Kvalifikacijski doktorski ispit

Nadzor kvalitete usluge u Servisno orijentiranim aplikacijama

Goran Trlin

Fakultet elektrotehnike, strojarstva i brodogradnje u Splitu

Rujan 2014.

Sadržaj

1. Uvod

U ovom radu dan je pregled stanja istraživanja na području nadzora kvalitete usluge (engl. QoS) kod servisno orijentiranih (engl. Service Oriented). Trenutno stanje istraživanja detaljno je izloženo, te su identificirani najvažniji izazovi. Pod najvažnijim izazovima ovdje podrazumijevamo one probleme čije bi rješavanje, ili barem korak prema rješavanju, bitno pridonio poboljšanju razine usluge Servisno orijentiranih aplikacija (SOA).

Koncept Servisno orijentirane arhitekture je posljednji evolucijski korak razvoja arhitektura informacijskih sustava. Prema brojnim istraživanjima i analizama, većina aplikacija budućnosti bit će realizirana kroz servisno orijentiranu paradigmu. Servisno orijentirana arhitektura podrazumijeva informacijski sustav sastavljen od zasebnih funkcionalnih cjelina koje se nazivaju servisi. Svaki servis pruža određenu funkcionalnost i može biti programski pozvan od strane klijenta. Interakcija klijenata i servisa definira ponašanje SOA aplikacije. Servisno orijentirana arhitektura pruža brojne prednosti prilikom dizajna i implementacije sustava, kao što će biti predstavljeno u ovom radu.

U posljednjem desetljeću primjetna je sveopća raširenost World Wide Web-a kao dominantnog medija za prijenos informacija. U Republici Hrvatskoj, kao i u ostatku svijeta, svakodnevno svjedočimo puštanju u pogon novih IT sustava baziranih na web tehnologijama, kao što su eZdravstvo, eUpisi na fakultete i slično. Sve većim prelaskom na digitalne tehnologije u servisima državne uprave, te privatnim i javnim poduzećima, svaki pripadnik modernog društva na ovaj ili onaj način postaje ovisan o pouzdanom radu web baziranih i Servisno orijentiranih aplikacija. Servisno orijentirane aplikacije prirodno su povezane s web tehnologijama. Zbog svoje arhitekture, u kojoj aplikacije izvođene na web serveru imaju ulogu servisa, web aplikacije mogu se promatrati kao podskup Servisno orijentiranih aplikacija. Servisno orijentirane aplikacije implementirane putem web tehnologija nazivaju se Web servisi (engl. Web services) i predstavljaju značajan postotak ukupnog broja SOA aplikacija danas u upotrebi.

Kako Servisno orijentirane aplikacije u praksi postaju sve češće i sve više kompanija i organizacija o njima ovisi, u fokus istraživanja znanstvenika i inženjera dolazi nadzor i podizanje kvalitete usluge u takvim sustavima (engl. Quality of Service). Jedan od elementarnih aspekata kvalitete usluge je učinkovitost (engl. Performance). Učinkovitost aplikacije, kao što je objašnjeno u ostatku ovog teksta, može se mjeriti na različitim razinama sustava – od razine mrežne infrastrukture do aplikacijske razine. U posljednje vrijeme, zbog specifičnosti koje donosi računarstvo u oblaku (engl Cloud computing), a koje su opisane u ovom radu, sve više istraživača se fokusira na nadzor učinkovitosti na aplikacijskoj razini (engl. Application Level Monitoring).

2. Servisno orijentirane aplikacije

Postoji više različitih mogućih pogleda na Servisno orijentirane aplikacije. Može se reći da su Servisno orijentirane aplikacije nastale kao sljedeći evolucijski korak komponentno orijentiranih aplikacija. Komponentno orijentirane aplikacije dijele sličnu filozofiju kao i SOA. I jedna i druga arhitektura nastoji razdijeliti aplikacije u manje, međusobno što manje ovisne dijelove. Kod komponentno orijentiranih aplikacija postoje brojne specijalizirane platforme i tehnologije kojima se to postiže. Primjerice, Microsoft DCOM ili CORBA. Kod Servisno orijentiranih aplikacija, naglasak je na otvorenim, tekstualnim protokolima komunikacije, kakvi su XML i JSON. Ovakvi protokoli pogodni su za komunikaciju između servisa i klijenata pisanih u najrazličitijim vrstama programskih jezika i pokretanih različitim tipovima operacijskih sustava.

Gledano iz drugom ugla, Servisno orijentirane aplikacije su pogodne za programsko upravljanje i automatizaciju. Stoga je moguće smatrati SOA arhitekturu prirodnim evolucijskim korakom u izgradnji heterogenih višeagentskih sustava. Neovisno u kutu gledanja na SOA aplikacije, u osnovi SOA sustav se temelji na standardnim i dobro definiranim elementima. Osnovni elementi svake Servisno orijentirane aplikacije su:

-Servisi

-Sučelja (ugovori) servisa

-Klijenti

-Komunikacijski protokoli

Servisi su funkcionalne jedinke SOA sustava zadužene za realizaciju određene funkcionalnosti. Servisu klijenti pristupaju preko odgovarajućeg sučelja (engl. Interface). Sučelje može biti definirano u nekom deklarativnom jeziku visoke razine, kao što je WSDL (engl.Web Service Definition Language)[[1].]. Sučelje ne mora biti javno dostupno, niti defnirano u nekom jeziku za opis sučelja. Međutim, u takvom slučaku, klijent mora poznavati sučelje servisa prije korištenja servisa.

U samoj prirodi SOA aplikacija nalazi se raspodijeljenost. Ona može biti samo logička ali najčešće je i fizička, odnosno prostorna. Naime, različiti servisi mogu biti geografski raspodijeljeni po cijelom kontinentu ili šire. Shodno tome, jedna od presudnih stvari za kvalitetu rada servisno orijentiranog sustava je izbor komunikacijskog protokola. U praksi, klijenti i servisi za razmjenu poruka obično koriste standardne komunikacijske protokole kao što su TCP i HTTP. Protokol UDP ne pruža metode osiguravanja isporuke pojedinog podatkovnog paketa (engl. Datagram), pa se stoga rijetko koristi u SOA aplikacijama.

SOA aplikacije temelje se na jednostavnoj, ali veoma značajnoj premisi – da je jedini način za očuvanje fleksibilnosti velikog distribuiranog sustava uvođenjem decentralizacije i oslanjanjem na svojstvene prednosti svakog modula [2].

2.1. Razvoj servisne orijentacije

Razdvajanje složenih aplikacija u skup modula specijaliziranih za pojedinu funkcionalnost praksa je prisutna u računarstvu još od pojave prvih programskih jezika visoke razine u osamdesetim godinama prošlog stoljeća. Tijekom godina, objektno orijentirani jezici su nudili različite tehnologije za razdvajanje funkcionalnosti programa u komponente. Komponentno orijentirani pristupi uključuju tehnologije kao što su CORBA i Microsoftov COM [3]. Servisno orijentirane aplikacije u istraživačkoj literaturi javljaju se negdje nakon 2004. godine i pojave Microsoftove tehnologije WCF (engl. Windows Communication Foundation). Servisno orijentirane aplikacije možemo promatrati kao prirodan slijed razvoja komponentne orijentirane arhitekture. U istraživačkoj literaturi postoje i rasprave o tome koliko se Servisno orijentirana arhitektura uopće razlikuje od komponentno orijentirane arhitekture [4]. Servisno orijentirana arhitektura ipak, za razliku od komponente arhitekture, u fokus stavlja heterogenost i komunikaciju razmjenom poruka. Heterogeni sustavi koji komuniciraju porukama ne moraju nužno imati klasnu ili komponentnu dimenziju. Iz navedenoga se može zaključiti da je SOA arhitektura širok i općenit pojam, koji se donekle preklapa s nekim drugim arhitekturama i pristupima razvoju softvera. Isto tako, očigledno je da trend kretanja razvoja softvera prema SOA arhitekturi postoji odavno, te da svakom godinom jača sve više.

2.2. Primjene Servisno orijentiranih aplikacija

Svaka aplikacija koja se sastoji od više logičkih ili funkcionalnih cjelina može se napisati kao SOA aplikacija. U praksi su poznate upotrebe SOA arhitekture pri izgradnji najrazličitijih softverskih sustava. Posebno su česte primjene servisno orijentiranih arhitektura u izgradnji velikih poslovnih sustava.

Takvi sustavi imaju potrebu za pouzdanim upravljanjem distribuiranim transakcijama. Transakcije u ovakvim slučajevima podrazumijevaju koordinaciju stanja geografski udaljenih sustava, često implementiranih u različitim tehnologijama. Popularna rješenja za ovakve probleme u praksi uključuju korištenje specijaliziranih, složenih tehnologija, kao što su Microsoft Windows Communication Foundation (WCF) u Windows svijetu ili Enterprise Java Beans u Java svijetu.

Osim navedenih tehnologija, u složenim poslovnim sustavima susreću se i brojni dodatni alati više razine. Jedan takav standardizirani dio složenog SOA sustava je i

Enterprise Service Bus (ESB). ESB je softverski koordinator velikih heterogenih poslovnih sustava koji olakšava izvođenje promjena i povećava fleksibilnost cjelokupne Servisno orijentirane aplikacije. ESB softver u mnogome služi integraciji i konsolidaciji poslovnih resursa u kompaniji, za što postoji i standardan engleski naziv - Enterprise Application Integration (EAI). Sve navedene tehnologije za razmjenu podataka koriste standardizirani format naziva SOAP (engl. Simple Object Access Protocol), baziran na XML sintaksi. SOAP je postao popularan zbog svoje baziranosti na XML formatu, koji je moguće napisati i pročitati na svim poznatim platformama. Međutim, SOAP kao ima i neke bitne nedostatke. Jedan takav je njegova složenost – svaka SOAP poruka ima precizno definirano zaglavlje, imeničke prostore i brojne druge, često nepotrebne, dodatne informacije.

Upravo opisane sustave mogli bismo nazvati složenim, ceremonijalnim i u mnogim situacijama suviše zahtjevnim za implementaciju i održavanje. Iz tog razloga, u posljednje vrijeme u SOA svijetu sve više jača alternativa SOAP baziranim protokolima u vidu jednostavnih servisa baziranih na REST (engl. REpresentational State Transfer) principima. Ovaj pristup prakticira jednostavnost implementacije, korištenja i održavanja što je uvelike prihvatljivo većini kompanija. Ne postoje striktni REST protokoli, nego je na programerima da odaberu najpogodniji format razmjene za pojedinu primjenu. Primjeri ovakvih sustava su SOA sustavi bazirani na web servisima čiji su klijenti mobilne aplikacije na iOS i Android operativnim sustavima. U REST svijetu transakcije nisu u prvom planu i one se izvode na višim razinama programske logike, umjesto na razini infrastrukture. Važno je primijetiti da je ovakav tip softvera pogodan za razvoj u malim timova što implicira mogućnost razvoja bez pretjeranih ulaganja u programerske resurse.

Servisno orijentirana filozofija nije ograničena na poslovne aplikacije. Ona je zapravo jako bliska filozofiji dizajna i razvoja višeagentskih sustava. U pojednostavljenoj verziji, jedan servis bi se mogao predstaviti kao jedan agent u heterogenom višeagentskom sustavu. Taj isti agent može slati poruke drugim agentima, a to u SOA svijetu odgovara slanju poruka od klijenta prema serveru. Primanje poruke od strane drugog agenta može se prikazati kao pozivi funkciji servisa u SOA aplikaciji. Već iz ovoga može se izvući zaključak da su višeagentski sustavi i SOA aplikacije srodni pojmovi i u nekom kontekstu mogu se čak i poistovjetiti.

Automatizacija je proces kojem svjedočimo u zadnjih nekoliko desetljeća. Težnja za automatizacijom odvela je znanstvenike prema istraživanju umjetne inteligencije i robotike. Navedeni sustavi su izuzetno kompleksni i sastoje se od mnoštva različitih međuovisnih funkcija. Upravo ta kompleksnost se može barem donekle razdijeliti korištenjem servisne orijentacije.

Iz svega navedenoga, očigledno je da servisna orijentacija nije samo trenutni hit u IT svijetu, nego ispravan i dugoročno isplativ način razmišljanja koji donosi brojne prednosti svim strana uključenim u implementaciju i održavanje informacijskih

sustava. Izvjesno je da će u budućnosti SOA aplikaciji biti sve rasprostranjenije i njihov nadzor će predstavljati još veći izazov IT industriji u cjelini.

2.3. Izazovi mobilnog računarstva (engl. Mobile computing)

Povećanjem broja tableta i pametnih telefona dostupnih na tržištu, mobilni uređaji se sve češće pojavljuju kao klijenti SOA aplikacija [5]. Dok krajnji korisnici očekuju visoku kvalitetu usluge, ona u praksi nije uvijek moguća. Naime, mobilni uređaji suočavaju se sa specifičnim izazovima, kao što su nedostupnost podatkovne veze ili smanjena propusnosti. Ovakvi problemi odavno su identificirani [6], ali kvalitetnih rješenja za ove probleme i dalje nedostaje. Osim izravnog negativnog utjecaja na zadovoljstvo krajnjih korisnika, loša ili nepostojeća podatkovna veza otežava i nadzor mobilnih klijenata. Iz navedenog razloga podatke o učinkovitosti i stanju mobilnih klijenata nije uvijek moguće dostaviti u realnom vremenu.

Upravo navedena činjenica upućuje na potrebu za razvojem novih, adaptivnih pristupa nadzoru mobilnih klijenata. Novi pristupi morali bi se fokusirati na ograničene i često nedostupne mrežne resurse kod ovakvih uređaja, te ponuditi adaptivne načine rada u cilju što brže reakcije na problem. U istraživačkom svijetu postoje apstraktni komunikacijski modeli koji opisuju komunikaciju u najširem mogućem smislu. Jedan takav model je predstavljen u [7]. Navedeni apstraktni model opisuje komunikaciju na razini uređaja i usmjernika (engl. Router). Pomoću modela moguće je opisati procese više razine, provesti simulacije i verificirati različite protokole. U SOA svijetu još ne postoji apstraktni model koji omogućio iste ove operacije nad Servisno orijentiranim aplikacijama, a koji bi uključivao specifičnosti mobilnih klijenata.

2.4. Izazovi računarstva u oblaku (engl. Cloud computing)

Računarstvo u oblaku (engl. Cloud computing) u najkraćem predstavlja tendenciju premještanja računarskih resursa s osobnih računala i računala instaliranih u kompanijama (engl. On premises computing) u velike podatkovne centre (engl. data centers). Na ovaj način, krajnji korisnik je oslobođen troškova upravljanja i održavanja hardverskih resursa sustava, koje za njega obavlja pružatelj Cloud usluga. Cloud resursi su posebno pogodni za tvrtke budući da se naplata usluga korištenja Cloud resursa najčešće vrši kao mjesečna rata na osnovu realne potrošnje (engl. „Pay as you go“) [8]. U praksi možemo razlikovati tri osnovne vrste Cloud ponuditelja usluga [9]:

IaaS – Infrastructure as a Service

Podrazumijeva iznajmljivanje čistih računarskih resursa kao što su procesorska snaga ili disk prostor. Najčešće se implementira kao sustav za unajmljivanje virtualnih strojeva (engl. Virtual Machines). Primjer IaaS usluge je Amazon EC2 [10].

PaaS – Platform as a Service

Osim samih računarskih resursa, PaaS uključuje i softversko okruženje pogodno za brzi razvoj korisničkih aplikacija prilagođenih izvršavanju u oblaku. Primjer PaaS usluge je Windows Azure.

SaaS – Software as a Service

SaaS usluga podrazumijeva ponudu gotovog softvera koji se izvršava u oblaku, a krajnji korisnik mu najčešće pristupa preko web sučelja ili nekog sličnog tankog (engl.thin) klijenta. Primjeri SaaS usluga su Yahoo Mail, GMail, Salesforce.

Budući da se račun za Cloud usluge formira na temelju potrošnje Cloud resursa, od velike je važnosti ponuditi potpunu kontrolu nad aplikacijama koje su instalirane u oblaku. Detekcija svakog prekomjernog ili nepravilnog korištenja tvrdog diska, memorije ili prometa u oblaku može donijeti značajne uštede poslovnim korisnicima. U praksi, pružatelji usluga često nude vlastiti modul za provjeru potrošnje [11], međutim za objektivno i nezavisno mjerenje bilo bi potrebno imati unificiran mjerni sustav, odvojen od sustava pružatelja usluga. Osim toga, mnoge SOA aplikacije uključuju servise poslužene od strane različitih Cloud pružatelja usluga. Efikasan nadzor takvih složenih sustava zahtjeva rješenja neovisna o platformi pojedinog davatelja usluga.

2.5. Razine istraživanja u SOA svijetu

Servisno orijentirane aplikacije su u fokusu brojnih istraživanja u svijetu. Istraživanja se ugrubo mogu podijeliti na tri razine [12]:

- **Osnovna razina**

  Na ovoj razini istraživači se bave elementarnim dijelovima SOA sustava, kao što su ugovori, servisi, mrežni komunikacijski protokoli i slično. Istraživanja osnovne razine predstavljaju temelj za izgradnju svih vrsta SOA sustava.

- **Razina kompozicije**

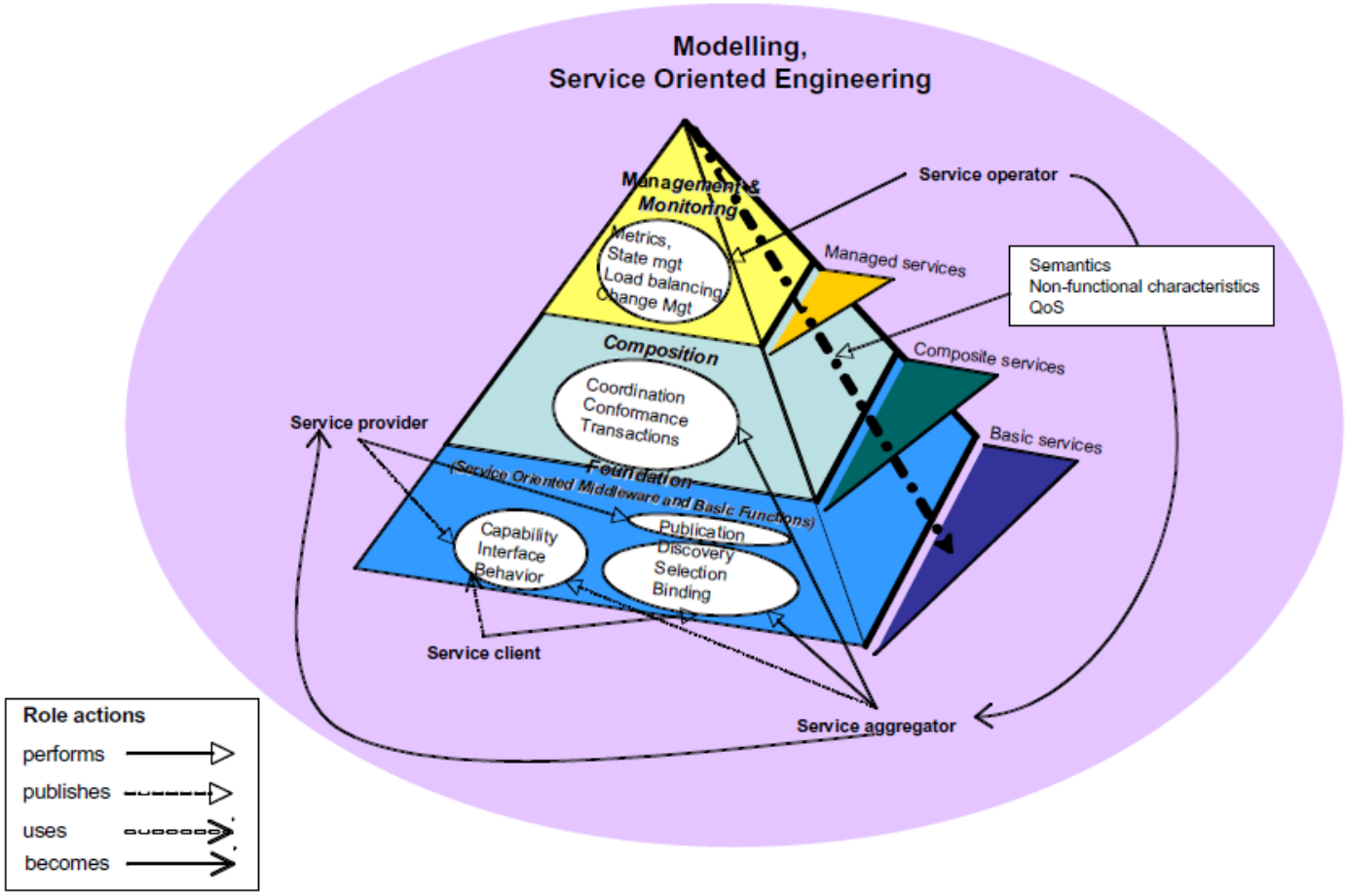
  SOA aplikacije se uvijek sastoje od više servisa i klijenata. Razina kompozicije je istraživačko područje u kojem se nastoji odgovoriti na izazove povezivanja više servisa u složene distribuirane aplikacije. Na ovoj razini istraživači se bave definiranjem i nadzorom kvalitete usluge, uspostavom protokola za izvođenje distribuiranih transakcija i brojnim drugim, u praksi važnim, problemima. Ovo je ujedno i razina na kojoj postoje brojni izazovi. Jedan takav izazov je i definiranje unificiranog apstraktnog modela za nadzor kvalitete usluge u realnom vremenu koji bi uključivao i SOA i web 2.0 aplikacije. Takav model bi trebao zadati univerzalnu strukturu mjernog sustava za nadzor učinkovitosti i kvalitete usluge u SOA svijetu, te bi trebao pružiti mogućnost daljnjeg proširivanja u vidu izgradnje DSL (engl. Domain Specific Language) za upravljanje nadzorom u realnom vremenu. Kvaliteta usluge može se deklarativno definirati u obliku jezika. Jezici za definiranje kvalitete usluge, kao što su QML [13] i HQML, također su zastarjeli i neprilagođeni modernim SOA primjenama. Definicija jednostavnog i univerzalnog QoS jezika je svakako jedan od izazova koji leži pred znanstvenom zajednicom u budućnosti.

- **Razina upravljanja**

  Složenost topologije i ponašanja servisno orijentiranih sustava zahtjeva napredne mehanizme upravljanja. Servisi se mogu dinamički pojavljivati i nestajati. Svaki servis može također postati i klijent nekom drugog servisa. Takve promjene izazivaju promjenu topologije sustava, što predstavlja izazov svim sustavima za nadzor SOA aplikacija. Neka istraživanja prikazuju interakciju servisa putem Petrijevih mreža [14], dok drugi koriste ubacivanje i praćenje pojedinih elemenata kroz cijeli graf SOA aplikacije (engl. dope and trace) [15].

  Razina upravljanja može se promatrati zajedno s razinom kompozicije, budući da se one jednim dijelom preklapaju. Primjerice, područje

nadzora kvalitete usluge primarno se nalazi na razini kompozicije. Međutim, u praksi je nemoguće dizajnirati i implementirati kvalitetan sustav za nadzor kvalitete usluge bez razmatranja problema kojima se bavi razina upravljanja. Takvi problemi uključuju promjenu topologije za vrijeme izvođenja, povećanje broja servisa, varijaciju opterećenja i mnoge druge važne fenomene.



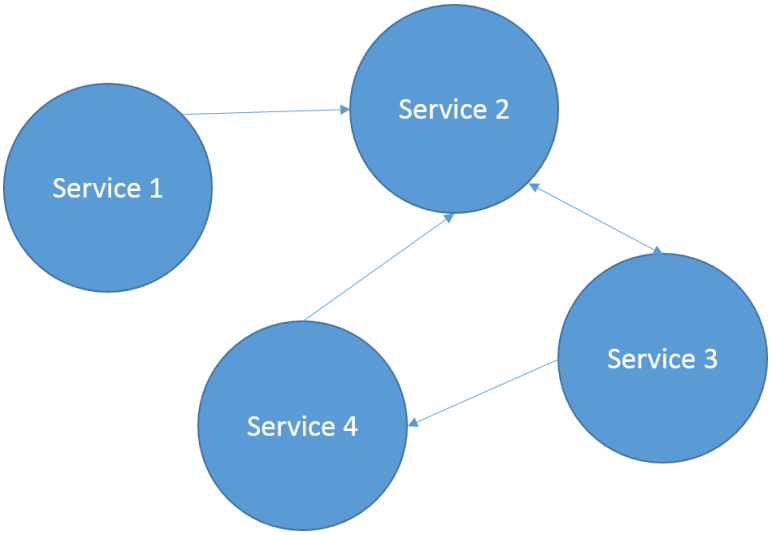
Slika 1 – Razine istraživanja u SOA svijetu [12]

Brojna istraživanja uključuju sve razine ove istraživačke piramide [16] [17] . Navedena temeljna SOA istraživanja mogu se povezati i s nekim drugim istraživanjima, primjerice iz teorije igara ili paralelnog računanja i time stvoriti nove istraživačke prostore i razine. Neka rubna područja SOA istraživanja dana su u zadnjem poglavlju ovog rada.

2.6. Osnovni arhitekturalni obrasci komuniciranja (engl. Communication patterns) u SOA arhitekturi

2.6.1. Od točke do točke (engl. Point-to-Point)

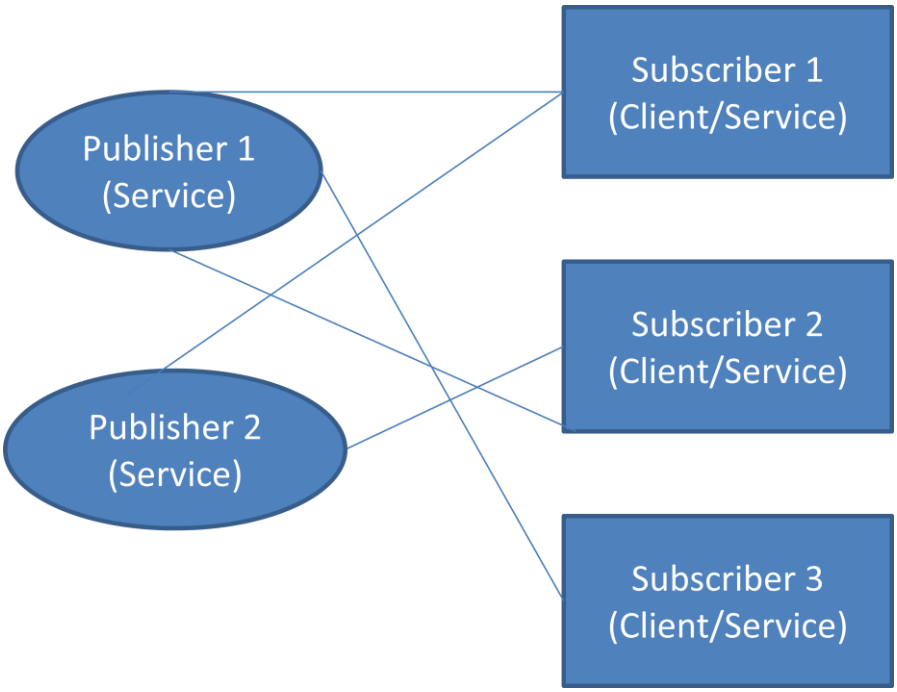
Arhitekturni obrazac od točke do točke odnosi se na SOA sustave u kojima klijent C i servis S komuniciraju izravno razmjenom poruka [18]. Između klijenta i servisa uspostavlja se komunikacijski kanal baziran na jednom od standardnim komunikacijskih protokola (HTTP, TCP, Named Pipes,…). Komunikacija može biti jednosmjerna (engl. one-way) ili dvosmjerna (engl.two-way). Kao što je prikazano na slici ispod, kod ovog obrasca servisi mogu imati i ulogu klijenta prema drugom servisu. Ovaj obrazac ne podrazumijeva centralno čvorište, nego se komunikacija ostvaruje izravnim slanjem poruka između određena dva servisa u sustavu.



Slika 2 - Point To Point obrazac

2.6.2 Objava - pretplata (engl. Publish - subscribe)

Kao što je prikazano na Slika 3, Objava – pretplata [18] podrazumijeva skup od N klijenata i M servisa. Klijenti su zainteresirani za određene poruke pojedinih servisa. Servisi nude mogućnost pretplate na određen događaje. Svaki klijent zainteresiran C zainteresiran za događaj E na servisu S mora se preplatiti kako bi bio obaviješten o tom događaju. Na ovaj način, servis S na događaju E uvijek ima X prijavljenih pretplatnika. Generiranjem događaja E svaki od tih pretplatnika dobit će obavijest (engl. notification).



Slika 3 - Publish/Subscribe obrazac

3. Nadzor kvalitete usluge SOA sustava

Kvaliteta usluge (engl. Quality of Service) je širok pojam u svijetu softvera. Kada je riječ o Servisno orijentiranim aplikacijama, autori najčešće pišu o mjerama pouzdanosti i učinkovitosti [19] [20] kao o temeljnim parametrima kvalitete usluge. Kvaliteta usluge SOA aplikacije ovisi o:

1. Učinkovitosti (engl. Performance monitoring)
2. Pouzdanosti / raspoloživosti (engl. System availability)
3. Nadzoru i upravljanju pogreškama i iznimkama (engl. Exception / error handling)
4. Nadzoru i upravljanju sigurnosnim mehanizmima i iznimkama (engl. Security management)
5. Pouzdanosti izvođenja distribuiranih transakcija (engl. Transaction management)
6. Nadzoru i upravljanju opterećenjem (engl. Load balancing)

Svaki od ovih segmenata predstavlja značajno područje istraživanja i u ovom radu je prikazano njegovo trenutno stanje razvoja. Kao što je prikazano na Slika 4, sva ova područja podjednako su važna u nadzoru ukupne kvalitete usluge SOA sustava.

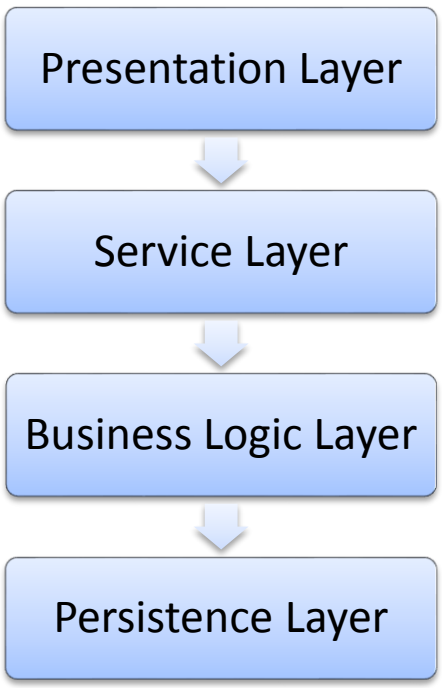
Slika 4 – Područja istraživanja u SOA svijetu

Moderne SOA aplikacije rade najčešće u Cloud okruženju. Cloud okruženja naplaćuju korištenje svojih hardverskih i softverskih resursa. Područje nadzora takvih sustava jedno je od ključnih područja za ispravno i kvalitetno iskorištavanje svih

resursa oblaka [21] i postizanje najviše moguće kvalitete usluge u Cloud SOA sustavima [22]. U radu [19] autori prezentiraju CSLA – jezik za definiranje SLA zahtjeve u Cloud okruženjima. Ovakav jezik nudi mogućnosti kvantifikacije minimalnih vrijednosti učinkovitosti koje pojedini servis mora ispuniti da bi ispunio zahtjeve klijenta. CSLA jezik je baziran na XML sintaksi.

3.1. Instrumentacija

Postupak umetanja mjernih sondi na odgovarajuća mjesta u nadziranoj aplikaciji naziva se instrumentacija i predstavlja jedan od najvećih izazova u izgradnji nadzornog sustava [23]. U praktičnom kontekstu SOA aplikacija, instrumentacija je pojam koji označava ubacivanje programskog koda za nadzor rada pojedinog servisa u SOA aplikaciji. Instrumentacija se može provoditi na različitim razinama nadzirane aplikacije.



Slika 5 - Slojevi arhitekture poslovne aplikacije

Kao što je prikazano na slici iznad, svaka se poslovna SOA aplikacija može podijeliti na četiri osnovna sloja:

-Prezentacijski sloj (engl. Presentation Layer)

Prezentacijski sloj obuhvaća svu funkcnionalnost zaduženu za prikaz podataka krajnjem korisniku. Primjerice, u slučaju web aplikacija, prezentacijski sloj se realizira tehnologijama HTML / CSS. Prezentacijski sloj

se u dobro dizajniranoj aplikaciji može naknadno mijenjati i praktično je neovisan o ostalim slojevima aplikacije.

-Servisni sloj (engl. Service Layer)

Servisni sloj zadužen je za komunikaciju sa ostalim aplikacijama putem razmjene poruka. Servisni sloj se može iskoristiti za automatizaciju same aplikacije. U kontekstu SOA aplikacija, ovaj sloj je najpogodniji za instrumentaciju jer na ovoj razini je moguć pristup svakoj poruci koju servis i klijent međusobno razmjenjuju.

-Sloj poslovne logike (engl. Business Logic Layer)

Sloj poslovne logike sadrži algoritme kojima aplikacija obavlja svoju temeljnu funkcionalnost sa stajališta krajnjeg korisnika. Ovaj sloj je zanimljiv u pogledu instrumentacije jer bi se njegovim nadzorom mogao dobiti potpuniji uvid u funkcioniranje sustava iz perspektive krajnjeg korisnika.

-Sloj pristupa bazi podataka (engl. Persistence Layer)

Gotovo sve poslovne aplikacije ovise o bazi podataka. Značajan dio vremena obrade kod modernih aplikacija otpada upravo na obradu upita relacijskim i objektnim bazama podataka. Instrumentacijom baze podataka mogao bi se dobiti uvid u eventualne probleme i uska grla aplikacije.

Instrumentacija se može implementirati na sljedeće načine:

Za vrijeme pisanja izvornog koda (engl. Source Code Instrumentation)

Podrazumijeva upisivanje dodatnog koda za mjerenje i nadzor u izvorni kod nadzirane aplikacije. Prednost ovog pristupa je u činjenici da se ovakva instrumentacija uvijek može implementirati. Problem ovakve instrumentacije je u pojavi jake sprege između nadzirane i mjerne aplikacije. Zbog toga, iznimka u mjernom dijelu aplikacije može poremetiti rad nadzirane aplikacije.

Modifikacijom komunikacijske biblioteke (engl. Library Modification)

Većina distribuiranih aplikacija za potrebe komunikacije koristi neku već gotovu programsku biblioteku ili komponentu. Da bi se postojećoj komponenti dodao instrumentacijski kod, potrebno je izmijeniti biblioteku implementacijom Adapter obrasca. To znači da u biblioteci treba pronaći originalnu funkciju f() i zamijeniti je proširenom verzijom f*(). Modifikacija komunikacijske biblioteke je složen zadatak koji u praksi može uzrokovati neželjene nuspojave.

Za vrijeme prevođenja (engl. Compile Time Instrumentation)

Jezici koji se izvode pomoću virtualnog stroja, poput Jave ili C#-a, pogodni su za instrumentaciju korištenjem tehnika Aspektno orijentiranog programiranja (engl. Aspect Oriented Programming). AOP tehnike omogućavaju odvajanje mjernog koda od koda aplikacije koja se nadzire. Za vrijeme prevođenja, točnije neposredno prije samog prevođenja, moguće je uključiti dodatni kod za mjerenje i nadzor u nadziranu aplikaciju. Ovakve tehnike su najčešće ograničene na Javu i slične programske jezike.

Za vrijeme izvođenja (engl. Runtime Instrumentation)

Instrumentacija se može obaviti i za vrijeme izvođenja aplikacije. Primjerice, korištenjem dinamičkim predstavnika (engl dynamic proxy) u Java programskog jeziku [24]. Instrumentacija za vrijeme izvođenja je fleksibilna, međutim zahtjeva dodatno računanje za vrijeme izvođenja i zahtjeva izgradnju odgovarajuće programske infrastrukture kako bi se mogla kvalitetno implementirati.

Na području instrumentacije postoji iznimno važno i veliko područje za buduća istraživanja. Naime, u praksi ne postoji apstraktni model za instrumentaciju servisnog sloja aplikacija koji bi pokrivao različite platforme i tehnologije. Postoje specifična rješenja, ovisna o pojedinoj platformi, kao što je WCF Extension Points tehnologija, koja omogućava kvalitetnu instrumentaciju u Windows baziranim aplikacijama. Dizajn apstraktnog modela za instrumentaciju servisnih aplikacija na različitim platformama zadatak je koji očekuje istraživačku zajednicu u nadolazećim godinama.

Dodatne informacije o stanju pojedinog servisa u aplikaciji mogu se dobiti instrumentacijom operacijskog sustava (engl. Deep diving). Deep diving metode pružaju uvid u stanje resursa operacijskog sustava kao što su:

-Dužina reda čekanja na CPU

-Zauzeće memorije

-Stanje diska

-Stanje mrežnog adaptera

Ove informacije, iako nesumnjivo korisne, postaju teško dostupne kod mjerenja u oblaku. Naime, davatelji usluga imaju potpunu kontrolu nad hardverom i operacijskim sustavom na kojem se SOA aplikacija izvodi, te iz sigurnosnih razloga mogu ograničiti ili potpuno ukinuti izravan pristup operacijskom sustavu ili pojedinim njegovim dijelovima. Stoga je bolje rješenje instrumentacija na razini servisnog sloja. Osim samog mjerenja, iznimno su važne i faze prikupljanja i analize podataka.

Uspješno prikupljeni podaci omogućavaju analizu i predviđanje buduće učinkovitosti [25]. Predviđanjem učinkovitosti je moguće predvidjeti eventualne probleme u budućnosti i na vrijeme ih spriječiti.

Kod svake instrumentacije posebna se pažnja mora obratiti na promjene koje mjerne sonde unose u sami sustav (engl. Impact on the System Being Monitored). Ove promjene odnose se na smanjenje propusnosti pojedinog SOA servisa, zbog operacija mjerenja učinkovitosti. Uneseno kašnjenje / usporenje naziva se engl. Impact. Cilj svakog mjernog sustava je smanjiti taj utjecaj koliko je to moguće. Softverskim metodama ovaj utjecaj nije moguće eliminirati. To je moguće jedino hardverskim metodama, kao što je zrcaljenje porta (engl. port mirroring).

Osim što manjeg utjecaja na nadzirani sustav, mjerne sonde bi u idealnom slučaju trebale biti potpuno odvojene od čvora aplikacije koju nadziru. To je poželjno zbog toga što bi u protivnom fatalna pogreška u mjernom sustavu automatski značila i fatalnu pogrešku u nadziranom sustavu, i samim time negativno utjecala na kvalitetu usluge nadziranog SOA sustava. Istraživanja se stoga često bave aspektno orijentiranim pristupima za instrumentaciju [26], međutim takvi sustavi su uglavnom ograničeni na Java platformu i platforme koje omogućavaju modifikaciju koda u koraku između prevođenja i izvršavanja.

3.2. Raspoloživost

Jedna od temeljnih mjera kvalitete usluge SOA aplikacije je njena raspoloživost. Raspoloživost se definira kao omjer vremena u kojem SOA aplikacija ispravno radi, tj. odgovara na zahtjeve klijenata i ukupnog vremena aktivnosti aplikacije.

$$a_{service} = t_{up} \ / \ (t_{up} + t_{down})$$

Raspoloživost je intuitivno jasan i koristan pokazatelj kvalitete usluge. Od modernih SOA aplikacija očekuje se dostupnost u svakom trenutku (24/7). Izmjerena ili izračunata raspoloživost sama nije dovoljna informacija da bi se dobio potpun uvid u funkcioniranje SOA aplikacije.

U praksi se raspoloživost ne može precizno odrediti na samom servisu. Razlog tomu je u činjenici da dostupnost ovisi i o kvaliteti komunikacijskog kanala, a ne samo o ispravnom radu servisa. Stoga je potrebno raspoloživost ispitati izvana, odnosno iz perspektive klijenta. U praksi, raspoloživost se najčešće ispituje putem automatskih robot programa koji servis pozivaju u pravilnim vremenskim intervalima. Automatizirani postupak detekcije i popravke servisa jedna je osnovnih ideja vodilja grane računarstva koja se naziva dizajn samo popravljajućih sustava (engl.Self healing system design). Upravo je raspoloživost jedan od ključnih elemenata koji treba pravilno nadzirati da bi bilo moguće utvrditi pojavu problema u sustavu, i pokušati sustav dovesti u ispravno stanje.

3.3. Nadzor učinkovitosti (engl. Performance Monitoring)

Učinkovitost SOA aplikacije je primarni faktor na temelju kojega krajnji korisnik ocjenjuje kvalitetu usluge SOA aplikacije. Za dobivanje informacije o učinkovitosti pojedinog servisa i cijele SOA aplikacija potrebno je izmjeriti i nadzirati sljedeće mjere [15]:

- Latencija ili vrijeme čekanja (engl. Latency)
- Vrijeme obrade (engl. Processing time)
- Frekvencija poziva (engl. Frequency)
- Mjerenjem uzrokovano kašnjenje (engl. Impact)

Postupak umetanja mjernih modula u nadziranu aplikaciju naziva se instrumentacija. Svaka instrumentacija unosi u sustav određenu količinu kašnjenja.

3.3.1. Latencija ili vrijeme čekanja (engl. Latency)

Latenciju t_l definiramo kao vrijeme proteklo od kada je servis S1 poslao poruku M do trenutka kada je servis S2, primio tu poruku. Visoko vrijeme čekanja najčešće je indikator problema u komunikaciji između servisa (kašnjenje na fizičkoj ili nekoj drugoj nižoj razini OSI protokola).

$t_l = t_{send}(S1) - t_{rec}(S2)$

Kod računanja vremena latencije javlja se poznati problem izračuna vremena u distribuiranim sustavima. Naime, budući da satovi na mjernim sondama servisa S1 i S2 ne moraju biti sinkronizirani, nije moguće izračunati t_l korištenjem bilo kojeg algoritma koji koristi globalno vrijeme. Za izračun vremena latencije potrebno je pomoću Lamport [27] algoritma odrediti vremenski redoslijed mjernih događaja i pomoću lokalnih vremena izračunati vrijeme latencije [24].

3.3.2. Vrijeme obrade (engl. Processing time)

Ukupno vrijeme izvođenja operacije na servisu možemo podijeliti na vrijeme obrade t_p i vrijeme delegiranja [24]. Delegirano vrijeme t_D čini vrijeme za koje servis poziva ostale servise. Razlika između ukupnog vremena poziva i delegiranog vremena vraća kao rezultat vrijeme obrade p. Veliko vrijeme obrade ukazuje na potrebu za provjerom i optimiranjem pojedinih algoritama.

$t_p = t_{EXEC} - t_{DEL}$

Vrijeme delegacije dalje se može raščlaniti na vrijeme obrade pozvanog servisa i vrijeme kašnjenja. Prilikom izvršavanja SOA servisi međusobno komuniciraju razmjenom poruka. Razmjenu poruka možemo predstaviti usmjerenim stablom u kojemu je početni čvor klijent, inicijator poziva. Tematika SOA grafova izvršavanja detaljnije je obrađena u radu [24].

3.3.3. Frekvencija poziva (engl. Frequency)

Frekvencija poziva f je intuitivno jasan pojam. Podrazumijeva broj pozivanja servisa u određenom vremenskom intervalu.

$f = N / 1s$

Frekvencija je važan podatak jer se analizom učestalosti pozivanja nekog servisa mogu dobiti informacije o ponašanju i strukturi cjelokupne SOA aplikacije za vrijeme izvođenja. Frekvencija je osnova za potragu za mogućim uskim grlima SOA aplikacije (engl. Bottlenecks). Naime, servisi i njihove funkcije s najvišim frekvencijama poziva su oni elementi SOA aplikacije čija učinkovitost najviše doprinosi kvaliteti usluge cijele SOA aplikacije.

3.3.4. Mjerenjem uzrokovano kašnjenje (engl. Impact)

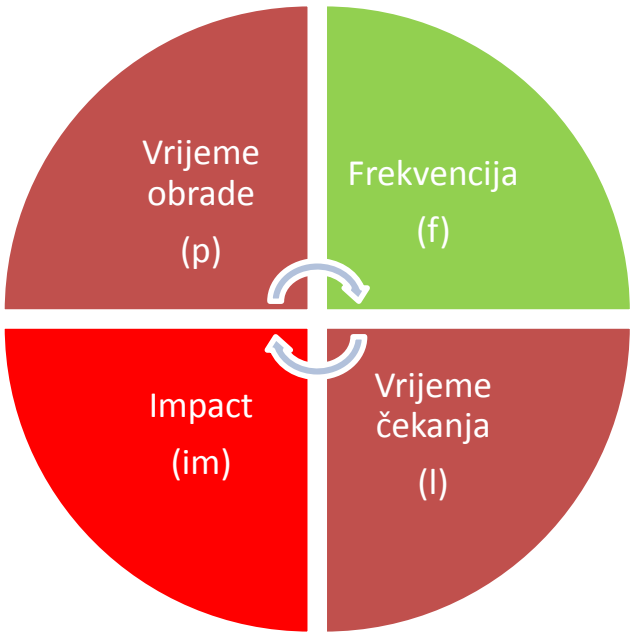
Svako softversko mjerenje unosi određeno dodatno kašnjenje u sustav. Jedna od temeljnih mjera kvalitete mjernog sustava jest vrijednost unesenog kašnjenja t_{IM}. Idealno, vrijeme unesenog kašnjenja trebalo bi biti nula, ili što je moguće bliže nuli. Vrijeme unesenog kašnjenja definiramo kao:

$t_{IM} = t_p - t_p'$

t_{IM} – vrijeme unesenog kašnjenja

t_p – vrijeme izvršavanja, s uključenom instrumentacijom

t_p' – vrijeme izvršavanja, bez instrumetacije



Slika 6 - Temeljne mjere učinkovitosti SOA sustava

Na osnovu navedenih mjera, prikazanih na Slika 6, može se provesti temeljna analiza rada nadziranog sustava i identifikacija eventualnih uskih grla u SOA aplikaciji. Navedene mjere mogu se dobiti samo ako mjerni sustav ima pristup svim porukama koje se razmjenjuju između pojedinih čvorova SOA aplikacije.

3.4. Upravljanje iznimkama i pogreškama

Iznimke (engl. Exceptions) i pogreške (engl. Errors) su mogući događaji u izvođenju procesa koji uzrokuju njegovo neispravno funkcioniranje ili prekid rada. Iako se često ova dva pojma koriste u istom kontekstu, ipak valja napraviti razliku između njih. Naime, iznimke su još uvijek popravljiva stanja procesa. Kažemo da se iznimke „bacaju“ (engl. Throw)). Ako postoji dio koda koji ih može „uhvatiti“ (engl. Catch), onda neće doći do rušenja procesa. Ukoliko ne postoji odgovarajući kod za obradu iznimke, proces dolazi to nepopravljive pogreške (engl. Fatal error) i nakon toga se ruši, uzrokujući brojne probleme kako administratorima, tako i korisnicima sustava.

U kontekstu SOA aplikacija, kvalitetno upravljanje pogreškama je iznimno važno upravo zbog njihovog izravnog utjecaja na stabilnost i pouzdanost sustava. Ideje kojima se istraživači bave uglavnom se bave detekcijom pogrešaka u web servisima [28] lociranim u velikim Cloud sustavima [29], zbog njihove široke rasprostranjenosti u praksi. Osim toga, rasprostranjenost ESB sustava u velikim poduzećima motivira

razvoj sustava za detekciju i upravljanje iznimkama na razini ESB komunikacijskog kanala [30].

Detekcija pogrešaka podjednako je važna i u razvojnoj i u produkcijskoj fazi. U razvojnoj fazi cilj je doći do što detaljnijih informacijama o pogreškama i time na vrijeme otkloniti potencijalne izvore problema u produkcijskoj fazi. Prva prilika za detekciju grešaka je prilikom prevođenja koda (engl. compile time). Moderni programi pišu se u strogo tipiziranim programskim jezicima [31], što donekle smanjuje broj grešaka koji se pojavljuje za vrijeme izvođenja. Iako se značajan dio pogrešaka može eliminirati već u fazi prevođenja i testiranja, ipak u praksi postoje greške koje se ne mogu predvidjeti do produkcijske faze. Ispravljanje pogrešaka u produkcijskoj fazi mnogo je skuplje i nepovoljnije za sve korisnike i same autore SOA aplikacija. U slučaju SOA aplikacija, topologija i ponašanje sustava može se promijeniti u bilo kojem trenutku izvođenja što daje dodatnu važnost detekciji pogrešaka u realnom vremenu.

Fatalne pogreške u produkcijskoj fazi izravno utječu na smanjenje ukupne percepcije kvalitete usluge SOA aplikacije iz perspektive krajnjeg korisnika (engl. End User Experience). Osim toga, ovisno o poslovnoj logici koju SOA aplikacija implementira, fatalne pogreške u produkcijskoj fazi mogu dovesti i do izravnih financijskih troškova vlasniku / operateru SOA aplikacije. Iz ovog razloga, iznimno je važno da SOA aplikacije bude robusne i, koliko je to moguće, neosjetljive na pogreške prilikom izvođenja (engl. Fault tolerant).

Znanstvena zajednica ulaže značajne napore u modeliranje servisno orijentiranih sustava otpornih na pogreške [32]. U realnim situacijama, SOA sustavi komuniciraju preko nepouzdanih i nesigurnih komunikacijskih kanala. Poruke koje se razmjenjuju između čvorova SOA aplikacije mogu biti neispravne ili zlonamjerno modificirane. Pametni SOA sustavi imaju za cilj identificirati takve probleme i ponuditi adekvatna rješenja [33] [34]. U istraživanjima nailazimo i na pokušaje izgradnje teorijskog okvira za identifikaciju i upravljanje različitim scenarijima pojave pogrešaka za vrijeme izvođenja. Primjerice, u radu [35] autori predstavljaju proširenje procesnog jezika SOCK s novim elementima za detekciju pogrešaka u BPEL baziranim SOA aplikacijama. Ovakvi radovi potrebni su i u području REST mobilnih SOA aplikacija koje nalazimo sve češće u praksi.

Izgradnja sustava otpornih na pogreške oslanja se na dvije strategije:

- Podjela sustava u komponente
    - Podjela sustava u komponente odgovara osnovnoj ideji SOA arhitekture – podijeli sustava na servise. Iz ovog razloga, SOA arhitektura osigurava ovaj osnovni preduvjet za izgradnju sustava otpornih na pogreške
- Redundancija
    - Redundancija podrazumijeva uvrštavanje dodatnih komponenti koje su suvišne u slučaju idealnog funkcioniranja sustava. Ove komponente se

koriste tek u slučajevima kad se dogodi fatalna pogreška zbog koje neka osnovna komponenta ne može više obavljati svoju funkciju. Mogućnost dinamičkog dodavanja servisa prilikom izvođenja (engl. Runtime binding) u SOA sustavima olakšava implementaciju redundancije.

Jednom kad se dogodi pogreška, sustav ima dvije mogućnosti reakcije:

- Popraviti pogrešku i nastaviti s radom (engl. Roll-forward)
    - Ako je grešku moguće ispraviti, sustav ili neka vanjska komponenta mogu ukloniti kvar koji je uzrokovao pogrešku i sustav može nastaviti raditi u prvobitnom stanju.
- Vratiti se na neko prethodno stanje i nastaviti s radom (engl. Roll-back)
    - Ako se greška ne može ispraviti, sustav se može vratiti u neko od prethodnih stanja i nastaviti s radom.

3.5. Sigurnosni menadžment

Sigurnost je područje istraživanja koje svakim danom dobiva sve više na važnosti. Priroda podataka koji se pohranjuju u modernim poslovnim aplikacijama sve je osjetljivija i njihova zaštita predstavlja apsolutni imperativ u mnogim slučajevima. U kontekstu SOA arhitekture, s aspekta sigurnosti zainteresirani smo za dva cilja:

a) Postizanje i osiguravanje CIAA svojstava komunikacijskih protokola
b) Osiguravanje integriteta svakog pojedinog servisa u SOA aplikaciji

3.5.1 CIAA svojstva sigurne komunikacije

Osnova SOA komunikacije je razmjena poruka. Poruke u SOA svijetu su najčešće spremljene u obliku teksta. Tekst je medij koji razumiju svi servisi koji čine jednu SOA aplikaciju, bez obzira na tehnologiju u kojoj su implementirani. Tekstualne poruke često sadrže povjerljive (engl. Confidental) informacije za koje pošiljatelj A i primatelj B vjeruju da su tajna. Isto tako, primatelj B želi biti siguran da je poruka došla izravno od pošiljatelja A, tj. da je autentična (engl. Authenticity). Bitno svojstvo sigurne komunikacije jest i osiguranje da poruka na putu od pošiljatelja A do primatelja B nije mijenjana, tj. da je sačuvala svoj integritet (engl. Integrity). Zadnje važno svojstvo sigurne komunikacije u SOA svijetu je vjerodostojnost ovlasti (engl. Authorization) koje pošiljatelj A ima, a koje mu omogućavaju komunikaciju sa primateljem B.

-Povjerljivost (engl. **C**onfidentiality)

Podrazumijeva da poruku mogu pročitati isključivo pošiljatelj i primatelj. U praksi se osigurava kodiranjem poruke ili korištenjem osiguranog komunikacijskog kanala. Šifriranje i odgovarajuće dešifriranje poruke podrazumijeva korištenje simetričnog kripto sustava (npr. AES 256). Prednost ovakvog pristupa je

-Integritet (engl. **I**ntegrity)

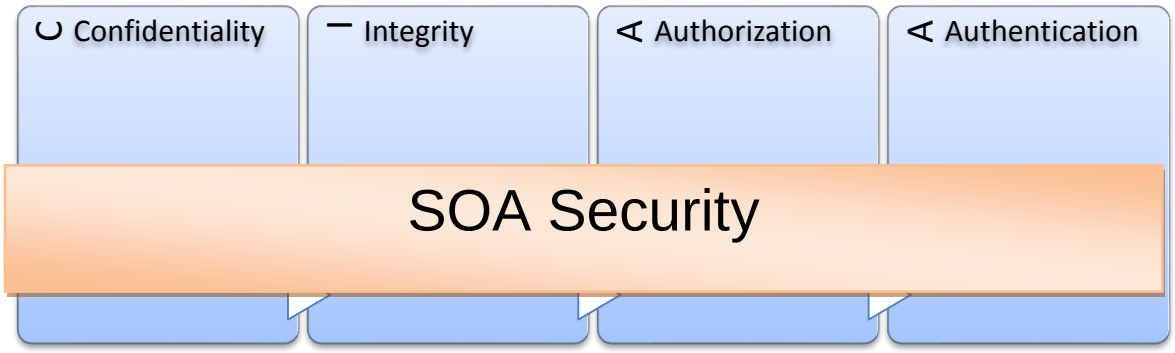
Podrazumijeva da poruka nije mijenjana od strane treće osobe na putu od pošiljatelja prema primatelju. Osigurava se provjerom hash funkcije poruke. Ako se rezultat hash funkcije poruke ne poklapa s očekivanom vrijednošću, poruka je sigurno bila izmjenjena negdje u komunikacijskom kanalu.

-Izvornost (engl. **A**uthenticity)

Podrazumijeva da je navedeni pošiljatelj stvarni pošiljatelj primljene poruke. Najčešće se implementira digitalnim potpisom. Digitalni potpis temelji se na certifikatu koji izdaje vjerodostojna treća strana.

-Autorizacija (engl. **A**uthorization)

Podrazumijeva da pošiljatelj ima dovoljne ovlasti da od primatelja traži ili pruža informacije navedene u poruci. Za utvrđivanje razine prava pošiljatelja, prvo se mora provjeriti da je primljena poruka izvorna i da je primljena od od deklariranog pošiljatelja poruke. Nakon toga, uvidom u vlastitu tablicu ovlasti, svaki servis može dozvoliti ili odbiti traženu operaciju.



Slika 7 - Elementi sigurne komunikacije u SOA sustavu

Gore navedena, i na slici prikazana, četiri zahtjeva su stupovi sigurnosti u SOA svijetu. Bez ispunjenja ovih preduvjeta, nije moguća povjerljiva komunikacija između klijenata i servisa. Osim navedenih generalnih zahtjeva, u razmatranje moramo uključiti i specifične pojave koje nalazimo u praksi kao potencijalne probleme za ispravno funkcioniranje SOA sustava. Na razini procesno orijentiranih jezika za kompoziciju i orkestraciju servisa, kakav je BPEL, ne postoje elementi sigurnosne politike (engl. Security policy) koji bi omogućili slaganje složenih i sigurnih SOA aplikacija koje se prostiru kroz više geografskih i logičkih domena. U radu [36] autori predstavljaju jedan sigurnosni apstraktni model sigurnosnog aspekta SOA sustava. Ovakvi modeli potrebni su i na nižim razinama od razine kompozicije.

3.5.2. Očuvanje integriteta servisa

Osiguranje integriteta i dostupnosti pojedinog servisa drugi je važan cilj istraživanja sigurnosnih aspekata SOA sustava. U osnovi, prijetnje integritetu pojedinog servisa možemo podijeliti u dvije skupine:

-Sigurnosne prijetnje izvana

-Sigurnosne prijetnje iznutra

3.5.2.1 Sigurnosne prijetnje izvana

Presretanje podataka i lažno predstavljanje

Najčešći tip ovakvog napada naziva se Man-in-the-middle (engl.). Ovaj napad izvodi se ubacivanjem treće strane u komunikacijski kanal, između klijenta i servera. Ako zlonamjerna osoba ili program ima pristup komunikacijskom kanalu, onda može izvesti presretanje i promjenu poruka namijenjenih bilo kojoj strani u komunikacijskom kanalu. U praksi ovakvi napadi se često izvode pomoću virusa koji se neprimjetno instaliraju na računalo i ubacuju u komunikacijski kanal [37].

Onesposobljavanje servisa

Servisi su najčešće dostupni putem interneta. Internet kao medij podložan je različitim vrstama zlouporaba, od kojih je jedna od najčešćih tzv. Denial of Service (engl.) napad u kojem maliciozni korisnik ili više njih preopterećuje server fiktivnim prometom i time mu smanjuje učinkovitost ili ga potpuno onesposobljava . Istraživači se već dugo vremena bave analizom i rješavanjem ovog teškog sigurnosnog problema kojemu su izložene sve web bazirane aplikacije [38].

Preuzimanje kontrole nad bazom podataka

Gotovo svi poslovni servisi interno koriste bazu podataka za spremanje poslovnih informacija. Danas su većina baza podataka relacijske baze upravljane jezikom SQL (engl. Structured Query Language). Zbog svoje strukture, jezik SQL može biti zloupotrijebljen od strane zlonamjernog korisnika. U slučaju da servis ne vrši provjeru unosa dobivenog od klijenta, u SQL izraz moguće je ubaciti zlonamjerni kod koji može dohvatiti ili izbrisati podatke iz baze podataka bez znanja administratora servisa. Problem i izazovi rješavanja preuzimanja kontrole nad bazom podataka već dugo su predmet interesa znanstvene zajednice. Većina modela za analizu i sprječavanje napada predviđa instrumentaciju koda koji pristupa bazi podataka. Instrumentacijom je moguće analizirati i detektirati sumnjive komande prema bazi. Primjer Java implementacije jednog takvog sustava, pod nazivom AMNESIA, nalazimo u radu [39].

3.5.2.2. Sigurnosne prijetnje iznutra

Poslovne informacije imaju presudnu važnost za funkcioniranje tvrtki. Gubitak ili krađa poslovnih informacija može nanijeti nemjerljivu štetu tvrtkama. Prema nekim istraživanjima, zlonamjerni zaposlenici predstavljaju treću najveću prijetnju sigurnosti podataka u tvrtkama, uzrokujući milijune dolara gubitaka godišnje [40]. Zaštita SOA sustava iznutra može se unaprijediti kvalitetnom raspodjelom ovlasti unutar sustava. Svaki korisnik SOA sustava bi trebao biti točno identificiran i eksplicitno autoriziran za pojedinu operaciju (engl. whitelist approach). Dodijeljene ovlasti bi trebale biti što manje, ali ipak dovoljne da korisnik može obavljati svoje radne zadatke (engl. need to know basis). Polje sigurnosti sustava iznutra interesantno je područje istraživanja koje se nalazi na granici informatike, teorije igara i sociologije.

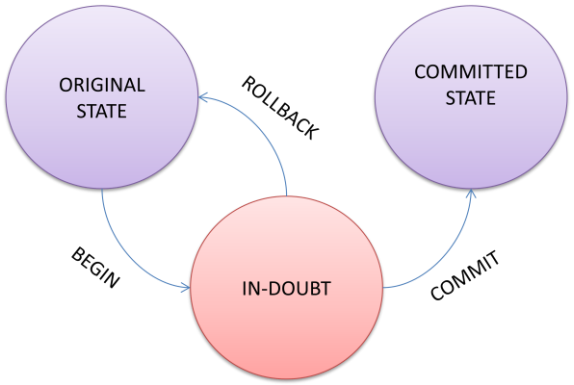
3.6. Distribuirane transakcije i njihov nadzor

Transakcije su osnovna građevna jedinica svakog poslovnog sustava. Transakcije podrazumijevaju skup međuovisnih operacija koje mogu biti u dva stanja:

Stanje A – stanje prije izvršenja transakcije. Često se naziva i originalno stanje (engl. Original state)

Stanje B – stanje nakon uspješne transakcije (engl. Committed state)

Transakcija prelazi iz stanja A u stanje B ako su sve operacije koje ona sadrži uspješno završene. Ako bilo koja od operacija nije uspješno završena, transakcija se vraća u početno stanje A. Kao što je prikazano na Slika 8, dok transakcija nije okončana ona se nalazi u međustanju (engl In-Doubt).



Slika 8 – Stanja transakcije

U kontekstu SOA sustava, razlikujemo dvije vrste distribuiranih transakcija:

-Poslovne transakcije

-Tehnologijske (IT) transakcije

Poslovne transakcije su pojam koji dolazi iz svijeta web aplikacija. Pod ovim pojmom podrazumijevamo niz akcija koje korisnik ili agent treba napraviti u određenom redoslijedu, da bi se transakcija mogla proglasiti uspješno završenom. Poslovne transakcije nisu transakcije u klasičnom smislu, budući da je kod njih dozvoljeno još jedno stanje – neodlučeno. U slučaju problema na bilo kojem dijelu lanca operacija, poslovna transakcija ulazi u međustanje koje se može definirati kao neuspješno ili kao neodređeno stanje transakcije, ovisno o kriterijima nadzora.

Tehnologijske transakcije kao pojam dolaze iz svijeta klasičnih poslovnih aplikacija pisanih najčešće u Javi ili jeziku C#. Tehnologijske transakcije se izvode kao cjelina, što znači da transakcija može završiti samo u stanju A ili u stanju B. Nije dozvoljeno nikakvo međustanje. Tehnologijske transakcije su podržane u standardnim SOA tehnologijama kao što su WCF ili J2E.

U istraživanjima postoje tendencije prema stvaranju QoS aware transakcijskih frameworka. Jedan takav je predstavljen u [41]. Autori su predstavili dodatak na komunikacijski framework Axis koji podržava pouzdane i sigurne SOAP bazirane transakcije. Osim SOAP protokola, istraživanja se bave i industrijskim transakcijama. U radu [42] predstavljaju metodologiju za formalnu verifikaciju SOA transakcija implementiranih u SCADA industrijskim sustavima.

Razvoj jednostavnih transakcijskih protokola na aplikacijskoj razini, posebno za REST arhitekture, je područje zanimljivo za buduća istraživanja. Ovakvi protokoli ne bi ovisili o cijelom nizu infrastrukturnih preduvjeta, kakvi su danas podrazumijevani od strane WCF / Java tehnologijskih transakcija. Samim time, bili bi mnogo pogodniji za razvoj jednostavnih i malih SOA aplikacija kakve se očekuju u budućnosti.

Posebno interesantno područje istraživanja je područje nadzora korisnički definiranih transakcija u okruženju s promjenjivom kvalitetom i dostupnošću Internet veze [6]. Naime, u mnogobrojnim primjenama SOA aplikacija klijenti su mobilni uređaji koji nisu cijelo vrijeme u području pokrivenosti 3G/4G ili nekim drugim tipom Internet signala. Ovakvi klijenti se spajaju putem Internet veze na udaljene REST servise. Od mobilnih klijenata se često očekuje da mogu funkcionirati i bez Internet veze, prikupljanjem podataka i njihovom pripremom za slanje kada Internet veza bude dostupna. Nadzor ovakvih transakcija je iznimno bitan u poslovnim sustavima, jer često prenose važne poslovne informacije. Taj nadzor nije jednostavan jer i sama mjerna sonda i nadzorni sustav na mobilnom klijentu bez Internet veze nije u mogućnosti komunicirati sa svojim serverom i razmijeniti informacije o kvaliteti usluge nadzirane aplikacije. Rješenja koja nedostaju su adaptivni modeli nadzora, prilagodljivi uvjetima okruženja za vrijeme izvođenja. U takvim modelima, mjerna

sonda bi smanjila svoju ovisnost o Internet vezi kombiniranim pristupom slanju podataka. Jedan dio podataka mogao bi se prikupljati i slati uz same pozive prema servisima. Tako formatirani QoS podaci mogli bi se prikupiti od strane sonde na web servisu i proslijediti na daljnju obradu.

3.7. Upravljanje opterećenjem

Da bi se očuvala kontinuirana visoka razina kvalitete usluge, SOA sustav mora biti sposoban funkcionirati i u uvjetima visokog opterećenja. Jedna od osnovnih prednosti SOA arhitekture je njezina dinamičnost. SOA sustavi su visoko prilagodljivi za vrijeme izvođenja (engl.Runtime). Upravo ta karakteristika važna je za uspješnu raspodjelu opterećenja (engl. Load Balancing) u ovakvim sustavima. Veliki broj poruka prema bilo kojem čvoru može preopteretiti taj čvor i usporiti cijeli SOA sustav [43].

Da bi se izbjeglo preopterećenje, SOA sustavi u osnovi koriste tri kriterija za raspodijele opterećenja:

1. Raspodjela opterećenja na osnovi raspoloživosti
2. Raspodjela opterećenja na osnovi izmjerene učinkovitosti
3. Raspodjela opterećenja na osnovi sadržaja

Zbog kompleksnosti SOA sustava, nerijetko su potrebni i decentralizirani sustavi raspodjele opterećenja. U decentraliziranim sustavima pojedini servisi imaju informaciju samo o servisima s kojima su bili u interakciji, tj. susjednim servisima (engl. Neighbours). Pregled trenutno dostupnih, decentraliziranih algoritama raspodjele opterećenja dan je u [44]. Predstavljeni algoritmi su inspirirani Bees algoritmom. Bees algoritam oponaša ponašanje pčela u potrazi za izvorima hrane [45]. Bees algoritam u ovom kontekstu modelira skup servisa S kao resurse, dok su virtualni serveri koji zaprimaju zahtjeve i traže najpogodniji realni server za njihovo izvođenje u ulozi pčela.

3.7.1. Raspodjela opterećenja temeljena na mjerenju raspoloživosti

Topologija SOA sustava može se mijenjati za vrijeme izvođenja. Na osnovu usporedbe sučelja dvaju servisa, može se utvrditi njihova kompatibilnost, odnosno zamjenjivost. Sučelje servisa možemo definirati kao skup funkcija koje servis pruža:

$I_{S1} = (f_1, f_2, \ldots, f_N)$

U slučaju da svaka funkcija sučelja I_{S1} postoji i u sučelju I_{S2}, onda servis S2 može obavljati sve funkcije koje obavlja S1. Upravo je zamjenjivost servisa važna za raspodjelu opterećenja u SOA sustavima. Naime, ukoliko servis S1 ne funkcionira ispravno ili nije optimalan za izvođenje pojedinog poziva, raspoređivač opterećenja može taj zahtjev proslijediti servisu S2.

Kontinuiranim mjerenjem raspoloživosti svakog servisa, moguće je doći do pravodobne informacije o nedostupnosti servisa S1. Nakon utvrđivanja te činjenice, servis ili modul ESB-a zadužen za upravljanje opterećenjem, može sav promet preusmjeriti na servis S2 i time očuvati istu ili približno istu razinu kvalitete usluge.

3.7.2. Raspodjela opterećenja temeljena na mjerenju učinkovitosti

Raspodjela prometa na osnovu učinkovitosti temelji se na stalnom praćenju učinkovitosti svakog pojedinog čvora u SOA aplikaciji. Temeljem dobivenih podataka, može se utvrditi koji čvorovi se nalaze pod kritičnim opterećenjem. Nakon identifikacije kritičnih čvorova, uloga raspoređivača opterećenja (engl. Load balancer) je da promet namijenjen kritično opterećenom čvoru preusmjeri na alternativne čvorove koji nude istu funkcionalnost. Modularni dizajn SOA aplikacija, kao i njihova unificiranost sučelja, predstavljaju prirodne prednosti SOA aplikacija kod dizajna skalabilnih sustava sa raspodjelom opterećenja. U praksi, ukoliko sustav koristi Publish-subscribe arhitekturni obrazac, ulogu raspoređivača opterećenja preuzima ESB ili neki njegov modul.

3.7.3. Raspodjela opterećenja temeljena na sadržaju

Osim po kriteriju učinkovitosti, promet u SOA aplikaciji bi se mogao raspodjeljivati i po drugim kriterijima. Jedan od zanimljivih kriterija je sadržaj (engl. content). Analizom sadržaja poruke moglo bi se zaključiti koji čvor u SOA sustavu može najkvalitetnije odgovoriti poslanom zahtjevu. Ovakav vid usmjeravanja mogao bi imati značajne prednosti u vidu poboljšane učinkovitosti cijelog sustava. Osim toga, ovakav sustav usmjeravanja je proaktivan i ne čeka da neki čvor sustava dođe do točke kritičnog opterećenja, nego osigurava da do preopterećenja uopće ne dođe. Raspodjela opterećenja temeljena na sadržaju je relativno novo područje istraživanja [43] u kojem leži veliki potencijal za unapređenja.

4. Dodatna područja istraživanja

4.1. Paralelna obrada prikupljenih podataka

U posljednje vrijeme postoji snažan trend paralelizacije algoritama. Paralelizacija algoritma preduvjet je za njegovo izvođenje na paralelnim sustavima, kakvi su superračunala. Jedan poseban tip paralelnih računala su GPU (engl. General Processing Units) čipovi na današnjim modernim grafičkim karticama. GPU čipovi danas sadržavaju i po više stotina jezgri od kojih se svaka može izvršavati istovremeno. Dostupnost i prednosti ovakvih arhitektura još nisu iskorištene za obradu i analizu podataka prikupljenih nadzorom SOA aplikacija. Samim time, ovo je jedno od mogućih značajnijih područja za istraživanja u budućnosti.

4.2. Upravljanje SOA sustavima

Značajan problem napretku SOA arhitekture predstavlja i nepostojanje univerzalnog, jednostavnog i široko prihvaćenog standarda za upravljanje servisima. Sustav za upravljanje servisima ima odgovornost da omogući međusobno pronalaženje pojedinih servisa i olakša njihovo povezivanje. Najčešće korišteni način upravljanja servisima, posebice web servisima, jest putem kataloga servisa (engl. SOA registry). Katalog servisa podržava jezik UDDI za pronalaženje pojedinih servisa koji odgovaraju danim karakteristikama. Jezik UDDI (engl.Universal Description Discovery and Integration) je baziran na XML sintaksi. UDDI se koristio dugo vremena kao de facto standardni jezik otkrivanja servisa i u istraživačkom i u poslovnom svijetu. Međutim, zadnja standardizirana verzija jezika UDDI, verzija ,2 objavljena je 2002.godine. UDDI je složen i nepraktičan jezik za brojne agilne primjene, posebno za manje projekte. Sve to ukazuje na potrebu za modernijim i jednostavnijim jezikom za otkrivanje servisa u SOA svijetu [46].

4.3. Upravljanje promjenama

SOA aplikacije su dinamični sustavi podložni stalnim promjenama. Svaka SOA aplikacija je živi sustav koji raste, razvija se i stari [47]. Cijeli taj proces izaziva brojne nove probleme koji mogu biti zanimljiva područja budućih istraživanja. Primjerice, rast sustava izaziva i rast troškova korištenja resursa oblaka. U interesu je i davatelja usluga i krajnjih korisnika pronaći metode optimalnog iskorištavanja postojećih resursa prije zauzimanja novih.

5. Zaključak i buduća istraživanja

U ovom radu dan je pregled aktualnih područja istraživanja u svijetu Servisno orijentiranih aplikacija. Identificirana su temeljna područja važna za kvalitetu usluge u SOA sustavima. Napravljen je pregled dosadašnjih radova na tim područjima i ukazano je na moguća poboljšanja istih ili na moguće nove smjerove istraživanja. Identificirana su i neka dodatna područja istraživanja Servisno orijentiranih aplikacija, za koja se tek u posljednje vrijeme pojavljuje interes u istraživačkoj zajednici.

Na osnovu ovdje provedenih istraživanja trenutnog stanja područja istraživanja, planiran je doktorat koji će se fokusirati na rješavanje nekih od aktualnih istraživačkih problema opisanih u ovom radu. Fokus doktorata će biti na REST mobilnim aplikacijama u Cloud okruženju. Budući da je to područje novo i tek ulazi u primjenu nedostaju odgovarajući apstraktni modeli i metodologije nadzora. Cilj je teorijski opisati SOA aplikacije, ponuditi unificirani model komunikacije, a nakon toga razmotriti njegovu primjenu za izgradnju apstraktnog nadzornog SOA sustava. Nadzorni sustav morao bi uzeti u obzir činjenice i specifičnosti Cloud i mobile okruženja. U doktoratu će biti razmatrana i mogućnost izgradnje adaptivnog, samosvjesnog (engl. Self-Aware) nadzornog sustava koji bi omogućio automatiziranu kontrolu i ispravljanje problema u složenim SOA sustavima.

Bibliografija

[1] W3C, »Web Services Description Language (WSDL) Version 2.0 part1 : Core Language,« 2007. [Mrežno]. Available: http://www.w3.org/TR/2007/REC-wsdl20-20070626/. [Pokušaj pristupa 18 04 2013].

[2] N. M. Jossutis, SOA in practice, O'Reilly, 2007.

[3] D. Box, Essential COM, Addison-Wesley Professional, 1998.

[4] A. Gokhale, B. Kumar i A. Sahuguet, »Reinventing the wheel? CORBA vs. Web Services,« *Proceedings of international world wide Web conference,* 2002.

[5] S. Ickin, K. Wac, M. Fiedler, L. Janowski, J.-H. Hong i A. K.Dey, »Factors influencing quality of experience of commonly used mobile applications,« *Communications Magazine, IEEE,* svez. 50, br. 4, pp. 48-56, 2012.

[6] D. Chalmers i M. Sloman, »A SURVEY OF QUALITY OF SERVICE IN MOBILE COMPUTING ENVIRONMENTS,« *IEEE Communications Surveys,* 1999.

[7] U. Glasser, Y. Gurevich i M. Veanes, »An Abstract Communication Model,« *Technical Report, Microsoft Research,* 2002.

[8] D. C. Marinescu, Cloud Computing: Theory and Practice, 2012.

[9] D. Limited, »UNDERSTANDING The Cloud Computing Stack SaaS, Paas, IaaS,« 2011.

[10] InfoWorld.com, *The state of cloud computing: New public cloud plays leap id as the private cloud slowly evolves,* 2013.

[11] M. Zink, P. Shenoy, D. Irwin i E. Cecchet, »Amazon CloudWatch to monitor cloud resource usage,« *Whitepaper,* 2010.

[12] M. P. Papazoglou, P. Traverso i S. Dustdar, »SERVICE-ORIENTED COMPUTING: A RESEARCH ROADMAP,« *International Journal of Cooperative Information Systems,* pp. 223-255, 2008.

[13] S. Frolund i J. Koistinen, »QML: A Language for Quality of Service Specification,« *Hewllet Packard Whitepaper,* 1998.

[14] M. H. Kees, N. Sidorova, C. Stahl i H. Verbeek, »A Price of Service in a

Compositional SOA Framework,« *Computer Science Report,* 2007.

[15] I. Zoraja, G. Trlin i M. Matijević, »Monitoring SOA Applications with SOOM Tools: A Competitive Analysis,« *Business Systems Research,* 2013.

[16] J. Shao i Q. Wang, »A Performance Guarantee Approach for Cloud Applications Based on Monitoring,« *35th IEEE Annual Computer Software and Applications Conference Workshops,* pp. 25-30, 2011.

[17] M. Dhingra, J. Lakshmi i S. Nandy, »Resource Usage Monitoring in Clouds,« *ACM/IEEE 13th International Conference on Grid Computing,* pp. 184-190, 2012.

[18] S. Chatterjee, »Messaging Patterns in Service-Oriented Architecture, Part 1,« *The Architecture Jorunal,* April 2004.

[19] D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux, J. Lejeune i J. Sopena, »Towards QoS-Oriented SLA Guarantees for Online Cloud Services,« *In proceeding of: 13th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID'13),* 2013.

[20] G. F. Anastasi, »Quality of Service Management in Service Oriented Architectures,« *Doctoral Thesis,* November 2011.

[21] D. E. Sarna, Implementing and Developing Cloud Computing Applications, CRC Press, 2011.

[22] V. Deora, J. Shao, W. Gray and N. Fiddian, "A quality of service management framework based on user expectations," *Service-Oriented Computing - ICSOC 2003,* pp. 104-114, 2003.

[23] N. Looker, M. Munro i J. Xu, »A Comparison of Network Level Fault Injection with Code Insertion,« *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International,* pp. 479-484, 2005.

[24] I. Zoraja, G. Trlin i V. Sunderam, »ELICITING THE END-TO-END BEHAVIOR OF SOA APPLICATIONS IN CLOUDS,« *Computing and Informatics (Prihvaćen za objavu),* 2013.

[25] S. Punitha i C. Babu, »Performance Prediction Model for Service Oriented Applications,« *The 10th IEEE International Conference on High Performance Computing and Communications,* pp. 995-100, 2008.

[26] A. Popovici, T. Gross i G. Alonso, »Dynamic weaving for aspect oriented programming,« *Proceedings of the 1st international conference on Aspect-oriented software development,* pp. 141-147, 2002.

[27] L. Lamport, »Time, clocks, and the ordering of events in a distributed system,« *Communications of ACM,* 1978.

[28] C. Liu, Y. Xu, F. Deng, J. Xiu i Z. Lu, »A rule-based exception handling approach in SOA,« *Computer Application and System Modeling (ICCASM), 2010 International Conference on,* pp. 137-141, 2010.

[29] K. Xiong and H. Perros, "Service Performance and Analysis in Cloud Computing," *Services - I, 2009 World Conference on,* pp. 693-700, 2009.

[30] P. Bhuyan, T. K. Choudhury i D. P. Mahapatra, »A Novel Approach for Exception Handling in SOA,« *The Second International Conference on Computer Science, Engineering and Applications (CCSEA-2012), Proceedings,* pp. 425-433, 2012.

[31] B. C. Pierce, Types and Programming Languages, The MIT Press, 2002.

[32] G. Fan, H. Yu, L. Chen i D. Liu, »A Method for Modeling and Analyzing Fault-Tolerant Service Composition,« *Software Engineering Conference, 2009. APSEC '09. Asia-Pacific,* pp. 507-514, December 2009.

[33] S. Kounev, F. Brosig, N. Huber i R. Reussner, »Towards self-aware performance and resource management in modern service-oriented systems,« *IEEE International Conference on Services Computing,* pp. 621-624, 2010.

[34] R. Zhang, »Autonomic Performance Recuperation for Service-oriented Systems,« *Services Computing, 2007. SCC 2007. IEEE International Conference on,* pp. 544-551, 2007.

[35] C. Guidi, I. Lanese, F. Montesi i G. Zavattaro, »Dynamic Error Handling in Service Oriented Applications,« *Fundameta Informaticae,* pp. 73-102, 2009.

[36] M. Menzel, I. Thomas, C. Wolter i C. Meinel, »SOA Security - Secure Cross-Organizational Service Composition,« *Proceedings of the Stuttgarter Softwaretechnik Forum,* 2007.

[37] D. Serpanos i R. Lipton, »Defense against man-in-the-middle attack in client-server systems,« *Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on,* pp. 9-14, 2001.

[38] D. Gretsy, Q. Shi i M. Merabti, »Requirements for a General Framework for Response to Distributed Denial-of-Service,« *Computer Security Applications Conference, 2001. ACSAC 2001. Proceedings 17th Annual,* pp. 422-429, 2001.

[39] W. G. Halfond i A. Orso, »AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks,« *IEEE/ACM international Conference on Automated*

software engineerin, pp. 174-183, 2005.

[40] C. Hadnagy, Social Engineering: The Art of Human Hacking, Wiley, 2011.

[41] P. H. Phu, D. S. Yoo i M. Yi, »A Framework Supporting Quality of Service for SOA-based Applications,« *Management of Convergence Networks and Services,* svez. 4238, pp. 232-241, 2006.

[42] I. Popovic, V. Vrtunski i M. Popovic, »Formal Verification of Distributed Transaction Management in a SOA Based Control System,« *18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems,* pp. 206-215, 2011.

[43] A. K. Y. Cheung i H.-A. Jacobsen, »Dynamic Load Balancing in Distributed Content-Based Publish/Subscribe,« *Middleware,* pp. 141-161, 2006.

[44] M. Randles, D. Lamb i A. Taleb-Bendiab, »A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing,« *IEEE 24th International Conference on Advanced Information Networking and Applications Workshops,* pp. 551-556, 2010.

[45] S. Nakrani i C. Tovey, »On Honey Bees and Dynamic Server,« *Adaptive Behavior,* svez. 12, br. 3-4, pp. 223-240, 2004.

[46] A. Mintchev, »Interoperability among Service Registry Implementations: Is UDDI Standard Enough?,« *Web Services, 2008. ICWS '08. IEEE International Conference on,* pp. 724-731, 2008.

[47] F. Machida, D. Kim i K. Trivedi, » Modeling and Analysis of Software Rejuvenation in a Server Virtualized System,« *Performance Evaluation: Special Issue on Software Aging and Rejuvenation,* pp. 212-230, 2013.