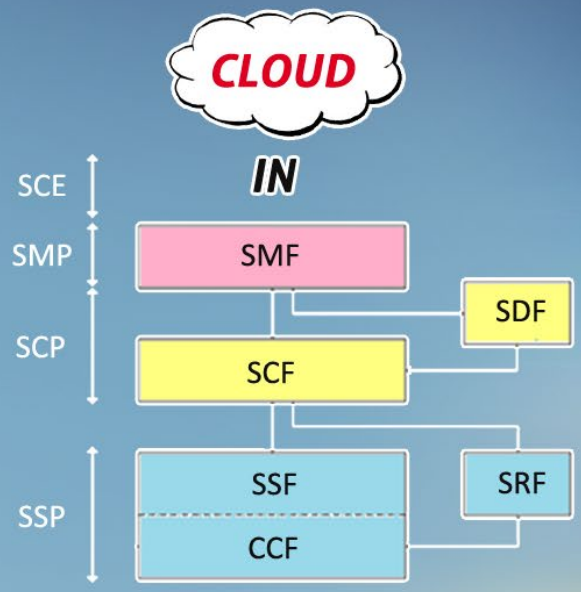
REŢELE INTELIGENTE DE TELECOMUNICAŢII



Dan Nicolae ROBU
Cosmin COSTACHE
- editori -

EDITURA UNIVERSITĂŢII TRANSILVANIA DIN BRAŞOV

D.N.Robu **C.Costache**

- editori -

REŢELE INTELIGENTE DE TELECOMUNICAŢII

Editura TRANSILVANIA

ISBN: 978-606-19-0820-2

Braşov - 2016

Cuprins

Pag.

1 Cuvânt înainte - *D.Robu, C.Costache*

4 Contextul Reţelelor Inteligente de Telecomunicaţii - *V.Cazacu, D.Robu, T.Bălan*

59 Modelul de Procesare a Apelurilor - *S.Bâlbâe, F.Sandu, T.Bălan*

85 Modelul de Controler al Reţelei Inteligente - *I.Rîpea, F.Sandu*

99 Servicii Tipice în Reţelele Inteligente - Implementare - *V.Cazacu, D.Robu*

127 Software pentru Crearea de Servicii - *C.Costache, V.Cazacu, G.Stelea*

135 JAIN - Java for Inteligent Networks - *S.Dovăncescu*

182 Mediul de Creare a Serviciilor pentru Retele Inteligente de Telecomunicatii - *R.David, C.Costache, F.Sandu*

249 Implementarea apelurilor cu extensia IBM Rational pentru SIP - *F.Sandu*

266 Comunicații Cloud - Abordarea orientată pe servicii - *C.Costache, G.Stelea, D.Robu*

314 WebRTC - Comunicații prin web în timp real - *G.Stelea, Ș.Șurariu, T.Bălan*

364 Managementul Business al Comunicațiilor - *A.Balica, F.Sandu*

435 Bibliografie

447 Lista abrevierilor

COMUNICAȚII CLOUD

Folosirea paradigmei orientării pe servicii reprezintă o direcție inovatoare și actuală în eforturile de a dezvolta noi tehnologii pentru integrarea resurselor de calcul în sistemele eterogene distribuite în contextul Cloud computing.

Abordarea orientată pe servicii

C.Costache, G.Stelea, D.Robu

Noțiunile de serviciu, e-serviciu sau serviciu web sunt utilizate frecvent pentru a denumi o componentă software autonomă care este identificată în mod unic si care poate fi accesată folosind protocoale standard Internet cum SOAP sau HTTP sau cele bazate pe XML. Serviciile web au simplificat intr-o foarte mare măsura procesul de integrare al numeroaselor componente software și au facilitat orientarea către o arhitectură orientată pe servicii.

Conceptul de serviciu web a apărut in urma necesității de a interconecta diversele aplicații care rulează la nivel de organizație printr-o modalitate rapida si flexibila.
Exista diferite definiți pentru serviciile web, enunțate de diverse organizații. Conform acestora, serviciile web pot fi definite astfel:

- Un serviciu web reprezintă orice componentă software cu o funcționalitate bine definită, independentă de limbaj și de platformă, care este publicată, și accesibilă prin Internet și care folosește un sistem standardizat de comunicație bazat pe mesaje XML. Această componentă software este accesibilă folosind protocoale standard Internet get HTTP și poate fi apelată programatic.

- Serviciile web sunt aplicații independente, modulare și distribuite care pot fi descrise, publicate, localizate și apelate într-o rețea pentru a crea noi aplicații sau procese.

- Serviciile web sunt sisteme de transfer a informației în format XML, care folosesc Internetul pentru o interacțiune directă aplicație-la-aplicație sau mașina-la-mașina (M2M).

- Un serviciu web este o colecție de protocoale și standarde folosite pentru a facilita schimbul de informații între diverse aplicații sau sisteme. Aplicațiile software scrise in diferite limbaje de programare și executate pe platforme diferite pot folosi serviciile web pentru a schimba date în cadrul unei rețele gen Internet. Acest schimb de date poate fi comparat cu comunicația inter-proces pe un singur computer.

Scopul principal urmărit prin folosirea serviciilor web este interoperabilitatea. Aceasta permite ca sisteme sau aplicații eterogene, rulând pe platforme diferite, sa poată comunica intre ele intr-un mod standardizat. Interacțiunea dintre consumatori (clienți) și furnizorul de serviciu se realizează printr-un schimb de mesaje în format standardizat. Din acest motiv detaliile de implementare si platforma pe care rulează serviciul, nu sunt relevante pentru consumatorii serviciului web.

Se pot evidenția următoarele caracteristici principale ale serviciilor web:

- Sunt bazate pe XML. Serviciile web folosesc XML pentru reprezentarea si transportul datelor. Prin folosirea acestui limbaj standard se elimina orice dependență față de un anumit limbaj de programare, sistem de operate sau platformă.

- Eliminarea interdependențelor. Consumatorul unui serviciu web nu este afectat de modificări in implementarea serviciului web, lucru care permite o integrare mai simpla a sistemelor. Furnizorii de servicii au posibilitatea de a actualiza sau modifica implementarea serviciilor web fără a afecta consumatorii respectivului serviciu.

- Posibilitatea de comunicare sincronă sau asincronă. In cazul unei comunicații sincrone între consumator și serviciul web, clientul oprește execuția și așteaptă primirea răspunsului pentru cererea efectuată. Modul asincron de comunicare ii permite clientului săapeleze serviciul web și să își continue execuția. In modul asincron răspunsul de la serviciul web pentru cerere efectuata este trimis ulterior către client.

- Sunt suportate apeluri de proceduri la distanta (RPC-Remote Procedure Call). Serviciile web permit clienților săapeleze proceduri, funcții sau metode pe obiecte aflate la distanță folosind un protocol bazat pe XML (XMLP).

Serviciile web pot fi grupate în două categorii principale : servicii bazate pe protocolul SOAP și servicii de tip REST

Standardele care definesc serviciile web

Autoritatea care reglementează standardele legate de serviciile web este organizația World Wide Web Consortium (W3C). Pe lângă protocoalele de baza folosite pentru

serviciile web cum ar fi WSDL, SOAP sau UDDI, organizația W3C a mai definit o serie de standarde ce iau in considerare aspecte precum securitatea serviciilor web, managementul, tranzacțiile sau parametrii QoS. Toate aceste standarde se bazează pe XML și folosesc XML Schema pentru a limita formatul datelor în cadrul documentelor XML.

În continuare vor fi prezentate principalele standarde și tehnologii definite pentru serviciile web. Vor fi tratate în detaliu protocoalele de bază pentru servicii web, cum ar fi WSDL, SOAP si UDDI.

WSDL este un limbaj bazat pe XML ce oferă un model de descriere a serviciilor web. Un document WSDL descrie interfața oferită de către un serviciu web și modul în care o aplicație client poate interacționa cu serviciul web.
Versiunea curentă a standardului WSDL este 2.0 și aduce modificări substanțiale față de versiunea anterioara WSDL 1.1, prin adăugarea de specificații pentru serviciile web de tip REST.

Un consumator (client) care vrea să folosească un serviciu web, pe baza fișierului WSDL poate determina operațiile (funcțiile) puse la dispoziție de către serviciul web și felul în care pot fi apelate acestea. În cazul în care serviciul web specifică tipuri speciale de date, aceste tipuri sunt incluse in fișierul WSDL sub forma unor definiți XSD (XML Schema). Astfel fișierul WSDL conține toate informațiile necesare unui consumator pentru a putea accesa operațiile puse la dispoziție de către serviciul web.

Un fișier WSDL conține următoarele elemente de baza:

- `<types>` : elementul `<types>` conține definiția tipurilor de date care sunt folosite in mesajele care sunt schimbate de către serviciul web. Aceste tipuri de date sunt definite folosind standardul XSD (XML Schema Definition)

- `<message>` : un element de tip <message> definește nu mesaj care poate fi schimbat de către serviciul web si clienții acestuia

- `<portType>` : un element de tip `<portType>` descrie setul de operații suportat de către serviciul web. Fiecare sub-element de tip `<operation>` definește o

operație si furnizează numele operației precum si mesajele de intrare si ieșire asociate cu operația.

- <binding> : elementul de tip <binding> descrie modul de acces la un serviciu web si anume protocolul folosit in schimbul de mesaje.

- <service> : Elementul de tip <service> precizează adresa (URL) unde poate fi accesat serviciul web.

In următorul exemplu sunt evidențiate toate elementele de baza ale unui fișier WSDL

```
<definitions name="HelloService"
   targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<message name="SayHelloRequest">
<part name="firstName" type="xsd:string"/>
</message>
<message name="SayHelloResponse">
<part name="greeting" type="xsd:string"/>
</message>

<portType name="Hello_PortType">
<operation name="sayHello">
       <input message="tns:SayHelloRequest"/>
       <output message="tns:SayHelloResponse"/>
</operation>
</portType>

<binding name="Hello_Binding" type="tns:Hello_PortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="sayHello">
       <soap:operation soapAction="sayHello"/>
       <input>
       <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:examples:helloservice" use="encoded"/>
       </input>
       <output>
       <soap:bodyencodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:examples:helloservice" use="encoded"/>
       </output>
</operation>
</binding>

<service name="Hello_Service">
<port binding="tns:Hello_Binding" name="Hello_Port">
       <soap:address location="http://www.examples.com/SayHello/">
</port>
</service>

</definitions>
```

Elementul XML principal intr-un fișier WSDL este elementul <defintions>. Acesta este un container pentru celelalte elemente definite de standardul WSDL și specifică numele serviciului web la care se referă (exemplu: <definitions name="HelloService" />)

Un serviciu web poate defini una sau mai multe interfețe (portType) iar fiecare interfață poate avea una sau mai multe operații. Fiecare operație are asociat un mesaj de intrare (input)și un mesaj de răspuns (output). Formatul acestor mesaje este definit de către protocolul SOAP.

SOAP este un protocol de comunicație simplu bazat pe XML, folosit pentru transferul de informație intr-un mediu distribuit. Protocolul SOAP a fost printre primele standarde definite pentru serviciile web, el fiind in continuare cel mai important standard si totodată cel mai răspândit. Mesajele SOAP pot fi expediate folosind orice protocol de transport cum ar fi HTTP (cel mai frecvent), SMTP, JMS.

Un mesaj SOAP este un document XML care poate conține următoarele elemente:

- `<Envelope>` : acesta este un element obligatoriu si delimitează începutul si finalul mesajului
- `<Header>` : element opțional care conține informații suplimentare (meta-date) despre mesaj
- `<Body>` : in cadrul acestui element este conținuta informația propriu-zisa a mesajului (numita si payload sau încărcătura)
- `<Fault>` : element opțional care furnizează informații despre eventualele erori apărute in timpul procesării cererii de către serviciul web.

Toate a ceste elemente sunt declarate in namespace standard definit pentru SOAP și anume http://www.w3.org/2001/12/soap-envelope.

Structura unui mesaj SOAP este prezentată in exemplul următor:

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap= "http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
       <!-- elemente de antet -->
</soap:Header>
<soap:Body>
       <!-- conținutul mesajului SOAP -->
</soap:Body>
</soap:Envelope>
```

Elementul <Envelope> are rolul de a delimita un mesaj SOAP astfel încât entitatea care recepționează mesajul sa poată determina daca mesajul a fost recepționat in întregime.

Următoarele proprietăți pot fi evidențiate despre acest element:

- fiecare mesaj SOAP are un element părinte de tip <Envelope>
- acest element este obligatoriu pentru toate mesajele de tip SOAP
- fiecare element <Envelope> conține doar un singur element de tip Body
- poate conține doar un element de tip <Header>. In cazul in care conține un astfel de element, elementul <Header> trebuie sa fie pe prima pozitie (primul copil)
- elementul <Envelope> specifica versiunea protocolului SOAP care este folosita

Elementul <Header> oferă un mecanism flexibil pentru a include informații adiționale unui mesaj SOAP. Astfel elementul <Header> poate cuprinde orice număr de elemente care conțin informații de control ce pot fi folosite in etapa de prelucrare a informației cuprinsa in mesaj. De exemplu un astfel de element poate conține date pentru autentificarea expeditorului mesajului. Fiecare din aceste elemente poate avea definit atributul „mustUnderstand” cu valorile posibile 1 sau 0. In cazul in care valoare atributului este setata pe 1, entitatea care recepționează mesajul trebuie sa tina cont de informațiile cuprinse in elementul care conține acest atribut.

Un alt element obligatoriu in cadrul unui mesaj SOAP este elementul <Body>. Acest element conține informația propriu-zisa a mesajului. Pentru reprezentarea informației se folosește formatul XML. In structura mesajului, elementul <Body> apare imediat după elementul <Header> in cadrul elementului părinte de tip <Envelope>.

Un exemplu de mesaj SOAP în care este definită informația conținuta in elementul <Body> este prezentat in continuare:

```
<soap:Envelope xmlns:soap= "http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
        <!--Header blocks-->
</soap:Header>
<soap:Body>
        <m:GetQuotation xmlns:m="http://www.tp.com/Quotation">
                <m:Item>Computers</m:Item>
        </m:GetQuotation>
</soap:Body>
</soap:Envelope>
```

Acest exemplu prezinta un mesaj SOAP care conține un apel de metoda in format RPC (Remote Procedure Call). In cadrul elementului <Body> este specificat numele metodei care urmează să fie apelata precum și parametrii necesari pentru efectuarea apelului.

In cazul in care apar erori in timpul procesării unui mesaj primit de către serviciul web, mesajul SOAP de răspuns trimis clientului, va conține un element de tip <Fault> care specifica informații referitoare la cauza care a dus la apariția erorii. Elementul de tip <Fault> este inclus in cadrul elementului <Body>. Un mesaj SOAP poate avea doar un singur element de tip <Fault>

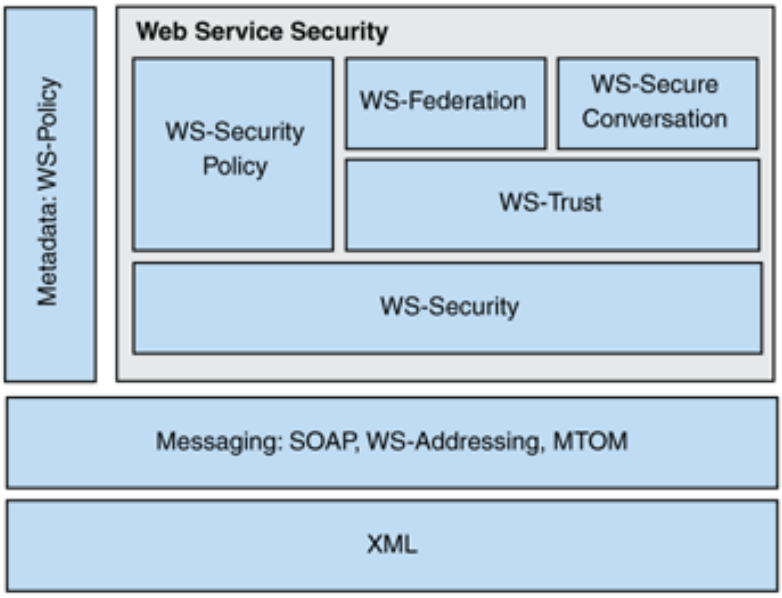
Elementul <Fault> definește următoarele sub-elemente:

- <faultCode> : acest element conține un cod care indica o clasa de erori. Aceste coduri sunt predefinite.
- <faultString> : elementul va conține un text care descrie cauza erorii
- <faultActor> : conține un text care indica sistemul sau componenta care a cauzat eroarea
- <detail> : acest element conține informații suplimentare despre eroarea apăruta

În următorul exemplu este prezentat un mesaj SOAP de răspuns care conține un element de tip <Fault>

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap= "http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
       <soap:Fault>
              <faultcode xsi:type="xsd:string">soap:Client</faultcode>
              <faultstring xsi:type="xsd:string">
                     Failed to locate method (ValidateCreditCard) in class
                     (examplesCreditCard) at /usr/local/ActivePerl-5.6/lib/
                            site_perl/5.6.0/SOAP/Lite.pm line 1555
              </faultstring>
       </soap:Fault>
</soap:Body>
</soap:Envelope>
```

Pe lângă standardele de baza pentru servicii web, în timp au fost definite de către organizația W3C si alte standarde care tratează aspecte cum ar fi securitatea mesajelor SOAP, modalitățile de adresare, publicarea politicilor unui serviciu, compunerea si orchestrarea serviciilor web. In continuare vor fi prezentate pe scurt cele mai importante dintre aceste standarde



Standardele definite de către W3C pentru serviciile web (sursa w3c.org)

WS-Addressing (Web Services Addressing)

Aceasta specificație definește un mod standard pentru inserarea de informații pentru rutarea mesajelor SOAP in cadrul elementului <Header>.

WS-Policy

WS-Policy este o specificație care permite serviciilor de web sa folosească XML pentru a face publice politicile lor referitor la securitate, quality of service, etc. De asemenea aceasta specificație permite clienților serviciilor web sa specifice cerințele lor referitor la aceste politici.

WS-Security (Web Services Security sau WSS)

WS-Security este o extensie a protocolului SOAP care are ca scop securizarea serviciilor web. Protocolul specifica modul in care integritatea si confidențialitatea pot fi aplicate pentru mesajele SOAP. In acest scop protocolul permite trimiterea informațiilor de autentificare sub forma de jeton (token) de securitate, fiind suportate multiple formate cum ar fi SAML (Security Assertion Markup Language), Kerberos sau X.509. Obiectivul principal este utilizarea semnăturii XML (XML Signature) si criptarea XML (XML Encryption) pentru a furniza securitate end-to-end.

Specificația WS-Security descrie 3 mecanisme principale pentru asigurarea securității mesajelor:
- modul in care pot fi semnate mesajele SOAP pentru a asigura integritatea informațiilor.
- criptarea mesajelor pentru a asigura confidențialitatea datelor
- atașarea informațiilor de autentificare sub forma de tokens de securitate

WS-Reliable Messaging (WS-RM)

Standardul WS-RM sau Web Services Reliable Messaging reprezintă un standard care permite ca mesajele SOAP sa fie livrate la destinație chiar și în cazul in care rețeaua de comunicație sau anumite componente sunt indisponibile.

WS-BPEL si WS-CDL(Choreography Description Language)

Aceste standarde permit compunerea și orchestrarea serviciilor web. Posibilitățile de orchestrare a serviciilor web folosind BPEL sunt descrise detaliat în secțiunea dedicată arhitecturilor orientate pe servicii.

In dezvoltarea serviciilor web bazate pe SOAP exista două abordări principale:

- Dezvoltarea top-down. Aceasta abordare presupune întâi definirea interfeței serviciului web, adică a fișierului WSDL, apoi urmând a fi făcută implementarea propriu zisa. Aceasta metoda de dezvoltare poarta denumirea top-down deoarece se începe de la un nivel abstract – interfața serviciului – care apoi este implementata.

- Bottom-up. Acest al doilea model de dezvoltare presupune ca produsul software este deja implementat si se adaugă doar un nivel suplimentar care expune funcționalitatea software ca serviciu web. Metoda poarta denumirea bottom-up deoarece se pornește de la implementarea deja existenta pentru care se definește apoi interfața serviciului web.

Doua dintre cele mai importante platforme de programare, .NET și Java, oferă soluții pentru implementarea serviciilor Web. Tehnologia Windows Communication Foundation (WCF) a fost introdusa in .NET 3.0 cu scopul de a permite construirea de aplicații distribuite bazate pe principiile SOA.
La rândul ei, tehnologia Java oferă un set de tehnologii pentru dezvoltarea serviciilor Web: Java API for XML Processing (JAXP), Java API for XML Registries (JAXR), Java API for XML Web Services (JAX-WS), Java Architecture for XML Binding (JAXB), SOAP with Attachments API for Java (SAAJ).

Serviciile web de tip REST

Serviciile web de tip REST au apărut ca o alternativa la serviciile web bazate pe SOAP. Termenul REST este o abreviere pentru Representational State Transfer si a fost folosit pentru prima dată în lucrarea de doctorat "Architectural Styles and the Design of Network- based Software Architectures" scrisă de Roy Fielding, unul din principalii autori a specificațiilor protocolului HTTP. REST se referă strict la o colecție de principii arhitecturale care definesc felul în care sunt definite si adresate resursele. În mod uzual o resursă reprezintă o entitate care poate fi stocată într-un computer şi poate fi reprezentată printr-un sir de biți: un document, un rând dintr-o baza de date, sau rezultatul rulării unui algoritm. Fiecare resursa are atașat un identificator global (un URI).
REST se referă strict la o colecție de principii arhitecturale într-o rețea care subliniază felul în care resursele sunt definite si adresate. Într-un context mai puțin riguros, REST descrie orice interfață care transmite date prin HTTP fără un nivel adițional de mesagerie cum ar fi SOA sau folosirea sesiunilor prin cookies HTTP.

REST este un model arhitectural care propune următoarele principii de baza pentru serviciile web:

- se folosesc numai metodele definite de protocolul HTTP
- serviciile sunt fără stare (stateless)
- fiecare resursa are atașat un identificator unic (URI – Uniform Resource Identifier)
- resursele pot fi reprezentate atât in format XML cat si format JSON (JavaScript Object Notation)

Lipsa stării se traduce prin faptul că orice cerere HTTP se întâmplă într-o izolare completă. Atunci când un client face o cerere HTTP, include informațiile necesare pentru ca serverul să îndeplinească acea cerere. Serverul nu se bazează niciodată pe cereri precedente. Lipsa stării aduce și noi funcționalități. Este mai ușor să distribui o aplicație fără stare pe mai multe servere. Deoarece două cereri nu depind niciodată una de cealaltă, pot fi tratate de două servere diferite fără să comunice între ele. Acest aspect îmbunătățește scalabilitatea aplicației.

Una dintre caracteristicile principale ale serviciilor de tip REST este folosirea metodelor HTTP. Principiile REST stabilesc o corelare intre operațiile de creare, citire, actualizare sau ștergere (CRUD – create, read, update, delete) a unei resurse si metodele HTTP.

Conform acestor principii metodele HTTP sunt folosite astfel :

- pentru a crea o resursa pe server se folosește metoda POST
- citirea unei resurse se face folosind metoda GET
- actualizarea unei resurse se face prin folosirea metodei PUT
- pentru ștergerea resursei se folosește metoda DELETE

Astfel, conform principiilor REST, fiecare „serviciu" este văzut ca o resursa identificata in mod unic de către un URL, iar pentru interacțiunea cu resursele este permisa doar folosirea metodelor definite de protocolul HTTP si anume GET, PUT, POST, DELETE.

Serviciile REST au câștigat popularitate odată cu creșterea numărului de aplicații mobile si răspândirea platformelor de socializare.

Fiecare resursa poate fi reprezentata fie folosind formatul XML fie JSON (JavaScript Object Notation). JSON reprezintă un format flexibil pentru reprezentarea datelor. Acest format este mult mai ușor de citit si editat in comparație cu XML-ul. Formatul JSON are următoarea structura: { „cheie" : „valoare" }. Modul in care este reprezentata o resursa, poate fi controlat prin intermediul antetelor (header) HTTP. Astfel in momentul interogării unui serviciu rest, clientul poate sa specifice formatul dorit folosind header-ul „Accept". Valorile posibile sunt „application/xml" pentru o reprezentare in format XML sau „application/json" pentru o reprezentare in format JSON. De asemenea, răspunsul primit de la serviciul web conține informații despre modul de reprezentare al informației in header-ul „Content-Type"

Următorul exemplu prezinta folosirea metodei GET pentru a obține reprezentarea unei resurse de tip utilizator specificate in format XML

```
GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

Folosirea metodei GET indica o interogare (se doreste citirea resursei), iar resursa este identificata printr-un URI unic (/users/Robert). In acest exemplu antetul „Accept" semnalizeaza faptul ca resursa va trebui reprezentata folosind formatul XML.

Acest mecanism de „negociere" a formatului de reprezentare a unei resurse, oferă un grad ridicat de flexibilitate, serviciul web putându-se adapta la cerințele consumatorului.

In limbajul Java, suportul pentru servicii REST este oferit de către specificația Java API for Restful Web Services (JAX-RS) care este definită de JSR-311. Există mai multe implementări ale specificației JAX-RS printre care se pot enumera: Apache CXF, JBoss WebServices.

Beneficiile utilizării arhitecturii REST în dezvoltarea serviciilor web sunt multiple. Datorită faptului că reprezentările pot fi stocate în cache, timpul de răspuns al serverului şi încărcarea acestuia sunt reduse. Scalabilitatea serverului este îmbunătățită reducându-se nevoia ca serverul să mențină anumite stări care țin de o sesiune. Astfel servere diferite pot fi folosite pentru a trata diferite cereri dintr-o sesiune. Codul de pe client este redus deoarece browserul poate fi folosit pentru a accesa orice resursă. De asemenea, un serviciu web RESTful depinde mai puțin de formate proprietare și de frameworks de mesagerie deasupra HTTP.

Servicii REST in diferite limbaje de programare

In aceasta sectiune sunt prezentate exemple de accesare a serviciilor REST în diferite limbaje de programare. Sunt accentuate mecanismele de transmitere HTTP GET și/sau cereri POST în fiecare limbaj in parte, folosirea practica a serviciilor REST fiind doar o problema de design arhitectural.

1 Servicii REST in C#

Cererile HTTP GET se efectueaza folosind *HttpWebRequest* și *HttpWebResponse* din *System.Net*. Următoarul exemplu emite o cerere și returnează întregul răspuns ca un singur șir:

```
static string HttpGet(string url) {
  HttpWebRequest req = WebRequest.Create(url)
                       as HttpWebRequest;
  string result = null;
  using (HttpWebResponse resp = req.GetResponse()
                                as HttpWebResponse)
  {
    StreamReader reader =
        new StreamReader(resp.GetResponseStream());
    result = reader.ReadToEnd();
  }
  return result;
}
```

Daca URL-ul contine parametrii, acestia trebuie sa fie codificati in mod corespunzator (de exemplu un spatiu este reprezentat cu "20%"). Namespace-ul *System.Web* este definit de o clasa numita *HttpUtility*, cu o metoda statica numita *UrlEncode* care poate fi folosita pentru astfel de codificari ale URL-ului.

Pentru cererile de tip HTTP POST este obligatorie codificarea URL impreuna cu codificarea datelor din formular (form encoding) asa cum este prezentat in exemplul urmator:

```
static string HttpPost(string url,
    string[] paramName, string[] paramVal)
{
  HttpWebRequest req = WebRequest.Create(new Uri(url))
                       as HttpWebRequest;
  req.Method = "POST";
  req.ContentType = "application/x-www-form-urlencoded";

  // Build a string with all the params, properly encoded.
  // We assume that the arrays paramName and paramVal are
  // of equal length:
  StringBuilder paramz = new StringBuilder();
  for (int i = 0; i < paramName.Length; i++) {
    paramz.Append(paramName[i]);
    paramz.Append("=");
    paramz.Append(HttpUtility.UrlEncode(paramVal[i]));
    paramz.Append("&");
  }

  // Encode the parameters as form data:
  byte[] formData =
      UTF8Encoding.UTF8.GetBytes(paramz.ToString());
  req.ContentLength = formData.Length;

  // Send the request:
  using (Stream post = req.GetRequestStream())
  {
    post.Write(formData, 0, formData.Length);
  }

  // Pick up the response:
  string result = null;
  using (HttpWebResponse resp = req.GetResponse()
                                as HttpWebResponse)
  {
    StreamReader reader =
        new StreamReader(resp.GetResponseStream());
    result = reader.ReadToEnd();
  }

  return result;
}
```

2 Servicii REST in Java

In Java cererile HTTP GET se realizeaza folosint clasa *HttpURLConnection*, invocand metoda *openConnection* pe un obiect de tip URL.

In următoarul exemplu este prezentata emiterea unei cereri și returnarea întregului răspuns ca un *șir lung* (long string):

```
public static String httpGet(String urlStr) throws IOException {
  URL url = new URL(urlStr);
  HttpURLConnection conn =
      (HttpURLConnection) url.openConnection();

  if (conn.getResponseCode() != 200) {
    throw new IOException(conn.getResponseMessage());
  }

  // Buffer the result into a string
  BufferedReader rd = new BufferedReader(
      new InputStreamReader(conn.getInputStream()));
  StringBuilder sb = new StringBuilder();
  String line;
  while ((line = rd.readLine()) != null) {
    sb.append(line);
  }
  rd.close();

  conn.disconnect();
  return sb.toString();
}
```

Daca URL-ul contine parametrii, acestia trebuie sa fie codificati in mod corespunzator folosind clasa *URLEncoder*.

Pentru cererile HTTP POST codificarea URL-ului (URL encoding) este de asemenea obligatorie asa cum este prezentat in exemplul urmator:

```
public static String httpPost(String urlStr, String[] paramName,
String[] paramVal) throws Exception {
  URL url = new URL(urlStr);
  HttpURLConnection conn =
      (HttpURLConnection) url.openConnection();
  conn.setRequestMethod("POST");
  conn.setDoOutput(true);
  conn.setDoInput(true);
  conn.setUseCaches(false);
  conn.setAllowUserInteraction(false);
  conn.setRequestProperty("Content-Type",
      "application/x-www-form-urlencoded");

  // Create the form content
  OutputStream out = conn.getOutputStream();
  Writer writer = new OutputStreamWriter(out, "UTF-8");
  for (int i = 0; i < paramName.length; i++) {
    writer.write(paramName[i]);
    writer.write("=");
    writer.write(URLEncoder.encode(paramVal[i], "UTF-8"));
    writer.write("&");
  }
```

```
  writer.close();
  out.close();

  if (conn.getResponseCode() != 200) {
    throw new IOException(conn.getResponseMessage());
  }

  // Buffer the result into a string
  BufferedReader rd = new BufferedReader(
      new InputStreamReader(conn.getInputStream()));
  StringBuilder sb = new StringBuilder();
  String line;
  while ((line = rd.readLine()) != null) {
    sb.append(line);
  }
  rd.close();

  conn.disconnect();
  return sb.toString();
}
```

Suportul limbajului Java pentru manipularea conexiunilor web este definit la un nivel destul de scăzut, fiind necesara adaugarea de structuri "try/catch/finally". Biblioteca Apache Commons este o solutie populara pentru manipularea conexiunilor web, in special setul de pachete *httpclient.*

3 Servicii REST in JavaScript

In exemplele urmatoare va fi prezentata trimiterea de cererei (solicitari) REST in limbajul JavaScript pe partea de client (in-browser client-side). Dacă se doreste utilizarea JavaScript pe partea de server (server-side), trebuie consultata documentația distribuitorul serverului pentru crearea de cereri HTTP.

Pentru ca JavaScript este un limbaj bazat pe conceptul prototipurilor, pentru trimiterea de cereri HTTP se foloseste obiectul *XMLHttpRequest* (trebuie retinut faptul ca in ciuda numelui acestiua, nici cererea, nici raspunsul nu trebuie sa implice XML):

```
function createRequest() {
  var result = null;
  if (window.XMLHttpRequest) {
    // FireFox, Safari, etc.
    result = new XMLHttpRequest();
    if (typeof xmlhttp.overrideMimeType != 'undefined') {
      result.overrideMimeType('text/xml'); // Or anything else
    }
  }
  else if (window.ActiveXObject) {
    // MSIE
    result = new ActiveXObject("Microsoft.XMLHTTP");
  }
  else {
    // No known mechanism -- consider aborting the application
  }
  return result;
}
```

Un obiect *XMLHttpRequest* permite trimiterea unei solicitari, fie GET sau POST; cu toate acestea, nu se returneaza in mod predefinit o valoare. Trebuie implementata o funcție de "callback" (apel invers), care va fi invocata atunci când cererea se finalizează:

```
var req = createRequest(); // defined above
// Create the callback:
req.onreadystatechange = function() {
  if (req.readyState != 4) return; // Not there yet
  if (req.status != 200) {
    // Handle request failure here...
    return;
  }
  // Request successful, read the response
  var resp = req.responseText;
  // ... and use it as needed by your app.
}
```

In cazul în care răspunsul este de tip XML (serverul folosind MIME de tip *text/xml*), acesta poate fi citit folosind proprietatea *responseXML*. Această proprietate conține un document XML care poate fi folosite utilizand proprietatile DOM (Document Object Model) JavaScript.

Trimiterea cererii: odată creat obiectul de tip "cerere" și configurata funcția de *callback*, poate să se emită cererea:

```
req.open("GET", url, true);
req.send();
```

În cazul solicitărilor POST, se utilizeaza:

```
req.open("POST", url, true);
req.setRequestHeader("Content-Type",
                     "application/x-www-form-urlencoded");
req.send(form-encoded request body);
```

4 Servicii REST in PHP

Pentru emiterea de cereri GET in limbajul PHP se pot folosi functiile *fopen*, *file_get_contents*, sau orice alta functie nativa dedicata citirii fisierelor:

```
$url = "http://www.example.com/products/1234";
$response = file_get_contents($url);
echo $response;
```

Orice parametru folosit pentru cererile de tip GET trebuie să fie codificat (de exemplu, un spațiu este reprezentat de "20%").In PHP codificarea sirurilor de caractere se poate face folosind funcția nativa *urlencode*.

Spre deosebire de cererile GET, cererile de tip POST se realizeaza mai complex in limbajul PHP. In lipsa unui URL definit cu parametrii, pentru cererile de tip POST trebuie sa se deschida o conexiune la server si sa se trimita manul toate antetele HTTP:

```
function httpRequest($host, $port, $method, $path, $params) {
  // Params are a map from names to values
  $paramStr = "";
  foreach ($params as $name, $val) {
```

```
    $paramStr .= $name . "=";
    $paramStr .= urlencode($val);
    $paramStr .= "&";
  }

  // Assign defaults to $method and $port, if needed
  if (empty($method)) {
    $method = 'GET';
  }
  $method = strtoupper($method);
  if (empty($port)) {
    $port = 80; // Default HTTP port
  }

  // Create the connection
  $sock = fsockopen($host, $port);
  if ($method == "GET") {
    $path .= "?" . $paramStr;
  }
  fputs($sock, "$method $path HTTP/1.1\r\n");
  fputs($sock, "Host: $host\r\n");
  fputs($sock, "Content-type: " .
               "application/x-www-form-urlencoded\r\n");
  if ($method == "POST") {
    fputs($sock, "Content-length: " .
                 strlen($paramStr) . "\r\n");
  }
  fputs($sock, "Connection: close\r\n\r\n");
  if ($method == "POST") {
    fputs($sock, $paramStr);
  }

  // Buffer the result
  $result = "";
  while (!feof($sock)) {
    $result .= fgets($sock,1024);
  }

  fclose($sock);
  return $result;
}
```

Emiterea unei cereri POST, se poate realiza precum in exemplul urmator:

```
$resp = httpRequest("www.example.com",
    80, "POST", "/userDetails",
    array("firstName" => "John", "lastName" => "Smith"));
```

5 Servicii REST in Python

In Python emiterea cererilor de tip GET se realizeaza folosind modulul *urllib2*.

```
import urllib2

url = 'http://www.example.com/products/1234'
response = urllib2.urlopen(url).read()
```

Erorile sunt raportate ca excepții (*urllib2.HTTPError* sau *urllib2.URLError*).

Cererile de tip POST se emit folosind date codificate (encoded data) ca parametrii suplimentari la functia *urlopen*:

```
import urllib
import urllib2

url = 'http://www.example.com/users/details'
params = urllib.urlencode({
  'firstName': 'John',
  'lastName': 'Smith'
})
response = urllib2.urlopen(url, params).read()
```

Pentru codificare se foloseste functia *urlencode* din biblioteca *urllib*.

6 Servicii REST in Ruby

In Ruby emiterea cererilor de tip GET se realizeaza folosind clasa *Net:HTTP*:

```
require 'net/http'

url = 'http://www.example.com/products/1234'
resp = Net::HTTP.get_response(URI.parse(url))

resp_text = resp.body
```

Raspunsul cererii contine *codul de raspuns HTTP* (*HTTP response code*, ex: *HTTP OK code*, *HTTP Not Found code* etc.). O ierarhie completă a claselor de răspuns este definita în documentația Net :: HTTP.

Pentru cererile de tip POST se foloseste metoda *post_form* din clasa *Net:HTTP*:

```
require 'net/http'

url = 'http://www.example.com/user/details'
params = {
  firstName => 'John',
  lastName => 'Smith'
}

resp = Net::HTTP.post_form(url, params)

resp_text = resp.body
```

In limbajul Ruby codificarea implicita (form encoding) este "application/x-www-form-urlencoded".

Arhitecturi orientate pe servicii

Arhitectura orientată pe servicii (SOA – Service Oriented Architecture) este un model arhitectural care presupune dezvoltarea aplicațiilor software pe baza unor componente reutilizabile numite servicii. SOA impune un model bazat pe un set de servicii interconectate care comunica prin interfețe si protocoale standardizate. Deseori arhitectura orientata pe servicii este văzută ca o evoluție a programării distribuite.

Capacitatea de reutilizare a serviciilor în cadrul aplicațiilor reprezintă o caracteristica principala a acestui model arhitectural. Datorita avantajelor oferite, programarea orientata pe servicii s-a impus pe piața ca un model fiabil de dezvoltare software. SOA permite construirea diverselor sisteme informatice intr-o maniera care sa permită cooperarea (interoperabilitatea, integrarea) eficienta intre ele.

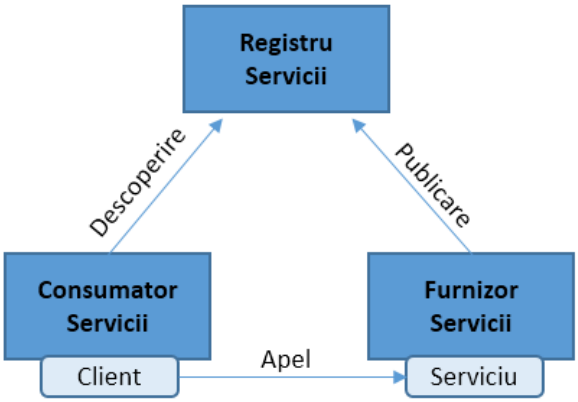
Principiile SOA recomandă ca noile servicii săfie dezvoltate prin combinarea și reutilizarea serviciilor deja existente, dar in același timp componentele sistemului trebuie sa își păstreze un grad ridicat de independentă. De asemenea un alt principiu important este utilizarea si descoperirea serviciilor in mod dinamic.

Unul dintre cele mai întâlnite modele de referința SOA, definește trei entități de baza:

- Director de servicii (Service registry). Aceasta entitate are rol de intermediar intre furnizorii de servicii si clienti, care au posibilitatea de a caută si accesa serviciile publicate de către furnizori. Un director de servicii poate fi comparat cu un registru sau catalog care conține o lista de servicii înregistrate.

- Furnizor de servicii (Service provider). Aceasta entitate are rolul de a defini si implementa un serviciu si eventual de a publica serviciul respectiv in directorul de servicii.

- Utilizator de servicii (Service consumer). Consumatorul sau clientul folosește capabilitățile de căutare oferite de registrul de servicii pentru a localiza un anumit serviciu



Entitățile definite de către arhitectura SOA

Registrul UDDI (Universal Description, Discovery, and Integration) are rolul de a cataloga serviciile web si de a permite căutarea acestora. Registrul UDDI poate fi accesat si la runtime in vederea descoperiri si invocării dinamice a unui serviciu Web.

Registrul UDDI este asociat cu trei tipuri de informații:

- White pages – conține informații despre furnizorii de servicii cum ar fi nume, adresa, contact, etc.
- Yellow pages – permite clasificarea serviciilor in funcție de o taxonomie standard, cum ar fi de exemplu SIC (Standard Industrial Classification).
- Green pages – conțin informații tehnice legate de servicii, cum ar fi documentul WSDL asociat serviciului.

Arhitectura orientata pe servicii definește un set de caracteristici si principii care pot fi rezumate astfel:

- Scalabilitate Deoarece diversele componente – serviciile – care alcătuiesc soluția software dezvoltata sunt distribuite si comunica intre ele prin protocoale standard gen HTTP, este posibila scalarea individuala a acestora.

- Dependenta scăzută (loose coupling). Arhitectura orientată pe servicii reprezintă o evoluție de la sistemele distribuite in care exista o dependenta rigida intre elementele ce alcătuiesc sistemul distribuit. Astfel, dependenta intre un serviciu si consumatorul acelui serviciu este la nivel de contract sau interfața. Serviciul oferă un anumit contract in care se specifica funcționalitatea pe care acesta o oferă.

- Interoperabilitate. Datorita folosirii XML ca standard pentru codarea informației care este transmisa sub forma de mesaje, SOA permite comunicarea intre aplicații dezvoltate in limbaje de programare si tehnologii diferite, rulând pe platforme diferite.

- Descoperire. Prin folosirea unui registru de servicii, este posibila descoperirea si invocarea dinamica a serviciilor.

- Abstractizare. Interfața unui serviciu web este descrisa intr-un mod abstractizat sub forma unui fisier WSDL. Un client poate invoca si consuma serviciul web fără a fi nevoie sa cunoască detaliile de implementare ale serviciului consumat.

- Standardizare.

În cadrul SOA se pot distinge trei perspective arhitecturale importante:

- Arhitectura aplicației – specifică mediului business. Este caracterizată de conceptul de utilizare a serviciilor achiziționate de la mai mulți furnizori și de integrare a acestora pentru a forma un proces business;

- Arhitectura serviciului – asigură o punte între implementarea și utilizarea aplicațiilor, prin crearea unei viziuni logice a funcţiilor disponibile. Apelarea se realizează prin intermediul unei interfețe comune și a unei arhitecturi de gestiune;

- Arhitectura componentelor – descrie diversele medii care stau la baza implementării aplicațiilor, obiectele de tip business și implementările acestora.

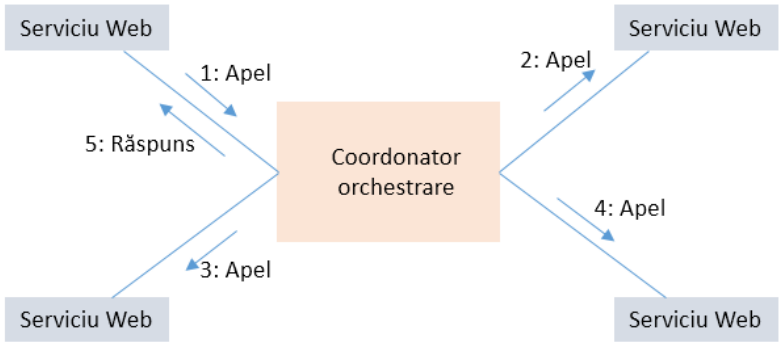
Orchestrarea serviciilor web

In cadrul organizațiilor folosirea serviciilor web se poate încadra in doua mari categorii: integrare de aplicații (EAI – Enterprise Application Integration) sau integrare de parteneri externi in scenarii de tip Business-to-Business (B2B). In fiecare din aceste categorii, serviciile web pot varia in complexitate, de la servicii simple de tip cerere-răspuns pana la interacțiuni complexe implicând tranzacții intre mai multe entități si care pot sa se întindă pe o durata mai lunga de timp. Scenariile de business complexe impun agregarea si orchestrarea mai multor servicii web sub forma unui proces business.

Serviciile web pot fi integrate in doua moduri diferite : orchestrare sau coregrafie.

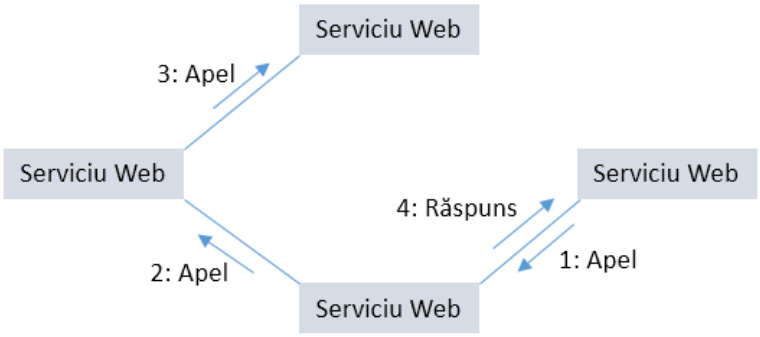
Orchestrarea serviciilor (uneori numita si compunere sau coordonare) precizează modul in care vor fi apelate mai multe servicii care cooperează pentru îndeplinirea unui obiectiv complex. In acest caz exista o entitate centrala, numita proces business, care deține controlul asupra execuției si coordonează apelarea diverselor operații ale serviciilor web.

Pentru orchestrarea serviciilor au fost dezvoltate mai multe standarde si specificații. Dintre acestea cel mai des folosite standarde sunt BPEL (Business Process Execution Language) si WS-Coord/WS-CDL (WS-Choreography Description Language).



Orchestrarea serviciilor web

Coregrafia serviciilor in contrast cu orchestrarea, nu se bazează pe un punct central de control. In acest caz, fiecare serviciu web implicat in coreografie știe când sa execute anumite operații si care sunt serviciile cu care trebuie sa interacționeze.



Coregrafia serviciilor web

Din punct de vedere al integrării serviciilor web in cadrul unui scenariu complex, orchestrarea reprezintă o alegere mult mai buna decât coreografia si are următoarele avantaje:

- coordonarea se face dintr-un punct central care este ușor de gestionat si actualizat
- serviciile web pot fi integrate fără a necesita modificări
- se pot implementa scenarii alternative pentru a trata cazurile de eroare

BPEL sau BPEL4WS (Business Process Execution Language for Web Services) este un limbaj de modelare a proceselor dezvoltat de către organizația OASIS. Standardul BPEL a rezultat in urma procesului de contopire a doua limbaje predecesoare: Web Services

Flow Language (WSFL) dezvoltat de către IBM si a limbajului XLANG dezvoltat de către Microsoft.

BPEL combina concepte preluate de la cele doua limbaje si pune la dispoziție un vocabular bogat pentru descrierea de procese.Un proces BPEL specifica ordinea in care serviciile web participante la orchestrare trebuie invocate precum si mesajele care sunt trimise la fiecare serviciu in parte. Intr-un scenariu simplu, un proces BPEL primește o cerere de la un consumator (client), apelează serviciile web conform logicii definite in proces si returnează rezultatul clientului.

Procesul BPEL este compus dintr-o succesiune de pași care poarta numele de activități. Se disting doua categorii de activități: activități primitive si activități structurate.

Activitățile primitive definite in cadrul BPEL sunt următoarele:

- <receive> : in momentul recepționarii unui mesaj de la un alt proces client sau partener, este invocata aceasta activitate.

- <invoke> : prin intermediul acestei activități este invocat un serviciu web (numit si partener) prin trimiterea unui mesaj.

- <reply> : aceasta activitate poate fi utilizata pentru a trimite un mesaj de răspuns la o cerere primita anterior in cadrul unei activități de tip <receive>

- <assign> : activitate prin care se asignează o valoare unei variabile definite in cadrul procesului

- <throw> : in cazul detectării unei erori in execuția procesului, poate fi aruncata o excepție prin intermediul acestei activități. Aceasta excepție va indica cauza erorii care a dus la întreruperea execuției procesului.

- <wait> : la execuția unei activități de acest tip, procesul va aștepta un anumit interval de timp, specificat in cadrul activității.

- <terminate> : aceasta activitate este utilizata pentru terminarea întregului proces.

Cele mai importante activități structurate sunt:

- <sequence> : permite definirea unui set de activități, care vor fi rulate in mod secvențial
- <flow> : permite definirea unui set ce activități care pot fi rulate in paralel
- <switch> : este utilizata pentru implementarea cazurilor de decizie
- <while> : permite definirea unui set de activități care vor fi executate in bucla pana la îndeplinirea condiției de terminare

Aceste activități pot fi combinate in diverse moduri pentru a crea procese complexe.

In exemplul care urmează este prezentat un model de bază al unui proces BPEL folosind sintaxa XML. In mod tipic, un proces BPEL începe cu o activitate de tip <receive>, care reprezintă punctul de acces la procesul BPEL si se termina cu o activitate de tip <reply> care trimite răspuns clientului (consumator) care a inițiat procesul.

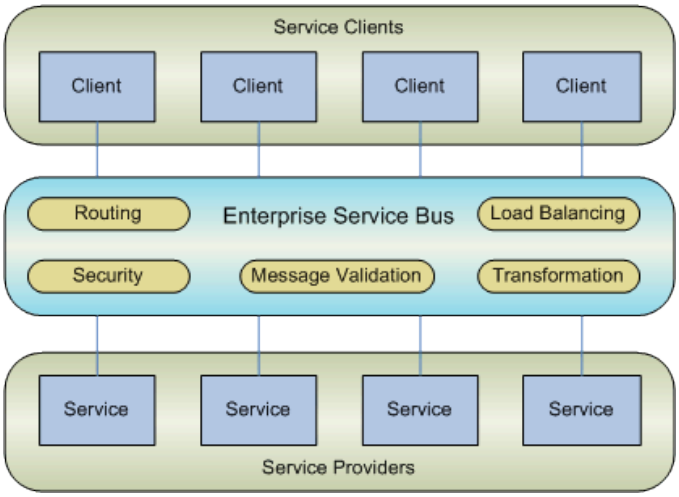
```
<process name="BpelProcessName"
targetNamespace="..."
xmlns="http://schemas.xmlsoap.org/ws/2004/03/business-process/">
<partnerLinks>
<partnerLink name="partnerA"
    partnerLinkType="wsdl:partnerALinkType"
    myRole="myRoleInRelationToPartner"/>
            .....
</partnerLinks>
<variables>
<variable name="varA" messageType="wsdl:MessageA"/>
    .....
</variables>
<!--the executable part of the workflow-->
<sequence>
<receive partnerLink="partnerA"
    portType="wsdl:partnerALinkType"
    operation="doSomething" variable="varA"/>
    <assign>
            <copy>
                    <from>$varA.someParameter</from>
                    <to>$varB.anotherInfo</to>
            </copy>
    </assign>
<invoke partnerLink="partnerB" portType="pb:anotherPartnerPT" operation="doSomethingElse"
inputVariable="varB" outputVariable="varC"/>
    .....
<reply partnerLink="partnerA" portType="wsdl:partnerALinkType" operation="doSomething"
variable="result"/>
</sequence>
</process>
```

Un proces BPEL este executat într-un mediu special numit BPEE (Business Process Execution Engine) sau BPE (Business Process Engine). BPE este de fapt un container care are rolul de a gestiona procesele business pe toată durata lor de viată. Mediul de

execuție este responsabil pentru instantierea proceselor, gestionarea resurselor alocate acestor procese, interacțiunea cu alte procese sau servicii.
Deoarece oferă posibilitatea de a crea procese business portabile, standardul BPEL a fost adoptat cu succes in industrie, la ora actuala existând mai multe implementări in cadrul diverselor platforme SOA existente. Printre cele mai cunoscute implementări ale standardului BPEL pot fi enumerate :

- Apache ODE : implementare open source care suporta standardul WS-BPEL 2.0
- WebSphere Process Server : implementare proprietara IBM
- RiftSaw : implementare open source de la JBoss a unui runtime BPEL
- jBPM : JBoss Business Process Management
- Oracle BPEL Process Manager
- BizTalk Server : implementare proprietara Microsoft

Enterprise Service Bus (ESB) este unul din cele mai importante modele ale arhitecturii orientate pe servicii (SOA). Conceptul de ESB a fost la început descris ca fiind 'o nouă arhitectură care exploatează serviciile Web, trimițând mesaje, realizând o rutare inteligenta precum si transformare' spune Roy Schulte in lucrarea sa 'Predicts 2003: Enterprise Services Buses Emerge', 2002.ESB reprezintă o soluție de tip middleware care are rolul de a interconecta aplicații software diverse in cadrul unei arhitecturi orientate pe servicii (SOA). Magistrala de servicii implementează interfețe standard pentru comunicație, conectivitate, transformare, portabilitate, si securitate. Se poate face o analogie cu conceptul de magistrala întâlnit in arhitectura hardware a computerelor. Diversele aplicații si servicii sunt conectate la aceasta magistrala care facilitează interconectarea acestora.



Modelul Enterprise Service Bus

Primele implementări ale unei astfel de magistrale (ESB) au fost sub forma unor servere de mesaje care, in conexiune cu anumite componente incluse la nivel de mesaj, au fost utilizate pentru direcționarea inteligenta a mesajelor, sau pentru transformarea acestora înaintea de a fi livrate către destinație.

Studiu de caz BPEL - Implementarea unui serviciu de management al conexiunilor multi-modale

Conform standardelor ITU pentru TMN (Telecom Management Networks), sistemele mobile de comunicaţii includ servere centrale cu funcţionalitate pentru colectarea datelor de performanţă (PDC – Performance Data Collection). PDC poate oferi valori măsurate în permanenţă (şi câteva statistici), prin teste de performanţă, care sunt specifice Calității Serviciilor - QoS (Quality of Service). În scopul de a colecta valorile de trafic, PDC, poate fi asociat cu mecanismul de charging (taxare) - înregistrare în timp real a tuturor informaţiilor utile - contoare (de timp, pentru volum), jitter, RTT (round-trip-Time), rata de biţi etc, incluse în "charging tickets".

În domeniul comunicaţiilor mobile de date pot fi luate caracteristicile unui PDP Context, prin sistemul de semnalizare SS7, direct pentru un terminal de date mobil, în faza de activare a contextului. Acest set de parametri poate fi totodată folosit pentru condiţionarea calculului QoS.

Formula propusă pentru QoS cu dublă ponderare

S-au considerat două tipuri de seturi de criterii, suficient de generale pentru a fi comune pentru diferite reţele fără fir (de exemplu, 3G, Wi-Fi, EV-DO etc), care pot funcţiona în cadrul unei comunicări optimizate multi-modale.

- *Profilul de abonat* care poate conţine un set primar de ponderi convenţionale - de exemplu, pe o scară de la "student" la "manager": de la nivelul "student" profilul va fi orientat pe costuri reduse în timp ce la nivelul „manager" alocarea resurselor este mai importantă (lăţime de banda garantată, întârzieri reduse etc.). În dezvoltările viitoare, unele dintre aceste ponderi ar putea fi configurate chiar de către abonat, care poate transmite la server o serie de modificări ale propriilor profile.
- *Termeni ponderaţi*asociaţi nivelului aplicaţie din stiva OSI. Se pot lua, ca referinţă, cele 4 clase definite de standardele 3GPP, definite de altfel şi la descrierea sistemului UMTS:
    - CONVCL - Clasa conversaţională: aplicaţii în timp real. Utilizarea cea mai bine cunoscută din clasa conversaţională este telefonia. Dar cu dezvoltarea Internetului şi multimedia, un număr de noi aplicaţii vor necesita acest sistem, de exemplu, Voice-over-IP, instrumente de video-conferinţe, etc.

- STREAMCL - Clasa streaming: servicii cum ar fi video sau audio în timp real, care sunt transporturile cu sens unic şi foarte sensibile la variaţiile de timp (jitter) între entităţile de informaţii (pachete).
- INTACTCL - Clasa interactivă: aplicaţii non-timp real. Clasa interactivă este utilizată de aplicaţiile ce aşteaptă mesaje (răspunsuri) într-un anumit interval de timp.
- BACKGRCL – Clasa de fundal (background) - Aplicaţii în fundal, la care receptorul nu se aşteaptă la date într-un anumit interval de timp.

La prima vedere pare că toate intrările (în procedura de calcul) sunt la acelaşi nivel de importanţă, dar unele condiţii trebuie să fie impuse de către aplicaţie. De exemplu:

- La nivelul „conversaţional", atât pentru legătura uplink şi downlink, jitter-ul va fi luat în considerare cu un factor de ponderare diferit de zero, în timp ce, în restul claselor, aceasta poate fi considerat egal cu zero.
- Pentru „clasa background" sau „clasa conversaţională", cel mai important factor de ponderare este costul, nu rata de biţi. Se va opta pentru o rată de transfer mai redusă dar ieftină (adesea WiFi e practic gratuit faţă de 3G), în defavoarea unui bitrate sporit dar mai costisitor.

Calculul QoS va lua în considerare costuri diferite pentru traficul efectiv de „uplink" şi „downlink" şi tarife speciale pentru rezervarea de resurse (lărgime de bandă etc.) şi garantarea unor limite conexiuni cu întârziere minimă, jitter minim, RTT minim etc.

Factorul „întârziere" („delay") este foarte important şi de aceea el va fi considerat separat, chiar dacă ar fi putut fi considerat direct în calculele factorului de calitate pentru „uplink" şi „downlink".

Valorile obţinute de la reţea pot fi tratate separat faţă cele de la terminalul mobil (nod de calcul mobil, "telefon inteligent", "comunicator" etc.)

Formula propusă pentru calculul QoS prin sumă dublu ponderată:

$$QoS = \sum_{k=1}^{N} w_{profile,\ k} \cdot w_{class,\ k} \cdot p_{normalized,\ k}$$

Normalizarea parametrului p_k se face la cerinţele aplicaţiei, maximul valorii fiind limitat la 1.

Valoarea cerută de aplicaţie se va afla fie la numărător, fie la numitor, în funcţie de modul în care valoarea ne-normalizată ar trebui să fie maximă sau minimă în mod ideal.

De exemplu, pentru N = 4 parametri măsuraţi - GBR (Garantat Bit Rate), TDELAY (Total Delay), JITTER şi ER (Error Rate) - şi 4 profile (Premium, Voce, normal, şi de bază), este prezentat un posibil set de ponderi şi normalizări:

$w_{Premium,\ 1} = 1,\ w_{Premium,\ 2} = 1,\ w_{Premium,\ 3} = 1,\ w_{Premium,\ 4} = 1$

$w_{Voice,\ 1} = 0.75,\ w_{Voice,\ 2} = 1,\ w_{Voice,\ 3} = 1,\ w_{Voice,\ 4} = 0.5$

$w_{Normal,1}=1, w_{Normal,\ 2}= 0.5, w_{Normal,\ 3}= 0.5, w_{Normal,\ 4}=0.75$

$w_{Basic\ ,\ 1} = 1\ ,\ w_{Basic\ ,\ 2} = 0.25\ ,\ w_{Basic\ ,\ 3} = 0.25\ ,\ w_{Basic\ ,\ 4} = 0.5$

Alte exemple, pentru alte tipuri de aplicaţii ar fi:

- Aplicaţie VoIP - clasă de trafic conversaţional:
  - $w_{Voice,1} = 0.75$, $w_{Voice,2} = 1$, $w_{Voice,3} = 1$, $w_{Voice,4} = 0.5$ (pentru un utilizator interesat, în general, de aplicaţii VoIP)
  - $p_{normalized,1}$ = GBR /Appl_GBR = 1
  - $p_{normalized,2}$ = Appl_ TDELAY/ TDELAY = 1
  - $p_{normalized,3}$ = Appl_ JITTER / JITTER = 1
  - $p_{normalized,4}$ = Appl_ JITTER / JITTER = 0.5

- Web Browsing:
  - $w_{Normal,1} = 1$, $w_{Normal,2} = 0.5$, $w_{Normal,3} = 0.5$, $w_{Normal,4} = 0.75$
  - $p_{normalized,1}$ = GBR /Appl_GBR = 0.5
  - $p_{normalized,2}$ = Appl_ TDELAY/ TDELAY = 0.5
  - $p_{normalized,3}$ = Appl_ JITTER / JITTER = 0.25
  - $p_{normalized,4}$ = Appl_ JITTER / JITTER = 1

QoS_{3G} şi $QoS_{Wi\text{-}Fi}$ sunt calculate, precum şi raportul lor; handover-ul Wi-Fi la 3G este decis dacă QoS_{3G} / $QoS_{Wi\text{-}Fi}$ este de peste 1.1 ($QoS_{Wi\text{-}Fi}$ / QoS_{3G} scade sub 0.9). Valoarea de 10% este aleasă convenţional pentru toleranţă la handover - cu scopul de a asigura un handover fără oscilaţie.

Logica serviciilor de management al conexiunilor multi-modale pe baza QoS

Serviciul care face obiectul acestui paragraf este implementat „Top-Down" pe baza BPEL, cu un bloc de calcul programat în limbajul C.

Integrarea şi testele de validare se vor face într-un studiu de caz bazat pe scenarii de evaluare a QoS pentru comunicaţiile mobile bi-modale (3G/WLAN).

Se poate lua în considerare şi o abordare bazată pe programarea PHP a unei funcţii de consolă asociată acestui server rezident la operatorul de comunicaţii, conformă cerinţelor specifice OMA (Operation-Maintenance-Administration). Consola poate avea, pe lângă funcţiile de configurare şi monitorizare, o importantă funcţionalitate de abonare.

Datele de intrare sunt următoarele (ele au fost deja prezentate mai sus în detaliu):

- ponderi caracteristice profilului abonatului
- ponderi caracteristice aplicaţiei folosită de abonat.

Se pot trata separat valorile provenite de la reţea faţă de cele provenite de la terminalul mobil (nod mobil de calcul, „smartphone", „communicator" etc).

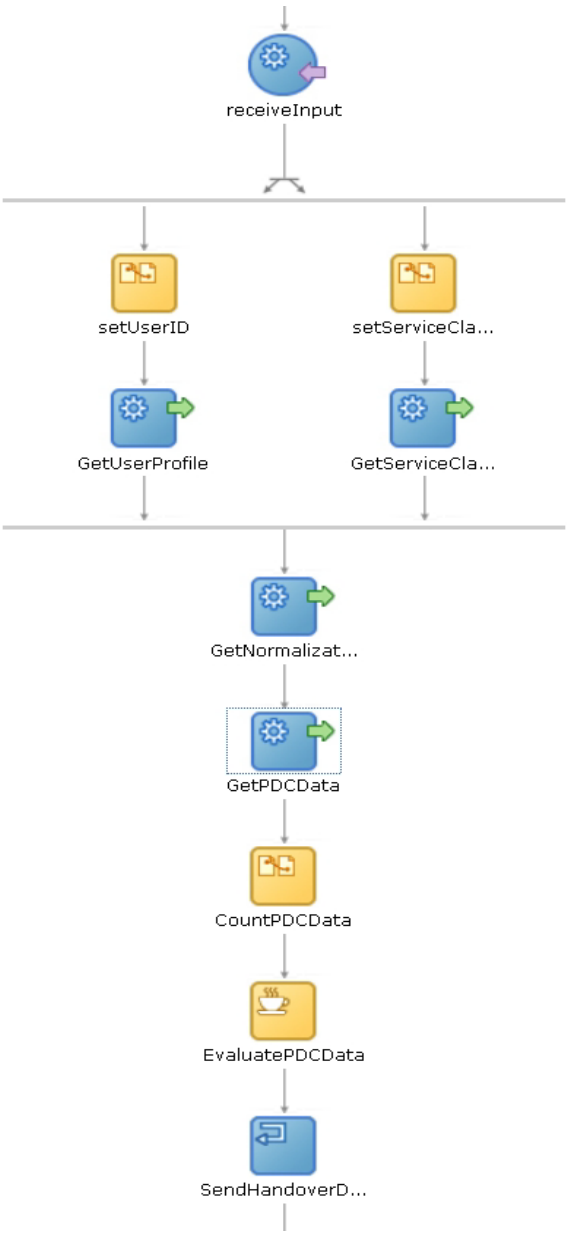
Se alege formatul XML pentru cele două „intrări" în calcul:

- „Profile"
- „Performance Data"

Formula prezentată anterior pentru calculul QoS ca sumă dublu-ponderată este implementată în următoarea porţiune de cod (se observă detalierea lăţimii de bandă atât pentru uplink cât şi pentru downlink):

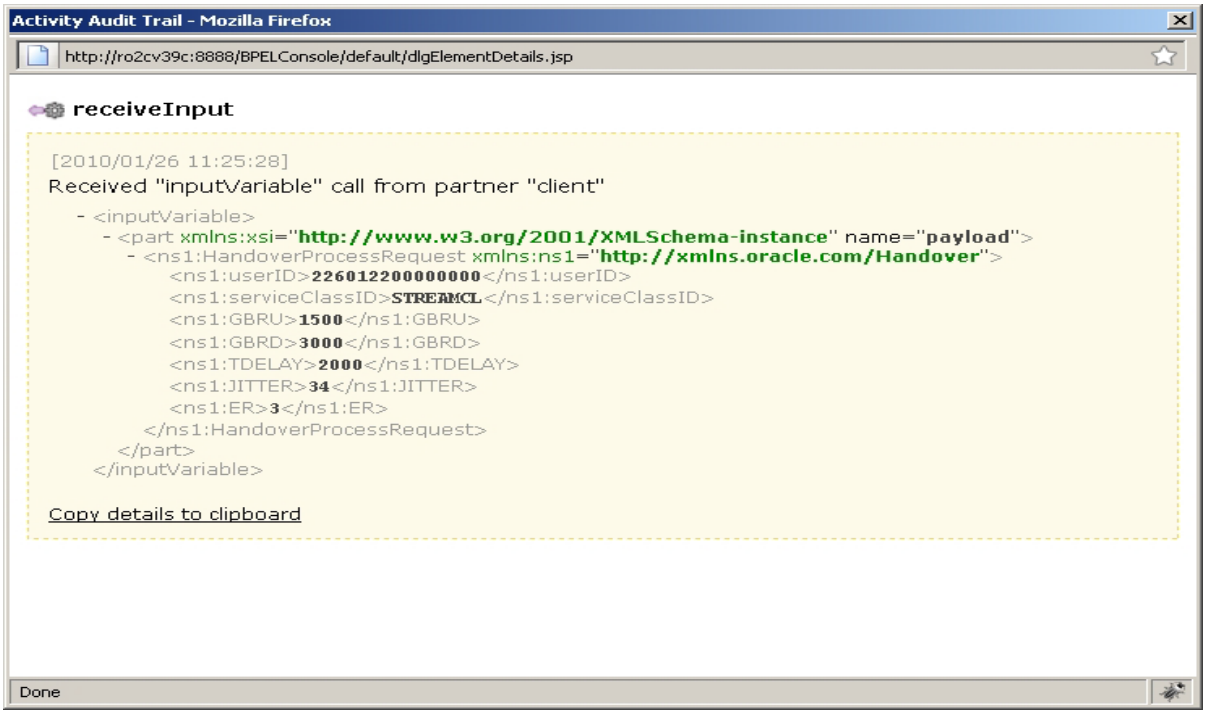
```
// evaluate pdc data
double qos =
w_pr_gbru * w_cl_gbru * avbu / np_gbru +
w_pr_gbrd * w_cl_gbrd * avbd / np_gbrd +
w_pr_tdelay * w_cl_tdelay * np_tdelay / delay +
w_pr_jitter * w_cl_jitter * np_jitter / jitter +
w_pr_er * w_cl_er * np_er / errorRate;
```

Schema de bază a serviciului implementat este:



Schema serviciului implementat

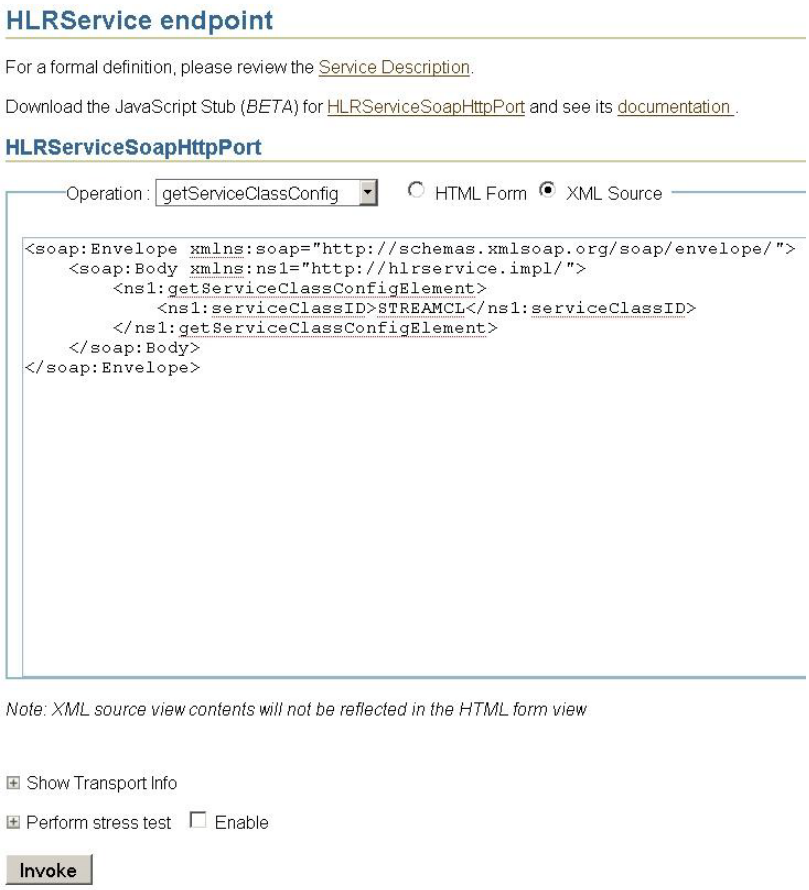
Mai joseste exemplificată invocarea serviciului de evaluare pentru handover cu datele aplicaţiei (clientului).



Datele aplicaţiei clientului – cerere de procesare handover

Interfaţa dezvoltată emulează funcţionalitatea HLR (Home Location Register - sau a unei baze de date similare pentru WLAN, ca de exemplu HSS pentru reţelele ce au implementat IMS), care oferă ServiceClassConfig(uration) şi este prezentată în figurile următoare, atât în forma HTML cât şi sursă XML.

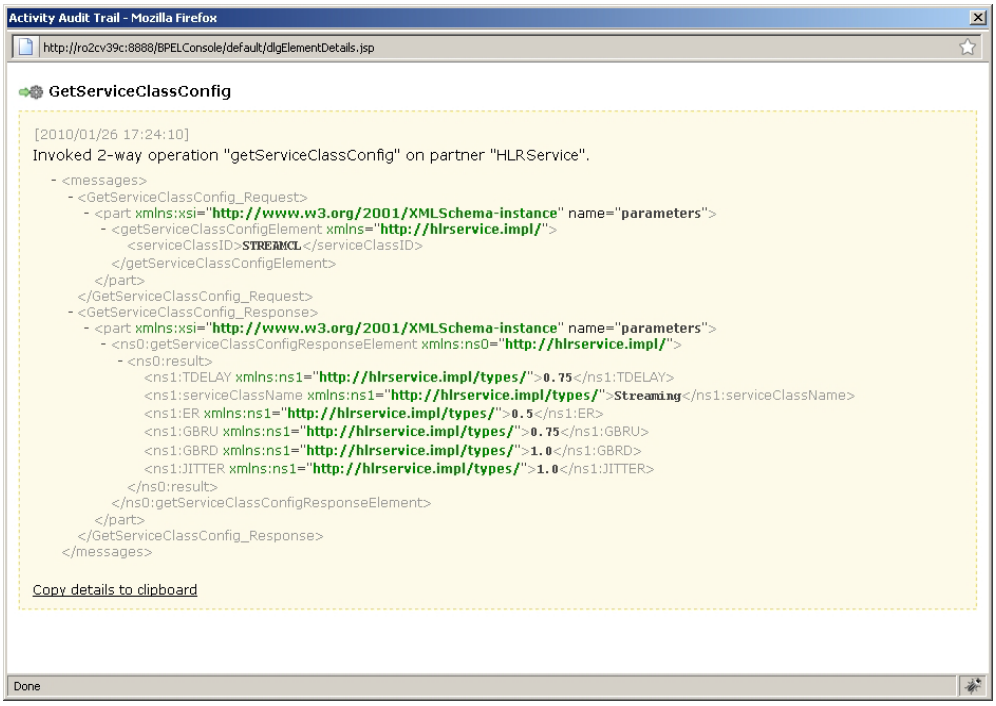
Interfaţa dezvoltată pentru emularea HLR – format HTML



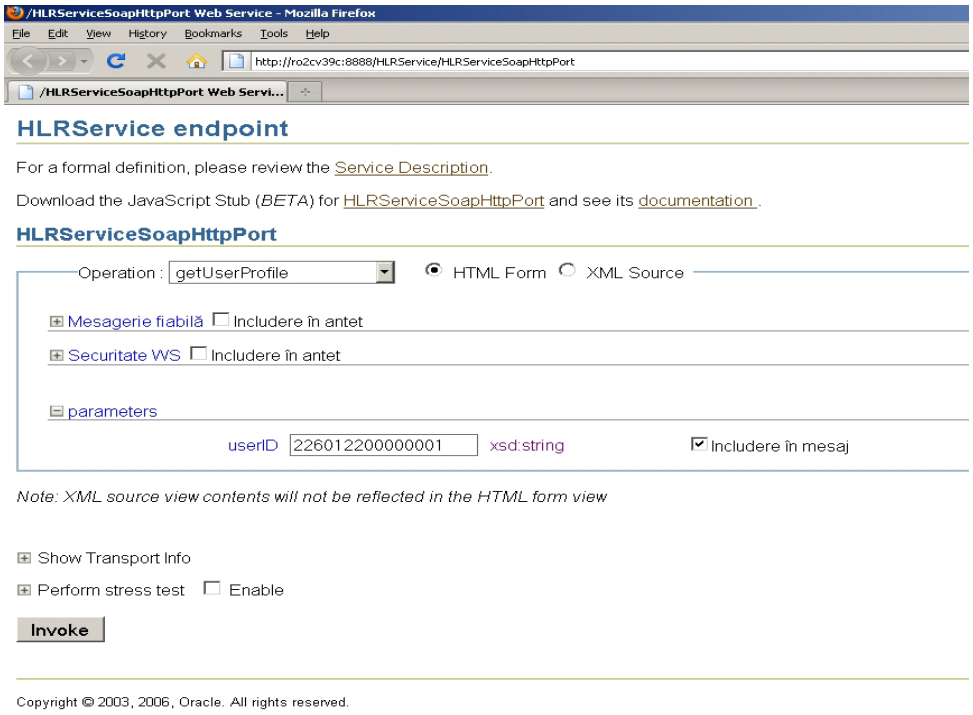
Interfaţa dezvoltată pentru emularea HLR – format XML

Prin această interfaţă, ponderile pot fi introduse pentru diferitele clase de aplicaţii cât şi a profilelor abonat (profiluri de utilizator).

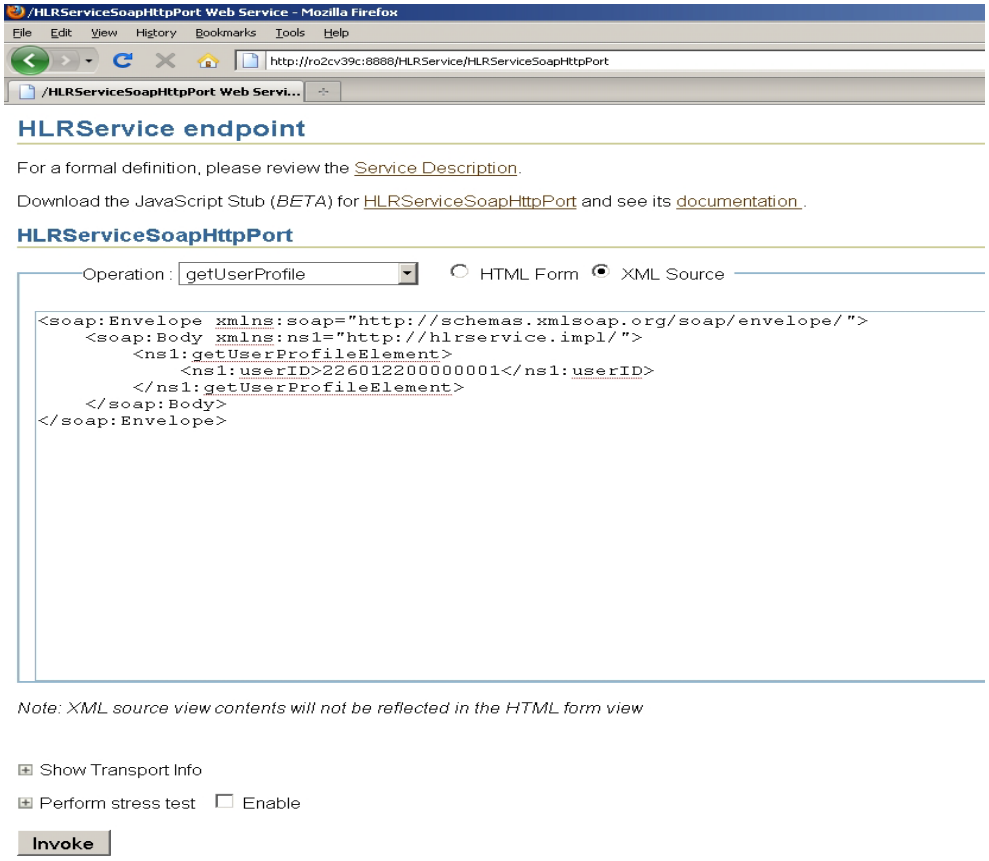
Un astfel de exemplu XML pentru servicii, cât şi un exemplu HTML, respectiv XML, pentru userprofile sunt prezentate în figurile următoare.



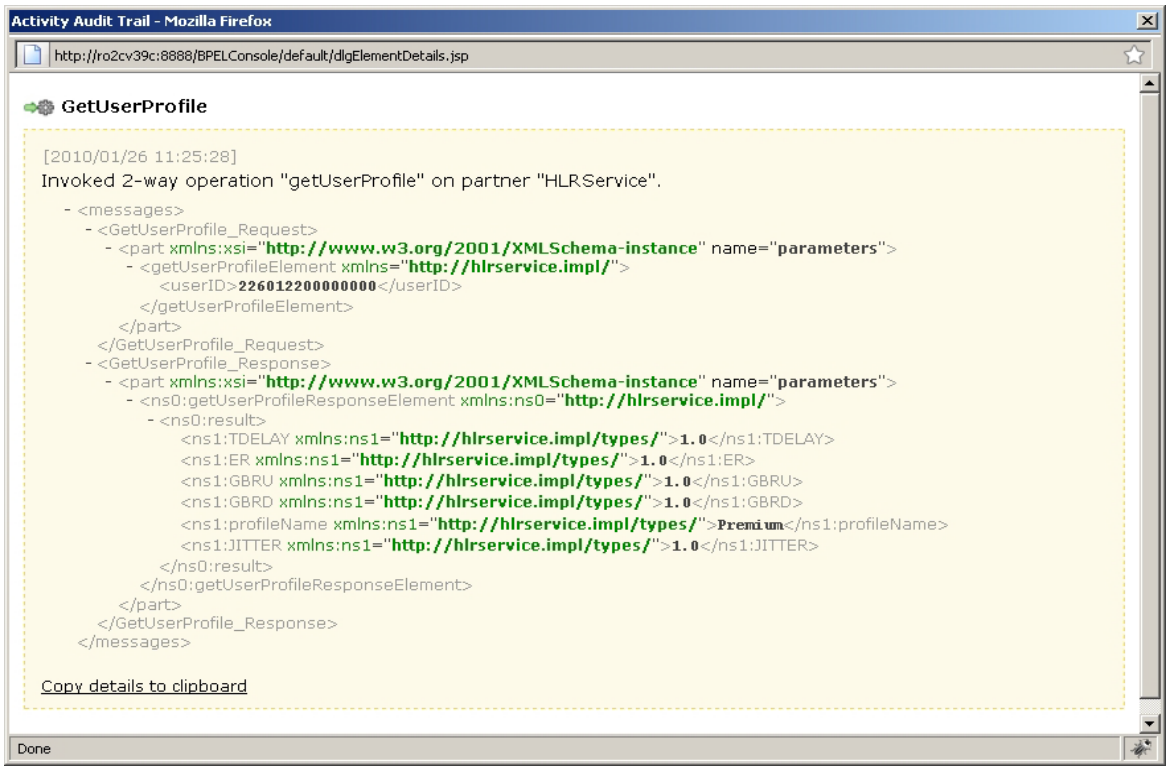
Configurarea serviciilor



Interfaţa userprofile – format HTML

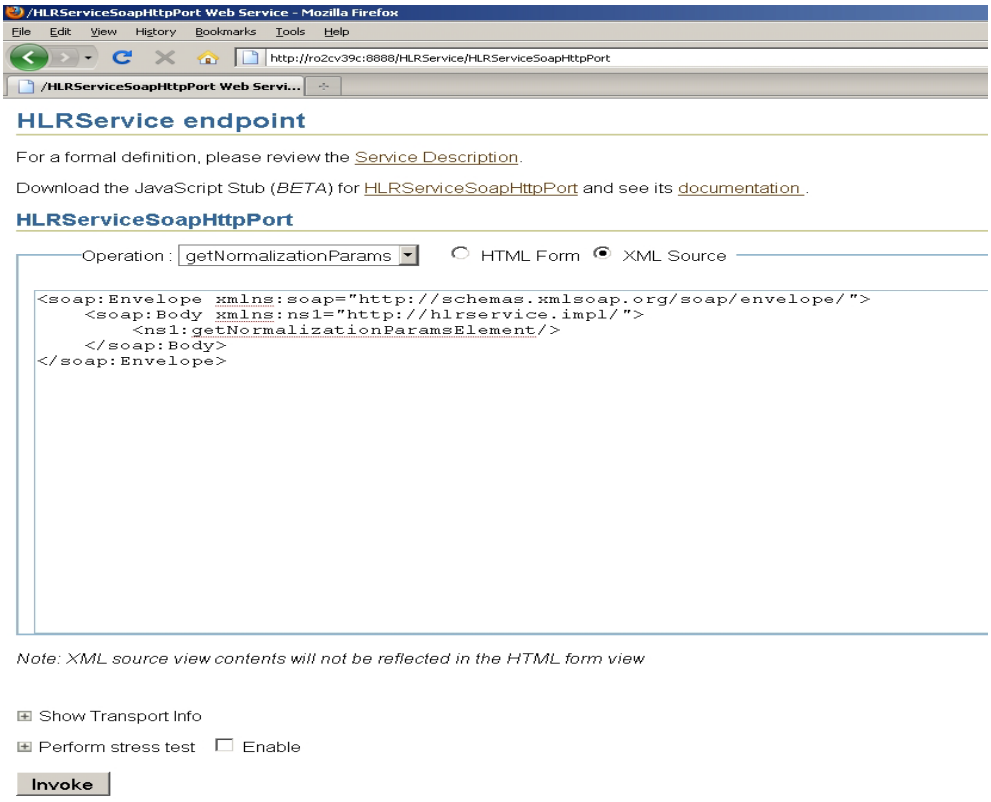


Interfaţa userprofile – format XML

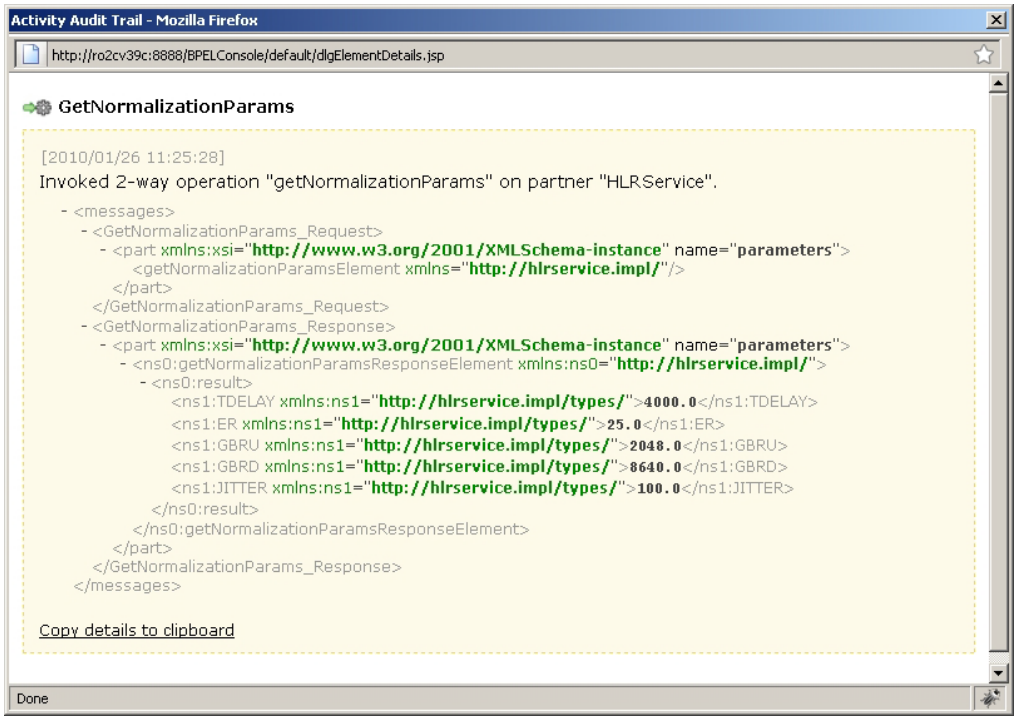


Preluarea profilului utilizatorului

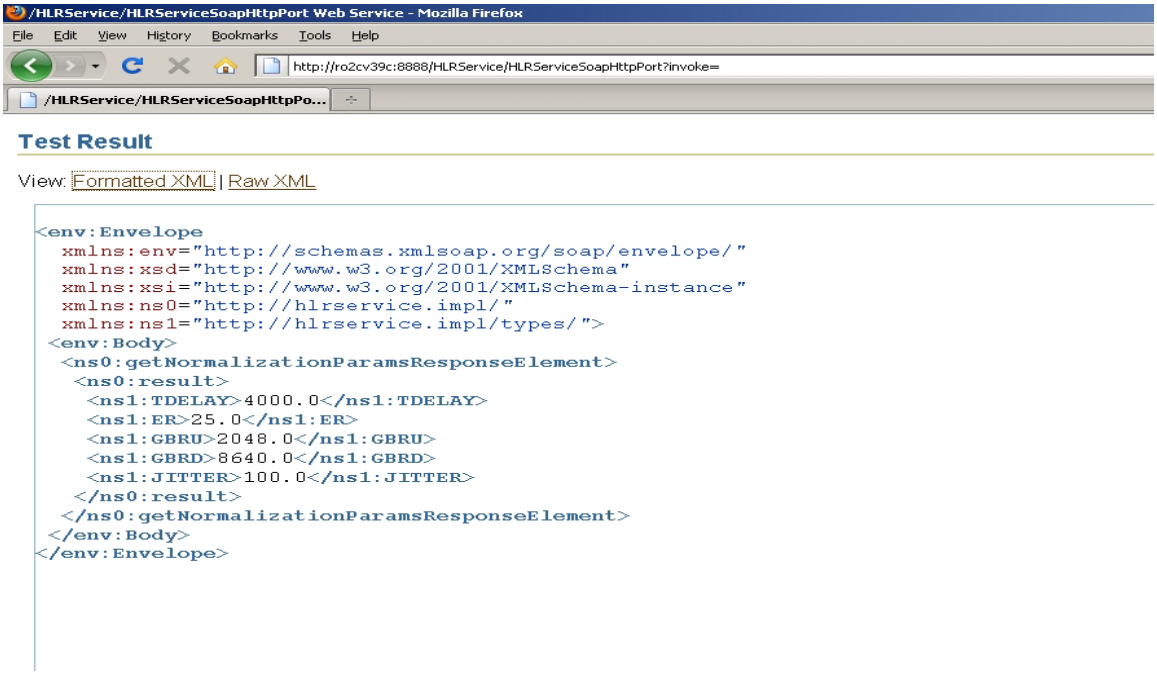
Acelaşi input poate fi utilizat pentru date de normalizare (cu toate acestea, aceşti numitori şi numărători pot fi introduşi direct în calculul QoS dublu-ponderat).



Serviciile HLR bazate pe SOAP

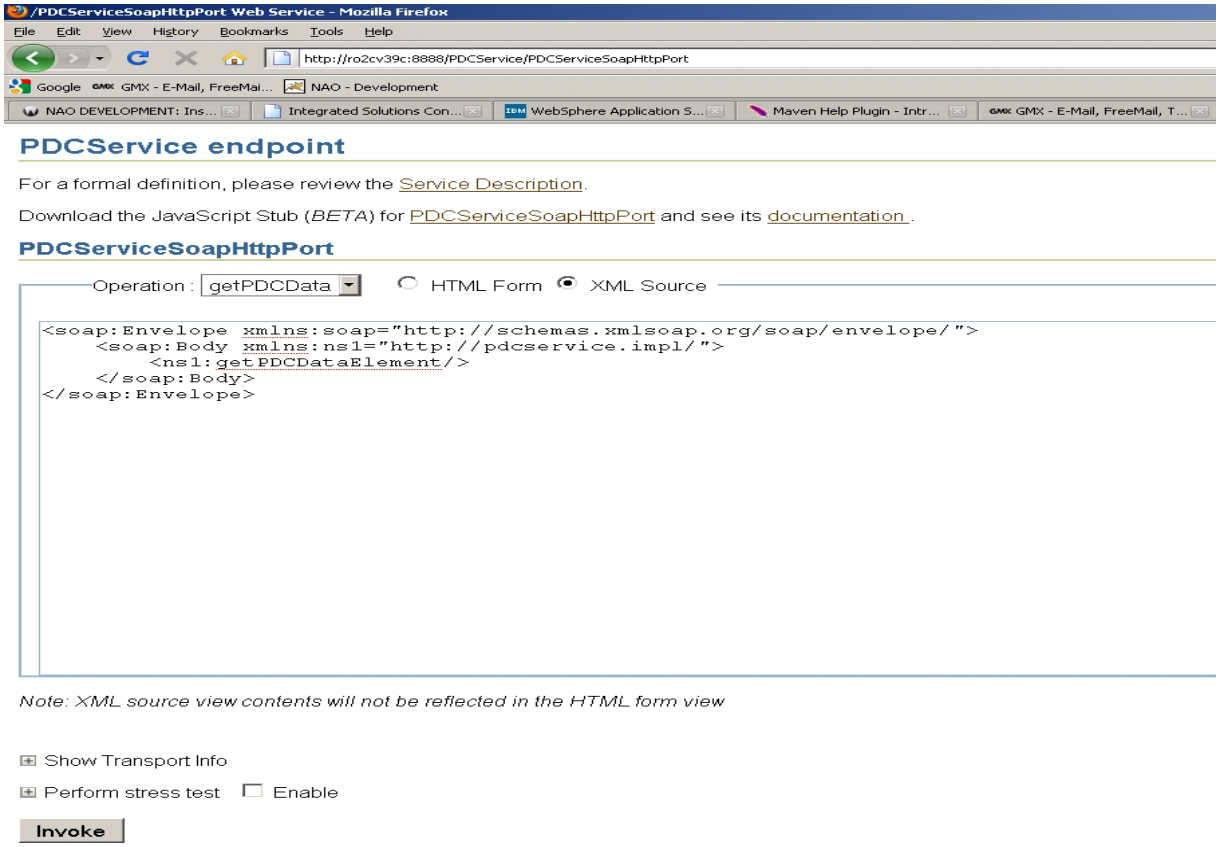


Preluarea datelor de normalizare

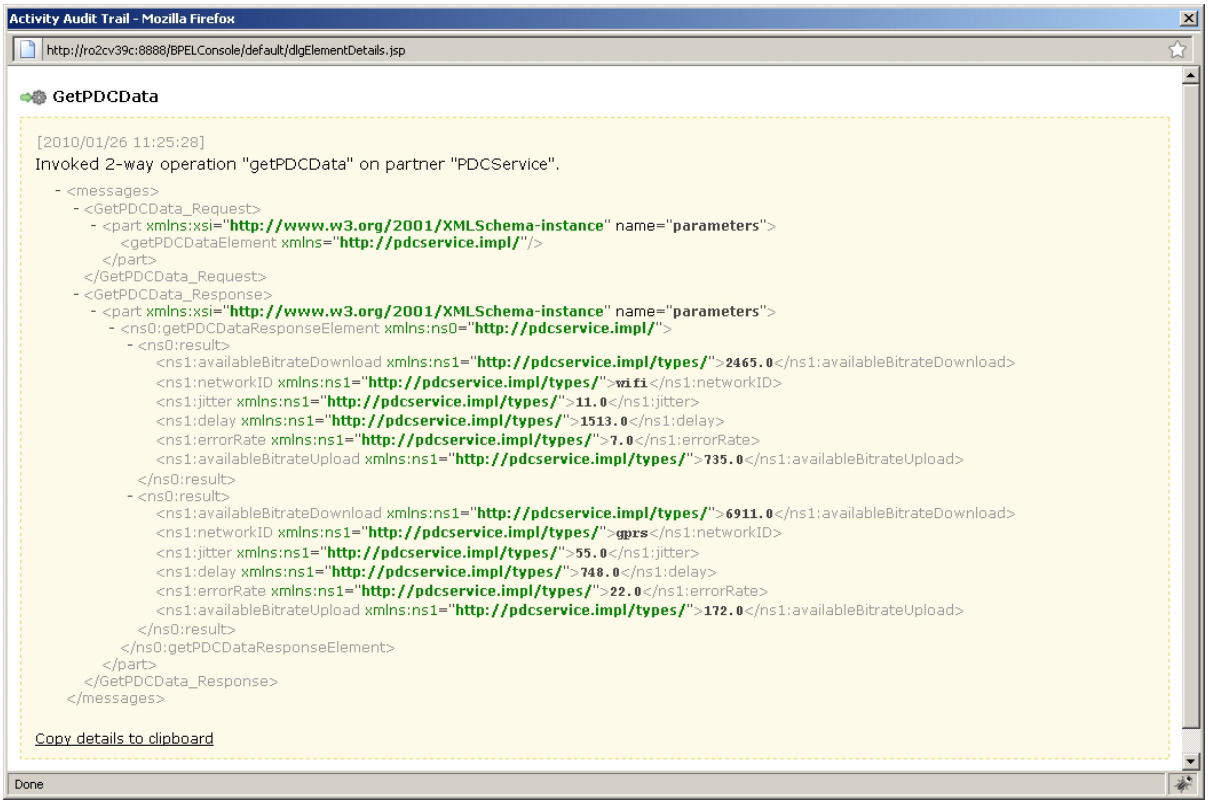


Rezultatele testului

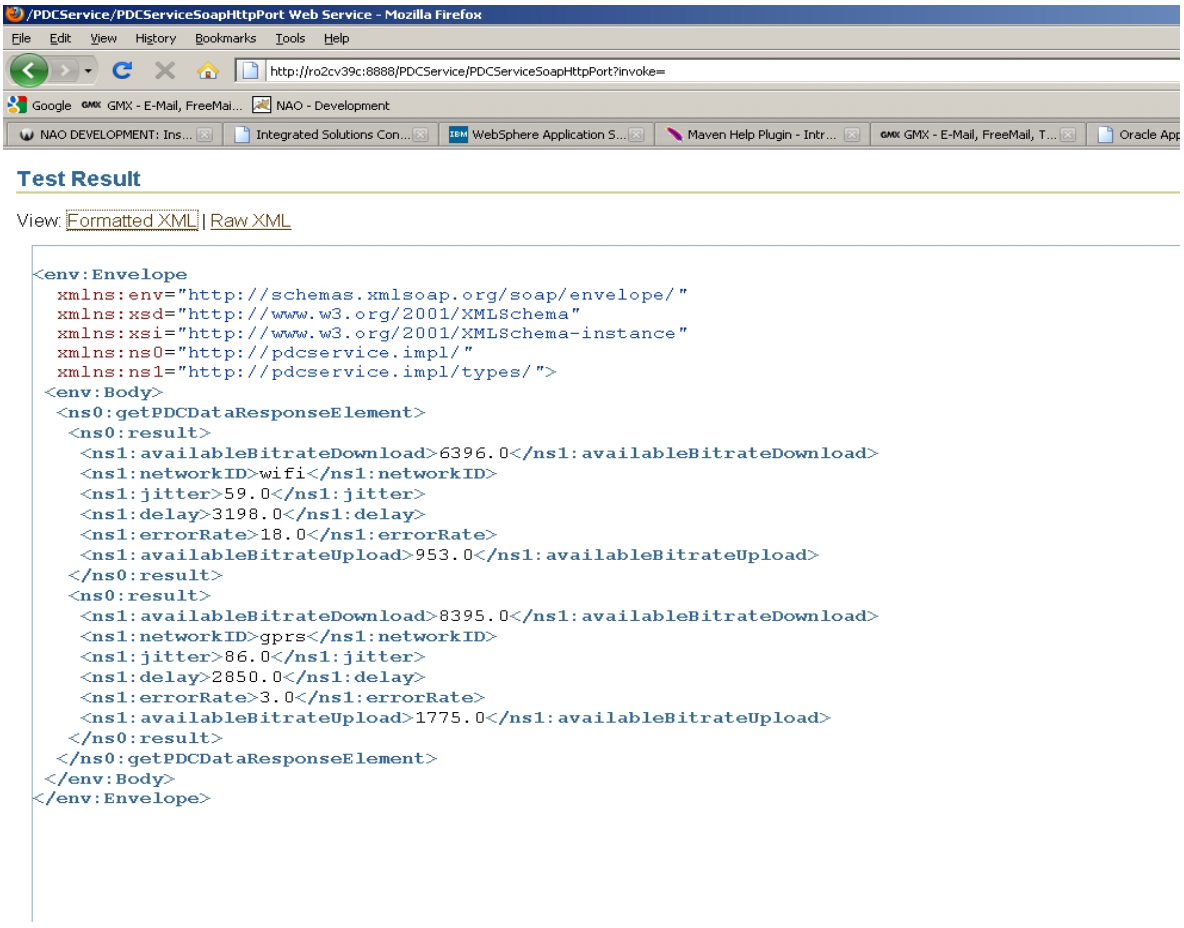
Deoarece un PDC se poate găsi numai la operatori, acesta se poate emula folosind mai multe seturi de date, unele dintre aceste date se pot vedea mai jos



Serviciul PDC – format XML



Preluarea datelor PDC



Rezultatele testului pentru serviciul PDC

Având aceste date, acum se poate calcula nivelul de QoS oferit de reţelele de transmisie şi apoi să se decidă care dintre reţele va ruta majoritatea traficului.

În evaluarea QoS disponibilă, fiecare valoare relevantă măsurată de Performance Data Collector (avbu – rata de biţi disponibila pentru upload; avbd – rata de biţi disponibilă pentru download, etc.) va fi întâi multiplicată cu o pondere reprezentând profilul abonatului (w_pr_), apoi cu o pondere reprezentând clasa de servicii, şi în cele din urmă normalizată (multiplicată sau împărţită) cu parametrii (np_) de normalizare de referinţă.

```
// evaluate pdc data
double qos =
w_pr_gbru * w_cl_gbru * avbu / np_gbru +
w_pr_gbrd * w_cl_gbrd * avbd / np_gbrd +
w_pr_tdelay * w_cl_tdelay * np_tdelay / delay +
w_pr_jitter * w_cl_jitter * np_jitter / jitter +
w_pr_er * w_cl_er * np_er / errorRate;
```

Ar putea fi, de asemenea, luat în considerare, într-un mod similar, QoS necesare aplicaţiei, cu un handover parţial la QoS cel mai bun actual calculat numai dacă această valoare scade sub QoS necesar (aceasta înseamnă că nu este niciun efort de handover pentru aplicaţii de cost mic, ce nu cer resurse multe).

Din RFC2326, se poate lua un exemplu de o aplicaţie de streaming care poate determina parametrii necesari de la un mesaj "DESCRIBE" trimis la serverul de streaming media:

```
C->W: GET /concert.sdp HTTP/1.1
  Host: www.example.com
W->C: HTTP/1.1 200 OK
  Content-Type: application/x-rtsl
<session>
<track src="rtsp://live.example.com/concert/audio">
</session>
C->M: DESCRIBE rtsp://live.example.com/concert/audio RTSP/1.0
  CSeq: 1
M->C: RTSP/1.0 200 OK
  CSeq: 1
  Content-Type: application/sdp
  Content-Length: 44
  v=0
  o=- 2890844526 2890842807 IN IP4 192.16.24.202
  s=RTSP Session
  m=audio 3456 RTP/AVP 0
  a=control:rtsp://live.example.com/concert/audio
  c=IN IP4 224.2.0.1/16
C->M: SETUP rtsp://live.example.com/concert/audio RTSP/1.0
  CSeq: 2
```

În mod similar, aplicaţiile VoIP pot determina cerinţele pentru bitrate, întârzieri/jitter şi rata de erori acceptabilă de la codec-urile folosite sa codeze datele de voce (la nivel de utilizator, de obicei, este folosit acelaşi protocol (RTP) pentru streaming şi pentru conferinţe voce/video).

În cazul în care nu sunt găsiţi toţi parametrii pentru o aplicaţie specifică (cum ar fi navigarea web pentru care întârzierea nu este foarte importantă şi jitter-ul nu este deloc relevant), se pot folosi valori implicite acceptabile. În mod ideal, dacă avem aplicaţii care rulează într-un nod mobil multimodal, ar trebui sa aibă măcar o descriere a profilului QoS care sa poată fi folosită pentru calcularea sumei ponderate.

Alternative în alegerea mediului de implementare software

S-a luat în considerarea şi posibilitatea de abordare BPEL cu suita „BizAgi". Am optat, totuşi, în primă instanţă, pentru modelarea proceselor folosind BPEL standard. Pentru modelare s-a folosit Oracle JDeveloper 10g.

Comparativ cu aceasta abordare, BizAgi are mici dezavantaje, nefiind o implementare standard a BPEL:

- BizAgi este dependent de Microsoft SQL Server;
- BizAgi ruleaza numai pe platforma Microsoft Windows, fiind necesară instalarea unui „Runtime Engine" pentru C++ şi a unuia de J# (în condiţiile în Microsoft nu va mai asigura nici măcar suport pentru limbajul J#, dezvoltarea fiind oprită de ceva vreme);
- la modelarea procesului cu ajutorul editorului furnizat de BizAgi, structura procesului şi toate configurările sunt salvate în baza de date, ceea ce îngreunează portarea pe alte platforme;
- sunt foarte puţine controale şi elemente care pot fi folosite pentru modelarea proceselor, lipsesc elemente foarte importante cum ar fi procesarea paralelă, posibilitatea de handling a erorilor, posibilitatea invocării unor servicii web partenere.

Totuşi, în final, s-a avut în vedere testarea portabilităţii serviciului implementat pe alte platforme BPEL – „logica de servicii", exprimată standard în format XPDL (XML for Process Description Languages), fiind preluată şi evaluată în context BizAgi sau „Intalio".

OpenWhisk - implementarea ”serverless” a serviciilor

Bluemix este o suită Cloud de tip PaaS (Platform as a service) dezvoltata de IBM. OpenWhisk ofera suport pentru multiple servicii si limbaje de programare, precum si DevOps (compound of development and operations) integrat pentru construirea, rularea, implementarea și gestionarea aplicațiilor pe cloud. Bluemix se bazeaza pe tehnologia open Cloud Foundry (o platforma ca serviciu, open source, folosita pentru cloud computing) si ruleaza pe infrastructura SoftLayer (platforma folosita pentru integrarea și automatizarea multiplor componente). Bluemix suporta mai multe limbaje si medii de programare, inclusiv Java, Node.js, Go, PHP, Swift, Python, Ruby Sinatra sau Ruby on Rails si poate fi extinsa pentru a oferi suport si altor limbaje, cum ar fi Scala prin utilizarea "buildpack"-urilor (servicii care oferta suport de rulare si executie pentru aplicatii).

IBM Bluemix include un serviciu de programare open source, sistem omolog cu Amazon Lambda sau Google Cloud Functions care permite apelarea unei funcții specifice ca răspuns la un eveniment, fără a necesita nici o gestionare a resurselor de catre dezvoltator.

OpenWhisk este un serviciu de programare in cloud bazat pe evenimente distribuite (cloud-first distributed event-based programming service), care permite executarea codului de programare ca răspuns la evenimente. OpenWhisk oferă un model de implementare și operațiuni fara server *(serverless)* care permit executarea de cod, fara a adminsitra infrastructura si a gestiona complexitatea acesteia.

Platforma OpenWhisk oferă un model de tarifare dedicat la orice nivel, oferind exact resursele necesare pentru codul executat. Este oferit un model de programare flexibil care include suport pentru limbaje de programare precum NodeJS sau Swift, chiar și pentru executarea logicii personalizate prin intermediul containerelor Docker. Acest lucru permite echipelor de dezvoltare să reutilizeze resurse deja existente pentru dezvoltarea aplicatiilor într-un mod cat mai optimizat scopului dorit. Un alt avantaj al serviciului sunt instrumentele folosite pentru a concatena blocurile dezvoltate.

OpenWhisk poate fi folosit atat pentru aplicatii si solutii web, mobile cat si IoT (Internet of Things), de exemplu, poate permite dezvoltatorilor de aplicatii si telefonie mobilă sa integreze propriile solutii cu logica back-end care rulează in cloud fără a instala o infrastructura de tip server sau middleware. Deoarece este disponibil pe IBM Bluemix, OpenWhisk ofera dezvoltatorilor web acces la functii adiacente precum servicii cognitive, analitice sau sarcini automatizate de tip DevOps.

Beneficii pentru procesul de dezvoltare

Arhitecturile de tip microservicii au apărut ca modalitate preferată de a proiecta soluții cloud native (cloud-native solutions) robuste si scalabile. Acestea incapsuleaza logica aplicatiilor in servicii distribuite de mici dimensiuni care comunica prin API-uri (Application Programming Interface). Chiar daca microserviciile sunt de dimensiuni reduse și independente necesita un cost operațional ridicat, cu privire la monitorizare, alertare, echilibrare a resurcelor, gestiune servere suplimentare pentru fiecare microserviciu component, atunci cand cele mai bune practici de tip DevOps nu sint urmate in dezvoltarea si implementarea acestora. In ciuda beneficiilor, soluțiile bazate pe microservicii rămân dificile de construit folosind tehnologii cloud care necesită de cele mai multe ori un flux de lucru complicat pentru a controla operatii si evenimente complexe.

Deoarece OpenWhisk introduce un nou mod de dezvoltare a solutiilor, lucrand in mod inerent "serverless" (fara server) elimina aceste probleme abstractizand complexitatea microserviciilor, oferind un serviciu de calcul distribuit pentru a executa logica de aplicare ca răspuns la evenimente.

Principalul avantaj al OpenWhisk este că "ascunde" complexitatea infrastructurală permitand dezvoltatorilor să se concentreze pe logica de afaceri. Dezvoltatorii nu trebuie să isi faca griji cu privire la detalii low-level, cum ar fi scalarea, echilibrarea sarcinii, autentificarea, toleranța la erori sau sirurile de mesaje (message queues). Platforma sprijină un ecosistem deschis, care permite partajarea microserviciilor prin intermediul pachetelor OpenWhisk. Dezvoltatorii pot construi soluții folosind abstractizari si înlănțuiri moderne folosind multiple medii de rulare (NodeJS, Swift etc.)

Astfel, serviciile definesc evenimentele pe care le emit ca declanșatori, iar dezvoltatorii definesc acțiunile să se ocupe de evenimente. Dezvoltatorul are nevoie doar să aibe grijă la punerea în aplicare a logicii aplicației dorite, iar sistemul se ocupă de restul.

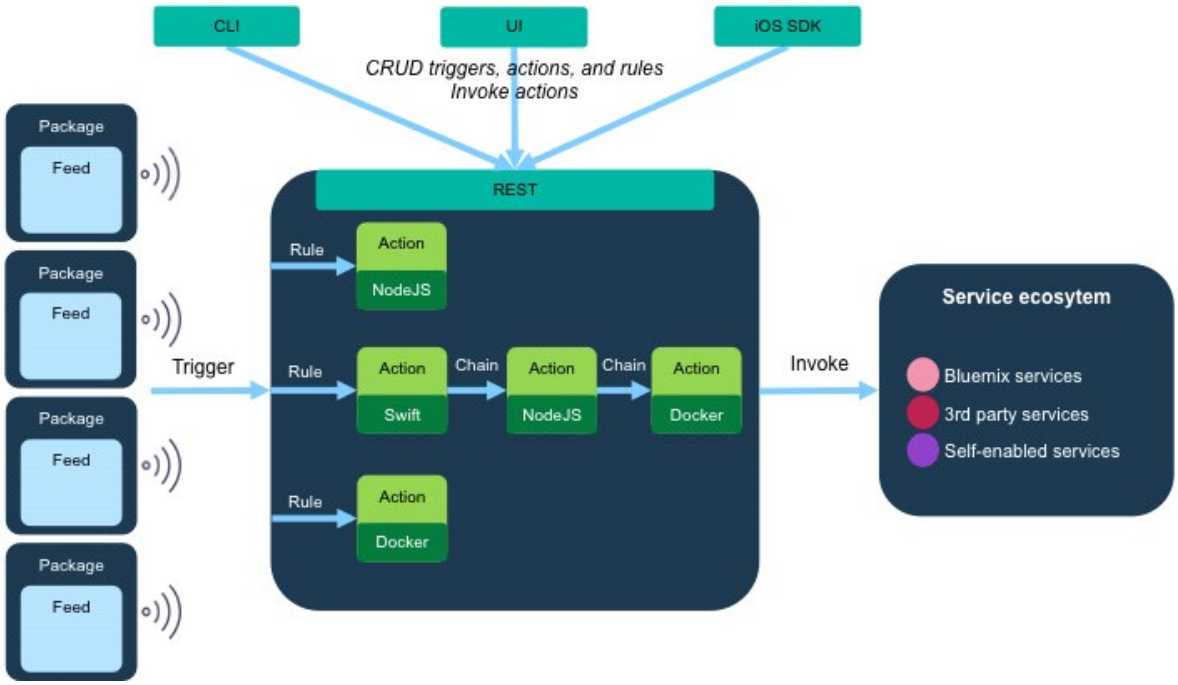
Structura OpenWhisk

OpenWhisk oferă un serviciu de calcul distribuit pentru a executa logica de aplicare ca răspuns la evenimente. Există mai multe concepte arhitecturale-cheie:

- Declanșările (Triggers) - o clasă de evenimente, emise de surse de evenimente;
- Acțiuni (Actions) - încapsuleaza codul care urmează să fie executat, cu suport pentru multiple limbaje de programare si medii de rulare, inclusiv NodeJS, Swift și programe binare arbitrare încapsulate în containere Docker. Acțiunile invocă

orice parte a unui ecosistem deschis, inclusiv serviciile Bluemix existente pentru analiză, sisteme cognitive, sau orice alt serviciu terț (third party service);

- Reguli (Rules) - asocieri între declanșatori și acțiuni;
- Pachete (Packages) - descriu serviciile externe într-un mod uniform.



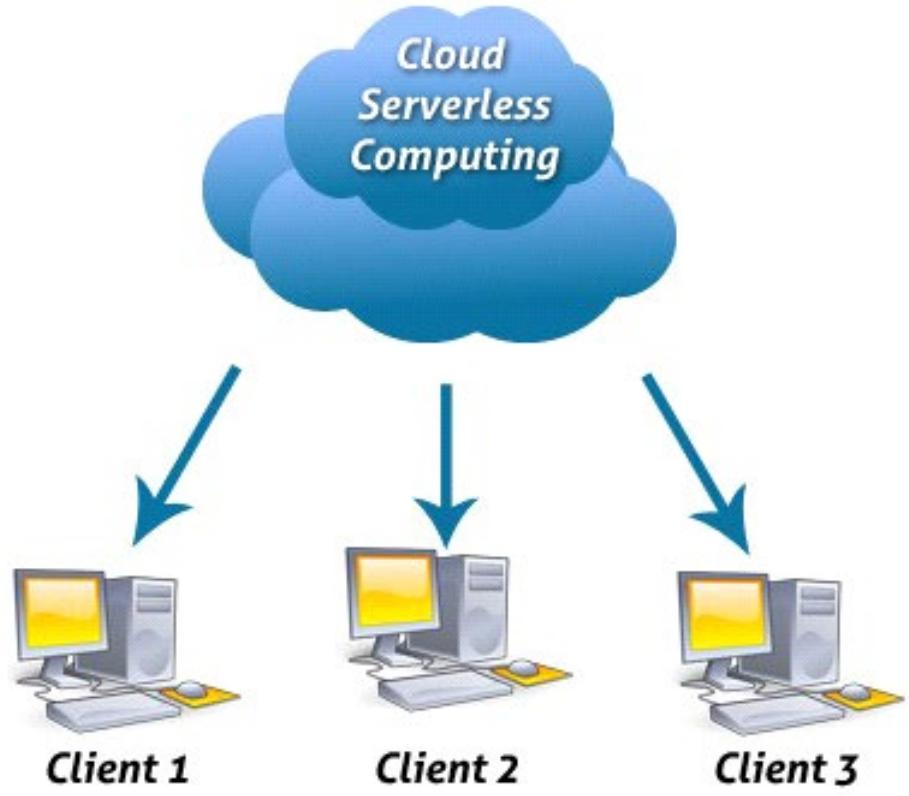
Structura OpenWhisk

Serverless Computing (Tehnica de calcul fara server)

Conceptul de "Serverless Computing" se referă la un model în care existența serverelor este pur și simplu ascunsa dezvoltatorilor (chiar dacă serverele există în continuare dezvoltatorii sunt scutiți de necesitatea administratii, gestionarii si funcționarii acestora). Detaliile operationale, infrastructura de securitate, scalabilitatea si disponibilitatea sunt acum eliminate din grija dezvoltatorilor. Conceptul de "Serverless Computing" este orientat în esență catre reducerea eforturilor de întreținere, pentru a permite dezvoltatorilor să se concentreze strict si rapid pe dezvoltarea codului de valoare adăugată.

Tehnica de calcul fara server încurajează și simplifică dezvoltarea de soluții orientate spre microservicii, în scopul de a descompune aplicații complexe în module mici și independente, care pot fi ușor schimbate.

Folosind clasicele mașini sau containere virtuale de lungă durată (long-running virtual machines or containers), este o practică comună implementarea mai multor containere/mașini virtuale care sa fie rezistente la întreruperi (fie multiple, fie ale unei singure instanțe). Cu toate acestea, OpenWhisk oferă un model alternativ, executarea la cerere a acțiunilor oferind scalabilitate inerenta și o utilizare optimă a resurselor deoarece numărul de acțiuni coincide întotdeauna cu evenimentele de declanșare. In plus, dezvoltatorul se concentrează acum doar pe cod și nu pe monitorizarea, stocarea, securizarea serverului, precum și infrastructura retelei si a sistemului de operare.



Serverless Computing (Tehnica de calcul fara server)

Integrarea de servicii suplimentare și furnizori de evenimente pot fi adăugate folosind pachete de fluxuri și acțiuni. Un flux este o bucată de cod care configurează o sursă de eveniment extern la un eveniment de declanșare. Acțiunile în pachete reprezintă logică reutilizabilă astfel un furnizor de servicii poate pune la dispoziție dezvoltatorilor nu numai serviciul ca sursă de eveniment, dar si invocarea de API-uri ale acelui serviciu.

Avantaje si cazuri de utilizare:

- **Descompunerea aplicațiilor în microservicii** - Natura modulară și inerent scalabila a serviciului OpenWhisk il face potrivit pentru punerea în aplicare a granularitatii de logică computatiuonala în aplicatii. De exemplu, OpenWhisk poate fi util pentru îndepărtarea sarcinilor complexe de back-end (de fundal) din codul de front-end și punerea în aplicare a acestor sarcini ca acțiuni;
- **Back-end pentru aplicații mobile -** multe aplicații mobile necesită logica server-side. Având în vedere faptul că dezvoltatorii de aplicații mobile, de obicei, nu au experiență în gestionarea logică de tip server-side și se vor concentra mai degrabă pe aplicatii care se execută pe dispozitiv, folosirea OpenWhisk ca back-end server-site este o soluție bună. In plus, suportul pentru limbajul Swift permite dezvoltatorilor iOS să utilizeze cod nativ in aplicatii;
- **Procesarea de date** - OpenWhisk oferta abilitatea de a procesa date noi, și potențial să reacționeze la acestea (incluzand atât procesarea de înregistrări structurate de baze de date precum și documente nestructurate, imagini sau clipuri video);
- **Internet of Things** - in conceptul de IoT (Internetul lucrurilor) obiectele sunt adesea bazate pe senzori. De exemplu, o acțiune în OpenWhisk poate fi declanșată dacă există necesitatea de a reacționa la un senzor care depășește o anumită temperatură.

Instalarea si Configurarea OpenWhisk:

Instalarea OpenWhisk dintr-un depozit (repository) clonat va avea ca rezultat generarea de unei interfete de tip linie de comandă (CLI). Intefata generata va fi localizata în `openwhisk/bin/go-cli/`. Interfata CLI este prezenta sub forma de executabil localizat în directorul menționat mai sus care va rula pe sistemul de operare și arhitectura procesorului pe care a fost construit. Executabilele pentru alte sisteme de operare și arhitecturi CPU sunt localizate în următoarele directoare:

- `openwhisk/bin/go-cli/mac` (pentru Apple Mac);
- `openwhisk/bin/go-cli/linux` (pentru Linux);
- `openwhisk/bin/go-cli/windows` (Windows).

Pentru a descărca interfata CLI de la o implementare existenta, trebuie să se descărce executabilul utilizând URL-ul de bază al instalarii acestuia. Lista interfetelor CLI care pot fi descărcate pentru diferite sisteme de operare și arhitecturi CPU pot fi obținute din locația `{BASE URL}/cli/go/download`. Structura `{BASE URL}` este numele de

gazda (hostname) al API-ului OpenWhisk si adresa IP a acestuia (exemplu, `openwhisk.ng.bluemix.net`).

Interfata CLI foloseste pentru configurarea a trei proprietati:

- **API host** - gazdă API (numele sau adresa IP) pentru implementarea OpenWhisk dorita;
- **Authorization key** - cheia de autorizare (nume de utilizator și parola), care acordă acces la API-ul OpenWhisk;
- **Namespace** - unde sunt stocate configurarile OpenWhisk.

Interfata CLI va avea de obicei o gazdă API deja stabilită. Valoarea acesteia se poate verifica prin comanda `wsk property get --apihost`.

Dupa aflarea cheii de autorizare si a namespace-ului, trebuie configurata interfata CLI ca să le folosească. În caz contrar, vor trebui furnizate in momentul executarii operatiilor folosind interfata.

Exemplu: `wsk property set [--apihost <openwhisk_baseurl>] --auth <username:password> --namespace <namespace>`

Gazda API-ului (API host) este setată automat atunci când se construieste CLI-ul pentru mediul de executie (execution environment). Un cont de oaspete (guest account) este disponibil în versiunile instalate local, cu o cheie de autorizare situata în `ansible/files/auth.guest` impreuna cu namespace-ul `guest`. Pentru a configura CLI-ul să utilizeze contul `guest`, exista posibilitatea executării următoarei comenzi din directorul `openwhisk`:

```
./bin/wsk property set --namespace guest --auth `cat ansible/files/auth.guest`
```

Pentru a verifica configurarea interfetei CLI, se poate crea și executa o acțiune.

Folosirea interfetei CLI in OpenWhisk

Dupa ce mediul de executie a fost configurat, interfata CLI OpenWhisk se poate folosi pentru:

- Rularea/executarea fragmentelor de cod, sau crearea si invocarea acțiunilor;
- Folosirea declanșatorilor (triggers) si a regulilor (rules) pentru a permite acțiunilor să răspundă la evenimente;

- Executarea pachetelor de acțiuni și configurarea surselor de evenimente externe;
- Explorearea catalogului de pachete și înglobarea aplicațiilor cu serviciile externe, cum ar fi o sursă de eveniment Cloudant (Cloudant event source - DBaaS - distributed database as a service);

Configurarea CLI pentru a utiliza un proxy HTTPS

Interfata CLI in OpenWhisk poate fi configurata pentru a utiliza un proxy HTTPS. Pentru a configura un proxy HTTPS, trebuie să fie creată o variabilă de mediu numit `https_proxy`. Variabila trebuie setată pentru adresa proxy HTTPS impreună cu portul acesteia, utilizând următorul format: `{PROXY IP}:{PROXY PORT}`.

Folosirea de API-uri REST in OpenWhisk

Dupa ce mediul OpenWhisk este activat, acesta poate fi utilizat si integrat in aplicații web sau mobile folosind cereri de tip API REST pentru *actiuni*, *namespace*-uri, *pachete*, *reguli* sau *declansatori*:

- **Actions (actiuni):**
  - **GET -** `/namespaces/{namespace}/actions` - Obține toate acțiunile;
  - **DELETE** - `/namespaces/{namespace}/actions/{actionName}`- Șterge o acțiune;
  - **GET -** `/namespaces/{namespace}/actions/{actionName}` - Obține informații despre acțiuni;
  - **POST -** `/namespaces/{namespace}/actions/{actionName}` - Invoca o acțiune;
  - **PUT -** `/namespaces/{namespace}/actions/{actionName}` - Creaza sau actualizeaza o acțiune.
- **Activations (activări):**
  - **GET -**`/namespaces/{namespace}/activations` - Preia ID-urile activărilor;

- **GET -** `/namespaces/{namespace}/activations/{activationid}` - Obține informațiile activarilor;
- **GET -** `/namespaces/{namespace}/activations/{activationid}/logs` - Obține jurnalele (logs) pentru o activare;
- **GET -** `/namespaces/{namespace}/activations/{activationid}/result`- Preia rezultatul unei activari.

- **Namespaces (spații de nume):**
  - **GET -** `/namespaces` - Obține toate "namespace"-urile pentru un utilizator autentificat;
  - **GET -** `/namespaces/{namespace}` - Obține toate entitățile într-un "namespace".
- **Packages (pachete)**:
  - **GET -** `/namespaces/{namespace}/packages` - Obține toate pachetele;
  - **DELETE -** `/namespaces/{namespace}/packages/{packageName}` - Șterge un pachet;
  - **GET -** `/namespaces/{namespace}/packages/{packageName}` - Obține informații despre pachet;
  - **PUT -** `/namespaces/{namespace}/packages/{packageName}` - Crează sau actualizează un pachet.
- **Rules (reguli):**
  - **GET -** `/namespaces/{namespace}/rules` - Obține toate regulile;
  - **DELETE -** `/namespaces/{namespace}/rules/{ruleName}` - Șterge o regulă;
  - **GET -** `/namespaces/{namespace}/rules/{ruleName}` - Obține informatiile regulilor;

- **POST -** `/namespaces/{namespace}/rules/{ruleName}`**-** Activează sau dezactivează o regula;
- **PUT -** `/namespaces/{namespace}/rules/{ruleName}` - Actualizează o regulă.

- **Triggers (Declanșatori):**
  - **GET** - `/namespaces/{namespace}/triggers` - Obține toți declanșatorii;
  - **DELETE** - `/namespaces/{namespace}/triggers/{triggerName}` - Șterge un declanșator;
  - **GET** - `/namespaces/{namespace}/triggers/{triggerName}` - Obțineți informațiile declanșatorilor;
  - **POST** - `/namespaces/{namespace}/triggers/{triggerName}` - Execută un declanșator;
  - **PUT** - `/namespaces/{namespace}/triggers/{triggerName}` - Actualizează un declanșator;

Exemplu "Bine ai venit!" in OpenWhisk:

Se execută urmatorul cod JavaScript in OpenWhisk:

```
/**
 * Bine ai venit in OpenWhisk!
 */

function main(params) {
    var name = params.name || 'in OpenWhisk';
    return {payload:  'Bine ai venit ' + name + '!'};
}
```

Pentru a utiliza acest exemplu, se parcurg urmatorii pași:

- Se salveaz ă codul într-un fișier. De exemplu, `salut.js`.
- Din linia de comandă CLI OpenWhisk, se crează acțiunea introducând urmatoarea expresie:

```
$ wsk action create hello salut.js
```

- Apoi se invocă acțiunea prin introducerea următoarelor comenzi:

```
$ wsk action invoke hello --blocking --result
```

Această comandă returnează:

```
{
  "payload": "Bine ai venit in OpenWhisk!"
}
```

```
$ wsk action invoke hello --blocking --result --param name
George
```

Această comandă returnează:

```
{
  "payload": "Bine ai venit George!"
}
```