Oblikovanje programske potpore

2012./2013. grupa P01

Ostale arhitekture programske potpore

Prof.dr.sc. Vlado Sruk

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

Sadržaj

- Arhitektura zasnovana na događajima
- Arhitektura repozitorija podataka
- Slojevita arhitektura
- Virtualni strojevi
- Arhitektura programske potpore u upravljanju procesima

- Cilj:
  - upoznavanje specifičnih arhitektura programske podrške
  - razumijevanje prilagodbe arhitekture programske podrške rješavanju specifičnih problema i uporabi za donošenje ranih odluka o oblikovanju
  - razumijevanje karakteristika važnijih arhitekturnih stilova

Literatura

- Sommerville, I., ***Software engineering***, 8th ed, Addison Wesley, 2007.
- Taylor, R., Medvidovic N., Dashofy E.M.: ***Software Architecture: Foundations, Theory, and Practice***; 2008 John Wiley & Sons, Inc.

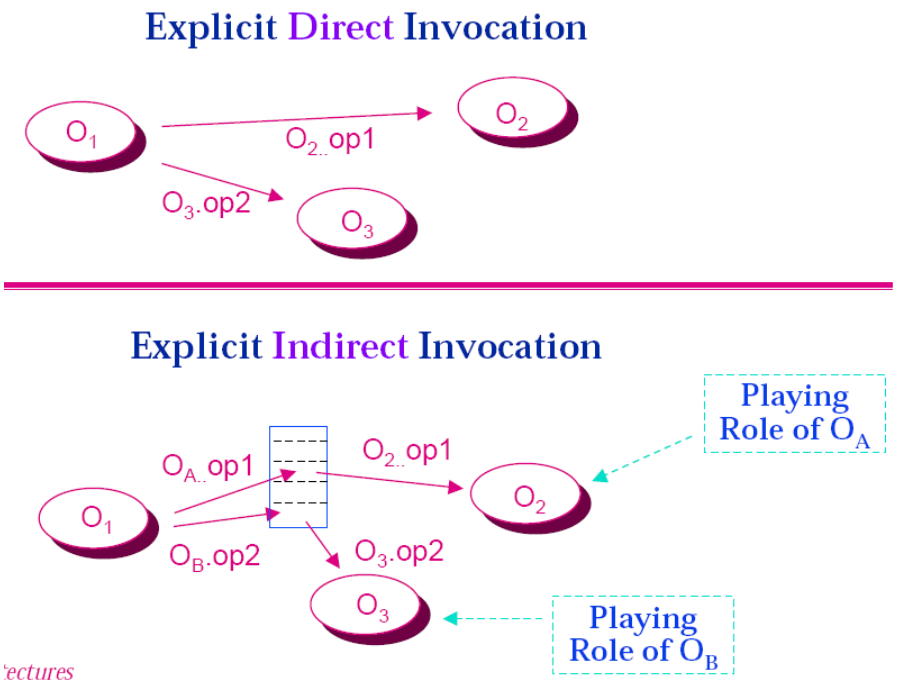
ARHITEKTURA ZASNOVANA NA DOGAĐAJIMA

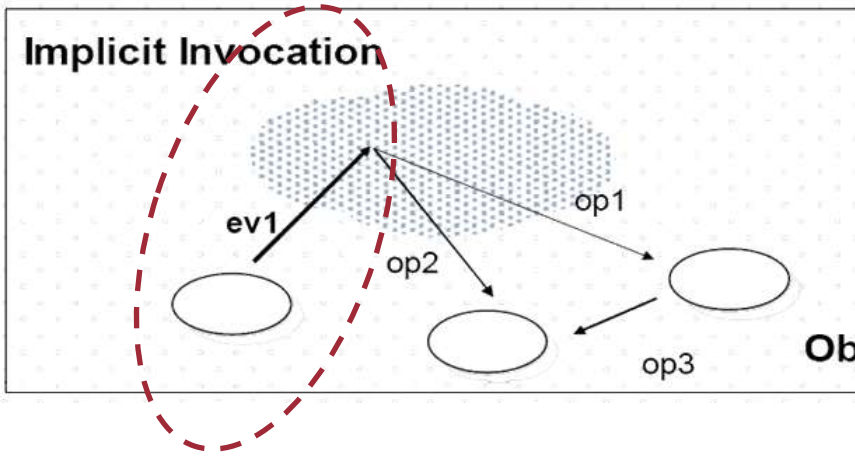
Arhitektura zasnovana na događajima

- *engl. event based architecture, implicit invocation*
- Temeljne značajke:
  - komponente se međusobno ne pozivaju eksplicitno.
  - neke komponente generiraju signale = događaje.
  - neke komponente su zainteresirane za pojedine događaje, te se prijavljuju na strukturu za povezivanje komponenata.
  - komponente koje objavljuju događaj nemaju informaciju koje će sve komponente reagirati i kako na događaj.
  - asinkrono rukovanje događajima.
  - nedeterministički odziv na događaj.
  - model izvođenja: Događaj se javno objavljuje te se pozivaju registrirane procedure.

Arhitektura zasnovana na događajima

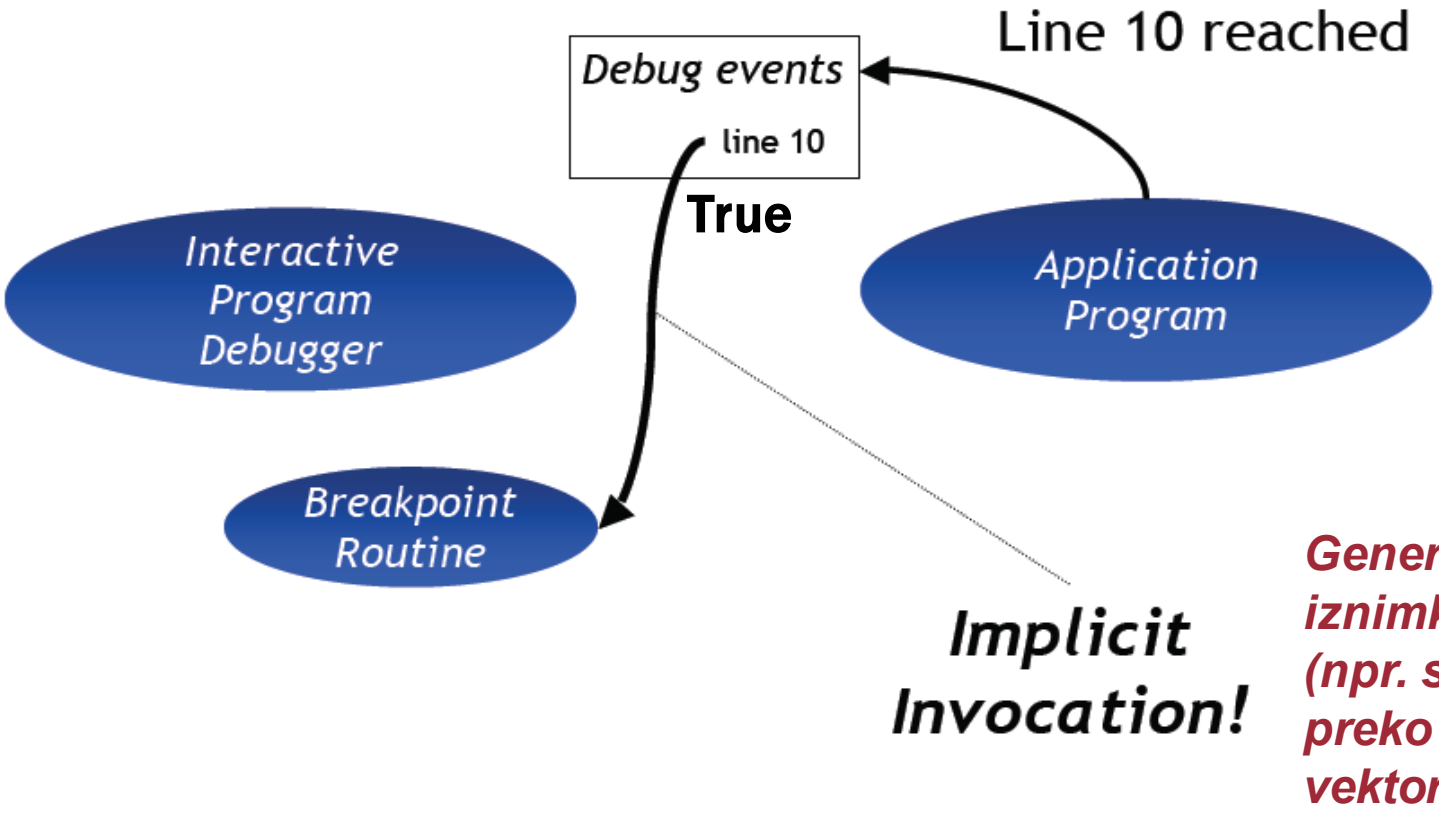
- Tipovi povezivanja:
  - izričito/eksplicitno (engl. explicit)
  - implicitno (engl. implicit)



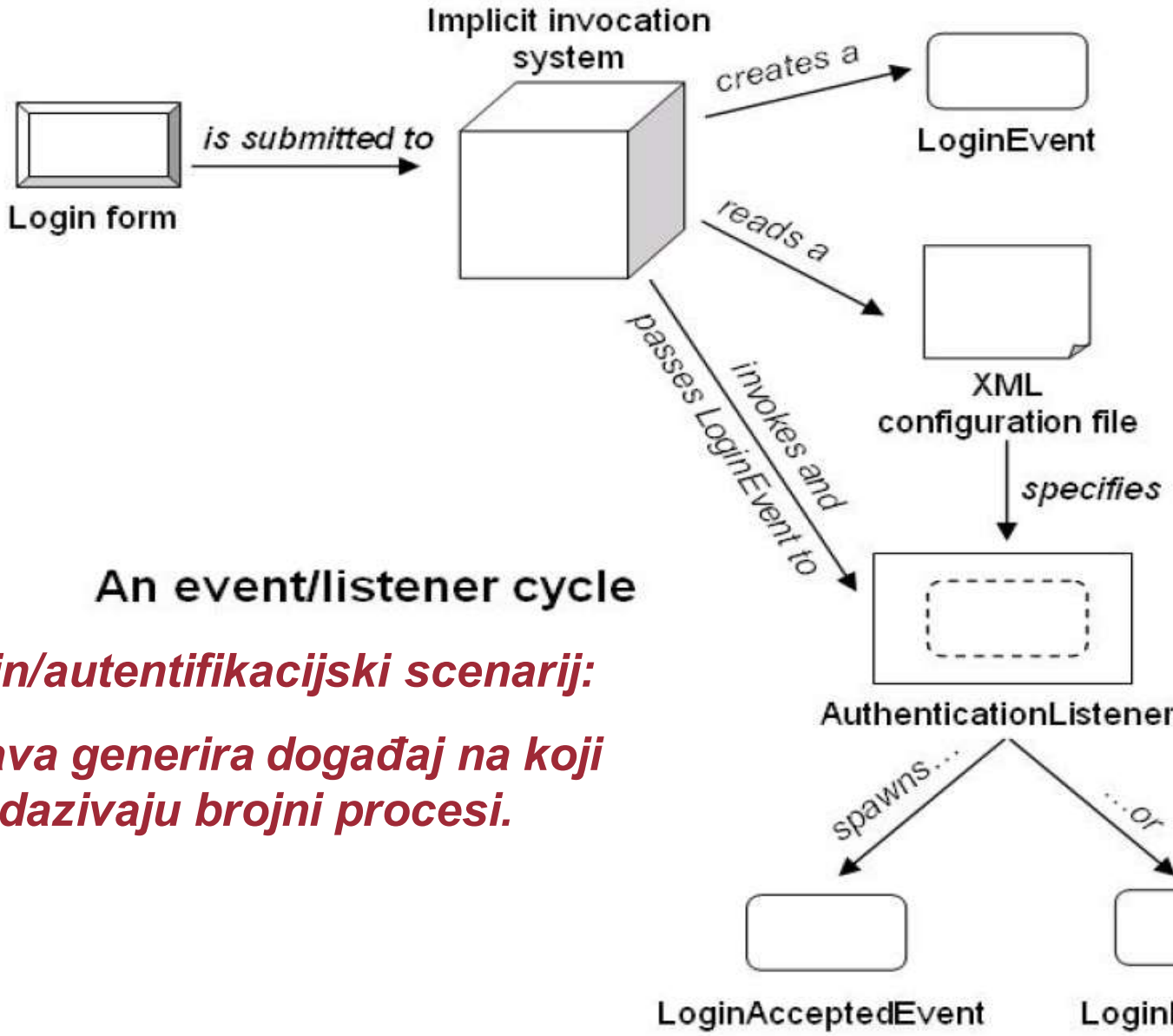


Primjer implicitnog pozivanja

- Primjenski program ne poziva rutinu za obradu događaja izravno niti neizravno.
  - Prekidna rutina (odziv na iznimku) se aktivira preko vektora.



Primjer 2:



An event/listener cycle

Login/autentifikacijski scenarij:

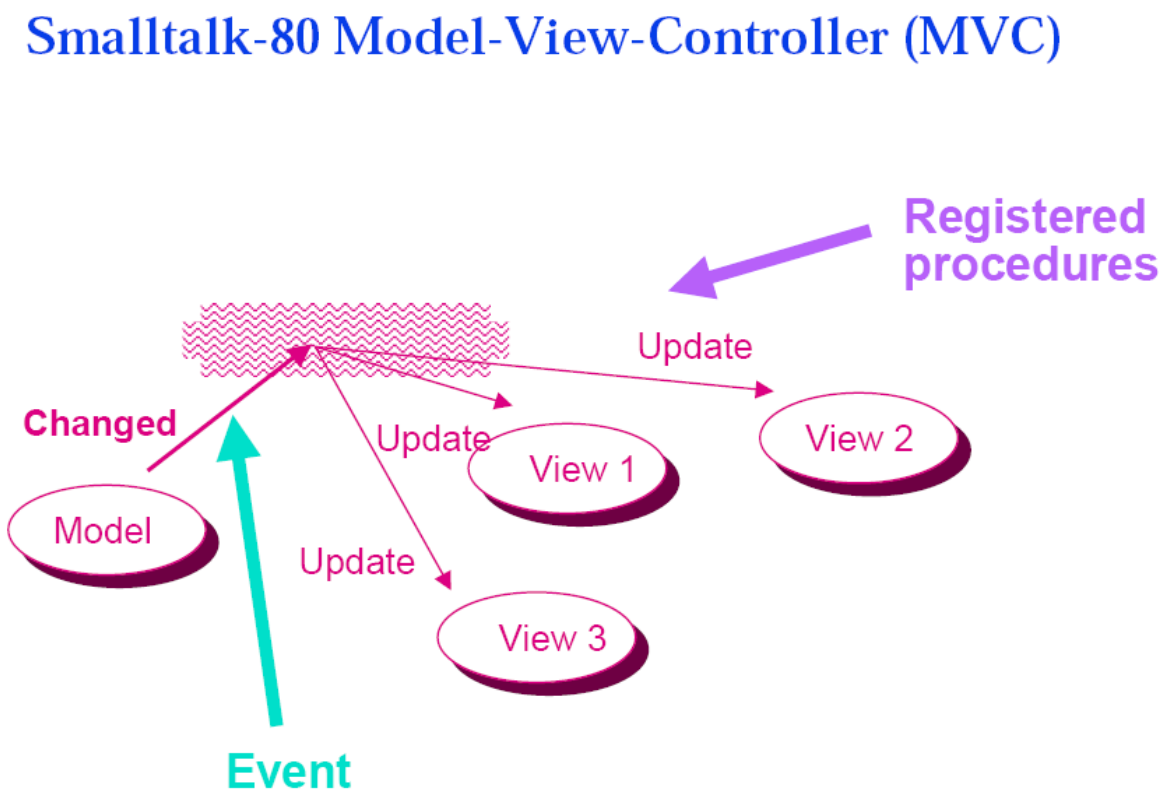
Prijava generira događaj na koji se odazivaju brojni procesi.

Prednosti i nedostaci

- Prednosti:
  - omogućuje razdvajanje i autonomiju komponenata.
  - snažno podupire evoluciju i ponovno korištenje.
  - jednostavno se uključuju nove komponente bez utjecaja na postojeće.
- Nedostaci:
  - komponente koje objavljuju događaje nemaju garancije da će dobiti odziv.
  - komponente koje objavljuju događaje nemaju utjecaja na redoslijed odziva.
  - apstrakcija događaja ne vodi prirodno na postupak razmjene podataka (možda je potrebno uvođenje globalnih varijabli).
  - teško rasuđivanje o ponašanju komponenata koje objavljuju događaje i pridruženim komponentama koje su registrirane uz te događaje (nedeterministički odziv)

Obrazac model-pogled-nadglednik

- MVC obrazac, *engl. Model View Controller*
- Stilistička varijacija arhitekture zasnovane na događajima

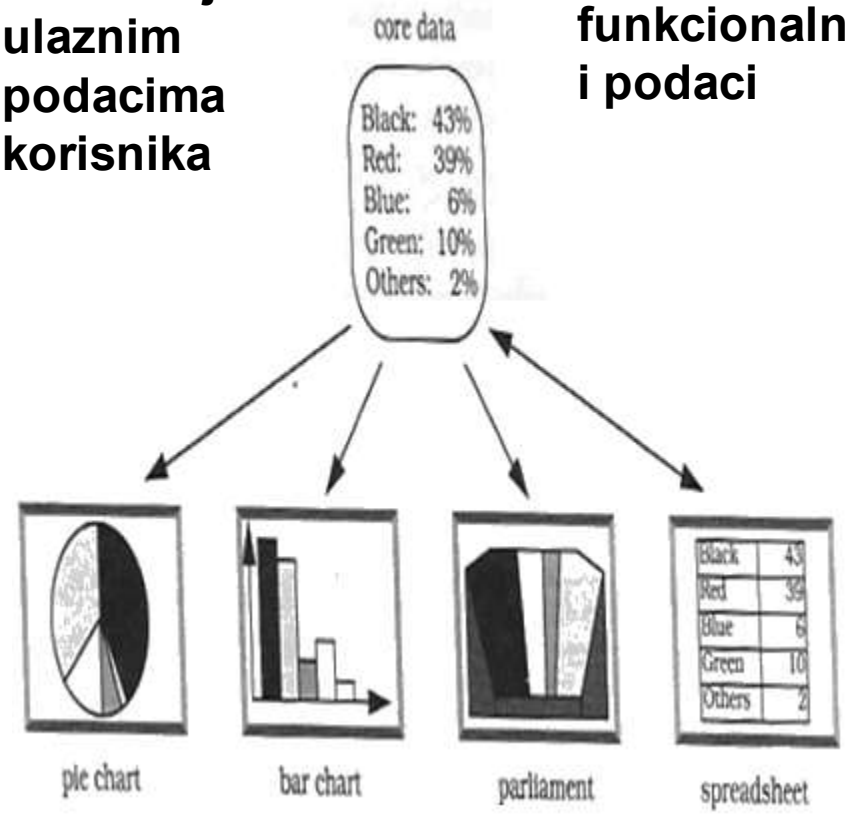


MVC Primjer: Proračunske tablice

- Sustav proračunskih tablica za upis i pregled podataka, npr. Excel.
- Bez značajnih promjena cjelokupnog sustava postoji potreba za:
  - dodavanjem nove neinteraktivne vizualizacije podataka.
  - dodavanjem nove interaktivne vizualizacije podataka

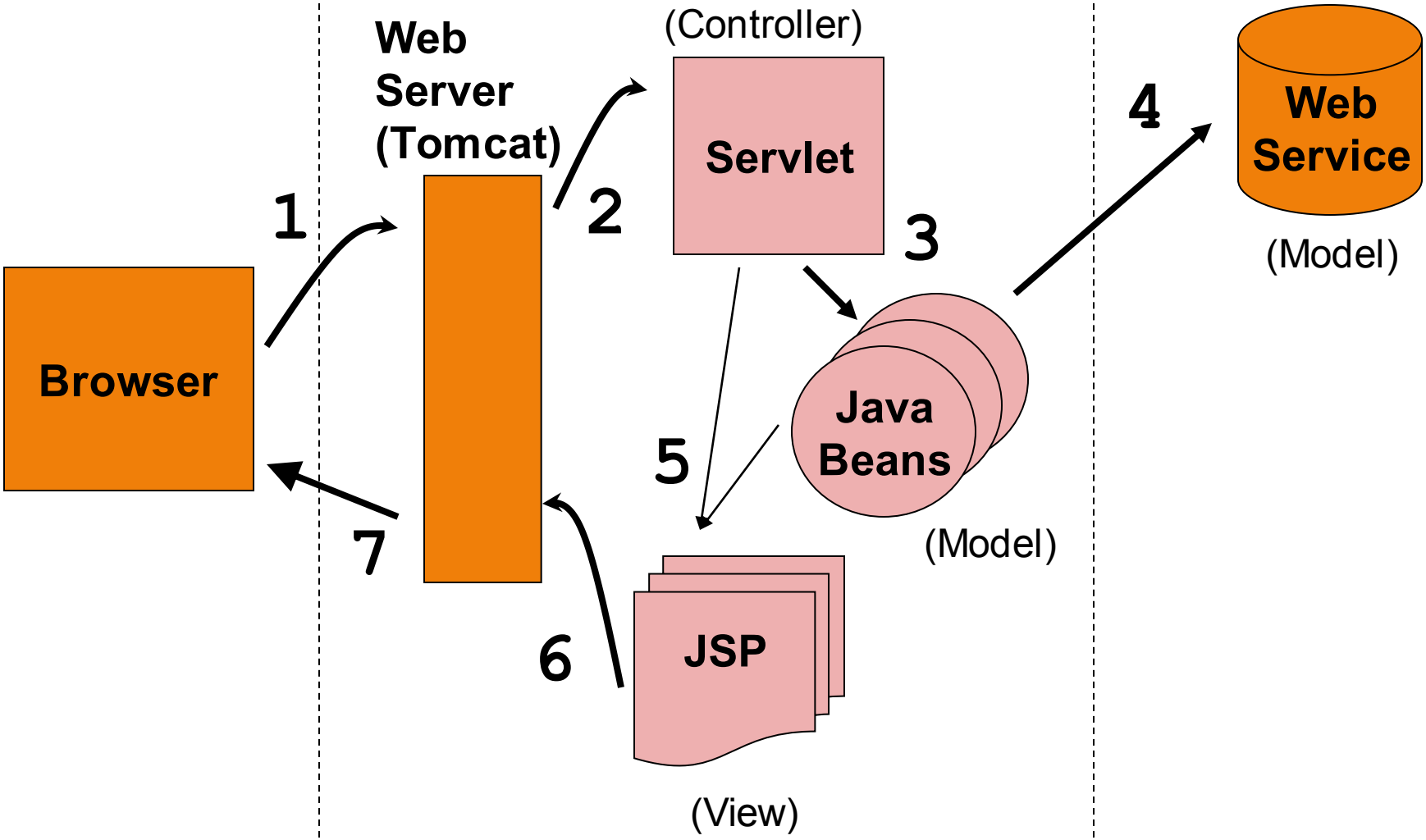
Controller: **Rukovanje ulaznim podacima korisnika**

Model: **Jezgrene funkcionalnosti i podaci**



View. **Prikaz informacija**

Primjer Web aplikacija



MVC obrazac

1. Akcija klijenta generira zahtjev za određenom URL stranicom.
2. Tomcat otprema Java servlet koji rukuje s URL-om.
3. Servlet kao "***Controller***" interpretira akciju klijenta i šalje upit ili traži promjenu u Java Beans ("***Model***").
4. Java Beans koristi vanjske resurse kao što je npr. Web usluga.
5. Ovisno o stanju modela servlet odabire prikladnu vizualizaciju i prosljeđuje Java Beans do Java Servlet Pages (JSP), t.j. do "***View***" komponente.
6. Tomcat konvertita JSP u HTML.
7. HTML se prosljeđuje klijentu.

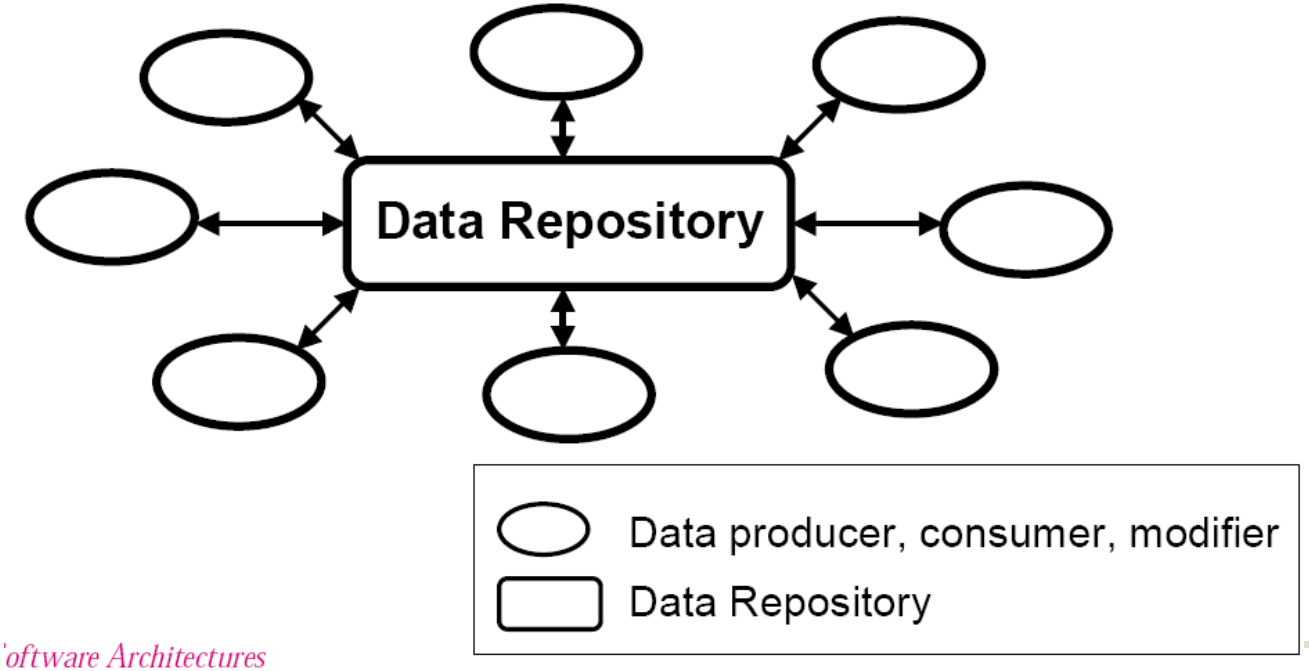
Zaključak

- Vrlo značajna arhitektura programske potpore.
- Intenzivna primjena u procesorskim sustavima s više jezgara (*engl. multicore*).
- Uključena u najnoviju specifikaciju UML-a (UML 2.0).

ARHITEKTURA REPOZITORIJA PODATAKA

Arhitektura repozitorija podataka

- Predstavlja veliku skupinu sustava uz mnoge varijacije upravljanja i dijeljenja podataka. Arhitektura obuhvaća načine prikupljanja, rukovanja i očuvanja velike količine podataka.
- Prirodni primjer su baze podataka, ali ne i jedini (npr. oglasna ploča)
- Pogled s najviše razine na ovu arhitekturu (dvije vrste komponenta):

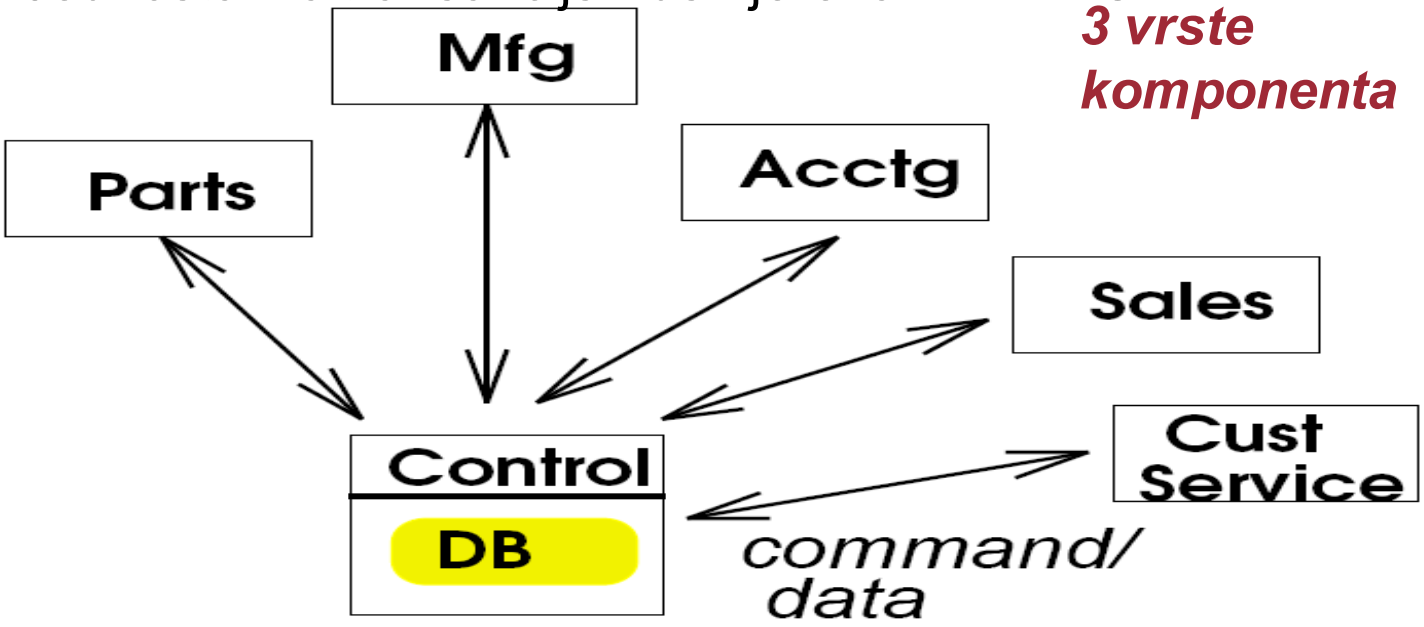


Komponente repozitorija

- Dvije različite vrste komponenta:
  - Središnji repozitorij podataka (predstavlja trenutno stanje)
  - Kolekcija nezavisnih komponenta koje operiraju nad središnjim repozitorijem (proizvođači i korisnici podataka).
- Temeljna prednost ove arhitekture je jednostavno dodavanje i povlačenje proizvođača i korisnika podataka.
- Problemi su koncentrirani oko:
  - sinkronizacije
  - konfiguracije i upravljanje shemama struktura podataka
  - atomičnosti
  - konzistencije
  - očuvanja (perzistencije)
  - performanse

Arhitektura repozitorija podataka

Stil: Jednostavna transakcijski usmjerena BAZA PODATAKA



Komponente modificiraju ili čitaju podatke (preko *transakcijskog upita, engl. query)* iz baze - DB.

Baza podataka skladišti perzistentne podatke između različitih transakcija.

Nema fiksnog uređenja redoslijeda između transakcija.

Komponenta “Control” upravlja paralelnim pristupima u DB.

E-R model

- Model entiteti-odnosi (engl. Entity – Relationship) baze podataka:
  - je koncepcijski model podataka koji opisuje podatkovne zahtjeve u informacijskom sustavu na jednostavan i razumljiv grafički način.
  - služe za modeliranje scenarija
  - postoje mnoge verzije E-R dijagrama koji se razlikuju kako u svom izgledu i značenju
  - E-R sheme imaju formalnu semantiku (značenje) koje se mora temeljito razumjeti, kako bi se stvorili ispravni dijagrami
- Podatkovni zahtjevi se opisuju simbolima E-R sheme.
- E-R shema je komparabilna (nije identična!) UML dijagramu razreda.

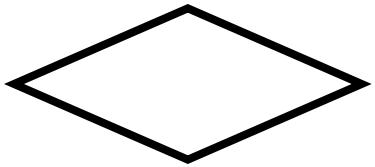
Entities

- *Represent classes of objects (facts, things, people,...) that have properties in common and an autonomous existence, e.g.,* `City`, `Department`, `Employee`, `Purchase` *and* `Sale`.
- *An instance of an entity is an object in the class represented by the entity, e.g.,* `Stockholm`, `Helsinki`, *are examples of instances of the entity* `City`; `Peterson` *and* `Johanson` *are examples of instances of the* `Employee` *entity.*

E-R odnosi

- *odnosi, veze*

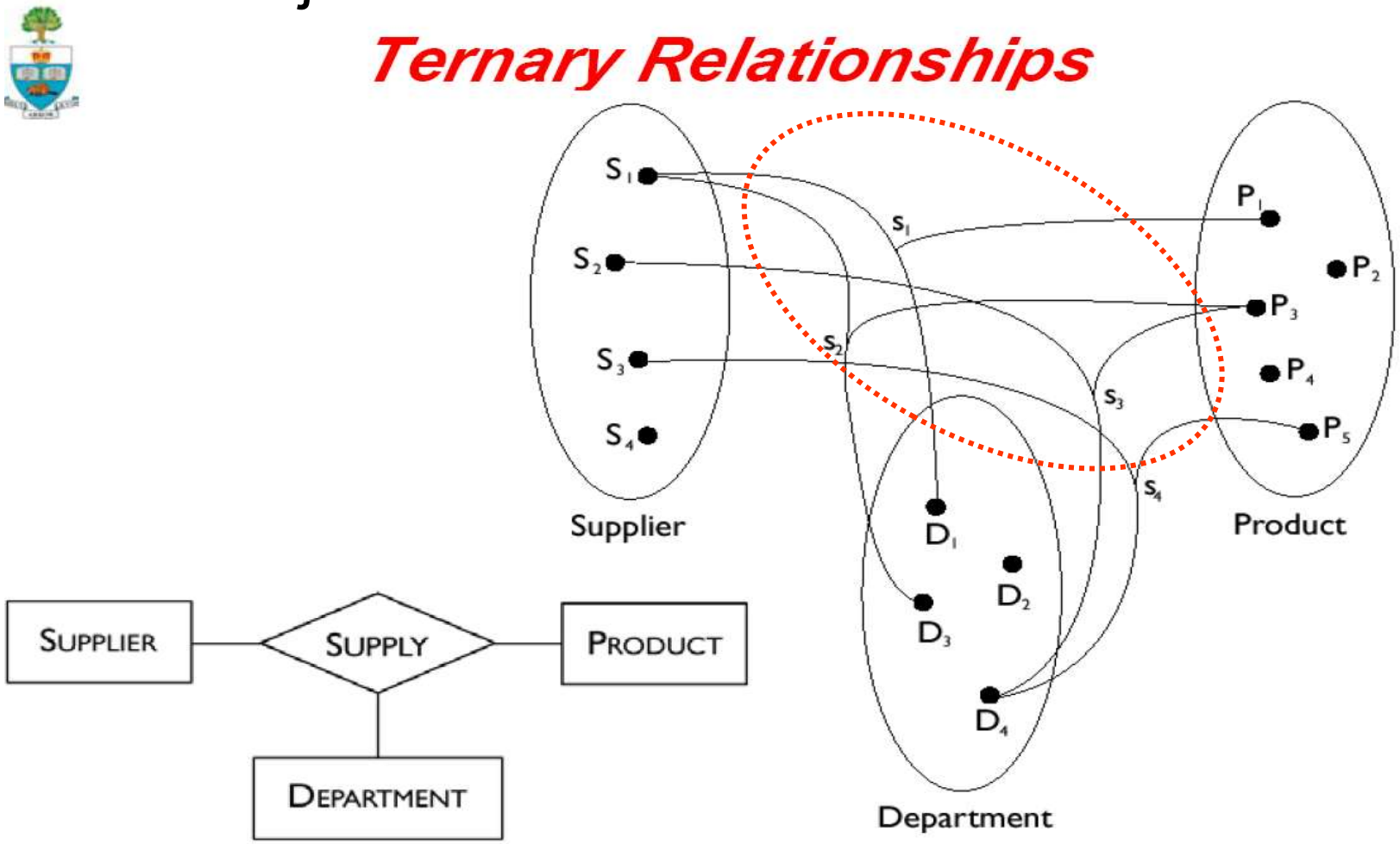
Relationships



- *They represent associations between two or more entities.*
- *Residence is an example of a relationship that can exist between the entities* `City` *and* `Employee`*;* `Exam` *is an example of a relationship that can exist between the entities* `Student` *and* `Course`*.*
- *An instance of a relationship is an n-tuple made up of instances of entities, one for each of the entities involved.*
- *The pair* `(Johanssen,Stockholm)`*, or the pair* `(Peterson,Oslo)`*, are examples of instances of the relationship* `Residence`*.*

E-R odnosi 2

- Svaka instancija odnosa *Supply* predstavlja jednu određenu trojku.



E-R: značajke

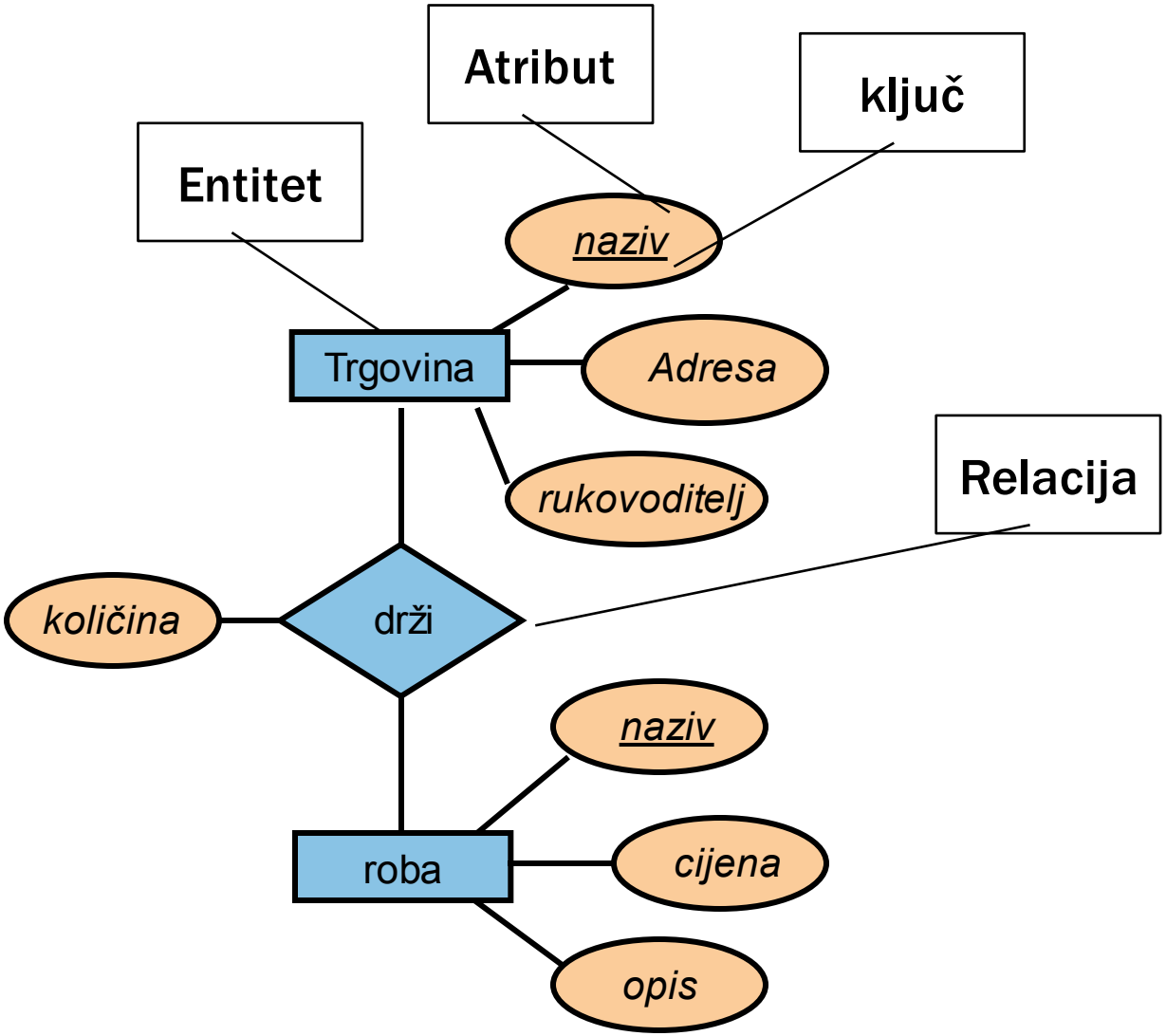
- značajke (engl. attributi)

Attributes

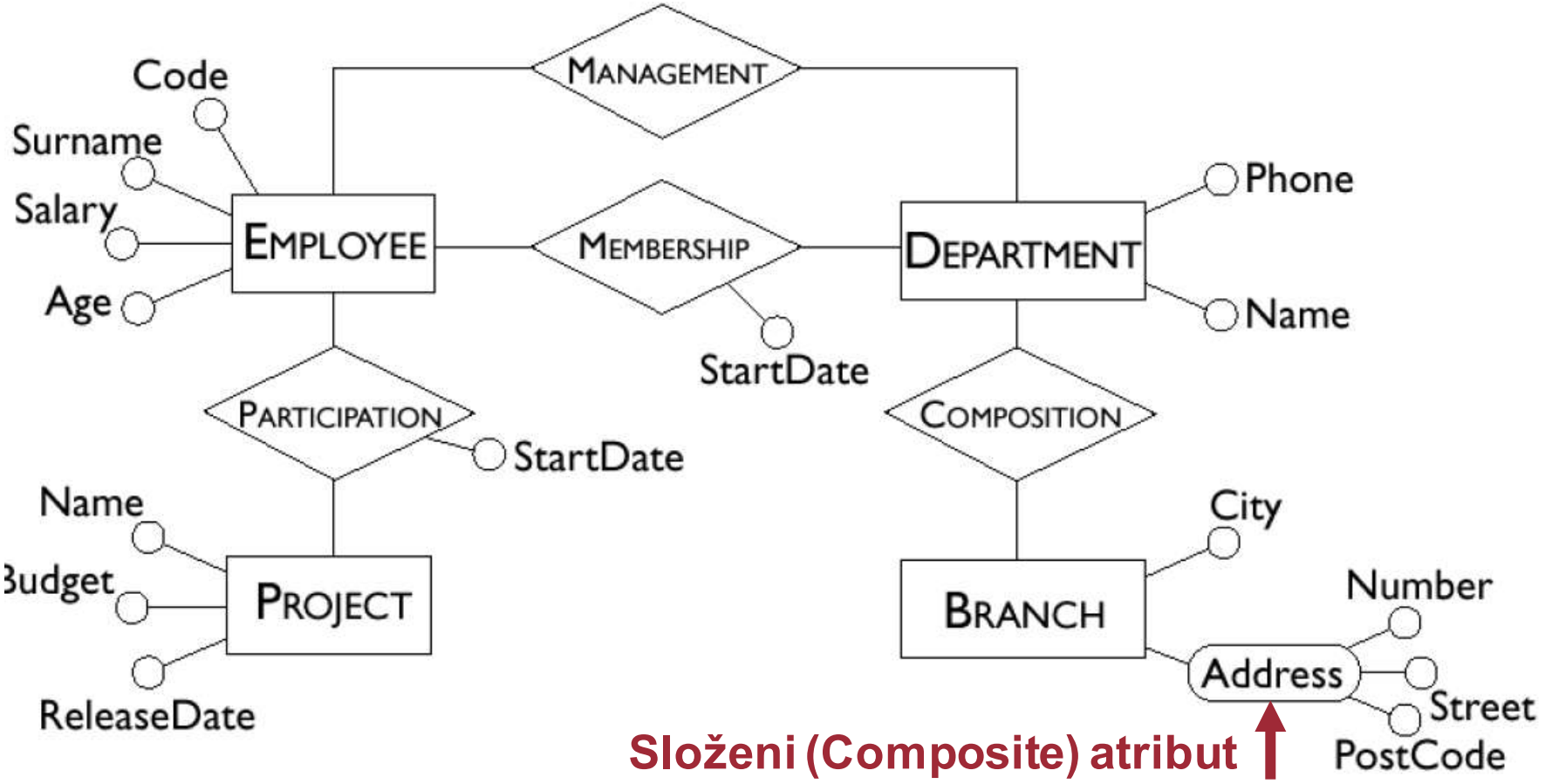
- *Describe elementary properties of entities or relationships.*
- *For example,* `Surname`, `Salary` *and* `Age` *are attributes of* `Employee`, *while* `Date` *and* `Mark` *are attributes of relationship* `Exam` *between* `Student` *and* `Course`.
- *An attribute associates with each instance of an entity (or relationship) one or more values belonging to its* ***domain****.*
- *Attributes may be single-valued, or multi-valued.*

Odnosi se npr. na "slot" s više polja.

Primjer: E-R shema



Primjer E-R sheme



E-R: brojnost

Cardinalities

- *These are specified for each entity participating in a relationship and describe the maximum and minimum number of relationship instances that an entity instance can participate in.*

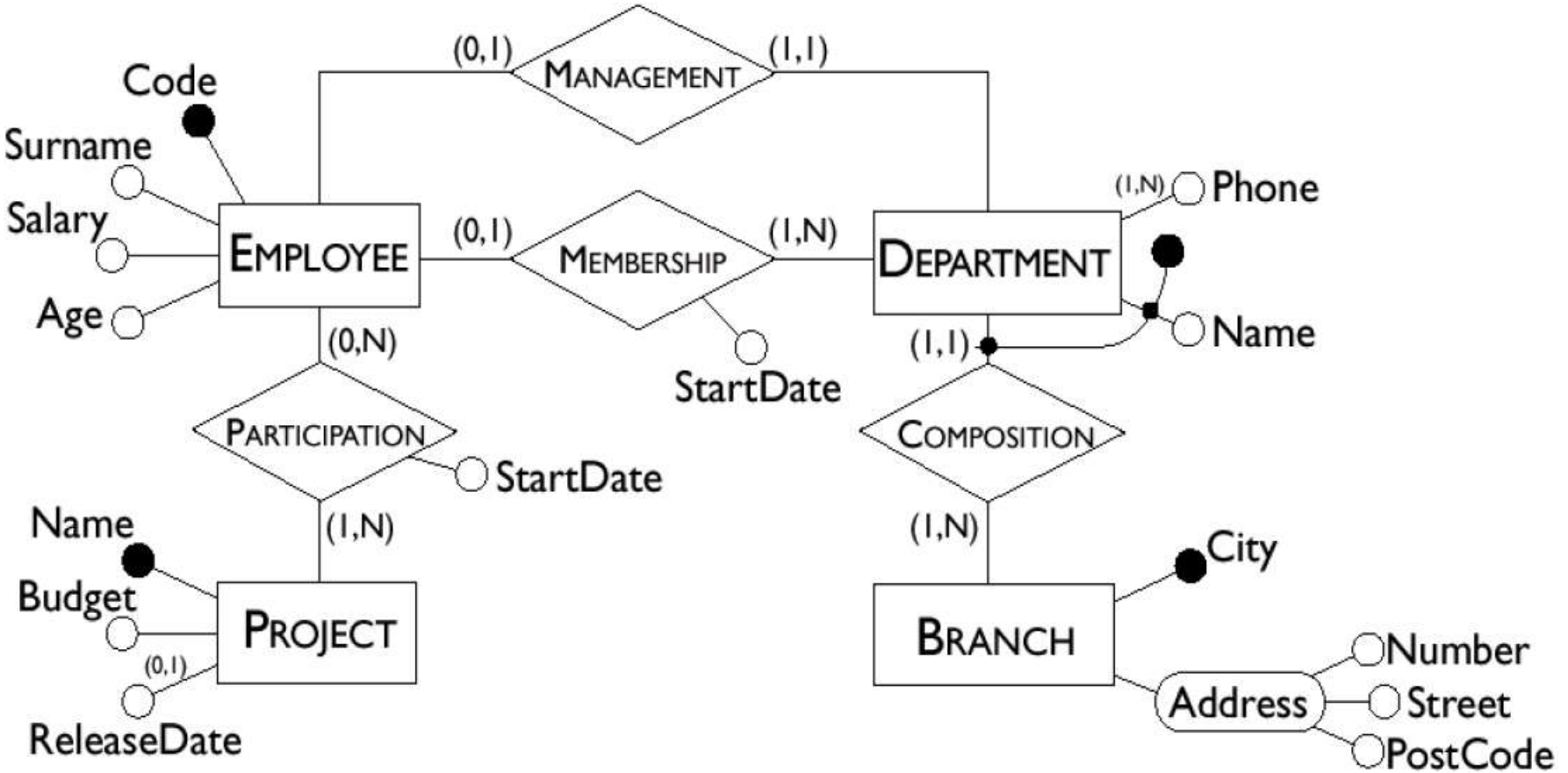
1 Employee min 1 max 5 assignments **1 Task up to 50 assignments**

E-R: identifikatori

(identifikatori, ključevi) ***Identifiers***

- ***Identifiers*** *(or* ***keys****) consist of one or more attributes which identify uniquely instances of an entity.*
- *For example,* `Person` *may be identified by the attribute* `socialInsurance#`*. Alternatively, it may be identified by* `firstName`, `middleName`, `lastName`, `address`.
- *In most cases, an identifier is formed by one or more attributes of the entity itself (****internal*** *identifier).*
- *Sometimes, the attributes of an entity are not sufficient to identify its instances and other entities are involved in the identification (****external*** *identifiers).*
- *An identifier for a relationship consists of identifiers for all the entities it relates.*

Primjer: E-R shema s identifikatorima



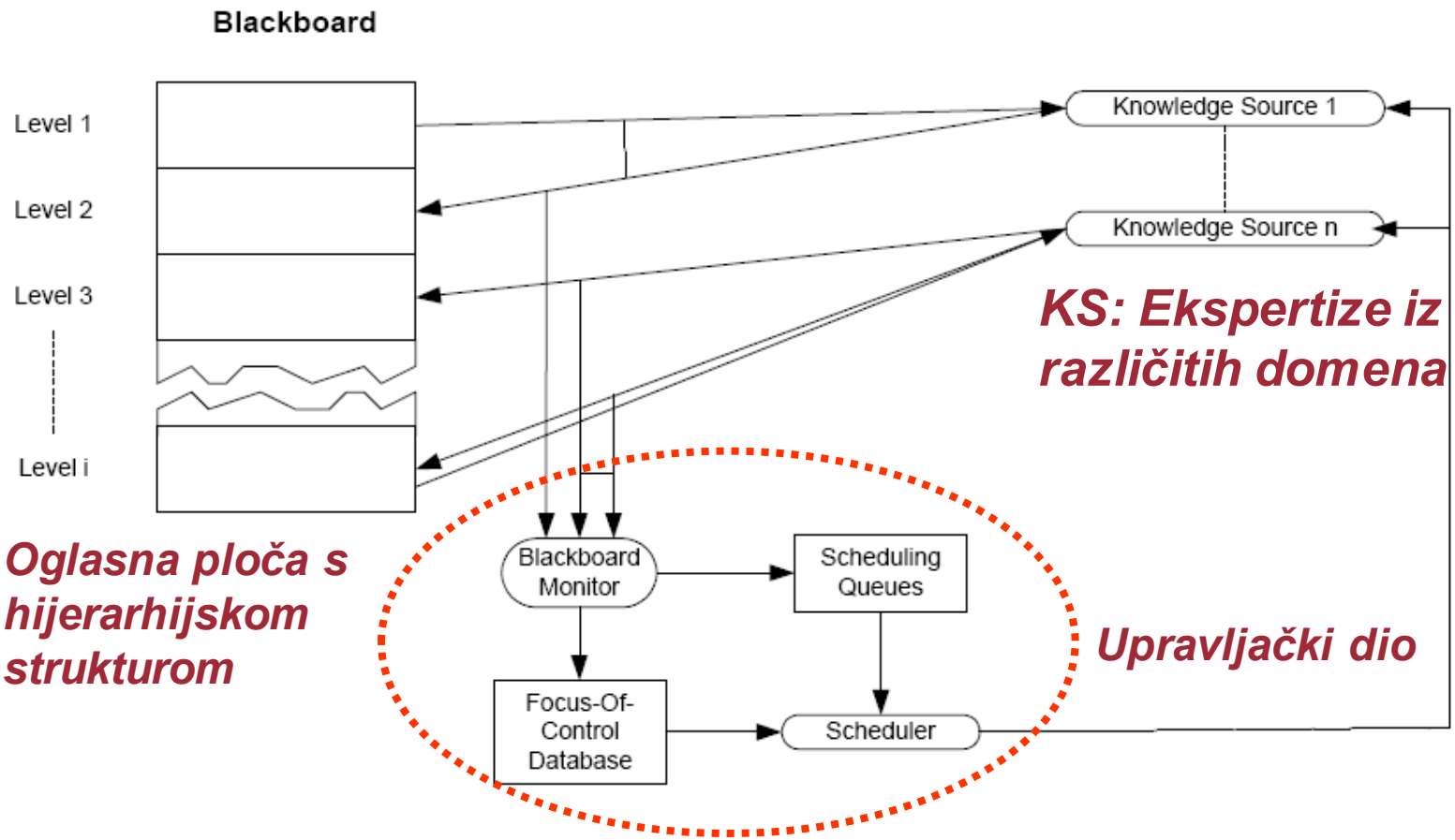
Usporedba E-R modela i UML dijagrama

- E-R dijagrami dozvoljavaju n-struku relaciju.
  - U dijagramima razreda postoji samo relacija (asocijacija) između dva razreda.
- E-R dijagrami dozvoljavaju atribute s više vrijednosti. UML razredi ne.
- E-R dijagrami specificiraju identifikatore. Dijagrami razreda ne.
- Dijagrami razreda omogućuju dinamičku klasifikaciju
  - tijekom izvođenja objekt može promijeniti razred
  - To je složen postupak, ali u E-R modelu nije ni predviđen.
- UML razredi imaju pridružene metode, ograničenja (engl. constraints) i pre/post uvjete.

Stil: Oglasna ploča

- engl. blackboard
- Temeljni koncept:
  - Mnogi eksperti promatraju rješavanje zajedničkog problem na ploči (npr. kriminalisti).
  - Svaki ekspert dodaje svoj dio u rješavanju cjelokupnog problema.
- Osnovni dijelovi:
- Izvori znanja – KS (engl. knowledge sources), odvojeni i nezavisni dijelovi primjenskog znanja (eksperti). Ne surađuju međusobno izravno već samo putem oglasne ploče.
- Oglasna ploča, podaci o stanju problema, organizirani prema primjenskoj hijerarhiji. Izvori znanja, aktivirani promjenom sadržaja na oglasnoj ploči, mijenjaju pojedine sadržaje što inkrementalno dovodi do rješenja problema.
- Upravljanje stanjem oglasne ploče. Izvori znanja se odazivaju oportunistički kada se promjene na oglasnoj ploči na njih odnose.

Stil: Oglasna ploča



Važna distinkcija:

***Baza podataka*: vanjski procesi iniciraju promjenu sadržaja.**
***Oglasna ploča*: promjena sadržaja inicira vanjske procese (KS).**

Stil: Oglasna ploča

- Obilježja problema prikladnih za arhitekturu oglasne ploče
  - Nema izravnog algoritamskog rješenja (višestruki pristup rješavanju, potrebna ekspertiza iz različitih domena).
  - Neizvjesnost (pogreške i varijabilnost u podacima, srednji ili niski omjer “signala prema šumu” u podacima).
  - Aproksimativno rješenje je dovoljno dobro (ne postoji jedan diskretan odgovor na problem ili ispravan odgovor može varirati.
- Primjeri primjene:
  - Obradba signala
  - Planiranje, logistika, dijagnostika
  - Optimizacija prevodioca programskih jezika

Svojstva

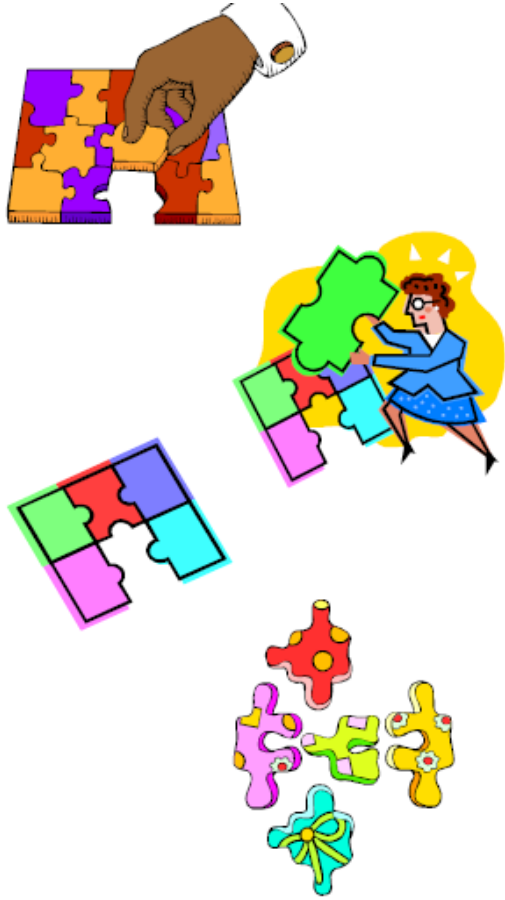
- U ovoj arhitekturi nije unaprijed specificirano:
  - Struktura znanja (kako je predstavljeno znanje).
  - Kako izvori znanja (KS) dohvaćaju podatke.
  - Kako izvori znanja (KS) zapisuju djelomične odgovore na oglasnu ploču.
  - Kako izvori znanja (KS) koriste podatke sa oglasne ploče.
  - Kako se određuje kvaliteta rješenja.
  - Kako se ustvari upravlja ponašanjem izvora znanja (KS).

- Znanje može biti predstavljeno na različite načine:
  - jednostavnim funkcijama,
  - kolekcijom složenih logičkih izraza, pravilima i sl.
- Predstavljeno znanje izaziva odgovarajuću akciju na oglasnoj ploči.

Struktura podataka oglasne ploče

- podaci na oglasnoj ploči se uobičajeno struktruriaju hijerarhijski

Level 4	assemble chunks
Level 3	build chunks of edges build chunks of sky :
Level 2	collect edge pieces collect sky pieces :
Level 1	Turn all pieces picture side up



Upravljanje oglasnom pločom

- Mehanizmi upravljanja:
  - Upravljano događajima ili signalima (prekidima)
  - Upravljanje pozivanjem procedura
- Strategije upravljanja:
  - Podatkovno inicirano (engl. Data driven). Slijedeći korak je definiran stanjem podataka, te se poziva odgovarajući KS.
  - Inicirano ciljem (engl. Goal driven). Upravljačka funkcija poziva KS koji najviše doprinosi pomaku prema cilju. Općenito je teško kodirati ciljeve.
  - U praksi se koristi kombinacija ovih dviju strategija.

Prednosti i nedostaci oglasne ploče

- *Intencijsko i reaktivno ponašanje – kako se napreduje prema rješenju*
  - Prednost: jednostavno integriranje različitih autonomnih sustava.
  - Nedostatak: Oglasna ploča može biti vrlo složena arhitektura, posebice ako su komponente prirodno međuzavisne.
- *Predstavljanje i obradba neizvjesnog znanja*
  - Prednost: Oglasna ploča je dobra arhitektura za ovaj tip problema.
- *Sigurnost i tolerancija na kvarove*
  - Prednost: podsustavi mogu promatrati oglasnu ploču i obratiti pažnju na potencijalne poteškoće.
  - Nedostatak: Oglasna ploča je kritičan resurs.
- *Fleksibilnost*
  - Prednost: oglasna ploča je inherentno fleksibilna

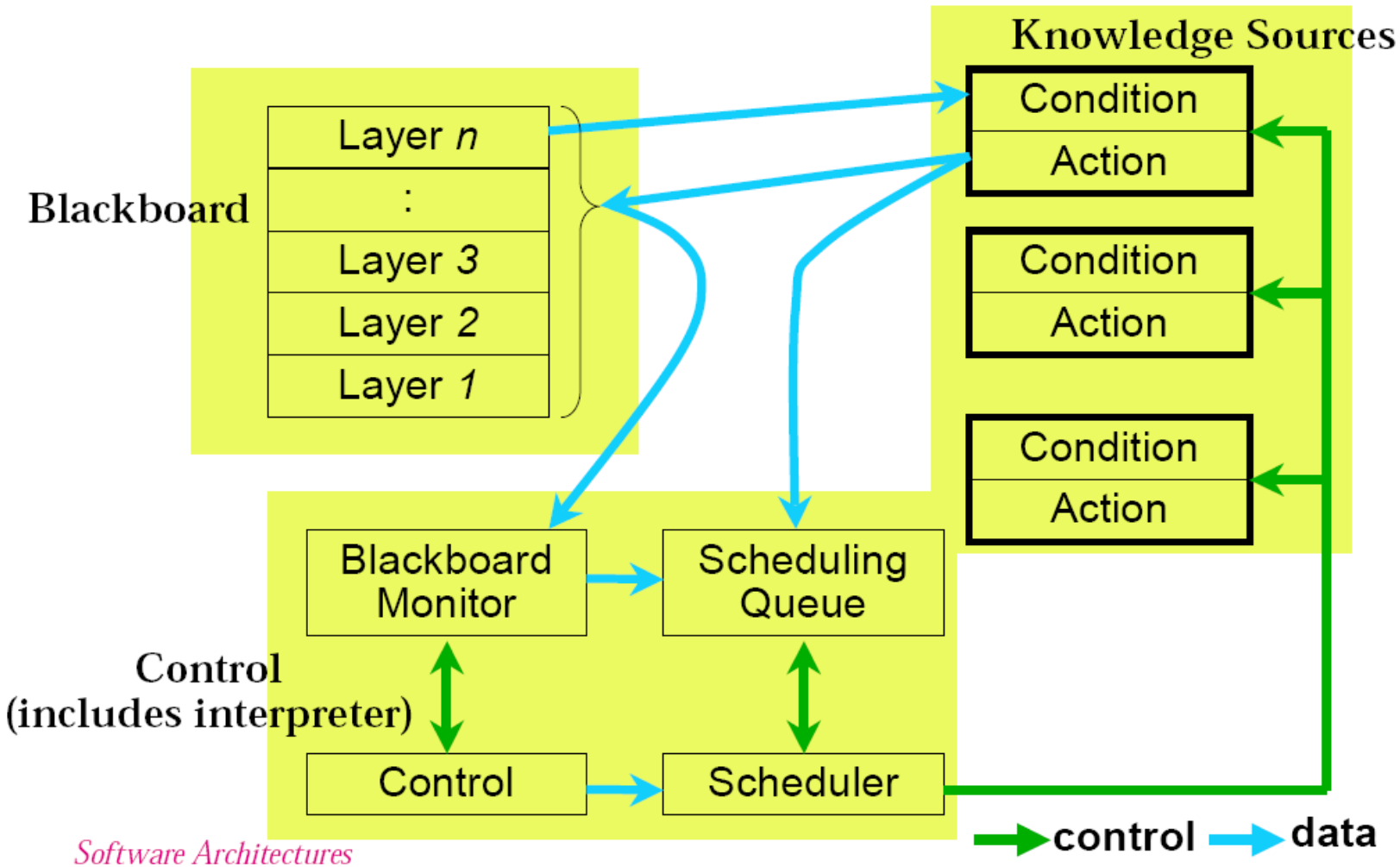
Primjer: Hearsay

- sustav za raspoznavanje govora s velikim vokabularom (CMU, 1976). Složena struktura problema motivirala je istraživanje organizacije i uporabe znanja u računalnim sustavima.

- Oglasna ploča daje višestruko i hijerarhijski organizirano predstavljanje govora:
  - Valni oblici
  - Fonemi
  - Slogovi
  - Rečenice

- Cjelokupni problem se razdvaja na akustičku, fonetski, leksičku, sintaktičku i semantičku razinu (aktiviranje različitih KS).
- Sustav rangira verzije potencijalnih rečenica metrikom vjerodostojnosti i daje najvjerojatniji prijevod.

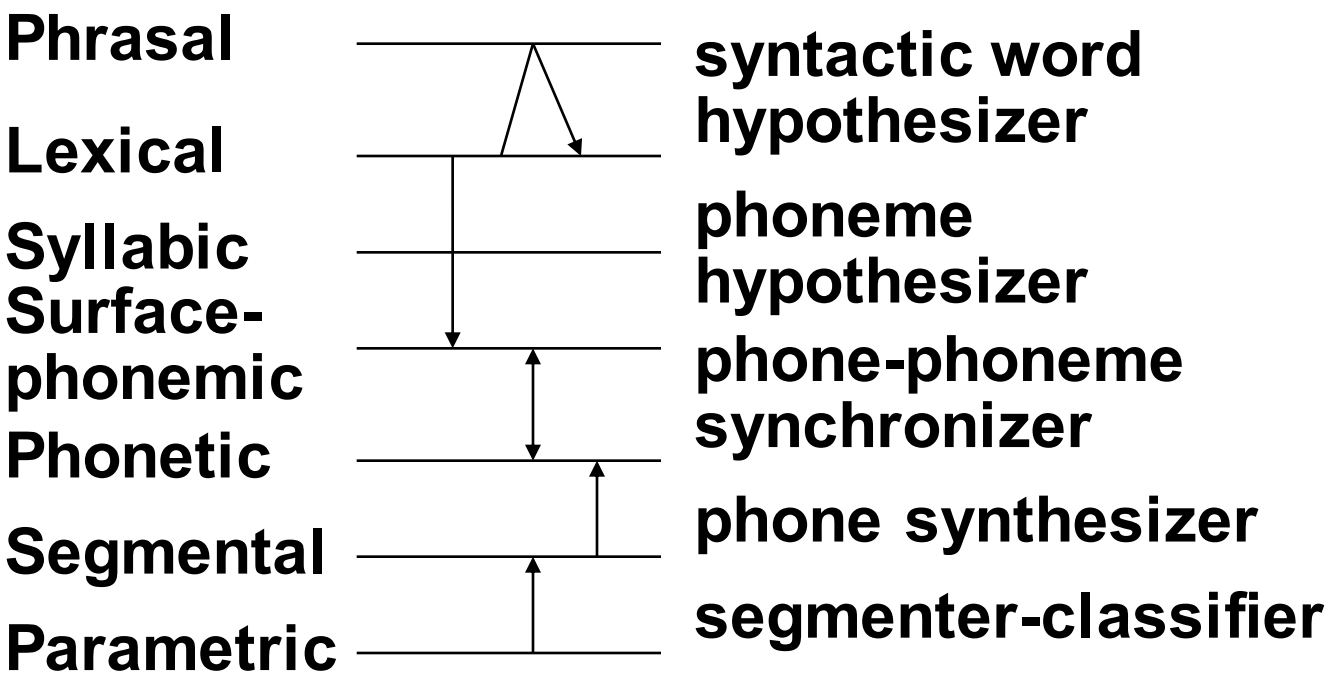
Primjer: Hearsay



Hearsay II: Organizacija oglasne ploče

Hijerarhijske razine

Izvori znanja



Primjer: Prevoditelj

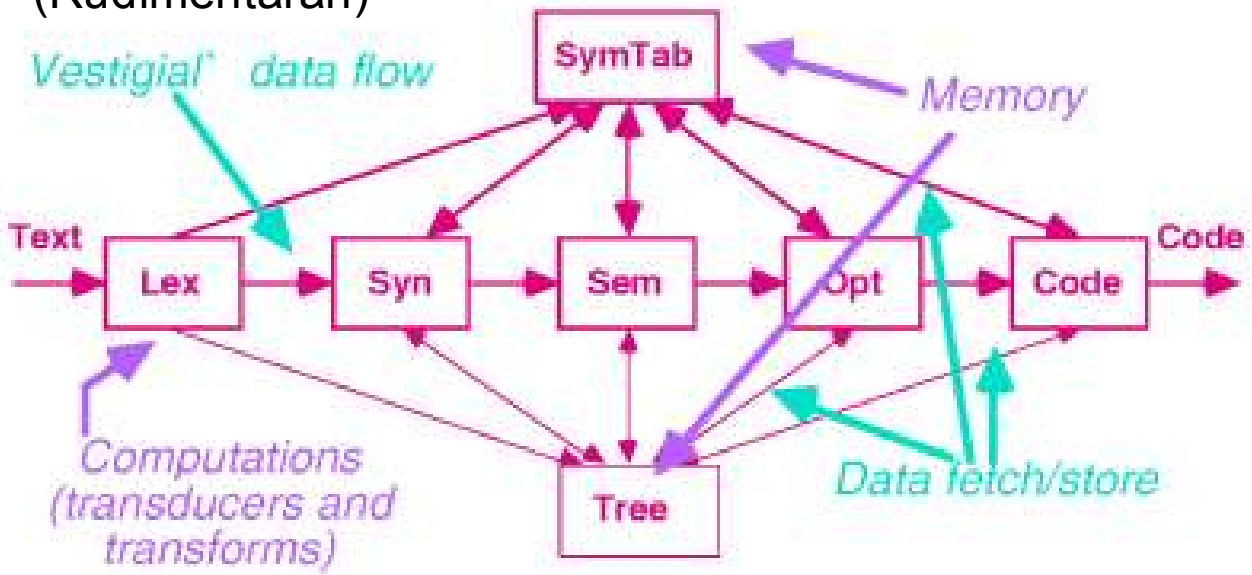
- Inicijalno: prevođenje izvornog u objektni kod (kompilator, knjižnica - library, povezivanje – linker, “make”).
- Razvoj uključuje zapis verzije, dokumentaciju, analizu, upravljanje konfiguracijama, itd.
- Traži se čvršća integracija pojedinačnih alata (u 20 godina još nije postignuta).
- Tradicijski prevoditelj (1970, sekvencijski)
- *(arhitektura protoka podataka)*

plus odvojena tablica simbola

Primjer: Prevoditelj

Example: Modern Canonical Compiler

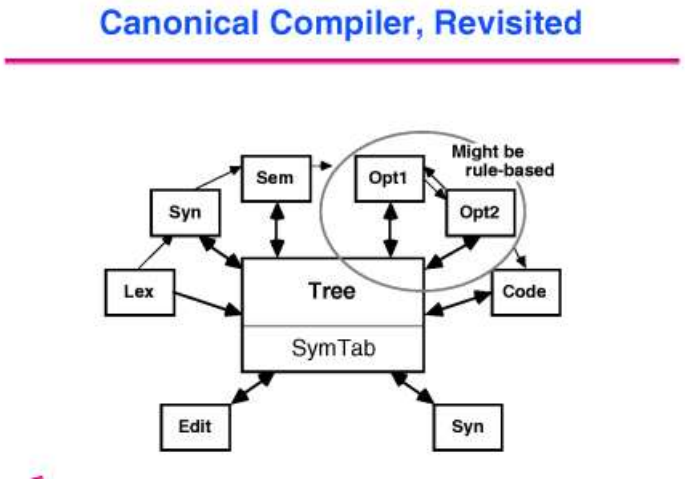
(Rudimentaran)



1985, "attribute" gramatika, stablo parsiranja

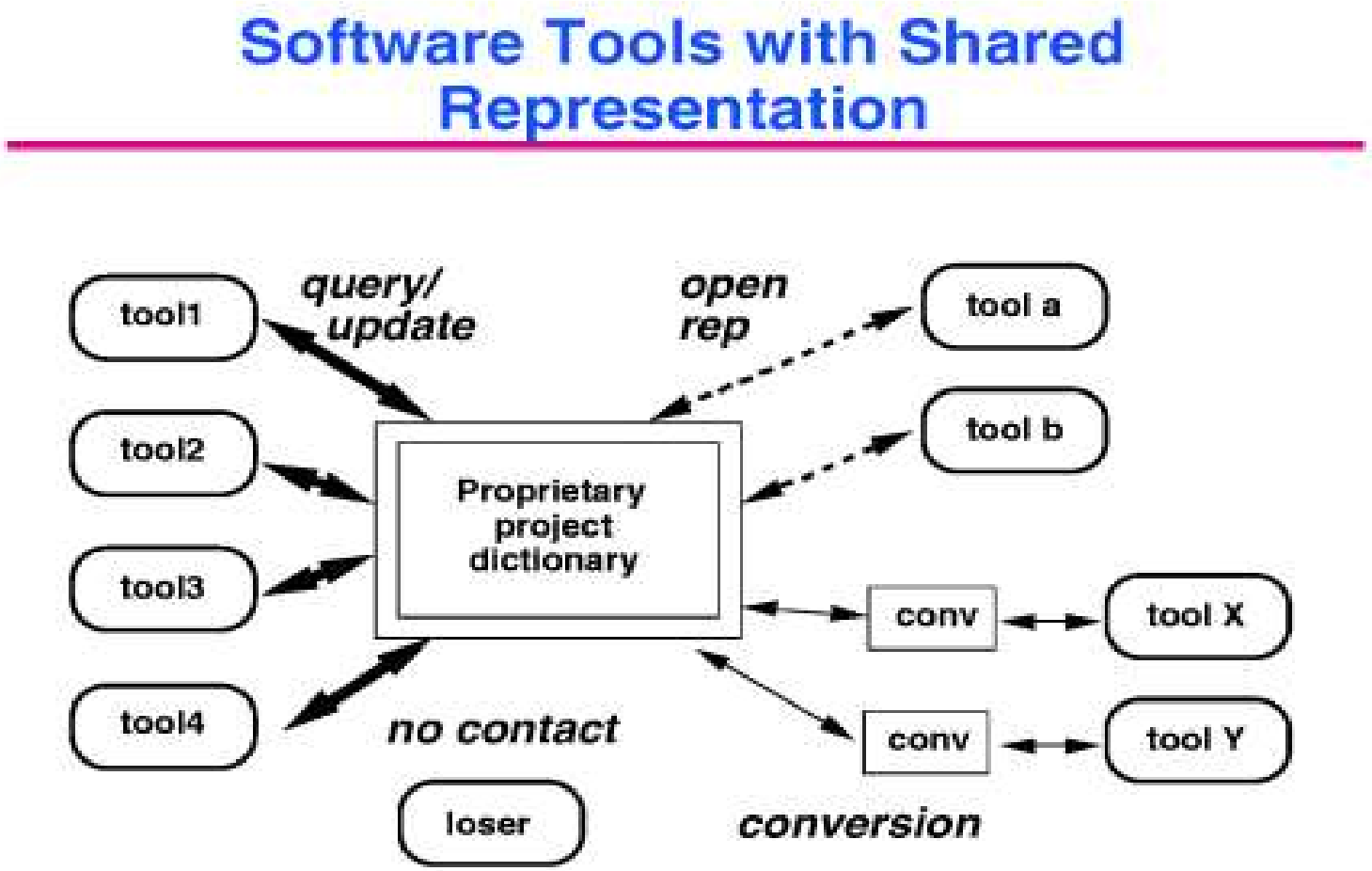
Primjer: Prevoditelj

- Informacija u repozitoriju nije perzistentna od jedne uporabe (prevođenja) do druge.



CASE okruženje

Poželjno generičko predstavljeno arhitekturom repozitorija podataka:



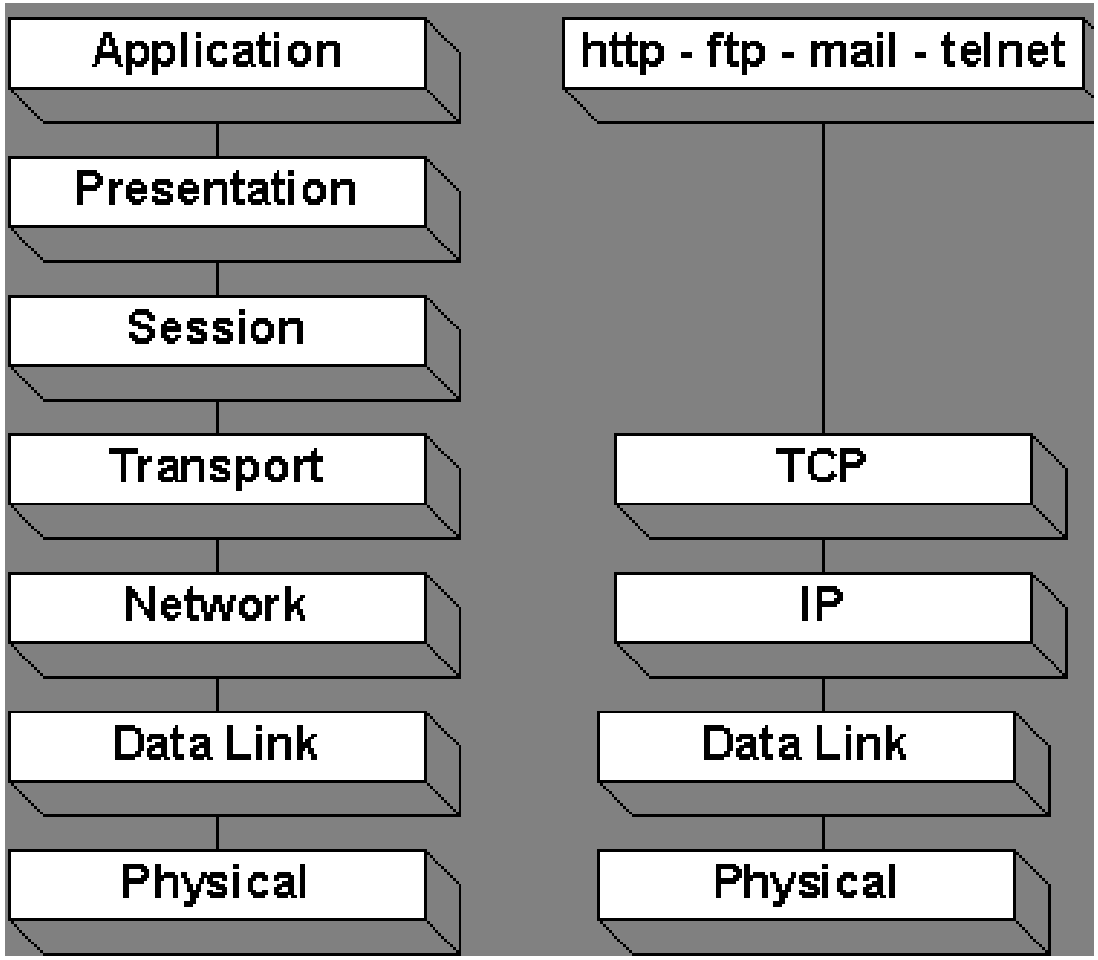
SLOJEVITA ARHITEKTURA

Slojevita arhitektura

- Primjenski program ili operacijski sustav se strukturira tako da se dekomponira u podskupove modula koji čine različite razine apstrakcije.
- Dekompozicija je na razini *statičkog izvornog koda*.
- Ako je dekompozicija na razini izvođenja (*engl. Run-time*) radi se o razinama ili naslagama (*engl. Tiers*).
- *Temeljni elementi:*
  - Komponente = slojevi
  - Konektori = protokoli koji definiraju interakciju između slojeva
- *Hijerarhijska organizacija*:
  - Samo susjedni slojevi komuniciraju
  - Svaki sloj daje uslugu sloju iznad i postaje klijent sloju ispod.
  - Svaki sloj skriva slojeve ispod.

Primjer

- **Tipični primjeri: OSI, TCP/IP**

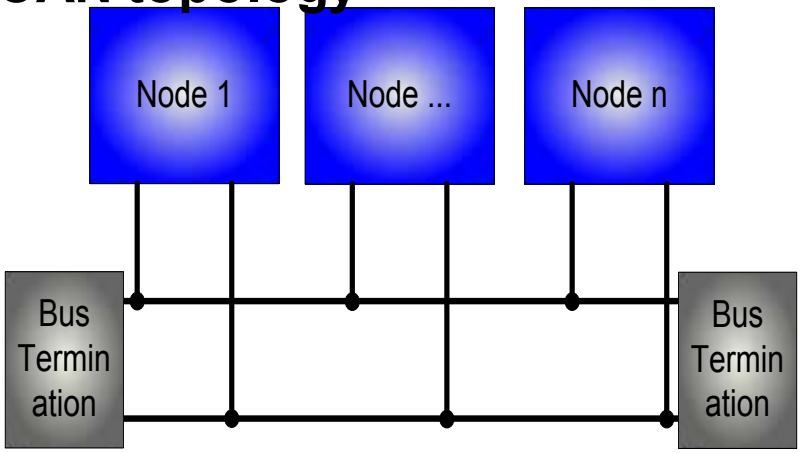


OSI slojeviti model je danas zastario.

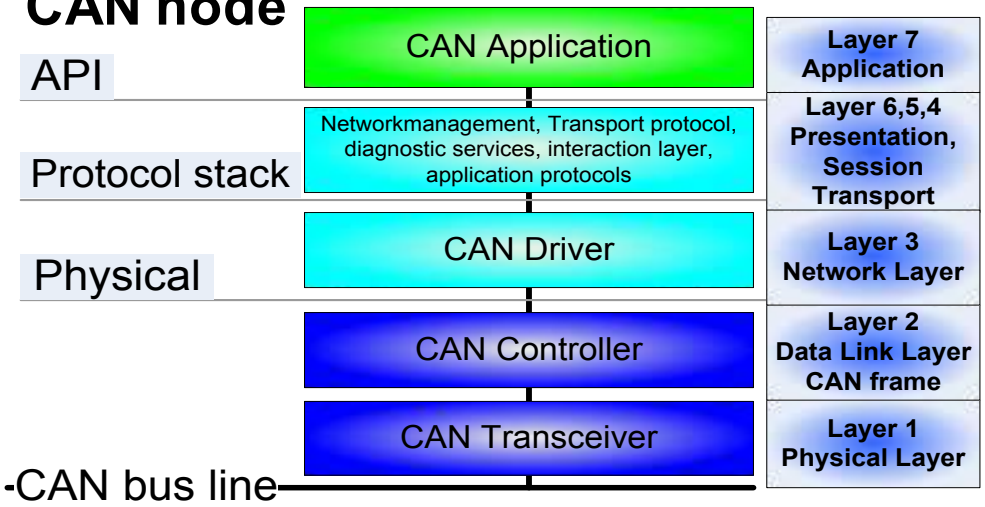
Zamijenjen je popularnim Internet Protocol Stack–om.

Primjer: CAN

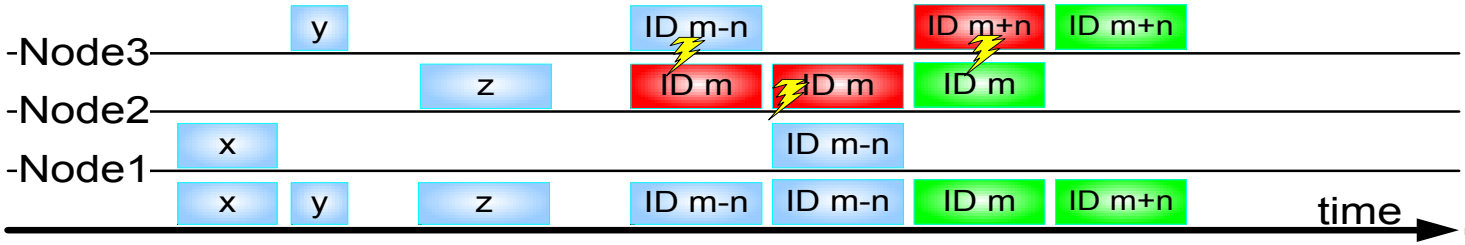
CAN topology



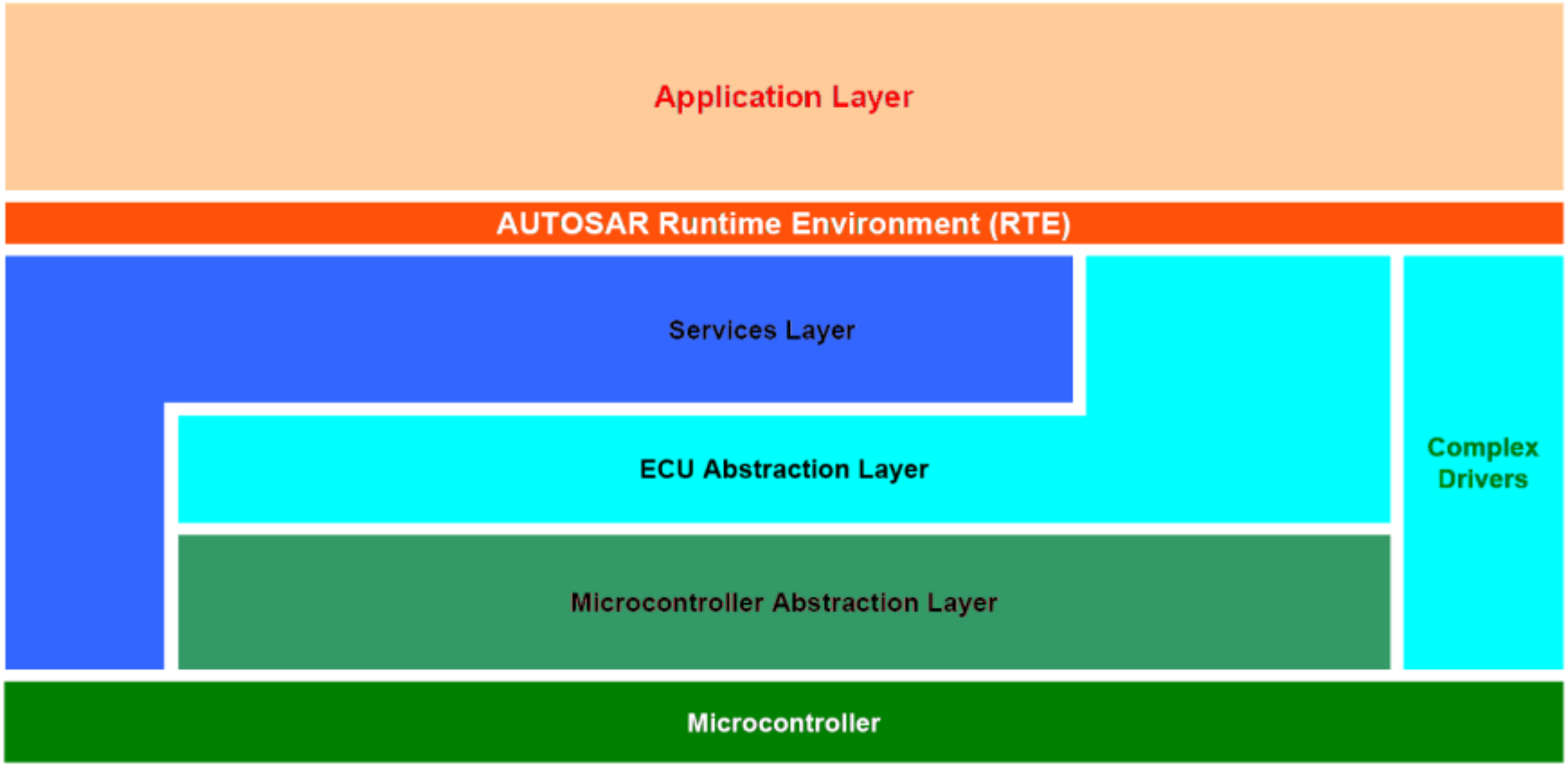
CAN node



CAN communication

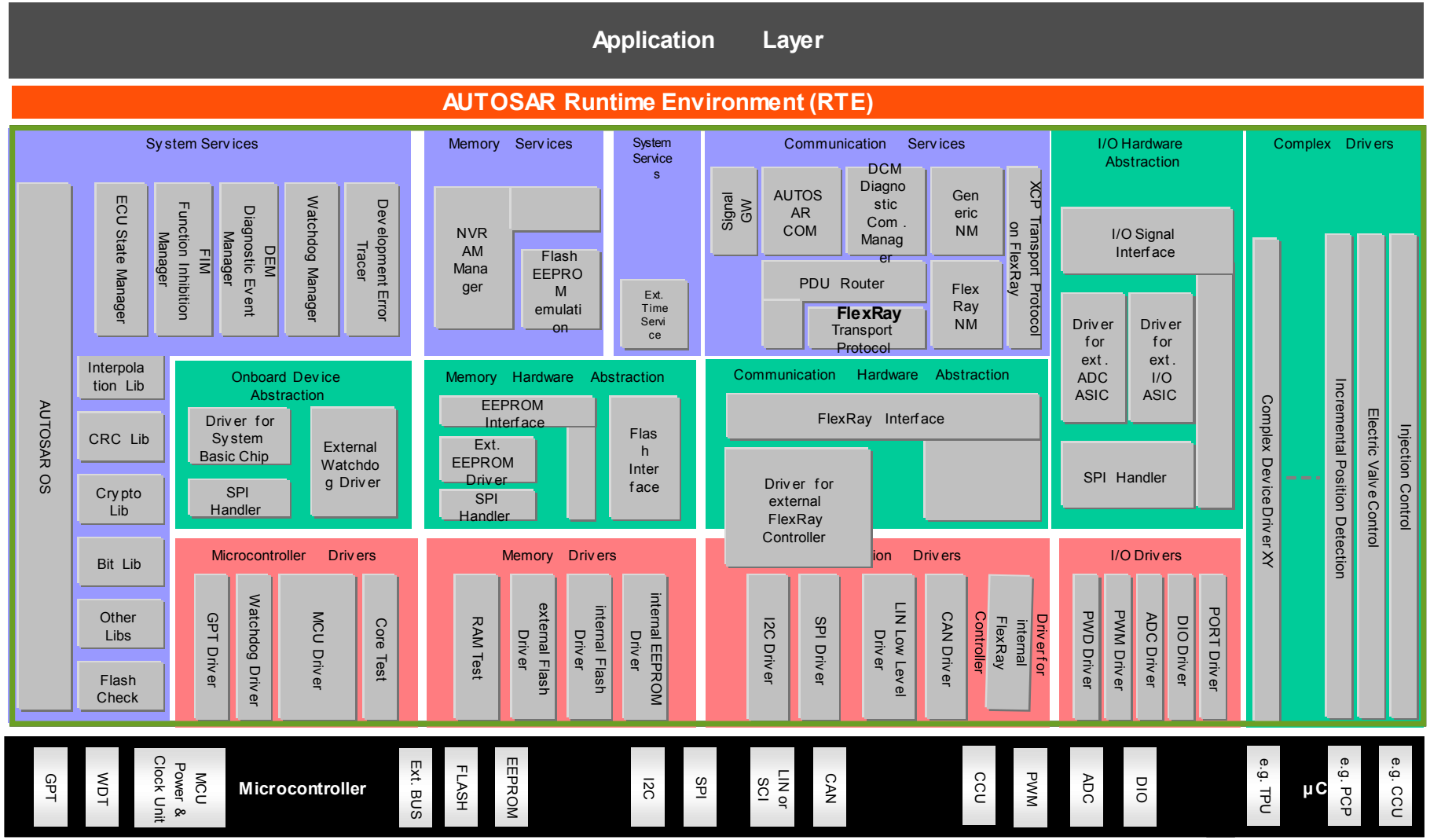


Primjer AUTOSAR



Primjer AUTOSAR

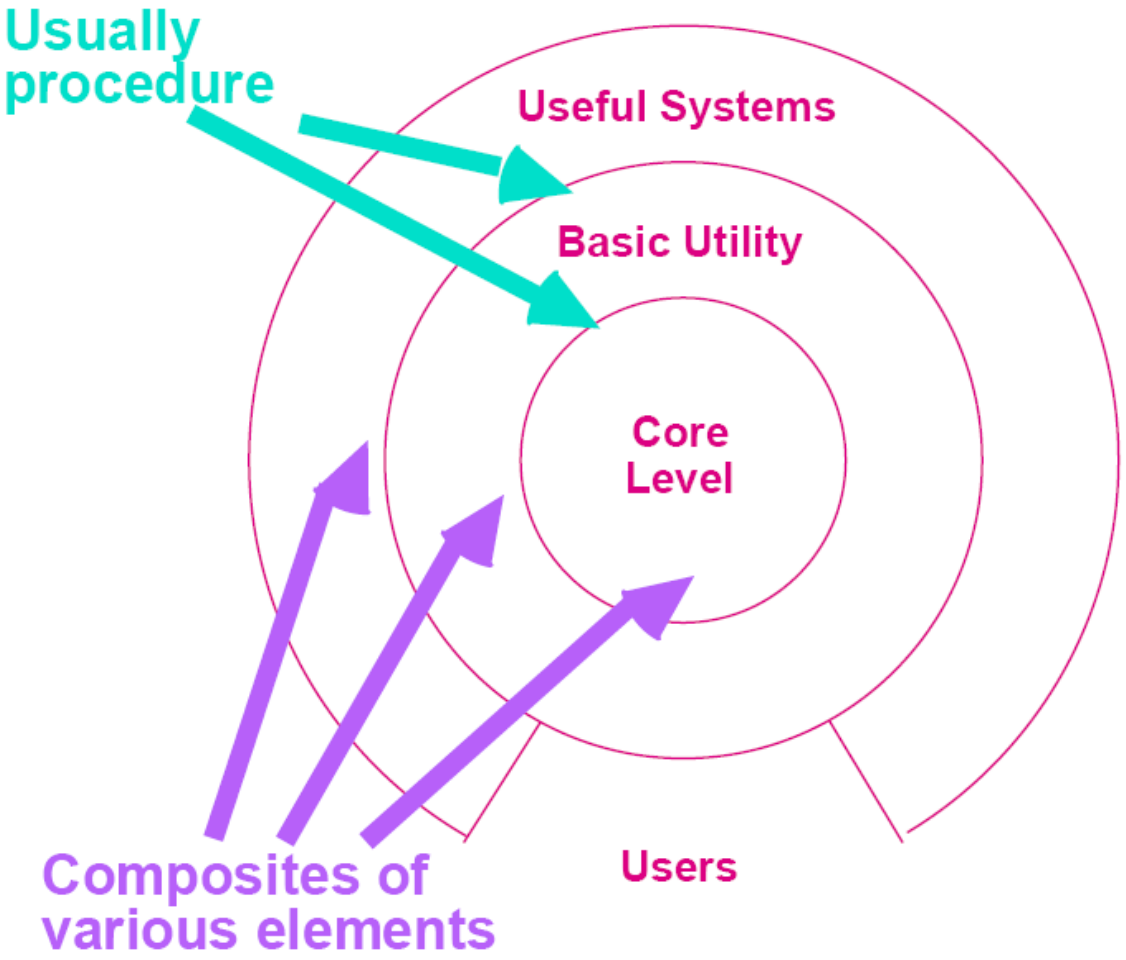
Automotive standard for Software Architecture



Izvor: SIEMENS VDO Automotive

Primjer 2

- Organizacija programskih cjelina u računalu



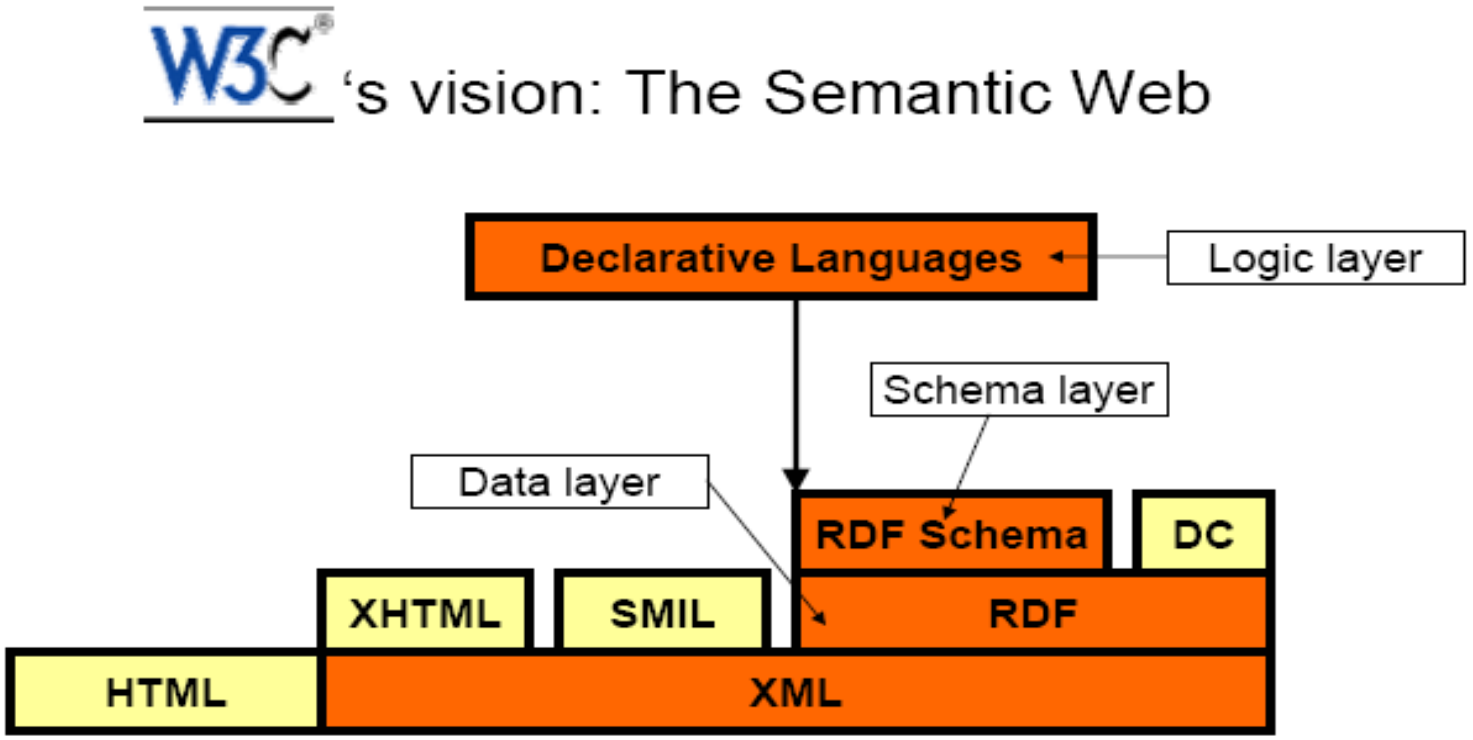
Primjeri:

UNIX (Linux) jezgra,

X Window sustav (Xlib, Xt, Motif), API

Primjer 3

- Predstavljanje znanja na Web-u
  - semantički Web
  - znanje predstavljeno kategorijama i hijerarhijom pojmova



Prednosti i nedostaci slojevite arhitekture

- Prednosti:
  - sloj djeluje kao koherentna cjelina.
  - vrlo jednostavna zamjena sloja novijom inačicom.
  - jednostavno održavanje jer svaki sloj ima dobro definiranu ulogu.
  - minimizirana je međuovisnost komponenata.
  - podupire oblikovanje programske potpore na visokoj apstraktnoj razini.
  - podupire ponovno korištenje (*engl, reuse*).
- Nedostaci:
  - smanjene performanse jer gornji slojevi samo indirektno dohvaćaju donje slojeve.
  - teško se pronalazi ispravna apstrakcija (posebice u sustavima s većim brojem slojeva).
  - svi sustavi se ne preslikavaju izravno u slojevitu arhitekturu (djelokrug programske jedinice može zahvaćati više slojeva).

VIRTUALNI STROJEVI

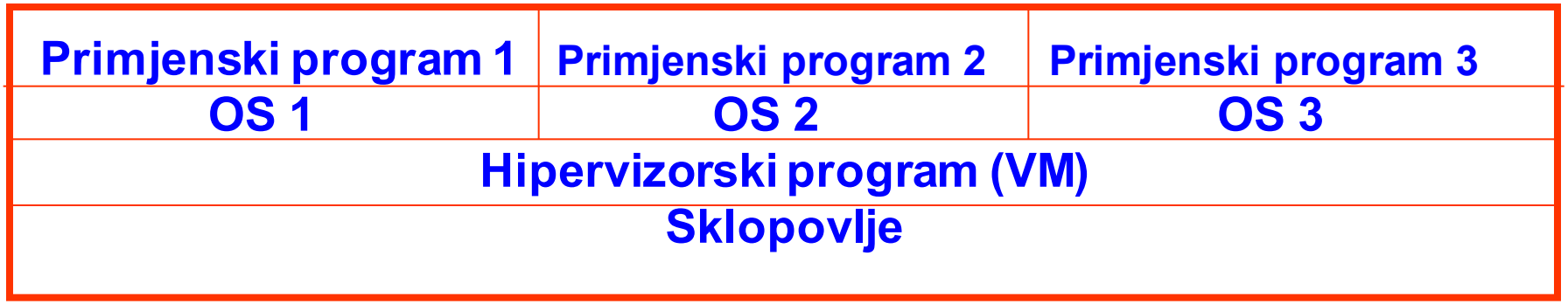
Virtualni strojevi

- Virtualni stroj (VM) je računalno okruženje čiji skup resursa i funkcionalnost je izgrađeno (kroz programski sloj) iznad nekog drugog programskog okruženja (apstrakcija računalnih resursa).
  - npr. Java primjenski program se izvodi u okviru Java virtualnog stroja (JVM), koji se pak izvodi u okviru nekog operacijskog sustava.
- Temeljna prednost je u *odvajanju primjenskog programa* od ostatka sustava
  - nedostatak je slabljenje performansi

- VM može simulirati funkcionalnost nepostojeće sklopovske platforme
  - pomaže u razvoju prog. potpore za nove naprave
- Konfiguracija više ne slijedi tradicijsku organizaciju (sklopovlje – OS – primjenski program) već jednu od nekoliko mogućnosti pod zajedničkim nazivom Virtualni stroj (VM).
- Može se promatrati na nivou sustava ili procesa
- Konfiguracije:
  - hipervizorski VM
  - udomaćen VM
  - primjenski VM
  - paralelni VM.

Hipervizorski VM

- *engl. Hypervisor*
- Virtualni stoj kao dodatni sloj odmah iznad sklopovlja. Na njega se oslanjaju jedan ili više OS gdje svaki smatra da je jedini OS u računalu
- Npr.: VMware ESX server, IBM z/VM

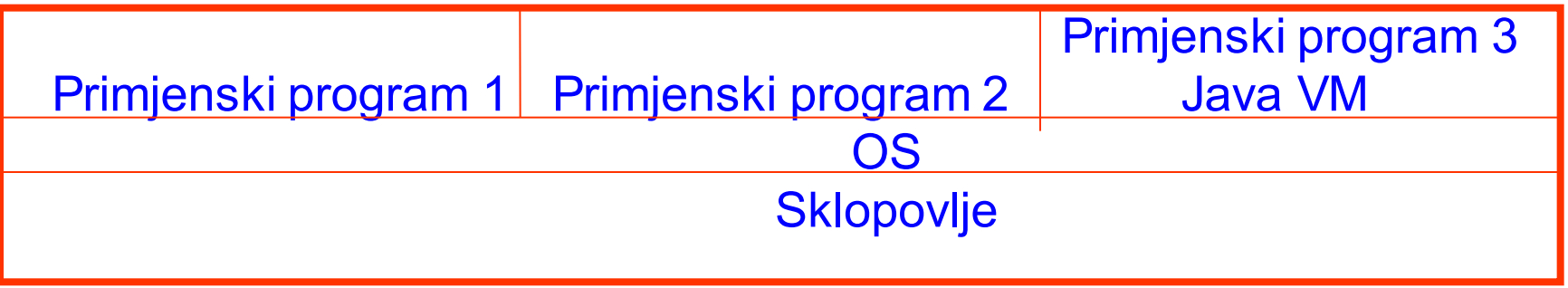


Udomaćen VM

- *engl. Hosted VM*
- Virtualni stoj kao i svaki drugi primjenski program izvodi se iznad OS-a.
  - poseban proces
  - domaćinski OS osigurava pristup sklopovlju
  - manje efikasno od Hipervizorskog VM, ali osnovna prednost odvajanja ostaje
  - npr. VMware GSX server, Microsoft Virtual PC, Linux user mode

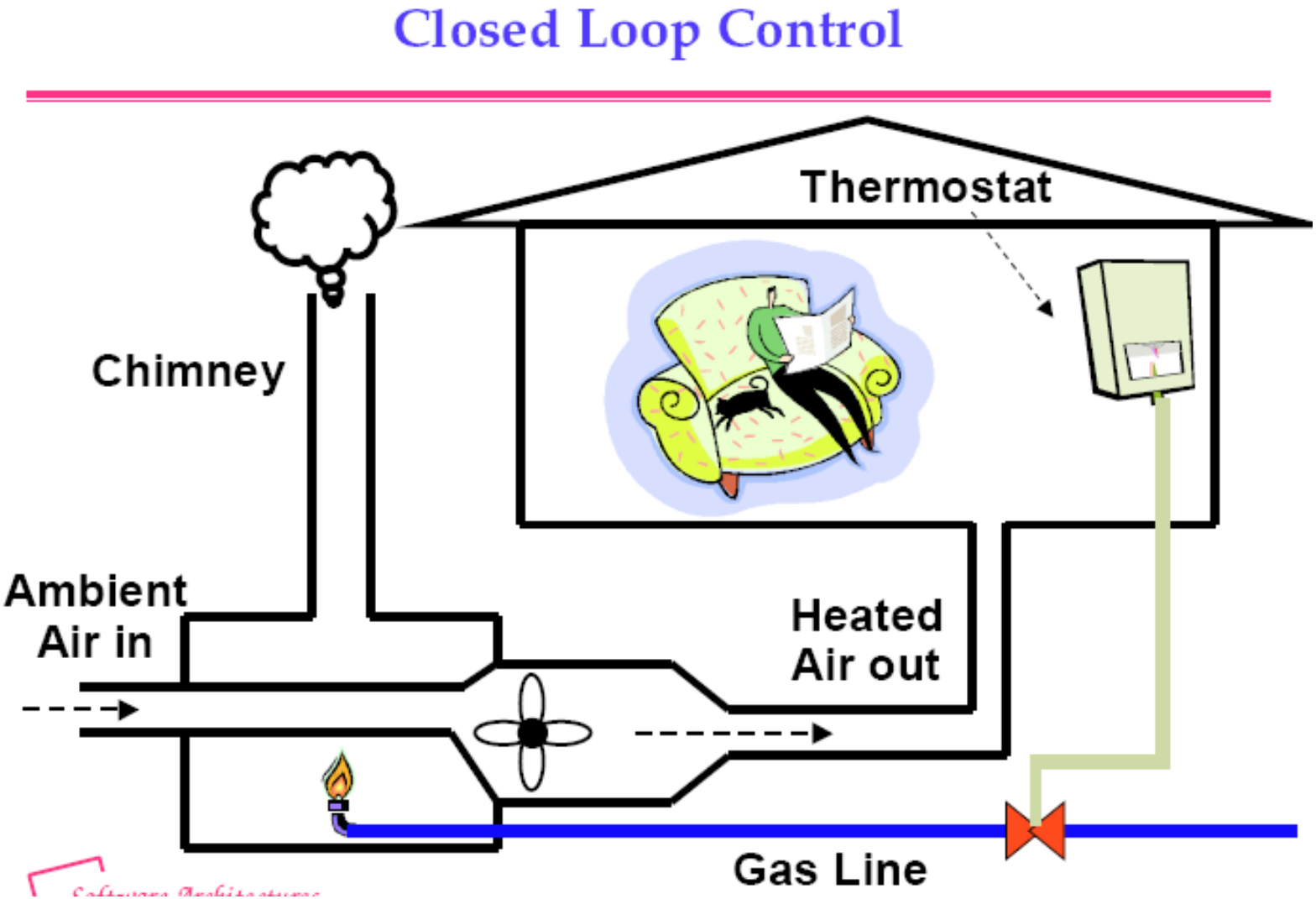
Virtualni stroj primjenske razine

- *engl. Process VM*
- Slično udomaćenu VM, ali na primjenskoj razini. Npr. Java VM je primjenski program (izvodi se na izvornom OS-u), a Java primjene se izvode na Java VM. Prednost: Java primjenski program izvodit će se bez prevođenja na svakom Java VM (koji je različit za svaki osnovni računalni sustav).



- Paralelni virtualni stroj
- Pretpostavlja se postojanje međusloja.
- Međusloj egzistira kao programski (zlo)duh (UNIX "demon") ili uslužni program, zajedno sa pozivima u skup rutina ("library calls"). Pozivi rutina su uključeni (prevoditeljem) u primjenski program. Međusloj omogućuje da su rutine raspodijeljene u mreži računala.

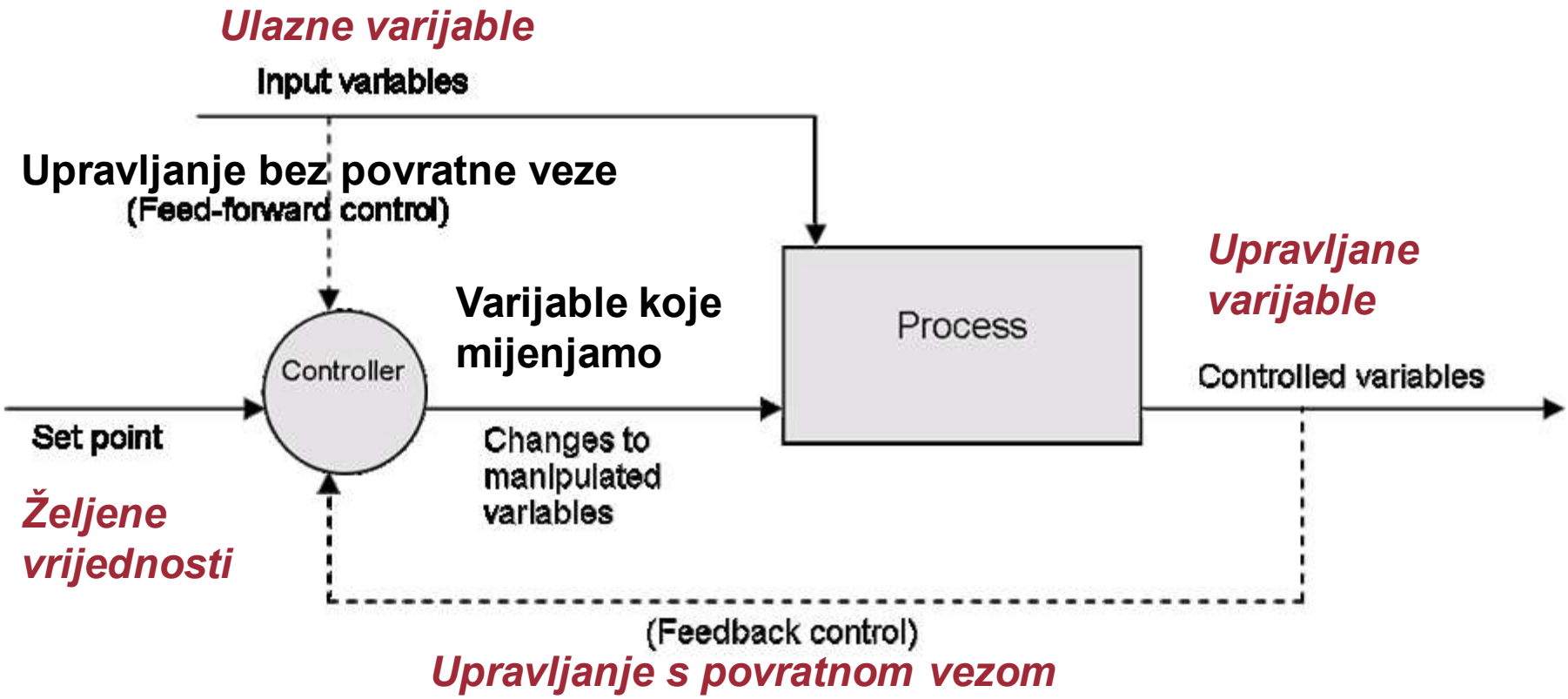
Arhitektura programa u upravljanju procesima



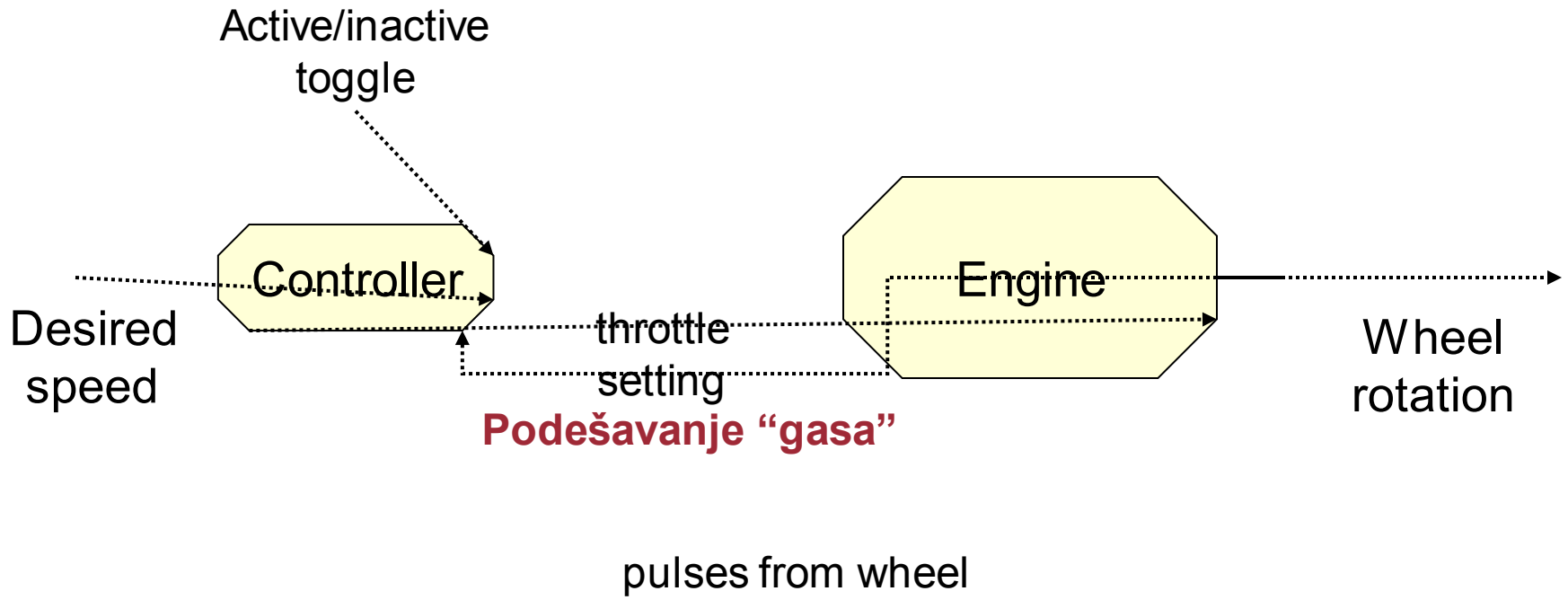
Automatsko upravljanje grijanjem u zatvorenoj petlji.

Sustav automatskog upravljanja

- Opći dijagram:



Primjer 2



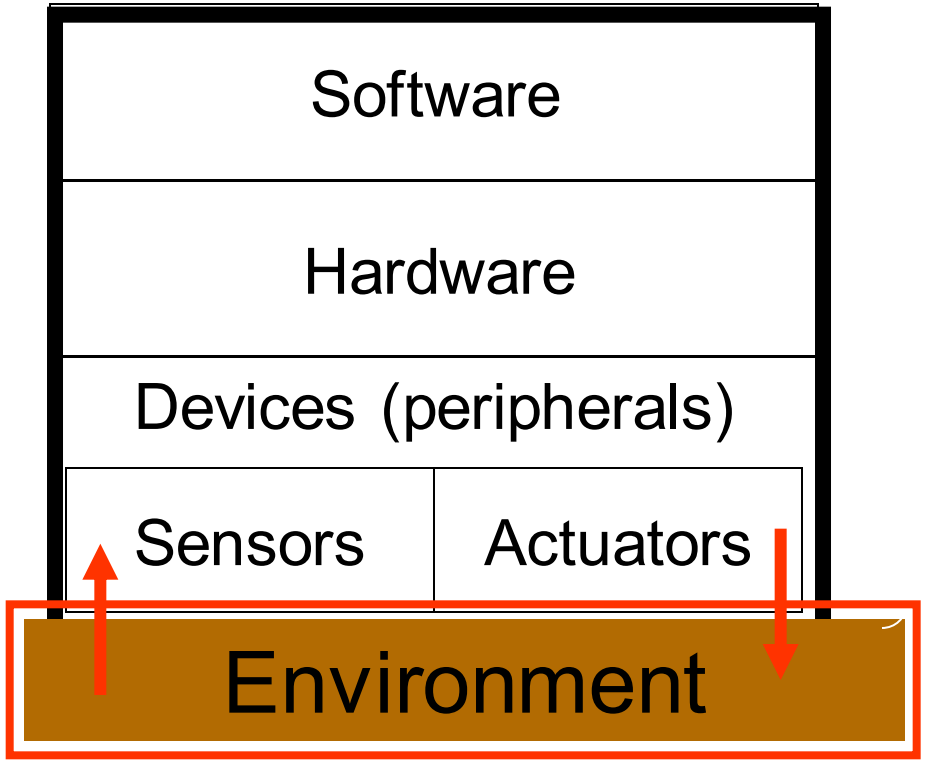
The **set point** consists of the **desired speed** and the **active/inactive toggle**. Desired speed can only be calculated when the system is active, and is based on the system on and increase/decrease speed inputs. Active/inactive status is calculated based on the engine on, system on, accelerator, brake, current speed, and resume inputs.

Arhitektura programa u upravljanju procesima

Tehnička shema fizikalnog procesa je razumljiva

- **poteškoće koje programske jedinice uporabiti.**

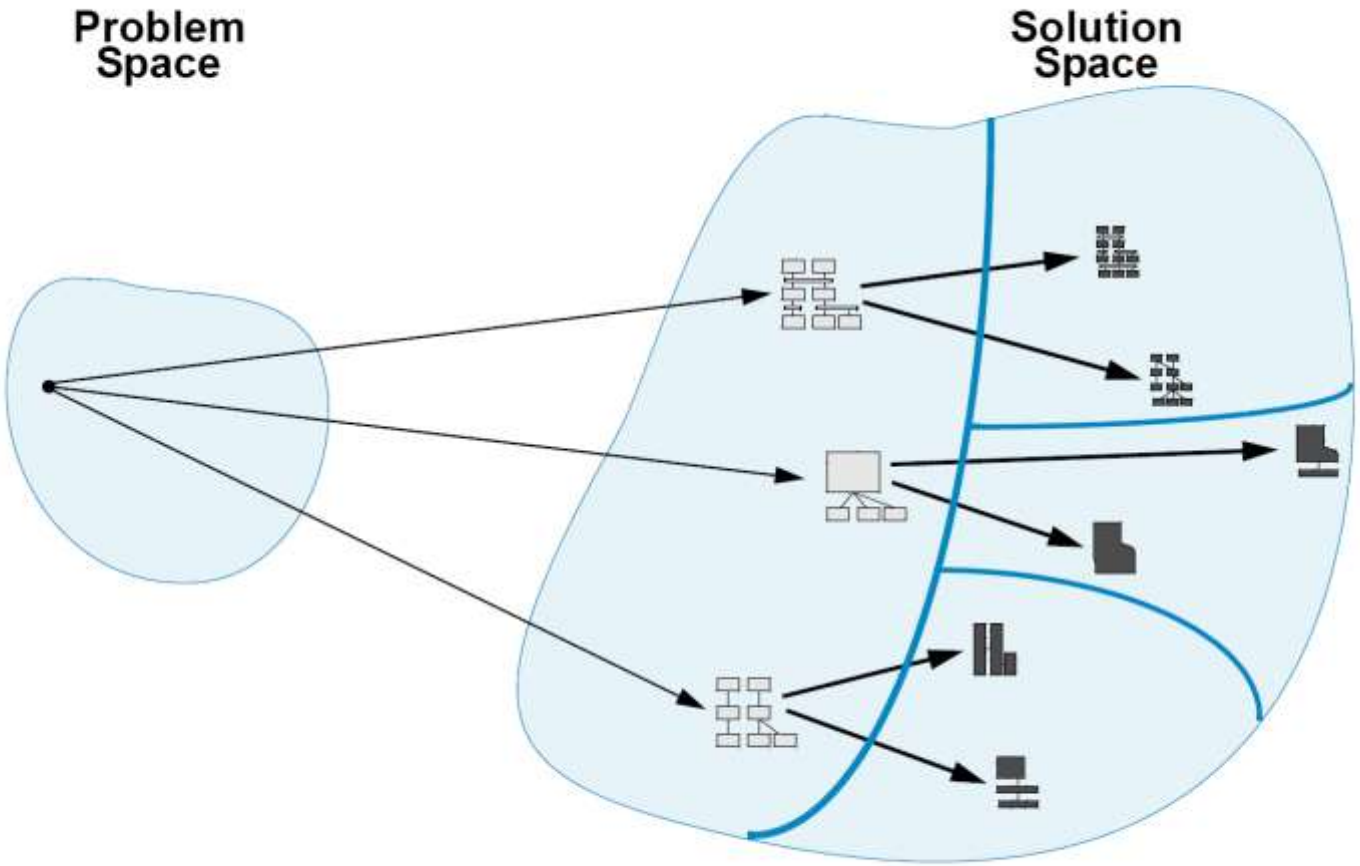
Potrebna općenitija analiza programskih i sklopovskih dijelova te okoline (*engl. HW + SW + environment*).



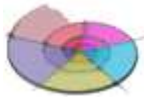
Arhitektura programa u upravljanju procesima

- Realizacija
- Budući da u ovoj arhitekturi pojedine programske komponente nisu specificirane, najčešće se koristi neka od temeljnih programskih arhitektura kao npr.:
  - Cjevovodi i filtri
  - Arhitektura zasnovana na događajima
  - …
  - Programske komponente se ugrađuju u fizički sustav (nužno je razmatranje ograničenja okoline te intenzivnu uporabu ekspertnog znanja iz domene primjene).

Transformacija programskog inženjerstva



Izvor: Taylor, Medvidovic, Dashofy: *Software Architecture: Foundations, Theory, and Practice*;



Oblikovanje programske podrške

- Danas “proizvođači” programske podrške surađuju i pregovaraju s kupcima
- Konstantan proces nadogradnje
  - nekad: instalacijska procedura + CD/medij
- Posebna pažnja na nefunkcijske zahtjeve
  - efikasnost, skalabilnost, heterogenost, pouzdanost, potrošnju

Što poslije OPP-a

- Oblikovni obrasci u programiranju
  - http://www.fer.hr/predmet/ooup
  - ...Razmatraju se klasifikacije obrazaca prema razini apstrakcije, svrsi i području primjene, zajedno s odgovarajućim predstavnicima. Pretpostavljena su osnovna znanja iz domene objektno orijentiranog programiranja stečena na uvodnim kolegijima.
- Razvoj primijenjene programske potpore
  - http://www.fer.hr/predmet/rppp
  - razrađuju se koncepti programskog inženjerstva i njihova primjena na razvoj programske potpore za krajnjeg korisnika. Obrađuju se elementi inženjerstva zahtjeva, oblikovanje i ugradnja programskih komponenti za različite tipove aplikacija, tehnike programiranja, dokumentiranje, uvođenje i održavanje aplikacija.
- .....

Diskusija