Introduzione ai Web Services

Marco Giunti

FSE 2009

FSE'09 - Argomenti

1. Cosa sono i Web Services
2. XML: Il linguaggio dei Web Services
3. Tecnologia dei Web Services
4. Come costruire un sistema SOA con Apache Axis
5. Advanced Topics: messaging e composizione di WS

Definizione di Web Service (W3C)

- **"Un servizio web è un'applicazione software identificata da una URI, la cui interfaccia può essere definita, descritta, e scoperta usando XML. Un web service interagisce con altri componenti software usando messaggistica XML scambiata su protocolli internet"**
- sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete
- offre un'interfaccia software tramite la quale altri sistemi possono interagire con il Web Service stesso.
- L'interfaccia descrive le operazioni alle quali si accede tramite appositi messaggi trasportati tramite il protocollo HTTP e formattati secondo lo standard XML.

Web Services

- Entità computazionali
  - Interoperabili
  - Autonome e indipendenti
  - Componibili
  - Scopribili
- Interoperabilita’
  - Logica del servizio Indipendente da piattaforma o linguaggio di programmazione
  - WS in C# eseguito su .NET può essere composto con WS in JAVA eseguito su JAVA EE

A cosa servono i Web Services?

- Garantire a dispositivi di natura differente pieno accesso a tipologie di dati eterogenei.
- Quando due entità si mettono d'accordo per scambiarsi una serie di informazioni e per astrarre il procedimento, si affidano ad un sistema in grado di garantire una manutenibilità ed una durata della soluzione il più lunga possibile.

Struttura Internet: comunicazione client server

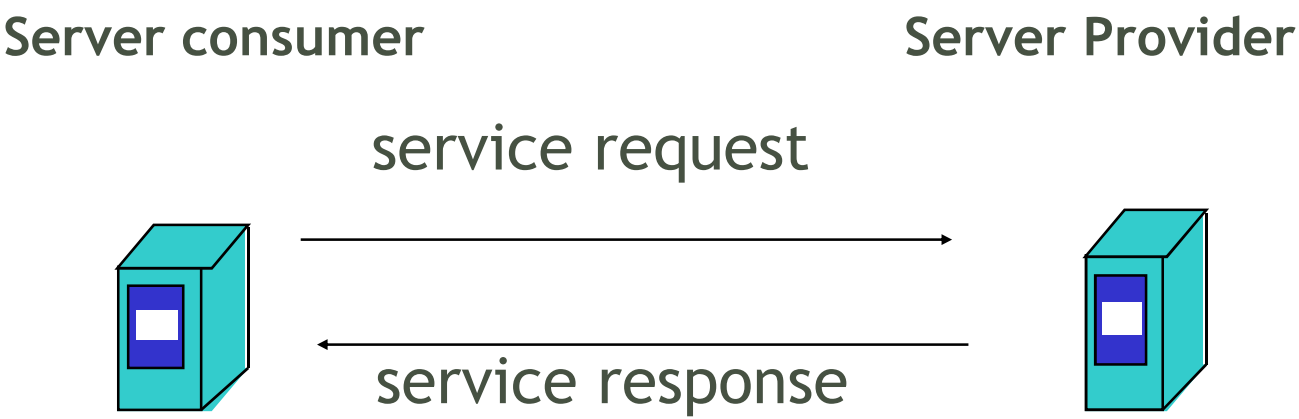
Componenti in Internet

- Client effettua la richiesta (anche chiamato **consumer**)
- Server fornisce una risposta (anche chiamato **provider**).

Schema generale dei WS

- Principalmente un web service espone all'esterno una serie di funzionalità, attraverso un **listener,** anche chiamato server.
- Un listener è un particolare programma che si mette in ascolto delle richieste che provengono da eventuali client ed ovviamente, cerca di rispondere nel modo migliore.

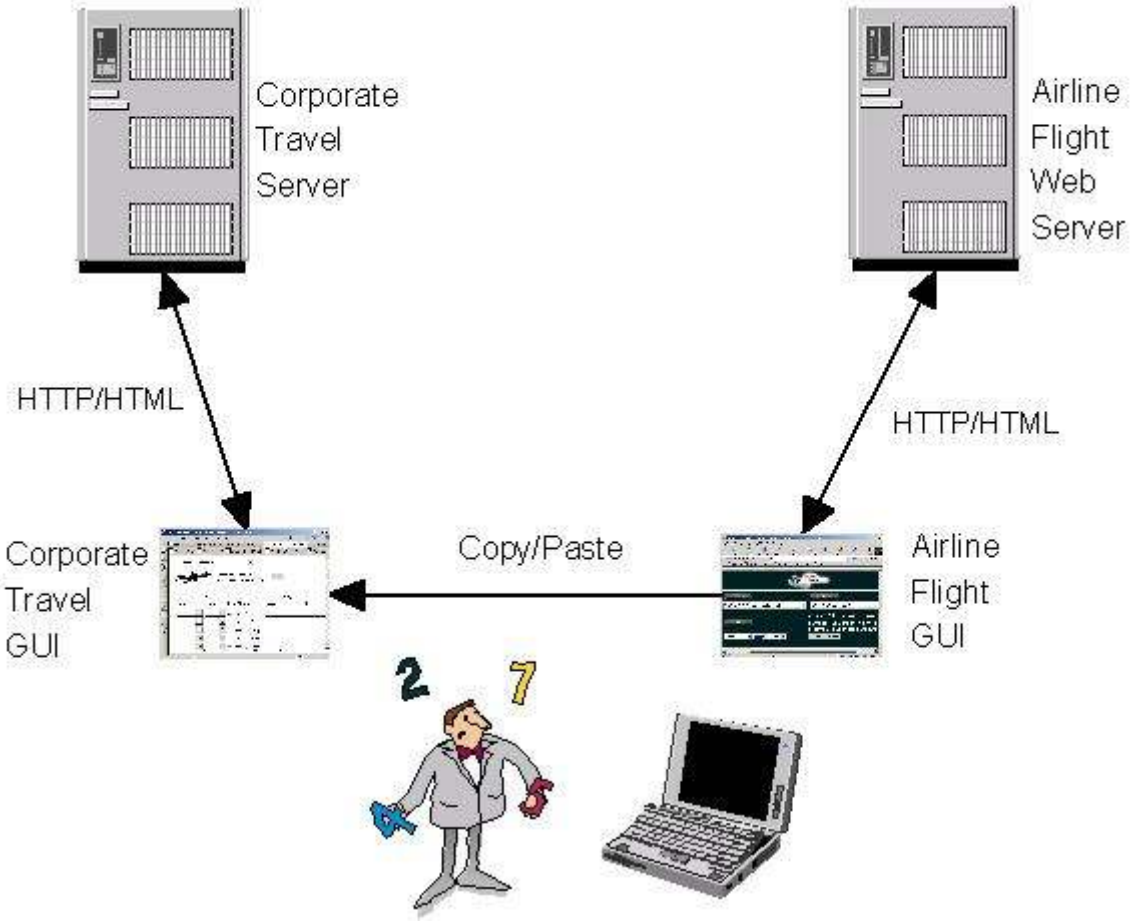
Struttura WS



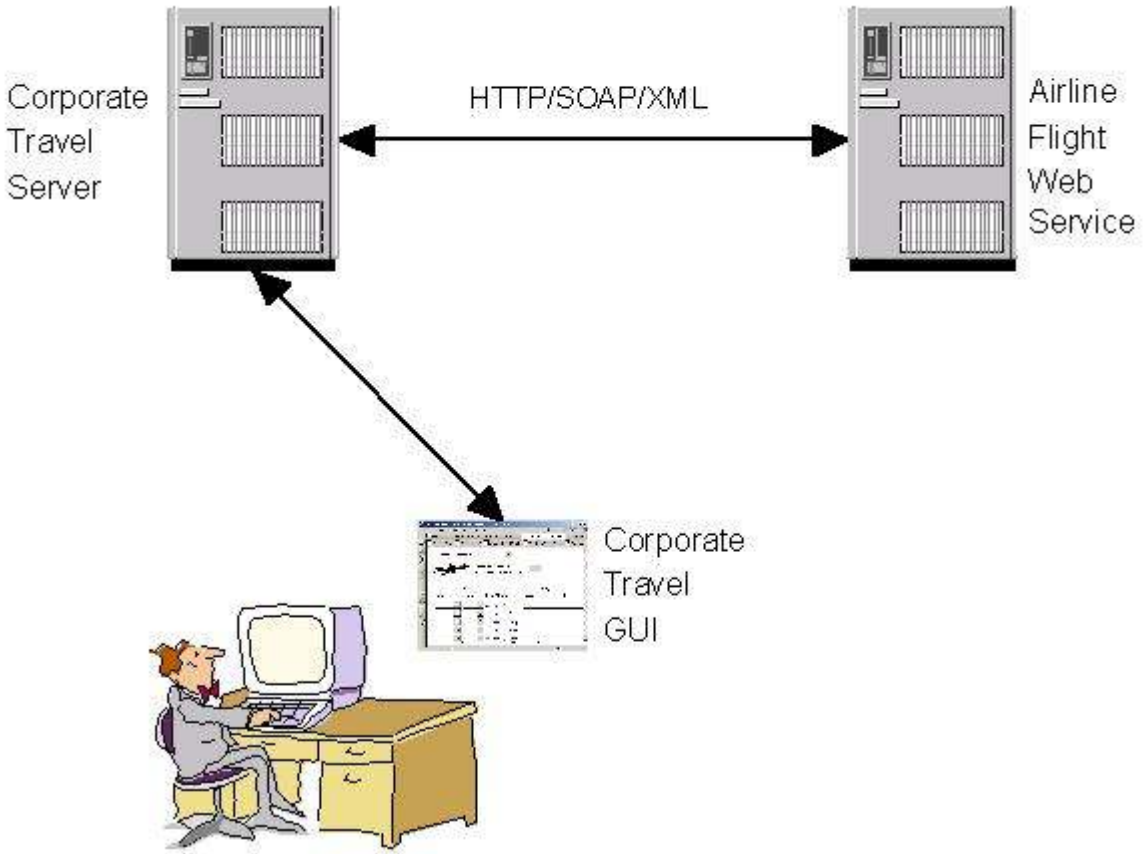
Componenti dei WS

A differenza di ciò che avviene con la normale navigazione attraverso un browser, nei Web Services gli attori non sono utente e server, ma **due server**.

Agenzia viaggio – Schema senza Web Service



Integrazione migliorata – Schema con Web Service



Caratteristiche dei Web Services

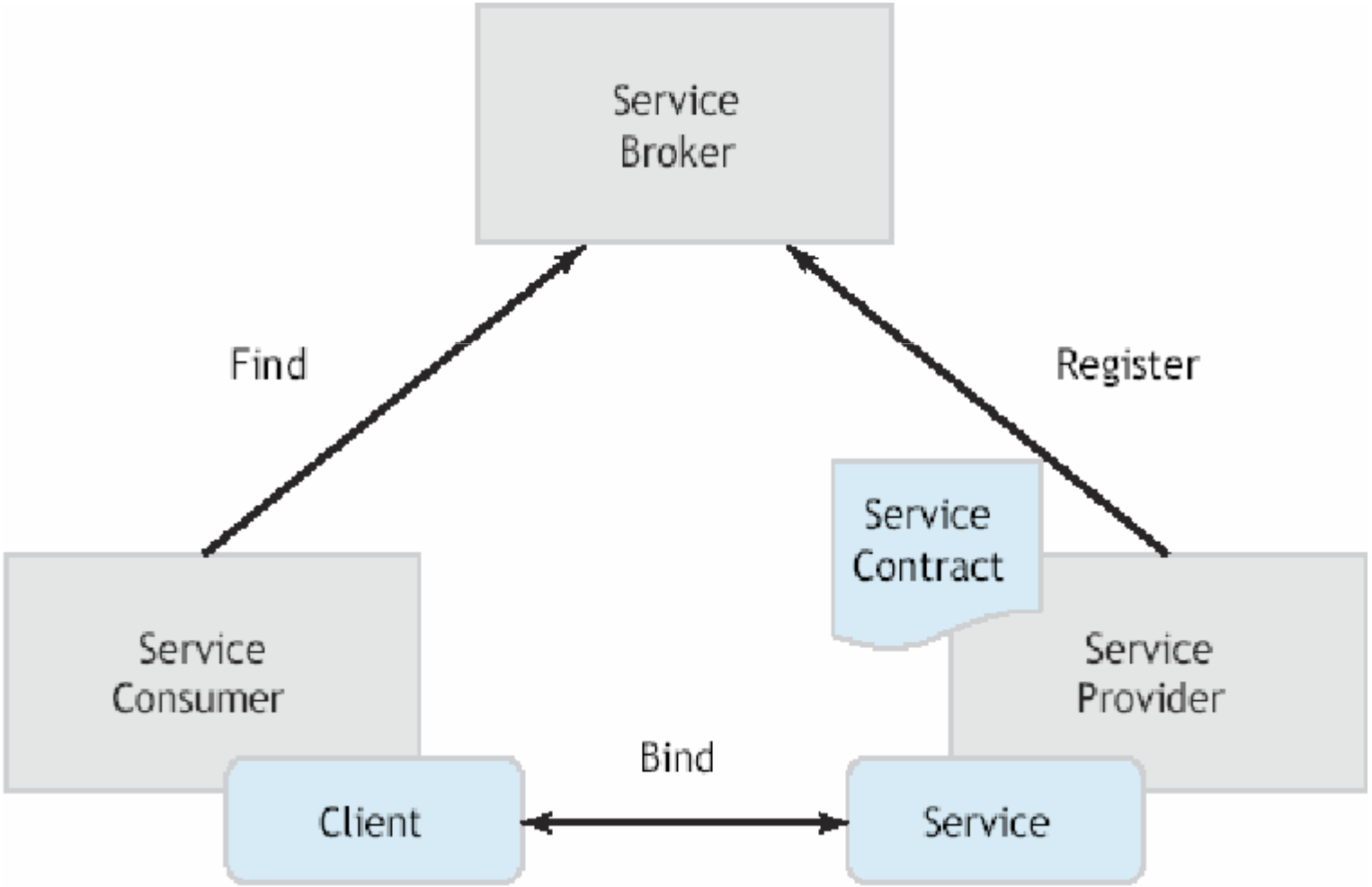
- I WS sono applicabili in qualsiasi tipo di ambiente Web:Internet,intranet,extranet.
- ogni tipo di applicazione può essere offerta come WS.
- rappresentano la convergenza tra le architetture orientate ai servizi(SOA) e il web.
- sono accessibili da ogni nodo della rete.
- possono essere combinati, in modo da ottenere servizi complessi, semplicemente facendo interagire un insieme di programmi che forniscono, ognuno, un servizio semplice.

Service Oriented Architecture

I WS si basano sulla Service Oriented Architecture (SOA). I tre componenti principali sono:

1. Service Provider ,rende disponibile il servizio e pubblica il contratto che ne descrive l'interfaccia(tramite il broker).
2. Service Requestor o Consumer,effettua le queries al service broker e questo cerca il servizio compatibile.
3. Service Registry o Broker,da info al consumer su quale servizio utilizzare e dove trovarlo.

Web Services roles



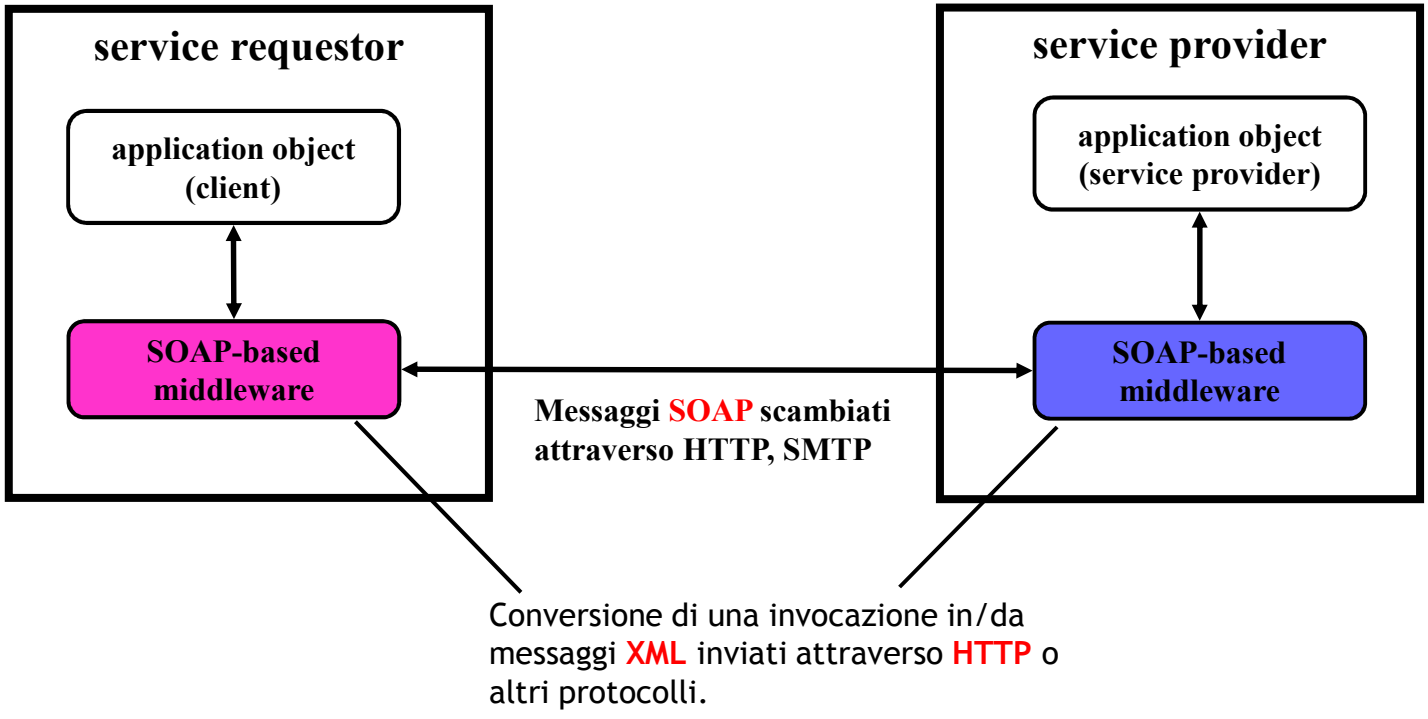
Comunicazione dei Web Services

- Il requester invoca un servizio reso disponibile dal provider
  - Scambio messaggi
  - Interazione
- Web services comunicano usando standard
  - XML
  - SOAP
  - HTTP

Invocazione

- Sintassi per tutte le specifiche: **XML**
- Un meccanismo che permette componenti remote di interagire
- Rendere l'invocazione di un servizio in un messaggio XML
  - Scambiare messaggi
  - Attivare il servizio alla ricezione di un messaggio
- Un formato di dati comune per lo scambio dei messaggi (**SOAP**)
- La possibilita' di inviare I messaggi attraverso differenti protocolli di "trasporto"
  - TCP/IP
  - HTTP
  - SMTP

Web Service: Processo di invocazione



FSE'09 - Argomenti

1. Introduzione
2. XML: Il linguaggio dei Web Services
3. Tecnologia dei Web Services
4. Come costruire un sistema SOA con Apache Axis
5. Advanced Topics: messaging e composizione di WS

XML

- Linguaggio di markup text based
- Come in HTML si identificano i dati attraverso tag
- I dati contenuti tra un tag e la sua chiusura individuano un element
- I tag possono contenere attributi
- Pro di XML:
  - Plain text
  - Composizionale
  - Gerarchico

XML: esempio di un documento

```
<message to="you@yourAddress.com" from="me@myAddress.com" >

   <subject>XML Is Really Cool</subject>

   <text> How many ways is XML cool? Let me count the ways...
   </text>

</message>
```

XML: element vs attribute

- **Element**
  - **Possono essere ripetuti**
  - **L'ordine conta!**
  - **Possono contenere altri element**
- **Attribute**
  - **Determinano informazioni aggiuntive di un element**
  - **Per ogni element non ci possono essere piu' attributi con lo stesso nome**
  - **Possono rappresentare solo valori base come Stringhe**
- Scelta obbligata
  - Il dato puo' contenere sottostrutture (element)
  - Il dato puo' contenere linee multiple (element)
  - Il dato e' una piccola e semplice stringa (attribute)
  - Il dato puo' assumere solo un numero relativamente piccolo di valori o una scelta fissa (attribute)
- Scelte stilistiche

Realizzare un semplice documento XML (2)

- Aggiungere un commento

```
<?xml version=“1.0” encoding=“utf-8” ?>
<!- Un semplice documento per descrivere una presentazione -->
```

Realizzare un semplice documento XML (3)

- Definire l'elemento root. Ogni documento XML ne contiene solamente uno.

```
<?xml version="1.0" encoding="utf-8" ?>
<!- Un semplice documento per descrivere una presentazione -->

<slideshow>
</slideshow>
```

Realizzare un semplice documento XML (4)

- Aggiungere attributi ad un elemento.

```
<?xml version="1.0" encoding="utf-8" ?>
<!- Un semplice documento per descrivere una presentazione -->

<slideshow
   title="Esempio di slide"
   date="29/01/2009"
   author="Marco Giunti">

</slideshow>
```

Realizzare un semplice documento XML (5)

- Aggiungere elementi annidati

```
<?xml version="1.0" encoding="utf-8" ?>
<!- Un semplice documento per descrivere una presentazione -->

<slideshow
   …
   <slide type="intro">
    <title>Contenuti della presentazione</title>
   </slide>
   <slide type="content">
    <title>Prima pagina</title>
    <item>Primo argomento <b>poco</b> interessante</item>
    <item/>
    <item>Primo argomento <b>molto</b> interessante</item>
   </slide>

</slideshow>
```

Realizzare un semplice documento XML (6)

- Attenzione, chiudere sempre I tag!
- Esempio di codice errato

Validazione di Documenti XML

- Disponibili vari SCHEMA per validare documenti XML
- Concretamente sono type system per XML
- DTD, XML-SCHEMA (W3C)

Esempio di validazione con XML Schema

- XML Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="country" type="Country"/>
<xs:complexType name="Country"> <xs:sequence> <xs:element
    name="name" type="xs:string"/>
<xs:element name="population" type="xs:decimal"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

- Documento **XML validato dalla specifica**

```
<country xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="country.xsd">
<name>France</name>
<population>59.7</population>
 </country>
```

DTD: Definire la sintassi per documenti XML

- Dal punto di vista di un DTD un documento e' realizzato attraverso questi blocchi
  - Elements : blocchi principali (slide, item, etc)
  - Attributes : Informazioni extra
  - Entities : variabili usate per definire testo comune (&lt = <)
  - PCDATA : testo semplice contenuto tra I tag
  - CDATA : testo semplice
    - CDATA rappresenta testo che non sara' parsato da un parser. I tag contenuti in questo non sono da trattare com marcatori

DTD: Element

```
<!- Elemento vuoto -->
   <!ELEMENT br EMPTY>
<!- Elemento con solo PCDATA -->
   <!ELEMENT from (#PCDATA)>
<!- Elemento che puo' contenere qualsiasi cosa -->
   <!ELEMENT note ANY>
<!- Elemento che contiente child in sequenza -->
   <!ELEMENT note (to,from,heading,body)>
   <!ELEMENT to      (#PCDATA)>
   <!ELEMENT from    (#PCDATA)>
   <!ELEMENT heading (#PCDATA)>
   <!ELEMENT body    (#PCDATA)>
<!- Elemento contenente una/opzionale/qualsiasi/almeno una occorrenza -->
   <!ELEMENT note (to)>
   <!ELEMENT note (to?)>
   <!ELEMENT note (to*)>
   <!ELEMENT note (to+)>
<!- OR -->
   <!ELEMENT note (to|body)>
```

DTD: Attribute

- Per definire un attributo:
  - `<!ATTLIST element-name attribute-name attribute-type default-value>`
- Dove attribute-type puo' essere
  - CDATA : stringa
  - (en1|en2|..) : il valore deve appatenere alla lista
  - ID : il valore e' un identificativo univoco
  - IDREF : il valore e' l'ID di un altro elemento
  - IDREFS : il valore e' una lista di ID di altri elementi
- Dove default-value puo' essere
  - value : il valore di default
  - #REQUIRED : l'attributo e' obbligatorio
  - #IMPLIED : opzionale
  - #FIXED value : il valore deve essere sempre "value"

DTD: Entity

- Per definire un nuovo entity:
  - `<!ENTITY entity-name “entity-value”>`
  - `<!ENTITY entity-name SYSTEM "URI/URL">`

XML e DTD: Un esempio (tvschedule.dtd)

- Programmi televisivi
  - Uno o piu' canali
  - Ogni canale ha un banner testuale
  - Per ogni canale venegono specificati i programmi di piu' giorni
- Per ogni giorno ed ogni canale
  - Data
  - L'elenco dei programmi se questi sono disponibili, Holiday se la trasmissione e' sospesa per festivita'
- Ogni programma e' descritto da
  - Codice VTR opzionale
  - Titolo
  - Rating e lingua opzionali
  - Una descrizione opzionale

Esempio 2: modellazione dei dati

- Catalogo Prodotti
  - Nome
  - Categoria (libri, cd, dvd)
  - Codice
  - Peso e volume opzionali (se c'e' il peso ci deve essere anche il volume)
  - Piu' costi, differenti per utente
  - Delle note opzionali

Parsing di documenti XML

- SAX parser basato su eventi
  - XML Text nodes
  - XML Element nodes
  - XML Processing Instructions
  - XML Comments
  - Evento lanciato quando inizio/fine feature incontrata
- DOM parser basato su Document Object Model (W3C)
  - Il documento e' visto come un albero
  - Ogni nodo contiene una delle componenti del documento XML
  - Manipolazione nodi possibile (create, change,remove)

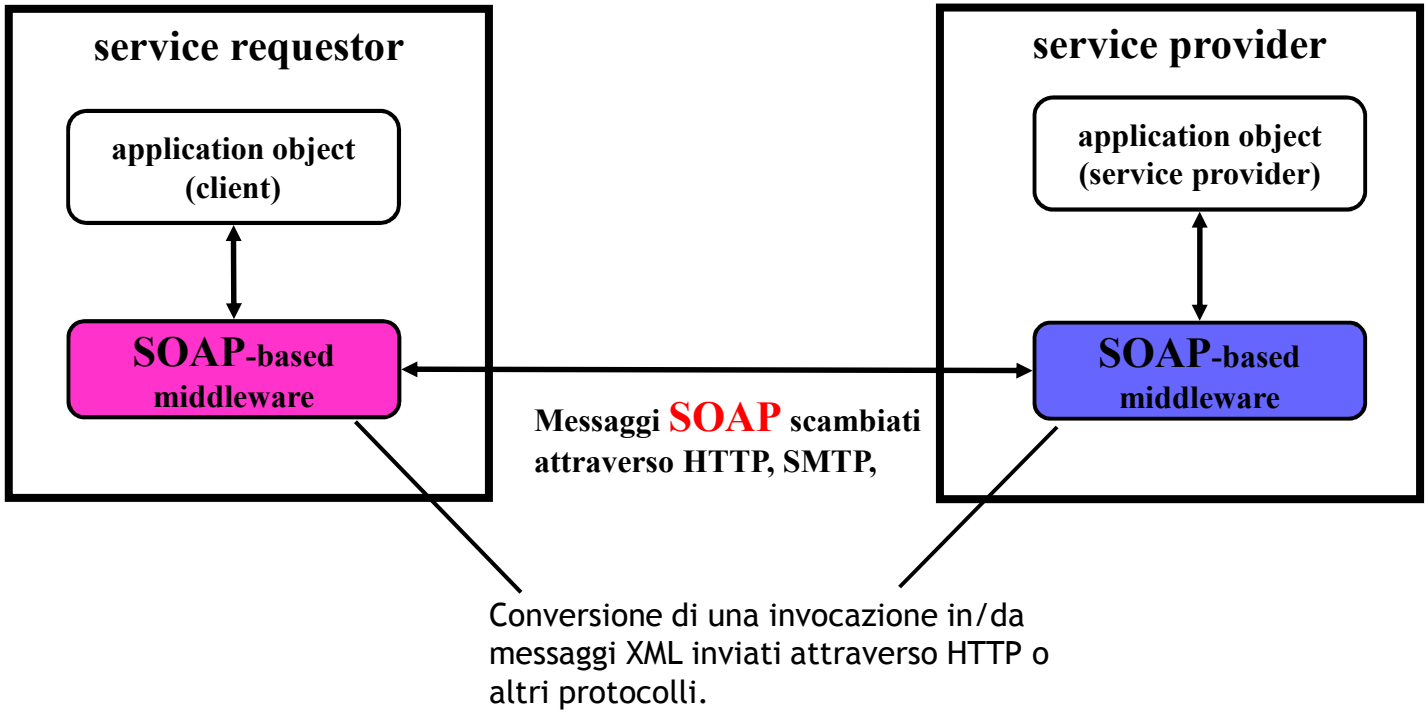
FSE'09 - Argomenti

1. Introduzione
2. XML: Il linguaggio dei Web Services
3. Tecnologia dei Web Services
4. Come costruire un sistema SOA con Apache Axis
5. Advanced Topics: messaging e composizione di WS

SOAP

Comunicare in SOAP

- **Web Services: processo di invocazione**



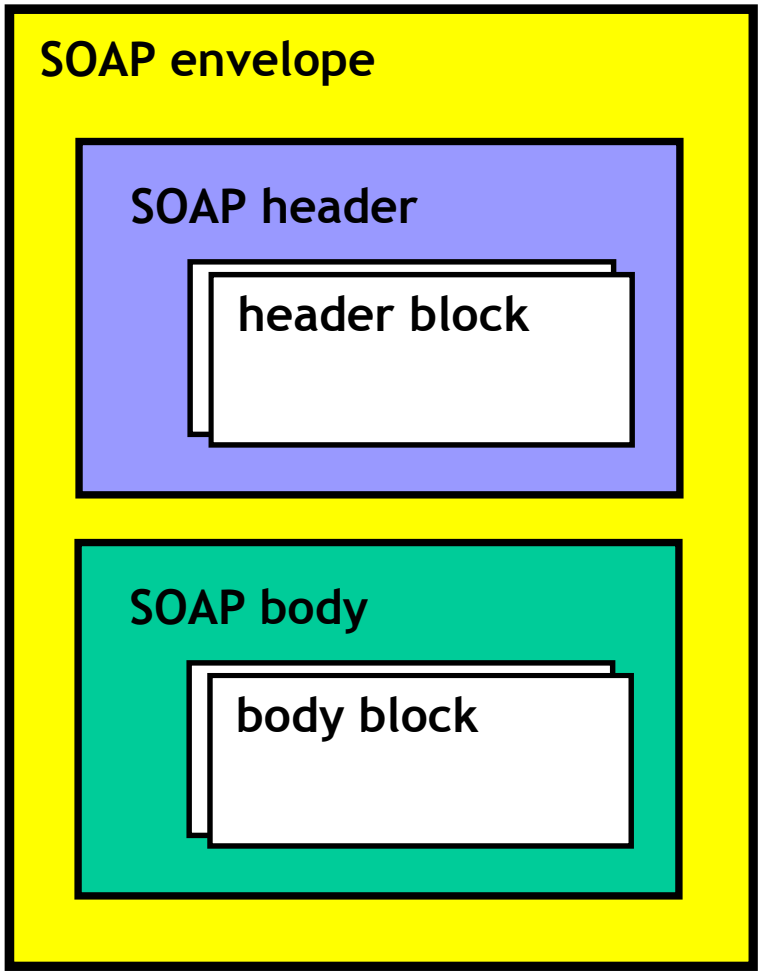
SOAP

- Protocollo leggero per lo scambio di messaggi tra componenti software
- Framework estensibile e decentralizzato che può operare sopra varie pile di protocolli per reti di computer
  - HTTP (W3C Standard), TCP/IP, SMTP
- SOAP basato su XML
  - Body trasporta informazioni (XML-Schema)
  - Header contiene meta-informazioni (routing,sicurezza)

Specifica SOAP

- Formato dei messaggi per la comunicazione one-way, codifica informazioni in documenti XML
- Convenzioni per la realizzazione di interazioni
  - Client invoca procedura remota attraverso messaggio SOAP
  - Servizio rsiponde al client con messaggio SOAP
- Un insieme di regole seguite da ogni entita'
  - Gli elementi da leggere e comprendere in un messaggio
  - Le azioni da eseguire se non si puo' comprendere il contenuto
- Una descrizione di come i messaggi SOAP possono essere trasportati attraverso HTTP e SMTP

SOAP: Struttura di un messaggio

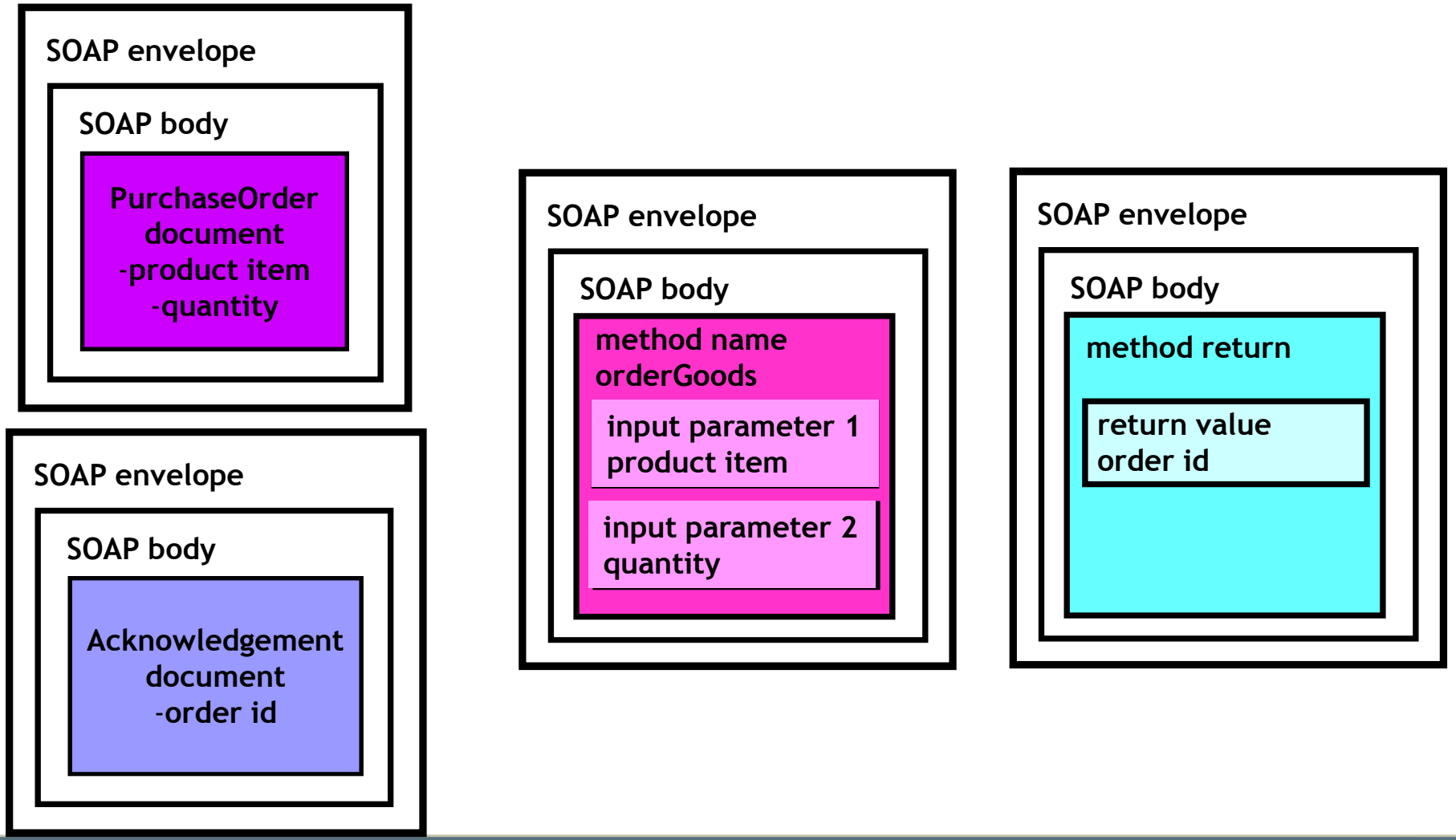


- SOAP assume che per ogni messaggio ci sia un **sender**, un **ultimate receiver** e vari intermediari
- Header: Informazioni aggiuntive per l'elaborazione da parte degli intermediari
- Body: Informazione base della trasmissione

SOAP: Tipi di interazione

- Vari aspetti influenzano la struttura di un messaggio SOAP
  - Document-style
    - I componenti si accordano sulla struttura dei messaggi scambiati
    - I messaggi di request e response sono documento
      - Il messaggio di request puo' contenere gli articoli e le quantita' ed il suo header informazioni per il fornitore per identificare il cliente
      - Il messaggio di response puo' contenere l'order-id per notificare l'accettazione
    - Tipicamente asincrono
  - RPC-style
    - Il messaggio di request contiene il nome della procedura da chiamare ed I suoi parametri
    - Il messaggio di response I parametri di output
    - L'uso della signature dei metodi in SOAP viene nascosto dal middleware
  - Interazioni sincrone ed asincrone non influiscono sul messaggio

SOAP: Tipi di interazione (2)



SOAP: Elaborare un messaggio

- La divisione tra Header e Body indica che I messaggi possono essere elaborati da differenti nodi in un path
  - Ogni blocco nell'Header SOAP indica il ruolo per cui e' destinato
- Roles
  - None : Il blocco non deve essere elaborato da nessuno
  - UltimateReceiver : il blocco e' destinato solo al destinatario del messaggio
  - Next : Ogni nodo puo' processare il messaggio
- Flag MustUnderstand
  - Il nodo che gioca il ruolo indicato dal blocco deve essere in grado di elaborarlo, altrimenti deve emettere un errore e fermare la trasmissione.

SOAP: Esempio di un messaggio

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope" >
 <env:Header>
  <t:transactionID
          xmlns:t="http://intermediary.example.com/procurement"
          env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
          env:mustUnderstand="true" >
          57539
  </t:transactionID>
 </env:Header>

 <env:Body>
  <m:orderGoods
     env:encodingStyle="http://www.w3.org/2002/06/soap-encoding"
     xmlns:m="http://example.com/procurement">
   <m:productItem>
         <name>ACME Softener</name>
   </m:productItem>
   <m:quantity>
     35
   </m:quantity>
  </m:orderGoods>
 </env:Body>

</env:Envelope>
```

envelope

header

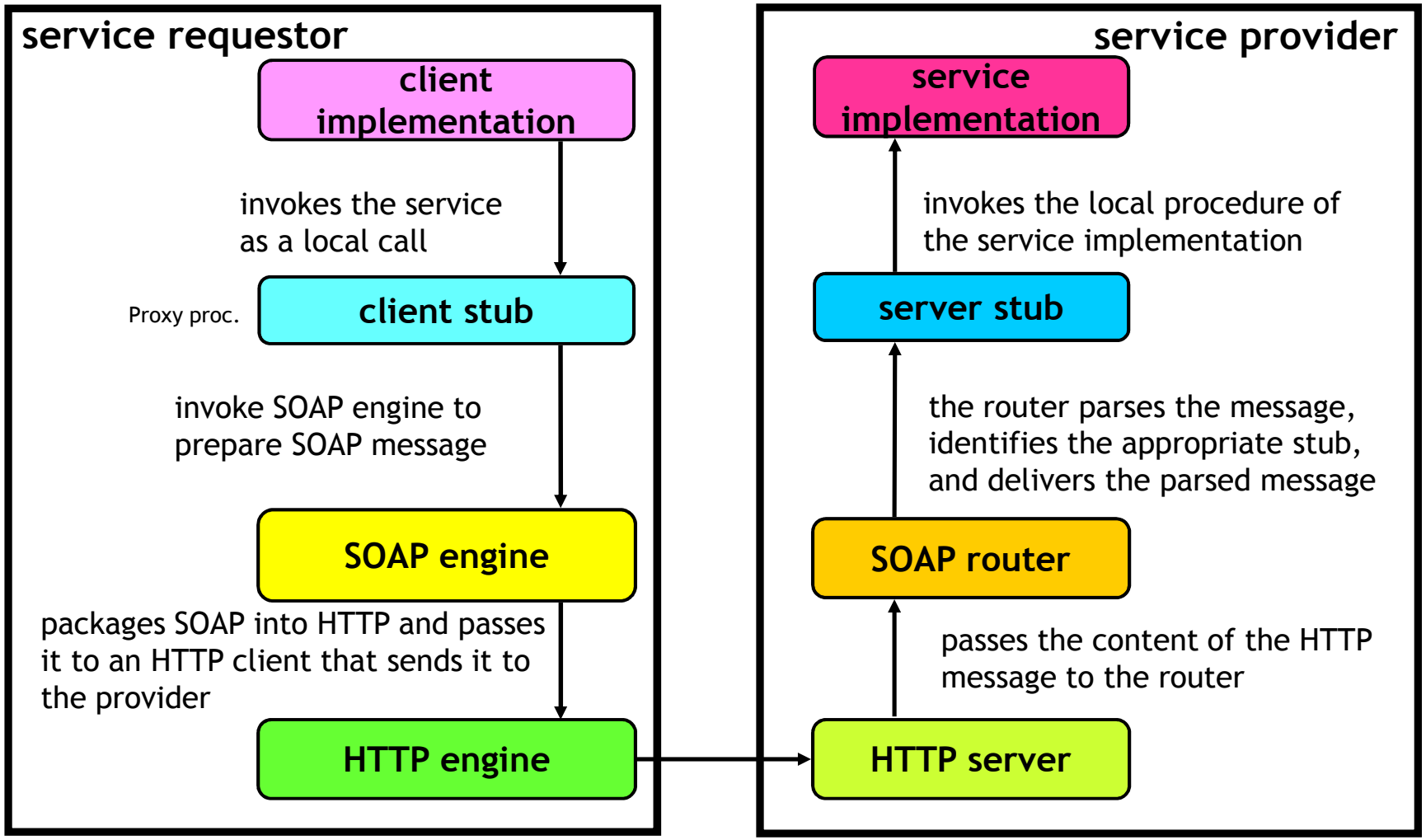
blocks

body

SOAP: Collegare SOAP al “protocollo di trasporto”

- Addressing
  - L’indirizzo dell’UltimateReceiver non e’ una parte del messaggio SOAP
  - Spesso viene identificato attraverso il protocollo di trasporto
    - HTTP: URL della risorsa
    - SMTP: l’indirizzo “to” nella mail
- Routing
  - Non c’e’ ancora un meccanismo per descrivere il path SOAP come parte del messaggio SOAP
  - Il Path dei massaggi attualmente segue quello del protocollo di trasporto
- Alcuni standard emergenti forniscono soluzioni alternative.

SOAP: Una semplice implementazione



SOAP: Asincronia

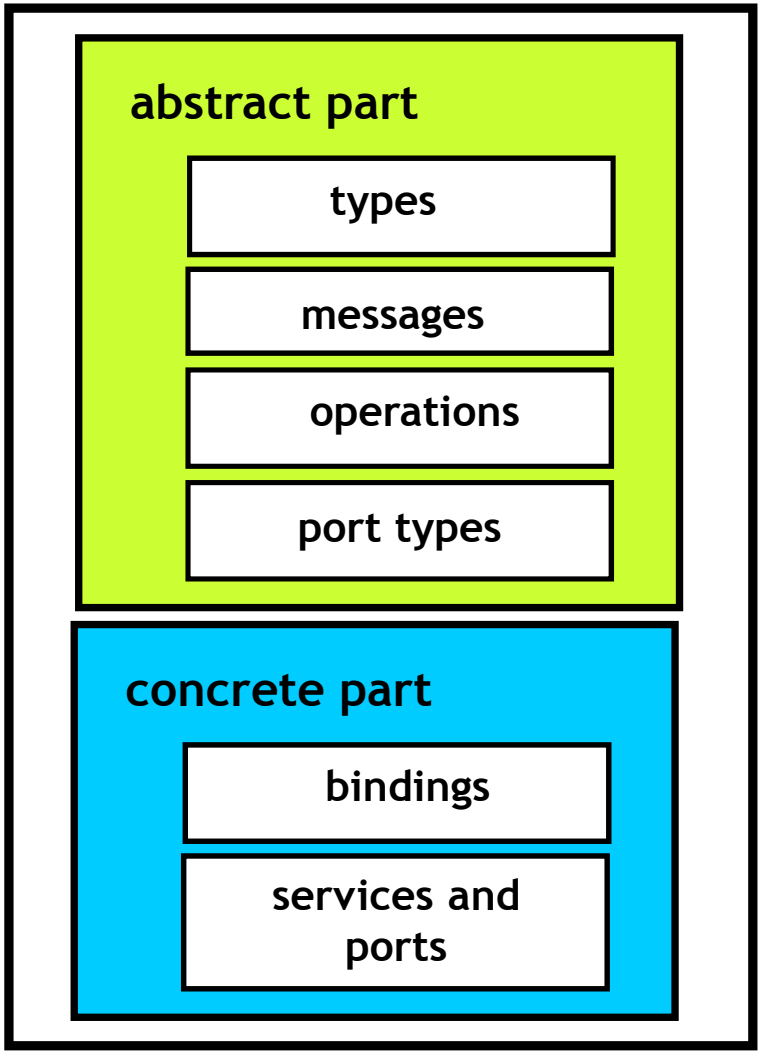
- La comunicazione in stile RPC e' possibile ma richiede una forte integrazione tra le componenti
  - Inaccettabile in ambienti industriali
  - Molte comunicazioni B2B sono Document-style, spesso eseguite in bodalita' batch
- Differenti modi per usare SOAP in modo asincrono
  - Usare un protocollo di trasporto asincrono come SMTP
  - Usare thread separati
    - Uno continua con la logica principale
    - Gli altri effettuano le chiamate SOAP ed attendono i risultati
    - Stessa tecnica spesso usata nell'RPC asincrono

WSDL

Web Service Description Language (WSDL)

- Un linguaggio per la definizione di interfacce
- Interfaccia specificata attraverso i metodi supportati dal WS
- Ogni metodo prende un messaggio in input e puo' restituire un messaggio come output
- Nelle interazioni in stile RPC i messaggi rappresentano i parametri di input e di output di una procedura
- Strumenti per generare stubs per rendere l'invocazione di un WS trasparente

WSDL: Specifica WSDL



WSDL: Struttura

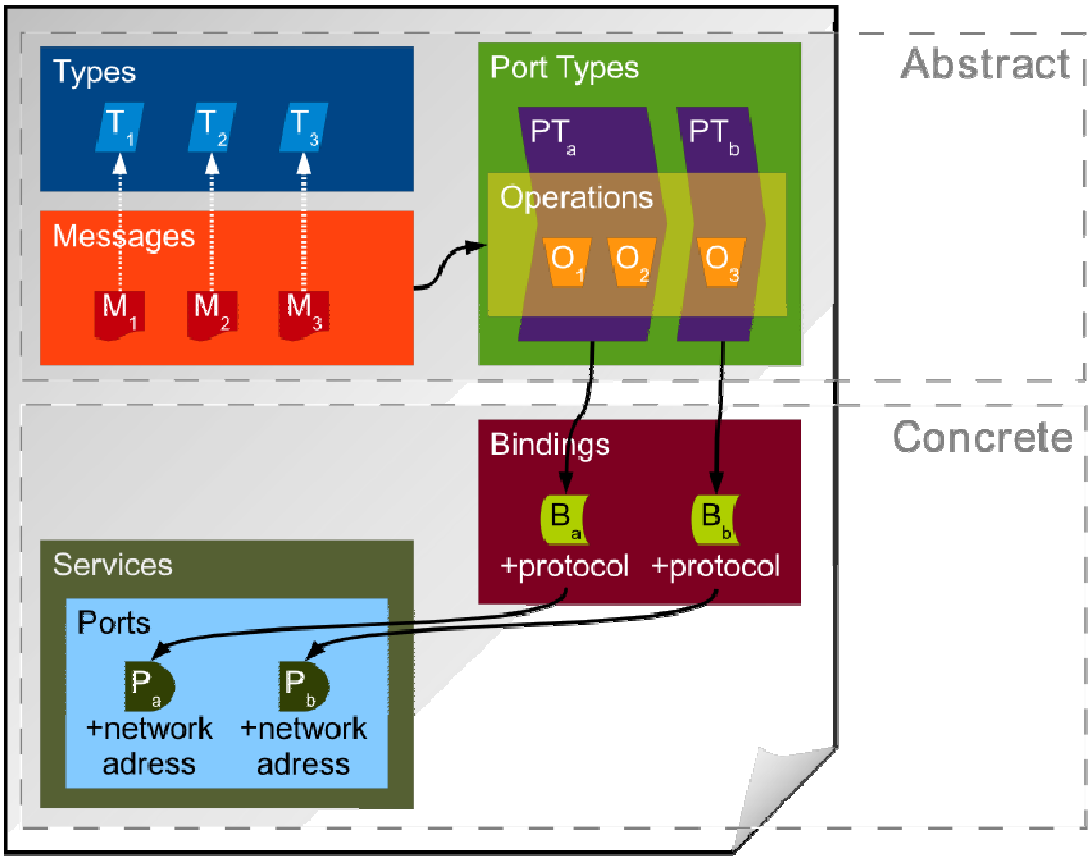
- Separazione delle interfacce, del collegamento con il protocollo di trasporto, delle informazioni di addressing
  - Servizi differenti possono implementare la stessa interfaccia, ma con differenti indirizzi o protocolli di trasporto
- Parte astratta
  - Port types:
    - Analogo alle interfacce degli **IDL**
    - Collezione logica di operazioni correlate
  - Operation:
    - Definisce un semplice scambio di messaggi
  - Messages:
    - Unita' di comunicazione nel WS (I dati sono comunicati in una sola trasmissione)
    - Ona o piu' **part**

WSDL: Struttura (2)

- Parte concreta
  - Una instanza reale del servizio
  - **Interface Binding** per elementi di tipo **port**
    - Tipo di interazione (RPC o Document-style)
    - Collegamento con il protocollo (Protocol binding)
  - Ports
    - Combina l'interface binding con indirizzi di rete (URI) dove il port type e' implementato.
  - Services:
    - Raggruppamento logico di porte
      - Porte correlate allo stesso indirizzo
      - Differenti collegamenti dello stesso Port Type a differenti indirizzi
- Type System basato su XML Schemas

W3C Web Service Description Language

- Definizione astratta di WS come endpoints operanti su messaggi



Costruire un Interfaccia astratta

1. Definire messaggi input/ouput definiti in **parts**
   - Nome
   - Tipo (XML Schema)
2. Definire operazioni
   - One-way,request-response,solocit-response,notification
3. Raggruppare operazioni in port types

WSDL: Esempio

```
<?xml version="1.0"?>
<definitions name="Procurement"
   targetNamespace="http://example.com/procurement/definitions"
   xmlns:tns="http://example.com/procurement/definitions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://schemas.xmlsoap.org/wsdl/" >
   <message name="OrderMsg">
      <part name="productName" type="xs:string"/>
      <part name="quantity" type="xs:integer"/>
   </message>
   <portType name="procurementPortType">
      <operation  name="orderGoods">
         <input message = "OrderMsg"/>
      </operation>
   </portType>
   <binding name="ProcurementSoapBinding" type="tns:procurementPortType">
      <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="orderGoods">
         <soap:operation soapAction="http://example.com/orderGoods"/>
         <input>
             <soap:body use="literal"/>
         </input>
         <output>
             <soap:body use="literal"/>
         </output>
      </operation>
   </binding>
   <service name="ProcurementService">
      <port name="ProcurementPort" binding="tns:ProcurementSoapBinding">
         <soap:address location="http://example.com/procurement"/>
      </port>
   </service>
</definitions>
```

abstract part

messages

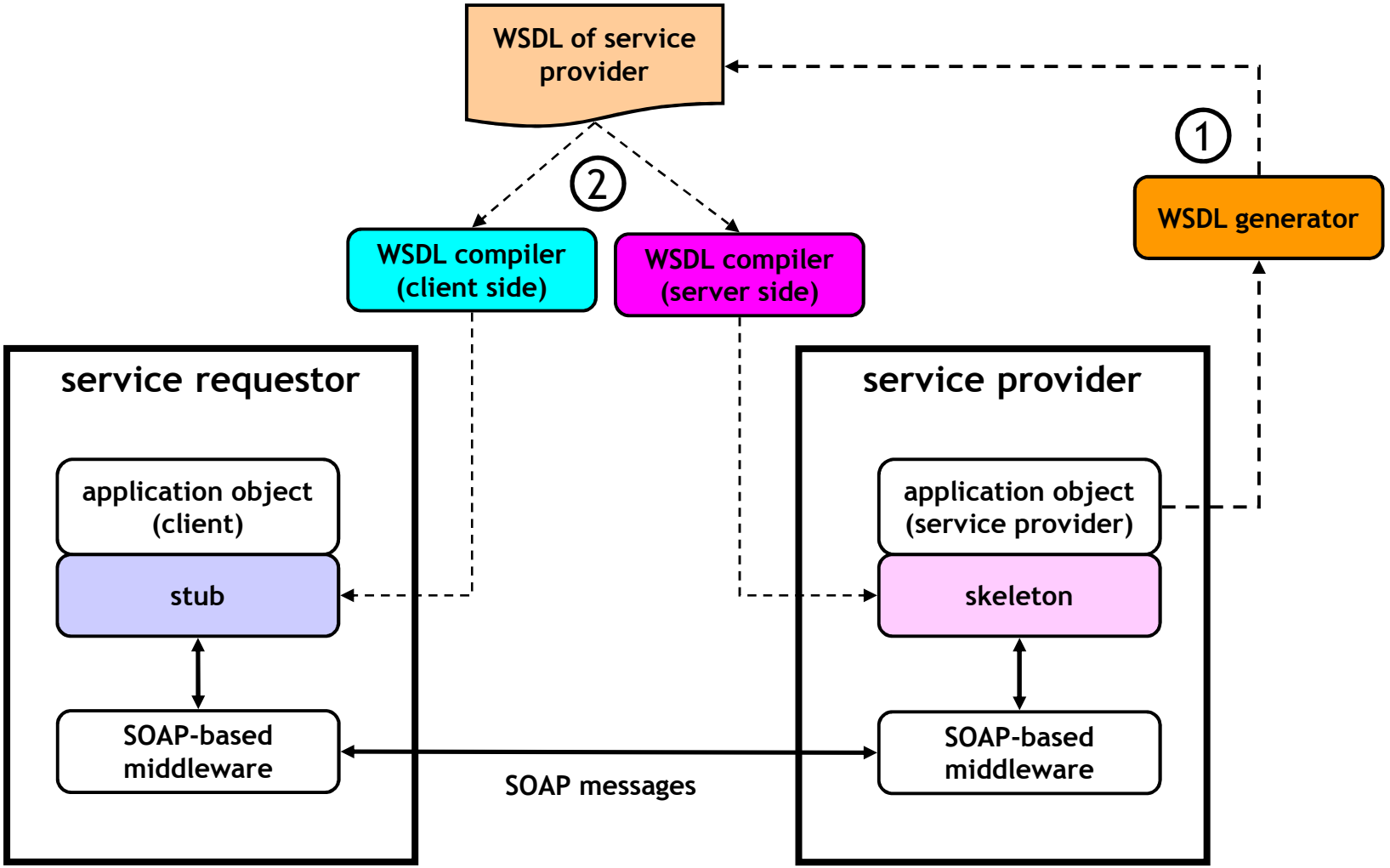
operation and port type

binding

concrete part

port and service

Generazione WSDL da API

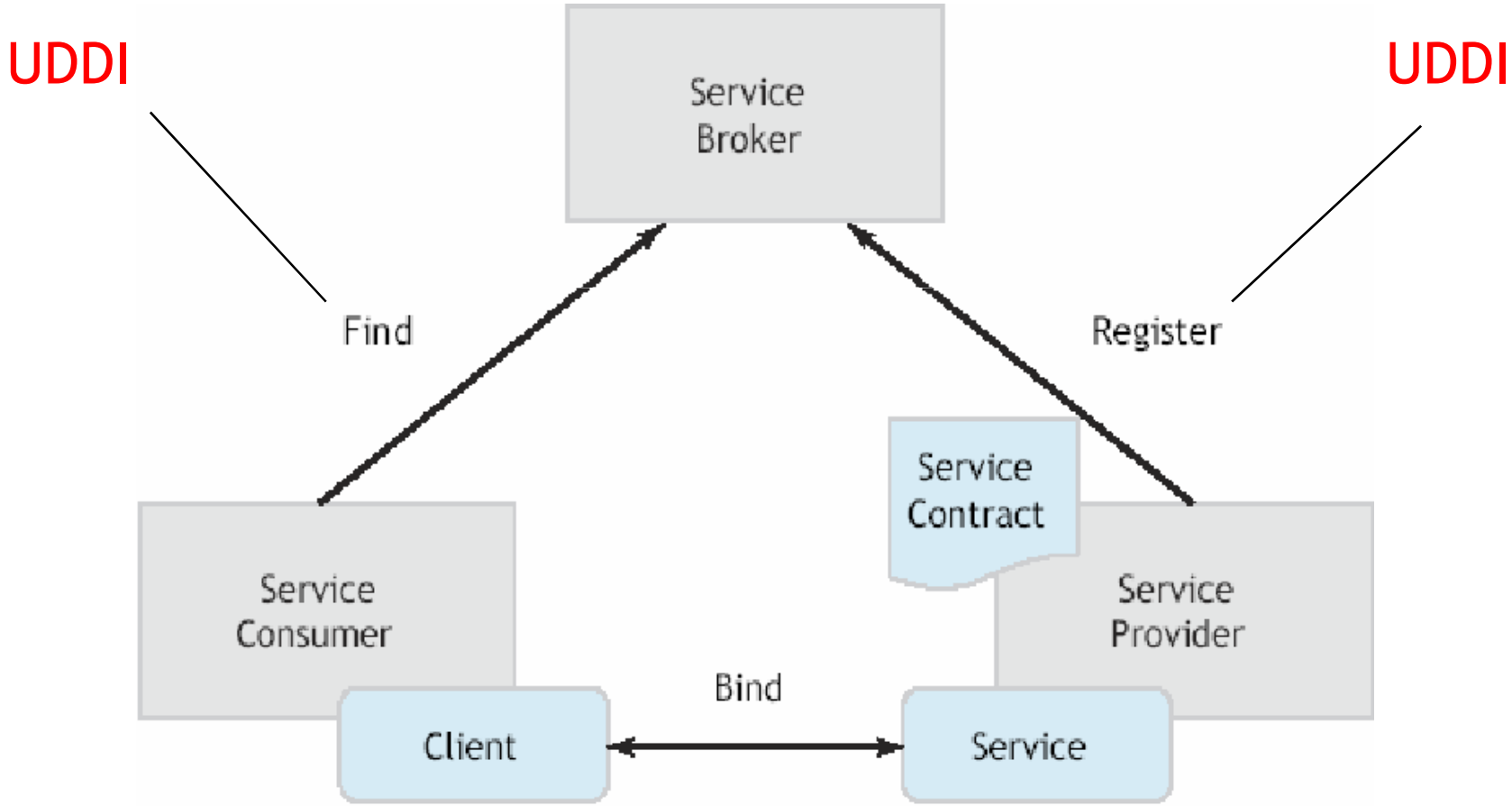


WSDL: Conclusioni

- La descrizione di un servizio non presume alcuna particolare forma di comunicazione per lo scambio
- Riusabilita' della parte astratta
- Collegamento tra WSDL e SOAP
  - L'Interface Binding ha tutte le informazioni necessarie per coustruire automaticamente I messaggi SOAP
- WSDL e' l'input a compilatori di stub ed altri tool
  - Generano gli stub ed altre informazioni per semplificare lo sviluppo
- La descrizione WSDL di un servizio puo' essere realizzata automaticamente

UDDI

Universal Description, Discovery and Integration



Scoperta di Web Services – UDDI

La funzione di scoperta si realizza tramite UDDI che offre un meccanismo standard per registrare e ricercare i WS:

- **Tipo di servizio:** si definisce il servizio e si assegna a questo un identificatore unico chiamato tModel. Un tModel punta a una specifica che definisce una risorsa. Oltre al servizio si definisce un'interfaccia astratta.

Pagine bianche

- **Service providers** registra i businesses e tutti i servizi che offrono. Tramite il costrutto Template fornisce info sul binding e sul punto di accesso.

Pagine verdi

- **Categorizzazione:** si usa una varietà di categorie per classificare le entità in base alla localizzazione geografica, al codice prodotto ecc.

Pagine gialle

UDDI

- **Ricerca.** I service consumer possono ricercare il servizio effettuando queries al registro Uddi, questa ricerca può avvenire tramite tipo di servizio o service providers
- **Binding :** il collegamento tra client e servizio può avvenire sia in fase di compilazione che di runtime.

  Statico: La ricerca di un service viene effettuata una volta sola e se ne memorizza il risultato. Quindi la localizzazione dei servizi si conosce prima di iniziare l'esecuzione del programma e si tratta di indirizzi assoluti.

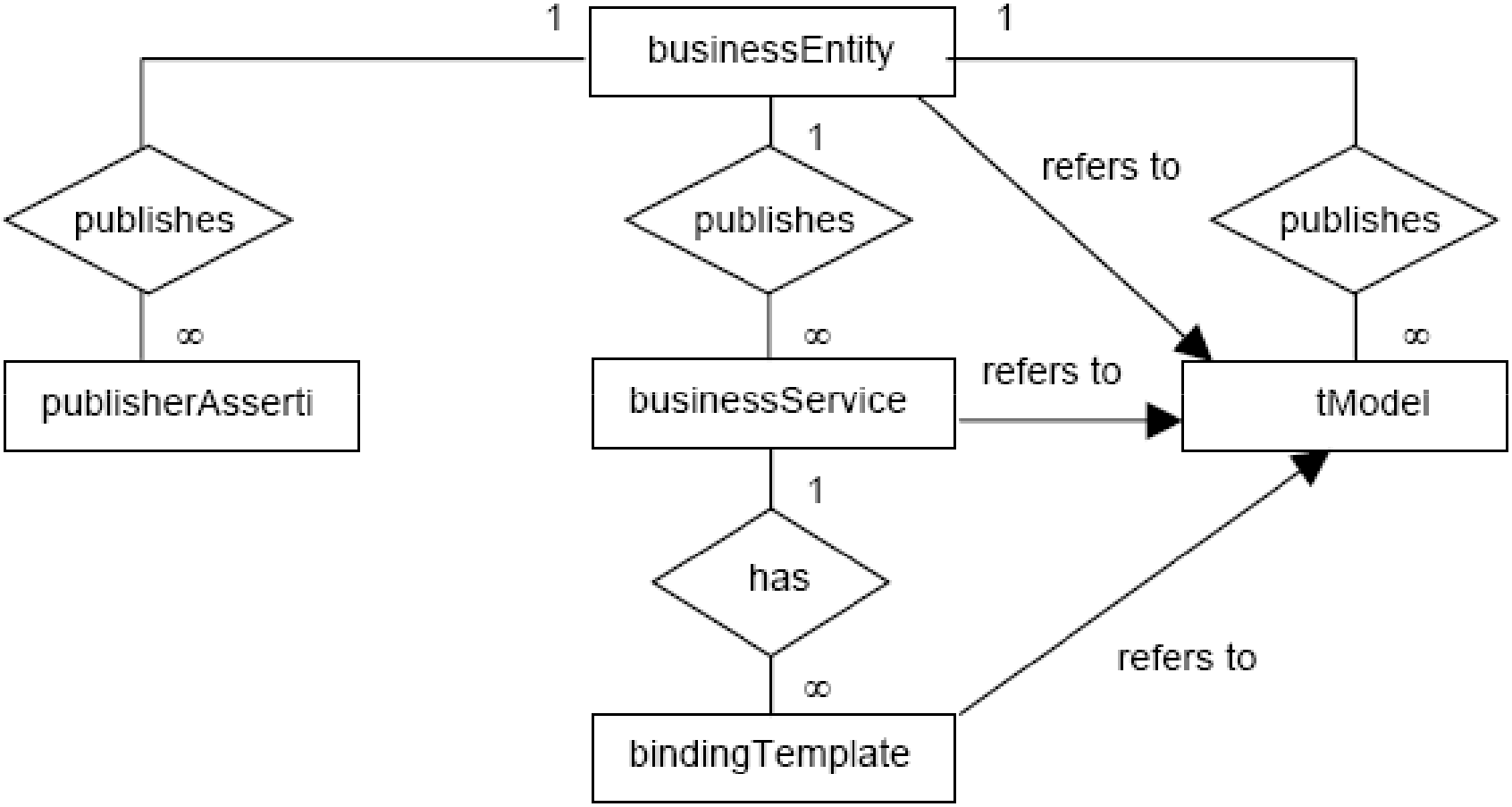
  Dinamico: la ricerca viene effettuata in fase di runtime e di solito perché è stata modificata la posizione del servizio.
- **Scoperta dinamica.** Siccome Uddi è esso stesso un WS, un'applicazione può fare richieste al servizio, trovare dinamicamente il servizio, localizzare il suo punto di accesso, recuperare il Wsdl e collegarsi ad esso tutto in runtime.

UDDI registry

Type of informations

1. businessENtity
2. businessService
3. bindingTemplate
4. tModel

UDDI Information Model



Information model of UDDI

UDDI data structures

businessEntity

- provides information, including identifiers, contact information etc…

[white-pages information]

- includes one or more businessService (service entity) elements that represents the services it provides
- specifies a categoryBag to categorize the business [yellow-pages information]
- a unique key identifies each businessEntity

UDDI data structures

businessService (service entity)

- includes information such as name, description. [white-pages information]
- uniquely identified by a service key
- specifies a categoryBag to categorize the service [yellow-pages information]
- contains a list of bindingTemplates which in turn contains tModelInstanceDetails encoding the technical service information [green-pages information]
- includes reference to its host with a businessKey

UDDI data structures

bindingTemplate

- Address at which the WS is available along with set of tmodels
- Define setting and default of operations parameters
- A businessService may have more templates
- A template belongs to a single business Service

UDDI data structures

tModel

- A WSDL document is registered as a tModel into UDDI registry
- In order to describe a service in more expressive way, an external information is referenced where the type and format of this information should be arbitrary.
- UDDI Specs. leaves the responsibility of defining such arbitrary information types and formats to programmers.
  - to better describe a service we tend to reference information
  - such information type or format should not be anticipated
  - replacing such information about a service with a unique key provides a reference to arbitrary information types

A simplified tModel definition

```
<tModel tModelKey=“”>
   <name>http://www.travel.org/e-checkin-interface</name>
   <description xml:lang=“en”>
          Standard service interface definition for travel services
   </description>
   <overviewDoc>
          <description xml:lang=“en”>
                 WSDL Service Interface Document
          </description>
          <overviewURL>
                 http://www.travel.org/services/e-checkin.wsdl
          </overviewURL>
   </overviewDoc>
   <categoryBag> ...
   </categoryBag>
</tModel>
```

tModels for Categorization

- Using categorization, UDDI directory can be queried for specific type of services.
- Each classification in a taxonomical system is registered as a tModel.
- Three standard taxonomies cited by UDDI are
  - North American Industry Classification System (NAICS) taxonomy – an industry classification
  - The Universal Standard Products and Services Code System (UNSPSC) taxonomy – a classification of products and services
  - The International Organization for Standardization Geographic taxonomy (ISO 3166)

UDDI Registry

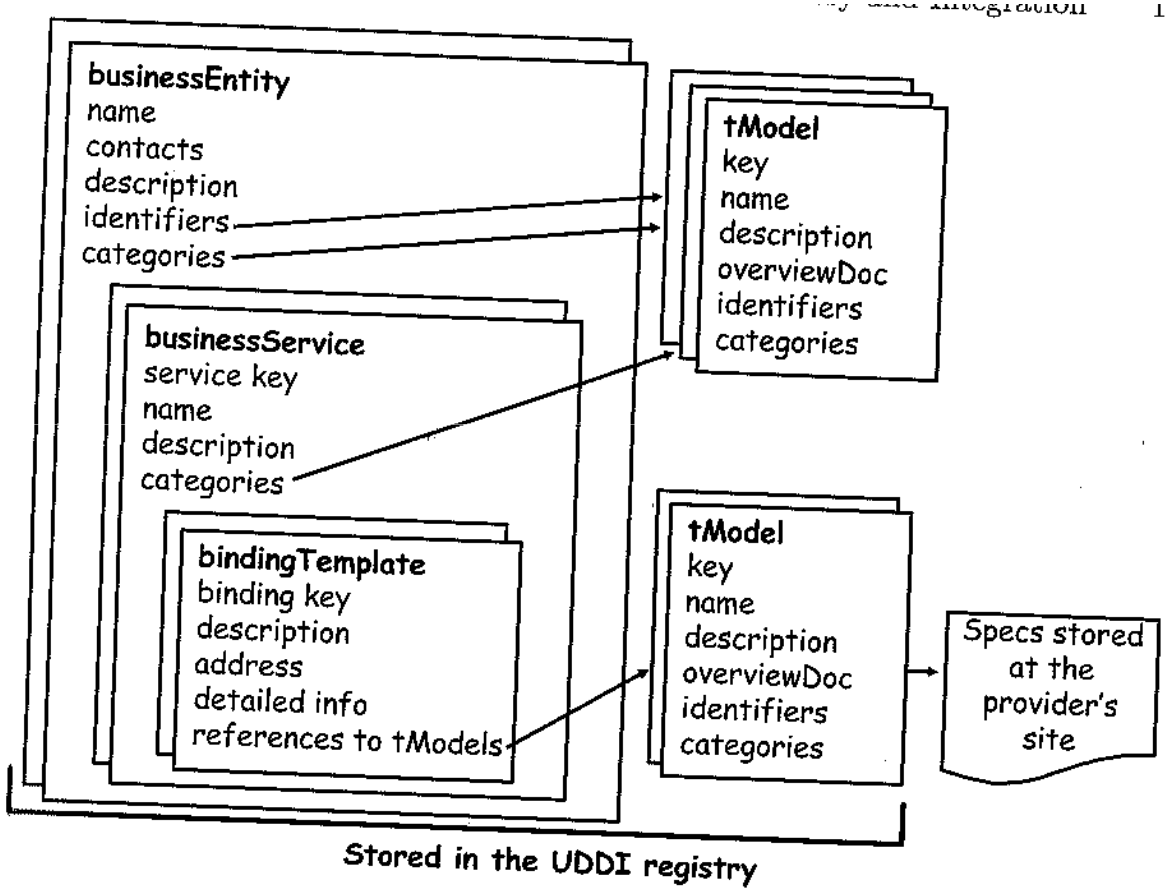


Fig. 6.13. A schematic view of a UDDI registry entry

Service Key

```
<businessService serviceKey=
        "894B5100-3AAF-11D5-80DC-002035229C64"
        businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
  <name>ElectronicTravelService</name>
  <description xml:lang="en">Electronic Travel Service</description>
  <bindingTemplates>
        <bindingTemplate bindingKey=
                "6D665B10-3AAF-11D5-80DC-002035229C64"
                serviceKey="89470B40-3AAF-11D5-80DC-002035229C64">
          <description>
                SOAP-based e-checkin and flight info
          </description>
          <accesssPoint URLType="http">
                http://www.acme-travel.com/travelservice
          </accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo tModelKey="D2033110-3BGF-1KJH-234C-09873909802">
            ...
            </tModelInstanceInfo>
          </tModelInstanceDetails>
        </bindingTemplate>
  </bindingTemplates>
  <categoryBag>
    ...
  </categoryBag>
</businessService>
```

Service Name

Binding Template

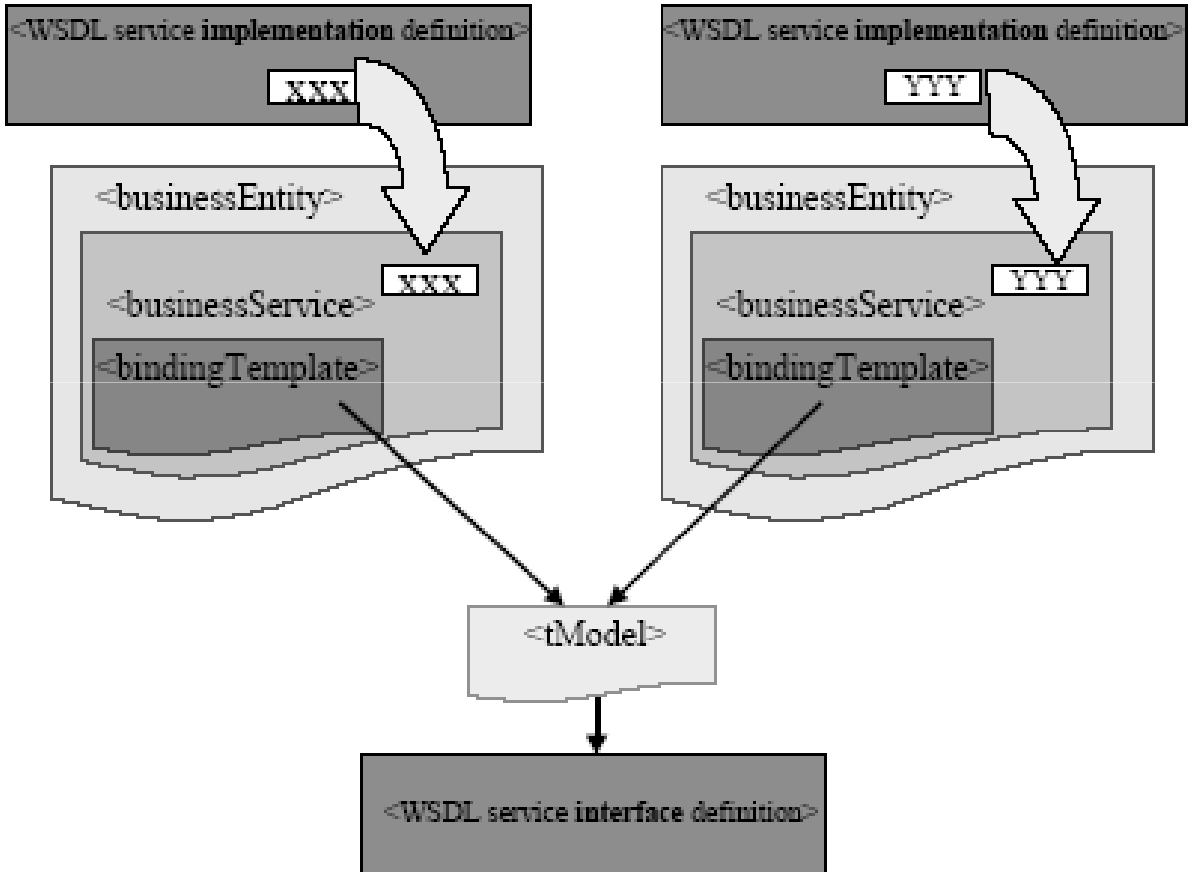
tModelDetails

Category

UDDI Query

- UDDI Search API allows users to query for service providers that provide particular service.
- A UDDI query may be for
  - A business entity
    - Using business name, business key or business category (i.e. find_business())
  - A list of publisher assertions
    - Using business key (i.e. find_relatedBusiness())
  - A business service
    - Using the business key of service key and service name (i.e. find_service())
  - Service key of a bussiness entity
    - Using a binding template (i.e. find_binding())
  - A set of business entities and business services adopting same tModel
    - Using a tModel (i.e. find_tModel())
- After finding the requited UDDI entry, a set of API is used to get details of those entries from UDDI
  - get_businessDetail(), get_serviceDetail(), get_bindingDetail(), get_tModelDetail()

UDDI and WSDL relationship



UDDI and WSDL Relationship.

Binding WSDL a UDDI

```
<?xml version="1.0" encoding="utf-8"?>
<definitions name="StockQuote" targetNamespace="http://example.com/stockquote/" xmlns:tns=http://example.com/stockquote/
  xmlns:xsd1="http://example.com/stockquote/schema/" xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
  xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
      <schema
         targetNamespace="http://example.com/stockquote/schema/"
         xmlns="http://www.w3.org/2001/XMLSchema">
         <element name="TradePriceRequest">
            <complexType><all><element name="tickerSymbol" type="string"/></all></complexType>
         </element>
         <element name="TradePrice">
            <complexType><all><element name="price" type="float"/></all></complexType>
         </element>
      </schema>
   </types>
   <message name="GetLastTradePriceInput">
      <part name="body" element="xsd1:TradePriceRequest"/>
   </message>
   <message name="GetLastTradePriceOutput">
      <part name="body" element="xsd1:TradePrice"/>
   </message>
   <portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
         <input message="tns:GetLastTradePriceInput"/>
         <output message="tns:GetLastTradePriceOutput"/>
      </operation>
   </portType>
   <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
      ....
   </binding>
   <service name="StockQuoteService"> <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
         <soap:address location="http://location/sample"/> </port>
   </service>
</definitions>
```

"Input\Output" Details

input

output

"Method" Specification

"Binding" Details

A sample WSDL

1 portType

- 1 binding

- 1 service

- 1 port

Web Services at work: Putting all together

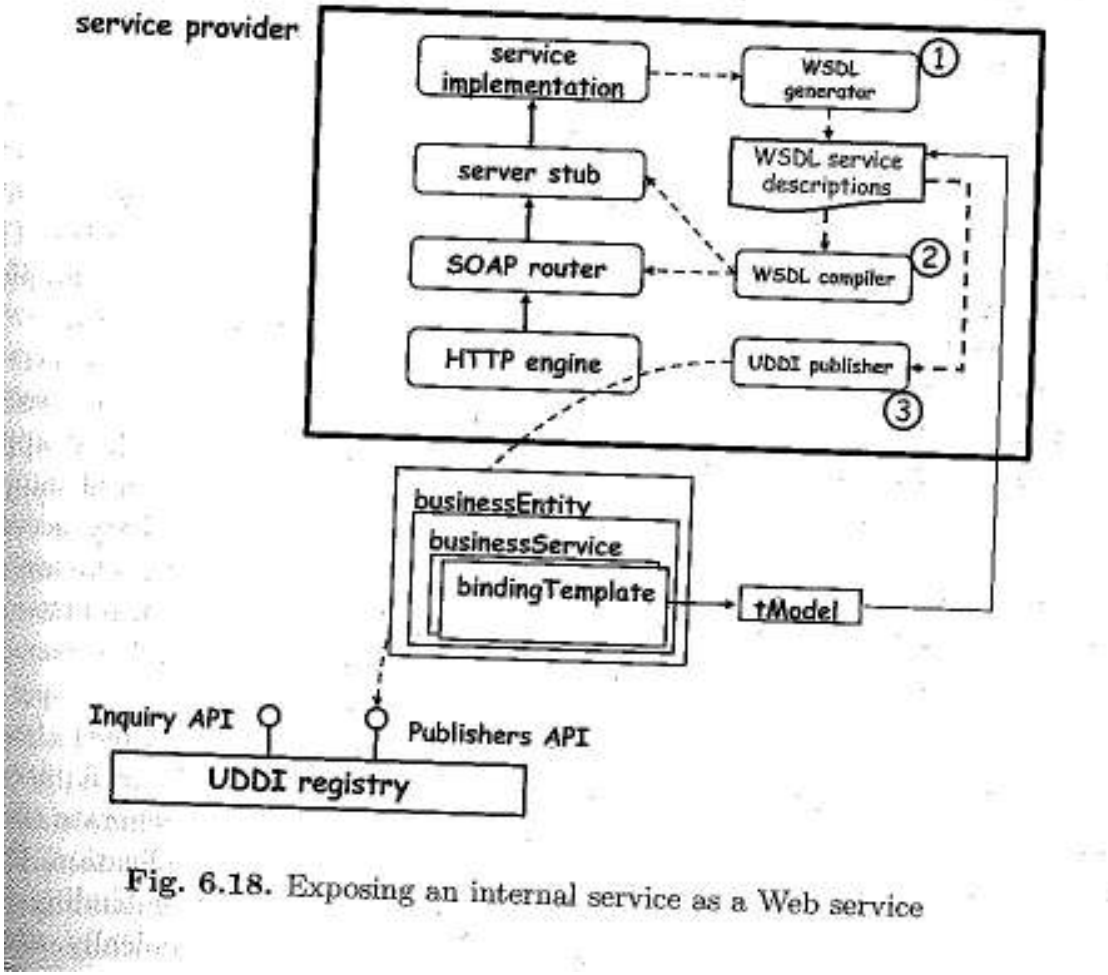


Fig. 6.18. Exposing an internal service as a Web service

GEOGRAPHIC WEB SERVICES

Web Services per GeoWeb

- 2005 : Release API di Google Maps
- Oggi: migliaia di websites utilizzano API di Google, MS Virtual Earth e Yahoo Maps
- Obiettivo: versione georeferita del Web 2.0
- Website conferenza
  - Localizzazione su mappa, percorsi e orari mezzi pubblici
  - Ristoranti, hotel, recensioni della community
  - Mappa sociale città:geotags utenti, statistiche sicurezza,densità demogr.,sviluppo edilizio, aree verdi

Web Services come APIs

- WS può essere accessibile da Application Programming Interface
  - Descrizione WS espressa come interfaccia
- API è specifica (non implementazione) di insieme:
  - Funzioni, Procedure,Metodi,Classi,Protocolli
- API fornite da
  - piattaforma , e.g. J2EE
  - linguaggio, e.g. JavaScript
  - OS, e.g. MS Windows

Pubblicare una mappa - Google API

- Codice javascript nella pagina web, utilizzo funzioni fornite da Google Maps API
- Classe Gmap2(container,opts?)
- Metodi:
  - addOverlay(overlay) %Aggiunge un overlay
  - openInfoWindow(latlng) %Apre finestra info da coord
  - fromLatLngToPixel %compute pixel from coord
  - click(overlay,latlng)%lancia evento da click su mappa

Open Geospatial Consortium ®

- OGC ~ 400 organizzazioni ~30 standards per geo-web
- Attività
  - Standards per
    - linguaggi(Geometry Markup Language,SensorML,.)
    - accesso ai dati nei DB (Simple features,..)
    - web services (Web Map Service,..)
  - abstract specifications (in UML)
  - compliance testing (analisi metadati)

Standard per interoperabilità

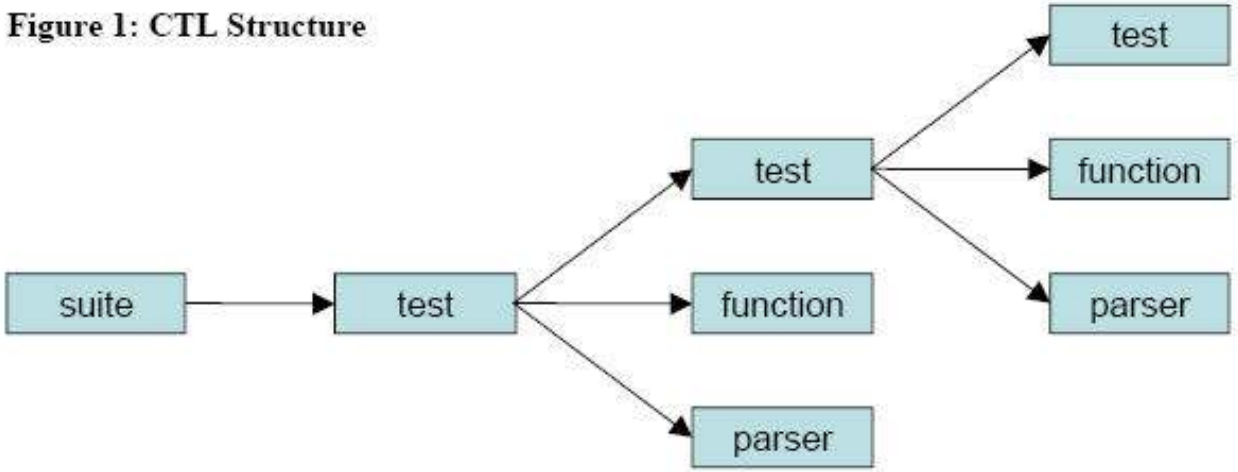
- OGC Specifica interfaccia operazioni rese disponibili da servizi
- WMS: GetCapabilities, GetMap, GetFeatureInfo
- Specifica parametri input/ouput
  - Indipendente da implementazione
- Esempio: GetCapabilities (descrizione WMS metadata)

Request Parameter	Mandatory/ Optional	Description
VERSION=version	O	Request Version
SERVICE=WMS	M	Service Type
REQUEST=GetCapabilities	M	Request Name
FORMAT=MIME_type	O	Output format of service metadata

OGC Compliance Testing Program

- Processo per testare implementazioni di specifiche OGC
- CTL engine: analisi metadati XML spediti/ricevuti da implementazioni di client e servers
- Compliance solo se metadati concordano con specifica schema

Figure 1: CTL Structure



WMS OGC compliance

```
<!-- ********************************************************************** -->
<!-- **  Type Definitions.                                                 ** -->
<!-- ********************************************************************** -->

<simpleType name="longitudeType">
  <restriction base="double">
    <minInclusive value="-180" />
    <maxInclusive value="180" />
  </restriction>
</simpleType>

<simpleType name="latitudeType">
  <restriction base="double">
    <minInclusive value="-90" />
    <maxInclusive value="90" />
  </restriction>
</simpleType>

</schema>
```

Overview of CTL

CTL objects

- Suite: defines a set of tests
- Test: contains assertion and code to test the assertion
- Function: declares functions (user defined, external java)
- Instruction: retrieves user input (uses XHTML form)
  - Request: submits HTTP reques to WS and return HTML response
  - For-each: provides cycles of test
  - Message: presents a message to the user
  - Fail: indicates that a test has failed

Built-in parsers provided

Open problems in the OGC Compliance specification

- *Desiderata*: “OGC Compliance Testing Program is based on a more **formal** process for testing compliance of products to OpenGIS® Implementation Specifications. Compliance Testing determines that a product implementation of a particular OpenGIS® Implementation Specification fulfills all **mandatory** elements as specified and that these elements are operable”
- *Actual situation*: lack of formalization/semantics
  - Verbose specification is confusing, formal specification should be preferred
  - Human reading XML Schemas specification could be hard
  - Interoperability still not treated
  - Need for a (semi-)automatic procedure based on specifications rather than on test

Problem's example: WMS compliance

Making a map server WMS (the server instance) compliant is mostly about implementing the Capabilities XML file. See Chapter 3 for recipes on how to do this. The lat/lon **deegree** and UMN MapServer recipes are good examples of this.

Usually a map server needs to be compiled so that the various WMS parameters are set (described earlier and/or referenced to the WMS 1.1.1 specificaiton). Knowing that the server can produce a valid XML `GetCapabilities` response, the `GetMap` request can be utilized. Simply adding `"VERSION=1.1.0&REQUEST=GetMap"` to your server's URL should generate a map with the default map size.

For example, in the case of the UMN Mapserver, a mapfile - regular MapServer mapfile - was made in which some parameters and some metadata entries are mandatory (some parameters are the map level and others at the layer level). Most of the metadata is required in order to produce a valid `GetCapabilities` output for the WMS client.

Making a Web client WMS compliant is mostly about implementing the `GetMap` request and its parameters. See Chapter 3, for the WMS client setup with UMN MapServer, which is a good example of this.

Harvard University user experience in Chapter 2 discusses the use of the Proj4 Cartographic Projection Library, necessary for WMS Clients.

Geo-sicurezza:controllo degli accessi

- Recentemente OGC definisce Geospatial eXtensible Access Control Markup Language (GeoXACML)
  - estensione geografica di OASIS XACML (XML)
- Controllo degli accessi interoperabile nelle SDI
- Policies esprimibili:
  - Accesso in lettura se risorsa non contiene oggetti di classe building
  - Bob può leggere oggetti di classe building se geometria di proprietà shape è in data area
  - Alice non può leggere oggetti building se location è tra area Gp1 e Gp2

Ambiti di ricerca per Geo Web Services

- Linguaggi estesi con caratteristiche spazio-temporali
- Teorie,tecniche e standard per web services geografici
  - Coreografia? (WS-CDL)
  - Geo-sicurezza
  - Controllo discrezionale degli accessi
- Ingegneria del Geo-software
    - Implementazioni documentate e re-ingegnerizzazione
- Verifica automatica di OGC compliance a partire di specifiche WSDL e OGC reference models in XML-Schemas

FSE'09 - Argomenti

1. Cosa sono i Web Services
2. XML: Il linguaggio dei Web Services
3. Tecnologia dei Web Services
4. Come costruire un sistema SOA con Apache Axis
5. Advanced Topics: messaging e composizione di WS

Apache Axis (http://ws.apache.org/axis2/)

- Java/C++ framework open source per realizzare Web Services
- SOAP and REST (Axis2) engine
- Server stand-alone o instanziabile in servlet (Tomcat)
- SAX parsing
- Generazione automatica WS da WSDL (e viceversa) !

Axis: un semplice Service Requestor

```
// Definisce l'address dell'ultimateReceiver del messaggio
String endpoint = "http://localhost:8080/axis/Calculator.jws";

// Instanzia un nuovo servizio (non il server!!!)
Service  service = new Service();

// Crea una chiamata remota ad un endpoint di un servizio
Call     call    = (Call) service.createCall();

// Setta l'indirizzo del WebService da chiamare/destinazione del msg SOAP
call.setTargetEndpointAddress( new java.net.URL(endpoint) );

// Setta il nome dell'operazione da chiamare/metodo del Web Service
call.setOperationName(new QName("http://soapinterop.org,echoString"));

// Invoca il servizio e ne prende il risultato
String answer = (String) call.invoke( new Object[] {"Hello"} );

// Stampa il risultato
System.out.println("Sent Hello!, Got " + answer);

> example1.TestClient
Sent Hello!, got Hello!
```

La richiesta SOAP

- Stringa “Hello!” automaticamente serializzata in XML
- Server risponde con “Hello!” deserializzandola

```
<?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>
```

Pubblicare un WS con AXIS: Java Web Service (JWS)

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

- Copiare il sorgente in
  - <your-webapp-root>/axis/<service-name>.jws
- Il Web service e' gia' funzionante in
  - <axis-url>/<service-name>.jws
- E' possibile ottenerne il WSDL attraverso
  - <axis-url>/<service-name>.jws?WSDL

JWS: Considerazioni

- JWS e' usato solo per debug e sviluppo di semplici Web Service
  - Non e' possibile usare packages
  - La compilazione a run time rende impossibile fare type checking
  - E' necessario distribuire il sorgende del Web Service per pubblicarlo
  - Limitate possibilita' di configurazione
    - Non e' possibile specificare particolari mapping per I tipi
    - Non e' possibile definire Handler specifici per il servizio

Pubblicazione con Web Service Deployment Descriptor (WSDD)

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="MyService" provider="java:RPC">
    <parameter name="className" value="Sample.MyService"/>
    <parameter name="allowedMethods" value="*"/>
  </service>
</deployment>
```

- Descrive come rendere disponibile un servizio in Axis
  - L'element service definisce un servizio. Un servizio e' un chain, che puo' avere un request flow, un pivot Handler (provider) e un response flow. In questo caso il pivot e' java:RPC, che e' all'inerno di Axis
  - Il parametro className specifica all'RPCProvider quale classe implementa la logica del servizio
  - Il parametro allowedMethod specifica all'engine che tutti i metodi pubblici dell'oggetto possono essere chiamati via SOAP (questi possono essere ristretti specificando il nome di un metodo o dando una lista di metodi separati da ",")

WSDD: Web Service Deployment Descriptor (2)

- Per pubblicare un servizio e' sufficiente usare l'AdminClient
  - java org.apache.axis.client.AdminClient deploy.wsdd
- E' necessario comunque fornire l'implementazione del servizio all'engine di axis
  - Copiare il bytecode del servizio in:
    <your-webapp-root>/axis/WEB-INF/classes
- E' possibile ottenere l'elenco dei servizi attivi attraverso
  - java org.apache.axis.client.AdminClient list

WSDD: Scope di un servizio (1)

- Quale instanza della classe java implementa il servizio?
- Tre modelli di scope:
  - Request (default) : viene creato un nuvo oggetto per ogni request SOAP del servizio.
  - Application : Viene creato un oggetto singleton condiviso tra tutte le request.
  - Session : Viene creato un nuovo oggetto per ogni client session-enabled che accede al servizio.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="MyService" provider="java:RPC">
    <parameter name="className" value="Sample.MyService"/>
    <parameter name="allowedMethods" value="*"/>
    <parameter name="scope" value="value"/>
  </service>
</deployment>
```

WSDD: Scope di un servizio (2)

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="MyService" provider="java:RPC">
    <parameter name="className" value="sample.MyService"/>
    <parameter name="allowedMethods" value="*"/>
  </service>
  <service name="MyServiceApplication" provider="java:RPC">
    <parameter name="className" value="sample.MyService"/>
    <parameter name="allowedMethods" value="*"/>
   <parameter name="scope" value="Application"/>
  </service>
  <service name="MyServiceSession" provider="java:RPC">
    <parameter name="className" value="sample.MyService"/>
    <parameter name="allowedMethods" value="*"/>
   <parameter name="scope" value="Session"/>
  </service>
</deployment>
```

WSDD: Request Flow

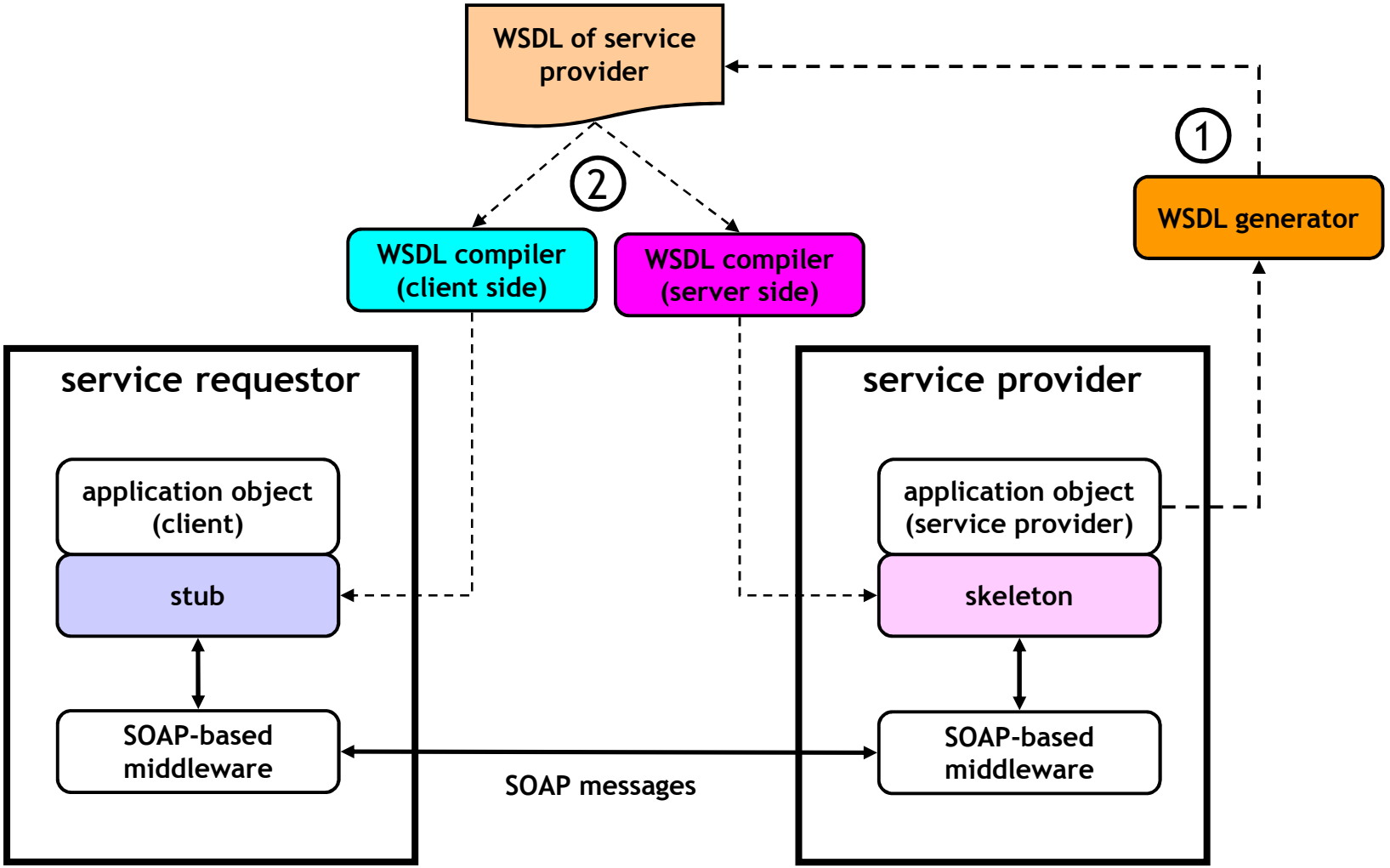
- Come loggare le request ad un servizio?

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <!-- define the logging handler configuration -->
  <handler name="track" type="java:sample.LogHandler">
    <parameter name="filename" value="MyService.log"/>
   </handler>

  <!-- define the service, using the log handler we just defined -->
  <service name="LogTestService" provider="java:RPC">
    <requestFlow>
      <handler type="track"/>
    </requestFlow>

    <parameter name="className" value="sample.MyService"/>
    <parameter name="allowedMethods" value="*"/>
  </service>
</deployment>
```

WSDL2Java: Generazione di Stub e Skeleton da WSDL



WSDL clauses 2 Java classes

Per ogni entry nella sezione type	Una classe java Un Holder se il e' usato come parametro di input o di output
Per ogni PortType	Una interfaccia java
Per ogni binding	Una classe stub
Per ogni servizio	Un'interfaccia per il servizio Una implementazione del servizio (locator)

WSDL2Java address.wsdl: Type

```
<xsd:complexType name="phone">
  <xsd:all>
    <xsd:element name="areaCode" type="xsd:int"/>
    <xsd:element name="exchange" type="xsd:string"/>
    <xsd:element name="number" type="xsd:string"/>
  </xsd:all>
</xsd:complexType>
```

```
public class Phone implements java.io.Serializable {
  public Phone() {...}
  public int getAreaCode() {...}
  public void setAreaCode(int areaCode) {...}
  public java.lang.String getExchange() {...}
  public void setExchange(java.lang.String exchange) {...}
  public java.lang.String getNumber() {...}
  public void setNumber(java.lang.String number) {...}
  public boolean equals(Object obj) {...}
  public int hashCode() {...}
}
```

WSDL2Java address.wsdl: PortType

```
<message name="empty">
<message name="AddEntryRequest">
  <part name="name" type="xsd:string"/>
  <part name="address" type="typens:address"/>
</message>
<portType name="AddressBook">
  <operation name="addEntry">
    <input message="tns:AddEntryRequest"/>
    <output message="tns:empty"/>
  </operation>
</portType>
```

```
public interface AddressBook extends java.rmi.Remote {
  public void addEntry(String name, Address address) throws
      java.rmi.RemoteException;
}
```

WSDL2Java address.wsdl: Binding

```
<binding name="AddressBookSOAPBinding" type="tns:AddressBook">
  ...
</binding>
```

```
public class AddressBookSOAPBindingStub extends
   org.apache.axis.client.Stub
    implements AddressBook {
  public AddressBookSOAPBindingStub() throws org.apache.axis.AxisFault
   {...}

  public AddressBookSOAPBindingStub(URL endpointURL,
      javax.xml.rpc.Service service) throws org.apache.axis.AxisFault
   {...}

  public AddressBookSOAPBindingStub(javax.xml.rpc.Service service)
      throws org.apache.axis.AxisFault {...}

  public void addEntry(String name, Address address)
      throws RemoteException {...}
}
```

WSDL2Java address.wsdl: Services

```
<service name="AddressBookService">
  <port name="AddressBook" binding="tns:AddressBookSOAPBinding">
    <soap:address
   location="http://localhost:8080/axis/services/AddressBook"/>
  </port>
</service>
```

```
public interface AddressBookService extends javax.xml.rpc.Service {
  public String getAddressBookAddress();

  public AddressBook getAddressBook() throws
   javax.xml.rpc.ServiceException;

  public AddressBook getAddressBook(URL portAddress)
      throws javax.xml.rpc.ServiceException;
}

public class AddressBookServiceLocator extends
   org.apache.axis.client.Service
    implements AddressBookService {
  ...
}
```

WSDL2Java address.wsdl: Esempio d'uso

```
public class Tester {
  public static void main(String [] args) throws Exception {
    // Prende il servizoi
    AddressBookService service = new AddressBookServiceLocator();

    // Usa il servizio per prendere uno stub che implementi l'SDI.
    AddressBook port = service.getAddressBook();

    // Effettua la chiamata
    Address address = new Address(...);
    port.addEntry("Russell Butek", address);
  }
}
```

check.wsdl client

```
public class CheckClient {
   public static void main(String[] args) {
        // Prende il servizio
        CheckLocator srv = new CheckLocator();

        try {
            // Prende lo stub
            CheckSoap check = srv.getcheckSoap();

            // Effettua l'invocazione
            DocumentSummary doc = check.checkTextBody("Helllo, i'm not
   mor interested", "");

            // Stampa i risultati
            for (int i=0; i<doc.getMisspelledWordCount(); i++) {
                System.out.println(doc.getMisspelledWord(i).getWord());
            }
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
   }
}
```

FSE'09 - Argomenti

1. Cosa sono i Web Services
2. XML: Il linguaggio dei Web Services
3. Tecnologia dei Web Services
4. Come costruire un sistema SOA con Apache Axis
5. Advanced Topics: messaging e composizione di WS

WS MESSAGING

XML and SOAP Messaging

- **Message**: a collection of data fields sent or received together between software applications. A message contains a ***header*** (which stores control information about the message) and a ***payload*** (the actual content of message, also called ***body***).
- **Messaging** can be either ***synchronous*** (two-way) or ***asynchronous*** and use many *different protocols* such as HTTP or SMTP, as well as MOM (message-oriented middleware) products (e.g. JMS).
- **SOAP** is an XML messaging/RPC (Remote Procedure Call) standard to enable the exchange of a variety of information.

Alternatives for communication

- **RPC-based** approach: the **client** (or caller) of the procedure call **knows** the **procedure** it wants to invoke and knows the **parameters** it wishes to pass to the procedure. Also, the caller wishes to **wait** while the called **server** (the application) **completes** the request.
- **Message oriented** approach: the caller **does not necessarily know** the exact **procedure** that will be invoked, **but instead** creates a **message** of a specific **format** known to both the client and the server
  - the **client does not depend on the server or the server's procedures**, but is dependent on the message format
  - communication likely takes place in an **asynchronous** fashion

Advantages of message oriented approach

- Easier message routing and transformation
- More flexible payload (can include binary attachments, for example)
- Can use several messages together to invoke a given procedure

Disadvantages

- **More work to develop** a client/server interaction w.r.t. RPC
- Complexity is added through the **creation of the message on the client side** (versus a procedure invocation in an RPC approach) **and on the server side** with message-processing code.
- More difficult to understand and debug
- Risk of losing type information for the programming language in which the message was sent
  - Double in Java may not translate as a double in the message
- Transactional **context** in which the message was created in general **does not propagate** to the messaging server.

Why XML messaging...?

- The most common scenario occurs when the client/server communication takes place over the Internet and the

  client and server belong to different companies

  In this scenario it could be fairly **difficult** to have the two companies **agree on the procedure interface**.

- The companies involved may want to use an **asynchronous** communication model.
- Another cenario occurs when you're developing an **event-based system** in which events are created and then consumed by interested parties.

Message broker

- Acts as the ***server*** in a message-oriented system.
- Performs operations on received messages
- Header processing
- Security
- Error and exception handling
- Routing
- Invocation
- Transformation

Header processing

- Header processing could include ***adding*** a *tracking number* to an incoming message or ensuring that all of the header fields necessary to process the message exist.
- For example, the message broker could check the **to** header field to ensure that this is the proper destination for this message.

Security

- Security operation
  - ***Authentication***, ***authorization***, and ***encryption***.
- The message broker:

1. A*uthenticates messages* against a security database or directory.
2. *Authorizes operations* that can be performed with this type of message and an *authorized* principal.
3. Decrypt *messages* that arrives *encrypted* using some encryption scheme and process it further.

Error and exception handling

- *Message* sent to the broker *does not contain sufficient or accurate information*
  - Broker will respond to the client (assuming a synchronous invocation) with an error message
- Servicing *an undefined request*
  - invoking a procedure/method in the payload of the message where Method (service) not defined.

Message Routing

- Branching logic for messages at **two different levels**

1. **header-level routing** determines if an incoming message is bound for this application or needs to be resent to another application.
2. **payload routing** determines which procedure or method to invoke once the broker determines that the message is bound for this application.

Invocation & Transformation

- **Invocation** means to actually call or invoke a method with the data (payload) contained in the incoming message. This could produce a result that is then returned through the broker back to the client.
- **Transformation** converts or maps the message to some other format. With XML, XSLT is commonly used to perform transformation functionality.

Example.

```
<?xml version="1.0"?>
  <message>


      <header>
        <to>companyReceiver</to>
        <from>companySender</from>
        <type>saveInvoice</type>
      </header>
```

Example..

```
<body>
    <saveInvoice>
      <invoice date="01-20-2000" number="123">
        <address country="US"><name>John Smith</name>
            <street>123 George St.</street><city>Mountain View</city>
            <state>CA</state><zip>94041</zip>
        </address>
        <billTo country="US">
            <name>…</name><street>...</street>
            <city>…</city><state>…</state><zip>…</zip>
        </billTo>
        <items>
            <item number="1">
              <name>…</name><quantity>…</quantity>

              <USPrice>…</USPrice>
            </item>
        </items>
      </invoice>
    </saveInvoice>
```

Broker for the example

- **Broker** accepting **two types of messages**
  - a **request** to create an invoice on the filesystem
  - a **client code component** reading the XML message from a file
- The broker comprises **three main parts**

1. listener receives incoming messages on some transport, places the incoming message into a string variable, and calls the main broker
2. main broker parses the message into a DOM and decides which invoker class to call based on the contents of the message.
3. invocation piece performs some logic based on the incoming message.

SOAP messaging

- SOAP (Simple Object Access Protocol) was originally proposed by Microsoft but has been subsequently adopted by IBM and many other companies, including, more recently, Sun Microsystems.
- SOAP is an **XML messaging/RPC** standard to enable the exchange of a variety of information.
- Comprises three major **components**: a *messaging framework*, an **encoding** standard, and an **RPC** mechanism.
- It **does not detail** what **properties** are associated with the **exchange** (e.g. transactional, encrypted,…) **but** simply specifies a generic **message template** to add to the top of the application data.
- It is **not tied to** any particular **transport** protocol.
- It in **not tied to** any particular **operating system** or **programming language**.

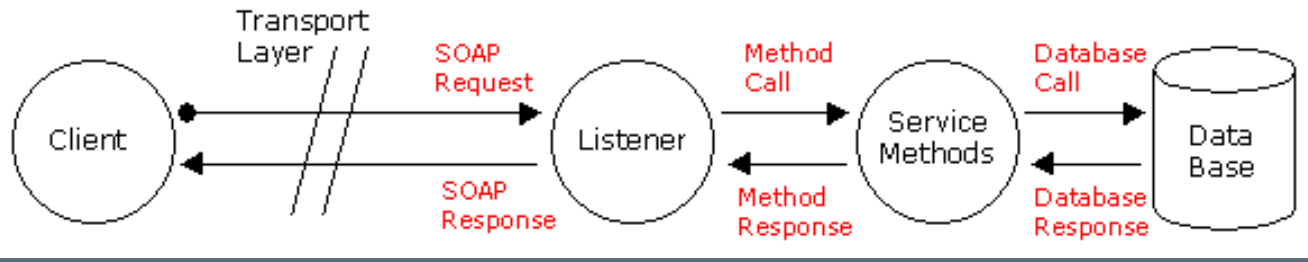
SOAP RPC messaging by examples

- Imagine you have a very simple **corporate database** that holds a **table** specifying **employee** *reference number*, *name and telephone number*.
- You want to offer a service that enables other systems in your company to do a **lookup** on **this data**.
- The service should **return** a *name* and *telephone number* (a two element *array of strings*) **for** a given *employee reference number* (an *integer*).

    String[] **getEmployeeDetails** (**int** employeeNumber)

SOAP by examples …

- The SOAP developer's **approach** to such a problem is to **encapsulate** the database **request logic** for the service in a method.
- The next step is to set up a **process** that **listens for requests** to the service.
- Such **requests** being in **SOAP format** and containing the *service name* and any *required parameters*.
- The **listener process decodes** the incoming **SOAP request** and **transforms** it **into** an **invocation** of the method.
- It then **takes** the **result** of the method call, **encodes** it **into** a **SOAP** mes



Web Services and the Service Web

- A **Web Service** is a **method** that is **callable remotely** across a network.
- The fundamental differences from traditional content-based internet services are:
  - *Content-Based Services* serve up web pages (whether static or dynamically generated) for *human consumption*.
  - *Web Services* serve up data for *computers*.

SOAP Messages

- A valid **SOAP Message** is a **well-formed** XML document.
- The **XML prolog** is optional and should contain only an XML Declaration (e.g. it should not contain any DTD references or XML processing instructions).
- It should use the **SOAP Envelope** and **SOAP Encoding namespaces** and have the following form:
  - An **XML Declaration** (*optional*), followed by
  - A **SOAP Envelope** (the **root** element) which is made up of:
    - A **SOAP Header** (optional)
    - A **SOAP Body**

Example

String[] ***getEmployeeDetails*** (int employeeNumber);

SOAP Messages… (Request)

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">


  <SOAP-ENV:Body>
     <ns1:getEmployeeDetails xmlns:ns1="urn:MySoapServices">
       <param1 xsi:type="xsd:int">1016577</param1>
      </ns1:getEmployeeDetails>
  </SOAP-ENV:Body>


</SOAP-ENV:Envelope>
```

SOAP Messages… (Response)

```
<?xml version="1.0" encoding="UTF-8" ?>
 <SOAP-ENV:Envelope
      xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema"
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body>
<ns1:getEmployeeDetailsResponse
      xmlns:ns1="urn:MySoapServices"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
      xsi:type="ns2:Array" ns2:arrayType="xsd:string[2]">
      <item xsi:type="xsd:string">Bill Posters</item>
      <item xsi:type="xsd:string">+1-212-7370194</item>
</return>
</ns1:getEmployeeDetailsResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

COMPOSIZIONE DI WS

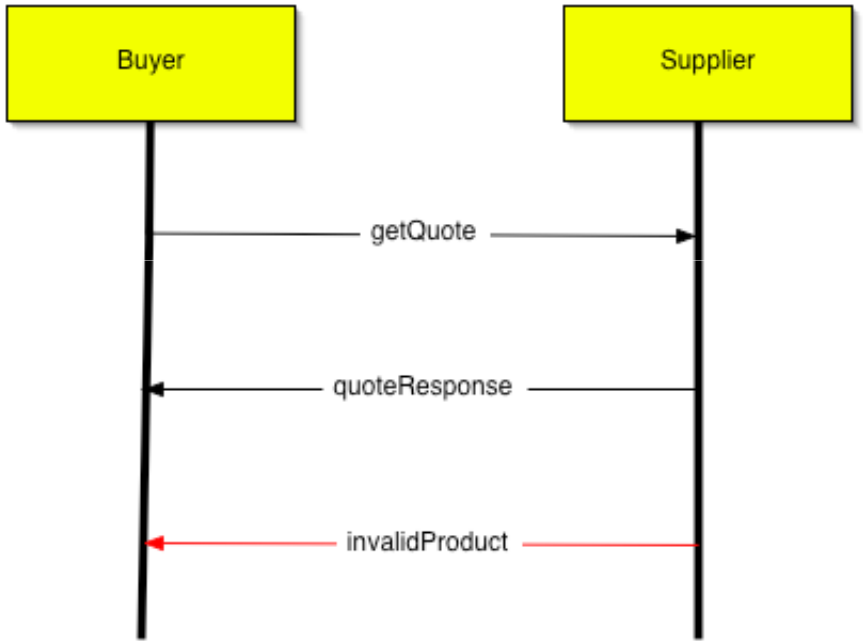
Un’ applicazione composta

Acquisto formula volo+noleggio auto (**1 Client 3 Server**)

- Cliente invia dati a Web Service (WS) compagnia area
- WS airline inoltra dati cliente a WS autonoleggio
- WS autonoleggio comunica importo a WS airline
- WS airline inoltra importo a WS carta credito
- WS carta credito chiede nr carta a cliente
- Cliente fornisce numero a WS carta credito
- WS carta credito dà ok a WS airline
- WS airline dà ok a WS autonoleggio e Cliente

Global view: Choreography Description Language (WS-CDL)

- Buyer and seller services, from a global perspective
- Each has WSDL description
- WS-CDL describes observable interactions
- Interaction=collaboration between roles



WS-CDL Example

```
<interaction name="QuoteElicitation" operation="getQuote"
     channelVariable="tns:Buyer2SellerC">
   <description type="documentation">
      Quote Elicitation
   </description>
   <participate relationshipType="tns:Buyer2Seller"
    fromRoleTypeRef="tns:BuyerRole" toRoleTypeRef="tns:SellerRole"/>
   <exchange name="QuoteRequest" informationType="tns:QuoteRequestType"
    action="request">
      <description type="documentation">
            Quote Request Message Exchange
      </description>
      <send variable="cdl:getVariable('quoteRequest','','')"/>
      <receive variable="cdl:getVariable('quoteRequest','','')"/>
   </exchange>
```

What to do to describe a choreography

1. Define our roleTypes,
2. Define our relationshipTypes,
3. Define our informationTypes,
4. Define our tokenType,
5. Define our channelTypes

Roles

```
<roleType name="BuyerRole">
    <description type="documentation">
        Role for Buyer
    </description>
    <behavior name="BuyerBehavior" interface="BuyerBehaviorInterface">
        <description type="documentation">
            Behavior for Buyer Role
        </description>
    </behavior>
</roleType>
<roleType name="SellerRole">
    <description type="documentation">
        Role for Seller
    </description>
    <behavior name="SellerBehavior" interface="SellerBehaviorInterface">
        <description type="documentation">
            Behavior for Seller
        </description>
    </behavior>
</roleType>
```

Relationship types

```
<participantType name="Seller">
     <description type="documentation">
        Seller Participant
    </description>
    <roleType typeRef="tns:SellerRole"/>
</participantType>
<participantType name="Buyer">
    <description type="documentation">
        Buyer Participant
    </description>
    <roleType typeRef="tns:BuyerRole"/>
</participantType>
```

Information types

```
<informationType name="QuoteRequestType" type="primer:QuoteRequestMsg">
   <description type="documentation">
      Quote Request Message
   </description>
</informationType>
<informationType name="QuoteResponseType" type="primer:QuoteResponseMsg">
   <description type="documentation">
      Quote Response Message
   </description>
</informationType>
<informationType name="QuoteResponseFaultType"
     type="primer:QuoteResponseFaultMsg">
   <description type="documentation">
      Quote Response Fault Message
   </description>
</informationType>
```

Information types

```
<informationType name="IdentityType" type="xsd:string">
    <description type="documentation">
        Identity Attribute
    </description>
</informationType>
<informationType name="URI" type="xsd:uri">
    <description type="documentation">
        Reference Token For Channels
    </description>
</informationType>
```

Channel type

```
<informationType name="IdentityType" type="xsd:string">
    <description type="documentation">
        Identity Attribute
    </description>
</informationType>
<informationType name="URI" type="xsd:uri">
    <description type="documentation">
        Reference Token For Channels
    </description>
</informationType>
```

Channel type

```
channelType::=
<channelType  name="ncname"
    usage="once"|"unlimited"?
    action="request-respond"|"request"|"respond"? >

<passing  channel="qname"
     action="request-respond"|"request"|"respond"?
    new="true"|"false"? />*
<role  type="qname"  behavior="ncname"? />

<reference>
<token name="qname"/>
</reference>

<identity>
<token name="qname"/>+
</identity>?

</channelType>
```

The WS-CDL for the Buyer-Seller

- Declare the relationships, which act as an additional type check on the channels.
- Declare variables, for instances of the channel and the information types.
- Define the interactions and their ordering constraints
- The complete definition

Local view: orchestration

- Orchestration vs choreography
- Orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer
- A choreography specifies a protocol for peer-to-peer interactions, defining, e.g., the legal sequences of messages exchanged with the purpose of guaranteeing interoperability. Such a protocol is not directly executable, as it allows many different realizations
- A choreography can be realized by writing an orchestration (e.g. in the form of a BPEL process) for each peer involved in it

Business Process Execution Language Design Goals

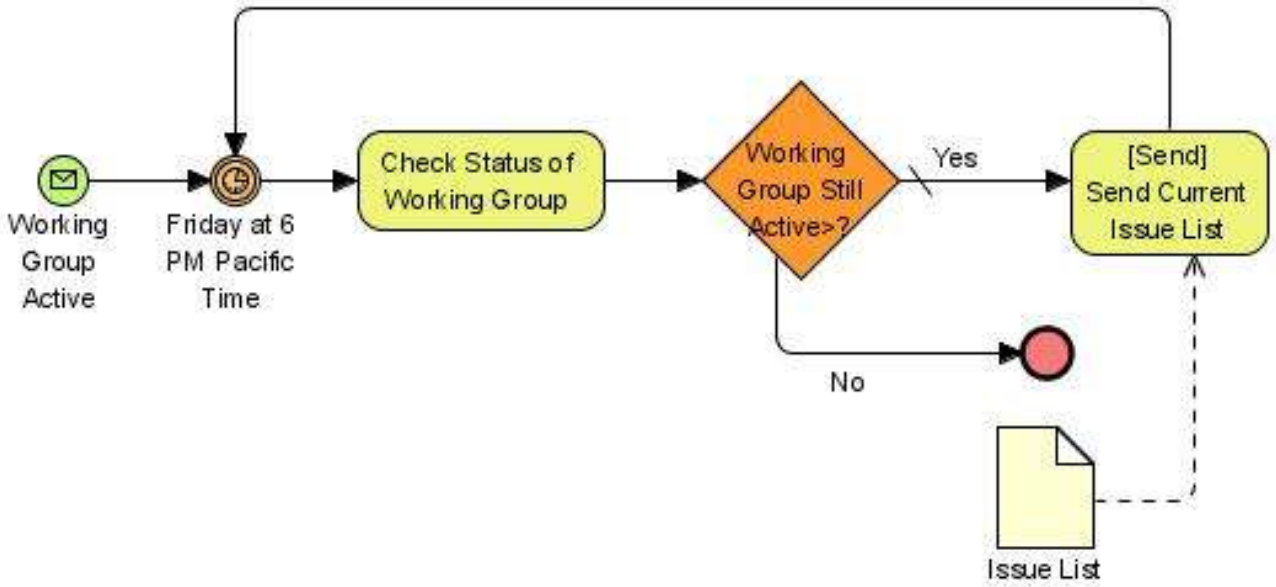
- Define business processes that interact with external entities through Web Service operations defined using WSDL.The interactions are “abstract” in the sense that the dependence is on portType definitions, not on port definitions.
- Define business processes using an XML-based language
- Define a set of Web service orchestration concepts that are meant to be used by both the external (abstract) and internal (executable) views of a business autonomous process
- Support an identification mechanism for process instances
- Support the implicit creation and termination of process instances
- Define a long-running transaction model with compensations

BPEL

- A property-based message correlation mechanism
- XML and WSDL typed variables
- An extensible language plug-in model to allow writing expressions and queries in multiple languages: BPEL supports XPath 1.0 by default
- Structured-programming constructs including if-then-elseif-else, while, sequence (to enable executing commands in order) and flow (to enable executing commands in parallel)
- A scoping system to allow the encapsulation of logic with local variables, fault-handlers, compensation-handlers and event-handlers
- Serialized scopes to control concurrent access to variables

BPMN

- Business Process Modeling Notation is a graphical front-end to capture BPEL process descriptions



CORRECT CHOREOGRAPHIES

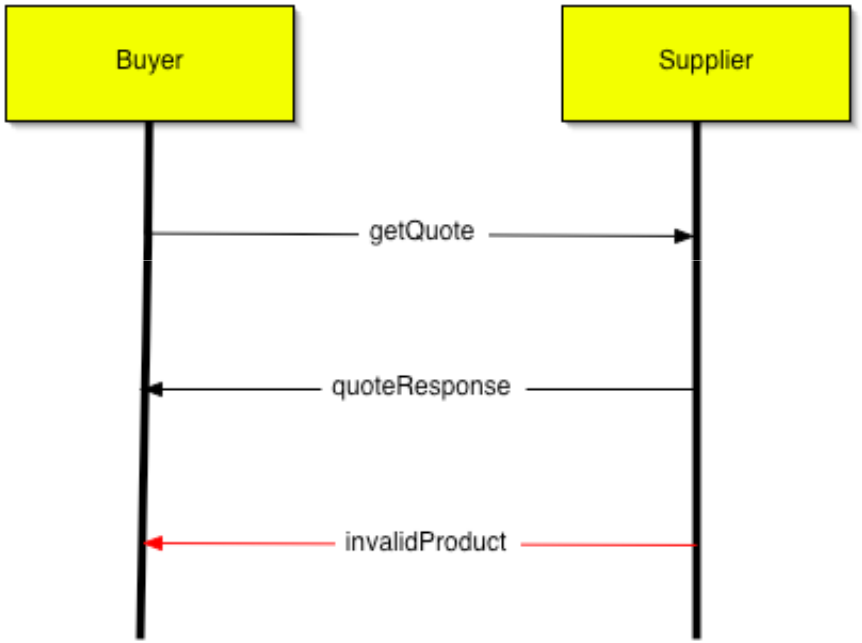
Typing correct choreographies

- Framework: abstract prototypes of real web services
- Issues
  - composition,
  - Choreography
  - Coordination
- Goal: enforce desiderable properties by inspecting the code
  - Only correct programs could be compiled and executed
- Technology: type systems

Sessions

- Communication of WS abstracted in notion of session
  - WS-CDL channel
- A channel is bi-directional
- !getQuote | ?getQuote

  buyer:Output Supplier:Input



Typable WS do not generate errors

- Error: two input/output on the same session channel
  - !getQuote | !getQuote
  - ?getQuote | ?getQuote
- Property enforced during the whole execution
- Prevents programming errors due to composition of two or more web services in a choreography

Coreografia d'esempio

Acquisto formula volo+noleggio auto (**1 Client 3 Server**)

- Cliente invia dati a Web Service (WS) compagnia area
- WS airline inoltra dati cliente a WS autonoleggio
- WS autonoleggio comunica importo a WS airline
- WS airline inoltra importo a WS carta credito
- WS carta credito chiede nr carta a cliente
- Cliente fornisce numero a WS carta credito
- WS carta credito dà ok a WS airline
- WS airline dà ok a WS autonoleggio e Cliente

Stronger Properties

Progress

- Choreographies engaging a session always complete it
- Acquisto formula volo+noleggio auto (**1 Client 3 Server**)
  - Il client chiede quotazione volo+noleggio
- Garanzia che
  1. Client riceve quotazione
  2. Se client accetta quotazione riceve richiesta carta credito
  3. Se client immette carta credito riceve risposta
  4. Se risposta = ok volo+noleggio riservato
- *Prototipo, accade cosi' in realta' ?*

Stronger Properties

Deadlock avoidance

- Instance of WS in choreographies never go stuck
- Avoid circolarity
  - WS1 waits WS2 waiting for WS3 waiting for WS1

Acquisto formula volo+noleggio auto (**1 Client 3 Server**)

- Web Service Compagnia area sempre pronto a ricevere richieste
  - Il WS non e' bloccato in qualche computazione
- Se noleggio invia importo direttamente a WS carta credito, WS Compagnia Area bloccato *per sempre* in attesa risposta

Current Research

- Delegation of sessions while preserving correctness #
  - Delegation: a ftp web service may accept requests and dynamically delegate to threads
  - Sessions “move” through the internet
  - Error avoidance need to be enforced in the presence of mobility
- Automatic translation of choreographies into orchestrations *
  - Typed choreographies map to correct orchestrations
  - Result: reasoning at the global level is preserved locally
- Automatic translations of abstract orchestrations into JAVA API §

 # University of Lisbon

** University Queen Mary and Imperial College of London,*
§ Universita di Firenze

References

- Web Services. Concepts, Architectures and Applications. G.Alonso, F.Casati, H.Kuno and V.Machiraju. Springer, 2004.
- SOAP specification. http://www.w3.org/TR/SOAP
- Understanding WSDL. Aaron Skonnard. www.msdn.microsoft.com
- UDDI specification. http://uddi.org
- XML Messaging. Dirk Reinshagen. http://www.javaworld.com
- WS Choreography Working group at http://www.w3.org
- EU-IST Software Engineering for Service-Oriented Overlay Computers (SENSORIA) publications. http://www.sensoria-ist.eu
- List of public web services. http://www.xmethods.com
- Apache Axis 2. http://ws.apache.org/axis2/

Fine.

Marco Giunti