Web Services en Java

Mohamed DERKAOUI
18/05/2017

Plan

- Comprendre le besoin
- Manipuler du XML et du JSON en Java
- Implémenter et interroger des services web SOAP
- Implémenter et interroger des services web REST
- Sécuriser des web services

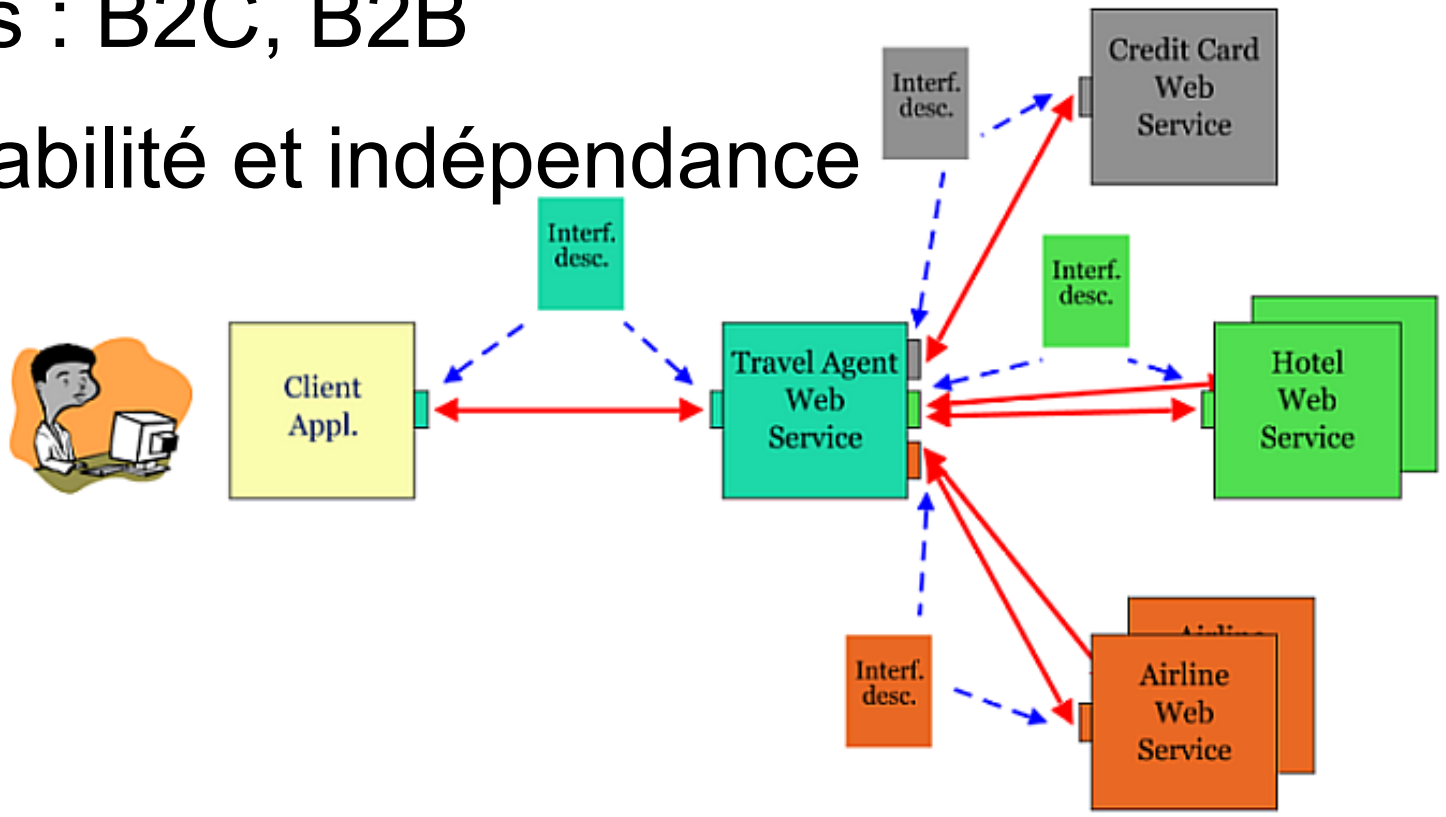
Comprendre le besoin

Plan

- Architecture Java EE : multi-tiers, composants distribués
- Architecture orientée service (SOA) : composantes, technologies
- Présentation des Web Services (WS) : fonctionnement, intérêt, interopérabilité
- Technologies : protocole SOAP, Architecture REST
- Plates-formes à services web
- Choix de l'implémentation : Axis, CXF, JBossWS, Metro...
- Liste des API Java

Contexte

- Application web distribuée s'appuyant sur un assemblage de services
- Échanges : B2C, B2B
- Interopérabilité et indépendance



Web Service

- Application (composant) web interopérable qui expose un ensemble de méthodes accessibles via HTTP.
- 2 types de services web :

  - SOAP : protocole basé sur des messages XML

  - Architecture REST : utilisation de ressources

Java EE

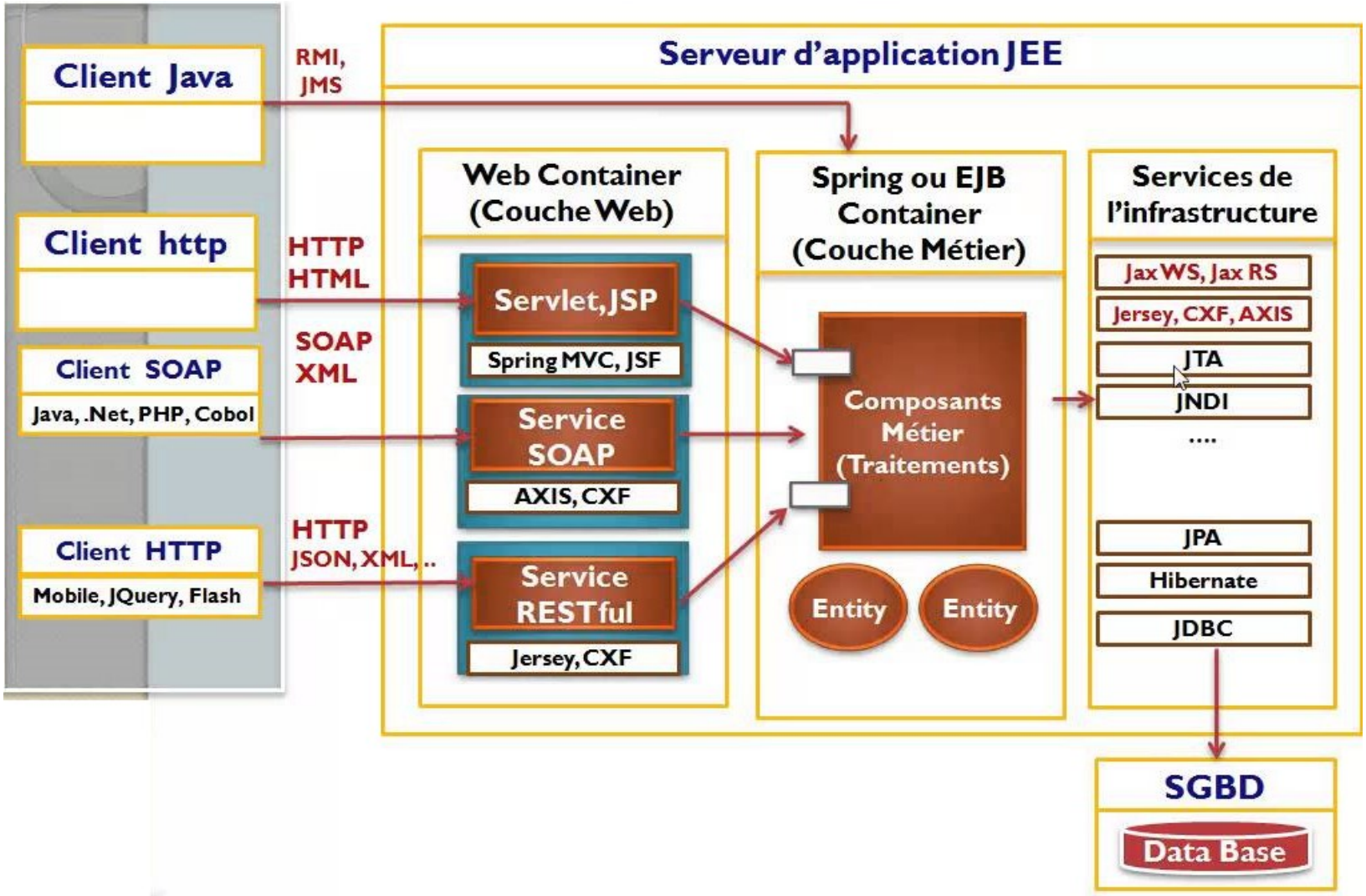
- Java EE est la version "entreprise" de Java, elle a pour but de faciliter le développement d'applications distribuées.
  Mais en fait, Java EE est avant tout une norme.
- C'est un ensemble de standard décrivant des services techniques comme, par exemple, comment accéder à un annuaire, à une base de données, à des documents...

Important : Java EE définit ce qui doit être fournit mais ne dit pas comment cela doit être fournit.

Exemple de services :

- JNDI (Java Naming and Directory Interface) est une API d'accès aux services de nommage et aux annuaires d'entreprises tels que DNS, NIS, LDAP…
- JTA (Java Transaction API) est une API définissant des interfaces standard avec un gestionnaire de transactions.

Architecture Java EE



Serveurs d'applications JEE

- Les applications JEE sont hébergées par des serveurs certifiés JEE : Web-Profile ou Full-Profile
  http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html
- Exemples de serveurs d'applications JEE :
  - Apache Tomcat (conteneur web uniquement)
  - Oracle GlassFish (implémentation de référence) :
    https://glassfish.java.net/download.html
  - Oracle WebLogic
  - IBM WebSphere
  - JBoss

  …

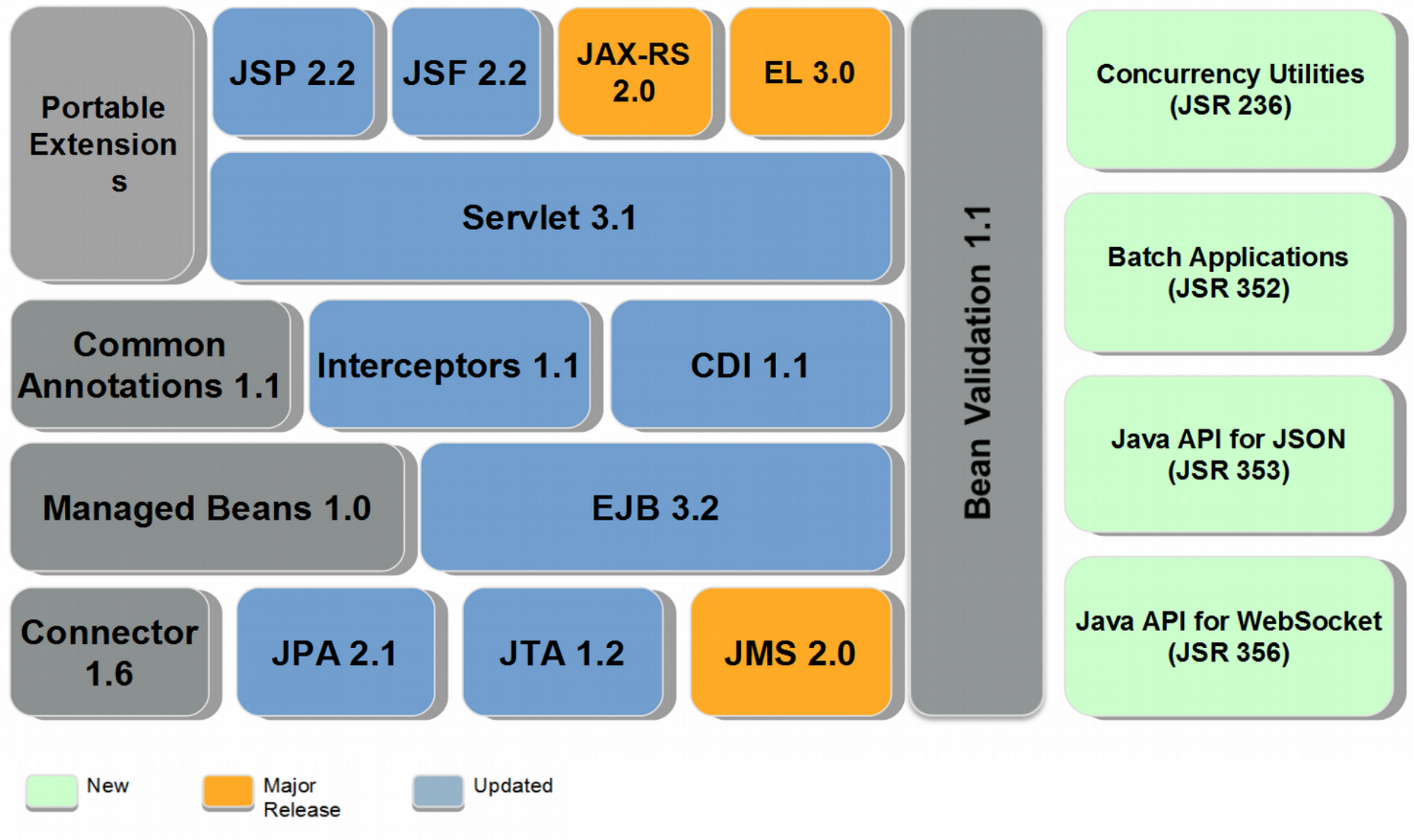
JEE 7

- More annotated POJOs
- Less boilerplate code
- Cohesive integrated platform

- WebSockets
- JSON
- Servlet 3.1 NIO
- REST

- Batch
- Concurrency
- Simplified JMS

JEE 7 : APIs

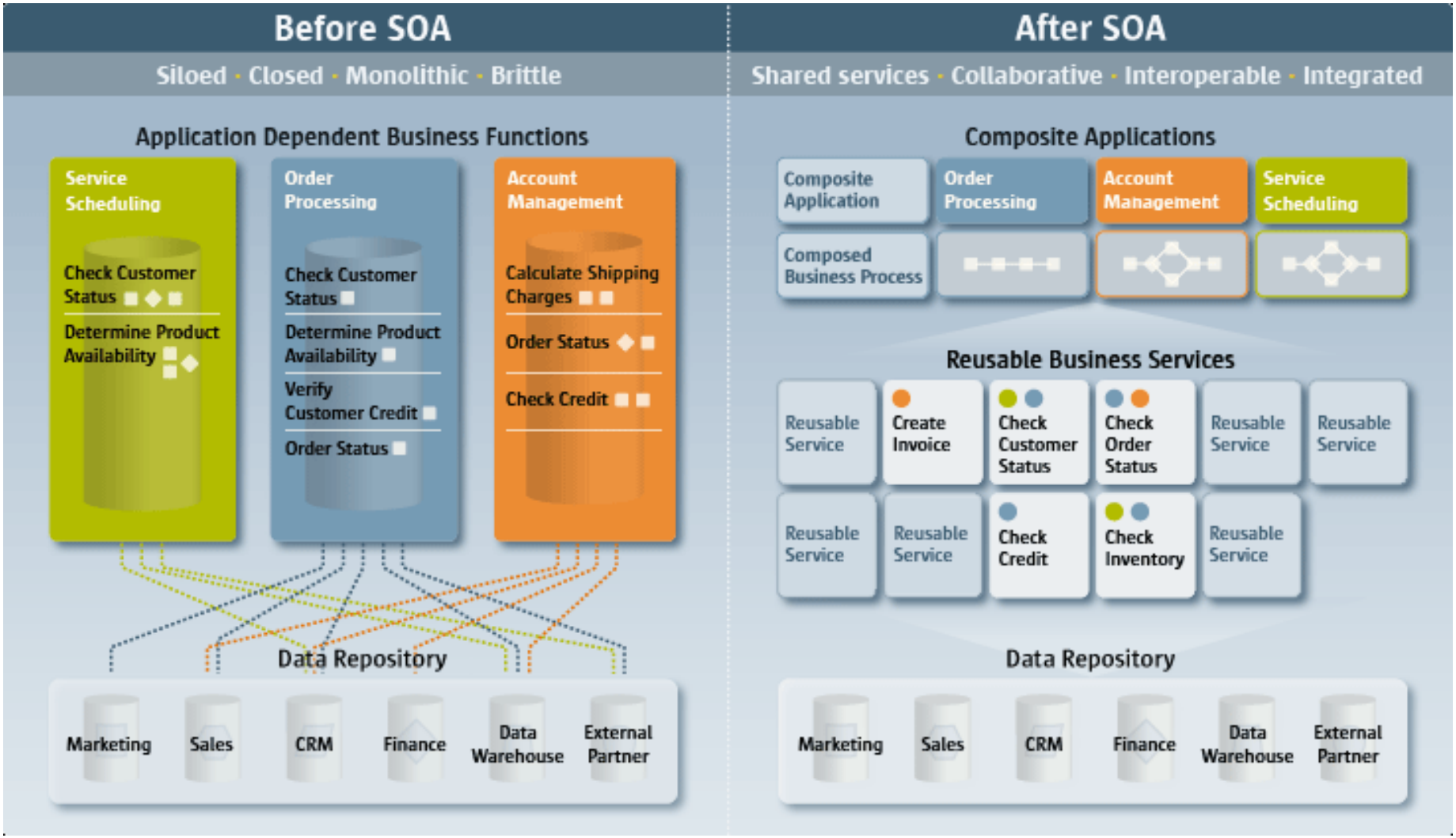

Portable Extensions
JSP 2.2
JSF 2.2
JAX-RS 2.0
EL 3.0
Servlet 3.1
Common Annotations 1.1
Interceptors 1.1
CDI 1.1
Managed Beans 1.0
EJB 3.2
Connector 1.6
JPA 2.1
JTA 1.2
JMS 2.0
Bean Validation 1.1
Concurrency Utilities (JSR 236)
Batch Applications (JSR 352)
Java API for JSON (JSR 353)
Java API for WebSocket (JSR 356)
New
Major Release
Updated

SOA
(Services Oriented Architecture)

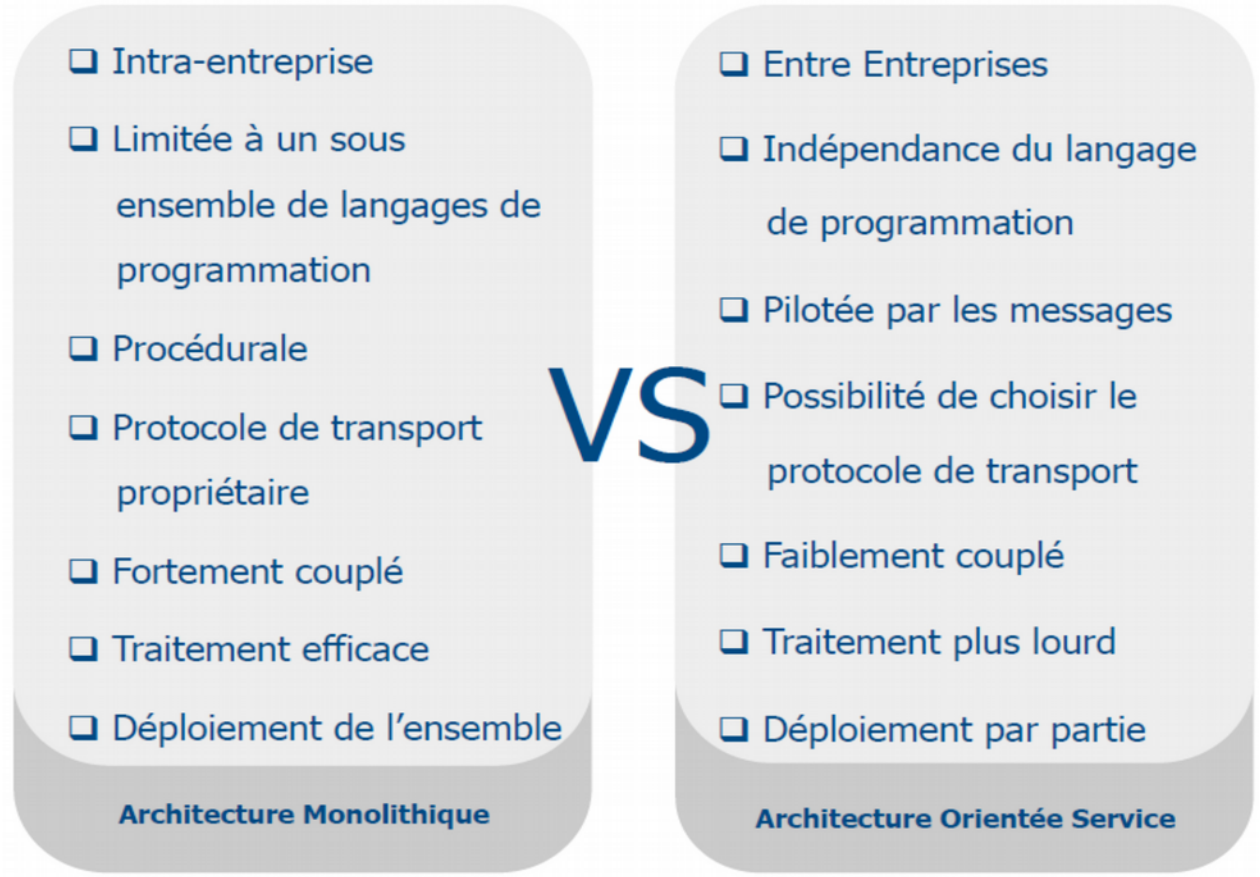
- Un ensemble de principes et de méthodologies pour la conception et le développement de logiciels sous forme de services interopérables .

- Ces services sont des fonctionnalités métiers définies, construites comme des composants pouvant être réutilisés à des fins différentes.

SOA (2)


Before SOA
Siloed · Closed · Monolithic · Brittle
Application Dependent Business Functions
Service Scheduling
Check Customer Status
Determine Product Availability
Order Processing
Check Customer Status
Determine Product Availability
Verify Customer Credit
Order Status
Account Management
Calculate Shipping Charges
Order Status
Check Credit
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner
After SOA
Shared services · Collaborative · Interoperable · Integrated
Composite Applications
Composite Application
Order Processing
Account Management
Service Scheduling
Composed Business Process
Reusable Business Services
Reusable Service
Create Invoice
Check Customer Status
Check Order Status
Reusable Service
Reusable Service
Reusable Service
Reusable Service
Check Credit
Check Inventory
Reusable Service
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner

Architectures Monolitique vs SOA



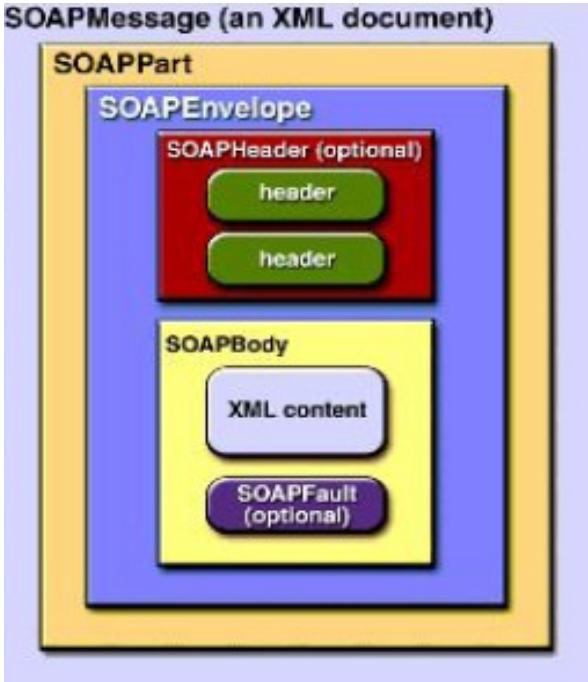
SOAP

- Simple Object Access Protocol
  - Accès à distance à un Web Service
  - Analogue à un protocole d'objets distribués
  - Standard W3C (interopérabilité)

- Protocole d'échange de messages en XML
  - Format du message
  - Encodage
  - Règles d'échange

Messages SOAP

- **Message unidirectionnel** (d'un expéditeur vers un récepteur)
- Structure
  - **– Envelope**
  - • Élément racine
  - • Namespace : SOAPENV
    http://schemas.xmlsoap.org/soap/envelope/
  - **– Header**
  - • Élément optionnel
  - • Contient des entrées non applicatives :
    - – Sessions (SessionId de servlet/jsp/asp), Transactions (BTP), …
  - **– Body**
  - • Contient les entrées du message :
    - – Nom d'une procédure, valeurs des paramètres, valeur de retour
  - • Peut contenir des éléments « fault » (erreurs)



SOAP - Requête

Exemple de message envoyé respectant SOAP :

```
POST /test/WebService.asmx HTTP/1.1
Host: dawan.fr
Content-Type: text/xml; charset=utf-8
Content-Length: 422
SOAPAction: "http://dawan.fr/test/GetNbLivres"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
               xmlns:soap = "http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetNbLivres xmlns="http://dawan.fr/test/" />
  </soap:Body>
</soap:Envelope>
```

SOAP - Réponse

Exemple de réponse au message précédant :

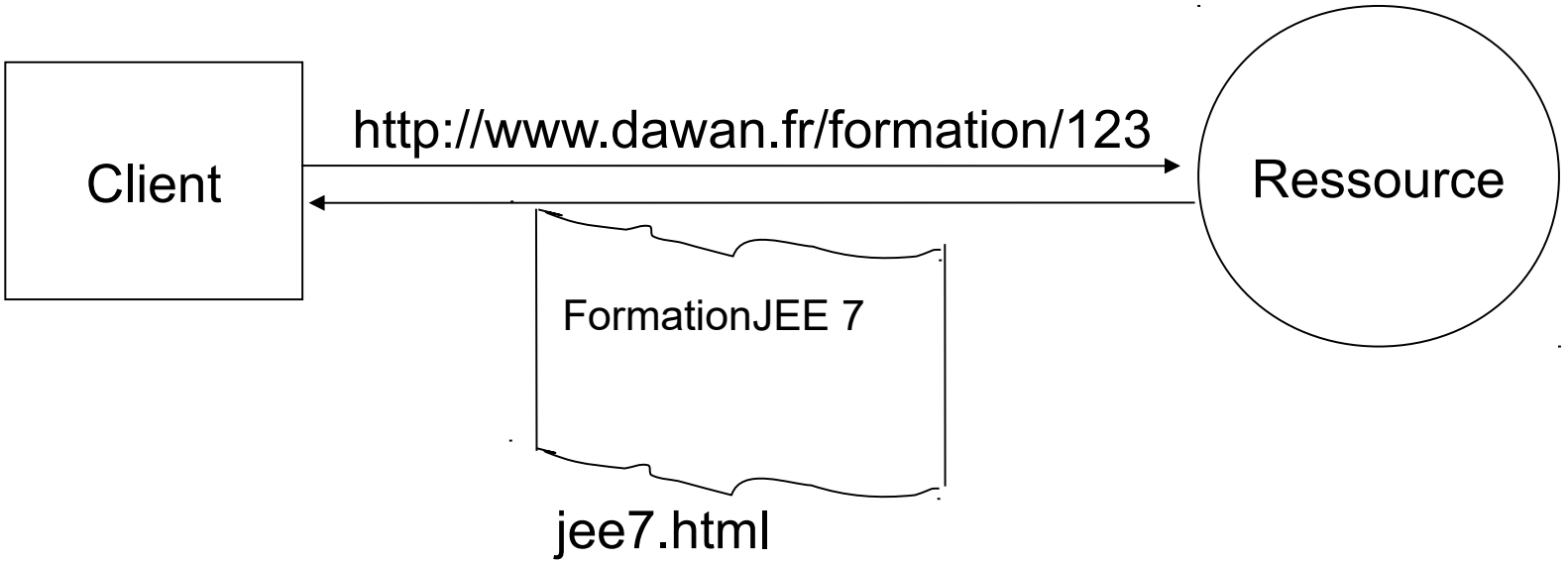
```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 418

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi = "http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
xmlns:soap = "http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetNbLivresResponse xmlns = "http://dawan.fr/test/">
      <GetNbLivresResult>
         <int> 3 </int>
      </GetNbLivresResult>
    </GetNbLivresResponse>
  </soap:Body>
</soap:Envelope>
```

REST Web Services

- REST : un style d'architecture logicielle pour les systèmes distribués, tels que le www. Il est axé sur les ressources.
- Dans une architecture REST, vous avez généralement un serveur REST qui permet d'accéder aux ressources et un client REST qui accède et modifie les ressources REST .
- REST est conçu pour utiliser un protocole de communication apatride, généralement HTTP .
- REST permet aux ressources d'avoir des représentations différentes, par exemple texte , XML , JSON , etc.
  Le client REST peut demander une représentation spécifique via le protocole HTTP (négociation de contenu) .

Representational State Transfer

- **Le client fait référence à une ressource Web à l'aide d'une URL. Une représentation de la ressource est retournée (dans ce cas comme un document HTML) .**
- La représentation (par exemple, jee7.html) place l'application cliente dans un état. Le résultat du client traversant un lien hypertexte dans une autre ressource jee7.html est consultée. La nouvelle représentation place l'application cliente dans un autre état > Représentation State Transfer !



Plateformes de services web

Intégrés aux serveurs d'applications ou modules

- Apache (opensource)
  - Axis : http://ws.apache.org/axis2
  - CXF : http://cxf.apache.org
- **Sun Metro / Glassfish (opensource)**
  https://metro.dev.java.net/
- IBM WebSphere
- Oracle Application Server
- ...

API Java

- **SOAP :**
  JAX-WS 2 (Java API for XML Web Services)
  SAAJ (SOAP with Attachment Api for Java)
  JAX-RPC (Java API for XML RPC) -
- **REST :**
  JAX-RS

Atelier

Installation de l'environnement de développement et d'exécution, choix d'une implémentation de Web Services

Manipuler du Xml et du JSON en Java

Plan

- Rappels XML
- APIs Java pour XML
- Manipulation de JSON

Rappels XML

- Structure d'un document
- Namespaces
- Schémas

API's Java / Xml

- JAXP : Java API for Xml Parsing
- StAX : Streaming API for XML
- JAXB : Java Architecture for XML Binding
- JAXR : Java API for XML Registries
- JAX-WS (JAX-RPC) : Java API for XML Web Services
- SAAJ : SOAP with Attachments API for Java
- JDOM : JDOM se veut une API légère, rapide et facile d'utilisation, pour travailler avec des documents XML. Compatible avec SAX, DOM et JAXP.
- DOM4J : DOM, XPath et XSLT

JAXP

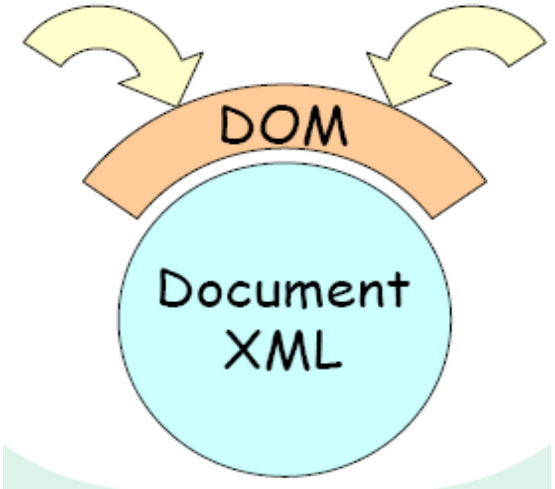
- Fonctionnalités : modélisation, parsing et transformation
- Packages :
- - javax.xml.parsers : contient les interfaces devant être implémentées par les différents parseurs (SAX ou DOM). Ce package fournit aussi un ensemble de factory permettant l'accès aux parseurs.
- - org.w3c.dom : contient l'ensemble des classes et interfaces nécessaires pour travailler avec DOM (modélisation)
- - org.xml.sax : contient l'ensemble des classes et interfaces nécessaires pour travailler avec SAX (parsing)
- - javax.xml.transform : contient l'ensemble des classes et interfaces nécessaires pour la transformation XSLT

JAXP / Spécifications

- SAX 2.0.2 : http://www.saxproject.org/
- XML 1.1 : http://www.w3.org/TR/xml11/
- XML 1.0 : http://www.w3.org/TR/REC-xml
- XInclude : http://www.w3.org/TR/xinclude/
- DOM Level 3 Core : http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407
- DOM Level 3 Load and Save : http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407
- W3C XML Schema : http://www.w3.org/TR/xmlschema-1/
- XSLT 1.0 : http://www.w3.org/TR/1999/REC-xslt-19991116
- XPath 1.0 : http://www.w3.org/TR/xpath

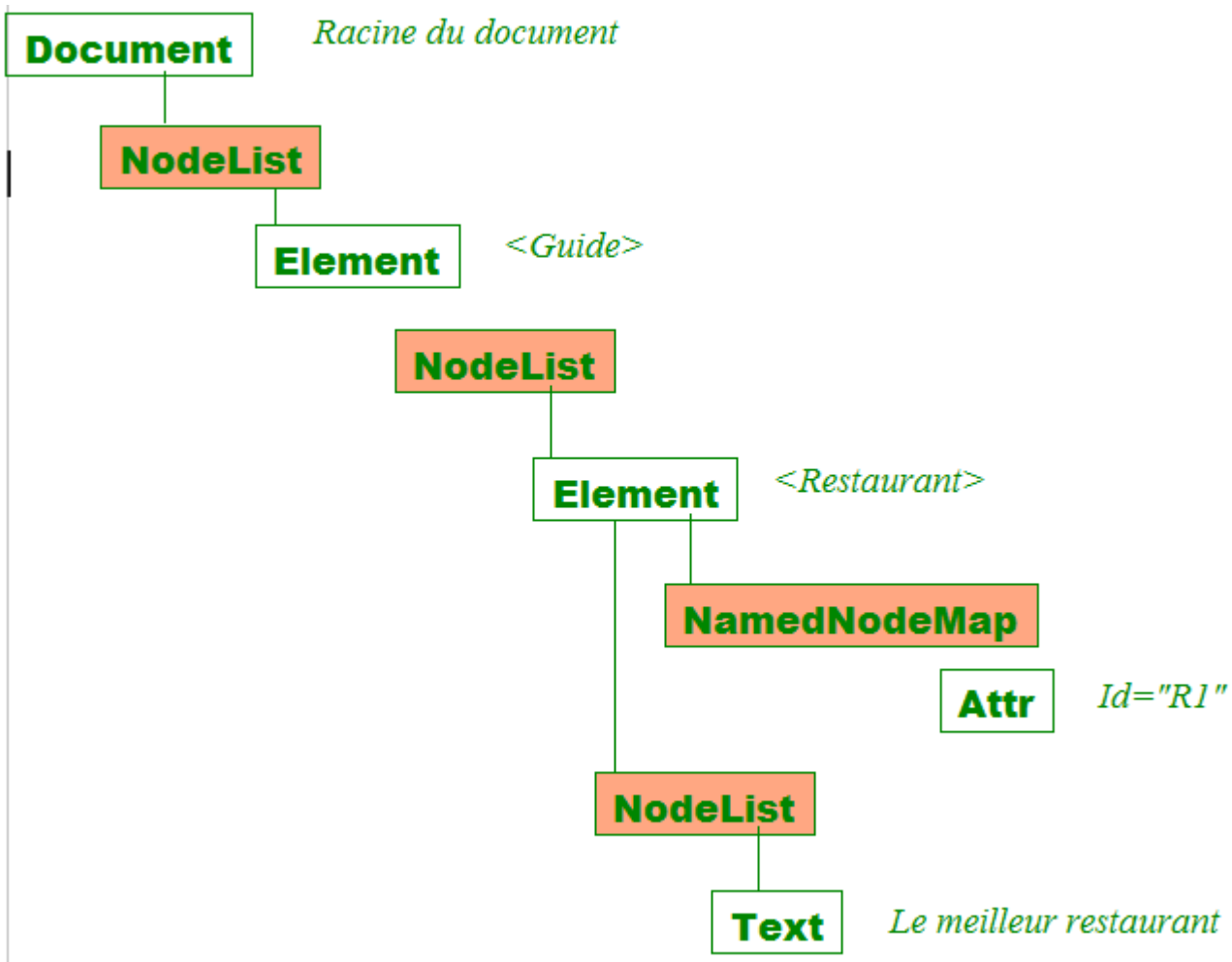
DOM

- DOM = Document Object Model.
- Standard W3C fait pour HTML et XML.
- Structure d'objets pour représenter un document :
  - Résultat d'un "parser".
  - Arbre d'objets reliés entre eux.
- Interface objet pour naviguer dans un document
  - Orientée objet
  - Peut être utilisée en :
    - Java, C++
    - C#, VB
    - Python, PHP



Exemple d'arbre DOM

```
<Guide>
      <Restaurant id="R1">Le meilleur restaurant</Restaurant>
</Guide>
```



API DOM

Le modèle d'objet spécifié par le W3C défini 12 types de noeuds différents.
Le modèle d'objet de document fournit tout une panoplie d'outils destinés à construire et manipuler un document XML.
Pour cela, le DOM met à disposition des interfaces, des méthodes et des propriétés permettant de gérer l'ensemble des composants présents dans un document XML.
Le DOM spécifie diverses méthodes et propriétés permettant notamment, de **créer** (*createNode*…), **modifier** (*replaceChild*…), **supprimer** (*remove*…) ou d'**extraire des données** (*get*…) de n'importe quel élément ou contenu d'un document XML.
De même, **le DOM définit les types de relation entre chaque noeud, et des directions de déplacement dans une arborescence XML.**
Les propriétés : *parentNode*, *childNodes*, *firstChild*, *lastChild*, *previousSibling* et *nextSibling* permettent de retourner respectivement le père, les enfants, le premier enfant, le dernier enfant, le frère précédent et le frère precedent du noeud courant.
Le modèle d'objet de document offre donc au programmeur les moyens de traiter un document XML dans sa totalité

Interfaces DOM

org.w3c.dom.Document

- Interface qui représente une arborescence XML

`Attr createAttribute(String name)`	Crée un attribut
`Element createElement(String name)`	Crée un élément
`Element createElementNS(String namespaceURI, String qualifiedName)`	Crée un élément dans un espace nominal
`Text createTextNode(String data)`	Crée un noeud textuel
`Element getDocumentElement()`	Récupère l'élément racine
`NodeList getElementsByTagName(String name)`	Renvoie une liste de tous les éléments de nom donné.

org.w3c.dom.Element

- Interface représentant un élément XML

String	**getAttribute**(String name) Retrieves an attribute value by name.
Attr	**getAttributeNode**(String name) Retrieves an attribute node by name.
NodeList	**getElementsByTagName**(String name) Returns a NodeList of all descendant Elements with a given tag name, in the order in which they are encountered in a preorder traversal of this Element tree.
String	**getTagName**() The name of the element.
void	**removeAttribute**(String name) Removes an attribute by name.
void	**setAttribute**(String name, String value) Adds a new attribute.

org.w3c.dom.Attr

- Interface représentant un attribut XML

String	**getName**() Returns the name of this attribute.
Element	**getOwnerElement**() The Element node this attribute is attached to or null if this attribute is not in use.
boolean	**getSpecified**() If this attribute was explicitly given a value in the original document, this is true; otherwise, it is false.
String	**getValue**() On retrieval, the value of the attribute is returned as a string.
void	**setValue**(String value) On retrieval, the value of the attribute is returned as a string.

DOM : utilisation (exple JDOM)

```
Public class ExempleDOM
public static main (String argc[]) throws IOException, DOMExcetion
{XMLDocument xmlDoc = new XmlDocument();
// creation des nœuds
ElementNode nom = (ElementNode) xmlDoc.createElement("nom");
ElementNode prenom = (ElementNode) xmlDoc.createElement("prenom");
ElementNode nomfam = (ElementNode) xmlDoc.createElement("nomfam");
// creation de l'arbre
xmlDoc.appendChild(nom);
nom.appenChild(prenom);
prenom.appendChild(xmlDoc.createTextNode("Jean");
nom.appenChild(nomfam);
nomfam.appendChild(xmlDoc.createTextNode("Dupont");
// positionnement d'un attribut
nom.setAttribute("ville", "Paris");
// sortie
System.exit(0); } }
```

Document:

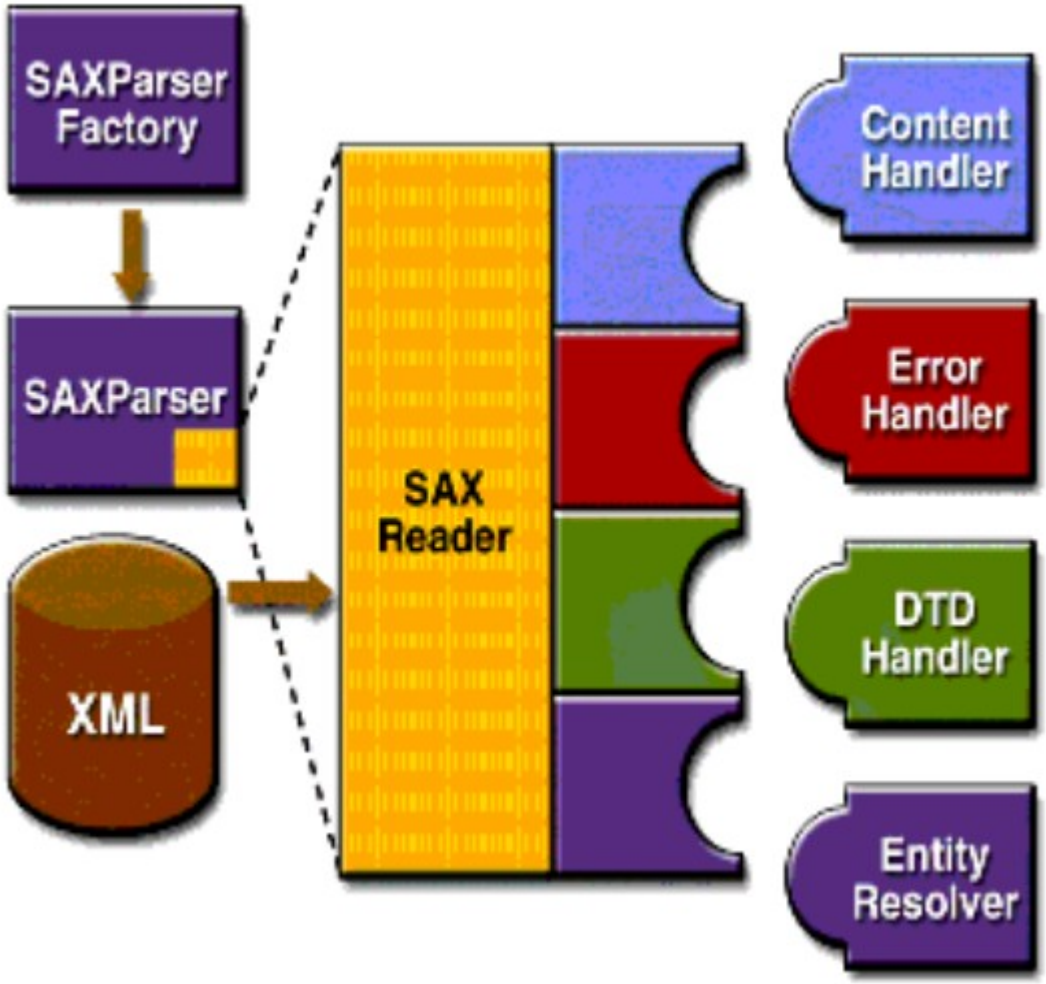
```
<nom ville ="Paris">
   <prenom> Jean </prenom>
   <nomfa> Dupont </nomfa>
</nom>
```

Push parsing / Pull parsing

- **Push parsing (SAX)** : le parseur pousse l'information vers l'application à chaque nouvel événement XML rencontré

- **Pull parsing (StAX)** : l'application déplace elle-même le parseur d'évènement en événement XML
- (design pattern iterator)

SAX


SAXParser Factory
SAXParser
XML
SAX Reader
Content Handler
Error Handler
DTD Handler
Entity Resolver

SAX (2)

- Pour commencer, l'application récupère un parseur (javax.xml.parsers.SAXParser) à partir d'une factory (javax.xml.parsers.SAXParserFactory).
- Ce parseur parcourt le document XML grâce à un lecteur (org.xml.sax.XMLReader). Ce dernier contient plusieurs gestionnaires (ou handlers). Ce sont ces différents gestionnaires qui sont chargés du traitement des "événements" lors du parsing. Voici les quatre principaux types de handlers (interfaces du package org.xml.sax) :
- - Gestionnaire de Contenu : Le ContentHandler est chargé des événements comme le début ou la fin du document, l'ouverture ou a fermeture de balises ou encore la lecture de caractères.
- - Gestionnaire d'Erreurs : Le ErrorHandler va traiter les trois types d'erreurs possibles lors du parsing : les erreurs simples, les erreurs fatales et les warnings.
- - Gestionnaire de DTD : Le DTDHandler (Document Type Definition) gère les événements relatifs aux DTD.
- - Gestionnaire d'entités externes : L'EntityResolver est chargé de gérer les entités externes, en fournissant une InputSource adéquate.

StAX

- JSR-173
- **Pull parsing** : l'évènement est demandée du côté du code client (comprendre le code client en dehors de l'API XML). Le code client soumet la portion de code à analyser par le parseur et récupère entre autre un évènement au quel on choisira de réaliser une action.
- Le pull parsing oblige une analyse vers l'avant uniquement et permet l'écriture mais pas la modification de nœuds XML. Les performances sont très élevées pour le parcours de fichiers volumineux.

JAXB

- Java XML Binding
- Permet le passage de données en java (des structures en mémoire) à des documents XML, et inversement
- Comparable à SAX ou DOM
- Associé (nécessaire) à JAX-RPC
- Utilisable de façon isolée (ex. : application avec un fichier de configuration en XML)

JAXB (2)

- Produire un Schéma (plus facile) ou des classes Java
- grâce aux outils disponibles inclus dans JDK JSE 1.6+
- outil XJC: XSD Schema <-> Java source annoté JAXB.
- Obtenir un schema depuis sources Java :outil schemagen : Java -> schéma XSD
- Marshalling : transformer des données java en XML
- Unmarshalling : transformer un document XML en données java

JAXB
Unmarshalling

- JAXBContext : ensemble d'informations (dont l'emplacement de classes essentielles)
- Unmarshaller : outil qui réalise la transformation

```
JAXBContext jaxbContext =
   JAXBContext.newInstance("fr.dawan.test");

Unmarshaller unmarshaller =
   jaxbContext.createUnmarshaller();

Collection c = (Collection)
   unmarshaller.unmarshal(
      new FileInputStream("etc/Books1.xml"));
```

JAXB
Marshalling

```
JAXBContext jaxbContext =
   JAXBContext.newInstance("fr.dawan.test");

Marshaller marshaller =
   jaxbContext.createMarshaller();

marshaller.setProperty(
     Marshaller.JAXB_FORMATTED_OUTPUT,
     new Boolean(true)); //alinea, ...

StringWriter sw = new StringWriter();

marshaller.marshal(monObjet, sw);
```

Manipuler du JSON en Java JSON-P

- Une API portable pour parser, générer, transformer et requêter des chaînes JSON (JavaScript Object Notation).
- Exemple d'objet JSON :

```
{
   "firstName": "Mohamed",
   "lastName": "DERKAOUI",
   "phoneNumbers": [
      {
         "type": "office",
         "number": "0101010101"
      },
      {
         "type": "fax",
         "number": "0122222222"
      }
   ]
}
```

Object Model API

- Les structures JSON sont représentées par un model objet utilisant les types : **JsonObject** et **JsonArray :**
  - JsonObject fournit une vue (sous forme de Map) pour accéder aux paires clé/valeur.
  - JsonArray fournit une vue (sous forme List) pour accéder aux différents objets du tableau.
- **Classes et interfaces :**

  - **Json** :Contient des méthodes static pour la création de readers, writers, builders, ...
  - **JsonGenerator** : générateur de flux à partir de données JSON.
  - **JsonReader** : flux de lecture et de création d'un modèle objet.
  - **JsonObjectBuilder**, **JsonArrayBuilder** : Builder d'un modèle objet ou tableau en mémoire par ajout de valeurs depuis le code.
  - **JsonWriter** : flux d'écriture d'un modèle objet.
  - **JsonValue, JsonObject, JsonArray, JsonString, JsonNumber** : types de données pour les valeurs JSON.

Lecture / écriture

- **Lecture :**

```
String chaineJSON = "{\"firstName\":\"Mohamed\",\"lastName\":\"DERKAOUI\",\"phones\":
[\"011111111\",\"022222222\"]}";
InputStream is = new ByteArrayInputStream(chaineJSON.getBytes());
JsonReader reader = Json.createReader(is);
is.close();
JsonObject obj = reader.readObject();
try (PrintWriter pw = response.getWriter()) {
    pw.println("<br />Conteu de la lecture : " + obj.toString());
    pw.println("<br />Conteu de firstName : " + obj.getString("firstName"));
}
```

- **Ecriture :**

```
JsonArray jsonArrayPhones = Json.createArrayBuilder().add("011111111").add("022222222").build();
JsonObject jsonObject = Json.createObjectBuilder().add("firstName", "Mohamed")
                                                  .add("lastName", "DERKAOUI")
                                                  .add("phones", jsonArrayPhones)
                                                  .build();
try (PrintWriter pw = response.getWriter()) {
    Map<String, Boolean> config = new HashMap<>();
    config.put(JsonGenerator.PRETTY_PRINTING, true);
    JsonWriter jsonWriter = Json.createWriterFactory(config).createWriter(pw);
    jsonWriter.writeObject(jsonObject);
    jsonWriter.close();
}
```

Streaming API

- Similaire à StAX (Streaming API for XML) ; API basée sur de événements.

- **Classes et interfaces :**

  - **Json** :Contient des méthodes static pour la création de readers, writers,
  - **JsonParser** : parseur basé sur des événements qui permet de lire le JSON.
  - **JsonGenerator** : générateur de flux à partir de données JSON.

- Streaming API est une API bas niveau conçue pour accéder efficacement à un volume important de données JSON.

  D'autres frameworks JSON (comme JSON binding) peuvent être implémentés en utilisant cette API.

Parsing

```
try (PrintWriter out = response.getWriter()) {
    String chaineJSON = "{\"firstName\":\"Mohamed\",\"lastName\":\"DERKAOUI\",\"phones\":
                         [\"011111111\",\"022222222\"]}";
    InputStream is = new ByteArrayInputStream(chaineJSON.getBytes());
    try (JsonParser parser = Json.createParser(is)) {
       while (parser.hasNext()) {
            Event e = parser.next();
            if (e == Event.KEY_NAME) { // Event est une énumération
               switch (parser.getString()) {
                 case "firstName":
                      parser.next();
                      out.print("firstName : " + parser.getString());
                      break;
                 case "lastName":
                      parser.next();
                      out.println("lastName : " + parser.getString());
                      break;
               }
            }
       }
    }
    is.close();
}
```

Implémentation de services web SOAP

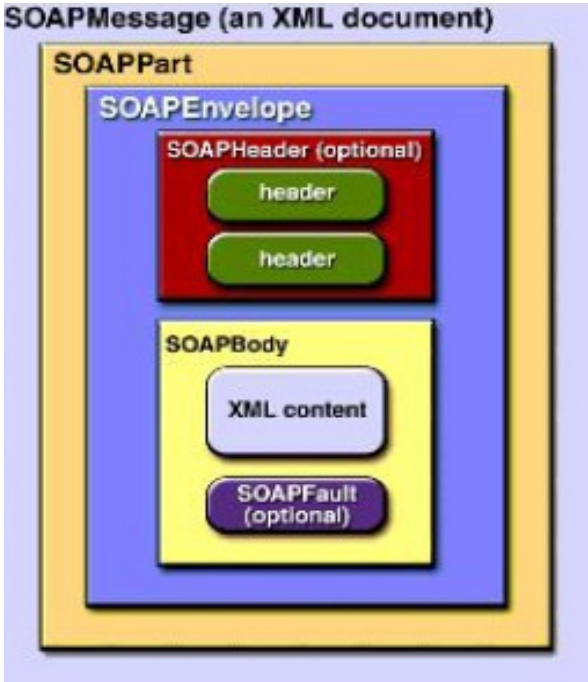
SOAP : Portée

- SOAP est « simple » et extensible

- Il ne couvre pas les fonctions suivantes :
  - Distributed garbage collection
  - Regroupement de messages
  - Passage d'objets par référence
  - Activation (nécessite le passage par référence)

Messages SOAP

- **Message unidirectionnel** (d'un expéditeur vers un récepteur)
- Structure
  - **– Envelope**
  - • Élément racine
  - • Namespace : SOAPENV
    http://schemas.xmlsoap.org/soap/envelope/
  - **– Header**
  - • Élément optionnel
  - • Contient des entrées non applicatives :
    - – Sessions (SessionId de servlet/jsp/asp), Transactions (BTP), …
  - **– Body**
  - • Contient les entrées du message :
    - – Nom d'une procédure, valeurs des paramètres, valeur de retour
  - • Peut contenir des éléments « fault » (erreurs)



SOAP : Entête du message (header)

- **Contient des entrées non applicatives**
  – Transactions, sessions, …

- **L'attribut mustUnderstand**
  – Rien ou =0 : l'élément est optionnel pour l'application réceptrice
  – =1 : l'élément doit être compris de l'application réceptrice
  (sinon échec du traitement du message)
  – Exemple

```
<SOAP-ENV:Header>
  <t:Transaction xmlns:t="some-URI"  SOAP-ENV:mustUnderstand="1">
        5
  </t:Transaction>
</SOAP-ENV:Header>
```

SOAP : Corps du message

- Contient des entrées applicatives
  - Encodage des entrées
- Namespace pour l'encodage
  - SOAPENC

    http://schemas.xmlsoap.org/soap/encoding/
  - xsd : XML Schema

SOAP : Encodage

- Types primitifs

```
<element name="price" type="float"/>                <price>15.57</price>
<element name="greeting"  type="xsd:string"/>       <greeting id="id1">Bonjour</greeting>
```

- Structures

```
<element name="Book"><complexType>          <e:Book>
<element name="author" type="xsd:string"/>   <author>J.R.R Tolkien</author>
<element name="title" type="xsd:string"/>    <title>Bilbo le Hobbit</title>
</complexType></element>                    </e:Book>
```

- Références

```
<element name="salutation" type="xsd:string"/>     <salutation href="#id1"/>
```

- Tableaux

```
<SOAPENC:Array id="id3" SOAPENC:arrayType=xsd:string[2,2]>
  <item>r1c1</item>  <item>r1c2</item>  <item>r2c1</item>  <item>r2c2</item>
</SOAPENC:Array>
```

Sérialisation des données

- **Encoded :** les données sont encodées selon les règles de la sérialisation SOAP
- **Literal :** les données sont sérialisées selon un schéma XML (types définis dans le fichier WSDL, pas dans SOAP) « choisi par WS-I Basic Profile »

- 4 combinaisons :
  - RPC/Encoded (non conforme WS-I)
  - RPC/Literal
  - Document/Encoded (non implémentée)
  - Document/Literal (.NET, AXIS, gSOAP...)

Requête HTTP

Demande de cotation à un serveur

```
POST /StockQuote HTTP/1.1
Host: www.dawan.fr Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<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>IBM</symbol>
      </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Réponse HTTP

```
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>
```

Retours d'erreur (Fault)

4 éléments :

- **Faultcode (obligatoire)**

  Code d'erreur utilisé par le logiciel (switch(faultcode) { case …)
- **Faultstring (obligatoire)**

  Explication lisible par un humain
- **Faultactor (optionnel)**

  • Erreur en cours de cheminement du message (firewall, proxy, MOM)
- **Detail**

  • Détail de l'erreur non lié au Body du message
- **Autres**

  D'autres éléments qualifiés par un namespace peuvent être ajoutés

Faultcode

- 4 groupes de code d'erreurs :

  • Client, Server, MustUnderstand, VersionMismatch

  • Ex: Client.Authentication

WSDL

- Web Services Description Language
- Objectifs
- - Décrire les services comme un ensemble d'opérations et de messages abstraits reliés (bind) à des protocoles et des serveurs réseaux
- Grammaire XML (schema XML)
- Modulaire:
  import d'autres documents WSDL et XSD)

WSDL : Structure

- **<types>**
  – Contient les définition de types (comme XmlSchema).
- **<message>**
  – Décrit les noms et types d'un ensemble de champs à transmettre : Paramètres d'une invocation, valeur du retour, …
- **<portType>**
  – Décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou de fautes
- **<binding>**
  – Spécifie une liaison d'un <porttype> à un protocole concret (SOAP1.1, HTTP1.1, MIME, …). Un porttype peut avoir plusieurs liaisons.
- **<port>**
  – Spécifie un point d'entrée (endpoint) comme la combinaison d'un <binding> et d'une adresse réseau.
- **<service>**
  – Une collection de points d'entrée (endpoint) relatifs.

WSDL : messages

```
<wsdl:message name="SommeResponse">
    <wsdl:part element="impl:SommeResponse"
               name="parameters"/>
</wsdl:message>

<wsdl:message name="SommeRequest">
  <wsdl:part element="impl:Somme" name="parameters"/>
</wsdl:message>
```

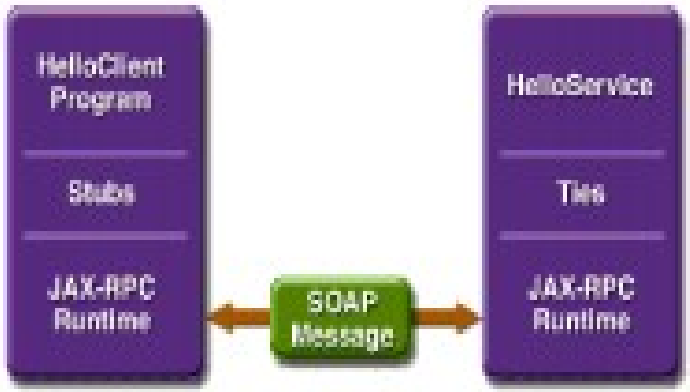
WSDL : portType

PortType : Ensemble des opérations

```
<wsdl:portType name="WebService1_Impl">
    <wsdl:operation name="Somme">
        <wsdl:input message="impl:SommeRequest"
                name="SommeRequest"/>
        <wsdl:output message="impl:SommeResponse"
                name="SommeResponse"/>
    </wsdl:operation>
</wsdl:portType>
```

JAX-WS 2

- Java API for XML WS (javax.xml.ws)
  - Intégrée à Java SE 6+
- Cache la complexité de SOAP
  - Génération de proxy (« stub ») client
  - Déploiement serveur par annotations
  - Spécification d'interface WSDL ou java
- Synchrone / asynchrone



Outils J2SE 1.6+

- wsimport
  – Génération de stubs à partir d'une description WSDL
- wsgen / apt
  – Annotations Web Services
- xjc
  – JAXB : Générateur XSD>JAVA
- schemagen
  – JAXB : Générateur java > XSD

JAX-WS : EndPoint

- Implémentation côté serveur
- Service Endpoint Interface
  - Interface java (explicite ou implicite : classe)
  - Annotations : @WebService / @WebMethod ...
  - Méthodes métier : paramètres / retours compatibles JAXB
  - Constructeur public par défaut, pas de finalize()
- Cycle de vie
  - Annotations : @PostConstruct / @PreDestroy

JAX-WS : EndPoint (1)

```
package calculette;
import javax.jws.WebMethod;
import javax.jws.WebService;

@WebService
public class Calculette {
 @WebMethod
 public int addition(int n1, int n2){
  return n1+n2;
 }
 @WebMethod
 public int soustraction(int n1, int n2){
   return n1-n2;
 }
}
```

JAX-WS : déploiement

- Le déploiement dépend de l'implémentation

- On peut également définir le EndPoint :

  javax.xml.ws.Endpoint

  et utiliser la méthode publish() pour publier le service web

JAX-WS : EndPoint (2)

```
package calculette;
import javax.xml.ws.Endpoint;

public class CalcServer {
 public static void main(String[] args) {
  String port="8080";
  String address =
     "http://localhost:"+port+"/calculette";

  Endpoint endPoint = Endpoint.publish(address,
                                new Calculette());

  System.out.println("WSDL :
     http://localhost:8080/calculette?wsdl");
  System.out.println("XSD :
     http://localhost:8080/calculette?xsd=1");
 }
}
```

JAX-WS : Client

- **Génération du stub :**

  wsimport d stub s . http://localhost:8080/calculette?wsdl

- d = répertoire cible, s = cible pour les sources java
- **Stub généré (classes utilisables dans le client) :**

  • Une interface (« Service Endpoint Interface » ou SEI) par « port » du WSDL (ex. Calculette)

  • Une classe par « service » du WSDL (ex. CalculetteService)

  • Mapping JAXB : 1 classe par « message » du WSDL (ex. Addition, AdditionResponse), et 1 classe ObjectFactory

- **Ecriture du client**

```
Calculette port = new CalculetteServiceLocator().getCalculettePort();
System.out.println("1+2=" + port.addition(1, 2));
```

JAX-WS : à partir du .wsdl

- Génération des stubs (et du mapping JAXB)

avec wsimport

- Ecriture d'une classe qui implémente la «Service Endpoint Interface » générée :
  - Annotation @WebService(endpointInterface= « interfaceGeneree »)
  - Annotations @Webmethod pour l'implémentation des méthodes de l'interface

Handlers

Intercepteurs

– Message entrant, sortant, fault
– Implémente LogicalHandler ou SOAPHandler
  • Package javax.xml.ws.handler [.soap]
– Possibilité de définir une chaîne de Handlers

Déclaration

– Annotation @HandlerChain(file=«handlers.xml»)
  • Appliquée à la classe d'implémentation du Web Service
– Méthode setHandlerChain( ) de javax.xml.ws.Binding
  • endPoint.getBinding();

Usage : logging, sécurité, modification de contenu

Handlers : exemple (déclaration)

```
@WebService
@HandlerChain(file="handlers.xml")
public class Calculette {
  // ...
}
```

```
<?xml version="1.0" encoding="UTF8"?>
<! handlers.xml >
<handlerchains xmlns="http://java.sun.com/xml/ns/javaee">
   <handlerchain>
     <handler>
       <handlerclass>monPackage.MyLogHandler</handlerclass>
     </handler>
   </handlerchain>
</handlerchains>
```

Handlers : exemple (code)

```
public class MyLogHandler implements SOAPHandler<SOAPMessageContext> {
  public boolean handleMessage(SOAPMessageContext smc) {
    try {
       smc.getMessage().writeTo(System.out);
    } catch(Exception ignore) { }
    return true;
  }
  public boolean handleFault(SOAPMessageContext smc) {
    return handleMessage(smc);
  }
  public Set<QName> getHeaders() { return null; }
  public void close(MessageContext messageContext) { }
}
```

SAAJ

SOAP with Attachments for Java

– javax.xml.soap

– Avantages : attachements MIME

– Inconvénients : API bas niveau (SOAP)

SAAJ

API de niveau message

– SOAPConnectionFactory / SOAPConnection

– SOAPMessage

• (SOAPPart (SOAPEnvelope (SOAPBody (SOAPElement)*))

– SOAPMessage reply = connection.call(message, destination);

SAAJ : Exemple

```
SOAPConnectionFactory soapCF = SOAPConnectionFactory.newInstance();
SOAPConnection connection = soapCF.createConnection();
MessageFactory messageFactory = MessageFactory.newInstance();
SOAPMessage message = messageFactory.createMessage();
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPBody body = envelope.getBody();
SOAPElement bodyElement = body.addChildElement(
    envelope.createName("addition", "ns1", "http://calculette/"));
bodyElement.addChildElement("n1").addTextNode("1");
bodyElement.addChildElement("n2").addTextNode("3");
message.saveChanges();
SOAPMessage reply = connection.call(message, "http://localhost:8080/calculette");
```

SAAJ : Attachement

```
import javax.xml.soap.AttachmentPart;
MessageFactory messageFactory = MessageFactory.newInstance();
SOAPMessage m = messageFactory.createMessage();
AttachmentPart ap1 = m.createAttachmentPart();
ap1.setContent("Ceci est un cours SAAJ", "text/plain");
m.addAttachmentPart(ap1);
AttachmentPart ap2 = m.createAttachmentPart();
ap2.setRawContent(new FileInputStream("logo.jpg"),  "image/jpeg");
m.addAttachmentPart(ap2);
```

Implémentation de services web REST

Plan

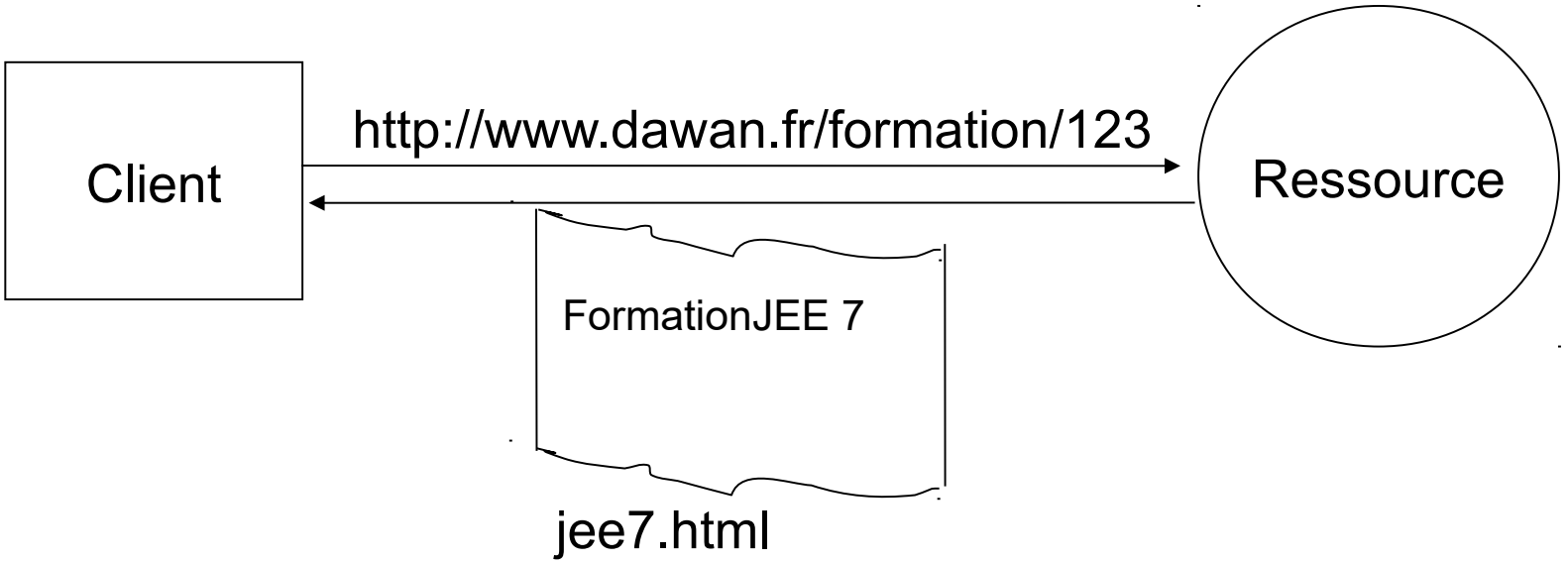
- Architecture REST
- API JAX-RS : présentation, implémentation
- Implémentation de clients
- Intercepteurs et filtres

REST Web Services

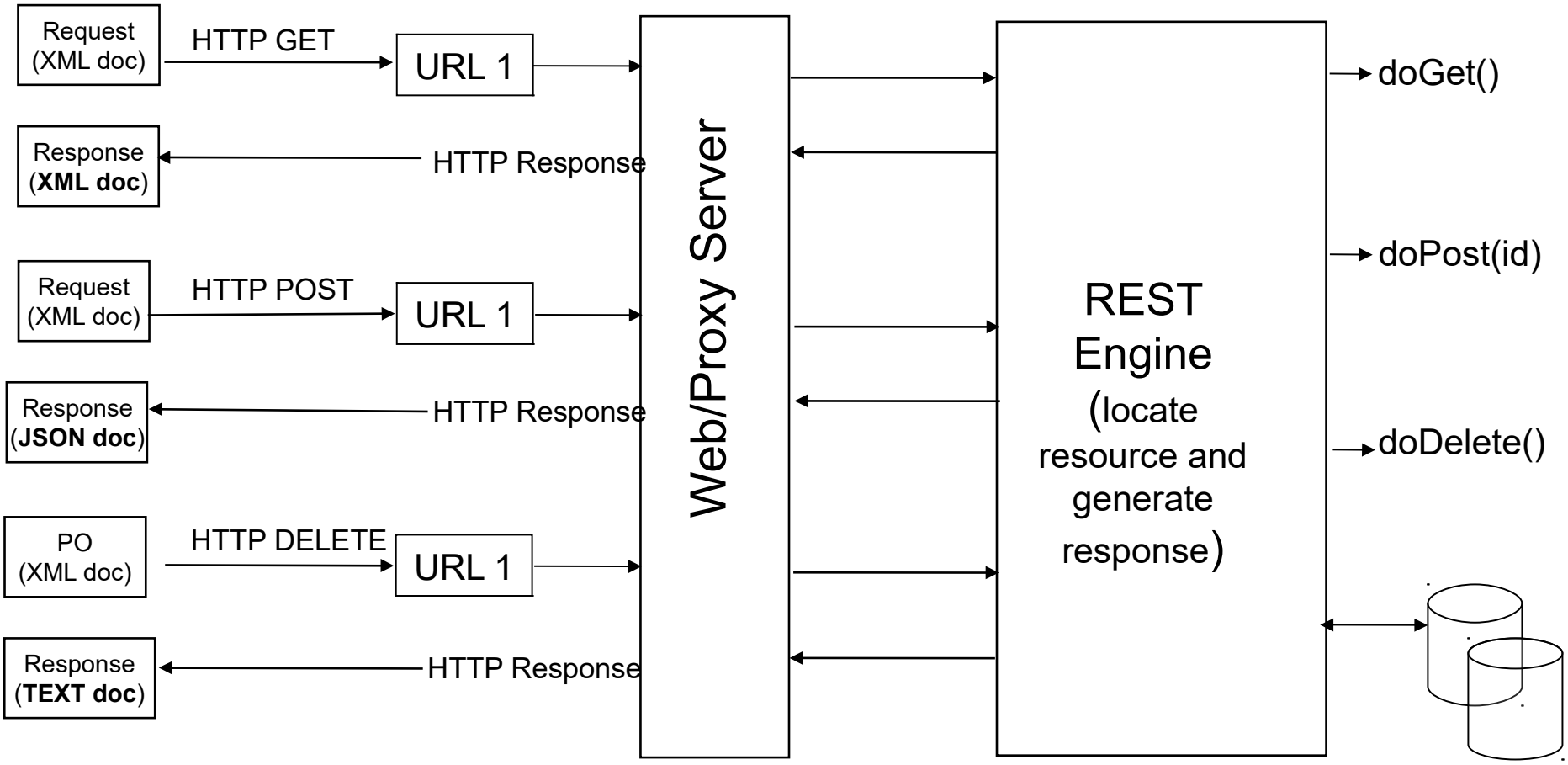
- REST : un style d'architecture logicielle pour les systèmes distribués, tels que le www. Il est axé sur les ressources.
- Dans une architecture REST, vous avez généralement un serveur REST qui permet d'accéder aux ressources et un client REST qui accède et modifie les ressources REST .
- REST est conçu pour utiliser un protocole de communication apatride, généralement HTTP .
- REST permet aux ressources d'avoir des représentations différentes, par exemple texte , XML , JSON , etc.
  Le client REST peut demander une représentation spécifique via le protocole HTTP (négociation de contenu) .

Representational State Transfer

- **Le client fait référence à une ressource Web à l'aide d'une URL. Une représentation de la ressource est retournée (dans ce cas comme un document HTML) .**
- La représentation (par exemple, jee7.html) place l'application cliente dans un état. Le résultat du client traversant un lien hypertexte dans une autre ressource jee7.html est consultée. La nouvelle représentation place l'application cliente dans un autre état > Représentation State Transfer !



Architecture REST



REST : méthodes HTTP

- **GET :** définit un accès en lecture à la ressource sans modification (sans effets secondaires = “idempotent”).
- **PUT :** crée une nouvelle ressource, doit également être idempotent .
- **DELETE :** Supprime les ressources. Les opérations sont idempotent.
  Elles peuvent se répéter sans aboutir à différents résultats.
- **POST :** met à jour une ressource existante ou crée une nouvelle ressource.

REST - Exemple

- URI de base du service web : *http://example.com/resources/*

Ressource	GET	PUT	POST	DELETE
URI d'une collection **http://example.com/resources/**	Liste les URIs et (peut-être) les détails des membres de la collection.	Remplace la collection entière par une autre collection.	Crée une nouvelle entrée dans la collection. La nouvelle URL est assignée et généralement retournée par l'opération.	Supprime la collection entière.
URI d'un élément **http://example.com/resources/item17**	Récupère une représentation du membre adressé de la collection (représentation exprimée dans un média type).	Remplace le membre adressé de la collection. S'il n'existe pas, il est créé.	(Non utilisée) Traite le membre adressé comme une collection et crée une nouvelle entrée.	Supprime le membre adressé de la collection.

JAX-RS

- Java API for RESTful Web Services :

  - API simplifiant le développement et le déploiement de web services.

  - Fournit les interfaces nécessaires à l'implémentation de Web Services REST.

  - API basée sur l'utilisation d'annotations

- API intégrée dans Java EE 6 (mise à jour dans Java EE 7) : aucune configuration supplémentaire nécessaire.
- Implémentations : **GlassFish Jersey (RI)**, Restlet, JBoss RESTEasy, Project Zero.

Annotations JAX-RS

- **@Path** : spécifie du chemin relatif pour la classe ou la méthode de la ressource.
- **@GET**, **@PUT**, **@POST**, **@DELETE** et **@HEAD** : spécifient le type de requête HTTP pour la ressource.
- **@Produces** : spécifie le type MIME de la réponse.
- **@Consumes** : spécifie le type MIME de la requête.

- Annotations de paramètres de méthodes @*Param :

  @PathParam : binding d'un paramètre avec un segment du path.

  @QueryParam : binding d'un paramètre avec la valeur d'un paramètre d'une requête HTTP.

  @MatrixParam : binding d'un paramètre avec la valeur d'un paramètre HTTP matrix.

  @HeaderParam : binding d'un paramètre avec la valeur d'un header HTTP.

  @CookieParam : binding d'un paramètre avec la valeur d'un cookie.

  @FormParam : binding d'un paramètre avec la valeur d'un champs de formulaire.

  @DefaultValue : spécifie la valeur par défaut d'un binding si la clé est inexistante.

- **@Context** : retourne le contexte global d'un objet (Ex. :@Context HttpServletRequest request)

Implémentation Jersey Configuration


```
<servlet>
   <servlet-name>jersey-servlet</servlet-name>
   <servlet-class>
        org.glassfish.jersey.servlet.ServletContainer
   </servlet-class>
   <init-param>
     <param-name>javax.ws.rs.Application</param-name>
     <param-value>fr.dawan.projet1.ws.MyApplication</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
   <servlet-name>jersey-servlet</servlet-name>
   <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>
```

Implémentation Jersey Configuration (2)

```java
@ApplicationPath("rest")
public class MyApplication extends ResourceConfig {

    public MyApplication() {
        packages("fr.dawan.projet1.ws");

        //support de l'upload
        register(MultiPartFeature.class);

        //properties du serveur
        property(ServerProperties.TRACING, "ALL");
    }
}
```

EndPoint Serveur

```
@Path("/bases")
public class BasesWS {
    @PostConstruct
    public void init(){ … }

    @PreDestroy
    public void destroy(){ … }

    @Context
    private UriInfo uriInfo;

    @GET
    @Path("/{param}")
    public Response getMsg(@PathParam("param") String msg) {
        …
        return Response.status(201).entity(result).build();
    }
}
```

EndPoint Serveur

```
@Path("/compagnie")
public class CompagnieWS {
    @GET
    @Path("/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public CompagnieAerienne getCompagnieJson(@PathParam("id") long id) { … }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public GenericEntity<List<CompagnieAerienne>> getlisteJson() {
        …
        GenericEntity<List<CompagnieAerienne>> entity
                = new GenericEntity<List<CompagnieAerienne>>(lc) {};
        return entity;
    }

    @POST
    @Path("/post")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response post(CompagnieAerienne compagnie) { … }


}
```

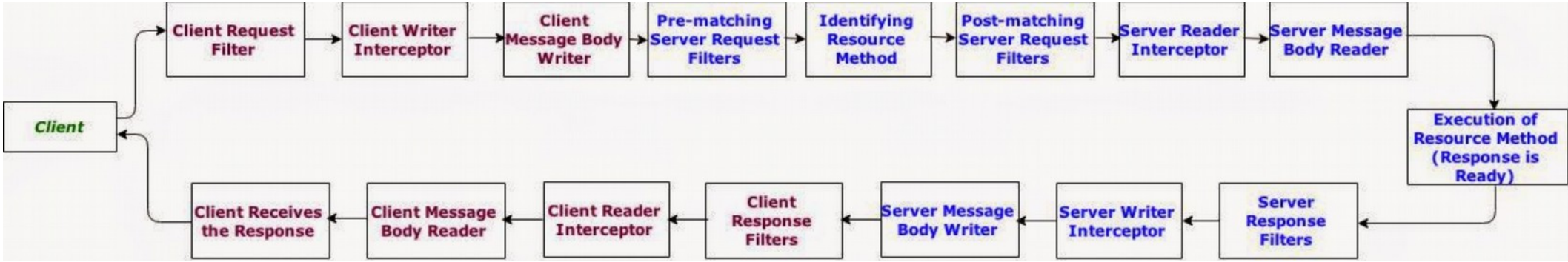
Client Java

```
URL url = new URL("http://localhost:8080/projetRS/rest/voyage");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

conn.setRequestMethod("GET");

conn.setRequestProperty("Accept", "application/xml");

if (conn.getResponseCode() != 200) {

   throw new RuntimeException("Failed : HTTP error code : "
                              + conn.getResponseCode());

}

InputStream result = conn.getInputStream();
…
conn.disconnect();
```

Client Jersey

```
final Client client = ClientBuilder.newClient().register(JacksonFeature.class);

final Response res
    = client.target("http://localhost:8080/projetRS/rest/voyage").request().get();

if (res.getStatus() != 200) {

   throw new RuntimeException("Failed : HTTP error code : " + res.getStatus());

}

GenericType<List<Voyage>> voyageListType = new GenericType<List<Voyage>>() {};

List<Voyage> lv =  res.readEntity(voyageListType);
…

client.close();
```

Intercepteurs et filtres



Mapping des exceptions

- On peut créer un ExceptionMapper pour renvoyer une exception dans la réponse et pouvoir la capturer sous format JSON ou autre :

```
@Provider
public class MyExceptionMapper implements ExceptionMapper<Throwable>{

   @Override
   public Response toResponse(Throwable exception) {

      //sérialisation d'un objet dans la réponse
   }

}
```

Sécurité de service web

Sécurité : Menaces « STRIDE »

Spoofing identity : un utilisateur non accrédité usurpe l'identité d'un utilisateur valide de l'application

Tampering with data : un utilisateur détruit ou modifie les informations sans autorisation

Repudiability : la possibilité qu'un utilisateur puisse nier avoir effectué telle ou telle opération

Information disclosure : des données confidentielles sont rendues visibles à des utilisateurs non accrédités ;

Denial of service : l'application est rendue indisponible

Elevation of privilege : un utilisateur dispose un niveau d'accès à l'application supérieur à celui qui devrait lui être accessible

Sécurité : Préoccupations

Authentification – identité

Qui est l'appelant ?

Qui prouve que c'est bien le bon appelant ?

Autorisation – contrôle d'accès

Quelle opérations peut effectuer l'appelant ?

Est-ce que l'appelant a la permission d'appeler cette opération ?

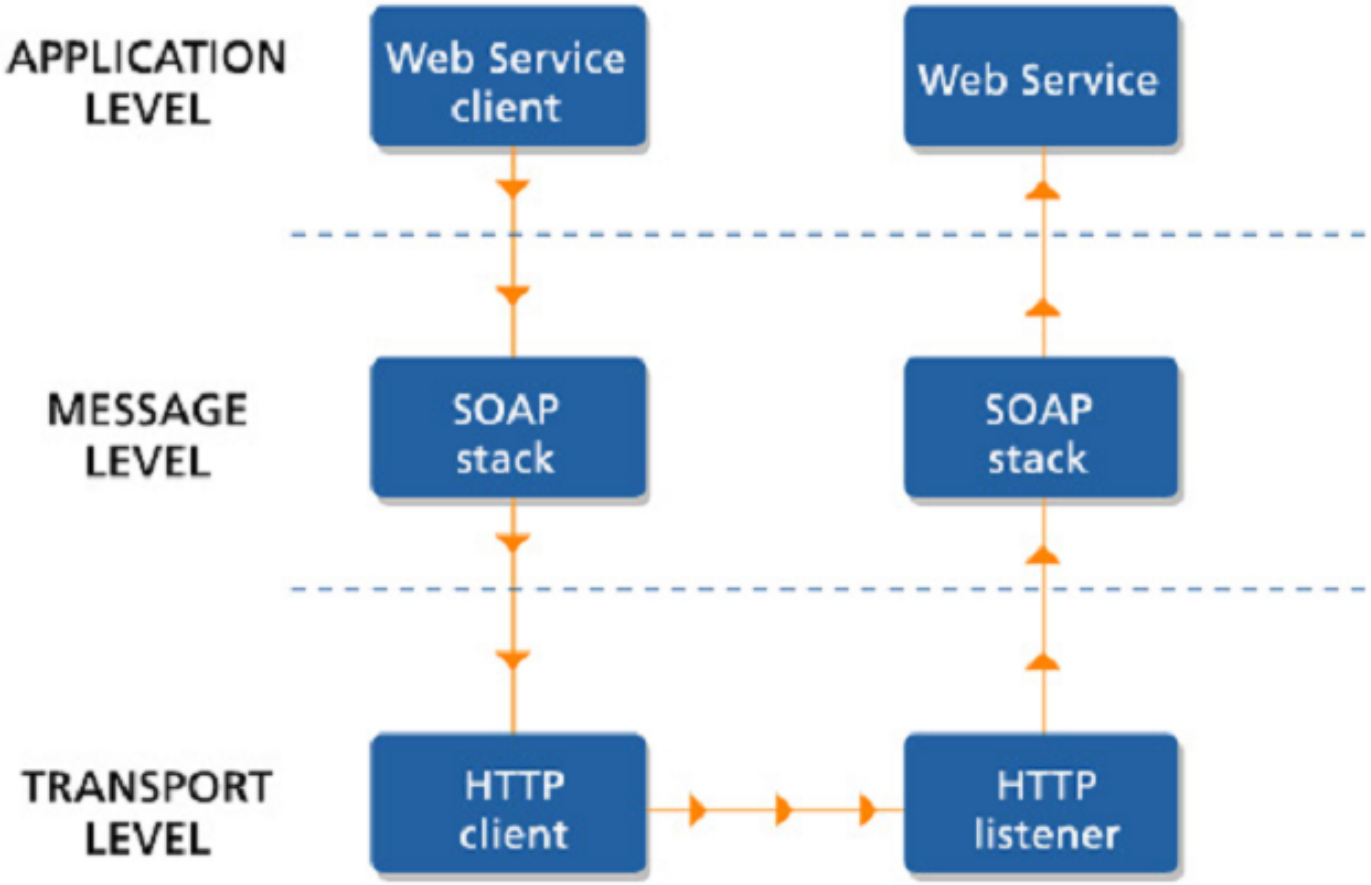
Confidentialité – cryptage

Comment chiffrer vos messages pour éviter l'interception ?

Intégrité

Comment maintenir l'intégrité entre l'émetteur et le récepteur ?

Sécurité : Niveau d'interaction



Sécurité niveau Transport

Utilisation de technologies web 3-tiers avec HTTP et SSL

Authentification

Schémas d'authentification HTTP : Basic, Digest

Certificats côté client : SSL

Autorisation

Contrôle d'accès aux URLs (politiques de sécurité des applications webs JEE)

Confidentialité

Connections cryptées SSL

Intégrité

Eviter l'interception des données :Point-to-point SSL encryption

Sécurité niveau Message

La sécurité des messages XML échangés s'appuie sur l'ajout d'éléments dans l'entête SOAP :

Authentification

- SSO (single sign-on) header tokens
- SAML authentication assertions

Autorisations

- SSO session details
- SAML attribute assertions

Confidentialité

- XML Encryption specification

Intégrité des données

- XML Digital Signatures specification

Sécurité niveau application

Authentification

Messages spécifiques pour l'authentification et maintien d'un domaine de sécurité

Autorisation

Ajout de points de contrôle

Confidentialité

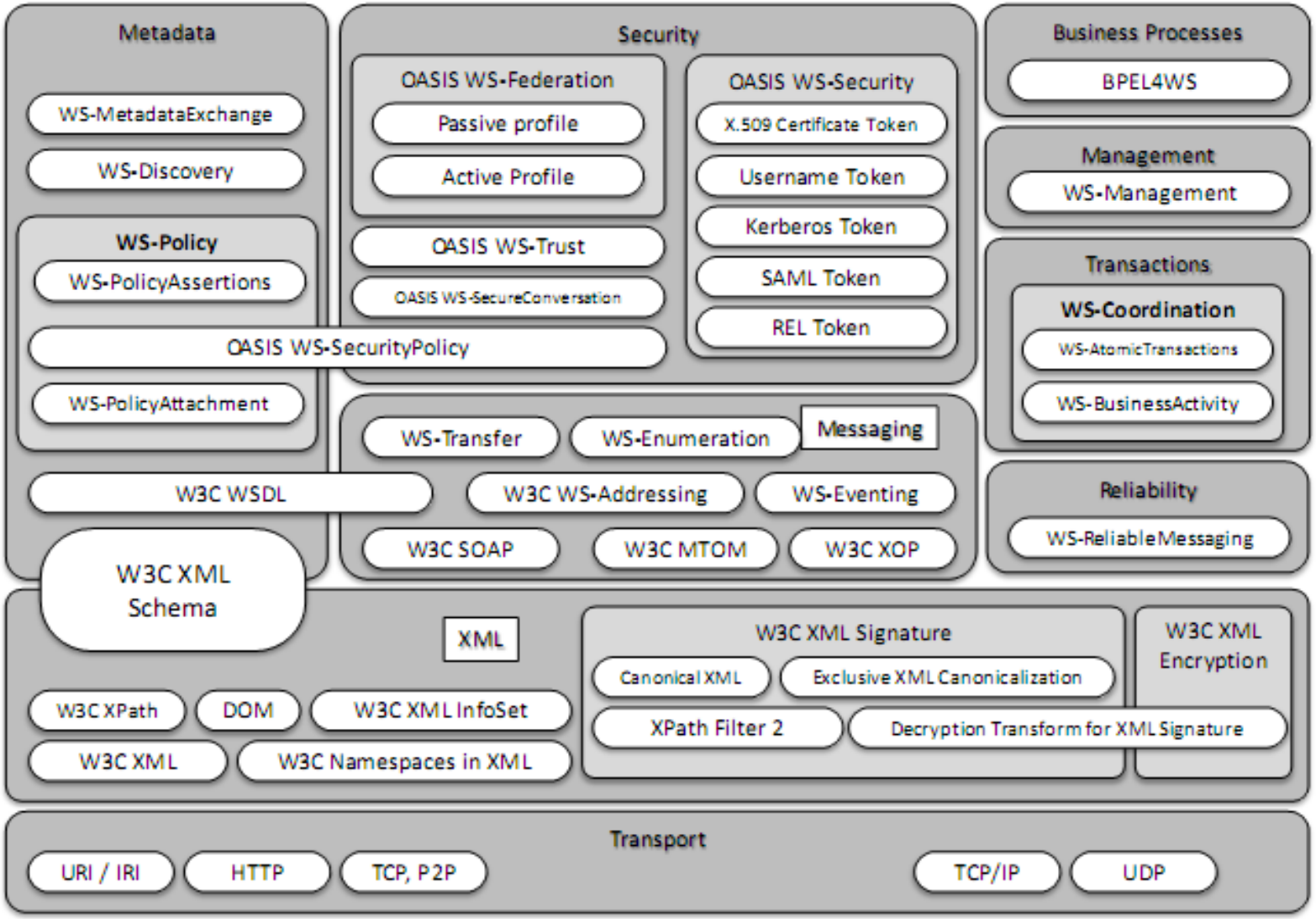
Application d'un mécanisme de cryptage des données

Intégrité

XML Digital Signature

Ajout d'information spécifiques (MD5 hash par exemple)

Standards de sécurité



Survol de standars

Quelques standards :

- XML Digital Signatures
- XML Encryption
- SAML
- WS-Security
- WS-Trust
- WS-Policy
- WS-Secure Conversation
- WS-Security Policy

Plus d'informations sur http://www.dawan.fr

Contactez notre service commercial au
09.72.37.73.73 (prix d'un appel local)