjem Power Designer alata.

akcija učesnika	**odgovor sistema**
1. Poziva formu za unos podataka	Prikazuje formu
2. Unosi ime, jmbg ili sifru kandidata čije poene želi da unese	
3. Poziva sistem da pronađe kandidata	4 ispisuje podatke na formi
5. Bira iz liste kandidata čije poene želi da unese	6. Sistem poziva formu za unos poena sa prijemnog ispita
7. Unosi poene sa prijemnog ispita	
9. Potvrđuje unos	10. Sistem prihvata podatke i pamti u bazu

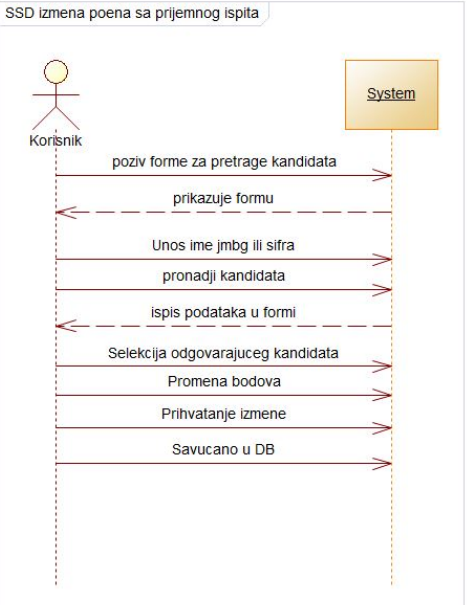
Slika 4.4.2 Tabela prvog primera



Slika 4.4.3 Sistemski sekvencijalni dijagram dizajniran na osnovu scenarija sa tabele prikazane na slici 2

akcija učesnika	**odgovor sistema**
1. Počinje pozivom odgovarajuće forme za pretragu kandidata	2. Sistem poziva formu za pretragu kandidata
3. Unosi ime, jmbg ili sifru kandidata čije podatke želi da promeni	
4. Poziva sistem da pronađe kandidata	5. Povlači podatke iz baze podataka i ispisuje na formi
7. Bira iz liste kandidata čije podatke želi promeniti	
8. Unosi izmene podataka	
9. Potvrđuje izmenu	10. Sistem prihvata izmenu i pamti u bazu

10.1: ukoliko su pogrešno uneti podaci, sistem prijavljuje grešku adekvatnom porukom



ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji se odnose na sistemske dijagrame sekvenci.

Zadatak 1: Sistem za apoteku. Apoteka poseduje dva magacina koji u sebi sadrže lekove. Svaki lek u sebi sadrži informaciju o imenu i datumu isteka. Apoteka vodi evidenciju svoji klijenata. Više klijenata može kupovati više lekova. Od podataka klijenta čuvaju se informacije (ime, prezime, adresa). Napraviti odgovarajući klasni dijagram i transformisati taj dijagram u kod. Nakon transformacije dijagrama kreirati main metodu koja će demonstrirati prvo kreiranje magacina, zatim 3 leka u magacinu, kreiranje jednog klijenta i njegovou kupovunu 2 leka.

Modelovati sistemski dijagram sekvenci za opisani sistem.

Zadatak 2: Aplikacija za prodaju mobilnih uređaja. Aplikacija se bavi online prodajom (kupovinom) mobilnih uređaja, zavisno da li korisnik pristupa sistemu kao prodavac (ima mogućnost dodavanja uređaja za prodaju i odgovaranju poruka koje je dobio od kupaca) ili korisnik pristupa sistemu kao kupac (ima mogućnost kupovine i kontaktiranja prodavca). Korisnik ne može koristiti ovaj sistem bez naloga.

Modelovati sistemski dijagram sekvenci za opisanu aplikaciju.

4.5 Dijagram stanja

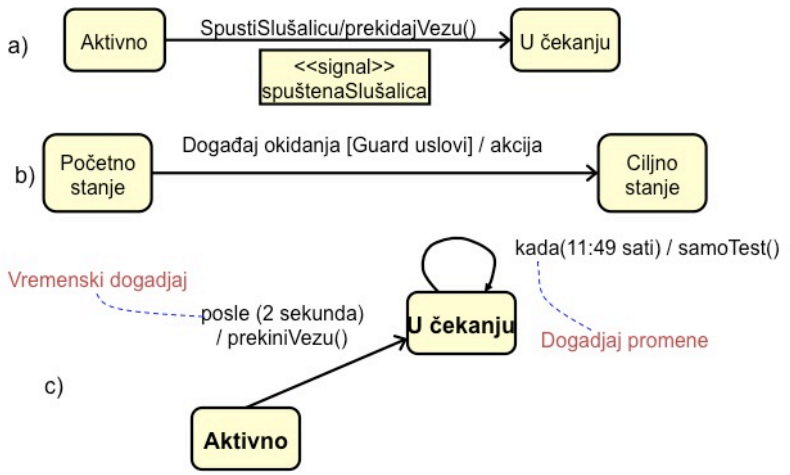
POJMOVI

Stanje je uslov ili situacija u toku života nekog objekta pri kojoj objekat zadovoljava neke uslove, izvršava neke aktivnosti ili čeka na neki događaj

Stanje je uslov ili situacija u toku života nekog objekta pri kojoj objekat zadovoljava neke uslove, izvršava neke aktivnosti ili čeka na neki događaj. Objekat može da promeni svoje stanje ako se pojavi određeni događaj. U UML-u, stanje objekta se predstavlja pravougaonikom sa zaobljenim uglovima (slika 1 .a)

Dogadjaj je specifikacija neke značajne pojave koja ima lokaciju u vremenu i prostoru. Događaj se opisuje sa: signalom, vremenom i promenom stanja. Signal predstavlja imenovani objekat koji jedan objekat asinhrono šalje, a drugi objekat ga prima **(**slika 1 .b). **Vremenski dogadjaj** je dogadjaj koji predstavlja određeni vremenski trenutak. **Dogadjaj promene** je dogadjaj koji predstavlja promenu stanja ili zadovoljenje nekog uslova. Znači, kada pod određenim okolnosti objekat primi određeni signal, dolazi do promene njegovog stanja, tj. desio se određeni događaj. Promenom stanja, menjaju se vrednosti jednog ili više atributa objekta (slika 1 .c) .

Tranzicija je relacija između dva stanja koji pokazuje da objekat u prvom stanju izvršava neke akcije i ulazi u drugo stanje kada se navedeni događaj pojavi i kada se navedeni uslovi zadovolje (slika 1 .b),



Slika 4.5.1 Promena stanja objekta

Dogadjaj okidanja – događaj čiji prijem kod objekta u početnom stanju izaziva tranziciju sposobnu da se realizuje, ako su njegovi povezani uslovi – zadovoljeni;

Uslovi čuvanja stanja (guard uslovi) – Logički izraz koji se izvršava kada je pokrenuta tranzicija prijemom događaja okidanja;

Akcija - Izvršavanje neke elementarne akcije koje direktno deluje na objekat koji poseduje mašinu stanja, i indirektno – na druge objekte koji su vidljivi objektu.

UML DIJAGRAM STANJA

Modelovanje vođeno događajima pokazuje kako sistem odgovara na eksterne i unutrašnje događaje. UML podržava modelovanje podržano događajima sa dijagramom stanja.

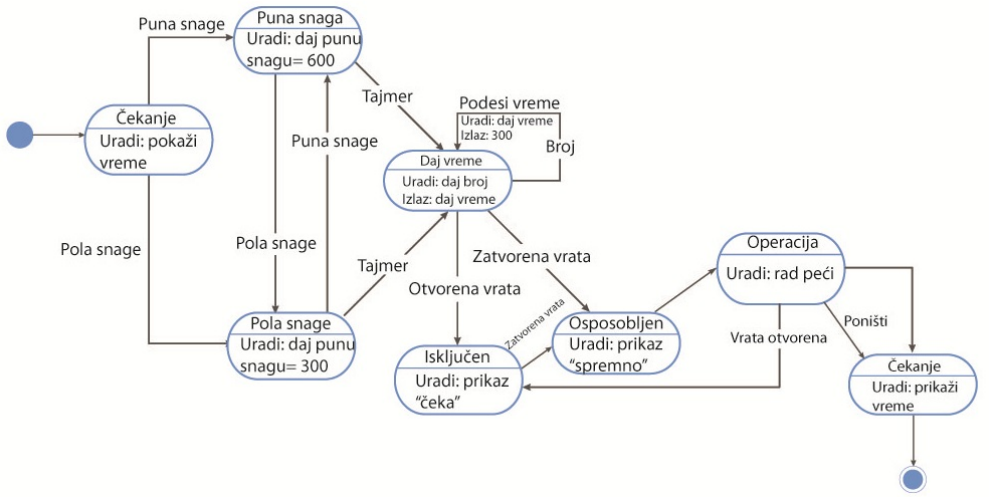
Modelovanje vođeno događajima pokazuje kako sistem odgovara na eksterne i unutrašnje događaje. Polazi se od pretpostavke da sistem ima konačan broj stanja i da događaji (kao podsticaji) prouzrokuju prelazak sistema iz jednog stanja u drugo. Ovo se naročito primenjuje kod sistema koji rade u realnom vremenu (tzv. sistemu u realnom vremenu).

UML podržava modelovanje podržano događajima upotrebom dijagrama stanja i događaja, vrše prelazak sistema iz jednog stanja u drugo. Oni ne pokazuju tok podataka unutar sistema, ali mogu da sadrže dodatne informacije vezane za proračune koji se rade u svakom stanju.

Na slici 2 dat je primer dijagrama stanja u cilju modelovanja rada mikrotalasne peći, metodama modelovanja vođenim događajima. Svako stanje je označeno pravougaonikom sa zaobljenim uglovima.

.

Strelice pokazju podsticaje (događaje) koji dovode do promene stanja sistema. Krgovi pokazuju početak i kraj promene stanja.



Slika 4.5.2 Dijagram stanja u mikrotalasnoj peći

DEFINISANJE STANJA I DOGAĐAJA

Stanje je definisano vrednostima atributa objekta. Događaji dovode do promene stanja objekta.

U Tabeli 1i 2 prikazana su stanja i podsticaji u radu mikrotalasne peći

Podsticaj	Opis
Pola snage	Korisnik je pritisnuo dogme za pola snage
Puna snaga	Korisnik je pritisnuo dugme za punu snagu
Tajmer	Korisnik je pritisnuo jedno od dugeta tajmera
Broj	Korinik je pritisnuo dugme sa brojevima
Otvorena vrata	Prekidač na vratima nije nije zatvoren
Zatvorena vrata	Prekidač na vratima je zatvoren
Start	Korisnik je pritisnuo dugme za start
Poništeno	Korisnik je pritisnuo dugme "Poništi"

Slika 4.5.3

Stanje	Opis
Čekanje	Peć čeka ulaz. Displej pokazuje trenutno vreme.
Pola snage	Peć radi sa snagom od 300 vati. Displej pokazuje "Pola snage"
Puna snaga	Peć radi sa snagom 600 vati. Displej pokazuje "Puna snaga".
Podešeno vreme	Vreme kuvanja je podešeno prema unetoj vrednosti od strane korisnika. Displej pokazuje vreme trajanja kuvanja i prikazuje unetu vrednost.
Isključeno	Rad peći je zaustavljen iz bezbednosnih razloga. Displej pokazuje "Nije spreman"
Uključen	Peć je uključena. Isključeno je unutrašnje svetlo. Displej pokazuje "Spermno za kuvanje"
Radi	Peć je u radu. Unutrašnje svetlo je upaljeno. Displej pokazuje preostalo vreme kuvanja. Na kraju kuvanja, čuje se zujalica u trajanju od 5 sekundi. Svetlost u peći je uključena. Displej pokazuje "Kuvanje je gotovo" dok radi zujalica.

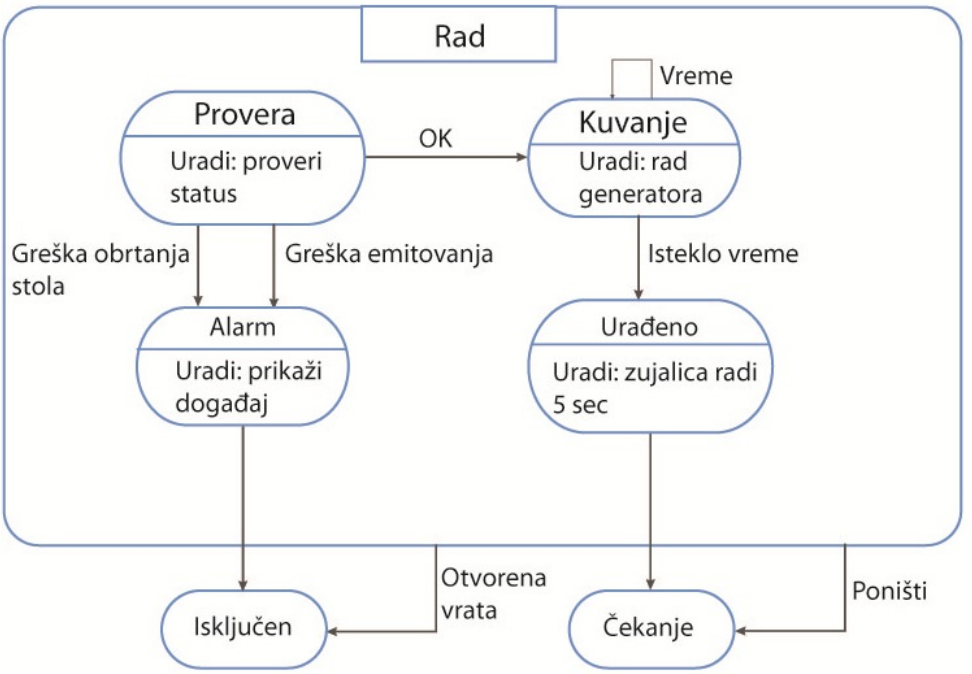
Slika 4.5.4

HIJERARHIJA STANJA

Super stanje sadrži više posebnih stanja.

Problem kod modelovanja stanja sistema je u tome što se broj mogućih stanja brzo povećava sa složenošću sistema. Kod velikih sistema, dijagram stanja, zbog toga, ne može da prikaže sve detalje, već samo pokazuje samo najvažnije informacije. Da bi se ovaj problem rešio, mogu se koristiti stanja na više nivoa, tj. koristiti tzv. super stanja. To znači da super stanje sadrži više posebnih stanja. U dijagramu stanja, super stanje izgleda kao i svako drugo stanje, ali se ono na „dodir", može detaljnije prikazati sa svim detaljnim stanjima (kao pod stanjima). Stanje „Radi" na slici 1 je na slici 3 prikazano, kao super stanje, u detaljnom obliku, sa svojim posebnim (pod) stanjima.

Super stanje „Radi" sadrži više pod-stanja. Rad počinje sa proverom statusa, i ako ima problema uključuje se alarm i zaustavlja se kuvanje. Kuvanje se odvija uključivanjem generatora mikro talasa u određenom trajanju. Kad je vreme trajanja kuvanja isteklo, zujalica zuji. Ako se vrata otvore za vreme rada, sistem odmah prelazi u stanje prekida rada, kao što je prikazano na slici 3.



Slika 4.5.5

ZADACI ZA SAMOSTALNI RAD

Dati su zadaci koji se odnose na dijagrame stanja.

Zadatak 1: Modelovati dijagram stanja za aplikaciju za naručivanje autodelova. Aplikacija omogućava svakom korisniku laku kupovinu bilo kog autodela koji mu je potreban i brzu isporuku na kućnu adresu.

Zadatak 2: Modelovati dijagram stanja za proizvoljnu aplikaciju. Prikazati hijerarhiju stanja i definisati događaje.

4.6 Dijagram aktivnosti

AKTIVNOST I DIJAGRAM AKTIVNOSTI

Dijagram aktivnosti prikazuje tok upravljanja od aktivnosti do aktivnosti.

Dijagram aktivnosti prikazuje tok upravljanja od aktivnosti do aktivnosti. Upotrebljava se za modeliranje dinamičkog aspekta sistema. Modelira se redosled koraka u procesu računanja (modeliranje operacije). Dijagramom aktivnosti se modelira tok menjanja stanja nekog objekta u različitim trenucima toka upravljanja (tok rada).

Aktivnost je neatomsko izvršenje unutar mašine stanje. Dovodi do neke akcije, koju čine atomska izračunavanja koja dovode do promene stanja sistema ili dobijanja neke vrednosti. Upotrebljava se i radi konstruisanja izvršnih sistema.

Dijagrami aktivnosti pokazuju proceduralni tok upravljanja između dva ili više objekata za vreme izvršavanja neke aktivnosti. Dijagrami aktivnosti se mogu koristiti za modeliranje poslovnih procesa (koji su višeg nivoa, tj. apstrakniji nego mnogi modeli sofverskih sistema). Međutim, mogu se koristiti i za modeliranje unutrašnjih akcija klasa, na mnogo nižem nivou. Ipak, dijagrami aktivnosti se najčešće koriste za modeliranje poslovnih procesa koji objašnjavaju kako se u preduzeću nešto radi. Njih i lakše razumeju poslovni ljudi, jer je manje tehnički orijentisan (u odnosu na sekvencijalne dijagrama, na primer).

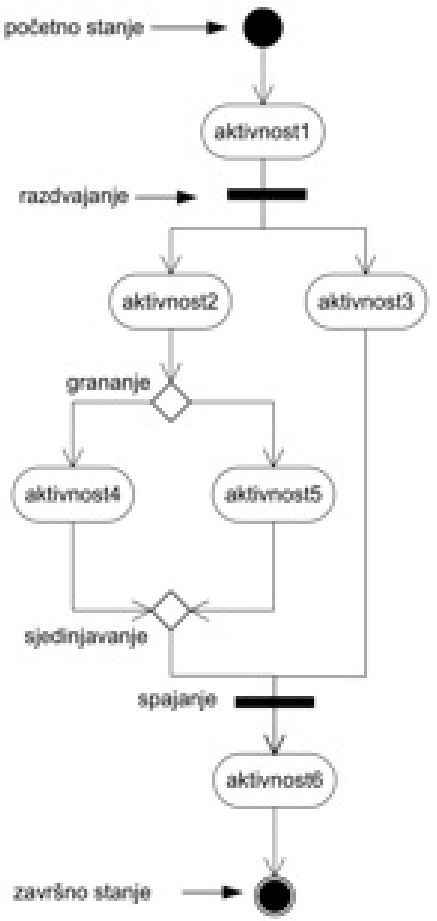
Grafički simboli koji se koriste u dijagramima aktivnosti su slični simbolima koji se koriste u dijagramima stanja. Kao i dijagram stanja, i dijagram aktivnosti polazi od crnog kruga koji je povezan sa početnom aktivnošću. Aktivnost se predstavlja pravougaonikom sa zaobljenim uglovima, unutar kojeg se upisuje naziv aktivnosti. Aktivnosti se međusobno povezuju linijama tranzicije, a mogu i da vode do tačaka odlučivanja u kojima dolazi do račvanja linija tranzicije. Tačke odlučivanja se prikazuju zaokrenutim kvadratič. Za svaku granu definišu se uslovi za aktiviranje tranzicije, tekstualno napisanim između srednjih zagrada. Završna aktivnost se povezuje sa završnom tačkom, koja je predstavljena sa dva koncentrična kruga, od kojih je manju popunjen crnom bojom

NAČIN PRIKAZA DIJAGRAMA AKTIVNOSTI

Dijagrami se čitaju odozgo na dole i imaju grananje i razdvajanje za opisivanje uslova i paralelnih aktivnosti.

Dijagram aktivnosti prikazuje tok aktivnosti kroz sistem. Dijagrami se čitaju odozgo na dole i imaju grananje i razdvajanje za opisivanje uslova i paralelnih aktivnosti. Razdvajanje se koristi kada se javi veći broj aktivnosti u isto vreme. Na slici 1 nakon aktivnosti aktivnost1 sledi razdvajanje, što znači da se aktivnosti aktivnost2 i aktivnost3 odvijaju paralelno u istom vremenu. Nakon aktivnosti aktivnost2 javlja se grananje, koje se prikazuje simbolom odluke (prazan romb). Grananje opisuje koje aktivnosti će da se odigraju nakon odgovarajućeg uslova. Sva grananja na kraju se završavaju i istoj tački koja predstavlja sjedinjavanje i označava kraj grananja. Nakon sjedinjavanja sve paralelne aktivnosti se kombinuju i spajaju pre završnog stanja.

Dijagram aktivnosti ima izvesne sličnosti sa dijagramom stanja, ali stanja kod dijagrama aktivnosti se menjaju automatski tj. čim se završi jedno stanje prelazi se na drugo (nije nephodan događaj kao sto je to slučaj kod dijagrama stanja). Stanja kod dijagrama aktivnosti zovu se stanja aktivnosti, ili još jednostavnije, aktivnosti. Aktivnosti mogu da se podele na podaktivnosti. Podaktivnosti se dalje mogu deliti na akcije, koje se ne mogu dalje deliti. Akcije se prikazuju istim simbolom kao i aktivnosti (pravougaonik sa zaobljenim ivicama). Aktivnosti i akcije su povezane putem tranzicija, gde tranzicije na dijagramu aktivnosti čine kontrolni tok.

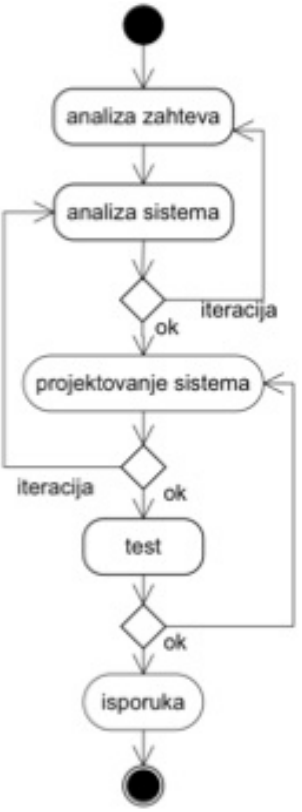


Slika 4.6.1 Objašnjenje formiranja jednog dijagrama aktivnosti

PRIMENA INTERACIJA I PETLJI

Iteracije i petlje se u dijagramima aktivnosti vezuju za aktivnosti odlučivanja.

Na slici 2 pokazan je primer u kome je simbol odluke iskorišćen za prikaz iteracija.

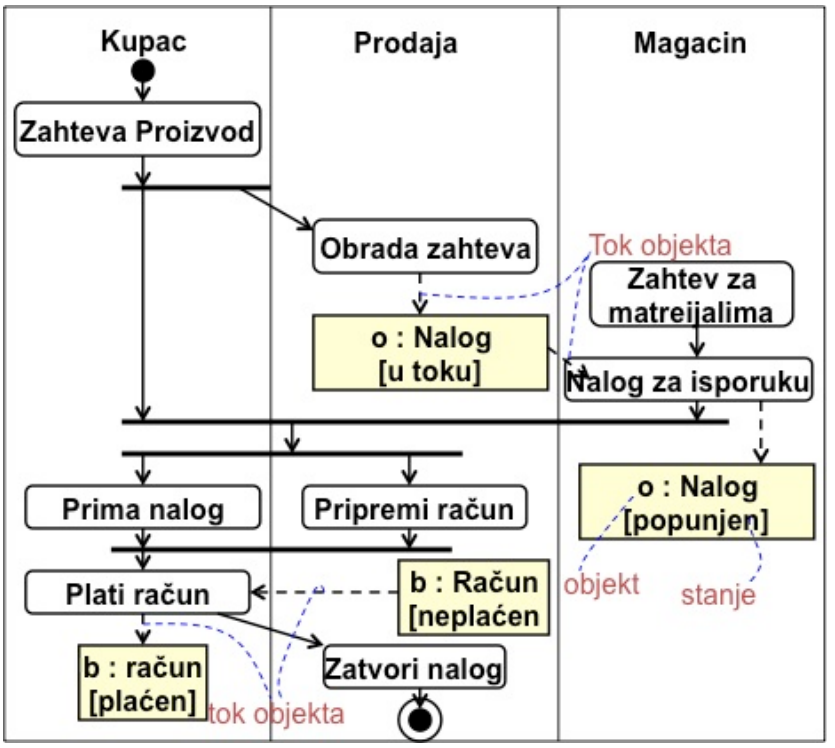


Slika 4.6.2 Primer dijagrama aktivnosti sa petljama i iteracijama

PRIMERI DIJAGRAMA AKTIVNOSTI

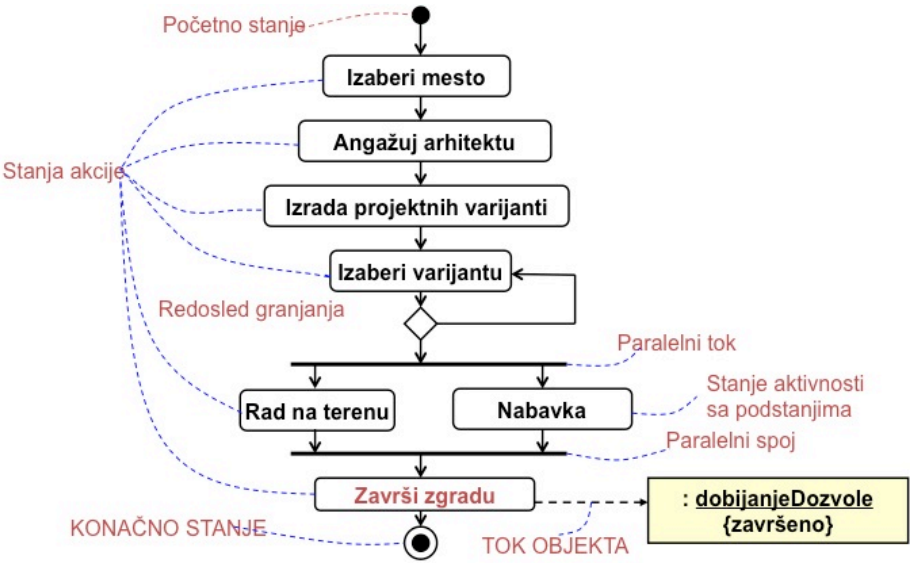
Pri izradi dijagrama aktivnosti, mogu se prikazati i trake (polja) koje ukazuju na objekat (pojedinac ili organizaciona jedinica) koji vrši aktivnosti naznačene unutar njegove trake.

Na slici 3 je prikazan tok aktivnost ktivnosti procesa dobijanja upotrebne dozvole za izgradnju zgradu. On obuhvata aktivnosti od izbora lokacije i arhitekte, pa do pregleda završene zgrade.



Slika 4.6.3 Dijagram aktivnosti procesa izgeadnje zgrade i dobijanje njene upotrebne dozvole

Opciono, pri izradi dijagrama aktivnosti, mogu se prikazati i trake (polja); trake ukazuju na objekat koji vrši aktivnosti naznačene unutar njegove trake (engl. swimlines). Njihovo korišćenje se ilustruje na primeru dijagrama aktivnosti koji se odnosi naproceduru obrade narudžbenice proizvoda (zahteva kupca), koji je prikazan na slici 2 . Debele horizontalne linije predstavljaju tačke sinhronizacije, tj. tačke u kojima se vrši vremensko usklađivanje kraja ili početka dve ili više aktivnosti (u zavisnosti od toga da li se prikupljaju ili račvaju linije tranzicije



Slika 4.6.4 Dijagram aktivnosti koji pokazuje i nosioce aktivnosti

ZADATAK ZA KREIRANJE MODELA PONAŠANJA

Po izboru studenta, za određeni primer, kreiraju se UML dijarami, kao pogledi na jedinstven OO model pod kontrolom Power Designer alata.

Sami izeberite primer iz realnog života koji vam je poznat, i pokrenite Power Designer alat i kreirajte objektno-orijentisani model u Power Designer okruženju:
– Generišite po jedan dijagram sledećih tipova: Dijagram slučajeva korišćenja (Use Case), dijagram saradnje (Collaboration), SD dijagram sekvenci (Sequence), dijagram stanja (StateChart), i dijagram aktivnosti (StateActivity),
– Na svakom od dijagrama postavite elemente koji su Vam na raspolaganju i proučite osobine svakog od njih,

- Sve elemente koji su na raspolaganju,
- Sve relacije koje su na raspolaganju.

Zaključak

REZIME LEKCIJE

Glavne pouke ove lekcije

1. Model je apstrakti pogled sistem koji ignoriše neke detalje sistem. Mogu se razviti komplementarni modeli sistema da bi se prikazao kontekst sistem, interakcije, struktura i ponašanje.
2. Kontekstni modeli pokazuju kako se sistem, koji se modeluje, uklapa u okruženje sa drugim sistemima i procesima. Oni pomažu da se definišu granice sistema koji se razvija.
3. Dijagrama slučajeva korišćenja i sekvencijalni dijagrama se koriste radi opisa interakcija između korisnika sistema i drugih korisnika i sistema. Slučajevi korišćenja opisuju interakcije između sistema i spoljnih aktera. Sekvencijalni dijagrami daju više informacija, jer pokazuju interakcije između objekata sistema, redosled interakcija zajedno sa drugim detaljima.
4. Strukturni modeli pokazuju organizaciju i arhitekturu sistema. Dijagrami klasa se koriste za definisanje statičke strukture klasa u sistemu i njihove interakcije.
5. Modeli ponašanja se koriste radi opisa dinamičkog ponašanja izvršnog sistema. To se može modelovati iz perspektive obrade podataka od strane sistema, ili od strane događaja koji podstiču reakcije sistema.
6. Dijagrami aktivnosti se koriste radi modelovanja obrade podataka, gde svaka aktivnost predstavlja jedan korak u obradi.
7. Dijagrami stanja se koriste radi modelovanja ponašanja sistema pri odgovaranju na unutrađnje i spoljne događaje.
8. Modelima vođeno inženjerstvo je pristup razvoju softvera u kome je sistem predstavljen kao skup modela koji se automatski transformišu u izvršni kod.