WebServices

Il Web usato dai programmi

Vincenzo Gervasi

Dipartimento di Informatica

Università di Pisa

Distribuire una computazione

- L'idea di distribuire una computazione è vecchia quasi quanto l'informatica
- Eppure, anche se sembra una buona idea, la sua realizzazione ha sempre comportato molti problemi
- La programmazione di sistemi distribuiti è error-prone e richiede la conoscenza di molti dettagli

RPC

- Remote Procedure Call è un sistema per supportare l'invocazione di procedure remote
- Il cliente invoca una procedura che in realtà non svolge direttamente la computazione, ma traduce la chiamata ad un sistema remoto
- Un insieme di tool sollevano il programmatore del compito di "remotizzare" la chiamata
  - il linguaggio o le librerie usate si occupano di nascondere gli aspetti spiacevoli
  - non possono però nascondere del tutto alcuni effetti collaterali, come i ritardi di rete

Sistemi che usano RPC

- Sun RPC
  - l'originale RPC per UNIX, base di NFS
- CORBA
  - la tecnologia del futuro, da 8 anni a questa parte
- Java RMI
  - Remote Method Invocation
  - Tecnologia Java-only per RPC
- DCOM
  - Distributed Component Object Model, MS-only
- Web Services

Invocare servizi remoti

- L'invocazione di un servizio remoto comprende le seguenti fasi:
  - **Discovery** del servizio
  - **Invocazione** lato client:
    - marshalling degli argomenti in un canale di rete
    - Attesa risposta
    - Unmarshalling del risultato
  - **Invocazione** lato server:
    - Unmarshalling argomenti
    - Invocazione chiamata
    - Marshalling del risultato

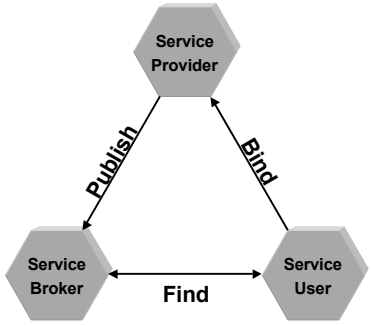
Problemi

- I principali problemi dei sistemi RPC sono:
  - Il modello uniforma chiamate locali e remote
  - Il marshalling/unmarshalling può essere reso automatico solo se si dispone di informazioni relative alla struttura dei tipi
  - Marshalling dei parametri di tipo riferimento
- La disponibilità di reflection in Java e .NET consente di automatizzare il processo di *marshalling/unmarshalling*

Web Services

- L'esperienza CORBA/DCOM/RMI ha mostrato:
  - Problemi di interoperabilità nell'implementazione di protocolli binari
  - Importanza di un modello orientato al servizio
  - Necessità di accedere un servizio da sistemi e piattaforme differenti
- I Web Services introducono un insieme di standard basati su XML per un'**architettura orientata ai servizi (SOA)**

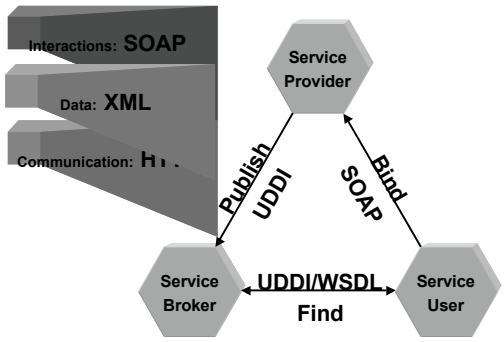
Architettura orientata ai Servizi



Scenario di uso dei Web Services

- Il provider crea il servizio e la sua descrizione WSDL
- Il provider registra il servizio in UDDI
- L'utente trova il servizio cercando il registro UDDI
- L'applicazione si collega al Web Service e invoca le sue operazione via SOAP

Architettura Web Services



Sigle svelate

- **SOAP**: Simple Object Application Protocol
- **WSDL**: Web Services Description Language
- **UDDI**: Universal Description, Discovery and Integration
  - un servizio di catalogo per i web service
  - linguaggio per descrivere un singolo web service
- **WSIL**: Web Services Inspection Language
  - linguaggio per ispezionare un singolo web service
  - formato XML per trasmettere "oggetti" fra applicazioni eterogenee
- **WS-I**: Web Services Interoperability

Protocolli Web Services

Elementi dell'Infrastruttura

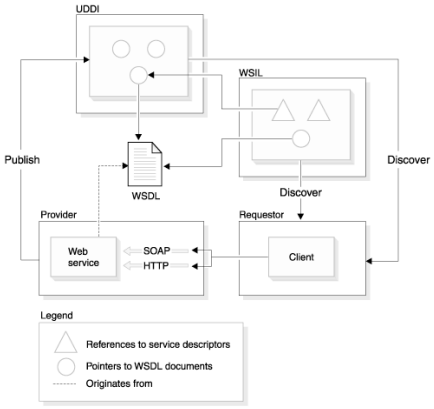
Directories
locazione centralizzata per trovare Web Services offerti da altre organizzazioni (UDDI)

Discovery
trovare il WSDL per un Web Service particolare

Description
definisce che interazioni il WS supporta

Wire Formats
abilita la comunicazione universale (SOAP)

Relazioni fra gli elementi



SOAP

- Protocollo basato su XML e tipicamente trasportato su HTTP che consiste di:
  - Un **envelope** per descrivere il contenuto di un messaggio e istruzioni su come processarlo
  - Un insieme di **regole di codifica** per esprimere istanze di tipi di dato definiti dall'applicazione
  - Una **convenzione sulle comunicazioni (binding style)** per rappresentare chiamate di procedura remote e le loro risposte

SOAP: Envelope

- Una **envelope** descrive il modo in cui il messaggio è codificato e come deve essere processato
  - fra l'altro, contiene l'indirizzo di destinazione finale
- L'envelope contiene uno o più **header ...**
  - **gli header descrivono attributi del messaggio, per esempio la politica di inoltro e le richieste di quality-of-service**
- **... ed esattamente un body**
  - **il body contiene il nome (identificativo) e i parametri del messaggio**

SOAP: Regole di codifica

- Le regole di codifica specificano in che modo le strutture dati complesse (oggetti) devono essere espresse nel corpo del messaggio XML
- In pratica, definiscono una **serializzazione**
- **La serializzazione è (più o meno) quella usata da XSD per la definizione di schemi**
- **Esistono però diversi encoding:**
  - **l'encoding "Literal" (cioè nessun encoding) garantisce la massima portabilità e la compatibilità con WS-I**
  - **l'encoding "SOAP" è richiesto per SOAP1.1 e non è compatibile con WS-I**

SOAP: Comunicazioni

- SOAP definisce due "stili" di comunicazione:
  - Remote Procedure Call (RPC)
    - in questo stile, una comunicazione SOAP è vista come una chiamata di procedura remota
    - il messaggio di richiesta porta il nome di una "procedura" e, opzionalmente, un insieme di parametri (serializzati)
    - segue un messaggio di risposta, contenente zero, uno o più risultati e/o una indicazione di fallimento (Fault)
    - lo stile RPC **non è compatibile con WS-I**
  - Document style
    - comunicazione trattata come la spedizione di un documento
    - stile meno astratto, richiede più lavoro di programmazione a basso livello

Esempio di transazione SOAP

- Supponiamo di aver sviluppato un servizio che fornisce, aggiornato in tempo reale, il tasso di cambio fra Euro e qualunque altra valuta
- Le richieste di servizio porteranno il nome della valuta (per esempio, "USD", "GBP", ecc.)
- Le risposte porteranno il tasso di cambio (per esempio, 1.267, 0.750, ecc.)
- Assumiamo che il cliente abbia già trovato in qualche modo (per esempio, tramite UDDI) la locazione del servizio

Esempio di richiesta SOAP

```
POST /CurrencyServer/CurrencyExchange.asmx HTTP/1.1
Host: theseus
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: http://di.unipi.it/webservices/Euro

<?xml version="1.0" encoding="utf-8"?>
 <soap:Envelope
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:soap=http://schemas.xmlsoap.org/soap/envelope>
 <soap:Body>
  <Euro xmlns=http://di.unipi.it/webservices>
   <currency>string</currency>
  </Euro>
 </soap:Body>
</soap:Envelope>
```

Esempio di risposta SOAP

```
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
   xmlns:xsd=http://www.w3.org/2001/XMLSchema

  xmlns:soap=http://schemas.xmlsoap.org/soap/envelop
  e>
 <soap:Body>
  <EuroResponse
  xmlns=http://di.unipi.it/webservices>
   <EuroResult>double</EuroResult>
  </EuroResponse>
 </soap:Body>
</soap:Envelope>
```

Altri trasporti

- SOAP è un protocollo orientato ai messaggi
- Tipicamente, viene spedito su HTTP
- I messaggi XML possono essere instradati anche su altri protocolli:
  - SMTP
    - invocare un web service via posta elettronica!
  - Message Queue
    - dispatch garantito, anche se pesantemente asincrono
  - Tramite passi multipli
    - il messaggio può anche essere processato ad ogni passo; l'header Actor specifica cosa deve fare ogni attore intermedio
  - Ad hoc

SOAP e Java

- Java dispone di una buona API per manipolare messaggi SOAP
- In particolare:
  - la mappatura fra i tipi base di Java (interi, booleani, ecc.) e tipi SOAP è immediata
  - alcuni tipi non-base in Java (per esempio, Date) hanno un tipo base in SOAP
  - i tipi complessi di SOAP sono:
    - struct, mappate su oggetti Java (eventualmente con getter/setter, e quindi JavaBean)
    - sequenze, mappate su array Java

SOAP e Java

- L'ultimo anello mancante è la mappatura fra nomi XML (per esempio, nomi di struct) e nomi Java (per esempio, nomi delle classi che rappresentano una struct)
- Delle regole di mapping uniformi garantiscono la ragionevolezza di queste corrispondenze
- Usando le API di alto livello, dettagli di questo tipo sono pressoché invisibili!

Approfondimenti su SOAP

- Lo **standard SOAP** (corrente) è definito dal World Wide Web Consortium (W3C)
  - http://www.w3.org/TR/SOAP
- Esistono poi varie implementazioni
  - Apache: http://xml.apache.org/soap
  - Java: JAX-RPC è l'API standard per effettuare le chiamate di procedura remota via XML
    - definita dalla proposta JSR-101
    - dettagli all'URL http://www.jcp.org/en/jsr/detail?id=101
    - anche la documentazione di J2EE è ben fornita

Web Service Description Language

WSDL

- WSDL è uno standard che definisce una codifica (XML) per una *descrizione di un servizio (web)*
- *Una descrizione WSDL contiene:*
  - *il **nome** del servizio offerto e il suo **indirizzo***
  - *quale **protocollo** e **encoding** usare per accedere al servizio*
  - *la **segnatura del servizio, comprendente:***
    - ***i nomi delle operazioni***
    - ***le segnature delle operazioni***
    - ***i tipi utente usati nelle segnature delle operazioni***

Contenuto di un file WSDL

- **Types**: descrizione dei tipi di dato usati (stile XSD)
- **Message**: definizione dello schema dei messaggi scambiati
- **Operation**: descrizione astratta dell'operazione offerta dall'intero web service
- **Port Type**: descrizione astratta delle operazioni offerte da uno o più *end point*
- **Binding**: specifica del protocollo (di solito SOAP) e del formato (di solito literal) per una particolare Port Type
- **Port**: uno specifico *end point, dato dalla combinazione di un binding e di un indirizzo*
- **Service**: un insieme di *end point correlati*

Struttura di WSDL

Types	Data type definitions
Message	Signature of request and reply for each method (≈ IDL)
Port Type	<service, protocol> ⇨ operations
Operation	method ⇨ messages
Binding	Protocol and data-format specification
Service	{ Port ⇨ binding }
Port	Address (≈ URL)

Esempio WSDL

- Currency Exchange Service
- Metodi

```
double rate(String From, String To)
double euro(String Currency)
```

- Service URL

  http://localhost:8080/ExchangeSvc/xchg?WSDL

Esempio WSDL

```
<message name="RateSoapIn">
  <part name="parameters" element="s0:Rate" />
</message>
<message name="RateSoapOut">
  <part name="parameters" element="s0:RateResponse" />
</message>
<message name="EuroSoapIn">
  <part name="parameters" element="s0:Euro" />
</message>
<message name="EuroSoapOut">
  <part name="parameters" element="s0:EuroResponse" />
</message>
```

Come comunicare con il servizio in SOAP

```
<message name="RateHttpGetIn">
  <part name="from" type="s:string" />
  <part name="to" type="s:string" />
</message>
<message name="RateHttpGetOut">
  <part name="Body" element="s0:double" />
</message>
<message name="EuroHttpGetIn">
  <part name="currency" type="s:string" />
</message>
<message name="EuroHttpGetOut">
  <part name="Body" element="s0:double" />
</message>
```

Come comunicare con il servizio tramite HTTP/GET

```
<message name="RateHttpPostIn">
  <part name="from" type="s:string" />
  <part name="to" type="s:string" />
</message>
<message name="RateHttpPostOut">
  <part name="Body" element="s0:double" />
</message>
<message name="EuroHttpPostIn">
  <part name="currency" type="s:string" />
</message>
<message name="EuroHttpPostOut">
  <part name="Body" element="s0:double" />
</message>
...
```

Come comunicare con il servizio tramite HTTP/PUT

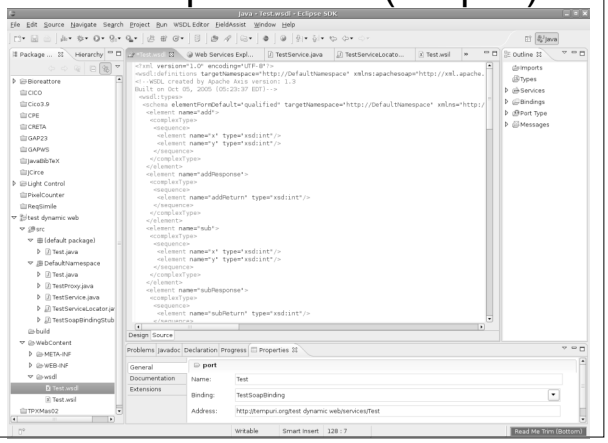
WSDL e Java

- WSDL è essenzialmente un **Interface Definition Language** (IDL) specializzato per i web services
- Grazie a **strumenti** appositi, il matrimonio fra WSDL e Java è ricco di frutti:
  - **bottom-up**: JavaBeans e Enterprise Beans possono essere trasformati in Web Services generando automaticamente il WSDL relativo
  - **top-down**: dato un WSDL, si generano degli "scheletri" di implementazione in Java dei servizi descritti (come bean)
  - **client-side**: dato il WSDL di un servizio remoto, si genera un proxy che consente a un cliente di usarlo come se fosse un oggetto locale

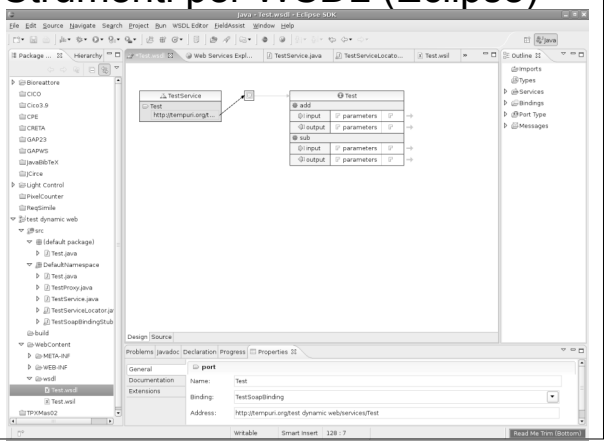
Strumenti per WSDL

- Oltre alle API di Java, un certo numero di strumenti gestiscono WSDL nativamente
  - Apache Axis WSDL → Java compiler (`wsdl2java`)
  - Apache Axis Java → WSDL compiler (`java2wsdl`)
    - usati anche in Jboss
    - `wsdl2java` può anche creare test case di JUnit per il testing
  - Novell (jBroker) Java RMI → WSDL compiler
  - Eclipse, vari compilatori + validatori + display
  - AntEater (http://aft.sourceforge.net) effettua test funzionali di web service basati su messaggi SOAP predefiniti
  - ecc...

Strumenti per WSDL (Eclipse)



Strumenti per WSDL (Eclipse)



Approfondimenti su WSDL

- Il riferimento principale per WSDL è la specifica dello standard W3C:
  - http://www.w3.org/TR/wsdl
- Se si è dotati di buoni strumenti, non serve praticamente mai guardare dentro un .wsdl
  - generazione, validazione, deployment, scaricamento, ispezione... tutti affidati ai tool di sviluppo
  - può però essere utile in fase di debug o per placare la curiosità!

UDDI

- UDDI definisce un modo per **pubblicare e ricercare informazioni sui servizi web**
- **Vengono specificati:**
  - **il protocollo (basato su SOAP) con cui i clienti comunicano con i registri UDDI**
  - **un particolare insieme di registri pubblici globali (replicati)**
    - **simile per certi versi al sistema dei DNS**
- **Le API di pubblicazione, interrogazione, replicazione sono a loro volta web services!**

Registri UDDI

- I registri pubblici sono gestiti da grosse organizzazioni, tipicamente broker o fornitori di web services
- Esistono poi anche registri privati, identici a quelli pubblici eccetto per il fatto che il loro uso è limitato all'interno di una organizzazione
  - molti application server/development environment, come WebSphere, includono un registro UDDI a scopo di test

Registri UDDI

- Un po' di storia
  - Inizialmente, UDDI pensava di implementare un piccolo insieme di “super-registri” contenenti tutti i web services esistenti...
  - ... poi ci si è resi conto che era più efficiente lasciare che ciascun fornitore di web services pubblicizzasse i propri servizi individualmente...
  - ... aspettando un “UDDI-Google” che faccia da broker
- Oggi, il maggiore uso di UDDI è fra partner commerciali
  - I registri UDDI pubblici di IBM, Microsoft, SAP sono stati chiusi a inizio 2006!

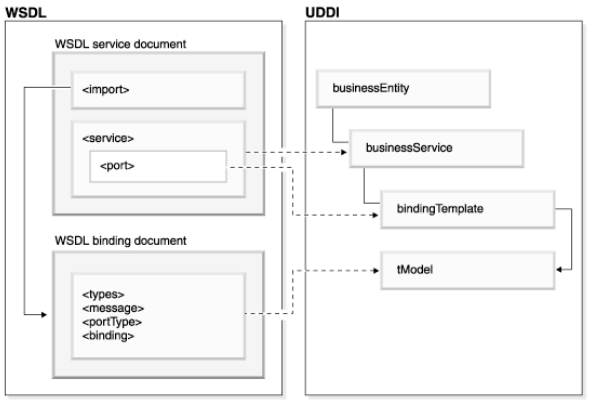
Il contenuto di un registro UDDI

- Concettualmente, un registro UDDI contiene tre raccolte di informazioni:
  - Pagine bianche: nome e contatti di un fornitore di servizi, con una descrizione (in linguaggio naturale) dei servizi forniti
  - Pagine gialle: classificazione standard dei fornitori (come nelle pagine gialle nostrane), secondo standard internazionali
  - Pagine verdi: informazioni tecniche (URL) sui meccanismi di discovery disponibili per ogni fornitore
- Complessivamente, si tratta di servizi di tipo commerciale

UDDI e WSDL

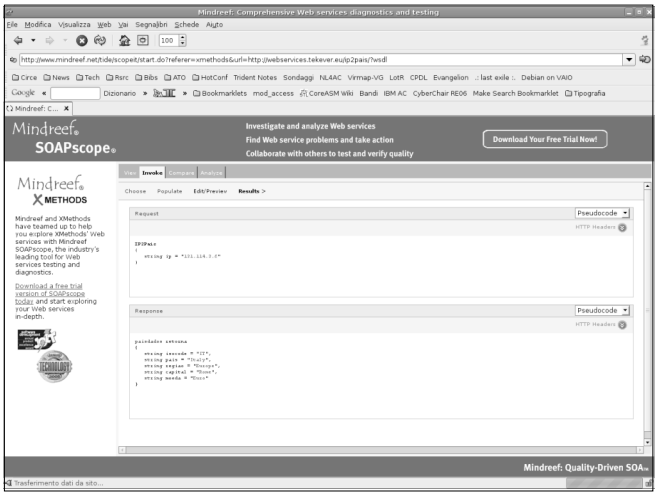
- Tecnicamente, UDDI contiene quattro tipi di dato:
  - **businessEntity** descrive un fornitore di servizi
  - **businessService** descrive un singolo servizio
  - **bindingTemplate** descrive come invocare un servizio
  - **tModel** descrive i tipi di dato usati
- Per parte sua, WSDL contiene due tipi di documenti
  - il **Binding Document** descrive i tipi usati nell'interfaccia del servizio e i protocolli supportati (servizio astratto)
  - il **Service Document** descrive la locazione del servizio, e contiene le informazioni su come accedervi (servizio concreto)
- I documenti WSDL possono essere unici o spezzati su più file (con opportuni link fra l'uno e l'altro)

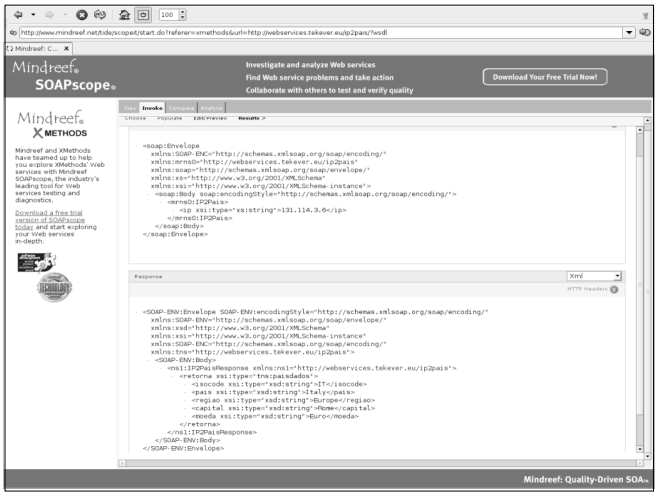
UDDI e WSDL



Approfondimenti su UDDI

- La “casa” di UDDI è http://www.uddi.org
- Un esempio di catalogo di servizi (per le “pagine gialle” di UDDI) è quello dell'ONU all'URL http://www.unspsc.org/
  - Per esempio:
    - i fumetti hanno codice 55101505; le raccolte di fumetti 49101607
    - i DVD sono il prodotto 43202003; i film su DVD sono il 55111514
- Alcuni esempi di web services pubblicati sono all'URL http://www.xmethods.com/
  - realizzati con una varietà di tecnologie
  - consente anche di invocare i servizi e vedere i messaggi scambiati in vari formati





WSIL

- **WSIL** è il Web Services Inspection Language
- Analogo ma alternativo e complementare a UDDI
- Un diverso modello di ricerca:
  - con UDDI, si cerca in un elenco di servizi di vari fornitori uno che faccia al caso nostro
  - con WSIL, si chiede a un fornitore quali servizi fornisce
- Come al solito, tutto è basato su XML
  - WSIL = un formato XML per elencare servizi
  - ogni servizio può apparire più volte, una per ogni modo di accedere *a una sua descrizione*
    - ***sono descrizioni i file .wsdl ma anche le entries UDDI!***

WSIL

- WSIL specifica che il documento “master” di descrizione deve chiamarsi **`inspection.wsil`** ed essere nella root della URL che fornisce i servizi
- **`inspection.wsil`** può poi linkare documenti (sempre .wsil) memorizzati altrove
- Le descrizione puntate dai .wsil possono essere di qualunque tipo
  - oltre che .wsdl, anche pagine HTML con descrizioni in linguaggio naturale!

Approfondimenti su WSIL

- WSIL non è un soggetto particolarmente affascinante...
- La definizione formale dello standard è all'URL www.ibm.com/developerworks/webservices/library/ws-wsilspec.html
- Non ci dilunghiamo oltre...

WS-I

- A differenza di tutte le altre sigle, che indicavano formati, standard e specifiche, WS-I è una **organizzazione**
- Scopo di WS-I è promuovere l'interoperabilità nel mondo dei web services
  - WS-I definisce dei profili di interoperabilità a cui le varie implementazioni devono adeguarsi
  - Principalmente rivolta a WSDL e SOAP/HTTP
  - Laddove questi standard permettono varie opzioni, WS-I ne incoraggia una specifica
- Dettagli a **http://www.ws-i.org/**

Gli elementi chiave

- XML – il nuovo alfabeto universale
- XMLns – uso dei namespace per fare ordine
- XSD – definizioni di tipi di dato
- RPC – chiamate di procedura remota
- SOAP – scambio messaggi con XML e HTTP
- WSDL – descrizione astratta e concreta dei servizi
- UDDI e WSIL – indicizzazione dei servizi
- Java – un linguaggio di implementazione dei servizi

Alcune note: Reflection

- L'uso di reflection consente la generazione automatica di *stub* e *proxy*
- Il sistema dei tipi XSD è ricco e consente un mapping fedele dei tipi del linguaggio
- Dei tool (wscompile e altri) leggono la definizione di classi ed interfacce e generano i *wrapper* lato client e lato server
- La parte di codifica "clericale" è quasi scomparsa grazie a tool più efficienti

Alcune note: passaggio per valore, singleton, factory

- SOAP non prevede alcun supporto per il passaggio di parametri per riferimento
- Questo a volte richiede un'implementazione "a mano" di questo pattern
- L'invocazione SOAP non prevede il passaggio di "this"
- È necessario codificare esplicitamente il factory pattern

Creare un web service in Java

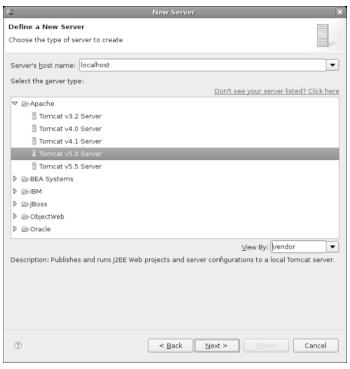
- Due strade principali (e una ferrata di montagna):
  - Usare un ambiente integrato di sviluppo
    - Eclipse, Netbeans, WebSphere, ecc.
  - Usare i tool a riga di comando del SDK J2EE
    - wsdl2java, java2wsdl, javac, jar, ant, ecc.
  - Scrivere a mano i file di corredo
    - .wsdl, .wsil, .ear, .war, ecc.
- Noi considereremo solo la prima

Creare un web service con Eclipse

- La versione più aggiornata dei tool di sviluppo su Eclipse è data dalla release *Callisto*
  - *Eclipse 3.2 + WebTools + altra roba buona*
- *Occorre poi installare un application server*
  - *Tomcat 5, Tomcat 5.5, JBoss, WebSphere...*
- *Può essere complicato crearsi un ambiente di sviluppo funzionante*
  - *diritti sulle directory di deployment, versioni compatibili dei vari took, JDK diversi fra macchina di sviluppo e macchina di test... ouch!*

Creare un web service con Eclipse

- Una volta impostato l'ambiente, i passi sono (relativamente) semplici:
  - creare un Server all'interno del workbench, con New... / Other / Server / Server
  - fornire i dati richiesti
    - nome
    - dir di installazione
    - JRE da usare

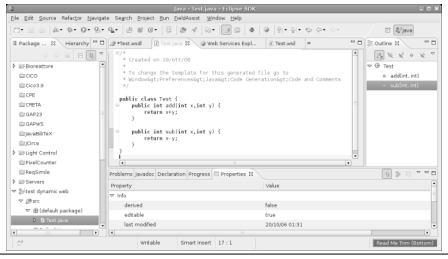


Creare un web service con Eclipse

- Poi, per ogni progetto:
  - creare un progetto di tipo Web (dinamico)
  - dare un nome
  - scegliere come target uno dei server definiti in precedenza
  - opzionalmente, specificare i dettagli sulla configurazione, sul deployment, ecc.

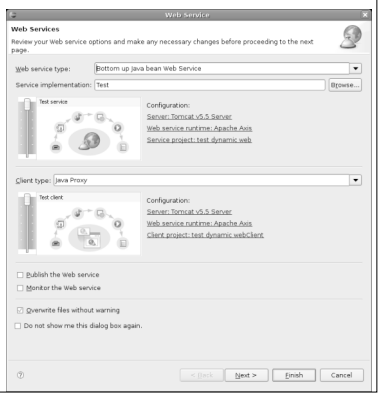
Creare un web service con Eclipse

- Creare o importare una o più classi Java
  - magari JavaBean o EnterpriseBean
- Alcuni metodi di queste classi diventeranno i nostri web services (con la stessa segnatura)



Creare un web service con Eclipse

- Selezionare la/e classe/i i cui metodi devono divenire web services
- Dal menu contestuale, selezionare *Web Services / Create Web services*
- *Si apre il wizard di creazione:*

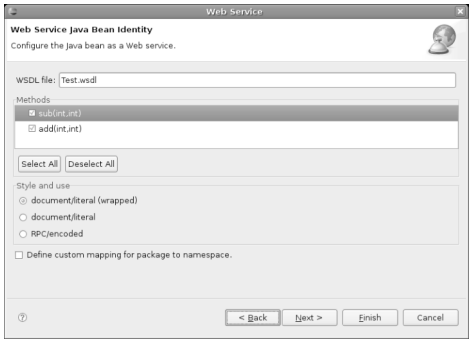


Creare un web service con Eclipse

- Sul wizard, selezionare:
  - la modalità di creazione
    - bottom-up: da Java a WSDL (nostro esempio)
    - top-down: da WSDL a Java
  - il tipo di processo desiderato per il server e per l'eventuale client:
    - server: develop, assemble, deploy, install, start, test
    - client: no client, develop, assemble, deploy, install, start test
  - la configurazione e le eventuali altre opzioni di generazione
- Premere Next>

Creare un web service con Eclipse

- Scegliere i metodi da trasformare in web services:



Creare un web service con Eclipse

- Finito!
- Il wizard genera
  - il WSDL, ed eventualmente il WSIL
  - un'interfaccia Java che contiene i metodi esportati (e che è implementata dalla nostra classe iniziale),
  - eventualmente, un Proxy per il client
  - altre classi di supporto (SoapBinding & co.)
  - tutto l'occorrente per il deploy (WEB-INF/...)
- Eventualmente, carica già il servizio sull'application server, pronto per il test

Creare un web service con Eclipse

- Se si è scelto di creare il client, viene generata una classe proxy che implementa la stessa interfaccia del web service
- Il corpo dei metodi si limita a inoltrare le chiamate tramite SOAP al web service remoto
- Per il cliente, il proxy locale è indistinguibile da una normale classe Java, però
  - il tempo di esecuzione può essere **molto** lungo!
  - si possono verificare errori (di rete) inattesi

Demo
(o esercitazione?)

Creare un web service in Java

- Il supporto alla creazione dei web service negli ambienti di sviluppo integrati è già molto buono, e sta migliorando ancora
- È anche possibile usare Java “puro” per la creazione di web services senza troppa fatica
- Grazie alle **annotazioni introdotte in Java 5**
  - In particolare, l'annotazione @WebService premessa alla dichiarazione di una classe la marca – ovviamente – come un web service
  - Le annotazioni @WebMethod, @WebParam, @WebResult, @OneWay, @HandlerChain consentono di fare fine tuning

Creare un web service in Java

- Esempio:

```
package server;
import javax.jws.WebService;

@WebService
public class HelloImpl {
  /**
   * @param name
   * @return Say hello to the person.
   */
  public String sayHello(String name) {
    return "Hello, " + name + "!";
  }
```

Creare un web service in Java

- Il passo successivo consiste nell'eseguire il tool **`apt`** (Annotation Processing Tool) sul nostro sorgente
- Vengono generati numerosi altri file:
  - HelloServiceImpl.wsdl
  - schema1.xsd
  - classes/server/HelloImpl.class
  - classes/server/jaxrpc/SayHello.class
  - classes/server/jaxrpc/SayHelloResponse.class
  - classes/server/jaxrpc/SayHello.java
  - classes/server/jaxrpc/SayHelloResponse.java
- Tutto il necessario per il web service!

Creare un web service in Java

- Vediamo il contenuto di **HelloServiceImpl.wsdl**:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
   name="HelloImplService"
   targetNamespace="http://server/jaxrpc"
   xmlns:tns="http://server/jaxrpc"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
<types>
    <xsd:schema>
      <xsd:import namespace="http://server/jaxrpc"
        schemaLocation="schema1.xsd"/>
    </xsd:schema>
</types>
<message name="sayHello">
   <part name="parameters" element="tns:sayHello"/>
</message>
<message name="sayHelloResponse">
   <part name="result" element="tns:sayHelloResponse"/>
</message>
<portType name="HelloImpl">
   <operation name="sayHello">
     <input message="tns:sayHello"/>
     <output message="tns:sayHelloResponse"/>
   </operation>
</portType>
<binding name="HelloImplBinding" type="tns:HelloImpl">
   <soap:binding
     transport="http://schemas.xmlsoap.org/soap/http"
      style="document"/>
   <operation name="sayHello">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
   </operation>
</binding>
<service name="HelloImplService">
   <port name="HelloImplPort" binding="tns:HelloImplBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
</service>
</definitions>
```

Creare un web service in Java

- Rimane da fare (manualmente o con dei task ant) il packaging e il deploy del web service
- Occorre creare un .war (come al solito)
  - Al suo interno, un deployment descriptor chiamato **`jaxrpc-ri.xml`** fornisce informazioni sul web service appena definito
    - il nome è standard, definito da JAX-RPC
  - Un altro deployment descriptor, il consueto **`web.xml`**, fornisce il mapping fra il web service e una URL sul server, oltre a tutti gli altri dettagli tipici delle servlet (timeout, descrizione, persistenza, ...)

Conclusioni

- I Web services sono un tassello fondamentale per l'integrazione di sistemi
- Garantiscono interoperabilità fra sistemi diversi, su S.O. diversi, scritti in linguaggi diversi, su architetture diverse
- Dopo un periodo iniziale "difficile", ora Java supporta bene i Web services
- Idem per i più comuni ambienti di sviluppo, che ormai "chiudono il cerchio" fra DB, logica applicativa e web con numerosi strumenti
- ... ma il diavolo è nei dettagli! Occorre *veramente studiare un po' di materiale!*