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
  - intefejsi kojima se može pristupiti sa udaljene lokacije su prethodno bili posledica razvoja komponenti aplikacije (pozivanje metoda na udaljenim objektima - **R**emote **P**rocedure **C**all)
- SOA menja paradigmu, sistemi se osmišljavaju tako da se aplikacija “sklapa” od modula koji obavljaju jasno definisane usluge, i čine ih dostupne preko javnog intefejsa.
- 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 modulia, 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 pristupnuu 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 zamenjena 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.)

Enabled by Web services	*Technology neutral*	Endpoint platform independence.
	Standardized	Standards-based protocols.
	Consumable	Enabling automated discovery and usage.
Enabled by SOA	*Reusable*	Use of Service, not reuse by copying of code/implementation.
	Abstracted	Service is abstracted from the implementation.
	Published	Precise, published specification functionality of service interface, not implementation.
	Formal	Formal contract between endpoints places obligations on provider and consumer.
	Relevant	Functionality presented at a granularity recognized by the user as a meaningful service.

* The above table is from Microsoft's website [1]

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 funckije 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, registar 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 šablone
  - šabloni arhitekture

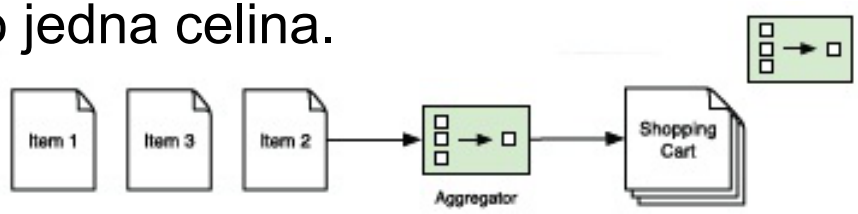
    - predstavljaju šablone karakteristične za određena dizajnerska rešenja u vezi sa implementacijom SOA arhitektura
  - kombinovani šabloni

    - 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 šablona preuzeti sa: https://dzone.com/refcardz/soa-patterns

SOA šabloni - osnovni šabloni

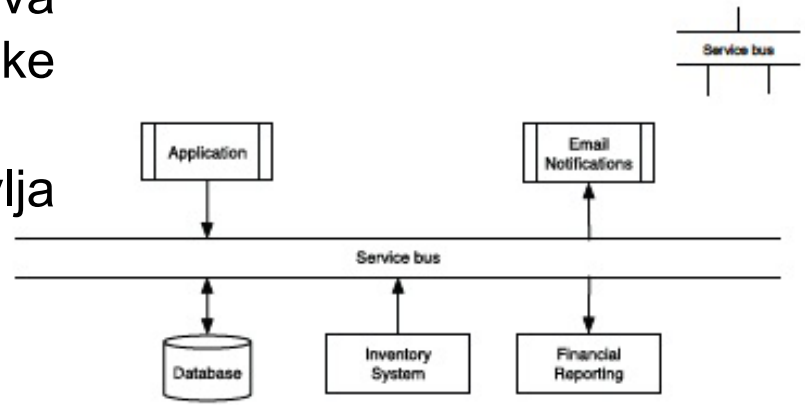
- **Agregator** - kombinuje više individualnih poruka kako bi se obrađivale kao jedna celina.



  - **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 agregator koji prima tok podataka i grupiše određene poruke kako bi ih poslao na dalju obradu kao jednu celinu. Očigledno je da je logika agregatora 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

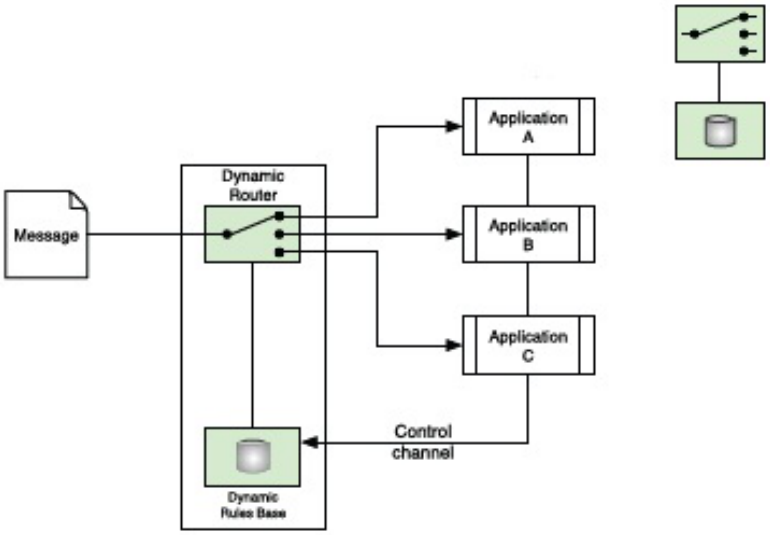
- **Service Bus** - servisna magistrala - komunikacioni kanal kojim se omogućava da jedna emisiona tačka proizvodi poruke koje mogu da "pokupe" jedna ili više izlaznih tačaka. Moguće je i da se obavlja obrada poruka dok prolaze kroz komunikacioni kanal.



  - **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 aplikacija nezavisno od njihove tehnologije.
  - **primena** - za integrisanje 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

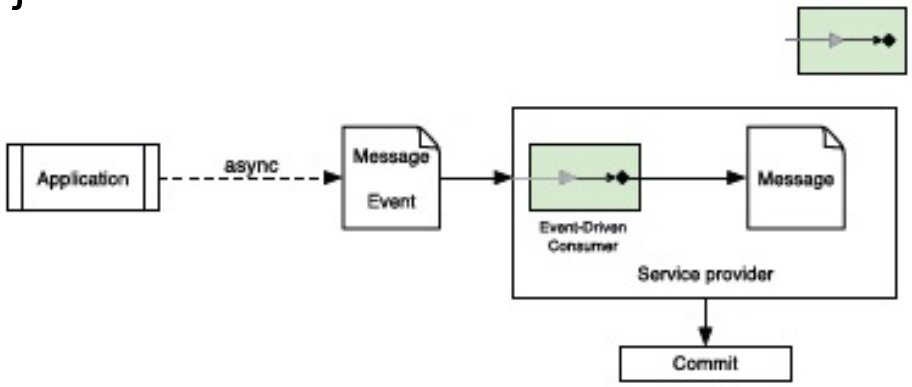
- **Dynamic Routing** - Mehanizam za efikasno prosleđivanje poruka na jednu ili više destinacija na osnovu konfigurabilnih nefiltrirajućih pravila koja se primenjuju na sadržaj poruka.



- **problem** - rutiranje poruka na osnovu filtera je neefikasno jer se poruke prosleđuju do svake destinacije i rutera da bi se tamo filtrirale i proverile, bez obira 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

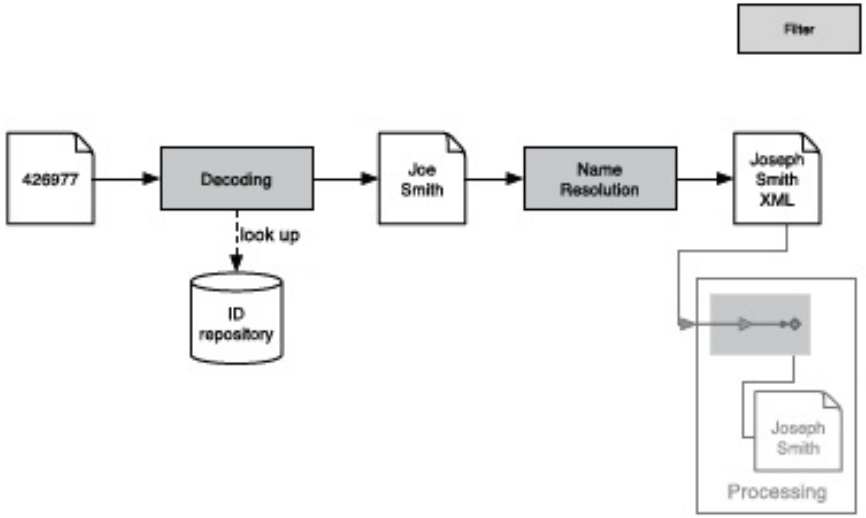
- **Event-Driven Consumer** - okruženje u kome se poruke odmah dostave određenim *service providerima* u momentu kada se pojave na komunikacionom kanalu.



- **problem** - Sistemi za razmenu poruka bazirani na blokirajućim listenerima ili prozivanju (*polling)* koriste bezrazložno određene 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 zavisan 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 oslobađaju. Bolja je iskorišćenost procesnih resursa.

SOA šabloni - osnovni šabloni

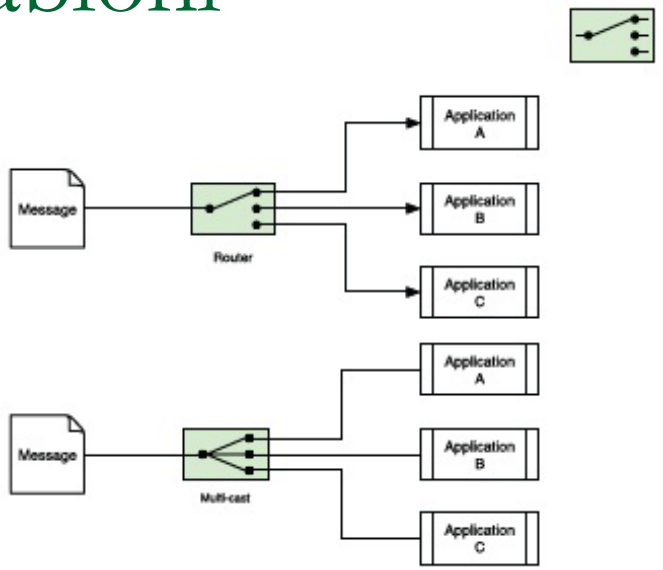
- **Filter** - deo prenosnog puta koji po određenim pravilima uklanja delove poruka ili primenjuje funkciju obrade dok poruke prolaze od izvora ka destinaciji.



- **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 primenjeni 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

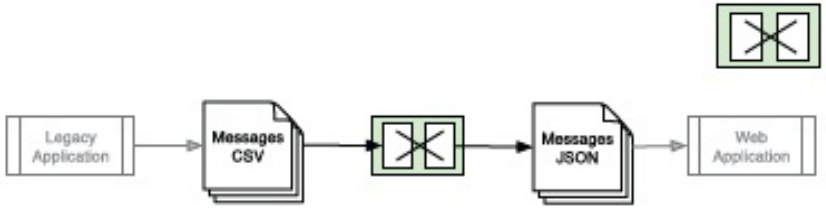
- **Router** - uopšteni mehanizam za prosleđivanje poruka do jednog ili više korisnika na osnovu konfigurabilnih pravila ili filtera koji se primenjuju na sadržaj poruke



- **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 sekvencijalno (*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

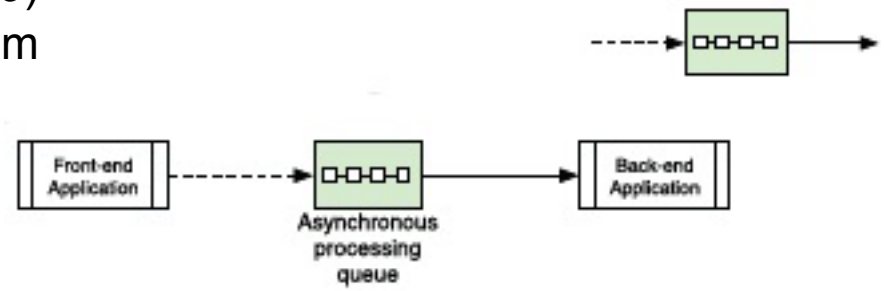
- **Translator / Transformer -**
  mehanizam za konvertovanje
  sadržaja poruka iz jedne reprezentcije
  u neku drugu tokom njenog prolaska
  kroz komunikacioni sistem



- **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

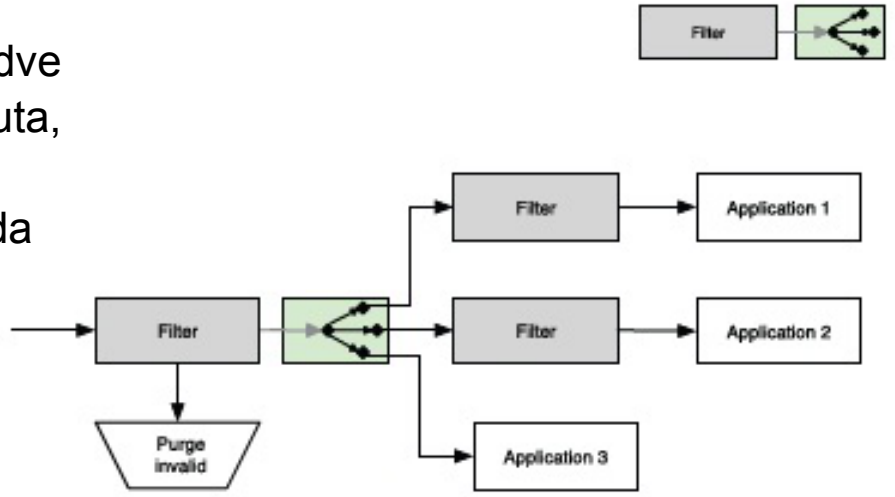
- **Asynchronous Processing** - mehanizam formiranja **reda poruka** (*message queue*) između jedne ili više krajnjih tačaka. Ovim se "rasparuje" vreme procesiranja i korišćenje resursa na obe strane.



- **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

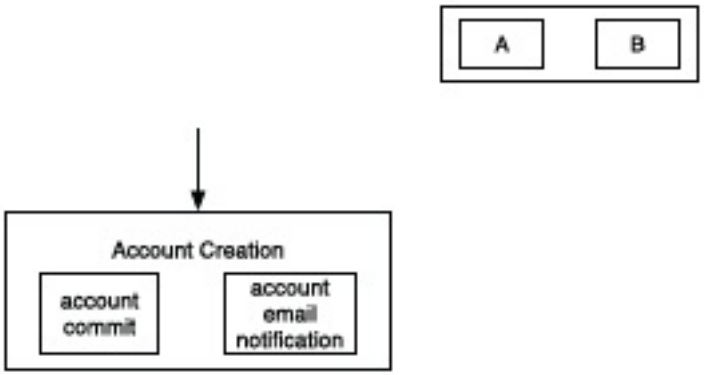
- **Bridge** - (most) mehanizam povezivanja dve ili više aplikacija preko istog prenosnog puta, ali kada one koriste različite protokole, a moguće je da je potrebna i analiza i obrada poruka dok prolaze kroz komunikacioni kanal.



  - **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 krajnjih 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

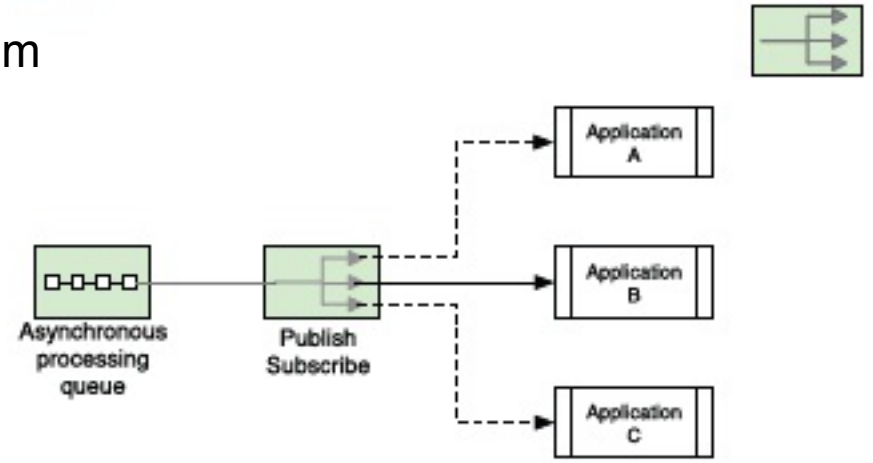
- **Cross-Service Operation** - mehanizam koji obezbeđuje koordinaciju više aktivnosti koje se sastoje od više servisa, pri čemu se garantuje završetak i/ili roll-back.



- **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 komponenete “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

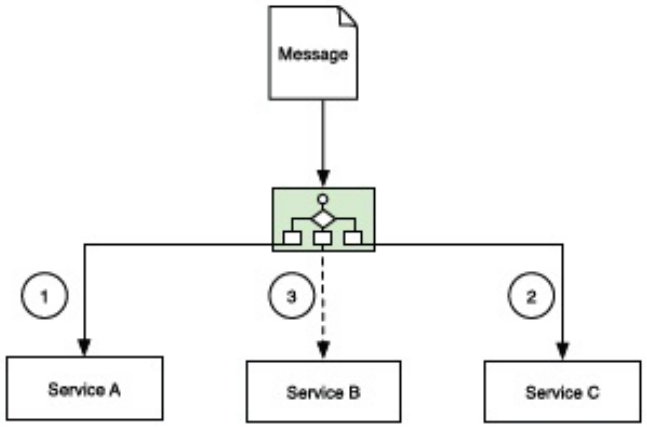
- **Event-Driven Dispatching** - mehanizam koji obezbeđuje rutiranje poruka konzumentima na osnovu reakcije na odrešene događaje koje su proizvele aplikacije koje su deo SOA sistema.



- **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 pretplate 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

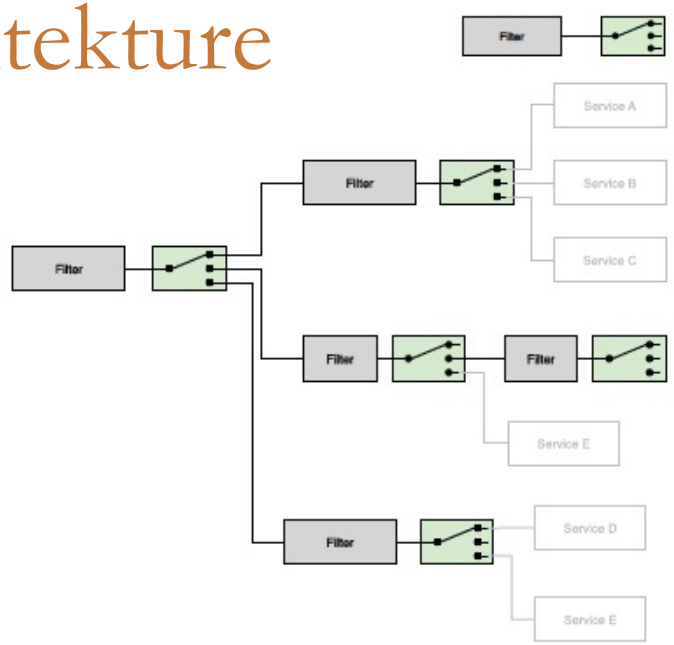
- **Process Aggregation** - metod za kombinovanje dva ili više nesekvencijalna, međusobno zavisna koraka u izvršavanju poslovnih procesa.



- **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 transakciona kompletnost).
- **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 transakcioni, ili imaju neku kombinaciju transakcionih i netransakcionih komponenti.
- **rezultat** - agregacija procesa obezbeđuje veliku fleksibilnost, ali je dosta teška za realizaciju - neophodno je stoga rastaviti složen sistem na manje klastere 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

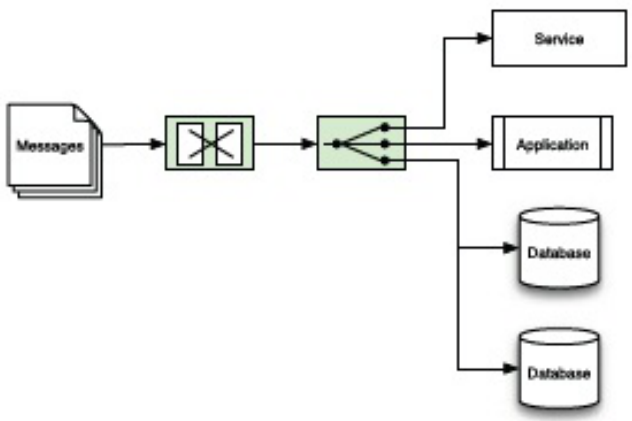
- **Routing and Filtering -** formalni mehanizam za rutiranje poruka ka krajnjim tačkama aplikacija preko više tačaka.



- **problem** - neophodno je proslediti poruke različitim aplikacijama, pri čemu se filtiriranje obavlja na osnovu sadržaja poruka, atributa, protokola ili svega ovoga zajedno.
- **rešenje** - formirati formalni mehanizam za rutiranje poruka rekurzivnom upotrebom filtera i rutera.
- **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 rutera (njihovu hijerarhiju) kako bi se izbeglo nepotrebno uparivanje aplikacija ili ciklično rutiranje.

SOA šabloni - šabloni arhitekture

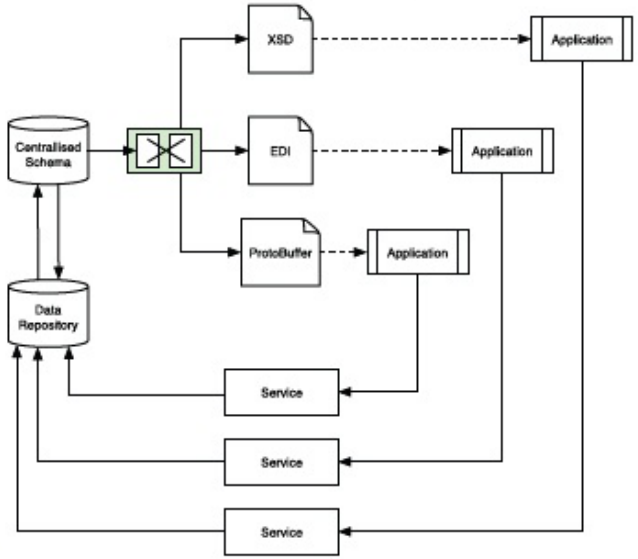
- **Replicator** - poruke ili njihov informacioni sadržaj se repliciraju na više *endpointa* sa identičnom konfiguracijom.



- **problem** - raspareni (*decoupled*), horizontalno skalabilni servisi se “zaglavljuju” na uskom grlu prouzrokovanom 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

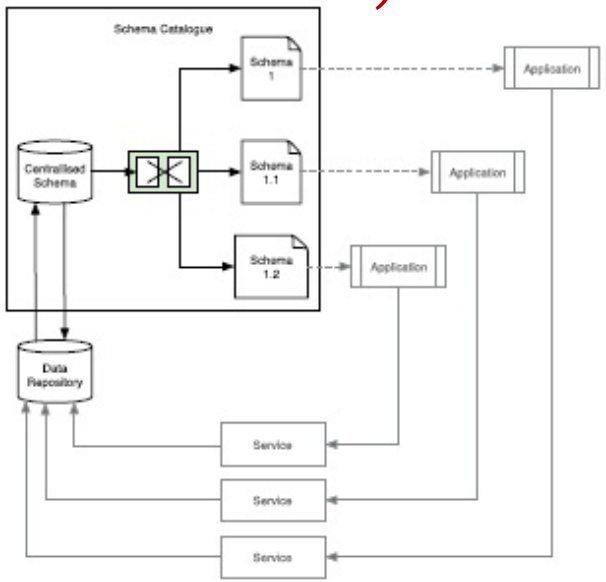
- **Centralized Schema** - definiše načine i uslove pod kojima je moguće deliti šeme podataka preko granica pojedinih aplikacija kako bi se izbegla redundantna reprezentacija podataka (i servisa).



- **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 donesu 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

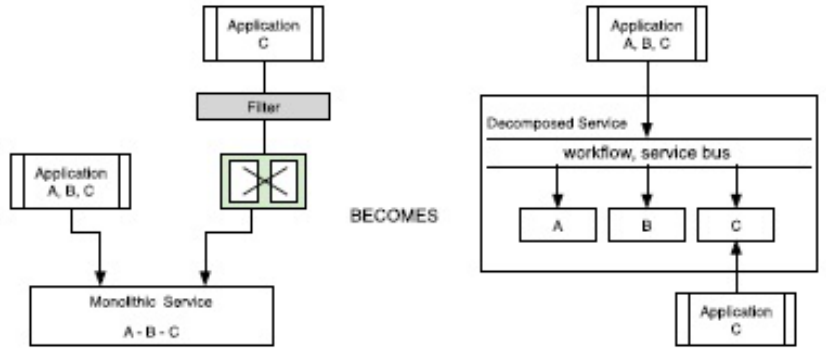
- **Concurrent Contracts** - metod koji omogućava da više različitih klijenata (konzumenata) sa različitim apstakcijam ili implementacijama pristupnih tačaka mogu simultano koristiti isti servis.



- **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

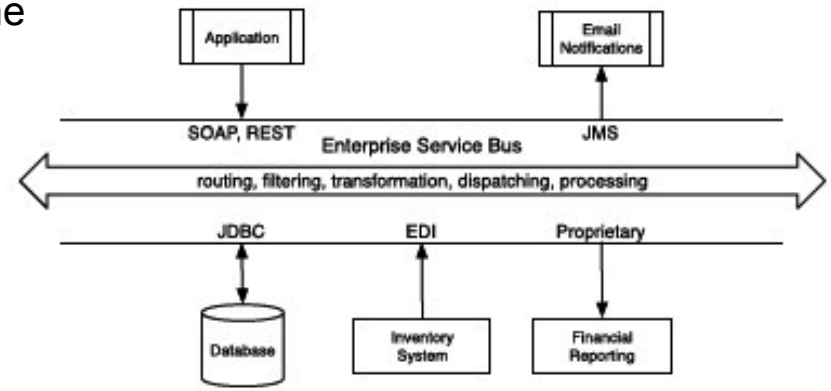
- **Decompose Capability** - "mogućnost dekomponovanja" - način da se sistem (re)dizajnira tako da se izbegne nepovoljan uticaj moguće naknadne funkcionalne dekompozicije do koje može doći ako sistem "naraste" ili se desi promena poslovnih zahteva.



- **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

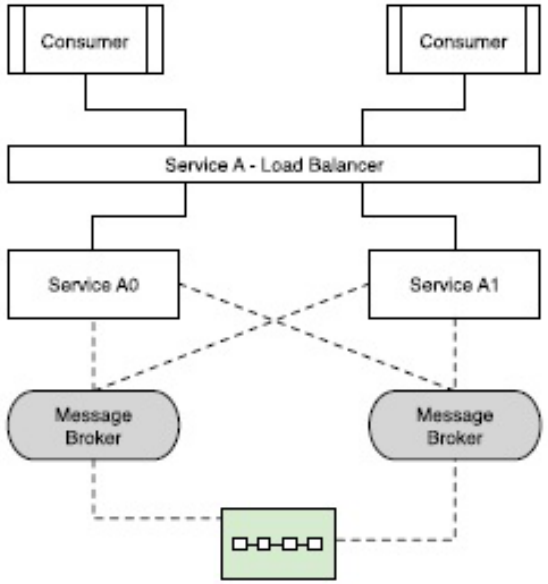
- **Enterprise Service Bus** - komunikacioni kanal koji obezbeđuje isporuku poruke od jedne ulazne tačke do jedne ili više izaznih tačaka, obezbeđujući pri tome upravljanje protokolima, filtriranje poruka, tranformacije i rutiranje i eventualno dodatno procesiranje poruka



- **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 definisanom 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

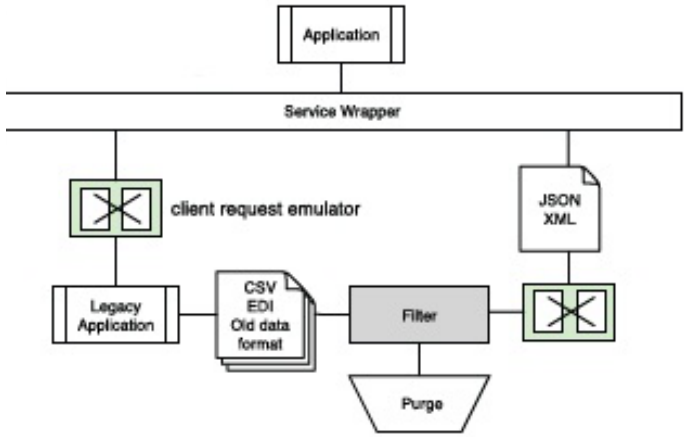
- **Fault-Tolerant Service Provider** - mehanizam za implementaciju servisnih platformi tako da se postigne skoro nulto vreme u kome je servis nedostupan u slučaju otkaza servisa ili cele platfrome.



- **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 *statelss 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 zastarelim *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* funkcionlanosti.

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 pojednostavljeno 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?

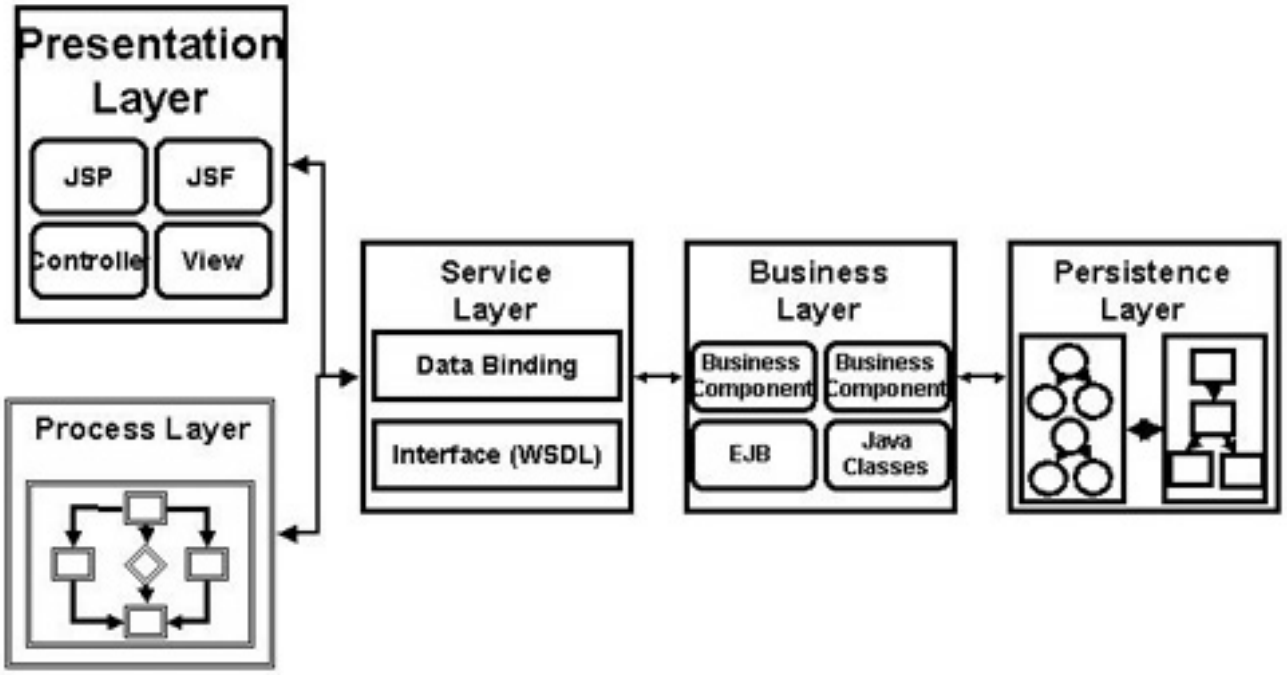
- Neophodno je razumeti poslovni model
- Fokus je na interoperabilnosti (komponenti)
- Fokus na poslovnoj agilnosti
- Definisati i sprovesti politiku aplikativne interoperabilnosti
  - Kako će komponente (web servisi) biti korišćeni?
  - Koji standarde koristiti i koju politiku interoperabinosti sprovesti?
  - 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 komponenete 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 koje 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 palikacija 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 pretpostavlja 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.