## 6.1 Uvod

- Porazdeljen sistem je sistem, kjer se procesiranje izvaja na več računalnikih.
- Računalniki med seboj komunicirajo preko omrežja.
- Program je razdeljen na več delov, ki se vzporedno izvajajo na več računalnikih.
- Porazdeljen sistem ima pogosto opravka z:
  - heterogenimi okolji,
  - nepredvidljivimi latenčnimi časi in kapacitetami omrežja,
  - Nepredvidljivimi napakami na omrežju ali računalnikih povezanih v omrežje



## 6.1 Uvod

- Večina sodobnih IS je zasnovanih v obliki porazdeljenih sistemov.
- Zasnove porazdeljenih sistemov sledijo različnim arhitekturnim stilom in vzorcem kot npr.:
  - Odjemalec-strežnik
  - Storitveno usmerjene arhitekture (glej ločeno podpoglavje o SOA)
  - Cevovodi in filtri
  - Peer-to-peer
  - Data-centric
  - Event driven
  - Blackboard
  - …

## 6.2 Strežniki in nivojske arhitekture

- Zelo pogoste so arhitekture temelječe na vzorcu odjemalec-strežnik.
- Osnovni komponenti tako porazdeljenega sistema sta odjemalec (zahteva storitev) in strežnik (izvaja storitev).
- Procesiranje strežnik-odjemalec naj bi potekalo relativno uravnoteženo, pri čemer specializirane storitve izvaja strežnik.



## 6.2 Strežniki in nivojske arhitekture

- V porazdeljenem sistemu lahko sodelujejo različni strežniki:
  - Datotečni strežnik (File server)
  - Podatkovni strežnik (Database server)
  - Strežnik za skupinsko delo (Groupware server)
  - Spletni strežnik (Web server)
  - Poštni strežnik (Mail server)
  - Strežnik objektov (Object server)
  - Strežnik za tiskanje (Print server)
  - Aplikacijski strežnik (Application server)
  - ...

## 6.2 Strežniki in nivojske arhitekture

- Porazdeljeni sistemi delujejo na arhitekturah z več nivoji (ravnmi).
- Najbolj znani sta dvo in tri-nivojska arhitektura.

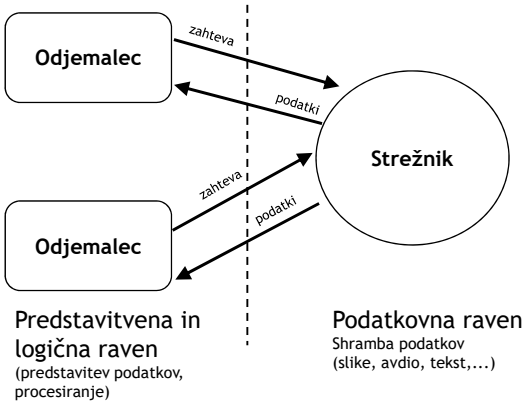


## 6.2 Strežniki in nivojske arhitekture

- Dvo-nivojsko arhitekturo (tudi arhitektura odjemalec-strežnik) sestavlja:
  - Predstavitvena in logična raven (predstavitev in procesiranje podatkov, ki jih vrne podatkovni strežnik); Fizično realizirana na odjemalcu.
  - Podatkovna raven (shramba podatkov); Fizično realizirana na strežniku.
- Dvo-nivojska arhitektura je smiselna, kadar je malo procesiranja podatkov.
- Primeri: spletni strežnik, datotečni strežnik, podatkovni strežnik, itd. do katerih neposredno dostopajo odjemalci

## 6.2 Strežniki in nivojske arhitekture

**Primer dvo-nivojske arhitekture**






## 6.2 Strežniki in nivojske arhitekture

- Slabosti dvo-nivojske arhitekture se pokažejo, če:
  - imamo veliko odjemalcev;
  - je potrebno veliko procesiranja;
- Posledice:
  - Visoki stroški skrbništva, delovanja in vzdrževanja sistema;
  - Visoki stroški vpeljave tehnoloških trendov;
  - Slaba izkoriščenost programske opreme. Tipičen uporabnik v povprečju uporablja cca 10% zmogljivosti programske opreme, ki mu je na voljo.

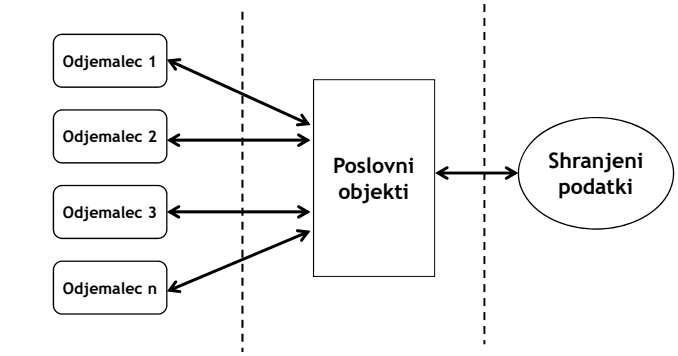
## 6.2 Strežniki in nivojske arhitekture

- Tri-nivojska arhitektura – pojavi se kot rešitev težav dvo-nivojske arhitekture.
- V tri-nivojski arhitekturi se predstavitveni in procesni nivo ločita. Nivoji so:
  - Predstavitvena raven (vmesnik med uporabnikom in sistemom);
  - Procesna raven (procesiranje, ki ga zahteva sistem); Fizično realizirana na aplikacijskem strežniku.
  - Podatkovna raven (shramba za trajne podatke); Fizično realizirana na podatkovnem strežniku.



## 6.2 Strežniki in nivojske arhitekture

**Tipična tri-nivojska arhitektura**




Predstavitvena raven | Procesna raven | Podatkovna raven

## 6.2 Strežniki in nivojske arhitekture

- Glavne prednosti tri-nivojske arhitekture:
  - Neodvisnost aplikacije od tehnologije za shrambo podatkov;
  - Procesiranje se seli iz odjemalca na strežnik (cenejša vpeljava, skrbništvo, vzdrževanje in delovanje);
  - Procesiranje se nanaša na objekte - skladno z objektno paradigmo;
  - Odjemalci lahko uporabljajo podatke različnih podatkovnih virov (na različnih podatkovnih strežnikih),
  - Zmanjšanje števila povezav (pri dvo-nivojski arhitekturi reda velikosti M×N povezav, pri tri-nivojski pa samo M+N).
  - Večja zanesljivost delovanja sistema;
  - Večja prilagodljivost in odprtost sistemov za nove tehnologije;



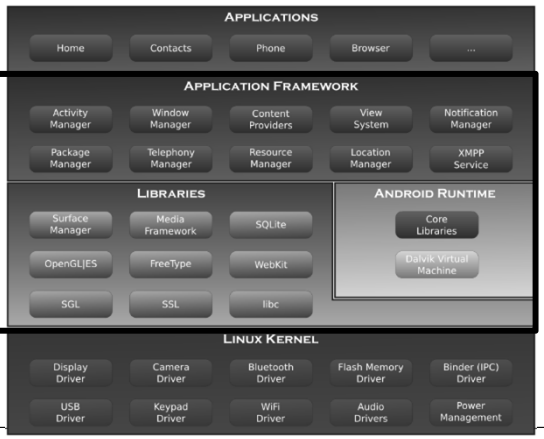
## 6.3 Vmesni sloj (Middleware)

- Vmesni sloj zagotavlja aplikacijam dodatne storitve, ki jih operacijski sistem ne ponuja.
- Gre za „lepilo“ med aplikacijami in operacijskim sistemom.
- V porazdeljenih sistemih nudi storitve, ki omogočajo medsebojno povezovanje programskih komponent oz. poslovnih aplikacij

## 6.3 Vmesni sloj (Middleware)

- Diskusija – Android kot vmesni sloj med aplikacijami in Linuxom

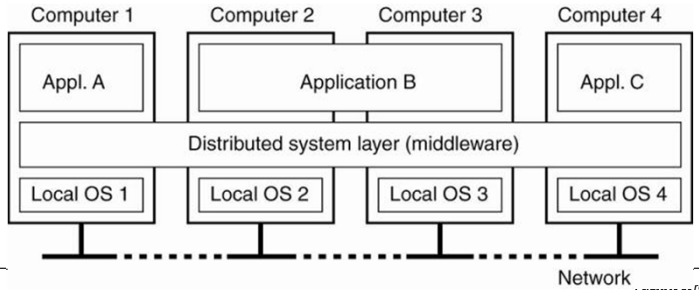
Čeprav Android navadno razumemo kot OS, pa v odnosu do Linux-a pravzaprav igra vlogo vmesnega sloja.






## 6.3 Vmesni sloji in druge storitve v porazd. sistemih

- **Vmesni sloj** pri porazdeljenih sistemih so vmesne programske ravni, ki prevzemajo ukaze od odjemalcev in jih posredujejo ustreznim strežnikom.
- Vmesni sloj je ključen pri vsaki komunikaciji odjemalec-strežnik.
- Pri dobri arhitekturi razvijalec vmesni sloj uporablja, brez da bi vedel podrobnosti njegovega delovanja – vmesni sloj je transparenten.



## 6.3 Vmesni sloji in druge storitve v porazd. sistemih

- Vmesni sloji so ključen element pri **sistemski integraciji** (povezavi različnih sistemov v enotno, celovito rešitev).
- V preteklosti je bila razvitih vrsta standardov in tehnologij za realizacijo vmesnih slojev.
  - CORBA (Common Object Request Broker Architecture),
  - DCOM (Distributed Component Object Model),
  - RMI (Remote Method Invocation).
- Omenjene standarde danes pogosto nadomeščajo standardi in tehnologije, ki podpirajo koncept storitev.



## 6.3 Vmesni sloji in druge storitve v porazd. sistemih

- Za delovanje porazdeljenih sistemov so v okviru vmesnega sloja na voljo številne storitve. Na primer:
  - Delo z datotekami (File services)
  - Dodeljevanje imen (Name services)
  - Delo z imenikom (Directory services)
  - Nadzor časa (Time services)
  - Replikacija (Replication services)
  - Nadzor transakcij (Transaction service)
  - Nadzor sočasnega dostopa (Concurrency control services)
  - Nadzor varnosti (Security services)

## 6.4 Integracijska arhitektura

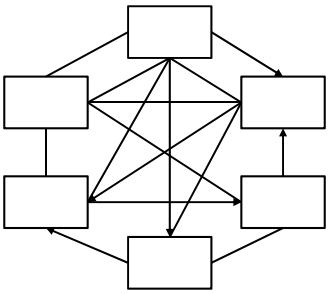
- Ponovitev: Namen integracije poslovnih aplikacij (EAI) je zagotoviti povezavo med uporabniškimi in zalednimi sistemomi in na ta način omogočiti hitrejši in učinkovitejši odziv na različne poslovne dogodke in uporabniške zahteve.
- Pomen integracije poslovnih aplikacij (EAI) za poslovanje je opisan v poglavju 2.6.2
- Na voljo imamo različne integracijske arhitekture



## 6.4 Integracijska arhitektura

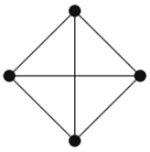
- Tipične integracijske arhitekture:
  - Integracija točke do točke (point-to-point integration)
  - Integracija s pomočjo centralne točke (hub and spoke)
  - Integracija z uporabo storitvenega vodila (ESB – enterprise service bus)

## 6.4 Integracija točke do točke



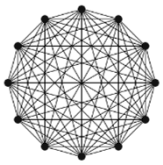
- Starejši pristop
- Enostavnost in hitrost vzpostavljanja na začetku, ko je povezanih malo sistemov
- Velik problem ob širjenju integracije (skalabilnost), število vseh povezav (poln graf) je *p=n(n-1)/2, kjer n = število vozlišč*
- *„Špageti“*

n=4, p=6



Vir: wiki, Koko90

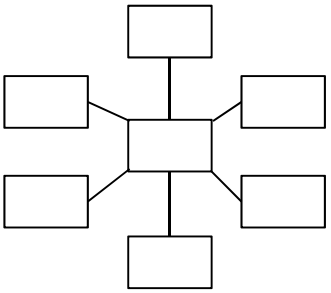
n=12, p=66



Vir: wiki, Koko90



## 6.4 Integracija s pomočjo centralne točke (hub and spoke)



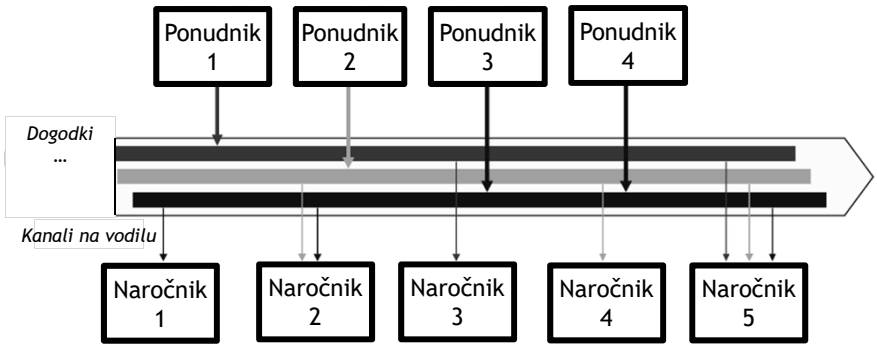
Vir: wiki, Koko90

- Število povezav je enako številu integriranih sistemov (rast števila povezav ni eksponentna)
- Slabost je da odpoved/preobremenitev centralnega sistema povzroči izpad celotne integracije
- Tudi kadar je prevladujoč vzorec integracija s pomočjo centralne točke, se pogosto uporablja še dodatne integracije tipa točka do točke.
- Sinhrona zahteva-odgovor (model „pull)
- Tak tip integracije uporablja tudi CORBA, RMI

*Ta vzorec uporablja tudi EAI.*
*Problem še vedno visoka kompleksnost*

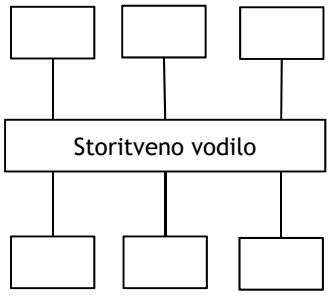
## 6.4 Integracija s pomočjo storitvenega vodila

- Ideja: ponudnik storitve (strežnik) – naročnik (odjemalec)






## 6.4 Integracija s pomočjo storitvenega vodila




Vir: wiki, Koko90

- Omogoča prilagodljiv način za izmenjavo podatkov med aplikacijami
- Uporaba s storitveno usmerjeno arhitekturo (SOA) (glej naslednje poglavje)
- Model ponudnik – naročnik
- Asinhrona dostava dogodkov preko vodila (model „push“)

## 6.4 Integracija s pomočjo storitvenega vodila

- Prednosti:
  - Skupinska komunikacija
  - Inteligentno usmerjanje sporočil
  - Mediacija sporočil med različnimi sistemi (različni protokoli, zapisi, varnostni nivoji)
  - Zagotavljanje varnosti
  - Zagotavljanje kakovosti dostave in zagotavljanje transakcijske obravnave sporočil
  - Upravljanje procesov
  - Nadzor nad delovanjem



## 6.5 Storitveno usmerjena arhitektura (SOA)

- Vsebina:
  - Kaj je SOA in glavna načela storitvene usmerjenosti
  - Temeljni pojmi SOA
  - Tehnološko ozadje
  - SOA kot storitveni nivo
  - Primernost

## 6.5 Storitveno usmerjena arhitektura (SOA)

- Kaj je SOA?
  - Storitveno usmerjena arhitektura (SOA) temelji na ohlapno povezanih sistemih, združenih v celoto, pri čemer so posamezni deli med seboj neodvisni in tečejo na poljubnih platformah.
  - Gre za koncept, ki je zaradi težav s kompleksnimi sistemi, za katere značilen problem so visoki stroški povezovanja in vzdrževanja, v zadnjih letih postal zelo popularen.
  - Osnovni gradniki storitveno usmerjene arhitekture so seveda storitve.



## 6.5 Storitveno usmerjena arhitektura (SOA)

- Glavna načela storitvene usmerjenosti so (1/2):
  - Šibka sklopljenost - storitve ohranjajo odnose, ki minimizirajo odvisnosti med njimi in ohranjajo le zavedanje ena druge.
  - Storitvena pogodba – storitve se držijo komunikacijskega dogovora, ki je določen z enim ali več opisom storitev in podobnih dokumentov.
  - Neodvisnost – storitev je neodvisna od drugih storitev, in sicer v smislu nadzora nad svojo logiko.
  - Abstrakcija – razen opisa storitve je logika storitve nedostopna zunanjemu svetu.

## 6.5 Storitveno usmerjena arhitektura (SOA)

- Glavna načela storitvene usmerjenosti so (2/2):
  - Ponovna uporaba – logika je razdeljena, ločena oziroma razbita v različne storitve z namenom možnosti ponovne uporabe.
  - Storitve minimizirajo količino informacij, ki pripada določeni aktivnosti – so brez stanja.
  - Odkrivanje – storitve so načrtovane tako, da jih lahko opišemo in najdemo; tako lahko do njih dostopamo preko temu namenjenih mehanizmov.



## 6.5 Storitveno usmerjena arhitektura (SOA)

- Temeljni pojmi SOA:
  - Storitev
  - Opis storitve
  - Sporočilo

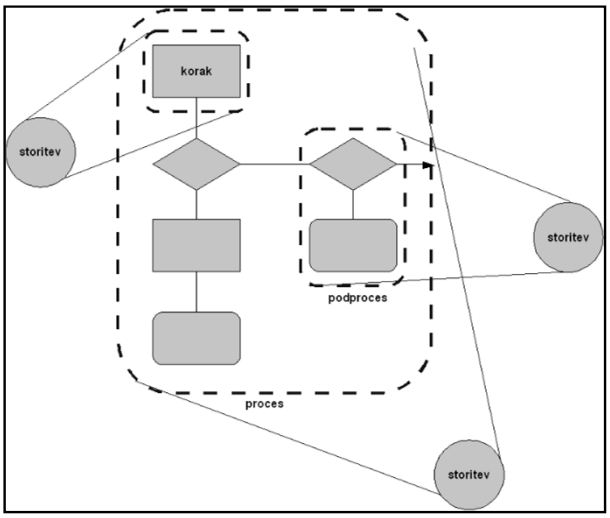
## 6.5 Storitveno usmerjena arhitektura (SOA)

- Storitev (1/2):
  - Storitev je ponovljivo poslovno opravilo, npr. preveri stanje na računu, odpri nov račun.
  - Storitvi pravimo tudi storitveno usmerjena logična enota.
  - Da ohranjajo neodvisnost, storitve zajemajo logiko znotraj točno določenega konteksta. Ta kontekst je lahko specifičen za poslovno opravilo, poslovno entiteto ali kakšen drug logičen skupek.
  - Logika, ki jo storitev vsebuje, je namenjena za reševanje določenega problema, katerega kompleksnost je lahko zelo različna.



## 6.5 Storitveno usmerjena arhitektura (SOA)

- Storitev (2/2):



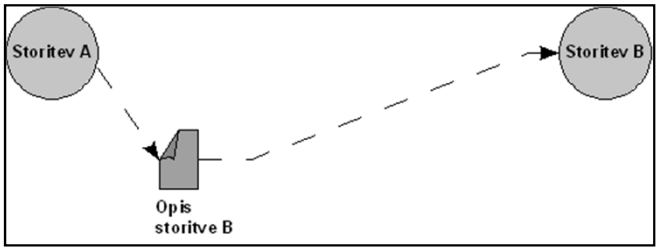
## 6.5 Storitveno usmerjena arhitektura (SOA)

- Opis storitve (1/2):
  - Znotraj SOA storitve lahko uporabljajo druge storitve ali drugi programi. Da bi storitev lahko uporabili morajo poznati njeno delovanje. Zato je potreben opis storitve.
  - V osnovi mora opis storitve zajemati ime storitve, podatke, ki jih pričakuje na vhodu ter podatke, ki jih vrne na izhodu.
  - Z uporabo koncepta opisa storitev dosežemo šibko sklopljenost sistema.



## 6.5 Storitveno usmerjena arhitektura (SOA)

- Opis storitve (2/2):



## 6.5 Storitveno usmerjena arhitektura (SOA)

- Sporočilo:
  - Sporočilo je neodvisna enote komunikacije.
  - Ko storitev pošlje sporočilo, nad njim zgubi nadzor (samoobvladovanje delov procesne logike).
  - Sporočilo lahko obravnavamo kot način komunikacije, ki ohranja šibko sklopljenost.



## 6.5 Storitveno usmerjena arhitektura (SOA)

- SOA ne predpisuje uporabe konkretnih tehnologij. Prvotno uveljavljeni standardi:
  - WSDL (Web Service Description Language) – podaja opis storitve
  - SOAP - format sporočanja med storitvijo in uporabnikom
  - UDDI - standardiziran format za registracijo storitev

## 6.5 Storitveno usmerjena arhitektura (SOA)

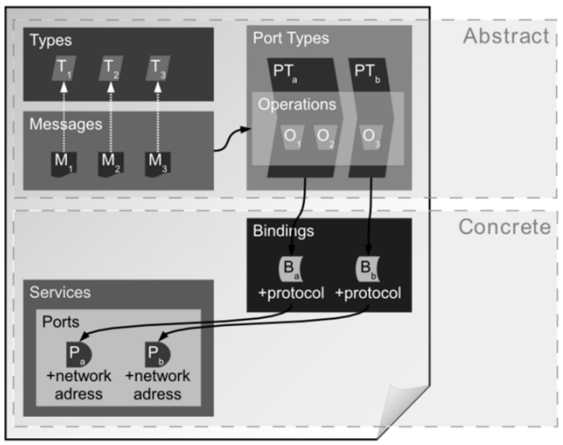
- Kaj je WSDL (1/2):
  - opredeljuje spletne storitve kot zbirke vrat (port, endpoint).
  - storitve opisuje v formatu XML
  - povezuje konkretno izvedbo storitve z njeno abstraktno definicijo
  - definicija opredeljuje:
    - abstraktne tipe
    - abstraktne vmesnike (klici operacij oz. sporočila)
    - tipe vrat v katerih so združeni abstraktni vmesniki
    - povezave med tipi vrat in konkretnimi stroritvami





## 6.5 Storitveno usmerjena arhitektura (SOA)

- Kaj je WSDL (2/2):



## 6.5 Storitveno usmerjena arhitektura (SOA)

WSDL

- Primer WSDL (1/3): abstraktni tipi

```
<!-- Abstract types -->
   <types>
      <xs:schema xmlns="http://www.example.com/wsdl20sample"
                 xmlns:xs="http://www.w3.org/2001/XMLSchema"
                 targetNamespace="http://www.example.com/wsdl20sample">

         <xs:element name="request">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="header" maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:simpleContent>
                           <xs:extension base="xs:string">
                              <xs:attribute name="name" type="xs:string" use="required"/>
                           </xs:extension>
                        </xs:simpleContent>
                     </xs:complexType>
                  </xs:element>
                  <xs:element name="body" type="xs:anyType" minOccurs="0"/>
               </xs:sequence>
               <xs:attribute name="method" type="xs:string" use="required"/>
               <xs:attribute name="uri" type="xs:anyURI" use="required"/>
            </xs:complexType>
         </xs:element>
```



## 6.5 Storitveno usmerjena arhitektura (SOA)

WSDL

- Primer WSDL (2/3): abstraktni vmesniki

```
<!-- Abstract interfaces -->
   <interface name="RESTfulInterface">
      <fault name="ClientError" element="tns:response"/>
      <fault name="ServerError" element="tns:response"/>
      <fault name="Redirection" element="tns:response"/>
      <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
         <input messageLabel="GetMsg" element="tns:request"/>
         <output messageLabel="SuccessfulMsg" element="tns:response"/>
      </operation>
      <operation name="Post" pattern="http://www.w3.org/ns/wsdl/in-out">
         <input messageLabel="PostMsg" element="tns:request"/>
         <output messageLabel="SuccessfulMsg" element="tns:response"/>
      </operation>
      <operation name="Put" pattern="http://www.w3.org/ns/wsdl/in-out">
         <input messageLabel="PutMsg" element="tns:request"/>
         <output messageLabel="SuccessfulMsg" element="tns:response"/>
      </operation>
      <operation name="Delete" pattern="http://www.w3.org/ns/wsdl/in-out">
         <input messageLabel="DeleteMsg" element="tns:request"/>
         <output messageLabel="SuccessfulMsg" element="tns:response"/>
      </operation>
   </interface>
```

Fakulteta za računalništvo in informatiko
Univerza v Ljubljani

WSDL

## 6.5 Storitveno usmerjena arhitektura (SOA)

- Primer WSDL (3/3): povezava s konkretno storitvijo in vrata

```
<!-- Concrete Binding with SOAP-->
   <binding name="RESTfulInterfaceSoapBinding" interface="tns:RESTfulInterface"
            type="http://www.w3.org/ns/wsdl/soap"
            wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
            wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
      <operation ref="tns:Get" />
      <operation ref="tns:Post" />
      <operation ref="tns:Put" />
      <operation ref="tns:Delete" />
   </binding>

<!-- Web Service offering endpoints for both the bindings-->
   <service name="RESTfulService" interface="tns:RESTfulInterface">
      <endpoint name="RESTfulServiceHttpEndpoint"
                binding="tns:RESTfulInterfaceHttpBinding"
                address="http://www.example.com/rest/"/>
      <endpoint name="RESTfulServiceSoapEndpoint"
                binding="tns:RESTfulInterfaceSoapBinding"
                address="http://www.example.com/soap/"/>
   </service>
```



SOAP

## 6.5 Storitveno usmerjena arhitektura (SOA)

- Kaj je SOAP (1/2)?
  - včasih: “Simple Object Access Protocol”; od različice 1.2 le še SOAP
  - SOAP je specifikacija protokola za izmenjavo strukturiranih podatkov pri implementaciji spletnih storitev v računalniških omrežjih oz. format sporočanja med storitvijo in uporabnikom storitve.
  - Sporočila so zapisana v obliki XML.
  - SOAP lahko uporablja različne transportne protokole. Tipično se uporablja HTTP, vendar je mogoče uporabiti tudi druge (npr. SMTP).

SOAP

## 6.5 Storitveno usmerjena arhitektura (SOA)

- Kaj je SOAP (2/2)?
  - primer uporabe SOAP: “Sporočilo v obliki SOAP pošljemo določeni spletni storitvi (npr. vrednost delnic na borzi) skupaj s parametri za iskanje. Spletna storitev vrne odgovor v obliki XML (npr. vrednosti delnic).



SOAP

## 6.5 Storitveno usmerjena arhitektura (SOA)

- Primer SOAP (1/2): zahteva

```
POST /InStock HTTP/1.1
Host: www.example.org
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 xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

SOAP

## 6.5 Storitveno usmerjena arhitektura (SOA)

- Primer SOAP (2/2): odgovor

```
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 xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```



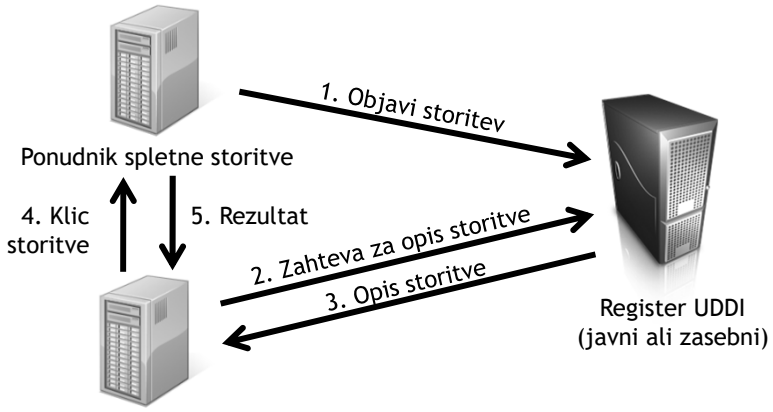
UDDI

## 6.5 Storitveno usmerjena arhitektura (SOA)

- Kaj je UDDI?
  - "Universal Description, Discovery and Integration"
  - Omogoča objavljanje seznamov storitev ter iskanje ustreznih storitev po teh seznamih.
  - UDDI je posebna vrsta spletne storitve, ki upravlja s podatki o ponudnikih storitev

## 6.5 Storitveno usmerjena arhitektura (SOA)

- Kako UDDI deluje?

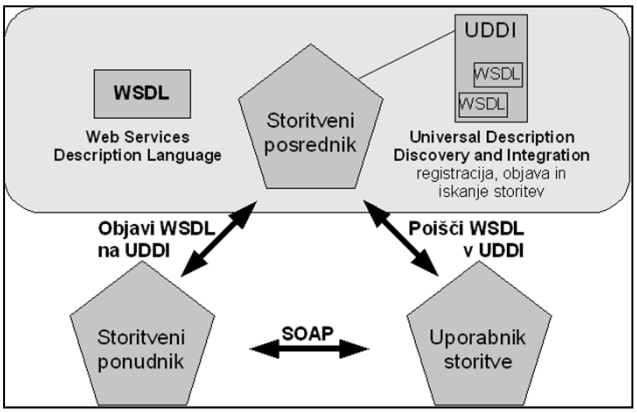







## 6.5 Storitveno usmerjena arhitektura (SOA)

- WSDL, SOAP, UDDI in spletne storitve



## 6.5 Storitveno usmerjena arhitektura (SOA)

- Osnovne karakteristike SOA (1/2):
  - Enkapsulacija
    - Implementacija strežnika je skrita odjemalcu, saj strežnik objavi zgolj vmesnik, tako da je uporabniku podrobnost implementacije skrita.
  - Strežnik ni vezan na fizično lokacijo
    - Strežnik se prijavi v imenik storitev, katero uporabnik povpraša o mrežnem naslovu storitve.
    - Če strežnik zamenja fizično lokacijo, se, ob ponovni prijavi v imenik, ažurira mrežni naslov in uporabniku ni potrebno pomniti naslova storitve

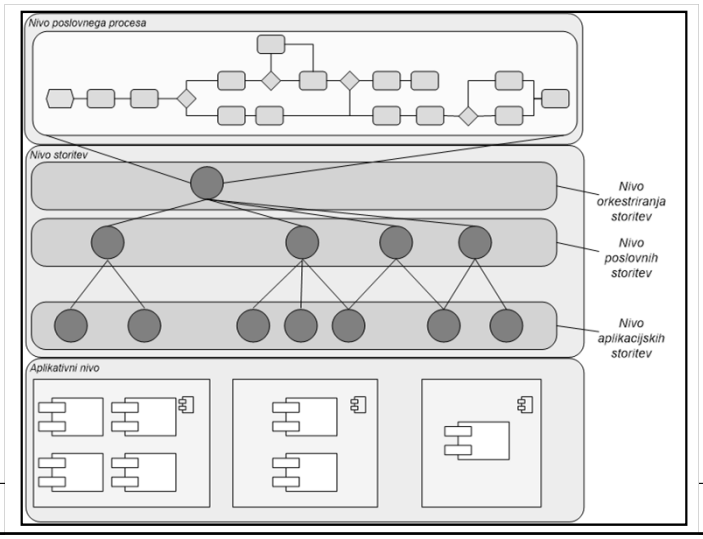


## 6.5 Storitveno usmerjena arhitektura (SOA)

- Osnovne karakteristike SOA (2/2):
  - Skupno delovanje aplikacij
    - Aplikacije, ki se izvajajo na različnih platformah, skupno delujejo tako, da vsaka aplikacija izpostavi vmesnik z uporabo standardnega protokola.

## 6.5 Storitveno usmerjena arhitektura (SOA)

- SOA kot storitveni nivo med poslovnim in aplikativnim nivojem




poslovna pravila, kompozicijska logika

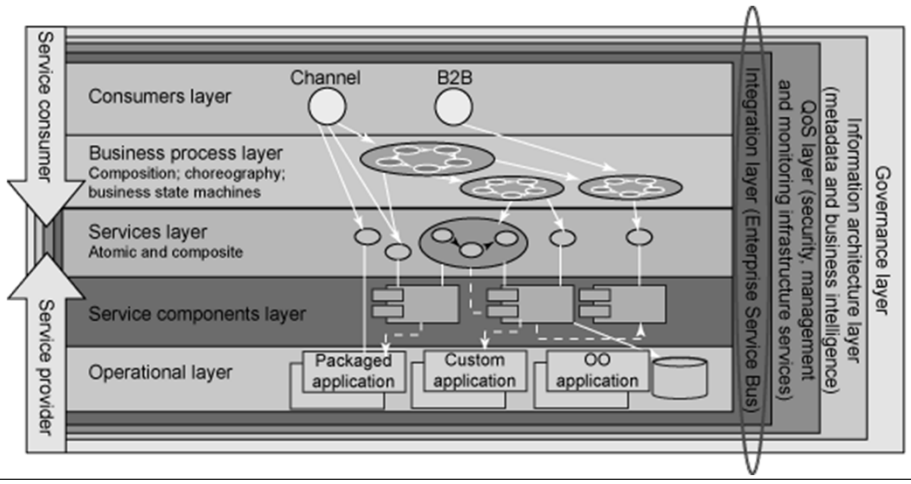
poslovna logika

aplikativna logika



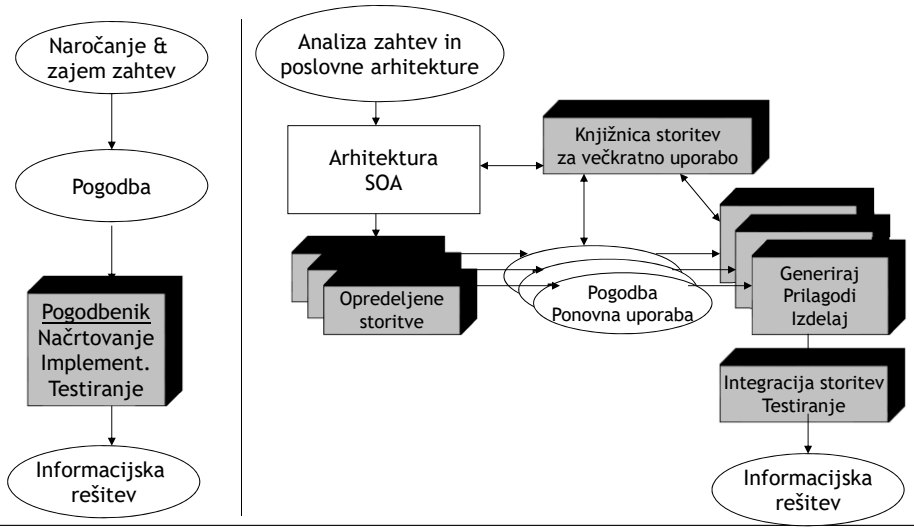
## 6.5 Storitveno usmerjena arhitektura (SOA)

Primer – Enterprise Service Bus (IBM)



## 6.5 Storitveno usmerjena arhitektura (SOA)

Primerjava klasičnega postopka naročanja IR in razvoja/naročanja IR z uporabo SOA






## 6.5 Storitveno usmerjena arhitektura (SOA)

- SOA ni univerzalna rešitev:
  - V stabilnem (nespremenljivem) okolju ponavadi cena vpeljave presega učinkovitost naložbe.
  - Ni potrebe po ponujanju aplikacij v obliki storitev zunanjim poslovnim partnerjem.
  - Realnočasovni sistemi niso primerni za SOA, saj le ta temelji na ohlapno povezani asinhronski komunikaciji, ki ne zagotavlja najhitrejšega odziva.

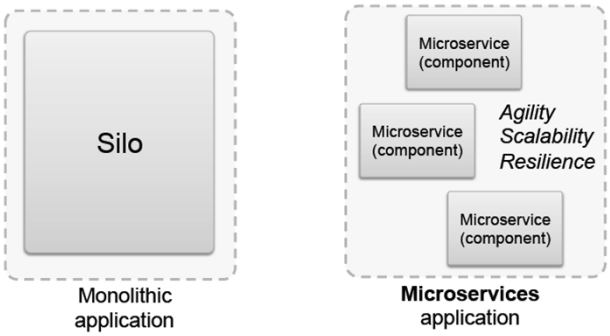
## 6.5 Storitveno usmerjena arhitektura (SOA)

- Težave SOA:
  - Prvotna ideja SOA, da bi skupaj z lahko implementacijo (lahkega) storitvenega vodila poenostavila sodelovanje med različnimi platformami, storitvami oz. aplikacijsko podporo v podjetjih se v veliki meri ni uresničila.
  - Uveljavitev oblačnega računalništva oz. njegova porazdeljenost postavlja uporabo centraliziranega storitvenega vodila pod vprašaj
  - Razvoj in izvedba projektov z uporabo SOA se je pogosto celo zavlekla v primerjavi z uporabo starejših pristopov
  - Sistemi so v marsikaterem pogledu postali celo bolj kompleksni pa tudi dražji
  - Agilni pristopi pri razvoju programske opreme



## 6.6 Mikrostoritve

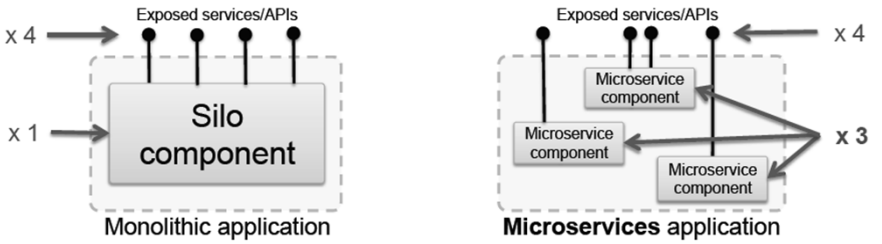
Kaj so mikrostoritve?




Vir: Kim Clark, MuCon

## 6.6 Mikrostoritve

Kaj so mikrostoritve?




Vir: Kim Clark, MuCon

*„Storitev“ v besedi „mikrostoritve“ se nanaša na drobljenje komponent in ne drobljenje zunanjih vmesnikov!*

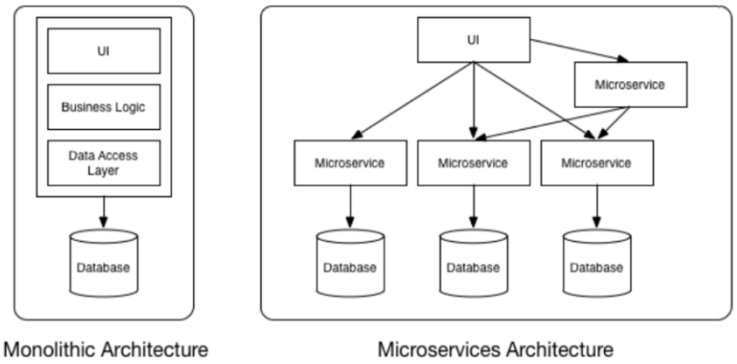


## 6.6 Mikrostoritve

- Kaj je arhitektura mikrostoritev?
  - Arhitektura mikrostoritev je posebna „lahka“ oblika SOA, termin se pojavi 2011 (fine-grained SOA).
  - Pristop temelji na sestavljanju aplikacij iz šibko povezanih drobnozrnatih storitev z uporabo lahkih protokolov.
  - Komunikacija med storitvami preko API neodvisnih od programskega jezika
  - Prednosti pristopa:
    - Lažje razumevanje (drobnozrnate storitve)
    - Paralelizacija razvoja (majhne neodvisne skupine)
    - Neodvisno izboljševanje kode mikrostoritev (npr. refactoring)
    - Neprekinjena dostava in postavitev (continuous delivery and deployment)

## 6.6 Mikrostoritve

- Arhitektura mikrostoritev vs. monolitne aplikacije

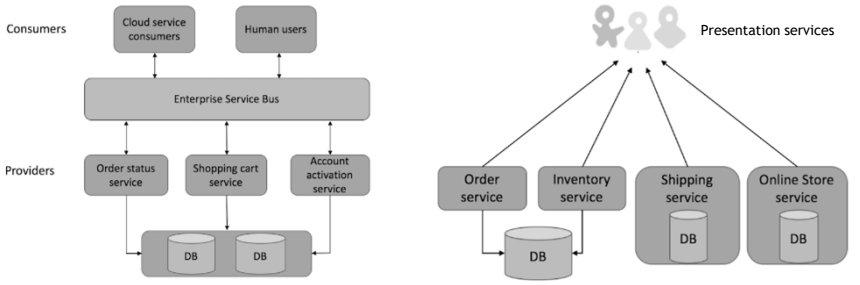



Vir: akioz.com



## 6.6 Mikrostoritve

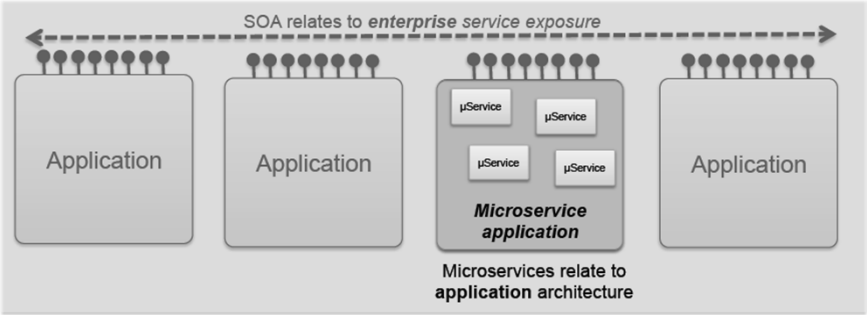
- Arhitektura mikrostoritev vs. SOA




Vir: DZone

## 6.6 Mikrostoritve

- Arhitektura mikrostoritev vs. SOA:

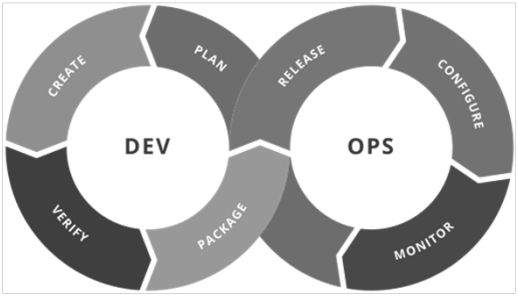



Vir: Kim Clark, MuCon



## 6.6 Mikrostoritve

- Arhitektura mikrostoritev in sodoben razvoj
  - Arhitektura mikrostoritev dobro podpira neprekinjeno dostavo in postavitev (continuous delivery and deployment) oz. podpira paradigmo DevOps
-




Vir: Kharnagy

## 6.6 Mikrostoritve

- Možna alternativa mikrostoritvam – samovsebovani sistemi (self-contained systems SCS)
- V čem se razlikujejo od mikrostoritev:
  - Večji obseg
  - Tipično manjše število SCS
  - V idealnem primeru SCS med seboj ne komunicirajo
  - SCS imajo uporabniški vmesnik
  - Preferirajo integracijo na ravni uporabniškega vmesnika