

Univerzitet u Beogradu

Matematički fakultet

Seminarski rad

predmet: Metodika stručnog i naučnog rada

SOAP i REST mrežni servisi

Student:

Perica Trajkov 1023/2012

Profesor:

dr. Vladimir Filipović

1. UVOD

Mrežni servis (web service) predstavlja način komunikacije između dva elektronska uređaja na mreži. Ova komunikacija se obično vrši tako što klijent šalje zahtev pomoću mrežnog servisa serveru i rezultujuća informacija se vraća klijentu u nekom obliku. Klijent može biti i neki server. Danas se koriste dva dominantna mrežna servisa SOAP i REST (RESTful). U nastavku će biti reči o oba.

2. SOAP

SOAP (Simple Object Access Protocol), predstavlja način da neki program, koji se pokreće na jednom operativnom sistemu, komunicira sa drugim programom, koji se pokreće na istom ili drugom operativnom sistemu. Ovo se rešava korišćenjem HTTP-a (Hypertext Transfer Protocol) i XML-a (Extensible Markup Language) koji služi kao mehanizam za razmenu informacija. HTML i XML obezbeđuju priručno rešenje za problem komunikacije dva programa na mreži na istim ili različitim operativnim sistemima. SOAP tačno specifikira kako će izgledati HTTP zaglavlje, kao i XML fajl, da bi se informacija mogla razmeniti između dva programa.

SOAP je nastao 1998 kao koprodukt Microsoft-a, DevelopMentor-a i Userland Software-a. Njegovi tvorci su Dejav Viner, Don Boks, Bob Atkinson i Mohsen Al Goseina. Predložen je za standardni interfejs od strane IETF-a. Sličan je donekle sa IIOP protokolom (Internet Inter-ORB Protocol), koji je deo CORBA-e. Još jedan protokol koji mu je sličan je RMI protokol (Remote Method Invocation) Sun Microsystems-a, koji predstavlja protokol za programe pisane u Javi.

Prednost SOAP-a je što je korišćenjem njega mnogo verovatnije da će pozivi programa biti dozvoljeni od strane serverskog firewall-a. S obzirom na to da su HTTP zahtevi obično dozvoljeni preko firewall-a, sledi da će i programi koji koriste SOAP moći neometano da komuniciraju sa drugim programima.

Postoji nekoliko argumenata koji govore u korist korišćenja SOAP. Neki od njih su:

- Korišćenjem SOAP-a omogućava se lakša komunikacija preko proksija i firewall-a

- SOAP je dovoljno svestran da omogući korišćenje različitih transportnih protokola. Obično je to HTTP, ali može da bude i neki drugi transportni protokol (SMTP, RSS)
- SOAP je platformski nezavisan
- SOAP je nezavisan od programskog jezika
- SOAP je jednostavan i nadogradiv

Međutim postoje i neki argumenti koji govore o manama korišćenja SOAP. Jedan od njih je da zbog korišćenja XML-a SOAP može biti značajno sporiji od konkurentnih rešenja.

SOAP poruka predstavlja zapravo validan XML dokument koji se sastoji od sledećih elemenata:

- Element koverta (Envelope element), koji identifikuje XML dokument kao SOAP poruku
- Element zaglavlja (Header element), koji sadrži neke informacije zaglavlja
- Element tela (Body element), koji sadrži informacije o pozivu i odgovoru
- Element za greške (Fault element), koji sadrži informacije o greškama i status greške

Postoje neka pravila kojih se moramo pridržavati prilikom pisanja SOAP poruka:

- SOAP poruka mora biti enkodirana korišćenjem XML-a
- SOAP poruka mora koristiti imenski prostor Envelope elementa
- SOAP poruka ne sme sadržati DTD referencu
- SOAP poruka ne sme sadržati instrukcije za procesiranje XML-a

U nastavku je prikazan skeleton neke SOAP poruke:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Header>
    ...
  </soap:Header>

  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>

</soap:Envelope>
```

Element koverta je koreni element SOAP poruke. To nam govori `soap:Envelope` tag. Podešavanjem parametra `xmlns:soap` definiše se imenski prostor koji će biti korišćen. `soap:encodingStyle` parametar služi za definisanje tipova podataka koji će biti korišćeni. Ne postoji podrazumevano enkodiranje SOAP poruke, pa se ono mora navesti uvek.

Element zaglavlja je opcion i sadrži informacije o SOAP poruci koje su zavisne od aplikacije. Ukoliko ovaj element postoji on mora biti prvi dete-element. Primer elementa zaglavlja je dat u nastavku:

```
<soap:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
    soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
```

SOAP definiše tri atributa u podrazumevanom imenskom prostoru. To su mustUnderstand, actor i encodingStyle. Oni definišu kako primalac treba da procesira SOAP poruku. mustUnderstand atribut se koristi da označi da li je unos zaglavlja obavezan ili proizvoljan. Atribut actor služi da adresira element zaglavlja određenoj međutački na putu poruke do krajnje tačke. Atribut encodingStyle služi da definiše tipove podataka koji će se koristiti u dokumentu.

Element tela sadrži pravu SOAP poruku koja je namenjena krajnjoj tački.

Element za greške je opcion i služi da prikaže poruke o greškama. I on se sastoji od podelemenata (faultcode, faultstring, faultactor, detail).

Povezivanje HTML i SOAP-a se vreši na sledeći način:

```
POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
```

3. REST

REST (REpresentational State Transfer) arhitekturni stil je razvijen od strane Technical Architecture Group (TAG) paralelno sa HTTP 1.1. Ovaj stil je dobio najveću primenu na World Wide Web-u. REST arhitektura se tipično sastoji od klijenta i servera. Klijent inicira zahtev serveru, server procesira zahtev i vraća odgovor klijentu. Glavni ciljevi REST-a su:

- skalabilnost
- generalizovanost interfejsa
- nezavisni razvoj komponenata

REST definiše skup arhitekturnih principa. Pomoću njih možemo dizajnirati mrežne servise koji se fokusiraju na sistemske resurse, uključujući i to kako se stanja resursa

adresiraju i prebacuju preko HTTP-a od strane velikog broja klijentskih aplikacija pisanih u različitim programskim jezicima. Danas se REST izdvojio kao dominantan mrežni servis dizajn model, ukoliko se meri po broju mrežnih servisa koji ga koriste. REST ima toliko jak uticaj na Web da je veoma često potisnuo SOAP i WSDL jer je značajno jednostavniji za korišćenje.

REST nije privukao mnogo pažnje u trenutku kada je predstavljen 2000. godine od strane Roja Fieldinga sa Univerziteta Kalifornija u njegovoj akademskoj disertaciji [2]. Sada, nakon nekog vremena, REST privlači veliku pažnju računarske javnosti.

U svojoj najjednostavnijoj formi, konkretna implementacija REST Web servisa prati četiri osnovna principa:

- eksplicitno korišćenje HTTP metoda
- nema korišćenja stanja
- izlaganje URI-ja koji su nalik strukturi direktorijuma
- transfer preko XML-a, JSON-a (JavaScript Object Notation)

U nastavku će malo detaljnije biti opisan svaki od ovih principa.

3.1. Eksplicitno korišćenje HTTP metoda

Jedna od glavnih karakteristika RESTful web servisa predstavlja eksplicitno korišćenje HTTP metoda na način koji prati protokol definisan kao RFC 2616. Na primer, HTTP GET metod je namenjen za korišćenje od strane klijentske aplikacije za preuzimanje resursa, podataka sa Web servera ili da izvršava upit uz očekivanje da će mu mrežni server odgovoriti odgovarajućim rezultatom.

REST zahteva od razvijaoa da eksplicitno koriste HTTP metode na način koji je saglasan sa definicijom protokola. Osnovni REST dizajn princip uspostavlja 1-1 mapiranje između CRUD operacije (kreiranje, čitanje, modifikovanje, brisanje) i odgovarajućih HTTP metoda. Tako, prema ovom mapiranju važi:

- za kreiranje resursa na serveru koristiti POST
- za preuzimanje resursa koristiti GET
- za menjanje stanja resursu ili njegovu modifikaciju koristiti PUT
- za uklanjanje resursa koristiti DELETE

Postoji mnogo slučajeva kada se u mrežnom API-ju na nekorektan način koriste HTTP metodi. Na primer, često se koristi GET metod da se izazove neka transakcija na serveru (na primer dodavanje podataka u bazu), iako mu to nije namena, bar ne po standardu HTTP 1.1-RFC. Ovakva, nepravilna, primena metoda se ne dozvoljava u REST-u. Kad mrežni API koristi GET da pozove udaljene procedure to izgleda na primer ovako:

```
GET /dodajkorisnika?name=Janko HTTP/1.1
```

Ovo nije veoma prihvatljiv dizajn jer se GET koristi za menjanje nekog stanja na serveru. Dakle, ovaj metod ima neželjena dejstva. Drugi problem sa GET metodom je što se on može koristiti i za brisanje ili izmenu podataka u bazi, odnosno da promeni stanje servera na neki način. Jednostavan način da se prevaziđe ovaj problem je premeštanje imena parametara i vrednosti iz URI-ja zahteva u XML tagove. Rezultujući tagovi mogu biti poslani kao telo HTTP POST metoda:

```
POST /korisnici HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Janko</name>
</user>
```

Metod naveden iznad, predstavlja primer REST zahteva: pravilnu upotrebu POST metoda i uključanje zahteva u telo POST metoda. Na drugom kraju, kod prvog zahteva, zahtev može biti procesiran dodavanjem resursa telu zahteva. U ovom slučaju novi resurs treba da se doda kao dete od /korisnici. Ovaj odnos otac-dete izmenjuje novog entiteta i njegovog roditelja analogan je odnosu datoteke i njegovog roditeljskog direktorijuma. Klijent određuje odnos između entiteta i njegovog roditelja i definiše URI novog entiteta u POST zahtevu.

Klijentska aplikacija onda može dobiti reprezentaciju resursa korišćenjem novog URI-ja, znajući da se novi resurs logički nalazi unutar /korisnici:

```
GET /korisnici/Janko HTTP/1.1
Host: myserver
Accept: application/xml
```

Korišćenje GET-a na ovaj način je eksplicitno jer GET služi samo za pribavljanje podataka. GET je operacija koja treba biti oslobođena sporednih efekata odnosno idempotencije. Slično, refaktorisanje Web metoda je neophodno u slučaju kada se GET koristi za ažuriranje podataka.

Generalno, potrebno je pratiti smernice REST-a koje kažu da se HTTP metodi eksplicitno koriste i to korišćenjem “imenica” umesto “glagola” unutar URI-ja. U RESTful Web servisu “glagoli” POST, GET, PUT i DELETE su predefinisani protokolom. Idealno, da bi interfejs bio generalizovan, i da bi se klijentima dozvolilo da budu eksplicitni sa operacijama koje pozivaju, Web servis ne sme da definiše više “glagola” poput /dodajkorisnika.

3.2. Nema korišćenja stanja

Rest Web servisi moraju biti skalabilni kako bi podržali stalno rastuće zahteve performansi. Zahtevi koji se šalju se obično prosleđuju od jednog do drugog servera, po potrebi, kako bi se smanjilo ukupno vreme odziva. Korišćenje posredničkih servera zahteva da klijenti REST mrežnog servisa šalju zahteve koji su kompletni i nezavisni, tj. zahteve koji sadrže sve podatke koji su neophodni za dobijanje odgovora kako bi posrednički serveri mogli da prosleđuju i balansiraju opterećenja bez lokalnog čuvanja stanja između različitih zahteva.

Kompletnan, nezavisan zahtev ne zahteva da server, dok procesira zahtev, dobija neke informacije o stanju zahteva. Aplikacija REST mrežnog servisa, odnosno klijent, uključuje unutar HTTP zaglavlja i tela zahteva, sve parametre i podatke koji su neophodni serverskoj strani za generisanje odgovora. Ukoliko se ne koriste stanja, poboljšavaju se performanse i pojednostavljuje dizajn i implementacija serverskih komponenti.

3.3. URI je nalik strukturi direktorijuma

Sa stanovišta klijentske aplikacije adresiranje resursa preko URI-ja određuje koliko će biti intuitivan REST mrežni servis.

URI REST mrežnog servisa treba biti intuitivan toliko da je lak za pogađanje. URI treba da bude takav da se na osnovu njegovog sadržaja može znati na šta on pokazuje. Dakle, struktura URI-ja treba da bude jednostavna, predvidiva i lako razumljiva.

Jedan način da se ovo postigne je da se URI dizajnira tako da imitira strukturu direktorijuma. Ovakav tip URI-ja je hijerarhijski, sa jednim čvorom koji se onda grana na druge čvorove.

Neki od saveta koji se mogu primeniti da bi URI bio što intuitivniji:

- sakriti ekstenzije skriptova na serverskoj strani
- pisati sve malim slovima
- zameniti razmake sa podvlakama ili crticama (ali bez njihovog mešanja)
- izbegavati stringove sa upitima

Još jedna važna karakteristika URI-ja je da trebaju biti statični, tj. kada se resurs promeni, ili se promeni implementacija servisa, URI mora ostati isti.

3.4. transfer preko XML-a, JSON-a (JavaScript Object Notation)

Reprezentacija resursa tipično odražava trenutno stanje resursa i njegovih atributa u trenutku kada ga klijentska strana zahteva. Dakle, ovaj poslednji princip REST-a se odnosi na format podataka koje aplikacija i servis razmenjuju pri zahtevu i odgovoru.

Objekti unutar modela podataka su obično povezani na neki način, i veze između objekata modela podataka (resursi) treba da se ogledaju u načinu na koji su oni predstavljeni za transfer klijentskoj aplikaciji.

Da bi klijentskoj aplikaciji pružili mogućnost da zahteva specifičan tip sadržaja, servis treba da koristi HTTP Accept zaglavlje, gde je vrednost zaglavlja MIME tip. Neki od korišćenijih MIME tipova su JSON, XML, XHTML

4. Zaključak - Poređenje između SOAP i RESTful veb protokola

Postoji višegodišnja prepirka oko toga koji je od ovih protokola bolji. Vilijam Brogden [1] govori o tome kako poređenje ova dva i nije baš adekvatno. REST je više stil nego standard. Giganti poput Google-a i eBay-a koriste REST servise. Oba pristupa rade svoj posao, imaju svoje prednosti i mane, tako da je sve do razvijaoca koji će pristup koristiti. Većina razvijaoca se susrela, barem delimično, sa REST pristupom, koji koristi standardni URI za slanje zahteva (npr. <https://www.mycompany.com/program/method?Parameters=xx>). Ovaj pristup je veoma jednostavan za razumevanje i može biti izvršen na bilo kom klijentu ili serveru koji ima HTTP/HTTPS podršku. Komanda može da se izvrši korišćenjem standardnog HTTP GET metoda. Podržavaoci ovog pristupa navode jednostavnost, korišćenje postojeće mrežne infrastrukture i brzo savladavanje tehnike kao njegove glavne prednosti.

S druge strane SOAP, praotac svih interfejsa mrežnih servisa, se poboljšava iz dana u dan, i sa pojavom SOAP 1.2 ispravljene su mnoge mane ove tehnologije, uključujući i jednostavnost korišćenja. SOAP 1.2 ima neke dodatne mogućnosti koje ne postoje u REST-u. Jedna od prednost SOAP-a je korišćenje „generičkog“ transporta. Dok REST koristi HTTP/HTTPS, SOAP može da koristi bilo koji transportni protokol za slanje zahteva (uključujući SMTP, JMS i druge). Sa druge strane jedna od mana je i prevelika „opširnost“ pri korišćenju XML-a, kao i potrebno vreme da bi se on parsirao.

Dobra stvar kod oba mrežna servisa je što su održivi na današnjem tržištu. I REST i SOAP mogu da reše veliki broj mrežnih problema i izazova.

Obe tehnologiju mogu da se kombinuju. REST je veoma lak da se razume i veoma je pristupačan, ali mana mu je nedostatak standarda i to što se smatra za „arhitekturni pristup“. S druge strane SOAP je industrijski standard sa dobro definisanim protokolom, i skupom dobro zasnovanih pravila, pa se može koristiti i na malim i na velikim sistemima.

Oblasti u kojima se REST ponaša vrlo dobro su sledeće:

- kod ograničenog propusnog opsega i resursa (povratna informacija može biti u bilo kom obliku)
- kod operacija koje ne koriste stanja (ukoliko neka operacija treba da bude nastavljena onda REST nije pravi pristup i SOAP verovatno predstavlja bolje rešenje)
- kod situacija gde je moguće keširanje (ukoliko informacija može biti keširana zbog operacija koje ne koriste stanja onda je ovaj pristup odličan)

Oblasti u kojima se SOAP ponaša vrlo dobro su sledeće:

- asinhrono procesiranje (ako aplikacija treba da ima garantovani nivo pouzdanosti i bezbednosti onda SOAP 1.2 nudi dodatne standarde za ovakve uslove)
- formalni ugovori (ukoliko obe strane trebaju da se slože oko formata za razmenu informacija onda je SOAP 1.2 pravi pristup)
- kod operacija koje koriste stanja

Kao što je prikazano u seminarskom radu, oba pristupa imaju svoju primenu. Većina analitičara se, međutim, slaže da će REST i SOAP koegzistirati u budućnosti.

5. Literatura

1. <http://searchsoa.techtarget.com/tip/REST-versus-SOAP-the-SOAP-story>
2. Roy Fielding "[Architectural Styles and the Design of Network-based Software Architectures.](#)"
3. <http://www.infoq.com/articles/rest-soap-when-to-use-each>
4. <http://www.ibm.com/developerworks/webservices/library/ws-restful/>
5. <http://www.w3schools.com/soap/>