DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

PRACTICAL DEVELOPMENT OF A WEB SERVICE

JAX-WS 2.0 – Java API for XML Web Services

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma

- Specifica basata su **annotazioni**
- Applicata su classi ed interfacce in modo da definire e gestire automaticamente
  - il protocollo di comunicazione remota
    - **SOAP**
  - il *marshalling*
    - **XML**
  - lo scambio di messaggi previsto
    - **WSDL / XML-Schema**
- Condiziona anche la codifica client (se Java)
- Alternativa a **JAX-RPC** (*Java API for XML Remote Procedure Calls*)
  - basata su file di mapping e descrittori XML

Premesse ed osservazioni

- Il file WSDL verrà automaticamente generato dal framework
  - JBossWS
- Non è previsto il mantenimento di alcuno stato conversazionale !!
  - I Web Service, per propria natura, sono **stateless**
    - La realizzazione di **Web Service conversazionali** è ottenibile tramite estensioni dello standard
      - **WS-Addressing**
- Occorre ricordare che i Web Service sono creati per essere ***platform-independent***
  - Moduli client e server possono essere codificati in linguaggi differenti
  - Non è richiesta *alcuna* importazione di file bytecode nel client che siano residenti nel server

Implementazione basata su POJO/Servlet

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

1. Creare un'interfaccia che dichiari le operazioni offerte dal Web Service

```
package it.uniroma1.dis.pseudoinfostud.control;

public interface ElencoUtentiAttiviService {
    public String[] getElencoUtentiAttivi();
}
```

Implementazione basata su POJO/Servlet

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma

2. Dichiarare, per mezzo di annotazioni, le caratteristiche del Web Service

```
package it.uniroma1.dis.pseudoinfostud.control;

import javax.jws.WebMethod;
import javax.jws.WebResult;
import javax.jws.WebService;

@WebService(targetNamespace =
"http://www.dis.uniroma1.it/master/pseudoinfostud/ElencoUtenti")

public interface ElencoUtentiAttiviService {

@WebMethod(operationName="getElencoUtentiAttivi")
@WebResult(name="elencoUtentiAttivi")
        public String[] getElencoUtentiAttivi();
}
```

Implementazione basata su POJO/Servlet

3. Codificare una classe che implementi quei metodi (**endpoint**)

```
package it.uniroma1.dis.pseudoinfostud.control;

public class ElencoUtentiAttiviEndpoint
implements ElencoUtentiAttiviService {

@Override
public String[] getElencoUtentiAttivi() {
        List<String> utentiAttivi =
                new ArrayList<String>();
         ...
        return utentiAttivi.toArray();
}
}
```

Implementazione basata su POJO/Servlet

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI
SAPIENZA
UNIVERSITÀ DI ROMA

4. Dichiarare, per mezzo di annotazioni, le caratteristiche dell'endpoint

```
package it.uniroma1.dis.pseudoinfostud.control;

import javax.jws.WebService;

@WebService(
        name = "ElencoUtentiAttiviEndpoint",
        serviceName = "ElencoUtentiAttivi",
        targetNamespace = "http://www.dis.uniroma1.it/asos/pseudoinfostud/ElencoUtenti",
        endpointInterface = "it.uniroma1.dis.pseudoinfostud.control.ElencoUtentiAttiviService")

public class ElencoUtentiAttiviEndpoint implements ElencoUtentiAttiviService {
        public String[] getElencoUtentiAttivi() {
        // ...
        }
}
```

Implementazione basata su POJO/Servlet

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI
SAPIENZA
UNIVERSITÀ DI ROMA

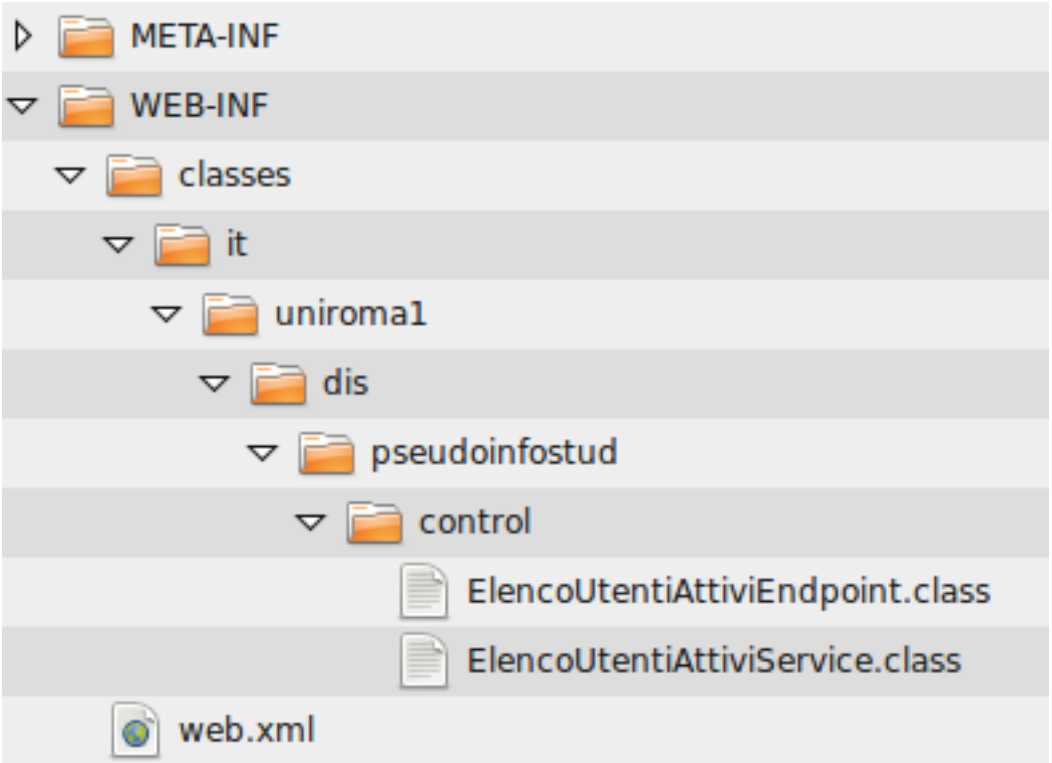
5. Impostare, su web.xml, i riferimenti remoti verso l'endpoint

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>PseudoInfostudWSServer</display-name>
  <servlet>
    <display-name>ElencoUtentiAttiviEndpoint</display-name>
    <servlet-name>ElencoUtentiAttiviEndpoint</servlet-name>
    <servlet-class>
it.uniroma1.dis.pseudoinfostud.control.ElencoUtentiAttiviEndpoint
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ElencoUtentiAttiviEndpoint</servlet-name>
    <url-pattern>/ElencoUtentiAttivi</url-pattern>
  </servlet-mapping>
</web-app>
```

Implementazione basata su POJO/Servlet

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma

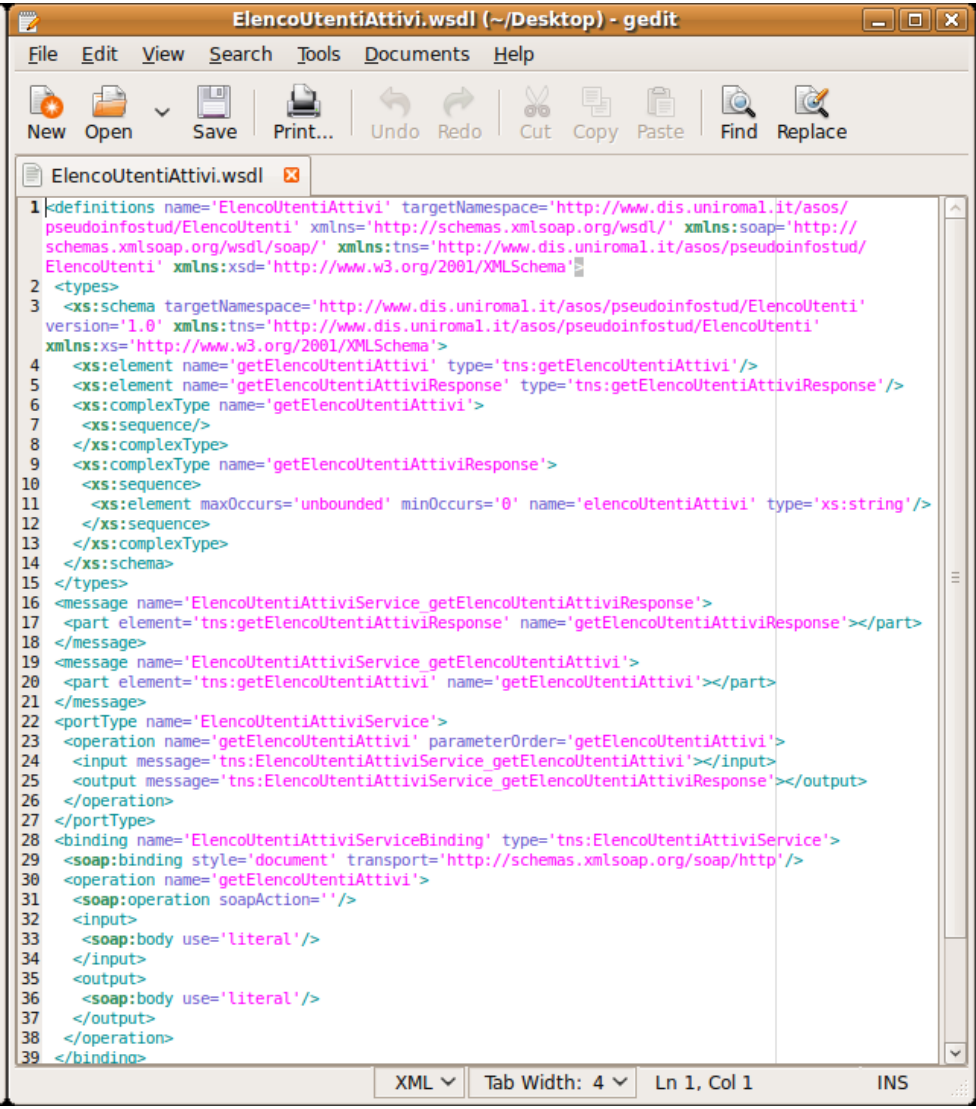
6. Dispiegare sull'Application Server il **WAR** contenente bytecode e metadati creati



Implementazione basata su POJO/Servlet

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA



Osservazioni POJO / Servlet

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma

- Sebbene si compili un'applicazione web basata su Servlet, la classe che implementa il Web Service non deve dichiarare esplicitamente l'estensione della classe javax.servlet.GenericServlet
- L'implementazione è estremamente semplice
- Qualunque classe può divenire l'endpoint di un Web Service, una volta apposte opportunamente le annotazioni

Il client

- Non è necessario che il client sia codificato in Java
  - Esula dagli scopi di questo corso l'integrazione con altre tecnologie...
    - ... dunque nel resto della lezione considereremo la codifica in Java!
- Occorre ricordare che non solo i valori e le istanze, ma anche la definizione degli stessi, sono basati su XML
  - WSDL contiene tutte le informazioni, con riferimenti a XML-Schema interni o esterni
- Nel progetto client, anche se codificato in Java, non si devono importare direttamente le classi definite sul server!
  - L'esperienza insegna che, provando, si ottengono a run-time errori alquanto criptici...
    com.sun.xml.ws.model.RuntimeModelerException: runtime modeler error: Wrapper class *bla.bla.bla.Bla* is not found. Have you run APT to generate them?

Gli stub

- Occorre generare gli **stub** delle classi presenti sul server
  - Gli stub devono essere prodotti sulla base del solo WSDL
- Il comando per ottenere da shell (prompt) la codifica e la compilazione di tali stub è wsimport
  - Esempio d'uso

```
wsimport
    -d ‹directory_destinazione_file_compilati›
    -s ‹directory_destinazione_file_sorgente_generati›
    -keep
    -p ‹package_classi_stub›
    ‹uri_wsdl›
```

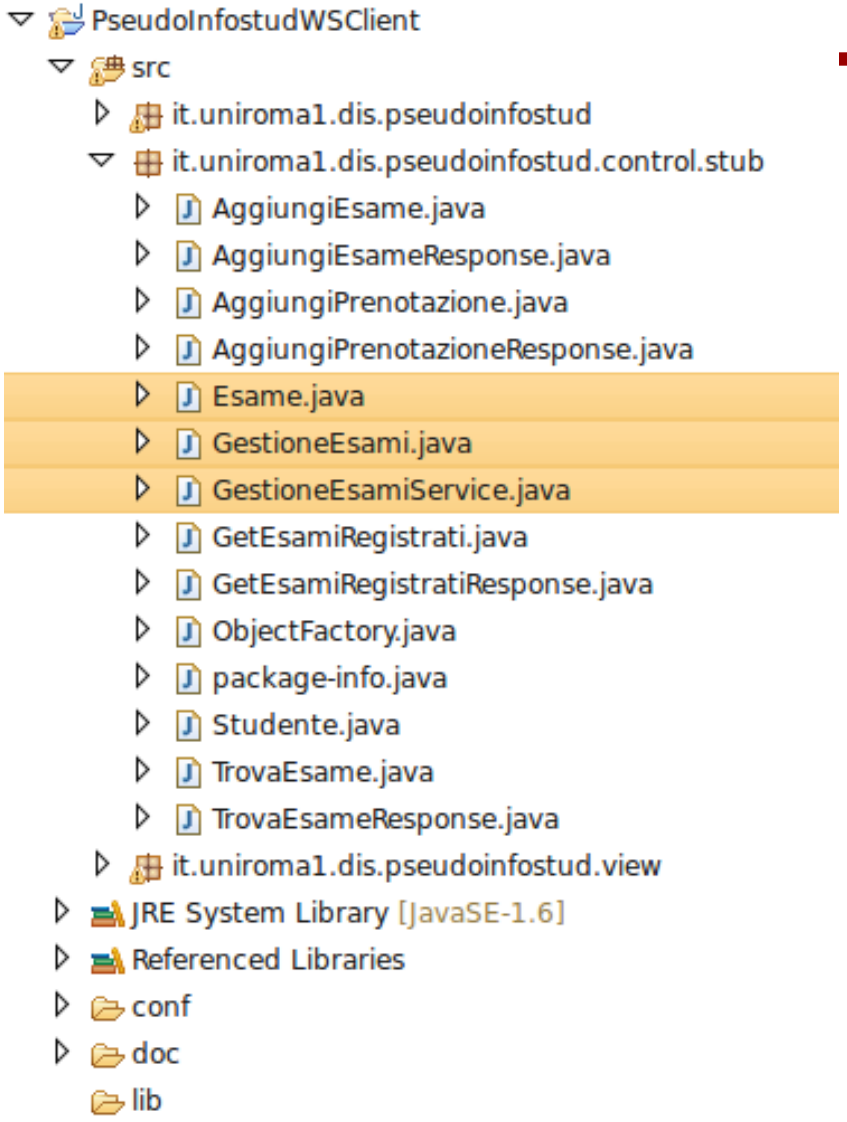
Gli stub

- Occorre generare gli **stub** delle classi presenti sul server
  - Gli stub devono essere prodotti sulla base del solo WSDL
- Il comando per ottenere da shell (prompt) la codifica e la compilazione di tali stub è wsimport
  - Caso reale

```
wsimport
-d /home/mobidis/workspace/SimpleWSClient/bin
-s /home/mobidis/workspace/SimpleWSClient/src
-keep
-p stub
http://127.0.0.1:8080/ElencoUtentiAttivi/ElencoUtentiAttivi?wsdl
```

Gli stub

- Le classi di Stub vanno aggiunte al progetto Java del client
  - Il mantenimento dei sorgenti è opzionale...
  - ... ma molto utile a scopo didattico
- Attenzione: sono classi diverse dalla controparte sul server



Invocazione di operazioni del WS

- Dati gli stub (importati nella classe client), per ottenere un riferimento locale all'endpoint (proxy):
  ‹EndpointInterfaceName› endpoint =
  new **‹ServiceName›**()
  .get**‹ServicePortName›**();

Invocazione di operazioni del WS

Dipartimento di Informatica
e Sistemistica Antonio Ruberti

Sapienza
Università di Roma

- Su tale classe, potranno essere invocati i metodi dichiarati tramite annotazione @WebService
  - Il nome dei metodi corrisponderà al nome specificato come attributo name dell'annotazione, se presente!