UDC: 004.738.4 Info M str. 16-26

SOA I ORKESTRACIJA SERVISA – ISTORIJAT, ULOGA I OPEN SOURCE TEHNOLOGIJE
SOA AND SERVICES ORCHESTRATION – HISTORY, ROLE AND OPEN SOURCE TECHNOLOGIES

Boris Damjanović, FON

REZIME: Ideja o korišćenju višekratno upotrebljivih servisa koja čini osnovu servisno orijentisane arhitekture (SOA – Service Oriented Architecture) postaje jedan od najvažnijih koncepata u dizajnu sistema koji se koristi za integrisanje softverskih aplikacija jedne ili više organizacija. Namjena servisno orjentisane arhitekture je da omogući izmijene poslovnih procesa bez potrebe da se fokusiraju na podređenu tehnologiju. Osnovna namjena SOA da omogući da poslovanje postane lakše za upravljanje i fleksibilnije nije mogla uspješno da bude implementirana sve do pojave prograsmkog jezika BPEL. BPEL (Business Process Execution Language for Web Services, takođe i WS-BPEL, BPEL4WS) je jezik koji se koristi za kompoziciju, orkestraciju i koordinaciju Web servisa. On posjeduje bogat riječnik za opisivanje ponašanja poslovnih procesa. BPEL dozvoljava programerima da definišu poslovne procese tako da različiti poslovni servisi mogu da budu povezani ili orkestrirani zajedno da bi dovršili neki poslovni zadatak. U orkestraciji, centralni proces (koji može da bude još jedan Web servis) kontroliše ponašanje ostalih Web servisa i koordinira izvršenje različitih operacija u Web servisima koji su uključeni u operaciju. U ovom radu će biti prikazan pregled najvažnijih tehnika, tehnologija, standarda i protokola koji su rješavanjem pojedinačnh problema, korak po korak doveli do koncepta orkestracije servisa.

KLJUČNE REČI: SOA, BPEL, orkestracija servisa

ABSTRACT: The idea of using service oriented architecture (SOA) is becoming one of the most important concepts in the design of a system used to integrate software applications of one or more organizations. The purpose of SOA is to enable business process changes without the need to focus on subordinate technology. The basic purpose of SOA - to enable business to become more manageable and more flexible - could not be successfully implemented without the appearance of programming language BPEL. BPEL (Business Process Execution Language for Web Services, also WS-BPEL, BPEL4WS) is the language used to compose, orchestrate, and coordinate a Web services. He has a rich vocabulary for describing business process behavior. BPEL allows programmers to define business processes in such a way that different business services can be linked or orchestrated together to complete a business task. In orchestration, the central process (which can be another Web service) controls the behavior of other Web services and coordinates the execution of various operations in Web Services involved in the operation. This paper will present an overview of the most important techniques, technologies, standards and protocols that, by solving individual problems, step by step brought the concept of orchestration of services.

KEY WORDS: SOA, BPEL, service orchestration

I. UVOD

Service oriented architecture (SOA) je tokom prethodnih godina postao prioritetan koncept u pristupu dizajnu sistema, njegovom razvoju i njegovoj integraciji. Oslanjanjem na otvorene standarde i sveprisutni internet, SOA nam predstavlja ideju višekratno upotrebljivih servisa koji odgovaraju samostalnim logičkim jedinicama posla [1].

SOA postaje sve više prihvaćena kao paradigma za integrisanje softverskih aplikacija jedne ili više organizacija. Po ovoj paradigmi, nezavisno razvijene i upravljane aplikacije se predstavljaju kao Web servisi koji su u stanju da međusobno komuniciraju pomoću XML-orjentisanih standarda, a najčešće SOAP i njemu pridruženih specifikacija [2].

Dinamičko stanje u kojem je neka organizacija osposobljena da efikasno koristi informacione tehnologije i da ostvaruje svoje poslovne ciljeve naziva se Business-IT alignment. U današnjem svijetu generalno postoji velika potreba za poravnanjem (alignment) poslovnih organizacija i IT sektora. Orjentacija ka servisima kako na nivou poslovne organizacije tako i na nivou arhitekture predstavlja najbolji način da se dostigne robustan nivo ovog poravnanja. Tokom prilagođenja i implementacije Service oriented architecture (SOA) rješenja, neophodno je iskoristiti različite funkcionalnosti koje nam pružaju već postojeće aplikacije. Takve aplikacije mogu generalno dolaziti od različitih poznatih isporučilaca poput SAP-a ili Oracle-a, od aplikacija koje su samostalno razvijane unutar neke organizacije ili mogu biti neke veoma stare (legacy) aplikacije. Građenje, dizajniranje i isporuka servisno orjentisanih aplikacija upravo i zahtjeva integraciju sa postojećim aplikacijama neke poslovne organizacije [3].

Cilj i osnovna namjena SOA je da omogući da poslovanje postane lakše za upravljanje, fleksibilnije i da počne da brže odgovara na promjene. Namjena servisno orjentisane arhitekture (Service Oriented Architecture - SOA) je da omogući izmijene poslovnih procesa bez potrebe da se fokusiraju na podređenu tehnologiju.

S druge strane, BPM (Business Process Management) je skup tehnologija koje omogućavaju kompaniji da izgradi, obično kroz vizualno prikazan tok (korak po korak), izvršenje nekog procesa koje se proteže duž nekoliko organizacija ili sistema uz pokušaj da se opis poslovanja odvoji od tehnologije. BPM (Business Process Management) je jedna od tehnologija koja je najavljivana sa fanfarama ali koja na žalost sve do pojave BPEL jezika, nije ispunila očekivanja.

Programski jezik BPEL dozvoljava programerima da definišu poslovne procese tako da različiti poslovni servisi mogu da budu povezani ili orkestrirani zajedno da bi dovršili neki poslovni zadatak. U orkestraciji, centralni proces (koji može da bude još jedan Web servis) kontroliše ponašanje ostalih

Web servisa i koordinira izvršenje različitih operacija u Web servisima koji su uključeni u operaciju, pri čemu uključeni Web servisi ne znaju da su uključeni u kompoziciju i da su dio višeg poslovnog procesa.

U ovom radu će biti predstavljen veći broj tehnika, tehnologija i protokola čije poznavanje je potrebno da bi se shvatio dosadašnji razvoj, današnji značaj i mogući pravci razvoja koncepta orkestracije Web servisa.

II. ISTORIJA – MAINFRAME RAČUNARI

U vrijeme mainframe sistema kakav je bio IBM System 360, tokom šezdesetih i sedamdesetih godina, računari su rijetko međusobno komunicirali. Ako je bilo potrebno ostvariti međusobnu komunikaciju, takav proces se obično svodio na prebacivanje podataka pomoću magnetne trake sa jednog sistema na drugi [1]. Tokom šezdesetih i sedamdesetih godina prošlog vijeka postojao je samo jedan tip softverske arhitekture. Mainframe računaru koji je obrađivao podatke u to doba mogli smo da pristupamo jedino preko običnih terminala. Ovakav terminal-mainframe model je dosta jednostavan - mainframe radi sav posao a zatim na terminale šalje samo tekstualne podatke. Jedini posao terminala je da prihvati tekst i da ga predstavi korisniku. Prednost ovakvog modela je jednostavnost implementacije njegovih aplikacija, jer nam je radno okruženje aplikacije (mainframe) unaprijed poznato, a poznat nam je i način pristupanja ovoj aplikaciji (terminal). Negativne strane ovakvog modela vezane su za nedostatak skalabilnosti, npr. u slučaju kada se pojavi više korisnika ove aplikacije ili kada se pojavi potreba da se doda više funkcionalnosti u navedenu aplikaciju. Kako se u mainframe računaru nalaze svi podaci, kompletna poslovna logika kao i logika za prezentaciju, s jedne strane je jednostavno razviti prostije aplikacije, a s druge strane ih je nešto teže mijenjati u skladu sa zahtjevima posla.

III. SEKVENCIJALNI, PROCEDURALNI I OBJEKTNO ORJENTISANI PRISTUP

Prvi programski jezici bili su sekvencijalni jezici. Ovi jezici nisu dozvoljavali nikakvo preuzimanje koda jer su bili dizajnirani isključivo za obradu instrukcija u nekom nizu. Zbog toga je programer morao svaki put ponovo da unosi sekvencu naredbi koja je morala ponovo da se izvrši. Sa pojavom proceduralnih jezika poput Fortran-a, C-a, Cobol-a i Basic-a pojavila se i osnovna forma pojma usluge (servisa). Kod koji je morao da se ponavlja sada je odvojen u posebnu proceduru (metod, funkciju ili potprogram). Ova odvojena procedura je kasnije mogla biti pozivana iz različitih mjesta u programskom kodu. Ovakav način pisanja programa olakšavao je održavanje koda jer je neka izmjena morala da se napravi samo na jednom mjestu. Ovakvi programi su bili i efikasniji jer je ponovljeni kod bio samo jednom smješten u memoriju računara. Nakon proceduralnih jezika došlo je do pojave objektno orjentisanih jezika poput C++ i Jave. Ovakvi jezici su predstavili koncept klasa, koje su enkapsulirale kod i podatke. Ovakve klase su zatim mogle biti korištene bilo gdje u programu [2].

IV. KLIJENT SERVER ARHITEKTURA

Kada su se tokom osamdesetih godina pojavili personalni računari, programeri su počeli da traže načine da iskoriste njihovu snagu [1]. Ovakva arhitektura, koja se nazivala klijent/server arhitektura se i dalje oslanjala na mainframe računare, ali je sada dobar dio posla koji je mainframe nekada radio mogao da bude prepušten PC računarima koji su na njega bili povezani. Npr. logika za prezentaciju, dio poslovne logika i validacija je sada mogla da bude prebačena na stranu personalnih računara, dok su na mainframe računaru ostali podaci i dio poslovne logike. Prednosti ovakvog pristupa su bile u povećanom nivou skalabilnosti, jer je zahvaljujući činjenici da su PC računari počeli da rade dio posla, postalo jednostavnije dodavati nove korisnike ili novu poslovnu logiku na sistem. Negativne strane ovakvog pristupa vidljive su kroz otežano održavanje aplikacija. Ako pretpostavimo da imamo 500 korisnika ovakvog sistema, kada dođe vrijeme za njegovo ažuriranje sa novom aplikacijom koja dodaje neku novu funkcionalnost sistemu, takva aplikacija se mora instalirati na 500 klijenata (personalnih računara). Slijedeći problem koji se u ovakvoj konfiguraciji može javiti je činjenica da različiti PC računari imaju potpuno različite konfiguracije. U nekoj velikoj organizaciji moglo bi se desiti da imamo korisnike koji na računarima imaju Windows 95, Windows 98se, Windows 2000, Windows XP, Linux Fedora, Debian Linux itd. Pravi izazov bi bio kreirati aplikaciju koja bi bila u stanju da radi samo na svim PC računarima na kojima su instalirane različite verzije Windows operativnih sistema.

V. DISTRIBUIRANO RAČUNARSTVO - CORBA, DCOM I EJB

Distribuirano računarstvo počelo je svoj razvoj sa razvojem programiranja socket-a, koji su dozvoljavali aplikacijama da uspostave konekcije i dijele podatke u realnom vremenu. Pojava socketa je stvorila pretpostavke za dijeljenje podataka, ali nije po definiciji omogućila dijeljenje. Zbog toga je bio potreban dalji razvoj da bi se aplikacijama omogućilo da dijele funkcionalnost. Ova mogućnost pojavila se sa razvojem RPC (Remote Procedure Call) koncepta, koji se još nazivao i klijent-server programiranje. Ovaj koncept se zasnivao na konceptu socketa, pri čemu je uveo još jedan nivo apstrakcije između low lewel mrežnog programiranja i programera. Pored toga, kada govorimo o dijeljenju funkcionalnosti između aplikacija, RPC je uveo primitivan način deklarisanja interfejsa između različitih servisa. Nakon koncepta RPC pojavio se i ORB (Object Request Broker) koncept koji je u svijet distriburanog računarstva doveo objektno orjentisano programiranje. ORB tehnologija je omogućila programiranje udaljenih objekata koji se nalaze u različitim aplikacijama. Najpoznatiji primjeri ORB tehnologije su bili CORBA (Common Object Request Broker Architecture) i Java RMI (Remote Method Invocation), od kojih je posebno CORBA donijela veliki broj novih ideja vezanih za razvoj servisa i SOA [3].

Ovakvi trendovi su doveli do nezavisnog razvoja nekoliko bitnih poboljšanja u distribuiranim sistemima. Rješenje pod nazivom CORBA (Comon Object Request Broker Architecture) pojavilo se 1991. godine. Ovaj koncept je predstavljao pokušaj da se standardizuje distibuirano izvršenje programskih funkcija [1]. Iako je u prvim verzijama podržavao samo programski jezik C, od 1998. godine i verzije 2.0 počinje da podržava i ostale programske jezike. Međutim, sa razvojem Jave koja je u velikoj mjeri pojednostavila distribuirano programiranje pomoću koncepta RMI (Remote Method Invocation) i kasnije sa pojavom XML-a, popularnost CORBA koncepta je bitno opala. DCOM (Distributed Computing Object Model) je tehnologija u vlasništvu Microsofta čije je postojanje u velikoj mjeri bilo motivisano postojanjem CORBA-e. Ova tehnologija koja se po prvi put pojavila 1993. godine u Microsoftovom svijetu je postigla određeni uspjeh, ali je zbog svoje vlasničke prirode van tog svijeta bila ograničena. Veliki broj ERP (Enterprise Resource Planning) sistema tog vremena najčešće je koristio tehnologije koje nisu pripadale Microsoftu. Kasnija pojava Javine EJB (Enterprise Java beans) platforme mogla je biti posmatrana kao Java alternativa DCOM-u, jer je dijelila brojne slične karakteristike [1].

U kasnim devedesetim godinama prethodnog vijeka, sa sve širim prihvatanjem interneta, poslovne kompanije su počele da shvataju prednosti proširenja njihove komunikacione platforme na partnere i kupce. Prije pojave interneta, komunikacije između organizacija bile su skupe i ograničene na iznajmljene linije, osim za najveće kompanije. Međutim, pokazalo se da je korištenje CORBA i DCOM tehnologija vrlo veliki izazov zbog postojanja ograničenja u mrežnoj strukturi (npr. firewall) kao i zbog činjenice da je većina kompanija koristila različite, međusobno suprotstavljene tehnologije [1].

VI. SOAP PROTOKOL

Kada se pojavila SOAP (Simple Object Access Protocol) tehnologija u januaru 2000. godine, ona je predstavljana kao lijek za navedene probleme. Na ovu tehnologiju se gledalo kao na RPC alternativu za CORBA i DCOM. Kako je RPC bio predominantan model u distribuiranom računarstvu, prirodno je bilo da ga SOAP naslijedi u punom kapacitetu. Rješenja zasnovana na RPC su sa sobom donosila i određene probleme. Čvrsta sprega (tight coupling) između lokalnog i udaljenog sistema zahtjevala je vrlo veliku propustnu moć mreže. Prefinjena priroda RPC-a zahtjevala je predvidljivu mrežu, dok je osobina komunikacija baziranih na internetu nepredvidiva latencija. Takođe, RPC pokušava da pruži podršku svim prirodnim tipovima podataka (integer, string,array, ...), a to postaje veliki izazov kada treba uspostaviti vezu između nekompatibilnih jezika kao što su Java i C++. Zbog toga su SOAP poruke koje su pratile RPC stil naslijedile sve pomenute nedostatke i ograničenja [1].

Paralelno sa razvojem ORB (Object Request Broker) tehnologije razvijen je i asinhroni način prenošenja poruka. Ova tehnologija se takođe oslanjala na sockete, ali je pružala odgovarajuće prednosti u smislu skalabilnosti integracije aplikacija. Ova skalabilnost je proizilazila iz asinhrone prirode razmjene poruka koja je dozvoljavala aplikaciji pošiljaocu da nastavi sa radom bez čekanja na odgovor od primaoca. Ovakav metod razmjene poruka između aplikacija omogućavao je korištenje redova za slanje i primanje poruka. Ovakav indirektan način slanja poruka omogućio je kreiranje labave sprege (loose coupling) između aplikacija. Još jedna prednost ovakvog pristupa ogledala se u činjenici da je sada postalo moguće garantovati isporuku, jer su se poruke mogle zadržavati na oba kraja mreže. Sada je postalo moguće čak i simulirati sinhroni rad na način da se svakoj poruci dodjeli korelacioni ID koji omogućava poređenje poruke sa zahtjevom i poruke sa odgovorom [3].

Kao dodatak RPC stilu razmjene SOAP poruka, kreatori standarda su osmislili ono što je danas poznato pod imenom dokument stil (document-style) SOAP poruka. RPC stil kreiranja poruka služio je za kreiranje čvrsto povezanih aplikacija, gdje je program pokrenut na jednoj mašini mogao gotovo transparentno pozivati neku funkciju na drugoj mašini. Namjena RPC-a je da tretira udalljene funkcije kao da se one nalaze na istoj mašini, bez zalaženja u način funkcionisanja mreže. Npr. konvenconalna klient - serevr aplikacija bi koristila SOAP RPC stil prenosa poruka [1].

Dokument stil je osmišljen više kao sredstvo za razmjenu poruka između aplikacija, na takav način da omogući integracije zasnovane na labavoj sprezi, poput razmjene dokumenata ili podataka. Iako je Microsoft bio rani predlagač dokument stila, ipak ga je prvo objavio Sun, koji je ubrzo nakon toga objavio i Java API za XML Web servise (JAX-WS) [1].

VII. WEB SERVISI

Pojava Web servisa (Web Services) donijela je sa sobom potrebu standardizacije da bi se smanjila heterogenost izazvana upotrebom različitih tehnologija, kao što su RPC, ORB i različini načini razmjene poruka. Posebno važno je da su oni predstavili jedan novi standard zvan XML (Extensible Markup Language). Interfejsi Web servisa su poboljšani pojavom WSDL-a (Web Services Definition Language) standarda koji je omogućio da interfejs nekog servisa bude nezavistan od jezika, platforme i upotrebljene midleware tehnologije. Slične ideje su rafinirane sa pojavom (UDDI Universal Description, Discovery, and Integration) XML baziranog registra, gdje su organizacije mogle da registruju svoje odnosno da lociraju tuđe servise. Konačno, standardan format za razmjenu poruka dobijen je sa pojavom SOAP standarda [3].

U određenim slučajevima, sami po sebi Web servisi nisu dovoljni da se izađe na kraj sa svim problemima koje izaziva pomenuta heterogenost aplikacija. Na primjer, oni ne mogu da riješe problem do koga dolazi ako dobavljač (service provider) i korisnik servisa koriste različit protokol. Kao rješenje tog problema pojavio se model arhitekture softvera nazvan ESB (Enterprise service Bus). ESB nam obezbjeđuje veliki broj funkcija, uključujući transformaciju protokola i poruka, rutiranje poruka na osnovu sadržaja i konteksta, kvalitet usluge (QoS), obogaćenje podataka i brojne druge funkcije. Kao dodatak Web servisima i ESB tehnologiji, SOA koncept mora

da obezbjedi sredstva za integraciju postojećih aplikacija da bi postao zaokruženo rješenje za integraciju. Ovo u velikom broju slučajeva zahtjeva pakovanje postojećih aplikacija u Web servise ili korištenje aplikacija za komunikaciju sa modernijim aplikacijama [3].

VIII. SOA – SERVICE ORIENTED ARCHITECTURE

Koncepti koje danas povezujemo sa pojmom SOA počeli su da se javaljaju sa širokim prihvatanjem interneta. U 2003. godini Roy Schulte iz Gartner Group kompanije je prvi upotrijebio termin SOA, koji je ubrzo postao sveprisutan. Tokom vremena, pojavile su se i odgovarajuće definicije ovog termina. "SOA je IT strategija koja organizuje diskretne funkcije koje su sardžane u aplikacijama preduzeća u interoperabilne, na standardima bazirane servise koji se mogu kombinovati i ponovo upotrebljavati veoma brzo da bi zadovoljili potrebe poslovanja."

Informacioni sistemi moraju da brzo pruže podršku poslovnom sistemu, ali i da se dovoljno brzo prilagode razvoju novih tehnologija. Najveći dio poslovnih informacionih sistema su heterogeni, jer u sebi sadrže širok rang sistema, aplikacija, tehnologija i arhitektura. U cjelini gledano, SOA nije radikalno nova arhitektura, već više predstavlja evoluciju dobro poznatih arhitektura i metoda integracije. „SOA je stil arhitekture koji podržava labavo spregnute sisteme koji pružaju fleksibilnost poslovanja na interoperabilan, tehnološki nezavistan način [4]."

SOA nije prost skup tehnoligija, jer ovaj koncept nije direktno vezan bi za jednu tehnologiju. Web servisi su najprikladnija tehnologija za realizaciju SOA, jer SOA treba da zadovolji slijedeće koncepte:

- Servisi pružaju poslovnu funkcionalnost, poput aplikacije za poslovna putovanja ili aplikacije za davanje kredita itd.
- Interfejsi - korisnici pristupaju servisima preko interfejsa. Interfejs treba da je odvojen od implementacije, on mora da bude nezavistan od platforme. Svaki interfejs definiše skup operacija, koje treba da budu pravilno granulisane.
- Poruke - operacije se definišu kao skupovi poruka. Poruke navode podatke koji treba da budu razmjenjeni a opisuju ih na platformski nezavistan način. Servisi razmjenjuju samo podatke i u tome se bitno razlikuju od objektno orjentisanog pristupa.
- Sinhronost - korisnici usluga pristupaju servisima preko servisnog busa (sabirnice). To može biti neki transportni protokol kao što je SOAP ili ESB. Korisnici servisa mogu koristiti sinhrone ili asinhrone načine komunikacije da bi pozvali neki servis. U sinhronom režimu rada, servis vraća odgovor korisniku nakon što završi sa obradom. U ovom slučaju, korisnik servisa mora da čeka dok servis ne završi obradu. U asinhronom načinu rada, korisnik ne čeka da mu servis odgovori, iako servis može da obavjesti korisnika da je uzeo njegov zahtjev u proces obrade.
- Labavo spregnuti servisi su oni servisi koji javno izlažu samo neophodne zavisnosti. Ovo je posebno važno u slučaju frekventnih promjena, jer će minimalne javno izložene zavisnosti u slučaju izmjene jednog servisa minimalno uticati na potrebu izmjene drugih servisa.
- Registri - da bi olakšali automatsko pretraživanje odgovarajućeg servisa, servisi se održavaju u odgovarajućim registrima, koji se ponašaju kao listinzi direktorija. UDDI je jedan primjer takvog direktorija.
- QoS - kvalitet servisa. Servisi obično imaju pridružene određene QoS atribute. Takvi atributi uključuju sigurnost, pouzdanu razmjenu poruka, transakcije, menadžment i druge zahtjeve. Kada govorimo o Web servisima, QoS atributi su često pokriveni tzv. WS specifikacijama, poput WS-security, WS-Addressing, WS-Coordination itd.
- Kompozicija servisa u biznis proces. Zadnji i vjerovatno najvažniji SOA koncept je kompozicija servisa u biznis proces. Servisi se komponuju odgovarajućim redoslijedom i prate određeni skup pravila da bi omogućili podršku biznis procesima. Kompozicija servisa omogućava nam da omogućimo podršku za biznis procese na relativno jednostavan način. Kompozicija nam takođe daje relativno jednostavan način da izmjenimo biznis procese i da na taj način pružimo podršku promjenama u poslovnom okruženju na jednostavan i brz način. Za realizovanje kompozicije koristi se jezik koji je posvećen rješavanju ovog problema - BPEL i odgovarajući engine na kojem biznis proces definicije mogu da se izvršavaju. Tek kada dostignemo nivo kompozicije servisa, moći ćemo da iskoristimo sve prednosti SOA [4].

IX. PROBLEM HETEROGENOSTI I XML

Razvoj SOA i servisa (poput Web servisa) duguje mnogo shvatanju da postoji problem heterogenosti tehnologija u velikim preduzećima. Ovaj problem odnosi se na činjenicu da u velikim preduzećima ili u sistemima koji moraju da povezuju više preduzeća uvijek postoji više različitih tehnologija koje su korištene za razvoj aplikacija. Moguće je da će se u takvim sistemima pojaviti različite vrste tehnoloških heterogenosti, poput [3]:

- Heterogenost posredničkog (middleware) softvera - u velikim preduzećima obično se koristi više vrsta posredničkog (middleware) softvera.
- Heterogenost protokola. Ova vrsta heterogenosti odnosi se na različite transportne protokole koji se moraju koristiti za pristup servisima različitih aplikacija.
- Heterogenost upotrebljenih načina sinhronizacije. Gotovo uvijek postoji potreba da se pruži podrška i za sinhrone i za asinhrone interakcije među aplikacijama. Zbog toga se često javljaju situacije u kojima stilovi interakcije koji su podržani od strane različitih aplikacija nisu kompatibilni.
- Raznolikost tipova podataka. Slijedeći problem se javlja kada dođe do pojave raznovrsnosti u vrstama formata podataka koji moraju da se razmjenjuju. U većini slučajeva, podaci zavise od vrste posredničkog (middleware) softvera koji se koristi.
- Raznolikost deklaracija interfejsa. Problem se u velikim organizacijama može javiti i kada se pojavi razlika u na-

činu na koji su deklarisani i na koji se koriste interfejsi servisa. Npr. načini na koji su deklarisani interfejsi u CORBA i Java RMI konceptima se razlikuju.
- Nepostojanje uobičajenog mjesta za pretragu servisa. Još jedan problem može da se javi kada ne postoji zajedničko mjesto na kojima se mogu tražiti servisi

Još jedan u nizu problema koji se može javiti u ovakvim organizacijama je da sa svakom novom verzijom softvera dobavljača (servera) korisnička aplikacija (klijent) mora da se mijenja. Svi ovi problemi koji su vezani za raznolikosti i proširenja su djelimično riješeni pomoću razvoja novih tehnologija i standarda. Standardi obezbjeđuju bazu za istovjetnost i omogućavaju široku prihvaćenost kroz interoperabilnost. Primjeri ovakvih standarda uključuju zajednički komunikacioni jezik (XML), zajednički format za razmjenu poruka (SOAP), zajednički format za specifikaciju servisa (WSDL) i zajedničko sredstvo za pretragu servisa (UDDI). Najvažniji među navedenim standardima je XML, jer on formira bazu na kojoj je izgrađena većina drugih standarda [3].

XML je prihvaćen kao popularan middleware-nezavistan standard kao format za razmjenu poruka i tabela. XML je u stvari najmanji zajednički sadržalac oko kojeg je IT industrija mogla da se složi. Za razliku od CORBA, IDL i Java interfejsa, XML nije vezan ni za jednu posebnu tehnologiju ni middleware standard, pa se danas često koristi kao ad-hoc format za obradu podataka između različitih a često nekompatibilnih middleware platformi. XML je besplatan, a dolazi uz veliki broj open-source API-ja i na mnogo različitih platformi, uključujući i brojne open-source API-je za parsiranje kao što su SAX, StAX i DOM. Ovi alati omogućavaju obradu i upravljanje XML dokumentima. Još jedna prednost XML-a je da on zadržava strukturu svojih podataka i u tranzitu. Kao dodatak svemu ovome, XML je vrlo fleksibilan, a ta fleksibilnost ga pozicionira kao najpogodniji standard za rješavanje problema heterogenosti aplikacija i posredničkog (middleware) softvera [3].

XML je vjerovatno jedan od najvažnijih standarda na kojima su izgrađeni Web servisi. XML dokumenti se često koriste kao sredstvo za pružanje informacija između davaoca usluga i njihovog konzumenta. XML takođe predstavlja i osnovu za WSDL (Web Services Description Language) jezik, koji se koristi za deklaraciju interfejsa koji Web servise predstavljaju konzumentima. XML takođe stoji u osnovi SOAP protokola za pristupanje Web servisima. Na kraju, mora se spomenuti da je i UDDI (Universal Description, Discovery and Integration) koji se koristi za objavljivanje i otkrivanje Web servisa takođe zasnovan na XML-u.

X. SOAP I PROBLEM RAZMJENE PORUKA

Iako je razvoj XML-a bio veoma važan za rješavanje problema heterogenosti, XML sam po sebi nije bio dovoljan da bi dvije strane (apliakcija davalac usluga i apliakcija konzument) mogle ispravno da komuniciraju. Da bi komunikacije bile efikasne, učesnici u komunikaciji morali su da budu u stanju da razmjenjuju poruke prema unaprijed utvrđenom formatu. Simple Object Access Protocol (SOAP) je takav protokol koji obezbjeđuje zajednički format poruka za sve servise. SOAP je tekstualno baziran format za enkodiranje podataka. On je nezavistan i od programskog jezika i od platforme na kojoj se izvršava, ne zahtjeva nikakvu specifičnu tehnologiju u komunikacionim tačkama, što ga čini potpuno nezavisnim od dobavljača software-a, platformi i tehnologija. Njegov tekstualni format čini ga takođe i firewall - friendly protokolom. Iako je ovaj protokol prvenstveno razvijen da radi samo sa HTML-om, bilo koji transportni protokol ili middleware za prenos poruka može da bude iskorišten da prenosi SOAP poruke [3].

XI. WSDL – WEB SERVICE DESCRIPTION LANGUAGE

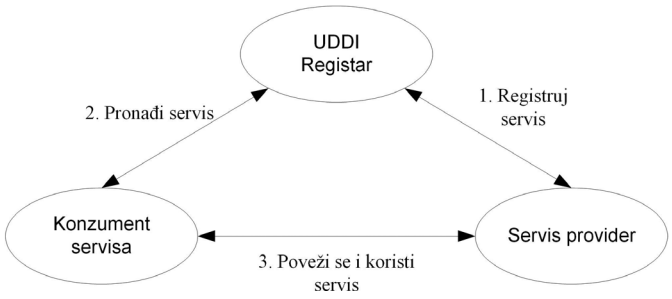
WSDL (Web Services Description Language) je jezik zasnovan na XML-u koji je namjenjen za opisivanje interfejsa i drugih karakteristika Web servisa. Ovo je druga primjena XML-a kojom se rješava problem heterogenosti koji je ranije pominjan. WSDL nudi slijedeće prednosti u odnosu na neke ranije spominjane tehnologije [3]:

- Za razliku od CORBA-inih IDL datoteka i datoteka iz RPC specifikacije, WSDL je mnogo više nezavistan od programskih jezika i middleware tehnologija. Ovo je direktan proizvod činjenice sa je WSDL zasnovan na XML-u, što ga čini pogodnim za skoro svaki tip servisa.
- WSDL nam pruža metod za navođenje komunikacionog protokola za pozivanje servisa. Zahvaljujući tome, servisi su slobodni da konvencionalno implementiraju bilo koji protokol koji će dalje pozivati WSDL protokole.
- WSDL nam takođe pruža način da navedemo format poruke za komunikaciju sa navedenim servisom. Zbog toga, servis je slobodan da izabere bilo koji pogodan format poruke. Jedan primjer formata poruke je SOAP.
- WSDL takođe pruža aplikacijama koje dobavljaju servise veliku širinu u navođnju servisnih operacija koje one pružaju. Generalno, moguće je specificirati četiri različite vrste operacija, ukllјučujući i sinhrone i asinhrone operacije.
- Na kraju, WSDL ima mogućnost navođenja krajnje tačke servisa. Generalno, krajnja tačka servisa (service end-point) je mrežna adresa na kojoj je servis dostupan za pozivanje [3].

XII. UDDI REGISTAR

Pored deklaracije interfejsa (WSDL) i formata SOAP poruka, velikim organizacijama je potrebno i centralno mjesto na kojem dobavljač usluga (service provider) mogu da objave svoje servise pomoću WSDL-a i na kojem konzumenti servisa mogu da pronađu tražene servise. Ovo se radi najviše zbog činjenice da u velikim organizacijama resursi mogu biti geografski dislocirani. Takvo centralno mjesto se naziva registar (registry). UDDI (Universal Description, Discovery and Integration) specifikacija definiše standardan način za registrovanje, deregistrovanje i potragu za servisima. Na slijedećoj slici

prikazan je način na koji UDDI dozvoljava dinamičko otkrivanje, opisivanje i integraciju servisa [3].



Slika 1: *Način rada UDDI registra*

Važnost UDDI registra za Web servise je slična ulozi koju pretraživači igraju na internetu. Snaga pretraživača dolazi od ključnih riječi koje se koriste za klasifikaciju sadržaja. Na sličan način, fino podešena pretraga Web servisa moguća je jedino ako su servisi ispravno klasifikovani. Klasifikacija i identifikacija taksonomije predstavljene u UDDI registru daje polaznu osnovu za opisivanje Web servisa.

XIII. BPM – UPRAVLJANJE POSLOVNIM PROCESIMA U OKVIRU SOA

Cilj i osnovna namjena SOA je da omogući da poslovanje postane lakše za upravljanje, fleksibilnije i da počne da brže odgovara na promjene. Preduzeća konstantno mijenjaju načine na koje obavljaju određene djelatnosti, pri čemu najčešće ne mijenjaju finalni proizvod. Npr. osiguravajuće društvo može promjeniti metode koje koristi da bi obradilo žalbe kupaca, ali bez obzira na to, ono i dalje prodaje osiguranje. SOA omogućava poslovnim ljudima da promijene poslovne procese bez potrebe da se fokusiraju na podređenu tehnologiju. SOA nam omogućava da se koncentrišemo na dizajniranje i poboljšanje poslovnih procesa pomoću upredanja različitih poslovnih servisa. BPM (Business Process Management) je moderan pristup dizajniranju i upravljanju poslovnim procesima. Sam po sebi, BPM je u velikoj mjeri uticao da se poslovanje odvoji od tehnologije. A kada je povezan sa SOA, BPM je je još dodatno osnažen [5].

Ono što mi danas nazivamo BPM je zapadna adaptacija najboljih praksi koje su evoluirale prvenstveno iz japanskih iskustava iz upravljanja proizvodnjom. Najbliži ekvivalent japanskom terminu je Kaizen koji se može definisati kao „neprekidno poboljšavanje" ili slobodnije „rastaviti dijelove i ponovo ih složiti na bolji način". Osim neprekidnog poboljšanja, BPM obuhvata i druge metode upravljanja, poput TQM i Six Sigma [5].

BPM je jedna od tehnologija koja je najavljivana sa fanfarama i sa velikim očekivanjima. Proizvođači koji su nudili ovakva rješenja predstavljali su ovu tehnologiju kao revolucionarno poboljšanje u načinu na koji će se poslovne aplikacije dalje razvijati. Na žalost, kao i u slučaju mnogih drugih obećavajućih tehnologija, nakon mnogih neispunjenih očekivanja od nje se odustalo sa razočarenjem. Međutim, od momenta od kada je ova tehnologija povezana sa SOA, koristi od njenog korištenja su počele da postaju opipljive [1].

BPM (Business Process Management) je skup tehnologija koje omogućavaju kompaniji da izgradi, obično kroz vizualno prikazan tok (korak po korak), izvršenje nekog procesa koje se proteže duž nekoliko organizacija ili sistema. Prije BPM-a, takvi sistemi su manje elegantno nazivani mašinerije za obradu toka. Optimistička izjava Howarda Smitha i Petera Fingera da BPM ne samo da ubrzava razvoj aplikacija već i eliminiše potrebu za razvojem [1], je prije pojave BPEL-a bila samo još jedno neispunjeno obećanje.

XIV. BPEL – KARIKA KOJA JE NEDOSTAJALA

BPEL (Business Process Execution Language) je jezik zasnovan na OASIS standardima a realizovan u XML-u, koji takođe predstavlja i osnovu za SOA. BPEL dozvoljava programerima da definišu poslovne procese tako da različiti poslovni servisi mogu da budu povezani ili orkestrirani zajedno da bi dovršili neki poslovni zadatak [5].

BPEL se može posmatrati kao podskup BPM koncepta. BPEL nam daje semantički bogat jezik za kreiranje biznis procesa koji se sastoje od Web servisa baziranih na SOAP tehnologij. BPEL je u stvari specifikacija koja navodi kako da materijalizujemo poslovni proces koji se sastoji od Web servisa baziranih na SOAP-u. BPEL sam po sebi nema specifične odredbe za izvršavanje ljudskih aktivnosti koje se obično povezuju sa BPM procesima baziranim na toku. BPEL standard se takođe ne koncentriše ni na kreiranje izvještaja, analizu ni monitoring, iako pojedini dobavljači BPEL-a pokušavaju da svoje proizvode obogate ovakvim konceptima. Kada bi morali da formulišemo razliku između termina BPM i BPEL, mogli bi reći da je BPM koncentrisan na kompletnu ponudu, dok BPEL predstavlja standard za orkestraciju Web servisa [1].

BPEL koristi riječnik zasnivan na XML-u da bi naveo i opisao poslovne procese. BPEL verzija 1.1 je zasnovan na XML, WSDL, XML Schema i XPATH specifikacijama. Poznavanje ovih specifikacija u velikoj mjeri pomaže pri učenju BPEL-a [4].

BPEL (Business Process Execution Language for Web Services, takođe i WS-BPEL, BPEL4WS) je jezik koji se koristi za kompoziciju, orkestraciju i koordinaciju Web servisa. On posjeduje bogat riječnik za opisivanje ponašanja poslovnih procesa [4].

U smislu Web servisa, interakcija između klient aplikacije i servisa treba da bude izolovana i jednostavna. Ovakve aktivnosti ne zadržavaju stanje i rezultiraju nepovezanim pozivima servisa. Ako neki program ili aplikacija pozove servis A, a zatim isti pozove servis B, tada Web servis B nema nikakvog znanja o tome šta se desilo u Web servisu A. Međutim, u mnogin drugim scenarijima interakcija klijenta sa servisom nije tako jednostavna. Takav je slučaj sa mnogim, ako ne i sa svim poslovnim procesima. Poslovni proces je kolekcija povezanih, strukturiranih aktivnosti ili zadataka koji proizvode specifičan proizvod (služe određenoj svrsi). Pošto poslovni proces uključuje kompleksne, povezane i struktuirane aktivnosti, on zahtje-

va okruženje koje je u stanju da upamti vlastito stanje da bi mogao da poziva niz Web servisa koji implementiraju poslovni proces. Samim tim potreban je i model za opisivanje kompleksnih razmjena koje karakterišu poslovne interakcije, kako kratkotrajne tako i one dugotrajne interakcije. Primjer poslovnog procesa koji zahtjeva dugotrajnu interakciju je narudžba nekog proizvoda. Ovaj poslovni proces počinje u momentu kada se primi narudžba, a prestaje kada se isporuči roba. Business Process Execution Language (BPEL) for Web Services (ili BPEL4WS) je jezik za opisivanje takvih dugotrajnih interakcija koje su svjesne vlastitog stanja. Prema BPEL reprezentaciji, poslovni proces se predstavlja kao kolekcija povezanih poziva poziva servisa i povezanih aktivnosti koje proizvode rezultat bilo u jednom preduzeću ili više njih [3].

BPEL4WS ili jednostavno BPEL je zasnovan na tri XML specifikacije: WSDL, XML Schema i XPath, kako je to prikazano na slijedećoj slici [3]:

Slika 2: *Odnos između BPEL-a, XML-a, XML Schema-e, XPath-a i WSDL-a*

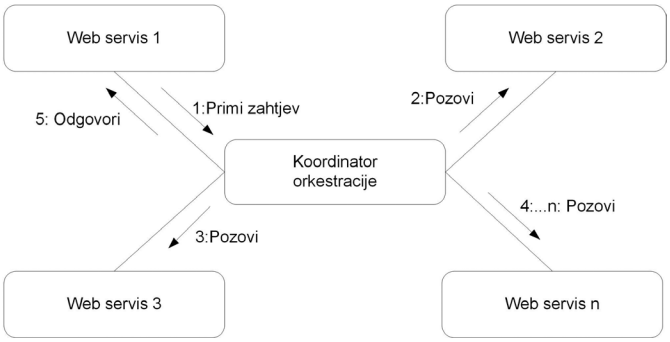
WS-BPEL 2.0 predstavlja unapređenje BPEL 1.1 standarda koje donosi veliki broj unapređenja. Unapređenja su vezana za pritup i manipulaciju podacima, nove aktivnosti (forEach i repeatUntil), poboljšano upravljanje greškama i porukama, unapređenja sintakse (if elseif else) I druga [6].

XV. ORKESTRACIJA I KOREOGRAFIJA

U zavisnosti od zahtjeva, kompozicija servisa može da se odnosi na privatne ili javne procese za šta se koriste dva pojma:

- Orkestracija i
- Koreografija.

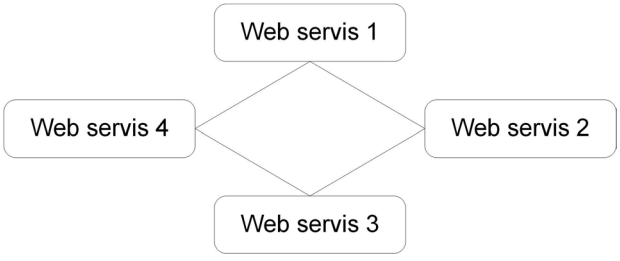
U orkestraciji, centralni proces (koji može da bude još jedan Web servis) kontroliše ponašanje ostalih Web servisa i koordinira izvršenje različitih operacija u Web servisima koji su uključeni u operaciju. Uključeni Web servisi ne znaju (i ne moraju da zaju) da su uključeni u kompoziciju i da su dio višeg poslovnog procesa. Samo centralni koordinator orkestracije zna za ovo, tako da je orkestracija centralizovana sa eksplicitnom definicijom operacija i redoslijedom pozivanja Web servisa. Orkestracija se obično koristi u poslovnim procesima, a šematski može da bude prikazana kako slijedi [4]:



Slika 3: *Primjer orkestracije*

Orkestracija servisa je ključna komponenta koja je dovela do toga da koncept SOA postane funkcionalan. Koncept orkestracije servisa funkcioniše putem razmjene poruka na nivou poslovnih aplikacija. Kako da se pojedini servisi nisu programirani za komunikaciju s drugim servisima, poruke se moraju razmjenjivati prema predodređenoj (programiranoj) poslovnoj logici i redoslijedu izvršenja kako bi svi zajedno mogli da funcionišu u skladu sa zahtjevima korisnika. Iz perspektive korisnika se čini da se umjesto niza pojedinačnih servisa izvršava jedna funkcionalna aplikacija. Orkestracija servisa predstavlja proces u kojem se integriše dvije ili više različitih aplikacija i/ili servisa radi automatizacije izvršenja nekog procesa.

S druge strane, koreografija se ne oslanja na jednog centralnog koordinatora. Umjesto toga, svaki Web servis umješan u koreografiju zna tačno kada treba da pokrene svoje operacije i sa kim treba da bude u interakciji. Koreografija je kolaborativni napor koji je fokusiran na razmjenu poruka u javnim poslovnim procesima. Svi učesnici u koreografiji treba da budu svjesni postojanja poslovnog procesa, operacija koje treba da izvrše, poruka koje moraju da razmjene kao i tajminga za razmjenu poruka. Koreografija u kompoziciji Web servisa je prikazana na slijedećoj slici [4]:



Slika 4: *Primjer koreografije*

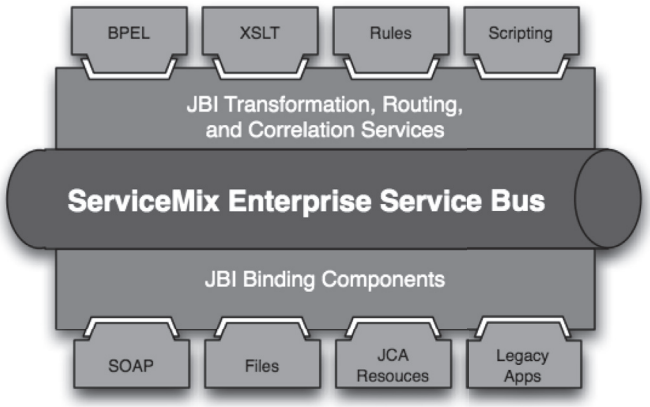
Iz perspektive komponovanja Web servisa koji treba da izvršavaju neki poslovni proces, orkestracija ima određene prednosti pred koreografijom. Iako je vrlo tanka linija koja razdvaja ova dva pojma, orkestracija predstavlja mnogo fleksibilniju paradigmu od koreografije. Orkestracija ima slijedeće prednosti [4]:

- Tačno znamo ko je odgovoran za izvršenje cijelog poslovnog procesa;

- Možemo da inkorporiramo Web servise, čak i one koji nisu svjesni da su dio poslovnog procesa;
- Možemo da obezbjedimo alternativne scenarije kada dođe do otkaza sistema.

XVI. DOSTUPNE OPEN SOURCE TEHNOLOGIJE

Danas postoje brojna komercijalna rješenja koja omogućavaju implemetaciju koncepta orkestracije servisa. Na tržištu postoji veći broj BPEL dizajnera, gdje se osim Eclipse između ostalih ističu proizvođači Oracle, IBM, Microsoft, Red Hat, Intalio, Active endpoints, Attachmate Verastream i drugi. Dio ovih kompanija nudi i vlastita integrisana rješenja u vidu ESB (Enterprise Service Bus) arhitektura. ESB arhitektura koristi se za dizajniranje i implementaciju interakcije između softverskih apliakcija koje se nalaze u interakciji u okviru SOA. Najpoznatija ovakva rješenja poznatih proizvođača obuhvataju Oracle Enterprise Service Bus (uključen u Oracle SOA Suite), zatim IBM Integration Bus ko naslijednik IBM WebSphere ESB proizvoda, Microsoft BizTalk Server, JBoss Enterprise Service Bus (u sastavu JBoss Enterprise SOA Platform) i Apache service mix. Nedostatak nekih od ovakvih integrisanih rješenja ponekad leži u pokušaju razvoja vlastitih rješenja (Microsoft korsiti vlastiti jezik xLANG umjesto WS-BPEL jezika definisanog OASIS standardom). Veliki nedostatak nekih od navedenih rješenja jeste cijena kompletnih paketa ili cijena nekih njihovih dijelova. Zbog toga treba razmisliti o mogućnostima upotrebe open source rješenja, alata i tehnologija koji se mogu koristiti za implementaciju orkestracije servisa.



Slika 5: *ESB Apache Service Mix-a [7]*

Iako open source rješenja zahtjevaju daleko manja ulaganja pri implementaciji, najveći motiv za navedeni način pristupa implementaciji jeste sticanje znanja i potpuniji uvid u način funkcionisanja te koje su nam komponente potrebne za implementaciju orkestracije. Zbog toga vrijedi razmisliti o izboru skupa open source alata i tehnologija predvođen Apache ODE (Orchestation Director Engine) i Eclipse BPEL dizajnerom, zajedno sa Javom, Apache Axis2 framework-om, nekog web servera poput Apache Tomcat servera i npr. MySQL ili MariaDB sistema za upravljanje bazama podataka. U nastavku će biti opisane Apache ODE, Axis 2 i Eclipse BPEL tehnologije. Opis nekog web servera poput Tomcata ili nekog sistema za upravljanje bazama podataka prevazilazi ovire jednog ovakvog teksta.

XVII. APACHE ODE (ORCHESTRATION DIRECTOR ENGINE)

Apache ODE (Orchestation Director Engine) predstavlja implementaciju OASIS-ovog WS-BPEL 2.0 (Web Services Business Process Execution Language) i BPEL4WS 1.1 standarda [8]. Apache ODE komunicira sa jednim ili više Web servisa na način koji je opisan WS-BPEL jezikom. Principijelno, ODE komunicira sa Web servisima na način da im šalje poruke, manipuliše podacima i upravlja izuzecima (exceptions). Apache ODE (Orchestation Director Engine) izvršava poslovne procese koji su pisani u skladu sa WS-BPEL 2.0 standardom.

Mašinerija na kojoj je zasnovan Apache ODE posjeduje dva sloja (dva načina) za komunikaciju sa Web servisima:

- Sloj zadužen za integraciju sa Apache Axis serverom, koji će ovdje biti opisan i
- Sloj zasnovan na JBI standardu koji omogućava komunikaciju pomoću JBI poruka. Java Business Integration (JBI) je specifikacija koja je razvijena unutar Java Community Process (JCP) kao pristup u implementaciji servisno orjentisane arhitekture (SOA). JBI rješava probeme integracije tako što omogućava third-party aplikacijama i komponentama da se uključe u standardnu infrastrukturu i da međusobno sarađuju na predvidiv način, bez obzira na činjenicu da su ih proizveli različiti proizvođači [9].

Pored navedenog, u Apache ODE ugrađena je podrška za HTTP WSDL povezivanje, mapiranje internih varijabli u tabele baza podataka, isporuka procesa bez resetovanja (hot-deployment). U okviru Apache ODE projekta kreiran je veliki broj proširenja WS-BPEL jezika koji obuhvataju oporavak u slučaju greške, proširenja XPath i XQuery dijelova, kreiranje RESTful BPELa, proširenja forEach iteratora i konteksta u kom se procei izvršavaku.

XVIII. APACHE AXIS I APACHE AXIS 2

Apache Axis je open source Web service framework koji je baziran na XML jeziku. On se sastoji od Java i C++ implementacija SOAP servera i od različitih korisničkih programa i API-ja za generisanje i isporuku (deployement) Web servisa. Pomoću Apache Axis-a, programeri mogu da razvijaju inetroperabilne i distribuirane Web aplikacije.

Apache Axis 2 je kompletno redizajnirana i ponovo napisana verzija prethodno široko prihvaćene verzije ovog frameworka. Axis 2 ne samo da omogućava da Web servisi komuniciraju sa Web aplikacijama, već on istovremeno funkcioniše kao samostalna serverska aplikacija.

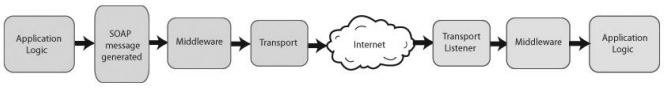
Apache Axis 2 je implementacija i klijent i server jednačine Web servisa. Dizajniran da iskoristi lekcije koje su naučene iz Apache Axis 1.0, Axis 2.0 nam pruža objektni model i modularnu arhitekturu koja olakšava dodavanje funkcionalnosti i

podrške novih preporuka i specifikacija koje su vezane za Web servise [10].

Apache Axis 2 omogućava lako izvršenje slijedećih zadataka:

- Slanje SOAP poruka,
- Prijem i obradu SOAP poruka,
- Kreiranje Web servisa iz čistih Java klasa,
- Kreiranje implementacionih klasa za klijenta i servera pomoću WSDL-a,
- Jednostavno preuzimanje WSDL-a kao servisa,
- Slanje i primanje SOAP poruka sa prilozima
- Usklađenost sa WS-Security, WS-ReliableMessaging, WS-Addressing, WS-Coordination i WS-Atomic Transaction preporukama.

Da bi shvatili Axis2, moramo razumjeti životni ciklus jedne Web servis poruke. Tipično, ona izgleda ovako:

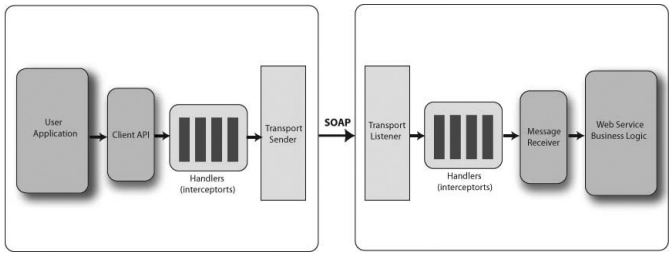


Slika 6: *Životni ciklus neke poruke Web servisa [10]*

Aplikacija koja šalje poruku kreira originalnu SOAP poruku koja je, kako je već prikazano, u stvari XML poruka. Ako sistem zahtjeva korištenje WS* preporuka, poruka može proći kroz dodatnu obradu prije napuštanja pošiljaoca. U momentu kada je poruka spremna, ona se šalje putem nekog transportnog protokola, poput HTTP ili JMS.

Poruka tako stiže do prijemnika, koji njen dolazak osluškuje pomoću tzv. transport listener-a. Ako aplikacija nema pokrenut HTTP listener, ona neće moći da primi nikakve HTTP poruke. Ako je poruka dio nekog sistema koji koristi WS-Security i slične preporuke, ona će biti dodatno obrađena. Na kraju, dispečer određuje specifičnu aplikaciju ili neku drugu komponentu (npr, Java metod) kojem je poruka namjenjena i šalje je toj komponenti.

Axis2 upravlja radom pošiljaoca i prijemnika u jednoj transakciji. Iz Iz Axis2 perspektive, struktura izgleda kao na slici [10]:



Slika 7: *Slanje SOAP poruke*

Na svakoj strani nalazi se po jedna aplikacija koja je dizajnirana da upravlja (poslanim i primljenim) porukama, dok se u sredini nalazi Axis2. Vrijednost Web servisa ogleda se u činjenici da pošiljalac i primalac čak ne moraju ni da se izvršavaju na istoj platformi. Pod pretpostavkom da se Axis2 izvršava na obje strane, proces izgleda ovako:

- Pošiljalac kreira SOAP poruku,
- Axis2 upravljači (handlers) nad tom porukom izvršavaju neophodne akcije, npr. šifrovanje ako se zahthjeva WS-Security,
- Transport sender šalje poruku,
- Na strani prijemnika, transport listener detektuje poruku,
- Transport listener proslijeđuje poruku odgovarajućim handlerima na prijemnoj strani,
- Jednom kada je poruka obrađena u pre-dispeč fazi, ona dolazi do dispečera koji je šalju odgovarajućoj aplikaciji.

U okviru Axis2 servera ove akcije su podijeljene u faze poput „pre-dispatch", „dispatch", i „message processing".

XIX. ECLIPSE BPEL DIZAJNER

Cilj Eclipse BPEL projekta je da u Eclipse IDE doda opsežnu podršku za definisanje WS-BPEL 2.0 procesa, te za njihovo kreiranje, uređivanje, implementaciju, testiranje i ispravljanje grešaka na njima [11]. Eclipse BPEL Designer je softverski paket koji je pisan u programskom jeziku Java korišćenjem Eclipse EMF (Eclipse Modeling Framework) i GEF (Graphical Editing Framework) okruženja, pri čemu je realizovana kao skup Eclipse dodataka (plug-ins). Eclipse BPEL dizajner pruža nam mogućnost da grafički modelujemo BPEL procese, uz statičku validaciju i interfejse za uljučenje različitih BPEL runtime okruženja [12].

Ključne osobine Eclipse BPEL dizajnera su [11]:

- Editor zasovan na GEF (Graphical Editing Framework) okruženju.
- EMF Model zasnovan na WS-BPEL 2.0 specifikaciji.
- Validator koji provjerava EMF model i po potrebi generiše greške i upozorenja.
- Runtime okruženje koje omogućava isporuku i izvršavanje BPEL procesa.
- Okruženje koje omogućava dibagiranje, odnosno izvršavanje korak po korak uz upotrebu prekidnih tačaka (breakpoints).

XX. DRUGA OPEN SOURCE RJEŠENJA

Red Hat JBoss Middleware predstavlja čitav portfolio Middleware aplikacija za integraciju softverskih proizvoda. U okviru JBoss Tools [13] projekta razvijen je JBoss BPEL editor koji zadovoljava zahtjeve WS-BPEL 2.0 specifikacije, a nastao je kao proširenje Eclipse BPEL editora. JBoss koristi RiftSaw [14] open source runtime BPEL implementaciju zasnovanu na Apache ODE softveru koji se izvršava na JBoss Application Serveru.

OpenESB [15] je open source implementacija Enterprise Service Bus tehnologije zasnovana na programskom jeziku Java. Ovo rješenje je nastalo kao projekt u okviru Sun korporacije, ali je nakon spajanja kompanija Oracle i Sun Microsystems dalji razvoj preuzela OpenESB Community i Pymma kompanija. OpenESB je rješenje koje se oslanja na JBI (Java Business Integration), XML, XML Schema, WSDL i BPEL je-

ziku da bi kreirao platformu za integraciju enterprise aplikacija i orkestraciju servisa. OpenESB se sastoji od OpenESB Studio IDE, OpenESB instance i OpenESB komponenata. OpenESB Studio IDE razvojno okruženja zasnovano je na NetBeans IDE okruženju. Ono može da se koristi za razvoj klasičnih Java aplikacija, ali i većeg broja priključaka (plug-ins) koji služe za razvoj XML, XML schema, WSDL, BPEL i drugih dokumenata. OpenESB instance je ključni dio ovog paketa koji služi kao server koji od verzije 3.0, za razliku od prethodnih verzija, više ne zahtjeva instalaciju Glassfish servera.

Orchestra [16] je još jedan open source projekt koji je namjenjen za orkestraciju web servisa koji organizuje Web servise u skladu sa BPEL XML gramatikom. Isporučuje se sa Tomcat serverom i koristi Web konzolu za pristup konfiguraciji servera, procesima i BPEL editoru.

Petals ESB [17] je open-source Enterprise Service Bus (ESB) koji razvija OW2 Middleware konzorcijum. Petals ESB je Java platforma zanovana na SOA principima koja je namjenjena za povezivanje heterogenih sistema, aplikacija i usluga.

XXI. BPEL EKSTENZIJE

Iako postoje brojna komercijalna rješenja koja omogućavaju implemetaciju koncepta orkestracije servisa, u ovom tekstu je dat pregled skupa open source alata i tehnologija predvođen Apache ODE (Orchestation Director Engine) i Apache Axis2 framework-om te Eclipse BPEL dizajnerom. Prikazana rješenja implementiraju osnovu WS-BPEL 2.0 specifikacije. Međutim, veći broj organizacija i nezavisnih istraživača u svijetu objavio je proširenja WS-BPEL specifikacije. U okviru OASIS organizacije osnovan je tehnički komitet koji radi na razvoju WS-BPEL Extension for People (BPEL4People) i WS-HumanTask specifikacija koje treba da obezbjede da i ljudi mogu da uzmu učešće u poslovnim procesima i da mogu da utiču na tok izvršenja procesa [18]. Da bi riješili Bring-Your-Own-Device (BYOD) zahtjev prema kojem zaposleni sve češće koriste svoje vlastite mobilne uređaje u rješavanju zadataka, grupa istraživača na Univerzitetu u Pekingu razvila je novi middleware pod imenom MUIT koji donosi mobilnost (**M**obility) interakcije sa korisnicima (**U**ser **I**nteractions) i zadatke (**T**asks) u BPEL. MUIT ddonosi vlastit domenski jezik (Domain-Specific Language) koji podržava izradu adaptibilnih, mobilno orjentisanih Web interfejsa u WS-BPEL [19] [20]. Pored ranije pomenutih radova na proširenju WS-BPEL specifikacije koji su realizovani u okviru Apache ODE projekta, postoji i veliki broj drugih unapređenja specifikacije, od kojih treba posebno izdvojiti radove u okviru kompanija IBM i Oracle. U okviru kompnije IBM razvijeno je proširenje ii4BPEL koje integriše SQL iskaze u BPEL, zatim proširenje pod imenom Generalizovani tok, koje omogućava proizvoljno povezivanje aktivnosti, proširenje za izvršavanje podprocesa, kao nastavak rada na BPEL-SPE ekstenziji [21]. Takođe, u kompaniji IBM razvijen je dodatni tip procesa (microflow), dodatni jezik za pisanje izraza (Java) i dodatne akcije (Human Task, Snippet, Generalized Flow, Collaboration scope) [22]. U okviru kompanije Oracle razvijen je veliki broj vrsta aktivnosti (Assert, Bind Entity, Create Entity, Dehydrate, Email, FlowN, IM, Java Embedding, Phase, Receive Signal, Replay, Signal, SMS, Transform, Voice) kao proširenje WS-BPEL specifikacije [21] [23].

XXII. ZAKLJUČAK

U ovom tekstu predstavljen je veliki broj tehnika, tehnologija i standarda koji su doveli do pojave koncepata servisno orjentisane arhitekture (SOA) i orkestracije servisa. U tekstu je predstavljen istorijski razvoj dizajna sistema koji se koristi za integrisanje softverskih aplikacija, od mainframe računara i klijent server arhitekture, preko RPC (Remote Procedure Call) koncepta i CORBA, DCOM, EJB i SOAP tehnologija. Svi pomenuti koncepti i tehnologije doveli su do razvoja koncepta web servisa i servisno orjentisane arhitekture. Pojava često veoma različitih web servisa sa sobom je donijela probleme heterogenosti korišćenih tehnologija, u smislu heterogenosti softvera, protokola, upotrebljenih načina sinhronizacije, raznolikosti tipova podataka, raznolikosti deklaracija interfejsa, te nepostojanja uobičajenog mjesta za pretragu servisa. Postojanje navedenih problema dovelo je do pojave WSDL jeizka za opis web servisa i UDDI registra koji definiše standardan način za registrovanje, deregistrovanje i potragu za servisima. U ovom momentu dolazi do pojeve BPM (Business Process Management) koncepta, kao skupa tehnologija koje omogućavaju izvršenje nekog procesa koje se proteže duž nekoliko organizacija ili sistema. Međutim, optimističke najave da BPM u velikoj mjeri ubrzava razvoj aplikacija su prije pojave programskog jezika BPEL spadale u sferu neispunjenih obećanja. WS-BPEL je dozvolio programerima da definišu poslovne procese tako da različiti poslovni servisi mogu da budu povezani ili orkestrirani zajedno da bi dovršili neki poslovni zadatak. Dakle, WS-BPEL je kao programski jezik omogućio orkestraciju Web servisa, odnosno proces u kojem se integrišu dvije ili više različitih aplikacija i/ili servisa radi automatizacije izvršenja nekog procesa. Orkestracija servisa je ključna komponenta koja je dovela do toga da koncept SOA postane funkcionalan.

Kako je prikazano u ovom tekstu, zahtjevi koji dolaze od korisnika te odgovori velikih kompanija iz IT industrije na navedene zahtjeve veoma često su vršili uticaj i oblikovali brojne standarde, specifikacije, protokole i tehnologije. Prikazana proširenja realizovana u okviru Apache ODE projekta, ali i rezultati istraživanja i implementacije proširenja koja su razvijena kako u okviru velikih kompanija tako i od strane nezavisnih istraživača gotovo sigurno će u velikoj mjeri uticati na specifikaciju ili dovesti do novih proširenja iste, te obezbjediti dalji razvoj koncepta orkestracije servisa i programskog jezika WS-BPEL.

REFERENCE

[1] J.Davis, Open Source SOA, Manning Publications Co.(2009)
[2] C.Ouyang, E.Verbeek, W.P.van der Aalst, S.Breutel, M.Dumas, A.H.M.ter Hofstede, Formal Semantics and Analysis of Control Flowin WS-BPEL, Science of Computer Programming, Volume 67 Issue 2-3, July, 2007, Pages 162-198, (2007)

[3] W.Roshen, SOA-Based Enterprise Integration: A Step-by-Step Guide to Services-Based Application Integration, The McGraw-Hill (2009)
[4] Matjaz B. Juric, Benny Mathew, Poornachandra Sarang, Business Process Execution Language for Web Services, Second Edition, Packt Publishing (2006)
[5] J.Hurwitz, R.Bloor, C.Baroudi, M.Kaufman, Service Oriented Architecture For Dummies, Wiley Publishing, Inc., (2007)
[6] Manoj Das & Alex Yiu, 2006, Business Process Management and WS-BPEL 2.0, An Oracle White Paper, Oracle Corporation, Redwood Shores, CA 94065, U.S.A., (2006)
[7] Axel Irriger, Apache ServiceMix, Methods & Tools magazine, Winter 2010 issue, (2010)
[8] Paul Brown, An Introduction to Apache ODE, InfoQ, Dostupno na: http://www.infoq.com/articles/paul-brown-ode, (2007)
[9] Ron Ten-Hove, Peter Walker, Java™ Business Integration (JBI) 1.0, Sun Microsystems, Inc., (2005)
[10] Apache Foundation, Apache Axis2 User's Guide, Dostupno na: https://axis.apache.org/axis2/java/core/docs/userguide.html, (2017)
[11] Eclipse Foundation, BPEL Designer project, Dostupno na: https://eclipse.org/bpel/, (2017)
[12] David Schumm, Dimka Karastoyanova, Frank Leymann, Jörg Nitzsch, On Visualizing and Modelling BPEL with BPMN, IEEE Proceedings of the 4th International Workshop on Workflow Management, IWWM2009, 4-8 May 2009, Geneva, Switzerland, (2009)
[13] Red Hat JBoss, JBoss Tools Documentation project, dostupno na: https://tools.jboss.org/features/bpel.html, (2017)
[14] Gary Brown, Kurt Stam, and Jeff Yu, RiftSaw 2.3.0.Final - Getting Started Guide, dostupno na: https://tools.jboss.org/features/bpel.html, (2017)
[15] Pymma Services, OpenESB Standalone Enterprise Edition Installation, Pymma Services 2014
[16] Orchestra Team, Orchestra User Guide, Bull SAS - OW2 Consortium, dostupno na: http://download.forge.objectweb.org/orchestra/Orchestra-4.8.0-UserGuide.pdf, (2011)
[17] Mohammed El Jai, Vincent Zurzak, PEtALS-SE-SCA, OW2 Consortium, (2009)
[18] Dave Ings, Luc Clément, Dieter König, Vinkesh Mehta, Ralf Mueller, Ravi Rangaswamy, Michael Rowley, Ivana Trickovic, WS-BPEL Extension for People (BPEL4People) Specification Version 1.1, OASIS BPEL4People Technical Committee, (2010)
[19] Xuan-zhe Liu, Mengwei Xu, Teng Teng, Gang Huang, Hong Mei, „MUIT: A Domain-Specific Language and its Middleware for Adaptive Mobile Web-based User Interfaces in WS-BPEL", IEEE Transactions on Services Computing, vol. , no. , pp. 1, 5555, doi:10.1109/TSC.2016.2633535, (2016)
[20] Xuanzhe Liu, Mengwei Xu, Gang Huang, Teng Teng, Zibin Zheng, Hong Mei: MUIT: A Middleware for Adaptive Mobile Web-based User Interfaces in WS-BPEL. CoRR abs/1602.09125 (2016)
[21] Oliver Kopp, Katharina Görlach, Dimka Karastoyanova, Frank Leymann, Michael Reiter, David Schumm, Mirko Sonntag, Steve Strauch, Tobias Unger, Matthias Wieland, Rania Khalaf, A Classification of BPEL Extensions , Journal of Systems Integration, Vol 2, No 4, DOI: http://dx.doi.org/10.20470/jsi.v2i4.103, (2011)
[22] IBM Corporation, IBM Business Process Manager documentation, Working with BPEL extensions dostupno na: https://www.ibm.com/support/knowledgecenter/en/SSFPJS_8.5.6/com.ibm.wbpm.wid.bpel.doc/topics/cextent.html, (2017)
[23] Anirban Ghosh, Mark Kennedy, Richard Smith, Carol Thom, and Savija Vijayaraghavan, Oracle® Fusion Middleware Developer's Guide for Oracle SOA Suite 11g Release 1 (11.1.1.6.3), Oracle Corporation, (2012)

Doc. dr Boris Damjanović, zaposlen u Visokoj školi Banja Luka College u Banja Luci.
Kontakt: boris.damjanovic@blc.edu.ba
Oblasti interesovanja: zaštita računarskih sistema, kriptografija, informacione tehnologije i njihova primjena, programiranje i jezici.

infoM

UPUTSTVO ZA PRIPREMU RADA

1. Tekst pripremiti kao Word dokument, A4, u kodnom rasporedu 1250 latinica ili 1251 ćirilica, na srpskom jeziku, bez slika. Preporučen obim – oko 10 strana, single prored, font 11.
2. Naslov, abstakt (100-250 reči) i ključne reči (3-10) dati na srpskom i engleskom jeziku.
3. Jedino formatiranje teksta je normal, bold, italic i bolditalic, VELIKA i mala slova (tekst se naknadno prelama).
4. Mesta gde treba ubaciti slike, naglasiti u tekstu (Slika1...)
5. Slike pripremiti odvojeno, VAN teksta, imenovati ih kao u tekstu, radi identifikacije, u sledećim formatima: rasterske slike: jpg, tif, psd, u rezoluciji 300 dpi 1:1 (fotografije, ekranski prikazi i sl.), vektorske slike – cdr, ai, fh,eps (šeme i grafikoni).
6. Autor(i) treba da obavezno priloži svoju fotografiju (jpg oko 50 Kb), navede instituciju u kojoj radi, kontakt i 2-4 oblasti kojima se bavi.
7. Maksimalni broj autora po jednom radu je 5.

Redakcija časopisa Info M