

Markup jezici i XML

Šta je markup

- oznaka (tag, code) koja opisuje deo sadržaja

The Supply of Money

CT:
(chapter title)

To understand chronic inflation and, in general, to learn what determines prices and why they change, we must now focus on the behavior of the two basic causal factors: the supply of and the demand for money.

The supply of money is the total number of currency units in the economy. Originally, when each currency unit was defined strictly as a certain weight of gold or silver, the name and the weight were simply interchangeable. Thus, if there are \$100 billion in the economy, and the dollar is defined as 1/20 of a gold ounce, then M can be equally considered to be \$100 billion or 5 billion gold ounces. As monetary standards became lightened and debased by governments, however, the money supply increased as the same number of gold ounces were represented by an increased supply of francs, marks or dollars.

Debasement was a relatively slow process. Kings could not easily have explained continuous changes in their solemnly defined standards. Traditionally, a new king ordered a recoinage with his own likeness stamped on the coins and, in the process, [p. 44] often redefined the unit so as to divert some much needed revenue into his own coffers. But this variety of increased money supply did not usually occur more than once in a generation. Since paper currency did not yet exist, kings had to be content with debasement and its hidden taxation of their subjects.

H1: →
(first-level
heading)

1. What Should the Supply of Money Be?

What should the supply of money be? What is the “optimal” supply of money? Should M increase, decrease, or remain constant, and why?

Šta je markup

- markup je korišćen i na papiru prilikom slaganja teksta za označavanje pojedinih delova
 - naslovi, citati, brojevi poglavlja, itd.
- markup ne predstavlja sadržaj,
- već uputstvo kako neki sadržaj obraditi ili prikazati

Šta je markup

- markup se najčešće ne prikazuje eksplicitno
- krajnji čitalac ne vidi markup tagove
- ako se markup koristi za definisanje prezentacije dokumenta on je vidljiv iz grafičkog izgleda pojedinih elemenata
 - broj poglavlja, glavni naslov, podnaslov, citat

Ko definiše markup tagove

- firme
 - definicije su u vlasništvu firme
 - korisnik ne mora da zna sve detalje o markupu
 - Microsoft: Word .doc format
 - Adobe: PostScript, PDF
- otvoreni standardi
 - definišu ih tela za standardizaciju (ISO, W3C, ...)
 - svi detalji su poznati i javno dostupni
 - SGML, HTML, XML

Rad sa više markup sistema istovremeno

- tehnički komplikovan
- traži puno ručnih korekcija
- primer
 - autor koristi Microsoft Word (.doc)
 - lektor koristi OpenOffice Writer (.odt)
 - priprema i korektura koriste Adobe InDesign (.indd)
 - štampa koristi PDF (.pdf)
 - web izdanje koristi HTML (.html)
- šanse da dođe do problema u konverziji formata su vrlo velike

Otvoreni standardi za markup

- korišćenje otvorenih standarda bi trebalo da pojednostavi razmenu podataka između
 - različitih sistema
 - različitog softvera
 - različitih učesnika u proizvodnji sadržaja

Primer markupa

- Quark XPress

@LFI: Prva stavka

@LMI: Druga stavka

@LMI: Treća stavka

@LLI: Četvrta stavka

- HTML

Prva stavka

Druga stavka

Treća stavka

Četvrta stavka

Razlike:

- HTML ima hijerarhiju tagova, Quark nema
- Quark mora posebno da označava prvi i poslednji element liste zbog formatiranja
- Quark ne može da definiše listu u listi

Nejasan markup

- nejasan (nedorečen, višeznačan) markup
 - u formatiranju: oslanjanje na vizuelni izgled delova teksta kao opis značenja
 - u strukturi: vidi prethodni primer
 - nedostajući tagovi
 - “na slici ispod je prikazano...”

Razdvajanje strukture i prikaza

- osnovni razlozi za pojavu sistema sa nejasnim markupom
 - fokus na prikaz dokumenta
 - ograničenost na sopstveni, zatvoreni sistem
 -
- otvoreni standardi za markup se zasnivaju na konceptima
 - markiranja strukture
 - markiranja značenja

Označavanje strukture i značenja

- označavanje strukture

```
<chapter>  
  <section>  
    <title>...</title>  
    ...  
  </section>  
  <section>  
    <title>...</title>  
    ...  
  </section>  
</chapter>
```

- označavanje značenja

```
<opis>  
  Vlade Divac je centar i  
  visok je <visina>211  
  cm</visina> i težak je  
  <masa>120 kg</masa>. ...  
</opis>
```

Ovi koncepti nisu suprotstavljeni i mogu se kombinovati

Istorija standarda za strukturirani markup

- 1969: Generic Markup Language (GML)
 - definisanje različitih tipova dokumenata (skupova tagova)
 - dokument kao hijerarhija elemenata
 - razvijen u okviru IBM-a kao komercijalan proizvod
- 1980-1986: Standard Generalized Markup Language (SGML)
 - razvijen na bazi GML-a kao otvoreni standard
 - ANSI: American National Standards Institute
 - GCA: Graphic Communications Association
 - ISO: International Organization for Standardization
 - vrlo fleksibilan, sa širokim područjem primene
 - Association of American Publishers (AAP): The Manuscript Project

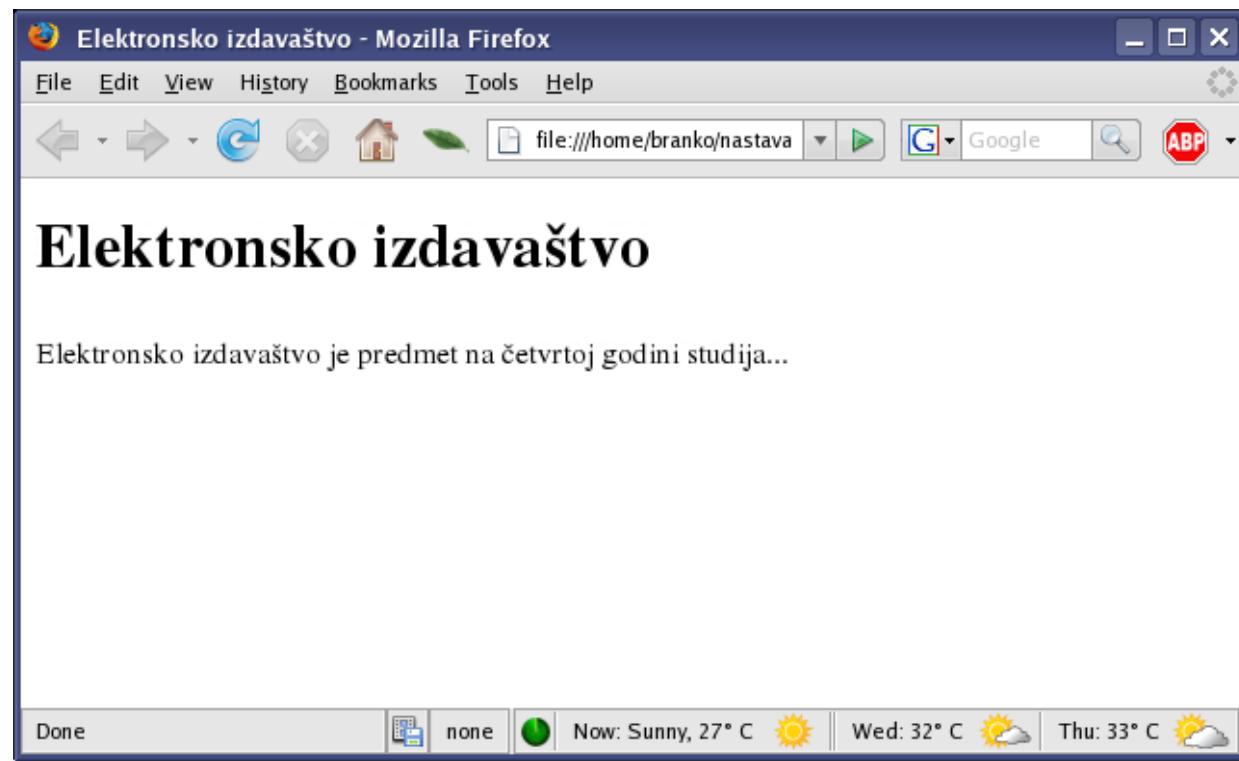
Istorija standarda za strukturirani markup

- 1989-1992: HyperText Markup Language (HTML)
 - razvijen u CERN-u kao jednostavan, fiksan skup tagova
 - kasnije formalno definisan kao jedna SGML gramatika
 - jednostavnost HTML-a je jedan od razloga za brzi rast Interneta
- 1998: Extensible Markup Language (XML)
 - HTML nije mogao da posluži svrsi u svim primenama na webu
 - SGML je suviše složen (fleksibilan) standard
 - XML je definisan kao podskup SGML-a koji je znatno jednostavnije implementirati
 - izbaciti 10% funkcionalnosti SGML-a (koje se ionako retko koriste) i smanjiti složenost za 90%

HTML

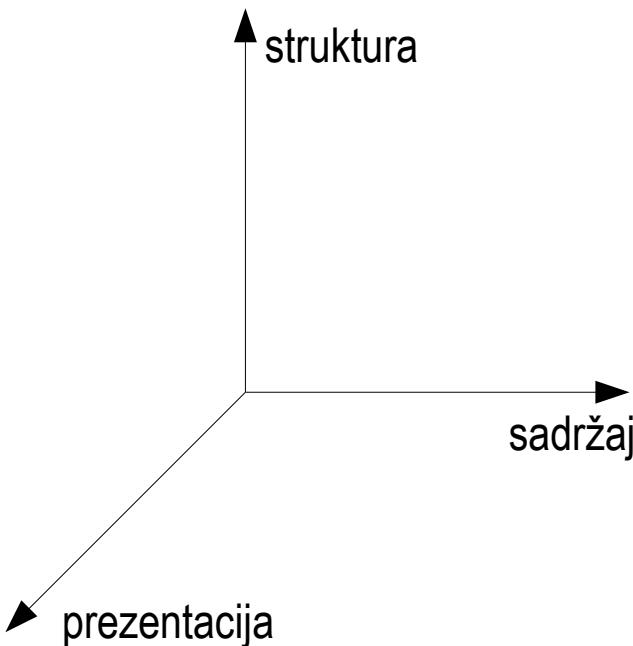
- dokument je hijerarhija elemenata
- mogu se koristiti samo unapred definisani tagovi

```
<html>  
  <head>...</head>  
  <body>  
    <h1>Elektronsko izdavaštvo</h1>  
    <p>...</p>  
    ...  
  </body>  
</html>
```



Pojam dokumenta

- osnovni nosilac informacija
- osnovna jedinica razmene informacija
- tri karakteristike (dimenzije) dokumenta
 - sadržaj
 - struktura
 - prezentacija



XML

- ne definiše nikakve tagove unapred, već ih korisnik sam definije
- XML je jezik za definisanje markup jezika, tj. metajezik
- HTML se može posmatrati kao jedan od jezika definisanih pomoću XML-a
- drugi primeri
 - CrossRef (povezivanje bibliografskih podataka)
 - CML (opis hemijskih jedinjenja)
 - OFX (razmena finansijskih dokumenata i elektronsko plaćanje)
 - NewsML (novinsko izdavaštvo)
 - SVG (vektorska grafika)
 - MathML (matematičke formule)

XML podrazumeva familiju standarda

- XSL (Extensible Stylesheet Language) - vizuelizacija
 - XSLT: XSL Transformations
 - XSL-FO: XSL Formatting Objects
- XPath: označavanje strukture dokumenata
- XLink: povezivanje dokumenata
- XQuery: pretraživanje dokumenata (po sadržaju i strukturi)

Ciljevi XML-a

- da odgovara upotrebi na Internetu
- mogućnost korišćenja od strane različitih aplikacija
- (jednosmerna) kompatibilnost sa SGML-om
- da se programi za obradu XML dokumenata pišu lako
- jednoznačnost prilikom obrade XML dokumenata
- XML dokumenti čitki i rezonski jasni
- dizajn XML-a se vrši brzo
- dizajn XML-a je formalan i koncizan
- kreiranje XML dokumenata je lako
- konciznost je od minimalne važnosti

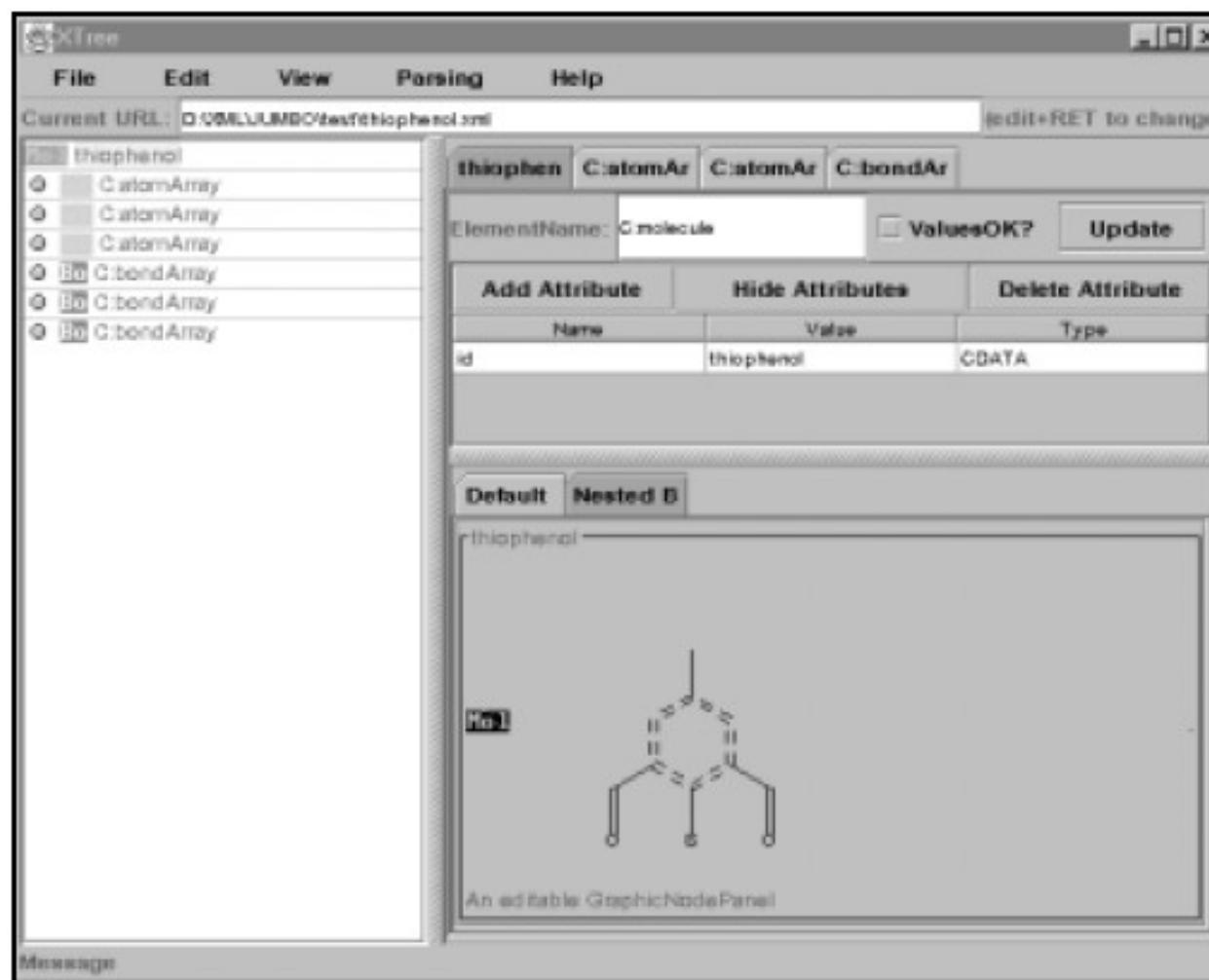
Primer XML dokumenta

- CML opis molekula H₂O

```
<CML>
  <MOL TITLE="Water">
    <ATOMS>
      <ARRAY BUILTIN="ELSYM">H 0 H</ARRAY>
    </ATOMS>
    <BONDS>
      <ARRAY BUILTIN="ATID1">1 2</ARRAY>
      <ARRAY BUILTIN="ATID2">2 3</ARRAY>
      <ARRAY BUILTIN="ORDER">1 1</ARRAY>
    </BONDS>
  </MOL>
</CML>
```

Primer XML dokumenta

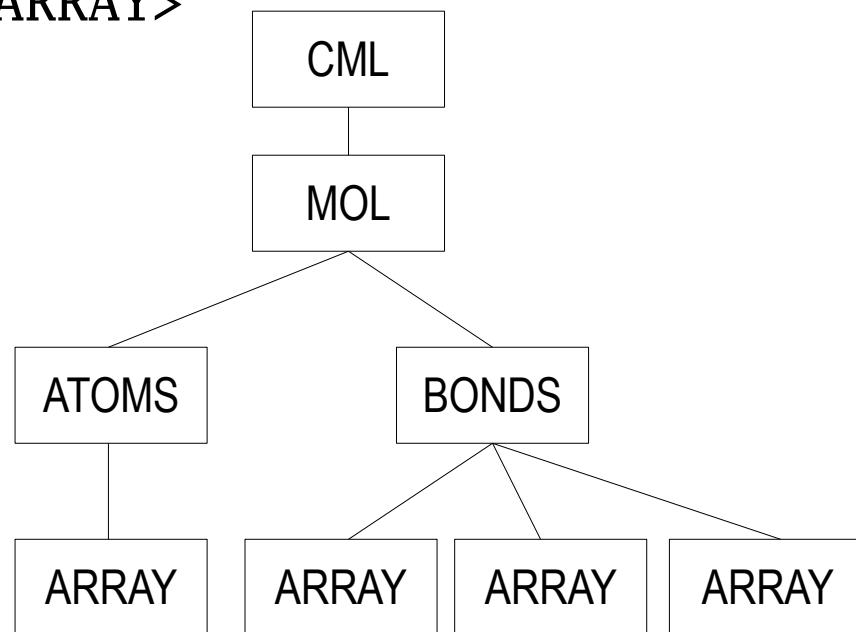
- prethodni dokument nema prikaz sam po sebi
- može se vizuelizovati na razne načine
- prikaz prethodnog dokumenta u CML browseru



Struktura XML dokumenta

- XML dokument predstavlja hijerarhiju elemenata

```
<CML>
  <MOL TITLE="Water">
    <ATOMS>
      <ARRAY BUILTIN="ELSYM">H 0 H</ARRAY>
    </ATOMS>
    <BONDS>
      <ARRAY BUILTIN="ATID1">1 2</ARRAY>
      <ARRAY BUILTIN="ATID2">2 3</ARRAY>
      <ARRAY BUILTIN="ORDER">1 1</ARRAY>
    </BONDS>
  </MOL>
</CML>
```



Element i tag

- element je čvor u hijerarhijskoj strukturi dokumenta
 - on može sadržati druge čvorove
- tag je tekstualna oznaka (markup) za početak ili kraj elementa
 - početni (otvarajući) tag: <tag>
 - završni (zatvarajući tag): </tag>
- sadržaj elementa nalazi se između početnog i završnog taga:
 - <tag>sadržaj</tag>
- ako je element prazan, može da se piše dvojako:
 - <tag/>
 - <tag></tag>



Tagovi su način da se hijerarhijska struktura (stablo elemenata) prikaže pomoću linearne strukture (tekstualnog dokumenta, tj. niza slova).

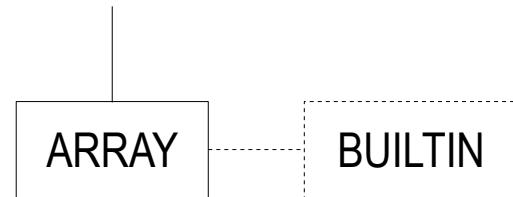
Sadržaj elementa

- sadržaj može biti
 - tekst
`<A>trlababal`
 - podelementi
`<A>
 ...
 <C>...</C>
`
 - mešavina teksta i podelemenata
`<A>
 trlababal trlb trlababal
`
 - prazan sadržaj
`<A>
<A/>`

Atribut

- element može da ima attribute
- atribut ima naziv i sadržaj
- sadržaj atributa je nestrukturiran tekst
- atribut se (najčešće) ne smatra posebnim čvorom stabla
- primer

```
<ARRAY BUILTIN="ELSYM">H O H</ARRAY>
```



Dobro-formirani XML dokument

- #1 ima jedan korenski element
- #2 elementi se mogu ugnježdavati ali ne i preklapati
- #3 vrednosti atributa moraju biti unutar navodnika
- #4 element ne može imati dva atributa sa istim imenom
- ... (*biće još kasnije*)

Dobro-formirani XML dokument

- #1 ima jedan korenski element

```
<CONTACT>
  <NAME>Petar Petrović</NAME>
  <ADDRESS>Dunavska 1, Novi Sad</ADDRESS>
  <COMPANY>Gradska biblioteka u Novom Sadu</COMPANY>
  <TEL_NUM>444-333</TEL_NUM>
  <EMAIL>pp@yahoo.com</EMAIL>
</CONTACT>
```

- ne bi bilo ispravno:

```
<CONTACT>
  ...
</CONTACT>
<CONTACT>
  ...
</CONTACT>
```

neki tekst

```
<CONTACT>
  ...
</CONTACT>
  još neki tekst
```

Dobro-formirani XML dokument

- #2 elementi se mogu ugnježdavati ali ne i preklapati

pravilno

```
<A>...<B>...</B>...</A>
```



nepravilno

```
<A>...<B>...</A>...</B>
```



Dobro-formiran XML dokument

- #3 vrednosti atributa moraju biti unutar navodnika

```
<telnum type="mobile">444-333</telnum>
<telnum type='mobile'>444-333</telnum>
```

- nije ispravno

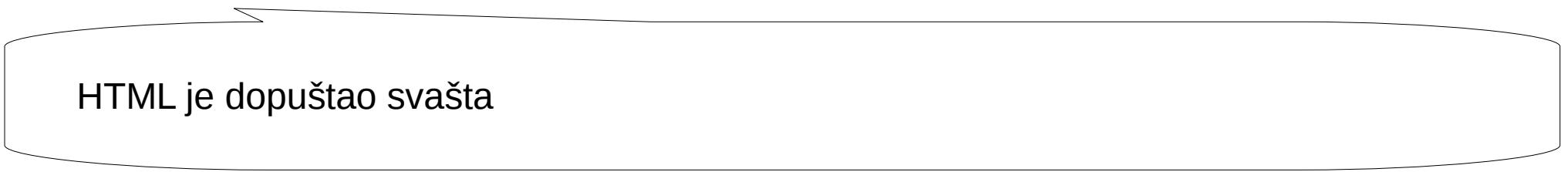
```
<telnum type=mobile>444-333</telnum>
```

Dobro-formirani XML dokument

- #4 element ne može imati dva elementa sa istim imenom
- primer
`<osoba ime="..." telefon="021/..." telefon="064/...>`

Dobro-formiran XML dokument

- XML dokument mora biti dobro formiran da bi mogao biti mašinski obradivan!



HTML je dopuštao svašta

Strukturiranje dokumenta

- šta je bolje?

```
<CONTACT>
  <NAME>Petar Petrović</NAME>
  <ADDRESS>Dunavska 1, Novi Sad</ADDRESS>
  <COMPANY>Gradska biblioteka</COMPANY>
  <TEL_NUM>444-333</TEL_NUM>
  <EMAIL>pp@yahoo.com</EMAIL>
</CONTACT>
```

```
<CONTACT>
  <NAME>
    <FIRST>Petar</FIRST>
    <LAST>Petrović</LAST>
  </NAME>
  <ADDRESS>Dunavska 1, Novi Sad</ADDRESS>
  <COMPANY>Gradska biblioteka</COMPANY>
  <TEL_NUM>444-333</TEL_NUM>
  <EMAIL>pp@yahoo.com</EMAIL>
</CONTACT>
```

Strukturiranje dokumenta

- šta je bolje?

<NAME>

 <FIRST>Petar</FIRST>

 <LAST>Petrović</LAST>

</NAME>

- ili

<NAME FIRST="Petar" LAST="Petrović"/>

Strukturiranje dokumenta

- da li koristiti atribute ili elemente?
- nema univerzalnog odgovora
- neka pravila
 - podaci bi trebalo da budu smešteni u elementima
 - podaci o podacima (metapodaci) u atributima
 - kada nismo sigurni, onda u elementima
- podaci u atributima se ne mogu strukturirati
- elementi se mogu lakše prilagođavati izmenama u budućnosti

Strukturiranje dokumenta

- šta mogu biti metapodaci?
- primer

```
<poet language="english">Homer</poet>
```

```
<poet language="greek">Ομηρος</poet>
```



Davanje imena elementima i atributima

- ime elemenata i atributa mora biti tzv. XML ime:
 - pravi se razlika između velikih i malih slova
 - za davanje imena mogu da se koriste:
 - slova, cifre, donja crta _, crtica -, dvotačka : i tačka .
 - ime može početi slovom ili donjom crtom
 - ne može početi slovima xml (rezervisano)
 - primeri
 - <BrojMotora>123456789</BrojMotora>
 - <broj_sasije>123456789</broj_sasije>
 - <регистарскиБрој>NS 123-456</регистарскиБрој>

Ostali delovi XML dokumenta

- komentari
- procesne instrukcije
- entiteti
- CDATA sekcije

XML komentari

- navode se između <!-- i -->
- predstavljaju komentar autora XML dokumenta
- ignorišu se kao sadržaj, ne prikazuju se, ne obrađuju se programski
- mogu se nalaziti bilo gde izvan taga
- primer

```
<contact>
<!-- ovaj kontakt je zastareo -->
  <name>
    <first>...</first>
    ...
  ...
  ...
```

XML procesne instrukcije

- navode se između <? i ?>
- predstavljaju instrukcije softveru
- nisu namenjene krajnjem korisniku (čoveku)
- nisu deo sadržaja dokumenta
- mogu se nalaziti bilo gde izvan taga
- primeri
 - <?xml version="1.0" encoding="utf-8" standalone="yes"?>
 - <?xml-stylesheet href="main.css" type="text/css"?>

XML deklaracija

- XML deklaracija <?xml . . . ?> se nalazi na početku dokumenta
- ima tri atributa
 - version: verzija XML jezika koja se koristi u dokumentu
 - encoding: kodni raspored koji se koristi u dokumentu
 - standalone: da li će biti neophodno učitati i DTD
- nije obavezna, ali je poželjna (obavezna od XML v1.1)
- primer

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

Dobro-formirani XML dokument

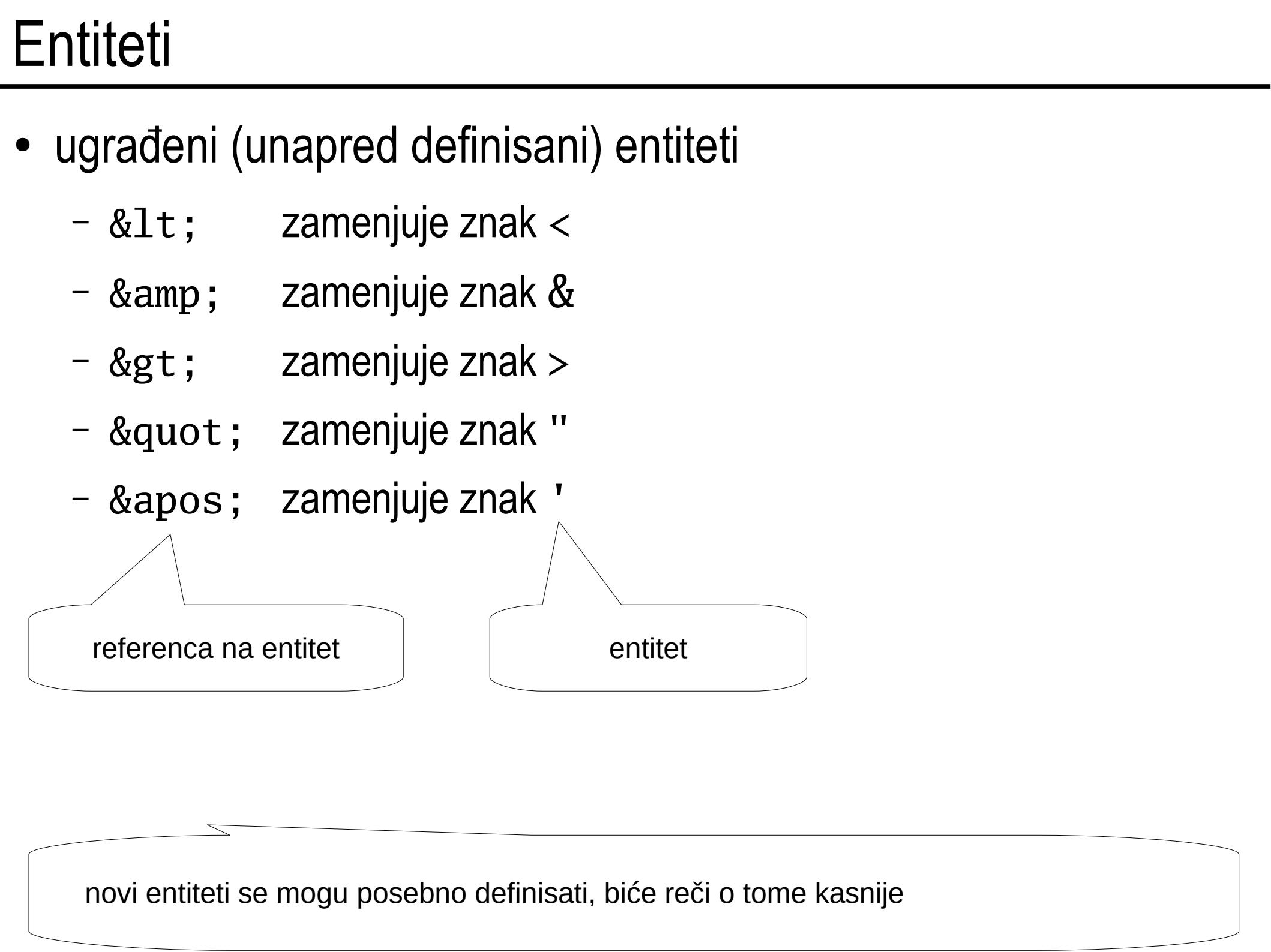
- #5 komentari i procesne instrukcije se ne smeju nalaziti unutar taga
- primeri

```
<osoba <!-- ovaj kontakt je zastareo --> ime="..."><osoba <?ignore?> ime="...">
```

Entiteti

- ugrađeni (unapred definisani) entiteti

- < zamenjuje znak <
- & zamenjuje znak &
- > zamenjuje znak >
- " zamenjuje znak "
- ' zamenjuje znak '



referenca na entitet

entitet

novi entiteti se mogu posebno definisati, biće reči o tome kasnije

CDATA sekcije

- navode se između `<![CDATA[i]]>`
- sadrži tekst koji se interpretira direktno, bez zamene entiteta
- primer

```
<p>You can use a default <code>xmlns</code> attribute to  
avoid having to add the svg prefix to all your  
elements:</p>  
<![CDATA[  
    <svg xmlns="http://www.w3.org/2000/svg"  
        width="12cm" height="10cm">  
        <ellipse rx="110" ry="130" />  
        <rect x="4cm" y="1cm" width="3cm" height="6cm" />  
    </svg>  
]]>
```

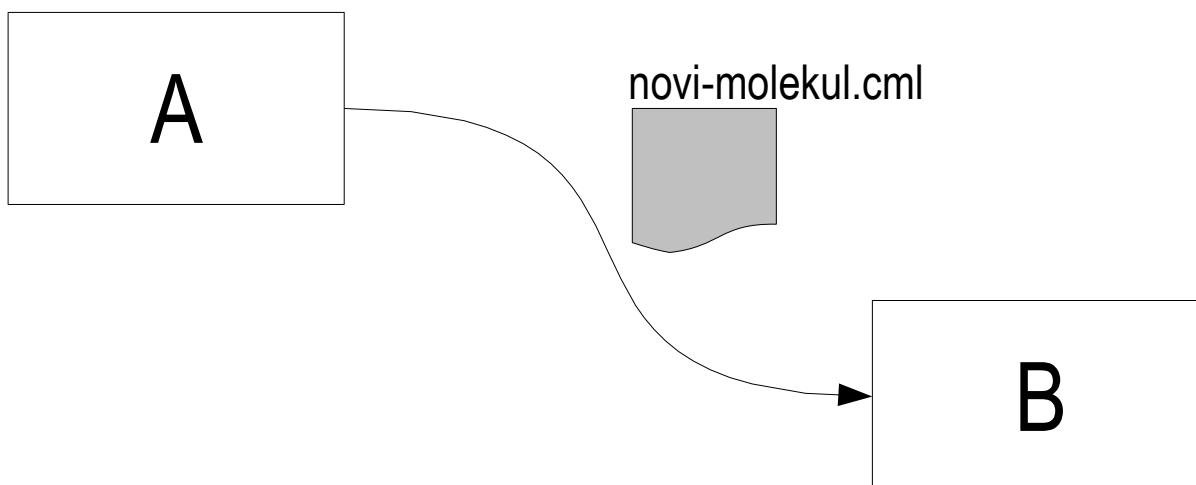
SVG slika ugrađena u HTML dokument;

- ako je stavimo u CDATA sekciju, biće prikazan SVG listing
- ako je ne stavimo u CDATA sekciju, biće prikazana slika

Document Type Definition (DTD)

XML za razmenu podataka

- scenario: dva ili više učesnika komuniciraju putem XML dokumenata
 - XML dokumenti su i human-readable i machine-readable
 - softver za rad sa XML dokumentima postoji na svim računarskim platformama (PC/Mac, Windows/Linux/Unix/MacOS)
 - primer: učesnik A šalje učesniku B opis novog molekula
 - opis je dat u XML obliku, po CML standardu
 - nije važno koji konkretni softver koriste A i B, sve dok mogu da čitaju CML



XML za razmenu podataka

- primer 2: svi učesnici koji razmenjuju vektorske slike koriste SVG (Scalable Vector Graphics) format
- učesnici mogu da koriste različit softver
 - Adobe Illustrator
 - Inkscape
 - CorelDRAW
- ukoliko softver potpuno podržava SVG format, prilikom razmene ne dolazi do gubitaka

XML za razmenu podataka

- za uspešnu razmenu podataka potrebno je da svi učesnici koriste isti format
- ako koriste XML, format može da se definiše u posebnom dokumentu
 - dokument koristi posebnu sintaksu
 - može se publikovati svim zainteresovanim učesnicima
 - predstavlja specifikaciju poruka koje se koriste u komunikaciji

Document Type Definition (DTD)

- deo osnovnog XML standarda
- DTD fajl (dokument) opisuje format klase/familije/tipa XML dokumenata
 - koji elementi i entiteti se mogu pojaviti na kom mestu u dokumentu
 - šta je sadržaj elemenata i atributa

Document Type Definition (DTD)

- primer 1: CML (Chemical Markup Language)

```
<!-- =====>
<!-- ELEMENTS for chemical and crystallographic concepts -->
<!-- =====>
<!-- NOTE
for elements which have element-specific values for the
builtin attribute, those values are already listed as
entities
-->
<!-- =====>
<! [%fullCML;[
<!ELEMENT %molecule.content 'ANY'
]]>
<!ELEMENT %molecule.content '(atomArray, bondArray)'>
<!ELEMENT molecule      %molecule.content;>
<!ATTLIST molecule
                  %tit_id_conv_dict;
                  %count;
>
<!ELEMENT formula        ANY>
<!ATTLIST formula
                  %tit_id_conv_dict;
                  %count;
>
```

Document Type Definition (DTD)

- primer 2: SVG (Scalable Vector Graphics)

```
<!-- Shape
      path, rect, circle, line, ellipse, polyline, polygon
      This module declares markup to provide support for graphical
      shapes.
-->
<!-- a list of points -->
<!ENTITY % Points.datatype "CDATA" >

<!-- Qualified Names (Default) ..... -->
<!ENTITY % SVG.path.qname "path" >
<!ENTITY % SVG.rect.qname "rect" >
<!ENTITY % SVG.circle.qname "circle" >
<!ENTITY % SVG.line.qname "line" >
<!ENTITY % SVG.ellipse.qname "ellipse" >
<!ENTITY % SVG.polyline.qname "polyline" >
<!ENTITY % SVG.polygon.qname "polygon" >
... 
```

Primer DTD-a

- primer XML dokumenta

```
<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
  <profession>computer scientist</profession>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

- korenski element je person, i sadrži četiri podelementa

- element name, koji sadrži dva podelementa
 - first_name
 - last_name
 - tri elementa profession

Primer DTD-a

- DTD koji odgovara prethodnom dokumentu

```
<!ELEMENT person      (name, profession*)>
<!ELEMENT name        (first_name, last_name)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name   (#PCDATA)>
<!ELEMENT profession (#PCDATA)>
```

- element **person** sadrži podelemente **name** (tačno jednom) i **profession** (nula ili više puta)
 - element **name** sadrži podelemente **first_name** i **last_name**
 - element **first_name** sadrži tekst
 - element **last_name** sadrži tekst
 - element **profession** sadrži tekst

- dtd se obično čuva kao poseban fajl, npr. **person.dtd**

Primer DTD-a

- prethodni DTD je definisao klasu *validnih* dokumenata
- dokument je validan ako odgovara svom DTD-u
- primer validnog dokumenta

```
<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
</person>
```



element profession
nije obavezan

Primer DTD-a

- primer invalidnog dokumenta

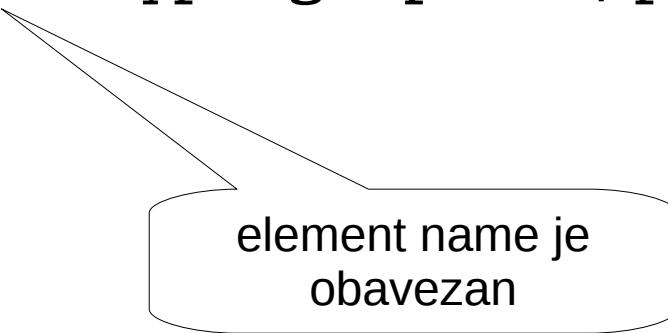
```
<person>
```

```
    <profession>computer scientist</profession>
```

```
    <profession>mathematician</profession>
```

```
    <profession>cryptographer</profession>
```

```
</person>
```

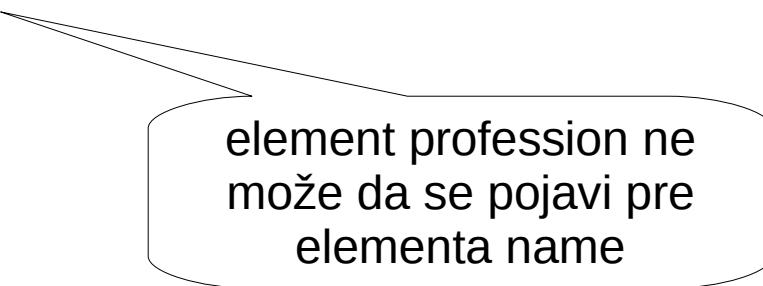


element name je
obavezан

Primer DTD-a

- primer invalidnog dokumenta

```
<person>
  <profession>computer scientist</profession>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
</person>
```

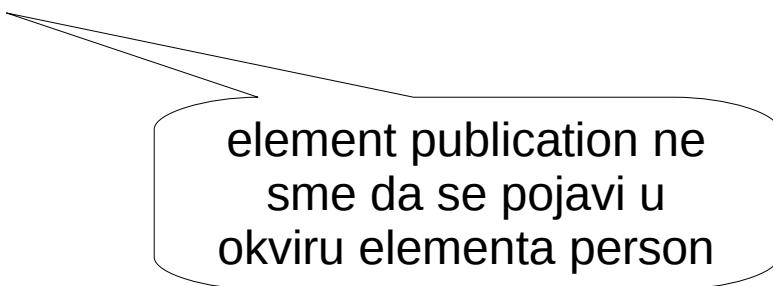


element profession ne
može da se pojavi pre
elementa name

Primer DTD-a

- primer invalidnog dokumenta

```
<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
  <profession>mathematician</profession>
  <profession>cryptographer</profession>
  <publication>On Computable
    Numbers...</publication>
</person>
```

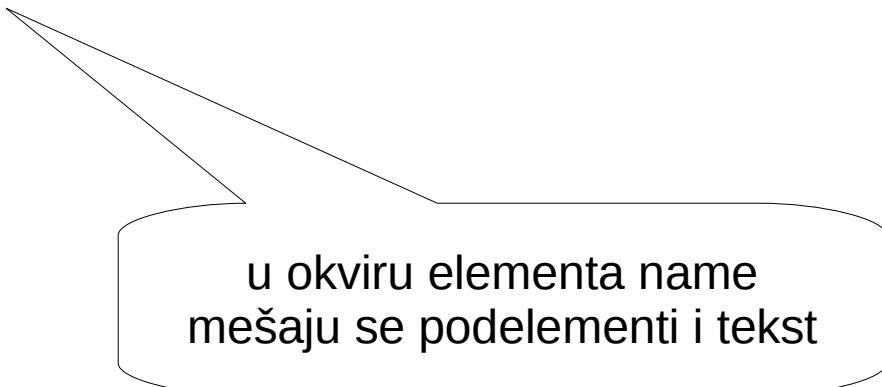


element publication ne
sme da se pojavi u
okviru elementa person

Primer DTD-a

- primer invalidnog dokumenta

```
<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
  was a <profession>computer scientist</profession>,
  a <profession>mathematician</profession>, and a
  <profession>cryptographer</profession>
</person>
```



Vrste deklaracija u DTD-u

- deklaracija elementa
- deklaracija atributa
- deklaracija entiteta
- deklaracija notacija
- ...

Deklaracija elementa

- opšti oblik
`<!ELEMENT naziv (specifikacija_sadržaja)>`
- naziv elementa mora da odgovara pravilima
 - nema razmaka; počinje slovom ili donjom crtom; uključuje slova, cifre, donju crtu, crticu i tačku

Specifikacija sadržaja elementa

- #PCDATA (parsed character data)
 - sadržaj elementa je tekst, bez podelemenata
 - tekst je parsiran, tj. reference na entitete su razrešene
 - primer
 - <!ELEMENT phone_number (#PCDATA)>

Specifikacija sadržaja elementa

- sekvenca podelemenata
 - podelementi se razdvajaju zarezom
 - uz naziv podelementa navodi se oznaka broja ponavljanja
 - redosled podelemenata je bitan
 - broj ponavljanja podelemenata je bitan

Specifikacija sadržaja elementa

- sekvenca podelemenata

- primer

```
<!ELEMENT name (first_name, last_name)>
```

- validan XML:

```
<name>
  <first_name>...</first_name>
  <last_name>...</last_name>
</name>
```

- invalidan XML:

```
<name>
  <last_name>...</last_name>
  <first_name>...</first_name>
</name>
```

```
<name><first_name>...</first_name></name>
```

Specifikacija sadržaja elementa

- sekvenca podelemenata
 - broj ponavljanja podelemenata izražava se posebnim znacima:
 - ? : element se pojavljuje jednom ili nijednom (0..1)
 - * : element se pojavljuje nijednom, jednom, ili više puta (0.. ∞)
 - + : element se pojavljuje jednom ili više puta (1.. ∞)
 - ako se oznaka ne navede, podrazumeva se tačno jednom (1..1)

Specifikacija sadržaja elementa

- sekvenca podelemenata

- primer

```
<!ELEMENT name (first_name, middle_name?, last_name?)>
```

- validni elementi:

```
<name>
  <first_name>Madonna</first_name>
  <last_name>Ciccone</last_name>
</name>
```

```
<name>
  <first_name>Madonna</first_name>
  <middle_name>Louise</middle_name>
  <last_name>Ciccone</last_name>
</name>
```

```
<name>
  <first_name>Madonna</first_name>
</name>
```

Specifikacija sadržaja elementa

- sekvenca podelemenata

- primer

```
<!ELEMENT name (first_name, middle_name?, last_name?)>
```

- invalidni elementi:

```
<name>
  <last_name>Ciccone</last_name>
  <first_name>Madonna</first_name>
</name>
```

redosled nije
ispravan

```
<name>
  <first_name>Madonna</first_name>
  <middle_name>Louise</middle_name>
  <middle_name>Veronica</middle_name>
  <last_name>Ciccone</last_name>
</name>
```

middle_name ne
može da se
ponavlja više
puta

Specifikacija sadržaja elementa

- izbor (jednog od navedenih podelemenata)
 - podelementi se razdvajaju uspravnom crtom
 - izbor može obuhvatati dva ili više podelemenata

Specifikacija sadržaja elementa

- izbor
 - primer

```
<!ELEMENT methodResponse (params | fault)>
```
 - validan XML

```
<methodResponse><params>...</params></methodResponse>
<methodResponse><fault>...</fault></methodResponse>
```
 - invalidan XML

```
<methodResponse>
  <params>...</params>
  <fault>...</fault>
</methodResponse>

<methodResponse></methodResponse>
```

Specifikacija sadržaja elementa

- upotreba zagrada
 - sekvence, izbori i sufiksi se mogu kombinovati
 - sekvenca ili izbor može se staviti unutar zagrada (. . .)
 - na zgradu se može dodati sufiks *, +, ?
 - zgrada se može koristiti kao podelement u sekvencama i izborima

Specifikacija sadržaja elementa

- upotreba zagrada

- primer 1

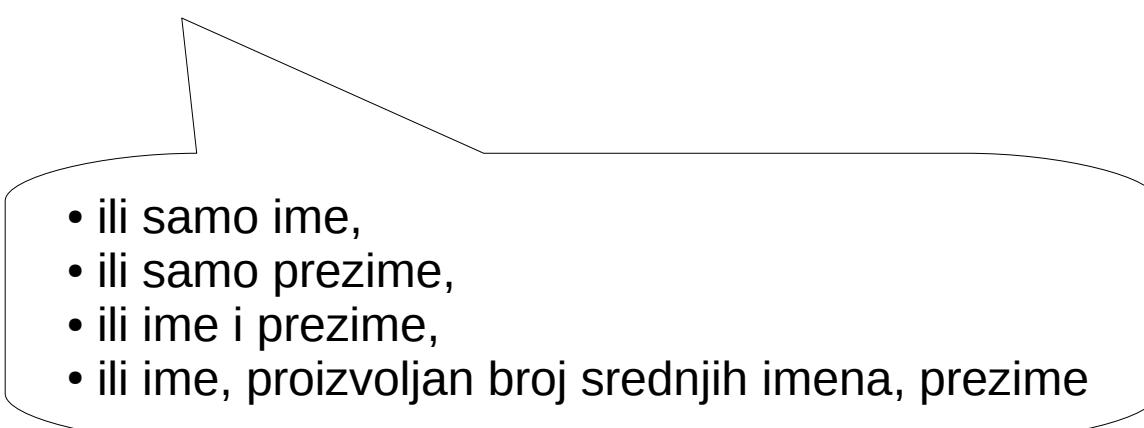
```
<!ELEMENT circle (center, (radius | diameter))>
<!ELEMENT center ((x, y) | (r, t))>
```

- primer 2

```
<!ELEMENT center ((x,y)|(y,x)|(r,t)|(t,r))>
```

- primer 3

```
<!ELEMENT name (last_name |
(first_name, ((middle_name+, last_name) |
(last_name?))))>
```

- 
- ili samo ime,
 - ili samo prezime,
 - ili ime i prezime,
 - ili ime, proizvoljan broj srednjih imena, prezime

Specifikacija sadržaja elementa

- mešani sadržaj
 - element sadrži tekst i podelemente
 - definiše se kao izbor čiji prvi element je #PCDATA, ostali elementi su mogući podelementi, a cela grupa ima broj ponavljanja *
 - primer

```
<!ELEMENT definition (#PCDATA | term)*>
```
 - validan XML

<definition>The **<term>**Turing Machine**</term>** is an abstract finite state automaton with infinite memory that can be proven equivalent to any other finite state automaton with arbitrarily large memory. Thus what is true for a Turing machine is true for all equivalent machines no matter how implemented.**</definition>**

Specifikacija sadržaja elementa

- mešani sadržaj
 - nije moguće navesti broj ponavljanja podelemenata
 - nije moguće navesti da tekst mora biti samo ispred ili iza podelementa
 - deklaracija mešanog sadržaja se ne može koristiti za dalje grupisanje
 - nije moguće fiksirati redosled podelemenata

Specifikacija sadržaja elementa

- prazan sadržaj
 - prazan sadržaj se označava ključnom reči EMPTY
 - primer

```
<!ELEMENT img EMPTY>
```
 - validan XML

```

```

Specifikacija sadržaja elementa

- bilo kakav sadržaj
 - označava se ključnom reči ANY
 - uključuje tekst, podelemente, isti element ponavljen rekurzivno
 - nalik mešanom sadržaju bez fiksirane liste podelemenata
 - podelementi koji se pojavljuju u sadržaju moraju biti deklarisani
 - u praksi se koristi samo u toku razvoja DTD-a
 - primer
`<!ELEMENT page ANY>`

Deklaracija atributa

- opšti oblik
`<!ATTLIST imeElementa imeAtr tipAtr default>`
- naziv atributa mora da odgovara pravilima za formiranje imena
- tip atributa se bira iz konačnog skupa
- obaveznost atributa se bira iz konačnog skupa
- jedna deklaracija može da obuhvati više atributa jednog elementa
 - u praksi se svi atributi jednog elementa stavljaju u jednu deklaraciju
 - u praksi se deklaracija atributa navodi odmah ispod deklaracije elementa

Specifikacija sadržaja atributa

- primer

```
<!ELEMENT img EMPTY>
<!ATTLIST img
    src      CDATA #REQUIRED
    width   CDATA #REQUIRED
    height  CDATA #REQUIRED
    alt     CDATA #IMPLIED
```

>

- validan XML

```

```

Specifikacija sadržaja atributa

- mogući tipovi atributa
 - CDATA : bilo koji dobro formirani tekst
 - NMTOKEN : XML *name token* je (slično XML imenima) tekst koji nema razmake, sastavljen je od slova, cifara, znakova _, -, :, . i može početi bilo kojim od dozvoljenih znakova
 - NMTOKENS : jedan ili više NMTOKEN-a razdvojenih razmacima
 - *nabranje mogućih vrednosti*: navodi se lista mogućih vrednosti atributa međusobno razdvojenih uspravnom crtom, slično kao kod definicije izbora. Ukupna lista se stavlja u zagrade.
 - ID : string koji je XML ime i jedinstven je u okviru celog dokumenta, odnosno nijedan drugi atribut tipa ID ne može imati tu vrednost. Element može imati najviše jedan atribut tipa ID.

Specifikacija sadržaja atributa

- mogući tipovi atributa
 - IDREF : postojeća vrednost ID atributa nekog elementa u istom dokumentu
 - IDREFS : niz IDREF vrednosti razdvojenih razmakom
 - ENTITY : ime neparsiranog entiteta koji je deklarisan u DTD-u (*kasnije*)
 - ENTITIES : više ENTITY vrednosti razdvojenih razmakom
 - NOTATION : sadrži ime notacije koja je deklarisana u DTD-U (*kasnije*)

Specifikacija sadržaja atributa

- primer upotrebe različitih tipova atributa
 - tim inženjera radi na više projekata; jedan projekat uključuje više inženjera, a jedan inženjer može da radi na više projekata
 - za projekte i inženjere uvodimo identifikatore - po jedan atribut tipa ID
 - za povezivanje inženjera sa projektima uvodimo atribut tipa IDREFS
 - za povezivanje projekta sa inženjerima uvodimo atribut tipa IDREFS

```
<!ELEMENT project (name)>
<!ELEMENT engineer (firstName, lastName)>
<!ATTLIST engineer
          engineerID      ID      #REQUIRED
          assignments    IDREFS #REQUIRED>
<!ATTLIST project
          projectID     ID      #REQUIRED
          team          IDREFS #REQUIRED>
```

Specifikacija sadržaja atributa

- primer upotrebe različitih tipova atributa
 - definisanje dana u nedelji

```
<!ATTLIST date
dayOfWeek (mon|tue|wed|thu|fri|sat|sun)>
```
 - definisanje godišnjih doba

```
<!ATTLIST date season (spring|summer|autumn|winter)>
```

Specifikacija sadržaja atributa

- obaveznost pojavljivanja atributa
 - #IMPLIED : atribut nije obavezan; može se navesti ali i ne mora
 - #REQUIRED : atribut je obavezan; mora se navesti
 - #FIXED "value" : atribut nije obavezan, ali se smatra da uvek postoji u elementu i da ima datu fiksnu vrednost *value*; ako se eksplicitno navede u elementu, mora imati baš tu vrednost
 - "default" : atribut, ako se ne navede, ima podrazumevanu vrednost *default* datu pod navodnicima

Specifikacija sadržaja atributa

- obaveznost pojavljivanja atributa - primeri
 - datum rođenja osobe je obvezan, a datum smrti nije

```
<!ATTLIST person
    born CDATA #REQUIRED
    died CDATA #IMPLIED>
```
 - element biography ima atribut xmlns:xlink i njegova vrednost je uvek <http://www.w3.org/1999/xlink>, bez obzira da li je atribut naveden

```
<!ATTLIST biography xmlns:xlink CDATA #FIXED
    "http://www.w3.org/1999/xlink">
```

Deklaracija entiteta

- osim predefinisanih entiteta (< > & ' ") moguće je deklarisati nove
- opšti oblik deklaracije je
`<!ENTITY naziv "sadržaj">`
- primer
`<!ENTITY i18n "internationalization">`
 - referenca &i18n; koja se javlja u dokumentu po ovom DTD-u biće zamjenjena sadržajem internationalization
 - zamena se vrši prilikom parsiranja (učitavanja) XML dokumenta
- sadržaj entiteta može da sadrži druge entitete
 - samo-referenciranje i cirkularno referenciranje nisu dozvoljeni

Deklaracija entiteta

- eksterni entitet = smešten u posebnom fajlu, izvan DTD-a
 - zgodno kod velikih sadržaja
- opšti oblik deklaracije eksternog entiteta
`<!ENTITY naziv SYSTEM "adresa_fajla">`
- primjeri
`<!ENTITY footer SYSTEM
"http://www.ns.ac.yu/parts/footer.xml">`
`<!ENTITY footer SYSTEM "parts/footer.xml">`

Deklaracija entiteta

- eksterni entitet može imati i tzv. javni identifikator:
`<!ENTITY name PUBLIC "pubid" SYSTEM "sysid">`
- javni identifikator može da identificuje dobro poznati resurs (iz liste takvih) koji je javno dostupan
- liste javno dostupnih resursa su slabo razvijene
- PUBLIC identifikatori se retko koriste

Deklaracija entiteta

- parametarski entiteti = entiteti koji se definišu za potrebe DTD-a, a ne XML dokumenta

- opšti oblik deklaracije

```
<!ENTITY % ime "vrednost">
<!ENTITY % ime SYSTEM "adresa">
```

- primer (iz SVG DTD-a)

```
<!ENTITY % descTitle
          "((desc,title?)|(title,desc?))">
<!ENTITY % geExt  "">
<!ENTITY % circleExt  "">
<!ELEMENT circle (%descTitle;,
                  (animate|set|animateMotion|animateColor|
                  animateTransform %geExt;%circleExt;)*)>
```

Deklaracija entiteta

- neparsirani entiteti = fajlovi koji sadrže ne-XML podatke
- opšti oblik deklaracije

```
<!ENTITY naziv PUBLIC "pubid" notacija>
<!ENTITY naziv SYSTEM "sysid" notacija>
```
- referenca na neparsirani entitet može se naći samo kao vrednost atributa tipa ENTITY
- notacija predstavlja dodatnu informaciju za XML aplikaciju kako da rukuje podacima iz neparsiranog entiteta
- primer

```
<!ENTITY turing_getting_off_bus SYSTEM
    "http://www.turing.org.uk/turing/pil/bus.jpg"
    jpeg>
```

Deklaracija notacije

- notacija = pomoćni podaci za XML aplikaciju prilikom rukovanja sa neparsiranim entitetima
- opšti oblik deklaracije

```
<!NOTATION naziv PUBLIC "pubid">
<!NOTATION naziv SYSTEM "sysid">
```
- primer

```
<!NOTATION jpeg PUBLIC "image/jpeg">
<!NOTATION jpeg SYSTEM
    "C:/Program Files/imageViewer/viewer.exe">
```
- nema standarda niti preporuke kako treba da izgledaju notacije, sve je stvar aplikacije

Primer složenijeg DTD-a

- primer XML dokumenta

```
<CHAPTER id="c12345">
<TITLE>Markup: XML and Related Technologies</TITLE>
<AU><NAME>William E. Kasdorf</NAME>
<AFF><ROLE>President</ROLE><ORG>Impressions Book and
Journal Services, Inc.</ORG></AFF></AU>
<INTRO>Markup enables the various parts and features of a
given set of content to be distinguished and named...</INTRO>
<SECT level="1"><HEAD>Overview</HEAD>
<SECT level="2"><HEAD>What is Markup?</HEAD>
<IDXENT>Markup<SUBENT>editorial</SUBENT></IDXENT>
<KW>markup</KW><KW>tags</KW><KW>codes</KW><KW>coding</KW>
<PARA>At the most basic level, markup can be thought of as the
tags and codes embedded in a given set of content that
delineate and describe the component parts of that content.
An editor marks up a <LINKXMPL>paper manuscript</LINKXMPL>...
</PARA></SECT></SECT></CHAPTER>
```

Primer složenijeg DTD-a

- DTD za prethodni XML dokument

```
<!ELEMENT CHAPTER (TITLE, AU+, INTRO, SECT)>
<!ATTLIST CHAPTER id ID #REQUIRED>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT AU (NAME, AFF)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT AFF (ROLE, ORG)>
<!ELEMENT ROLE (#PCDATA)>
<!ELEMENT ORG (#PCDATA)>
<!ENTITY % emph.grp "IT | B | BI | CSC">
<!ELEMENT INTRO (#PCDATA | %emph.grp;)*>
<!ELEMENT SECT (HEAD, (PARA | SECT)+)>
<!ATTLIST SECT level (1|2|3|4) #REQUIRED>
<!ELEMENT HEAD (#PCDATA|IT)*>
<!ELEMENT PARA (#PCDATA|LINKXMPL|%emph.grp;)*>
<!ELEMENT LINKXMPL (#PCDATA)>
<!ELEMENT IT (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT BI (#PCDATA)>
<!ELEMENT CSC (#PCDATA)>
```

Povezivanje XML dokumenta sa DTD-om

- tzv. prolog XML dokumenta sadrži
 - XML deklaraciju
 - deklaraciju tipa dokumenta koja povezuje dokument sa DTD-om
- opšti oblik deklaracije tipa dokumenta

```
<!DOCTYPE koren (PUBLIC "pubid" | SYSTEM) "sysid" [
<!-- interne deklaracije --> ]>
```
- primer (za HTML DTD)

```
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Povezivanje XML dokumenta sa DTD-om

- primer sa dodatnom internom deklaracijom

```
<!DOCTYPE html PUBLIC  
"-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"  
[  
    <!ENTITY slika "images/mypic.gif" GIF>  
]>
```

Povezivanje DTD-ova

- uslovne sekciјe INCLUDE i IGNORE

- sekција INCLUDE укључује део DTD-а

- секција IGNORE искључује део DTD-а

- општи облик

```
<! [IGNORE [DTD section]]>
<! [INCLUDE [DTD section ]]>
```

- пример

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >
<! [%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>]]>
<! [%final;[
<!ELEMENT book (title, body, supplements?)>]]>
```

XML Namespaces

Razlozi za uvođenje

- svako može da definiše sopstvenu XML gramatiku
- ukoliko želimo da koristimo različite gramatike može doći do poklapanja imena

Razlozi za uvođenje

- primer: podaci o osobi, gde je biografija formatirana po XHTML-u

```
<?xml version="1.0" encoding="utf-8"?>
<person>
  <name>
    <title>Sir</title>
    <first>John</first>
    <middle>Fitzgerald Johansen</middle>
    <last>Doe</last>
  </name>
  <position>Vice President of Marketing</position>
  <resume>
    <html>
      <head><title>Resume of John Doe</title></head>
      <body>
        <h1>John Doe</h1>
        <p>John's a great guy, you know?</p>
      </body>
    </html>
  </resume>
</person>
```

dva elementa title sa različitim značenjem

Razlozi za uvođenje

- šta je problem?
 - dva elementa `title` se formalno ne razlikuju
 - aplikacije nemaju način da prepoznaju razliku
 - pretraga po elementu `title` može biti dvosmislena

Razlozi za uvođenje

- šta može biti rešenje?
 - učiniti da nazivi svih elemenata budu jedinstveni
 - možemo svakom elementu dodati prefiks
 - zavisno od toga gde su elementi definisani, prefiks će u primeru biti
 - pers (za elemente koji služe za opis osobe)
 - XHTML (za elemente iz XHTML formata)

Razlozi za uvođenje

- prethodni primer sa prefiksima

```
<?xml version="1.0" encoding="utf-8"?>
<pers:person>
  <pers:name>
    <pers:title>Sir</pers:title>
    <pers:first>John</pers:first>
    <pers:middle>Fitzgerald Johansen</pers:middle>
    <pers:last>Doe</pers:last>
  </pers:name>
  <pers:position>VP of Marketing</pers:position>
  <pers:resume>
    <xhtml:html>
      <xhtml:head>
        <xhtml:title>Resume of John Doe</xhtml:title>
      </xhtml:head>
      <xhtml:body>
        <xhtml:h1>John Doe</xhtml:h1>
        <xhtml:p>John's a great guy, you know?</xhtml:p>
      </xhtml:body>
    </xhtml:html>
  </pers:resume>
</pers:person>
```

sada nema
kolizije imena

Pojam prostora imena

- skup elemenata koji imaju isti prefiks nazvaćemo prostor imena (*namespace*)
- u prethodnom primeru, biće dva prostora imena:
 - pers
 - xhtml

Pojam prostora imena

- prostor imena \neq tip dokumenta
 - prostor imena je skup elemenata
 - jedan tip dokumenta može uključivati elemente iz više prostora
 - jedan element može biti korišćen u više tipova dokumenata

Identifikacija prostora imena

- koncept prefiksa nije dovoljno robustan
 - više ljudi može da se odluči da koristi prefiks pers
 - morala bi postojati organizacija koja se bavi administracijom prefiksa
 - slično kao administriranje Internet domena
- umesto da se formira novi sistem za administraciju prefiksa, može da se koristi postojeći - Internet

Identifikacija prostora imena

- prostori imena identifikuju se nazivom
- naziv je niz znakova u URI formatu
- URI - Uniform Resource Identifier

Uniform Resource Identifier (URI)

- resurs je sve što ima identitet
 - fizički objekti (npr. knjiga)
 - digitalni objekti (fajlovi)
 - apstraktni pojmovi (stanje saobraćaja u Novom Sadu)
- resurs može biti dostupan putem Interneta (npr. fajl), ali i ne mora (npr. autor fajla)
- URI standard, IETF RFC 2396, opisuje identifikatore resursa
- URI može biti formiran kao
 - URN
 - URL

Uniform Resource Locator (URL)

- standard definisan u IETF RFC 1738
- predstavlja podatke koji se mogu upotrebiti za dobavljanje resursa
- URL format - protokol:putanja-do-resursa
- primeri

`http://www.ns.ac.yu/stara/fakulteti.htm
mailto:xyz@gmail.com`

Universal Resource Name (URN)

- IETF RFC 2141
- definiše trajno ime resursa koje ne zavisi od njegove lokacije
- URN format: urn:nid:nss
 - urn: fiksno
 - nid: namespace identifier
 - nss: namespace specific string
- primer - recimo da se JMBG brojevi koriste kao URN
 - nid bi mogao da glasi rs . jmbg
 - nss bi bio JMBG broj konkretnе osobe
 - urn:rs . jmbg : 3107006805027

Identifikacija prostora imena

- zašto se u praksi za identifikaciju prostora imena koriste URL?
 - lakše je napraviti jedinstveno ime (firma već poseduje svoj Internet domen)
 - iako se ne može sprečiti zloupotreba (da jedna firma koristi domen druge kada formira URL za identifikaciju prostora imena), izbegavaju se slučajne kolizije
 - DTD koji definiše elemente iz datog prostora imena se može zaista i postaviti na lokaciju na koju pokazuje URL - na taj način definicije elemenata postaju javne i odmah dostupne
- identifikator prostora imena ne mora da nosi nikakvo značenje!

Identifikacija prostora imena

- ako se koristi URL format za identifikator, uvek se koristi puna adresa, dakle nešto kao
`http://www.mojafirma.com/primer`
- XML aplikacije ne smatraju jednakim sledeća imena:
`http://www.mojafirma.com/primer`
`http://mojafirma.com/primer`
`www.mojafirma.com/primer`

Identifikacija elemenata u prostoru imena

- primer: elementi koji opisuju osobu pripadaju namespace-u sa imenom `http://www.ftn.ns.ac.yu/dtds/person.dtd`
- puno ime elementa `name` iz tog prostora glasilo bi `http://www.ftn.ns.ac.yu/dtds/person.dtd:name`
 - ovo nije validno XML ime
 - vrlo je nezgrapno za pisanje

Identifikacija elemenata u prostoru imena

- elementi se identifikuju **kvalifikovanim imenima** (*qualified name*, *QName*)
- kvalifikovano ime se sastoji iz dva dela
 - lokalno ime
 - namespace
- u XML dokumentima se, umesto naziva prostora imena, koristi (kraći, XML-dozvoljeni) prefiks
- prefiks mora da se poveže sa svojim prostorom imena
- ime prefiksa više nije bitno
 - može biti bilo koje XML-dozvoljeno ime

Identifikacija elemenata u prostoru imena

- primer

person pripada prostoru
označenim prefiksom pers

```
<pers:person  
xmlns:pers="http://www.ftn.ns.ac.yu/dtds/person.dtd">
```

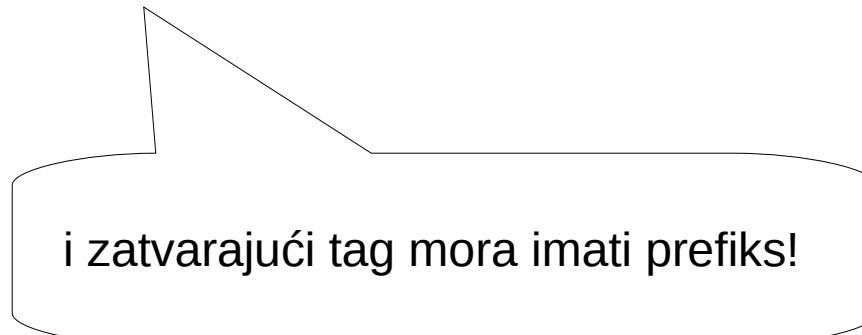
prefiks pers zamenjuje URL
http://www.ftn.ns.ac.yu/dtds/person.dtd

prefiks sada ne služi za identifikaciju prostora imena, već samo
da poveže elemente sa URI identifikatorom!

Identifikacija elemenata u prostoru imena

- primer

```
<pers:person
    xmlns:pers="http://www.ftn.ns.ac.yu/dtds/person.dtd">
    <pers:name>
        <pers:title>Sir</pers:title>
        <pers:first>John</pers:first>
        <pers:last>Doe</pers:last>
    </pers:name>
    ...
</pers:person>
```



i zatvarajući tag mora imati prefiks!

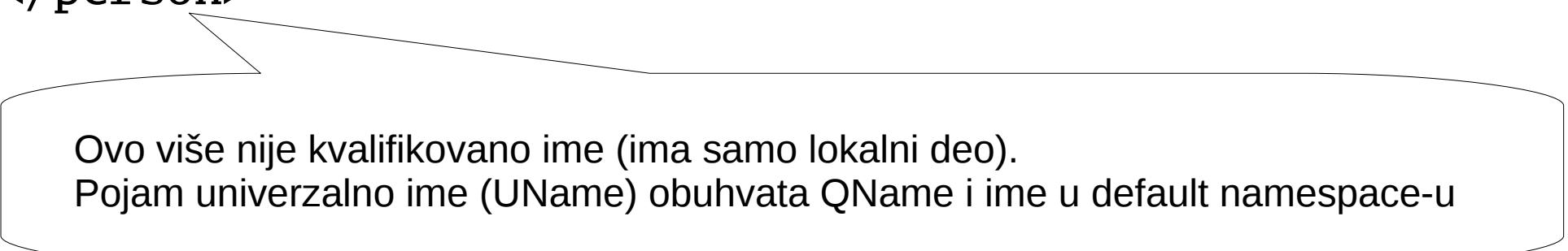
Identifikacija elemenata u prostoru imena

```
<?xml version="1.0" encoding="utf-8"?>
<pers:person
    xmlns:pers="http://www.ftn.ns.ac.yu/dtds/person.dtd"
    xmlns:xhtml="http://www.w3.org/1999/xhtml">
    <pers:name>
        <pers:title>Sir</pers:title>
        <pers:first>John</pers:first>
        <pers:last>Doe</pers:last>
    </pers:name>
    <pers:position>VP of Marketing</pers:position>
    <pers:resume>
        <xhtml:html>
            <xhtml:head>
                <xhtml:title>Resume of John Doe</xhtml:title>
            </xhtml:head>
            <xhtml:body>
                <xhtml:h1>John Doe</xhtml:h1>
                <xhtml:p>John's a great guy, you know?</xhtml:p>
            </xhtml:body>
        </xhtml:html>
    </pers:resume>
</pers:person>
```

Default namespace

- šta raditi sa starim XML dokumentima gde se ne koriste namespace prefksi?
- postoji pojam **podrazumevanog** (default) prostora imena
 - namespace kao i svaki drugi
 - njegovi elementi se navode bez prefiksa
- primer

```
<person xmlns="http://www.ftn.ns.ac.yu/dtds/person.dtd">
    <name>
        <title>Sir</title>
    </name>
</person>
```



Ovo više nije kvalifikovano ime (ima samo lokalni deo).

Pojam univerzalno ime (QName) obuhvata QName i ime u default namespace-u

Deklarisanje prostora imena

- u jednom elementu može se deklarisati više prostora imena

```
<pers:person
    xmlns:pers="http://www.ftn.ns.ac.yu/dtds/person.dtd"
    xmlns:xhtml="http://www.w3.org/1999/xhtml">
```
- ne moraju svi prostori imena biti deklarisani u korenskom elementu
- prefiksi imaju značenje samo *u okviru elementa* u kome su definisani!

Deklarisanje prostora imena

- primer

```
<?xml version="1.0" encoding="utf-8"?>
<pers:person
    xmlns:pers="http://www.ftn.ns.ac.yu/dtds/person.dtd">
    <pers:name>
        <pers:title>Sir</pers:title>
        <pers:first>John</pers:first>
        <pers:last>Doe</pers:last>
    </pers:name>
    <pers:position>VP of Marketing</pers:position>
    <pers:resume>
        <xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml">
            <xhtml:head>
                <xhtml:title>Resume of John Doe</xhtml:title>
            </xhtml:head>
            <xhtml:body>
                <xhtml:h1>John Doe</xhtml:h1>
                <xhtml:p>John's a great guy, you know?</xhtml:p>
            </xhtml:body>
        </xhtml:html>
    </pers:resume>
</pers:person>
```

Deklarisanje prostora imena

- primer 2 - koristi se default namespace

```
<?xml version="1.0" encoding="utf-8"?>
<person xmlns="http://www.ftn.ns.ac.yu/dtds/person.dtd">
    <name>
        <title>Sir</title>
        <first>John</first>
        <last>Doe</last>
    </name>
    <position>VP of Marketing</position>
    <resume>
        <xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml">
            <xhtml:head>
                <xhtml:title>Resume of John Doe</xhtml:title>
            </xhtml:head>
            <xhtml:body>
                <xhtml:h1>John Doe</xhtml:h1>
                <xhtml:p>John's a great guy, you know?</xhtml:p>
            </xhtml:body>
        </xhtml:html>
    </resume>
</person>
```

Deklarisanje prostora imena

- primer 3 - redefiniše se default namespace

```
<?xml version="1.0" encoding="utf-8"?>
<person xmlns="http://www.ftn.ns.ac.yu/dtds/person.dtd">
    <name>
        <title>Sir</title>
        <first>John</first>
        <last>Doe</last>
    </name>
    <position>VP of Marketing</position>
    <resume>
        <html xmlns="http://www.w3.org/1999/xhtml">
            <head>
                <title>Resume of John Doe</title>
            </head>
            <body>
                <h1>John Doe</h1>
                <p>John's a great guy, you know?</p>
            </body>
        </html>
    </resume>
</person>
```

Deklarisanje prostora imena

- isključenje default namespace-a

```
<p xmlns="http://www.w3.org/1999/xhtml">  
    I've worked with <name xmlns="">Jane Doe</name>  
    for over a <em>year</em> now.  
</p>
```

Deklarisanje prostora imena

- koji način je bolji?
 1. uvek koristiti prefiks
 2. koristiti jedan namespace kao default, a ostale navoditi sa prefiksom
 3. svuda redefinisati default namespace
- za XML aplikacije je svejedno
- možda nam je bitno da znamo i koji je prefiks korišćen
 - tada ćemo koristiti varijantu 1

Identifikacija atributa u prostoru imena

- primer
`<pers:name id="i25">`
- da li atribut nasleđuje namespace od svog elementa?
 - u principu ne, ali
 - može, ako aplikacija to tako interpretira (i ako namespace nije eksplicitno naveden)
- primer: formalno, sledeća dva atributa nisu ista
`<pers:name id="i25">`
`<pers:name pers:id="i25">`
- ali mogu biti ako aplikacija to tako interpretira

Namespaces & DTDs

- DTD je formalno definisan pre XML Namespaces standarda
- DTD ne podržava potpuno namespaces
- elementi u DTD-u se moraju definisati zajedno sa prefiksom
 - prefiks se mora zadati unapred i mora biti fiksan
- namespace deklaracije se u DTD-u tretiraju kao atributi
 - nije svejedno gde deklarišemo namespace

XML Schema

Šta nije dobro kod DTD-a?

- neuobičajena, ne-XML sintaksa
- slaba podrška za XML prostore imena
- nema tipizacije podataka, naročito za sadržaj elementa
 - nije moguće definisati datumska ili numerička polja
- ograničena proširivost
- ograničene mogućnosti za opisivanje strukture podataka
 - ne može se nametnuti broj podelemenata bez nametanja redosleda
 - ne može se nametnuti redosled i broj podelemenata kada se koristi mešani sadržaj

Šta je šema?

- šema je dokument koji opisuje strukturu drugih dokumenata
- prevazilazi mane DTD-a
 - šeme se pišu kao XML dokumenti, a ne korišćenjem sintakse nasleđene od SGML-a
 - potpuna podrška za rad sa XML namespaces
 - validacija tekstualnog sadržaja na bazi ugrađenih ili novodefinisanih tipova podataka (datum, ceo broj, itd)
 - kreiranje i jednostavno ponovno korišćenje modela sadržaja

Da li je DTD i dalje potreban?

- šeme nemaju funkcionalnost rada sa entitetima
- DTD može biti ugrađen direktno u dokument koji opisuje, dok
- šeme moraju uvek biti smeštene u posebne fajlove
- softver i dalje potpuno podržava rad sa DTD-ima

Šema standardi

- postoji više standarda za šema jezike
 - W3C XML Schema
 - RELAX NG
 - Schematron
- XML Schema je standard koga propisuje W3C
 - najrašireniji
 - najmoćniji
 - najkomplikovaniji

W3C XML Schema

- kreirana od strane W3C XML Schema Group, na osnovu mnogih predloga
- nema patenata ili drugih restrikcija nad intelektualnom svojinom
- definisana kroz sledeće dokumente
 - XML Schema Part 1: Structures
<http://www.w3.org/TR/xmlschema-1/>
 - XML Schema Part 2: Datatypes
<http://www.w3.org/TR/xmlschema-2/>

Primer šeme

- primer XML dokumenta

greeting.xml

```
<?xml version="1.0"?>
<GREETING>
    Hello XML!
</GREETING>
```

Primer šeme

- šema za prethodni dokument

greeting.xsd

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="GREETING" type="xsd:string"/>
</xsd:schema>
```

korenski element W3C XML Schema dokumenta je xsd:schema

Povezivanje šeme i dokumenta

- standardni XML Instance namespace
 - identifikator je
`http://www.w3.org/2001/XMLSchema-instance`
 - uobičajeni prefiks je xsi
 - u njemu su definisana četiri atributa
 - `xsi:type`
 - `xsi:nil`
 - `xsi:schemaLocation`
 - `xsi:noNamespaceSchemaLocation`

Povezivanje šeme i dokumenta

- atribut xsi:schemaLocation
 - označava lokaciju šeme koja definiše strukturu datog dokumenta
 - vrednost atributa čine dva URI-ja
 - identifikator namespace-a
 - adresa fajla
- atribut xsi:noNamespaceSchemaLocation
 - lokacija šeme za elemente koji pripadaju default namespace-u

Povezivanje šeme i dokumenta

- primer 1 - elementi su u default namespace-u

greeting.xml

```
<?xml version="1.0"?>
<GREETING
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="greeting.xsd">
    Hello XML!
</GREETING>
```

greeting.xsd

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="GREETING" type="xsd:string"/>
</xsd:schema>
```

Povezivanje šeme i dokumenta

- primer 2 - elementi nisu u default namespace-u

greeting.xml

```
<?xml version="1.0"?>
<GREETING
    xmlns="http://www.ftn.ns.ac.yu/greetings"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="greeting.xsd">
    Hello XML!
</GREETING>
```

greeting.xsd

```
<?xml version="1.0"?>
<xsd:schema
    targetNamespace="http://www.ftn.ns.ac.yu/greetings"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="GREETING" type="xsd:string"/>
</xsd:schema>
```

Validacija dokumenata

- primer neispravnog dokumenta

```
<?xml version="1.0"?>
<GREETING
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="greeting.xsd">
    <P>Hello XML!</P>
</GREETING>
```

Tipizacija podataka u W3C XML Schema

- prosti tipovi podataka
 - ne mogu imati podelemente ili atributе
 - primeri
 - <datum_racuna>2007-06-01</datum_racuna>
 - <kolicina>17</kolicina>
- složeni tipovi podataka
 - mogu imati podelemente ili atributе
 - primeri
 - <name><first>John</first><last>Doe</last></name>
 - <person id="i73">...</person>

Četiri osnovna elementa šeme

- `xsd:element` deklariše element i dodeljuje mu tip
- `xsd:attribute` deklariše atribut i dodeljuje mu tip
- `xsd:simpleType` definiše novi prosti tip
- `xsd:complexType` definiše novi složeni tip

Deklaracija elementa

- tip elementa može da se navede u okviru njegove deklaracije
 - „anonimni tip“
- kao tip elementa može da se navede neki postojeći tip
 - „imenovani tip“

Deklaracija elementa

- primer XML dokumenta

```
<?xml version="1.0"?>
<SONG
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="simple_song.xsd">
    <TITLE>Hot Cop</TITLE>
    <COMPOSER>Jacques Morali</COMPOSER>
    <COMPOSER>Henri Belolo</COMPOSER>
    <COMPOSER>Victor Willis</COMPOSER>
    <PRODUCER>Jacques Morali</PRODUCER>
    <PUBLISHER>PolyGram Records</PUBLISHER>
    <LENGTH>6:20</LENGTH>
    <YEAR>1978</YEAR>
    <ARTIST>Village People</ARTIST>
</SONG>
```

Deklaracija elementa

- Šema za prethodni dokument sa anonimnim tipom

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="TITLE" type="xsd:string"/>
        <xsd:element name="COMPOSER" type="xsd:string"
                     maxOccurs="unbounded"/>
        <xsd:element name="PRODUCER" type="xsd:string"
                     minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="PUBLISHER" type="xsd:string"
                     minOccurs="0"/>
        <xsd:element name="LENGTH" type="xsd:duration"/>
        <xsd:element name="YEAR" type="xsd:gYear"/>
        <xsd:element name="ARTIST" type="xsd:string"
                     maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Deklaracija elementa

- Šema za prethodni dokument sa imenovanim tipom

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE" type="xsd:string"/>
      <xsd:element name="COMPOSER" type="xsd:string"
                    maxOccurs="unbounded"/>
      <xsd:element name="PRODUCER" type="xsd:string"
                    minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="PUBLISHER" type="xsd:string"
                    minOccurs="0"/>
      <xsd:element name="LENGTH" type="xsd:duration"/>
      <xsd:element name="YEAR" type="xsd:gYear"/>
      <xsd:element name="ARTIST" type="xsd:string"
                    maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Deklaracija elementa

- anonimni vs imenovani tip
 - ako očekujemo da će se isti tip pojavljivati na više mesta, isplati se definisati ga na jednom mestu - koristićemo imenovani tip
 - ako ne želimo da se definicija tipa vidi kao nešto što je dostupno korisniku šeme - koristićemo anonimni tip

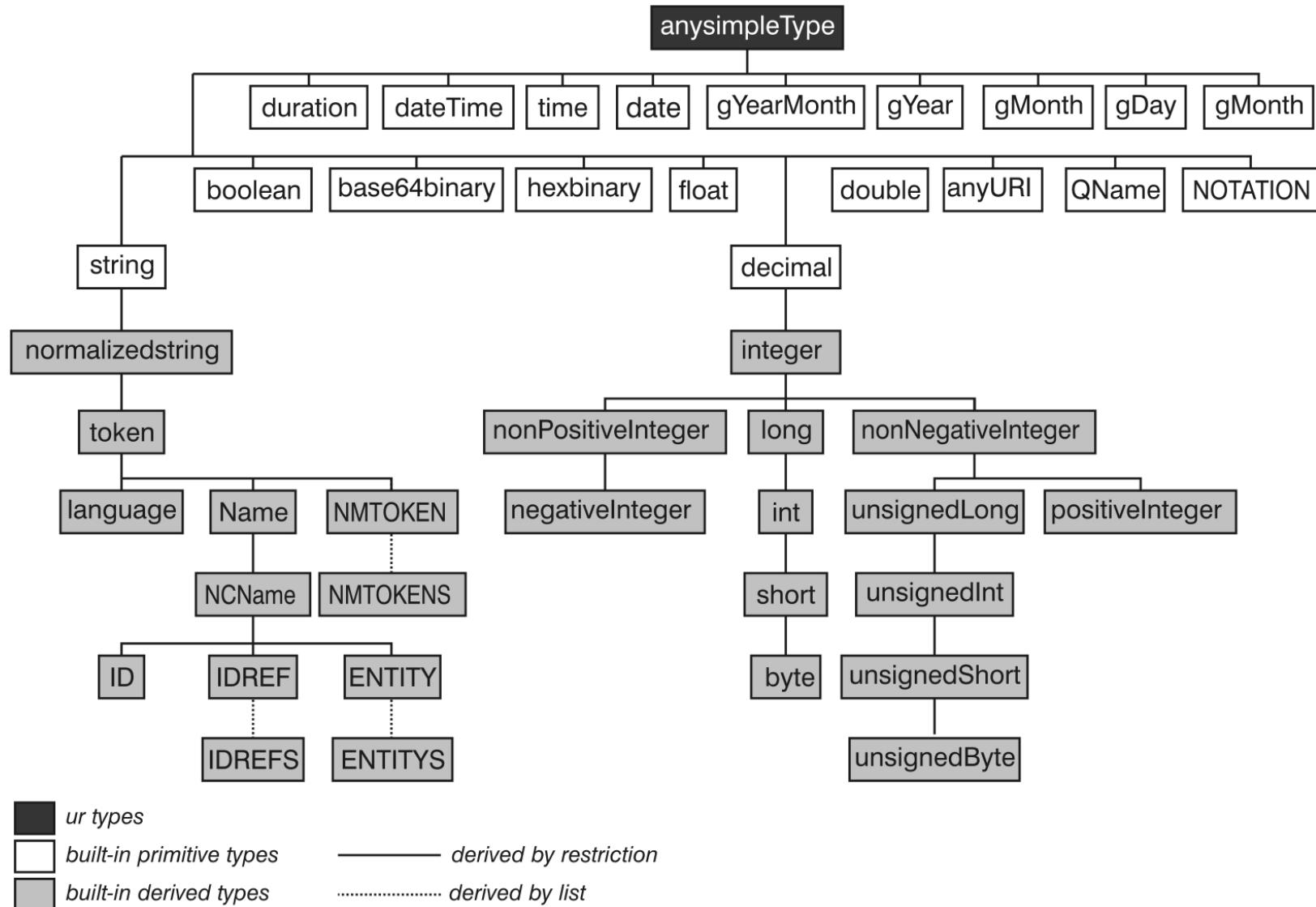
Deklaracija tipa

- svaka deklaracija novog tipa se oslanja na neki od postojećih (već definisanih) tipova
- kada definišemo prvi tip, koji tipovi mogu da se koriste?
- tipovi korišćeni u prethodnim primerima
 - xsd:string
 - xsd:year
 - xsd:duration

Ugrađeni tipovi

- definisani u XML Schema Part 2: Datatypes

- logički
- tekstualni
- URI
- numerički
- vremenski
- XML tipovi



Ugrađeni numerički tipovi

float	IEEE 754 32-bitni broj u pokretnom zarezu
double	IEEE 754 64-bitni broj u pokretnom zarezu
decimal	decimalni broj proizvoljne preciznosti
integer	proizvoljno velik ceo broj
nonPositiveInteger	ceo broj manji ili jednak nuli
negativeInteger	ceo broj manji od nule
long	64-bitni ceo broj
int	32-bitni ceo broj
short	16-bitni ceo broj
byte	8-bitni ceo broj
nonNegativeInteger	ceo broj veći ili jednak nuli
unsignedLong	neoznačeni 64-bitni ceo broj
unsignedInt	neoznačeni 32-bitni ceo broj
unsignedShort	neoznačeni 16-bitni ceo broj
unsignedByte	neoznačeni 8-bitni ceo broj
positiveInteger	ceo broj veći od nule

Ugrađeni vremenski tipovi

dateTime	trenutak u vremenu
gMonth	mesec u godini
gYear	godina
gMonthDay	datum nevezan za godinu
gDay	dan u mesecu
duration	dužina vremenskog intervala, bez fiksnog početka ili kraja
date	konkretan dan u vremenu
time	konkretno vreme u toku dana

Ostali ugrađeni tipovi

string	
normalizedString	string koji ne sadrži TAB, CR, LF
token	string čija normalizovana vrednost ne sadrži više uzastopnih razmaka ili razmaka na krajevima stringa
boolean	logički tip
anyURI	relativna ili apsolutna internet adresa
hexBinary	binarni podaci zapisani u heksadecimalnom obliku
base64Binary	binarni podaci zapisani u Base64 obliku

Deklaracija novog prostog tipa

- novi prosti tipovi se definišu na osnovu postojećih - ugrađenih
- mehanizam definisanja novog tipa
 - restrikcija: navođenje ograničenja na vrednost novog tipa
 - lista: definisanje konačne liste mogućih vrednosti
 - unija: kombinovanje vrednosti više različitih osnovnih tipova
- <xsd:simpleType name="ImeTipa">...</simpleType>

Deklaracija novog prostog tipa

- restrikcija
 - skup mogućih vrednosti novog tipa je podskup vrednosti osnovnog tipa
 - primer: `integer` i `nonNegativeInteger`
 - ograničenja se navode pomoću 12 *facet-a*
 - `length`
 - `minLength`
 - `maxLength`
 - `pattern`
 - `enumeration`
 - `whiteSpace`
 - `maxInclusive`
 - `maxExclusive`
 - `minInclusive`
 - `minExclusive`
 - `totalDigits`
 - `fractionDigits`

Deklaracija novog prostog tipa

- restrikcija
 - primer: string dužine 1-255 znakova (minLength i maxLength)

```
<xsd:simpleType name="Str255">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="1"/>
    <xsd:maxLength value="255"/>
  </xsd:restriction>
</xsd:simpleType>
```

Deklaracija novog prostog tipa

- restrikcija
 - primer: nabranje mogućih vrednosti (enumeration)

```
<xsd:simpleType name="Tseason">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="spring"/>
    <xsd:enumeration value="summer"/>
    <xsd:enumeration value="autumn"/>
    <xsd:enumeration value="winter"/>
  </xsd:restriction>
</xsd:simpleType>
```

Deklaracija novog prostog tipa

- restrikcija
 - primer: definisanje tipa za novčane iznose (pattern)

```
<xsd:simpleType name="money">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\p{Sc}\d+(\.\d\d)?"/>
    <!--
        Regularni izraz:
        \p{Sc}      oznaka valute $, €, £, ¥.
        \d          decimalna cifra
        \d+         jedna ili više decimalnih cifara
        \.          znak tačka
        (\.\d\d)?  nula ili jedan string oblika .35
    -->
  </xsd:restriction>
</xsd:simpleType>
```

Deklaracija novog prostog tipa

- lista
 - tip koji definišemo predstavljaće listu elemenata nekog tipa
 - elementi liste su razdvojeni razmacima
 - primer

```
<element name="ages">
    <simpleType>
        <list itemType="positiveInteger"/>
    </simpleType>
</element>

<ages>4 9 12</ages>
```

Deklaracija novog prostog tipa

- unija
 - vrednost novog tipa može biti vrednost bilo kog tipa koji je član unije
 - primer

```
<element name="timePeriod">
    <simpleType>
        <union memberTypes="season xsd:date"/>
    </simpleType>
```

```
</element>
```

```
<timePeriod>summer</timePeriod>
<timePeriod>2007-05-25</timePeriod>
```

Deklaracija novog složenog tipa

- deklaracija složenog tipa koristi jedan od modela sadržaja
 - sekvenca podelemenata
 - izbor između podelemenata
 - neuređeni skup podelemenata
- deklaracija složenog tipa može da sadrži deklaracije atributa
- ako je deklaracija složenog tipa prazna, definisan je prazan tip

Deklaracija novog složenog tipa

- sekvenca - sequence
 - sekvenca podelemenata ili drugih modela sadržaja
 - primer

```
<xsd:complexType name="TName">
  <xsd:sequence minOccurs="1" maxOccurs="1">
    <xsd:element name="first" type="xsd:string"/>
    <xsd:element name="middle" type="xsd:string"/>
    <xsd:element name="last" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="title" type="xsd:string"/>
</xsd:complexType>
```

Deklaracija novog složenog tipa

- sekvenca - sequence
 - primer: sekvenca u sekvenci

```
<xsd:complexType name="TName">
  <xsd:sequence minOccurs="1" maxOccurs="1">
    <xsd:element name="first" type="xsd:string"/>
    <xsd:element name="middle" type="xsd:string"/>
    <xsd:element name="last" type="xsd:string"/>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="year" type="xsd:year"/>
    </xsd:sequence>
  </xsd:sequence>
  <xsd:attribute name="title" type="xsd:string"/>
</xsd:complexType>
```

Deklaracija novog složenog tipa

- sekvenca - sequence
 - redosled podelemenata je bitan
 - broj pojavljivanja podelemenata je bitan
 - ako se ne navede, podrazumeva se `minOccurs="1"` i `maxOccurs="1"`
 - cela sekvenca može da se ponavlja
 - ima atribute `minOccurs` i `maxOccurs`

Deklaracija novog složenog tipa

- izbor - choice
 - bira se jedan od podelemenata ili modela sadržaja
 - primer

```
<xsd:complexType name="TName">
  <xsd:choice minOccurs="1" maxOccurs="1">
    <xsd:element name="first" type="xsd:string"/>
    <xsd:element name="middle" type="xsd:string"/>
    <xsd:element name="last" type="xsd:string"/>
  </xsd:choice>
  <xsd:attribute name="title" type="xsd:string"/>
</xsd:complexType>
```

Deklaracija novog složenog tipa

- neuređeni skup podelementa - all
 - u sadržaju se mogu pojaviti svi podelementi u bilo kom redosledu
 - podelement se može pojaviti tačno jednom
 - ne može da sadrži sekvene ili izbore
 - ne može da bude uključen u sekvene ili izbore
 - primer

```
<xsd:complexType name="TName">
    <xsd:all minOccurs="0 ili 1" maxOccurs="1">
        <xsd:element name="first" type="xsd:string"/>
        <xsd:element name="middle" type="xsd:string"/>
        <xsd:element name="last" type="xsd:string"/>
    </xsd:all>
    <xsd:attribute name="title" type="xsd:string"/>
</xsd:complexType>
```

Novi složeni tip sa mešanim sadržajem

- tekst može biti isprepletan sa podelementima
- atribut `mixed` ima vrednost `true`
- primer

```
<element name="p">
  <complexType mixed="true">
    <choice minOccurs="0" maxOccurs="unbounded">
      <element name="b" type="string"/>
      <element name="i" type="string"/>
    </choice>
  </complexType>
</element>
```

```
<p>Without question, <b>Ramses II</b> is the
<i>greatest</i> pharaoh <b>of all time!</b></p>
```

Deklaracija atributa

- atributi moraju biti prostog tipa
- deklaracija

```
<attribute
    name="ime atributa"
    type="ime tipa"
    form="qualified/unqualified"
    use="optional/required/prohibited"
    default="podrazumevana vrednost"
    fixed="fiksna vrednost">
```

Deklaracija atributa

- primeri

```
<xsd:attribute name="title" type="xsd:string"/>
```

```
<xsd:attribute name="title" type="xsd:string"  
default="Mr." use="optional"/>
```

```
<xsd:attribute name="season" type="Tseason"/>
```

Primer šeme

- primer dokumenta

```
<?xml version="1.0"?>
<SONG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ftn.ns.ac.yu/songs"
      xsi:schemaLocation="http://www.ftn.ns.ac.yu/songs
                           songs.xsd">
  <TITLE>Hot Cop</TITLE>
  <PHOTO ALT="Victor Willis" WIDTH="100" HEIGHT="200"/>
  <COMPOSER>
    <NAME><GIVEN>Jacques</GIVEN><FAMILY>Morali</FAMILY></NAME>
  </COMPOSER>
  <PRODUCER>
    <NAME><GIVEN>Jacques</GIVEN><FAMILY>Morali</FAMILY></NAME>
  </PRODUCER>
  <PUBLISHER>PolyGram Records</PUBLISHER>
  <YEAR>1978</YEAR>
  <ARTIST>Village People</ARTIST>
  <LENGTH>6:20</LENGTH>
</SONG>
```

Primer šeme

- tip koji predstavlja osobu

```
<xsd:complexType name="PersonType">
  <xsd:sequence>
    <xsd:element name="NAME">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="GIVEN" type="xsd:string"/>
          <xsd:element name="FAMILY" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Primer šeme

- tip koji predstavlja pesmu

```
<xsd:complexType name="SongType">
  <xsd:sequence>
    <xsd:element name="TITLE" type="xsd:string"/>
    <xsd:element name="COMPOSER" type="xsd:string"
                  maxOccurs="unbounded"/>
    <xsd:element name="PRODUCER" type="xsd:string"
                  minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="PUBLISHER" type="xsd:string"
                  minOccurs="0"/>
    <xsd:element name="LENGTH" type="xsd:duration"/>
    <xsd:element name="YEAR" type="xsd:gYear"/>
    <xsd:element name="ARTIST" type="xsd:string"
                  maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Primer šeme

- definicija korenskog elementa

```
<xsd:element name="SONG" type="SongType"/>
```

Primer šeme

- celo šema

```
<xsd:schema  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
    targetNamespace="http://www.ftn.ns.ac.yu/songs">  
<xsd:complexType name="PersonType">  
    ...  
</xsd:complexType>  
<xsd:complexType name="SongType">  
    ...  
</xsd:complexType>  
<xsd:element name="SONG" type="SongType"/>  
</xsd:schema>
```

XML PARSERI

Novi Sad, 2017

Miroslav Zarić

ŠTA JE XML PARSIRANJE

- Podrazumeva prolazak kroz XML dokument sa ciljem:
 - pristupa određenim (ili svim) podacima u njemu
 - modifikacije podataka (ne dozvoljavaju svi parseri)
- XML parser - program koji putem odgovarajućih (poželjno jednostavnih) funkcija omogućava laku manipulaciju sadržajima u XML dokumentima

VRSTE PARSERA

- **SAX Parser** - obrađuje dokument na osnovu događaja - za određeni tip događaja piše se odgovarajući *handler*. Ne učitava odjednom ceo dokument u memoriju (ne pravi njegovu memorijsku predstavu).
- **DOM Parser** - obrada XML dokumenta se obavlja tako što se ceo dokument učita u memoriju - formiranjem strukture hijerarhijskog stabla. Sva manipulacija se zatim obavlja nad ovom memorijskom reprezentacijom dokumenta.

VRSTE PARSERA

- **StAX Parser** - Koristi slične principe kao SAX, bolje optimizovan.
- **XPath Parser** - Obradu dokumenta vrši na osnovu odgovarajućih izraza - koristi se intenzivno u kombinaciji sa XSLT.

SAX PARSER

- **SAX Parser** - obrađuje dokument na osnovu događaja - za određeni tip događaja piše se odgovarajući *handler*. Ne učitava odjednom ceo dokument u memoriju (ne pravi njegovu memorijsku predstavu).
- **Osnovne karakteristike:**
 - Stream orijentisan interfejs za obradu XML-a. Aplikacija koja koristi ovaj tip parsera dobija notifikaciju svaki put kada se procesira element, atribut...
 - Čita dokument počevši od root elementa i generiše događaje svaki put kada prepozna neki “token” u dobro formiranom dokumentu.
 - obrada se vrši isključivo po redosledu pojavljivanja u dokumentu
 - obaveštava aplikaciju o tipu tokena na koji je trenutno naišao
 - da bi obezbedila smislenu obradu aplikacija mora parseru registrovati svoj *event handler*
 - kada se određeni token detektuje, trigeruje se odgovarajući registrovani *event handler* za dati tip tokena

SAX PARSER

- **ContentHandler** interfejs (specificira *callback* metode koje parser koristi da notifikuje aplikaciju o tipu sadržaja na koji je naišao):
 - void **startDocument()**
 - void **endDocument()**
 - void **startElement(String uri, String localName, String qName, Attributes atts)**
 - void **endElement(String uri, String localName, String qName)**
 - void **characters(char[] ch, int start, int length)**
 - void **ignorableWhitespace(char[] ch, int start, int length)**
 - void **processingInstruction(String target, String data)**
 - void **setDocumentLocator(Locator locator)**) - (postavlja lokator kojim je moguće pratiti poziciju u dokumentu)
 - void **skippedEntity(String name)**
 - void **startPrefixMapping(String prefix, String uri)**
 - void **endPrefixMapping(String prefix)**

SAX PARSER

- **Kada ga je pogodno koristiti:**
 - Kada je moguće obradu vršiti linearно - redom kojim se elementi pojavljuju od vrha ka dnu dokumenta
 - Dokument nije “duboko ugnježden”
 - Kada se procesira veliki dokument (ili veliki broj dokumenata u paraleli) - u tom slučaju bi formiranje modela dokumenta bilo resursno prezahtevno (DOM implementacije koriste i do 10 bajta memorije za 1 bajt XML podataka)
 - Kada problem koji se rešava zahteva korišćenje samo dela dokumenta
 - Podaci su dostupni čim ih parser “prepozna” - idealno za obradu podataka koji se “streamuju”

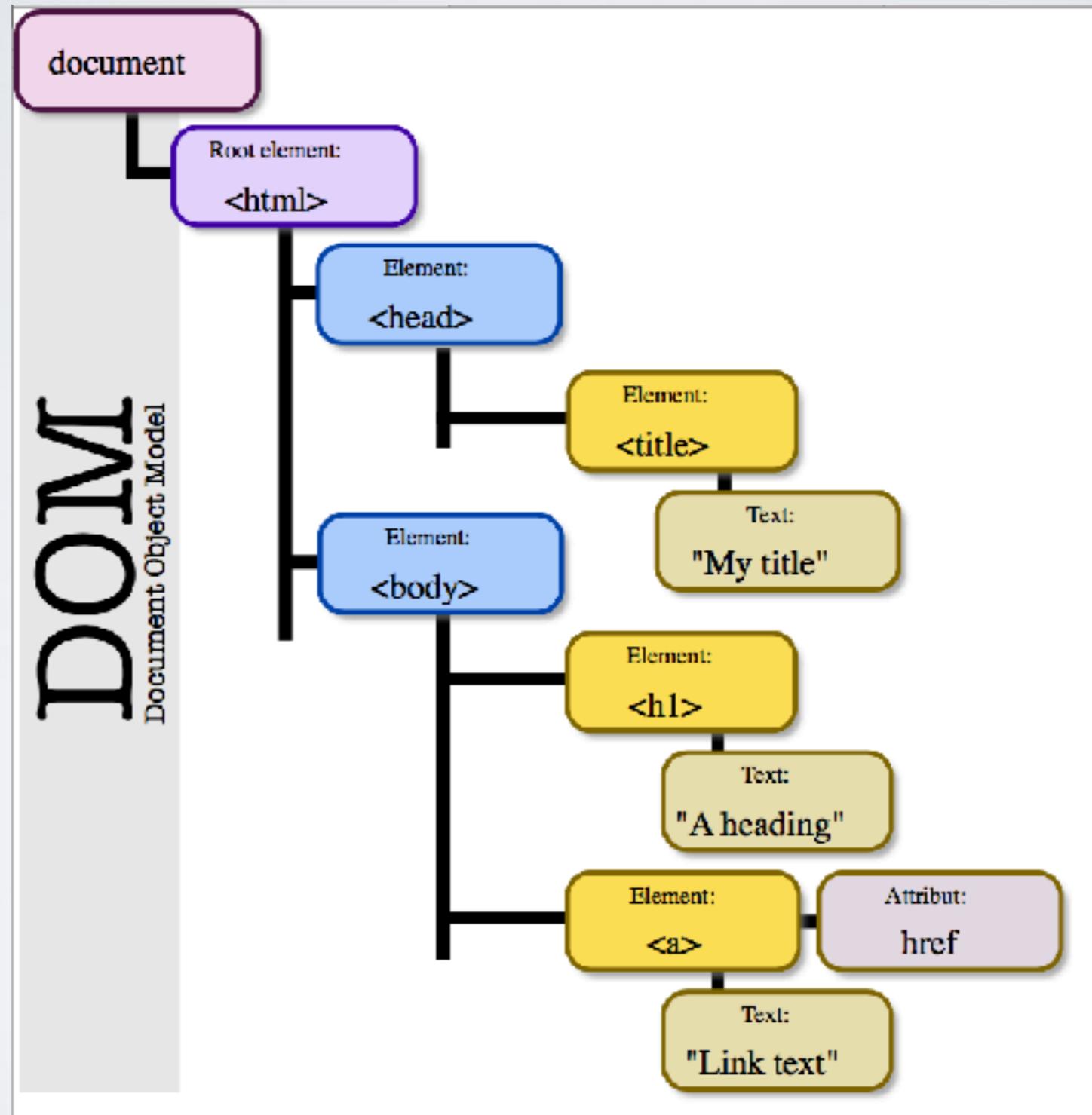
SAX PARSER

- **Nedostaci:**
 - Nije moguć direktni pristup određenom elementu u strukturi (jer se sve procesira linearno)
 - Ukoliko je potrebno imati naknadni pristup podacima koje je parser već “prošao” neophodno je pisati dodatni sopstveni kod koji bi takve podatke negde privremeno čuvao

DOM

- **Document Object Model (DOM)** - oficijelna preporuka modela XML dokumenta World Wide Web Consortium-a (W3C).
- Definiše interfejse koji aplikacijama omogućavaju pristup i manipulaciju sadržaja XML dokumenata. Parseri koji implementiraju DOM implementiraju sledeće interfejse:
 - **Node** - osnovni tip u DOM.
 - **Element** - predstavlja elemente XML-a.
 - **Attr** - reprezentuje atribute elementa XML-a.
 - **Text** - predstavlja tekstualni sadržaj elementa ili tributa.
 - **Document** - predstavlja ceo dokument.

DOM



DOM PARSERI

- **Osnovne karakteristike:**

- Po završenom parsiranju aplikaciji je na raspolaganju objektna reprezentacija sadržaja dokumenta formirana kao hijerarhijsko stablo (DOM).
- DOM obezbeđuje veliki broj funkcija za pristup i manipulaciju nad DOM struktrom i sadržajem.

DOM PARSERI

- **Osnovna prednost:** DOM je de facto standardni interfejs za manipulaciju strukturom XML dokumenata. Kod napisan da koristi jedan DOM parser trebao bi da radi i ako mu se “podmetne” druga implementacija usklađena sa W3C preporukama.
- **Glavni nedostatak:** nepogodan je za jako velike dokumente jer formiranje *in-memory* strukture može biti resursno prezahtevno

DOM PARSERI

- **Kada ga je pogodno koristiti?**
 - Kada je neophodno dobro sagledati i za obradu dobro poznavati strukturu celog dokumenta
 - Kada je neophodno reorganizovati dokument (premeštati, dodavati, sortirati elemente...)
 - Kada postoji velika verovatnoća da će nam određene informacije iz dokumenta biti potrebne više puta tokom obrade.

Simple API for XML - SAX

Parsiranje XML dokumenata

- svako bi mogao da napravi parser za XML dokumente, ali...
- bolje da koristimo gotove parsere
 - testirani
 - vođeno je računa o svim detaljima XML specifikacije
 - optimizovani
 - obavljaju i validaciju

Parsiranje XML dokumenata

- postoji više kvalitetnih parsera za XML
- koji koristiti?
 - vezivanje za neki specifičan parser nije poželjno
 - šta ako razvoj tog parsera prestane?
- standardni API za XML parsere
 - možemo koristiti svaki parser koji poštuje ovaj API
 - ako zamenimo parser ne menjamo naš kod koji ga koristi
- → Simple API for XML (SAX)

Simple API for XML

- SAX je definisan za različite jezike
 - Java
 - C++
 - ...
- specifikacije SAX-a za svaki jezik su međusobno vrlo slične

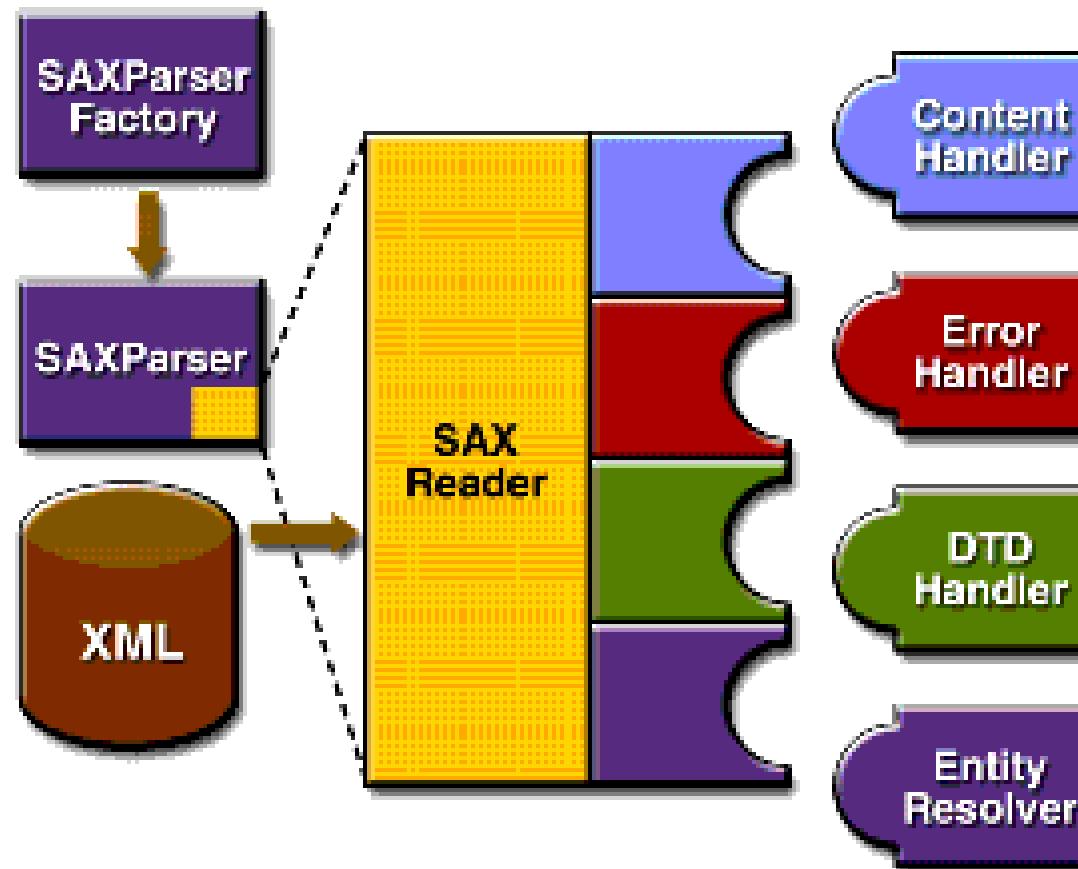
Simple API for XML

- nije W3C standard
 - razvijen kroz saradnju na xml-dev mailing listi
 - jeste de facto standard
- verzije
 - 1998: v 1.0
 - 2000: v 2.0: namespace podrška, property mehanizam

SAX koncept

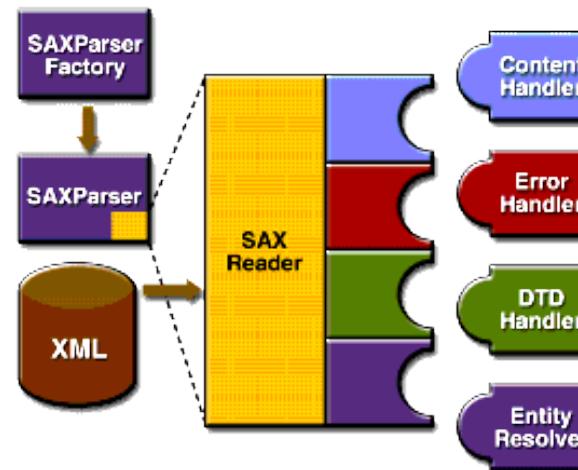
- parsiranje pomoću SAX-a je *event-driven*
- parser tokom parsiranja „generiše događaje“
 - npr. počeo dokument, počeo element, završio se element, ...
- naš kôd je zadužen da obradi „događaj“
 - pišemo tzv. *handlere*
 - njih poziva parser (*callback*)

SAX koncept



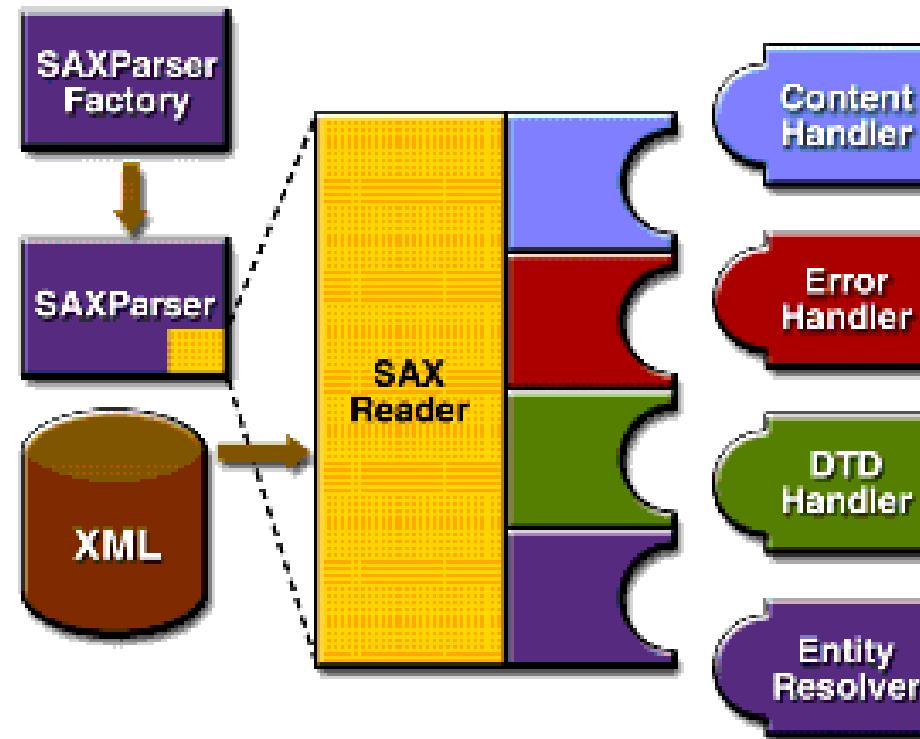
SAX API

- SAXParserFactory – kreira instance parsera
 - određene sistemskim promenjivama
- SAXParser – interfejs sa nekoliko parse() metoda
 - tipični parametri: izvor XML podataka i DefaultHandler objekat obrađuje događaje
- SAXReader – nalazi se unutar SAXParser-a
 - ukoliko je potrebno da se preciznije konfiguriše pozivamo metod getXMLReader(). SAXReader poziva event handlere



SAX handleri

- ContentHandler - interfejs sa *callback* metodama za obradu događaja vezanih za sadržaj dokumenta
 - startDocument(), endDocument(), startElement(), endElement(), characters(), processingInstructions()



SAX handleri

- ErrorHandler - obrada događaja vezanih za greške tokom parsiranja
 - error(), fatalError(), warning()
- DTDHandler - obrada događaja vezanih za parsiranje DTD-a
 - notationDecl(), unparsedEntityDecl()
- EntityResolver - pribavljanje eksternih entiteta
 - resolveEntity()

SAX handleri

- da bismo upotrebili parser trebalo bi implementirati 4 interfejsa!
 - naporno
 - nisu nam potrebne sve funkcije (DTD, eksterni entiteti...)
- pomoćna klasa: DefaultHandler
 - implementira sva 4 interfejsa praznim metodama
 - dovoljno je naslediti DefaultHandler i redefinisati samo one *callback* metode koje su nam stvarno potrebne

SAX API i Java

- javax.xml.parsers
 - SAXParserFactory
 - SAXParser
- org.xml.sax
 - ContentHandler
 - ErrorHandler
 - DTDHandler
 - EntityResolver
- org.xml.sax.helpers
 - DefaultHandler

Echo.java

- napravićemo program koji čita XML dokument i ispisuje ga na konzolu
- klasa Echo naslediće DefaultHandler
 - kao reakciju na svaki događaj ispisatićemo ono što nam parser isporuči
- dodaćemo main()
 - inicijalizacija parsera
 - pokretanje parsera

Echo.java

- main() - inicializacija i pokretanje parsera

```
//instanca nase klase - event handler  
DefaultHandler handler = new Echo();  
  
// factory kreira parsere  
SAXParserFactory factory = SAXParserFactory.newInstance();  
  
// instanciramo jedan parser  
SAXParser saxParser = factory.newSAXParser();  
  
// pokrenemo parsiranje  
saxParser.parse(new File(args[0]), handler);
```

Echo.java

- dve utility metode
 - nl() - ispisuje znak za novi red u skladu sa operativnim sistemom

```
private void nl() throws SAXException {  
    String lineEnd = System.getProperty("line.separator");  
    try {  
        out.write(lineEnd);  
    } catch (IOException e) {  
        throw new SAXException("I/O error", e);  
    }  
}
```

Echo.java

- dve utility metode
 - emit() - ispisuje string na konzolu

```
private void emit(String s) throws SAXException {  
    try {  
        out.write(s);  
        out.flush();  
    } catch (IOException e) {  
        throw new SAXException("I/O error", e);  
    }  
}
```

startDocument() i endDocument()

```
public void startDocument() throws SAXException {  
    emit("<?xml version='1.0' encoding='UTF-8'?>");  
    nl();  
}  
  
public void endDocument() throws SAXException {  
    try {  
        nl();  
        out.flush();  
    } catch (IOException e) {  
        throw new SAXException("I/O error", e);  
    }  
}
```

startElement()

```
public void startElement(String namespaceURI,  
                         String sName,           ...bez prefiksa  
                         String qName,          ...sa prefiksom  
                         Attributes attrs)  
throws SAXException {  
  
    if ("\".equals(sName))  
        sName = qName;  
    emit("<" + sName);  
    if (attrs != null) {  
        for (int i = 0; i < attrs.getLength(); i++) {  
            String aName = attrs.getLocalName(i);  
            if ("\".equals(aName))  
                aName = attrs.getQName(i);  
            emit(" ");  
            emit(aName + "=\""+ attrs.getValue(i) +"\"");  
        }  
    }  
    emit(">");  
}
```

endElement()

```
public void endElement(String namespaceURI,
                      String sName,
                      String qName)
throws SAXException {
    if ("".equals(sName))
        sName = qName;
    emit("</" + sName + ">");
```

```
}
```

characters()

```
StringBuffer textBuffer;

public void characters(char buf[], int offset, int len) throws SAXException {
    String s = new String(buf, offset, len);
    if (textBuffer == null) {
        textBuffer = new StringBuffer(s);
    } else {
        textBuffer.append(s);
    }
}

private void echoText() throws SAXException {
    if (textBuffer == null)
        return;
    emit(s.toString());
    textBuffer = null;
}
```

characters() može biti pozvana više puta za jedan
kontinualni tekstualni sadržaj iz dokumenta!

characters()

- ispisujemo sadržaj svaki put kad počinje i završava element

```
public void startElement(...) throws SAXException {  
    echoText();  
    ...  
}  
  
public void endElement(...) throws SAXException {  
    echoText();  
    ...  
}
```

→primer1

Analiza rezultata

- očuvani whitespace
 - parser nema DTD na raspolaganju - prepostavlja da svaki element ima mixed content model - pa čuva sve whitespace karaktere u sadržaju elemenata
 - nije očuvan između atributa!
- komentari su ignorisani
 - trebalo bi implementirati LexicalHandler
- prazni elementi
 - <item/> se događajima predstavlja kao <item></item>

Whitespace test

- dodajemo ispis opisa svakog događaja

→primer2

ignorableWhitespace()

- služi da parser „javi“ postojanje nebitnog whitespace sadržaja
 - (parser mora da zna koji whitespace je nebitan, treba mu DTD)

setDocumentLocator(Locator loc)

- Locator - objekat koji sadrži podatke o lokaciji na kojoj se desio događaj
 - Locator je validan samo u trenucima poziva event-handling metoda

processingInstruction()

- parametri
 - target - aplikacija koja treba da procesira instrukciju
 - data - podaci za obradu

```
public void processingInstruction(String target, String data)
    throws SAXException {
    nl();
    emit("PROCESS: ");
    emit("<?" + target + " " + data + "?>");
}
```

→primer3

Obrada grešaka

- SAXException
 - može da sadrži i izuzetak koji se desio u event-handleru
- SAXParseException
 - nasleđuje SAXException
 - sadrži informacije o redu u kome je greška

Obrada grešaka

- tri nivoa grešaka
 - warning(SAXParseException e)
 - npr. element definisan dva puta u DTD-u (jeste greška, ali ne pravi probleme)
 - error(SAXParseException e)
 - npr. dokument nije validan
 - fatalError(SAXParseException e)
 - npr. dokument nije dobro formiran

→primer4

CDATA sekcije

- reference na entitete parser automatski zamenjuje njihovim vrednostima
- CDATA sekcije parser automatski pretvara u nizove znakova
 - za Echo primer trebalo bi još zameniti znakove &, <, > referencama na entitete &; < > u characters()

Parsiranje uz validaciju

- dokument može da poseduje svoj DTD ili referencu na spoljašnji DTD
- ako DTD postoji
 - nevalidirajući parser ignoriše whitespace tamo gde je to moguće
 - ako nam ipak trebaju, koristimo ignorableWhitespace() događaj
 - validirajući radi sve to plus validaciju

Parsiranje uz validaciju

- kreiranje parsera
 - izbor fabrike - pomoću sistemskog property-ja
javax.xml.parsers.SAXParserFactory=com.foo.MyFactory
 - da li je parser validirajući
factory.setValidating(true)
 - da li parser vodi računa o namespace-ovima
factory.setNamespaceAware(true)

→primer5

Validacija pomoću šeme

1) napraviti validirajući parser koji radi sa namespaces

- factory.setValidating(true)
- factory.setNamespaceAware(true)

2) definisati koji šema-jezik se koristi tako što se postavi vrednost za sledeći property parsera (mi koristimo XML Schema)

- property se zove
<http://java.sun.com/xml/jaxp/properties/schemaLanguage>
- a vrednost mu je
<http://www.w3.org/2001/XMLSchema>
- saxParser.setProperty(
 "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
 "http://www.w3.org/2001/XMLSchema");

ime propertija je u URL formatu

Validacija pomoću šeme

- povezivanje dokumenta sa šemom - na dva načina
 - šema deklaracija u dokumentu
 - programsko povezivanje (bezbednije...), tada se šema deklaracije u dokumentu ne uzimaju u obzir

```
saxParser.setProperty(  
    "http://java.sun.com/xml/jaxp/properties/schemaSource",  
    new File(schemaSource));
```

Upozorenja

- prilikom rada sa DTD-om
 - duplirane definicije u DTD-u
 - korišćenje elemenata koji nisu definisani u DTD-u
 - deklarisanje atributa za nepostojeće elemente
- prilikom rada parsera
 - parser je validirajući, a dokument nema <!DOCTYPE ...>
 - refenciranje nedefinisanih parametarskih entiteta kada nema validacije (kada ima validacije, to je greška)

Obrada leksičkih događaja

- opcionalna
- implementiramo LexicalHandler
 - comment()
 - startEntity()
 - startCDATA()
 - endEntity()
 - endCDATA()
 - startDTD()
 - endDTD()
- naglasimo parseru da se koristi i LexicalHandler

```
xmlReader.setProperty("http://xml.org/sax/properties/lexical-handler", handler);
```

→primer6

Obrada DTD događaja

- kada se nađe na neparsirani entitet

```
<!ENTITY myEntity SYSTEM "URL..." NDATA gif>
<!NOTATION gif SYSTEM "URL...">
```

- za obradu ovih događaja: DTDHandler

- unparsedEntityDecl()
 - notationDecl()

EntityResolver

- za pribavljanje entiteta
`resolveEntity(String publicId, String systemId)`

→primer7

Document Object Model - DOM

Document Object Model

- W3C standard
- standard za objektno-orientisani reprezentaciju dokumenata sa hijerarhijskom strukturom - stablo
- varijante za različite jezike
 - Java
 - C++
 - JavaScript
 - ...
- varijante za različite dokumente
 - XML
 - HTML

Document Object Model

- koncept rada sa DOM parserom razlikuje se od SAX-a
- rezultat parsiranja je stablo objekata
 - stalno prisutno u memoriji, za ceo dokument
 - redosled obrade elemenata ne mora se poklapati sa redosledom u dokumentu
 - možemo serijalizovati stablo nazad u XML
 - problem sa zauzećem memorije
 - za velike dokumente
 - za serverske aplikacije (puno istovremeno procesiranih dokumenata)

DOM + Java

- rad sa XML dokumentima preko objektne reprezentacije stabla dokumenta
- od Jave 1.4
- paketi
 - javax.xml.*
 - org.w3c.dom.*
 - org.xml.sax.*

...DOM parseri se po pravilu oslanjaju na SAX parsere

Parsiranje dokumenta

- fabrika parsera

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();
```

- napravimo parser

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

- parsiramo dokument

```
Document doc = builder.parse(xml);
```

→primer1

DocumentBuilderFactory

- kreira parsere
- moguće je podešavati osobina generisanih parsera
 - setValidating()
 - setNamespaceAware()
 - setIgnoringElementContentWhitespace()
 - setIgnoringComments()
 - setCoalescing() - da li se CDATA čvorovi pretvaraju u text čvorove i spajaju sa okolnim text čvorovima
 - setAttribute() - postavljanje dodatnih atributa (npr. validacija pomoću šeme)

DocumentBuilder

- parsira postojeći XML dokument u DOM stablo
 - parse(File f)
 - parse(InputSource is)
- kreira novo DOM stablo
 - newDocument()
- obrada grešaka
 - setErrorHandler() - ako se ne postavi, koristi se default
 - obrada grešaka istovetna kao i kod SAX parsera

Kretanje kroz DOM stablo

- DOM stablo reprezentuju interfejsi
 - Node - apstraktni čvor stabla, nasleđuju ga
 - Document - čitav XML dokument
 - Element
 - Attr
 - Text - tekstualni sadržaj
 - Comment
 - CDATASection
 - Entity
 - Notation
 - ProcessingInstruction

Kretanje kroz DOM stablo

- Node interfejs
 - pristup karakteristikama samog čvora
 - getnodeName
 - getNodeType
 - getNodeValue
 - getAttributes

Kretanje kroz DOM stablo

- Node interfejs
 - pristup povezanim čvorovima
 - getChildNodes
 - getFirstChild
 - getLastChild
 - getParentNode
 - getNextSibling

Kretanje kroz DOM stablo

- Node interfejs
 - metode za ažuriranje
 - appendChild
 - insertBefore
 - replaceChild
 - setNodeValue
 - removeChild

Kretanje kroz DOM stablo

- Document interfejs
 - getDocumentElement - pristup korenskom elementu dokumenta
 - getElementById - pristup elementu sa datim ID-em
 - getElementsByTagName - lista elemenata sa datim nazivom
 - sve metode vraćaju samo Element čvorove!
 - primer2 (konstruiše JTree od DOM-a)
 - primer3 (filtrira prikazane čvorove)
 - primer4 (prikazuje i sadržaj elemenata)

Kretanje kroz DOM stablo

- tipovi čvorova

Interface	nodeName	nodeValue	attributes
Attr	name of attribute	value of attribute	null
CDATASection	"#cdata-section"	content of the CDATA Section	null
Comment	"#comment"	content of the comment	null
Document	"#document"	null	null
DocumentFragment	"#document-fragment"	null	null
DocumentType	document type name	null	null
Element	tag name	null	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Text	"#text"	content of the text node	null

Kreiranje DOM stabla

- kreiranje praznog dokumenta

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();
```

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

```
Document doc = builder.newDocument();
```

Kreiranje DOM stabla

- primer - programski kreiramo stablo koje odgovara dokumentu

```
<team name="Sacramento">
  <player num="21">
    Vlade Divac
  </player>
</team>
```

Kreiranje DOM stabla

- kreiramo root element

```
// kreiramo root element
Element root = (Element)doc.createElement("team");

// kreiramo cvor za atribut i dodajemo ga root-u
root.setAttributeNode(doc.createAttribute("name"));

// postavljamo vrednost atributa
root.setAttribute("name", "Sacramento");

// jedino dete Document čvora je root
doc.appendChild(root);
```

Kreiranje DOM stabla

- dodavanje elementa

```
Element player = (Element)doc.createElement("player");
player.setAttributeNode(doc.createAttribute("num"));
player.setAttribute("num", "21");
root.appendChild(player);
```

Kreiranje DOM stabla

- dodavanje tekstualnog čvora

```
Text name = (Text)doc.createTextNode("Vlade Divac");  
player.appendChild(name);
```

→primer5 (programski se kreira dokument)
→primer6 (dokument se i normalizuje)

Validacija pomoću šeme

- slično kao i za SAX

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
  
factory.setNamespaceAware(true);  
factory.setValidating(true);  
  
factory.setAttribute(  
    "http://java.sun.com/xml/jaxp/properties/schemaLanguage",  
    "http://www.w3.org/2001/XMLSchema");
```

Povezivanje dokumenta sa šemom

- na dva načina
 - šema deklaracija u dokumentu
 - programsko povezivanje

```
factory.setAttribute(  
    "http://java.sun.com/xml/jaxp/properties/schemaSource",  
    new File("mySchema.xsd"));
```

Povezivanje dokumenta sa šemom

- dokument sa više namespace-ova

```
factory.setAttribute(  
    "http://java.sun.com/xml/jaxp/properties/schemaSource",  
    new String[] {  
        "nikon.xsd",  
        "pentax.xsd",  
        "olympus.xsd"});
```

StAX

XML i web servisi

Stevan Gostojić

Fakultet tehničkih nauka, Novi Sad

5. april 2016.

Agenda

1 XML Parsers

2 StAX

XML Parsers

- Moguće je napraviti XML parser koristeći tehnike sa kojima ste se upoznali na programskim prevodiocima (za dobro formirane ili validne XML dokumente)
- U većini slučajeva je bolje koristiti postojeće XML parsere
 - manje posla
 - manja verovatnoća greške

XML Parsers

Postoji nekoliko programskih modela za parsiranje XML dokumenata

- streaming parseri
 - pull (StAX parseri)
 - push (SAX parseri)
- DOM parseri
- XSL-T parseri

Streaming vs. DOM parseri

	streaming	DOM
+	koristi malo resursa (procesor, memo- rija)	slučajan pristup, lakši za korišćenje
-	sekvencijalni pristup, teži za korišćenje	koristi puno resursa (procesor, memo- rija)

Table 1: Streaming vs. DOM parseri

Push i pull parseri

- Push parseri implementiraju programski model u kome XML parser šalje (gura) podatke programu koji ga koristi nailazeći na elemente XML informacionog skupa (elemente, attribute, tekst, itd.)
- Pull parseri implementiraju programski model u kome programi koji ih koriste pozivaju metode XML parsera (vuku podatke) kada im treba element XML informacionog skupa (element, atribut, tekst, itd.)

StAX vs. SAX vs. DOM vs. XSLT parseri

	StAX	SAX	DOM	TrAX
Tip	pull, streaming	push, streaming	in memory tree	XSLT rules
lakoća upotrebe	lako	srednje	lako	srednje
XPath	ne	ne	da	da
CPU/RAM	niski zahtevi	niski zahtevi	visoki za- htevi	visoki za- htevi
jednosmerni	da	da	ne	ne
čita XML	da	da	da	da
piše XML	da	ne	da	da
CRUD	ne	ne	da	ne

Table 2: StAX vs. SAX vs. DOM vs. XSLT parseri

Agenda

1 XML Parsers

2 StAX

StAX API

- StAX API implementira pull koncept što znači da program traži podatke od StAX parsra onda kada su mu potrebni
- Parser čita XML dokument od početka do kraju i prepoznaje elemente XML informacionog skupa (tokene koji čine dobro formiran XML dokument)
- StAX API može da se koristi i za čitanje i za pisanje XML dokumenata (za razliku od SAX API-a)
- XML dokumente može da parsira korišćenjem iteratora (iterira kroz listu događaja da bi dobio podatke) ili kurzora (podatke dobija preko pokazivača na XML čvorove)

Cursor API vs. Iterator API

- Cursor API koristi kurzor koji se kreće od početka do kraja XML dokumenta i pokazuje na elemente XML informacionog skupa
- Iterator API predstavlja XML dokument kao niz diskretnih događaja (koji odgovaraju XML informacionom skupu) koji se mogu obraditi

StAX API

Klasa/Interfejs	Opis
XMLEventReader	obezbeđuje iterator nad događajima koji se koriste za parsiranje XML dokumenta
XMLEventWriter	sadrži metode za pravljenje događaja
XMLStreamReader	obezbeđuje kurSOR koji se koristi za parsiranje XML dokumenta
XMLStreamWriter	sadrži metode za pravljenje čvorova

Table 3: Bitni elementi StAX API-a

XMLEventReader

Klasa	Opis
XMLEvent nextEvent()	vraća sledeći događaj
public boolean hasNext()	proverava da li postoji još događaja

Table 4: Metode XMLEventReader klase

XMLEvent

Klasa	Opis
StartDocument	sadrži xml version, encoding i standalone svojstva
StartElement	sadrži lokalno ime, deklaracije prostora imena i atributе
EndElement	kraj elementa
Characters	sadrži tekstualni sadrža (uključujući i CDATA, ignorable whitespace)

Table 5: Naslednice XMLEvent klase

XMLEventReader

```
1 XMLEventReader eventReader = factory.createXMLEventReader(reader);
2
3 while(eventReader.hasNext()) {
4     XMLEvent event = eventReader.nextEvent();
5
6     switch(event.getEventType()) {
7         case XMLStreamConstants.START_ELEMENT:
8             StartElement startElement = event.asStartElement();
9             // ...
10            break;
11        case XMLStreamConstants.CHARACTERS:
12            Characters characters = event.asCharacters();
13            // ...
14            break;
15        case XMLStreamConstants.END_ELEMENT:
16            EndElement endElement = event.asEndElement();
17            // ...
18            break;
19        default:
20    }
21 }
```

XMLEventWriter

Klasa	Opis
add(Event event)	dodaje događaj koji predstavlja element

Table 6: Metode XMLEventWriter klase

XMLEventWriter

```
1 XMLEventWriter eventWriter = factory.createXMLEventWriter(writer);
2
3 XMLEvent event = eventFactory.createStartDocument();
4 eventWriter.add(event);
5
6 event = eventFactory.createStartElement("prefix", "namespaceUri", "localName");
7 eventWriter.add(event);
8
9 event = eventFactory.createNamespace("prefix", "namespaceUri");
10 eventWriter.add(event);
11
12 event = eventFactory.createAttribute("attributeName", "attributeValue");
13 eventWriter.add(event);
14
15 event = eventFactory.createCharacters("text");
16 eventWriter.add(event);
17
18 event = eventFactory.createEndElement("prefix", "namespaceUri", "localName");
19 eventWriter.add(event);
20
21 eventWriter.flush();
22 eventWriter.close();
```

XMLStreamReader

Metoda	Opis
<code>int next()</code>	vraća sledeći čvor
<code>boolean hasNext()</code>	proverava da li postoji još čvorova
<code>String getText()</code>	vraća tekstualni sadržaj elementa
<code>String getLocalName()</code>	vraća ime elementa
...	...

Table 7: Metode XMLStreamReader klase

XMLStreamReader

```
1 XMLStreamReader streamReader = factory.createXMLStreamReader(reader);
2
3 while(streamReader.hasNext()) {
4     streamReader.next();
5
6     switch(streamReader.getEventType()) {
7         case XMLStreamConstants.START_ELEMENT:
8             // ...
9             break;
10        case XMLStreamConstants.CHARACTERS:
11            // ...
12            break;
13        case XMLStreamConstants.END_ELEMENT:
14            // ...
15            break;
16        default:
17    }
18 }
```

XMLStreamWriter

Metoda	Opis
writeStartElement(String localName)	dodaje početni tag
writeEndElement(String localName)	dodaje krajnji tag
writeAttribute(String localName, String value)	dodaje atribut
...	...

Table 8: Metode XMLStreamWriter klase

XMLStreamWriter

```
1 XMLStreamWriter streamWriter = factory.createXMLStreamWriter(writer);
2
3 streamWriter.writeStartDocument();
4 streamWriter.writeStartElement("localName");
5 streamWriter.writeStartElement("localName");
6 streamWriter.writeAttribute("attributeName", "attributeValue");
7 streamWriter.writeCharacters("text");
8 streamWriter.writeEndElement();
9 streamWriter.writeEndElement();
10 streamWriter.writeEndDocument();
11
12 writer.flush();
13 writer.close();
```

Zaključak

- Upoznali smo se sa tipovima XML parsera
- Naučili smo kako se pravi XML parser koji koristi StAX API (kursor ili iterator)

Reference

- StAX API, <https://goo.gl/wfoT0J>

Hvala na pažnji!
Pitanja?

JAXB

XML i web servisi

Stevan Gostojić

Fakultet tehničkih nauka, Novi Sad

12. april 2016.

Agenda

- 1 Java Architecture for XML Binding

JAXB

- JAXB (Java Architecture for XML Binding) je framework za generisanje Java klasa na osnovu DTD ili XML Schema šema (i obrnuto) i iz transformaciju XML dokumenata u graf Java objekata (i obrnuto)
- JAXB je API, dostupna je i referentna implementacija
- JAXB (gotovo da) ne zahteva poznavanje XML-a za programsku obradu (transakcionalih) XML dokumenata

JAXB vs. SAX/StAX/DOM

- JAXB parseri rade na višem nivou apstrakcije od SAX, StAX-a i DOM parsera
- SAX i StAX parseri predstavljaju XML dokumente kao tok generičkih događaja ili čvorova
- DOM parseri predstavljaju XML dokumente kao stabla generičkih čvorova
- JAXB parseri predstavljaju XML dokumente kao Java klase iz određenog domena

book.xml

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <book>
3   <title>Document Engineering</title>
4   <author>Robert J. Glushko and Tim McGrath</author>
5   <publisher>MIT Press</publisher>
6 </book>
```

Test.java

```
1  public static void main(String[] args) {  
2      Book book = new Book(  
3          "Document Engineering",  
4          "Robert J. Glushko and Tim McGrath",  
5          "MIT Press");  
6  }
```

book.xsd

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3      <xsd:element name="book">
4          <xsd:complexType>
5              <xsd:sequence>
6                  <xsd:element name="title" type="xsd:string" />
7                  <xsd:element name="author" type="xsd:string" />
8                  <xsd:element name="publisher" type="xsd:string" />
9              </xsd:sequence>
10             </xsd:complexType>
11         </xsd:element>
12     </xs:schema>
```

Book.java

```
1  public class Book {  
2      public String title;  
3      public String author;  
4      public String publisher;  
5  }
```

SAX/StAX/DOM?

- JAXB se može posmatrati kao jedan način (de)serijalizacije stanja Java objekta
- Koriste se za parsiranje transakcionih dokumenata (koji se relativno lako mapiraju na java objekte)
- Za parsiranje narativnih dokumenata (koji se teško mapiraju na java objekte) koriste se SAX/StAX/DOM parseri

Korišćenje JAXB-a

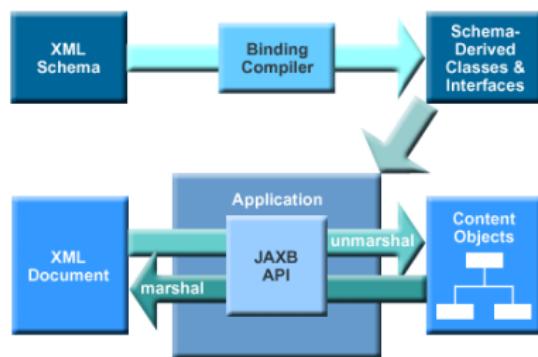


Figure 1: JAXB koraci.

- ① Pomoću binding prevodioca DTD ili XML Schemu prevesti u Java klase (ili obrnuto)
- ② Pomoću Java prevodioca prevesti Java klase (u byte kod)
- ③ Napisati Java program koji transformiše XML dokumente u graf Java objekata (i obrnuto)

JAXB Binding Compiler (xjc)

```
xjc.sh -p com.example -d work books.xsd
```

Parametar	Opis
-p com.example	paket u kome će se nalaziti Java klase
-d work	direktorijum u kome će se nalaziti Java klase (work)
books.xsd	izvorna XML šema

Table 1: JAXB Binding Compiler (xjc)

JAXB Schema Generator (schemagen)

```
schemagen.sh -d work -cp $CLASSPATH Foo.java Bar.java
```

Parametar	Opis
-d work	direktorijum u kome se nalaze Java klase
-cp \$CLASSPATH	classpath
Foo.java Bar.java	izvorne Java klase

Table 2: JAXB Schema Generator (schemagen)

JAXB API

Class/Interface	Opis
JAXBContext	ulazna tačka u JAXB API
Unmarshaller	deserializuje XML dokument u graf objekata (uz opcionu validaciju)
Marshaller	serijalizuje graf objekata u XML dokument

Table 3: Bitni elementi JAXB API-a

Unmarshalling

```
1 // Definiše kontekst (tj. pakete u kojima se nalaze Java klase)
2 JAXBContext context = JAXBContext.newInstance("com.example");
3 Unmarshaller unmarshaller = context.createUnmarshaller();
4
5 // Validira XML dokument u odnosu na šemu
6 unmarshaller.setValidating(true);
7
8 // Deserializuje XML dokument u graf Java objekata
9 Book book = (Book) unmarshaller.unmarshal(new File("book.xml"));
```

Marshalling

```
1 // Pravi graf Java objekata
2 ObjectFactory factory = new ObjectFactory();
3 Book book = (Book) factory.createBook();
4 book.setTitle("Document Engineering");
5 book.setAuthor("Robert J. Glushko and Tim McGrath");
6 book.setPublisher("MIT Press");
```

Marshalling

```
1 // Definiše kontekst (tj. pakete u kojima se nalaze Java klase)
2 JAXBContext context = JAXBContext.newInstance("com.example");
3 Marshaller marshaller = context.createMarshaller();
4
5 // Formatira XML dokument
6 marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
7
8 // Serijalizuje graf Java objekata u XML dokument
9 marshaller.marshal(book, System.out);
```

Pravila povezivanja

XML	Java
prostori imena	paketi
složeni tipovi podataka	klase
prosti tipovi podataka	prosti tipovi podataka ili (ugrađene) klase
atributi, elementi	svojstva

Table 4: Pravila povezivanja.

Prilagođavanje povezivanja

- JAXB koristi podrazumevana pravila povezivanja
- Podrazumevana pravila povezivanja mogu se promeniti anotacijama u XML šemi ili Java klasi

Prilagođavanje povezivanja

```
1 <xsd:element ref="book" minOccurs="0" maxOccurs="unbounded">
2   <xsd:annotation>
3     <xsd:appinfo>
4       <jxb:property name="books"/>
5     </xsd:appinfo>
6   </xsd:annotation>
7 </xsd:element>
```

Anotacije

Annotation	Opis
<globalBindings>	globalna povezivanja
<schemaBindings>	povezivanja na nivou šeme
<property>	povezivanje XML elementa i Java svojstva

Table 5: Bitne JAXB anotacije

Prilagođavanje povezivanja

```
1  @XmlAccessorType(XmlAccessType.FIELD)
2  @XmlType(name="book", propOrder={"title", "author", "publisher"})
3  public class Book {
4
5      @XmlElement(name="title", required=false)
6      private String title;
7
8      @XmlElement(name="author", required=true)
9      private String author;
10
11     @XmlElement(name="publisher", required=true)
12     private String publisher;
13
14     // ...
15
16 }
```

Anotacije

Annotation	Opis
@XmlAccessorType	izbor serijalizacije polja ili svojstava
@XmlType	mapira klasu na XML tip
@XmlElement	polje će biti XML element
@XmlAttribute	polje će biti XML atribut
@XmlValue	polje će biti (prost) sadržaj XML elementa

Table 6: Bitne JAXB anotacije

Zaključak

- Upoznali smo se sa razlikama između JAXB parsera i DOM parsera
- Naučili smo kako se implementira JAXB parser
- Naučili smo kako se prilagođava povezivanje između XML šeme i Java klasa

Reference

- JAXB API, <http://goo.gl/VK5Y>
- A JAXB Tutorial, <https://jaxb.java.net/tutorial>

Hvala na pažnji!
Pitanja?

XPath

Šta je XPath

- jezik za označavanje delova XML dokumenta
- zasniva se na konceptu navigacije kroz stablo dokumenta
- ima biblioteku standardnih funkcija
- neophodan za korišćenje XSLT
- W3C standard

XPath izrazi

- izraz je namenjen za označavanje čvora ili skupa čvorova u dokumentu
- ovi izrazi liče na izraze za rad sa fajl-sistemom
 - fajl-sistem
 - C:\windows\fonts\arial.ttf
 - \windows\fonts\arial.ttf
 - ..\..\windows\fonts\arial.ttf

Standardne XPath funkcije

- XPath sadrži oko 100 ugrađenih funkcija
 - operacije sa stringovima
 - operacije numeničkim vrednostima
 - operacije sa datumom i vremenom
 - manipulacija čvorovima
 - manipulacija sekvencama
 - ...

XPath čvorovi

- XML dokument se, sa stanovišta XPath-a, posmatra kao stablo
- više tipova čvorova
 - element
 - atribut
 - tekst
 - namespace
 - procesna instrukcija
 - komentar
 - dokument (koren)

XPath čvorovi

- primer

```
<?xml version="1.0"?>
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J. K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

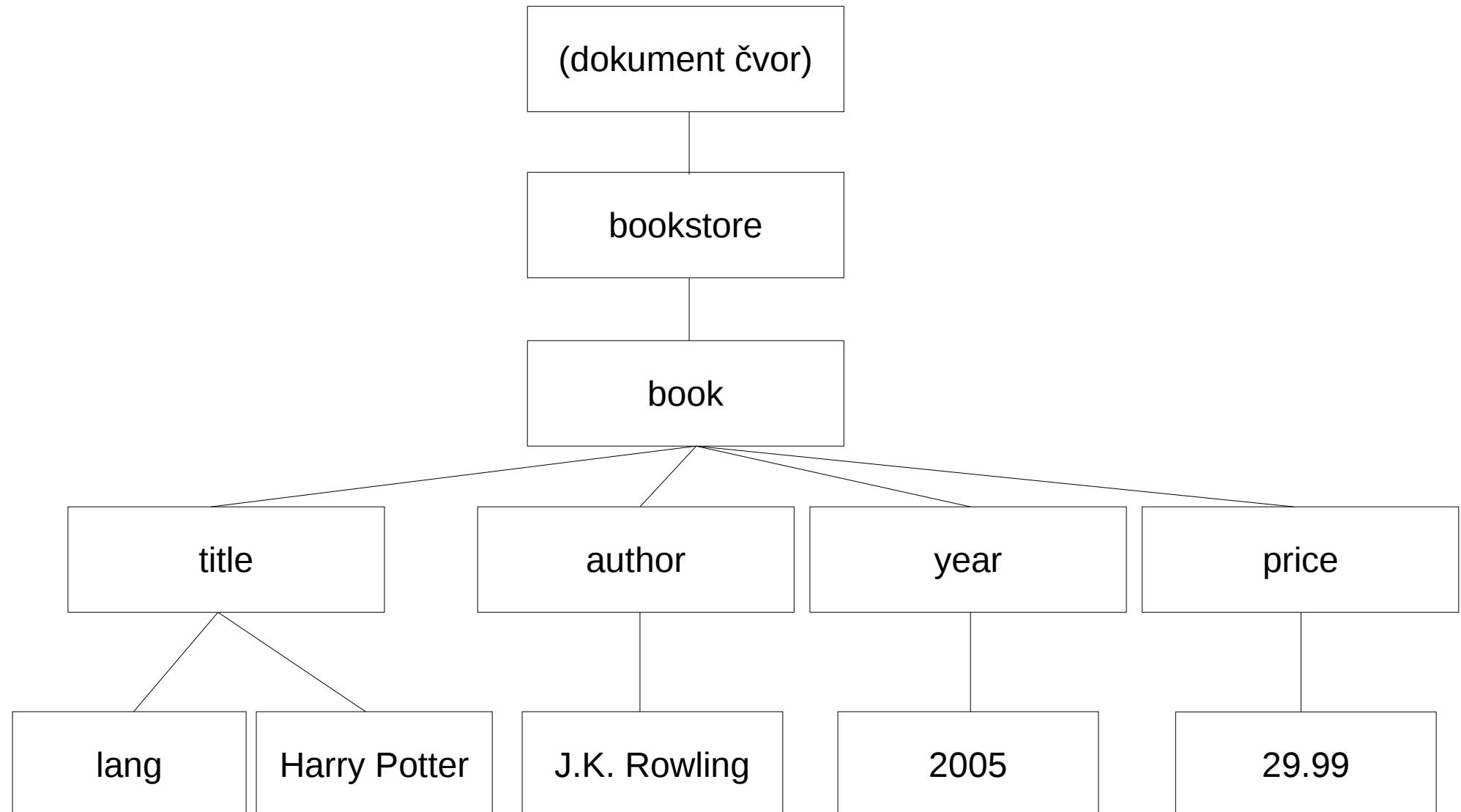
dokument čvor

atribut čvor

element čvor

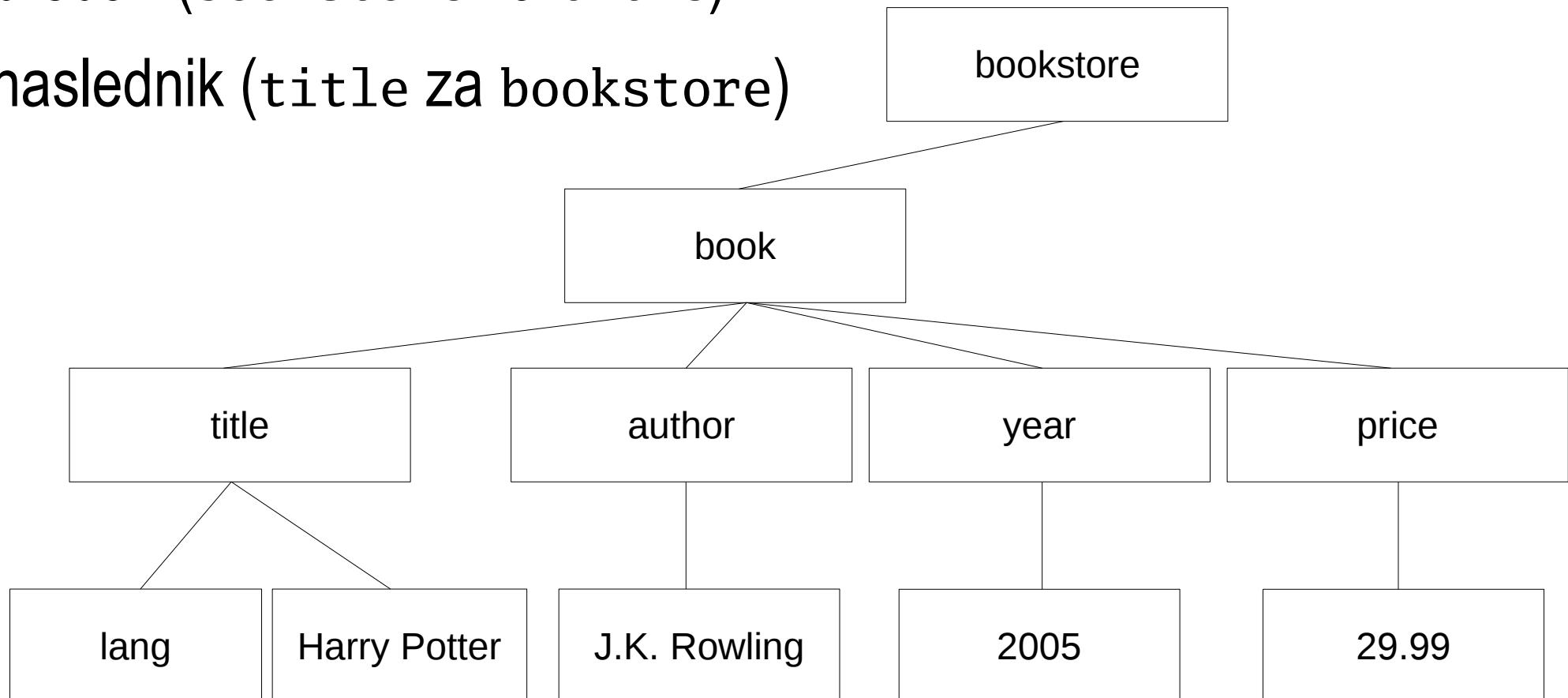
XPath čvorovi

- primer



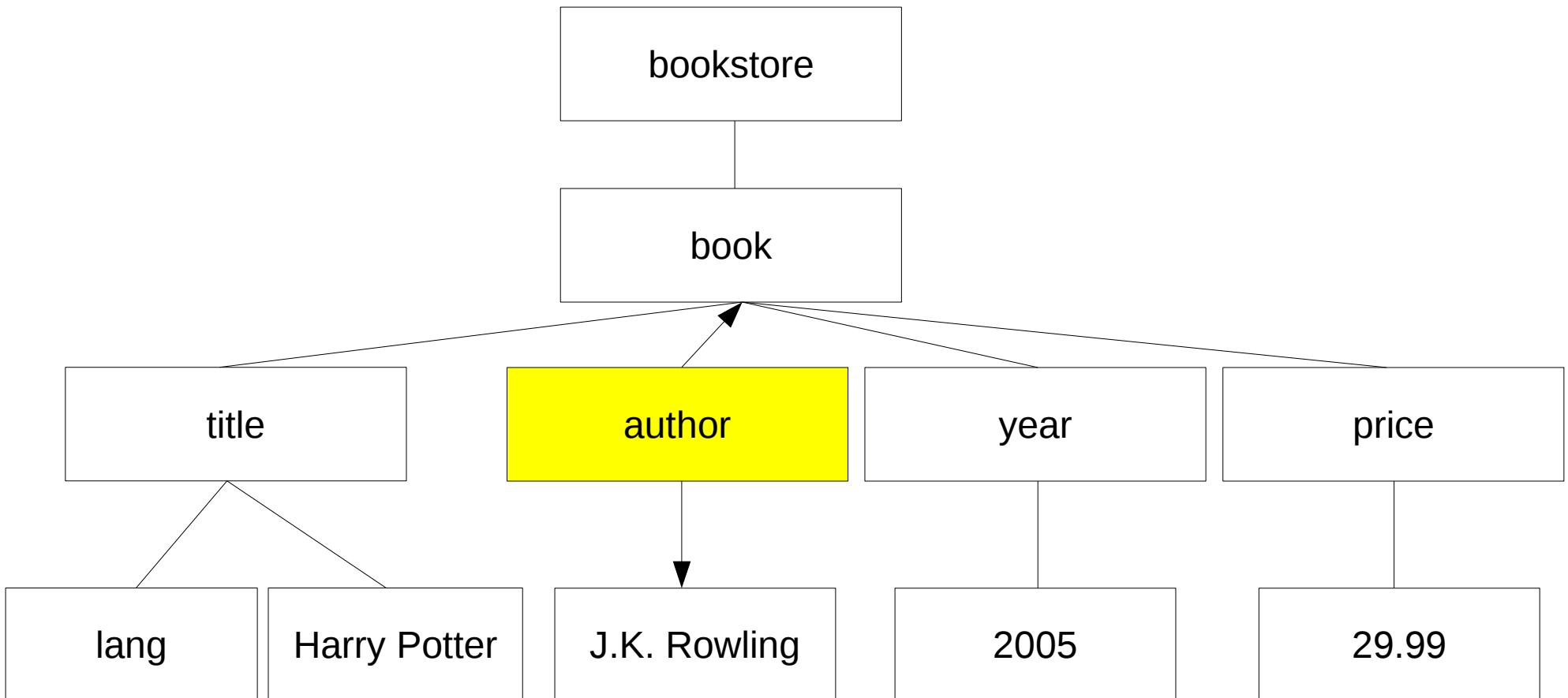
Odnosi među čvorovima

- roditelj (book za title)
- dete (title za book)
- brat/sestra (title i author)
- predak (bookstore za title)
- naslednik (title za bookstore)



Navigacija kroz stablo dokumenta

- koncept tekućeg čvora = “gde se trenutno nalazim”
- ako je tekući čvor author
 - u smeru roditelj nalazi se book
 - u smeru dete nalazi se `text()="J.K. Rowling"`



Navigacija kroz stablo dokumenta

- primer dokumenta

```
<?xml version="1.0"?>
<bookstore>
    <book>
        <title lang="eng">Harry Potter</title>
        <price>29.99</price>
    </book>
    <book>
        <title lang="eng">Learning XML</title>
        <price>39.95</price>
    </book>
</bookstore>
```

XPath putanja

- sastoji se iz više koraka razdvojenih znakom /
- absolutna putanja: počinje znakom /
 - /korak/korak/...
 - kreće uvek od dokument-čvora
- relativna putanja: ne počinje znakom /
 - korak/korak/...
- svaki korak se izračunava u odnosu na čvorove u tekućem skupu čvorova (node-set)

XPath putanja

- korak putanje se sastoji iz
 - ose kretanja
 - testa čvora
 - nula ili više predikata
- opšti oblik izraza za korak je

`osa::test[predikat]`

XPath ose

- osa definiše skup čvorova u odnosu na tekući čvor
 - ancestor svi preci
 - ancestor-or-self svi preci ili sam čvor
 - attribute svi atributi
 - child sva deca
 - descendant svi potomci
 - descendant-or-self svi potomci ili sam čvor
 - following sve u dokumentu iza krajnjeg taga tekućeg
 - following-sibling sva braća posle tekućeg
 - namespace svi namespace čvorovi tekućeg
 - parent roditelj
 - preceding sve u dokumentu iza početnog taga tekućeg
 - preceding-sibling sva braća pre tekućeg
 - self tekući čvor

XPath ose

- primeri
 - `child::book`
 - svi *book* čvorovi koji su deca tekućeg čvora
 - `attribute::lang`
 - atribut *lang* tekućeg čvora (može biti najviše jedan)
 - `attribute::*`
 - svi atributi tekućeg čvora
 - `child::text()`
 - svi tekstualni čvorovi koji su deca tekućeg čvora

XPath ose

- primeri
 - `descendant::book`
 - svi *book* naslednici tekućeg čvora
 - `ancestor::book`
 - svi *book* preci tekućeg čvora
 - `ancestor-or-self::book`
 - svi *book* preci tekućeg čvora i tekući čvor ako je on *book*
 - `child::* / child::price`
 - svi *price* unuci tekućeg čvora

XPath putanja

- skraćeno pisanje
 - ako se osa ne navede podrazumeva se child::
 - self:: se može pisati kao tačka .
 - parent:: se može pisati kao dve tačke ..
 - descendant:: se može pisati kao dve kose crte //
 - attribute:: se može pisati kao @
 - following-sibling:: se može pisati kao ../

XPath ose

- ako se osa ne navede, podrazumeva se child::
 - primeri
 - child::book ↔ book
 - child::book/child::price ↔ book/price
 - child::* / child::price ↔ */price

XPath ose

- ako se umesto ose navede @, podrazumeva se attribute::
- primeri
 - child>::title/attribute::lang \leftrightarrow title/@lang
 - child>::*/attribute::lang \leftrightarrow */@lang

XPath ose

- ako se umesto ose navede //, podrazumjeva se descendant::
- primeri
 - descendant)::title ↔ //title
 - descendant)::@lang ↔ //@lang

Predikati

- koriste se za pronalaženje čvora koji zadovoljava dati uslov
 - osnovna primena: filtriranje čvorova
 - primer: od svih *title* čvorova, izdvoj one čiji je tekstualni sadržaj duži od 20 znakova
- uvek se pišu unutar uglastih zagrada [. . .]

Predikati

- primeri
 - `//title[@lang]`
 - svi elementi *title* koji imaju atribut *lang*
 - `//title[@lang='eng']`
 - svi *title* elementi koji imaju atribut *lang* sa vrednošću 'eng'
 - `/bookstore/book[price > 35.00]`
 - svi *book* elementi deca *bookstore* korena koji imaju podelement *price* sa vrednošću većom od 35.00
 - `/bookstore/book[price > 35.00]/title`
 - svi *title* elementi koji pripadaju *book* elementima koji pripadaju *bookstore* korenju koji imaju *price* element sa vrednošću većom of 35.00
 - `/bookstore/book[price > 35.00]/price`
 - svi *price* elementi koji pripadaju *book* elementima koji pripadaju *bookstore* korenju koji imaju *price* element sa vrednošću većom of 35.00

Predikati

- `[position() = X]` se može pisati kao `[X]`

Predikati

- primeri
 - `/bookstore/book[1]`
 - prvi *book* element koji je dete *bookstore* korena
 - `/bookstore/book[last()]`
 - poslednji *book* element koji je dete *bookstore* korena
 - `/bookstore/book[last()-1]`
 - pretposlednji *book* element koji je dete *bookstore* korena
 - `/bookstore/book[position()<3]`
 - prva dva *book* elementa koji su deca *bookstore* korena

XPath operatori

Operator	Opis	Primer	Rezultat
	unija dva skupa čvorova	//book //cd	skup čvorova sa svim book i cd elementima
+	sabiranje	6 + 4	10
-	oduzimanje	6 - 4	2
*	množenje	6 * 4	24
div	deljenje	6 div 4	1
mod	ostatak pri deljenju	6 mod 4	2
=	jednako	price = 9.80	true ako je price = 9.80, inače false
!=	nejednako	price != 9.80	
<	manje	price < 9.80	
<=	manje ili jednako	price <= 9.80	
>	veće	price > 9.80	
>=	veće ili jednako	price >= 9.80	
or	disjunkcija	price = 9.80 or price = 9.70	
and	konjunkcija	price > 9.00 and price < 9.90	

Biblioteka standardnih funkcija

- core functions: 27 funkcija

- boolean()
- ceiling()
- concat()
- contains()
- count()
- false()
- floor()
- id()
- lang()
- last()
- local-name()
- name()
- namespace-uri()
- normalize-space()
- not()
- number()
- position()
- round()
- starts-with()
- string()
- string-length()
- substring()
- substring-after()
- substring-before()
- sum()
- translate()
- true()

Biblioteka standardnih funkcija

- primeri
 - `position()` - vraća redni broj čvora u tekućem kontekstu
 - `//book[position()=last()]` - poslednji book element
 - `count()` - vraća broj elemenata u skupu čvorova
 - `count("//book")` - broj book elemenata
 - `substring()` - vraća podstring - deo tekstualnog sadržaja
 - `substring('trt mrt', 1, 3)` - vraća trt

Označavanje više putanja odjednom

- navođenjem više XPath izraza povezanih operatorom |
- rezultat je unija skupova čvorova dobijenih osnovnim izrazima
- primeri
 - `//book/title | //book/price`
 - svi *title* elementi deca *book*-a i svi *price* elementi deca *book*-a
 - `//title | //price`
 - svi *title* elementi i svi *price* elementi
 - `/bookstore/book/title | //price`
 - svi *title* elementi *book* elementa *bookstore* korena i svi *price* elementi

XPointer

Šta je XPointer

- jezik za označavanje delova XML dokumenta
 - proširuje XPath
 - pronalaženje podataka poređenjem teksta
 - može da se doda na kraj URI-ja
 - označava ne samo cele čvorove u dokumentu, već i delove čvorova
- koristi se za povezivanje dokumenata

Upotreba XPointera

- ugradivanje u URI-je
 - na kraj URL adrese se doda #xpointer(...)
 - primer:
`http://www.ftn.ns.ac.yu/students.xml#xpointer(//indeks[@id='1234'])`

XPointer lokacija

- pojam lokacije je uopštenje pojma XPath čvora
- lokacija može biti
 - XPath čvor
 - tačka (point)
 - opseg (range)
- opšti oblik XPointer izraza

`xpointer(lokacija)`

XPointer + XPath

- primeri

```
xpointer()
xpointer("//first_name")
xpointer(id('sec_intro'))
xpointer(/people/person/name/first_name/text())
xpointer("//middle_initial[1]/../first_name")
xpointer("//professional[.=\"physicist\"]")
xpointer(/child::people/child::person[@id<4000])
xpointer(/child::people/child::person/attribute::id)
```

XPointer + XPath: identifikacija više elemenata

- spajanjem više XPointer izraza:

```
xpointer("//first_name")xpointer("//last_name")
```

XPointer tačka

- tačka se definiše pomoću
 - čvora koji je sadrži (container node)
 - indeksa i
 - mesto odmah nakon i -tog podelementa (za strukturirani sadržaj)
 - mesto odmah nakon i -tog karaktera (za tekstualni sadržaj)

XPointer tačka

- za definisanje tačke mogu se koristiti funkcije
 - start-point()
 - end-point()

XPointer tačka

- start-point(): tačka “ispred”

tačka

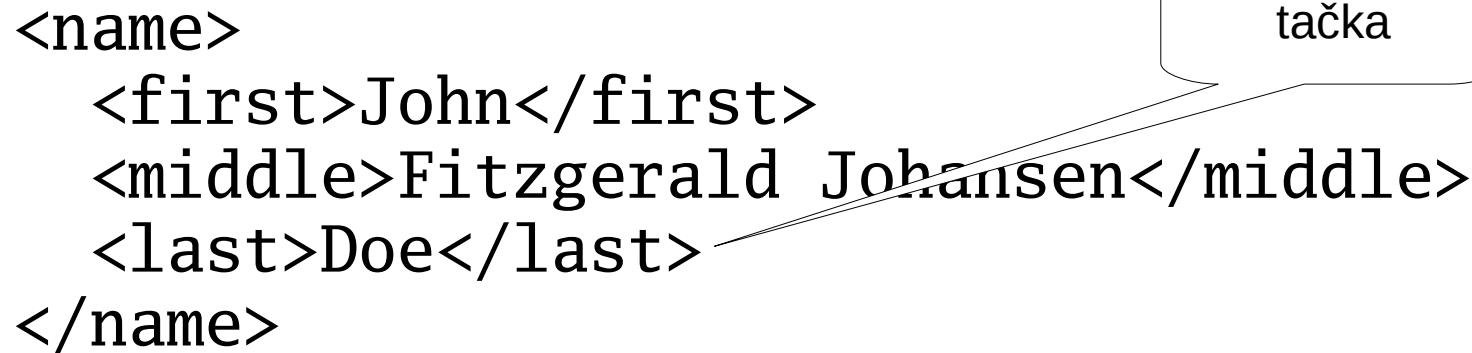
```
<name>
  <first>John</first>
  <middle>Fitzgerald Johansen</middle>
  <last>Doe</last>
</name>
```

xpointer(start-point(/name/last))

XPointer tačka

- end-point(): tačka “iza”

```
<name>
  <first>John</first>
  <middle>Fitzgerald Johansen</middle>
  <last>Doe</last>
</name>
```



A speech bubble containing the word "tačka" (dot) points towards the final closing tag of the XML element, "</last>".

```
xpointer(end-point(/name/last))
```

XPointer tačka

- tačka u tekstualnom sadržaju

```
<name>
  <first>John</first>
  <middle>Fitzgerald Johansen</middle>
  <last>Doe</last>
</name>
```

tačka

xpointer("//middle/text()/**point[2]**))

XPointer opseg

- opseg predstavlja interval između dve tačke, početne i krajnje
- definiše se funkcijama
 - range-to()
 - range()
 - range-inside()
 - string-range()

XPointer range()

- parametar funkcije je XPath izraz
- opseg je ono što obuhvata rezultat XPath izraza

```
xpointer(range("//title"))
```

```
<novel copyright="public domain">
    <title>The wonderful wizard of Oz</title>
    <author>L. Frank Baum</author>
    <year>1900</year>
</novel>
```

XPointer range()

- može da vrati i više opsega od jednom

```
xpointer(range(/novel/*))
```

```
<novel copyright="public domain">
  <title>The wonderful wizard of Oz</title>
  <author>L. Frank Baum</author>
  <year>1900</year>
</novel>
```

XPointer range-inside()

- vraća unutrašnjost izabranog čvora

```
xpointer(range-inside(//title))
```

```
<novel copyright="public domain">
  <title>The wonderful wizard of Oz</title>
  <author>L. Frank Baum</author>
  <year>1900</year>
</novel>
```

XPointer range-to()

- početna tačka: XPath čvor
- krajnja tačka: parametar funkcije

```
xpointer("//title/range-to(year))
```

```
<novel copyright="public domain">
    <title>The wonderful wizard of Oz</title>
    <author>L. Frank Baum</author>
    <year>1900</year>
</novel>
```

XPointer range-to()

-

```
xpointer("//title/range-to("//title/text()))")
```

```
<novel copyright="public domain">
  <title>The wonderful wizard of Oz</title>
  <author>L. Frank Baum</author>
  <year>1900</year>
</novel>
```

XPointer string-range()

- operiše nad tekstrom dokumenta kada se ukloni sav markup
- pronalazi traženi tekst i to vraća kao rezultat

```
xpointer(string-range("//title, “wizard”))
```

```
<novel copyright=“public domain”>  
  <title>The wonderful wizard of Oz</title>  
  <author>L. Frank Baum</author>  
  <year>1900</year>  
</novel>
```

XPointer string-range()

- može se navesti offset i dužina

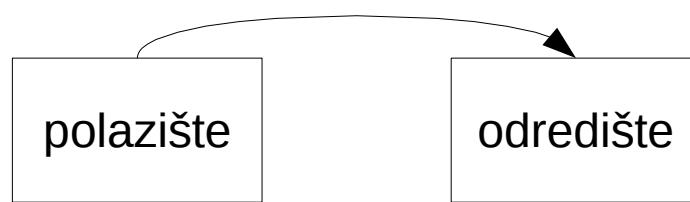
```
xpointer(string-range("//title, “wizard”, 5, 4))
```

```
<novel copyright=“public domain”>
  <title>The wonderful wizard of Oz</title>
  <author>L. Frank Baum</author>
  <year>1900</year>
</novel>
```

XLink

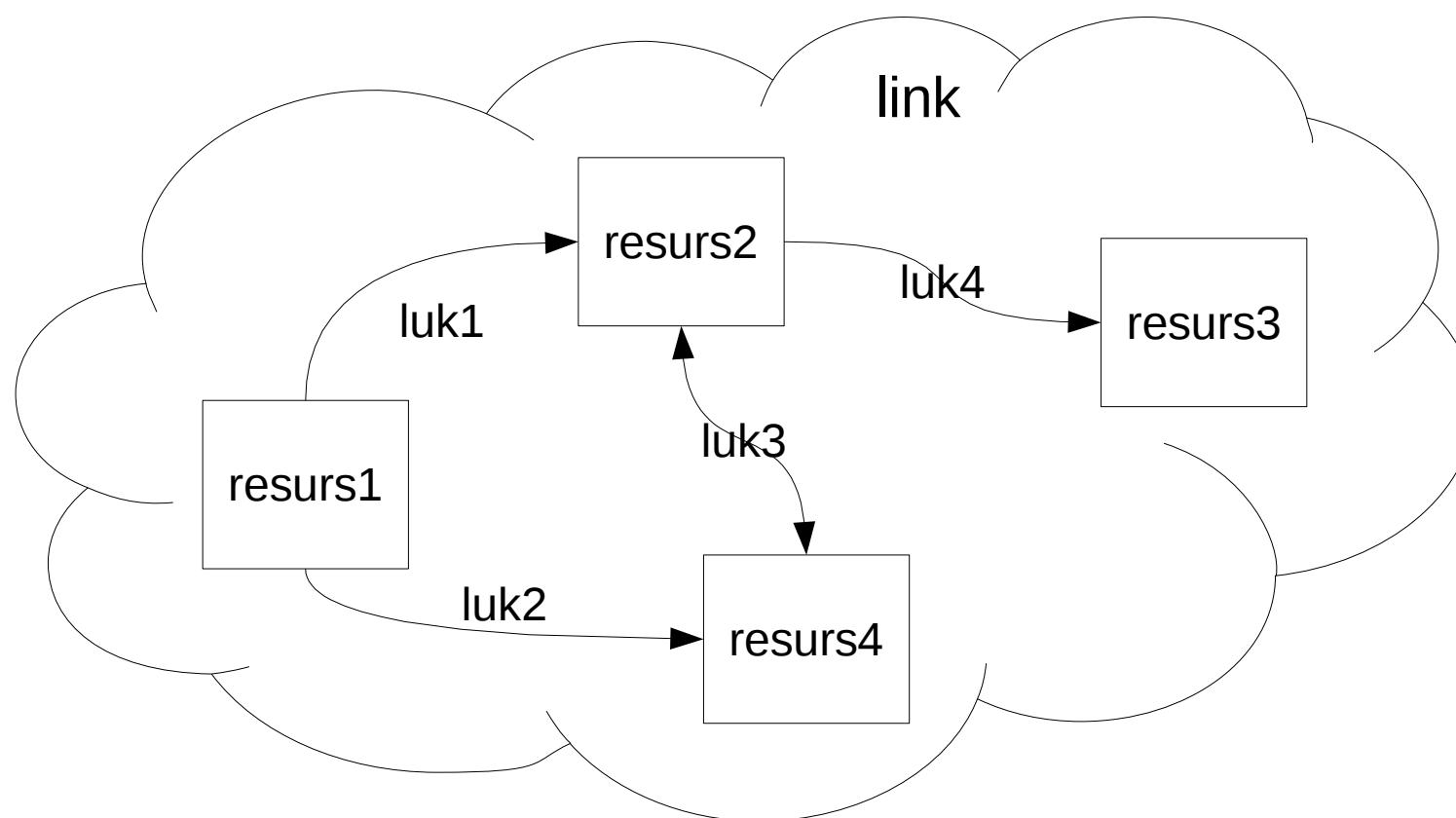
Šta je XLink

- jezik za definisanje linkova između (delova) dokumenata
- pojam linka je opštiji nego u HTML-u
 - HTML link povezuje dva resursa jednom *usmerenom* granom
 - polazište
 - odredište
 - link je uvek smešten u polazištu



XLink link

- link povezuje dva ili više resursa
- resurs je identifikovan URL-jem
- link ima jedan ili više lukova



XLink link

- gde je smešten link?
 - u nekom od resursa koji učestvuju u linku
 - negde drugde
- HTML: link je uvek definisan u polazišnom resursu jedinog luka
 - HTML link je jednostavna varijanta XLink linka

XLink link

- definisanje linka
 - za definisanje linka koriste se globalni atributi definisani u <http://www.w3.org/1999/xlink>
 - uobičajeni prefiks u XML dokumentima za ovaj namespace je `xlink`
 - ne postoji posebni elementi za definisanje linka, svaki element iz nekog dokumenta može poslužiti toj svrsi

XLink link

- atributi linka
 - `xlink:type` - tip linka
 - `xlink:href` - URI resursa
 - `xlink:role` - mašinski čitljiv opis uloge resursa
 - `xlink:title` - ljudski čitljiv opis uloge resursa
 - `xlink:show` - kako će resurs biti prikazan kada se dobavi
 - `xlink:actuate` - kada će resurs biti dobavljen
 - `xlink:from` - polazište linka
 - `xlink:to` - odredište linka

xlink:type

- xlink:type je obavezan atribut u elementu koji definiše link
- opisuje ulogu elementa koji ga sadrži u samom linku
- od njegove vrednosti zavisi koji drugi atributi će biti prisutni
- šest mogućih vrednosti
 - simple
 - extended
 - locator
 - arc
 - resource
 - title

xlink:href

- xlink:href sadrži adresu resursa koji učestvuje u linku
- adresa je u URI obliku
- primer
`<myElement xlink:href="http://www.ftn.ns.ac.yu/">`

xlink:role i xlink:title

- `xlink:role` predstavlja mašinski čitljiv opis uloge resursa u linku
- `xlink:title` predstavlja tekstualni opis prilagođen čoveku
- primer

```
<element xmlns:name="http://www.ftn.ns.ac.yu/name"  
         xlink:type="simple"  
         xlink:href="http://www.somewhere.com"  
         xlink:role="name:first"  
         xlink:title="Retrieve first name">  
    Click here!  
</element>
```

xlink:actuate i xlink:show

- `xlink:actuate` navodi kada će resurs biti dobavljen
 - `onLoad` - resurs se dobavlja čim se dokument (koji sadrži link) učita
 - `onRequest` - resurs se dobavlja kada korisnik to zatraži
 - `undefined` - aplikacija može da ga tretira kako hoće
 - možemo definisati sopstvene vrednosti u QName obliku, npr. `ftn:EveryFiveMinutes` (`ftn` je prefiks za neki namespace)

xlink:show

- xlink:show opisuje kako će se resurs prikazati kada se učita
 - new - biće prikazan u novom prozoru
 - replace - biće prikazan u prozoru u kome je link aktiviran
 - embed - biće prikazan u polaznom dokumentu na mestu linka
 - undefined - aplikacija može da ga tretira kako hoće
 - možemo definisati sopstvene vrednosti u QName obliku
- primer

```
<JobList xlink:type="simple"
          xlink:show="replace"
          xlink:actuate="onRequest"/>
```

xlink:from i xlink:to

- `xlink:from` i `xlink:to` definišu luk
- njihova vrednost je jednaka vrednosti atributa `role` na mestu gde se definiše resurs

Vrste linkova

- XLink poznaje dve vrste linkova
 - simple
 - povezuje dva resursa jednim jednosmernim lukom
 - extended
 - opšti oblik linka

Simple link

- definicija simple linka
 - xlink:type="simple"
 - xlink:href="URI-odredišta"

- primer

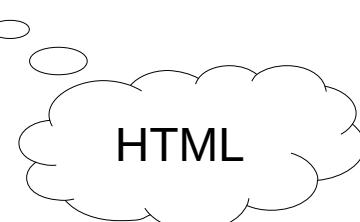
```
<SomeElement
```

```
    xlink:type="simple"
    xlink:href="http://somewhere.com"
    xlink:title="Click to go somewhere"
    xlink:show="new">
        Click me!

```

```
</SomeElement>
```

```
<a href="http://somewhere.com/"
    title="Click to go somewhere"
    target="_blank">
    Click me!
</a>
```



Simple link

- primer: person

```
<person>
  <name>...</name>
  <picture>...</picture>
  <homepage>...</homepage>
</person>
```

Simple link

- primer: person

```
<person>
  <name>...</name>
  <picture xlink:type="simple"
    xlink:href="picture.jpg"
    xlink:actuate="onRequest"
    xlink:show="embed"
    xlink:title="Click to see"/>Picture</picture>
  <homepage>...</homepage>
</person>
```

Simple link

- primer: person

```
<person>
  <name>...</name>
  <picture>...</picture>
  <homepage xlink:type="simple"
    xlink:href="http://www.ftn.ns.ac.yu/personal/"
    xlink:actuate="onRequest"
    xlink:show="replace">Homepage</homepage>
</person>
```

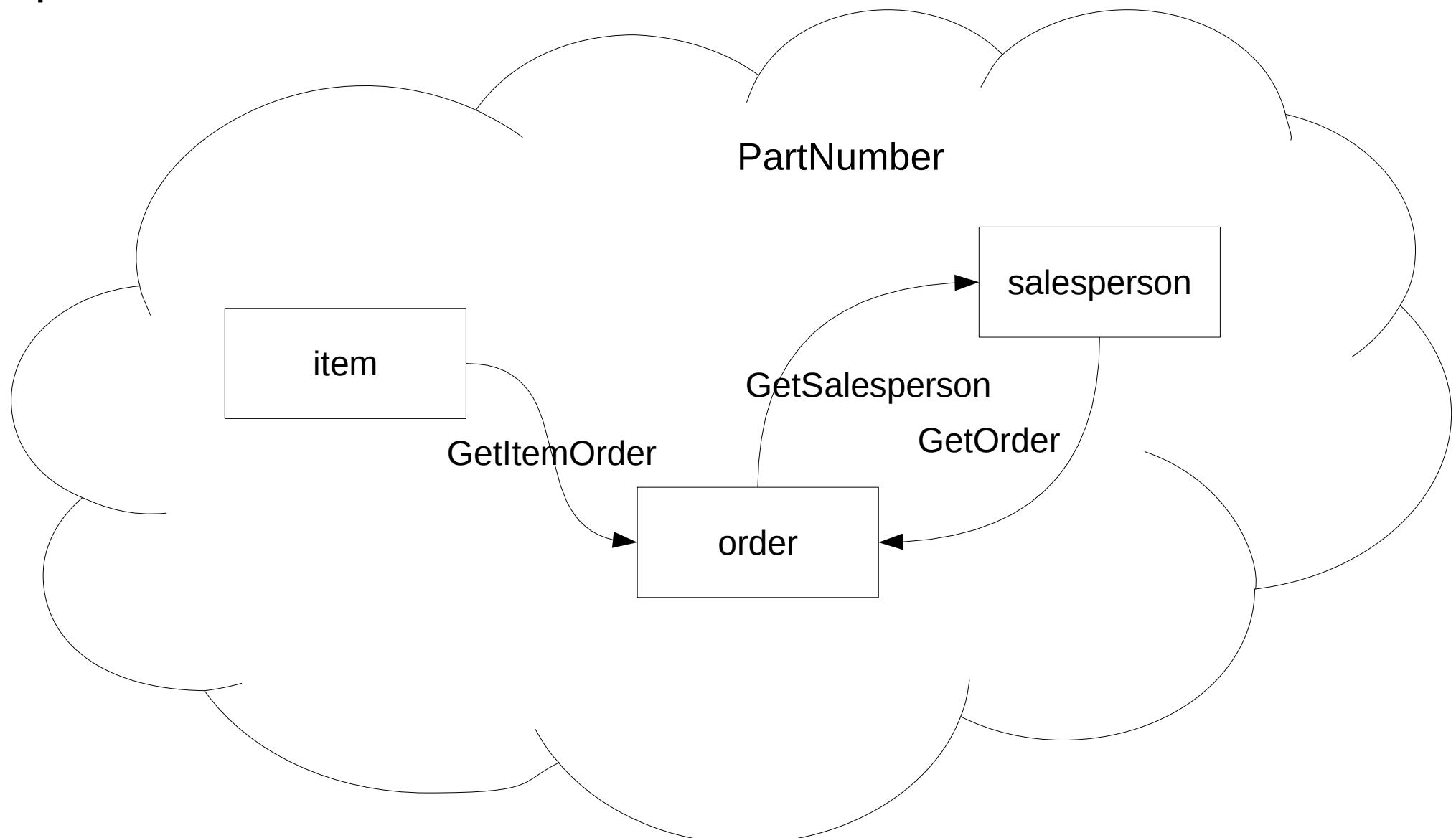
Simple link

- primer: person

```
<person>
  <name xlink:type="simple"
    xlink:href="name.xml"
    xlink:actuate="onLoad"
    xlink:show="embed"/>
  <picture>...</picture>
  <homepage>...</homepage>
</person>
```

Extended link

- xlink:type="extended"
- primer



Extended link

- svaki deo linka opisan je posebnim elementom u dokumentu

```
<PartNumber xmlns:xlink="http://www.w3.org/1999/xlink"  
    xmlns:op="http://foo.com/OrderProcessingSystem"  
    xlink:type="extended">  
    <item xlink:type="resource">...</item>  
    <salesperson xlink:type="locator"/>  
    <order xlink:type="locator"/>  
    <GetOrder xlink:type="arc"/>  
    <GetSalesperson xlink:type="arc"/>  
    <GetItemOrder xlink:type="arc"/>  
</PartNumber>
```

ugrađeni resurs

udaljeni resurs

luk

Extended link

- ugradeni resurs

```
<PartNumber xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:op="http://foo.com/OrderProcessingSystem"
    xlink:type="extended">
    <item xlink:type="resource"
        xlink:role="op:item"
        xlink:title="Item">
        <part-number>E16-25A</part-number>
        <description>Production-Class Widget</description>
    </item>
    <salesperson xlink:type="locator"/>
    <order xlink:type="locator"/>
    <GetOrder xlink:type="arc"/>
    <GetSalesperson xlink:type="arc"/>
    <GetItemOrder xlink:type="arc"/>
</PartNumber>
```

Extended link

- udaljeni resurs 1

```
<PartNumber xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:op="http://foo.com/OrderProcessingSystem"
    xlink:type="extended">
    <item xlink:type="resource">...</item>
    <salesperson xlink:type="locator"
        xlink:href="http://foo.com/JohnDoe.xml"
        xlink:role="op:salesperson"
        xlink:title="Salesperson"/>
    <order xlink:type="locator"/>
    <GetOrder xlink:type="arc"/>
    <GetSalesperson xlink:type="arc"/>
    <GetItemOrder xlink:type="arc"/>
</PartNumber>
```

Extended link

- udaljeni resurs 2

```
<PartNumber xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:op="http://foo.com/OrderProcessingSystem"
    xlink:type="extended">
    <item xlink:type="resource">...</item>
    <salesperson xlink:type="locator"/>
    <order xlink:type="locator"
        xlink:href="http://sernaferna.com/order256.xml"
        xlink:role="op:order"
        xlink:title="Order"/>
    <GetOrder xlink:type="arc"/>
    <GetSalesperson xlink:type="arc"/>
    <GetItemOrder xlink:type="arc"/>
</PartNumber>
```

Extended link

- luk 1

```
<PartNumber xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:op="http://foo.com/OrderProcessingSystem"
    xlink:type="extended">
    <item xlink:type="resource">...</item>
    <salesperson xlink:type="locator"/>
    <order xlink:type="locator"/>
    <GetOrder xlink:type="arc"
        xlink:from="op:salesperson"
        xlink:to="op:order"
        xlink:show="replace"
        xlink:actuate="onRequest"
        xlink:role="op:GetOrder"
        xlink:title="Last order processed."/>
    <GetSalesperson xlink:type="arc"/>
    <GetItemOrder xlink:type="arc"/>
</PartNumber>
```

Extended link

- Luk 2

```
<PartNumber xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:op="http://foo.com/OrderProcessingSystem"
    xlink:type="extended">
    <item xlink:type="resource">...</item>
    <salesperson xlink:type="locator"/>
    <order xlink:type="locator"/>
    <GetOrder xlink:type="arc"/>
    <GetSalesperson xlink:type="arc"
        xlink:from="op:order"
        xlink:to="op:salesperson"
        xlink:show="replace"
        xlink:actuate="onRequest"
        xlink:role="op:GetSalesperson"
        xlink:title="Salesperson's name"/>
    <GetItemOrder xlink:type="arc"/>
</PartNumber>
```

Extended link

- luk 3

```
<PartNumber xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:op="http://foo.com/OrderProcessingSystem"
    xlink:type="extended">
    <item xlink:type="resource">...</item>
    <salesperson xlink:type="locator"/>
    <order xlink:type="locator"/>
    <GetOrder xlink:type="arc"/>
    <GetSalesperson xlink:type="arc"/>
    <GetItemOrder xlink:type="arc"
        xlink:from="op:item"
        xlink:to="op:order"
        xlink:show="new"
        xlink:actuate="onRequest"
        xlink:role="op:GetItemOrder"
        xlink:title="Last order placed for this item"/>
</PartNumber>
```

Extended link

- element sa xlink:type="extended" obuhvata
 - resurse: elemente sa xlink:type="resource" ili xlink:type="locator"
 - lukove: elemente sa xlink:type="arc"
- resursi i lukovi mogu da obuhvataju element sa xlink:type="title" sa tekstualnim opisom

XQuery

XQuery

- standardni upitni jezik za XML
- W3C standard
 - XQuery 1.0: An XML Query Language
 - <http://www.w3.org/TR/xquery/>
- dugotrajan razvoj (1998-2007)
- ekvivalent SQL-a za XML dokumente

XQuery tipovi podataka

- svi ugrađeni XML Schema tipovi
- još 7 tipova vezanih za tipove čvorova u stablu dokumenta
- još 6 tipova specifičnih za XQuery

XQuery tipovi podataka

- svaka XQuery vrednost je sekvenca
koja sadrži 0 ili više elemenata
- element sekvence je *singleton*
sekvenca dužine 1, koja sadrži baš taj element
 - $(1) = 1$
- sekvenca može biti prazna
 - $()$
- ali ne može sadržati druge sekvenце
 - sekvence se „poravnavaju“
 - $(0, (), (1, 2)) = (0, 1, 2)$

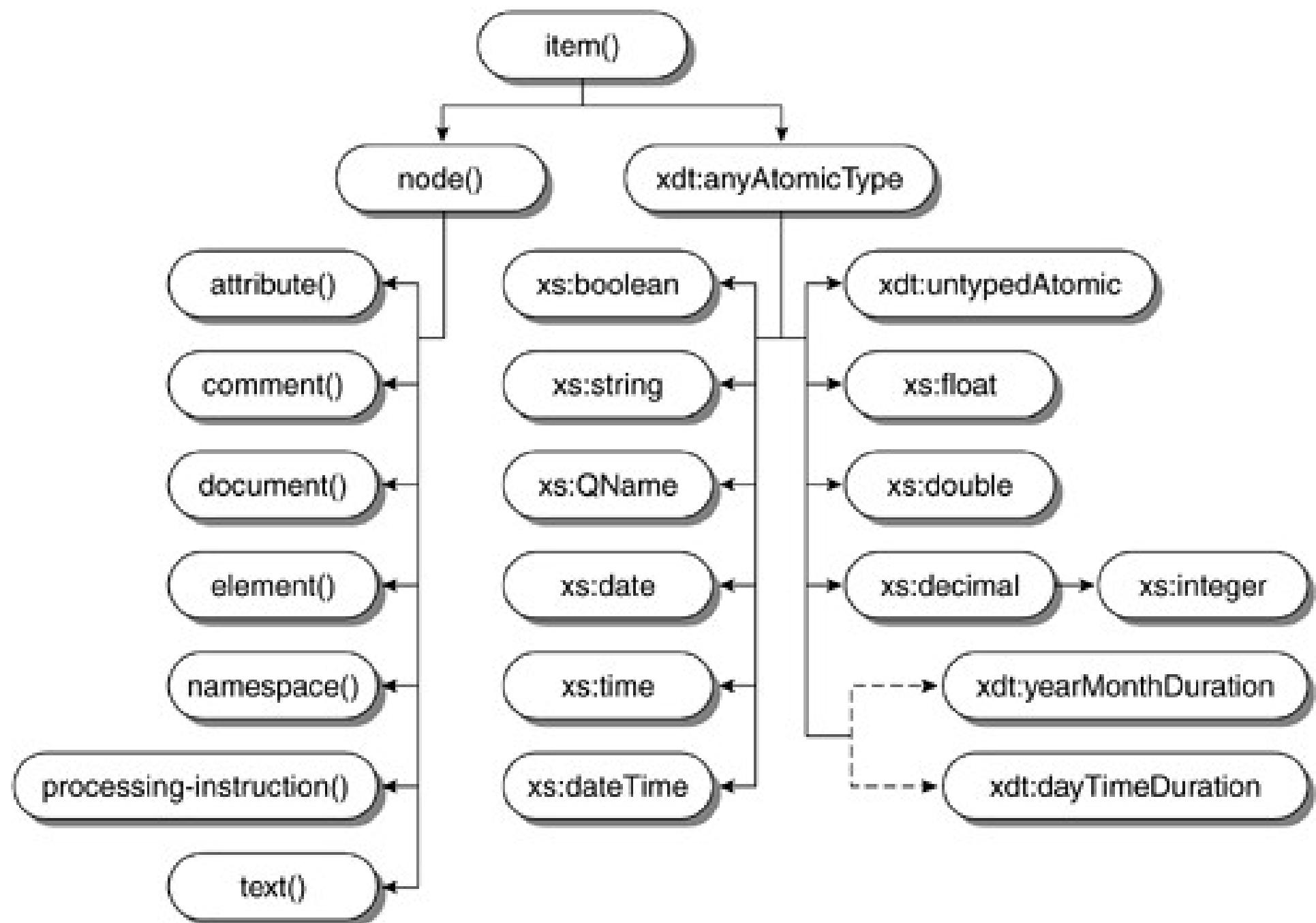
XQuery tipovi podataka

- svaki *singleton* ima svoj tip izведен iz `item()`
 - slično kao `java.lang.Object`, ali je apstraktan - ne može se instancirati
 - piše se sa zagradama da bi se razlikovao od korisničkih tipova istog imena
 - da bude nalik XPath testu čvora

XQuery tipovi podataka

- postoji dve vrste `item()`-a
 - XML čvorovi
 - nasleđuju `node()`
 - atomičke vrednosti
 - nasleđuju `xdt:anyAtomicType`

XQuery tipovi podataka



XQuery tipovi podataka

- sekvenca ima tip koji se sastoji od
 - imena tipa ili `empty()` - prazna sekvenca,
 - (opciono) indikatora ponavljanja
 - `*` - 0 ili više
 - `+` - 1 ili više
 - `?` - 0 ili 1
- primeri
 - `item()` - sekvenca koja sadrži jedan element bilo kog tipa
 - `item()*` - sekvenca koja sadrži $0..*$ elemenata bilo kog tipa
 - `xsd:integer*` - sekvenca koja sadrži $0..*$ celih brojeva
 - `xsd:integer*` je ujedno i `item()*`

XQuery tipovi podataka

- svaki XQuery izraz ima
 - statički tip (*compile-time*)
 - dinamički tip (*run-time*)
 - tip dobijenog rezultata
 - vrednost rezultata je instanca tog tipa
- provera se može vršiti compile-time i run-time
- primer dinamičke provere:
`12 + if ($foo) then 30 else "0"`
(: ako je \$foo=true izraz je ispravan,
inače nije ispravan :)

Whitespace i komentari

- whitespace znakovi su
 - U+0020 (space),
 - U+0009 (tab),
 - U+000D (carriage return),
 - U+000A (line feed)
- komentar može da se pojavi bilo gde na mestu whitespace znakova
- komentar se navodi između (: . . . :)

Whitespace i komentari

- primer

```
(: Komentar. :)
let $i := 42 (: Komentar. :)
return <x>(: Nije komentar. :)</x>
```

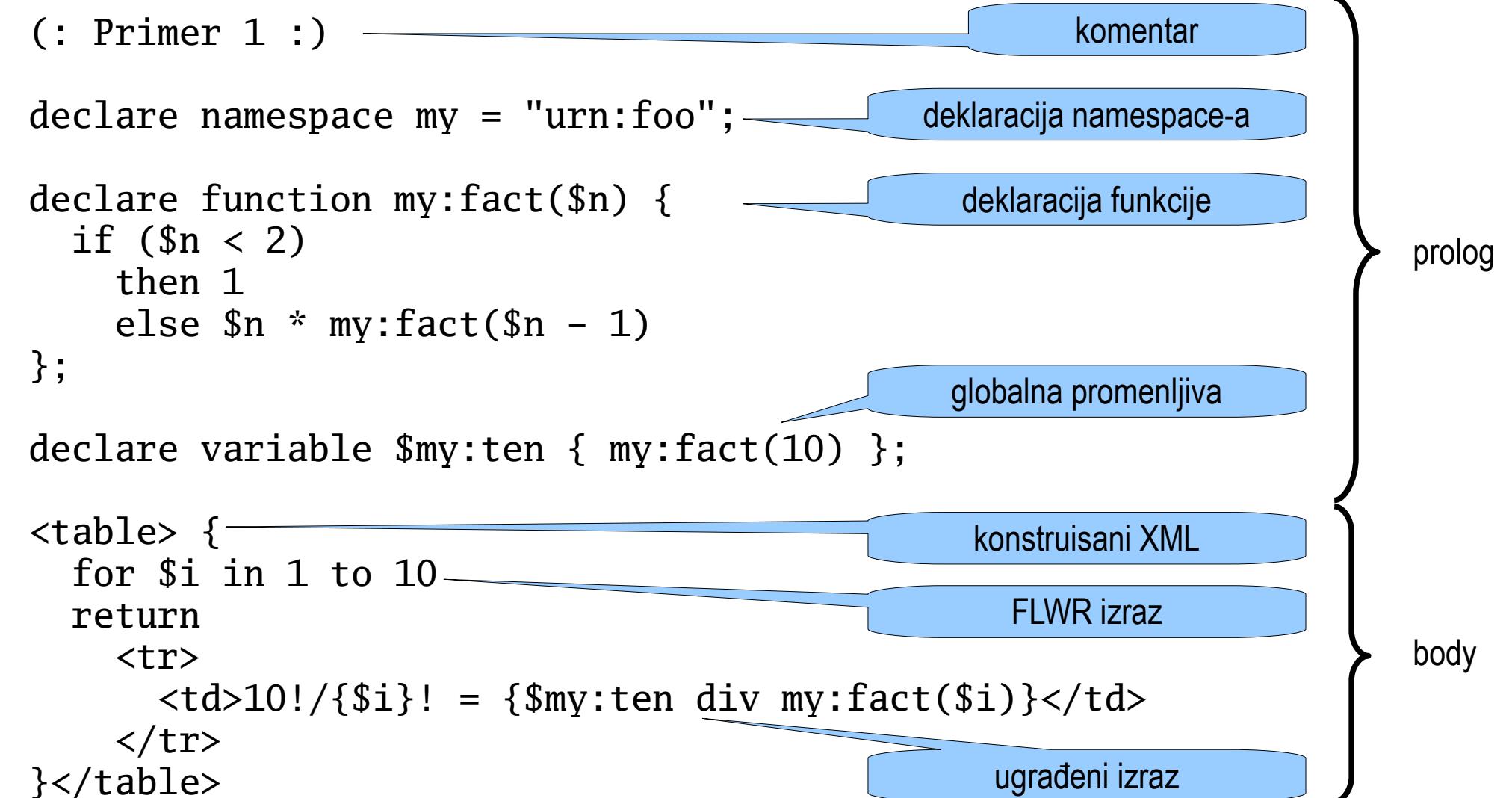
- rezultat

```
<x>(: Nije komentar. :)</x>
```

Konstante

- prazna sekvenca: ()
- logičke (xs:boolean): true(), false()
- stringovi (xs:string): "hello", 'world'
- celi brojevi (xs:integer): 42
- brojevi u fiksnom zarezu (xs:decimal): 42., 4.2, .42
- brojevi u pokretnom zarezu (xs:double): 42E0, 4.2e+0, 42E-2
- drugi tipovi: xs:float("1.25"), xs:ID("X1")

Primer XQuery upita



XQuery prolog

- nije obavezan
- definiše kontekst za upit (compile-time)
 - namespaces
 - korisničke funkcije
 - importovani šema tipovi
 - importovani moduli
 - promenljive

XQuery prolog

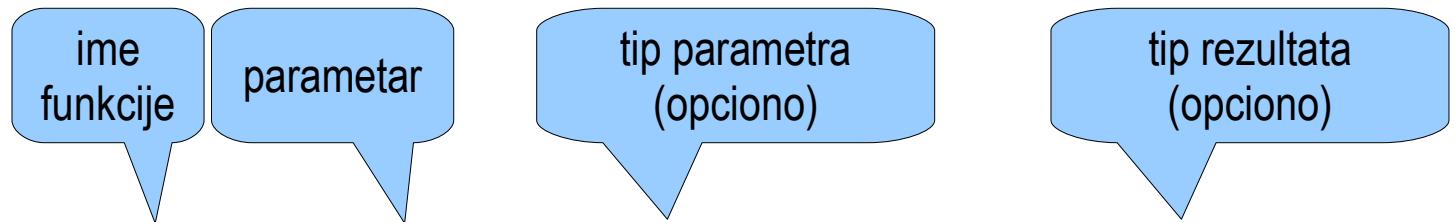
- deklaracije namespace-ova u prologu

```
declare namespace x = "http://www.foo.com";
```

```
<x:foo/>
```

Korisničke funkcije

- deklaracija korisničke funkcije



```
declare function my:fact($n as xs:integer) as xs:integer
{
  if ($n < 2)
    then 1
    else $n * my:fact($n - 1)
};
```

XML sadržaj

- rezultat XQuery upita može biti
 - atomička vrednost
 - XML sadržaj
 - XQuery se često koristi za generisanje XML dokumenata
 - slično kao u SQL-u (rezultat upita je relacija)
- XML node constructor
`<hello>world</hello>`

XML sadržaj

- dinamičko generisanje sadržaja: XQuery izrazi unutar { . . . }

- primer

```
<x y="2*2 = {2*2}">  
Velika istina: 2*2={2*2}.  
</x>
```

- rezultat

```
<x y="2*2 = 4">  
Velika istina: 2*2=4.  
</x>
```

XML sadržaj

- navođenje vitičastih zagrada: {{ i }}

- primer

```
<add>  
  {{ 1 + 1 = { 1+1 } }}  
</add>
```

- rezultat

```
<add>{ 1 + 1 = 2 }</add>
```

XML sadržaj

- alternativni način za konstrukciju elemenata:
`element {"ime"} {"sadržaj"} → <ime>sadržaj</ime>`
- konstruktori za različite tipove čvorova

```
document {  
    element foo {  
        attribute bar { 1 + 1 }  
        text { "trt" }  
        <x xmlns='urn:x'>Može i da se meša</x>  
    }  
}  
↓  
<foo bar="2">trt<x xmlns='urn:x'>Može i da se meša</x></foo>
```

XML sadržaj

- sekvence se poravnaju pre ugrađivanja u XML

```
<x y="{} () {}">{ (1, 2) }</x>
```



```
<x y="">1 2</x>
```

Operatori

- zarez: definiše sekvencu
1, "trt"
 - zarez ima najniži prioritet, pa se sekvene često smeštaju u zagrade

- zagrade: mogu da grupišu izraze različitih tipova

$$() \rightarrow ()$$

$$(1, 2) \rightarrow (1, 2)$$

$$1 + 2 * 3 \rightarrow 7$$

$$(1 + 2)*3 \rightarrow 9$$

Operatori

- logički operatori
 - and
 - or
 - not() ... piše se kao funkcija zbog kompatibilnosti sa XPath
- if-then-else operator
 - else je obavezan
 - if (true()) then "true" else "false"

Operatori

- aritmetički
 - binarni: +, -, *, div, idiv, mod
 - unarni: +, -
- ima i aritmetičkih funkcija
`min((2, 1, 3, -100))`
`round(9 div 2)`
`round-half-to-even(9 div 2)`

Operatori

- poređenje
 - poređenje vrednosti: za poređenje dva singletona
 - eq, ne, gt, ge, lt, le
 - generalno poređenje: za poređenje dve sekvene
 - vraćaju true() ako u obe sekvene postoji bar po jedan element za koje poređenje po vrednosti vraća true()
 - $(1, 2, 3) = 4 \rightarrow \text{false}$
 - $(1, 2, 3) = 3 \rightarrow \text{true}$
 - $(1, 2) = (3, 4) \rightarrow \text{false}$
 - $(1, 2) != (3, 4) \rightarrow \text{true}$
 - $(1, 2) = (2, 3) \rightarrow \text{true}$
 - $(1, 2) != (2, 3) \rightarrow \text{true}$
 - $(1, 2) != (1, 2) \rightarrow \text{true}$

Operatori

- poređenje
 - poređenje čvorova: operišu nad sekvencama čvorova
 - << - „before“: vraća true ako je levi čvor ispred/pre desnog u dokumentu
 - >> - „after“: vraća true ako je levi čvor iza/posle desnog u dokumentu
 - is - vraća true ako su čvorovi isti (po identitetu)
 - isnot - negacija od is
 - primer:

<a/> is → false

<a/> isnot <a/> → true

x/.. << x → true

Operatori

- funkcije za poređenje
 - `compare()`: poredi dve atomičke vrednosti
 - `deep-equal()`: poredi cele sekvene, sa dubokim poređenjem svih elemenata

Ugrađene funkcije

- 110 ugrađenih funkcija
- razlikuju se po imenu i listi parametara
- namespace: <http://www.w3.org/2003/11/xpath-functions>
 - uobičajeni prefiks: fn
- primer 1: broj elemenata sekvence
$$\text{fn:count}((\text{"a"}, \text{ 2, "c"})) \rightarrow 3$$
- primer 2: podsekvenca
$$\text{fn:subsequence}((-5,4,-3,2,-1), \text{ 2, 3}) \rightarrow (4,-3,2)$$

Ugrađene funkcije

- funkcije nad sekvencama
 - `count()`: dužina sekvenca
 - `distinct-values()`: ukloni sve duplike
 - `empty()`: da li je sekvenca prazna
 - `exists()`: da li sekvenca nije prazna
 - `index-of()`: položaj elementa u sekvenci
 - `insert-before()`: ubaci element u sekvencu
 - `remove()`: ukloni element iz sekvenca
 - `reverse()`: obrni redosled sekvence
 - `subsequence()`: izdvoj podsekvencu
 - `unordered()`: naglasi da redosled nije važan

Ugrađene funkcije

- aritmetičke funkcije
 - floor()
 - ceiling()
 - abs()
 - min()
 - max()
 - avg()
 - sum()
 - round()
 - round-half-to-even()

Sintaksni biseri

- znak - je validan znak za ime
 - a-1 nije isto što i a - 1
- znak / je separator koraka u XPath putanji
 - a/b nije isto što i a div b

Sintaksni biseri

- $\text{not}(=)$ i $!=$
 - operatori $=$ i $!=$ proveravaju egzistenciju
 - operatori eq i ne ne proveravaju egzistenciju
 - $\text{not}(a=b)$ nije isto što i $a != b$
 - prva varijanta negira i test egzistencije i test jednakosti
 - primer

$x[@y = 1]$

(: nalazi $<x y="1"/>$ ali ne i $<x/>$ i $<x y="2"/>$:)

$x[\text{not}(@y=1)]$

(: nalazi $<x y="2"/>$ i $<x/>$ ali ne i $<x y="1"/>$:)

$x[@y != 1]$

(: nalazi $<x y="2"/>$ ali ne i $<x/>$ i $<x y="1"/>$:)

Sintaksni biseri

- aritmetički operatori nisu (uvek) asocijativni

- primer dokumenta:

```
<x>
  <y>2</y>
  <y>3</y>
</x>
```

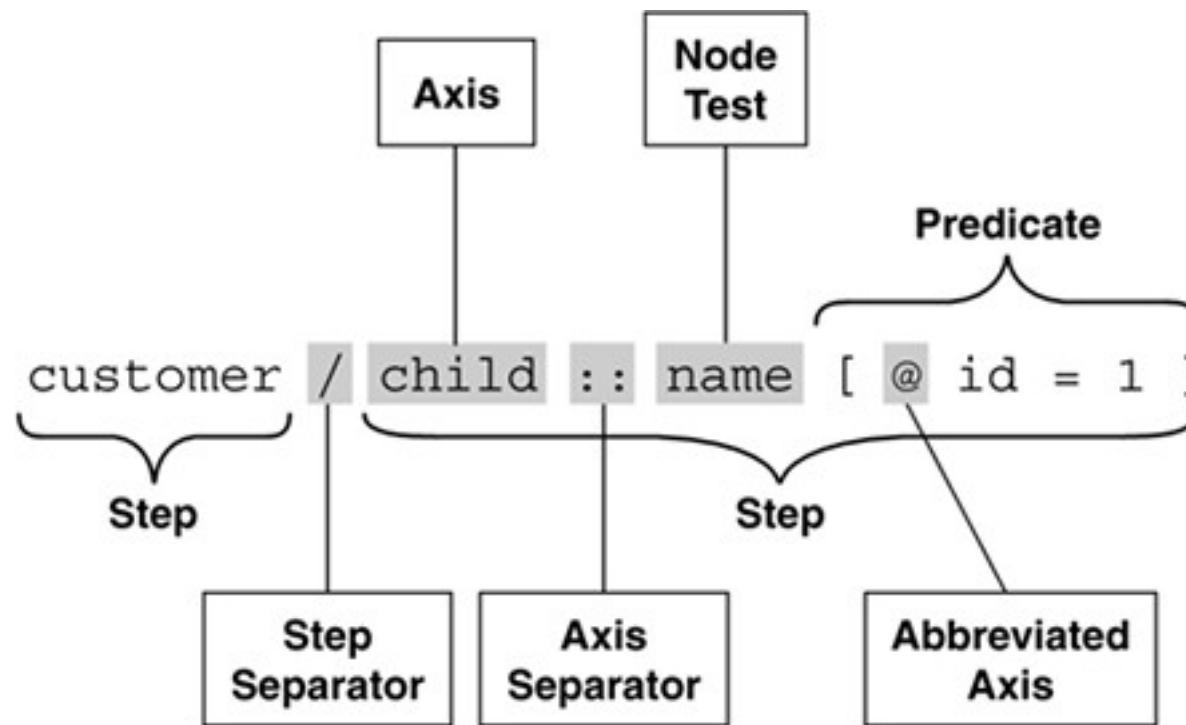
- primer dva različita izraza

```
/x[1 + 2 = y]
  (: pronalazi x sa detetom y jednakim 3 :)
```

```
/x[      2 = y - 1]
  (: greška: y-1 nije atomička vrednost :)
```

Putanje

- koristi se neznatno izmenjen XPath
 - struktura elementa putanje je ista



Putanje

- funkcije za navigaciju
 - collection(): imenovana sekvenca
 - doc(): koren datog XML dokumenta
 - id(): element sa datim ID-jem
 - idref(): elementi koji pokazuju na dati ID
 - root(): koren tekućeg dokumenta
- primeri:
`doc("team.xml")/Team/Player[Name/Last="Divac"]`
`collection("teams")//Player[Name/Last="Divac"]`

Promenljive

- navode se sa znakom \$ ispred imena
- ime može biti nekvalifikovano ili kvalifikovano
 - prefiks zamenjuje namespace
- promenljive nisu promenljive
tj. ne može im se promeniti vrednost

FLWOR izrazi

- centralni izraz u XQuery
- čita se „flower“
- po prvom slovu klauzula: for, let, where, order by, return
- više namena
 - za definisanje promenljivih
 - za iteraciju kroz sekvencu
 - za filtriranje rezultata
 - za sortiranje sekvenci
 - za spajanje različitih izvora podataka

FLWOR izrazi

- primer: koristi svih pet klauzula

```
for $i in doc("orders.xml")//Customer
let $name := concat($i/@FirstName, $i/@LastName)
where $i/@ZipCode = 91126
order by $i/@LastName
return
<Customer Name="{{$name}}">
  { $i//Order }
</Customer>
```

FLWOR izrazi

- **for** i **let** mogu da se pojave u bilo kom redosledu,
- i u bilo kom broju,
- sve dok postoji bar jedna **for** ili **let** klauzula
- **for** iterira kroz sekvencu
- **let** dodeljuje promenljivoj vrednost datog izraza
 - promenljive se nalaze u opsegu do kraja FLWOR izraza
- **where** filtrira
- **order by** sortira
- **return** konstruiše rezultat

FLWOR primeri

```
let $variable := "any expression here"  
return concat("xx", $variable, "xx")
```

=>

"xxany expression herexx"

FLWOR primeri

```
for $i in (1, 2, 3, 4, 5)  
where $i > 3  
return $i
```

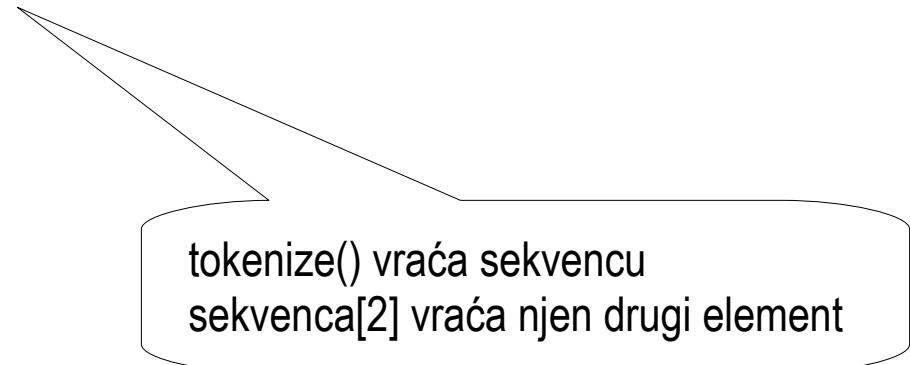
=>

(4, 5)

(: kraći zapis pomoću putanje :)
(1,2,3,4,5)[. > 3]

FLWOR primeri

```
for $e in doc("team.xml")//Employee  
let $name := $e/Name  
order by tokenize($name)[2] (: Prezime :)  
return $name
```



tokenize() vraća sekvencu
sekvenca[2] vraća njen drugi element

FLWOR primeri

```
for $i in doc("one.xml")//fish,  
    $j in doc("two.xml")//fish  
where $i/red = $j/blue  
return <fishes> { $i, $j } </fishes>
```

može i iz više dokumenata

Kvantifikacija

- operatori some i every

- skraćeni FLWOR
- vraćaju logičku vrednost
 - postoji
 - za svaki

```
some $emp in doc("team.xml")//Employee satisfies $emp/@years > 5
```

```
every $emp in doc("temp.xml")//Employee satisfies $emp/@years > 5
```

FLWOR join

1. Dekartov proizvod

```
for $i in (1, 2, 3)
for $j in (3, 4, 5)
return ($i, $j)
```

=>

(1, 3, 1, 4, 1, 5, 2, 3, 2, 4, 2, 5, 3, 3, 3, 4, 3, 5)

FLWOR join

2.inner join

```
for $i in (1, 2, 3)
for $j in (3, 4, 5)
where $i = $j
return ($i, $j)
```

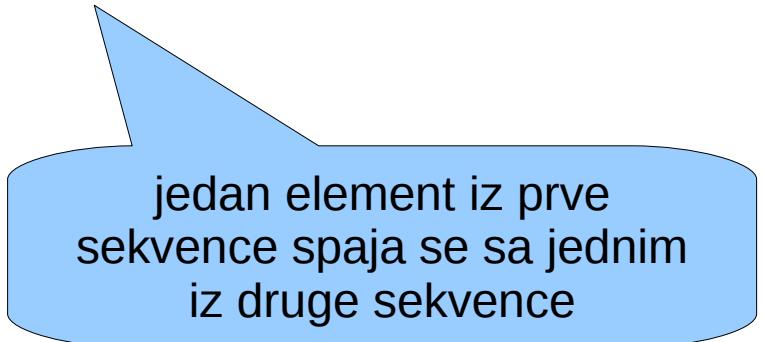
=>

(3, 3)

FLWOR join

2. inner join / one-to-one

```
for $proj in doc("projects.xml")/Projects/Project  
for $emp in doc("team.xml")//Employee  
where $proj/@owner = $emp/@id  
return $proj/Name, $emp/Name
```



jedan element iz prve sekvence spaja se sa jednim iz druge sekvence

FLWOR join

2. inner join / one-to-one

```
for $proj in doc("projects.xml")/Projects/Project  
for $emp in doc("team.xml")//Employee  
where $proj/@owner = $emp/@id  
return <Assignment>{$proj/Name,$emp/Name}</Assignment>
```

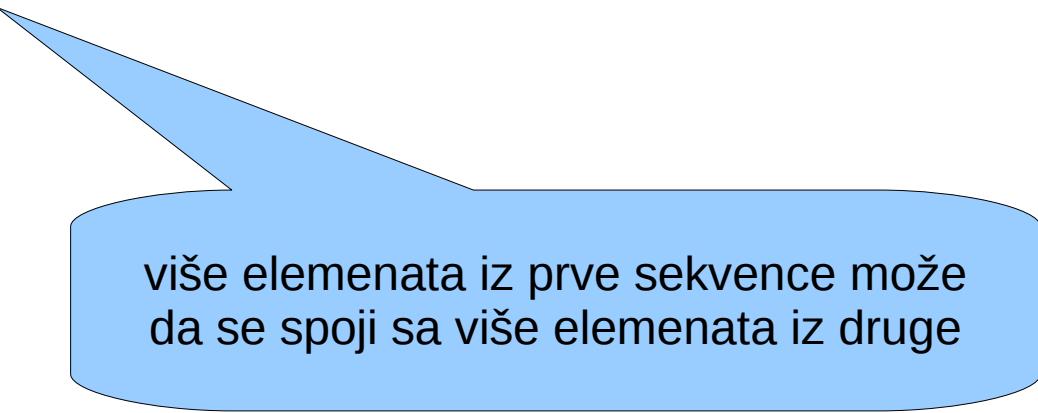


grupišemo rezultat u novi element

FLWOR join

3. inner join / many-to-many

```
for $proj in doc("projects.xml")/Projects/Project  
for $emp in doc("team.xml")//Employee  
where $proj/Category = $emp/Expertise  
return  
  
<Assignment proj="${$proj/Name}" emp="${$emp/Name}" />
```



više elemenata iz prve sekvene može da se spoji sa više elemenata iz druge

FLWOR join

3. outer join / left outer join

```
for $proj in doc("projects.xml")/Projects/Project  
return  
  
<Assignment proj="{{$proj/Name}}">{  
  
    for $emp in doc("team.xml")//Employee  
        where $emp/Expertise = $proj/Category  
  
    return $emp/Name  
}</Assignment>
```

imena
zaposlenih se
mogu ponavljati

rezultat sadrži po jednu stavku za svaki
element prve (leve) sekvene, čak i ako
nema odgovarajućeg elementa u drugoj
sekvenci

FLWOR join

3. outer join / left outer join

```
for $proj in doc("projects.xml")/Projects/Project
```

```
return
```

```
<Assignment proj="{{$proj/Name}}" emps="{"
```

```
    for $emp in doc('team.xml')//Employee
```

```
    where $emp/Expertise = $proj/Category
```

```
    return $emp/@id
```

```
}">
```

emps je tipa IDREFS, nema ponavljanja imena zaposlenih!

rezultat sadrži po jednu stavku za svaki element prve (leve) sekvene, čak i ako nema odgovarajućeg elementa u drugoj sekvenci

FLWOR join

3. outer join

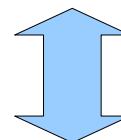
```
for $proj in doc("projects.xml")/Projects/Project  
for $emp in doc("team.xml")//Employee  
where $proj/@owner = $emp/@id  
and not($proj/Category = $emp/Expertise)  
return <Mismatch>{$emp/Name, $proj/Name}</Mismatch>
```

svi projekti gde vlasnik nema ekspertizu u toj kategoriji

FLWOR join

4. self join

```
doc("team.xml")//Employee[Title =  
    doc("team.xml")//Employee[@id="E0"]/Title]/Name
```



```
let $emp := doc("team.xml")//Employee  
for $i in $emp, $j in $emp  
where $i/Title = $j/Title and $j/@id = "E0"  
return $i/Name
```

Poređenje sekvenci

- egzistencijalno poređenje
 - $\$seq1 > \$seq2$
 - da li postoji bar jedan element u $\$seq1$ veći od bilo kog elementa u $\$seq2$
 - $\text{some } \$a \text{ in } \$seq1, \$b \text{ in } \$seq2 \text{ satisfies } \$a > \$b$
 - isto što i prethodni izraz
 - $\text{some } \$a \text{ in } \$seq1, \$b \text{ in } \$seq2 \text{ satisfies } \$a > \$b+3$
 - da li postoji bar jedna element u $\$seq1$ veći od bilo kog elementa u $\$seq2$ uvećanog za 3
 - fleksibilnija varijanta
 - $\text{exists}(\$seq1[. > \$seq2])$
 - XPath varijanta prvog izraza
 - $\text{exists(for } \$a \text{ in } \$seq1, \$b \text{ in } \$seq2 \text{ where } \$a > \$b \text{ return } \$a)$
 - puni FLWOR izraz

Poređenje sekvenci

- poređenje član-po-član
 - deep-equal() funkcija - ponaša se rekurzivno
 - shallow-equal() ne postoji kao ugrađena funkcija

```
declare function shallow-equal($seq1 as node()*,
    $seq2 as node()) as xs:boolean {
  if (count($seq1) != count($seq2))
    then false()
  else
    empty(
      for $i at $p1 in $seq1
      for $j at $p2 in $seq2
      where $p1 = $p2 and $i is not $j
      return 1
    )
};
```

Poređenje sekvenci

- univerzalno poređenje - uslov je zadovoljen za svaki element
 - primer: svaki element iz \$seq1 veći od svakog elementa iz \$seq2

```
every $a in $seq1, $b in $seq2 satisfies $a gt $b
```

```
empty(for $a in $seq1  
      where not($a > $seq2)  
      return $a )
```

```
empty($seq1[not(. > $seq2)])
```

FLWOR sortiranje

- order by klauzula u FLWOR izrazu
 - sortira se po datim ključevima

```
for $i in doc("team.xml")//Employee  
where exists($i//Employee)  
stable order by $i/@id descending  
return $i/Name
```

stavke sa jednakim sort
ključem čuvaju originalni
redosled

u opadajućem redosledu

FLWOR sortiranje

- tretiranje prazne sekvence i NaN
 - `empty least`: prazna sekvenca je manja od svake neprazne; NaN je manji od svake ne-NaN vrednosti i neprazne sekvene
 - `empty greatest`: prazna sekvenca je veća od svake neprazne; NaN je veći od svake ne-NaN vrednosti i neprazne sekvene

```
for $i in (1E0, 2E0, 3E0, 0E0 div 0)
let $key := if ($i < 2.5) then $i else ()
order by $key empty least, $i descending
return $i
```

=>

(2E0, 1E0, NaN, 3E0)

FLWOR grupisanje

- nema posebnog group by operatora
- rezultujući XML predstavlja grupisanje
 - primer: pronaći zaposlene koji imaju svoje podređene i sortirati ga po broju podređenih u opadajućem redosledu

```
for $i in doc("team.xml")//Employee
let $reports := count($i/Employee)
where $reports > 0
order by $reports descending
return
<Employee name="{{$i/Name}}" reports="{{$reports}}"/>
```

Obrada grešaka

- staticka ili dinamička - zavisno od implementacije
 - npr. xs:decimal("X") - može biti prijavljeno prilikom kompajliranja ili izvršavanja
 - \$a + \$b - ako je \$a ili \$b sekvenca sa više od jednog elementa, ili ako se vrednosti ne mogu sabrati
 - "1"+2 će izazvati grešku
 - u XPath-u je to validan izraz: "1" se konvertuje u ceo broj, pa se onda sabere sa 2
- funkcija error() - za programsko izazivanje greške
- funkcija trace() - za generisanje poruke o grešci bez prekidanja izvršavanja

Service-Oriented Computing (SOC) i Service-Oriented Architecture (SOA)

Delimično bazirano na predavanju:
W. T. Tsai, Department of Computer Science and Engineering
Arizona State University

Definicija servisa (usluge)

- Servis (usluga) predstavlja funkcionalnu celinu - posao koji davalac usluge obavlja sa ciljem da postigne neki željeni rezultat za korisnika usluge.
- I davalac i korisnik usluge su uloge koje obavljaju softverski agenti (komponente).
 - dakle povezujemo softverske komponente
- Servis (usluga) u SOA može biti web service (WS), ali to nije obavezno. Bilo koja celovita dobro definisana komponenta, ne neophodno na webu, može se koristiti kao deo SOA aplikacije.

Definicija SOA

- Servisno orijentisane arhitekture – (*Service Oriented Architecture - SOA*) predstavlja sistem koji se sastoji od modularnih softverskih komponenti, sa **jasno definisanim i standardizovanim načinom pristupa i interfejsima** koje su nezavisne od specifične platforme i tehnologije implementacije.
- U najjednostavnijem obliku, SOA predstavlja kolekciju standardizovanih servisa (usluga) dostupnih preko mreže, koji međusobno komuniciraju u kontekstu izvršenja poslovnog procesa.

Kako se došlo do SOA?

- Evolucijom distribuiranih sistema
 - interfejsi kojima se može pristupiti sa udaljene lokacije su prethodno bili posledica razvoja komponenti aplikacije (pozivanje metoda na udaljenim objektima - **Remote Procedure Call**)
- SOA menja paradigmu, sistemi se osmišljavaju tako da se aplikacija “sklapa” od modula koji obavljaju jasno definisane usluge, i čine ih dostupne preko javnog interfejsa.
- Ključna razlika u odnosu na RPC - distribuirane komponente su “slabo povezane” (*loosely coupled*) - interfejs komponenti ne zavisi od tehnologije implementacije - komunikacija se tipično obavlja razmenom poruka.

Zašto SOA?

- SOA omogućava rešavanje izazova razvoja velikih (enterprise) poslovnih aplikacija:
 - Pojednostavljuje prilagođavanje aplikacije promeni poslovnih zahteva.
U poređenju sa monolitnom arhitekturom - lakše je dodati novu funkcionalnost dodavanjem novog “loosely coupled” modula.
 - Omogućava korišćenje postojeće infrastrukture i aplikacija.
Postojeće aplikacije se povezuju preko postojeće mreže putem servisnih modulja, umesto da se sve piše od nule, što bi remetilo obavljanje poslova.
 - Omogućava otvaranje novih kanala za interakciju sa klijentima, partnerima, dobavljačima...

Osnovne komponente SOA

- Servisni orijentisani sistem može biti i krajnje jednostavan - jedan klijent koji koristi uslugu druge komponente. Ali mogu biti i vrlo kompleksni sa mnogo komponenti koje se integrišu preko neke *bus* infrastrukture.
- U opštem slučaju SOA predviđa da će sistem da se sastoji od:
 - **Service providers** - ponuđač usluge (servisa) nudi pristupnu tačku (endpoint) i opisuje koje funkcionalnosti se mogu obaviti preko nje.
 - **Service consumers** - klijenti servisa - kada imaju potrebu kreiraju zahteve ka servisnim tačkama i koriste rezultate koje dobiju kao odgovor servisa.
 - **Service registry** - registar servisa, omogućava klijentima da "pronađu" servise kada im nije unapred poznata pristupna tačka.
- **Servisne arhitekture počivaju na korišćenju dogovorenih i usvojenih standarda!**

SOA nije (samo) Web Service

- *Web services != service-oriented architecture.*
Web servisi predstavljaju kolekciju tehnologija, koja uključuje XML, SOAP, WSDL (i UDDI), naknadno i REST koja omogućava da se povežu programska rešenja u specifičan sistem razmene poruka i omogućavaju integraciju aplikacija.
- Web servis je samo jedna implementacija principa SOA.
- Tokom vremena ovakva rešenja sazrevaju, ali i vremenom bivaju zamjenjena novijim, robusnijim ili efikasnijim rešenjima.
- Ovakva rešenja predstavljaju implementacija SOA principa - "dokaz koncepta" da SOA pristup ima svoju implementaciju.

Razlika u odnosu na Web Service (nast.)

- SOA je samo tip arhitekture sistema.
- Ne predstavlja bilo koji specifičan skup tehnologija, kao što su Web servisi – ona je na višem nivou apstrakcije.
- U idealnom svetu SOA je potpuno nezavisna od implementacione tehnologije.
- U poslovnom smislu, čista definicija SOA bi mogla da se iskaže kao: “arhitektura aplikacija u kojoj se sve funkcije definišu kao nezavisni servisi (usluge) sa dobro definisanim interfejsima preko kojih je moguće pokrenuti izvršavanje date funkcionalnosti (*invokable interface*).”
- Ove interfejse je moguće pozivati u željenom redosledu kako bi se formirao poslovni proces”. Ključne odrednice:
 - Sve funkcije se definišu kao servisi(usluge)
 - **Svi servisi su nezavisni.**
 - U najopštijem obliku svi interfejsi omogućavaju pokretanje funkcija

Ključne postavke SOA

- Softverski sistem se opisuje kao skup servisa nezavisnih od implementacione tehnologije
- Interakcija sa servisom se implementira razmenom poruka
- SOA podrazumeva postojanje ponuđača servisa i klijenta koji koristi (konzumira) servis
- Bilo koji sistem koji učestvuje u komunikaciji može biti ponuđač i klijent ili da nastupa u obe uloge u zavisnosti od poslovnog procesa (redosleda pozivanja servisa)
- Servisi i same poruke su **stateless**
- Servis i klijent najčešće nisu implementirani u istoj tehnologiji
- SOA najčešće obuhvata servise, register dostupnih servisa (mogućnost otkrivanja servisa - *service discovery*), i javno dostupan opis servisa (*public contract*) koji omogućava klijentu da se podesi i poveže na servis (*service negotiation*).

Po čemu se SOA razlikuje od klijent-server arhitekture

- Servisi su *stateless* i dostupni putem javnog interfejsa
- Interakcija se ostvaruje “slabim povezivanjem”
- klijent-server je zahtevao uspostavljanje “čvrste” veze između klijenta i servera

Šta servis mora da obezbedi

- Konzistentnost dizajna SOA sistema se obezbeđuje kroz:
 1. Standardizovan “ugovor” servisa
 2. “Slabu” povezanost između klijenta i servisa i između servisa međusobno.
 3. Apstrahovanje implementacionih detalja - mogućnost klijenta da sarađuje sa servisom nikako nije uslovljena tehnologijom koju on koristi da razvije klijentsku stranu - on samo mora da ispoštuje “servisni ugovor”.
 4. Mogućnost da se složeniji servisi dobiju kompozicijom već postojećih.
 5. Autonomnost u odnosu na izvršno okruženje.
 6. Servisi garantuju ***statelessness***.
 7. Servisi su ponovo iskoristivi.
 8. Moguće ih je “otkriti” (pretraživanjem na osnovu tipičnih metapodataka).

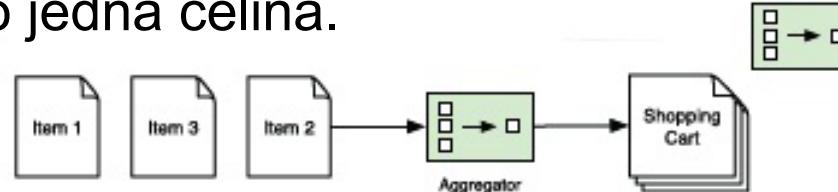
SOA šabloni (*patterns*)

- Kao i u mnogim drugim oblastima, i za SOA postoje dizajn šabloni (*design patterns*) koji se koriste kao dokazana dobra rešenja za određene probleme
 - osnovni servisni šabloni
 - šabloni koji predstavljaju standardna rešenja, i koriste se kao gradivni za složenije šablonе
 - šabloni arhitekture
 - predstavljaju šablonе karakteristične za određena dizajnerska rešenja u vezi sa implementacijom SOA arhitektura
 - kombinovani šabloni
 - agregacija osnovnih šablona kako bi se definisala povezana reprezentacija sistema. Šabloni se po pravilu ne koriste izolovano, niti su sami sebi cilj. Podsistemi se najčešće formiraju kombinovanjem dva ili više šablona.

slike i opis šablon preuzeti sa: <https://dzone.com/refcardz/soa-patterns>

SOA šabloni - osnovni šabloni

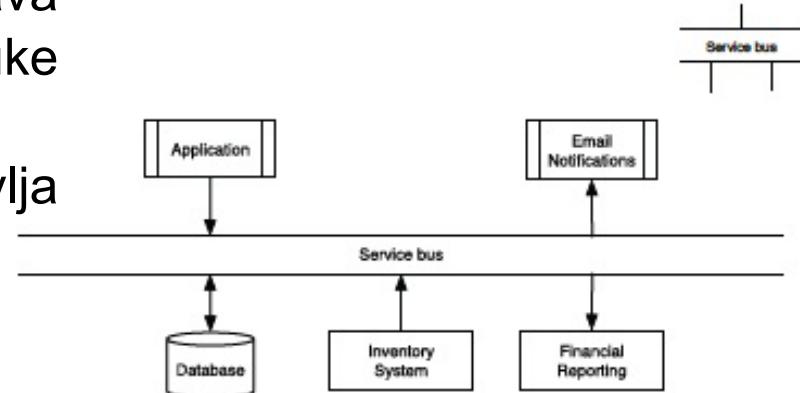
- **Agregator** - kombinuje više individualnih poruka kako bi se obrađivale kao jedna celina.



- **problem**: *Stateless* poruke neće stići do servisne tačke u nekom unapred željenom redosledu. Različite poruke mogu biti obrađivane od strane različitih servisa različitom brzinom. SOA sistemi garantuju isporuku poruka, ali ne garantuju određeni redosled.
- **rešenje** - definiše se aggregator koji prima tok podataka i grupiše određene poruke kako bi ih poslao na dalju obradu kao jednu celinu. Očigledno je da je logika aggregatora statefull kako bi utvrdio koje poruke treba agregirati u jedan paket, ali se isporuka i dalje obavlja u *stateless* režimu.
- **primena** - za potrebe daljeg rutiranja i procesiranja, grupisanje poruka koje se razmenjuju preko *service bus-a* na osnovu njihovog tipa ili nekog zajedničkog svojstva
- **rezultat** - fleksibilnost sistema, jer individualni servisi mogu i dalje asinhrono obrađivati poruke

SOA šabloni - osnovni šabloni

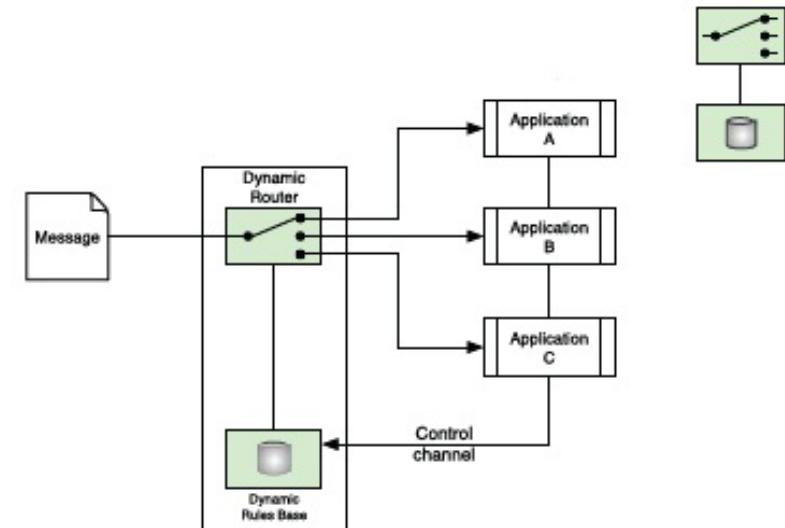
- **Service Bus** - servisna magistrala - komunikacioni kanal kojim se omogućava da jedna emisiona tačka proizvodi poruke koje mogu da "pokupe" jedna ili više izlaznih tačaka. Moguće je i da se obavlja obrada poruka dok prolaze kroz komunikacioni kanal.



- **problem:** Aplikacije moraju da komuniciraju međusobno, a ponekad koriste različite prokole i tehnologije za komunikaciju. Jednostavna rešenja kao *point-to-point*, *dedicated conduit*, povećavaju kompleksnost sistema, vreme implementacije, i povećavaju probleme pri integrisanju jer zahtevaju *tight-coupling*.
- **rešenje** - obezbediti prenosni put koji je protkol-neutralan i neutralan u odnosu na tip podataka koji se prenose, sa apstraktnim ulaznim i izlaznim tačkama za povezivanje sa postojećim aplikacijama nezavisno od njihove tehnologije.
- **primena** - za integriranje heterogenih sistema, omogućavanje interoperabilnosti novih i starih (*legacy*) sistema, apstrahovanje protokola komunikacije
- **rezultat** - Message-Oriented Middleware (MOM): *publish/subscribe queuing* i *enterprise service buses*.

SOA šabloni - osnovni šabloni

- **Dynamic Routing** - Mehanizam za efikasno prosleđivanje poruka na jednu ili više destinacija na osnovu konfigurabilnih nefiltrirajućih pravila koja se primenjuju na sadržaj poruka.

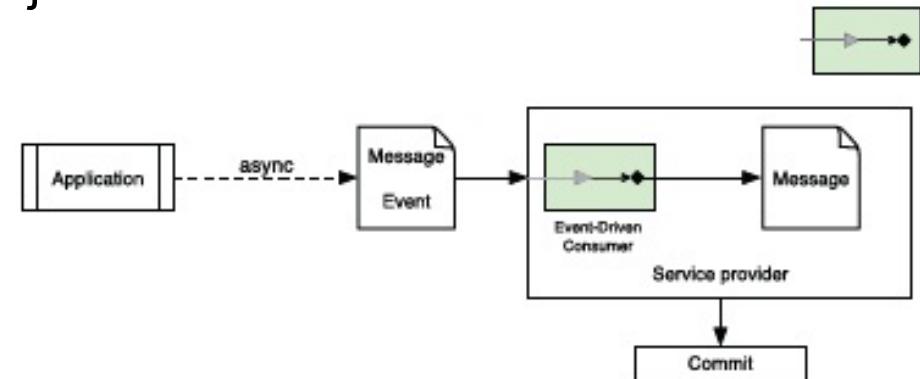


- **problem** - rutiranje poruka na osnovu filtera je neefikasno jer se poruke prosleđuju do svake destinacije i ruteru da bi se tamo filtrirale i proverile, bez obzira da li taj čvor u mreži uopšte može da procesira takvu poruku.
- **rešenje** - kreirati ruter koji sadrži informacije i o pravilima za filtriranje i znanje o krajnjim destinacijama tako da se poruke isporučuju samo onim tačkama koje su u stanju da ih obrade. Za razliku od filtera ovakvi ruteri ne menjaju sadržaje poruka i samo utiču na utvrđivanje destinacije poruke.
- **primena** - Isporuka poruka na osnovu podataka specifičnih za određenu aplikaciju npr. na osnovu klijenta, tipa poruke itd.
- **rezultat** - poboljšana isporuka poruka i performanse sistema, ali an račun povećanja kompleksnosti sistema (ruter mora imati složeniju logiku i dodatnu bazu znanja). Idealan za *decoupling* aplikacija jer one same više ne moraju znati kome bi trebale slati poruke.

SOA šabloni - osnovni šabloni

▪ **Event-Driven Consumer** - okruženje

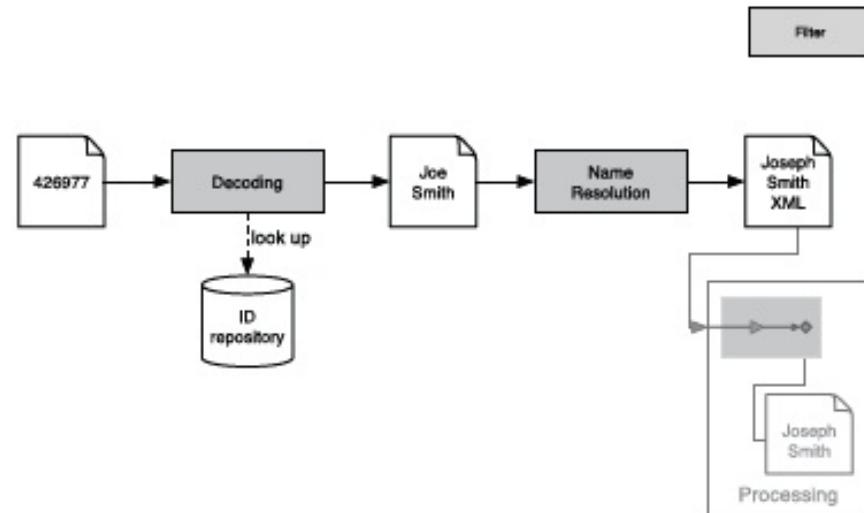
u kome se poruke odmah dostave određenim *service providerima* u momentu kada se pojave na komunikacionom kanalu.



- **problem** - Sistemi za razmenu poruka bazirani na blokirajućim listenerima ili prozivanju (*polling*) koriste bezrazložno odredene resurse ukoliko je komunikacioni kanal prazan. Ciljna komponenta u blokirajućem stanju bespotrebno troši niti (*thread-ove*) koje bi sistem mogao koristiti za obavljanje drugih zadataka.
- **rešenje** - implementacija *callback* mehanizma na nivou komunikacione magistrale ili aplikacije koji se automatski budi kada se dolazna poruka pojavi na komunikacionom kanalu. Komunikacioni sistem može ovo pozivati sinhrono ili asinhrono. (Faktički *message event-listener*).
- **primena** - distribuirani sistemi sa promenljivim brojem konzumenata i provajdera, i promenljivim stepenom iskorišćenja CPU-a, koji je direktno zavisан od sadržaja i količine poruka, kao i sistemi koji zahtevaju veliku skalabilnost.
- **rezultat** - procesiranje poruka se skalira linearno sa brojem distribuiranih poruka. Niti (*threads*) konzumiraju poruke čim postanu dostupne i odmah nakon toga se niti oslobođaju. Bolja je iskorišćenost procesnih resursa.

SOA šabloni - osnovni šabloni

- **Filter** - deo prenosnog puta koji po određenim pravilima uklanja delove poruka ili primenjuje funkciju obrade dok poruke prolaze od izvora ka destinaciji.

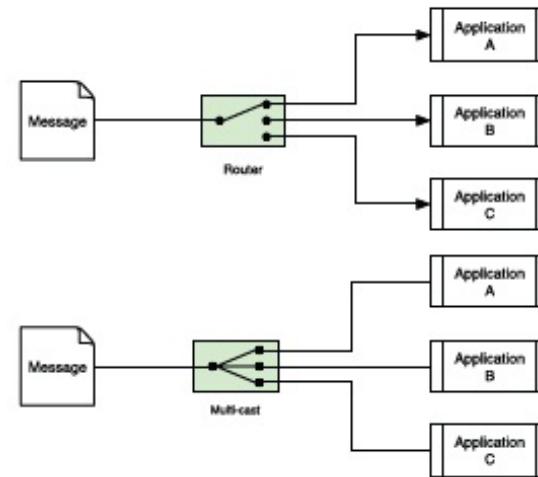


- **problem** - potreba da se implementira fleksibilna obrada poruka između sistema koji šalju i primaju poruku, na platformski nezavisan način i pazeći da se ne uvede striktna međuzavisnost sistema.
- **rešenje** - implementirati komponentu na prenosnom putu sa relativno jednostavnim interfejsom za obradu - transformaciju ulaznih i izlaznih poruka (ulaz/obrada/izlaz), koji se zasniva na principu primene određene transformacione funkcije ili *pipeline* obrade - kaskadnom primenom filtera. Svi primjenjeni filteri moraju da dele isti eksterni interfejs kako bi se lako kombinovali.
- **primena** - upotreba diskretnih funkcija na poruke kao što su enkripcija, konsolidacija poruka, eliminacija redundantnih stvari, validacija podataka, itd. Filteri omogućavaju da se kompleksnije obrade rastave na jednostavne obrade koje se mogu kombinovati po potrebi.
- **rezultat** - filteri eliminišu međuzavisnosti sistema i njihovu zavisnost od određenih vrsta podataka, primenjujući jednostavan princip obrade baziran na jasno opisanom *contractu* (inbound/outbound interface). Filteri se naknadno lako kombinuju u redosledu koji je pogodan da se postignu željene obrade, a bez potrebe da se menja logika samog filtera)

SOA šabloni - osnovni šabloni



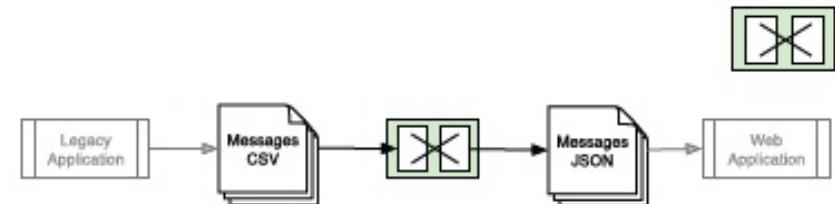
- **Router** - uopšteni mehanizam za prosleđivanje poruka do jednog ili više korisnika na osnovu konfigurabilnih pravila ili filtera koji se primenjuju na sadržaj poruke



- **problem** - Aplikacija treba da ostvari vezu sa *endpoint*-om jedne ili više drugih aplikacija, ali bez potrebe da se ostvari “čvrsto” povezivanje.
- **rešenje** - kreirati prenosni put takav da omogući isporuku na osnovu konfigurabilnih pravila koja se zasnivaju na utvrđivanju sadržaja poruke, filtriranju podataka, ili na osnovu sadržaja poruke. Rutiranje može biti sekvensijalno (*endpoint*-i poruku primaju jedan po jedan), ili paralelno (svim *endpoint*-ima se poruka prosleđuje u praktično istom momentu).
- **primena** - dostavljanje poruka na service bus, raspoređivanje poruka, *proxy*-ji za poruke, integracioni slojevi aplikacija i drugi sistemi gde poruke treba da budu isporučene *endpoint*-ima koji se utvrđuju na osnovu određenih pravila.
- **rezultat** - apstrakcija rutera je inherentno upotrebljena u svim SOA sistemima, bez obzira o kom tipu implementacije se radi, jer oni obezbeđuju jednostavan, a moćan sistem za isporuku poruka koji ne zavisi od pojedinačne implementacije sistema.

SOA šabloni - osnovni šabloni

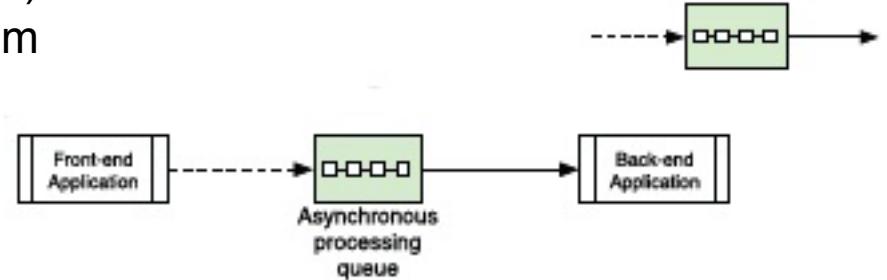
- **Translator / Transformer** - mehanizam za konvertovanje sadržaja poruka iz jedne reprezentacije u neku drugu tokom njenog prolaska kroz komunikacioni sistem



- **problem** - pri integraciji heterogenih sistema često se susreće problem da se skoro isti podaci reprezentuju na različite načine.
- **rešenje** - obezbediti sistemski nezavisan mehanizam za promenu strukture poruka i njihovih metapodataka pre nego što se ista poruka isporuči na destinaciju.
- **primena** - prevođenje poruke na mestu aplikativnog *endpoint-a* jer su ove transformacije zavisne od posmatrnog sistema i protokola (za razliku od filtera).
- **rezultat** - Translator (prevodilac) je jedan od najefikasnijih mehanizama za transformisanje poruka, jer omogućava da se razvoj jedne aplikacije može obavljati i izolovano, a njene ulaze/izlaze prilagoditi po potrebi za razmenu sa drugim aplikacijama ili specifikacijom za komunikacioni kanal (faktički se adapter postavlja kao *front end* aplikacije ka sloju za razmenu poruka).

SOA šabloni - šabloni arhitekture

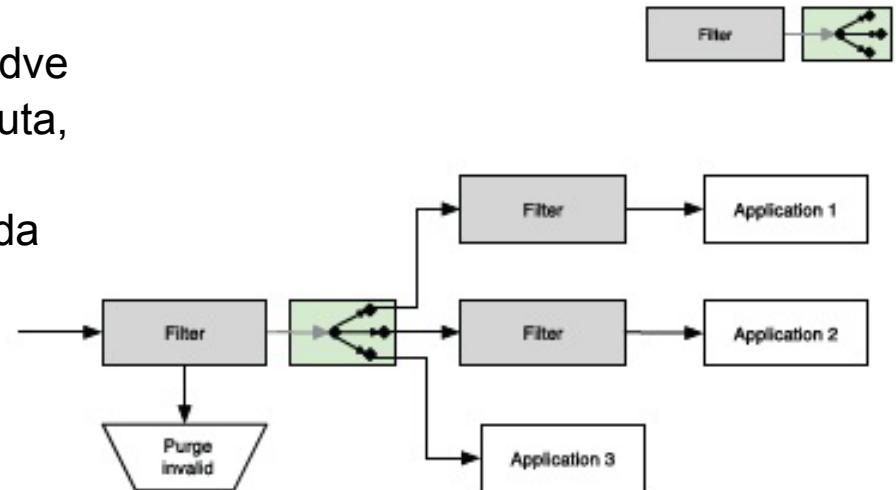
- **Asynchronous Processing** - mehanizam formiranja **reda poruka** (*message queue*) između jedne ili više krajnjih tačaka. Ovim se "rasparuje" vreme procesiranja i korišćenje resursa na obe strane.



- **problem** - sinhrono procesiranje bi u većini slučajeva rezultovalo lošim performansama i smanjenjem pouzdanosti sistema.
- **rešenje** - klijenti razmenjuju poruke sa servisom kroz red poruka - red za procesiranje. Na ovaj način razdvaja se *front-end* (prijem poruke) od *back-end-a* (same obrade poruke). Faktički omogućava da se brzine prijema i obrade poruka (čak i znatno) razlikuju.
- **primena** - bilo koja aplikacija kojoj je potrebna mogućnost nezavisnog skaliranja kapaciteta za prijem i obradu poruka. Npr. aplikacija za konsolidaciju podataka (back-end procesiranje je dugotrajno), obrada porudžbina na e-commerce aplikacijama...
- **rezultat** - redovi su opšteprihvaćen koncept za integraciju distibuiranih sistema, dobro se sklairaju i horizontalno i vertikalno. Postoji veliki broj raspoloživih implementacija.

SOA šabloni - šabloni arhitekture

- **Bridge** - (most) mehanizam povezivanja dve ili više aplikacija preko istog prenosnog puta, ali kada one koriste različite protokole, a moguće je da je potrebna i analiza i obrada poruka dok prolaze kroz komunikacioni kanal.

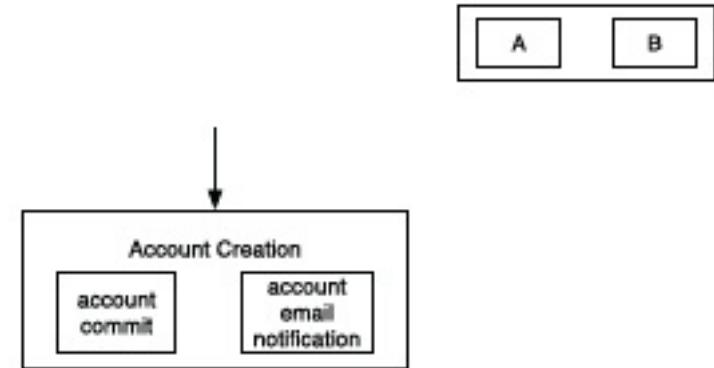


- **problem** - komunikacione tačke aplikacija koje treba povezati mogu biti u različitim delovima enterprise mreže, mogu koristiti različite protokole, ili je potrebna obrada na osnovu specifičnih atributa poruke.
- **rešenje** - kreirati "most" između aplikacija koji obezbeđuje odgovarajući mehanizam za rutiranje poruka, njihovo filtriranja, adaptaciju protkola između dve strane...
- **primena** - SOA proxy između aplikativnih krajinjih tačaka u srednjem sloju, ili proxy-ji za pritupne tačke na cloud-u, upravljanje ESB-om
- **rezultat** - dobro rešenje za proširenje aplikacija i širenje njihovih međusobnih interkonekcija, jer se razvoj koncentriše samo u srednjem (integracionom) sloju, dok se aplikacije ne menjaju. Tipično se koristi da omogući povezivanje starijih aplikacija na novije sisteme.

SOA šabloni - šabloni arhitekture

▪ **Cross-Service Operation** - mehanizam

koji obezbeđuje koordinaciju više aktivnosti
koje se sastoje od više servisa, pri čemu
se garantuje završetak i/ili roll-back.

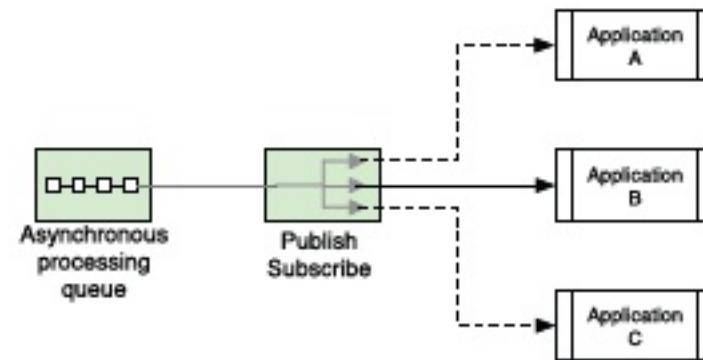
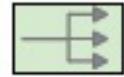


- **problem** - dva ili više servisa, koji mogu biti i u različitim sistemima, moraju se izvršiti uspešno. Ukoliko se bilo koji ne završi uspešno, svi servisi koji su s njim u vezi moraju se vratiti u prethodno stanje (roll-back) kako bi se u maksimalno mogućoj meri očuvao integritet aplikacije.
- **rešenje** - osnovni servisi mogu se “umotati” u dodatni servis koji obezbeđuje proveru integriteta i uspešan završetak funkcionalnosti servisa, ili obezbeđuje *gracefull degradation* ukoliko bilo koji od osnovnih servisa otkaže (ne obavi svoj posao uspešno).
- **primena** - u svim transakcionim sistemima
- **rezultat** - primena ovog rešenja može zahtevati dodatne komponente “transakcione procesore” kako bi se obezbedila tražena funkcionalnost ka ostatku SOA infrastrukture. U svakom slučaju potrebni su dodatni resursi kako bi se privremeno čuvalo prethodno stanje za svaki od osnovnih servisa i omogućio roll-back.

SOA šabloni - šabloni arhitekture

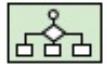
▪ **Event-Driven Dispatching** - mehanizam

koji obezbeđuje rutiranje poruka konzumentima na osnovu reakcije na odrešene događaje koje su proizvele aplikacije koje su deo SOA sistema.

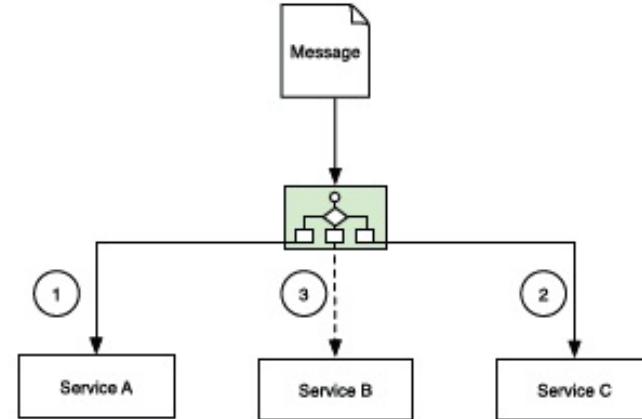


- **problem** - konzumenti poruka moraju procesirati poruke u momentu kada postanu dostupne u sistemu, ali princip "prozivanja" da bi se proverilo postojanje poruke nije više dovoljno efikasan.
- **rešenje** - klijenti (konzumenti) se implementiraju kao blokirajuće, reentrant aplikacije koje se pretplaćuju na određeni komunikacioni kanal. Ove aplikacije ostaju u stanju čekanja sve dok se ne pojavi poruka koja ih "budi" - pojavu poruke objavljuje SOA sistem svim onima koji su na taj komunikacioni kanal pretplaćeni.
- **primena** - sistemi za asinhronu ramenu poruka bazirani na principu preplate i objavljivanja informacija (publish/subscribe)
- **rezultat** - Obaj šablon se dobro primenjuje na osnovne servise, ali se teže primenjuje na cross-service operacije.

SOA šabloni - šabloni arhitekture



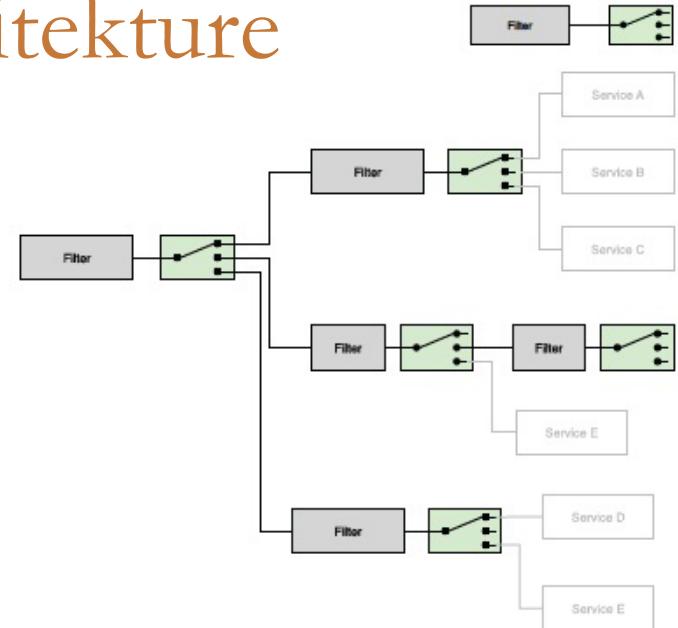
- **Process Aggregation** - metod za kombinovanje dva ili više nesekvencijalna, muđusobno zavisna koraka u izvršavanju poslovnih procesa.



- **problem** - više servisa je neophodno kako bi se kompletirala određena poslovna operacija, ali nisu svi unapred poznati, sekvenca koraka koju je potrebno izvršiti može zavisiti od raznih poslovnih pravila. Nije neophodno da se apsolutno svi osnovni servisi izvrše (nije neophodna transakcionalna kompletност).
- **rešenje** - procesni servis izvršava osnovne servise, čuva interno stanje, utvrđuje sledeće neophodne korake i na kraju obezbeđuje sinhroni ili asinhroni odgovor krajnjem klijentu.
- **primena** - sistemi u kojima se izvršava više paralelnih procesa, ali koji nisu transakcionalni, ili imaju neku kombinaciju transakcionalnih i netransakcionalnih komponenti.
- **rezultat** - agregacija procesa obezbeđuje veliku fleksibilnost, ali je dosta teška za realizaciju - neophodno je stoga rastaviti složen sistem na manje klaster aplikacija (koristeći cross-service ili agregaciju) na osnovu sličnih funkcionalnosti, ili na osnovu istih zahteva po pitanju sinhronog rada ili na osnovu nekog drugog kriterijuma.

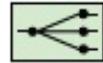
SOA šabloni - šabloni arhitekture

- **Routing and Filtering** - formalni mehanizam za rutiranje poruka ka krajnjim tačkama aplikacija preko više tačaka.

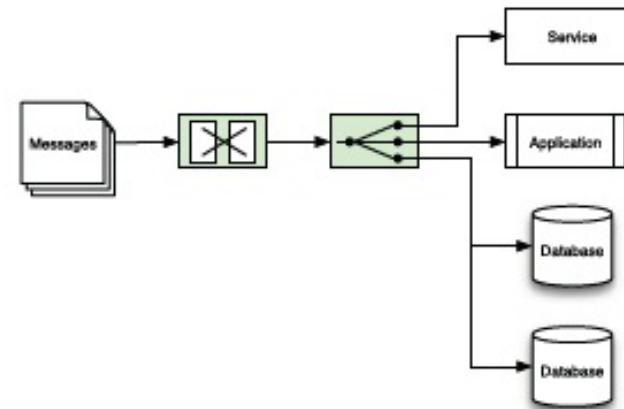


- **problem** - neophodno je proslediti poruke različitim aplikacijama, pri čemu se filtriranje obavlja na osnovu sadržaja poruka, atributa, protokola ili svega ovoga zajedno.
- **rešenje** - formirati formalni mehanizam za rutiranje poruka rekurzivnom upotrebom filtera i ruteru.
- **primena** - sistemi bazirani na pravillima, sistemi za obradu radnih tokova (workflow), dispečeri u event-driven sistemima.
- **rezultat** - rekurzivni obrazac formiranja sistema olakšava njegovo upravljanje. Neophodno je obratiti pažnju da je potrebno formalizovati redosled filtera i ruteru (njihovu hijerarhiju) kako bi se izbeglo nepotrebno uparivanje aplikacija ili ciklično rutiranje.

SOA šabloni - šabloni arhitekture



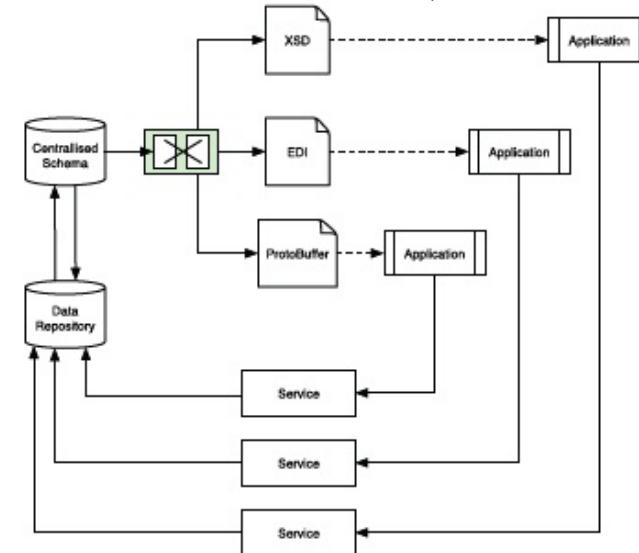
- **Replicator** - poruke ili njihov informacioni sadržaj se repliciraju na više *endpointa* sa identičnom konfiguracijom.



- **problem** - rasporeni (*decoupled*), horizontalno skalabilni servisi se “zaglavljaju” na uskom grlu prouzrokovanim potrebom da pristupaju deljenom resursu.
- **rešenje** - replikatori poruka ili podataka se implementiraju kao deo sistema za prenos podataka kako bi različite aplikacije mogle da im pristupaju istovremeno.
- **primena** - u svim aplikacijama gde je potrebno povećati propusnu moć sistema za readonly resurse (podatke ili poruke koje se prenose kroz sistem).
- **rezultat** - jako dobar način za povećanje skalabilnosti sistema, ali može doneti povećanje troškova i bespotrebno komplikovanje sistema ukoliko se dodavanje replikatora ne kontroliše (radi nasumično).

SOA šabloni - složeni (kombinovani) šabloni

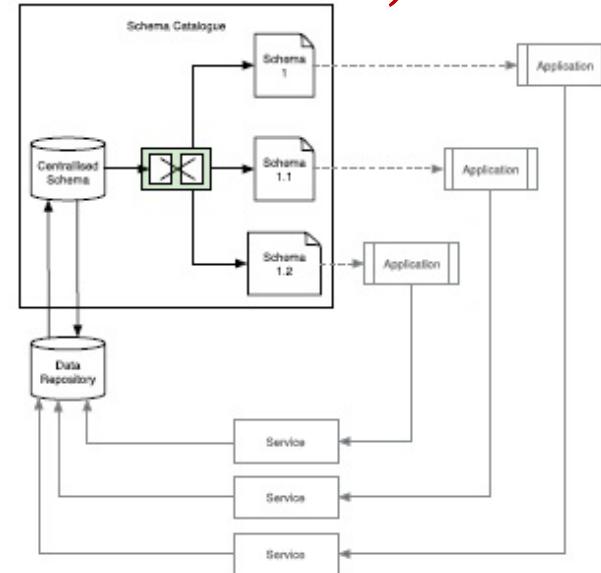
- **Centralized Schema** - definiše načine i uslove pod kojima je moguće deliti šeme podataka preko granica pojedinih aplikacija kako bi se izbegla redundantna reprezentacija podataka (i servisa).



- **problem** - slični skupovi podataka se obrađuju od strane različitih servisa ili aplikacija što može da rezultuje nejasnim servisnim "ugovorima" ili šemama baza podataka koje nepotrebno repliciraju i na nekonzistentan način čuvaju podatke.
- **rešenje** - definisati opsežnu šemu podataka - entitete koji su odvojeni od samih servisa i od fizičke reprezentacije podataka u pojedinim sistemima.
- **primena** - u svim web servisima baziranim na principu "contract first", bez obzira na tehnologiju kojom se implementira (JMS, SOAP...) u kojima više sistema može slati, procesirati ili snimati podatke.
- **rezultat** - lako se implementira ako projektanti donešu dobru odluku da odvoje šemu podataka od samih servisa koji ih koriste. Dobra implementacija centralizovane šeme podataka može generisati više različitih formata podataka koje iako su međusobno nekompatibilne, sve imaju korektno mapiranje na model podataka.

SOA šabloni - složeni (kombinovani) šabloni

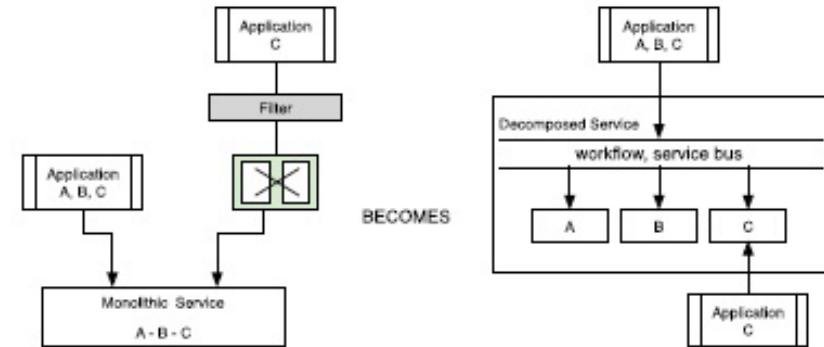
- **Concurrent Contracts** - metod koji omogućava da više različitih klijenata (konzumenata) sa različitim apstrakcijama ili implementacijama pristupnih tačaka mogu simultano koristiti isti servis.



- **problem** - servisni “ugovor” u svom osnovnom obliku možda nije pogodan za sve potencijalne korisnike servisa.
- **rešenje** - za isti servis može se obezbediti više “ugovora”, svaki sa različitim nivoom apstrakcije kako bi se servis učinio pogodnim za različite sisteme.
- **primena** - sistemi u kojima različiti korisnici servisa imaju različite nivoe potreba prema istom servisu.
- **rezultat** - ovaj šablon se lako implementira, ali ako je u kombinaciji sa centralizovanom šemom, i ako se iskoriste automatizovane transformacije ili sistemi bazirani na pravilima kako bi se automatski izgenerisali različite reprezentacije servisnog “ugovora”. Postaje komplikovan ako se za svaku verziju ugovor pravi ručno.

SOA šabloni - složeni (kombinovani) šabloni

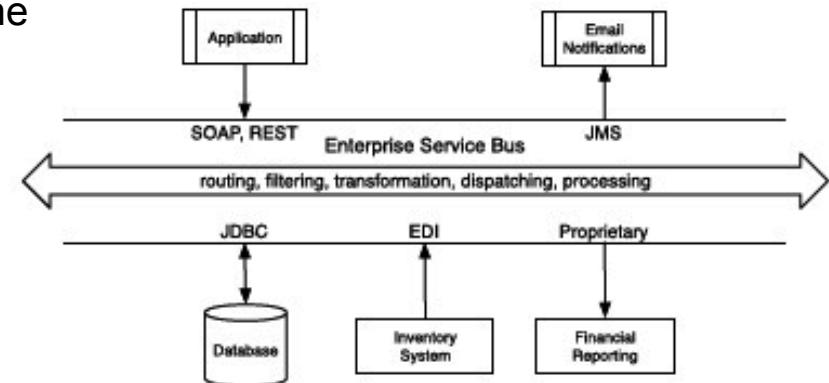
- **Decompose Capability** - “mogućnost dekomponovanja” - način da se sistem (re)dizajnira tako da se izbegne nepovoljan uticaj moguće naknadne funkcionalne dekompozicije do koje može doći ako sistem “naraste” ili se desi promena poslovnih zahteva.



- **problem** - može doći do potrebe da se servis dekomponuje, a da se pri tome njegova ukupna funkcionalnost ne naruši (uključujući i sam servisni ugovor).
- **rešenje** - neophodno je održati fizičku separaciju između šema podataka i definicije servisa, pri čemu se oni kombinuju samo kada se formira definicija specifične implementacije servisa. Ovakvom separacijom i podaci i servisi mogu se menjati nezavisno jedni od drugih. Definišu se evolucione promene servisa koristeći postojeće servise i osnovne šablone (filteri, rutiranje, transformacije).
- **primena** - u evolutivnom razvoju velikih, i kritičnih sistema koji obezbeđuju nove funkcionalnosti kako se sistem razvija. Bilo koja aplikacija kod koje je dodavanje funkcionalnosti dovelo do “naduvavanja” kompleksnosti komponenti ili do “uskih grla” u performansama sistema.
- **rezultat** - dekompozicija po pravilu rezultuje u novoj topologiji servisa, pri čemu se zadržava originalna funkcionalnost, a da se istovremeno dodaju nove funkcionalnosti. Dekompozicija bi trebalo da je neprimetna za klijente, ali da rezultuje u modularizaciji servisa.

SOA šabloni - složeni (kombinovani) šabloni

- **Enterprise Service Bus** - komunikacioni kanal koji obezbeđuje isporuku poruke od jedne ulazne tačke do jedne ili više izaznih tačaka, obezbeđujući pri tome upravljanje protokolima, filtriranje poruka, transformacije i rutiranje i eventualno dodatno procesiranje poruka

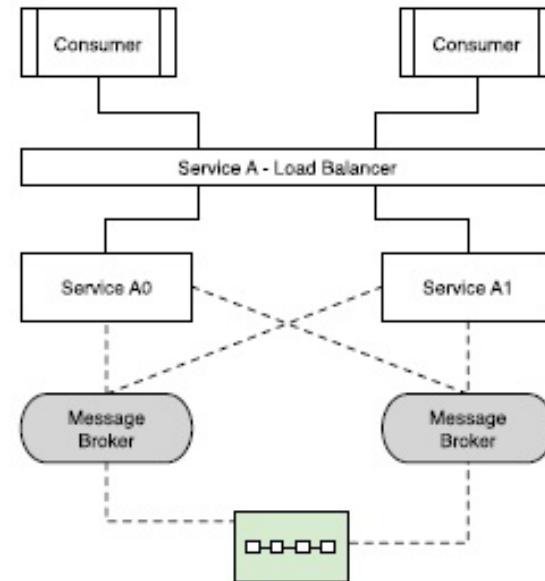


- **problem** - potreba da aplikacije međusobno komuniciraju, a da pri tome možda koriste i različite protokole i tehnologije. Proste implementacije preko point-to-point pristupa bi sa iole većim brojem aplikacija brzo postale prekomplikovane.
- **rešenje** - obezbediti prenosni put koji je neutralan po pitanju podataka i protokola sa apstraktno definisanim ulaznim i izlaznim tačkama, omogućavajući aplikacijama da se povežu bez obzira na njihovu internu tehnologiju.
- **primena** - integracija poslovnih sistema, integracija heterogenih sistema, apstrakcija protokola, interoperabilnost starih i novih sistema.
- **rezultat** - postoji veliki broj rešenja koji implementiraju ovaj princip, najčešće pod nazivima Message-Oriented Middleware (MOM): publish/ subscribe queuing and enterprise service buses.

SOA šabloni - složeni (kombinovani) šabloni

▪ **Fault-Tolerant Service Provider** -

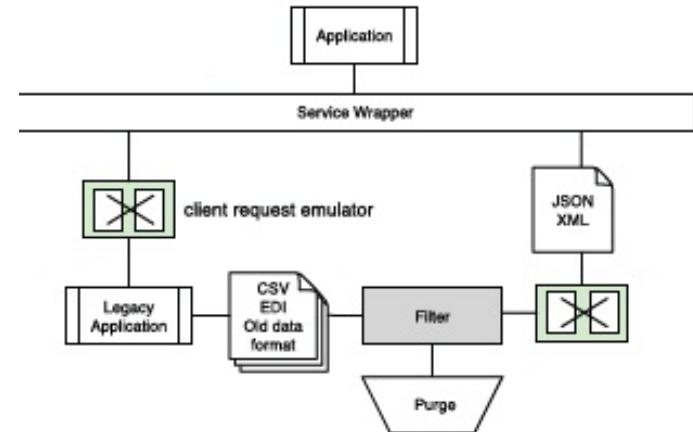
mehanizam za implementaciju servisnih platformi tako da se postigne skoro nulto vreme u kome je servis nedostupan u slučaju otkaza servisa ili cele platfrome.



- **problem** - poslovno kritične SOA aplikacije koje moraju obezbediti odgovarajuću otpornost na otkaze i vreme oporavka u slučaju katastrofalnih otkaza.
- **rešenje** - obezbediti redundantne servisne kontejnere i upravljanje tokovima poruka, uz odgovarajuće balansere opterećenja (load balancer) i rutiranje na mrežnom nivou. Pojedinačni servisi treba da su stateless i *re-entrant* kako bi se ovo efikasno obezbedilo.
- **primena** - svi servisi u brzorastućem servisnom okruženju koji treba da obezbede veliku dostupnost (*high-availability*).
- **rezultat** - lako se implementira za *stateless* servise dupliranjem servisa i odgovarajućim upravljanjem saobraćajem. Ovaj šablon se može koristiti da se obezbedi skalabilnost, ali i otpornost na otkaze.

SOA šabloni - složeni (kombinovani) šabloni

- **Wrapper** - enkapsulira API nekog zastarelog sistema u generički stateless servis.



- **problem** - stariji sistemi mogu imati vrlo limitirane načine upotrebe servisa, jer njihovi interfejsi na primer mogu biti ograničeni samo na razmenu fajlova ili zastarele API pozive.
- **rešenje** - iskoristiti *façade* šablon da se interoperabilni deo "umota" u servis koji komunicira sa legacy sistemom kao da ga direktno poziva *legacy* klijent. Novim klijentima se prikazuje normalizovani servisni interfejs.
- **primena** - integracija sa zastarem *mainframe* sistemima, ili klijent-server sistemima, kako bi se njihova funkcionalnost učinila dostupnom i novim klijentima.
- **rezultat** - iako ovaj pristup pomaže, mnogi klijent-server sistemi zahtevaju vrlo čvrsto povezivanje klijenta i servera, i "omotač" možda nije dovoljan kako bi se funkcionalnosti ovakvog sistema prikazale kao servisni interfejsi. U tom slučaju se možda mogu obezbediti samo *readonly* funkcionalnosti.

Prednosti SOA

- **Interoperabilnost** je ugrađena karakteristika IT sistema i aplikacija koje se razvijaju na principima SOA.
- Rezultujuće sisteme karakteriše **slaba povezanost i modularnost** komponenti koje komuniciraju preko **standardizovanih komunikacionih metoda** i kanala i dobro opisanih interfejsa.
- Povezivanje aplikacija je pojednostavljen i ne zahteva prepravljanje velikih delova aplikacija kada se mala izmena funkcionalnosti unese u sistem.
- SOA čine IT sisteme više agilnim i prilagodljivim na izmene poslovnih procesa. Pošto servisi (usluge) često predstavljaju poslovnu logiku, to stimuliše IT da razmišlja u smeru poslovnih funkcija koje treba realizovati.
- Koncepti SOA omogućavaju da IT rešenja budu i visoko-tolerantna na izmenjene zahteve, jer je rekonfigurisanje slabo povezanih komponenti (servisa) jednostavnije, relativno brzo i jeftino. Samim tim SOA omogućava da IT podrška brže reaguje na izmenjene poslovne zahteve, nove zahteve ili uvođenje novih procesa.

Kako razmišljati kada razvijate rešenje bazirano na SOA?

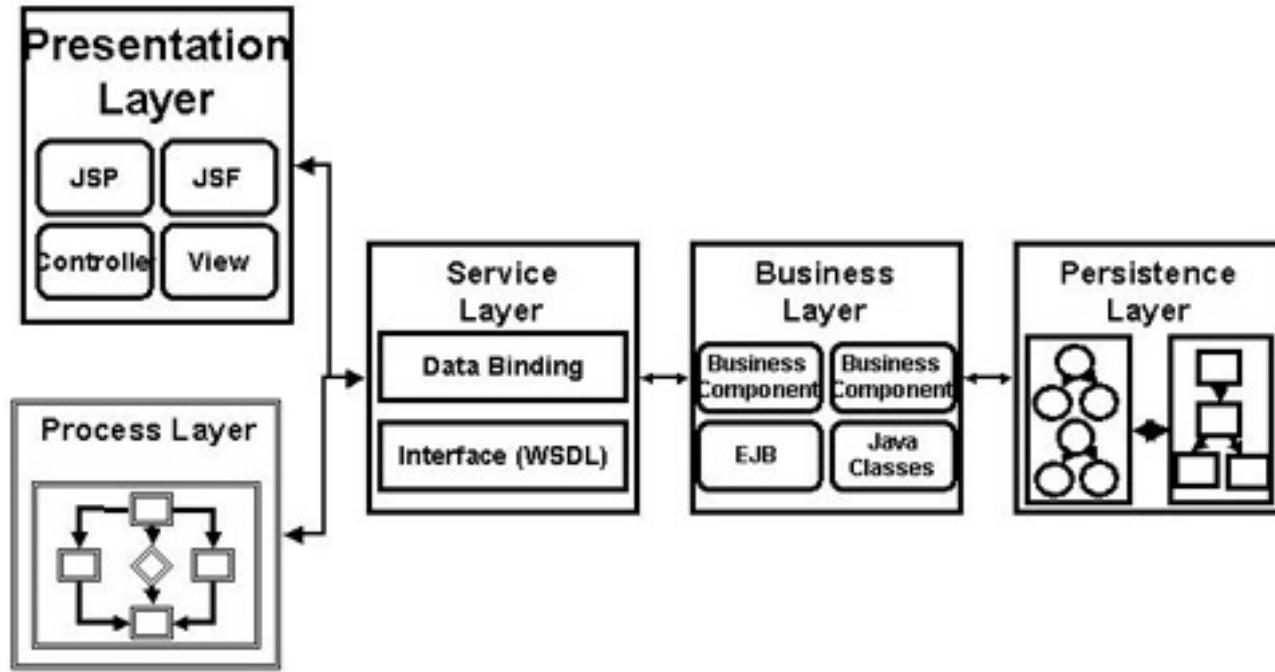
- Neophodno je razumeti poslovni model
- Fokus je na interoperabilnosti (komponenti)
- Fokus na poslovnoj agilnosti
- Definisati i sprovesti politiku aplikativne interoperabilnosti
 - Kako će komponente (web servisi) biti korišćeni?
 - Koji standarde koristiti i koju politiku interoperabilnosti sprovesti?
 - Kako će ovo biti implementirano u kontekstu SOA?
- Promene politike IT nabavki
 - Tekuće upravljanje SOA rešenjima podrazumeva da će od isporučioca softvera morati da se zahteva da se usklade sa vašom SOA strategijom i politikom interoperabilnosti.

SOA Dynamic Discovery (dinamičko pronalaženje servisa)

- Ovo je vrlo važan koncept u SOA. Tri osnovne komponente SOA su ponuđač servisa (usluge), korisnik servisa (usluge) i katalog usluga (registar).
- Katalog usluga je posrednička uloga, ponuđači svoje usluge prijavljuju u katalog, a korisnici mogu da pretraže katalog kako bi pronašli odgovarajuću uslugu.
- Većina kataloga organizuje servise po određenim pravilima kategorizacije. Korisnici koriste pretraživače koje katalog nudi da pronađu što im treba. Ugradnjom kataloga servisa u sam SOA postiže se sledeće:
 - Skalabilnost servisa – servisi se mogu inkrementalno dodavati
 - Potpuno razdvajanje (*decoupling*) korisnika od ponuđača usluge
 - Omogućava brz update servisa
 - Korisniku obezbeđuje servis za pregled servisa
 - Omogućava korisnicima, da čak i u momentu izvršavanja, izaberu ponuđača čiji servis trenutno najbolje ispunjava poslovne potrebe - mnogo bolje nego da se u samom kodu “zapeče” servis određenog ponuđača.

Tipična arhitektura SOA aplikacije

- Višeslojne aplikacije koje sadrže prezentacioni sloj, sloj poslovne logike, sloj za skladištenje podataka. Dva ključna sloja u SOA arhitekturi su servisni sloj i sloj poslovnih procesa.



Perspektive SOA arhitekture

- Neophodno je sagledati arhitekturu iz sledećih perspektiva:
 - **Arhitektura aplikacije.** Ovo je rešenje koje je orijentisano na zadovoljavanje poslovnih potreba korisnika i koje koristi (konzumira) servise (jednog ili više) ponuđača kako bi njihovom integracijom u svoj poslovni proces ostvario poslovni cilj.
 - **Arhitektura servisa.** Obezbeđuje vezu između implementacije servisa i klijentskih aplikacija – kreira logički pogled na skup servisa koji je moguće koristiti, a koji se pozivaju korišćenjem zajedničkog interfejsa i upravljačkih delova arhitekture.
 - **Arhitektura komponenti.** Opisuje različita okruženja koja podržavaju aplikacije koje implementiraju servise, poslovne objekte kojima se manipuliše i njihov način implementacije.

Izazovi i rešenja za SOA

- Pri implementaciji SOA suočavamo se sa sledećim izazovima:
 - **Identifikacija servisa.** Šta je servis? Koja je poslovna funkcionalnost koju bi on obezbeđivao? Koja je optimalna granularnost servisa?
 - **Lokacija servisa.** Gde je logično da se servis nalazi u poslovnom sistemu?
 - **Definicija domena servisa.** Kako se servisi grupišu u logične domene?
 - **“Pakovanje” servisa.** Kako se postojeća funkcionalnost *legacy* sistema može reinženjeringom zapakovati u ponovno iskoristive servise?
 - **Orkestracija servisa.** Kako je moguće orkestracijom od postojećih kreirati kompozitne servise?
 - **Rutiranje servisa.** Kako će zahtevi klijentskih aplikacija za određenim servisom biti rutirani do odgovarajućeg servisa i/ili domena?
 - **Upravljanje servisom.** Kako će poslovni sistem upravljati procesom administracije i održavanja servisa?
 - **Usvajanje standarda za razmenu poruka u okviru servisa.** Kako će poslovni sistem konzistentno usvojiti određeni standard razmene poruka?

Dodatni izazov sadašnjih SOA rešenja

- Trenutna rešenja često nisu u stanju da zadovolje zahteve izdržljivosti (*survivability requirements*). U slučaju da je radni servis preopterećen ili otkaže često ne uspeva automatsko pronalaženje i prevezivanje na neki alternativni servis.
 - Kritični sistemi (*Mission-critical systems*) će biti izloženi različitim napadima (elektronskim, fizičkim...)
 - Ključna osobina izdržljivosti jeste da je sistem sposoban da se dinamički rekonfiguriše u slučaju otkaza ili preopterećenja.
- Rešenje za ovo je servisno orijentisana distribuirana dinamička rekonfiguracija.

Problemi sa testiranjem SOA

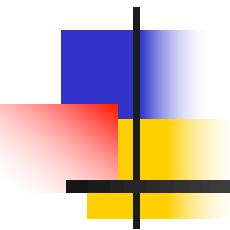
- Čak i male promene servisa mogu proizvesti značajan uticaj na današnje aplikacije za testiranje.
- Ali testiranje se mora obaviti.
- Postoje *frameworci* za testiranje SOA
- Indirektna testiranja i standardizacija su omogućavali da se izbegne bar deo problema.
- Oba ova pristupa nisu dovoljno dobra za SOA rešenja.

Problem indirektnog testiranja

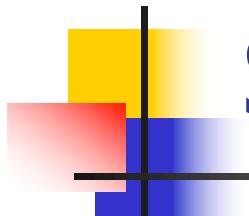
- Da li je moguće testirati servise indirektno? Zar samo "simuliranje" rada aplikacije ponuđača i klijenta nije dovoljno da se otkrije potencijalni problem? Problem je u vremenu kada je moguće obaviti testiranje i u podacima koji se za to koriste. Testiranje servisa moguće je tek kada servis već postoji – dakle dosta kasna faza razvoja, tako da veće korekcije zahtevaju dosta dodatnog rada (troškova). Često sami podaci nisu dovoljni da identifikuju gde problem nastaje:
 - U aplikaciji ponuđača servisa, načinu kodiranja podataka, transportu, sadržaju poruka ili klijentskoj aplikaciji
- Dodatna frustracija pri indirektnom testiranju proizilazi iz same prirode predložene arhitekture i njene najveće obećane prednosti: SOA ne prepostavlja ko će koristiti servise, SOA može biti korišćena i od strane aplikacija koje nisu u potpunosti razvijene i od strane korisnika koji nisu u okviru same organizacije. Da li standardi mogu da pomognu u ovom slučaju? Možda ne u potpunosti.

Problem indirektnog testiranja (nastavak)

- Iako principi koji čine integraciju aplikacija mnogo fleksibilnijom nego prethodni pristupi direktnog povezivanja, samo testiranje čine kompleksnijim. Tradicionalne monolitne aplikacije imaju korisnički interfejs koji se koristi i za verifikaciju funkcionalnosti, ali slojevi SOA ne nude takav interfejs - a svaki sloj se mora pojedinačno testirati tokom razvoja, a zatim i sprovesti integracione testove.
- Jedini način da testiramo poslovnu funkcionalnost razmene poruka (recimo u XML formatu) je putem test interfejsa koji dozvoljava da se poruke kreiraju, šalju, primaju i verifikuju (bilo u simulacionom okruženju ili na živom sistemu).
Bez toga ne može se govoriti o verifikovanosti integracionih tačaka.
- Dosada, svaki razvojni tim je pisao sopstveni kod za testiranje interfejsa na sloj za razmenu poruka. Ovo naravno povećava vreme i troškove. Testiranje integracije podrazumeva da postoje test interfejsi na svakom od slojeva, kako bi se poslovni proces mogao sprovesti od početka do kraja.

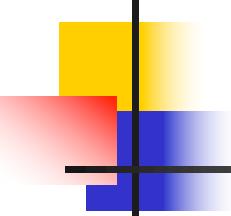


Web Services



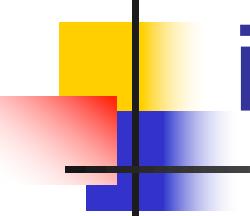
Šta su Web Servisi?

- Navikli smo da web sajtovima pristupamo direktno iz browsera
 - Browser pronalazi dokument na osnovu URL
 - Browser i server razmenjuju zahteve i odgovore, odgovor servera je najčešće HTML. Sve se transportuje preko HTTP
- Web Service uopštava ovaj model tako da računari (softverske komponente) mogu direktno da komuniciraju (putem interneta)



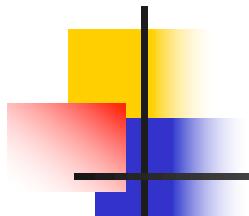
Istorijat

- Web servisi su razvijeni na osnovu prethodnih tehnologija koje su služile istoj nameni- RPC, ORPC (DCOM, CORBA i JAVA RMI).
- Web Servisi su somišljeni da reše neke osnovne probleme:
 - Interoperabilnost
 - Prolazak kroz Firewall
 - Kompleksnost rešenja



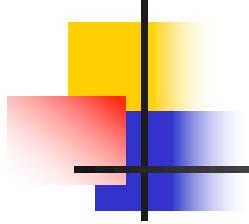
Prethodni problem - interoperabilnost

- Raniji distribuirani sistemi su imali probleme sa interoperabilnošću jer je svaki proizvođač implementirao sopstvene formate za razmenu poruka između distribuiranih objekata.
- DCOM je na primer bio vezan za Windows OS.
- RMI je bio striktno vezan sa Javom.



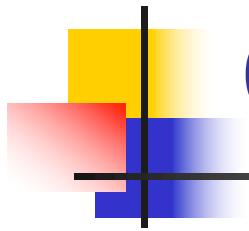
Prethodni problem - firewall-i

- Saradnja između korporacija je bila problem jer su sistemi poput CORBA i DCOM koristili nestandardne portove.
- Web Servisi koriste HTTP kao transportni protokol, a većina firewalla je konfigurisana da propušta konekcije na port 80 (HTTP).



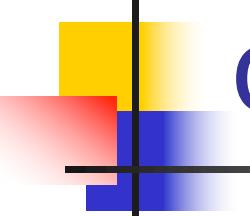
Prethodni problem - kompleksnost

- Web Servisi su u suštini developer-friendly.
- Većina prethodnih tehnologija kao što su RMI, DCOM, i CORBA zahtevali su mnogo više učenja i navikavanja na koncepte.
- Često je bilo potrebno naučiti dodatne jezike ili tehnologije.



CORBA pristup

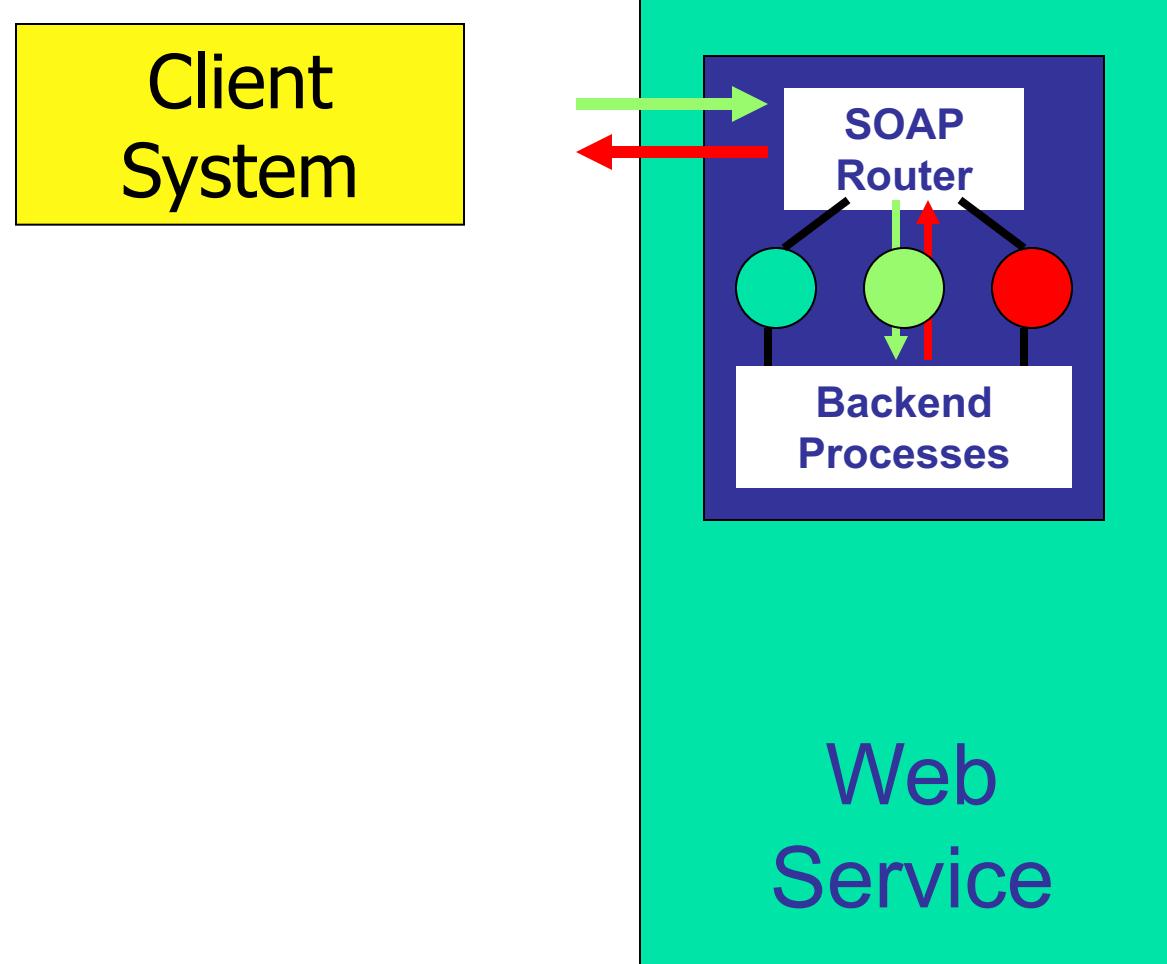
- CORBA je imao
 - Način da se eksportuju “client stubs”
 - “client stub” je mogao da sadrži i određenu logiku za odlučivanje koju je server isporučivao npr. “na koji datacenter se povezati”
 - Na ovaj način se serveru daje određeni vid daljinske kontrole
 - Factory servisi: kreiranje određenih vrsta objekata na zahtev
 - “pronalaženje” se moglo realizovati i kao aktivnost “kreiranja servisa”



CORBA je objektno orijentisana

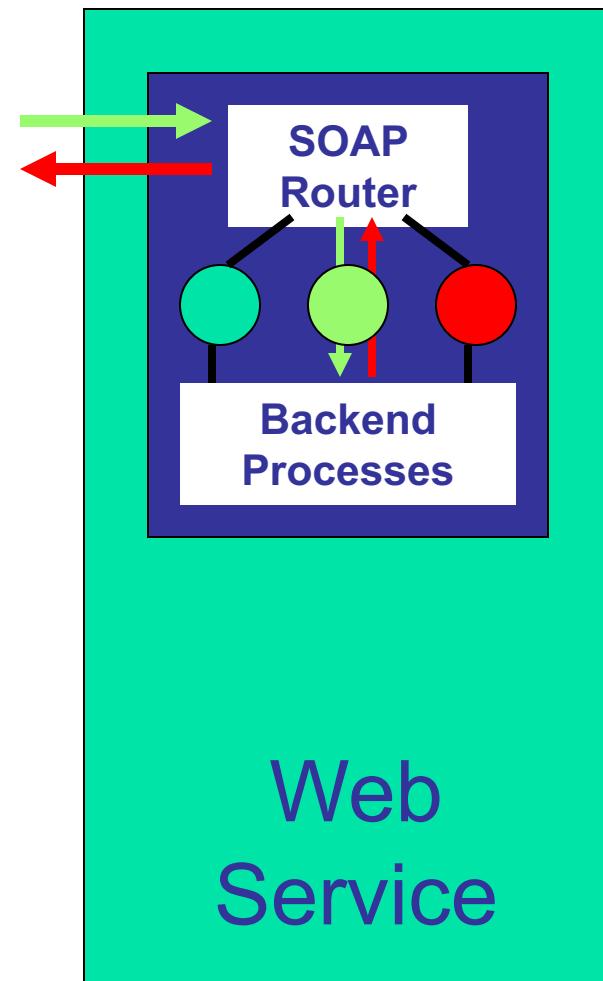
- CORBA je u potpunosti objekt-centrična
 - Objekti mogu biti
 - ... pasivni (podaci)
 - ... aktivni (programske funkcije)
 - ... perzistentni (podaci koji se snimaju)
 - ... volatile (stanje definisano samo tokom izvršavanja)
- U CORBA-i aplikacija koja upravlja objektom je nerazdvojiva od samog objekta
 - "client stub" je praktično deo aplikacije

Šta su onda Web Servisi?



Šta su Web Servisi?

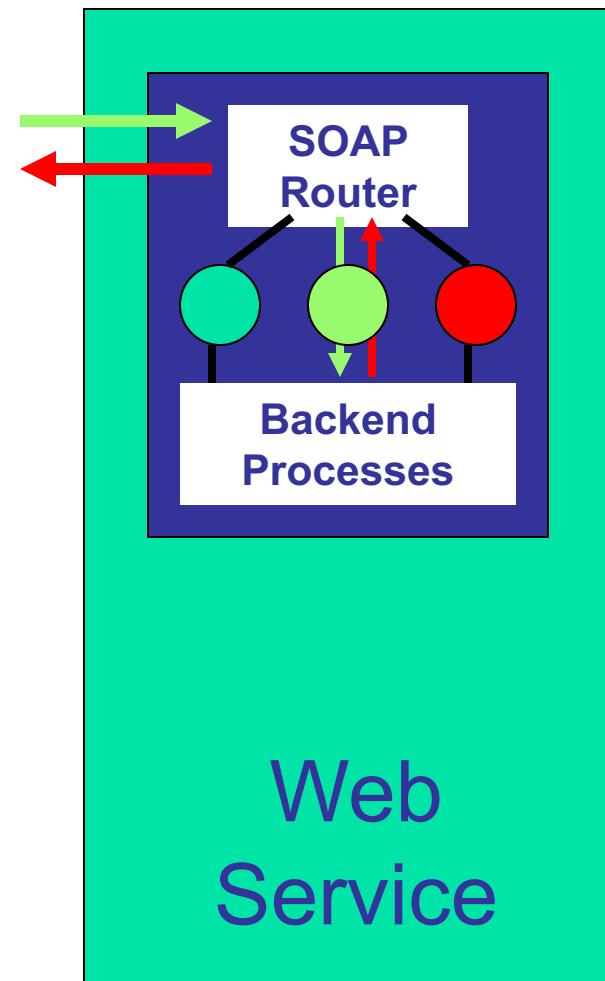
- "Web servisi su softverske komponente opisane pomoću WSDL, a kojima je moguće pristupiti preko standarnih mrežnih protokola (SOAP over HTTP)"



Šta su Web Servisi?

- "Web servisi su softverske komponente opisane pomoću WSDL, a kojima je moguće pristupiti preko standarnih mrežnih protokola (SOAP over HTTP)"

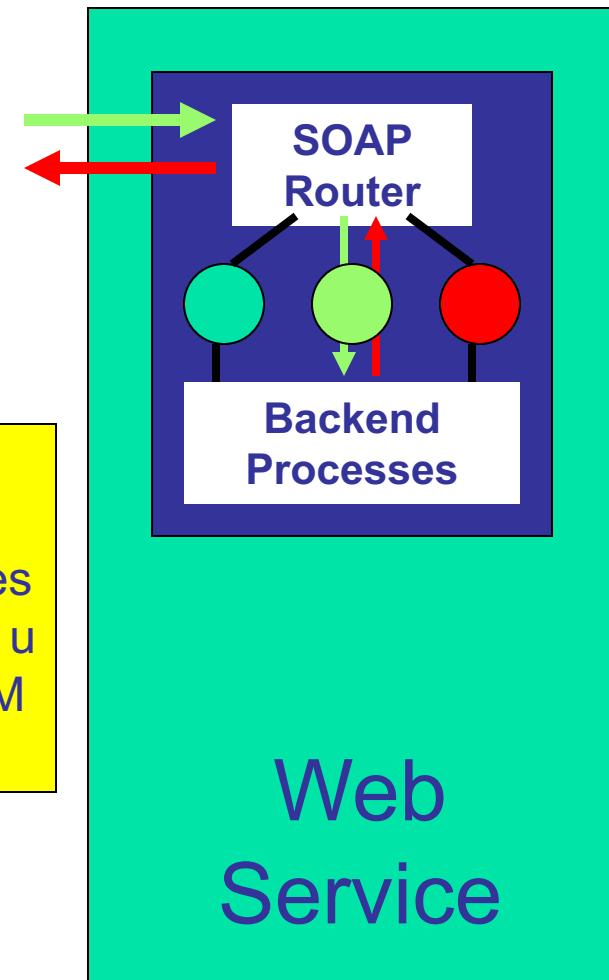
SOAP je primarni standard. Obezbeđuje pravila za kodiranje zahteva i njegovih argumenata – format pakovanja poruke.



Šta su Web Servisi?

- "Web servisi su softverske komponente opisane pomoću WSDL, a kojima je moguće pristupiti preko standarnih mrežnih protokola (SOAP over HTTP)."

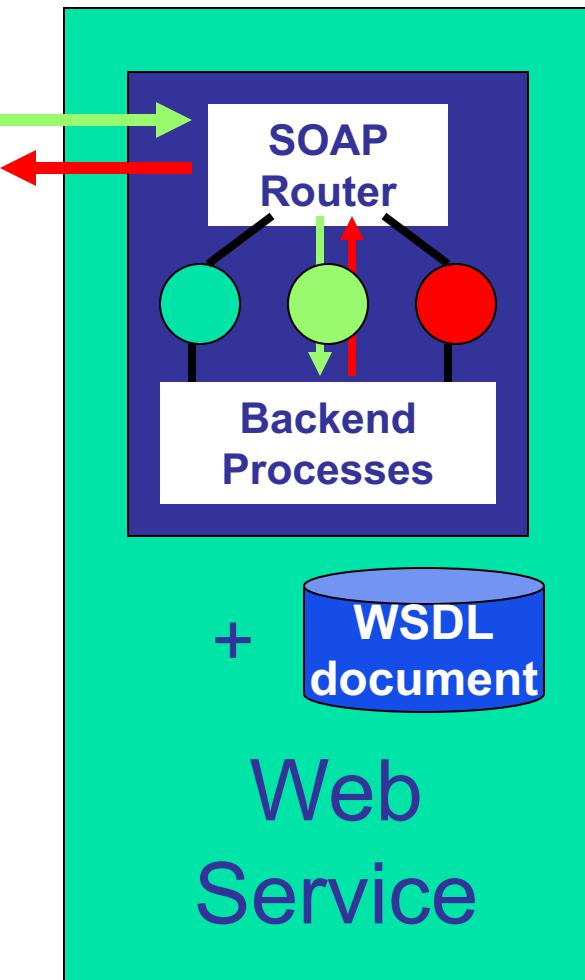
Arhitektura WS ne pravi pretpostavku da mora da se radi pristup preko HTTP-a na TCP-u. Na primer .NET koristi Web Services "interno" čak i u okviru jedne mašine. Ali se u tom slučaju komunikacija odvija preko COM ili WCF

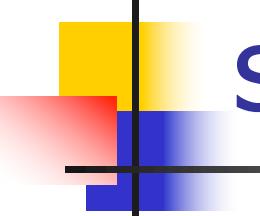


Šta su Web Servisi?

- "Web servisi su softverske komponente opisane pomoću WSDL, a kojima je moguće pristupiti preko standarnih vežnih protokola (SOAP over HTTP)."

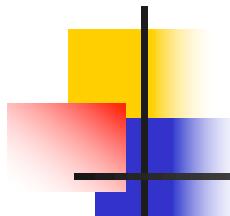
WSDL dokumente razvojni alati koriste kao uputstvo za formiranje objekata, generisanje koda...





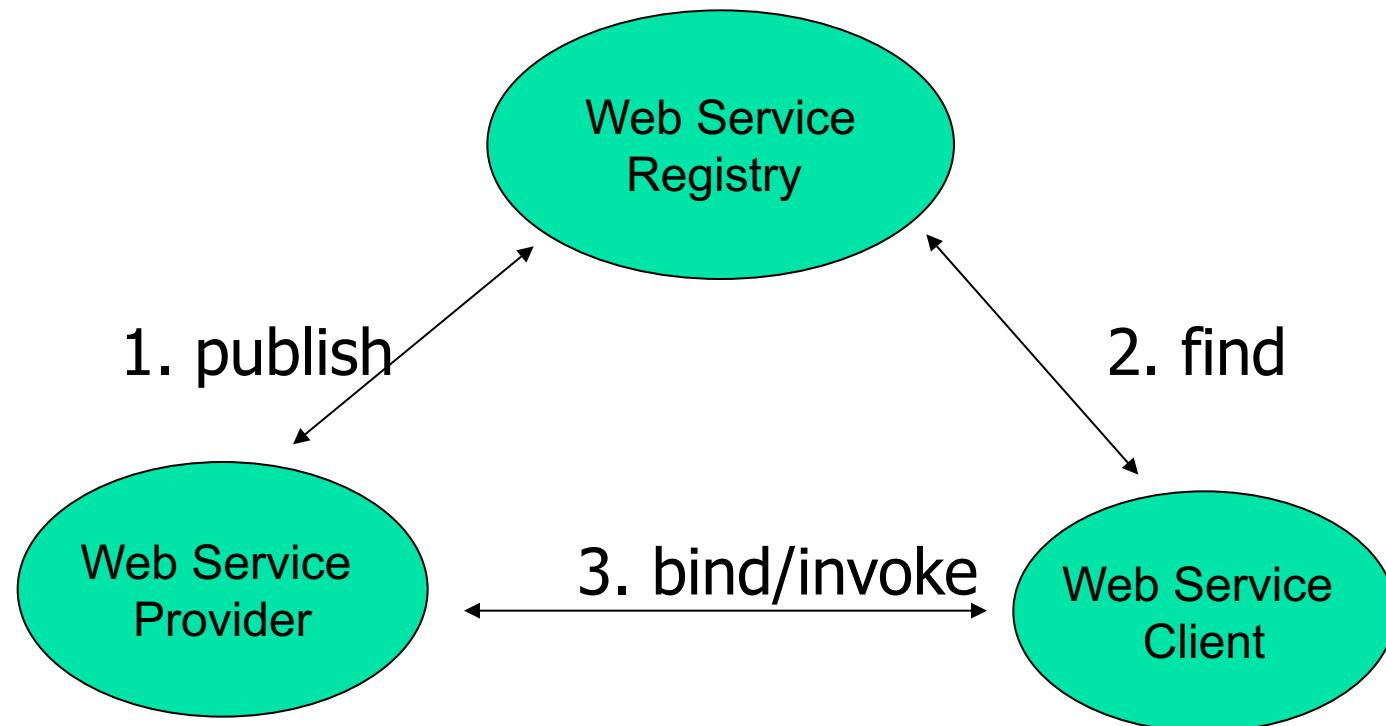
Komponente koje čine Web servise

- **XML** – Koristi se kao jedinstven način reprezentacije i razmene podataka.
- **SOAP** – (nekada Simple Object Access Protocol) – standardni način komunikacije.
- **UDDI** – Universal Description, Discovery and Integration specification - mehanizam za registraciju i lociranje servisa u katalogu.
- **WSDL** – Web Services Description Language – standardni meta jezik za opis ponuđenih servisa.

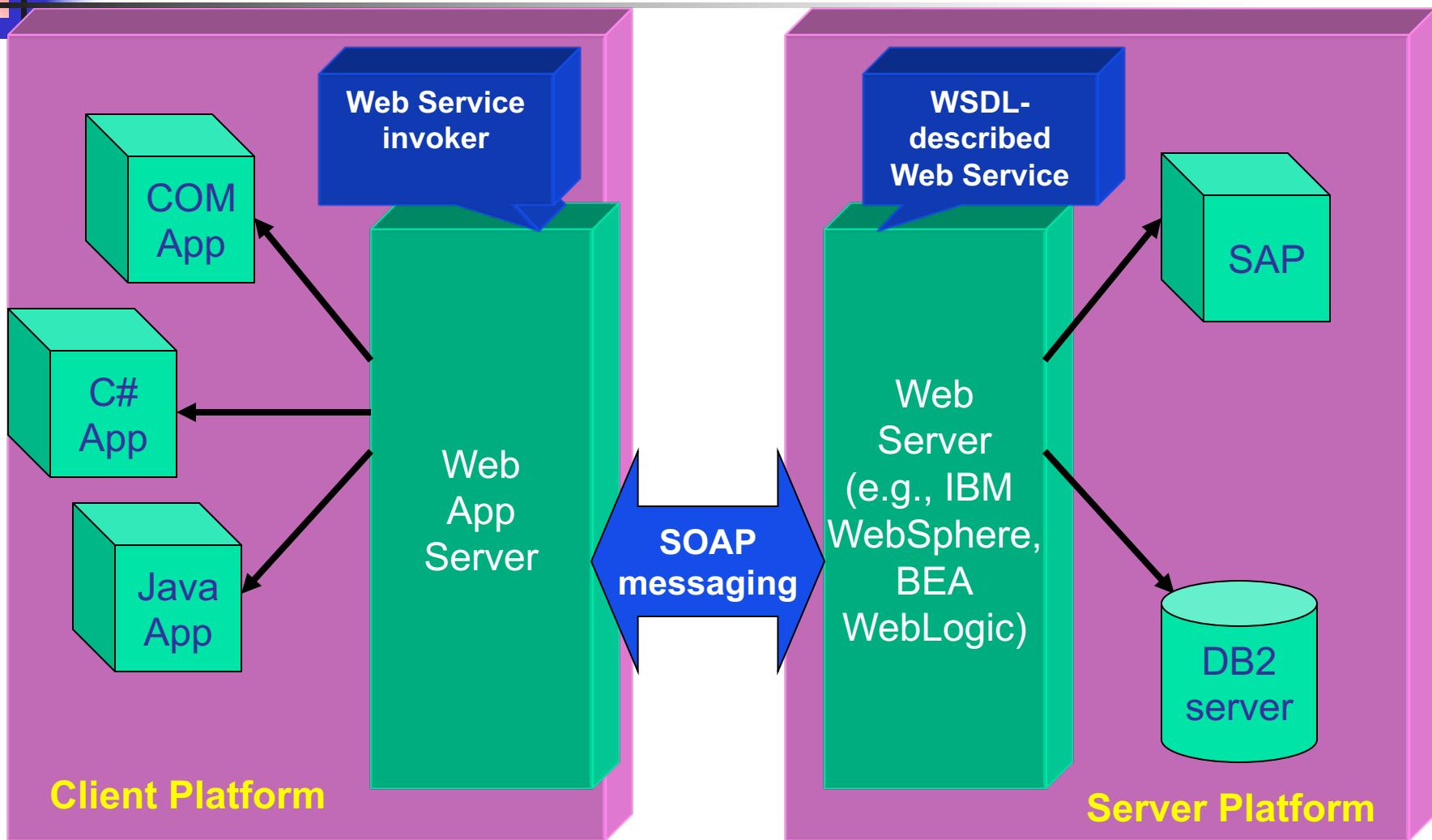


Model web servisa

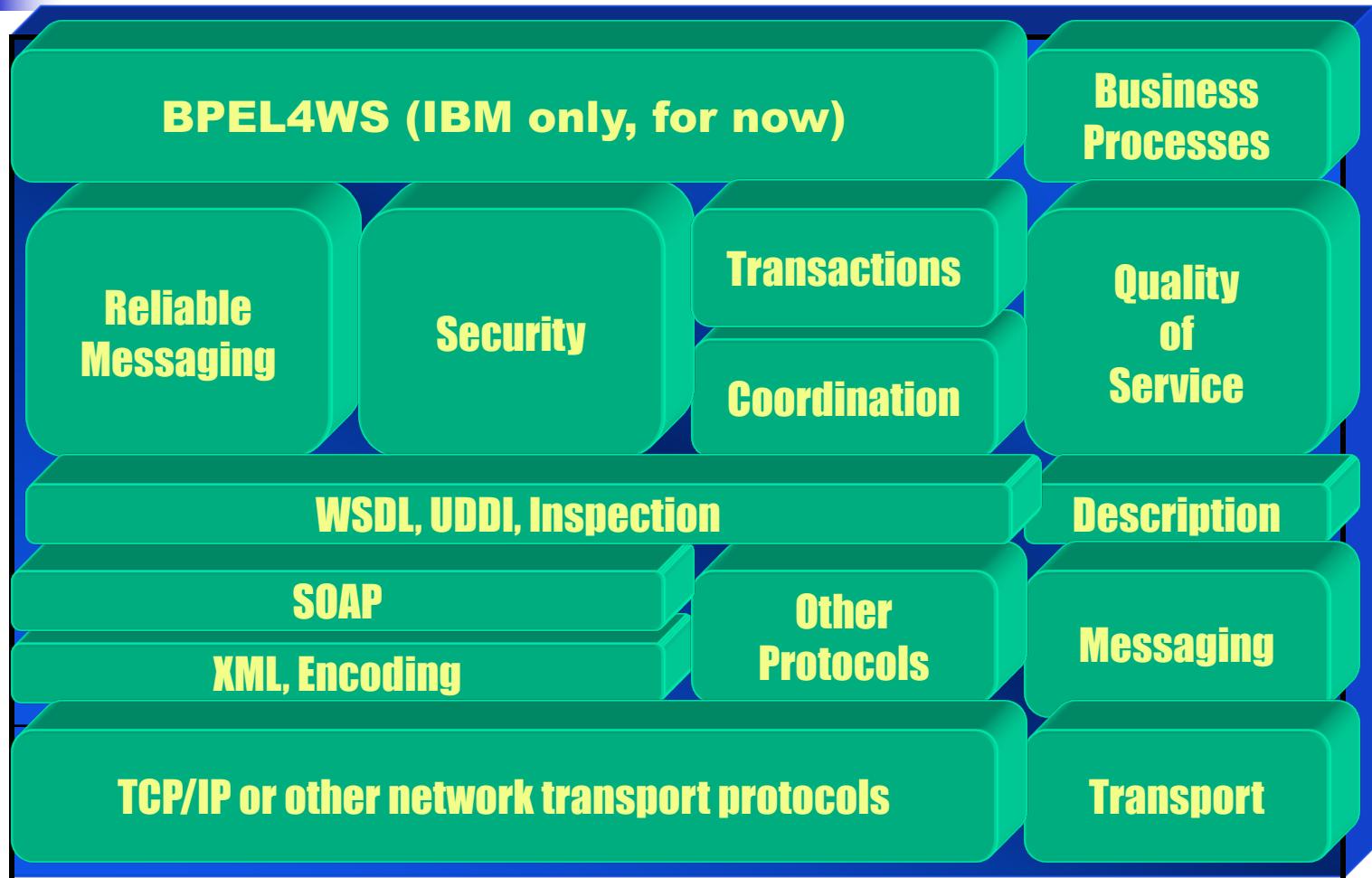
publish, find, i bind paradigma.

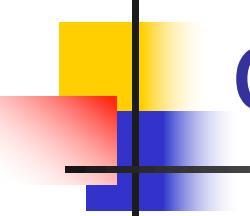


Web Servisi su često front-end vaše aplikacije



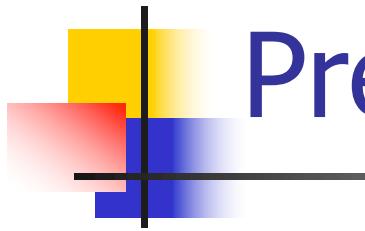
“Stack” Web servisa





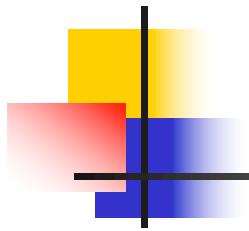
Web Servisi su dokument-centrični

- Sva komunikacija se obavlja slanjem dokumenata na server i primanjem dokumenta-odgovora sa servera
- Većina garancija na sadržaj i podatke je vezana za te dokumente a ne za sam servis
 - WS_RELIABILITY se ne odnosi na pouzdanost samog servisa, već definiše pravila kako se zahtevi za pouzdanost iskazuju i kako se povezuju sa dokumentom.
 - Nasuprot tome - CORBA fault-tolerance standard je specificirao kako obezbediti da CORBA servis bude visoko dostupan.



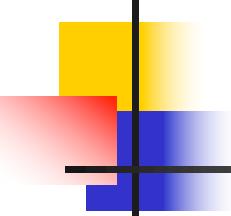
Prednosti Web servisa?

- Obezbeđuju interoperabilnost između heterogenih sistema koji se izvršavaju na heterogenim platformama.
 - "agnostik po pitanju proizvođača, platforme i programskog jezika"
- Web servisi koriste otvorene standarde i protokole. Kad god je moguće protokoli i formati podataka su tekst-bazirani
 - Relativno lako razumljivi.
- Time što se prenos podataka vrši "na leđima" HTTP web servisi mogu raditi kroz većinu uobičajenih firewall-a, a da pri tome nije neophodno njihovo posebno podešavanje



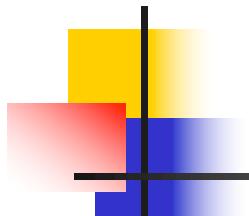
Kako rade web servisi

- Klijent prvo *pronalazi* servis.
- Tipično klijent potom izvršava povezivanje na servis (binding).
 - Tako što se uspostavi TCP veza ka adresi pronađenoj u opisu servisa.
 - Ovo povezivanje na servis nije uvek neophodno.



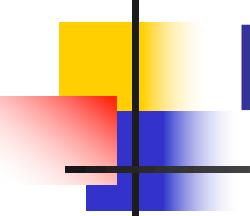
Kako rade web servisi

- Klijent potom kreira SOAP zahtev:
(Marshaling)
 - Popunjava informacije o tome koji servis je potrebno izvršiti, kao i podatke (argumente) koji su servisu neophodni za izvršavanja. Zapakovan zahtev šalje serveru.
 - XML je standardni način “pakovanja” podatka (ali uvodi jako veliki “overhead”)
- SOAP ruteri usmeravaju zahtev na odgovarajući server (ukoliko ih ima više od jednog)



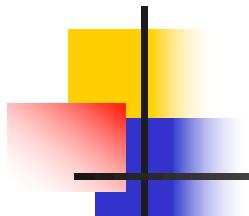
Kako rade web servisi

- Server raspakuje zahtev (*Unmarshaling*) obrađuje ga i proizvodi rezultat.
- Rezultat se šalje nazad.



Problemi sa *Marshalling-om*

- Podaci koji se razmenjuju moraju biti predstavljeni u platformski nezavisnom formatu.
 - “Endian”ness se razlikuje na različitim platformama.
 - Pitanje preciznosti (poravnanja) podataka (16/32/64 bits)
 - Različite reprezentacije *floating pointa*.
 - Pokazivači
 - Podrška za legacy sisteme



Pronalaženje servisa

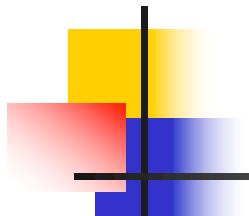
- Problem pronalaženja "pravog" servisa
 - Jedan način pomoću URL
 - Kod Web servisa se koristi i URN: Uniform Resource Name
- Uopšteniji pristup je da se koristi posrednik: servis za pronalaženje servisa (katalog)

Primer repozitorijuma (kataloga) servisa

Name	Type	Publisher	Toolkit	Language	OS
Web Services Performance and Load Tester	Application	LisaWu		N/A	Cross-Platform
Temperature Service Client	Application	vinuk	Glue	Java	Cross-Platform
Weather Buddy	Application	rdmgh724890	MS .NET	C#	Windows
DreamFactory Client	Application	billappleton	DreamFactory	Javascript	Cross-Platform
Temperature Perl Client	Example Source	gfinke13		Perl	Cross-Platform
Apache SOAP sample source	Example Source	xmethods.net	Apache SOAP	Java	Cross-Platform
ASS 4	Example Source	TVG	SOAPLite	N/A	Cross-Platform
PocketSOAP demo	Example Source	simonfell	PocketSOAP	C++	Windows
easysoap temperature	Example Source	a00	EasySoap++	C++	Windows
Weather Service Client with MS- Visual Basic	Example Source	oglimmer	MS SOAP	Visual Basic	Windows
TemperatureClient	Example Source	jgalyan	MS .NET	C#	Windows

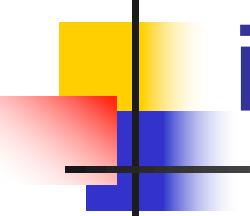
Repozitorijum (katalog) servisa

- Praktično baza koja izlistava spisak servera koji nude servise
- Opisan je korišćenjem UDDI jezika (XML notacija)
 - Samim tim moguće ga je pretraživati na iste načine kao i bilo koji XML
- Proširiv standard
 - Definiše određene zahtevane informacije (dostupne interfejse, tipove argumenata...)
 - Ali sami servisi mogu obezbediti dodatne informacije.



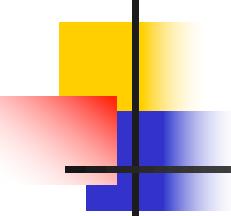
Uloga kataloga i opisa

- UDDI se koristi da opiše informacije koje predstavljaju jedan zapis u katalogu
- WSDL dokumentuje interfejse i tipove podataka koje servis koristi



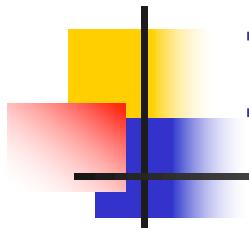
Pronalaženje i konvencije imenovanja

- The topic raises some tough questions
 - Dosta okruženja, poput veliki data-centara, često imaju neku standardnu strukturu (sadržaja i servisa). Da li se pronalaženje može automatizovati?
 - Kako debugovati aplikaciju ako se ponekad poveže sa pogrešnim servisom?
 - Delegiranje i migracija mogu biti problematični
 - Da li sistem može i treba da automatski pokrene servis na zahtev?



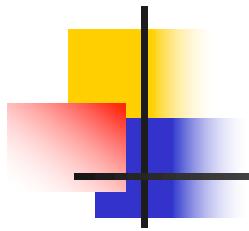
Zašto je otkrivanje servisa problematično

- Client ima svoje viđenje stvari
 - "Želim na mapi prikazane dužine svih linija prevoza i to odmah"
- Service ima svoje
 - Amazon.com npr. želi da svi zahtevi iz mesta Ithaca budu usmereni na njegov NJ-3 datacenter, i ukoliko je moguće na istu instancu unutar klastera
- DNS ima svoje
 - Dosta sistema se "poigrava" sa mapiranjem imena na IP adrese
- Internet ima svoje (kako obaviti rutiranje)



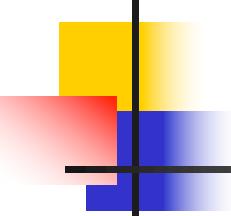
I šta je stvarno problem?

- Web Servis ne opisuje neki standardizovani način kako obaviti ove korake pretpostavlja da ih je na neki način moguće obaviti
- UDDI i WSDL su samo deo celokupne slike!



Network address translation...

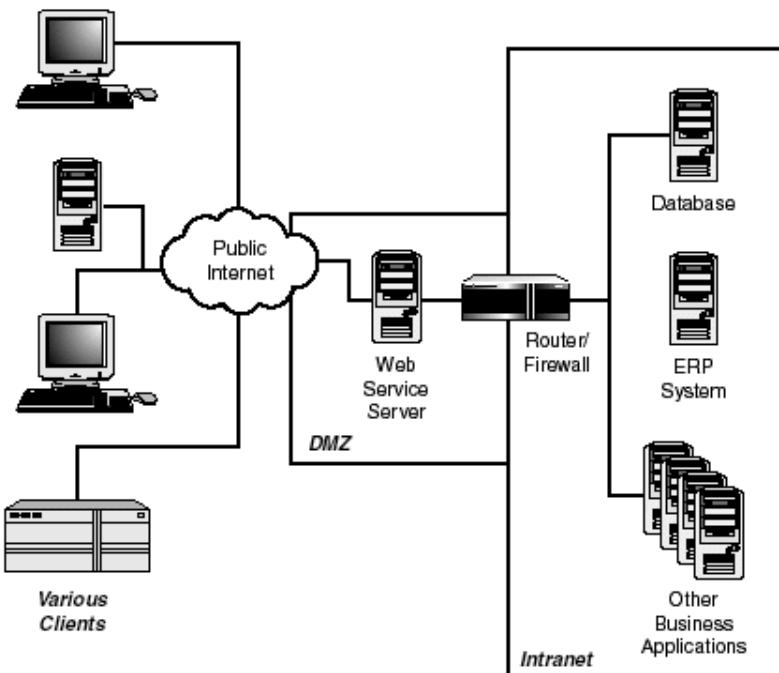
- Ovo je dodatni problem: interne adrese servera na mreži često i nisu dostupne spolja!
 - Dodaje se ponekad i kao mera zaštite.
 - Ali ako je RPC ili WS server iza NAT onda je to problem!
 - Neophodno je konfigurisati NAT da propušta određeni saobraćaj.
 - U opštem slučaju saobraćaj ka WS pošto se koristi HTTP se propušta.



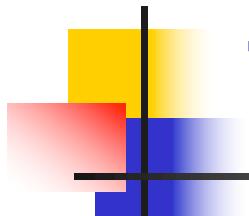
Firewalls

- Dozvoljavaju ili ne dozvoljavaju prolazak saobraćaja zavisno od izvora, destinacije, protokola....
 - Često se konfigurišu da samo dozvoljavaju ODLAZNE konekcije
- Mogu biti stateful: pamti aktivne tokove i odbacuje neočekivane pakete (NAT)
 - Opet moraju se konfigurisati da bi određeni saobraćaj prošao kroz njih (generalni problem RPC)
 - (WS)HTTP se ne suočava sa ovim problemom.

Demilitarized Zone (DMZ)

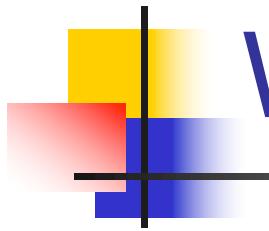


- DMZ: često se formira da se u nju smeste javno dostupni servisi webpages, ftp, dns.
- Dobro mesto i za Web Service!
- DMZ se smešta van privatne mreže.
- Ako je napadnut DMZ, šteta je limitirana na tu zonu.



Da li Web servisi "pomažu" pri imenovanju i otkrivanju servisa?

- Web Servisi nam govore kako klijent može:
 - ... pronaći server i...
 - ... povezati se na njega...
 - ... poslati zahtevi koji ima smisla...
 - ... i kako razumeti odgovor
- Dakle pomažu



Web servisi neće ...

- Omogućiti da data center (aplikacija) ima uticaj na odluke klijenta
- Rešiti probleme u skalabilnim klaster okruženjima
 - Kako uraditi balansiranje opterećenja?
 - Šta se dešava ako server koji izvršava servise padne?
 - Kako upravljati dinamičkim konceptima? Definisanje najboljeg servera za opsluživanje određenog klijenta može biti funkcija opterećenja, afiniteta, nedavnih taskova...
- Rešavanje ovakvih stvari nije u domenu WS

Web servisi i SOAP protokol

Pojam web servisa

- puno raznih definicija, ali...
- web servisi predstavljaju programe koji su dostupni putem javno objavljenih interfejsa i putem standardnih komunikacionih protokola.
- ono što se danas najčešće podrazumeva pod web servisima su programi
 - dostupni putem SOAP protokola,
 - sa interfejsom za pristup opisanim pomoću WSDL jezika i
 - (potencijalno) registrovani u UDDI servisu.

Pojam web servisa

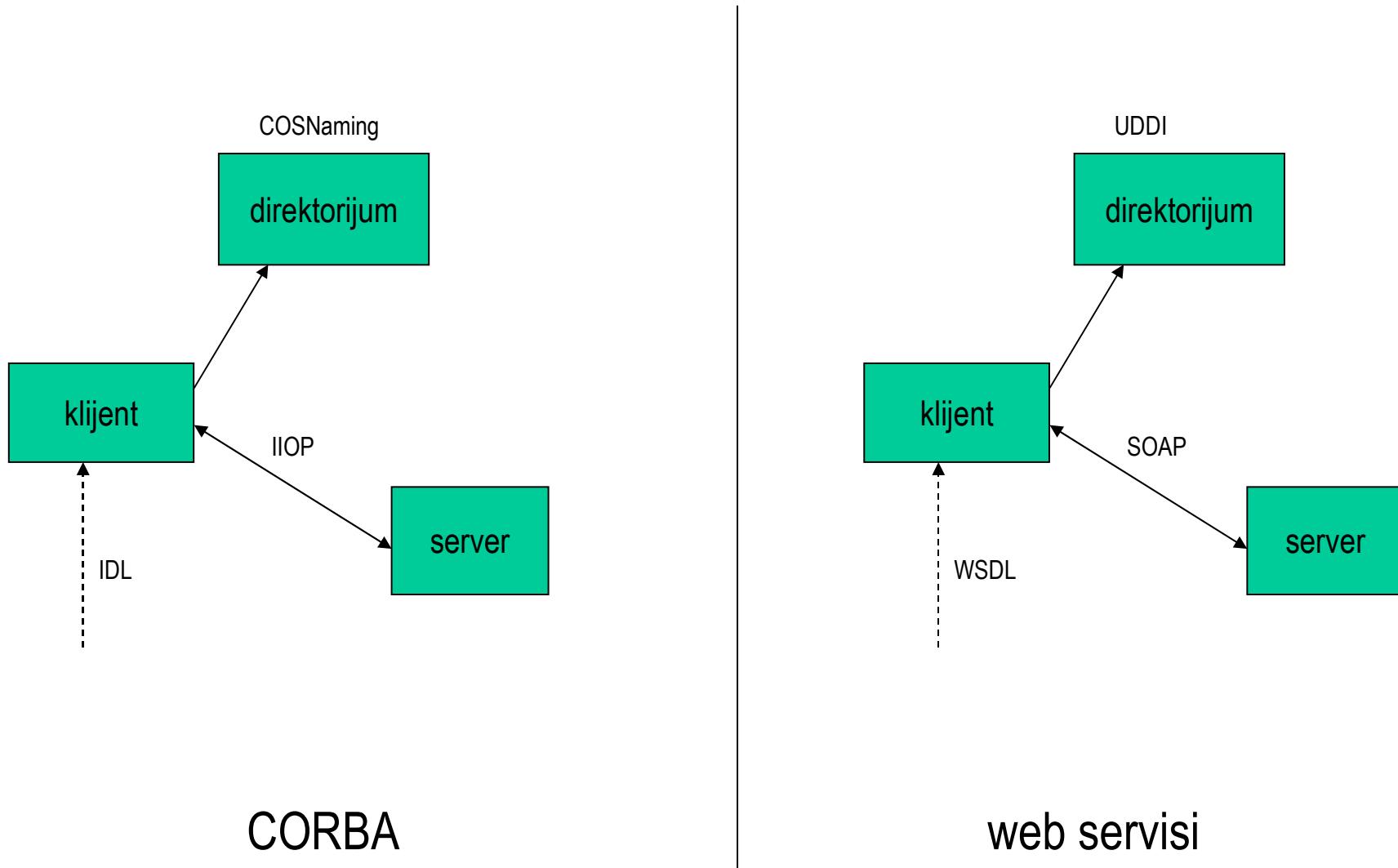
- skraćenice:
 - SOAP = Simple Object Access Protocol
<http://www.w3.org/>
 - WSDL = Web Services Description Language
<http://www.w3.org/>
 - UDDI = Universal Description, Discovery, and Integration
<http://www.uddi.org>

Web servisi i tehnologije distribuiranih sistema

- DCOM
 - tehnologija distribuiranih objekata specifična za Windows platformu
- Java RMI
 - tehnologija distribuiranih objekata specifična za Java platformu
- CORBA
 - tehnologija distribuiranih objekata nezavisna od platforme i programskog jezika
- EJB
 - tehnologija distribuiranih objekata koja se oslanja na RMI i CORBA
- .NET
 - tehnologija distribuiranih objekata specifična za Windows platformu

Web servisi i tehnologije distribuiranih sistema

- slični koncepti



Web servisi – zašto?

- čemu još jedna tehnologija?
 - svi elementi arhitekture, uključujući i komunikacioni protokol, su zasnovani na XML-u
 - jednostavnija implementacija nego kod binarnih formata
 - veća interoperabilnost među različitim implementacijama
 - upotreba standardnih Internet transportnih mehanizama
 - firewall-friendly
 - globalna dostupnost web servisa

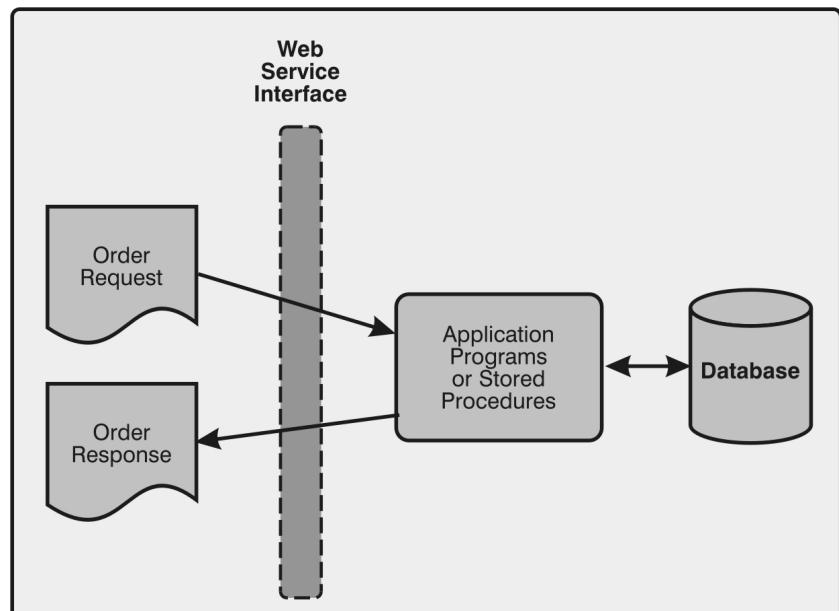
Web servisi – zašto?

- fokus je na interoperabilnosti
- konverzija podataka u/iz XML formata
 - nisu pogodni za sisteme gde su performanse komunikacije od posebnog značaja
- homogeni sistemi u celosti implementirani na jednoj razvojnoj platformi
 - nepotrebno komplikovanje implementacije sistema

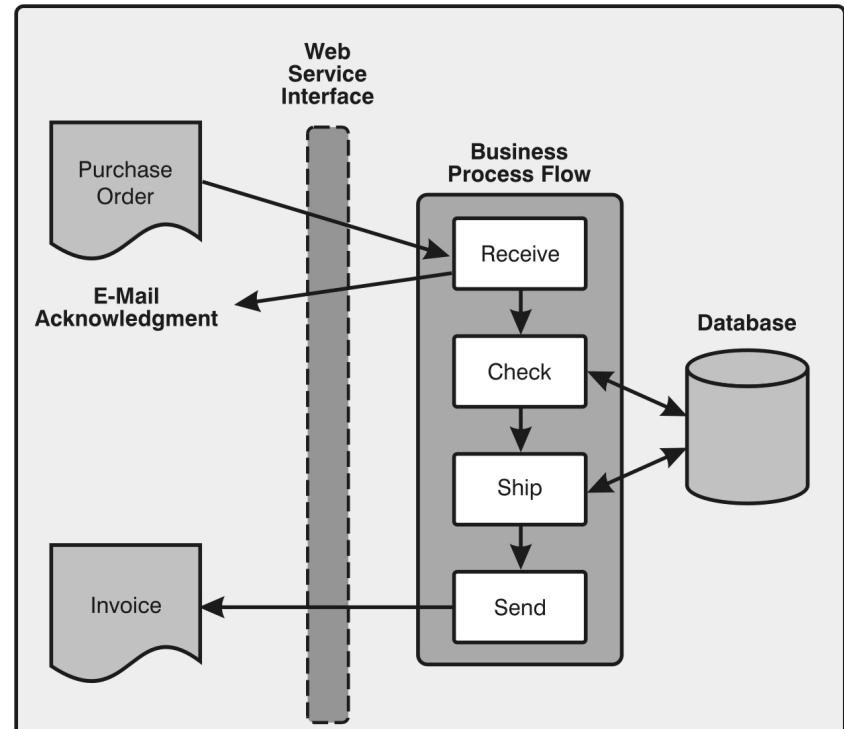
Dve paradigme distribuiranih sistema

- RPC = Remote Procedure Calls
 - podražavaju sintaksu i semantiku pozivanja funkcija/metoda
 - jednostavno za učenje
 - efikasno za kodiranje
 - tipično za sinhronu komunikaciju
- slanje poruka (message passing, document-style)
 - komunikacija između sistema pomoću slanja (strukturiranih) poruka
 - veza između sistema je data formatima poruka
 - bolje razdvajanje delova sistema (loosely coupled)
 - veća međusobna nezavisnost pojedinih delova
 - tipično za asinhronu komunikaciju

Dve paradigme distribuiranih sistema



RPC-style



document-style

Dve paradigme distribuiranih sistema

	RPC	MP
DCOM	x	
RMI	x	
CORBA	x	
EJB	x	x ¹
.NET	x	?
web servisi	x	x

¹Podrška za komunikaciju putem poruka kroz Message-Driven Beans i Java Message Service (JMS)

API-level i wire-level specifikacije

- API-level specifikacije
 - predstavljaju definicije klasa, interfejsa, metoda, itd. koje su načinjene za date potrebe
 - primeri:
 - JDBC – pristup relacionim bazama podataka
 - JDO – pristup objektnim bazama podataka
 - JNDI – pristup direktorijumskim sevisima
- wire-level specifikacije
 - predstavljaju definicije formata poruka koji se razmenjuju između učesnika u komunikaciji i postupka razmene poruka
 - primeri:
 - HTTP
 - SMTP

API-level i wire-level specifikacije

- API-level specifikacije omogućavaju pojednostavljenu zamenu programskih modula koji su apstrahovani datim API-jem
 - zamena baze podataka => zamena JDBC drijvera, neznatna promena programa
 - zamena direktorijumskog servera => zamena JNDI provajdera, neznatna promena programa
 - uspešna komunikacija JDBC drijvera jednog proizvođača sa bazom podataka drugog nije garantovana
- wire-level specifikacije omogućavaju komunikaciju heterogenih delova sistema
 - zamena jednog web servera drugim => klijenti će i dalje moći da komuniciraju

API-level i wire-level specifikacije

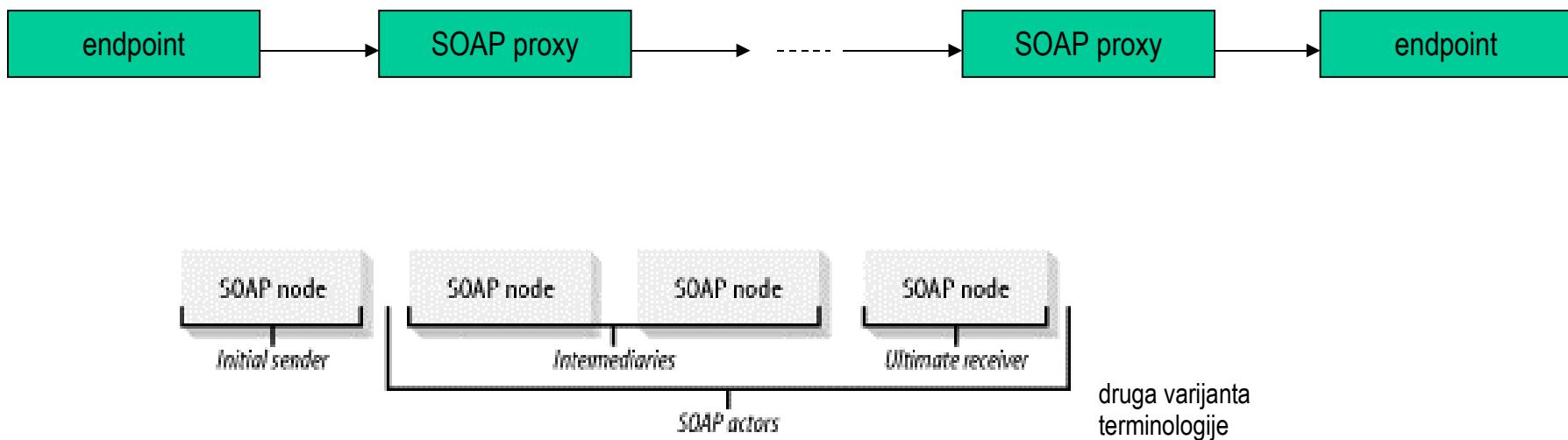
- API-level
 - konkretna implementacija predstavlja programsku biblioteku koja podržava API
 - preuzima se od nekog proizvođača
 - pišemo je sami
- wire-level
 - konkretna implementacija podrazumeva biblioteku koja implementira dati protokol
 - način pristupa biblioteci nije unapred propisan
 - programski kod zavisi od upotrebljene biblioteke

API-level i wire-level specifikacije

- API-level specifikacija za pristup wire-level protokolima
 - najbolje rešenje u smislu prenosivosti i interoperabilnosti programa
 - komplikovano za upotrebu sa stanovišta programera
(više isprepletanih standarda)
 - primeri:
 - JAXM – Java API for XML Messaging
 - JAXB – Java API for XML Binding
 - JAX-RPC – Java API for XML-based RPC
 - JAXR – Java API for XML Registries

SOAP – Simple Object Access Protocol

- protokol za komunikaciju sa web servisima
- definiše format poruka koje razmenjuju učesnici
- oslanja se na neki transportni mehanizam za prenos SOAP poruka
 - najčešće HTTP, ali nije obavezno, može i npr. SMTP
- verzije
 - SOAP verzija 1.2 je standard koga propisuje W3C
- u komunikaciji između krajnjih učesnika (endpoints) može biti posrednika (SOAP proxies)



SOAP i HTTP

- upotreba HTTP protokola za prenos SOAP poruka – “HTTP binding”
- za slanje zahteva koristi se HTTP POST komanda

SOAP i HTTP

- primer SOAP+HTTP zahteva i odgovora

```
POST /item HTTP/1.1
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: 250
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body>
    <m:GetPrice xmlns:m="http://www.nebitno.com/prices">
      <m:Item>Apples</m:Item>
    </m:GetPrice>
  </soap:Body>
</soap:Envelope>
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/soap+xml; charset=utf-8
```

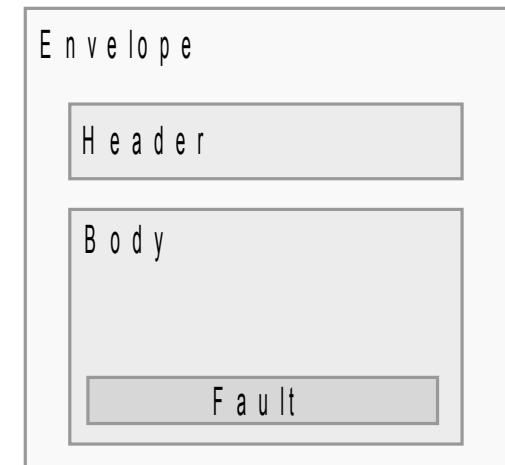
```
Content-Length: nnn
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body>
    <m:GetPriceResponse xmlns:m="http://www.nebitno.com/prices">
      <m:Price>1.90</m:Price>
    </m:GetPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Struktura SOAP poruke

- XML dokument sa
 - korenskim elementom Envelope
 - opcionim podelementom Header
 - obaveznim podelementom Body
 - sa opcionim podelementom Fault

```
<?xml version="1.0"?>
<soap:Envelope>
  <soap:Header>
    ...
    ...
  </soap:Header>
  <soap:Body>
    ...
    ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```



soap:Envelope

- obuhvata celu SOAP poruku
- definisan je u prostoru imena:
<http://www.w3.org/2001/12/soap-envelope>
- atribut **encodingStyle** definiše namespace sa tipovima podataka koji se koriste u dokumentu
 - odnosi se na element u kome je definisan i sve njegove podelemente
 - formalno nema podrazumevanu vrednost
 - u praksi se koristi
<http://www.w3.org/2001/12/soap-encoding>
 - zapravo definiše način serijalizacije podataka iz aplikacije u XML

soap:Envelope

- izgled elementa Envelope

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    ...
    ...
    ...
    ...
</soap:Envelope>
```

soap:Header

- sadrži podatke koji opisuju kontekst u kome se šalje poruka ili uputstva za posrednike u komunikaciji između krajnjih učesnika
- ovi podaci ne predstavljaju samu poruku, već pomoćne podatke koji utiču na način obrade poruke
- na primer:
 - podaci za autentifikaciju
 - podaci za praćenje sesije
 - podaci za upravljanje transakcijama
- Header pripada istom namespace-u kao i Envelope
- svi Header podelementi moraju biti kvalifikovani u odgovarajući namespace

soap:Header

- primer: identifikacija korisnika

```
<soap:Header>
  <usr:user xmlns:usr="http://www.nebitno.com/userauth">
    <usr:username>mbranko</usr:username>
    <usr:password>*****</usr:password>
  </usr:user>
</soap:Header>
```

soap:Header@actor

- atribut **soap:actor** označava onaj čvor u komunikaciji (proxy ili endpoint) za koga je namenjen dati podatak u zaglavju

```
<soap:Header>
  <usr:user xmlns:usr="http://www.nebitno.com/userauth"
            soap:actor="http://www.nebitno.com/AppServer">
    <usr:username>mbranko</usr:username>
    <usr:password>*****</usr:password>
  </usr:user>
</soap:Header>
```

soap:Header

- svaki SOAP procesor može da dodaje elemente u zaglavlje na putu poruke od pošiljaoca do konačnog primaoca
- svaki SOAP procesor je dužan da ukloni one elemente iz zaglavlja koji su namenjeni njemu
 - ono što je namenjeno njemu može ponovo dodati u zaglavlje prilikom daljeg slanja

soap:Header@mustUnderstand

- Koncept opcionalih i obaveznih elemenata u zaglavljiju
 - ne u smislu pojavljivanja u SOAP dokumentu, već
 - u smislu da je primalac dužan da razume i na pravilan način upotrebi dati podatak u zaglavljiju.
- Efekat: primalac poruke ne može da obradi poruku jer ne ume da interpretira obavezni podatak u zaglavljiju
- ako je element zaglavlja obavezan, atribut **soap:mustUnderstand** ima vrednost 1

```
<soap:Header>
  <usr:user xmlns:usr="http://www.nebitno.com/userauth"
            soap:actor="http://www.nebitno.com/AppServer"
            soap:mustUnderstand="1">
    <usr:username>mbranko</usr:username>
    <usr:password>*****</usr:username>
  </usr:user>
</soap:Header>
```

soap:Body

- sadrži konkretan SOAP zahtev ili odgovor
- pripada istom namespace-u kao i Envelope i Header
- koristi se kod svih vrsta web servisa (RPC-style, document-style)
- primer:

```
<soap:Body>
  <m:getCurrentTemperature xmlns:m="WeatherStation"
    <m:scale>Celsius</m:scale>
  </m:getCurrentTemperature>
</soap:Body>
```

soap:Fault

- opcioni podelement elementa Body
- sadrži podatke o nastalim greškama namenjene klijentu
- ima četiri podelementa:
 - faultcode
 - faultstring
 - faultactor
 - detail

soap:Fault/faultcode

- indikacija greške namenjena programskoj obradi
- obavezan
- moguće vrednosti:
 - **VersionMismatch**
element Envelope pripada pogrešnom namespace-u
 - **MustUnderstand**
neposredni Header podelement, sa atributom **mustUnderstand="1"**, nije interpretiran
 - **Client**
poruka sa zahtevom je nepravilno formirana ili sadrži neispravne podatke. Klijent ne bi trebalo da istu poruku ponovo šalje
 - **Server**
poruka nije mogla biti obrađena zbog problema u radu servera; sama poruka ima ispravan format i sadržaj. Klijent može pokušati sa istom porukom kasnije.
- Predefinisani kodovi greške su proširivi, na primer:
 - **Server.DatabaseFailure**
 - **Server.DatabaseFailure.MaxUsersConnected**

soap:Fault/faultstring

- tekstualni opis greške namenjen čoveku (human-readable)
- obavezan

soap:Fault/faultactor

- indikacija koji čvor u komunikaciji je uzrok greške (u paru sa **actor** atributom)
- ako se ne navede, podrazumeva se da je greška nastala na krajnjem čvoru komunikacije (endpoint)

soap:Fault/detail

- opisuje greške koje su posledica neispravnog sadržaja Body elementa u zahtevu
- ako sadržaj (ili nedostatak sadržaja) u Body elementu sprečavaju obradu poruke, element **detail** sadrži opis greške
- ako greška nije nastala usled sadržaja Body elementa u zahtevu, element **detail** se ne sme pojaviti

SOAP greške: primer 1

- nije bilo greške

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <m:getCurrentTemperature xmlns:m="http://www.nebitno.com/WeatherStation">
            <m:scale>Celsius</m:scale>
        </m:getCurrentTemperature>
    </soap:Body>
</soap:Envelope>
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <m:getCurrentTemperatureResponse
            xmlns:m="http://www.nebitno.com/WeatherStation">
            <m:temperature>1.90</m:temperature>
        </m:getCurrentTemperatureResponse>
    </soap:Body>
</soap:Envelope>
```

SOAP greške: primer 2

- neispravni podaci u Body elementu

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <m:getCurrentTemperature xmlns:m="http://www.nebitno.com/WeatherStation">
            <m:scale>Celzijus</m:scale>
        </m:getCurrentTemperature>
    </soap:Body>
</soap:Envelope>
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <soap:Fault>
            <soap:faultcode>soap:Client</soap:faultcode>
            <soap:faultstring>Illegal temperature scale</soap:faultstring>
            <soap:detail>
                <m:weatherfaultdetails xmlns:m="http://www.nebitno.com/WeatherStation">
                    <m:message>No such temperature scale: Celzijus</m:message>
                    <m:errorCode>1234</m:errorCode>
                </m:weatherfaultdetails>
            </soap:detail>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

SOAP greške: primer 3

- obavezni element zaglavlja nije interpretiran

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
    <usr:user xmlns:usr="http://www.nebitno.com/userauth"
               soap:actor="http://www.nebitno.com/AppServer"
               soap:mustUnderstand="1">
        <usr:username>mbranko</usr:username>
        <usr:password>*****</usr:password>
    </usr:user>
</soap:Header>
</soap:Envelope>

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
    <soap:Fault>
        <soap:faultcode>soap:MustUnderstand</soap:faultcode>
        <soap:faultstring>Did not understand the element user</soap:faultstring>
        <soap:faultactor>http://www.nebitno.com/AppServer</soap:faultactor>
    </soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP greške i HTTP greške

- HTTP kodovi 2xx – poruka je primljena i uspešno obrađena
- HTTP kodovi 3xx – redirekcija: traženi servis je premešten na drugu adresu
- HTTP kodovi 4xx – greška u zahtevu
- HTTP kodovi 5xx – greška je na strani servera, u toku obrade poruke

SOAP reprezentacija podataka

- dve mogućnosti:
 - *SOAP encoding*: mapiranje podataka iz Java/C++/... na XML u skladu sa SOAP specifikacijom
 - *XML Schema*: podaci koji se prenose definisani su XML šemom

SOAP encoding

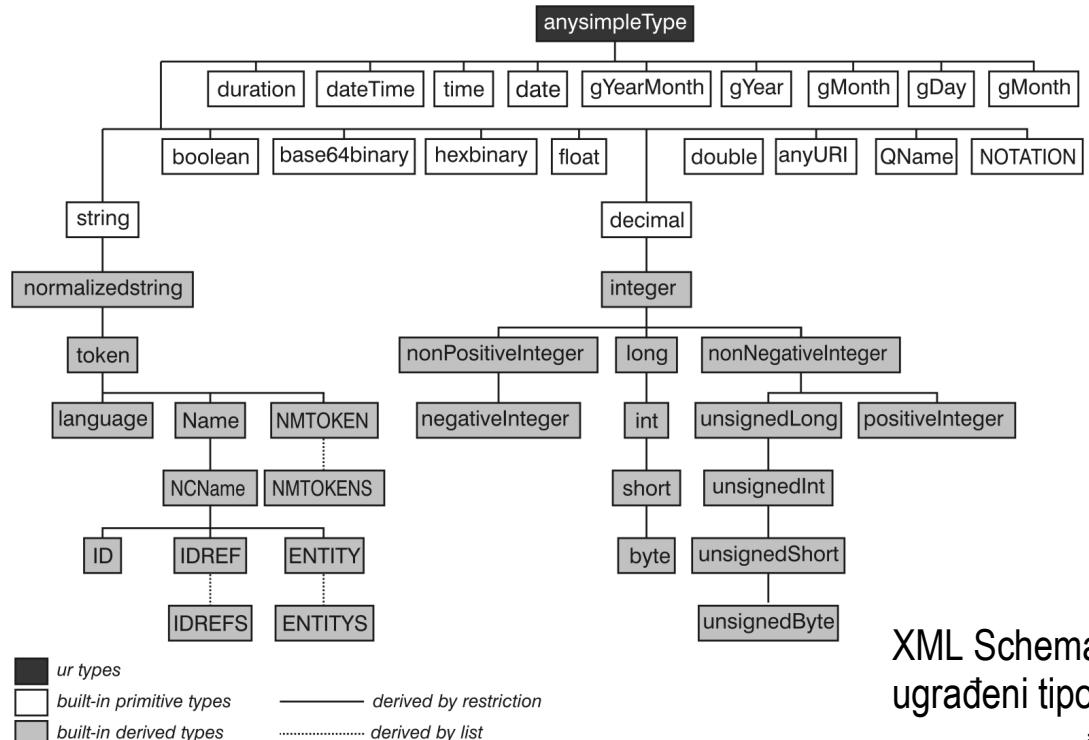
- podaci koji se nalaze u okviru Body elementa nastali su serijalizacijom podataka iz aplikacije u XML format u skladu sa nekom XML šemom
- SOAP ne definiše podrazumevanu šemu
- može se koristiti više različitih šema u jednoj SOAP poruci
- izbor aktivne šeme određuje se atributom **encodingStyle**
- jedina unapred definisana šema pripada namespace-u
<http://www.w3.org/2001/12/soap-encoding>
- ona sadrži sve predefinisane tipove podataka iz specifikacije
XML Schema Part 2: Datatypes
- npr: **xsd:string**, **xsd:int**, **xsd:boolean**, itd.
- svi podaci u telu poruke predstavljeni su kao sadržaj elementa

SOAP tipovi podataka

- jednostavnji (simple)
 - atomičke vrednosti
 - ne sadrži podelemente niti attribute
 - primeri

```
int a = 10;  
float x = 3.14159;  
String s = "abcd";
```

```
<a xsi:type="xsd:int">10</a>  
<x xsi:type="xsd:float">3.14159</x>  
<s xsi:type="xsd:string">abcd</s>
```



SOAP tipovi podataka

- agregirani (compound)
 - sadrži više atomičkih podataka organizovanih u neku strukturu
 - primeri

```
int array[3] = {1, 2, 3};  
<array xsi:type="soapenc:Array" soapenc:arrayType="xsd:int[3]">  
    <val>1</val>  
    <val>2</val>  
    <val>3</val>  
</array>
```

```
class Sample {  
    public int iVal = 10;  
    public String sVal = "Ten";  
}  
<sample>  
    <iVal xsi:type="xsd:int">10</iVal>  
    <sVal xsi:type="xsd:string">Ten</sVal>  
</sample>
```

SOAP tipovi podataka

- navođenje tipa podatka
 - atributom xsi:type
 - elementima niza je tip definisan za ceo niz
 - tipovi su
 - predefinisani tipovi
 - novi tipovi definisani pomoću XML Schema jezika

```
<complexType name="TAutomobile">
  <sequence>
    <element name="make" type="xsd:string"/>
    <element name="model" type="xsd:string"/>
    <element name="year" type="xsd:string"/>
  </sequence>
</complexType>
```

```
<car xsi:type="TAutomobile">
  <model>Corvette</model>
  <make>Chevrolet</make>
  <year>1999</year>
</car>
```

SOAP tipovi podataka

- često korišćeni prostori imena
 - XML Schema
`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
 - XML Schema instances
`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
 - SOAP Encoding
`xmlns:soapenc="http://www.w3.org/2001/12/soap-encoding"`

SOAP tipovi podataka

- prethodni primer, uz uvažavanje prostora imena

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xsd:targetNamespace="http://www.nebitno.com/cars">
  <xsd:complexType name="TAutomobile">
    <xsd:sequence>
      <xsd:element name="make" type="xsd:string"/>
      <xsd:element name="model" type="xsd:string"/>
      <xsd:element name="year" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

<car
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.nebitno.com/cars"
  xsi:type="TAutomobile">
  <model>Corvette</model>
  <make>Chevrolet</make>
  <year>1999</year>
</car>
```

SOAP tipovi podataka

- reference
 - element na koji ukazuje referenca mora sadržati atribut `id` tipa `ID`
 - elementi koji predstavljaju referencu su prazni i imaju atribut `href` čiji sadržaj je identifikator

```
<employee id="Bob">
  <firstName>Bob</firstName>
  <lastName>Englander</lastName>
</employee>
<employee>
  <firstName>Ben</firstName>
  <lastName>Jones</lastName>
  <manager href="#Bob"/>
</employee>
<employee>
  <firstName>Andrew</firstName>
  <lastName>Smith</lastName>
  <manager href="#Bob"/>
</employee>
<employee>
  <firstName>Lorraine</firstName>
  <lastName>White</lastName>
  <manager href="#Bob"/>
</employee>
```

korisno za serijalizaciju grafova objekata!

SOAP tipovi podataka

- stringovi
 - **xsd:string**
 - **soapenc:string** – dodata podrška za **id** i **href**

- nabrojivi tipovi

```
<simpleType name="TTrafficSignal" base="xsd:string">
    <enumeration value="Red"/>
    <enumeration value="Yellow"/>
    <enumeration value="Green"/>
</simpleType>

<trafficSignal xsi:type="TTrafficSignal">Green</trafficSignal>
```

- binarni podaci

- **xsd:hexBinary**
- **xsd:base64Binary**
- **soapenc:base64**

SOAP tipovi podataka

- null vrednosti

```
<name xsi:null="1"/>
```

SOAP tipovi podataka

- nizovi
 - soapenc:Array
 - tip niza je definisan pomoću soapenc:arrayType

```
int array[3] = {1, 2, 3};

<array xsi:type="soapenc:Array" soapenc:arrayType="xsd:int[3]">
    <val>1</val>
    <val>2</val>
    <val>3</val>
</array>
```
 - null vrednosti u nizovima

soapenc:root

- specijalni atribut koji označava

- početak liste
- koren stabla
- itd.

```
class Node {  
    int value;  
    Node next;  
}  
  
<list>  
    <node>  
        <value id="node-3" xsi:type="xsd:int">70</value>  
        <next xsi:null="1"/>  
    </node>  
    <node>  
        <value id="node-2" xsi:type="xsd:int">60</value>  
        <next href="#node-3"/>  
    </node>  
    <node>  
        <value soapenc:root="1" id="node-1" xsi:type="xsd:int">50</value>  
        <next href="#node-2"/>  
    </node>  
</list>
```

SOAP i "polimorfizam"

- recimo da je element **quantity** u nekoj šemi definisan kao tip **xsd:int**
- "polimorfizam" predstavlja dinamičku promenu tipa u telu poruke
- primer

```
<quantity>37</quantity>
<quantity xsi:type="xsd:float">37</quantity>
```

Web servisi RPC tipa

- pozivanje metoda udaljenog objekta
 - SOAP zahtev = poziv metode
 - SOAP odgovor = rezultat metode
- za poziv je potrebno
 - identifikator objekta kome se upućuje poruka
 - naziv metode
 - parametri metode
 - podaci za SOAP zaglavlje

Web servisi RPC tipa

- identifikacija objekta kome se upućuje poruka
 - naziv objekta se ugrađuje u POST komandu HTTP zahteva

```
POST /PriceService HTTP/1.1
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: 250
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <m:GetPrice xmlns:m="http://www.nebitno.com/PriceService">
            <m:Item>Apples</m:Item>
        </m:GetPrice>
    </soap:Body>
</soap:Envelope>
```

Web servisi RPC tipa

- naziv metode
 - određen je podelementom Body elementa
 - servis mora prepoznati dati element i pozvati odgovarajuću metodu stvarnog objekta

```
POST /PriceService HTTP/1.1
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: 250
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <m:GetPrice xmlns:m="http://www.nebitno.com/PriceService">
            <m:Item>Apples</m:Item>
        </m:GetPrice>
    </soap:Body>
</soap:Envelope>
```

Web servisi RPC tipa

- parametri metode
 - dobijeni serijalizacijom podataka iz aplikacije

```
POST /PriceService HTTP/1.1
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: 250
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body>
        <m:GetPrice xmlns:m="http://www.nebitno.com/PriceService">
            <m:Item>Apples</m:Item>
        </m:GetPrice>
    </soap:Body>
</soap:Envelope>
```

```
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"

    xsd:targetNamespace="http://www.nebitno.com/PriceService">
    ...
    <xsd:element name="Item" type="TItem"/>
    <xsd:simpleType name="TItem" base="xsd:string">
        <enumeration value="Apples"/>
        <enumeration value="Oranges"/>
        <enumeration value="Mushrooms"/>
    </xsd:simpleType>
</xsd:schema>
```

Web servisi RPC tipa

- rezultat metode
 - strukturiran prema datoj šemi

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body>
    <m:GetPriceResponse xmlns:m="http://www.nebitno.com/prices">
      <m:Price>1.90</m:Price>
    </m:GetPriceResponse>
  </soap:Body>
</soap:Envelope>
```

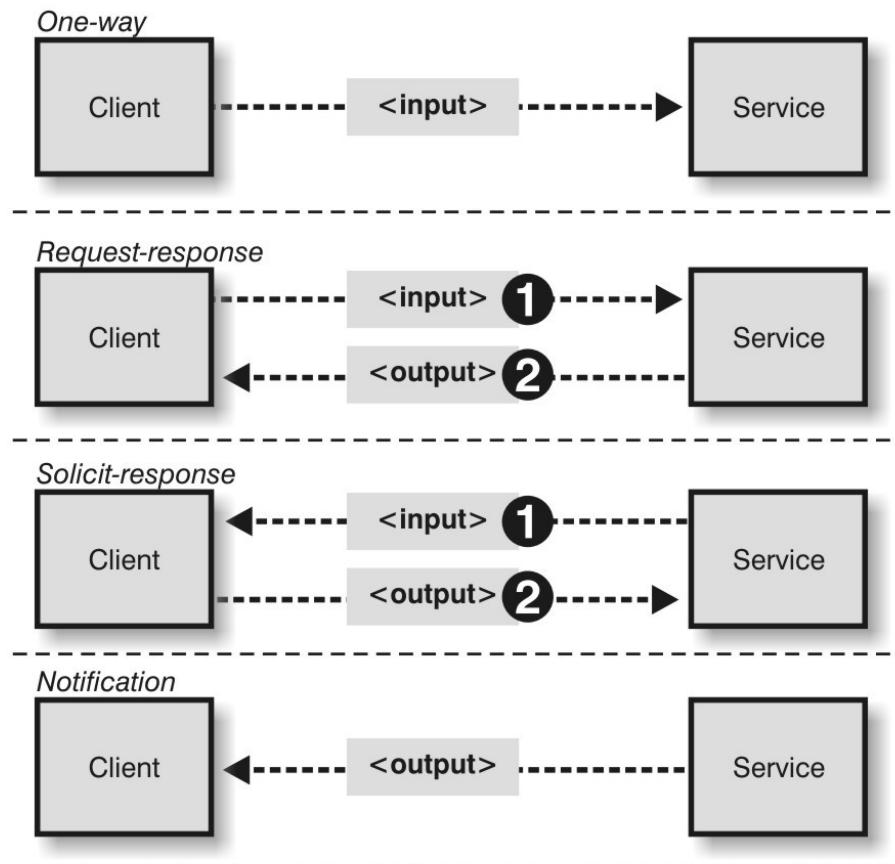
Document-style web servisi

- ne postoji koncept metode i parametara
- servisu se šalje XML dokument
- metoda servisa koja će obraditi dokument se ipak mora nekako navesti
 - naziv prvog Body podelementa mora naznačavati metodu koja će se pozvati za obradu datog dokumenta
 - namespace tog elementa identificuje servis koji će obraditi zahtev

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body>
    <recordTemperature xmlns="urn:WeatherDiary"/>
    <temperature>75.5</temperature>
    <zipcode>50328</zipcode>
  </soap:Body>
</soap:Envelope>
```

Document-style web servisi

- komunikacija nije obavezno po modelu zahtev/odgovor



WSDL

WSDL koncepti

- WSDL = Web Services Description Language
 - XML gramatika za opisivanje web servisa kao skupa krajnjih pristupnih tačaka (access endpoints) koji mogu da razmenjuju poruke na RPC- ili document-style način
 - access endpoint = URL na koji se šalje zahtev
 - WSDL ~ CORBA IDL
 - oba standarda su namenjena definisanju interfejsa i tipova podataka za servise dostupne sa udaljenih klijenata
 - WSDL obezbeđuje proširivost koju CORBA IDL nema
 - opisivanje krajnjih pristupnih tačaka (endpoints) i poruka bez obzira na korišćeni mrežni protokol ili format poruka za razmenu
 - tretman poruka kao apstraktnih opisa podataka koji se razmenjuju
 - tretman tipova portova kao apstraktnih kolekcija operacija web servisa

WSDL koncepti

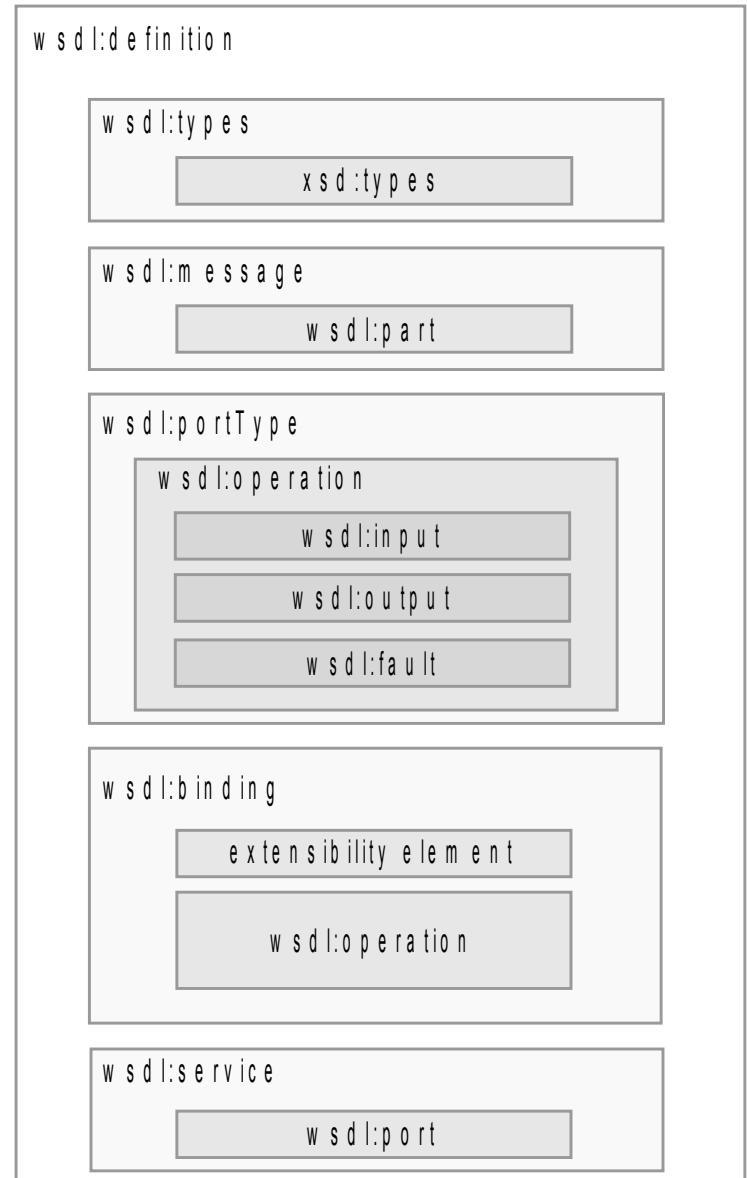
- WSDL fajl opisuje
 - šta servis radi
 - kako pozvati njegove operacije
 - gde ga pronaći
- na osnovu WSDL specifikacije servisa moguće je generisati
 - implementaciju servisa (delimičnu – tela metoda ćemo morati sami da napišemo!)
 - klijentske klase za pristup servisu
- na osnovu datih interfejsa u implementaciji servisa moguće je generisati
 - WSDL fajl koji opisuje dati servis

WSDL koncepti

- koga/jaje problem
 - da li prvo napisati WSDL specifikaciju servisa pa onda pisati implementaciju servisa
 - kada očekujemo da će međusobno komunicirati delovi sistema zasnovani na različitim tehnologijama (npr. Java i .NET)
 - da li prvo definisati interfejs servisa u jeziku implementacije pa onda generisati WSDL opis
 - kada obe strane u komunikaciji koriste istu implementacionu tehnologiju

Struktura WSDL dokumenta

```
<definitions>
  <import>*
  <types>
    <schema></schema>*
  </types>
  <message>*
    <part></part>*
  </message>
  <PortType>*
    <operation>*
      <input></input>
      <output></output>
      <fault></fault>*
    </operation>
  </PortType>
  <binding>*
    <operation>*
      <input></input>
      <output></output>
    </operation>
  </binding>
  <service>*
    <port></port>*
  </service>
</definitions>
```



wsdl:definitions

- korenski element WSDL dokumenta
- često se koristi za globalne namespace deklaracije

```
<definitions
    targetNamespace="urn:3950"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    xmlns:soapenc="http://www.w3.org/2001/12/soap-encoding"
    xmlns:tns="urn:3950">
    ...
</definitions>
```

wsdl:import

- dodaje sadržaj datog namespace-a u tekući WSDL dokument
- način za modularizaciju WSDL dokumenata
- dodati namespace definisan je u datom fajlu
 - preporučuje se navođenje absolutne URL putanje zbog prenosivosti
- dodati tipično sadrži zajedničke XML Schema definicije tipova

```
<import namespace="http://www.nebitno.com/CommonTypes"
          location="http://www.nebitno.com/schemas/CommonTypes.xsd"/>
```

wsdl:types

- kontejner za definicije tipova podataka koji se koriste dalje u dokumentu
- kao jezik za definisanje tipova podataka najčešće se koristi W3C XML Schema
 - WSDL ne ograničava izbor jezika za definisanje tipova podataka, moguće je koristiti i druge jezike osim W3C XML Schema
 - drugi jezici se vrlo retko koriste
 - pitanje je koliko su podržani u softverskim bibliotekama
- ugrađeni W3C XML Schema tipovi podataka ne moraju se eksplisitno importovati

wsdl:types

- primer

```
<types>
<xsd:schema>
  <xsd:simpleType name="KnownProximityUnit">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="character" />
      <xsd:enumeration value="word" />
      <xsd:enumeration value="sentence" />
      <xsd:enumeration value="paragraph" />
      <xsd:enumeration value="section" />
      <xsd:enumeration value="chapter" />
      <xsd:enumeration value="document" />
      <xsd:enumeration value="element" />
      <xsd:enumeration value="subelement" />
      <xsd:enumeration value="elementType" />
      <xsd:enumeration value="byte" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
</types>
```

wsdl:message

- za definisanje poruka koje se razmenjuju u komunikaciji sa web servisom
- poruka se sastoji iz delova (parts)
 - svaki deo pripada nekom tipu podataka

```
<types>
  <xsd:schema xsd:targetNamespace="http://www.nebitno.com/prices">
    <xsd:simpleType name="TItem">
      <xsd:restriction base="xsd:string">
        <enumeration value="Apples"/>
        <enumeration value="Oranges"/>
        <enumeration value="Mushrooms"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
</types>

<message name="GetPrice">
  <part name="Item" type="TItem"/>
</message>
<message name="GetPriceResponse">
  <part name="Price" type="xsd:float"/>
</message>
```

wsdl:message

- primer SOAP/HTTP komunikacije sa prethodno definisanim porukama

```
POST /item HTTP/1.1
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: 250
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body>
    <m:GetPrice xmlns:m="http://www.nebitno.com/prices">
      <m:Item>Apples</m:Item>
    </m:GetPrice>
  </soap:Body>
</soap:Envelope>
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: nnn
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body>
    <m:GetPriceResponse xmlns:m="http://www.nebitno.com/prices">
      <m:Price>1.90</m:Price>
    </m:GetPriceResponse>
  </soap:Body>
</soap:Envelope>
```

wsdl:portType

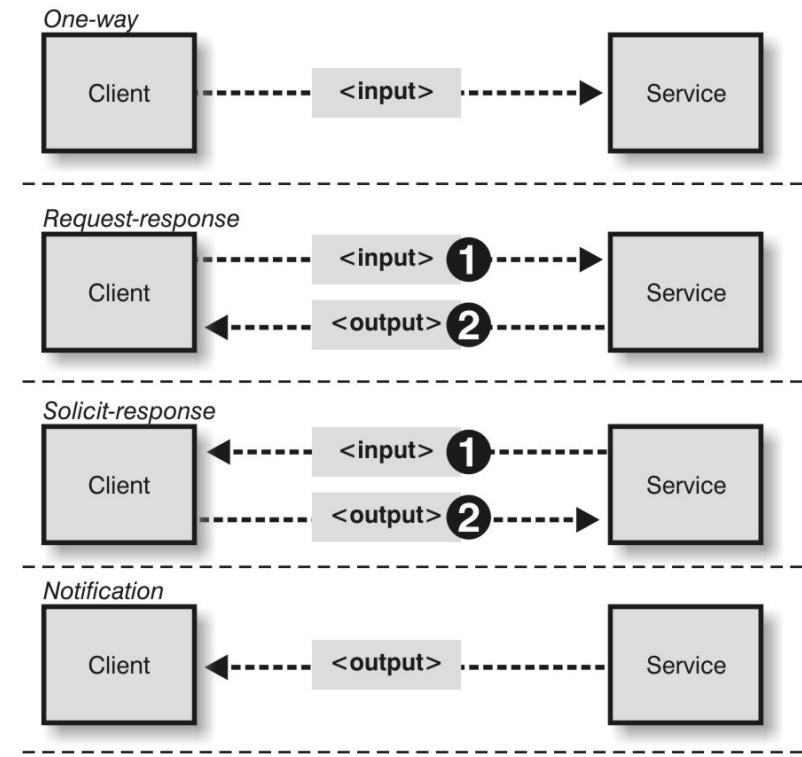
- definiše skup operacija koje se mogu izvršiti nad web servisom
- operacija, kao element ovog skupa, može sadržati
 - ulaznu poruku
 - izlaznu poruku

```
<portType name="PricesPort">
  <operation name="GetPrice">
    <input message="GetPrice"/>
    <output message="GetPriceResponse"/>
  </operation>
</portType>
```

wsdl:portType

- četiri tipa operacija

```
<!-- one-way operations -->  
<operation name="triggerResourceControl">  
  <input message="triggerResourceControl"/>  
</operation>  
  
<!-- request-response operations -->  
<operation name="init">  
  <input message="init"/>  
  <output message="initResponse"/>  
</operation>  
  
<!-- solicit-response operations -->  
<operation name="accessControl">  
  <output message="accessControlResponse"/>  
  <input message="accessControl"/>  
</operation>  
  
<!-- notification operations -->  
<operation name="Segment">  
  <output message="SegmentRequest"/>  
</operation>
```



wsdl:binding

- definiše konkretni protokol i format podataka za port type
- može se koristiti
 - standardan protokol (HTTP, SOAP, MIME, itd)
 - definisati novi
- port type predstavlja apstraktnu definiciju operacija
- pomoću binding elementa ove operacije se konkretizuju u skladu sa izabranim protokolom

wsdl:binding

- za svaku operaciju navedenu u portType sekciji mora se navesti odgovarajuća stavka u binding sekciji

```
<binding name="PricesBinding" port="PricesPort">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http">
    <operation name="GetPrice">
      <soap:operation soapAction="">
        <input>
          <soap:body use="encoded"
                     encodingStyle="http://www.w3.org/2001/12/soap-encoding"/>
        </input>
        <output>
          <soap:body use="encoded"
                     encodingStyle="http://www.w3.org/2001/12/soap-encoding"/>
        </output>
      </soap:operation>
    </operation>
  </soap:binding>
</binding>
```

The diagram illustrates the structure of a wsdl:binding element. It shows the XML code with annotations. An arrow points from the 'use' attribute of the first soap:body element (which is 'encoded') to a box labeled 'encoded | literal'. Another arrow points from the entire binding element to a box containing soap:binding, http:binding, and mime:binding.

```
<soap:binding>
<http:binding>
<mime:binding>
```

encoded | literal

wsdl:service

- u prethodnim elementima nigde nije navedeno na kojoj URL adresi se nalazi web servis
- element service nije obavezan
- koristi se kada je potrebno definisati konkretan endpoint za web servis
- servis je skup portova
 - svaki port predstavlja po jedan endpoint u komunikaciji
 - moguće je definisati servis koji se sastoji od portova koji su dostupni na različitim adresama

wsdl:service

- primer

```
<service name="NS Quantash Market">
  <documentation>Current prices at the NS Quantash Market</documentation>
  <port name="PricesPort" binding="PricesBinding">
    <soap:address location="http://www.nsmarket.co.yu/Quantash/Prices"/>
  </port>
</service>
```

Preporuke za definisanje web servisa

- operacije web servisa bi trebalo da predstavljaju "krupnije" poslovne transakcije
 - nije potrebno svaku metodu Java klase proglašiti za WSDL operaciju
- iako je WSDL definicija tesno vezana sa programskim kodom, u nju ne treba da uđe ništa što korisnik ne bi trebalo da zna
 - WSDL definicija predstavlja metapodatke o programskom kodu
- korišćenje W3C XML Schema tipova za postizanje maksimalne interoperabilnosti

Dizajn web servisa

- postoji nekoliko varijanti u dizajnu web servisa
 - RPC / encoded
 - RPC / literal
 - document / encoded
 - document / literal
 - document / literal wrapped

```
<binding name="PricesBinding" port="PricesPort">
    <soap:binding style="rpc|document" ...>
        <operation name="GetPrice">
            <soap:operation soapAction="">
                <input>
                    <soap:body use="encoded|literal" .../>
                </input>
                <output>
                    <soap:body use="encoded|literal" .../>
                </output>
            </soap:operation>
        </operation>
    </soap:binding>
</binding>
```

Primeri

- 08 wsdl.zip
 - RPC/encoded
 - RPC/literal
 - document/literal
 - document/literal, varijanta 2
 - document/literal wrapped

UDDI

UDDI koncepti

- UDDI = Universal Description, Discovery, and Integration
 - <http://www.uddi.org>
 - standardno okruženje za publikovanje i otkrivanje informacija o web servisima

UDDI koncepti

- softver jedne firme koristi web servise druge firme kako bi obavio poslovnu transakciju
- u poslovnom okruženju sa puno partnera potrebno je organizovati podatke o njima
 - ne samo uobičajene podatke o samim partnerima, već i podatke o njihovim web servisima
 - takvih web servisa potencijalno ima mnogo
- UDDI predstavlja jedno moguće rešenje za organizaciju kataloga poslovnih partnera i sa podacima o njihovim servisima

UDDI koncepti

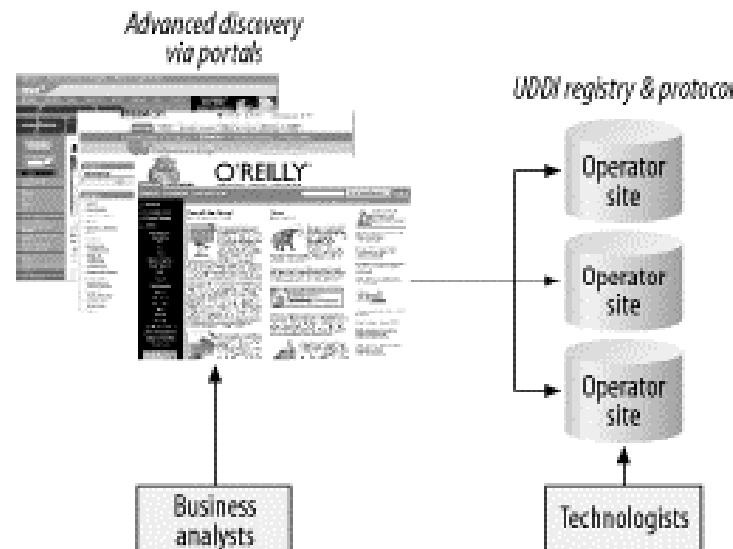
- standard za objavljivanje podataka
 - o sopstvenoj poslovnoj organizaciji ("business") zajedno sa
 - podacima o pristupu softverskim servisima koje organizacija nudi
- UDDI omogućuje formiranje registra je na globalnom (planetarnom) nivou

UDDI koncepti

- UDDI registar sadrži tri tipa informacija o organizaciji
 - **white pages**
osnovne informacije za kontakt, poreski identifikacioni broj (PIB), i slično
 - ovi podaci omogućavaju pronalaženje web servisa na osnovu podataka o konkretnom poslovnom partneru
 - **yellow pages**
informacije koje opisuju usluge web servisa pomoću svrstavanja u razne kategorije
 - ovi podaci omogućavaju pronalaženje web servisa po vrsti prozvoda/usluga koje organizacija nudi
 - **green pages**
tehničke informacije koje opisuju funkcije web servisa, uključujući i podatke o lokaciji servisa
 - ovi podaci omogućavaju pristup konkretnim web servisima date organizacije

Korisnici UDDI servisa

- poslovni službenici
 - koriste UDDI registar slično kao i Internet pretraživač
 - rezultat pretrage je skup podataka o nekoj organizaciji, ne samo njen URL
 - pošto svi podaci u UDDI registru nisu uvek reader-friendly, korisnici im mogu pristupati kroz razne portale
- softverski inženjeri
 - koriste UDDI registar da pronađu podatke o odgovarajućim servisima
 - ili da registruju sopstveni servis



UDDI arhitektura

- jedan logički jedinstveni servis na svetskom nivou
 - sastoji se iz više čvorova
 - lokalne baze podataka u čvorovima sinhronizuju se replikacijom
-
- dodavanje podataka u registar odvija se na jednom čvoru
 - taj čvor postaje "vlasnik" tih podataka (tj. upravlja njima)
 - sve kasnije izmene nad podacima moraju se izvesti na istom čvoru
-
- pristup bilo kom čvoru registra omogućava pristup svim podacima registra

UDDI arhitektura

- javni svetski UDDI registar - UBR (UDDI Business Registry)
- privatni UDDI registri po organizacijama
 - ne moraju biti deo UBR
- svaka organizacija može postaviti UDDI čvor i uključiti ga u UBR

UDDI specifikacije

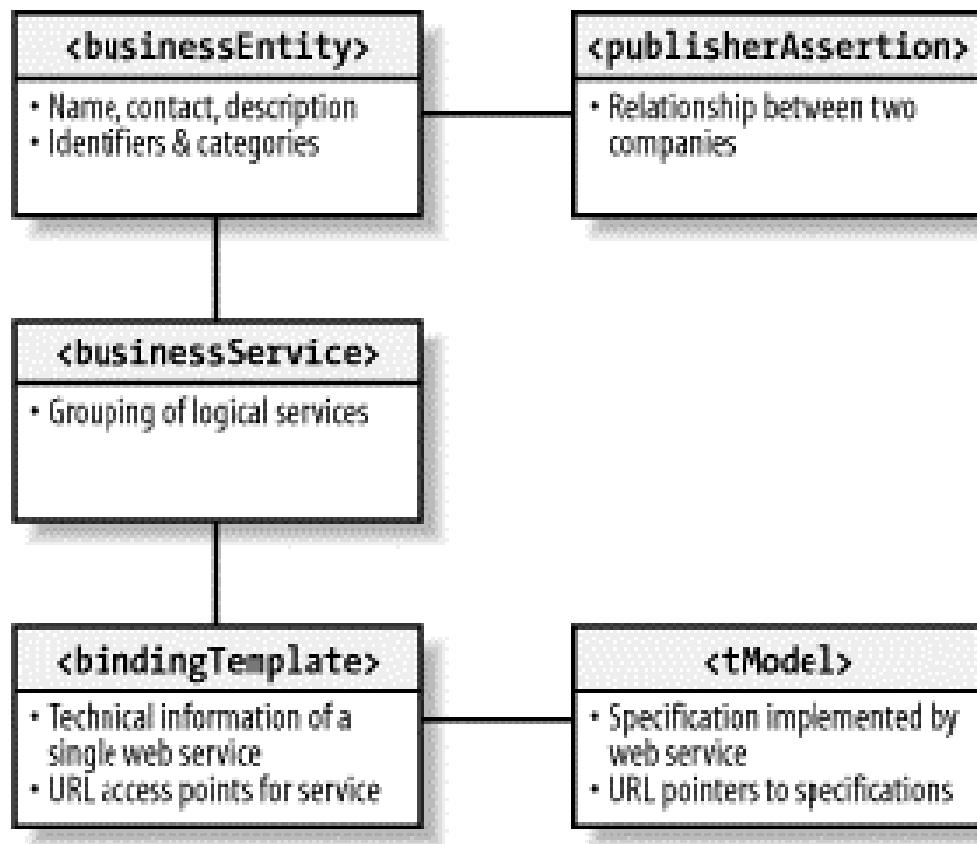
- sve specifikacije su dostupne na <http://www.uddi.org>
 - **UDDI replication**
opisuje procese i interfejse koje treba da implementira UDDI čvor kako bi obezbedio replikaciju u okviru UBR
 - **UDDI operators**
funkcionalni zahtevi postavljeni pred svaki čvor: skladištenje i backup podataka; nije obavezno za privatne registre
 - **UDDI programmer's API**
definiše funkcije za publikovanje i otkrivanje podataka o servisima; definisan je na nivou SOAP poruka
 - **UDDI data structures**
definiše strukture sadržane u SOAP porukama kojima se komunicira sa registrom
- UDDI XML API - XML Schema dokument koji definiše tipove podataka i strukture za UDDI

UDDI model podataka

- podaci u UDDI registru sastoje se iz entiteta koji se skladište u čvorovima i izražavaju se XML jezikom
- model poseduje sledeće tipove entiteta
 - **businessEntity**
opisuje organizaciju
 - **businessService**
opisuje usluge (servise) koje nudi organizacija
 - **bindingTemplate**
opisuje tehničke informacije potrebne za pristup uslugama
 - **tModel**
opisuje "tehnički model" nekog koncepta, npr. tip web servisa, protokol koji koristi web servis u cilju ponovnog korišćenja (reuse) iste definicije kroz celokupan sistem
 - **publisherAssertion**
opisuje vezu jednog businessEntity sa drugim businessEntity-ima
 - **subscription**
opisuje zahtev za praćenjem promena u okviru nekog entiteta

UDDI model podataka

- veze između tipova entiteta



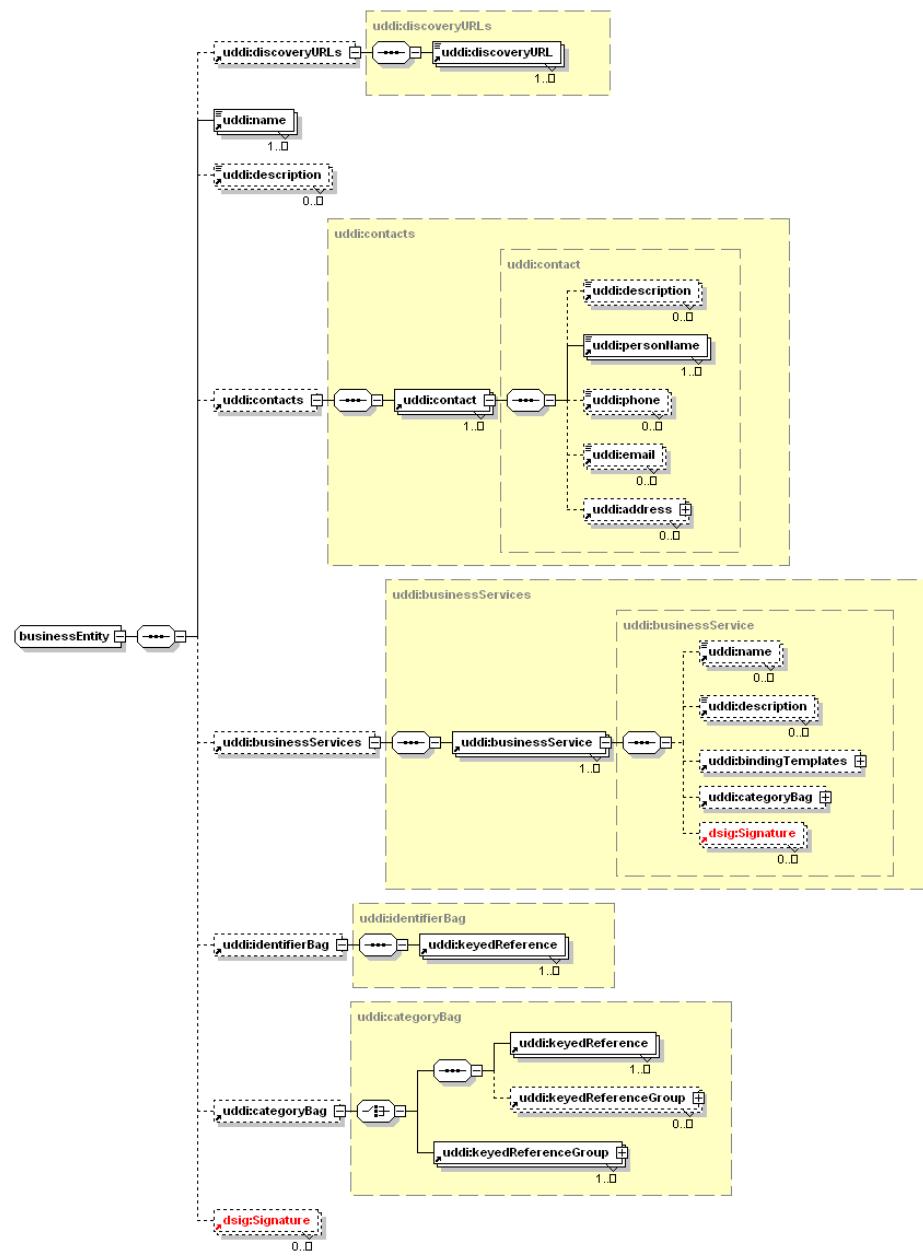
tModel može biti izdvojen kao posebna stavka u bazi

- tModel predstavlja interfejs nekog servisa
- ako se tModel definiše na nivou grupe organizacija, sve organizacije dele isti tModel
 - npr. avio kompanije dogovore jedinstveni WSDL za svoje servise za rezervaciju karata
 - WSDL se u UDDI bazi predstavi kroz tModel

UDDI model podataka

- nad skupom podataka UDDI registra moguće je definisati više taksonomskih klasifikacija
- mogućnosti za šifriranje
 - delatnosti
 - proizvoda
 - geografskog položaja
- kontrolisani i nekontrolisani šifarnici
 - *checked value sets*: unete vrednosti moraju pripadati konačnom skupu
 - *unchecked value sets*: unete vrednosti se ne proveravaju

uddi:businessEntity



uddi:businessEntity

- korenski element u UDDI modelu podataka
- atributi
 - businessKey: identifikator organizacije
 - operator: identifikator UDDI servera koji je vlasnik podataka
 - authorizedName: identifikator osobe koja je postavila informacije na server

```
<businessEntity  
    xmlns="urn:uddi-org:api_v2"  
    businessKey="e9355551-32aa-494f-8eb41ce59"  
    operator="IBM Inc."  
    authorizedName="Sampler">  
  
    ...  
  
</businessEntity>
```

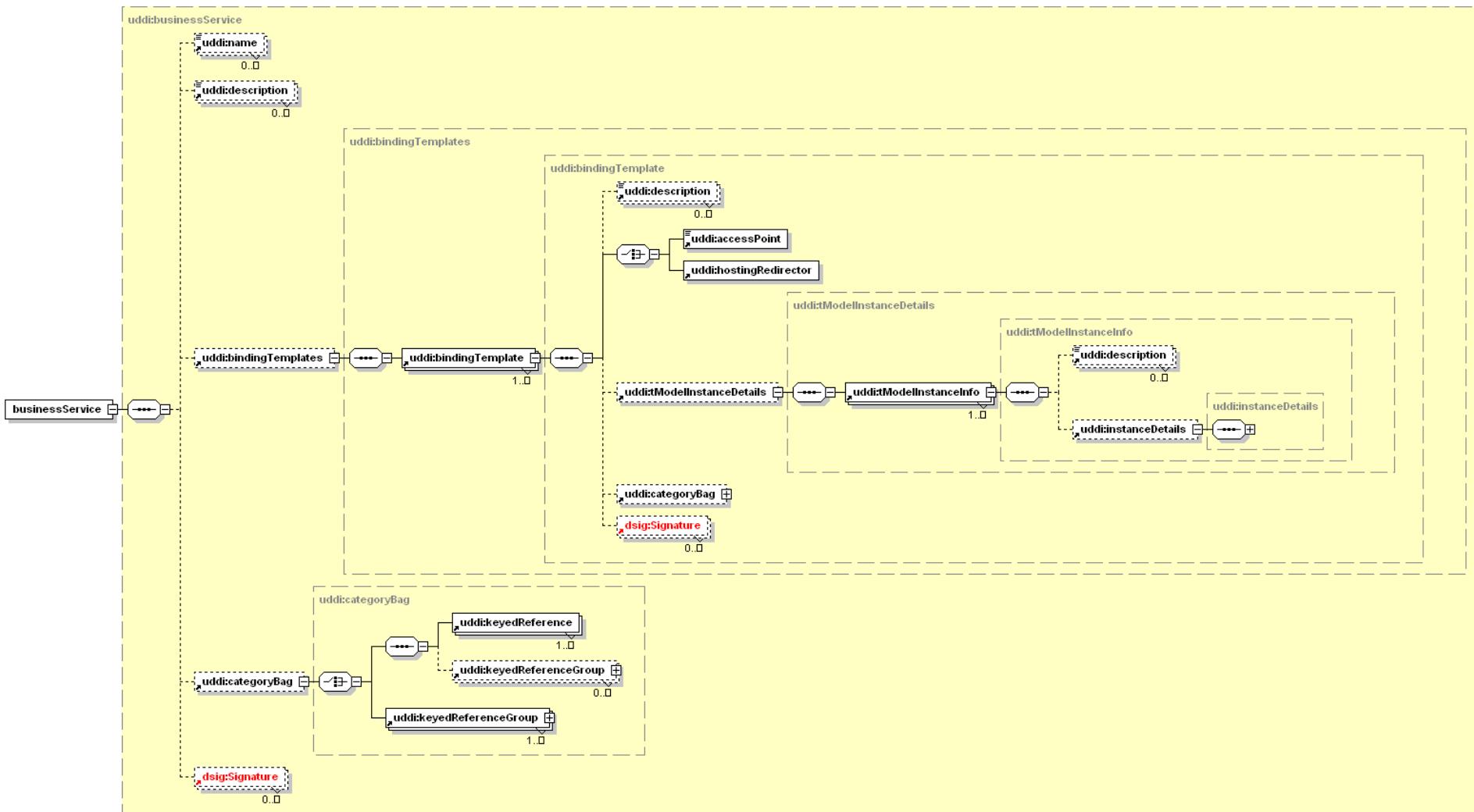
uddi:businessEntity

- podelementi

- discoveryURLs: linkovi ka resursima sa više detalja o organizaciji
- name: naziv organizacije
- description: kratak opis organizacije
- contacts: informacije za kontakt
- businessServices: businessService elementi koji opisuju delatnost
- identifierBag: identifikatori organizacije
- categoryBag: šifrirane oznake delatnosti organizacije

```
<businessEntity . . .>
  <discoveryURLs>
    <discoveryURL useType="businessEntity">
      http://...
    </discoveryURL>
  </discoveryURLs>
  <name xml:lang="en">Sampling Inc</name>
  <description xml:lang="en">
    Sampling is a one-stop portal...
  </description>
</businessEntity>
```

uddi:businessService



uddi:businessService

- opisuje usluge (servise) koje nudi organizacija
- atributi
 - serviceKey: identifikator usluge koju pruža organizacija
 - businessKey: identifikator organizacije (iz businessEntity elementa)

```
<businessEntity>
    ...
<businessServices>
    <businessService
        serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a"
        businessKey="e9355551-32aa-494f-8eb41ce59">
        <name xml:lang="en">
            Sample Code Home Page
        </name>
    </businessService>
</businessServices>

</businessEntity>
```

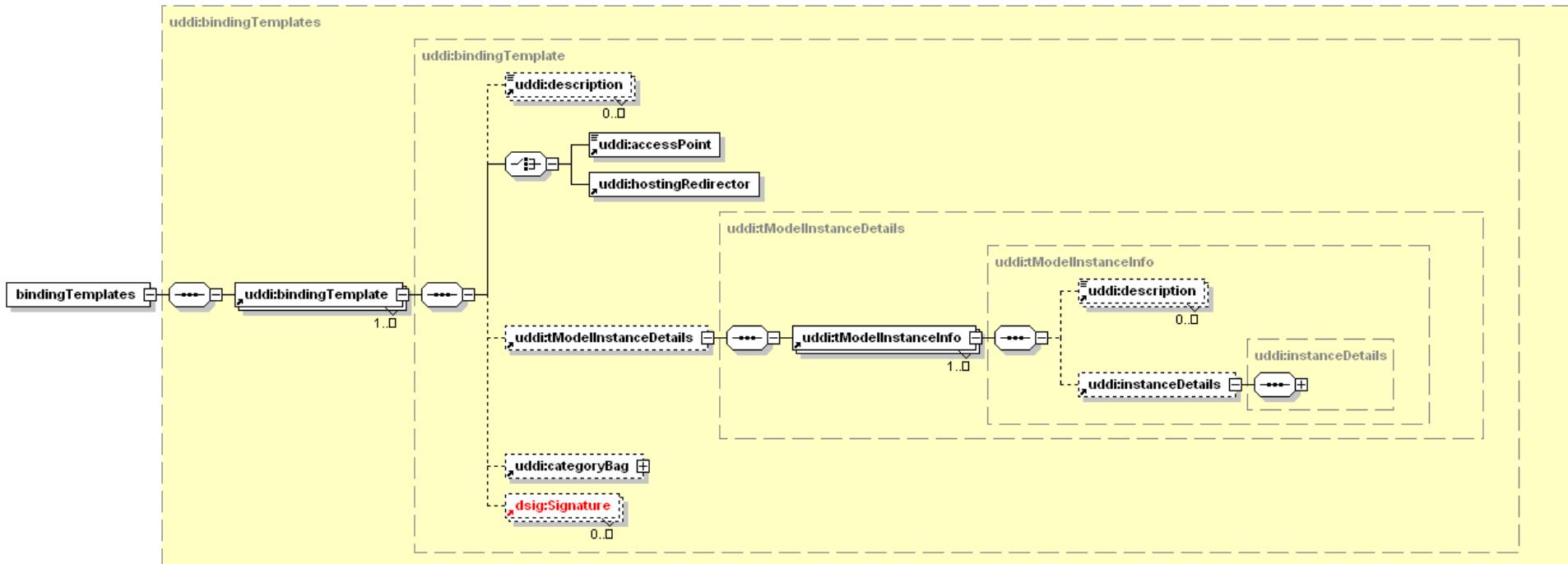
uddi:businessService

- podelementi
 - name: naziv usluge
 - serviceDescription: kratak opis usluge
 - bindingTemplates: dalje opisuju uslugu
 - categoryBag: kategorizacija delatnosti

```
<businessEntity>
    ...
<businessServices>
    <businessService
        serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a"
        businessKey="e9355551-32aa-494f-8eb41ce59">
        <name xml:lang="en">
            Sample Code Home Page
        </name>
    </businessService>
</businessServices>

</businessEntity>
```

uddi:bindingTemplate



uddi:bindingTemplate

- opisuje tehničke informacije potrebne za pristup uslugama
- atributi
 - serviceKey: identifikator usluge
 - bindingKey: identifikator bindingTemplate elementa

```
<businessEntity>
...
<businessServices>
...
<bindingTemplate
    bindingKey="48b02d40-0312-4293-a7f5-4449ca190984"
    serviceKey="1eeecfa1-6f99-460e-a392-8328d38b763a">
...
</bindingTemplate>

</businessServices>

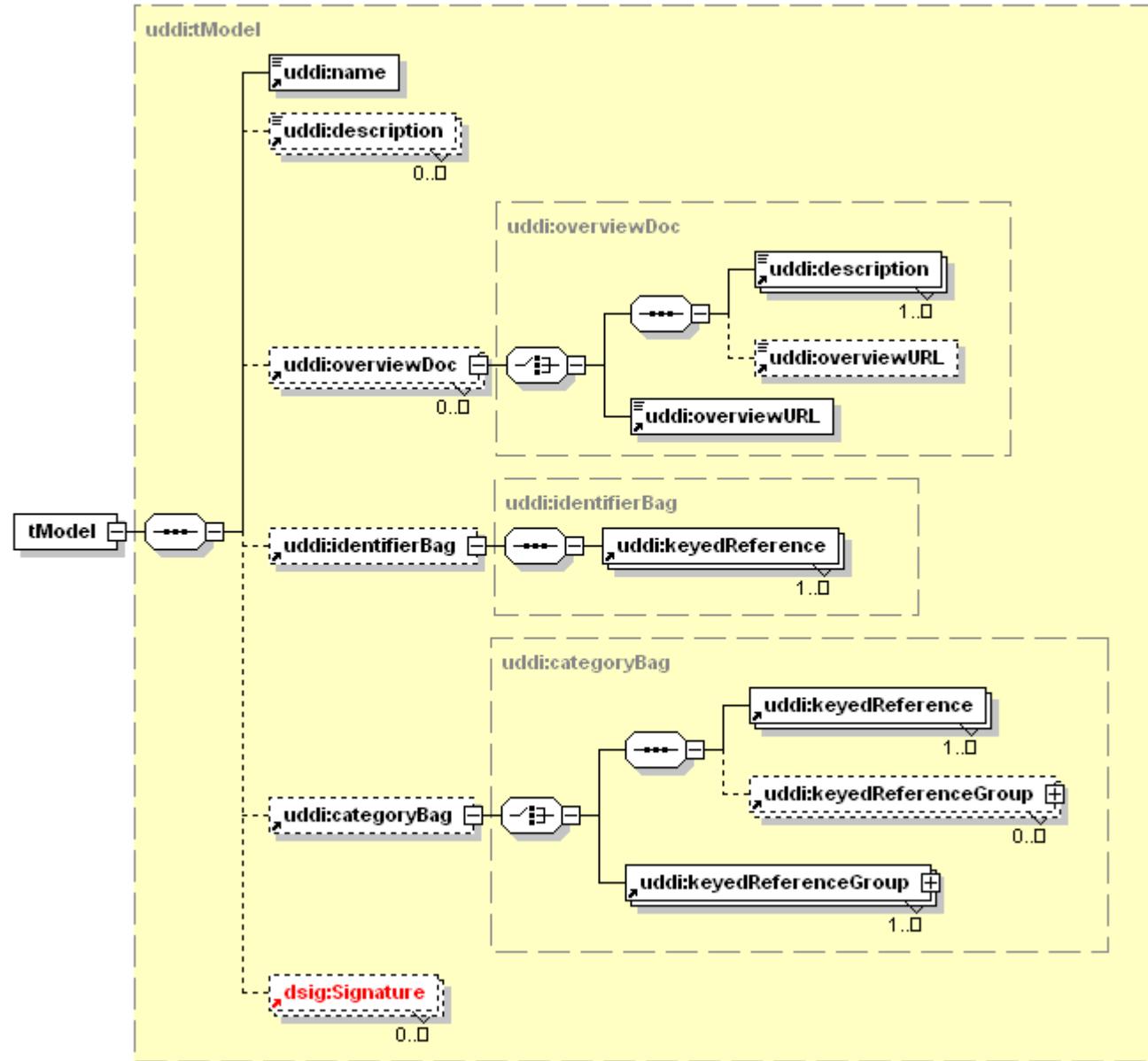
</businessEntity>
```

uddi:bindingTemplate

- podelementi
 - description: kratak opis servisa
 - accessPoint: adresa na kojoj se nalazi web servis
 - hostingRedirector: referencia na drugi bindingTemplate
 - tModelInstanceDetails: reference na tehničke modele

```
<bindingTemplate ...>
  <description xml:lang="en">...</description>
  <accessPoint URLType="http">http://www.sample-code.com</accessPoint>
  <tModelInstanceDetails>
    <tModelInstanceInfo
      tModelKey="uuid:8a056b70-bfe8-4fac-90cd-820c26dc2e48">
      <instanceDetails>
        <overviewDoc>
          <overviewURL>
            http://www.example.com/sample/rss.aspx
          </overviewURL>
        </overviewDoc>
      </instanceDetails>
    </tModelInstanceInfo>
  </tModelInstanceDetails>
</bindingTemplate>
```

uddi:tModel



uddi:tModel

- ukazuje na definicije web servisa
- tModel ima UUID koji se generiše prilikom registracije
 - koristi se za ukazivanje na tehničke detalje: transportni protokol, formati podataka
- atributi
 - tModelKey: vrednost UUID-a tModela
 - operator: UDDI server koji je vlasnik tModela
 - authorizedName: identifikator osobe koja je publikovala informacije

```
<tModel  
    tModelKey="uuid:5DD52389-B1A4-4fe7-B131-0F8EF73DD175">  
    <name>this is the interface we created</name>  
    <description xml:lang="en">...</description>  
    <overviewDoc>  
        <description xml:lang="en">The service's WSDL document</description>  
        <overviewURL>  
            http://www.sample-code.com/services/interfaces/sample.wsdl  
        </overviewURL>  
    </overviewDoc>  
</tModel>
```

uddi:tModel

- podeljeni elementi
 - name: naziv tModela
 - description: kratak opis tModela
 - overviewDoc: resursi sa detaljnim opisom tModela
 - identifierBag: identifikatori tModela
 - categoryBag: kategorizacija tModela

```
<tModel  
    tModelKey="uuid:5DD52389-B1A4-4fe7-B131-0F8EF73DD175">  
    <name>this is the interface we created</name>  
    <description xml:lang="en">...</description>  
    <overviewDoc>  
        <description xml:lang="en">The service's WSDL document</description>  
        <overviewURL>  
            http://www.sample-code.com/services/interfaces/sample.wsdl  
        </overviewURL>  
    </overviewDoc>  
</tModel>
```

tModel kao pointer na WSDL

uddi:tModel

- kategorizacija (na primeru tModela)
 - način da tModel opišemo metapodacima pomoću kojih će neko pronaći naš servis
 - UDDI core models: predefinisani tModeli namenjeni klasifikaciji
 - wsdlSpec uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4
 - namenjen da neki konkretni tModel predstavi kao tModel koji se oslanja na WSDL-bazirani web servis

```
<tModel  
    tModelKey="uuid:5DD52389-B1A4-4fe7-B131-0F8EF73DD175">  
    <name>this is the interface we created</name>  
    <description xml:lang="en">...</description>  
    <overviewDoc>  
        <description xml:lang="en">The service's WSDL document</description>  
        <overviewURL>  
            http://www.sample-code.com/services/interfaces/sample.wsdl  
        </overviewURL>  
    </overviewDoc>  
    <categoryBag>  
        <keyedReference  
            tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"  
            keyName="Specification for a web service described in WSDL"  
            keyValue="wsdlSpec"/>  
    </categoryBag>  
</tModel>
```

sada nas mogu pronaći po
ključu "wsdlSpec"

uddi:tModel

- kategorizacija (na primeru tModela)
 - dodamo još jednu kategorizaciju
 - "On-Line Information Services" uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2

```
<tModel  
    tModelKey="uuid:5DD52389-B1A4-4fe7-B131-0F8EF73DD175">  
    ...  
    <categoryBag>  
        <keyedReference  
            tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"  
            keyName="Specification for a web service described in WSDL"  
            keyValue="wsdlSpec"/>  
        <keyedReference  
            tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2"  
            keyName="On-Line Information Services"  
            keyValue="514191"/>  
    </categoryBag>  
</tModel>
```

sada nas mogu pronaći i po ključu
"On-Line Information Services"

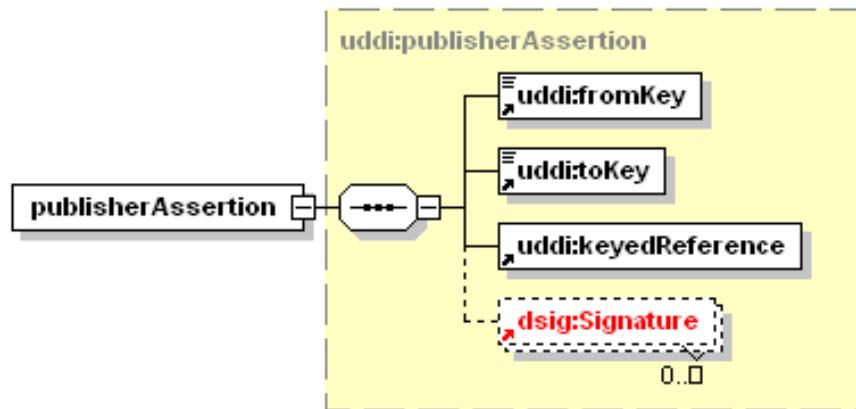
uddi:tModel

- kategorizacija (na primeru tModela)
 - dodamo još opis parametara našeg servisa (da se ne mora analizirati WSDL)
 - ulaz - string
 - izlaz - string

```
<categoryBag>
  <keyedReference
    tModelKey="uuid:c1acf26d-9672-4404-9d70-39b756e62ab4"
    keyName="Specification for a web service described in WSDL"
    keyValue="wsdlSpec"/>
  <keyedReference
    tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2"
    keyName="On-Line Information Services"
    keyValue="514191"/>
  <keyedReference
    tModelKey="uuid:E27972D8-717F-4516-A82D-B688DC70170C"
    keyName="whatever-input-name"
    keyValue="xsd:string"/>
  <keyedReference
    tModelKey="uuid:Key_for_output_tModel"
    keyName="whatever-output-name"
    keyValue="xsd:integer"/>
</categoryBag>
```

posebni tModeli koji predstavljaju ulazne i izlazne parametre!

uddi:publisherAssertion



uddi:publisherAssertion

- veza između organizacija (businessEntity)
- podelementi
 - fromKey: identifikator prve organizacije u vezi
 - toKey: identifikator druge organizacije u vezi
 - keyedReference: opisuje vrstu veze

UDDI Programmer APIs

- Inquiry API
pronalaženje i preuzimanje podataka o entitetima u registru
- Publication API
publikovanje i ažuriranje informacija o entitetima u registru
- Security Policy API
autentifikacija UDDI klijenata
- Custody and Ownership Transfer API
definisanje vlasništva nad entitetima među čvorovima UDDI registra
- Subscription API
registracija radi praćenja promena nad željenim entitetima u registru
- Value Set API
"održavanje šifarnika"

Inquiry API

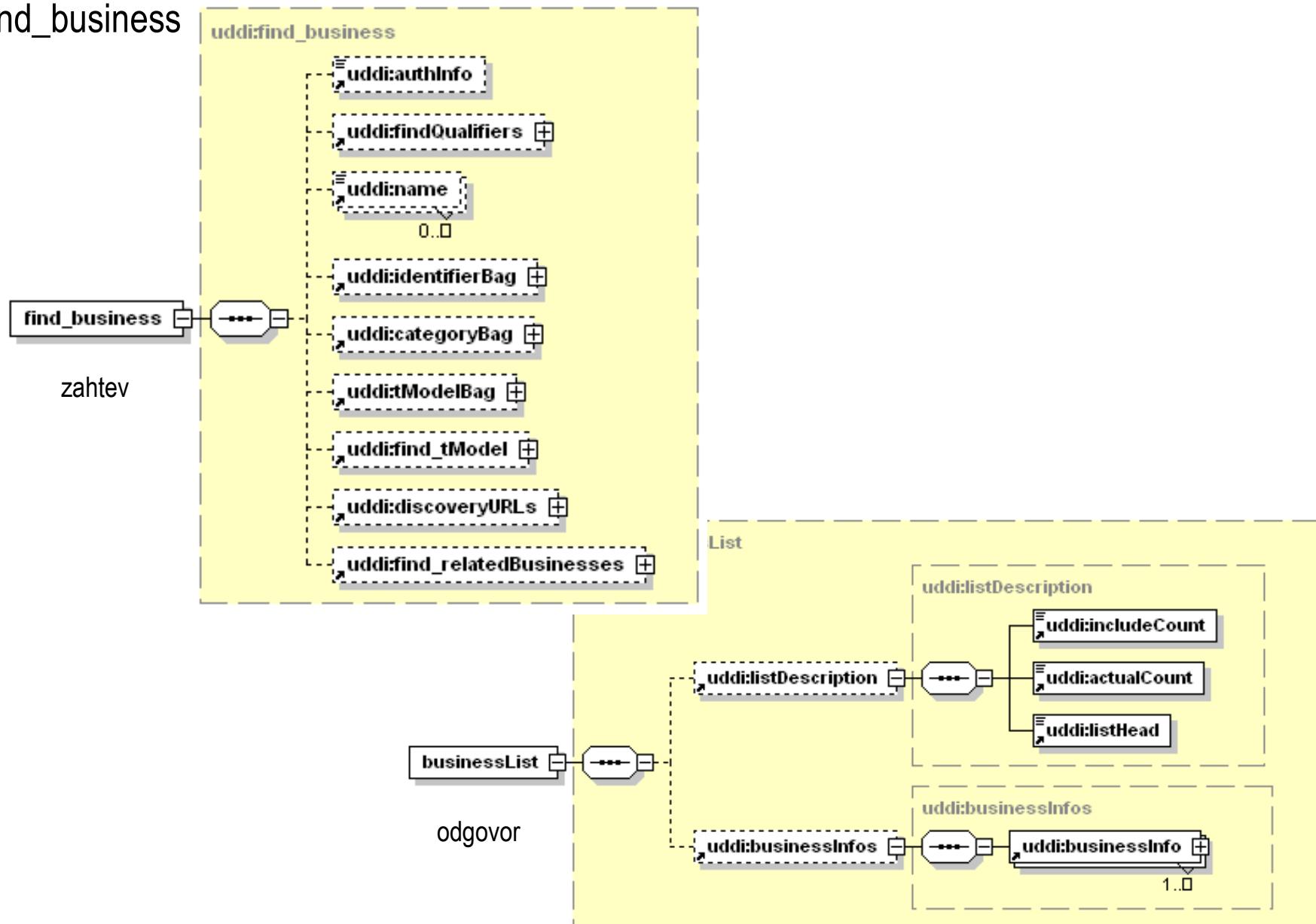
- tri režima preuzimanja podataka
 - browse pattern
 - pregledanje širokog skupa podataka, izbor nekog entiteta i prelazak na drill-down
 - find_xx metode
 - drill-down pattern
 - na osnovu rezultata find_xx metoda - identifikatora entiteta - preuzimanje detalja o entitetu
 - get_xx metode
 - invocation pattern
 - na osnovu bindingTemplate podataka za dati servis on se može i pozvati
 - statički klijent (unapred napisan za dati WSDL)
 - dinamički klijent (otkriva metode servisa tokom izvršavanja i samostalno bira one koje će pozvati)

Inquiry API

- find_xx
 - find_business
 - find_service
 - find_binding
 - find_tModel
 - find_relatedBusiness
- get_xx
 - get_businessDetail
 - get_serviceDetail
 - get_bindingDetail
 - get_tModelDetail
 - get_operationallInfo

Inquiry API

- find_business



Inquiry API

- find_business

```
<?xml version="1.0">
<SOAP:Envelope
    SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP="http://schemas.xmlsoap.org/ soap/envelope/">
    <SOAP:Body>
        <find_business xmlns="urn:uddi-org:api">
            <name>
                Sampling Inc
            </name>
        </find_business>
    </SOAP:Body>
</SOAP:Envelope>
```

Inquiry API

- get_serviceDetail

```
<?xml version="1.0">
<SOAP:Envelope
    SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP="http://schemas.xmlsoap.org/ soap/envelope/">
    <SOAP:Body>
        <get_serviceDetail xmlns="urn:uddi-org:api">
            <serviceKey>
                1eecfa1-6f99-460e-a392-8328d38b763a
            </serviceKey>
        </get_serviceDetail>
    </SOAP:Body>
</SOAP:Envelope>
```

Inquiry API

- get_bindingDetail

```
<?xml version="1.0">
<SOAP:Envelope
    SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP="http://schemas.xmlsoap.org/ soap/envelope/">
    <SOAP:Body>
        <get_bindingDetail xmlns="urn:uddi-org:api">
            <bindingKey>
                48b02d40-0312-4293-a7f5-4449ca190984
            </bindingKey>
        </get_bindingDetail>
    </SOAP:Body>
</SOAP:Envelope>
```

Inquiry API

```
<find_business xmlns="urn:uddi-org:api_v3">

<!-- Iznajmljivanje odela i venčanica na teritoriji Srbije --&gt;

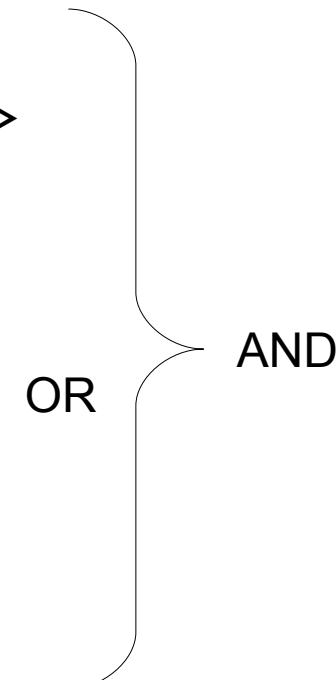
&lt;findQualifiers&gt;
&lt;findQualifier&gt;uddi:uddi.org:findqualifier:orlikekeys&lt;/findQualifier&gt;
&lt;findQualifier&gt;uddi:uddi.org:findqualifier:combinecategorybags&lt;/findQualifier&gt;
&lt;findQualifier&gt;uddi:uddi.org:findqualifier:approximatematch&lt;/findQualifier&gt;
&lt;/findQualifiers&gt;

&lt;categoryBag&gt;
    <!-- Bilo kakva radnja za odeću --&gt;
    &lt;keyedReference keyValue="7810%" tModelKey="uddi:uddi.org:ubr:categorization:naics:1997"/&gt;

    <!-- Iznajmljivanje odela --&gt;
    &lt;keyedReference keyValue="91.10.18.01.00" tModelKey="uddi:uddi.org:ubr:categorization:unspsc"/&gt;

    <!-- Iznajmljivanje venčanica --&gt;
    &lt;keyedReference keyValue="91.10.18.02.00" tModelKey="uddi:uddi.org:ubr:categorization:unspsc"/&gt;

    <!-- Teritorija Srbije --&gt;
    &lt;keyedReference keyValue="RS" tModelKey="uddi:uddi.org:ubr:categorization:iso3166"/&gt;
&lt;/categoryBag&gt;
&lt;/find_business&gt;</pre>
```



RESTful Web servisi

Šta je REST?

- REST = Representational State Transfer
- Stil softverske arhitekture namenjen distribuiranim hipermedijalnim sistemima, kao što je World Wide Web
- Termin „skovao“ Roy Fielding u svojoj disertaciji na University of California, Irvine.

Šta je REST?

- Representational State Transfer (REST) - stil arhitekture sistema koji specificira određena ograničenja:
 1. Način identifikacije resursa
 2. uniformisani interfejs
GET, PUT, DELETE, POST (HEAD, OPTIONS...)
 3. Samoopisive poruke
 4. Stanje aplikacije upravljano hipermedijom (linkovi predstavljaju endpointe kojim se manipuliše resursima, manipulacija resursima menja tekuće stanje aplikacije)
 5. Stateless interakcije
- Primena ovih ograničenja na web servise pojačava pozitivne osobine, kao što su performanse, skalabilnost, izmenjivost.

Šta je REST?

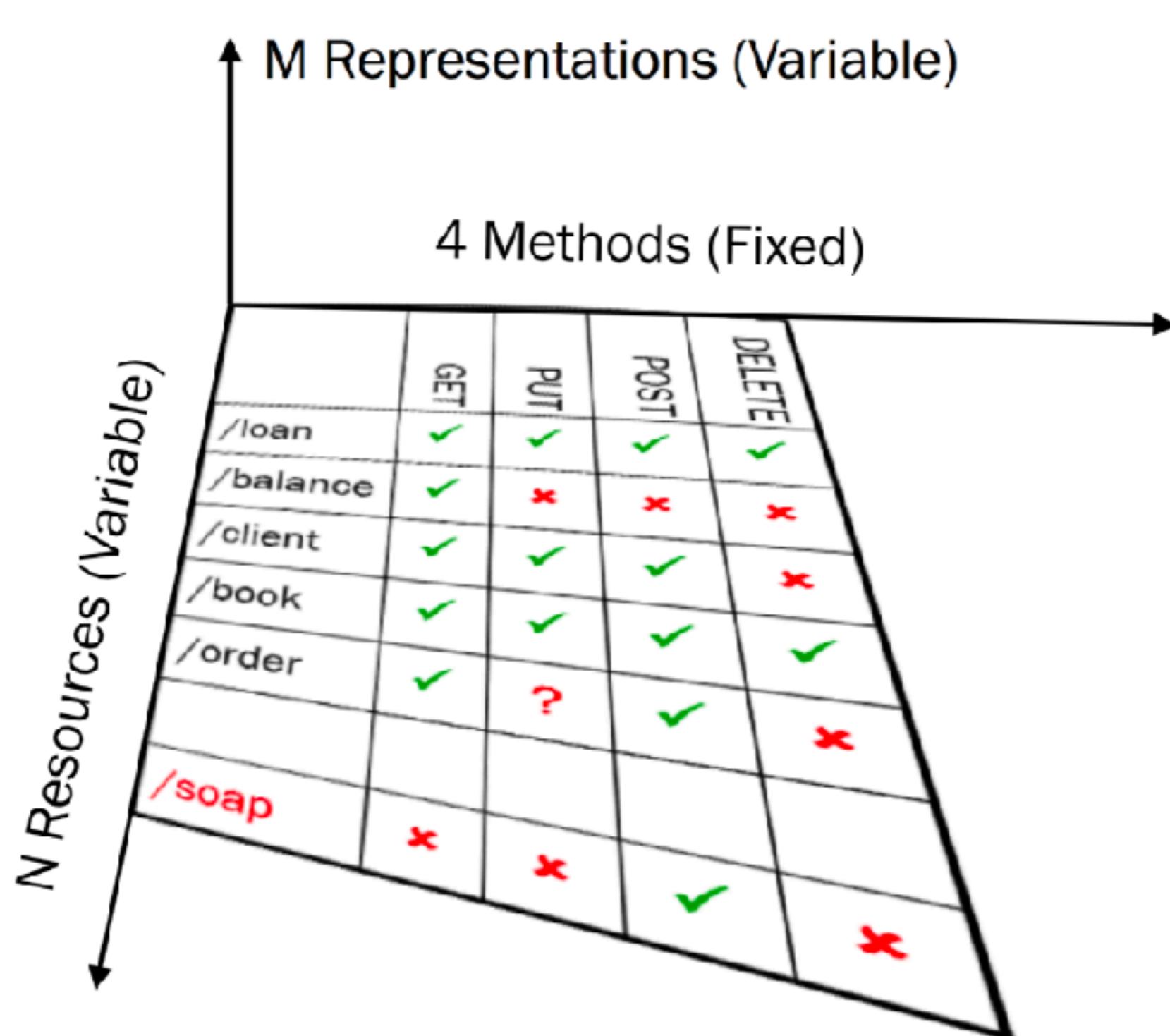
- U REST arhitekturi, podaci i funkcionalnost se posmatraju kao resursi i pristupa im se putem Uniform Resource Identifiers (URIs), tipično kao linkovima na web-u.
- Resursima se manipuliše primenom skupa jednostavnih, dobro definisanih operacija.
- REST arhitektura je klijent/server i dizajnirana je da koristi stateless komunikacioni protokol, tipično HTTP.
 - klijent i server razmenjuju **reprezentacije** resursa koristeći pri tome standardizovan interfejs i protokol.

Dizajn REST aplikacije / metodologija

1. Identifikovanje resursa koji treba da su vidljivi kao servis (npr. godišnji izveštaji, katalozi knjiga, porudžbine...)
2. Modelovanje relacija između resursa kao hiperlinkova koje je moguće slediti kako bi se dobilo više detalja (ili kako bi se izvela promena stanja resursa)
3. Definisati „lepe“ URI-je za adresiranje resursa
4. Razumeti smisao izvršavanja GET, POST, PUT, DELETE zahteva na svaki od resursa (i sa li su svi i dozvoljeni za svaki resurs)
5. Dizajnirati i dokumentovati reprezentaciju resursa (može biti više)
6. Implementirati i postaviti na web server
7. Testirati (browser, PostMen...)

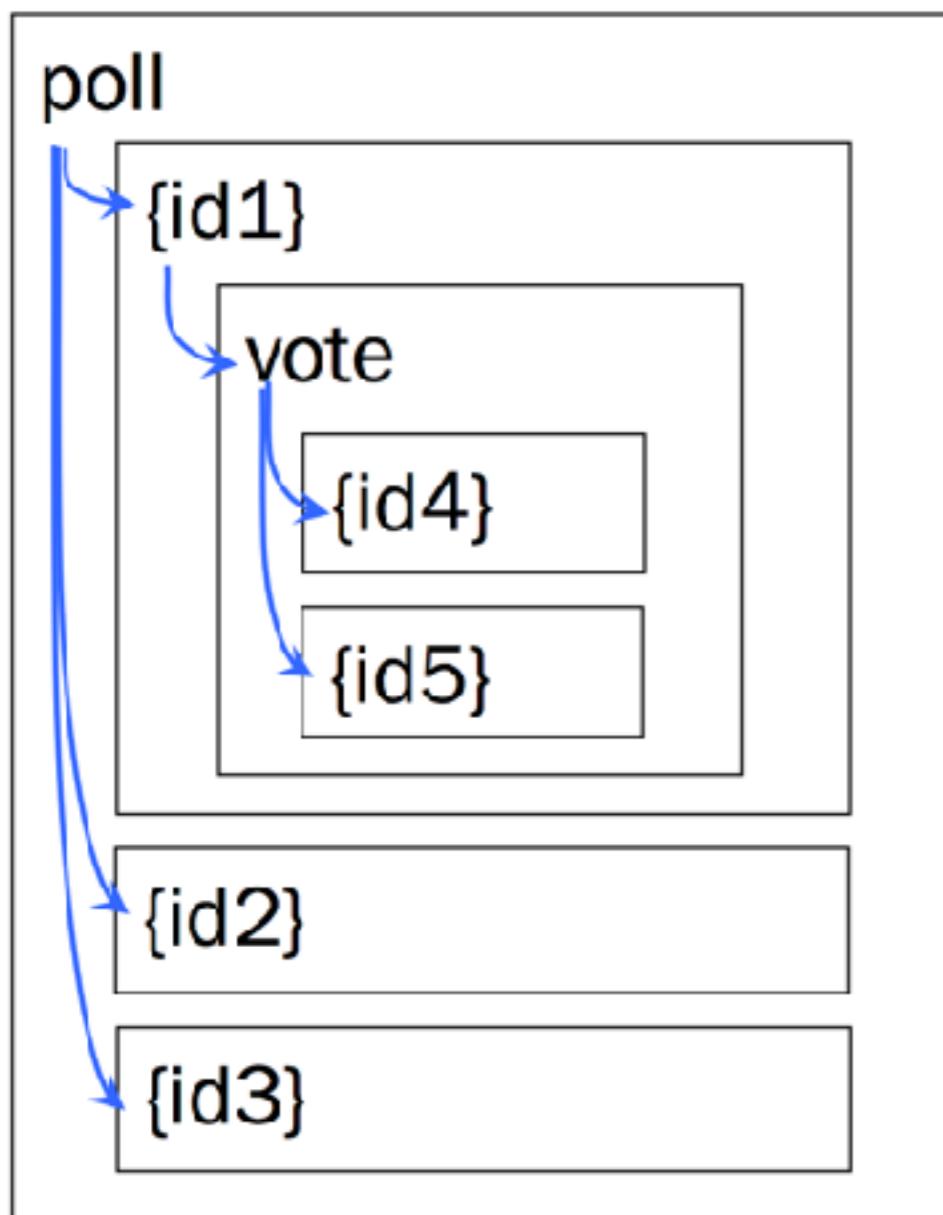
	GET	PUT	POST	DELETE
/loan	✓	✓	✓	✓
/balance	✓	✗	✗	✗
/client	✓	✓	✓	✗
/book	✓	✓	✓	✓
/order	✓	?	✓	✗
/soap	✗	✗	✓	✗

REST aplikacije / prostor modelovanja i razvoja



Primer simplifikovanog Doodle API-ja

1. Resursi su:
anketa i glasovi
2. Relacije između resursa:



	GET	PUT	POST	DELETE
/poll	✓	✗	✓	✗
/poll/{id}	✓	✓	✗	✓
/poll/{id}/vote	✓	✗	✓	✗
/poll/{id}/vote/{id}	✓	✓	✗	?

3. URLs sadrže ID-eve resursa
4. POST na URI kontejnera se koristi da kreira novi resurs
5. PUT/DELETE se koristi za ažuriranje ili brisanje resursa **/{id}**

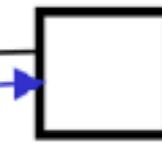
Primer simplifikovanog Doodle API-ja

1. Kreiranje ankete

(vrši se transfer stanja nove ankete na Doodle servis)



/poll
/poll/090331x
/poll/090331x/vote



POST /poll
<options>A, B, C</options>

201 Created
Location: /poll/090331x

GET /poll/090331x

200 OK
<options>A, B, C</options>
<votes href="/vote"/>

2. Čitanje ankete

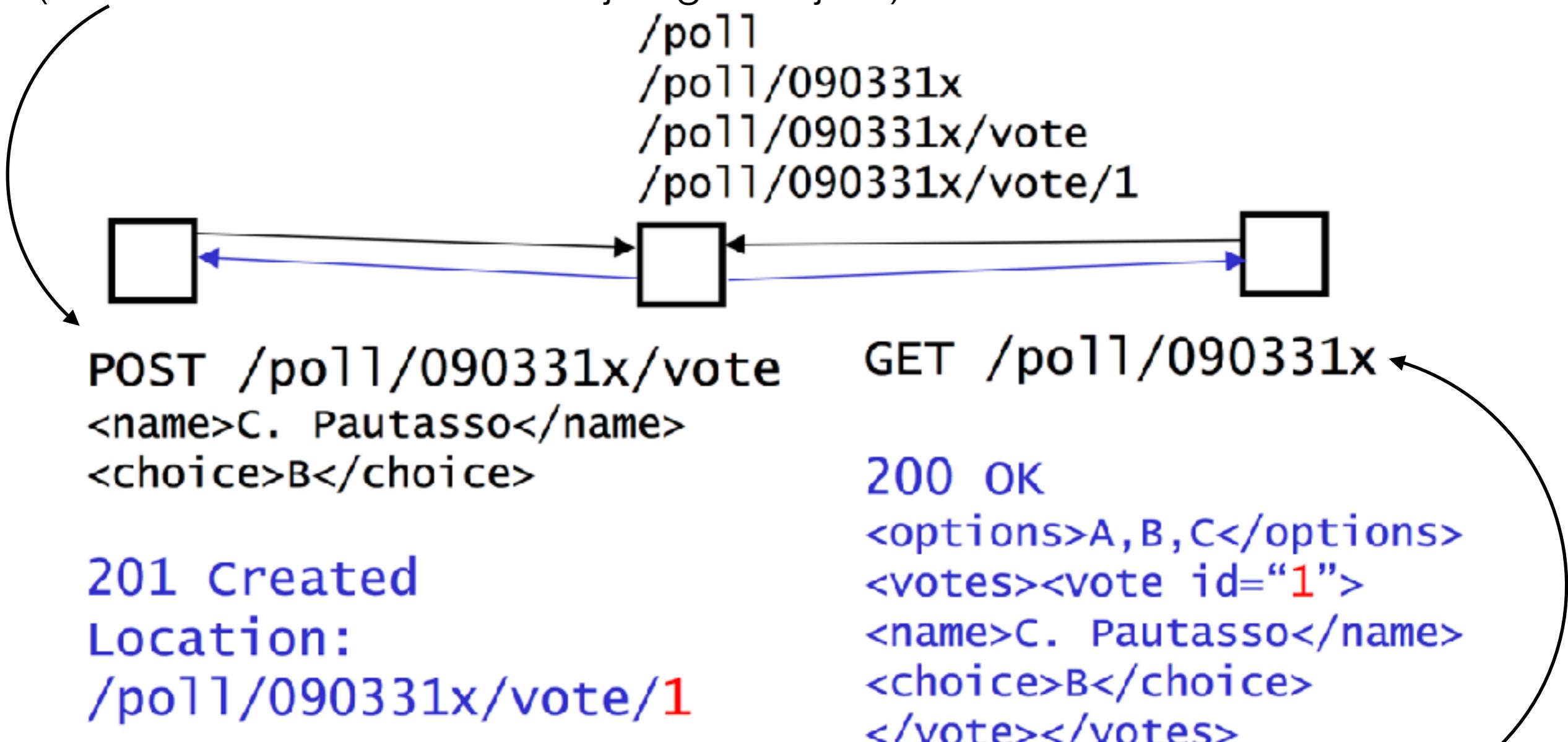
(vrši se transfer stanja ankete sa Doodle servisa)

Primer simplifikovanog Doodle API-ja

3. Učestvovanje u anketi

(kreira se novi resurs sa mojim glasanjem)

/poll
/poll/090331x
/poll/090331x/vote
/poll/090331x/vote/1

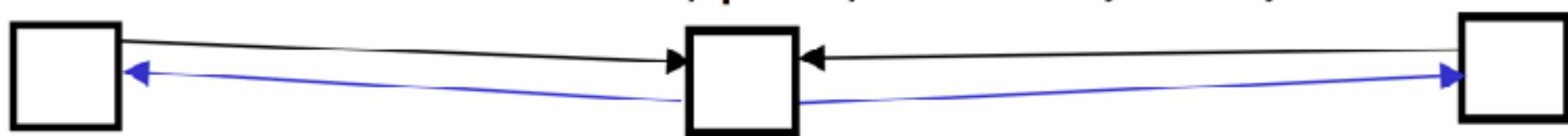


4. Čitanje ankete
(vrši se transfer stanja ankete sa Doodle servisa)

Primer simplifikovanog Doodle API-ja

5. Izmena postojećeg glasanja

/poll
/poll/090331x
/poll/090331x/vote
/poll/090331x/vote/1



PUT /poll/090331x/vote/**1** GET /poll/090331x
<name>C. Pautasso</name>
<choice>**C**</choice>

200 OK

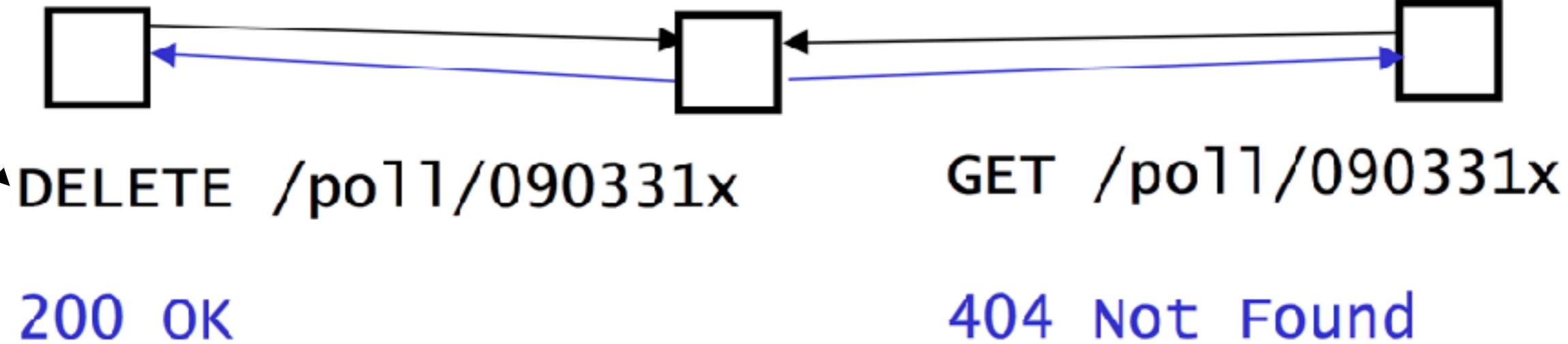
200 OK
<options>A,B,C</options>
<votes><vote id="/1">
<name>C. Pautasso</name>
<choice>**C**</choice>
</vote></votes>

6. Opet čitanje ankete
(vrši se transfer stanja ankete sa Doodle servisa)

Primer simplifikovanog Doodle API-ja

7. Po potrebi cela anketa se može obrisati

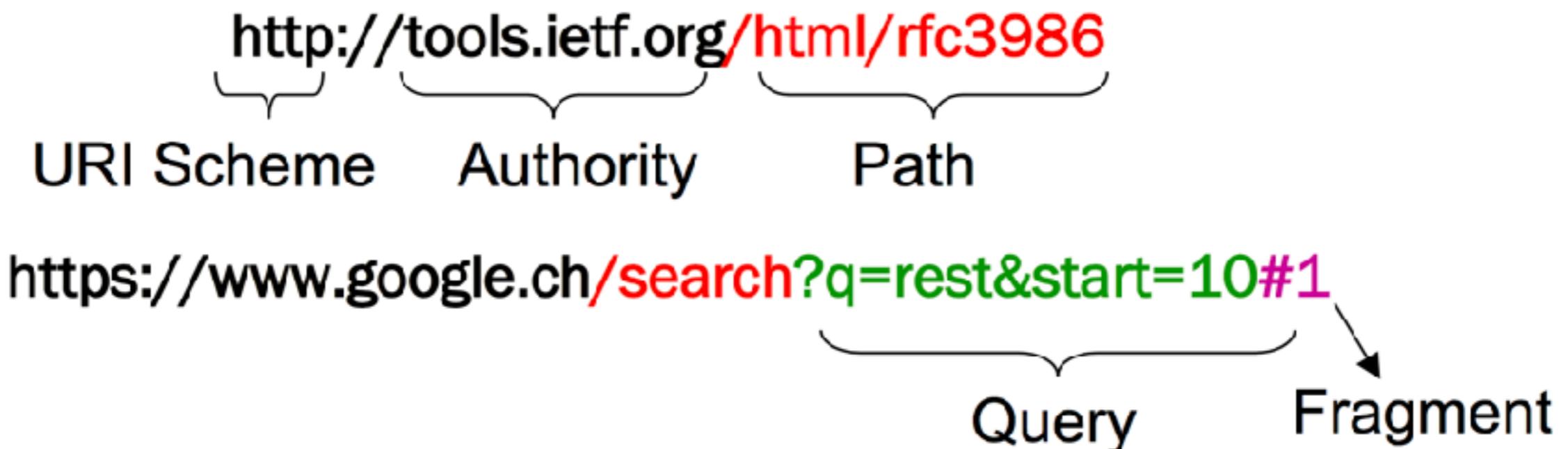
/poll
/poll/090331x
/poll/090331x/vote
/poll/090331x/vote/1



6. Pokušaj čitanja sada više ne uspeva

URI

- Internet Standard za imenovanje i identifikaciju resursa (originalni iz 1994, revidiran od 2005)
- Primer:



Šta je „lep“ URI

- Poželjno je putanju do resursa pretvoriti u niz segmenata, a ne koristiti key=value parove u query-ju
- bolje
<http://www.mojaknjizara.com/knjige/beletristica>
- nego
[http://www.mojaknjizara.com?
katalog=knjige&kategorija=beletristica](http://www.mojaknjizara.com?katalog=knjige&kategorija=beletristica)

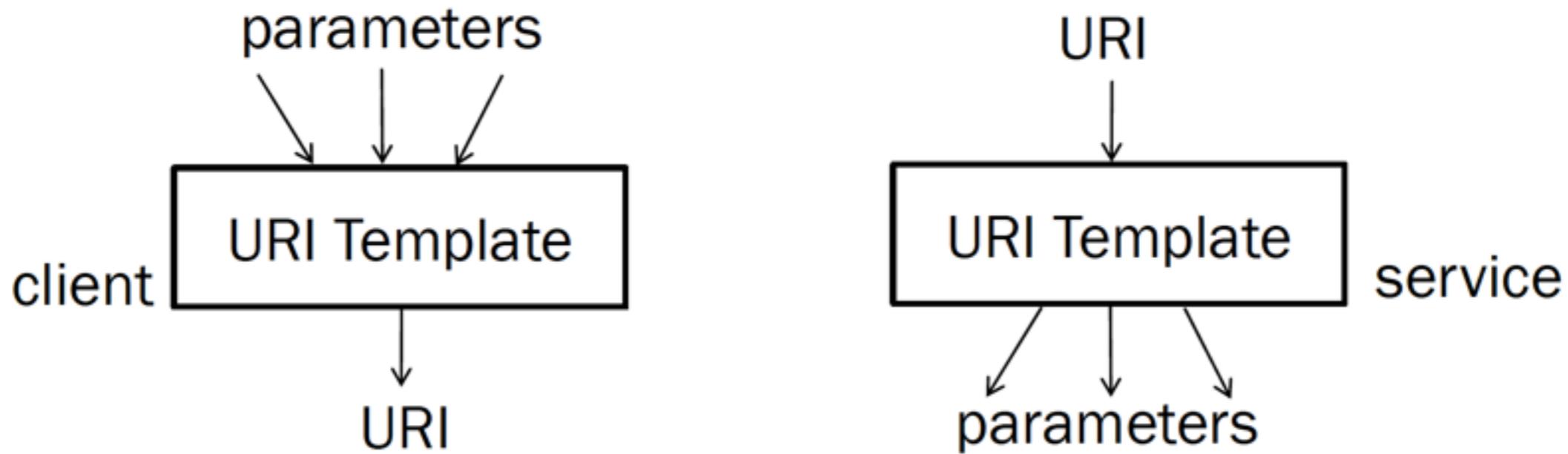
Smernice za osmišljavanje URI-ja

- Preferifaju se imenice a ne glagoli za imenovanje resursa
 - Težite ka tome da URI bude kratak
 - Ako je moguće koristite pozicionu šemu prosleđivanja parametara, a ne query string
 - Ponekad se koriste URI postfixi da se specificira tip sadržaja
 - Ne menjajte URI-je za resurse
 - Ako baš morate koristite redirekciju
- GET /book?isbn=24&action=delete
- DELETE /book/24
- Često se koristi URI templeting, ali to onda ostvaruje čvršće povezivanje klijenta sa serverskom implementacijom

URI šabloni

- URI Templates (šabloni) specificiraju kako konstruisati i parsirati parametrizovane URI-je.
- Na servisnoj strani obično se koriste “ruting pravila”
- Na klijentskoj strani ovi šabloni se koriste da se konsturiše URI do resursa na osnovu lokalnih parametara.

URI šabloni



- Izbegnite hardkodiranje URI-ja u klijentskoj aplikaciji
- Smanjite međuzavisnost od servera tako što ćete povući URI šablon sa servera i dinamički popunjavati na klijentu

URI šablony

- Template:

`http://www.myservice.com/order/{oid}/item/{iid}`

- Example URI:

`http://www.myservice.com/order/XYZ/item/12345`

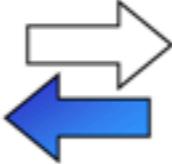
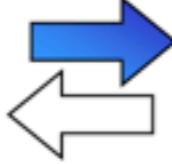
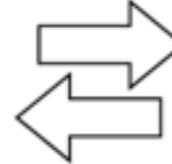
- Template:

`http://www.google.com/search?{-join|&|q,num}`

- Example URI:

`http://www.google.com/search?q=REST&num=10`

Uniforman interfejs

CRUD	REST	
CREATE	POST 	Krerira (pod)resurs
READ	GET 	Preuzima trenutno stanje resursa
UPDATE	PUT 	Ažurira stanje resursa na zadatom URI-ju
DELETE	DELETE 	Uklanja resurs. Nakon toga URI više nije validan.

HTML forme

- HTML4/XHTML
`<form method="GET|POST">`
- HTML5
`<form method="GET|POST|PUT|DELETE">`

GET / POST

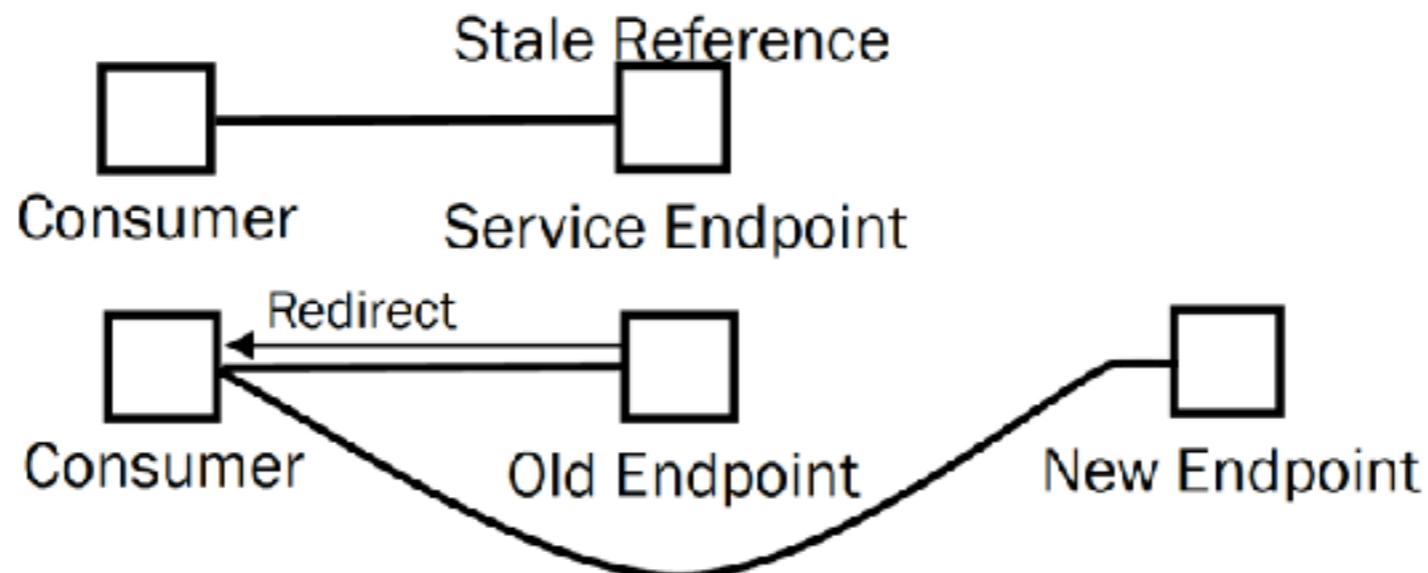
- GET je operacija čitanja - *read-only*. Može se ponavljati proizvoljan broj puta, pri čemu se stanje resursa neće menjati (idempotentna operacija). Može se i keširati.
 - Napomena: Ovo ne znači da će uvek biti vraćena ista *reprezentacija* resursa.
- POST je operacija i pisanja i čitanja, i može promeniti stanje resursa. Može izazvati i bočne efekte na serveru.

POST / PUT

- Koji je dobar način za kreiranje resursa?
 1. Klijent kreira id resursa.
Problem: Kako obezbediti stvarnu jedinstvenost?
PUT /resource/{id} //po definiciji može da se koristi i za inicijalizaciju kreiranih resursa
 2. Servis kreira id resursa, i vraća ceo kreirani resurs klijentu. Problem: moguće je kreiranje višestrukih instanci ako se više puta pozove endpoint.
POST /resource
 3. Server bi za novi resurs trebao da odgovori sa **201 Created** statusom.

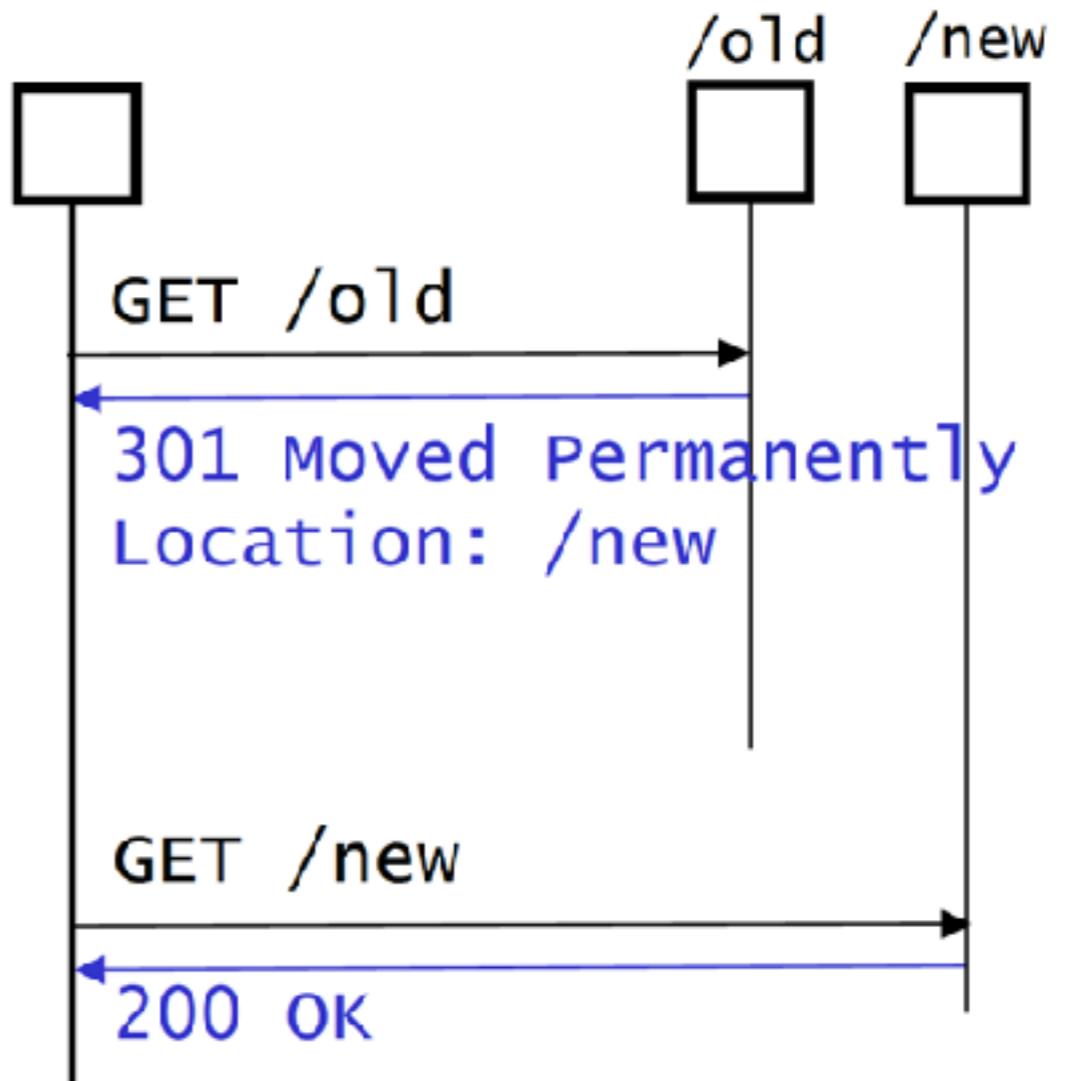
Kako se adaptirati na promene URL-ja

- Kako obezbediti da servis funkcioniše za krajnjeg korisnika čak i kada je stvarno došlo do promene endpoint(ova)?
 1. Do ove promene može doći kako iz poslovnih razloga tako i iz tehničkih
 2. Možda neće biti moguće odjednom promeniti sve linkove do servisnih endpointa, što dovodi do potencijalnih problema
- Korisititi automatsku redirekciju tako da sve korisnike koji se obrate starom endpointu redirektuje na novi endpoint



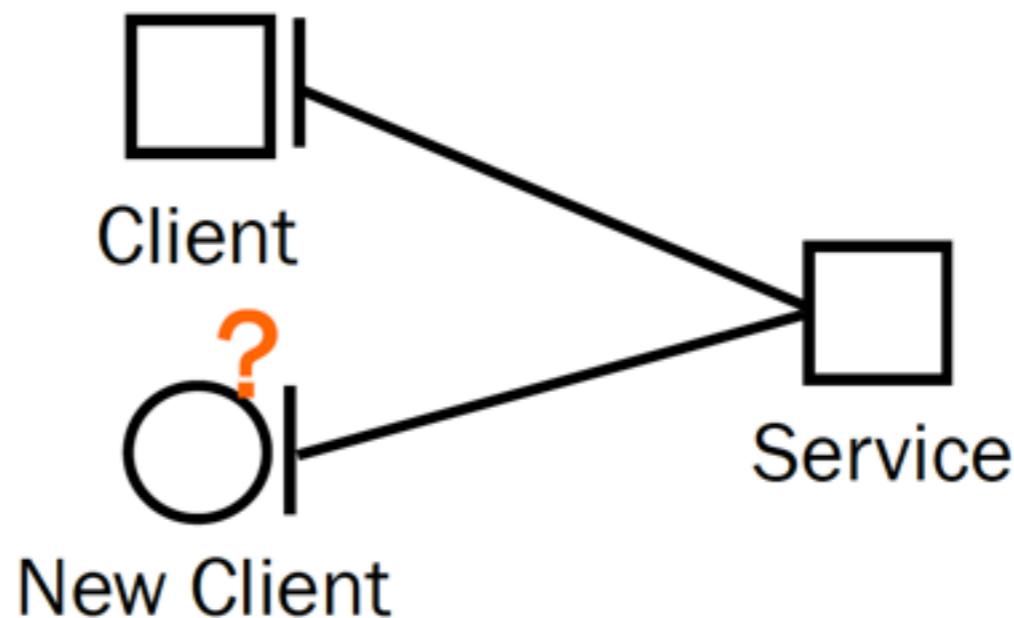
Redirekcija uz pomoć HTTP-a

- HTTP ima ugrađenu podršku za ovakve situacije
- Status kodovi 3xx
 - 301 Moved Permanently
 - 307 Moved Temporarily
 - Location: /newURI
- Redirekcije se mogu ulančavati, ali se pri tome mora paziti da se ne napravi cirkularna redirekcija



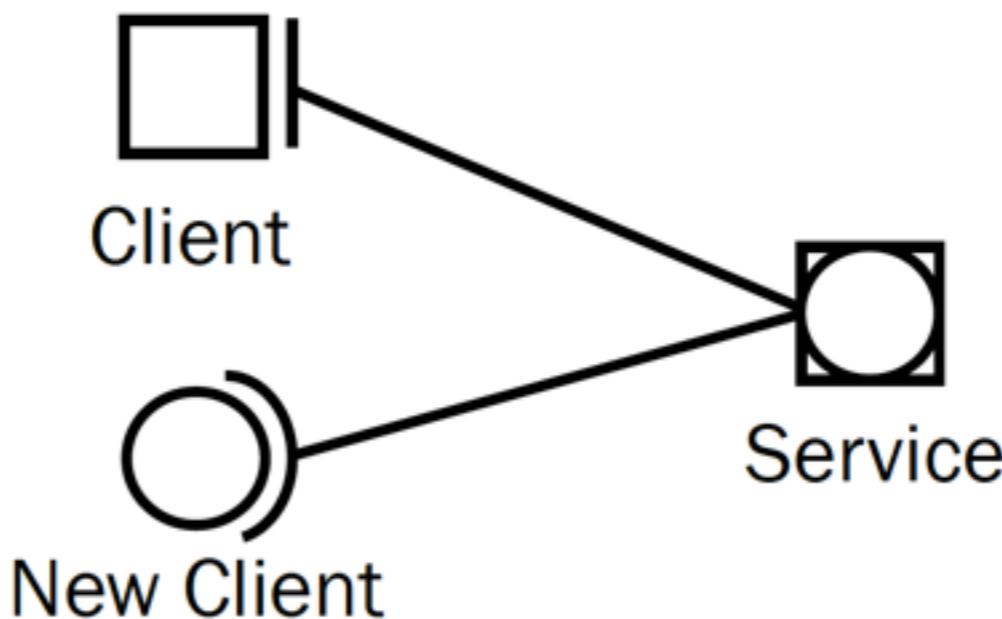
Da li se svi moraju dogovoriti o zajedničkom formatu

- Kako obezbediti da servis podržava korisnike koji koriste ili žele da koriste različite formate poruka (prenosa podataka)
- Ponekad klijenti mogu da promene svoje zahteve
- Servis bi možda trebalo da može da podrži različite korisnike bez potrebe da se pravi poseban interfejs za svakog od njih



Rešenje: *Content Negotiation*

- Specifičan sadržaj i formati reprezentacije podataka koji će biti prihvaćeni ili vraćeni od strane servisa može se dogovoriti u momentu izvršavanja servisa kao deo njegovog poziva.
- Dogovor o načinu korišćenja servisa se zasniva na tipovima sadržaja “*media types*”.
- Prednosti ovog pristupa: obezbeđuje slabu povezanost (uslovljenost), povećanu interoperabilnost, povećanu agilnost za adaptiranje



Content Negotiation u HTTP

- Dogovaranje o formatu razmene poruka je podržano u samom HTTP-u i ne zahteva slanje dodatnih poruka između klijenta i servera

GET /resource

Accept: text/html, application/xml, application/json

- Klijent saopštava koje formate može da razume (MIME types)

200 OK

Content-Type: application/json

- Server bira najpogodniji i njega vraća (status 406 ako nije u stanju da odgovori ni u jednom od traženih formata)

Content Negotiation u HTTP

- Dogovaranje o formatu razmene poruka je podržano u samom HTTP-u i ne zahteva slanje dodatnih poruka između klijenta i servera

GET /resource

Accept: text/html, application/xml, application/json

- Klijent saopštava koje formate može da razume (MIME types)

200 OK

Content-Type: application/json

- Server bira najpogodniji i njega vraća (status 406 ako nije u stanju da odgovori ni u jednom od traženih formata)

Content Negotiation - malo naprednije

- “Faktor kvaliteta” može da se koristi da sugeriše koliko koji format klijent preferira da dobije kao odgovor.

Media/Type; q=X

- Ako se za neki tip navede $q=0$, takav sadržaj nije prihvatljiv za klijenta.

Accept: text/html, text/*; q=0.1

- Klijent preferira HTML (ali prihvatiće i bilo koji drugi tekstualni format, ali sa nižim prioritetom)

Accept: application/xhtml+xml; q=0.9, text/html; q=0.5, text/plain; q=0.1

- Klijent preferira XHTML, ili HTML ako već ne može prvo, a kao *fallback* prihvata i običan tekst

Forsirani Content Negotiation

- Sam generički URI podržava ovaj koncept.

GET /resource

Accept: text/html, application/xml, application/json

- Napravi se poseban URI za svaki tip odgovora dodavanjem ekstenzije (postfixa) na URI

GET /resource.html

GET /resource.xml

GET /resource.json

- Napomena: Ovo je samo prihvaćena praksa nije standard.

Content Negotiation - mogu se podešavati razni aspekti

- Content Negotiation - veoma je fleksibilan i može se sprovesti po različitim aspektima (dimenzijama). Svaki od njih se podešava specifičnim parom HTTP zaglavlja

Request Header	Example Values	Response Header
Accept:	application/xml, application/json	Content-Type:
Accept-Language:	en, fr, de, es	Content-Language:
Accept-Charset:	iso-8859-5, unicode-1-1	charset parameter fo the Content-Typeheader
Accept-Encoding:	compress, gzip	Content-Encoding:

Definisanje medijskih tipova za REST

- Kako naći najbolji medijski tip za reprezentaciju podataka?
- Da li koristiti generičke ili smisliti novi specifični medijski tip?
- Da li uvek standardizovati tipove?

Neke preporuke za medijske tipove za REST

- Kad je moguće koristite postojeće dobro poznate tipove
- Ali kada je potrebno, slobodno kreirajte svoj
 - ali ga onda standardizujte i koristite kad god je moguće
- Medijski tipovi zapravo prenose informaciju o reprezentaciji resursa predstavljenih našim modelom
- Ne postoji “najbolji tip” za neki servis, sve zavisi od potreba i očekivanja klijenata
- Klijenti ne moraju nužno da procesiraju određeni medijski tip onako kako mi očekujemo

Obrada grešaka

Learn to use HTTP Standard Status Codes

100 Continue
200 OK
201 Created
202 Accepted
203 Non-Authoritative
204 No Content
205 Reset Content
206 Partial Content
300 Multiple choices
301 Moved Permanently
302 Found
303 See other
304 Not Modified
305 Use Proxy
307 Temporary Redirect

4xx Client's fault →

400 Bad Request
401 unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed

500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported

5xx Server's fault

RESTful Web servisi (2)

Vrste zahteva nad resursima

- Idempotentne
 - Sigurne
 - Nesigurne

Vrste zahteva nad resursima

- Idempotentni

- Mogu se ponavljati više puta, a da ne izazivaju negativne bočne efekte na serveru

GET /book

PUT /order/x

DELETE /order/y

- Ukoliko server ne funkcioniše ok, zahtev se može ponavljati dok server ne proradi
 - Sigurni su oni idempotentni zahtevi koji ne menjaju stanje samog resursa na serveru

GET /book

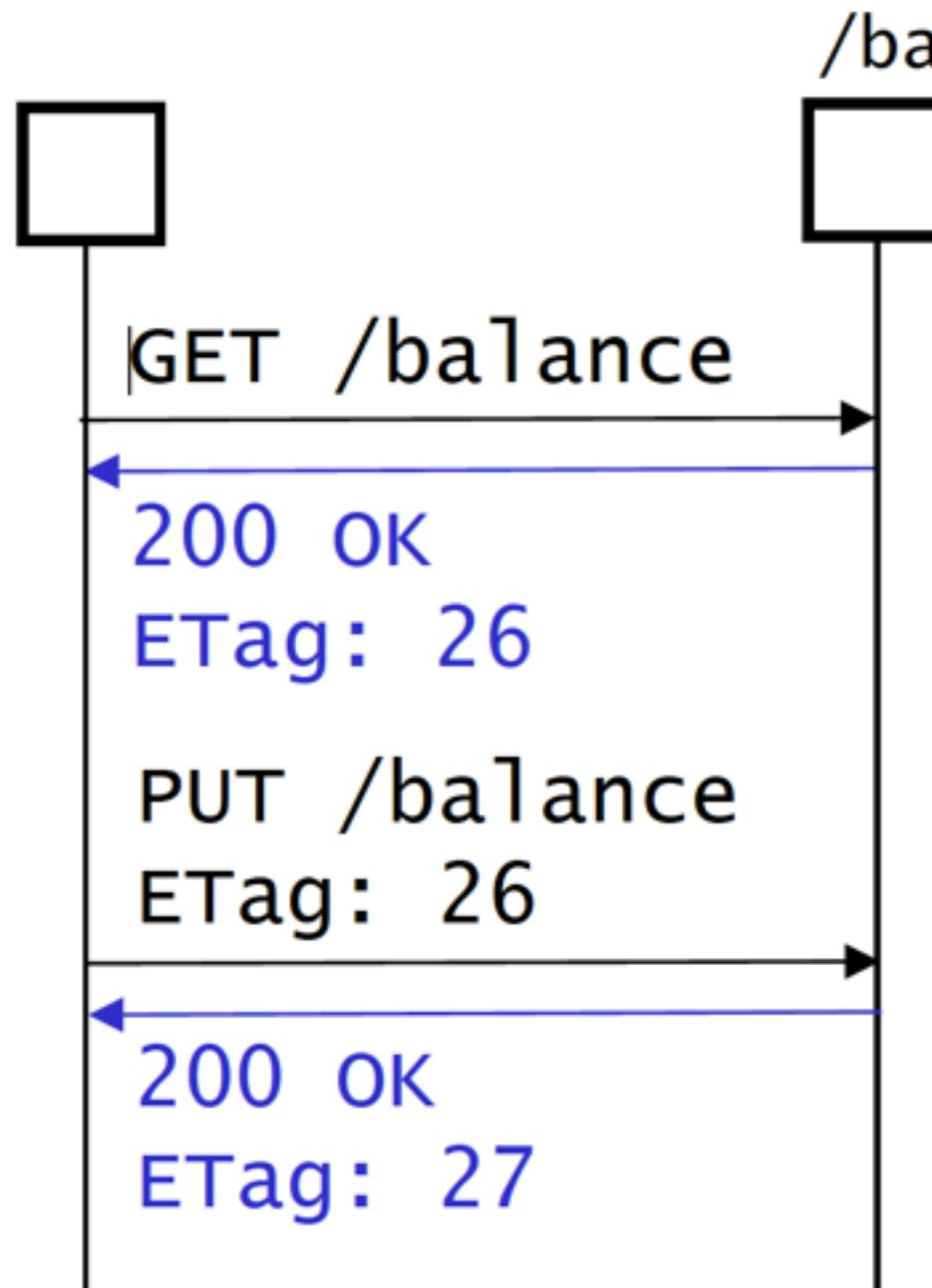
Vrste zahteva nad resursima

- Nesigurni zahtevi
 - ovi zahtevi modifikuju stanje na serveru i ne mogu se ponavljati, a da pri tome ne izazovu neželjene efekte
 - za ovakve zahteve neophodne su dodatne akcije u posebnim situacijama (tzv. state reconciliation)

POST /order/x/payment

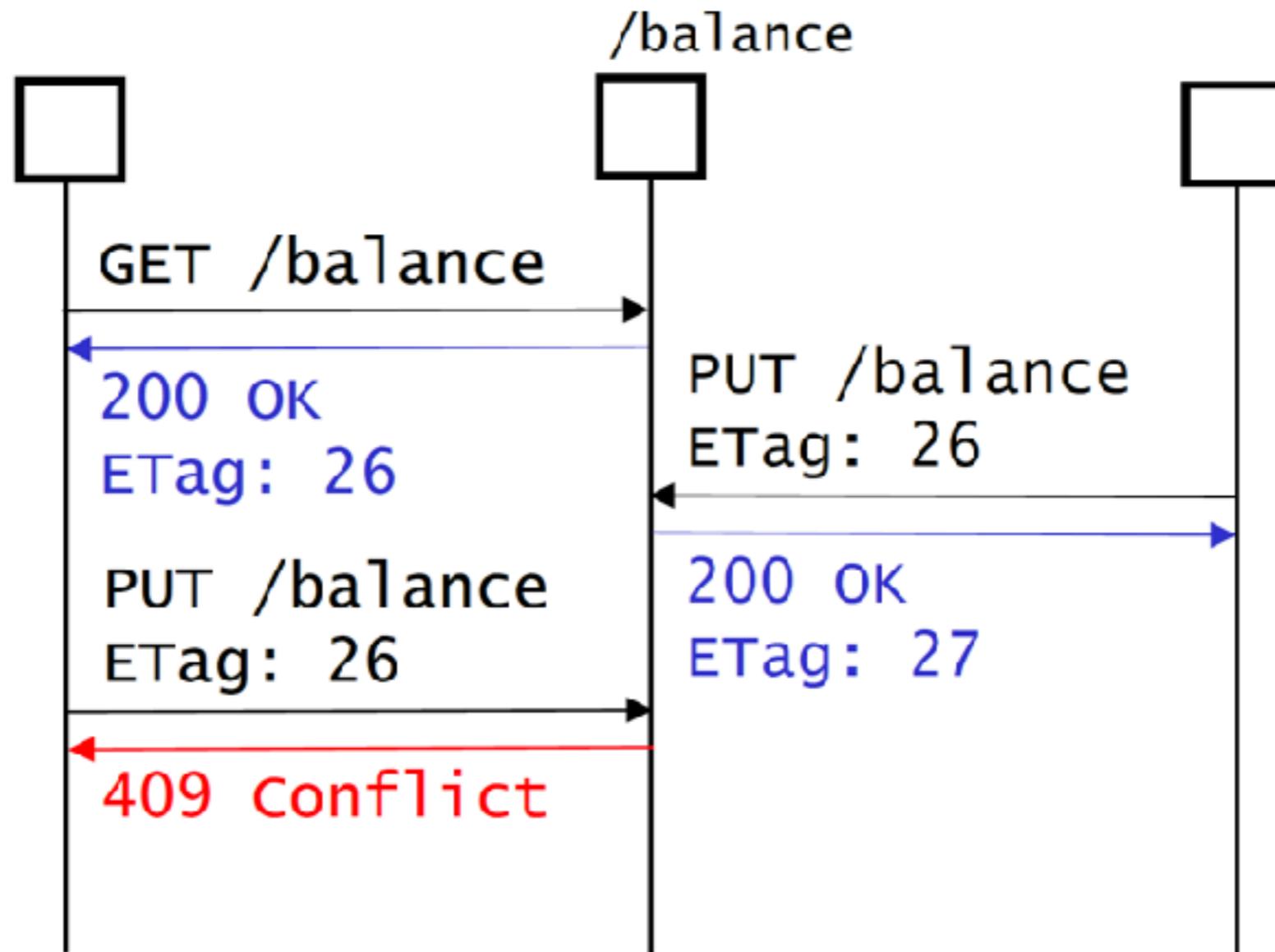
- u nekim situacijama API se može refaktorisati tako da se koriste idempotentne operacije

Problem konkurentnosti



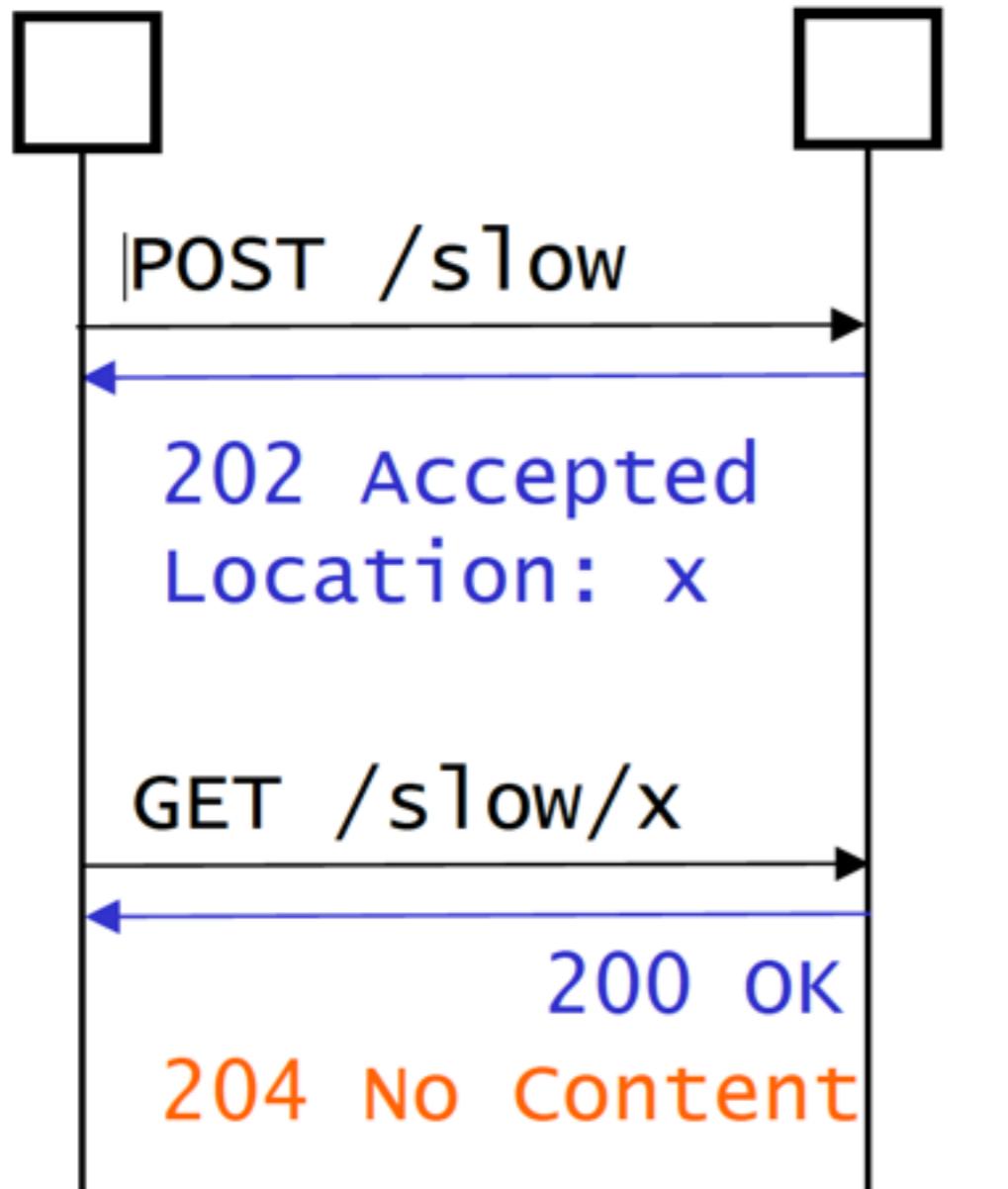
- Razlaganje APIja u niz idempotentnih zahteva pomaže da se prevaziđu problemi privremenih otkaza servisa
- Šta ako drugi korisnik konkurentno ažurira resurs?
- Da li raditi pesimističko zaključavanje resursa (eksplicitno zaključavanje) ili je moguće i neko drugo rešenje?

Problem konkurentnosti



- Status 409 može se iskoristiti da se korisniku prenese informacija da bi izvršenje zahteva dovelo do nekonzistentnosti resursa

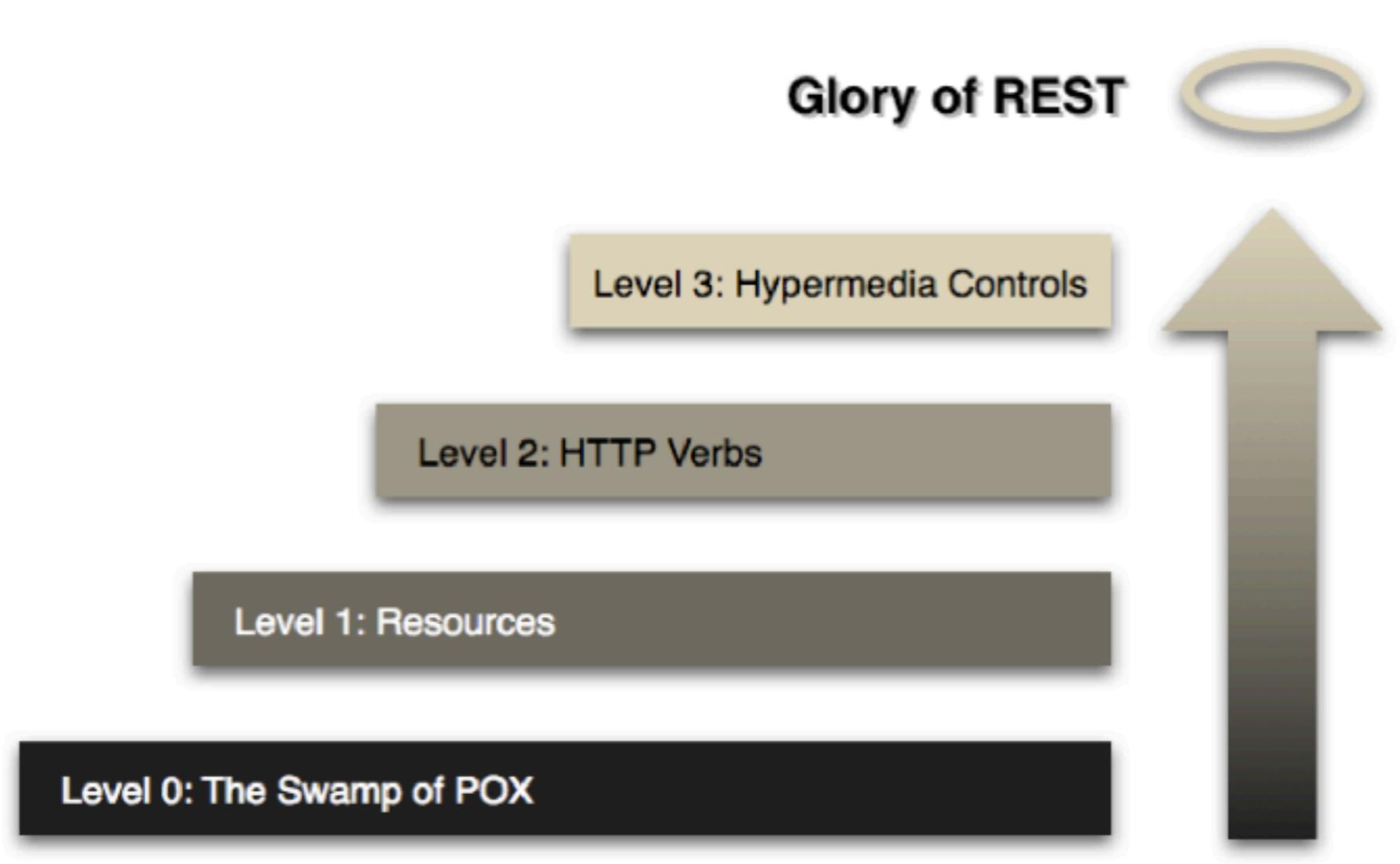
Blokirajući ili neblokirajući zahtevi?



- HTTP je u svojoj suštini sinhron, ali ipak ne mora biti blokirajući
- za zahteve za koje znamo da mogu dugo da traju može se vratiti kod 202, sa informacijom gde naknadno preuzeti rezultat
- koliko često klijent treba da "proba" čitanje rezultata
 - odgovor sa /slow/x bi mogao da sadrži i estimaciju trajanja obrade

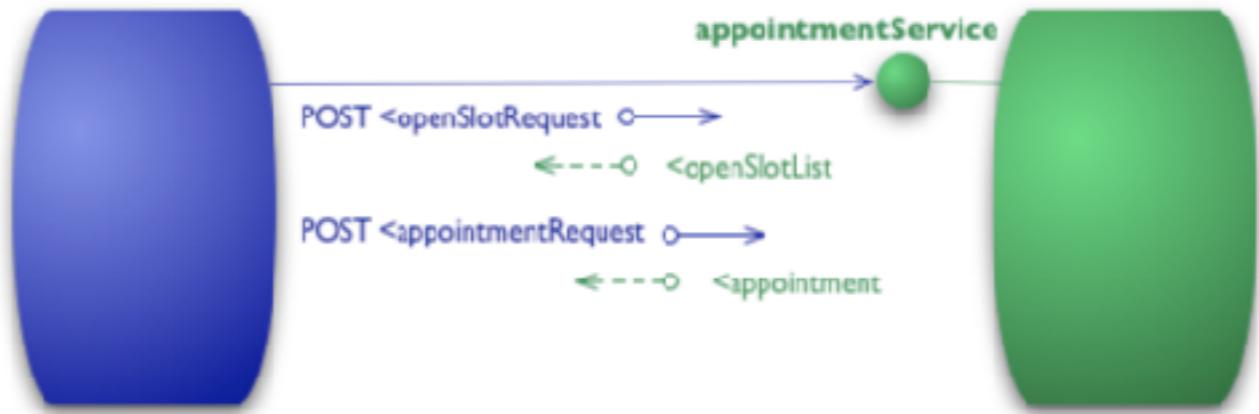
Richardsonov model zrelosti REST servisa

- Model koji je definisao Leonard Richardson koji promenu osnovnih principa REST-a razbija na nekoliko koraka.

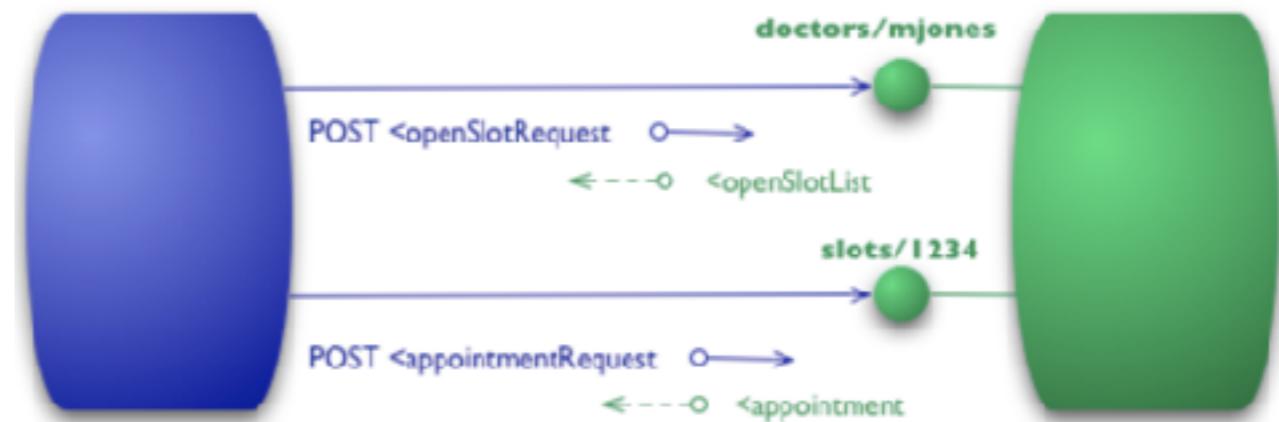


Richardsonov model zrelosti REST servisa

- Level 0: HTTP se koristi kao RPC Protocol (Tuneluje se POST+POX ili POST+JSON). Suštinski se koristi HTTP za udaljeni pristup ali bez korišćenja bilo kojih mehanizama weba. Obično se sve operacije obavljaju preko jedne adrese.

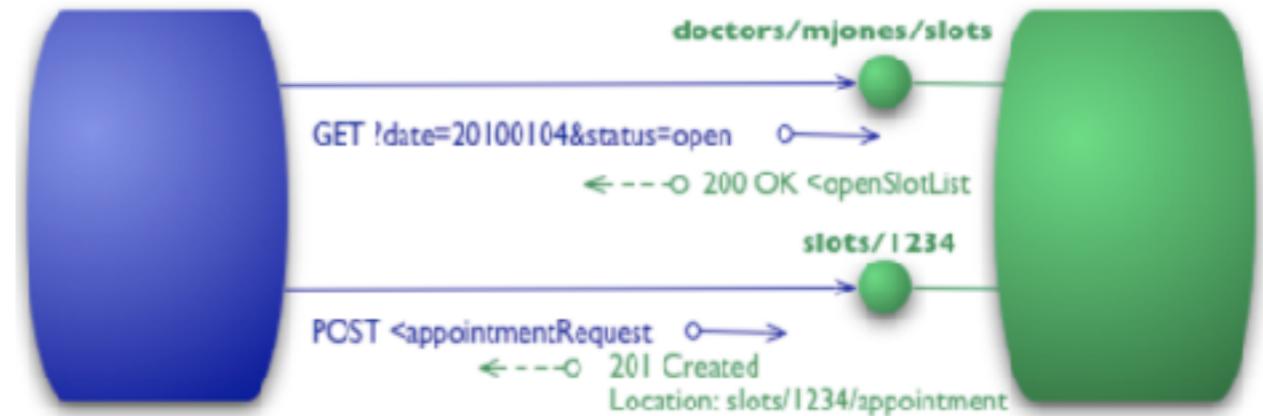


- Level 1: Identifikacija resursa - uvođenje više URI-ja za različite resurse (opšta adresabilnost resursa).

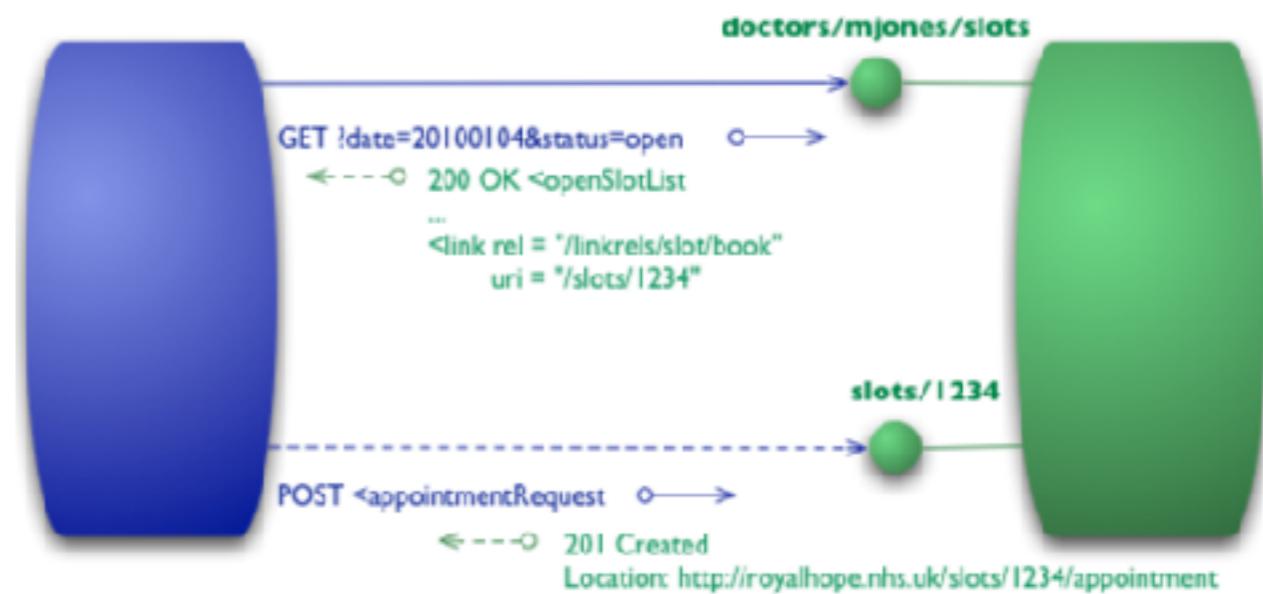


Richardsonov model zrelosti REST servisa

- Level 2: Uniformno korišćenje HTTP metoda (*Contract Standardization*). Na ovom nivou uvodi se konzistenta upotreba HTTP metoda - pojednostavljuje se razumevanje API-ja i bez previše pojašnjenja očekuje se određeno ponašanje. Obratiti pažnju i na povratne statusne kodove.



- Level 3: Hypermedia Controls / HATEOAS (Hypertext As The Engine Of Application State) Na ovom nivou i same osobine servisa i mogućnosti koje nudi klijentu mogu da se odrede pristupom servisu



Level 0, *antipattern (antišablon)* za REST

HTTP samo za tunelovanje zahteva

- Tunelovanje kroz 1 HTTP metod

GET /api?method=addCustomer&name=Wilde

GET /api?method=deleteCustomer&id=42

GET /api?method=getCustomerName&id=42

GET /api?method=findCustomers&name=Wilde*

- Sve se radi preko GET zahteva
 - Prednost: lako se testira iz browsera
 - Mana: GET bi trebalo sa se koristi samo za čitanje
 - Ograničenje: max. količina podataka koju je moguće ovako slati je oko 4KB

Antišablon za REST

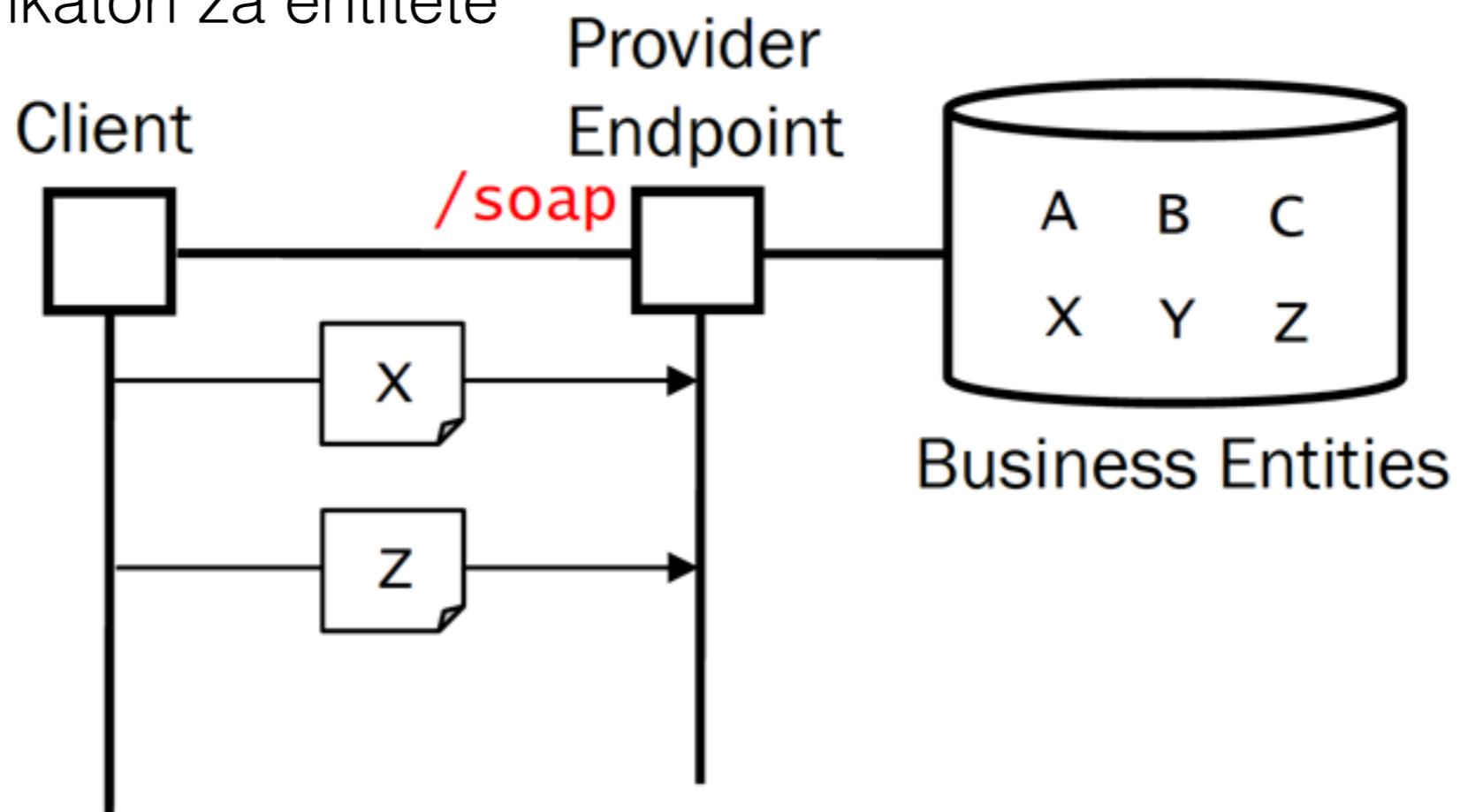
HTTP samo za tunelovanje zahteva

- Tunelovanje kroz 1 HTTP metod
- Sve se radi preko POST zahteva
 - Prednost: moguće je preneti veće količine podataka, pa i uploadovati fajlove
 - Mana: POST zahtev se ne smatra idempotentnim niti sigurnim (pa se stoga ne kešira, a i koristi se za potencijalno “nesigurne operacije”)

Antišablon za REST

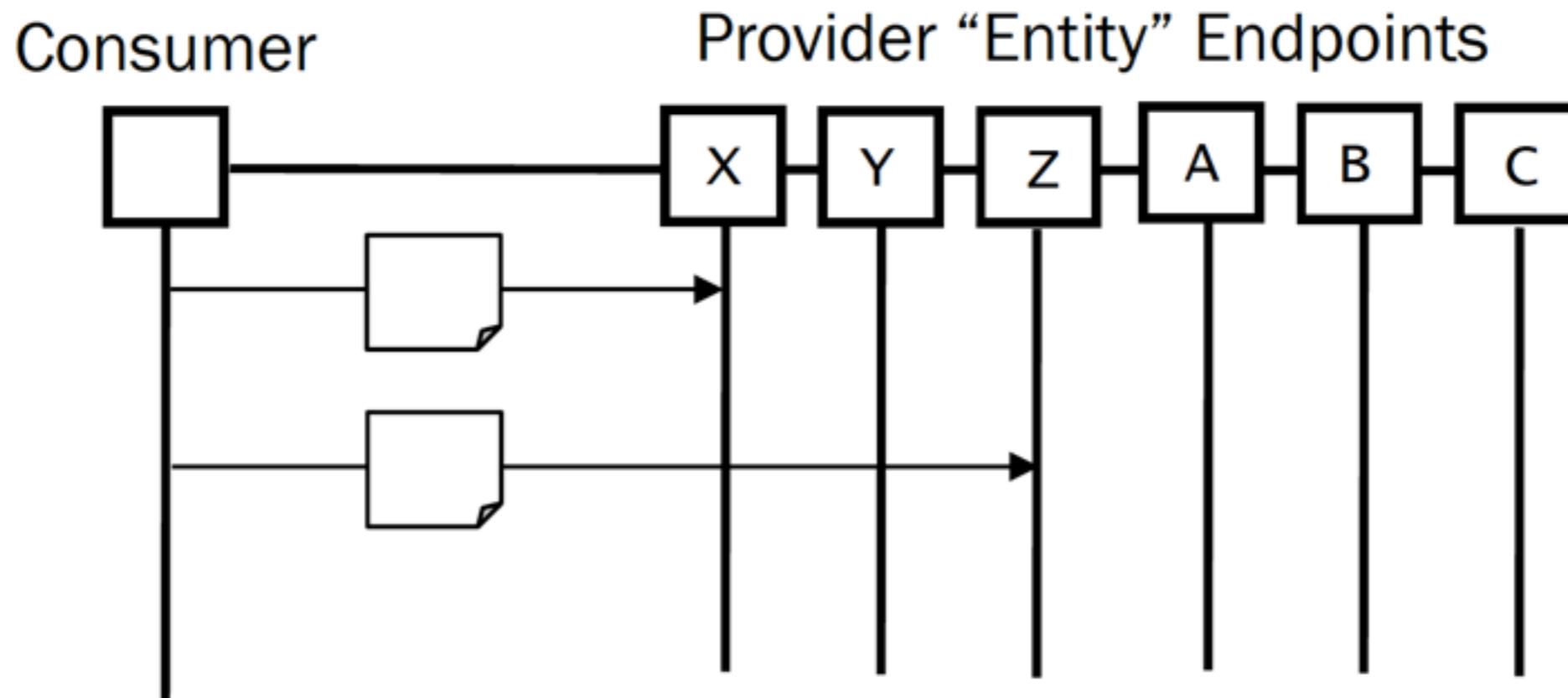
HTTP samo za tunelovanje zahteva

- Tunelovanje kroz 1 endpoint
- Servis sa samo jednim *endpoint-om* nije dovoljno granularan u smislu pristupa pojedinim entitetima-resursima. Klijent tada mora da radi sa najmanje dva identifikatora, jedan identifikator za sam servis, identifikatori za entitete



Opšta adresabilnost

- Rešenje za prethodni problem - svaki resurs postaje dostupan preko svog individualnog *endpoint-a*



Antišablon - Cookies

- Da li su Cookies RESTful?
 - Zavisi od načina primene - REST se zasniva na *stateless* komunikaciji
- 1. “kolačići” mogu i sami biti “zatvoreni”/samoopisujući - tako da sadrže sve informacije koje su neophodne za njihovu interpretaciju pri svakom zahtevu/odgovoru
- 2. “kolačići” sadrže samo reference na stanje aplikacije (koje se samo ne održava kao resurs)
 - prenose samo tzv. ključ sesije
 - Prednost: manje podataka se prenosi
 - Mane: Zahtevi više nisu “zatvoreni” pošto nose informaciju o nekom kontekstu koji bi server trebao da pamti.
Takođe neophodno je imati mehanizam koji će počistiti nekorišćene reference (istekle sesije)

Stateless vs. Stateful?

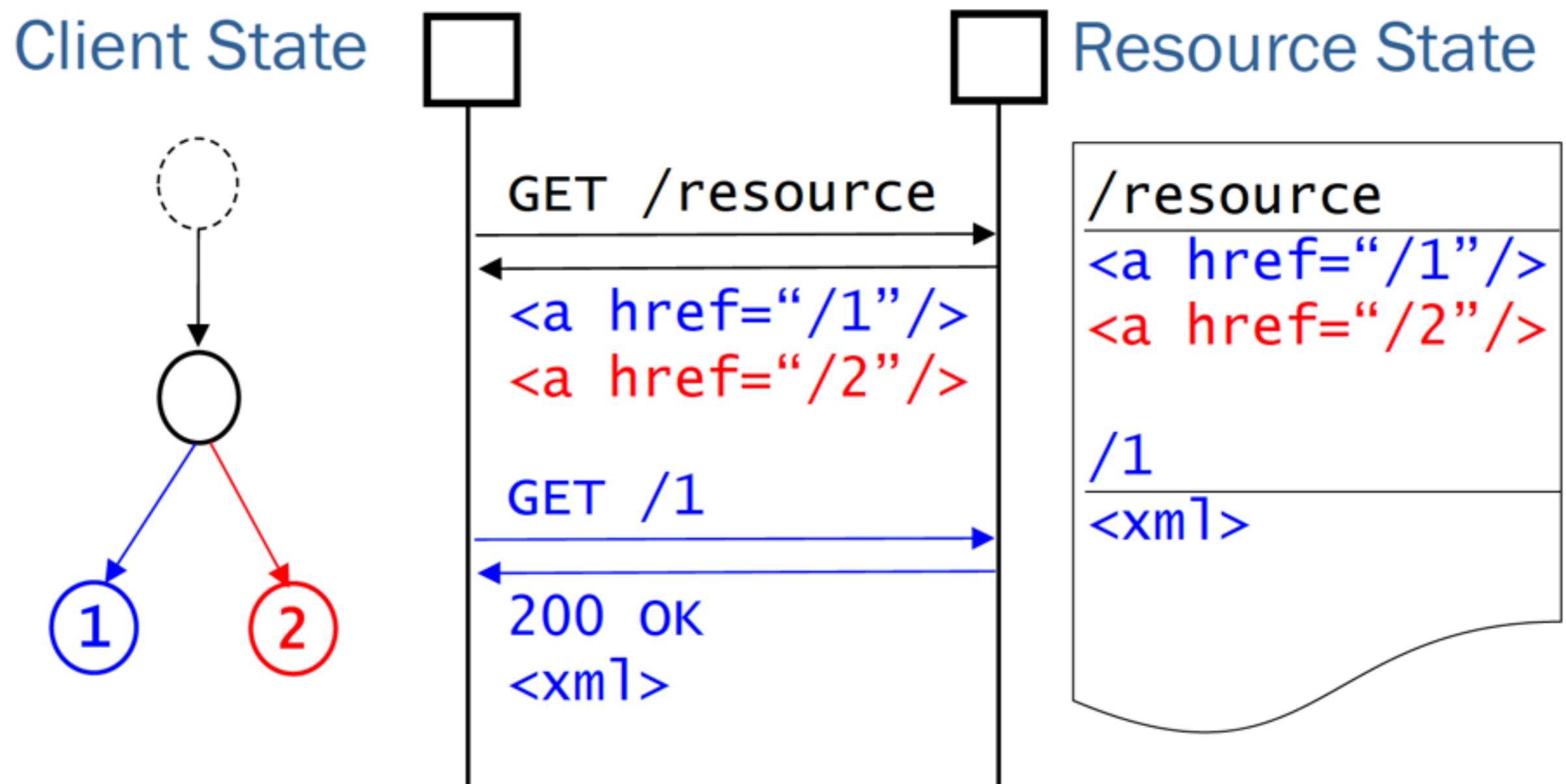
Stanje aplikacije

- Samo ime REST-a sugeriše da se posmatra stanje resursa u distribuiranom sistemu, ali komunikacija je stateless.
- Stanje na klijentu:
 - Klijent se kreće po raspoloživim resursima tako što prati linkove, njegovo stanje je određeno posećenim linkovima
 - Server može da utiče na tranzicije stanja koje su klijentu raspoložive tako što mu u odgovoru na GET zahteve može slati hiperlinkove do resursa koje treba pratiti

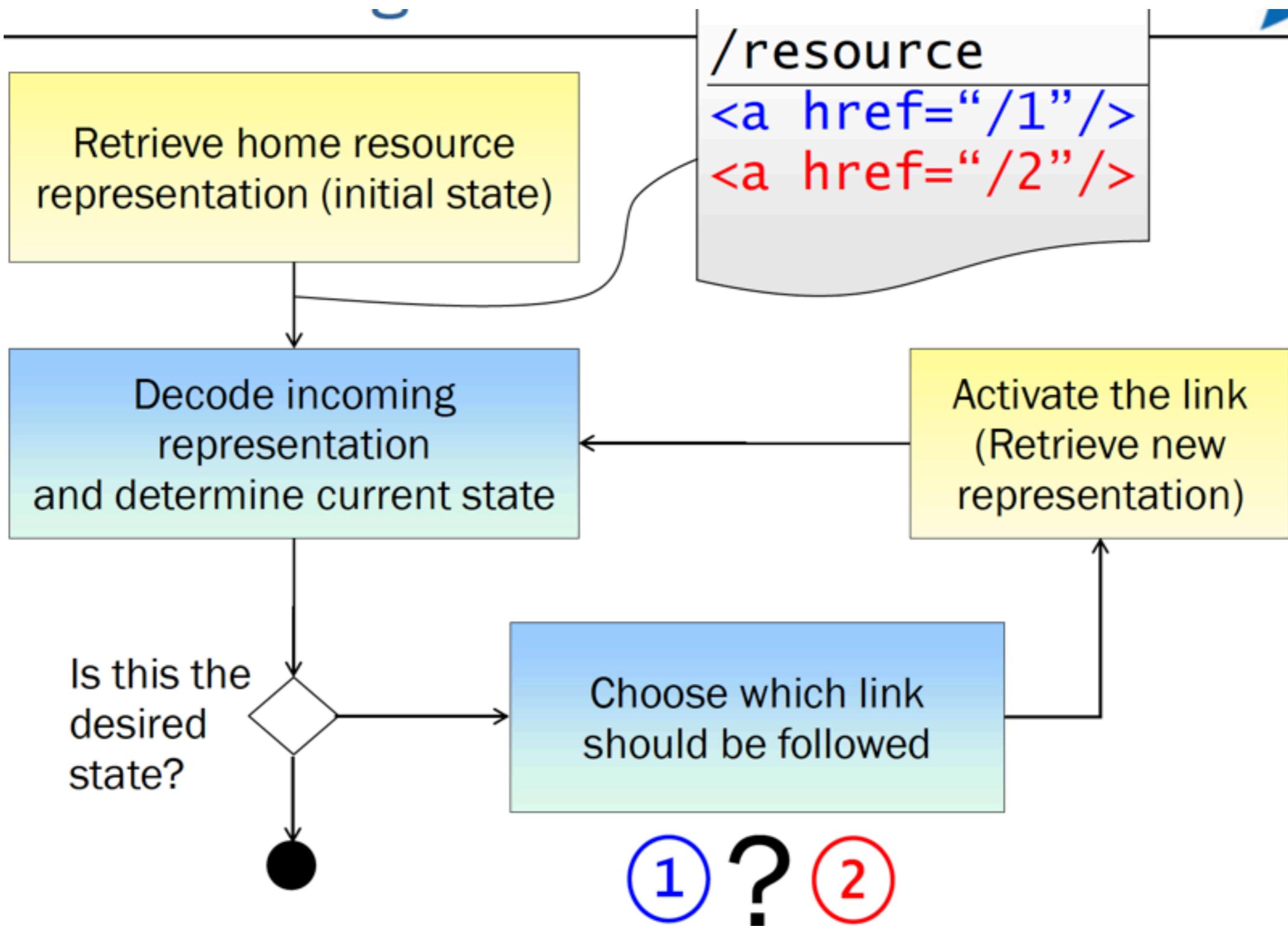
Stanje aplikacije

- Stanje na serveru:
 - Stanje resursa definiše perzistetno trenutno stanje aplikacije
 - Klijenti mogu pristupiti resursima koristeći različite reprezentacije
 - Klijent manipuliše stanjem resursa na serveru preko uniformisanog interfejsa koji obavlja CRUD operacije

Stanje aplikacije

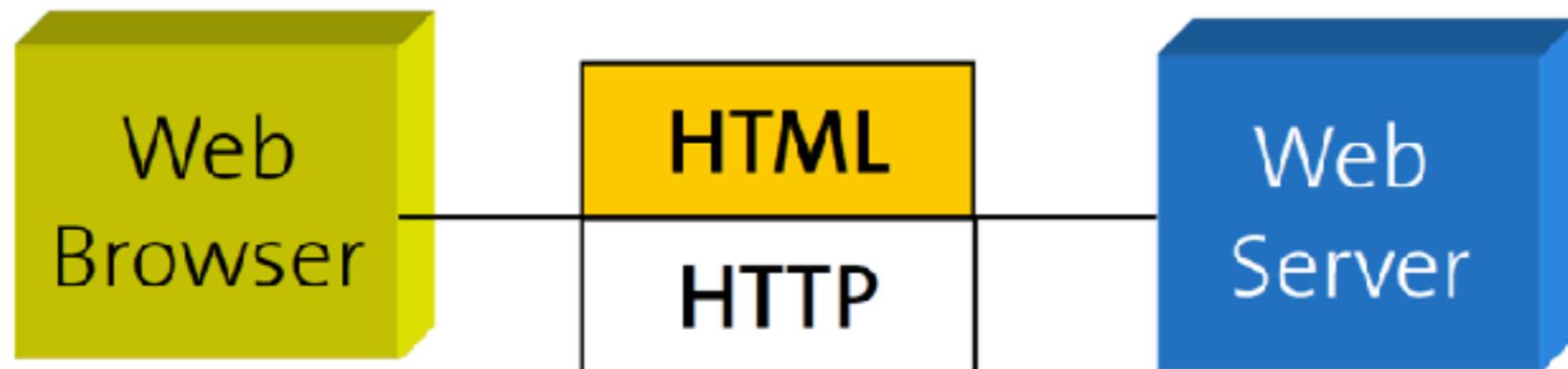


Algoritam rada klijenta

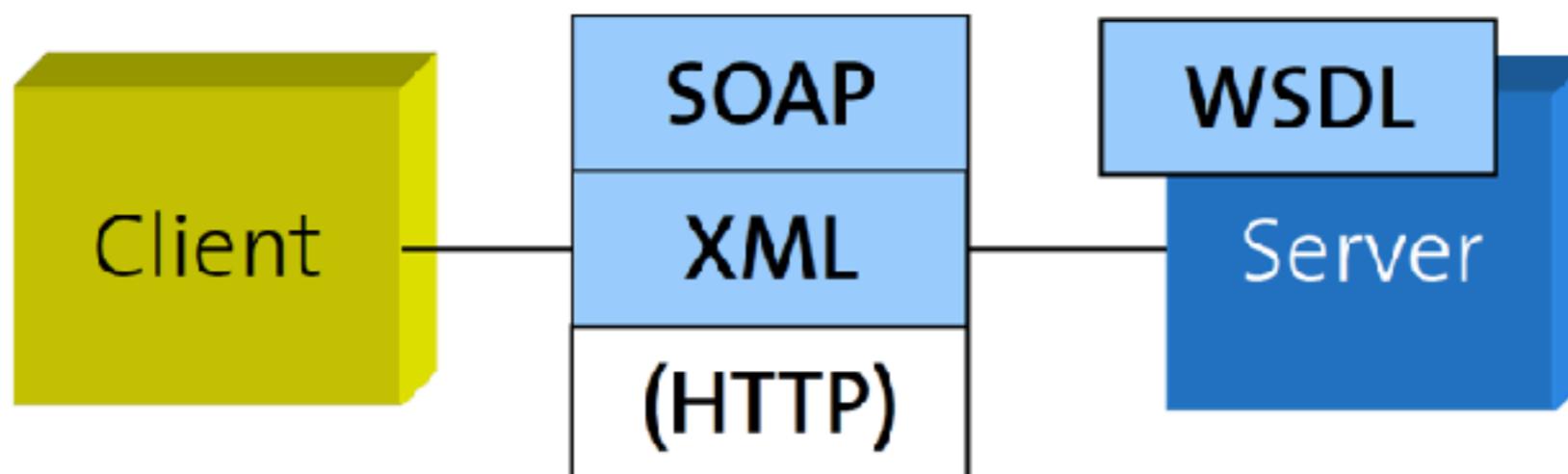


REST vs. *WS-**

Web Sites (1992)

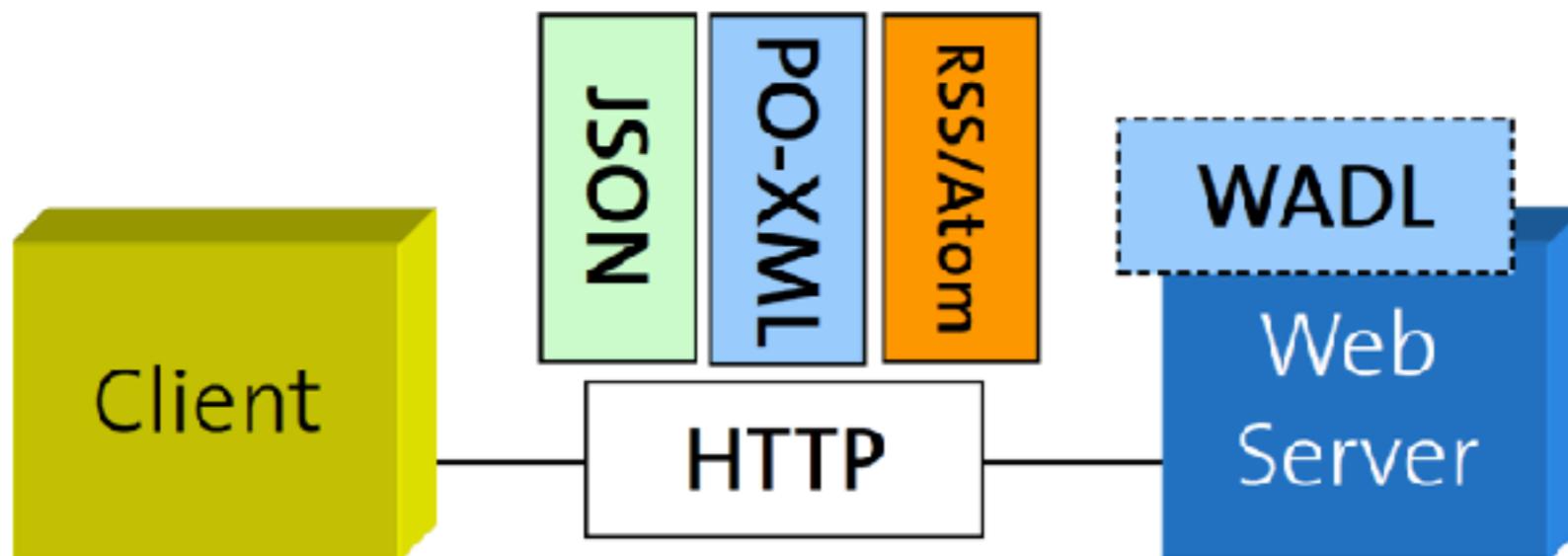


WS-* Web Services (2000)

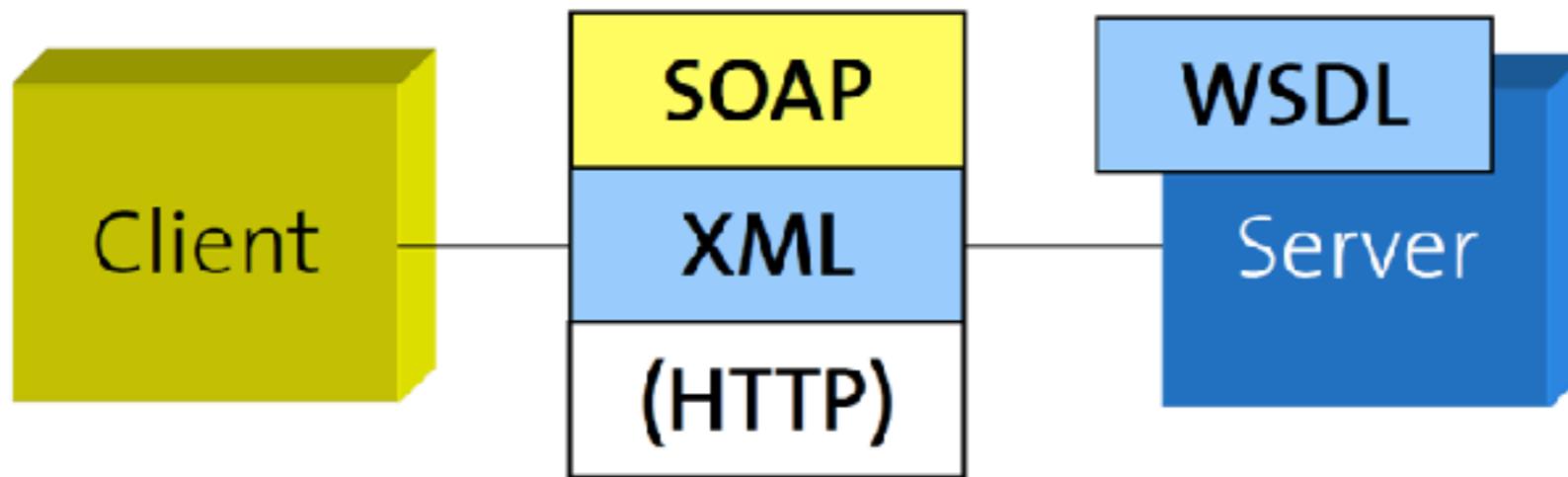


REST vs. *WS-**

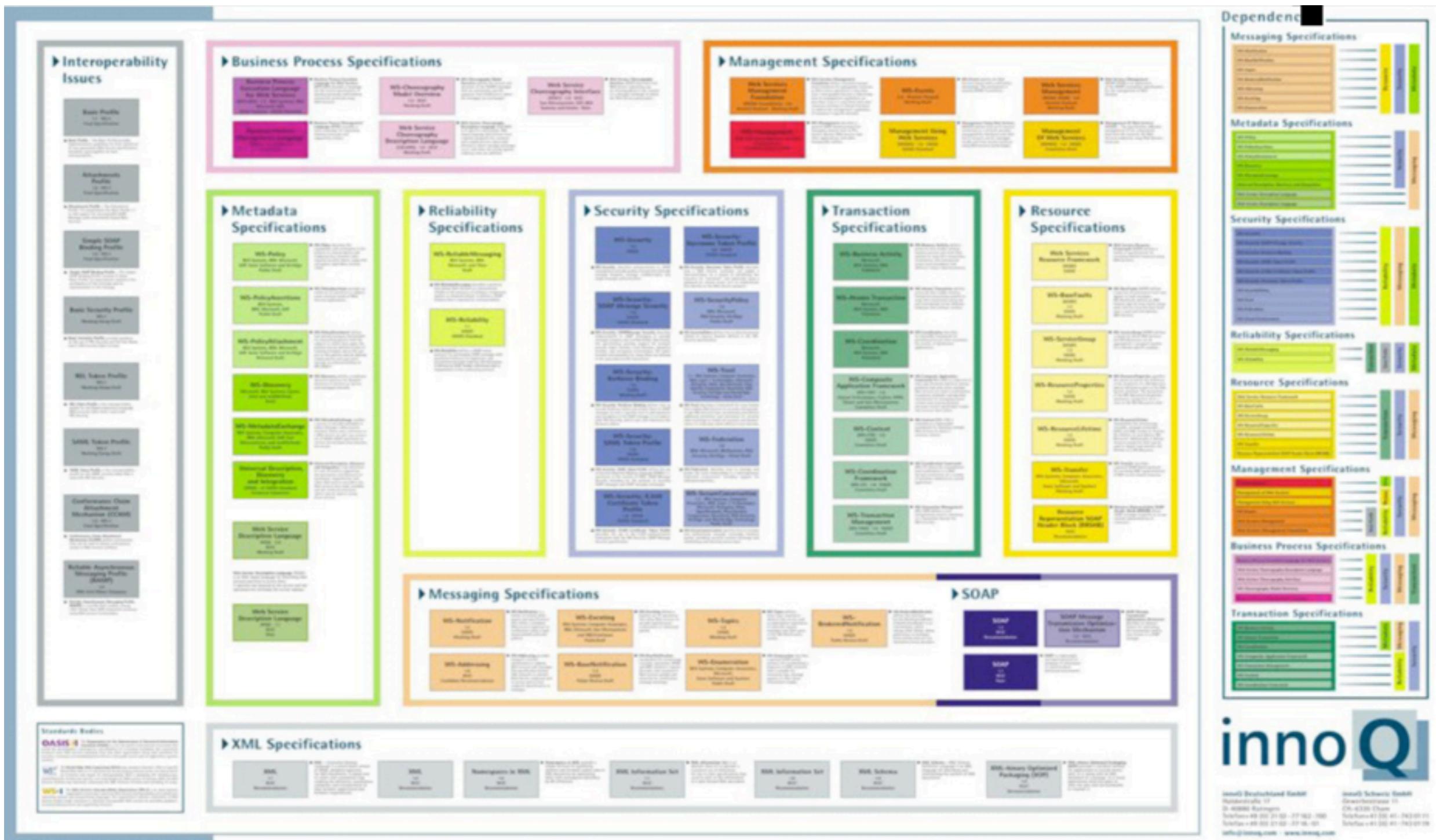
RESTful Web Services (2007)



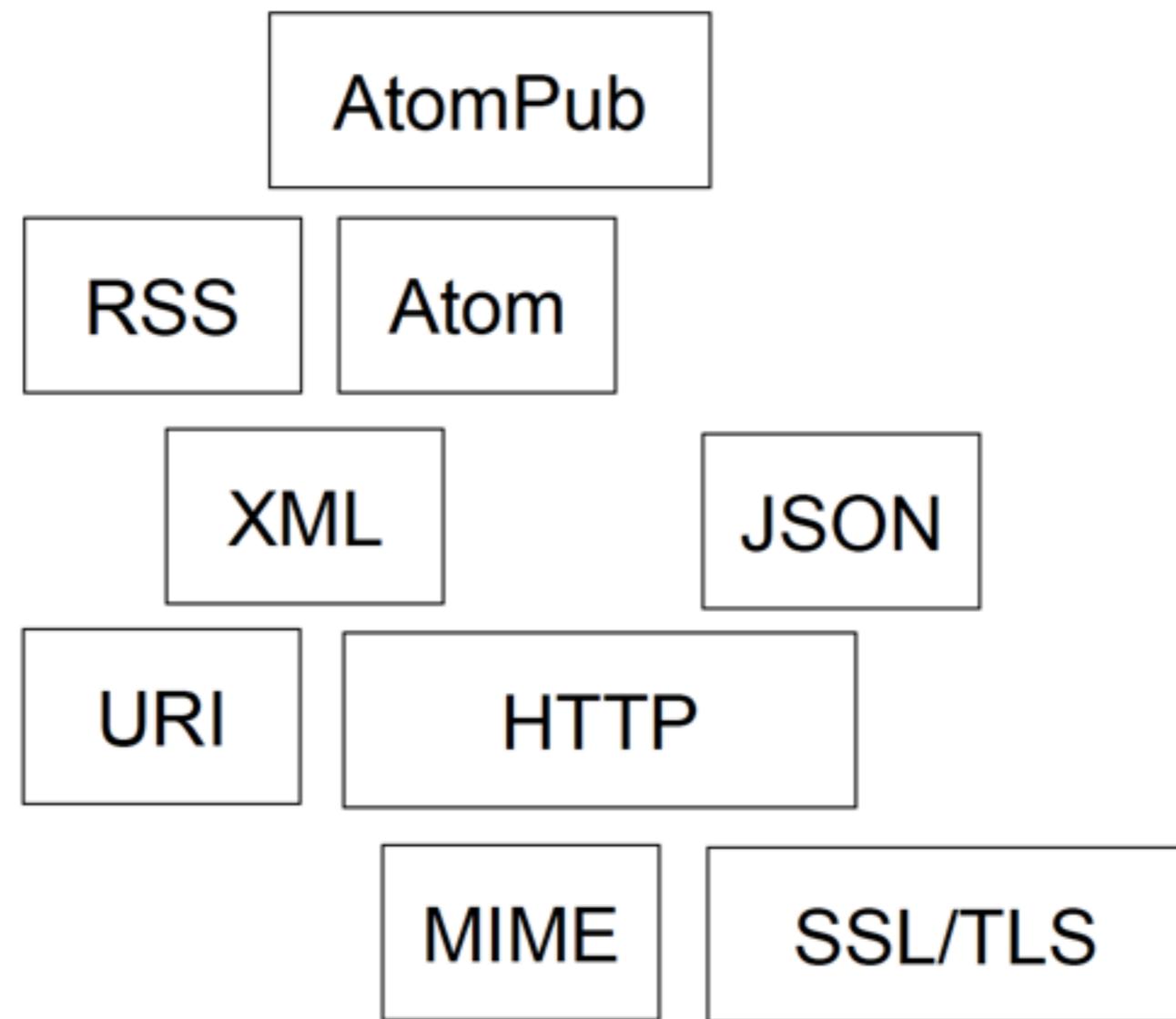
WS-* Web Services (2000)



WS- standardni stack tehnologija*



REST standardni stack tehnologija



Mikroservisi

Šta su mikroservisi?

- Još jedan stil arhitekture softverskih sistema, u kome se velike složene aplikacije komponuju sklapanjem pojedinačnih servisa.
 - Koncept nije potpuna novina - **predstavlja samo još jedan pristup implementaciji SOA**
- Mikroservisi mogu biti nezavisno *deployovani* i međusobno su slabo spregnuti.
- Kod mikroservisnih arhitektura pojedinačni servisi obavljaju jedan zadatak
 - Taj jedan zadatak predstavlja jednu poslovnu funkciju celokupnog sistema

Zašto novi stil arhitekture?

- Promenilo se okruženje u kojem se aplikacije izvšavaju
- Promene u poslednjih 15-ak godina:

Yesterday	Today
Single machines	Clusters of machines
Single core processors	Multicore processors
Expensive RAM	Cheap RAM
Expensive disk	Cheap disk
Slow networks	Fast networks
Few concurrent users	Lots of concurrent users
Small data sets	Large data sets
Latency in seconds	Latency in milliseconds

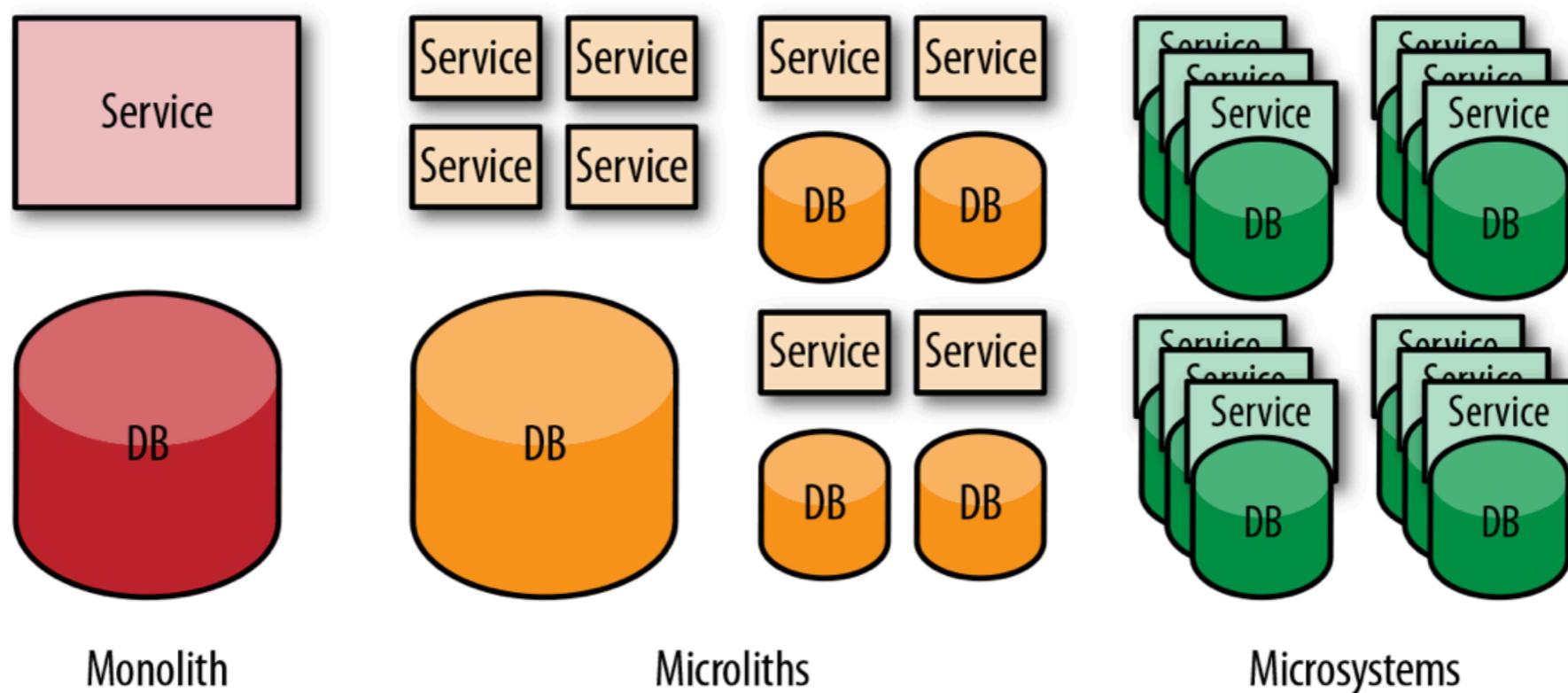
Zašto još jedan stil arhitekture?

- Dosadašnje arhitekture su dobro zadovoljavale potrebe, čak i velikih aplikacija, ali ne zadovoljavaju u uslovima visokodistribuiranih sistema.
- Današnje aplikacije su često miks aplikacija/modula koji se izvršavaju na raznim platformama - mobilnim uređajima, desktop mašinama, velikim serverima, a broj korisnika je često jako veliki i često vrlo promenljiv.
- Da bi se odgovorilo na potrebe novih aplikacija neophodno je “napusititi monolitnu arhitekturu i dekomponovati sistem u upravljive diskrete servise koji mogu samostalno da se skaliraju, i koji mogu u potpunoj izolaciji da otkazuju, da se puštaju u produkciju i da se ažuriraju” (dakle nezavisno od svih ostalih)¹

¹ prevod citata iz: Jonas Bonér. “Reactive Microsystems”

Zašto mikroservisna arhitektura?

- Evolucija sistema tokom vremena:



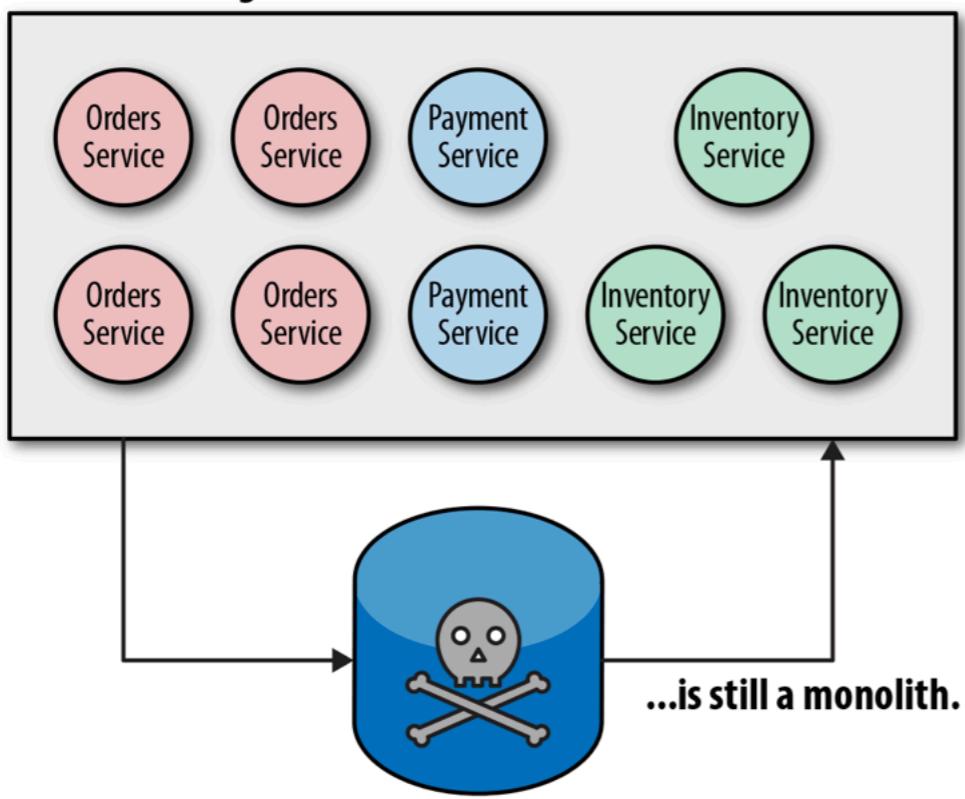
Koje su ključne osobine mikroservisa

- **Izolacija** - svaki mikroservisni modul trebao bi biti potpuno izolovan, razvijati se samostalno, čuvati svoje podatke samostalno i isporučivati se samostalno.
- **Autonomija** - svaki mikroservisni modul bi trebalo da je u mogućnosti da obavi svoju funkcionalnost autonomno - sve odluke o tome šta se radi, i kako se radi (kompletna logika) mora biti u samom mikroservisu za datu funkcionalnost (izolacija je preduslov za ovo). Autonomni servisi međusobno komuniciraju da bi obavili zadatke koji su složeniji od njihove funkcionalnosti.
- **Odgovornost za jednu stvar (*single responsibility*)** - mikroservis bi trebao da je napisan tako da obavlja jednu stvar i da je obavlja dobro - obavlja malu, jasno definisanu funkcionalnost koju je moguće uklopiti sa funkcionalnošću i drugih modula kada se formira složeni sistem.
Ovaj princip direktno određuje šta je to tačno "mikro" (nije bitan broj linija koda, već jedna jasna i zaokrežena funkcija).

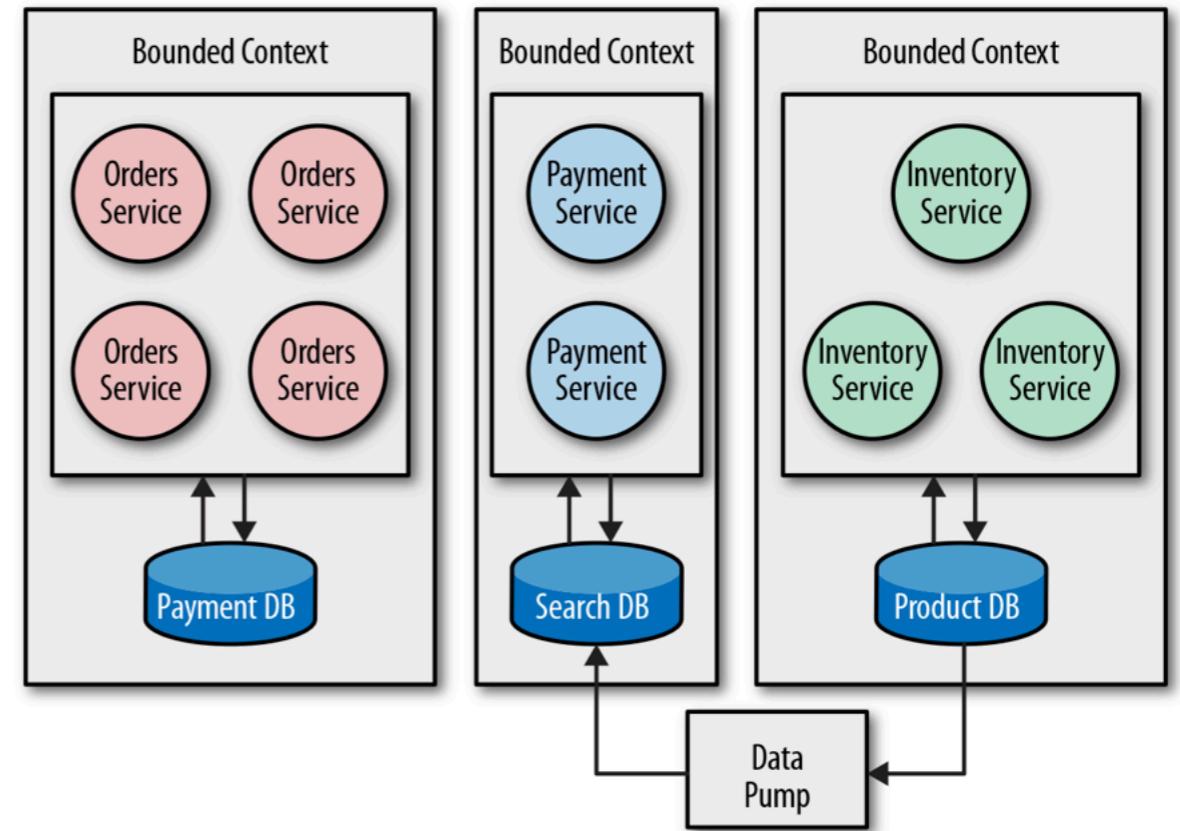
Koje su ključne osobine mikroservisa

- **Ekskluzivnost čuvanja sopstvenog stanja** - svaki mikroservisni modul je jedini vlasnik podataka nad kojima operiše. Ovo je činjenica koja ima ogroman efekat na ceo sistem - svi podaci su potpuno konzistentni samo unutar granica jednog mikroservisa, ali se to nikada ne prepostavlja za podatke izvan ovog ograničenog konteksta (za njih se prepostavlja *eventual consistency* ili *causal consistency*).

A monolith disguised as a set of microservices...



A microservice owns its data!



Koje su ključne osobine mikroservisa

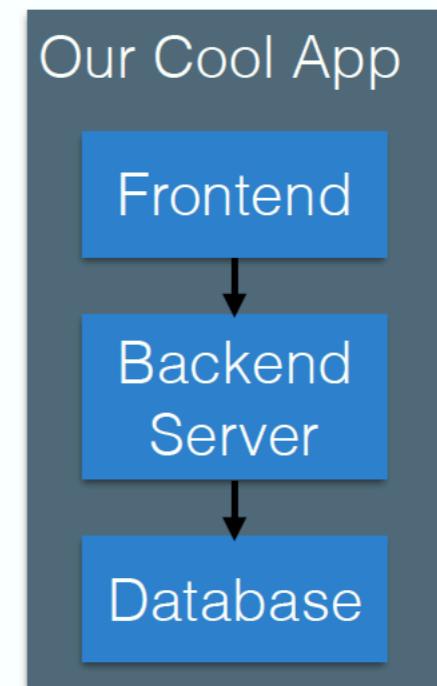
- **mobilnost** - mobilnost servisa podrazumeva mogućnost sistema da bude elastičan i dinamički adaptabilan - mikroservisni moduli mogu da se pokrenu na različitim serverima u različitim momentima, da postoji 1 ili više instanci istog mikroservisa u bilo kom momentu.
 - virtualizacija, kontejnerizacija su odlični alati koji ovo omogućavaju, ali ništa od svega toga ne vredi ako su nam servisi adresirani fiksno.
 - mora na nivou upravljanja celim sistemom postojati mehanizam da se mikroservisu može pristupiti, a da se nigde ne *hardcodira* njegova adresa

Osnovne karakteristike?

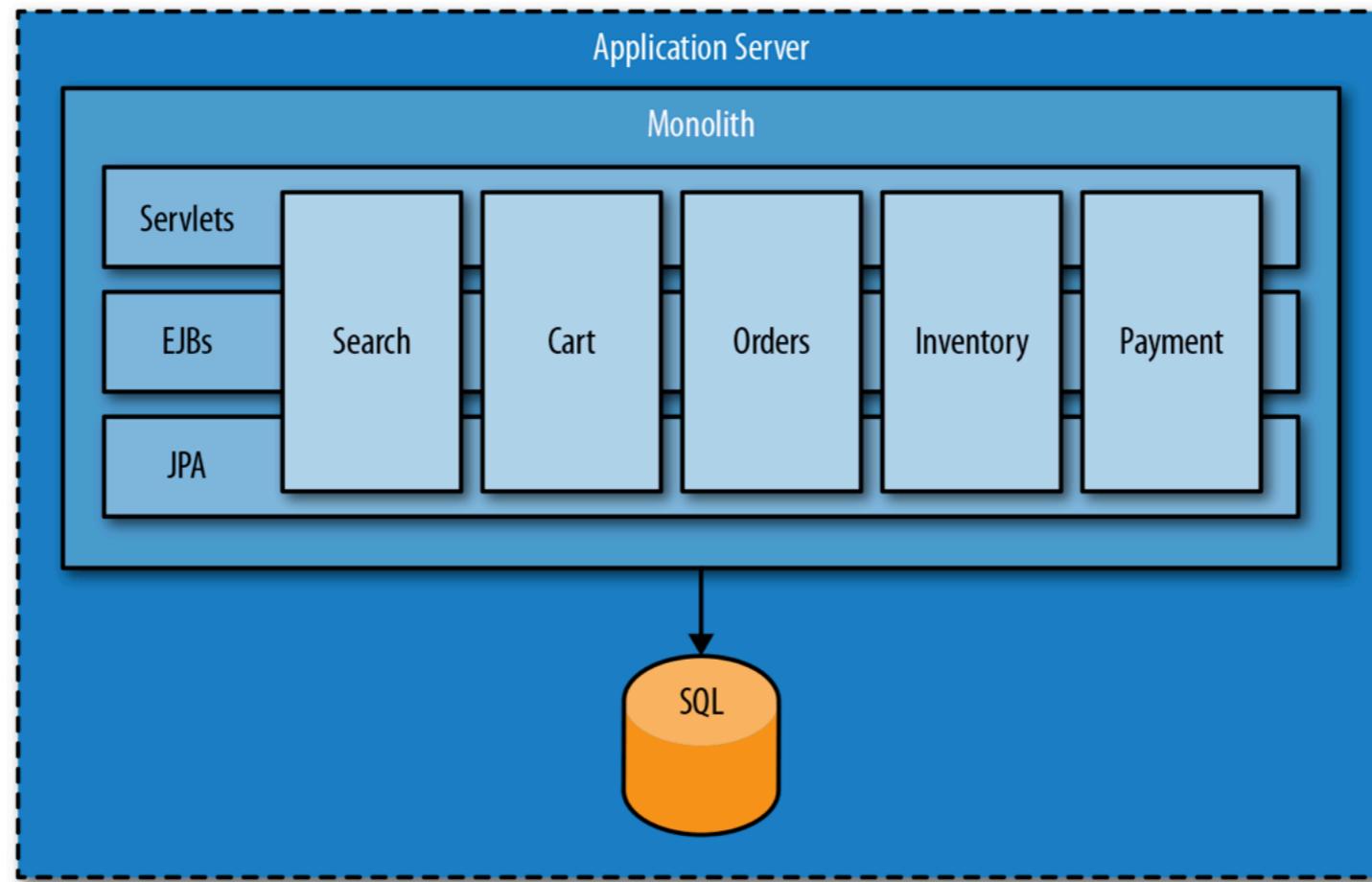
- Svaki mikroservis moguće je razvijati u programskom jeziku koji je najpogodniji, nezavisno od svih ostalih.
- Komunikacija između mikroservisa se obavlja programskim interfejsima (API) koji su nezavisni od programske jezike (npr. Representational State Transfer (REST)).
- Mikroservisi (moduli koji ih realizuju) imaju potpuno ograničen kontekst (*bounded context*) - ne moraju biti svesni nikakvih implementacionih detalja i arhitekture drugih mikroservisnih modula.

Monolitni vs. mikroservisni sistemi

- Monolitne nasuprot mikroservisnim arhitekturama
- Kako najčešće gradimo velike softverske sisteme?
- Najčešće ih sagledavamo izdeljene po slojevima
- **Višeslojne** arhitekture
 - klijentski sloj
 - sloj biznis logike
 - sloj podataka



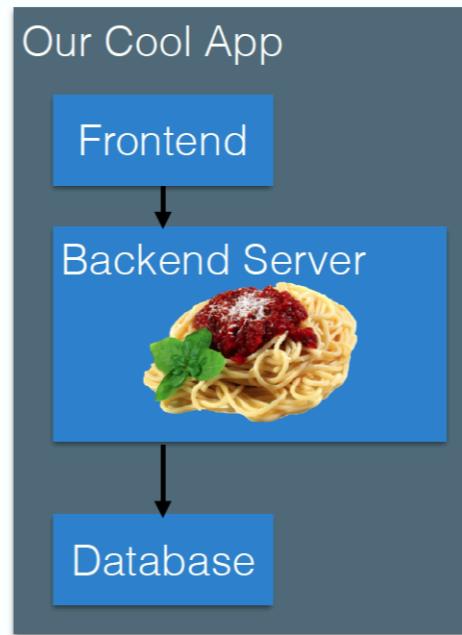
Ali kako monolitne aplikacije izgledaju ako se pogledaju funkcionalnosti koje obezbeđuju?



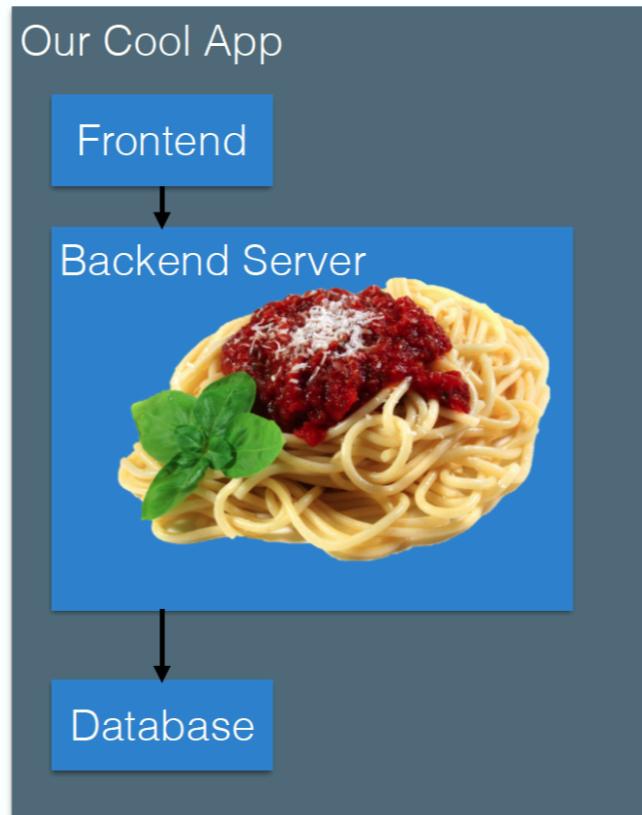
- Ovakav dizajn dovodi do vrlo bliske povezanosti komponenti kako unutar samog servisa tako i između servisa, jer se komunikacija zaniva na direktnom sinhronom pozivanju metoda
 - Pozivi metoda mogu dugo trajati - krajnji korisnik, kao i thread u kome je poziv izvršen su blokirani dok čekaju završetak operacije

Šta se dešava kada višeslojnoj arhitekturi dodajemo nove funkcionalnosti?

- Obično se poveća složenost *backend* sloja (poslovna logika)

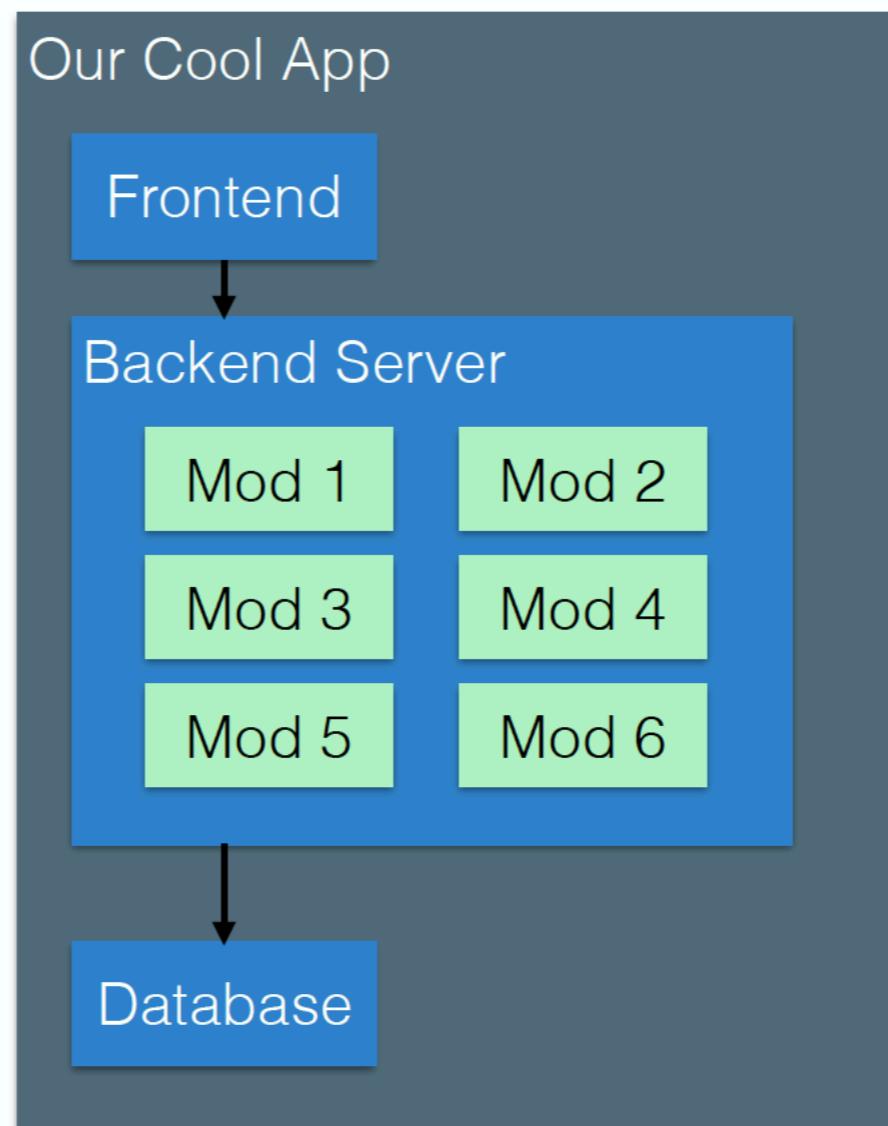


- Ako želimo još više funkcionalnosti?
 - značajno zakomplikujemo srednji sloj



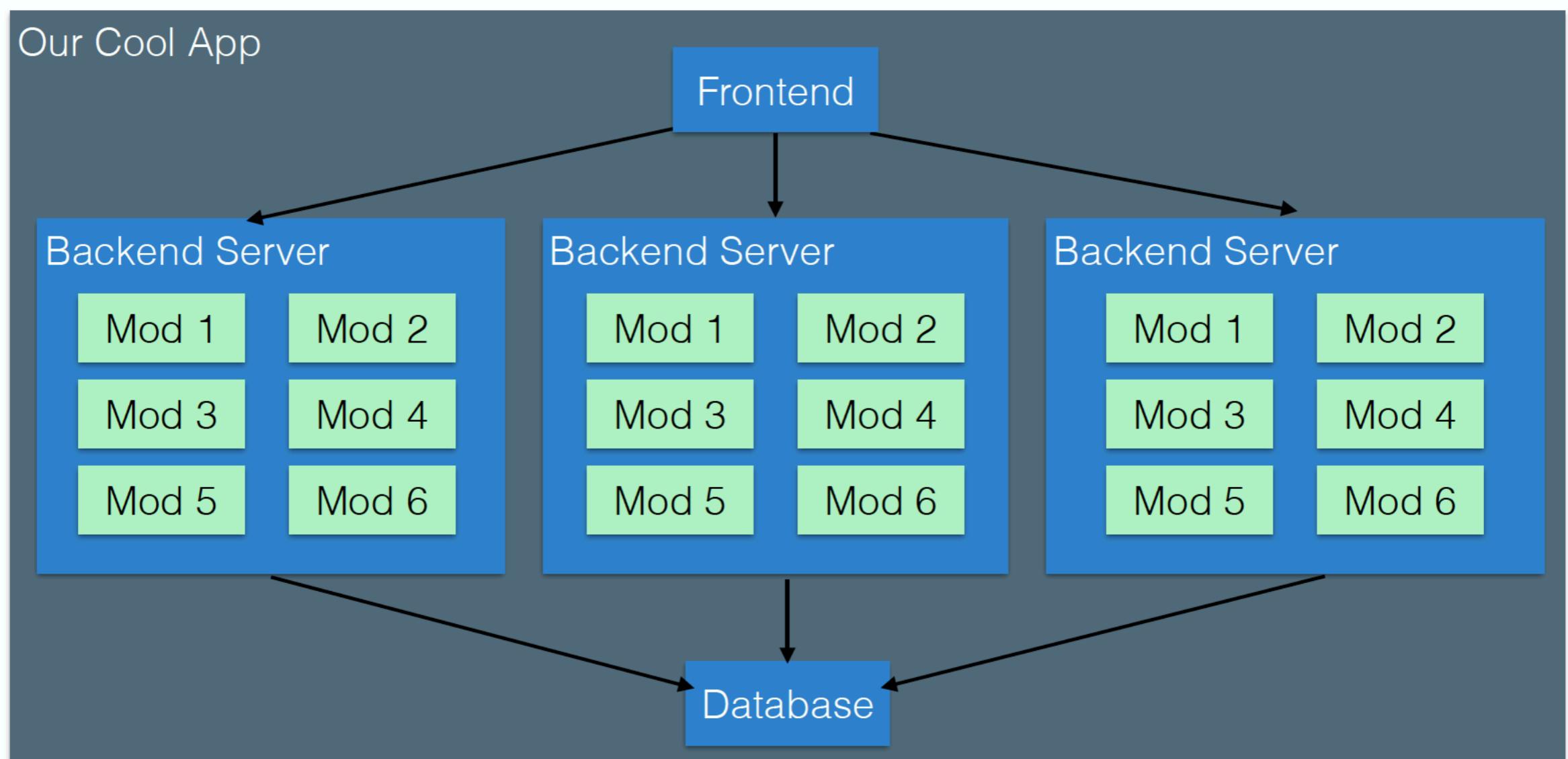
Šta se dešava kada višeslojnoj arhitekturi dodajemo nove funkcionalnosti?

- Kod je na sreću ipak organizovan po modulima, pa povećanje broja funkcija dovodi do povećanja broja modula u srednjem sloju aplikacije



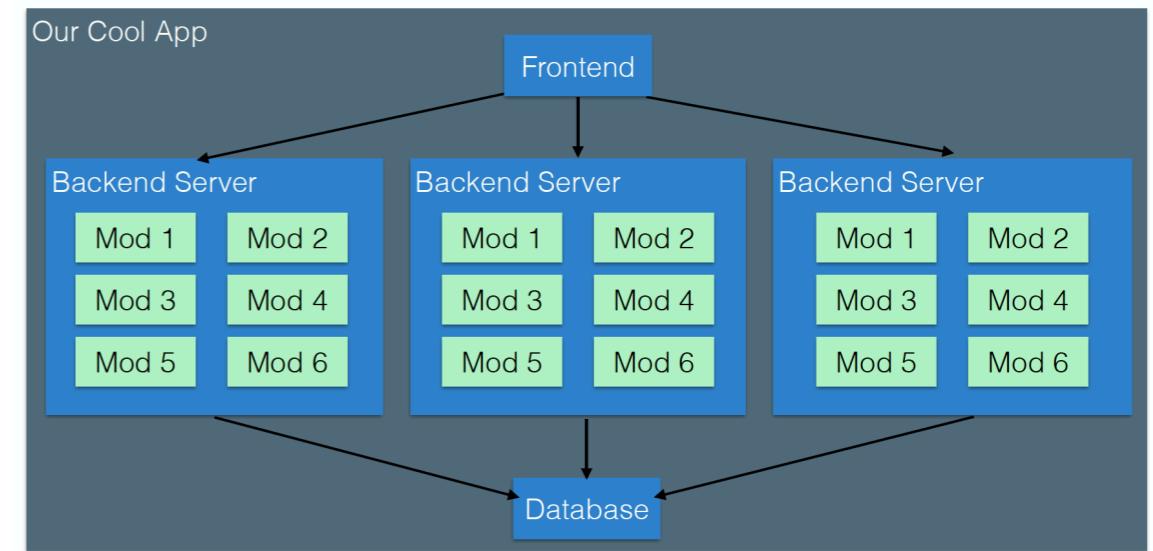
Kako se skaliraju monolitne aplikacije da odgovore povećanim zahtevima?

- Pokreće se više instanci *backend* aplikacija sa identičnim modulima kako bi odgovorile na povećane zahteve



Kako se skaliraju monolitne aplikacije da odgovore povećanim zahtevima?

- Ovakav princip skaliranja je tipičan za monolitne aplikacije
- Ako imamo veliki broj servera, na svakom se “vrte” isti moduli
- Šta ako su nam neke funkcije sistema više opterećene nego neke druge?
- Da li je baš najoptimalnije da su nam svi moduli implementirani na isti način (isti programski jezik, isti runtime, koriste istu bazu...)



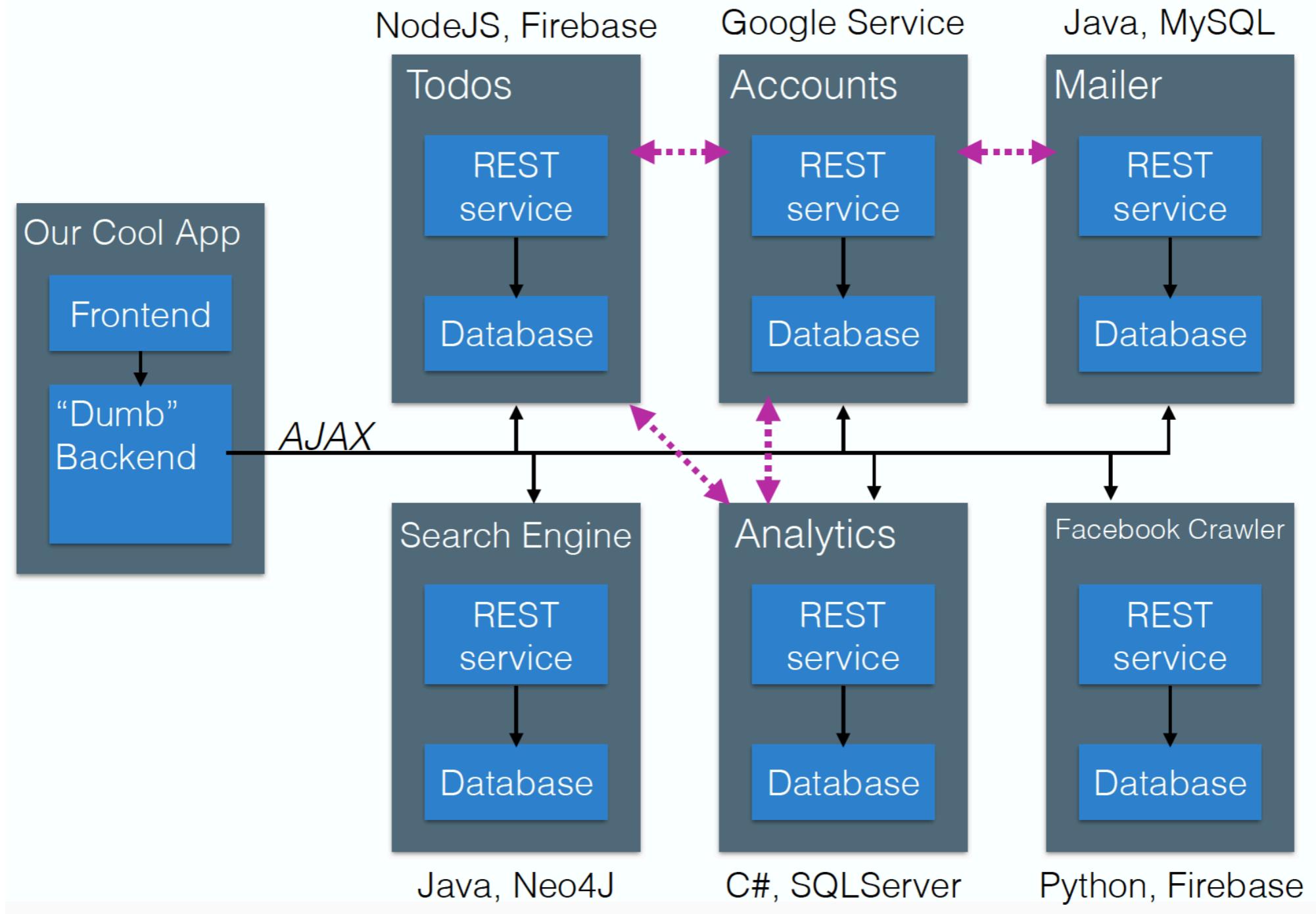
Mirkoservisni pristup?

- Mikroservisi se orijentišu na jednostavnu poslovnu funkcionalnost - jedan zadatak, i kao takvi su po pravilu mali moduli.
- Nema pravila koliko mali moraju biti, i ne treba se koncentrisati na broj linija koda nego na funkcionalnost.
 - “Pravilo 2 pizze” - ako vam je tim koji je neophodan da realizujete mikroservisni modul toliko veliki da ne možete da ga nahranite sa samo 2 pizze - nešto ste omašili
- Ključna je jednostavnost interfejsa - ona obično dovodi i do relativno male implementacije, ali to ne mora uvek biti slučaj.
- Mikroservisni modul treba tretirati kao nezavisnu aplikaciju ili nezavisni proizvod. Poželjno je da ima sopstveni repozitorijum za upravljanje kodom, i sopstveni *build* i *deployment*.

Ponovna iskoristivost i granularnost mikroservisa

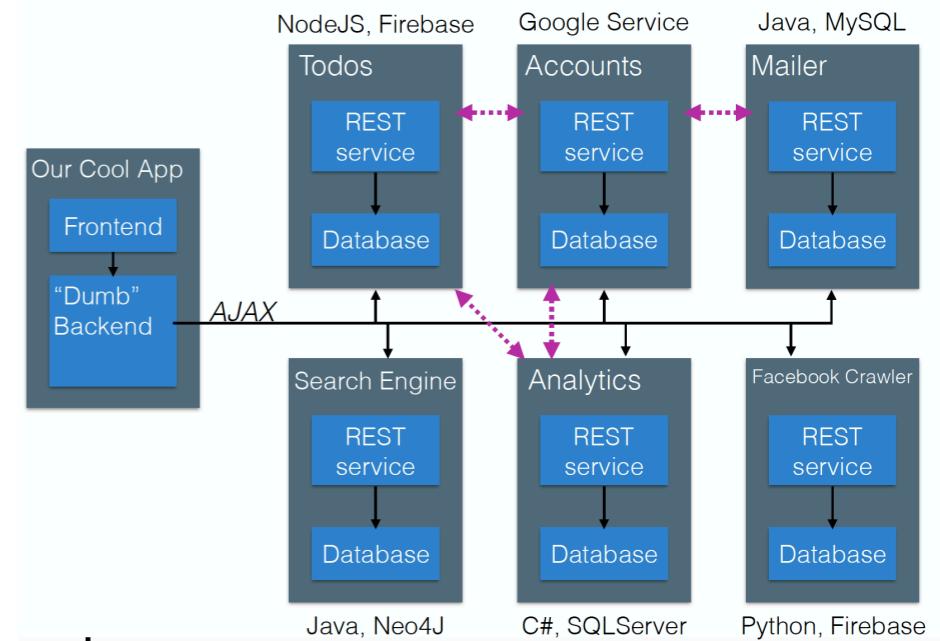
- Iako je ponovna iskoristivost poželjna, nije i obavezna i nije jedini razlog njihovog uvođenja
 - lokalne optimizacije korisničkog interejsa kako bi se poboljšao odziv,
 - lakše prilagođavanje potrebama korisnika...
- Granularnost mikroservisa se takođe određuje na osnovu poslovnih potreba
 - npr. praćenje paketa, prognoze vremena se danas vrlo često mogu koristiti kao servisi treće strane
- Problem latentnosti servisa - ukoliko je previše usitnjen i zateva previše poziva ka drugim mikroservisima, može se osetiti problem usporenja aplikacije

Kako izgleda složena aplikacija realizovana po principima mikroservisa

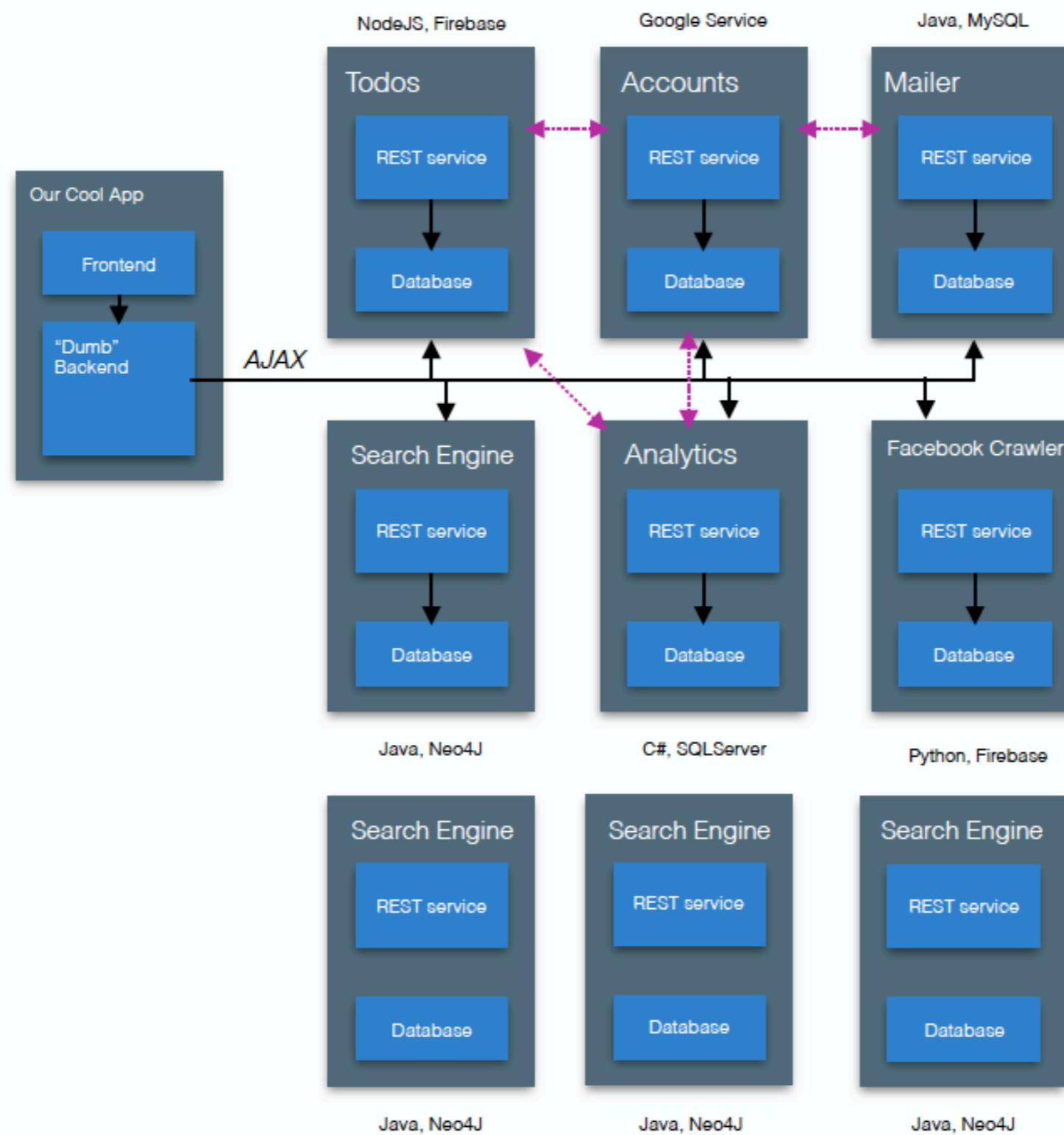


Šta je ovde bitno različito

- Nema glomaznih modula sa “špageti” kodom
- Komponente se mogu razvijati potpuno nezavisno jedna od druge
 - različiti programski jezici, izvršna okruženja, OS, hardware, DB
- Komponente se mogu relativno jednostavno menjati
 - Može se zamjeniti i kompletna tehnologija modula
 - Moguće različito skaliranje različitih komponenti



Kako se skaliraju mikroservisne aplikacije da odgovore povećanim zahtevima?



Razlika u skaliranju

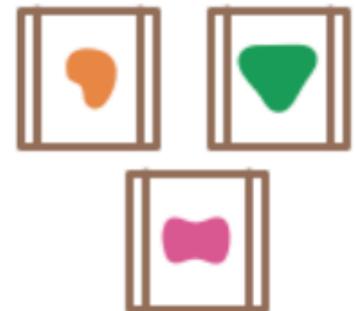
A monolithic application puts all its functionality into a single process...



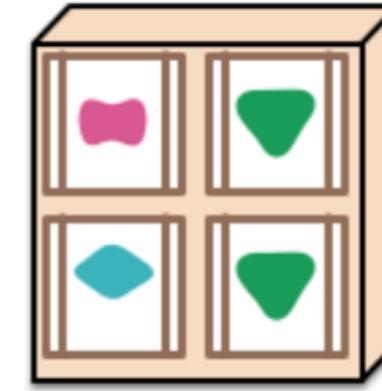
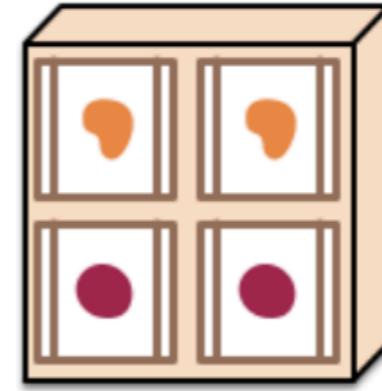
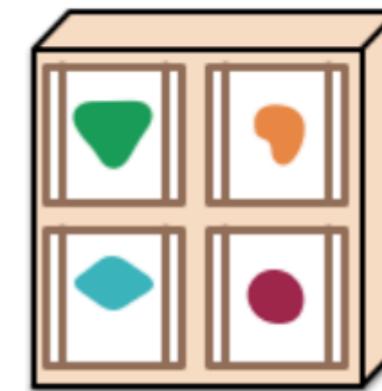
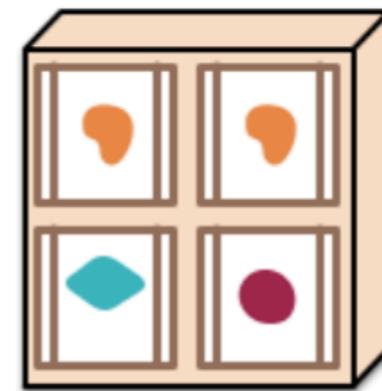
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



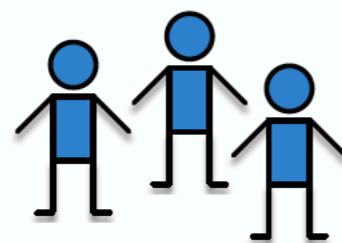
Prepostavke za uspešnu mikroservisnu arhitekturu

- 1 komponenta = 1 servis
- “pametni” endpointi i “glupi” komunikacioni kanali (za razliku od npr. ESB, koji vrlo često sadrži složene mehanizme rutiranja, transformacija i sl.)
- decentralizovano upravljanje
- decentralizovano upravljanje podacima
- automatizacija infrastrukture
- dizajnirati arhitekturu da trpi otkaze
- evolutivni dizajn

Koliko velike komponente treba da budu?

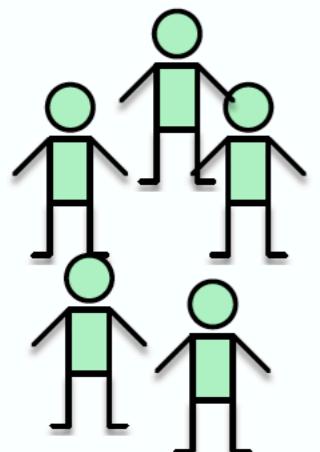
- Komponente se mogu formirati kao
 - biblioteke (moduli)
 - servisi
- Mikroservisi promovišu modularizaciju pomoću servisa
 - svaku komponentu možete pojedinačno zamjeniti
 - svaku komponentu možete nezavisno ažurirati
- Ovo znači da se mogu i nezavisno razvijati, testirati...

Klasična organizacija razvoja



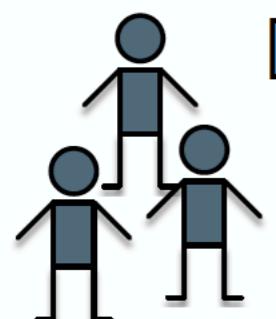
Frontend

Orders, shipping, catalog



Backend

Orders, shipping, catalog

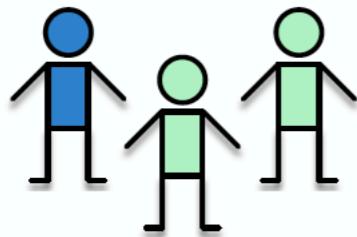


Database

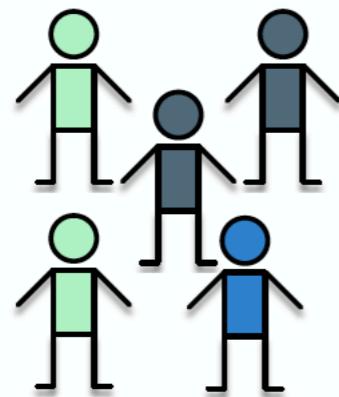
Orders, shipping, catalog

- 1 tim po sloju, i svaki tim se bavi funkcionalnostima koje su u datom sloju bez obzira na to koji segment aplikacije opslužuju

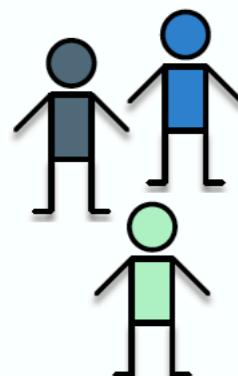
Mikroservisi - organizacija razvoja usmerena na prozivod ili segment poslovanja



Orders



Shipping



Catalog

- 1 tim se bavi jednim segmentom poslovanja, obezbeđuje funkcionalnost za jedan proizvod i sastoji se od ljudi svih profila koji su neophodni da taj segment proradi
- Timovi se koncentrišu na samo jedan poslovni zadatka i mogu da komuniciraju direktno sa klijentima

Decentralizovano upravljanje

- Odluke o načinu implementacije se donose decentralizovanno
- Servisi podatke razmenjuju ISKLJUČIVO preko javno dostupnih API-ja, **nema oslanjanja na deljene baze podataka**
- Omogućava svakom servisu da podacima upravlja na način koji je najpogodniji sa stanovišta tog servisa

Automatizacija infrastrukture

- Za efikasno korišćenje mikroservisnih arhitetura najverovatnije će biti neophodno da:
 - Razvijate servise nezavisno
 - Servise nezavisno puštate u rad (deployment)
- Morate obezbediti:
 - mogućnost da dobijete serverske kapacitete brzo kako bi mogli da iskoristite skalabilnost rešenja
 - dobar monitoring kako bi bili u stanju da vidite kada servisi ne komuniciraju kako treba
 - brz *deployment* novih ili ažuriranih servisa
 - razvijenu kulturu jake integracije timova koji rade nadzor servisa u radu i tim koji radi razvoj ("devops")
- Ključna stvar za uspeh mikroservisa je automatizacija svega navedenog

Dizajniranje sistema tako da bude otporan na otkaze

- Konceptom mikroservisa ukupna struktura sistema može ozbiljno da se zakomplikuje
- Mnogo “pokretnih” delića koji mogu da otkažu:
 - Pojedini servisi mogu imati greške
 - Pojedini servisi mogu raditi vrlo sporo
 - Celi serveri mogu pasti
 - Sa 60,000 HD ova 3 dnevno će verovanto otkazati
- Ključna stvar - dizajnirati svaki servis prepostavljajući da u nekom momentu sve ono od čega taj servis zavisi može prosto da nestane i bude nedostupno
 - servis tada mora otkazati “gracefully”
 - Netflix - “ChaosMonkey”

Održavanje konzistencije

- Jedno od pravila mikroservisa je ne koristiti deljene baze podataka
- Neke podatke ipak verovatno koristi više servisa
- Ažuriranje se vrši preko asinhronih poziva
- Nema garancije da će svi moduli obaviti ažuriranje u istom trenutku
- Garantuje se da će oni u nekom momentu ažurirati podatke (*eventual consistency*)

Održavanje konzistencije

- Glavni problem je što različiti servisi mogu odgovoriti na zahtev u različitim vremenskim trenucima
- Šta ako jedan zahtev rezultuje promenom resursa na jednom servisu, ali ostali još nisu procesirali korespondirajuće zahteve?
 - Moguće je da se dobiju nekonzistentna stanja za korespondirajuće resurse.
 - Mora se napisati dodatna logika da korektno obradi ovakve situacije.

Da li su mikroservisi idealno rešenje?

- Zavisi od konteksta
- Za svaku aplikaciju neophodno je analizirati dobre i loše strane mikroservisne arhitekture, pa tek onda odlučiti.
- Jako su dobro rešenje za sisteme koji su po prirodi distribuirani, namenjeni velikom broju različitih korisnika, koji koriste različite funkcionalnosti sistema, za izgradnju elastičnih i otpornih sistema koji se lako prilagođavaju promenama opterećenja i zahteva korisnika...
- ... ali mogu povećati kompleksnost kompletног rešenja, usporiti odziv celokupnog sistema, ako je za neku spolja zahtevanu akciju neophodna koordinacija više mikroservisnih modula (moduli previše “pričaju međusobno”)

Kada koristiti koji pristup arhitekturi?

- Monolitne:
 - Jednostavnija za razvoj - jedna tehnologija za celokupno rešenje, lakše testiranje celokupnog sistema
 - Pogodne za nedistribuirane sisteme
 - Dobra za početne faze razvoja sistema
- Mirkoservisi:
 - Kada nam odgovara i treba modularna implementacija
 - Kada je neophodna visoka dostupnost - čak i ako jedan mikroservis otkaže ostatak funkcionalnosti se i dalje može koristiti
 - Modularnost sistema - jasno odvajanje funkcionalnosti se forsira mikroservisnim pristupom
 - Lako se po potrebi razvija na različitim platformama - omogućavajući da se iskoriste najbolje tehnologije za pojedine probleme - integracija se mora dobro testirati
 - Netflix ima vrlo pozitivno iskustvo - praktično mu omogućava dnevne i čak satne ažuriranja komponenti
- Često se u praksi nalazi i neko hibridno rešenje