

SVEUČILIŠTE U ZAGREBU  
FAKULTET ORGANIZACIJE I INFORMATIKE  
VARAŽDIN

**Davor Sauer**

**DIPLOMSKI RAD**

Varaždin, 2009.



SVEUČILIŠTE U ZAGREBU  
FAKULTET ORGANIZACIJE I INFORMATIKE  
VARAŽDIN

Davor Sauer

**RAZVOJ WEB SERVISA PRIMJENOM  
SERVISNO ORIJENTIRANE ARHITEKTURE**

DIPLOMSKI RAD

Varaždin, rujan 2009.

SVEUČILIŠTE U ZAGREBU  
FAKULTET ORGANIZACIJE I INFORMATIKE  
VARAŽDIN

Davor Sauer  
Redoviti student  
Broj indeksa: 33898/03- R  
Smjer: Informacijski sustavi  
VII/1 stupanj

**RAZVOJ WEB SERVISA PRIMJENOM  
SERVISNO ORIJENTIRANE ARHITEKTURE**

DIPLOMSKI RAD

Voditelj rada:  
prof.dr.sc. Strahonja Vjeran, redoviti profesor

Varaždin, rujan 2009.

# Sadržaj

1. Uvod .....	1
2. Servisna orijentacija u programskom inženjerstvu .....	3
3. SOA .....	6
3.1. Principi servisne orijentacije .....	7
3.1.1. Standardizirani servisni ugovori .....	9
3.1.2. Servisi su slabo povezani .....	10
3.1.3. Servisi su apstraktni.....	11
3.1.4. Servisi su višestruko iskoristivi .....	12
3.1.5. Servisi su autonomni .....	13
3.1.6. Servisi su protočni .....	14
3.1.7. Servisi su samo otkrivljivi .....	15
3.1.8. Servisi su sastavljivi .....	16
3.2. Strateški ciljevi korištenja servisno orijentirane arhitekture .....	17
3.2.1. Povećanje interoperabilnosti .....	17
3.2.2. Povećanje povezanosti.....	18
3.2.3. Povećanje opcija ponude dobavljača .....	18
3.2.4. Povećanje regulacije poslovne i tehnološke domene.....	19
3.2.5. Povećanje indeksa ulaganja (ROI) .....	19
3.2.6. Povećanje prilagodljivosti poduzeća .....	20
3.2.7. Reduciranje opterećenja IT .....	20
3.3. Slojevi SOA-e .....	21
3.4. Životni ciklus .....	23
3.5. SOA servisi .....	24
4. Web servisi .....	26
4.1. Arhitektura Web servisa.....	28
4.2. Web servisi u okviru SOA modela .....	29
4.3. Tehnologija Web servisa .....	30
4.3.1. Ugovori Web servisa .....	30
4.3.2. SOAP protokol .....	33
4.3.3. UDDI registar .....	34

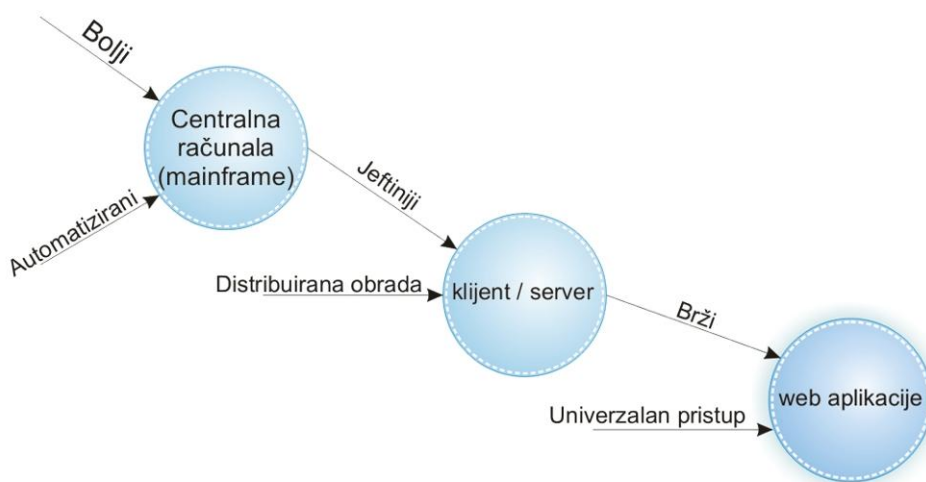
5. Izrada Web servisa primjenom servisno orijentirane arhitekture .....	36
5.1. Servisno orijentirano modeliranje.....	36
5.1.1. Identifikacija servisa .....	37
5.1.1.1. Odozgo prema dolje .....	38
5.1.1.2. Odozdo prema gore .....	39
5.1.1.3. Pristup od sredine .....	40
5.1.2. Specifikacija servisa .....	40
5.1.2.1. Analiza podsustava.....	41
5.1.2.2. Specifikacija komponenata .....	42
5.1.3. Realizacija servisa .....	43
5.2. Implementacija servisa.....	43
5.2.1. Platforme za razvoj Web servisa .....	44
5.2.2. Implementacija.....	45
5.2.3. Testiranje.....	46
5.2.4. Objavljivanje .....	46
5.3. Održavanje web servisa.....	47
6. Projektni primjer .....	48
6.1. Korisnički zahtjev.....	48
6.1.1. Opis korisničkih zahtjeva .....	48
6.1.2. Slučajevi korištenja.....	49
6.1.2.1. Pretraživanje gradova SAD-a .....	49
6.1.2.2. Popis arhiviranih prognoza grada .....	50
6.1.2.3. Prikaz prognoze za grad .....	51
6.1.2.4. Pribavljanje vremenskih prognoza .....	52
6.1.2.5. Registracija korisnika .....	52
6.1.2.6. Autentifikacija korisnika .....	53
6.1.2.6. Izmjena korisničkih podataka.....	53
6.2. Prikupljanje podataka.....	54
6.3. Plan projekta .....	55
6.4. Modeliranje servisa .....	56
6.4.1. Identifikacija servisa .....	56
6.4.1.1. Dijagram aktivnosti arhiviranja prognoza .....	56

6.4.1.2. Dijagram aktivnosti pristupanja arhiviranim podacima .....	57
6.4.1.3. Dijagram aktivnosti korištenja web stranice .....	58
6.4.2. Servisna topologija .....	59
6.4.3. Specifikacija web servisa .....	60
6.4.3.1. Specifikacija web servisa „prognoze“ .....	60
6.4.3.2. Specifikacija web servisa „korisnici“ .....	63
6.5. Implementacija.....	65
6.5.1. Organizacija projekta .....	66
6.5.2. Organizacija paketa .....	67
6.6. Postavljanje .....	68
6.7. Testiranje.....	69
6.7.1. Web stranice .....	69
6.7.2. Testiranje servisa „prognoze“ SOAP klijentom .....	73
6.8. Upravljanje servisima .....	75
6.9. Pregled SOA principa kroz projekt .....	77
7. Zaključak.....	79
8. Literatura .....	81
Prilog.....	83
Instalacijski DVD .....	83

# 1. Uvod

Napredak tehnologije iz perspektive razvoja programske podrške odražava se i na razvoj programskih jezika, njihovih mogućnosti i alata u kojima realiziramo funkcionalnosti koje korisnici očekuju. Zahtjevi korisnika rastu i opsezi usluga koje korisnik očekuje da programska podrška ispunjava/zadovoljava/omogućuje. S porastom opsega mogućnosti programske podrške, programi se razvijaju u sustave, koji postaju sve kompleksniji i povećava se broj korisnika koji koriste takav sustav ili čak više sustava, gdje su korisniku predstavljeni kroz jedno zajedničko sučelje.

Takvi sustavi su danas prerasli u globalne servise, koje objedinjuje jedan zajednički medij i pruža im praktički neograničenu veličinu, povezujući različite dijelove firmi, usluga, s drugim firmama i njihovim uslugama, neovisno o fizičkoj lokaciji, kako bi se korisniku pružila određena i pouzdana usluga. Medij koji povezuje takve sustave je Internet, odnosno način na koji je takva usluga predstavljena korisniku, pomoću weba, tj. Web servisa. Iz perspektive programskog inženjerstva i perspektive upravljanja projektima, razvoj aplikacija baziranih na web tehnologiji je različit od razvoja tradicionalnih stolnih (*engl. mainframe*) aplikacija.



**Slika 1.1. Prikaz evolucije aplikacijskih platformi i utjecaja na evoluciju<sup>1</sup>**

<sup>1</sup> Brandon D.M.. (2008). Software Engineering for Modern Web Applications: Methodologies and Technologies. IGI Global., str XV.

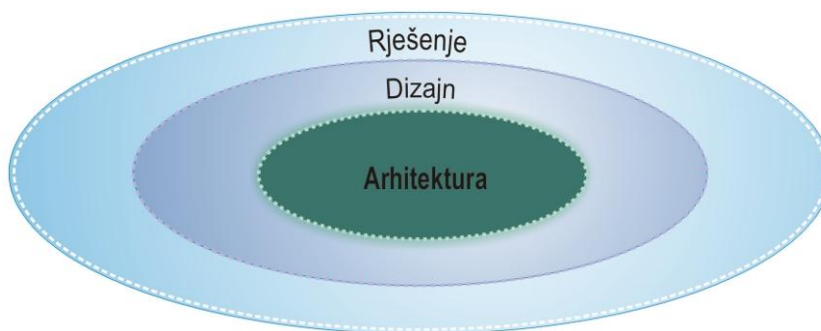


Mnoge firme imaju više područja djelovanja, te stoga pružaju različite usluge, i za svaku uslugu mogu imati posebne poslužitelje, koji rade neovisno jedan o drugome. Firmama je cilj olakšat korisniku korištenje svih tih usluga, te ih želi korisniku prikazati preko jednog sučelja, kao jednu jedinstvenu uslugu. Tako da korisnik zapravo vidi jedan servis, tj. aplikaciju koja mu omogućava cijeli spektar usluga, a ako je jedna od usluga nedostupna, sustav i dalje funkcionira oslanjajući se na preostale funkcionalne servise, pri čemu korisnik i dalje može koristiti trenutne usluge. Odnosno korisniku sustav nije u cijelosti nedostupan nego samo neki njegovi dijelovi i funkcionalnosti.

Postavlja se pitanje kako izgraditi i kako pristupiti izgradnji tako velikog i kompleksnog sustava, koji je spoj više manjih sustava koji mogu raditi neovisno jedan o drugome. Takvi veliki sustavi se moraju pomno planirati, prilikom čega takav plan čini arhitekturu tog sustava.

Odgovor na ova i pitanja koja si postavljamo uz ovakvu problematiku se krije u arhitekturi koja je usmjerena na pružanje usluga, tj. servisno-orijentirana arhitektura ili skraćeno SOA (*engl. Service-Oriented Architecture*).

SOA je jedna od najvažnijih transformacija IT aplikacijske arhitekture u zadnjih nekoliko godina, prezentirajući otvoreni standard i štoviše nekoliko pristupa dizajniranju, izradi i postavljanju aplikacije.



**Slika 1.2. Prikaz slojeva i ovisnosti za postizanje rješenja<sup>2</sup>**

---

<sup>2</sup> Christudas B.A., Barai M., Caselli V. (2008). Service Oriented Architecture with Java. Pack Publishing Ltd., str 5.

## 2. Servisna orijentacija u programskom inženjerstvu

Već sredinom šezdesetih godina nailazi se na prve poteškoće i neuspjehe prilikom izgradnje velikih i opsežnih računalnih informacijskih sustava. Česti uzroci poteškoća bili su loša ili pak nepostojeća tehnika i metodologija razvoja takvih sustava, iz čijih je neuspjeha proizašla važnost razvoja primjerenih tehnika i metoda projektiranja informacijskih sustava. Od tad su razvijena brojna sredstva, tehnike i metode projektiranja, koja imaju jednu zajedničku osobinu, a to je isticanje inženjerske prirode projektiranja. Odnosno vještine i intuicije su nadomještene ili potpomognute dobro definiranim sredstvima za projektiranje i organizacijom procesa projektiranja.

Zbog sve veće složenosti informacijske tehnologije, a time i složenosti sustava, suvremena metodologija projektiranja informacijskih sustava teži da se proces projektiranja podigne na logičku razinu, tj. stavlja se naglasak na logičko definiranje tokova podataka, procesa njihove obrade i strukture korištenih podataka sustava, neovisno o načinu fizičke realizacije.

Proces razvoja informacijskog sustava zahtjeva poznavanje organizacijske strukture, poslovnih ciljeva i tehnološkog sustava čiji se informacijski sustav razvija. Projektant mora imati znanje o sustavu koji razvija, koje stječe temeljem dokumentacije i neposrednom komunikacijom s korisnicima i informatičarima. Pokušava se na što lakši način prevladati jaz između te dvije strane, kako bi se razvili što kvalitetniji i korisniku prilagodljiviji informacijski sustavi.

Informacijski sustavi su uglavnom vrlo kompleksni i složeni sustavi. Rad na realizaciji takvih sustava često zahtjeva rad i suradnju više osoba koje posjeduju različita znanja, i imaju različite interese u pogledu razvoja takvog sustava, te takva situacija čini proces analize i projektiranja još složenijim.

U projektiranju i razvoju informacijskih sustava često se susreće pojam programskog inženjerstva (*engl. software engineering*). Disciplina programskog inženjerstva obuhvaća znanje, alate i metode za definiranje zahtjeva programske podrške, obavljanja zadataka dizajna programske podrške, realiziranje programske podrške, testiranja programske podrške i održavanja programske podrške. Programsko inženjerstvo koristi znanja iz mnogih disciplina, kao što je računalno inženjerstvo, računarstvo, menadžment, matematika, upravljanje projektima, upravljanje kvalitetom i sistemsko inženjerstvo. Dakle programsko inženjerstvo obuhvaća opseg aktivnosti od samog projektiranja do tehničke realizacije i održavanja informacijskog sustava.

Kroz povijest razvoja programskog inženjerstva, ono teži podizanju na logičku, apstraktnu razinu, gdje se odraz te težnje vidi u dozrijevanju programskih jezika, platformi, alata i obrazaca rada i pristupa. Svaki dobro strukturirani i softverski razvijen sustav pun je obrazaca kojeg oblikuje određeni programski jezik, do mehanizama koji definiraju suradnju između objekata, komponenata i drugih dijelova sustava. Na najvišoj razini apstrakcije, svaki sustav ima arhitekturu, koja obuhvaća ključne ideje (apstrakcije) i mehanizme koji definiraju strukturu i ponašanje sustava iz perspektive različitih interesnih skupina, od kojih svaka ima svoje zahtjeve.

Na toj najvišoj logičkoj, odnosno apstraktnoj razini razvoja sustava, cilj je razviti servise koji će funkcionirati kao jedan sustav, pri čemu pojedini servisi međusobno komuniciraju. Ovdje se počinje govoriti o servisno-orijentiranim programskim sustavima (*engl. service-oriented software systems*) ili skraćeno SOSS, koji postaje jedna od ozbiljnijih tema u programskom inženjerstvu. Potrebno je imati u vidu da servisna orijentacija (SO) nije usmjerena samo na Web servise i Internet. Već postoji nekoliko varijacija SOSS-a, ovisno o raznolikosti programskih domena, korisničkih prohtjeva, procesa izrade i različitih programskih metoda razvoja. A glavni uvjeti su korisničko-orijentirana sučelja servisa, korištenje peer-to-peer<sup>3</sup> filozofije, kombinacije različitih tehnologija komunikacije između servisa, te autonomija servisa. Takvi uvjeti servisne orijentacija omogućuju da programi postanu odgovarajuće razvijeni proizvodi.

Veliki informacijski sustavi često se razvijaju kao mreža slabo povezanih i autonomnih komponenti ili servisa, korištenjem peer-to-peer (*p2p*) pristupa. Jedan od pristupa korištenja *p2p* se koristi u elektroničkom poslovanju, gdje jedan servis prvo mora pronaći partnera s kojim će komunicirati, nakon čega partner mora ponuditi sučelje preko koje može komunicirati (npr. sučelje definirano pomoću WSDL). Prilikom komunikacije između dva servisa koristi se protokol za razmjenu poruka, kojeg oba sustava poznaju (npr. SOAP<sup>4</sup> protokol.)

Uz takve principe razvoja informacijskog sustava korištenjem servisa koji pružaju određene usluge, veže se pojam arhitekture usmjerene pružanju usluga (SOA – Service-Oriented Architecture) ili servisno orijentirana arhitektura.

Servisna orijentacija je usmjerena oblikovanju pojedinačnih logičkih jedinica koje se mogu udruživati i višestruko koristiti, kao podrška u realizaciji određenih strateških ciljeva i benefita povezanih sa servisno orijentiranom arhitekturom (SOA) i distribuiranih računalnih

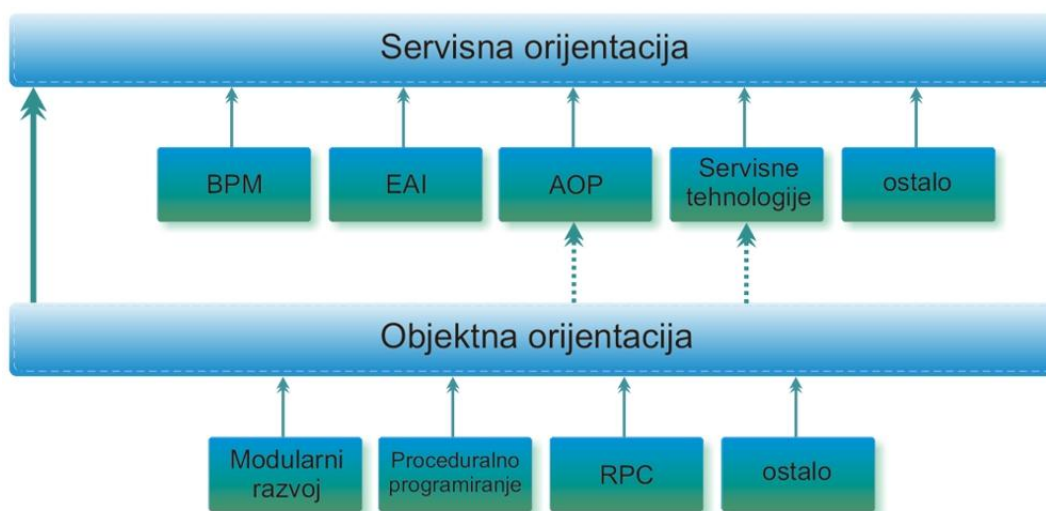
---

<sup>3</sup> Peer-to-peer, koncept umrežavanja računala bez poslužitelja, gdje se komunikacija odvija izravno između računala bez posredničkog računala.

<sup>4</sup> SOAP - Simple Object Access Protokol, komunikacijski protokol koji je baziran na XML-u i neovisan je o platformi, a za prijenos koristi HTTP protokol

platformi, koje sadrže pojedine logičke jedinice. Od kojih svaka logička jedinica može imati drugačiji dizajn, arhitekturu, korištene tehnologije i strukturu.

Servisna orijentacija kao dio paradigme distribuiranog korištenja informatičke tehnologije, može se usporediti s objektno-orijentiranim dizajnom. Zato što servisna orijentacija ima mnoge korijene u objektnoj orijentaciji, i postoje mnogi utjecaji iz drugih industrijskih razvoja.



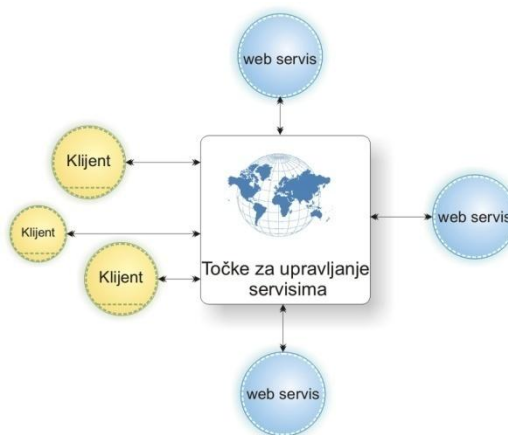
**Slika 2.1. Povezanost servisne orijentacije i objektno orijentacije, te utjecaji ostalih tehnologija<sup>5</sup>**

<sup>5</sup> Erl T. (2009). SOA Design Patterns. Prentice Hall., str. 36.

### 3. SOA

SOA je kratica za servisno-orijentiranu arhitekturu (*engl. Service-Oriented Architecture*), tj. arhitektura usmjerena pružanju usluga. Aplikacije koje proizlaze iz SOA-e su dizajnirane u obliku servisa koje koriste poslovni procesi ili druge integrirane aplikacije. Servisno-orijentirana arhitektura posjeduje koncepte dizajniranja i koncepte arhitekture. Koncepti dizajniranja odnose se na dizajniranje aplikacija/sustava, koje imaju dobro definirane parametre samo-opisnih pristupnih sučelja<sup>6</sup>, sa servisima koji su komponirani unutar poslovnih procesa i izvršavaju funkcije poslovnog procesa. Funkcije i usluge definiraju se standardiziranim jezikom, a preko samo-opisnih pristupnih sučelja se pozivaju, s ciljem potpore određenom segmentu poslovnog procesa. Gdje je svaka interakcija neovisna od ostalih interakcija i komunikacijskih protokola. A smisao arhitekture je omogućit jednostavan mehanizam korištenja pristupnih sučelja prema funkcijama poduzeća, ili aplikacijama koje nude određene usluge putem samo-opisnih sučelja.

SOA koncept omogućuje sustavima i entitetima sustava da komuniciraju međusobno i neovisno s drugim sustavima i entitetima, koristeći što jednostavniji i nezavisniji komunikacijski kanal, koji je standardiziran i primjenjiv unutar svakog od entiteta komunikacije. Smisao je da distribuirani programski entiteti koji postoje samostalno budu u interakciji s drugim uslugama i programima, te se povezivanjem usluga pojedinih entiteta stvara cjeloviti programski sustav.



**Slika 3.1. Odnos entiteta servisa i klijenata**

<sup>6</sup> Samo-opisna pristupna sučelja (self-describing access interface) – „sučelja“ pomoću kojih se definira način komunikacije s poslovnim procesima, odnosno servisima. Npr. kod web servisa je sučelje opisano kroz WSDL dokument.

Suvremeni poslovni sustavi nastoje odgovoriti tržišnim zahtjevima, kao i konkurenciji, te nastoje biti agilni, prilagodljivi i u što kraćem roku iskoristiti tržišne prilike koje se javljaju. Takvi poslovni sustavi moraju imati i sukladan informacijski sustav, koji poduzeću može omogućiti konkurentnu prednost. Zbog toga informacijski sustavi trebaju biti što je više moguće prilagođeni svojstvima i potrebama poslovnih sustava, kako bi bili što učinkovitiji, te kako bi se izbjeglo trošenje nepotrebnih resursa. Kao jedno od mogućih rješenja za razvoj informacijskog sustava takvih svojstava su arhitekture usmjerene na pružanje usluga (SOA). One trebaju omogućiti poslovnom sustavu da brzo i relativno jednostavno razviju programske usluge (servise) koji će obuhvaćati i implementirati potrebne poslovne procese i funkcije. A isto tako trebaju omogućiti da prema potrebama i zahtjevima poslovanja omogućuju brzu i jednostavnu nadogradnju postojećih ili razvoj novih usluga, čija uspješnost ovisi o postignutoj razini interoperabilnosti i mogućnosti višestruke uporabe programskih usluga, što su ujedno i jedni od osnovnih zahtjeva za ostvarenje SOA-e.

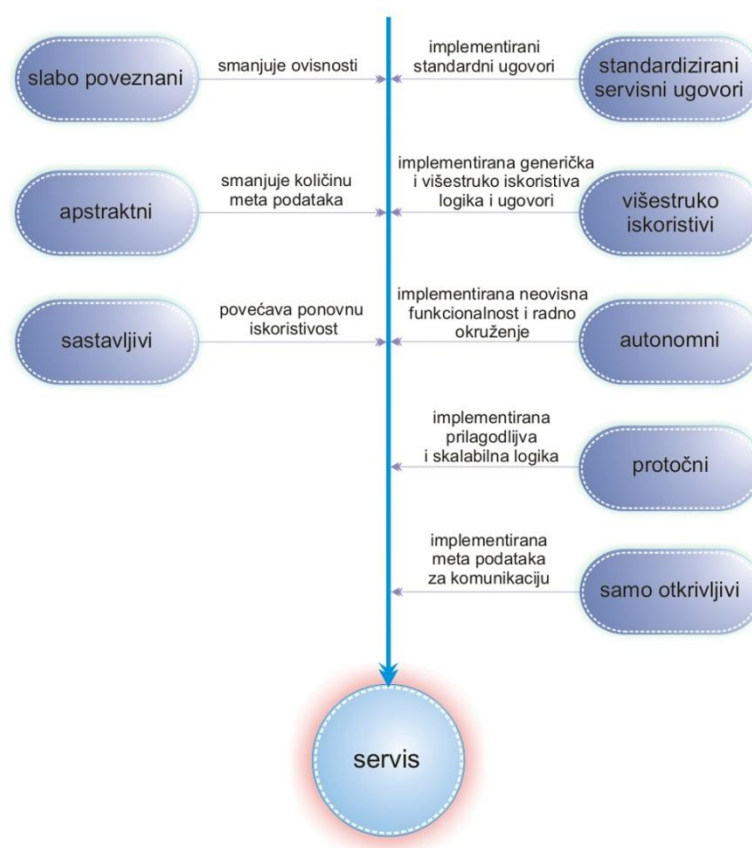
### **3.1. Principi servisne orijentacije**

Sa tehnološkog stajališta, SOA može biti spoj korištenja više tehnologija, alata, korištenih API-ja i mnogih drugih potrebnih dijelova za postizanje konačnog cilja razvoja sustava baziranog na servisno orijentiranoj arhitekturi. Prilikom izrade takve tehnološke arhitekture primjenom servisno orijentirane arhitekture, stvara se okolina u kojoj se mogu realizirati logična rješenja (*engl. solution logic*) koja su dizajnirana u skladu sa servisno orijentiranim principima.

Postoji osam ključnih principa dizajna servisa, gdje svaki od njih osigurava da osnovna načela dizajniranja servisne orijentacije budu konzistentno uključena u svaki od servisa.

Ključni principi servisno-orijentiranog dizajna<sup>7</sup>:

- Standardizirani servisni ugovori
- Servisi su slabo povezani
- Servisi su apstraktni
- Servisi su višestruko iskoristivi
- Servisi su autonomni
- Servisi su protočni
- Servisi su samo otkrivljivi
- Servisi su sastavljivi



**Slika 3.2. Principi servisno-orijentiranog dizajna**

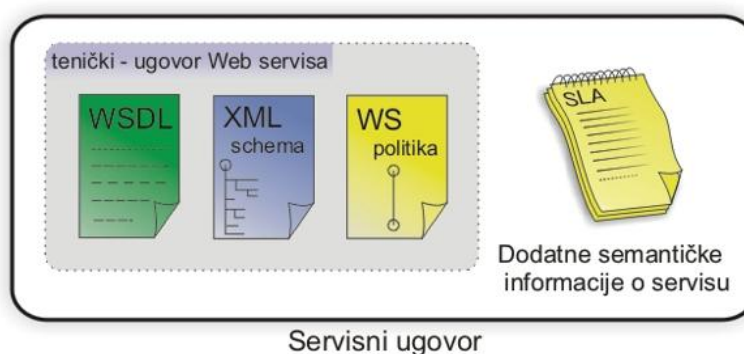
<sup>7</sup> Erl T. (2008). Web Service Contract Design & Versioning for SOA. Prentice Hall., str 39

### 3.1.1. Standardizirani servisni ugovori

Ugovor o servisu ili servisni ugovor je ekvivalent tehničkom sučelju. Kod SOA-e je riječ o proširenoj definiciji koja postavlja uvijete ugovaranja, pružajući tehnička ograničenja i zahtjeve kao i bilo koju semantičku informaciju za koju želimo da bude javno dostupna.

Pomoću servisnih ugovora se opisuju koje usluge ili operacije servis daje na korištenje drugim korisnicima sustava (drugi servisi, aplikacije), klijentima. Servisnim ugovorima se definira kako pojedine operacije u servisu funkcioniraju, kako su definirani tipovi i model podataka, te na koji način se koriste pojedine operacije. Prilikom izrade servisnih ugovora, teži se da budu optimizirani, prikladno podijeljeno i standardizirani kako bi se osiguralo da konačan servis bude konzistentan, pouzdan i upravljiv.

Glavna namjena servisnih ugovora je njihova razmjena između pojedinih klijenata koji koriste taj servis, kako bi se moglo znati kako i na koji način se koriste usluge koje nude. Servisni ugovori se mogu sastojati od grupe opisnih dokumenata, a prilikom korištenja Web servisa, to mogu biti WSDL definicija, XML shema (XSD), WS-Politika korištenja (*engl. WS-Policy*).



**Slika 3.3. Primjer servisnog ugovora Web servisa**



### 3.1.2. Servisi su slabo povezani

Prilikom izgradnje sustava, povezujemo ga s različitim komponentama koje osiguravaju njegovo funkcioniranje. Sustav ili program može biti povezan s bazom podataka, ili s nekim drugim sustavima ili programima koji trenutni sustav opskrbljuju informacijama. Povezanost između komponenti koje čine sustav mogu biti slabije ili jače. Na primjer povezanost s bazom podataka ima jaču vezu, jer se sve potrebne informacije nalaze u bazi, te bez njih sustav ne može funkcionirati. Ako sustav dobiva informacije od nekog drugog sustava, bez kojih informacija može dalje funkcionirati i usluživati korisnike, tada imamo slabiju vezu.

Najbolji način da se opiše razina povezanosti pojedinih elemenata, je prema ovisnosti jednog elementa o drugome. Prilikom izgradnje sustava, povezanost elemenata je nemoguće izbjeći, već je riječ o tome koliko jaku povezanost, ovisnost između elemenata želimo ostvariti.

Kod servisno orijentirane arhitekture servisi su slabo povezani, odnosno njihova pojedinačna funkcionalnost ne ovisi o drugim servisima. Pojedini servisi mogu biti svjesni drugih servisa, te komunicirati s njima, ali prilikom prestanka rada ili nedostupnosti nekog od servisa, servis treba nesmetano nastaviti s radom.

Za komunikaciju između servisa, servisi klijenti koriste servisne ugovore od servisa s kojima žele komunicirati, kako bi znali na koji način pristupati uslugama udaljenog servisa, i u kojem obliku će biti primljene informacije.

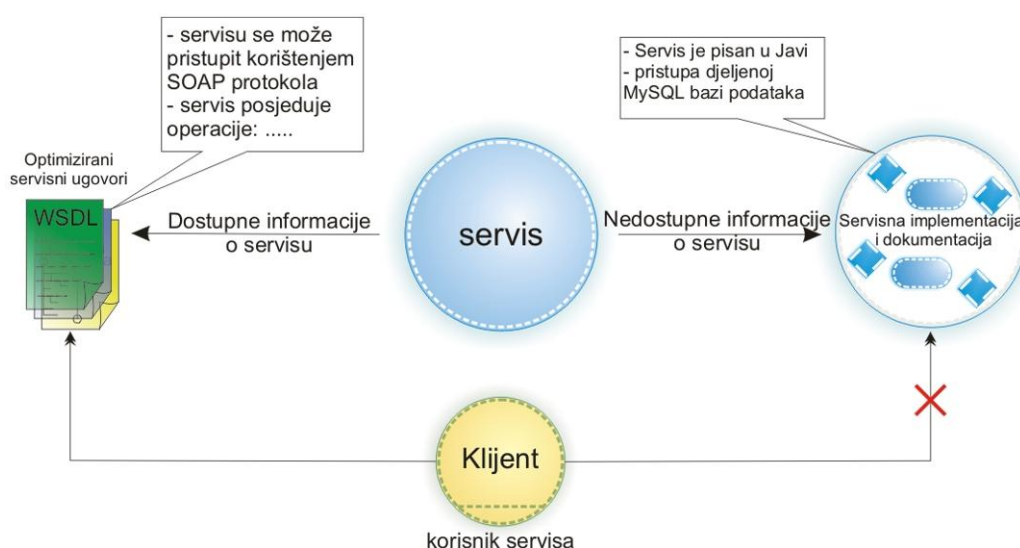
Slabo povezani servisi obično troše više resursa za razliku od onih koji su jače (dobro) povezani. Pogotovo se više resursa troši kad je servis pod velikim opterećenjem. Zbog takvih scenarija potrebno je pravilno izbalansirati povezanost servisa, za što su potrebne veće vještine prilikom dizajniranja servisnog ugovora. Prilikom balansiranja 'povezanosti' servisa, utječe se na njegovu servisnu logiku (aplikacijsku logiku) i servisni ugovor.

### 3.1.3. Servisi su apstraktni

Informacije o servisu koje ne objavljujemo, štite integritet između servisa i korisnika. Skrivanjem određenih detalja o servisu, omogućujemo servisnoj logici i njezinoj implementaciji da se razvija tijekom vremena i ispunjava obveze koje su navedene u objavljenom servisnom ugovoru. Skrivanje informacija o servisu, neće se utjecati na efikasnost korištenja servisa. Zapravo je cilj apstrakcije servisa postići pravi omjer dostupnih informacija o servisu. Pitanje je zapravo u kojoj količini primijeniti apstrakciju i dostupnost informacija o servisu.

Što je više informacija o servisu dostupno, to je i veća povezanost i ovisnost korisnika o servisnom ugovoru. Ali se time povećava i njihovo znanje o pozadinskoj logici, platformi, i detaljima o servisu. A kao rezultat veće količine informacija, korisnici mogu stvarati pretpostavke i predrasude, što nužno nije dobra stvar, jer se na primjer nekim tehničkim karakteristikama korisnici ne trebaju opterećivati.

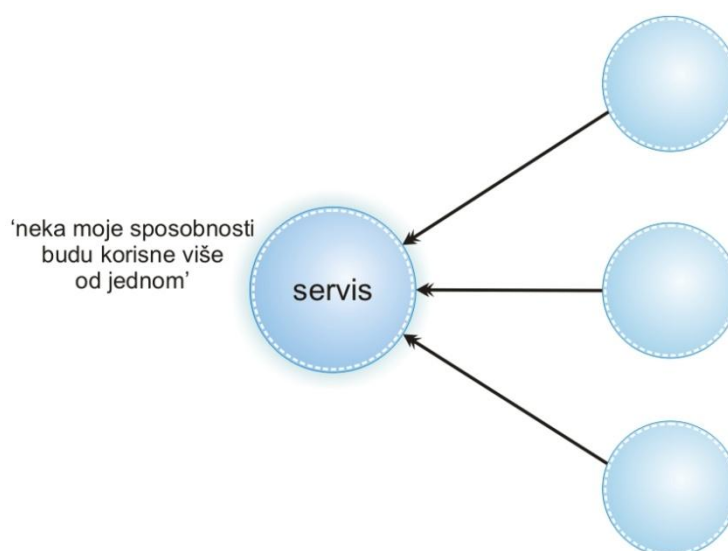
Jedan od glavnih razloga skrivanja detalja o servisu je kako bi smo vlasniku servisa omogućili slobodu razvoja implementacije servisa po potrebi. Na primjer vlasnik može mijenjati načine izvršavanja pojedinih procedura, ali za korisnika servisa, on i dalje dobiva jednak rezultat korištene usluge.



**Slika 3.4. Korisnik može pristupiti sučelju Web servisa i vidjeti osnovne informacije, ali ne može vidjeti i nema pristup implementaciji servisne logike**

### 3.1.4. Servisi su višestruko iskoristivi

Prilikom izrade servisa koji imaju karakteristiku višestruke upotrebljivosti, nastoji se izgraditi servis koji će biti koristan u više svrha, osim samo jedne, tj. omogućit da logika koju ima servis, bude primjenjiva u više situacija ili područja. Jer servisi s tehničkog stajališta imaju niz povezanih funkcija, koji svojim zajedničkim funkcioniranjem predstavljaju servisnu logiku, koja se može primijeniti u raznim situacijama, ovisno o korisnicima servisa, i da li takva logika njima predstavlja rješenje. Prilikom izmjena ili uvođenja novih zahtjeva, nastoji se da te promjene napravimo sa što manje posla i promjena.



**Slika 3.5. Jednu servisnu logiku koristi više korisnika (aplikacija)**

Višestruka iskoristivost servisa je orijentirana na to kako će servisna logika biti oblikovana, pri čemu je ova karakteristika jedna od osnovnih principa za servisnu orijentaciju. Koncept takvog oblikovanja je vrlo jednostavan, ali njegova izvedba nije lak zadatak.

Web servisi su povećali potencijal višestruke iskoristivosti. Jer logiku iz servisa predstavljamo putem Web servisa, te je ona dostupna svima koji podržavaju tehnologiju za razmjenu poruka putem Web servisa (SOAP protokolom).

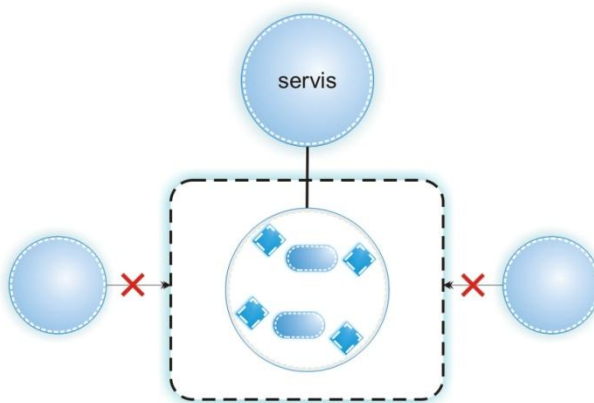
### 3.1.5. Servisi su autonomni

Autonomnost predstavlja sposobnost samo-upravljivosti. Nešto što je autonomno ima slobodu i kontrolu prilikom donošenja osobnih odluka, bez potrebe za odobrenjem ili uključivanjem neke druge strane. Što je sustav neovisniji od nepredvidljivih vanjskih utjecaja, to će biti pouzdaniji. Stoga predvidljivost i pouzdanost predstavljaju dva osnovna faktora autonomnosti.

Ako je riječ o autonomnosti programa, tada je on sposoban iznijeti svoju logiku neovisno o vanjskim utjecajima, a za to vrijeme mora imati mogućnost kontrole nad samim sobom.

Što više kontrole ima program tijekom svog izvođenja u okolini u kojoj se izvodi, to je njegova autonomnost veća.

Autonomnost za servise predstavlja kvalitetu kojom servis može neovisno predstaviti i iznijeti servisnu logiku. Pri čemu se na neovisnost misli da servis ne ovisi o drugim servisima, te da može nesmetano funkcionirati samostalno.



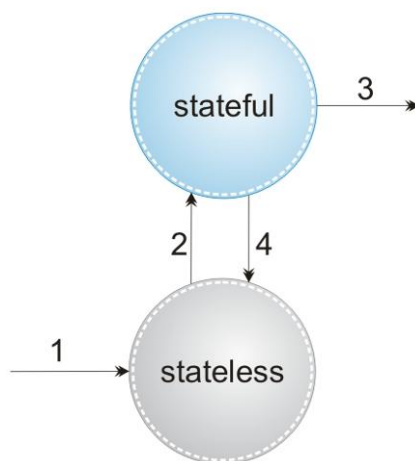
### 3.6. Sustavi su neovisni od svog okruženja

### 3.1.6. Servisi su protočni

Što je servis složeniji, povećava se i količina podataka s kojom servis treba upravljati. Prilikom obrade podataka, servis komunicira s ostalim servisima ili komponentama u kompoziciji s ciljem ispunjavanja vlastite servisne logike. Pri tome mora čekati na ostale servise ili komponente u kompoziciji, što povećava vrijeme obrade podataka. Problem nastaje ako postoji više takvih instanci servisa koje rade istovremeno, te se opterećuju i crpe sistemski resursi servera na kojem se nalazi servis.

Zbog toga upravljamo stanjem (*engl. state management*) koje se ne pamti za prethodne instance, koje može varirati od platforme do platforme. Jer upravljanjem stanjem manipuliramo podacima koje servis obrađuje, te je ključno vrijeme koje je potrebno da informacije dobavimo i na koji način ćemo rasporediti proces obrade podataka i komunikacije s ostalim elementima u kompoziciji kako bi se što manje trošili sistemski resursi.

Stoga nastojimo povećati skalabilnost, tj. mogućnost prilagođenosti prema opterećenju, kako bi bili osigurani svi potrebni radni resursi za nesmetan rad. Servis i njegova okolina trebaju biti dizajnirani tako da podržavaju prijenos ovlasti i odgađanje odgovornosti. Takav pristup u servisnom dizajnu osigurava da servisi budu više protočni, te se takvo stanje naziva *statelessness*.



**Slika 3.7.** Nastoji se održat servis '*stateless*', u bilo kojoj prilici

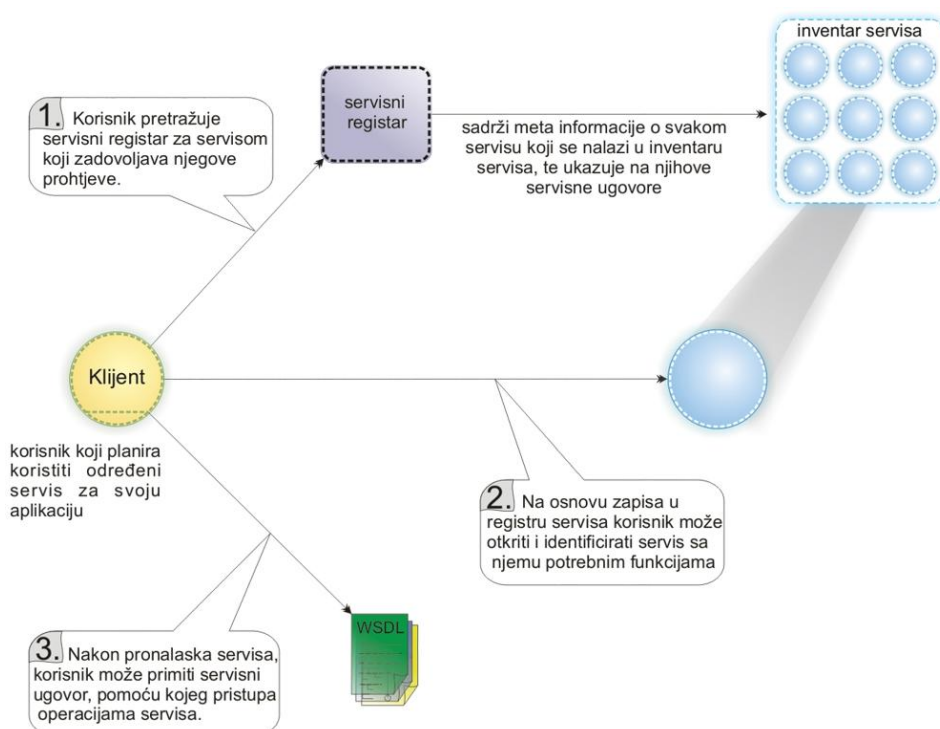
### 3.1.7. Servisi su samo otkrivljivi

Sposobnost servisa da budu samo otkrivljivi osigurava korisniku da po potrebi efikasno pretražuje i locira dostupne servise kako bi mogao koristiti njihovu logiku. Prilikom pronalaska nekog od rješenja, korisnik mora biti u mogućnosti interpretirati karakteristike servisa, kako bi znao da li pronađeno rješenje, tj. logika i mogućnosti servisa odgovaraju njegovim zahtjevima i da li se uklapaju u realizaciju njegovog rješenja.

Prilikom pretraživanja servisa moramo dobro znati što želimo, jer nam se najčešće nameće pitanje "Da li funkcionalnost koju trebamo već postoji, ili ju moramo izgraditi?".

Parametre i karakteristike prema kojima se pretražuju servisi nalazimo u servisnim ugovorima. Ugovori su javno dostupni, ali ih je potrebno objaviti na jednom centralnom mjestu (*engl. service registry*), za kojeg znaju svi korisnici unutar neke radne okoline (npr. poduzeća).

Moguće su razne metode implementiranja samo otkrivljivosti, pri čemu možemo koristiti razne vrste dokumenata koje su dostupne u javnim mapama dostupnima preko mreže ili se mogu objaviti na web stranicama. S pojavom UDDI (*engl. Universal Description, Discovery and Integration*) mehanizma, formalizirana je samo otkrivljivost, koja se pojavila s prvom generacijom Web servisa, koja se koristi kao jedan od ključnih mehanizama prilikom korištenja servisno-orijentirane arhitekture.



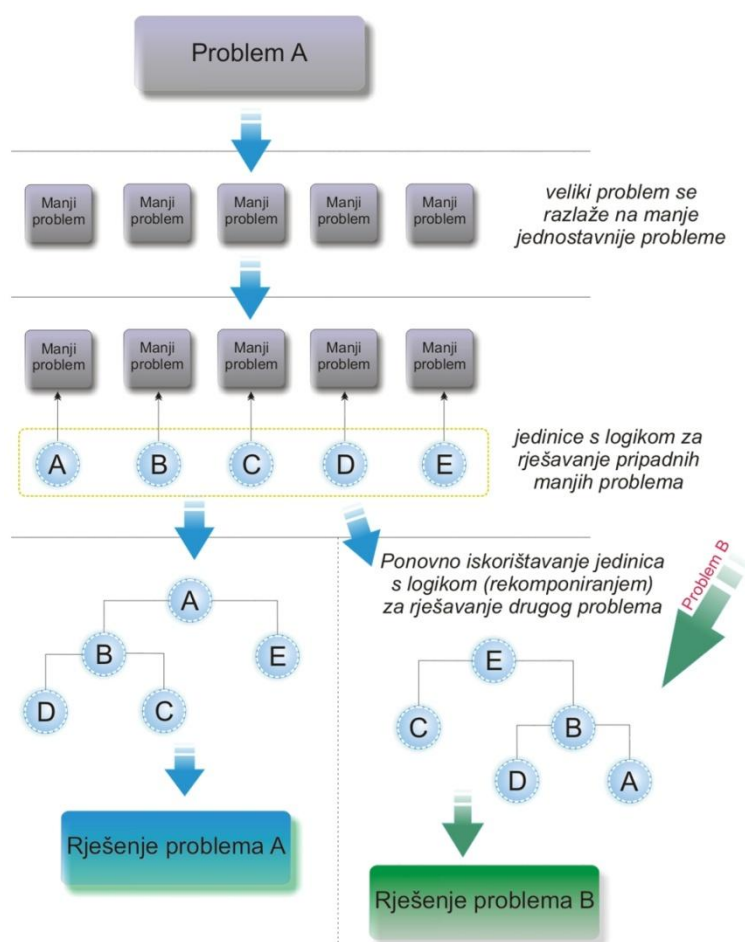
**Slika 3.8. Koraci prilikom otkrivanja servisa**

### 3.1.8. Servisi su sastavljivi

Ponovna iskoristivost servisa je jedna od bitnih karakteristika servisne-orijentacije, ali mogućnost spajanja i sastavljanja servisa od postojećih je jedna od ključnih dizajnerskih karakteristika servisno orijentirane arhitekture.

Mogućnost korištenja postojećih komponenti i njihovih funkcionalnosti često se koristi prilikom programiranja pri uporabi DLL datoteka (*engl. dynamic link library*). Gdje je moguća fizička odvojenost pojedinih funkcionalnih komponenti, pri čemu je glavni problem raščlanjen na više manjih korisnih jedinica, koje se mogu višestruko koristiti u različite namjene.

Isti pristup se koristi i kod sastavljanja servisa, gdje se za rješavanje jednog problema može koristiti određen skup servisa, pri čemu se kombiniraju njihove funkcionalnosti da bi se postiglo rješenje. A ti isti servisi se mogu ponovo koristiti i spojiti u neku drugu kompoziciju kako bi se riješio neki drugi problem (*engl. choreography*). Na taj način servisi su višestruko iskoristivi i sastavljivi.



**Slika 3.9. Višestruka iskoristivost i sastavljivost servisa kod rješavanja problema A i B**

## 3.2. Strateški ciljevi korištenja servisno orijentirane arhitekture

Vizija koja se krije iz servisno-orijentirane arhitekture je vrlo ambiciozna, a ujedno i vrlo atraktivna organizacijama koje imaju ozbiljan interes poboljšati svoj IT sektor. Zajednički ciljevi i beneficije su ojačale i definirale jasnu sliku ove vizije. Što je poduzećima dalo dodatan poticaj da uspješno prihvate servisno-orijentiranu arhitekturu.

Neki od ciljeva i benefita su<sup>8</sup>:

- povećanje interoperabilnosti
- povećanje povezanosti
- povećanje opcija ponude dobavljača
- povećanje regulacije poslovne i tehnološke domene
- povećanje indeksa ulaganja (ROI)
- povećanje prilagodljivosti poduzeća
- reduciranje opterećenja IT

### 3.2.1. Povećanje interoperabilnosti

Interoperabilnost je sposobnost informacijskih i komunikacijskih sustava i poslovnih procesa da podrže protok podataka i omoguće razmjenu informacija i znanja. Kod SOA-e, interoperabilnost se odnosi na dijeljenje podataka. Što je aplikacija interoperabilnija, lakše je s njom raditi i izmjenjivati podatke. Cilj SOA-e je uspostaviti čistu interoperabilnost između servisa, s ciljem da se smanji potreba za integriranjem. Na interoperabilnost se gleda kroz principe i standarde za dizajniranje aplikacija. Što omogućava stvaranje okoline u kojoj servisi mogu nastajati u različitim projektima u različito vrijeme, te ih se može zajedno spajati kako bi se automatizirali razni poslovni procesi.

---

<sup>8</sup> Erl T. (2007). SOA Principles of Service Design. Prentice Hall. str. 104



Čista interoperabilnost predstavlja temeljni cilj servisne-orijentacije, koja osigurava temelje za realizaciju strateških ciljeva i pogodnosti. Standardizacija ugovora, mjerljivost, predviđanje ponašanja i pouzdanost su samo neke od karakteristika dizajniranja, potrebnih za olakšavanje interoperabilnosti.

### 3.2.2. Povećanje povezanosti

Povezana IT okolina je ona okolina u kojoj su povezani resursi i aplikacije uslijed održavanja individualne autonomije i samouprave. Jedan od ciljeva SOA-e je povećanje te perspektive na bilo koji aspekt poduzeća. Što se postiže najviše standardizacijom, te mogućnošću razlaganja servisa na komponente, od kojih svaka predstavlja zatvoreni segment poduzeća.

### 3.2.3. Povećanje opcija ponude dobavljača

Raznolikost isporučitelja (usluga/tehnologija/sredstava) upućuje na mogućnost organizacije da odabere najbolji proizvod i tehnologiju koja se nudi. Nije nužno imati prednost poduzeća u obliku da postoje različiti isporučitelji, već je prednost imati opciju raznolikosti kad je to potrebno. Da bi imali i zadržali tu opciju, potrebno je osigurati da tehnološka arhitektura ne bude vezana i zatvorena na samo jednu platformu koja ju opslužuje. Ovisnost o platformi najviše se smanjuje prilikom implementacije SOA-e pomoću Web servisa, koji mogu biti podržani od više platformi.

To predstavlja određeno stanje organizaciji, gdje se ona može konstantno mijenjati, proširivati i zamjenjivati implementirano rješenje i tehnološke resurse bez ometanja cjelokupne povezanosti servisne arhitekture. Ova mjera upravljačke autonomije je privlačna jer produžuje životni ciklus i povećava dobit automatiziranih rješenja.

Prilikom dizajniranja SOA-e, dobro je omogućiti dosljednu komunikacijsku radnu okolinu između servisa, te na taj način dobivamo organizaciju sa stalnom opcijom da se poduzeće mijenja koliko je to potrebno.

### 3.2.4. Povećanje regulacije poslovne i tehnološke domene

SOA nas uvodi u paradigmu koja promovira apstrakciju na mnogo razina. A jedna od najučinkovitijih je funkcionalna apstrakcija prilikom uspostavljanja servisne razine, koja točno realizira i prati poslovni model. Zbog tog svojstva poslovna logika može postojati implementirana u obliku fizičkog servisa.

To svojstvo je postignuo objedinjavanjem strukturne analize i procesa modeliranja, koji mora uključivati rad eksperta iz poslovnog okruženja, koji mora definirati koncepte servisa. Rezultat projektiranog servisa je mogućnost automatiziranja tehnologije s poslovnim izvještajima na nekoj razini.

Servisi su dizajnirani tako da u potpunosti budu interoperativni direktno s poslovnim promjenama, tako da se servisi mogu rekonfigurirati, kako bi podržavali novu poslovnu logiku.

Takva svojstva omogućuju servisno-orijentiranoj tehnologiji da se direktno veže sa samim poslovima koji se odvijaju.

### 3.2.5. Povećanje indeksa ulaganja (ROI)

Mjerenje indeksa ulaganja automatiziranih rješenja je kritičan faktor za određivanje koliko je isplativa aplikacija ili sustav. Što je veći povrat, organizacija ima veću korist od rješenja. Ali što je indeks ulaganja (*engl. Return On Investment - ROI*) manji, to je cijena automatiziranog rješenja veća, te više šteti profitu poduzeća.

Tradicionalno razvijene aplikacije se proširuju vremenom, te je rezultat potencijalno kompleksna okolina koja zahtjeva izuzetna ulaganja za održavanje. Za mnoge organizacije je problem što mnogo ulažu u svoju informatičku podršku, ali to ulaganje nema odgovarajuću ili željenu povratnu vrijednost.

Servisna orijentacija se zalaže za kreiranje agnostičke logike<sup>9</sup> rješavanja problema. Koja servisu povećava potencijal kojim se može realizirati, tako da se servisima dopusti da se u više navrata sklope u različite kompozicije. Tako se svaki agnostički servis može ponovo upotrijebiti mnogobrojno puta, za automatizaciju različitih poslovnih procesa, kao jednog od dijelova različitog servisno-orijentiranog rješenja.

---

<sup>9</sup> Logike koja je shvatljiva za svaku pojedinu i višestruku uporabu

Činjenica je da servisna-orijentacija cilja na osiguravanje višestrukog korištenja pojedinog servisa u različitim poslovnim procesima, što predstavlja glavnu karakteristiku servisne-orijentacije. A kao sporedna karakteristika je realizirati što više servisa koji posjeduju agnostičko svojstvo.

### 3.2.6. Povećanje prilagodljivosti poduzeća

Agilnost prilagodljivosti na razini organizacije odnosi se na efikasnost kako organizacija reagira na promjene. Povećanje agilnosti organizacije je vrlo privlačna, pogotovo onima koje su u privatnom sektoru.

Dijelovi servisne orijentacije se protežu kroz sve dijelove poduzeća, preko servisa koji su standardizirani i mogu se višestruko koristiti u poslovnim procesima i aplikacijskim okruženjima, ovisno gdje se stvori potreba za tim pojedinim servisima.

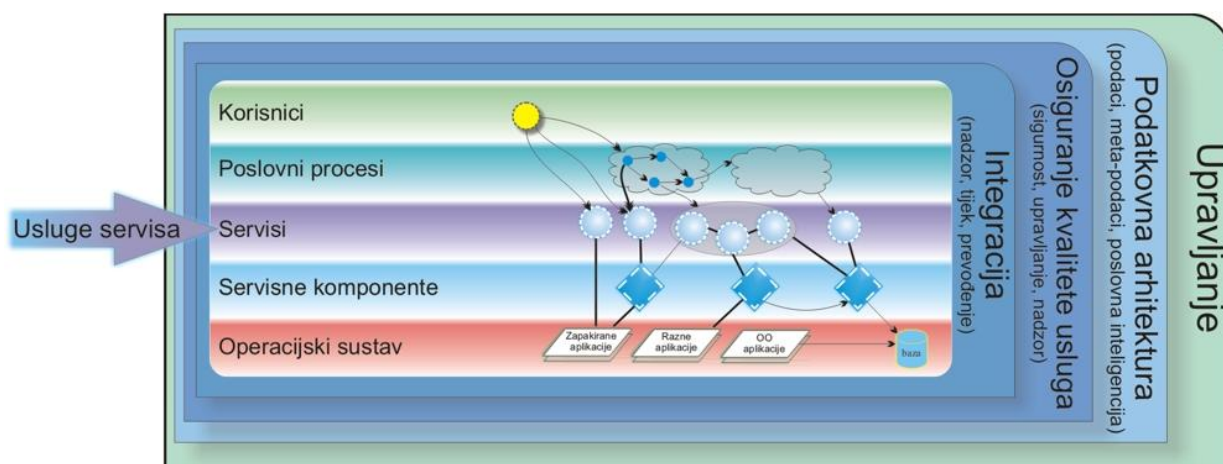
Ti servisi su zapravo višestruko upotrebljiva IT sredstva, te se mogu komponirati u različite konfiguracije. Koje kao rezultat imaju smanjenje vremena i napora potrebnih za automatizaciju nove ili promjenu postojećih poslovnih procesa, te se razvoj projekta može realizirati sa značajno manjim razvojnim naporom.

Rezultat je dostavljen projekt s pojačanom odgovornošću i reduciranim vremenom za postizanje tržišnog potencijala, što se sve zajedno manifestira u povećanje prilagodljivosti organizacije.

### 3.2.7. Reduciranje opterećenja IT

Dosljedno primjenjivanje servisne orijentacije rezultira u IT-u poduzeća smanjenje gubitaka redundancije (preopterećenja), smanjenje veličine i cijene operacija i smanjenje općih troškova povezanih s upravom i evolucijom poduzeća.

### 3.3. Slojevi SOA-e



Slika 3.10. Slojevi servisno orijentirane arhitekture<sup>10</sup>

Slojevi SOA nude pregled konceptualnog rješenja, na većem nivou apstrakcije. Točnije na koje sve slojeve poduzeća SOA ima utjecaj i kroz koje se sve slojeve manifestira realizacija.

Funkcionalni slojevi su<sup>11</sup>:

- Operacijski sustav - predstavlja postojeća IT rješenja, sredstva i tehnologije, preko čega se vidi vrijednost IT-a. To mogu biti J2EE ili .Net aplikacije, transakcijski sustavi, baze podataka, sustavi kao ERP i CRM.
- Servisne komponente – ovaj sloj sadrži programske komponente koje realiziraju servise, pri čemu se mogu realizirati i korištenjem aplikacija u sloju 'Operacijskog sustava'. Servisnim komponentama imaju pristup samo servisi, posredstvom kojih pristupaju i svi ostali. Postojeće komponente se mogu ponovno iskoristiti, što je prikladno za realizaciju SOA-e, jer je omogućena fleksibilnost korištenja postojećih komponenata.
- Servisi - predstavlja dostupne servise, koji mogu predstavljati kolekciju jedne ili više poslovnih funkcija. I načini na koje se može pristupiti i koristiti pojedine funkcije (operacije), što se kod Web servisa opisuje WSDL dokumentom.
- Poslovni procesi - predstavljaju operacijske jedinice koje implementiraju poslovne procese kao koreografije servisa

<sup>10</sup> <http://www.ibm.com/developerworks/webservices/library/ws-soa-term1/Fig2.gif>, očitano: 12.07.2009.

<sup>11</sup> Arsanjani A, Zhang L., Ellis M., Allam A., Channabasavaiah K., Design an SOA solution using a reference architecture, Dostupno na: <[http://www.ibm.com/developerworks/webservices/library/ar-archtemp/index.html?S\\_TACT=105AGX01&S\\_CMP=LP](http://www.ibm.com/developerworks/webservices/library/ar-archtemp/index.html?S_TACT=105AGX01&S_CMP=LP)>, očitano: 12.07.2009.

- Korisnici - sloj pomoću kojeg se pristupa poslovnom sloju, servisnom sloju i aplikacijama u sloju operacijskog sustava. Ovaj sloj predstavlja korisnike sustava.

Nefunkcionalni slojevi su<sup>12</sup>:

- Integracija - ovaj sloj omogućuje integraciju usluga uvođenjem funkcionalnosti kao što su inteligentno usmjeravanje, upravljanje protokolima, te ostali transformacijski mehanizmi koji su objedinjeni u povezan sustav ESB (*engl. Enterprise Service Bus*). Sa stajališta Web servisa, WSDL definira način povezivanja, s kojim se implicira lokacija na mjesto gdje se pruža usluga, a ESB pruža lokacijski neovisan mehanizam za integraciju.
- Osiguranje kvalitete usluga - ovaj sloj osigurava mogućnosti koje su neophodne za održavanje kvalitete usluga, sigurnosti, upravljanja i nadzora (*engl. QoS – quality of service*).
- Podatkovna arhitektura - osigurava podršku za rad s podacima, meta-podacima i poslovnom inteligencijom (*engl. business intelligence*)
- Upravljanje - osigurava podršku za upravljanje životnog ciklusa poslovnih operacija

---

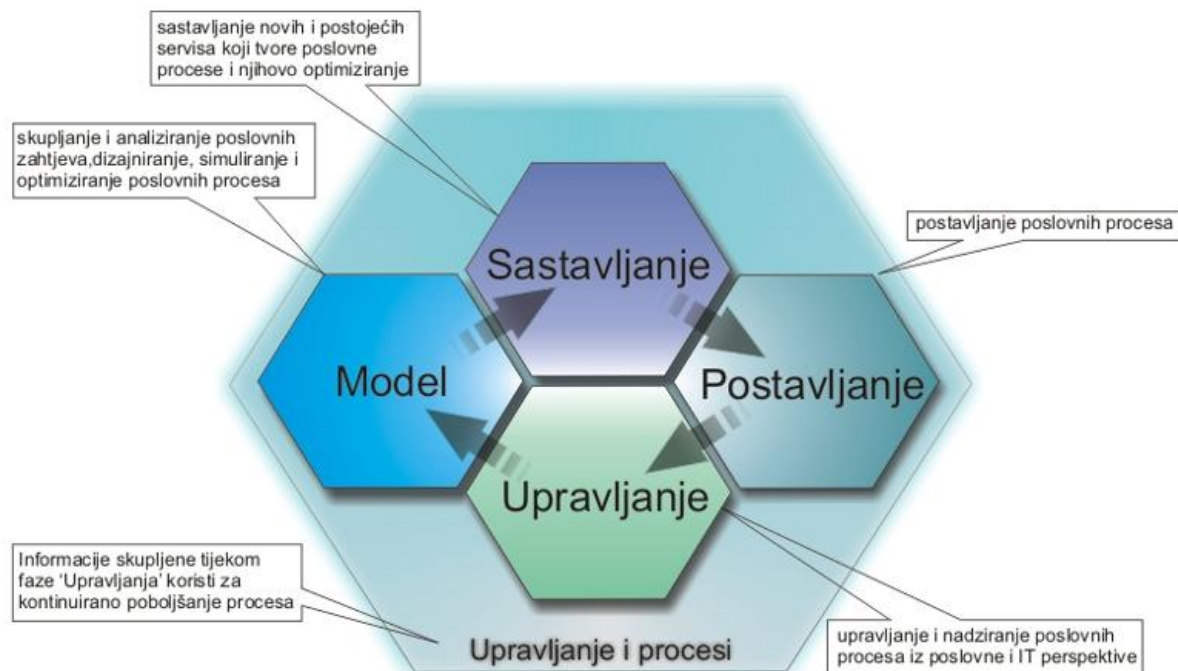
<sup>12</sup> Arsanjani A, Zhang L., Ellis M., Allam A., Channabasavaiah K., Design an SOA solution using a reference architecture , Dostupno na: <[http://www.ibm.com/developerworks/webservices/library/ar-archtemp/index.html?S\\_TACT=105AGX01&S\\_CMP=LP](http://www.ibm.com/developerworks/webservices/library/ar-archtemp/index.html?S_TACT=105AGX01&S_CMP=LP)>, očitano: 12.07.2009.

### 3.4. Životni ciklus

Tijekom razvoja svaki proizvod ima svoj životni ciklus, odnosno faze kroz koje prolazi taj proizvod, od ideje, planiranja, izrade, testiranja, održavanja, upravljanja i iskorištavanja dijelova servisa u neke druge svrhe i na kraju odumiranja.

Životni ciklus SOA-e i servisa rađenih prema toj arhitekturi, imaju sljedeće faze životnog ciklusa:

- Model - u kojem se radi poslovna analiza i dizajn (zahtjevi, procesi, ciljevi, indikatori ključnih performansi (engl. KPI)), te analiza i dizajn informacijske tehnologije kroz identifikaciju servisa i njihovih specifikacija
- Sastavljanje - koje podrazumijeva implementaciju servisa i izgradnju servisnih komponenata
- Postavljanje - kod kojeg se postavljaju servisne komponente i radna okolina
- Upravljanje – upravljanje i nadziranje poslovnih procesa iz poslovne i IT perspektive
- Upravljanje i procesi – upravlja se radnom okolinom, nadziru performanse servisa i dodatno po potrebi definira politika servisa tijekom rada. SOA upravljanje govori o planiranju, definiranju, omogućavanju i mjerenju tijekom životnog ciklusa servisa. Ova razina definira koja su stanja servisa, koje akcije su potrebne za promjenu stanja, kako, koje procese i metode koriste servisi i tko im ima pristup. Stanja servisa kod razine upravljanja mogu biti utvrđivanje karakteristike servisa, odobrena sredstva za realizaciju, specifikacija servisa, implementacija, odobravanje, operativno stanje, objavljivanje (publiciranje), gubljenje na vrijednosti i povlačenje servisa iz uporabe.



**Slika 3.11. Životni ciklus servisno orijentirane arhitekture<sup>13</sup>**

### 3.5. SOA servisi

SOA je neovisna o tehničkoj platformi, zbog toga što predstavlja arhitekturni model. S takvom karakteristikom, postoji sloboda korištenja novih i naprednih tehnologija, kako bi se na što efikasniji način postigli strateški ciljevi SOA-e. Za razvoj servisno-orijentiranih sustava (servisa) može se koristiti bilo koja tehnologija koja omogućuje implementaciju distribuiranih sustava.

Trenutno najbolji pristupi za razvoj servisa je u obliku<sup>14</sup>:

- Servis kao komponenta
- REST servisa
- Web servisa

<sup>13</sup> Portier B., SOA terminology overview, Part 1: Service, architecture, governance, and business terms, Dostupno na: <<http://www.ibm.com/developerworks/webservices/library/ws-soa-term1>> , očitano: 18.07.2009.

<sup>14</sup> Erl T. (2009). *SOA Design Patterns*. Prentice Hall, str. 44

## **Servisi kao komponente**

Kod implementacije servisa kao komponente, tu komponentu predstavlja aplikacija koja je dizajnirana tako da bude dio distribuiranog sustava. Koja posjeduje tehničko sučelje, preko kojeg se može javno pristupiti njezinim metodama, dozvoljavajući da budu eksplicitno korištene od drugih aplikacija.

Takve komponente se obično oslanjaju na određenu razvojnu platformu i radnu okolinu za tu tehnologiju (*engl. runtime technology*). Razvojne platforme na primjer mogu biti Java ili .NET tehnologije i njihove radne okoline, koje imaju odgovarajuće tehnološke zahtjeve.

## **REST servisi**

REST (Representational State Transfer) servisi ili RESTful servisi su distribuirani sustavi koji se predstavljaju kao resursi. S tehničkog pogleda, to su jednostavni programi, koji su dizajnirani s naglaskom na jednostavnost, skalabilnost i iskoristivost. Svako od funkcija servisa može se pristupiti putem jedinstvene adrese, URL adrese.

## **Web servisi**

Servisi koji su implementirani kao Web servisi, posjeduju logička rješenja koja su navedena u servisnim ugovorima koja se definiraju u WSDL definiciji i jednoj ili više XML shema, pri čemu se može koristiti i dodatna osiguranja (WS-Policy). Pomoću web servisnih ugovora, se javno omogućuje pristup operacijama Web servisa, osiguravajući tehničko sučelje, ali bez vezanja na neki osnovni komunikacijski protokol. Detaljniji opis i funkcionalnost Web servisa nalazi se u poglavlju 4 (Web servisi).

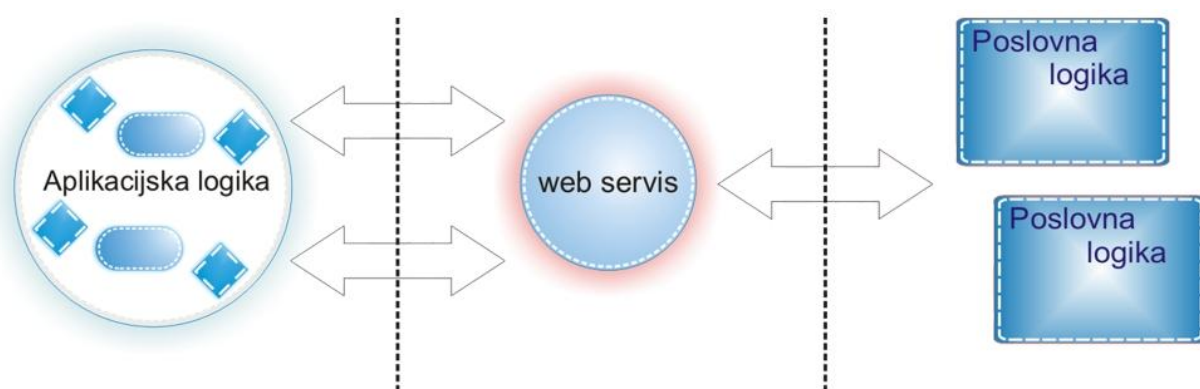


## 4. Web servisi

SOA arhitekturu moguće je realizirati koristeći bilo koju tehnološku platformu za realizaciju distribuiranih servisa. Na taj način poduzećima dajemo slobodu da ostvaruju strateške ciljeve servisno-orijentirane arhitekture koristeći se novim naprednim tehnologijama. Na tehnološkom tržištu za realiziranje SOA koncepta je trenutno najutjecajnija implementacija distribuiranih servisa korištenjem Web servisa.

Web servisi predstavljaju aplikacije, koje su dostupne preko računalne mreže. Jer se u današnje vrijeme javljaju sve veći zahtjevi za povezivanjem raznih elemenata (aplikacija na raznim uređajima, koje možemo na neki način umrežiti, s drugim aplikacijama), pri čemu to mogu biti i različiti dijelovi unutar poslovanja poduzeća. Moguće je i povezivanje s drugim poduzećima i njihovim poslovnim procesima ili uslugama, pa do povezivanja pojedinih korisničkih aplikacija i dijelova aplikacija kako bi se omogućila određena funkcionalnost i pristupačnost što većem broju informacija, koje su korisniku on interesa.

Web servisi koji su dizajnirani prema principima servisno orijentirane arhitekture imaju dobru tehničku podlogu za povezivanje poslovnih procesa unutar ili izvan poduzeća. Tako da se definira skup usluga dostupnih preko sučelja Web servisa, koje su definirane pomoću WSDL-a dokumenata.



**Slika 4.1. Komunikacija poslovne logike (klijent), web servisa i njegove servisne logike<sup>15</sup>**

<sup>15</sup> Christudas B.A., Barai M., Caselli V. (2008). Service Oriented Architecture with Java. Pack Publishing Ltd., str. 21

Distribuiranost sustava je ključ u napretku modernih aplikacija, a Web servisi su jedan od vodećih pristupa, zbog svoje jednostavnosti i neovisnosti o platformama za komunikaciju s drugim Web servisima, jer se komunikacija odvija pomoću poruka, a ne operacija (*npr. RPC – remote procedure call*). Stoga Web servisi mogu biti razvijeni na različitim platformama, te će moći i dalje u mogućnosti međusobno komunicirati. Jer servisi za komunikaciju koriste standardizirani jezik, XML, koji se prenosi SOAP protokolom, koji funkcionira na aplikacijskoj razini, pri čemu se za prijenos SOAP poruka koristi HTTP protokol.

S korištenjem HTTP protokola, omogućena je nesmetana komunikacija Web servisa i klijenata kroz vatrozide (*engl. firewall*) ili proxy servise, jer velika većina poduzeća dozvoljava promet HTTP protokolom.

## Generacije Web servisa

Razvoj Web servisa podijeljen je u dvije generacije, od kojih svaka ima svoje standarde i specifikacije.

Prva generacija Web servisa se sastojala od tehnologija kao što su WSDL (Web Services Description Language), XSD (XML Schema Definition Language), SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery and Integration) i WS-I profile.

Druga generacija Web servisa (**WS-\***) donosi poboljšanja korištenih tehnologija iz prve generacije, ali donosi i mnoge nove specifikacije, koje sve zajedno objedinjujemo pod kraticom **WS-\***.

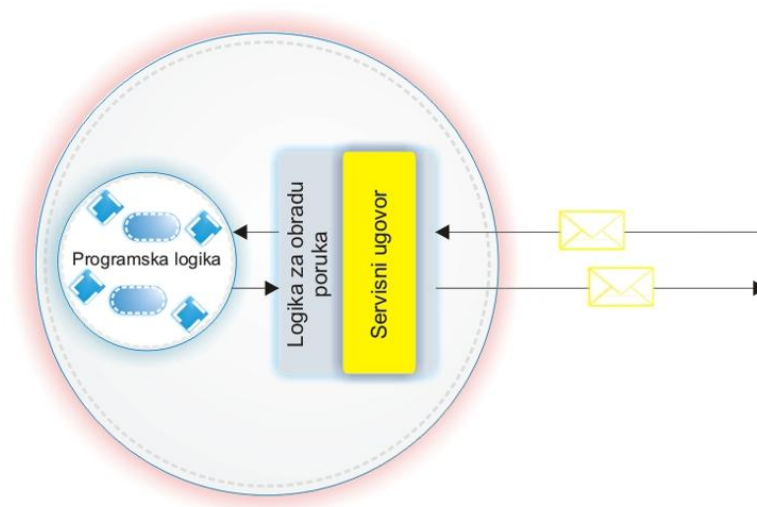
Neke od specifikacija su:

- WS-Security koja definira kako koristiti XML enkripciju i primjenu sigurnosti prilikom razmjene SOAP poruka, kao alternativa ili dodatak HTTPS protokolu.
- WS-Reliability za pouzdanu razmjenu poruka između Web servisa
- WS-Transaction način na koji se obrađuju transakcije
- WS-Addressing standardizira način za umetanje adresa u SOAP zaglavlje.

## 4.1. Arhitektura Web servisa

Web servisi najčešće obuhvaćaju:

- fizički odvojen servisni ugovor temeljen na WSDL dokumentu, definiciji XML sheme (XLS) i moguće definicije politike Web servisa (WS-Policy). Pomoću navedenih dokumenata se javno omogućuje pristup funkcijama servisa. Navedeni dokumenti čine sučelje Web servisa, koje se tradicionalno može usporediti s sučeljem prema API funkcijama.
- logika za obradu poruka, koja raščlanjuje („parsira“) i obrađuje poruke.
- Programska logika, koja može biti posebno programirana u Web servisu, koja je kreirana kao komponenta. Ili može biti logika nekog sustava (programa) kojem smo omogućili pristup putem Web servisa, pri čemu je Web servis posrednik između korisnika i sustava koji se nalazi iza Web servisa.



Slika 4.2. Osnovna arhitektura Web servisa<sup>16</sup>

<sup>16</sup> Erl T. (2008). Web Service Contract Design & Versioning for SOA. Prentice Hall., str. 34

## 4.2. Web servisi u okviru SOA modela

Osnovni model SOA-e se sastoji od:

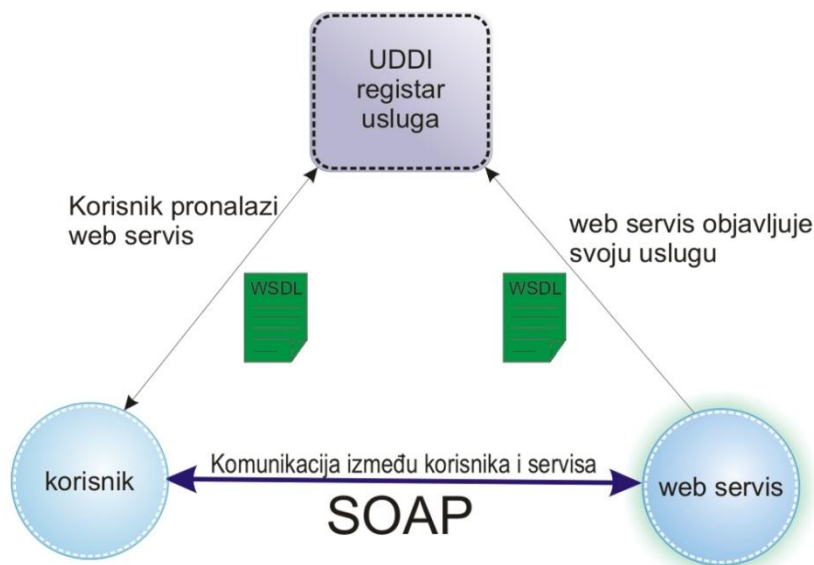
- element koji pruža usluge, Web servis (*engl service provider*)
- korisnik koji traži usluge (*engl. service consumer*)
- registar usluga (*engl. service registry*)

Web servisi pružaju usluge koje su implementirane u servisnoj logici i koje su javno objavljene u servisnim ugovorima koji se kod Web servisa definiraju pomoću WSDL dokumenata. U njima se nalazi opis dostupnih usluga, kako ih koristiti.

Kako bi se znalo koji Web servisi, s kojim karakteristikama postoje i na kojim lokacijama se nalaze, odnosno kako bi smo im mogli pristupiti, informacije o njima objavljujemo u registru usluga. U registru usluga se nalaze WSDL dokumenti dostupnih servisa, čijim opisima pristupamo korištenjem UDDI standarda koji je baziran na XML-u. Koji služi za objavljivanje dostupnih usluga Web servisa, koje su vlasnici, autori servisa objavili da postoji.

Ako korisnik ne zna koji Web servisi su mu dostupni ili traži servise sa specifičnim karakteristikama, tada ih može pretraživati u registru usluga, koji mu nudi adrese dostupnih servisa. Pomoću kojih korisnik može pristupiti Web servisu, te koristiti njegove usluge.

Ovaj koncept se uklapa u jedan od principa SOA dizajna, prilikom čega su servisi samo otkrivljivi.



**Slika 4.3. Način funkcioniranja Web servisa<sup>17</sup>**

## 4.3. Tehnologija Web servisa

### 4.3.1. Ugovori Web servisa

Kroz cijeli ovaj rad se spominju servisni ugovori, koja je njihova namjena i osnovni pojmovi koji se uz njih vežu. U ovom poglavlju biti će riječ o konkretnoj tehnologiji koja se koristi za realizaciju servisnih ugovora, tj. ugovora koji se koriste prilikom implementacije Web servisa.

Web servisni ugovori su kolekcija meta-podataka, koji opisuju različite aspekte koji se nalaze iza sučelja Web servisa, a neki od osnovnih su:

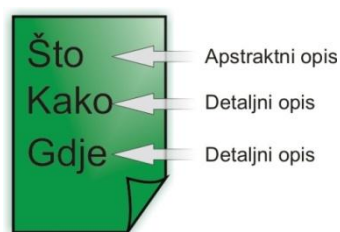
- Namjena i funkcije sadržanih operacija Web servisa
- Poruke koje se razmjenjuju kako bi se pokrenule određene funkcije
- Podatkovni model koji definira strukturu poruka koje se razmjenjuju i pravila za provjeru podataka kako bi se očuvao integritet podataka
- Skup uvjeta koje treba zadovoljiti kako bi se operacijama moglo pristupiti
- Informacije o tome gdje i kako se može pristupiti Web servisu

<sup>17</sup> [http://www.ibm.com/developerworks/websphere/library/techarticles/0409\\_amanuddin/images/image2.gif](http://www.ibm.com/developerworks/websphere/library/techarticles/0409_amanuddin/images/image2.gif), očitano: 15.06.2009.

Osnovna struktura Web servisa sastoji se od odgovora na pitanja:

- **Što** web servis može i pod kojim uvjetima može iznijeti svoje mogućnosti (operacije Web servisa). Apstraktni opis
- **Kako** se servisu može pristupiti. Detaljni opis
- **Gdje** se servisu može pristupiti. Detaljni opis

Razdvajajući proces izrade na tri cjeline pomoću ključnih pitanja, postignuta je fleksibilnost, gdje svaki dio ugovora može imati vlastiti životni ciklus. Pri čemu se odgovor na pitanje '**što**', mogu posvetiti analitičari i arhitekti. A pitanja '**kako**' i '**gdje**' nisu toliko bitna dok se servis ne razvije u cijelosti i dio ugovora Web servisa sa apstraktnim opisom.



**Slika 4.4. Smjernice za strukturiranje Web servisa**

## **WSDL (Web Services Description Language)**

WSDL je jedan od osnovnih tehnologija koja se koristi prilikom kreiranja Web servisa, zato što je postao jedan od osnovnih jezika za sastavljanje ugovora Web servisa. Unutar kojeg se nalazi cijela potrebna struktura ugovora Web servisa sa odgovorima na pitanja 'što', 'kako' i 'gdje'.

Jezik kojim se definira WSDL dokument je XML, te na taj način dokument ne treba dodatno kompilirati, već se dokument jednostavno preuzme. WSDL dokument je razumljiv računalu koje ga je preuzelo, te parsira elemente dokumenta. S druge strane WSDL dokument je čitljiv i razumljiv ljudima, zato što je pisan u XML-u, sa ljudima prepoznatljivim i standardiziranim tagovima. Pomoću WSDL-a se definira kako treba komunicirati sa udaljenim Web servisom, te koje funkcije postoje, o kojim tipovima podataka se radi, i sve potrebne informacije koje su potrebne kako bi se kreirala pravovaljana SOAP poruka za komunikaciju sa Web servisom.

Najlakši način za kreiranje WSDL dokumenata je pomoću gotovih alata koji imaju automatizirano i pojednostavljeno manipuliranje potrebnim parametrima servisnog sučelja, koje

je najčešće vezano uz projekt Web servisa. Prilikom objavljivanja projekta (*engl. deploy*), moguće je automatski generirati WSDL dokument, s dodatnim parametrima koje je korisnik mogao dodati ručno, preko odgovarajućeg sučelja koje nudi korišteni alat.

Postoje dvije verzije WSDL-a, jedna je 1.1, koja nije prihvaćena od strane W3C organizacije, a verzija 1.2, koja je kasnije zbog većih promjena definirana kao verzija 2.0, te je ujedno prihvaćena kao standard od W3C organizacije.

## **XSD (XML Shema Definition Language)**

Pomoću XSD se definira struktura i osigurava valjanost XML dokumenta. XSD služi kao dio servisnog ugovora pomoću kojeg se detaljno definira struktura poruka, u kojoj mogu biti sadržani opisi tipovi, atributi i redoslijed podatak koji se mogu nalaziti u pojedinoj poruci.

XSD može biti dio WSDL dokumenta, ili može biti odvojeno od njega, s pripadnom referencom u WSDL dokumentu.

## **WS-Politika (WS-Policy)**

Pomoću politike Web servisa (WS-Politika) proširuju se definirane karakteristike WSDL dokumenta. Proširuju se dodatne ovisnosti, zahtjevi i karakteristike koje treba ispunjavati koje su nužne za pravovaljano funkcioniranje Web servisa.

Definirati se mogu uvjeti ili pravila pod kojima se usluge Web servisa mogu koristiti, koji tipovi podataka se trebaju dostaviti Web servisu prilikom slanja podataka za pojedinu operaciju ili u kojem obliku, dali komunikacija treba biti kriptirana, ili samo neki dijelovi komunikacije.

Dokument za WS-Politiku može biti pisan tako da bude čitljiv i ljudima, te ga onda računalo parsira i prevodi kako bi proveo opisanu politiku, ili može biti pisan u „računalnom“ kodu koji je čitljiv samo računalu. Datoteka u kojoj je opisana WS-Politika može biti dio WSDL dokumenta, ali isto tako kao i XSD shema, može biti i izvan WSDL-a, naravno sa referencom na datoteku u WSDL dokumentu.

WS-Politika može djelovati na sve razine dizajna servisa, kao i na sve dijelove WSDL dokumenta.

#### 4.3.2. SOAP protokol

SOAP (Simple Object Access Protokol) protokol predstavlja standardizirani format poruka i mehanizam za razmjenu poruka između programa, baziran na XML-u. Pomoću kojih se pokreću operacije na udaljenom Web servisu, koji odgovor šalje istim protokolom, klijentu. Jedini zahtjev koji treba biti ispunjen je mogućnost korištenja HTTP protokola, preko kojeg se prenose SOAP poruke, zato što SOAP protokol radi na aplikacijskoj razini.

Na taj način je omogućena interoperabilnost između različitih sustava i aplikacija koje su pisane u različitim jezicima. Pri čemu svaki od elemenata podržava HTTP protokol, pomoću kojeg se razmjenjuju XML poruke koje čine SOAP protokol. Omogućeno je i filtriranje na temelju HTTP i SOAP zaglavlja. Kod novijih verzija SOAP protokola jasno su definirana pravila za korištenje SMTP i FTP protokola, te je tako povećan opseg mogućih protokola za korištenje.

Pomoću SOAP protokola moguće je slati i poruke preko određenog lanca posrednika, tako da informacije koje govore o kojim posrednicima je riječ spremamo u zaglavlje SOAP poruke, a informacije koje su namijenjene krajnjem korisniku nalaze u tijelu poruke, te se one ne mijenjaju.

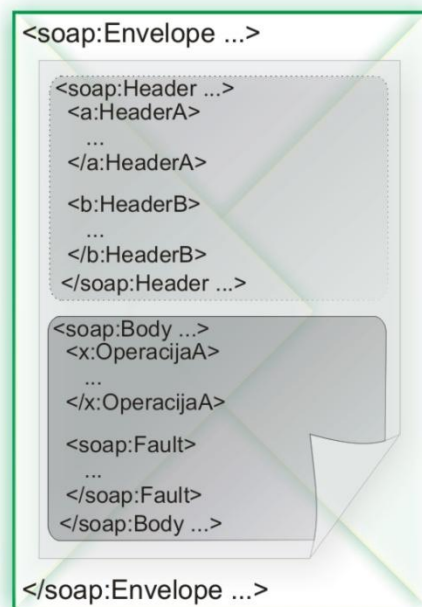
SOPA protokol sastoji se od tri cjeline<sup>18</sup>:

- Omotnica – *envelope*, predstavlja osnovni dio SOAP protokola, preko koje je predstavljena poruka. U omotnici se nalazi verzija SOAP protokola, link na strukturu poruke, i pravila za kodiranje poruke
- Zaglavlje – *header*, je proizvoljan element, koji može sadržavati više unosa. U zaglavlju se mogu nalaziti informacije o primjeni poruke, identifikatoru prijenosa, podaci o pošiljatelju i primatelju poruke
- Tijelo poruke – *body*, unutar kojeg se nalaze podaci o pozivu operacije ili odgovoru na operaciju od Web servisa i greškama nastalim tijekom obrade
- Poruke o greškama – *fault*, dio tijela poruke koji nije obavezan, u kojem se nalaze informacije (kodovi) o greškama nastalima tijekom prijenosa poruke.

---

<sup>18</sup> Erl T. (2008). Web Service Contract Design & Versioning for SOA. Prentice Hall., str.72.





**Slika 4.5. Dijelovi SOAP poruke**

#### 4.3.3. UDDI registar

UDDI (*engl. Universal Description, Discovery and Integration*) je registar za objavljivanje popisa servisa i njihovih usluga na Internetu, ili unutar nekog poduzeća na nekom od dostupnih poslužitelja. UDDI je neovisan o platformi, te se bazira na XML-u, koristeći SOAP komunikacijskom protokol.

UDDI registar organiziran je u:

- Bijele stranice (Osnovne informacije i identifikatori o firmi, kao naziv firme, adresa, kontaktne informacije, identifikacijski broj. Pomoću ovih informacija omogućava se pronalaženje web servisa prema poslovnim identifikatorima)
- Žute stranice (Informacije koje opisuju Web servis prema kategorijama, na osnovu kojih se može pronaći servis, tj. industrijska kategorizacija (npr. proizvodnja, prodaja))
- Zelene stranice (Tehničke informacije koje opisuju web servis i njegove funkcije. Pomoću ovih informacija dolazimo do grupiranih informacija o web servisu i njegovoj lokaciji.)

UDDI je razvijen kao jedan od osnovnih standarda web servisa i dizajniran da mu se može pristupati putem SOAP protokola. Ispitivati sadržaj registra i pristupati WSDL dokumentu registriranih web servisa.

Način na koji se koristi UDDI ovisi o perspektivi tko ga koristi. Iz perspektive poslovnog korisnika, UDDI je sličan Internet pretraživaču poslovnih procesa. Poslovni korisnik može pregledavati jedan ili više UDDI registara, kako bi uspoređivao web servise i ponuđena rješenja za poslovne procese i pripadne funkcije. Iz te perspektive, poslovni korisnik neće direktno pristupati UDDI registru, jer mu takvi podatci nisu u potpunosti čitljivi. Zbog čega se koristi prikaz tih informacija putem web stranica, kao što su <http://seekda.com/>, <http://www.xmethods.com/ve2/index.po> i slične.

Dok iz perspektive programera, koji koristi UDDI programski API, kako bi objavio servise. I koristi upite koji se direktno prosljeđuju UDDI registru, putem SOAP protokola. Kako bi se servisi dinamički otkrivali tijekom rada neke aplikacije ili servisa i isto tako dinamički koristile funkcionalnosti otkrivenih servisa.

## 5. Izrada Web servisa primjenom servisno orijentirane arhitekture

### 5.1. Servisno orijentirano modeliranje

Servisno orijentirana arhitektura je prirodni napredak nad objektnom orijentacijom i dizajna baziranog na komponentama.

Poslovni procesi su podržani s manjim programima, koje nazivamo 'komponentama', a logika (poslovna) koja se nalazi u tim komponentama bazirana je na objektno orijentiranom programiranju.

Pred nama se nalazi problem kako dizajnirati i osigurati funkcionalne servise koji će se kasnije moći lako prilagođavati i uklapati u nadolazeće promjene, što poslovne to i tehnološke, odnosno kako razviti servise prema principima SOA-e. Da bi smo omogućili što lakše upravljanje i prilagođavanje i daljnji razvoj servisa, moramo razviti okolinu o kojoj će takve promjene biti moguće. Odnosno razviti arhitekturu, u ovom slučaju servisno orijentiranu, koja je ključ za održavanje i ispunjavanje ciljeva servisno orijentirane arhitekture.

Da bi smo razumjeli smisao arhitekture, vodimo se sljedećim definicijama<sup>19</sup>:

1. arhitektura je formalni opis sustava, ili detaljan plan sustava za provođenje implementacije na razini komponentata
2. arhitektura je struktura komponentata, njihovih međusobnih veza, principa i smjernica za upravljanje njihovim dizajnom i evolucijom tijekom vremena

Arhitektura nam omogućuje:

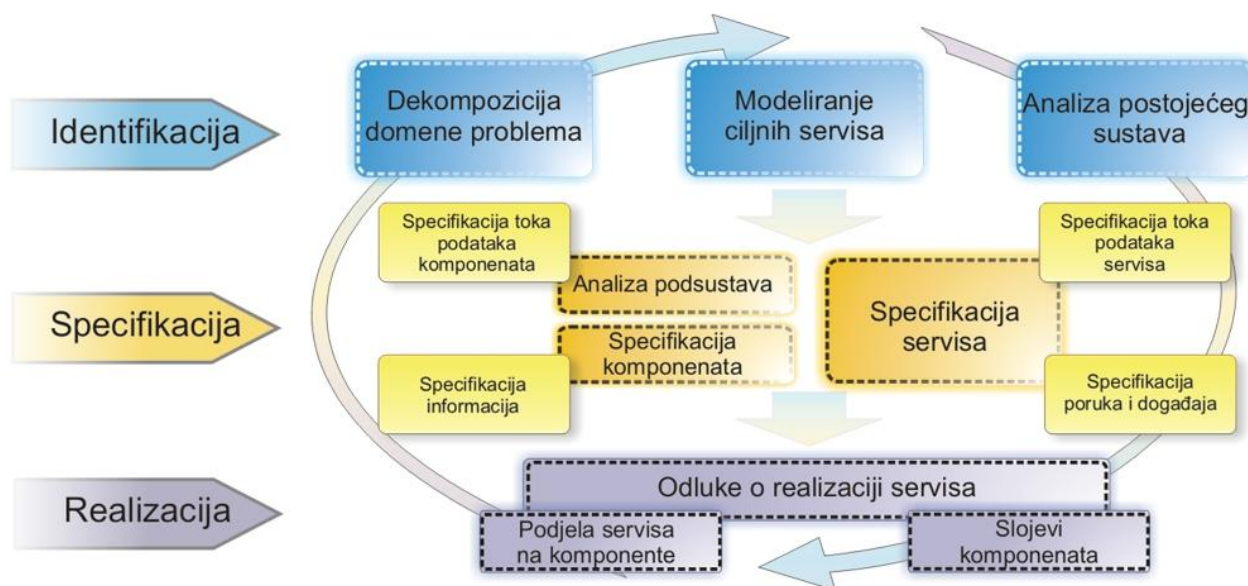
- dizajniranje i modeliranje na različitim razinama apstrakcije
- odvajanje specifikacija od implementacije
- izgradnju fleksibilnih sustava
- osigurava poznavanje poslovnih zahtjeva
- analizira utjecaj promjena u zahtjevima
- osigurava da se slijede definirani principi

---

<sup>19</sup> <http://www.ibm.com/developerworks/webservices/library/ws-soa-term1> , očitano: 18.07.2009.

Kako bi smo osigurali izgradnju kvalitetne arhitekture i implementirali servise, koristimo se servisno orijentiranim modeliranjem i arhitekturnim metodama (Slika 5.1.). Koja se sastoje od tri osnovne faze koje se mogu ponavljati ciklički tijekom procesa modeliranja<sup>20</sup>:

- identifikacija
- specifikacija
- realizacija



**Slika 5.1. Proces modeliranja arhitekture sustava**

### 5.1.1. Identifikacija servisa

Identifikacija servisa je ključna aktivnost servisno orijentirane analize. Cilj identifikacije servisa je identificirati grupe konceptualnih servisa i njihovih operacija.

Kako bi se servisi mogli identificirati, postoje mnogi pristupi, čiji se koraci kreću po osnovnim slojevima SOA-e, opisanim u poglavlju "3.3. Slojevi SOA-e".

Proces identifikacije sastoji se od kombinacije pristupa odozgo prema dolje, pristupa odozdo prema gore i pristupa od sredine (*engl. middle-out*).

Kako bi smo lakše prolazili kroz proces identifikacije, možemo se voditi sljedećim pitanjima<sup>21</sup>:

<sup>20</sup> <http://www.ibm.com/developerworks/library/ws-soa-design1>, učitano: 18.07.2009.

- *Koje promjene zahtjevi nastoje postići?*

To je najčešće glagol, koji ukazuje što funkcije nastoje postići. Najčešće proizlaze iz naziva poslovnih procesa.

- *Tko sve surađuje, kako bi se funkcije izvršile?*

Odgovorom na ovo pitanje možemo definirati tko izvodi osnovne akcije plivajućih traka (*engl. swimlanes*) u kojima se odvijaju srodne i povezane funkcije (npr. prodaja, dostava, narudžba).

- *Koje su funkcije odgovorne za izvršenje?*

- *Koje funkcije komuniciraju s kojim funkcijama?*

Na taj način se pomoću veza definiraju ovisnosti između funkcija, te kontrola i smjer toka podataka između funkcija i plivajućih traka

- *Koja su pravila. prema kojima funkcije međusobno djeluju?*

Definiraju se pravila ponašanja suradnje, pri čemu to mogu biti aktivnosti, međusobni odnosi, stanja.

#### **5.1.1.1. Odozgo prema dolje**

Nerazumno je očekivati efikasnu razinu integracije bez preciznog planiranja i dizajna. Jer se postojeće aplikacije ne mogu samo povezivati bez ikakvog plana, zbog čega će nastati mnogo poveznica koje će kasnije biti teško uređivati i održavati. Kako bi smo izbjegli takve probleme, moramo definirati arhitekturu u kojoj ćemo vršiti integraciju, nakon čega planiramo kako povezati postojeće aplikacije u skladu s pripremljenom arhitekturom. U tako pripremljenu arhitekturu moguće je uklapati nove aplikacije i tehnologije, te stvarati nove poveznice među njima, pri čemu nadograđujemo postojeću arhitekturu. Ovakav pristup integracije nazivamo pristupom odozgo prema dolje (*engl. top-down*).

Ovaj pristup zapravo definira arhitekturu za integraciju na najvišoj, globalnoj razini. Koja je sveobuhvatna, te dekomponira poslovnu domenu i njezina funkcijska područja i podsustave

---

<sup>21</sup> Amsden J., Modeling SOA: Part 1. Service identification, Dostupno na:  
<[http://www.ibm.com/developerworks/rational/library/07/1002\\_amsden/index.html?S\\_TACT=105AGX15&S\\_CMP=ART](http://www.ibm.com/developerworks/rational/library/07/1002_amsden/index.html?S_TACT=105AGX15&S_CMP=ART)>, Učitano: 18.07.2007.

definirajući tokove između pojedinih dijelova funkcija i procesa od kojih se sastoje. Nastoje se definirati sve očekivane promjene vezane uz integraciju, te se analiziraju postojeće ovisnosti pojedinih dijelova integracije, smjernice i definirane prioritete. Na taj način se pojačava integritet i konzistentnost arhitekture.

U kontekstu izrade servisa pristup odozgo prema dolje zagovara završavanje kompletne analize inventara. Što zahtjeva veća inicijalna ulaganja, zato što je na početku fokus na kreiranje plana servisnog inventara, zbog čega je dizajn servisa normaliziran, standardiziran i usklađen s pojedinim dijelovima integracije, smjernicama i definiranim prioritetima.

#### **5.1.1.2. Odozdo prema gore**

Pristup integracije odozdo prema gore (*engl. bottom-up*) usmjeren je na rješavanje prioritetnih poslovnih zahtjeva (pojedino nastali problemi) i glavnih objekta projekta. Na taj način se izbjegavaju dodatni troškovi, naponi i vrijeme koje je potrebno za izradu i isporuku servisa.

Ovaj pristup analizira postojeće IT komponente, te pronalazi funkcionalnosti koje možemo izložiti u obliku web servisa, kako bi bila korištena od više korisnika. Svojstvo višestruke iskoristivosti je jedna od važnih dijelova servisno orijentirane arhitekture, te je i kritična za njezin uspjeh. Prilikom izrade servisa prema funkcijama poduzeća, važno je paziti koje funkcije izlažemo, tj. dajemo na korištenje i treba paziti koje mogućnosti ćemo izložiti, kao što su kreiranje, čitanje, ažuriranje i brisanje podataka pomoću tih funkcija.

U pristupu odozdo prema gore, procesi ili postojeće komponente sustava su analizirani i prihvaćeni kao kandidati za daljnju izradu implementacije. U ovom procesu se analiziraju i procjenjuju API funkcije, transakcije, moduli iz postojećih paketa i aplikacija.

### 5.1.1.3. Pristup od sredine

Pristup od sredine sastoji se od modeliranja ciljnih servisa (*engl. goal-service modeling*) koji nisu prepoznati i definirani prilikom pristupa odozgo prema dolje i odozdo prema gore.

Svrha pristupa od sredine je usklađivanje potreba identificiranih pristupom odozgo prema dolje i odozdo prema gore. Gdje se zahtjeva suradnja poslovnih analitičara, programskih inženjera i specijalista za aplikacije.

### 5.1.2. Specifikacija servisa

Prilikom dizajniranja servisa prema SOA-i, važno je biti svjestan principa servisno-orijentiranog dizajna, te ih nastojati ispuniti. Ključni principi koje trebamo imati na umu tijekom procesa specifikacije su:

- Višestruka iskoristivost
- Slaba povezanost
- Protočnost
- Sastavljivost
- Autonomnost

Svrha specifikacije servisa je dizajniranje strukture i ponašanja servisnih komponenata koje su od arhitekturne važnosti.

Specifikacija servisa bi trebala sadržavati sljedeće informacije<sup>22</sup>:

- Naziv servisa, koji ukazuje što servis radi ili o čemu je riječ (svrha)
- Definirana sučelja servisa, gdje se opisuju funkcionalne osobine koje trebaju sadržavati:
  - naziv, koji je obično glagol
  - definirani ulazni ili izlazni zahtjevi ili potrebni podaci
  - treba osigurati standardne uvjete korištenja od strane korisnika

---

<sup>22</sup> Amsden J., Modeling SOA: Part 2. Service specification, Dostupno na:  
<[http://www.ibm.com/developerworks/rational/library/07/1009\\_amsden/index.html?S\\_TACT=105AGX15&S\\_CMP=ART](http://www.ibm.com/developerworks/rational/library/07/1009_amsden/index.html?S_TACT=105AGX15&S_CMP=ART)>, Učitano: 18.07.2007.

- treba osigurati standardizirane izlaze koje korisnik može očekivati pri uspješnom izvršenju funkcije
- predvidjeti mogućnost pogrešaka i obavještanje korisnika, ako se funkcija iz nekog razloga ne može izvršiti
- komunikacijski protokol i pravila prema kojima se funkcije mogu koristiti i u kojem redoslijedu
- mogućnosti koje se očekuju od korisnika, kako bi mogao koristiti servis
- zahtjevi koje mora ispuniti implementator prilikom pružanja servisa
- uvjeti prema kojima se može ocijeniti u kojoj mjeri servis ispunjava svoju funkcionalnost
- kvaliteta servisa koju korisnik može očekivati i davatelj usluge servisa može dati (npr. cijena, dostupnost, performanse, pokrivenost, prilagođenost rješavanju problema, kompetentne informacije)
- pravila korištenja i sigurnost servisa

#### **5.1.2.1. Analiza podsustava**

U jedan servis se mogu staviti sve operacije, kako bi smo imali jednostavno rješenje. U tom slučaju klijenti ovise o tom jednom servisu, što kao rezultat ima visok stupanj povezanosti. Ako se naprave neke promjene na servisu, tada one mogu uzrokovati i promjene korištenja kod korisnika. Cilj je da korisnici što manje osjete promjene ili im se prilagode na što jednostavniji način.

Može se napraviti i servis za svaku funkciju, što opet rezultira sa složenim sustavom povezanih servisa koji nemaju dobro organiziranu cjelinu i koheziju. Kod takvog pristupa teško je definirati i komunikacijski model između korisnika i servisa, kako bi se održao komunikacijski protokol koji mora biti zadovoljen prilikom korištenje funkcija.

Na koji način će se servisi grupirati i sa kojim skupom funkcionalnosti, ovisi o vještinama dizajnera i traženih zahtjeva. U ovom koraku se analiziraju podsustavi koji predstavljaju servise koji zajedno čine rješenje. Kreira se objektni model s podsustavima, definiraju se ovisnosti i tokovi između njih u cilju postizanja zadovoljavajućeg traženog rješenja.

Bitna je i lokacija gdje je servis postavljen, koja je povezana s korisnikom i ostalim servisima, te može imati veliki efekt na cjelokupno rješenje, dostupnost i sigurnost. Iz tog razloga treba paziti na koje podsustave, odnosno servise ćemo podijeliti projektno rješenje.



### **5.1.2.2. Specifikacija komponenata**

Odvajanjem poslovnih zahtjeva od servisa, stvaramo fleksibilnost dizajniranja SOA-e koja može ispuniti poslovne i tehnološke zahtjeve.

Modeliranje servisa započinje njihovom identifikacijom. Tako identificirane servise klasificiramo u hijerarhiju i određujemo veze između pojedinih servisa. Za veze definiramo tipove poruka i stanja koja su moguća prilikom komunikacije. Za svaki servis definiramo od kojih komponenata se servis sastoji i definiramo veze i ovisnosti između pojedinih komponenata. Te komponente mogu biti već postojeći moduli, koje možemo ponovno iskoristiti ili komponente koje je potrebno izraditi iz početka.

Prilikom specifikacije detalja o komponenata koje izgrađujemo iz početka, možemo definirati sljedeće specifikacije:

- podatkovne
- pravila
- servise
- konfiguracijske profile
- varijacije

Ako odvojimo pojedine servise, odnosno da servisi ne ovise jedni o drugima, možemo nesmetano razvijati svaki od njih. Gdje na kraju možemo kombinirati funkcionalnosti pojedinih servisa, kako bi se ostvarili traženi zahtjevi.

Dok pojedine komponente servisa možemo rasporediti u pakete, pri čemu odvajamo domene problema u različite funkcionalne skupine. Što pomaže u upravljanju izrade pri povećanju složenosti projekta (zahtjeva), osigurava se potrebna radna okolina, povećava mogućnost ponovnog iskorištavanja i drže granice pojedinih problema unutar paketa.

Pri radu na projektu, možemo i servise rasporediti prema paketima, te ako kasnije postoji potreba razlaganja servisa izvan projektnog rješenja, Lako možemo te pakete prebaciti u nove projekte, s paketima o kojima servisi ovise. Paketi o kojima servisi najčešće ovise su paketi koji služe za rad s bazama podataka, čitanje lokalnih datoteka i slično.

### 5.1.3. Realizacija servisa

Korak realizacije servisa nam pomaže prepoznati programe koje već imamo ili njihove module, kako bi smo ih mogli ponovo iskoristiti za funkcionalnost servisa. Postojeće komponente koje se nalaze na razini sustava integracijom, transformacijom ili outsourcingom pridružujemo web servisu, kako bi njihove funkcije bile dostupne većem broju korisnika. Čime se osigurava višestruka iskoristivost postojećih komponenti.

Postoji i mogućnost da moramo razvijati potrebne komponente od samog početka, ili kombinacijom postojećih modula i nekih dodatnih operacija koje moramo isto tako razviti iz početka. Odluke koje se još vežu uz realizaciju servisa su sigurnost, upravljanje i nadziranje servisa.

Nakon konačnog plana realizacije, i odluke koje komponente se ponovno iskorištavaju, a koje kodiraju ili prilagođavaju, slijedi korak implementacije, odnosno izrada rješenja prema izrađenom arhitekturnom modelu.

## 5.2. Implementacija servisa

Kako bi smo implementirali rješenje, moramo kreirati platformu, koja je u skladu s arhitekturnim dizajnom definiranim prilikom specifikacije i modeliranja servisa.

Za kreiranje takve platforme koristimo se postojećim modelom, kako bi smo izgradili osnovnu strukturu („kostur“) rješenja. Nakon čega se izrađuju, programiraju ostale potrebne komponente nužne za funkcioniranje željenih operacija.

Prije izrade osnovne strukture, odabiremo platformu na kojoj ćemo razvijati projektno rješenje kao što su J2EE, .Net, u nekim od podržanih programskih jezika od strane platforme (Java, ASP, C#).

### 5.2.1. Platforme za razvoj Web servisa

Za implementaciju Web servisa možemo koristiti danas najpopularnije dvije razvojne platforme, Java2EE (*Enterprise Edition*) i .Net. Te razvojna okruženja koja podržavaju programski jezik PHP. Od kojih svaki ima vlastite karakteristike, razvojna okruženja i ključne razvojne API funkcije i datoteke koje podržavaju razvoj Web servisa.

Kako bi se pomoću PHP-a razvio Web servis, potrebno je koristiti određene dodatke, koji podržavaju rad sa SOAP protokolom od strane PHP servera. Korištenjem tih dodataka, moguće je kreirati Web servis, temeljen na XML-RPC tehnologiji. Dodatci koji omogućuju rad sa SOAP protokolom su standardni dio instalacije PHP-a verzije 5, samo ih je potrebno naknadno uključiti nakon instalacije, kako bi ih server inicijalizirao.

.Net je razvojno okruženje koje je proizveo Microsoft, koje ima podršku za više programskih jezika. Za izradu Web servisa, koristi se ASP.NET Web services, a kako bi se podržali razni dodatci koji proširuju mogućnosti Web servisa koristi se Microsoft WSE (*Web Services Enhancements*). Za programiranje logike Web servisa može se koristiti Visual Basic, C++ ili najčešće C#.

Prilikom razvoja Web servisa pomoću J2EE, koristi se programski jezik Java, pri čemu se mogu koristiti razne razvojne okoline koje podržavaju Javu. Jedna od najpristupačnijih je NetBeans, koji ima mnoge implementirane mehanizme za pomoć prilikom izgradnje Web servisa, kao što su generiranje WSDL dokumenata, te uključivanje mnogih API-ja za implementaciju Web servisa. Za integraciju Web servisa koji predstavljaju Web aplikacije pisane u Javi, koristi se 'Web Service Stack' koji nosi naziv Metro. On sadrži podršku za JAX-WS, JAXB i WSIT, koji omogućuju razvoj sigurnih, pouzdanih, interoperabilnih i transakcijskih Web servisa i klijenata.

Web servisi izrađeni u Javi, mogu se razvijati kao Web aplikacije, koje posjeduju sučelje Web servisa, ili mogu biti izgrađene kao aplikacije koje umećemo u gotove Web servise sa definiranim sučeljem prema željenim funkcijama naše aplikacije. Neke od korištenijih Web aplikacija čija je namjena usluživanje Web servisa su Axis2 i Spring-WS.

Prednosti korištenja JAX-WS nalazi se u tome što je implementiran u obje verzije Java SE 6 i Java EE 5. Te podržava SOAP i RESTfull komunikacijski protokol, a za rad s podatkovnim modelom koristi JAXB, a parsiranje XML poruka radi s StAX (Streaming API for XML). Protokoli koji su podržani za prijenos poruka su HTTP, SMTP i JMS.

Korištenjem Axis2, koji je zapravo Web aplikacija u koju umećemo gotovu programsku logiku servisa ili logika neke aplikacije kojoj smo dodali sučelje servisa. Axis2 za obradu XML poruka koristi AXIOM (AXIs Object Model), koji je baziran na StAX-u, te nudi neke dodatne opcije. Pomoću Axis2 mogu se implementirati Web servisi koji su bazirani na SOAP ili RESTfull protokolu. Protokoli koji su podržani za prijenos poruka su HTTP, TCP, SMTP i JMS.

Spring-WS je jedan od najpopularnijih proizvoda u području razvoja Web servisa. Koji nastoji što više olakšati i pojednostaviti izradu Web servisa. Web servisi se mogu kreirati iz WSDL dokumenata, a kod Spring-WS, taj proces započinje već s XSD dokumentom, pri čemu se generira WSDL dokument, na temelju kojeg se generira Java kod. Jedan od ključnih pogleda na korištenje Spring-WS je uvođenje aspektno-orijentirane filozofije i uvođenje koncepata 'preokret u kontroli' (IoC – Inversion of Control) i injekcija ovisnosti (Dependency Injection).

## 5.2.2. Implementacija

SOA je prirodni napredak nad objektnom orijentacijom i dizajna baziranog na komponentama (*engl. CBD - component-based development*).

Poslovni procesi su podržani s manjim programima, koje nazivamo komponentama, a poslovna logika koja se nalazi u tim komponentama bazirana je na objektno orijentiranom programiranju.

Prilikom programiranja servisne logike, oslanjamo se na model koji smo izradili tijekom procesa modeliranja. Tijekom tog procesa organizirali smo i programske komponente u pakete, koji nam prilikom pisanja programa mogu poslužiti kao osnova za strukturiranje i organiziranje projektnih datoteka.

Kodiranje Web servisa možemo raditi u nekoj od podržanih platformi, korištenjem priпадnih programskih jezika od strane platforme. Prilikom izrade projekta, dostupni su nam i mnogi alati koji podržavaju podržane platforme, te izbor takvog alata ili više njih ovisi o programeru, timu programera ili ako je specificirano nekim dodatnim uvjetima izrade u projektnom zahtjevu.

Prije kodiranja, potrebno je osigurati radno okruženje u kojem će se servisi i njegove komponente izvršavati. Osigurati dostupnost potrebnim podacima, bazama podataka (te ih izraditi ako je potrebno u kontekstu projekta), konfiguracijskim datotekama i slično.

Nakon čega slijedi proces kodiranja, prema projektnim zahtjevima, koji su dodatno elaborirani tijekom procesa modeliranja. I služe programeru za lakše razumijevanje zahtjeva.

Mnogi alati omogućavaju da se prilikom programiranja, dizajniraju i servisna sučelja. Prema kojima se može generirati servisni ugovor, tj. WSDL dokument, kojeg je moguće putem sučelja alata dodatno modificirati, te ga kao takvog objaviti zajedno sa servisom na nekom od servera koji podržavaju korištenu platformu.

Izrada servisa može kretati od izrade programskog koda, prema kojem se generira servisni ugovor. Ili može kretati od izrade servisnog ugovor, odnosno WSDL dokumenta, pisanog u XML jeziku, na temelju kojeg se može generirati pripadni programski kod, koji dodatno možemo upotpuniti vlastitim preinakama i dopunama, ako su potrebne.

### 5.2.3. Testiranje

Testiranje se može odvijati tijekom cijelog procesa izrade. Prilikom čega se najčešće testira programska logika i izlazi koje daje servis, za pojedine komponente koje će na kraju činiti servisne komponente.

Na kraju je potrebno testirati cjelokupni servis, njegovu logiku i ponašanje servisa pri većem opterećenju. Prije testiranja, potrebno je organizirati testiranje, definirati parametre testiranja, uvjete izvođenja, te znati koji izlazi se očekuju, kako bi se provjerila programska logika. Ako servis prođe sve definirane testove, može se krenuti na objavljivanje servisa.

### 5.2.4. Objavljivanje

Objavljivanje servisa je proces obznanjivanja javnosti ili određenim grupama korisnika da servis postoji. Objavljuje se njegov informacije koje opisuje namjenu i funkcije servisa, načine korištenja pojedinih funkcija i moguće rezultate.

Service je moguće objaviti javno, kako bi pristup tim servisima imala javnost. U tom slučaju service i njihove lokacije možemo registrirati na nekim od specijaliziranih stranica, kao

što su `http://webservices.seekda.com/` i `http://www.xmethods.com/ve2/index.po`, koje putem web sučelja, omogućavaju pretraživanje web servisa, s njihovim opisima i WSDL dokumentima.

Drugi način objavljivanja informacija o web servisu je na jednom centralnom mjestu za kojeg znaju određeni korisnici ili korisnici koji imaju pravo pristupa (na primjer unutar firme). Na takvom centralnom mjestu, serveru, potrebno je imati podršku za UDDI registar. Postoje mnoge implementacije UDDI registra. Neki od njih su jUDDI i OpenUDDI, koji se postavljaju na Apache Tomcat u obliku web aplikacije.

### 5.3. Održavanje web servisa

Nakon objavljivanja servisa, potrebno ih je dodatno održavati. Kako servisi ne bi zastarjeli, i kako bi im se produžio životni vijek. Načine na koje možemo održavati web servise mogu biti od održavanja servera, pa do doradivanja servisa. Dodatnim konfiguriranjem servisa, proširivanjem njihovih mogućnosti, i usklađivanjem funkcija s potrebama korisnika, kao mijenjanje prava korištenja servisa.

Sljedeća obilježja karakteriziraju upravljanje web servisima<sup>23</sup>:

- upravljanje s pogodbama servisnih razina (*engl. SLA. - Service-Level Agreement*), koji uključuje upravljanjem kvalitete servisa (*engl. QoS*), vremenom odaziva servisa
- provjeravanje, nadziranje i uklanjanje pogrešaka – uključuje prikupljanje statističkih podataka o servisu i performansama, i pogreškama
- rezerviranje resursa za servise – kako bi servis bio pouzdaniji i sigurniji
- upravljanje servisom – prosljeđivanje poruka do drugih servisa, kompatibilnost između Java i .Net klijenata, modifikacija servisnog registra i SOAP poruka, izmjene u programskoj podršci servisa, nadogradnja servisa i upravljanje komponentama servisa

---

<sup>23</sup> \*\*\*, Web Services Life Cycle: Managing Enterprise Web Services, Dostupno na:  
<[http://www.sun.com/software/whitepapers/webservices/wp\\_mngwebsvcs.pdf](http://www.sun.com/software/whitepapers/webservices/wp_mngwebsvcs.pdf)>, učitano: 01.08.2009.

## 6. Projektni primjer

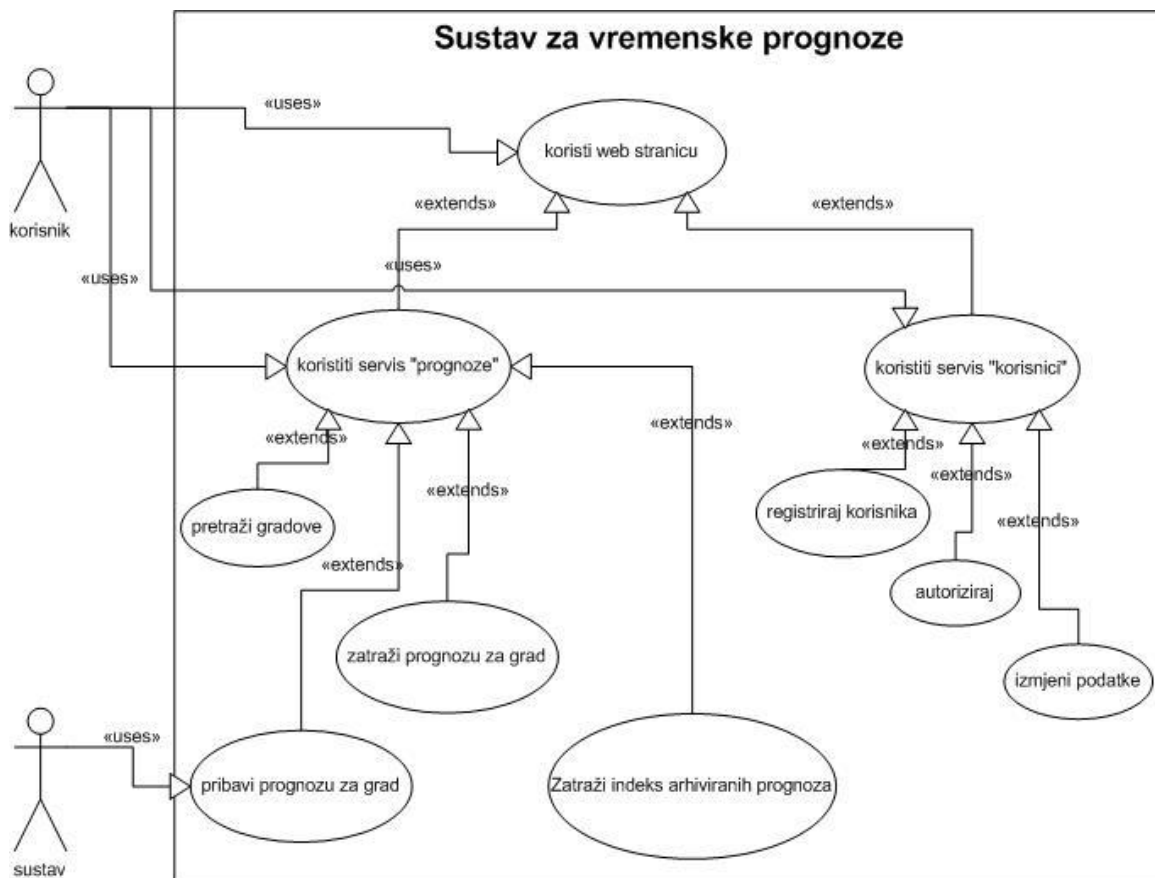
### 6.1. Korisnički zahtjev

#### 6.1.1. Opis korisničkih zahtjeva

Web aplikacija treba omogućavati korisnicima pristup informacijama o vremenskim prognozama putem servisa, za gradove SAD-a, za koje postoje mjerenja. Podaci o vremenskim prognozama se bilježe na dnevnoj bazi, kako bi korisnici imali uvid u prijašnje vremenske prognoze, za pojedine gradove. Korisnik može pretraživati gradove SAD-a prema nazivu grada, ZIP kodu, nazivu države, okruga ili svojstvu da se pretražuju gradovi za koje postoji arhiva vremenskih prognoza. Prilikom čega u rezultatu pretrage treba biti naglašeno za koje gradove postoje vremenske prognoze, a za koje trenutno ne postoje. Ako prognoza ne postoji za željeni grad, korisniku treba ponuditi prognozu za prvu bližu lokaciju. Korisniku treba omogućiti i izlistanje onih gradova za koje postoji arhiva vremenskih prognoza. Isto tako treba biti moguće izlistati arhivirane vremenske prognoze po datumima i vremenu arhiviranja, za gradove koji imaju arhivu vremenskih prognoza.

Korisnici koji žele pristupati informacijama putem web stranice, a ne direktno putem servisa, trebaju biti prethodno registrirani. Putem web stranice, treba omogućiti dostupnost svim informacijama koje su dostupne i putem servisa vremenskih prognoza.

Projektno rješenje nema unaprijed predefinirane vremenske prognoze i gradove za koje se traže prognoze, te treba omogućiti da se tijekom vremena i učestalim korištenjem, povećava opseg dostupnih vremenskih prognoza za pojedine gradove.



Slika 6.1. Slučajevi korištenja

## 6.1.2. Slučajevi korištenja

### 6.1.2.1. Pretraživanje gradova SAD-a

Use Case	Pretraživanje gradova SAD-a
<b>Opis (Description)</b>	Pretraživanje gradova SAD-a prema nazivu grada i/ili ZIP koda, i/ili nazivu države i/ili nazivu okruga i/ili gradove za koje postoji arhiva vremenskih prognoza Korisnik šalje pomoću SOAP protokola (a) ili putem Web stranice (b), zahtjev za izlistanje dostupnih gradova SAD-a
<b>Sudionici (Actors)</b>	Korisnik, sustav
<b>Pretpostavke (Assumptions)</b>	a. Korisnik web servisa uvijek može pretraživati gradove prema definiranim kriterijima b. Korisnik web stranice mora biti prijavljen da bi pretraživao gradove prema definiranim kriterijima



<b>Koraci (Steps)</b>	1a. klijent šalje upit s definiranim parametrima pretraživanja servisu pomoću SOAP poruke 1b. Klijent definira parametre pretraživanja na Web stranici 2a. dobiva rezultate od Web servisa u XML formatu 2b. Dobiva rezultate pretrage prikazane na Web stranici 3. Korisnik zatražuje vremensku prognozu za željeni grad (use case 3)
<b>Varijacije (Variations)</b>	Korisnik ne mora unijeti sve parametre pretraživanja, samo željene, kako bi suzio izbor pretrage. U rezultatu pretrage mora biti koji grad ima koliko arhiviranih vremenskih prognoza.
<b>Nefunkcionalni zahtjevi (Non-Functional)</b>	
<b>Problematika (Issues)</b>	

#### 6.1.2.2. Popis arhiviranih prognoza grada

<b>Use Case</b>	Popis arhiviranih prognoza grada
<b>Opis (Description)</b>	Korisnik šalje upit web servisu ( <b>a</b> ) ili putem web stranice ( <b>b</b> ) za izlistanje datuma svih arhiviranih vremenskih prognoza za željeni grad.
<b>Sudionici (Actors)</b>	Korisnik, sustav
<b>Pretpostavke (Assumptions)</b>	a. Korisnik web servisa uvijek može pregledavati listu arhiviranih datuma vremenskih prognoza b. Korisnik web stranice mora biti prijavljen da bi pregledavao arhivirane datume vremenskih prognoza
<b>Koraci (Steps)</b>	1a. Korisnik šalje zahtjev web servisu za popisom datuma arhiviranih vremenskih prognoza, prema ZIP kodu grada 1b. Korisnik web stranice prilikom odabira grada (use case 1), zatražuje popis arhiviranih datuma 2a. Korisnik dobiva popis datuma arhiviranih vremenskih prognoza za traženi grad 2b. Korisniku web stranice se prikazuju arhivirani datumi ako ih grad posjeduje.
<b>Varijacije (Variations)</b>	Ako korisnik zatraži popis arhiviranih datuma za grad koji nema vremenskih prognoza, tada se korisniku vraća prva bliži grad koji ima vremensku prognozu, ali ne nužno i arhivu.
<b>Nefunkcionalni zahtjevi (Non-Functional)</b>	
<b>Problematika (Issues)</b>	

### 6.1.2.3. Prikaz prognoze za grad

<b>Use Case</b>	Prikaz prognoze za grad
<b>Opis (Description)</b>	a. Korisnik šalje SOAP upit web servisu za prikaz vremenskih prognoza , na određeni datum, ako postoji u arhivi b. Korisnik odabirom rezultata pretrage na web stranici (use case 1) zatražuje prikaz vremenske prognoze za željeni grad
<b>Sudionici (Actors)</b>	Korisnik, sustav
<b>Pretpostavke (Assumptions)</b>	c. Korisnik web servisa uvijek može pregledavati prognoze gradova d. Korisnik web stranice mora biti prijavljen da bi pregledavao vremenske prognoze za gradove
<b>Koraci (Steps)</b>	1a. Korisnik zatražuje od web servisa zahtjev za vremensku prognozu za željeni grad 1b. Korisnik na web stranici zatražuje prikaz prognoze za željeni grad 2a. Korisnik dobiva vremensku prognozu za željeni grad 2b. Korisnik dobiva vremensku prognozu za željeni grad, i prikazuje se arhiva vremenskih prognoza ako postoji (use case 2)
<b>Varijacije (Variations)</b>	<ul style="list-style-type: none"><li>• Ako korisnik ne definira datum, prikazuje se zadnja vremenska prognoza.</li><li>• Za svaki grad se prikazuje koliko ima arhiviranih vremenskih prognoza.</li><li>• Ako ne postoji prognoza za traženi datum, prikazuje se prognoza za prvi bliži datum, s dodatnom informacijom korisniku da ne postoji prognoza na zatraženi datum.</li></ul>
<b>Nefunkcionalni zahtjevi (Non-Functional)</b>	
<b>Problematika (Issues)</b>	Ako ne postoji prognoza za traženi grad, treba ju pribaviti s udaljenog servisa. Ako i tamo ne postoji vremenska prognoza za traženi grad, treba obavijestiti korisnika o nepostojanju informacija i dati mu na izbor prvu bližu lokaciju

#### 6.1.2.4. Pribavljanje vremenskih prognoza

<b>Use Case</b>	Pribavljanje vremenskih prognoza
<b>Opis (Description)</b>	Pribavljanje vremenskih prognoza za određeni grad
<b>Sudionici (Actors)</b>	Sustav
<b>Pretpostavke (Assumptions)</b>	
<b>Koraci (Steps)</b>	<ol style="list-style-type: none"><li>1. sustav pretražuje lokalnu bazu podataka za vremenskom prognozom za određeni grad</li><li>2. ako nema vremenskih prognoza za traženi grad, kontaktira se udaljeni servis</li><li>3. ako udaljeni servis posjeduje vremensku prognozu za taj grad, vraća pripadne podatke. Ako ne posjeduje vremensku prognozu, vraća prognozu prve bliže lokacije.</li><li>4. Ako je sustavu vraćena prognoza za željeni grad, arhivira se.</li></ol>
<b>Varijacije (Variations)</b>	<ul style="list-style-type: none"><li>• Ako je zahtjev za vremenskom prognozom proizašao iz „Prikaz prognoze za grad“ (use case 3), tada se korisniku prosljeđuje zamjenska lokacija.</li><li>• Sustav periodički na dnevnoj bazi zatražuje dnevne vremenske prognoze od udaljenog servisa, za gradove koji posjeduju arhivirane podatke.</li></ul>
<b>Nefunkcionalni zahtjevi (Non-Functional)</b>	
<b>Problematika (Issues)</b>	Treba optimizirati broj upita prema udaljenom servisu koji posjeduje dnevne vremenske prognoze.

#### 6.1.2.5. Registracija korisnika

<b>Use Case</b>	Registracija korisnika
<b>Opis (Description)</b>	Da bi korisnik koristio web stranicu, prethodno mora biti registriran.
<b>Sudionici (Actors)</b>	Korisnik, sustav
<b>Pretpostavke (Assumptions)</b>	
<b>Koraci (Steps)</b>	<ol style="list-style-type: none"><li>1. korisnik otvara formu za prijavu na sustav</li><li>2. na formi se nalazi link na formular za registraciju</li><li>3. korisnik popunjava formular, i šalje zahtjev</li><li>4. korisnik dobiva informaciju o registraciji</li><li>6. korisnik koristi sustav (use case 1)</li></ol>
<b>Varijacije (Variations)</b>	
<b>Nefunkcionalni zahtjevi (Non-Functional)</b>	
<b>Problematika (Issues)</b>	Postoji mogućnost da je servis za rad s korisnicima nedostupan

#### 6.1.2.6. Autentifikacija korisnika

<b>Use Case</b>	Autentifikacija korisnika
<b>Opis (Description)</b>	Nakon registracije korisnika, prilikom prijavljivanja korisnika na stranicu, provjeravaju se dali su korisnički podaci ispravni.
<b>Sudionici (Actors)</b>	Korisnik, sustav
<b>Pretpostavke (Assumptions)</b>	
<b>Koraci (Steps)</b>	1. korisnik na web stranici otvara formu za prijavu na stranicu 2. na formi za prijavu unosi svoje korisničke podatke za prijavljivanje na sustav (korisničko ime i lozinku) 3. korisnički servis provjerava dali su podaci ispravni, te dali je korisnik registriran 4a. Ako su korisnički podaci ispravni, korisnik se uspješno prijavljuje na sustav 4b. Ako korisnički podaci nisu ispravni, korisnik se ne može spojiti na sustav
<b>Varijacije (Variations)</b>	Korisnika treba obavijestiti o mogućim problemima prilikom prijavljivanja na sustav
<b>Nefunkcionalni zahtjevi (Non-Functional)</b>	
<b>Problematika (Issues)</b>	Postoji mogućnost nedostupnosti servisa za rad s korisnicima, o čemu korisnika treba obavijestiti.

#### 6.1.2.6. Izmjena korisničkih podataka

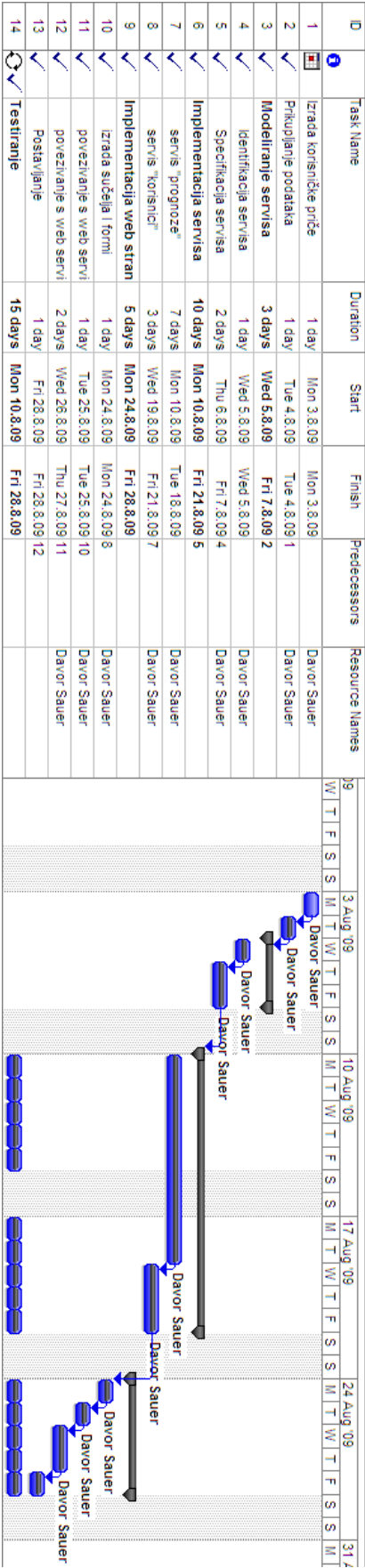
<b>Use Case</b>	Izmjena korisničkih podataka
<b>Opis (Description)</b>	Nakon registracije korisnika i uspješne autentifikacije, korisnik može promijeniti lozinku za pristup sustavu.
<b>Sudionici (Actors)</b>	Korisnik, sustav
<b>Pretpostavke (Assumptions)</b>	
<b>Koraci (Steps)</b>	1. korisnik na web stranici otvara formu o korisničkim podacima 2. odabirom opcije izmjena lozinke, otvara se mjesto za unos nove lozinke 3. odabirom linka prihvati, prihvaća se nova korisnička lozinka
<b>Varijacije (Variations)</b>	
<b>Nefunkcionalni zahtjevi (Non-Functional)</b>	
<b>Problematika (Issues)</b>	Postoji mogućnost nedostupnosti servisa za rad s korisnicima, o čemu korisnika treba obavijestiti.

## 6.2. Prikupljanje podataka

Potrebni su podaci o gradovima SAD-a (naziv grada, države, okruga i ZIP kod), kojima će sustav moći lokalno pristupati. Podatke o gradovima moguće je pronaći na internet adresi: <http://www.free-zipcodes.com/download-zip-code-database.php>, koji su već pripremljeni za unos u bazu podataka, koje će sustav moći koristiti za lokalno pretraživanje gradova SAD-a.

Servis koji nudi dnevne vremenske prognoze WeatherBug (<http://weather.weatherbug.com>), i njegov WSDL nalazi se na internet adresi: <http://api.wxbug.net/weatherservice.asmx?WSDL>.

6.3. Plan projekta



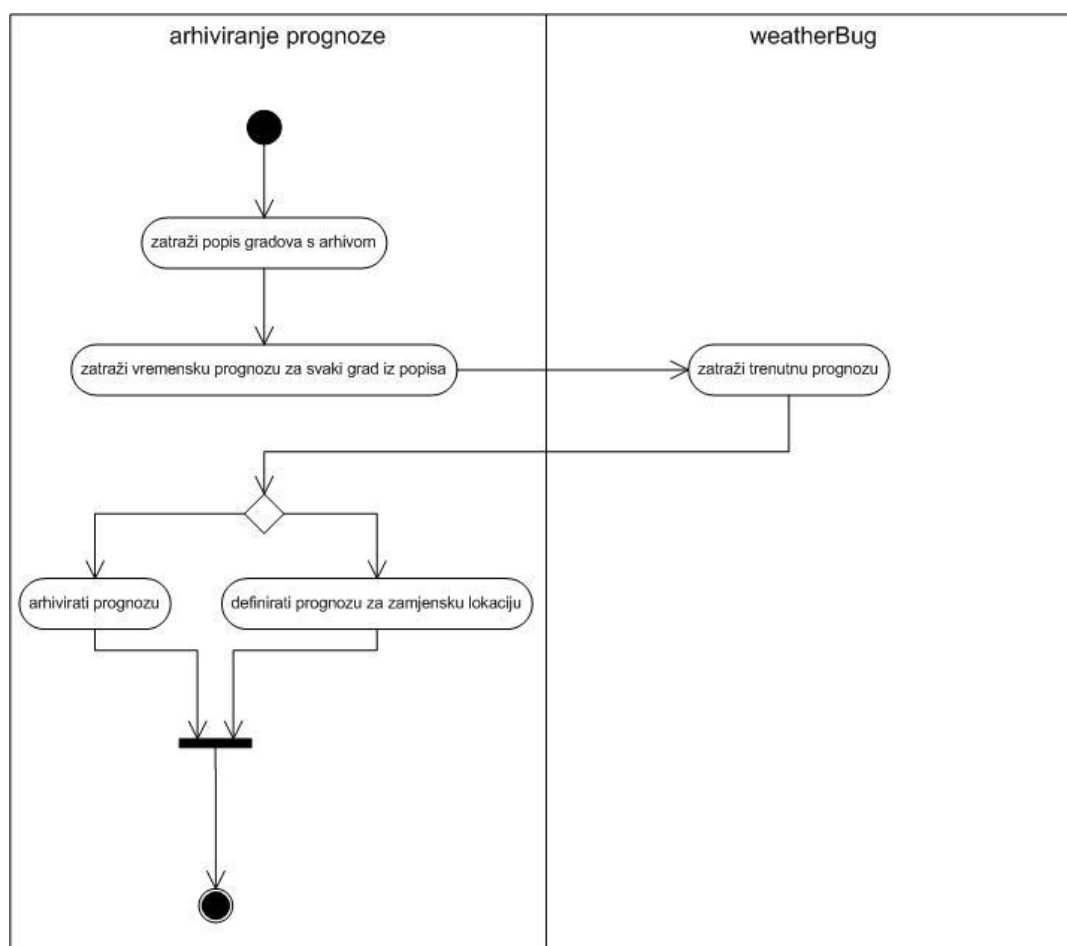
Slika 6.2. Plan izrade projekta

## 6.4. Modeliranje servisa

### 6.4.1. Identifikacija servisa

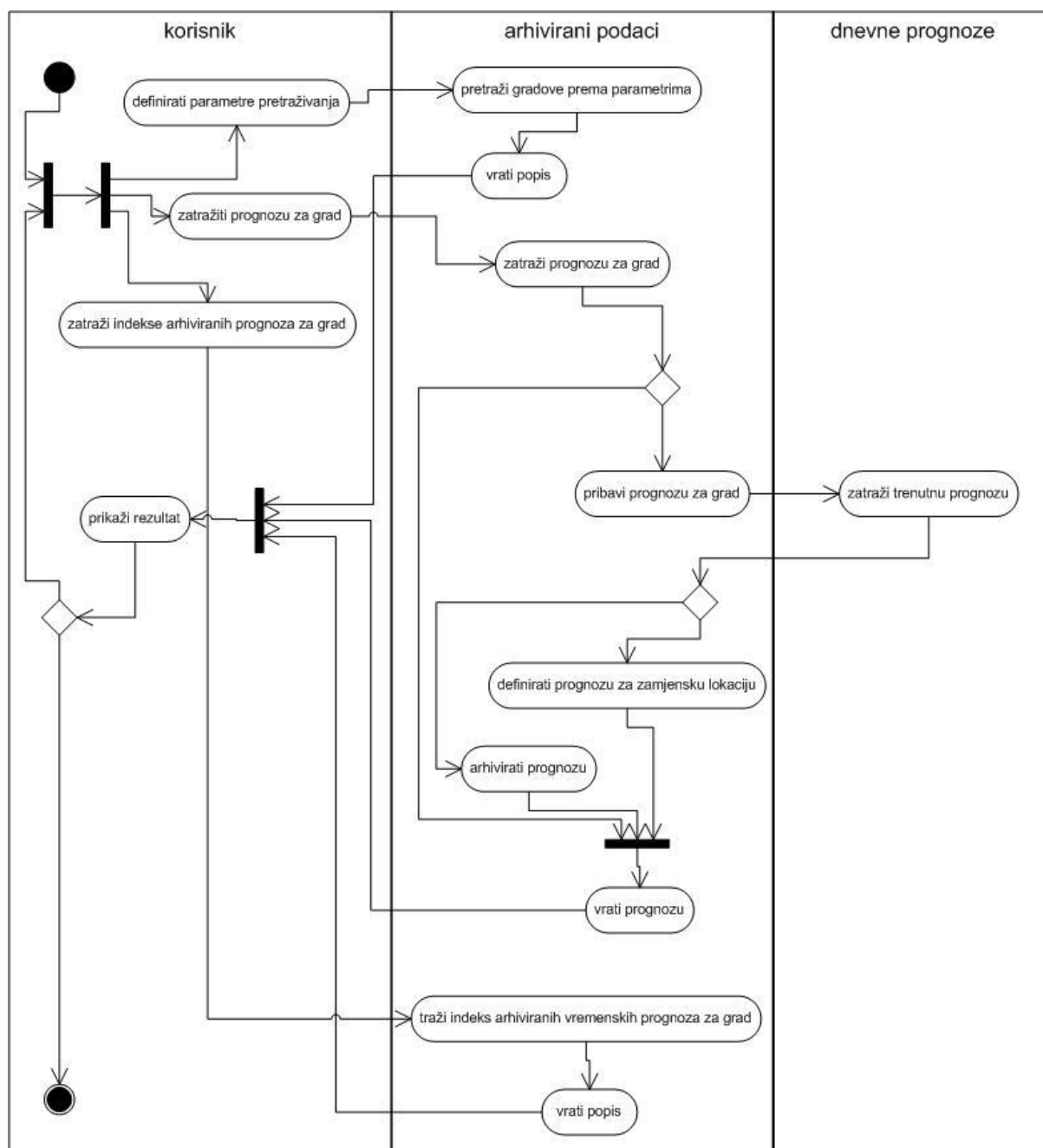
Iz korisničkih priče možemo definirati tri dijagrama aktivnosti. Jedan dijagram aktivnosti se odnosi na korištenje servisa koji arhivira vremenske prognoze za pojedine gradove od strane korisnika, dok se drugi odnosi na korištenje web stranice od strane korisnika, koja koristi aktivnosti iz prvog dijagrama aktivnosti za pristupanje i interpretiranje arhiviranih podataka. Zadnji dijagram aktivnosti se odnosi na pozadinski proces koji se izvodi za aktivnosti arhiviranja dnevnih vremenskih prognoza.

#### 6.4.1.1. Dijagram aktivnosti arhiviranja prognoza



**Slika 6.3. Prikaz aktivnosti pozadinskog procesa  
za arhiviranje dnevnih vremenskih prognoza**

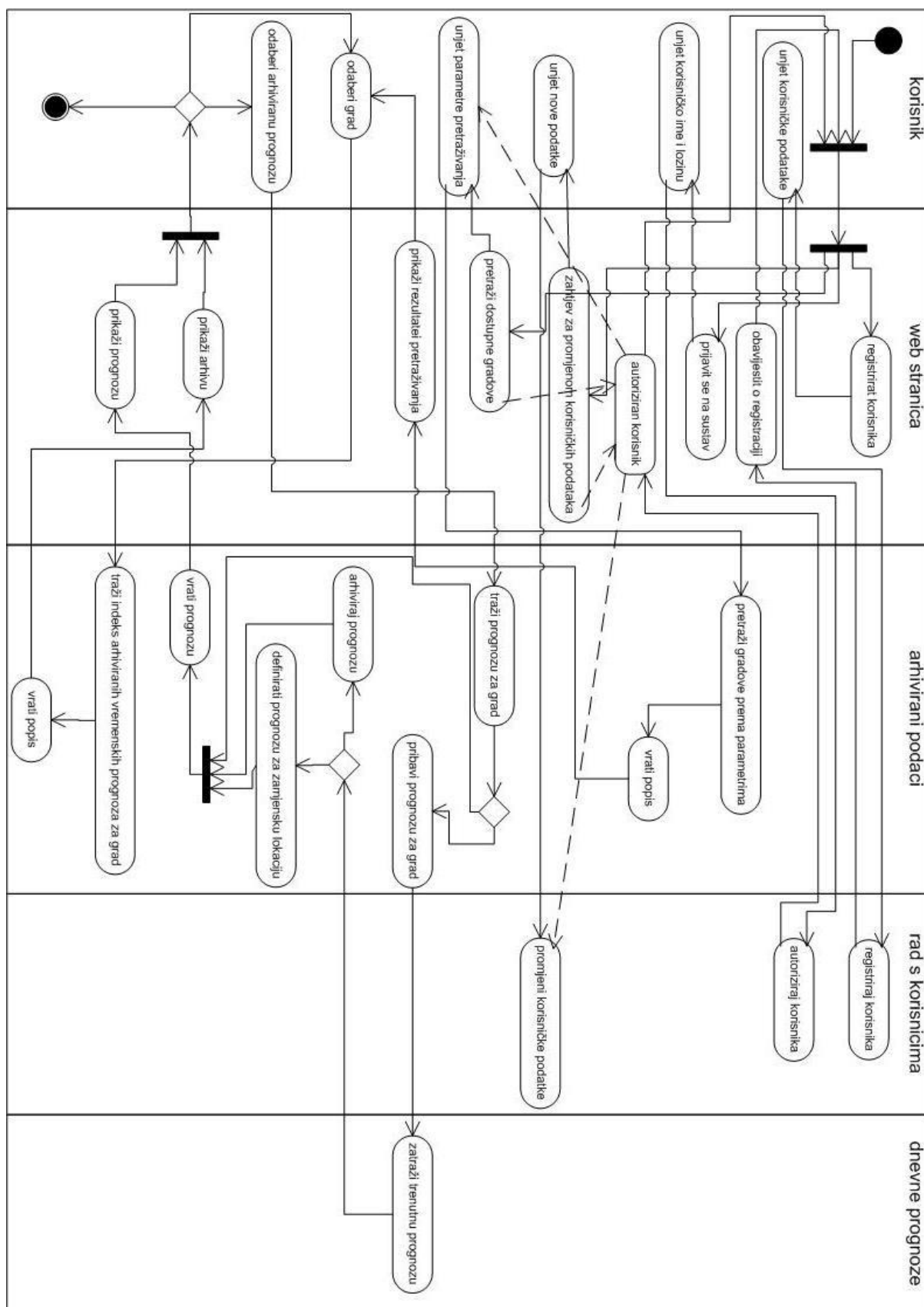
#### 6.4.1.2. Dijagram aktivnosti pristupanja arhiviranim podacima



**Slika 6.4. Prikaz aktivnosti prilikom pristupanja korisnika podacima o vremenskim prognozama**



### 6.4.1.3. Dijagram aktivnosti korištenja web stranice

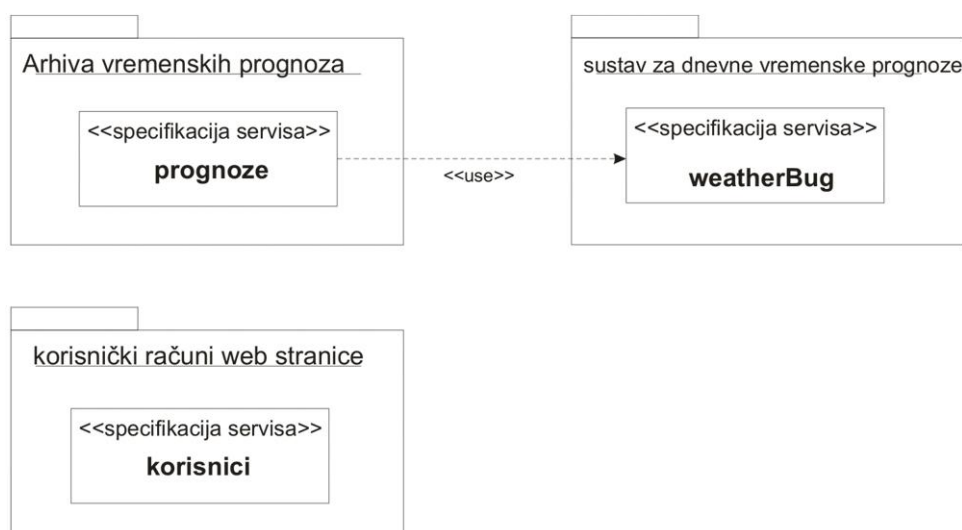


Slika 6.5. Prikaz aktivnosti korisnika prilikom korištenja web stranice

## 6.4.2. Servisna topologija

Iz dijagrama aktivnosti vidimo da koristimo jedan udaljeni servis, weatherBug, koji dobavlja dnevne vremenske prognoze za gradove za koje se vrši arhiviranje prognoze.

Na nama je da odredimo kakav će biti servis kojeg trebamo izraditi. Koji treba davati arhivirane vremenske prognoze i voditi računa o korisnicima web stranice. Kako bi smo izradili servis u skladu s SOA principima, usluge davanja arhiviranih vremenskih prognoza i rada s korisnicima ćemo razdijeliti u dva servisa, kako bi se zadovoljila autonomnost i slaba povezanost servisa. Servis koji će davati podatke o arhiviranim vremenskim prognozama i gradovima, nazvat ćemo „prognoze“, a servis koji će raditi s korisničkim računima korisnika web stranice, zvat će se „korisnici“. Oba servisa su autonomna i ne ovise jedan o drugom, a samim time su i slabo povezani. Web servis za vremenske prognoze je jedino povezan sa udaljenim web servisom za dnevne vremenske prognoze weatherBug, ali ako taj servis prestane raditi, funkcionalnost ovog servisa neće biti umanjena, osim dostupnosti novijih arhiviranih podataka .



Slika 6.6. Servisna topologija

### 6.4.3. Specifikacija web servisa

#### 6.4.3.1. Specifikacija web servisa „prognoze“

Servis za vremenske prognoze uslužuje korisnika informacijama o vremenskim prognozama za grad na određeni datum, ako je prognoza za taj datum arhivirana. Servis sadrži sljedeće korisniku dostupne funkcije, preko sučelja web servisa, koje koristi i web stranica.

<<specifikacija servisa>> prognoze	
+searchCity(in city : string, in country : string, in county : string, in zipCode : string, in hasArchive : bool) : cityData	
+listCityArchive(in zipCode : string) : archiveData	
+cityWeather(in zipCode : String, in date : Date) : weatherData	
+cityInfo(in zip : String) : cityData	

Slika 6.7. Specifikacija servisa „prognoze“

#### Opis funkcija

- searchCity
  - pretraživanje dostupnih gradova prema parametrima
  - funkcija vraća listu s klasom podataka „cityData“ u XML formatu
- listCityArchive
  - zatraživanje popisa arhiviranih vremenskih prognoza za grad sa ZIP kodom
  - funkcija vraća listu s klasom podataka „archiveData“ u XML formatu, s popisom datuma koji su arhivirani
- cityWeather
  - vremenska prognoza za grad prema ZIP kodu na određeni datum
  - funkcija vraća klasu podatka „weatherData“ u XML formatu, s podacima o vremenskoj prognozi.
- cityInfo
  - vraća podatke o gradu s traženim ZIP kodom
  - funkcija vraća klasu podataka „cityData“

## Klase podataka

Servis „prognoze“ koristi klasu podataka „cityData“ za pohranjivanje podataka o gradovima iz baze podataka, koje prosljeđuje sučelju web servisa, koje transformira tu klasu u XML, koji se vraća korisniku prilikom pretraživanja gradova.

Dok klasa „weatherData“, sadrži podatke o vremenskim prognozama, koje se isto transformiraju u XML dokument, kao odgovor korisniku, na upit kojim se zatražuje vremenska prognoza.

Klasa „archiveData“ služi za vraćanje popisa arhiviranih datuma za neki od gradova. Svaka klasa sadrži atribut „error“ koji govori da je prilikom upita došlo do pogreške, a u „errCode“ se pohranjuje kod pogreške, dok se u polje „message“ pohranjuje opis pogreške, ili neka druga informativna poruka od strane servisa.

errCode	Message
1	Error: Neispravan upit web serveru za prognoze.
2	Error: N postoji grad sa traženim ZIP kodom.
3	Error: Pogreška u sql upitu, u funkciji getCityDataUSZipCode.
4	Error: Udaljeni servis je nedostupan!
5	Error: Podaci za traženi grad su nedostupni.
6	Info: N postoje podaci za traženi ZIP kod. Već za prvu bližu lokaciju.
7	Error: Pogreška na serveru, prilikom zatraživanja prognoze za ZIP kod.
8	Error: N postoji vremenska prognoza prema zahtjevanim parametre.
9	Error: Neispravno unesen datum!
10	Error: Neispravno unesen id.
11	Error: Pogreška na servisu.
12	Error: Nepotpuni podaci za upit.

**Tablica 6.1. Poruke o pogreškama koje servis vraća korisniku**

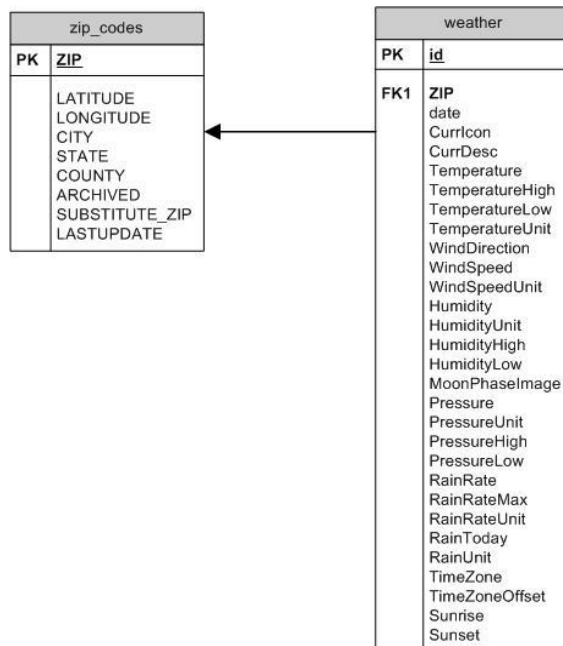
cityData	weatherData	archiveData
+ZIPcode : String	+id : long	+id : long
+city : String	+ZIP : String	+ZIP : String
+latitude : String	+date : Date	+date : Date
+longitude	+CurrIcon : String	+error : bool
+state	+CurrDesc : String	+errorCode : int
+county	+Temperature : float	+message : String
+archived : int	+TemperatureHigh : float	
+substitute_zip	+TemperatureLow : float	
+lastupdate	+TemperatureUnit : String	
+error : bool	+WindDirection : String	
+errorCode : int	+WindSpeed : float	
+message : String	+WindSpeedUnit : String	
	+Humidity : float	
	+HumidityUni : String	
	+HumidityHigh : float	
	+HumidityLow : float	
	+MoonPhaseImage : String	
	+Pressure : float	
	+PressureUnit : String	
	+PressureHigh : float	
	+PressureLow : float	
	+RainRate : float	
	+RainRateMax : float	
	+RainRateUnit : String	
	+RainToday : float	
	+RainUnit : String	
	+TimeZone : String	
	+TimeZoneOffset : int	
	+Sunrise : Date	
	+Sunset : Date	
	+error : bool	
	+errorCode : int	
	+message : String	
	+substitute : bool	
	+substitute_zip : String	

**Slika 6.8. Korištene klase podataka**

## ERA model

Model podataka se sastoji od dvije relacije. U relaciji „zip\_codes“ se nalaze svi gradovi SAD-a s pripadnim ZIP kodovima, nazivom grada, države i okruga, te geografskim koordinatama. U toj relaciji se nalazi i koliko pojedini grad ima arhiva, kad je napravljena zadnja arhiva i zamjenski ZIP kod, ako za traženi kod ne postoji prognoza.

U relaciji „weather“ se nalaze arhivirani podaci o vremenskim prognozama za definirane gradove. U relaciji se nalaze i podaci o datumu i vremenu arhiviranja, za trenutnu vremensku prognozu.



Slika 6.9. ERA model baze podataka za servis „prognoze“

#### 6.4.3.2. Specifikacija web servisa „korisnici“

Servis „korisnici“ koristi se kako bi se registrirali i autorizirali korisnici web stranice i promijenili korisnički podaci.

<<specifikacija servisa>> korisnici
+registerUser(in ime : String, in prezime : String, in korIme : String, in lozinka : String, in email : String) : userData +userAutentification(in korIme : String, in lozinka : String) : userData +changeUserProperties(in userId : String, in lozinka : String) : userData

Slika 6.10. Specifikacija servisa „korisnici“

#### Opis funkcija

- registerUser
  - registracija korisnika s unesenim parametrima
  - kao rezultat vraća klasu podataka „userData“ u XML formatu
- userAutentification
  - autentifikaciju korisnika
  - kao rezultat vraća klasu podataka „userData“ u XML formatu

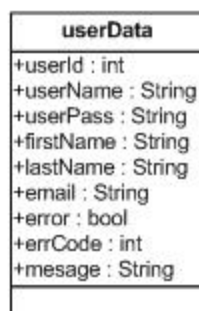
- `changeUserProperties`
  - promjena podataka korisnika (lozinka, e-mail)
  - kao rezultat vraća klasu podataka „`userData`“ u XML formatu

## Klase podataka

Klasa podataka „`userData`“ služi za pohranjivanje podataka o korisniku, koji se ujedno prosljeđuju od web servisa, u kojem se nalaze potrebni odgovori. Ako je autentifikacija fila uspješna, tada se korisniku vraćaju svi podaci o njemu, osim lozinke. U klasi se nalaze i podaci o pogreškama, neke od mogućih se nalaze u sljedećoj tablici.

<b>errCode</b>	<b>Message</b>
1	Error: Neispravno korisnicko ime ili lozinka
2	Error: Pogreška na servisu, prilikom autentifikacije korisnika.
3	Error: Nepostojeći korisnik.
4	Error: Nisu uneseni svi potrebni podaci.
5	Error: Korisničko ime već postoji.
6	Error: Pogreška na servisu prilikom kreiranja korisničkog računa.

**Tablica 6.2. Poruke o pogreškama koje servis vraća korisniku**



**Slika 6.11. Korištena klasa podataka „`userData`“**

## ERA model

Relacijski model sadrži samo jednu relaciju, u kojoj se nalaze osnovni podaci o korisnicima i podaci potrebni za autentifikaciju.

user	
PK	<u>idUser</u>
	ime prezime email korIme lozinka

Slika 6.12. ERA model baze podataka za servis „korisnici“

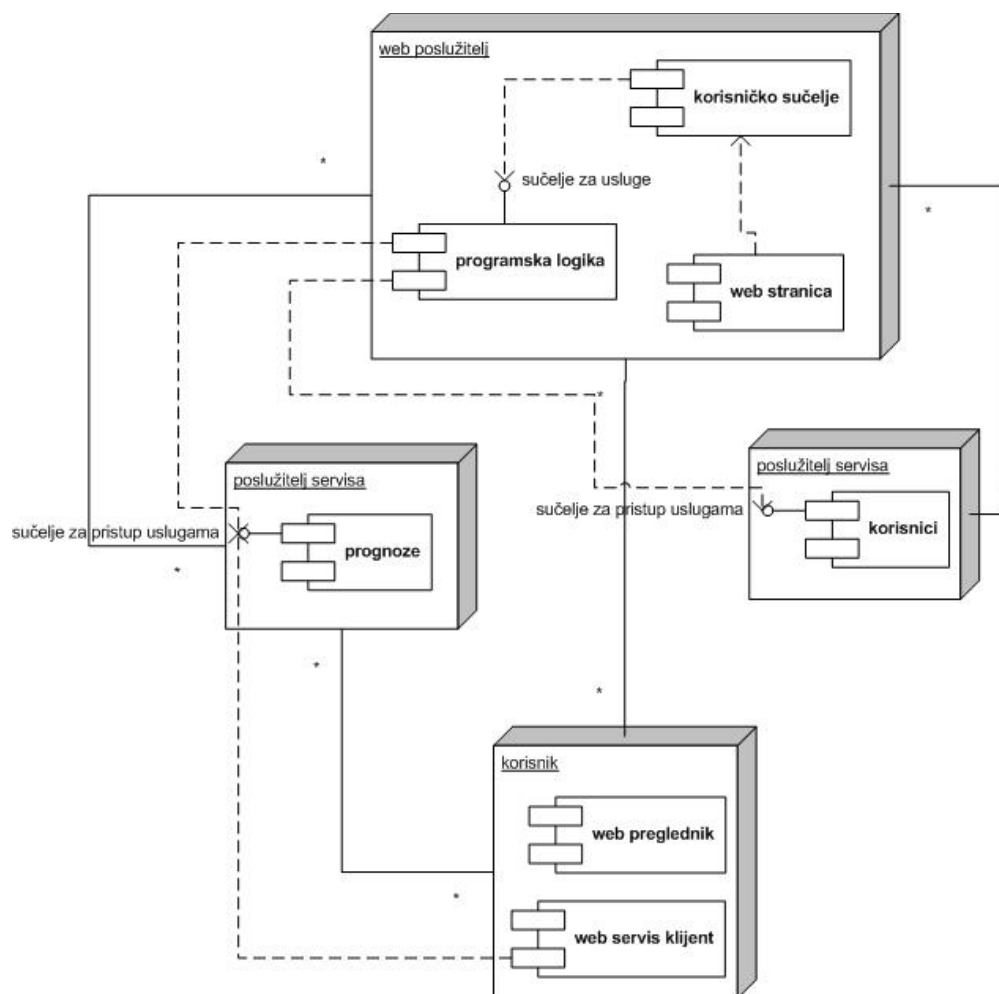
## 6.5. Implementacija

Za implementaciju projekta koristi se Java platforma u razvojnom alatu NetBeans 6.7. Gdje će servisi biti implementirani u obliku web servisa. Koristeći JAX-WS (*Java API for XML Web Services*) tehnologiju za realizaciju web servisa, koji će biti postavljeni zajedno sa web stranicom na Apache Tomcat web server.

Za pohranjivanje i pristupanje podacima koristi se MySQL server, koji sadrži dvije odvojene baze podataka za svaki od servisa. Web servis „prognoze“ koristi bazu podataka pod nazivom „bp\_prognoze“, a web servis „korisnici“ bazu pod nazivom „bp\_korisnici“.



### 6.5.1. Organizacija projekta



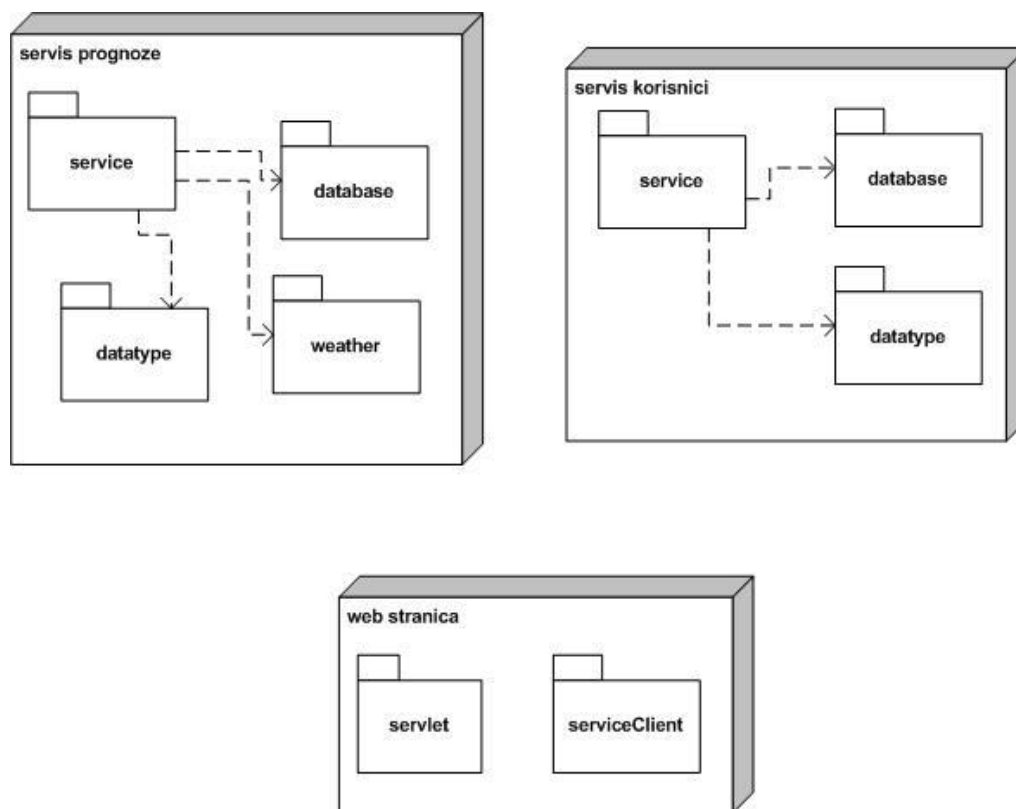
Slika 6.13. Fizički prikaz organizacije projekta

### 6.5.2. Organizacija paketa

Svaki servis je podijeljen na pakete, kako bi se određeni dijelovi projekta mogli nezavisno razvijati. Oba servisa sadrže pakete „datatype“ u kojima se nalaze klase podataka koje pojedini servisi koriste, a u paketima „database“ se nalaze potrebne datoteke za rad s bazom podataka i potrebni SQL upiti prema bazi podataka za svaki od servisa. Svaki od servisa ima paket „service“, u kojem se nalaze funkcije koje predstavljaju sučelja servisa, te se upiti dalje obrađuju koristeći datoteke u preostalim paketima.

Servis prognoze sadrži paket „weather“ u kojem se nalaze datoteke i funkcije potrebne za dohvaćanje vremenskih prognoza s udaljenog web servisa „weather Bug“. U tom paketu se nalazi i datoteka koja služi za pozadinski proces koji arhivira vremenske prognoze.

Web stranica sadrži dva paketa, „servlet“ sadrži datoteke koje obrađuju zahtjeve web stranice. A paket „serviceClient“ sadrži klijente prema servisu „prognoze“ i „korisnici“.



**Slika 6.14. Organizacija paketa za pojedini podsustav projekta**

## 6.6. Postavljanje

Za postavljanje servisa potrebno je imati instaliran Java SE Development Kit (JDK 6, ili noviji), Apache Tomcat 6.xx i MySQL server i Apache ANT. Za korištenje servisa „korisnici“ Apache Tomcat mora posjedovati certifikat, koji se koristi za SSL protokol.

Prije postavljanja web servisa, potrebno je kreirati baze podataka i pripadne relacije koje servisi koriste. Uz svaki servis se nalazi kazalo „bp\_prognoze“ i „bp\_korisnici“ u kojima se nalaze pripadne ANT skripte „build\_database.xml“ za kreiranje korisnika baze podataka i relacija. U svakom kazalu se nalazi i datoteka „options.properties“ u koju upisujemo postavke za spajanje na željenu bazu podataka. Nakon postavljanja baze, prelazi se na proces postavljanja web servisa na Apache Tomcat.

Servisi i web stranica su zapakirani u web archive (\*.war), te svaki od servisa dolazi kao web aplikacija, koju je potrebno postaviti na server. Način na koji se postavlja ovisi o serveru na koji se postavlja. U ovom primjeru korišten je Apache Tomcat. Kod kojeg nakon pokretanja servera, na lokalnoj stranici u web pregledniku (<http://localhost:8084/>), možemo pristupiti sustavu za postavljanje stranice, koji zahtjeva da se unese lokacija na kojoj se nalaze *war* datoteke, koje želimo postaviti na server. Prije postavljanja servisa, ulaskom u \*.war arhivu u kazalu „META-INFO“ se nalaze „\*.properties“ datoteke, u kojim konfiguriramo parametre potrebne za spajanje na baze podataka koje smo prethodno kreirali. Zatim u Apache Manageru postavljamo „\*.war“ datoteke servisa i web stranice, te nakon toga možemo pristupiti web stranici i servisima.

## 6.7. Testiranje

### 6.7.1. Web stranice

Nakon postavljanja web stranice na server, stranici pristupamo lokalno na adresi:  
[http://localhost:8084/dsauer\\_webpage/](http://localhost:8084/dsauer_webpage/).



Slika 6.15. Početni ekran web stranice

Test	Korisnik	Rezultat
Otvaranje forme za vremenske prognoze	Nije prijavljen	Pristup zabranjen
Otvaranje forme za vremenske prognoze pomoću linka prijavljenog korisnika, nakon odjave korisnika	Nije prijavljen	Pristup zabranjen
Otvaranje forme o korisničkim podacima pomoću linka prijavljenog korisnika, nakon odjave korisnika	Nije prijavljen	Pristup zabranjen

Tablica 6.3. Test pristupa web stranici neautoriziranim korisnicima

Da bi korisnik mogao koristiti sustav, potrebno je prvo registrirati se. Za registraciju, prvi kliknemo na link „Prijava „ na naslovnoj stranici, zatim se ispod forme za prijavu nalazi link „Kreirajte novi korisnički račun“.

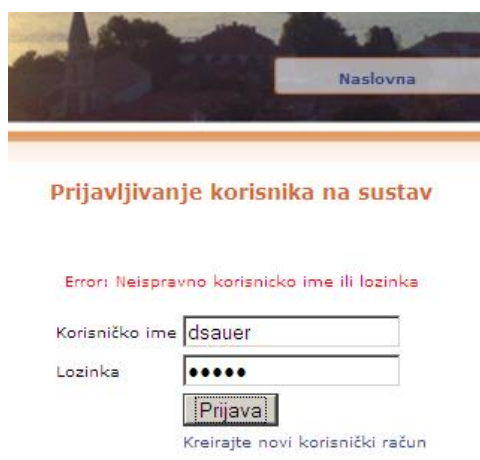


**Slika 6.16. Prikaz forme za registraciju**

Test	Rezultat
Unos postojećih podataka (e-mail i korisničko ime)	Zabrana registracije
Unos postojećeg e-maila	Zabrana registracije
Unos postojećeg korisničkog imena	Zabrana registracije
Ispravan unos podataka	Korisnik uspješno registriran

**Tablica 6.4. Testiranje web servisa „korisnici“ i funkcije za kreiranje korisničkog računa**

Nakon uspješne registracije korisnika, vraćamo se na link „Prijava“, i unosimo potrebne podatke za prijavu na sustav.



**Slika 6.17. Forma za prijavljivanje korisnika na sustav**

Test	Rezultat
Unos nepostojećeg korisnika	Pogreška o neispravnosti
Unos neispravne lozinke ili korisničkog imena	Pogreška o neispravnosti

**Tablica 6.5. Testiranje web servisa „korisnici“ i funkcije za autorizaciju korisnika**

Nakon uspješne autorizacije, možemo pristupiti uslugama pretraživanja vremenskih prognoza, na linku „Prognoza“. Gdje gradove SAD-a možemo pretraživati prema ZIP kodu, ili početnom djelu ZIP koda. Forma omogućuje slaganje upita, unošenjem višestrukih parametara kao što su naziv ili dio naziva grada, okruga, države i sužavanje rezultata samo na gradove koji imaju arhivirane prognoze. Nakon čega se pojavljuje popis gradova prema traženim uvjetima pretraživanja. Prilikom odabira nekog od gradova, otvara se dodatna forma za prikaz „Arhiviranih prognoza“, ako ih traženi grad ima. Arhivirane prognoze predstavljene su vremenskom linijom, gdje horizontalne crtice „-“ predstavljaju arhivirane dane, a ako mišem prelazimo preko njih, tada se iznad prikazuje datum za koji je arhivirana prognoza. Klikom na neku od horizontalnih crtica, učitavamo vremensku prognozu za taj dan. Vertikalne crtice, predstavljaju granice arhiviranja unutar jednog mjeseca. Za gradove koji nemaju vremenske prognoze, prognozu zatražujemo kolikom na gumb „Prikaži najnoviju vremensku prognozu“. Ako ne postoji prognoza za traženu lokaciju, servis će vratiti odgovor o zamjenskoj prvoj bližoj lokaciji.

Test	Rezultat
Unos nepostojećeg ZIP koda	Nema rezultata pretrage
Odabir ZIP koda bez arhive	Dobavlja se trenutna vremenska prognoza
Odabir ZIP koda koji nema prognoze	Ponujen zamjenski ZIP kod

**Tablica 6.6. Testiranje web stranice za pretraživanje gradova i prognoza**

- Vremenske prognoze

Pretraživanje gradova

Pretraživanje prema:

(popuniti minimalno jedan unos)

ZIP kod  Naziv grada  Naziv okruga  Naziv države



Gradovi s arhivama

Pretraži

Stranica

1 2

Država: **Arizona**

Okrug: **PINAL**

BAPCHULE (85221)

Okrug: **MARICOPA**

PHOENIX (85086)

Država: **Nevada**

Okrug: **CLARK**

HENDERSON (89074)

LAS VEGAS (89101)

LAS VEGAS (89103)

LAS VEGAS (89106)

LAS VEGAS (89107)

LAS VEGAS (89108)

LAS VEGAS (89110)

LAS VEGAS (89113)

LAS VEGAS (89117)

LAS VEGAS (89118)

LAS VEGAS (89121)

LAS VEGAS (89122)

LAS VEGAS (89123)

LAS VEGAS (89124)

LAS VEGAS (89129)

LAS VEGAS (89130)

LAS VEGAS (89131)

LAS VEGAS (89134)

- Arhivirane prognoze

Prikaži najnoviju vremensku prognozu

Odobrani datum: 28.8.2009 21:7

godina: 2009

|.....|...

28.8.2009 21:7

**LAS VEGAS**

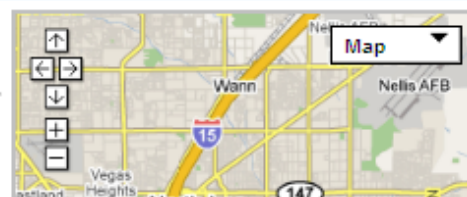
datum: 1.9.2009. 14:51

Pošteni broj: 89101

Država: Nevada

Vremenska prognoza Fair

Temperatura 27.8 °C



Slika 6.18. Pretraživanje, rezultati pretraživanja, prikaz archive i vremenske prognoze iz archive

Zadnja mogućnost koja je korisniku na raspolaganju, je promjena lozinke za autorizaciju. Nakon što se korisnik prijavi na sustav, u glavni izbornik se dodaje link s korisničkim imenom, koji korisnika vodi na prikaz korisničkih podataka.



**Korisnička stranica**

Ime: davor

Prezime: sauer

e-mail: davor.sauer@gmail.com

Korisničko ime: dsauer

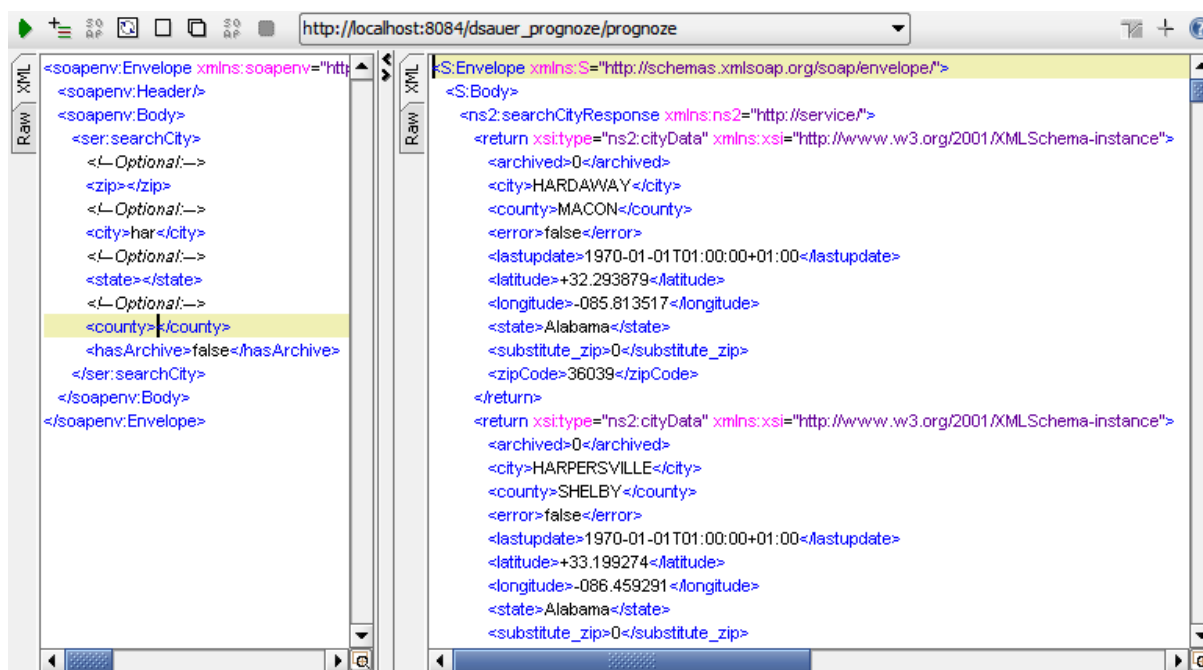
Lozinka: [masked]

Prihvati Odustani

**Slika 6.19. Promjena lozinke za autorizaciju korisnika**

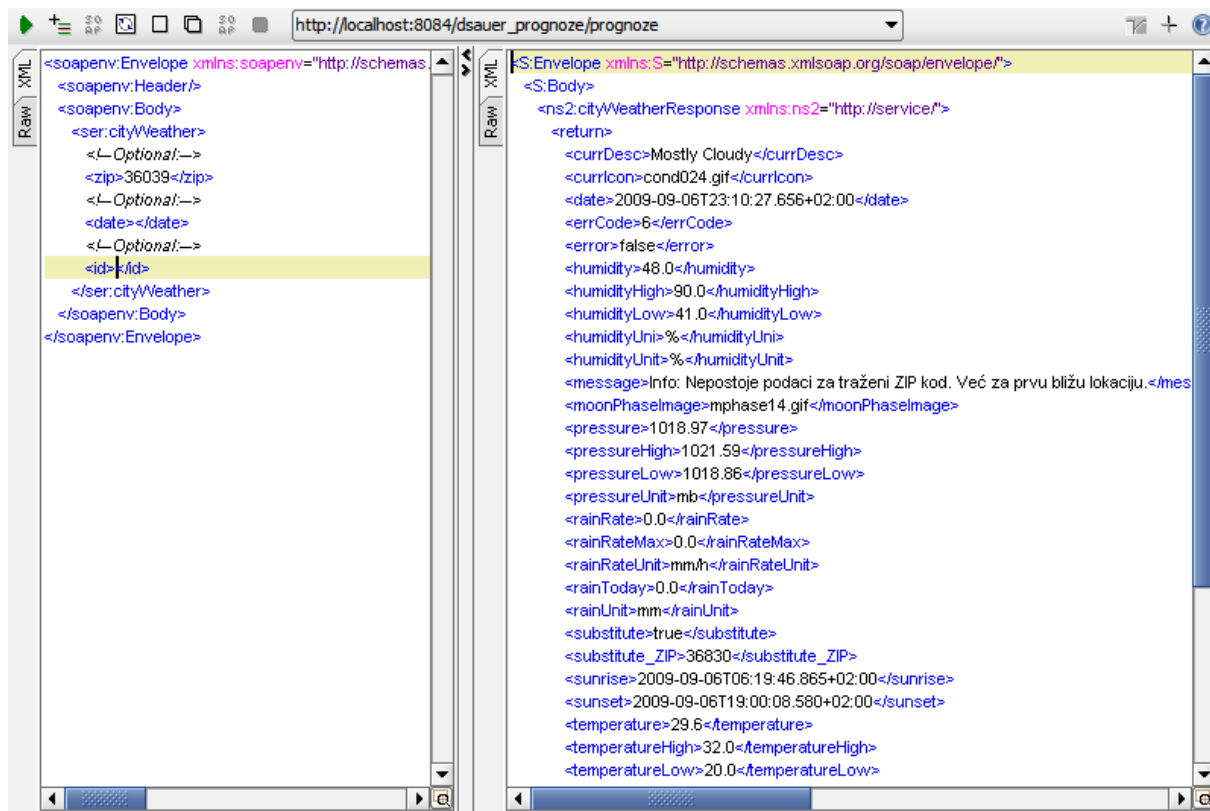
#### 6.7.2. Testiranje servisa „prognoze“ SOAP klijentom

Za testiranje servisa „korisnici“, koristimo se programom *soapUI 3.0*, kojeg možemo pronaći na adresi <http://www.soapui.org/>.

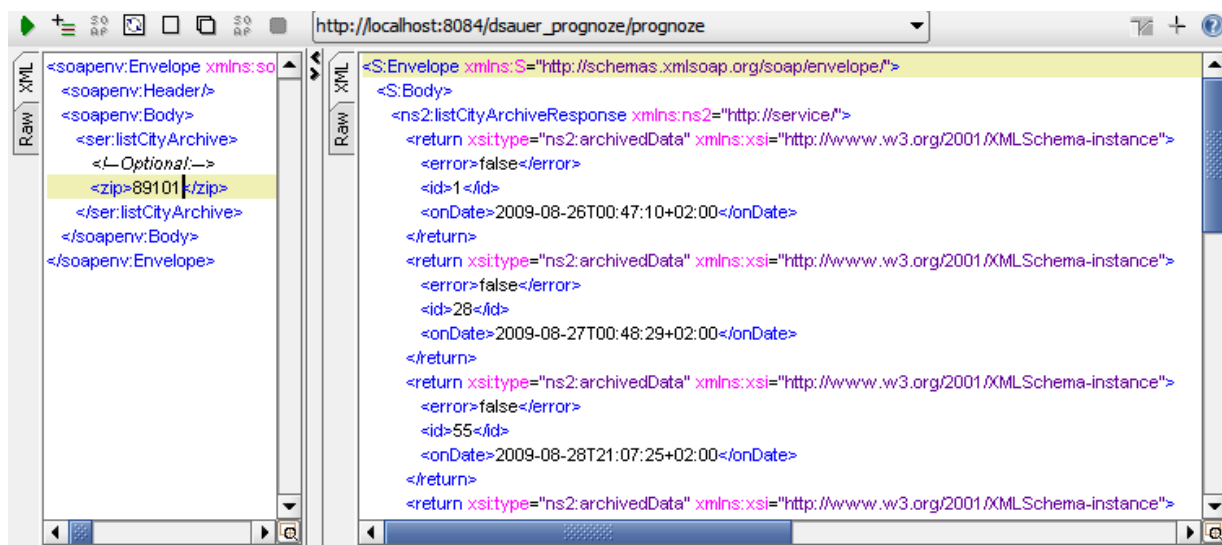


**6.20. Primjer slanja SOAP zahtjeva funkcije „searchCity“ i SOAP odgovor web servisa „prognoze“**

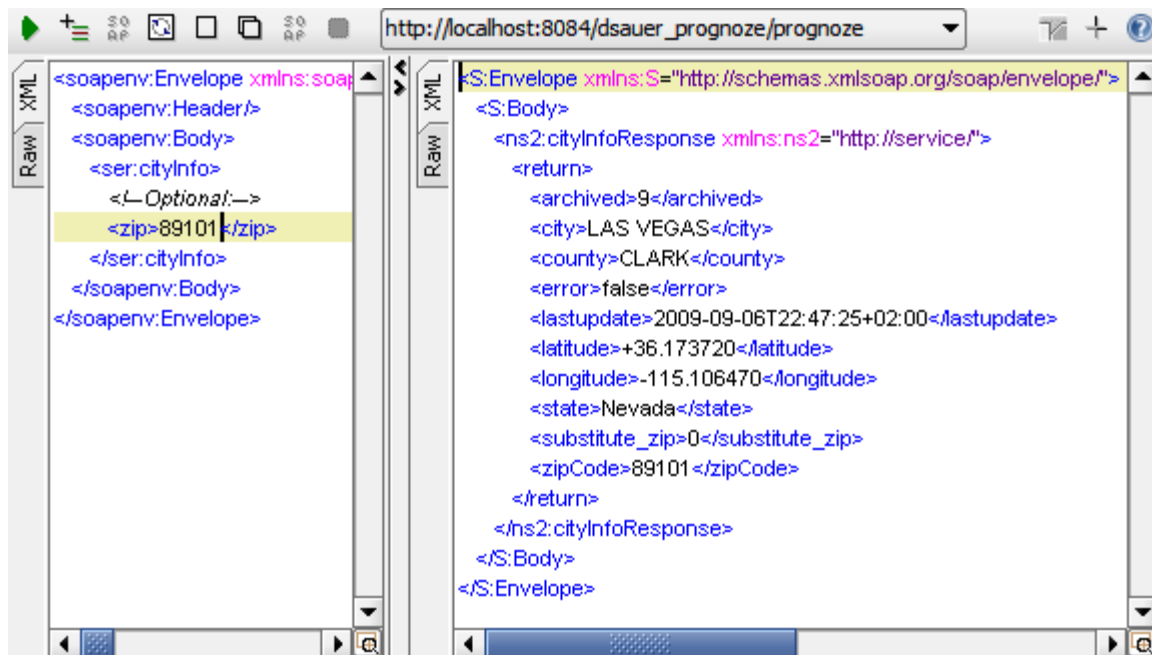




**6.21. Primjer slanja SOAP zahtjeva funkcije „cityWeather“ i  
SOAP odgovor web servisa „prognoze“**



**6.22. Primjer slanja SOAP zahtjeva funkcije „listCityArchive“ i  
SOAP odgovor web servisa „prognoze“**



**6.23. Primjer slanja SOAP zahtjeva funkcije „cityInfo“ i  
SOAP odgovor web servisa „prognoze“**

## 6.8. Upravljanje servisima

Dodatno upravljanje servisima i web stranicom je omogućeno kroz datoteke u kojima se nalaze opcije. U svakoj *.war* datoteci se nalazi kazalu „META-INF“, u kojem se nalaze konfiguracijske datoteke.

Servis „prognoze“ ima konfiguracijsku datoteku „prognoze.properties“. Gdje se mogu konfigurirati parametri koje koristi servis. Može se definirati „host“ na kojem se nalazi baza podataka, naziv baze podataka, korisničko ime i lozinka za pristup bazi podataka, i potrebne datoteke koje se koriste prilikom rada s bazom podataka.

S opcijom „weather.thread=true(/false)“ se kontrolira dali će sustav periodički u pozadini arhivirati podatke, a pod „weather.wsdl“ se upisuje wsdl adresa WeatherBug servisa, i „weather.acode“ se upisuje pristupni kod za weatherBug servis.

Opcija „weather.lastupdate“ definira koliko mora biti stara zadnja arhivirana vremenska prognoza, da bi se zatražila nova prognoza. U ovom slučaju je to 86400 sekundi=24h, što znači da će se ažurirati sve vremenske prognoze starije od 24h. A pomoću opcije „thread.sleep“ se definira koliko učestalo će se provjeravati koji datum je stariji od „weather.lastupdate“.

```

host=127.0.0.1
db.name=bp_prognoze
db.url=jdbc:mysql://${host}/${db.name}
db.user=dsauer
db.pass=diplomski

mysql.lib=lib/mysql-connector-java-5.1.7-bin.jar
db.driver=com.mysql.jdbc.Driver
mysql.driver=./${mysql.lib}

#true/false - skidanje novih podataka s WeatherBug servisa
weather.thread=true
weather.wsdl=http://api.wxbug.net/weatherservice.asmx?WSDL
weather.acode=A5469138662

#(sec) arhiviranje vremenskih prognoza koje su starije od N sekundi (24h=86400 sec)
weather.lastupdate=86400
#(min) ucestalost provjere arhiviranja , kako se udaljeni servis nebi opteretio odjednom
thread.sleep=20

```

### Kod 6.1. Prikaz datoteke prognoze.properties

Servis „korisnici“ ima konfiguracijsku datoteku „korisnici.properties“. Gdje su značenja parametara identična parametrima u datoteci „prognoze.properties“.

```

host=127.0.0.1
db.name=bp_korisnici
db.url=jdbc:mysql://${host}/${db.name}
db.user=dsauer
db.pass=diplomski

mysql.lib=lib/mysql-connector-java-5.1.7-bin.jar
db.driver=com.mysql.jdbc.Driver
mysql.driver=./${mysql.lib}

```

### Kod 6.2. Prikaz datoteke korisnici.properties

Web stranica ima konfiguracijsku datoteku „webpage.properties“, koja sadrži dva parametra s kojima se definira wsdl adresa web servisa, koje stranica koristi.

```

wsdl.prognoze=http://localhost:8084/dsauer_prognoze/prognoze?wsdl
wsdl.korisnici=https://localhost:8443/dsauer_korisnici/korisnici?wsdl

```

### Kod 6.3. Prikaz datoteke webpage.properties

## 6.9. Pregled SOA principa kroz projekt

<i>SOA principi</i>	<i>Web servis</i>	
	<i>prognoze</i>	<i>korisnici</i>
<i>Standardizirani servisni ugovori</i>	Servisni ugovori su standardizirani pomoću WSDL dokumenata, kojeg posjeduje svaki web servis.	
<i>Slabo povezani</i>	Servisi nisu međusobno povezani, te ne ovise jedan o drugome.	
	Servis za pribavljanje informacija o prognozama koristi udaljeni servis, a prestankom rada servis i dalje neometano može raditi i davati pohranjene informacije.  Servis koristi bazu podataka, o kojoj ovisi njegova funkcionalnost, jer mu ona pruža potrebne informacije.	Servis koristi samo bazu podataka, s u kojoj su pohranjeni podaci o korisnicima web stranice.
<i>Apstraktni</i>	Web servisi su apstraktni tako što korisnik može vidjeti jedino informacije koje su objavljene u WSDL dokumentu. Logika koja se nalazi iza sučelja web servisa, kao i platforma na kojoj je realiziran web servis nije vidljiva.	
<i>Višestruko iskoristivi</i>	Servis za prognoze, mogu koristiti i drugi klijenti u svoje vlastite svrhe. Jer servis nije ograničen samo na korištenje od pripadne web stranice.	Servis za rad s korisnicima se može koristiti i za druge radnje prilikom kojih je potrebna autorizacija korisnika koji ujedno koriste tu web stranicu.
<i>Autonomni</i>	Servis je autonoman, zato što	Korisnički web servis je u

	ako servis od kojeg dobavlja vremenske prognoze, prestane s radom, ovaj web servis će nastaviti nesmetano s radom. Ako neke od traženih informacija ne budu dostupne, tada će korisnik biti pravovaljano obaviješten.	potpunosti autonoman, jer njegova funkcionalnost ne ovisi o drugim servisima. Ako ne bude u mogućnosti iznijeti svoju logiku, tada će korisnik biti pravovaljano obaviješten o tome.
<i>Protočni</i>	Web servis za prognoze je slabije protočan, zato što prikuplja informacije od udaljenog servisa. Te ako ne postoje informacije za traženi grad, servis mora kontaktirati udaljeni servis. Tijekom vremena, servis prikuplja sve više i više informacija, te se povećava protočnost, jer će potrebne informacije o gradovima biti pohranjene na bazu podataka servisa.	Web servis za rad s korisnicima ne koristi druge servise, i sve informacije koje on uslužuje realizirane su unutar njega. Tako je njegova protočnost maksimalna.
<i>Samo otkrivljivi</i>	Samo otkrivljivost servisa je omogućena korištenjem servisnog registra, na jednom centralnom mjestu, koje je poznato korisniku. O administratoru sustava ovisi kako će realizirati UDDI registar. Postoje i gotove implementacije UDDI registra, kao što su jUDDI i OpenUDDI.	
<i>Sastavljivi</i>	Servisi ne posjeduju operacije koje rješavaju konkretne probleme, koje bi se nekom drugom kompozicijom mogli rješavati drugi problemi.	

## 7. Zaključak

Koncept servisno orijentirane arhitekture postoji dugi niz godina, ali tehnologija kojom bi se takva arhitektura mogla realizirati nije bila razvijena, ili nije omogućavala u velikoj mjeri realizaciju svih principa koje SOA definira.

Servisno orijentirana arhitektura posjeduje koncepte dizajniranja i koncepte arhitekture. Predstavlja metodologiju, skup principa i načina izgradnje i povezivanja informacijskih sustava.

S današnjom tehnologijom taj cilj je mnogo jednostavniji, a najviše s razvojem Web servisa, koji omogućuju jednostavniji razvoj takve arhitekture. Posjedujući standardizirana sučelja, neovisnost o platformi zbog standardizirane komunikacija, pomoću HTTP protokola. Kao i UDDI registar, koji je standard web servisa, još od samog početka razvoja standarda, koji servisima osigurava samo otkrivljivost. Na taj način web servisi na samom početku pružaju dio podrške koja je potrebna za razvoj servisa ili sustava temeljnih na servisno orijentiranoj arhitekturi.

Dok tijekom procesa specifikacije servisa nastoji primjenjivati principe servisno orijentirane arhitekture, kao što su višestruka iskoristivost, autonomnost, slaba povezanost, protočnost i sastavljivost između različitih programskih i poslovnih sustava. Nastoji se što više iskoristiti postojeće komponente, programski kod i module, te korisnicima omogućiti pristup tim funkcionalnostima i njihovom kombinacijom omogućiti neke nove funkcionalnosti.

SOA je zanimljiva poduzećima zbog njezinih ciljeva, ponajprije zbog smanjenja troškova izrade novih, postojećih sustava i povećanje fleksibilnosti. Isto tako postoje mnogi alati koji pojednostavljuju proces izrade sustava i servisa baziranih na SOA-i, a neke od vodećih firmi sa svojim alatima su IBM, SUN i Microsoft.

Načine na koje možemo samostalno razviti sustav baziran na servisno orijentiranoj arhitekturi je prikazan i kroz ovaj diplomski rad. Gdje se prvo upoznajemo s osnovnim principima servisno orijentirane arhitekture i njezinim ciljevima. Nakon čega se upoznajemo s web servisima, pomoću kojim realiziramo servise, pri čemu logiku i principe koje posjeduju servisi oblikujemo servisno orijentiranim načelima. A kao rezultat imamo jedan manji projekt koji nam prikazuje kako izgleda i funkcionira takav sustav.

Zbog takvih karakteristika, smatram da SOA predstavlja temelj razvoju budućih sustava, a jedan od najboljih načina za realizaciju takvih sustava su web servisi, što je bio i jedan od glavnih razloga pisanja ovog diplomskog, a to je upoznavanje s novim i vodećim tehnologijama, za koje smatram da imaju potencijala u daljnjoj budućnosti. I kao još jedan razlog koji pridodaje

važnosti je što web servisi predstavljaju napredak u web tehnologiji, koja u kombinaciji sa servisno orijentiranom arhitekturom predstavlja razvoj Web-a u smjeru semantičkog Web-a.

## 8. Literatura

1. Erl T. (2009): SOA Design Patterns. Prentice Hall.
2. Erl T. (2008): Web Service Contract Design & Versioning for SOA. Prentice Hall.
3. Erl T. (2007): SOA Principles of Service Design. Prentice Hall.
4. Christudas B.A., Barai M., Caselli V. (2008): Service Oriented Architecture with Java. Pack Publishing Ltd.
5. Brandon D.M.. (2008): Software Engineering for Modern Web Applications: Methodologies and Technologies. IGI Global.
6. Sarang P., Jennings F., Juric M., Loganathan R. (2007): SOA Approach to Integration XML, Web services, ESB, and BPEL in real-world SOA projects. Packt Publishing Ltd.
7. Vohra D., Developing Web Services Using PHP. , Dostupno na: <http://www.onlamp.com/pub/a/php/2007/07/26/php-web-services.html>, učitano: 02.07.2009.
8. Howard R., Web Services with ASP.NET., Dostupno na: <http://msdn.microsoft.com/en-us/library/ms972326.aspx>, učitano: 02.07.2009.
9. Utley C., Creating a .NET Web service., Dostupno na: [http://articles.techrepublic.com.com/5100-10878\\_11-1045209.html](http://articles.techrepublic.com.com/5100-10878_11-1045209.html), učitano: 02.07.2009.
10. Arsanjani A, Zhang L., Ellis M., Allam A., Channabasavaiah K., Design an SOA solution using a reference architecture , Dostupno na: [http://www.ibm.com/developerworks/webservices/library/archtemp/index.html?S\\_TACT=105AGX01&S\\_CMP=LP](http://www.ibm.com/developerworks/webservices/library/archtemp/index.html?S_TACT=105AGX01&S_CMP=LP), učitano: 12.07.2009.
11. Arsanjani A., Service-oriented modeling and architecture, Dostupno na: <http://www.ibm.com/developerworks/library/ws-soa-design1>, učitano: 18.07.2009.
12. Portier B., SOA terminology overview, Part 1: Service, architecture, governance, and business terms, Dostupno na: <http://www.ibm.com/developerworks/webservices/library/ws-soa-term1> , učitano: 18.07.2009.



13. Portier B., SOA terminology overview, Part 2: Development processes, models, and assets, Dostupno na: <[http://www.ibm.com/developerworks/webservices/library/ws-soa-term2/index.html?S\\_TACT=105AGX04&S\\_CMP=EDU](http://www.ibm.com/developerworks/webservices/library/ws-soa-term2/index.html?S_TACT=105AGX04&S_CMP=EDU)> , učitano: 18.07.2009.
14. Portier B., SOA terminology overview, Part 3: Analysis and design, Dostupno na: <[http://www.ibm.com/developerworks/webservices/library/ws-soa-term3/index.html?S\\_TACT=105AGX04&S\\_CMP=EDU](http://www.ibm.com/developerworks/webservices/library/ws-soa-term3/index.html?S_TACT=105AGX04&S_CMP=EDU)> , učitano: 18.07.2009.
15. Amsden J., Modeling SOA: Part 1. Service identification, Dostupno na: <[http://www.ibm.com/developerworks/rational/library/07/1002\\_amsden/index.html?S\\_TACT=105AGX15&S\\_CMP=ART](http://www.ibm.com/developerworks/rational/library/07/1002_amsden/index.html?S_TACT=105AGX15&S_CMP=ART)>, učitano: 18.07.2007.
16. Amsden J., Modeling SOA: Part 2. Service specification, Dostupno na: <[http://www.ibm.com/developerworks/rational/library/07/1009\\_amsden/index.html?S\\_TACT=105AGX15&S\\_CMP=ART](http://www.ibm.com/developerworks/rational/library/07/1009_amsden/index.html?S_TACT=105AGX15&S_CMP=ART)>, učitano: 18.07.2007.
17. Amsden J., Modeling SOA: Part 3. Service realization, Dostupno na: <[http://www.ibm.com/developerworks/rational/library/07/1016\\_amsden/index.html?S\\_TACT=105AGX15&S\\_CMP=ART](http://www.ibm.com/developerworks/rational/library/07/1016_amsden/index.html?S_TACT=105AGX15&S_CMP=ART)>, učitano: 18.07.2007.
18. Amsden J., Modeling SOA: Part 4. Service composition, Dostupno na: <[http://www.ibm.com/developerworks/rational/library/07/1023\\_amsden/index.html?S\\_TACT=105AGX15&S\\_CMP=ART](http://www.ibm.com/developerworks/rational/library/07/1023_amsden/index.html?S_TACT=105AGX15&S_CMP=ART)>, učitano: 18.07.2007.
19. Amsden J., Modeling SOA: Part 5. Service implementation, Dostupno na: <[http://www.ibm.com/developerworks/rational/library/07/1030\\_amsden/index.html?S\\_TACT=105AGX15&S\\_CMP=ART](http://www.ibm.com/developerworks/rational/library/07/1030_amsden/index.html?S_TACT=105AGX15&S_CMP=ART)>, učitano: 18.07.2007.
20. Jewell T., Chappell D., UDDI: Universal Description, Discovery, and Integration , Dostupno na: <<http://oreilly.com/catalog/javawebsevr/chapter/ch06.html>>, učitano: 29.07.2009.
21. \*\*\*, Web Services Life Cycle: Managing Enterprise Web Services, Dostupno na: <[http://www.sun.com/software/whitepapers/webservices/wp\\_mngwebsvcs.pdf](http://www.sun.com/software/whitepapers/webservices/wp_mngwebsvcs.pdf)>, učitano: 01.08.2009.

# Prilog

## Instalacijski DVD

Na DVD-u se nalazi bootabilna Live Linux distribucija, koju se umetne u DVD-ROM prilikom pokretanja računala. Linux sustav se ne instalira, već se pokreće sa DVD-a, prilikom čega se inicijaliziraju potrebni servisi za rad projektnog primjera. Nakon što se Linux sustav pokrene u grafičkom sučelju, otvara se i web preglednik, pomoću kojeg možemo lokalno pristupiti web servisima i web stranici.

Na DVD-u u kazalu „Diplomski“ se nalazi izvorni kod web servisa i web stranice, i datoteke potrebne za instalaciju. A u kazalu „Programi“ se nalaze potrebni programi za postavljanje web servisa i web stranice na Linux ili Windows sustav.