Razvoj aplikacija pomoću webservisa

Arzon, Lana

Master's thesis / Diplomski rad

2014

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:904318

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Lana Arzon

RAZVOJ APLIKACIJA POMOĆU WEB SERVISA

Diplomski rad

Voditelj rada:
prof. dr. sc. Robert Manger

Zagreb, rujan 2014.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Sadržaj **iii**

Uvod **1**

1 Ponovna upotreba softvera **2**

2 Arhitektura usmjerena na servise **5**

3 Standardi vezani uz web servise **8**
3.1 SOAP 10
3.2 WSDL 12
3.3 WS-BPEL 13
3.4 UDDI 14

4 RESTful web servisi **16**
4.1 REST 16

5 Softverski procesi vezani uz web servise **19**
5.1 Razvoj web servisa 19
5.2 Razvoj aplikacija korištenjem web servisa 22

6 Java web servisi **28**
6.1 Izrada servisa 28
6.2 Testiranje servisa 33
6.3 Korištenje servisa 34

7 Zaključak **38**

Bibliografija **39**

Uvod

Pojava web-a u 1990-tim godinama donijela je revoluciju u razmjeni podataka preko Interneta. Međutim, podaci i usluge s web-a isprva su bili dostupni samo web preglednicima što nije uvijek bilo praktično. Kao rješenje ovog problema pojavljuju se web servisi. Oni predstavljaju nadogradnju klasične web tehnologije. Korištenjem web servisa sadržaji s web-a postaju dostupni i drugim programima. Preciznije, web servis je resurs na web-u koji je prikazan na standardizirani način i koji se može koristiti iz nekog drugog programa. To može biti podatkovni resurs, na primjer katalog proizvoda, ili računalni resurs, na primjer pretvarač slika iz jednog grafičkog formata u drugi, ili kombinacija jednog i drugog.

Struktura diplomskog rada

Poglavlje 1 opisuje ponovnu upotrebu softvera, njezin razvoj i dvije podjele: s obzirom na veličinu i s obzirom na prirodu onoga što se upotrebljava. Također su nabrojene i opisane neke tehnike ponovne upotrebe među kojima je i razvoj zasnovan na web servisima.

Poglavlje 2 posvećeno je novoj vrsti arhitekture distribuiranih sustava koju omogućuju web servisi. Riječ je o arhitekturi usmjerenoj na servise (SOA). Ukratko je opisana ideja SOA te su navedeni neki od osnovnih principa na kojima se ona zasniva.

Poglavlje 3 govori o standardima koji prate razvoj web servisa. Spominje se jezik XML na kojem se zasnivaju svi takvi standardi. Detaljnije su obrađeni glavni standardi za web servise: SOAP, WSDL, WS-BPEL i UDDI.

Poglavlje 4 spominje RESTful web servise kao alternativu standardnim web servisima. Saznajemo što je REST i koja su glavna načela REST modela.

Poglavlje 5 bavi se softverskim procesima vezanim uz web servise. Postoje dvije vrste takvih procesa: razvoj samih web servisa koje će netko drugi upotrebljavati i razvoj aplikacija korištenjem postojećih web servisa. Nabrojene su i opisane faze obiju vrsta procesa.

Poglavlje 6 objašnjava zašto je programski jezik Java pogodan za implementaciju web servisa. Opisan je postupak izrade Java web servisa za pretvaranje mjernih jedinica temperature, točnije za pretvaranje Celzijevih stupnjeva u Fahrenheitove i obratno. Napravljene su i dvije klijentske aplikacije koji prikazuju način na koji se koristi spomenuti web servis.

Poglavlje 1

Ponovna upotreba softvera

Web servisi jedan su od oblika ponovne upotrebe softvera. Ponovna upotreba bavi se pitanjem kako već razvijeni softver ponovo upotrijebiti za neku drugu svrhu ili za neke druge korisnike. Ideja ponovne upotrebe softvera prisutna je više od 40 godina, ali se tek početkom 21. stoljeća, kada su bile razvijene odgovarajuće tehnike, počela intenzivnije primjenjivati. Ispostavilo se da je ponovna upotreba odgovor za smanjenje količine softvera kojeg stvarno treba razviti te tako uzrokuje smanjenje troškova održavanja, bržu isporuku sustava i povećanje kvalitete softvera. Naime, smatra se da su prethodno razvijeni dijelovi softvera koji će se ponovno upotrijebiti pouzdani i dobro testirani pa se na njih stavlja oslonac.

Dijelovi softvera koji će se ponovno upotrijebiti mogu biti različitih veličina pa prema tome razlikujemo tri vrste ponovne upotrebe:

1. *Ponovna upotreba cijelog sustava.* Postojeći sustav se nakon odgovarajuće konfiguracije i prilagodbe daje na upotrebu novim korisnicima ili se ugradnjom manjih promjena stvara nova verzija postojećeg sustava koja je prilagođena potrebama novih korisnika.

2. *Ponovna upotreba dijela sustava.* Dio postojećeg sustava srednje veličine ugrađuje se u novi sustav. Također može biti riječ o upotrebi komponente koja je bila razvijena zato da bi se ponovno upotrebljavala ili o korištenju web servisa opće namjene.

3. *Ponovna upotreba funkcije ili klase.* Manji dio postojećeg sustava, na primjer jedna funkcija ili klasa, ugrađuje se u novi sustav. Također može biti riječ o korištenju funkcije ili klase iz biblioteke opće namjene.

Osim s obzirom na veličinu dijelova, ponovnu upotrebu možemo podijeliti i s obzirom na prirodu onoga što se upotrebljava i to na dvije vrste:

1. *Ponovna upotreba programskog koda ili izvršivog programa.* Već napisani softver uključuje se u novi sustav. Na primjer, može biti riječ o korištenju biblioteke, nasljeđivanju klase, uključivanju komponente, pozivu web servisa.

2. *Ponovna upotreba modela, koncepta ili dizajnerskog rješenja.* Na primjer, koristi se poznati obrazac za oblikovanje, no na osnovi njega piše se vlastiti programski kod. Također se u skladu s modelom grafičkog razvoja može koristiti već postojeći UML[1] dijagram iz kojeg se automatski generira programski kod.

Danas su već razvijene i usavršene mnoge tehnike za ponovnu upotrebu. Takve tehnike koriste činjenicu da su sustavi s istom aplikacijskom domenom slični pa imaju potencijal za ponovnu upotrebu i to na različitim razinama, od ponovne upotrebe jednostavnih funkcija do ponovne upotrebe cijelih aplikacija. Također koriste i činjenicu da standardi za ponovno upotrebljive dijelove znatno olakšavaju ponovnu upotrebu. Slijedi popis nekih od tehnika koje su se s više ili manje uspjeha koristile kao oblik ponovne upotrebe.

Obrasci za oblikovanje. Riječ je o ponovnoj upotrebi provjerenih generičkih rješenja za probleme oblikovanja koji se često pojavljuju u raznim kontekstima.

Aplikacijski okviri. Riječ je o korištenju skupa apstraktnih ili konkretnih klasa koje treba nadograditi ili proširiti da bi se od njih dobio željeni aplikacijski sustav.

Produktne linije. Mijenjanjem polazne generičke aplikacije nastaje niz sličnih konkretnih aplikacija sa zajedničkom arhitekturom. Svaka aplikacija u nizu prilagođena je određenoj grupi korisnika.

Ponovna upotreba COTS (*Commercial Off-the-Shelf*) produkata. Potrebe korisnika rješavaju se korištenjem pogodno konfiguriranih postojećih javno dostupnih softverskih paketa ili njihovim integriranjem.

Razvoj zasnovan na komponentama. Sustav se razvija integriranjem gotovih komponenti, dakle integriranjem dijelova koji su razvijeni baš za ponovnu upotrebu u skladu s odgovarajućim standardima za komponente.

Razvoj zasnovan na web servisima. Sustav se razvija povezivanjem servisa koji su javno dostupni na web-u i s kojima se može komunicirati preko odgovarajućih protokola.

Ponovna upotreba baštinjenih (*legacy*) sustava. Korištenjem odgovarajućih *wrapper*-a uspostavlja se sučelje prema starom baštinjenom sustavu te se na taj način stari sustav ponovo koristi u suvremenom okruženju.

[1] *Unified Modeling Language* – UML je grafički jezik za modeliranje u području softverskog inženjerstva koji je danas prihvaćen kao standard.

Korištenje standardnih biblioteka. U novom sustavu koriste se funkcije ili klase iz neke biblioteke, na primjer koristi se matematička funkcija, ili klasa koja enkapsulira komunikaciju preko mreže, ili klasa za implementaciju rada sa stogom, i tako dalje. Ova tehnika zapravo je najstariji oblik ponovne upotrebe i postoji već 40-tak godina.

Grafički razvoj softvera (*model-driven engineering*). Softver se opisuje grafičkim modelom, na primjer skupom UML dijagrama. Programski kod automatski se generira iz tog modela. Ovdje je riječ o ponovnoj upotrebi znanja kako se pojedini element grafičkog modela treba pretvoriti u programski kod.

Korištenje generatora programa. U nekim usko specijaliziranim domenama postoje alati koji iz specifikacije problema automatski generiraju odgovarajući program. Na primjer, softveri za baze podataka za zadanu SQL[2] tablicu mogu generirati aplikaciju koja omogućuje ažuriranje i pretraživanje te tablice. Riječ je o ponovnom korištenju znanja o domeni i programskih rješenja koja su ugrađena u alat.

Razvoj softvera zasnovan na komponentama i razvoj zasnovan na web servisima dvije su tehnike koje predstavljaju današnji *state-of-the-art* ponovne upotrebe. Riječ je o tehnikama koje su u konceptualnom pogledu vrlo slične, no bitno se razlikuju u pogledu tehničke realizacije. Obje tehnike zasnivaju se na konceptu pružanja usluga i koristimo ih preko (ponuđenog) sučelja koje se sastoji od operacija s parametrima. U oba slučaja aplikaciju gradimo na sličan način – kombiniranjem raspoloživih usluga, pretvorbom rezultata jedne usluge u ulaz za drugu uslugu, i tako dalje. Ove tehnike nagovještavaju nastanak nove softverske industrije koja bi se umjesto razvojem aplikacija bavila isključivo razvojem ponovno upotrebljivih dijelova za aplikacije. Najveća razlika između ovih tehnika je to što web servis funkcionira u svojem okruženju neovisno o aplikaciji pa mu nije potrebno traženo sučelje koje kod komponenti opisuje uvjete koje aplikacija treba pružiti da bi komponenta mogla raditi. Ako više aplikacija koristi istu komponentu, tada svaka od njih pokreće svoju kopiju te komponente. Ako više aplikacija koristi isti web servis, tada sve one dobivaju usluge od istog primjerka servisa. Komponenta može prelaziti iz stanja u stanje te pamtiti svoje trenutno stanje. To znači da se komponenta kod svakog uzastopnog poziva može drukčije ponašati. Budući da isti primjerak web servisa može istovremeno služiti raznim aplikacijama, on nema stanja pa se kod uzastopnih poziva ponaša na potpuno isti način. Zahvaljujući ovim i drugim razlikama, web servisi mogu se pokazati boljim rješenjem od komponenti u jednoj situaciji no lošijim rješenjem u nekoj drugoj situaciji. Za razliku od komponenti koje se uglavnom koriste interno unutar velikih kompanija, web servisi su u većoj mjeri ostvarili ideju globalne softverske industrije gdje se rješenje stvara na jednom mjestu, a ponovo se upotrebljava bilo gdje drugdje u svijetu.

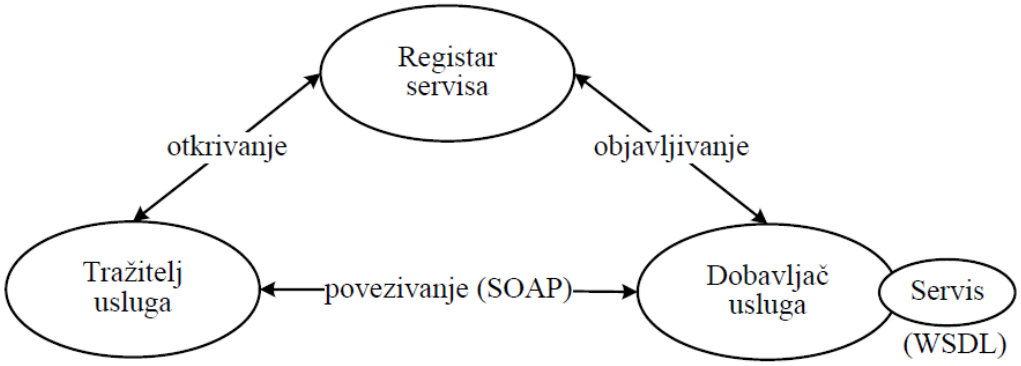
[2] *Structured Query Language* – SQL je najpopularniji računalni jezik za izradu, traženje, ažuriranje i brisanje podataka iz relacijskih baza podataka.

Poglavlje 2

Arhitektura usmjerena na servise

Web servisi temelj su nove vrste arhitekture distribuiranih sustava koja se naziva arhitektura usmjerena na servise ili servisno orijentirana arhitektura (*Service Oriented Architecture – SOA*). Prema SOA, aplikacija se gradi povezivanjem servisa koji su dostupni na globalnoj mreži Internet i izvršavaju se na geografski udaljenim računalima. Preciznije, SOA je arhitektura u kojoj samostalni, međusobno slabo povezani servisi (usluge) dobro definiranih sučelja pružaju poslovnu funkcionalnost koja može biti na određeni način otkrivena i kojoj se može pristupiti kroz odgovarajuću infrastrukturu. Servis je zapravo aplikacija izložena preko standardiziranog sučelja te na taj način dostupna i razumljiva ostalim sustavima u okruženju. SOA omogućuje unutarnju i vanjsku integraciju, kao i ponovnu iskoristivost aplikacijske logike kroz kompoziciju i rekompoziciju servisa. Aplikacije se mogu izgrađivati povezivanjem jednog ili više servisa bez znanja o njihovoj konkretnoj implementaciji.

Ideja SOA prikazana je na slici 2.1. Tri su glavna sudionika SOA: dobavljač usluga, tražitelj usluga i registar servisa. Dobavljači usluga oblikuju i implementiraju servise te definiraju sučelja preko kojih se pristupa tim servisima. Također, oni objavljuju informacije o svojim servisima u nekom registru. Tražitelji usluga otkrivaju u registru specifikacije servisa koji ih zanimaju i lociraju dobavljače usluga. Nakon toga oni mogu povezati svoju aplikaciju s odabranim servisom i komunicirati s njime preko njegovog sučelja razmjenom poruka u skladu s predviđenim protokolom. SOA se razlikuje od standardnog klijent-poslužitelj modela u svom isticanju principa modularnosti i niske međuovisnosti servisa te standardizaciji i upravljanju vezama između servisa i njihovih korisnika.



Slika 2.1: SOA – arhitektura usmjerena na servise.

Iako ne postoji službena SOA specifikacija, možemo navesti nekoliko osnovnih principa na kojima se ona zasniva:

- Opis servisa prilagođen je dogovorenim standardima komunikacije, dostupan je korisnicima servisa te sadrži opis ulaznih i izlaznih parametara i podataka.
- Veza između servisa takva je da minimizira njihovu međusobnu ovisnost te zahtijeva samo da oni znaju jedan za drugoga (*low coupling*).
- Korisnici servisa nemaju pristup detaljima o načinu implementacije servisa.
- Aplikacijska logika podijeljena je na servise s namjerom promicanja ponovne upotrebe.
- Servisi imaju kontrolu nad logikom koju enkapsuliraju (*autonomy*).
- Servisi su *stateless*, to jest ne pamte stanje između dva poziva te tako minimiziraju potrošnju resursa.
- Servisi su nadopunjeni komunikacijskim metapodacima[1] koji omogućuju učinkovito pronalaženje i interpretaciju servisa (*discoverability*).
- Servisi se mogu efektivno upotrebljavati kao dijelovi kompozicije, bez obzira na veličinu i složenost kompozicije (*composability*).

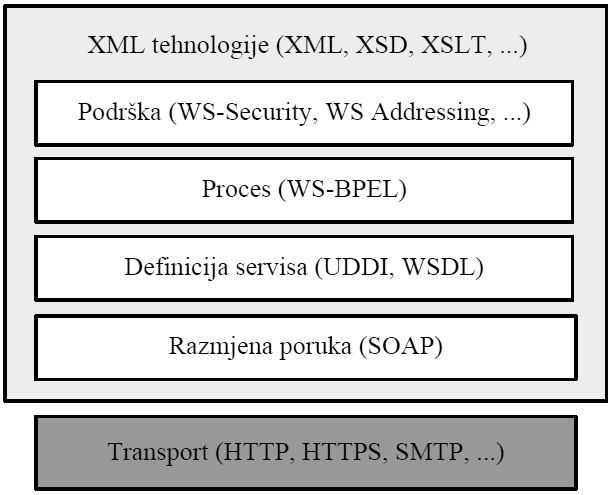
[1]Metapodaci su "podaci o podacima". Korisni su kod pregledavanja, prijenosa i dokumentiranja informacijskog sadržaja. To su strukturirani podaci koji opisuju, objašnjavaju, lociraju ili na neki drugi način omogućuju lakše upravljanje resursima.

Servisno orijentirana arhitektura pojednostavljuje ponovnu iskoristivost elemenata informatičkog sustava. Pomaže u povećanju brzine i produktivnosti razvoja jer su elementi jednom izloženi preko servisa dostupni drugim aplikacijama na korištenje. Proširivanje sustava postaje jednostavnije jer se sastoji većinom od dodavanja novih servisa. Velika modularnost u SOA omogućuje jednostavniju promjenu funkcionalnosti ako je to potrebno. Također, održavanje sustava sa servisno orijentiranom arhitekturom relativno je lako.

Poglavlje 3

Standardi vezani uz web servise

Razvoj web servisa bio je od početka popraćen razvojem odgovarajućih standarda. Sve važne hardverske i softverske kompanije prihvatile su te standarde osiguravajući tako kompatibilnost svojih rješenja. Problem nekompatibilnosti uobičajena je pojava pri razvoju i korištenju novih tehnologija koji je ovako ranom standardizacijom vješto izbjegnut. Slika 3.1 prikazuje jezike i protokole koji su postali ključni standardi za funkcioniranje web servisa. Oni pokrivaju sve aspekte SOA arhitekture, od osnovnih mehanizama za razmjenu informacija (SOAP) do programskih jezika za implementaciju aplikacija koje koriste web servise (WS-BPEL).



Slika 3.1: Standardi za web servise.

Svi ovi standardi za web servise zasnivaju se na jeziku koji se zove XML (*EXtensible Markup Language*). XML je jezik za označavanje podataka koji je stvoren da bude jednostavno čitljiv i ljudima i računalima. Princip realizacije vrlo je jednostavan: odgovarajući sadržaj treba se uokviriti odgovarajućim oznakama koje ga opisuju i imaju poznato ili lako shvatljivo značenje (slika 3.2).

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
<to>Alice</to>
<from>Bob</from>
<heading>Hello</heading>
<body>Hello Alice!</body>
</message>
```

Slika 3.2: Primjer jednostavnog XML dokumenta.

Ukratko, kao glavni standardi za web servise, koje ćemo detaljnije opisati u nastavku, usvojeni su sljedeći protokol i jezici:

1. *Service Oriented Architecture Protocol – SOAP*,
2. *Web Service Definition Language – WSDL*,
3. *Web Service Business Process Execution Language – WS-BPEL*.

Uz spomenuta tri standarda SOAP, WSDL i WS-BPEL za glavni standard predlagao se i *Universal Description, Discovery and Integration – UDDI*, ali nije bio općenito prihvaćen.

Osim glavnih standarda za web servise postoji i niz pomoćnih standarda koji se odnose na neke posebne aspekte. Neki od njih su sljedeći:

1. *WS-Reliable Messaging*. Standard za razmjenu poruka koji osigurava da svaka poruka bude dostavljena točno jednom.
2. *WS-Security*. Skup standarda za sigurnost web servisa koji sadrži standarde za specifikaciju definicije sigurnosnih pravila i standarde za korištenje digitalnog potpisa.
3. *WS-Addressing*. Standard koji definira reprezentaciju informacije o adresi u SOAP porukama.
4. *WS-Transactions*. Standard koji definira način odvijanja transakcija u distribuiranim servisima.

3.1 SOAP

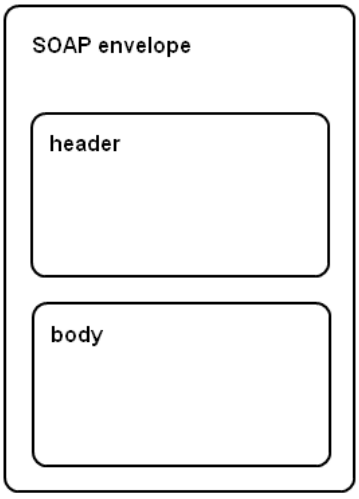
Service Oriented Architecture Protocol – SOAP, izvorno definiran kao *Simple Object Access Protocol*, je protokol za razmjenu poruka između aplikacija i web servisa. Baziran je na XML-u te za razmjenu informacija koristi protokole nižeg sloja, najčešće HTTP (*Hypertext Transfer Protocol*). Razvijen je kako bi se omogućila jednostavna komunikaciju tekstualnim sadržajem preko HTTP komunikacijskog protokola koji je prilagođen upravo razmjeni tekstualnih sadržaja. Ovaj protokol neovisan je o programskom jeziku i platformi te je jednostavno proširiv.

SOAP definira standardni oblik poruka, to jest obavezne i opcionalne dijelove. Glavni elementi SOAP poruke (slika 3.3) su:

Omotnica (*envelope*). Obavezan element koja identificira XML dokument kao SOAP poruku. Definira početak i kraj SOAP poruke.

Zaglavlje (*header*). Nije obavezan element, a može sadržavati jedan ili više blokova metapodataka o samoj poruci. U zaglavlju se nalaze informacije o primjeni poruke, identifikatoru prijenosa te podaci o pošiljatelju i primatelju poruke.

Tijelo (*body*). Obavezan element koji sadrži podatke o pozivu i odgovoru te poruke o greškama koje su se dogodile tijekom obrade poruka. Dio s porukama o greškama (*fault*) nije obavezan.



Slika 3.3: Struktura SOAP poruke.

Opišimo sada princip rada SOAP protokola. SOAP klijent kreira XML dokument koji sadrži odgovarajući zahtjev (*request*). Taj dokument formatiran je u skladu sa SOAP specifikacijom. Dokument dolazi do SOAP poslužitelja koji obrađuje pristigle zahtjeve i na osnovi pristiglih zahtjeva pokreće odgovarajući servis. Po završenoj obradi SOAP poslužitelj, korištenjem SOAP protokola vraća poruku odgovora (*response*) SOAP klijentu. Na slici 3.4 prikazan je primjer poruke zahtjeva kojom se od servisa zahtjeva pozdrav za ime "Alice", dok slika 3.5 prikazuje primjer odgovarajuće poruke odgovora kojom servis vraća pozdrav "Hello Alice!"

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:hello xmlns:ns2="http://service.time.org/">
            <name>Alice</name>
        </ns2:hello>
    </S:Body>
</S:Envelope>
```

Slika 3.4: Primjer SOAP poruke – SOAP zahtjev.

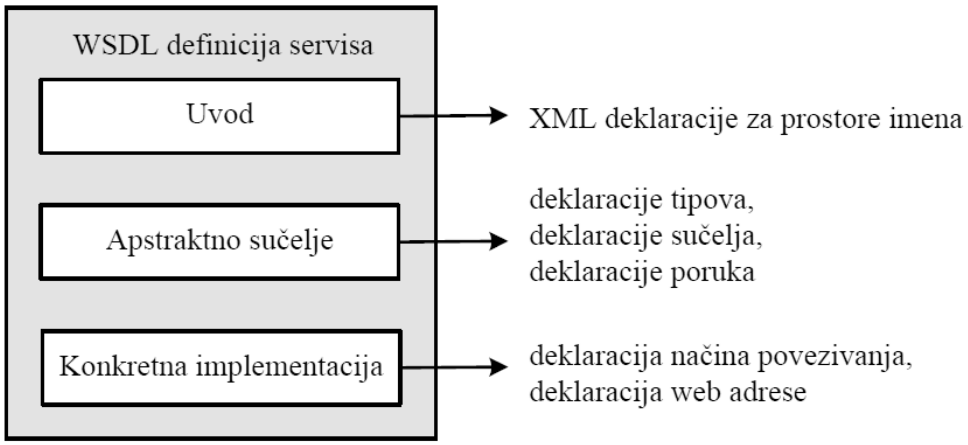
```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:helloResponse xmlns:ns2="http://service.time.org/">
            <return>Hello Alice!</return>
        </ns2:helloResponse>
    </S:Body>
</S:Envelope>
```

Slika 3.5: Primjer SOAP poruke – SOAP odgovor.

3.2 WSDL

Web Service Definition Language – WSDL je jezik za opisivanje sučelja web servisa baziran na XML-u. Omogućuje zadavanje imena operacija koje servis nudi, parametara tih operacija i njihovih tipova, iznimki, adrese na web-u gdje se servis nalazi te načina na koji se aplikacija povezuje sa servisom.

Struktura WSDL dokumenta prikazana je na slici 3.6. Uvodni dio takvog dokumenta sadrži XML-ove deklaracije prostora imena (*namespace*). Nakon uvodnog dijela slijedi apstraktno sučelje koje definira što web servis radi – opisane su operacije (imena operacija, pripadnih parametara i njihovih tipova), poruke i iznimke. Na kraju se nalazi konkretna implementacija u kojoj se definira kako servis komunicira (*binding*) i gdje se on nalazi. *Binding* specificira protokol za komunikaciju sa servisom – u pravilu to je SOAP, no može se odabrati i neki drugi protokol. Odgovor na pitanje "gdje se web servis može naći" određuje adresa na web-u gdje je servis instaliran, dakle njegov *Uniform Resource Identifier — URI*.



Slika 3.6: Organizacija specifikacije u WSDL-u.

Na slici 3.7 je primjer jednog WSDL dokumenta na kojem se detaljnije vidi kako izgledaju elementi koji sačinjavaju prethodno opisanu strukturu.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://service.hello/" name="HelloWebService"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:wsp="http://www.w3.org/ns/ws-policy"
        xmlns:tns="http://service.hello/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
        xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://service.hello/"
        schemaLocation="HelloWebService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="hello">
    <part name="parameters" element="tns:hello"/>
  </message>
  <message name="helloResponse">
    <part name="parameters" element="tns:helloResponse"/>
  </message>
  <portType name="HelloWebService">
    <operation name="hello">
      <input wsam:Action="http://.../helloRequest" message="tns:hello"/>
      <output wsam:Action="http://.../helloResponse" message="tns:helloResponse"/>
    </operation>
  </portType>
  <binding name="HelloWebServicePortBinding" type="tns:HelloWebService">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="hello">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="HelloWebService">
    <port name="HelloWebServicePort" binding="tns:HelloWebServicePortBinding">
      <soap:address location="http://localhost:8080/SimpleWebApplication/HelloWebService"/>
    </port>
  </service>
</definitions>
```

Slika 3.7: Primjer WSDL dokumenta.

3.3 WS-BPEL

Web Service Business Process Execution Language – WS-BPEL je jezik za pisanje aplikacije koja poziva web servise i kombinira njihove ulaze i izlaze. Specificira akcije unutar

i između poslovnih procesa pomoću web servisa. Procesi u WS-BPEL-u izvoze i uvoze informacije koristeći isključivo sučelje web servisa.

Iako se umjesto direktne implementacije u WS-BPEL-u najčešće koriste alati za modeliranje procesa koji generiraju WS-BPEL dokument, korisno je i važno znati osnovne WS-BPEL elemente (slika 3.8). WS-BPEL alati za modeliranje procesa često koriste te elemente, a može se i pojaviti potreba za izmjenom ili nadopunom izvornog koda pisanog u WS-BPEL-u radi daljnjih profinjenja koja se ne mogu postići korištenjem alata.

```
<process>
        <partnerLinks>
                ...
        </partnerLinks>
        <variables>
                ...
        </variables>
        <faultHandlers>
                ...
        </faultHandlers>
        <sequence>
                <recieve ...>
                <invoke ...>
                <reply ...>
                ...
        </sequence>
        ...
</process>
```

Slika 3.8: Struktura WS-BPEL definicije procesa.

3.4 UDDI

Universal Description, Discovery and Integration – UDDI je standard koji propisuje način na koji se servis treba dokumentirati da bi ga tražitelji usluga lako mogli naći. Uključuje informacije o uslugama koje servis pruža, dobavljaču usluga, lokaciji WSDL opisa sučelja te informacije o poslovnim vezama.

Namjera je bila omogućiti dobavljačima usluga stvaranje svojih UDDI registara s opisima servisa koje nude. Tražitelji usluga koristili bi takav UDDI registar kao svojevrstan oblik "poslovnog imenika". Moguće je registrirati tri tipa informacija u UDDI registar:

Bijele stranice. Sadrže osnovne informacije o dobavljaču usluga. To uključuje ime kompanije, opis čime se ona bavi te kontaktne informacije kao što su adresa i broj telefona. Također, omogućuju metodu pronalaska servisa preko identifikatora kompanije.

Žute stranice. Koriste klasifikaciju servisa ili kompanije pomoću standardnih taksonomija te na taj način omogućuju pronalazak servisa prema biranoj kategoriji.

Zelene stranice. Opisuju gdje i kako pristupiti web servisu te sadrže informacije o načinu komunikacije (*binding*). Sadrže tehničke detalje i podržane funkcije servisa.

Neke kompanije, uključujući Microsoft, koristile su UDDI registre na početku 21. stoljeća. No zbog napretka tražilica oni su postali suvišni i danas su svi zatvoreni, a UDDI je pao u zaborav. Servisi se sada mogu uspješno pronaći korištenjem standardnih tražilica pomoću odgovarajućih WSDL opisa.

Poglavlje 4

RESTful web servisi

Sadašnje standarde za web servise neki kritiziraju da su preglomazni, preopćeniti i neefikasni. Korištenje tih standarda zahtijeva znatnu količinu procesiranja za stvaranje, prijenos i interpretaciju povezanih XML poruka. Iz tog su razloga neke kompanije, kao što su Google i Amazon, umjesto standarda počele koristiti jednostavniji i efikasniji pristup servisnoj komunikaciji korištenjem takozvanih RESTful web servisa. RESTful pristup podržava efikasnu interakciju servisa, ali ne podržava značajke na razini poduzeća kao što su *WS-Reliability* i *WS-Transactions*.

4.1 REST

Representational State Transfer – REST je teorijski model programske arhitekture za ostvarivanje raspodijeljenih sustava. Opisao ga je Roy Fielding u petom poglavlju svoje doktorske disertacije [6]. Nastao je iz WWW (*World Wide Web*) tehnologije uvođenjem određenih ograničenja. Ova ograničenja čine osnovne principe REST modela koji određuju kako se resursi na globalnoj mreži Internet mogu koristiti. Motivacija za uvođenje ovih ograničenja je stvaranje konačnog sustava koji koristi sve pogodnosti arhitekture web-a kako bi ostvario bolje funkcioniranje novog sustava.

Budući da spomenuti nestandardni web servisi često implementiraju samo dio teorijskih načela REST modela, za njih se koristi naziv RESTful web servisi. Time se naglašava da se koriste neka REST načela, ali ne sva. U nastavku se nalazi opis ključnih načela REST modela.

Svaki resurs ima jedinstveni identifikator. Osnovni gradivni elementi REST modela nazivaju se resursi (*resources*). Svaki web servis korisnicima nudi pristup konačnom broju resursa. Na primjer, popularni web servisi, kao što je Amazon API, omogućuju dohvaćanje naslova knjiga, korisničkih kritika i drugih informacije. U kontekstu

ovog servisa, knjige i kritike predstavljaju resurse koje web servis nudi korisniku. Kao jedinstveni identifikator resursa odabran je URI (*Uniform Resource Identifier*). Na taj se način resursu dodjeljuje jedinstveni identifikator unutar globalne mreže Internet.

Međusobno povezivanje resursa. Budući da resursi često imaju prirodnu vezu jedan s drugim potrebno ih je međusobno povezati. Na primjer, kritike i knjige kojima te kritike pripadaju međusobno su povezane semantikom. Točnije, kritike nemaju smisla bez knjige na koju se odnose, dok se knjige mogu detaljnije opisati kritikama. Iz tog se razloga prilikom davanja odgovora na zahtjev za resursom često daje skup dodatnih poveznica koje identificiraju druge resurse. Na taj način klijent ima mogućnost promjene stanja sustava prateći poveznice koje je dobio u sklopu odgovora.

Upotreba standardnih metoda. Svaki resurs podržava isti skup metoda kojima se mogu pokrenuti određene operacije ili dohvatiti željeni rezultat. Skup standardnih REST metoda preuzete su iz skupa HTTP metoda. Teorijski model predviđa uporabu GET, POST, PUT i DELETE metoda. Iako je način uporabe tih metoda opisan HTTP standardom i REST teorijskim modelom, one se rijetko kada koriste dosljedno. Iz tog se razloga, kao što smo već spomenuli, koristi pojam RESTful servisi kako bi se naglasilo da neka načela nisu strogo ispunjena. Na primjer, GET metoda bi se trebala koristiti isključio za dohvaćanje resursa. Popularni web servis Flickr ima dokumentiranu operaciju brisanja resursa uporabom GET metode, umjesto standardne DELETE metode.

Resursi s višestrukim reprezentacijama. Jedan od osnovnih problema prilikom izrade web servisa je osiguravanje ispravnog tumačenja poslanih podataka na strani klijenata. Naime, web servisi šalju samo podatke, a klijentska aplikacija ih mora prikazati. Točnije, klijentska aplikacija mora znati kako protumačiti rezultate te ih onda prikazati korisniku. Kako bi se pokrilo što više mogućih reprezentacija, moguće je HTTP zahtjevom zatražiti željeni format. Naravno, web servis mora podržavati isporuku resursa u željenom formatu. Na primjer, kada se web servis koristi s mobilnog uređaja poželjno je dobiti odgovor u XML zapisu. U drugim će uvjetima možda biti pogodnije odgovor primiti u JSON[1] obliku. Neovisno o odabranom formatu, klijent je uvijek zadužen za konačnu reprezentaciju resursa.

Komunikacija bez održavanja stanja. REST model predlaže da informacija o stanju servisa uvijek bude sadržana unutar samog resursa koji se koristi ili da se pohranjuje na klijentu. Stanje se odnosi na bilo koje informacije koje utječu na rezultat odziva

[1] *JavaScript Object Notation* – JSON je jednostavni tekstualni standard dizajniran kako bi omogućio razmjenu i prijenos podataka koje je lagano iščitati bez dodatnih programskih rješenja. Uglavnom se koristi za prijenos podataka između servera i web aplikacije kao alternativa XML-u.

web servisa, a koje su nastale putem prethodnih interakcija s web servisom. Točnije, poslužitelj ne bi trebao pamtiti stanje prilikom komuniciranja s pozivateljem servisa. Jedan od razloga za to je osiguravanje razmjernog rasta. Ukoliko se pamti stanje za svakog klijentu to bi narušilo vrijeme odziva prilikom visokog opterećenja. No, postoje i druge pogodnosti ovog pristupa. Na primjer, klijent postaje neovisan o promjenama na poslužitelju jer ne ovisi o stanju prilikom komunikacije.

Poglavlje 5

Softverski procesi vezani uz web servise

U softverskom inženjerstvu zasnovanom na web servisima razlikujemo dvije vrste softverskih procesa:

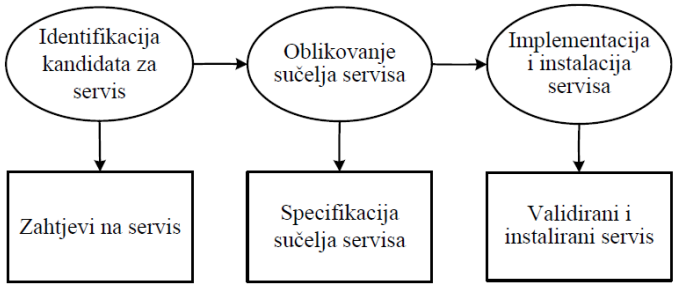
- Razvoj samih web servisa koje će netko drugi upotrebljavati.
- Razvoj aplikacija korištenjem postojećih web servisa.

5.1 Razvoj web servisa

Proces razvoja web servisa prikazan je na slici 5.1 i sastoji se od tri faze:

1. *Identifikacija kandidata za servis.*
2. *Oblikovanje sučelja servisa.*
3. *Implementacija i instalacija servisa.*

Cilj je razviti servise za ponovnu upotrebu u servisno orijentiranim aplikacijama. Dobavljači usluga moraju osigurati da se razvijeni servis može upotrijebiti u različitim sustavima. Da bi to postigli, moraju dizajnirati i razviti općenito korisne funkcionalnosti koje čine servis robusnim i pouzdanim. Također, potrebno je dokumentirati servis tako da ga tražitelji usluga mogu lako otkriti i pravilno koristiti.



Slika 5.1: Proces razvoja web servisa.

Identifikacija kandidata za servis

Razvoj ponovno upotrebljivog servisa obično počinje identifikacijom nekog postojećeg softverskog modula koji se pokazao korisnim i koji bi mogao biti od šireg interesa. U daljnjem postupku razvoja postojeći modul mora se generalizirati, to jest iz njega treba izbaciti specifičnosti njegove originalne aplikacije te mu treba dodati novu funkcionalnost koja će ga učiniti cjelovitijim i nezavisnijim. Postupkom identifikacije kandidata za servis uočavamo moguće servise i definiramo zahtjeve za njih.

Osnovna ideja servisno orijentirane arhitekture je da servisi trebaju podržati poslovne procese. Budući da svaka organizacija ima širok raspon procesa, postoji mnogo potencijalnih servisa koji mogu biti implementirani. Identifikacija kandidata za servis stoga uključuje razumijevanje i analizu poslovnih procesa unutar organizacije u svrhu odlučivanja koji ponovno upotrebljivi servisi mogu biti implementirani da podrže te procese.

Thomas Erl [5] predlaže tri osnovna tipa servisa koji se mogu identificirati:

1. *Komunalni servisi.* To su servisi koji implementiraju neke općenite funkcionalnosti koje mogu koristiti različiti poslovni procesi.

2. *Poslovni servisi.* To su servisi koji su povezani sa specifičnim poslovnim funkcijama.

3. *Koordinacijski ili procesni servisi.* To su servisi koji podržavaju općenitije poslovne procese koji obično uključuju različite aktere i aktivnosti.

Erl također predlaže i podjelu prema kojoj servisi mogu biti orijentirani na zadatke ili na entitete. Servisi orijentirani na zadatke su oni servisi koji su povezani s nekom aktivnošću, dok su servisi orijentirani na entitete kao objekti i povezani su s nekim poslovnim entitetom. Slika 5.2 prikazuje neke primjere servisa s obzirom na spomenute dvije podjele.

Komunalni i poslovni servisi mogu biti orijentirani ili na zadatke ili na entitete, dok su koordinacijski servisi uvijek orijentirani na zadatke.

	Komunalni servisi	**Poslovni servisi**	**Koordinacijski servisi**
Servisi orijentirani na zadatke	Pretvarač valuta Lokator zaposlenika	Forma za validaciju zahtjeva Provjera kreditnog rejtinga	Potvrda procesnih troškova Plaćanje vanjskom dobavljaču
Servisi orijentirani na entitete	Provjera stila dokumenta Pretvarač web forme u XML	Forma za troškove Forma za prijavu studenata	

Slika 5.2: Klasifikacija servisa.

Cilj identifikacije kandidata za servis je pronalaženje servisa koji bi trebali biti logički koherentni, samostalni i ponovno upotrebljivi. Erlova klasifikacija korisna je u ovom pogledu zato što nam sugerira kako otkriti ponovno upotrebljive servise promatrajući poslovne entitete i aktivnosti. Međutim, identifikacija kandidata za servis ponekad je vrlo teška jer trebamo predvidjeti kako će servis biti korišten. Od mogućih kandidata za servis potrebno je odabrati one koji će biti korisni, a za to je potrebno mnogo vještine i iskustva.

Rezultat identifikacije kandidata za servis je skup izabranih servisa te pripadni zahtjevi za njih. Funkcionalni zahtjevi opisuju što servis treba raditi. Nefunkcionalni zahtjevi definiraju zahtjeve za sigurnost, performanse i dostupnost servisa.

Oblikovanje sučelja servisa

Nakon identifikacije kandidata za servis, sljedeća faza razvoja web servisa je oblikovanje sučelja servisa. Ova faza uključuje definiranje operacija vezanih za servis i njihovih parametara. Također, potrebno je posvetiti pažnju dizajnu operacija i poruka. Cilj je minimizirati broj razmjena poruka koje su potrebne da bi servis odgovorio na zahtjev. Bolje je da što više informacija bude proslijeđeno u jednoj poruci nego da se koristi sinkrona komunikacija pomoću više poruka. Važno je još imati na umu da web servisi nemaju stanja pa upravljanje stanjima aplikacije koja koristi servise treba prepustiti korisnicima servisa, a ne samom servisu. Zbog toga će možda biti potrebno da se informacije o stanju aplikacije prenose u i iz servisa pomoću poruka.

Oblikovanje sučelja servisa sastoji se od sljedećih koraka:

1. *Logičko oblikovanje sučelja.* Definiramo operacije koje će servis izvršavati, njihove ulazne i izlazne parametre te iznimke vezane uz te operacije.

2. *Oblikovanje poruka.* Definiramo strukturu poruka koje servis prima i šalje.

3. *Razvoj u WSDL-u.* Rezultati logičkog oblikovanja i oblikovanja poruka prevode se u apstraktni opis sučelje zapisan u jeziku WSDL.

Implementacija i instalacija servisa

Zadnja faza razvoja web servisa je implementacija koja se svodi na programiranje u nekom od standardnih programskih jezika kao što su Java ili C#. Oba jezika posjeduju odgovarajuće biblioteke i podršku za razvoj web servisa. Alternativni način implementacije servisa je modificiranje već postojećeg programskog koda.

Nakon implementacije, servis je potrebno testirati prije instaliranja. Testiranje uključuje ispitivanje ulaznih parametara servisa, stvaranje ulaznih poruka koje daju informacije o ulaznim parametrima te provjeru jesu li dobiveni izlazni parametri očekivani. Tijekom testiranja treba pokušati stvoriti iznimke da bi se provjerilo može li se servis nositi s nevažećim ulaznim parametrima. Postoje alati za testiranje servisa koji generiraju automatske testove iz WSDL specifikacije. Međutim, takvi testovi omogućuju samo provjeru valjanosti sučelja servisa, pomoću njih nije moguće testirati funkcionalno ponašanje servisa.

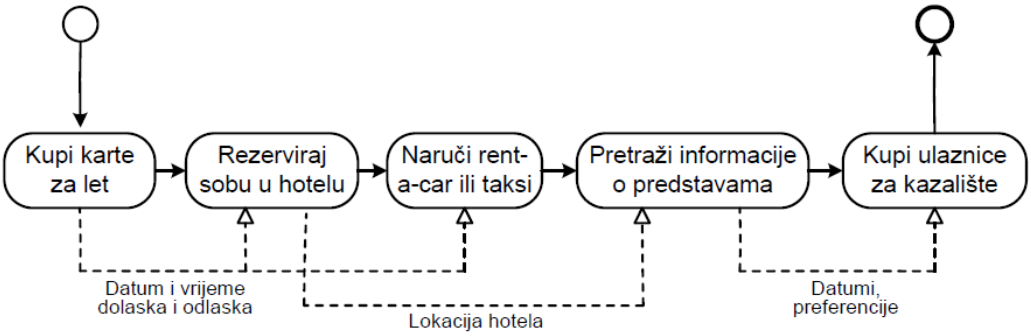
Na samom kraju slijedi instalacija kojom servis postaje dostupan za korištenje na web serveru. Instalacija servisa svodi se na kopiranje izvršivog programa u određeni direktorij te od stavljanja WSDL opisa i ostalih informacija o servisu na web.

5.2 Razvoj aplikacija korištenjem web servisa

Razvoj aplikacija korištenjem web servisa temelji se na ideji sastavljanja i konfiguriranja već postojećih servisa da bi se stvorili novi, složeni servisi. Takva kompozicija servisa može se integrirati s korisničkim sučeljem implementiranim u pregledniku za stvaranje web aplikacija ili se može iskoristiti kao sastavni dio za neku drugu vrstu aplikacije. Servisi koji se koriste u kompoziciji mogu biti razvijeni posebno za aplikaciju, mogu se koristiti poslovni servisi razvijeni unutar kompanije ili servisi od vanjskih dobavljača.

Tijek aplikacije koja koristi web servise možemo zamisliti kao niz odvojenih koraka. Iz svakog se koraka sve potrebne informacije prenose u sljedeći korak. Ovaj niz koraka nazivamo radni tok (*workflow*). *Workflow* je model poslovnog procesa i označava niz aktivnosti koje se moraju izvršiti vremenski jedna za drugom da bi se postigao neki poslovni cilj. U našem slučaju aktivnosti se izvršavaju tako da se pozove odgovarajući web servis. Ideja *workflow*-a je prilično jednostavna, ali se zbog česte složenosti kompozicije servisa može dogoditi da je u *workflow* potrebno naknadno dodati korake ili se zbog nekompatibilnosti u izvršavanju servisa može pojaviti potreba za promjenom *workflow*-a.

Na slici 5.3 prikazan je *workflow* aplikacije za planiranje odmora. Riječ je o aplikaciji koja klijentu avionske kompanije omogućuje kupovinu avionske karte, rezervaciju hotelske sobe u odredištu, narudžbu rent-a-car-a ili taksija te kupovinu ulaznica za kazalište. Prikazane aktivnosti zapravo su pozivi različitih web servisa. Prvi servis pripada avionskoj kompaniji, a preostali su vanjski servisi koji pripadaju kompanijama iz odredišta.



Slika 5.3: *Workflow* aplikacije za planiranje odmora.

Proces razvoja aplikacije korištenjem web servisa prikazan je na slici 5.4 i sastoji se od šest koraka:

1. *Oblikovanje polaznog workflow-a.* Na osnovi zahtjeva na aplikaciju predlažemo idealni tijek aktivnosti unutar aplikacije.

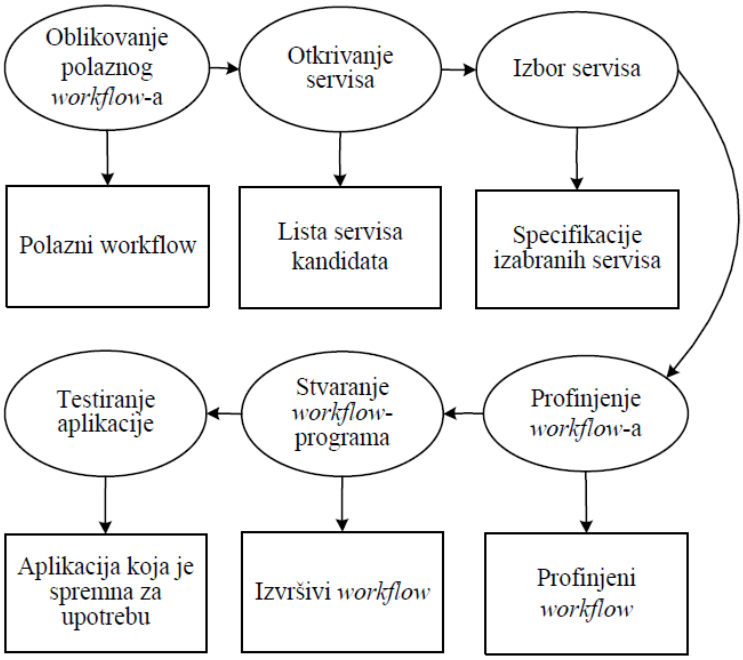
2. *Otkrivanje servisa.* Pretražujemo odgovarajuće registre ili kataloge da bi ustanovili koji nama zanimljivi servisi nam stoje na raspolaganju, te tko ih daje i pod kojim uvjetima.

3. *Izbor servisa.* Iz skupa mogućih kandidata koje smo otkrili biramo one servise koji najbolje mogu implementirati aktivnosti iz našeg *workflow*-a.

4. *Profinjenje workflow-a.* Na osnovi informacija o izabranim servisima profinjujemo polazni *workflow*. To znači da dodajemo detalje o operacijama i parametrima te možda dodajemo ili izbacujemo aktivnosti. Da bi se izabrani servisi što bolje mogli upotrijebiti, često je potrebno napraviti manje kompromise u funkcionalnosti aplikacije.

5. *Stvaranje workflow-programa.* Profinjeni *workflow* pretvaramo u izvršivi program. U tu svrhu koristimo konvencionalni programski jezik poput Java ili C#, ili specijalizirani jezik WS-BPEL.

6. *Testiranje aplikacije.* Provjerava se da li *workflow*-program stvoren u prethodnom koraku ispravno radi i da li on zadovoljava uvjete iz specifikacije.



Slika 5.4: Proces razvoja aplikacije koja koristi web servise.

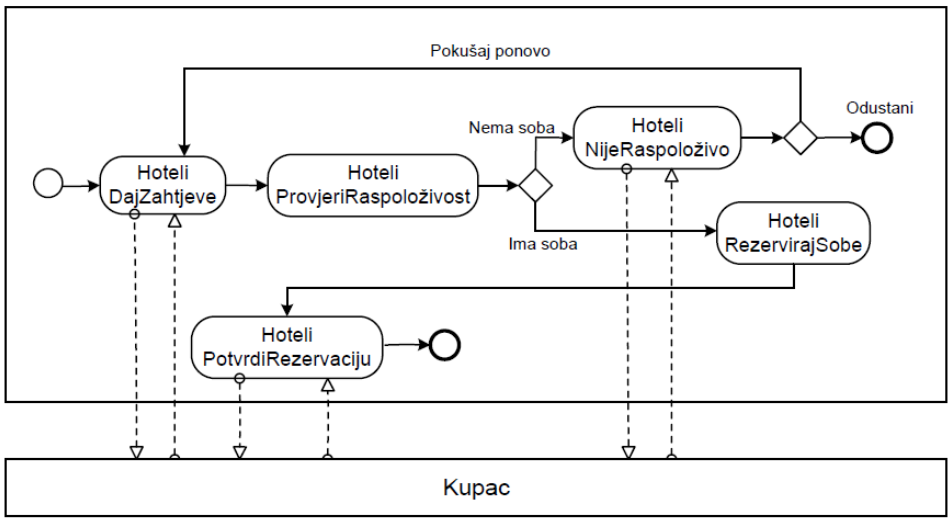
Oblikovanje i implementacija *workflow*-a

Oblikovanje *workflow*-a uključuje analizu postojećih ili planiranih poslovnih procesa u svrhu razumijevanja različitih aktivnosti od kojih se sastoje i načina razmjene informacija između njih. Na temelju toga definira se novi poslovni proces korištenjem notacije za oblikovanje *workflow*-a. Prikazuju se faze od kojih se proces sastoji i informacije koje se prenose između različitih faza tog procesa.

Workflow predstavlja model poslovnog procesa i uglavnom se oblikuje korištenjem grafičke notacije, pomoću UML *activity* dijagrama ili BPMN (*Business Process Modeling Notation*). UML *activity* dijagrami i BPMN su vrlo slični pa postoji mogućnost da se u budućnosti integriraju i tako stvoreni jezik postane standard za oblikovanje *workflow*-a. *Activity* dijagram prikazuje aktivnosti u nekom procesu te tok kontrole ili tok podataka u tom procesu. BPMN je grafički jezik koji je relativno lako razumjeti. Prednost BPMN-a je u tome što postoje alati koji BPMN prevode u WS-BPEL, jezik niže razine zasnovan na XML-u koji je kompliciran za ručno pisanje.

Slika 5.5 primjer je jednostavnog BPMN dijagrama koji prikazuje dio *workflow*-a aplikacije za planiranje odmora sa slike 5.3. Točnije, ovo je dio za rezervaciju hotela koji

uključuje traženje i dobivanje zahtjeva od kupaca, provjeru slobodnih soba te samu rezervaciju ako postoji odgovarajuća slobodna soba.



Slika 5.5: BPMN dijagram za rezervaciju hotela.

Na prethodnoj slici vide su neki osnovni grafički elementi BPMN-a koji se koriste za oblikovanje *workflow*-a:

- Aktivnosti su nacrtane kao pravokutnici s oblim rubovima, a pokreću ih ljudi ili automatizirani servisi.
- Događaji su nacrtani kao krugovi i pokreću se tijekom poslovnih procesa. Krug s običnim rubom koristi se za početni događaj, a krug s podebljanim rubom za završni. Na slici nije prikazan krug s dvostrukim rubom koji se inače koristi za događaje koji se pojavljuju između početnog i završnog događaja.
- Rombovi označavaju grananje, dio procesa u kojem se donosi odluka. Jedan primjer na slici 5.5 je donošenje odluke ovisno o tome ima li slobodnih soba ili ne.
- Pune strelice predstavljaju tok kontrole, a isprekidane strelice tok podataka. Na slici 5.5 prikazana je razmjena podataka između servisa za rezervaciju hotela i kupca.

Nakon završetka početnog oblikovanja, *workflow* je potrebno profiniti na osnovi otkrivenih, odnosno izabranih servisa. Profinjenje *workflow*-a može zahtijevati niz iteracija sve

dok ne bude stvoren dizajn koji omogućava maksimalnu ponovnu upotrebu raspoloživih servisa.

Na osnovi konačnog, profinjenog *workflow*-a stvara se izvršivi program. To može uključivati sljedeće dvije aktivnosti:

1. Implementacija servisa koji nisu dostupni za ponovnu upotrebu, a potrebni su za rad aplikacije. U tu svrhu koristimo standardne programske jezike kao što su Java i C# jer imaju podršku za razvoj web servisa. Razvoj web servisa za ponovnu upotrebu opisan je u odjeljku 5.1.

2. Stvaranje *workflow*-programa koji predstavlja izvršivu verziju *workflow*-a. *Workflow*-program obično se generira ručnim ili automatskim prevođenjem u WS-BPEL. Iako postoje alati koji automatski prevode BPMN u WS-BPEL, pod nekim je okolnostima teško proizvesti čitljivi kod u WS-BPEL-u pa je potrebno dodatno ručno kodiranje. Ipak, razvojem odgovarajućih standarda za web servise opisanih u poglavlju 3 omogućava se sve kvalitetnija direktna implementacija.

Testiranje aplikacije

Testiranje je važno u svim razvojnim procesima jer ono pokazuje da sustav zadovoljava funkcionalne i nefunkcionalne zahtjeve te otkriva greške koje se mogu proizvesti tijekom razvojnog procesa. Mnoge tehnike za testiranje koriste analizu programskog koda. Međutim, ako su servisi u vlasništvu vanjskih dobavljača, izvorni kod implementacije servisa nije dostupan. Iz ovog razloga testiranje aplikacije koja koristi web servise nije moguće pomoću tehnika koje koriste analizu programskog koda.

Osim problema razumijevanja implementacije servisa, kod testiranja servisa i aplikacija koje koriste servise mogu se pojaviti i sljedeće teškoće:

- Servisi su pod kontrolom dobavljača servisa, a ne korisnika servisa. Dobavljači servisa mogu u svakom trenutku povući svoje servise ili napraviti promjene na njima što svaki put zahtjeva novo testiranje aplikacije ako želimo da ona sa sigurnošću ispravno funkcionira.

- Dugoročna vizija SOA je da servisi budu dinamički povezani sa servisno orijentiranim aplikacijama. To znači da aplikacija prilikom izvršavanja ne treba svaki put koristiti isti servis. Iako testiranje može biti uspješno kada aplikacija koristi određeni servis, ne postoji garancija da će upravo taj servis biti korišten prilikom stvarnog izvršavanja aplikacije pa njezina ispravnost tada može biti upitna.

- Teško je provjeriti hoće li biti ispunjeni nefunkcionalni zahtjevi jer njihovo ispunjavanje ne ovisi samo o tome kako aplikacija radi kad se testira. Moguće je da servis

dobro radi tijekom testiranja jer nema velikog opterećenja, ali u praksi ponašanje servisa može biti drukčije zbog potencijalno velikog broja korisnika pa zbog toga postoji mogućnost da aplikacija neće raditi korektno.

- Troškovi i kvaliteta testiranja ovise o modelu plaćanja servisa. Ako su servisi besplatni, tada dobavljači servisa vjerojatno neće posvetiti veliku pažnju testiranju. Ako je za servise potrebna pretplata ili ako se plaćaju po korištenju, korisnici servisa očekuju da servisi budu dobro testirani za što je potrebno uložiti vrijeme i novac.

- Akcije koje se pozivaju nakon pojave iznimaka mogu ovisiti o neispravnom radu drugih servisa. Problem kod testiranja takvih akcija je u tome što je teško osigurati da odgovarajući servisi zakažu baš prilikom testiranja.

Spomenuti problemi posebno su izraženi ako se koriste vanjski servisi. Oni su manje izraženi ako se koriste servisi unutar iste kompanije ili ako kompanije koje surađuju međusobno razmjenjuju svoje servise. U takvim je slučajevima izvorni kod najčešće dostupan pa može olakšati testiranje aplikacije. Rješavanje problema koji se pojavljuju kod testiranja te pronalaženje tehnika i alata za testiranje servisno orijentiranih aplikacija i dalje su važan predmet istraživanja.

Poglavlje 6

Java web servisi

Java je objektno orijentirani programski jezik koji je razvio James Gosling u kompaniji Sun Microsystems. Razvoj ovog programskog jezika počeo je 1991. godine kao dio projekta Green, a objavljen je 1995. godine. Ideja je bila stvoriti programski jezik neovisan o platformi, baziran na C-u i C++-u, ali s pojednostavljenom sintaksom, stabilnijom *runtime* okolinom te boljom kontrolom memorije.

Java je, uz C#, najčešće korišteni programski jezik za implementaciju web servisa. Naime, današnje programerske okoline za rad s Javom poput Eclipse ili NetBeans IDE (*Integrated Development Environment*) sadrže svu potrebnu podršku za razvoj web servisa. Opskrbljene su odgovarajućim bibliotekama i u stanju su proizvesti odgovarajući izvršivi program s propisanim sučeljem.

U nastavku slijedi opis izrade i primjer korištenja JAX-WS[1] web servisa. Riječ je o servisu za pretvaranje mjernih jedinica temperature, točnije za pretvaranje Celzijevih stupnjeva u Fahrenheitove i obratno. Web servis i klijentske aplikacije izrađene su u NetBeans IDE-u.

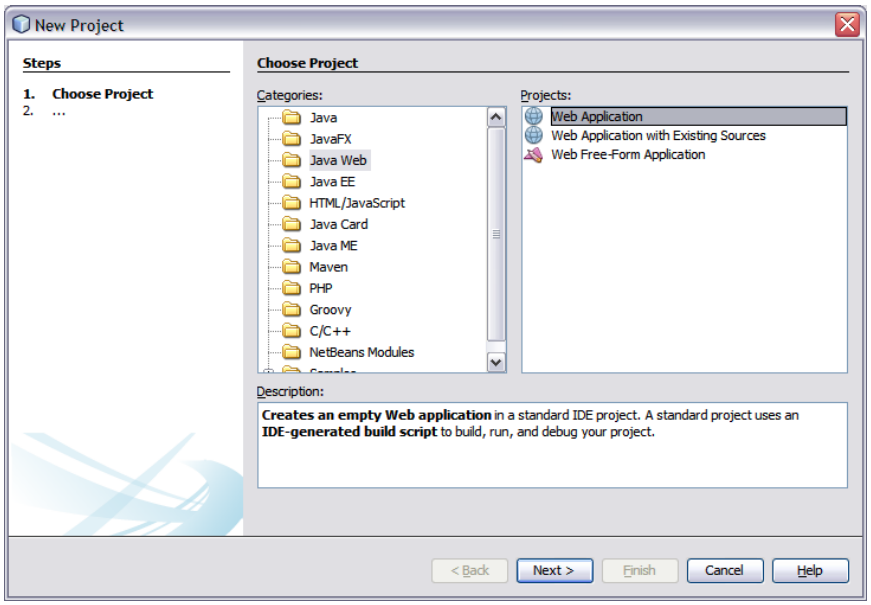
6.1 Izrada servisa

Na početku je potrebno stvoriti novi projekt i odabrati odgovarajući spremnik (*container*) za razvoj web servisa. Java web servise moguće je razvijati u web spremniku ili EJB spremniku. U ovom primjeru korišten je web spremnik.

Postupak stvaranja novog projekta za razvoj web servisa za pretvaranje mjernih jedinica temperature je sljedeći:

[1] *Java API for XML-Web Services* – JAX-WS je aplikacijski okvir za izradu web servisa koji koriste XML poruke koje slijede SOAP standard.

1. Odabiremo File > New Project (Ctrl+Shift+N) i zatim označimo Web Application iz Java Web kategorije (slika 6.1) te kliknemo Next.



Slika 6.1: Izbor spremnika za projekt.

2. Nazovimo projekt `ConverterWSApplication`, izaberemo lokaciju za projekt i kliknemo Next.

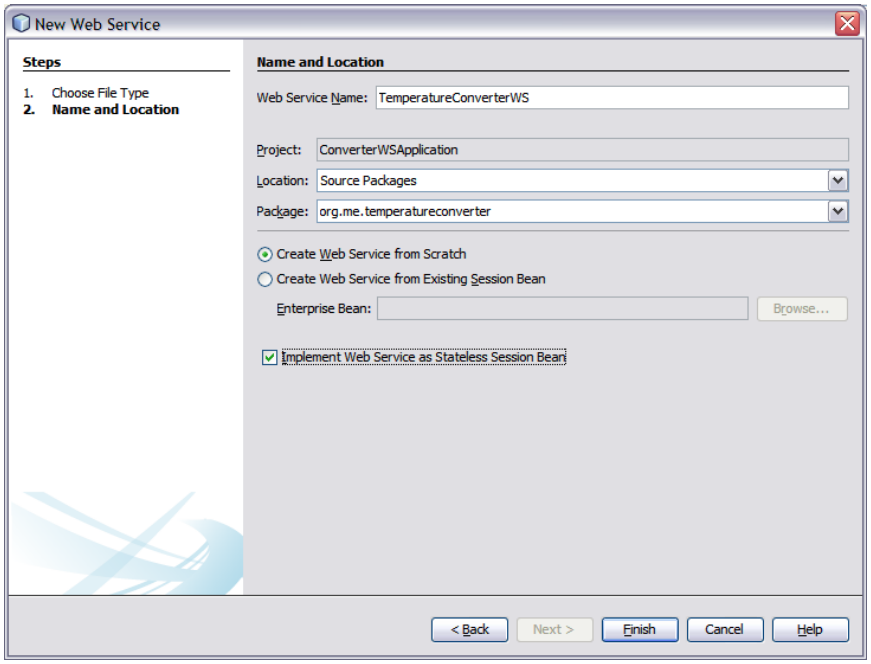
3. Za poslužitelja odabiremo GlassFish Server, a za verziju Java EE 7 Web. Na kraju kliknemo Finish.

Nakon stvaranja projekta možemo stvoriti web servis iz Java klase i to činimo na sljedeći način:

1. Desnim klikom na čvor `ConverterWSApplication` otvara se padajući izbornik iz kojeg odabiremo New > Web Service.

2. Nazovimo web servis `TemperatureConverterWS` i upišemo u Package za ime paketa `org.me.temperatureconverter`. Ostavimo Create Web Service from Scratch označeno i označimo Implement Web Service as a Stateless Session Bean. (slika 6.2)

3. Kliknemo Finish. Nakon toga se u dijelu s projektima prikazuje struktura stvorenog web servisa, a izvorni kod servisa pojavljuje se u editoru.

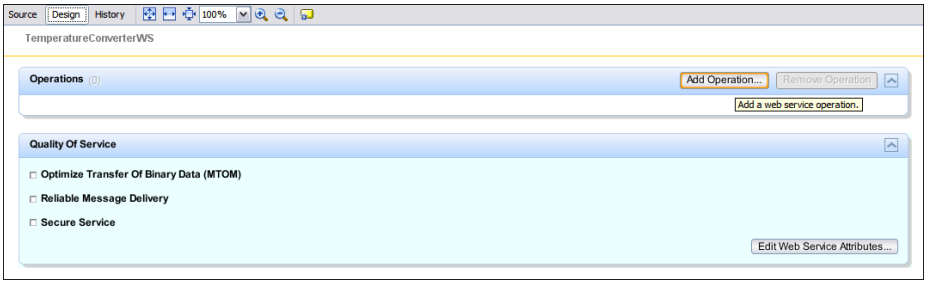


Slika 6.2: Stvaranje web servisa.

Cilj web servisa za pretvaranje mjernih jedinica temperature je pretvaranje temperature primljene od klijenta u Celzijevim stupnjevima u Fahrenheitove, i obratno temperature zadane u Fahrenheitovovim stupnjevima u Celzijeve, pa dodajemo odgovarajuće operacije. NetBeans IDE pruža prozor za dodavanje operacija u web servis (slika 6.5).

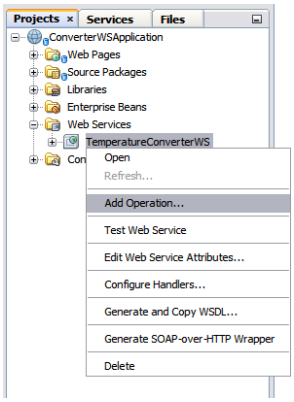
Postupak dodavanja operacije `convertCelsiusToFahrenheit` u servis je sljedeći:

1. Otvorimo Design View u editoru i kliknemo Add Operation (slika 6.3) ili desnim



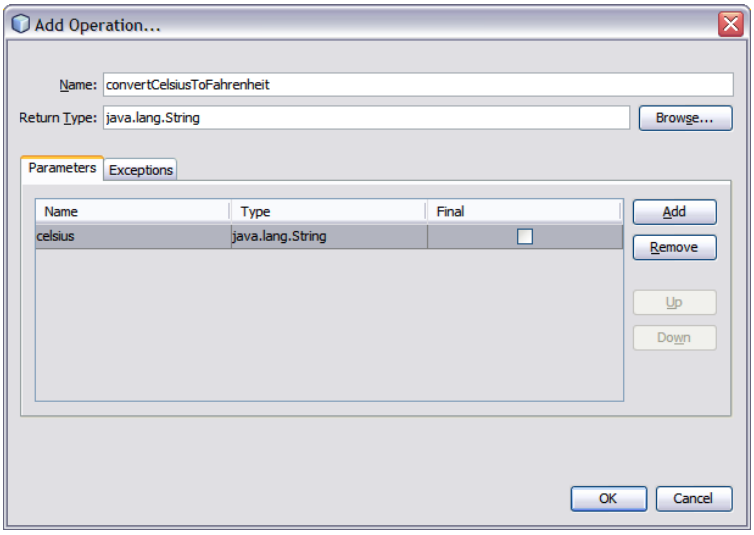
Slika 6.3: Dodavanje operacije.

klikom na čvor `TemperatureConverterWS` otvaramo padajući izbornik i iz njega odaberemo Add Operation (slika 6.4).



Slika 6.4: Dodavanje operacije.

2. Otvara se prozor za dodavanje operacije u čijem gornjem dijelu za ime operacije upisujemo `convertCelsiusToFahrenheit`, a za povratni tip `java.lang.String`. U donjem dijelu prozora klikom na Add dodajemo parametar `celsius` koji će biti tipa `java.lang.String`. (slika 6.5)



Slika 6.5: Prozor za dodavanje operacije.

3. Na kraju kliknemo OK i otvorimo Source View u editoru za uređivanje operacije, to jest za pisanje postupka pretvorbe Celzijevih stupnjeva u Fahrenheitove.

Analogno se dodaje operacija `convertFahrenheitToCelsius` kojom se Fahrenheitovi stupnjevi pretvaraju u Celzijeve. Na slici 6.6 nalazi se gotov izvorni kod web servisa `TemperatureConverterWS` koji sadrži obje operacije.

```
package org.me.temperatureconverter;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.ejb.Stateless;

/**
 *
 * @author Administrator
 */
@WebService(serviceName = "TemperatureConverterWS")
@Stateless()
public class TemperatureConverterWS {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "convertCelsiusToFahrenheit")
    public String convertCelsiusToFahrenheit
            (@WebParam(name = "celsius") String celsius) {
        try {
            double cel = Double.parseDouble(celsius.replace(",", ".").trim());
            return String.valueOf(cel * 9 / 5 + 32);
        } catch(NullPointerException|NumberFormatException e) {
            return "Error";
        }
    }

    /**
     * Web service operation
     */
    @WebMethod(operationName = "convertFahrenheitToCelsius")
    public String convertFahrenheitToCelsius
            (@WebParam(name = "fahrenheit") String fahrenheit) {
        try {
            double fahr = Double.parseDouble(fahrenheit.replace(",", ".").trim());
            return String.valueOf((fahr - 32) / 9 * 5);
        } catch(NullPointerException|NumberFormatException e) {
            return "Error";
        }
    }
}
```

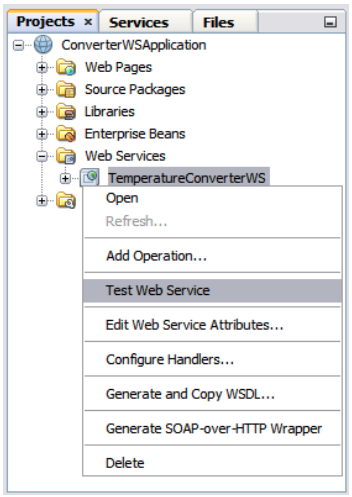
Slika 6.6: Izvorni kod web servisa `TemperatureConverterWS`.

6.2 Testiranje servisa

Nakon instalacije web servisa na poslužitelja, u IDE-u možemo otvoriti klijenta za testiranje servisa ako ga poslužitelj ima. GlasFish Server kojeg koristimo omogućava takvo testiranje na sljedeći način:

1. Desnim klikom na čvor `ConverterWSApplication` otvara se padajući izbornik iz kojeg odabiremo Deploy. IDE pokreće poslužitelja, gradi aplikaciju i instalira je na poslužitelja.

2. Ako je potrebno, proširimo čvor Web Services u projektu i desnim klikom na čvor `TemperatureConverterWS` otvorimo padajući izbornik i iz njega odaberemo Test Web Service (slika 6.7).

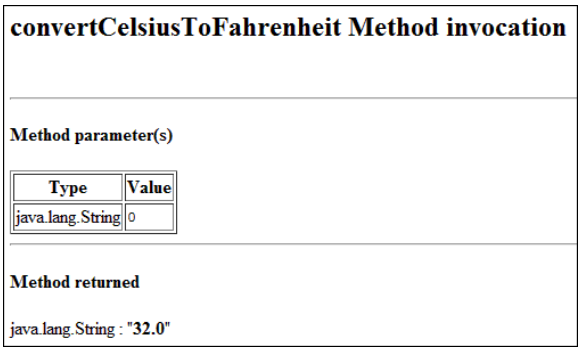


Slika 6.7: Testiranje web servisa.

3. IDE u web pregledniku otvara stranicu za testiranje (slika 6.8) koja omogućava poziv metoda `convertCelsiusToFahrenheit` i `convertFahrenheitToCelsius` tako da u odgovarajuće polje upišemo željenu vrijednost parametra te pritisnemo Enter ili kliknemo na gumb s imenom metode koju želimo pozvati. Na primjer, na slici 6.9 prikazan je poziv metode `convertCelsiusToFahrenheit` s parametrom "0".



Slika 6.8: Stranica za testiranje web servisa.



Slika 6.9: Poziv metode `convertCelsiusToFahrenheit`.

6.3 Korištenje servisa

Za korištenje web servisa preko mreže potrebno je stvoriti klijentsku aplikaciju. U NetBeans IDE-u postoji čarobnjak *Web Service Client* koji u klijentskoj aplikaciji generira kod za korištenje web servisa. U nastavku slijede primjeri dva klijenta koji koriste web servis `TemperatureConverterWS` – Java klasa u Java SE aplikaciji i JSP stranica u web aplikaciji.

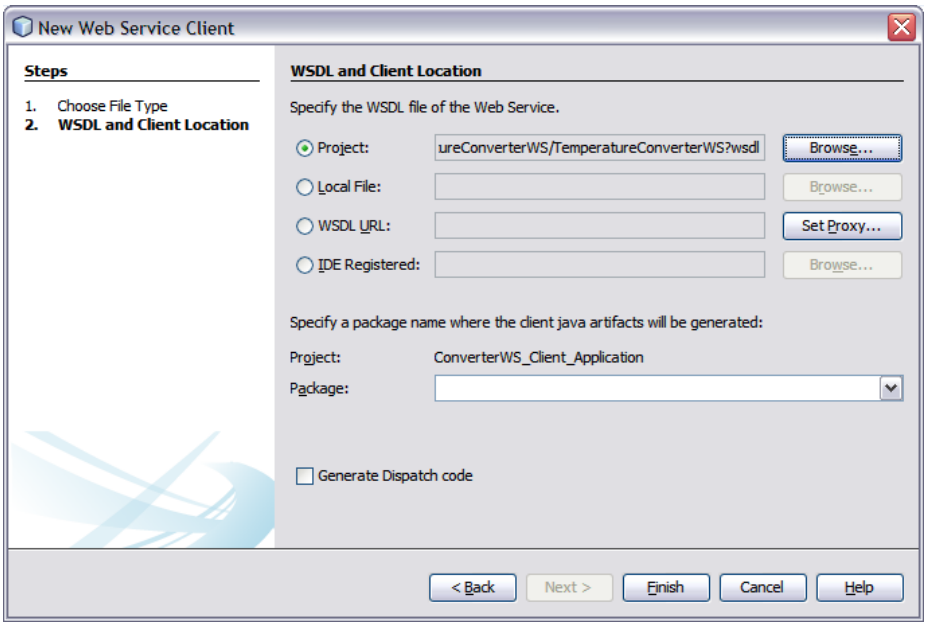
Klijent 1: Java klasa u Java SE aplikaciji

Standardna Java aplikacija koja koristi web servis `TemperatureConverterWS` stvorena je na sljedeći način:

1. Odabiremo File > New Project (Ctrl+Shift+N) i zatim označimo Java Application iz Java kategorije. Nazovimo aplikaciju `ConverterWS_Client_Application`. Ostavimo Create Main Class označeno i prihvatimo ostale zadane postavke. Kliknemo Finish.

2. Desnim klikom na čvor `ConverterWS_Client_Application` otvara se padajući izbornik iz kojeg odabiremo New > Web Service Client.

3. Otvara se prozor New Web Service Client (slika 6.10). Označimo Project za WSDL izvor i kliknemo Browse. Pronađemo web servis `TemperatureConverterWS` u projektu `ConverterWSApplication`. Označimo ga i kliknemo OK. Ostavimo polje za unos imena paketa prazno i prihvatimo ostale zadane postavke. Kliknemo Finish.



Slika 6.10: Stvaranje klijenta za web servis.

4. U glavnu klasu dodajemo operacije web servisa `TemperatureConverterWS` tako da dovučemo čvor s imenom željene operacije ili desnim klikom u editoru otvorimo padajući izbornik i iz njega izaberemo Insert Code > Call Web Service Operation. Na slici 6.11 prikazan je kod koji se tada automatski generira za operaciju `convertCelsiusToFahrenheit`. To je funkcija koja prima parametar `celsius` tipa `java.lang.String`, poziva operaciju `convertCelsiusToFahrenheit` web servisa `TemperatureConverterWS` i vraća njezin izlazni parametar.

```
private static String convertCelsiusToFahrenheit(java.lang.String celsius) {
    org.me.temperatureconverter.TemperatureConverterWS_Service service =
            new org.me.temperatureconverter.TemperatureConverterWS_Service();
    org.me.temperatureconverter.TemperatureConverterWS port =
             service.getTemperatureConverterWSPort();
    return port.convertCelsiusToFahrenheit(celsius);
}
```

Slika 6.11: Generirani kod za operaciju `convertCelsiusToFahrenheit`.

5. U tijelu `main()` metode zamijenimo `TODO` komentar s kodom koji inicijalizira parametre `cel` i `fahr` te ispisuje rezultat operacije `convertCelsiusToFahrenheit`, odnosno `convertFahrenheitToCelsius` (slika 6.12). U slučaju iznimke, program ispisuje `"Exception:"` te tip iznimke koja se dogodila.

```
public static void main(String[] args) {
        try {
        double cel = 0;
        System.out.println("Converting␣Celsius␣to␣Fahrenheit...");
        System.out.println(cel + "␣C␣=␣" +
                           convertCelsiusToFahrenheit(String.valueOf(cel)) + "␣F");
        double fahr = 32;
        System.out.println("Converting␣Fahrenheit␣to␣Celsius...");
        System.out.println(fahr + "␣F␣=␣" +
                           convertFahrenheitToCelsius(String.valueOf(fahr)) + "␣C");
        } catch (Exception ex) { System.out.println("Exception:␣" + ex); }
}
```

Slika 6.12: Implementirana `main()` metoda koja koristi operacije web servisa.

Aplikaciju `ConverterWS_Client_Application` pokrećemo s Run (F6). Na slici 6.13 prikazan je dobiveni ispis u Output prozoru.

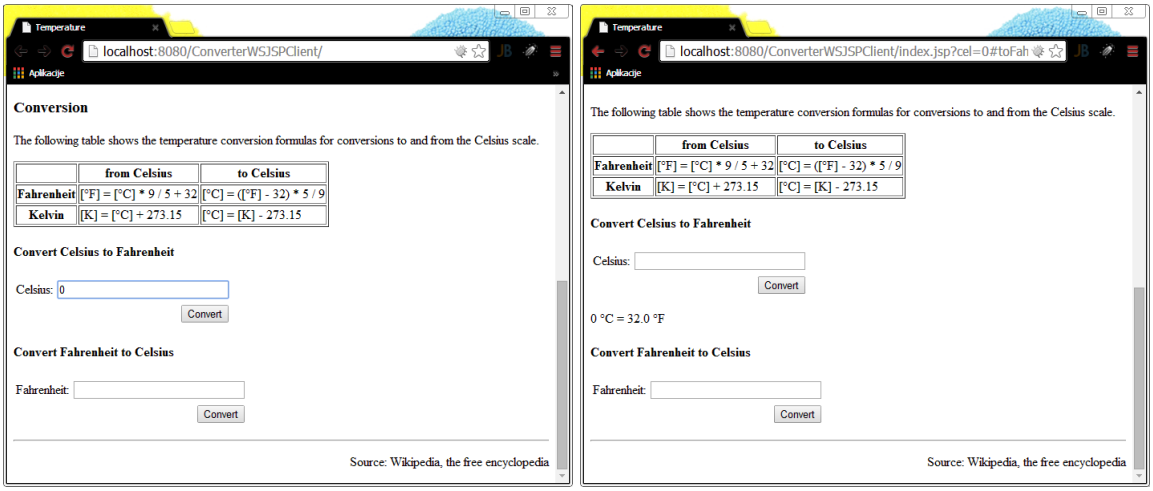
```
run:
Converting Celsius to Fahrenheit...
0.0 C = 32.0 F
Converting Fahrenheit to Celsius...
32.0 F = 0.0 C
BUILD SUCCESSFUL (total time: 1 second)
```

Slika 6.13: Ispis u Output prozoru.

Klijent 2: JSP stranica u web aplikaciji

Web aplikacija koja koristi web servis `TemperatureConverterWS` stvara se na sličan način kao i standardna Java aplikacija. Razlika je u tome što, umjesto u klasu, operacije web servisa dodajemo u JSP datoteku (`index.jsp`).

Na slici 6.14 prikazan je dio JSP stranice web aplikacije `ConverterWSJSPClient`. Operacije web servisa `TemperatureConverterWS` pozivaju se tako da se nakon unosa vrijednosti parametra klikne Convert. Rezultat pretvorbe tada postaje vidljiv na stranici u paragrafu ispod polja za unos.



Slika 6.14: Web aplikacija koja koristi web servis `TemperatureConverterWS`.

Poglavlje 7

Zaključak

U ovom radu bavimo se razvojem softvera zasnovanom na web servisima koji je jedna od aktualnih tehnika ponovne upotrebe. Najprije uvodimo pojam ponovne upotrebe softvera, ukratko opisujemo vrste i neke tehnike ponovne upotrebe te uspoređujemo dvije tehnike koje predstavljaju današnji *state-of-the-art* ponovne upotrebe: razvoj zasnovan na komponentama i razvoj zasnovan na web servisima. Nešto više o ponovnoj upotrebi softvera i nekim tehnikama moguće je pročitati u poglavlju 16 knjige [10], a o razvoju zasnovanom na komponentama u poglavlju 17 iste knjige. U sljedećem poglavlju govorimo o web servisima kao temelju nove arhitekture distribuiranih sustava koja se naziva arhitektura usmjerena na servise (SOA) (više u [5] i [8]). Nakon toga nabrajamo i opisujemo standardne jezike i protokole koji omogućuju upotrebu web servisa, a zatim govorimo o RESTful web servisima [9]. U nastavku proučavamo softverske procese koji su vezani uz web servise, dakle proces njihovog razvoja, odnosno proces razvoja aplikacija koje ih koriste. Na kraju govorimo o Java web servisima [7], opisujemo postupak izrade jednog web servisa te klijentskih aplikacija koje ga koriste.

Vidimo da web servisi predstavljaju nadogradnju klasične web tehnologije. Korištenjem web servisa sadržaji s web-a, koji su isprva bili dostupni samo web preglednicima, postaju dostupni i drugim programima. Tehnologija web servisa omogućuje kompanijama da jedna drugoj stave na raspolaganje svoje poslovne funkcije što omogućuje razvoj fleksibilnih sustava s intenzivnom međusobnom razmjenom informacija. Na taj način web servisi ostvaruju ideju globalne softverske industrije gdje se rješenje stvara na jednom mjestu, a ponovo se upotrebljava bilo gdje drugdje u svijetu. No, zbog načina komuniciranja preko XML-a i internetskih protokola, web servisi su relativno spori pa nisu pogodni za aplikacije gdje se traže izrazito dobre performanse. Također, dijeljenje funkcionalnosti znači da sigurnosni nedostaci jednog web servisa imaju utjecaj i na poslovanje kompanija koje taj servis koriste. Porast upotrebe web servisa u svakodnevnom poslovanju povećava potrebu za konkretnim metodologijama ispitivanja sigurnosti koje su još uvijek predmet istraživanja.

Bibliografija

[1] *Developing JAX-WS Web Service Clients*, https://netbeans.org/kb/docs/websvc/client.html.

[2] *Getting Started with JAX-WS Web Services*, https://netbeans.org/kb/docs/websvc/jax-ws.html.

[3] *Web Services Tutorial*, http://www.w3schools.com/webservices/.

[4] R. Daigneau, *Service design patterns : fundamental design solutions for SOAP/WSDL and RESTful Web services*, Addison-Wesley, Upper Saddle River, NJ, 2012.

[5] T. Erl, *Service-oriented architecture : concepts, technology, and design*, Prentice Hall Professional Technical Reference, Upper Saddle River, NJ, 2005.

[6] R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, doktorski rad, 2000.

[7] M. Kalin, *Java web services : up and running*, O'Reilly, Beijing Sebastopol, Calif, 2009.

[8] E. Newcomer i G. Lomow, *Understanding SOA with Web services*, Addison-Wesley, Upper Saddle River, NJ, 2005.

[9] L. Richardson, *RESTful web services*, O'Reilly, Farnham, 2007.

[10] I. Sommerville, *Software engineering*, Pearson, Boston, 2011.

[11] S. A. White, *Process modeling notations and workflow patterns*, Workflow handbook **2004** (2004), 265–294.

Sažetak

Web servis je računalni ili podatkovni resurs na web-u koji je prikazan na standardizirani način i može se koristiti iz nekog drugog programa. Razvoj zasnovan na web servisima, zajedno s razvojem zasnovanom na komponentama, predstavlja današnji *state-of-the-art* ponovne upotrebe softvera koja se bavi pitanjem kako već razvijeni softver ponovo upotrijebiti za neku drugu svrhu ili za neke druge korisnike. Web servisi omogućuju posebnu vrstu arhitekture distribuiranih sustava, takozvanu arhitekturu usmjerenu na servise ili servisno orijentiranu arhitekturu (SOA), prema kojoj se sustav gradi povezivanjem samostalnih servisa koji se izvršavaju na geografski udaljenim računalima. Razvoj tehnologije web servisa bio je popraćen razvojem odgovarajućih standarda koji osiguravaju međusobnu kompatibilnost servisa. Međutim, sadašnji standardi za web servise ponekad se kritiziraju da su preglomazni, preopćeniti i neefikasni pa su neke kompanije počele koristiti nestandardnu, ali jednostavniju verziju web servisa, takozvane RESTful web servise. U softverskom inženjerstvu zasnovanom na web servisima razlikujemo dvije vrste softverskih procesa. Jedna vrsta je razvoj samih web servisa koje će netko drugi upotrebljavati, a druga je razvoj aplikacija korištenjem postojećih web servisa.

Summary

Web service is a standard representation for some computational or information resource on the Web that can be used by other programs. Service-based development, along with component-based development, is state-of-the-art in software reuse where the development process is geared to reusing existing software for other purposes or for other users. Service-oriented architecture (SOA) is a way of developing distributed systems where the system components are stand-alone services, executing on geographically distributed computers. To provide direct support for the implementation of web service compositions, several web service standards have been developed. However, current web services standards have been criticized as being "heavyweight" standards that are over-general and inefficient. For this reason, some organizations use a simpler, more efficient approach to service communication using so-called RESTful services. There are two kinds of software processes in service-based software engineering – service engineering and software development with services. Service engineering is the process of developing services for reuse in service-oriented applications. On the other hand, development of software using services is based on the idea that you compose and configure services to create new, composite services.

Životopis

Lana Arzon rođena je 27. lipnja 1990. godine u Varaždinu. Školovanje započinje 1997. godine u Osnovnoj školi Antuna i Ivana Kukuljevića u Varaždinskim Toplicama, a nastavlja u Prvoj gimnaziji Varaždin koju završava 2009. godine. Iste godine upisuje preddiplomski sveučilišni studij Matematika na Matematičkom odsjeku Prirodoslovno–matematičkog fakulteta u Zagrebu. Godine 2012. dobiva titulu sveučilišne prvostupnice matematike te upisuje diplomski sveučilišni studij Računarstvo i matematika. Na zadnjoj godini studija počinje se baviti razvojem i održavanjem web aplikacija te zbog toga odlučuje istraživati i razvoj aplikacija pomoću web servisa koji postaje tema njenog diplomskog rada.