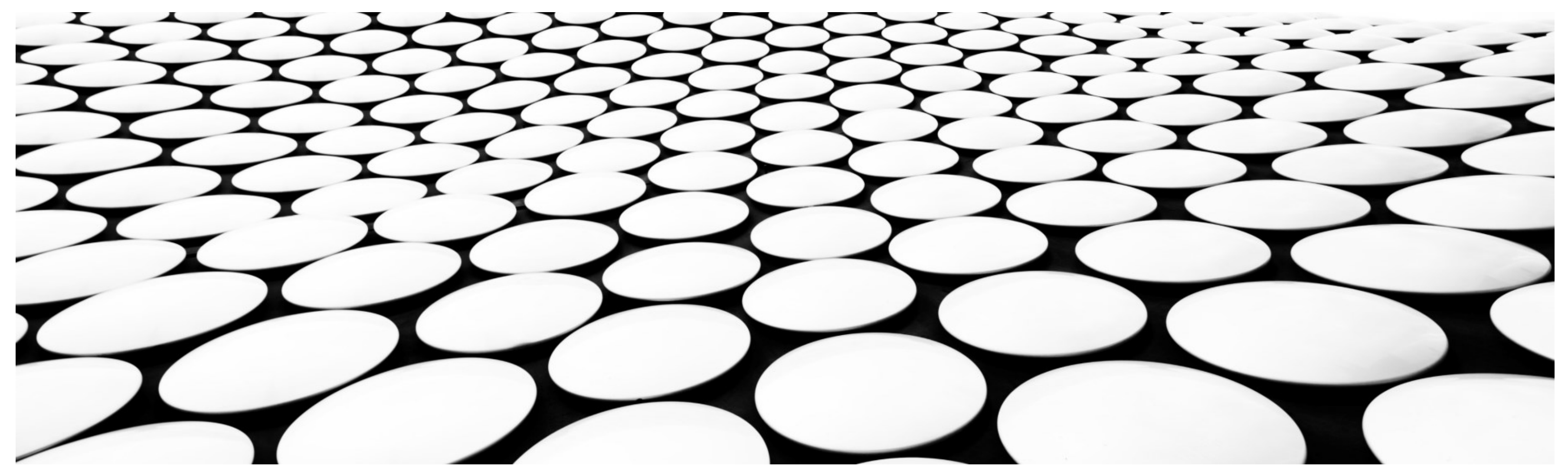
WEB SERVICES: XML & SOAP

VINCENZO CALABRÒ



AGENDA

- Introduzione
- XML
- La sintassi di XML
- Validazione di documenti XML
- Aspetti avanzati
- Remote Procedure Call (RPC)
- XML-RPC
- Web services: SOAP
- SOAP
- WSDL

Programmable Web

- Simile al web
  - La più grande differenza è che usa documenti XML o simili (non HTML + banner + loghi)
  - Non è per forza orientato agli utenti umani
  - Può fornire input a software
- Si basa su HTTP e XML
  - Alternative a XML: HTML, JSON, plain text, binary
- Se non si usa HTTP non si può parlare di programmable web

Programmable Web: Building blocks

- HTTP è il punto comune di tutti

- Method information
  - Come il client comunica con il server i suoi bisogni
  - HTTP method vs application-specific method name (nella richiesta HTTP o nell'URI)

- Scoping information
  - Come il client dice al server su quale parte dei dati deve agire
  - Nel percorso dell'URI vs nell'entity-body

Programmable Web: Architetture

- RESTful, Resource-Oriented Architecture
  - Method information nell'HTTP method
  - Scoping information nell'URI
  - Esempi di applicazioni RESTful: Amazon S3, Yahoo!'s web service, siti web statici…
- RPC-style architecture
  - Accetta un envelope con dati dal client e risponde con un envelope simile
  - Method e scoping information interni all'envelope
  - HTTP envelope è il più usato, SOAP un altro molto usato
  - Come per funzioni tradizionali ogni applicazione ha il suo vocabolario
  - Esempi: XML-RPC, servizi SOAP

Programmable Web: Architetture

- REST-RPC hybrid architecture
  - Servizi a metà tra i due paradigmi discussi
  - Illusione ottica che capita quando servizi RPC-style usano HTTP base come envelope
  - Esistono casi in cui alcuni servizi RPC-style hanno elementi RESTful "per sbaglio" (nessuno sviluppa architetture ibride from scratch)
    - Ad esempio faccio una HTTP GET e in quel caso voglio davvero ottenere dell'informazione
    - Scoping information sono nell'URI (ad es., quando faccio una ricerca)
  - HTTP è solo un envelope conveniente, usato in una modalità sovrapposta a REST
    - Molti web service read-only sviluppati RPC-style sono RESTful e resource-oriented

Programmable Web: Tecnologie

- HTTP
- URI
- XML-RPC
- SOAP
- WS-*
- WSDL
- UDDI

XML

Perchè XML

- 1960-1980 Infrastruttura Internet
- 1986 SGML (Standard Generalized Markup Language) per definire e rappresentare documenti strutturati
- 1991 Introduzione WWW e HTML
- 1991 Business adottano la tecnologia WWW; espansione nell’utilizzo Internet
- 1995 Nuovi tipi di business basati sulla connettività delle persone in tutto il mondo e sulla connettività delle applicazioni costruite attraverso il software di diversi provider (B2C, B2B)

Bisogno urgente di un nuovo e comune formato dei dati per internet

Perchè XML

- Necessità di regole semplici e comuni semplici da comprendere per persone con diversi background (come HTML)

- Capacità di descrivere risorse Internet e loro relazioni (come HTML)

- Capacità di definire struttura delle informazioni per diversi domini di business (*non come* HTML, come SGML)

Perchè XML

- Formato abbastanza formale per computer e abbastanza chiaro per essere leggibile da uomini (come SGML)

- Regole semplici per permettere uno sviluppo software semplice (non come SGML)

- Supporto per diversi linguaggi naturali (non come SGML)

Cos'è XML

- XML = Extensible Markup Language

A set of rules for defining and representing information as structured documents for applications on the Internet; a restricted form of SGML

T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds.), Extensible Markup Language (XML) 1.0, W3C Recommendation 10- February-1998, http://www.w3.org/TR/1998/REC-xml-19980210/.

T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler (Eds.), Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006/.

Cos'è XML

- EXtensible Markup Language
- XML è un linguaggio di marcatura (markup) come HTML
- XML è stato progettato descrivere i dati
- I tag di XML non sono predefiniti
- XML usa DTD e/o Schemi per descrivere i dati
- XML insieme ad un DTD o uno Schema è progettato per essere auto-descrittivo

Cos'è XML

- Regola 1: informazioni rappresentate in unità chiamate documento XML
- Regola 2: un documento XML contiene uno o più elementi
- Regola 3: un elemento ha un nome, è denotato nel documento con un markup esplicito contiene altri elementi e può essere associato con attributi
- E molte altre regole...

XML non fa niente!

- XML è progettato per non fare niente!
- XML è stato creato per strutturare, memorizzare e trasmettere dati
- Esempio

```
<nota>
        <a>Gianni</a>
        <da>Monica</da>
        <titolo>Promemoria</titolo>
        <corpo>Ricordati di me questo week end</corpo>
</nota>
```

XML non fa niente !

- La nota ha una intestazione e un corpo, ha anche un mittente ed un destinatario
- Però continua a non fare niente!
- Abbiamo solamente delle informazioni *intrappolate* tra dei tag
- Qualcuno dovrà scrivere del software per spedire e ricevere il messaggio della nota

XML è gratis ed estensibile

- XML non ha tag predefiniti, si inventano
- I tag di HTML sono predefiniti e si possono usare solo quelli per scrivere documenti
- XML permette di definire i tag che ritiene necessari e la struttura di documento adeguata

XML è complementare a HTML

- XML non sostituisce HTML
- La tendenza è quella di rappresentare i dati con XML e mostrarli con HTML
- Una buona definizione di XML è : “**XML è uno strumento per trasmettere informazioni, indipendente dalla piattaforma, dal software e dall’hardware**”

XML può separare i dati da HTML

- Con XML i dati vengono memorizzati separatamente dai documenti HTML
- Solitamente quando si visualizzano dei dati con HTML i dati sono all'interno del documento HTML stesso
- Con XML possono essere memorizzati in file separati

XML può separare i dati da HTML

- Questa divisione permette di potersi concentrare sulla visualizzazione con la sicurezza che delle modifiche ai dati non richiederanno modifiche al layout HTML

- È anche possibile inserire dati XML all’interno di pagine HTML, tenendoli isolati

XML per lo scambio di dati

- Con XML è possibile scambiare dati fra sistemi incompatibili
  - Encoding di dati
  - Encoding di protocolli
    - Definizione di funzioni
    - Marshalling di argomenti
- Nel mondo reale diversi sistemi e diversi database contengono dati in formato non uniforme
- E' incalcolabile il tempo speso dai programmatori per fare dialogare sistemi incompatibili

XML per la condivisione

- Con XML semplici file di testo possono essere usati per condividere dati

- I dati memorizzati con XML sono in formato testo
  - XML fornisce un modo indipendente da hardware e software di condividere dati

- Ciò rende semplice la creazione di documenti di cui hanno necessità applicazioni diverse

XML per la memorizzazione

- Con XML i dati sono disponibili a un maggior numero di utilizzatori
- Altri client ed applicazioni possono accedere ai file XML come se fossero sorgenti di dati (database)

La sintassi di XML

Sintassi

- Le regole sintattiche di XML sono nel contempo semplici e rigide
- Sono semplici da imparare e ancora di più da usare
- È quindi molto facile creare software che possa leggere e manipolare file XML

Un documento d’esempio

- La sintassi è semplice ed auto-descrittiva

```
<?xml version=“1.0” encoding=“ISO-8859-1”?>
<nota>
        <a>Gianni</a>
        <da>Monica</da>
        <titolo>Promemoria</titolo>
        <corpo>Ricordati di passare a prendermi domani</corpo>
</nota>
```

Un documento d'esempio

- La prima riga, la dichiarazione XML, definisce la versione di XML e il tipo di codifica dei caratteri utilizzati nel file

- In questo caso il documento è conforme alla versione 1.0 di XML ed è codificato con lo standard ISO-8859-1 (latin/western europe)
  - <?xml version="1.0" encoding="ISO-8859-1"?>

Un documento d’esempio

- La riga successiva descrive l’elemento radice del documento (*note*)
- Le quattro righe che seguono descrivono gli elementi figli (*a*, *da*, *intestazione* e *corpo*) dell’elemento radice
- L’ultima riga definisce la fine dell’elemento radice

Tutti i tag devono essere chiusi

- In XML è illegale omettere il tag di chiusura
- In HTML tale obbligo non vale per tutti i tag
- Nell'esempio precedente la dichiarazione XML non è chiusa, infatti, essa non fa parte del documento XML stesso

Nomi dei tag

- I nomi dei tag XML sono *case-sensitive*
- HTML non è *case-sensitive*
- In XML il tag <Lettera> è diverso dal tag <lettera>

Annidamento dei tag

- Significa che ogni tag, deve avere al suo interno
  - L’apertura e la chiusura di un altro tag
  - Nessun altro tag

- Esempio
  - <i>testo vario</i> corretto!
  - <i>testo vario</i> errato!

Nodo radice

- Tutti i documenti XML devono avere un nodo radice
- Tutti gli altri elementi devono essere compresi nell'elemento radice
- Tutti gli elementi possono avere dei nodi figli
- Tutti i nodi figli devono essere correttamente annidati nel tag genitore

Attributi

- I valori di tutti gli attributi devono essere tra doppi apici
- Gli elementi XML possono avere degli attributi che devono essere in coppia attributo-valore
- Esempio
  - <nota data="01/02/2003"> corretto!
  - <nota data=01/02/2003> errato!

Gli spazi

- In XML gli spazi bianchi non vengono troncati

- In HTML la frase

  Ciao, il mio   nome è Pippo

  Verrebbe visualizzata con

  Ciao, il mio nome è Pippo

Il ritorno a capo

- In XML il CR/LF viene convertito in LF
  - In XML il ritorno a capo è sempre memorizzato con un LF
  - LF indica "l'andare a capo"

- Normalmente nei sistemi Windows viene usato il CR/LF, nei sistemi Unix/Linux viene usato LF e nei Macintoch il CR

Commenti in XML

- È possibile inserire righe di commento in documenti XML

- La sintassi di un commento è la seguente

```
<!-- Questo è un commento -->
```

XML è semplice

- XML non è niente di speciale
- XML è un documento di testo in cui ci sono dei tag completamente liberi
- Ogni software in grado di manipolare file di testo, è in grado di manipolare file XML

Elementi XML

- Gli elementi in un documento XML possono essere estesi per rappresentare più informazioni

- Riprendiamo l'esempio della nota ed immaginiamo di avere un'applicazione che produca un output come il seguente

MESSAGGIO

DA Monica A Gianni

Ricordati di passare a prendermi domani

Elementi XML

- Supponiamo ora che l'autore aggiunga un tag data per aumentare le informazioni

```
<nota>
<data>21-11-15</data>
<a>Gianni</a>
<da>Monica</da>
<titolo>Promemoria</titolo>
<corpo>Ricordati di passare a prendermi domani</corpo>
</nota>
```

Elementi XML

- L'applicazione andrà in errore ?

- No perché sarà comunque in grado di trovare i tag *a, da, corpo, intestazione e nota*
  - Produrrà il medesimo output

Relazioni tra elementi

- Gli elementi sono in relazione padre-figlio
  - L'elemento **radice** deve essere unico
  - Gli elementi che nascono da esso sono detti nodi **figli**
  - Nodi **figli** che hanno lo stesso nodo **padre** sono detti **fratelli** (sibling)

Contenuto degli elementi

- Un elemento XML può contenere
  - Altri elementi
  - Un contenuto semplice
  - Un contenuto misto
  - Nessun contenuto

Nomi degli elementi

- Il nome degli elementi deve seguire alcune semplici regole
  - Può contenere lettere, cifre ed altri caratteri
  - Non può iniziare con un numero o con un carattere di punteggiatura
  - Non può iniziale con le tre lettere XML
  - Non può contenere spazi

Nomi degli elementi

- Quando si inventano dei nomi, è bene seguire le seguenti regole dettate dal semplice buon senso
  - I nomi devono essere *descrittivi*
  - Cercare di evitare caratteri *ambigui* come '-' e '.' e ':'
  - I nomi non hanno lunghezza massima, ma non esagerare

Attributi XML

- Gli elementi XML possono avere degli attributi nel tag iniziale
- Forniscono maggiori informazioni sul tag
- Il valore dell'attributo deve essere compreso tra doppi apici oppure tra singoli apici
- Nel caso in cui nel valore ci siano dei doppi apici, allora il valore deve essere compreso tra singoli apici e viceversa

Elementi vs Attributi

- Si considerino i due brani di codice

```
<persona sesso="femmina">
<nome>Anna</nome>
<cognome>Rossi</cognome>
</persona>
```

```
<persona>
<sesso>femmina</sesso>
<nome>Anna</nome>
<cognome>Rossi</cognome>
</persona>
```

Elementi vs Attributi

- Non esistono regole che dicano quando usare i tag e quando usare gli attributi

- Tendenze
  - Evitare il più possibile di usare gli attributi
  - Usare gli attributi solo per delle istruzioni di controllo (simile ad HTML)

Problemi usando attributi

- Alcuni problemi che si possono incontrare usando gli attributi
  - Possono contenere un solo valore
  - Non sono estendibili in caso di future revisioni
  - Non possono descrivere strutture
  - Sono più difficili da manipolare da parte delle applicazioni

Validazione di documenti XML

Schema e DTD

- Sono documenti che definiscono la struttura di un documento XML
- Gli schemi sono a loro volta dei documenti XML
- I DTD hanno una sintassi un po' più complessa e sono obsoleti

XML Schema

- Standard per la validazione dei file XML
- Scritto in XML, definisce tipo e cardinalità degli elementi
- Basato sui concetti di tipi semplici e complessi

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://xml.netbeans.org/schema/travelXSD"
            xmlns:tns="http://xml.netbeans.org/schema/travelXSD"
            elementFormDefault="qualified">
    <xsd:complexType name="inputC">
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="origin" type="xsd:string"/>
            <xsd:element name="destination" type="xsd:string"/>
            <xsd:element name="passengers" type="xsd:int"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="outputC">
        <xsd:sequence>
            <xsd:element name="ticket" type="xsd:string"/>
            <xsd:element name="voucher" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="faultC">
        <xsd:sequence>
            <xsd:element name="fault" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="input" type="tns:inputC"/>
    <xsd:element name="output" type="tns:outputC"/>
    <xsd:element name="fault" type="tns:faultC"/>
</xsd:schema>
```

Validazione

- Un documento XML sintatticamente corretto è detto ***ben formato***
- Un documento XML che rispetta uno schema (o un DTD) è detto ***valido***
- Un documento valido è anche ben formato, ma non è sempre vero il viceversa

XML 1.0 fundamentals

- Documento ben formato (sintassi)
  - Sintatticamente corretto
  - Tutti i tag di apertura e di chiusura corrispondono
  - I tag vuoti utilizzano una sintassi XML speciale
  - Tutti i valori degli attributi sono racchiusi tra virgolette
  - Tutte le entità sono dichiarate

- Valido (semantica)
  - Ben formato
  - Il documento rispetta le definizioni e la struttura proposte nello schema relativo

Visualizzazione di XML

- XML non ha informazioni di visualizzazione

- Per visualizzare file XML si usa una delle seguenti tecnologie

  - CSS (Cascading Style Sheets)
    - http://www.w3schools.com/xml/xml_display.asp

  - XSL (eXtensible Stylesheet Language)
    - http://www.w3schools.com/xsl/default.asp

Parser

- Per leggere, modificare o creare un documento XML è necessario un parser XML
- Il parser fornisce un modello che supporta
  - Javascript, VBScript, Perl, VB, Java, C++ ed altri
  - W3C XML 1.0 e XML DOM
  - DTD e validazione
- Esempio : per creare un oggetto di tipo documento XML con JavaScript si usa la seguente
  - var xmlDoc = new activeXobject("Microsoft.XMLDOM")

Esempio

- Note.xml
- CD.xml
- Plant_catalog.xml
- Simple.xml
- http://www.w3schools.com/xml/xml_examples.asp

Aspetti avanzati

Namespaces

- Sono un metodo per evitare conflitti fra i nomi

- Siccome in XML non ci sono nomi di tag prefissati, capita spesso che due documenti abbiamo uno o più tag con lo stesso nome, ma con significato diverso

- Se i due documenti vengono uniti si avrà un errore di conflitto di nomi

Prefisso

- Il problema può essere risolto usando un prefisso per i tag

```
<a:auto>
<a:marca>FIAT</a:marca>
<a:nome>Stilo</a:nome>
</a:auto>
```

```
<p:persona>
<p:nome>Mario</p: nome>
<p:cognome>Rossi</p:cognome>
</p:persona>
```

Usare i namespaces

- Aggiungendo un’informazione in più otteniamo l’uso dei namespaces

```
<a:auto xmlns:a=“http://www.qualcheindirizzo/qualchepagina”>
<a:marca>FIAT</a:marca>
<a:nome>Stilo</a:nome>
</a:auto>
```

- L’uso dell’attributo ha dato al tag auto un nome qualificato associato a un namespace

L’attributo namespace

- Deve essere messo nell’elemento di un documento e ha la seguente sintassi

  xmlns:prefisso_namespace=“nomespazio”

- La raccomandazione del w3c dice che il “nomespazio” deve essere un URI (Uniform Resource Identifiers)

- Quando in un elemento viene definito un namespace, esso sarà associato anche ai suoi figli

Uniform Resource Identifiers

- Un URI è una stringa di caratteri che identifica una risorsa su internet
  - L'URI più conosciuto e usato è l'URL (Uniform Resource Locator) che identifica un indirizzo internet

- Il parser NON userà tale indirizzo per cercare informazioni, ma solo per creare un namespace unico

Namespace di default

- Definire un namespace di default evita di usare un prefisso in tutti i nodi figli

- Ha la seguente sintassi

<element xmlns="namespace">

Caratteri di escape

- I caratteri che non sono legali per XML devono essere sostituiti con riferimenti a entità
- Se si mette un carattere ‘<‘ all’interno di un documento XML, avremo un errore perché il parser lo interpreta come l’inizio di un nuovo elemento

Caratteri di escape

- Sarà necessario usare una referenza, in questo caso ‘>’;

- Esistono cinque riferimenti predefiniti in XML

<	<	Minore di
>	>	Maggiore di
&	&	E commerciale
'	‘	Apice
"	“	Doppio apice

Caratteri di escape

- I riferimenti iniziano sempre con ‘&’ e finiscono sempre con ‘;’

- Solo il ‘<‘ e la ‘&’ sono strettamente illegali in XML, ma è una buona abitudine usare le referenze anche per gli altri caratteri

CDATA

- Tutto ciò che è inserito in un tag CDATA verrà ignorato dal parser

- Una sezione CDATA
  - Inizia con <![CDATA[
  - Finisce con]]>

- Una sezione CDATA non può contenere altre sezioni CDATA né le sequenze di caratteri che iniziano e terminano una sezione CDATA

Conclusioni

- Usare un editor che supporti diversi tipi di codifica
- Sapere sempre che codifica si sta usando
- Usare lo stesso tipo di codifica per tutti i file XML

Remote Procedure Call (RPC)

RPC – Obiettivo

- Fornire *distribution transparency*
  - Programmare come se non ci fosse distribuzione
    - Ci avvicineremo a questo obiettivo ma non lo raggiungeremo mai
  - “We argue that objects that interact in a distributed system need to be dealt with in ways that are intrinsically different from objects that interact in a single address space. These differences are required because distributed systems require that the programmer be aware of latency, have a different model of memory access, and take into account issues of concurrency and partial failure.” [1]

[1] S. C. Kendall, J. Waldo, A. Wollrath and G. Wyant: A Note on Distributed Computing

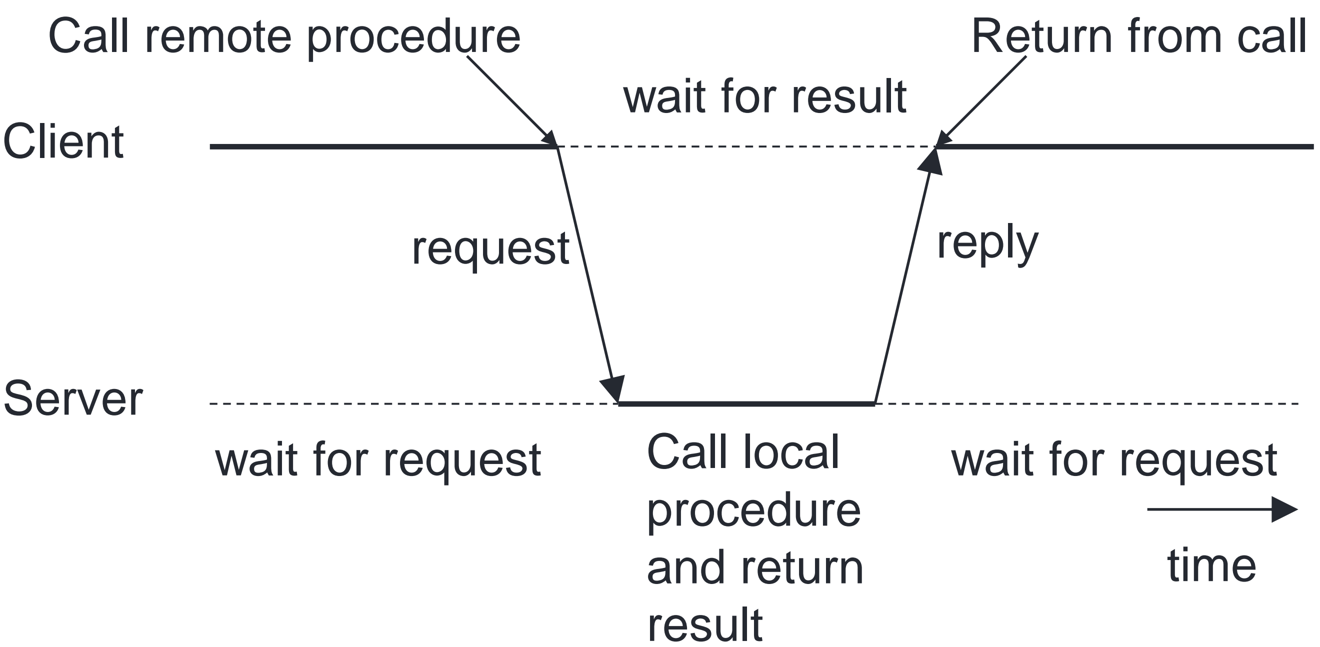
RPC - Definizione

- RPC si trova al di sopra del livello di trasporto (livelli di presentazione/sessione)
  - Nasconde la rete di comunicazione dal programmatore applicativo (astrazione)
- Protocollo di richiesta-risposta
  - Messaggio per invocare una procedura (Richiesta)
  - Messaggio contenente il risultato (Risposta)
- RPC si occupa di
  - Marshalling & unmarshaling dei dati (parametri e risultati)
  - Gestione delle diverse rappresentazioni dei dati
  - Indirizzamento

Architettura

- Client-side proxy (stub)
  - Implementa l’interfaccia della procedura remota lato client
  - Il client chiama l’interfaccia localmente (-> trasparenza)
  - Procedura invocata sulla macchina remota
- Server-side proxy (stub o skeleton)
  - Implementa l’interfaccia della procedura remota lato server
  - Richieste in ingresso sono distribuite a questa interfaccia localmente
    - Il server non si rende conto che la chiamata è remota
- Risultato: quasi distribution transparency …
  - … se non ci sono fallimenti

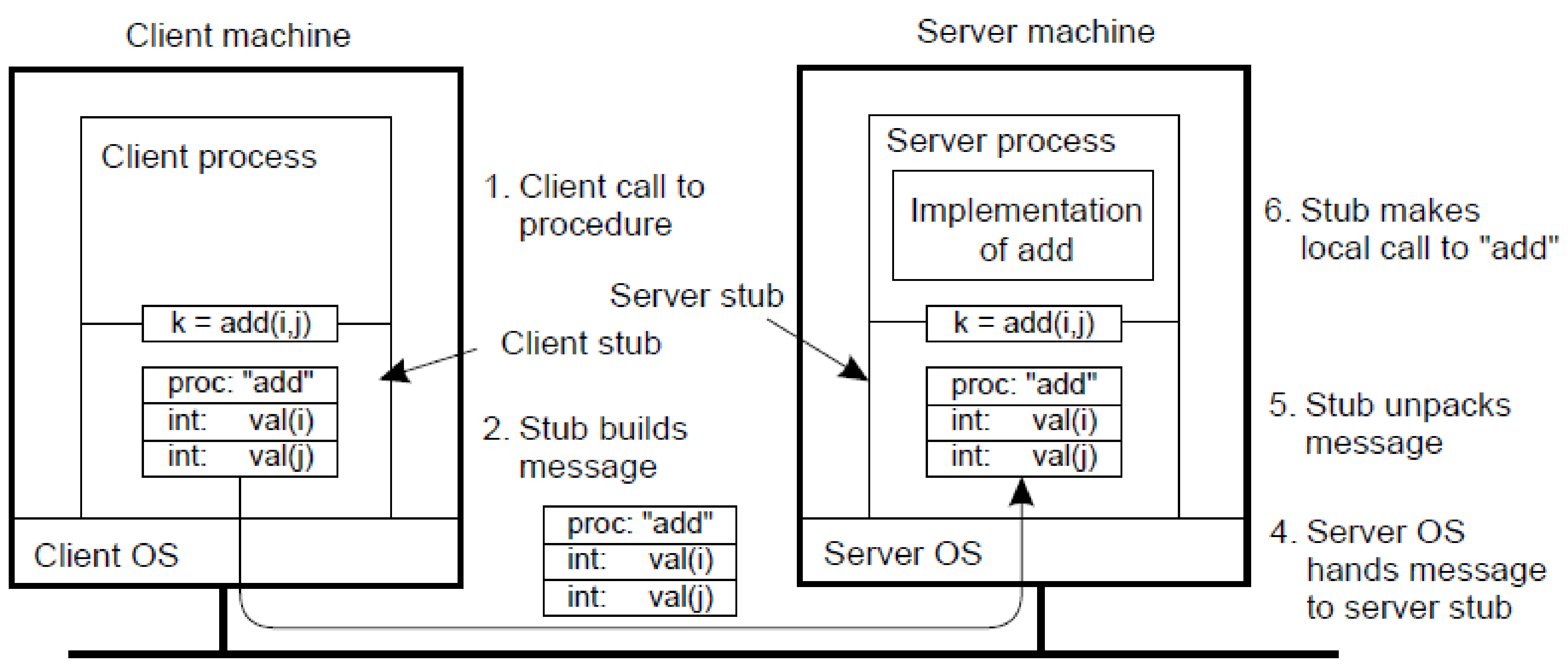
Flusso di esecuzione



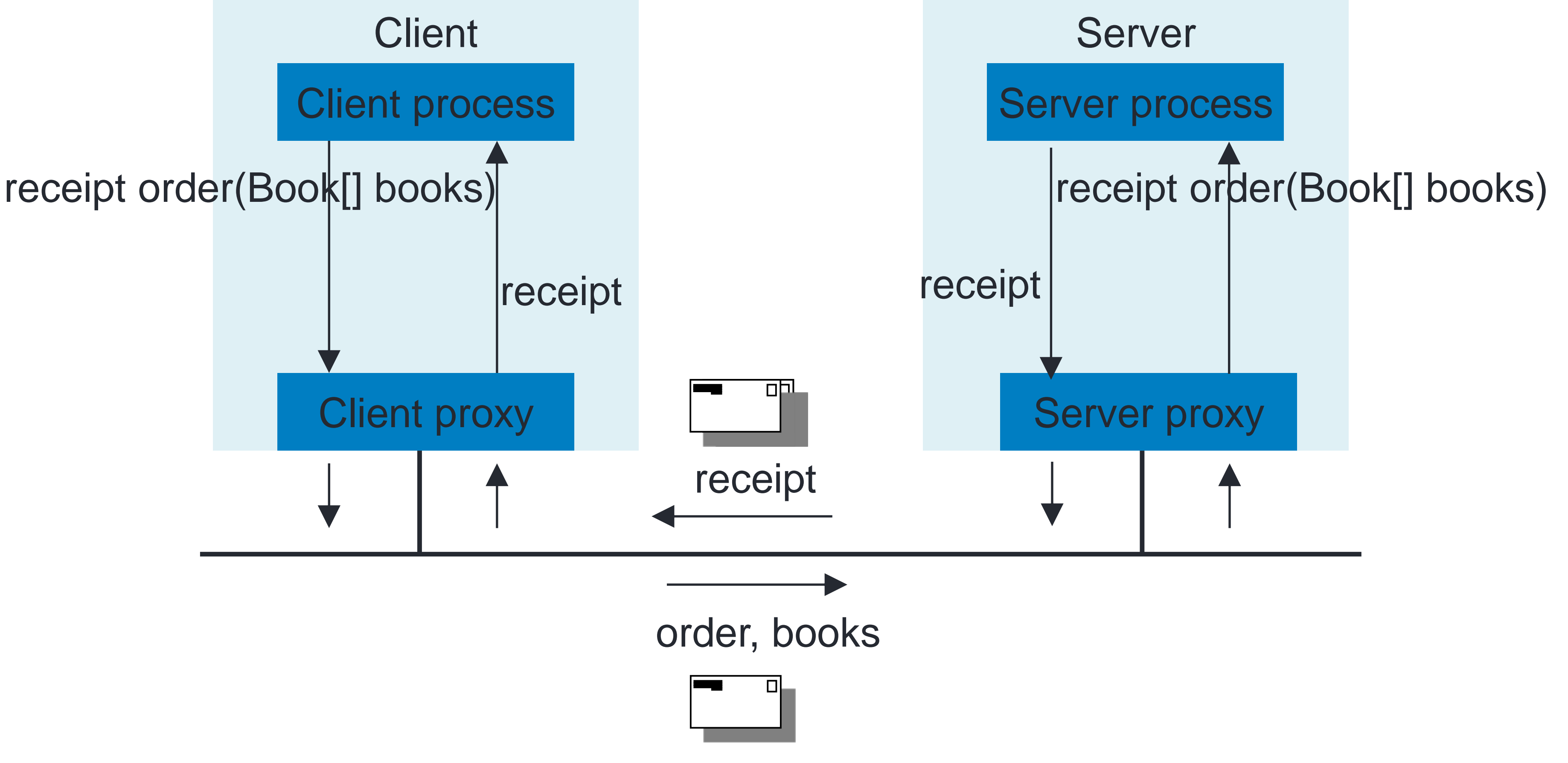
Flusso di esecuzione

- Client chiama la procedura sul client-side proxy locale
- Client proxy prepara (marshal) i parametri e invia il messaggio al server proxy
- Server proxy traduce (unmarshal) i parametri e chiama la procedura localmente
- Procedura viene eseguita e ritorna i risultati al proxy
- Server proxy prepara (marshal) il resultato e invia il messaggio al client proxy
- Client proxy traduce (unmarshal) i risultati e li ritorna al client

Flusso della richiesta



Esempio libreria



Marshalling / Unmarshalling

- Assemblaggio verso un formato esterno: Marshalling
- Disassemblaggio da un formato esterno: Unmarshalling
- Tipi di dato complessi devono essere serializzati
  - Liste, Struct, Grafi, …
- Quando il formato dei dati differiscono
  - Usare un formato esterno comune
    - Il sender transforma verso il formato esterno
    - Il receiver transforma dal formato esterno
  - Usare il formato del sender e "receiver makes it right"
    - Sender deve inviare un'indicazione del formato

Proxy Generation

- Bisogno di un tool per generare proxy

- Il tool non ha conoscenza
  - Dell'interfaccia
    - Nome procedure
    - Tipi dei parametri
    - Tipi di ritorno
    - Eccezioni
  - Tipi dei dati

Definizione delle interfacce

- Interfacce sono definite attraverso un Interface Definition Language (IDL)
  - Language-neutral
  - Di solito una sintassi C-style

- Proxy possono essere generate da IDL
  - Diversi proxy per diversi linguaggi di programmazione
    - Client in Java -> client proxy in Java
    - Server in C -> server proxy in C

File shop.IDL

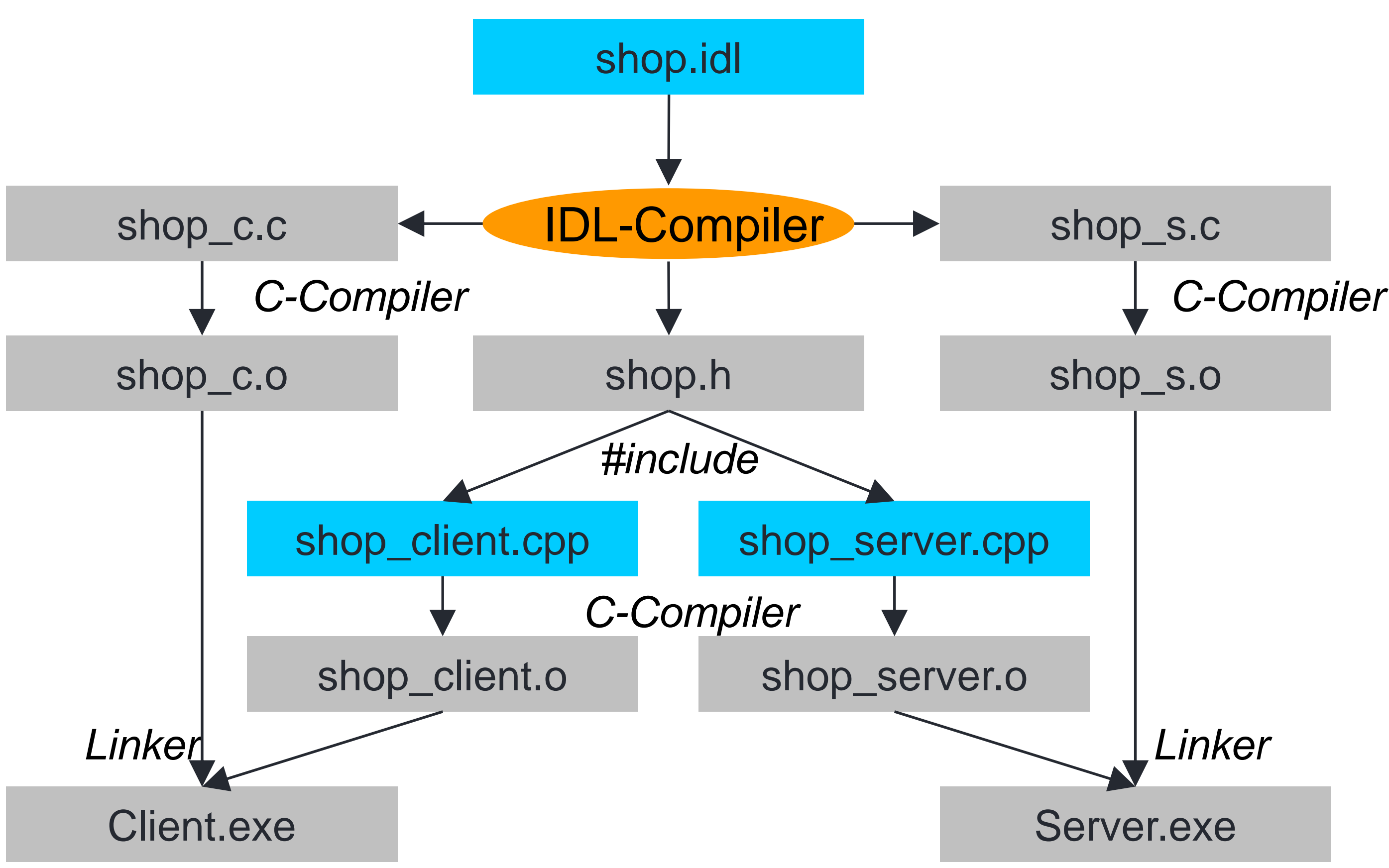
Interface Header

```
[ uuid (906B0CE0-C70B-1067-B317-00DD010662DA),
  version(1.0)
]
```

Interface Body

```
interface bookshop{

        struct Book{
            [string] unsigned char* name;
            long isbn;
        };

        struct Receipt{
            [string] unsigned char* bank;
            long accountnumber;
            int amount;
        };

        struct Receipt order([in] struct Book book);
        struct Book search([in,string]unsigned char* keyword);
};
```

Descrizione Struttura del progetto



IDL: Vantaggi e svantaggi

- Vantaggi
  - Indipendente dal linguaggio

- Svantaggi
  - Interfaccia generata può essere complicata
  - Sviluppatori devono imparare due linguaggi
  - Richiede un approccio top-down
    - Prima IDL, poi implementazione
    - Non si può usare un codice e strutture dati esistenti

XML-RPC

Introduzione

- XML-RPC fornisce un meccanismo basato su XML e HTTP per eseguire chiamate a funzioni/metodi attraverso la rete
  - XML usato per messaging (solo un piccolo vocabolario XML)
  - HTTP usato per passare informazioni dal computer client a quello server
- Nessuna nozione di oggetto
- Nessun meccanismo per includere informazioni che usano altri vocabolari
- Definiti all'inizio del 1998

Introduzione

- I sistemi distribuiti diventano tecnologia comune e nasce il bisogno di integrare applicazioni che risiedono su diversi computer (anche all'interno della stessa azienda)
- XML-RPC
  - Fornisce un tool semplice per connettere diverse parti in una rete privata
  - Fornisce un'interfaccia semplice verso un computer per altri computer sparsi sulla rete globale che voglio accedervi
  - XML semplice da inviare attraverso la rete (marshal)
  - XML ha supporto su larga scala
- Concetto simile a quello di pagina web come un'interfaccia human readable di un computer

Componenti

- Data Model
  - Un insieme di tipi per passare parametri, ritornare valori e messaggi di errore (fault)
  - Usato sia nella richiesta che nella risposta
- Struttura della richiesta
  - Una richiesta HTTP post contiene il method name e parameter information
- Struttura della risposta
  - Una risposta HTTP contiene i valori di ritorno o informazioni sui fault

Data Model

- Definisce sei tipi di dati base e due tipi di dati composti
- Sembra sufficiente per molte applicazioni

Basic Type names	Description
int or i4	32-bit integers
double	64-bit floating pt
boolean	true(1) or false(0)
string	ASCII text, usually support Unicode
dateTime.iso8601	Dates in ISO8601 format: CCYYMMDDTHH:MM:SS
base64	Binary defined as in RFC2045

Data Model: Esempi

- <int>27</int>
- <double>27.31415</double>
- <boolean>1</boolean>
- <string>Hello</string>
- <dateTime.iso8601>
  20020125T02:20:04
  </dateTime.iso8601>
- <base64>SGVsbG8sIFdvcmxkIQ==</base64>
- Tipi base sempre racchiusi in un elemento value
  - <value>
    <double> -12.45 </double>
    </value>

Data Model: Complex Type

- Tipi base possono essere combinati in due tipi complessi
  - Array, Struct

```
<value>
  <array>
    <data>
      <value><string>This </string></value>
      <value><string>is </string></value>
      <value><string>an </string></value>
      <value><string>array.</string></value>
    </data>
  </array>
</value>
```

Data Model: Arrays

- Array contiene un misto di diversi tipi

```
<value>
  <array>
    <data>
      <value><boolean>1</boolean></value>
      <value><string>Chan Tai-Man </string></value>
      <value><int> -91 </int></value>
      <value><double>0.1234</double></value>
    </data>
  </array>
</value>
```

Data Model: Array

- Array possono essere multi-dimensional
- E.g.

$$\begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

```
<value>
  <array>
    <data>
      <value>
        <array>
          <data>
            <value><int>10</int></value>
            <value><int>20</int></value>
          </data>
        </array>
      </value>
      <value>
        <array>
          <data>
            <value><int>30</int></value>
            <value><int>40</int></value>
          </data>
        </array>
      </value>
    </data>
  </array>
</value>
```

Data Model: Struct

- Struct composta da un contenuto non ordinato, identificato da un nome
- Nomi sono stringhe, nonostante non sia necessario racchiuderli in un elemento stringa
- Ogni elemento struct contiene una lista di elementi figli (member)
- Ogni elemento figlio è una coppia nome valore
- L'ordine degli elementi figli non è importante

Data Model: Struct

```
<value>
  <struct>
    <member>
      <name>givenName</name>
      <value><string>Tai-Man </string></value>
    </member>
    <member>
      <name>familyName</name>
      <value><string>Chan </string></value>
    </member>
    <member>
      <name>age</name>
      <value><int>27</int></value>
    </member>
  </struct>
</value>
```

First element

Second element

Third element

Data Model: Array e Struct

- Possiamo innestare struct/array all'interno di struct

```
<value>
  <struct>
    <member>
      <name>Name</name>
      <value><string>a</string></value>
    </member>
    <member>
      <name>attributes</name>
      <value><struct>
           <member><name>href</name>
          <value><string>http://ex.com</string></value>
        </member>
        <member><name>target</name>
          <value><string>_top</string></value>
           </member>
      </struct></value>
    </member>
  </struct>
</value>
```

Request Structure

- Richieste XML-RPC sono una combinazione di contenuto XML e header HTTP
  - HTTP header: un wrapper per passare una richiesta attraverso il Web
  - XML content: passa parametri e identifica le procedure da chiamare

```
POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.124.2
Content-Type: text/html
Content-Length: 169

        :
   XML statements
        :
```

Request Structure

- Statement XML contengono il method name e i parametric passati
  - Metodo da invocare
    - circleArea
  - Parametri di input
    - Un double
    - Un array

```xml
<?xml version="1.0"?>
<methodCall>
  <methodName>circleArea</methodName>
  <params>
    <param>
      <value><double>2.42</double></value>
    </param>
    <param>
      <value>
        <array>
          <data>
            <value><int>10</int></value>
            <value><int>20</int></value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodCall>
```

Response Structure

- Se richiesta ha successo – procedura trovata, eseguita correttamente – il risultato sarà ritornato attraverso la risposta al client
- In maniera simile a una richiesta, una risposta deve essere attaccata a un HTTP header per essere scambiata attraverso il Web

```
HTTP/1.1 200 OK
Date: Sat, 06 Oct 2004 23:20:04 GMT
Server: Apache.1.3.12 (Unix)
Connection: close
Content-Type: text/html
Content-Length: 124

          :
   XML statements
          :
```

Response Structure

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <double>2.4</double>
      </value>
    </param>
  </params>
</methodResponse>
```

- Ritorna un Double

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value><string>No such method!</string></value>
  </fault>
</methodResponse>
```

- Ritorna un errore
- Nessun errore standardizzato

Sviluppo con XML-RPC

- In applicazioni reali, non si programmano statement XML
- Si usa una libreria XML-RPC e si fanno alcune chiamate a funzione attraverso la libreria
- Possibile attraverso qualunque linguaggio di programmazione come Java
  - The Apache XML Project's Apache XML-RPC fornisce un package per integrare XML-RPC con Java in maniera semplice (http://ws.apache.org/xmlrpc/)
  - XML RPC per C/C++ (http://xmlrpc-c.sourceforge.net/)

Esempio

- Somma di due numeri
  - XML-RPC_esempio
  - Implementazione servizio XML RPC
  - Monitoraggio scambio dei messaggi con TCPMON

Vantaggi

- Semplicità
- Singolo formato per richiesta e risposta
- Protocolli standard
- Specifiche standard

Svantaggi

- XML-RPC permette di trasmettere pochi tipi di dati e si basa su messaggi di grandi dimensioni
- XML-RPC trasmette messaggi senza stato e soffre di bottleneck sul canale
- XML-RPC non supporta meccanismi di sicurezza e un modello basato su oggetti che sia robusto
- Dal punto di vista della rappresentazione, XML-RPC è lento, complicato, e incompleto confronto a meccanismi basati su linguaggi di programmazione nativi come Java

Conclusioni

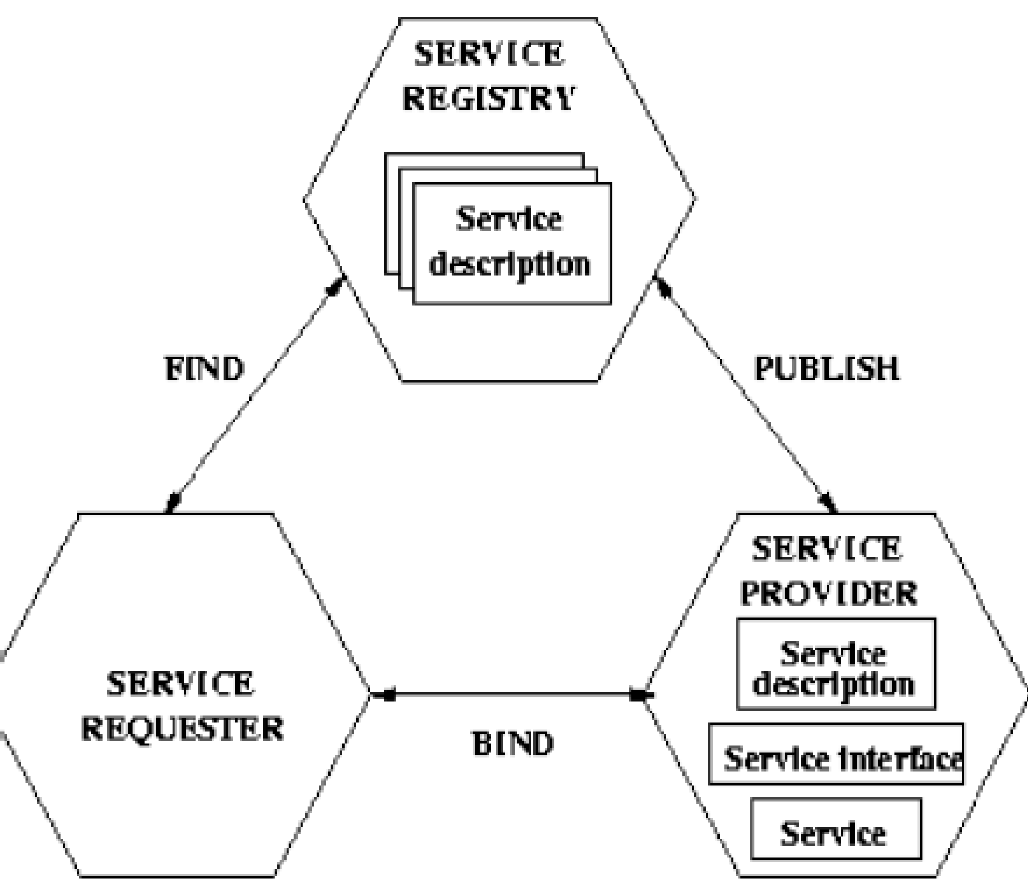
- XML-RPC fornisce un meccanismo basato su XML e HTTP per supportare chiamate a metodi/funzioni tramite la rete
- Client deve avere informazioni sulle funzioni rese disponibili dal server
- Il server non può pubblicizzare le sue funzioni né permettere al client di collezionarle automaticamente
- Mancanza di sicurezza

Web service: SOAP

Architettura web service

- Composta da tre elementi
  - Service requester: utente del servizio (client)
  - Service provider: entità che implementa il servizio e lo esegue per il requester (server)
  - Service registry: registro che elenca i servizi disponibili
    - Permette ai provider di pubblicizzare i loro servizi e ai requester di ricercare e richiedere i servizi stessi

Architettura web service

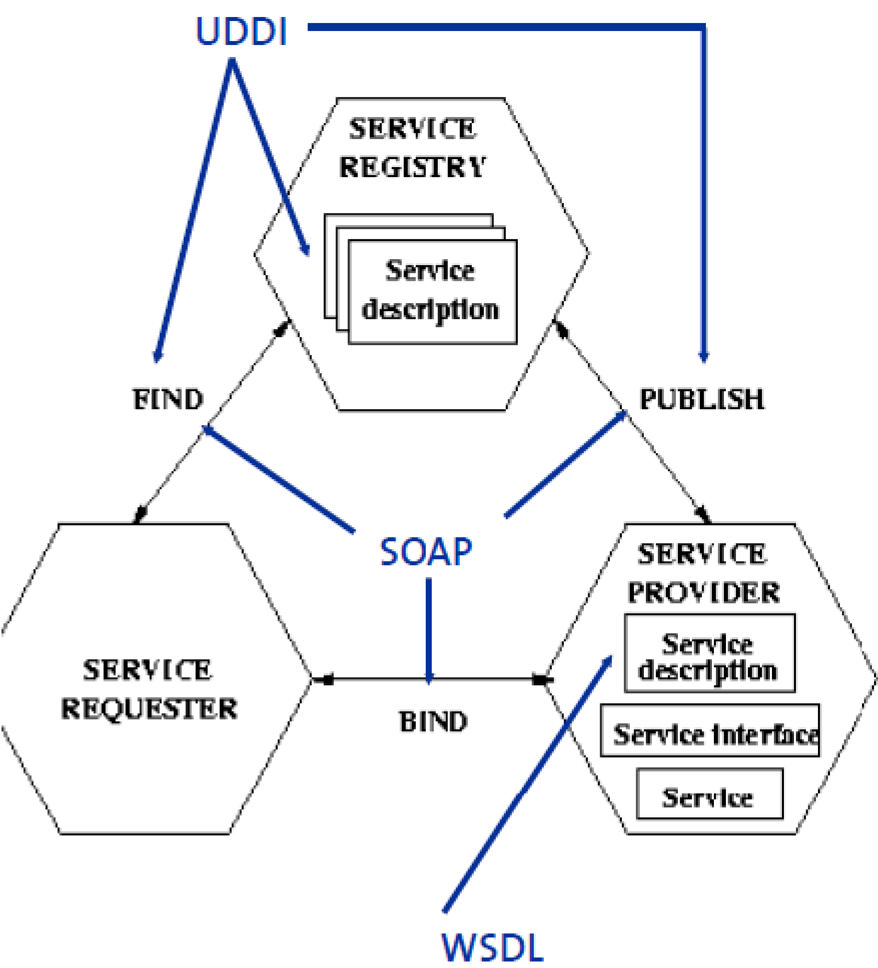


Standard

- Architettura web service
  - Si ispira all'architettura di piattaforme middleware esistenti
  - Implementa un paradigma client/server

- Definisce cosa può essere fatto con
  - SOAP (Simple Object Access Protocol)
  - UDDI (Universal Description and Discovery Protocol)
  - WSDL (Web Services Description Language)

Standard



Benefici

- Una delle differenze principle con middleware convenzionali è rappresentata dallo sforzo si stanzardizzazione del W3C che garantisce
  - Independenza dalla piattaforma (hardware, sistema operativo)
  - Utilizzo di infrastrutture di rete esistenti (HTTP)
  - Programming language neutrality (.NET parla con Java)
  - Portabilità tra tool di middleware di diversi vendor
- Web service sono componenti “loosely coupled” (debolmente accoppiati) che semplifica riutilizzo del software
- Tecnologie a web service sono componibili e possono essere adottate incrementalmente

Standard

Transport	HTTP, IIOP, SMTP, JMS		
Messaging	XML, **SOAP**		WS-Addressing
Description	XML Schema, **WSDL**		WS-Policy, SSDL
Discovery	**UDDI**		WS-MetadataExchange
Choreography	WSCL	WSCI	**WS-Coordination**
Business Processes	**WS-BPEL**	BPML	WSCDL
Stateful Resources	**WS-Resource Framework**		
Transactions	WS-CAF		**WS-Transactions** WS-Business Activities
Reliable Messaging	**WS-Reliability**		**WS-ReliableMessaging**
Security	**WS-Security** **SAML, XACML**		WS-Trust, WS-Privacy **WS-SecureConversation**
Event Notification	WS-Notification		WS-Eventing
Management	**WSDM**		**WS-Management**
Data Access	OGSA-DAI		SDO

SOA vs WS

- Web service considerano
  - Standardizzazione
  - Interoperabilità
  - Integrazione tra sistemi distribuiti, eterogenei

- Service Oriented Architecture considera
  - Un design del software su larga scala
  - Software Engineering
  - Architettura dei sistemi distribuiti

- SOA può essere implementata anche senza web service (più difficile) e introduce alcuni cambiamenti radicali al software:
  - Indipendenza dal linguaggio (quello che conta è l'interfaccia)
  - Interazione event based (non è un modello sincrono)
  - Scambio basato su messaggi (no RPC)
  - Composizione e orchestrazione

SOA vs WS

- SOA non è un concetto nuovo
  - Definito da Sun verso la fine degli anni novanta per descrivere Jini
  - "It is an environment for dynamic discovery and use of services over a network"
- Web service prendono il concetto di servizio introdotto da Jini e lo implementano come servizio rilasciato sul web tramite tecnologie come XML, WSDL, SOAP, e UDDI
- SOA fornisce una delle principali integrazioni e framework architetturale per computing environment complessi ed eterogenei
  - Aiuta le organizzazioni a organizzare i loro processi
  - Rende il business più efficiente
  - Permette di adattarsi a cambiamenti nei bisogni e nella competizione
  - Implementa il concetto di *software as a service*
- Web service sono lo standard principale per realizzare SOA

SOA vs WS

- SOA è uno stile architetturale per implementare applicazioni software che usano servizi disponibili in una rete come il web
- Supporta loose coupling tra componenti software facilitandone il riutilizzo
  - Applicazioni SOA sono basate su servizi
  - Un servizio è un'implementazione di una funzionalità di business ben definita
  - Un servizio può essere utilizzato da client di diverse applicazioni o processi di business
- SOA permette il riutilizzo di asset esistenti dove nuovi servizi possono essere creati a partire da un'infrastruttura IT esistente
  - Permette alle aziende di far fruttare investimenti precedenti permettendo di riutilizzare applicazioni esistenti
  - Promette interoperabilità tra applicazioni e tecnologie eterogenee

SOA vs WS

- Web service sono sistemi software progettati per supportare interazioni machine-to-machine interoperabili sulla rete

- Questa interoperabilità è ottenuta grazie a un insieme di standard basati su XML come WSDL, SOAP e UDDI

- Questi standard forniscono un approccio comune per definire, pubblicare e usare web service

SOAP

Problemi con invocazioni distribuite

- Come rendere l'invocazione a servizio parte di un linguaggio in modo trasparente

- Come scambiare dati tra macchine che possono usare rappresentazioni diverse per tipi di dati
  - Due aspetti principali: formato dei tipi di dati data (byte order in diverse architetture) e strutture dati (devono essere appiattite e ricostruite)

Problemi con invocazioni distribuite

- Come trovare il servizio giusto tra tanti servizi e server
  - Il client non deve necessariamente conoscere dove è il server o quale server fornisce quale servizio

- Come gestire errori nell'invocazione di servizi in maniera elegante
  - Server non raggiungibile o sovraccarico
  - Comunicazione non disponibile
  - Richieste duplicate

SOAP

- Per soddisfare i problemi precedenti viene definito SOAP
  - Protocollo basato su XML
  - SOAP è concettualmente semplice: RPC che usano HTTP
  - (client) traduce la chiamata RPC in un documento XML
  - (server) traduce un documento XML in una chiamata a procedura
  - (server) traduce la risposta della procedura in un documento XML
  - (client) traduce un documento XML in una risposta RPC
  - Usa XML per serializzare gli argomenti seguendo le specifiche SOAP

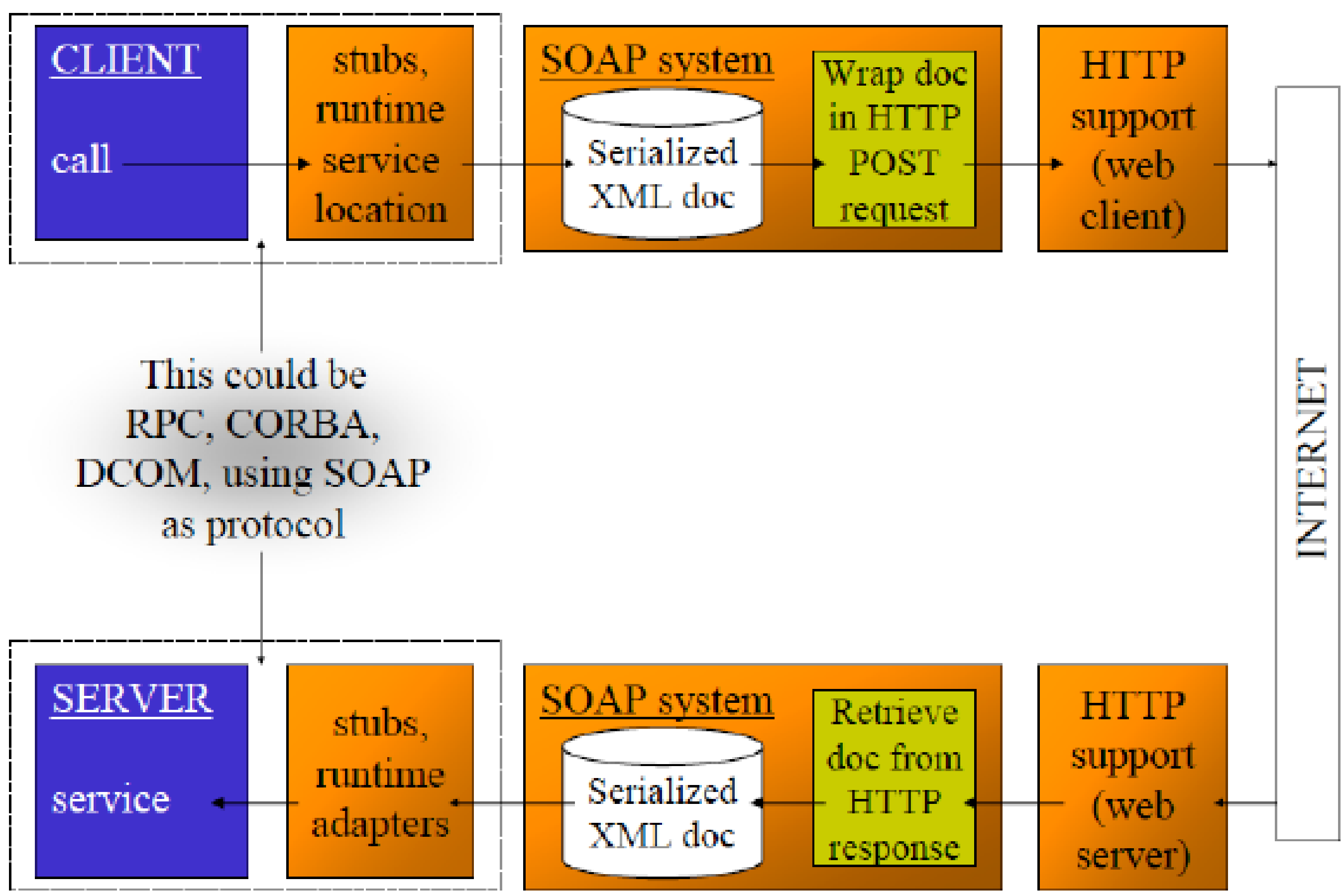
SOAP background

- SOAP pensato originariamente con un'infrastruttura minima per eseguire RPC attraverso Internet
  - Uso di XML come rappresentazione intermedia tra sistemi
  - Struttura dei messaggi semplice
  - Mapping verso HTTP per fare tunneling attraverso i firewall e per usare l'infrastruttura web

SOAP background

- SOAP diventa un veicolo generico per scambio di messaggi tra computer sulla rete Internet
  - Obiettivo: avere un'estensione facilmente integrabile al di sopra di piattaforme esistenti
  - Successivamente aperto per supportare interazioni che esulano da RPC e HTTP

Flusso di esecuzione



Storia

- W3C inizia a lavorare su SOAP nel 1999
  - Originariamente: Simple Object Access Protocol

- SOAP copre tre aree principali
  - Message construct: un formato dei messaggi per una comunicazione one-way, che descrive come i messaggi sono inseriti in un documento XML
  - Processing model: regole per processare messaggi SOAP e una semplice classificazione delle entità coinvolte nel processamento dei messaggi SOAP
    - Quale parti del messaggio devono essere lette da chi e come reagire in casi di contenuto incomprensibile
  - Extensibility model: come i costrutti dei messaggi base possono essere estesi con costrutti specifici dell'applicazione

Storia

- Protocol binding framework: permette a messaggi SOAP di essere trasportati usando diversi protocolli (HTTP, SMTP, …)
  - Un binding concreto per HTTP
  - Convenzioni su come trasformare chiamate RPC in messaggi SOAP
  - Come implementare l'interazione classica di RPC

Caratteristiche

- SOAP è “a lightweight protocol intended for exchanging structured information [...]”, “a stateless, one-way message exchange paradigm”
  - Definisce un formato generale di un messaggio e come processarlo
  - RPC implementato al di sopra delle specifiche dettate da “SOAP RPC representation”

Caratteristiche

- SOAP ≠ RPC: dalla Versione 1.1, SOAP astrae dal modello di programmazione RPC

- SOAP ≠ HTTP: dalla Versione 1.1, SOAP astrae dal protocollo HTTP per trasportare messaggi
  - HTTP è uno dei possibili mezzi di trasporto

Percorso del messaggio

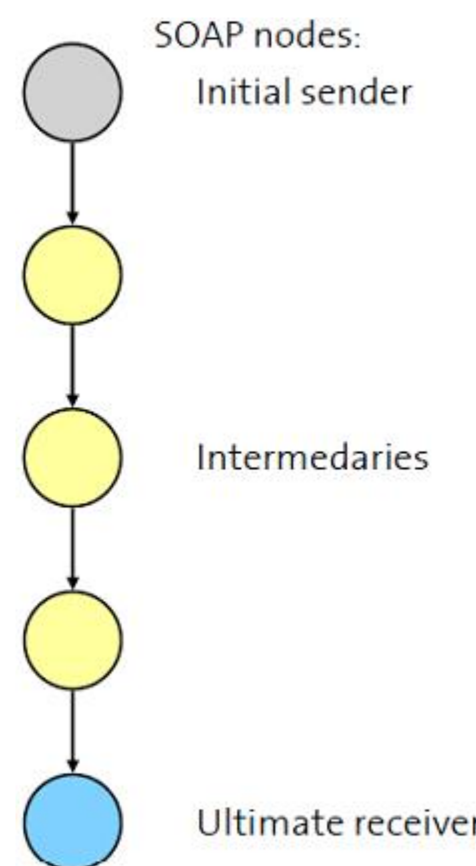
- Un messaggio SOAP attraversa diversi hop sulla strada tra mittente e destinatario

- Entità coinvolte nel trasporto sono chiamate nodi SOAP

- Nodi intermediari inoltrano (e manipolano) il messaggio

Percorso del messaggio

- Ogni nodo SOAP assume un ruolo che influenza il processamento del messagio



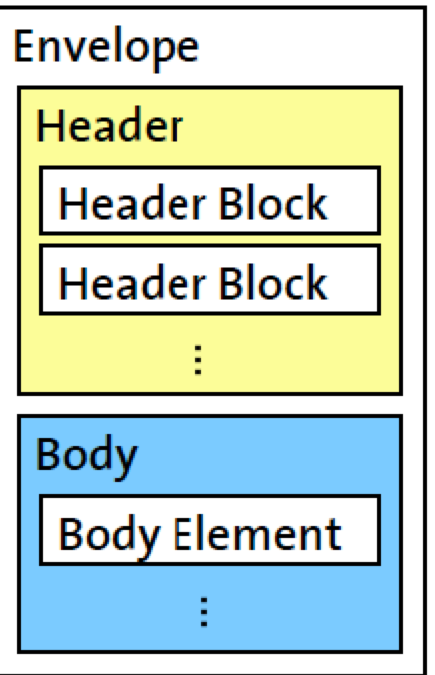
Struttura del messaggio

- SOAP message = SOAP envelope

- Envelope contiene
  - Header (opzionale): blocchi header indipendenti con metadata (security, transaction, session,…)
  - Body: blocchi con dati dell’applicazione

Struttura del messaggio

- SOAP non definisce la semantica dell'header e del body, ma solo la struttura del messaggio



Struttura del messaggio

```
<?xml version="1.0"?>
        <soap:Envelope
        xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
        soap:encodingStyle="http://www.w3.org/2001/12/soap-
        encoding"> <soap:Header>
...
...
</soap:Header>
<soap:Body>
...
...
<soap:Fault>
...
...
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

SOAP header

- Inteso come un punto generico per informazioni che non sono dipendenti dall'applicazione
  - L'applicazione potrebbe non sapere dell'esistenza dell'header
  - Utilizzo tipico dell'header
    - Coordination information
    - Identifier (ad es., per transazione)
    - Security information (ad es., certificato)

SOAP header

- SOAP fornisce meccanismi per specificare chi deve gestire gli header e cosa deve farne
- Include
  - Actor attribute: chi deve processare l’header block
  - Boolean mustUnderstand attribute: indica se è mandatorio processare l’header
    - Se l’header è diretto a un nodo (actor attribute), l’attributo mustUnderstand determina se è mandatorio processarlo
  - SOAP 1.2 aggiunge l’attributo relay (inoltra l’header se non processato)

Esempio

```
<?xml version="1.0"?>
<soap:Envelope
   xmlns:soap="http://www.w3.org/2001/12/soap-
   envelope"
   soap:encodingStyle="http://www.w3.org/2001/12/soap
   -encoding">
<soap:Header>
<m:Trans
   xmlns:m="http://www.w3schools.com/transaction/"
   soap:mustUnderstand="1">234</m:Trans>
</soap:Header>
... ...
</soap:Envelope>
```

SOAP body

- Body contiene i dati specifici dell’applicazione
  - Un elemento body è equivalente a un header block con attribute actor=ultimateReceiver e mustUnderstand=1
- A differenza degli header block, SOAP deve specificare il contenuto di alcuni elementi del body
  - Ad esempio, fornisce il mapping tra RPC e gli elementi del SOAP (RPC convention)
  - Fault entry (per riportare errori nel processamento del messaggio)

Esempio

XML name space identifier for SOAP serialization

XML name space identifier for SOAP envelope

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

    <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
            <symbol>DIS</symbol>
        </m:GetLastTradePrice>
    </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Esempio complessivo

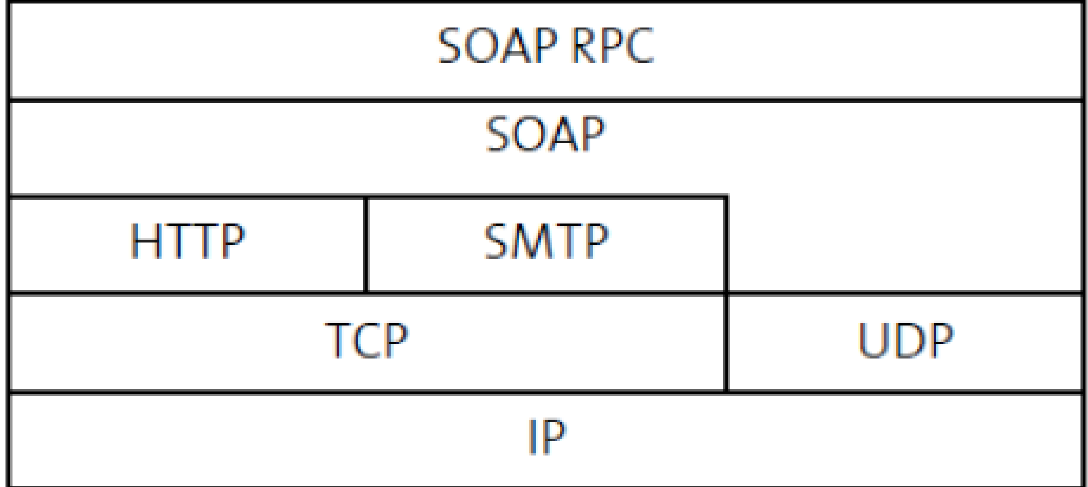
```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
      SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"/>
            <SOAP-ENV:Header>
                <t:Transaction
                    xmlns:t="some-URI"
                    SOAP-ENV:mustUnderstand="1">
                        5
                 </t:Transaction>
            </SOAP-ENV:Header>

            <SOAP-ENV:Body>
                <m:GetLastTradePrice xmlns:m="Some-URI">
                    <symbol>DEF</symbol>
                 </m:GetLastTradePrice>
            </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

From the: Simple Object Access Protocol (SOAP) 1.1. © W3C Note 08 May 2000

HTTP binding

- Messaggi SOAP possono essere trasferiti usando qualsiasi protocollo
- Un binding di SOAP verso un protocollo di trasporto è la descrizione di come un messaggio SOAP può essere inviato usando quell protocollo
- Binding specifica come messaggi di richiesta e risposta sono correlati
- Il framework di binding SOAP esprime linee guida per specificare il binding a un particolare protocollo

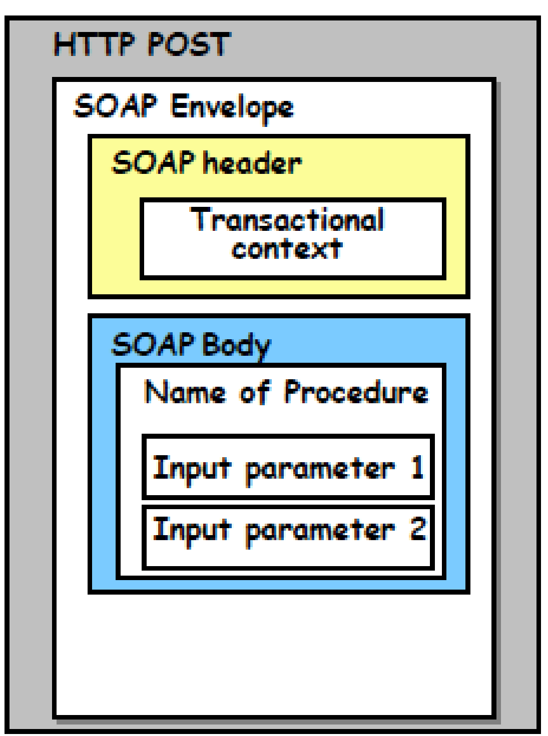
HTTP binding



HTTP binding

- Messaggi SOAP tipicamente trasferiti usando HTTP
- Il binding con HTTP definito nelle specifiche SOAP
- SOAP può usare GET o POST
  - Con GET, la richiesta non è un messaggio SOAP mentre la risposta è un messaggio SOAP
  - Con POST, sia richiesta che risposta sono messaggi SOAP

HTTP binding



Esempio di richiesta POST

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "GetLastTradePrice"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV=
  "http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle=
  "http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
   <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
   </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Esempio di risposta POST

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV=
  "http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle=
  "http://schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Body>
  <m:GetLastTradePriceResponse xmlns:m="Some-URI">
   <Price>34.5</Price>
  </m:GetLastTradePriceResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Conclusioni

- SOAP fornisce un meccanismo base per incapsulare messaggio in documenti XML
  - Mapping tra documenti XML e messaggi SOAP in richieste HTTP
  - Trasforma chiamate RPC in messaggi SOAP
  - Semplici regole su come processare un messaggio SOAP (regole maggiormente precise in SOAP v1.2)

Conclusioni

- Protocollo semplice inteso per trasferire dati da una piattaforma a un'altra

- Seguendo lo spirito *open*, specifiche e implementazioni sono spesso associate a RPC e HTTP

- SOAP trae vantaggio dalla standardizzazione di XML per risolvere i problemi di rappresentazione e serializzazione dei dati
  - Usa XML Schema per rappresentare dati e strutture
  - Usa XML per serializzare i dati prima di trasmetterli

WSDL

Introduzione

- Semplificando, WSDL è una versione XML di un IDL che copre aspetti relativi all'integrazione attraverso Internet e la complessità introdotta dai Web service
- Un IDL tradizionale non include
  - La locazione del servizio (implicita nella piattaforma e trovata attraverso binding statico o dinamico)
  - Diversi binding (tipicamente un IDL è associato a un protocollo di trasporto)
  - Set di operazioni (siccome un'interfaccia definisce un singolo punto di accesso e non c'è qualcosa di simile a una sequenza di operazioni coinvolte nello stesso servizio)

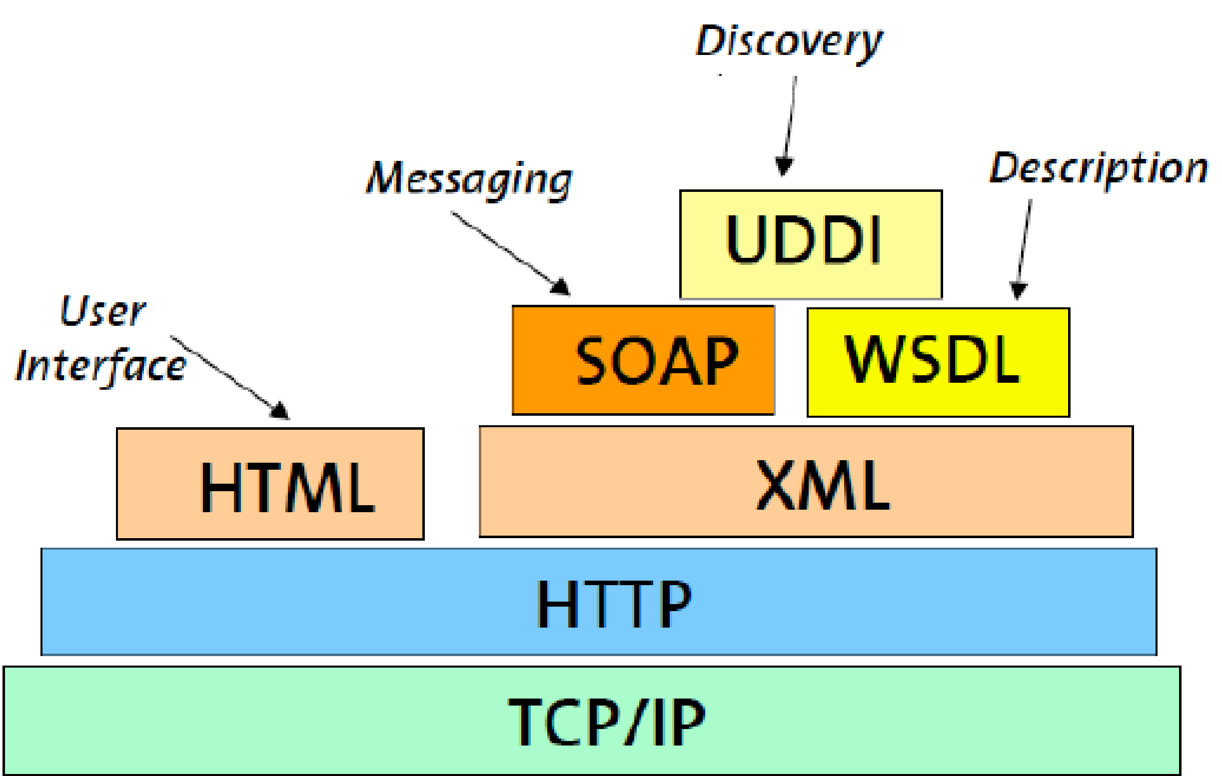
Introduzione

- Un IDL nei middleware tradizionali e nelle piattaforme di integrazione di applicazioni ha diversi obiettivi
  - Descrizione delle interfacce dei servizi proposti (ad es., RPC)
  - Fornisce una rappresentazione intermedia per gestire l'eterogeneità, fornendo un mapping tra i tipi di dati nativi verso una rappresentazione intermedia associata con IDL
  - Fornisce la base per lo sviluppo attraverso un compilatore IDL che fornisce stub e librerie necessarie per lo sviluppo dell'applicazione

Web Services Description Language (WSDL)

- WSDL Version v2.0 (Giugno 2007)
  - Definisce come descrivere le diverse parti che compongono un'interfaccia Web Service che rappresentano il data model del servizio (XML Schema)
  - Definisce i messaggi coinvolti nell'interazione con il servizio
  - Definisce le operazioni individuali composti di 4 pattern di scambio dei messaggi
  - Definisce l'insieme di operazioni del servizio
  - Definisce il mapping verso il protocollo di trasporto dei messaggi
  - Definisce la locazione dove il service provider risiede
  - Definisce le locazioni usate per accedere al servizio
- Include anche la specifica su come associare il WSDL con SOAP, HTTP (POST/GET) e MIME

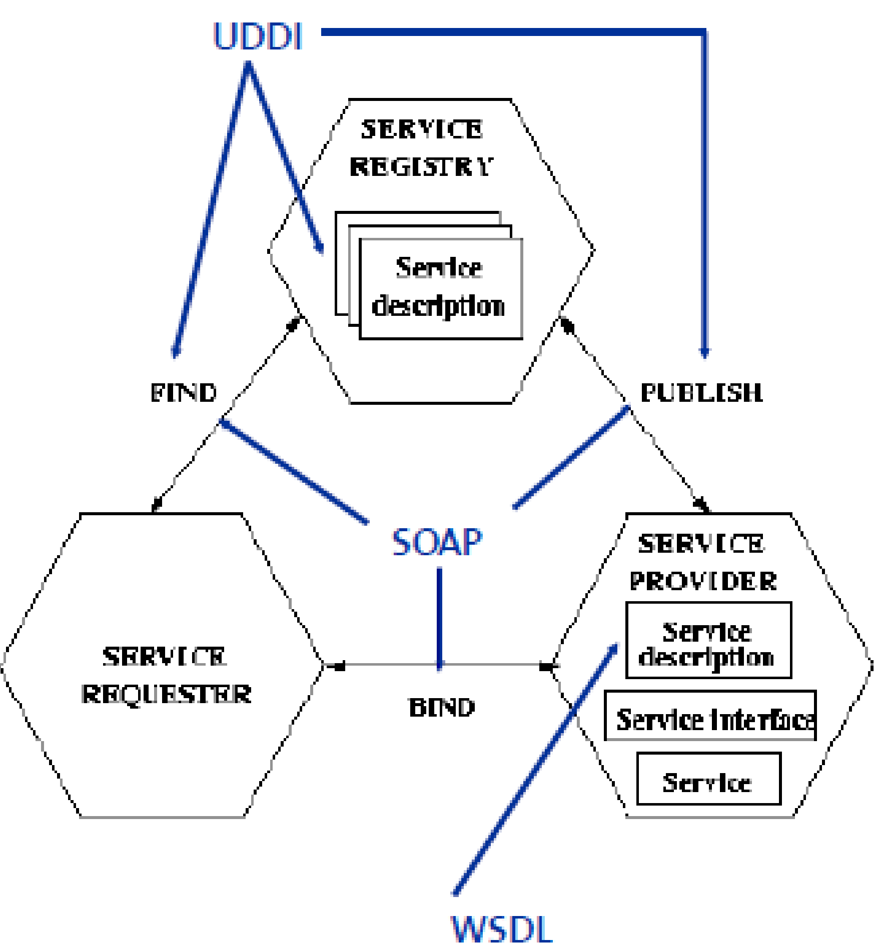
Layering



Ruolo di WDSL/UDDI

- Una volta che è possibile interagire con un service provider usando SOAP, è necessario
  - Descrivere il servizio (WSDL)
  - Scoprire la posizione del servizio (UDDI, Universal Description, Discovery and Integration)

Ruolo di WDSL/UDDI



WSDL: Data type

- Tipi in WSDL usati per definire il contenuto dei messaggi (normal message e fault message) che sarà scambiato come parte dell’interazione con il servizio
  - Tipi basati su XML Schema (struttura e tipi di dati)
  - Supporto mandatorio per tutti i processor di WSDL
  - Un elemento aggiuntivo può essere usato per definire schemi diversi da XML Schema

Esempio

```
<types>
  <xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://greath.example.com/2004/schemas/resSvc"
      xmlns="http://greath.example.com/2004/schemas/resSvc">

    <xs:element name="checkAvailability" type="tCheckAvailability"/>
    <xs:complexType name="tCheckAvailability">
      <xs:sequence>
        <xs:element  name="checkInDate" type="xs:date"/>
        <xs:element  name="checkOutDate" type="xs:date"/>
        <xs:element  name="roomType" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>

    <xs:element name="checkAvailabilityResponse" type="xs:double"/>

    <xs:element name="invalidDataError" type="xs:string"/>

  </xs:schema>
</types>
```

http://www.w3.org/TR/wsdl20-primer/

WSDL: Operation

- In WSDL 2.0, un’operazione è un insieme di messaggi e fault
- La sequenza e il numero di messaggi nell’operazione è determinate dal pattern di scambio dei messaggi
  - Operation può avere un RPC-like behavior, document oriented message exchange o (nella versione 2.0) set-and get-of attribute
  - Operation possono essere annotate con funzionalità e proprietà (ad es., reliability, security, routing)

```
<operation name="opCheckAvailability"
        pattern="http://www.w3.org/ns/wsdl/in-out"
        style="http://www.w3.org/ns/wsdl/style/iri"
        wsdlx:safe = "true">
    <input messageLabel="In"
          element="ghns:checkAvailability" />
    <output messageLabel="Out"
          element="ghns:checkAvailabilityResponse" />
    <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
</operation>
```

http://www.w3.org/TR/wsdl20-primer/

WSDL: Interface

- Un’interface corrisponde alla definizione astratta di un Web service
  - Astratta perchè non specifica informazioni sulla locazione del servizio o quale protocollo usare per invocarlo

- L’interface è semplicemente una lista di operazioni di un servizio che possono essere chiamate
  - Operation sono definite solo come parte di un’interfaccia

Esempio

```
<interface  name = "reservationInterface" >

  <fault name = "invalidDataFault"
          element = "ghns:invalidDataError"/>

  <operation name="opCheckAvailability"
          pattern="http://www.w3.org/ns/wsdl/in-out"
          style="http://www.w3.org/ns/wsdl/style/iri"
          wsdlx:safe = "true">
      <input messageLabel="In"
            element="ghns:checkAvailability" />
      <output messageLabel="Out"
            element="ghns:checkAvailabilityResponse" />
      <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
  </operation>

</interface>
```

http://www.w3.org/TR/wsdl20-primer/

Binding

- Un binding definisce il formato dei messaggi e i dettagli dei protocolli per le operazioni e i messaggi di un dato end point (Port Type nelle vecchie specifiche)

- Un binding corrisponde a uno specifico end point
  - Riferimento a operazioni e messaggi di un end point

- Un end point può avere diversi binding
  - Fornisce diversi canali di accesso allo stesso servizio astratto
  - Il binding è estendibile con elementi che permettono di specificare mapping di messaggi e operazioni a qualsiasi formato o protocollo di trasporto
    - WSDL diventa indipendente dal protocollo

Binding

```
<binding name="reservationSOAPBinding"
        interface="tns:reservationInterface"
        type="http://www.w3.org/ns/wsdl/soap"
        wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

  <fault ref="tns:invalidDataFault"
    wsoap:code="soap:Sender"/>

  <operation ref="tns:opCheckAvailability"
    wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

</binding>
```

http://www.w3.org/TR/wsdl20-primer/

End point

- Un end point specifica l’indirizzo del binding
  - Definisce come accedere al servizio usando un formato e protocollo specifici
- End point possono solo specificare un indirizzo e non devono contentere informazioni di binding
- L’end point è spesso specificato come parte di un servizio

Service

- Service raggruppa un insieme di porte e rappresenta la definizione completa del servizio come vista dall'esterno
- Un service supporta diversi protocolli e ha diversi binding
  - Accesso al servizio tramite un dato protocollo avviene a uno specifico indirizzo (specificato nel port di ogni binding)
  - Operation e message sono definiti nell'end point
  - Port parte dello stesso servizio
    - Potrebbero non comunicare tra loro
    - Considerate come alternative con lo stesso behavior (determinate dall'end point) ma raggiungibili attraverso diversi protocolli

Service

```
<service name="reservationService"
     interface="tns:reservationInterface">

   <endpoint name="reservationEndpoint"
             binding="tns:reservationSOAPBinding"
             address ="http://greath.example.com/2004/reservation"/>

</service>
```

http://www.w3.org/TR/wsdl20-primer/

Esempio

```xml
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions">

  <documentation>
    This document describes the GreatH Web service.  Additional
    application-level requirements for use of this service --
    beyond what WSDL 2.0 is able to describe -- are available
    at http://greath.example.com/2004/reservation-documentation.html
  </documentation>

  <types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://greath.example.com/2004/schemas/resSvc"
        xmlns="http://greath.example.com/2004/schemas/resSvc">

      <xs:element name="checkAvailability" type="tCheckAvailability"/>
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element  name="checkInDate" type="xs:date"/>
          <xs:element  name="checkOutDate" type="xs:date"/>
          <xs:element  name="roomType" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>

      <xs:element name="checkAvailabilityResponse" type="xs:double"/>

      <xs:element name="invalidDataError" type="xs:string"/>

    </xs:schema>
  </types>
```

```xml
  <interface  name = "reservationInterface" >

    <fault name = "invalidDataFault"
            element = "ghns:invalidDataError"/>

    <operation name="opCheckAvailability"
            pattern="http://www.w3.org/ns/wsdl/in-out"
            style="http://www.w3.org/ns/wsdl/style/iri"
            wsdlx:safe = "true">
        <input messageLabel="In"
              element="ghns:checkAvailability" />
        <output messageLabel="Out"
              element="ghns:checkAvailabilityResponse" />
        <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
    </operation>

  </interface>

  <binding name="reservationSOAPBinding"
          interface="tns:reservationInterface"
          type="http://www.w3.org/ns/wsdl/soap"
          wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

    <fault ref="tns:invalidDataFault"
      wsoap:code="soap:Sender"/>

    <operation ref="tns:opCheckAvailability"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

  </binding>

  <service name="reservationService"
       interface="tns:reservationInterface">

     <endpoint name="reservationEndpoint"
              binding="tns:reservationSOAPBinding"
              address ="http://greath.example.com/2004/reservation"/>

  </service>

</description>
```

Integrare con UDDI

- Specifiche UDDI in 3 versioni
  - Versione 1 definisce le basi per un business service registry
  - Versione 2 adattata per lavorare con SOAP e WSDL
  - Versione 3 ridefinisce il ruolo e l'obiettivo dei registry UDDI
    - Rafforza il ruoli di implementazioni private
    - Gestisce problemi di interazione tra registry pubblici e privati

Integrare con UDDI

- Originariamente pensato come un “Universal Business Registry” simile a search engine (ad es., Google)
  - Ricerca di servizi forniti da aziende in tutto il mondo
- Successivamente, diventa più pragmatico e si focalizza su interazioni B2B
- Presentato come “infrastructure for Web services”
  - Stesso ruolo di un name e directory service (binder in RPC) applicato a Web service
  - Usato soprattutto in ambienti vincolati: internamente in una azienda o federazioni di aziende (business partner)

Esempio

Per l’esercizio

- Web Container: Apache Tomcat
  http://tomcat.apache.org/

- Web Services / SOAP / WSDL engine: Apache AXIS2
  http://axis.apache.org/axis2/java/core/

- TCP SOAP Messages monitor: Apache TCPMon
  http://ws.apache.org/commons/tcpmon

Applicazione somma: Codice servizio

- Gestisce la parte di codice del business
- I metodi della classe sono i metodi forniti dal servizio
- Le connessioni tra classe e il Service Engine sono gestate dal file *service.xml*

```
/**
 * Secure Service implementation class
 */
public class SecureService {
        public int add(int a, int b) {
                return a+b;
        }
}
```

Applicazione somma: Codice servizio

- AXIS2 deploya servizi in file AAR (Axis ARchive)
- File AAR sono semplici archivi WAR che contengono il codice del servizio, service.xml, MANIFEST.MF, e i file di configurazione
- AAR sono create dal comando java **jar -cvf <service name>.aar ***
- Per deployare un servizio bisogna copiare il file AAR nella directory dei servizi di AXIS2 *webapps* subfolder

Applicazione somma: Service.xml

- Contiene nome del servizio, nome dei metodi e tipo dei parametri
- **messageReceiver** definisce la metodologia per lo scambio dei messaggi

```
<service name="SecureService">
   <description>Secure Service</description>
   <parameter name="ServiceClass"
        locked="false">SecureService</parameter>
   <operation name="add">
        <messageReceiver
        class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
   </operation>
</service>
```

Applicazione somma: Codice client

```
public class SecureServiceClient {

    public static void main(String[] args) throws Exception {

        ConfigurationContext ctx =
        ConfigurationContextFactory.createConfigurationContextFromFileSystem
        ("axis-repo", "null");

        SecureServiceStub stub = new SecureServiceStub
        (ctx,"http://localhost:8888/axis2/services/SecureService");

        ServiceClient sc = stub._getServiceClient();

        sc.engageModule("rampart");

        int a = 3;

        int b = 4;

        int result = stub.add(a, b);

        System.out.println(a + " + " + b + " = " + result);

    }

}
```

Applicazione somma: Codice client

- Codice client gestisce la richiesta del servizio
- Le configurazioni del client sono gestate dall'oggetto *ConfigurationContext*
- Classi Stub sono generate dal comando *wsdl2java* AXIS2 **wsdl2java -uri <service address>?wsdl -uw -p <package> -o <source directory>**
- Classi Stub replicano la signature dei metodi dei servizi e gestiscono la connessione tra client e service

Applicazione somma: Codice client

- Subfolder *axis-repo* contiene
  - **AXIS2.xml** configuration file: gestisce le configurazioni locali del client – livello di sicurezza, azioni fornite, utenti…
  - **modules**: contiene moduli locali aggiuntivi che sono utilizzati dal client – rampart, rahas, addressing
  - **keys**: contiene keyrings dell'utente che contiene le coppie di chiavi pubblica-privata

Applicazione somma: Messaggi

- Messaggi sono scambiati all'interno di envelope SOAP

```xml
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
      <soapenv:Body>
         <ns1:add xmlns:ns1="http://ws.apache.org/axis2">
            <ns1:args0>3</ns1:args0>
            <ns1:args1>4</ns1:args1>
         </ns1:add>
      </soapenv:Body>
   </soapenv:Envelope>
```

```xml
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
      <soapenv:Body>
         <ns:addResponse xmlns:ns="http://ws.apache.org/axis2">
            <ns:return>7</ns:return>
         </ns:addResponse>
      </soapenv:Body>
   </soapenv:Envelope>
```

Conclusioni

- XML come linguaggio di markup alla base di servizi web
- RPC e loro implementazione basata su XML (XML-RPC)
- Web service
  - SOAP
  - WSDL
  - UDDI

QUESITI?

vincenzocalabro.it