http://www.jugsardegna.org

Java Web Services con Axis

II Parte

di

Nicola Mura

nikhol@tiscali.it

Introduzione

Nella prima parte dell'articolo(1) è stato introdotto il tema dei Web services dal punto di vista offerto dal tool Apache Axis (2). Sono stati illustrati i passi fondamentali ma sufficienti a creare un Web service in grado di ricevere e restituire tipi semplici (interi, stringhe, date, etc.). Ciò è ancora troppo poco per implementare soluzioni circa le problematiche più comuni e per dimostrare la reale utilità di Axis. Possiamo infatti pensare alle classiche applicazioni, rappresentate da cataloghi, shopping cart, ricerche ed elaborazioni ad hoc, per cui risulta necessario raffinare ulteriormente la gestione dei tipi.

Serializzazione e deserializzazione

Abbiamo visto, nello scorso articolo (1) (la cui lettura è consigliata per comprendere ciò che verrà detto di seguito), come sia semplice per Axis gestire Web services che utilizzano tipi semplici i quali vengono mappati in maniera automatica negli appositi tipi SOAP/WSDL (3), in base alle definizioni di schema XML, XSD (Xml, Schema, Definition). A questo punto occorre soffermarci un momento sul fatto che qualsiasi informazione viaggi sul canale SOAP sia in realtà un documento XML. In questo formato, dunque, deve essere convertito tutto ciò che stia per percorrere il cammino client/server e viceversa. Per quanto concerne gli oggetti necessari come parametri di una invocazione RPC e il corrispettivo valore di ritorno, essi devono appunto essere rappresentati all'interno di una struttura XML. Axis esegue questa operazione per noi, ovvero è in grado di:

- serializzare un oggetto Java in un blocco XML contenente le medesime informazioni,
- deserializzare un blocco XML rappresentante un dato nel suo omologo oggetto Java.

E' facile intuire, dunque, con quale ordine vengano eseguite queste operazioni.
La serializzazione viene effettuata sugli oggetti che il client invia al service engine come parametri richiesti dal metodo. Il Web service engine deserializza tali informazioni XML istanziando gli appositi oggetti Java e valorizzandone i campi. A seguito dell'esecuzione della logica Web service, il valore da restituire subisce anch'esso una serializzazione prima di essere spedito al client sul "canale" SOAP. Infine Il client deserializza il blocco XML restituito rendendo possibile un utilizzo locale dell'oggetto associato.
Questa procedura di [de]serializzazione viene eseguita per tutti i tipi di dato, da quelli semplici che Axis gestisce in maniera trasparente, a quelli costituiti da aggregati di tipi semplici, che Axis gestisce in base ad alcune indicazioni che devono essere fornite durante la fase di deployment.

L'esempio (un semplice catalogo)

Implementare un Web service di un catalogo, permette di mettere a fuoco gli accorgimenti necessari alla gestione dei tipi complessi in Axis. In realtà quello che verrà presentato sarà un approccio "leggero", in quanto si lascerà fare il grosso del lavoro alle funzionalità automatiche di Axis. Tuttavia occorre ricordare che Axis offre una estesa e approfondita serie di strumenti di controllo di tipi complessi. L'obiettivo dell'articolo è dunque evidenziare cosa cambia rispetto all'approccio utilizzato per realizzare l'esempio già trattato nella prima parte (1). Supponiamo, infatti, che il catalogo che stiamo per realizzare debba impiegare il tipo presentato nel **listato 1** il quale segue il pattern Value Object (6).

Listato 1 – Classe ProdottoVO

```
package miopackage.vo;

import java.io.*;

public class ProdottoVO implements Serializable {
```

```
  private int codice;
  private String nome;
  private boolean disponibile;
  private float prezzo;

  public ProdottoVO() {
  }
  public int getCodice() {
    return codice;
  }
  public void setCodice(int codice) {
    this.codice = codice;
  }
  public void setNome(String nome) {
    this.nome = nome;
  }
  public String getNome() {
    return nome;
  }
  public void setDisponibile(boolean disponibile) {
    this.disponibile = disponibile;
  }
  public boolean isDisponibile() {
    return disponibile;
  }
  public void setPrezzo(float prezzo) {
    this.prezzo = prezzo;
  }
  public float getPrezzo() {
    return prezzo;
  }

}
```

ProdottoVO sarà appunto il tipo per gli oggetti che il nostro servizio catalogo fornirà al client.
Il Web service, per esempio, dovrebbe avere le seguenti funzionalità:

- poter fornire l'elenco di tutti i prodotti presenti in catalogo,
- ricercare un singolo prodotto in base al suo codice identificativo.

Ciò viene fatto in maniera semplificata dalla classe *CatalogoWS* mostrata nel **listato 2**.

Listato 2 – Classe CatalogoWS

```
package miopackage;

import java.util.*;
import miopackage.vo.*;

public class CatalogoWS {

  public CatalogoWS() { }

  public Collection getListaProdotti() {
    Vector lista = new Vector();

    //prodotto1
    ProdottoVO prodotto1 = new ProdottoVO();
    prodotto1.setCodice(1);
    prodotto1.setNome("Lettore MP3");
    prodotto1.setDisponibile(true);
    prodotto1.setPrezzo(25.99f);
    lista.add(prodotto1);

    //prodotto2
    ProdottoVO prodotto2 = new ProdottoVO();
    prodotto2.setCodice(2);
    prodotto2.setNome("Display LCD");
    prodotto2.setDisponibile(true);
```

```
    prodotto2.setPrezzo(199.99f);
    lista.add(prodotto2);

    return lista;
  }


  public ProdottoVO getProdotto(int codice) {
    ProdottoVO prodotto = new ProdottoVO();
    prodotto.setCodice(codice);
    prodotto.setNome("Sub Woofer attivo");
    prodotto.setDisponibile(true);
    prodotto.setPrezzo(154.99f);
    return prodotto;
  }
}
```

Il deployment di questo servizio, non è dissimile da quello mostrato per la parte I dell'articolo (1).
La differenza, oltre alla presenza di due metodi (*getProdotto* e *getListaProdotti*), sta nel fatto che ora occorre comunicare ad Axis i tipi di oggetto da inviare al client, uno *ProdottoVO*, l'altro *Collection* (7) di *ProdottoVO,* in relazione a quale dei metodi previsti venga invocato. Nel **listato 3** è rappresentato il deployment descriptor per la classe CatalogoWS

Listato 3 – deploy2.wsdd

```
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="urn:catalogoWS" provider="java:RPC" style="rpc" use="encoded">
    <parameter name="className" value="miopackage.CatalogoWS"/>
    <parameter name="allowedMethods" value="getProdotto, getListaProdotti"/>
    <parameter name="scope" value="Session"/>
    <beanMapping qname="myNS:ProdottoVO" xmlns:myNS="urn:catalogoWS"
                 languageSpecificType="java:miopackage.vo.ProdottoVO"/>
  </service>
</deployment>
```

Oltre ai vari parametri generali del Web Service, si può notare la sezione dedicata al tipo *ProdottoVO* dentro il tag *<beanMapping>*. Il fatto molto importante è che quest'ultimo è sufficiente ad Axis per gestire i tipi di ritorno dei due metodi ovvero *ProdottoVO* e *Collection* di *ProdottoVO*!
Ciò è dovuto al fatto che oggetti collettivi vengono gestiti in maniera autonoma da Axis a patto di avere le informazioni riguardanti il tipo elementare (*ProdottoVO*, per il nostro caso). Il tag *<beanMapping>* è un tag compatto che sostituisce la seguente forma più generale:

Listato 4 – segmento di type-mapping alternativo al tag <beanMapping>

```
<typeMapping
  xmlns:ns=" urn:catalogoWS "
  qname=" myNS:ProdottoVO "
  type="java:miopackage.vo.ProdottoVO "
  serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
  deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
/>
```

Il **listato 4** è quindi totalmente equivalente all'elemento *<beanMapping>* presente all'interno del **listato 3**; infatti i primi tre attributi sono presenti in entrambi, mentre in neretto sono visibili quelli che specificano le classi serializer e deserializer di default. Questa seconda forma, dunque, consente di indicare eventuali nostre classi di [de]serializzazione.
L'utilizzo delle classi di default *org.apache.axis.encoding.ser.BeanSerializerFactory* e *org.apache.axis.encoding.ser.BeanDeserializerFactory* ha, però, un importante vincolo che è quello di trattare oggetti che seguano lo standard dei Bean java (costruttore senza parametri, get e set per ogni proprietà, etc.)
Come già spiegato (1), il deployment deve essere effettuato dopo aver copiato, sotto Tomcat (4), le classi necessarie al funzionamento del WS (in questo caso *CatalogoWS* e *ProdottoVO*). Dopo un restart di Tomcat,

occorre eseguire la classe *org.apache.axis.client.AdminClient* passando come parametro il nome del file definito nel **listato 3**, *deploy2.wsdd*.

E' utile controllare il documento WSDL generato durante la fase di deployment per il servizio così da rendersi conto dell'indispensabile lavoro che Axis compie per noi. Nel **listato 5** è visibile il documento WSDL che a termine deployment è possibile raggiungere all'URL:

http://localhost:8080/axis/services/urn:catalogoWS?wsdl

Listato 5 – rappresentazione WSDL del Web Service CatalogoWS

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions    targetNamespace="http://localhost:8080/axis/services/urn:catalogoWS"
xmlns="http://schemas.xmlsoap.org/wsdl/"     xmlns:apachesoap="http://xml.apache.org/xml-
soap"                     xmlns:impl="http://localhost:8080/axis/services/urn:catalogoWS"
xmlns:intf="http://localhost:8080/axis/services/urn:catalogoWS"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"     xmlns:tns1="urn:catalogoWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <schema targetNamespace="urn:catalogoWS" xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
        <complexType name="ProdottoVO">
          <sequence>
            <element name="codice" type="xsd:int"/>
            <element name="disponibile" type="xsd:boolean"/>
            <element name="nome" nillable="true" type="xsd:string"/>
            <element name="prezzo" type="xsd:float"/>
            </sequence>
          </complexType>
    </schema>
  </wsdl:types>
  <wsdl:message name="getListaProdottiRequest">
  </wsdl:message>
  <wsdl:message name="getProdottoResponse">
    <wsdl:part name="getProdottoReturn" type="tns1:ProdottoVO"/>
  </wsdl:message>
  <wsdl:message name="getListaProdottiResponse">
    <wsdl:part name="getListaProdottiReturn" type="soapenc:Array"/>
  </wsdl:message>
  <wsdl:message name="getProdottoRequest">
    <wsdl:part name="codice" type="xsd:int"/>
  </wsdl:message>
  <wsdl:portType name="CatalogoWS">
    <wsdl:operation name="getListaProdotti">
      <wsdl:input message="impl:getListaProdottiRequest" name="getListaProdottiRequest"/>
      <wsdl:output message="impl:getListaProdottiResponse"
                 name="getListaProdottiResponse"/>
    </wsdl:operation>
    <wsdl:operation name="getProdotto" parameterOrder="codice">
      <wsdl:input message="impl:getProdottoRequest" name="getProdottoRequest"/>
      <wsdl:output message="impl:getProdottoResponse" name="getProdottoResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="urn:catalogoWSSoapBinding" type="impl:CatalogoWS">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getListaProdotti">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getListaProdottiRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                  namespace="http://miopackage" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getListaProdottiResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                  namespace="http://localhost:8080/axis/services/urn:catalogoWS"
                  use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getProdotto">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getProdottoRequest">
```

```
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   namespace="http://miopackage" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getProdottoResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   namespace="http://localhost:8080/axis/services/urn:catalogoWS"
                   use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CatalogoWSService">
    <wsdl:port binding="impl:urn:catalogoWSSoapBinding" name="urn:catalogoWS">
      <wsdlsoap:address location="http://localhost:8080/axis/services/urn:catalogoWS"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Spiegare tutto il **listato 5** richiederebbe troppo tempo per cui conviene soffermarsi sugli aspetti legati ai tipi, in linea con il tema di questo articolo.
Si nota, in neretto, la definizione del tipo complesso *ProdottoVO* che viene descritto mediante il tag *<complexType>* come sequenza di elementi di tipi base.
Nel secondo blocco in neretto viene definito il tipo di ritorno dei due metodi ovvero *ProdottoVO* per *getProdotto()* e *Array* per *getListaProdotti()*. Notare che l'attribuzione del tipo WSDL *Array,* al tipo nativo *Collection* è stata effettuata in maniera trasparente da Axis.

Anche per il client di questo Web service non si avranno grosse differenze rispetto all'esempio mostrato in precedenza. L'unico accorgimento da impiegare è la "registrazione" del tipo *ProdottoVO* grazie alla quale Axis individua nel listato XML di risposta, quel segmento da deserializzare ed utilizzare.

Listato 6 – ClasseClient CatalogoWS

```
package miopackage;

import java.net.*;
import java.rmi.*;
import java.util.*;

import javax.xml.namespace.*;
import javax.xml.rpc.*;

import miopackage.vo.*;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.ser.*;

public class ClientCatalogoWS {

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

      QName qnameProd = new QName("urn:catalogoWS", "ProdottoVO");
      Class classeProd = ProdottoVO.class;
      call.registerTypeMapping(classeProd, qnameProd, BeanSerializerFactory.class,
                                                      BeanDeserializerFactory.class);

      //richiamo il metodo getProdotto
      call.setOperationName(new QName("urn:catalogoWS", "getProdotto"));
      Object rispostaWS = call.invoke(new Object[]{new Integer(1)});
      ProdottoVO prodotto = (ProdottoVO) rispostaWS;
      System.out.println("Il prodoto scelto è:");
      visualizza(prodotto);

      //richiamo il metodo getListaProdotti
      call.setOperationName(new QName("urn:catalogoWS", "getListaProdotti"));
      rispostaWS = call.invoke(new Object[]{});
      Collection lista = (Collection) rispostaWS;
      System.out.println("Il catalogo comprende:");
      Iterator iter = lista.iterator();
      while (iter.hasNext()) {
        ProdottoVO item = (ProdottoVO) iter.next();
```

```
        visualizza(item);
      }
    } catch (Exception ex) {
      System.out.println("Si è verificata l'eccezione: " + ex.toString());
    }
  }

  public static void visualizza(ProdottoVO prodotto) {
    if (prodotto == null) {
      return;
    }
    System.out.println("nome: " + prodotto.getNome());
    System.out.println("codice: " + prodotto.getCodice());
    String disponibile = (prodotto.isDisponibile()) ? "si" : "no";
    System.out.println("disponibile: " + disponibile);
    System.out.println("prezzo: " + prodotto.getPrezzo());
  }
}
```

Come per l'esempio descritto nella prima parte (1), scegliamo come client il modello DII (Dynamic Invocation Interface) secondo le specifiche JAX-RPC (5). Il codice è analogo ma in più contiene la sezione di registrazione dei tipi lato client, visibile in neretto.
Dapprima si istanzia un *javax.xml.namespace.Qname* (qualified name) contenente il nome del tipo, "*ProdottoVO"* e il nome del servizio, "*urn:catalogoWS*", cui è associato. Notare come questi valori siano gli stessi inseriti nel deployment descriptor (**listato 3**) all'interno del tag *<beanMapping>*. La registrazione del tipo *ProdottoVO* avviene grazie all'invocazione del metodo *registerTypeMapping()* dell'oggetto *Call*. I parametri passati oltre al *Qname* appena descritto e all'oggetto *ProdottoVO.class,* sono le classi di serializzazione e deserializzazione di Axis raggiungibili al package *org.apache.axis.encoding.ser*

E' utile dare uno sguardo agli envelope SOAP che vengono scambiati tra client ed endpoint del Web service durante l'esecuzione dell'RPC.
Nel caso dell'invocazione del metodo *getProdotto()* si ha:

Richiesta	`<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"`
	`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">`
	`<soapenv:Body>`
	`<ns1:getProdotto`
	`soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`
	`xmlns:ns1="urn:catalogoWS">`
	`<ns1:arg0 xsi:type="xsd:int">1</ns1:arg0>`
	`</ns1:getProdotto>`
	`</soapenv:Body>`
	`</soapenv:Envelope>`

Risposta	`<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"`
	`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">`
	`<soapenv:Body>`
	`<ns1:getProdottoResponse`
	`soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`
	`xmlns:ns1="urn:catalogoWS">`
	`<ns1:getProdottoReturn href="#id0"/>`
	`</ns1:getProdottoResponse>`
	`<multiRef id="id0" soapenc:root="0"`
	`soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`
	`xsi:type="ns2:ProdottoVO"`
	`xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"`
	`xmlns:ns2="urn:catalogoWS">`
	`<codice xsi:type="xsd:int">1</codice>`
	`<disponibile xsi:type="xsd:boolean">true</disponibile>`
	`<nome xsi:type="xsd:string">Sub Woofer attivo</nome>`
	`<prezzo xsi:type="xsd:float">154.99</prezzo>`
	`</multiRef>`
	`</soapenv:Body>`
	`</soapenv:Envelope>`

Tabella 1 – Envelope dei messaggi SOAP relativi all'invocazione del metodo getProdotto()

Nella richiesta mostrata in **tabella 1** si nota la presenza del parametro necessario all'invocazione del metodo *getProdotto()*. Nella risposta è facilmente riconoscibile l'oggetto serializzato restituito, di tipo *ProdottoVO*.

Per la chiamata al metodo *getListaProdotti()* si ha:

Richiesta	`<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"`
	`  xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
	`  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">`
	`  <soapenv:Body>`
	`  <ns1:getListaProdotti`
	`  soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`
	`  xmlns:ns1="urn:catalogoWS"/>`
	`  </soapenv:Body>`
	`</soapenv:Envelope>`

Risposta	`<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"`
	`  xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
	`  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">`
	`  <soapenv:Body>`
	`  <ns1:getListaProdottiResponse`
	`  soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`
	`  xmlns:ns1="urn:catalogoWS">`
	`  <ns1:getListaProdottiReturn href="#id0"/>`
	`  </ns1:getListaProdottiResponse>`
	`  <multiRef id="id0" soapenc:root="0"`
	`  soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`
	`  xsi:type="ns2:Vector"`
	`  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"`
	`  xmlns:ns2="http://xml.apache.org/xml-soap">`
	`  <item href="#id1"/>`
	`  <item href="#id2"/>`
	`  </multiRef>`
	`  <multiRef id="id1" soapenc:root="0"`
	`  soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`
	`  xsi:type="ns3:ProdottoVO" xmlns:ns3="urn:catalogoWS"`
	`  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">`
	`  <codice xsi:type="xsd:int">1</codice>`
	`  <disponibile xsi:type="xsd:boolean">true</disponibile>`
	`  <nome xsi:type="xsd:string">Lettore MP3</nome>`
	`  <prezzo xsi:type="xsd:float">25.99</prezzo>`
	`  </multiRef>`
	`  <multiRef id="id2" soapenc:root="0"`
	`  soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"`
	`  xsi:type="ns4:ProdottoVO" xmlns:ns4="urn:catalogoWS"`
	`  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">`
	`  <codice xsi:type="xsd:int">2</codice>`
	`  <disponibile xsi:type="xsd:boolean">true</disponibile>`
	`  <nome xsi:type="xsd:string">Display LCD</nome>`
	`  <prezzo xsi:type="xsd:float">199.99</prezzo>`
	`  </multiRef>`
	`  </soapenv:Body>`
	`</soapenv:Envelope>`

Tabella 2 – Envelope dei messaggi SOAP relativi all'invocazione del metodo getListaProdotti()

Nella **tabella 2** si nota come, correttamente, non vi siano parametri associati alla chiamata del metodo *getListaProdotti()*. Nel messaggio di risposta è evidenziata, in neretto, la sezione corrispondente alla restituzione degli oggetti presenti in catalogo, ovviamente ognuno di essi è poi rappresentato secondo lo stesso schema visto per la risposta mostrata **in tabella1**.

Conclusioni

In questo articolo sono stati illustrati i passaggi necessari all'impiego di tipi non semplici nei Web Services implementati con Axis .

E' stato evidenziato come Axis aiuti notevolmente il programmatore nella implementazione/configurazione di Web services con trattamento evoluto dei tipi, infatti esegue autonomamente alcune importanti operazioni che richiedono notevole dispendio di tempo, se effettuate manualmente.

Questi aspetti sono stati messi in luce da un punto di vista semplice e facilmente comprensibile che spinga maggiormente il lettore a concentrarsi sui punti chiave dell'approccio ai Web services fornito da Axis.

Bibliografia

(1) David A. Chappell & Tyler Jewell, "Java Web Services", O'Reilly HOPS, 2002

(2) W.Crawford, J.Kaplan, “J2EE Design Patterns”, O’Reilly, 2003

Riferimenti

(1) Nicola Mura, “Java Web Services con Axis – I Parte”, Programmazione.it, 2003: http://www.programmazione.it/index.php?entity=earticle&idArticle=1405&idArea=18
(2) Apache Axis: http://ws.apache.org/axis/
(3) Web Services Descritpion Language (WSDL): http://www.w3.org/TR/wsdl
(4) Apache Tomcat: http://jakarta.apache.org/tomcat/
(5) Java API for XML-Based RPC (JAX-RPC): http://java.sun.com/xml/jaxrpc/
(6) Fabrizio Gianneschi, “Recensione: J2EE Design Patterns”, Programmazione.it, 2003: http://www.programmazione.it/index.php?entity=enews&idNews=8448&idArea=18
(7) The Java Tutorial – Collection: http://java.sun.com/docs/books/tutorial/collections/