Java et les Web Services
Objectifs

version 1.3

Objectifs

- Connaître les architectures à base de web services
- Connaître le mode de fonctionnement des web services de type SOAP
- Connaître le mode de fonctionnement des web services de type REST
- Connaître les fichiers utilisé par SOAP
- Savoir consommer les web services
- Savoir coder des web services
  - SOAP et REST

Chapitres

copyleft

Support de formation créé par

Franck SIMON

http://www.franck-simon.com

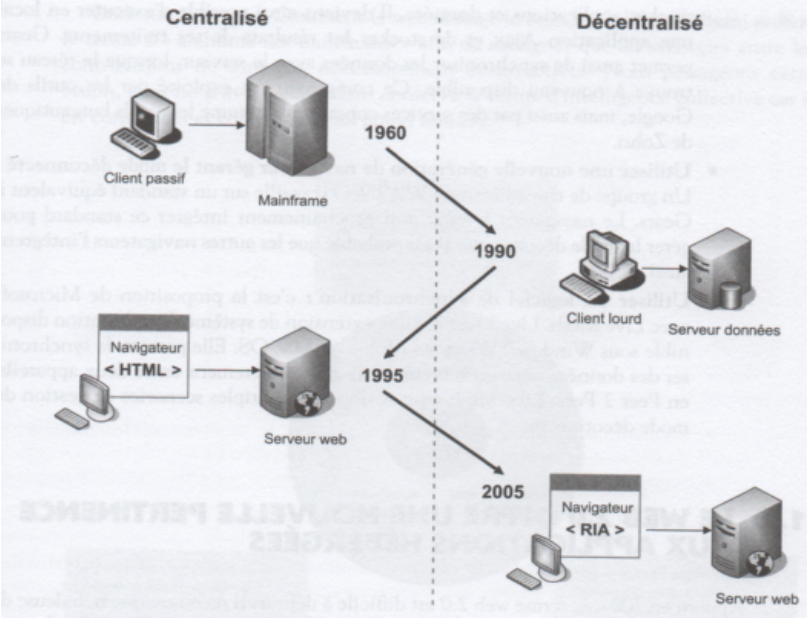
copyleft

Web services
Introduction

Historique

- Les architectures informatiques suivent un cycle régulier de centralisation / décentralisation
  - 1960 : mainframes
  - 1990 : architectures client/serveur
    - protocoles de communication type CORBA
  - 1995 : explosion du web
  - 2005 : RIA

Historique

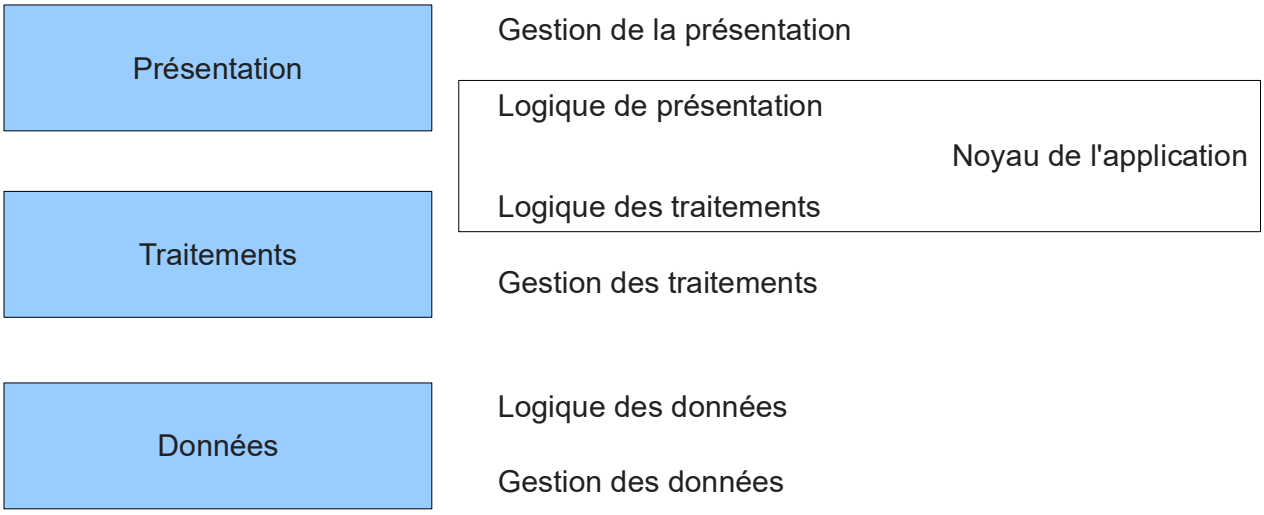


source : Cloud computing et SaaS - Dunod

Évolution des architectures

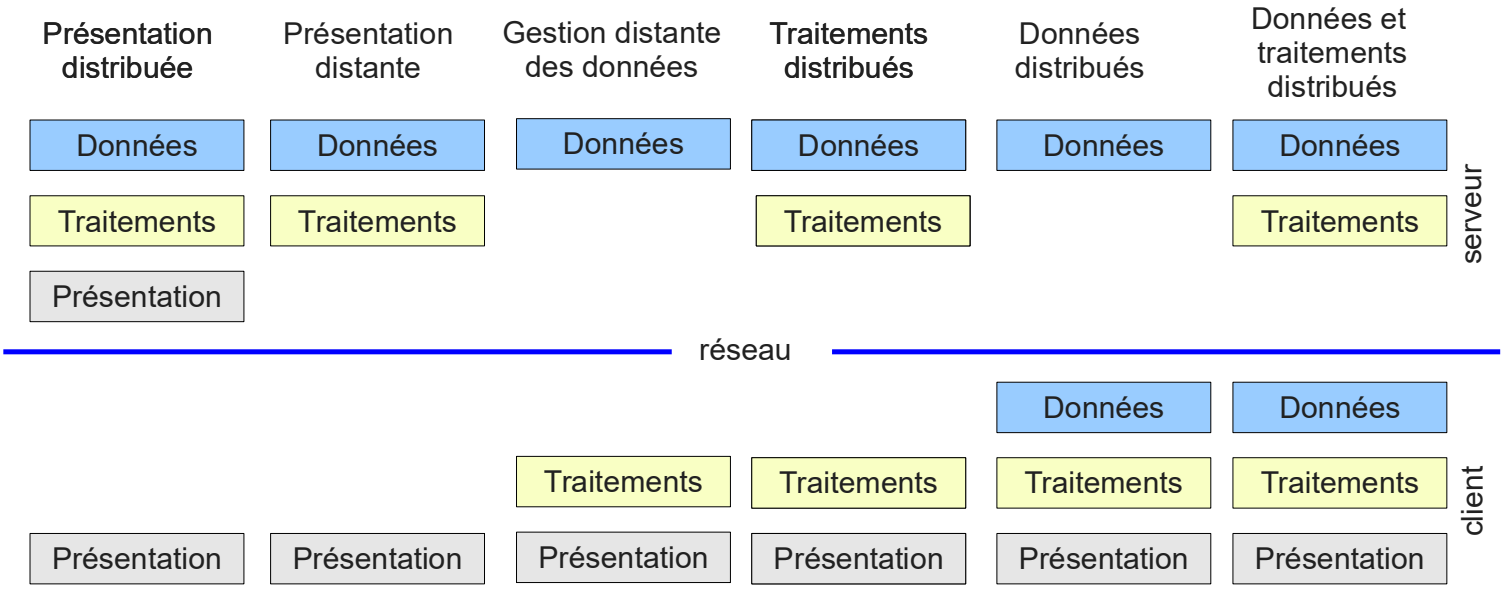
- Classiquement une application peut être divisée en trois niveaux
  - la couche de présentation
    - IHM, GUI
  - la couche applicative
    - les traitements
  - l'accès aux données
- Ce principe n'est qu'un découpage abstrait
  - les trois couches peuvent être imbriquées ou réparties

Évolution des architectures



Modèles de répartition

- Gartner Group classifie les différents modèles de répartition



Modèles de répartition

- Le client serveur est un modèle 2 tiers
- Le découpage précédent illustre sous forme simplifiée la structure possible d'une application
  - la réalité est plus complexe
    - architectures multi-niveaux
      - le serveur peut-être client d'un autre serveur
    - prise en compte des applications existantes
      - legacy

Architecture trois-tiers

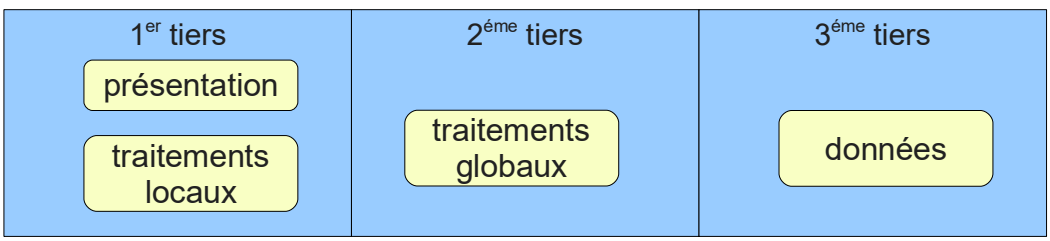
- Les limites de l'architecture deux-tiers
  - frontal complexe et non standard
    - généralement sous Windows
    - à déployer
  - middleware non standard
- La solution :
  - utilisation d'un poste client très simple
  - communication avec le serveur via un protocole standard

Architecture trois-tiers

- Principes de base
  - les données sont gérées de manière centralisée
  - la présentation est gérée par le poste client
  - la logique applicative est gérée par un serveur intermédiaire
- Premières tentatives
  - introduction d'un serveur d'application centralisé
    - dialogue avec les clients par un protocole propriétaire

Architecture trois tiers

- Répartition des traitements
  - 1er tiers : affichage et traitement locaux
    - contrôles de surface, mises en forme des données, …
  - 2ème tiers : traitements applicatifs globaux pris en charge par le serveur d'application
  - 3ème tiers : base de données

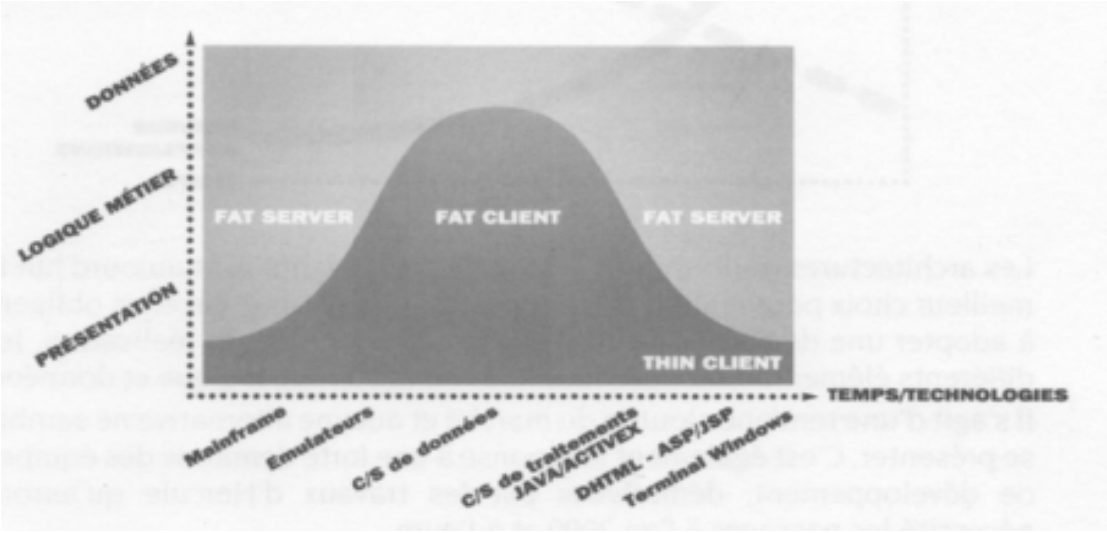


Architecture trois tiers

- Internet révolutionne l'architecture
  - corrige les excès du client lourd
    - la partie applicative est centralisée sur le serveur
- Le serveur HTTP devient central
  - problème de dimensionnement de serveur
  - gestion de la montée en charge
  - complexité de la maintenance des applications
  - gestion des sessions
  - le serveur est fortement sollicité

Architecture trois tiers

- Du mainframe en mode texte au mainframe en mode graphique : retour à la case départ



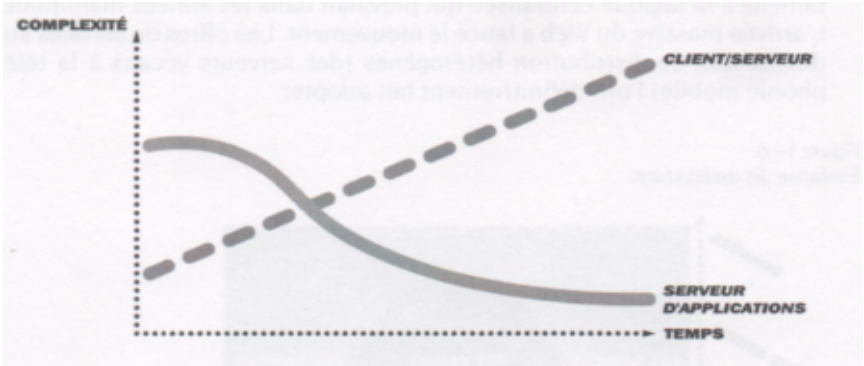
source : Serveurs d'applications - Eyrolles

Architectures N-tiers

- Permet de pallier aux limitations du 3 tiers
  - distribution plus libre de la logique applicative
  - répartissions de la charge
- N tiers pour la distribution de l'application entre de multiples services
  - et non pas la multiplication des niveaux de service
- Utilise des composants
  - chaque composant rend un service clairement identifié
  - concepts orientés objets

Architectures N-tiers

- Complexité de réutilisation



source : Serveurs d'applications - Eyrolles

Architecture N-tiers

- Solution Oracle / Sun
  - EJB : Enterprise Java Bean
    - s'exécutent côté serveur
  - JavaBean (POJO)
    - objets métiers
- Solution Microsoft
  - modèle de communication COM
  - DCOM étend COM pour les architectures distribuées

Serveur transactionnel

- Issu des technologies IBM des années 1970
  - mise à disposition à grande échelle d'applications en mode texte
  - plusieurs écrans se succèdent avant qu'une modification soit réellement effectuée
- Le serveur héberge un moteur transactionnel
  - met en relation le client avec un ensemble de serveurs de données

WOA

- Web Oriented Architecture
- Implémentation SOA qui utilise le web comme support de service
- Tous les services doivent être exposés sur le web
  - problème potentiel de performance

ORB

- Object Request Broker
  - appartient à la famille des middlewares
- Bibliothèques de fonctions implémentant un bus logiciel
  - les objets communiquent de manière transparente sur le réseau
    - invocation à distance de la méthode d'un objet
- Deux ORB peuvent communiquer via IIOP
  - Internet Inter-ORB Protocol

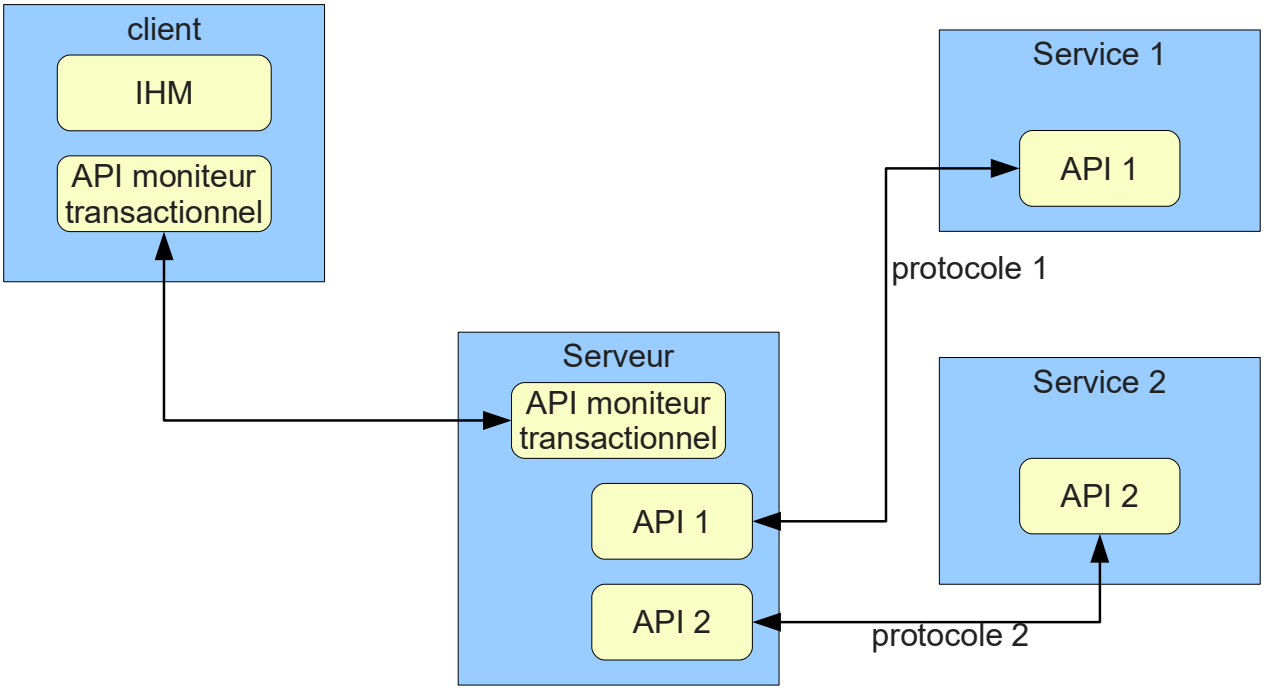
MOM

- Message Oriented Middleware
- Architecture permettant l'échange de message entre applications via le réseau
  - permet un couplage faible entre applications
- Deux modes de fonctionnement
  - point à point
  - par abonnement

Serveur transactionnel

- permet de garantir la règle ACID
  - Atomicité
    - la transaction ne peut pas être partiellement effectuée
  - Cohérence
    - la transaction fait passer la base d'un état cohérent à un autre état cohérent
  - Isolation
    - un transaction n'est pas affectée par le résultat des autres transactions
  - Durée
    - les modifications de la transaction sont durablement garanties

Serveur transactionnel



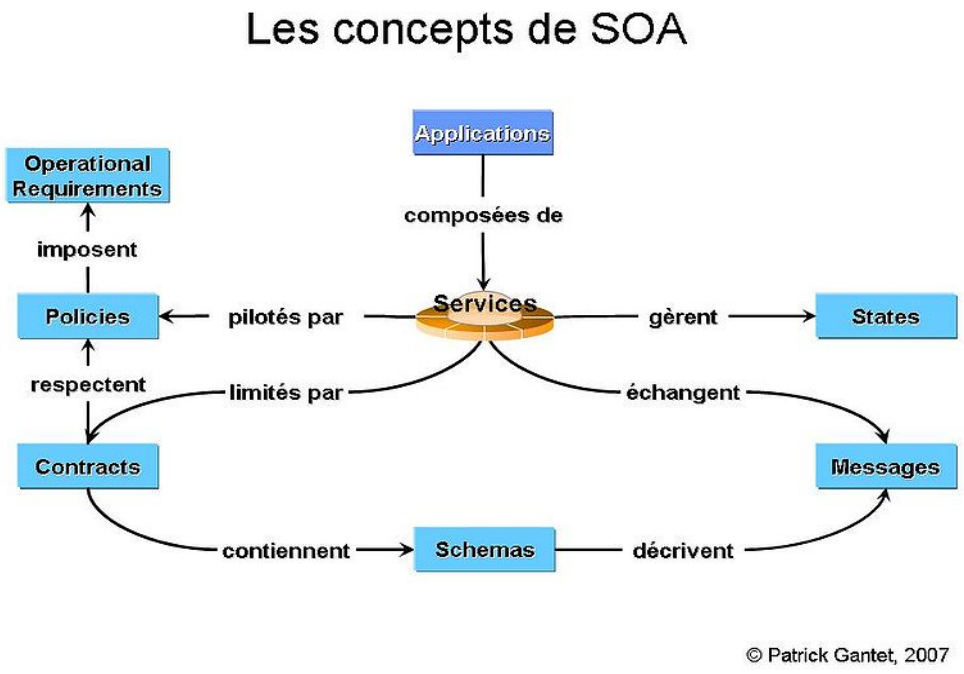
EAI

- Enterprise Application Integration
- Architecture permettant à des applications hétérogènes d'échanger des messages
  - EAI va gérer les flux inter-applicatifs
    - notion de workflow
      - le middleware ne fait que véhiculer les flux entre les applications
    - prend en charge la traduction des données entre les applications

SOA

- Service Oriented Architecture
- Architecture de médiation
  - les services sont des composants logiciels
- Popularisé avec l'utilisation des web services
  - commerce électronique, B2B, B2C, …
  - souvent basé sur les plateformes .Net ou Java EE
- Consiste en une collection de services qui interagissent et communiques entre eux

SOA



source : Wikipedia Commons

SOA et ESB

- Enterprise Service Bus
- L'implémentation de SOA est basée sur un bus de services
- ESB est une évolution des EAI (Enterprise Application Integration
  - ESB propose une intégration distribuée via l'utilisation de conteneurs de services
    - interfaces normalisées : SOAP, JMS, ...

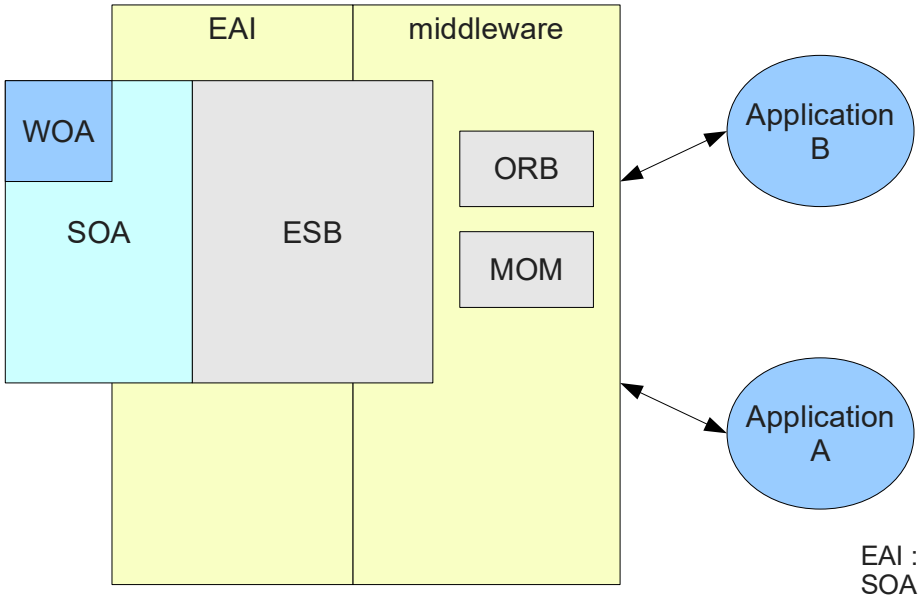
Web service

- Web service
  - service permettant l'échange de données entre systèmes et applications hétérogènes
  - ensemble de fonctionnalités exposées sur Internet ou sur un intranet
  - le web service est donc avant tout un service accessible via le réseau internet
    - via le protocole HTTP (ou HTTPS)
    - ne présuppose en rien d'une technologie ou d'un protocole particulier

Web service

- Approche technologiques possibles
  - web services de type WS-*
    - reposent sur l'utilisation de SOAP et WSDL
      - WS-* pour l'ensemble des spécifications utilisées
  - web services de type REST
    - Representational State Transfert
    - repose sur l'utilisation des méthodes HTTP

En résumé



EAI : Enterprise Application Integration
SOA : Service Oriented Architecture
WOA : Web Oriented Architecture
ESB : Entreprise Service Bus
ORB : Object Request Broker
MOM : Message Oriented Midddleware

SOAP
Introduction

SOAP

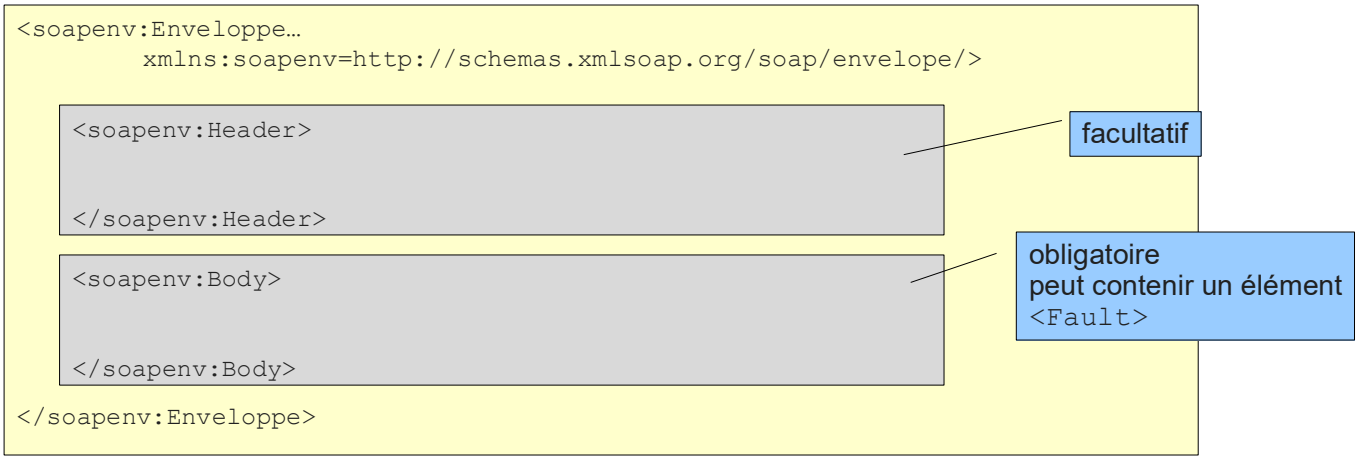
- SOAP
  - protocole de communication
  - communication entre applications
  - format d'échange de messages
  - indépendant de tout langage
  - basé sur XML
  - recommandation du W3C

SOAP

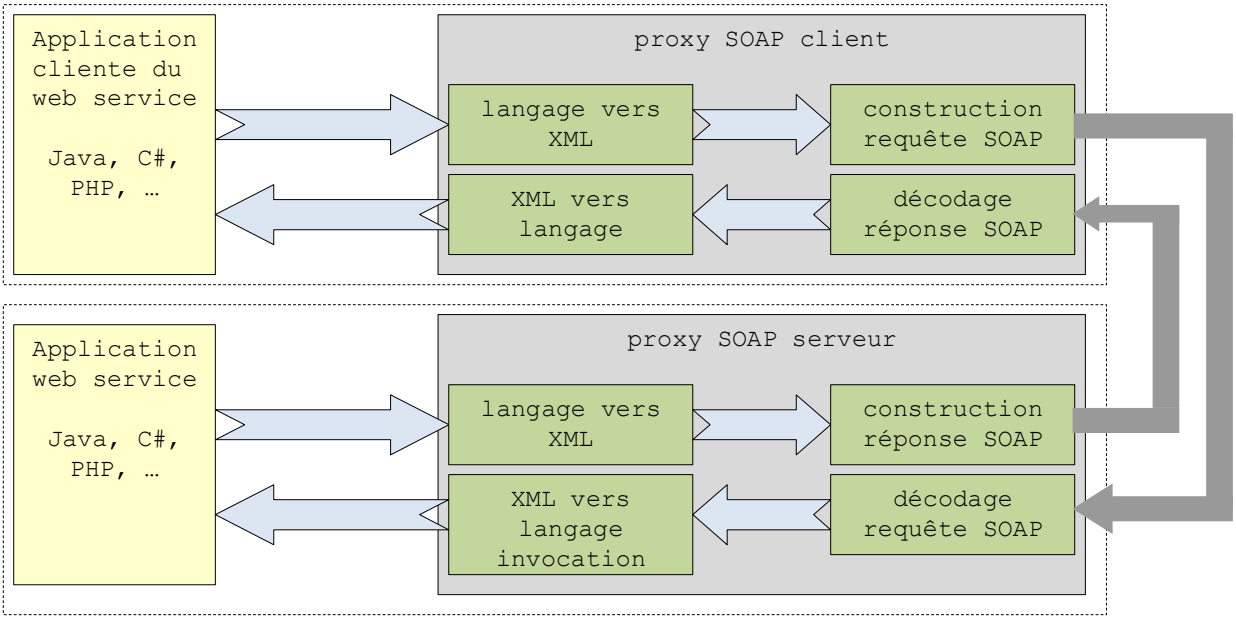
- SOAP est un document XML composé
  - d'une enveloppe `<Enveloppe>` identifiant le document XML comme un message SOAP
  - d'une en-tête facultative `<Header>`
  - d'un corps de message obligatoire `<Body>` contenant les informations de requête ou de réponse
    - qui peut contenir un éventuellement élément `<Fault>` contenant la description des erreurs
  - de la description du type d'encodage utilisé pour la sérialisation et la dé-sérialisation des données
    - attribut `encodingStyle`

SOAP

- L'enveloppe SOAP est elle-même incluse dans la partie body du protocole de transmission
  - HTTP par exemple



SOAP



SOAP

- SOAP définit plusieurs méthodes pour envoyer en XML les données échangées entre les applications
  - RPC : Remote Procedure Call
    - appel de méthode
    - le corps du message SOAP contient le nom de la méthode à invoquer
      - et les paramètres envoyés
  - Document (message)
    - échange de messages
    - pas de règle de format du corps du message SOAP

SOAP

- SOAP définit aussi le type d'encodage pour la sérialisation et la dé-sérialisation
  - `encoded`
    - règle d'encodage définie par SOAP 1.1
      - sous la référence "section 5 encoding"
    - spécifie comment les objets, structures, tableaux, graphes sont sérialisés
  - `literal`
    - les données sont sérialisées en accord avec un schéma XML
      - aucune règle prédéfinie
      - en pratique utilisation de la spécification du W3C XMLSchema

SOAP RPC

- Approche aisée pour le développement
  - car utilise par défaut le mode encoding
- Il s'agit d'un appel d'une méthode distante en passant tous les paramètres nécessaires
  - le proxy SOAP client sérialise les paramètres et l'appel vers le format XML
  - le transport des informations est effectué via HTTP
  - le proxy SOAP serveur dé-sérialise les paramètres de l'appel et invoque la méthode dans le langage natif

SOAP RPC

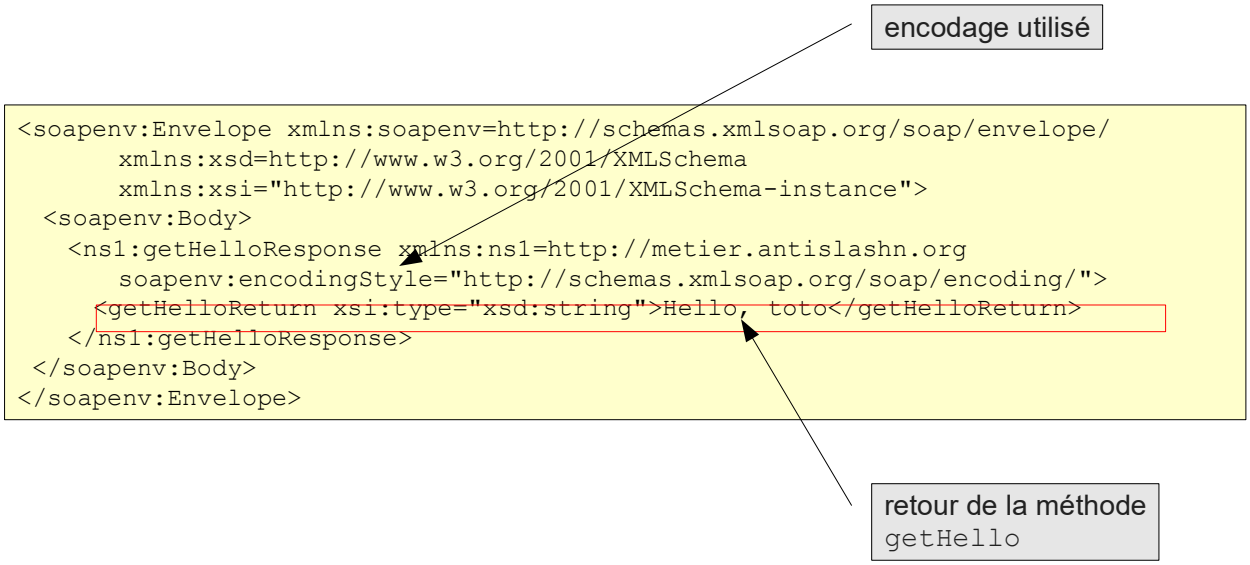
- La valeur de retour de la méthode invoquée est prise en charge par la couche SOAP serveur
  - même mécanisme que pour l'appel de la méthode
    - sérialisation par la couche SOAP serveur, et dé-sérialisation par la partie SOAP client
  - SOAP – RPC autorise aussi l'encodage littéral
    - envoie d'une partie d'arbre XML par exemple
    - la couche SOAP n'a alors qu'un paramètre à sérialiser

SOAP RPC

- Requête SOAP-RPC

SOAP-RPC

- Réponse SOAP-RPC



SOAP-Document

- Le consortium W3C préconise l'utilisation de SOAP Document
- La couche SOAP envoie le document complet au serveur sans attendre le résultat de retour
- Le message peut contenir n'importe quel type de données XML
- En mode Document le développeur peut choisir :
  - le mode de transport : HTTP, SMTP, MOM, …
  - la sérialisation
  - format de l'enveloppe SOAP
  - en pratique les proxys fournissent un mode fonctionnement par défaut

SOAP-Document

- L'élément `<Body>` contient une ou plusieurs parties
  - il n'y a pas de règle de format sur ce que le `<Body>` doit contenir
    - la seule règle est l'accord entre le client et le web service sur le contenu
    - les frameworks type Axis rendent transparente l'utilisation du mode Document
      - les proxys client et serveur sont générés symétriquement par le framework

SOAP-Document

- Requête SOAP-Document

appel de la méthode `getHello`

```
<soapenv:Envelope xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
       xmlns:q0=http://metier.antislashn.org
       xmlns:xsd=http://www.w3.org/2001/XMLSchema
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <q0:getHello>
      <q0:nom>toto</q0:nom>
    </q0:getHello>
  </soapenv:Body>
</soapenv:Envelope>
```

passage du paramètre
pas de précision sur l'encodage

SOAP-Document

- Réponse SOAP-Document

```
<soapenv:Envelope xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
       xmlns:xsd=http://www.w3.org/2001/XMLSchema
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <getHelloResponse xmlns="http://metier.antislashn.org">
      <getHelloReturn>Hello, toto</getHelloReturn>
    </getHelloResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

retour de la méthode `getHello`
pas d'encodage précisé

SOAP : retour d'erreur

- En cas d'erreur, la réponse peut contenir des erreurs HTTP ou SOAP
  - Une erreur SOAP contient l'élément `<Fault>` qui contient :
    - `<Code>` code d'erreur (obligatoire)
    - `<Reason>` explication sur l'erreur (obligatoire)
    - `<Node>` nœud SOAP source de l'erreur (facultatif)
    - `<Role>` rôle du nœud SOAP (facultatif)
    - `<Detail>` informations supplémentaires (facultatif)

SOAP : gestion des attachements

- Une URI peut être précisée
- Un flux binaire codé en base 64
- Utilisation de SOAP Message with Attachment
  - MIME pour web services
  - transmet les flux par SOAP en utilisant MIME/Multipart
- Utilisation de WS-Attachments
- Utilisation de XOP (XML-Binary Optimized Packaging)

WSDL

- WSDL pour Web Service Description Language
  - document XML
  - décrit un web service
  - localise le web service
  - spécification du consortium W3C

WSDL : structure

- Un Web service est décrit par les éléments principaux :
  - `<types>` : types de données du web service
  - `<message>` : les messages utilisés par le web service
    - en entrée et en sortie, avec précision des paramètres
  - `<portType>` : interface, opérations abstraites proposées par le web service
  - `<binding>` : liaison avec l'implémentation concrète du service, protocoles et formats d'échange
  - `<service>` : adresse du service, le plus souvent une URL invoquant un service SOAP
    - comporte un ensemble de ports (endpoints)

WSDL : `<portType>`

- Définit l'interface du web service
- Les opérations qui sont prises en charge par le web service
  - les messages qui participent à l'invocation complète de l'opération
  - une opération peut être comparée à une fonction
- C'est le point d'entrée du web service
- Peut être comparé à une librairie de fonctions

WSDL : `<portType>`

- Types d'opérations applicables
  - `One-way` : opération recevant un message et ne retournant pas de réponse
  - `Request-response` : opération recevant une requête et retournant une réponse
    - la plus courante
  - `Solicit-response` : opération pouvant envoyer une requête et attendre une réponse
  - `Notification` : opération pouvant envoyer un message et n'attendant pas de réponse

WSDL : `<binding>`

- Définit le format et le protocole du message
  - `<binding>` a deux attributs
    - `name` : nom du binding
    - `type` : port du binding
  - `<soap:binding>` liaison avec l'implémentation
    - `style` : format (`rpc` ou `document`)
    - `transport` : protocole utilisé
  - `<operation>` définit les opérations exposées
    - pour chaque opération une action SOAP est définie

WSDL

- En général les WSDL sont générés par les frameworks, ou les outils de développement
  - à partir de Java EE 5 et en utilisant les annotations le serveur génère le WSDL
- Le WSDL décrit l'utilisation du web service, il est donc différent selon le SOAP utilisé
  - `rpc` ou `document`
  - `encoded` ou `literal`

WSDL pour SOAP-RPC

- Extrait `<message>`

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace=http://metier.antislashn.org
       xmlns:apachesoap=http://xml.apache.org/xml-soap
       xmlns:impl=http://metier.antislashn.org
       xmlns:intf=http://metier.antislashn.org
       xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
       xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
       xmlns:wsdlsoap=http://schemas.xmlsoap.org/wsdl/soap/
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:message name="getHelloRequest">
    <wsdl:part name="nom" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getHelloResponse">
    <wsdl:part name="getHelloReturn" type="xsd:string" />
  </wsdl:message>
  …
  …
</wsdl:definitions>
```

type d'encodage

déclaration des messages et des paramètres

WSDL pour SOAP-RPC

- Extrait `<portType>`

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace=http://metier.antislashn.org
       xmlns:apachesoap=http://xml.apache.org/xml-soap
       xmlns:impl=http://metier.antislashn.org
       xmlns:intf=http://metier.antislashn.org
       xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
       xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
       xmlns:wsdlsoap=http://schemas.xmlsoap.org/wsdl/soap/
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  …
  <wsdl:portType name="Hello">
    <wsdl:operation name="getHello" parameterOrder="nom">
      <wsdl:input message="impl:getHelloRequest" name="getHelloRequest" />
      <wsdl:output message="impl:getHelloResponse" name="getHelloResponse" />
    </wsdl:operation>
  </wsdl:portType>
  …
</wsdl:definitions>
```

description des opérations disponibles sur le web service

WSDL pour SOAP-RPC

- Extrait `<binding>`

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace=http://metier.antislashn.org
       …
  <wsdl:binding name="HelloSoapBinding" type="impl:Hello">
    <wsdlsoap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getHello">
      <wsdlsoap:operation soapAction="" />
      <wsdl:input name="getHelloRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://metier.antislashn.org" use="encoded" />
      </wsdl:input>
      <wsdl:output name="getHelloResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://metier.antislashn.org" use="encoded" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  …
</wsdl:definitions>
```

encodage utilisé

WSDL pour SOAP-RPC

- Extrait `<service>`

lien avec le binding

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace=http://metier.antislashn.org
       …
  <wsdl:service name="HelloService">
    <wsdl:port binding="impl:HelloSoapBinding" name="Hello">
      <wsdlsoap:address location="http://localhost:8080/hellows/services/Hello" />
    </wsdl:port>
  </wsdl:service>

…
</wsdl:definitions>
```

url du web service

WSDL pour SOAP-Document

- Extrait `<types>`

```
<wsdl:definitions targetNamespace=http://metier.antislashn.org
       xmlns:apachesoap=http://xml.apache.org/xml-soap
       xmlns:impl=http://metier.antislashn.org
       xmlns:intf=http://metier.antislashn.org
       xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
       xmlns:wsdlsoap=http://schemas.xmlsoap.org/wsdl/soap/
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>

     …

  </wsdl:types>

…
</wsdl:definitions>
```

déclaration des types utilisés
cf. slide suivant

WSDL pour SOAP-Document

- Extrait `<types>`

```
…
   <schema elementFormDefault="qualified"
        targetNamespace=http://metier.antislashn.org
        xmlns="http://www.w3.org/2001/XMLSchema">
     <element name="getHello">
       <complexType>
         <sequence>
         <element name="nom" type="xsd:string" />
         </sequence>
       </complexType>
     </element>
     <element name="getHelloResponse">
       <complexType>
         <sequence>
         <element name="getHelloReturn" type="xsd:string" />
         </sequence>
       </complexType>
     </element>
   </schema>
…
```

utilisation des types spécifiés par XMLSchema

WSDL pour SOAP-Document

- Extrait `<message>`

```
<wsdl:definitions targetNamespace=http://metier.antislashn.org
       xmlns:apachesoap=http://xml.apache.org/xml-soap
       xmlns:impl=http://metier.antislashn.org
       xmlns:intf=http://metier.antislashn.org
       xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
       xmlns:wsdlsoap=http://schemas.xmlsoap.org/wsdl/soap/
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
…
  <wsdl:message name="getHelloRequest">
    <wsdl:part element="impl:getHello" name="parameters" />
  </wsdl:message>
  <wsdl:message name="getHelloResponse">
    <wsdl:part element="impl:getHelloResponse" name="parameters" />
  </wsdl:message>

…
</wsdl:definitions>
```

WSDL pour SOAP-Document

- Extrait `<portType>`

```
<wsdl:definitions targetNamespace=http://metier.antislashn.org
       xmlns:apachesoap=http://xml.apache.org/xml-soap
       xmlns:impl=http://metier.antislashn.org
       xmlns:intf=http://metier.antislashn.org
       xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
       xmlns:wsdlsoap=http://schemas.xmlsoap.org/wsdl/soap/
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
…
  <wsdl:portType name="Hello">
    <wsdl:operation name="getHello">
      <wsdl:input message="impl:getHelloRequest" name="getHelloRequest" />
      <wsdl:output message="impl:getHelloResponse" name="getHelloResponse" />
    </wsdl:operation>
  </wsdl:portType>
…
</wsdl:definitions>
```


WSDL pour SOAP-Document

- Extrait `<binding>`

```
<wsdl:definitions targetNamespace=http://metier.antislashn.org
  …
    <wsdl:binding name="HelloSoapBinding" type="impl:Hello">
    <wsdlsoap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getHello">
      <wsdlsoap:operation soapAction="" />
      <wsdl:input name="getHelloRequest">
        <wsdlsoap:body use="literal" />
      </wsdl:input>
      <wsdl:output name="getHelloResponse">
        <wsdlsoap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
…
</wsdl:definitions>
```

encodage utilisé

Développement d'un web service

- En règle général, nous pouvons partir
  - du WSDL pour créer le code
    - méthodes de classe ou fonction
  - du code pour créer le WSDL
- Les frameworks automatisent le développement
  - génération du WSDL
  - génération des classes proxy côtés serveur et client
  - utilisation des annotations Java EE

Développer des web services
SOAP avec Java EE

Web services avec JAX-WS

- Java API for XML Web Services
  - ensemble d'API pour construire des web services utilisant SOAP
    - orienté message et orienté RPC (Remote Procedure Call)
  - cache la complexité de la traduction java ↔ SOAP
  - peut aussi être utilisé par la partie consommatrice de web services

Création d'un web service

- Annotation `@WebService`
  - package `javax.jws`
- SEI est une interface ou classe Java
  - Service Endpoint Interface : interface
  - Service Endpoint Implementation : classe
  - déclare les méthodes qui sont invocables par le client

Création d'un web service

- Étapes principales pour créer un web service
  - coder la classe du web service
  - compiler cette classe
  - packager dans une archive WAR
  - déployer l'archive
    - les structures du web service sont créés par le serveur
  - coder la classe du client
  - utiliser la tâche Ant wsimport

Création d'un web service

- Étapes principales pour créer un client
  - coder la classe du client
  - utiliser la tâche Ant `wsimport`
    - génération et compilation de la structure nécessaire pour ce connecter au web service
  - compiler les classes

Caractéristique d'un JAX-WS endpoint

- La classe doit être annotée par `@WebService` ou `@WebServiceProvider`
  - `@WebServiceProvider` permet l'implémentation sans stub, ni WSDL
    - permet de créer des web services dynamiquement
  - la classe d'implémentation
    - doit posséder un constructeur par défaut
    - peut utiliser `@PostConstruct` et `@PreDestroy`
    - ne doit pas être `final`, ni `abstract`
  - les méthodes exposées doivent être publiques
    - les paramètres et le retour doivent être compatible avec JAXB

Principales annotations

- `javax.jws.WebService`
  - marque une classe Java implémentant un web service
  - propriétés
    - `name` : `wsdl:portType` - par défaut le nom qualifié de la classe
    - `targetNameSpace` : espace du nom XML du WSDL
    - `serviceName` : `wsdl:service` - nom du web service, par défaut nom simple de la classe + *Service*
    - `endpointInterface` : interface endpoint du service
    - `portName` : `wsd:portName` - par défaut WebService.name + *Port*
    - `wsdlLocation` : adresse du document WSDL

Principales annotations

- `javax.jws.WebMethod`
  - indique une méthode correspondant à une opération de web service
  - propriétés
    - `operationName` : `wsdl:operation` - nom de la méthode, par défaut nom de la méthode Java
    - `exclude` : indique si la méthode doit être exclue du web service, `false` par défaut
    - `action` : définit la valeur de l'en-tête `SOAPAction`, par défaut le nom de la méthode

Autres annotations

- `javax.jws.WebParam`
  - personnalise le mappage d'un paramètre individuel vers un élément XML
- `javax.jws.OneWay`
  - opération unidirectionnelle d'un web service
    - un message en entrée, aucun message en sortie
- `javax.jws.WebResult`
  - personnalise le mappage d'une valeur de retour vers une partie WSDL ou un élément XML
- Se référer à la Javadoc pour les autres annotations

RESTful

REST

- REpresentational State Transfert
  - architecture créée en 2000 par Roy Fielding
- Architecture
  - client-serveur, sans état
  - interface uniforme, 4 règles
    - chaque ressource est identifiée de manière unique (URI)
    - les ressources ont des représentations définies
      - les méta-données permettent au client de modifier l'état de la ressource
    - message auto-descriptif
    - moteur d'état hypermédia

RESTful

- Le marketing met en avant les web services de type REST et RESTful
- Les web services RESTful utilisent de manière explicite les méthodes HTTP
  - GET pour récupérer une ressource
  - POST pour créer une ressource
  - PUT pour modifier une ressource
  - DELETE pour supprimer une ressource

RESTful

- Des en-têtes de requêtes HTTP plus explicites
  - avec une application web classique

```
GET /adduser?name=Toto HTTP/1.1
```

  - avec RESTful

```
POST /users HTTP/1.1
Host : localhost
Content-Type: application/xml
<?xml version="1.0"?>
<user>
    <name>Toto</name>
</user>
```

RESTful

- Des URIs plus propres
  - plus intuitive

```
http://my.domain.org/discussion/topics/java
```

```
http://my.domain.org/discussion/2011/12/23/java
```

  - qui peuvent être analysées

```
http://my.domain.org/discussion/topics/{topic}
```

```
http://my.domain.org/discussion/{year}/{month}/{day}/{topic}
```


RESTful

- Règles pour la structure d'une URI d'un web service RESTful
  - cacher la technologie utilisée par le serveur
    - pas d'extension .jsp, .php, etc.
  - tout en minuscule
  - les espaces sont remplacés par des - ou _
  - évite les requêtes SQL dans les URL
  - toujours fournir une page par défaut
    - à l'instar du code 404 Not Found

RESTful

- Les transferts s'effectuent généralement en
  - XML - POX (Plain Old XML)
    - type MIME : application/xml
  - JSON (JavaScript Object Notation)
    - type MIME : application/json
  - XHTML
    - type MIME : application/xhtml+xml
- D'autres format existent
  - texte pur, YAML, ...

RESTful

- Exemples POX et JSON

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<identite>
    <civilite>M</civilite>
    <prenom>Gaston</prenom>
    <nom>LAGAFFE</nom>
    <adresse>
        <rue>Rue de Bruxelles</rue>
        <ville>Paris</ville>
        <code-postal>75000</code-postal>
    </adresse>
</identite>
```

```
{
    "civilite":"M",
    "prenom":"Gaston",
    "nom":"LAGAFFE",
    "adresse":{
        "rue":"Rue de Bruxelles",
        "ville":"Paris",
        "codePostal":"75000"
    }
}
```

JAXB - Sérialisation XML et Java

- Le passage Java ↔ XML est grandement facilité par l'API JAXB
  - Java Architecture for XML Binding
  - facilite la manipulation des documents XML
    - avec JAXP (SAX et DOM) le traitement des données XML est à coder
  - analyse le schema XML
  - génère un ensemble de classes

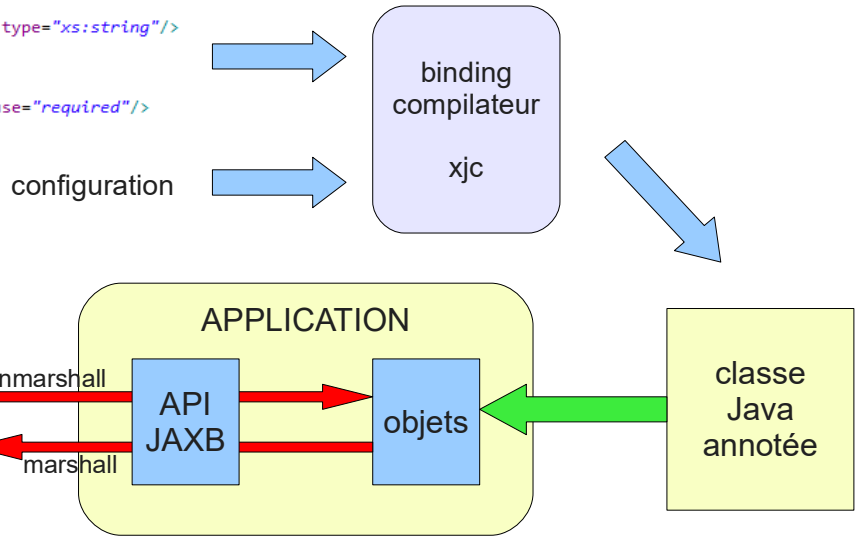
JAXB - Sérialisation XML et Java

- Deux étapes principales
  - génération des classes et interfaces à partir du schéma XML
  - utilisation des classes générées pour transformer un document XML en graphe d'objets, ou inversement
  - Spring rend transparent ces étapes

Utilisation de JAXB

XML Schema

```
<xs:element name="entreprise">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="raison_sociale" type="xs:string"/>
            <xs:element ref="adresse"/>
            <xs:element name="web"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
</xs:element>
```



document XML

```
<entreprise id="e2">
    <raison_sociale>Le bateau ivre</raison_sociale>
    <adresse type="pro">
        <rue>3, rue des Vagues</rue>
        <code_postal>35000</code_postal>
        <ville>RENNES</ville>
        <pays>FRANCE</pays>
    </adresse>
    <web>www.le-bateau-ivre.bzh</web>
</entreprise>
```


JAXB

- Les classes doivent être annotées pour pouvoir être sérialisée dans un fichier XML
  - une classe doit être `@XmlRootElement`
  - cf. documentation pour ensemble des annotations
- Un fichier de configuration peut être ajouté

```
<?xml version="1.0" encoding="UTF-8"?>
<bindings xmlns="http://java.sun.com/xml/ns/jaxb"
     xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/jaxb
                  http://java.sun.com/xml/ns/jaxb/bindingschema_2_0.xsd"
     version="2.1">
          <schemaBindings>
               <package name="org.antislashn.jaxb" />
          </schemaBindings>
</bindings>
```

Java et les web services
RESTful avec Java EE

Création d'un RESTful

- JAX-RS : Java API for RESTful web Services
  - simplifie la création des web services RESTful
  - un simple classe comme implémentation
  - un jeu d'annotations
    - package javax.ws.rs
  - automatise un grand nombre de conversions
    - JSON ↔ Java
    - XML ↔ Java

Principales annotations

- `@Path`
  - URI relative, au niveau de la classe et des méthodes
  - peut contenir des paramètres entre accolades `{}`
- `@GET`, `@POST`, `@PUT`, `@DELETE`, `@HEAD`
  - annotations de mappage avec les méthodes HTTP
  - au niveau des méthodes
- `@PathParam`
  - permet d'extraire des paramètres du path pour l'injecter dans un paramètre de méthode

Principales annotations

- `@Consumes`
  - type MIME envoyé par le client
- `@Produces`
  - type MIME retourné au client
- `@QueryParam`
  - permet d'extraire des paramètres de la requête pour l'injecter dans un paramètre de méthode

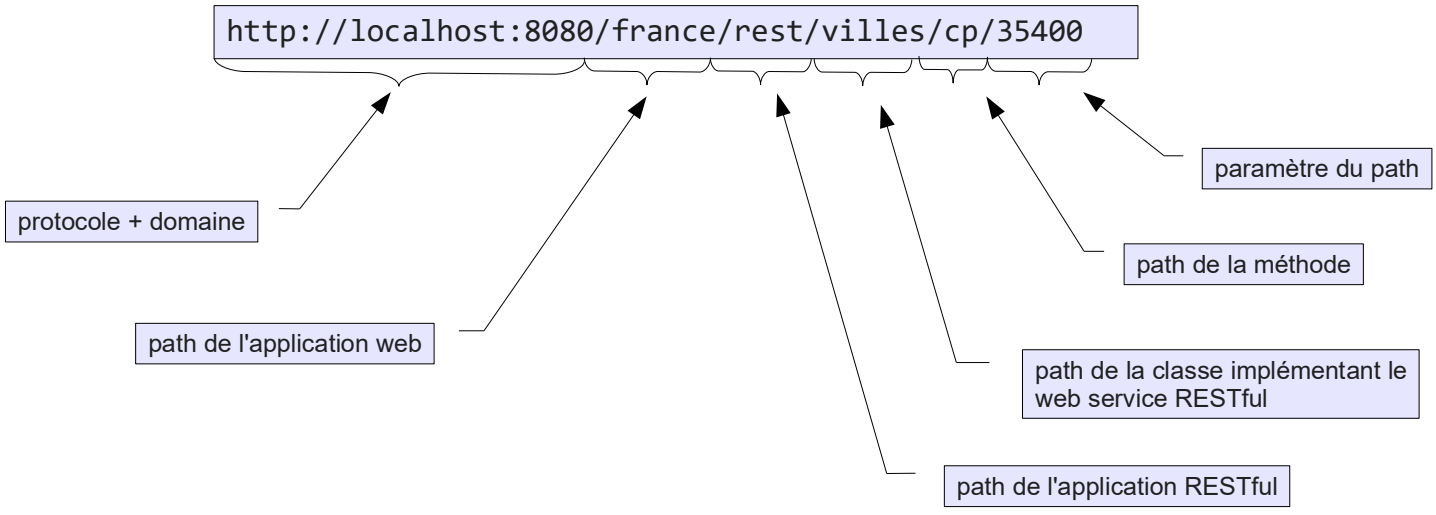
Création d'un web service RESTful

- `javax.ws.rs.core.Application`
  - permet d'ajouter des informations supplémentaires au service RESTful
  - le développeur doit implémenter cette classe
  - par exemple ajout d'un chemin pour tous les web services RESTful
    - annotation `javax.ws.rs.ApplicationPath`

```
@ApplicationPath("/rest")
public class FranceConfig extends Application {
}
```


Création d'un web service RESTful

- Exemple



Création d'un web service RESTful

- Exemple simple