

Approfondimento

Web Services

Esame di Programmazione per il Web

Fedele Ladisa

INDICE

Capitolo 1. Introduzione

- 1.1 Introduzione ai Web Services
- 1.2 Architettura dei Web Services
- 1.3 Stack protocollare di un Web Service
- 1.4 Servizio Discovery
- 1.5 Servizio Description
- 1.6 Servizio XML messaging
- 1.7 Servizio Transport

Capitolo 2. XML-RPC

- 1.1 Introduzione XML-RPC
- 1.2 XML-RPC data model
- 1.3 Struttura di una richiesta XML-RPC
- 1.4 Struttura di una risposta XML-RPC

Capitolo 1 . Introduzione

I Web Services, per la loro architettura e per il loro funzionamento rappresentano la soluzione a tutti i problemi di eterogeneità dell'informatica distribuita.

L'obiettivo dei Web Services è realizzare un ambiente distribuito nel quale tutti i componenti applicativi possano interagire senza curarsi delle piattaforme e dei linguaggi utilizzati dagli altri componenti.

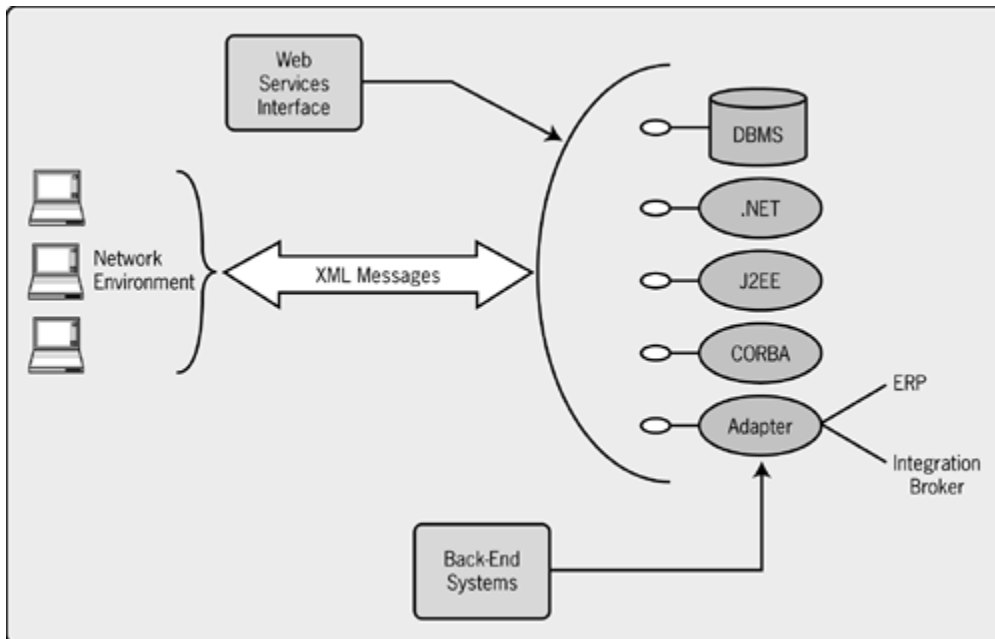
Si raggiunge così un livello di astrazione tale da poter garantire, senza alcun problema, l'interoperabilità tra gli elaboratori e dunque risolvere i problemi di eterogeneità.

Guardiamo ora più da vicino i Web Services.

1.1 Introduzione ai Web Services

Un Web Services è un componente software, disponibile su internet, che può essere richiamato utilizzando XML e non è dipendente da nessun Sistema Operativo o linguaggio di programmazione.

Secondo il W3C un Web Service è un sistema software progettato per supportare l'interoperabilità nell'interazione tra diverse macchine su una rete. Un Web Service è dotato di un'interfaccia descritta su una macchina con un adeguato formato (WSDL). Altri sistemi interagiscono con il Web Service, nel modo descritto nell'interfaccia del Web Services stesso, tramite messaggi SOAP solitamente inviati con il protocollo http.



In altri termini i Web Services rappresentano uno standard per interfacciarsi con il back-end di un sistema software.

Come si vede dalla figura in alto l'interfaccia di un Web Service riceve un messaggio in formato XML dalla rete e, a seguito di tale messaggio viene attivato un determinato servizio.

1.2 Architettura dei Web Services

Ci sono due modi di vedere l'architettura dei web services. Il primo è quello di esaminare il ruolo dei singoli componenti, il secondo è quello di esaminare lo stack protocollare di un web service.

L'architettura base dei Web Services include tecnologie che permettono di:

- Scambiare messaggi
- Descrivere i web services
- Pubblicare e scoprire le descrizioni dei web services

Sulla base di queste funzionalità si possono distinguere tre diversi e fondamentali ruoli: il fornitore di servizi (service provider), il richiedente di servizi (service requestor) e il registro di servizi (service registry or service discovery agency).

Service Oriented Architecture



Il fornitore di servizi(service provider) propone un servizio, definisce una descrizione di tale servizio e la pubblica nel registro(service discovery agency). Il richiedente(service requestor) recupera la descrizione del servizio, si collega al web service e invia una richiesta al service provider per eseguire un'operazione. Il service provider riceve la richiesta, la processa e invia una risposta. Le tecnologie tipicamente utilizzate da questo tipo di web services si basano su SOAP, WSDL e http.

1.3 Stack protocollare di un Web Service

Attualmente lo stack protocollare di un web service si compone di quattro differenti livelli:

- Servizio Transport
Questo livello si occupa di trasferire i messaggi tra le varie applicazioni. Attualmente questo livello include protocolli come hypertext transfer protocol (http), Simple Mail

Transfer Protocol (SMTP), file transfer protocol (FTP) e nuovi protocolli come Blocks Extensible Exchange Protocol (BEEP).

- XML messaging
Questo livello è responsabile della codifica di messaggi in formato XML. Attualmente tale livello include: XML-RPC e SOAP.
- Servizio Description
Questo livello è responsabile della descrizione dell'interfaccia pubblica di uno specifico web service. Attualmente questo servizio viene gestito attraverso il Web Service Description Language (WSDL).
- Servizio Discovery
Questo livello si occupa di centralizzare i vari servizi offerti da uno o più web services in un registro comune, e fornisce funzionalità di pubblicazione e ricerca dei vari servizi. Questo livello è gestito attraverso Universal Description, Discovery and Integration (UDDI).

Discovery	UDDI
Description	WSDL
XML messaging	XML-RPC, SOAP, XML
Transport	HTTP, SMTP, FTP, BEEP

Lo stack protocollare di base che è stato descritto può essere ampliato aggiungendo altre tecnologie riguardanti: sicurezza, management ecc

1.3 Servizio Discovery: UDDI

UDDI rappresenta il livello Discovery dello stack protocollare di un web service. UDDI offre la possibilità non solo di memorizzare le informazioni di un determinato servizio in uno specifico formato XML, ma anche di ricercare informazioni già presenti relativamente ad altri servizi.

I dati presenti in UDDI si possono suddividere in tre grandi categorie:

White Pages

Questa categoria include informazioni generali riguardo società: per esempio, nome, descrizione, indirizzo.

Yellow Pages

Questa categoria offre informazioni riguardo i prodotti o i servizi che le varie società offrono.

Green Pages

Quest'ultima categoria offre informazioni tecniche riguardo il web service, per esempio l'indirizzo al quale invocare il web service stesso.

1.4 Servizio Description: WSDL

WSDL attualmente rappresenta il livello Description dello stack protocollare di un web service.

WSDL è una grammatica XML usata per specificare l'interfaccia pubblica di un web service. Questa interfaccia contiene informazioni riguardo: le funzioni pubbliche disponibili, il formato dei dati, i protocolli da utilizzare per il trasporto dei messaggi e gli indirizzi per localizzare i diversi servizi.

Prendiamo in esame un servizio meteo ipotetico.

Soffermiamoci su due aspetti del seguente codice tralasciando le altre cose.

In primo luogo l'elemento *message* che specifica il formato dei singoli messaggi XML che vengono scambiati tra i computer. In questo caso abbiamo un *getWeatherRequest*, metodo che verrà utilizzato nelle richieste e un *getWeatherResponse* che verrà utilizzato per la risposta alla suddetta richiesta.

In secondo luogo l'elemento *service* che specifica che il servizio in questione è disponibile tramite SOAP all'indirizzo *http://localhost:8080/soap/servlet/rpcrouter*.

ServizioMeteo.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="WeatherService"
  targetNamespace="http://www.ecerami.com/wsdl/WeatherService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ecerami.com/wsdl/WeatherService.wsdl"
```

```

xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<message name="getWeatherRequest">
  <part name="zipcode" type="xsd:string"/>
</message>
<message name="getWeatherResponse">
  <part name="temperature" type="xsd:int"/>
</message>

<portType name="Weather_PortType">
  <operation name="getWeather">
    <input message="tns:getWeatherRequest"/>
    <output message="tns:getWeatherResponse"/>
  </operation>
</portType>

<binding name="Weather_Binding" type="tns:Weather_PortType">
  <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getWeather">
    <soap:operation soapAction=""/>
  <input>
    <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:examples:weatherservice"
use="encoded"/>
  </input>
  <output>
    <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
"
namespace="urn:examples:weatherservice"
use="encoded"/>
  </output>
</operation>
</binding>

<service name="Weather_Service">
  <documentation>WSDL File for Weather Service</documentation>
  <port binding="tns:Weather_Binding" name="Weather_Port">
    <soap:address
location="http://localhost:8080/soap/servlet/rpcrouter"/>
  </port>
</service>
</definitions>

```

Dunque utilizzando WSDL un client può localizzare un web service e invocare le funzioni pubbliche che questo mette a disposizione.

1.5 Servizio XML messaging

XML ha acquisito un grande successo grazie al fatto che permette la condivisione di dati tra diversi computer indipendentemente dal sistema operativo o dal linguaggio di

programmazione. Proprio per questa sua caratteristica XML si presta alla perfezione allo sviluppo di un web service.

Possiamo dunque individuare due diversi protocolli che lavorano a livello XML messaging dello stack protocollare di un web service, entrambi basati su XML: XML-RPC e SOAP.

XML-RPC è un semplice protocollo che utilizza messaggi XML per eseguire Remote Procedure Call (RPC). Le richieste sono codificate in XML e inviate tramite il metodo POST di http. Le risposte invece sono inglobate nel body del messaggio http di risposta. Dato che XML-RPC è indipendente dalla piattaforma permette a diverse applicazioni di comunicare tra loro; per esempio un client java può attivare RPC su un server PERL.

Facciamo ora un esempio di XML-RPC tenendo presente sempre l'ipotetico servizio meteo già visto in precedenza.

Il servizio in questione si aspetta come parametro un codice postale e restituisce la temperatura corrente dell'area relativa al codice inviato.

Vediamo dunque una richiesta XML-RPC

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
  <methodName>weather.getWeather</methodName>
  <params>
    <param><value>10016</value></param>
  </params>
</methodCall>
```

Come si può notare da questo esempio la richiesta consiste semplicemente in un elemento *methodName* che specifica il nome del metodo da richiamare e gli eventuali parametri, in questo caso il codice postale.

Vediamo ora la risposta XML-RPC

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
  <params>
    <param>
      <value><int>65</int></value>
    </param>
  </params>
</methodResponse>
```

La risposta è altrettanto semplice, consiste in un singolo elemento *methodResponse* che specifica il valore ritornato dalla funzione `weather.getWeather`.

1.6 Servizio Transport

Il livello di trasporto, come facilmente intuibile, si occupa di trasportare i messaggi XML da un computer ad un altro. I protocolli utilizzati in questo livello sono essenzialmente due: HTML e BEEP.

http è un protocollo semplice, stabile e molto diffuso, inoltre la maggior parte dei firewalls non bloccano il traffico http. Quest'ultima caratteristica permette a XML-RPC o a messaggi SOAP di non trovare alcun ostacolo lungo il loro percorso da un computer ad un altro. Sebbene questa sia una cosa positiva solleva una marea di problematiche in tema di sicurezza.

Una valida alternativa al protocollo http è il protocollo BEEP (Blocks Extensible Exchange Protocol). BEEP è un framework adatto per la creazione di nuovi protocolli di rete. Esso contiene funzioni per la gestione di errori, di autenticazione e sicurezza. Utilizzando BEEP si possono creare protocolli per ogni tipo di applicazione: instant messaging, file transfer ecc.

Capitolo 2 . XML – RPC

Come abbiamo già detto XML-RPC permette di effettuare chiamate a procedure o funzioni su una rete. XML-RPC utilizza essenzialmente il protocollo http per passare informazioni tra un client e un server.

Il client specifica il nome della procedura da attivare e i vari ed eventuali parametri da passare a tale procedura e il server processa la richiesta e risponde in caso negativo o positivo.

XML-RPC consiste essenzialmente in tre parti:

XML-RPC data model

Una serie di tipi da usare per passare parametri, restituire valori ed errori

XML-RPC request structure

Una richiesta http POST contenente informazioni su metodi e parametri

XML-RPC response structure

Una risposta http che contiene i valori di ritorno della funzione attivata o messaggi di errori

2.1 XML-RPC data model

In XML-RPC sono definiti sei diversi tipi di dati semplici e due tipi composti che rappresentano la combinazione di più tipi.

Ovviamente rispetto ad altri linguaggi di programmazione XML-RPC fornisce un numero molto ridotto di tipi di dati, nonostante questo è possibile rappresentare una gran quantità di informazioni.

Tutti i tipi semplici sono rappresentati da elementi XML il cui contenuto fornisce il valore. Per esempio per definire una stringa contenente il valore “Ciao Mondo”, dovremmo scrivere:

```
<string> Ciao Mondo </string>
```

Riporto qui di seguito una tabella che riassume i sei tipi base di dati.

Tipo	Valore	Esempio
Int o i4	Integer a 32 bit	<int>27</int> <i4>27</i4>
Double	Numero floating-point a 64 bit	<double>27.1234</double>
Boolean	True(1) o False(0)	<Boolean>1</Boolean>
String	Testo ASCII	<string>hello</string>
dateTime.iso8601	Data in format ISO8601	<dateTime.iso8601>20021125t02:20:04</dateTime.iso8601>
Base64	Informazioni binarie codificate in base 64	<base64>SGVsbG8sIFdvcmxkIQ==</base64>

Questi tipi di dati sono sempre inclusi all’interno dell’elemento *value* . Solo per quanto riguarda il tipo string, questo può essere incluso nell’elemento *value* omettendo l’elemento *<string>*.

Questi tipi, che sono tipi semplici, possono essere combinati all’interno di due tipi di dati complessi quali array e strutture.

Gli array rappresentano i dati in modo sequenziale, mentre le strutture sono coppie nome-valore.

Ma facciamo degli esempi.

Gli array sono indicati dall’elemento *array* e come gli altri tipi deve essere contenuto nell’elemento *value*.

Array contenente quattro stringhe

```
<value>
  <array>
    <data>
      <value><string>This </string></value>
      <value><string>is </string></value>
```

```
        <value><string>an </string></value>
        <value><string>array.</string></value>
    </data>
</array>
</value>
```

Array contenente quattro int

```
<value>
  <array>
    <data>
      <value><int>7</int></value>
      <value><int>1247</int></value>
      <value><int>-91</int></value>
      <value><int>42</int></value>
    </data>
  </array>
</value>
```

Array contenente diverse tipi

```
<value>
  <array>
    <data>
      <value><boolean>1</boolean></value>
      <value><string>Chaotic collection, eh?</string></value>
      <value><int>-91</int></value>
      <value><double>42.14159265</double></value>
    </data>
  </array>
</value>
```

In modo molto semplice è possibile creare array multidimensionali, basta definire un array all'interno di un altro array

```
<value>
  <array>
    <data>
      <value>
        <array>
          <data>
            <value><int>10</int></value>
            <value><int>20</int></value>
            <value><int>30</int></value>
          </data>
        </array>
      </value>
    </data>
  </array>
</value>
```

```

        <value>
            <array>
                <data>
                    <value><int>15</int></value>
                    <value><int>25</int></value>
                    <value><int>35</int></value>
                </data>
            </array>
        </value>
    </data>
</array>
</value>

```

Le strutture contengono un insieme di elementi *member*, a loro volta gli elementi *member* contengono un elemento *name* e un elemento *value*.

Una semplice struttura

```

<value>
    <struct>
        <member>
            <name>givenName</name>
            <value><string>Joseph</string></value>
        </member>

        <member>
            <name>familyName</name>
            <value><string>DiNardo</string></value>
        </member>

        <member>
            <name>age</name>
            <value><int>27</int></value>
        </member>
    </struct>
</value>

```

Una struttura può anche contenere un'altra struttura, o un array.

```

<value>
    <struct>
        <member>
            <name>name</name>
            <value><string>a</string></value>
        </member>

        <member>
            <name>attributes</name>

```

```
<value><struct>
  <member>
    <name>href</name>
    <value><string>http://example.com</string></value>
  </member>
  <member>
    <name>target</name>
    <value><string>_top</string></value>
  </member>
</struct></value>
</member>

<member>
  <name>contents</name>
  <value><array>
    <data>
      <value><string>This </string></value>
      <value><string>is </string></value>
      <value><string>an example.</string></value>
    </data>
  </array></value>
</member>
</struct>
</value>
```

2.2 Struttura di una richiesta XML-RPC

Una richiesta XML-RPC è una combinazione tra contenuto XML e header http. Nel senso che con XML specifichiamo la procedura che vogliamo attivare, ed eventualmente i vari parametri della stessa procedura, mentre con http inviamo il messaggio sul web.

L'elemento root di una richiesta XML-RPC è l'elemento *methodCall*.

Ogni elemento *methodCall* a sua volta contiene un elemento *methodName* e un elemento *params*.

L'elemento *methodName* contiene il nome della procedura che vogliamo chiamare, mentre l'elemento *params* contiene i parametri per quella procedura che dobbiamo chiamare. I parametri possono essere sia tipi di dati semplici: string, int, boolean che tipi composti: strutture e array.

Ma vediamo come sempre un semplice esempio.

Vogliamo chiamare un metodo (circleArea) che prende come parametro un valore double.

La richiesta XML-RPC sarà più o meno così:

```
<?xml version="1.0"?>
  <methodCall>
    <methodName>circleArea</methodName>
    <params>
      <param>
        <value><double>2.41</double></value>
      </param>
    </params>
  </methodCall>
```

Altro esempio, vogliamo chiamare la funzione sortArray passandogli come parametro due array:

```

<?xml version="1.0"?>
<methodCall>
  <methodName>sortArray</methodName>
  <params>
    <param>
      <value>
        <array>
          <data>
            <value><int>10</int></value>
            <value><int>20</int></value>
            <value><int>30</int></value>
          </data>
        </array>
      </value>
    </param>
    <param>
      <value>
        <array>
          <data>
            <value><string>A</string></value>
            <value><string>C</string></value>
            <value><string>B</string></value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodCall>

```

L'header http per queste richieste conterrà informazioni relative al contenuto, al mittente.

```

POST /target HTTP 1.0
User-Agent: Identifier
Host: host.making.request
Content-Type: text/xml
Content-Length: length of request in bytes

```

Per esempio l'header http per il metodo circleArea visto in precedenza sarà:

```

POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.124.2
Content-Type: text/xml
Content-Length: 169

```

La prima riga specifica che lo scambio dei dati deve avvenire utilizzando il metodo post secondo le specifiche http 1.0 accedendo all' URI specificato (/xmlrpc).

La seconda riga invece specifica lo user agent che viene utilizzato. Cioè il programma tramite il quale l'utente si sta collegando alla risorsa sul server.

La riga tre determina l'host al quale collegarsi.

La penultima riga specifica il contenuto della chiamata.

L'ultima riga invece la dimensione in byte della chiamata.

Ed ecco ora la richiesta completa.

```
POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.124.2
Content-Type: text/xml
Content-Length: 169

<?xml version="1.0"?>
  <methodCall>
    <methodName>circleArea</methodName>
    <params>
      <param>
        <value><double>2.41</double></value>
      </param>
    </params>
  </methodCall>
```

2.3 Struttura di una risposta XML-RPC

Le risposte XML-RPC sono molto simili alle richieste tranne che per qualche piccola differenza.

Tali differenze riguardano l'elemento *methodCall* che in una risposta viene sostituito con l'elemento *methodResponse* e non è presente l'elemento *methodName*.

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><double>18.24668429131</double></value>
    </param>
  </params>
</methodResponse>
```

Una risposta XML-RPC può contenere un solo parametro, che naturalmente potrebbe anche essere un array o una struttura, così sarebbe possibile ricevere più valori. Se il metodo chiamato non prevede un valore di ritorno il metodo restituisce un "success value" che corrisponde in altri termini ad un booleano settato a 1.

Se per caso venissero riscontrati dei problemi nel processare la richiesta XML-RPC l'elemento *methodResponse* conterrà un elemento *fault* al posto dell'elemento *params*. L'elemento *fault* dunque conterrà una breve descrizione del problema riscontrato.

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value><string>No such method!</string></value>
  </fault>
</methodResponse>
```

O anche:

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>code</name>
          <value><int>26</int>
        </member>
        <member>
          <name>message</name>
          <value><string>No such method!</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

L'header di una risposta XML-RPC è molto simile all'header di una richiesta che abbiamo analizzato in precedenza.

```
HTTP/1.1 200 OK
Date: Sat, 06 Oct 2001 23:20:04 GMT
Server: Apache/1.3.12 (Unix)
Connection: close
Content-Type: text/xml
Content-Length: 124
```

Tutte le risposte XML-RPC utilizzano il codice di risposta *200ok* anche se nella risposta è contenuto un elemento *fault*, quindi un errore nel processamento della richiesta.

L'elemento *Server* nell'header (riga 3) indica il tipo di web server che ha processato la richiesta inviata.

```
HTTP/1.1 200 OK
Date: Sat, 06 Oct 2001 23:20:04 GMT
Server: Apache/1.3.12 (Unix)
Connection: close
Content-Type: text/xml
Content-Length: 124
<?xml version="1.0"?>
  <methodResponse>
    <params>
      <param>
        <value><double>18.24668429131</double></value>
      </param>
    </params>
  </methodResponse>
```

Dopo che la risposta è inviata dall' XML-RPC server al XML-RPC client la connessione viene chiusa, come si vede anche dalla riga 4 dell'ultimo esempio.

Bibliografia

LIBRI

- [1] O'Reilly *"Web Services Essentials – distributed Applications with XML- RPC, SOAP, UDDI & WSDL"*.
- [2] Eric Newcomer *"Understanding Web Serevices – XML, WSDL, SOAP and UDDI"*

DOCUMENTAZIONE W3C

- [1] Michael Champion, Software AG,
Chris Ferris, IBM,
Eric Newcomer, Iona,
David Orchard, BEA System, *"Web Services Architecture"*

<http://www.w3.org/TR/2002/WD-ws-arch-20021114/>