

Oblikovanje programske podpore

Arhitektura programske podpore,
stilovi i modeli

Oblikovanje programske potpore

Generičke aktivnosti u procesu programskog inženjerstva:

- **Specifikacija (temeljem analize zahtjeva)**
- **Razvoj i oblikovanje – uključuje izbor i dokumentiranje arhitekture programske potpore**
- **Validacija i verifikacija**
- **Evolucija**

Teme ove prezentacije

- Uvesti **pojmove arhitekture** programske potpore.
- Objasniti proces **donošenja odluka** u izboru i oblikovanju arhitekture programske potpore.
- Definirati kriterije za **izbor** arhitekture.
- Definirati strukturu i sadržaj **dokumenata oblikovanja**.



R&D laboratorij

Carnegie Mellon University, Pittsburgh, PA, USA

Utemeljen 1984

zapošljava >300 ljudi (Pittsburgh, Pennsylvania, Arlington, Virginia, i Frankfurt)

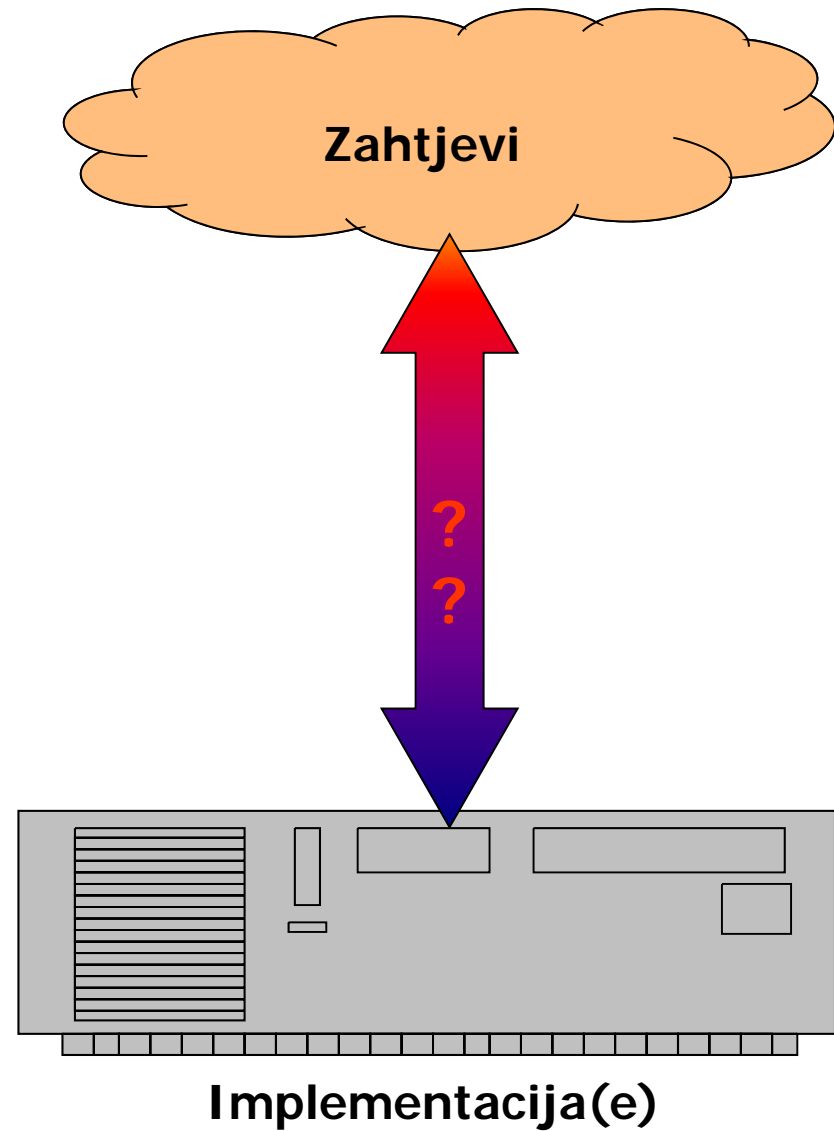
1. Mary Shaw and Paul Clements: *The golden age of software architecture*, IEEE Software, vol 23, no 2, March/April 2006.
2. Timothy C. Lethbridge and Robert Laganière, *Object-Oriented Software Engineering: Practical Software Development using UML and Java*, Second Edition, McGraw Hill, 2001
3. Ian Sommerville, *Software Engineering*, 8th, ed.

Teme ove prezentacije

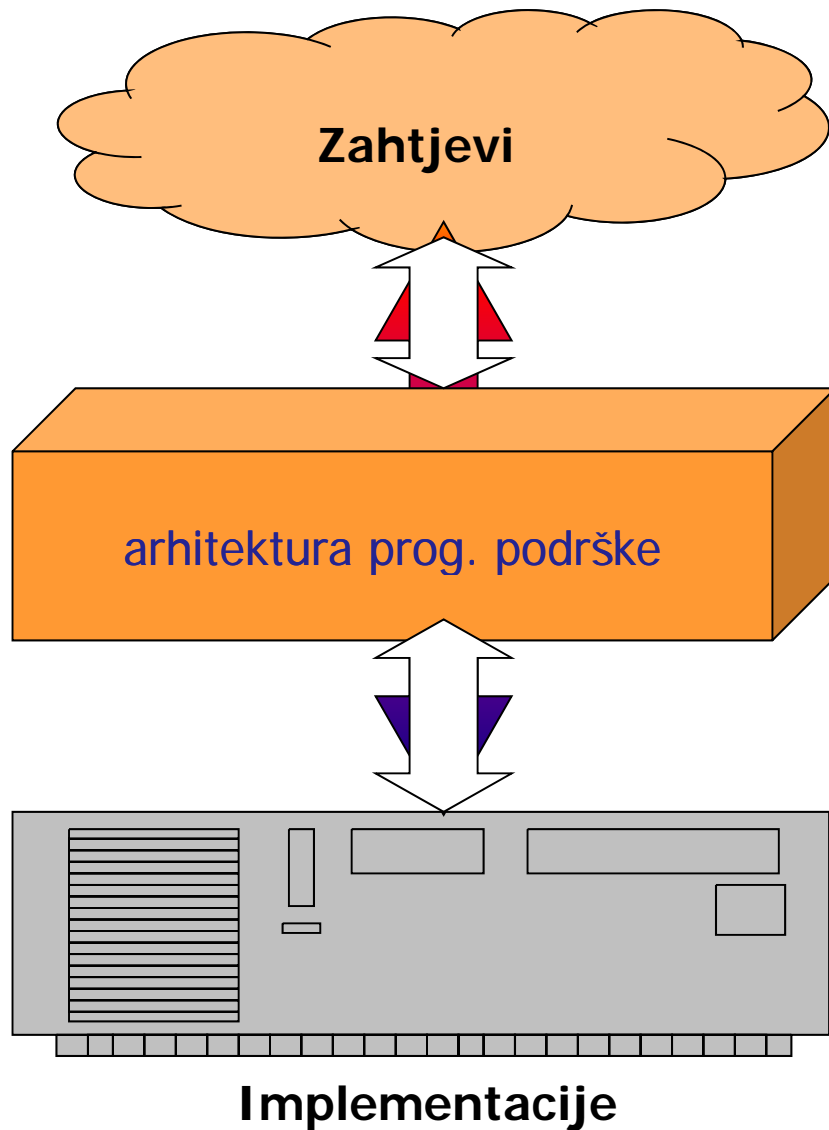
- **Uvesti pojmove arhitekture programske potpore.**
- **Objasniti proces donošenja odluka u izboru i oblikovanju arhitekture programske potpore.**
- **Definirati kriterije za izbor arhitekture.**
- **Definirati strukturu i sadržaj dokumenata oblikovanja.**

Od zahtjeva do koda

- Veliki raskorak između problema i rješenja



Uloga arhitekture prog. podrške



- Daje apstrakciju sustava na visokoj razini.
 - Komponente, konektori, ..
- Dio strukturiranog procesa programskog inženjerstva.
- Osnovni je nositelj kvalitete sustava.
- Kapitalna investicija koja se može ponovno koristiti.
- Osnovica za komunikaciju između dionika.
- Olakšava implementaciju.

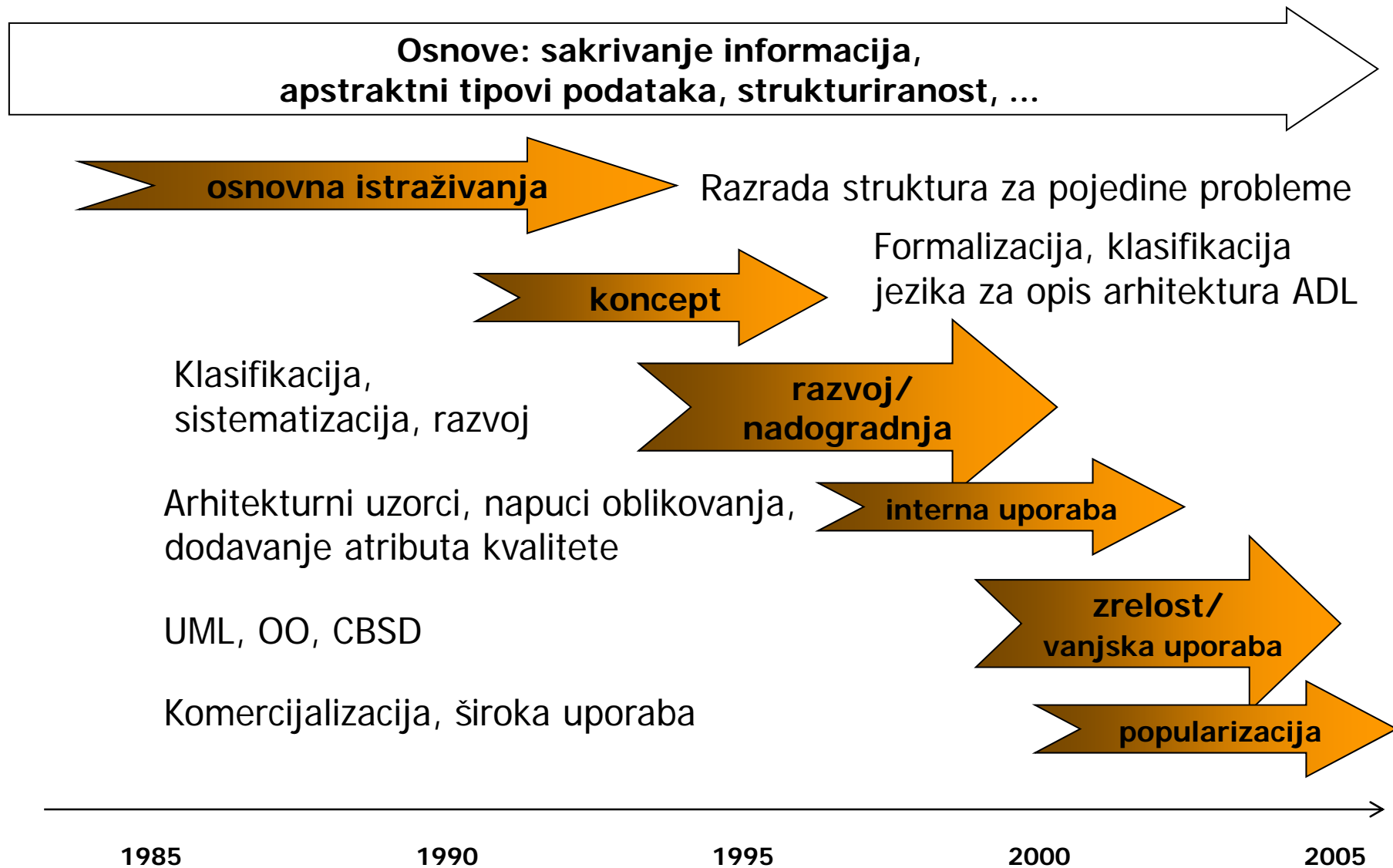
Od programiranja do arhitekture

- 1950 – programiranje na bilo koji način
- 1960 – potprogrami i posebno prevođenje dijelova (engl. **programming in the small**).
- 1970 – apstraktni tipovi podataka, objekti, skrivanje informacije (engl. **programming in the large**).
- 1980 – razvojne okoline, cjevovodi i filtri
- 1990 – objektno usmjereni obrasci (engl. **patterns**), integrirana razvojna okruženja.
- 2000 – arhitektura programske potpore, jezici za oblikovanje (npr. **UML**) i metode oblikovanja (npr. modelno oblikovanje arhitekture – engl. **model driven architecture MDA**).

Razvoj arhitekture prog. potpore (1/2)

- 1968 - **Conway's zakon** Melvin Conway,
 - It concerns the structure of organizations and the corresponding structure of systems (particularly computer software) designed by those organizations.
- krajem 1980, istraživanja u području arhitekture programske podrške s ciljem **analize velikih programskih sustava**.
- Od početaka temeljenih na kvalitativnim opisima empirički promatranih organizacija sustava razvija se i obuhvaća formalne opise, alate i tehnike analize.
 - Od interpretacije **prakse** nastali su konkretni **napuci** za analizu, oblikovanje i razvoj složene programske potpore.
 - Danas predstavlja osnovni element oblikovanja i izrade programskih sustava.

Razvoj arhitekture prog. podpore (2/2)



Definicija oblikovanja arhitekture progr. potpore

- To je **proces identificiranja i strukturiranja podsustava** koji čine cjelinu te okruženja za upravljanje i komunikaciju između tih podsustava.
- Rezultat procesa oblikovanja je opis (**dokumentacija**) arhitekture programske potpore.

Prednosti definiranja arhitekture

- Smanjuje **cijenu** oblikovanja, razvoja i održavanja programskog produkta.
- Omogućuje **ponovnu uporabu** rješenja (engl. **re-use**).
- Poboljšava **razumljivost**.
- Poboljšava **kvalitetu** produkta.
- **Razjašnjava** zahtjeve.
- Omogućuje **donošenje temeljnih inženjerskih odluka**.
- Omogućuje **ranu analizu i uočavanje pogrešaka u oblikovanju**.

Od programera prema arhitektu


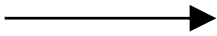
- *Just as good **programmers** recognized useful **data structures** in the late 1960s, good **software system designers** now recognize useful **system organizations**.*
 - Garlan & Shaw: An Introduction to Software Architecture
- Razumjeti potrebe poslovnog modela i zahtjeve projekta.
- Biti svjestan različitih tehničkih pristupa u rješavanju danog problema.
- Evaluirati dobre i loše strane tih pristupa.
- Preslikati potrebe i evaluirane zahtjeve u tehnički opis arhitekture programske potpore.
- Voditi razvojni tim u oblikovanju i implementaciji.
- Koristiti “meke” vještine kao i tehničke vještine.

Soft skills (meke vještine) su osnovne vještine koje ljudi trebaju za uspjeh u poslu. To su one vještine koje su nemjerljive, a predstavljaju osobne vještine, ne-tehničke naravi, koje u mnogočemu zapravo određuju vašu osobnost i vaše sposobnosti rješavanja zahtjevnih problema i zadataka na poslu. One vas karakteriziraju kao lidera, pregovarača, komunikatora, timskog igrača, motivatora... Za razliku od tih vještina tzv. "**hard skills**" su one koje se stječu obrazovanjem, a rezultiraju titulom koju dobivate jednom kada je u vašoj ruci diploma ili završna svjedožba.

Definicija arhitekture

- “as *the size and complexity* of software systems increases, the design problem goes beyond the algorithms and data structures of the computation: *designing and specifying the overall system structure* emerges as a new kind of problem. This is the software architecture level of design.”
 - Garlan, 1992
- “The software architecture of a program or computing system is *the structure or structures of the system*, which comprise software components the externally visible *properties of those components*, and the *relationships* among them.”
 - Bass, Clements, and Kazman. *Software Architecture in Practice*, Addison-Wesley 1997
- *The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.*
 - The IEEE Standard for Architectural Description of Software-Intensive Systems (IEEE P1471/D5.3)

Definicija arhitekture

- **Arhitektura programske potpore** je struktura ili strukture sustava koji sadrži elemente, njihova izvana vidljiva obilježja i odnose između njih.
- **Opis arhitekture** programske potpore je skup dokumentiranih **pogleda** raznih dionika.
- **Pogled** predstavlja djelomično obilježje razmatrane arhitekture programa i dokumentiran je **dijagramom** (nacrtom) koji sadrži:
 - **Elemente:** dijelovi sustava 
 - **Odnose između elemenata :** topologija 
 - **Vanjska vidljiva obilježja**
 - Veličina, performanse, sigurnost, API (sučelje)

Definicija arhitekture

- Više pogleda u okviru nekog konteksta predstavljaju model arhitekture programske potpore, npr.:
 - **Statičan strukturni model** - moduli
 - pokazuju kompoziciju/dekompoziciju sustava
 - **Dinamički procesni model**
 - komponente u izvođenju (engl. runtime)
 - **Alocirane elemente** (npr. datoteke).
 - ...

Arhitektura programske potpore opisuje se modelima koji svaki sadrži jedan ili više pogleda (dijagrama).

Primjer pogleda: komponente i konektori

- Dekompozicija sustava u komponente.
 - **Komponente:**
 - Osnovna jedinica izračunavanja i pohrane podataka (npr. klijent-poslužitelj).
- Tipično hijerarhijska dekompozicija.
 - **Konektori:**
 - Apstrakcija interakcije između komponenata (npr. poziv procedure).
 - **Uporaba stila arhitekture:**
 - Oblikovanje kompozicije komponenata i konektora.
- Respektirati ograničenja i invarijante.

Klasifikacija arhitekture po dosegu

- **Koncepcijska** - engl. *Conceptual Architecture*
 - Usmjeravanje pažnje na pogodnu dekompoziciju sustava.
 - Komunikacija s netehničkim dionicima (uprava, prodaja, korisnici).
 - Neformalna specifikacija komponenti (npr. CRC-R cards)

- **Logička** - engl. *Logical Architecture*
 - Precizno dopunjena.
 - Detaljan nacrt pogodan za razvoj komponenti.
 - Architecture Diagram (with interfaces), Component and Interface Specifications, Component Collaboration Diagrams, potrebna objašnjenja diskusije

- **Izvršna** - engl. *Execution Architecture*
 - Namijenjena distribuiranim i paralelnim sustavima.
 - Pridruživanje procesa fizičkom sustavu.

Stilovi (familije) arhitektura programske potpore

- **Skupovi srodnih arhitektura.** U jednom programskom produktu može postojati kombinacija više stilova.
- **Opisuju se:**
 - Rječnikom (tipovima komponenata i konektora).
 - Topološkim ograničenjima koja moraju zadovoljiti svi članovi familije (stila).
- **Primjeri stilova:**
 - protok podataka (engl. data-flow)
 - objektno usmjeren stil
 - repozitorij podataka
 - upravljan događajima
 - ...

Teme ove prezentacije

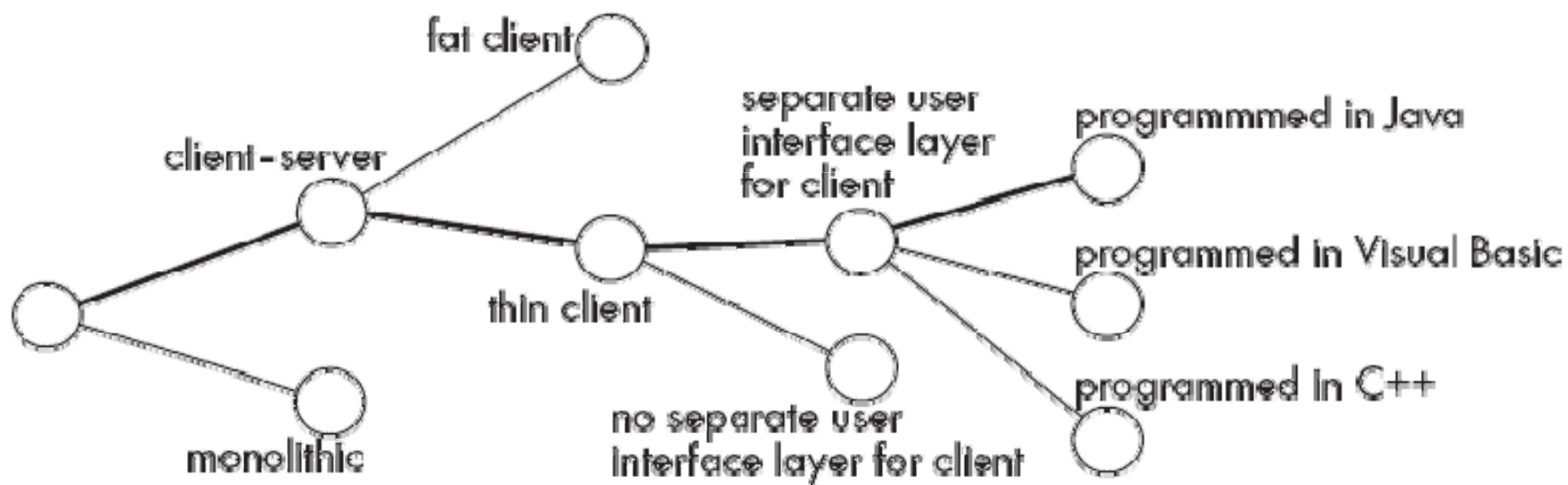
- Uvesti pojmove arhitekture programske potpore.
- Objasniti proces donošenja odluka u izboru i oblikovanju arhitekture programske potpore.
- Definirati kriterije za izbor arhitekture.
- Definirati strukturu i sadržaj dokumenata oblikovanja.

Proces izbora i evaluacije arhitekture = proces donošenja odluka

- Alternativni stilovi arhitekture programske potpore
- Oblikovanje kao niz odluka.
- Dizajner se sučeljava s rješavanjem niza problema (engl. *design issues*) - to su podproblemi ukupnog problema.
- Postoji nekoliko inačica rješenja (engl. *design options*)
- Dizajner donosi odluke (engl. *design decision*) za rješavanje problema.
- Odabir najbolje opcije između više mogućnosti rješenja problema.

Prostor oblikovanja

- To je skup različnih oblikovanja koja su na raspolaganju uporabom različitih izbora, engl. *design space*

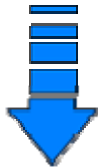


Donošenje odluka

- Za donošenje odluka potrebna znanja:
 - Zahtjevi (specifikacija)
 - Trenutno oblikovana arhitektura
 - Raspoloživa tehnologija
 - Principi oblikovanja i najbolja praksa (engl. *best practices*)
 - Dobra rješenja iz prošlosti
- Zadaće donošenja odluka:
 - Postavljanje prioriteta sustava
 - Dekompozicija sustava
 - Definiranje svojstva sustava
 - Postavljanje sustava u kontekst
 - Integritet sustava

Primjer okvira donošenja odluka

*guide
architects*



Meta-Architecture

- Architectural vision, principles, styles, key concepts and mechanisms
- *Focus: high-level decisions that will strongly influence the structure of the system; rules certain structural choices out, and guides selection decisions and tradeoffs among others*

Architecture

- Structures and relationships, static and dynamic views, assumptions and rationale
- *Focus: decomposition and allocation of responsibility, interface design, assignment to processes and threads*

Conceptual Architecture

Logical Architecture

Execution Architecture

*guide
designers*



Architecture Guidelines and Policies

- Use model and guidelines; policies, mechanisms and design patterns; frameworks, infrastructure and standards
- *Focus: guide engineers in creating designs that maintain the integrity of the architecture*

Oblikovanje od vrha prema dolje

- **Engl. Top-down design**
- **Oblikuj najvišu strukturu sustava**
- **Postepeno razrađuj detalje**
- **Na kraju detaljne odluke kao npr.:**
 - **Format podataka;**
 - **Uporaba/izbor algoritama**

Oblikovanje od dna prema vrhu

- **Bottom-up design**
- **Donošenje odluka o komponentama za ponovnu uporabu**
- **Odluke o njihovoj uporabi za stvaranje (kompoziciju) komponenti više razine**
- **Hibridno oblikovanje**
 - **Uporaba obje metode:**
 - **Top-down design**
 - Daje **dobru struktura** sustava
 - **Bottom-up design**
 - Stvara komponente pogodne za **ponovnu uporabu**

Dosezi (aspekti) oblikovanja

- *Oblikovanje arhitekture*
 - Podjela u podsustave i komponente
 - Način povezivanja?
 - Način međudjelovanja?
 - Sučelja?

- *Oblikovanje korisničkog sučelja*
- *Oblikovanje algoritma*
 - Za izračunavanje, upravljanje ..
- *Oblikovanje protokola*
 - Komunikacijski protokoli

ne u ovom
kolegiju

Postupak razvoja modela arhitekture

- **Započni s grubom skicom** arhitekture zasnovanoj na osnovnim zahtjevima i obrascima uporabe.
- **Odredi temeljne** potrebne **komponente sustava**.
- **Izaberi između raznih stilova** arhitekture
 - **Savjet:** nekoliko timova nezavisno radi grubu skicu arhitekture, a potom se spoje najbolje ideje.
- **Arhitekturu dopuni detaljima tako da se:**
 - **identificiraju osnovni načini komunikacije i interakcije između komponenata**
 - **odredi kako će se dijelovi podataka i funkcionalnosti raspodijeliti između komponenata**
 - **pokušaj identificirati dijelove za ponovnu uporabu (engl. reuse)**
 - **vрати se na pojedini obrazac uporabe i podesi arhitekturu.**

Tehnika donošena dobrih odluka oblikovanja

- **Uporaba prioriteta i ciljeva za odabir alternativa**
- **Pobroji i opiši opcije (alternative) odluka oblikovanja.**
- **Pobroji prednosti i nedostatke svake opcije u odnosu na prioritete i ciljeve.**
- **Odredi opcije koje su u sukobu s ciljevima.**
- **Odaberi opciju koja najbolje zadovoljavaju ciljeve.**
- **Prilagodi prioritete za daljnje donošenje odluka.**

Primjer prioriteta i ciljeva u području oblikovanja računalnog sustava

- **Sigurnost/Security:** Podaci se ne smiju moći dešifrirati poznatim tehnikama za < 100sati na 400Mhz Intel procesoru.
- **Održavanje/Maintainability:** Nema
- **Performanse/CPU efficiency:** Odziv < 1s na 400MHz Intel procesoru.
- **Mrežno opterećenje/Network bandwidth efficiency:** $\leq 8\text{KB}$ po transakciji.
- **Memorijski resursi/Memory efficiency:** $\leq 20\text{MB RAM}$.
- **Prenosivost/Portability:** Windows XP, Linux

Nešto analogno navedenim prioritetima i ciljevima trebalo bi definirati za postupak izbora najpogodnije arhitekture programske potpore. Vidi principe oblikovanja i kriterije.

Analiza opcija (alternativa) A, B, C, D, E

	<i>Security</i>	<i>Maintainability</i>	<i>Memory efficiency</i>	<i>CPU efficiency</i>	<i>Bandwidth efficiency</i>	<i>Portability</i>
Algorithm A	High	Medium	High	Medium; DNMO	Low	Low
Algorithm B	High	High	Low	Medium; DNMO	Medium	Low
Algorithm C	High	High	High	Low; DNMO	High	Low
Algorithm D	—	—	—	Medium; DNMO	DNMO	—
Algorithm E	DNMO	—	—	Low; DNMO	—	—

DNMO = Does Not Meet the Objective

Uporaba “cost-benefit” analize za odabir opcija

- Za procjenu CIJENE/troškova sumiraj:
 - Cijenu rada programskog inženjera, uključujući održavanje.
 - Cijenu uporabe razvojne tehnologije.
 - Cijenu koja tereti krajnje korisnike (uključivo i evoluciju produkta).

- Za procjenu KORISTI/dobiti sumiraj:
 - Uštedu vremena programskog inženjera.
 - Dobrobiti mjerene kroz povećanu prodaju ili ostale financijske uštede.

Teme ove prezentacije

- Uvesti pojmove arhitekture programske potpore.
- Objasniti proces donošenja odluka u izboru i oblikovanju arhitekture programske potpore.
- Definirati kriterije za izbor arhitekture.
- Definirati strukturu i sadržaj dokumenata oblikovanja.

Principi oblikovanja i kriteriji izbora arhitekture

- | | |
|---------------------------------------|---|
| 1. Podijeli i vladaj | - Divide and conquer |
| 2. Povećaj koheziju | - Increase cohesion |
| 3. Smanji međuovisnost | - Reduce coupling |
| 4. Zadrži što višu razinu apstrakcije | - Keep the level of abstraction as high as possible |
| 5. Povećaj ponovnu uporabivost | - Increase reusability |
| 6. Povećaj uporabu postojećeg | - Reuse existing designs and code |
| 7. Oblikuj za fleksibilnost | - Design for flexibility |
| 8. Planiraj zastaru | - Anticipate obsolescence |
| 9. Oblikuj za prenosivost | - Design for portability |
| 10. Oblikuj za ispitivanje | - Design for testability |
| 11. Oblikuj konzervativno | - Design defensively |
| 12. Oblikuj po ugovoru | - Design by contract |

1: Podijeli i vladaj

engl. **Divide and conquer**

- **Jednostavniji je rad s više malih dijelova.**
- **Odvojeni timovi rade na manjim problemima.**
- **Omogućava specijalizaciju pojedinca i tima..**
- **Manje komponente - povećana razumljivost.**
- **Olakšana zamjena dijelova (bez opsežne intervencije u cijeli sustav).**

Primjeri podjele programskih sustava

- **Distribuirani sustavi – klijenti i poslužitelji**
- **Podjela sustava u podsustave**
- **Podjela podsustava u pakete**
- **Podjela paketa u klase (kod npr. objektno usmjerene arhitekture)**

2: Povećanje kohezije

engl. **Increase cohesion**

- **Podsustav ili modul ima veliku koheziju ako grupira međusobno povezane elemente, a sve ostalo stavlja izvan grupe.**
- **Olakšava razumijevanje i promjene u sustavu.**
- **Klasifikacija**
 - **Funkcijska – Functional;**
 - **Razinska – Layer;**
 - **Komunikacijska – Communicational;**
 - **Sekvencijska – Sequential;**
 - **Proceduralna – Procedural;**
 - **Vremenska – Temporal;**
 - **Korisnička – Utility**

Funkcijska kohezija

engl. **Functional cohesion**

- Sav kod koji obavlja pojedinu operaciju je grupiran, sve ostalo izvan.
- To se npr. događa kad modul obavlja jednu operaciju, te vraća rezultat bez popratnih efekata.
 - prednosti:
 - Olakšano razumijevanje
 - Povećana ponovna uporabljivost modula
 - Lakša zamjena
 - Suprotan je princip nefunkcijske kohezije:
 - Modul mijenja bazu podatka, kreira datoteku, interakcija s korisnikom

Razinska kohezija (kohezija u istoj razini)

engl. **Layer cohesion**

- Svi resursi za pristup skupu povezanih usluga (servisa) su na jednom mjestu, sve ostalo izvan.
 - Razine formiraju hijerarhiju
 - Viša razina može pristupiti servisima niže razine
 - Niža razina ne pristupa višoj
 - Skup procedura kojima pojedina razina omogućava pristup servisima naziva se aplikacijsko (primjensko) programsko sučelje **application programming interface (API)**.
 - Moguća zamjena pojedine razine bez utjecaja na ostale (više ili niže) razine.

Komunikacijska kohezija

engl. **Communicational cohesion**

- Svi moduli koji pristupaju ili mijenjaju određene podatke su grupirani, sve ostalo izvan.
- Klasa (u objektnom pristupu) ima dobru komunikacijsku koheziju ako:
 - Sadrži sve sistemske usluge neophodne za rad s podacima.
 - Ne obavlja ništa drugo osim upravljanja podacima.
- Prednost:
 - Prilikom promjene podatka sav kod na jednom mjestu.

Sekvencijska kohezija

engl. **Sequential cohesion**

- **Postiže se grupiranjem procedura u kojoj jedna daje ulaz slijedećoj, sve ostale izvan.**
 - **Na postizanje sekvencijske kohezije fokusiramo se nakon svih prethodnih kohezijskih tipova.**

Proceduralna kohezija

engl. **Procedural cohesion**

- **Procedure koje se upotrebljavaju jedna nakon druge (u nizu).**
 - **Ne moraju razmjenjivati informacije kao kod sekvencijske kohezije.**
 - **Slabija od sekvencijske**

Vremenska kohezija

engl. **Temporal Cohesion**

- Operacije koje se obavljaju tijekom iste faze rada programa su grupirane, sve ostalo izvan.
 - Npr. Podizanje sustava ili inicijalizacija
 - Slabija od proceduralne kohezije

Kohezija pomoćnih programa

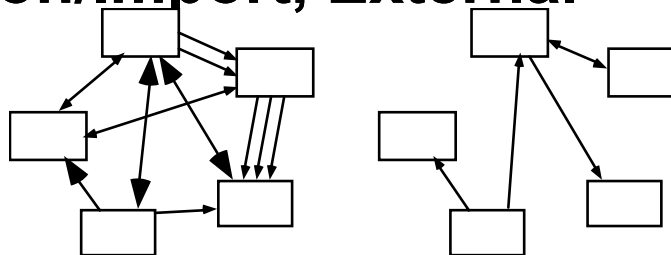
engl. **Utility cohesion**

- **Povezani pomoćni programi (engl. *utilities*) koji se logički ne mogu smjestiti u ostale kohezijske grupe.**
 - **Pomoćni program je procedura ili klasa koja je široko primjenjiva za različite implementirane sustave.**
 - **Npr. `java.lang.Math`**

3: Smanjivanje međuovisnosti

engl. **Reduce coupling where possible**

- *Povezivanje se javlja u slučaju međuovisnosti modula*
- **Međuovisnost \Rightarrow promjene na jednom mjestu zahtijevaju i promjene drugdje**
- **Kod velike međuovisnosti teško je jasno raspoznati rad komponente.**
- **Tipovi međuovisnosti:**
 - **Content, Common, Control, Stamp, Data, Routine Call, Type use, Inclusion/Import, External**



Međuovisnost sadržaja

engl. **Content coupling**

- Jedna komponenta prikriveno mijenja interne podatke druge komponente.
- U objektnom oblikovanju da bi se smanjila međuovisnost sadržaja potrebno je:
 - Uporaba enkapsulacije svih varijabli u instancijama
 - Deklarirati ih kao **private**
 - Osigurati **get** i **set** *methode* (za indirektan pristup).
 - Najgori oblik međuovisnosti sadržaja (i teško se detektira) javlja se kod izravne modifikacije varijable od instancirane varijable (višerazinska međuovisnost).

Opća međuovisnost

engl. **Common coupling**

- Javlja se pri uporabi globalne varijable
 - Sve komponente koje rabe globalnu varijablu postaju povezane.
 - Slabiji oblik međuovisnosti nastaje kad instancije jednog podskupa klasa pristupaju varijabli.
 - Npr. Java package
 - Globalne varijable mogu biti prihvatljive za postavljanje tipičnih vrijednosti sustava (engl. *default*).

Upravljačka međuovisnost

engl. **Control coupling**

- Izravna kontrola rada druge procedure uporabom zastavice ili naredbe.
 - Za neku promjenu potrebno mijenjati obje procedure
 - Izbjegavanje:
 - Uporabom polimorfnih operacija u objektnom pristupu. Tijekom rada određuje se koju proceduru treba pozvati i kako se ona izvodi.
 - *look-up tablice*
 - Pridruživanje naredbi odgovarajućoj metodi

Međuovisnost u objektnom oblikovanju

engl. **Stamp coupling in OO design**

- Javlja se kada je jedna klasa deklarirana kao tip argumenta metode (procedure).
- Jedna klasa upotrebljava drugu te na taj način otežava promjene sustava
 - Ponovna uporaba jedne klase zahtijeva i ponovnu uporabu druge.
- Načini smanjenja ove međuovisnosti:
 - Uporaba sučelja kao tipa argumenta procedure.
 - Prijenos jednostavnijih varijabli u argumentima.

Podatkovna međuovisnost u objektnom oblikovanju

engl. **Data coupling in OO design**

- Javlja se kada su tipovi argumenata procedure primitivi ili jednostavne klase iz biblioteke.
 - Međuovisnost je jača s većim brojem argumenata metode
 - Svi koje koriste tu moraju strogo prenijeti sve argumente.
- Smanjenje nepotrebne uporabe argumenata smanjuje stupanj međuovisnosti.
- Potreban kompromis podatkovnog i “stamp” povezivanja u objektnom oblikovanju.
 - Povećanje međuovisnosti jednog tipa smanjuje drugi.

Međuovisnost u povezivanim pozivima procedura

engl. **Routine call coupling**

- **Javlja se kada procedura (ili metoda u objektnom sustavu) poziva drugu**
 - **Procedure time postaju povezane jer ovise o ponašanju druge**
 - **Uvijek prisutno u sustavima**
- **Ako se česta rabi niz dvije ili više procedura/metoda:**
 - **Smanjenje povezivanja postiže se pisanjem jedinstvene procedure koja obuhvaća željeni niz**

Međuovisnost tipova

engl. **Type use coupling**

- Javlja se kada modul koristi podatkovni tip definiran u drugom modulu.
 - U objektnom oblikovanju to je slučaj kada klasa deklarira varijable instancija ili lokalna varijabla ima tip druge klase.
 - Posljedica takve međuovisnosti je potreba za promjenom svih korisnika neke definicije pri njezinoj promjeni.
 - Izbjegavanje: Deklarirati tip varijable kao najopćenitiju klasu ili sučelje koje sadrži zahtijevane operacije

Međuviznost uključivanjem

engl. **Inclusion or import coupling**

- **Javlja se kada komponenta importira paket**
 - **Java**
- **Jedna komponenta uključuje drugu**
 - **C++**
 - **Komponenta koja importira/uključuje drugu komponentu izložena je toj uključenoj komponenti.**
 - **Ako importirana/uključena komponenta nešto promjeni može doći do konflikta s komponentom koja importira/uključuje.**
 - **Ime nekog elementa uključene komponente može biti u konfliktu s imenom definiranim u komponenti koja uključuje.**

Vanjska međuovisnost

engl. **External coupling**

- Javlja se kad modul ovisi o OS, biblioteci, HW, ...
 - Potrebno je minimizirati broj mjesta na kojima se javlja takva povezanost.
 - Ili u objektnom pristupu oblikovati malo sučelje prema vanjskim komponentama (to se naziva *Facade Design Pattern*).

4: Zadrži što višu razinu apstrakcije

engl. Keep the level of abstraction as high as possible

- Osigurati da oblikovanje omogući **sakrivanje** ili odgodu razmatranja **detalja**, te na taj način smanji složenost.
 - Dobra apstrakcija podrazumijeva skrivanje informacija (engl. *information hiding*).
 - Omogućava razumijevanje biti (suštine) podsustava bez nepotrebnih detalja.

Apstrakcije i klase u objektnom oblikovanju

engl. **Abstraction and classes** (OO design)

- **Klase su podatkovne apstrakcije koje sadrže proceduralne apstrakcije.**
 - **Razina apstrakcije se povećava definiranjem privatnih varijabli.**
 - **Poboljšava smanjivanjem broja javnih metoda.**
 - **Superklase i sučelja (Superclass, interface) povećava razinu.**
 - **Atributi i asocijacije predstavljaju podatkovne apstrakcije.**
 - **Metode predstavljaju proceduralne apstrakcije.**

5: Povećaj ponovnu uporabivost

engl. Increase reusability where possible

- Oblikovanje različitih aspekata sustava može pridonijeti njihovoj ponovnoj uporabi.
- Generaliziraj oblikovanje što je više moguće.
- Uporabi prethodne principe oblikovanja **Povećaj koheziju**, **Smanji povezanost**, **Zadrži visoku razinu apstrakcije**
- Oblikuj sustav tako da sadrži kopče / sučelje koje omogućavaju pristup korisničkom kodu (**engl. hooks**). Korisnik vidi kopče kao otvore u kodu koji su dostupni u trenutku pojave nekog događaja ili zadovoljenja nekog uvjeta.
- Maksimalno pojednostavi oblikovanje.

6:Povećaj uporabu postojećeg

engl. **Reuse existing designs and code where possible**

- Princip je komplementaran povećanju ponovne uporabivosti.
 - Što veća aktivna ponovna uporaba komponenti smanjuje trošak i povećava stabilnost sustava.
 - Treba iskoristiti ranije investicije.
 - “kloniranje“, t.j. ponavljanje istih linija koda se ne broji (engl. Clonning).

7: Oblikuj za fleksibilnost

engl. **Design for flexibility**

- Aktivno predviđaj buduće moguće promjene i provedi pripremu za njih tako da:
 - Smanji međuovisnost i povećaj koheziju.
 - Stvaraj apstrakcije.
 - Ne upotrebljavaj izravno umetanje podataka ili konfiguracija u izvorni programski kod (engl. **hard code**)
 - neka se konfiguracija čita iz datoteke ili okoline
 - Ostavi otvorene opcije za kasniju modifikaciju
 - Upotrebljavaj ponovo gotovi kod i radi ga takvog (teži što većoj ponovnoj uporabi i uporabivosti).

8: Planiraj zastaru

engl. *Anticipate obsolescence*

- Planiraj promjene u tehnologiji ili okolini tako da program može i dalje raditi ili biti jednostavno promijenjen,
 - Izbjegavaj uporabu **novih izdanja** (engl. *release*) tehnologija.
 - Izbjegavaj knjižnice namijenjene **specifičnim okolinama**.
 - Izbjegavaj nedokumentirane ili rijetko upotrebljavane dijelove knjižnica.
 - Izbjegavaj SW/HW bez izgleda za **dugotrajniju** podršku.
 - Koristi tehnologiju i jezik podržane od **više dobavljača**.

9: Oblikuj za prenosivost

engl. Design for Portability

- Omogući rad programske potpore na što većem broju **različitih platformi**.
- Izbjegavaj specifičnosti neke okoline.
 - Npr. knjižnice koje postoje samo u Microsoft Windows.

10: Oblikuj za ispitivanje

engl. **Design for Testability**

- **Olakšaj ispitivanje**
 - **Oblikuj program za automatsko testiranje.**
 - **Omogući pokretanje svih funkcija uporabom vanjskih programa (npr. zaobilazeći grafičko sučelje).**
 - **Npr. U Javi, kreiraj main() metodu u svakoj klasi.**

11: Oblikuj konzervativno

engl. Design Defensively

- Ne koristiti pretpostavke **kako** će netko koristiti oblikovanu komponentu.
 - Obradi **sve slučajeve** u kojima se komponenta može neprikladno upotrijebiti.
 - Provjeri valjanost ulaza provjerom definiranih pretpostavki.
 - Pretjerano obrambeno oblikovanje – često dovodi do nepotrebnih provjera.

12. Oblikuj po ugovoru

- engl. **Design by contract**
- To je tehnika koja omogućava efikasan i sistematski pristup konzervativnom oblikovanju.
 - Osnovna ideja
 - Sve metode imaju **ugovor s pozivateljima**.
 - Ugovor ima skup definiranih zahtjeva, t.j.:
 - **Preduvjete** koje pozvana metoda zahtjeva da budu ispunjeni prije početka izvođenja (engl. *preconditions*).
 - **Uvjete** koje pozvana metoda mora osigurati **nakon** završetka izvođenja (engl. *postconditions*).
 - Na koje **invarijante** pozvana metoda neće djelovati pri izvođenju.

Teme ove prezentacije

- Uvesti pojmove arhitekture programske potpore.
- Objasniti proces donošenja odluka u izboru i oblikovanju arhitekture programske potpore.
- Definirati kriterije za izbor arhitekture.
- Definirati strukturu i sadržaj dokumenata oblikovanja.

Dokumentiranje arhitekture

- Potrebno zbog **rane analize sustava**.
- Temeljni nositelj obilježja **kvalitete**.
- Ključ za **održavanje, poboljšanja i izmjene**
- Dokumentacija **ne zastarijeva**
 - govori umjesto arhitekta danas i nakon 20 godina (ukoliko je sustav oblikovan, održavan i mijenjan sukladno dokumentaciji).
- U praksi je **danas** dokumentacija **nejednoznačna** i često kontradiktorna.
 - Najčešće samo pravokutnici i linije koje mogu značiti: A šalje upravljačke signale do B, A šalje podatke do B, A šalje poruku do B, A kreira B, A dobavlja vrijednost od B, ...

Pisanje dobrih dokumenata oblikovanja

- Dokumenti oblikovanja su pomoć izradi boljih dizajna.
 - Traže izričitost i obrađuju najvažnija pitanja prije faze implementacije.
 - Omogućuju **procjenu** oblikovanja i poboljšanje.
 - **Sredstvo komunikacije** prema
 - Timu za implementaciju
 - Budućim osobama zaduženim za unošenje promjena
 - Osobama za oblikovanje povezanih sustava ili podsustava

Struktura dokumenta oblikovanja

- Svrha (koji sustav ili dio sustava ovaj dokument opisuje te označi referencu prema dokumentu zahtjeva na koji se ovaj dokument oslanja - slijedivost).
- Opći prioriteti (opiši prioritete koji su vodili proces oblikovanja).
- Skica sustava (navedi opis sustava s najviše razine promatranja kako bi čitatelj razumio osnovnu ideju).
- Temeljna pitanja u oblikovanju (diskutiraj osnovne probleme koji su se morali razriješiti, navedi razmatrane opsijska rješenja, konačnu odluku i razloge za njeno donošenje).
- Detalji oblikovanja (koji su čitatelju zanimljivi a u dokumentu još nisu razmatrani).

Što nije sadržaj dokumenta oblikovanja

- Izbjegavati dokumentiranje informacija očitih iskusnim programerima i dizajnerima.
- Ne pisati detalje koji su dio komentara koda.
- Ne pisati detalje koji su vidljivi u strukturi koda.

Zaključak: Poželjne značajke arh. prog. potpore

Izbor i dokumentiranje mora rezultirati u arhitekturi programske potpore koja je:

- **Dobra**
 - Tehnički ispravna i jasno prezentirana
- **Ispravna**
 - Doseže potrebe i ciljeve ključnih dionika
- **Uspješna**
 - Upotrebljava se u stvarnom razvoju sustava kojim se postižu strateške prednosti

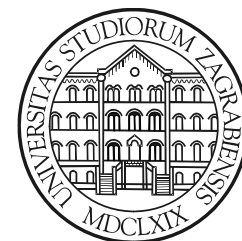
Oblikovanje programske potpore

Modularizacija i objektno usmjerena arhitektura

Prof.dr.sc. Vlado Sruk



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



Tema

- **Podsjetnik**
 - Metode programskog inženjerstva
- **Objektno usmjerena paradigma**
- **Koncepti objektnog usmjerenja**
- **UML Dijagrami razreda i objekata**
- **Kreiranje dijagrama razreda**

Literatura

- **Timothy C. Lethbridge, Robert Laganière: Object-Oriented Software Engineering: Practical Software Development using UML and Java, McGraw Hill, 2001**
 - **<http://www.lloseng.com>**
- **Sommerville, I., Software engineering, 8th ed, Addison Wesley, 2007.**
- **O'Docherty, Mike: Object-oriented analysis and design : understanding system development with UML 2.0 / Mike O'Docherty, John Wiley & Sons Ltd, 2005**
- **Šribar, J.; Motik, B.: Demistificirani C++, Element, 2001 ("Dobro upoznajte protivnika da biste njime ovladali")**

■ Programsko inženjerstvo

- sustavan i organiziran pristup procesu izrade;
- upotrebljavati prikladne alate i tehnike ovisno o problemu koji treba riješiti, ograničenjima u procesu izrade i postojećim resursima.

Metode programskog inženjerstva

- strukturni pristup razvoju i oblikovanju programske potpore
 - modele sustava;
 - notaciju (označavanje);
 - pravila;
 - preporuke i napuci.
- Opisi modela
 - najčešće grafički
- Pravila
 - ograničenja primijenjena na modele sustava
- Preporuke
 - “dobra inženjerska praksa”
- Naputke o procesu
 - slijed aktivnosti

Inženjerstvo zahtjeva

- To je postupak pronalaženja, analiziranja, dokumentiranja i provjere zahtijevanih usluga sustava, te ograničenja u uporabi.
- Zahtjevi sami za sebe su opisi usluga sustava i ograničenja koja se generiraju tijekom procesa inženjerstva zahtjeva.
- Obzirom na razinu detalja razlikujemo:
 - Specifikacija visoke razine apstrakcije
 - obično u okviru ponude za izradu programskog produkta = korisnički zahtjevi. Pišu se u prirodnom jeziku i grafičkim dijagramima. Moraju biti razumljivi netehničkom osoblju.
 - Vrlo detaljna specifikacija
 - uobičajeno nakon prihvaćanja ponude, a prije sklapanja ugovora = zahtjev sustava.
 - Pišu se strukturiranim prirodnim jezikom, posebnim jezicima za oblikovanje sustava, dijagramima i matematičkom notacijom.
 - Specifikacija programske potpore
 - najdetaljniji opis i objedinjuje korisničke i zahtjeve sustava.

Procesi inženjerstva zahtjeva

- Procesi koji su u upotrebi u inženjerstvu zahtjeva razlikuju se ovisno o domeni primjene, ljudskim resursima i organizaciji koja oblikuje zahtjeve.
- Postoje neke generičke aktivnosti zajedničke svim procesima:
 - Izlučivanje zahtjeva (engl. requirements elicitation)
 - Analiza zahtjeva
 - Validacija zahtjeva
 - Upravljanje zahtjevima

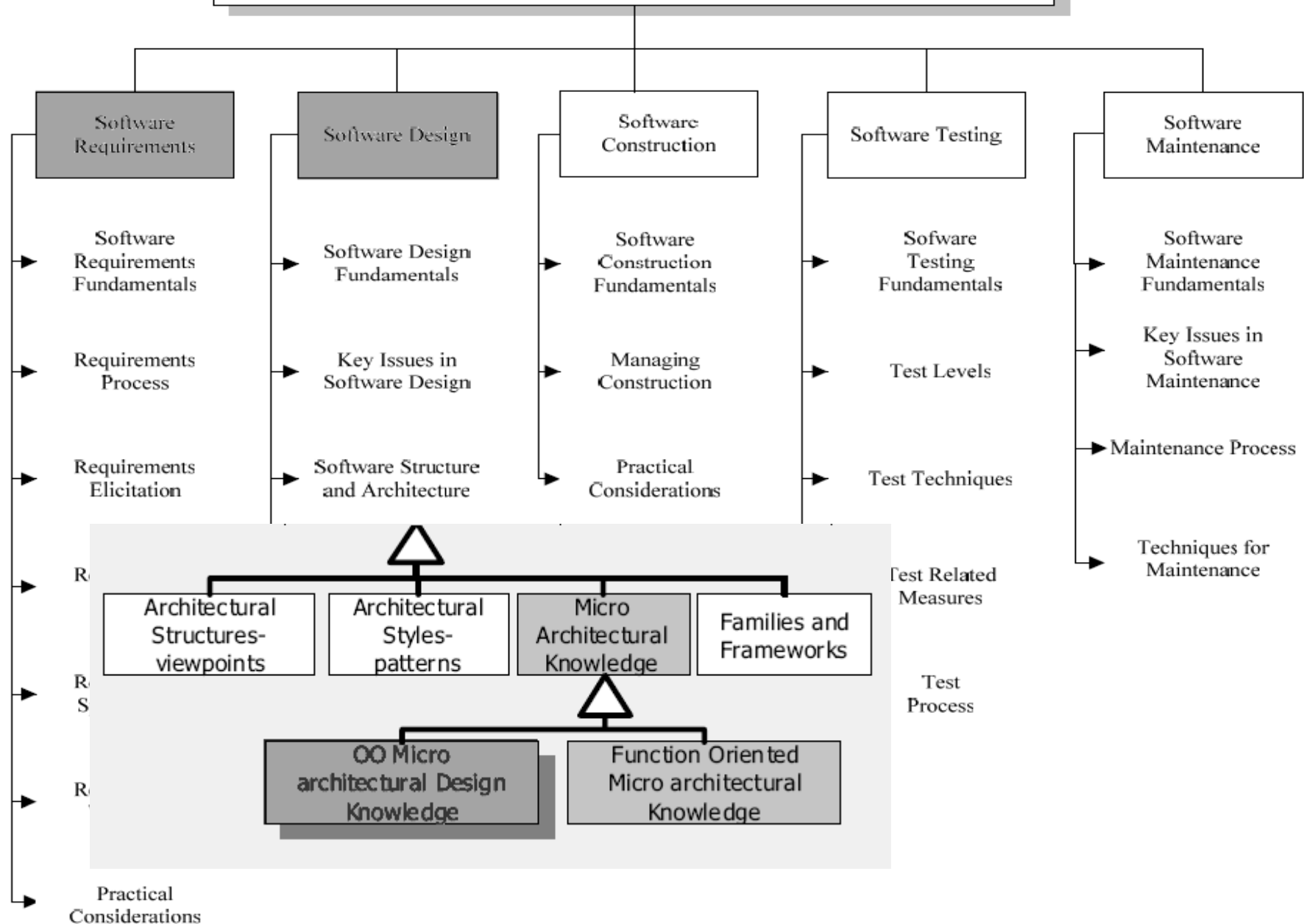
Arhitektura programske podrške

- Inženjerstvo zahtjeva
 - Što graditi?
- Oblikovanje
 - Kako graditi?
- Podjela u dvije faze
 - Oblikovanje arhitekture
 - engl. High-level design
 - Implementacija
 - engl. Low-level design

Područje programskog inženjerstva

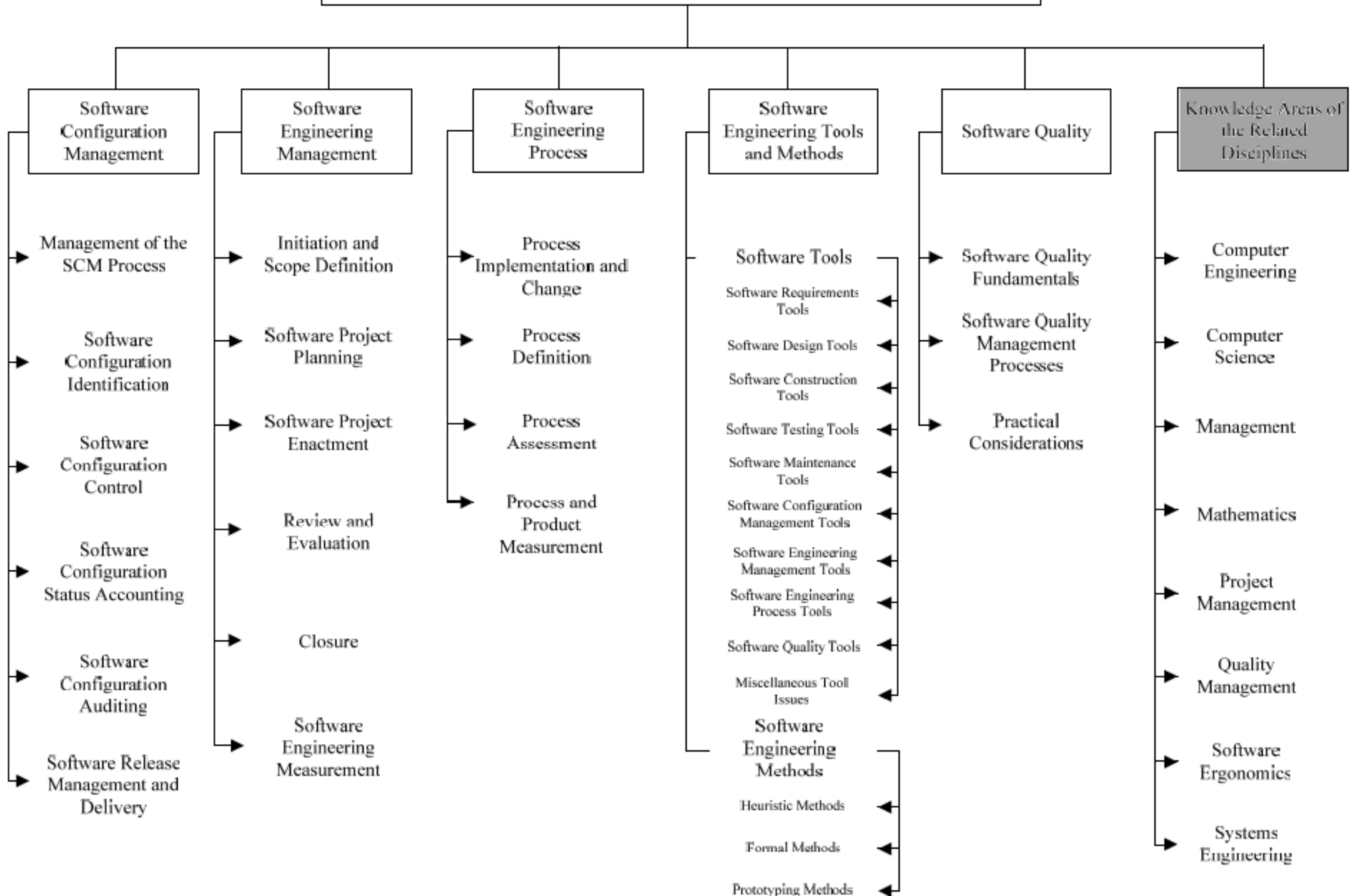
- **Guide to the Software Engineering Body of Knowledge**
- **<http://www.computer.org/portal/web/swebok>
2004 Version
SWEBOK®
A project of the IEEE Computer Society Professional Practices Committee**

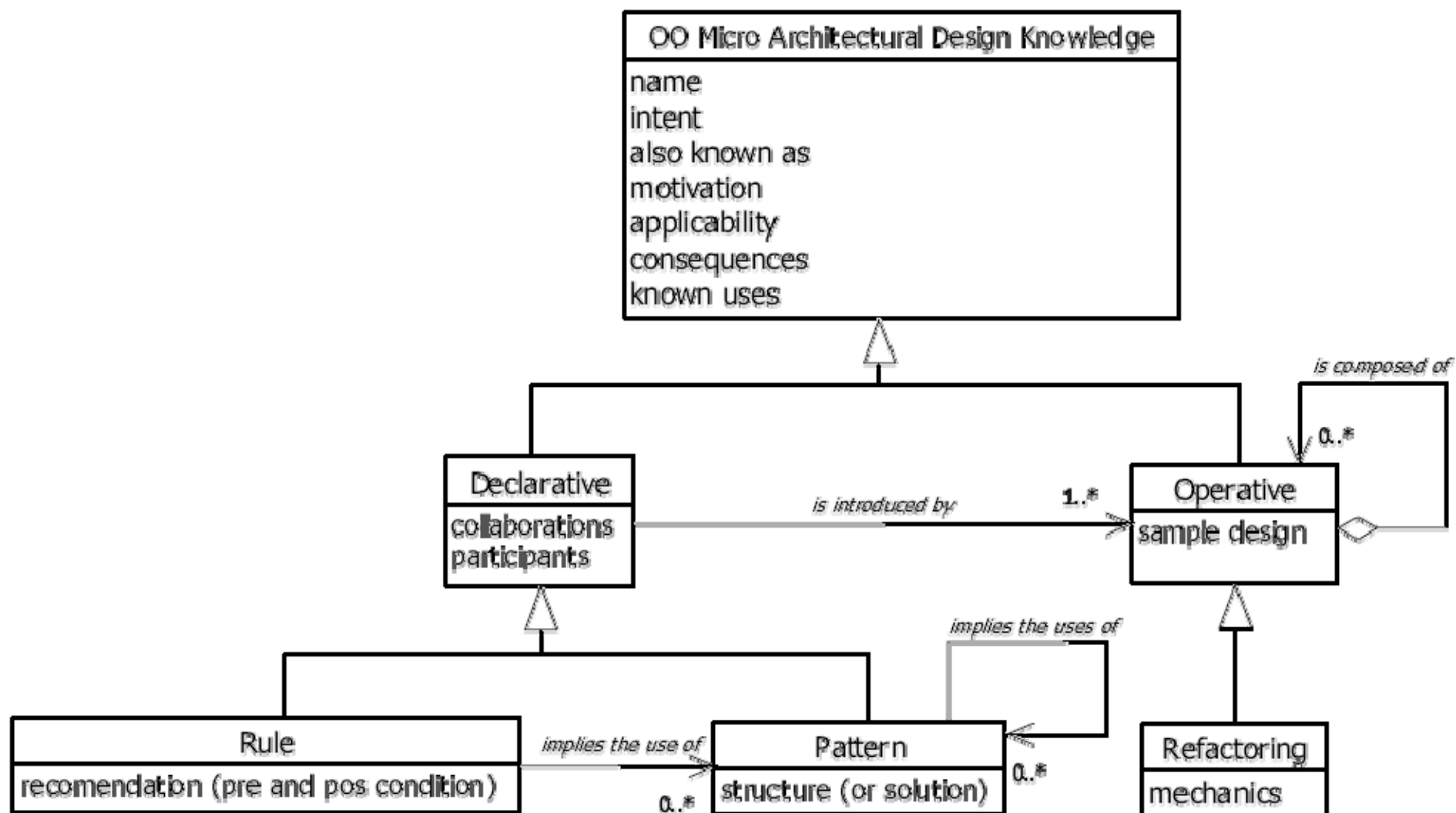
Guide to the Software Engineering Body of Knowledge 2004 Version



Guide to the Software Engineering Body of Knowledge

(2004 Version)





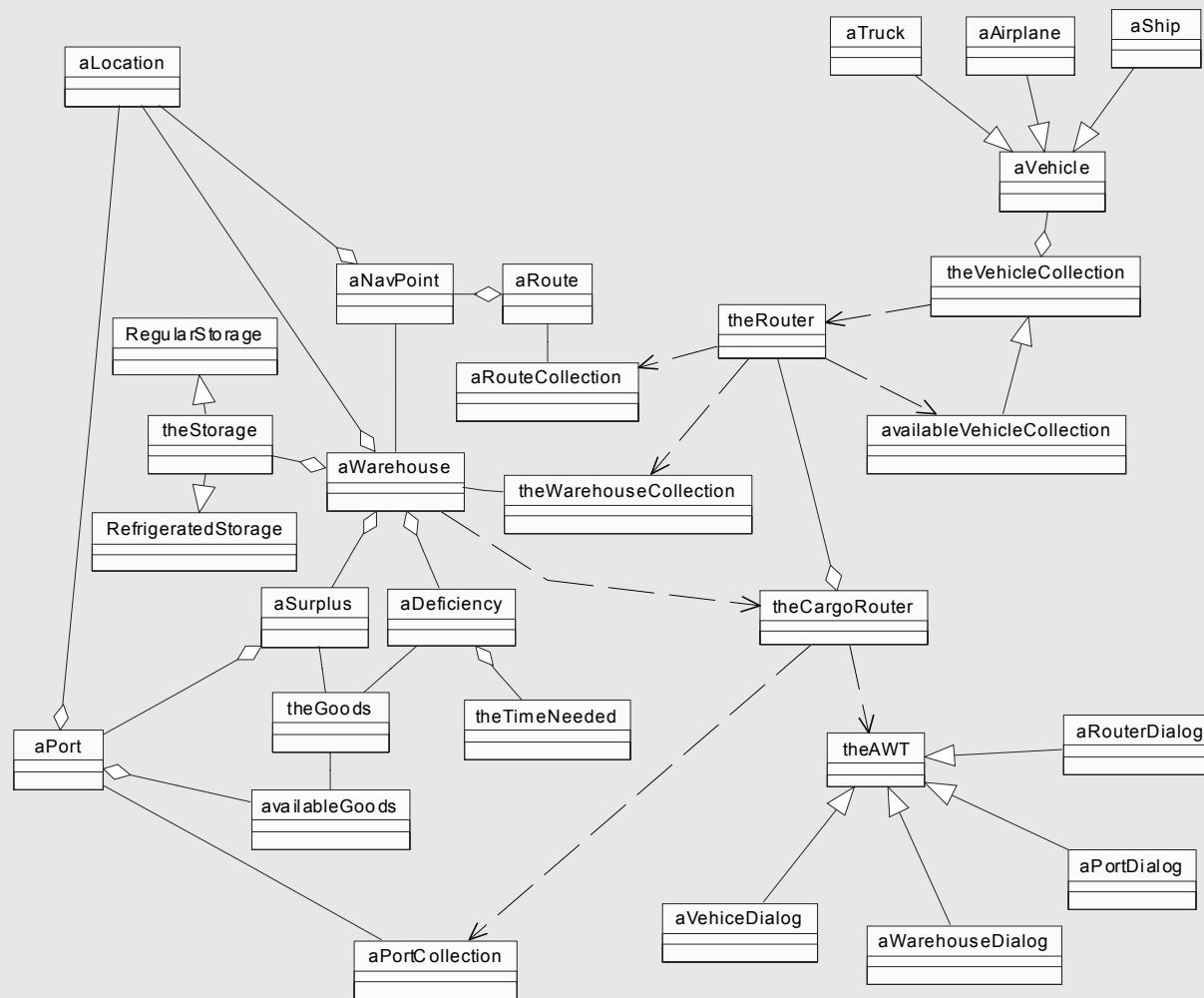
Problemi u oblikovanju programske potpore

- **Ranjivost na globalne (ili široko dijeljene) varijable**
 - Klasični programski jezici kreiraju dijeljenje (blokove strukture, globalne varijable).
- **Nenamjerno otkrivanje interne strukture**
 - Vidljiva reprezentacija može se manipulirati na neželjen način.
- **Prodiranje odluka o oblikovanju**
 - Jedna promjena utječe na mnoge module.
- **Disperzija koda koji se odnosi na jednu odluku**
 - Vrlo je teško utvrditi što je sve pogođeno promjenom.
- **Povezane odluke o oblikovanju**
 - Povezane definicije raspršuju odluke (trebale bi biti lokalizirane - na jednom mjestu)

Čitljivost?



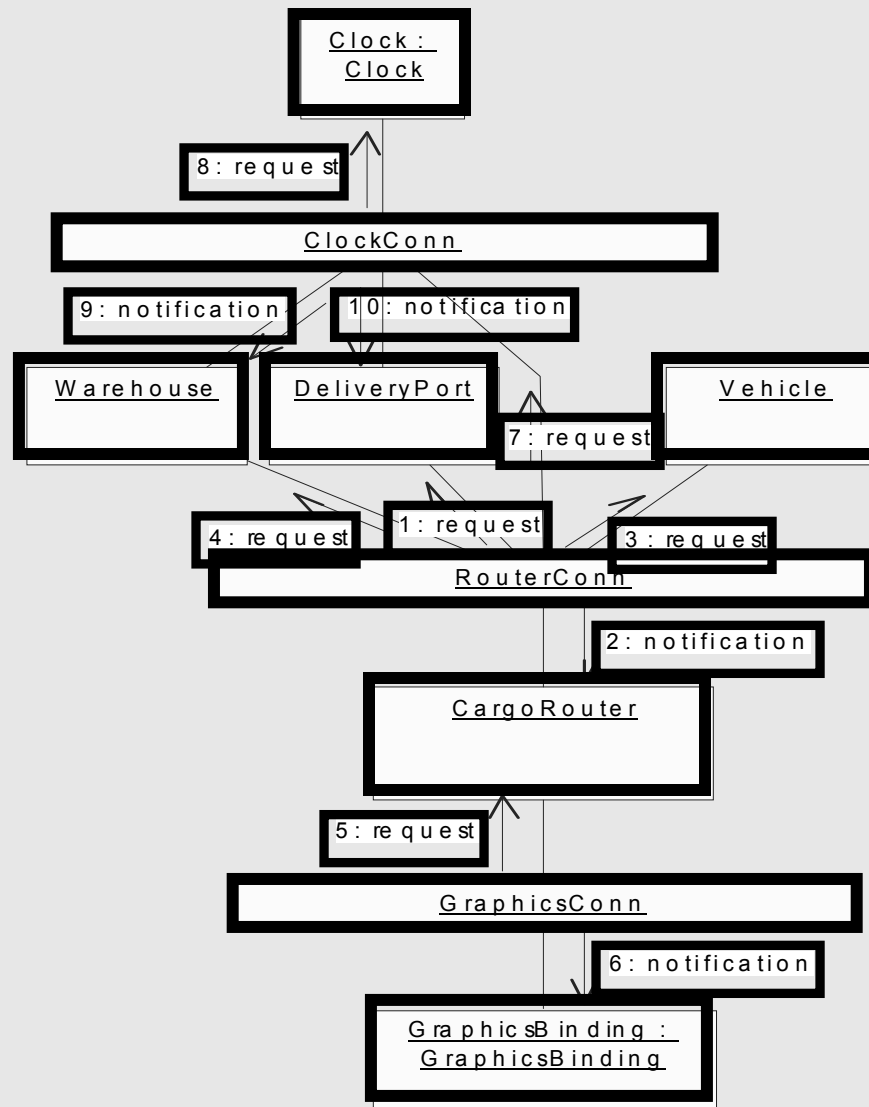
Apstrakcija



Bolje?

Arhitektura

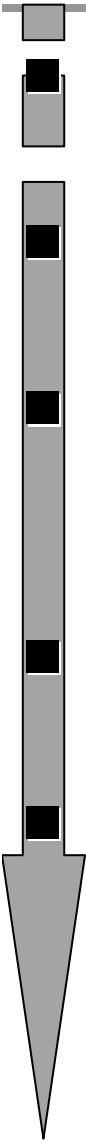
KnockoutJS



Moguće rješenje?

- **modularizacija**
 - **Jednostavnije upravljanje sustavom**
 - princip podijeli i vladaj
 - **Evoluciju sustava**
 - promjene jednog dijela ne utječu na druge dijelove
 - **Razumijevanje**
 - sustav se sastoji od razumno složenih dijelova
- **Koje kriterije koristiti za modularizaciju?**
- **Što je to modul?**
 - Dio koda
 - Jedinica kompilacije, koja uključuje deklaracije i sučelje.
 - D.Parnas, Comm. ACM, 1972. : “Jedinica posla.”

Povijest modularizacije

- 
- **Glavni program i potprogrami**
 - Dekompozicija u procesne korake s jednom niti izvođenja.
 - **Funkcijski moduli**
 - Agregacija (skupljanje) procesnih koraka u module.
 - **Apstraktni tipovi podataka (engl. Abstract Data Types - ADT)**
 - Zatvaranje podataka i operacija, skrivanje predstavljanja.
 - **Objekti i objektno usmjerena arhitektura**
 - Procedure (metode) se povezuju dinamički, polimorfizam, nasljeđivanje.
 - **Komponente i oblikovanje zasnovano na komponentama (engl. Component based design CBD)**
 - Višestruka sučelja, posrednici, binarna kompatibilnost.

Glavni program i potprogrami

- Obilježja:
- Hijerarhijska dekompozicija
 - Temeljena na odnosu definicija – uporaba. Pozivi procedura su interakcijski mehanizam.
- Jedna nit izvođenja
 - Potpomognuto izravno programskim jezikom.
- Hijerarhijsko rasuđivanje
 - Ispravno izvođenje rutine ovisi o ispravnom izvođenju subrutine koja se poziva.
- Implicitna struktura podsustava
 - Podprogrami/subrutine su tipično skupljene u (funkcijske) module.

Apstraktni tipovi podataka

- '60 - “Ako dobro definirate strukture podataka ostatak programa je mnogo jednostavniji.”
- '70 - intuicija o skrivanju informacija i apstraktnim tipovima podataka (ADT – abstract data types):
- Definicija:
- Apstraktni tip podataka - ADT je skup dobro definiranih elemenata i skup pridruženih operacija na tim elementima.
 - Skup elemenata je jedino dohvatljiv preko skupa operacija.
 - Skup operacija se često naziva sučelje (engl. interface).
 - Programski jezici mogu po volji i različito implementirati ADT.
- Taj je pristup potpuno usvojen u objektno usmjerenoj arhitekturi programske potpore.

Primjer: Apstraktni tipovi podataka

Neka k označuje cijeli broj (*integer*).

ADT integer

Podatak – engl. *Data*

Tip elemenata: cijeli brojevi s opcijским prefiksom plus ili minus.

Takav jedan cijeli broj s predznakom neka je N .

Operacije – engl. *Operations*

constructor – kreira novi cijeli broj.

$add(k)$ – kreira novi cijeli broj koji je suma N i k . Posljedica ove operacije je **$sum=N+k$** . To nije naredba pridruživanja, već matematička operacija koja je istinita za svaku vrijednost **sum** , N , k nakon operacije **add** .

$sub(k)$ – slično kao gore kreira novi cijeli broj. Posljedica je **$sum=N-k$** .

$set(k)$ – Posljedica je **$N=k$** .

...

- Gornji opis naziva se **specifikacija** za **ADT integer**. Imena **add** , **sub** , ... , predstavljaju **sintaksu**, dok je **semantika** određena preduvjetima i posljedicama (post-uvjetima) operacija.

Primjer: Apstraktni tipovi podataka

- Lista kao ADT: je sekvencija 0 ili više objekata danog tipa.
- Operacije: `insert(x, p, L)`; (*ubaci element x, na poziciju p, u listu L*), `first(L)`; `locate(x,L)`; `retrieve(p, L)`; `delete(p, L)`; `next(p,L)`;

Primjer programa koji eliminira duplikate u listi L:

```
p = first(L);
while (p != end_of_List) {
    q = next(p,L);
    while(q != end_of_List) {
        if (same(retrieve(p,L),retrieve(q,L))
            delete(q,L);
        else
            q = next(q,L)
    }
    p = next(p, L);
}
```

■ Prednost

- nezavisnost o strukturi podataka i moguća uporaba u bilo kojoj implementaciji liste (npr. od niza do povezane liste). Promjena implementacije liste ne utječe na ovaj kod.

Objektno usmjerena arhitektura (engl. *Object-Oriented Architecture*) temeljni koncepti

Izvor: T.C.Lethbridge, R.Laganier, 2005

Što je objektno usmjerenje?

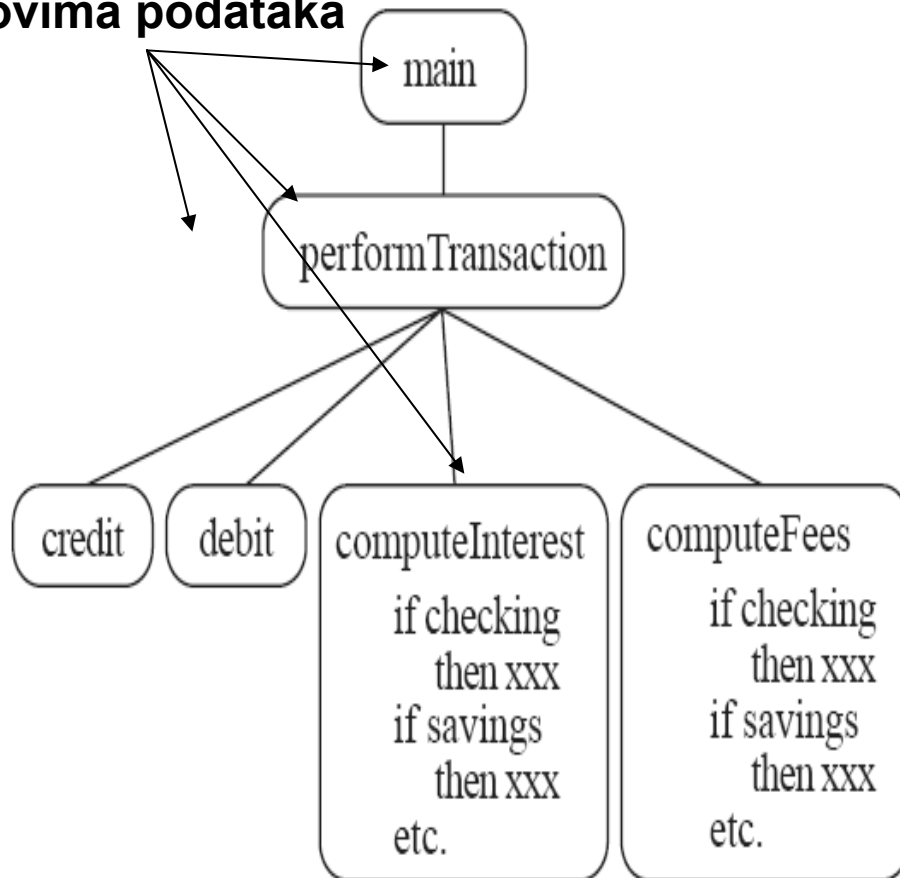
- **Proceduralna paradigma:**
 - Program je organiziran oko pojma *procedura* (*funkcija, rutina*).
 - ***Proceduralna apstrakcija***
 - zadovoljavajuće rješenje za jednostavne podatke
 - ***Dodaje se podatkovna apstrakcija***
 - Grupiranje dijelova podataka koji opisuju neki entitet i manipulacija s njima kao cjeline.
 - Pomaže u smanjenu složenosti sustava.
 - Npr. *records* i *structures* (*ali različiti zapisi traže različite procedure*).
- **Objektno usmjerena paradigma:**
 - ***Organiziranje proceduralnih apstrakcija u kontekstu podatkovnih apstrakcija.***

Objektno usmjerena paradigma

- **Pristup rješenju problema u kojem se sva izračunavanja (engl. computations) obavljaju u kontekstu objekata.**
- **Objekti su instance programskih konstrukcija koje nazivamo razredima (klasama).**
 - **Razredi:**
 - **Podatkovne apstrakcije**
 - **Sadrže proceduralne apstrakcije koje izvode operacija na objektima.**
- **Program u radu se može sagledati kao skup objekata koji u međusobnoj kolaboraciji obavljaju dani zadatak.**

Pogledi na primjer bankovnog sustava

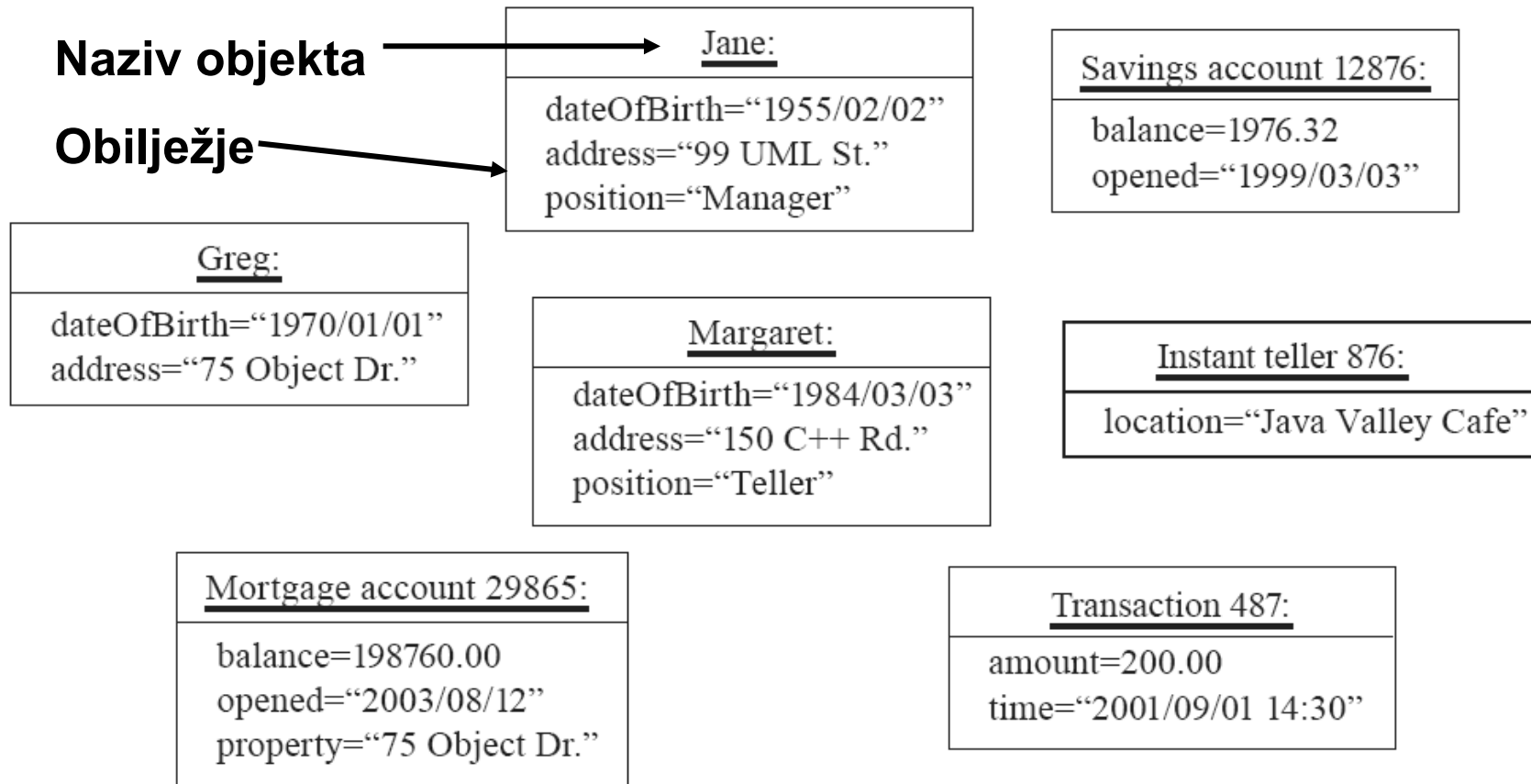
Procedure manipuliraju različitim tipovima podataka



Objekti

- Dio strukturiranih podataka programa u radu.
- Može predstavljati bilo što iz realnog svijeta čemu se mogu pridružiti:
 - Obilježja (engl. *properties*) koja karakteriziraju objekt.
 - opisuju trenutno stanje objekta.
 - Ponašanje (engl. *behaviour*)
 - Kako objekt reagira (što može rezultirati u promjeni stanja).
 - Može simulirati ponašanje objekata iz realnog svijeta.

Primjer: Objekti u bankovnom sustavu



U analizi sustava temeljenog na objektno usmjerenoj paradigmi ne bavimo se programskim kodom niti problemima smještaja u memoriji ili disku.

Razredi

- **Razred/Klasa – engl. Class:**
 - Jedinica apstrakcije u objektno usmjerenoj paradigmi.
 - Razredi predstavljaju slične objekte.
 - Objekti su instance razreda.
 - Vrsta programskog modula koji:
 - Opisuje strukturu instanca, t.j obilježja (engl. properties)
 - Sadrže metode (procedure) koje implementiraju ponašanje objekata.

Odnos razreda i instance

- Nešto je razred ako može imati instance.
- Nešto je instanca ako je jasno da je to jedan član skupa definiranog kao razred.
 - *Film*
 - Razred, instance su individualni filmovi.
 - *Bubanj (namotaj) filma:*
 - Razred; instance su fizikalni bubnjevi.
 - *Bubanj filma sa serijskim brojem W19876*
 - Instanca razreda ReelOfFilm
 - *Science Fiction*
 - Instanca razreda Zavr
 - *Science Fiction Film*
 - Razred, instanca je npr.: 'Star Wars'
 - *Prikazivanje filma 'Star Wars' u Cinestaru u 19:00:*
 - Instanca razreda PrikazivanjeFilma

Imenovanje razreda

- **Koristi velika slova**
 - BankAccount a ne bankAccount
- **Koristi imenice u singularu.**
- **Koristi ispravnu razinu generalizacije**
 - Ne Grad nego općenitiji pojam, npr. Naselje
- **Budi siguran da ime ima samo jedno značenje.**
 - Npr. 'Bus' ima više značenja.

Razlika instanca - objekt

- Nema razlike, odnosi se na istu jedinku (entitet)
 - Razlika je nastala u korištenju prirodnog jezika.
-
- Npr. kćer – djevojka
 - “Imala je sedam kćeri” (ne djevojaka).
 - “Vidio sam prekrasnu djevojku” (ne kćer).

Varijable instanci

- **engl. Instance Variables**
- **definirane unutar razreda**
- **Varijabla je mjesto (u instanciji) gdje se smještaju podaci (place-holder, slot, field).**
- **Variable definirane unutar razreda odgovaraju (različitim) podacima koji se nalaze u svakoj instanciji toga razreda.**
- **Sintaksa: `name[multiplicity]:type=initial_value`**
- **Postoje dvije skupine varijabli instanci:**
 - **Atributi (obilježja objekta)**
 - **To su jednostavni podaci kao npr.:**
 - `name`, `dateOfBirth`
 - **Asocijacije (pridruživanje) između instanci različitih razreda.**
 - **To su odnosi (engl. relationships) prema drugim instancama drugih razreda. Npr.:**
 - `supervisor`, (odnos prema instancijama razreda `Manager`).
 - `coursesTaken` (odnos prema instancijama razreda `Course`).

Varijable instanci

- Ako neki razred ima definiranu varijablu instanci `var`, tada sve instance toga razreda imaju mjesto (“field” ili “slot”) s nazivom `var`.
- Stvarni podaci smješteni u varijablu `var` razlikuju se od objekta do objekta. Npr.:
- Razred: `Employee`
- Variabla: `supervisor`
- Postoje različiti supervizori u svakoj instanci razreda `Employee`

Variable i objekti

- uobičajena zabuna:

Variable i objekti su *zasebni i različiti koncepti*

- Tip varijable:

- **Određuje koje razrede objekata može sadržavati.**

- U Javi postoje dva tipa:

- *Primitive* (sadrži jednu vrijednost, nije objekt, evaluira se u vrijednost koju sadrži.

- *Reference* (ili object) tip (evaluira se u adresu objekta). Slično pokazivaču ali u širem kontekstu.

- U različitim trenucima može se odnositi na različite objekte.

- **Jedan objekt može u isto vrijeme referencirati više različitih varijabli.**

Varijable razreda

- engl. Class variables
- Varijable razreda se identificiraju ključnom riječi (modifikatorom) *static*.
- Varijabla razreda može sadržavati vrijednost.
- Tu vrijednost *dijele* sve instance toga razreda (npr. u C++ “static data member”).
 - Ako jedna instanca upiše vrijednost u varijablu razreda, sve instance toga razreda vide izmijenjenu vrijednost.
 - Uvijek postoji samo jedna vrijednost te varijable
 - Varijable razreda su korisne za:
 - Default ili ‘constant’ vrijednosti (npr. PI).
 - Lookup tablice i slične strukture.

Lokalne varijable

- Slično kao što objekti pohranjuju svoja stanja u slotove, procedure (metode) će često privremeno pohraniti stanje u lokalne varijable.
- Sintaksa deklaracije lokalne varijable je slična
 - (npr. `int count = 0;`).
- Ne postoji poseban ključna riječ za određivanje lokalnih varijabli.
- Doseg varijable je unutar zagrada procedure (metode).
- Lokalna varijabla je vidljiva samo u metodi gdje je deklarirana.
- Nije joj moguće pristupiti iz ostatka razreda.

Parametri

- **Npr. neka `main` metoda ima oblik::**
- `public static void main(String[] args)`
- **Ovdje, `args` variabla je parametar procedure (metode) `main`.**
- **Parametri se uvijek klasificiraju kao "varijable".**
- **To se odnosi i na druge konstrukcije koje prihvataju parametre (npr. *constructors* i *exception handlers*).**

Metoda

- **Metoda (način izvođenja neke operacije) = procedura, funkcija, rutina (u C++ “function member”).**
 - **Proceduralna apstrakcija koja se koristi za implementaciju ponašanja razreda.**
 - **Nekoliko različitih razreda može imati metodu istog naziva.**
 - **Sve te metode implementiraju istu apstraktnu operaciju na način kako odgovara pojedinom razredu (različito).**
 - **Npr.: Operacija Izračun_površine u pravokutniku je različito implementirana nego za krug (iako je ime metode isto).**

Operacija

- definicija:

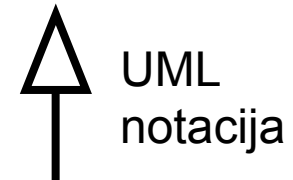
- Proceduralna apstrakcija više razine nego metoda.
- Operacija specificira tip ponašanja.
- Neovisna je o kodu koji implementira njeno ponašanje.
 - Npr. Izračun_površine (sasvim općenito).

Polimorfizam

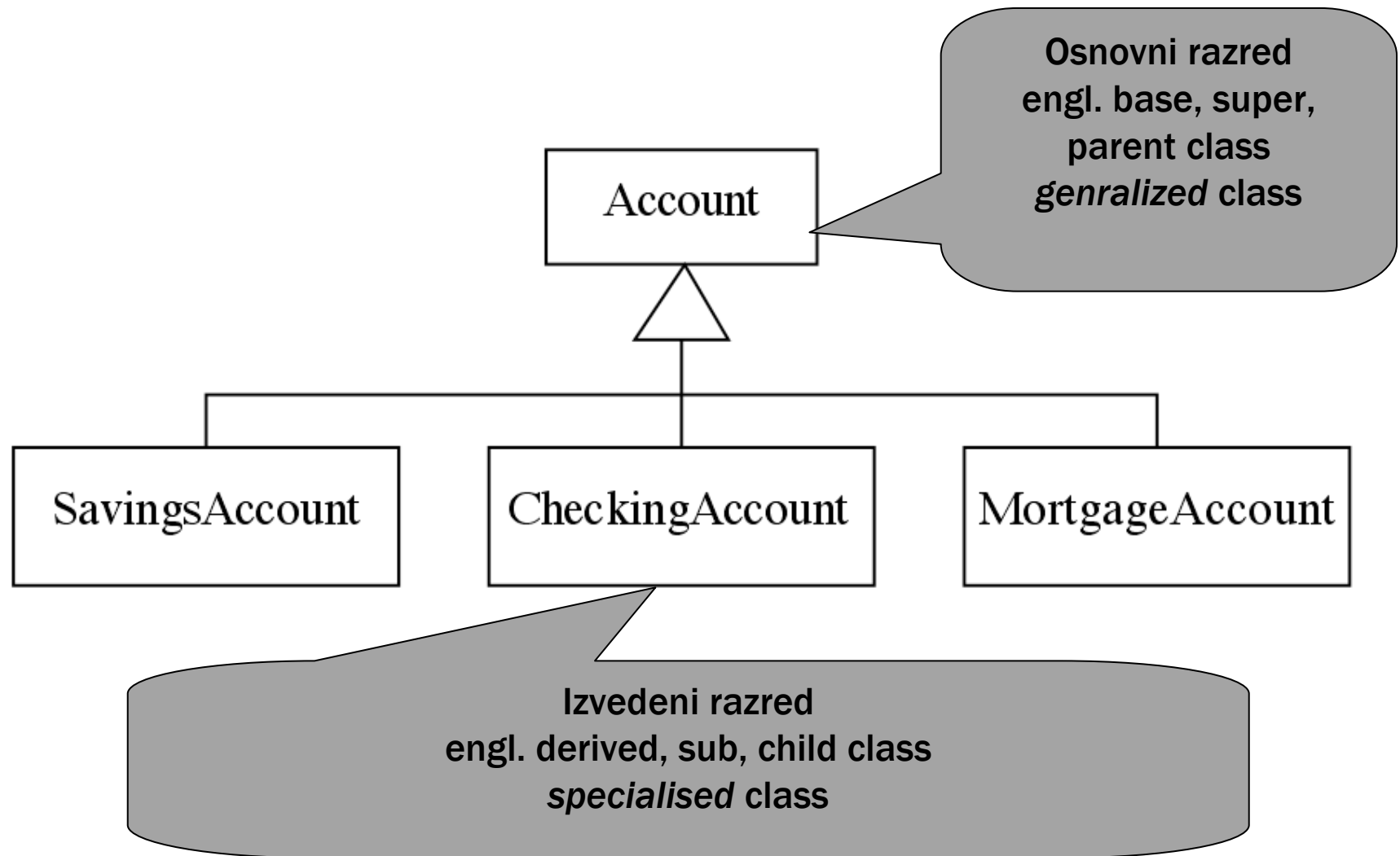
- Svojstvo objektno usmjerenog programa da se jedna apstraktna operacija može izvesti na različite načine u različitim razredima.
- Polimorfizam zahtjeva da postoji više metoda istog naziva.
 - Izbor koja metoda će se izvesti ovisi o razredu objekta koji se nalazi u varijabli.
 - Polimorfizam smanjuje potrebu da programeri kodiraju mnoge if-else ili switch naredbe

Organizacija razreda u hijerarhije

- Superrazredi (engl. Superclasses) (u C++ “base class”)
 - Sadrže značajke zajedničke jednom skupu razreda.
- Hijerarhije nasljeđivanja (engl. Inheritance hierarchies)
 - Pokazuju odnos između superrazreda i podrazreda (engl. superclasses i subclasses).
 - Oznaka trokuta pokazuje generalizaciju.
- Nasljeđivanje (engl. Inheritance)
 - Svi podrazredi *implicitno* posjeduju značajki koje su definirane u superrazredu.



Primjer: hijerarhija nasljeđivanja



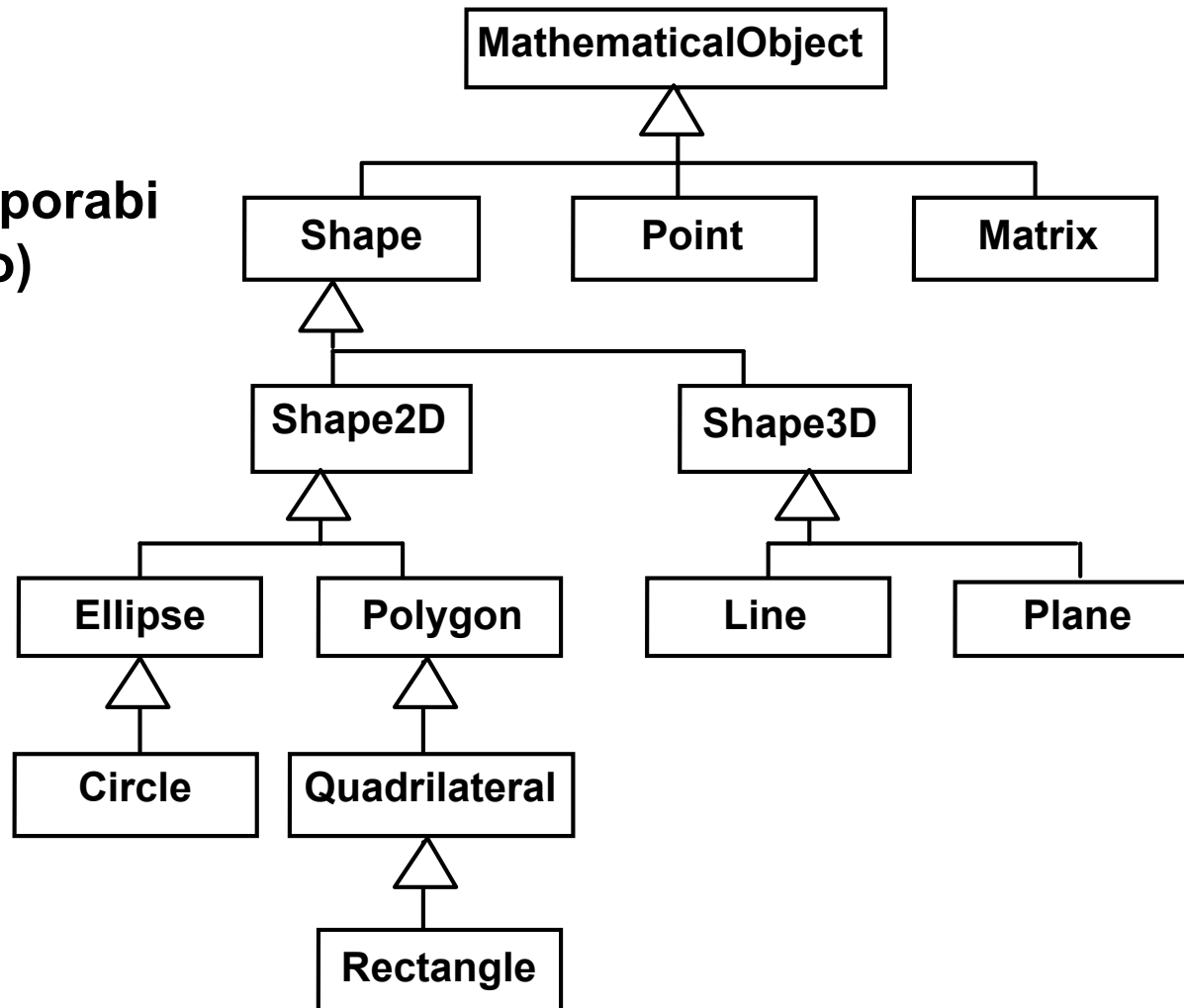
“Isa” pravilo

- Uvijek provjeri da li generalizacija zadovoljava “is a” pravilo (pravilo “je”, odnos podskup → skup).
 - “checking account *is an* account”
 - “village *is a* municipality”
- Da li bi “Županija” bila podrazredom “Država” ?
 - Ne, jer ne zadovoljava “isa” pravilo:
 - “Županija je država” ne vrijedi !

Primjer

■ Moguća hijerarhija nasljeđivanja matematičkih objekata

(U provjeri uporabi
“is a” pravilo)

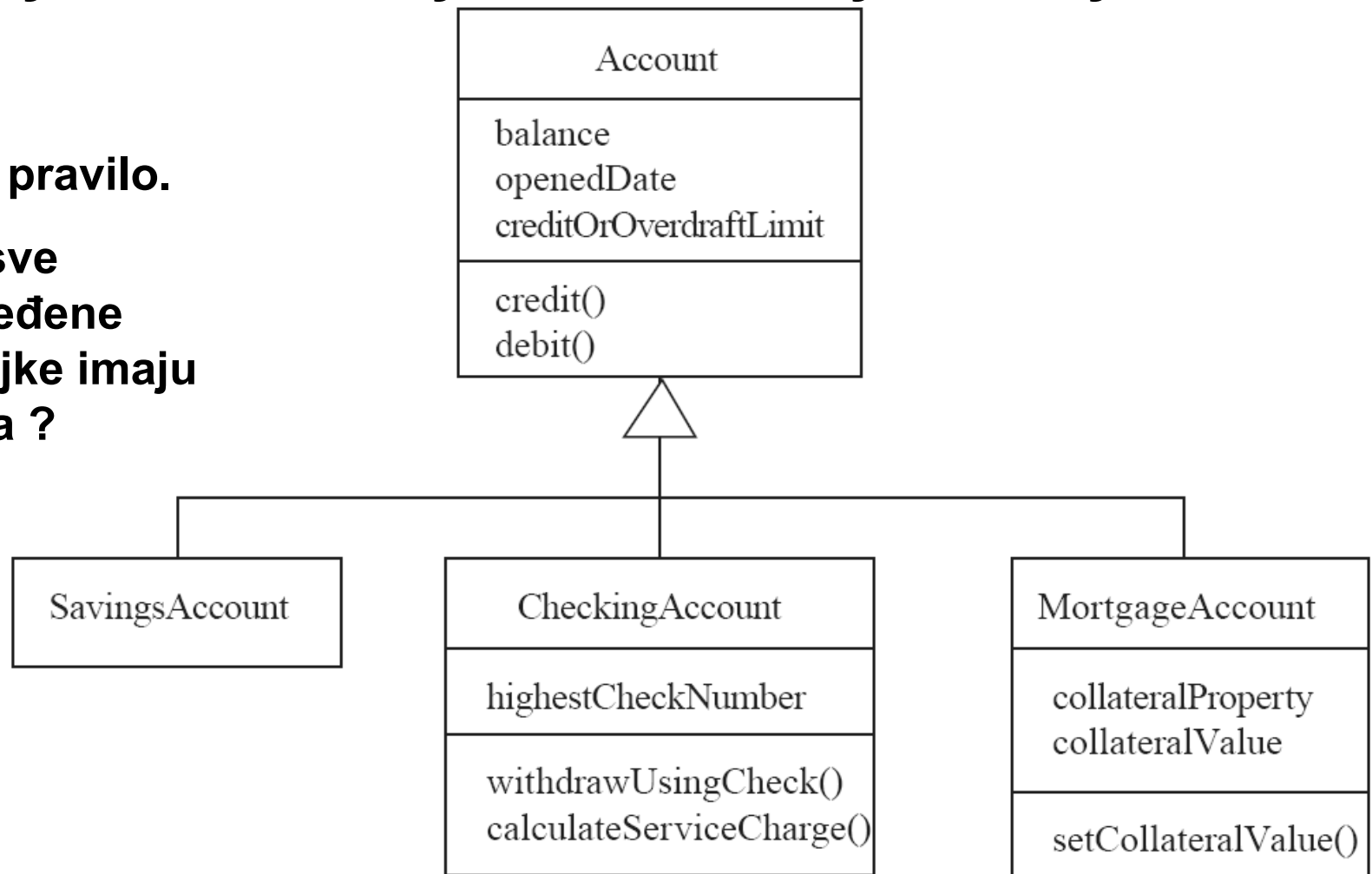


Primjer:

■ Osiguraj da sve naslijeđene značajke imaju

smisla
Provjeri:

1. “Is a” pravilo.
2. Da li sve naslijeđene značajke imaju smisla ?



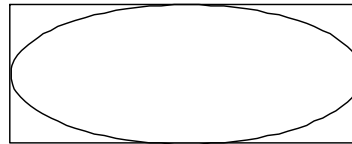
Liskov princip zamjene (supstitucije)

- Ako postoji varijabla čiji tip je superrazred, program se mora korektno izvoditi ako se u varijablu pohrani instancija tog superrazreda ili instancija bilo kojeg podrazreda.
 - podrazredi nasljeđuju sve od superrazreda
- Barbara Liskov
 - Prof. MIT,
 - 2008 ACM Turing nagrada “for her work in the design of programming languages and software methodology that led to the development of object-oriented programming”.

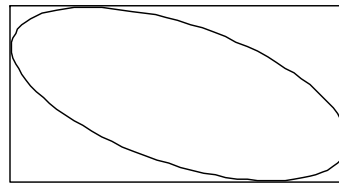
Primjer: operacije na grafičkim objektima

- operacije ništa ne govore o implementaciji

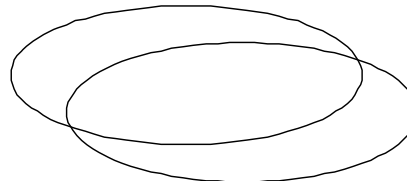
Original objects
(showing bounding rectangle)



Rotated objects
(showing bounding rectangle)



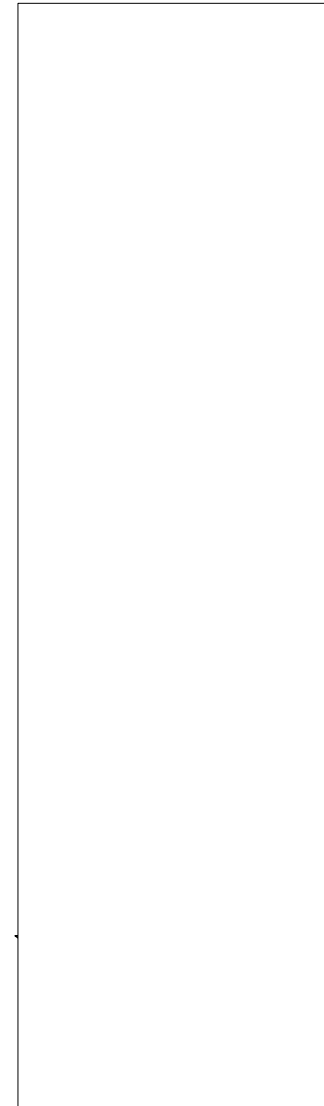
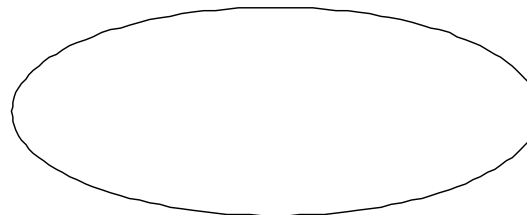
Translated objects
(showing original)



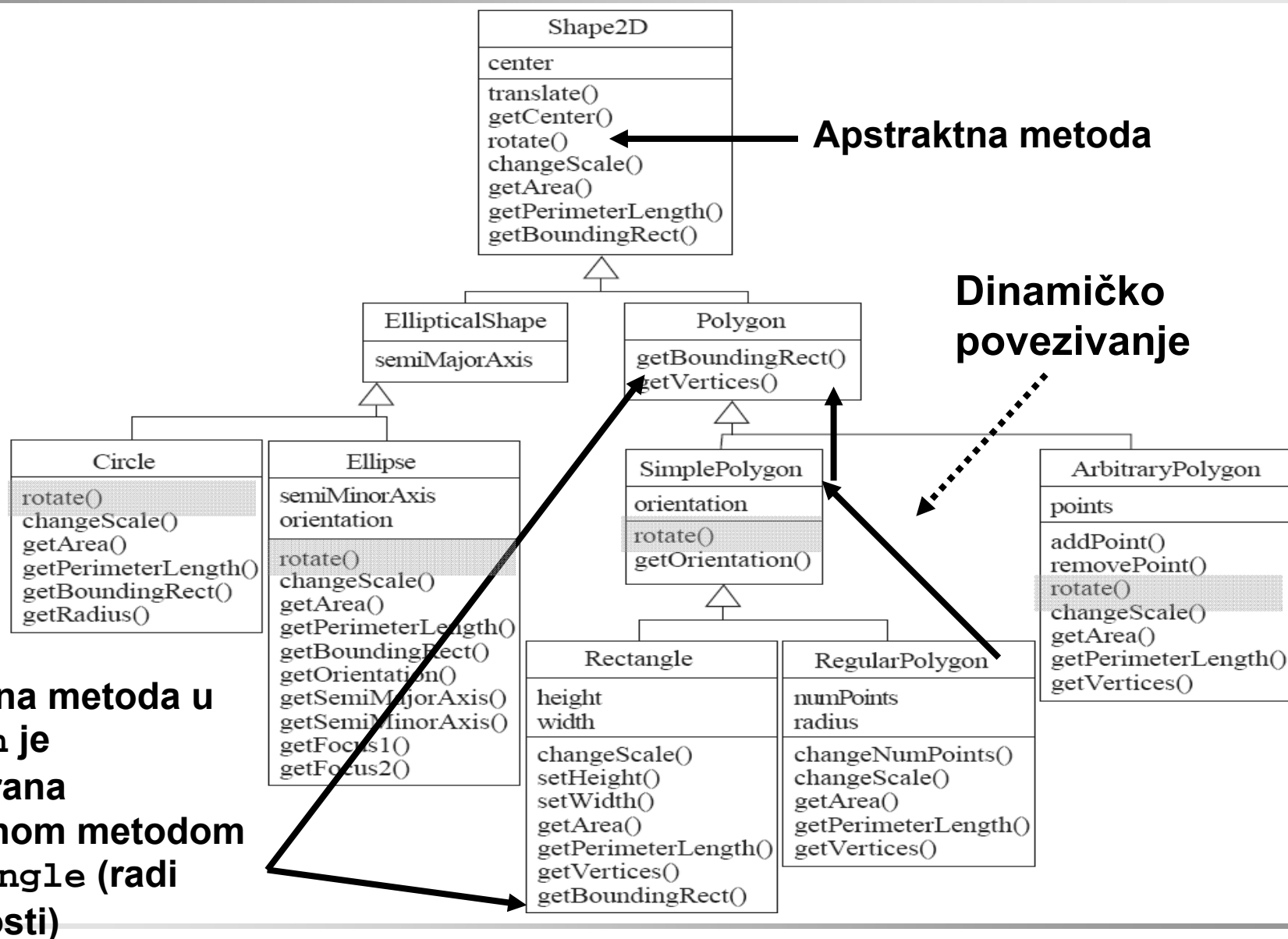
Scaled objects
(50%)



Scaled objects
(150%)



Nasljeđivanje, Polimorfizam i Varijable



Apstraktni razredi i metode

- Pojedina operacija treba biti *deklarirana* da postoji u najvišem hijerarhijskom razredu gdje ima smisla.
- Na toj razini operacija može biti *apstraktna* (bez implementacije)
 - Npr. `rotate` u `Shape2D` (vidi sliku hijerarhije matematičkih objekata, gdje će podrazredi će imati svoje specifične `rotate`).
- Ako neka operacija nema implementacije, cijeli razred je “*apstraktan*”
 - Ne mogu se kreirati instance.
 - Suprotno, ako postoje sve implementacije (naslijeđene ili definirane), razred je “*konkretan*”.
- Ako superrazred ima *apstraktnu* operaciju, tada na nekoj nižoj razini hijerarhije mora postojati konkretna metoda za tu operaciju.
 - Krajnji razredi u hijerarhiji (“lišće”) moraju implementirati ili naslijediti konkretne metode za sve operacije.
 - Ti razredi moraju biti konkretni.

Nadjačavanje

- **engl. Overriding**
- **redefiniranje, ne obaziranje, iako je definirana u superrazredu i može se naslijediti, redefinira se u podrazredu)**
- **Metoda bi se naslijedila ali umjesto toga podrazred sadrži novu verziju. To se koristi za:**
 - **Restrikciju**
 - Npr. `scale(x,y)` ne bi radila u `Circle` (`Circle` bi postao `Ellipse`)
 - **Proširenje (ekstenziju)**
 - Npr. `SavingsAccount` razred bi mogao zaračunavati neku dodatnu pristojbu.
 - **Optimizaciju**
 - Npr. `getPerimeterLength` metoda u `Circle` je jednostavnija od one u `Ellipse`.
- **U ranijoj slici `Shape2D` hijerarhije, konkretna metoda u `Polygon` je nadjačana/redefinirana konkretnom metodom u `Rectangle` (radi efikasnosti).**

Odabir metode za izvođenje

- koja će se metoda izvoditi donosi se temeljem slijedećeg algoritma:
 1. Ako postoji konkretna metoda za operaciju u trenutnom razredu, ta se metoda izvodi.
 2. Inače, metoda je naslijeđena i pogledaj da li postoji implementacija u neposrednom superrazredu.
 - Ako postoji izvedi ju.
 3. Ponovi korak pod 2., provjeravajući sukcesivno u višim superrazredima dok ne nađeš konkretnu metodu, te je izvedi.
 4. Ako metoda nije pronađena, postoji pogreška u programu.
 - Java i C++ program neće se moći kompilirati.

Dinamičko povezivanje

- **engl. dynamic binding**
- **Pojavljuje se u slučaju kada se odluka o izboru konkretne metode donosi za vrijeme izvođenja programa (engl. at run-time).**
- **To je potrebno kada:**
 - Varijabla je deklarirana da je tipa superrazreda (t.j. postoji hijerarhija podrazreda toga tipa varijable).
 - Postoji više polimorfnih metoda koje se mogu izvesti u sklopu hijerarhije razreda određene superrazred tipom varijable.
 - **Npr.:**

Neka postoji varijabla aShape, a njen tip je Shape2D.

To znači da aShape može sadržavati objekt (instancu) iz bilo kojeg konkretnog razreda u hijerarhiji razreda Shape2D.

Pretraživanje i odabir konkretne metode započinje pregledom razreda čiji objekt se stvarno nalazi u varijabli aShape.

Ako metoda nije pronađena u tom razredu, pretražuje se sukcesivno hijerarhija razreda idući prema gore sve do superrazreda Shape2D koji je naveden kao tip varijable.

Ako metoda nije pronađena deklarira se pogreška.

Primjer dinamičkog povezivanja

■ na primjeru Shape2D hijerarhije

- Neka postoji objekt iz razreda `RegularPolygon` u varijabli `aShape` koja je tipa `Shape2D`, i želi se izvesti metoda `getBoundingRect`.
- Program prvo traži tu konkretnu metodu u razredu `RegularPolygon`, zatim u razredu `SimplePolygon`, te u razredu `Polygon` (gore po hijerarhiji) gdje ju i nalazi. To je dinamičko povezivanje.
- Ako `aShape` (koja je tipa `Shape2D`) sadrži objekt iz razreda `Rectangle`, program odmah nalazi konkretnu metodu (`getBoundingRect` je konkretna metoda u tom razredu). To je također dinamičko povezivanje, jer se unaprijed ne zna koji je razred objekta u varijabli..
- Ako pak neka varijabla `myRect` je tipa `Rectangle` (koji nema podrazreda), te varijabla sadrži objekt iz razreda `Rectangle`, prevoditelj statički određuje metodu za izvođenje. Dinamičko povezivanje odigrava se samo u slučaju kada je tip varijable superrazred (t.j. kada postoje podrazredi toga tipa varijable).

Koncepti objektnog usmjerenja

- **Nužna obilježja da bi sustav ili programski jezik bio objektno usmjeren (definiraju OO) :**
 - **Identitet**
 - Svaki objekt je poseban i može se referencirati (adresom).
 - Dva objekta su posebna čak i ako imaju jednake podatke.
 - **Razredi**
 - Programski kod je organiziran uporabom koncepta razreda, koji svaki za sebe opisuju skup objekata.
 - **Nasljeđivanje**
 - To je mehanizam u kojem se značajke podrazreda implicitno nasljeđuju od superrazreda.
 - **Polimorfizam**
 - Mehanizam u kojem postoji više metoda istog naziva koje različito (ovisno o razredu objekta) implementiraju istu apstraktnu operaciju.

Osnovni principi objektno orijentacije

■ Apstrakcija

- olakšava savladavanje složenih problema
- Objekt -> nešto u realnom svijetu
- Razred -> Objekti (instancije)
- Superrazred -> Podrazred
- Operacija -> Metode
- Atributi i pridruživanje (asocijacije) -> Varijable instanci

■ Enkapsulacija

- detalji mogu biti skriveni u razredima.
- potiče skrivanje informacija (engl. information hiding):
 - Programeri ne moraju znati sve detalje razreda.

■ Modularnost

- Program se može oblikovati samo iz razreda (bez globalnih varijabli?).

■ Hijerarhija

- elementi istog hijerarhijskog nivoa moraju biti na istom nivou apstrakcije

Dobra praksa objektnog programiranja

■ Komentari

- Komentiraj sve što nije očigledno.
- Nemoj komentirati očigledno.
- Komentari čine 25-50% koda.

■ Organiziraj elemente razreda konzistentno

- Redom: varijable, konstruktore, public metode, privatne metode.

■ Izbjegavaj dupliciranje koda

- Ne “kloniraj” ako je to moguće (kloniranje može imati pogrešku u obje kopije, ispravljanje u jednoj ne ispravlja drugu).

■ Pridržavaj se principa objektnog usmjerenja

- Npr.: ‘is a’ pravilo.

■ Preferiraj nedostupnost informacija

- Npr. deklariranjem private.

■ Ne miješaj kod korisničkog sučelja s ostalim kodom u programu

- Interakciju s korisnicima stavi u posebne razrede.
- Time je ostatak koda ponovo uporabljiv.

Razvoj programskih jezika



- 2006, Brian Hayes, "The Semicolon Wars."

Objektno usmjereni programski jezici

- **Prvi objektno usmjeren programski jezik bio je Simula-67.**
 - Oblikovan kako bi programeri pisali simulacijske programe.
- **U ranim 1980-ima razvijen je Smalltalk u Xerox PARC.**
 - Nova sintaksa, velike knjižnice otvorenog koda spremnog za višestruku uporabu (engl. reuse), “bytecode”, nezavisnost o platformi, skupljanje smeća (engl. garbage collection).
- **Kasne 1980-te, razvijen je C++ (B. Stroustrup),**
 - Prepoznate su prednosti objektnog usmjerenja, ali također i činjenica da postoji ogromna skupina C programera.
- **1991, Sun Microsystems je počeo projekt koji bi predložio jezik za programiranje potrošačkih (engl. consumer) pametnih naprava.**
 - 1995., novi jezik je nazvan Java, i formalno predstavljen na konferenciji SunWorld '95.
- **2000., Microsoft predstavlja C# kao kompeticiju Javi.**
 - Prva specifikacija C# jezika dana je 2001.

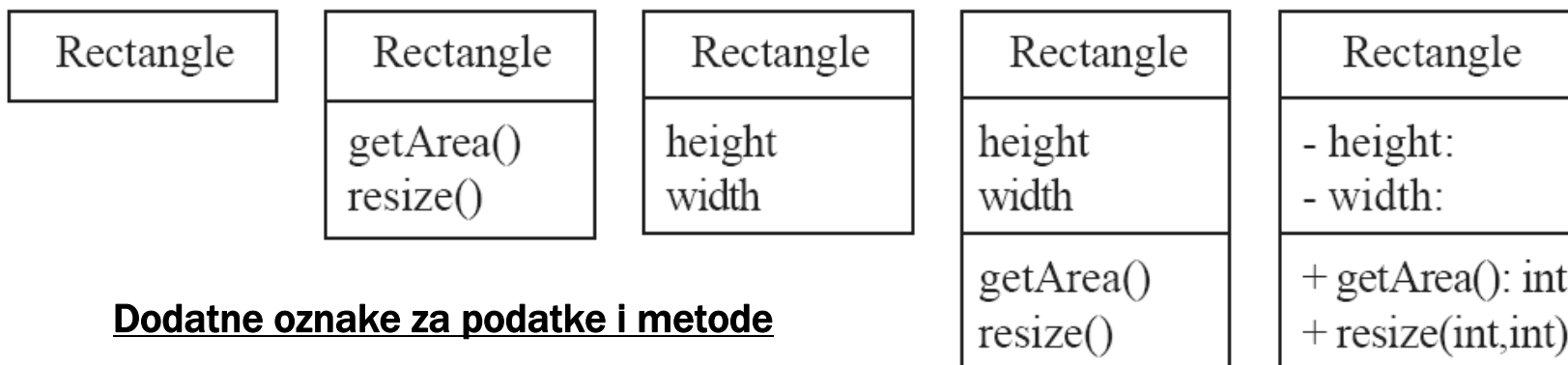
Temelji UML dijagrama razreda

- **Osnovni simboli prikazani na dijagramima razreda su:**
 - **Razredi**
 - predstavljanju tip podataka (podatkovne apstrakcije koje sadrže procedure).
 - **Pridruživanja (engl. Associations)**
 - predstavljaju vezu između instanci razreda.
 - **Atributi**
 - jednostavni podaci sadržani u razredima i njihovim instancama.
 - **Operacije**
 - predstavljaju funkcije koje izvode razredi i njihove instance.
 - **Generalizacije**
 - grupiranje razreda u hijerarhiju nasljeđivanja.

Razredi

- Razred je predstavljen pravokutnikom s imenom.
 - Grafički simbol razreda može prikazati attribute i operacije.
 - Cjeloviti potpis operacije (metode) je::

operationName(parameterName: parameterType ...): returnType



Dodatne oznake za podatke i metode

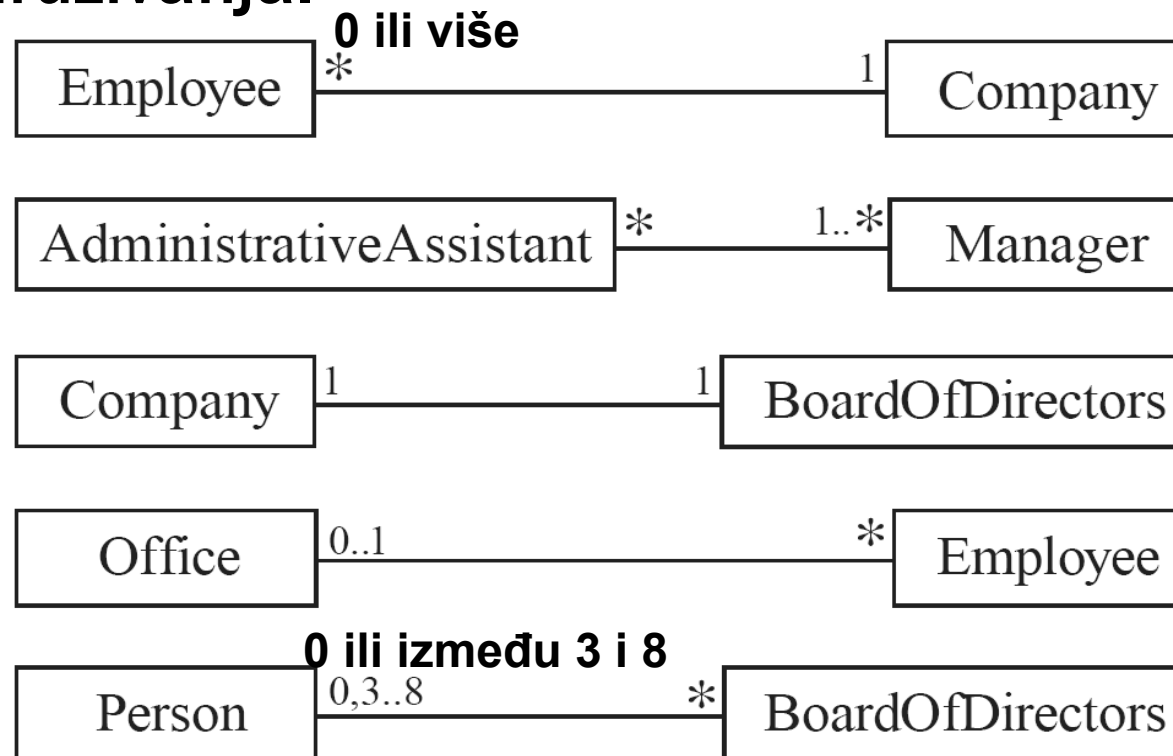
+ **public** (dostupni svima)

- **private** (dostupni unutar razreda)

protected (dostupni od podrazreda)

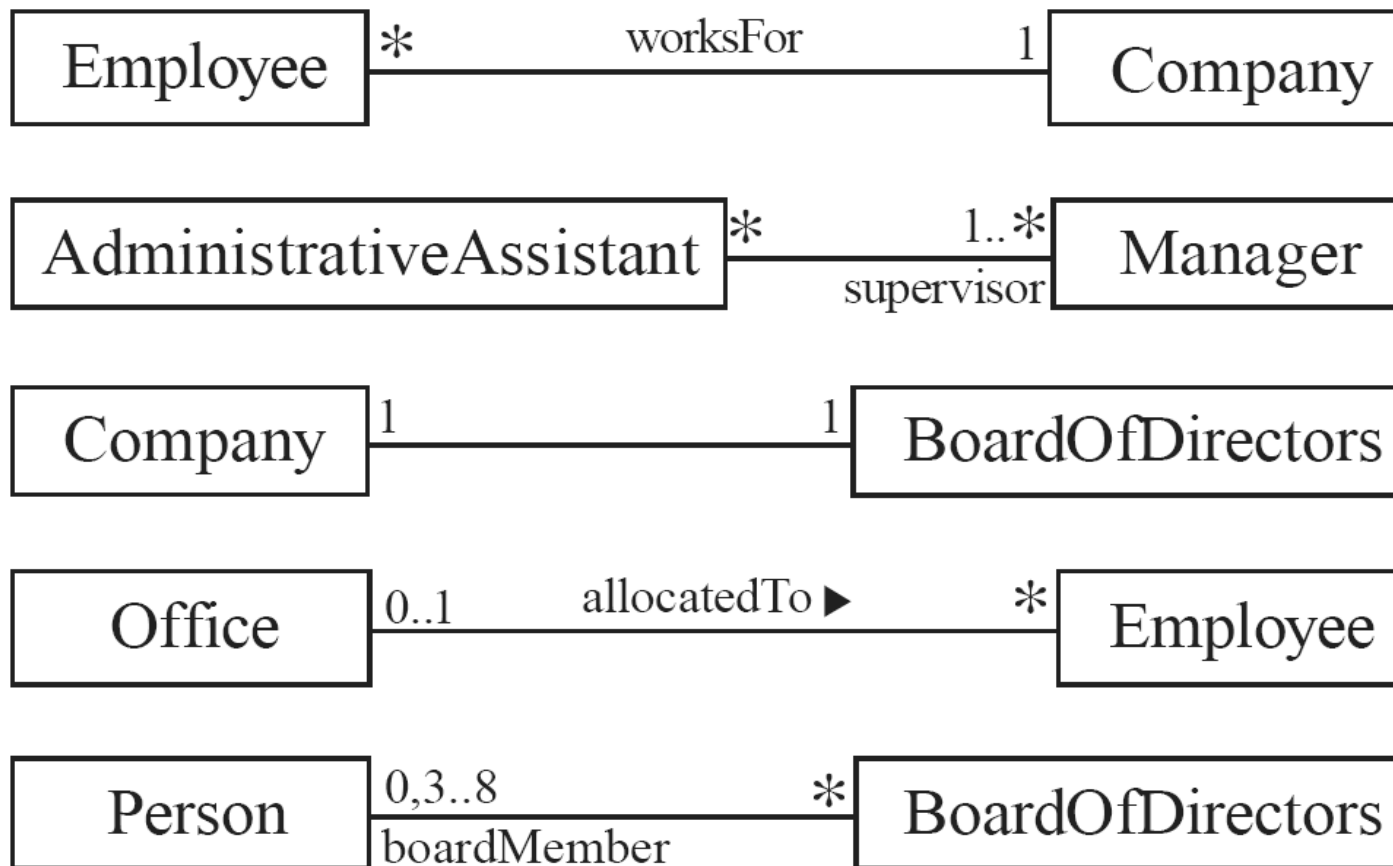
Pridruživanje i brojnost (višestrukost)

- Pridruživanje ili asocijacija pokazuje odnos između dva razreda.
- Brojnost pokazuje broj instanci razreda.
- Simboli koji pokazuju brojnost smješteni su na svakom kraju pridruživanja.



Označavanje pridruživanja

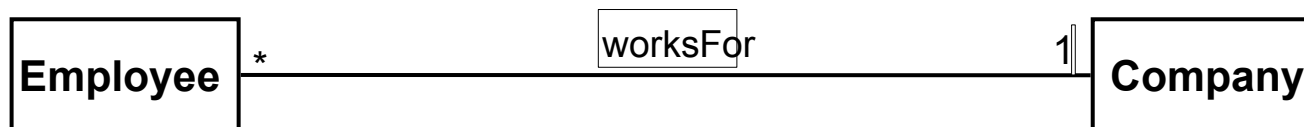
- Svako pridruživanje može se označiti kako bi se ekspliciralo njegovo značenje.



Analiza i validacija pridruživanja

■ Mnogo - jedan

- Jedna kompanija ima mnogo zaposlenika.
- Jedan zaposlenik može raditi samo za jednu kompaniju.
 - Ta kompanija nema podataka o radu “u fušu” (engl. *moonlighting*).
- Kompanija ima nula zaposlenika.
 - Npr. početna registracija, ili krovna kompanija.
- Nije moguće biti zaposlenik ako ne radiš za neku kompaniju.



Analiza i validacija pridruživanja

■ Mnogo - mnogo

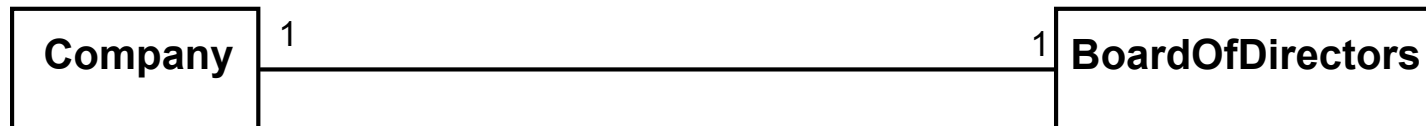
- Jedan asistent može raditi za mnogo menadžera.
- Jedan menadžer može imati mnogo asistenata.
- Asistenti mogu biti organizirani u skupove (engl. pool).
- Menadžeri mogu imati grupu asistenata.
- Neki menadžeri ne moraju imati asistenata.



Analiza i validacija pridruživanja

■ Jedan - jedan

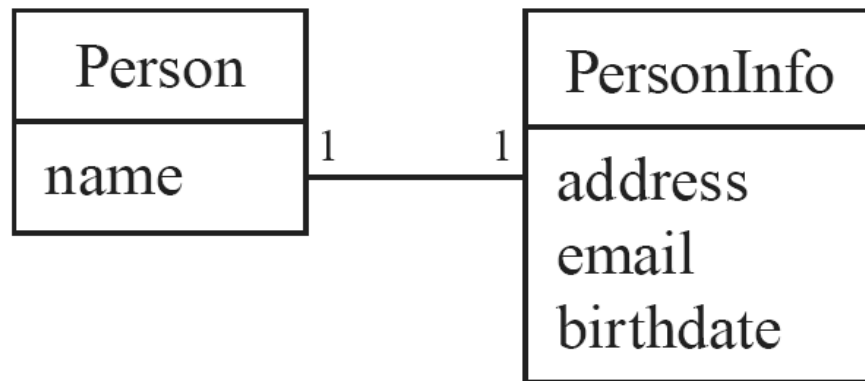
- U svakoj kompaniji postoji točno jedan odbor direktora.
- Odbor direktora je odbor samo jednoj kompaniji.
- Kompanija mora uvijek imati odbor direktora.
- Odbor direktora je uvijek odbor u nekoj kompaniji.



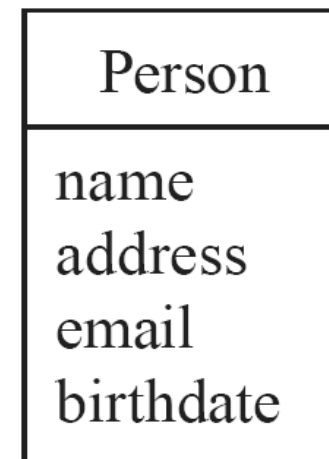
Analiza i validacija pridruživanja

- Izbjegavaj nepotrebna pridruživanja jedan – jedan.

- **Izbjegavati**

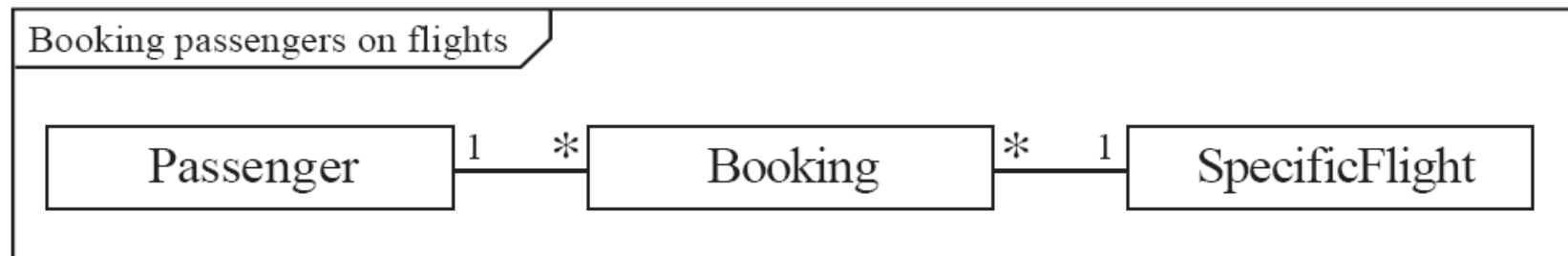


- **Učini ovo**



Primjer pridruživanja: rezervacija leta

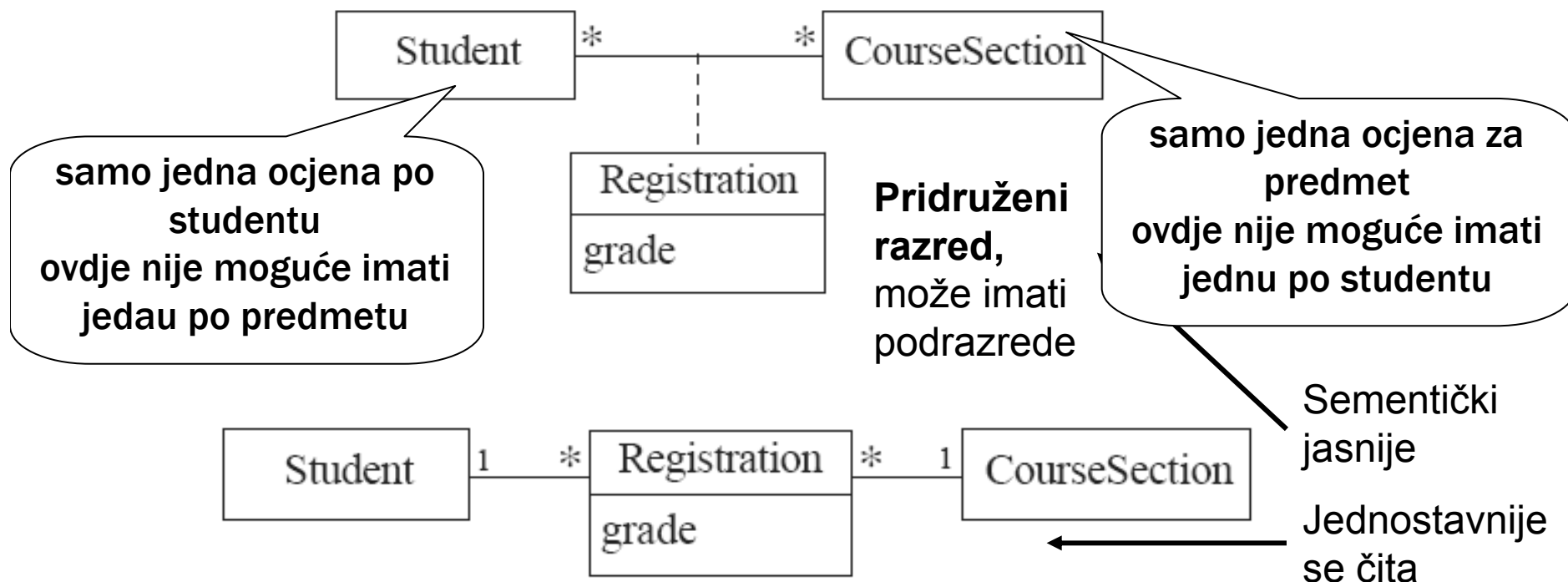
- Rezervacija je uvijek samo za jednog putnika.
 - Nema rezervacije za nula putnika.
 - Rezervacija *nikada* ne uključuje više od jednog putnika..
- Putnik može imati bilo koji broj rezervacija.
 - Putnik uopće ne mora imati neku rezervaciju.
 - Putnik može imati više od jednu rezervaciju.



- Okvir oko ovoga dijagrama je opcija koju predviđa UML 2.0. (nebitno za ova predavanja)

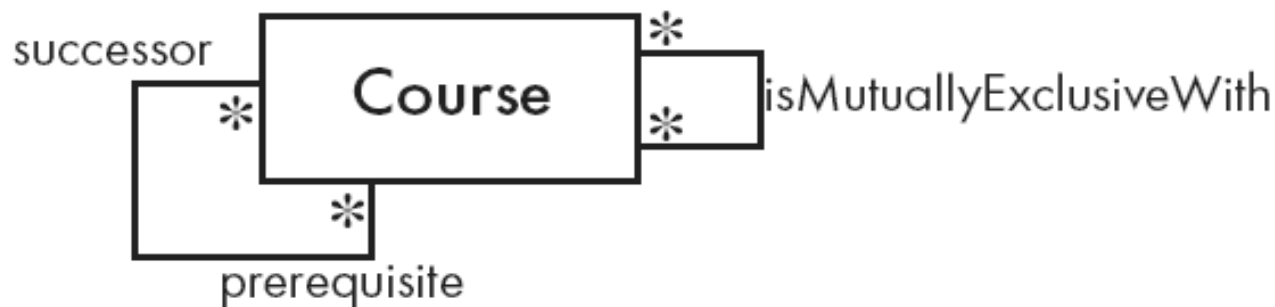
Pridruženi razredi

- Ponekad se atribut koji se tiče više razreda ne može smjestiti niti u jedan od navedenih razreda.
- Postoje dva ekvivalentna označavanja pridruženih razreda:



Refleksivno pridruživanje

- Moguće je da se pridruživanje spaja na isti razred.
- primjer:
 - predmet može imati druge predmete kao preduvjete
 - spriječiti upis vrlo sličnih predmeta



Smjer pridruživanja

- Pridruživanja su u osnovici *bidirekcijska*.
- Moguće je ograničiti smjer pridruživanja dodavanjem strelice na jednom kraju.



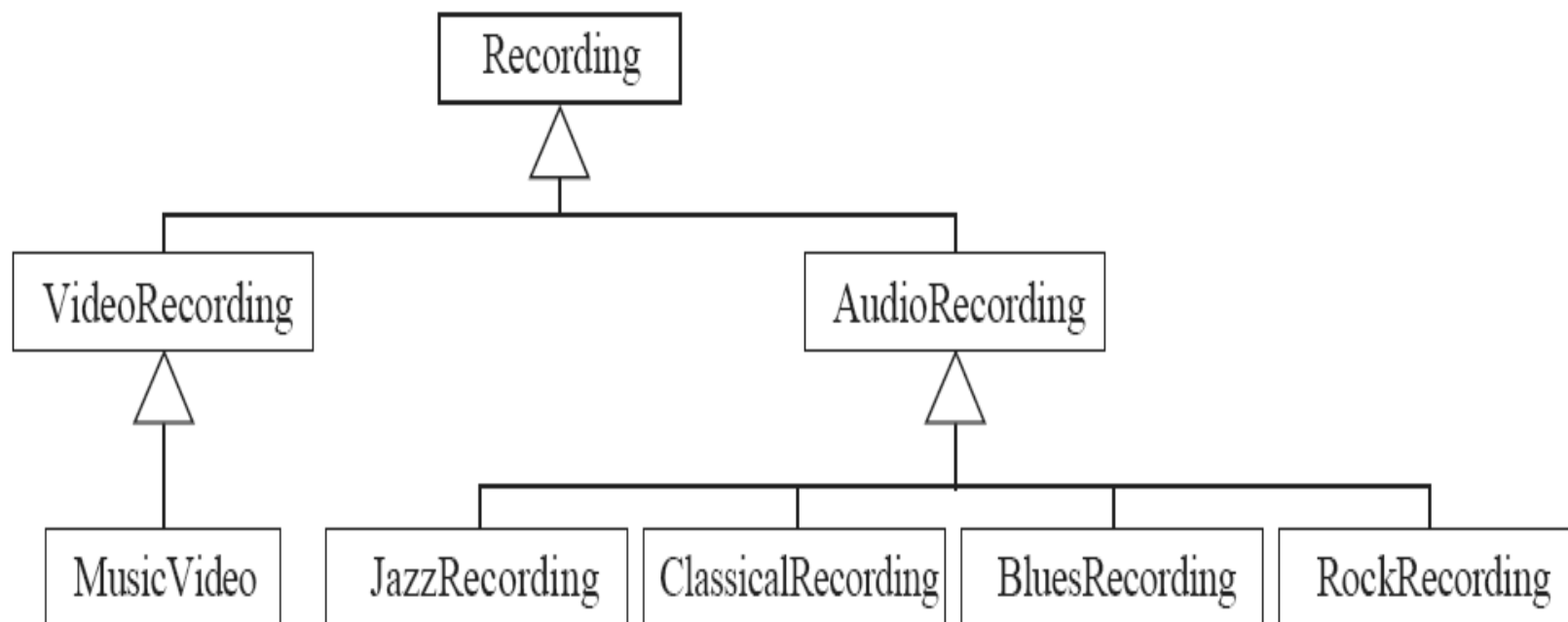
(za odabrani dan pogledaj bilješku, ali ne i obratno)

Generalizacija

- Specijalizacija superrazreda u jedan ili više podrazreda.
 - Generalizacijski skup (engl. *generalization set*)
 - označena grupa generalizacija s zajedničkim superrazredom.
 - Oznaka (katkad nazvana *diskriminator*)
 - opisuje kriterije za specijalizaciju.

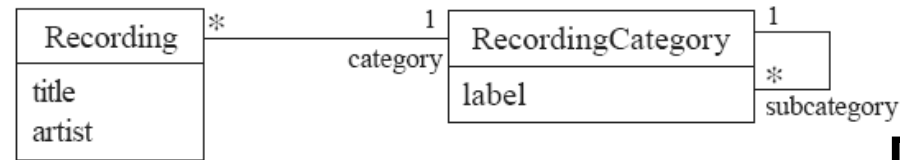


Izbjegavanje nepotrebne generalizacije



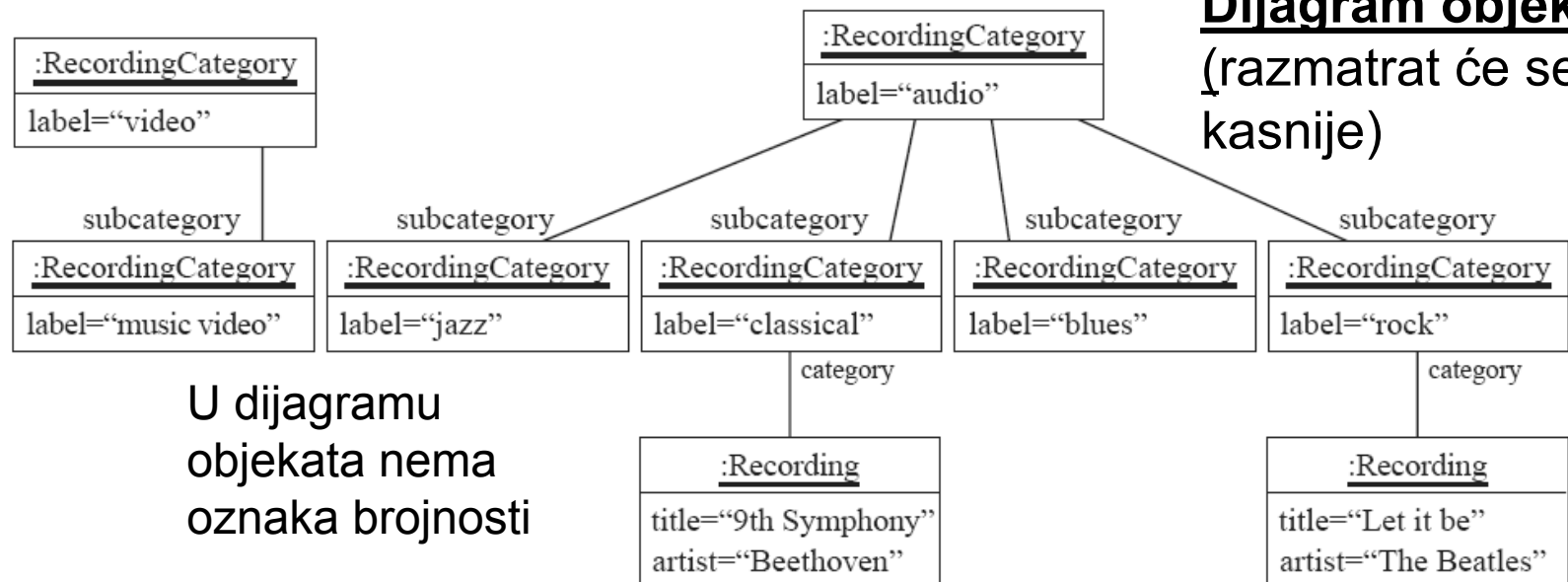
Nepogodna hijerarhija
razreda, Trebali bi biti
instance.

Izbjegavanje nepotrebne generalizacije



(a)

Dijagram razreda



(b)

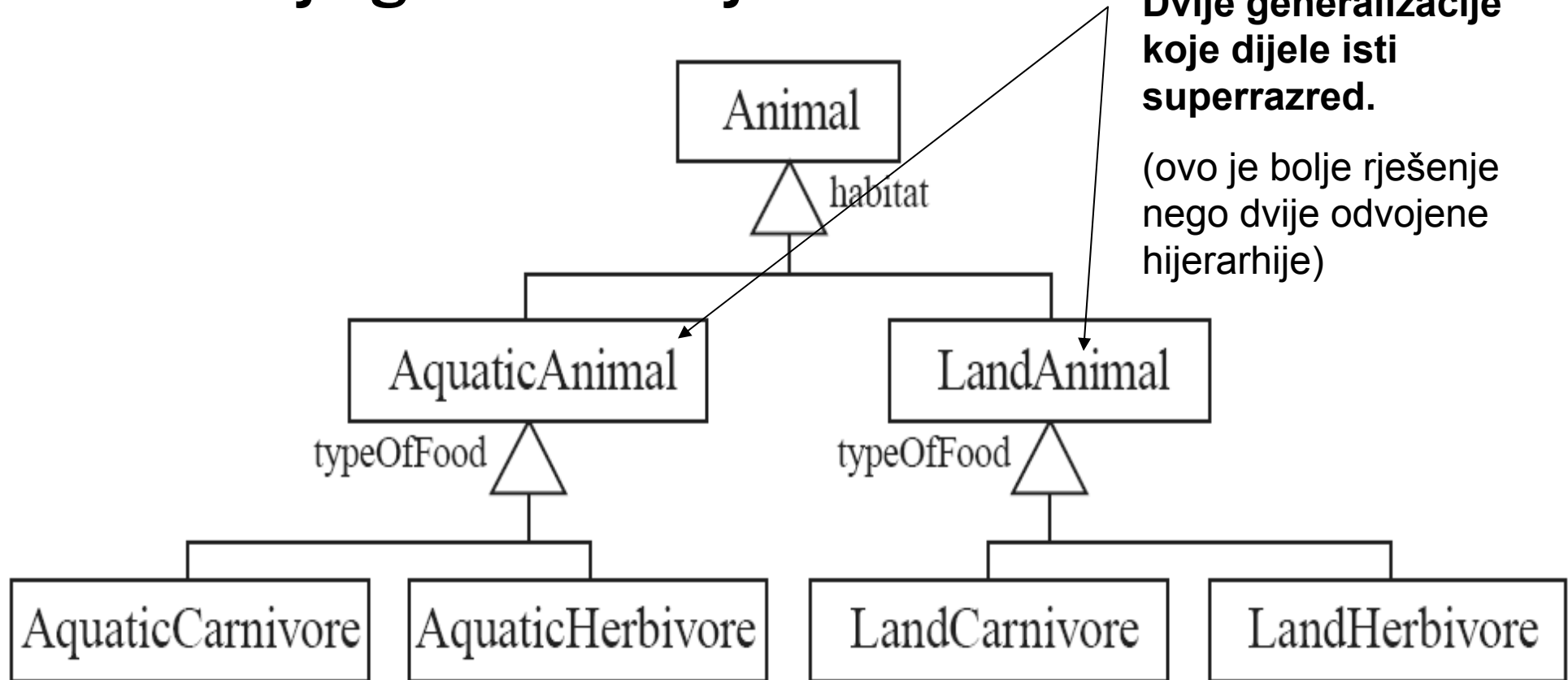
Dijagram objekata
(razmatrat će se kasnije)

U dijagramu objekata nema oznaka brojnosti

Slika pokazuje poboljšani dijagram razreda uz pridruženi dijagram objekata (instanci) Razredi iz prethodnog dijagrama su instance jednog razreda **RecordingCategory**

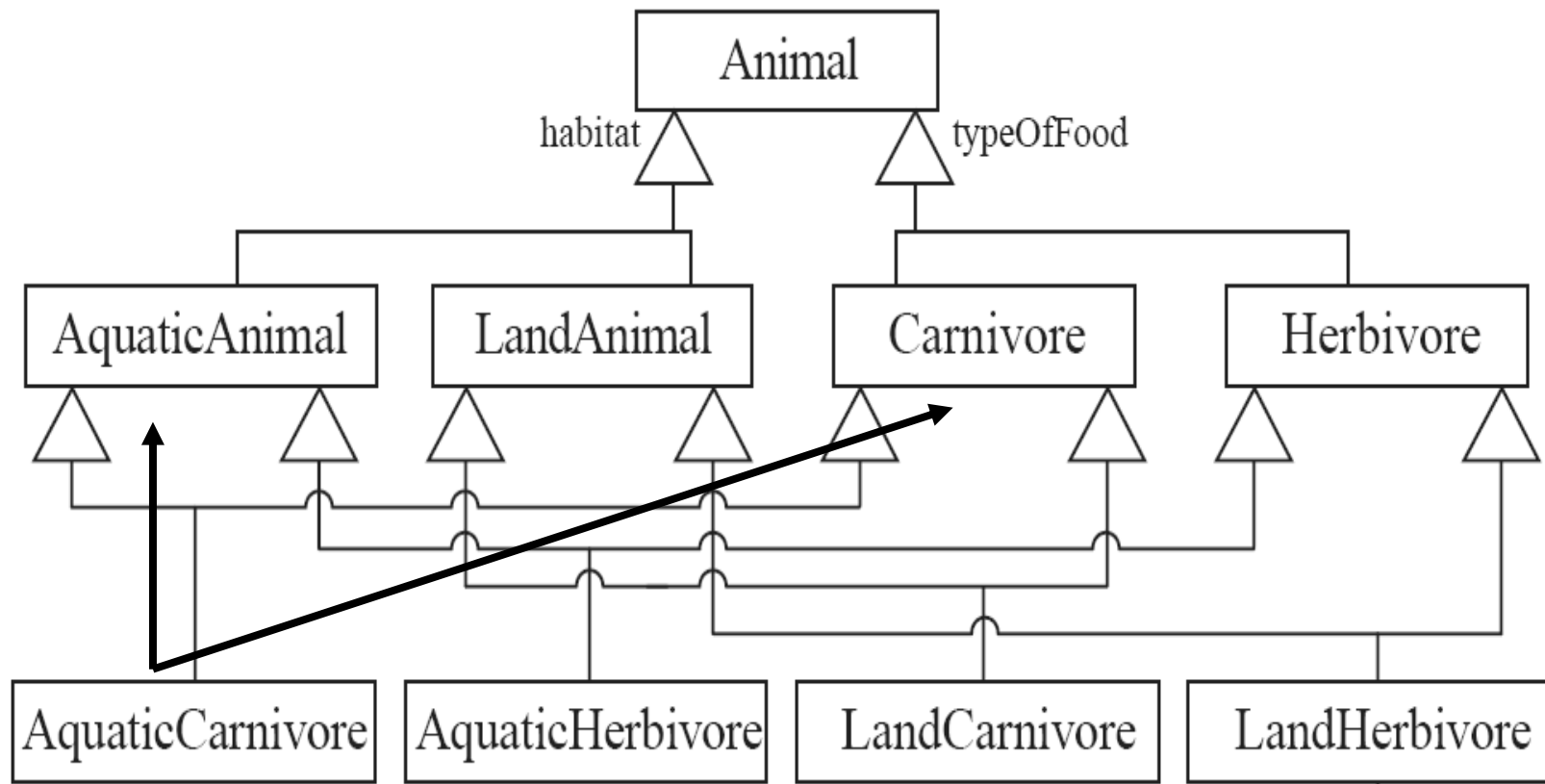
Rukovanje višestrukim diskriminatorima

■ Kreiranje generalizacije više razine.



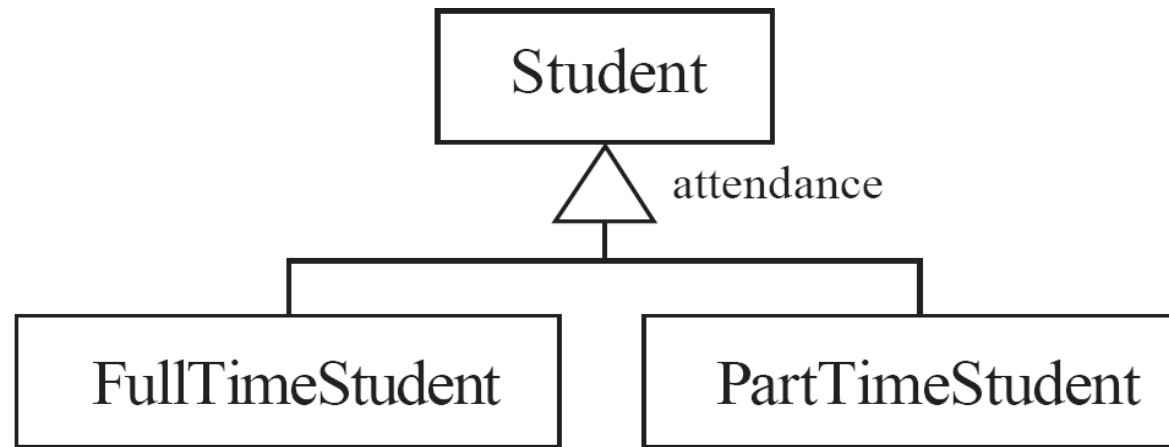
Rukovanje višestrukim diskriminatorima

- višestruko nasljeđivanje
 - ne u Javi



IZbjegavanje da instance moraju mijenjati razred

- Instanca nikad ne smije imati potrebu promjene razreda.

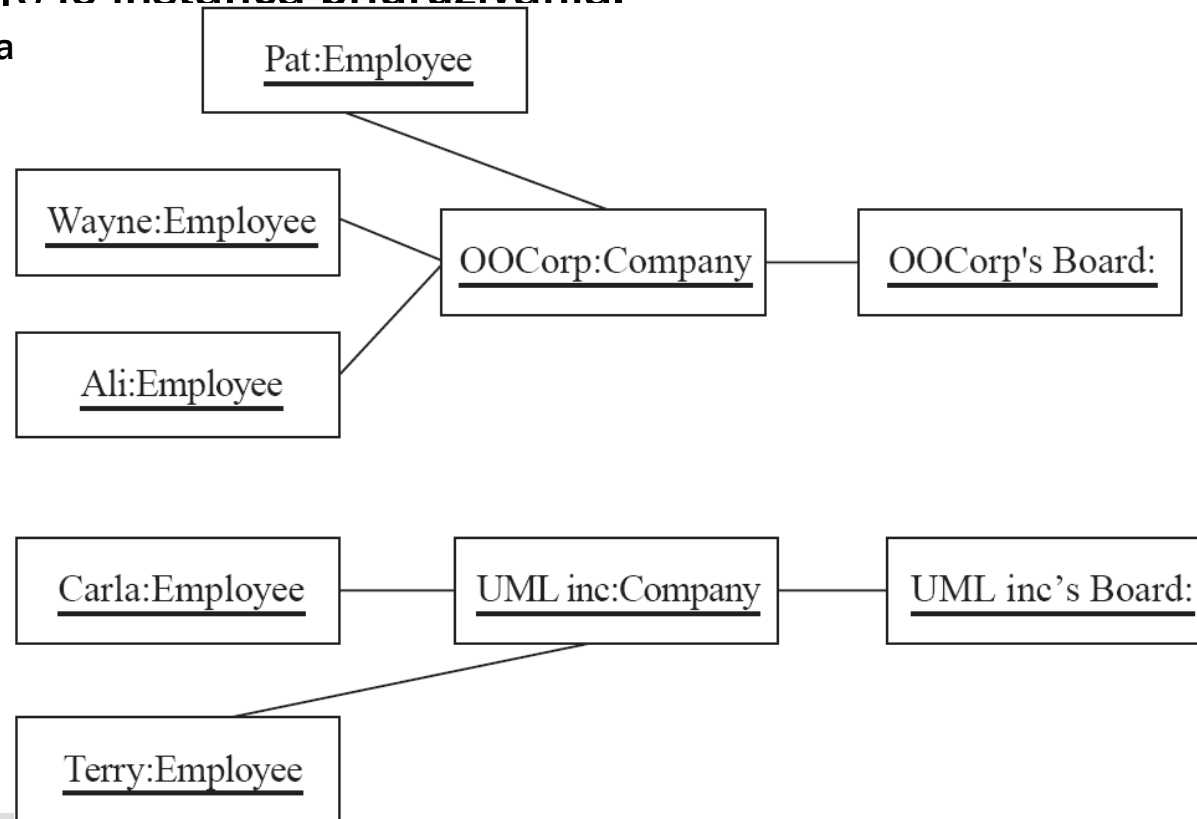


Studentov status se može promijeniti, pa taj objekt (instanca) mora promijeniti razred (traži destrukciju – kreaciju novog objekta).

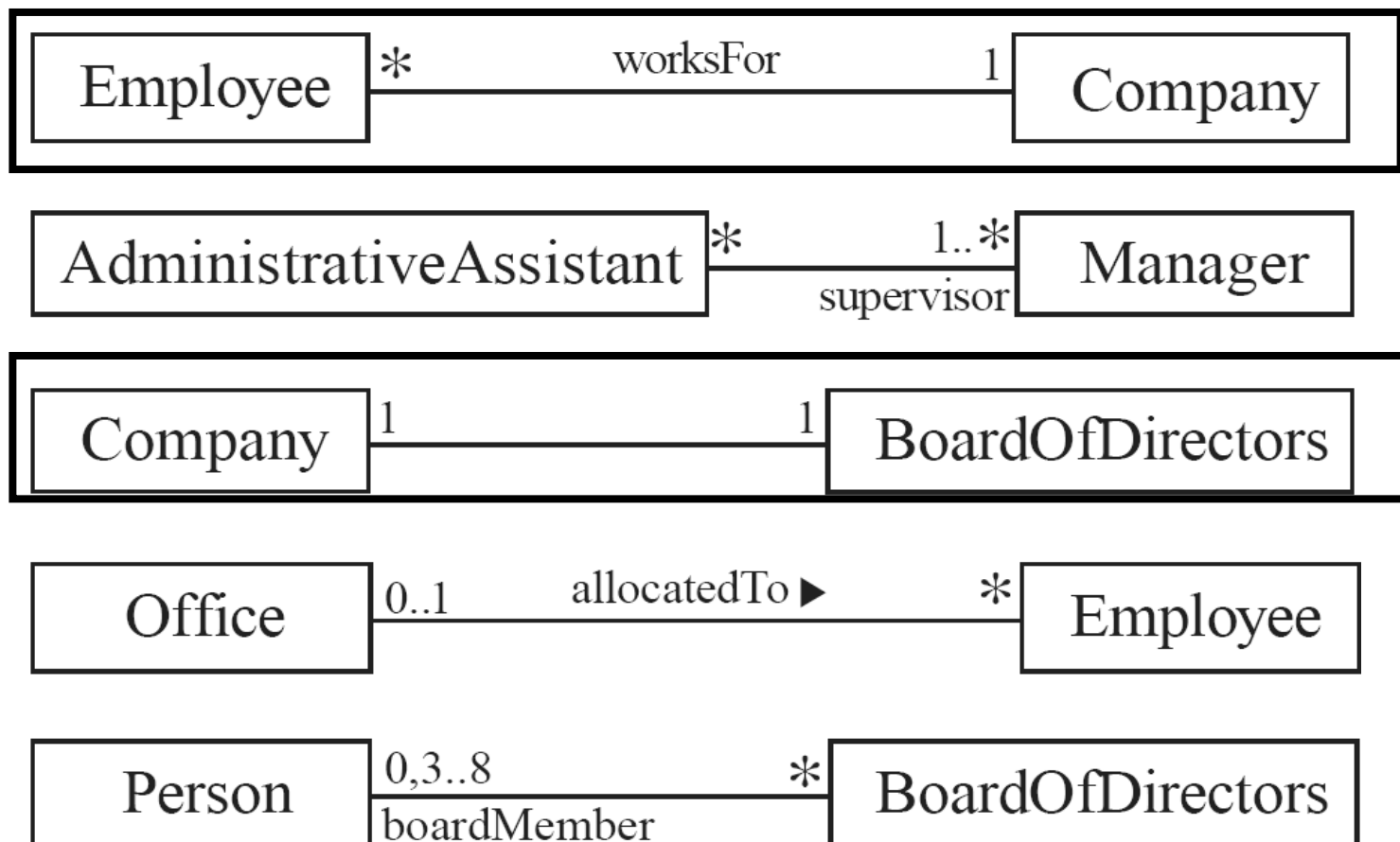
Jedno rješenje: uključi atribut **attendanceStatus** u razred **Student** te ne koristiti podrazrede (ali tako se gubi mogućnost polimorfizma - specifičnih metoda u različitim podrazredima).

Dijagram objekata

- engl. Object Diagrams
- Prikazuje instance i veze u jednom trenutku izvođenja sustava.
 - **Veza** (engl. link) je instanca pridruživanja.
 - Na isti način ka



Dijagrami objekata generiraju se iz dijagrama razreda

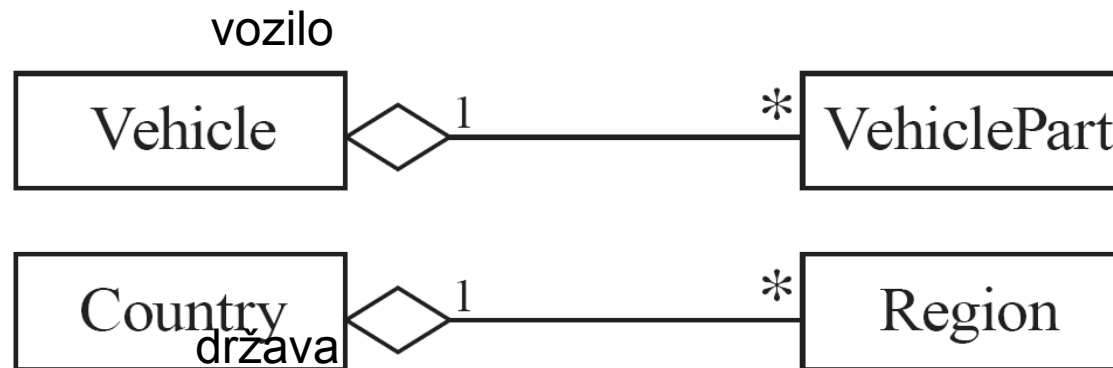


O pridruživanju i generalizaciji

- ***Pridruživanje*** (asocijacija) opisuje odnos između instanci koji će nastupiti za vrijeme izvođenja programa.
 - Kada se dijagram objekata generira iz dijagrama razreda, u dijagramu objekata postojat će instance oba razreda spojene pridruživanjem.
- ***Generalizacija*** opisuje odnos između razreda u dijagramu razreda.
 - Generalizacija se ne pojavljuje u dijagramu objekata.
 - Mora se uzeti u obzir da je jedna instanca nekog razreda ujedno i instanca svakog njegovog superrazreda.

Napredne značajke u dijagramu razreda:

- Agregacije su posebna vrsta pridruživanja koja predstavlja odnos “cjelina-dio”.
 - “Cijeli” dio se često naziva *agregat* (zbor, skup).
 - UML simbol označuje pridruživanje “ je dio od” (engl. `isPartOf`).



Kada koristiti agregaciju

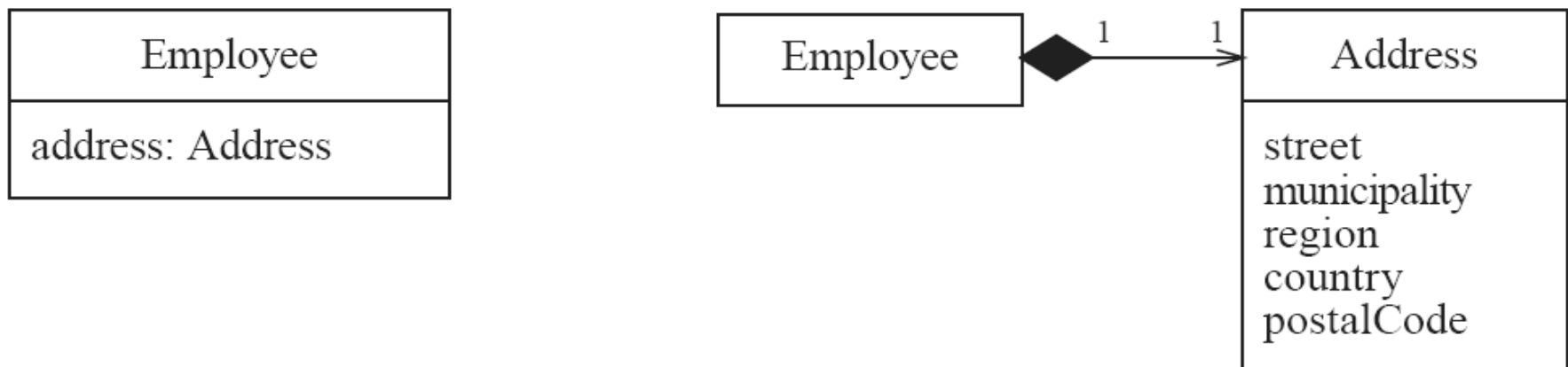
- Kao opće pravilo, agregacija se u pridruživanju koristi kada:
 - Može se tvrditi:
 - Dijelovi su dio agregata.
 - ili agregat je sastavljen od dijelova.
 - Kada je netko ili nešto vlasnik agregata (ili upravlja njime) tada je ujedno i vlasnik (upravlja) njegovim dijelovima.

Kompozicija

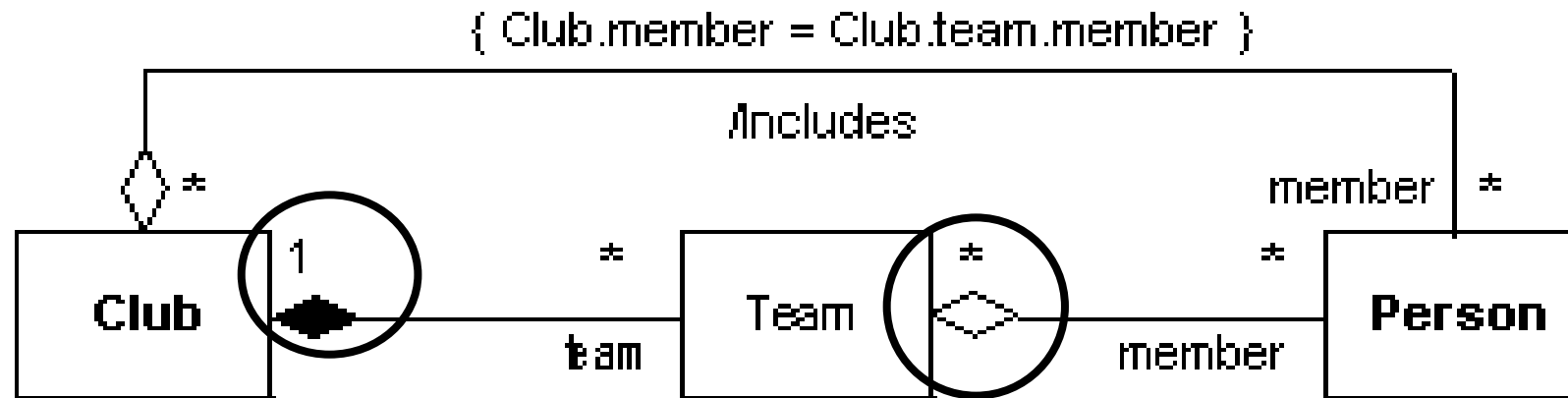
- Kompozicija je jaki tip agregacije.
 - Ako se agregat uništi, tada se uništavaju i njegovi dijelovi.



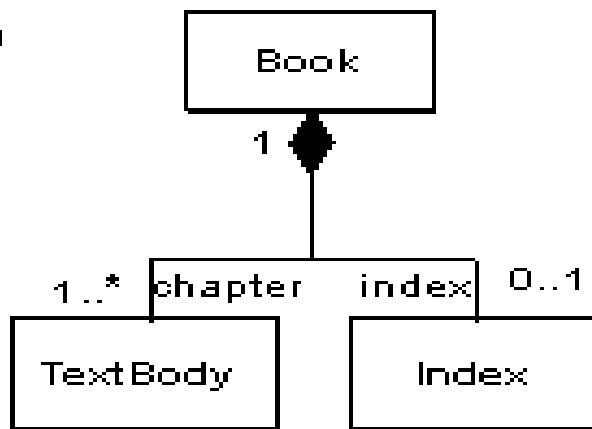
- Alternativni načini modeliranja adresa:



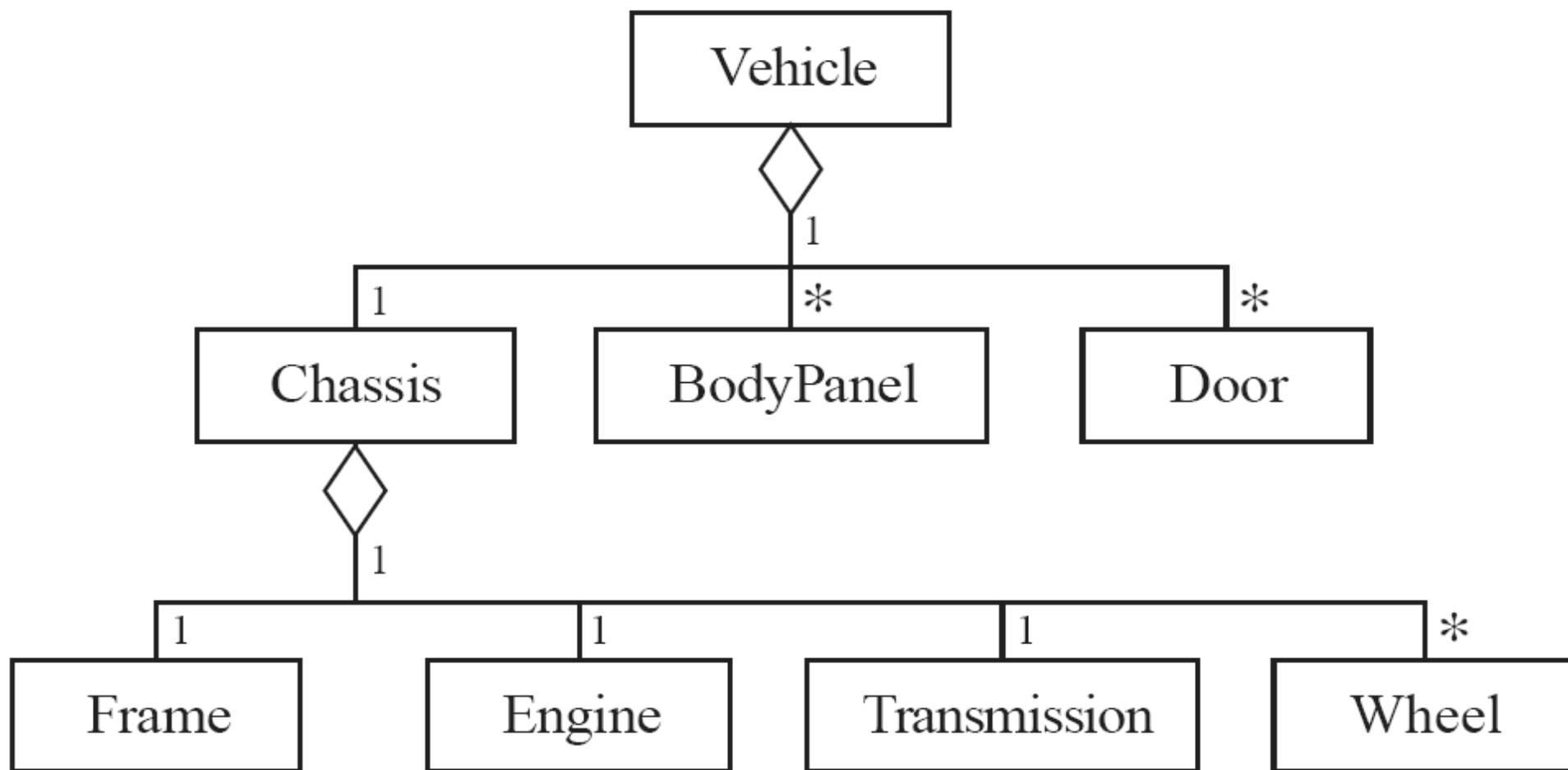
Primjeri agregacije i kompozicije



(a)

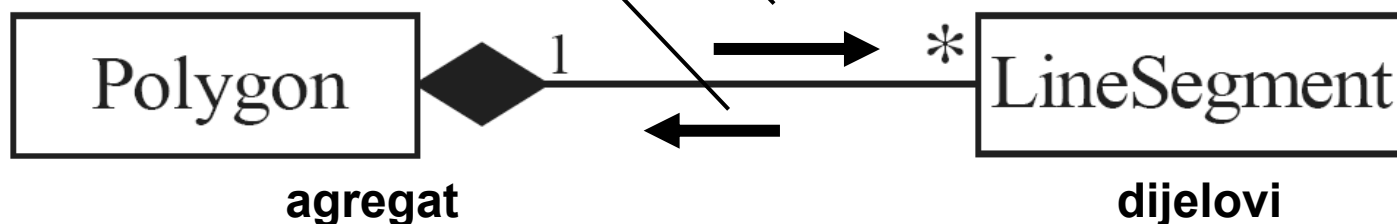


Agregacijska hijerarhija



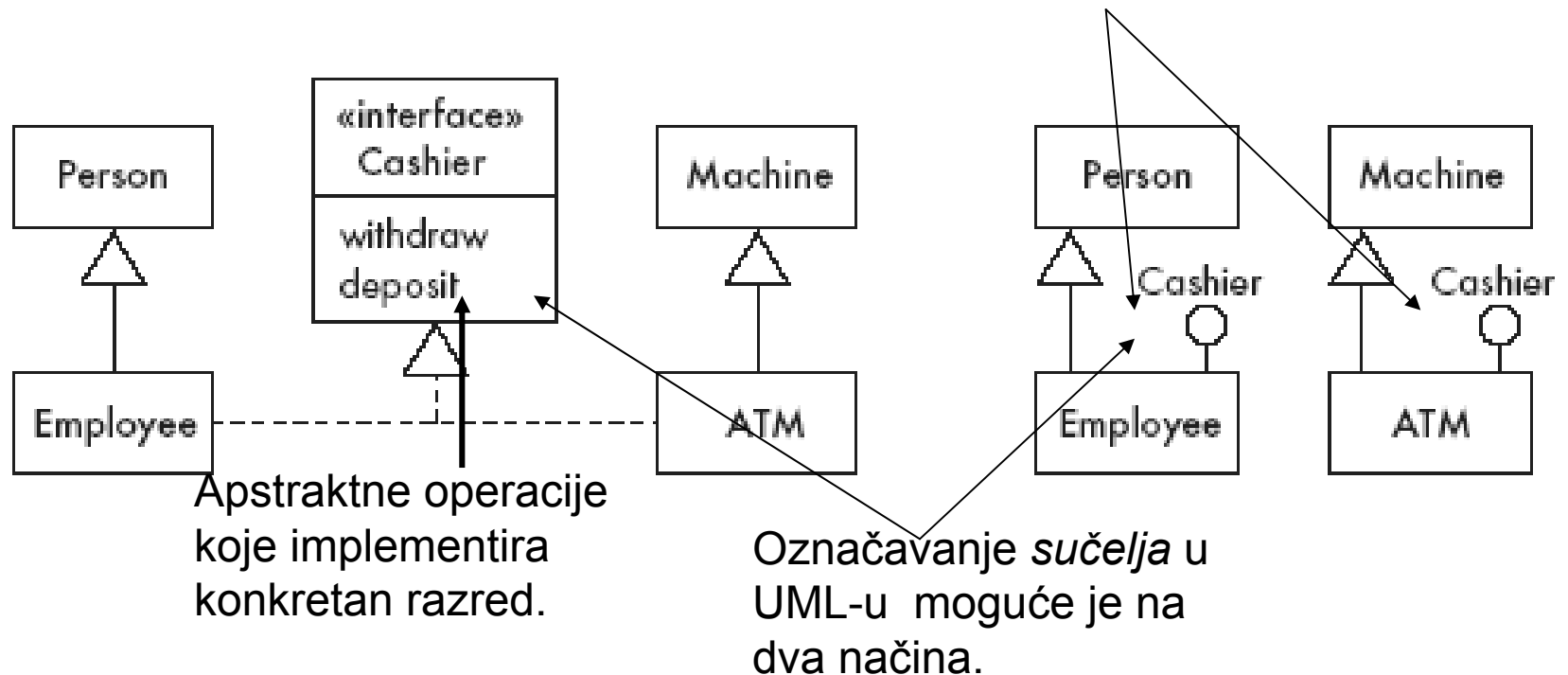
Propagacija u agregaciji

- Propagacija je mehanizam kojim se implementira operacija u agregatu tako da djeluje na njegove dijelove.
- U isto vrijeme, značajke dijelova propagiraju natrag prema agregatu.
- Propagacija u agregaciji predstavlja isto što i nasljeđivanje u generalizaciji.
 - Temeljna razlika:
 - Nasljeđivanje je implicitan mehanizam.
 - Propagacija se mora programirati kada je potrebna.



Sučelja

- engl. *Interfaces*
- Sučelje opisuje *dio vidljivog ponašanja* skupa objekata.
 - Jedno *sučelje* slično je razredu, osim što mu nedostaju varijable instanci i implementacija metoda. To je lista apstraktnih operacija. *Sučelje* može biti tip varijable, te u tom slučaju u varijablu se može staviti bilo koja instanca nekog konkretnog razreda koja implementira to sučelje. Moguće je i dinamičko povezivanje.



Bilješke i opisni tekst

- **Opisni tekst i drugi dijagrami**
 - Uključi dijagrame u veći dokument (vidi projekt u okviru predmeta Oblikovanje programske potpore).
 - Tekst može objasniti bilo koji aspekt sustava uporabom sasvim slobodne (nestandardizirane) notacije.
 - Ističe i proširuje opis važnih značajki sustava, te navodi argumente u donošenju odluka oblikovanja.
- **Bilješke (engl. Notes):**
 - Bilješka je mali blok teksta uključen u UML dijagram.
 - Ima istu ulogu kao i komentar u programskom jeziku.

Proces razvoja dijagrama razreda

- UML modeli mogu se kreirati u različitim fazama oblikovanja programske potpore. s različitim ciljem (svrhom) i s različitom razinom detalja.
 - Istraživački (engl. exploratory) model domene primjene:
 - malo detalja
 - Razvija se tijekom analize domene s ciljem razumijevanja te domene.
 - **Sistemska model domene:**
 - više detalja
 - Modelira aspekt domene koji će biti predstavljen programskim sustavom.
 - **Model sustava:**
 - najviše detalja
 - Uključuje razrede objekata potrebnih u izgradnji arhitekture sustava i korisničkog sučelja.

Modeli i razine detalja

Type of model	Elementi koji predstavljaju domenu <i>Contains elements that represent things in the domain</i>	Elementi domene koji će stvarno biti implementirani <i>Models only things that will actually be implemented</i>	Elementi (ne domene) potrebni za izgradnju sustava <i>Contains elements that do not represent things in the domain, but are needed to build a complete system</i>
Exploratory domain model: developed in domain analysis to learn about the domain	Yes	No	No
System domain model: models those aspects of the domain represented by the system	Yes	Yes	No
System model: includes classes used to build the user interface and system architecture	Yes	Yes	Yes

Sistemiški model domene i model sustava

- **Sistemiški model domene može izostaviti razrede potrebne u izgradnji cjelovitog programskog sustava.**
 - Može sadržavati manje od polovice od ukupnog broja razreda u sustavu.
 - Izgrađuje se i koristi neovisno o skupu
 - razreda koji modeliraju korisničko sučelje
 - razreda koji modeliraju arhitekturu sustava
- **Cjeloviti model sustava sadrži**
 - Sistemiški model domene
 - Razrede koji modeliraju korisničko sučelje
 - Razrede koji modeliraju arhitekturu sustava
 - Pomoćne razrede

Preporučena sekvenca aktivnosti

1. Identificiraj početni skup kandidata za razrede.
2. Dodaj pridruživanja (engl. Associations) i attribute.
3. Pronađi generalizacije.
4. Izlistaj temeljne odgovornosti (engl. Responsibilities) svakog razreda.
5. Odluči se za specifične operacije.
6. Iteriraj proces dok ne dobiješ zadovoljavajući model
 - Dodaj ili izbriši razrede, pridruživanja, attribute, generalizacije, odgovornosti ili operacije.
 - Identificiraj razrede sučelja.

Odgovornost je nešto što sustav mora obaviti. To je viša razina apstrakcije ponašanja od operacije.

Hijerarhija ponašanja: *odgovornost* <- *operacija* <- *metoda*.

Identificiraj razrede

- Tijekom razvoja modela domene otkrij razrede.
- Kada si usredotočen na korisničko sučelje ili na arhitekturu sustava razredi se osmišljavaju (engl. Invent) kako bi se riješio određen problem u oblikovanju.
 - Osmišljavanje je dopustivo i tijekom razvoja modela domene).
- Obrati pažnju na mogućnost ponovnu uporabu razreda (engl. Reuse).

Imenika otkrivanja razreda u domeni primjene

- **Pogledaj u izvorne dokumente opisa zahtjeva.**
- **Izdvoji imenice i imeničke izraze.**
- **Eliminiraj imenice koje:**
 - **su redundantne**
 - **predstavljaju instance**
 - **nejasne su i vrlo općenite**
 - **nepotrebne u primjeni**
- **Obrati pažnju na razrede u domeni koji predstavljaju tipove korisnika ili druge aktore.**

Primjer:

- Označavanja razreda: dobri, loši i razredi za koje nismo sigurni.
 - Sustav osigurava temeljne usluge za rukovanje bankovnom računima u banci koja se zove OOBank. Ta banka ima više poslovnica, koje imaju svoje adrese i oznaku poslovnice. Pojedini klijent otvara račun u poslovnici banke.

Identificiranje pridruživanja i atributa

- Započni s razredima koji su središnji i najvažniji.
- Odluči o jasnim i očiglednim podacima koje ti razredi moraju sadržavati, te o njihovim odnosima s drugim razredima.
- Proširuj dijagram iznutra prema van do razreda koji su manje važni.
- Izbjegavaj dodavanje mnogo pridruživanja i atributa u pojedinom razredu.
 - Sustav je jednostavniji ako rukuje s manje informacija.

Savjeti o identitificiranju i specitificiranju valjanih pridruživanja između razreda

■ Pridruživanje postoji ako razred:

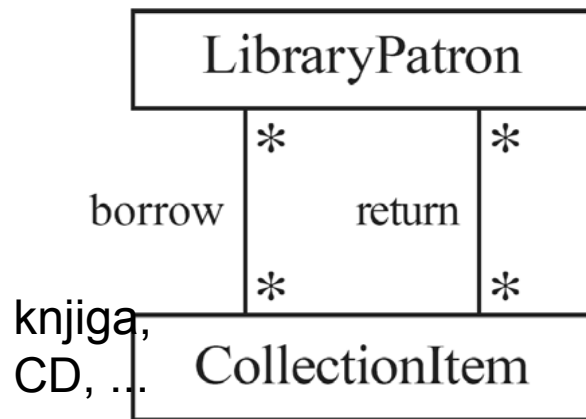
- *posjeduje (ili ovladava)*
- *upravlja*
- *spojen je s*
- *odnosi se prema (engl. is related to)*
- *dio je od*
- *ima dijelove*
- *član je*
- *ima članove*

neke druge razrede u modelu.

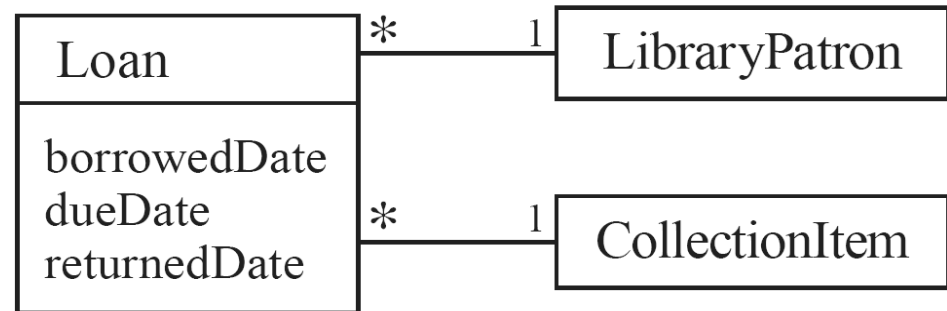
- Specificiraj brojnost na oba kraja pridruživanja.
- Jasno označi pridruživanje.

Akcije i pridruživanja

- Česta pogreška je predstaviti akcije kao pridruživanja.



Loše, akcije su predstavljene
kao pridruživanja



Bolje, operacija **borrow** kreira **Loan**.
Operacija **return** postavlja returnedDate
atribut.

Obrati pažnju da su **borrow** i **return**
operacije.

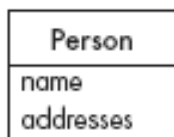
Identificiranje atributa

- Potraži informacije koje svaki razred mora podržavati.
- Neke imenice koje su odbačene kao razredi, sada mogu biti atributi.
- Atribut bi općenito trebao sadržavati jednostavnu vrijednost
 - Npr. string, broj

Savjeti o identificiranju i specificiranju valjanih atributa

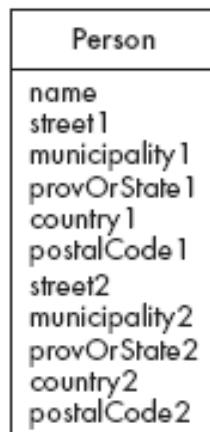
- Nije dobro imati mnogo kopija atributa.
- Ako neki podskup atributa u pojedinom razredu čini koherentnu grupu kreiraj poseban razred koji sadrži te attribute.

Loše,
atribut u
množini.



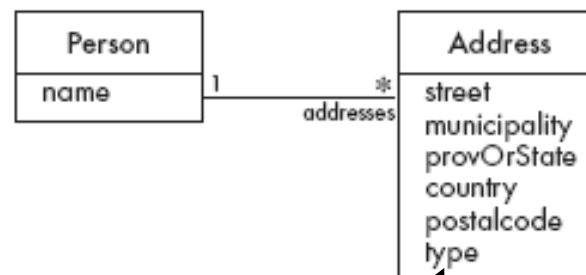
Bad, due to
a plural attribute

Loše,
atribut u
množini.



Bad, due to too many
attributes, and the
inability to add more
addresses

Previše atributa.



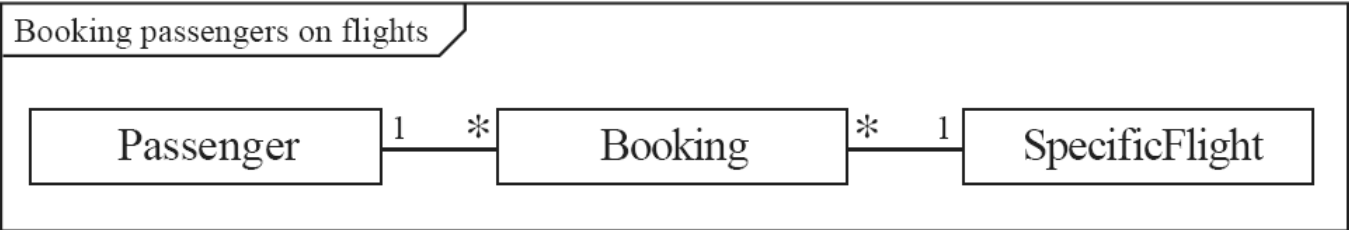
Good solution. The type indicates whether it
is a home address, business address etc.

Dobro rješenje. Tip označuje da li je
to adresa stanovanja ili posla.

Primjer: Sustav rezervacije avio-leta

- The reservation system keeps track of passengers who will be flying in specific seats on various flights, as well as people who will form the crew.
- For the crew the system needs to track what everyone does and who supervises whom.
- *****
- Postupak identificiranja razreda:
- Imenice na početnoj listi razreda: `Flight`, `Passenger`, `Employee`
- Ostale imenice:
- “reservation system” – nije razred, već dio sustava
- “seat” – atribut razreda `Flight`
- “crew” – `Employee` je mnogo fleksibilnije

Primjer: Sustav rezervacije avio-leta

- Flight – središnji razred (sadrži date, time, flightNumber).
- Letovi koji polijeću dnevno u isto vrijeme imaju isti broj leta (flight number).
- Podijeli Flight u RegularFlight (sadrži time and flight number) i SpecificFlight (odlazi na određen dan).
- Pridruž  nogo.
- Putnici:

Primjer: Sustav rezervacije avio-leta

■ atributi i pridruživanja



Identificiranje generalizacija i sučelja

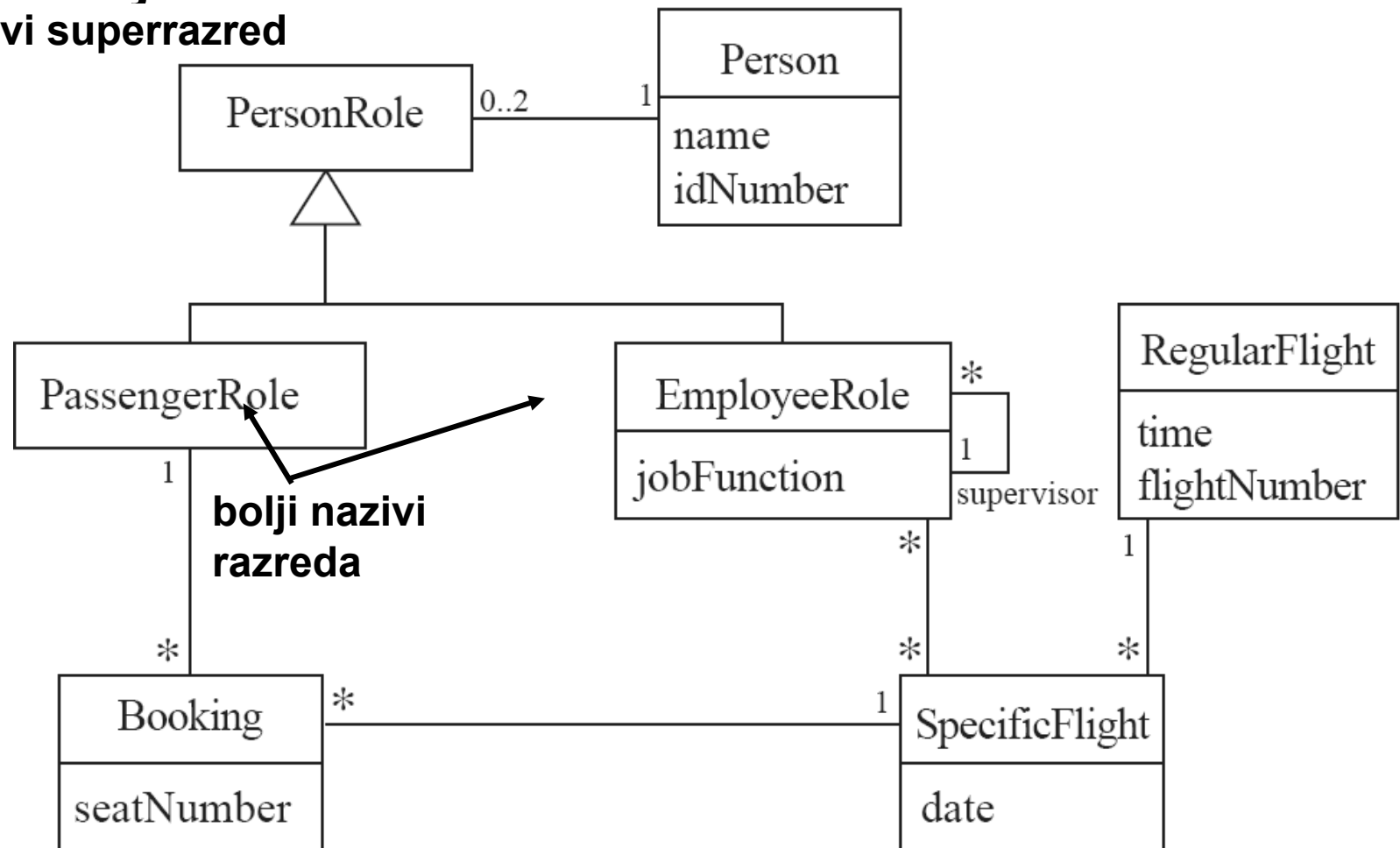
- Postoje dva načina identificiranja generalizacija:
 - Odozdo prema gore
 - Grupiraj slične razrede i kreiraj novi superrazred
 - Odozgo prema dolje
 - Najprije potraži općenitije razrede pa ih specijaliziraj ako je potrebno.
- Kreiraj sučelje (engl. Interface) umjesto superrazreda ako:
 - Razredi su vrlo različiti osim nekoliko zajedničkih operacija.
 - Jedan ili više razreda već imaju svoj superrazred (u Javi nema višestrukog nasljeđivanja).
 - Želi se ograničiti operacije s varijablom (koja je deklarirana da sadrži instance više razreda) samo na

Primjer: Sustav rezervacije avio-leta

- Razredi `Passenger` i `Employee` dijele zajedničku informaciju.
- Definiranjem superrazreda `Person` nije dovoljno dobro jer jedna osoba (`person`) može biti oboje: i putnik (`passanger`) i zaposlenik (`employee`).
- Instance može nastati i postojati samo od jednog razreda !!
- Stoga, je uveden razred `PersonRole` i pridruživanja koje dozvoljava da svaka osoba može imati dvije uloge.

Primjer: Sustav rezervacije avio-leta

- **generalizacija**
novi superrazred



Alociranje odgovornosti razredima

- **Odgovornost** (engl. responsibility) je nešto što sustav mora izvršiti.
 - Svaki funkcionalni zahtjev mora se pripisati nekom razredu.
 - Sve odgovornosti jednog razreda moraju biti *jasno povezane*.
 - Ako jedan razred ima previše odgovornosti, razmotri razmotri *podjelu* toga razreda u različite razrede.
 - Ako razred nema odgovornosti, tada je vjerojatno *beskoristan*.
 - Ako se neka odgovornost ne može pripisati niti jednom od postojećih razreda, mora se kreirati *novi* razred.
- **Kako bi se odredile odgovornosti:**
 - Analiziraj obrasce uporabe (engl. use case).
 - U opisu sustava potraži glagole i imenice koje opisuju akcije.

Kategorije odgovornosti

- Postavljanje i dohvaćanje vrijednosti atributa.
- Kreiranje i inicijalizacija novih instanci.
- Upis i čitanje iz trajne pohrane podataka.
- Dokidanje (uništavanje) instanci.
- Dodavanje i brisanje veza pridruživanja između instanci.
- Kopiranje, konverzija, preslikavanje, prijenos, izlaz podataka.
- Izračunavanje numeričkih vrijednosti.
- Navigacija i pretraživanje.
- Drugi specijalizirani poslovi.

Primjer: Sustav rezervacije avio-leta

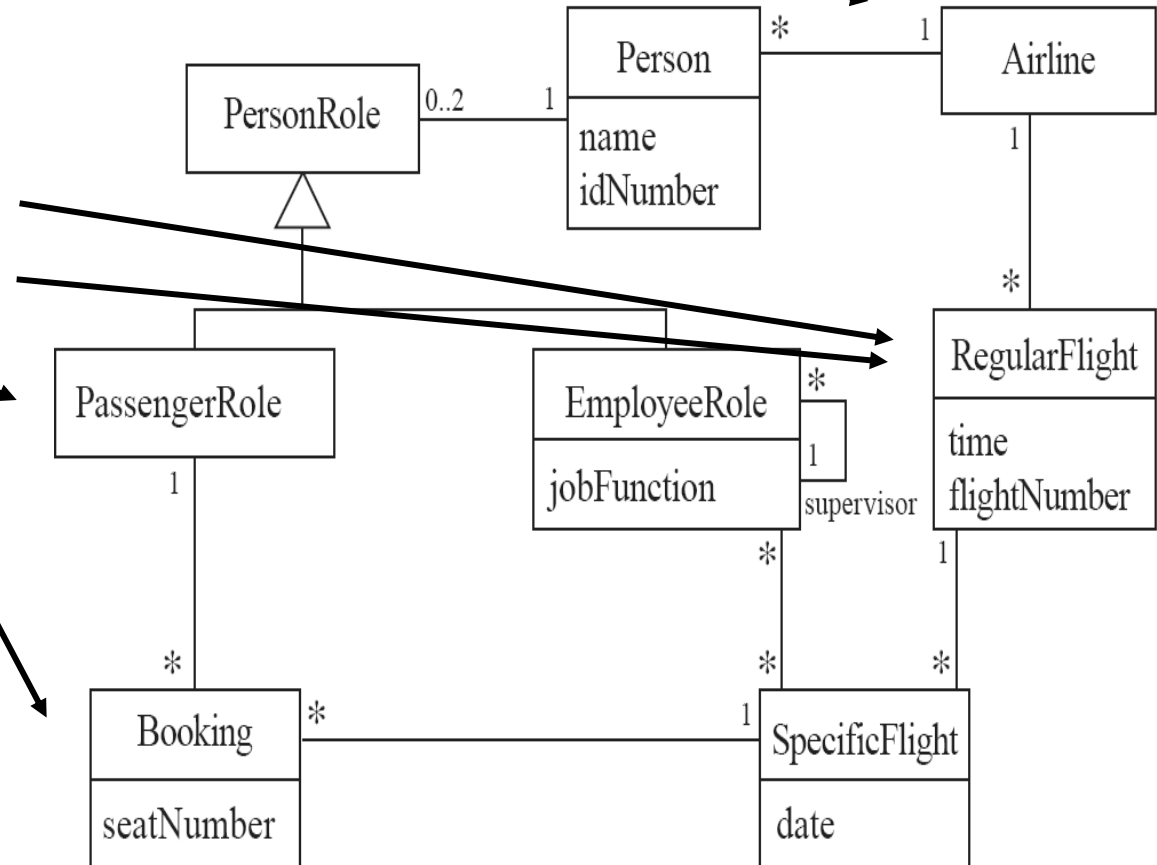
■ odgovornosti

- Kreiranje novog regularnog leta
- Pronalaženje leta
- Modificiranje atributa leta
- Kreiranje specifičnog leta
- Rezervacija za putnika
- Otkazivanje rezervacije

Može se pripisati u Booking, ali Booking još nije kreiran u trenutku kad se ova odgovornost inicijalizira

Pripisi odgovornost novom razredu

Novi razred



Objašnjenja alociranja odgovornosti:

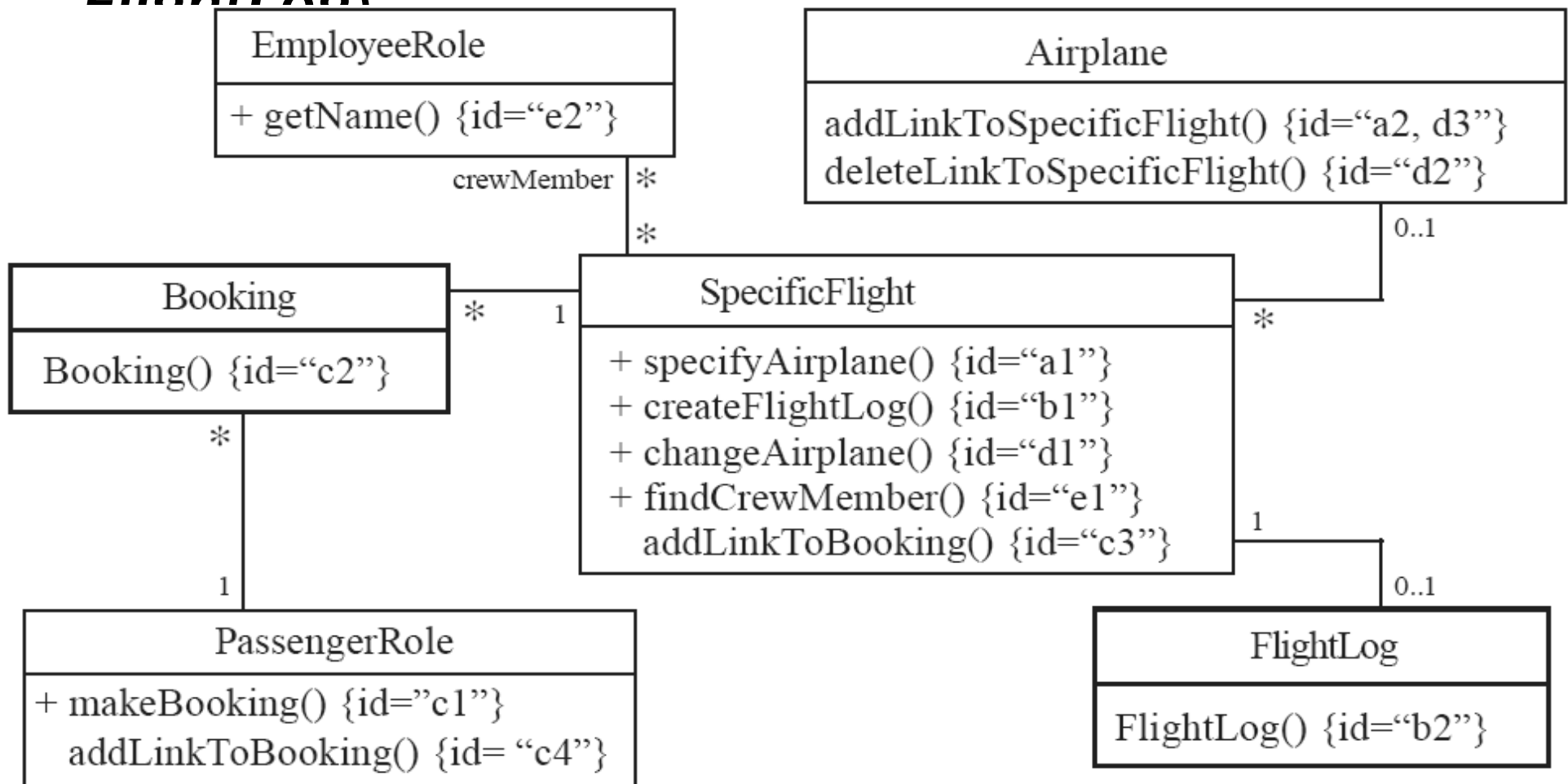
- Kreiranje **RegularFlight** prepušta se objektu. Stoga se uvodi novi razred **Airline**. Vjerojatno će postojati samo jedan objekt toga novoga razreda..
- U pronalaženju određenog **RegularFlight**, odgovornost za održavanje kolekcije objekata **RegularFlight** prepušta se također razredu **Airline**.
- Modificiranje atributa u **RegularFlight** je odgovornost tog istog razreda **RegularFlight**.
- Kreiranje **SpecificFlight** može biti odgovornost razreda **RegularFlight**, jer **RegularFlight** postoji kada se ta odgovornost inicira.
- Otkazivanje leta **SpecificFlight** može biti

Identificiranje operacija i metoda

- Hijerarhija ponašanja:
- odgovornost \Leftarrow operacije \Leftarrow metode.
- Operacije realiziraju odgovornosti pojedinog razreda i implementiraju se metodama.
 - Može postojati nekoliko operacija i metoda koje realiziraju jednu odgovornost.
 - Temeljne operacija (metode) koje implementiraju neku odgovornost normalno se deklariraju kao **public**.
 - Druge operacija (metode) koje surađuju u realizaciji odgovornosti trebaju biti što je moguće više deklarirane kao **private**.

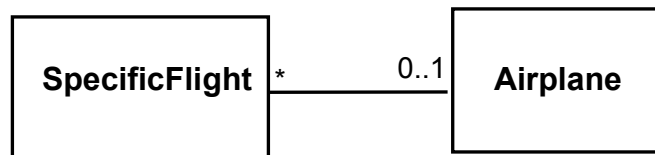
Primjer

- realizacija odgovornosti koje su pripisane razredu *SpecificFlight* (dodani su i novi razredi *Airplane* i *FlightLog*)



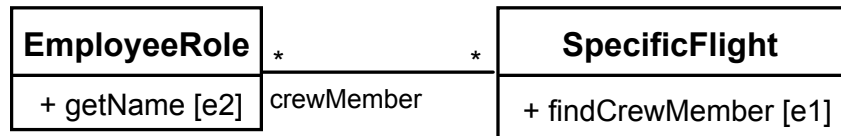
Kolaboracija razreda u realizaciji odgovornosti

- Temeljni zadatak: *Povezivanje dva postojeća objekta.*
- To je u objektnom dijagramu kreiranje veze (engl. link), a u implementaciji definiranje varijable u koju su upisani objekti.
- *Npr.: dodavanje bidirekcijske veze između instance od `SpecificFlight` i instance od `Airplane` (na ranijoj slici to su operacije označene “a”).*
- **1. (public) Instanca od `SpecificFlight`**
 - izgradi jednosmjernu vezu prema instanci od `Airplane`
 - zatim zove operaciju 2.
- **2. (non-public) Instanca od `Airplane`**
 - izgradi jednosmjernu vezu natrag do instance od `SpecificFlight`.



Kolaboracija razreda u realizaciji odgovornosti

- **Primjer jedne odgovornosti pripisane razredu** `SpecificFlight`:
- **Odgovornost: Pretraživanje i pronalaženje po imenima članova posade za određeni `SpecificFlight`:**
- **1. (public) Instanca od `SpecificFlight`**
 - kreira upit koji iterira preko svih veza tipa *crewMember* razreda `SpecificFlight`,
 - za svaku vezu zove operaciju 2 dok se ne pronađe podudarnost.
- **(može biti public) Instanca od `EmployeeRole` vraća ime člana posade.**



- **Na ranijoj slici to su metode označene “e”.**

Poteškoce i rizici u kreiranju dijagrama razreda

- Modeliranje je posebno teška vještina.
 - Čak i izvrsni programeri imaju poteškoća razmišljati na odgovarajućoj razini apstrakcije.
 - Obrazovanje se tradicijski više fokusira na programiranje nego modeliranje.
- Rješenje:
 - Osiguraj da članovi tima imaju adekvatno obrazovanje.
 - Imaj u timu jedni ili više iskusnih osoba za modeliranje.
 - Temeljito recenziraj sve modele.

Oblikovanje programske potpore

UML dijagrami

Prof.dr.sc. Vlado Sruk



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



Tema

- **Podsjetnik UML-a**
 - obrasci uporabe
 - sekvencijski dijagrami
- **UML dijagrami interakcija**
 - Dijagram kolaboracije
- **UML dijagrami stanja**

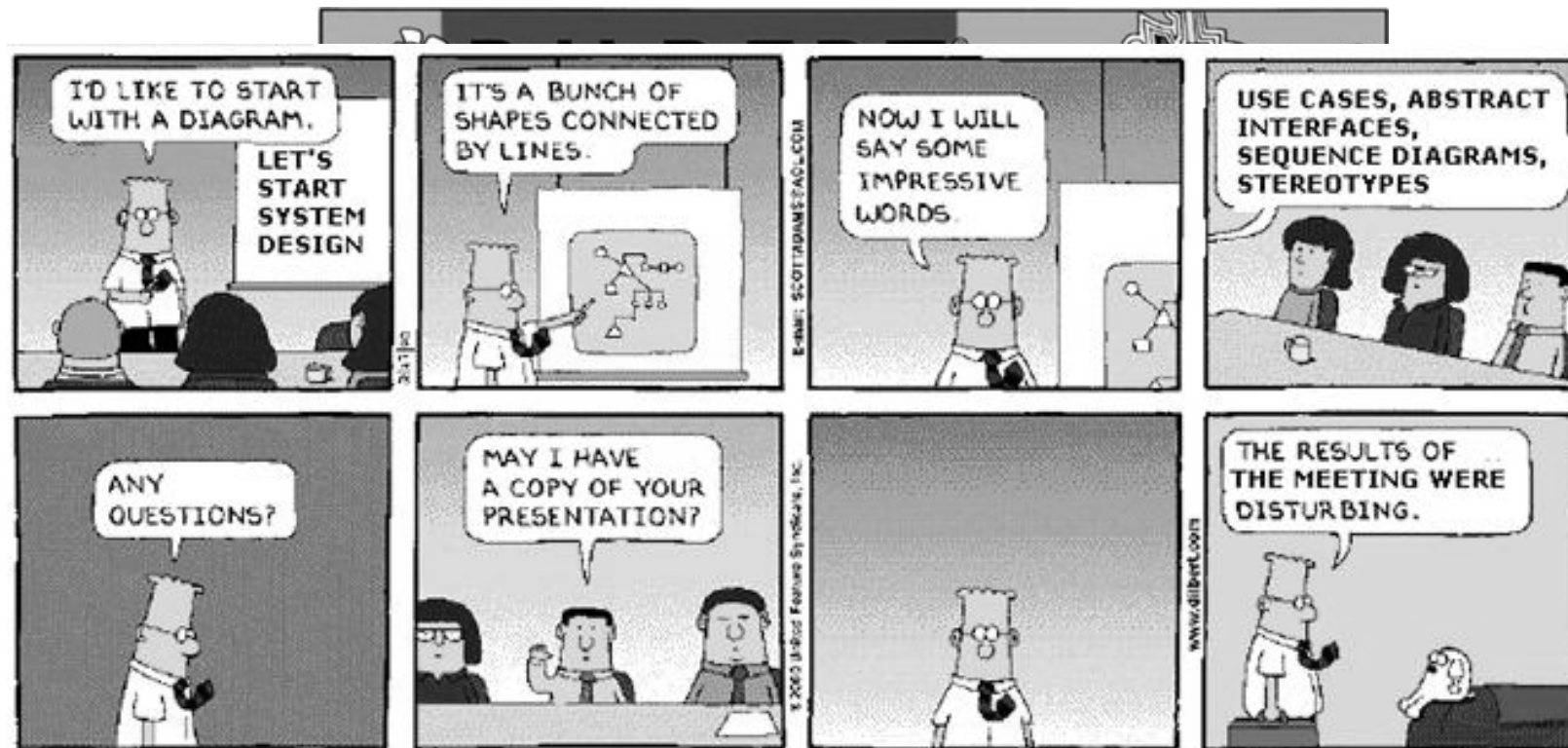
Literatura

- Sommerville, I., Software engineering, 8th ed., Addison-Wesley, 2007.
- Grady Booch, James Rumbaugh, Ivar Jacobson: **Unified Modeling Language User Guide, 2nd Edition, 2005**
- **OMG; OMG Unified Modeling Language, Superstructure Version 2.2 ; URL:**
www.omg.org/spec/UML/2.2/Superstructure/PDF/,
- **Simon Bennett, John Skelton, Ken Lunn: Schaum's Outline of UML, Second Edition, 2005**
- **Prezentacije**
 - **RATIONAL Software (IBM)**
 - **OMG Tutorial Series**
 - **Gunnar Overgaard, Bran Selic, Conrad Bock, Cris Kobryn**

Unified Modeling Language

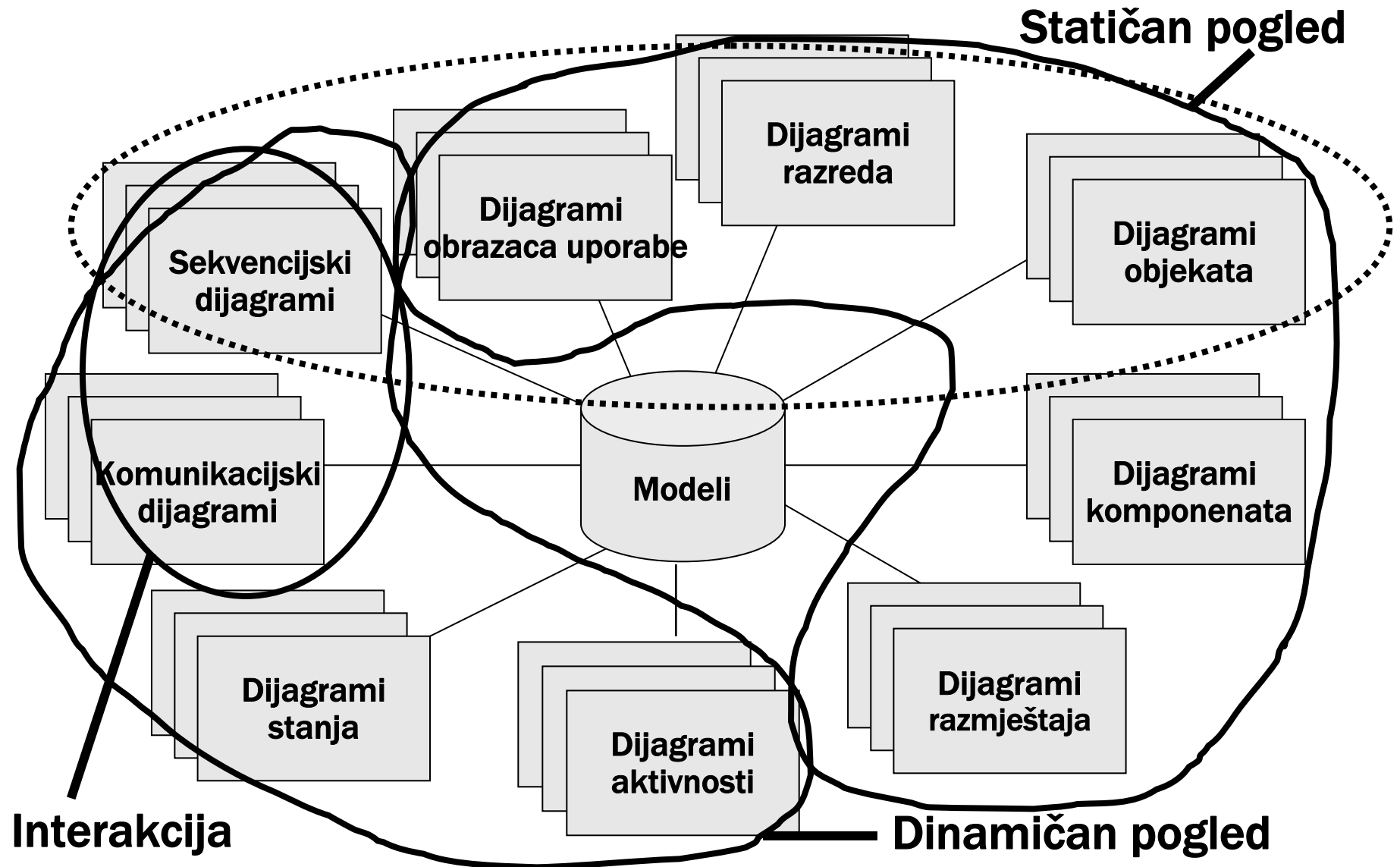
- omogućava različite poglede na model
- ‘de facto’ standardni jezik programskog oblikovanja
- Omogućava
 - Višestruke međusobno povezane poglede
 - Poluformalnu semantiku izraženu kao metamodel
 - Jezik za opis formalnih logičkih ograničenja

O UML-u

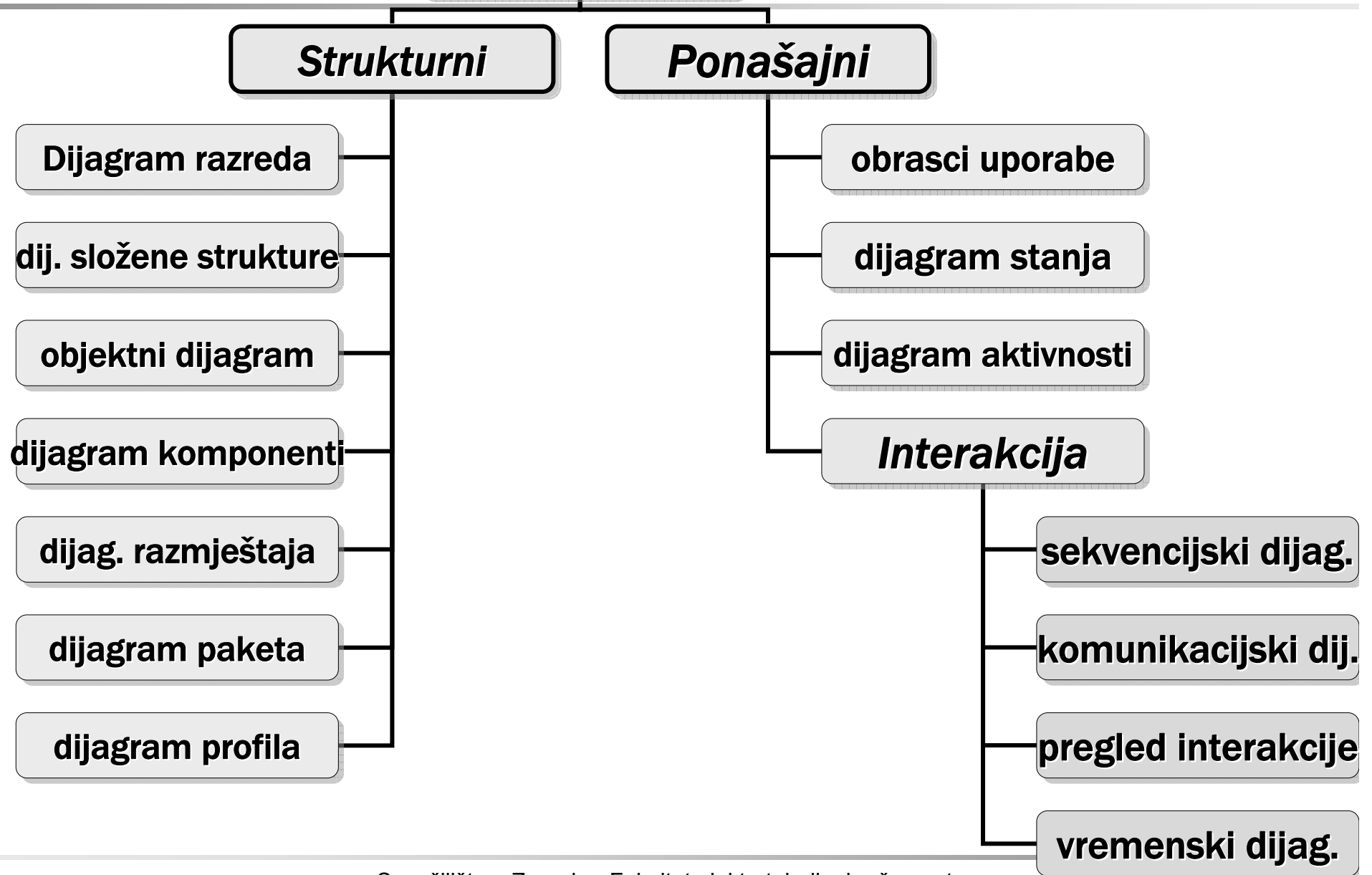


Copyright © 2000 United Feature Syndicate, Inc.

Modeli i dijagrami UML 1.3



UML dijagrami UML 2.2



Dijagrami

- **Pogled na model**
 - Iz perspektive dionika
 - Djelomična reprezentacija sustava
 - Semantički konzistentan s ostalim pogledima
- **Standardni dijagrami (UML 1.1)**
 - statični: use case, class, object, component, deployment
 - dinamični: sequence, collaboration, statechart, activity
- **Specifičnost dijagrama**
 - Svaki ima vlastitu sintaksu i semantiku (Zoo)

UML dijagrami

- Obrasci uporabe
- Sekvencijski dijagram
- Komunikacijski dijagram
- Dijagram stanja
- Dijagram aktivnosti
- Dijagram komponentni
- Dijagram razmještaja
- Dijagram paketa
- Dijagram pregleda interakcije
- Vremenski dijagram
- Dijagram profila
- Dijagram razreda
- Dijagram objekata
- Dijagram složene strukture

Primjer: Bankomat

- **Bankomat**
 - Čitač mag. Kartica
 - Tastatura, zaslon
 - Utor za umetanje omotnica
 - Spremnik novac
 - Printer za ispis potvrda
 - Ključ za uključivanje/isključivanje
 - Komunikacija s bankom

Opis rada

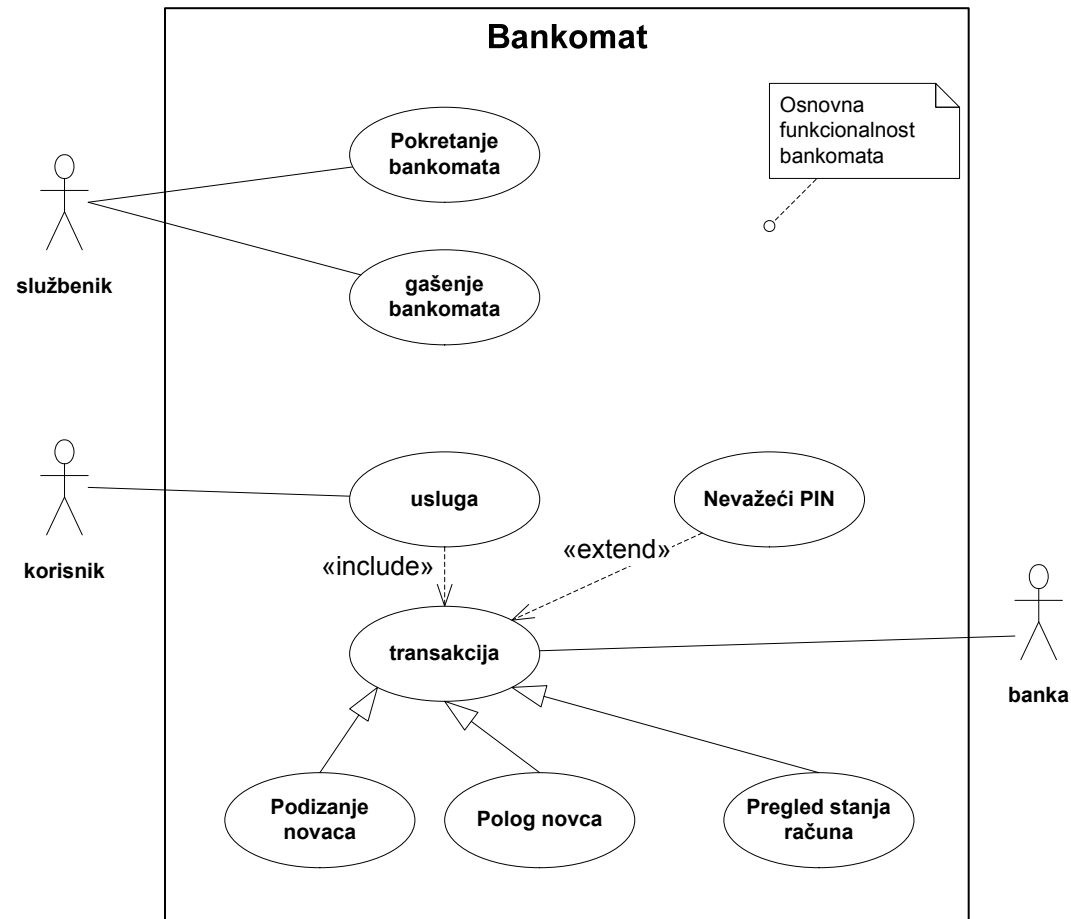
- Posluživanje jednog korisnika
- Ubacivanje kartice + identifikacija PIN-om, podaci se šalju banci na validaciju tijekom svake transakcije
- Korisnik može obaviti jednu ili više transakcija
- Kartica se zadržava u bankomatu sve dok korisnik obavlja transakcije, nakon završetka kartica se vraća (postoji iznimke)

Usluge bankomata

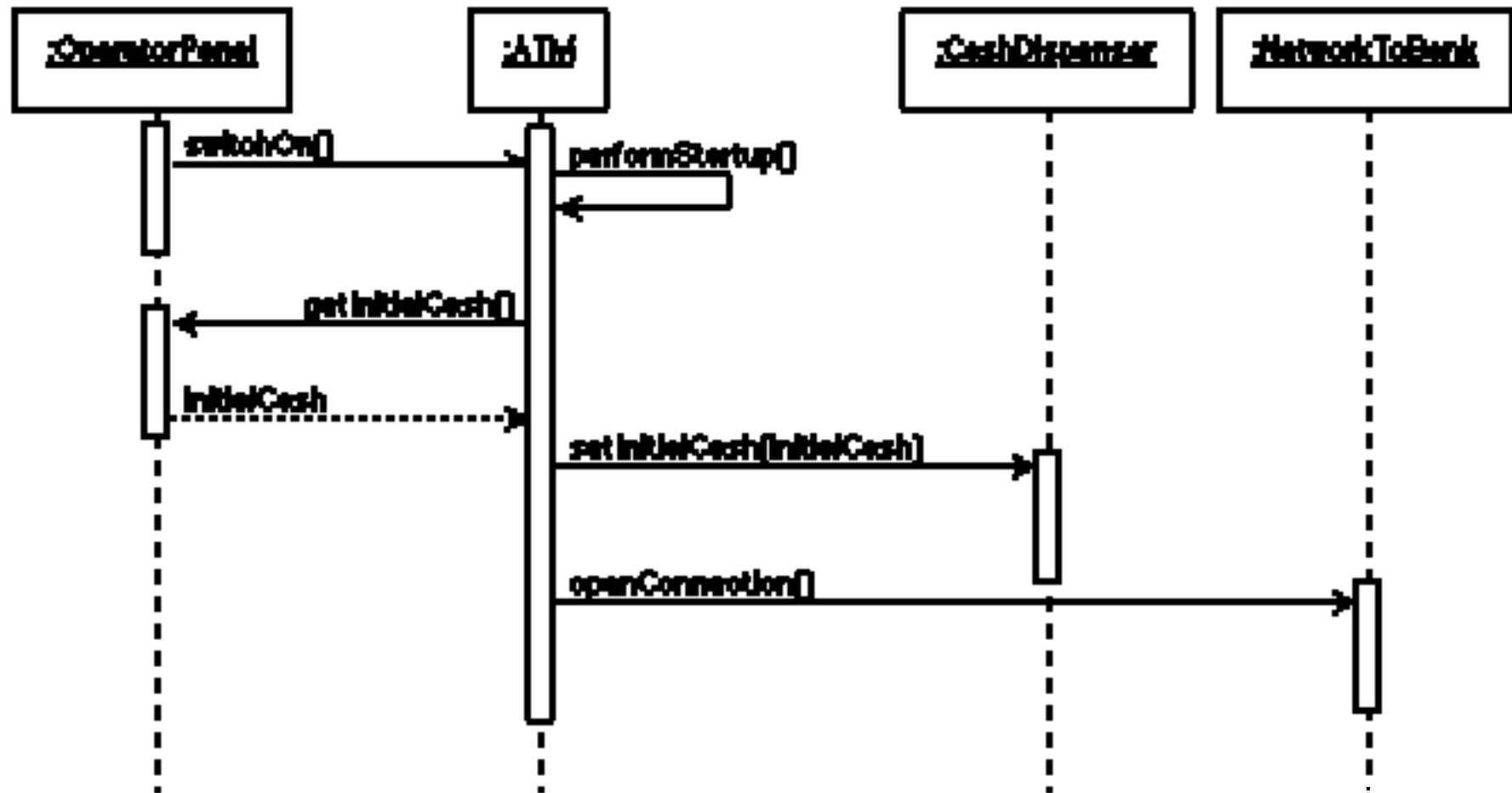
- **Korisnik može podići novce s računa kartice. Podizanje novac odobrava banka.**
- **Korisnik može uložiti novac na račun kartice (gotovina/ček)**
 - **Korisnik upisuje uloženi iznos**
 - **Operator ručno verificira iznos**
 - **Banka odobrava prihvaćanje uplate**
- **Korisnik može prebacivati novce između računa**
- **Korisnik može pregledati stanje računa**

-
- Korisnik može prekinuti započetu transakciju
 - Bankomat komunicira s bankom za verifikaciju svake transakcije
 - U slučaju unosa pogrešnog pina traži se ponovni unos. Nakon tri uzastopna pogrešna unosa kartica zadržava u bankomatu
 - Bankomat ispisuje poruke o pogreškama i pripadajuća objašnjenja prilikom neuspjelih transakcija i postavlja upit o slijedećoj transakciji
 - Bankomat ispisuje potvrdu svake transakcije
 - Posluga bankomata može prekinuti rad s korisnicima
 - Kada korisnik ne obavlja transakciju
 - Kada je bankomat isključen moguće je umetnuti novac, ...
 - Nakon uključivanja upisuje se iznos novca
 - Bankomat interno zapisuje rad svih transakcija
 - Nikada ne zapisuje PIN

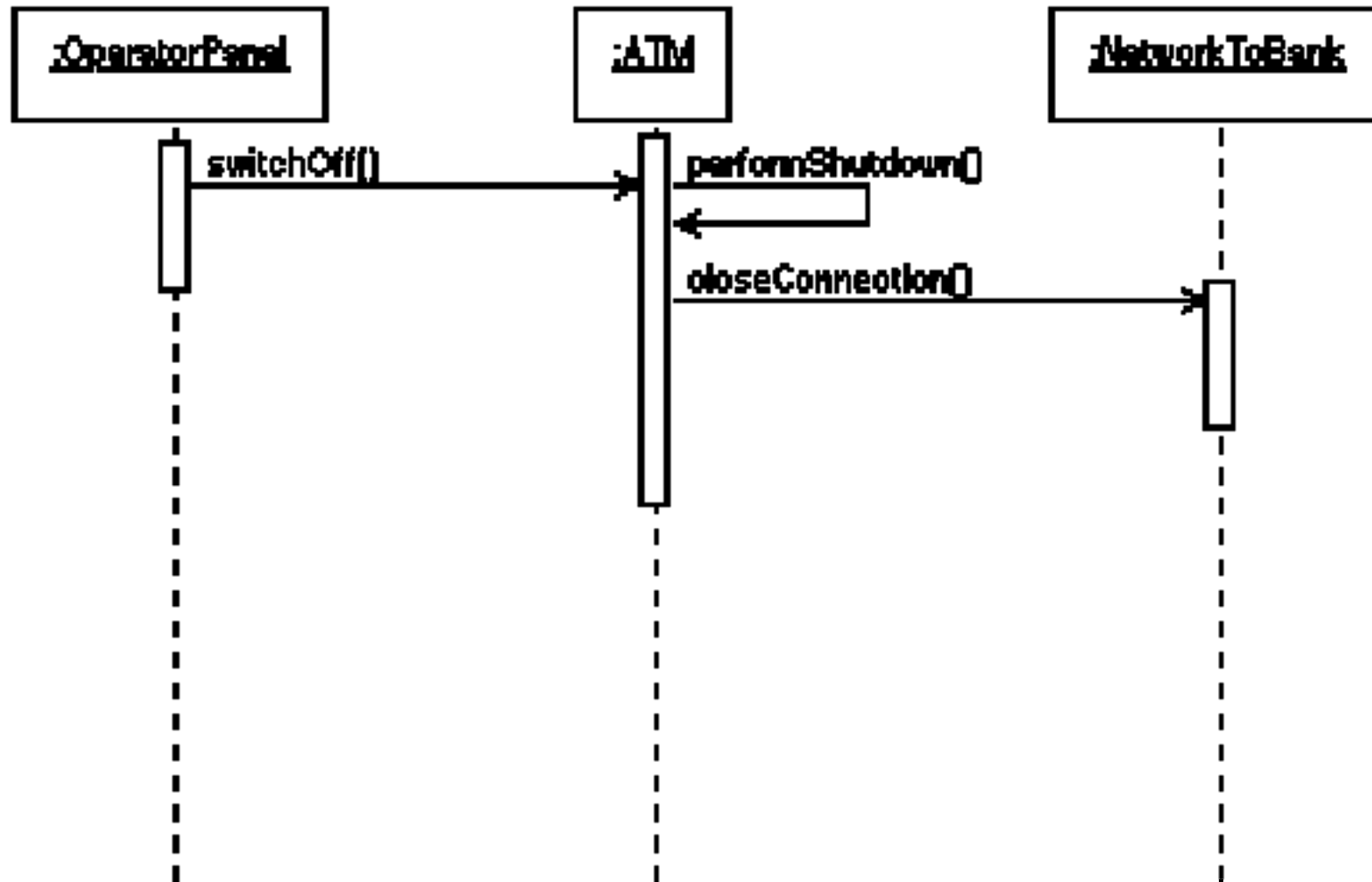
Obrazac uporabe



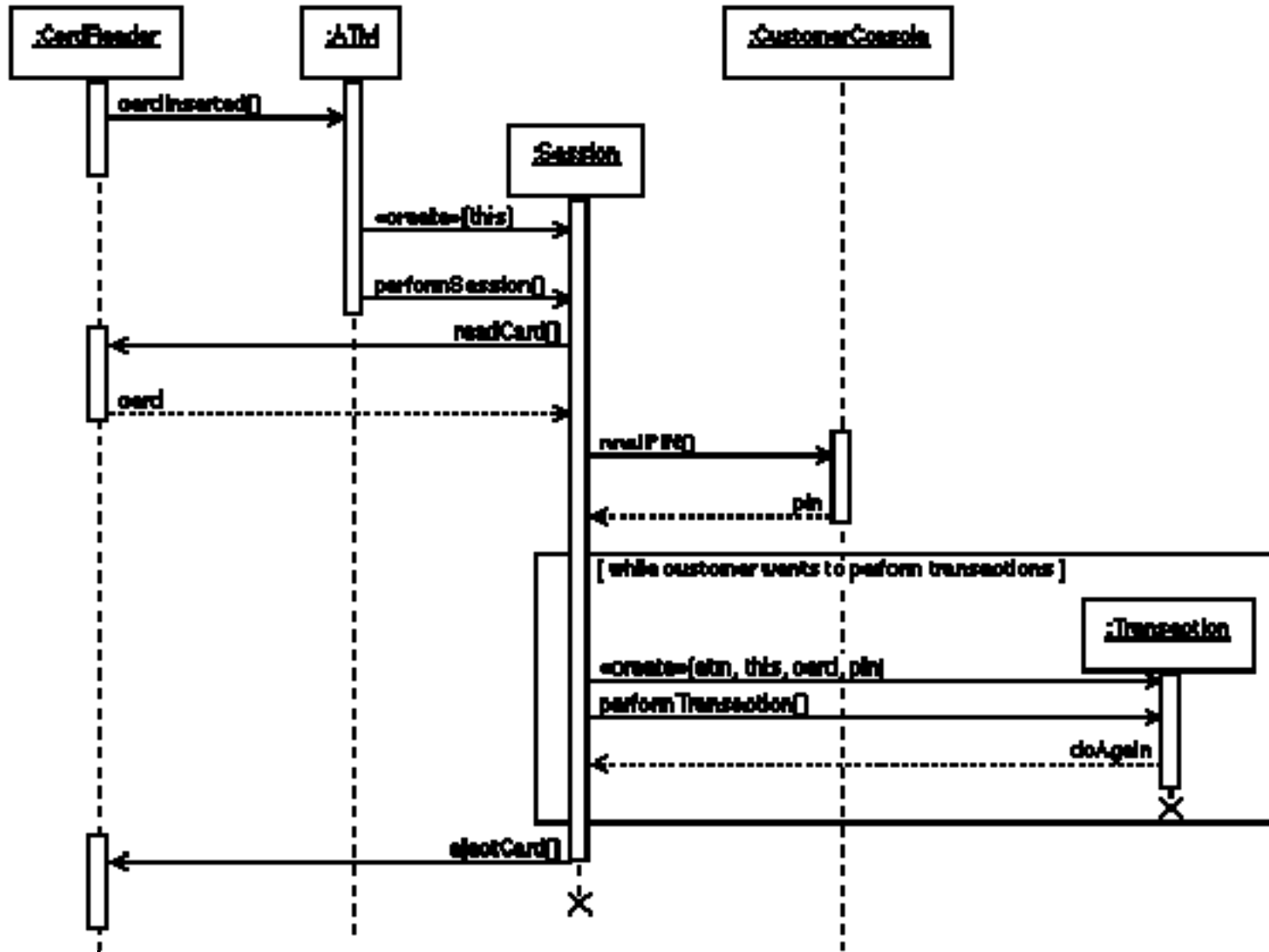
Sekvencijski dijagram pokretanja



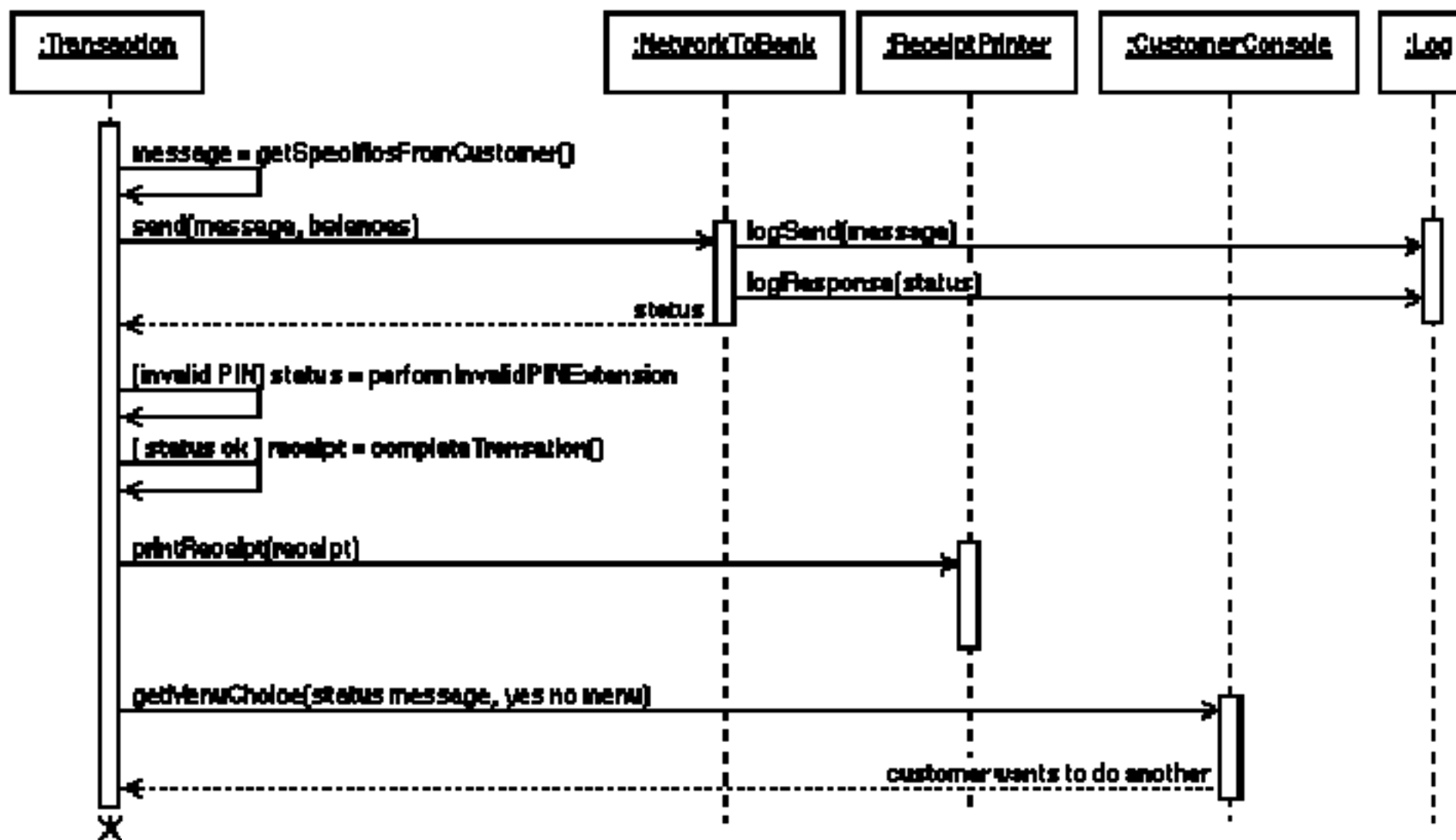
Sekvencijski dijagram gašenja



Sekvencijski dijagram usluge



Sekvencijski dijagram transakcije



UML dijagrami

- Obrasci uporabe
- Sekvencijski dijagram
- Komunikacijski dijagram
- Dijagram stanja
- Dijagram aktivnosti
- Dijagram komponentni
- Dijagram razmještaja
- Dijagram paketa
- Dijagram pregleda interakcije
- Vremenski dijagram
- Dijagram profila
- Dijagram razreda
- Dijagram objekata
- Dijagram složene strukture

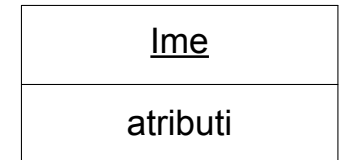
Dinamičke interakcije u sustavu

- **Prikaz interakcija instanci modela**
 - grafički prikaz instanci i podražaja
 - stvaranje i brisanje instanci
- **Utvrdjivanje sučelja razreda**
- **UML dinamički dijagrami interakcija:**
 - **sekvencijski – vrijeme**
 - eksplicitno uređenje odnosa između podražaja
 - modeliranje sustava za rada u stvarnom vremenu
 - **dijagram komunikacije/kolaboracije – struktura**
 - upotrijebiti za opis strukture
 - usredotočeno na efekte instanci

Osnovni elementi interakcija

- Objekti – engl. **Objects**

- različite uloge



- Veze - engl. **Associations**

- prikazuju povezanost objekata koji komuniciraju

- Poruke – engl. **Messages**

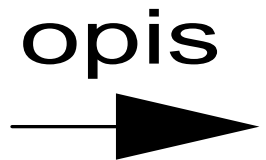
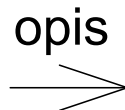
- komunikacija između instanci

- struktura:

[broj_sekvence]{*[petlja]}.{[uvjet]} [:]Ime(parametri) [: povratne vrijednosti]
[sequenceNumber]{*[loop]}.{[condition]} {:} methodName(parameters)[:
returnValue]

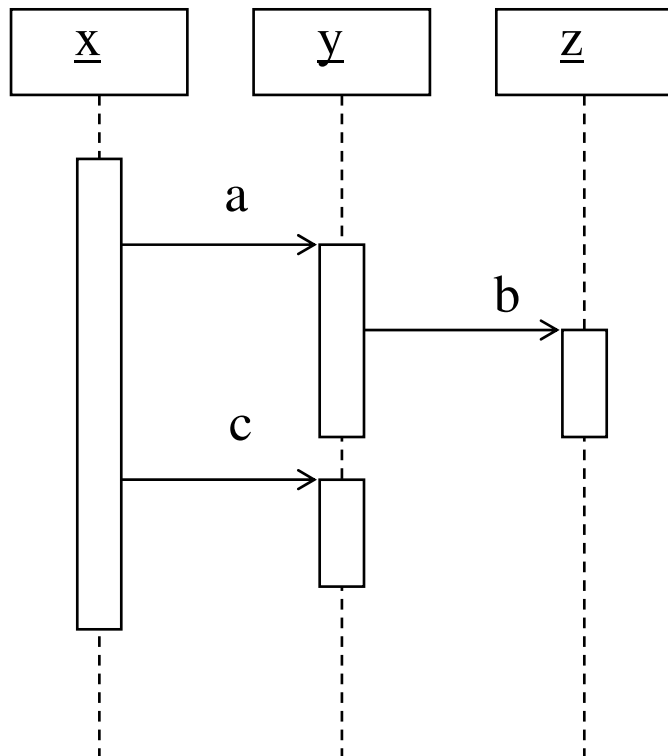
- Oznaka sekvence prema Deweyom sustavu. npr. 1.1, 1.2, 1.3

- asinkrone poruke

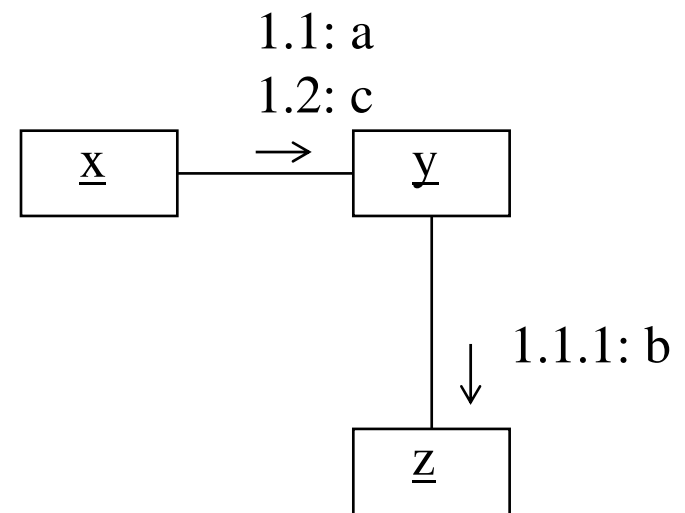


Dijagrami interakcija

■ sekvencijski dijagram



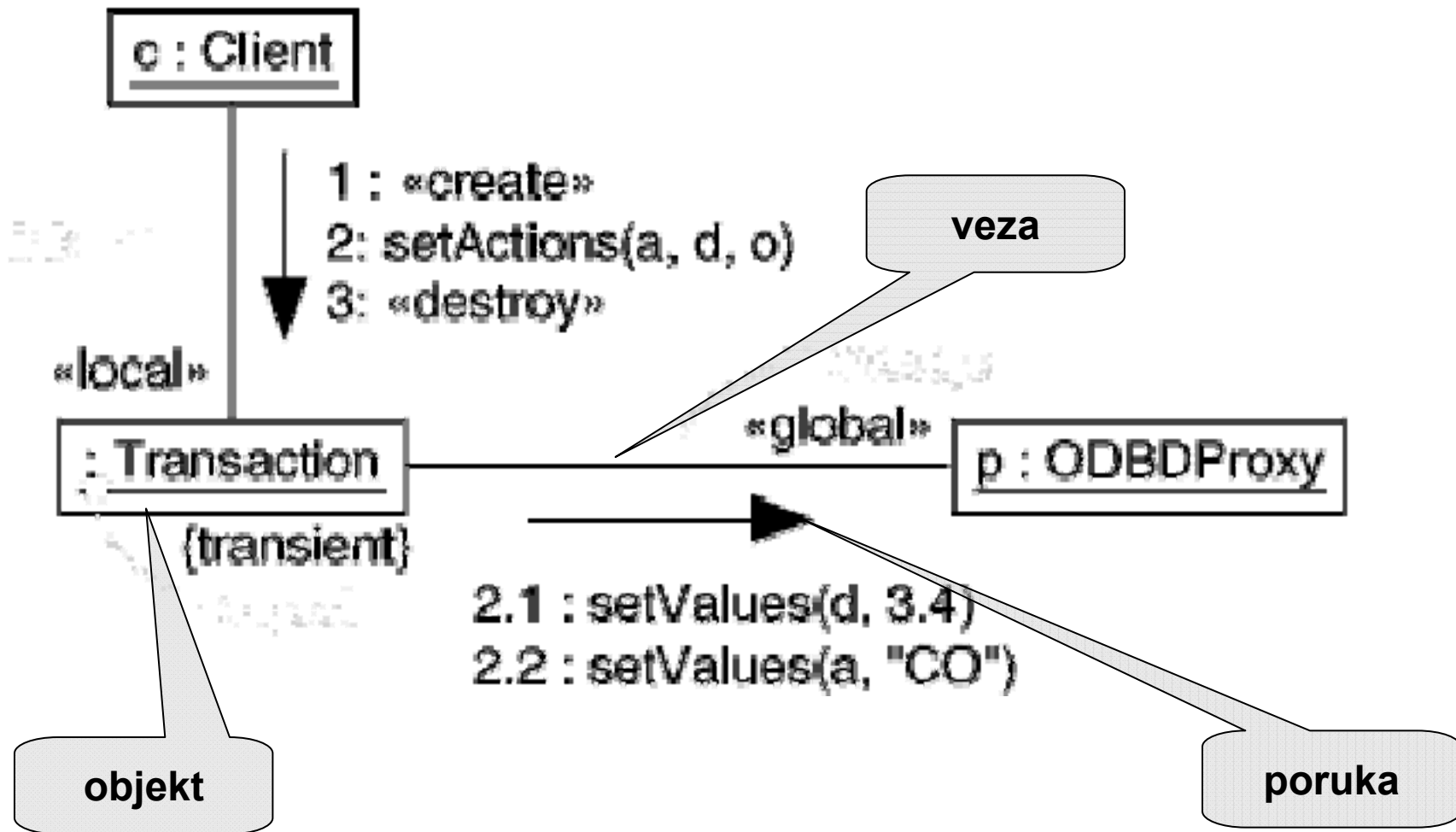
■ diagram komunikacije



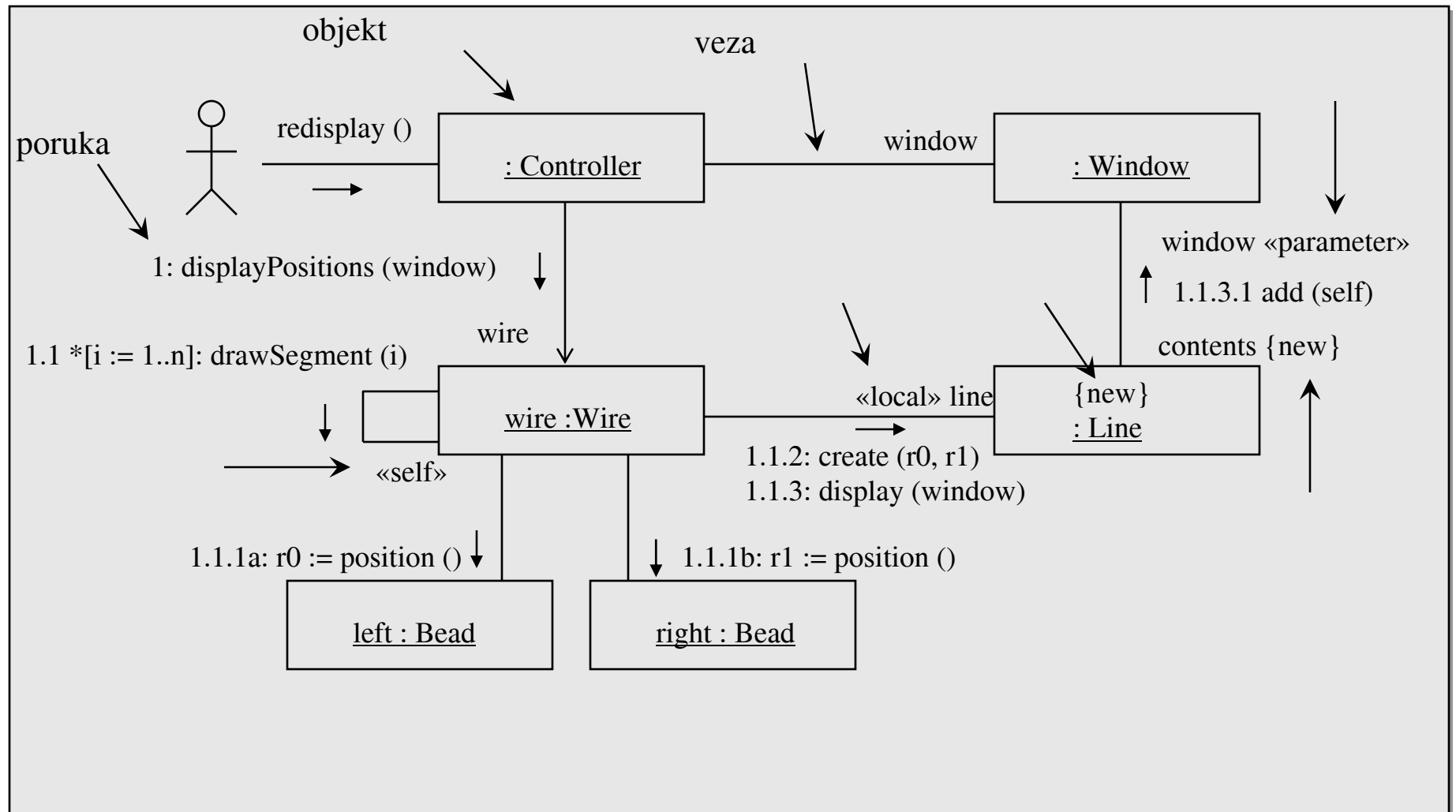
Dijagram komunikacije

- **engl. Communication Diagrams**
- **prijašnje inačice = kolaboracijski dijagram, Collaboration Diagram**
- **obuhvaća dinamičko ponašanje**
 - **poruke – tko šalje kome**
 - **definira uloge instanci tijekom obavljanja nekog zadatka**
- **modelira upravljački tok**
 - **prikaz koordinacije**
 - **nije izravno vidljiv**
 - **specificira tijek komunikacije između instanci tijekom kolaboracije**

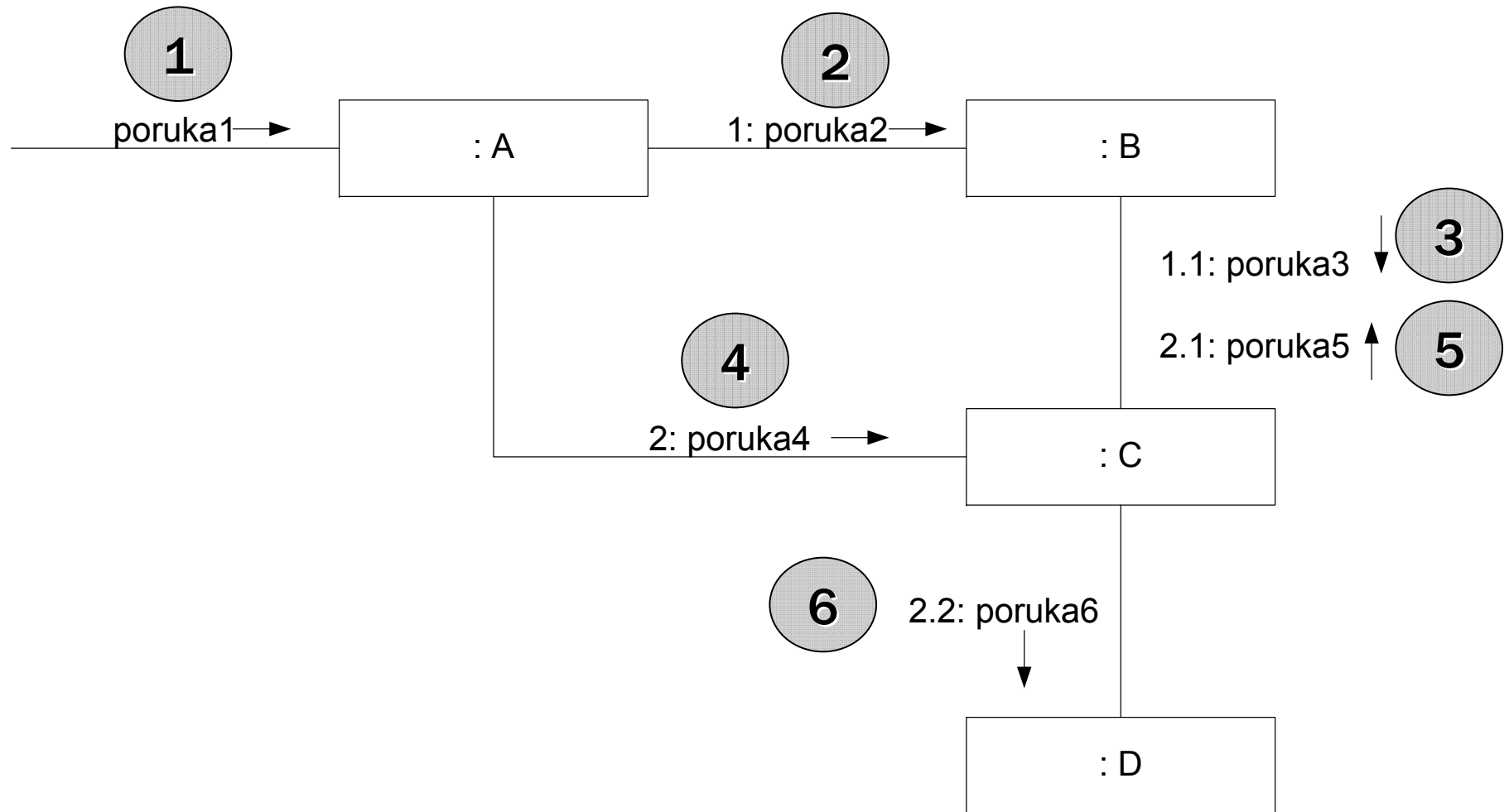
Primjer:



Dijagram komunikacije



Primjer: Označavanje i redoslijed poruka



Primjer: Promjena rute leta

■ Aktori:

- putnik, baza računa klijenta (s planom puta), rezervacijski sustav avio kompanije.

■ Preuvjeti:

- Putnik se prijavio na sustav i odabrao opciju “promjena leta”.

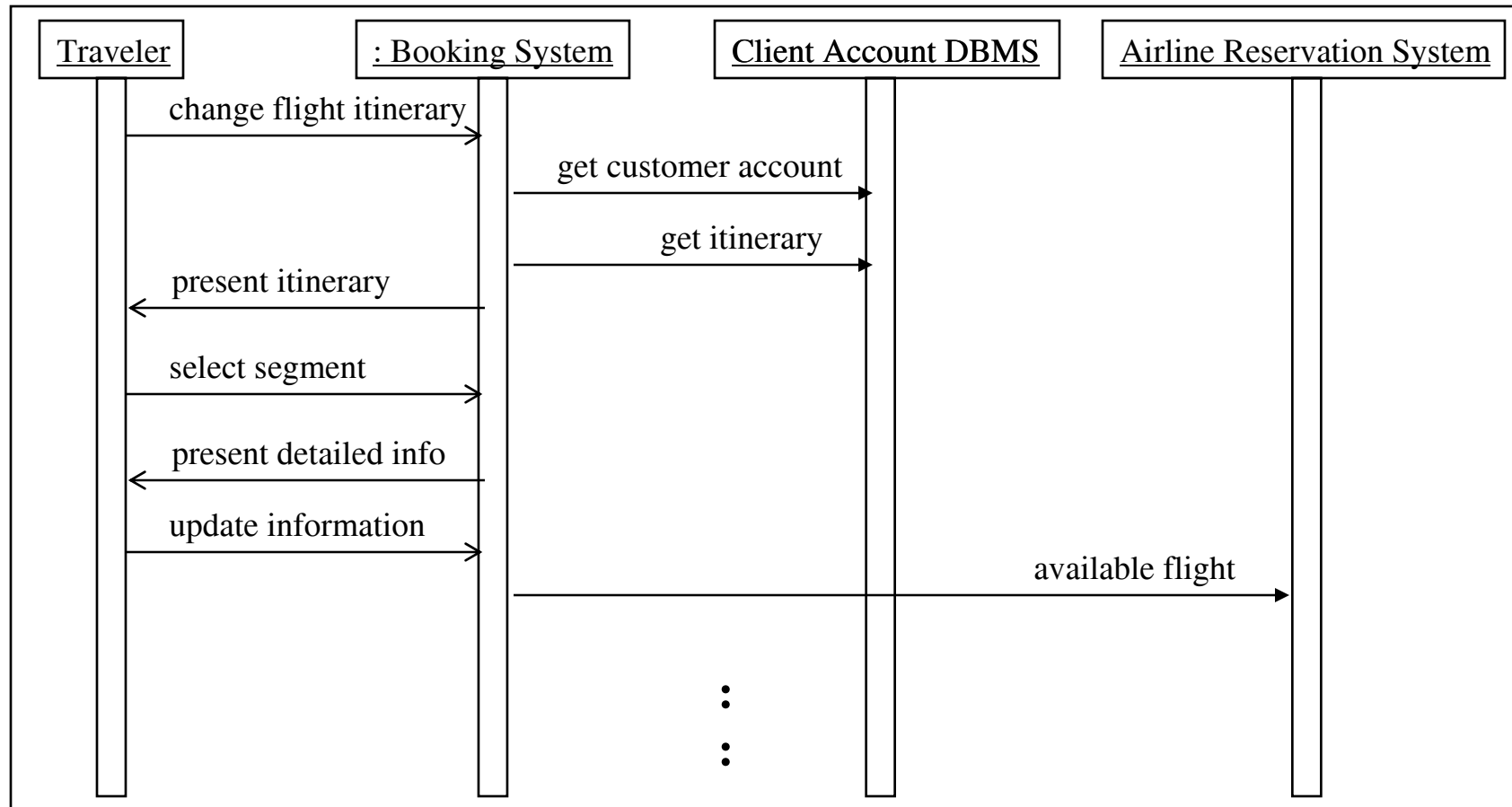
■ Temeljni tijek transakcija

- Sustav dohvaća putnikov bankovni račun i plan puta iz baze.
- Sustav pita putnika da odabere dio plana puta koji želi mijenjati; putnik selektira segment puta.
- Sustav pita putnika za novi odlaznu i dolaznu destinaciju; putnik daje traženu informaciju.
- Ako je let moguć, tada ...
- ...
- Sustav prikazuje sažetak transakcije..

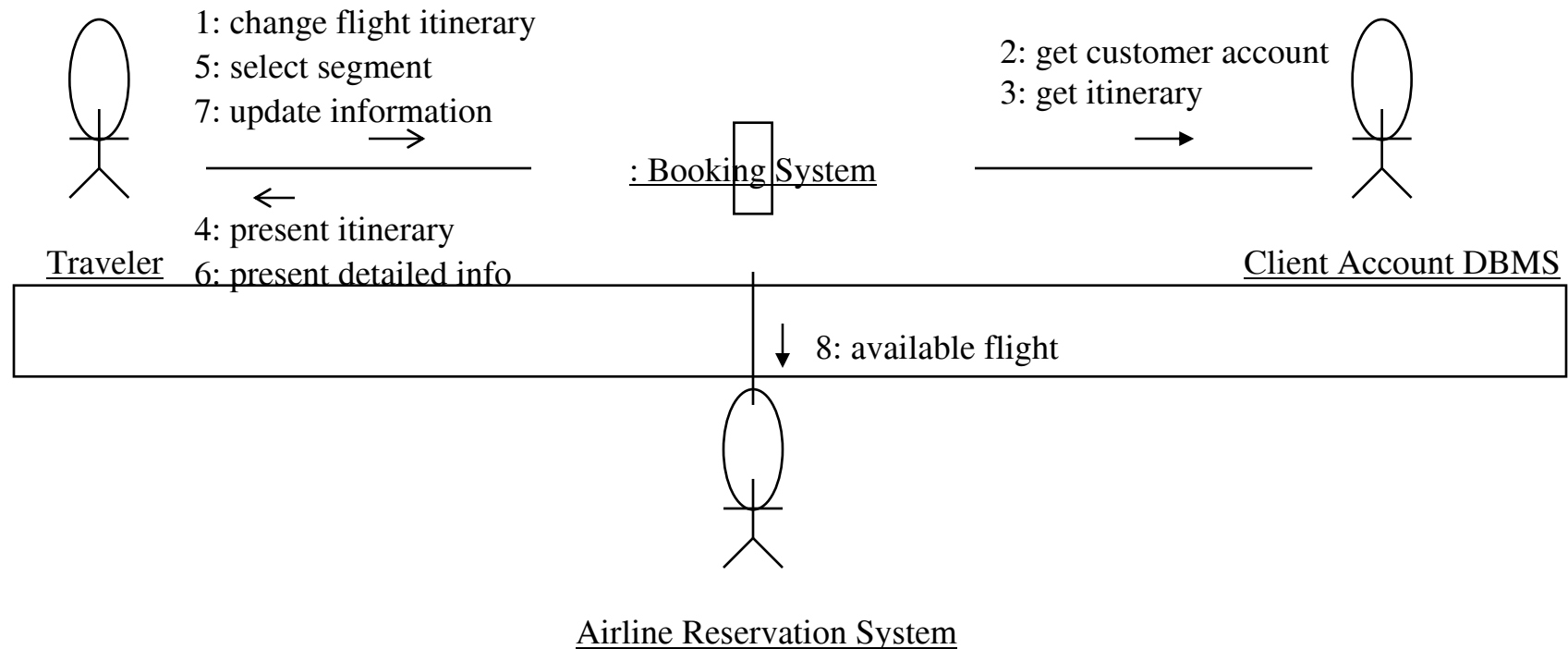
■ Alternativni tijek transakcija

- Ako let nije moguć, tada ...

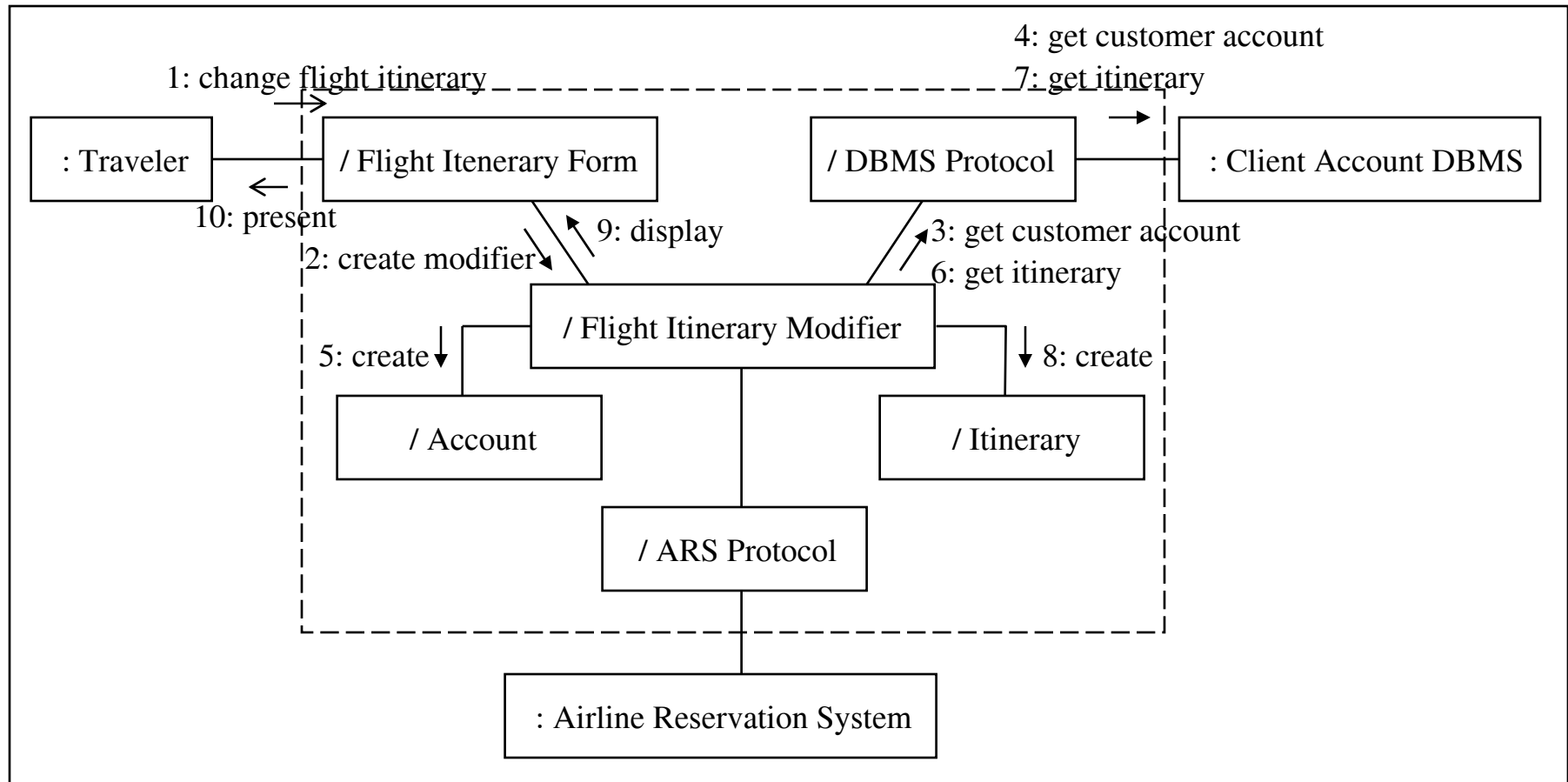
Primjer: sekvencijski dij. promjene rute leta



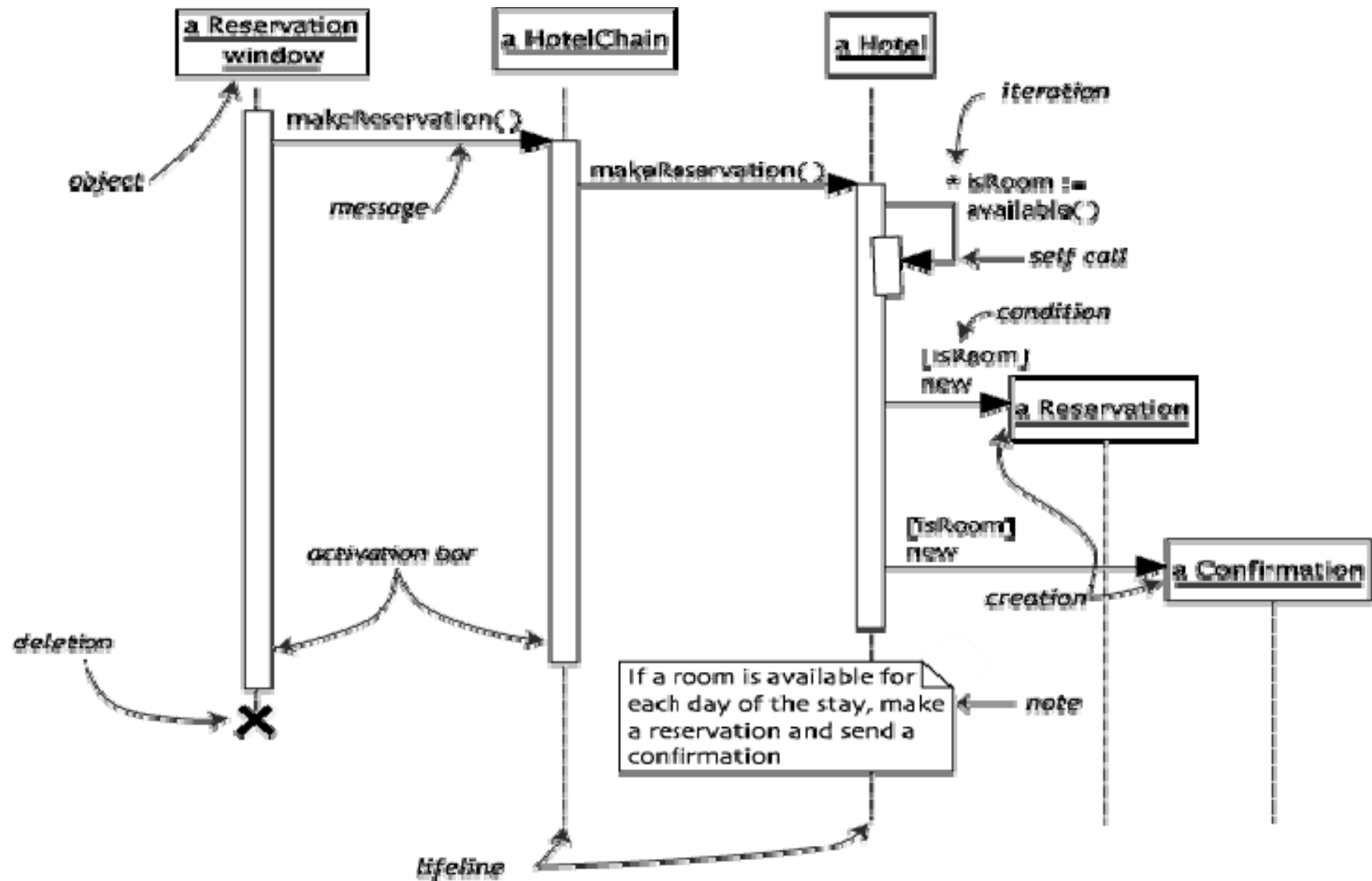
Promjene rute leta - dijagram komunikacije



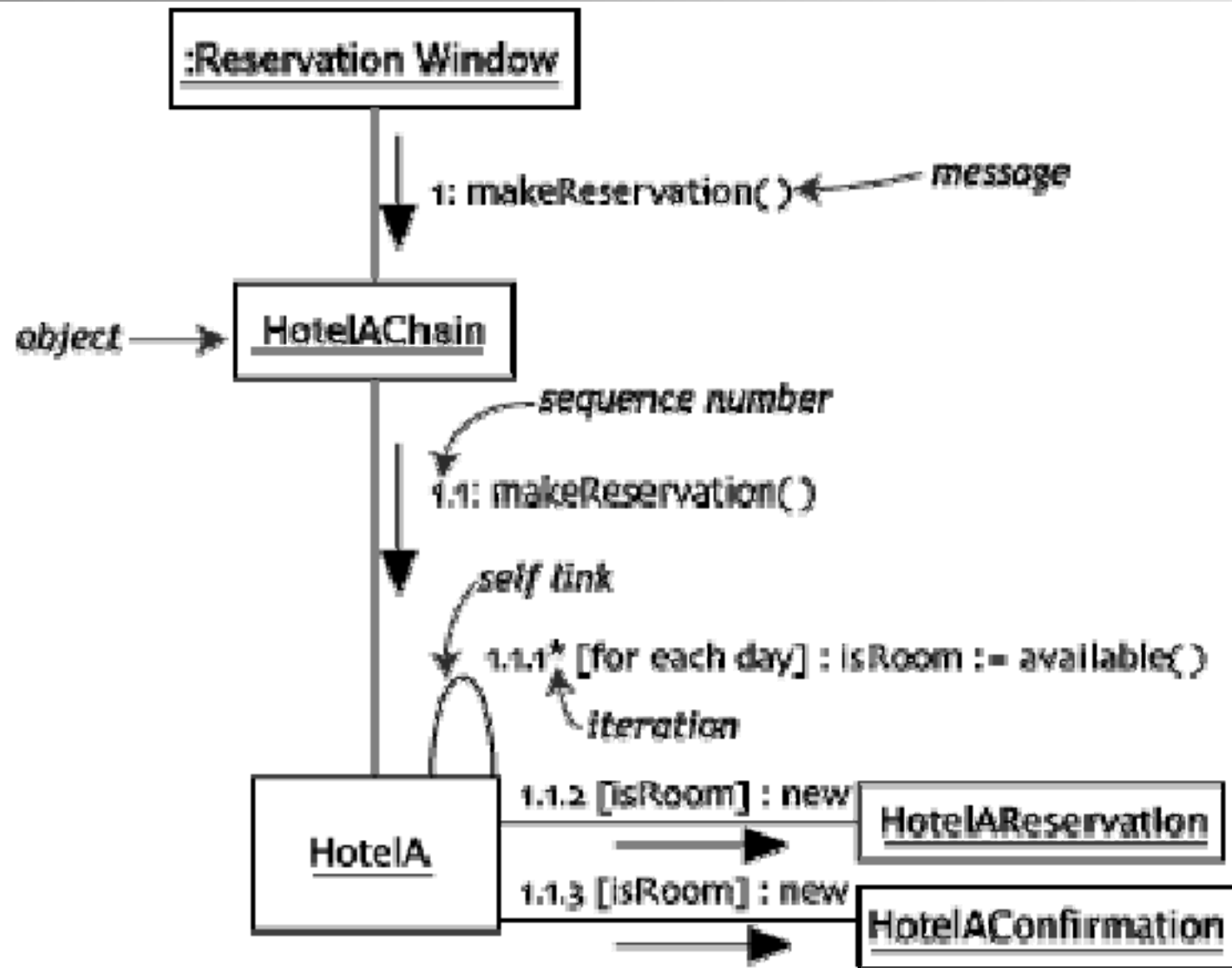
Promjene rute leta - dijagram komunikacije



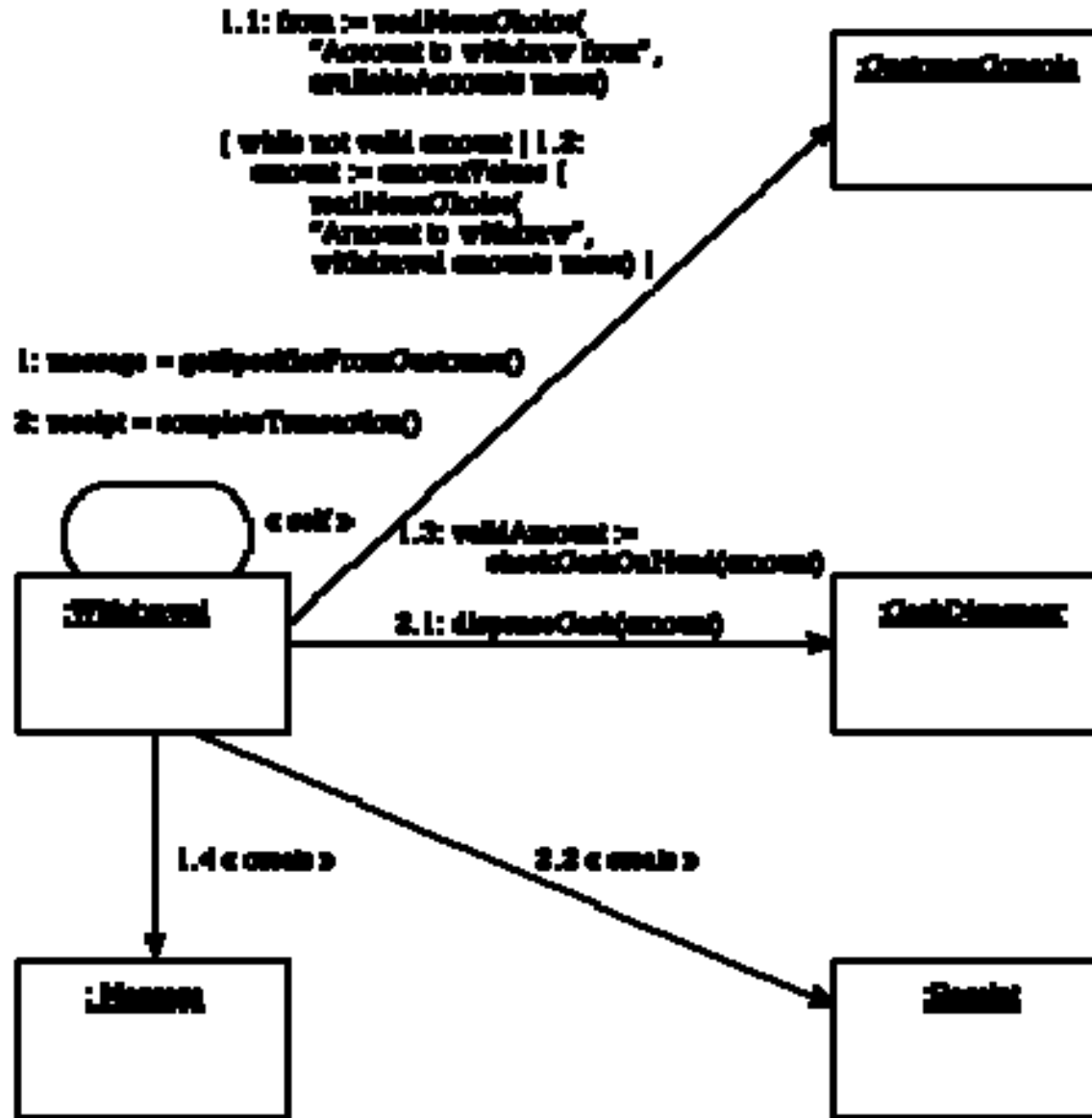
Primjer: Rezervacija hotela



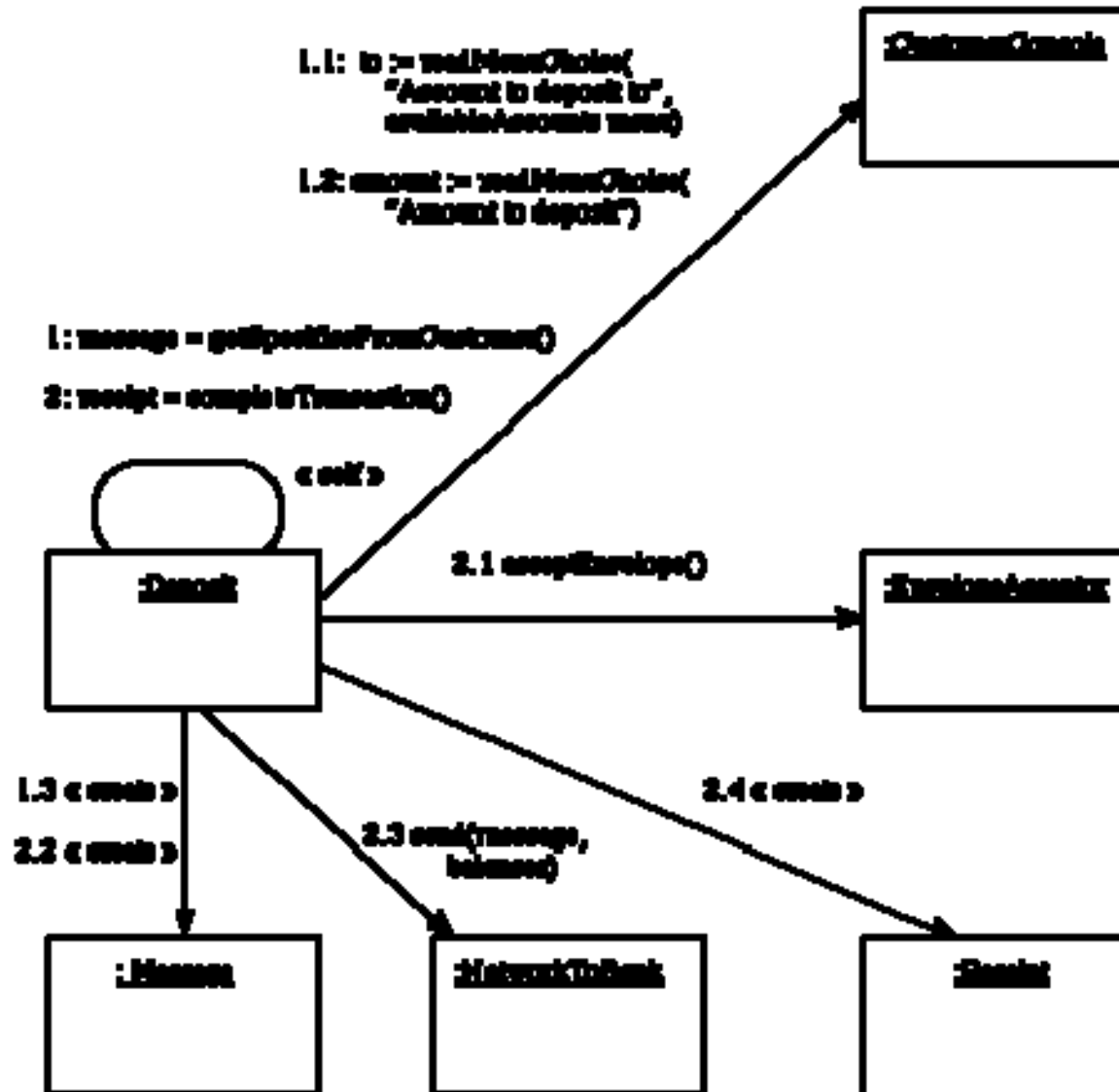
Dijagram komunikacije rezervacije hotela



Primjer: Dijagram komunikacije



Primjer: Dijagram komunikacije pologa



Preporuke

- odaberite ime koje prikazuje namjenu
- pri crtanju minimizirajte presijecanje linija
- upotrijebite samo nužne elemente
- važne elemente smjestite u centar dijagrama
- započnite s porukom koja započinje interakciju i nastavite sistematično
- jedan dijagram interakcije prikazuje samo jedan upravljački tok!!

UML dijagrami

- Obrasci uporabe
- Sekvencijski dijagram
- Komunikacijski dijagram
- Dijagram stanja
- Dijagram aktivnosti
- Dijagram komponentni
- Dijagram razmještaja
- Dijagram paketa
- Dijagram pregleda interakcije
- Vremenski dijagram
- Dijagram profila
- Dijagram razreda
- Dijagram objekata
- Dijagram složene strukture

Dijagrami stanja

- **Nedostatak standardnih dijagrama stanja**
 - velik broj stanja složenijih sustava, nepreglednost
- **1987. Harel, D.: Statecharts: A visual formalism for complex systems.**
 - vizualni formalizam za opis stanja i prijelaza s naglaskom na modularnost, grupiranje, ortogonalnost, konkurentnost i poboljšanja
 - modeliranje jednog reaktivnog objekta
 - osnova UML dijagrama stanja s modifikacijom semantike i terminologije

Dijagrami stanja

- **engl. State Machine Diagram**
- **opisuje dinamičko ponašanje jednog objekta u vremenu**
 - pogodno za opis značajnijeg dinamičkog ponašanja objekta
 - objekt se promatra izolirano od ostalih
 - izlaz ne ovisi samo o trenutnim ulazima nego i o povijesti
 - pogodno za opis diskretnog ponašanja (engl. discret-event)
- **prikazuje sekvencu stanja objekta te prijelaze iz jednog stanja u drugo temeljene na događajima**
- **stanje objekta**
 - opis stanja (okolnosti) u kojem se objekt nalazi kada zadovoljava određene uvjete
 - vrijednosti jednog ili više atributa objekta
 - u jednom stanju objekt može obavljati tri grupe aktivnosti:
 - **do** – aktivnosti koje se izvode za vrijeme dok je objekt u tom stanju
 - **entry**– aktivnosti koje se izvode pri ulasku u stanje
 - **exit** – aktivnosti koje se izvode izlasku iz stanja

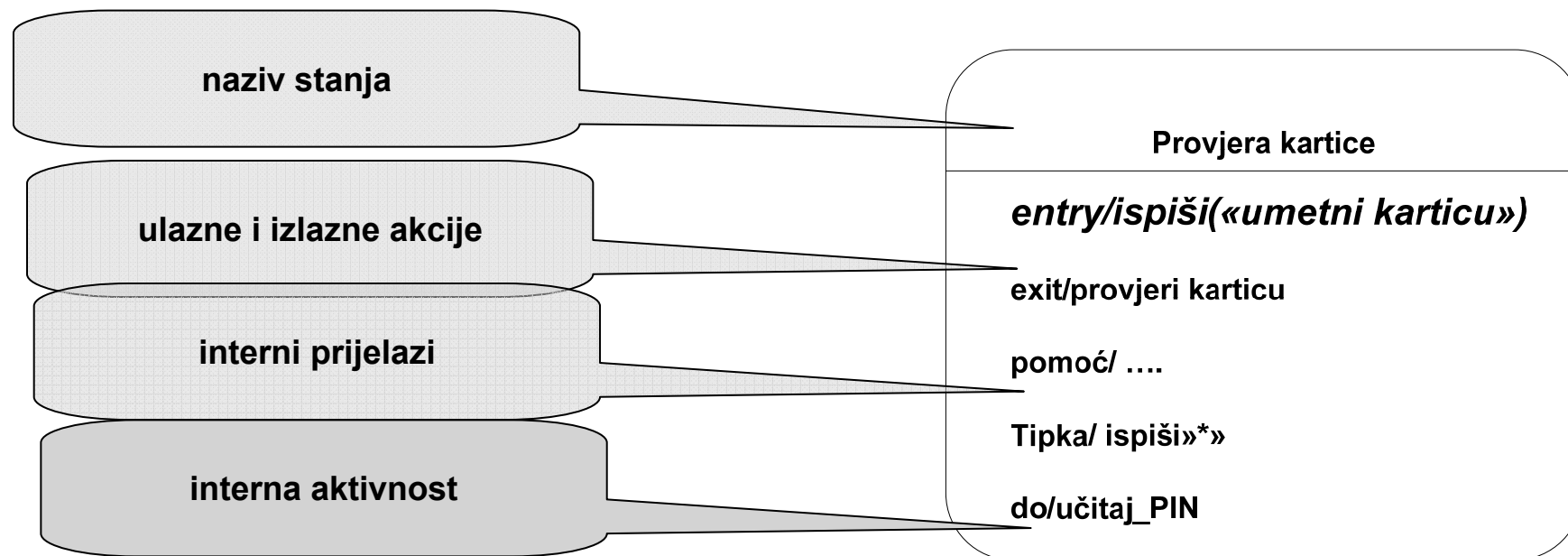
-
- početno stanje – jedno
 - krajnje stanje - može imati više njih
 - prijelaz
 - sve dozvoljene promjene stanja iz trenutnog u novo
 - novo stanje može biti to isto stanje
 - inicirani su događajima i uvjetima
 - [uvjet] događaj/akcija
 - događaji:
 - interakcija
 - asinkroni prijem signala - primljene poruke – engl. signal
 - sinkroni poziv objekta – engl. call
 - vremenski – engl. time
 - proteklo vrijeme
 - apsolutno vrijeme
 - ispunjeni uvjeti – engl. change
 - mogu prenositi parametre
 - trajanje izvođenja prijelaza je 0 i ne može se prekinuti

Aktivnost i akcije

- **Aktivnost – engl. Activity**
 - obavlja se sve dok je stanje aktivno
 - ovisi o dolaznim događajima
- **Akcija – engl. Action**
 - kratkotrajno, neprekidivo ponašanje

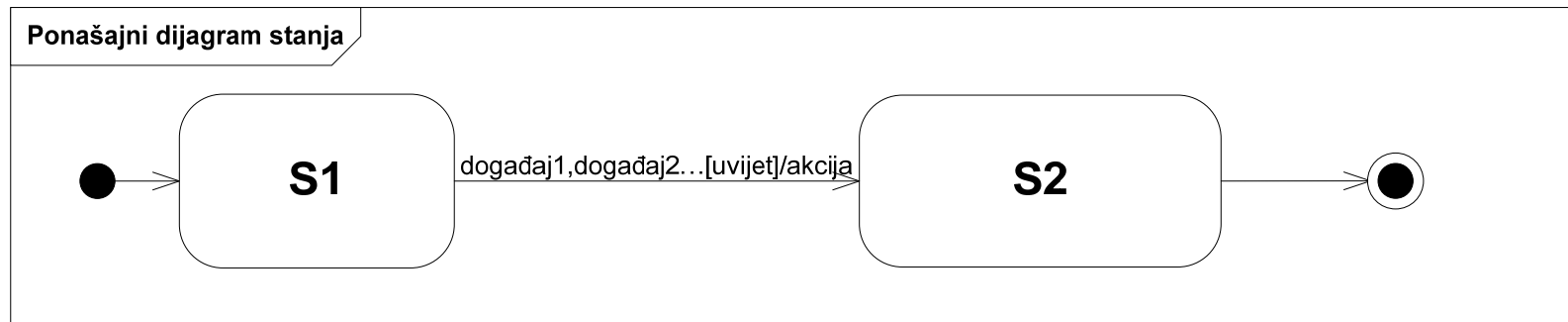
Stanje

- akcija:
 - naziv_događaja/Akcija
- aktivnost
 - do/Aktivnost



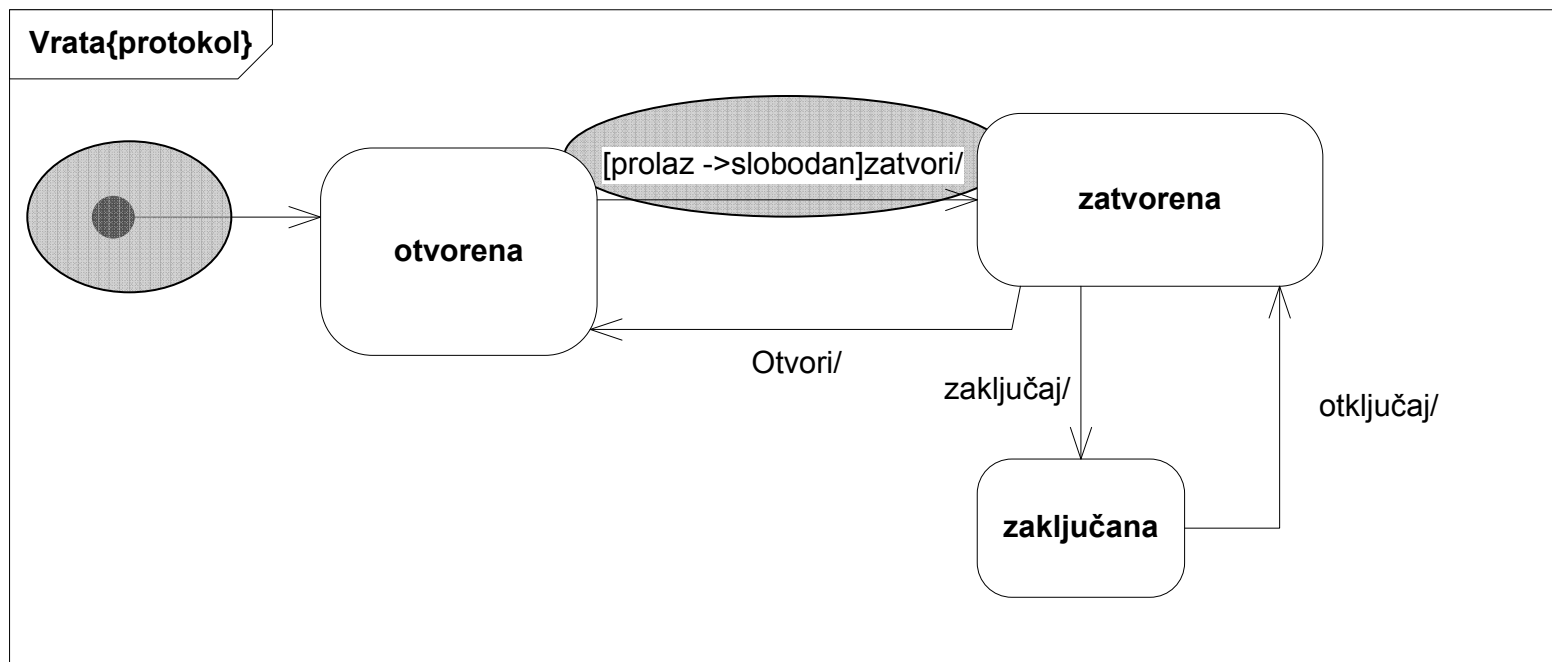
Prijelazi

- ponašajni dijagrami- engl. *behavioral state diagrams*
- događaji
 - vanjska ili unutarnja pojavljivanja događaja koja pokreću prijelaz
- uvjet
 - izraz koji kada je ispunjen(istinit/true) dopušta odvijanje prijelaza (naravno uz pojavu događaja)
- akcija
 - operacije koje se odvijaju



Primjer

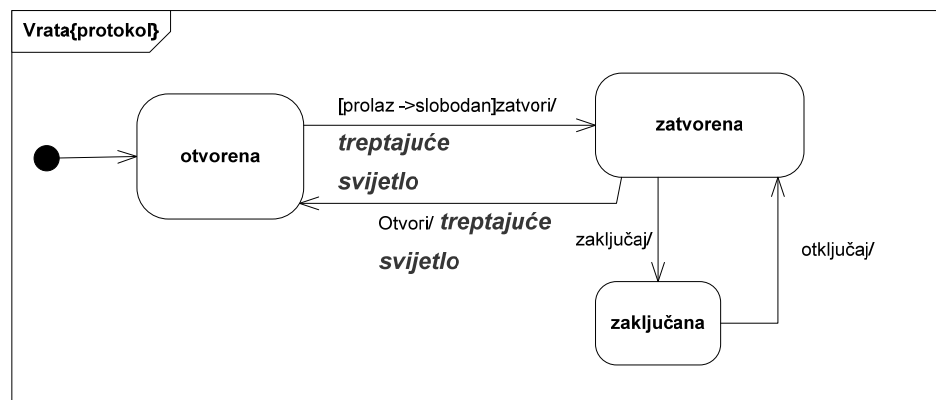
■ Grafički prikaz ponašanja ulaznih vrata



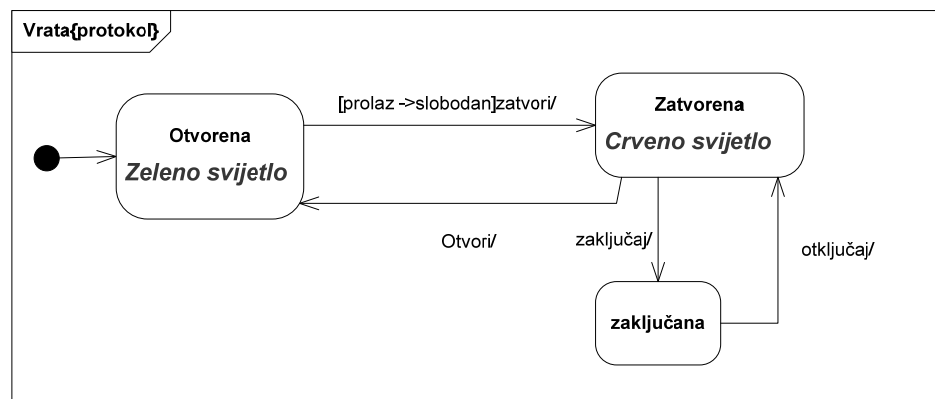
Promjena stanja i akcije

- automati s izlazom - može generirati akciju

- Mealyev automat

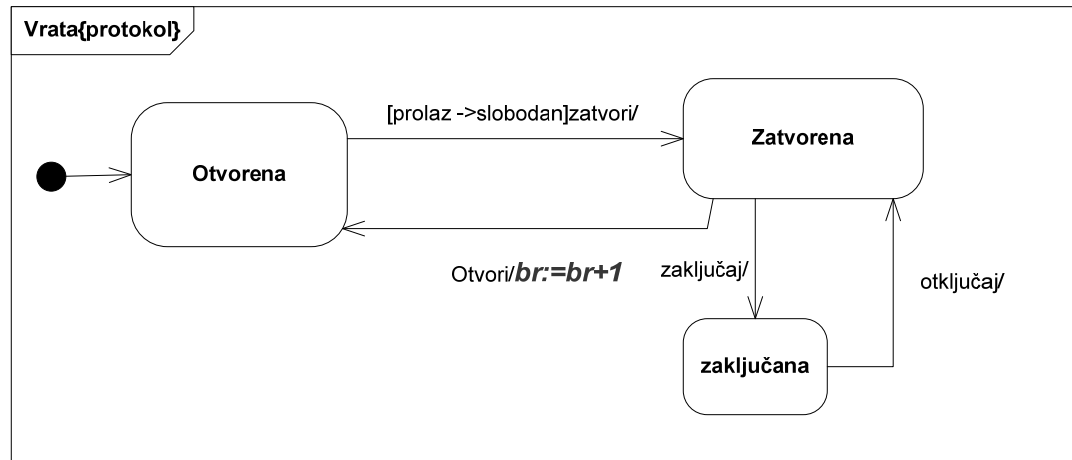


- Mooreov automat



Proširenje stanja

■ dodatne varijable stanja



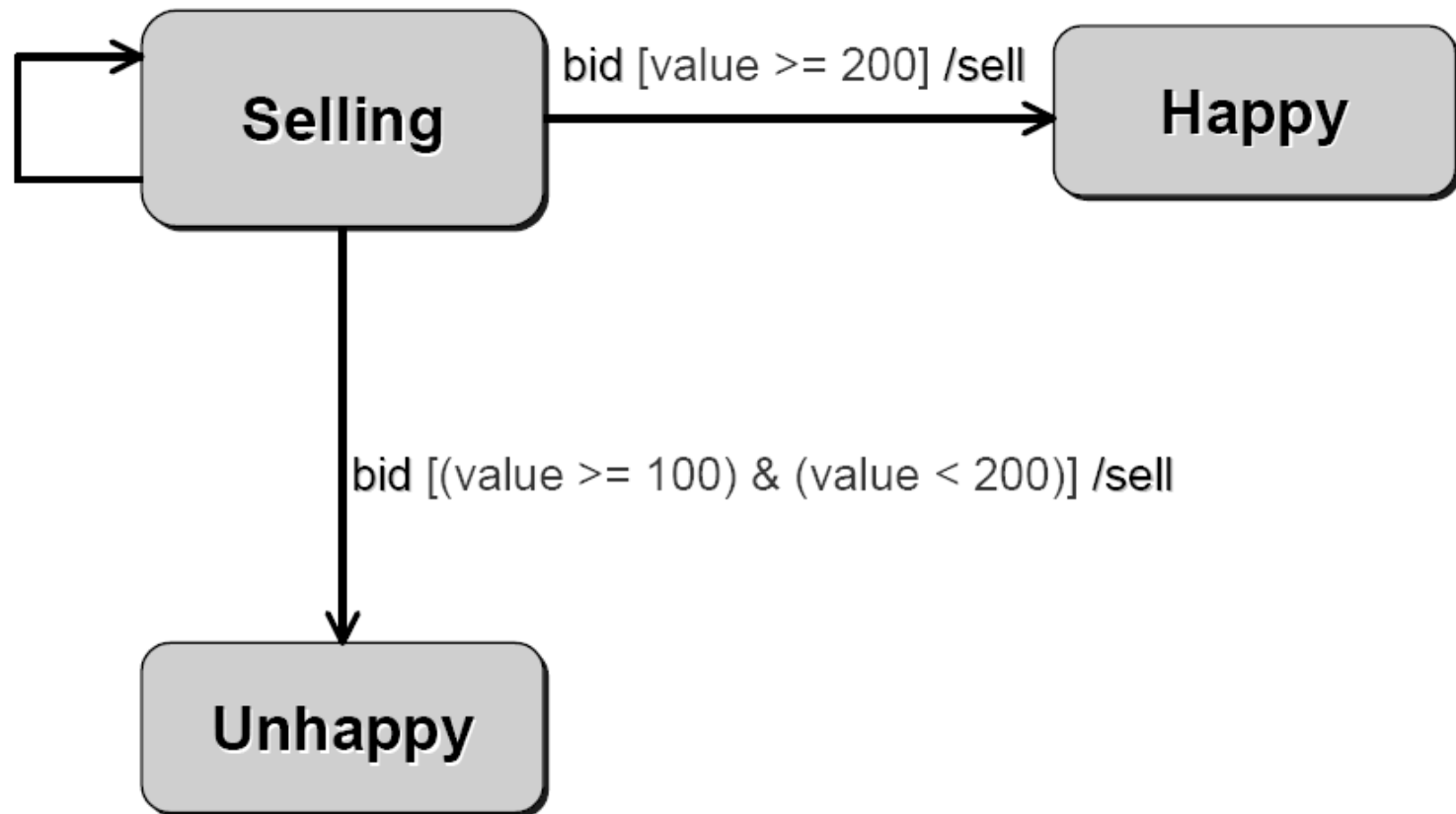
■ definiranje automata

- konačan skup ulaznih signala
- konačan skup izlaznih signala
- konačan skup stanja
- funkcija prijelaza
- konačan skup proširenih varijabli stanja
- početno stanje
- skup konačnih stanja

Uvjeti

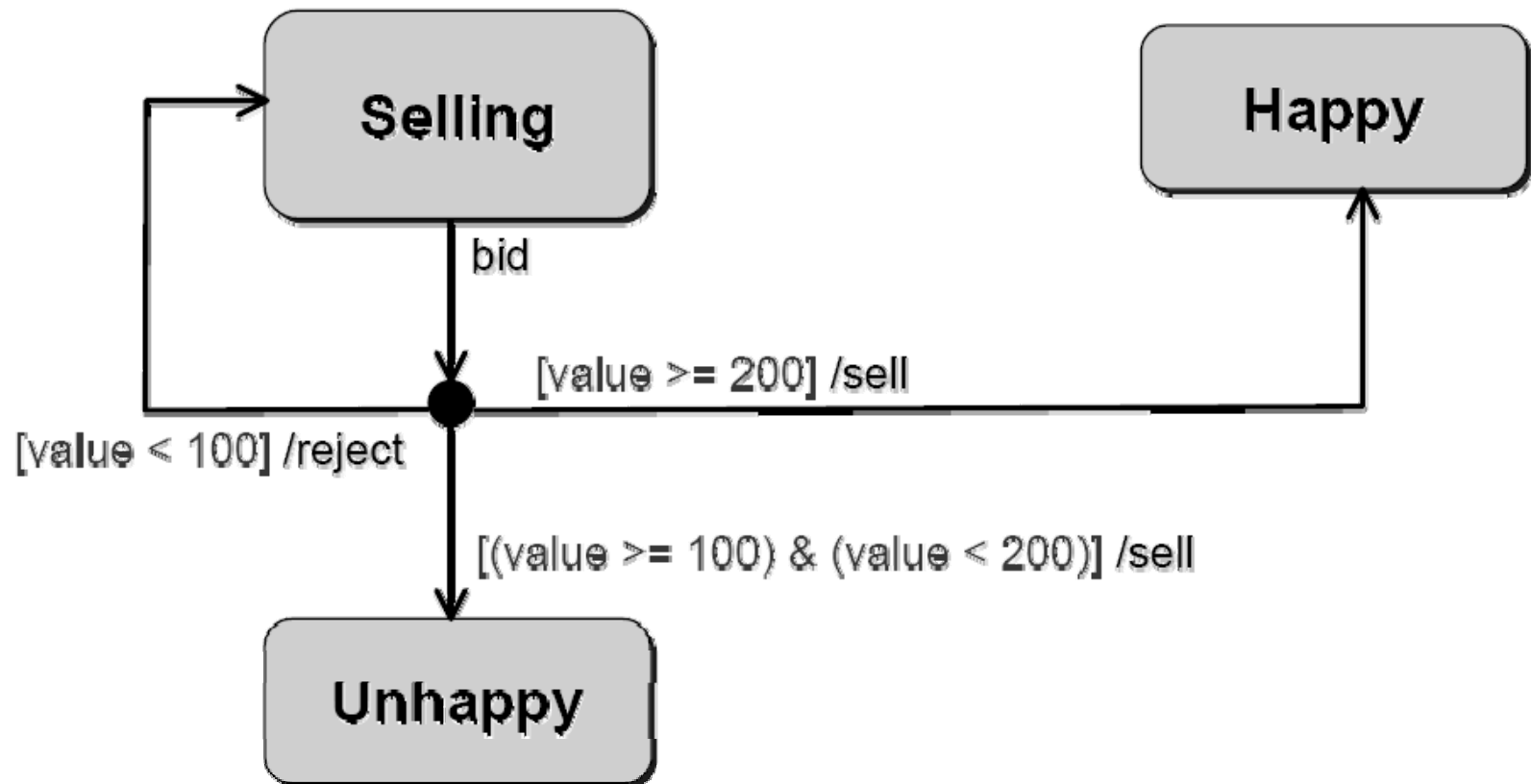
- uvjetno izvođenje prijelaza
- npr: prodaja dionica kada je ispunjen uvjet

bid [value < 100] /reject



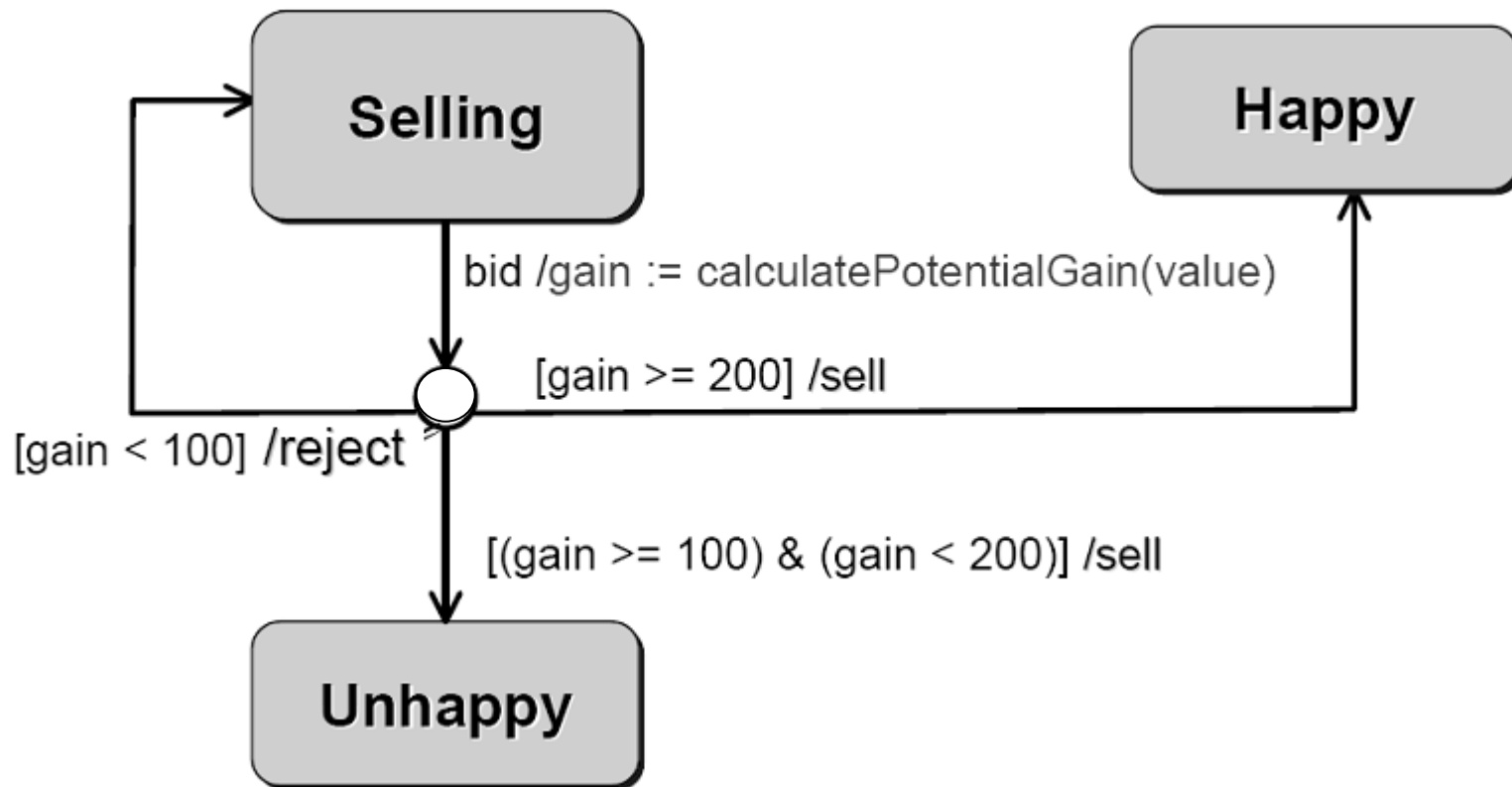
Uvjetna grananja

- pojednostavljivanje grafičkog prikaza
 - veća razumljivost












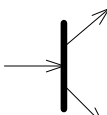
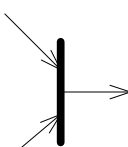
Dinamička uvjetna grananja

- uvode pseudo stanja
 - uvjeti se izračunavaju ulaskom u njih



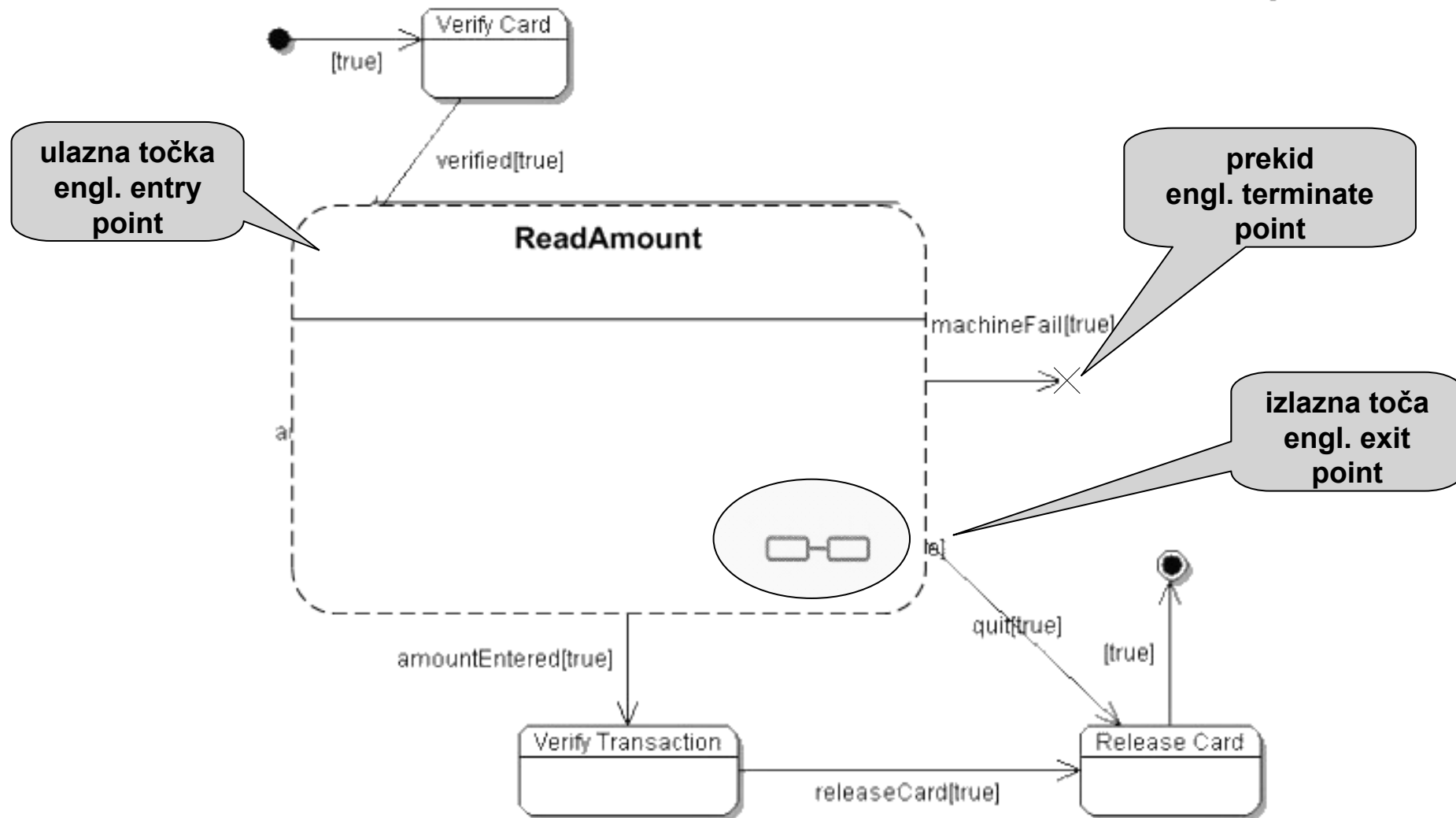
Pseudostanja UML dijagrama

- vrsta stanja u UML metamodelu koje predstavlja točke prijelaza unutar dijagrama stanja

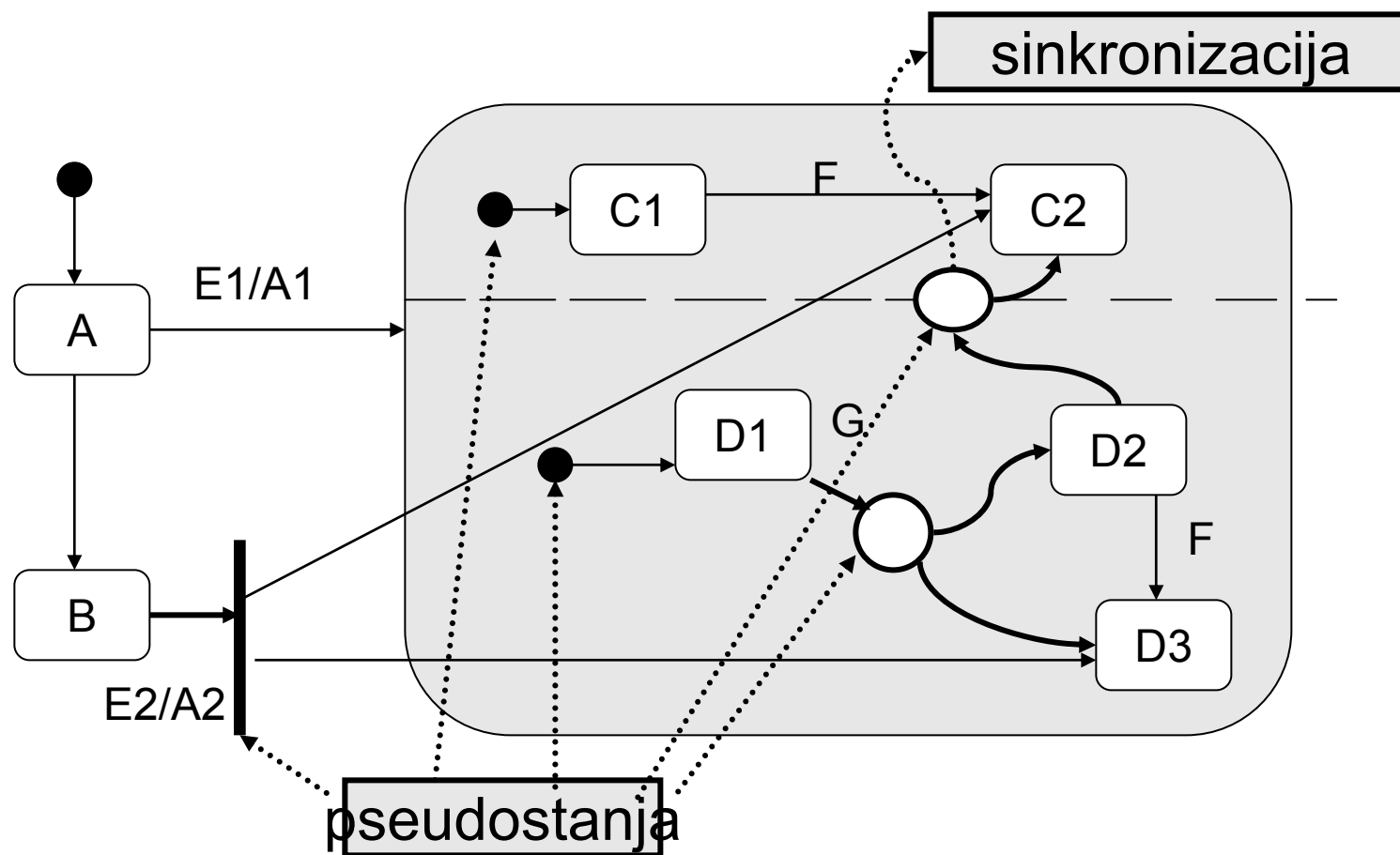
	Početno stanje - <i>Initial State</i>
	Završno stanje(a) - <i>Final State</i>
	Povijest - <i>History</i>
	Duboka povijest – Deep History
	Ulazna točka - <i>Entry Point</i>
	Izlazna točka - <i>Exit Point</i>
	Spajanje - <i>Junction Pseudo-State</i>
	<i>Izbor-Choice Pseudo-State</i>
	Završetak - <i>Terminate Pseudo-State</i>
	Grananje- Fork
	Spajanje - Join

Hijerarhija stanja

- olakšava prikaz složenih problema

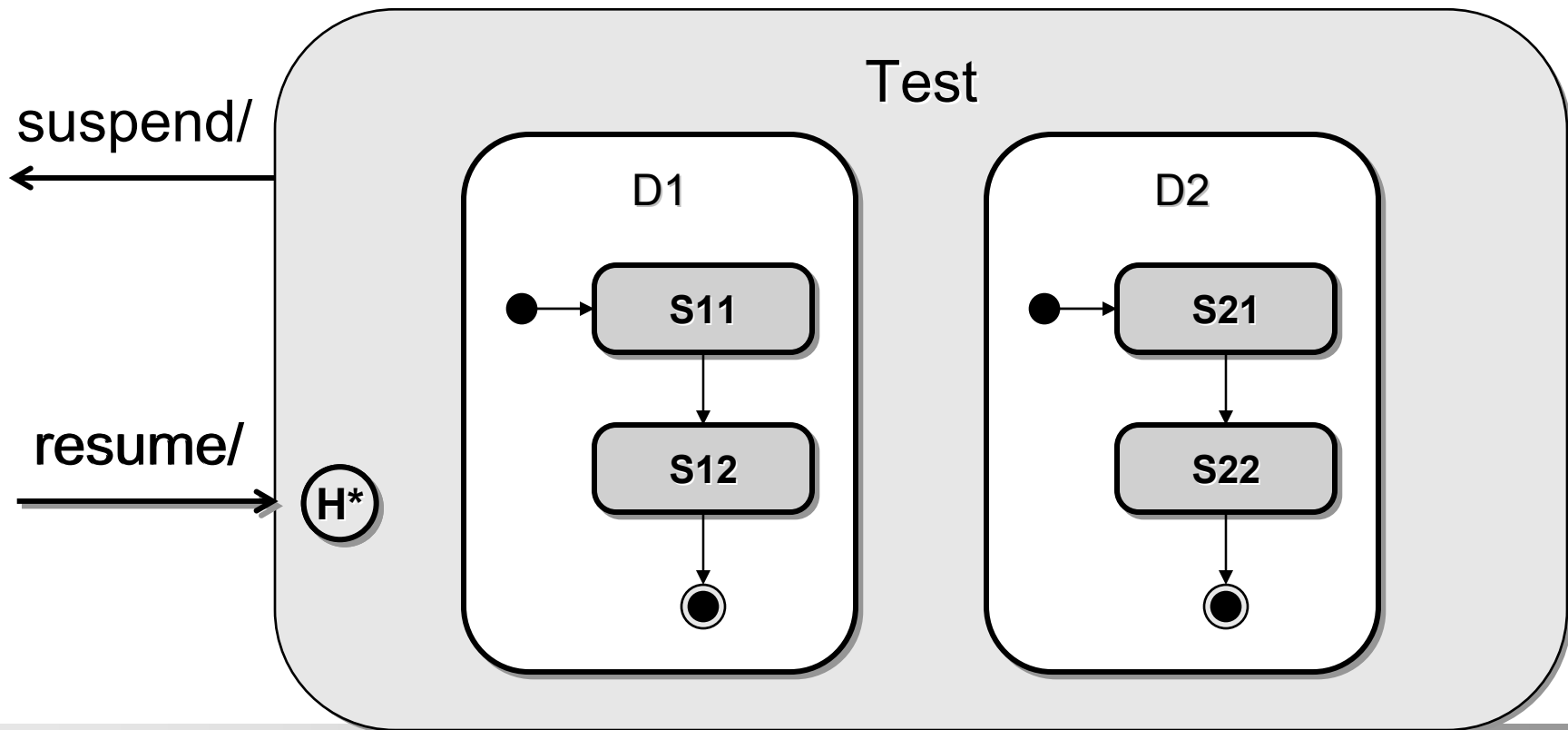


Primjer pseudostanja



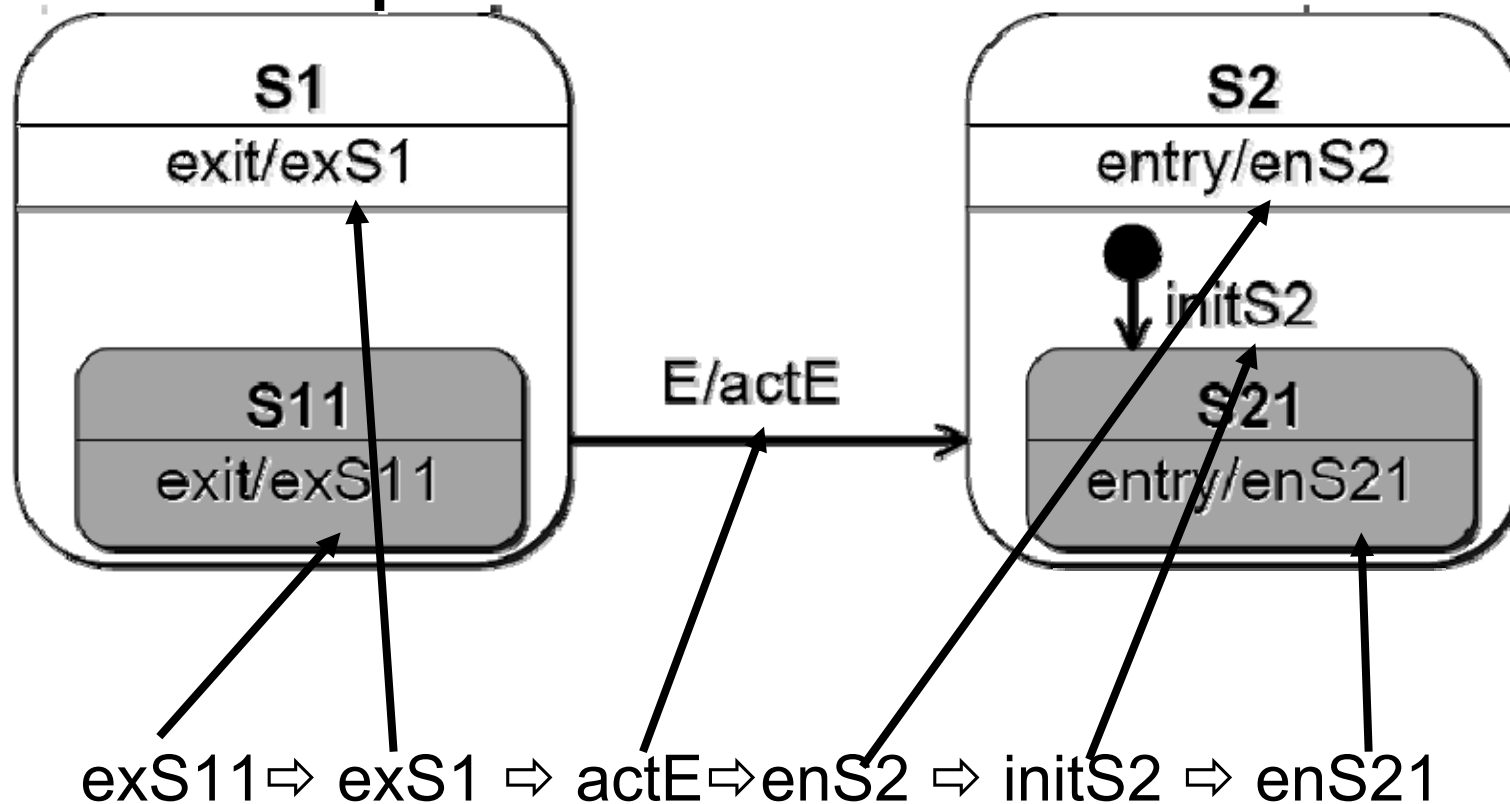
Povijest

- mogućnost povratka na prethodno stanje
 - Povijest – (Shallow) History (H)
 - povratak na posljednje stanje na istoj razini
 - Duboka povijest - Deep History (H*)
 - povratak na posljednje stanje (bez obzir na kojoj razini)

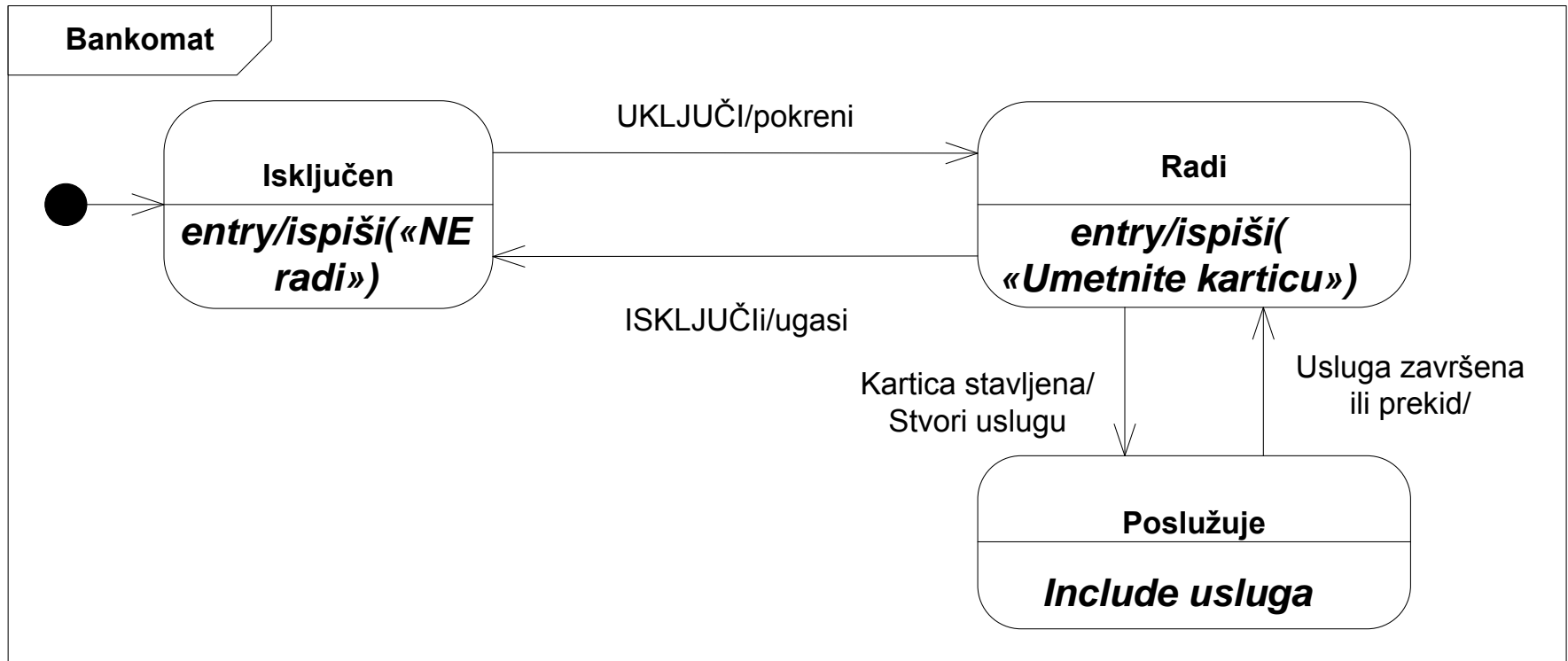


Prioritet akcija

- ulaz: izvana prema unutra
- izlaz: iznutra prema van

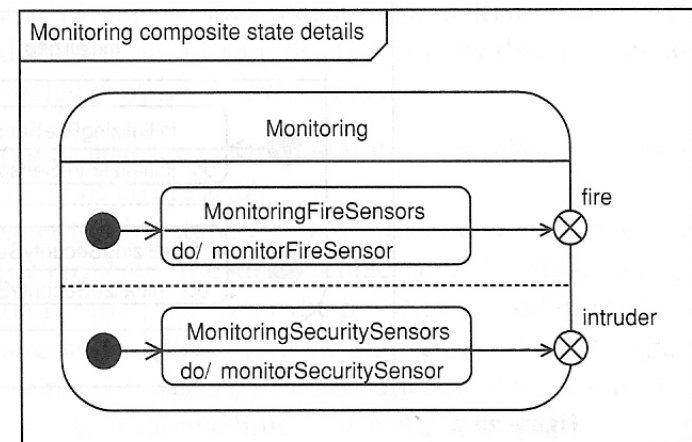
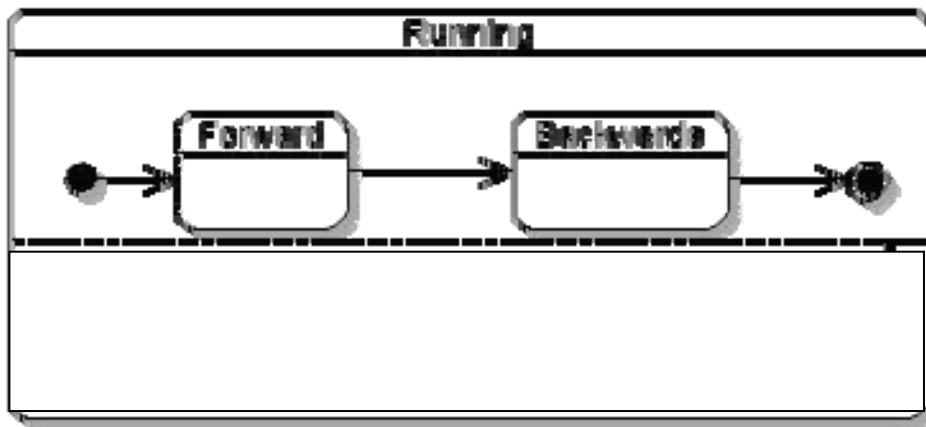


Primjer: Dijagram stanja bankomata



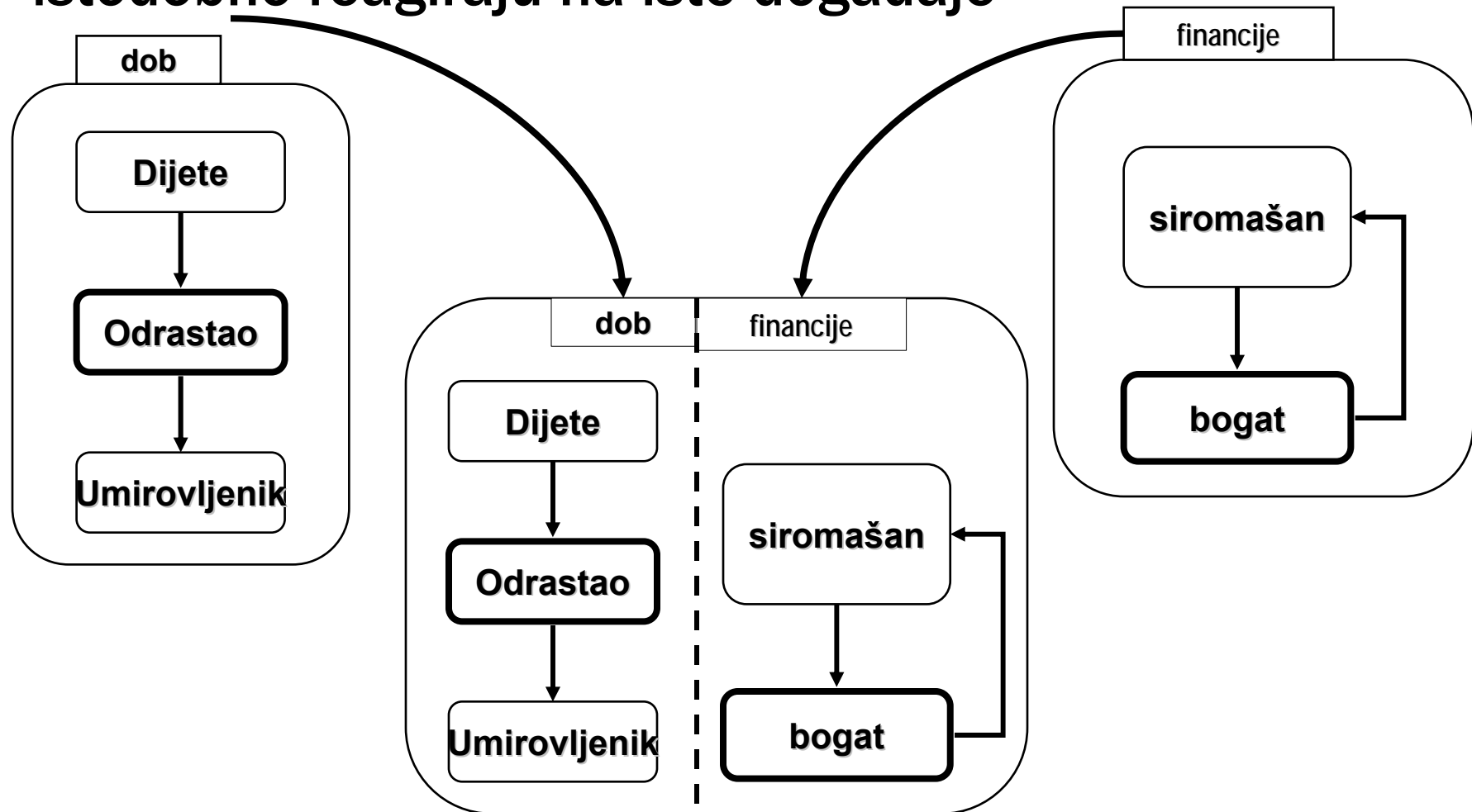
Konkurentna podstanja

- jedno stanje može imati više konkurentnih podstanja
 - višestruka perspektiva jednog objekta
 - smanjuje broj stanja



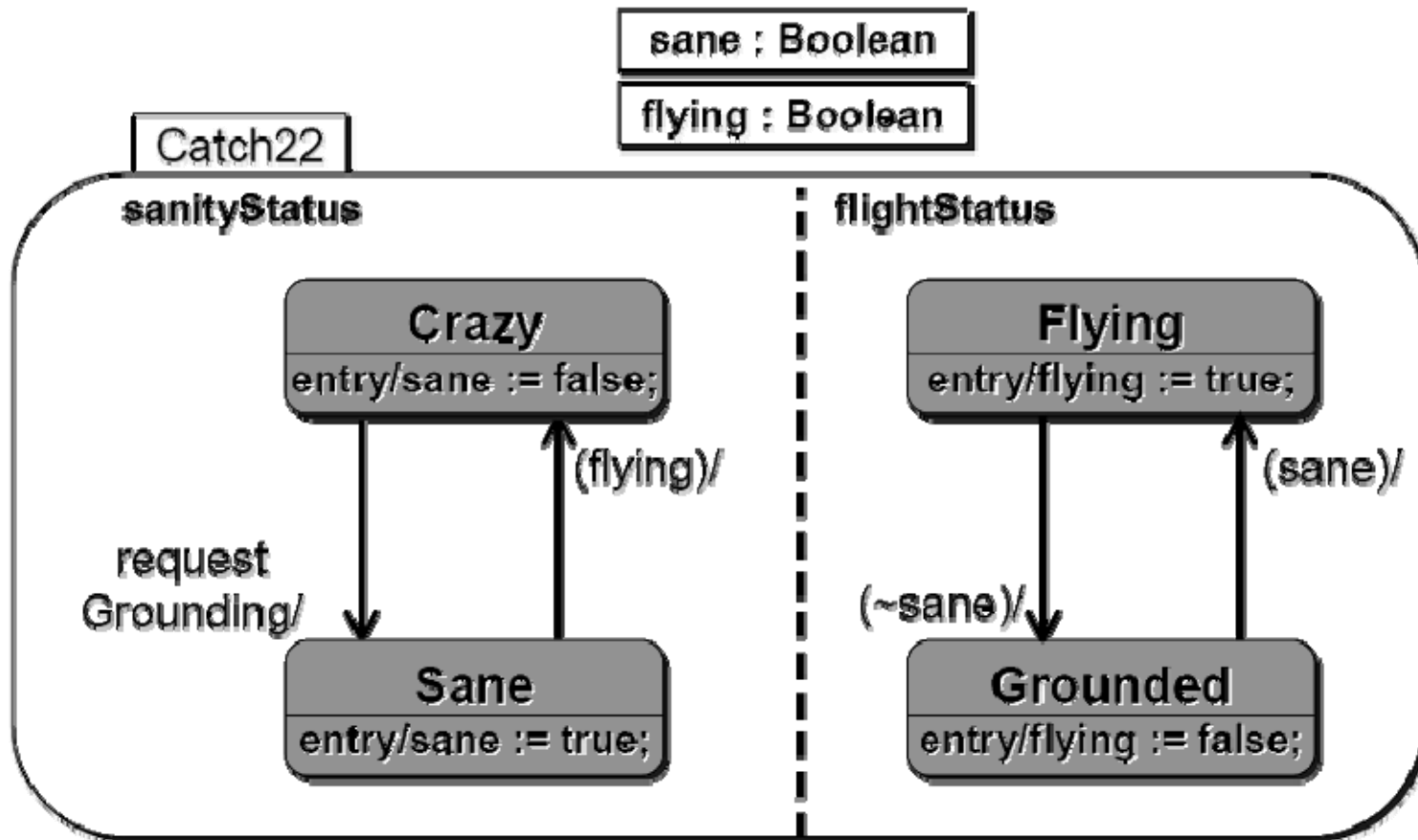
Ortogonalnost

- višestruka perspektiva istog objekta engl. orthogonality
- istodobno reagiraju na iste događaje



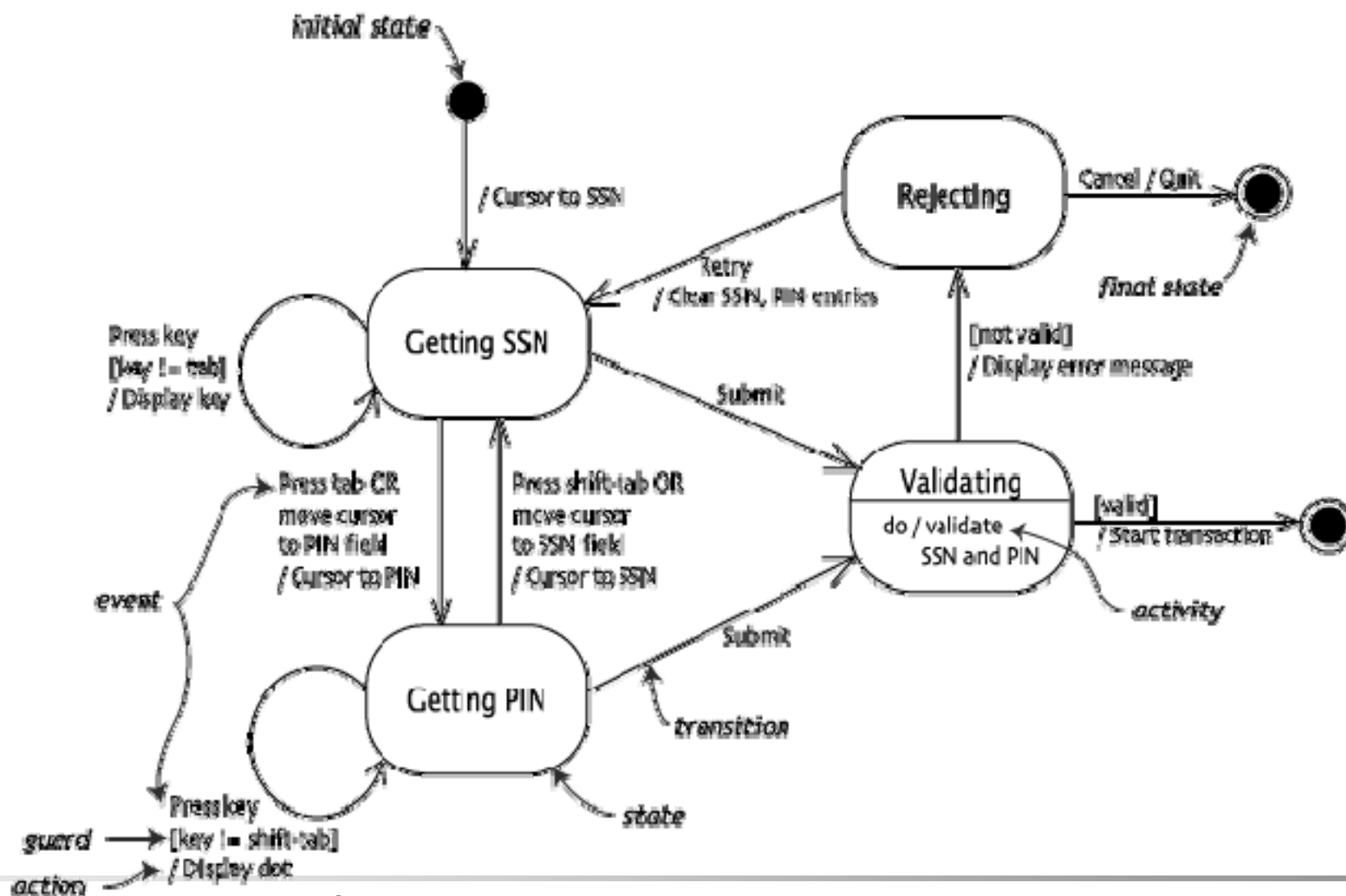
Interakcija između područja

- uporabom dijeljenih varijabli

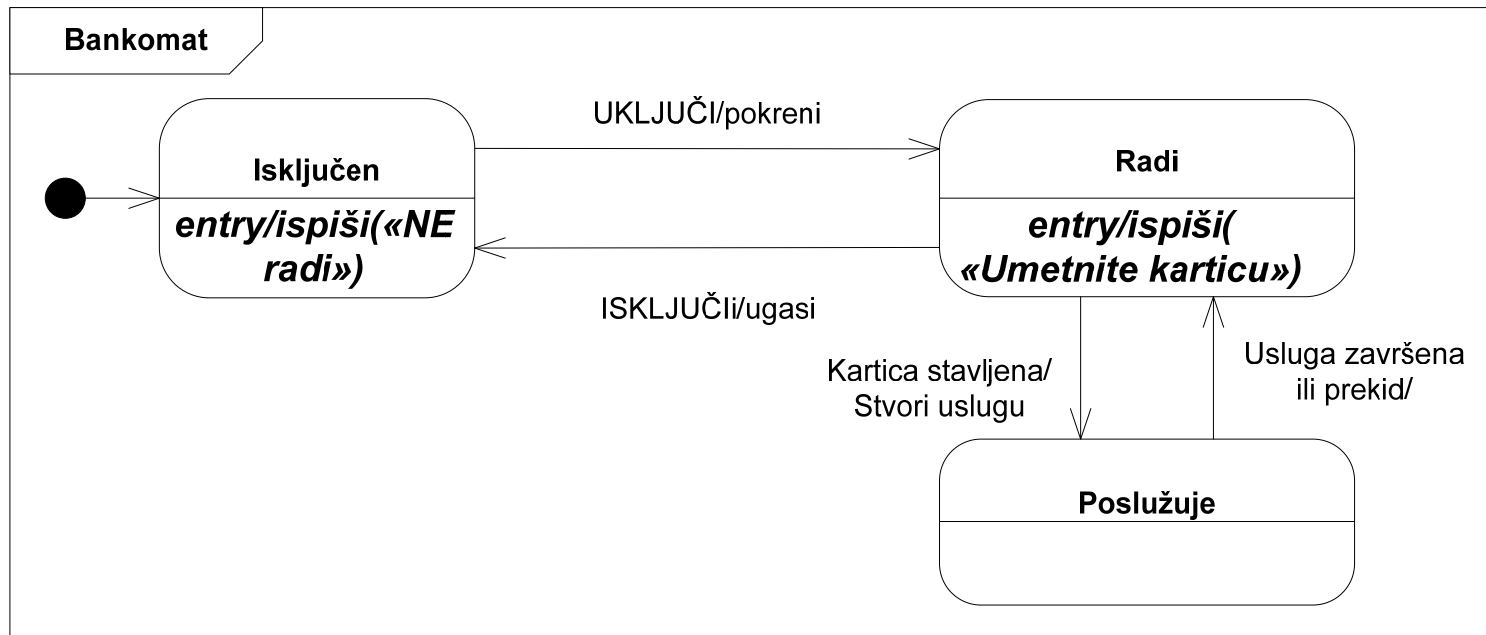


Primjer: Prijava na bankarski sustav

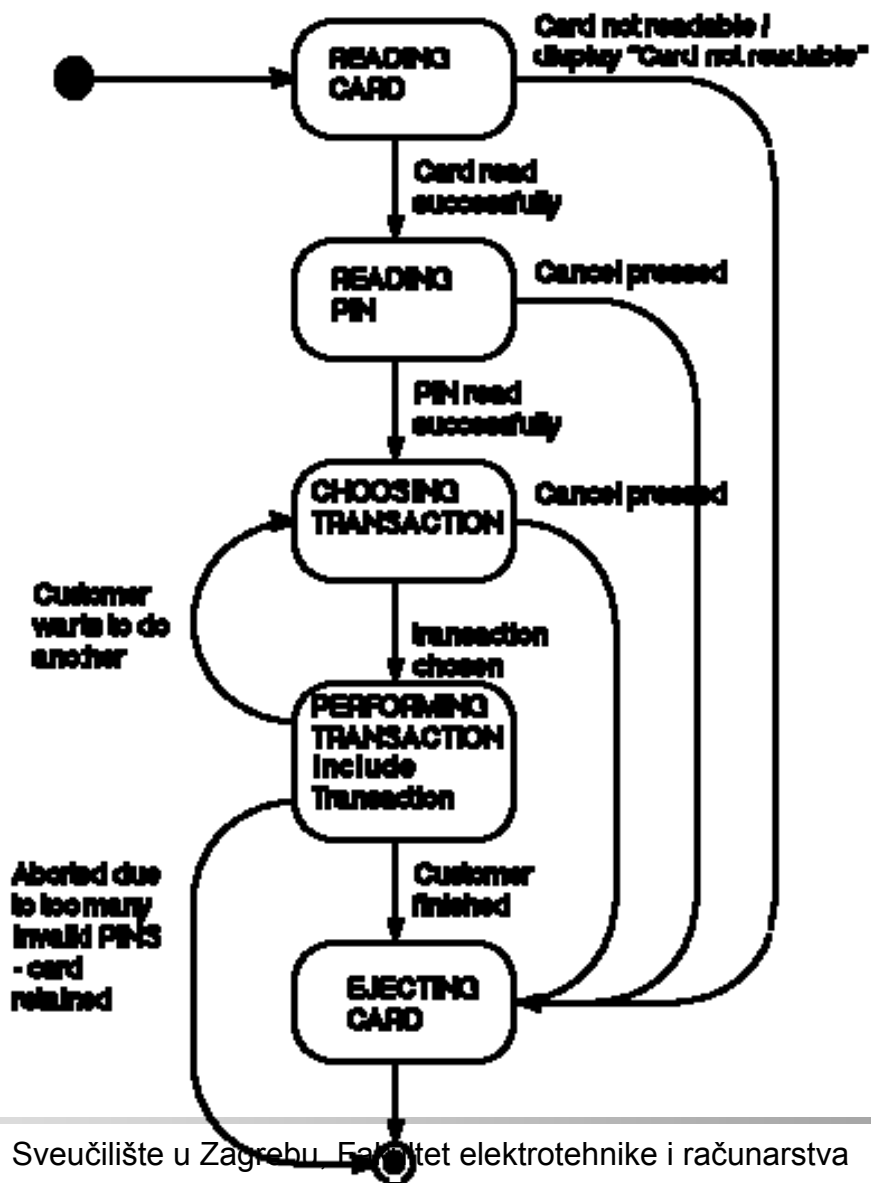
■ Prijava: unos OIB-a i PIN-a



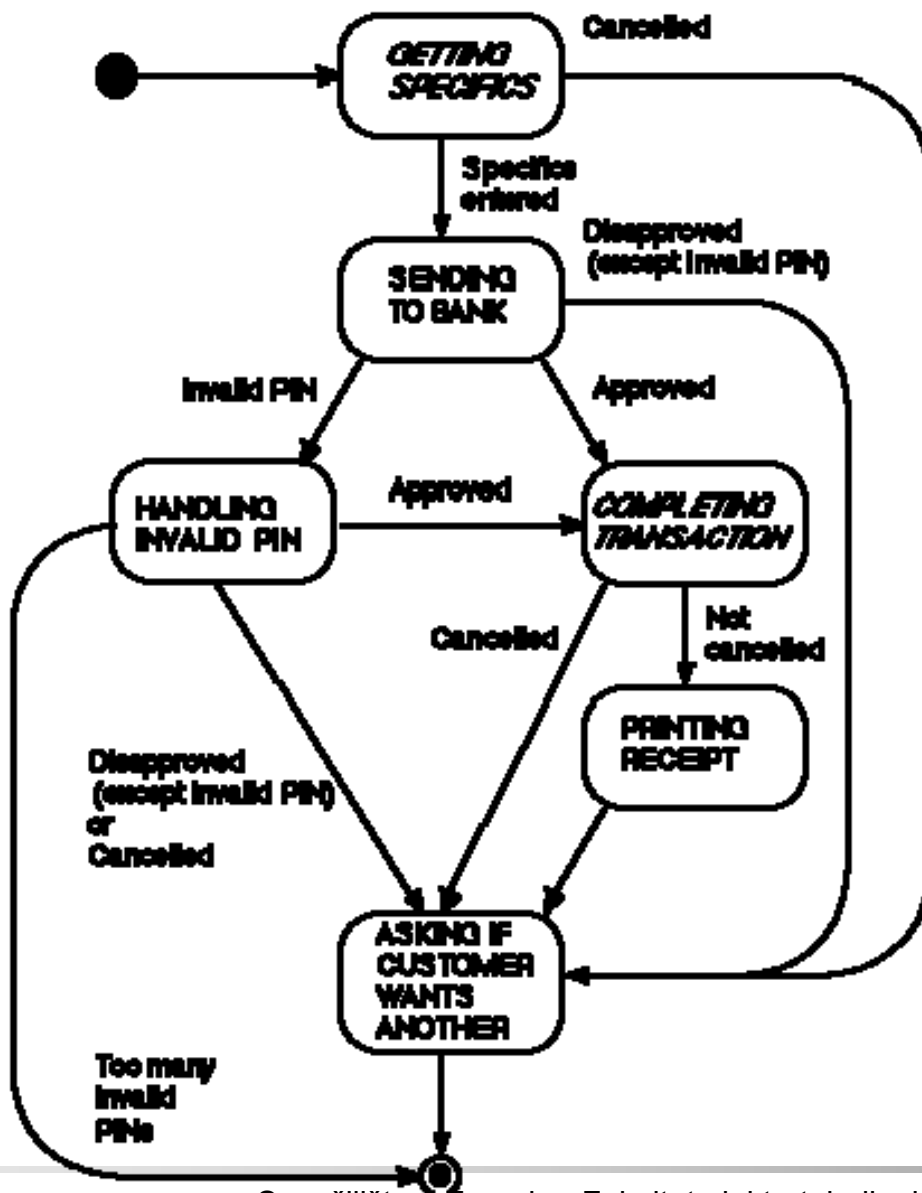
Primjer: Dijagram stanja bankomata



Primjer: Dijagram stanja usluge



Primjer: Dijagram stanja transakcije



UML dijagrami

- Obrasci uporabe
- Sekvencijski dijagram
- Komunikacijski dijagram
- Dijagram stanja
- Dijagram aktivnosti
- Dijagram komponentni
- Dijagram razmještaja
- Dijagram paketa
- Dijagram pregleda interakcije
- Vremenski dijagram
- Dijagram profila
- Dijagram razreda
- Dijagram objekata
- Dijagram složene strukture

Dijagrami aktivnosti

- engl. Activity diagrams:
- u prvoj inačici UML-a predstavljali su samo specifičan prikaz dijagrama stanja
- UML 2 definira novu semantiku zasnovanu na Petrijevim mrežama
 - povećana prilagodljivost modeliranju različitih tipova tijeka
 - jasno razdvajanje od dijagrama stanja
 - usmjereno na opis tijeka izvođenja i ponašanja sustava
- Namjena pobliže opisivanje obrazaca uporabe, dijagrama razreda, komponenti, sučelja i operacija
- Pogodni za
 - prikaz proceduralnog tijeka procesa
 - modeliranje poslovnih zahtjeva (procesa)
 - prikaz paralelnosti

Primjena dijagrama aktivnosti

- za opis upravljačkog i podatkovnog toka
- analiza
 - oblikovanje tijeka obrazaca uporabe
 - oblikovanje tijeka između različitih obrazaca uporabe
- oblikovanje:
 - detalja operacija
 - detalja algoritama
- modeliranje poslovnih procesa

Elementi dijagrama aktivnosti

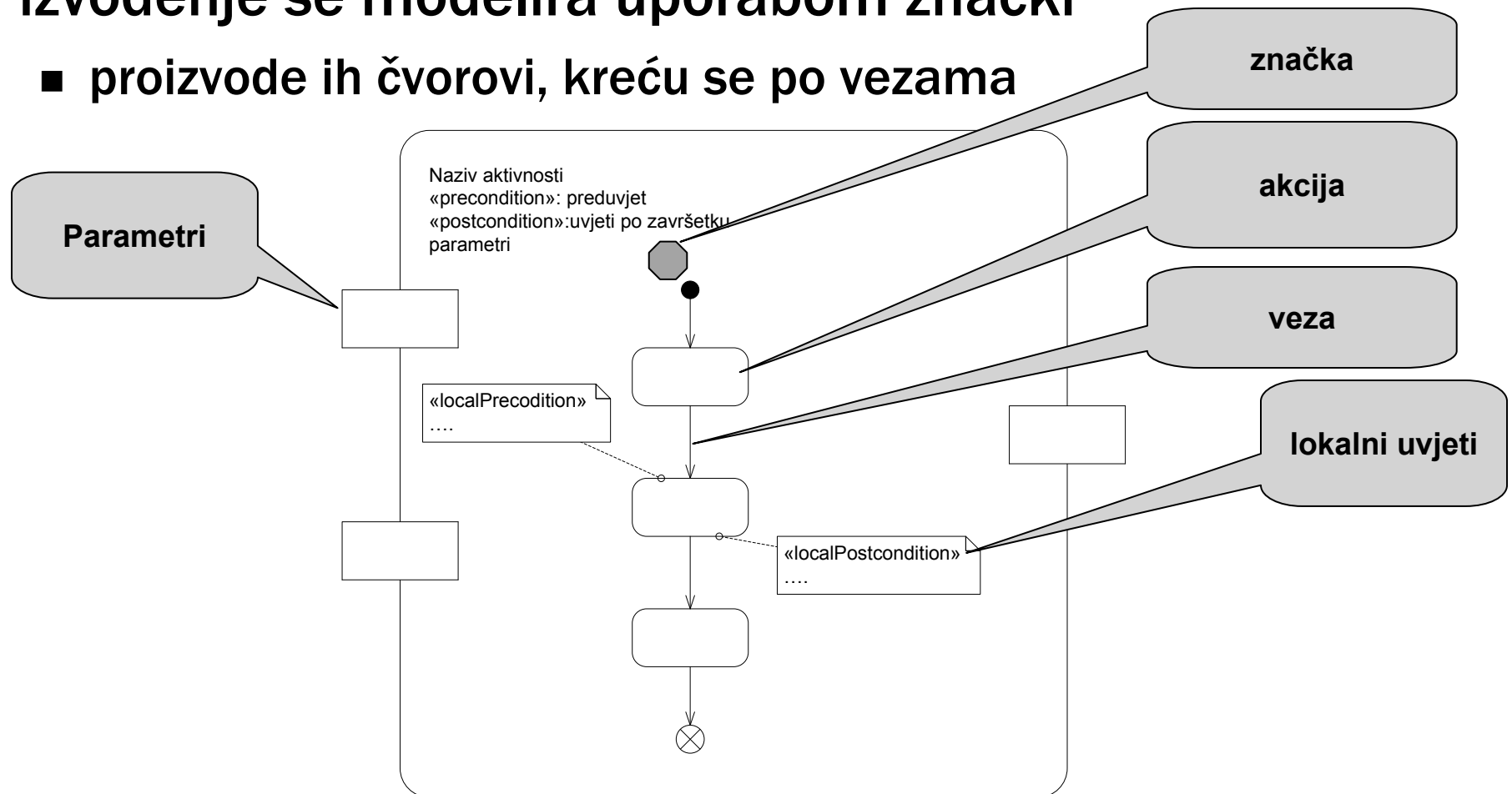
- prilagođeni opisu tijeka upravljanja i podataka
- Čvorovi – engl. activity nodes, nodes:
 - čvor akcije – engl. Action nodes
 - nedjeljive aktivnosti
 - upravljački čvorovi –engl. Control nodes
 - upravljanje tijekom
 - objekti – engl. Object nodes
 - predstavljaju objekte u aktivnostima
- Veze – engl. Activity edges
 - upravljački tijek – engl. Control flows
 - predstavlja tijek upravljanja aktivnostima
 - tijek objekta – engl. Object flows
 - tijek objekta kroz aktivnosti

Aktivnosti

- **Aktivnost – engl. Activity**
 - obuhvaća više čvorova i veza koji predstavljaju odgovarajući slijed zadataka
- **Akcija – engl. Action**
 - kratkotrajno, neprekidivo ponašanje unutar čvora akcije
- **za sve aktivnosti i akcije mogu biti definirani odgovarajući uvjeti prije izvođenja (engl. preconditions) i nakon izvođenja (engl. postconditions)**
- **Značke – engl. Tokens**
 - dio semantike bez grafičkog prikaza
 - značka može predstavljati:
 - upravljački tijek
 - objekt
 - podatak
 - kreću se od izvorišta prema odredištu vezama ovisno o:
 - ispunjenim uvjetima izvornog čvora
 - postavljenim uvjetima veza (engl. Edge guard conditions)
 - preduvjetima ciljnog čvora

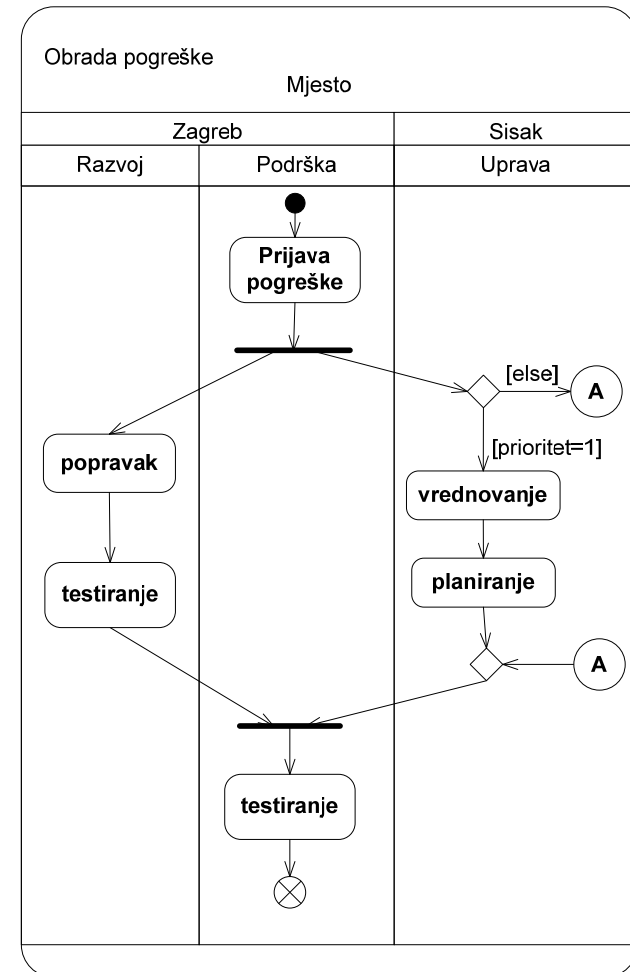
Aktivnost

- modelira ponašanje kao niz akcija
- izvođenje se modelira uporabom znački
 - proizvode ih čvorovi, kreću se po vezama



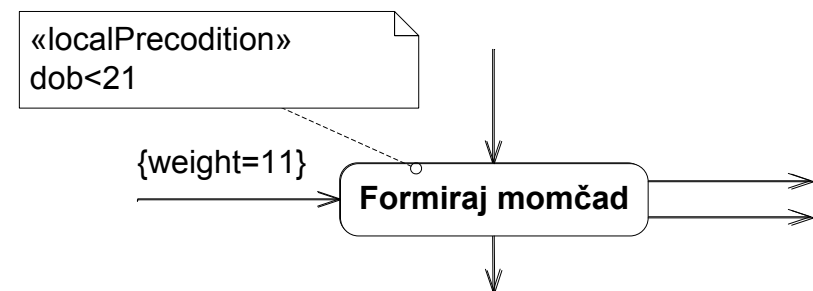
Podjela aktivnosti

- dijagram aktivnosti može biti podijeljen u particije (engl. swimlanes)
 - horizontalno, vertikalno, proizvoljno
 - hijerarhija
- ne utječu tijek odvijanja aktivnosti
 - način grupiranja



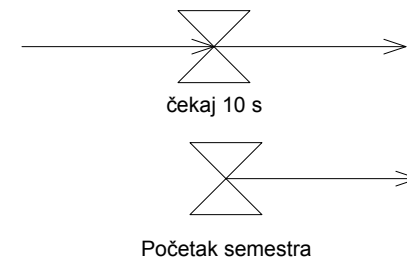
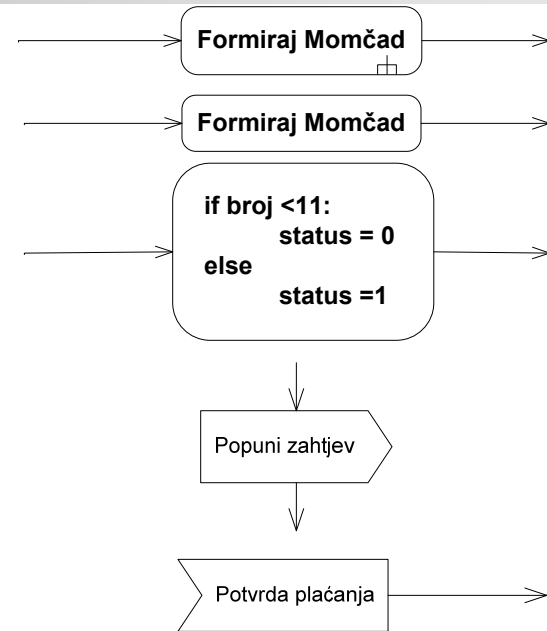
Čvor akcije

- **započinje izvođenje kada:**
 - postoji odgovarajući broj znački na svim ulazima
 - zadovoljeni su svi lokalni preduvjeti akcije
- **nakon izvođenja provjerava se zadovoljavanje uvjeta te proslijeđuju značke na sve izlaze**
- **tipovi:**
 - obrade osnovnih operacija, pozivanje složenih aktivnosti ili ponašanja - engl. call action
 - komunikacijske akcije
 - slanje signala
 - Manipuliranje objektima





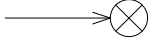
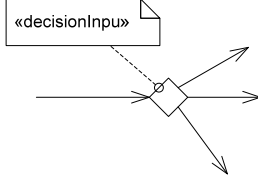
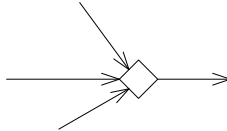

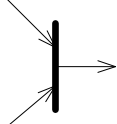
Tipovi čvorova akcije

- obrada osnovnih operacija, pozivanje složenih aktivnosti ili ponašanja - engl. call action
- slanje signala – engl. send signal
 - asinkroni signal
- prihvaćanje događaja – engl. accept event
 - čekanje na neki događaj
- vremenski događaja – engl. time event
 - definirana vremenskim izrazom



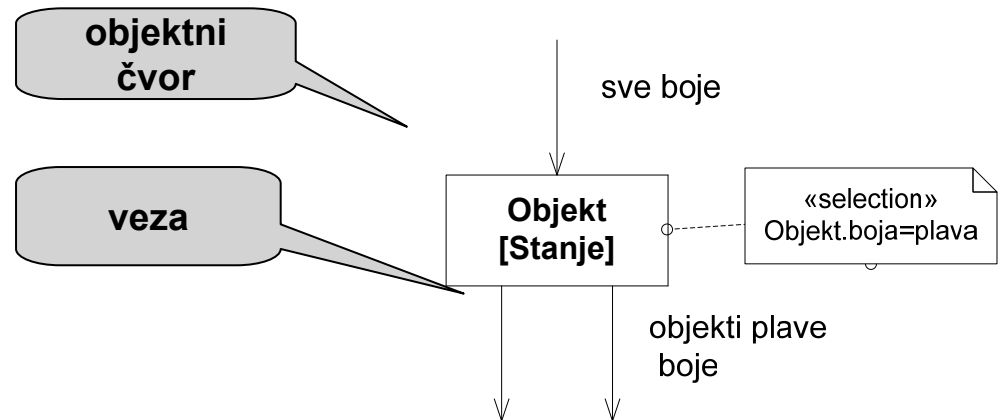
upravljački čvorovi

- rukuju upravljanjem aktivnostima
 - početak i završetak
 - odluke
 - paralelnost

	Početni čvor - <i>Initial node</i>
	Završni čvor aktivnosti - <i>Activity final node</i>
	Završetak toka - <i>Final flow node</i>
	Čvor odluke - <i>Decision node</i>
	Spajanje čvorova - <i>Merge node</i>
	Grananje toka- <i>Fork node</i>
	Spajanje - <i>Join node</i> Sinkronizira više paralelnih tijekova

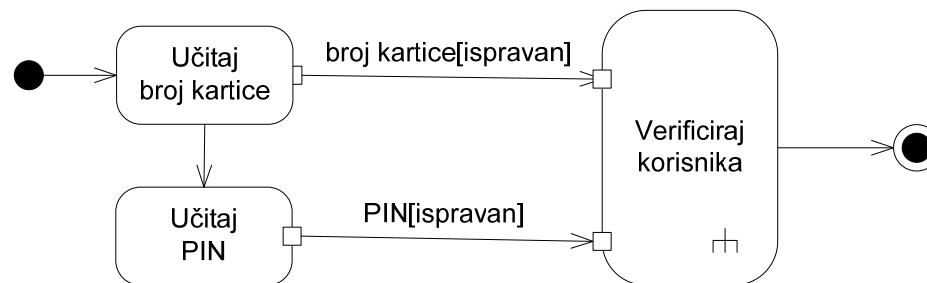
Objektni čvorovi

- prikazuju podatke i objekte
 - objektni čvor ukazuje na raspoloživost u danoj točki aktivnosti
 - uobičajeno označeni imenom razreda i predstavljaju instancu
- ulazne i izlazne veze predstavljaju tok objekta
 - opisuju kretanje objekta unutar aktivnosti
- objekte stvaraju i upotrebljavaju akcijski čvorovi
- objektne značke
 - kada objektni čvor primi značku, nudi ju na svim izlaznim vezama koje se natječu za značku. Značku dobiva prva veza koja ju je spremna prihvatiti
- objektni čvorovi se ponašaju kao međuspremnici (engl. buffer)
 - pohranjuju objektne značke

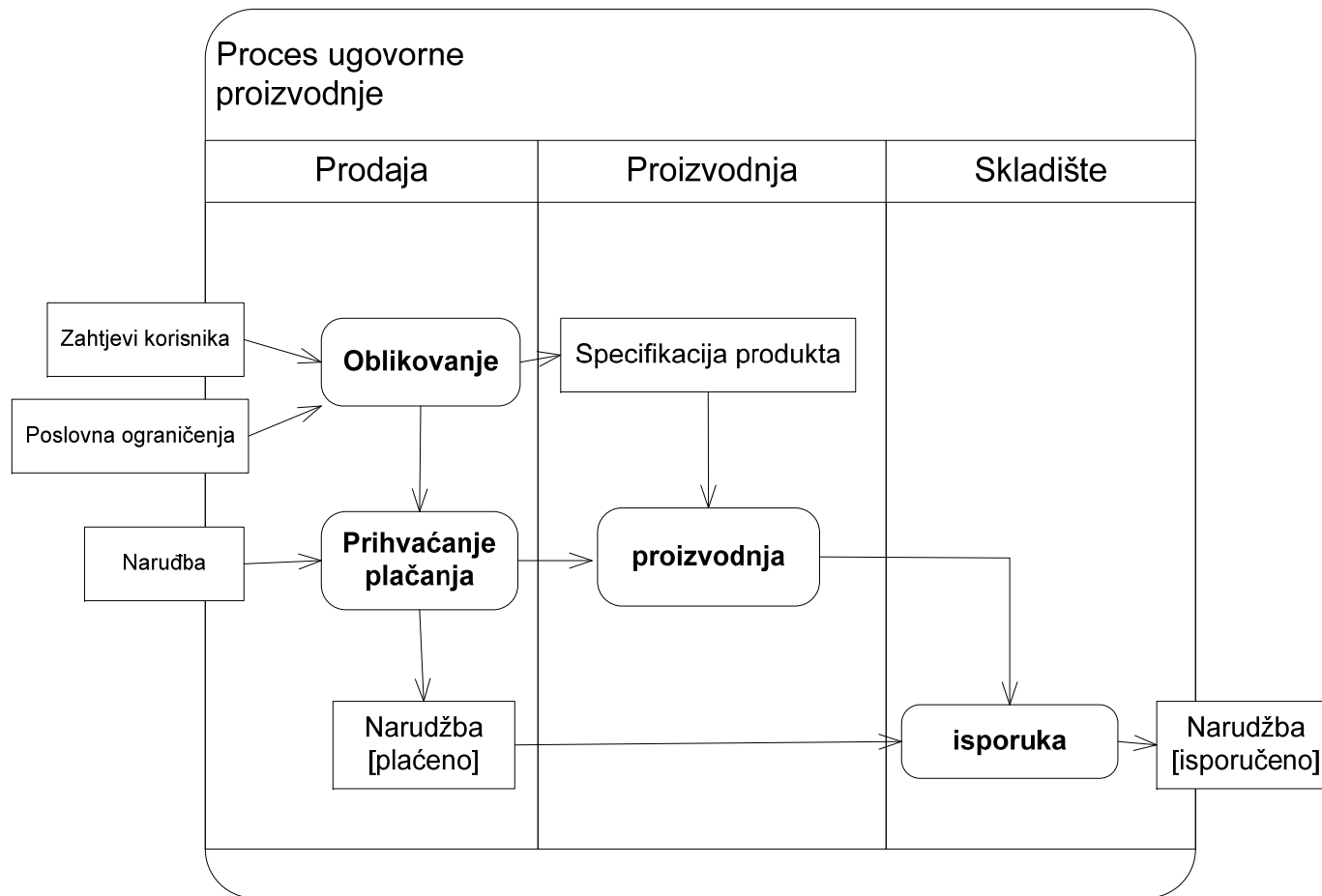


Priključnice

- engl. Pins
- objektni čvor s jednim ulazom ili izlazom prema čvoru akcije
- pojednostavljuju grafički prikaz dijagrama aktivnosti s većim brojem objektnih čvorova

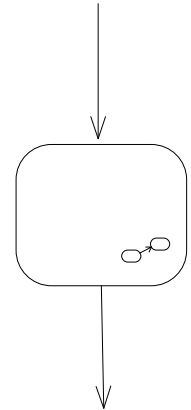


Primjer:



Podaktivnost

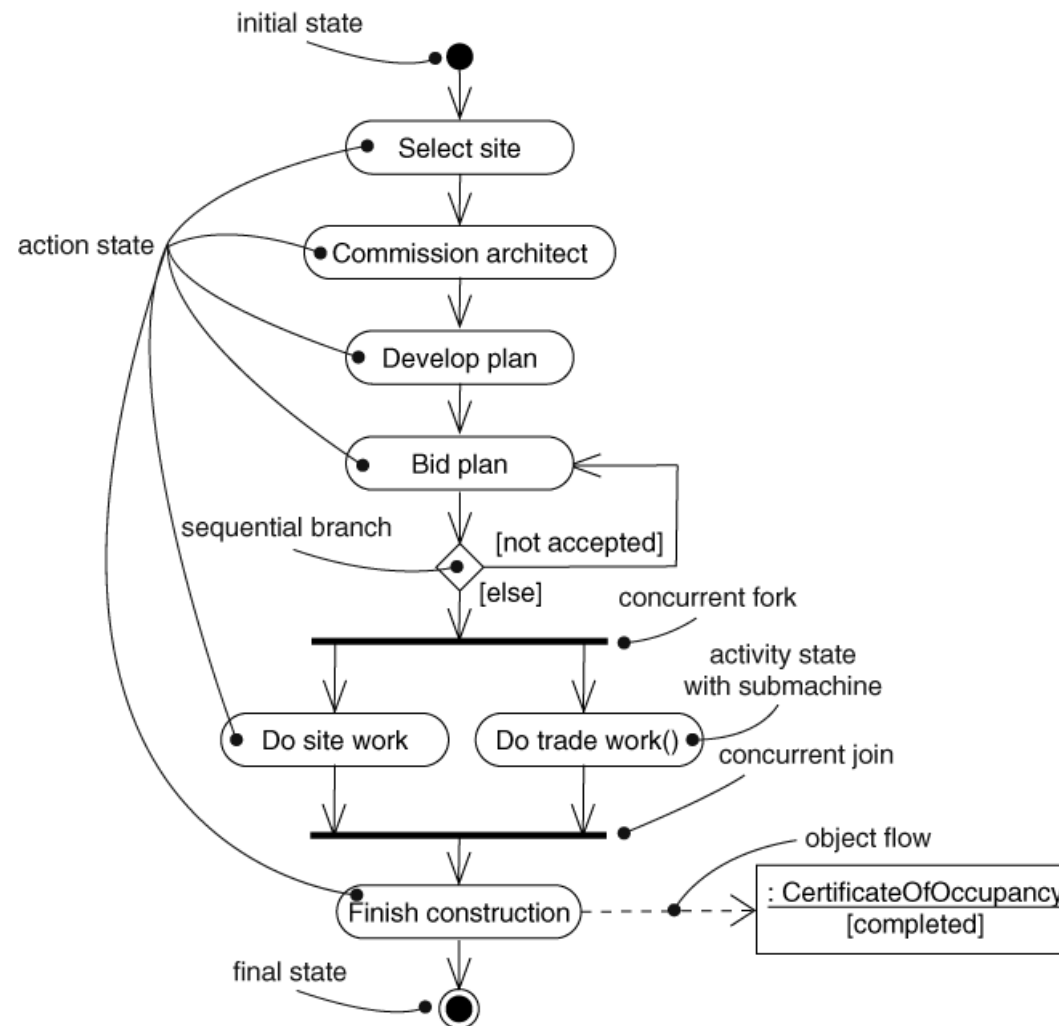
- engl. subactivity
- namjena funkcionalne dekompozicije
- ugnježđenje drugog dijagrama aktivnosti
- ulaskom u podaktivnost pokreće se njegova početna akcija
- po završetku podaktivnosti, nastavlja se izvođenje prethodne aktivnosti



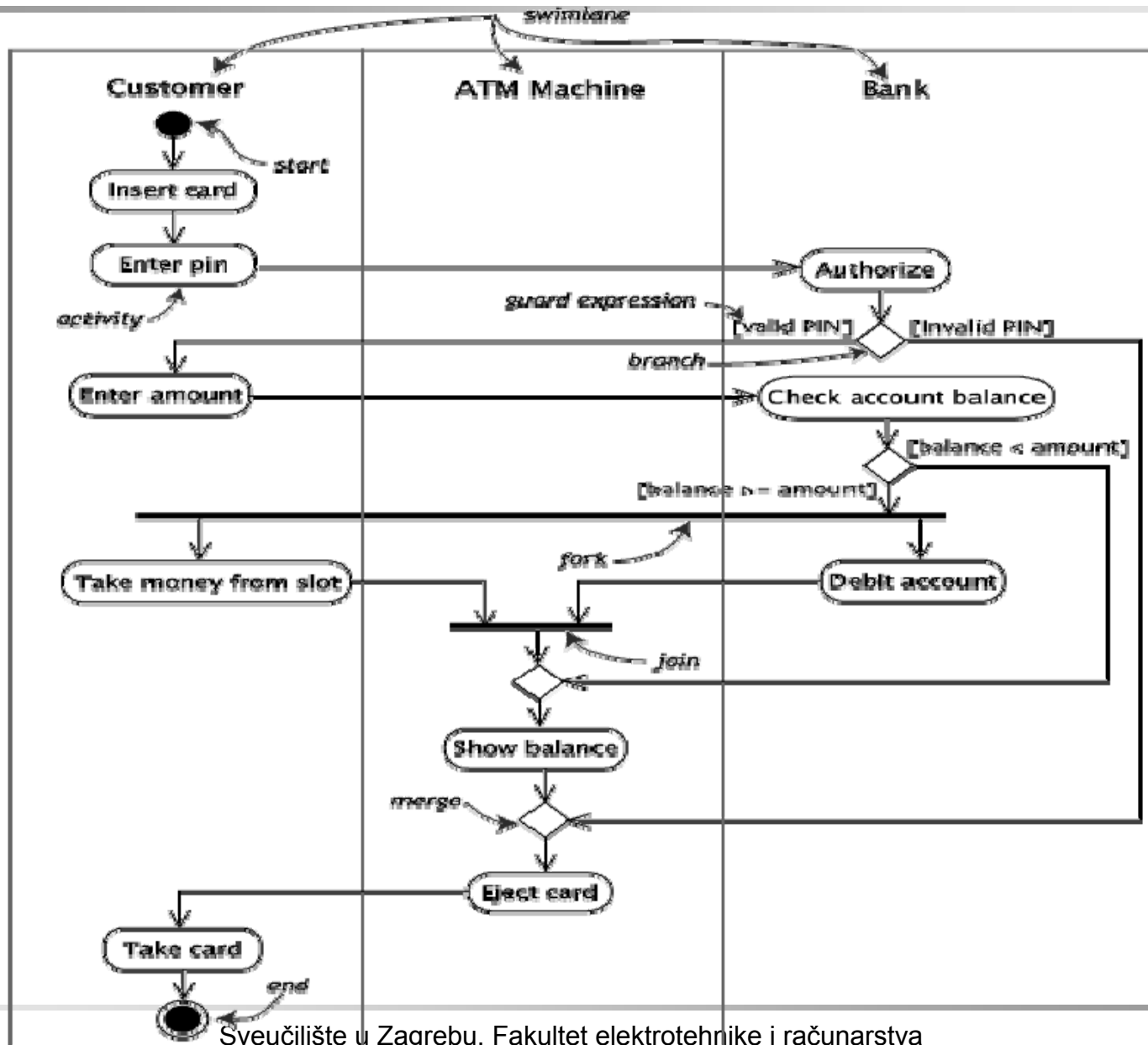
Primjena dijagrama aktivnosti

- pogodno za slučajeve:
 - kada ponašanje ne zavisi o velikom broju vanjskih događaja
 - postoje definirani koraci bez prekida
 - kada imamo tijek objekata između koraka
 - kada želimo pojasniti aktivnosti

Primjer: izgradnja kuće



Primjer: bankomat



UML dijagrami

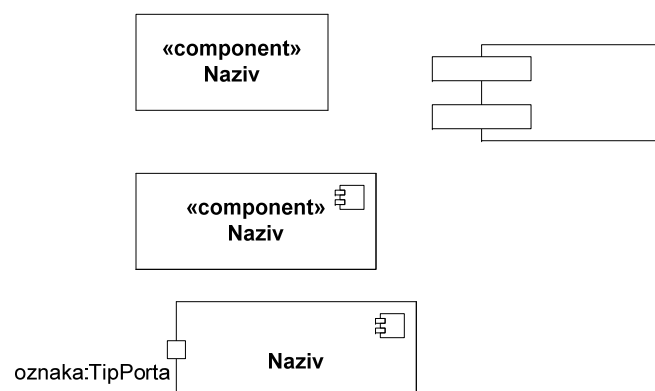
- Obrasci uporabe
- Sekvencijski dijagram
- Komunikacijski dijagram
- Dijagram stanja
- Dijagram aktivnosti
- Dijagram komponentni
- Dijagram razmještaja
- Dijagram paketa
- Dijagram pregleda interakcije
- Vremenski dijagram
- Dijagram profila
- Dijagram razreda
- Dijagram objekata
- Dijagram složene strukture

Programske komponente

- **engl. software component**
 - “are binary units of independent production, acquisition, and deployment that interact to form a functioning system”, C. Szyperski
 - zamjenjivi, ponovo iskoristivi dijelovi koda.
- **U programskom inženjerstvu zasnovanom na komponentama sustav se integrira**
 - višestrukom uporabom postojećih komponentata,
 - uporabom komercijalnih, gotovih komponentata (engl. commercial-of-the-shelf COTS) ili
 - modificiranim komercijalnim komponentama (engl. modified-of-the-shelf MOTS).

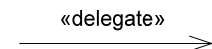
Dijagram komponenti

- engl. Component Diagram
- UML komponente predstavljaju fizičke modularne, zamjenjive jedinice s dobro definiranim sučeljem
 - obuhvaća neki sadržaj kojem se pristupa putem sučelja
 - definira ponašanje kroz ponuđena i zahtijevana sučelja
 - Razred - logička apstrakciju
 - attribute i operacije
 - može im se pristupiti izravno
 - Komponenta - fizička stvar, fizičko obuhvaćanje logičkih apstrakcija
 - operacije
 - može im se pristupiti samo kroz sučelja
- Namjena komponentnog dijagrama je prikaz komponenata, interne strukture i odnosa prema okolini
- Komponente mogu sadržavati druge komponente te na taj način prikazivati internu strukturu.
- mogu imati definirane portove (engl. port)
 - točka interakcije komponente s okolinom
- instanca komponente
 - ime_komponente: Tip_komponente

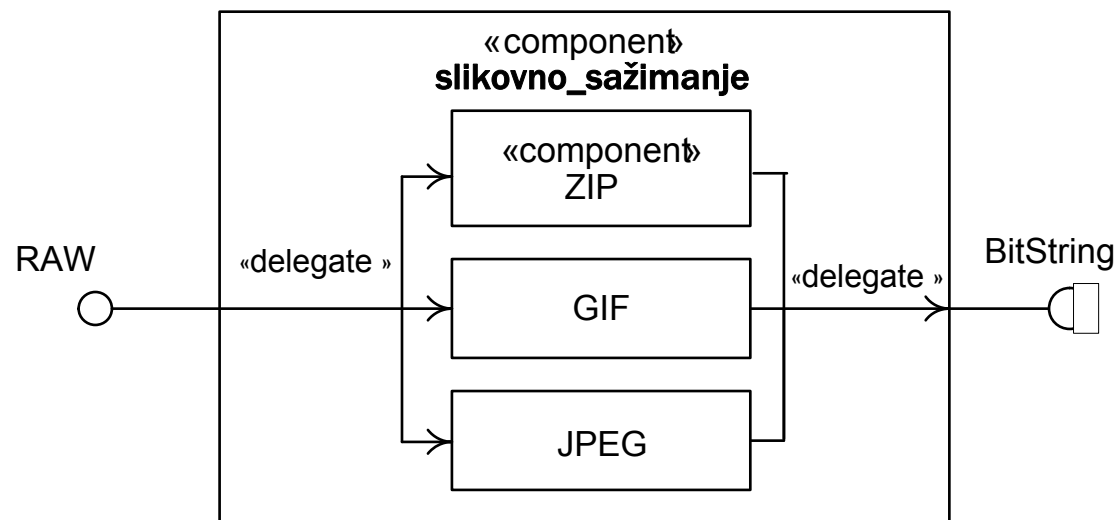


Sučelja/Konektori

- engl. Interface
- imenovan skup javno vidljivih atributa i apstraktnih operacija
 - implementaciju osigurava komponenta/razreda
- Komponente i klase ostvaruju sučelja
 - uključivanjem atributa i implementacijom operacija
- Tipovi sučelja:
- ponuđeno/implementirano sučelje – engl. provided interface
 - ostvaruje ga razred ili komponenta
 - usluga koja se nudi
- zahtijevano sučelje – engl. required interface
 - potrebno klasi ili komponenti
 - ono što je potrebno za njezin rad
- tipovi konektora
- spojnica - engl. assembly connector
 - povezuje dvije komponente
 - “ball-and-socket”, “lollipop”
- delegacija - engl. *delegation* connector
 - povezuje sučelje komponente s internom strukturom

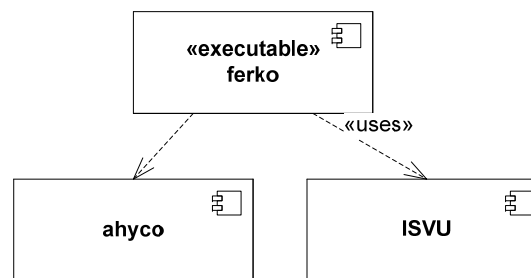


Primjer interne strukture komponente



Ovisnost

- engl. dependency
- ova relacija između komponenti upotrebljava se kada jedna komponenta zahtijeva uporabu druge radi potpunog ostvarenja implementacije
- predstavlja se crtkanom strelicom od ovisne komponente



Artefakti i stereotipi

■ artefakti

- iz razvojnog procesa: izvorni kod, podaci ...
 - engl. Source Component
- binarne komponente za isporuku
 - engl. Binary Component (object code file, static or dynamic libraries)
- izvršne komponente
 - engl. Executable component

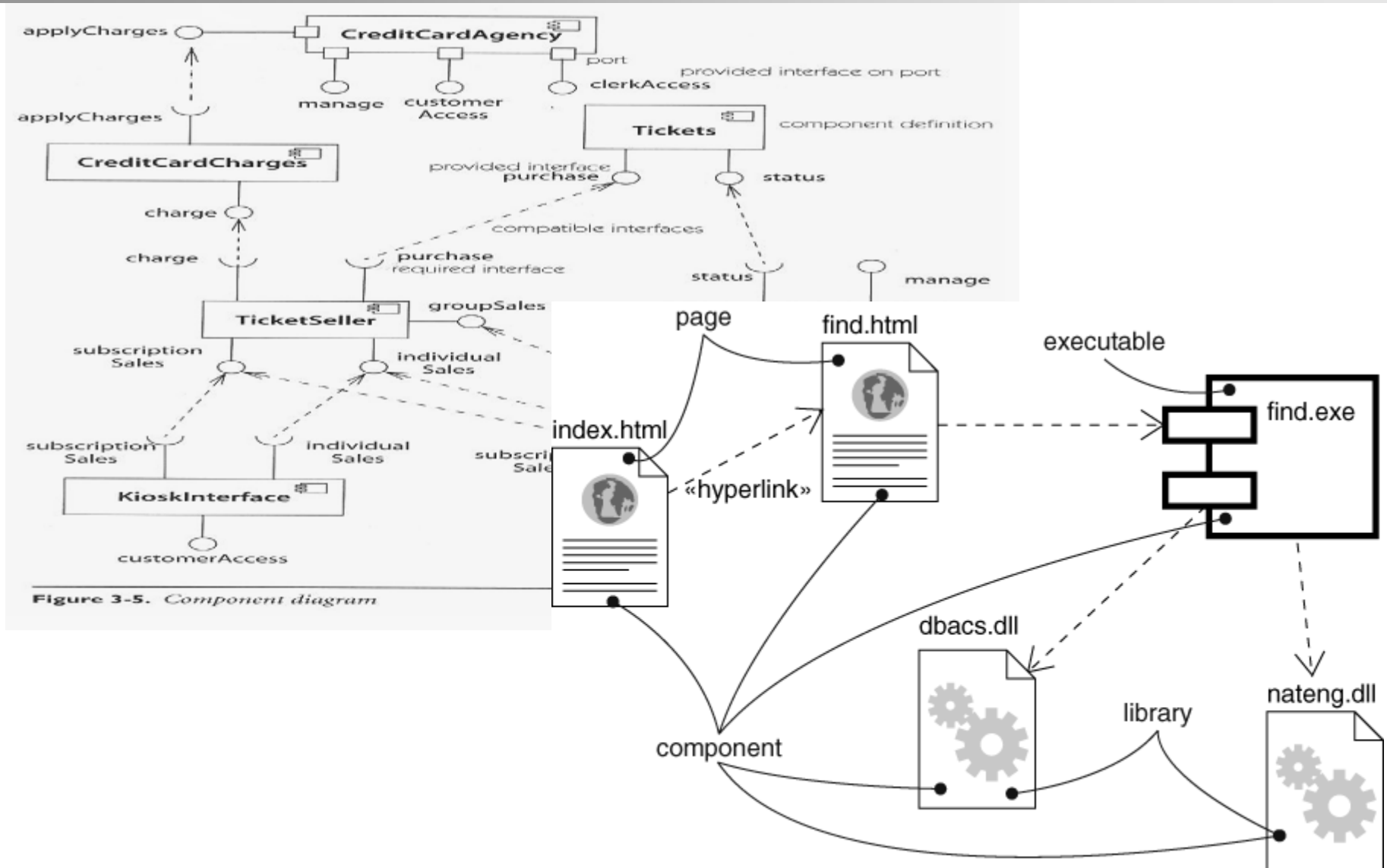
■ stereotipi

- **executable** - komponenta koja se može izvršavati
- **library** - statička ili dinamička biblioteka
- **file** - datoteka s proizvoljnim sadržajem
- **document** - dokument
- **script** - skript
- **source** - datoteka sa izvornim kodom

Primjena komponentnih dijagrama

- **statički model programskih komponenti**
 - ponovna uporaba i zamjena komponenti dijelova
- **oblikovanje programskih komponenti**
 - arhitekturni model
 - detalji oblikovanja
 - odnos s okolinom
- **oblikovanje interne strukture komponenti**

Primjer

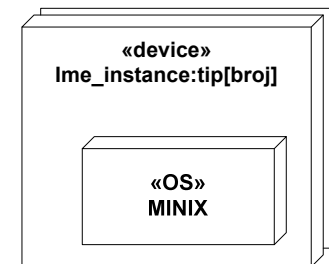


UML dijagrami

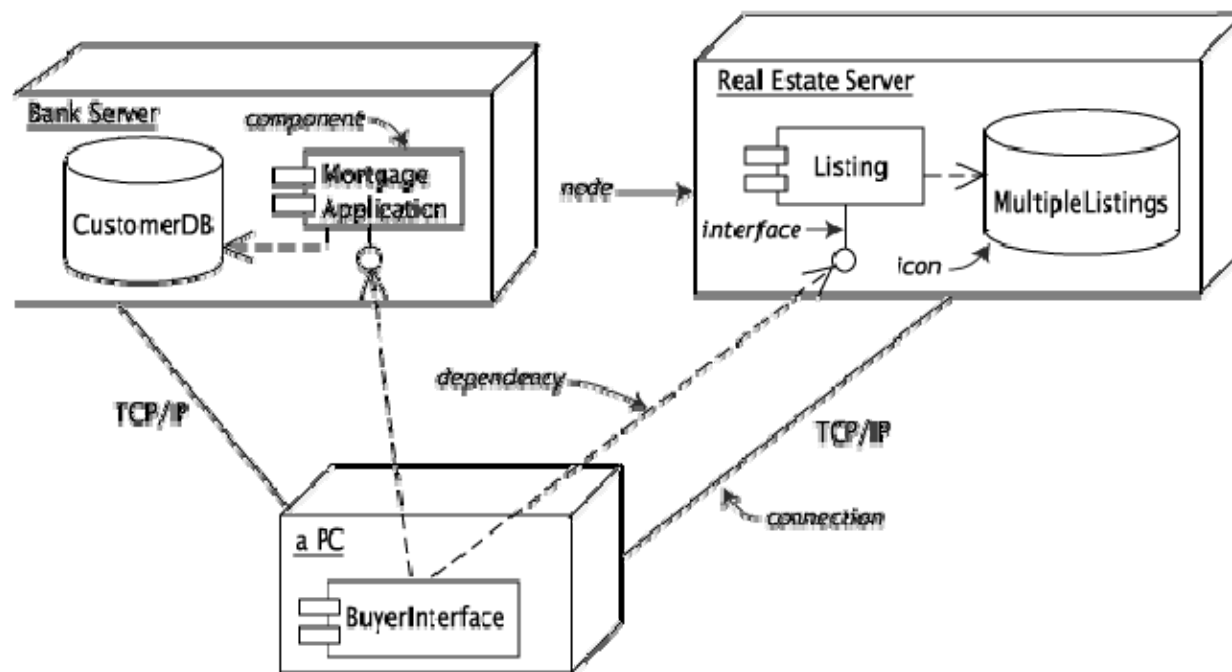
- Obrasci uporabe
- Sekvencijski dijagram
- Komunikacijski dijagram
- Dijagram stanja
- Dijagram aktivnosti
- Dijagram komponentni
- Dijagram razmještaja
- Dijagram paketa
- Dijagram pregleda interakcije
- Vremenski dijagram
- Dijagram profila
- Dijagram razreda
- Dijagram objekata
- Dijagram složene strukture

Dijagram razmještaja

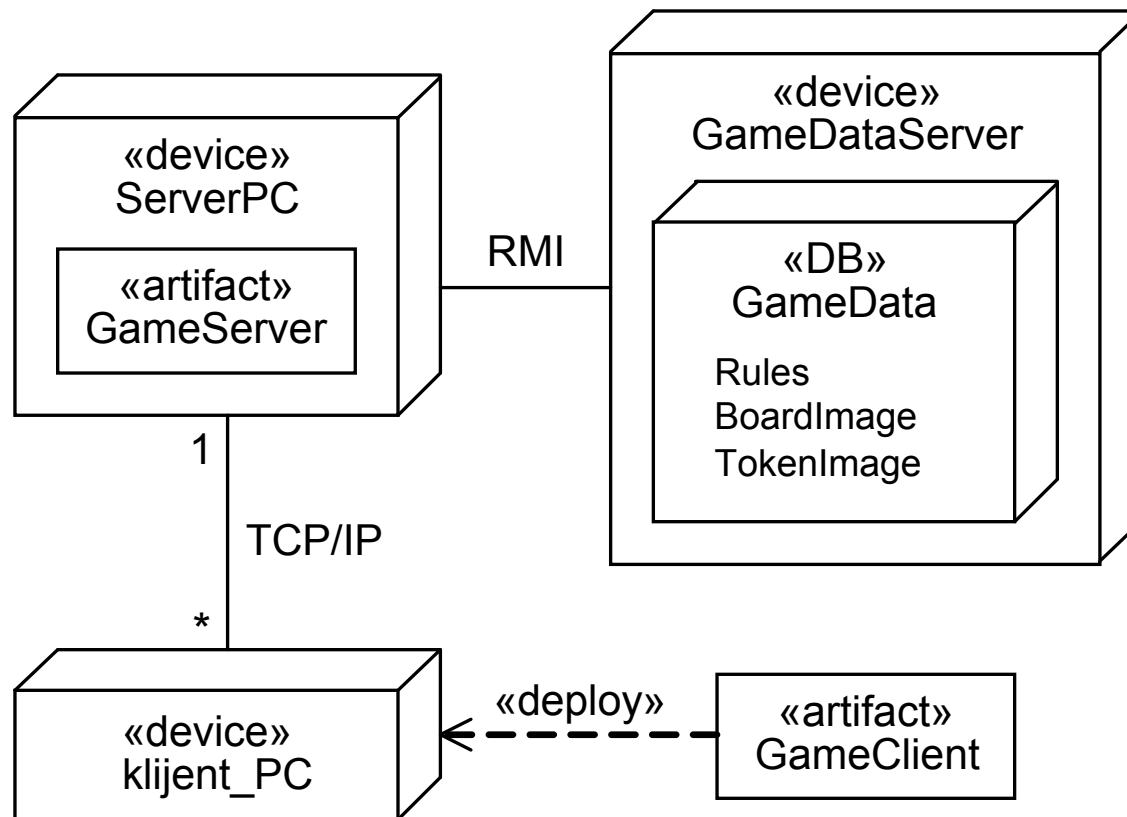
- engl. Deployment diagram
- opisuju fizičku arhitekturu sustava
 - ostvarenje u obliku koda i podataka koje se nalazi i izvršava na računalnim resursima
 - prikaz sklopovskih komponenti, komunikacijskih putova te smještaja i izvođenja programskih artifakta
 - naznaka gdje i kako implementirati komponente
- čvorovi – engl. nodes
 - uređaj – engl. device
 - stvarni i virtualni uređaji
 - procesna jedinka npr. računalo
 - oznaka «device»
 - okolina izvođenja – engl. execution environment
 - programski sustav npr. virtualni stroj, OS, interpreter
 - oznaka «execution environment»
- Spojevi – engl. connections
 - komunikacijski putevi
- Programski artifakti
 - Komponente – implementirani moduli i podaci
 - Ovisnosti - Dependencies
 - prikazuju odnos između komponenti



Primjer:



Primjer:

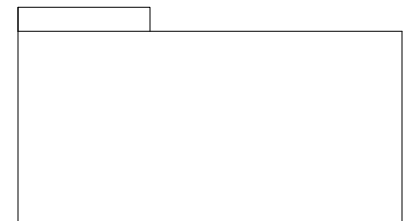


UML dijagrami

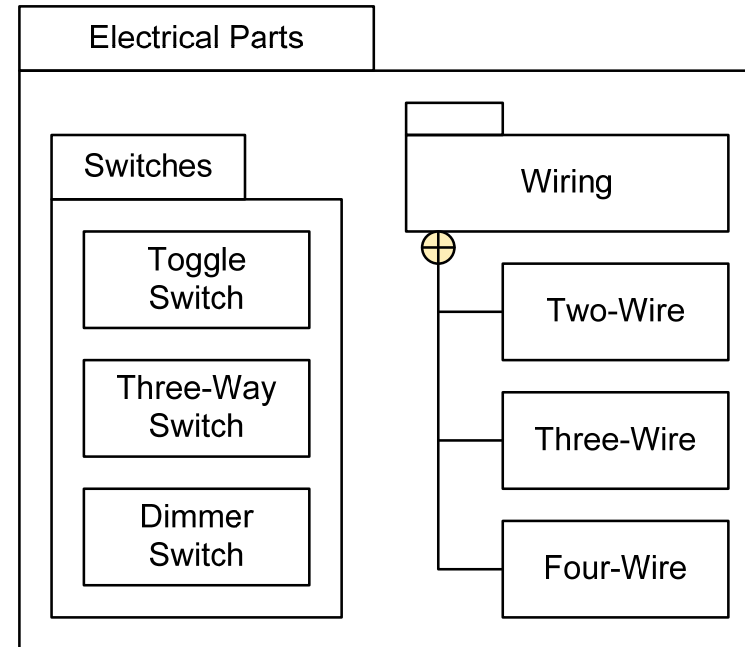
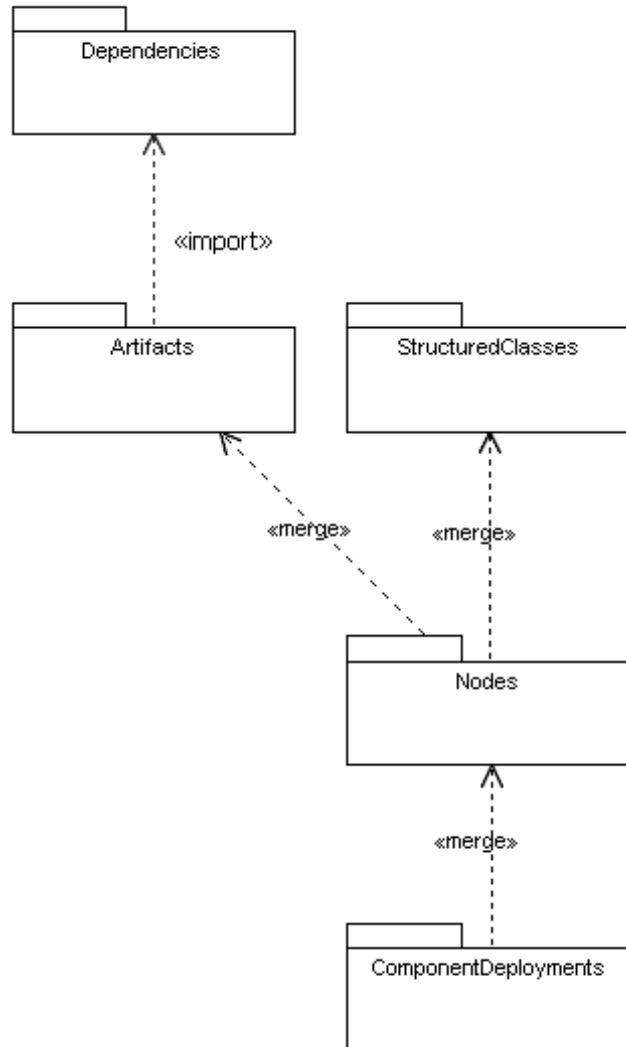
- Obrasci uporabe
- Sekvencijski dijagram
- Komunikacijski dijagram
- Dijagram stanja
- Dijagram aktivnosti
- Dijagram komponentni
- Dijagram razmještaja
- Dijagram paketa
- Dijagram pregleda interakcije
- Vremenski dijagram
- Dijagram profila
- *Dijagram razreda*
- *Dijagram objekata*
- *Dijagram složene strukture*

Dijagram paketa

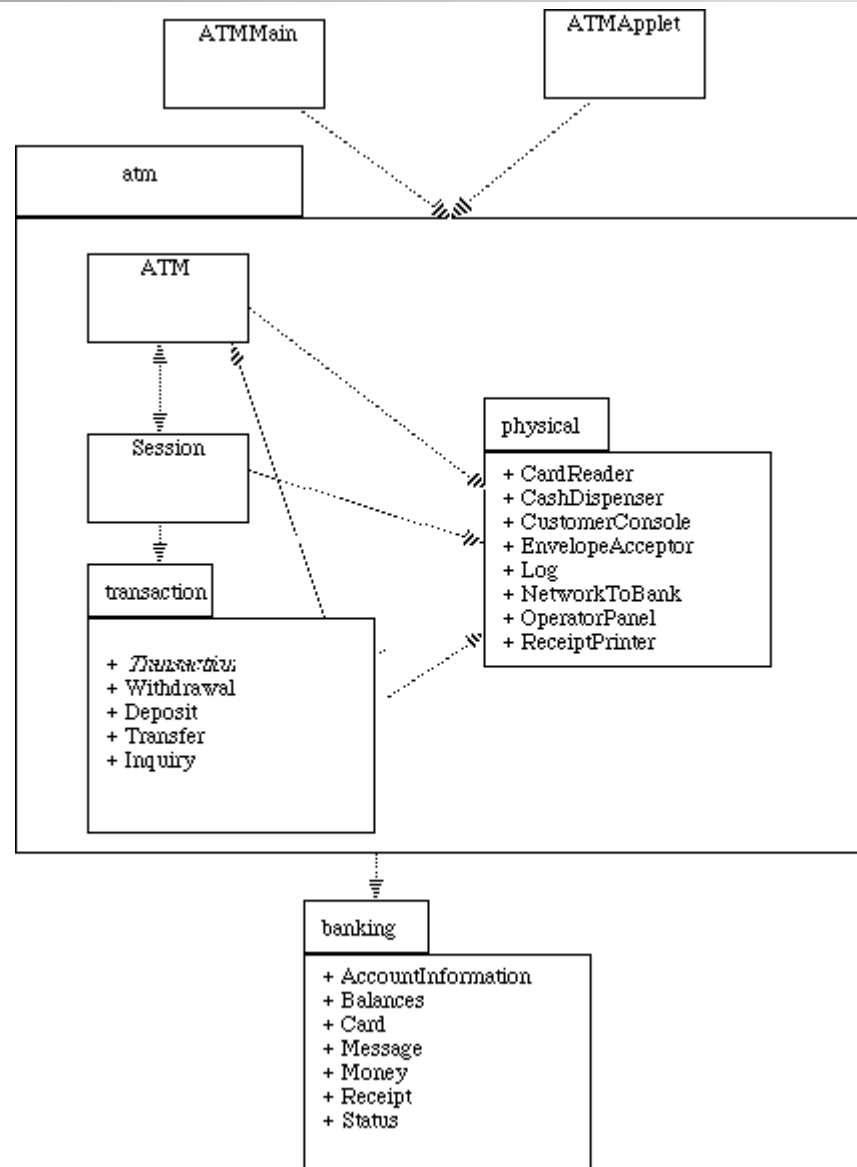
- engl. Package Diagram
- paket je mehanizam organiziranja elemenata u grupe
 - nema utjecaj na izvođenje
- dobro definiran paket povezuje semantički bliske elemente koji imaju tendenciju zajedničkih promjena
- u jednostavnim aplikacijama nema potrebe uvođenja paketa
- dijagram paketa prikazuje dekompoziciju modela u organizirane grupe i njihove međuodnose
 - hijerarhija
- vidljivost
- Moguća su tri nivoa vidljivosti paketa:
 - javni element (engl. public)
 - sadržaj vidljiv svim paketima koji koriste paket (+)
 - zaštićeni (engl. protected)
 - mogu ga vidjeti samo djeca tog paketa (#)
 - privatni (engl. private)
 - nije vidljiv izvan paketa (-)



Primjer: Dijagrami paketa

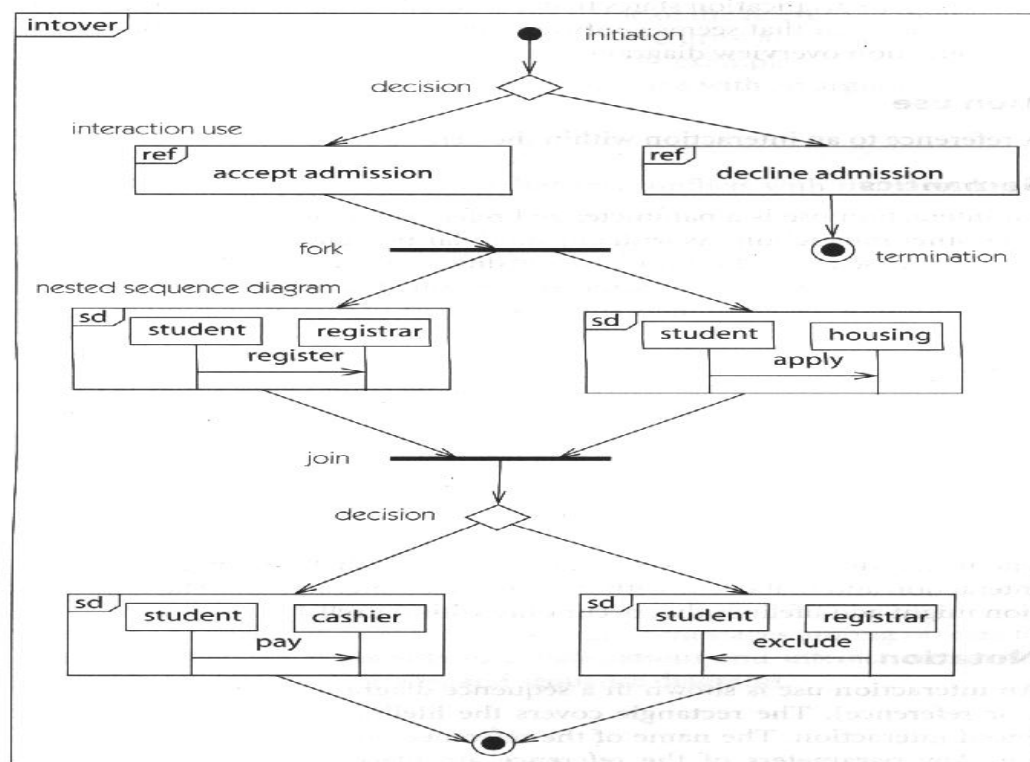


Primjer: bankomat



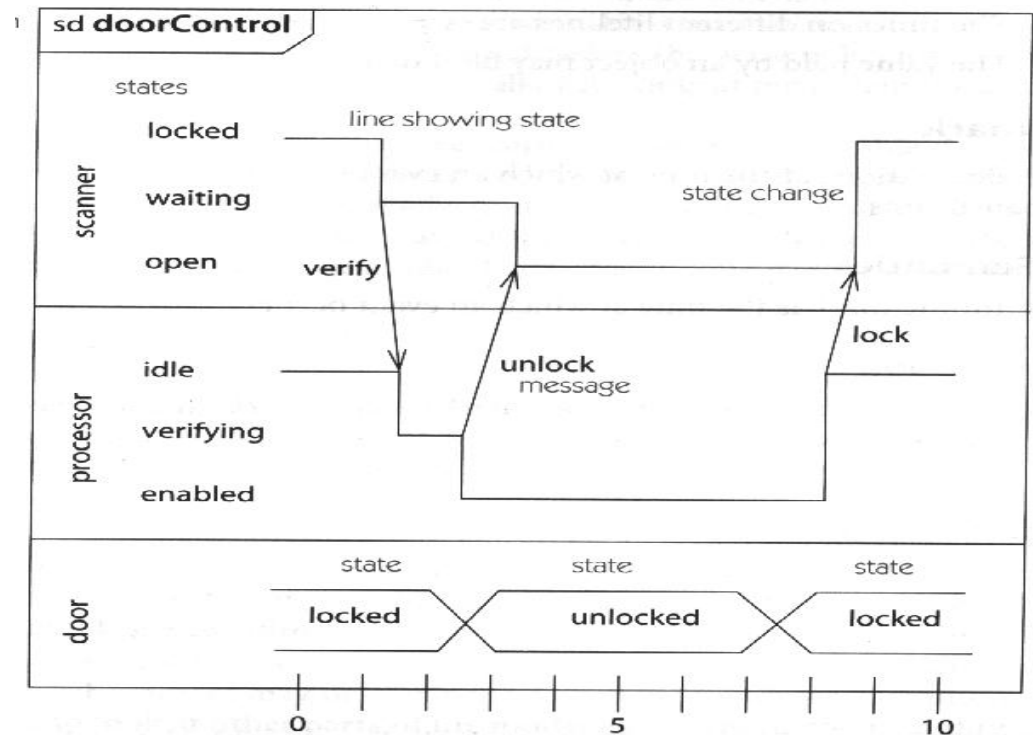
Dijagram pregleda interakcije

- engl. Interaction Overview Diagram
- kombinacija dijagrama aktivnosti i sekvencijskog dijagrama
 - dijagram aktivnosti s dijelovima sekvencijskih dijagrama i kontrolom tijeka
 - notacija odluka i grananja iz dijagrama aktivnosti



Vremenski dijagram

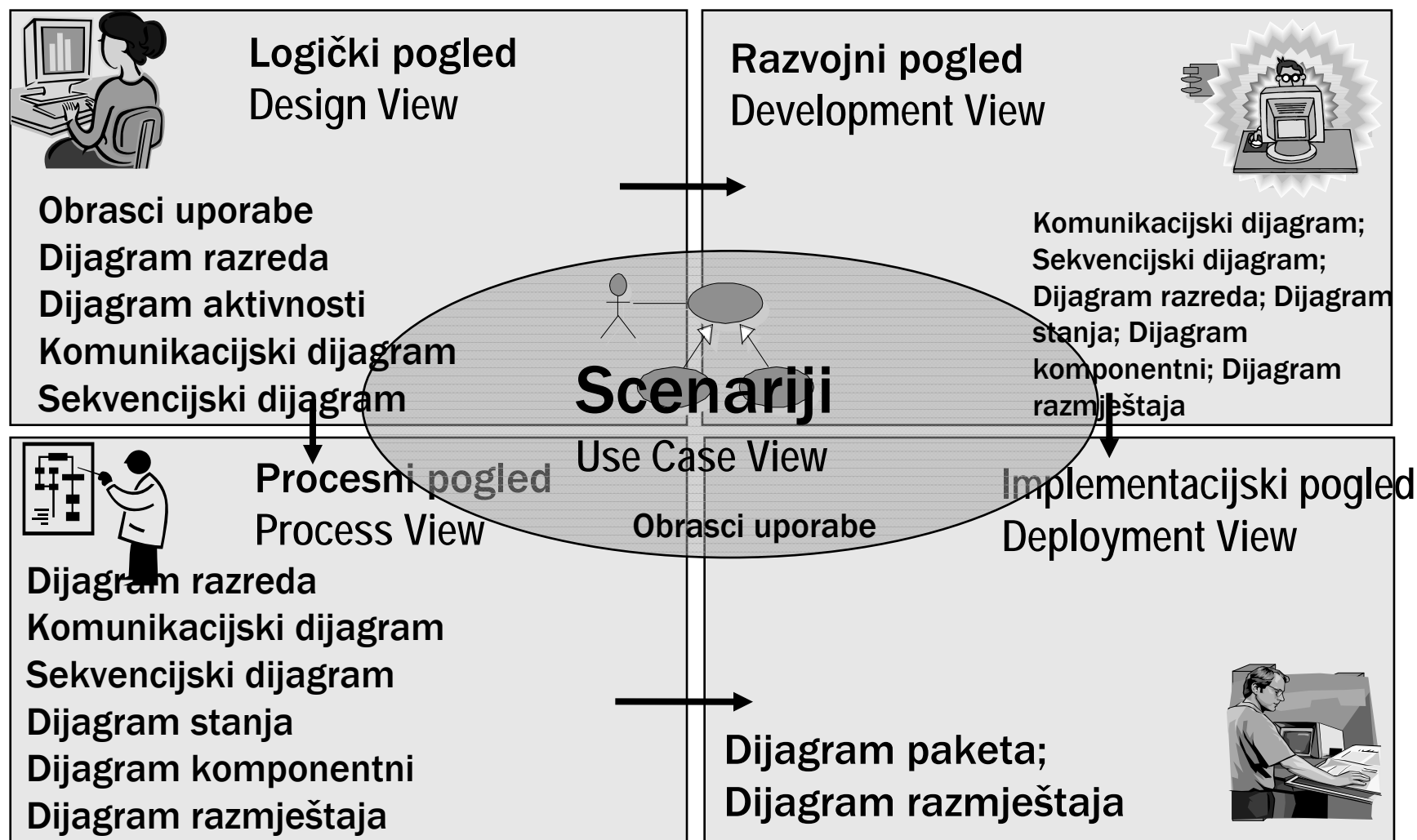
- engl. Timing Diagram
- vrsta dijagrama interakcije
- prilagođen za izričit prikaz stvarnih vremena
 - točniji zapis sekvencijskog dijagrama (vidljiv samo relativan odnos poruka)
 - prikazuje promjene stanja na liniji života u vremenu
- pogodan za primjenu u sustavima za rad u stvarnom vremenu (engl. real-time applications)

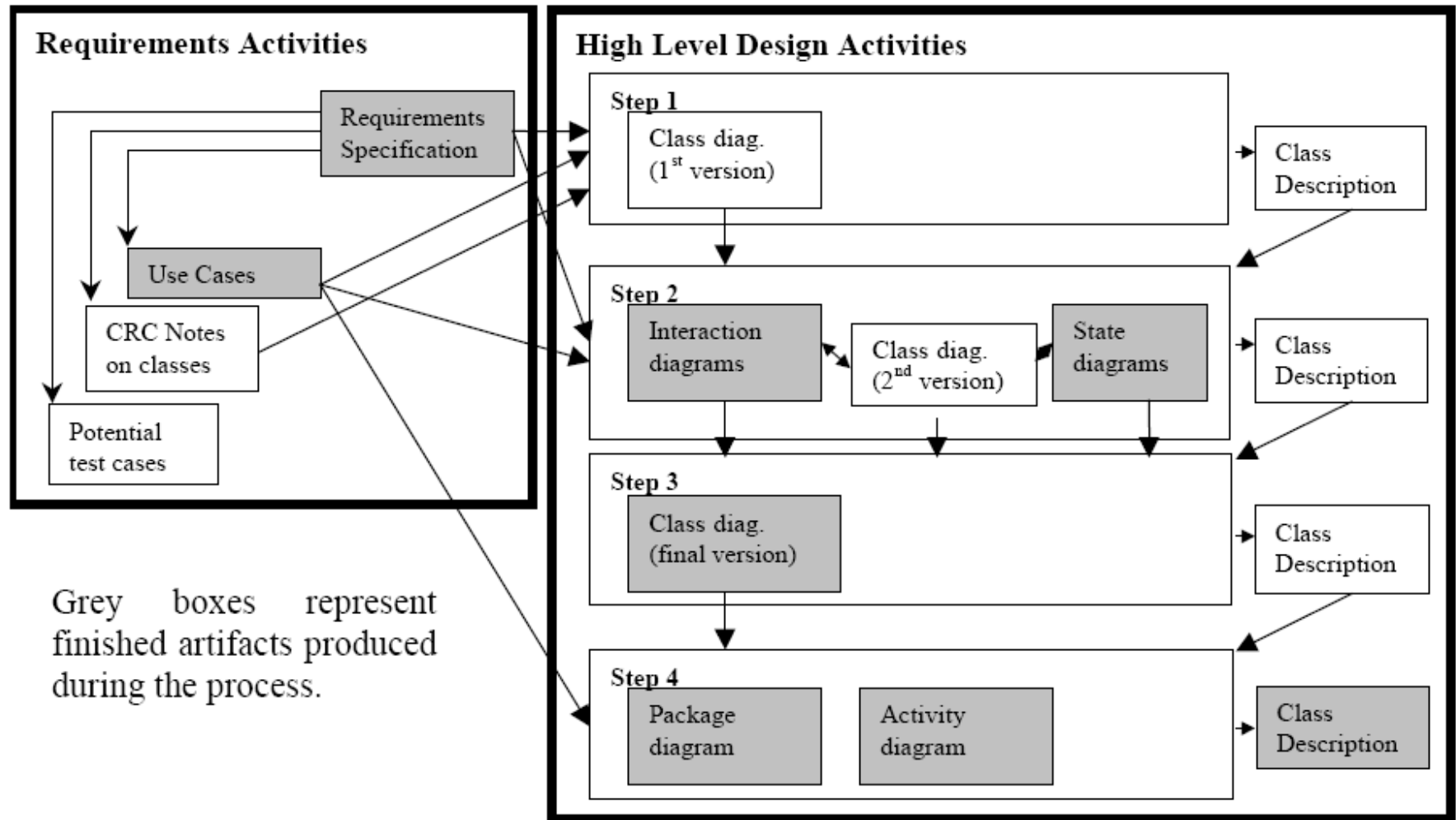


Dijagram profila

- **engl. Profile Diagram**
- **namjena proširenje jezika za stvaranje novih dijagrama**
- **omogućava definiranje korisničkih stereotipa, vrijednosti, ikona za specifična područja primjene, tehnologije ili metode.**

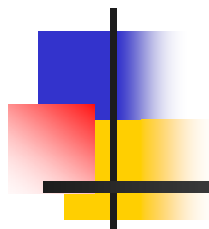
Uporaba UML dijagrama





Diskusija





Oblikovanje programske potpore

MODULARIZACIJA I OBJEKTNO USMJERENA ARHITEKTURA

2. dio



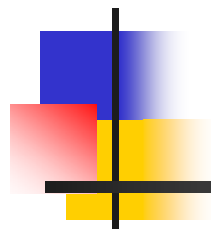
Objektno usmjerena arhitektura – 2. dio

Sadržaj prezentacije:

- Raspodijeljeni (engl. *Distributed*) sustavi
- Arhitektura klijent – poslužitelj
- Posrednička arhitektura
- Uslužno usmjerena arhitektura – SOA (engl. *Service oriented architecture*)
- Arhitektura sustava zasnovanih na komponentama

Izvor:

T.C.Lethbridge, R.Laganieri: Object-Oriented Software Engineering, 2nd ed., McGraw-Hill, 2005.



Objektno usmjerena arhitektura – 2.dio

Raspodijeljeni (engl. *Distributed*) sustavi

Raspodijeljeni sustavi

Značajke **raspodijeljenih sustava**:

- Obrada podataka i izračunavanja obavljaju **odvojeni programi**.
- Uobičajeno je da su ti odvojeni programi na odvojenom sklopovlju (računalima, čvorovima, stanicama).
- Odvojeni programi **međusobno komuniciraju** (kooperiraju) po računalnoj mreži.

Primjer raspodijeljenog sustava:

Klijent - poslužitelj:

Poslužitelj (engl. *server*)

- Program koji dostavlja uslugu drugim programima koji su spojeni na njega preko komunikacijskog kanala.

Klijent:

- Program koji pristupa poslužitelju (ili više njih) tražeći uslugu.
- Poslužitelju mogu pristupiti mnogi klijenti simultano.

Raspodijeljeni sustavi - primjeri

P2P (Peer-to-Peer)

Svaki čvor u sustavu ima jednake mogućnosti i odgovornosti (čvor je istovremeno poslužitelj i klijent).

Snaga obrade podataka i izračunavanja u P2P sustavu ovisi o pojedinim krajnjim čvorovima, a ne o nekom skupnom radu čvorova.

Afinitetna socijalna mreža (engl. *affinity communities*)

Jedan korisnik se povezuje s drugim korisnikom u cilju razmjene informacija ((MP3, video, slike itd.).

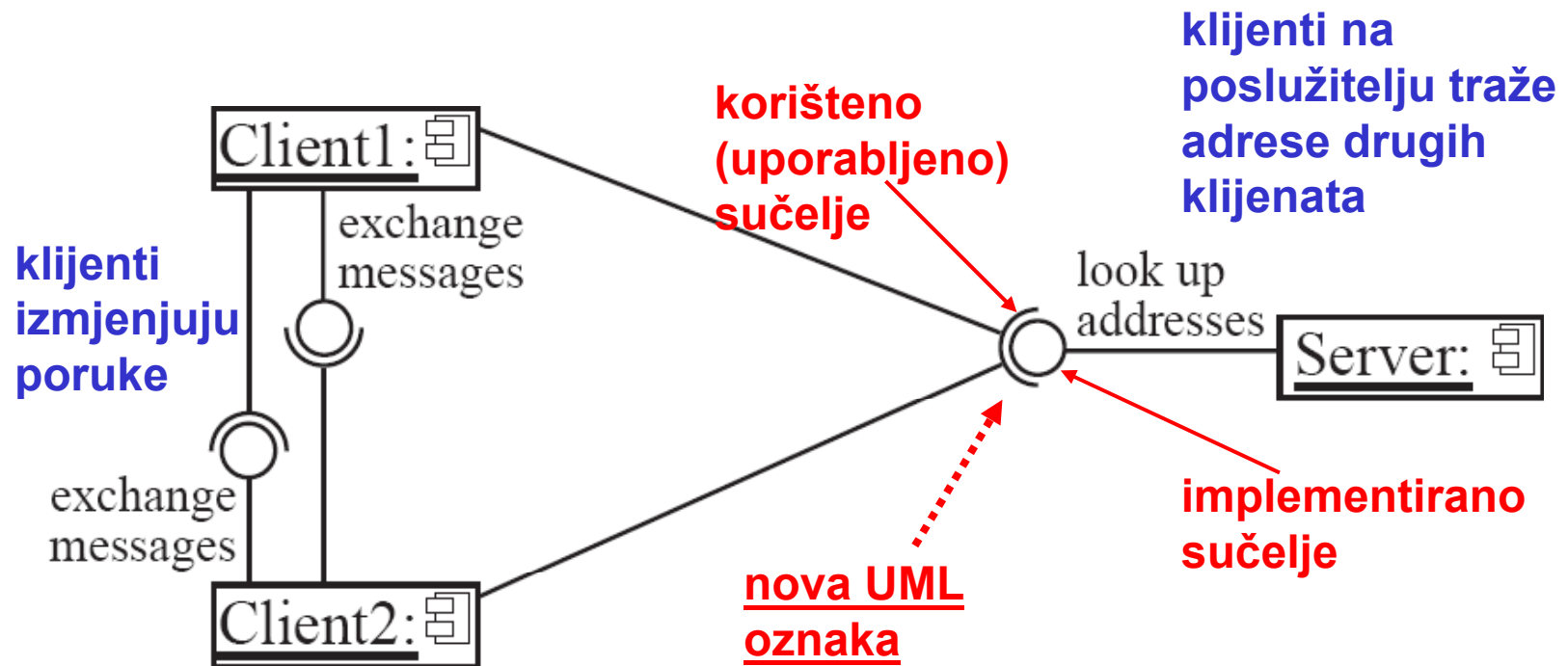
Kolaborativno izračunavanje (engl. *collaborative computing*)

Neiskorišteni resursi (CPU vrijeme, prostor na disku) mnogih računala u mreži kombiniraju se u izvođenju zajedničkog zadatka (GRID computing, SETI@home, ...).

Microsoft Instant Messaging

Izmjena tekstovnih poruka između korisnika u stvarnom vremenu.

Raspodijeljeni sustavi - primjer



Primjer raspodijelnog sustava u kojem klijenti preko poslužitelja doznaju za svoje interese te zatim komuniciraju izravno.

Problemi u oblikovanju raspodijeljenih sustava:

Kako se alociraju i pokreću funkcije poslužitelja ?

Kako se definiraju i šalju parametri između klijenta i poslužitelja ?

Kako se rukuje neuspjesima (pogreškama) u komunikaciji ?

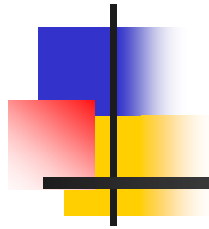
Kako se postavlja i rukuje sa sigurnošću ?

Kako klijent pronalazi poslužitelja ?

Koje strukture podataka koristiti i kako rukovati s njima ?

Koja su ograničenja u paralelnom radu dijelova raspodijeljenog sustava ?

Kako se uopće skupina komponenata usaglašava oko zajedničkih pitanja ?



Objektno usmjerena arhitektura – 2.dio

Arhitektura klijent – poslužitelj

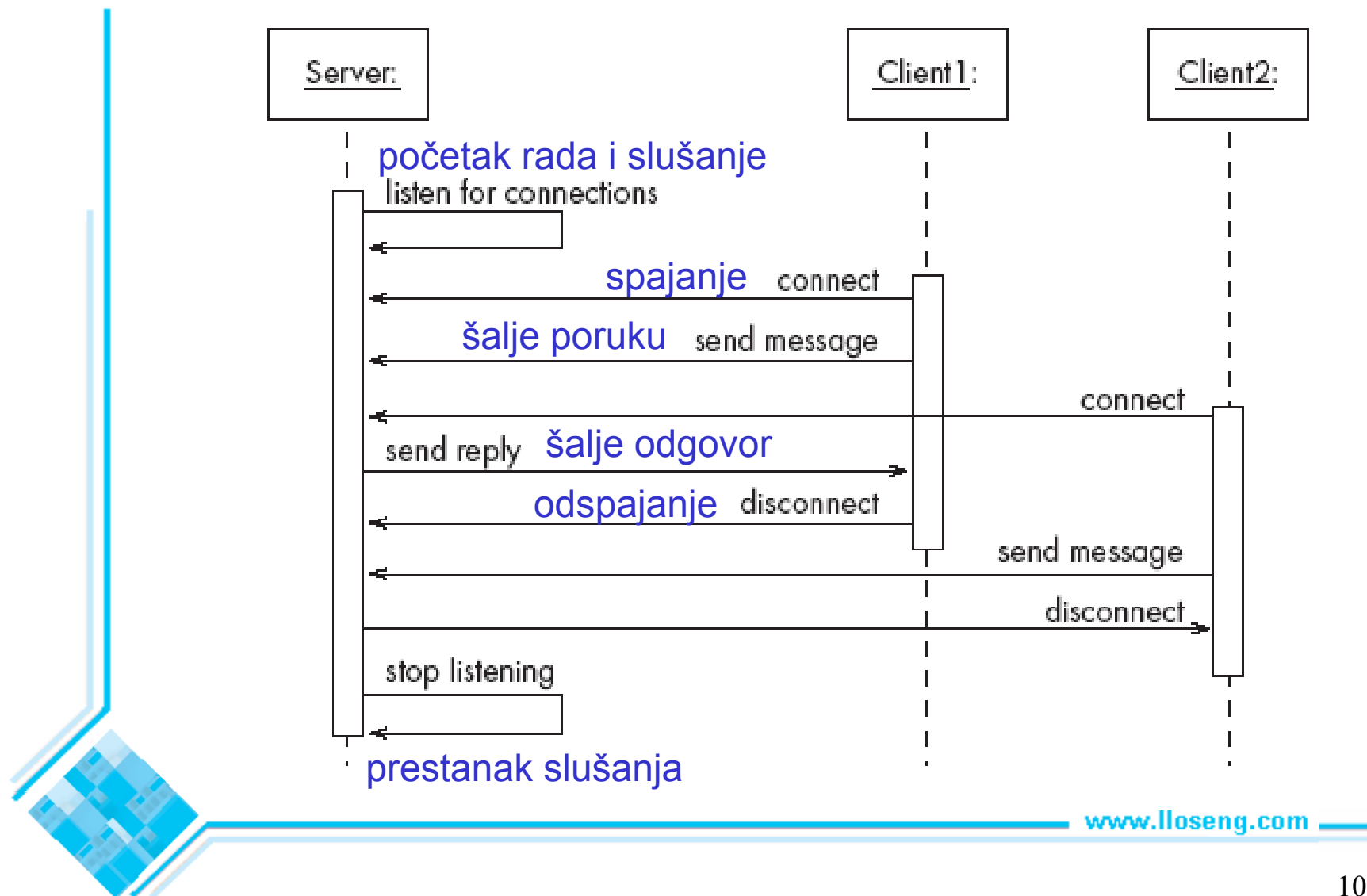
Ciljevi daljnje prezentacije

- Detaljnije razmatranje aktivnosti klijenata i poslužitelja.
- Metoda oblikovanja arhitekture klijent-poslužitelj temeljena na ponovnoj i višestrukoj uporabi komponenata (*engl. reuse*).

Klijent – poslužitelj: sekvenca aktivnosti

1. Poslužitelj započinje s radom.
2. Poslužitelj čeka na dolazak klijentskog zahtjeva (*poslužitelj sluša*).
3. Klijenti započinju s radom i obavljaju razne operacije,
 - Neke operacije traže zahtjeve (i odgovore) s poslužitelja.
4. Kada klijent pokuša spajanje na poslužitelja, poslužitelj mu to omogućiti (ako poslužitelj želi).
5. Poslužitelj čeka na poruke koje dolaze od spojenih klijenata.
6. Kada pristigne poruka nekog klijenta poslužitelj poduzima akcije kao odziv na tu poruku.
7. Klijenti i poslužitelj nastavljaju s navedenim aktivnostima sve do odspajanja ili prestanka rada.

Poslužitelj u komunikaciji s dva klijenta



Alternative klijent – poslužitelj arhitekture

- Postoji jedan program na jednom računalu koji obavlja sve.
- Računala nisu spojena u mrežu već svako računalo obavlja svoj posao odvojeno.
- Ostvariti neki drugi mehanizam (osim klijent-poslužitelj) kako bi računala u mreži razmjenjivala informacije.

Npr. jedan program upisuje u bazu podataka a drugi čita.

Prednosti klijent – poslužitelj arhitekture

- Posao se može **raspodijeliti** na više računala (strojeva).
- Klijenti **udaljeno** pristupaju funkcionalnostima poslužitelja.
- Klijent i poslužitelj mogu se **oblikovati odvojeno**.
- Oba entiteta mogu biti **jednostavnija**.
- Svi podaci mogu se držati **na jednom mjestu** (na poslužitelju).
- Obrnuto, podaci se mogu **distribuirati** na više udaljenih klijenata i poslužitelja.
- Poslužitelju mogu **simultano** pristupiti više klijenata.
- Klijenti mogu ući u **natjecanje** (kompeticiju) za uslugu poslužitelja (a i obrnuto).

Primjeri klijent – poslužitelj sustava

- The World Wide Web
- E-mail
- Network File System
- Transaction Processing System
- Remote Display System
- Communication System
- Database System

Funkcionalnosti poslužitelja (UML dijagram stanja)

Poslužitelj se inicijalizira.

Započinje slušati klijentska spajanja.

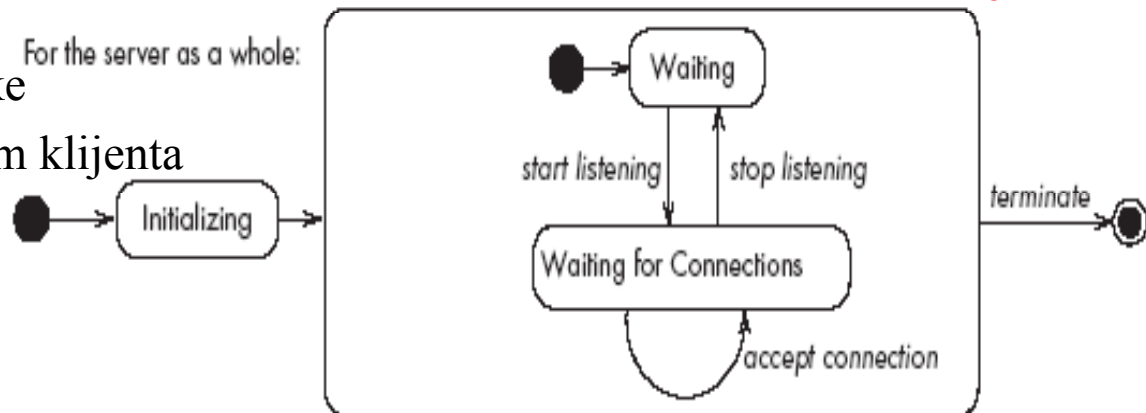
Rukuje slijedećim tipovima događaja koje potiču klijenti:

1. Prihvaća spajanje
2. Odgovara na poruke
3. Rukuje odspajanjem klijenta

Može prestati slušati.

Mora čisto završiti rad.

For the server as a whole:



poslužitelj
kao cjelina

For each connection:



za svako spajanje
rukuje spajanjem,
reagira na poruku

Funkcionalnosti klijenta (UML dijagram aktivnosti)

Klijent se inicijalizira.

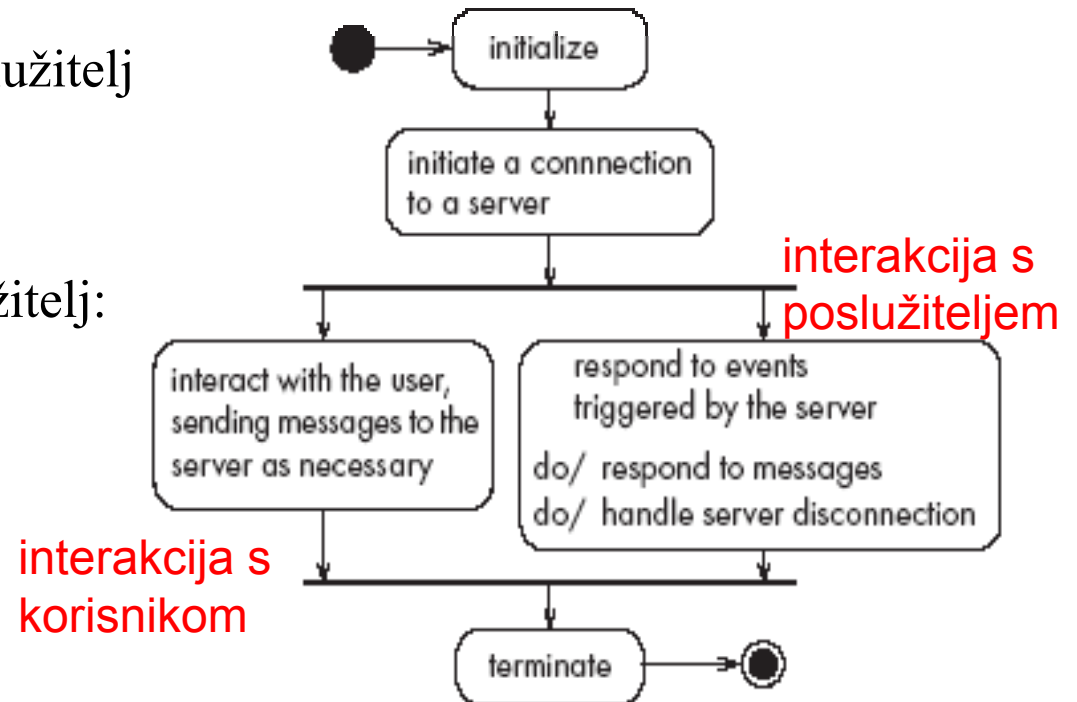
Inicijalizira spajanje na poslužitelj

Šalje poruke.

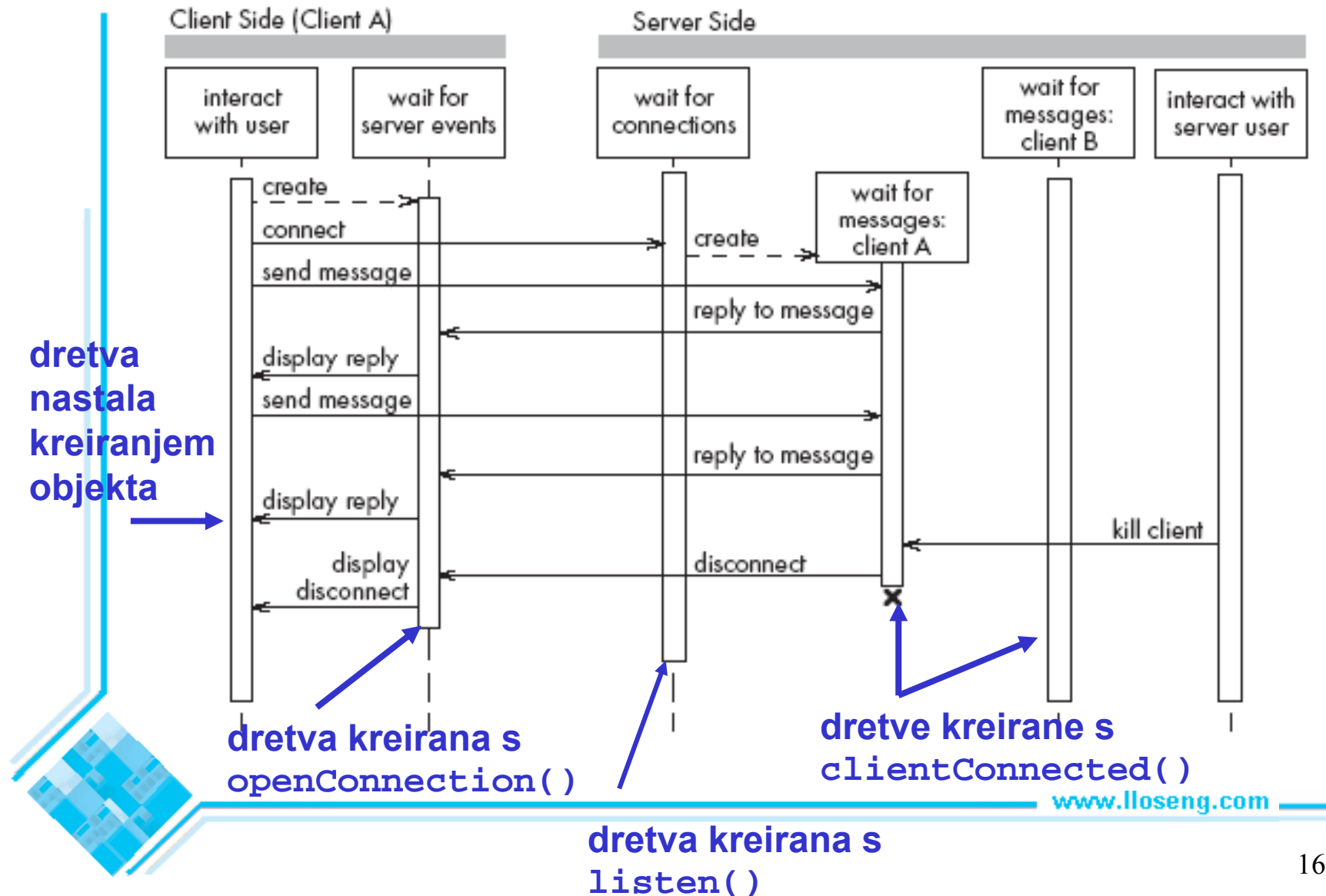
Rukuje slijedećim tipovima događaja koje potiče poslužitelj:

1. Odgovara na poruke.
2. Rukuje odspajanjem od poslužitelja.

Mora čisto završiti rad.



Paralelne dretve/niti izvođenja u klijent - poslužitelj sustavu



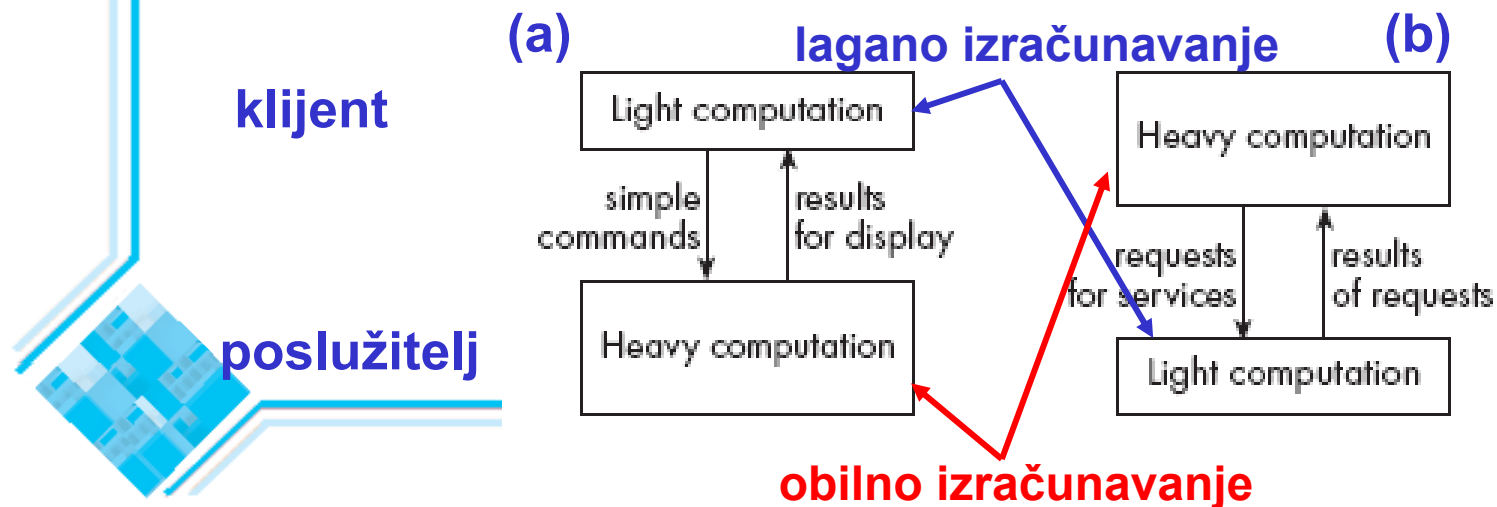
Tanki i debeli klijent

Sustav *tankog klijenta* (engl. *Thin-client*) (a)

- Klijent je oblikovan da bude što je moguće više mali i jednostavniji.
- Većina posla obavlja se na poslužiteljskoj strani..
- Oblikovnu strukturu klijenta i izvršni kod jednostavno se preuzima preko računalne mreže.

Sustav *debelog klijenta* (engl. *Fat-client*) (b)

- Što je moguće više posla delegira se klijentima.
- Poslužitelj može rukovati s više klijenata.



Komunikacijski protokoli

- Poruke koje klijenti šalju poslužitelju formiraju jedan jezik.
 - Poslužitelj mora biti programiran da razumije taj jezik.
- Poruke koje poslužitelj šalje klijentima također formiraju jedan jezik.
 - Klijenti moraju biti programirani da razumiju taj jezik.
- Kada klijent i poslužitelj komuniciraju oni razmjenjuju poruke na ta dva jezika.
- Ta dva jezika i pravila konverzacije čine zajedničkim imenom **protokol**.

Internet protokoli

Internet Protocol (IP)

- Određuje put poruka od jednog računala do drugog.
- Dugačke poruke se cijepaju u manje dijelove.

Transmission Control Protocol (TCP)

- Rukuje spajanjem između dva računala.
- Računala mogu nakon toga razmjenjivati mnoge IP poruke preko ostvarenog spajanja.
- Osigurava da je poruka uspješno primljena.

Svako računalo (čvor) u mreži ima jedinstvenu IP adresu i ime (host name).

- Nekoliko poslužitelja mogu raditi na jednom čvornom računalu u mreži. .
- Svaki poslužitelj na računalu je identificiran preko jedinstvenog broja ***ulaznog porta*** (engl. *port number*) u rasponu 0 to 65535.
- Kako bi započeo komunikaciju s poslužiteljem klijent mora znati **host name** i **port number**
- Brojevi ulaznih portova 0 – 1023 su rezervirani (npr. port 80 za Web poslužitelj).

Oblikovanje klijent-poslužitelj arhitekture

Oblikuj temeljne poslove poslužitelja i klijenta.

Odredi kako će se posao raspodijeliti (tanki nasuprot debelog klijenta).

Oblikuj detalje skupa poruka koje se razmjenjuju (komunikacijski protokol).

Oblikuj mehanizme (što se mora dogoditi) prilikom:

- Inicijalizacije
- Rukovanja spajanjima.
- Slanja i primanja poruka.
- Završetka rada.

U oblikovanju koristi princip “Uporabi postojeće gotove komponente” (princip oblikovanja br. 6)

Oblikovanje temeljem iskustva drugih

Inženjeri koji oblikuju programsku potporu trebaju izbjegavati razvoj programske potpore koja je već bila razvijena.

Tipovi ponovnog korištenja (engl. *reuse*):

- Ponovno koristi ekspertizu (posebna znanja).
- Ponovno koristi standardna oblikovanja i algoritme.
- Ponovno koristi knjižnice razreda ili procedura.
- Ponovno koristi snažne naredbe ugrađene u programski jezik ili operacijski sustav.
- Ponovno koristi radne okvire (engl. **frameworks**).
- Ponovno koristi cijela primjenska rješenja (aplikacije).

Radni okviri: Podsustavi za ponovno korištenje

Radni okvir (engl. *Framework*) je učestalo korišten dio programske potpore koji implementira generičko (opće ili zajedničko) rješenje generičkog problema.

- Osigurava opća (zajednička) sredstva koja se mogu uporabiti u različitim primjenskim programima (aplikacijama).

Princip:

Primjenski programi koji rade različite, ali na neki način povezane stvari imaju slične oblikovne obrasce.

Objektno usmjereni radni okviri

U objektno usmjereojoj paradigmi radni okvir se sastoji iz knjižnice razreda.

Radni okvir sadrži:

- Primjensko sučelje (engl *Application programming interface* – API) definirano je kao skup svih javnih (engl. *public*) metoda tih razreda.
- Neki od razreda u radnom okviru biti će apstraktni (operacije se moraju dodatno implementirati).

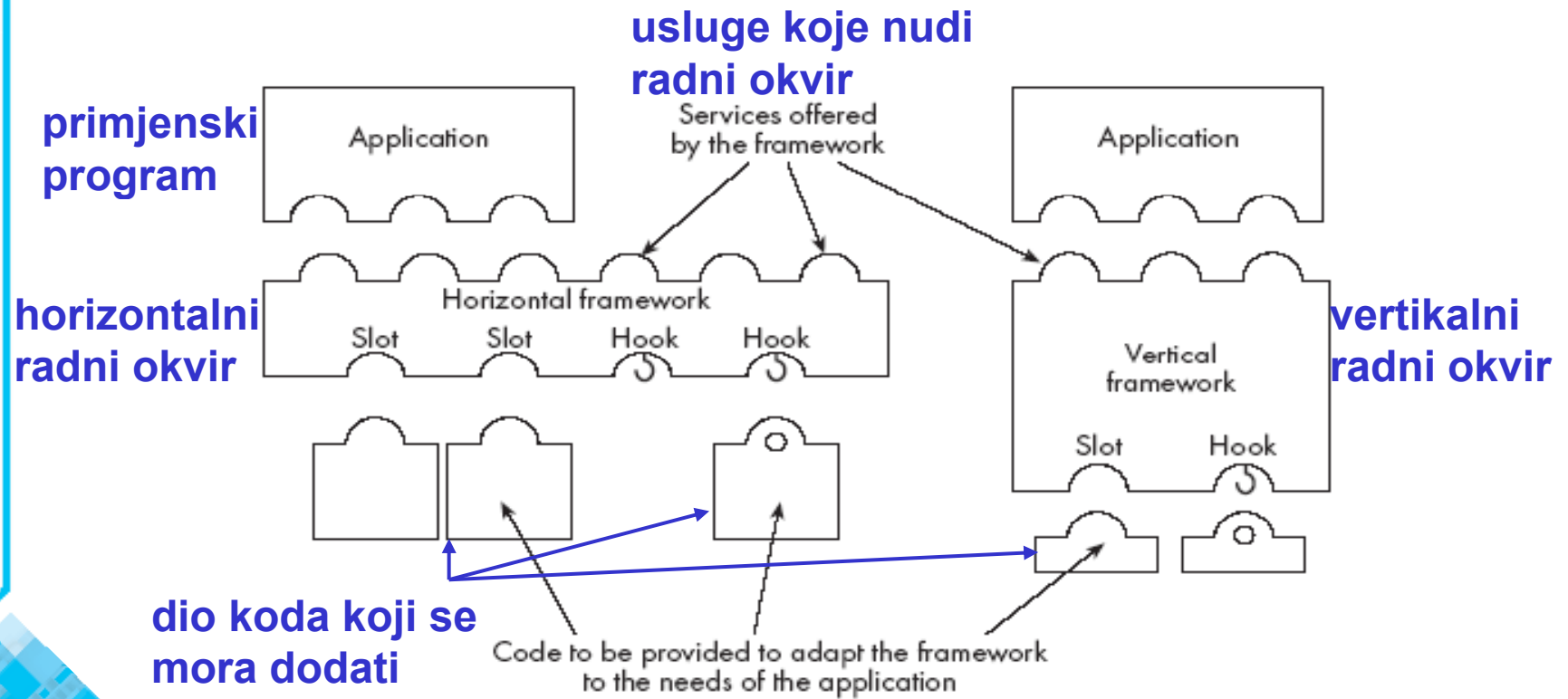
Radni okviri i linije proizvoda (engl. *product lines*)

- Linije proizvoda (ili porodica proizvoda) je skup svih produkata izrađenih na zajedničkoj osnovnoj tehnologiji.
- Različiti produkti u liniji proizvoda imaju različite značajke kako bi zadovoljili različite segmente tržišta.
 - Npr. “demo”, “pro”, “lite”, “enterprise” i sl. verzije.
 - Lokalizirane verzije su također linije proizvoda.
- Programska tehnologija zajednička svim proizvodima u liniji proizvoda uključena je u **radni okvir** (engl. *framework*).
- Svaki proizvod izrađen je temeljem radnog okvira u kojem su popunjena odgovarajuća prazna mjesta.

Vrste radnih okvira

Horizontalni radni okvir osigurava veći broj općih i zajedničkih sredstava koja mogu koristiti mnogi primjenski programi (aplikacije).

Vertikalni radni okvir (primjenski) je mnogo cjelovitiji ali još uvijek traži popunu nekih nedefiniranih mjesta kako bi se prilagodio specifičnoj primjeni.



Primjeri radnih okvira

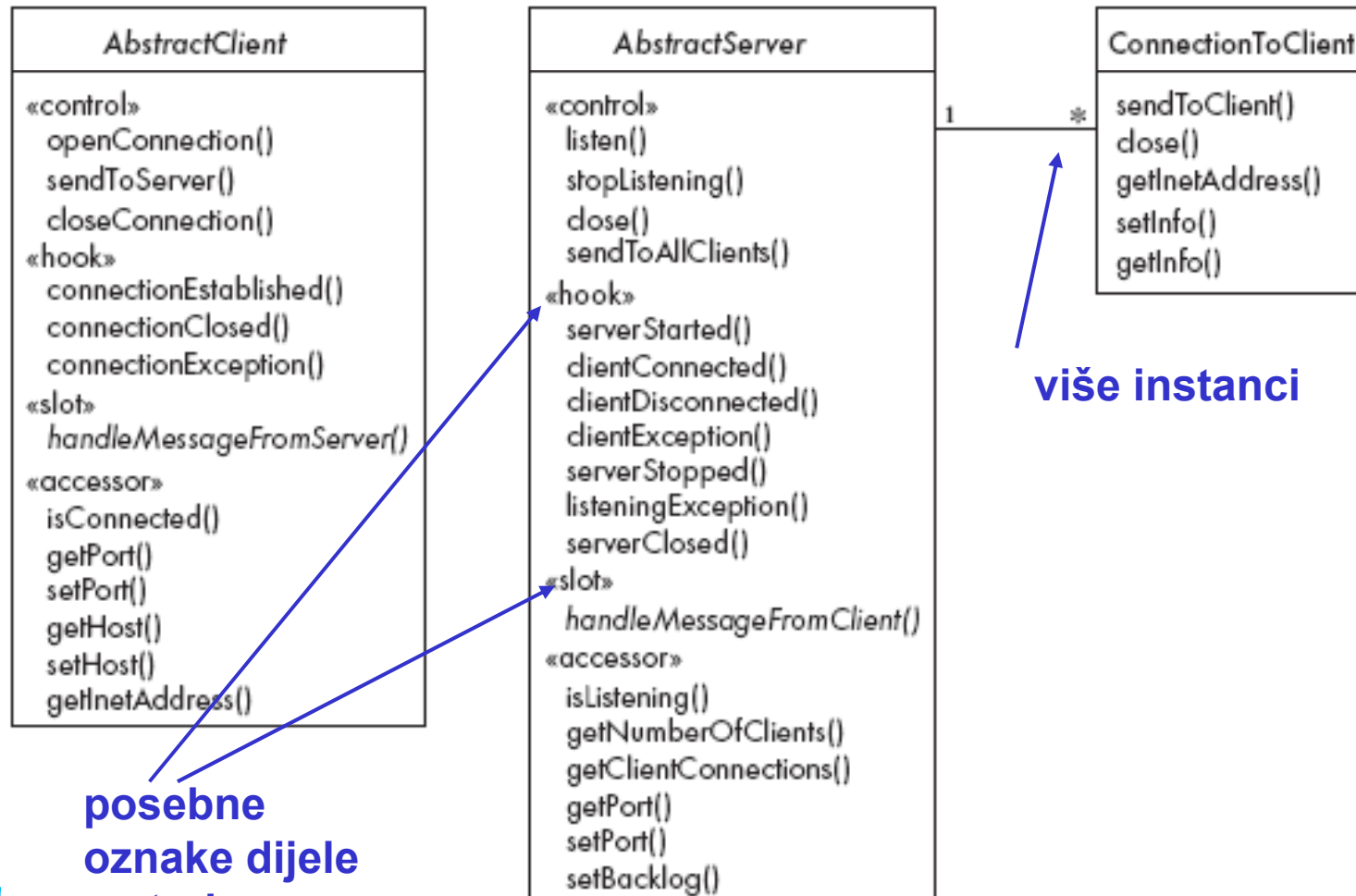
- Radni okvir za rukovanje plaćama.
- Radni okvir za rukovanje čestih putnika (engl. *frequent flyer*) u avioprometu.
- Radni okvir za rukovanje čestih kupaca.
- Radni okvir za upis i registraciju predmeta na fakultetu.
- Radni okvir za komercijalno web sjedište (e-dućan).
- Radni okvir za mrežnu uslugu XYZ .

Uporaba objektnog klijent-poslužitelj radnog okvira (engl. Object Client-Server Framework - OCSF):

Postupak uporabe:

- Ne mijenjati apstraktne razrede u OCSF.
- Kreirati podrazrede.
- Konkretizirati metode u podrazredima.
- Nanovo definirati (*engl. override*) neke metode u podrazredima.
- Napisati kod koji kreira instance i inicira akcije.

OCSF – tri razreda s istaknutim metodama



posebne
oznake dijele
metode u
kategorije

više instanci

Uporaba OCSF

Programski inženjeri u uporabi OCSF **nikada ne modificiraju** tri navedena razreda.

Umjesto modifikacije izvornih razreda treba:

- **Kreirati podrazrede** apstraktnih razreda u radnom okviru i implementirati metode (učiniti ih konkretnima).
- **Zvati javne metode** koje uključuje radni okvir. Te metode su usluge koje pruža radni okvir.
- **Redefinirati** neke metode posebno označene u kategorije “hook” i “slot” (vidi raniju sliku apstraktnih razreda) koje su eksplicitno namijenjene da budu redefinirane.

Klijentska strana

Sadrži jedan apstraktan razred: **AbstractClient**

- Iz toga razreda *moraju* se izvesti podrazredi.
 - Svaki podrazred mora osigurati implementaciju:
handleMessageFromServer
navedenu u skupu **<<slot>>** oznake. Metoda je zadužena za odgovarajuću operaciju po primitku poruke od poslužitelja.
- **AbstractClient** implementira sučelje prema poslužitelju kao posebnu nit izvođenja (dretvu) svoje instance (objekta) – vidi raniji sekvencijski dijagram. Dretva započinje izvođenje nakon što metoda **openConnection** pozove **start** koja pozove **run** metodu.
- **run** metoda sadrži petlju koja se izvodi tijekom životnog ciklusa dretve (prima poruke i odgovara na njih).

Javno (engl. *public*) sučelje **AbstractClient**

Constructor **AbstractClient**:

Inicijalizira *host* i *port* varijable na koje će se klijent spojiti.

Upravljačke (<<control>>) metode:

- **openConnection** (započinje posebnu dretvu, koristi *host* i *port* varijable koje postavlja constructor ili može koristiti metode **setHost**, **setPort**)
- **closeConnection** (dretva zaustavlja rad u petlji i završava)
- **sendToServer** (poruka koja može biti bilo koji objekt)

Pristupne (<<accessor>>) metode:

- **isConnected** (ispituje da li je klijent spojen)
- **getHost** (ispituje koji *host* je spojen)
- **setHost** (omogućuje promjenu *hosta* dok je klijent odspojen)
- **getPort** (ispituje na koji *port* je klijent spojen)
- **setPort** (omogućuje promjenu *porta* dok je klijent odspojen).
- **getInetAddress** (dobavlja neke detaljnije informacije)

Callback metode u **AbstractClient**

Metode koje se **mogu** redefinirati (ako podrazred ima namjeru poduzeti neke akcije kao odziv na događaj):

- **connectionEstablished** (poziva se po uspostavi veze s poslužiteljem)
- **connectionClosed** (poziva se nakon završetka veze)
- **connectionException** (poziva se kada nešto pođe krivo, npr. poslužitelj prekida vezu.)

Metoda koja se **mora** implementirati (najvažniji dio koda):

- **handleMessageFromServer** (definira se u podrazredima i pozove kada je primljena poruka od poslužitelja.

Callback funkcije su one koje se ne zovu izravno. (npr. u C++ zovu se preko pokazivača).

Najčešće se **callback** funkcije zovu kao odziv na neki asinkroni događaj (sva moderna grafička korisnička sučelja zasnovana su na **callback** funkcijama, npr. odziv na miša).

www.itseng.com

Uporaba razreda **AbstractClient**

- Kreiraj podrazred od **AbstractClient**
- U podrazredu implementiraj **handleMessageFromServer** `<<slot>>` metodu
- Napiši kod koji:
 - Kreira instancu novoga podrazreda
 - Pozove **openConnection** (start druge dretve)
 - Šalje poruku poslužitelju uporabom **sendToServer** metode
- Implementira **connectionClosed** callback (npr. izvješćuje korisnika o čistom završetku veze).
- Implementira **connectionException** callback (npr. kao odziv na kvar na mreži).

Privatni dijelovi razreda **AbstractClient**

Osobe zadužene za oblikovanje programske potpore ne moraju znati te detalje, ali može pomoći.

Variable instanci:

- **clientSocket** (tipa **Socket**) koja drži sve informacije o vezi s poslužiteljem.
- Dva niza tipa **ObjectOutputStream** i **ObjectInputStream** koji se koriste za slanje i prijam objekata uporabom varijable **clientSocket**.
- **clientReader** tipa **Thread** koja se izvodi **run** metodom objekta razreda **AbstractClient**. Dretva započinje kada **openConnection** pozove **start** koja pozove **run**. Petlja unutar **run** čeka na poruku koja dolazi od poslužitelja. Kada je poruka primljena, **run** zove metodu **handleMessageFromServer**.
- Dvije varijable koje čuvaju **host** i **port** poslužitelja.

Poslužiteljska strana

Poslužiteljska strana sadrži dva razreda pa je nešto složeniji rad. Potrebne su dvije niti izvođenja (dretve): jedna koja sluša novo spajanje klijenta, a druga (jedna ili više) koja rukuje vezama s klijentima (vidi raniji sekvencijski dijagram).

Instanca razreda **AbstractServer** – sluša novo spajanje klijenta.

Jedna ili više instanci razreda **ConnectionToClient** – rukuje vezana s klijentima.

Javno (engl. *public*) sučelje **AbstractServer**

Constructor **AbstractServer**:

s brojem porta na kojem poslužitelj sluša. Kasnije se može promijeniti sa **setPort**.

Upravljačke (<<control>>) metode:

- **listen** (kreira varijablu **serverSocket** tipa **Socket** koja će slušati na portu specificiranom konstruktorom, inicira dretvu, a **run** metoda čeka na spajanje klijenta) – vidi raniji sekvencijski dijagram.
- **stopListening** (signalizira **run** metodi da prestane slušati. Spojeni klijenti i dalje komuniciraju).
- **close** (kao **stopListening** ali odspaja sv eklijente)
- **sendToAllClients** (šalje poruku svim spojenim klijentima)

Pristupne (<<accessor>>) metode:

- **isListening** (vraća da li poslužitelj sluša)
- **getClientConnections** (može se iskoristiti za neki rad sa svim klijentima)
- **getPort** (vraća na kojem portu poslužitelj sluša)
- **setPort** (koristi se za slijedeći **listen()**, nakon zaustavljanja)
- **setBacklog** (postavlja veličinu repa čekanja)

Callback metode u **AbstractServer**

Metode koje mogu biti redefinirane (ukoliko su konkretni podrazredi zainteresirani za poseban odziv na događaj).

- **serverStarted** (poziva se kada poslužitelj započinje prihvaćati spajanja).
- **clientConnected** (poziva se kada se novi klijent spoji, sadrži instancu razreda **ConnectionToClient** kao argument) – vidi OCSF.
- **clientDisconnected** (poziva se kada poslužitelj odspaji klijenta, argument je instanca razreda **ConnectionToClient**).
- **clientException** (poziva se kada se klijent sam odspoji ili kada nastupi kvar u mreži)
- **serverStopped** (poziva se kada poslužitelj prestane prihvaćati povezivanje s klijentima a kao rezultat **stopListening**).
- **listeningException** (poziva se kada poslužitelj prestane slušati zbog nekog kvara).
- **serverClosed** (poziva se nakon završetka rada poslužitelja).

Metoda koja se mora implementirati (najvažniji dio koda):

- **handleMessageFromClient** (argumenti su primljena poruka te instanca razreda **ConnectionToClient**).

Javno sučelje **ConnectionToClient**

Za svakog klijenta postoji instanca razreda **ConnectionToClient** za vrijeme dok je klijent spojen na poslužitelja.

ConnectionToClient je konkretan razred. Korisnici ne moraju kreirati podrazrede:

Upravljačke (<<control>>) metode:

- **sendToClient** (središnja metoda, koristi se za komunikaciju s klijentom).
- **close** (odspaja klijenta)

Pristupne (<<accessor>>) metode:

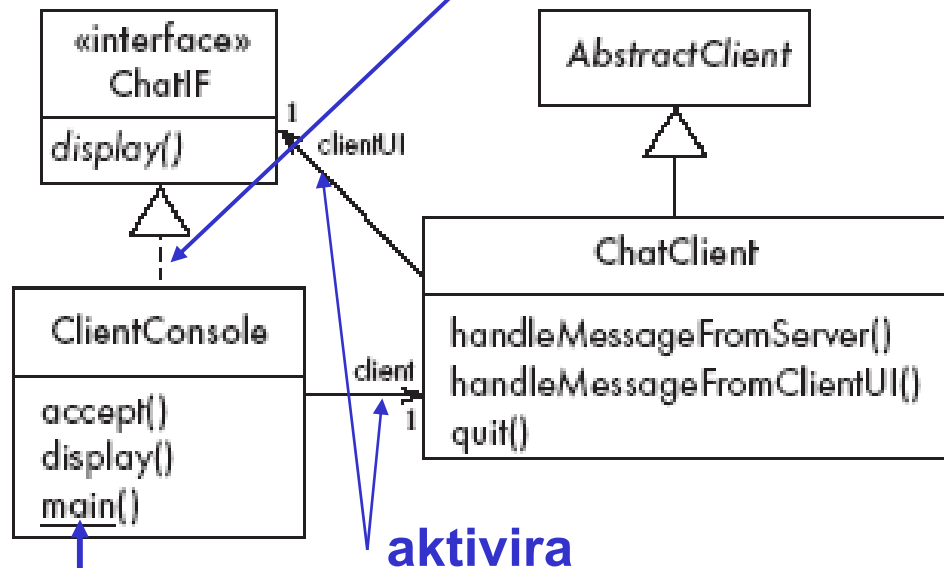
- **getInetAddress** (dobavlja Internet adresu klijenta)
- **setInfo** (omogućuje spremanje proizvoljnih informacija o klijentu. Npr. posebne privilegije)
- **getInfo** (omogućuje čitanje informacija o klijentu).

Uporaba **AbstractServer** i **ConnectionToClient**

- Kreiraj podrazred od **AbstractServer** razreda.
- Implementiraj **handleMessageFromClient**.
- Napiši kod koji:
 - Kreira instancu podrazreda od **AbstractServer**
 - Poziva **listen** metodu
 - Odgovara na **clientConnected** slanjem poruke uporabom metoda:
 - **getClientConnections** i **sendToClient**, ili **sendToAllClients** (to nije callback metoda)
- Implementiraj jednu ili više drugih callback metoda.

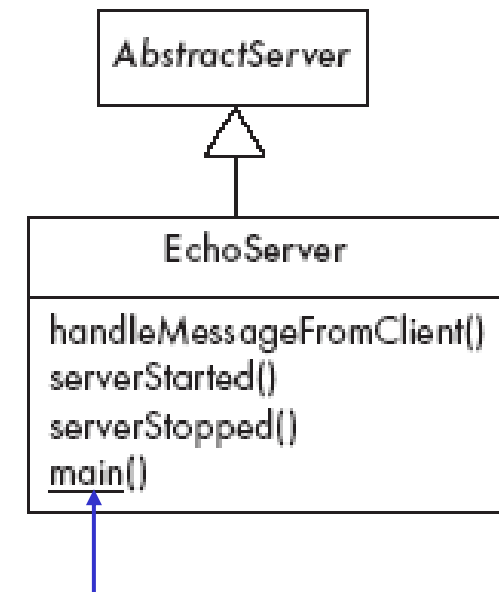
Primjena: Jednostavan "Chat"

sučelje je odvojeno
ClientConsole implementira sučelje



“static” metoda

Vraća poruke
klijenta svim
ostalim spojenim
klijentima



“static” metoda

“Chat” poslužitelj

EchoServer je podrazred od **AbstractServer**

- **main** metoda kreira novu instancu i pokreće ju.
 - Instanca razreda **EchoServer** sluša spajanje klijenata (pozivom **listen** metode) i rukuje s vezama sve dok se poslužitelj ne zaustavi.
- Tri *callback* metode samo ispisuju poruku korisniku.
 - **handleMessageFromClient**,
 - **serverStarted**
 - **serverStopped**
- Metoda **handleMessageFromClient** radi jednu dodatnu stvar, t.j.zove **sendToAllClients**
 - time se proslijeđuju poruke svim klijentima.

“Chat” klijent

Klijentski program započinje s radom **main** metodom u **ClientConsole**. Ona kreira instance od dva razreda (**ChatClient** i **ClientConsole**) koje se izvode u dvije odvojene dretve :

- **ChatClient**
 - To je podrazred od **AbstractClient**
 - Redefinira se **handleMessageFromServer**
 - jer samo zove **display** metodu korisničkog sučelja.
- **ClientConsole** (nije podrazred od **AbstractClient**)
 - Korisničko sučelje kao razred je odvojeno je od funkcionalnog dijela klijenta. **ClientConsole** implementira ovo sučelje, t.j implementira **display** metodu koja prikazuje na konzoli.
 - Prihvaća ulaz korisničkih podataka pozivom **accept** metode koja šalje sve ulazne podatke objektu razreda **ChatClient** pozovom metode **handleMessageFromClientUI**
 - Metoda **handleMessageFromClientUI** zove metodu **sendToServer** (javna metoda u **AbstractClient**).

Rizici u primjeni klijent – poslužitelj arhitekture

- **Sigurnost**

Sigurnost je veliki problem bez savršenog rješenja. Potrebno uporabiti enkripciju, zaštitne zidove (engl. *firewalls*) i sl.

- **Potreba za adaptivnim održavanjem**

Budući da se programska potpora za klijente i poslužitelja oblikuje odvojeno potrebno je osigurati da sva programska potpora bude kompatibilna prema unatrag (engl. *backwards*) i prema unaprijed (engl. *forwards*), te kompatibilna s drugim verzijama klijenata i poslužitelja.

Principi oblikovanja i raspodijeljena arhitektura

1. **Podijeli i vladaj**: Podjelom sustava na klijenta i poslužitelja je uspješan način optimalne podjele.
—Klijent i poslužitelj mogu se oblikovati odvojeno.
2. **Povećaj koheziju**: Poslužitelj osigurava kohezijski spoju uslugu klijentima.
3. **Smanji međuovisnost**: Uobičajeno je da postoji samo jedan komunikacijski kanal preko kojega se prenose jednostavne poruke.
4. **Povećaj apstrakciju**: Odvojene raspodijeljene komponente su dobar način povećanja apstrakcije.
6. **Povećaj uporabu postojećeg**: Često je moguće pronaći odgovarajući radni okvir (engl. *framework*) temeljem kojega se oblikuje raspodijeljeni sustav. Međutim, klijent-poslužitelj arhitektura je često specifična obzirom na primjenu.

Principi oblikovanja i raspodijeljena arhitektura

- 7. **Oblikuj za fleksibilnost**: Raspodijeljeni sustavi se često vrlo lako mogu rekonfigurirati dodavanjem novih poslužitelja ili klijenata.
- 9. **Oblikuj za prenosivost**: Klijenti se mogu oblikovati za nove platforme bez promjene poslužiteljske strane.
- 10. **Oblikuj za ispitivanje**: Klijenti i poslužitelji mogu se ispitivati neovisno.
- 11. **Oblikuj konzervativno**: U kod koji rukuje porukama mogu se ugraditi vrlo stroge provjere.