Osnove arhitekture sistema u Cloud-u

Fakultet tehničkih nauka
Univerzitet u Novom Sadu

Pregled

- **Uvod**
- **Istorijat razvoja distribuiranih sistema**
- **Osnovni pojmovi i definicije**
- **Softverske tehnologije i pristupi**
- **Arhitektura sistema u Cloud-u**
- **Otvoreni problemi i Zaključak**

Uvod

Sta je Cloud?

- **Je operativni sistem za velike Data Centre**
  - Upravlja resursima
  - Definiše životni ciklus aplikacije
- **Aplikacijama pruža viši nivo abstrakcije pri korišćenju compute, storage i network resursa**
- **Lako se skalira i omogućuje klijentima iluziju da imaju na raspolaganju beskonačne resurse**

Data Centri

- **Tradicionalni Data Centeri – Microsoft Dublin Datacenter**
  - 27,300 m^2
  - 22.2 Megawatt
- **Container-based Data Centers – Chicago Data Center**
  - 65,000 m^2
  - 60 Megawatt
  - Svaki kontajner sadrži 2500 servera

© www.datacenterknowledge.com

© www.datacenterknowledge.com

Vodeće kompanije

- **Komercijalni Cloud**
  - Microsoft Azure (Computing and Storage)
  - Amazon EC2 (Computing), S3 (Storage)
  - Google AppEngine (Computing), BigTable (Storage)
  - Yahoo, Salesforce, IBM, itd.
- **Open Source**
  - Eucalyptus Systems (University of California, Santa Barbara spin off)
  - Globus (open-source grid software)
  - itd.

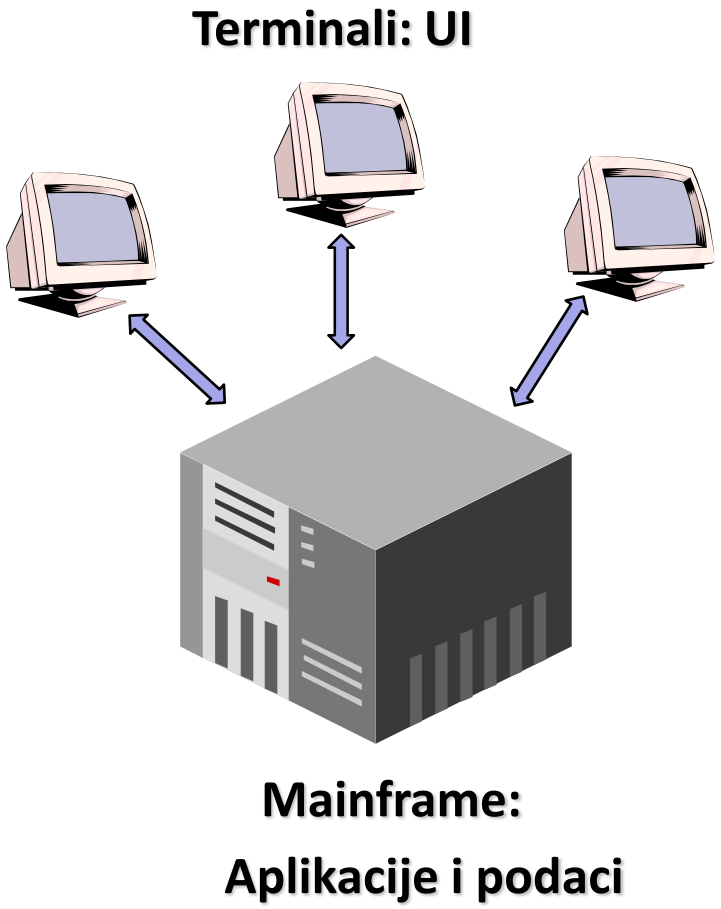
Istorijat

Enterprise aplikacije

- **Karakteristike velikih aplikacija**
  - Komleksna logika,
  - Velika količina kompleksnih podataka,
  - Specifični sigurnosni zahtevi,
  - Transakcioni problemi,
  - Veliki broj klijenata,
  - Podrška za različite platforme,
  - Heterogena razvojna okruženja: C, C++, Java, C#, COBOL, ...

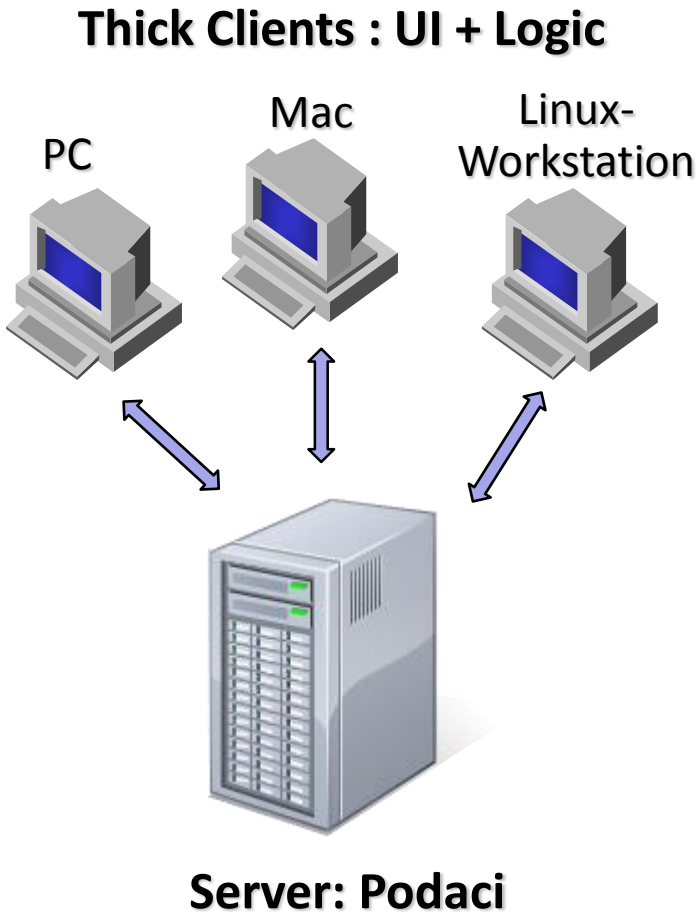
Mainframe Arhitektura

- **Pristup**
  - Mainframe sadrži sve resurse i logiku
  - Terminal sadrži samo UI
- **Prednosti**
  - Laka instalacija
- **Problemi i ograničenja**
  - Siromašan UI
  - Svaki klijent zauzima resurse na serveru →
  - Ograničena skalabilnost



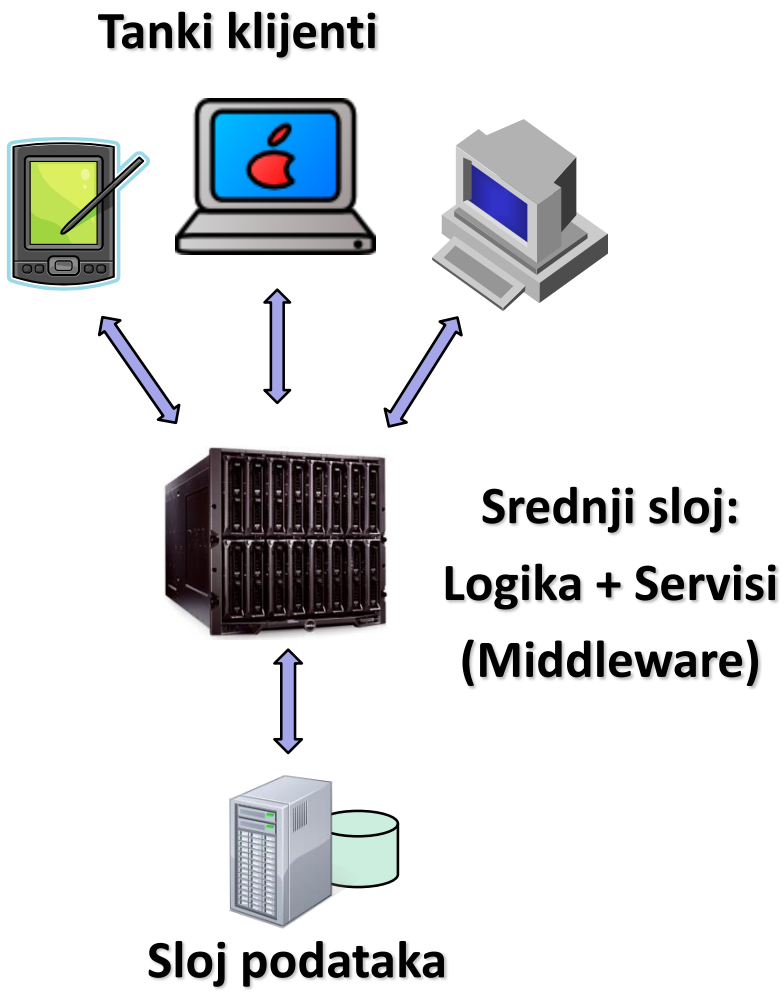
Client/Server Arhitektura

- **Pristup**
  - Povećan kapacitet PC-a
  - Logika je delimično prebačena na klijenta → thick client
  - Centralni server baze
- **Prednosti**
  - Bolji UI
- **Problemi i ograničenja**
  - Svaki klijent ima stalnu - stateful konekciju
  - Svaki klijent drži serverske resurse →
  - Ograničena skalabilnost



Middle-Tier Arhitektura

- **Pristup**
  - Logika je smeštena u srednji sloj
  - Klijenti nemaju stalnu vezu sa srednjim slojem → stateless
- **Prednosti**
  - Skalabilni srednji sloj
  - Srednji sloj daje servise (pooling, security, …)
  - Resursi su deljeni među klijentima → poboljšana skalabilnost
- **Problemi i ograničenja**
  - Novi programski model

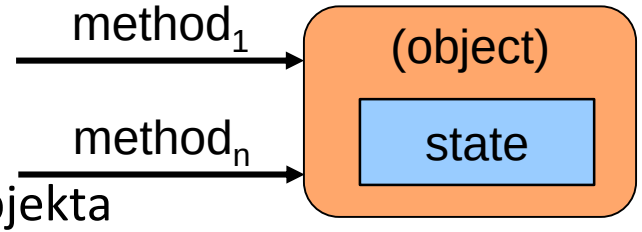


Šta možemo da naučimo iz Istorije?

- **Promena u arhitekturi hardvera dovodi do promene u sofverskoj arhitekturi:**
- **Novi zahtevi klijenata utiču i na sofversku i na hardversku arhitekturu:**
- **Arhitektura sa srednjim slojem ima mnogo sličnosti sa Cloud arhitekturom**

Arhitektonski izazovi za Cloud programiranje

Stateful i Stateless Komponente

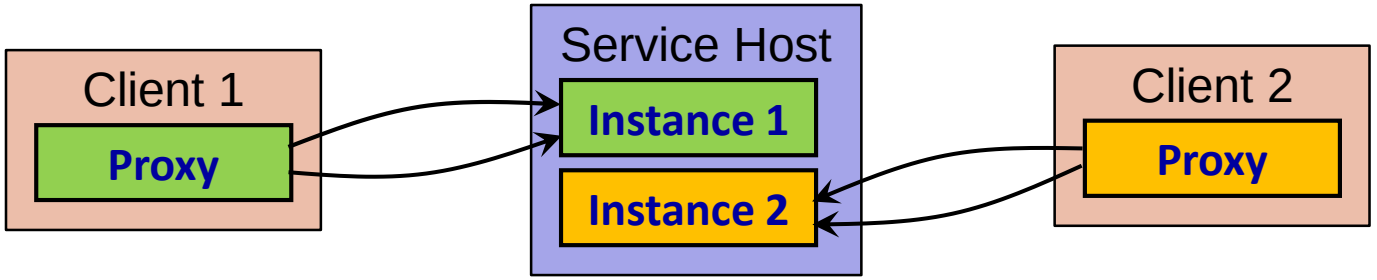


- **objekat = metode + podaci**
  - Metode izršavaju akcije nad objektima
  - Metode menjaju podatke (= stanje) objekta
  - OO aplikacije = skup objekata koji menjaju svoje stanje pozivom metoda
- **Zastarele RPC-bazirane tehnologije za distriburano programiranje**
  - CORBA
  - DCOM
  - RMI, .NET Remoting

  koriste OOP paradigmu za distibuirane aplikacije.
- **Programeri vole ovaj pristup zato što**
  - Navikli su na takav pristup
  - Lak je za implementaciju.

Stateful Servisi

- **Service host pravi novu instancu servera za klijenta koja ostaje aktivna sve dok traje sesija.**



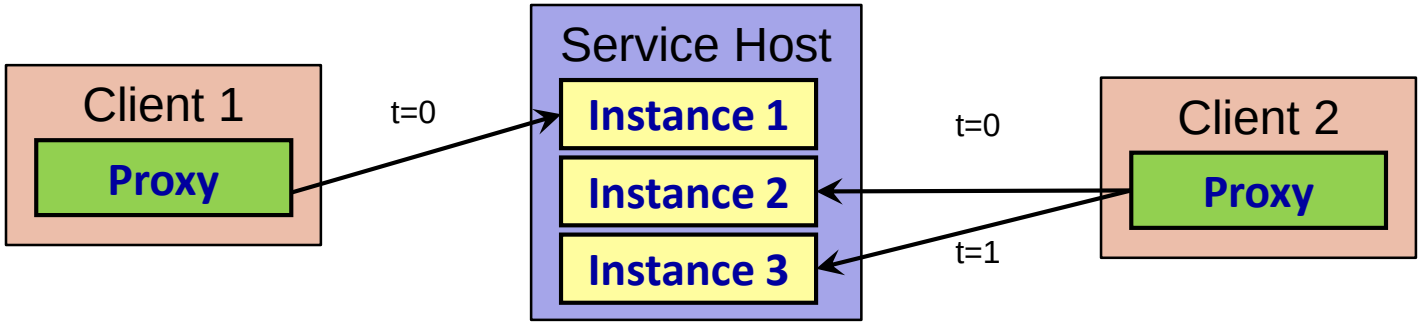
- **Uporediva sa client/server arhitekturom**
  - Servis je “ekstenzija” klijenta
- **Tehnologije**
  - EJB: Stateful Session Beans
  - WCF: Per-Session Services
  - CORBA
  - Java RMI

Stateful Servisi – Problemi

- **Svaka instanca servisa drži resurse na serveru**
  - Samo ograničeni broj klijenata može biti opslužen
- **Šta se dešava sa instancom servisa kada se klijent ugasi neočekivano?**
  - Distributed garbage collection, session timeout, …
- **Koje je bilo stanje servisa kada je konekcija prekinuta?**
  - Ponovno uspostavljanje konekcije može biti teško.
- **Problemi sa transakcijama**
  - Transakcije treba da budu vezane za pojedinu operaciju a ne za sesiju.
  - Kada se desi greška u transakciji stanje sesije je verovatno loše.

Stateless Servisi

- **Instanca servisa se pravi za svaki poziv.**



- **Ne postoji stalna konekcija između klijenta i servisa**
  - Resursi se mogu deliti među klijentima →
  - Povećana skalabilnost.
- **Tehnologije**
  - EJB: Stateless Session Beans
  - WCF: Per-Call Services

Dizajn Stateless Servisa

- **Stateless servis ne sme da čuva svoje stanje u poljima instance servisa.**
- **Service methoda je odgovorna za upravljanje stanjima**
  - Stanje se mora perzistirati na nekom eksternom medijumu:

```
public class MyService : IMyService {
  public BusinessMethod(long stateId) {
    LoadState(stateId);
    DoWork();
    SaveState(stateId);
  }
}
```

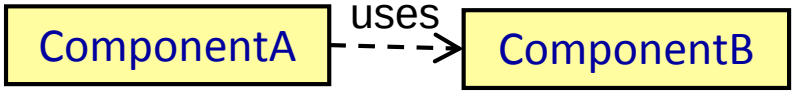
- **Implementacija stateless servisa može biti komplikovanija**
  - Treba koristiti ovaj pristup samo kada je skalabilnost bitna.

Separacija zaduženja

- **Svaka aplikacija se sastoji iz dva dela**
  - Application code implementira logiku aplikacije
  - Technology code kod potreban za rad sa tehnologijama koje aplikacija koristi
- **Loša je arhitektura u kojoj se application i technology kod mešaju**
  - U tom slučaju promena u tehlologiji utiču i na aplikacioni kod.
- **Tehnolgije za razdvajanje ove dve vrste koda**
  - Dependency Injection
  - Domain Objects
  - Layered Architecture
  - Aspect-oriented Programming (AOP)

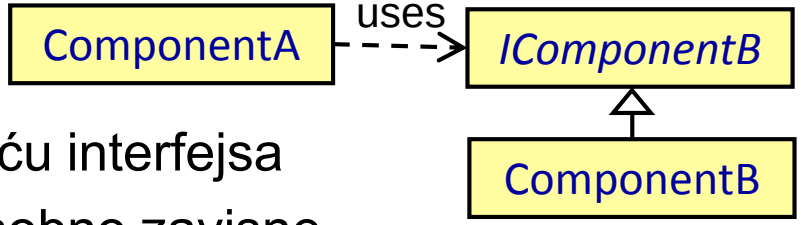
Povezanost Komponenti

- **Čvrsta povezanost**




```
public class ComponentA {
  public ComponentB ComponentB { get; set; }
  public ComponentA() { ComponentB = new ComponentB(); }
}
```

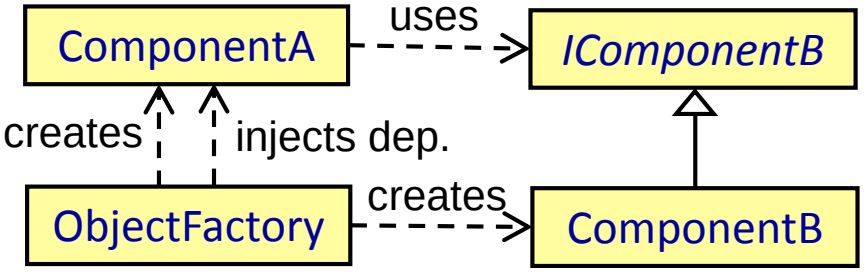
- **Manje čvrsta povezanost**
  - Smanjenje zavisnosti pomoću interfejsa
  - Komponente su ipak međusobno zavisne




```
public class ComponentA {
  public IComponentB ComponentB { get; set; }
  public ComponentA() {
    ComponentB = NamingContext.Lookup("name of ComponentB")
  }
}
```

Coupling kompoenenti. – Dependency Injection

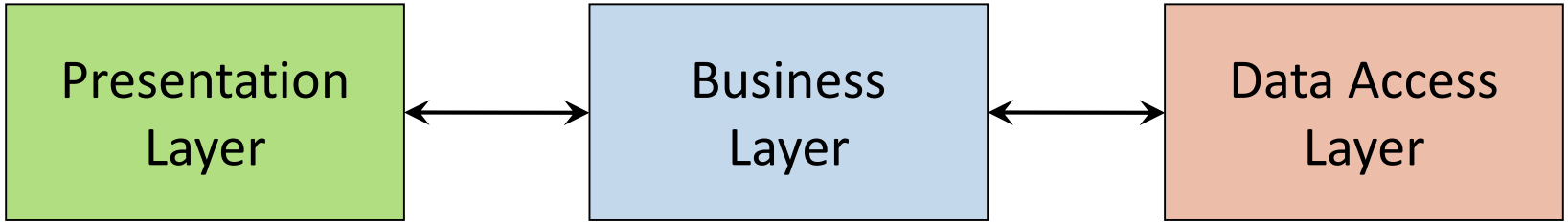
- **Slaba povezanost**
  - Komponente su povezane samo kroz interfejse
  - Dodela zavisnosti se radi “spolja”.




```
public class ComponentA {
  public IComponentB ComponentB { get; set; } // setter injection
  public ComponentA(IComponentB compB) {      // constructor injection
    ComponentB = compB;
  }
}
```

```
public class ObjectFactory {
  public GenerateComponents() {
    ComponentA compA = new ComponentA();
    ComponentB compB = new ComponentB();
    compA.ComponentB = compB;
  }
}
```

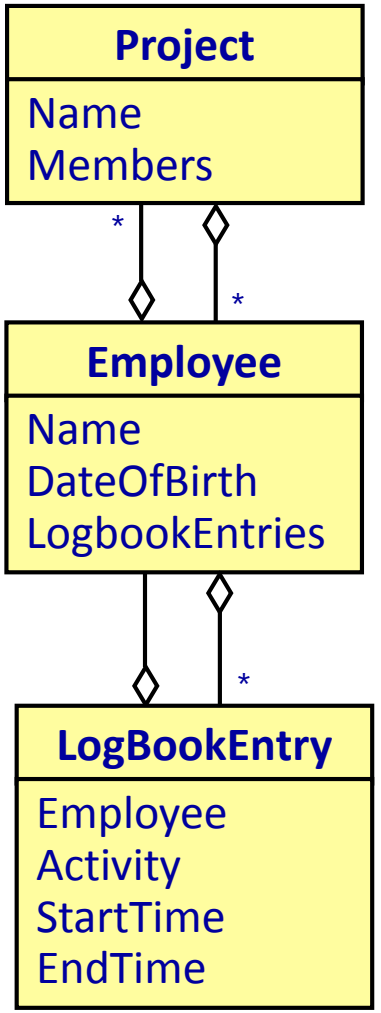
Slojevita Arhitektura



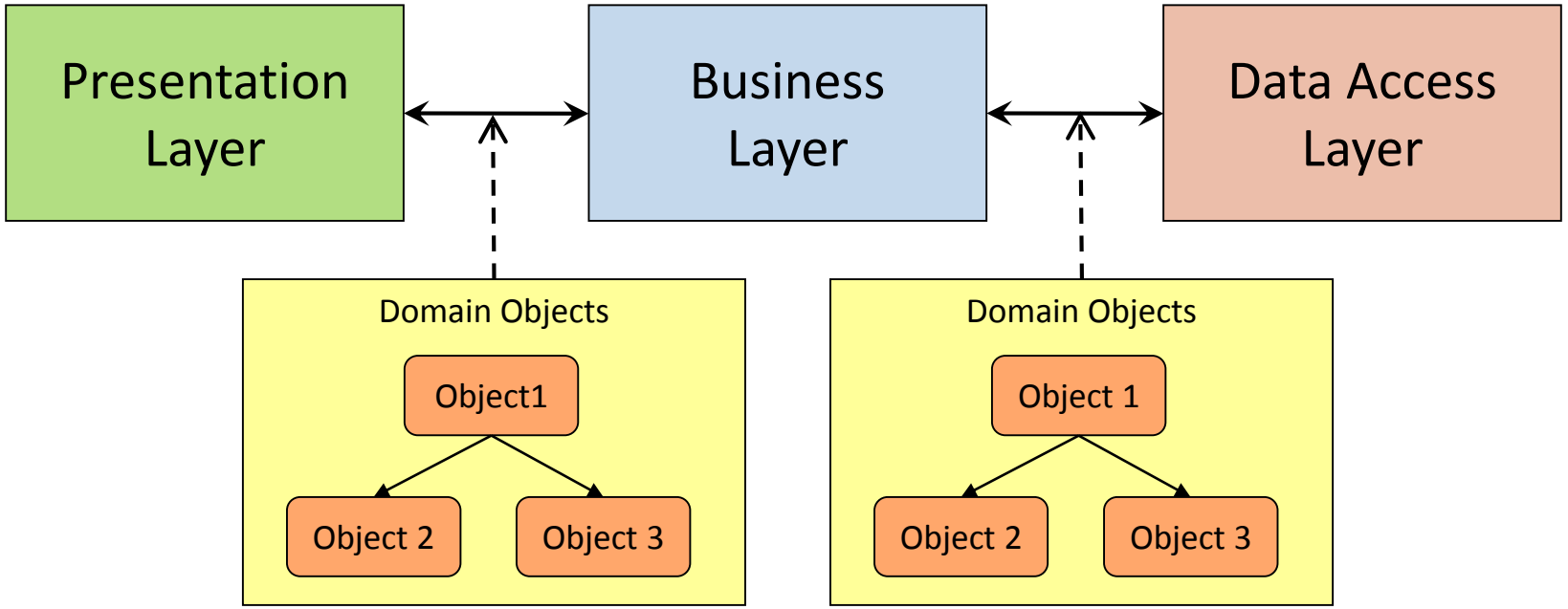
- **Aplikacija se deli u različite slojeve**
  - Svaki sloj može da pristupi samo susednim slojevima
  - Slojevi su slabo povezani
- **Sloj za korisnika**
  - UI: Thin Client, Rich Client, RIA
- **Sloj sa logikom aplikacije**
  - Implementira logiku aplikacije
  - Odgovoran je za transakcije, sigurnost, validacije
- **Sloj za pristup podacima (Persistence Layer)**
  - Sinhronizuje podatke sa bazom pomoću **Data Access Objects**

Domain Objekti

- **Domain Objects modeluju aplikacione podatke.**
- **Zahtevi**
  - Domain Objects ne treba da budu zavisni od specifične tehologoije.
  - Domain Objects moraju biti serijalizabilni.
  - Format serijalizacije mora biti interoperabilan
- **Tipovi Domain Object-a**
  - Change Sets (e.g. ADO.NET DataSets)
  - Simple Entities (POJOs/POCOs)
  - Self-Tracking Entities
  - Data Transfer Objects (DTOs)



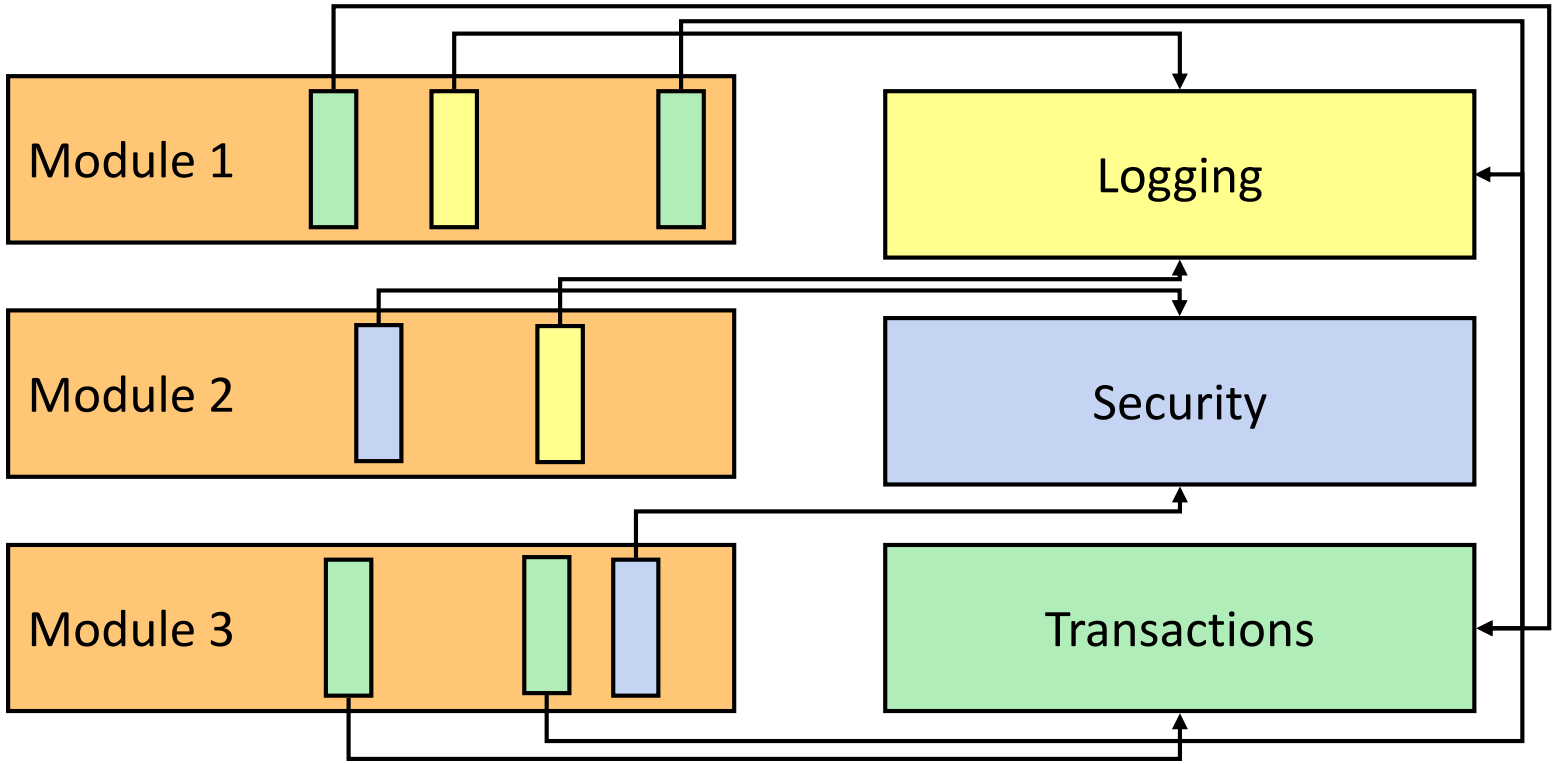
Uloga Domain Objekata



- Domain Objects služe za prenos aplikacionih podataka između slojeva.
- Pošto se koriste za komunikaciju između više slojeva moraju biti technology agnostic.

Aspekto-Orijentisano Programiranje (AOP)

- **Motivacija**
  - Neki tehnološki taskovi se ponavljaju u svim delovima aplikacije



AOP u Aplikacionim Okruženjima

- **Aspect je funkcionalnost koja se ponavlja na više mesta u aplikaciji**
  - Implementira ga **advice.**
  - Aspekti se izvršavaju na predefinisanim mestima u programu, zvanim **joinpoint**.
  - **Pointcut** je podskup joinpoint-a kada se advice izvrši.
- **U većini okruženja za razvoj postoje ugrađeni aspekti za**
  - Upravljanje resursima i instancama
  - Upravljanje transakcijama
  - Upravljanje pravima pristupa
- **Pointcut-ovi se najčešće definišu kao atributi ili anotacije.**
- **Osnovni cilje je separation of concerns.**

AOP Primer

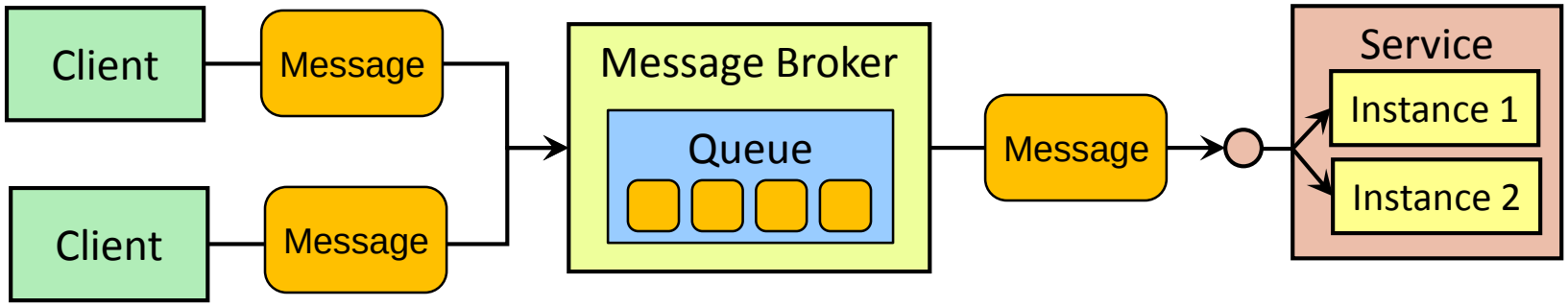
- **EJB**

```
@Stateless
public class WorklogBean implements WorklogRemote {
  @PersistenceContext
  private EntityManager manager;
  @TransactionAttribute(TransactionAttributeType.REQUIRED)
  @RolesAllowed({"Managers"})
  public Long saveEmployee(Employee empl) { … }
}
```

- **WCF**

```
[ServiceBehaviour(InstanceContextMode=InstanceContextMode.PerCall]
public class WorklogService : IWorklogFacade {
  [OperationBehavior(TransactionScopeRequired=true)]
  [PrincipalPermission(SecurityAction.Demand,
                       Role=(@"<domain>\Managers")]
  public int SaveEmployee(Employee empl) { … }
}
```

Asinhrona Obrada Poruka



- **Poruke se čuvaju u perzistentnom redu (queue) i obrađuju se asinhrono.**
- **Prednosti**
  - Klijenti nisu blokirani dok se poruka obrađuje.
  - Klijent i server ne moraju da budu aktivni istovremeno.
  - Moguće je distribucija posla na serveru (load balancing).
- **Programski model se razlikuje od sinhronog pristupa**
  - Klijent ne dobija povratnu vrednost u slučaju greške.

Otvoreni problemi i Zaključak

Otvoreni problemi

- **Softverski izazovi**
  - Migriranje postojećih aplikacija na Cloud
  - Programsko okruženje za velike distribuirane sisteme
  - Zaključavanje podataka
  - Skalabilne strukture podataka
  - Predvidljivost rada sistema i promenjive performanse
- **Ekonomski izazovi**
  - Odabir pravog okruženja i resursa
  - Projekovanje ukupne cene koštanja sistema
  - Biznis model se takodje menja
- **Zakonodavstvo**
  - Problemi sa tajnosti podataka

Zaključak

- **Razmotrene su Osnove arhitekture sistema u Cloud-u**
- **Postoji potreba za poznavanje**
  - Algoritama i programskih jezika
  - Softverske arhitekture
  - Koncepata prava pristupa
  - Web arhitektura i okruženja
- **Za izbor prave platfome potrebno je voditi računa o**
  - Nivou usluga koje se nude (IaaS, PaaS, SaaS)
  - Pouzdanosti sistema i sigurnosti pristupa
  - Postojanju komponenti za razvoj sistema: za proračune, čuvanje podataka i komunikaciju