SOAP e Web Services

1

SOAP: introduzione

- **Attualmente le applicazioni distribuite rappresentano una grossa parte della produzione software. Inoltre lo sviluppo di Internet e delle Intranet rende utile creare applicazioni che comunicano e si scambiano informazioni attraverso la rete**

- **Domanda: quale protocollo utilizzare?**

- **Attualmente esistono diversi standard per la codifica e la trasmissione delle chiamate e delle risposte**

2

CORBA, RMI, .NET, ...

- **CORBA, un noto framework per la gestione di oggetti distribuiti, utilizza l' internet Inter- ORB protocol (IIOP)**
- **DCOM, di Microsoft, utilizza l' Object Remote Procedure Call (ORPC)**
- **.NET Remoting, per la piattaforma .NET di Microsoft, può utilizzare diversi protocolli, compreso SOAP stesso**
- **Java, per la Java Remote Method Invocation (RMI) utilizza il Java Remote Method Invocation Protocol (JRMP)**
- **SOAP si propone come sostituto per tutti questi protocolli**

3

Che cos'è SOAP

- Invece di usare complicati bridge per tradurre un protocollo in un altro, quando due framework diversi devono comunicare tra loro, SOAP si propone come protocollo universale per la trasmissione dei dati di RPC
- SOAP, non si basa su tecnologie proprietarie e la sua applicazione è completamente libera

4

Che cos'è SOAP

SOAP è un protocollo leggero che permette di scambiare informazioni in ambiente distribuito:

- **SOAP è basato su XML**
- **SOAP gestisce informazione strutturata**
- **SOAP gestisce informazione tipata**
- **SOAP non definisce alcuna semantica per applicazioni o scambio messaggi, ma fornisce un mezzo per definirla**

5

SOAP è basato su XML

- **Tutti i protocolli citati in precedenza sono binari, mentre SOAP è basato su XML, quindi testuale**
- **Il debbugging è notevolmente semplificato perché XML è leggibile anche da esseri umani**
- **I dati sono molto più firwall-friendly: un firewall può analizzare e dedurre che sono innocui. Tra l'altro SOAP è stato pensato per usare HTTP come trasporto**
- **Il principale svantaggio di SOAP è costituito proprio dalla natura testuale, che lo rende molto meno performante rispetto alle sue controparti binarie (CORBA e .NET Remoting in particolare)**

6

SOAP non ha una semantica predefinita

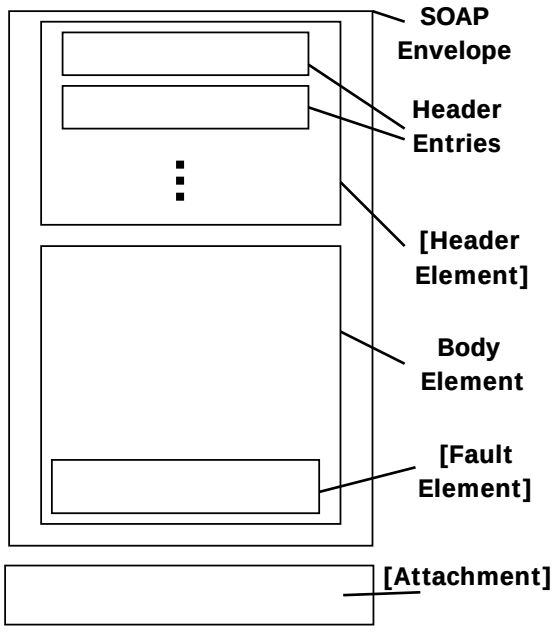
- La strutturazione dei messaggi SOAP, che deriva direttamente della strutturazione implicita di XML, è molto adatta al trasporto
- SOAP non definisce semantiche per i dati e le chiamate, ma fornisce agli sviluppatori i mezzi per farlo
- Con un intenso uso dei Namespace XML, SOAP permette agli autori dei messaggi di dichiararne la semantica usando grammatiche XML definite per lo scopo in particolari namespace

Struttura di un messaggio SOAP

- **Un messaggio SOAP è composto da:**
  - **Un elemento radice, *envelope*, obbligatorio. Il namespace di SOAP viene dichiarato all' interno di questo elemento**
  - **Un elemento *header* 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.**
  - **Un elemento *body* obbligatorio. Questo elemento contiene il messaggio vero e proprio**

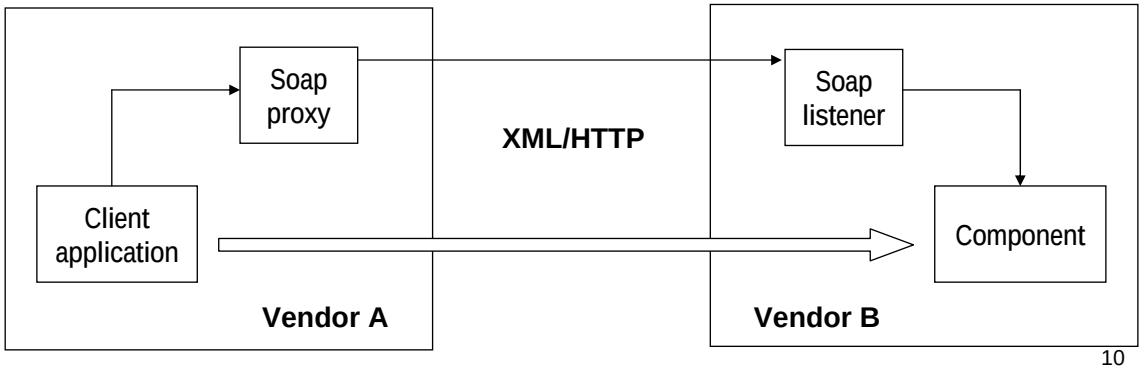
SOAP: struttura del messaggio

- Elementi esterni: **envelope** ed **attachments**
- Elementi interni: l'**header**
  - Info su: sicurezza, routing, formati, ecc…
- Elementi interni: il **body**
  - Contenuto vero e proprio del messaggio (richiesta o risposta)



Comunicazione SOAP

- **Una comunicazione SOAP include:**
  - **SOAP Request**
    - **Specifica il nome del metodo, i parametri del metodo, etc.**
  - **SOAP Response**
    - **Specifica il valore di ritorno o condizioni di errore**
- ***Tutti i messaggi SOAP sono codificati in XML***



10

Esempio di SOAP Request

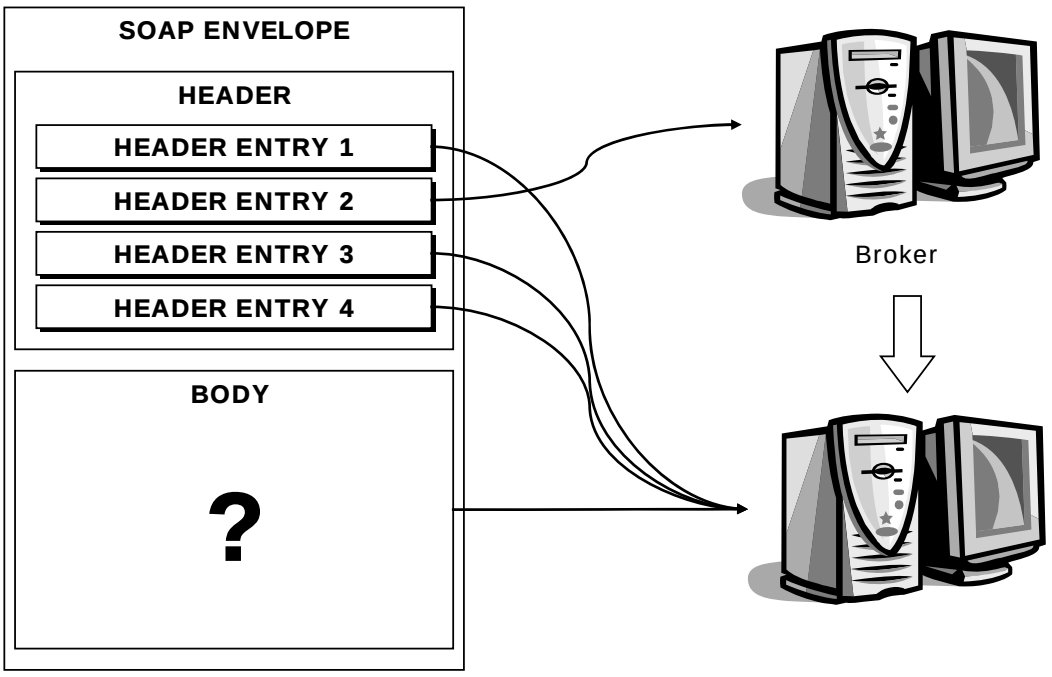
```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
     <ns1:getTemp xmlns:ns1="urn:xmethods-Temperature"
      SOAP-ENV:encodingStyle=
           "http://schemas.xmlsoap.org/soap/encoding/">
       <zipcode xsi:type="xsd:string">10016</zipcode>
     </ns1:getTemp>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

11

Esempio di SOAP Response

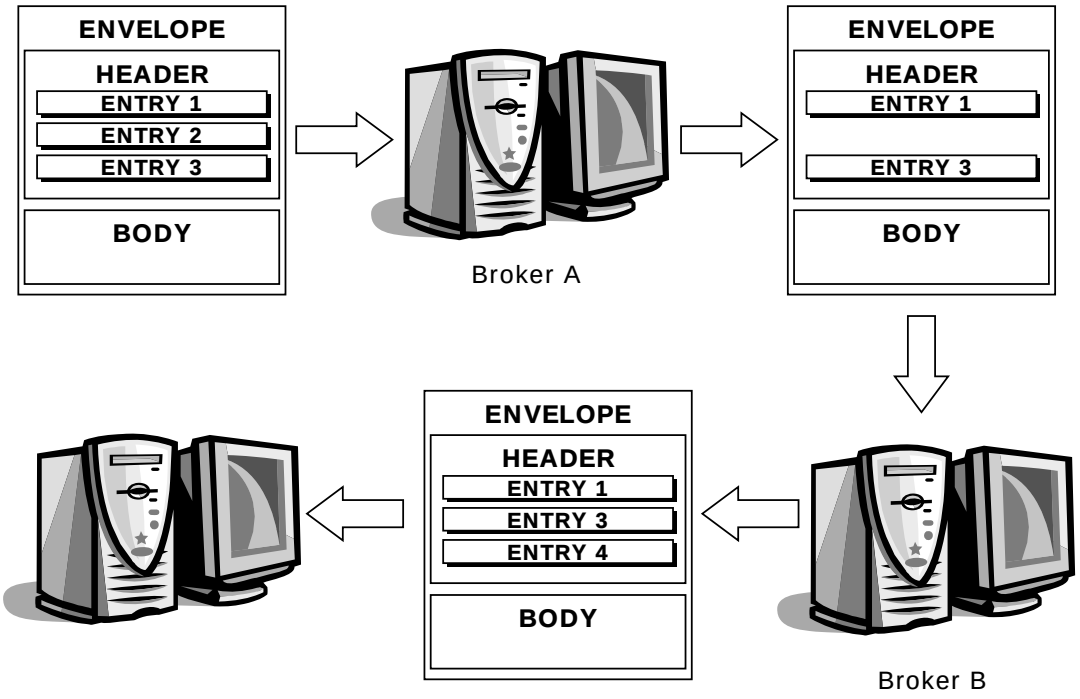
```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <SOAP-ENV:Body>
      <ns1:getTempResponse xmlns:ns1="urn:xmethods-Temperature"
       SOAP-ENV:encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/">
       <return xsi:type="xsd:float">71.0</return>
      </ns1:getTempResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Header


SOAP ENVELOPE
HEADER
HEADER ENTRY 1
HEADER ENTRY 2
HEADER ENTRY 3
HEADER ENTRY 4
BODY
?
Broker

13

Actors


ENVELOPE
HEADER
ENTRY 1
ENTRY 2
ENTRY 3
BODY
Broker A
ENVELOPE
HEADER
ENTRY 1
ENTRY 3
BODY
ENVELOPE
HEADER
ENTRY 1
ENTRY 3
ENTRY 4
BODY
Broker B

14

SOAP Encoding

- **Il concetto di encoding riguarda le informazioni applicative contenute nel tag Body: definisce come i dati vengono rappresentati in XML.**
- **Le specifiche SOAP lasciano una certa libertà, ma suggeriscono l'uso dell'encoding SOAP. Altri, come Apache, hanno storicamente proposto un econding più semplice, denominato encoding letterale.**
- **L'encoding SOAP definisce regole (derivate da una prima versione di XML Schema), per rappresentare il tipo di dato utilizzato nel blocco dati applicativo, mutuando le tipologie dai più diffusi linguaggi di programmazione e dai tipi di dati utilizzati nei database.**

15

SOAP Encoding: tipi semplici

Ad esempio, è possibile capire direttamente dal flusso XML se un determinato tag è di tipo stringa o numerico.
Per esempio, nel blocco XML:

```
<X1>12</X1>
```

il valore 12 potrebbe essere numerico, ma il tag potrebbe anche potenzialmente essere di tipo alfanumerico. Con l' encoding SOAP avremmo:

```
<X1 xsd:type="String">12</X1>
```

SOAP encoding: tipi composti

- ***Il tipo Array*: il SOAP encoding fornisce un elemento Array, che può essere usato per serializzare matrici e vettori. Le istanze di un elemento di tipo Array devono contenere un attributo di tipo *array Type* che specifica in tipo degli elementi dell' array. Ogni elemento è identificato dalla posizione**

- ***Le Strutture*: una struttura dati in SOAP è codificata tramite un elemento che contiene altri elementi nidificati. Ogni elemento è identificato da un nome.**

17

SOAP su HTTP

- **SOAP è stato esplicitamente pensato per usare HTTP come protocollo "di trasporto". Tuttavia, sono stati codificati anche metodi per inserire messaggi SOAP in altri tipi di protocolli internet, ad esempio l' SMTP.**

- **I framework SOAP come Apache SOAP (o Apache AXIS) rendono quasi del tutto trasparente questo approccio.**

SOAP su HTTP

- In HTTP esistono due metodi fondamentali per inviare una richiesta ad un server:
  - POST, invia al server il path della risorsa richiesta, seguita da un blocco di dati ("payload")
  - GET, invia al server il path della risorsa richiesta ed eventuali altre informazioni accodate al path

```
GET /index.html?campo=valore HTTP/1.0
```

- Per i messaggi SOAP si usa la modalità POST. Dove il blocco di dati, cioè il payload è costituito da messaggio SOAP vero e proprio

Web Services

Web Services: motivazioni

- **Uno dei problemi principali nell'industria è quello di integrare applicazioni informatiche sviluppate in maniera indipendente:**
  - **Altissimo numero di tecnologie eterogenee esistenti**
  - **Proliferare delle applicazioni distribuite**
- **L'integrazione applicativa può essere considerata a diversi livelli:**
  - **All'interno della stessa azienda**
  - **Tra partner dell'azienda**
  - **Verso utenti generici**

21

Web Services: integrazione

- L'integrazione è necessaria quando un processo coinvolge diversi sistemi informatici
- Sfruttare Internet come piattaforma globale di integrazione è un'opportunità notevole, soprattutto per l'integrazione tra diverse aziende
- L'integrazione è però resa più difficile dalle politiche di sicurezza
  – ad es. firewall aziendali, restrizioni d'accesso, etc.

Web Services: interoperabilità

- **Lo scopo primario di un servizio web è fornire una via estremamente semplice e versatile per far comunicare componenti software attraverso la rete.**
- **La vera chiave è l' interoperabilità:**
  - **I servizi web sono descritti astrattamente**
  - **Non sono dipendenti da architetture software/hardware particolari**
  - **Possono essere implementati praticamente con qualsiasi linguaggio**
  - **Il client e il server possono essere basati su linguaggi e tecnologie diverse**

Web Services: caratteristiche

- **Un Web Service è un'applicazione messa a disposizione (*pubblicata*) da una macchina ed accessibile attraverso protocolli standard su Internet (http e porta 80 per evitare i firewall)**

- **I Web Services (WS) presentano le seguenti caratteristiche**

  – *Interoperabilità*: un WS può essere invocato da client di tipo diverso, indipendenti dalla piattaforma tecnologica su cui il servizio è eseguito

  – *Incapsulamento*: gli utilizzatori di un WS sono ignari dei dettagli dell'implementazione

  – *Accessibilità*: un WS può essere reso pubblicamente disponibile per l'utilizzo

Web Services e XML

- I servizi web sono basati su XML:
  - Il protocollo SOAP è definito come linguaggio XML
  - I documenti di descrizione del servizio sono descritti in un linguaggio XML (WSDL)
  - Le strutture di classificazione e pubblicazione dei servizi (UDDI) sono definite in XML

Web Services: standard utilizzati

Gli standard utilizzati per i WS sono tutti dialetti di XML

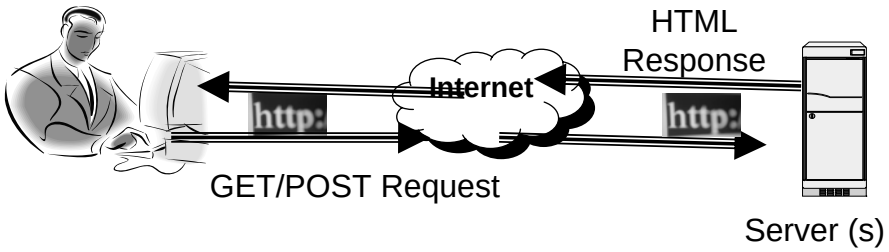
- **SOAP *(Simple Object Access Protocol):* descrive un protocollo basato su XML che definisce i meccanismi con cui un WS è invocato ed il formato dell'input e dell'output**
- **WSDL *(Web Service Definition Language)*: descrive l'interfaccia esterna di un WS affinché uno sviluppatore possa creare un client capace di accedervi**
- **UDDI *(Universal Discovery, Description and Integration)*: descrive registri contenenti informazioni per la scoperta e l'accesso ai WS**

26

Web e Web Services

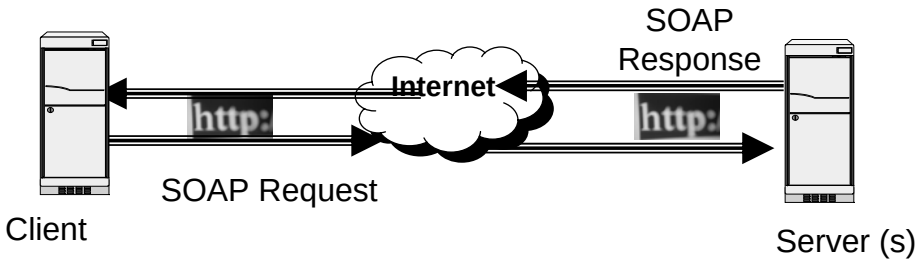
WEB:

un browser richiede una pagina Web tramite HTTP



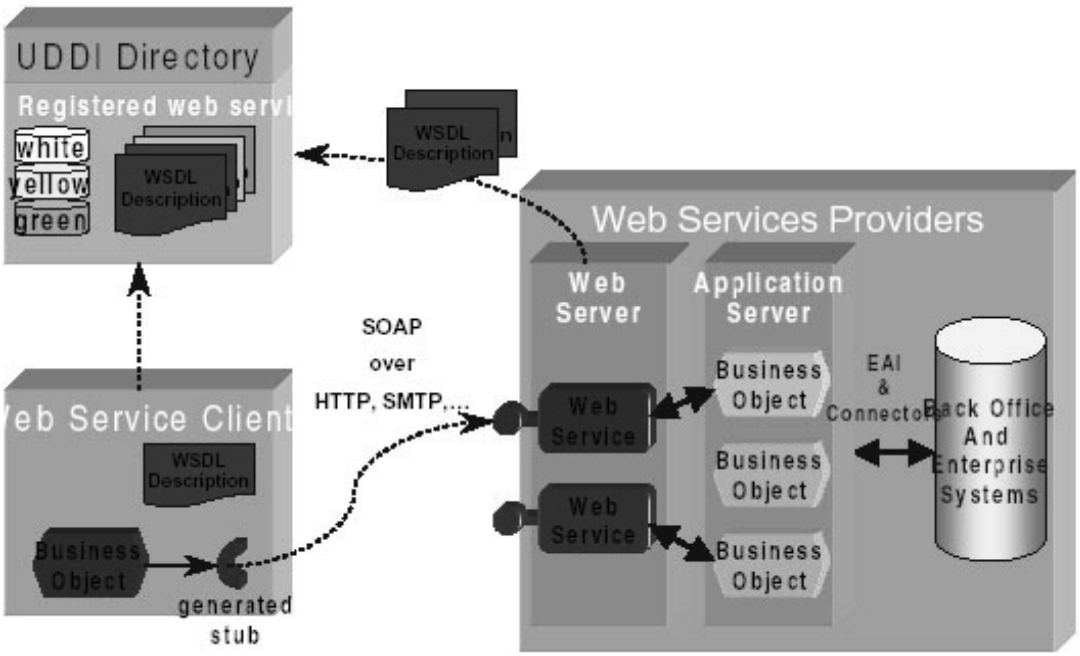
Web Service:

un client (es. un programma Java) invoca un Web Service tramite SOAP ed HTTP



27

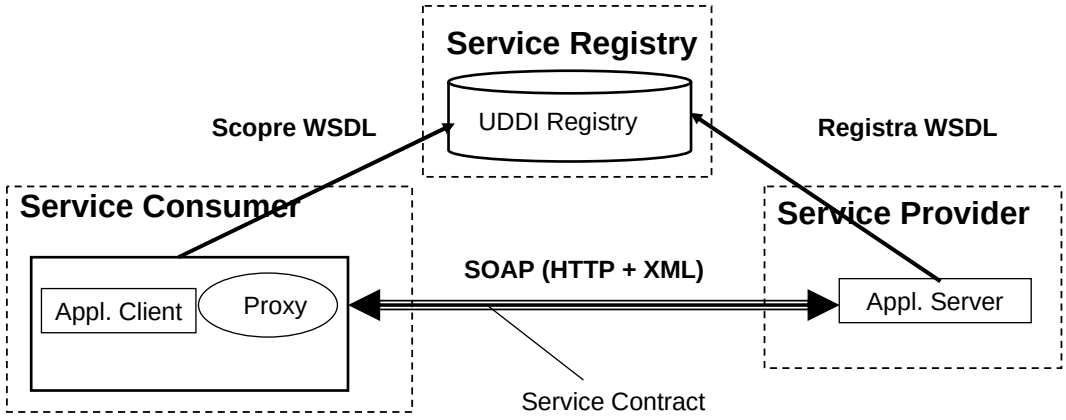
Come funzionano i Web Services



28

Come funzionano i Web Services

- Si realizza e si pubblica un WS (WSDL) in un registro UDDI:
- Il client ricerca il WS nel registro
- Il client costruisce dinamicamente il proxy
- Il client invoca il servizio e riceve la risposta
- Le API di ricerca e di pubblicazione di UDDI sono anch'esse Web Services!



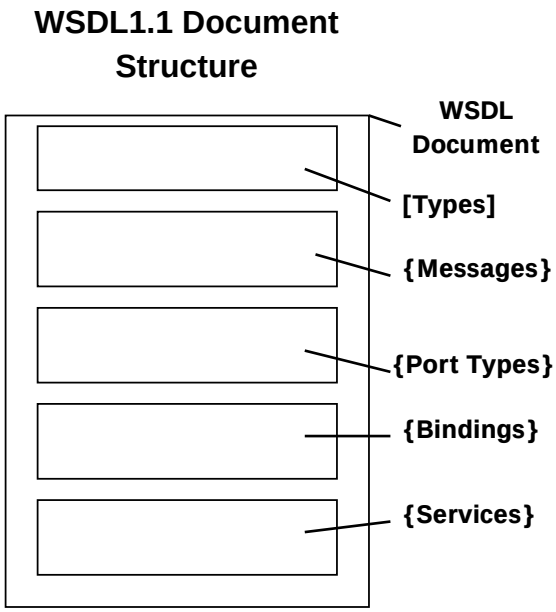
Web Service Description Language

- Standard W3C per la descrizione in XML dell'interfaccia dei Web Services
  - Contiene anche la locazione del servizio
- Un file WSDL è associato ad un Web Service ed è sufficiente ad un client per invocare il servizio

30

WSDL: struttura del documento

- Descrive:
  - *Cosa* un WS può fare
  - *Dove* risiede
  - *Come* invocarlo
- Documenti WSDL possono essere resi disponibili su registri UDDI



31

Componenti di un documento WSDL

Un documento WSDL è costituito essenzialmente da 5 elementi XML:

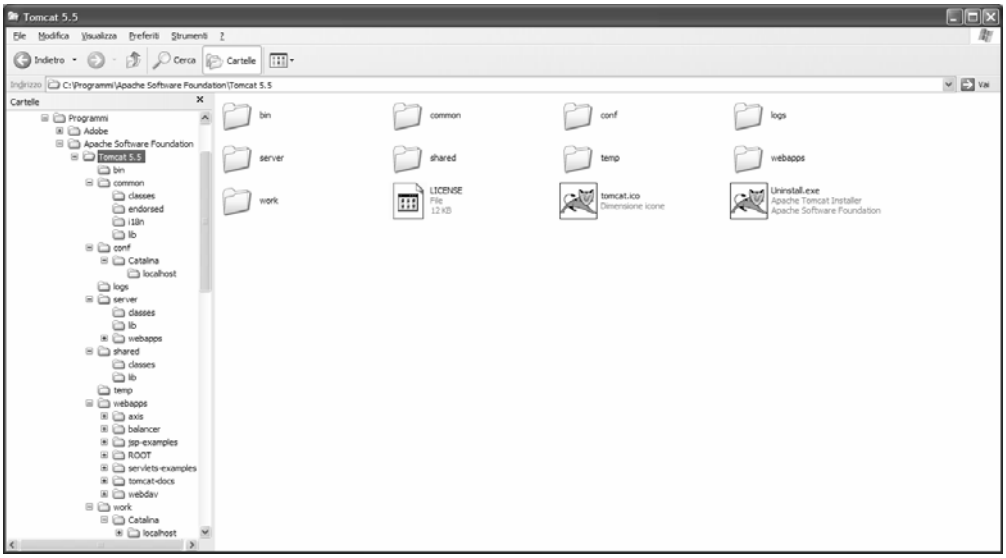
- **Types: i tipi di dati usati dal web service**
- **Messages: definiti come composizione o aggregazione dei tipi (elementari). La definizione astratta dei dati trasferiti**
- **portType: definizione di operazioni come messaggi di input e di output (è simile ad un'interfaccia)**
- **Bindings: fornisce dettagli implementativi per il tipo di porta, informazioni su come realizzare (implementare) la "porta" ed in particolare sul metodo di trasporto (soap, http,smtp,...)**
- **Services: dove le porte sono fisicamente realizzate (deployed). Combina tutti gli elementi precedenti.**

Web Services: realizzazione

- **Esistono diverse tecniche e tools per sviluppare Web Services, tra questi si distinguono Apache Tomacat e Apache AXIS**
- **Tomcat è uno dei Web Server più conosciuti ed utilizzati per applicazioni web**
- **Apache AXIS**
  - **È una Web Application**
  - **Implementa gli standard per i Web Services**
  - **Fornisce tools e librerie per lo sviluppo dei Web Services**
- ***http://jakarta.apache.org***

Installazione e Directory

Dopo l'installazione la struttura delle directory di Tomcat appare nel seguente modo.

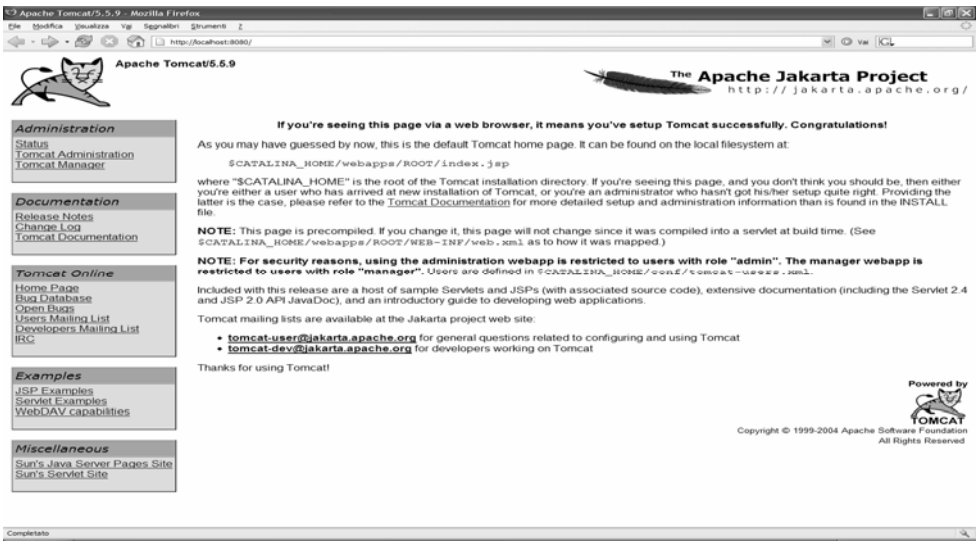


Struttura delle Directory

- ***bin***: **contiene gli script per l'avvio e l'arresto di Tomcat**
- ***common***: **le classi contenute nelle sue sotto-directory sono disponibili sia a Tomcat che a tutte le web-application in esecuzione sotto Tomcat**
- ***conf***: **contiene tutti i file di configurazione di Tomcat e delle web-application**
- ***logs***: **contiene i file di log delle applicazioni**
- ***server***: **le classi contenute nelle sue sotto-directory sono disponibili solo a Tomcat**
- ***shared***: **le classi contenute nelle sue sotto-directory sono condivise da tutte le web-application ma non sono disponibili per Tomcat**
- ***webapps***: **è la cartella predefinita da Tomcat in cui vengono inserite tutte le web-application (tutto ciò che viene copiato qui è automaticamente deployed)**
- ***work***: **contiene il codice delle servlet ottenuto dalla compilazione delle pagine JSP**

Installazione

Se l'installazione è andata ha buon fine e il Web Server Tomcat è stato avviato, potete connettervi ad esso tramite un browser. Se ci si trova sulla macchina locale basta digitare http://localhost:8080 e apparirà la seguente pagina.



Struttura di una Web Application

- Una Web Application è una collezione di risorse web (pagine JSP, HTML, Servlets, file di configurazione, ecc.)
- Questa collezione di risorse deve rispettare un certo standard, ossia deve essere organizzata in una determinata gerarchia di cartelle:

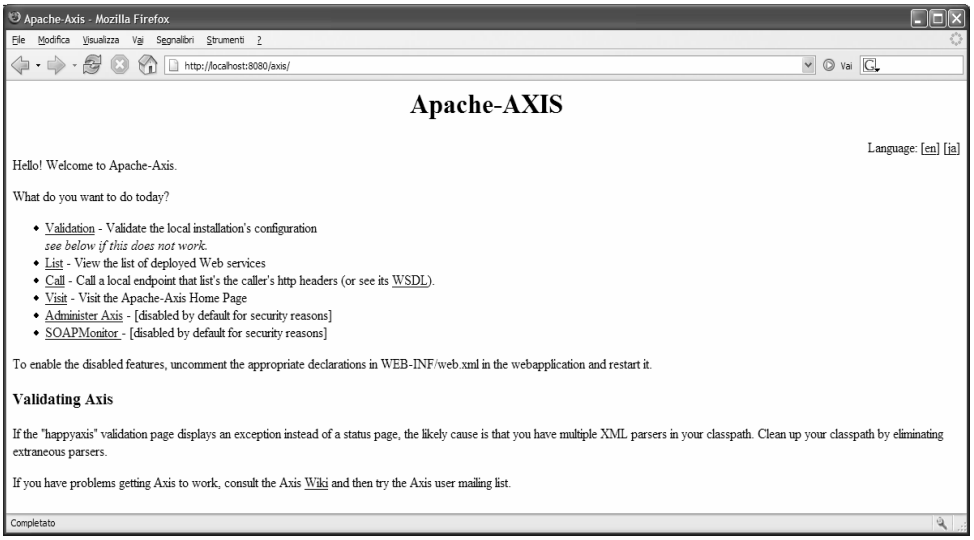
```
MyWebApplication/
    WEB-INF
        classes
        lib
```

- Per ogni applicazione deve essere creata una cartella contenente tutte le sue risorse (`MyWebApplication`)
- Le risorse devono necessariamente comprendere una cartella di nome `WEB-INF`, all'interno della quale sono contenute tutte le risorse private: non accessibili direttamente dall'utente web
- La cartella `WEB-INF` deve contenere un file `web.xml`, necessario all'inizializzazione dell'applicazione, una cartella `classes`, che conterrà i file `.class` dell'applicazione, e un'eventuale cartella `lib`, che conterrà le eventuali librerie esterne (ad es. file `.jar`)
- L'utente può accedere solo alla cartella principale dell'applicazione che, eventualmente, può essere strutturata in ulteriori sottocartelle, ad es. una cartella `images` che contiene tutte le immagini
- In fine la cartella `MyWebApplication` può essere posizionata all'interno della cartella `webapps` di Tomcat, in modo da avere un deploy automatico

37

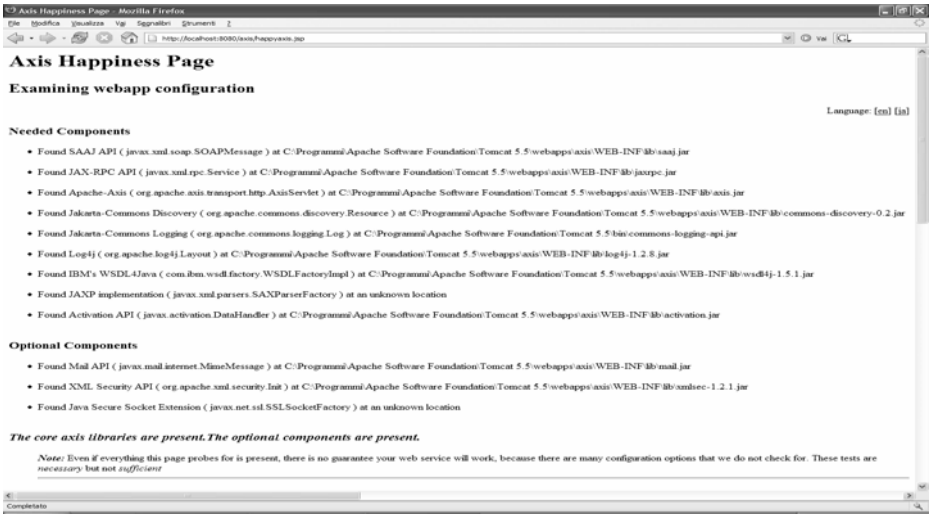
Installazione

Verificare il corretto funzionamento di Axis digitando il suo URL. L'home page che verrà visualizzata è la seguente



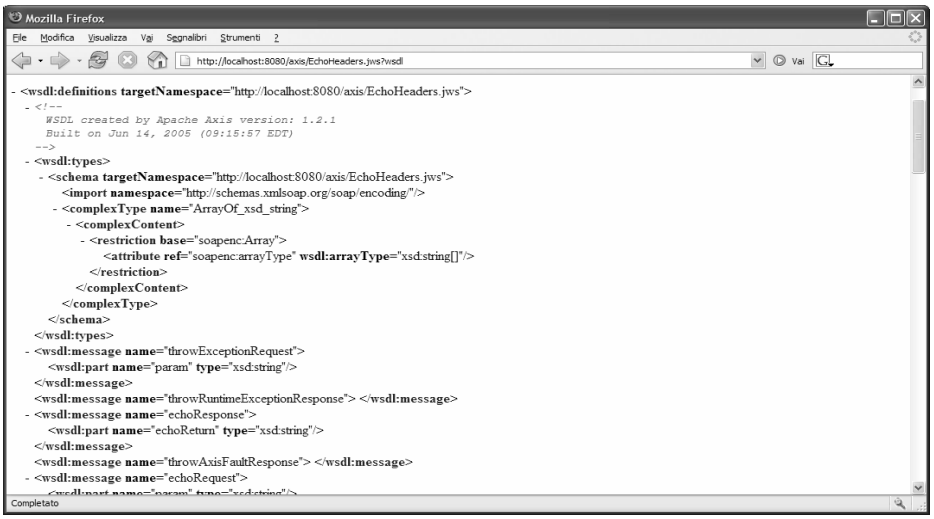
Validazione

Verificare che vengano caricate correttamente tutte le librerie cliccando sul link Validation presente nella home page di Axis. Verrà mostrata una pagina simile a quella seguente, nella quale non dovranno esserci messaggi di errori o di warning



Verifica WSDL

Cliccare infine sulla voce WSDL, nella home page di Axis, per verificare che venga generato correttamente un file WSDL. Il file è il seguente ed è relativo ad un sevizio già esistente



Lista dei servizi attivi

Se volete conoscere i servizi attualmente pubblicati sul vostro Web Server cliccate sul link List nell'home page di Axis

41

Un primo esempio

Ecco un primo semplice esempio di servizio. Si tratta di una classe java che contiene un solo metodo. Il metodo è ciò che rappresenta il nostro servizio, mentre la classe è il contenitore del servizio

```
//File Esempio1.java

/* Classe che implementa il servizio. */
public class Esempio1{

        /* Metodo che implementa il servizio.
        *  Questo servizio restituisce una stringa di saluto.
        **/
        public String saluto(){ return "Ciao Mondo"; }

}
```


42

Pubblicazione del servizio

- Compilazione
- Copia del file compilato nella cartella **`classes`**
- Creazione del file WSDD per il deploy del servizio
- Lanciare il comando **`AdminClient`**

Il file WSDD

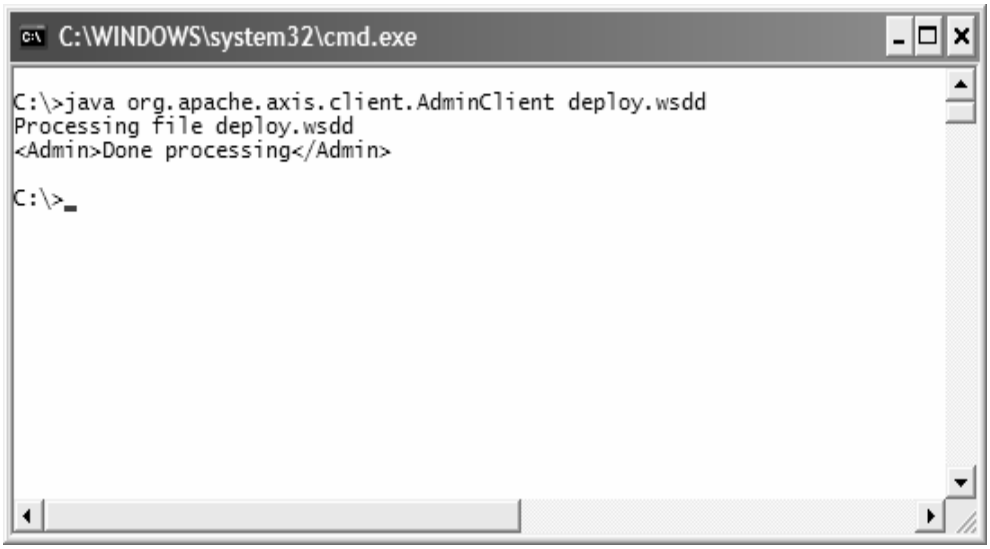
Nel file WSDD vanno inserite le direttive per la pubblicazione del servizio. Primo tra tutti bisogna creare un tag <service> per ogni servizio che si vuole pubblicare, poi bisogna specificare: nome del servizio, nome della classe e nome dei metodi da pubblicare. Per il nostro esempio il file WSDD è il seguente

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
        <service name="Esempio1" provider="java:RPC">
                <parameter name="className" value="Esempio1"/>
                <parameter name="allowedMethods" value="saluto"/>
                <parameter name="scope" value="Request"/>
        </service>
</deployment>
```

44

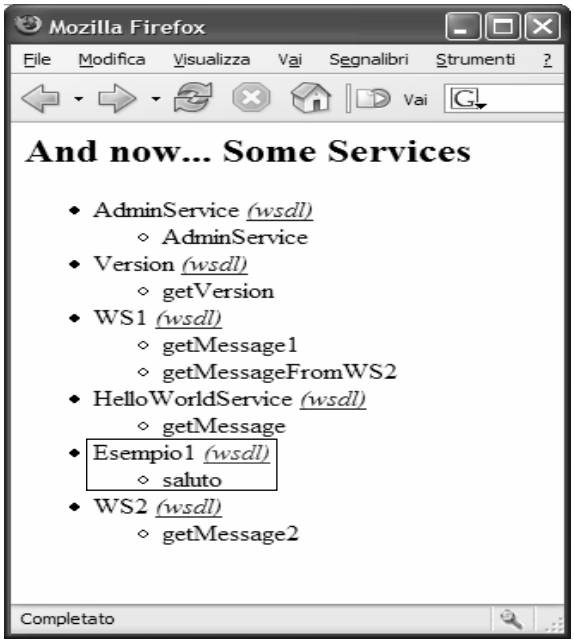
AdminClient

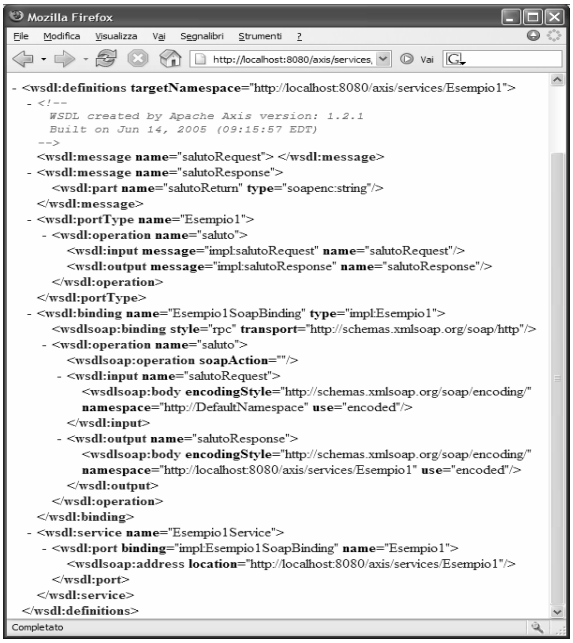
Di seguito è mostrata la sintassi da usare per pubblicare il servizio da riga di comando. Da notare che si è assunto che le variabili di sistema, PATH e CLASSPATH, siano settate correttamente.



45

Nuova lista dei Servizi e WSDL del nostro esempio





Pubblicazione del servizio: metodo 2 (servizi JWS)

- **Sono dei comuni Web Services il cui deployment avviene in maniera automatica**
- **Si ottengono ridenominando un file con estensione ".java" in un file con estensione ".jws"**
- **Il nuovo file, così ottenuto, va copiato nella home directory di Axis**
- **Il servizio sarà subito accessibile dal browser semplicemente digitando il suo URL, ad esempio: http://localhost:8080/axis/Esempio5.jws**
- **Il file WSDL che lo descrive sarà invece disponibile aggiungendo "?wsdl" al su indirizzo URL o cliccando sull'apposito link nella pagina che viene mostrata**

47

Servizi JWS

- **Come si è potuto notare ciò che si copia è il sorgente del servizio**
- **La sua compilazione avviene in automatico quando il servizio viene invocato per la prima volta e il risultato della compilazione va a finire nella sottocartella jwsClasses**
- **Al contrario dei servizi pubblicati tramite un file WSDD, per i servizi JWS non esiste in Axis un link che mostra i servizi attivi. Quindi bisogna conoscerne l'esistenza**

Servizi JWS

Ecco un esempio già visto, modificato per essere autodeployato

```
//File Esempio5.jws

/* Classe che implementa il servizio. */
public class Esempio5{

    /* Metodo che implementa il servizio.
    *  Questo servizio restituisce la stringa data in ingresso.
    **/
    public String echo2(String msg){ return msg; }

}
```


49

Lato client: Generated Stub (WSDL2Java)

- Tool java che consente la generazione delle classi necessarie alla creazione di un client
- Non è più necessario leggere il contenuto del file WSDL
- Si fa uso di una interfaccia locale del servizio remoto
  - L'interfaccia ha, generalmente, lo stesso nome del servizio remoto
  - La creazione di un'istanza del servizio viene fornita tramite la classe "_ServiceLocator"
- Il tool riceve in ingresso l'URL del file WSDL che descrive il servizio

Google WS

Vediamo ora un esempio di un servizio fornito da un provider remoto. Per prima cosa dobbiamo consultare il WSDL che descrive tale servizio

```
    note, ie and oe are ignored by server; all traffic is UTF-8.
  -->
- <message name="doGoogleSearch">
    <part name="key" type="xsd:string"/>
    <part name="q" type="xsd:string"/>
    <part name="start" type="xsd:int"/>
    <part name="maxResults" type="xsd:int"/>
    <part name="filter" type="xsd:boolean"/>
    <part name="restrict" type="xsd:string"/>
    <part name="safeSearch" type="xsd:boolean"/>
    <part name="lr" type="xsd:string"/>
    <part name="ie" type="xsd:string"/>
    <part name="oe" type="xsd:string"/>
  </message>
- <message name="doGoogleSearchResponse">
    <part name="return" type="typens:GoogleSearchResult"/>
  </message>
  <!-- Port for Google Web APIs, "GoogleSearch" -->
- <portType name="GoogleSearchPort">
  - <operation name="doGetCachedPage">
      <input message="typens:doGetCachedPage"/>
      <output message="typens:doGetCachedPageResponse"/>
    </operation>
  - <operation name="doSpellingSuggestion">
      <input message="typens:doSpellingSuggestion"/>
      <output message="typens:doSpellingSuggestionResponse"/>
    </operation>
  - <operation name="doGoogleSearch">
      <input message="typens:doGoogleSearch"/>
      <output message="typens:doGoogleSearchResponse"/>
    </operation>
  </portType>
- <!--
    Binding for Google Web APIs - RPC, SOAP over HTTP
  -->
- <binding name="GoogleSearchBinding" type="typens:GoogleSearchPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  - <operation name="doGetCachedPage">
      <soap:operation soapAction="urn:GoogleSearchAction"/>
    - <input>
        <soap:body use="encoded" namespace="urn:GoogleSearch" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
```

Google WS

- URL del WSDL:
  `http://api.google.com/GoogleSearch.wsdl`
- Comando:

```
java org.apache.axis.wsdl.WSDL2Java
http://api.google.com/GoogleSearch.wsdl
```

- Risultato:
  - DirectoryCategory.java
  - GoogleSearchBindingStub.java
  - GoogleSearchPort.java
  - GoogleSearchResult.java
  - GoogleSearchService.java
  - GoogleSearchServiceLocator.java
  - ResultElement.java

52

GoogleSearchPort

```
/**
 * GoogleSearchPort.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis 1.2.1 Jun 14, 2005 (09:15:57 EDT) WSDL2Java emitter.
 */

package GoogleSearch;

public interface GoogleSearchPort extends java.rmi.Remote {

    public byte[] doGetCachedPage(java.lang.String key,
                  java.lang.String url) throws java.rmi.RemoteException;

    public java.lang.String doSpellingSuggestion(java.lang.String key,
                  java.lang.String phrase) throws java.rmi.RemoteException;

    public GoogleSearch.GoogleSearchResult doGoogleSearch(java.lang.String
         key, java.lang.String q, int start, int maxResults, boolean filter,
         java.lang.String restrict, boolean safeSearch, java.lang.String lr,
         java.lang.String ie, java.lang.String oe) throws
                  java.rmi.RemoteException;
}
```


53

Google WS

Dopo che il tool ha generato le classi stub, possiamo costruire il nostro client. In particolare il servizio usato nell'esempio effettua la correzione ortografica di una stringa. L'uso di questo servizio, e di quelli presenti nel file WSDL visto prima, richiede la registrazione sul sito Google e necessita di una chiave da inserire come primo parametro di ogni servizio invocato, inoltre, il provider limita l'utilizzo del servizio a 1000 volte al giorno.

```
package GoogleSearch;

import java.rmi.RemoteException;

import javax.xml.rpc.ServiceException;

public class ClientGoogle{
    public static void main(String args[]){
        GoogleSearchServiceLocator service = new GoogleSearchServiceLocator();
        GoogleSearchPort gsp = null;
        try {
            gsp = service.getGoogleSearchPort();
            String msg = (String)gsp.doSpellingSuggestion("IUFgUcBQFHKKSg3bnQBdUNehsVolOEj2", "Mi name iss Mario");
            System.out.println(msg);
        }
        catch (RemoteException e) {e.printStackTrace();}
        catch (ServiceException e) {e.printStackTrace();}
    }
}
```

Un esempio di client dinamico

Il client si realizza semplicemente facendo uso dell'interfaccia Call e della classe Service, che fornisce un'istanza di Call. Inoltre è necessario conoscere, tramite la lettura del WSDL, l'URL del servizio, il suo nome, e i parametri di ingresso e uscita.

```
import java.net.*;
import java.rmi.*;
import javax.xml.namespace.*;
import javax.xml.rpc.*;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;

public class ClientEsempio1{
.
.
.
```

55

Un esempio di client dinamico

```
public static void main(String[] args){
   String messaggio = "";
   try{
        Call call = (Call)new Service().createCall();
        call.setTargetEndpointAddress(
                         new URL("http://localhost:8080/axis/services/"));

        call.setOperationName(new QName("Esempio1", "saluto"));
        Object rispostaWS = call.invoke(new Object[]{});
        messaggio = "il Web service ha risposto:" + (String)rispostaWS;
   }
   catch(MalformedURLException ex){messaggio = "errore: l'url non è
   esatta";}
   catch(ServiceException ex){messaggio = "errore: la creazione della
   chiamata è fallita";}
   catch(RemoteException ex){messaggio = "errore: l'invocazione del WS è
   fallita";}
   finally{
        System.out.println(messaggio);
   }
     }
}
```

Dynamic Invocation

- Call e Service sono i metadati che descrivono il servizio da invocare
- La classe Call genera automaticamente le richieste SOAP per il Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<SOAP-ENV:Body>
<ns1:Echo2 xmlns:ns1="http://localhost:8080/axis/Echo.jws">
<arg0 xsi:type="xsd:string">Hello!</arg0>
</ns1:echoService>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Chiamata del servizio