



Global Knowledge®

Formation Service Web Java
GKJAVWEB

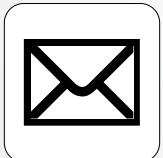
Logistique



Pause en milieu de session



**Vos questions sont les bienvenues.
N'hésitez pas !**



**Feuille d'évaluation à remettre remplie en fin de
session**



Merci d'éteindre vos téléphones

Sommaire

- **Chapitre 1** : INTRODUCTION
- **Chapitre 2** : Technologies des services Web
 - Exemple pratique de création et de déploiement d'un Web service SOAP et de son client
- **Chapitre 3** : Service Web REST
- **Chapitre 4** : Développement de service web REST avec JAX-RS
 - Etude de cas : *Développement d'un service RESTful retournant un flux JSON. + Invocation du service et parsing du résultat en Java.*
- **Chapitre 5** : la gestion des Persistance dans un WEB SERVICE REST
 - Etude de cas : Client : JAX-RS, JaxB, JAVA
- **Chapitre 6** : Sécurité des Web services SOAP VS REST

Chapitre 1

INTRODUCTION

Objectif du chapitre

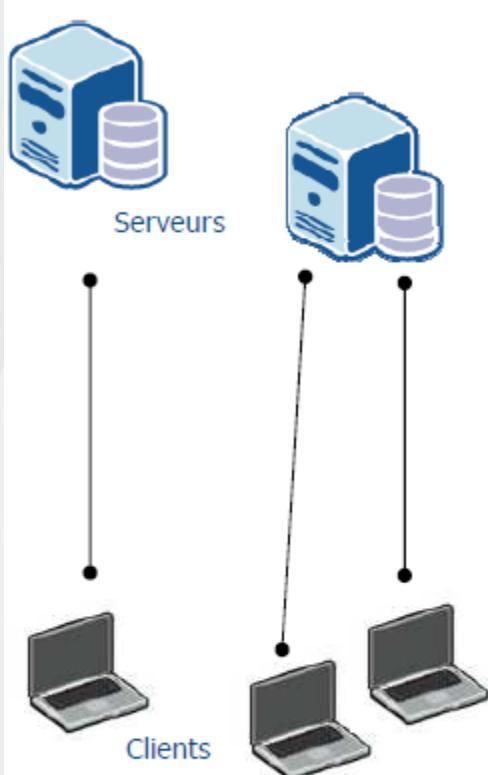
- L'architecture Orientée Service SOA
- Définition d'un service
- Services Web étendus et REST
- Plateformes de développement

L'architecture Orientée Service SOA

Vers une architecture SOA

- Une application distribuée est définie par un ensemble de composants
 - Collaborent pour l'exécution de tâches communes
 - Distants géographiquement
 - Interconnectés via un réseau de communication
 - Hétérogènes
- Solutions qui ont fait leur preuve
 - DCOM, CORBA, EJB, RMI, .Net Remoting, ...
- Faiblesses de ces solutions
 - Format de représentation données spécifiques
 - Interopérabilité si les composants utilisent la même solution
 - Protocole de transport spécifique nécessite une configuration réseau

Vers une architecture SOA



**Architecture
Client / Serveur**



**Architecture
Fondée sur les
Applications Web**



**Architecture
Orientée Service**

SOA : Généralité

- SOA est l'acronyme de **Service Oriented Architecture** qui est traduit comme « Architecture Orientée Service »
- Le Service (ou Composant) désigne le fondement de ce modèle d'interaction entre applications
- Le paradigme SOA : **Chercher, Publier et Consommer**

Objectif du chapitre

- L'architecture Orientée Service SOA
- **Définition d'un service**
- Services Web étendus et REST
- Plateformes de développement

Définition d'un service

SOA : Concepts de Service

- Qu'est-ce qu'un Service ?
 - « Un Service est un composant logiciel distribué, exposant les fonctionnalités à forte valeur ajoutée d'un domaine métier » [XEBIA BLOG : 2009]
- Huit aspects caractérisant un Service
 - Contrat standardisé
 - Couplage lâche
 - Abstraction
 - Réutilisabilité
 - Autonomie
 - Sans état
 - Découvrabilité
 - Composabilité

Service : Contrat Standardisé

- Contrat entre le fournisseur de service et le consommateur de service
- Trois types de contrat sont à distinguer
 - Lié à la syntaxe du service (opération, messages d'entrée, messages de sortie, ...)
 - Lié à la sémantique du service (définition de règles et de contraintes d'usage, ...)
 - Lié à la qualité de service (temps de réponse attendu, procédures en cas de panne, temps de reprise après interruption, ...)
- S'appuie sur des standards d'interopérabilité pour faciliter le dialogue (exemple : WSDL)

Service : Couplage lâche

- L'échange entre le fournisseur de service et le consommateur doit se faire à travers des messages (couplage lâche vis-à-vis de son environnement)
- L'utilisation d'une orchestration évite que les services aient besoin de connaître les autres services

Service : Abstraction

- Le contrat du service ne doit contenir que les informations pertinentes à son invocation
- Fonctionnement du service dit en « boîte noire »
 - Seul le contrat exposé au consommateur du service est connu
 - Le fonctionnement interne du service ne doit pas être visible
 - Logique métier
 - Implémentation
- Il est par conséquent important d'assurer la **prédictabilité** d'un service
 - Pas de variation dans le comportement et dans la réponse d'un service lors de la réception d'une requête

Service : Réutilisabilité / Découvrabilité

- Un service doit être accessible depuis un entrepôt ou un annuaire pour faciliter sa découverte
- Le fournisseur de services a la charge de déposer et de mettre à jour ses services depuis l'annuaire
- Le service est enrichi par un ensemble de métadonnées pour faciliter la recherche du consommateur de services
- S'appuie sur des standards (UDDI, ebXML)
- D'après la gouvernance SOA
 - Un service est défini avec l'intention d'être réutilisé

Service : Autonomie / Sans état

- Un service doit disposer
 - de l'ensemble des informations nécessaires à son exécution
 - ne doit dépendre d'aucun service externe (couplage lâche)
- Garantir l'autonomie d'un service permet de s'assurer de sa prédictabilité
- Un service doit être sans état de façon à minimiser la consommation de ressources
 - Maintenance : rend complexe la composition de services
 - Performance : gourmand en ressources système

Service : Composabilité

- Un service doit fonctionner de manière modulaire et non pas intégrée
- Assurer la décomposition d'un service complexe en sous services plus simples entre eux (garantie l'autonomie)
- S'inscrire dans une logique de composition de services à travers l'utilisation de l'orchestration (couplage lâche)
- L'orchestration favorise l'indépendance des services et assure que des services n'appellent pas directement d'autres services

Client/Serveur VS SOA



- Intra-entreprise
- Limitée à un sous ensemble de langages de programmation
- Procédurale
- Protocole de transport propriétaire
- Fortement couplé
- Traitement efficace

Architecture Client / Serveur

VS

- Entre Entreprises
- Indépendance du langage de programmation
- Pilotée par les messages
- Possibilité de choisir le protocole de transport
- Faiblement couplé
- Traitement plus lourd

Architecture Orientée Service



Application Web VS SOA



- Interaction Programme / Utilisateur
- Intégration statique des composants
- Service monolithique
- Référencement via des annuaires de sites non standardisés

Architecture Fondée sur les Applications Web

VS

- Interaction Programme / Programme
- Intégration dynamique des services
- Décomposition en sous service avec possibilité de réutilisation
- Annuaires standardisés

Architecture Orientée Service



Objectif du chapitre

- L'architecture Orientée Service SOA
- Définition d'un service
- **Services Web étendus et REST**
- Plateformes de développement

Services Web étendus et REST

Solutions pour une SOA

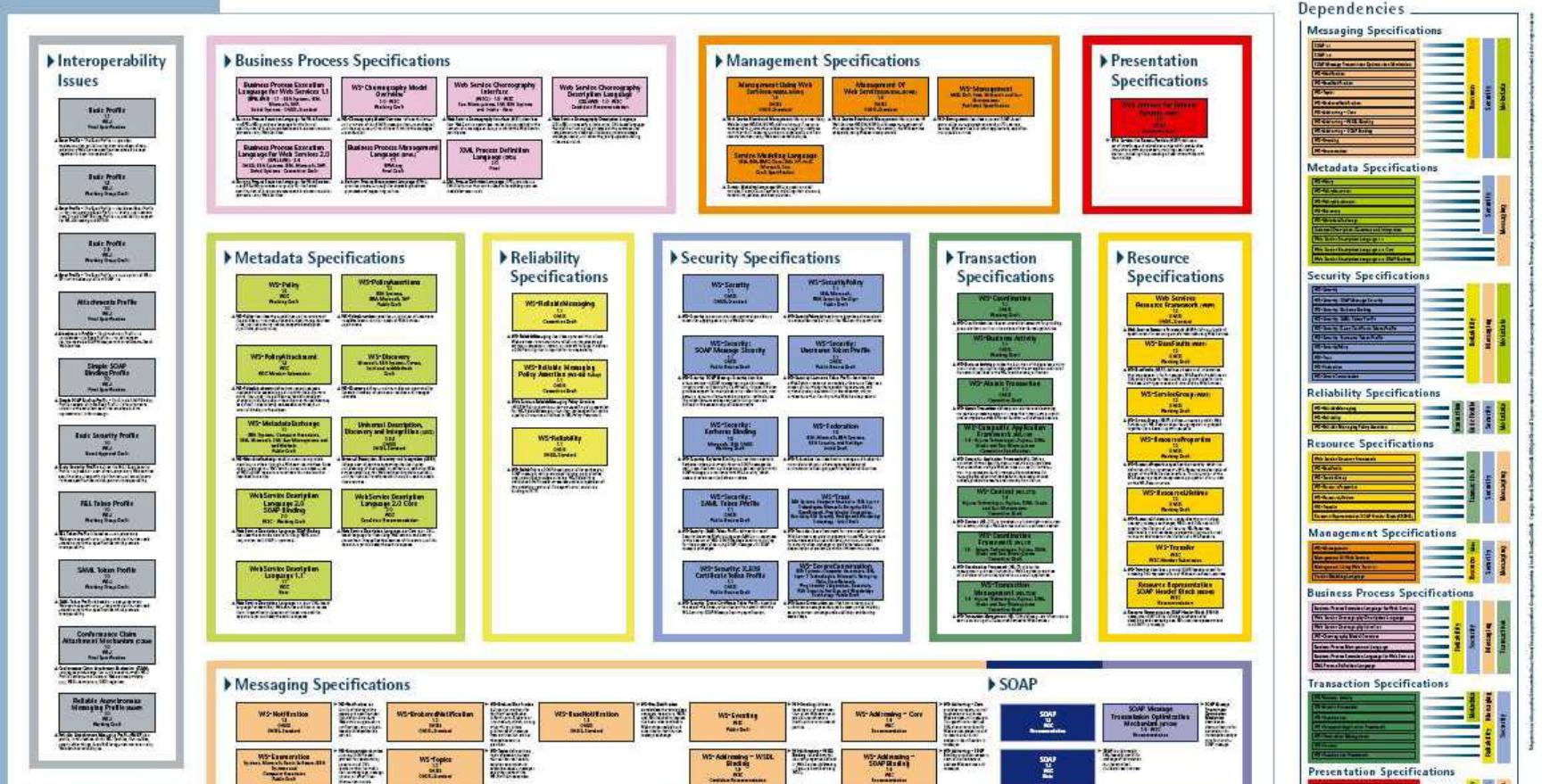
- Plusieurs solutions technologiques sont adaptées pour développer une architecture orientée service
 - Services Web (le plus courant)
 - Framework OSGi, ...
- Un point sur OSGi (**Open Service Gateway Initiative**)
 - Spécification définie par l'OSGi Alliance (<http://www.osgi.org>)
 - Longtemps exploité dans le monde de l'embarqué, utilisé dans les serveurs (GlassFish 3, Spring DM) et application (Eclipse)
 - Concepts ...
 - **Dynamique** : installé, arrêté, mise à jour, désinstallé
 - **Découvrabilité** : registre des services
 - **Abstraction** : gestion détaillée des classes à exposer
 - Pour aller plus loin :
<http://mbaron.developpez.com/eclipse/introplugin>

Services Web : réponses au SOA

- Les Services Web sont basés sur les protocoles et les langages du Web
 - HTTP, XML, TCP/IP pour la couche réseau
 - Ne nécessite pas une configuration réseau particulière
- Les Services Web sont auto-suffisants puisqu'ils contiennent toutes les informations à leurs utilisations
 - Chercher, publier et consommer
 - Annuaire, contrat de fonctionnement et un client pour les consommer
- Les Services Web sont modulaires
 - Une application doit être décomposée en un ensemble de services
 - Utilisation d'une orchestration
- Les Services Web peuvent être définis par des standards
 - OASIS, W3C, WS-I et IETF

Services Web : réponses au SOA

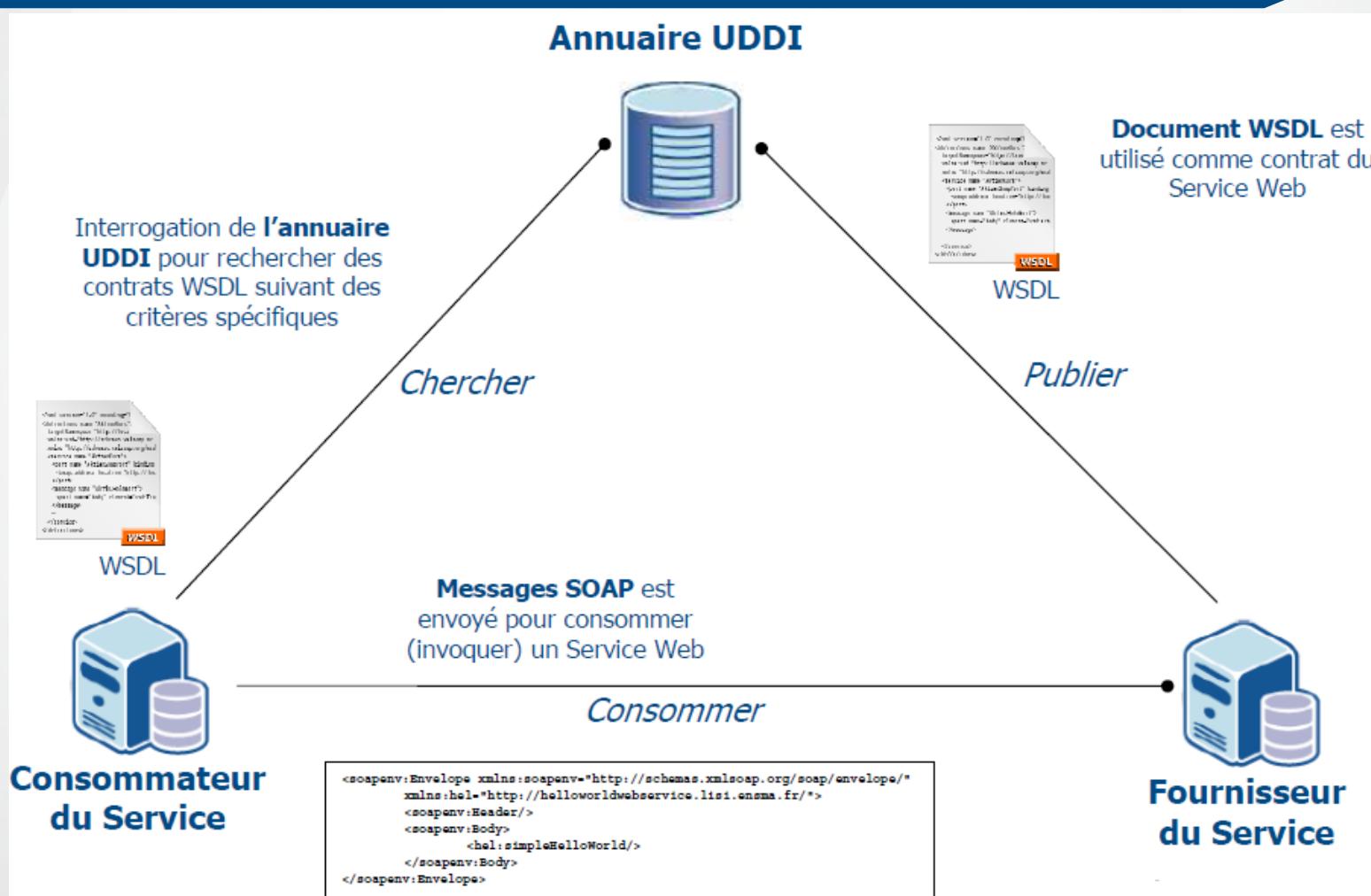
Web Services Standards Overview



Services Web : technologies disponibles

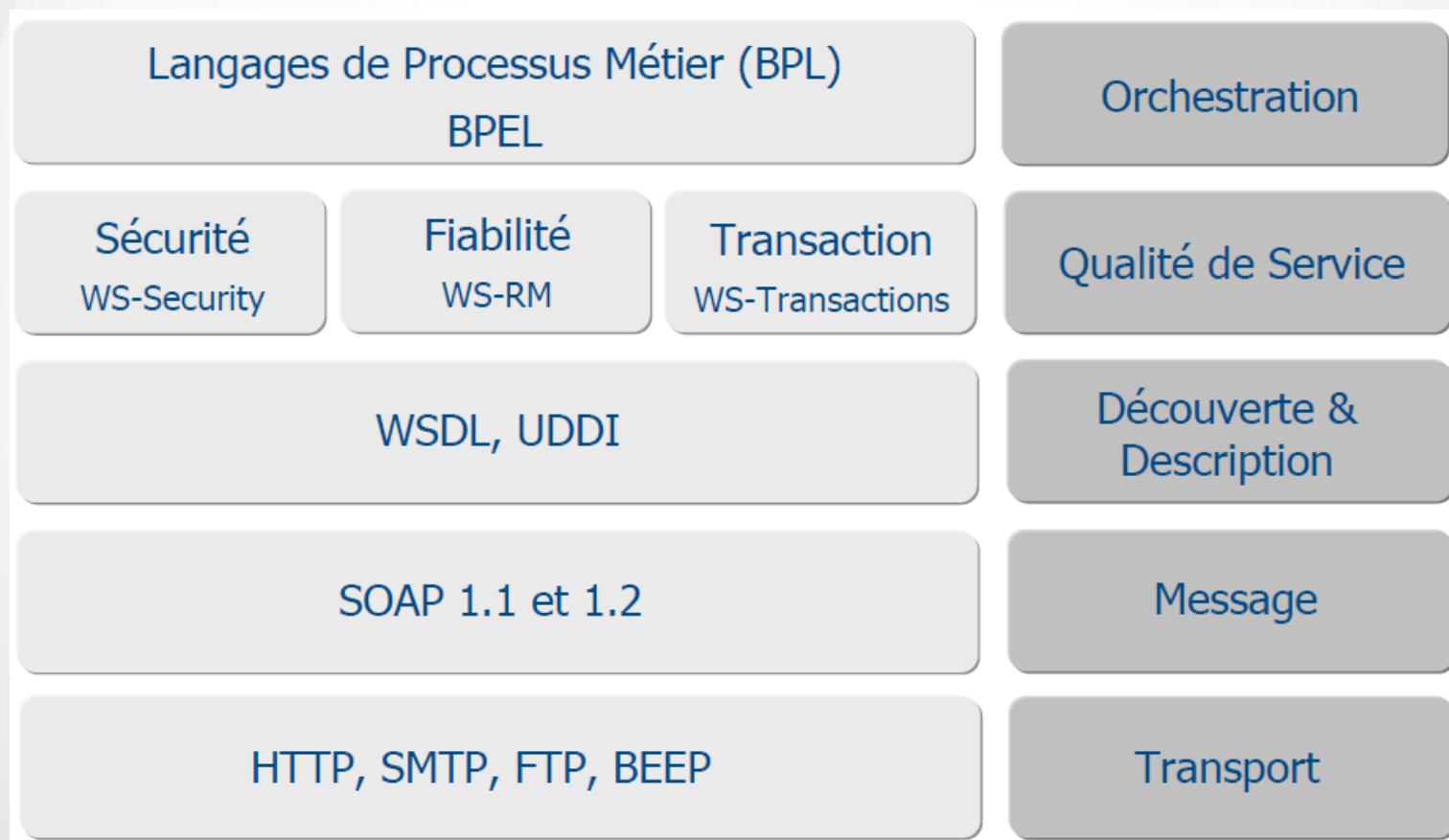
- Deux familles de Services Web se distinguent actuellement
- **Services Web « étendus »**
 - S'appuie sur des standards UDDI / WSDL / SOAP
 - Annuaire de Services Web : UDDI
 - Contrat : WSDL
 - Consommer : SOAP
- **Services Web REST (Representational State Transfer)**
 - Défini par la thèse de Roy Fielding en 2000
 - Utilise directement HTTP au lieu d'utiliser une enveloppe SOAP
 - URI est utilisé pour nommer et identifier une ressource
 - Méthodes HTTP (POST, GET, PUT et DELETE) sont utilisées pour effectuer les opérations de base CRUD

Services Web étendus



Services Web étendus

➤ Pile des standards pour les Services Web étendus



Services Web REST

- Exploités pour les Architectures Orientées Données (DOA)
- REST n'est pas un standard, il n'existe pas de spécification W3C définissant une spécification
- REST est un style d'architecture basé sur un mode de compréhension du Web
- REST s'appuie sur des standards du Web :
 - Protocole HTTP
 - URLs
 - Formats de fichiers
 - Sécurisation via SSL

Services Web REST

- Pile des protocoles et langages pour les Services Web REST

Langages de Processus Métier (BPL)
BPEL

Orchestration

HTTP Basic, SSL / TLS

Qualité de Service /
Sécurité

WADL, ATOM, ...

Découverte &
Description

MIME Types (Text, JSON, XML, ...)

Message

HTTP, FTP, ...

Transport

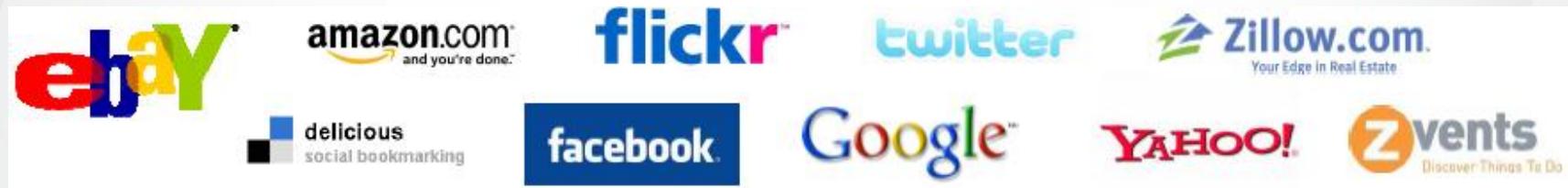
Objectif du chapitre

- L'architecture Orientée Service SOA
- Définition d'un service
- Services Web étendus et REST
- **Plateformes de développement**

Plateformes de développement

Les fournisseurs de Services Web

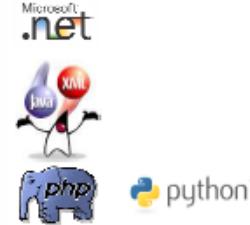
- Deux types de fournisseurs sont à distinguer
 - Fournisseurs de Services Web « Orientés Web » (public)
 - Fournisseurs de Services Web « Entreprise » (privé)
- Les grands noms du Web sont présents et leurs services sont accessibles
 - Amazon, eBay, Delicious, Facebook, Flickr, Google, Twitter, WeatherBug, Yahoo, Zillow, Zvents



- Oui mais ...
 - Pratiquement tous les fournisseurs de Services Web exploitent l'architecture REST (besoins de performance)
 - Certains (comme Google) ont arrêtés les Services Web étendus
 - eBay propose encore des Services Web étendus

Les Plateformes de développement

- La grande majorité des plateformes de développement fournissent le support de Services Web (outils et APIs)
 - Plateforme .NET
 - Plateforme Java
 - Plateforme PHP, C++, Python, ...
- Les outils permettent de
 - Manipuler des messages SOAP
 - Manipuler des données au format XML
 - Mapping XML / Classe (Marshall, Unmarshall)
 - Accéder à la couche HTTP
- Dans ce cours, nous utiliserons la plateforme Java
 - Outilée, gratuite, accessible, légère, respect des standards



Les Plateformes de développement

- Dans cette formation nous allons utiliser essentiellement :
 - **JDK 1.7 (Ou Plus)**
 - **Maven 3** : outil pour la gestion et l'automatisation de production des projets JAVA
 - **Eclipse 4.3 Kepler (Ou Plus)** : environnement de développement logiciels
 - **Jersey 1.X** : framework open source écrit en java permettant de développer des services Web selon l'architecture REST

Sommaire

- **Chapitre 1 : INTRODUCTION**
- **Chapitre 2 : Technologies des services Web**
 - Exemple pratique de création et de déploiement d'un Web service SOAP et de son client
- **Chapitre 3 : Service Web REST**
- **Chapitre 4 : Développement de service web REST avec JAX-RS**
 - Etude de cas : *Développement d'un service RESTful retournant un flux JSON. + Invocation du service et parsing du résultat en Java.*
- **Chapitre 5 : la gestion des Persistance dans un WEB SERVICE REST**
 - Etude de cas : Client : JAX-RS, JaxB, JAVA
- **Chapitre 6 : Sécurité des Web services SOAP VS REST**

Chapitre 2

Technologies des

services web

Objectif du chapitre 2

- Fondation des services Web (les protocoles Internet)
 - URI, URL, URN
 - MIME
 - HTTP 1.1
 - SMTP
 - les protocoles SSL et TLS
- Fondation des services Web (les technologies XML)
 - XML1.0
 - XML namespaces
 - Xlink
 - XML Base
 - XPath
 - XML Schema
 - L'interface DOM
 - Les analyseurs syntaxiques XML
- Principe et objectifs des Services Web SOAP
 - SOAP par l'exemple HelloWorld Service
 - Structure message SOAP
 - SOAP et le transport http
 - Client SOAP UI
 - WSDL
- Exemple pratique de création et de déploiement d'un Webservice SOAP et de son client

Fondation des Services Web

Les protocoles Internet

- On va voir dans l'ordre :
 - URI, URN, URL, trois sigles qui se rapportent au mécanisme utilisé par le Web pour identifier et/ou localiser une ressource ;
 - MIME, technologie permettant de véhiculer des objets de toute sorte sur Internet, très importante dans la mesure où de nombreux autres protocoles l'utilisent, et en particulier SMTP et HTTP ;
 - HTTP/1.1, protocole réseau fondamental en tant que tel, qui est en outre le moyen de transport des messages le plus utilisé pour les services Web ;
 - SMTP, alternative à HTTP en tant que moyen de transport des services Web.

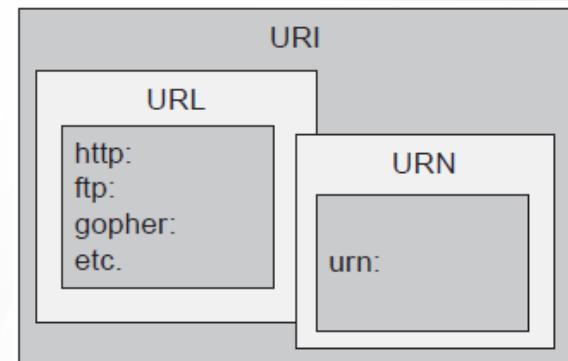
URI, URL, URN

➤ On va voir dans l'ordre :

- URI (Uniform Ressource Identifier) c'est un mécanisme permettant aux utilisateurs et aux programmes de nommer et de localiser les ressources Web (information, documents, programmes, services)
- L'URI est utilisé par les protocoles de base du Web (HTTP, FTP) et aussi par la plupart des techniques récentes telles que :
 - Les espace de noms (namespaces)
 - XML
 - SMIL (Synchronized Multimedia Integration Language)
 - SVG (Scalable Vector Graphic)

URI, URL, URN

- Les URI sont classés en trois groupes :
 - ceux qui permettent de localiser des ressources sur un réseau, appelés URL (Uniform Resource Locator) ;
 - ceux qui permettent d'identifier et de nommer des ressources de manière unique et persistante, appelés URN (Uniform Resource Name) ;
 - et ceux qui permettent à la fois de localiser et d'identifier une ressource.



URI, URL, URN : Syntaxe

- Un URI n'utilise qu'un jeu restreint de caractères (chiffres, lettres et quelques symboles) car il doit pouvoir être utilisé tout aussi bien avec des moyens de communication informatisés que non informatisés (papier, etc.). Il est constitué
 - de caractères réservés (« ; », « / », « ? », « : », « @ », « & », « = », « + », « \$ », « , ») qui servent de délimiteurs ;
 - de chaînes de caractères codés en ASCII US ou à l'aide de séquences d'échappement commençant par le signe « % » (par exemple : « %2D » qui est le caractère « - »).

URI, URL, URN : Composition

- Un URI est toujours constitué de la manière suivante :

<modèle>:<chemin ou partie spécifique du modèle>

- Le modèle ou schéma définit l'espace de noms de l'URI et peut donc introduire des restrictions dans la syntaxe ou la sémantique du chemin. En d'autres termes, la syntaxe d'un URI dépend du modèle utilisé.

- Il existe néanmoins une syntaxe générique des URI, que voici :

<modèle>://<autorité><chemin> ?<requête>

- Par exemple :
 - <ftp://www.monsite.com/pub>
 - <http://www.monsite.com/index.htm?param1=1¶m2=essai>
 - <file:///c:/program%20files/monfichier.txt>

- Les modèles qui impliquent l'utilisation d'un protocole sur IP utilisent la syntaxe suivante pour décrire la partie autorité :

<utilisateur>@<hôte>:<port>

- L'exemple suivant ouvre une connexion FTP sur le site www.monsite.com sur le port 8000, login « anonymous », mot de passe « nopwd » :
 - <ftp://anonymous:nopwd@www.monsite.com:8000>

- La syntaxe de la partie « requête » dépend du modèle, mais une forme commune est la suivante :

<paramètre1>=<valeur>&<paramètre2>=<valeur> ...

URI, URL, URN : Composition

- Comme nous l'avons dit, un URN a pour objectif de nommer une ressource indépendamment de sa localisation.
- Un modèle (*schéma*) spécifique a été défini par le RFC2141 et est utilisé en tant que standard pour identifier des ressources sur le Web.
- Sa syntaxe est la suivante :

urn:<espace de noms><identifiant au sein de l'espace de noms>

- Par exemple :

urn:MonEspace:MonIdentifiant

MIME

- MIME (Multipurpose Internet Mail Extensions) est un standard Internet proposé par l'IETF sous les références RFC2045, RFC2046, RFC2047, RFC2048 et RFC2049.
- Cette spécification a pour objectif :
 - de permettre l'échange sur Internet de messages dont le contenu textuel est codé avec un autre jeu de caractères que l'ASCII US sur 7 bits (utilisé historiquement sur le Web) ;
 - de définir un ensemble extensible de formats binaires permettant de transporter dans ces messages tout type de contenu non textuel (audio, vidéo, HTML, etc.) ainsi que des contenus mixtes (« multipart ») ;
 - de permettre le codage des informations d'en-têtes de ces messages avec un autre jeu de caractères que l'ASCII US.
- En d'autres termes, c'est un standard qui permet d'échanger des messages *multimédia* sur Internet entre des systèmes informatiques hétérogènes.

MIME : Description

- MIME introduit des lignes d'en-têtes dans les messages :
 - une ligne « MIME-Version: 1.0- » ;
 - une ligne « Content-Type » qui précise le format du message ;
 - une ligne « Content-Transfer-Encoding » qui précise l'encodage de ce type de contenu.
- Deux encodages fondamentaux sont utilisés par les messages MIME :
 - Quoted-Printable (ou QP), qui permet de coder n'importe quel jeu de caractères sur 7 bits (par souci de compatibilité) ;
 - Base64, qui permet de coder n'importe quel fichier binaire.
- Ces deux encodages ne sont pas obligatoires mais fortement recommandés.

MIME : Description

- Le format du message est défini par un type MIME, et l'implémentation de chacun de ces types est du ressort des applications informatiques.
- Un type MIME est identifié par un label type/sous-type; paramètres, ce qui permet d'organiser les formats d'après des types de base :
 - text, image, audio, video, application ;
 - plus deux types composites : message et multipart.
- Cela permet ensuite de décliner ces types de base en fonction des besoins, comme : image/jpeg ou text/xml.
- Dans l'exemple suivant, le message contient du texte, utilise un jeu de caractères ISO-Latin-1 et est codé en QP :

MIME-Version: 1.0

...

Content-Type: text/plain; charset=**iso-8859-1**

Content-Transfer-Encoding: **quoted-printable**

...

HTTP 1.1

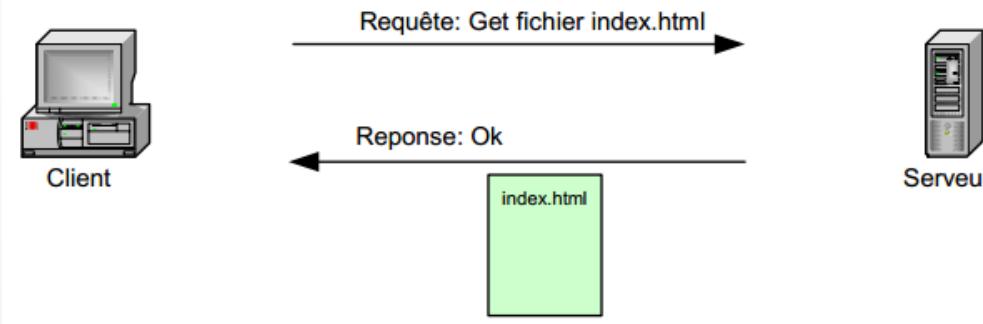
- HTTP (HyperText Transfer Protocol) est un standard Internet et utilisé sur le Web depuis 1990
- HTTP est un protocole application générique (couche 7 de l'ISO), qui permet de transférer des messages au format MIME entre un client et un serveur
- Il est largement utilisé par de nombreux types de clients (PC, PDA, CD-Rom, etc.), des moyens de transport variés (depuis les réseaux sans fil jusqu'aux liaisons optiques transocéaniques) et sur des architectures plus ou moins complexes composées de passerelles, de hiérarchies de caches, etc.

HTTP 1.1 : Présentation

- Le protocole HTTP utilise un jeu de *requêtes/réponses* entre un client, qui initie le dialogue, et un serveur. La communication peut être directe entre les deux acteurs mais elle peut également faire intervenir trois types d'intermédiaires, que voici :
 - **un proxy**, c'est-à-dire un agent qui transfère les messages vers le serveur après en avoir réécrit tout ou partie du contenu ;
 - **une passerelle**, c'est-à-dire un agent qui agit comme une surcouche pour un serveur sous-jacent utilisant un autre protocole ; cet agent se charge de traduire les messages pour permettre leur transfert vers ce serveur tiers ;
 - **un tunnel**, c'est-à-dire un relais qui se charge de transmettre le message entre deux points de connexion sans modification du message (à travers un intermédiaire tel qu'un pare-feu).
- En dehors des tunnels, tous les autres intermédiaires peuvent implémenter des fonctions de cache : il s'agit de garder localement une copie de la réponse tant que celle-ci est valide et de retourner cette réponse au client sans interroger à nouveau le serveur (ce qui amène un gain de performance et de trafic réseau).

HTTP1.1 : Présentation

- En matière de protocole, HTTP se place au dessus de TCP et fonctionne selon un principe de requête/réponse : le client transmet une requête comportant des informations sur le document demandé et le serveur renvoie le document si disponible ou, le cas échéant, un message d'erreur.



- Le HTTP est un protocole sans connexion et chaque couple requête/réponse et de ce fait indépendant
- Le port 80 est utilisé par défaut (443 pour le secure)

HTTP1.1 : Description requête

- Un message HTTP est soit la requête d'un client, soit la réponse d'un serveur (ou d'un intermédiaire).
- Une requête HTTP 1.1 est composée de :
 - une ligne de requête qui précise la méthode utilisée, l'URI auquel s'applique cette méthode et la version du protocole HTTP utilisé ;
 - zéro ou plusieurs champs d'en-têtes du type « champ:valeur ». Les en-têtes sont de type *général*, *requête* ou *entité*
 - une ligne vide ;
 - un corps de message MIME optionnel : la présence ou non de ce contenu dépend de la commande utilisée. La forme de ce corps de message dépend du type et de l'encodage utilisé (champs Content-Type et Content-Encoding).

HTTP1.1 : Description requête

- Les méthodes HTTP les plus fréquemment utilisées sont : GET, POST et HEAD
- Cinq autres méthodes sont cependant définies par la version 1.1 du protocole.
- Certaines de ces méthodes nécessitent , en général, une authentification du client.

Méthodes	1.0	1.1	Description
Get			Permet de demander un document
Post			Permet de transmettre des données (d'un formulaire par exemple) à l'URL spécifiée dans la requête. L'URL désigne en général un script Perl, PHP...
Head			Permet de ne recevoir que les lignes d'en-tête de la réponse, sans le corps du document
Options			Permet au client de connaître les options du serveur utilisables pour obtenir une ressource
Put			Permet de transmettre au serveur un document à enregistrer à l'URL spécifiée dans la requête
Delete			Permet d'effacer la ressource spécifiée
Trace			Permet de signaler au serveur qu'il doit renvoyer la requête telle qu'il la reçue
Connect			Permet de se connecter à un proxy ayant la possibilité d'effectuer du <i>tunneling</i>

HTTP1.1 : Description réponse

- Une réponse HTTP 1.1 est composée de :
 - une ligne de statut qui précise la version du protocole HTTP utilisée (en général HTTP 1.0), un code réponse numérique et une description textuelle ;
 - zéro ou plusieurs champs d'en-têtes du type « champ:valeur ». Les en-têtes sont de type *général*, réponse ou entité ;
 - une ligne vide ;
 - un corps de message MIME optionnel : la présence ou non de ce contenu dépend de la commande utilisée. La forme de ce corps de message dépend du type et de l'encodage utilisé (champs Content-Type et Content-Encoding).
- L'indication de la version de protocole utilisée est très importante car elle permet la prise en compte d'intermédiaires sur le réseau qui ne prennent pas tous en charge les mêmes versions : la version 1.1 assure une compatibilité ascendante avec la version 1.0.

HTTP1.1 : Description requête

- Code statut réponse
 - 1xx (Informationnel) : non utilisé, réservé pour des applications futures
 - 2xx (Success) : requête correctement reçue, comprise, traitée
 - 200 : requête satisfaitة
 - 201 : une URI a été créée - cf. corps et directive Location pour plus d 'info
 - 202 : requête acceptée mais non traitée - cf. corps pour plus d'info
 - 204 : requête satisfaitة - le serveur n 'a rien de plus à dire - pas de corps - l 'écran présenté à l 'utilisateur ne devrait pas être modifié par le client
 - 3xx (Redirection) : Il faut une autre requête pour accéder à la ressource
 - 300 : la ressource demandée existe sous plusieurs formes - cf. corps et autres directives pour plus d 'info - exemple : on demande index.html et le serveur dispose de index.html.fr et de index.html.en
 - 301 : la ressource a changé d 'adresse - cf. directive Location
 - 302 : la ressource existe mais est temporairement inaccessible - cf. directive Location et corps pour une alternative présentée à l 'utilisateur (pas de redirection automatique)
 - 4xx (Client Error) : requête syntaxiquement incorrecte ou incomprise
 - 400 : requête syntaxiquement incorrecte
 - 401 : la ressource nécessite une authentification de l 'utilisateur. La réponse comporte forcément une directive WWW-Authenticate pour permettre une autre requête avec la directive Authorization.
 - 403 : le serveur refuse de délivrer la ressource et il ne s 'agit pas d 'un problème d 'authentification
 - 404 : ressource inexistante (faute de frappe dans l 'URL ou document supprimé)
 - 5xx (Server Error) : requête correcte mais non satisfaitة
 - 500 : le serveur a eu un problème (cf. procedure core dump ou syntax error)
 - 501 : le serveur est incapable d'appliquer la requête
 - 502 : le serveur est un proxy ou une passerelle, et a reçu une réponse erronée d 'une machine située en amont
 - 503 : le serveur n'est pas en mesure de satisfaire la requête à cause d'un problème temporaire (cf. surcharge ou maintenance). Ce code implique que le problème sera résolu dans un certain délai.

SMTP

- SMTP (*Simple Mail Transfer Protocol*) est un standard Internet (*Proposed Standard*) proposé par l'IETF sous la référence RFC2821 (dernière RFC de avril 1996).
- Cette spécification a pour objectif de définir un protocole application (couche 7) de transfert de courrier électronique (e-mail) en utilisant un canal de distribution tel que TCP (mais non limité à ce canal). Sa simplicité explique sans doute sa robustesse, il est par ailleurs très largement utilisé aujourd'hui pour la messagerie Internet, associé à POP ou IMAP4 : SMTP se charge du transfert des messages alors que POP (Post Office Protocol) ou IMAP (Internet Mail Access Protocol) permettent à l'utilisateur de gérer sa boîte aux lettres et de récupérer ses messages.
- Une fonction importante de SMTP est la possibilité de transférer un message en s'appuyant sur un réseau de serveurs relais et de garantir ainsi sa livraison à travers des environnements de transport différents : LAN, WAN, Internet, etc.

SMTP : Transmission d'un message

- Le transfert d'un message se passe de la manière suivante :
 - Un client SMTP (tel que MS Outlook Express ou Mozilla), appelé aussi émetteur ou MUA (Mail User Agent), envoie un message vers un serveur SMTP.
 - Si le serveur SMTP en question est le destinataire final du message, il stocke ce message dans la boîte aux lettres de l'utilisateur. Souvent, un serveur SMTP est aussi appelé MTA (Mail Transfert Agent).
 - Mais si le serveur SMTP n'est pas ce destinataire final, on parle d'un serveur relais, puisqu'il se charge de transmettre ce message au serveur de destination finale. Un tel serveur est appelé également passerelle (gateway) lorsque le transfert se fait entre deux environnements de transport différents.
- Il est important de comprendre qu'un même serveur SMTP peut-être amené à jouer le rôle de destinataire final (stockage des messages), d'émetteur et de relais (ou de passerelle).
- À noter enfin que chaque serveur est responsable de la transmission d'un message et, en cas d'erreur ou de problème, de la notification du problème à l'émetteur.

SMTP : Description du message

- SMTP transporte un objet message composé :
 - d'une enveloppe, elle-même composée d'une série de champs, dont l'adresse de l'émetteur, une ou plusieurs adresses de destinataires et des données d'extension ;
 - d'un contenu, composé d'une en-tête et d'un corps de message MIME.

Delivered-To : gerard.florin@cnam.fr

Date: Thu, 21 Mar 2002 15:15:39 +0100

From: Bruno Traverson <bruno.traverson@der.edf.fr>

Organization: EDF-DER

X-Accept-Language: fr

MIME-Version: 1.0

To: th-rntl-accord@rd.francetelecom.com

Cc: TRAVERSON Bruno <Bruno.Traverson@der.edfgdf.fr>

Subject: Un premier retour sur le modèle

Corps du courrier électronique

SMTP : commandes

- Le transfert des messages est réalisé lors d'un dialogue entre deux serveurs SMTP : respectivement
 - le client qui émet le message (qui peut être la source du message ou un relais) et
 - le serveur qui le réceptionne (qui peut être le destinataire du message ou un relais).
- Ce dialogue s'établit dans le cadre d'une session. Le client envoie ensuite une séquence de commandes qui attendent systématiquement une réponse du serveur pour notifier du succès ou de l'échec de la commande (par exemple : 250 OK).
- Une séquence type est la suivante¹ :
 - La commande EHLO (ou HELO) initie le dialogue, identifie le client et lui permet de connaître les extensions SMTP supportées par le serveur.
 - La commande MAIL FROM débute une transaction d'envoi de message en précisant l'adresse de l'émetteur.
 - La commande RCPT TO identifie l'(les)adresse(s) des destinataires.
 - La commande DATA envoie, ligne à ligne, le contenu du message. La fin de la transmission et donc de la transaction est indiquée par une ligne contenant uniquement un caractère « . ». Seuls les messages transmis dans le cadre d'une transaction valide et complète seront traités par le serveur.
 - La commande QUIT termine la session.

SMTP : commandes

- D'autres commandes sont définies par SMTP :
 - RESET, qui arrête la transaction en cours ;
 - VERIFY, qui vérifie l'argument comme étant une adresse de messagerie ;
 - EXPAND, qui vérifie que l'argument est une liste de messagerie et retourne le contenu de cette liste ;
 - HELP, qui demande une information ;
 - NOOP, qui n'a aucune action si ce n'est une réponse du serveur.
- Enfin, des extensions SMTP sont possibles : il s'agit de commandes commençant par un caractère « X » et qui dépendent des logiciels serveurs SMTP utilisés. La commande EHLO permet de connaître ces extensions.

SSL et TLS

- SSL (Secure Socket Layer) est un protocole qui a pour objectif d'assurer la sécurité des échanges entre un client et un serveur sur le Web (authentification et chiffrement).
- Il a été développé par la société Netscape qui a d'ailleurs déposé un brevet sur cette technologie en 1997 (n°5657390),
- La version 2.0 de SSL date de 1994 (http://wp.netscape.com/eng/security/SSL_2.html) : cette version est obsolète, mais elle est pourtant toujours prise en charge par les principaux navigateurs.
- En 1996, un groupe de travail est créé par l'IETF afin de standardiser un protocole de sécurité pour remplacer SSL. Cette date coïncide avec la publication de la version 3.0 de SSL (Internet Draft disponible à <http://wp.netscape.com/eng/ssl3>), qui est par ailleurs la version la plus récente de ce protocole.
- Enfin, le groupe de travail de l'IETF a publié en 1999, sous la référence RFC2246 (Proposed Standard), la version 1.0 de TLS (Transport Layer Protocol). Les différences entre SSL v3 et TLS v1 sont mineures et d'ailleurs ce dernier s'identifie lui-même comme la version 3.01 de SSL.
- Les principaux navigateurs Internet prennent en charge les trois protocoles : SSL v2, SSL v3 et TLS v1. Mais contrairement à Netscape Navigator et Mozilla, Internet Explorer n'active pas par défaut TLS, qu'il faut donc configurer manuellement.
- Lorsqu'un navigateur négocie une connexion sécurisée avec un serveur, il cherche d'abord à établir une session TLS, puis dans l'ordre une session SSL v3 et une session SSL v2, en fonction des possibilités du serveur.

SSL et TLS

- Les techniques mises en œuvre pour assurer la sécurité des échanges ont pour objectif :
 - pour l'émetteur, de crypter ses données et pour le récepteur, de les décrypter ;
 - de contrôler l'intégrité des informations reçues ;
 - d'authentifier l'émetteur du message ;
 - d'obliger l'émetteur à reconnaître l'émission des informations grâce à un mécanisme de non répudiation.

SSL et TLS

- Le chiffrement/déchiffrement de l'information est réalisé à l'aide d'un algorithme (ou *cipher* en anglais) : l'intérêt de cette technique est que la sûreté du chiffrement ne repose pas sur la méthode de calcul utilisée, qui est connue et publiée, mais sur l'utilisation de chiffres, appelés *clés*, qui permettent à l'algorithme de générer un document crypté, puis de le déchiffrer.
- Plus ces clés sont grandes (en nombre de bits), plus il est difficile, voire « impossible », de déchiffrer un document si l'on ne connaît pas les chiffres qui ont permis sa génération.
- Deux types de clés sont utilisés :
 - Les clés *symétriques* : comme leur nom l'indique, la même clé est utilisée pour chiffrer et déchiffrer les données. Cette technique est rapide et fournit en outre un moyen d'authentification. Elle est néanmoins un peu sensible, puisque toute la sécurité repose sur la connaissance d'une clé que l'émetteur et le récepteur doivent garder secrète.
 - Les clés *publiques* (ou asymétriques) : le principe consiste à chiffrer les données avec une clé publique, c'est-à-dire connue de tout le monde, et de les déchiffrer avec une clé privée connue uniquement par le récepteur. Les deux clés publique/privée fonctionnent par paire et dans les deux sens puisque, à l'inverse, la clé publique peut déchiffrer ce qui a été généré à l'aide de la clé privée.
 - Cette technique est plus lourde et n'offre pas de mécanisme d'authentification de l'émetteur. En revanche, elle permet d'authentifier le récepteur qui peut chiffrer sa signature à l'aide de sa clé privée.

SSL et TLS

- Le contrôle d'intégrité d'un message est réalisé à partir d'une *signature électronique*, elle-même cryptée, et qui est le résultat d'une fonction de hachage.
 - Une fonction de hachage appliquée à des données fournit un chiffre unique : la moindre altération de ces données modifie obligatoirement le résultat du hachage.
 - Le résultat d'une fonction de hachage ne permet pas de déterminer les données qui ont permis sa génération.
- Un *certificat* est un document électronique qui permet d'identifier une entité, c'est-à-dire un utilisateur, une entreprise, un serveur, etc. Il est associé à la clé publique de l'entité qu'il identifie. Il est également lié à une *autorité de certification* (CA ou Certification Authority) : il s'agit d'une application serveur qui peut être spécifique à une entreprise, ou gérée par une organisation tierce (telle que Verisign : <http://www.verisign.com>).
 - Le rôle de cette autorité est de générer le certificat et de proposer à l'utilisateur qui le reçoit une fonction de validation.
- Les certificats, associés aux signatures électroniques, sont utilisés pour authentifier à la fois le client et le serveur dans le cadre d'une connexion SSL, mais aussi des e-mails (S/MIME), du code Java ou JavaScript, etc.

SSL/TLS : Présentation générale

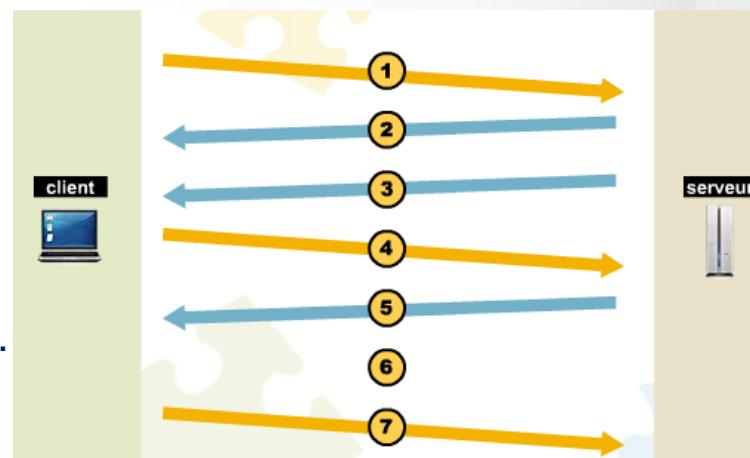
- SSL et TLS sont des protocoles qui se situent à un niveau intermédiaire, entre la couche transport et la couche application. Cette position permet donc a priori à toutes les applications qui s'appuient sur TCP/IP (HTTP, SMTP, Telnet, etc.) d'utiliser les fonctionnalités de SSL/TLS, soit :
 - permettre à un serveur de s'authentifier auprès d'un client, c'est-à-dire au client de vérifier l'identité du serveur ;
 - optionnellement, permettre au client de s'authentifier auprès du serveur, c'est-à-dire au serveur de vérifier l'identité du client ;
 - permettre au client et au serveur de sélectionner un algorithme de chiffrement ;
 - permettre aux deux machines d'établir une connexion cryptée pour assurer un haut niveau de confidentialité.
- Ces protocoles sont décomposés en deux sous-protocoles :
 - le protocole d'enregistrement (*record protocol*), qui définit le format de transmission des données ;
 - le protocole de négociation (*handshake protocol*), qui s'appuie sur le protocole d'enregistrement, et qui est chargé d'établir la connexion entre le client et le serveur (authentification, sélection de la méthode de chiffrement, etc.).

SSL/TLS : méthodes de chiffrement

- Tous les échanges de données réalisés dans le cadre d'une connexion SSL/TLS, y compris les messages initiaux gérés par le protocole de négociation, sont cryptés. Différents algorithmes mathématiques, classés en suites de chiffrement, sont pris en charge par SSL/TLS. Ces algorithmes sont très nombreux mais les principaux sont les suivants :
 - **DES** (Data Encryption Standard), un algorithme de chiffrement à base de clés symétriques créé par IBM en 1977
 - **Triple-DES**, l'algorithme DES appliqué trois fois ;
 - **AES** (Advanced Encryption Standard) est le nom du projet lancé en 1997 par le NIST pour trouver un remplaçant à DES. En octobre 2000, l'algorithme de chiffrement à base de clés symétriques *Rijndael*, créé par Joan Daemen et Vincent Rijmen, a été retenu et est devenu un standard fédéral américain (FIPS 197) ;
 - **IDEA** (International Data Encryption Algorithm), un algorithme de chiffrement à base de clés symétriques, créé par Xuejia Lai et James Massey et considéré comme très efficace (brevet international détenu par la société Ascom-Tech) ;
 - **MD5** (Message Digest), un algorithme d'empreinte numérique développé en 1991 par le professeur Ronald Rivest du MIT (RFC1321) ;
 - **RC2 et RC4** (Ron's Code), qui sont des algorithmes de chiffrement développés par le professeur Ronald Rivest du MIT pour la société RSA Security (brevet international) ;
 - **RSA**, qui est un algorithme à base de clé publique développé en 1977 par Rivest, Shamir et Adleman. Il est utilisé à la fois pour le chiffrement et l'authentification. (brevet US n° 4405829, tombé dans le domaine public depuis septembre 2000) ;
 - **SHA-1** (Secure Hash Algorithm), un algorithme d'empreinte numérique développé en 1994 et utilisé par le gouvernement américain ;

SSL TLS : Le protocole de négociation (handshake)

1. Le navigateur du client fait une demande de transaction sécurisée au serveur.
2. Suite à la requête du client, le serveur envoie son certificat au client.
3. Le serveur fournit la liste des algorithmes cryptographiques qui peuvent être utilisés pour la négociation entre le client et le serveur.
4. Le client choisit l'algorithme.
5. Le serveur envoie son certificat avec les clés cryptographiques correspondantes au client.
6. Le navigateur vérifie que le certificat délivré est valide.
7. Si la vérification est correcte alors le navigateur du client envoie au serveur une clé secrète chiffrée à l'aide de la clé publique du serveur qui sera donc le seul capable de déchiffrer puis d'utiliser cette clé secrète. Cette clé est un secret uniquement partagé entre le client et le serveur afin d'échanger des données en toute sécurité.



SSL/TLS

- Remarque
 - On remarquera que ce protocole, d'un point de vue cryptographique, n'authentifie pas le client. Seul le serveur l'est. L'authentification de l'utilisateur intervient généralement ensuite par un simple identifiant/mot de passe protégé par le chiffrement de la session SSL établie.
- Conseil
 - Afin d'éviter des attaques, il est recommandé d'utiliser la double authentification c'est-à-dire non seulement l'authentification du serveur mais également celle du client, qui, avec SSL, est facultative par défaut. On bénéficiera ainsi d'un accès distant authentifié par certificat de clé publique.
- Attention
 - Il est également recommandé d'interdire l'utilisation de protocoles obsolètes comme SSLv2.
- Remarque
 - Il existe d'autres méthodes de contrôle de l'accès distant, notamment celles qui emploient des protocoles de chiffrement comme IPsec. Toutefois, elles interviennent directement au niveau du transport,

Objectif du chapitre 2

- Fondation des services Web (les protocoles Internet)
 - URI, URL, URN
 - MIME
 - HTTP 1.1
 - SMTP
 - les protocoles SSL et TLS
- **Fondation des services Web (les technologies XML)**
 - XML1.0
 - XML namespaces
 - Xlink
 - XML Base
 - XPath
 - XML Schema
 - L'interface DOM
 - Les analyseurs syntaxiques XML
- Principe et objectifs des Services Web SOAP
 - SOAP par l'exemple HelloWorld Service
 - Structure message SOAP
 - SOAP et le transport http
 - Client SOAP UI
 - WSDL
- Exemple pratique de création et de déploiement d'un Webservice SOAP et de son client

Fondation des Services Web (Les technologies XML)

PROBLEMATIQUE

	1986 : SGML	1991 : HTML	1998 : XML
Objectifs	<ul style="list-style-type: none">■ adaptabilité,■ intelligence,■ gestion des liens	<ul style="list-style-type: none">■ simple,■ portable,■ gestion de liens	<ul style="list-style-type: none">■ puissance de SGML,■ simplicité du HTML
Inconvénients	<ul style="list-style-type: none">■ complexe,■ difficilement portable	<ul style="list-style-type: none">■ non adaptable,■ non intelligent	

DEFINITION 1/2

- eXtensible Markup Language
 - Recommandation (norme) du W3C
 - Spécifiant un langage
 - Constitué d'un ensemble d'éléments appelés balises
 - Utilisable pour créer d'autres langages
- 2 concepts fondamentaux
 - Structure, contenu et présentation sont séparés
 - Les balises ne sont pas figées

DEFINITION 2/2

➤ Conséquences :

- XML est un format de document
- XML est un format de données
- XML est un mode de structuration de l'information
- XML est un métalangage

Exemple

XML

```
<SONG>
<TITLE> Ma chanson </TITLE>
<COMPOSER> par l'auteur</COMPOSER>
<PRODUCER> Dupond</PRODUCER>
<EDITOR> Maison edition</EDITOR>
<DURATION> 6:20</DURATION>
<DATE> 1978</DATE>
<ARTIST> Toto</ARTIST>
</SONG>
```

HTML

```
<h1> Ma chanson (identification de la
définition d'un terme)
<ul>
<li> par l'auteur</li>
<li> Producteur : Dupond</li>
<li> Editeur : Maison edition</li>
<li> Duree : 6:20</li>
<li> Date : 1978</li>
<li> Artiste : Toto</li>
</ul>
```

- Balises “propriétaires” compréhensibles à la profession, dérivation des langages propriétaires.
- XML sépare le contenu de son aspect (à la différence de HTML où tout peut être mélangé)

Structure d'un document xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE exemple SYSTEM "exmple.dtd">

<DEVIS> ← Elément racine
    <!-- Entête -->
    <IDENTIFIANT>0401</IDENTIFIANT> ← Elément
    <LIBELLE>Achats Réseaux</LIBELLE>
    <STATUT>Demande d'achat</STATUT>
    <FOURNISSEUR>Alcatel Commutation</FOURNISSEUR>
    <MONTANT devise="FRF">1452805.30</MONTANT>
    <DATE_LIVRAISON>12/12/2000</DATE_LIVRAISON>
    <!-- Lignes article --> ← Commentaire
    <ARTICLE>
        <REFERENCE>ISML43S904</REFERENCE>
        <QUANTITE>280</QUANTITE> ← Attribut
        <PRIX_UNITAIRE devise="FRF">408.00</PRIX_UNITAIRE>
        <IMAGE href="media/ISML43S904.gif"/> ← Elément vide
    </ARTICLE>
    ...
</DEVIS>
```

Entête

Contenu

Specifications du langage xml

- Résumé des spécifications :
 - Un document doit commencer par une déclaration XML
 - Toutes les balises avec un contenu doivent être fermées
 - Toutes les balises sans contenu doivent se terminer par les caractères />
 - Le document doit contenir un et un seul élément racine
 - Les balises ne doivent pas se chevaucher
 - Les valeurs d'attributs doivent être entre guillemets
 - La casse doit être respectée pour toutes les occurrences de noms de balise (MAJUSCULES ou minuscules).
- Un document respectant ces critères est dit “**bien formé**”

Définitions de types de document 1/5

- DTD = Document Type Definition
- La DTD fournit :
 - la liste des éléments,
 - la liste des attributs,
 - des notations et
 - des entités du document XML associé ainsi que
 - les règles des relations qui les régissent.

Définitions de types de document 2/5

- La DTD est déclarée dans le document XML par la balise !DOCTYPE
- Elle peut être :
 - incluse dans le code source du fichier XML, ou DTD **interne** :
`<!DOCTYPE élément-racine [déclaration des éléments]>`
 - décrite dans un fichier externe, ou DTD **externe** :
`<!DOCTYPE élément-racine SYSTEM "nom_fichier.dtd">`

Définitions de types de document 3/5

Exemple de DTD interne

```
<?xml version="1.0" standalone="yes"?>  
  
<!DOCTYPE parent [  
  
    <!ELEMENT parent (garçon, fille)>  
  
    <!ELEMENT garçon (#PCDATA)>  
  
    <!ELEMENT fille (#PCDATA)>  
]  
  
<parent>  
    <garçon>François</garçon>  
    <fille>Elisabeth</fille>  
</parent>
```

Comme vous définissez une DTD interne, votre fichier est indépendant (standalone).

Début de la DTD interne avec parent comme élément de racine.

L'élément racine parent contiendra les sous-éléments garçon et fille.

#PCDATA indique au Parser XML que l'élément garçon contient des données exprimées en chiffres ou en lettres. Idem pour l'élément fille.

Fin de la DTD

Racine du document XML.

Fin du document XML.

Définitions de types de document 4/5

Exemple de DTD externe : reprise de l'exemple précédent, en version DTD externe

<pre><?xml version="1.0" standalone="no"?> <!DOCTYPE parent SYSTEM "parent.dtd"> <parent> <garçon>François</garçon> <fille>Elisabeth</fille> </parent></pre>	<p>Fichier XML. Comme vous définissez une DTD externe, votre fichier n'est plus indépendant (standalone).</p> <p>Déclaration de la DTD externe dans le Fichier parent.dtd.</p> <p>Racine du document XML.</p> <p>Fin du document XML.</p>
<pre><!ELEMENT parent (garçon,fille)> <!ELEMENT garçon (#PCDATA)> <!ELEMENT fille (#PCDATA)></pre>	<p>Fichier parent.dtd</p> <p>L'élément racine parent contiendra les sous-éléments garçon et fille.</p> <p>#PCDATA indique au Parser XML que l'élément garçon contient des données exprimées en chiffres ou en lettres.</p> <p>Idem pour l'élément fille.</p>

Définitions de types de document 5/5

- La DTD contient :
 - Une ou plusieurs définitions d'éléments introduites par la balise !ELEMENT :
`<!ELEMENT nom-élément valeur>`
 - Une ou plusieurs listes d'attributs introduites par la balise !ATTLIST :
`<!ATTLIST nom-élément attribut type défaut>`
 - Une ou plusieurs définitions d'entité introduites par la balise !ENTITY :
`<!ENTITY nom-entité "valeur">`
ou
`<!ENTITY nom-entité SYSTEM "nom_fichier">`

DTD : ELEMENTS 1/2

- Chaque balise d'un document XML valide doit être déclarée à l'aide d'un élément dans la DTD associée.
- Un élément est défini par la balise !ELEMENT :
`<!ELEMENT nom-élément valeur-élément>`
- Où :
 - nom-élément est une balise du document XML
 - valeur-élément prend l'une des trois formes:
 - (contenu)
Déclaration d'un élément à contenu explicite. Exemple :
`<!ELEMENT sujet (#PCDATA)>`
 - EMPTY
Déclaration d'un élément vide. Utilisé en HTML pour les sauts de ligne (BR), image, filet horizontal (HR) etc. Exemple :
`<!ELEMENT br EMPTY>`
 - ANY
Déclaration d'un élément pouvant contenir tout type de donnée. Exemple :
`<!ELEMENT note ANY>`

DTD : ELEMENTS 2/2

➤ Exemples

```
<!ELEMENT DEVIS (IDENTIFIANT, STATUT, MONTANT, ARTICLE+)>
<!ELEMENT IDENTIFIANT (#PCDATA)>                                ← Déclaration d'un élément dont le contenu est une suite d'éléments
<!ELEMENT STATUT (#PCDATA)>
<!ELEMENT MONTANT (#PCDATA )>
<!ATTLIST MONTANT devise CDATA #REQUIRED>                         ← Déclaration d'un attribut
<!ELEMENT ARTICLE (REFERENCE, QUANTITE, PRIX_UNITAIRE, IMAGE+)>
<!ELEMENT REFERENCE (#PCDATA)>                                     ← Déclaration d'un élément dont le contenu est une suite de caractères
<!ELEMENT QUANTITE (#PCDATA)>                                       ← Déclaration d'un élément dont le contenu est une suite de caractères
<!ELEMENT PRIX_UNITAIRE (#PCDATA)>
<!ATTLIST PRIX_UNITAIRE devise CDATA #REQUIRED>
<!ELEMENT IMAGE EMPTY>                                              ← Déclaration d'un élément vide
<!ATTLIST IMAGE href CDATA #REQUIRED>                                ← Déclaration de liste d'attributs
```

DTD : ATTRIBUTS 1/2

- Les balises ouvrantes et les balises vides peuvent contenir des couples nom-valeur des attributs
 - XML :

The diagram illustrates an XML element `<GREETING>`. A red oval encloses the attribute `LANGUAGE`, which is set to the value `“English”`. Two red arrows point from the label `nom` to the attribute name and from the label `valeur` to the attribute value.

```
<GREETING LANGUAGE = “English”>  
Hello everybody  
</GREETING>
```

- Les attributs possibles d'un élément du document XML sont déclarés dans la DTD associée.

DTD : ATTRIBUTS 2/2

- Exemples :

```
<!ELEMENT GREETING (#PCDATA)>
<!ATTLIST GREETING LANGUAGE CDATA "English">
```
- Définition d'un carré et de son attribut largeur :
 - DTD :

```
<!ELEMENT carre EMPTY>
<!ATTLIST carre largeur CDATA "0">
```
 - XML :

```
<carre largeur="100"/> ou <carre largeur="100"></carre>
```
- Liste de moyens de paiement:
 - DTD

```
<!ATTLIST paiement moyen (especes|cheque|CB) "CB">
```
 - XML

```
<paiement moyen="cheque">
<!-- ... -->
<paiement moyen="CB">
```

DTD : ENTITES 1/2

- Définition et rappel d'une entité XML
 - Une entité est une variable utilisée pour définir du texte.
 - L'intérêt d'une entité consiste à pouvoir remplacer autant de fois que nécessaire dans le document XML l'entité par le texte qui lui est associé.
 - Dans le document XML, la référence à une entité est introduite par le caractère "&" suivi du nom de l'entité, et terminée par le caractère ";". Il existe 5 entités prédéfinies en XML : lt, gt, amp, apos et quot (caractères < > & ' ")
- Une entité est définie avec la balise !ENTITY, et dite:
 - Interne si sa valeur est donnée dans la DTD
`<!ENTITY nom-entité "valeur">`
 - Externe si sa valeur est fournie dans un fichier externe à la DTD
`<!ENTITY nom-entité SYSTEM "nom_fichier">`

DTD : ENTITES 2/2

- Exemple de document XML avec DTD interne et entité externe "chapitre1" :

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

<!DOCTYPE livre [
    <!ENTITY chapitre1 SYSTEM "chapitre1.xml">
]>

<livre>
    <titre>Titre du livre</titre>
    &chapitre1;
</livre>
```

- Contenu du fichier "chapitre1.xml" :

```
<?xml version="1.0" ?>
<chapitre>
    <titre>titre du chapitre 1</titre>
    <section>1ère section</section>
    <section>2ème section</section>
</chapitre>
```

Définitions de types de document 1/2

- Résumé des spécifications :
 - Une DTD (grammaire) permet de déclarer :
 - un type d'élément,
 - une liste d'attribut d'un élément,
 - une entité
 - Chaque balise du langage doit faire l'objet d'une et d'une seule déclaration
- Un document XML est dit "valide" s'il possède une DTD et si sa syntaxe est conforme aux règles de la DTD
- Un document "valide" est obligatoirement "bien formé"

Définitions de types de document 2/2

- Espaces de nom
 - XML Namespaces est une recommandation permettant d'utiliser le vocabulaire (les balises) de 2 DTD distinctes sans risque d'ambigüité.
- Inconvénients des DTD :
 - Une DTD est difficile à lire
 - Une DTD est non extensible (ce n'est pas un document XML).
 - Une DTD ne permet pas de typer les données
 - Une DTD ne peut prendre en compte qu'un seul espace de nom (Namespace).

XML-SCHEMA 1/4

- En réponse aux lacunes des DTD, une alternative a été proposée comme recommandation : il s'agit de XML-Data dont XML-Schema est un sous-ensemble.
- Cette nouvelle norme achève de faire d'XML un format pivot...
- La version 1.1 de XML Schema (datée de mai 2001) se compose de 3 normes :
 - XML Schema tome 0 : Introduction
 - XML Schema tome 1 : Structures
 - XML Schema tome 2 : Types de données

XML-SCHEMA 2/4

- Les documents XML-Schema sont des documents :
 - respectant la syntaxe XML,
 - permettant de décrire la structure d'un document XML d'une façon beaucoup plus complète que les DTD.
- XML-Schema permet en effet de :
 - spécifier la typologie des données que va contenir le document XML décrit par le XML-Schema,
 - gérer une quarantaine de types de données simples,
 - gérer des types complexes,
 - gérer les occurrences des données.

XML-SCHEMA

3/4

➤ Exemple de document XML-Schema :

Le document XML :

```
<entree>
    <nom>Harry Cover</nom>
    <telephone>0102030405</telephone>
</entree>
```

Le document XML-Schema correspondant :

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:element name="entree"> ← Définition de la balise complexe entree
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="nom" type="xsd:string"
                minoccurs="1" maxoccurs="1"/>
            <xsd:element name="telephone" type="xsd:decimal"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Entete

← Définition de la balise String nom

← Définition de la balise de type Décimal telephone

XML-SCHEMA 4/4

- Référence à un XML-Shema dans un document XML :

```
<entree xmlns="http://www.annuaire.org"
        xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
        xsi:schemaLocation="http://www.annuaire.org/entree.xsd">
    <nom>Harry Cover</nom>
    <telephone>0102030405</telephone>
</entree>
```

L'espace de nommage xsi correspond aux instances de documents XML respectant les contraintes définies dans un document XML-Schema. Le W3C a défini une librairie de balises et attributs pouvant être utilisés par ces documents.

Constituants d'un XML-SCHEMA

1/8

➤ Déclaration de l'entête :

- L'élément `<xsd:schema>` permet de déclarer un document XML-Xschema.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
              targetNamespace="http://www.annuaire.org"
              xmlns="http://www.annuaire.org"
              elementFormDefault="qualified"/>
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!--balise schema obligatoire -->
```

L'attribut **targetNamespace** permet de préciser l'espace de nommage de ce type de documents.

L'attribut **elementFormDefault** précise si les documents XML respectant cette grammaire doivent référer à cet espace de nommage.

Constituants d'un XML-SCHEMA

2/8

- Déclaration des types de données :
 - Il est possible de déclarer un type de données
 - soit dans la déclaration d'un élément (local)
 - soit hors de la déclaration de l'élément (global)
 - XML-Schema permet d'utiliser des données :
 - de type prédéfini (string, int...)
 - de type complexe
 - dont le type est une restriction de type
 - dont le type est une extension de type

Constituants d'un XML-SCHEMA

3/8

➤ Déclaration des types de données :

• **Types prédéfinis :**

- byte, unsignedByte, hexBinary, integer, positiveInteger, negativeInteger, int, unsignedInt, long, unsignedLong, short, unsignedShort, decimal, float, double...
- string, NormalizedString, token
- boolean, anyURI, language
- time, dateTime, duration, date, gMonth, gYear, gYearMonth, gDay, gMonthDay
- ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATIN, NMTOKEN, NMTOKENS

Exemple : <xsd:element name="comment" type="xsd:string"/>

Constituants d'un XML-SCHEMA

4/8

➤ Déclaration des types de données :

- **Types complexes :**

Exemple : le type de données *TypeAdresse* se compose de 6 éléments *Numero*, *Rue1*, *Rue2*, *Ville*, *CP* et *Pays* :

```
<xsd:complexType name="TypeAdresse">
    <xsd:sequence>
        <xsd:element name="Numero" type="xsd:positiveInteger"/>
        <xsd:element name="Rue1" type="xsd:string"/>
        <xsd:element name="Rue2" type="xsd:string"/>
        <xsd:element name="Ville" type="xsd:string"/>
        <xsd:element name="CP" type="xsd:decimal"/>
        <xsd:element name="Pays" type="xsd:NMTOKEN" fixed="France"/>
    </xsd:sequence>
<xsd:element name="adresse" type="TypeAdresse"/>
```

Constituants d'un XML-SCHEMA

5/8

- Déclaration des types de données :
 - **Restriction de type existant :**

Exemple : le type de données string comprend 6 attributs optionnels : pattern, énumération, length, minlength, maxlength, whitespace. Si on désire représenter un choix Oui/Non (restriction sur l'attribut énumération) :

```
<xsd:simpleType name="choixOuiNon">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="oui"/>
    <xsd:enumeration value="non"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="choix" type="choixOuiNon"/>
```

Constituants d'un XML-SCHEMA

6/8

➤ Déclaration des types de données :

- **Extension / dérivation de type existant :**

Exemple : si l'on souhaite créer un type Personne contenant en plus du nom et du prénom, un élément de type Adresse (extension du type Adresse vu précédemment) :

```
<xsd:complexType name="Personne">
    <xsd:complexContent>
        <xsd:extension base="adresse">
            <xsd:sequence>
                <xsd:element name="Nom" type="xsd:string"/>
                <xsd:element name="Prenom" type="xsd:string"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Constituants d'un XML-SCHEMA

7/8

➤ Déclaration des éléments :

1. Définition d'un élément dont le type est déjà déclaré :

```
<xsd:complexType name="entree">
```

...

```
</xsd:complexType>
```

```
<xsd:element name="entree" type="entree">
```

2. Définition d'un élément contenant la définition du type :

```
<xsd:element name="entree">
```

```
  <xsd:complexType>
```

```
    <xsd:sequence>
```

```
      <xsd:element name="nom" type="xsd:string"/>
```

```
      <xsd:element name="telephone" type="xsd:decimal"/>
```

```
    </xsd:sequence>
```

```
  </xsd:complexType>
```

```
</xsd:element>
```

Constituants d'un XML-SCHEMA

8/8

➤ Déclaration des attributs :

Exemple de l'élément montant dans un document XML :

```
<montant devise="EURO" horsTaxe="true"/>.
```

L'attribut `devise` est optionnel et a comme valeur par défaut «EURO».

L'attribut `horsTaxe` est obligatoire et a comme valeur par défaut «true».

Modélisation en XML-Schema :

```
<xsd:element name="element">
  <xsd:complexType>
    <xsd:attribute name="devise" type="xsd:string"
      use="implied" value="EURO"/>
    <xsd:attribute name="horsTaxe" type="xsd:boolean"
      use="required" value="true"/>
  </xsd:complexType>
</xsd:element>
```

Conclusion : DTD versus XML-SCHEMA

- La DTD permet de définir facilement et rapidement des grammaires simples.
- XML-Schema permet de définir de manière plus formelle et complète une grammaire mais c'est au prix d'une complexité accrue.
- Un document XML-Schema respecte la syntaxe XML.
- Un document XML-Schema est généralement plus volumineux et plus difficile à lire qu'une DTD (pour un opérateur humain).

Exemple de XML-XSD

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<cave xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation='td2.xsd'>
<appellation id="01" nom="Saint Estèphe">
    <chateau id="0101">
        <nom> Chateau Bellevue</nom>
        <adresse> 12 Rue du Pont</adresse>
        <téléphone> 0556124321</téléphone>
    </chateau>
    <chateau id="0102">
        <nom> Chateau Le Bernadot</nom>
        <adresse> 21 Avenue du Cygne</adresse>
        <téléphone> 0556324231</téléphone>
    </chateau>
</appellation>
</cave>
```

CL



C

XSD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <xsd:element name="cave">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="appellation" minOccurs='1' maxOccurs='unbounded' />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="appellation">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="chateau" minOccurs='0' maxOccurs='unbounded' />
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:integer" use='required' />
      <xsd:attribute name="nom" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="chateau">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="nom" minOccurs='1' maxOccurs='1' />
        <xsd:element ref="adresse"/>
        <xsd:element ref="téléphone"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:integer" use='required' />
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="nom" type="xsd:string"/>
  <xsd:element name="adresse" type="xsd:string"/>
  <xsd:element name="téléphone" type="xsd:decimal"/>
</xsd:schema>
```

Deux rôles différents

- **Interopérabilité :**
Service Web, échange de fichier par l'intermédiaire de schéma
- Représentation et publication de document

Norme d'Interopérabilité

- JAX-Remote Procedure Call
- JAX-Registry
- JAX-Message
- SAAJ (SOAP)
- Java WSDP Registry

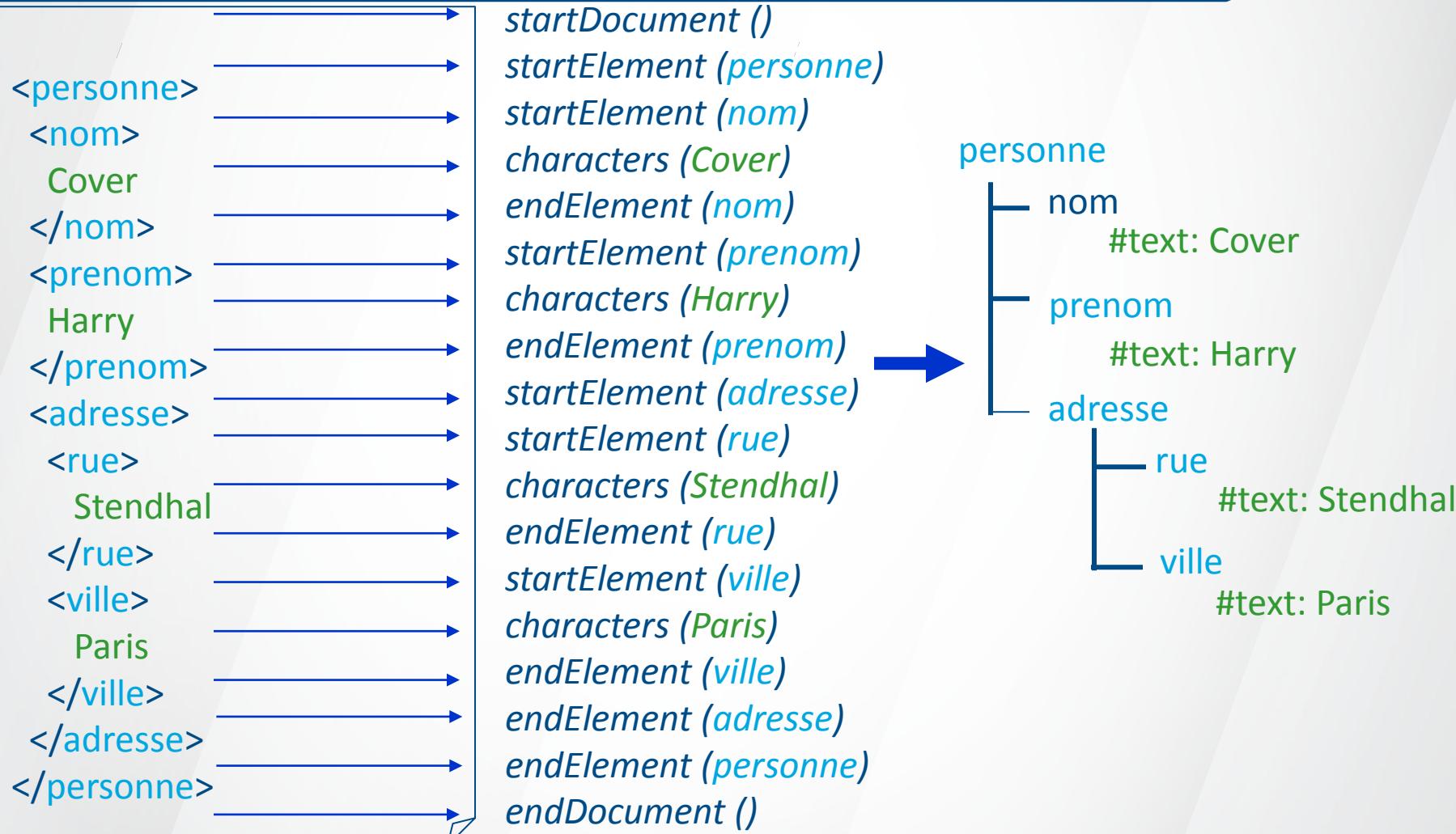
Norme Traitement de document

- JAX-Processing
java.xml.parser, javax.xml.transform
- JAX-Binding
représentation d'objet Java en XML
(Serialization, Deserialization)

Analyseur syntaxique (parser)

- Parseurs XML, transforme le texte du XML en éléments et attributs
 - 2 normes :
 - Simple Api for XML (SAX)
 - Document Object Model (DOM)
- Transformateur XML, effectue des transformations d'arbre
 - pas de norme

Document XML – Flux SAX – Arbre DOM



SAX (*Sample API for XML*)

- Modèle simplifié d'événement.
- Types d'événement :
 - début et fin de document ;
 - début et fin d'éléments ;
 - attributs, chaîne de caractères.
- Utilisé dans les implémentations des parseurs XML du domaine public.

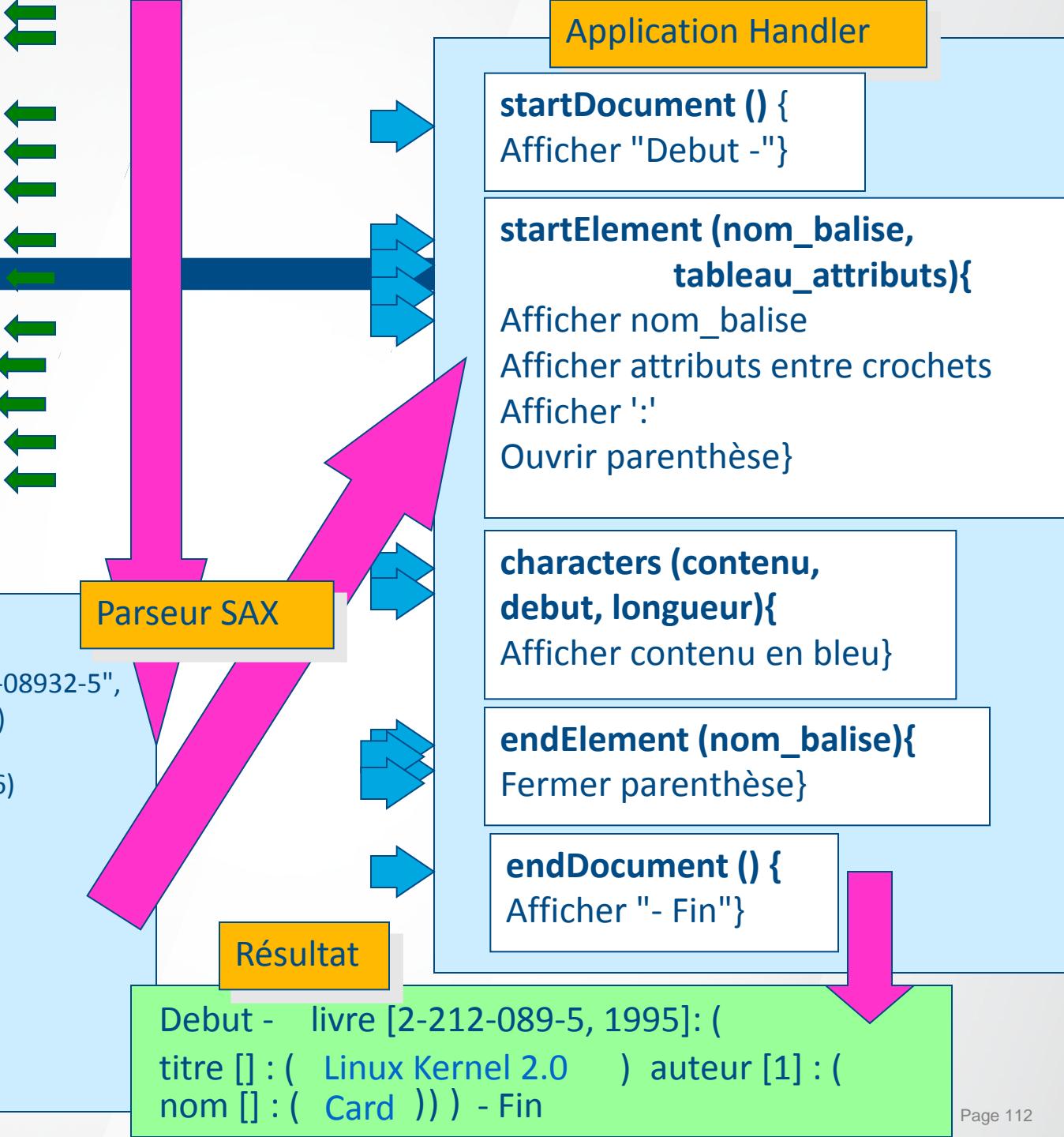
SAX

- Fonctionne sur le principe des callbacks (rétro-appels)
- Au fur et à mesure de la lecture document XML, le parseur appelle des fonctions correspondantes aux évènements rencontrés avec les paramètres appropriés
- Ces fonctions sont définies par une interface standards et doivent être implémentées de telle sorte à répondre aux besoins de l'application

```
<livre isbn="2-212-08932-5"  
date="1995">  
<titre>  
    Linux Kernel 2.0  
</titre>  
<auteur id="1">  
    <nom>  
        Card  
    </nom>  
</auteur>  
</livre>
```

XML

```
startDocument ()  
  
startElement ("livre", [isbn="2-212-08932-5",  
                      date="1995"] )  
startElement ("titre", [] )  
characters ("Linux Kernel 2.0", 0, 16)  
endElement ("titre")  
startElement ("auteur", [id="1"] )  
startElement ("nom", [] )  
characters ("Card", 0, 4)  
endElement ("nom")  
endElement ("auteur")  
endElement ("livre")  
endDocument ()
```

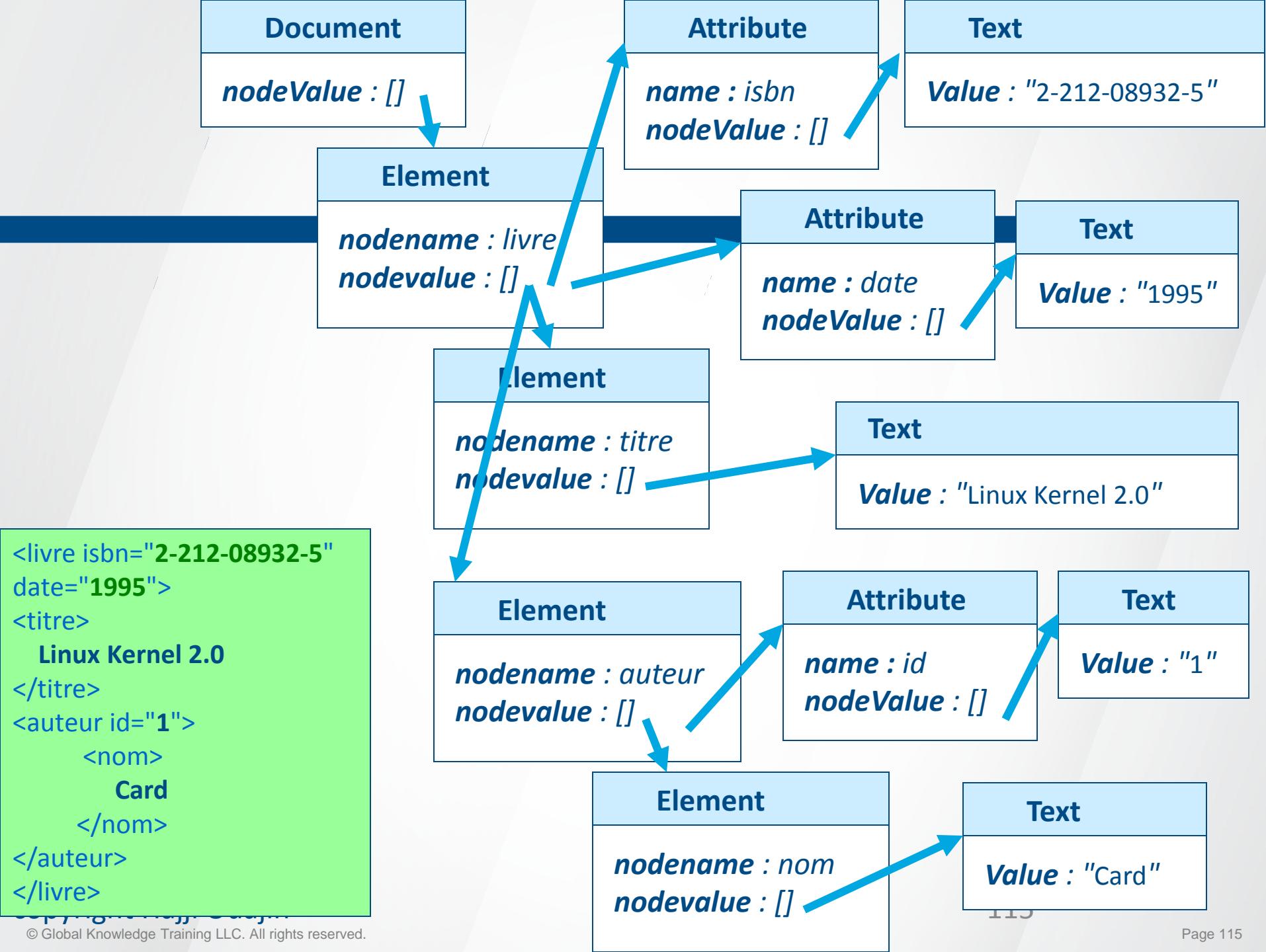


DOM (*Document Object Model*)

- API d'accès aux documents XML
- Interfaces d'accès en IDL
- Modèle de traitement d'arbres pour l'accès et la mise à jour
- API de manipulation d'arbres, d'objets typés avec des attributs
 - parcours
 - ajout
 - suppression

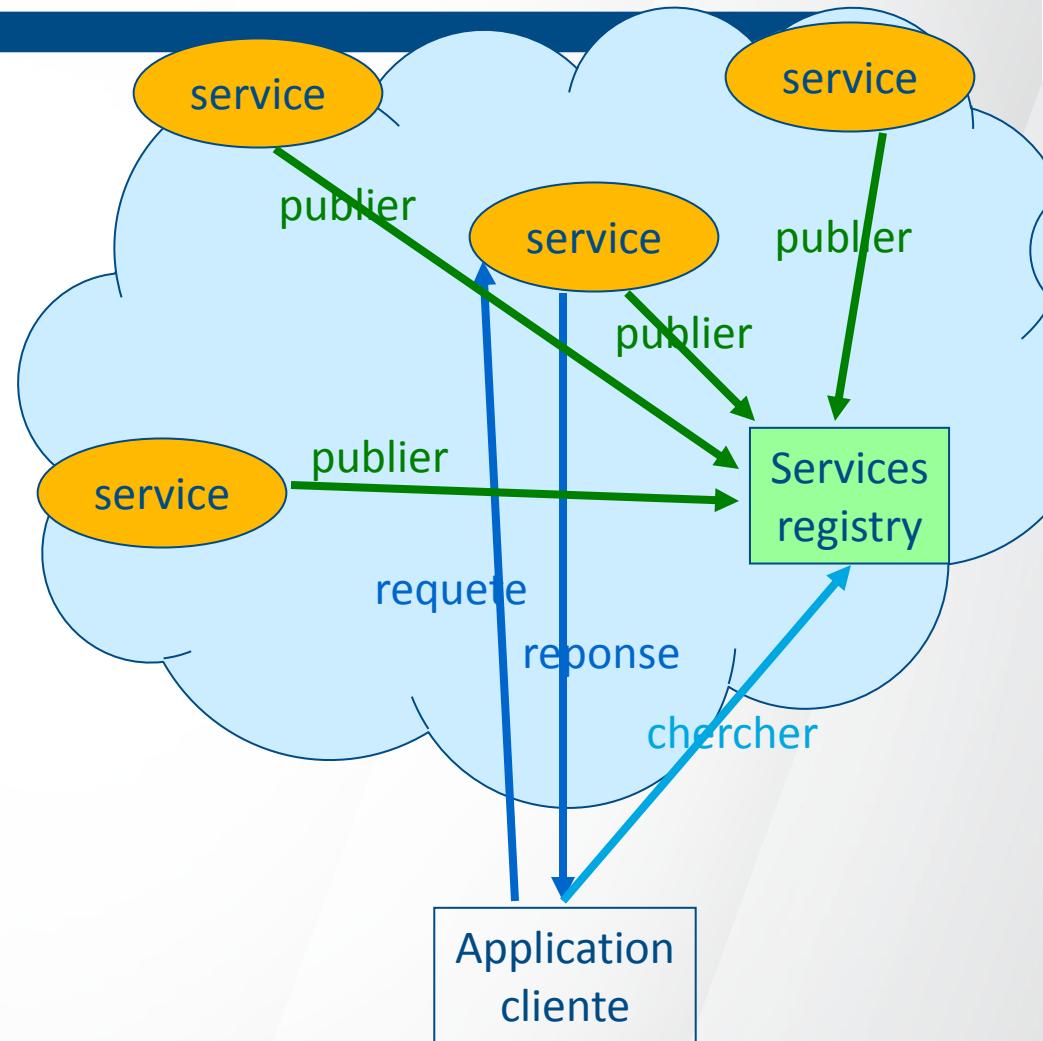
Modèle DOM

- Dans le modèle DOM, toutes les classes dérivent du type **Node**.
- La classe **Document** représente le document XML.
- La classe **Element** représente les éléments du document XML.
- **Document** ne peut posséder qu'un seul fils de type **Element** (XML n'a qu'une seule racine du document)
- **Attribute** représente les attributs d'un élément
- La classe **Text** représente le contenu textuel d'un **Element** ou d'un **Attribute**



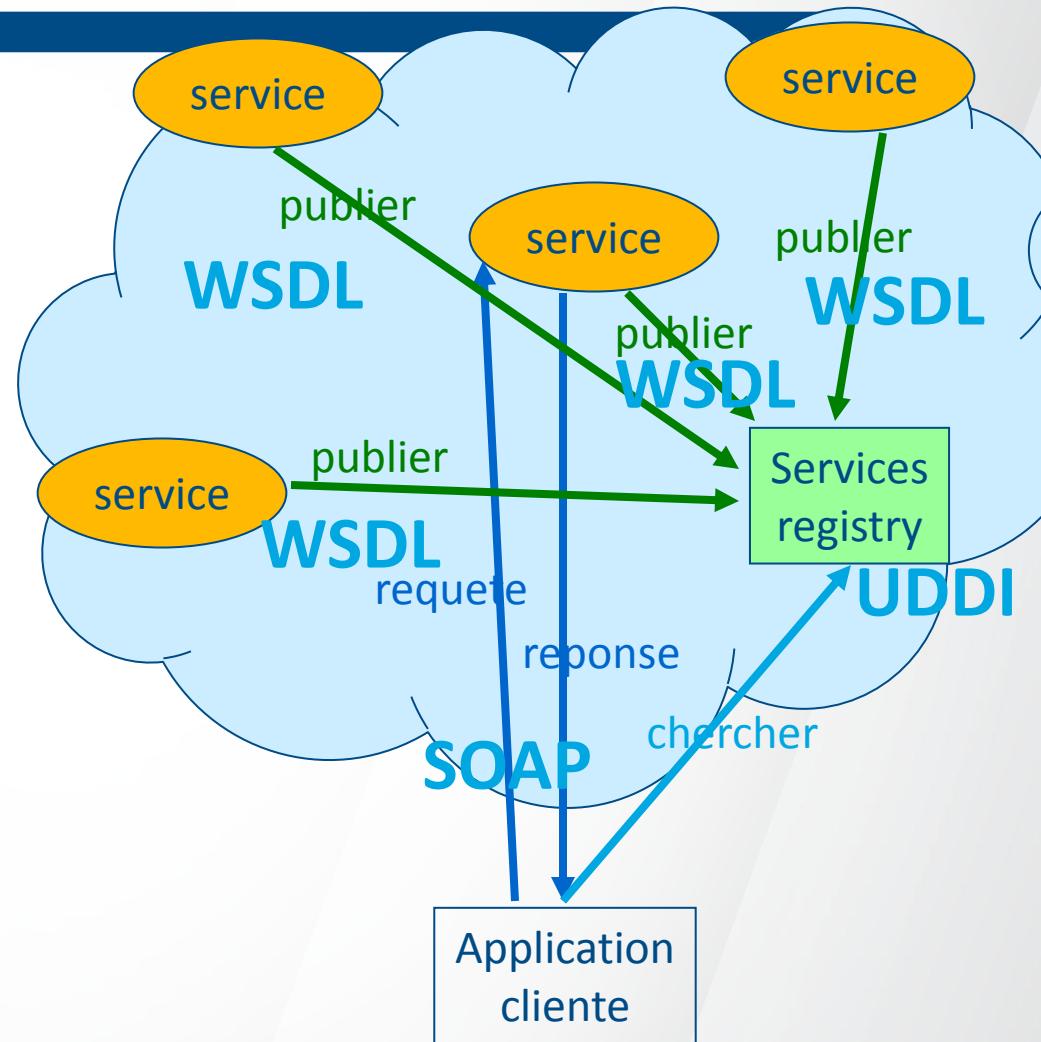
Invocation de services

- Invocations d'objets
 - RPC (1980)
 - CORBA
 - Java RMI
 - Microsoft DCOM
- Format spécifique
- Protocole de transport spécifique



Invocation de services

- Format XML ?
 - Système de typage (XML-Schema)
 - Format de codage universel
 - Sécurisation
 - Authentification, Gestion de clef
 - Chiffrement, Signature
 - Contrôle de transaction
- Protocole
 - HTTP : protocole web
 - SMTP : protocole de mail



Objectif du chapitre 2

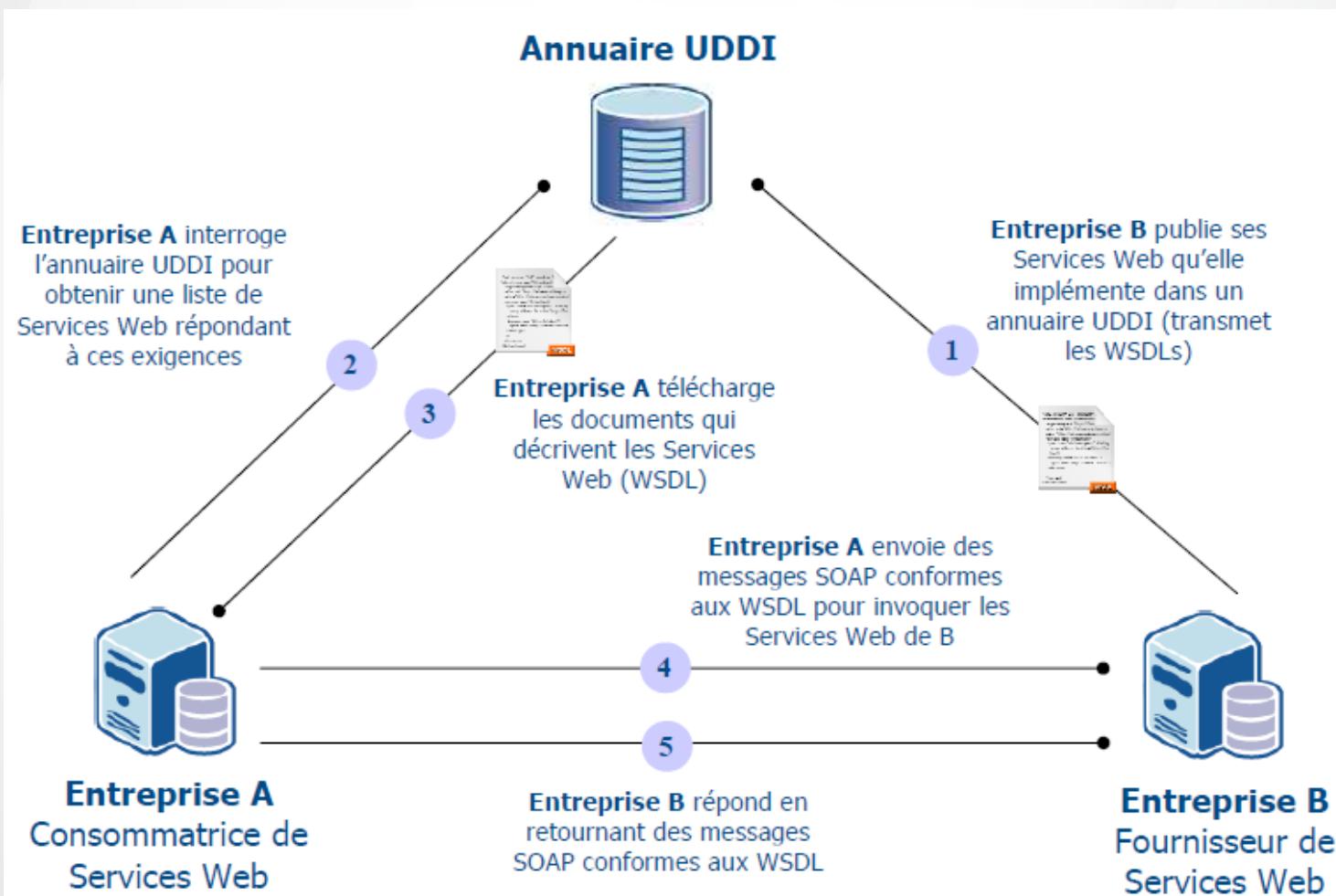
- Fondation des services Web (les protocoles Internet)
 - URI, URL, URN
 - MIME
 - HTTP 1.1
 - SMTP
 - les protocoles SSL et TLS
- Fondation des services Web (les technologies XML)
 - XML1.0
 - XML namespaces
 - Xlink
 - XML Base
 - XPath
 - XML Schema
 - L'interface DOM
 - Les analyseurs syntaxiques XML
- **Principe et objectifs des Services Web SOAP**
 - SOAP par l'exemple HelloWorld Service
 - Structure message SOAP
 - SOAP et le transport http
 - Client SOAP UI
 - WSDL
- Exemple pratique de création et de déploiement d'un Webservice SOAP et de son client

Principe et objectifs des Services Web SOAP

SOAP : Généralités

- SOAP est un protocole de communication entre application basé sur le langage XML
- Initialement SOAP désignait l'acronyme de Simple Object Access Protocol
- Qui est derrière SOAP (Microsoft et IBM)
- Objectifs visés
 - Assurer la communication entre applications d'une même entreprise (intranet)
 - Assurer les échanges interentreprises entre applications et services Web
- Spécification du W3C
 - SOAP 1.1 : <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
 - SOAP 1.2 : <http://www.w3.org/TR/soap12/>
- Pour comparaison, SOAP est similaire aux protocoles « RPC »

SOAP : Généralités



SOAP : Généralités

- Pour lire et écrire du SOAP, les prérequis sont
 - XML
 - XML Schéma
- Le pour
 - Utile pour débugger une application
 - Utile pour réaliser des tests via SOAP UI
 - Intercepter les messages bas niveau SOAP (via les handlers)
- Le contre
 - Les APIs fournissent une abstraction des messages SOAP

Concepts d'un message SOAP

- Les messages SOAP sont utilisés pour envoyer (requête) et recevoir (réponse) des informations d'un récepteur
- Un message SOAP peut être transmis à plusieurs récepteurs intermédiaires avant d'être reçu par le récepteur final (chaîne de responsabilité)
- Le format SOAP peut contenir des messages spécifiques correspondant à des erreurs identifiées par le récepteur
- Un message SOAP est véhiculé vers le récepteur en utilisant un protocole de transport (HTTP, SMTP, ...)

SOAP par l'exemple : Requête vers le service HelloWorld

- Exemple : Appeler les opérations du service HelloWorld

```
<soapenv:Envelope  
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:hel="http://helloworldwebservice.lisi.ensma.fr/">\n        <soapenv:Header/>\n        <soapenv:Body>\n            <hel:makeHelloWorld>\n                <value>Mickael BARON</value>\n            </hel:makeHelloWorld>\n        </soapenv:Body>\n    </soapenv:Envelope>
```

Message SOAP pour appeler l'opération *makeHelloWorld* contenant un paramètre *value*

Message SOAP pour appeler l'opération *simpleHelloWorld* ne contenant pas de paramètre

```
<soapenv:Envelope  
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:hel="http://helloworldwebservice.lisi.ensma.fr/">\n        <soapenv:Header/>\n        <soapenv:Body>\n            <hel:simpleHelloWorld/>\n        </soapenv:Body>\n    </soapenv:Envelope>
```

SOAP par l'exemple : Réponse du service HelloWorld

- Exemple (suite) : Message retour de l'appel des opérations du service HelloWorld

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <ns2:makeHelloWorldResponse xmlns:ns2="http://helloworldwebservice.lisi.ensma.fr/">
            <helloWorldResult>Hello World to Mickael BARON</helloWorldResult>
        </ns2:makeHelloWorldResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

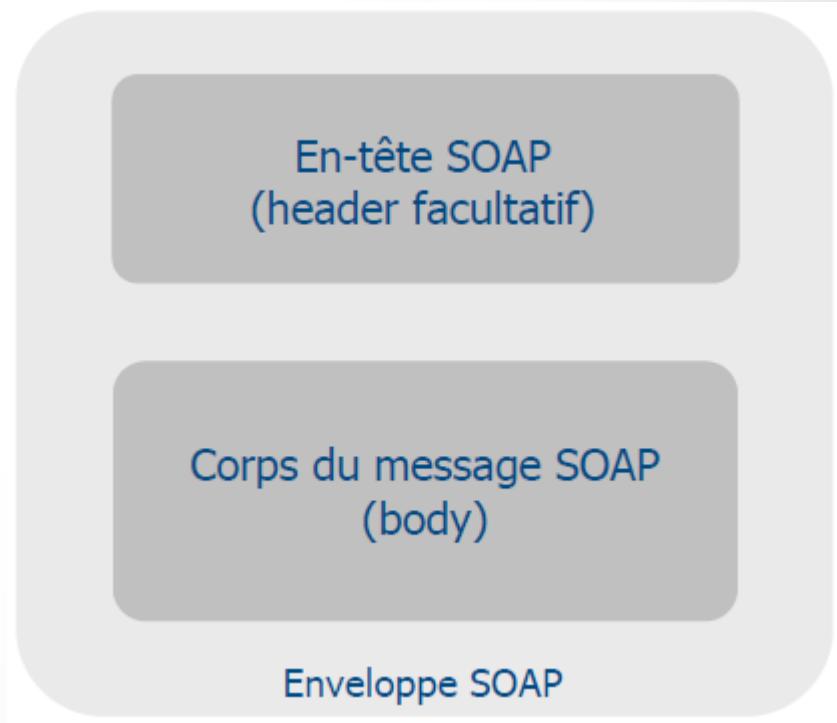
Les réponses sont
sensiblement identiques

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <ns2:simpleHelloWorldResponse xmlns:ns2="http://helloworldwebservice.lisi.ensma.fr/">
            <helloWorldResult>Hello World to everybody</helloWorldResult>
        </ns2:simpleHelloWorldResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Structure d'un message SOAP

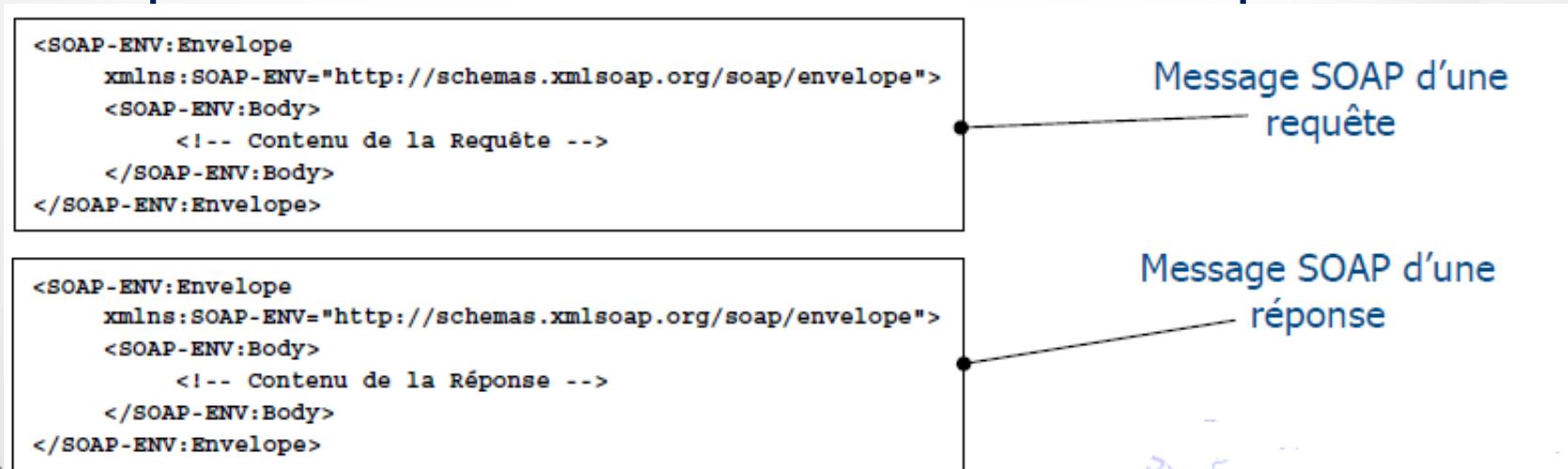
- Un message SOAP est un document XML constitué d'une enveloppe composée de deux parties

- Un en-tête (header)
qui peut être facultatif
- Un corps (body)



L'enveloppe SOAP

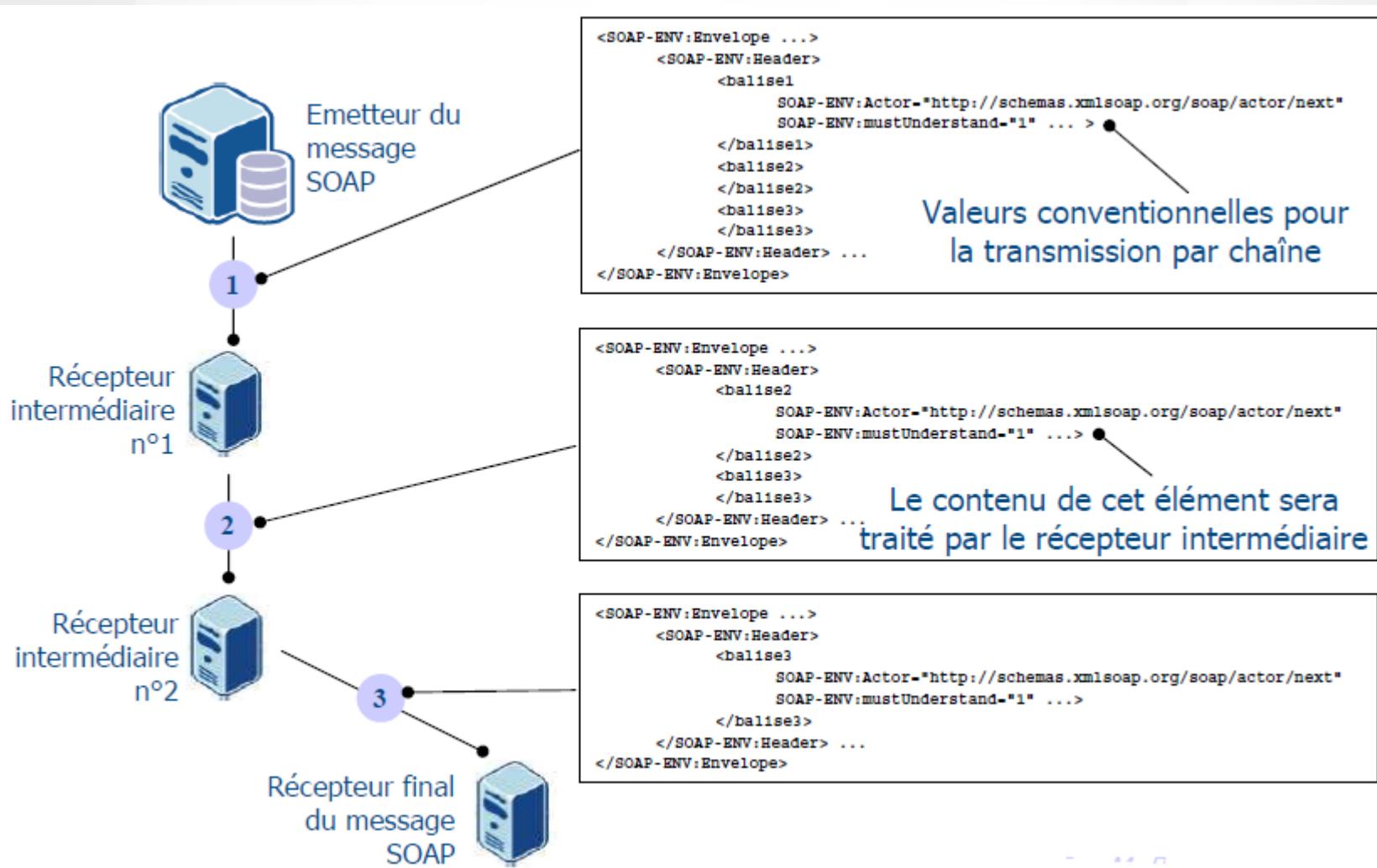
- L'enveloppe est la racine d'un message SOAP identifiée par la balise `<soapenv:Envelope>`
- La spécification impose que la balise et les sous balises soient explicitement associées à un namespace
- La spécification SOAP définit deux namespaces



En-tête SOAP

- L'en-tête d'un message SOAP est utilisé pour transmettre des informations supplémentaires sur ce même message
- L'en-tête est défini par la balise <SOAP-ENV:Header>
 - L'élément peut être facultatif
 - Doit être placé avant le corps
- Différents usages de l'en-tête ?
 - Informations authentifiant l'émetteur
 - Contexte d'une transaction
 - Pour certains protocole de transport (FTP par exemple), l'en-tête peut être utilisé pour identifier l'émetteur du message
- Un message SOAP peut transiter par plusieurs intermédiaires avant le traitement par le récepteur final
 - Pattern « Chaîne de responsabilité »
 - Zone lecture / écrite par les intermédiaires

En-tête SOAP



Corps SOAP

- Le corps d'un message SOAP est constitué par un élément <SOAP-ENV:Body>
- L'élément <SOAP-ENV:Body> peut contenir soit
 - Une erreur en réponse à une requête (élément <SOAP-ENV:Fault>)
 - Des informations adressées au destinataire du message SOAP respectant un encodage déterminé
- L'encodage des informations est précisé par les bindings du document WSDL
 - Attribut style (Document et RPC)
 - Attribut use (encoded et litteral)
- Pour faire simple nous utiliserons les services Web dans le cadre de l'appel à une procédure distante

Corps SOAP

- L'objectif visé par SOAP a été de fournir un mécanisme standardisé pour l'appel de procédures distantes (RPC)
- De ce fait les informations adressées au destinataire de messages SOAP doivent respecter un certain nombre de conventions
- Appel d'une opération représentée par une struct
 - Le nom de la structure est celui de l'opération à appeler
 - Chaque paramètre de l'opération est défini comme un sous élément de la structure
 - Si un paramètre est un type complexe (Person par exemple) une nouvelle structure est définie contenant à son tour des sous éléments ...
- Le résultat est également représenté par une struct
 - Le nom de la structure est celui de l'opération suivie de **Response**
 - Les paramètres sont également structurés

Corps SOAP

```
<soapenv:Envelope  
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:not="http://notebookwebservice.lisi.ensma.fr/">  
    <soapenv:Header/>  
    <soapenv:Body>  
        <not:addPersonWithComplexType>  
            <newPerson>  
                <address>Poitiers</address>  
                <birthyear>17081976</birthyear>  
                <name>BARON Mickael</name>  
            </newPerson>  
        </not:addPersonWithComplexType>  
    </soapenv:Body>  
</soapenv:Envelope>
```

Message SOAP pour appeler
l'opération
addPersonWithComplexType

Paramètre de type complexe
défini dans une structure
(*newPerson*)

```
<soapenv:Envelope  
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"  
    xmlns:not="http://notebookwebservice.lisi.ensma.fr/">  
    <soapenv:Header/>  
    <soapenv:Body>  
        <not:addPersonWithSimpleType>  
            <name>BARON Mickael</name>  
            <address>Poitiers</address>  
            <birthyear>17081976</birthyear>  
        </not:addPersonWithSimpleType>  
    </soapenv:Body>  
</soapenv:Envelope>
```

Message SOAP pour appeler
l'opération
addPersonWithSimpleType

Trois paramètres

Corps SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <ns2:addPersonWithComplexTypeResponse
            xmlns:ns2="http://notebookwebservice.lisi.ensma.fr/">
            <addPersonWithComplexTypeResult>true</addPersonWithComplexTypeResult>
        </ns2:addPersonWithComplexTypeResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

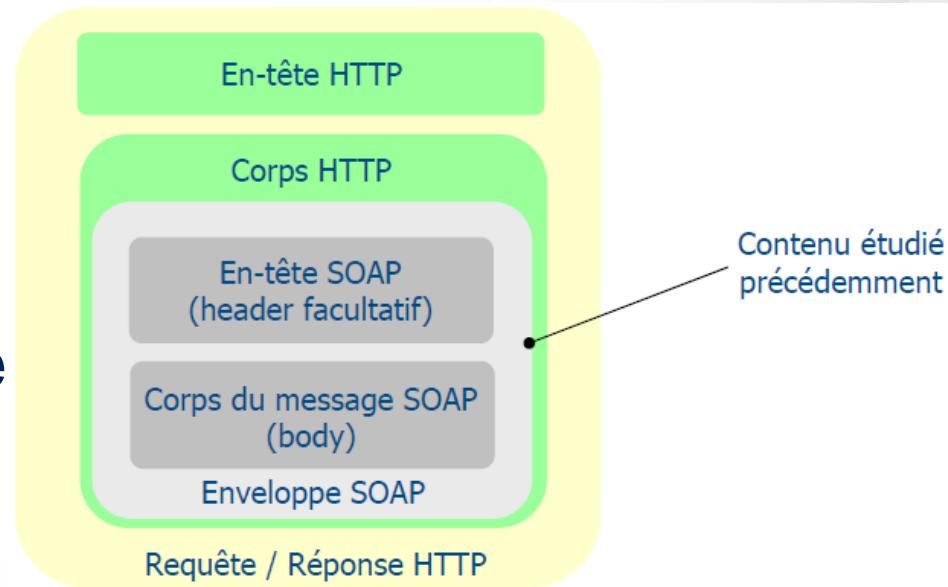
Messages SOAP pour la réponse
puisque les noms des opérations
sont suivis de *Response*

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Body>
        <ns2:addPersonWithSimpleTypeResponse
            xmlns:ns2="http://notebookwebservice.lisi.ensma.fr/">
            <addPersonWithSimpleTypeResult>true</addPersonWithSimpleTypeResult>
        </ns2:addPersonWithSimpleTypeResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Les paramètres de sorties suivent la
même convention que les
paramètres d'entrées

SOAP transporté par HTTP

- SOAP utilise un protocole de transport pour véhiculer les messages SOAP de l'émetteur au récepteur
 - HTTP, SMTP, FTP, POP3 et NNTP
- Le modèle requête/réponse de SOAP convient parfaitement au modèle requête/réponse HTTP



SOAP transporté par HTTP

➤ Requête SOAP HTTP

- Méthode de type POST
- Nécessite un attribut SOAPAction

```
POST http://localhost:8080/NotebookWebService/notebook HTTP/1.1
Content-Type: text/xml; charset=UTF-8
SOAPAction: ""
User-Agent: Jakarta Commons-HttpClient/3.1
Host: localhost:8080
Content-Length: 459
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:not="http://notebookwebservice.lisi.ensma.fr/"
    <soapenv:Header/>
    <soapenv:Body>
        <not:addPersonWithComplexType>
            <newPerson>
                <address>Poitiers</address>
                <birthyear>17081976</birthyear>
                <name>BARON Mickael</name>
            </newPerson>
        </not:addPersonWithComplexType>
    </soapenv:Body>
</soapenv:Envelope>
```

➤ Réponse SOAP HTTP

- Exploite les codes retours HTTP
- Si code de type 2xx, message SOAP reçu
- Si code 500, message en erreur, le corps SOAP doit contenir fault

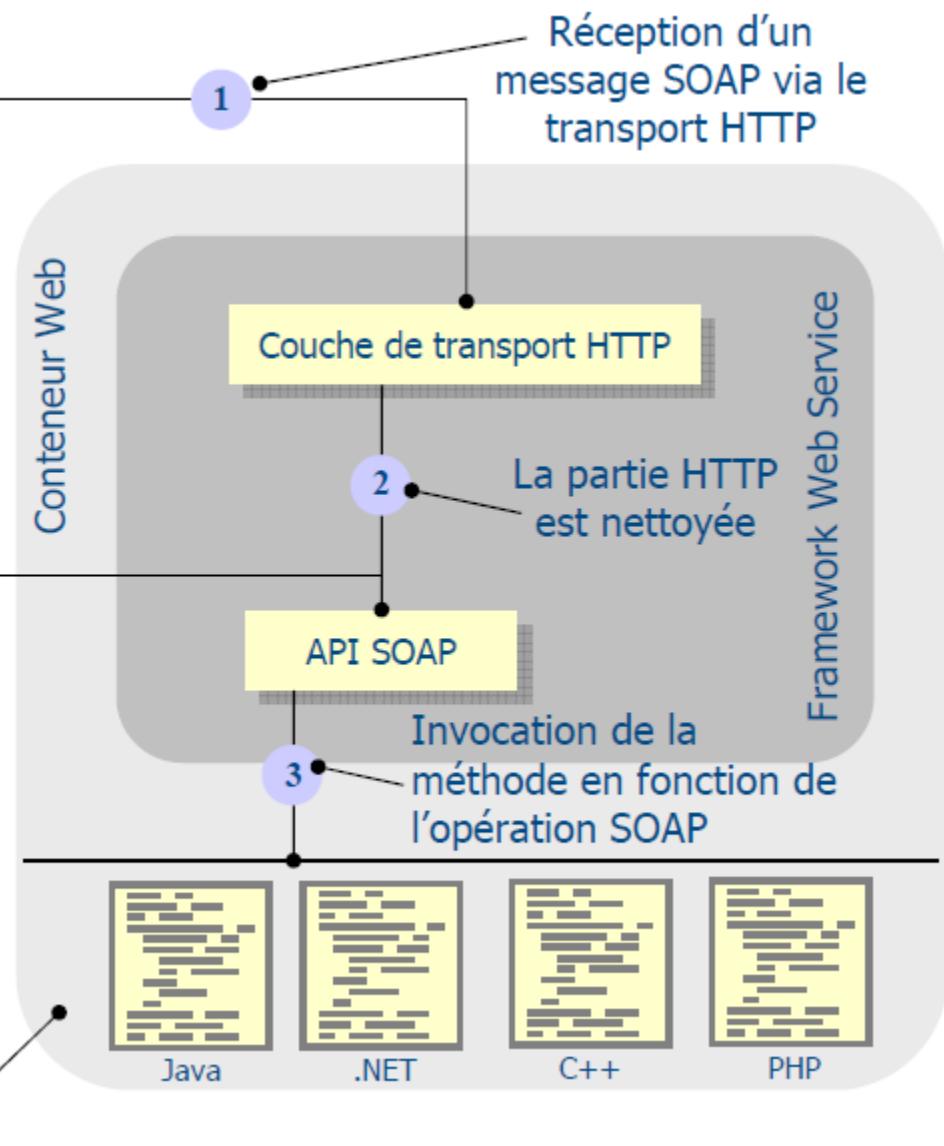
```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml; charset=utf-8
Transfer-Encoding: chunked
Date: Sun, 13 Dec 2009 12:00:33 GMT

<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    <soapenv:Body>
        <ns2:addPersonWithComplexTypeResponse
            xmlns:ns2="http://notebookwebservice.lisi.ensma.fr/">
            <addPersonWithComplexTypeResult>
                true
            </addPersonWithComplexTypeResult>
        </ns2:addPersonWithComplexTypeResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Traitement des messages SOAP

```
POST http://localhost:8080/NotebookWebService/notebook HTTP/1.1
Content-Type: text/xml; charset=UTF-8
SOAPAction: ""
User-Agent: Jakarta Commons-HttpClient/3.1
Host: localhost:8080
Content-Length: 459
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:not="http://notebookwebservice.lisi.ensma.fr/"
    <soapenv:Header/>
    <soapenv:Body>
        <not:addPersonWithComplexType>
            <newPerson>
                <address>Poitiers</address>
                <birthday>17081976</birthday>
                <name>BARON Mickael</name>
            </newPerson>
        </not:addPersonWithComplexType>
    </soapenv:Body>
</soapenv:Envelope>
```

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:not="http://notebookwebservice.lisi.ensma.fr/"
    <soapenv:Header/>
    <soapenv:Body>
        <not:addPersonWithComplexType>
            <newPerson>
                <address>Poitiers</address>
                <birthday>17081976</birthday>
                <name>BARON Mickael</name>
            </newPerson>
        </not:addPersonWithComplexType>
    </soapenv:Body>
</soapenv:Envelope>
```



SOAP UI : outil graphique de tests de Service Web

- SOAP UI est un outil pour tester des Services Web
 - www.soapui.org
 - Disponible en standalone ou intégré dans les environnements de développement (Eclipse, IntelliJ, Netbeans, Maven, ...)
 - Peut s'utiliser pour n'importe quelle plateforme de développement
- Fonctionnalités de SOAP UI
 - Supporte les Services Web étendus (WSDL + SOAP + UDDI) ou REST
 - Inspecter des Services Web
 - Invoquer des Services Web
 - Développer des Services Web
 - Simuler des Services Web via des bouchons (mocks)
 - Effectuer des tests qualités (temps de réponse, ...)



Practice: Test de SOAPUI dans Eclipse

SOAP UI : Practice

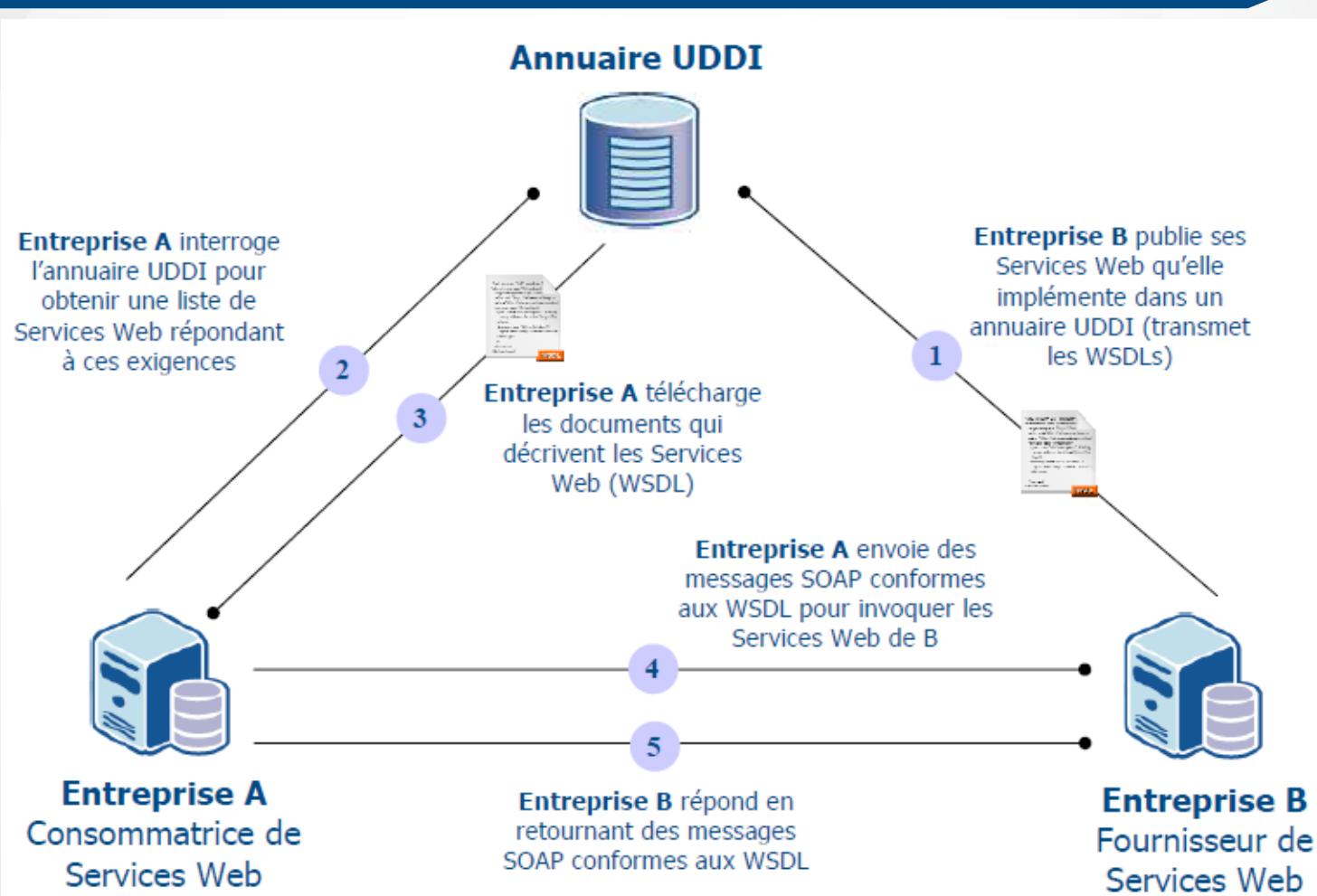
- Install SOAP UI
- Cr ation Projet SOAP
- Ajout d'un WSDL
- <http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL>
- Commencer   tester

https://graphical.weather.gov/xml/SOAP_server/ndfdXMLServer.php?wsdl

WSDL : Généralités

- WSDL est l'acronyme de **Web Service Description Language**
 - Basé sur le langage XML et permet de décrire un service Web
 - Fournit une description indépendante du langage et de la plate-forme
- Par comparaison WSDL est assez semblable au langage IDL défini par CORBA
- Spécification du W3C
 - WSDL 1.1 : <http://www.w3.org/TR/wsdl>
 - WSDL 2.0 : <http://www.w3.org/TR/wsdl20/>
- A partir d'un document WSDL il est possible
 - Générer un client pour appeler un Service Web
 - Générer le code pour implémenter un Service Web
- Où trouver des documents WSDL
- Amazon Associates Web Service
 - <http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl>
- ebaY
 - <http://developer.ebay.com/webservices/finding/latest/FindingService.wsdl>
- National Oceanic and Atmospheric Administration
 - <http://www.nws.noaa.gov/xml/>
 - <http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl>

WSDL : Généralités



Concepts d'un document WSDL

- **Types** : fournit la définition de types de données utilisés pour décrire les messages échangés.
- **Messages** : représente une définition abstraite (noms et types) des données en cours de transmission.
- **PortTypes** : décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou d'erreurs.
- **Binding** : spécifie une liaison entre un `<portType>` et un protocole concret (SOAP, HTTP...).
- **Service** : indique les adresses de port de chaque liaison.
- **Port** : représente un point d'accès de services défini par une adresse réseau et une liaison.
- **Opération** : c'est la description d'une action exposée dans le port.

Organisation d'un document WSDL

- Un document WSDL est décomposé en deux parties
- **Partie abstraite** qui décrit les messages et les opérations disponibles
 - Types (<types>)
 - Messages (<message>)
 - Types de port (<portType>)
- **Partie concrète** qui décrit le protocole à utiliser et le type d'encodage à utiliser pour les messages
 - Bindings (<binding>)
 - Services (<service>)
- Plusieurs parties concrètes peuvent être proposées pour la partie abstraite
- Motivation de cette séparation ? Réutilisabilité de la partie abstraite

WSDL par l'exemple : Carnet d'adresse

- Le service Notebook fournit trois opérations
 - Une opération addPerson qui prend en paramètre un objet Person et retourne un booléen pour indiquer l'état de création
 - Une opération addPerson qui prend en paramètre trois chaînes de caractères (name, address et birthyear) sans retour
 - Une opération getPersonByName qui prend en paramètre une chaîne de caractère et retourne un objet Person
 - Une opération getPersons sans paramètre en entrée et qui retourne un tableau d'objets Person
 - L'accès au service est réalisé par l'intermédiaire de messages SOAP (étudié en détail dans le prochain cours)
 - Le protocole utilisé pour l'échange des messages SOAP est HTTP et le style utilisé est du RPC

WSDL : Elément Types

- La définition des types peut également être importée à partir d'un fichier Schéma XML

```
<definitions
    name="Notebook"
    targetNamespace="http://notebookwebservice.lisi.ensma.fr/"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://notebookwebservice.lisi.ensma.fr/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <types>
        <xsd:schema targetNamespace="http://notebookwebservice.lisi.ensma.fr/">
            <xsd:complexType name="person">
                <xsd:sequence>
                    <xsd:element name="address" type="xs:string" minOccurs="0"/>
                    <xsd:element name="birthyear" type="xs:string" minOccurs="0"/>
                    <xsd:element name="name" type="xs:string" minOccurs="0"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="personArray" final="#all">
                <xsd:sequence>
                    <xsd:element name="item" type="tns:person" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </types>
    ...
</definitions>
```

Une personne est définie par une *adresse*, une *année de naissance* et un *nom*

Définition d'un type tableau de personne

WSDL : Elément Types

➤ Exemple : Définition des types pour Notebook service (bis)

```
<definitions
    targetNamespace="http://notebookwebservice.lisi.ensma.fr/"
    name="Notebook"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    ...>
    <types>
        <xsd:schema>
            <xsd:import namespace="http://notebookwebservice.lisi.ensma.fr/" schemaLocation="Notebook_schema1.xsd"/>
        </xsd:schema>
    </types>
    ...
</definitions>
```

Import le fichier XSD

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xss: schema version="1.0"
    targetNamespace="http://notebookwebservice.lisi.ensma.fr/"
    ...>
    <xss:complexType name="person">
        <xss:sequence>
            <xss:element name="address" type="xss:string" minOccurs="0"/>
            <xss:element name="birthyear" type="xss:string" minOccurs="0"/>
            <xss:element name="name" type="xss:string" minOccurs="0"/>
        </xss:sequence>
    </xss:complexType>
    <xss:complexType name="personArray" final="#all">
        <xss:sequence>
            <xss:element name="item" type="tns:person" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
        </xss:sequence>
    </xss:complexType>
</xss: schema>
```

WSDL : Elément Messages

- L'élément `<message>` permet de décrire les messages échangés par les services
 - Paramètres d'entrées des opérations
 - Paramètres de sorties
 - Exception
- Chaque `<message>` est identifié par un nom (attribut `name`) et est constitué d'un ensemble d'éléments `<part>`
- En quelque sorte un élément `<part>` correspond à un paramètre d'une opération
- Si une opération est décrite par plusieurs paramètres, plusieurs éléments `<part>` seront à définir
- L'élément `<part>` est défini par
 - un nom (attribut `name`)
 - un type (attribut `type`)

WSDL : Elément Messages

➤ Exemple : Définition des messages pour Notebook service

```
<definitions
    targetNamespace="http://notebookwebservice.lisi.ensma.fr/"
    name="Notebook"
    ...>
    <types>
        ...
    </types>
    <message name="addPersonWithComplexType">
        <part name="newPerson" type="tns:person"/>
    </message>
    <message name="addPersonWithComplexTypeResponse">
        <part name="addPersonWithComplexTypeResult" type="xsd:boolean"/>
    </message>
    <message name="addPersonWithSimpleType">
        <part name="name" type="xsd:string"/>
        <part name="address" type="xsd:string"/>
        <part name="birthyear" type="xsd:string"/>
    </message>
    <message name="getPerson">
        <part name="personName" type="xsd:string"/>
    </message>
    <message name="getPersonResponse">
        <part name="getPersonResult" type="tns:person"/>
    </message>
    <message name="getPersons"/>
    <message name="getPersonsResponse">
        <part name="getPersonsResult" type="tns:personArray"/>
    </message>
</definitions>
```

Message utilisé pour l'appel d'une opération avec une seule partie

Message utilisé pour le résultat d'une opération avec une seule partie

Message utilisé pour l'appel d'une opération avec trois parties

Une partie qui pointe sur un type défini par l'élément <types>

WSDL : Elément **portType** et sous élément **Operation**

- Un élément <portType> est un regroupement d'opérations et peut se comparer à une interface Java
- Caractéristique d'un élément <portType>
 - Identifiable par un nom (attribut name)
 - Composé de sous élément <operation>
- Une opération est comparable à une méthode Java
 - Identifiable par un nom (attribut name)
 - La description des paramètres est obtenue par une liste de messages
- Une opération exploite les messages via les sous éléments
 - <input> : message transmis au service
 - <output> : message produit par le service
 - <fault> : message d'erreur (très proche des exceptions)
- Chaque sous élément possède les attributs suivants
 - name : nom explicite donné au message (optionnel)
 - message : référence à un message (défini précédemment)
- La surcharge d'opération est autorisée sous condition
 - Messages <input> et/ou <output> soient différents

WSDL : Elément portType et sous élément Operation

➤ Exemple : Définition des Ports pour Notebook service

```
<definitions
    targetNamespace="http://notebookwebservice.lisi.ensma.fr/"
    name="Notebook"
    ...>
    <types>
        ...
    </types>
    <message>
        ...
    </message>
    <portType name="Notebook">
        <operation name="addPerson">
            <input message="tns:addPersonWithComplexType"/>
            <output message="tns:addPersonWithComplexTypeResponse"/>
        </operation>
        <operation name="addPerson" parameterOrder="name address birthyear">
            <input message="tns:addPersonWithSimpleType"/>
        </operation>
        <operation name="getPerson">
            <input message="tns:getPerson"/>
            <output message="tns:getPersonResponse"/>
        </operation>
        <operation name="getPersons">
            <input message="tns:getPersons"/>
            <output message="tns:getPersonsResponse"/>
        </operation>
    </portType>
</definitions>
```

L'opération *addPerson* est surchargée

Possibilité de fixer l'ordre des paramètres définis par cette opération

WSDL : Elément portType et sous élément Operation

➤ Possibilité de définir une opération suivant quatre modèles

➤ **One-way** : envoie de messages

- Le client du service envoie un message à l'opération et n'attend pas de réponse
- Uniquement un seul message utilisé <input>

```
<operation name="addPerson" parameterOrder="name address birthyear">
    <input message="tns:addPersonWithSimpleType"/>
</operation>
```

➤ **Request/Response** : question – réponse

- Le client du service envoie un message à l'opération et un message est retournée au client
- Un message <input>, un message <output> et un message <fault>

```
<operation name="addPerson">
    <input message="tns:addPersonWithComplexType"/>
    <output message="tns:addPersonWithComplexTypeResponse"/>
</operation>
```

➤ **Notification** : notification

- Le service envoie un message au client
- Uniquement un seul message utilisé <output>

```
<operation name="personStatus">
    <output message="trackingInformation"/>
</operation>
```

➤ **Solicit - response** : sollicitation - réponse

- Le client reçoit un message du service et répond au service
- Un message <output>, un message <input> et un message <fault>

```
<operation name="clientQuery">
    <output message="bandWithRequest"/>
    <input message="bandwidthInfo" />
    <fault message="faultMessasge" />
</operation>
```

WSDL : Elément Binding

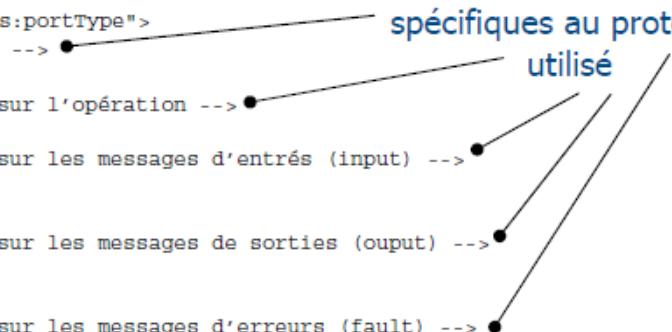
- Un élément `<binding>` permet de réaliser la partie concrète d'un élément `<portType>`
 - un nom (attribut `name`)
 - un `portType` (attribut `type`)
- Il décrit précisément le protocole à utiliser pour manipuler un élément `<portType>`
 - SOAP 1.1 et 1.2
 - HTTP GET & Post (pour le transfert d'images par exemple)
 - MIME
- Plusieurs éléments `<binding>` peuvent être définis de sorte qu'un élément `portType` peut être appelé de différentes manières
- La structure de l'élément `<binding>` dépend du protocole utilisé

WSDL : Elément Binding

- Structure générale de l'élément `<binding>` sans précision sur le protocole employé

```
<definitions>
  ...
  <binding name="NamePortBinding" type="tns:portType">
    <!-- Décrit le protocole à utiliser -->
    <operation name="operation1">
      <!-- Action du protocole sur l'opération -->
      <input>
        <!-- Action du protocole sur les messages d'entrés (input) -->
      </input>
      <output>
        <!-- Action du protocole sur les messages de sorties (output) -->
      </output>
      <fault>
        <!-- Action du protocole sur les messages d'erreurs (fault) -->
      </fault>
    </operation>
  </binding>
  ...
</definitions>
```

Ces informations sont spécifiques au protocole utilisé



- Le schéma XML de WSDL ne décrit pas les sous éléments de binding, operation, input, output et fault
- Ces éléments sont spécifiques aux protocoles utilisés

WSDL : Elément Binding

➤ Exemple : Définition d'un Binding SOAP 1.1

```
<definitions ...>
    <!-- Définition de la partie Abstraite du WSDL -->
    <binding name="NoteBookPortBinding" type="tns:Notebook">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
        <operation name="addPersonWithComplexType">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr/" />
            </input>
            <output>
                <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr/" />
            </output>
        </operation>
        <operation name="addPersonWithSimpleType">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr/" />
            </input>
        </operation>
        <operation name="getPerson">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr/" />
            </input>
            <output>
                <soap:body use="literal" namespace="http://notebookwebservice.lisi.ensma.fr/" />
            </output>
        </operation>
        ...
    </binding>
</definitions>
```

Eléments spécifiques au protocole SOAP 1.1

WSDL : Elément Service et Port

- Un élément service définit l'ensemble des points d'entrée du Service Web, en regroupant des éléments <port>
- L'élément <port> permet de spécifier une adresse pour un binding donné
- Un port est défini par deux attributs
 - name : nom du port
 - binding : nom du binding (défini précédemment)
- Le corps de l'élément <port> est spécifique au protocole utilisé pour définir le binding
- Dans le cas d'un binding de type SOAP, un élément <soap:address> précise l'URI du port
- Il est par conséquent possible d'appeler un service à des endroits différents (plusieurs éléments port)



WorkShop :

Mise en œuvre

des web services

avec JAX-WS

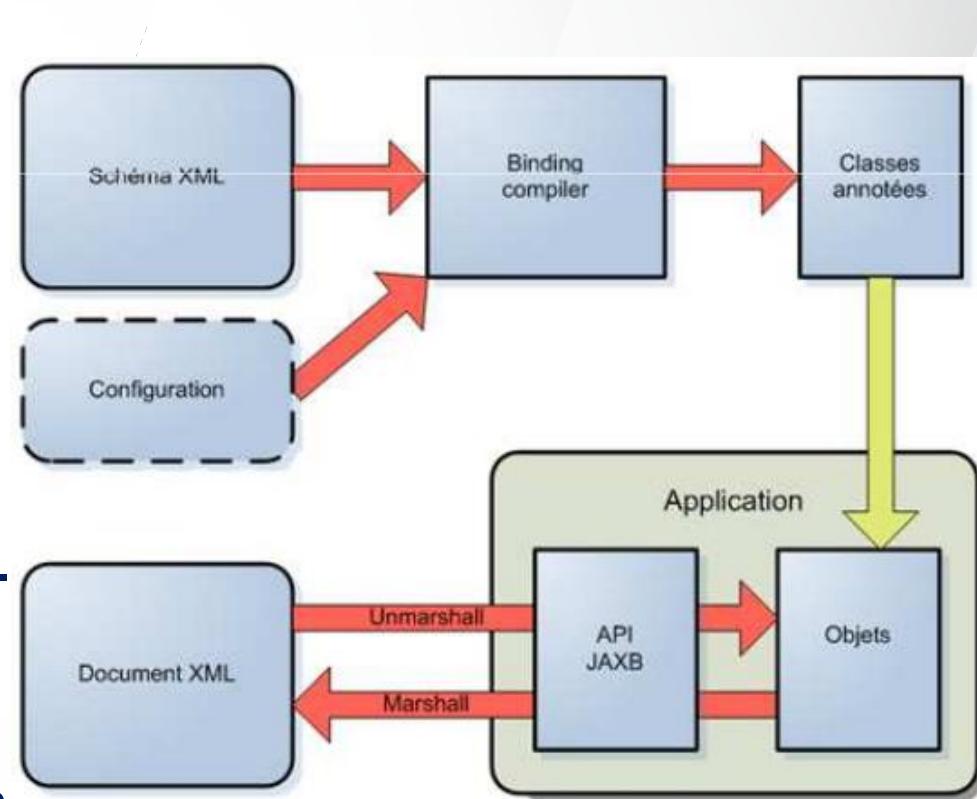
JAX-WS

- JAX-WS est la nouvelle appellation de JAX-RPC (Java API for XML Based RPC) qui permet de développer très simplement des services web en Java.

```
@WebService(serviceName="BanqueWS")
public class BanqueService {
    @WebMethod(operationName="ConversionEuroToDh")
    public double conversion(@WebParam(name="montant")double mt){
        return mt*11;
    }
    @WebMethod
    public String test(){    return "Test";    }
    @WebMethod
    public Compte getCompte(){ return new Compte (1,7000);    }
    @WebMethod
    public List<Compte> getComptes(){
        List<Compte> cptes=new ArrayList<Compte>();
        cptes.add (new Compte (1,7000)); cptes.add (new Compte (2,9000));
        return cptes;
    }
}
```

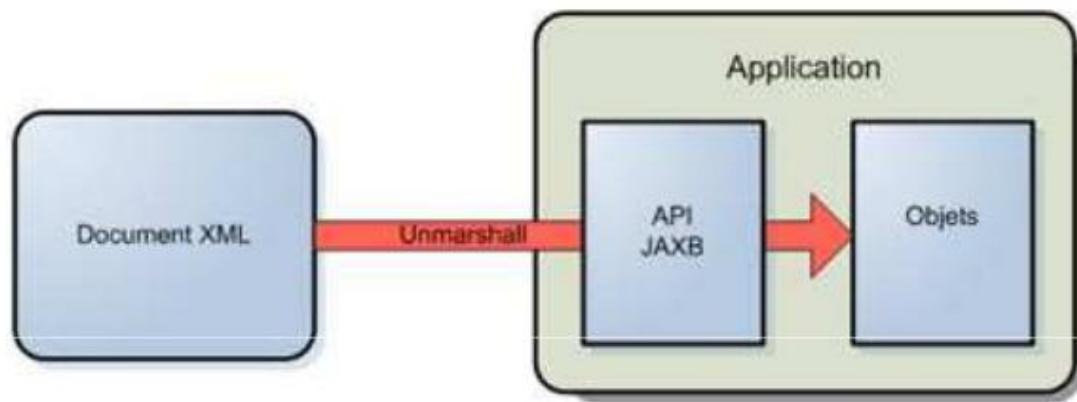
JAX-WS / JAXB

- JAX-WS s'appuie sur l'API JAXB 2.0 pour tout ce qui concerne la correspondance entre document XML et objets Java.
- JAXB 2.0 permet de mapper des objets Java dans un document XML et vice versa.
- Il permet aussi de générer des classes Java à partir un schéma XML et vice et versa.

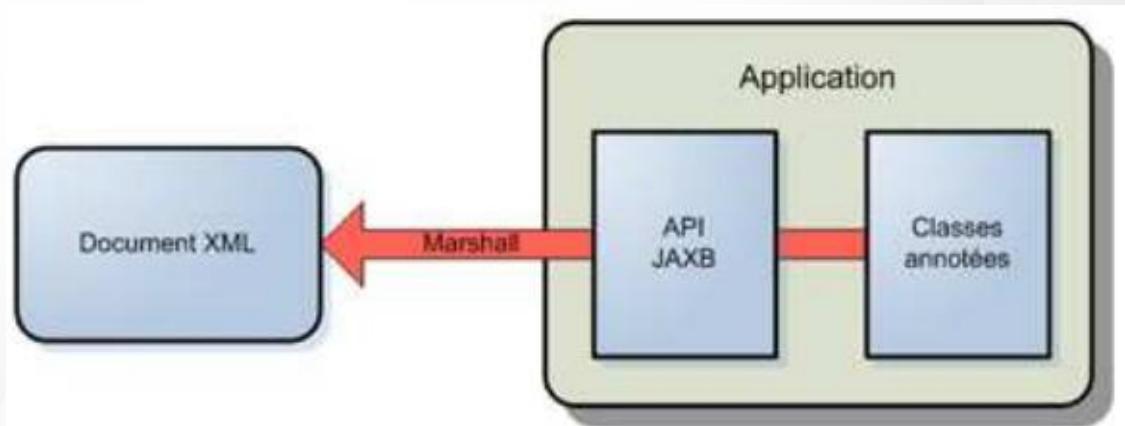


Principe de JAXB

- Le mapping d'un document XML à des objets (unmarshal)



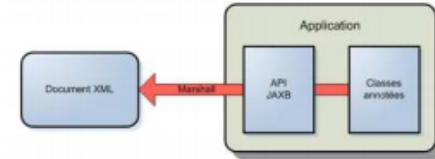
- La création d'un document XML à partir d'objets (marshal)



Générer XML à partir des objets Java avec JAXB

```
package ws;
import java.io.File; import java.util.Date;
import javax.xml.bind.*;
public class Banque {
    public static void main(String[] args) throws Exception {
        JAXBContext context=JAXBContext.newInstance(Compte.class);
        Marshaller marshaller=context.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,true);
        Compte cp=new Compte(1,8000,new Date());
        marshaller.marshal(cp,new File("comptes.xml"));
    }
}
```

```
package ws;
import java.util.Date;
import javax.xml.bind.annotation.*;
@XmlRootElement
public class Compte {
    private int code;
    private float solde;
    private Date dateCreation;
    // Constructeur sans paramètre
    // Constructeur avec paramètres
    // Getters et Setters
}
```



Fichier XML Généré : comptes.xml

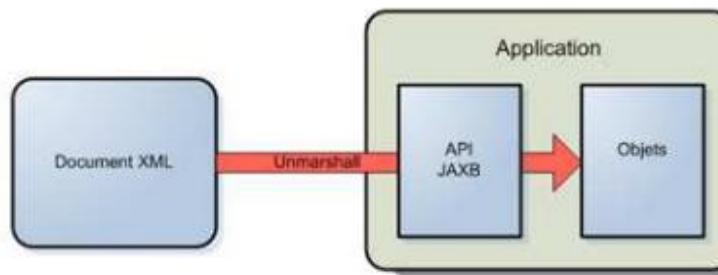
```
<?xml version="1.0" encoding="UTF-8"?>
<compte>
    <code>1</code>
    <dateCreation>
        2014-01-16T12:33:16.960Z
    </dateCreation>
    <solde>8000.0</solde>
</compte>
```

Générer des objets java à partir des données XML

```
package ws;
import java.io.*;
import javax.xml.bind.*;
public class Banque2 {
    public static void main(String[] args) throws Exception {
        JAXBContext jc=JAXBContext.newInstance(Compte.class);
        Unmarshaller unmarshaller=jc.createUnmarshaller();
        Compte cp=(Compte) unmarshaller.unmarshal(new File("comptes.xml"));
        System.out.println(cp.getCode()+" - "+cp.getSolde()+
                           "- "+cp.getDateCreation());
    }
}
```

Fichier XML Source : comptes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<compte>
    <code>1</code>
    <dateCreation>
        2014-01-16T12:33:16.960Z
    </dateCreation>
    <solde>8000.0</solde>
</compte>
```



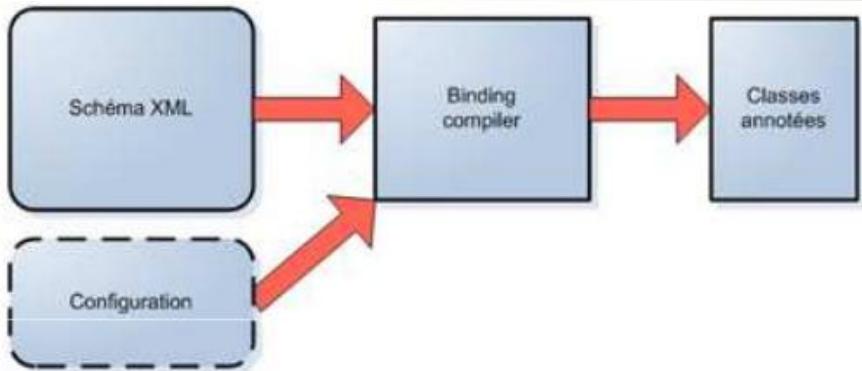
Générer un schéma XML à partir d'une classe avec JAXB

```
package ws;
import java.io.*;
import javax.xml.bind.*;import javax.xml.transform.Result;
import javax.xml.transform.stream.StreamResult;
public class Banque {
    public static void main(String[] args) throws Exception {
        JAXBContext context=JAXBContext.newInstance(Compte.class);
        context.generateSchema(new SchemaOutputResolver() {
            @Override
            public Result createOutput(String namespaceUri, String suggestedFileName)
throws IOException {
                File f=new File("compte.xsd");
                StreamResult result=new StreamResult(f);
                result.setSystemId(f.getName());
                return result;
            }
       ));}}
```

```
<?xml version="1.0" encoding="UTF-8" standaLone="yes"?>
<xss: schema version="1.0" xmlns:xss="http://www.w3.org/2001/XMLSchema">
    <xss:element name="compte" type="compte"/>
    <xss:complexType name="compte">
        <xss:sequence>
            <xss:element name="code" type="xs:int"/>
            <xss:element name="dateCreation" type="xs:dateTime" minOccurs="0"/>
            <xss:element name="solde" type="xs:float"/>
        </xss:sequence>
    </xss:complexType>
</xss: schema>
```

Génération des classes à partir d'un schéma XML

- Pour permettre l'utilisation et la manipulation d'un document XML, JAXB propose de générer un ensemble de classes à partir du schéma XML du document.
- L'implémentation de référence fournit l'outil **xjc** pour générer les classes à partir d'un schéma XML.
- L'utilisation la plus simple de l'outil xjc est de lui fournir simplement le fichier qui contient le schéma XML du document à utiliser



```
C:\Windows\system32\cmd.exe
C:\Users\youssef\w\JB2\src>xjc compte.xsd
parsing a schema...
compiling a schema...
generated\Compte.java
generated\ObjectFactory.java
C:\Users\youssef\w\JB2\src>
```

A screenshot of a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window displays the output of the 'xjc' command. The command 'xjc compte.xsd' is run, followed by the output: 'parsing a schema...', 'compiling a schema...', 'generated\Compte.java', and 'generated\ObjectFactory.java'. The command prompt's path is 'C:\Users\youssef\w\JB2\src'.

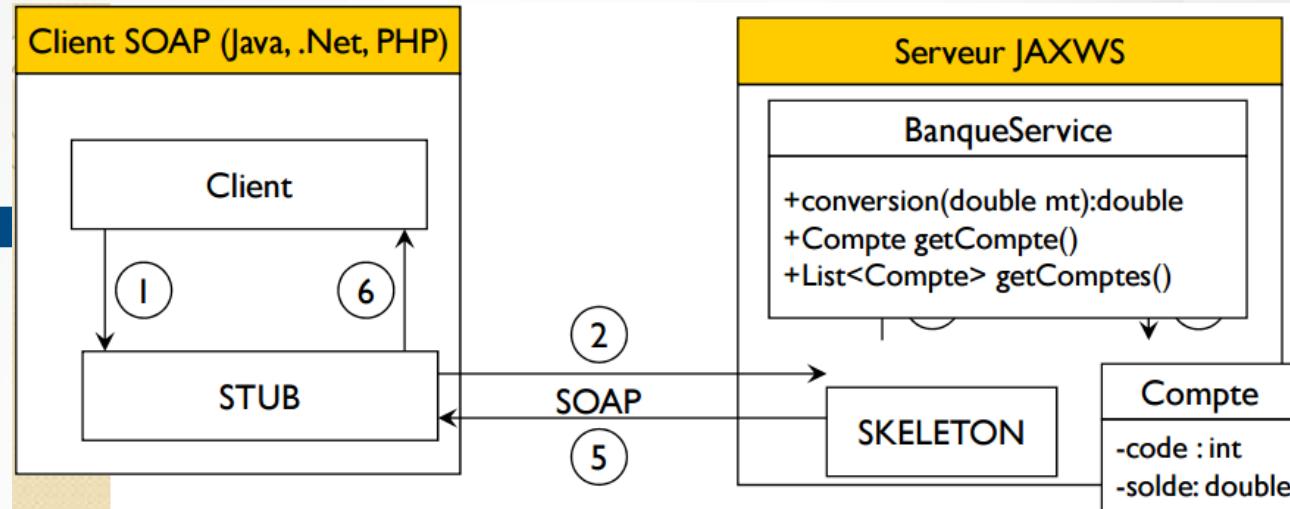
Comment développer un web service SOAP avec JAX-WS

1. Créer le service Web
 1. Développer le web service
 2. Déployer le web service
 - Un serveur HTTP
 - Un Conteneur WS (JaxWS, AXIS, CXF, etc...)
 3. Tester le web service avec un analyseur SOAP
 - SoapUI,
 - Oxygen, etc...
 4. Créer les clients :
 - Un client Java
 - Un client .Net
 - Un client PHP ...

Workshop

- Créer un web service java en utilisant JaxWS qui permet de :
 - Convertir un montant de l'euro en Dollar
 - Consulter un compte sachant son code
 - Consulter une liste de comptes
- Tester le web service avec un analyseur SOAP
- Créer un client Java

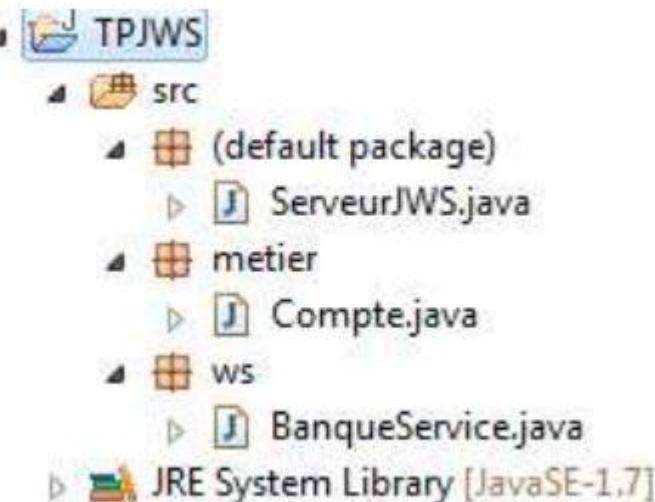
Architecture



1. Le client demande au stub de faire appel à la méthode conversion(12)
2. Le Stub se connecte au Skeleton et lui envoie une requête SOAP
3. Le Skeleton fait appel à la méthode du web service
4. Le web service retourne le résultat au Skeleton
5. Le Skeleton envoie le résultat dans une la réponse SOAP au Stub
6. Le Stub fourni le résultat au client

Développer le web service avec JAX-WS

- Outils à installer :
 - JDK1.7
 - Editeur java : Eclipse.
- Créer un projet java en utilisant JDK1.7 comme compilateur et environnement d'exécution java.
 - Structure du projet :



Implémentation de la classe Compte

```
package metier;  
import java.util.Date;  
import javax.xml.bind.annotation.XmlRootElement;  
import javax.xml.bind.annotation.XmlTransient;  
@XmlElement  
@XmlAccessorType(XmlAccessType.FIELD)  
public class Compte {  
    private Long code;  
    private double solde;  
@XmlTransient  
    private Date dateCreation;  
    // Constructeur sans paramètre et avec paramètre  
    // Getters et setters  
}
```

Implémentation de la classe Compte

- Avec assistant d'Eclipse continuer à créer :
 - Un constructeur par défaut
 - Un constructeur avec des paramètres
 - Tous les getters et les setters des attributs

Implémentation du web service

```
package ws;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

import metier.Compte;

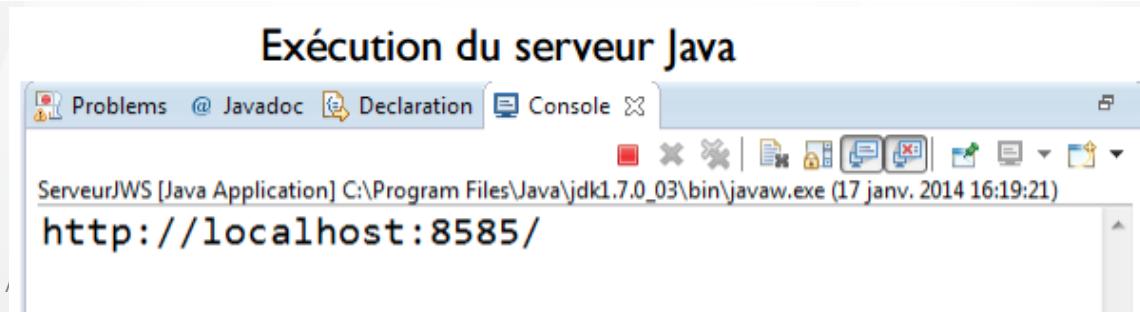
@WebService(serviceName = "BanqueWS")
public class BanqueService {
    @WebMethod(operationName = "ConversionEuroToD")
    public double conversion(@WebParam(name = "montant") double mt) {
        return mt * 1.11;
    }

    @WebMethod
    public Compte getCompte(@WebParam(name = "code") Long code) {
        return new Compte(code, 7000, new Date());
    }

    @WebMethod
    public List<Compte> getComptes() {
        List<Compte> cptes = new ArrayList<Compte>();
        cptes.add(new Compte(1L, 7000, new Date()));
        cptes.add(new Compte(2L, 7000, new Date()));
        return cptes;
    }
}
```

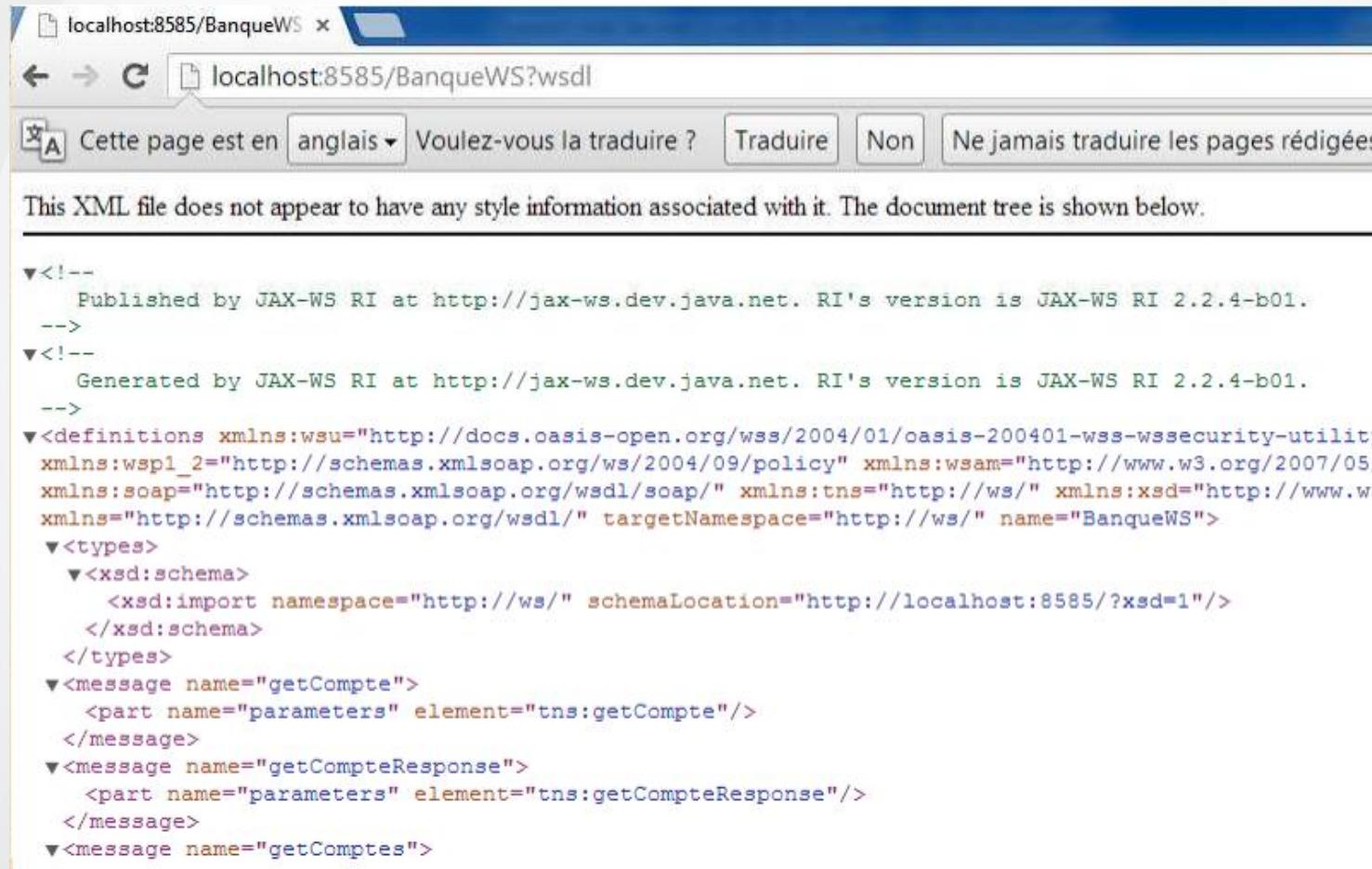
Simple Serveur JAX WS

```
import javax.xml.ws.Endpoint;
import ws.BanqueService;
public class ServeurJWS {
public static void main(String[] args) {
    String url="http://localhost:8585/";
    Endpoint.publish(url, new BanqueService());
    System.out.println(url);
}
}
```



Analyser le WSDL

- Pour Visualiser le WSDL, vous pouvez utiliser un navigateur web

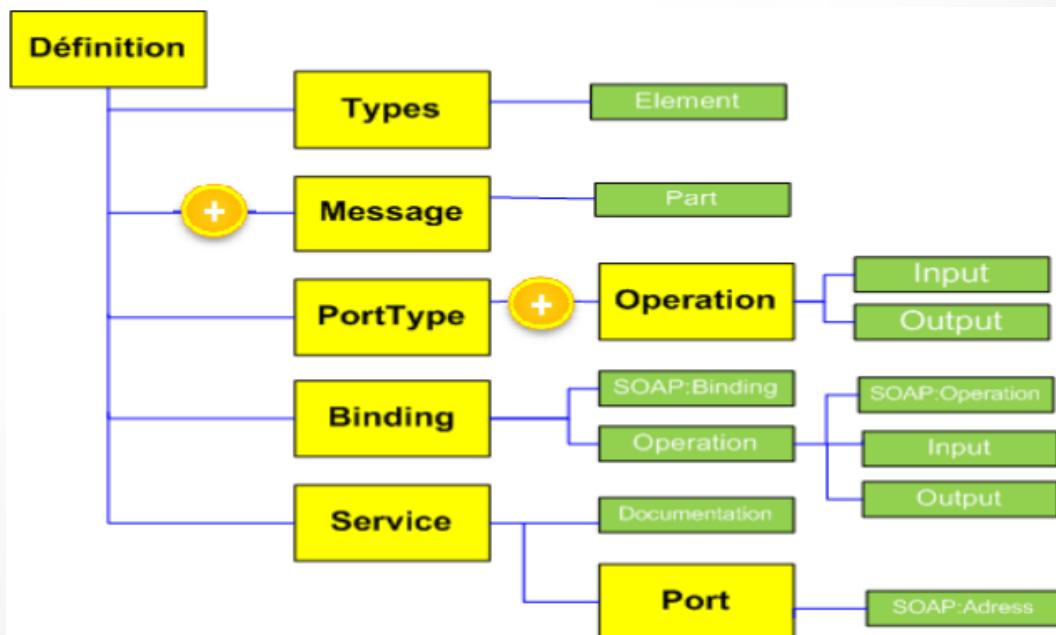


The screenshot shows a web browser window with the URL `localhost:8585/BanqueWS?wsdl`. The page content is an XML document representing the Web Services Description Language (WSDL) for the `BanqueWS` service.

```
<!--
Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
-->
<!--
Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility"
    xmlns:wspl_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://ws/" xmlns:xsd="http://www.w3
    xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://ws/" name="BanqueWS">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://ws/" schemaLocation="http://localhost:8585/?xsd=1"/>
        </xsd:schema>
    </types>
    <message name="getCompte">
        <part name="parameters" element="tns:getCompte"/>
    </message>
    <message name="getCompteResponse">
        <part name="parameters" element="tns:getCompteResponse"/>
    </message>
    <message name="getComptes">
```

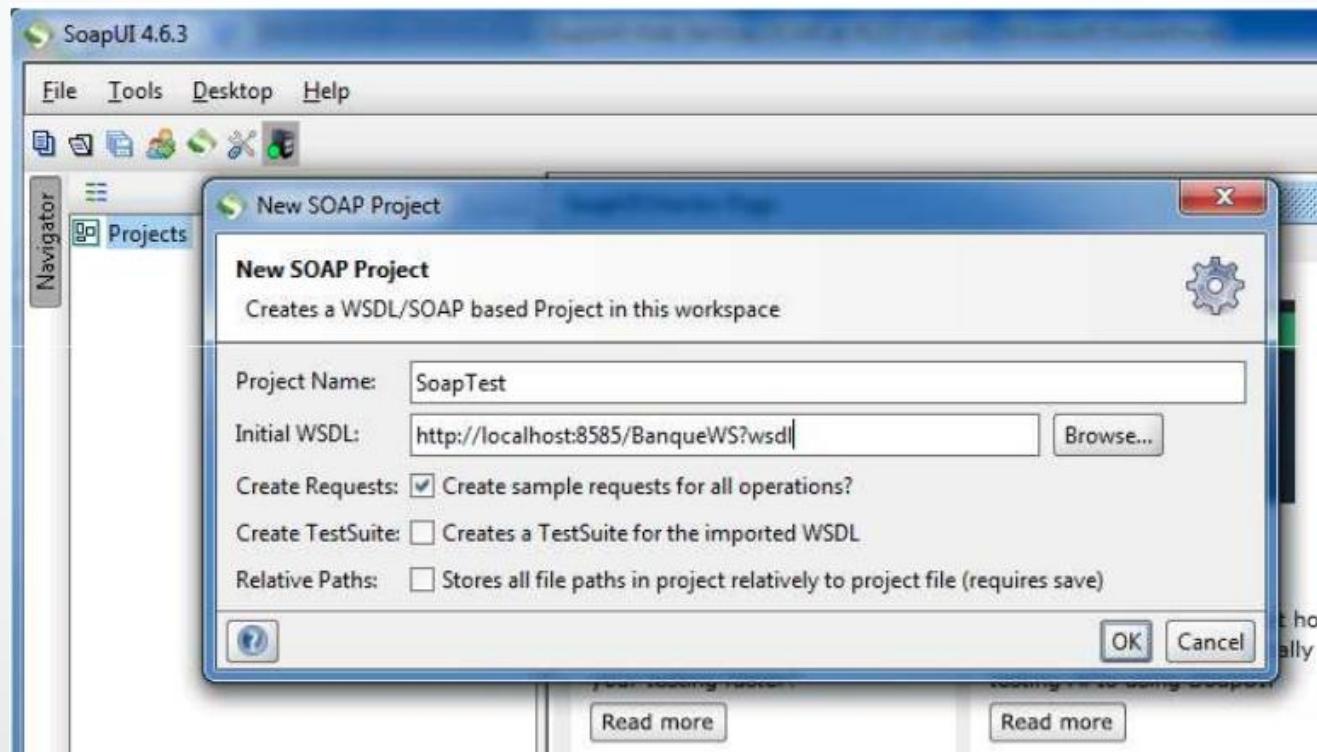
Structure du WSDL

- Un document WSDL se compose d'un ensemble d'éléments décrivant les types de données utilisés par le service, les messages que le service peut recevoir, ainsi que les liaisons SOAP associées à chaque message.
- Le schéma suivant illustre la structure du langage WSDL qui est un document XML, en décrivant les relations entre les sections constituant un document WSDL.

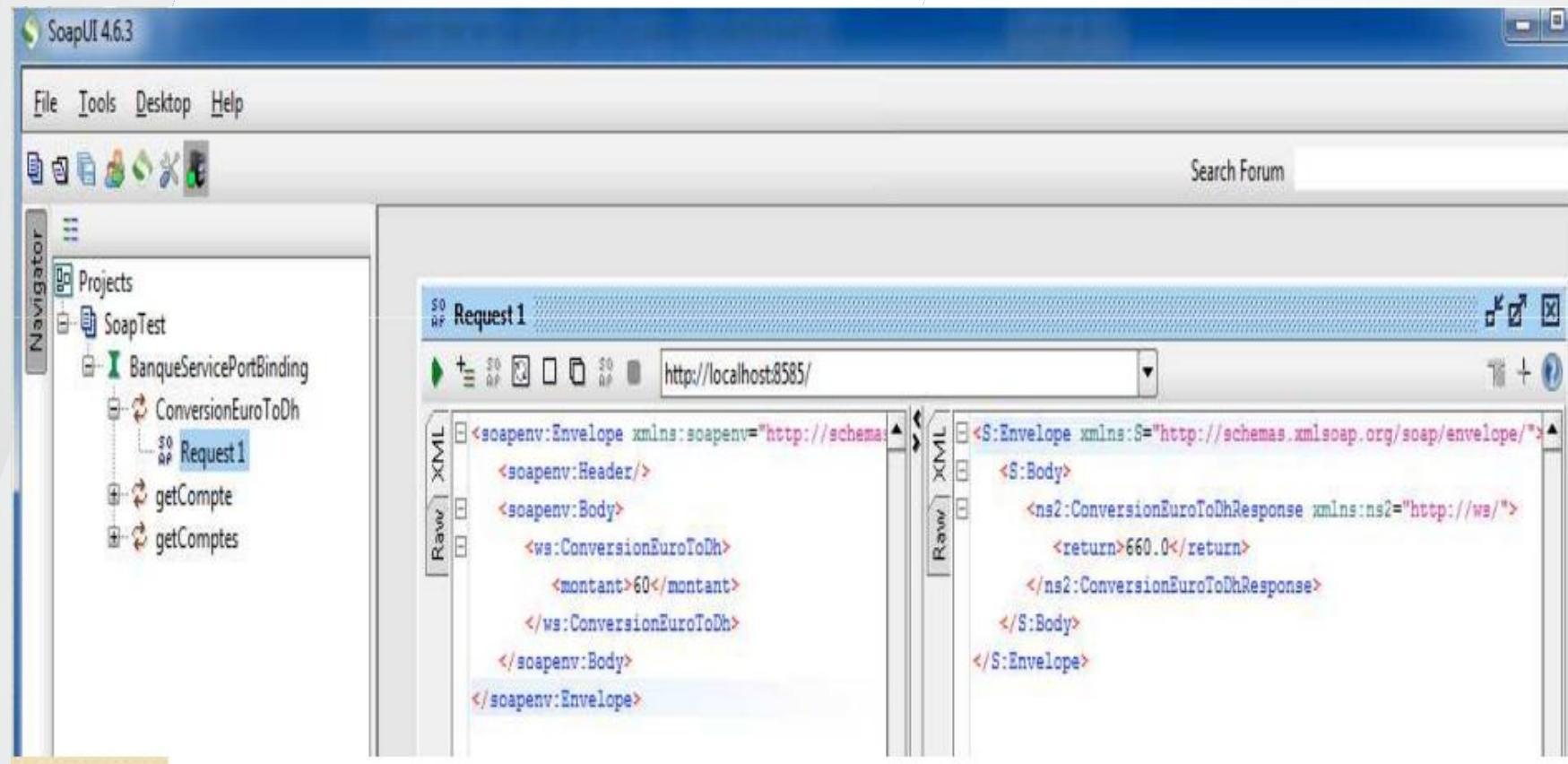


Tester les méthodes du web service

- Tester les méthodes du web service avec un analyseur SOAP : SoapUI



Tester les méthodes du web service



Tester la méthode conversion

➤ Requête SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws/">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:ConversionEuroToD>
      <montant>60</montant>
    </ws:ConversionEuroToD>
  </soapenv:Body>
</soapenv:Envelope>
```

➤ Réponse SOAP

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:ConversionEuroToDResponse xmlns:ns2="http://ws/">
      <return>66.6000000000001</return>
    </ns2:ConversionEuroToDResponse>
  </S:Body>
</S:Envelope>
```

Tester la méthode getCompte

Requête SOAP

- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws/">
 <soapenv:Header/>
 <soapenv:Body>
 <ws:getCompte>
 <code>2</code>
 </ws:getCompte>
 </soapenv:Body>
</soapenv:Envelope>

Réponse SOAP

- <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <ns2:getCompteResponse xmlns:ns2="http://ws/">
 <return>
 <code>2</code>
 <solde>7000.0</solde>
 </return>
 </ns2:getCompteResponse>
 </S:Body>
</S:Envelope>

Tester la méthode getComptes

Requête SOAP

- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws/">
 <soapenv:Header/>
 <soapenv:Body>
 <ws:getComptes/>
 </soapenv:Body>
</soapenv:Envelope>

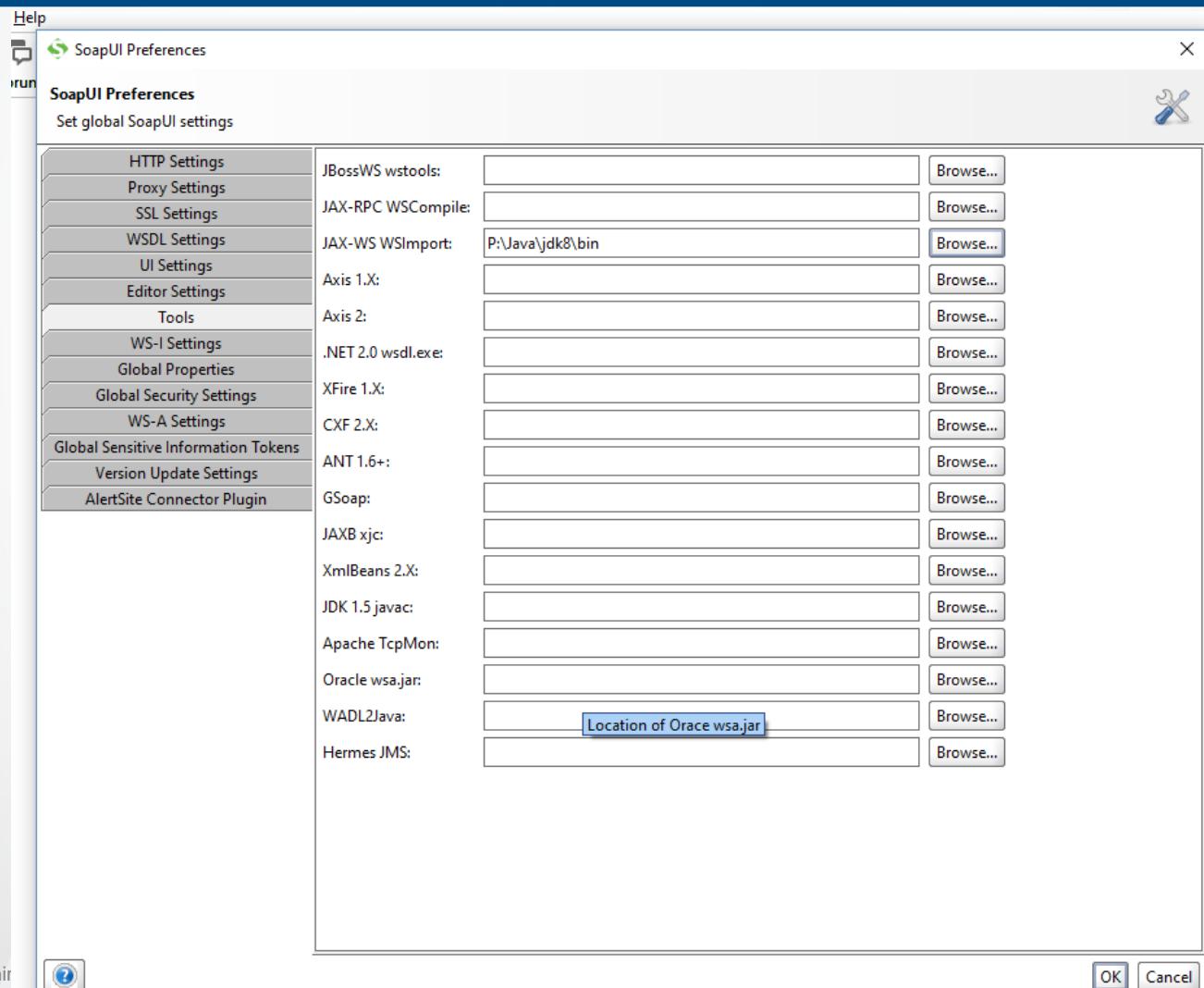
Réponse SOAP

- <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
 <ns2:getComptesResponse xmlns:ns2="http://ws/">
 <return>
 <code>1</code>
 <solde>7000.0</solde>
 </return>
 <return>
 <code>2</code>
 <solde>7000.0</solde>
 </return>
 </ns2:getComptesResponse>
 </S:Body>
</S:Envelope>

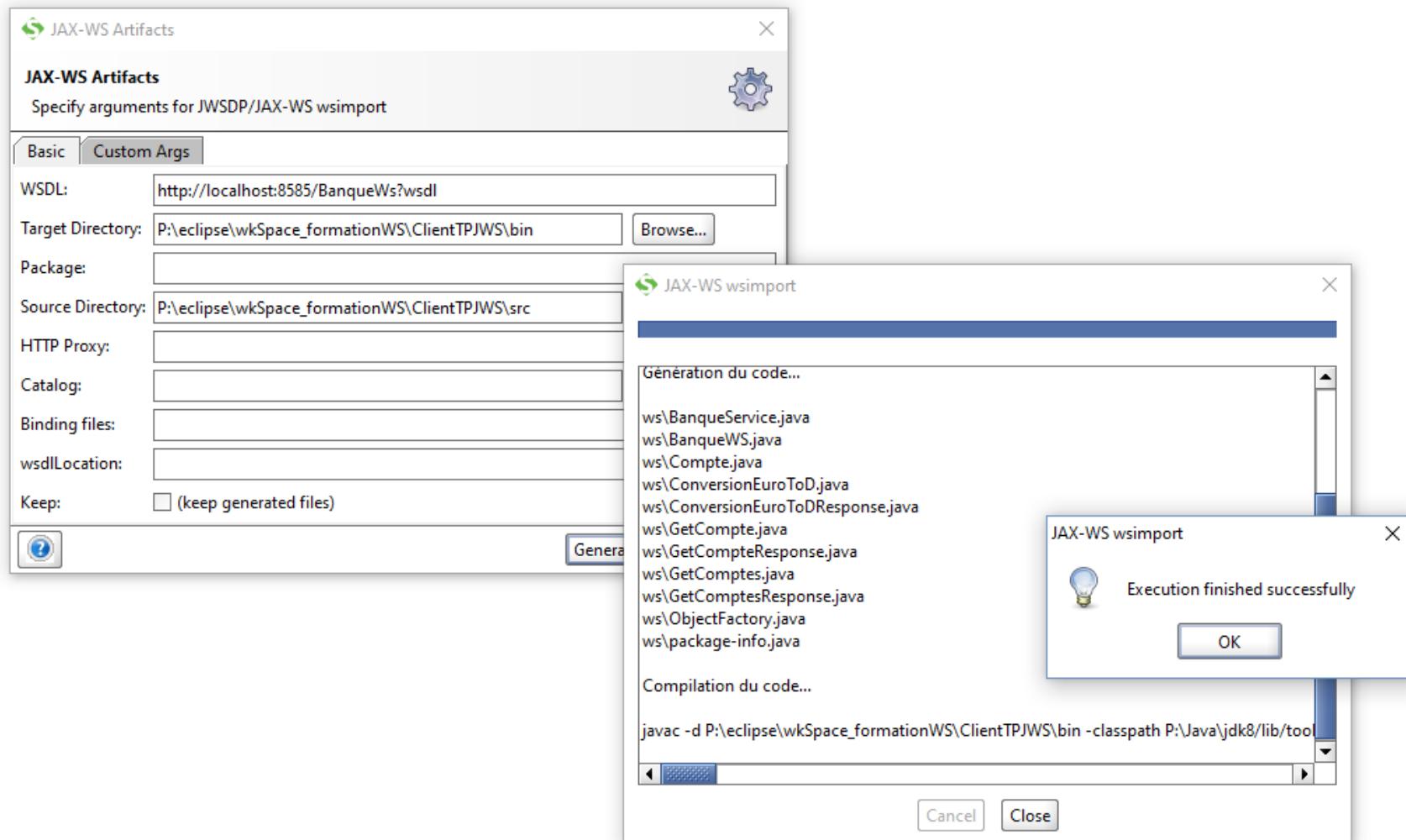
Client Java

- Créer un nouveau projet Java
- Générer un proxy
 - SoapUI est l'un des outils qui peuvent être utilisés pour générer les artefacts client en utilisant différents Framework (Axis, CXF, JaxWS, etc...)
 - Le JDK fournit une commande simple qui permet de générer un STUB JaxWS pour l'accès à un web service. Cette commande s'appelle wsimport.
 - SoapUI a besoin de savoir le chemin de cette commande
 - Avec la commande File > Preferences > Tools , vous pouvez configurer le chemin comme le montre la figure suivante :

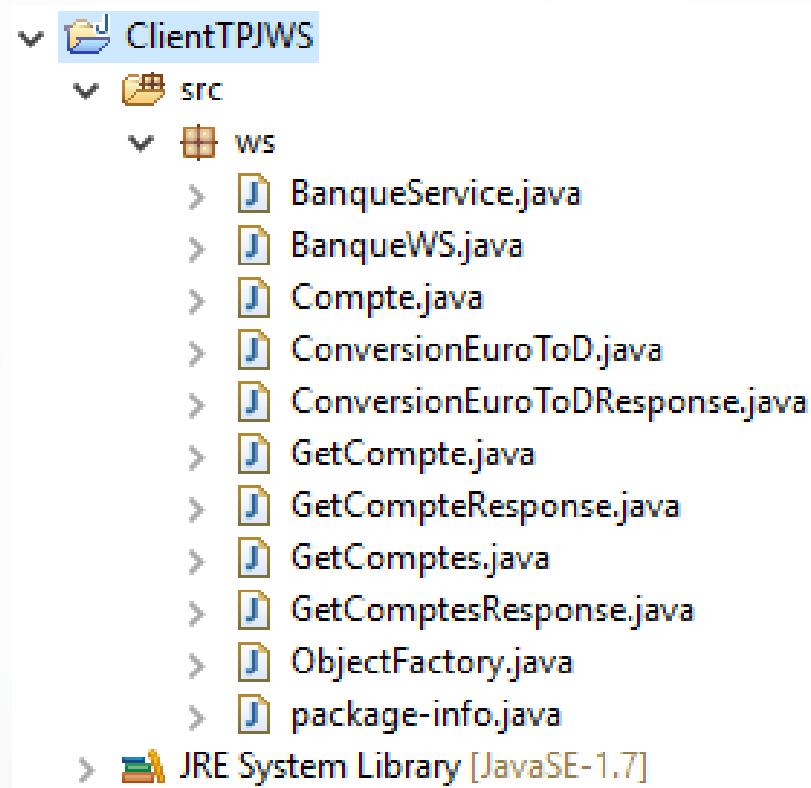
Préférence générale de SoapUI



Générer le STUB JaxWS



Fichiers Générés

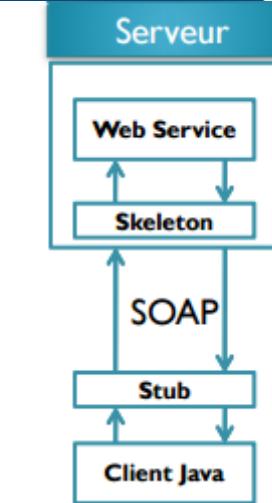


Client Java

```
import java.util.List;

import ws.BanqueService;
import ws.BanqueWS;
import ws.Compte;

public class ClientWS {
    public static void main(String[] args) {
        BanqueService stub = new BanqueWS().getBanqueServicePort();
        System.out.println("Conversion");
        System.out.println(stub.conversionEuroToD(9000));
        System.out.println("Consulter un compte");
        Compte cp = stub.getCompte(2L);
        System.out.println("Solde=" + cp.getSolde());
        System.out.println("Liste des comptes");
        List<Compte> cptes = stub.getComptes();
        for (Compte c : cptes) {
            System.out.println(c.getCode() + "----" + c.getSolde());
        }
    }
}
```

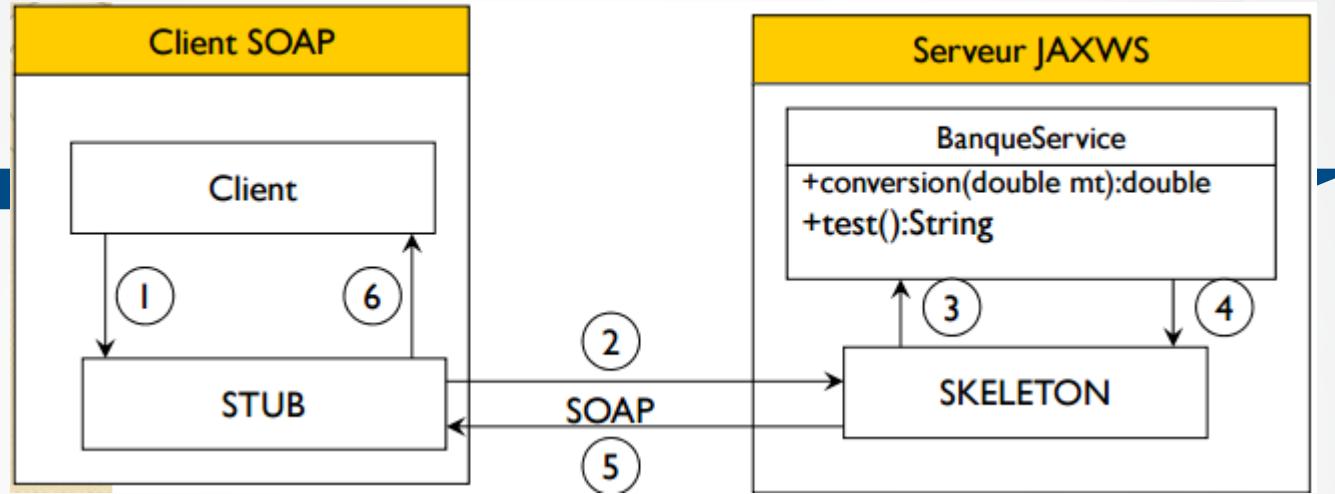


Problems @ Javadoc Declaration Console

<terminated> ClientWS [Java Application] P:\eclipse\eclipse\jre\bin\javaw.exe ()

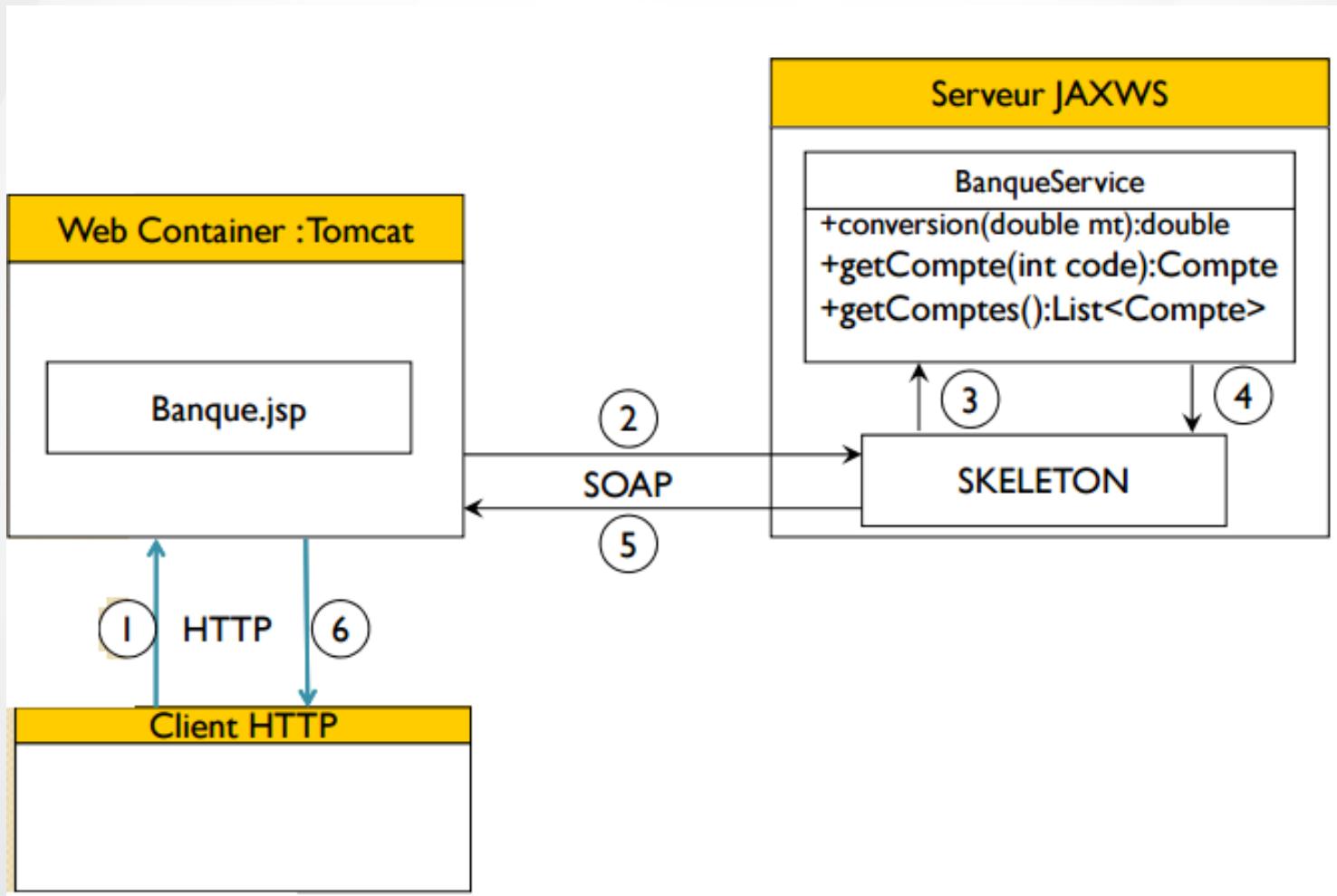
Conversion
9990.0
Consulter un compte
Solde=7000.0
Liste des comptes
1----7000.0
2----7000.0

Architecture

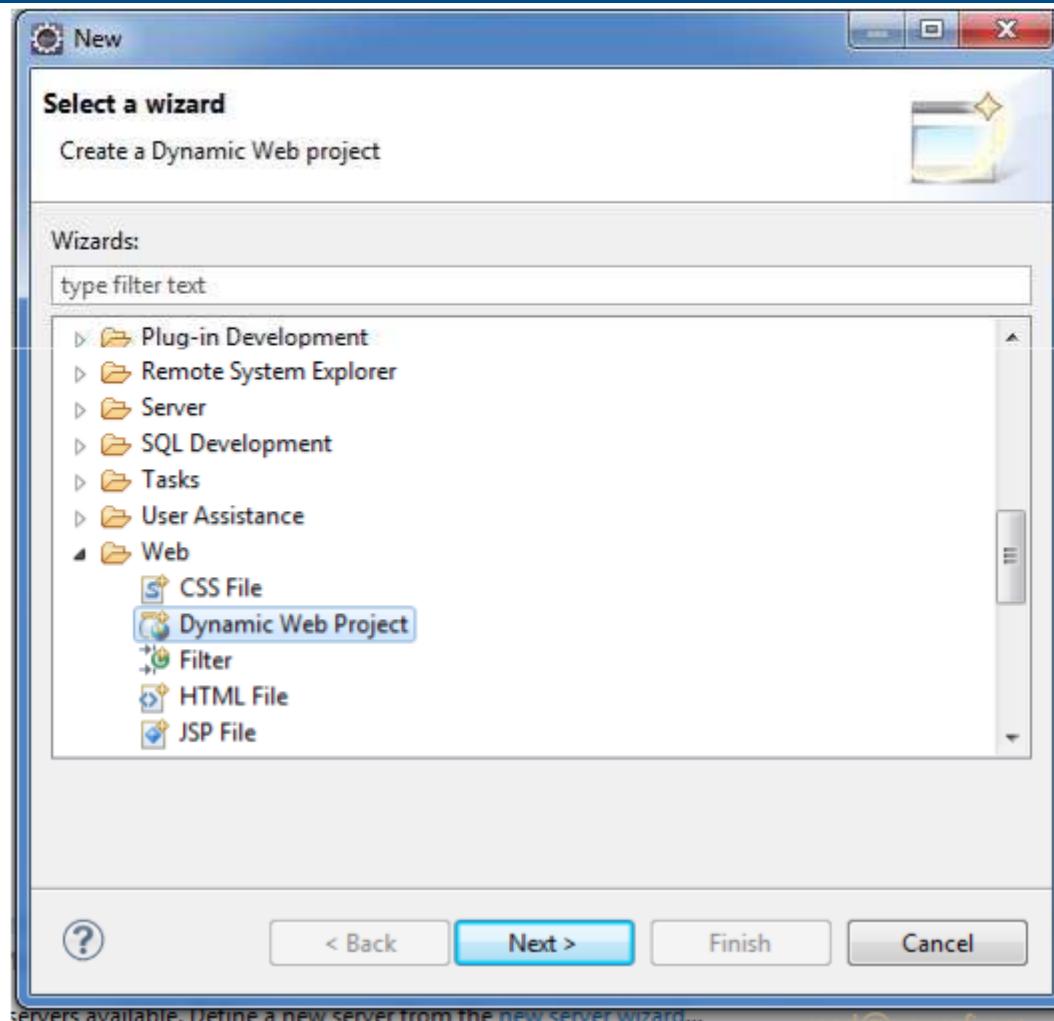


1. Le client demande au stub de faire appel à la méthode conversion(12)
2. Le Stub se connecte au Skeleton et lui envoie une requête SOAP
3. Le Skeleton fait appel à la méthode du web service
4. Le web service retourne le résultat au Skeleton
5. Le Skeleton envoie le résultat dans la réponse SOAP au Stub
6. Le Stub fournit le résultat au client

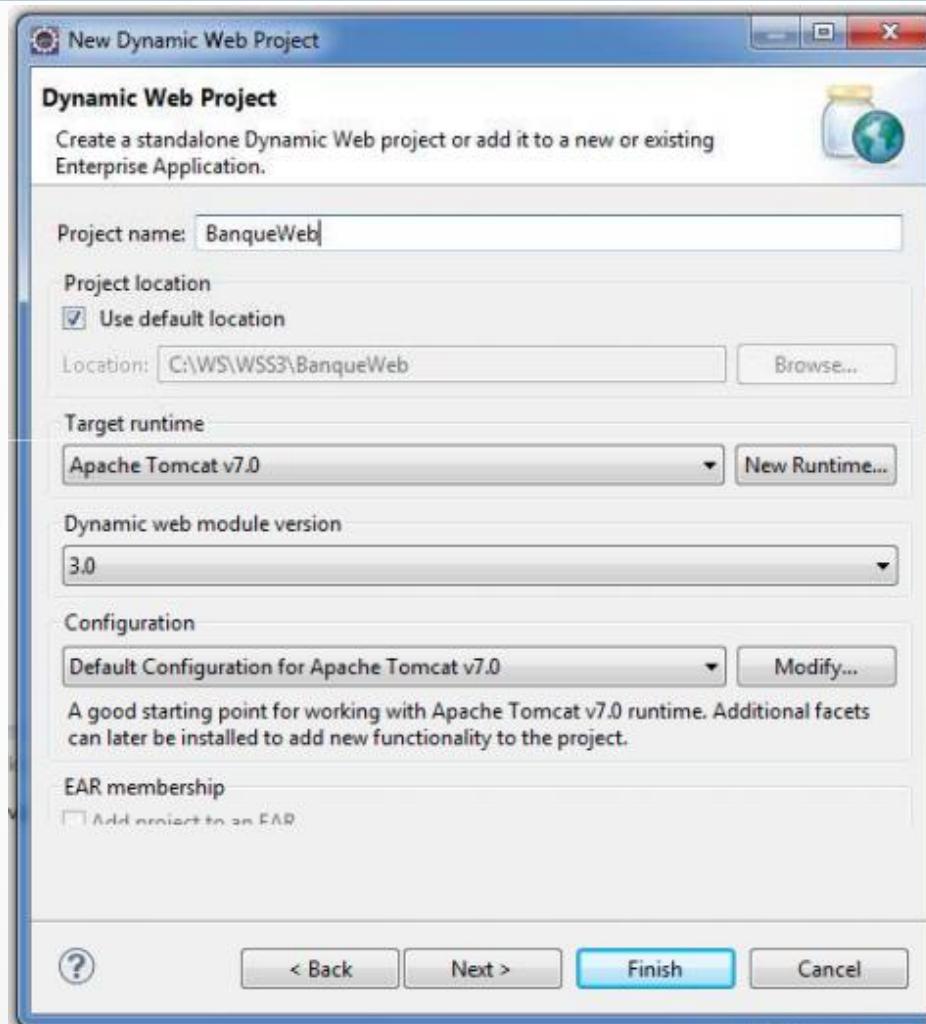
Client JSP



Création d'un projet Web Dynamique basé sur Tomcat 7 ou plus

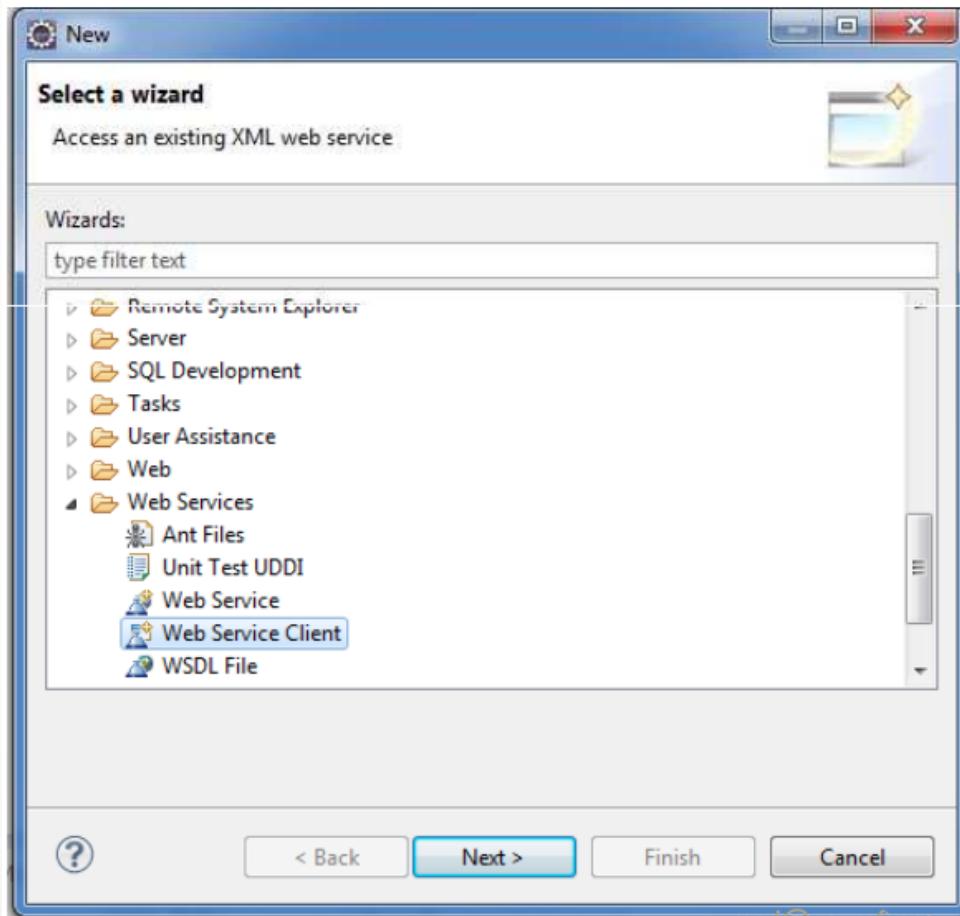


Projet Web Dynamique



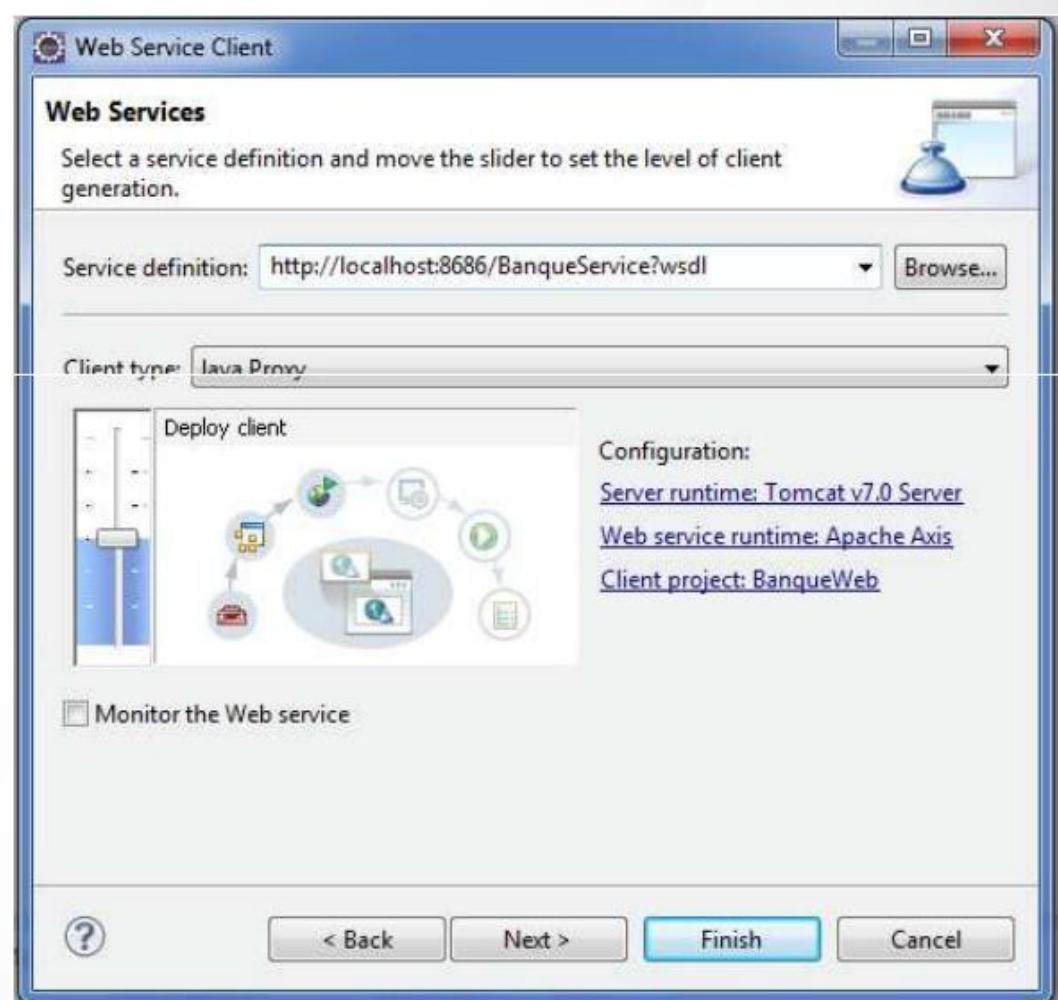
Générer un proxy (Stub) pour le web Service

- Fichier > Nouveau > Web Service Client



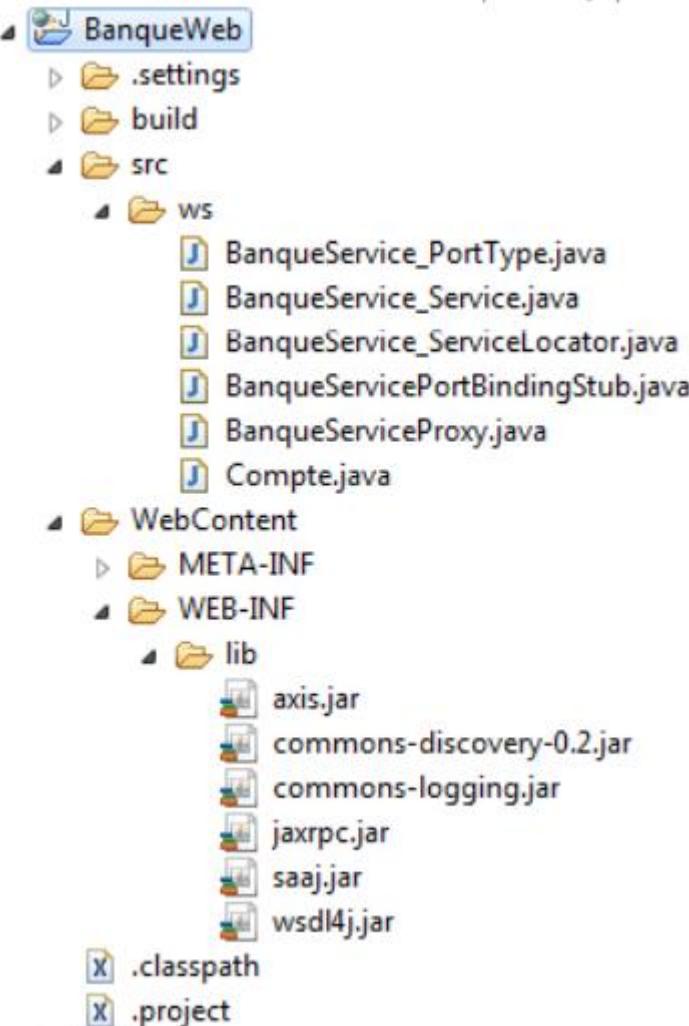
Génération du proxy

- Générer le proxy à partir du WSDL

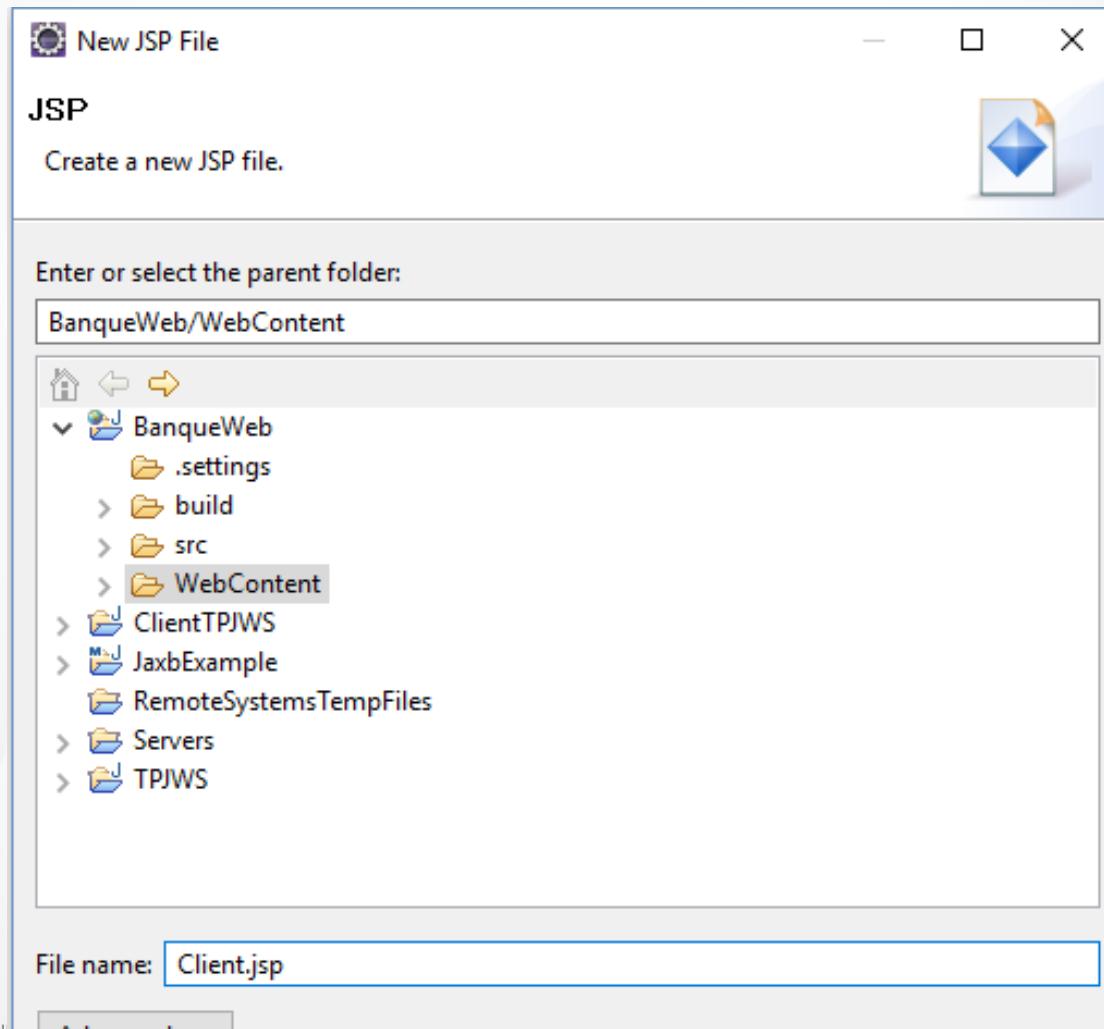


Fichier Générés

- Le proxy généré est basé sur AXIS



Client JSP



Client JSP

```
<%@page import="ws.BanqueServiceProxy"%>
<%
    double montant = 0;
    double resultat = 0;
    if (request.getParameter("montant") != null) {
        montant = Double.parseDouble(request.getParameter("montant"));
        BanqueServiceProxy service = new BanqueServiceProxy();
        resultat = service.conversionEuroToD(montant);
    }
%>
<html>
<body>
    <form action="Client.jsp">
        Montant :<input type="text" name="montant" value="<%=montant%>">
        <input type="submit" value="OK">
    </form>
    <%=montant%>
    en Euro est égale à
    <%=resultat%>
    en Dollars
</body>
</html>
```

The screenshot shows the Java code for Client.jsp. Below the code, there's a browser window with developer tools open. The address bar shows the URL: <http://localhost:8080/BanqueWeb/Client.jsp?montant=5>. The browser's status bar indicates the page is loaded from memory. The page content displays the input field with 'Montant : 5.0' and an 'OK' button. The result of the conversion is shown below the form: '5.0 en Euro est égale à 5.5500000000000001 en Dollars'.

Vue d'ensemble du jour

1

- Connaitre une architecture SOA

2

- Familiarisez-vous avec les protocoles Internet

3

- De l'XML au SOAP/WSDL

Ressources

➤ Services Web avec SOAP, WSDL, UDDI, ebXML

- Auteur : Jean-Marie Chauvet
- Éditeur : Eyrolles
- Edition : Mars 2003 - 524 pages - ISBN : 2212110472



➤ Understanding Web Services : XML, WSDL,

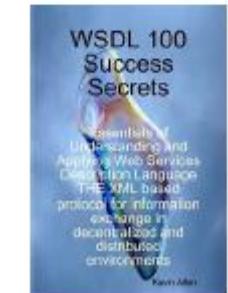
- Auteur : Eric Newcomer
- Éditeur : Addison-Wesley
- Edition : Mai 2002 - 368 pages - ISBN : 0201750813



Understanding Web Services
XML, WSDL, SOAP, and UDDI
Eric Newcomer

➤ WSDL 100 Success Secrets Essentials of ...

- Auteur : Kevin Allen
- Éditeur : Emero Pty Ltd
- Edition : Juillet 2008 - 144 pages - ISBN : 1921523220



Questions?



Sommaire

- **Chapitre 1 : INTRODUCTION**
- **Chapitre 2 : Technologies des services Web**
 - Exemple pratique de création et de déploiement d'un Web service SOAP et de son client
- **Chapitre 3 : Service Web REST**
- **Chapitre 4 : Développement de service web REST avec JAX-RS**
 - Etude de cas : *Développement d'un service RESTful retournant un flux JSON. + Invocation du service et parsing du résultat en Java.*
- **Chapitre 5 : la gestion des Persistances dans un WEB SERVICE REST**
 - Etude de cas : Client : JAX-RS, JaxB, JAVA
- **Chapitre 6 : Sécurité des Web services SOAP VS REST**

Chapitre 3

Service Web Rest

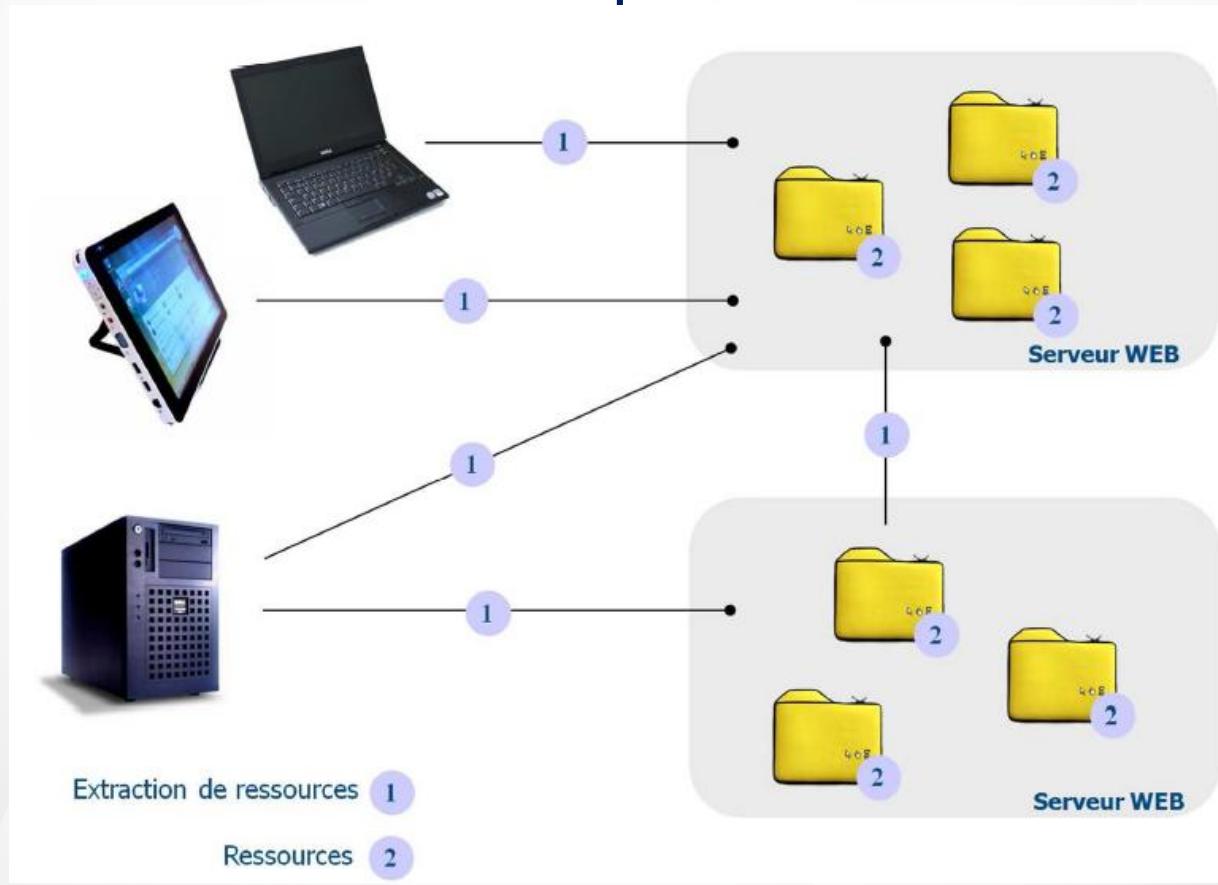
Objectif du chapitre

- C'est quoi REST
 - Ressources
 - Verbes
 - Représentations
- Exemples
- REST VS SOAP
- Outils

C'est quoi REST

REST : L'utilisation du Web aujourd'hui ...

- Les ressources sont récupérées au travers les URLs

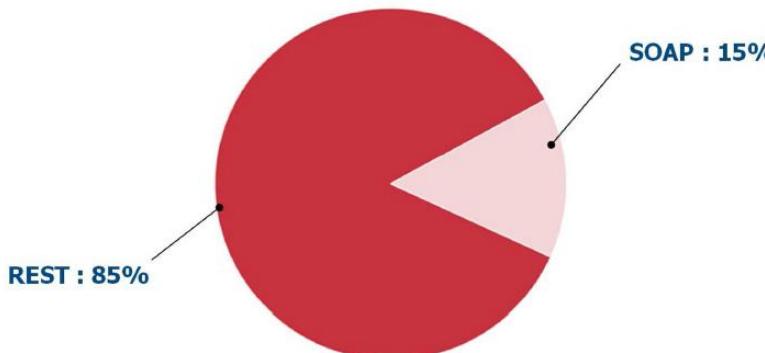


REST : C'est quoi REST

- Les Services Web REST sont utilisés pour développer des architectures orientées ressources
- Différentes nominations disponibles :
 - Architectures Orientées Données (DOA)
 - Architectures Orientées Ressources (ROA)
- Les applications qui respectent les architectures orientées ressources sont nommées RESTful

REST : C'est quoi REST

- **REST (REpresentational State Transfer) ou RESTful** est un style d'architecture pour les systèmes hypermédia distribués,
- Crée par Roy Fielding en 2000 dans le chapitre 5 de sa thèse de doctorat.
- **REST** est un style d'architecture permettant de construire des applications (Web, Intranet, Web Service).
 - Statistiques de l'utilisation de Services Web REST et SOAP chez AMAZON
 - www.oreillynet.com/pub/wlg/3005
- Il s'agit d'un ensemble de meilleures pratiques à respecter entière.
- L'architecture REST utilise le protocole HTTP (comme le font tous les sites web).



REST : caractéristiques

- Les services Web REST sont sans état (stateless)
 - Chaque requête envoyée vers le serveur doit contenir toutes les informations à leur traitement
 - Minimisation des ressources système, pas de session ni d'état
- Les services Web REST fournissent une interface uniforme basée sur les méthodes HTTP
 - GET, POST, PUT, DELETE
- Les architectures orientées REST sont construites à partir de ressources qui sont uniquement identifiées par des URL

REST : caractéristiques

- Dans une architecture orientée REST, les ressources sont manipulées à travers des formats de représentations
 - Une ressource liée à un Bon de Commande est représentée par un document XML
 - La création d'un Bon de commande est réalisée par la combinaison d'une méthode HTTP Post et d'un document XML
- Dans une architecture orientée REST, la communication est obtenue par le transfert de la représentation des ressources
- L'état est maintenu par la représentation d'une ressource
- Par conséquent, le client est responsable de l'état de la ressource

REST : caractéristiques

- **Ressources (Identifiant)**
 - Identifié par une URI
 - Exemple :
<http://mywebsite.com/books/87/comments/1568>
- **Méthodes (Verbes)**
 - Méthodes HTTP : GET, POST, PUT & DELETE
- **Représentation**
 - Information transférée entre le client et le serveur
 - Exemple : XML, JSON

REST : Ressources

- Une ressource est quelque chose qui est identifiable dans un système
 - Personne, Agenda, Collection, Document ...
- Une URI identifie une ressource de manière unique sur le système
- Une ressources peut avoir plusieurs URI et la représentation de la ressource peut évoluer avec le temps
- Il est nécessaire de prendre en compte:
 - La hiérarchie des ressources
 - La sémantique des URL pour les éditer

REST : Ressources

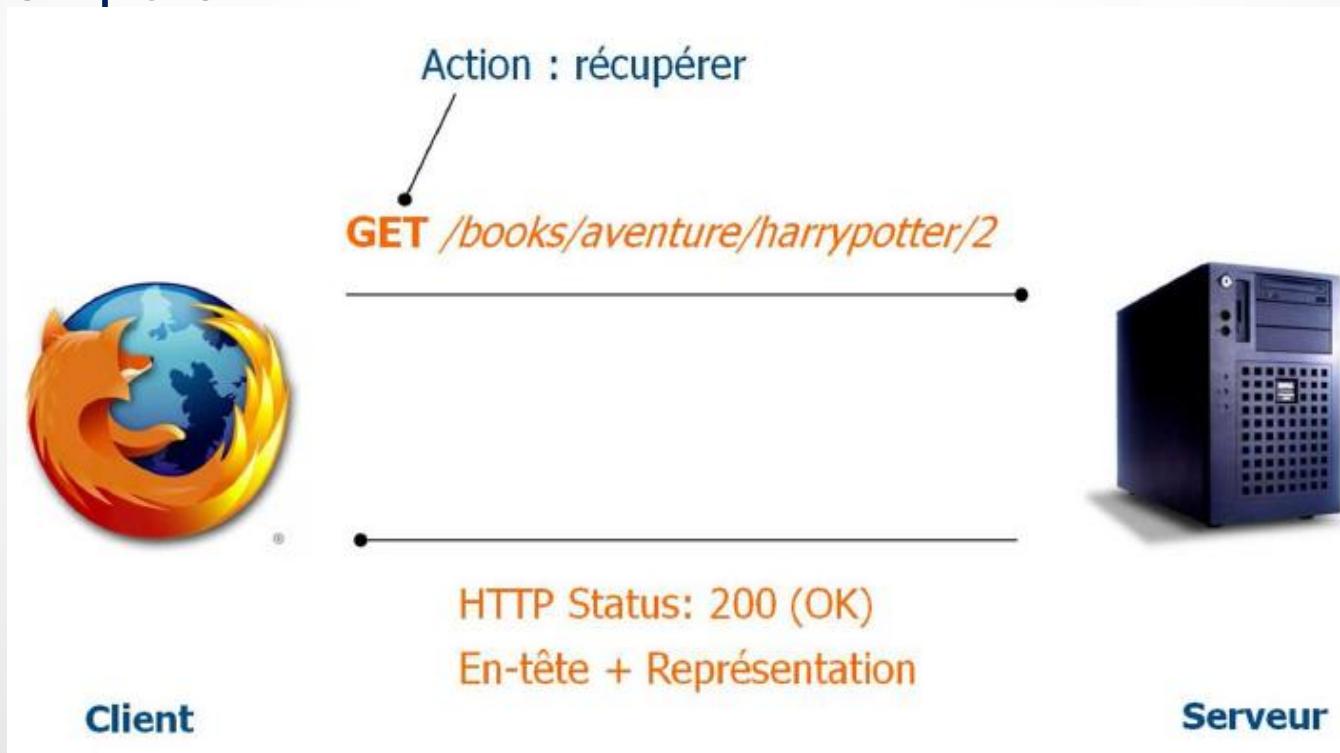
- **Liste des livres**
 - **NOK** : <http://mywebsite.com/book>
 - **OK** : <http://mywebsite.com/books>
- **Filtre et tri sur les livres**
 - **NOK** : <http://mywebsite.com/books/filtre/policier/tri/asc>
 - **OK** : <http://mywebsite.com/books?filtre=policier&tri=asc>
- **Affichage d'un livre**
 - **NOK** : <http://mywebsite.com/book/display/87>
 - **OK** : <http://mywebsite.com/books/87>
- **Tous les commentaires sur un livre**
 - **NOK** : <http://mywebsite.com/books/comments/87>
 - **OK** : <http://mywebsite.com/books/87/comments>
- En construisant correctement les URI, il est possible de les trier, de les hiérarchiser et donc d'améliorer la compréhension du système.
- L'URL suivante peut alors être décomposée logiquement :
 - <http://mywebsite.com/books/87/comments/1568> => **un commentaire pour un livre**
 - <http://mywebsite.com/books/87/comments> => **tous les commentaires pour un livre**
 - <http://mywebsite.com/books/87> => **un livre**
 - <http://mywebsite.com/books> => **tous les livres**

REST : Méthode

- Dans une architecture REST il faut utiliser les verbes HTTP existants plutôt que d'inclure l'opération dans l'URI de la ressource.
- Ainsi, généralement pour une ressource, il y a 4 opérations possibles (CRUD) :
 - Créer (create)
 - Afficher (read)
 - Mettre à jour (update)
 - Supprimer (delete)
- HTTP propose les verbes correspondant :
 - Créer (create) => **POST**
 - Afficher (read) => **GET**
 - Mettre à jour (update) => **PUT**
 - Supprimer (delete) => **DELETE**
- Il est possible d'exprimer des opérations supplémentaires via d'autre méthodes HTTP (**HEAD**, **OPTIONS**)

REST : Méthode GET

- La Méthode Get fournit la représentation de la ressource
 - Idempotent



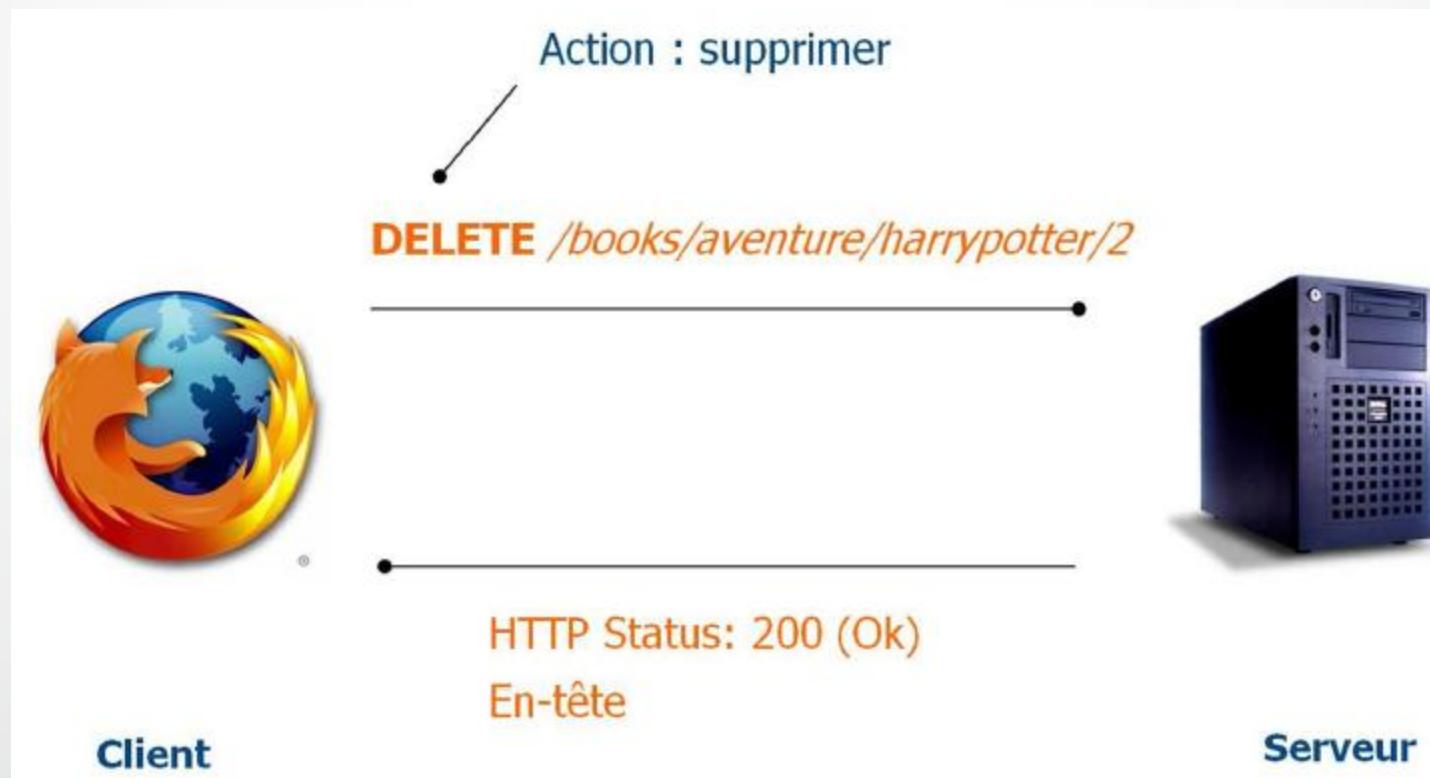
REST : Méthode POST

- Méthode POST crée une ressource
 - Nom idempotente (plusieurs création de la même ressource)



REST : Méthode DELETE

- Méthode DELETE supprime une ressource
 - Idempotent



REST : Méthode PUT

- Méthode PUT met à jour une ressource
 - Idempotent



REST : Méthode

- Exemple d'URL pour une ressource donnée (un livre par exemple) :
- **Créer un livre**
 - NOK : GET <http://mywebsite.com/books/create>
 - OK : POST <http://mywebsite.com/books>
- **Afficher**
 - NOK : GET <http://mywebsite.com/books/display/87>
 - OK : GET <http://mywebsite.com/books/87>
- **Mettre à jour**
 - NOK : POST <http://mywebsite.com/books/editer/87>
 - OK : PUT <http://mywebsite.com/books/87>
- **Supprimer**
 - NOK : GET <http://mywebsite.com/books/87/delete>
 - OK : DELETE <http://mywebsite.com/books/87>

REST : Représentation

- Fournir les données suivant une représentation pour:
 - Le client (GET)
 - Le serveur (PUT et POST)
- Données retournées sous différents formats
 - XML
 - JSON
 - XHTML
 - CSV
- Le format d'entrée (POST) et le format de sortie (GET) d'un service Web d'une ressource peuvent être différents

- Il est important d'avoir à l'esprit que la réponse envoyée n'est pas une ressource, c'est la représentation d'une ressource.
- Ainsi, une ressource peut avoir plusieurs représentations dans des formats divers :**HTML, XML, CSV, JSON, etc.**
- C'est au client de définir quel format de réponse il souhaite recevoir via l'entête **Accept**.

REST : Représentation

➤ Exemples : format JSON et XML

GET https://www.googleapis.com/urlshortener/v1/url?shortUrl=http://goo.gl/fbsS

```
{  
  "kind": "urlshortener#url",  
  "id": "http://goo.gl/fbsS",  
  "longUrl": "http://www.google.com/",  
  "status": "OK"  
}
```

Représentation des données en JSON

GET http://localhost:8080/librarycontentrestwebservice/contentbooks/string

```
<?xml version="1.0"?>  
<details>  
  Ce livre est une introduction sur la vie  
</details>
```

Représentation des données en XML

Exemple : Google URL Shortener

- Google maps API est un service gratuit de cartographie en ligne
 - <http://maps.googleapis.com/maps/api/geocode/>
- Les actions proposées sont les suivantes
 - Rechercher une adresse (GET)
 - Recherche itinéraire (GET)
 - ...
- Service gratuit et proposant une API REST pour le développement
- Pour utiliser l'API, nécessite l'utilisation d'une clé pour un certain nombre d'actions journalières
- Utilisation de SOAPUI pour les tests

<http://services.groupkt.com/country/get/all>

<http://services.groupkt.com/country/get/iso2code/FR>

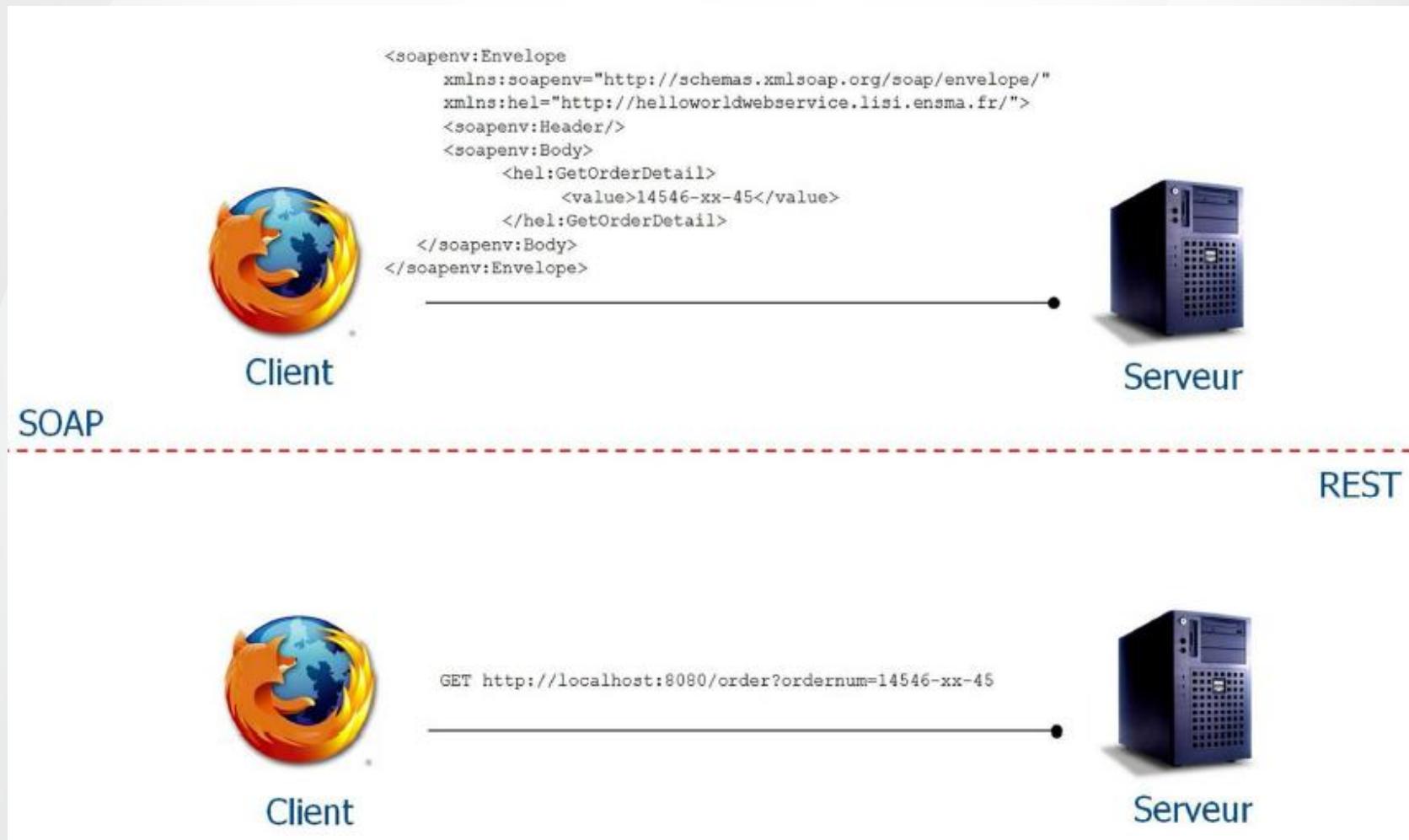
Pratique :

Comment tester

un Webservice

type REST

SOAP VS REST



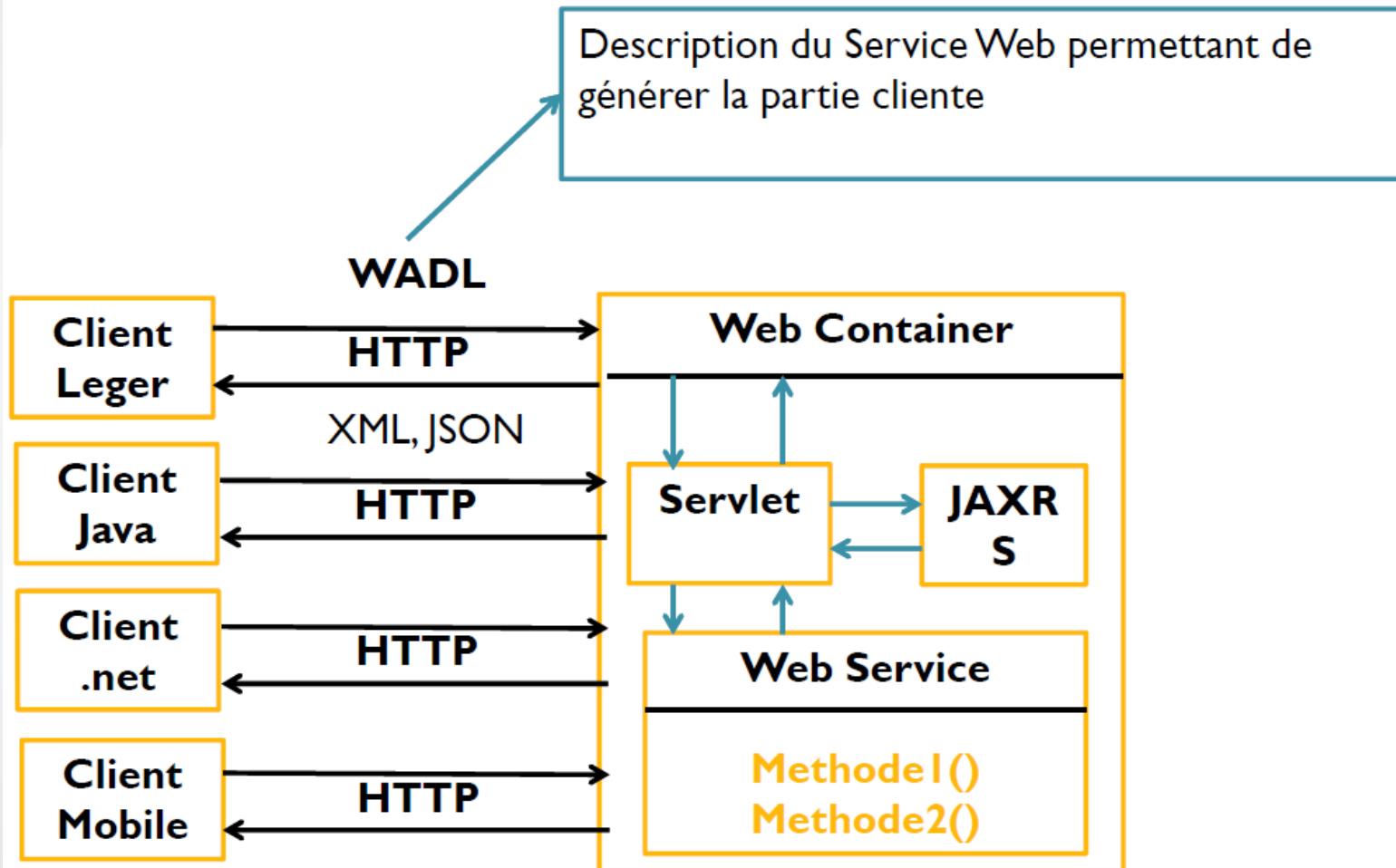
SOAP VS REST

- Les Services Web étendus (SOAP) et les Services Web REST diffèrent par le fait que :
 - Les Services Web étendus reposent sur des standards
 - REST est un style d'architecture
- Services Web étendus (SOAP)
 - Avantages :
 - Standardisé
 - Interopérabilité
 - Sécurité (WS-Security)
 - Outillé
 - Inconvénients
 - Performances (Enveloppe SOAP supplémentaire)
 - Complexité, lourdeur
 - Cible l'appel de service

SOAP VS REST

- Services Web REST
 - Avantages
 - Simplicité de mise en œuvre
 - Lisibilité par l'humain
 - Evolutivité
 - Repose sur les principes du Web
 - Représentations multiples
 - Inconvénients
 - Sécurité restreinte par l'emploi des méthodes HTTP
 - Cible uniquement l'appel de ressource

WADL : Web Application Description Language



WADL

- WADL est un fichier XML qui permet de faire la description des services web d'une application basée sur REST.
- Le WADL est généré automatiquement par le conteneur REST.
- Les types de données structurées échangées via ce web service sont décrites par un schéma XML lié au WADL.
- L'objectif est de pouvoir générer automatiquement les APIs clientes d'accès aux services REST
- Remarques
 - Peu d'outils exploitent la description WADL
 - Apparu bien plus tard

WADL

➤ Exemple :

```
<application>
<doc jersey:generatedBy="Jersey: 1.4 09/11/2010 10:30 PM"/>
<resources base="http://localhost:8088/librarycontentrestwebservice/">
    <resource path="/contentbooks">
        <resource path="uribuilder2">
            <method name="POST" id="createURIBooks">
                <request>
                    <representation mediaType="application/xml"/>
                </request>
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        <resource path="uribuilder1">
            <method name="POST" id="createBooksFromURI">
                <request>
                    <representation mediaType="application/xml"/>
                </request>
                <response>
                    <representation mediaType="*/*"/>
                </response>
            </method>
        </resource>
        ...
    </resource>
</resources>
</application>
```

Outils

- Des outils pour appeler des services REST
 - CURL : curl.haxx.se
 - Poster (plugin Firefox)
 - **SOAPUI**
- Des plateformes pour développer (serveur) et appeler (client) des services REST
 - JAX-RS (Jersey) pour la plateforme Java
 - .NET
 - PHP
 - Python

Sommaire

- **Chapitre 1 : INTRODUCTION**
- **Chapitre 2 : Technologies des services Web**
 - Exemple pratique de création et de déploiement d'un Web service SOAP et de son client
- **Chapitre 3 : Service Web REST**
- **Chapitre 4 : Développement de service web REST avec JAX-RS**
 - Etude de cas : *Développement d'un service RESTful retournant un flux JSON. + Invocation du service et parsing du résultat en Java.*
- **Chapitre 5 : la gestion des Persistances dans un WEB SERVICE REST**
 - Etude de cas : Client : JAX-RS, JaxB, JAVA
- **Chapitre 6 : Sécurité des Web services SOAP VS REST**

Chapitre 4

Développement

d'un service Web

avec JAX-RS

Objectif du chapitre

- Généralité JAX-RS
- Premier service Web JAX-RS
- Rappels http
- Développement Serveur
 - Chemin de ressource @Path
 - Paramètres des requêtes
 - Gestion du contenu, Response et UriBuilder
 - Déploiement
- Développement Client
- Outils

JAX-RS

Généralités JAX-RS : la spécification

- JAX-RS est l'acronyme **Java API for RESTful Web Services**
- Version courante de la spécification est la 2.0
- Depuis la version 1.1, JAX-RS fait partie intégrante de la spécification Java EE 6 au niveau de la pile Service Web
- Cette spécification décrit uniquement la mise en œuvre de services Web REST côté serveur
- Le développement des Services Web REST repose sur l'utilisation de classes Java et d'annotations

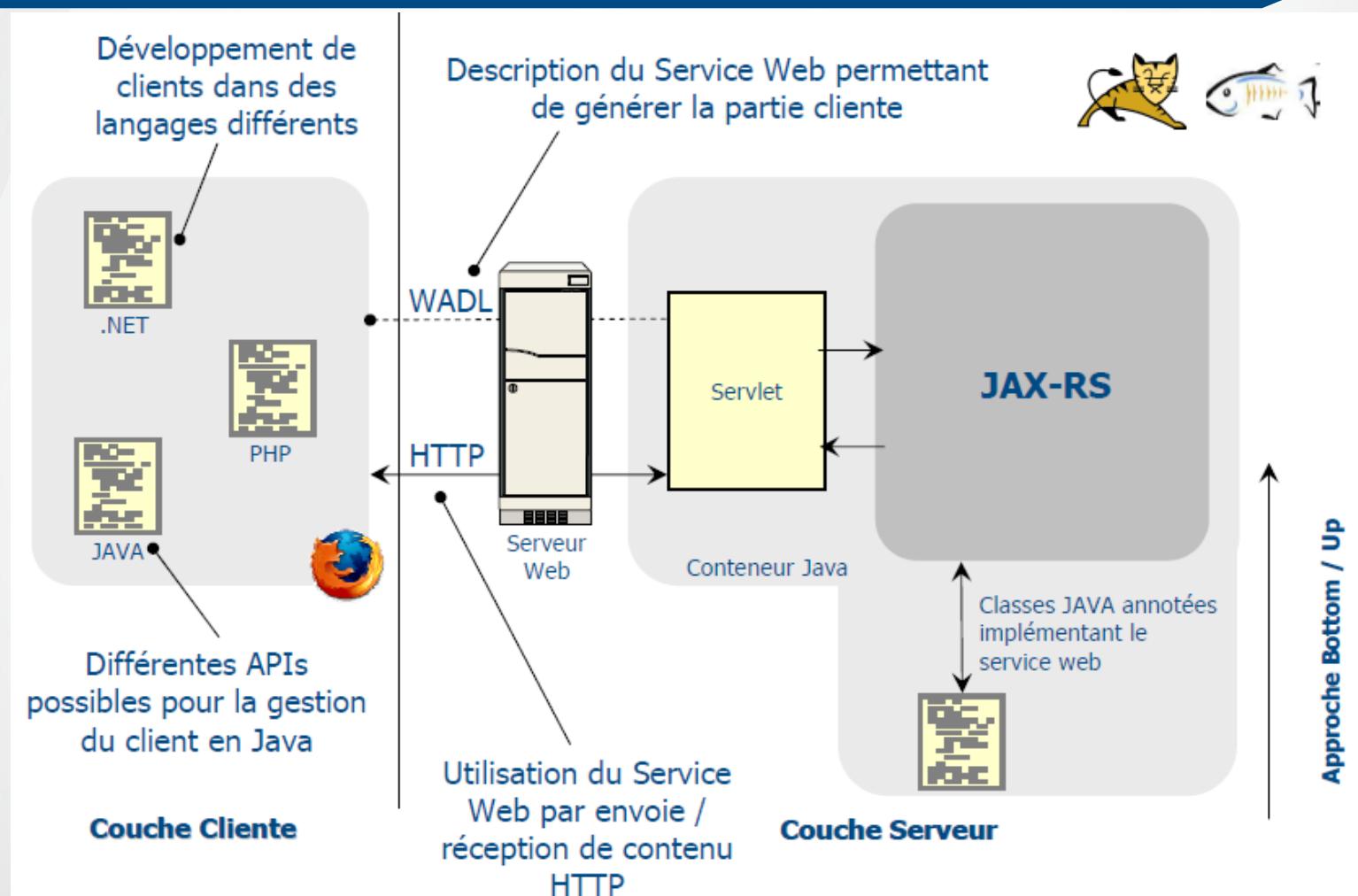
Généralités JAX-RS : la spécification

- Différentes implémentations de la spécification JAX-RS sont disponibles
- JERSEY : Implémentation de référence fournie par Oracle
 - Site projet : jersey.java.net
- CXF : fournie par Apache (fusion entre Xfire et Celtix)
 - Site projet : cxf.apache.org
- RESTEasy : fournie par Jboss
 - Site projet : www.jboss.org/resteasy
- RESTlet : un des premiers framework implémentant REST pour Java
 - Site projet : www.restlet.org

Généralités JAX-RS : la spécification

- Comme la spécification JAX-RS ne décrit pas la couche cliente, chaque implémentation fournit une API spécifique
- Dans la suite du support nous utiliserons l'implémentation de référence **JERSEY**
 - Version 1.18 respectant la spécification JAX-RS 1.1
 - Intégré dans GlassFish et l'implémentation Java EE 6
 - Supporté dans Eclipse
 - Description Maven
 - groupId : com.sun.jersey
 - artifactId : jersey-server
 - Version : 1.18

Généralités JAX-RS : fonctionnement



Généralités JAX-RS : fonctionnement

- Le développement de Services Web avec JAX-RS est basé sur des POJO (Plain Old Java Object) en utilisant des annotations spécifiques à JAX-RS
- Pas de description requise dans des fichiers de configuration
- Seule la configuration de la Servlet « JAX-RS » est requise pour réaliser le pont entre les requêtes HTTP et les classes Java annotées
- Un Service Web REST est déployé dans une application Web

Généralités JAX-RS : fonctionnement

- Contrairement aux Services Web entendus il n'y a pas de possibilité de développer un service REST à partir du fichier de description WADL
- Seule l'approche Botton / Up est disponible
 - Créer et annoter un POJO
 - Compiler, Déployer et Tester
 - Possibilité d'accéder au document WADL
- Le fichier de description WADL est généré automatiquement par JAX-RS (exemple :
 - <http://host/context/application.wadl>)

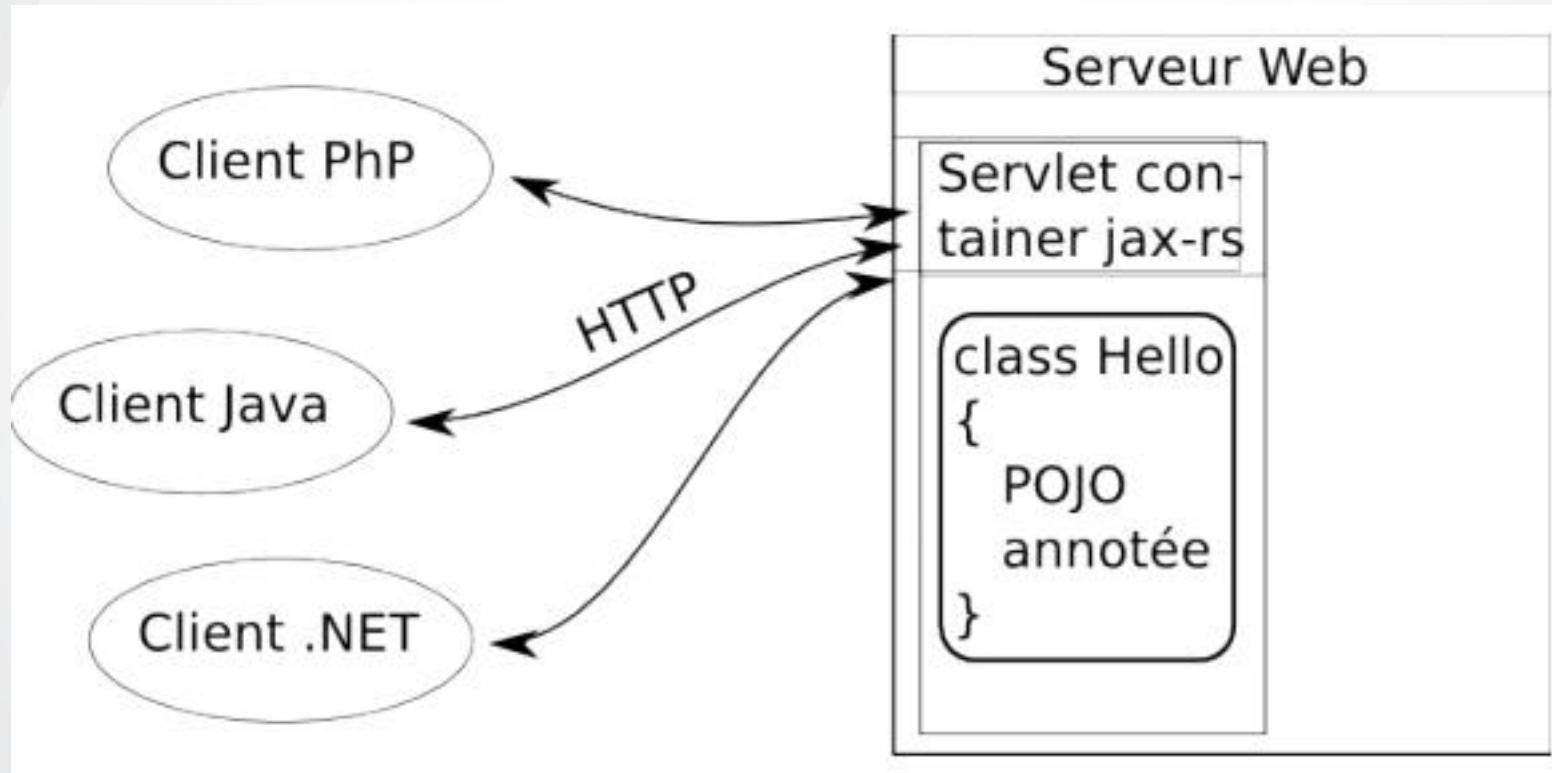
Objectif du chapitre

- Généralité JAX-RS
- **Premier service Web JAX-RS**
- Rappels http
- Développement Serveur
 - Chemin de ressource @Path
 - Paramètres des requêtes
 - Gestion du contenu, Response et UriBuilder
 - Déploiement
- Développement Client
- Outils

Premier Service Web Jax-RS

Exemple ServiceWeb Jax-RS

➤ Service Hello



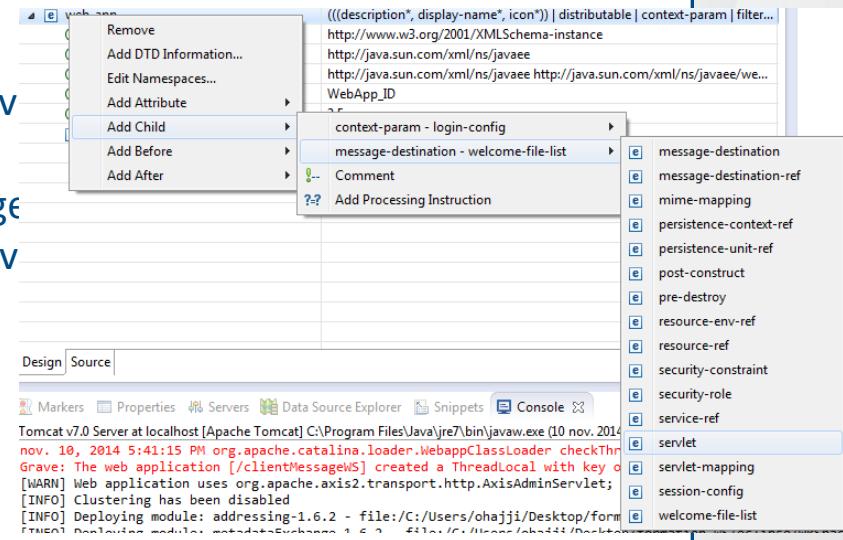
Exemple ServiceWeb Jax-RS

- Fichier>New>Dynamic Web Project
 - ProjectName : FirstRestWebService
 - Target RunTime : Tomcat 7
 - Dynamic Web Module version : 2.5
- Next > Next > Finish
- ➔ vérifiez si le fichier web.xml est créé sous WebContent>WEB-INF
- Copiez manuellement les lib. de Jersey vers le projet web (dans la suite du cour en utilisera Maven pour récupérer automatiquement les librairie)
- ➔ maintenant la structure du projet est prête

Exemple ServiceWeb Jax-RS

- Dans web.xml
 - Virez tous le Tag <welcome-file-list>
 - En mode design renseigner les informations suivantes :

```
<servlet>
  <display-name>Rest Servlet</display-name>
  <servlet-name>RestServlet</servlet-name>
  <servlet-class>com.sun.jersey.spi.container.servlet.Serv
<init-param>
  <param-name>com.sun.jersey.config.property.package
  <param-value>com.wsformation.ressources</param-v
</init-param>
</servlet>
<servlet-mapping>
  <servlet-name>RestServlet</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```



Exemple ServiceWeb Jax-RS

- Vérifiez que le contexte est bien renseigné dans Web Project Setting (en cliquant sur le projet puis properties)
- C'est grâce à ce contexte qu'on va appeler l'url en tapant :

<http://localhost:8080/FirstRestWebService/rest/>

- Passant à la création du WebService
- Créez une nouvelle classe comme suit :
 - Package : com.wsformation.ressources
 - Nom : Hello
 - Décochez inherit Abstract Class

Exemple ServiceWeb Jax-RS

```
package com.wsformation.ressources;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class Hello {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello(){
        return "Hello, World";
    }

    @GET
    @Produces(MediaType.TEXT_XML)
    public String sayPlainXmlHello(){
        return "<?xml version=\"1.0\"?>" + "<hello>Hello, World</hello>";
    }
}
```

Définition d'un chemin de ressource pour associer une ressource *hello* à une URI

Le type MIME de la réponse est de type *text/plain*

Lecture de la ressource *HelloWorld* via une requête HTTP de type GET

➔ Vous pouvez tester maintenant en tapant :

<http://localhost:8080/FirstRestWebService/rest/hello> ou
<http://127.0.0.1:8080/FirstRestWebService/rest/hello>

Cliquez sur runOnServer au niveau projet pour tester

Les type Mimes réponse : JSON

- **JSON** (JavaScript Object Notation – Notation Objet issue de JavaScript) est un format léger d'échange de données.
- Il est facile à lire ou à écrire pour des humains. Il est aisément analysable ou générable par des machines.
- JSON est un format texte complètement indépendant de tout langage, mais les conventions qu'il utilise seront familières à tout programmeur habitué aux langages descendant du C, comme par exemple : C lui-même, C++, C#, Java, JavaScript, Perl, Python et bien d'autres.
- Ces propriétés font de JSON un langage d'échange de données idéal
- Utilisé pour les services exposés directement vers le navigateur (Mashup)
- Structure :
 - Paires clé / valeurs
 - Les valeurs peuvent être des listes de paires, des tableaux de paires, des objets, chaînes de caractères...

XML VS JSON

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<comptes>
    <compte type="courant">
        <code>1</code>
        <solde>4300</solde>
        <dateCreation>2012-11-11</dateCreation>
    </compte>
    <compte type="epargne">
        <code>2</code>
        <solde>96000</solde>
        <dateCreation>2012-12-11</dateCreation>
    </compte>
</comptes>
```

JSON

```
[
    {type:"courant" , code:1, solde:4300.50,dateCreation:"2012-11-11"},
    {type:"epargne" , code:1, solde:4300.00,dateCreation:"2012-11-11"}
]
```

XML VS JSON

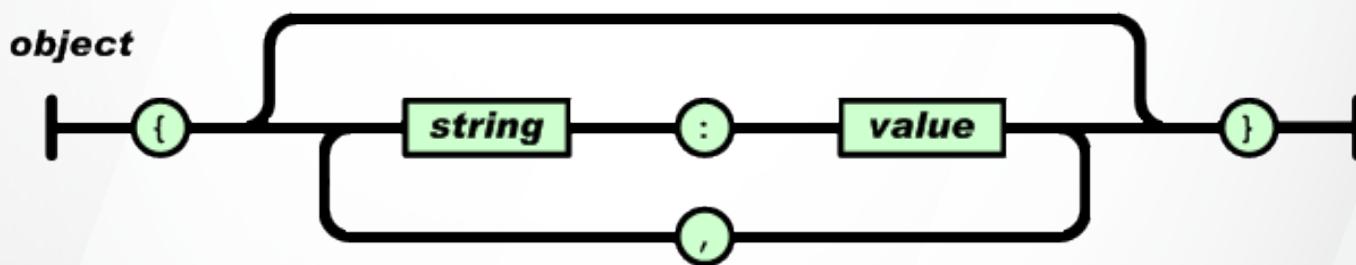
- XML est standard
- Manipulation aisée
- Verbeux
- JSON est créé pour les navigateurs web
- Interprété nativement, donc performant
- Supporté par la plupart des langages moyennant une API

Les structures de données JSON

- En JSON, les structures de données prennent les formes suivantes :
 - Un **objet**, qui est un ensemble de couples nom/valeur non ordonnés. Un objet commence par { (accolade gauche) et se termine par } (accolade droite). Chaque nom est suivi de : (deux points) et les couples nom/valeur sont séparés par , (virgule).
 - Un **tableau** est une collection de valeurs ordonnées. Un tableau commence par [(crochet gauche) et se termine par] (crochet droit). Les valeurs sont séparées par , (virgule).
 - Une **valeur** peut être soit une *chaîne de caractères* entre guillemets, soit un *nombre*, soit true ou false ou null, soit un *objet* soit un *tableau*. Ces structures peuvent être imbriquées.
 - Une **chaîne de caractères** est une suite de zéro ou plus caractères Unicode, entre guillemets, et utilisant les échappements avec antislash. Un caractère est représenté par une chaîne d'un seul caractère.

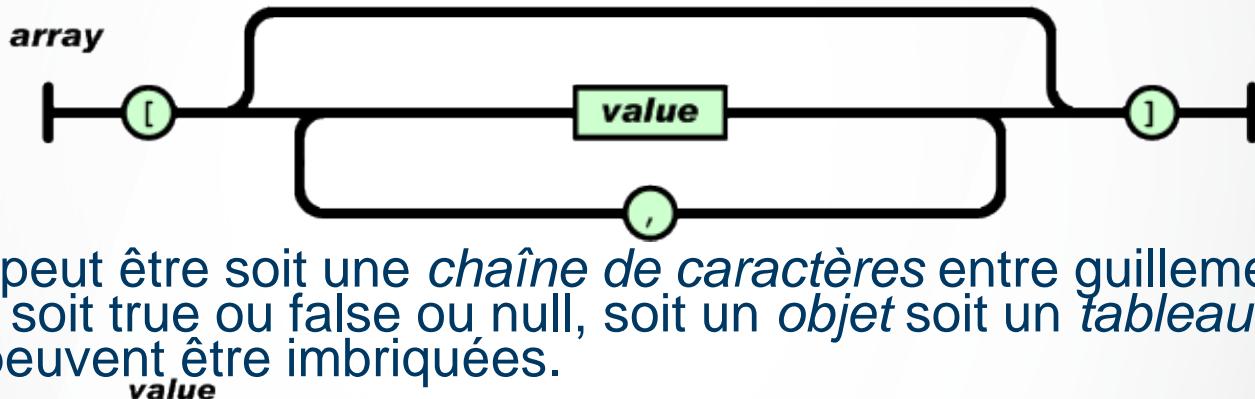
Les structures de données JSON

- Un *objet*, qui est un ensemble de couples nom/valeur non ordonnés.
 - Un objet commence par { (accolade gauche) et se termine par } (accolade droite).
 - Chaque nom est suivi de : (deux-points) et les couples nom/valeur sont séparés par , (virgule).

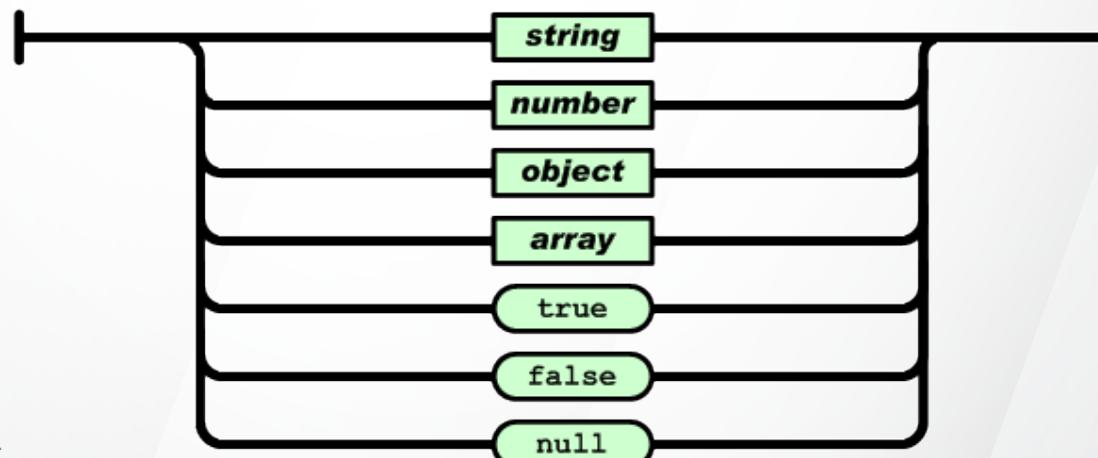


Les structures de données JSON

- Un *tableau* est une collection de valeurs ordonnées.
 - Un tableau commence par [(crochet gauche) et se termine par] (crochet droit).
 - Les valeurs sont séparées par , (virgule)

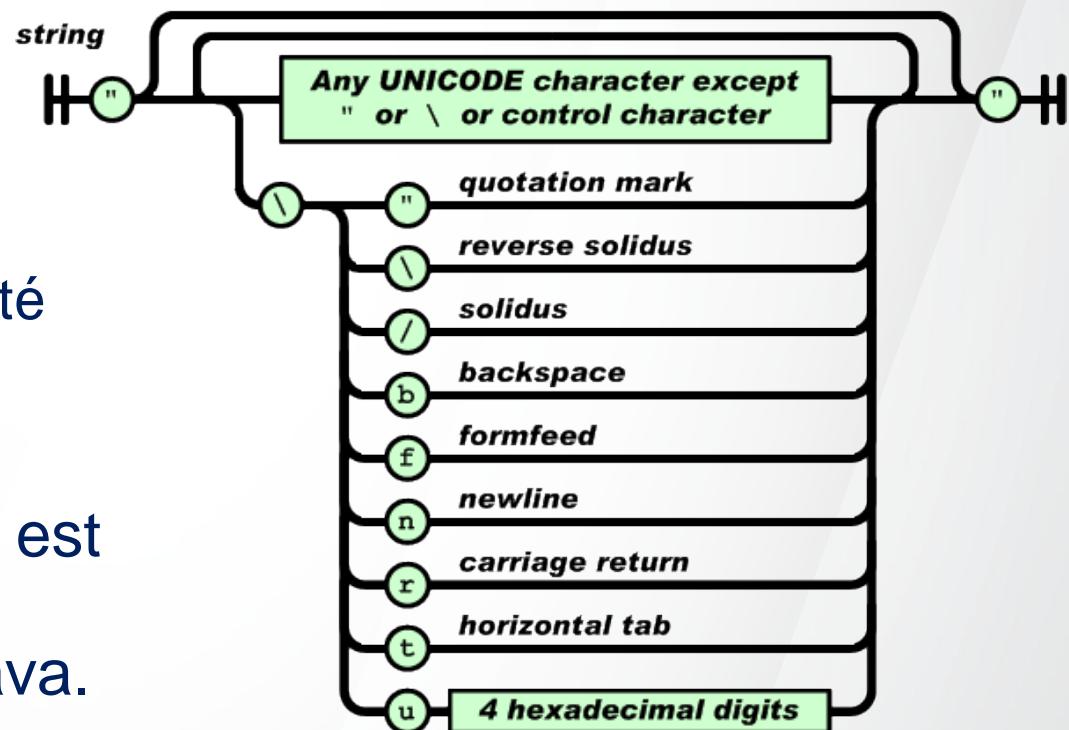


- Une *valeur* peut être soit une *chaîne de caractères* entre guillemets, soit un *nombre*, soit true ou false ou null, soit un *objet* soit un *tableau*. Ces structures peuvent être imbriquées.



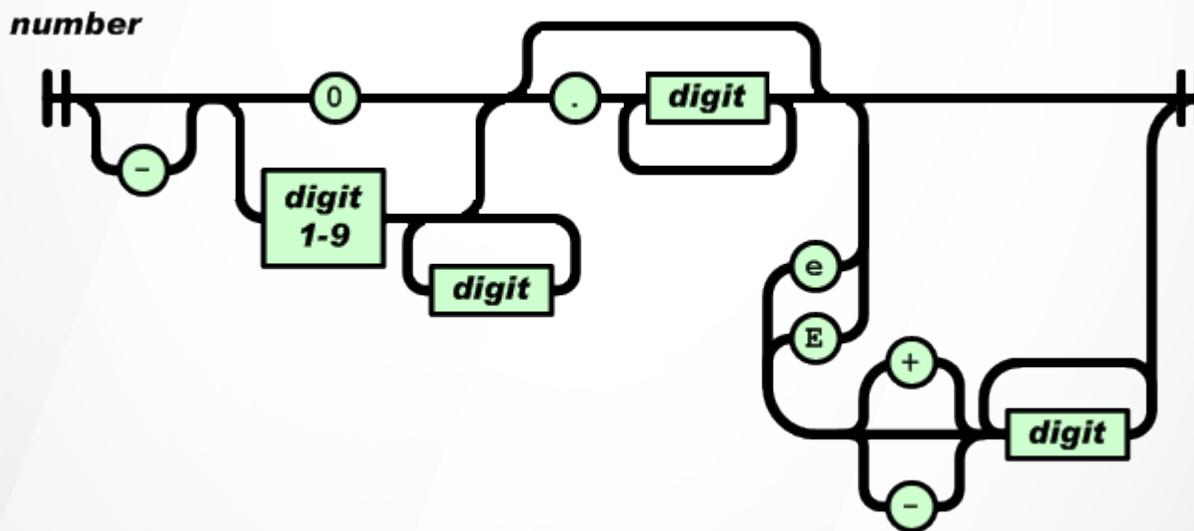
Les structures de données JSON

- Une *chaîne de caractères* est une suite de zéro ou plus caractères Unicode, entre guillemets, et utilisant les échappements avec antislash.
 - Un caractère est représenté par une chaîne d'un seul caractère.
- Une *chaîne de caractères* est très proche de ses équivalents en C ou en Java.



Les structures de données JSON

- Un *nombre* est très proche de ceux qu'on peut rencontrer en C ou en Java, sauf que les formats octal et hexadécimal ne sont pas utilisés.



Les structures de données JSON

➤ JSON

```
{ "menu": "Fichier",
  "commandes": [
    { "titre": "Nouveau", "action": "CreateDoc" },
    { "titre": "Ouvrir", "action": "OpenDoc" },
    { "titre": "Fermer", "action": "CloseDoc" }
  ]
}
```

➤ XML

```
<?xml version="1.0" ?>
<racine>
  <menu>Fichier</menu>
  <commandes>
    <item> <titre>Nouveau</titre> <action>CreateDoc</action></item>
    <item> <titre>Ouvrir</titre> <action>OpenDoc</action> </item>
    <item> <titre>Fermer</titre> <action>CloseDoc</action> </item>
  </commandes>
</racine>
```

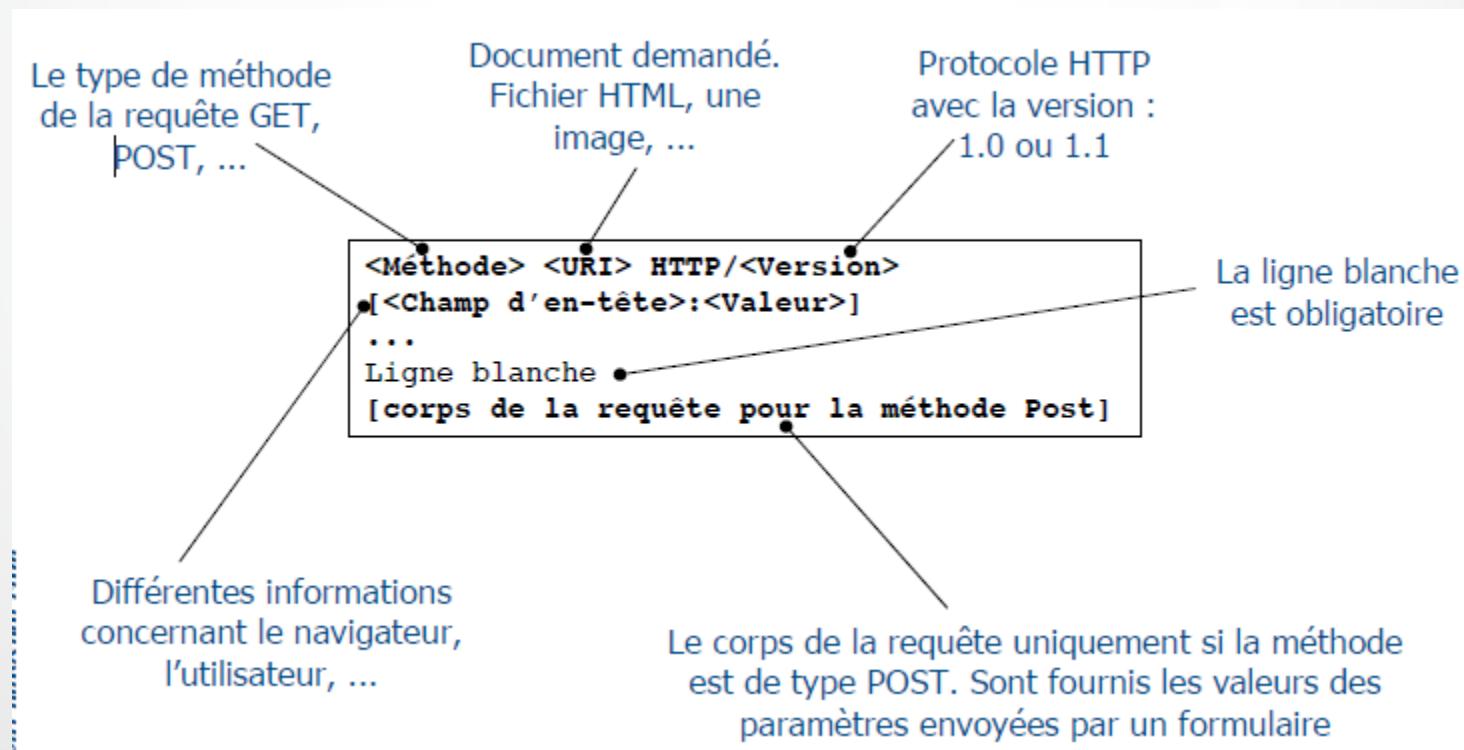
Objectif du chapitre

- Généralité JAX-RS
- Premier service Web JAX-RS
- **Rappels http**
- Développement Serveur
 - Chemin de ressource @Path
 - Paramètres des requêtes
 - Gestion du contenu, Response et UriBuilder
 - Déploiement
- Développement Client
- Outils

Rappels HTTP

Protocole HTTP : requête

- Requête envoyée par le client (Navigateur) au serveur



Protocole HTTP : en-tête de la requête

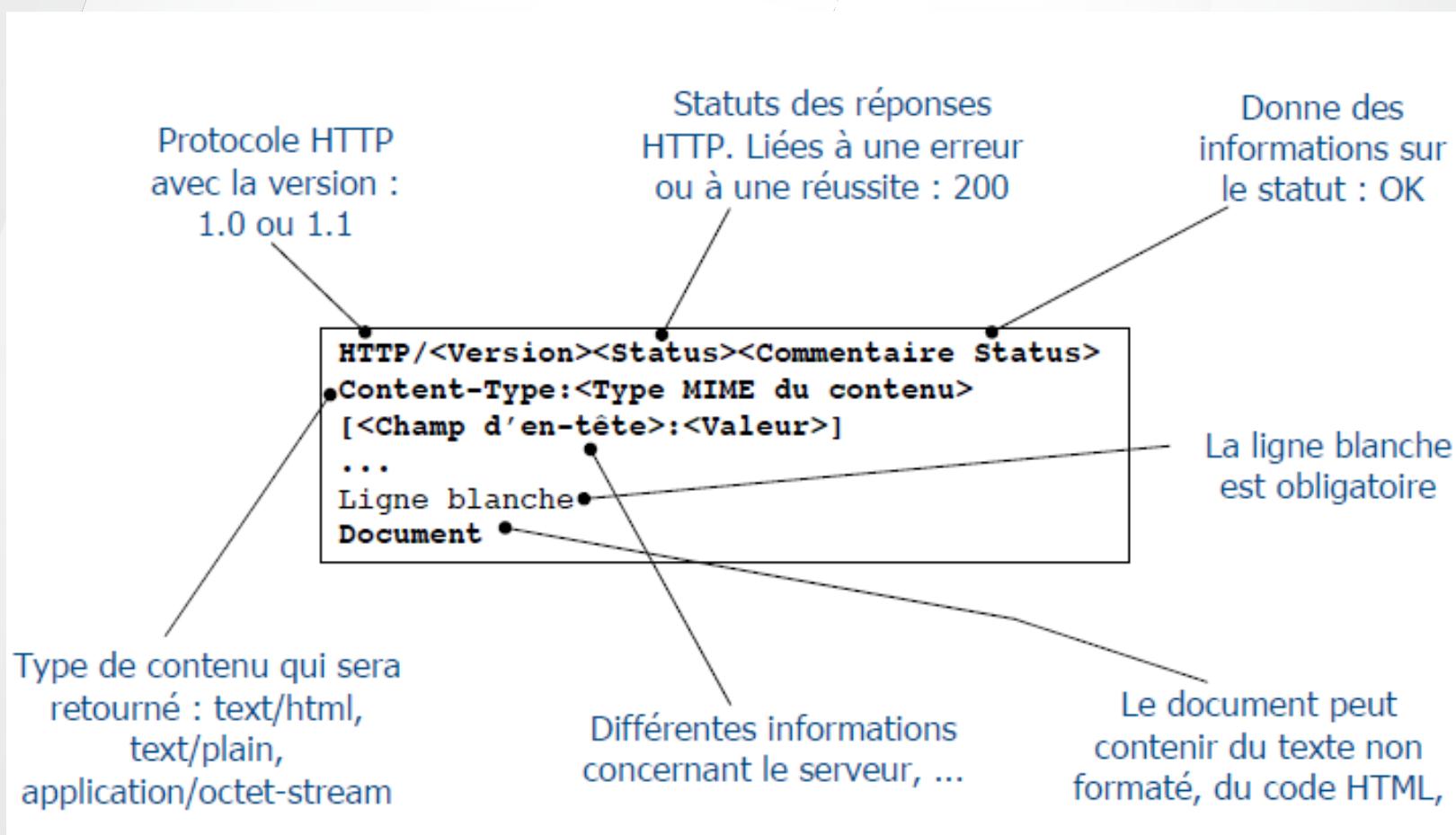
- Correspond aux formats de documents et aux paramètres pour le serveur
 - Accept = type MIME accepté par le client (text/html, text/plain ...)
 - Accept-Encoding = codage accepté (compress, x-gzip, x-zip)
 - Accept-Charset = jeu de caractère préféré du client
 - Accept-Language = liste de langue (fr, en, de)
 - Autorisation = type d'autorisation
 - BASIC nom:mots de passe (base 64)
 - Transmis en clair, facile à décrypter
 - Cookie = cookie retourné
 - From = Adresse email de l'utilisateur

Protocole HTTP : Type de méthode

- Lorsqu'un client se connecte à un serveur et envoie une requête, cette requête peut être de plusieurs types, appelés **méthode**
- **Type GET**
 - Pour extraire des informations (document, graphique ...)
 - Intègre les données de formatage à l'URL (chaîne d'interrogation)
- **Type POST**
 - Pour poster des informations secrètes, des données graphiques, ...
 - Transmis dans le corps de la requête

```
<Méthode> <URI> HTTP/<Version>
[<Champ d'en-tête>:<Valeur>]
...
Ligne blanche
[corps de la requête pour la méthode Post]
```

Protocole HTTP : réponse



Objectif du chapitre

- Généralité JAX-RS
- Premier service Web JAX-RS
- Rappels http
- **Développement Serveur**
 - Chemin de ressource @Path
 - Paramètres des requêtes
 - Gestion du contenu, Response et UriBuilder
 - Déploiement
- Développement Client
- Outils

Développement Serveur

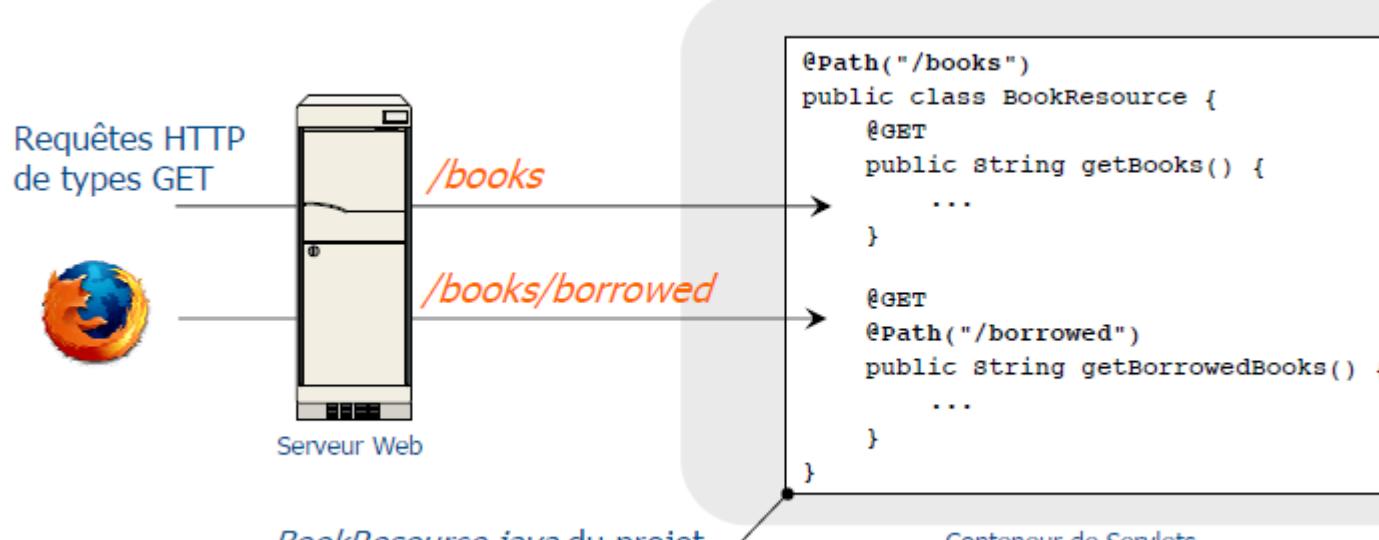
Annotations JAX-RS : @Path

- Une classe Java doit être annotée par **@path** pour qu'elle puisse être traitée par des requêtes HTTP
- L'annotation **@path** sur une classe définit des ressources appelées racines (Root Resource Class)
- La valeur donnée à **@path** correspond à une expression URI relative au contexte de l'application Web



Annotations JAX-RS : @Path

- L'annotation **@path** peut également annoter des méthodes de la classe (facultatif)
- L'URI résultante est la concaténation de l'expression du **@path** de la classe avec l'expression du **@path** de la méthode
- Exemple :



@Path : Template Parameters

- La valeur définie dans @path ne se limite pas seulement aux expressions constantes
- Possibilité de définir des expressions plus complexes appelées **Template Paramètres**
- Pour distinguer une expression complexe dans la valeur du @path, son contenu est délimité par { ... }
- Possibilité également de mixer dans la valeur de @path des expressions constantes et des expressions complexes
- Les Template Paramètres peuvent également utiliser des expressions régulières

@Path : Template Parameters

- Exemple : récupérer un livre par son identifiant

```
    @Path("/books")
    public class BookResource {
        ...
        @GET
        @Path("{id}")
        public String getBookById(@PathParam("id") int id) {
            return "Java For Life " + id;
        }
        ...
        @GET
        @Path("name-{name}-editor-{editor}")
        public String getBookByNameAndEditor(@PathParam("name") String name,
                                            @PathParam("editor") String editor)
            return "Starcraft 2 for Dummies (Name:" + name + " - Editor:" + editor + ")";
    }
```

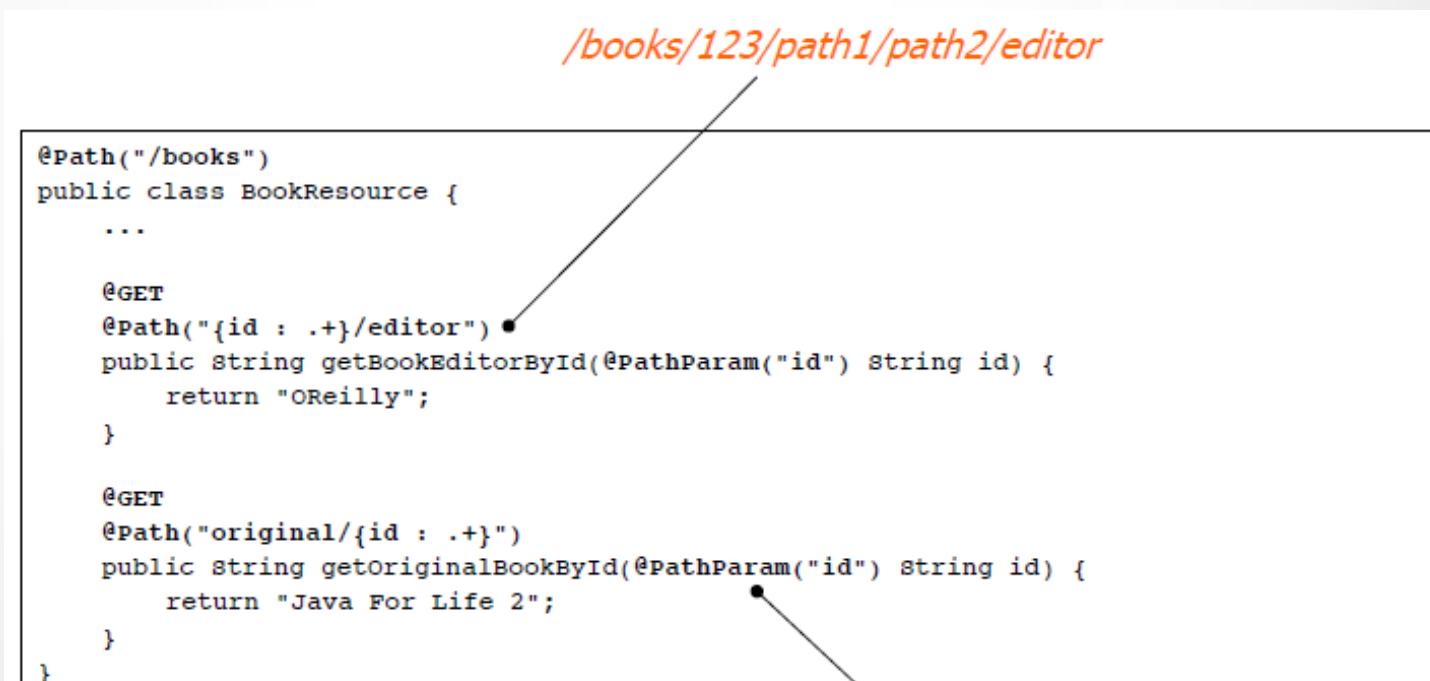
The diagram illustrates the mapping of Java annotations to URL templates. Two arrows point from specific annotations in the code to their corresponding URL patterns. One arrow points from the `@Path("{id}")` annotation to the URL `/books/123`. Another arrow points from the `@Path("name-{name}-editor-{editor}")` annotation to the URL `/books/name-sc2-editor-oreilly`.

BookResource.java du projet

LibraryRestWebService

@Path : Template Parameters

- Exemple (bis): Récupérer un livre par son identifiant



BookResource.java du projet
LibraryRestWebService

Méthodes HTTP :@GET, @POST, @PUT, @DELETE

- L'annotation des méthodes Java permet de traiter des requêtes HTTP suivant le type de méthode (GET, POST, ...)
- Les annotations disponibles par JAX-RS sont les suivantes **@GET, @POST, @PUT, @DELETE et @HEAD**
- Ces annotations ne sont utilisables que sur des méthodes Java
- Le nom des méthodes Java n'a pas d'importance puisque c'est l'annotation employée qui précise où se fera le traitement

Méthodes HTTP :@GET, @POST, @PUT, @DELETE

- La spécification JAX-RS, n'impose pas de respecter les conventions définies par le style REST
- Possibilité d'utiliser une requête HTTP de type GET pour effectuer une suppression d'une ressource
- Des opérations CRUD sur des ressources sont réalisées au travers des méthodes HTTP
- Généralement :
 - GET est utilisée pour consulter une ressource
 - POST est utilisée pour Ajouter une nouvelle ressource
 - PUT est utilisée pour mettre à jour une ressource
 - DELETE est utilisée pour supprimer une ressource

Méthodes HTTP :@GET, @POST, @PUT, @DELETE

➤ Exemple : CRUD sur la ressource livre

```
@Path("/books")
public class BookResource {
    @GET
    public String getBooks() {
        return "Cuisine et moi / JavaEE 18";
    }
    @POST
    public String createBook(String livre) {
        return livre;
    }
    @GET
    @Path("{id}")
    public String getBookById(@PathParam("id") int id) {
        return "Java For Life " + id;
    }
    @PUT
    @Path("{id}")
    public void updateBookById(@PathParam("id") int id) {
        ...
    }
    @DELETE
    @Path("{id}")
    public void deleteBookById(@PathParam("id") int id) {
        ...
    }
}
```

Récupère la liste de tous les livres

Créer un nouveau livre

Récupère un livre

Mise à jour d'un livre

Effacer un livre

*BookResource.java du projet
LibraryRestWebService*

Paramètres de requêtes

- JAX-RS fournit des annotations pour extraire des paramètres d'une requête
- Elles sont utilisées sur les paramètres des méthodes des ressources pour réaliser l'injection du contenu
- Liste des différentes annotations disponibles :
 - @PathParam : extraire les valeurs des Template Parameters
 - @QueryParam : extraire les valeurs des paramètres de requête
 - @FormParam : extraire les valeurs des paramètres de formulaire
 - @HeaderParam : extraire les paramètres de l'entête
 - @CookieParam : extraire les paramètres des cookies
 - @Context : extraire les informations liées aux ressources de contexte
- Une valeur par défaut peut être spécifiée en utilisant l'annotation @DefaultValue

L'annotation @PathParam

- L'annotation @PathParam est utilisée pour extraire les valeurs des paramètres contenus dans les **Template Parameters**
- Exemple :

```
@Path("/books")
public class BookResource {
    ...
    @GET
    @Path("{id}")
    public String getBookById(@PathParam("id") int id) {
        return "Java For Life " + id;
    }
    ...
    @GET
    @Path("name-{name}-editor-{editor}")
    public String getBookByNameAndEditor(@PathParam("name") String name,
                                        @PathParam("editor") String editor)
        return "Starcraft 2 for Dummies (Name:" + name + " - Editor:" + editor + ")";
}
}

BookResource.java du projet LibraryRestWebService
```

/books/123/path1/path2/editor

Injecte les valeurs dans les paramètres de la méthode

/books/name-sc2-editor-oreilly

L'annotation @queryparam

- L'annotation `@QueryParam` est utilisée pour extraire les valeurs des paramètres contenus d'une requête quelque soit son type de méthode HTTP
- Exemple :

/books/queryparameters?name=harry&isbn=1-111111-11&isExtended=true

```
@Path("/books")
public class BookResource {
    ...
    @GET
    @Path("queryparameters")
    public String getQueryParameterBook(
        @DefaultValue("all") @QueryParam("name") String name,
        @DefaultValue("-????????-?") @QueryParam("isbn") String isbn,
        @DefaultValue("false") @QueryParam("isExtended") boolean isExtented) {

        return name + " " + isbn + " " + isExtented;
    }
}
```

Injection de valeurs par défaut si les valeurs des paramètres ne sont pas fournies

BookResource.java du projet **LibraryRestWebService**

Paramètres de requêtes : @FormParam

- L'annotation `@FormParam` est utilisée pour extraire les valeurs des paramètres contenus dans un formulaire
- Le type de contenu doit être :
 - `application/x-www-form-urlencoded`
- Cette annotation est très utile pour extraire les informations d'une requête POST d'un formulaire HTML

```
@Path("/books")
public class BookResource {
    ...
    @POST
    @Path("createfromform")
    @Consumes("application/x-www-form-urlencoded")
    public String createBookFromForm(@FormParam("name") String name) {
        System.out.println("BookResource.createBookFromForm()");
        return name;
    }
}
```

*BookResource.java du projet
LibraryRestWebService*

Paramètres de requêtes : @Headerparam

- L'annotation `@HeaderParam` est utilisée pour extraire les valeurs des paramètres contenues dans l'entête d'une requête
- Exemple :

```
@GET  
public String getQueryParameterBook(  
    @DefaultValue("all") @QueryParam("name") String name,  
    @DefaultValue("true") @QueryParam("isReady") boolean isReady,  
    @HeaderParam("User-Agent") String userAgent){  
    return "Name="+name+" ready="+isReady+" Agent="+userAgent;  
}
```



Name=A ready=true Agent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2)
AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31

Paramètres de requêtes : @Context

- L'annotation **@Context** permet d'injecter des objets liés au contexte de l'application
- Les types d'objets supportés sont les suivants :
 - **UriInfo** : informations liées aux URLs
 - **Request** : informations liées au traitement de la requête
 - **HttpHeaders** : informations liées à l'entête
 - **SecurityContext** : informations liées à la sécurité
- Certains de ces objets permettent d'obtenir les mêmes informations que les précédentes annotations liées aux paramètres

Paramètres de requêtes : @Context / UriInfo

- Un objet de type UriInfo permet d'extraire les informations « brutes » d'une requête HTTP
- Les principales méthodes sont les suivantes :
 - **String getPath()** : chemin relatif de la requête
 - **MultivaluedMap<String, String> getPathParameters()** : valeurs des paramètres de la requête contenus dans Template Parameters
 - **MultivaluedMap<String, String> getQueryParameters()** : valeurs des paramètres de la requête
 - **URI getBaseUri()** : chemin de l'application
 - **URI getAbsolutePath()** : chemin absolu (base + chemins)
 - **URI getRequestUri()** : chemin absolu incluant les paramètres

Paramètres de requêtes: @Context / UriInfo

```
@GET  
@Path("/uriInfo/{name}")  
public String uriInfo(@Context UriInfo uriInfo, @PathParam("name")String name){  
    System.out.println("getPath() :" +uriInfo.getPath());  
    System.out.println("getAbsolutePath(): " +uriInfo.getAbsolutePath());  
    System.out.println("getBaseUri(): " +uriInfo.getBaseUri());  
    System.out.println("getRequestUri(): " +uriInfo.getRequestUri());  
    System.out.println("getPathSegments():");  
    List<PathSegment> pathSegments=uriInfo.getPathSegments();  
    for(PathSegment ps:pathSegments)  
        System.out.println(ps.getPath());  
    System.out.println("getPathParameters()");  
    MultivaluedMap<String, String> parameters=uriInfo.getPathParameters();  
    for(String key:parameters.keySet())  
        System.out.println(key+"="+parameters.get(key));  
    return "OK";  
} } }
```

Paramètres de requêtes: @Context / UriInfo

`getPath() : biblio/uriInfo/a`

`getAbsolutePath(): http://localhost:8080/TPJAXRS/biblio/uriInfo/a`

`getBaseUri(): http://localhost:8080/TPJAXRS/`

`getRequestUri(): http://localhost:8080/TPJAXRS/biblio/uriInfo/a?x=1`

`getPathSegments():`

`biblio`

`uriInfo`

`a`

`getPathParameters()`

`name=[a]`

Paramètres de requêtes : @Context / HttpHeaders

- Un objet de type `HttpHeader` permet d'extraire les informations contenues dans l'entête d'une requête
- Les principales méthodes sont les suivantes:
 - `Map<String, Cookie> getCookies()` : les cookies de la requête
 - `Locale getLanguage()` : le langue de la requête
 - `MultivaluedMap<String, String> getRequestHeaders()` : valeurs des paramètres de l'entête de la requête
 - `MediaType getMediaType()` : le type MIME de la requête
- A noter que ces méthodes permettent d'obtenir le même résultat que les annotations `@HeaderParam` et `@CookieParam`

Paramètres de requêtes : @Context / HttpHeaders

```
@GET  
@Path("httpHeaders")  
public String getInformationFromHttpHeaders(@Context HttpHeaders httpheaders) {  
    Map<String, Cookie> cookies = httpheaders.get Cookies();  
    Set<String> currentKeySet = cookies.keySet();  
    for (String currentCookie : currentKeySet) {  
        System.out.println(currentCookie+"="+cookies.get(currentCookie));  
    }  
    MultivaluedMap<String, String> requestHeaders = httpheaders.getRequestHeaders();  
    Set<String> requestHeadersSet = requestHeaders.keySet();  
    for (String currentHeader : requestHeadersSet) {  
        System.out.println(currentHeader+"="+requestHeaders.get(currentHeader));  
    }  
    return "ok";  
}
```

- host=[localhost:8080]
- connection=[keep-alive]
- cache-control=[max-age=0]
- accept=[text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8]
- user-agent=[Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31]
- accept-encoding=[gzip,deflate, sdch]
- accept-language=[fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4]
- accept-charset=[ISO-8859-1,utf-8;q=0.7,*;q=0.3]

Représentations : @Consumes, @Produces

- L'annotation `@Consumes` est utilisée pour spécifier le ou les types MIME qu'une méthode d'une ressource peut accepter
- L'annotation `@Producer` est utilisée pour spécifier le ou les types MIME qu'une méthode d'une ressource peut produire
- Possibilité de définir un ou plusieurs types MIME
- Ces annotations peuvent être portées sur une classe ou sur une méthode
- L'annotation sur la méthode surcharge celle de la classe
- Si ces annotations ne sont pas utilisées tous types MIME pourront être acceptés ou produits
- La liste des constantes des différents type MIME est disponible dans la classe **MediaType**

Représentations : @Consumes, @Produces

Requête HTTP

```
GET /books/details/12 HTTP/1.1
```

Host: localhost

Accept: text/html

Réponse HTTP

HTTP/1.1 200 OK

Content-Type: text/html

Date:Tue, 07 May 2013 09:58:36 GMT

Server:Apache-Coyote/1.1

<html>

...

</html>

Type MIME accepté par le client

Type MIME de la réponse

Représentations : @Consumes, @Produces

```
package service;
import java.util.*;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
@Path("/banque")
public class RestService {
    @GET
    @Path("/message")
    @Produces(MediaType.TEXT_PLAIN)
    public String getMessage(){
        return "Test Rest full";
    }
    @GET
    @Path("/conversion/{montant}")
    @Produces(MediaType.APPLICATION_JSON)
    public double conversion (@PathParam("montant") double mt){
        return mt*11;
    }
}
```

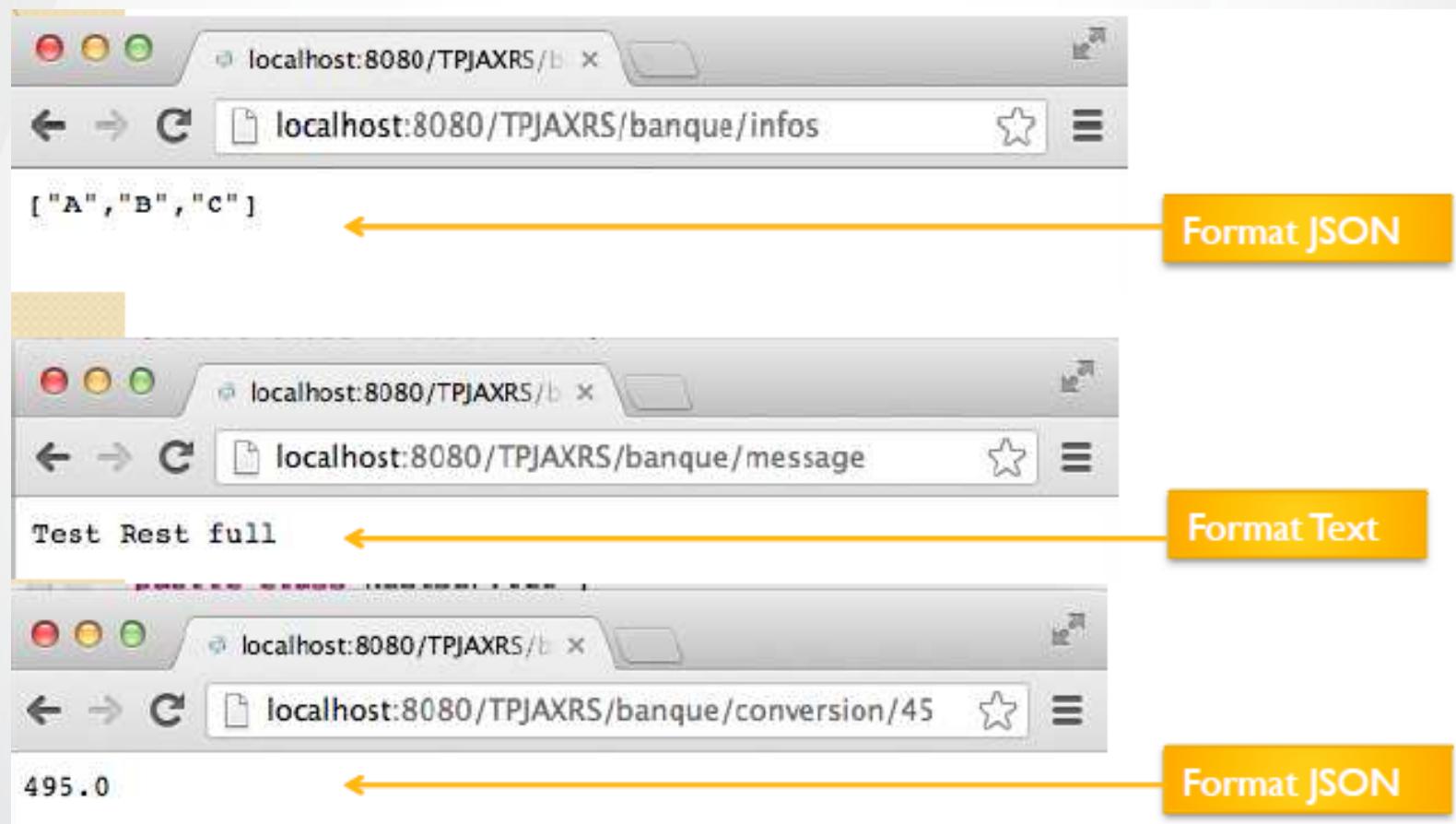
```
@GET
@Path("/infos")
@Produces(MediaType.APPLICATION_JSON)
public List<String> getInfos(){
    List<String> res=new ArrayList<String>();
    res.add("A");res.add("B");res.add("C");
    return res;
}

@GET
@Path("/clients")
@Produces(MediaType.APPLICATION_XML)
public List<Client> getClients(){
    List<Client> res=new ArrayList<Client>();
    res.add(new Client(1,"A"));
    return res;
}
```

```
package service;
import java.io.Serializable;import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class Client implements Serializable {
    private int code;
    private String nom;
    // Getters , Setters et Constructeurs
}
```

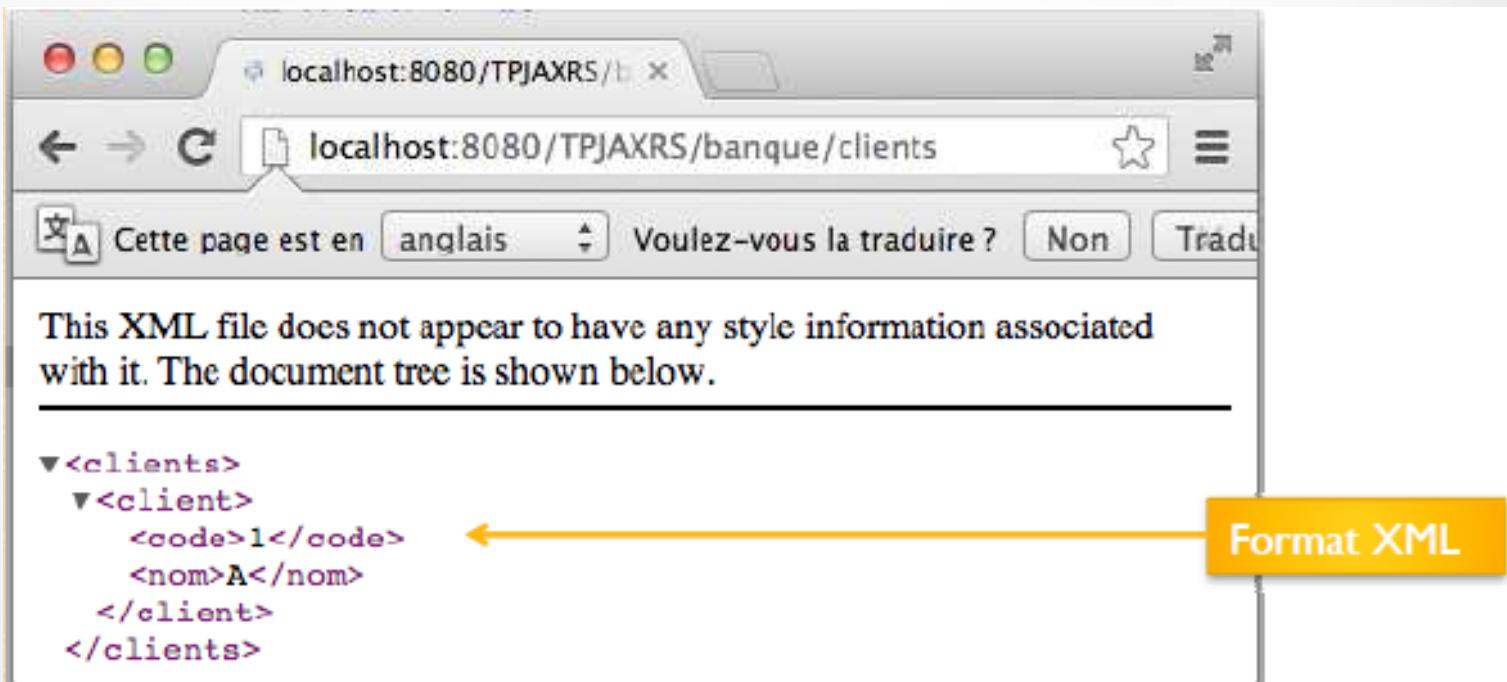
Représentations : @Consumes, @Produces

- Accès au service avec un browser



Représentations : @Consumes, @Produces

- Accès au service avec un browser



Gestion du contenu : statut des réponses

- Lors de l'envoi de la réponse au client un code statut est retourné
- **Réponse sans erreur** : Les statuts des réponses sans erreur s'échelonnent de 200 à 399
 - Le code est 200 « **OK** » pour les services retournant un contenu non vide
 - Le code est 204 « **No Content** » pour les services retournant un contenu vide
- **Réponse avec erreur** : Les statuts des réponses avec erreur s'échelonnent de 400 à 599
 - Une ressource non trouvée, le code de retour est 404 « **Not Found** »
 - Un type MIME en retour non supporté, 406 « **Not Acceptable** »
 - Une méthode HTTP non supportée, 405 « **Method Not Allowed** »

Response

- JAX-RS facilite la construction de réponses en permettant de:
 - de choisir un code de retour
 - de fournir des paramètres dans l'entête
 - de retourner une URI, ...
- Les réponses complexes sont définies par la classe **Response** disposant de méthodes abstraites non utilisables directement
 - **Object getEntity()** : corps de la réponse
 - **int getStatus()** : code de retour
 - **MultivalueMap<String, Object> getMetaData()** : données de l'entête
- Les informations de ces méthodes sont obtenues par des méthodes statiques retournant des **ResponseBuilder**
- Utilisation du patron de conception **Builder**

Principales méthodes de Response

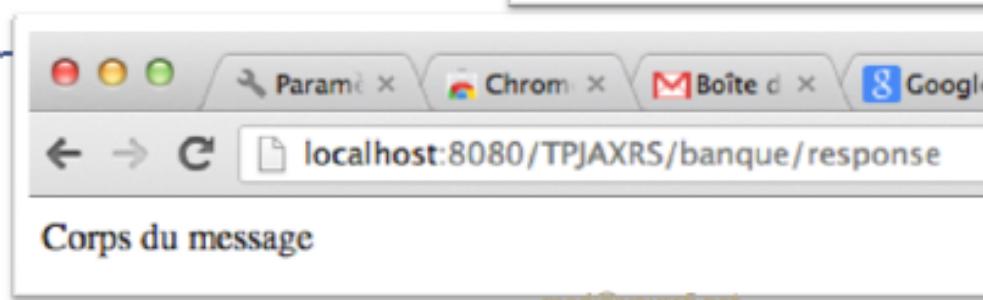
- ResponseBuilder created(URI location) : Modifie la valeur de Location dans l'entête, à utiliser pour une nouvelle ressource créée
- ResponseBuilder notModified() : Statut à « Not Modified »
- ResponseBuilder ok() : Statut à « Ok »
- ResponseBuilder serverError() : Statut à « Server Error »
- ResponseBuilder status(Response.Status) : définit un statut particulier défini dans Response.Status
- Response build() : crée une instance de Response
- ResponseBuilder entity(Object value) : modifie le contenu du corps
- ResponseBuilder header(String, Object) : modifie un paramètre de l'entête

Exemple de Response

```
@Path("response")
@GET
public Response gerReponse(){
    return Response
        .status(Response.Status.OK)
        .header("param1", "valeur1")
        .header("param2", "valeur2")
        .entity("Corps du message")
        .build();
}
```

▼ Response Headers [view source](#)

Content-Type: text/html
Date: Thu, 09 May 2013 17:18:44 GMT
param1: valeur1
param2: valeur2
Server: Apache-Coyote/1.1
Transfer-Encoding: chunked



Exemple de Response (bis)

```
@GET  
@Path("/comptes/v2/{code}")  
@Produces(value={MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})  
public Response compte(@PathParam(value="code")Long code){  
    Compte cp=metier.getCompte(code);  
    return Response  
        .status(Response.Status.OK)  
        .entity(cp)  
        .build();  
}
```



```
{"code":1,"solde":7998.0,"dateCreation":1390770424000}
```

```
- → C localhost:8080/banqueWS/rest/banque/comptes/v2/1
```

This XML file does not appear to have any style information assoc

```
▼<compte>  
  <code>1</code>  
  <dateCreation>2014-01-26T21:07:04Z</dateCreation>  
  <solde>7998.0</solde>  
</compte>
```

Déploiement

- Les applications JAX_RS sont construites et déployées sous le format d'une application Web Java (WAR)
- La configuration de JAX-RS déclare les classes ressources dans le fichier de déploiement (web.xml)
- Deux types de configuration sont autorisées
 - Web.xml pointe sur une sous classe d'Application
 - Web.xml pointe sur une Servlet fournie par l'implémentation JAX-RS
- La classe Application permet de décrire les classes ressources
 - Set<Class> getClasses() : classes des ressources
 - Set<Object> getSingletons() : instances des ressources
- Application fournit une implémentation à vide, la classe PackageResourceConfig fournit une implémentation complète

Déploiement

- Exemple : Déclaration des classes ressources via la Servlet fournie par l'implémentation JERSEY

Servlet fournie par Jersey pour le traitement des requêtes HTTP

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" ...>
    <display-name>HelloWorldRestWebService</display-name>
    <servlet>
        <servlet-name>HelloWorldServletAdaptor</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>fr.ensma.lisi.helloworldrestwebservice</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServletAdaptor</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
</web-app>
```

web.xml du projet

HelloWorldRestWebService

Déploiement

➤ Exemple : Déclaration des classes ressources via Application

The diagram illustrates the deployment configuration for the **LibraryRestWebService** project. It shows two files: **LibraryRestWebServiceApplication.java** and **web.xml**.

LibraryRestWebServiceApplication.java (du projet **LibraryRestWebService**):

```
public class LibraryRestWebServiceApplication extends Application {  
    @Override  
    public Set<Class<?>> getClasses() {  
        Set<Class<?>> classes = new HashSet<Class<?>>();  
        classes.add(BookResource.class);  
        return classes;  
    }  
}
```

web.xml (du projet **LibraryRestWebService**):

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app version="2.5" ...>  
    <display-name>HelloWorldRestWebService</display-name>  
    <servlet>  
        <servlet-name>HelloWorldServletAdaptor</servlet-name>  
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>  
        <init-param>  
            <param-name>javax.ws.rs.Application</param-name>  
            <param-value>  
                fr.ensma.lisi.libraryrestwebserice.LibraryRestWebServiceApplication  
            </param-value>  
        </init-param>  
        <load-on-startup>1</load-on-startup>  
    </servlet>  
    <servlet-mapping>  
        <servlet-name>HelloWorldServletAdaptor</servlet-name>  
        <url-pattern>/*</url-pattern>  
    </servlet-mapping>  
</web-app>
```

A dashed vertical line connects the two files, indicating they belong to the same project.

Objectif du chapitre

- Généralité JAX-RS
- Premier service Web JAX-RS
- Rappels http
- Développement Serveur
 - Chemin de ressource @Path
 - Paramètres des requêtes
 - Gestion du contenu, Response et UriBuilder
 - Déploiement
- **Développement Client**
- Outils

Développement Client

Développement Client : la création de la requête

- La création de la requête s'appuie sur le patron Builder
- Création d'une chaîne d'appel de méthodes dont le type de retour est **WebResource** ou **WebResource.Builder**
- La chaîne d'appel se termine par les méthodes correspondant aux méthodes HTTP (GET, POST, ...)
- La classe **WebResource.Builder** contient les méthodes de terminaison
 - **<T> get(Class<T> c)** : appelle méthode GET avec un type de retour T
 - **<T> post(Class<T> c, Object entity)** : appelle méthode POST en envoyant un contenu dans la requête
 - **<T> put(Class<T> c, Object entity)** : appelle méthode PUT en envoyant un contenu dans la requête
 - **<T> delete(Class<T> c, Object entity)** : appelle méthode DELETE en envoyant un contenu dans la requête

Développement Client : la création de la requête

- La classe **WebResource** fournit des méthodes pour construire l'entête de la requête
- Principales méthodes de **WebResource**:
 - WebResource **path(String)** : définition d'un chemin
 - WebResource **queryParam(String key, String val)** : paramètre requête
 - Builder **accept(MediaType)** : type supporté par le client
 - Builder **header(String name, Object value)** : paramètre entête
 - Builder **cookie(Cookie cookie)** : ajoute un cookie
- Possibilité d'appeler plusieurs fois la même méthode

Exemple de Client java REST

```
import java.net.URI;
import javax.ws.rs.core.UriBuilder;
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
public class ClientJaxRS {
    public static void main(String[] args) {
        ClientConfig config=new DefaultClientConfig();
        Client client=Client.create(config);
        URI uri=UriBuilder.fromUri("http://localhost:8080/TPJAXRS/").build();
        WebResource service=client.resource(uri);
        WebResource path=service.path("banque").path("conversion").path("5");
        String res=path.get(String.class);
        System.out.println(res);
    }
}
```

Client java REST: Méthodes POST et PUT

```
// Requête POST avec un paramètre
WebResource path2=service.path("banque").path("comptesParClient").queryParam("cc","45");
String res2=path2.post(String.class);
System.out.println(res2);

// Requête PUT pour envoyer un objet Client
WebResource path3=service.path("banque").path("newClient");
String res3=path3.put(String.class,new service.Client(3, "ABC"));
System.out.println(res3);
```

```
package service;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class Client implements Serializable {
    private int code;
    private String nom;
    // Constructeurs, Getters et Setters
}
```

Client java REST: ClientResponse

```
System.out.println("-----");
WebResource path4=service.path("banque").path("response");
ClientResponse res4=path4.get(ClientResponse.class);
MultivaluedMap<String, String> headers=res4.getHeaders();
System.out.println(headers.getFirst("param1"));
System.out.println(headers.getFirst("param2"));
System.out.println(res4.getEntity(String.class));
```

```
public class BookResourceIntegrationTest {

    @Test
    public void testGetBooksService() {
        ClientConfig config = new DefaultClientConfig();
        Client client = client.create(config);
        WebResource service = client.resource(getBaseURI());

        WebResource path = service.path("contentbooks").path("response");
        ClientResponse response = path.get(ClientResponse.class);

        MultivaluedMap<String, String> headers = response.getHeaders();
        Assert.assertEquals("Bonjour", headers.getFirst("param1"));
        Assert.assertEquals("Hello", headers.getFirst("param2"));
        String entity = response.getEntity(String.class);
        Assert.assertEquals("Ceci est le message du corps de la réponse", entity);
        Assert.assertEquals("Jetty(6.1.14)", headers.getFirst("server"));
    }

    private static URI getBaseURI() {
        return UriBuilder.fromUri("http://localhost:8088/librarycontentrestwebservice/").build();
    }
}
```



Manipuler le projet LibraryContentRestWebService

Vue d'ensemble du jour

1

- Connaitre les service Web type REST

2

- Comprendre et Développer un WS REST

3

- Manipuler le JAX-RS

Questions?



Sommaire

- **Chapitre 1 : INTRODUCTION**
- **Chapitre 2 : Technologies des services Web**
 - Exemple pratique de création et de déploiement d'un Web service SOAP et de son client
- **Chapitre 3 : Service Web REST**
- **Chapitre 4 : Développement de service web REST avec JAX-RS**
 - Etude de cas : *Développement d'un service RESTful retournant un flux JSON. + Invocation du service et parsing du résultat en Java.*
- **Chapitre 5 : la gestion des Persistances dans un WEB SERVICE REST**
 - Etude de cas : Client : JAX-RS, JaxB, JAVA
- **Chapitre 6 : Sécurité des Web services SOAP VS REST**

Chapitre 5

WS Rest

La gestion des persistances

Objectif du chapitre

- introduction à JAXB
- Mapping XML Schéma vers Java avec JAxB
- Etude de cas

JAXB

Introduction

- Le XML est aujourd'hui un format d'échange de données très utilisé.
- Il possède de nombreux avantages :
 - Standard, simple,
 - Facile à lire.
 - Il peut être lu par un homme, mais ce qui le plus intéressant c'est qu'il peut être lu par un ordinateur (logiciels).
- la puissance du XML repose sur le fait qu'il peut être analysé par un programme et le contenu de ce flux est compris par la machine.
- La structure de ce langage permet en effet de comprendre les relations entre toutes ces données.

Introduction

- De ce fait, les programmeurs ont réalisé de nombreuses API permettant d'accéder aux données XML à travers leurs langages favoris (DOM et SAX par exemple en Java).
- Ces dernières ont des inconvénients.
 - il faut étudier et apprendre une nouvelle API.
 - le programmeur doit adapter le code à son application à chaque fois qu'il veut accéder à des données XML.
 - Ensuite, il faut que le programmeur crée lui-même toutes les classes permettant de gérer ces nouvelles données dans son programme, cela lui prend donc beaucoup de temps.

Utilisation de JAXB

- Pour remédier à ces inconvénients, il existe le « Data Binding » également appelé en français : association de données.
- En Java, Sun a réalisé une API nommée JAXB (Java Architecture for XML Binding) pour simplifier les processus de transformation d'objets Java en fichier XML, et de fichier XML en objets Java.
- JAXB est une spécification qui permet de faire correspondre un document XML à un ensemble de classes et vice et versa via des opérations de sérialisation/désérialisation nommées en anglais (marshaling/unmarshaling).

Utilisation de JAXB

- Les services web envoient des requêtes et des réponses en échangeant des messages XML.
- En Java, il existe plusieurs API de bas niveau pour traiter les documents XML et les schémas XML.
- La spécification JAXB fournit un ensemble d'API et d'annotations pour représenter les documents XML comme des artefacts Java représentant des documents XML.
- JAXB facilite la désérialisation des documents XML en objets et leur sérialisation en documents XML.
- Même si cette spécification peut être utilisée pour n'importe quel traitement XML, elle est fortement intégrée aux services web.

JAXB Java Architecture for XML Binding

- JAXB est l'acronyme de Java Architecture for XML Binding.
- Le but de l'API et des spécifications JAXB est de faciliter la manipulation d'un document XML en générant un ensemble de classes qui fournissent un niveau d'abstraction plus élevé que l'utilisation de JAXP (SAX ou DOM).
- JAXP : Java API for XML Processing
- Avec ces deux API, SAX et DOM, toute la logique de traitement des données contenues dans le document est à écrire.

Qu'est-ce que le «Data Binding» ou association de données

- Le Data Binding est une technologie permettant d'automatiser la transformation d'un modèle de données en un modèle de données objets dans un langage de programmation. Autrement dit, il permet par exemple de convertir les fichiers XML en instances de classes Java.
- Pour réaliser cela, il y a trois étapes :
 - La génération de classes.
 - Le rassemblement des données.
 - La redistribution des données
- Le schéma suivant résume assez bien le principe : un document XML suit les règles de grammaire du « schéma », ce dernier une fois compilé permet de créer une classe correspondante. Cette dernière permettra de créer une instance d'objet correspondant

Schéma XML

```

3 <xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
4   <xs:element name="formes" type="formes" />
5   <xs:complexType name="formes">
6     <xs:sequence>
7       <xs:choice minOccurs="0" maxOccurs="unbounded">
8         <xs:element name="carré" type="carré" />
9         <xs:element name="cercle" type="cercle" />
10      </xs:choice>
11    </xs:sequence>
12  </xs:complexType>

```

Compilation de schéma (xjc)

```

// Définition de l'élément racine et de ses sous-éléments
@XmlRootElement
public class Formes {
  @XmlElements({
    @XmlElement(name = "carré", type = Carré.class),
    @XmlElement(name = "cercle", type = Cercle.class)
  })
  private ArrayList<Forme> formes = new ArrayList<Forme>();
  public ArrayList<Forme> getFormes() { return formes; }
  public void ajouterForme(Forme forme) { formes.add(forme); }
  public void supprimerFormes() { formes.clear(); }
}

```

Respecte

Génération de schéma (schemagen)

IDE interface showing files: ...xml, GestionRépertoire.java, Formes.xml, Formes.java, Principal.java.

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <formes>
3   <cercle>
4     <centre x="100" y="73"/>
5     <rayon>25</rayon>
6   </cercle>
7   <carré>
8     <centre x="147" y="111"/>
9     <côté>100</côté>
10  </carré>
11  <cercle>
12    <centre x="196" y="92"/>
13    <rayon>50</rayon>
14  </cercle>
15 </formes>

```

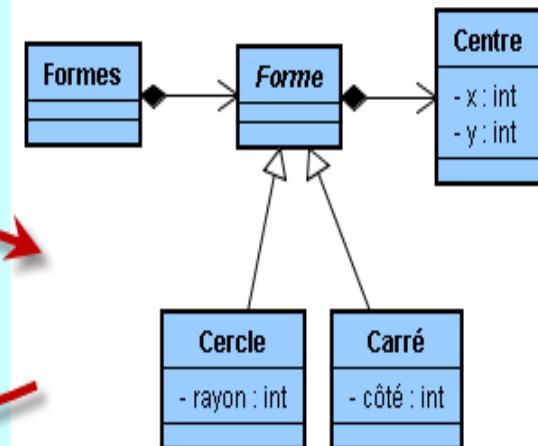
Document XML

Unmarshalling

Marshalling

Objets issus des classes

Instances de



JAXB *Java Architecture for XML Binding*

- JAXB au contraire fournit un outil qui analyse un schéma XML et génère à partir de ce dernier un ensemble de classes qui vont encapsuler les traitements de manipulation du document.
- Le grand avantage est de fournir au développeur un moyen de manipuler un document XML sans connaître XML ou les technologies d'analyse.
- Toutes les manipulations se font au travers d'objets java.
- Ces classes sont utilisées pour faire correspondre le document XML dans des instances de ces classes et vice et versa : ces opérations se nomment respectivement unmarshalling et marshalling.

JAXB *Java Architecture for XML Binding*

- Les données XML sérialisées peuvent être validées par un schéma XML-JAXB peut produire automatiquement ce schéma à partir d'un ensemble de classes et vice versa.
- JAXB fournit également un compilateur de schémas (xjc) et un générateur de schémas (schemagen).
- la sérialisation/désérialisation manipule des objets et des documents XML, ce compilateur et ce générateur de schémas manipulent des classes et des schémas XML.

JAXB 2.0

- JAXB 2.0 a été développée sous la JSR 222 et elle est incorporée dans Java EE 5 et dans Java SE 6.
- Les fonctionnalités de JAXB 2.0 par rapport à JAXB 1.0 sont :
 - Support uniquement des schémas XML (les DTD ne sont plus supportées).
 - Mise en œuvre des annotations.
 - Assure la correspondance bidirectionnelle entre un schéma XML et le bean correspondant.
- Utilisation de fonctionnalités proposées par Java 5 notamment les génériques et les énumérations.

JAXB 2.0

- Le nombre d'entités générées est moins important : JAXB 2.0 génère une classe pour chaque complexType du schéma alors que JAXB 1.0 génère une interface et une classe qui implémente cette interface.
- L'utilisation de JAXB implique généralement deux étapes :
 - Génération des classes et interfaces à partir du schéma XML.
 - Utilisation des classes générées et de l'API JAXB pour transformer un document XML en graphe d'objets et vice et versa, pour manipuler les données dans le graphe d'objets et pour valider le document.

JAXB 2.0

- En plus de son utilité principale, JAXB 2.0 propose d'atteindre plusieurs objectifs :
 - Être facile à utiliser pour consulter et modifier un document XML sans connaissance ni de XML ni de techniques de traitement de documents XML.
 - Être configurable : JAXB met en œuvre des fonctionnalités par défaut qu'il est possible de modifier par configuration pour répondre à ces propres besoins.
 - S'assurer que la création d'un document XML à partir d'objets et retransformer ce document en objets donne le même ensemble d'objets.
 - Pouvoir valider un document XML ou les objets qui encapsulent un document sans avoir à écrire le document correspondant.
 - Être portable : chaque implémentation doit au minimum mettre en œuvre les spécifications de JAXB.

Sérialisation

- La sérialisation d'un graphe d'objets Java est effectuée par une opération dite de marshalling.
- L'opération inverse est dite d'unmarshalling.
- Lors de ces deux opérations, le document XML peut être validé.

L'API JAXB

- L'API JAXB propose un framework composé de classes regroupées dans trois packages :
 - ***javax.xml.bind*** : Contient les interfaces principales et la classe JAXBContext.
 - ***javax.xml.bind.util*** : Contient des utilitaires.
 - ***javax.xml.bind.helper*** : Contient une implémentation partielle de certaines interfaces pour faciliter le développement d'une implémentation des spécifications de JAXB.
 - ***javax.xml.bind.annotation*** : Gestion des annotations pour préciser le mapping entre les classes Java et le document XML correspondant.

Annotations

- JAXB 2.0 utilise de nombreuses annotations définies dans le package javax.xml.bind.annotation essentiellement pour préciser le mode de fonctionnement lors des opérations de marshaling/unmarshaling.
- Ces annotations précisent le mapping entre les classes Java et le documentXML.
- La plupart de ces annotations ont des valeurs par défaut ce qui réduit l'obligation de leur utilisation si la valeur par défaut correspond au besoin.

Transformations

- JAXB 2.0 permet aussi de réaliser dynamiquement à l'exécution une transformation d'un graphe d'objets en document XML et vice et versa.
- C'est cette fonctionnalité qui est largement utilisée dans les services web via l'API JAX-WS 2.0.
- La classe abstraite JAXBContext fournie par l'API JAXB permet de gérer la transformation d'objets Java en XML et vice et versa.

Outils de JAXB 2.0

- JAXB 2.0 propose plusieurs outils pour faciliter sa mise en œuvre :
 - Un générateur de classes Java (schéma compiler) à partir d'un schéma XML nommé xjc dans l'implémentation de référence. Ces classes générées mettent en œuvre les annotations de JAXB.
 - Un générateur de schéma XML (schema generator) à partir d'un graphe d'objets nommé schemagen dans l'implémentation de référence.

La mise en œuvre de JAXB 2.0

- La mise en œuvre de JAXB requiert pour un usage standard plusieurs étapes :
 - La génération des classes en utilisant l'outil xjc de JAXB à partir d'un schéma du document XML.
 - Ecriture de code utilisant les classes générées et l'API JAXB pour :
 - Transformer un document XML en objets Java.
 - Modifier des données encapsulées dans le graphe d'objets.
 - Transformer le graphe d'objets en un document XML avec une validation optionnelle du document.
 - Compilation du code généré.
 - Exécution de l'application.

Appel d'un service Web

- Malgré tous ces concepts, spécifications, standards et organisations, l'écriture et la consommation d'un service web sont très simples.

- Le code suivant, par exemple, présente le code d'un service web qui s'occupe de la gestion du personnel :

```
package session;

import entité.Personne;

import java.util.List;

import javax.ejb.*;

import javax.jws.WebService;

import javax.persistence.*;

@WebService

@Stateless

@LocalBean

public class GestionPersonnel

{ @PersistenceContext

    private EntityManager bd;

    public void nouveau(Personne personne)

    { bd.persist(personne); }

    public Personne rechercher(Personne personne)

    { return bd.find(Personne.class, personne.getId()); }

    public void modifier(Personne personne)

    { bd.merge(personne); }
```

Appel d'un service Web

- Comme les entités ou les EJB, un service web utilise le modèle de classe annotée avec une politique de configuration par exception.
- Si tous les choix par défaut vous conviennent, ceci signifie qu'un service web peut se réduire à une simple classe Java annotée :
`@javax.ws.WebService.`
- Le service `GestionPersonnel` (exemple) propose plusieurs méthodes pour gérer l'ensemble du personnel, par exemple :
`nouveau()`, `rechercher()`, `modifier()`, `supprimer()` et `listePersonnels()`.

Appel d'un service Web

- Un objet Personne est échangé entre le consommateur et le service web.
- Lorsque nous avons décrit l'architecture d'un service web, nous avons vu que les données échangées devaient être des documents XML : nous avons donc besoin d'une méthode pour transformer un objet Java en XML et c'est là que JAXB entre en jeu avec ses annotations et son API.
- L'objet Personne doit simplement être annoté par:
`@javax.xml.bind.annotation.XmlRootElement` pour que JAXB le transforme en XML et réciproquement.

```
package entité;
```

Annotation qui permet de réaliser le parsing entre un document XML et le bean entité correspondant.

```
import java.io.Serializable;
import java.util.*;
import javax.persistence.*;
import javax.xml.bind.annotation.XmlRootElement;
```

Doit être placé en premier.

```
@XmlRootElement
@Entity
@NamedQuery(name="toutes", query="SELECT personne FROM Personne AS personne ORDER BY personne.nom")
public class Personne implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long id;
    private String nom;
    private String prénom;
    private int âge;
    @ElementCollection(fetch=FetchType.EAGER)
    private List<String> telephones = new ArrayList<String>();

    public long getId() { return id; }
    public void setId(long id) { this.id = id; }

    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }

    public String getPrénom() { return prénom; }
    public void setPrénom(String prenom) { this.prénom = prenom; }

    public int getÂge() { return âge; }
    public void setÂge(int âge) { this.âge = âge; }

    public void setTelephones(List<String> telephones) { this.telephones = telephones; }
    public List<String> getTelephones() { return telephones; }

    public Personne() {}

    public Personne(String nom, String prénom, int âge) {
        this.nom = nom;
        this.prénom = prénom;
        this.âge = âge;
    }
}
```

Il est également impératif d'avoir systématiquement tous les getter et setter (toutes les propriétés) de tous les attributs si vous désirez qu'ils transitent sur le réseau.

Appel d'un service Web

- XML est utilisé pour échanger les données et définir les services web via WSDL et les enveloppes SOAP.
- Pourtant, dans le code précédent, un consommateur invoquait un service web sans qu'il n'y ait aucune trace de XML car ce consommateur ne manipulait que des interfaces et des objets Java distants qui, à leur tour, gèrent toute le package XML nécessaire et les connexions réseau.
- Nous manipulons des classes Java à un endroit de la chaîne et des documents XML à un autre - le rôle de JAXB est de faciliter cette correspondance bidirectionnelle.

Appel d'un service Web

- JAXB définit un standard permettant de lier les représentations Java à XML et réciproquement.
- Il gère les documents XML et les définitions des schémas XML (XSD) de façon transparente et orientée objet qui masque la complexité du langage XSD.
- Grâce à cette simple annotation et à un mécanisme de sérialisation, JAXB est capable de créer une représentation XML d'une instance de Personne.

Appel d'un service Web

- Document XML représentant les données d'un personnel de l'entreprise
- ```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <personne>
<id>51</id>
<nom>REMY</nom>
<prenom>Emmanuel</prenom>
<age>71</age>
<telephones>05-78-96-45-45</telephones>
<telephones>06-45-87-85-21</telephones>
<telephones>04-89-77-11-42</telephones>
</personne>
```

# Appel d'un service Web

---

- La sérialisation, ici, consiste à transformer un objet en XML, mais JAXB permet également de faire l'inverse : la désérialisation qui prend ce document XML en entrée et crée un objet Personne à partir des valeurs de ce document.
- JAXB peut produire automatiquement le schéma qui valide automatiquement la structure XML du personnel afin de garantir qu'elle est correcte et que les types des données conviennent.

# Appel d'un service Web

Schéma XML validant le document XML précédent

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

<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI
 2.2.1-hudson-28-. -->

<xs:schema xmlns:tns="http://session/"
 xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"

 targetNamespace="http://session/">

<xs:element name="personne" type="tns:personne" />

<xs:complexType name="personne">

<xs:sequence>

 <xs:element name="id" type="xs:long" />

 <xs:element name="nom" type="xs:string" minOccurs="0" />

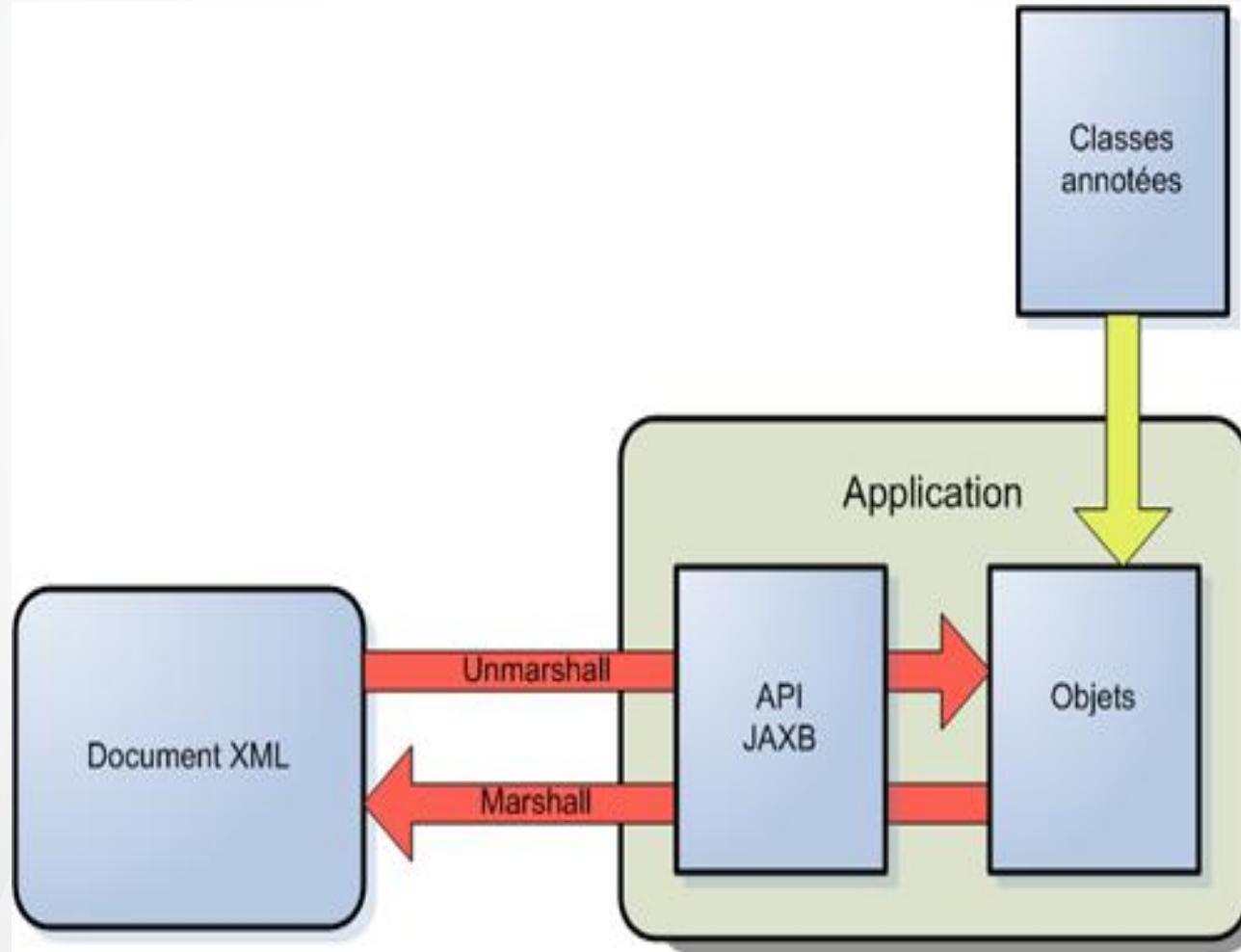
 <xs:element name="prenom" type="xs:string" minOccurs="0" />

 <xs:element name="age" type="xs:int" />
```

# Appel d'un service Web

---

- Le document ci-dessus montre le schéma XML (XSD) de la classe Personne.
- Ce schéma est constitué uniquement d'éléments simples.
- tous les marqueurs sont préfixés par xs(xs:int, xs:long, xs:string, etc.) : ce préfixe est un espace de noms et est défini dans l'élément xmlns (XML namespace) du marqueur d'en-tête du document.
- Les espaces de noms créent des préfixes uniques pour les éléments des documents ou des applications qui sont utilisés ensemble.



# Annotation

- Comme le montre le code suivant, l'équivalent de `@Entity` des objets persistants est l'annotation `@XmlRootElement` de JAXB :

Une classe Personne personnalisée

```
@XmlRootElement
```

```
public class Personne
```

```
{ private long id;
 private String nom;
 private String prenom;
 private int age;
 ... }
```

# Annotation

- L'annotation `@XmlRootElement` prévient JAXB que la classe Personne est l'élément racine du document XML.
- Si cette annotation est absente, JAXB lancera une exception lorsqu'il tentera de sérialiser la classe.
- Cette dernière est ensuite associée au schéma équivalent en utilisant les associations par défaut (chaque attribut est traduit en un élément de même nom).
- A l'aide du marshalling, nous obtenons aisément une représentation XML d'un objet de type Personne.
- L'élément racine `<personne>` représente l'objet Personne et inclut la valeur de chaque attribut de la classe.

# Annotation

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<personne>
 <id>51</id>
 <nom>REMY</nom>
 <prenom>Emmanuel</prenom>
 <age>31</age>
</personne>
```

- JAXB permet de personnaliser et de contrôler cette structure.
- Un document XML étant composé d'éléments (`<element>valeur</element>`) et d'attributs (`<element attribute="valeur" />`), JAXB utilise deux annotations pour les différencier :
  - `@XmlElement` et
  - `@XmlAttribute`.
- Chacune d'elles reconnaît un certain nombre de paramètres permettant de renommer un attribut, d'autoriser ou non les valeurs null, d'attribuer des valeurs par défaut, etc.

# Annotation

- Le code ci-dessous utilise ces annotations spécifiques pour transformer le numéro d'identifiant en attribut (au lieu d'un élément par défaut) et pour renommer l'ensemble des balises du document XML.

```
@XmlRootElement
public class Personne
{ @XmlAttribute(required = true)
 private long id;
 @XmlElement(name = "Nom")
 private String nom;
 @XmlElement(name = "Prenom")
 private String prenom;
 @XmlElement(name = « Age», defaultValue = "21")
 private int age;
 ... }
```

- Cette classe sera donc liée à un schéma différent dans lequel le numéro d'identifiant est un `<xs:attribute>` obligatoire et l'âge renommé possède une valeur par défaut de 21.

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net.
RI's version is JAX-WS RI 2.2.1-hudson-28-. -->
<xs:schema xmlns:tns="http://session/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
targetNamespace="http://session/">
<xs:element name="personne" type="tns:personne" />
<xs:complexType name="personne"> <xs:sequence>
<xs:element name="Age" type="xs:int" default="21" />
<xs:element name="Nom" type="xs:string" minOccurs="0" />
<xs:element name="Prenom" type="xs:string" minOccurs="0" />
</xs:sequence>
<xs:attribute name="id" type="xs:long" use="required" />
</xs:complexType>
</xs:schema>
```

# Annotation

- Document XML représentant les données d'un personnel

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<personne id="51">
 <Nom>REMY</Nom>
 <Prénom>Emmanuel</Prénom>
 <Âge>21</Âge>
</personne>
```

# Annotations

---

- Le tableau ci-dessous énumère les principales annotations de JAXB ;
- La plupart avec les éléments auxquels elles s'appliquent ; certaines peuvent annoter des attributs, d'autres des classes et certaines peuvent s'appliquer à tout un paquetage (@XmlSchema, par exemple).

<i>Annotation</i>	<i>Description</i>
<i>XmlAccessorOrder</i>	<i>Contrôler l'ordre des attributs et des propriétés dans la classe.</i>
<i>XmlAccessorType</i>	<i>Contrôler si l'attribut ou la propriété de la classe est sérialisé par défaut.</i>
<i>XmlAnyAttribute</i>	<i>Convertir une propriété de la classe en un ensemble d'attributs de type jocker dans le document XML.</i>
<i>XmlAnyElement</i>	<i>Convertir une propriété de la classe en une description représentant l'élément JAXB dans le document XML.</i>
<i>XmlAttachmentRef</i>	<i>Marquer un attribut ou une propriété de la classe comme une URI qui fait référence à un type MIME particulier.</i>
<i>XmlAttribute</i>	<i>Convertir une propriété de la classe en un attribut dans le document XML.</i>
<i>XmlElement</i>	<i>Convertir une propriété de la classe en un élément dans le document XML.</i>
<i>XmlElementDecl</i>	<i>Associer une fabrique à un élément XML.</i>
<i>XmlElementRef</i>	<i>Convertir une propriété de la classe en un élément dans le document XML qui est associé à un nom de propriété particulier.</i>
<i>XmlElementRefs</i>	<i>Marquer une propriété de la classe qui fait référence aux classes qui possèdent une XmlElement ou JAXBElement.</i>

<i>XmlElement</i>	<i>Créer un conteneur pour de multiples annotations <i>XmlElement</i>, qui précise ainsi le choix possible parmi celles qui sont proposées.</i>
<i>XmlElementWrapper</i>	<i>Créer un élément père dans le document XML pour des collections d'éléments.</i>
<i>XmlEnum</i>	<i>Convertir une énumération en une représentation équivalente dans le document XML.</i>
<i>XmlEnumValue</i>	<i>Convertir un énumérateur en une représentation équivalente dans le document XML.</i>
<i>XmlID</i>	<i>Convertir une propriété de la classe en un ID XML.</i>
<i>XmlIDREF</i>	<i>Convertir une propriété de la classe en un IDREF XML.</i>
<i>XmlInlineBinaryData</i>	<i>Encoder en base64 dans le document XML.</i>
<i>XmlList</i>	<i>Utilisé pour convertir une propriété de la classe vers une liste de type simple.</i>
<i>XmlMimeType</i>	<i>Associer un type MIME qui contrôle la représentation XML en rapport avec la propriété de la classe.</i>
<i>XmlMixed</i>	<i>Annoter une propriété de la classe qui peut supporter plusieurs types de valeur avec un contenu mixte.</i>
<i>XmlNs</i>	<i>Associer un prefix d'un espace de nommage à une URI.</i>
<i>XmlRegistry</i>	<i>Marquer une classe comme possédant une ou des méthodes annotées avec <i>XmlElementDecl</i>.</i>
<i>XmlRootElement</i>	<i>Associer une classe ou une énumération à un élément XML. Très souvent utilisé pour spécifier la racine du document XML.</i>

<b><i>XmlSchema</i></b>	<i>Associer un espace de nommage à un paquetage.</i>
<b><i>XmlSchemaType</i></b>	<i>Associer un type Java ou une énumération à un type défini dans un schéma.</i>
<b><i>XmlSchemaTypes</i></b>	<i>Conteneur pour plusieurs propriétés annotées par XmlSchemaType.</i>
<b><i>XmlTransient</i></b>	<i>Marquer une entité pour qu'elle ne soit pas mappée dans le document XML.</i>
<b><i>XmlType</i></b>	<i>Convertir une classe ou une énumération vers le type spécifié dans Schéma XML correspondant.</i>
<b><i>XmlValue</i></b>	<i>Mapper une classe vers le type complexe dans le Schéma XML ou vers le type simple suivant le cas.</i>

# Cas Pratique de création de WS REST avec Maven et Jaxb et son client Java

- **Maven**
- **Création WS**
- **Création Client**
- **Test et déploiement**



# Maven, le choix Projet

---

- Le projet peut-être librement découpé en modules
  - Maven ne cristallise pas l'architecture de l'application
- Gestion des dépendances
  - Déclaratif : Gestion automatique du téléchargement et de l'utilisation dans le projet.
  - Transitivité : On ne déclare que ce que l'on utilise

# Maven, le choix Projet

- Centralise et automatise les différents aspects du développement de logiciels
- Une seule chose qu'il ne peut faire à votre place ☺ :  
**Développer**
  - Construction
  - Tests
  - Packaging
  - Déploiement
  - Documentation
  - Contrôles et rapports sur la qualité des développements

# Industrialisez !

- Partagez vos versions recommandées d'artifacts :
  - POM de dependencyManagement
  - Scope import
- Partagez vos modèles d'applications :
  - Archetypes
- Automatisez votre processus de release

# Industrialisez !

- Mettez en place une démarche de tests
  - Tout l'outillage est pré-intégré :
    - Junit, TestNG – tests unitaires,
    - Selenium, Canoo – tests d'IHM,
    - Fitnesse, Greenpepper – tests fonctionnels,
    - SoapUI – tests d'intégration WS
    - Et bien d'autres encore

# Maven : POM

- Project Object Model (POM)
- Chaque projet ou sous-projet est configuré par un POM qui contient les informations nécessaires à Maven pour traiter le projet (nom du projet, numéro de version, dépendances vers d'autres projets, bibliothèques nécessaires à la compilation, noms des contributeurs etc.).
- Ce POM se matérialise par un fichier **pom.xml** à la racine du projet.
- Cette approche permet l'héritage des propriétés du projet parent.
  - Si une propriété est redéfinie dans le POM du projet, elle recouvre celle qui est définie dans le projet parent.
  - Ceci introduit le concept de réutilisation de configuration.
  - Le fichier pom du projet principal est nommé *pom parent*. Il contient une description détaillée de votre projet, avec en particulier des informations concernant le versionnage et la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe, la structure et bien plus.

# MAVEN : arborescence

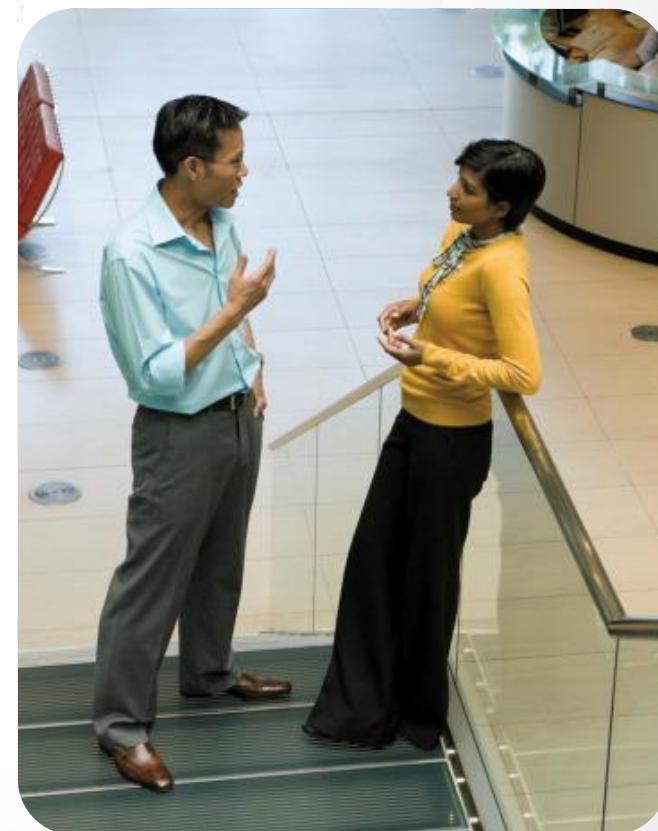
- Maven impose une arborescence et un nommage des fichiers du projet selon le concept de conventions.
- Ces conventions permettent de réduire la configuration des projets, tant qu'un projet suit les conventions. Si un projet a besoin de s'écarte de la convention, le développeur le précise dans la configuration du projet.
- Voici une liste non-exhaustive des répertoires d'un projet Maven :
  - /src : les sources du projet
  - /src/main : code source et fichiers source principaux
  - /src/main/java : code source
  - /src/main/resources : fichiers de ressources (images, fichiers annexes etc.)
  - /src/main/webapp : webapp du projet
  - /src/test : fichiers de test
  - /src/test/java : code source de test
  - /src/test/resources : fichiers de ressources de test
  - /src/site : informations sur le projet et/ou les rapports générés suite aux traitements effectués
- /target : fichiers résultat, les binaires (du code et des tests), les packages générés et les résultats des tests

# MAVEN : Cycle de vie

- Les buts (*goals* en anglais) principaux du cycle de vie d'un projet Maven sont:
  - compile
  - test
  - package
  - install
  - deploy
- L'idée est que, pour n'importe quel but, tous les buts en amont doivent être exécutés sauf s'ils ont déjà été exécutés avec succès et qu'aucun changement n'a été fait dans le projet depuis.
- Par exemple, quand on exécute mvn install, Maven va vérifier que mvn package s'est terminé avec succès (le jar existe dans target/), auquel cas cela ne sera pas ré-exécuté.
- D'autres buts sont exécutables en dehors du cycle de vie et ne font pas partie du cycle de vie par défaut de Maven (ils ne sont pas indispensables). Voici les principaux :
  - clean
  - assembly:assembly
  - site
  - site-deploy

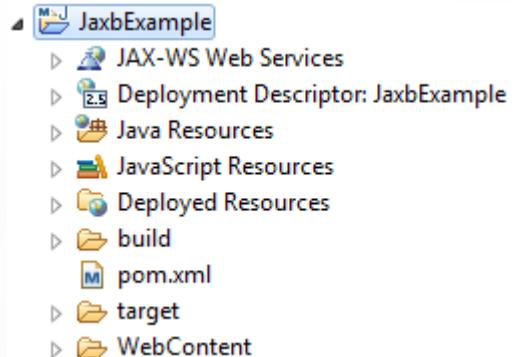
# Cas Pratique de création de WS REST avec Maven et Jaxb et son client Java

- Maven
- **Création WS**
- Création Client
- Test et déploiement



# Création du WS

- Sous eclipse :
  - Création d'un nouveau projet : dynamic web project
    - Dynamic web module version 2.5
    - Project Name : JaxbExample
  - Configuration en projet Maven ( clic droit sur le projet > configure > configure maven project)
    - NB : si il y a des erreur il faut faire maven update du projet pour se mettre à jour
- La structure du projet doit ressembler à ça :



# WS: web.xml

## ➤ Ajout du WS servlets dans le web.xml

```
<display-name>JaxbExample</display-name>
<servlet>
 <servlet-name>jersey-XMLExample-serlvet</servlet-name>
 <servlet-class>
 com.sun.jersey.spi.container.servlet.ServletContainer
 </servlet-class>
 <init-param>
 <param-name>com.sun.jersey.config.property.packages</param-name>
 <param-value>com.wsformation.enterprise.rest.jersey</param-value>
 </init-param>
 <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
 <servlet-name>jersey-XMLExample-serlvet</servlet-name>
 <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

## ➤ L'url sera alors de type :

- [http://localhost:8080/JaxbExample/rest/...](http://localhost:8080/JaxbExample/rest/)

# Ajout des objets java avec l'annotation jaxb

- Création d'une nouvelle classe:
  - Package : com.wsformation.enterprise.rest.jersey
  - Nom de la classe : Student.java

```
@XmlRootElement(name = "student")
public class Student {

 private int id;
 private String firstName;
 private String lastName;
 private int age;

 // Must have no-argument constructor
 public Student() {
 }

 public Student(String fname, String lname, int age, int id) {
 this.firstName = fname;
 this.lastName = lname;
 this.age = age;
 this.id = id;
 }

 @XmlElement
 public void setFirstName(String fname) {
 this.firstName = fname;
 }

 public String getFirstName() {
 return this.firstName;
 }
}
```

# Suite de la classe Student.java

- Dans cette classe on a utilisé :
  - `@XmlRootElement`: qui définit l'élément racine de l'XML.
  - `@XmlElement`: est utilisé pour définir les éléments dans l'XML.
  - `@XmlAttribute`: est utilisé pour définir un attribut de l'élément root

```
@XmlElement
public void setLastName(String lname) {
 this.lastName = lname;
}

public String getLastName() {
 return this.lastName;
}

@Override
public String toString() {
 return new StringBuffer(" First Name : ").append(this.firstName)
 .append(" Last Name : ").append(this.lastName)
 .append(" Age : ").append(this.age).append(" ID : ")
 .append(this.id).toString();
}
```

# Création du service REST

- Ajouter les dépendances de jersey dans le pom.xml du projet :

```
<dependencies>
 <dependency>
 <groupId>com.sun.jersey</groupId>
 <artifactId>jersey-server</artifactId>
 <version>1.9</version>
 </dependency>
 <dependency>
 <groupId>com.sun.jersey</groupId>
 <artifactId>jersey-client</artifactId>
 <version>1.9</version>
 </dependency>
</dependencies>
```

- Création d'une nouvelle classe :
  - Package : com.wsformation.enterprise.rest.jersey
  - Nom : JerseyRestService

# La classe : JerseyRestService.java

- Déployer l'application et tester le WS en tapant l'url :
  - <http://localhost:8080/JaxbExample/rest/xmlServices/print/ouajih>

```
@Path("/xmlServices")
public class JerseyRestService {
 @GET
 @Path("/print/{name}")
 @Produces(MediaType.APPLICATION_XML)
 public Student responseMsg(@PathParam("name") String name) {

 Student st = new Student(name, "Diaz",22,1);

 return st;
 }
}
```

The screenshot shows a browser interface with the following details:

- Method: GET
- Endpoint: http://127.0.0.1:8080/JaxbExample/rest/xmlServices/print/ouajih
- Resource: /JaxbExample/rest/xmlServices/print/ouajih
- Content Type: XML
- XML Response:

```
<student id="1">
 <age>22</age>
 <firstName>ouajih</firstName>
 <lastName>Diaz</lastName>
</student>
```

# Ajout d'un service qui consomme un objet de type Student

- Dans le même web service JerseyRestService.java ajouter la fonction suivante

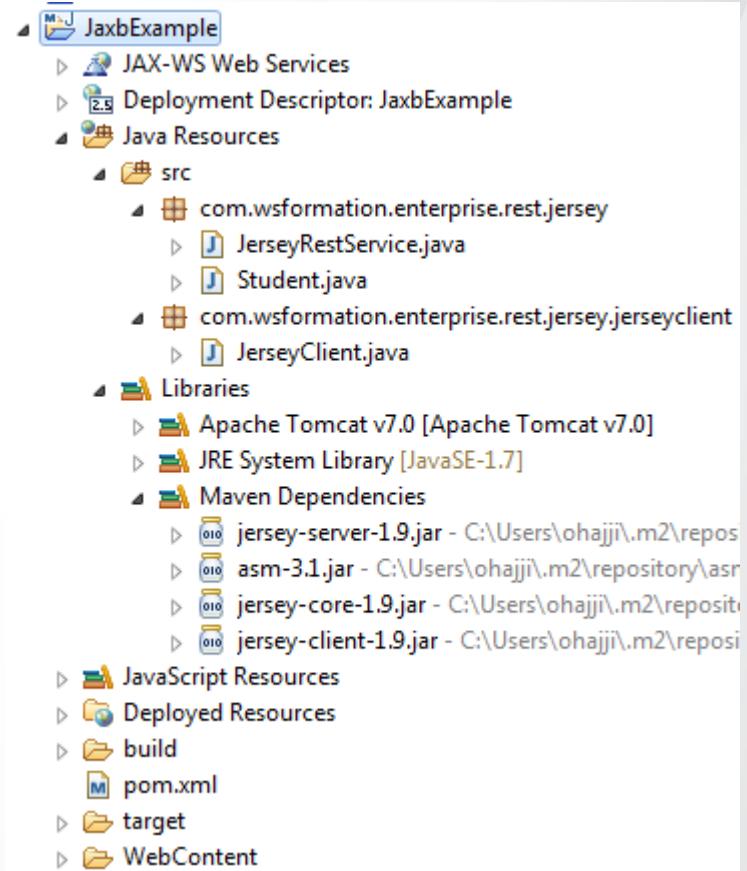
```
@POST
@Path("/send")
@Consumes(MediaType.APPLICATION_XML)
public Response consumeXML(Student student) {

 String output = student.toString();

 return Response.status(200).entity(output).build();
}
```

# Création du client du webservice

- Pour consommer le web service on va créer une nouvelle classe java standard qui va appeler le WS
  - Nom de la classe : JerseyClient.java
  - Package : com.wsformation.enterprise.rest.jersey.jerseyclient
  - Cocher l'ajout du constructeur main
  - L'arbo du projet doit maintenant correspondre à ça :



# Développement de la classe cliente

- → Comme vous le voyez , on a créé une simple instance de type Student et on l'a envoyé via un POST au service Web

```
public class JerseyClient {
 public static void main(String[] args) {

 try {
 Student st = new Student("Adriana", "Barrer", 12, 9);
 Client client = Client.create();

 WebResource webResource = client
 .resource("http://127.0.0.1:8080/JaxbExample/rest/xmlServices/send");

 ClientResponse response = webResource.accept("application/xml")
 .post(ClientResponse.class, st);

 if (response.getStatus() != 200) {
 throw new RuntimeException("Failed : HTTP error code : "
 + response.getStatus());
 }

 String output = response.getEntity(String.class);

 System.out.println("Server response \n");
 System.out.println(output);
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}
```

# Résultat

- *En exécutant le client ( run as java application ) on aura comme output :*

Server response :

First Name : Adriana Last Name : Barrer Age : 12 ID : 9

- **Et voici la requête poste :**

POST /JerseyXMLExample/rest/xmlServices/send HTTP/1.1 Accept: application/xml

Content-Type: application/xml

User-Agent: Java/1.7.0\_45

Host: localhost:8080

Connection: keep-alive Content-Length: 151

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<student id="9"> <age>12</age> <firstName>Adriana</firstName>
<lastName>Barrer</lastName> </student>
```

# Sommaire

- **Chapitre 1 : INTRODUCTION**
- **Chapitre 2 : Technologies des services Web**
  - Exemple pratique de création et de déploiement d'un Web service SOAP et de son client
- **Chapitre 3 : Service Web REST**
- **Chapitre 4 : Développement de service web REST avec JAX-RS**
  - Etude de cas : *Développement d'un service RESTful retournant un flux JSON. + Invocation du service et parsing du résultat en Java.*
- **Chapitre 5 : la gestion des Persistances dans un WEB SERVICE REST**
  - Etude de cas : Client : JAX-RS, JaxB, JAVA
- **Chapitre 6 : Sécurité des Web services SOAP VS REST**

# Chapitre 6

## Sécurité

# des Web services SO

## AP VS REST

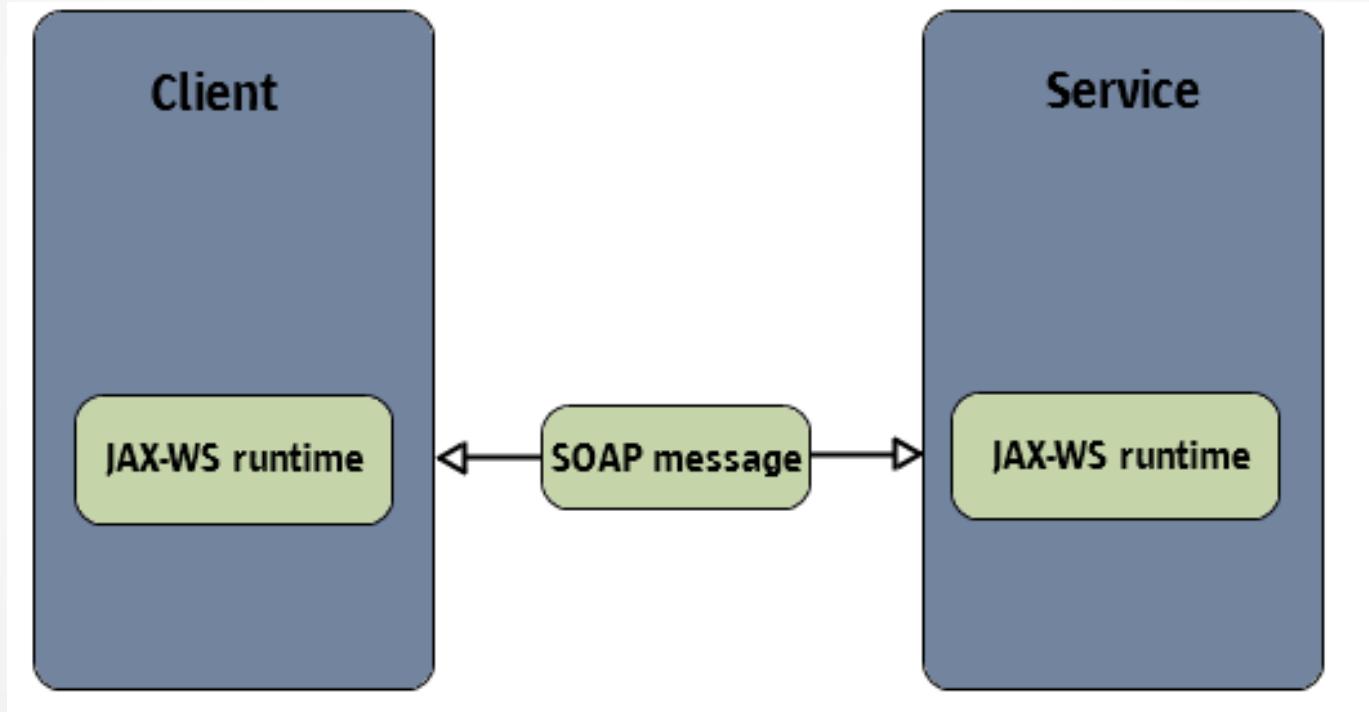
# Objectif du chapitre

---

- Qu'est-ce qu'un Web Service ?
- SOAP
- REST
- Threat Modeling / ACME SA
- Réduction des risques
- Conclusion
- Questions

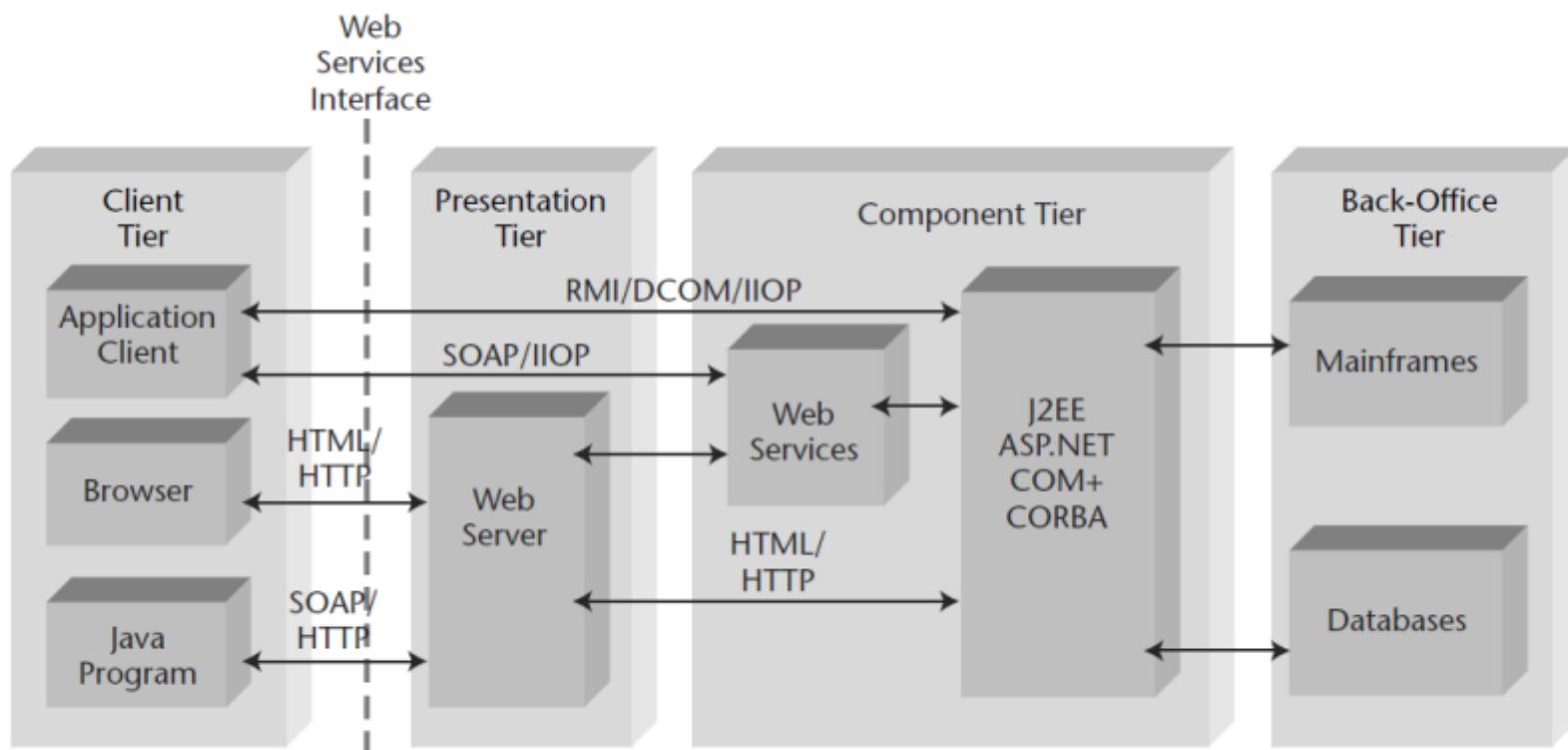
# Qu'est-ce qu'un Web Service

# Web Service ?



Un **service web** (ou **service de la toile**<sup>1</sup>) est un programme informatique permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur [internet](#) ou sur un [intranet](#), par et pour des applications ou machines, sans intervention humaine, et de manière synchrone.

# Architecture typique d'un WS

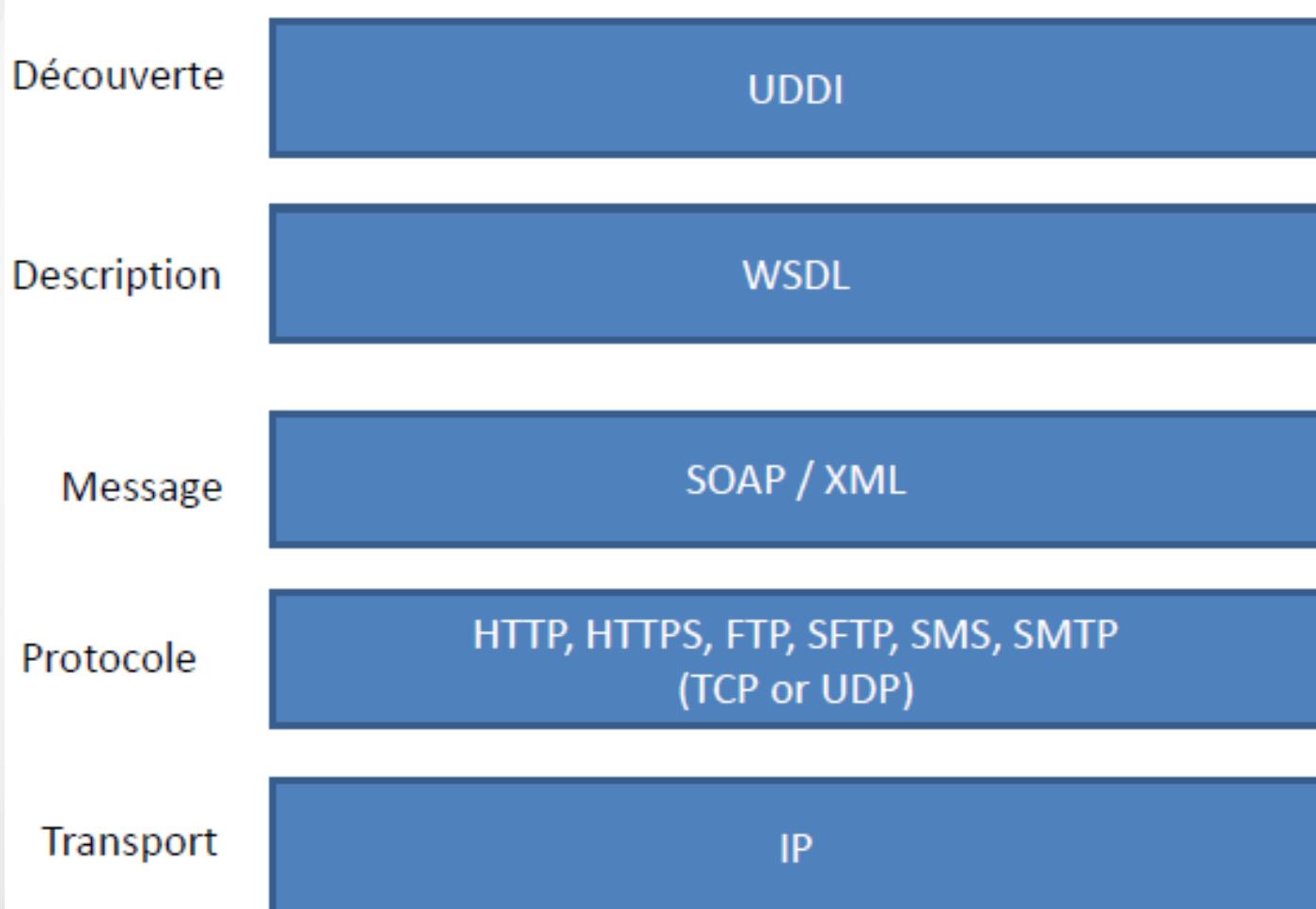


**Figure 1.1** Typical Web Services environment.

# SOAP

- Langages
  - XML
  - WSDL : Descripteur du service
  - UDDI: Annuaire des services
  - Xpath
- Protocoles
  - Transport: HTTP, HTTPS, SMTP, FTP, SMS, TFTP, SSH, etc.  
(TCP or UDP)
  - Message: Enveloppe SOAP
- Sécurité
  - WS-Security (Signature & Chiffrement)
- Autres éléments
  - AuthN: SAML, X509, Username & Password, Kerberos, HTTP Digest, etc

# SOAP : Démystification des protocoles



# REST :

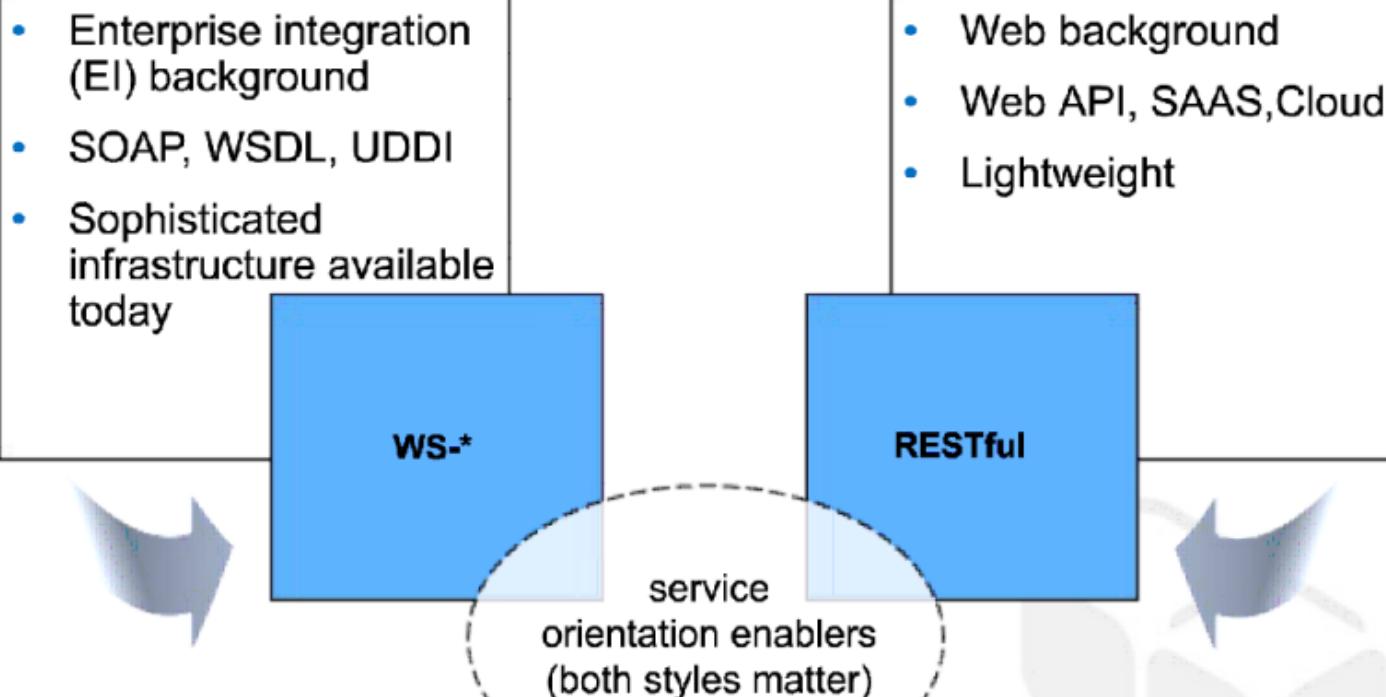
- Langages
  - XML
  - JSON
  - XHTML, HTML, PDF... as data formats
- Protocoles
  - HTTP(s) ☐ Utilisation d'une URL
  - Méthode de communication (GET, POST, PUT, DELETE)
- Sécurité
  - Sécurité du transport (SSL/TLS)
  - Sécurité des messages: HMAC / Doseta / JWS, etc. (Like XML Signature)
- Autres éléments
  - Oauth, API Keys, etc.

# REST : Démystification des protocoles

Découverte	???
Description	WADL, Swagger ***
Message	XML, JSON, etc.
Protocole	HTTP, HTTPS
Transport	TCP/IP

# SOAP VS REST

## Web services in the Enterprise



# Les attaques sur les WebServices : XML Bomb

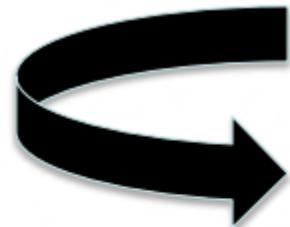
- Trivial à effectuer :
  - Référence récursive à une entité du même document :

```
<?xml ...
...
<!entity owasp0 « Owasp »>
<!entity owasp1 « &owasp0;&owasp0>
...
...
<!entity owasp424242
« &owasp424241;&owasp424241 »>
<owasptest>&owasp424242;</owasptest>
```

- Peut provoquer un déni de service !

# Injection XML entity

- La possibilité d'injecter du code XML de type entity system peut être catastrophique



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
 <!ELEMENT foo ANY >
 <!ENTITY xxe SYSTEM "file:///dev/random" >]><foo>&xxe;</foo>
```

Crash du système



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
 <!ELEMENT foo ANY >
 <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>
```

Obtention des mots de passe

# Injection XML

## DTD :

```
<!DOCTYPE users [
 <!ELEMENT users (user+) >
 <!ELEMENT user (username,password,userid,mail+) >
 <!ELEMENT username (#PCDATA) >
 <!ELEMENT password (#PCDATA) >
 <!ELEMENT userid (#PCDATA) >
 <!ELEMENT mail (#PCDATA) >
>]
```



[http://www.example.com/addUser.jsp?  
username=tony&password=Un6R34kble</password><!--&email=-->  
<userid>0</userid><mail>s4tan@hell.com](http://www.example.com/addUser.jsp?username=tony&password=Un6R34kble</password><!--&email=--><userid>0</userid><mail>s4tan@hell.com)



## Résultat :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
 <user>
 <username>tony</username>
 <password>Un6R34kble</password><!--</password>
 <userid>500</userid>
 <mail>--><userid>0</userid><mail>s4tan@hell.com</mail>
 </user>
</users>
```

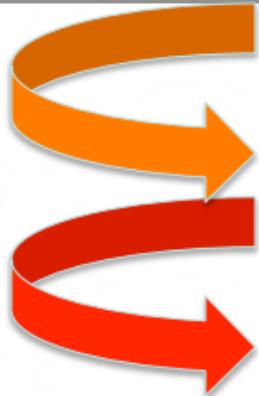
# Injection CDATA

- Le contenu des élément CDATA est éliminé lors du parsing soit :

```
<html>
$HTMLCode
</html>
```



```
$HTMLCode = <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
```



**Avant analyse du parser:**

```
<html>
 <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
</html>
```

**Après Analyse:**

```
<html>
 <script>alert('XSS')</script>
</html>
```

# Injection xPath

Imaginons la base d'authentification Xml suivante

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
<user>
<username>gandalf</username>
<password>!c3</password>
<account>admin</account>
</user>
<user>
<username>Stefan0</username>
<password>w1s3c</password>
<account>guest</account>
</user>
</users>
```

La chaine de recherche étant :



string(//user[username/text()='gandalf' and password/text()='!c3']/account/text())



Si l'utilisateur entre :

Username: ' or '1' = '1  
Password: ' or '1' = '1



string(//user[username/text()=" or '1' = '1' and password/text()=" or '1' = '1"]/account/text())

# Injection xPath - dump XML

Le dump d'un document XML est rendu possible via le caractère |

Soit le descriptif d'un champ

```
//item[itemID='$id']/description/text()
```



Si l'utilisateur entre :

```
$itemID=chaine'] | /* | //item[itemID='chaine
```

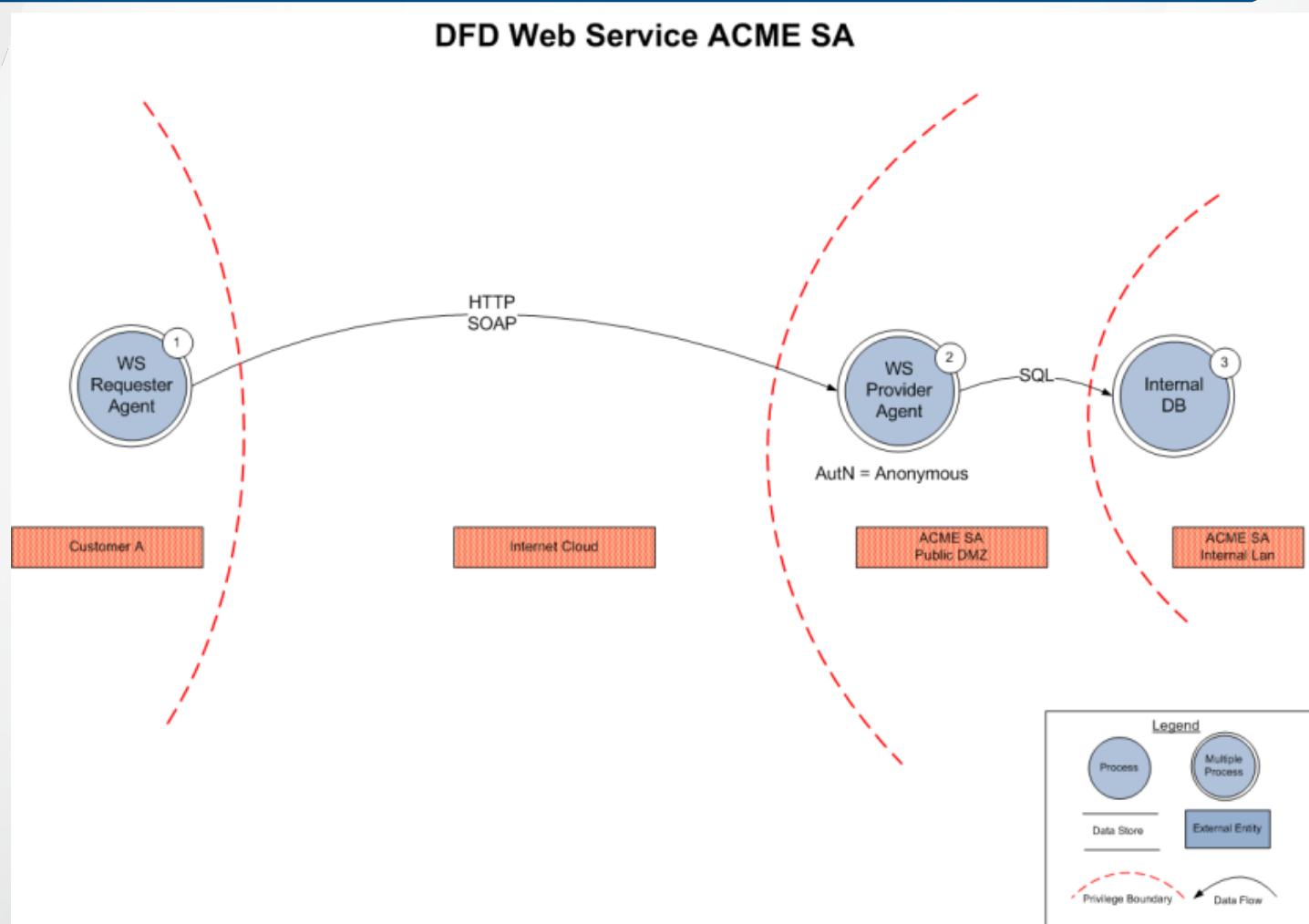


```
//item[itemID='chaine'] | /* | //item[itemID='chaine']/description/text()
```

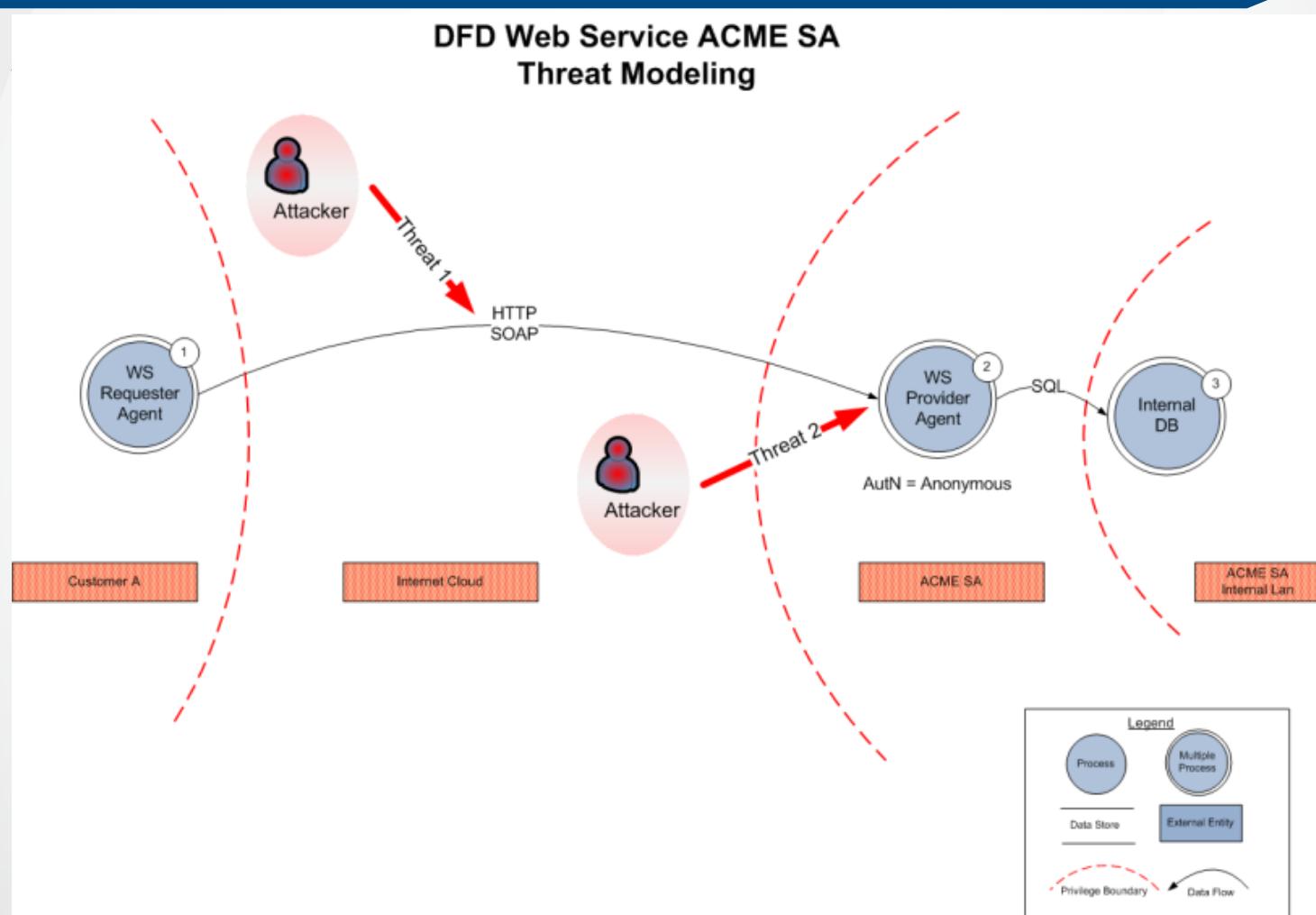


Match de tous les nœuds !!!!!

# Threat Modeling / ACME SA



# Threat Modeling / ACME SA



# Rejet des messages SOAP

---

- SOAP est un protocole d'échanges
- SOAP ne dispose pas d'un mécanisme de sessions :
  - Aucune relation entre les messages
  - Rejet possible très facilement :
    - Authentification
    - Messages
    - DOS...

# Modèle STRIDE

- STRIDE est un acronyme désignant le spectre des menaces de sécurité auxquelles votre application peut être confrontée. Les six catégories de menaces suivantes peuvent vous aider à identifier les vulnérabilités et les vecteurs d'attaque potentiels dans vos propres applications.
  - **Usurpation d'identité** : Usurper une identité équivaut à se faire passer pour quelqu'un d'autre lors de l'accès à l'ordinateur. Un exemple d'usurpation d'identité est le fait d'accéder illégalement aux informations d'authentification d'un autre utilisateur, telles qu'un nom d'utilisateur et un mot de passe, et de les utiliser.
  - **Falsification de données** : La falsification implique la modification malveillante de données. Il peut s'agir, par exemple, de la modification non autorisée de données persistantes, telles que celles contenues dans une base de données, et l'altération de données échangées entre deux ordinateurs sur un réseau ouvert tel qu'Internet.
  - **Répudiation** : Les menaces de répudiation impliquent des utilisateurs qui peuvent refuser d'exécuter une action sans que d'autres parties aient la possibilité de le constater. Par exemple, un utilisateur peut effectuer une opération non autorisée dans un système n'offrant pas la possibilité de tracer les opérations interdites.

# Modèle STRIDE

- **De la même façon, la non-répudiabilité** : désigne la capacité d'un système à contrer des menaces de répudiation. Par exemple, un utilisateur qui achète un article peut avoir à signer un document lors de sa réception. Le vendeur peut ensuite utiliser le reçu signé comme preuve que l'utilisateur a effectivement réceptionné le paquet.
- **Divulgation d'informations** : Les menaces de divulgation d'informations impliquent l'exposition d'informations à des individus qui ne sont pas supposés y avoir accès. Il peut s'agir, par exemple, de la capacité d'un utilisateur à lire un fichier auquel il n'est pas autorisé à accéder ou la capacité d'un intrus à lire des données transitant entre deux ordinateurs.
- **Refus de service** : Les attaques de refus de service entraînent une perte de service pour les utilisateurs valides, par exemple, en rendant un serveur Web temporairement indisponible ou inutilisable. Vous devez assurer une protection contre certains types de menace d'attaque de refus de service afin d'améliorer la disponibilité et la fiabilité du système.
- **Elévation du privilège** : Dans ce type de menace, un utilisateur non privilégié obtient un accès privilégié et dispose donc d'un accès suffisant pour compromettre ou détruire le système tout entier. Les menaces d'élévation du privilège incluent les situations dans lesquelles un agresseur a effectivement franchi toutes les défenses du système et fait partie du système de confiance proprement dit.

# Menaces – DFD Acme SA

---

## ➤ Threat 1

- Interception des messages (Information disclosure)
- Modification des messages (Tampering)
- Usurpation d'identité (Spoofing)

## ➤ Threat 2

- Attaque de l'application
  - BoF
  - Injection
  - DoS & DDoS
  - Etc

Relations de confiance – WS Trust/WS Federation/  
LibertyAlliance

SOAP – WS Security / SAML /WS Policy

XML – XML Encryption

XML – Xml Signature

HTTP – HTTP Authentification

TCP – SSL /TLS

IP - IPSec

Securité Web Classique

# Réduction des risques ?

- Chiffrement du transport /
- AuthN
- SSL Mutual AuthN / X509
- WAF / XML Gateway
- Intégrité et confidentialité des messages
- Secure Coding

# TOP 10 des risques et parades

A1: Manque  
d'authentification

A2: Manque  
d'habilitation

A3: Manque de  
trace d'audit

A4: Manque de  
politique de  
sécurité

A5: Défaillance  
XML

A6:  
Détournement  
d'identité

A7: Faiblesse des  
clefs

A8: Stockage  
cryptographique  
non sécurisé

A9:  
Communications  
non sécurisées

A10 : Top 10 Web

# A1 – Manque d'authentification

---

- **But :**
  - Rejet des transactions
  - Elévation de privilèges
- **Principe :**
  - Envoie d'un message SOAP au point d'accès
- **Dangerosité :**
  - Faible à Forte
- **Protection :**
  - Utilisation de SSL
  - Utilisation des couches WS-Security
  - Mettre en place des principes d'unicité des transactions

# A2 – Manque d'habilitation

---

- But :
  - Premier => modifier des fichiers/données
  - Final => Elévation de privilèges
- Principe :
  - Envoie d'un message SOAP au point d'accès contenant des identifiants valides
- Dangereux :
  - Faible à Forte
- Protection :
  - Mise en place d'une habilitation dans le code
  - Protéger les fichiers de politiques et des DTDs
  - Ne pas se contenter d'URLs non publiées

# A3 – Manque de trace d'audit

- Constat :
  - Les framework de WebServices ont peu de traces des événements.
  - Il devient quasi-impossible de pouvoir tracer correctement les flux dans le cas d'orchestrations complexes.
- Protection :
  - Mettre en place des traces d'audit dans le code en particulier des appels :
    - D'authentification
    - Changement dans le système (création/destruction/modification)
    - Dépassement des limites
    - Lancement, arrêt de fonctions

# A4 – Manque de politique de sécurité

- Constat :
  - Du à la conception des WebServices, il est très difficile de disposer d'une politique globale.
  - De base les framework ne comportent pas de contrôle d'accès
- Dangérosité :
  - Faible à Forte
- Protection :
  - Mettre en place une politique de sécurité des WebServices :
    - Sur les protocoles de communication (HTTP/HTTPS/...)
    - Sur l'échange des messages
    - Sur la gestion des clefs de chiffrement
    - Sur la protection contre les rejeux
    - # ....
  - Mettre en place les tags WS-SecurityPolicy

# A5 – Défaillance XML

- But :
  - Abuser les parseurs XML pour obtenir :
    - Des informations
    - Des privilèges
    - ....
  - Dangerosité :
    - Forte
  - Protection :
    - Vérifier que les parseurs XML utilisées sont immunes aux problèmes d'injection de type DOS/Entité, ....
    - Vérifier la taille des documents XML lors des utilisations
    - Vérifier que l'intégralité du document est signé par juste une partie (dans le cas d'utilisation des signatures)
    - Vérifier le document XML via le schéma le plus strict
    - Ne pas faire confiance à une pré-validation des données de l'expéditeur

# A6 – Détournement d'identité

- Constat :
  - Les WebServices utilisent l'identité de l'appelant dans :
    - Le contrôle d'accès
    - Les décisions de routage des appels
    - La logique métier
  - Les frameworks ne disposent pas de fonctions de protection de l'identité
- But :
  - Elévation de privilèges
  - Obtention d'informations
- Dangerosité :
  - Forte
- Protection :
  - Mettre en place WS-Security, les assertions SAML
  - Vérifier les signatures des messages
  - Utiliser l'authentification forte

# A7 – Faiblesse des clefs

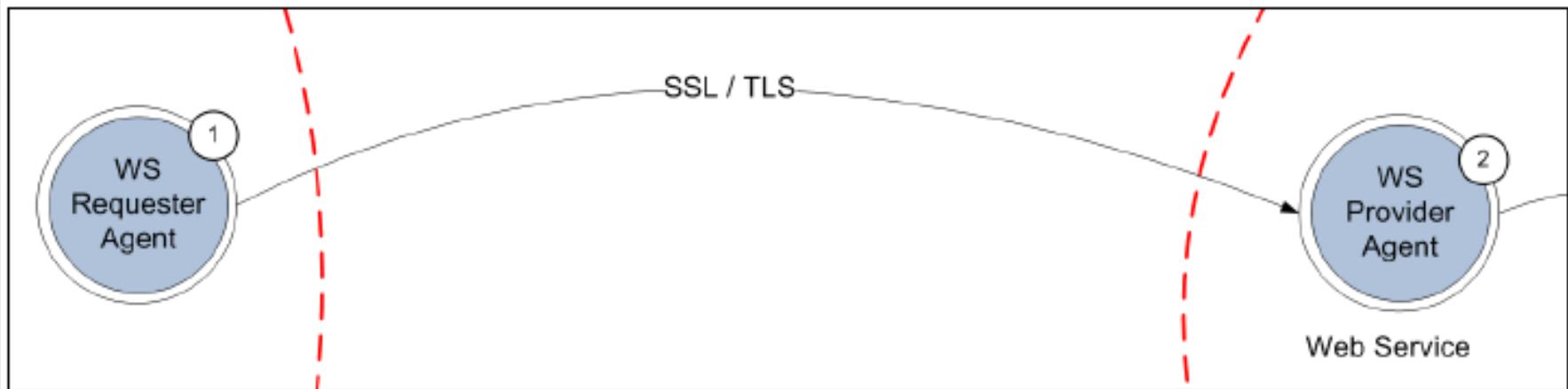
- Constat :
  - Il n'existe pas de protection des messages et de l'authentification en XML et HTTP.
  - Le standard WS-Security ne suffit pas à protéger les clefs d'accès car les données sont passées en clair.
- But :
  - Elévation de privilèges
  - Obtention d'informations
- Dangereux :
  - Forte
- Protection :
  - Mettre en place du chiffrement de bout en bout (SSL/IPSec)
  - Utiliser des authentifications fortes (certificats X509, OTP, ..)
  - Mettre en place des mécanismes anti-rejet
  - Ne pas autoriser d'authentification en clair

# A8/A9/A10

- Ces différentes failles sont à lier au Top10 OWASP classique, mais appliquées aux WebServices.

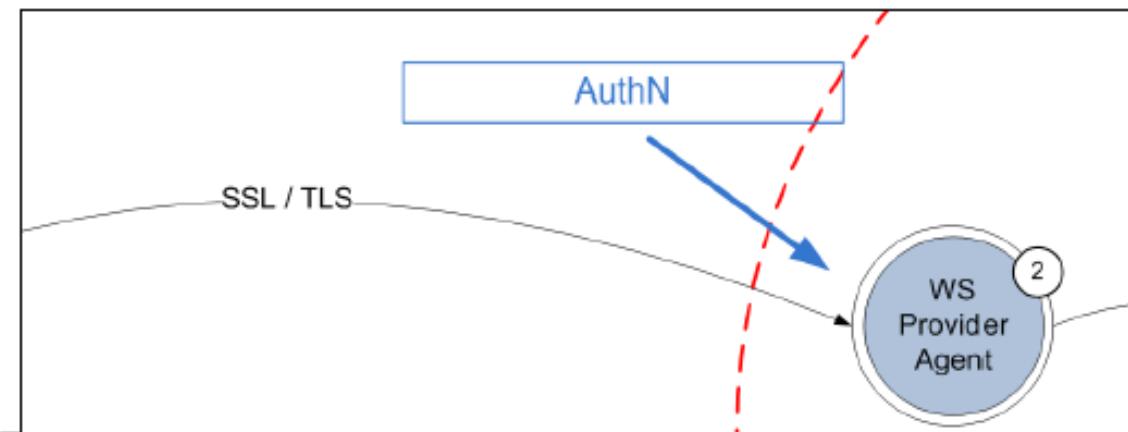
# Chiffrement du transport

SOAP / XML	REST
HTTPS SSL/TLS tunnel SSH IPSEC Etc.	HTTPS

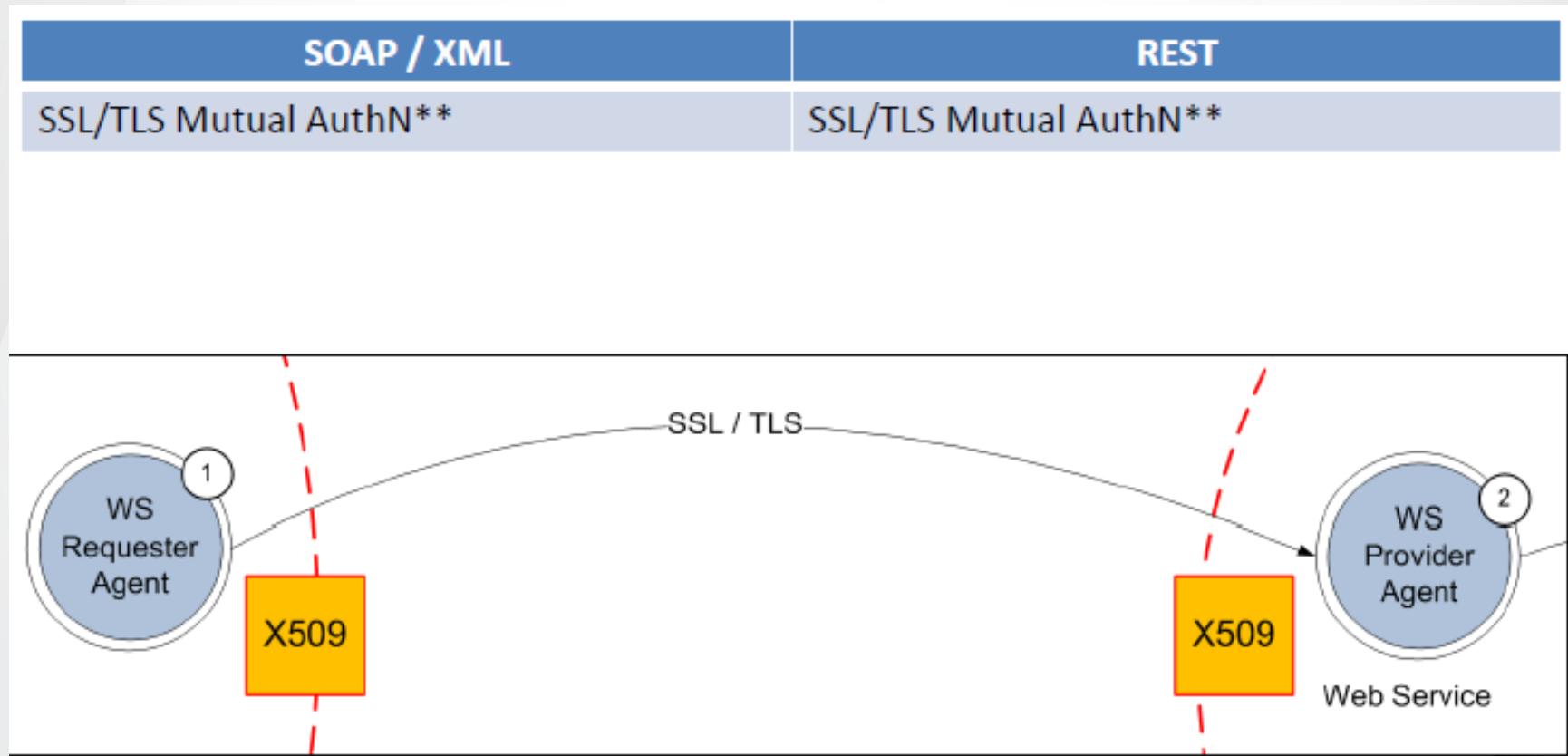
A diagram illustrating the transport encryption process. It shows two circular nodes: 'WS Requester Agent' on the left and 'WS Provider Agent' on the right. A curved line labeled 'SSL / TLS' connects them. Two vertical dashed red lines, numbered 1 and 2, indicate the boundaries of the secure communication channel. The 'Web Service' label is positioned below the provider agent node.

# AuthN

SOAP / XML	REST
HTTP Basic, Digest, HTTP Header Mutual SSL IP trust WS Security user name password WS SAML Authentication token XML Signature Kerberos Etc.	HTTP Basic, Digest, HTTP Header Mutual SSL IP trust Oauth API Keys JSON Web Token (JWT)

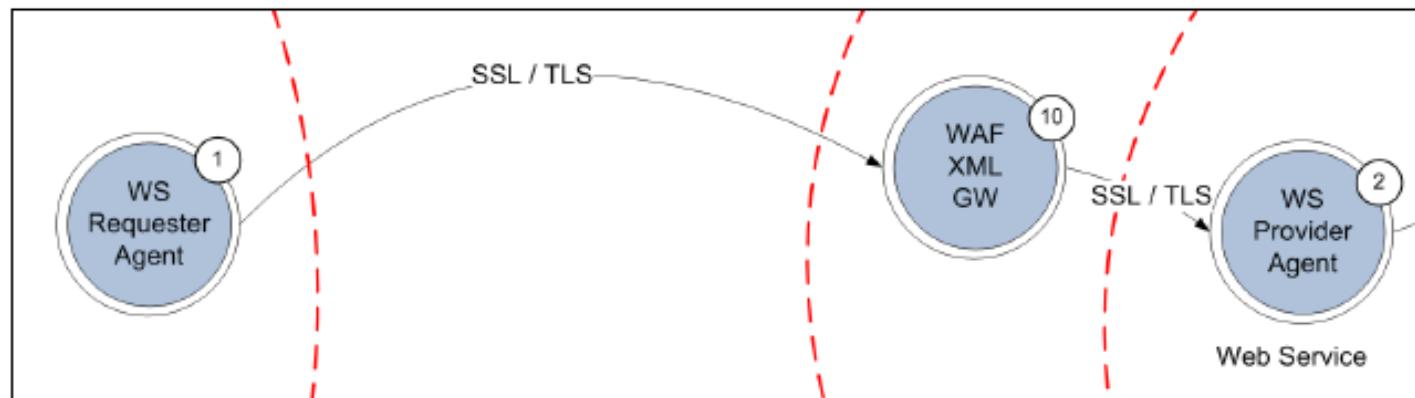


# SSL Mutual AuthN / X509 / PKI



# WAF / XML Gateway (Protection périphérique)

SOAP / XML	REST
Reverse Proxy Contrôle requêtes HTTP Rupture SSL/TLS Black List White List Validation WSDL Signature & Verification Encryption & Decryption SAML	Reverse Proxy Contrôle requêtes HTTP Rupture SSL/TLS Black List White List



# Intégrité et confidentialité des messages

SOAP / XML	REST
XML Signature XML Encryption	<ul style="list-style-type: none"><li>(p.ex: HMAC, Doseta)</li><li>JSON Web Signature (JWS) – Draft v7</li><li>JSON Web Encryption</li></ul>

```
81. POST /resources/rest/geo/comment HTTP/1.1[\r][\n]
82. hmac: jos:+9tn0CLfxXFbzPmbYwq/KYuUSUI=[\r][\n]
83. Date: Mon, 26 Mar 2012 21:34:33 CEST[\r][\n]
84. Content-Md5: r52FDQv6V2GHM4neZBvXLQ==[\r][\n]
85. Content-Length: 69[\r][\n]
86. Content-Type: application/vnd.geo.comment+json; charset=UTF-8[\r][\n]
87. Host: localhost:9000[\r][\n]
88. Connection: Keep-Alive[\r][\n]
89. User-Agent: Apache-HttpClient/4.1.3 (java 1.5)[\r][\n]
[\r][\n]
11. {"comment" : {"message":"blaat", "from":"blaat", "commentFor":123}}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
 <SOAP-ENV:Header><wsse:Security><wsse:UsernameToken wsu:Id="SecurityToken">
 <wsse:Username>user</wsse:Username>
 <wsse:Password>password</wsse:Password>
 <wsse:Nonce>098f6bcd4621d373aa4f32e29b486b1</wsse:Nonce>
 <wsse:Created>2012-03-26T21:34:33Z</wsse:Created>
 </wsse:Security></SOAP-ENV:Header>
 <SOAP-ENV:Body>
 <dss:SignRequest xmlns:dss="http://www.safelayer.com/TLS-Signature-Request-1.0.xsd">
 <dss:OptionalInputs><dss:KeySelector><css:KeySelector>
 <css:KeyIdentifier>urn:uuid:12345678-1234-1234-1234-123456789012</css:KeyIdentifier>
 </css:KeySelector></dss:KeySelector>
 <dss:SignatureAlgorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</dss:SignatureAlgorithm>
 <dss:DigestAlgorithm>http://www.w3.org/2000/09/xmldsig#sha1</dss:DigestAlgorithm>
 </dss:OptionalInputs>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

<http://tools.ietf.org/html/draft-ietf-jose-json-web-signature-07>

# Example XML Signature (SOAP)

```
Demande
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-2

<dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" Id="Id-
tBBsJX0Y1MS81fXsuPEAXR3Grq9F8VvNLpRpVmN+CRTH/ffPof5zSMsrmlwMg9rX
5rZI66D6WG3nqVaZAYX8f33hri46i4wESkPVCWLXwmfid+W84ycwKJ00CLhC7Kvg
UqagXeVWDAbvJEab/NEP9ZSr/c7eB6axLWcd0QbZD9FSn2Y7CBQWNxO9bP7ovYyb
oYrcgY1czNHuZNU88NhClBHqIcXQtmW5Yzcwj0rRKfrw/LGzcM0JlfR4SaaHIIi4y
PxRJuIFC6mbzHgoBvUX+zQ==</dsig:SignatureValue><dsig:KeyInfo Id="Id-00013547
</wsse:Security>
</soap:Header>

<soap:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-w
<ns:echo xmlns:ns="http://ws.apache.org/axis2">
<!-- Zero or one occurrences -->
<ns:args0>a</ns:args0>
</ns:echo>
</soap:Body>
</soap:Envelope>
```

# Exemple JSON “Signature”

## A.1. JWS using HMAC SHA-256

### A.1.1. Encoding

The following example JWS Header declares that the data structure is a JSON Web Token (JWT) **[JWT]** and the JWS Secured Input is se

```
{"typ":"JWT",
 "alg":"HS256"}
```

The following byte array contains the UTF-8 representation of the JWS Header:

```
[123, 34, 116, 121, 112, 34, 58, 34, 74, 87, 84, 34, 44, 13, 10, 32, 34, 97, 108, 103, 34, 58, 34, 72, 83, 50, 53, 54, 34, 125]
```

Base64url encoding these bytes yields this Encoded JWS Header value:

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
```

The JWS Payload used in this example is the bytes of the UTF-8 representation of the JSON object below. (Note that the payload can be a base64url encoded JSON object.)

```
{"iss":"joe",
 "exp":1300819380,
 "http://example.com/is_root":true}
```

The following byte array, which is the UTF-8 representation of the JSON object above, is the JWS Payload:

```
[123, 34, 105, 115, 115, 34, 58, 34, 106, 111, 101, 34, 44, 13, 10, 32, 34, 101, 120, 112, 34, 58, 49, 51, 48, 48, 56, 49, 57, 51, 56, 49, 109, 112, 108, 101, 46, 99, 111, 109, 47, 105, 115, 95, 114, 111, 111, 116, 34, 58, 116, 114, 117, 101, 125]
```

Base64url encoding the above yields the Encoded JWS Payload value (with line breaks for display purposes only):

# Code security

SOAP / XML	REST
<ul style="list-style-type: none"><li>- Data input validation</li><li>- Data output encoding</li><li>- Pseudorandom data generation, high entropy</li><li>- Strong / reliable data encryption algorithms</li><li>- Data leakage prevention</li><li>- Robust error &amp; exception handling</li><li>- Anti-automation and expiration measures</li></ul>	<ul style="list-style-type: none"><li>- Data input validation</li><li>- Data output encoding</li><li>- Pseudorandom data generation, high entropy</li><li>- Strong / reliable data encryption algorithms</li><li>- Data leakage prevention</li><li>- Robust error &amp; exception handling</li><li>- Anti-automation and expiration measures</li></ul>

OWASP Application Security Verification Standard (ASVS):

<https://www.owasp.org/index.php/ASVS>



WASC web application weaknesses:

<http://projects.webappsec.org/w/page/13246978/Threat%20Classification>

# Conclusion

- SOAP:
  - Implémenter les standards WS-\* liés à la sécurité?
  - Mettre en place un filtrage applicatif (WAF, XML GW)
  - Complexe à mettre en oeuvre (PKI, Secure coding, Cryptography, etc.)
  - Architecture à forte contrainte de sécurité
- REST
  - Mettre en place un filtrage applicatif (WAF, XML GW)
  - Implémentation rapide et facile ☐ tendance
  - Architecture de type Cloud, Intranet, Social Login, etc.
  - Emergence des standards (JSON Web Algorithms)
- On attend avec impatience les standards sécu pour REST ???
  - Pragmatique: protection périphérique, chiffrement et Secure Coding ???



# Practice: SOAP & Security

- Création WS SOAP
- Ajout de la couche SSL ( sécurisation de la couche transport)

# Création d'un WS

- Création d'un nouveau Java Project dans Eclipse (WSSecure)
- Ajoutez les deux classes suivantes :

```
package wh.wsformation;

public class TestClass {
 public int x, y;
 public String description;

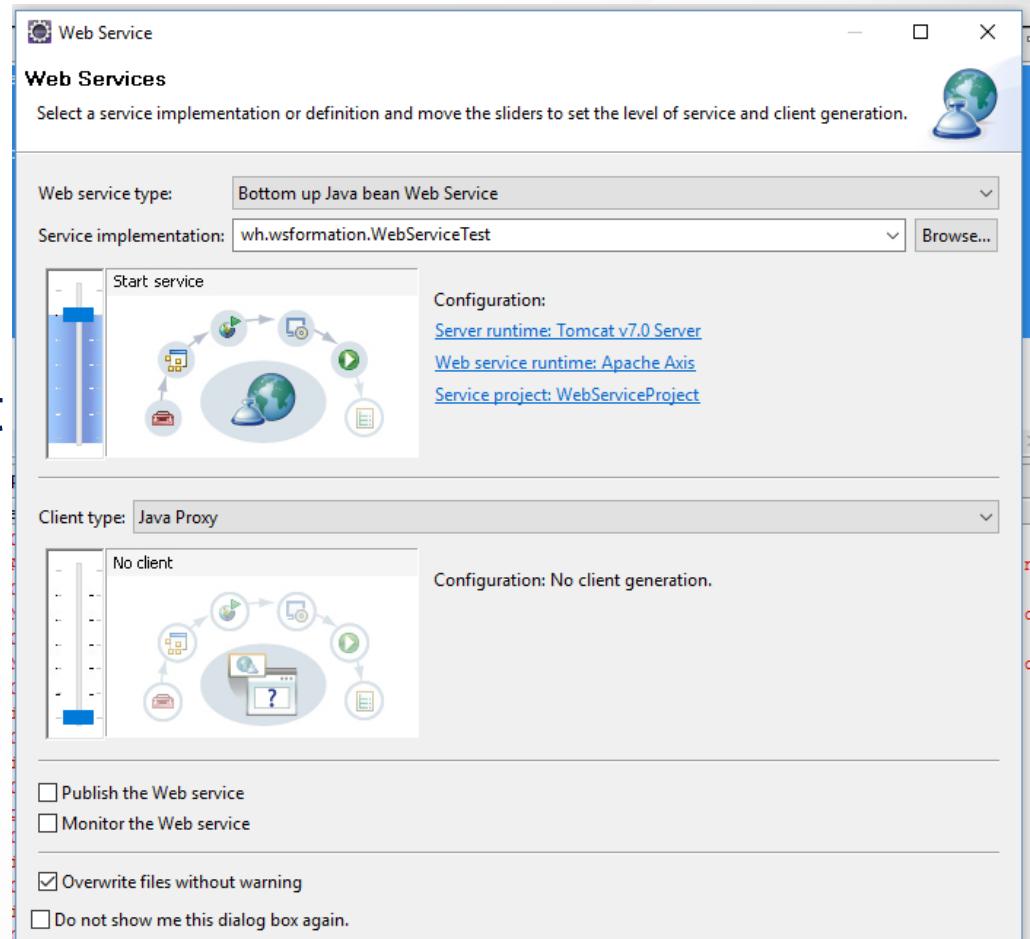
 public TestClass() {
 this.x = this.y = 2;
 this.description = "Description: ";
 }
}
```

- Right Click sur WebServiceTest.java puis choisissez WebService > Create Web Service

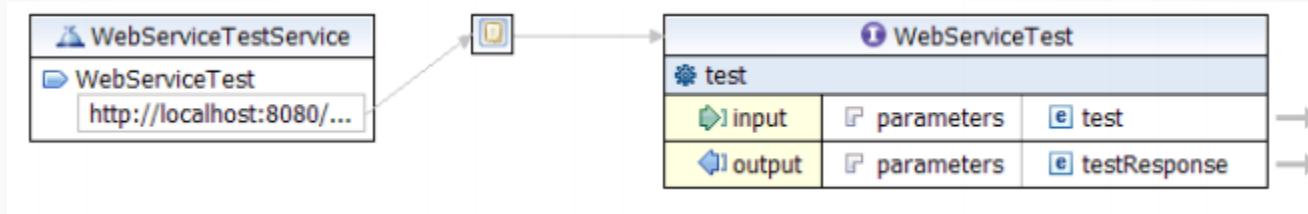
```
package wh.wsformation;

public class WebServiceTest {
 public TestClass test(int x, int y, String d) {
 TestClass tc = new TestClass();
 tc.x *= x;
 tc.y *= y;
 tc.description += d;
 return tc;
 }
}
```

- Dans configuration
  - Server runtime = Tomcat Server
  - Web service runtime : Apache Axis
  - Server Project : current project
- Puis sur : next > finish



- Notez la création du WSDL sous WebContent



- Récupérez le end point du WSDL et collez le dans un navigateur

http://localhost:8080/axis/services/WebServiceTest

## WebServiceTest

Hi there, this is an AXIS service!

Perhaps there will be a form for invoking the service here...

- Testez le webService avec SOAPUI

# Ajout de la couche SSL

---

- Pour crypter les échanges entre clients et serveur, HTTPS utilise une clé publique et une clé privée.
- Tout ce qui est crypté avec la clé privée n'est décryptable qu'avec la clé publique, et inversement.
- La clé publique est diffusée au client. La clé privée reste sur le serveur ( à protéger par l'administrateur , bien qu'elle soit déjà protégée par un mot de passe).

- Un keystore (*magasin de clefs*) Java est un fichier informatique qui stocke des certificats électroniques et éventuellement leurs clefs privées, le contenu de ce fichier sera utilisé par des applications de cryptographie à clef publique comme SSL
  - La JDK propose la commande `keytool` qui permet de manipuler ces fichiers. La commande permet d'ajouter des certificats ou des clefs privées dans un fichier keystore, de les supprimer, d'extraire un certificat, mais jamais d'extraire une clef privée
  - `>keytool -genkey -alias signtelindus -keystore wwwtelindus-keystore`

```
Administrator : Invite de commandes
P:\Java\jdk7\jre\bin>
P:\Java\jdk7\jre\bin>
P:\Java\jdk7\jre\bin>
P:\Java\jdk7\jre\bin>keytool -genkey -alias signtelindus -keystore wwwtelindus-keystore
Quels sont vos nom et prénom ?
[Unknown]: hajji
Quel est le nom de votre unité organisationnelle ?
[Unknown]: formation
Quel est le nom de votre entreprise ?
[Unknown]: telindus
```

```
C:\ Administrateur : Invité de commandes
P:\Java\jdk7\jre\bin>
P:\Java\jdk7\jre\bin>
P:\Java\jdk7\jre\bin>
P:\Java\jdk7\jre\bin>keytool -genkey -alias myalias -password hajji
Quels sont vos nom et prénom ?
[Unknown]: hajji
Quel est le nom de votre unité organisationnelle ?
[Unknown]: formation
Quel est le nom de votre entreprise ?
[Unknown]: telindus
Quel est le nom de votre ville de résidence ?
[Unknown]: paris
Quel est le nom de votre état ou province ?
[Unknown]: IDF
Quel est le code pays à deux lettres pour cette
[Unknown]: FR
Entrez CN=hajji, OU=formation, O=telindus, L=paris
```

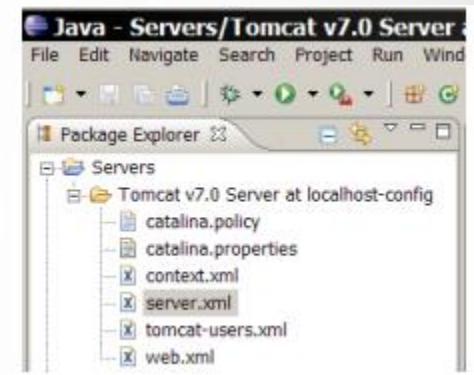
- Une fois le Keystore créé on va s'occuper de Tomcat a travers le fichier server.xml
  - Servers → Tomcat Server → server.xml
- Vous devez définir un connecteur **SSL** , lui fournissant le chemin et le mot de passe pour le fichier de clés que vient d'être créé .

Dans le wsdl on va aussi mettre à jour le end point en  
remplacant :

[http://localhost:8080/....](http://localhost:8080/)

Par

**https://localhost:8443/.....**

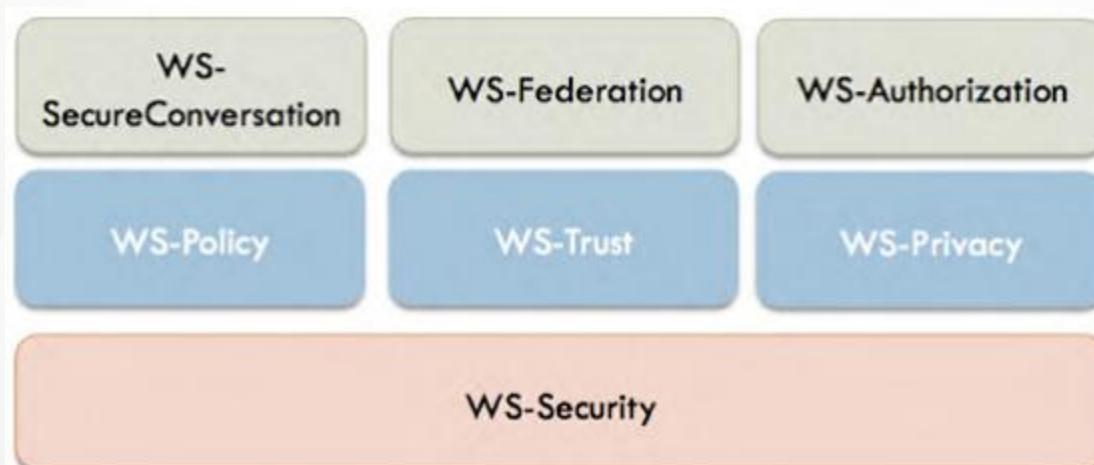


```
<Connector SSLEnabled="true" clientAuth="false"
keystoreFile="P:\Java\wwwtelindus-keystore"
keystorePass="azerty"
maxThreads="200" port="8443"
protocol="org.apache.coyote.http11.Http11NioProtocol"
scheme="https"
secure="true" sslProtocol="TLS"/>
```

- Executer run on server le fichier WSDL
- Dans SOAPUI vous pouvez testez le WS en notant le nouveau URL :
- <https://localhost:8443/WebServiceProject/services/WebServiceTest?wsdl>
- Dans SOAP response affichez SSL Info

# Utilisation de WS - Security

- Objectifs
  - Authentification
  - Confidentialité des messages
  - Intégrité des messages
- Fondation d'autres standards



# WS-Security

- Notion de jeton de sécurité (security token
  - Pour l'authentification ou l'autorisation
    - Ex: username/password, certificat X509, ...
  - Extension de SOAP
    - Définition d'un header SOAP contenant l'information de sécurité
      - Jetons de sécurité
      - Signatures numériques
      - Éléments encryptés

# WS-Security

- Confidentialité des messages SOAP
  - Utilisation de XML-Encryption
    - Encryption d'un ou plusieurs éléments du message SOAP
    - Référence vers les éléments encryptés dans le header
  - Clé partagée
  - Possibilité d'encrypter différents éléments avec des clés différentes

# WS-Policy

- Objectif: spécifier des informations et des exigences pour un WS
  - S'applique aussi bien au serveur qu'au client
- Exemples:
  - utilisation d'une version spécifique de SOAP
  - Exigence de signature
  - Information sur le format de la réponse (encrypté, signée...)

# WS-Trust

- Modèles de confiance nombreux et variés
  - Et transorganisations
- Problèmes
  - Émettre et obtenir des jetons de sécurité
  - Etablir et valider des relations de confiance
- Définition d'un Security Token Service
  - Émet, valide ou échange un jeton de sécurité



Merci de remplir  
l'évaluation de STAGE

# Questions ?

