Service-Oriented Architecture
Parte 1

Programmazione Distribuita - A.A. 2020/2021

Biagio Cosenza
Dipartimento di Informatica
Università di Salerno
http://cosenza.eu/
bcosenza@unisa.it

Organizzazione della Lezione

- Service-Oriented Architecture
- I Web Services
  - Definizioni
  - I ruoli nei WS
- Gli standard
  - WSDL
  - SOAP
  - UDDI
  - Le specifiche di WS
- Conclusioni

Service-Oriented Architecture

- La tecnologia (middleware) a servizi è una tecnologia per l'integrazione di applicazioni distribuite
  - volta a risolvere problemi pragmatici di interoperabilità
  - basata su standard accettati dalla maggior parte dei produttori di software
  - che focalizza l'attenzione sul concetto di servizio
- L'architettura orientata ai servizi (SOA)
  - fornisce il contesto metodologico (e di business) in cui utilizzare al meglio le tecnologie basate su servizi
- I Web Services rappresentano la tecnologia più rappresentativa in questo ambito

Service-Oriented Architecture

- I servizi sono *self-contained processes*
  - deployati su una piattaforma di middleware standard (ad esempio Java EE)
  - che possono essere descritti, pubblicati, localizzati ed invocati attraverso la rete
- Ogni pezzo di codice oppure ogni componente di un'applicazione deployata su un sistema può essere trasformata in un servizio accessibile via rete
- I servizi riflettono un nuovo approccio *service-oriented*, basato sull'idea di comporre applicazioni *scoprendo* ed invocando servizi sulla rete piuttosto che scrivere nuove applicazioni
  - combinazione di funzionalità (già esistenti)

Service-Oriented Architecture

- I servizi eseguono funzioni che spaziano fra rispondere a semplici richieste fino all'eseguire complesse logiche di business
- I servizi sono scritti in modo di essere indipendenti dal contesto in cui sono usati
  - *service providers* e *consumers* sono debolmente accoppiati

Service-Oriented Architecture

- I servizi eseguono funzioni che spaziano fra rispondere a semplici richieste fino all'eseguire complesse logiche di business
- I servizi sono scritti in modo di essere indipendenti dal contesto in cui sono usati
  - *service providers* e *consumers* sono debolmente accoppiati
- A livello middleware il loosely coupling richiede che l'approccio service-oriented sia indipendente dalle specifiche tecnologie o sistemi operativi
- Servizi e composizione di servizi sono indipendenti dai linguaggi di programmazione utilizzati
  - invocazione attraverso l'utilizzo di *self-describing interface* e di standards

Service-Oriented Architecture

- Attraverso l'utilizzo di standard (linguaggi e protocolli)
  - gli sviluppatori possono accedere a sistemi ed applicazioni sulla rete sulla base di quello che fanno
  - indipendentemente da come realizzano le loro funzionalità e come sono state implementate

Service-Oriented Architecture

- Affinchè si possa parlare di Service-Oriented Architecture, i servizi devono:

Service-Oriented Architecture

- Affinchè si possa parlare di Service-Oriented Architecture, i servizi devono:
  - Essere neutrali rispetto alla tecnologia
    - devono poter essere invocati attraverso tecnologie *standard compliant* (protocolli, meccanismi di discovery) con standard accettati
    - meccanismi di invocazione
  - Loosely Coupled
    - nessuna conoscenza della struttura interna o convenzioni (*context*) sia a livello client che server
  - Supportare *location transparency*
    - le definizioni e informazioni di localizzazione devono essere memorizzate in un repository (ad esempio, UDDI) ed accessibili a diversi client
      - che possono invocarli indipendentemente dalla locazione

Service-Oriented Architecture

- I servizi possono essere forniti su una singola macchina o su un gran numero di dispositivi distribuiti su
  - una rete locale
  - una rete distribuita (incluso mobile e ad hoc networks)
- Un caso interessante è quello dei servizi che usano Internet come communication medium e open Internet standards
  - i Web Services

Web Services: Cosa sono

- Banalmente, qualcosa accessibile sul Web che offre un servizio (*service*)
  - pagine HTML, servlet, ..
- Applicazioni che possono essere implementate usando diverse tecnologie
  - Simple Object Access Protocol (SOAP) e Representational State Transfer (REST)
- Concetto chiave: *loosely coupled* (debolmente accoppiati)
  - niente è conosciuto dal client (*consumer*) del servizio circa la sua implementazione
    - linguaggio usato per svilupparlo, la piattaforma che lo esegue, ecc.

Web Services: il Paradigma

- Costrutti per supportare lo sviluppo di applicazioni distribuite che siano:
  - veloci
  - basso costo
  - di facile composizione con altri servizi simili
- Web Service: logica di applicazione disponibile in maniera automatica ed esposta su Internet
  - ogni porzione di codice e componente può essere trasformata in un web service
- Approccio orientato ai servizi: comporre un'applicazione scoprendo e invocando servizi disponibili sulla rete
  - invece di costruire nuove applicazioni o invocare applicazioni disponibili

Web Services: Requisiti

- Neutri rispetto alla tecnologia
  - ricondotti al minimo comun denominatore tecnologico per essere disponibile in tutti gli ambienti
- Debolmente accoppiati
  - nessuna conoscenza circa le strutture interne, oppure il contesto, da parte del client
- Trasparenza alla locazione
  - definizione e locazione disponibile attraverso registri pubblici, che siano accessibili da un client generico

Web Services: Definizione

- Dalla W3C:

Un WS è un sistema software progettato per supportare interazioni interoperabili tra computer su una rete. L'interfaccia è specificata in una descrizione analizzabile automaticamente (come Web Service Definition Language). L'interazione con gli altri sistemi avviene mediante messaggi SOAP (Simple Object Access Protocol), tipicamente attraverso una richiesta HTTP con serializzazione XML.

Web Services: Esempi

- Cosa possono essere i WS
  - Un task auto-contenuto (depositare fondi su un conto bancario)
  - Un processo di business (acquisto automatico di forniture per ufficio)
  - Un'applicazione (calcolo di previsioni su valore di azioni)
  - Una risorsa accessibile mediante servizi (backend di un DB)

Web Services: un esempio

- Un esempio di gestione di ordini di acquisto:
  - Servizio di ricevimento di ordini di acquisto
  - Risposta sulla base di: buon credito del cliente ("pagherà?"), disponibilità della merce, etc.
  - Servizi coinvolti:
    - servizio di credito (controlla la validità economica del cliente)
    - servizio di inventario
    - servizio di billing
    - servizio di spedizione

Web Services: non facciamo confusione

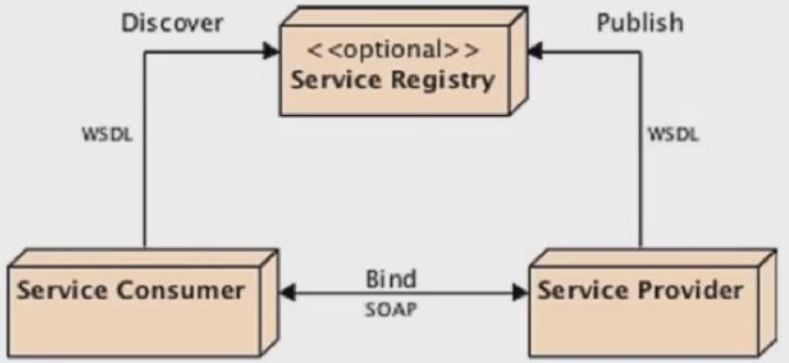
- Confronto tra Web services e servizi accessibili via Web
- I servizi accessibili via Web usano una interfaccia universale (browser) ma non sono necessariamente WS
- I WS sono utilizzabili sia automaticamente da altre applicazioni sia da utenti
- Necessario rivedere il concetto di *Software as a Service* (SaaS)

Software as a Service (SaaS)

- Concetto apparso per la prima volta con il modello software Application Service Provider (ASP)
- Application Service Providers
  - companies che combinano software e elementi infrastrutturali con servizi di business e professionali per fornire una soluzione completa ai customer sotto forma di servizio richiesto sulla base di una sottoscrizione
  - sono entità terze che eseguono il *deploy*, mantengono e gestiscono l'accesso a *packaged applications* e forniscono servizi basati sul software attraverso la rete a customer multipli

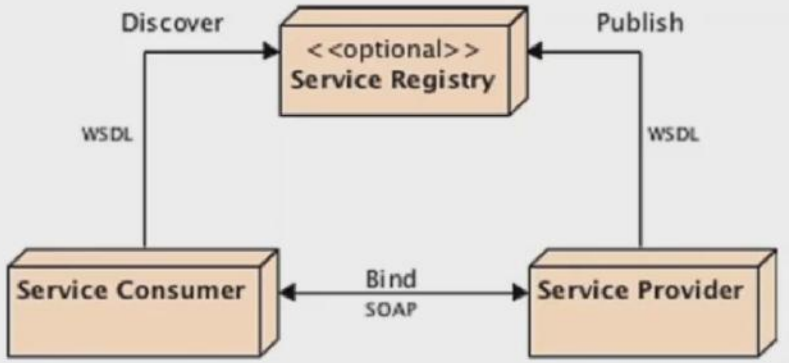
Web Services: i Ruoli

- Service Provider
  - organizzazione che fornisce i servizi
- Service Registry
  - fornisce un repository di *service descriptions* (WSDL) pubblicati dal Service Provider
- Service Consumer
  - fruitore dei servizi
    - individuo
    - applicazione



Web Services: le Iterazioni

- Il Web Service può essere (opzionalmente) registrato in un registro con *Universal Description Discovery and Integration* (UDDI)
- Quando il consumer conosce l'interfaccia del servizio e il formato del messaggio
  - può inviare una richiesta al provider del servizio e ricevere una risposta

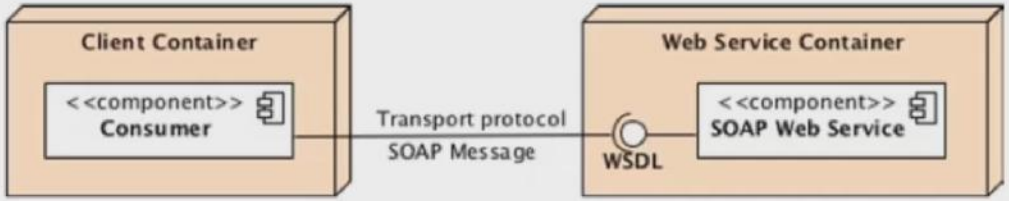


Gli Standard

- Gli standard dietro i WS
  - WSDL
  - SOAP
  - UDDI
  - le specifiche di WS

Gli Standard: WSDL

- Serve a specificare l'interfaccia di un Web Service: il contratto garantito
- Ruolo di Interface Definition Language tra i consumer e il servizio (fornito dal provider)
- Fornisce informazioni su:
  - tipo del messaggio
  - porta
  - protocollo di comunicazione
  - operazioni supportate
  - posizione
  - cosa si aspetta il consumatore come valore di ritorno



Elementi e attributi di WSDL

Element	Description
`definitions`	Is the root element of the WSDL, and it specifies the global declarations of namespaces that are visible throughout the document
`types`	Defines the data types to be used in the messages. In this example, it is the XML Schema Definition (`CardValidatorService?xsd=1`) that describes the parameters passed to the web service request and the response
`message`	Defines the format of data being transmitted between a web service consumer and the web service itself. Here you have the request (the `validate` method) and the response (`validateResponse`)
`portType`	Specifies the operations of the web service (the `validate` method). Each operation refers to an input and output message
`binding`	Describes the concrete protocol (here SOAP) and data formats for the operations and messages defined for a particular port type
`service`	Contains a collection of `<port>` elements, where each port is associated with an endpoint (a network address location or URL)
`port`	Specifies an address for a binding, thus defining a single communication endpoint

Esempio di file WSDL per un servizio di Credit Card Validation

- XML schema dati

`<definitions>`
Is the root element of the WSDL, and it specifies the global declarations of namespaces that are visible throughout the document

- Definizione del tipo di dato

`<types>`
Defines the data types to be used in the messages. In this example, it is the XML Schema Definition (CardValidatorService?xsd=1) that describes the parameters passed to the web service request and the response

- Formato del messaggio
  - dal consumer al provider e risposta

`<message>`
Defines the format of data being transmitted between a web service consumer and the web service itself. Here you have the request (the validate method) and the response (validateResponse)

- Specifica dell'operazione

`<portType>`
Specifies the operations of the web service (the validate method). Each operation refers to an input and output message

```
<?xmlversion="1.0"encoding="UTF-8"?>
<definitions  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://chapter14.javaee7.book.agoncal.org/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://chapter14.javaee7.book.agoncal.org/"
  name="CardValidatorService">
<types>
 <xsd:schema>
   <xsd:import namespace="http://chapter14.javaee7.book.agoncal.org/"
        schemaLocation="http://localhost:8080/chapter14/CardValidatorService?xsd=1">
 </xsd:schema>
</types>
<message name="validate">
  <part name="parameters"element="tns:validate"/>
</message>
<message name="validateResponse">
  <part name="parameters"element="tns:validateResponse"/>
</message>
<portType name="CardValidator">
  <operation name="validate">
    <input message="tns:validate"/>
    <output message="tns:validateResponse"/>
  </operation>
</portType>
 ...
```

Esempio di file WSDL per un servizio di Credit Card Validation

- Descrizione protocollo
  - codifica SOAP dell'operazione
  - ... con input e output

<binding>
Describes the concrete protocol (here SOAP) and data formats for the operations and messages defined for a particular port type

- Definizione del servizio
  - Posizione del servizio

<service>
Contains a collection of <port> elements, where each port is associated with an endpoint (a network address location or URL)

```
<binding name="CardValidatorPortBinding"
         type="tns:CardValidator">
  <soap:binding
         transport="http://schemas.xmlsoap.org/soap/http"
         style="document"/>
  <operation name="validate">
    <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
  </operation>
</binding>
<service name="CardValidatorService">
  <port name="CardValidatorPort" binding="tns:CardValidatorPortBinding">
    <soap:address
       location="http://localhost:8080/chapter14/CardValidatorService" />
  </port>
</service>
</definitions>
```

Standard: SOAP

- Da specifiche W3C:

"SOAP is a lightweight protocol for exchanging structured information in a decentralized, distributed environment.

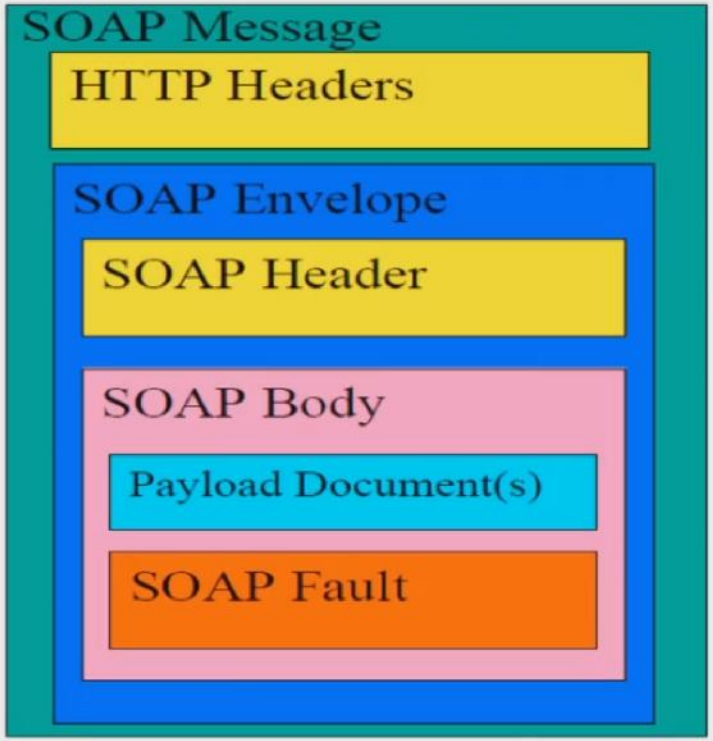
It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses."

Standard: SOAP

- Protocollo per lo scambio di messaggi
- *Simple Object Access Protocol* (SOAP)
- Fortemente basato su XML: i messaggi scambiati sono strutturati in diversi elementi (envelope, header, body) definiti in XML
- Usa HTTP come protocollo di scambio:
  - richiesta HTTP corrisponde all'invio della richiesta del servizio
  - la risposta HTTP corrisponde al risultato della invocazione
- Caratteristiche fondamentali: indipendente dalla piattaforma

Standard: Struttura di un messaggio in SOAP

- SOAP envelope
  - identifica il documento XML come messaggio SOAP
- SOAP header (opzionale)
  - contiene informazioni aggiuntive per il message processing
- SOAP Body
  - contenuto vero e proprio del messaggio
  - contiene chiamate e risposte
- SOAP fault
  - contiene informazioni su eventuali errori verificatisi durante il processing



Standard: Attributi ed eventi SOAP

- La tabella che segue mostra un sottoinsieme dei SOAP elements and attribute
- Come WSDL, anche SOAP definito dalla W3C

Element	Description
`Envelope`	Defines the message and the namespace used in the document. This is a required root element
`Header`	Contains any optional attributes of the message or application-specific infrastructure such as security information or network routing
`Body`	Contains the message being exchanged between applications
`Fault`	Provides information about errors that occur while the message is processed. This element is optional

Esempio Richiesta SOAP

- Richiesta HTTP

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://example.org/2001/06/quotes"

<env:Envelope xmlns:env="http://www.w3.org/2001/06/soap-envelope" >
    <env:Body>
        <m:GetLastTradePrice
            env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"
            xmlns:m="http://example.org/2001/06/quotes">
            <symbol>DIS</symbol>
        </m:GetLastTradePrice>
    </env:Body>
</env:Envelope>
```

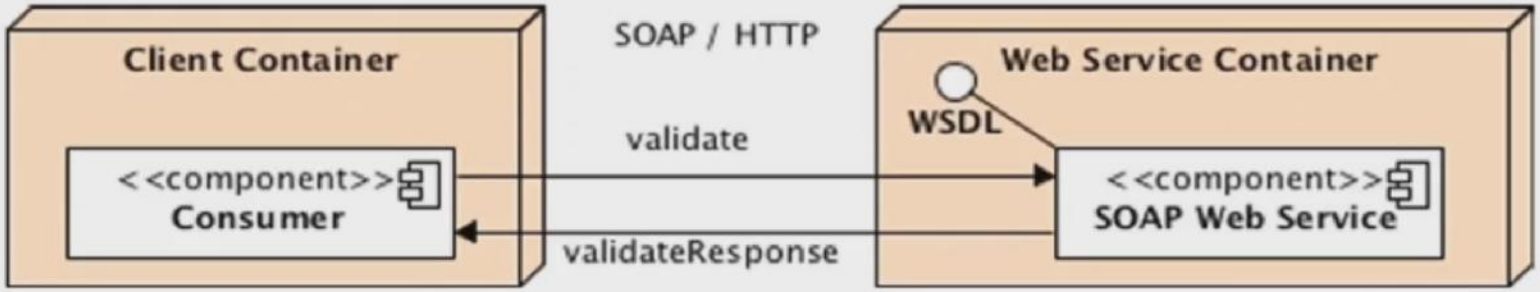
Esempio Risposta SOAP

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

<env:Envelope xmlns:env="http://www.w3.org/2001/06/soap-envelope" >
    <env:Body>
        <m:GetLastTradePriceResponse
            env:encodingStyle="http://www.w3.org/2001/06/soap-encoding"
            xmlns:m="http://example.org/2001/06/quotes">
            <Price>34.5</Price>
        </m:GetLastTradePriceResponse>
    </env:Body>
</env:Envelope>
```

Esempio SOAP e WSDL

- WSDL descrive un'interfaccia del web service mentre SOAP fornisce una concreta implementazione definendo i messaggi XML scambiati fra il consumer ed il provider



- Nel nostro esempio iniziale, quali sono i messaggi che vengono scambiati?

1. Il consumer invia le informazioni relative alla carta di credito all'interno di un SOAP envelope al metodo validate del web service card validator
2. Il servizio restituisce un altro SOAP envelope con il risultato della validazione (true o false)

Esempio Card Validator: Richiesta e Risposta SOAP

- Richiesta
  - "busta" del messaggio
  - l'*header* (vuoto)
  - Il corpo con
    - l'invocazione del servizio
    - i parametri passati
- Risposta
  - la "busta" della risposta
  - Il corpo
  - risultato (valore *true*)

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:cc="http://chapter14.javaee7.book.agoncal.org/">
  <soap:Header/>
  <soap:Body>
    <cc:validate>
      <arg0 number="123456789011"  expiry_date="10/12"
            control_number="544"   type="Visa"/>
    </cc:validate>
  </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:cc="http://chapter14.javaee7.book.agoncal.org/">
  <soap:Body>
    <cc:validateResponse>
      <return>true</return>
    </cc:validateResponse>
  </soap:Body>
</soap:Envelope>
```

Element	Description
Envelope	Defines the message and the namespace used in the document. This is a required root element
Header	Contains any optional attributes of the message or application-specific infrastructure such as security information or network routing
Body	Contains the message being exchanged between applications
Fault	Provides information about errors that occur while the message is processed. This element is optional

Standard: UDDI

- Universal Description Discovery and Integration (UDDI)
- Servizio di localizzazione dei servizi (un *registry*)
  - pubblicazione da parte del provider della descrizione del servizio in WSDL in un registro UDDI accessibile
  - può essere, in questa maniera, scoperto e scaricato
- E' opzionale, quando si conosce già la locazione del servizio
- UDDI è basato su XML
  - viene interrogato da messaggi SOAP e fornisce collegamenti a documenti WSDL
  - contiene informazioni addizionali sul servizio (come disponibilità, costo, etc.)
  - attualmente poco utilizzato e in disuso: WS un pò meno *loose coupled*

Standard: le specifiche di WS

- Una breve storia dei Web Services
  - necessario per comprendere tutti gli standard
  - protocolli distribuiti: lotta tra standard, tra ecosistemi diversi con diversi precursori:
    - DCOM (Microsoft), RPC/RMI (Sun/Oracle), CORBA
  - nessun vincitore né vinti: uso di HTTP, protocollo affermatosi come standard sul WWW
  - SOAP 1.0 fu lanciato da Microsoft e appoggiato da IBM
  - standard successivi: SOAP, WSDL, UDDI
  - supporto da Java a partire dal 2002
  - standard diversi gestiti da organismi diversi

Standard: le specifiche di WS

Specification	Version	Stand. body	JSR	URL
JAX-WS	2.2a	JCP	224	http://jcp.org/en/jsr/detail?id=224
Web Services	1.3	JCP	109	http://jcp.org/en/jsr/detail?id=109
Web Services Metadata	2.1	JCP	181	http://jcp.org/en/jsr/detail?id=181
JAXB	2.2	JCP	222	http://jcp.org/en/jsr/detail?id=222
SAAJ	1.3	JCP	67	http://jcp.org/en/jsr/detail?id=67
JAX-RPC	*1.1*	*JCP*	*101*	*http://jcp.org/en/jsr/detail?id=101*
JAXR	*1.1*	*JCP*	*93*	*http://jcp.org/en/jsr/detail?id=93*
SOAP	1.2	W3C		http://www.w3.org/TR/soap/
XML	1.1	W3C		http://www.w3.org/TR/xml
WSDL	1.1	W3C		http://www.w3.org/TR/wsdl
UDDI	1.0	OASIS		http://uddi.org/pubs/uddi_v3.htm

Standard WS: Implementazione di riferimento per Java

- Implementazione di riferimento è Metro (open source) per quanto riguarda il mondo Java
- Implementa JAX-WS (versione 2.3 ora) che permette di costruire e consumare WS in Java
- Metro stack prodotto dalla community di GlassFish
  - anche Apache CXF (precedentemente conosciuto come XFire) e Apache Axis2 implementano lo stack JWS
- JAX-WS permette di mascherare SOAP dal programmatore

Come sviluppare un Web Service

- Si può partire da due direzioni: dall'alto (specifiche) verso il basso (codice) oppure in direzione opposta

Come sviluppare un Web Service

- Si può partire da due direzioni: dall'alto (specifiche) verso il basso (codice) oppure in direzione opposta
- Approccio top-down: *contract first*
  - si parte dal contratto in WSDL, definendo operazioni, messaggi etc.
  - generazione automatica delle classi da WSDL fornite da Metro (`wsimport`)
- Approccio bottom-up: *program first*
  - l'implementazione in Java già esiste
  - generazione automatica del WSDL a partire dalle classi fornite da Metro (`wsgen`)
- In entrambi i casi il compito del programmatore è facilitato da JAX-WS, che permette di annotare un POJO e farlo "diventare" Web Service
  - anche qui si applica il *configuration-by-exception*

Come sviluppare un Web Service

- Esempio di web service:
  - Web service che valida una carta di credito

Esempio: WS Card Validator

- Annotazione che denota un WS
  - nome del POJO
  - metodo offerto via WS
    - input: oggetto di tipo `CreditCard`
    - restituisce un booleano
    - controlla se la carta è validata (lastDigit dispari) oppure no

```
@WebService
public class CardValidator {

  public boolean validate(CreditCard creditCard){
    Character lastDigit =
        creditCard.getNumber().charAt(
        creditCard.getNumber().length()-1);
    if(Integer.parseInt(lastDigit.toString()) % 2 != 0) {
      return true;
    } else{
      return false;
    }
  }

}
```

Esempio: Cosa viene scambiato: la carta di credito

- Un oggetto `CreditCard` viene scambiato tra il consumer e il SOAP web service
- I dati da scambiare devono essere documenti XML
  - necessario un metodo per trasformare un oggetto Java in XML
  - necessario trasformare un'invocazione di un metodo Java in una chiamata XML su HTTP

Esempio: Cosa viene scambiato: la carta di credito

- Un oggetto `CreditCard` viene scambiato tra il consumer e il SOAP web service
- I dati da scambiare devono essere documenti XML
  - necessario un metodo per trasformare un oggetto Java in XML
  - necessario trasformare un'invocazione di un metodo Java in una chiamata XML su HTTP
- JAXB (*Java Architecture for XML Binding*)
  - permette questo con semplici annotazioni
  - il programmatore Java può evitare di scrivere XML

Listing 14-6. The CreditCard Class with JAXB Annotations

```
@XmlRootElement
public class CreditCard {

  @XmlAttribute(required = true)
  private String number;
  @XmlAttribute(name = "expiry_date", required = true)
  private String expiryDate;
  @XmlAttribute(name = "control_number", required = true)
  private Integer controlNumber;
  @XmlAttribute(required = true)
  private String type;

  // Constructors, getters, setters
}
```

Esempio: la classe CreditCard con le annotazioni JAXB

- Definizione del root element del file XML
- Il nome della classe/tipo
- Attributi
  - un attributo XML, con lo stesso nome del campo Java
  - un attributo obbligatorio, con un nome diverso da quello del campo

```java
@XmlRootElement
public class CreditCard {
  @XmlAttribute(required = true)
  private String number;

  @XmlAttribute(name ="expiry_date", required = true)
  private String expiryDate;

  @XmlAttribute(name ="control_number", required = true)
  private Integer controlNumber;

  @XmlAttribute(required = true)
  private String type;

  //Constructors,getters,setters
}
```

Anatomia di un SOAP WS

- Requisiti:
  - essere annotata con `@WebService` (oppure nel file XML `webservices.xml`)
  - implementare zero o più interfacce (service endpoint) annotate con `@WebService`
  - esser pubblica (non final o astratta)
  - avere un costruttore pubblico di default
  - non avere un `finalize()`
  - essere annotata anche con `@Stateless` o `@Singleton` se si vuole renderla anche un *EJB endpoint*
    - un servizio è sempre stateless!

Esempio: SOAP Web Service Endpoints

- JAX-WS permette sia a regolari classi Java che EJBs di essere esposti come Web services

- Vediamo un esempio

Esempio: CardValidator come EJB

- E' un WS
  - anche un EJB Stateless
- Metodo offerto sia come WS che come EJB
  - restituisce vero oppure falso

```
@WebService
@Stateless
public class CardValidator {
  public boolean validate(CreditCard creditCard){
    Character lastDigit =
      creditCard.getNumber().charAt(
        creditCard.getNumber().length()-1);
    if(Integer.parseInt(lastDigit.toString()) % 2 != 0) {
      return true;
    } else{
      return false;
    }
  }
}
```

A cosa serve avere un EJB?

- Si guadagnano i servizi che sono offerti per EJB e non sono offerti per i WS
  - Transazioni e sicurezza in maniera automatica
- Gestione di interceptor, etc., sono possibili in questo caso
- Flessibilità ed efficienza:
  - lo stesso "servizio" accessibile per consumer fortemente accoppiati (EJB via RMI-IIOP) oppure per consumer debolmente accoppiati (WS via SOAP)
- Stesso servizio offerto in maniera dipendente (ed efficiente) e indipendente dalla piattaforma (meno efficiente)

Conclusioni

- Service-Oriented Architecture
- I Web Services
  - Definizioni
  - I ruoli nei WS
- Gli standard
  - WSDL
  - SOAP
  - UDDI
  - Le specifiche di WS
- Conclusioni