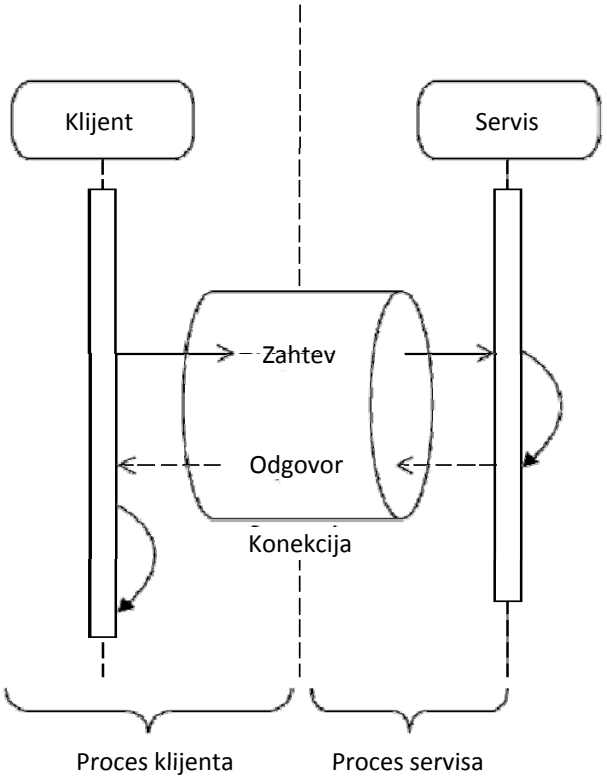
INTERAKCIJE SOA
Informacioni sistemi 2

dr Miloš CVETANOVIĆ
dr Vladimir BLAGOJEVIĆ

Interkacija sa servisom

Koji je najednostavniji način da servis obradi zahtev i vrati rezultat?



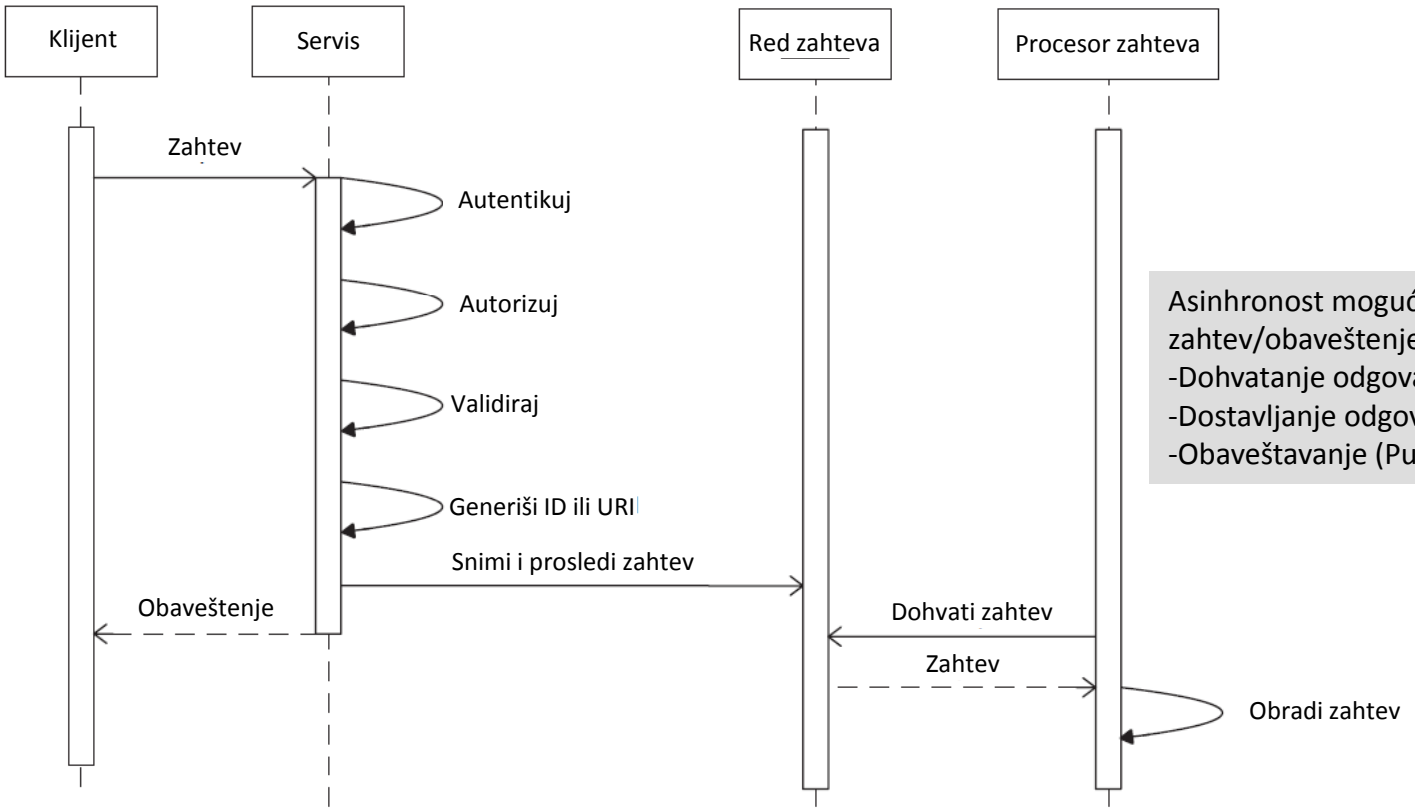
Vremenska vezanost između klijenta i servisa je **veoma velika** :
-Servisi mora da bude operativan u trenutku zahteva
-Mreža mora biti funkcionalna sve vreme trajanja interakcije
-Veliki broj konkuretnih zahteva mogu preopteretiti servis

Asinhronost moguća interekacijom po principu zahtev/obaveštenje (Request/Acknowledge) i to:
-Dohvatanje odgovara (Poll)
-Dostavljanje odgovora (Callback)

Interkacija po pricipu zahtev/odgovor → sinhrona komunikacija
Posrednici u komunikaciji → Keširanje odgovora (Proxies)
Posrednici u komunikaciji → Provera postojanja sertifikata u zahtevu (Firewall) npr. X.509 sertifikat
Interkacija zahtev/odgovor → ne ograničava API stil (ne samo RPC, već i stilovi inspirisani porukama i resursima)

Interkacija sa servisom

Kako se servis može zaštiti od vršnih opterećenja i obezbediti da zahtev bude obrađen čak iako neki od podsistema nisu trenutno dostupni?

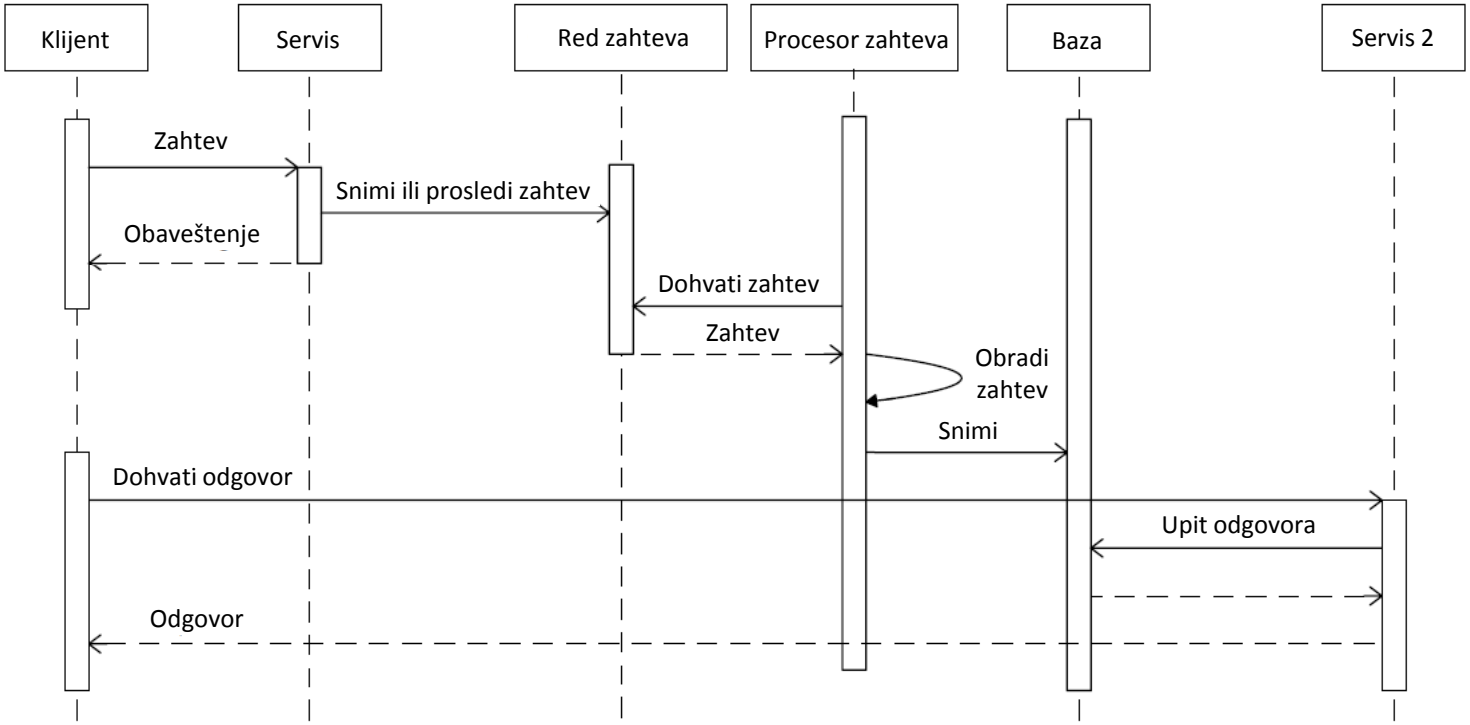


Asinhronost moguća interekacijom po principu zahtev/obaveštenje (Request/Acknowledge) i to:
-Dohvatanje odgovara (Poll)
-Dostavljanje odgovora (Callback)
-Obaveštavanje (Publish/Subrscribe)

Interkacija po pricipu zahtev/obaveštenje (npr. klijent obaveštava sistem, ili pokreće poslovnu aktivnost)
Smanjena vremenska zavisnost
Klijent se ne blokira
Autentikacija upotrebom presretanja zahteva (Service intereceptor) → Neuspeh vraća NAck ili SOAP Fault
Zahtev primljen, ali da li je obrađen (npr. HTTP status 202 – request received but not processed)

Interkacija sa servisom

Kako se servis može zaštiti od vršnih opterećenja i obezbediti da zahtev bude obrađen čak iako neki od podsistema nisu trenutno dostupni?



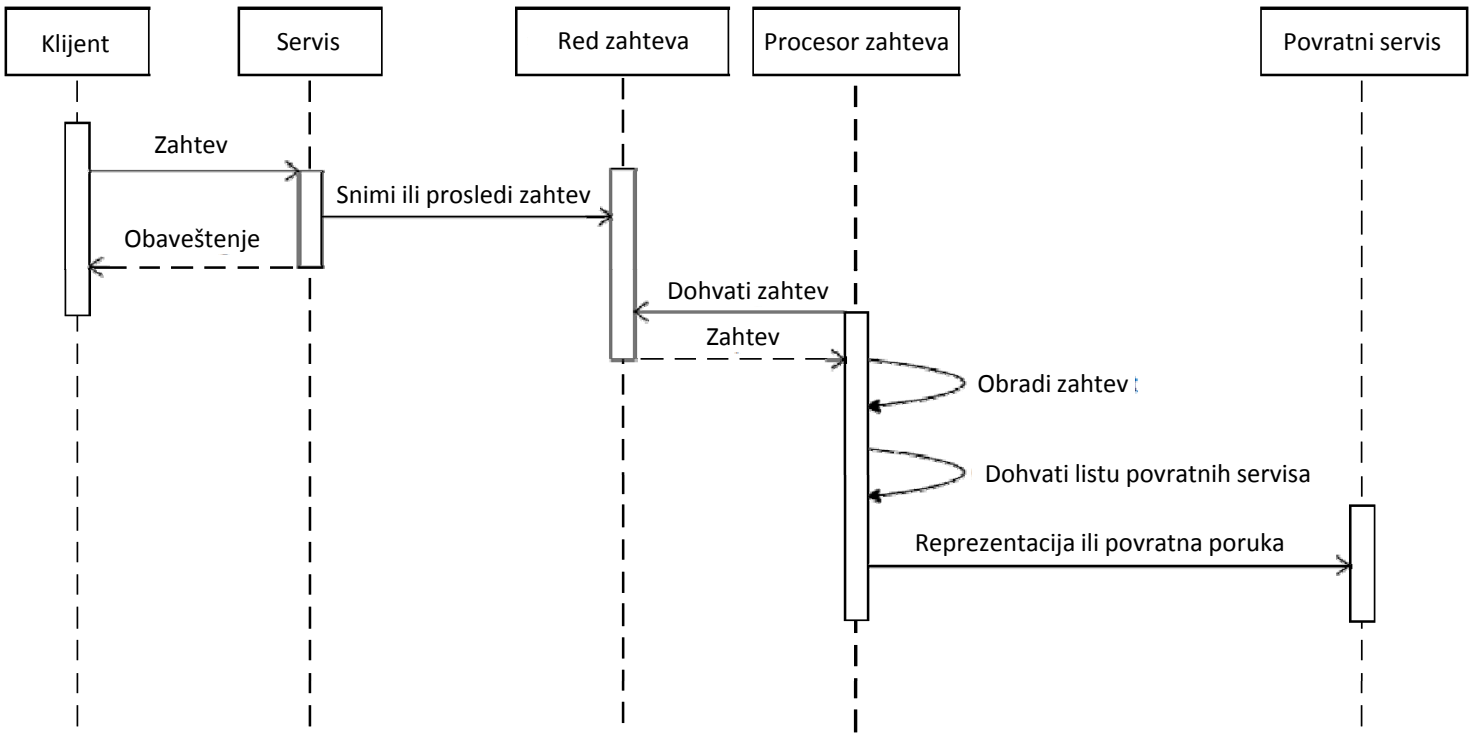
Interkacija po pricipu zahtev/obaveštenje/dohvati (Request/Acknowledge/Poll)
Suviše često dohvatanje statusa → veliko opterećenje servera
Suviše retko dohvatanje statusa → velika razlika između trenutka spremnosti rezultata i trenutka dohvatanja rezultata

Primer 30. REST – Komunikacija po principu zahtev/obaveštenje/dohvati

```
@Path("/orders")
public class OrdersResourceController {
      private static String BaseURL = "http://primer.rs/";
      public OrdersResourceController() {;}
     @POST
     @Consumes ("application/xml")
     @Produces ("text/plain")
      public String PlaceOrder(Order order) {
           String requestId = System.currentTimeMillis().toString() +
                                                java.util.UUID.randomUUID().toString();
           order.setIdentifier(requestId);
           Fulfillment.Submit(order); // npr. ubacivanje u red zahteva
           return BaseURL + requestId;
       }
     @GET
     @Path("/{requestId}")
     @Produces("application/xml")
      public Order GetOrder(@PathParam("requestId“) String requestId){
           return Fulfillment. getOrderStatus(requestId);
      }
}
```

Interkacija sa servisom

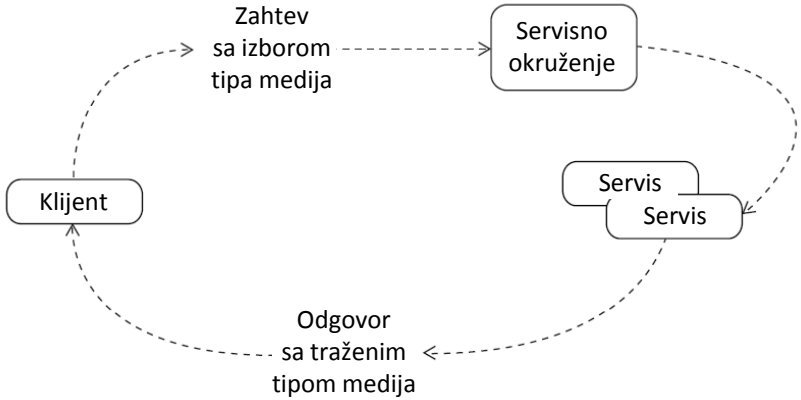
Kako se servis može zaštiti od vršnih opterećenja i obezbediti da zahtev bude obrađen čak iako neki od podsistema nisu trenutno dostupni?



Interkacija po pricipu zahtev/obaveštenje/dostavi (Request/Acknowledge/Relay)
Povratni servis (Callback servis) → zahtev za javno dostupnim servisom na klijentskoj strani komunikacije
Opterećenje servisne strane raste sa povećanjem broja povratnih servisa kojima treba dostaviti odgovor
Servisna strana možda mora da obezbedi idempotentnost pri pozivu povratnog servisa

Interkacija sa servisom

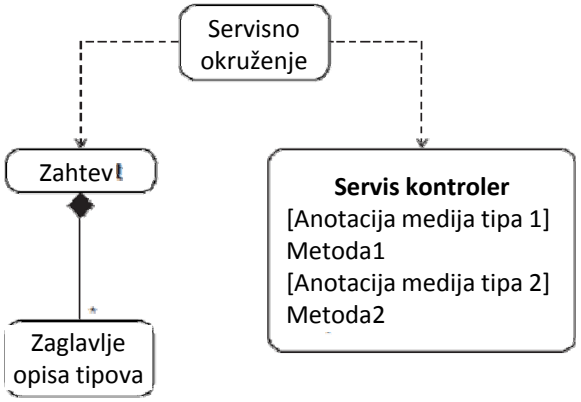
Kako servis može da obezbedi više reprezentacija jednog istog logičkog resursa uz minimizaciju različitih URI do tog resursa?



Veća vezanost klijenta nije poželjna, na primer:
http://primer.rs/proizvod/123/json
http:// primer.rs/proizvod/123.html
http:// primer.rs/proizvod/123.pdf
http:// primer.rs/proizvod/123?fmt=json

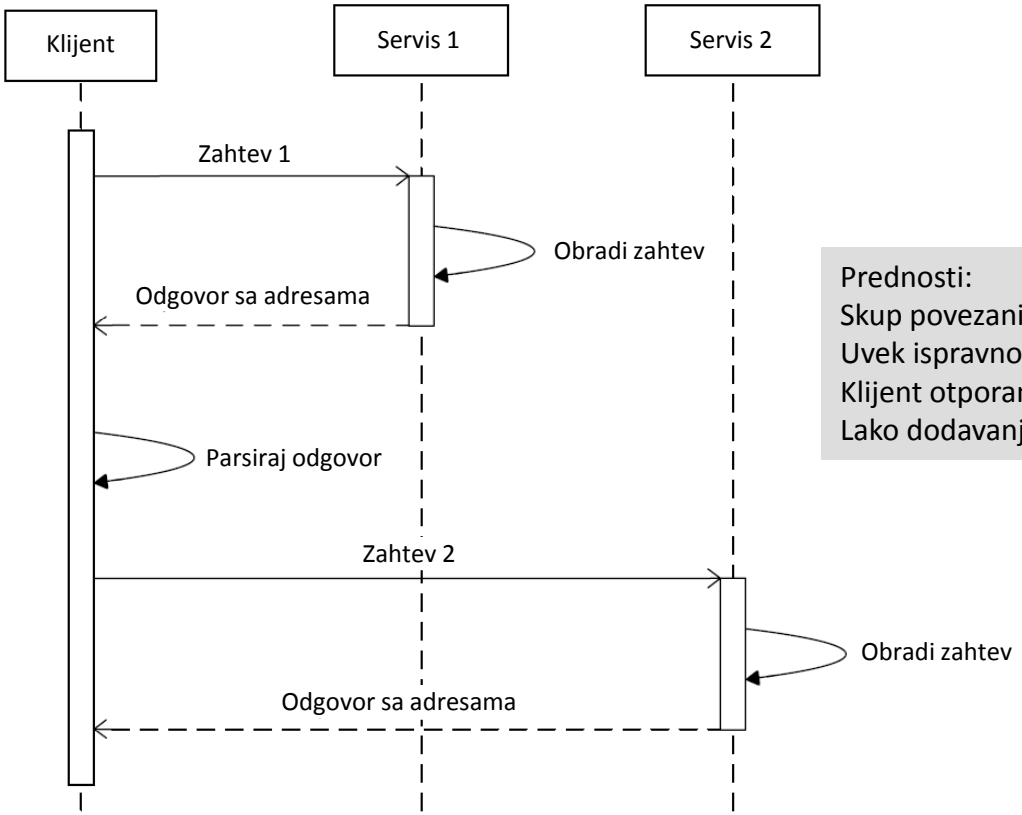
Interkacija uz pregovaranje tipa medija
Upotreba standardnog HTTP protokola za pregovaranje tipa medija
Pregovaranje na servisnoj strani upotrebom servis kontrolera (Service Controller)
Pregovaranje na klijentskoj strani povećava komunikaciju (zahtev za adresama, zahtev za rezultatom)

```
GET http://primer.rs/proizvod
Accept: application/json;q=1.0,
        text/*;q=0.9
```



Interkacija sa servisom

Nakon opsluženog zahteva, kako klijent može da zna koje servise dalje može da poziva, a da je ujedno bezbedan u slučaju promene adresa servisa?

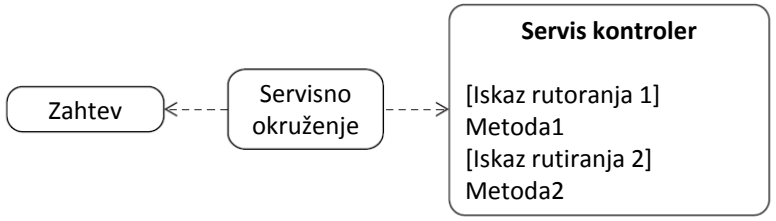


Prednosti:
Skup povezanih servisa može da zavisi od konteksta zahteva (“wizard like”)
Uvek ispravno formiranje adrese (klijent ne brine o tome)
Klijent otporan na promene u formatu adresa kao i na promene adresa
Lako dodavanje novih i uklanjanje zastarelih servisa

Potencijalni problemi:
Parsiranje odgovora
Sigurnost od MIMT (Man-in-the-middle)
Generisanje linkova ka servisima

Interkacija sa povezivanjem servisa (Linked Service)
Alternativa je registar servisa (Service Registry) → kompleksnost, dodatna tačka otkaza, vezanost za proizvođača

Interkacija sa servisom
Kako odrediti koji servis treba da se izvrši
bez parsiranja zahteva i održavanja kompleksne logike rutiranja?



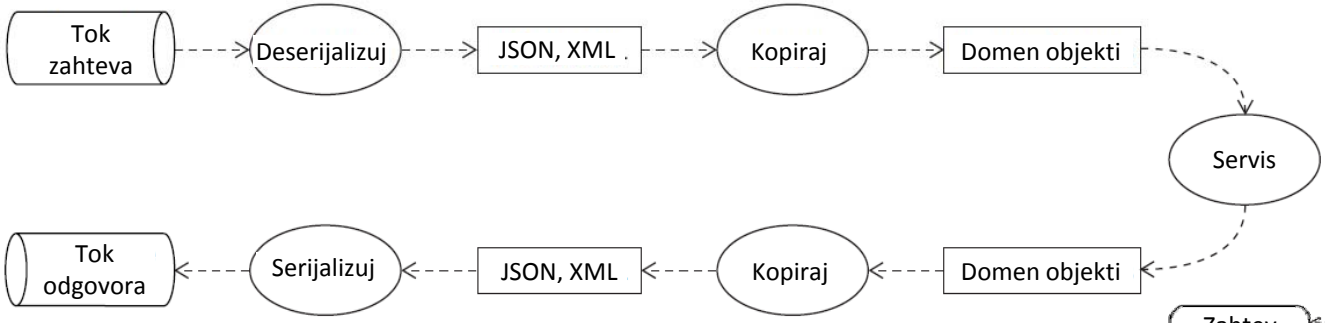
Iskazi rutiranja za API inspirisana resursima:
-Šabloni za URI
-Anotiranje metoda zahteva
-Anotiranje tipa medija

API stilovi i servis kontroleri:
Najpre interfejs?
Najpre kodiranje?
Servis kontroler po entitetu?
Servis kontroler po slučaju upotrebe?

Interkacija uz upotrebu servis kontrolera (Service Controller)
Oslanjanje na servisno okruženje → automatska deserijalizacija zahteva i serijalizacija odgovora
Upotreba klasa servis interfejsa (npr. WS - Service Endpoint Interface) → iskazi rutiranja odvojeni od implementacije

Interkacija sa servisom

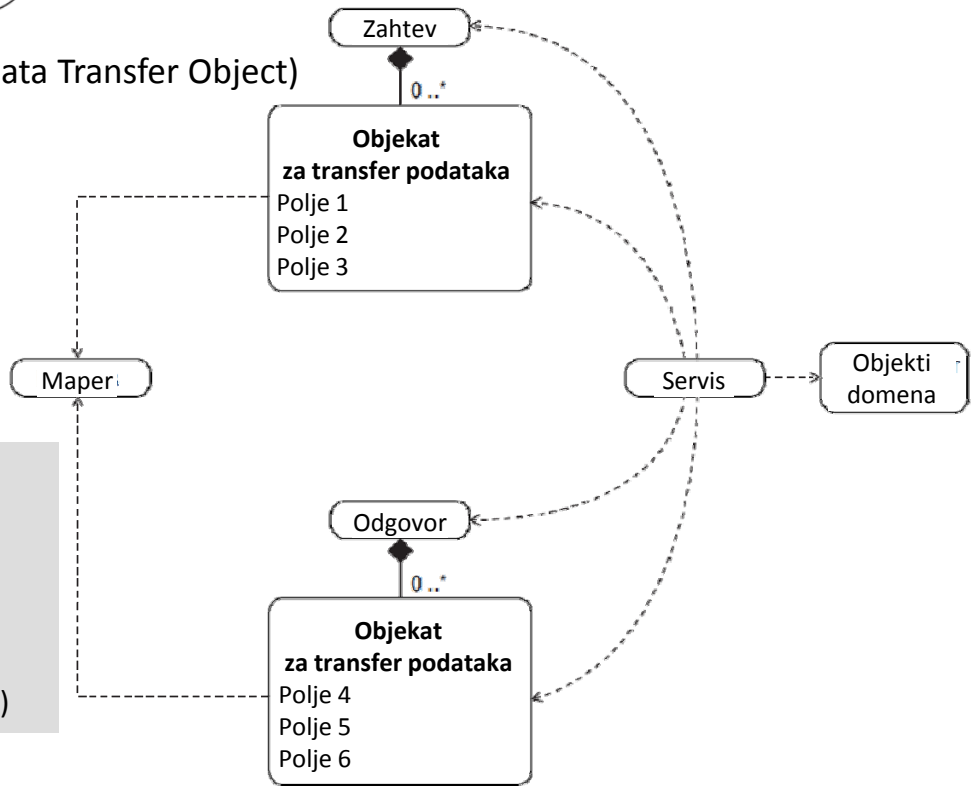
Kako obezbediti da zahtevi i odgovori mogu da variraju nezavisno od konkretnog formata koji se koristi u implementaciji servisa?



Interkacija uz upotrebu objekata za transfer podataka (Data Transfer Object)

Kreiranje različitih klasa za:

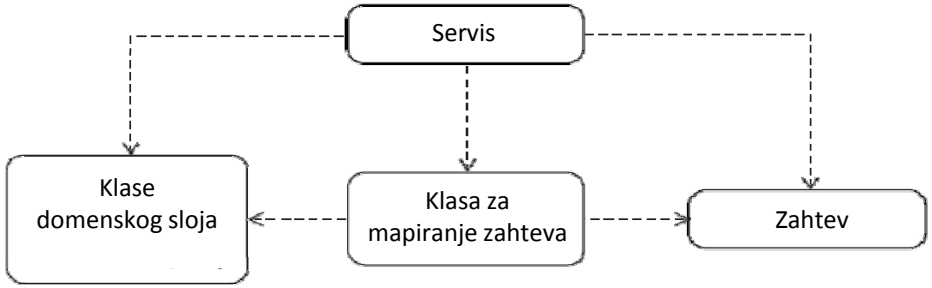
- zahteve
- odgovore
- domenske objekte
- potrebna mapiranja



Praktičnost
Dodatni posao (za kompleksne DTO)
Veličina DTO (opterećenje procesora i mreže)
Specifični DTO za pojedine klijente → mapiranje?
Binarna ili tekst forma? Klijent i servis koriste iste algoritme?
Razmena kolekcija DTO, a ne samo pojedinačnih DTO
Upotreba tolerantnog koda (delovi promenljivi ili pogrešno formatirani)

Interkacija sa servisom

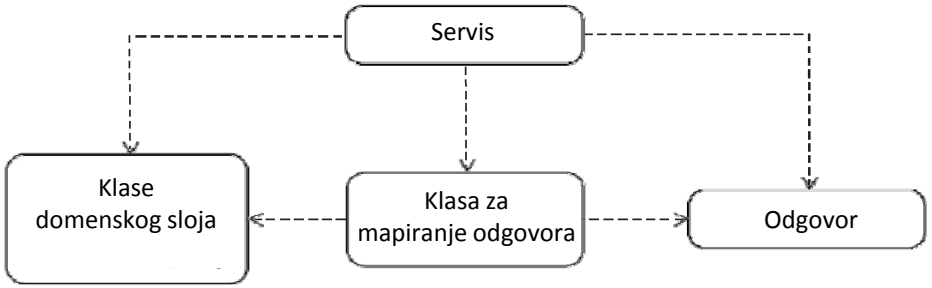
Kako da servis obrađuje zahteve od klijenata koji šalju semantički ekvivalentne, ali strukturno različite zahteve?



Interkacija uz upotrebu klasa za mapiranje zahteva (Request Mapper)
Kreiranje klasa za mapiranje koja delove iz zahteva:
- transformiše u domenske objekte
ili
- transformiše u posredničke objekte koji se koriste za kreiranje domenskih objekata

Ukoliko je samo jedna vrsta zahteva bolje je koristiti DTO
Specijalizacija klasa za mapiranje, kod složenijih struktura koje se sastoji od logički nezavisnih delova (npr. porudžbina sadrži info o klijentu)
Pogodno je koristiti klase za mapiranje kada je servisu potreban samo mali deo celokupnog zahteva (npr. zatevi definisani standardom)
Koriste se kod API stilova inspirisanih porukama i resursima, a ne i kod API inspirisanih RPC (poruke su definisane potpisom metoda)
Klase za mapiranje mogu imati kompleksne algoritme → otežano održavanje, opterećenje servera, povećano vreme odziva servisa

Interkacija sa servisom
Kako da veći broj servisa
deli logiku potrebnu za formiranje odgovora na zahtev?



Interkacija uz upotrebu klasa
za mapiranje odgovora (Response Mapper)

Kreiranje klasa za mapiranje koja:
- mapira podatke
- poseduje transformacionu logiku
- formatira odgovor

Ukoliko je više vrsta odgovora potrebno,
predlaže se kreiranje zasebnih klasa za mapiranje odgovora

Šta je odgovornost servisa,
a šta odgovornost klase za mapiranje odgovora?

U slučaju upotrebe povezivanja servisa,
logika kreiranja URI treba da bude van klase za mapiranje odgovora