



Création et utilisation d'applications client/serveur avec JEE

Le royaume des services Web

Plan du cours

- Partie I** Qu'est-ce qu'un service Web ?
- Partie II** Utiliser et tester un service Web existant
- Partie III** Création d'un premier service Web SOAP
- Partie IV** Réalisation d'un client de conversion de monnaie
- Partie V** Réalisation d'un client de gestion d'adresse IP
- Partie VI** Développer des services Web REST avec JAVA
- Partie VII** Projet



PARTIE I

Qu'est-ce qu'un service Web ?

Qu'est-ce qu'un service Web ?

La technologie des services Web est un moyen rapide de distribution de l'information entre clients, fournisseurs, partenaires commerciaux et leurs différentes plates-formes. Les services Web sont basés sur le modèle SOA .

D'autres technologies telles que RMI, DCOM et CORBA ont précédemment adopté ce style architectural mais ont généralement échoué en raison de la diversité des plates-formes utilisées dans les organisations et aussi parce que leur usage n'était pas adapté à Internet (problème de passage à travers des FireWalls, etc.) d'où la lenteur, voire l'absence de réponses sur ce réseau. Les applications réparties fondées sur ces technologies offrent des solutions caractérisées par un couplage fort entre les objets. Les solutions proposées par les services Web, permettent néanmoins un couplage moins fort.

De plus, l'utilisation des technologies standards du Web telles HTTP et XML par les services Web facilite le développement

d'applications réparties sur Internet, et permet d'avoir des applications très faiblement couplées. L'intégration est sans doute le facteur essentiel qui favorise l'utilisation des services Web.

Qu'est-ce qu'un service Web ?

Un service Web est tout simplement un programme accessible au moyen d'Internet, qui utilise un système de messagerie standard XML, et **n'est lié à aucun système d'exploitation ou langage de programmation** !

En reprenant la définition du consortium W3C, voici les principaux avantages d'un service Web, à savoir :

- son interface décrite d'une manière interprétable par les machines, qui permet aux applications clientes d'accéder aux services de manière automatique ;
- son utilisation de langages et protocoles indépendants des plates-formes d'implantation, qui renforcent l'interopérabilité entre services ;
- son utilisation des normes actuelles du Web, qui permettent la réalisation des interactions faiblement couplées et favorisent aussi l'interopérabilité.

L'intérêt d'un Service Web

Les services Web fournissent un lien entre applications. Ainsi, des applications utilisant des technologies différentes peuvent envoyer et recevoir des données au travers de protocoles compréhensibles par tout le monde.

Les services Web sont normalisés car ils utilisent les standards XML et HTTP pour transférer des données et ils sont compatibles avec de nombreux autres environnements de développement. **Ils sont donc indépendants des plateformes.**

C'est dans ce contexte qu'un intérêt très particulier a été attribué à la conception des services Web puisqu'ils permettent aux entreprises d'offrir des applications accessibles à distance par d'autres entreprises. Cela s'explique par le fait que **les services Web n'imposent pas de modèles de programmation spécifiques.**

Les services Web représentent donc la façon la plus efficace de partager des méthodes et des fonctionnalités. De plus, ils réduisent le temps de réalisation en permettant de tirer directement parti de services existants.

Caractéristiques d'un service Web

Un service Web possède les caractéristiques suivantes :

- il est accessible via le réseau ;
- il dispose d'une interface publique (ensemble d'opérations) décrite en XML ;
- ses descriptions (fonctionnalités, comment l'invoquer et où le trouver ?) sont stockées dans un annuaire ;
- il communique en utilisant des messages XML, ces messages sont transportés par des protocoles Internet (généralement HTTP, mais rien n'empêche d'utiliser d'autres protocoles de transfert tels : SMTP, FTP...) ;
- l'intégration d'application en implémentant des services Web produit des systèmes faiblement couplés, le demandeur du service ne connaît pas forcément le fournisseur. Ce dernier peut disparaître sans perturber l'application cliente qui trouvera un autre fournisseur en cherchant dans l'annuaire.

Architecture d'un service Web

Les services Web reprennent la plupart des idées et des principes du Web (HTTP, XML), et les appliquent à des interactions entre machines. Comme pour le World Wide Web, les services Web communiquent via un ensemble de technologies fondamentales qui partagent une architecture commune. Ils ont été conçus pour être réalisés sur de nombreux systèmes développés et déployés de façon indépendante. Les technologies utilisées par les services Web sont HTTP, WSDL, REST, XML-RPC, SOAP et UDDI.

SOAP

SOAP (Simple object Access Protocol) est un protocole standard de communication. C'est l'épine dorsale du système d'interopérabilité. SOAP est un protocole décrit en XML et standardisé par le W3C. Il se présente comme une enveloppe pouvant être signée et pouvant contenir des données ou des pièces jointes. Il circule sur le protocole HTTP et permet d'effectuer des appels de méthodes à distance.

REST

REST (Representational State Transfer) est une architecture de services Web. Élaborée en l'an 2000 par Roy Fielding, l'un des créateurs du protocole HTTP, du serveur Apache HTTPd et d'autres travaux fondamentaux, REST est une manière de construire une application pour les systèmes distribués comme le World Wide Web.

Architecture d'un service Web

XML-RPC

XML-RPC est un protocole simple utilisant XML pour effectuer des messages RPC. Les requêtes sont écrites en XML et envoyées via HTTP POST. Les requêtes sont intégrées dans le corps de la réponse HTTP.

WSDL

WSDL (Web Services Description Language) est un langage de description standard. C'est l'interface présentée aux utilisateurs. Il indique comment utiliser le service Web et comment interagir avec lui. WSDL est basé sur XML et permet de décrire de façon précise les détails concernant le service Web tels que les protocoles, les ports utilisés, les opérations pouvant être effectuées, les formats des messages d'entrée et de sortie et les exceptions pouvant être envoyées.

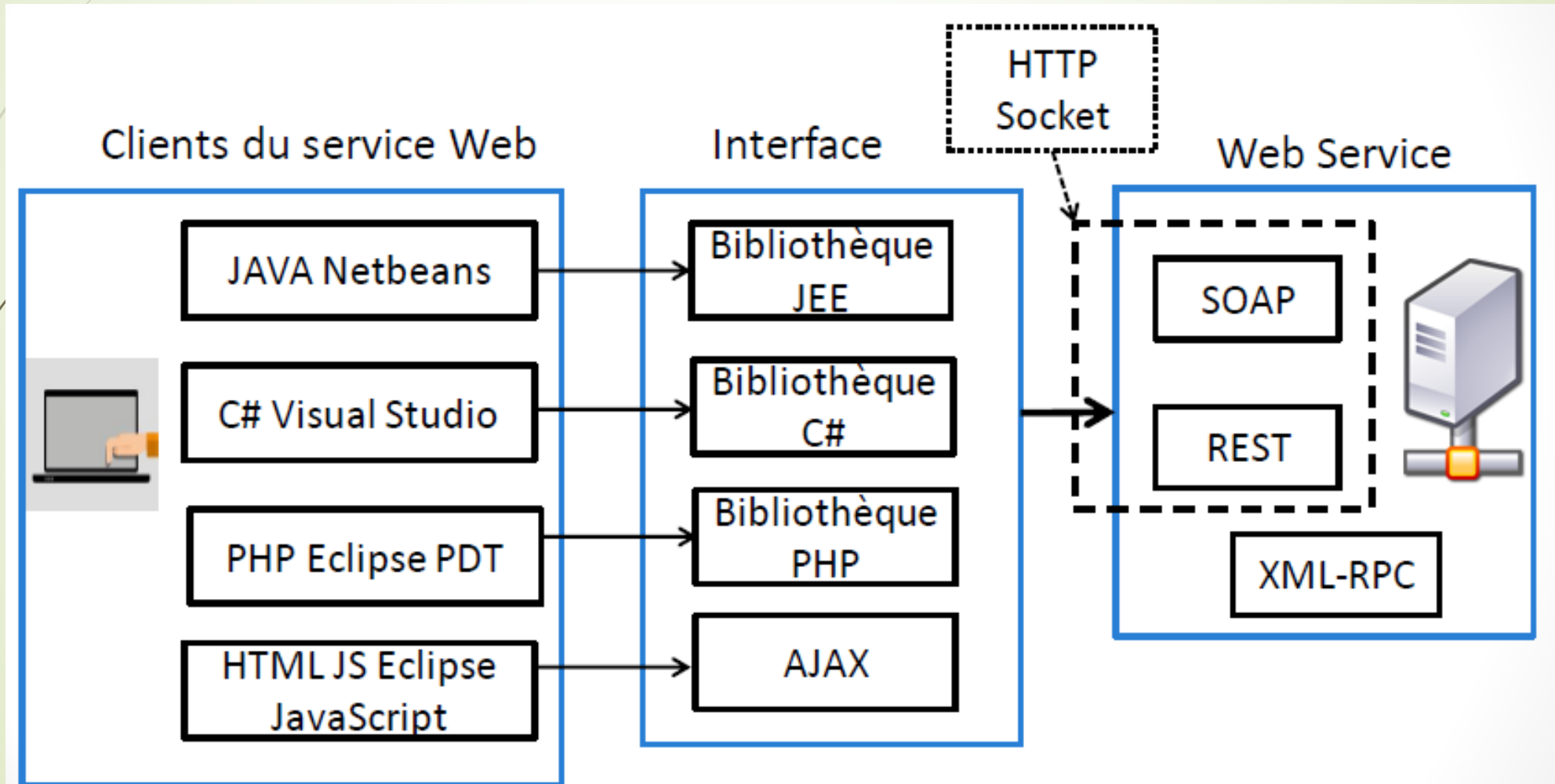
UDDI

UDDI (Universal Description, Discovery and Integration) est un annuaire de services. Il fournit l'infrastructure de base pour la publication et la découverte des services Web. UDDI permet aux fournisseurs de présenter leurs services Web aux clients.

Les informations qu'il contient peuvent être séparées en trois types :

- les pages blanches qui incluent l'adresse, le contact et les identifiants relatifs au service Web ;
- les pages jaunes qui identifient les secteurs d'affaires relatifs au service Web ;
- les pages vertes qui donnent les informations techniques.

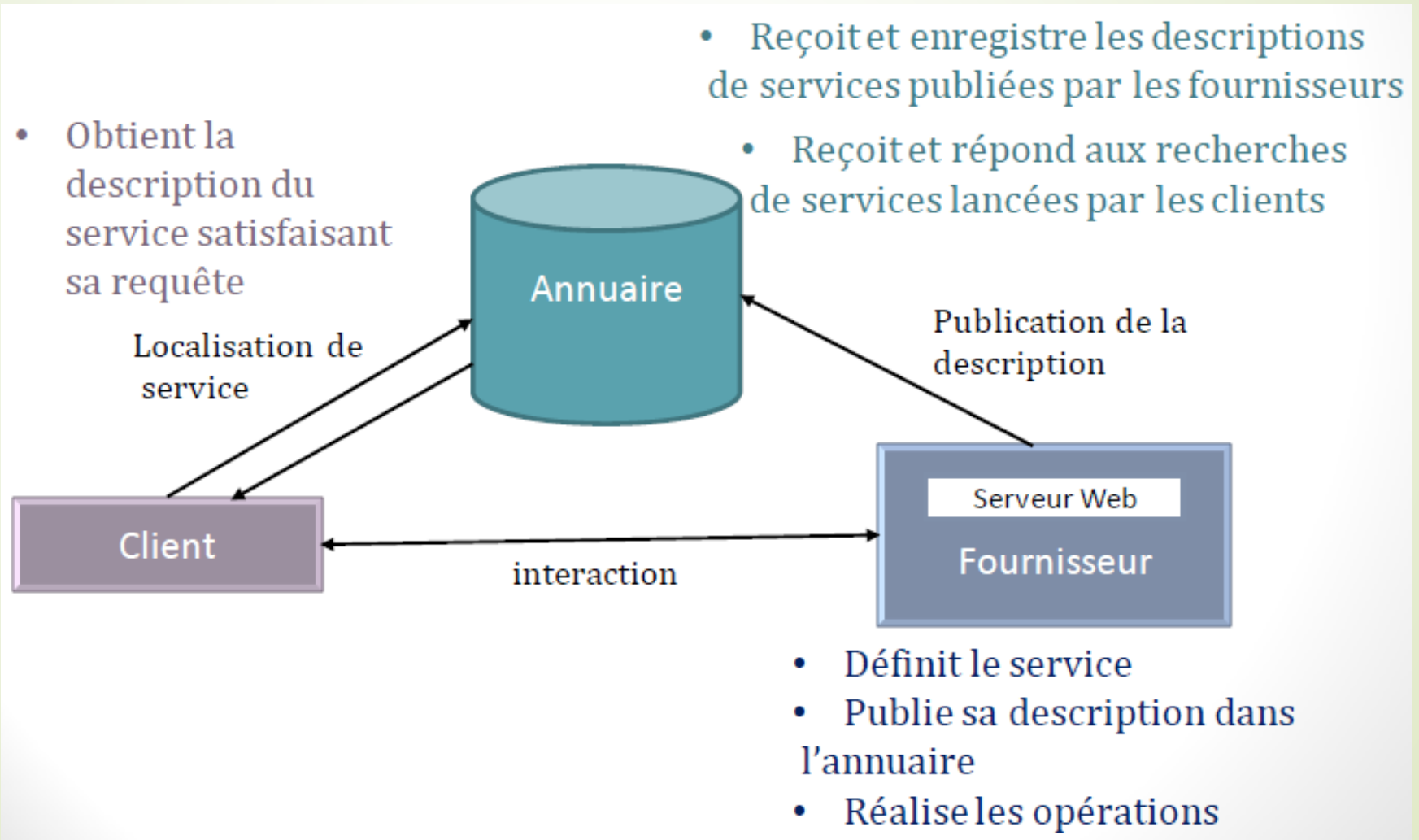
Architecture générale des WS et clients



Architecture Orientée Service (SOA)

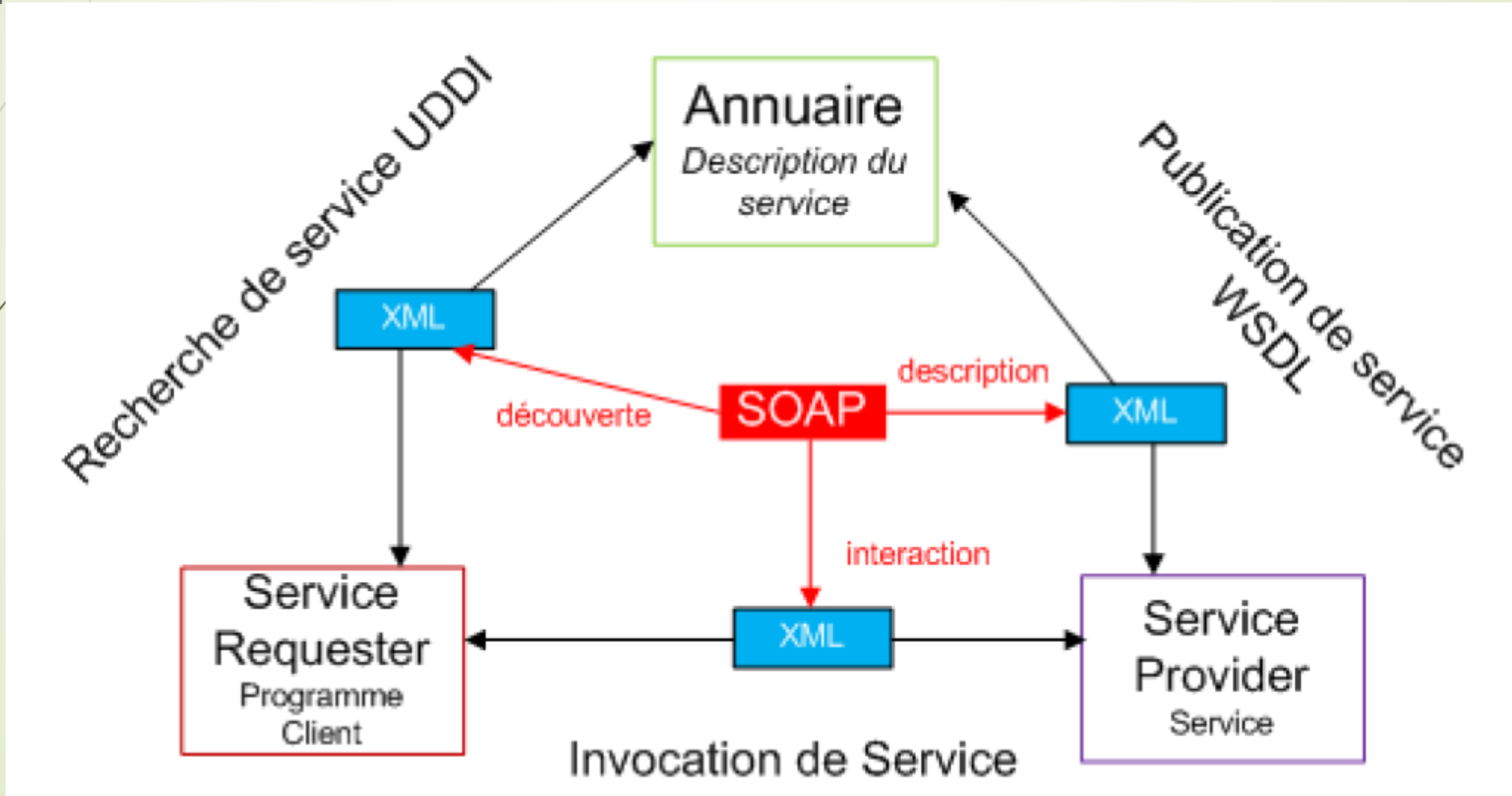
Le fonctionnement des services Web s'articule autour de trois acteurs principaux :

CLIENT – ANNUAIRE - FOURNISSEUR



Fonctionnement des services Web

Le fonctionnement des services Web s'articule autour de trois acteurs principaux illustrés par le schéma suivant :



Description en couche des services Web

Les services Web emploient un ensemble de technologies qui ont été conçues afin de respecter une structure en couches sans être dépendante de façon excessive de la pile des protocoles. Cette structure est formée de quatre couches majeures :

- Découverte de services **UDDI**
- Description de services **WSDL**
- Communication **SOAP**
- Transport **HTTP**

Couches technologiques des services Web.

- Le transport de messages XML-RPC ou SOAP est assuré par le standard HTTP.
- SOAP ou XML-RPC prévoit la couche de communication basée sur XML pour accéder à des services Web.
- La description d'un service Web se fait en utilisant le langage WSDL. WSDL expose l'interface du service.
- La publication et la découverte des services Web sont assurées par le biais du référentiel UDDI. Un référentiel UDDI est un catalogue de services Web.

Description en couche des services Web

Couche transport : Cette couche est responsable du transport des messages XML échangés entre les applications. Actuellement, cette couche inclut HTTP, SMTP, FTP, BEEP

Couche communication : Cette couche est responsable du formatage des données échangées de sorte que les messages peuvent être compris à chaque extrémité. Actuellement, deux styles architecturaux totalement différents sont utilisés pour ces échanges de données. Nous avons d'un côté l'architecture orientée opérations distribuées (protocoles RPC) basée sur XML et qui comprend XML-RPC et SOAP et de l'autre côté une architecture orientée ressources Web, REST (Representational State Transfer) qui se base uniquement sur le bon usage des principes du Web (en particulier, le protocole HTTP).

Couche description de service : Cette couche est responsable de la description de l'interface publique du service Web. Le langage utilisé pour décrire un service Web est WSDL qui est la notation standard basée sur XML pour construire la description de l'interface d'un service. Cette spécification définit une grammaire XML pour décrire les services Web comme des ensembles de points finaux de communication (ports) à travers lesquels on effectue l'échange de messages.

Couche publication de service : Cette couche est chargée de centraliser les services dans un registre commun, et de simplifier les fonctionnalités de recherche et de publication des services Web. Actuellement, la découverte des services est assurée par un annuaire UDDI (Universal Description, Discovery, and Integration).

Le protocole de communication SOAP

SOAP est bien plus populaire et utilisé que XML-RPC. C'est une recommandation du W3C. D'après cette recommandation, SOAP est destiné à être un protocole léger dont le but est d'échanger des informations structurées dans un environnement décentralisé et distribué. Une des volontés du W3C vis-à-vis de SOAP est de ne pas réinventer une nouvelle technologie.

SOAP a été construit pour pouvoir être aisément porté sur toutes les plates-formes et les technologies existantes.

Qu'est-ce que le SOAP ?

Beaucoup de définitions normalisées de SOAP ont été proposées. Une particulièrement intéressante définit SOAP comme étant une spécification pour une omniprésence, basée sur XML et sur des infrastructures distribuées.



Le protocole de communication SOAP

Spécification car SOAP est un document qui définit le modèle de communication. L'idée de base est que si les deux parties ont créé des programmes de mêmes spécifications, ils seront en mesure d'interagir de façon transparente.

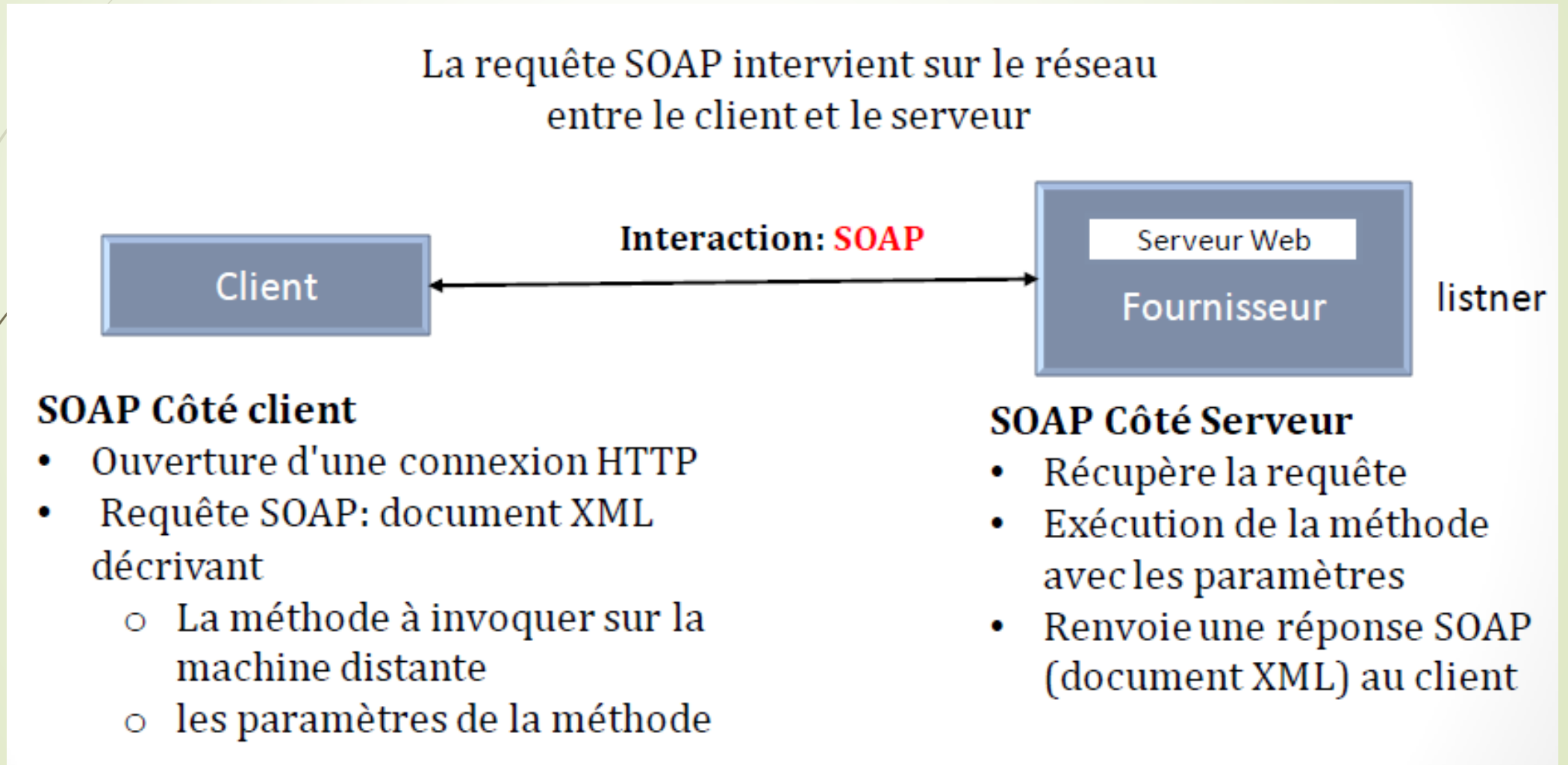
Omniprésente car SOAP est défini à un niveau suffisamment élevé d'abstractions que tout système d'exploitation et combinaison de langages de programmation peuvent être utilisés pour créer des programmes compatibles SOAP.

Basé sur XML, SOAP est construit sur XML, ce qui signifie que les documents SOAP sont des documents XML construits en fonction d'un cahier de charges plus strict.

Infrastructure distribuée, SOAP ne précise pas quelles données peuvent être déplacées ou bien quels appels de fonctions peuvent avoir lieu sur elle. Les applications construites sur la spécification SOAP peuvent déplacer les données d'un ordinateur A à un ordinateur B et par la suite à une autre application écrite sur la même spécification.

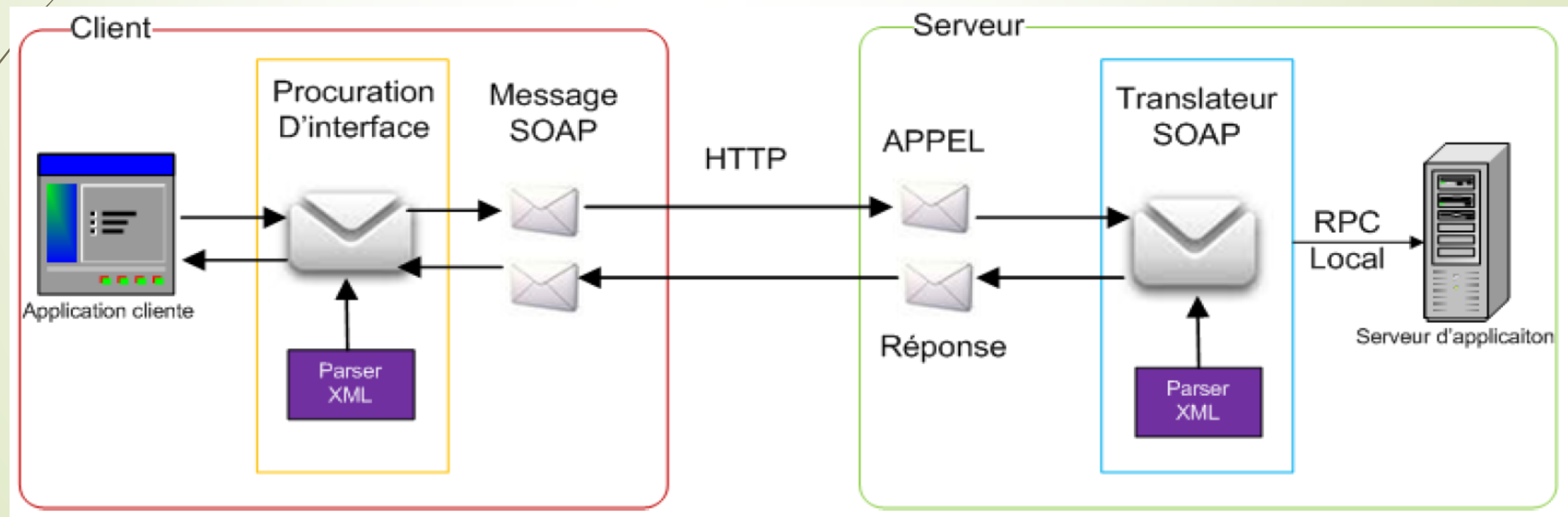
Le protocole de communication SOAP

La requête SOAP :



Le protocole de communication SOAP

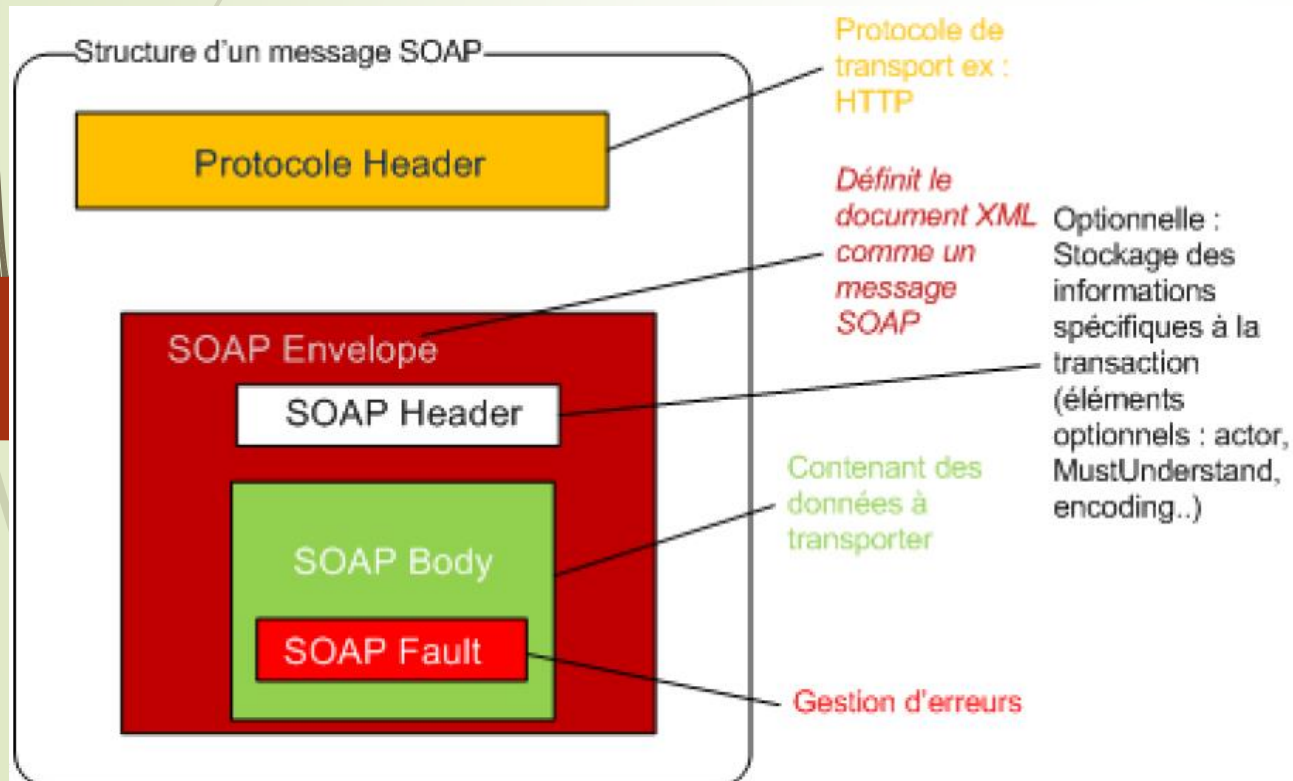
SOAP est un protocole d'invocation de méthodes sur des services distants. Basé sur XML, SOAP a pour principal objectif d'assurer la communication entre machines. Le protocole permet d'appeler une méthode RPC et d'envoyer des messages aux machines distantes via HTTP. Ce protocole est très bien adapté à l'utilisation des services Web, car il permet de fournir au client une grande quantité d'informations récupérées sur un réseau de serveurs tiers :



Structure d'un message SOAP

La grammaire de SOAP est assez simple à comprendre. Elle procure un moyen d'accès aux objets par appel de méthodes à distance. Les deux plus fortes fonctionnalités de SOAP sont sa simplicité et le fait que tout le monde a accepté de l'utiliser.


Un message SOAP est composé de deux parties obligatoires : l'enveloppe SOAP et le corps SOAP ; et une partie optionnelle : l'en-tête SOAP.



- **SOAP envelope** (enveloppe) est l'élément de base du message SOAP. L'enveloppe contient la spécification des espaces de désignation ([namespace](#)) et du codage de données.
- **SOAP header** (entête) est une partie facultative qui permet d'ajouter des fonctionnalités à un message SOAP de manière décentralisée sans agrément entre les parties qui communiquent. C'est ici qu'il est indiqué si le message est mandataire ou optionnel. L'entête est utile surtout, quand le message doit être traité par plusieurs intermédiaires.
- **SOAP body** (corps) est un *container* pour les informations mandataires à l'intention du récepteur du message, il contient les méthodes et les paramètres qui seront exécutés par le destinataire final.
- **SOAP fault** (erreur) est un élément facultatif défini dans le corps SOAP et qui est utilisé pour reporter les erreurs.

L'enveloppe SOAP

L'enveloppe SOAP sert de conteneur aux autres éléments du message SOAP, elle est définie au début par la balise `<soap:Envelope>` et se termine par la balise `</soap:Envelope>`. Les messages SOAP ne peuvent pas être envoyés en lots, autrement dit l'enveloppe contient un seul message constitué d'un entête facultatif (SOAP header) et d'un corps obligatoire (SOAP body).



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
3     soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
4
5     <soap:Header>
6         <!-- en-tête -->
7     </soap:Header>
8
9     <soap:Body>
10         <!-- corps -->
11     </soap:Body>
12 </soap:Envelope>
```

Exemple de communication en JAVA

Pour finir avec SOAP, voici un exemple de communication

SOAP Request

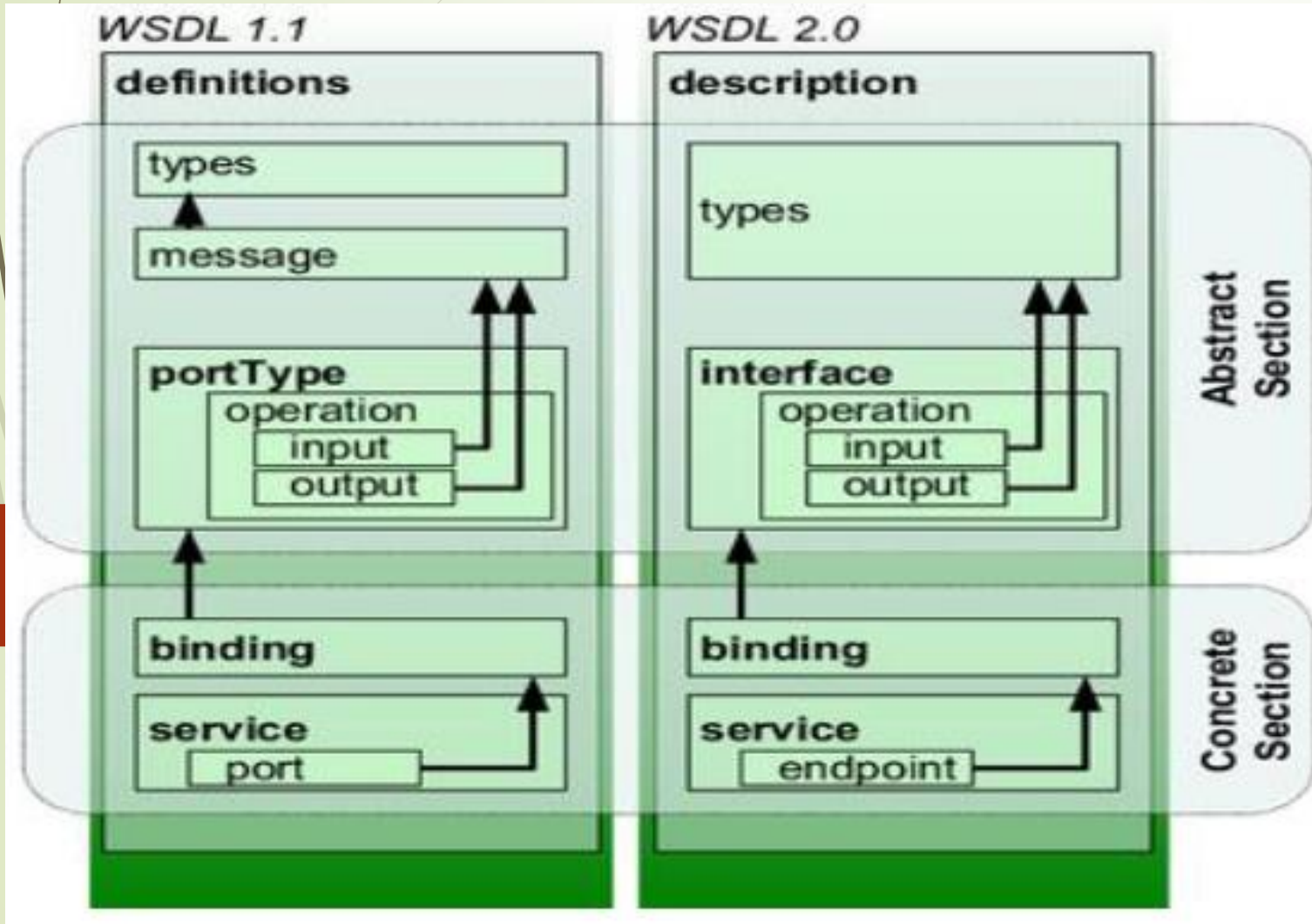
```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:add xmlns:ns2="http://calc/">
      <i>2</i>
      <j>3</j>
    </ns2:add>
  </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:addResponse xmlns:ns2="http://calc/">
      <return>5</return>
    </ns2:addResponse>
  </S:Body>
</S:Envelope>
```

Le langage de description WSDL

Description à 2 niveaux: Séparation entre la partie abstraite et concrète :

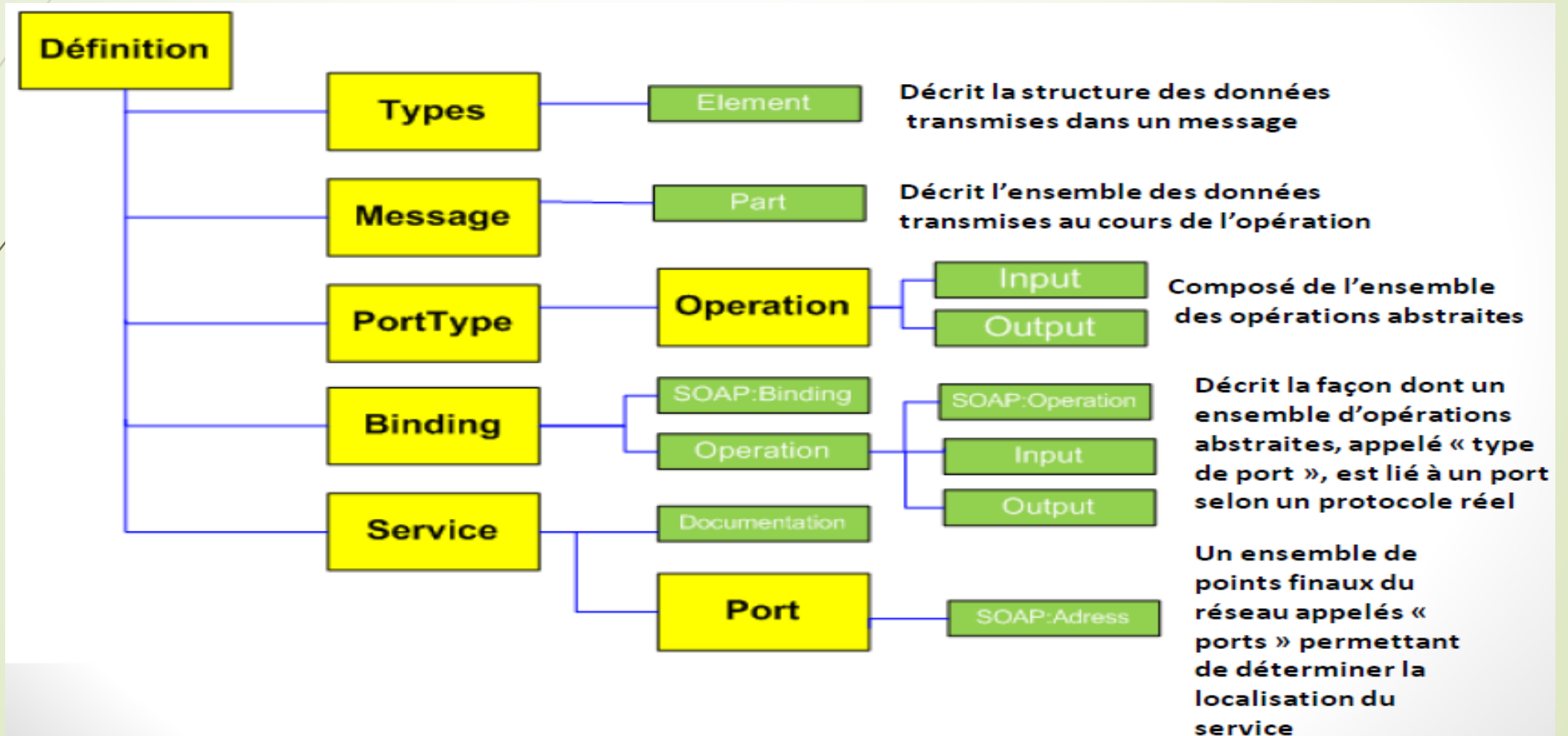


```
roswell:8080/WS_Project/WV x Method invocation trace x
roswell:8080/WS_Project/WSCalc?wsdl

This XML file does not appear to have any style information associated with it. The document tree is as follows:
<!--
  Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metadata-1.2.
-->
<!--...-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-wss-wssecurity-secext-1.1.xsd"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  targetNamespace="http://calc/" name="WSCalc">
  <wsp:Policy wsu:Id="WSCalcPortBindingPolicy"/>
  <types>...</types>
  <message name="add">...</message>
  <message name="addResponse">...</message>
  <message name="mul">...</message>
  <message name="mulResponse">...</message>
  <message name="Insert">...</message>
  <message name="InsertResponse">...</message>
  <message name="Exception">...</message>
  <message name="hello">...</message>
  <message name="helloResponse">...</message>
  <portType name="WSCalc">...</portType>
  <binding name="WSCalcPortBinding" type="tns:WSCalc">...</binding>
  <service name="WSCalc">
    <port name="WSCalcPort" binding="tns:WSCalcPortBinding">
      <soap:address location="http://roswell:8080/WS_Project/WSCalc"/>
    </port>
  </service>
</definitions>
```


Le langage de description 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.



Structure d'un document WSDL

Un fichier WSDL contient donc sept éléments.

- **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.

Le document WSDL peut être divisé en deux parties. Une partie pour les définitions abstraites, tandis que la deuxième contient les descriptions concrètes.

Mapping Java ↔ WSDL

Code Java	Fichier WSDL
Classe Java public class IpAddressSearchWebService	Service <wsdl: service name="IpAddressSearchWebService"> </wsdl:service>
Commentaire /*using this function, user can find the country and city by Ip */	Documentation <wsdl:documentation> using this function, user can find the country and city by Ip </wsdl:documentation>
Méthode public String[] getCountryCityByIp(String thelpAddress)	Operation <wsdl:operation name="getCountryCityByIp"> <wsdl:input/> <wsdl:output..../> </wsdl:operation>
Paramètre String thelpAddress	Types <wsdl:types> <s:schema elementFromDefault="qualified"> <s:element name="getCountryCityByIp"> <s:complexType> <s:sequence> <s:element minOccurs="0" maxoccurs="1" name="thelpAdriess" type="s:string" /> </s:sequence> </s:complexType>...</s:element> </wsdl:types>

L'annuaire des services UDDI

L'annuaire des services UDDI est un standard pour la publication et la découverte des informations sur les services Web. La spécification UDDI est une initiative lancée par ARIBA, Microsoft et IBM. Cette spécification n'est pas gérée par le W3C mais par le groupe OASIS. La spécification UDDI vise à créer une plate-forme indépendante, un espace de travail (framework) ouvert pour la description, la découverte et l'intégration des services des entreprises.

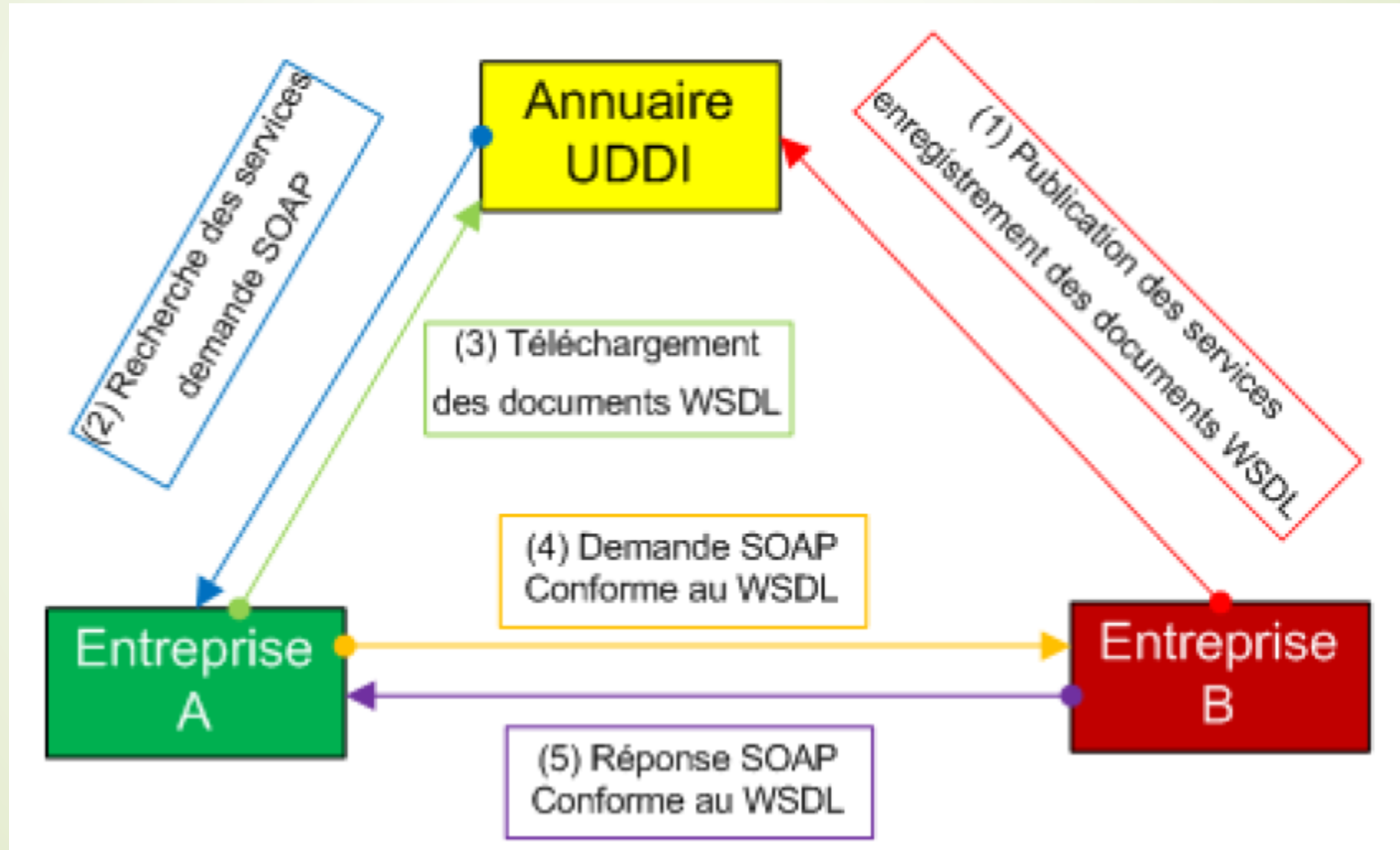
Consultation de l'annuaire

L'annuaire UDDI se concentre sur le processus de découverte de l'architecture orientée services (SOA), et utilise des technologies standards telles que XML, SOAP et WSDL qui permettent de simplifier la collaboration entre partenaires dans le cadre des échanges commerciaux. L'accès au référentiel s'effectue de différentes manières.

- Les pages blanches comprennent la liste des entreprises ainsi que des informations associées à ces dernières (coordonnées, description de l'entreprise, identifiants...).
- Les pages jaunes recensent les services Web de chacune des entreprises sous le standard WSDL.
- Les pages vertes fournissent des informations techniques précises sur les services fournis.

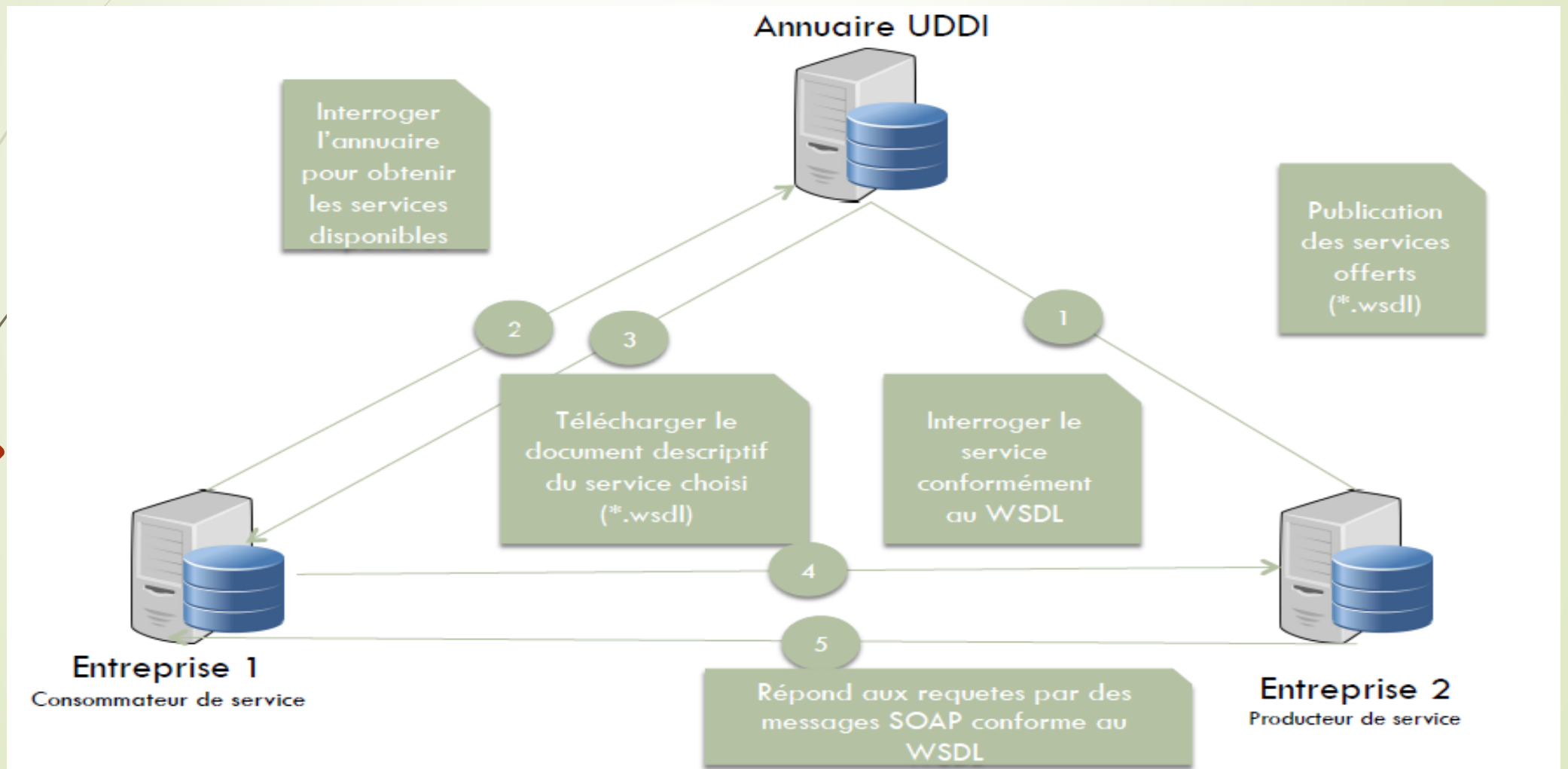
L'annuaire des services UDDI

Le scénario classique d'utilisation de UDDI est illustré ci-dessous. L'entreprise B a publié le service Web S, et l'entreprise A est client de ce service :



L'annuaire des services UDDI

Le scénario classique d'utilisation de UDDI est illustré ci-dessous. L'entreprise B a publié le service Web S, et l'entreprise A est client de ce service :



Services Web RESTful

Définition

- Acronyme de **RE**presentational **S**tate **T**ransfert défini dans la thèse de Roy Fielding en 2000.
- REST n'est pas un protocole ou un format, contrairement à SOAP, HTTP ou RCP, mais un style d'architecture inspiré de l'architecture du web fortement basé sur le protocole HTTP
- Il n'est pas dépendant uniquement du web et peut utiliser d'autre protocoles que HTTP

Services Web RESTful

Ce qu'il est :

- Un système d'architecture
- Une approche pour construire une application

Ce qu'il n'est pas

- Un protocole
- Un format
- Un standard

Services Web RESTful

Ce qu'il est :

- Un système d'architecture
- Une approche pour construire une application

Ce qu'il n'est pas

- Un protocole
- Un format
- Un standard

Utiliser dans le développement des applications orientés ressources (ROA) ou orientées données (DOA)

Les applications respectant l'architecture REST sont dites RESTful

REST → Fournisseurs

facebook



Google

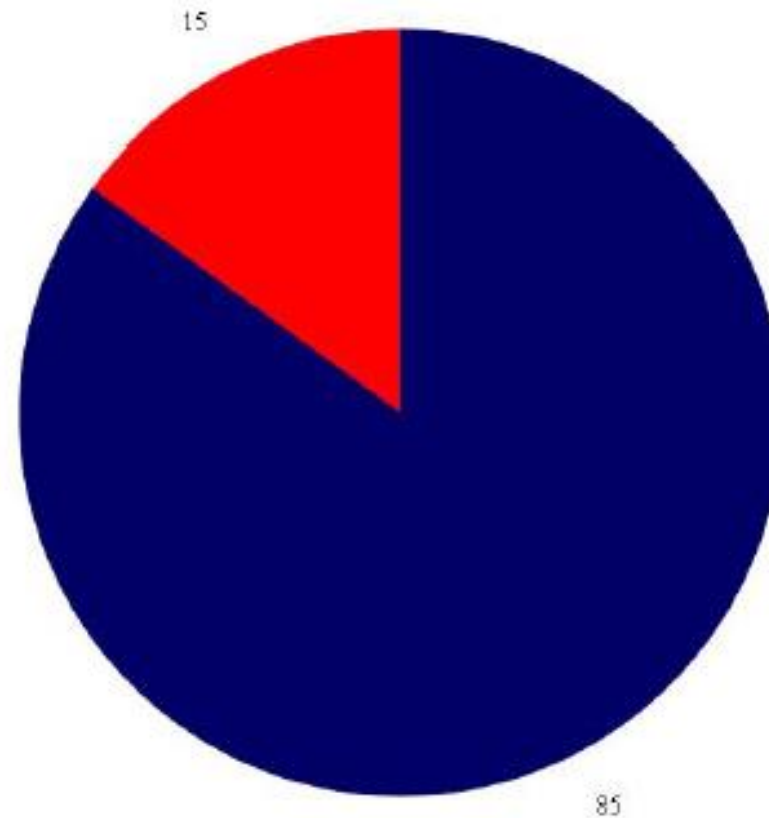
ebay



REST → Statistiques

Statistique d'utilisation des services web REST et SOAP chez AMAZON

■ REST ■ SOAP



REST → Caractéristiques

- Les services REST sont sans états (Stateless)
 - ✓ Chaque requête envoyée au serveur doit contenir toutes les informations relatives à son état et est traitée indépendamment de toutes autres requêtes
 - ✓ Minimisation des ressources systèmes (pas de gestion de session, ni d'état)
- Interface uniforme basée sur les méthodes HTTP (GET, POST, PUT, DELETE)
- Les architectures RESTful sont construites à partir de ressources uniquement identifiées par des URI(s)

REST → Caractéristiques

➤ Ressources

- ✓ Identifiée par une URI (<http://ESTS.ma/cursus/LicenceP/BigD>)

➤ Méthodes (verbes) permettant de manipuler les ressources (identifiants)

- ✓ Méthodes HTTP : GET, POST, PUT, DELETE

➤ Représentation : Vue sur l'état de la ressource

- ✓ Format d'échanges entre le client et le serveur (XML, JSON, text/plain,...)

Ressources

Une ressource est un objet identifiable sur le système

→ Livre, Catégorie, Client, Prêt

Une ressource n'est pas forcément un objet matérialisé (Prêt, Consultation, Facture...)

- ✓ Une ressource est identifiée par une URI : Une URI identifie uniquement une ressource sur le système

<http://Biblio.ESTS.ma/bookstore/books/1>

Clef primaire de la
ressource dans la BDD

Méthodes (Verbes)

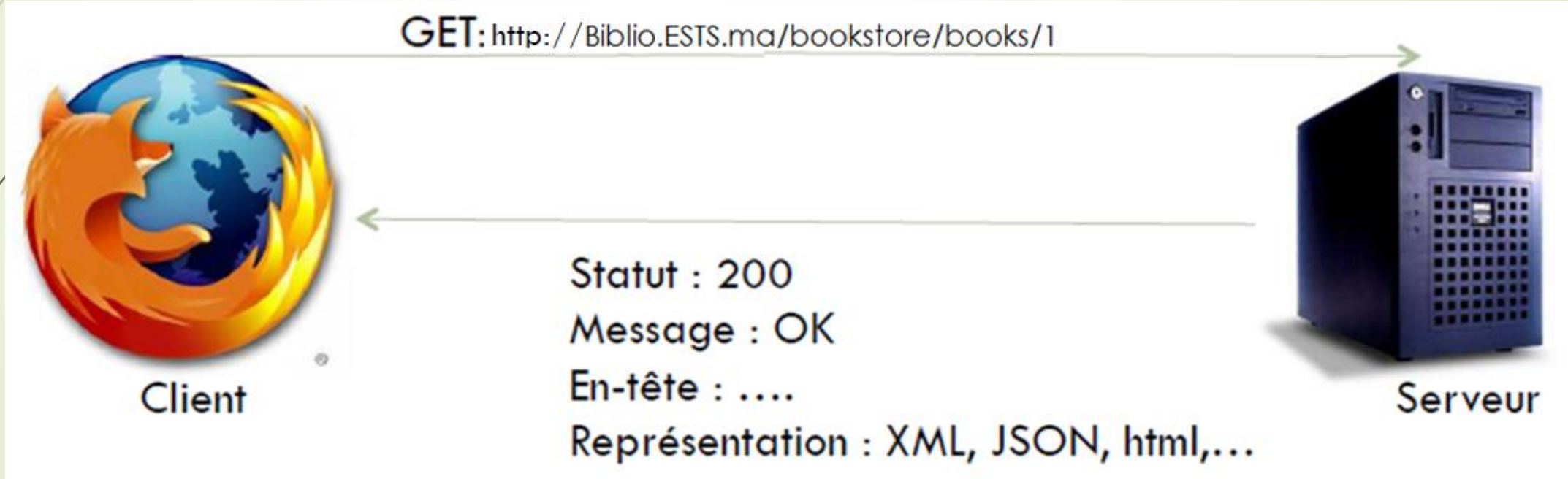
Une ressource peut subir quatre opérations de bases **CRUD** correspondant aux quatre principaux types de requêtes HTTP (GET, PUT, POST, DELETE)

→ REST s'appuie sur le protocole HTTP pour effectuer ces opérations sur les objets

- CREATE → POST
- RETRIEVE → GET
- UPDATE → PUT
- DELETE → DELETE

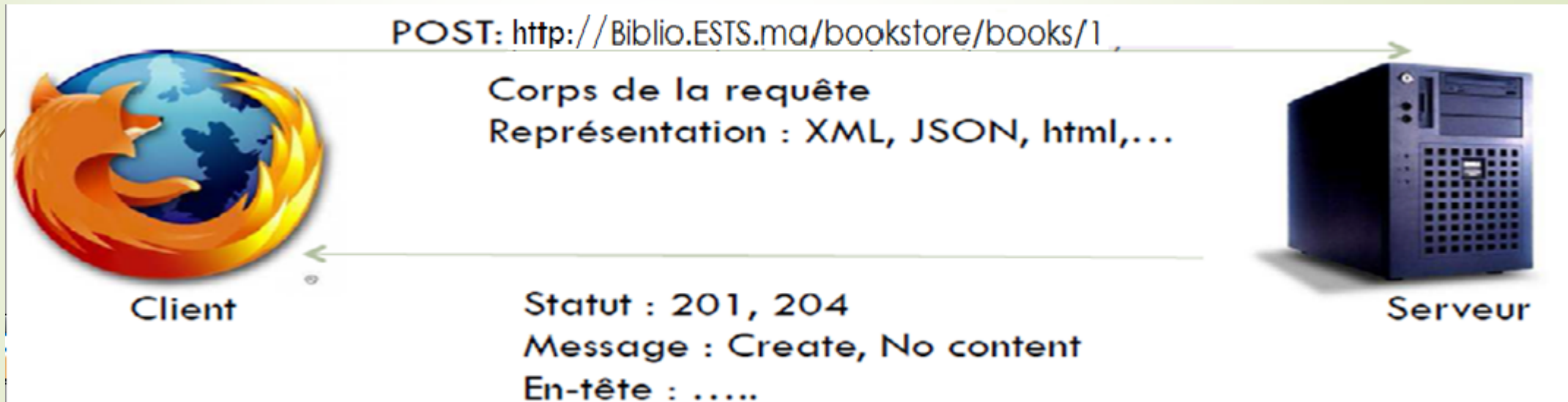
Méthode GET

La méthode GET renvoie une représentation de la ressource tel qu'elle est sur le système



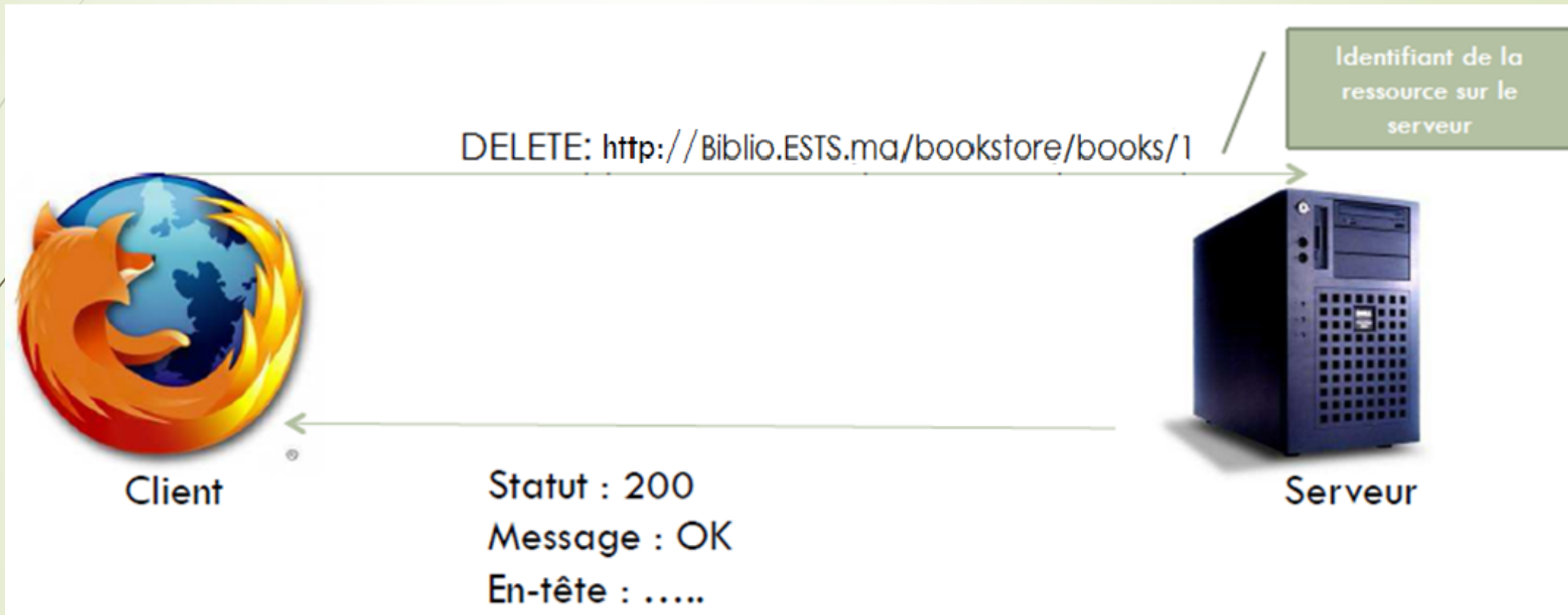
Méthode POST

La méthode POST crée une nouvelle ressource sur le système



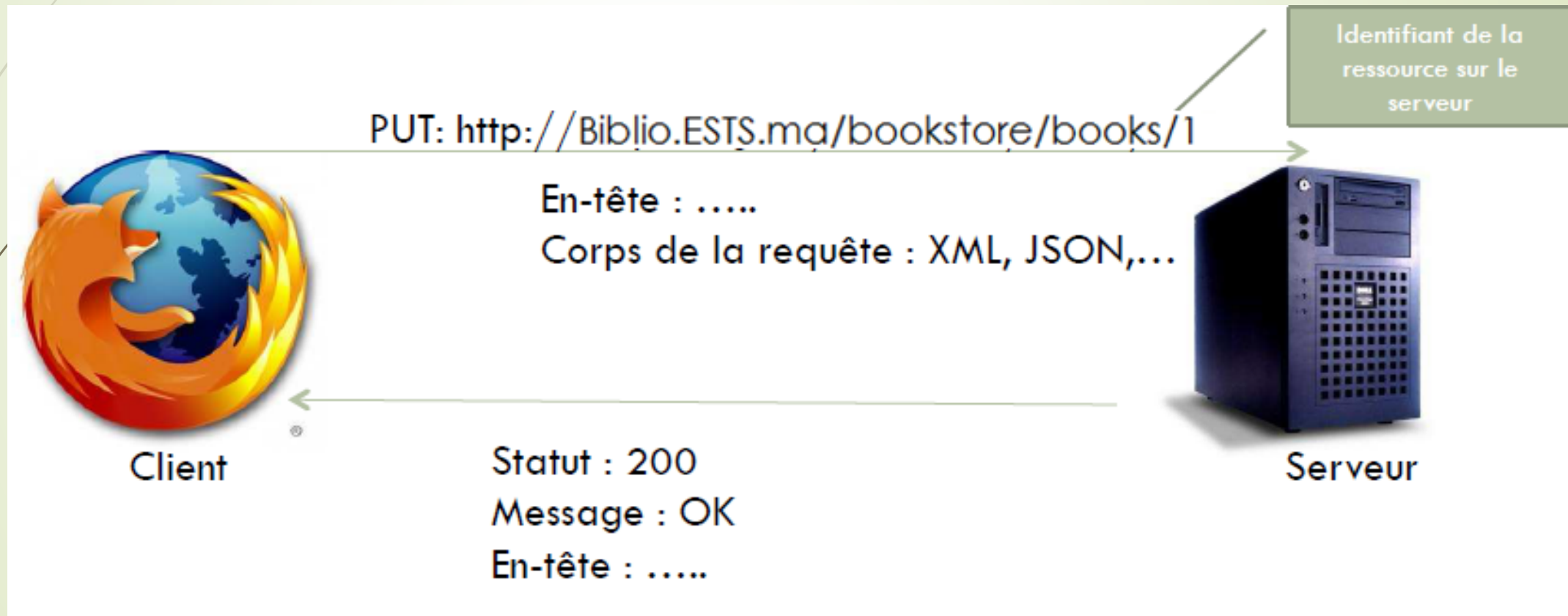
Méthode DELETE

Supprime la ressource identifiée par l'URI sur le serveur



Méthode PUT

Mise à jour de la ressource sur le système



Réflexions

➤ Que se passe t il

- ✓ si on fait de la lecture avec un POST ?
- ✓ Si on fait une mise à jour avec un DELETE ?
- ✓ Si on fait une suppression avec un PUT ?

REST ne l'interdit pas

→ Mais si vous le faites, votre application ne respecte pas les exigences REST et donc n'est pas RESTful

Représentation

Une représentation désigne les données échangées entre le client et le serveur pour une ressource:

- HTTP GET → Le serveur renvoie au client l'état de la ressource
- PUT, POST → Le client envoie l'état d'une ressource au serveur

Peut être sous différent format :

- ✓ JSON
- ✓ XML
- ✓ XHTML
- ✓ CSV
- ✓ Text/plain
- ✓

WADL

Web Application Description Language

- Standard du W3C
- Permet de décrire les éléments des services
 - ✓ Resource, Méthode, Paramètre, Réponse
- Permet d'interagir de manière dynamique avec les applications REST



→ Moins exploité que le WSDL pour les Services SOAP

WADL

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<application xmlns="http://wadl.dev.java.net/2009/02">
  <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 2.0 2013-05-03 14:50:15"/>
  <grammars/>
  ▼<resources base="http://localhost:8080/Bibliotheque/webresources/">
    ▼<resource path="category">
      ▼<method id="test" name="GET">
        ▼<response>
          <representation mediaType="application/xml"/>
          <representation mediaType="application/json"/>
        </response>
      </method>
      ▼<method id="apply" name="OPTIONS">
        ▼<request>
          <representation mediaType="*/*"/>
        </request>
        ▼<response>
          <representation mediaType="application/vnd.sun.wadl+xml"/>
        </response>
      </method>
      ▼<method id="apply" name="OPTIONS">
        ▼<request>
          <representation mediaType="*/*"/>
        </request>
        ▼<response>
          <representation mediaType="text/plain"/>
        </response>
      </method>
      ▼<method id="apply" name="OPTIONS">
        ▼<request>
          <representation mediaType="*/*"/>
        </request>
        ▼<response>
          <representation mediaType="*/*"/>
        </response>
      </method>
    </resources>
  </application>
```

Services Web étendus VS REST

SOAP

➤ Avantages

- ✓ Standardisé
- ✓ Interopérabilité
- ✓ Sécurité (WS-Security)

➤ Inconvénients

- ✓ Performances (enveloppe SOAP supplémentaire)
- ✓ Complexité, lourdeur
- ✓ Cible l'appel de service

Services Web étendus VS REST

REST

➤ Avantages

- ✓ Simplicité de mise en oeuvre
- ✓ Lisibilité par un humain
- ✓ Evolutivité
- ✓ Repose sur les principes du web
- ✓ Représentations multiples (XML, JSON,...)

➤ Inconvénients

- ✓ Sécurité restreinte par l'emploi des méthodes HTTP
- ✓ Cible l'appel de ressources



PARTIE II

Utiliser et tester un service Web existant



Utiliser et tester un service Web existant

Vous allez tester des services web existants. Ensuite, vous allez découvrir comment développer, de ployer un service web simple.

Environnement de travail

- ▀ Environnement de développement :
- ▀ IDE Netbeans
- ▀ JDK 7 ou 8
- ▀ Serveur Glassfish



Exemple d'une requête http pour récupérer un WSDL

L'objectif est de vous montrer comment on peut récupérer un fichier WSDL seulement avec une requête HTTP. Pour cela, il est possible d'utiliser l'extension de Firefox : Poster.

- télécharger le module Poster
- sur Outils → Poster et vous complétez le formulaire avec l'URL suivante :
<http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl>
- Cliquer sur GET. Le fait de cliquer sur GET provoque l'envoi d'une requête HTTP en utilisant la méthode GET. Une nouvelle fenêtre s'ouvre avec la réponse à la requête GET. Les informations sont renvoyées au format XML et représente la de finition WSDL du Web service Amazon.
- Examiner le contenu

Tester un service Web existant

Avant de créer votre propre service web, vous allez tester les services web déjà existants et déployés sur des portails de services web.

Il existe différents portails/annuaires de services web tels que webservicex.net, Titan3, ProgrammableWeb4, etc. Vous pouvez tester les différents portails en faisant une recherche de services.

Aller sur le site : <http://www.webservicex.net>

- Choisir par exemple la catégorie «Utilities» et un des services web disponibles par exemple Global Weather. Que fait ce service ?
- L'adresse suivante : <http://www.webservicex.net/globalweather.asmx> est identifiée comme le Endpoint (le point d'accès) du service web. :
- Combien d'Operations sont gérées par le service Web ?
- Confirmer ce nombre d'opération en analysant l'interface WSDL de description du service.
- Tester le service.
- Expliquer le résultat obtenu et analysez les documents SOAP d'échange entre votre navigateur et le Web service.

Tester un service Web existant

Vous pouvez tester d'autres services web par exemple un service web pour envoyer des SMS. Toujours sur liste webservicex : cliquer sur la catégorie « Messaging » puis sur SendSMSWorld. Vous pouvez tester ce service en remplissant le formulaire Test propose . Vous pouvez vous envoyer un SMS, malheureusement le résultat mettra un certain temps a vous parvenir (1 ou 2 jours).

Différents portails/annuaires de services web :

- <http://nordicapis.com/api-discovery-11-ways-to-find-apis/>
- <http://www.webservicex.net>
- <http://ccnt.zju.edu.cn:8080/>
- <http://www.programmableweb.com/>
- <http://www.webservicelist.com/>
- <https://www.publicapis.com/>
- <https://algorithmia.com/algorithms>

PARTIE III



Création d'un premier service Web SOAP



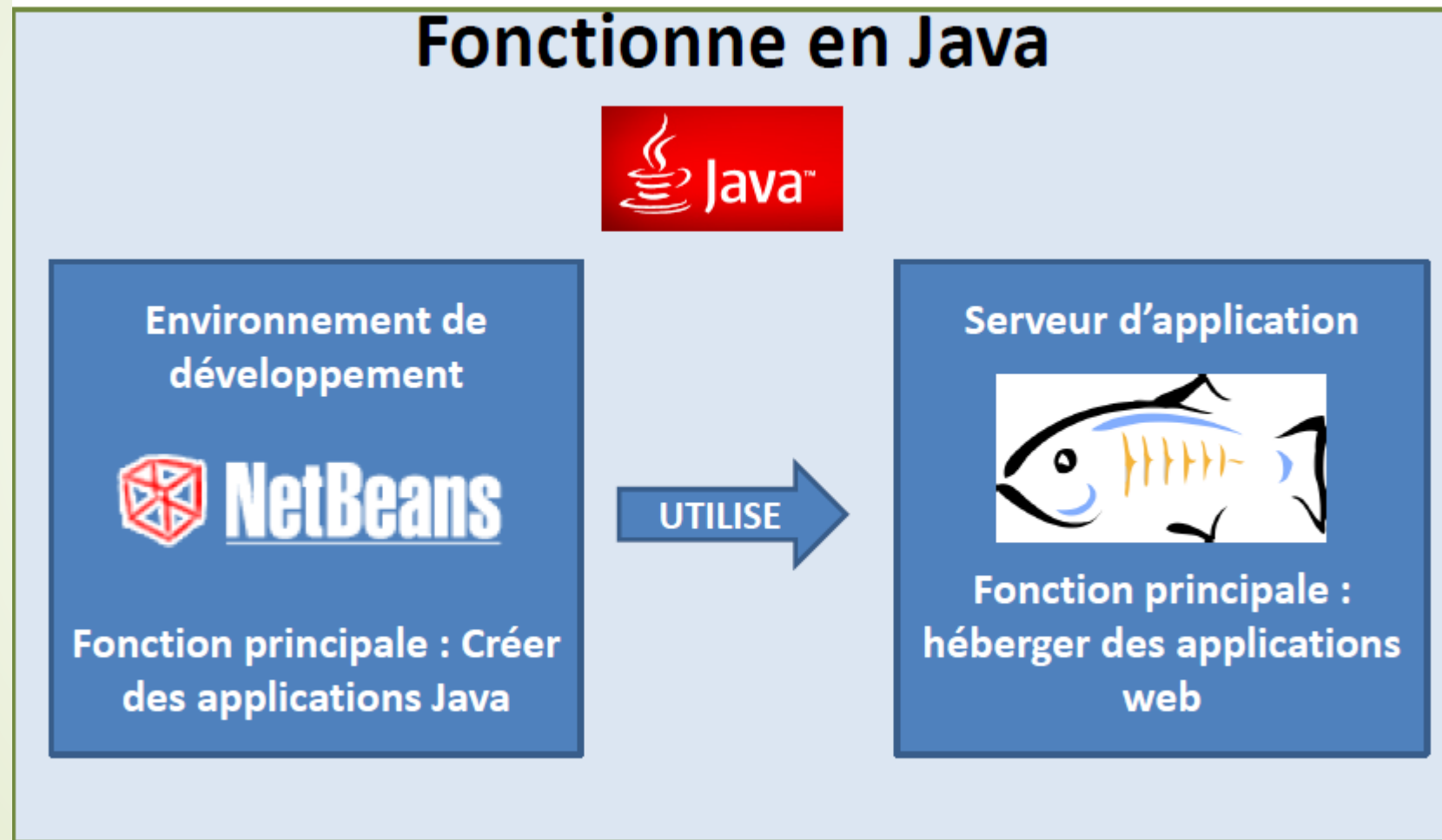
Avant de commencer

Environnement de travail

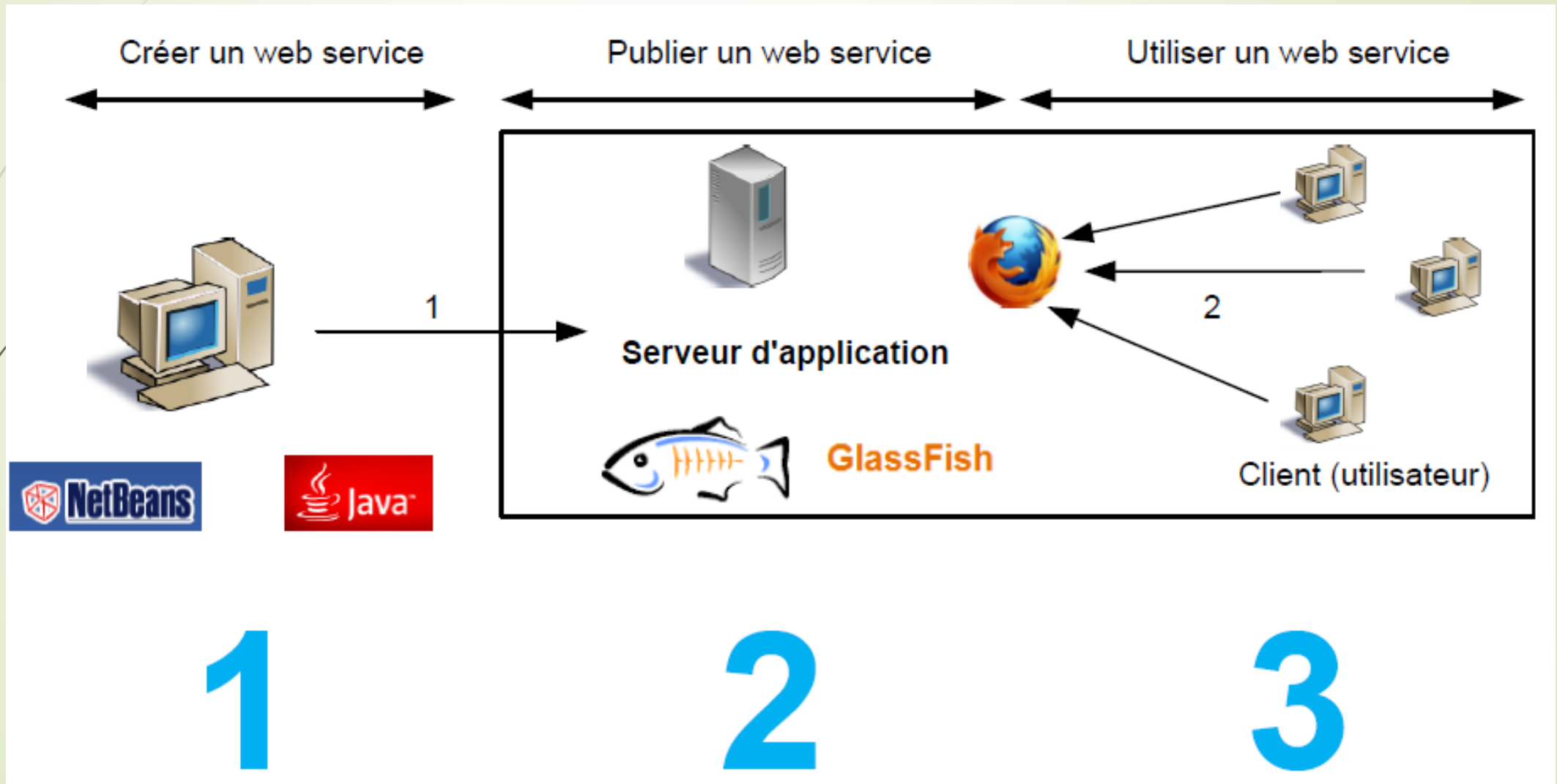
- ▀ Environnement de développement :
- ▀ IDE Netbeans
- ▀ JDK 7 ou 8
- ▀ Serveur Glassfish

Test d'ouverture de NetBeans

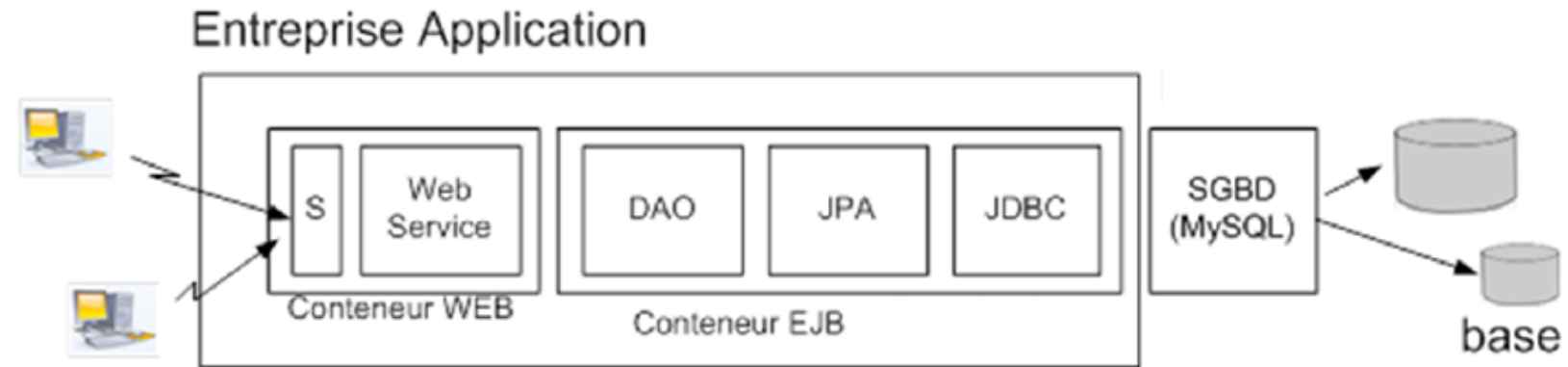
Java...NetBeans... GlassFish



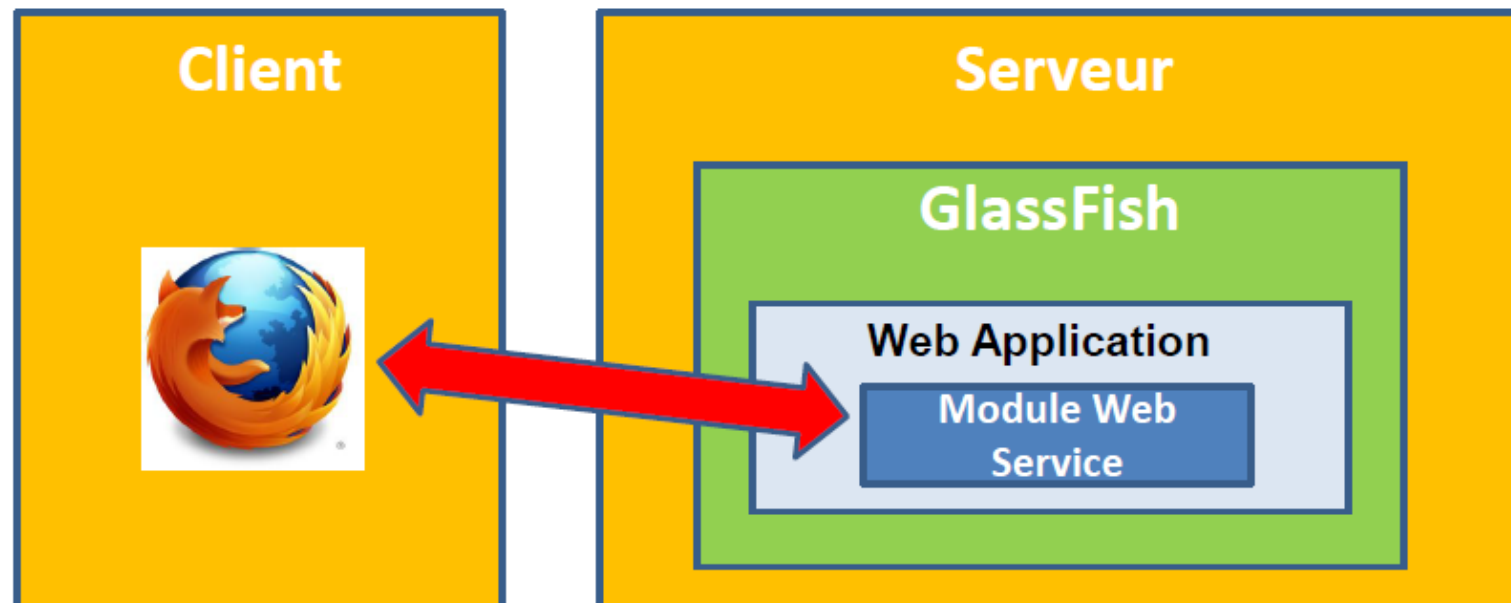
On crée un Service Web



Le cadre de travail

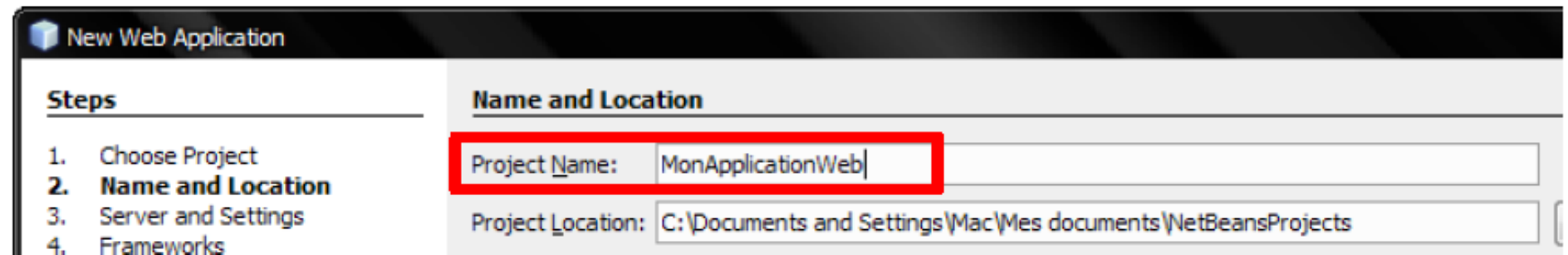
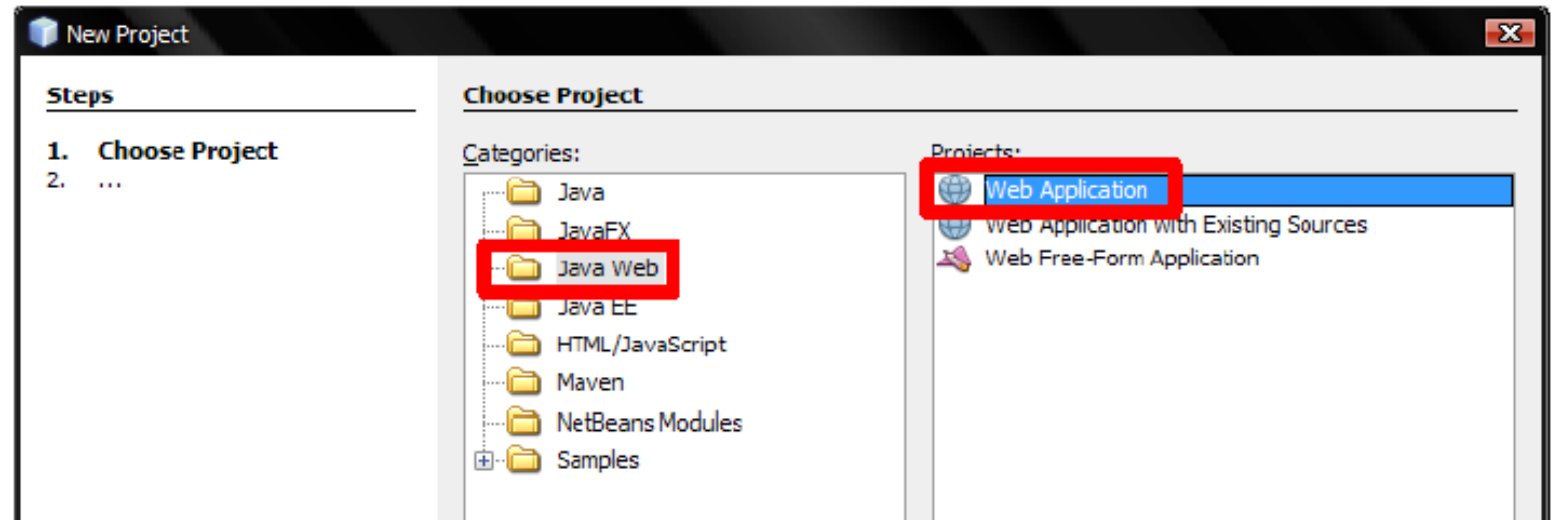


On va faire un peu plus simple pour commencer :



Définir un projet sur le serveur

- Démarrer Netbeans
- Créer un projet sur le serveur



Choisir son serveur

New Web Application

Steps

1. Choose Project
2. Name and Location
3. **Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application: <None>

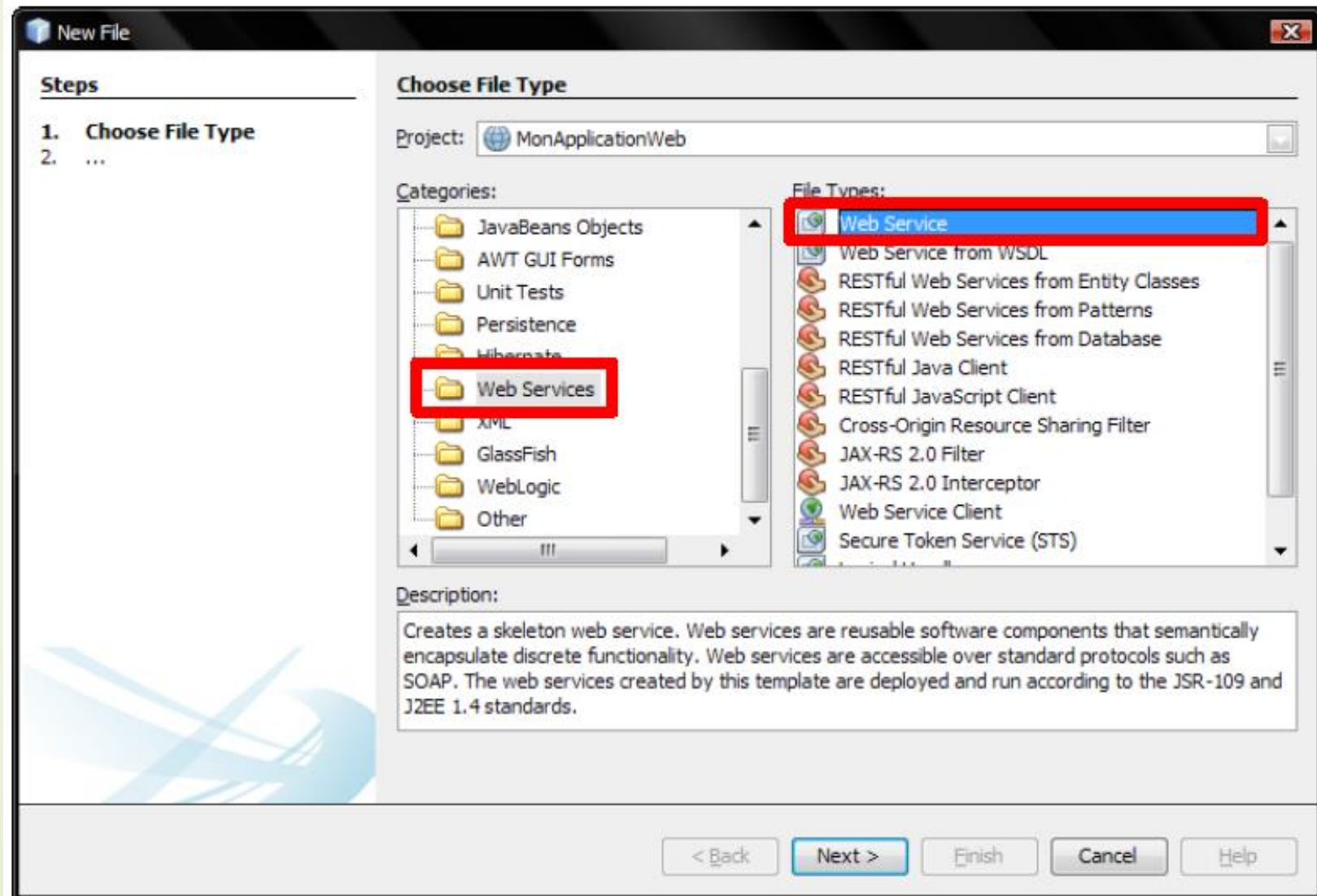
Server: GlassFish Server 4.0 Add...

Java EE Version: Java EE 7 Web

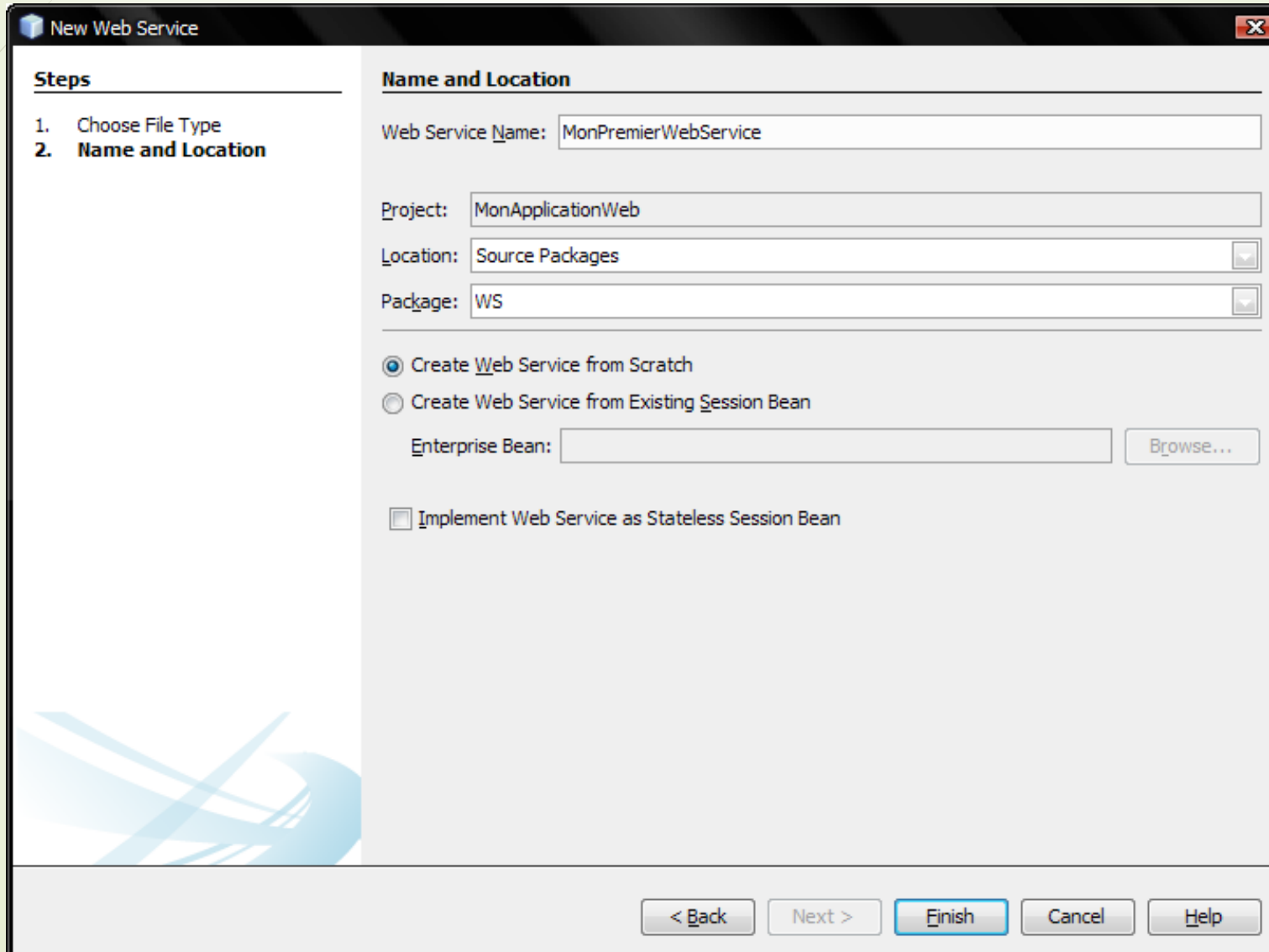
Context Path: /MonApplicationWeb

< Back Next > Finish Cancel Help

Ajouter un web service



Ajouter un web service



New Web Service

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Web Service Name: MonPremierWebService

Project: MonApplicationWeb

Location: Source Packages

Package: WS

☒ Create Web Service from Scratch

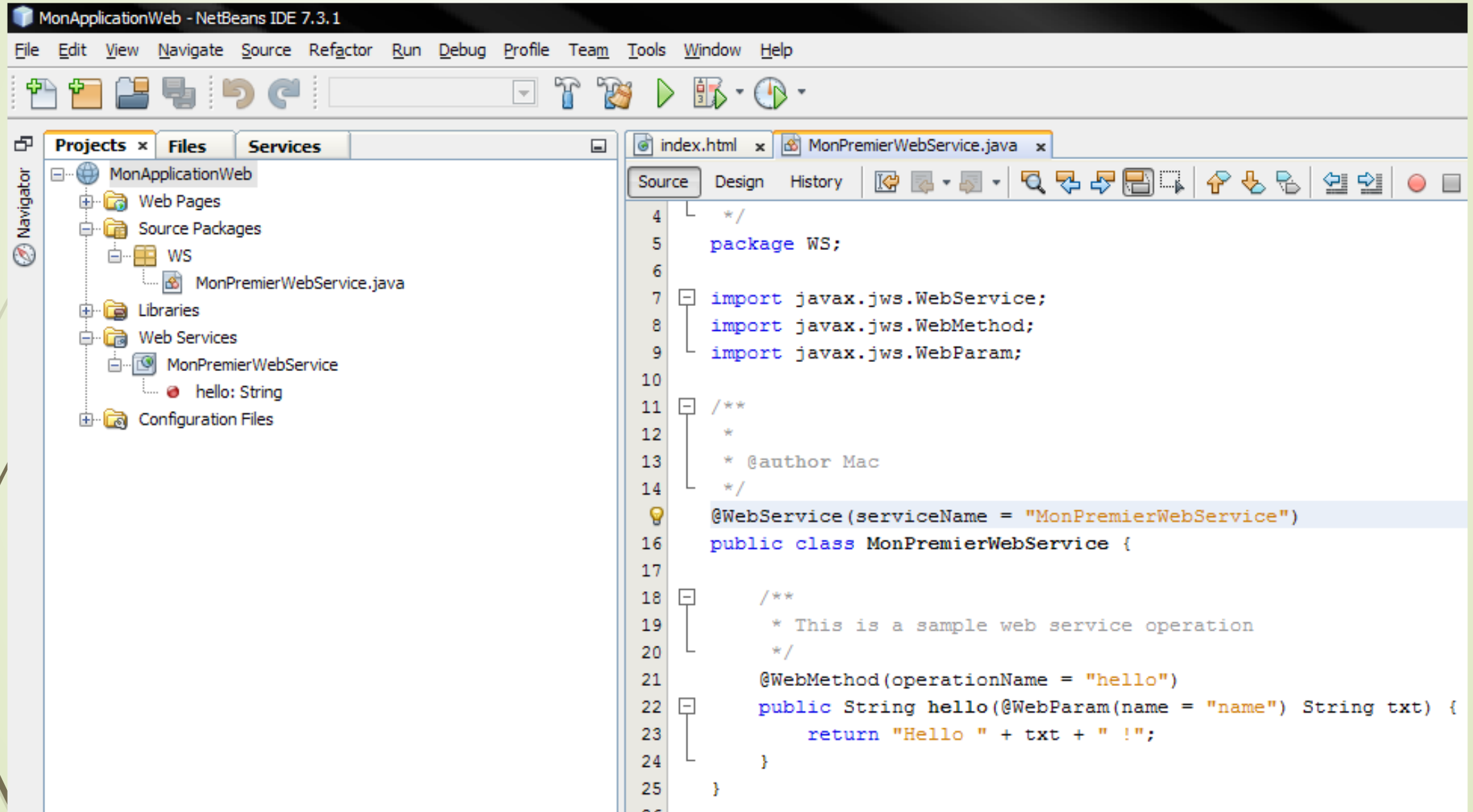
☐ Create Web Service from Existing Session Bean

Enterprise Bean:

☐ Implement Web Service as Stateless Session Bean

< Back Next > **Finish** Cancel Help

Un « nouveau » projet Java



Ajouter une méthode

```
@WebMethod(operationName = "hello")
public String hello(@WebParam(name = "name") String txt) {
    return "Hello " + txt + " !";
}

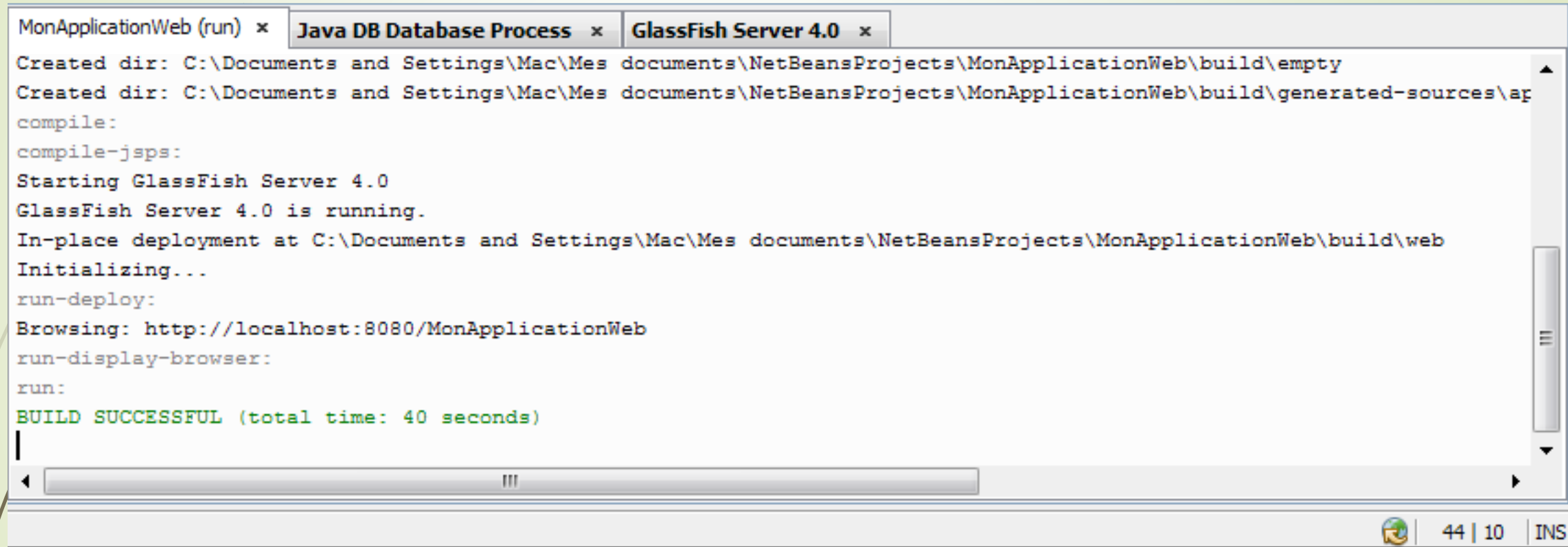
@WebMethod(operationName = "additionner")
public int additionner(@WebParam(name = "a") int a, @WebParam(name = "b") int b) {

    int s = 0;

    for(int i=1;i<5000;i++)
    {
        for(int j=1;j<5000;j++)
        {
            for(int k=1;k<5000;k++)
            {
                s = s + 1;
            }
        }
    }

    return (s+a+b-s);
}
}
```


Déployer le travail....



The screenshot shows a console window with three tabs: 'MonApplicationWeb (run)', 'Java DB Database Process', and 'GlassFish Server 4.0'. The 'MonApplicationWeb (run)' tab is active, displaying the following output:

```
Created dir: C:\Documents and Settings\Mac\Mes documents\NetBeansProjects\MonApplicationWeb\build\empty
Created dir: C:\Documents and Settings\Mac\Mes documents\NetBeansProjects\MonApplicationWeb\build\generated-sources\ap
compile:
compile-jsp:
Starting GlassFish Server 4.0
GlassFish Server 4.0 is running.
In-place deployment at C:\Documents and Settings\Mac\Mes documents\NetBeansProjects\MonApplicationWeb\build\web
Initializing...
run-deploy:
Browsing: http://localhost:8080/MonApplicationWeb
run-display-browser:
run:
BUILD SUCCESSFUL (total time: 40 seconds)
```

The bottom of the window shows a taskbar with a clock icon, the text '44 | 10', and 'INS'.

Adresse de la web application :
<http://localhost:8080/MonApplicationWeb>

Tester votre travail



The screenshot shows a Mozilla Firefox browser window titled "MonPremierWebService Testeur de service Web - Mozilla Firefox". The address bar shows the URL "localhost:8080/MonApplicationWeb/MonPremierWebService?Tester". The page content includes a title "MonPremierWebService Testeur de service Web", a description "Ce formulaire vous permettra de tester votre implémentation de service Web (Fichier WSDL)", and instructions "Pour appeler une opération, remplissez les zones de saisie des paramètres de la méthode et cliquez sur le bouton libellé avec". Below this, two methods are listed: "hello" and "addionner", each with a button and input fields for parameters.

MonPremierWebService Testeur de service Web

Ce formulaire vous permettra de tester votre implémentation de service Web ([Fichier WSDL](#))

Pour appeler une opération, remplissez les zones de saisie des paramètres de la méthode et cliquez sur le bouton libellé avec

Méthodes :

public abstract java.lang.String ws.MonPremierWebService.hello(java.lang.String)

hello ()

public abstract int ws.MonPremierWebService.addionner(int,int)

addionner (,)

Tester votre travail

Trace d'appel de méthode - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

☐ Trace d'appel de méthode +

localhost:8080/MonApplicationWeb/MonPremierWebService?Tester

ajouter Appel de méthode

Method parameter(s)

Type	Value
int	2
int	3

Méthode renvoyée

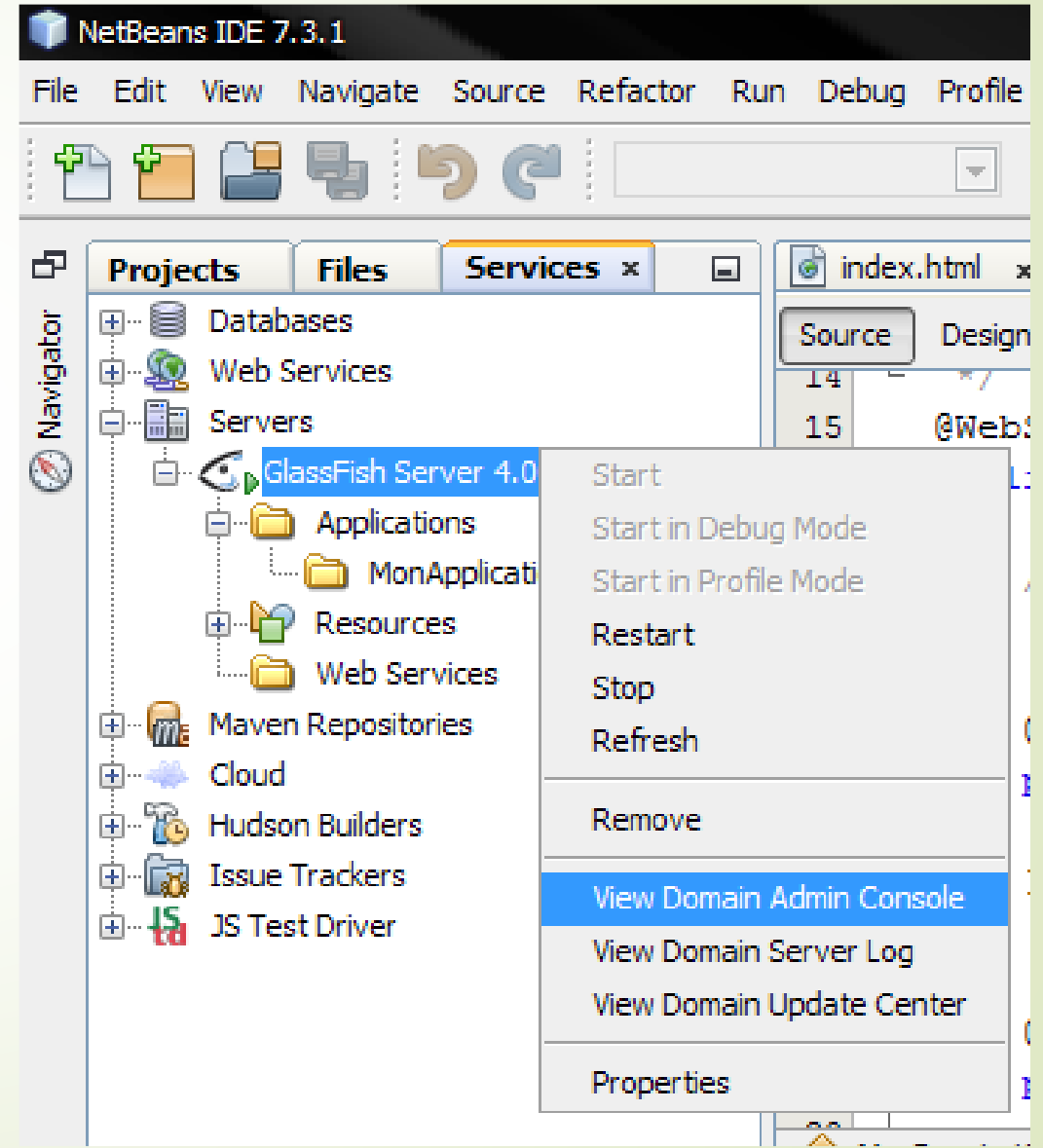
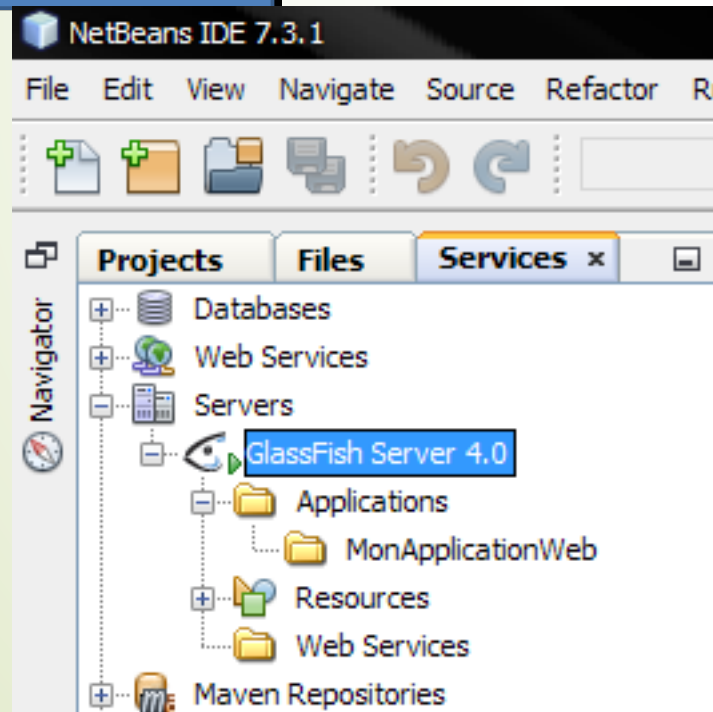
int : "5"

Comprendre son projet

Serveur d'application



Fonction principale :
héberger des applications
web



Administrer Glassfish

Administrer Glassfish

Applications - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

Trace d'appel de méthode Applications

localhost:4848/common/index.jsf

addionner

Accueil A propos de... Aide

Utilisateur : admin | Rôle : domain1 | Serveur : localhost

GlassFish™ Server Open Source Edition

Nombre total de mises à jour disponibles : 1

Arborescence

- Tâches courantes
- Domaine
 - serveur (serveur d'administration)
- Clusters
- Instances autonomes
- Noeuds
- Applications**
- Modules de cycle de vie
- Données de surveillance
- Ressources
 - Ressources simultanées
 - Connecteurs
 - JDBC
 - Ressources JMS
 - JNDI
 - Sessions JavaMail
 - Configurations d'adaptateurs de r
- Configurations
 - default-config
 - server-config
- Outil de mise à jour

Applications

Les applications peuvent être de type Enterprise ou Web, ou différentes sortes de modules. Pour redémarrer une application ou un module, cliquez sur le lien de rechargement ; cette action sera uniquement appliquée aux cibles sur lesquelles l'application ou le module est activé.

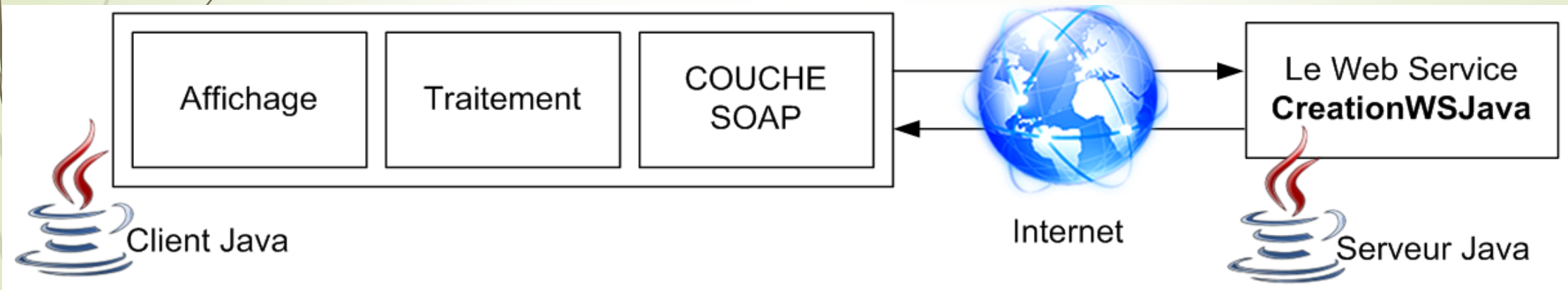
Applications déployées (1)

☒ ☐ Déployer... Annuler le déploiement Activer Désactiver Filtre : ▼

Select	Nom	Ordre de déploiement	Activé	Moteurs	Action
<input type="checkbox"/>	MonApplicationWeb	100	✓	webservices, web	Lancer Redéploiement Recharger

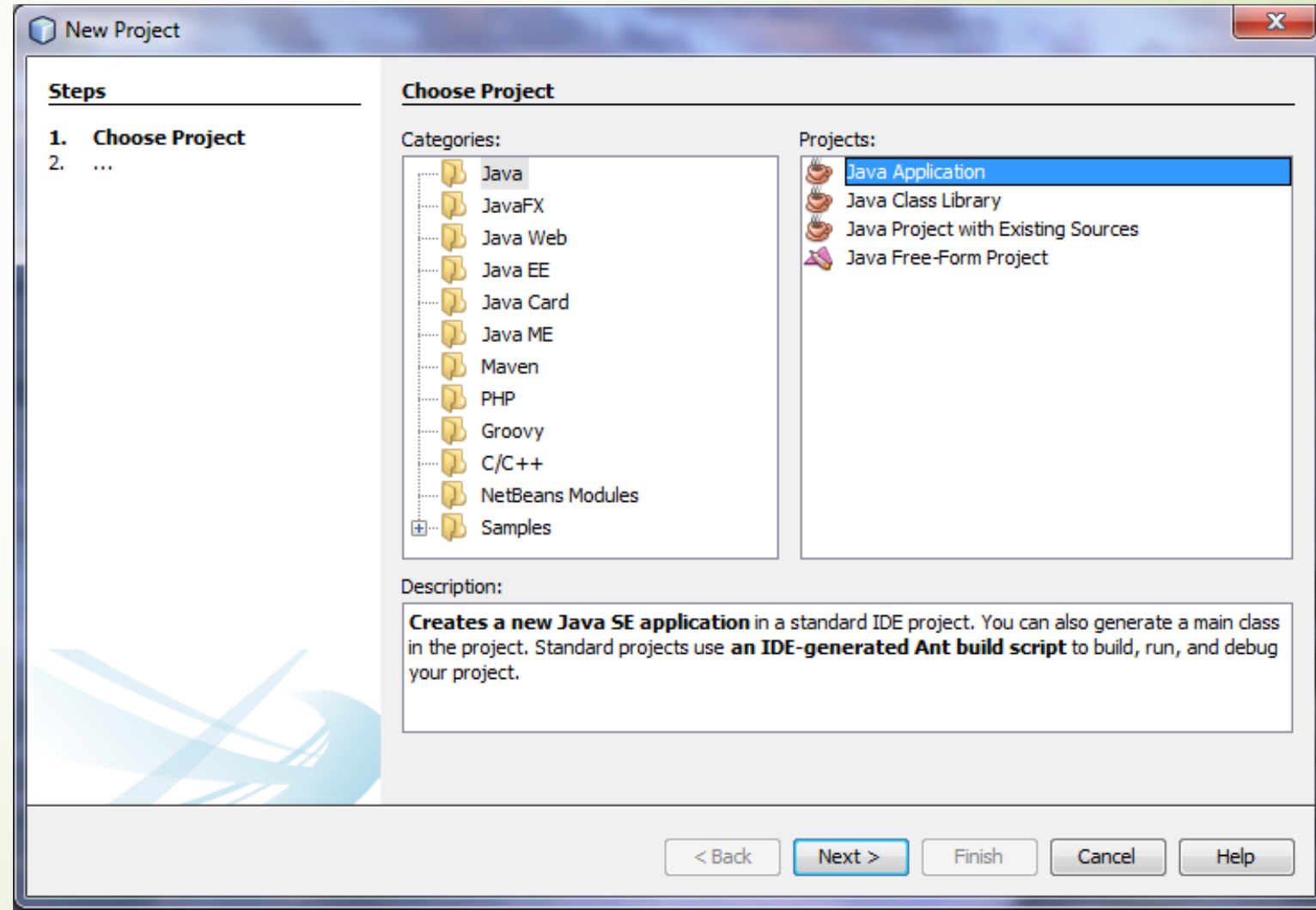
Au niveau client

Le cadre de travail



Définir un projet

1. Démarrer Netbeans
2. Créer un projet



Définir un projet

Name and Location

Project Name: UtiliseWSJava

Project Location: C:\Users\acomme.T3500-PC\Desktop\TutoWS\TestWebServiceJava

Browse...

Project Folder: C:\Users\acomme.T3500-PC\Desktop\TutoWS\TestWebServiceJava\UtiliseWSJava

☐ Use Dedicated Folder for Storing Libraries

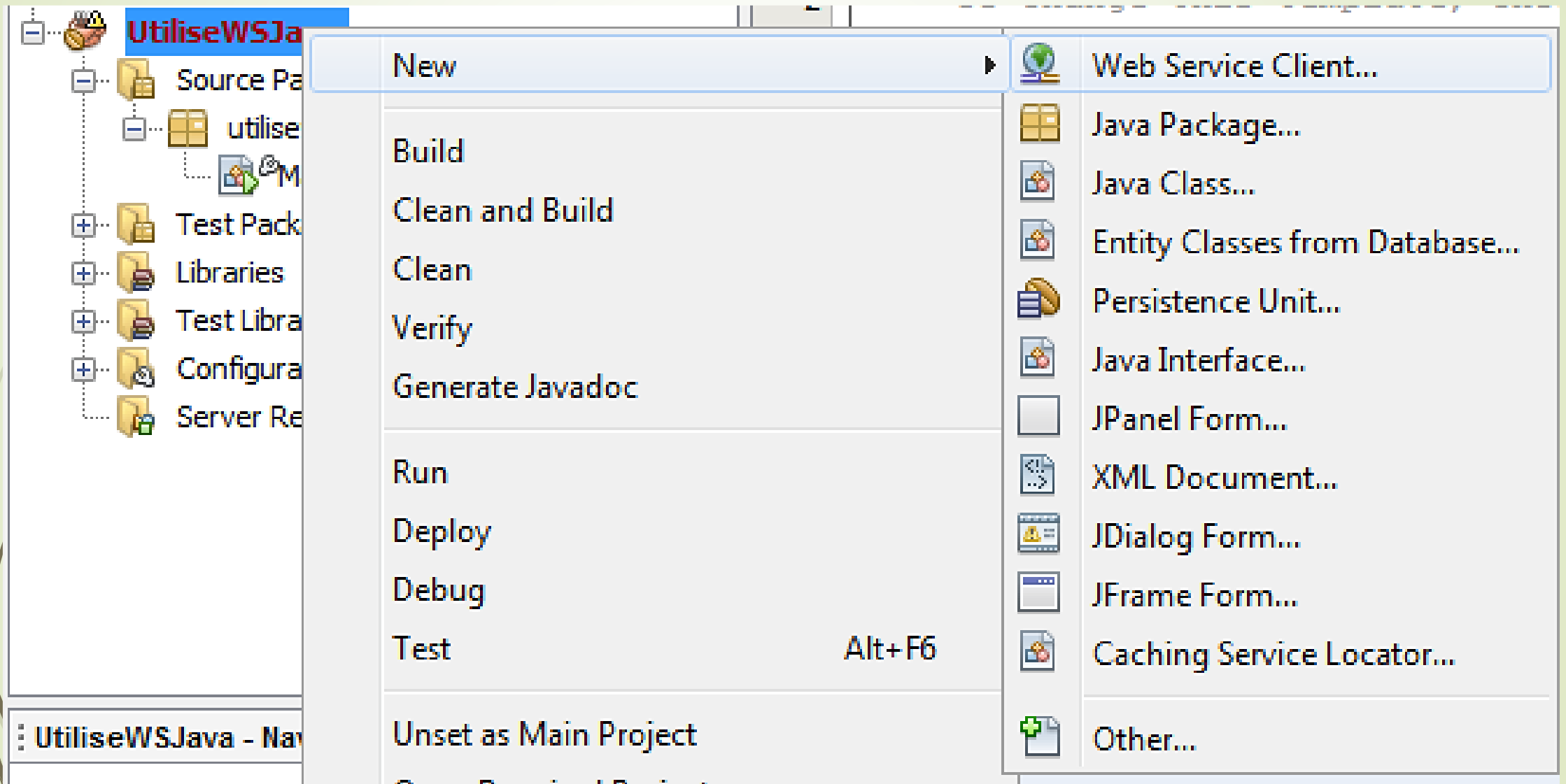
Libraries Folder:

Browse...

Different users and projects can share the same compilation libraries
(see Help for details).

☒ Set as Main Project

Ajout d'une Ref. de service



Ajout d'une Ref. de service

New Web Service Client

Steps

1. Choose File Type
2. **WSDL and Client Location**

WSDL and Client Location

Specify the WSDL file of the Web Service.

☐ Project:

☐ Local File:

☒ WSDL URL:

☐ IDE Registered:

Specify a package name where the client java artifacts will be generated:

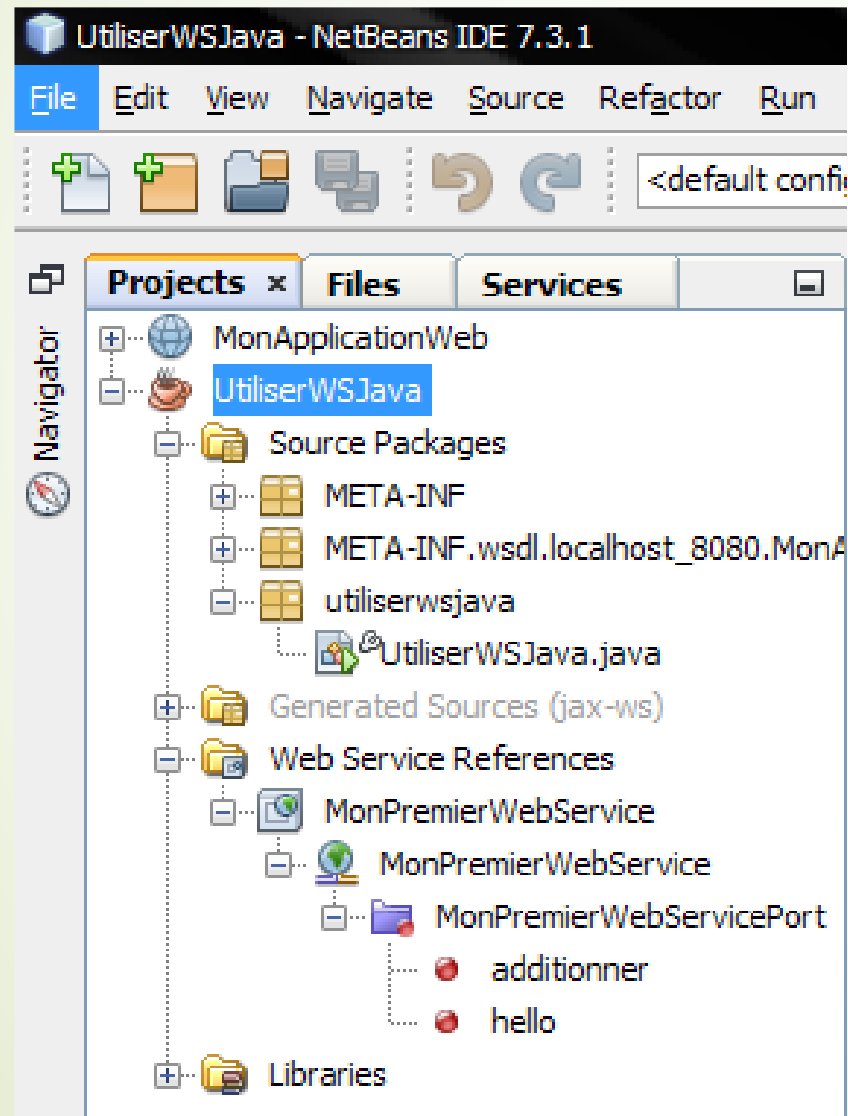
Project:

Package:

☒ Generate Dispatch code

< Back Next > **Finish** Cancel Help

Le projet mis à jour



Utiliser le WS

The screenshot displays an IDE interface with a project structure on the left and a code editor on the right.

Project Structure (Left):

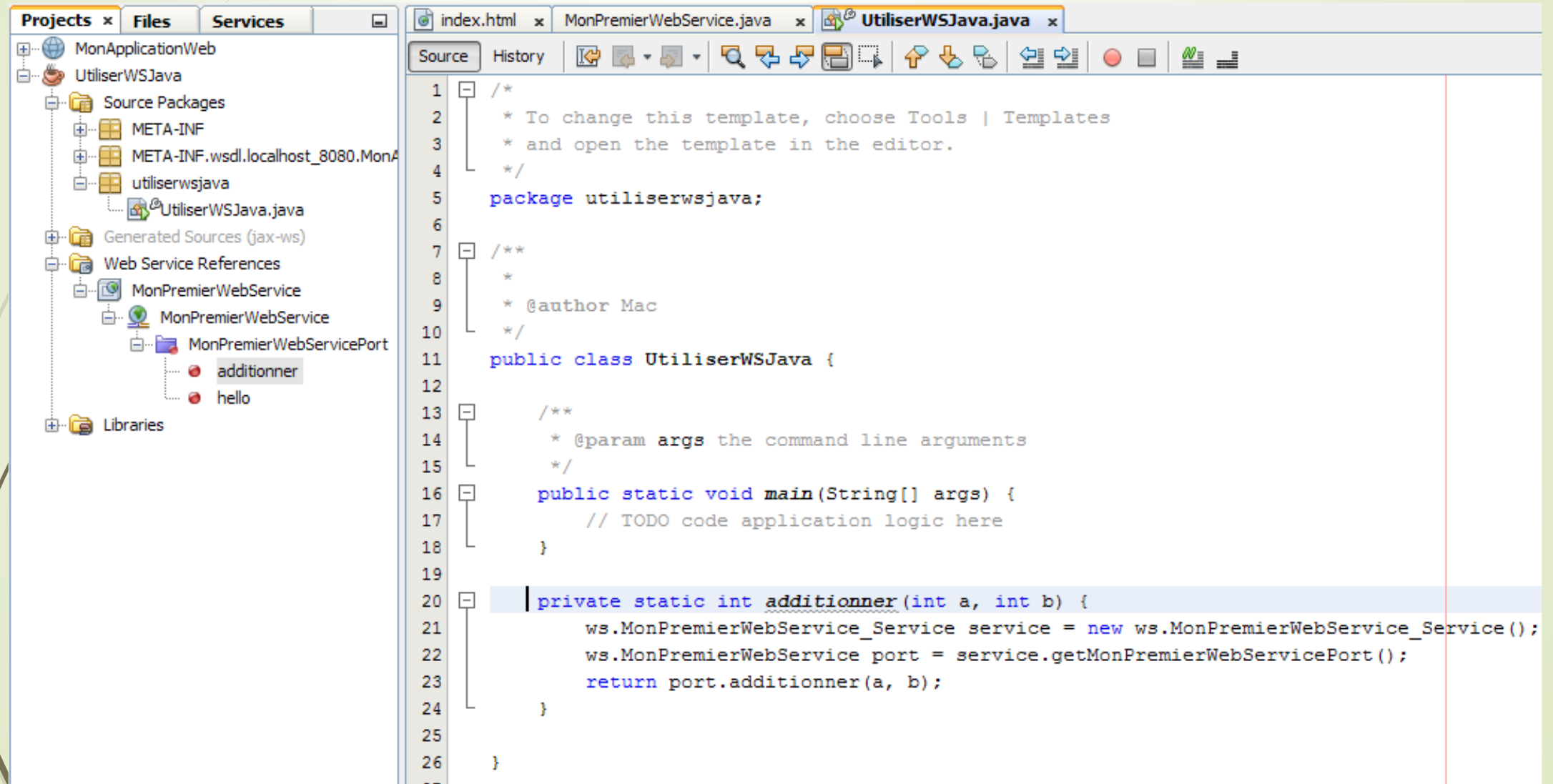
- CreationWSJava
- EJBWS
- UtiliseWSJava
 - Source Packages
 - utilisewsjava
 - Main.java
 - Test Packages
 - Generated Sources (jax-ws)
 - Web Service References
 - EssaiJavaWebService
 - EssaiJavaWebServicePort
 - additionner
 - hello
 - Libraries
 - Test Libraries

Code Editor (Right):

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package utilisewsjava;
6
7  /**
8   *
9   * @author lacomme
10  */
11 public class Main {
12
13     /**
14      * @param args the command line arguments
15      */
16     public static void main(String[] args) {
17         // TODO code application logic here
18
19
20
21
22     }
23 }
24
```

A red arrow points from the **additionner** port in the **EssaiJavaWebServicePort** to the **main** method in the code editor.

Utiliser le WS



The screenshot displays an IDE interface with a project explorer on the left and a code editor on the right.

Project Explorer (Left):

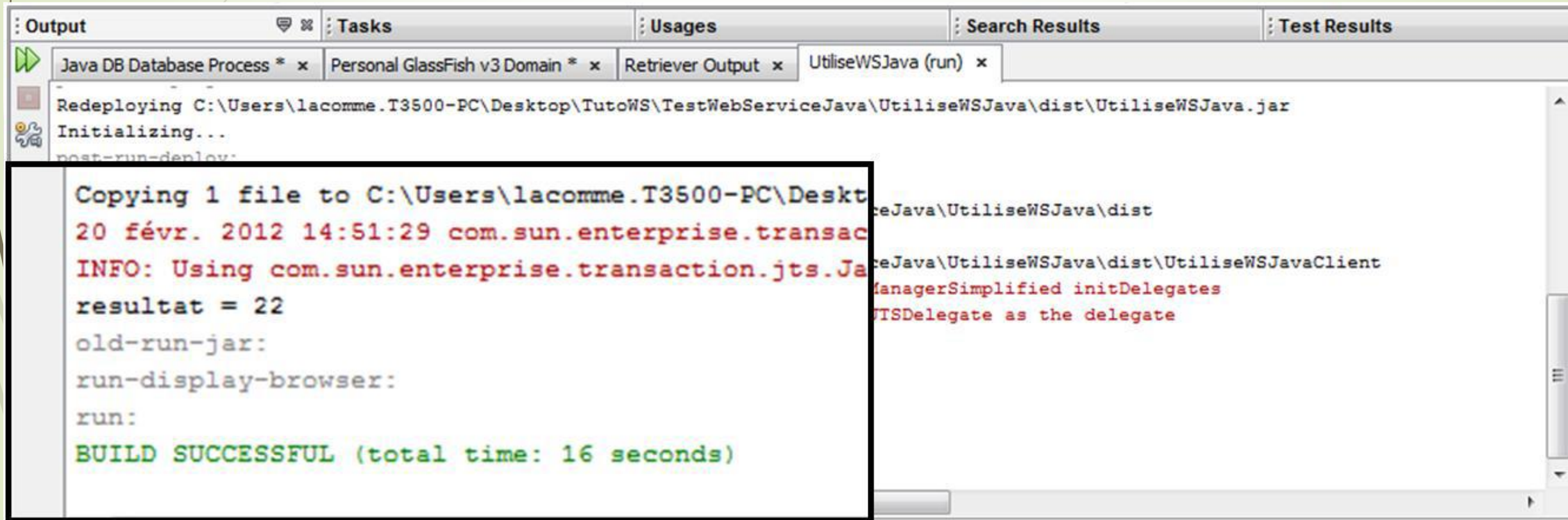
- MonApplicationWeb
 - UtiliserWSJava
 - Source Packages
 - META-INF
 - META-INF.wsdl.localhost_8080.MonA
 - utilisrwsjava
 - UtiliserWSJava.java
 - Generated Sources (jax-ws)
 - Web Service References
 - MonPremierWebService
 - MonPremierWebService
 - MonPremierWebServicePort
 - additionner
 - hello
 - Libraries

Code Editor (Right):

The code editor shows the file `UtiliserWSJava.java` with the following content:

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package utilisrwsjava;
6
7  /**
8   *
9   * @author Mac
10  */
11  public class UtiliserWSJava {
12
13      /**
14       * @param args the command line arguments
15       */
16      public static void main(String[] args) {
17          // TODO code application logic here
18      }
19
20      private static int additionner(int a, int b) {
21          ws.MonPremierWebService_Service service = new ws.MonPremierWebService_Service();
22          ws.MonPremierWebService port = service.getMonPremierWebServicePort();
23          return port.additionner(a, b);
24      }
25
26  }
```

Vérifier le résultat



The screenshot shows the 'Output' window of an IDE with several tabs. The 'UtiliseWSJava (run)' tab is active, displaying the following text:

```
Java DB Database Process * x Personal GlassFish v3 Domain * x Retriever Output x UtiliseWSJava (run) x
-
Redeploying C:\Users\lacomme.T3500-PC\Desktop\TutoWS\TestWebServiceJava\UtiliseWSJava\dist\UtiliseWSJava.jar
Initializing...
post-run-deploy:
Copying 1 file to C:\Users\lacomme.T3500-PC\Desktop\TutoWS\TestWebServiceJava\UtiliseWSJava\dist
20 févr. 2012 14:51:29 com.sun.enterprise.transaction
INFO: Using com.sun.enterprise.transaction.jts.Ja
resultat = 22
old-run-jar:
run-display-browser:
run:
BUILD SUCCESSFUL (total time: 16 seconds)
```

A black rectangular box highlights the following portion of the output:

```
Copying 1 file to C:\Users\lacomme.T3500-PC\Desktop\TutoWS\TestWebServiceJava\UtiliseWSJava\dist
20 févr. 2012 14:51:29 com.sun.enterprise.transaction
INFO: Using com.sun.enterprise.transaction.jts.Ja
resultat = 22
old-run-jar:
run-display-browser:
run:
BUILD SUCCESSFUL (total time: 16 seconds)
```



Éléments à retenir

1. **Un serveur** → une application web
2. **Déployer** → Glassfish
3. **Rôle de Glassfish**
 - ✓ héberger
 - ✓ administrer
 - ✓ simplifier
4. **Un client** → programme Java presque « standard »

Éléments à retenir

1. **Un serveur** → une application web
2. **Déployer** → Glassfish
3. **Rôle de Glassfish**
 - ✓ héberger
 - ✓ administrer
 - ✓ simplifier
4. **Un client** → programme Java presque « standard »



PARTIE IV

Réalisation d'un client de conversion de monnaie



PARTIE V

Réalisation d'un client de gestion d'adresse IP



PARTIE VI

Développer des services Web REST avec JAVA



PARTIE VII

PROJET

Le protocole BEEP

BEEP (Blocs Extensible Exchange Protocol) est un nouveau protocole de transport, défini par la **RFC 3080** publié en mars 2001, à l'instar de **HTTP**, BEEP définit un ensemble de messages codés en caractères ASCII, qui sont transportés par un protocole réseau **TCP/IP**. Par contre, BEEP est un protocole d'application, qui est plus adapté à la gestion des connexions et aux interactions asynchrones que peut l'être HTTP. De plus, ce protocole supporte plus d'échanges que le simple paradigme Requête/Réponse.

Notamment la traçabilité de l'activité (**Logging**) et les interactions points à points (**Peer-to-Peer**). Mais le caractère le plus intéressant de BEEP est sa capacité à supporter le **multiplexage**.

Un message BEEP peut être de trois types. Chacun étant défini par le modèle de réponse du serveur :

- **type Message/Reply** lorsque le serveur exécute une tâche et renvoie une réponse positive
- **type Message/error** quand le serveur n'a pu réaliser la tâche demandée et retourne un message d'erreur au client
- **type Message/answer** lorsque le serveur renvoie des données au client, suivi par un indicateur de terminaison