Java Enterprise Edition

Gabriele Tolomei

DAIS – Università Ca' Foscari Venezia

Java Web Services

Web Services: SOAP vs. RESTful

- 2 diversi tipi di “Web Services”
- I Web Services SOAP sono quelli “classici”
  - Si basano sullo scambio di messaggi SOAP (un dialetto XML) sul protocollo HTTP
  - Forniscono il supporto per transazioni, sicurezza, scambio asincrono di messaggi, etc.
  - Approccio “pesante”
- I Web Services RESTful sono più “recenti”
  - Si basano **esclusivamente** sullo scambio di messaggi tramite protocollo HTTP (GET, POST, PUT, DELETE)
  - Il contenuto dei messaggi è solitamente JSON o XML
  - Approccio “leggero”

Java Web Services: SOAP vs. RESTful

- Java mette a disposizione molte librerie per costruire ed interagire con i Web Services
- Alcuni esempi di librerie per i Web Services SOAP sono:
  - JAX-WS → **noi ci concentreremo su questa libreria!**
  - Apache Axis
- Alcuni esempi di librerie per i Web Services RESTful sono:
  - Restlets
  - Spring REST Facilities

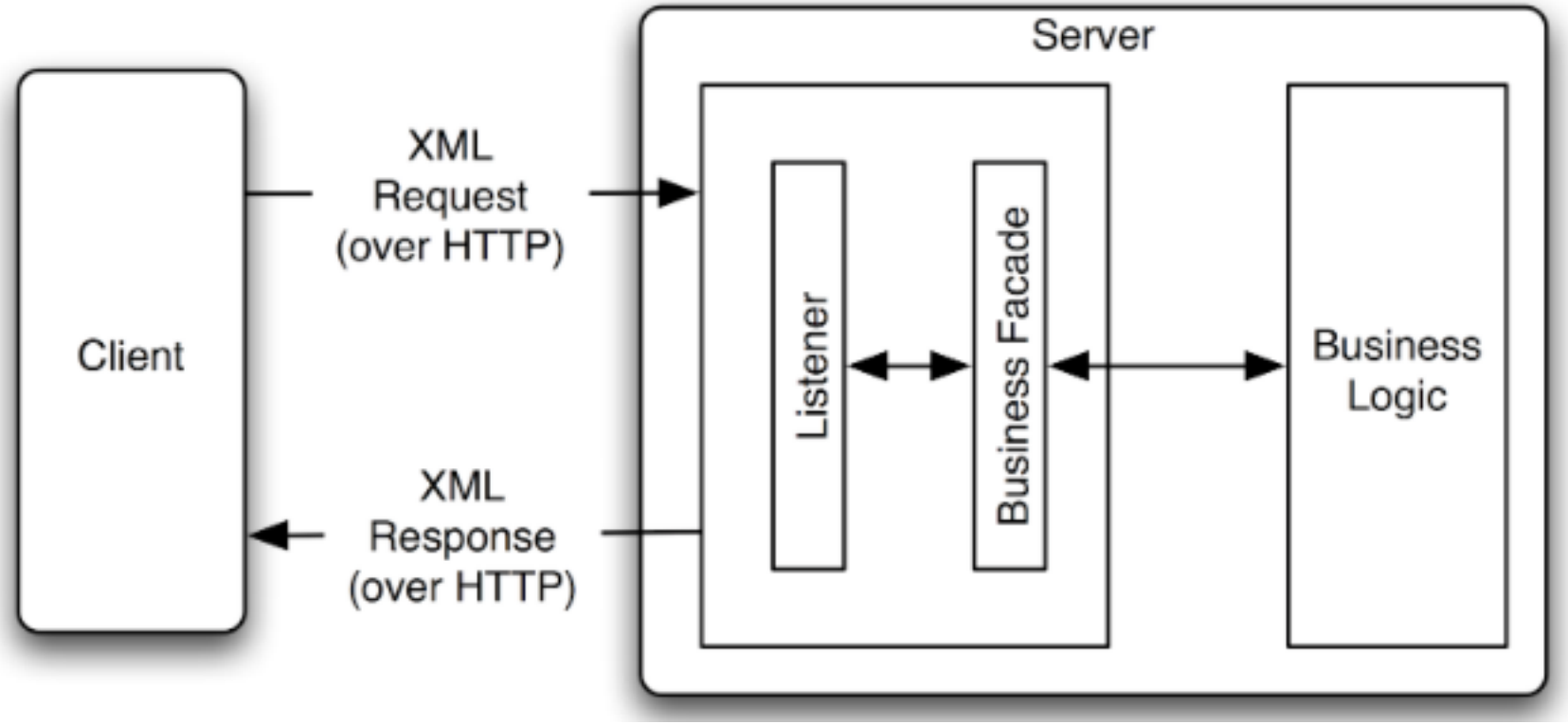
Web Services SOAP

- Nuova tecnologia che consente l'interazione e l'interoperabilità tra applicazioni distribuite
- Sono applicazioni a sé stanti che possono essere pubblicate, localizzate, ed invocate in modo remoto (via Internet)
- I Web Services possono implementare semplici funzioni ma anche processi più complessi

Web Services SOAP

- Costituiscono un altro esempio di Remote Procedure Call (RPC)
- Sono indipendenti dalla piattaforma e dalla tecnologia
  - Ad es. un client scritto in C può interagire con un WS in esecuzione su un container Java EE
- Si basano su standard noti: HTTP + XML
- Meno efficienti rispetto ad altre soluzioni come RMI, Jini, CORBA, DCOM, etc. ma più “facili”

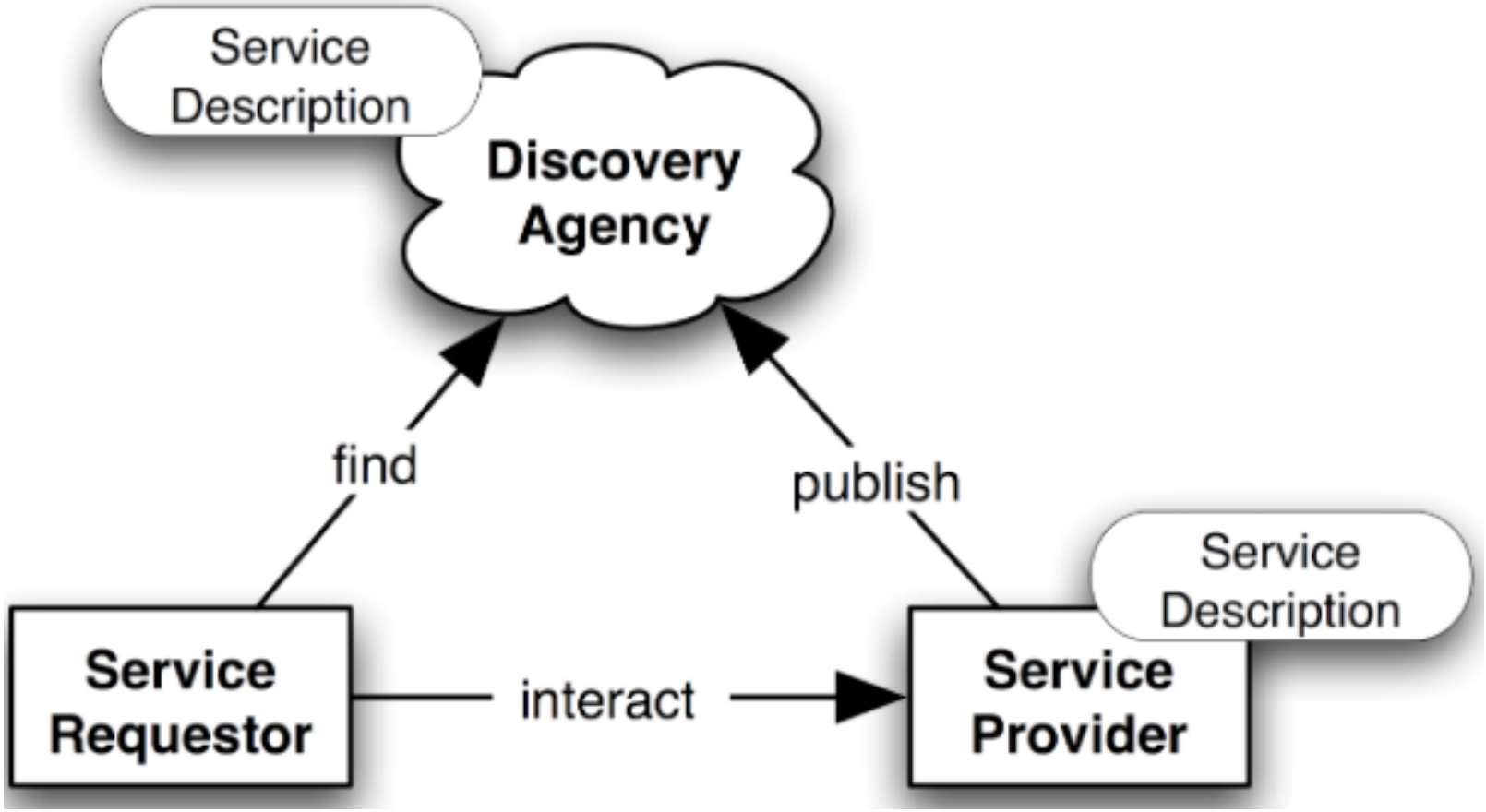
Web Services SOAP



Service Oriented Architecture (SOA)

- Modello di architettura “a servizi”
- Gli elementi di questo modello sono:
  - SOAP (Simple Object Access Protocol)
    - Specifica il formato del messaggio, l’encoding, e tutte le convenzioni per rappresentare la chiamata e la risposta della procedura remota
  - UDDI (Universal Description Discovery and Integration)
    - Standard per la pubblicazione dei WSs
  - WSDL (Web Service Description Language)
    - “Manifesto” del WS (descrizione delle caratteristiche del servizio)

Service Oriented Architecture (SOA)



JAX-WS: Java API for XML-bases Web Services

- Chiamata di procedura remota su XML
- Scambio di messaggi via SOAP
- Trasmissione dei messaggi via HTTP
- JAX-WS “nasconde” allo sviluppatore la complessità del protocollo SOAP
  - Il supporto a runtime traduce le chiamate della API JAX-WS da/a messaggi SOAP
  - Consente l’accesso a WS in esecuzione su altre piattaforme

JAX-WS: Java API for XML-bases Web Services

- Chiamata di procedura remota su XML
- Scambio di messaggi via SOAP
- Trasmissione dei messaggi via HTTP
- JAX-WS “nasconde” allo sviluppatore la complessità del protocollo SOAP
  - Il supporto a runtime traduce le chiamate della API JAX-WS da/a messaggi SOAP
  - Consente l’accesso a WS in esecuzione su altre piattaforme

JAX-WS: Java API for XML-bases Web Services

- Lato server:
  - Lo sviluppatore deve solo specificare le procedure remote che il WS espone
  - Ovvero definire i metodi in un'intefaccia apposita scritta in Java
  - Lo sviluppatore implementa l'interfaccia creata in una o più classi tramite l'uso di "annotazioni" (@)
- Lato client:
  - Si crea un "proxy" (ovvero un oggetto locale al client che rappresenta il servizio remoto)
  - Si invocano i metodi sul proxy locale.
  - La libreria JAX-WS penserà a "tradurre" le chiamate ai metodi sul proxy locale in chiamate al servizio remoto via SOAP
  - Infine, le risposte SOAP ottenute dal servizio remoto saranno nuovamente gestite da JAX-WS e tradotte per il client

Web Services su JBoss

- Su JBoss 5.x i componenti Java EE possono agire sia da “fornitori” (providers) che “consumatori” (consumers) di Web Services
- Le applicazioni Java EE possono esporre un WS come:
  - Stateless Session Bean (EJB Tier)
  - Oggetto Java “tradizionale” o Plain Old Java Object (Web Tier)

Web Services su JBoss

- Il supporto per i Web Services su JBoss è affidato a deploy/jbossws.sar
- Il protocollo per il trasporto dei messaggi (HTTP) è gestito dal Web container (Apache Tomcat)
- Il protocollo che specifica il formato dei messaggi (SOAP) è gestito da Apache Axis
- Per vedere la lista di tutti i WSs disponibili sulla propria istanza server JBoss:

  http://localhost:8080/jbossws/services

Java Web Services (Web Tier)

- I WSs creati nel Web Tier sono oggetti “normali” mascherati da Servlets
- Nessuna interfaccia specifica da implementare come nel caso delle HttpServlet

```
public class HelloWebService {
  public String hello(String name) {
    return "Hello " + name + "!";
  }
}
```

Java Web Services (Web Tier)

- Creare un progetto Web dinamico su Eclipse
  - Ad es., HelloWebService
- Definire un package
  - Ad es., it.sipe.javaee.ws.example
- Creare una nuova classe Java che rappresenta l'end-point (punto di accesso) del WS
  - Ad es., HelloWebService
  - Nota che su JBoss il nome di questa pseudo-Servlet **NON** deve essere del tipo *Servlet.java
- Implementare il metodo del servizio
  - Ad es., hello(String nome)

Java Web Services (Web Tier)

```java
package it.sipe.javaee.ws.example;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;


@WebService(targetNamespace = "http://www.sipe.it", serviceName = "HelloWebService") // 1.
@SOAPBinding(style = Style.RPC) // 2.
public class HelloWebService {

    public HelloWebService() {
        // TODO Auto-generated constructor stub
    }

    @WebMethod // 3.
    @WebResult(name="result") // 4.
    public String hello(@WebParam(name="nome") String nome) { // 5.
        return "Hello "+nome+"!";
    }

}
```

Java Web Services: Annotazioni

1. **@WebService** → è obbligatoria e specifica che la classe è un Web
2. **@SOAPBinding** → specifica il tipo di binding SOAP una volta deployato il servizio (RPC)
3. **@WebMethod** → espone il metodo come parte dei servizi offerti dal WS
4. **@WebResult** → indica il risultato dell'invocazione del metodo
5. **@WebParam** → parametro del metodo

Java Web Services: Deployment

- Occorre aggiungere nel deployment descriptor (web.xml) una entry apposita (tag) che specifichi la pseudo-Servlet che abbiamo creato
- Il nome della pseudo-Servlet è importante! Sarà quello con cui ci riferiremo al Web Service

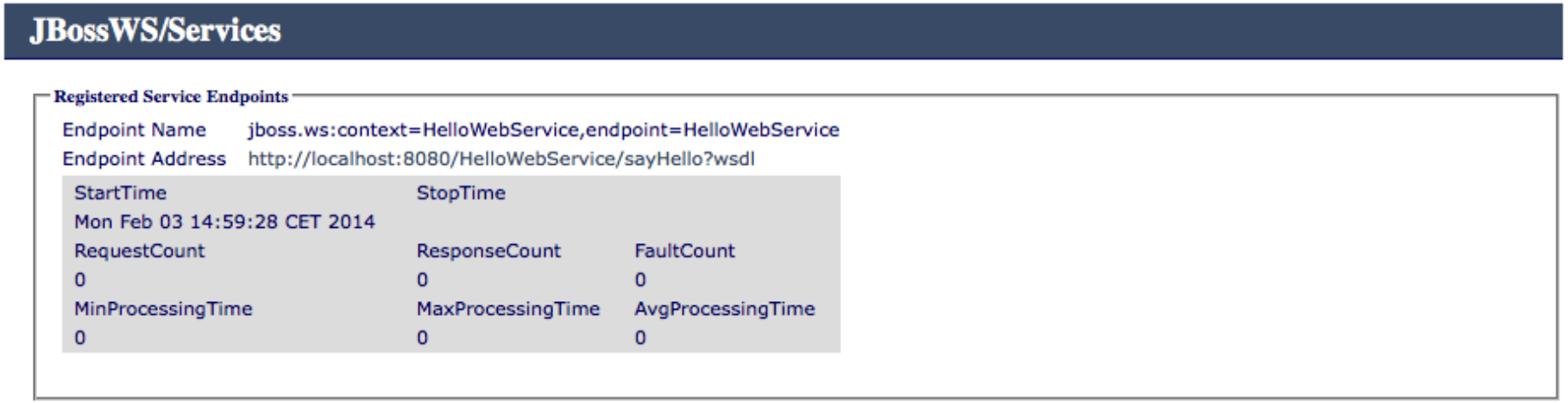
```
<servlet>
  <servlet-name>HelloWebService</servlet-name>
  <servlet-class>it.sipe.javaee.ws.example.HelloWebService</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloWebService</servlet-name>
  <url-pattern>/sayHello</url-pattern>
</servlet-mapping>
```

url-pattern indica dove si troverà il file WSDL associato al WS

Java Web Services: Packaging

- Creare l'archivio .war dal progetto Web dinamico
- Specificare il server su cui eseguire il deployment
- Verificare che il WS sia correttamente elencato nella lista dei WSs disponibili:

http://localhost:8080/jbossws/services



Java Web Services: Client

- Qualsiasi client, per poter dialogare con il nostro WS, deve "conoscere" le caratteristiche del/dei servizi esposti:
  - Firma del/dei metodi, parametri, etc.
- Tutte queste caratteristiche sono contenute sottoforma di un file XML standard (WSDL) associato al nostro WS e reperibile all'URL:

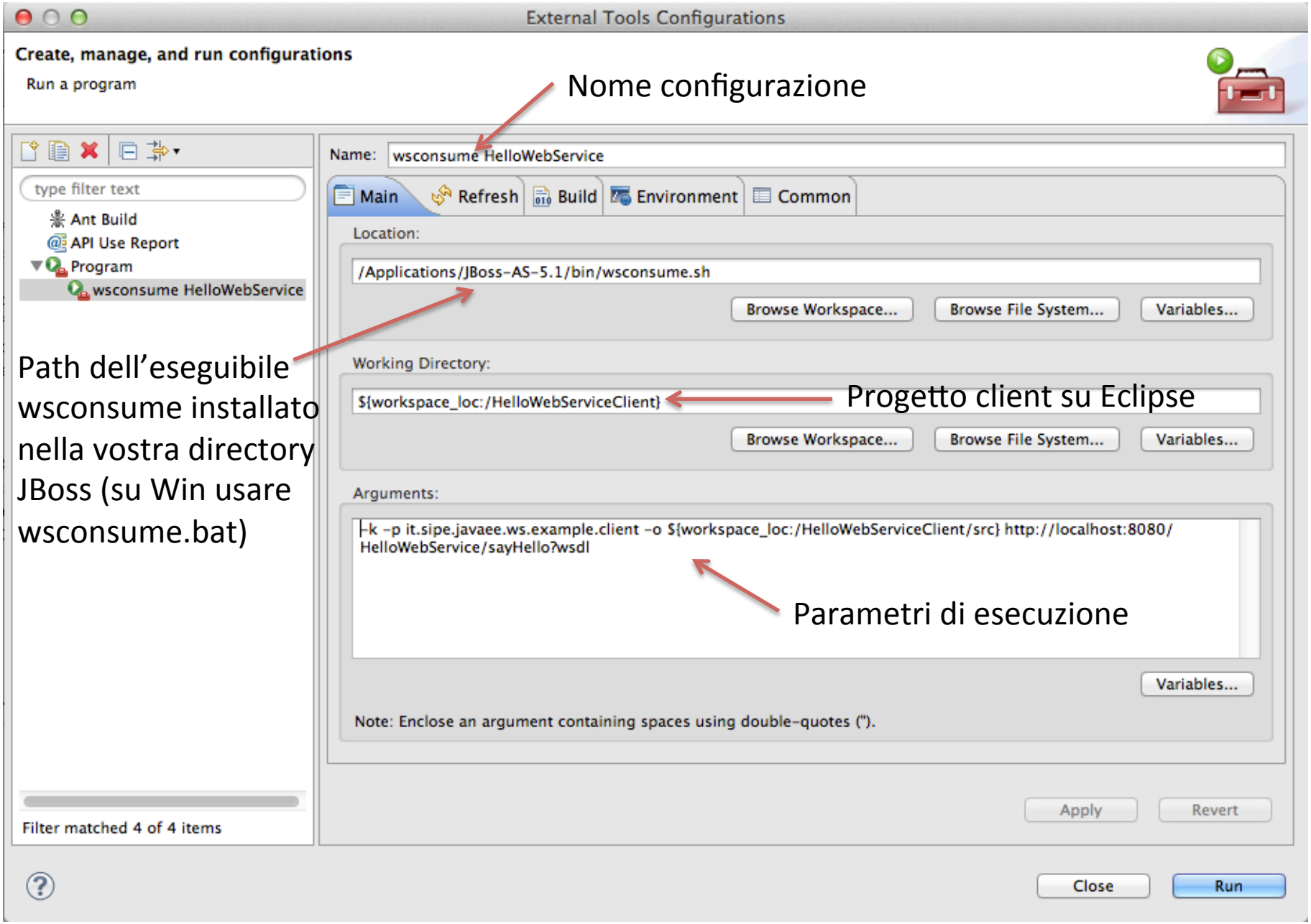
http://localhost:8080/HelloWebService/sayHello?wsdl

Java Web Services: Client

- Il client può accedere al WS in 2 modi:
  - Tramite interfaccia Java remota
    - Se questa è disponibile
    - Solitamente se il servizio è stato sviluppato/è in esecuzione su ambiente Java
  - Dinamicamente
    - Se non è disponibile alcuna interfaccia
    - Solitamente quando il servizio è sviluppato in un altro linguaggio

Java Web Services: Client

- Su Eclipse creare un nuovo progetto Java (non un progetto Web dinamico) che conterrà il codice del client
- Dal menu principale Run → External Tools → External Tool Configurations
- Cliccare su “Program”


External Tools Configurations
Create, manage, and run configurations
Run a program
Nome configurazione
Name: wsconsume HelloWebService
type filter text
Ant Build
API Use Report
Program
wsconsume HelloWebService
Filter matched 4 of 4 items
Main
Refresh
Build
Environment
Common
Location:
/Applications/JBoss-AS-5.1/bin/wsconsume.sh
Browse Workspace...
Browse File System...
Variables...
Working Directory:
${workspace_loc:/HelloWebServiceClient}
Progetto client su Eclipse
Browse Workspace...
Browse File System...
Variables...
Arguments:
-k -p it.sipe.javaee.ws.example.client -o ${workspace_loc:/HelloWebServiceClient/src} http://localhost:8080/HelloWebService/sayHello?wsdl
Parametri di esecuzione
Variables...
Note: Enclose an argument containing spaces using double-quotes (").
Apply
Revert
Close
Run
Path dell'eseguibile wsconsume installato nella vostra directory JBoss (su Win usare wsconsume.bat)

wsconsume.bat (.sh)

- I parametri di esecuzione specificano:

  -k → indica di generare il codice sorgente del client

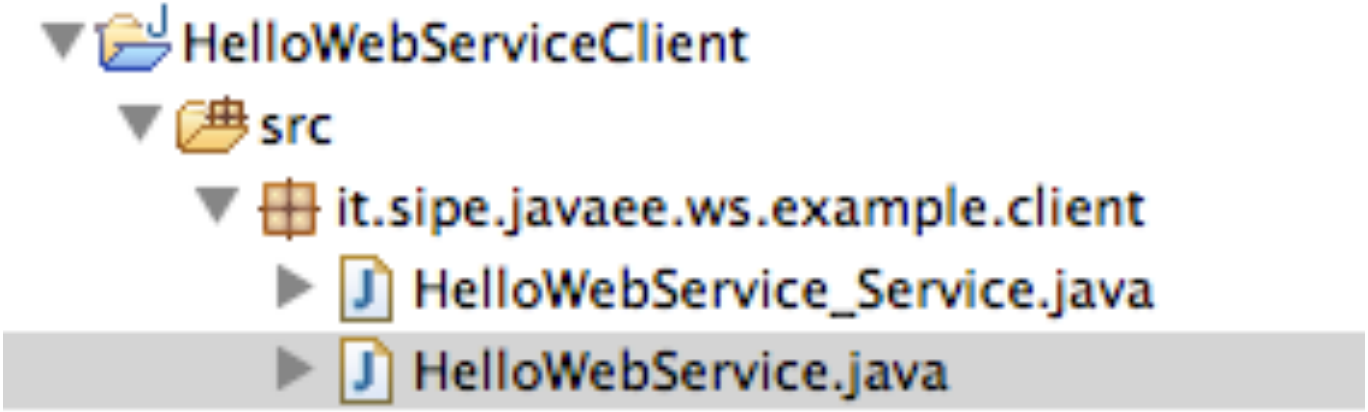
  -p → indica il package in cui creare il codice sorgente

  -o → indica in quale progetto generare il codice (nel nostro caso, il progetto Java che abbiamo appena creato)

- Eseguire cliccando su “Run” e fare un refresh del progetto

wsconsume.bat (.sh)

- Se tutto è andato a buon fine verranno creati 2 file:
  - HelloWebService.java → interfaccia al Web Service
  - HelloWebService_Service.java → "artefatto" locale lato client usato per accedere al Web Service remoto



HelloWebServiceClient.java

- Infine, occorre creare la vera e propria classe client che userà l'artefatto per invocare il Web Service remoto

```
package it.sipe.javaee.ws.example.client;

public class HelloWebServiceClient {

    public HelloWebServiceClient() {
        // TODO Auto-generated constructor stub
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        // 1. artefatto lato client
        HelloWebService_Service wsClient = new HelloWebService_Service();
        // 2. oggetto "proxy" locale che rappresenta il Web Service remoto
        HelloWebService wsClientProxy = wsClient.getHelloWebServicePort();
        // 3. chiamata del metodo (remoto) tramite la chiamata del corrispondente metodo dell'oggetto proxy locale
        System.out.println(wsClientProxy.hello("Gabriele"));

    }

}
```

Eseguire il Client: Nota

- La slide seguente presuppone che il server JBoss sia stato avviato da linea di comando (run.bat o run.sh)
- Se JBoss viene avviato dall'interno di Eclipse occorre:
  - aggiungere la seguente dipendenza nello script di avvio

    -Djava.endorsed.dirs=${JBOSS_HOME}/lib/endorsed
  - se non presenti, copiare all'interno di questa directory i seguenti .jar da ${JBOSS_HOME}/common/lib/:
    - jboss-(native-)saaj.jar, jboss-(native-)jaxws.jar, jboss-(native-)jaxrpc.jar, jaxb-api.jar, xercesImpl.jar, xalan.jar, serializer.jar

Eseguire il Client

- Occorre compilare il codice sorgente del client ed eseguire il packaging (.jar)
- Spostarsi nella directory di destinazione del file .jar
- A seconda della versione della piattaforma Java SE (runtime) usata testare il client come segue:
  - **Java SE 5** → occorre far uso di librerie incluse nel server JBoss (Java EE) ovvero utilizzare il tool wsrunclient.bat (.sh) che si trova nella directory ${JBOSS_HOME}/bin

    wsrunclient.bat –classpath HelloWebServiceClient.jar it.sipe.javaee.ws.example.client.HelloWebServiceClient
  - **Java SE 6** → include le librerie JAX-WS per cui è sufficiente

    java –classpath HelloWebServiceClient.jar it.sipe.javaee.ws.example.client.HelloWebServiceClient

Web Service come Stateless Session Bean

- È possibile esporre il WS come uno Stateless Session Bean
- Creare un nuovo progetto EJB su Eclipse
- Aggiungere uno Stateless Session Bean (ad es. HelloWSBean.java)
- Aggiungere le annotazioni viste in precedenza (@WebService, @WebMethod, etc.)