Sviluppo di servizi WEB per Smart Area

Loredana Versienti[1]

[1] ISTI-CNR, Via G. Moruzzi 1, 56124 Pisa, versienti@isti.cnr.it

Abstract. Questo articolo descrive l'architettura e l'implementazione di web service RESTful per l'esposizione di servizi che permettono la gestione di dati provenienti da sensori smart. L'uso dello standard W3C dei web service garantisce l'interoperabilità tra le diverse applicazioni coinvolte nella fruizione dei dati, mentre l'estrapolazione della conoscenza dai dati è resa possibile grazie ad un'opportuna ontologia gestita all'interno di un database Virtuoso.

Keywords: REST, ROA**,** RESTful, Smart, Virtuoso, SPARQL, Apache Tomcat

Introduzione

I web service, secondo la definizione fornita dal W3C, forniscono un mezzo standard per assicurare l'interoperabilità tra diverse applicazioni software, astraendo dalla piattaforma hardware su cui sono installati. In questo contesto si collocano i web service RESTful sviluppati nel progetto Smart Area, dove diverse applicazioni interagiscono in modo intelligente scambiandosi dati l'una con l'altra.

Stato dell'arte

L'architettura dei web service ha spostato la concezione del web dal modello centrato sull'utente, verso un modello *application-centric*, dove la comunicazione avviene direttamente tra applicazioni, portando così non solo ad uno scambio di informazioni, ma anche ad una condivisione di risorse computazionali che vengono utilizzate a richiesta. Questo modello è rappresentato da una architettura denominata SOA (Service-Oriented Architecture)[1] e la sua implementazione più diffusa ad oggi include tutte quelle tecnologie che vengono classificate come Web Service. Tale architettura focalizza l'attenzione sul concetto di servizio ed è ovvio immaginare come gli attori in causa siano necessariamente il fornitore ed il fruitore, secondo il medesimo paradigma che si riscontra nella tipica interazione *client-server*. L'approccio adottato dalle SOA ha il vantaggio di potersi integrare con diversi ambienti, permettendo in tal modo di realizzare applicazioni *multi-canale* fruibili tramite dispositivi diversi. Questo avviene mediante uno scambio di informazioni descritte su un formato XML. Esistono molte implementazioni tecnologiche che rispecchiano i concetti dell'architettura orientata ai servizi, ma non tutte riescono ad esprimere il concetto fondamentale di implementare entità che descrivano se stesse, tranne la tecnologia che sfrutta il protocollo chiamato SOAP (Simple Object Access Protocol) [2] e definito per assunto, fino a qualche tempo fa, come il "Web Service".

SOAP (Simple Object Access Protocol) è un protocollo leggero che fornisce un meccanismo semplice per lo scambio di messaggi tra componenti software, tramite XML. La parola Object contenuta nell'acronimo, manifesta che l'uso del protocollo dovrebbe effettuarsi secondo il

paradigma della programmazione orientata agli oggetti, infatti SOAP descrive un meccanismo semplice per esprimere la semantica dell'applicazioni fornendo dei meccanismi per la codifica dei dati all'interno dell'entità coinvolte. SOAP è un framework che può operare sopra varie pile protocollari e le chiamate alle procedure remote possono essere modellate come uno scambio di messaggi SOAP. La struttura di un messaggio SOAP è costituita da quattro componenti principali:

- "SOAP Envelope" è l'elemento principale, rappresenta la busta del messaggio che funge da raccoglitore di tutti gli altri elementi.

- "SOAP Header" è un elemento che contiene le informazioni specifiche dell'applicazione. E' opzionale, il suo scopo è quello di trasportare informazioni non facenti parte del messaggio, destinate agli 'attori', cioè alle varie parti che il messaggio attraverserà per arrivare al suo destinatario finale.

- "SOAP Body" è un elemento indispensabile che è il vero deputato a contenere le informazioni applicative che è necessario comunicare al ricevente.

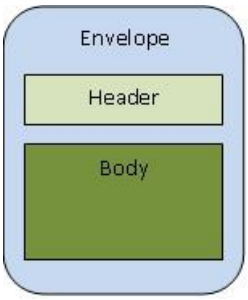

Fig.1: struttura di un messaggio SOAP

La busta del messaggio SOAP così strutturata viaggia su un canale di trasmissione utilizzato per invocare la procedura, proponendo come standard l'utilizzo del protocollo HTTP[3], senza tuttavia precludere l'utilizzo di altri protocolli, a patto che possano trasportare testo.

A differenza di HTTP, però, le specifiche di SOAP non affrontano argomenti come la sicurezza o l'indirizzamento, per i quali sono stati definiti standard a parte, nello specifico *WS-Security* e *WS-Addressing*: SOAP non sfrutta a pieno il protocollo HTTP, ma lo utilizza come semplice protocollo di trasporto. I web service basati su SOAP prevedono lo standard WSDL[4], *Web Service Description Language*, per definire l'interfaccia di un servizio. Questa è una chiara evidenza del tentativo di adattare al Web l'approccio di interoperabilità basato su chiamate remote (come DCOM, CORBA e RMI). Infatti il WSDL non è altro che un IDL (Interface Description Language) per un componente software e quindi da un lato favorisce l'uso di tool per creare automaticamente client in un determinato linguaggio di programmazione, ma allo stesso tempo induce a creare una forte dipendenza tra client e server.

Un'alternativa architetturale più recente è l'architettura ROA (Resource Oriented Architecture)[5], contrapposta alla SOA ed ispirata allo stile architettonico REST (REpresentational State Transfer)[6] consistente in un insieme di principi e metodi di progettazione che delineano come le risorse sono definite e indirizzate. I web service RESTful

implementano l'architettura ROA in quanto rispettano i vincoli fondamentali espressi dallo *stile* REST:

I Vincoli REST	
I	Client/Server
II	Comunicazione Stateless
III	Cacheable
IV	Codice a richiesta (Opzionale)
V	Interfaccia comune
VI	Organizzazione a livelli

Fig.2: vincoli REST

I. Client-Server: un'interfaccia separa i Client dai Server, questa separazione di rapporti significa che vengono separate le competenze, in tal modo migliora la portabilità e la scalabilità.

II. Stateless: ogni ciclo di request/response deve rappresentare un'interazione completa tra client e server. Ogni nuova richiesta è indipendente da quelle precedenti, pertanto deve contenere tutte le informazioni necessarie per la gestione. In questo modo non è necessario mantenere informazioni sulla sessione utente, minimizzando l'uso di memoria del server e la sua complessità. Rafforza la scalabilità, l'affidabilità e la visibilità.

III. Caching: i client possono memorizzare in cache le risposte, questo permette di eliminare parzialmente o completamente alcune interazione tra Client e Server, migliorando la scalabilità e le performance.

IV. Code on-demand: un componente Client ha accesso a un insieme di risorse, ma non al know-how su come elaborarle. Invia una richiesta a un Server remoto per il codice che rappresenta il know-how, ricevuto il codice viene eseguito in locale. Questo semplifica il lavoro dei Client e aumenta le possibilità di estensione.

V. Uniform interface: ogni risorsa deve avere un indirizzo univoco e ogni risorsa di ogni sistema presenta la stessa interfaccia, precisamente quella individuata dal protocollo HTTP. Semplifica l'architettura e fa sì che ogni parte si evolva indipendentemente.

VI. Layered system (Sistemi a più livelli): un Client non può, normalmente, dire se è connesso direttamente al Server oppure a uno stato intermedio. I Server intermedi sono in grado di migliorare la scalabilità del sistema rendendo il carico bilanciato e provvedendo alla condivisione della cache. Questo inoltre rafforza le politiche di sicurezza.

L'architettura ROA, a differenza della SOA, focalizza l'attenzione sul concetto di risorsa ed in tal modo si sposta la complessità dal protocollo alla rappresentazione astratta dello stato di una risorsa; naturalmente non è necessario trasmettere tutto lo stato ma solo quelle porzioni che interessano. Da specificare è che nell'architettura Web i singoli servizi sono considerati "risorse", individuate da URI (Uniform Resource Identifier)[7], tramite il protocollo HTTP che

è fornito di un unico schema d'indirizzamento. Un URI è un identificatore univoco di una particolare risorsa in rete, come ad esempio un documento HTML, un'immagine, un video oppure un servizio web. Un particolare URI è l'URL (Uniform Resource Locator)[8] che fornisce maggiori informazioni sulla risorsa referenziata, ovvero dove la stessa si trova fisicamente ed il suo tipo di dato.

La Risorsa (resource) è l'elemento chiave di un design RESTful, in opposizione ai "metodi" ed ai "servizi" rispettivamente usati nelle richieste SOAP dei servizi Web. Una risorsa è una qualunque entità che possa essere indirizzabile tramite Web, cioè accessibile e trasferibile tra Client e Server e spesso rappresenta un oggetto appartenente al dominio del problema che stiamo trattando.

Un servizio web basato sulla metodologia REST, ha come punto di forza l'utilizzo di un interfaccia che è diffusamente conosciuta: URI. Qualsiasi Client e Server che supporta invocazioni HTTP può facilmente invocare un servizio REST, il quale sfrutta HTTP per quello che è e cioè un protocollo di livello applicativo, e ne utilizza appieno le potenzialità. L'approccio REST non prevede esplicitamente nessuna modalità per descrivere come interagire con una risorsa poichè le operazioni sono implicite nel protocollo HTTP.

Nei servizi REST le richieste, vengono indirizzate verso URI differenti che si mappano sulle risorse. Il consumo di banda che porta un servizio REST è ridotto al minimo, infatti viene inviato insieme alla richiesta un documento XML o altre informazioni oltre all'URI solo quando bisogna creare o aggiornare lo stato di una risorsa; per quello che concerne la sicurezza invece, nella comunicazione di un servizio REST, apparati come firewall sono in grado di discernerne l'intento per ciascun messaggio, analizzando il comando HTTP utilizzato nella richiesta. Ad esempio, una richiesta GET può sempre essere considerata sicura, in quanto non può, per definizione, modificare nessun dato. Dall'altra parte invece, una tipica richiesta SOAP utilizza sempre il metodo POST per comunicare con i servizi e senza un'analisi completa del controllo dei messaggi, non si è in grado di predire se è una richiesta che può modificare le informazioni sul Server e quindi attuare gli opportuni meccanismi di controllo.

Infine per la parte di autenticazione e autorizzazione, SOAP utilizza una sua metodologia di autenticazione inserita nella specifica del protocollo di comunicazione demandando tutto alla fase di progettazione del servizio. La metodologia REST, invece, si basa sul fatto che i Web Server supportano già questo tipo di operazioni tramite l'uso di standard come lo scambio di certificati e comuni sistemi di gestione dell'identità (come ad esempio un LDAP server). Questo disaccoppiamento può aiutare lo sviluppatore del servizio a demandare la problematica dell'autenticazione ad applicazioni esterne che sono facilmente integrabili con l'applicazione RESTful.

Applicazione Smart Open Data

Il principale obbiettivo del progetto SOD è di supportare l'interoperabilità semantica delle applicazioni che sono state sviluppate all'interno del progetto Smart Area (SA), permettendo alle applicazioni di interagire l'una con l'altra in un modo intelligente, basato sulla

condivisione della semantica dei dati scambiati. Per raggiungere questo obbiettivo è stata sviluppata una ontologia che coinvolge diverse entità, e supporta le applicazioni coinvolte nel progetto a mappare i loro dati verso l'ontologia. Per maggiori dettagli che riguardano l'ontologia fare riferimento al documento *"SMART AREA OF CNR IN PISA, Italy"*[9].

L'architettura del sistema

Il sistema è caratterizzato da tre componenti principali in cui distinguiamo il *client* attraverso il quale le applicazioni invocano i servizi, i *servizi web* che processano le richieste e forniscono le informazioni al richiedente ed il *database* Virtuoso[10] in cui sono memorizzati i dati inviati dalle diverse applicazioni:

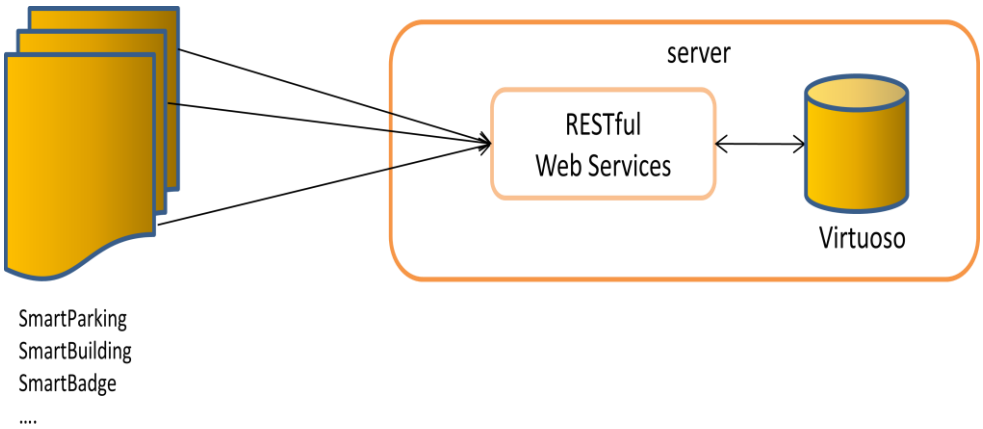

Fig.3 componenti principale del sistema

Nel seguito del documento vengono presentati e discussi in modo approfondito i servizi web.

Web Service

I web service rappresentano la logica dell'applicazione all'interno del sistema, sono stati sviluppati utilizzando il framework open source Jersey [11], secondo il modello architetturale REST. Tutte le comunicazioni tra il client e i servizi web sono eseguite con le richieste standard HTTP, mentre lo scambio di messaggi è stato codificato utilizzando il formato JSON [12].

Le caratteristiche dei web service sviluppati si possono raggruppare in due principali categorie, quelli di inizializzazione detti servizi di *init* è quelli di caricamento dei dati. Nella prima categoria ricadono i servizi web che devono essere invocati ogni volta che un nuovo evento si scatena e nello specifico sono:

- ✓ **registraApplicazione**

Ogni qual volta che una nuova applicazione inizia a far parte del progetto SA, essa deve registrasi nel repository semantico. Questa operazione è richiesta solo una volta.

```
@GET
```

```
@Path("/applicationName/{idApp}")
@Produces("application/json")
public String  registraApplicazione(@PathParam("idApp") String idApp) {
       String message ="";
       ......
}
```

Il metodo aggiorna il repository semantico andando a creare una relazione detta "`DatatypeProperty`" tra la classe Applicazione e la stinga che rappresenta il nome dell'applicazione.

- ✓ **registraSensore**

Registra sensore viene chiamato ogni qualvolta un nuovo sensore viene installato. La tipologia del sensore può essere varia, difatti le applicazioni sviluppate all'interno del progetto trattano sensori di diversa natura quali: sensori di parcheggio, sensori di rilevamento presenza umana, sensori di temperatura, sensori di umidità, eccetera, collocati all'interno di uffici o in aree più ampie come ad esempio l'atrio.

```
@POST
@Path("/sensorRegister")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces({"application/json","application/xml"})
public String registraSensore(@FormParam("sensoreStringGson") String sensoreStringGson) {
          String message ="";
          .......
}
```

Tutte le volte che il metodo viene invocato una serie di operazioni che vanno a modificare il registro semantico vengono eseguite. Il metodo accetta in ingresso una stringa che è deserializzata con l'uso dei JavaBean e la tecnica è conosciuta con il nome di Plain Old Java Object (POJO). I JavaBean sono dei componenti software riutilizzabili ed implementano la logica dei componenti sia a livello server, sia a livello client. Le proprietà dichiarate all'interno del bean sono**:**

```
String idLocalSensor;
String geoPosition;
String characteristic;
String idApp;
```

La proprietà `idApp` è fondamentale all'interno del metodo, a partire da questa nel modello viene individuata quella parte di grafo che andrà ad essere modificata con l'aggiunta di nuove relazioni. La proprietà `geoPosition` permette di individuare dove è allocato il sensore, ad esempio la stanza I40, mentre la proprietà `characteristic` è una stringa che contiene la descrizione del sensore come ad esempio *l'idMacAddress*.

- ✓ **associaSensoriAreaGeografica**

Il metodo viene invocato per associare il sensore con l'area geografica. Con area geografica intendiamo una porzione fisica che può essere una stanza, uno slot (nel caso di un parcheggio) o uno spazio aperto se si tratta di un atrio. Anche in questo caso si utilizzano i JavaBean, le cui proprietà sono:

```
String idApp;
ArrayList<String> areeGeograficheList;
ArrayList<String> sensoriList;
```

Anche quà la proprietà `idApp` è fondamentale per discernere all'interno del grafo la porzione che deve essere aggiornata. L'elemento della lista areeGeografiche individua un'entità fisica come un ufficio od uno slot; tale entità può essere monitorata da più sensori, come ad esempio un sensore di temperatura, di rumore, eccetera.

Nella seconda categoria troviamo il metodo utilizzato per caricare i dati rilevati dai sensori all'interno del registro semantico.

- ✓ **caricaDati**

```
@POST
@Path("/loadData")
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
@Produces({"application/json","application/xml"})
public String caricaDatiBuilding(@FormParam("caricaDatiGson") String caricaDatiGson) {
    String message = "";
    .......
}
```

Le caratteristiche del metodo esposte attraverso il JavaBean sono:

```
String idApp;
String idSensore;
ArrayList<DatiSensoriBean> datiSensoriList;
```

Notare la presenza costante della proprietà *nome applicazione* all'interno del metodo. La proprietà `idSensore` indica da quale sensore stiamo ricevendo i dati, ed è un identificatore univoco. L'ultima proprietà è una lista che rappresenta una serie temporale di valori: *datatime* ora in cui il sensore rileva un cambiamento, *valore* rilevato.

Ogni metodo sviluppato restituisce un messaggio, una sorta di *feedback* che permette al richiedente del servizio di essere messo a conoscenze dell'esito dell'operazione. Il messaggio è *"Successful"* se l'operazione ha dato esito positivo, altrimenti *"Failure"* con l'aggiunta di ulteriori informazioni atte a far capire sia al richiedente che allo sviluppatore le possibili cause del fallimento.

Architettura software

Per lo sviluppo del software abbiamo utilizzato come IDE Eclipse e come linguaggio di implementazione JAVA, entrambi installati su un sistema operativo Windows. In Fig.4 è mostrata la struttura del codice, avente la tipica disposizione a package del mondo Java:

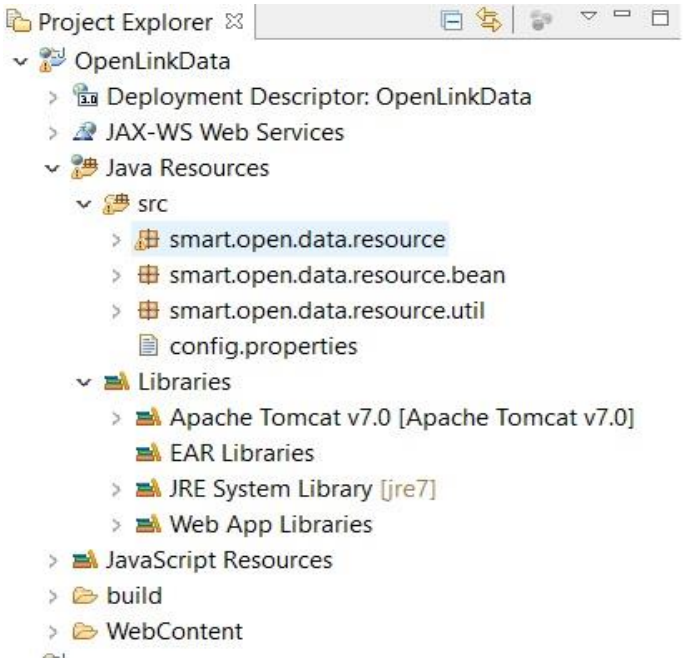

Fig.4 Architettura software

A livello più alto troviamo il package *smart.open.data.resource,* cioè il modulo principale del software dove sono stati implementati i servizi web.
Nel package *smart.open.data.resource.bean* troviamo i JavaBean utilizzati per deserializzare i messaggi Json, mentre nel package *smart.open.data.resource.util* ci sono le classi di utilità e di supporto ai package sopra citati.
Nella struttura del software appena descritta esiste anche un file che l'applicazione web deve prevedere obbligatoriamente: il *deployment descriptor*. Il deployment descriptor è un file XML denominato web.xml (Fig.5) che contiene tutte le definizioni necessarie al container per poter avviare ed eseguire correttamente l'applicazione.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>smart.campus</display-name>
  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
     <!-- Register resources and providers under progetto.ris.resource package. -->
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>smart.open.data.resource</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
 </web-app>
```

Fig.5 web.xml

Come ambiente di installazione ed esecuzione abbiamo utilizzato Apache Tomcat[13] che è uno dei web container più diffusi ed utilizzati; è sviluppato secondo il modello open ed è rilasciato in base alla Apache Software License.
Una volta installato il software è stato effettuato il *deployment* della nostra applicazione web, che è consistito nel creare una struttura di cartelle in base alle regole enunciate e nel copiare in ogni cartella i file opportuni. Effettuare a mano per ogni singolo file questa operazione sarebbe stato ovviamente molto dispendioso in termini di tempo e soprattutto molto soggetto ad errori. La soluzione a questo problema è stato il file WAR, Web Archive. Il WAR è un file che contiene tutti i file di una applicazione web già organizzati secondo la corretta struttura di deployment. In pratica è l'insieme dei file della nostra applicazione in una forma compressa, ottenuta creando un JAR di tutti i file e modificando l'estensione di questo jar a ".war". Il nome del file WAR su Tomcat diventa il nome dell'applicazione web stessa.

Testing

Tutto il software è stato testato con l'ausilio di un client appositamente sviluppato. Anche in questo caso abbiamo utilizzato il linguaggio di sviluppo JAVA, su una macchina avente come sistema operativo Windows e come libreria per l'invocazione dei servizi Web, la libreria Jersey. Segue un esempio di codice per testare la registrazione di una nuova applicazione:

```
public static void main(String[] args) {
       RegistraApplicazione registraApp = new RegistraApplicazione();
       String uriApp = registraApp.registraApp();
}

public String registraApp() {
       String nameApp = "SmartParking";
       Client client = Client.create();
       WebResource webResource = client.resource(URI+ nameApp);
       ClientResponse response = webResource.accept("application/json").
                                 get(ClientResponse.class);

       if (response.getStatus() != 200)
            throw new RuntimeException("Failed" + response.getStatus());

       output = response.getEntity(String.class);
       return output;
}
```

Fig.6 Client RegistraApplicazione

Conclusioni

Nell'ambito del progetto Smart Area abbiamo sperimentato da vicino la flessibilità dell'architettura RESTful, che a tutt'oggi rappresenta lo stato dell'arte per la realizzazione di web services, in termini di eterogeneità dei servizi richiesti per la gestione di dati provenienti dalle più disparate tipologie di sensori. Allo stato attuale lo sviluppo dei servizi tramite web services RESTful è tutt'altro che terminata e sempre più stanno maturando nuove esigenze di estrapolazione della conoscenza, al fine di rendere sempre più smart la gestione di risorse energetiche quali l'illuminazione, la condivisione di risorse pubbliche come parcheggi od il controllo del traffico e della mobilità.

Ringraziamenti

Questo lavoro è stato finanziato dal Dipartimento DIITET del CNR e fa parte del più ampio progetto "Energia da Fonti Rinnovabili e ICT per la Sostenibilità Energetica".

Riferimenti

[1] W3C Working Group. Web Services Architecture. http://www.w3.org/TR/ws-arch/.
[2] W3C Working Group. Simple object access protocol (soap) 1.1
http://www.w3.org/TR/2000/NOTE-SOAP20000508/#_Toc478383487
[3] W3C/MIT. Hypertext transfer protocol –http/1.1. http://tools.ietf.org/html/rfc2616.
[4] G. Meredith S. Weerawarana E. Christensen, F. Curbera. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl
[5] Leonard Richardson, Sam Ruby. RESTful web services O'Reilly, Sebastopol 2007
[6] Roy Thomas Fielding. Representational State Transfer (REST)
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.html
[7] Tim Berners-Lee. Universal Resource Identifiers –axioms of web architecture.
http://www.w3.org/DesignIssues/Axioms.
[8] W3CTim Berners-Lee. Uniform Resource Locators (URL)
http://www.w3.org/Addressing/URL/url-spec.txt
[9] http://cnr.isti/2015-TR-033
[10] http://virtuoso.openlinksw.com/
[11] https://jersey.java.net
[12] http://www.json.org
[13] https://httpd.apache.org/