Web Services

Ing. Enrico Oliva
Phd student, DEIS
eoliva@deis.unibo.it

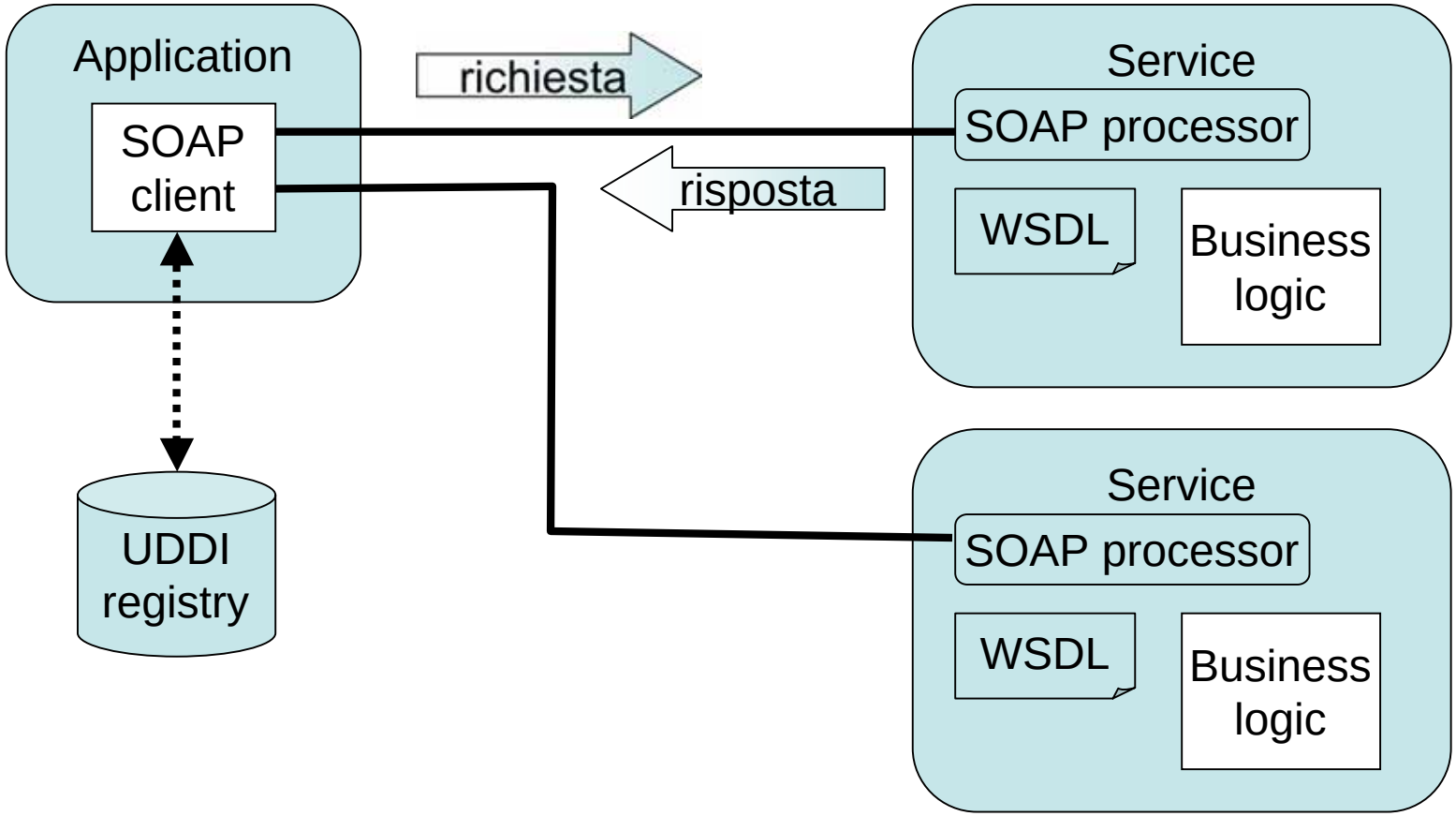
Web Services

- I servizi Web permettono alle applicazioni di invocare operazioni direttamente sulla rete da diversi sistemi fornendo un più ricco servizio ai clienti
- Web services forniscono un livello di astrazione dai sistemi software esistenti come: CORBA, .NET, J2EE
- L'interfaccia Web Service riceve un messaggio XML dal client e lo trasforma in un formato comprensibile a un particolare sistema software sottostante
- Alcune caratteristiche dei Web Services:
  - Sono basati sulla rappresentazione dei dati XML
  - L'utilizzatore non è legato al Web Service direttamente
  - Possono essere sincroni o asincroni
  - Supportare Remote Procedure Call (RPC)
  - Supportare lo scambio di documenti

Ingredienti principali dei Web Services

- Descrizione dell'informazione spedita sulla rete
  - XML è un linguaggio di Markup utilizzato dai WS per descrivere le informazioni
    - L'invocazione remota di una operazione comunemente avviene passando un parametro e ricevendo un risultato.
- Definizione delle capacità dei Web Services
  - WSDL Web Services Describe Language permette di definire l'interfaccia delle operazioni offerte dai Web Services
    - È un linguaggio definito usando XML
- Accesso ai Web Services
  - SOAP Simple Object Access Protocol fornisce un modo per specificare le invocazioni e i dati in XML
    - Il client una volta nota l'interfaccia d'uso del WS deve utilizzare un protocollo per invocare le operazioni
- Ricerca dei Web Services
  - UDDI Universal Description Discovery Integration fornisce ai WS un modo standard per pubblicizzare le loro interfacce

Interazione con Web Services

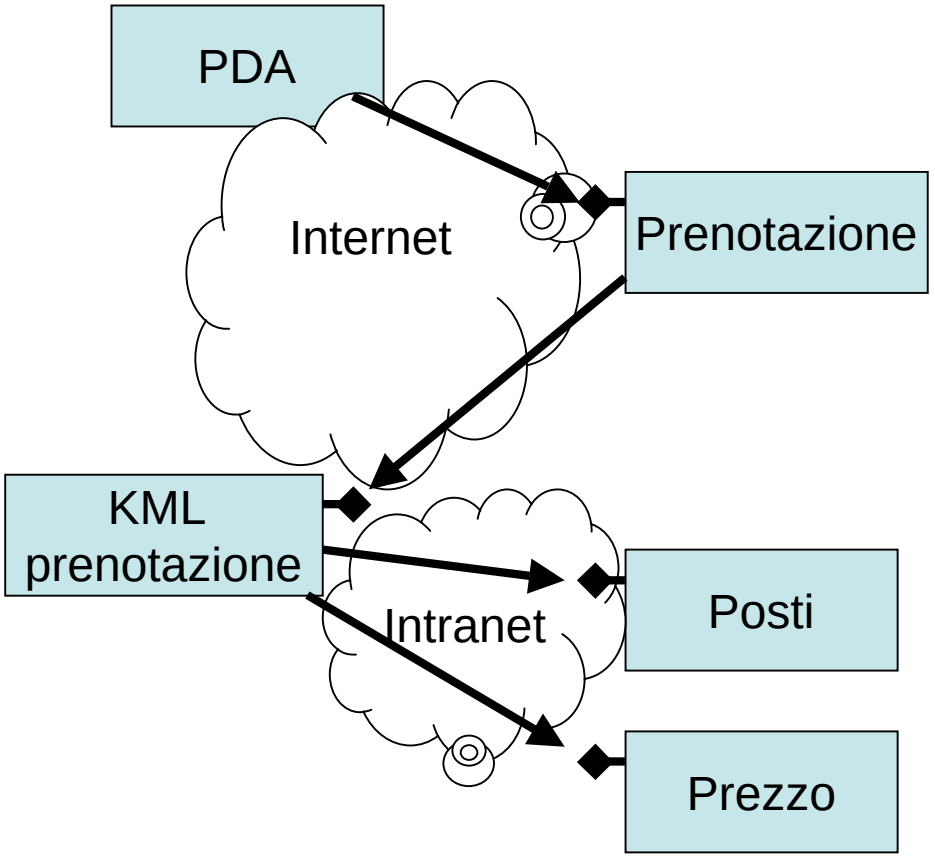


Applicazione dei Web Services

- Accesso alle applicazioni attraverso internet
  - Alcune applicazioni Web come prenotazioni, pagamenti possono funzionare anche come Web Services
- Business to Business (B2B)
  - Integrazione B2B attraverso il collegamento di software in esecuzione su varie organizzazioni
- Application to Application (A2A)
  - Integrazione tra applicazioni diverse anche su piattaforme diverse all'interno di una stessa azienda

Esempio: Prenotazione Volo

1. Contatto con una applicazione di prenotazione di voli
   - Mostra un client che accede direttamente al servizio attraverso la rete internet
2. L'applicazione contatta uno specifico sistema di prenotazione di una particolare compagnia
   - Mostra integrazione B2B basata sulla rete internet
3. Il sistema di prenotazione del volo interagisce attraverso la propria intranet con le applicazioni di determinazione del prezzo e di verifica della disponibilità dei posti
   - Mostra uno scenario di integrazione A2A su una rete intranet

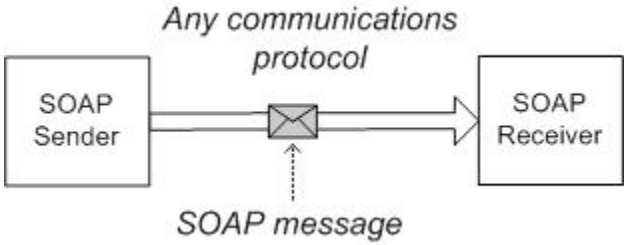


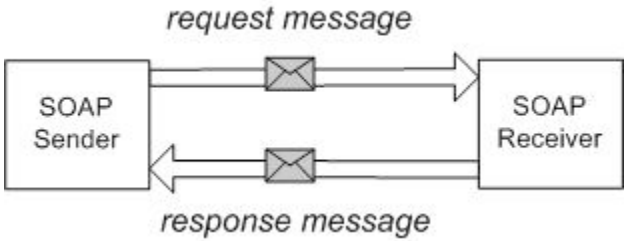
Descrizione informazioni: XML

- XML emerge come standard per la descrizione dell'informazione scambiata tra sistemi eterogenei
- Un documento XML è composto da elementi demarcati da tags
  - Un tags è una parola tra apici <nome>
    - Esempio <nome> Enrico </nome>
    - Qualsiasi informazione può essere espressa tra tags
  - Un elemento può contenere degli attributi
    - Esempio <nome tipo="M"> Enrico </nome>
- Il come gli elementi vengano definiti è specificato da uno schema XML definito attraverso il linguaggio XML Schema Definition (XSD)
  - Uno schema definisce uno o più elementi e le regole di come questi elementi possano essere usati
    - Es. all'interno di <Envelope> ci possono essere più <body>
  - Fornisce una specifica per la tipizzazione dei dati
- Un namespace permette di associare ad un tag il suo schema di riferimento specificando cosi uno scope per i nomi degli elementi
  - Un documento XML è solitamente composto da più schemi
  - Ad ogni namespace è associato un URI
    - Es. <s:account xmlns:s=http://www.qwickbank.com/bank> </account>

SOAP

- SOAP è un protocollo per lo scambio di informazioni strutturate in un ambiente decentralizzato e distribuito
- SOAP usa XML per la definizione dei messaggi
- SOAP può essere usato sopra ogni livello di trasporto come TCP, http, SMTP
- SOAP non è legato al modello RPC (remote procedure call)





Interazione

Stili di interazione supportati da SOAP

- Remote procedure call (RPC)
  - significa che il corpo del messaggio è una rappresentazione ad una chiamata ad un metodo
- Document oriented (Electronic Data Interchange EDI vs elettronic business XML ebXML)
  - significa che il corpo del messaggio è un documento XML che può essere formattato anche seguendo delle specifiche come EDI o ebXML

Messaggio SOAP

- **Envelope** incapsula il contenuto del messaggio
- **Header** nello specifica non ci sono regole sul contenuto
  - è un blocco di intestazione destinato a informazioni contestuali per l'elaborazione di un messaggio come la sicurezza, ID di transazione …
- **Body** incapsula le richieste e le risposte
- **Fault** eventuali messaggi di errore

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>  <!-- optional -->
<!-- header blocks go here... -->
  </soap:Header>
  <soap:Body>
<!-- payload or Fault element goes here... -->
  </soap:Body>
</soap:Envelope>
```

Le specifiche del messaggio SOAP 1.2 sono contenute nel XML Schema Definition in:
http://schemas.xmlsoap.org/soap/envelope/

Envelope

- <SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schema/xmlSoap"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
- Il primo attributo è una dichiarazione di namespace… che serve per evitare conflitti nella composizione di documenti ed è analogo all'uso dei package in Java
- Il namespace xsi permette l'uso dell'attributo xsi:type per specificare il tipo di dato all'elemento
- Il namespace xsd viene usato per definire i valori dell'attributo xsi:type ad esempio xsd:string, xsd:boolean, xsd:float…
  - xsi:type="xsd:boolean"
- Se nel documento non si fa uso di xsi e xsd si presuppone che tutti gli elementi siano di tipo stringa
  - Il tipo di default del messaggio SOAP è il tipo stringa

xsd e Java

- Mapping dei tipi xsd in Java

xsd:base64Binary	`byte[]`
xsd:boolean	`boolean`
xsd:byte	`byte`
xsd:dateTime	`java.util.Calendar`
xsd:decimal	`java.math.BigDecimal`
xsd:double	`double`
xsd:float	`float`
xsd:hexBinary	`byte[]`
xsd:int	`int`
xsd:integer	`java.math.BigInteger`
xsd:long	`long`
xsd:QName	`javax.xml.namespace.QName`
xsd:short	`short`
xsd:string	`java.lang.String`

Header

- In SOAP 1.1 e SOAP 1.2 non ci sono convenzioni sulla struttura del header
- Esistono protocolli costruiti sopra SOAP come ebXML che ne hanno formalizzato l'uso con tag come <From>,<To>…
- Usando SOAP per chiamate RPC si usa l'header per contenere informazioni riguardanti l'infrastruttura
  - Identificativo di transazione
  - Autenticazione
  - Sessione

Esempio messaggio SOAP

Richiesta

```
<soap:Envelope xmlns:soap=
      "http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
      <x:TransferFunds xmlns:x=
       "urn:examples-org:banking">
          <from>22-342439</from>
          <to>98-283843</to>
          <amount>100.00</amount>
      </x:TransferFunds>
  </soap:Body>
</soap:Envelope>
```

Risposta

```
<soap:Envelope xmlns:soap="http://schemas.
xmlsoap.org/soap/envelope/">
<soap:Body>
  <x:TransferFundsResponse
          xmlns:x="urn:examples-org:banking">
    <balances>
      <account>
        <id>22-342439</id>
        <balance>33.45</balance>
      </account>
      <account>
        <id>98-283843</id>
        <balance>932.73</balance>
      </account>
    </balances>
 </x:TransferFundsResponse>
</soap:Body>
</soap:Envelope>
```

SOAP-RPC

- Per costruire una RPC usando SOAP tra due applicazioni occorre conoscere:
  - L'URI
  - Nome del metodo
  - Parametri nome/valore
- Le regole SOAP RPC binding definiscono come rappresentare il metodo nel corpo del XML
  - La chiamata viene modellata come singola struttura dove ogni parametro è un campo
  - I nomi e l'ordine fisico dei parametri RPC deve corrispondere con quello del metodo che viene chiamato
- Il body di un messagio SOAP deve contenere il nome del metodo e l'insieme dei parametri
- Alcuni termini utilizzati
  - Accessor è il nome di un elemento che permette l'accesso ad un valore <x>
  - Valore rappresenta il dato es. temperatura corrente
  - Struct è un valore composto in cui ogni accessor ha un nome diverso
  - Array è un valore composto in cui gli accessor hanno lo stesso nome

Esempio RPC

SOAP

The following is a sample SOAP request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /WebService2/CalculateShipping.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/CalculateShippingCost"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlr
  <soap:Body>
    <CalculateShippingCost xmlns="http://tempuri.org/">
      <dblWeight>double</dblWeight>
      <strShippingMethod>string</strShippingMethod>
    </CalculateShippingCost>
  </soap:Body>
</soap:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlr
  <soap:Body>
    <CalculateShippingCostResponse xmlns="http://tempuri.org/">
      <CalculateShippingCostResult>double</CalculateShippingCostResult>
    </CalculateShippingCostResponse>
  </soap:Body>
</soap:Envelope>
```

Metodo da invocare:

double add(ref double x, double y)

Struttura di richiesta:

```
struct add {
  double x;
  double y; }
```

```
<add>
  <x>33</x>
  <y>44</y>
</add>
```

Struttura di risposta:

```
struct addResponse {
      double result;
      double x; }
```

WSDL

- E’ necessario che il client e il server si accordino con l’interfaccia
- WSDL è un linguaggio per descrivere il formato e il protocollo dei Web Service in una maniera standard
  - È definito utilizzando XML
  - Gli elementi WSDL descrivono i dati e le relative operazioni
  - Attraverso un file WSDL si permette a un client di conoscere i servizi offerti dal WS e come utilizzarli (vedi IDL Iterface Definition Language di COM, CORBA)
- Una interfaccia WSDL definisce delle operazioni e le collega a uno o più protocolli

WSDL

- I types, message e portType sono elementi del messaggio WSDL all'interno di definitions e sono usati per definire le operazioni
- **<types>** è usato per definire i tipi base necessari allo scambio dell'informazione
  - (BalanceRequest → numero di conto)
  - Nell'esempio si usano tipi primitivi
- **<message>** è usato per definire il messaggio spedito e ricevuto ed utilizza i tipi definiti in types
  - getFlightArrival → contiene la richiesta di arrivo del volo
- **<portTypes>** è usato per definire il funzionamento delle porte allocate dal servizio come i messaggi da utilizzare in input e output
  - FlightRemote → all'operazione GetFlightArrival è associato un msg di ri chiesta e uno di risposta

WSDL

- Gli elementi binding e service sono usati per definire il protocollo associato all'operazione
- **<binding>** è usato per definire il protocollo da utilizzare per comunicare con la porta su cui è allocata l'operazione (HTTP, SOAP,…)
  - FlightRemoteBinding realizza l'associazione esplicita tra l'operazione GetFlightArrival e il protocollo SOAP
- **<service>** è usato per definire una porta come URL attraverso la quale si trova il servizio
  - Flight realizza l'associazione tra FlightRemoteBinding e il particolare URL in cui si trova il servizio
- WSDL definisce quattro tipi di operazioni attraverso il tag <operations>
  - One-way: è una chiamata asincrona al servizio
  - Request-responce: chiamata sincrona al servizio
  - Sollicit-responce: invia una risposta dopo un sollecito
  - Notification: ricevere una notifica
- In generale WSDL e prodotto e consumato da software tools come Visual Studio .NET e non è necessario conoscerne i dettagli

Esempio WSDL I

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="FlightService" targetNamespace=...>
    <message name="getFlightArrival">
        <part name="String_1" type="xsd:string"/>
        <part name="int_2" type="xsd:int"/>
    </message>
    <message name="getFlightArrivalResponse">
        <part name="result" type="xsd:dateTime"/>
    </message>
    <portType name="FlightRemote">
        <operation name="getFlightArrival">
            <input message="tns:getFlightArrival"/>
            <output message="tns:getFlightArrivalResponse"/>
        </operation>
    </portType>

…continua
```

Esempio WSDL II

```
<binding name="FlightRemoteBinding" type="tns:FlightRemote">
    <operation name="getFlightArrival">
    <input><soap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" namespace="http://allflighttracking.com/wsdl"/>
    </input>
    <output><soap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      use="encoded" namespace="http://allflighttracking.com/wsdl"/>
    </output>
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="rpc"/>
</binding>
<service name="Flight">
    <port name="FlightRemotePort" binding="tns:FlightRemoteBinding">
      <soap:address location="http://localhost:8080/axis/..."/>
    </port>
</service>
</definitions>
```

UDDI

- E’ un registro che mi permette di memorizzare informazioni riguardanti i Web Services
- Sono definite delle API standard
  - Inquiry API definiscono i metodi per la ricerca dell’informazione all’interno del registro
  - Publisching API permettono di creare e modificare l’informazione nel registro
- L’accesso al registro non richiede autenticazione mentre la modifica di informazioni la richiede
- E’ un servizio necessario per collegare i Web Services con i potenziali clienti

UDDI

- E’ un servizio nato per trovare l’informazione necessaria per costruire dei client compatibili con i servizi web
- Oggi significherebbe trovare una definizione WSDL compatibile
- UDDI ha una struttura definita da uno schema XML
  - businessEntity → informazioni sull’organizzazione (nome, contatti, descrizione)
  - businessService → informazioni sul tipo di servizio fornito
- .NET Framework aggiunge Disco che è una tecnologia orientata a fornire le informazioni necessarie per la costruzione di client compatibili

Java APIs for Web Services

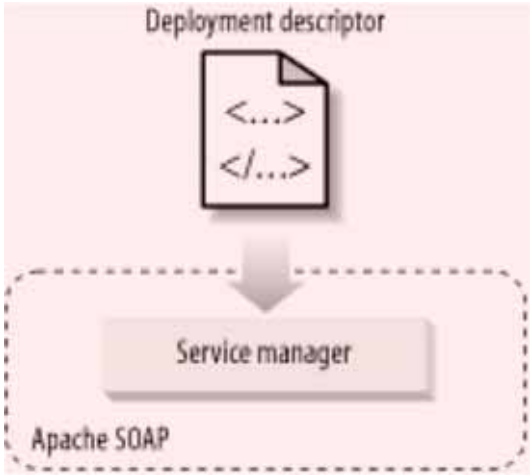
- SOAP messages as Java objects
  - SAAJ (SOAP with Attachments API for Java)
- Programming Model
  - JAX-RPC (JSR101), JSR109, EJB2.1
- Accessing WSDL descriptions
  - JWSDL (JSR110)
- Accessing Web Services Registries
  - JAXR (Java API for XML Registries)

Scrittura Web Services

- Un servizio web è composto da tre componenti
  - un listener per ricevere il messaggio
  - un proxy per tradurre il messaggio in una azione da eseguire
    - es. chiamare un metodo Java
  - codice applicativo per l'esecuzione dell'azione
- Esistono vari toolkit, per la gestione di messaggi SOAP, disponibili agli sviluppatori
  - Apache SOAP per Java
  - .NET Framework
  - Axis
- I toolkit possono essere veri e propri server HTTP o essere installati come parte di un particolare server Web

Apache SOAP

- Utilizziamo il toolkit Apache SOAP
  - implementazione del protocollo SOAP di Apache Software Foundation
  - opera come un servlet all'interno di ogni server HTTP java (Tomcat)
- Scriviamo una classe java Hello composta di un solo metodo
  - pubblic String sayHello (String name)
- Scriviamo il suo descrittore
  - un file XML con la descrizione del metodo e della posizione della classe Hello (questo perché non si utilizza WSDL)
- Distribuiamo il file creato al gestore dei servizi di Apache SOAP
  - TcpTunnelGui possiamo fare il debug dei messaggi SOAP scambiati



Server

- Codice lato server del servizio web
  - La classe va compilata e copiata nel classpath del server web

```
package samples;
public class Hello {
    public String sayHello(String name) {
        return "Hello " + name;
    }
}
```

Deployment Descriptor

- File di descrizione per permettere al server di pubblicare il servizio di sayHello della classe samples.Hello

```
<dd:service xmlns:dd="http://xml.apache.org/xml-soap/deployment"
    id="urn:Example1">
 <dd:provider type="java"
              scope="Application"
              methods="sayHello">
    <dd:java class="samples.Hello"
             static="false" />
 </dd:provider>
 <dd:faultListener>
  org.apache.soap.server.DOMFaultListener
 </dd:faultListener>
 <dd:mappings />
</dd:service>
```

Client per il servizio Hello

```
import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class Example1_client {

  public static void main (String[] args)
       throws Exception {

    System.out.println("\n\nCalling the SOAP Server to say hello\n\n");
    URL url = new URL (args[0]);
    String name = args[1];

    Call call = new Call (  );
    call.setTargetObjectURI("urn:Example1");
    call.setMethodName("sayHello");
    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC;);
    Vector params = new Vector (  );
    params.addElement (new Parameter("name", String.class, name, null));
    call.setParams (params);

    System.out.print("The SOAP Server says: ");

    Response resp = call.invoke(url, "");

    if (resp.generatedFault (  )) {
      Fault fault = resp.getFault (  );
      System.out.println ("\nOuch, the call failed: ");
      System.out.println ("  Fault Code   = " + fault.getFaultCode (  ));
      System.out.println ("  Fault String = " + fault.getFaultString (  ));
    } else {
      Parameter result = resp.getReturnValue (  );
      System.out.print(result.getValue (  ));
      System.out.println(  );
    }
  }
}
```

Chiamata: java samples.Hello
http://localhost/soap/servlet/rpcrouter James

Output:

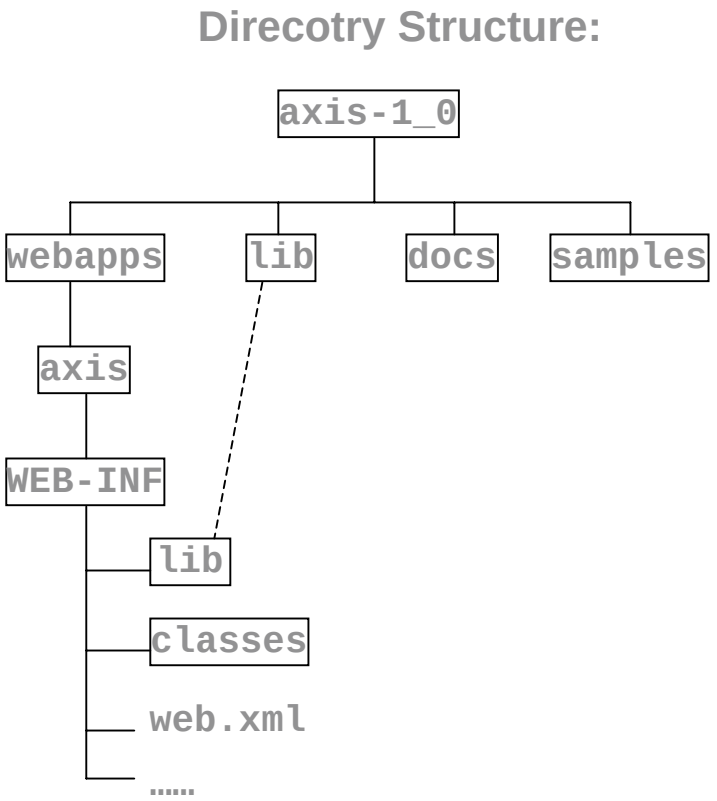
Calling the SOAP Server to say hello
The SOAP Server says: Hello James

AXIS

- Apache Axis è una nuova generazione di implementazione di SOAP
  - include un supporto integrato per l'uso e la creazione automatica di documenti WSDL
    - WSDL2Java e Java2WSDL
  - permette una invocazione dinamica dei servizi web basata sull'analisi a runtime dei meta dati riguardanti il servizio
  - è un semplice server stand-alone che può essere inserito anche in Tomcat
  - implementa le JAX-RPC API che sono delle API standard per la scrittura di servizi in Java
  - http://ws.apache.org/axis/

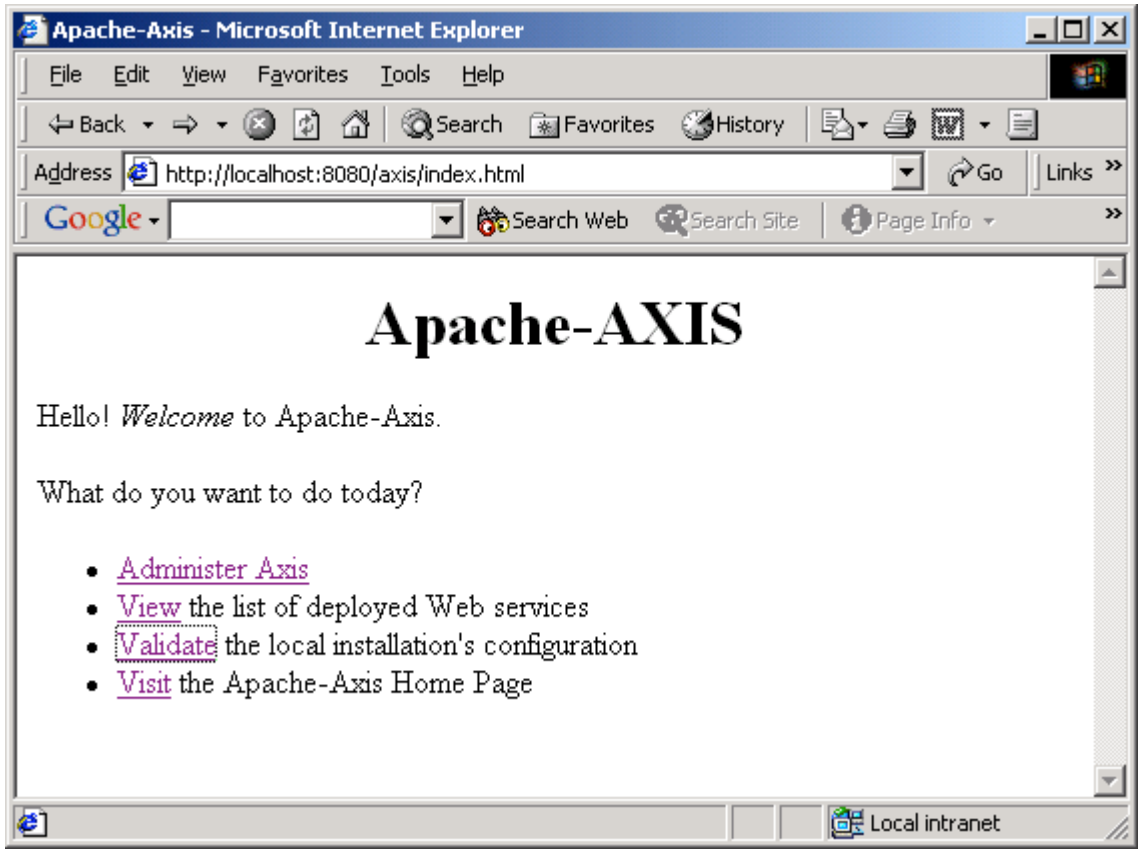
Install & Deploy Apache Axis

- Essere sicuri di avare
  - J2SE SDK 1.3 or 1.4: We will use 1.4
  - A Servlet Container: si puà usare Tomcat4.0.1
- Scaricare da axis1-1.zip http://xml.apache.org/axis
- Estrarre il file a posizionarlo in una dir.
- Deploy Axis.
  - Copiare `webapps\axis` in `webapps` directory di Tomcat.
  - In alternativa, modificare `server.xml` di Tomcat.
- Run Tomcat: **`bin\startup`**



Test AXIS

- Provare il seguente URL http://localhost:8080/axis



Pubblicare un WS con AXIS

- JWS (Java Web Service) Files - Instant Deployment
  - Rinominare il file Calculator.java in Calculator.jws e spostarlo nella cartella webapps/axis
  - …abbiamo già creato un servizio web!!! Lo si può gia accedere seguendo URL
    - http://localhost:8080/axis/Calculator.jws
  - Esaminiamo la descrizione WSDL utilizzando l'URL http://localhost:8080/axis/Calculator.jws?wsdl
  - Axis automaticamente ricerca il file, compila la classe e converte le chiamate SOAP in corrette invocazioni dei metodi

```
public class Calculator {
  public int add(int i1, int i2)
  {
 return i1 + i2;
  }

  public int subtract(int i1, int i2)
  {
 return i1 - i2;
  }
}
```

Scrivere il Client

- Ci sono diversi modi per scrivere il client
  - Usando Dynamic Invocation Interface (DII)
  - Usando uno Stubs generato dalla descrizione del Servizio WSDL
  - Usando un Dynamic Proxy

Scrivere il client richiede più lavoro di scrivere il servizio

Dynamic invocation

- Dynamic Invocation Interface (DII)
  - No è uno stubs statico per il client
  - Configurazione del servizio on the fly
  - Creare javax.xml.rpc.Service da una istanza di Javax.xml.rpc.ServiceFactory
  - Creare javax.xml.rpc.Call e configurare
    - gli endpoint, le operazioni, i parametri e i tipi di ritorno
  - Esecuzione con Call.invoke()
  - O con Axis che usa WSDL per le informazioni sul servizio
  - Flessibile e adatto per i servizi semplici

Client AXIS – usando DII

```
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.namespace.QName;

public class AddFunctionClient {
  public static void main(String [] args) {
    try {
      String endpoint = "http://localhost:8080/axis/Calculator.jws";
      Service  service = new Service();
      Call   call = (Call) service.createCall();
      call.setOperationName(new QName(endpoint, "add"));
      call.setTargetEndpointAddress( new java.net.URL(endpoint) );
      Integer ret = (Integer)call.invoke(new Object[]{new Integer(5), new Integer(6)});
      System.out.println("add(5, 6) = " + ret);
    } catch (Exception e) {
          System.err.println("Execution failed. Exception: " + e);
    }
  }
}
```

Punto di ingresso al servizio Web

Invochiamo il servizio passando un array di parametri

Client AXIS

- Richiesta SOAP generata dal client
  - con la chiamata al metodo call.invoke()

```
<?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:echoString xmlns:ns1="http://soapinterop.org/">
   <arg0 xsi:type="xsd:string">Hello!</arg0>
 </ns1:echoString>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Client – usando Stubs

Generate the stubs:

```
java org.apache.axis.wsdl.WSDL2Java \
  http://localhost:8080/axis/Calculator.jws?wsdl
```

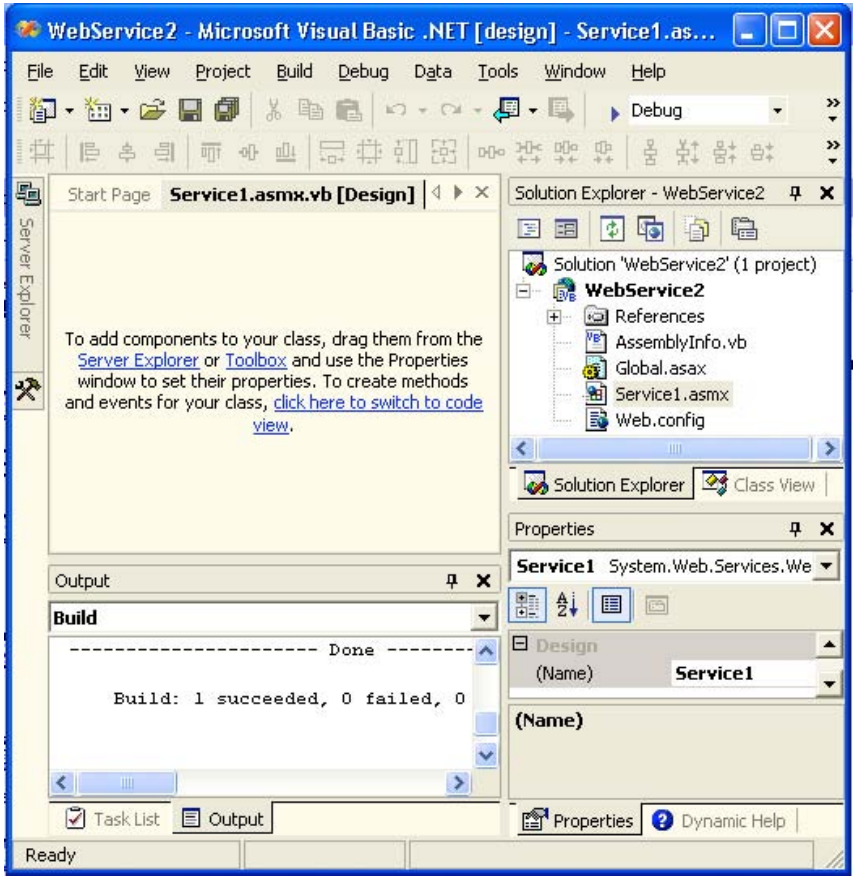
```
Import localhost.*;
public class AddFunctionClient{
  public static void main(String [] args) {
    try {
      AddFunctionService afs = new AddFunctionServiceLocator();
      AddFunction af = afs.getAddFunction();
      System.out.println("addInt(5, 3) = " + af.add(5, 3));
    } catch (Exception e) {
      System.err.println("Execution failed. Exception: " + e);
    }
  }
}
```

CalculatorClient – usando Dynamic Proxy

```
import javax.xml.namespace.QName;
import javax.xml.rpc.*;
public class CalculatorClient {
  public static void main(String [] args) {
    try {
      String wsdlUrl = "http://localhost:8080/axis/Calculator.jws?wsdl";
      String nameSpaceUri = "http://localhost:8080/axis/Calculator.jws";
      String serviceName = “CalculatorService";
      String portName = “Calculator";
      ServiceFactory serviceFactory = ServiceFactory.newInstance();
      Service afs = serviceFactory.createService(new java.net.URL(wsdlUrl),
                new QName(nameSpaceUri, serviceName));
      CalculatorServiceIntf afsIntf = (CalculatorServiceIntf)afs.getPort(
                new QName(nameSpaceUri, portName), CalculatorServiceIntf.class);
      System.out.println("addInt(5, 3) = " + afsIntf.add(5, 3));
      } catch (Exception e) {
         System.err.println("Execution failed. Exception: " + e);
      }
   }
}
```

Web Service in .NET

- Necessario avere installato:
  - IIS (attuale versione 6.0)
  - ASP.NET 1.1
  - Framework .NET 1.1
- Prima fermare Apache: Pannello di controllo -> Strumenti amministrazione -> Servizi -> Apache -> arresta
- Avviare IIS
- Si può creare un servizio web senza ricorrere a Visual Studio .NET utilizzando il compilatore csc.exe
- Utilizzando Visual Studio.NET i passi elementari:
  - Creare un nuovo progetto con il template ASP.NET
    - Nel campo name inserire il nome del servizio
    - Nel campo location inserire il server web dove dovrebbe essere collocato il servizio
- Aprire il file Service1.asmx rinominarlo con il nome della classe e scrivere il codice del servizio web



Codice per la costruzione WS

I servizi web devono importare la System.Web.Service

La classe deve essere dichiarata pubblica.

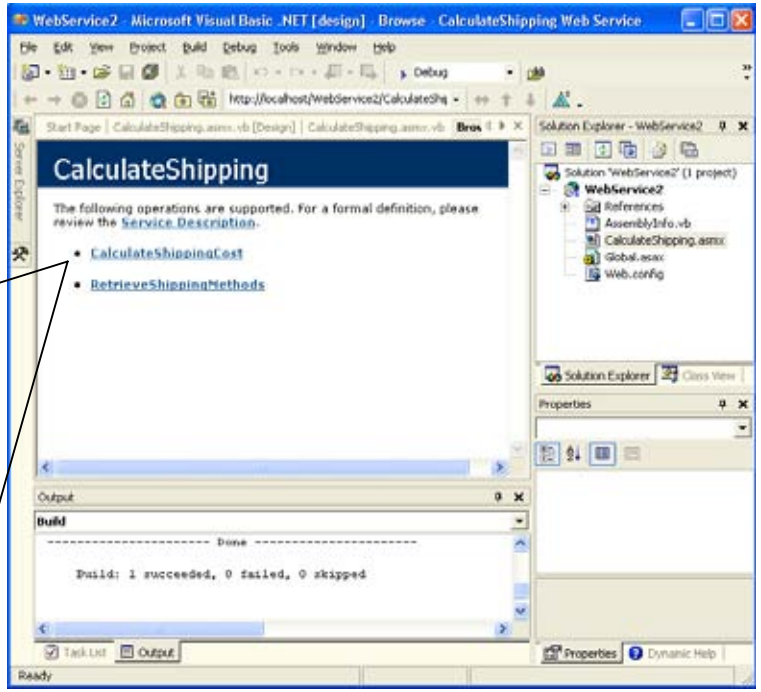
Eredito le funzionalità System. Web.Service.WebService

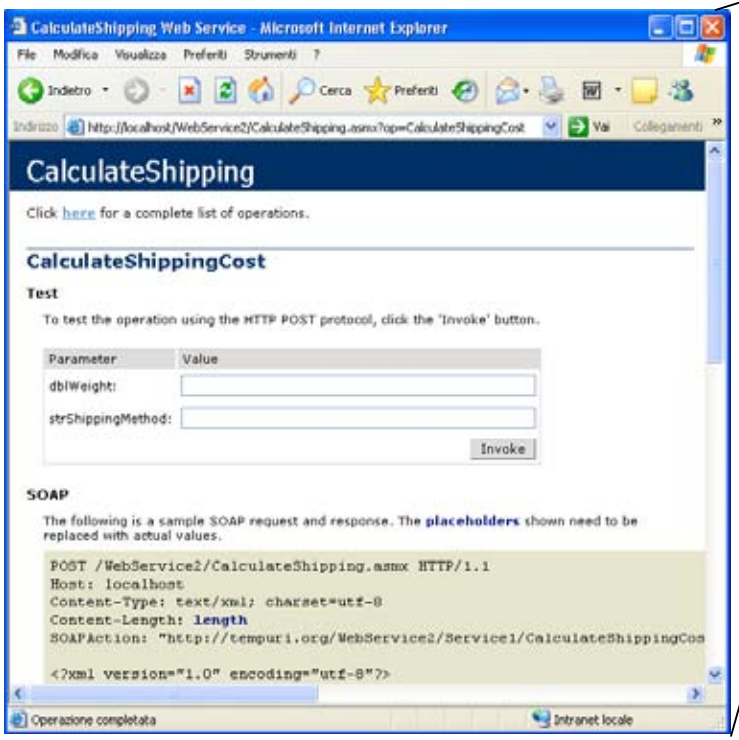
La dichiarazioni di funzioni vanno precedute dal suffisso <WebMethod()>. Questo le identifica come metodi web che verranno esposti dal servizio.

```
Imports System.Web.Services

<System.Web.Services.WebService(Namespace:="http://tempuri.org/WebService2/Service1")> _
Public Class CalculateShipping
    Inherits System.Web.Services.WebService

    Web Services Designer Generated Code

    <WebMethod()> Public Function CalculateShippingCost(ByVal dblWeight As Double, _
            ByVal strShippingMethod As String) As Double

        Dim dblHandlingCost As Double
        Dim dblShippingCost As Double

        dblHandlingCost = 2.0   'handling cost

        Select Case strShippingMethod
            Case "Overnight"
                dblShippingCost = dblWeight * 6
            Case "2nd Day"
                dblShippingCost = dblWeight * 4
            Case "Regular Mail"
                dblShippingCost = dblWeight * 1.5
        End Select

        CalculateShippingCost = dblHandlingCost + dblShippingCost

    End Function

    <WebMethod()> Public Function RetrieveShippingMethods() As String

        RetrieveShippingMethods = "Overnight, 2nd Day, Regular Mail"

    End Function
```

Esecuzione e prova del servizio

- Non è necessaria la compilazione della classe basta digitare l'URL del server web contenente CalculateShipping.asmx
- Vengono descritti i servizi del server web. Consente il test delle funzioni.





Client Visual Studio.NET

- Creare un progetto dal template ASP.NET Application
- Aggiungere un Web reference
  - Project|Add Web reference
  - Ricercare il servizio desiderato attraverso UDDI o Disco
  - Ottenuto un riferimento al servizio lo usiamo…
    ShippingService.CalculateShipping ws= new ShippingService.CalculateShipping();
    ws.calculateShippingCost(5,5);
- Aggiungendo il riferimento Visual Studio ha creato un Web Service proxy class DLL
  - Contiene i dettagli dei metodi disponibili e il codice per invocare e ricevere una risposta dal servizio
  - Per creare la DLL proxy si può anche utilizzare il file WSDL.exe

In Eclipse

- Web Tools Platform (WTP)
  - Include dei wizard per la costruzione dei Web Service
  - Web Standard Tools per il supporto alla creazione di applicazioni web
    - WSDL/XSD con editor grafici
    - Web service UDDI explorer
  - http://www.eclipse.org/downloads/index.php
- Web Service Validation Tools (WSVT)
  - Verifica e analisi dei Web Service
    - WS-I Profile 1.0 requisiti per l'interoperabiolità
    - http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html
  - http://www.eclipse.org/wsvt/

WS.*

- WS Security
  - Controllo degli accessi
  - Utilizzo di cifrature per la protezione dei messaggi
  - Specifiche per la sicurezza
    - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
- WS Trust
  - Utilizzare vari livelli di fiducia
  - Scambio di token sicuri
- WS Coordination
  - Orchestrazione/workflow
    - linguaggio BPEL standard de facto per la descrizione dei processi
- WS Transaction

Riferimenti

- J. Snell, D. Tidwell, P. Kulchenko *“Programming Web Service with SOAP”* O’Reilly 2002
- D.Chappell *“Java Web Services”* O’Reilly 2002
- K.Hoffaman *“.NET Framework”* Wrox 2001
- E.Newcomer *“Understanding Web Services”* Addison Wesley 2002