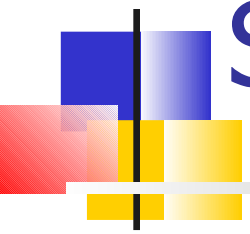


Services Web

Plan : généralités, REST, XML-RPC, services Web (XML, SOAP, UDDI), démo, conclusion

Applications Web distribuées (1)

- Besoin : permettre à un client d'exécuter une opération sur un serveur **distant** via l'Internet
  - Interopérabilité entre le client et le serveur
- Implications :
  - Utiliser un langage commun pour transmettre des données nécessaires = masquer les différences au niveau de la représentation des données
  - S'abstraire des problématiques liées au transport

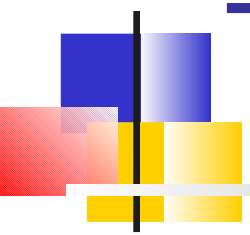
Applications Web distribuées

- Solution pratique : se baser sur les standards du WEB
  - HTTP
    - S'affranchir d'un protocole propriétaire (par exemple Sun Java RMI)
    - Ne pas implémenter son propre protocole d'invocation (socket...)
  - MIME : transport de données multimédia
  - XML : masquer les différences au niveau de la représentation des données
  - URI : identification unique des ressources (services, hotes, documents)

Comment développer une applications Web distribuée?

- En se basant sur l'architecture Web existante :
  - Le Web permet d'accéder à des documents
  - → Utiliser **l'approche REST** : les applications publient sur le Web leur résultats sous la forme d'un document
- Utiliser un mécanisme de programmation distribué moderne, les RPC, et les adapter pour le WEB (XML-RPC)
  - **Appel de procédure distante** ou RPC (*Remote Procedure Call)*
  - Une RPC permet d'exécuter une opération (procédure) sur un serveur **distant**

REST - Representational State Transfer



REST - Introduction

- Ces dernières années, le Web s'est développé de façon incrémentale sans nécessairement se baser sur une référence architecturale
- Ces principes ont été mis à jour et documentés a posteriori
- → REST correspond à cet effort de documentation
- → REST n'est pas un standard mais il prescrit l'utilisation de standards : HTML, HTTP, URL, MIME...

REST – Representational State Transfer

- REST est un modèle architectural :
  - Un client émet une requête HTTP pour obtenir en échange un document HTML (= représentation de la ressource)
  - Après réception du document, le client change d'état
  - → Notion de transfer **d'état** (state)
  - → D'où l'acronyme REST : Representational State Transfer

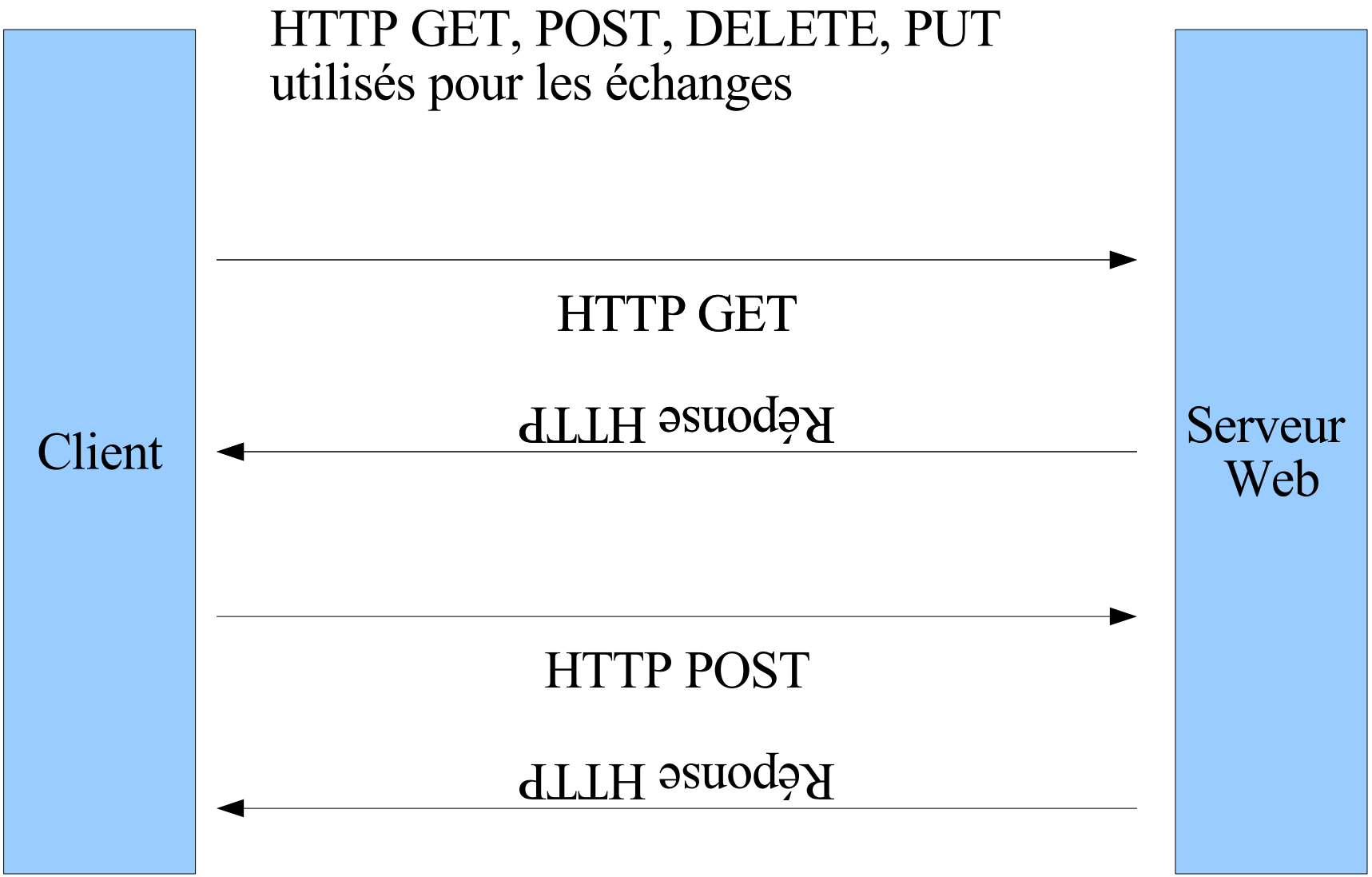
Le WEB – les bases (1/2)

- Tout d'abord, quelques rappels ...
  - HTTP est sans état
    - → aucune session, transaction
    - → l'ensemble des informations nécessaires sont contenues dans les messages
  - Chaque ressource est accessible à partir d'une URI (*Uniform resource Identifier*) unique
  - Les clients évoluent en suivant des liens hypertexts (GET) et en soumettant des représentations (POST)
- REST hérite de ces propriétés

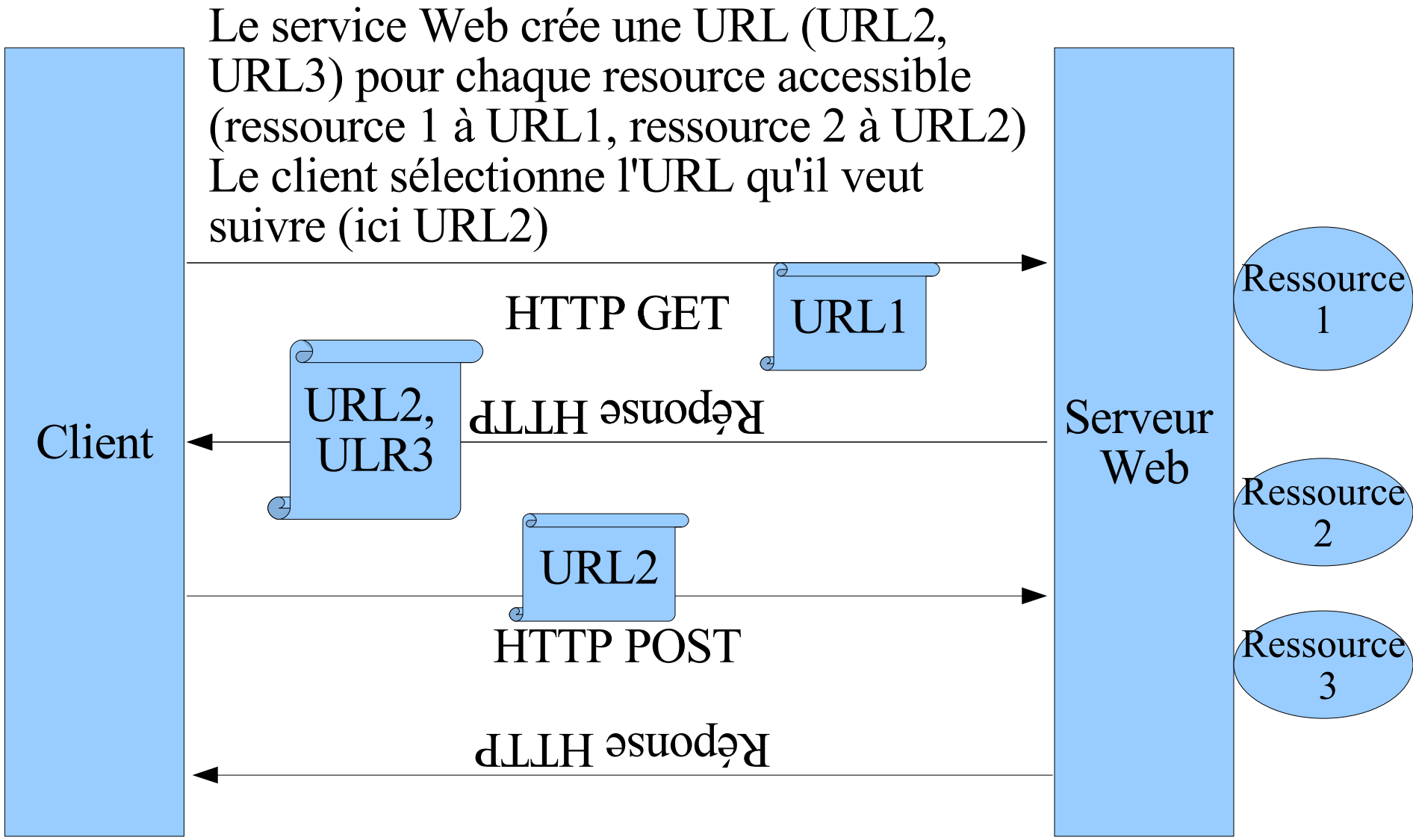
Le Web – les bases (2/2)

- Il est déjà possible avec HTTP d'effectuer des actions
  - Pour s'en convaincre, il suffit d'aller sur un site quelconque de vente en ligne
- Question : comment concevoir un service Web à la sauce REST, c'est-à-dire compatible avec les outils du Web

Web service à la sauce REST



Web service à la sauce REST



Rêgles de conception d'un service Web REST

- Créer une ressource logique pour chaque service
- Identifier cette ressource : créer l'URI associée
- Envoyer cette URI dans les réponses renvoyées au client
- L'URI est placée dans l'entête HTTP car les composants réseaux (cache, proxy, parre-feux) prennent leur décision en annalysant l'entête (pas toujours le corps)

Exemple : catalogue mettant en ligne des objets à vendre

Le client peut accéder à la ressource object 1 via http://www.catalogue.com/parts/object1

et la ressource object2 via http://www.catalogue.com/parts/object2

Récapitulatif -REST

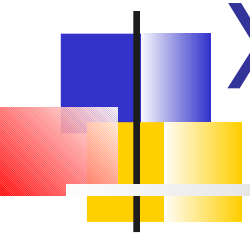
- La requête REST est simplement passée dans un paramètre HTTP
  - Très proche de la philosophie d'HTTP
  - REST est implémenté dans la plupart des outils WEB ..
    - → interopérabilité obtenue non pas en adhérant à un standard mais en se basant sur le fait que les outils du Web implémentent déjà les fonctionnalités REST
    - → REST est pied et poing lié (HTTP)
- Une approche orientée document → en droite lignée avec la philosophie Web

Récapitulatif REST

- REST c 'est un peu comme Mc Donald :
  - C'est simple (simpliste?),
  - C'est rapide
    - HTTP est sans état → faible consommation mémoire, mise au point plus simple, moins de cas à traiter
    - URI uniques = mise en cache possible donc meilleure montée en charge
  - Ce n'est pas très varié (HTTP seulement)

Petite aparté

- SOAP utilise le terme “document style” pour signifier qu'il n'y a pas d'appel de méthode
- Le service est représenté au travers de la manipulation de documents/objets
- REST + XML = SOAP document style

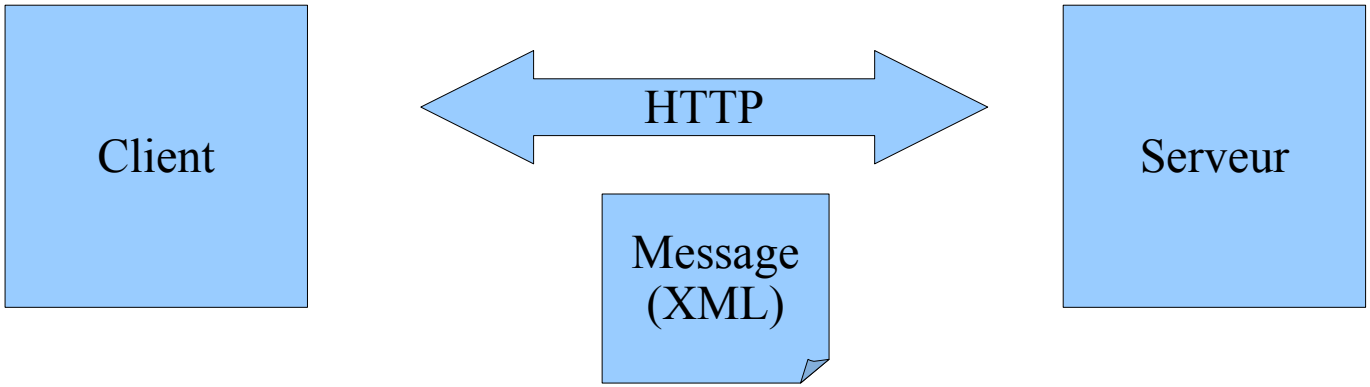


XML-RPC

Françoise Sailhan

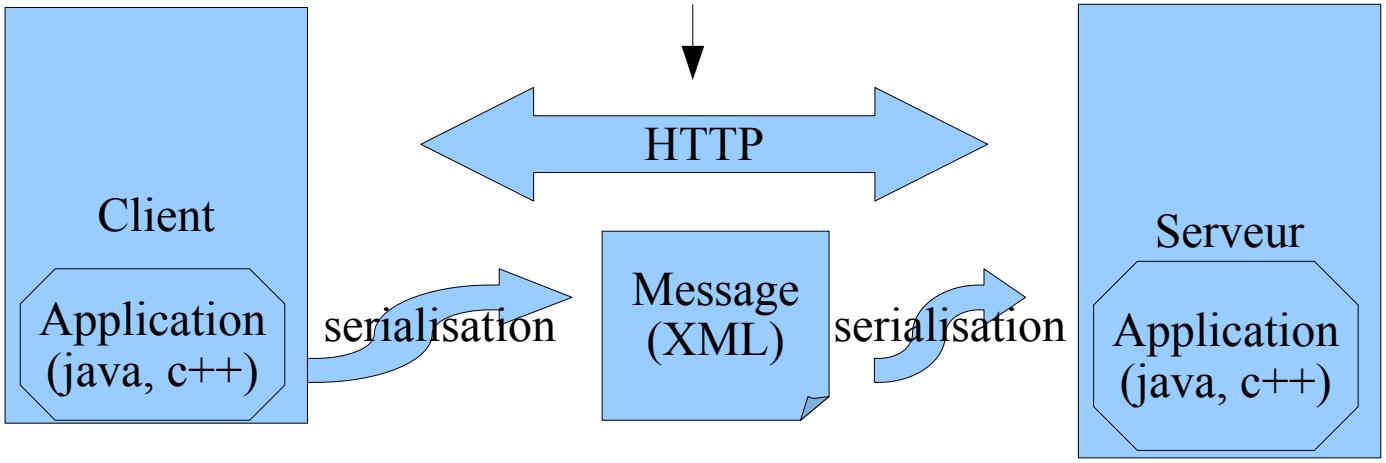
XML RPC - Introduction

- Un protocole pour gérer des appel à des procédures distantes avec :
  - Un appel représenté sous la forme d'un document **XML** (*serialisation*, *marshaling*, ou encodage)
  - Un transport effectué par le protocole **HTTP**



XML-RPC - architecture

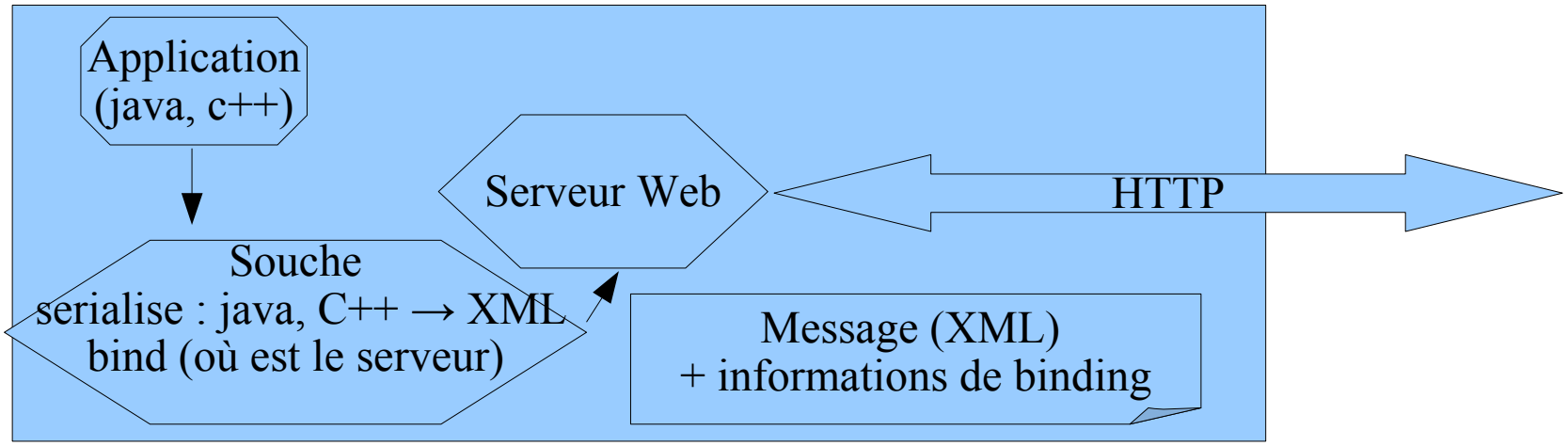
- Architecture client/serveur
  - Un client (code appelant) exécute un RPC sur le serveur
  - Un serveur XML-RPC qui analyse (*parse)* le document XML représentant l'appel, gère l'appel, renvoie (si nécessaire) le résultat au client



Comment représenter, coder, décoder les données échangées?

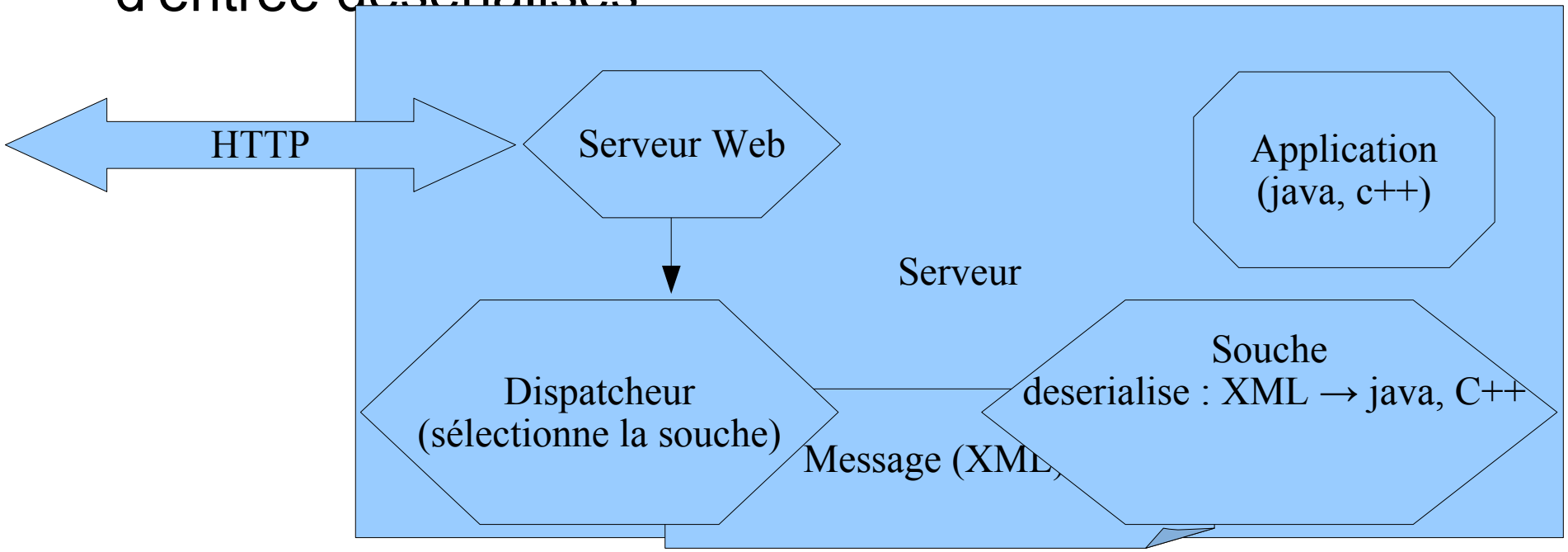
Traitements au niveau du client XML-RPC

- Le code appelant du client génère un appel local, comme si la procédure invoquée était locale
- Cet appel est intercepté (*wrapper*) puis transformé en appel distant par une souche (en anglais stub) en
  - *Sérialisant* (génèrant un document XML)
  - *Bindant* (où se trouve la RPC distante à invoquer?)
  - Fournissant au serveur le doc XML + les informations de binding



Traitement au niveau du serveur XML-RPC

- Le serveur Web :
  - reçoit le message HTTP, en extrait le document XML qui est dispatché à la souche qui décode le document XML, invoque la méthode java en passant le(s) paramètre(s) d'entrée déserialisés



Souche XML-RPC (1/2)

- Objectif : générer automatiquement les souches
- En utilisant une IDL (Interface Description Language) qui décrit en XML l'interface du service de façon **standard**, **abstraite**, **indépendamment du langage** utilisé (java, c++, ...)
- Un compilateur d'interface est utilisé pour générer la souche du client et du serveur

Souche XML-RPC (2/2)

- L'IDL définit le nom du service, le nom des procédures, les entrées/sorties de ces procédures
- Le programmeur implémente
  - les procédures du serveur
  - Les procédures appelantes du client
- La souche est générée automatiquement de façon à effectuer la transformation des données à envoyer, binder et expédier

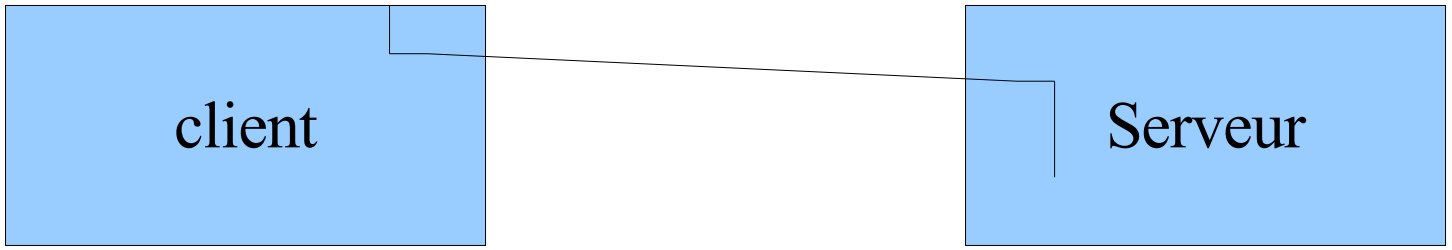
Protocole XML-RPC (1/2)

- Les messages XML-RPC sont encapsulés dans des requêtes HTTP-POST
- Le corps du message est un document XML
- Il existe aucune **spécification formelle officielle** de ce document XML : pas de DTD, pas de schémas XML
- En quelque sorte XML-RPC est une norme industrielle :
  - les implémentations sont toujours plus diverses (c, c++, objective c, Java, perl, Delphi, Python, TCL ...)
  - XML-RPC est léger (une spécification nécessaire?)

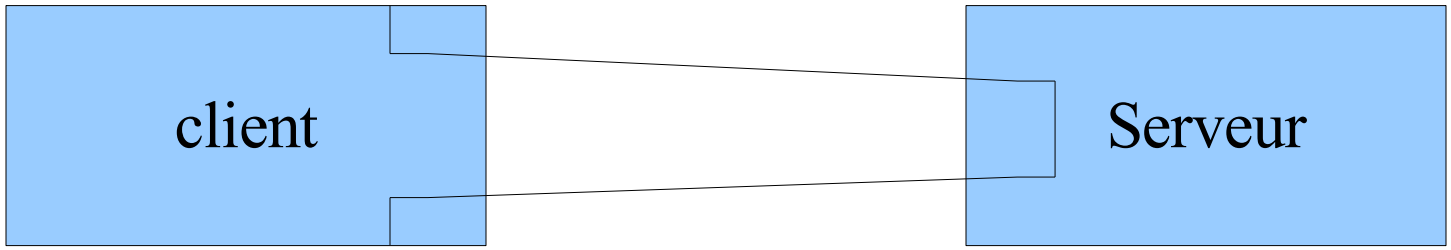
Protocole XML-RPC

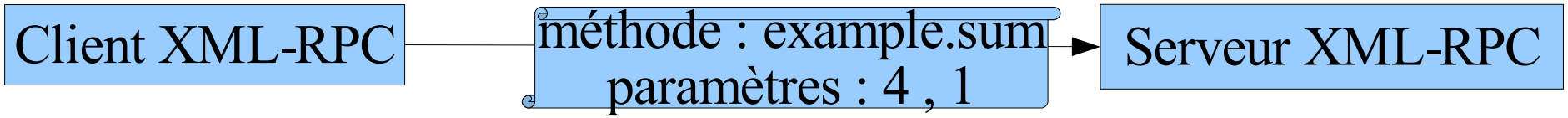
XML-RPC supporte une communication :

one way (dans un sens : invocation d'une RPC, sans résultat attendu)



Two ways (deux sens) **synchrone**





Requête XML-RPC

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: 192.168.2.3
Content-Type: text/xml
Content-length: 181
<?xml version="1.0"?>
<methodCall>
<methodName>example.sum</methodName>
<params>
  <param> <value><i4>4</i4></value>
    </param>
        <param><value><i4>1</i4></value>
    </param>
</params>
</methodCall>
```

Requête RPC-XML

Entête HTTP

Corps du message (payload) Document XML

Structure du corps du message (1/2)

```
<methodCall>
  <methodName>example.getShortInteger</methodName>
...
</methodCall>
```

- methodCall : élément racine du document XML contenant
  - Le sous élément methodName correspond au nom de la méthode
  - Remarque : l'interprétation du nom est laissée à la discrétion du serveur (par exemple, nom du fichier contenant le script + nom de la méthode ou encore chemin vers le fichier + nom du ficher + nom de la classe + nom de la méthode)

Structure du corps du message (2/2)

```
<methodCall>
  <methodName>...</methodName>
<params>
   <param> <value><i4>41</i4></value></param>
   <param><value><i4>12</i4></value></param>
</params>
</methodCall>
```

- L'étiquette <params> regroupe les paramètres (<param>) de l'appel à procédure
  - <params> est optionnelle (pas d'étiquette si aucun variable en entrée)
  - Le sous élément value caractérise la valeur d'un paramètre

Valeurs des paramètres

Les types primitifs des valeurs sont :

- des types simples (par exemple integers)
- Deux types complexes agrégés : array (tableau) et structure

- XML implique que des structures complexes peuvent être transmises, gérées et retournées
- En revanche, **XML-RPC ne supporte que ces types primitifs**

Types Simples

- int ou i4: entier signé de 32 bits, ex. -12
- double : nombre flottant, -1,4
- string : chaine de caractères ASCII pouvant potentiellement contenir NULL bits, “ff ee”
- boolean : soit 1 soit 0
- dateTime.iso8601 : date (norme iso 8601)
  - Ex . 20100706T14:23:12 pour 14h23mn12s le 6 juillet 2010, T pour Time
- base64 : données binaires avec un encodage en base 64

Types complexes - Array

- Tableau à une dimension dont les éléments peuvent être de types différents
- Est possiblement récursif, c'est-à-dire qu'il contient d'autres tableaux (ou des structures)
- Exemple : tableau contenant deux éléments

```
<array>
  <data>
    <value><i4>12</i4></value>
    <value><boolean>0</boolean></value>
  </data>
</array>
```

Type complexe - Structure

- Collection d'éléments (clé, valeur)
  - La clé est un string, la valeur est de n'importe quel type
  - Est possiblement récursif
- Exemple : structure contenant deux éléments

```
<struct>
  <member>
     <name>lowerBound</name>
     <value><i4>18</i4></value>
   </member>
   <member>
     <name>upperBound</name>
     <value><i4>139</i4></value>
  </member>
</struct>
```

Un exemple de mapping entre XML-RPC et JAVA

Type XML-RPC ↔ Type Java

int ↔ java.lang.Integer

double ↔ java.lang.Double

string ↔ java.lang.String

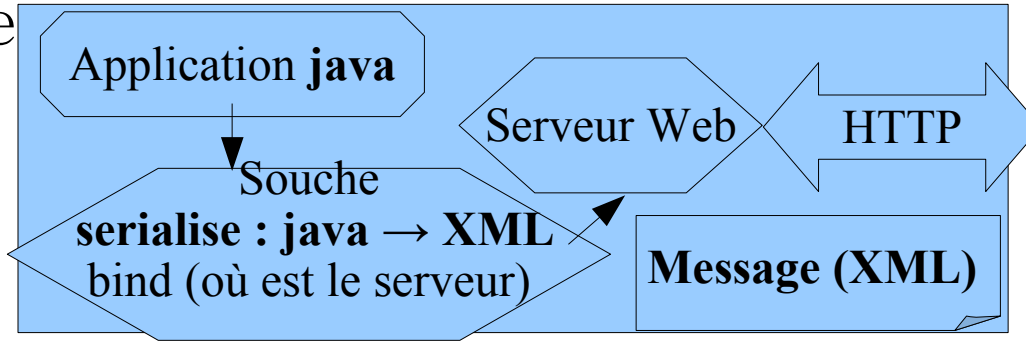
boolean ↔ java.lang.boolean

dateTime.iso8601 ↔ java.util.Date

base64 ↔ byte []

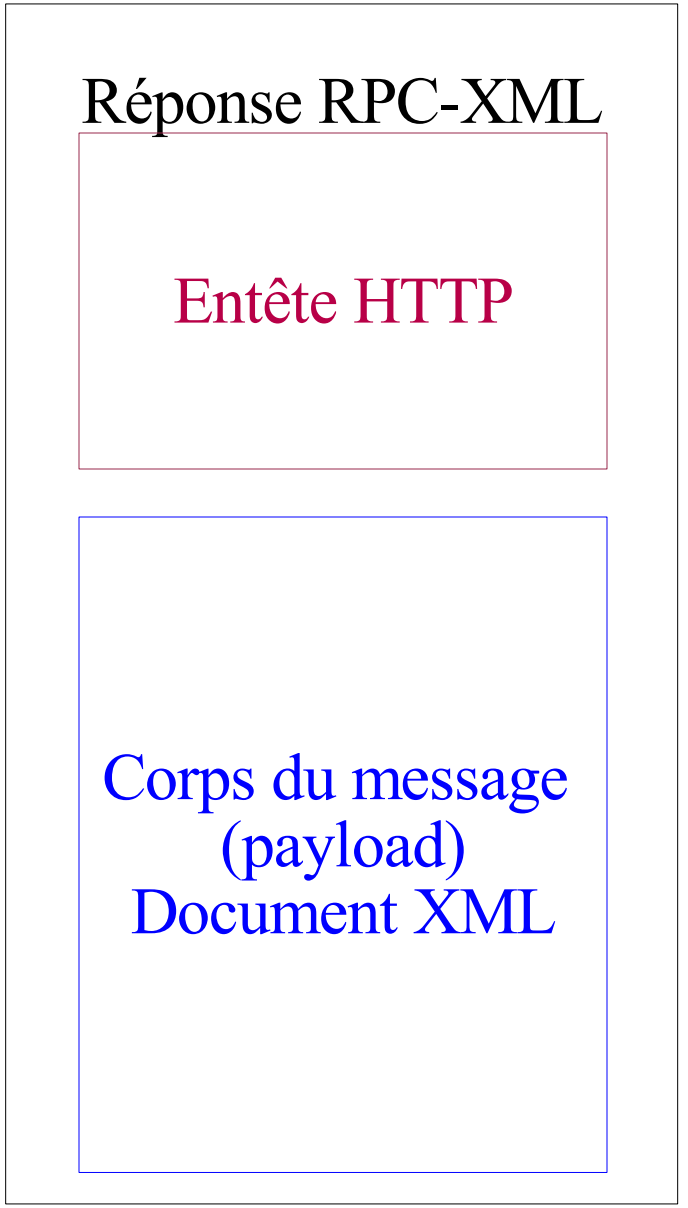
array ↔ java.util.Vector

struc ↔ java.util.Hashtable



Réponse XML-RPC

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 2010 19:55:08 GMT
Server: server.cnam.fr
<?xml version="1.0"?>
<methodResponse>
<params>
<param>
<value><i4>5</i4></value>
</param>
</params>
</methodResponse>
```



Fautes

- Deux types de fautes
  - Fautes au niveau HTTP
  - Fautes (exceptions) provenant du RPC

```
HTTP/1.1 200 OK
        ...
<?xml version="1.0"?>
<methodResponse><fault><value>
 <struct>
  <member><name>faultCode</name>
   <value><int>4</int></value></member>
  <member><name>fault for sum</name>
   <value><string>Too many parameters </string>
   </value></member>
</struct> </value></fault></methodResponse>
```

Bilan XML-RPC

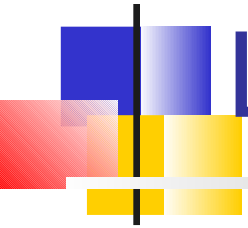
- Deux aspects importants
  - XML-RPC se base sur un unique protocole, HTTP
    - Or, HTTP est sans état
    - XML-RPC n'offre donc pas un support adapté aux transactions et au chiffrement (seulement HTTPS)
  - Malgré la “puissance d'XML”, XML-RPC définit un sous-ensemble restreint et non extensible de types
- En pratique, XML-RPC est léger et répond à la plupart des besoins
- XML-RPC est gelé au profit de SOAP

Les services Web

Jeremy Fierstone
Email : fierston@essi.fr

SAR5 – Novembre 2002

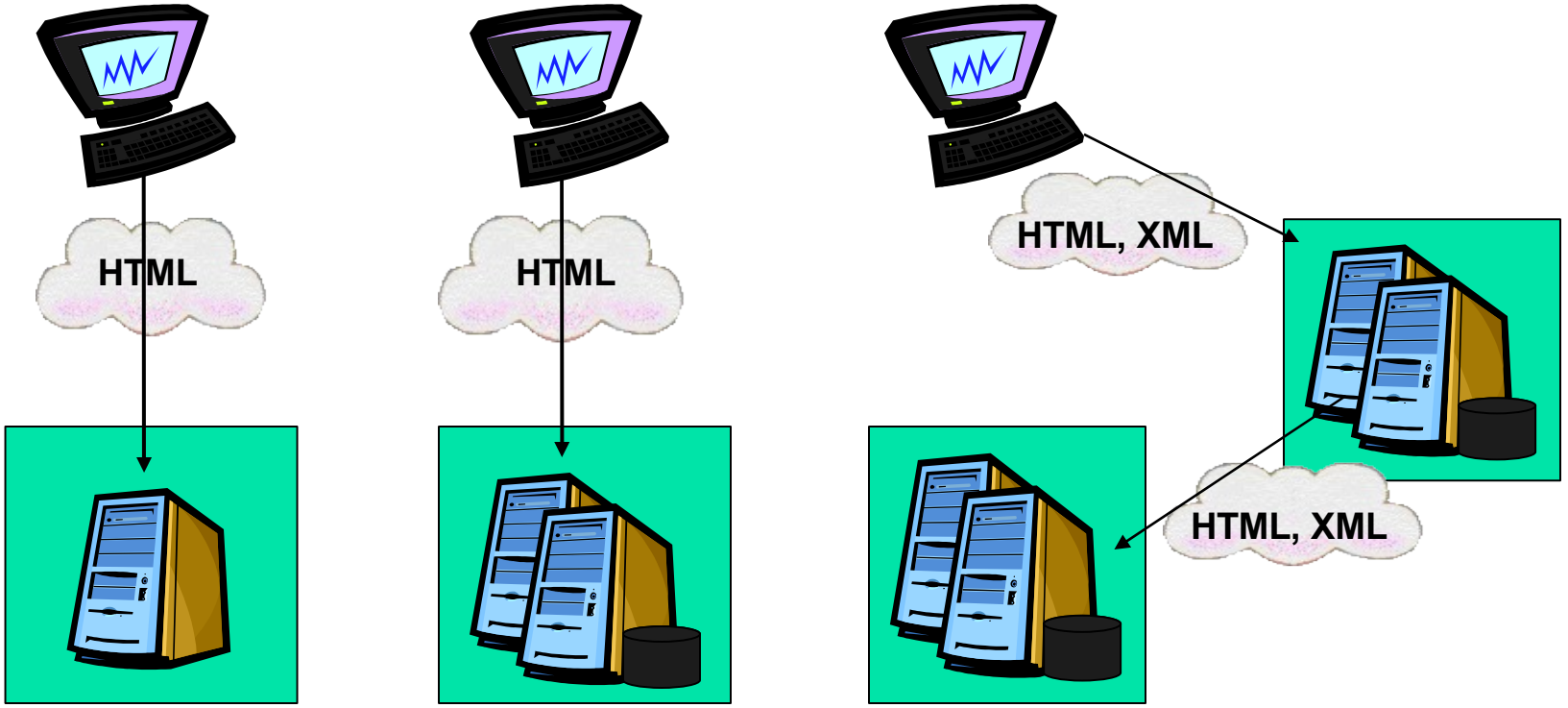
Merci à Mireille Blay-Fornarino, Didier Donsez
Michel Riveill, Microsoft, Sun … pour leurs slides



Les services Web

Généralités

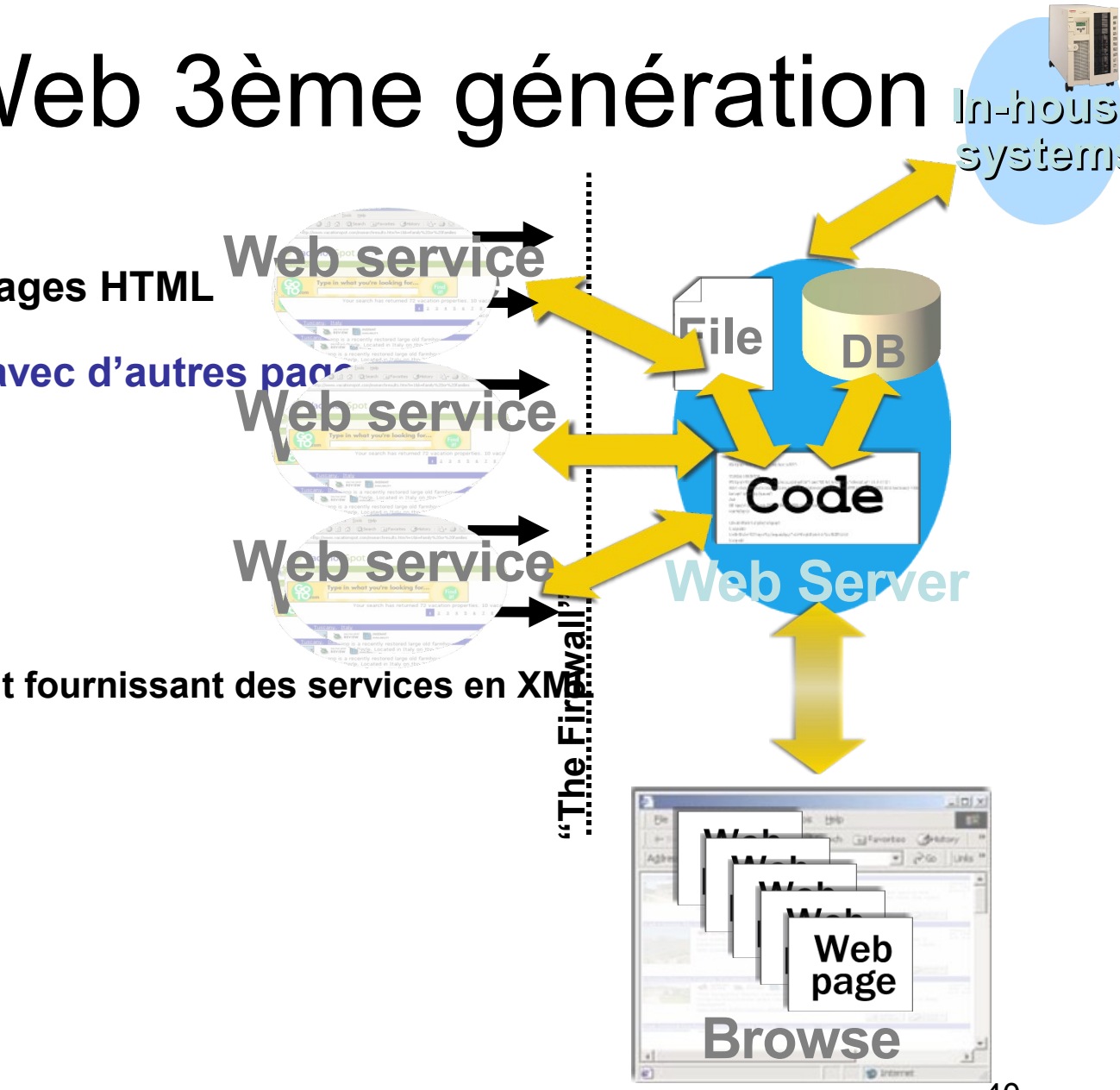
Evolution du Web



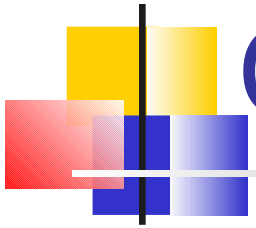
Le Web 3ème génération

Aujourd'hui
Un site Web fournie des pages HTML
- pas de structure
- impossible à fusionner avec d'autres pag

Demain
Un site Web est un composant fournissant des services en XML
- structure / sémantique
- fusion possible



"Dynamic Pages"

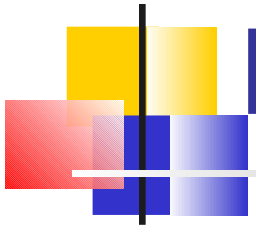


Quels objectifs ?

Remplacer les protocoles actuels (RPC,DCOM,RMI) par une approche entièrement ouverte et interopérable, basée sur la généralisation des serveurs Web avec scripts CGI.

Faire interagir des composants hétérogènes, distants, et indépendants avec un protocole standard (SOAP).

Dédiés aux applications B2B (Business to Business), EAI (Enterprise Application Integration), P2P (Peer to Peer).



Et plus concrètement ?

Une nouvelle technologie des objets distribués ?

Invocation distante des services Web : **SOAP** (~IIOP)

Description des services Web : **WSDL** (~IDL)

Enregistrement et découverte de services Web : **UDDI** (~NameService)

Basés sur des standards XML

Standards du W3C : XML, SOAP, WSDL

Standards industriels : UDDI, ebXML

Propriétaires : DISCO, WSDD, WSFL, ASMX, …

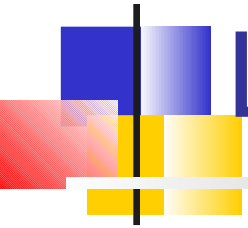
Implémentations actuelles :

Microsoft .Net

Sun JavaONE : J2EE + Web services (WSDP = JAXP,JAX-RPC,JAXM…)

Apache SOAP / Axis, IBM WSTK

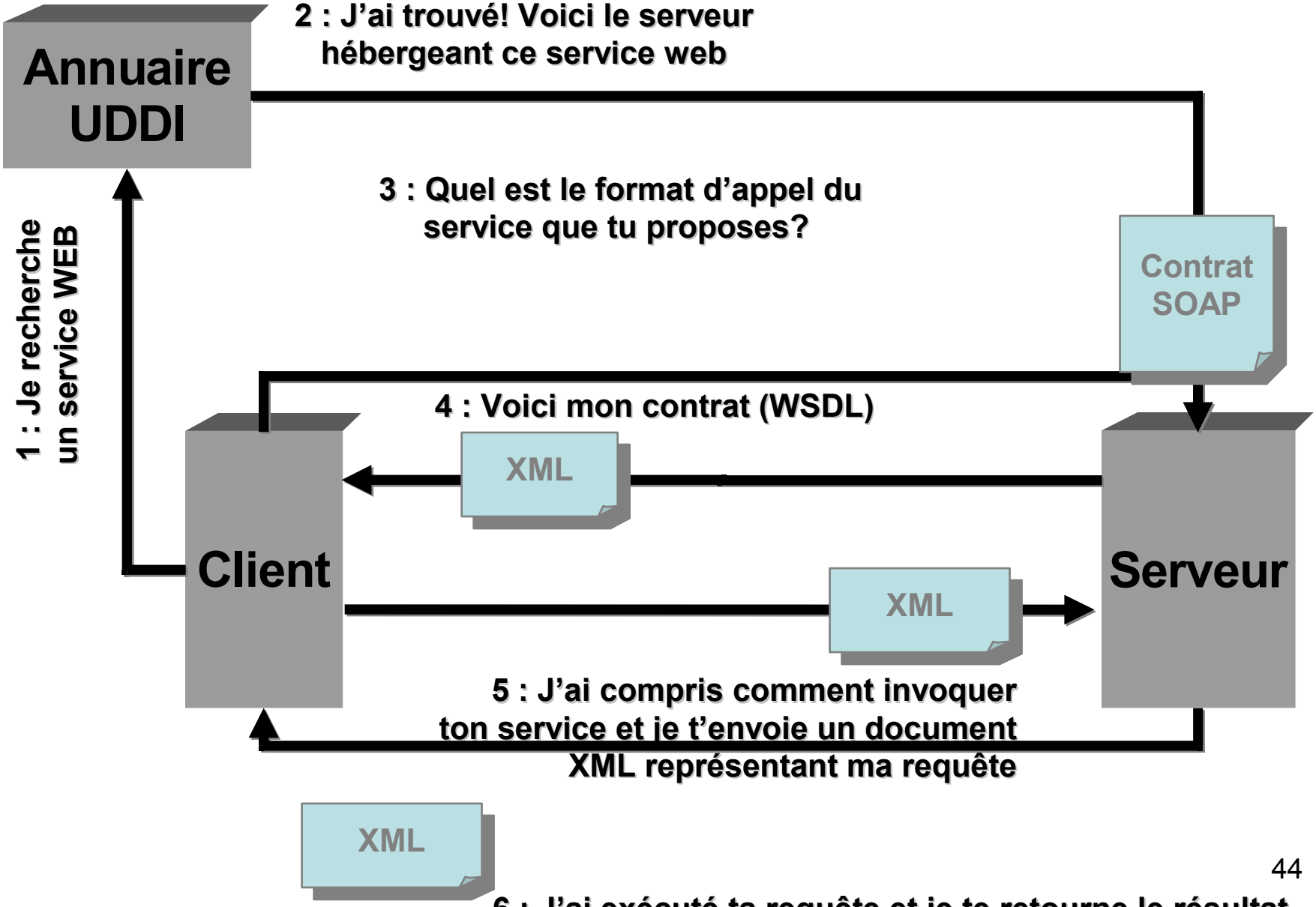
Oracle, Bea, Iona, Enhydra …

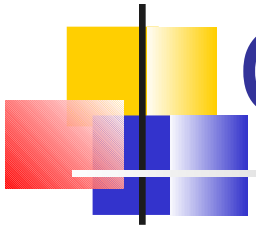


Les services Web

Architecture

Cycle de vie d'utilisation





Cycle de vie complet

Etape 1 : Déploiement du service Web

Dépendant de la plate-forme (Apache : WSDD)

Etape 2 : Enregistrement du service Web

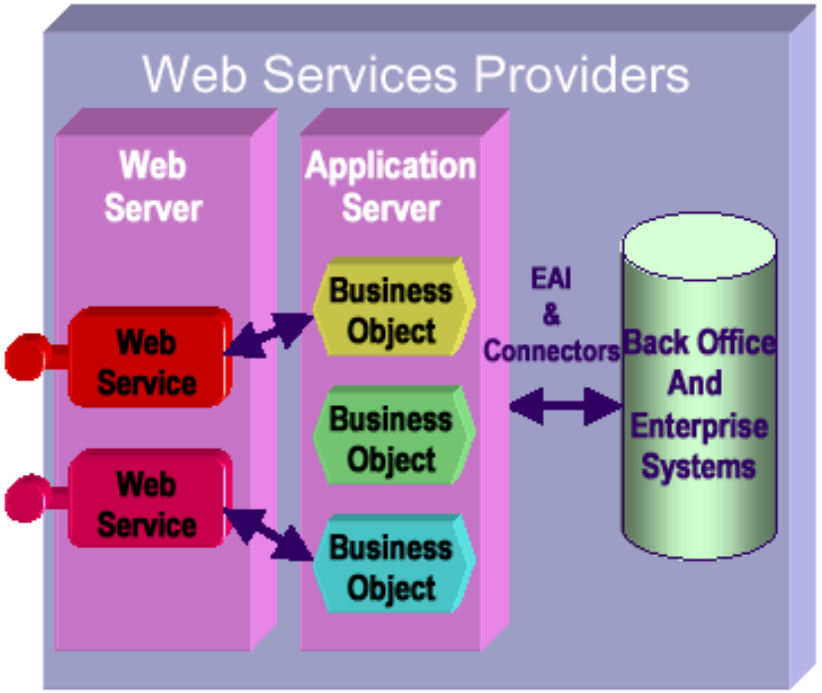
WSDL : description du service

Référentiels : DISCO (local), UDDI (global)

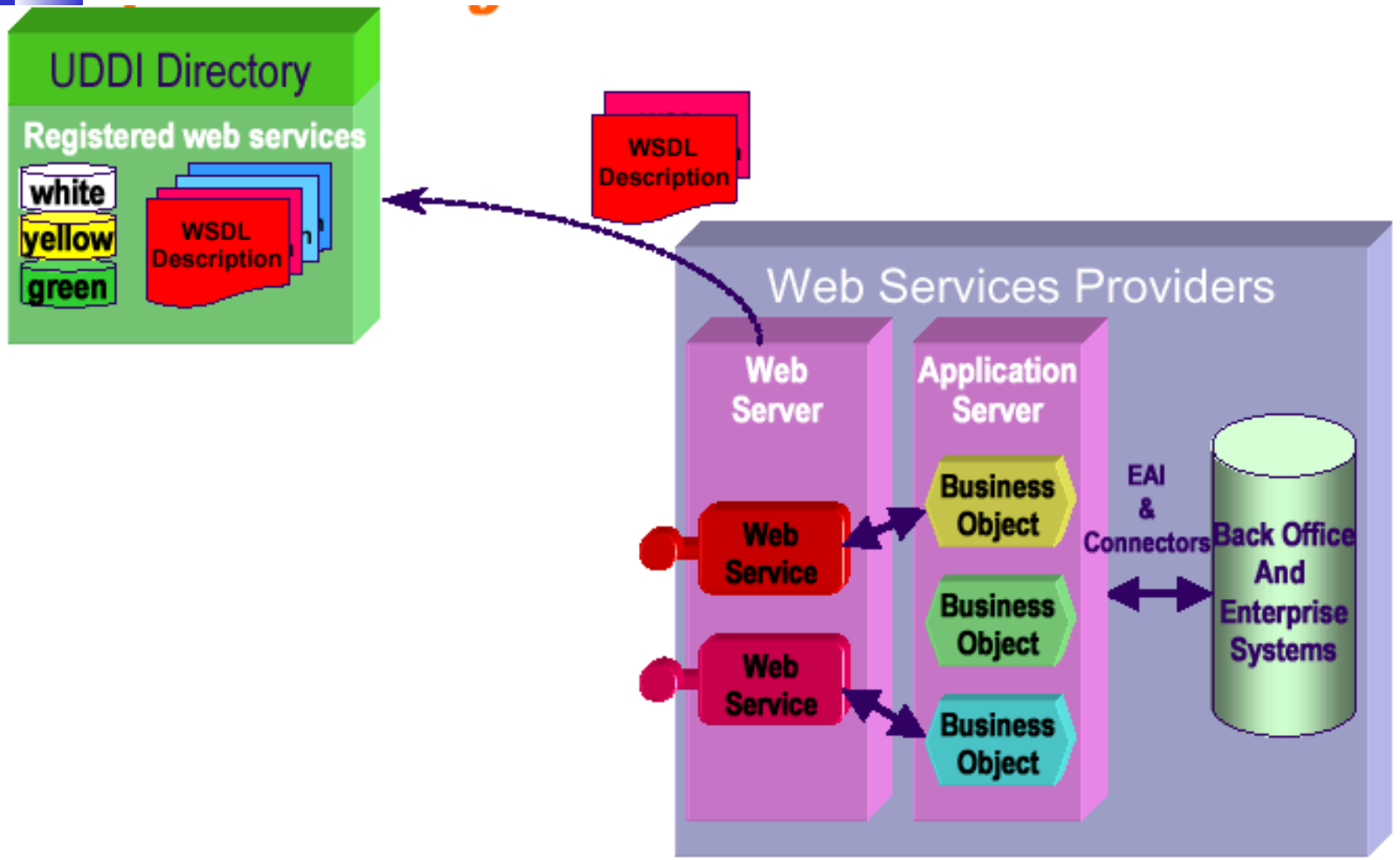
Etape 3 : Découverte du service Web

Etape 4 : Invocation du service Web par le client

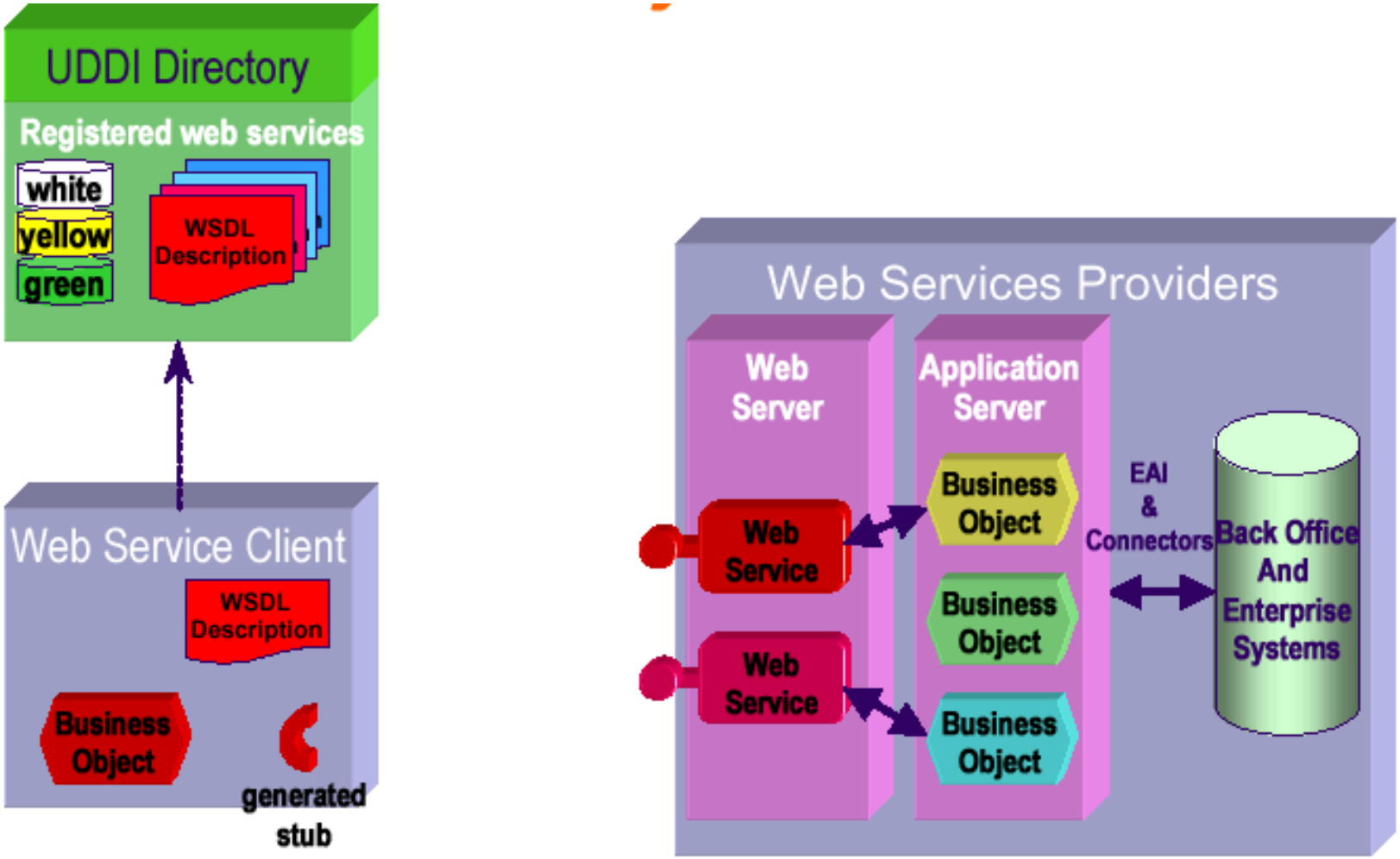
1: Déploiement du WS



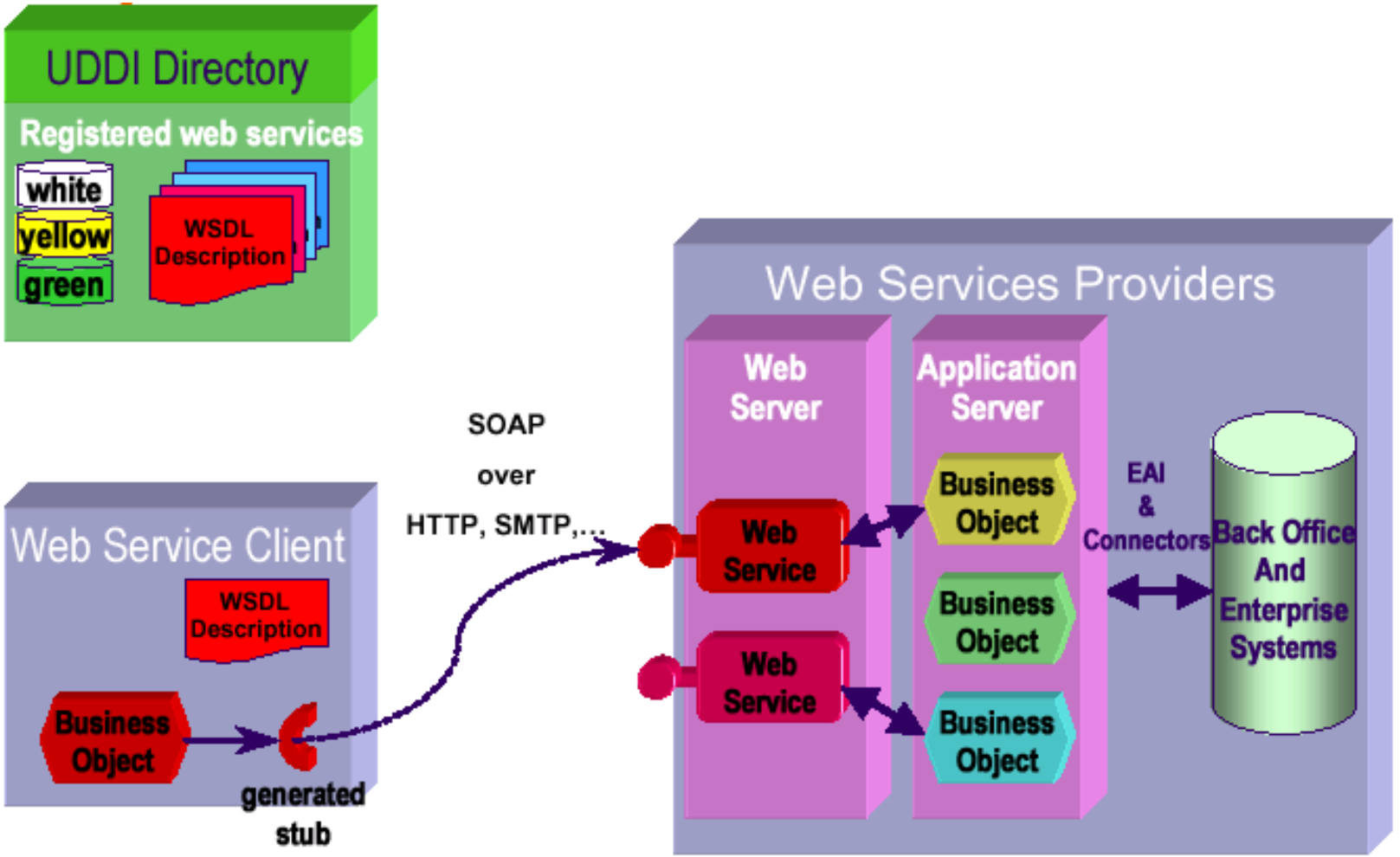
2: Enregistrement du WS

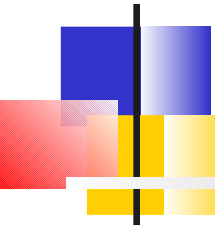


3: Découverte du WS



4: Invocation du WS





Les services Web

SOAP : Simple Object Access Protocol

Merci à Michel Riveill et Didier Donsez

Un peu d'historique (1/2)

Septembre 1999 : SOAP 0.9

Spécifications par Microsoft et DevelopMentor

Décembre 1999 : SOAP 1.0

Soumission des spécifications à l'IETF

Association de UserLand

Mai 2000 : SOAP 1.1 – Soumission au W3C

Nombreuses associations : IBM, HP, Lotus, Compaq, Intel …

XIDL : rapprochement de Corba

Septembre 2000

Groupe de travail W3C pour la standardisation de SOAP

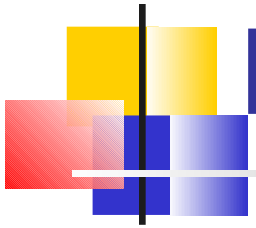
Corba/Soap Interworking RFP => *SCOAP*

Un peu d'historique (2/2)

Septembre 2003 : SOAP 1.2
annonce

Avril 2007 : SOAP 1.2
Entre temps, des notes et 7 recommendations concernant SOAP 1.2 soumises par le WG
Détail à http://www.w3.org/2000/xp/Group/

@Sailhan



Le Web et le client serveur

Proposition Web actuelle insuffisante

Autres plates-formes client / serveur

Java RMI

Java, multi-plateforme (JVM)

CORBA / IIOP

Multilangage, multi-plateforme, Multi-vendeurs, OMG

Installation « coûteuse » si on doit acheter un ORB

Mais les open-sources sont gratuits et souvent plus complet

www.objectweb.org

DCOM

multi-langages, plateforme Win32, Propriétaire Microsoft

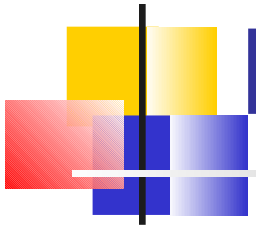
protocole orienté connexion

Échange de nombreux paquets pour créer/maintenir une session

Faible diffusion

Pas disponible sur MacOS, NT3.51, Win95, WinCE2

Coûteux sur UNIX, MVS, VMS ou NT



Le bilan…

Approche insatisfaisante :

Protocoles sophistiqués

Coût d'installation (faite par un administrateur, consomme des ressources : machines, personnels, …)

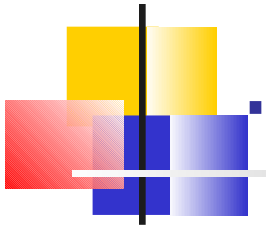
Difficile à porter sur d'autres plates-formes

Règles de fonctionnement strictes en environnement ouvert (le Net)

Environnement sécurisé (intérieur d'un intranet)

Incapacité à fonctionner en présence de pare-feu (utilisation impossible sur Internet)

Les nouvelles versions de CORBA peuvent ouvrir un port sur un pare-feu comme le port 80 d'HTTP



… et ses conséquences

Le Web a besoin d'un nouveau protocole

Multi-langages, multi-plateformes

Respectant les formats d'échanges du Web

Réponses et requêtes en XML

Facile à implémenter sur différents protocoles de transport

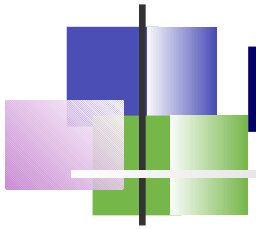
RPC, HTTP ou autre MOM

Permettant de franchir les « firewalls »

Avec une spécification non propriétaire garantie par un organisme indépendant

W3C

La réponse : SOAP (Simple Object Access Protocol)



La philosophie S.O.A.P

SOAP codifie simplement une pratique existante

Utilisation conjointe de XML et HTTP

SOAP est un protocole **minimal** pour appeler des méthodes sur des serveurs, services, composants, objets

Ne pas imposer une API ou un runtime

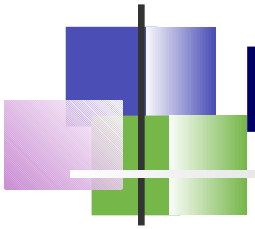
Ne pas imposer l'utilisation d'un ORB (CORBA, DCOM, …) ou d'un serveur web particulier (Apache, IIS, …)

Ne pas imposer un modèle de programmation

Plusieurs modèles peuvent être utilisés conjointement

Et "ne pas réinventer une nouvelle technologie"

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



Les 3 aspects d'un appel SOAP

SOAP peut être vu comme un autre RPC Objets

Les requêtes contiennent les paramètres IN et INOUT

Les réponses contiennent les paramètres INOUT et OUT

SOAP peut être vu comme un protocole d'échange de "message"

La requête contient un seul message (appel sérialisé d'une méthode sur un objet)

La réponse contient un seul message (retour sérialisé d'un appel de méthode sur un objet)

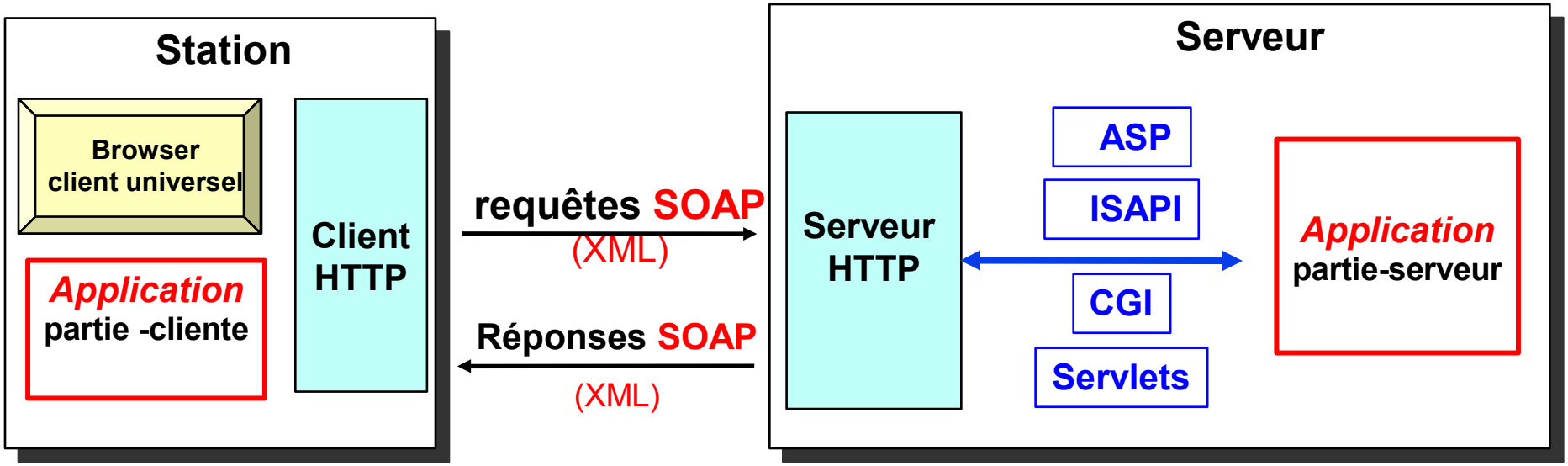
SOAP peut être vu comme un format d'échange de documents

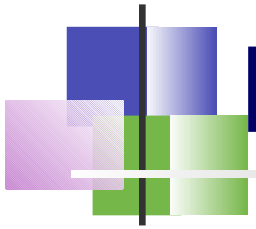
La requête contient un document XML

Le serveur retourne une version transformée

En résumé

SOAP = HTTP + XML





Pourquoi utiliser HTTP ?

HTTP (HyperText Transfer Protocol) est devenu de facto le protocole de communication de l'Internet

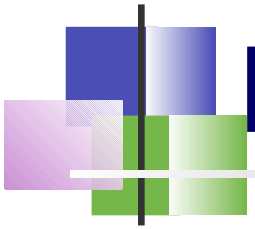
HTTP est disponible sur ***toutes*** les plates-formes – très rapidement

HTTP est un protocole simple, qui ne requière que peu de support pour fonctionner correctement

HTTP est un protocole sans connexion

Peu de paquets sont nécessaires pour échanger des informations

HTTP est le seul protocole utilisable à travers des pare-feu



Fonctionnement d'HTTP

HTTP utilise un protocole requête/réponse basé sur du texte

La première ligne de la requête contient 3 éléments

Verbe : POST/GET/HEAD

URI : /default.htm

Protocole : HTTP/1.0 - HTTP/1.1

La première ligne de la réponse contient 2 éléments

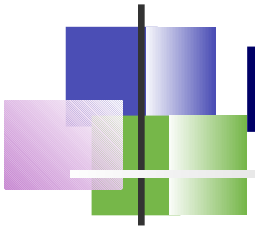
État : 200, 402

Phrase : OK, Unauthorized

Les lignes suivantes contiennent un nombre arbitraire d'entête

Le "contenu" suit une ligne d'entête vide

Utilisé essentiellement pour les réponses et pour les requêtes POST



Fonctionnement d'HTTP

HTTP Request

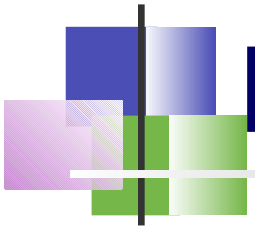
```
GET /bar/foo.txt HTTP/1.1
```

ou

```
POST /bar/foo.cgi HTTP/1.1
Content-Type: text/plain
Content-Length: 14

Goodbye, World
```

HTTP Response

```
200 OK
Content-Type: text/plain
Content-Length: 12

Hello, World
```



Pourquoi utiliser XML ?

- Utilise du texte (peut être lu et écrit directement)
- Construire correctement du texte XML est simple
  - Pas d'éléments qui se recouvrent (uniquement des imbrications)
  - Les attributs sont clairement identifiés (dir="in")
  - Les caractères "<", ">", "&" doivent être précédés d'un caractère d'échappement (ou il faut utiliser CDATA)
- XML est aujourd'hui adopté par tous les acteurs de l'Internet : plates-formes, éditeurs, …
- XML permet une extensibilité aisée par l'utilisation d'espaces de nommage (namespaces et URIs)
- XML permet d'ajouter du typage et de la structure à des informations
  - L'information peut être sauvegardée n'importe où sur le Net
  - Les données fournies par de multiples sources peuvent être agrégées en une seule unité
  - Chaque partie à sa propre structure XML
  - Chaque partie peut définir des types spécifiques
- W3C n'impose pas un API mais en recommande un (DOM)
  - D'autres sont utilisés : SAX, strcat

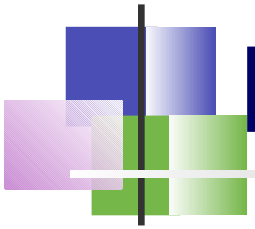
Exemple de requête utilisant HTTP

Demande de cotation à un serveur

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DIS</symbol>
       </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemple de réponse utilisant HTTP

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

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetLastTradePriceResponse
                 xmlns:m="Some-URI">
           <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



Eléments de SOAP

L'enveloppe (enveloppe)

Définit la structure du message

Les règles d'encodage (encoding rules)

Définit le mécanisme de sérialisation permettant de construire le message pour chacun des types de données pouvant être échangés

Fonctionnement en modèle client / serveur (RPC representation)

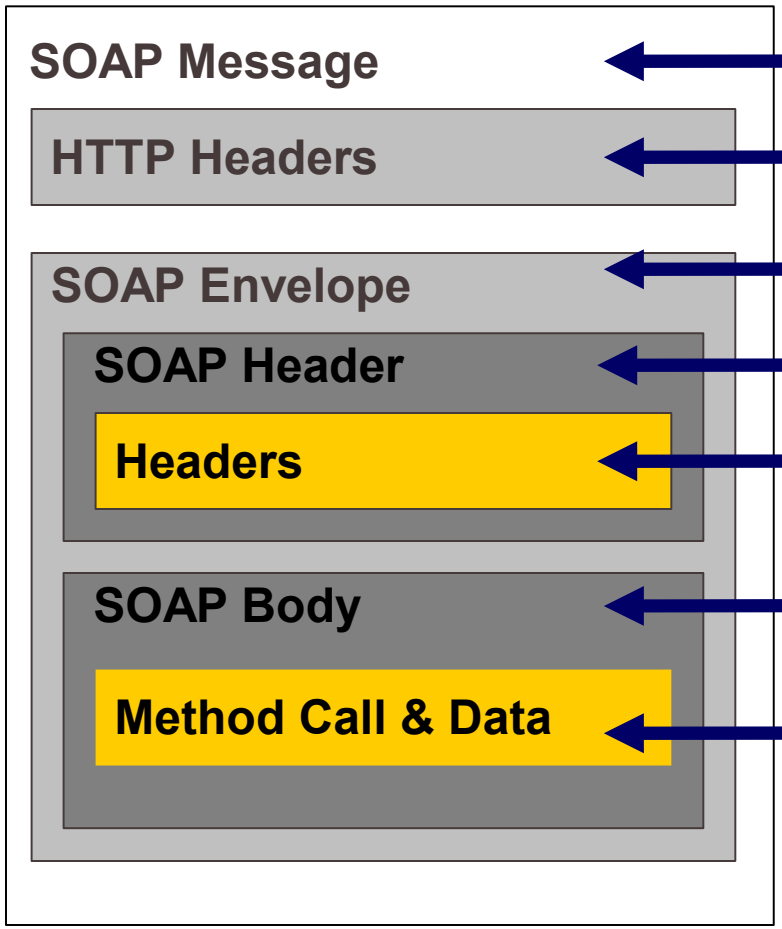
Définit comment sont représentés les appels de procédure et les réponses

Proposer une mise en œuvre sur HTTP (HTTP Extension Framework)

RFC 2774

Définir l'échange de message SOAP sur HTTP

SOAP Message Structure



Le message SOAP Complet

Entête standard HTTP
et entête SOAP HTTP

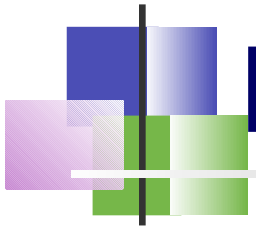
Enveloppe

Entête

Entête individuelle

Corps qui contient les
appels de méthodes SOAP

Appel de méthode et description
en XML de données



Modèle de message

SOAP permet une communication par message

d'un expéditeur vers un récepteur

Structure d'un message

Enveloppe / Envelope

Élément racine

Namespace :
`SOAP-ENVhttp://schemas.xmlsoap.org/soap/envelope/`

Entête / Header

Élément optionnel

Contient des entrées non applicatives

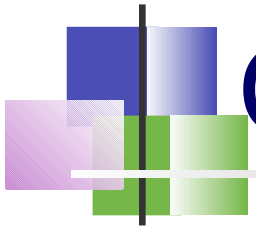
Transactions, sessions, …

Corps / Body

Contient les entrées du message

Nom d'une procédure, valeurs des paramètres, valeur de retour

Peut contenir les éléments « fault » (erreurs)



Corps d'un Message

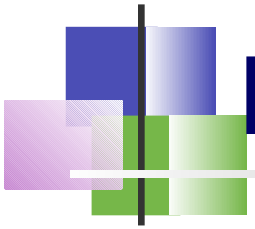
Contient des entrées applicatives

Encodage des entrées

Namespace pour l'encodage

`SOAP-ENC http://schemas.xmlsoap.org/soap/encoding/`

xsd : XML Schema



Principes des règles d'encodage

- Les règles d'encodage définissent un système de type
- Les types SOAP peuvent être décrit en utilisant XSD
- SOAP utilise les conventions XSD pour associer les instances aux types
- Les tableaux et les références sont typés de manière spécifique en utilisant XSD

Règles d'encodage

Types primitifs

```
<element name="price"
type="float"/>
<element name="greeting"
 type="xsd:string"/>
```

```
<price>15.57</price>
<greeting id="id1">Hello</greeting>
```

Structures

```
<element name="Book"><complexType>
 <element name="author"
type="xsd:string"/>
 <element name="title"
type="xsd:string"/>
</complexType></element>
```

```
<e:Book>
  <author>J.R.R Tolkien</author>
  <title>A hobbit story</title>
</e:Book>
```

Enumération

```
<element name="color">
  <simpleType base="xsd:string">
    <enumeration value="Green"/>
    <enumeration value="Blue"/>
  </simpleType>
    </element>
```

```
<color>Blue</color>
```

Règles d'encodage

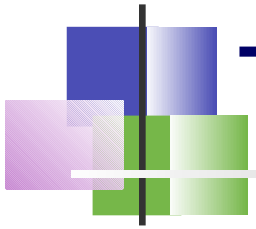
Références

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

<e:Book>
   <title>My Life and Work</title>
   <firstauthor href="#Person-1"/>
   <secondauthor href="#Person-2"/>
</e:Book>
<e:Person id="Person-1">
   <name>Henry Ford</name>
   <address xsi:type="m:Electronic-address">
       <email>mailto:henryford@hotmail.com</email>
       <web>http://www.henryford.com</web>
   </address>
</e:Person>
<e:Person id="Person-2">
   <name>Samuel Crowther</name>
   <address xsi:type="n:Street-address">
       <street>Martin Luther King Rd</street>
       <city>Raleigh</city>
       <state>North Carolina</state>
   </address>
</e:Person>
```

Un exemple d'échange

```
POST /path/foo.pl HTTP/1.1
Content-Type: text/xml
SOAPAction: interfaceURI#Add
Content-Length: nnnn

<soap:Envelope xmlns:soap='uri for soap'>
  <soap:Body>
    <Add xmlns='interfaceURI'>
      <arg1>24</arg1>
      <arg2>53.2</arg2>
    </Add>
  </soap:Body>
</soap:Envelope>
```

```
200 OK
Content-Type: text/xml
Content-Length: nnnn

<soap:Envelope
    xmlns:soap='uri for soap'>
  <soap:Body>
    <AddResponse xmlns='interfaceURI' >
      <sum>77.2</sum>
    </AddResponse>
  </soap:Body>
</soap:Envelope>
```



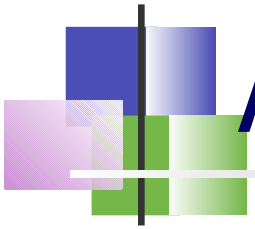
Types de message SOAP

SOAP définit trois types de message

Appel (Call) - obligatoire

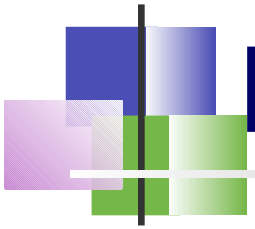
Réponse (Response) - optionnel

Erreur (Fault) - optionnel



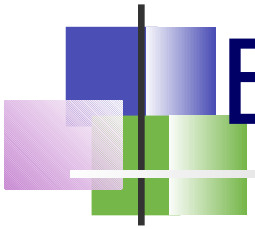
Appel simple

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml
Content-Length: nnnn
SOAPMethodName: Some-Namespace-URI#GetLastTradePrice

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1”>
    <SOAP:Body>
        <m:GetLastTradePrice
          xmlns:m="Some-Namespace-URI”>
            <symbol>DIS</symbol>
        </m:GetLastTradePrice>
    </SOAP:Body>
</SOAP:Envelope>
```



Réponse

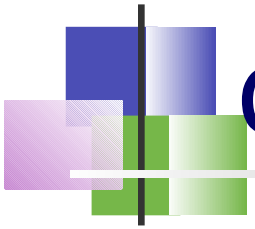
```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1”>
    <SOAP:Body>
        <m:GetLastTradePriceResponse
          xmlns:m="Some-Namespace-URI”>
            <return>34.5</return>
        </m:GetLastTradePriceResponse>
    </SOAP:Body>
</SOAP:Envelope>
```



Erreur

```
<SOAP:Envelope
  xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1>
  <SOAP:Body>
    <SOAP:Fault>
      <faultcode>200</faultcode>
      <faultstring>
        SOAP Must Understand Error
      </faultstring>
      <runcode>1</runcode>
    </SOAP:Fault>
   <SOAP:Body>
</SOAP:Envelope>
```

Autres éléments de SOAP sur HTTP



Le type MIME d'une requête SOAP est text/xml

Toutes les requêtes SOAP doivent pouvoir être reconnues comme telles par un serveur HTTP

Utilisation d'un entête HTTP spécifique

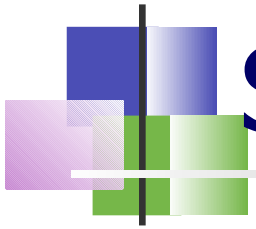
SOAPAction: interfaceURI#methodname

Les erreurs HTTP utilisent l'infrastructure HTTP

Les erreurs SOAP/app utilisent les éléments SOAP PDU

Modèle standard pour toutes les erreurs

Extensible pour prendre en compte les exceptions



Sécurité

Basé sur la sécurité dans http

HTTPS

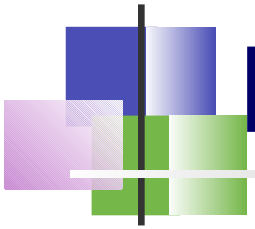
Certificats X.509

Les Firewalls peuvent filtrer les messages facilement

Pas de transfert de code applicatif

Uniquement des données

Les paramètres sont typés lors du transport



Portée de SOAP

SOAP est simple et extensible

Il permet de réaliser des appels de méthode sur le Web

Indépendant des OS, des modèles objets, des langages

Transport des messages par HTTP + XML on the wire

Fonctionne avec l'infrastructure Internet existante

Permet l'interopérabilité entre OS, langages et modèles objets

Ce n'est pas un système réparti à objets

Il ne couvre donc pas les fonctions suivantes :

Pas de ramassage des miettes

Pas de contrôle de types, pas de gestion de version

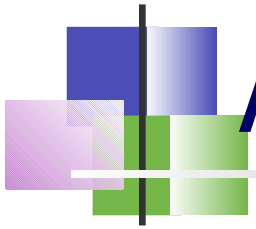
Pas de dialogue entre deux serveurs HTTP

Pas de passage d'objets par référence

Nécessite ramassage des miettes en réparti et HTTP bi-directionnel

Pas d'activation

Nécessite passage d'objets par référence



Autres Extensions

Transport

SOAP sur SMTP/FTP/POP3/IMAP4/RMI-IIOP

Voir implémentation IBM/Apache

SOAP sur MOM (JMS)

Encodage

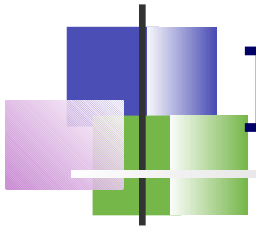
XMI (UML)

Voir implémentation IBM/Apache

Litteral XML

DOM org.w3c.dom.Element sérialisé

Voir implémentation IBM/Apache



Implémentation de SOAP

On peut installer SOAP dans un ORB

Nouveau, Orbix 2000, Voyager, COM

On peut installer SOAP dans un serveur Web

Apache, ASP/ISAPI, JSP/Servlets/WebSphere

Traitement d'un message SOAP (1/2)

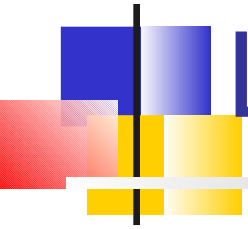
- L'ensemble des informations nécessaire à la gestion d'un message SOAP sont contenues dans l'entête
  - Le corps SOAP : informations destinées à l'application
  - L'entête SOAP : informations pour les intermédiaires (infrastructures réseau : proxy, pare-feux, cache) qui effectuent des traitements sur les entêtes SOAP
    - Exemple d'information contenues dans l'entête : certificat, identifiant des transactions, de sessions, informations pour le cache (ai-je le droit de cacher cette informations?)
    - A qui est destiné un élément (rôle : ultimateReceiver, next, none)
    - L'attribut mustUnderstand prenant les valeurs false ou true,
    - L'attribut relay (1 ou 0) définissant si une entête non comprise est relayée

Limitations de SOAP (1/2)

- La relation très privilégiée avec HTTP, RPC, XML, implique que SOAP 1.1 n'est pas un véhicule universel
  - HTTP : les spécifications SOAP (1.x) ne contiennent pas de binding pour des protocoles tels que SMTP (une note du W3C est promulgée avec SOAP 1.2)
- Interaction client-serveur rigide et synchrone (SOAP 1.1)
  - Que faire par exemple si un email est envoyé en réponse : SOAP ne gère pas la notification d'événements -> aucune réaction (le mail est mis dans le spool...)

Limitations de SOAP (2/2)

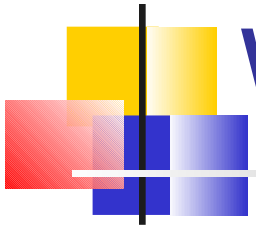
- XML :
  - problématique pour échanger des données (ex. images) ne pouvant pas être sérialisées en XML
    - -> SOAP 1.2 : note concernant les attachements (format MIME)
  - Problématique pour comprimer, chiffrer ...
    - SOAP 1.2 : découplage par rapport à XML
    - Une description abstraite du contenu d'un message SOAP, appelée infoset, est fournie
  - La spécification SOAP s'est éloignée de plus en plus de RPC



Les services Web

WSDL : Web Services Description Language

Merci à Didier Donsez



WSDL

Spécification (09/2000)

Ariba, IBM, Microsoft

TR W3C v1.1 (25/03/2001)

Objectif

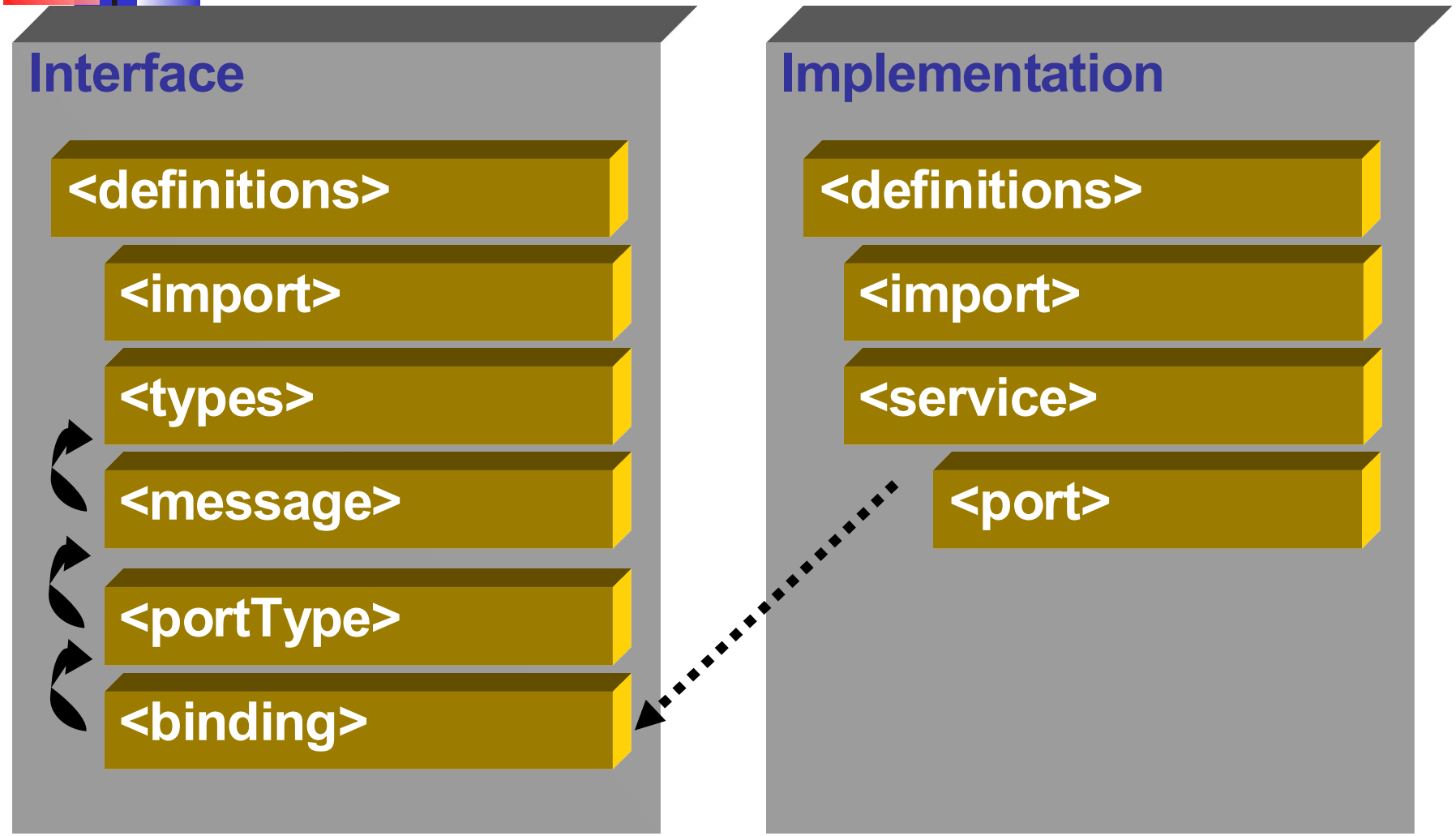
Décrire les services comme un ensemble d'opérations et de messages abstraits relié (*bind*) à des protocoles et des serveurs réseaux

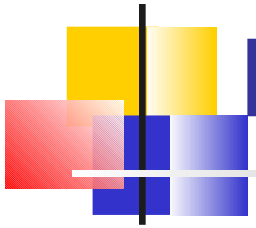
Grammaire XML (schema XML)

Modulaire (*import* d'autres documents WSDL et XSD)

Séparation entre la partie abstraite et concrète

WSDL


Interface
<definitions>
<import>
<types>
<message>
<portType>
<binding>
Implementation
<definitions>
<import>
<service>
<port>



Éléments d'une définition WSDL

<types>

Contient les définitions de types utilisant un système de typage (comme XSD).

<message>

Décrit les noms et types d'un ensemble de champs à transmettre

Paramètres d'une invocation, valeur du retour, …

<porttype>

Décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs message de sortie ou de fautes

<binding>

Spécifie une liaison d'un <porttype> à un protocole concret (SOAP1.1, HTTP1.1, MIME, …). Un <porttype> peut avoir plusieurs liaisons !

<port>

Spécifie un point d'entrée (endpoint) comme la combinaison d'un <binding> et d'une adresse réseau.

<service>

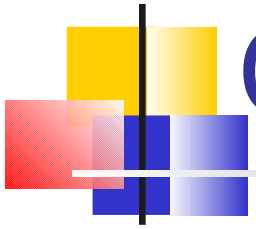
Une collection de points d'entrée (endpoint) relatifs.

Élément <types>

Contient les définition de types utilisant un système de typage (comme XSD).

Exemple

```
<!-- type defs -->
<types>
  <xsd:schema targetNamespace="urn:xml-soap-address-demo"
        xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <xsd:complexType name="phone">
      <xsd:element name="areaCode" type="xsd:int"/>
      <xsd:element name="exchange" type="xsd:string"/>
      <xsd:element name="number" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="address">
      <xsd:element name="streetNum" type="xsd:int"/>
      <xsd:element name="streetName" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zip" type="xsd:int"/>
      <xsd:element name="phoneNumber" type="typens:phone"/>
    </xsd:complexType>
  </xsd:schema>
</types>
```



Outils

Générateur WSDL à partir de déploiement SOAP ou EJB, …

Générateur de proxy SOAP à partir de WSDL

Toolkits (Wsdl2Java / Java2Wsdl, …)

Propriétaires (non normalisés)

UDDI - Universal Description Discovery and Integration

Françoise Sailhan
Francoise.sailhan@cnam.fr

UDDI – Universal Description Discovery and Integration - 2000

- Un annuaire publique mettant à disposition les information techniques et commerciales
- Utilisé pour publier et découvrir ces informations
- Supporte des interactions durant la conception et l'utilisation du service (runtime)
- Classifie/indexe ces informations en suivant une taxonomie standard
- Correspond à une des infrastructures du paysage des services Web

Mise en oeuvre

- Spécification gouvernée par le consortium UDDI
- Un ou plusieurs nœuds implémentent cette spécification
  - Cas 1 : un seul noeud
  - Cas 2 : plusieurs noeuds
    - Un système de réplication est implémenté entre les nœuds
    - On parle alors de cloud UDDI
- Ce(s) noeud(s) sont privé(s) (isolés : pare-feux, réseau privé), publiques (extranet : ouvert), affiliés (accès controlé, UDDI v.3)

Pourquoi UDDI?

- UDDI apporte une réponse concrête aux questions suivantes :
  - Comment découvrir quels sont les services disponibles ?
  - Quels sont les services offerts étant donné un domaine d'application particulier ?
  - Quels sont les services Web mis offerts par une compagnie donnée ?
  - Quels sont les caractéristiques (contact?) d'une compagnie mettant à disposition ce service
  - Quels sont les détails relatifs à l'implémentation d'un service Web

Etape 1 : Modélisation sous UDDI

- Pour répondre à ces questions efficacement, il faut modéliser les données stockées
- Deux entités :
  - L'entité business
    - Nom de la compagnie, sa description (plusieurs langues),
    - Point(s) de contact (email, téléphone, adresse postale)
    - Catégorie et identification de la companie
  - Le ou les services Web offerts par l'entité business
    - Description du service offerts
    - Catégorie

Etape 2 : Enregistrement sous UDDI

Conceptuellement, les informations commerciales sont classifiées en trois composants

Pages blanches	Pages jaunes	Pages vertes
Nom de l'entité business	Index services & produits	Règles
Contact(s)	Codes d'industrie	Description des services
Description (string multi-langage)	Index géographique	Invocation de l'application
Identifiant de l'entité business		Binding des données

Indépendant de la plateforme

Partie du document XML décrivant une entité business

Clé unique

```
<businessEntity businessKey= "A867FG00-52NM-EFT1-134-098765432124">
  <name> Horloge parlante  </name>
  <description xml:lang="fr">
    L'horloge parlante fournit l'heure
  </description>
  <contacts>
    <contact useType="France general">
    <personName>Responsable .</personName>
    <phone>01 42 38 CALL </phone>
    <email useType="">responsableBusiness@horlogel.com</email>
    <address>….</address>
    </contact>
  </contacts>
<businessServices> …   </businessServices>
  <identifierBag> …    </identifierBag>
  <categoryBag> …
  <keyedReference tModelKey="UUID:DB77450D-9FA"
keyName="Electronic check-in" keyValue="84121801"/>
  </categoryBag></businessEntity>
```

Attribut définissant la langue, ici l'anglais

Possiblement plusieurs contacts (par ex. contact business & contact technique)

Description des services

Identifiant & catégorie

Pages blanches

Nom de l'entité business

Contact

Description (liste de string

Identifiant unique pour cha

Description des services

Description des services (**nom, id**)

```
<businessService serviceKey=894B5100-3AAF-11D5-80DC-002035229C64"
        businessKey="D2033110-3AAF-11D5-80DC-002035229C64">
   <name>ElectronicTimingService</name>
   <description xml:lang="fr">Service electronique temps</description>
```

Binding : détail d'encodage

```
<bindingTemplates>
        <bindingTemplate bindingKey="6D665B10-3AAF-"serviceKey="89470B40-
           <description>Date, SOAP info </description>
           <accesssPoint URLType="http">http://horloge.com/timeservice
           </accessPoint>
           <tModelInstanceDetails>
             <tModelInstanceInfo tModelKey="D2033110-3BGF-1KJH-234C-2">
              ...
             </tModelInstanceInfo>
           </tModelInstanceDetails>
        </bindingTemplate>
   </bindingTemplates>
```

Catégorie du service

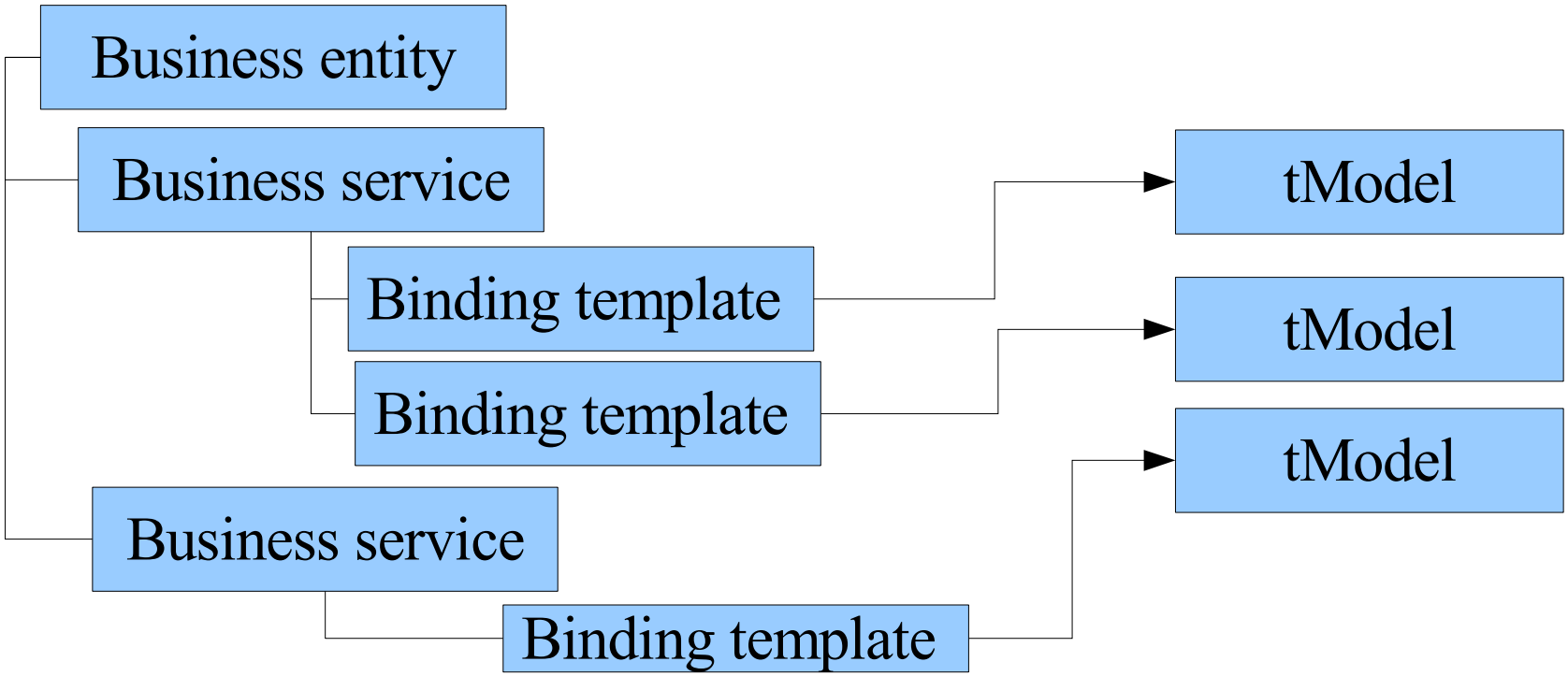
```
<categoryBag>
     ...
   </categoryBag>
</businessService>
```

Relations entre WSDL et UDDI (1/4)

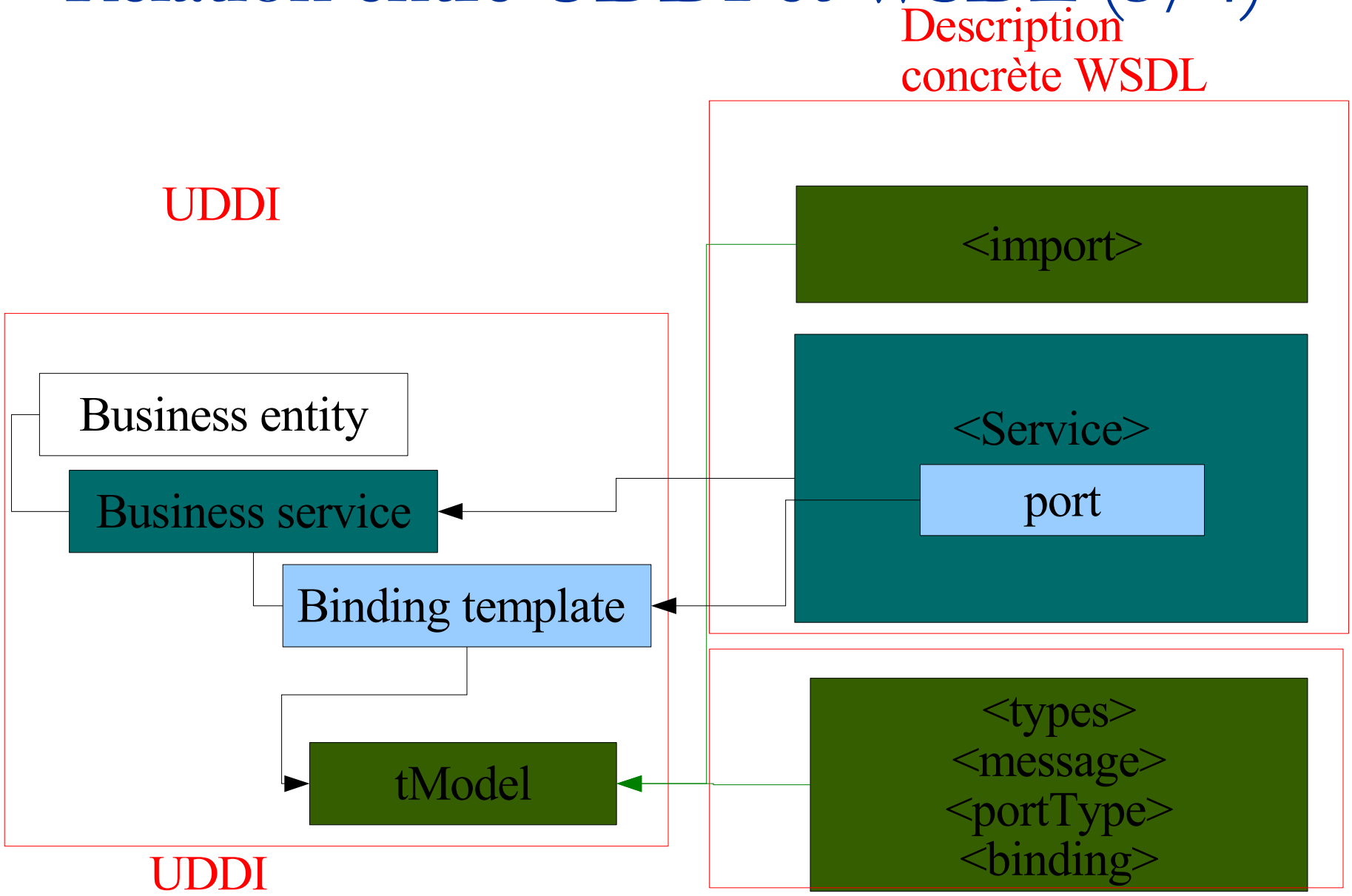
- UDDI ne fournit pas de support pour une spécification de services particulière
- L'élément Tmodel est utilisé pour définir la spécification technique d'un service
  - Tmodel fait référence à une information extérieur
  - Le format utilisé, les informations stockées sont laissés à la discrétion du développeur

Relations entre WSDL et UDDI (2/4)

- Une entité business contient des services dont la description technique d'un service est définit dans se(s) bindingTemplate(s)
- Chaque bindingTemplate référence au moins un tModel



Relation entre UDDI et WSDL (3/4)



Relations entre UDDI et WSDL (4/4)

- Un document WSDL est composé de deux parties
  - une description **concrète** dans laquelle sont décris le service, ses ports
  - une description **abstraite** dans laquelle sont décris les types des données échangés, les messages et le type de port
- La description abstraite est mappée dans un port

Version simplifiée de TModel

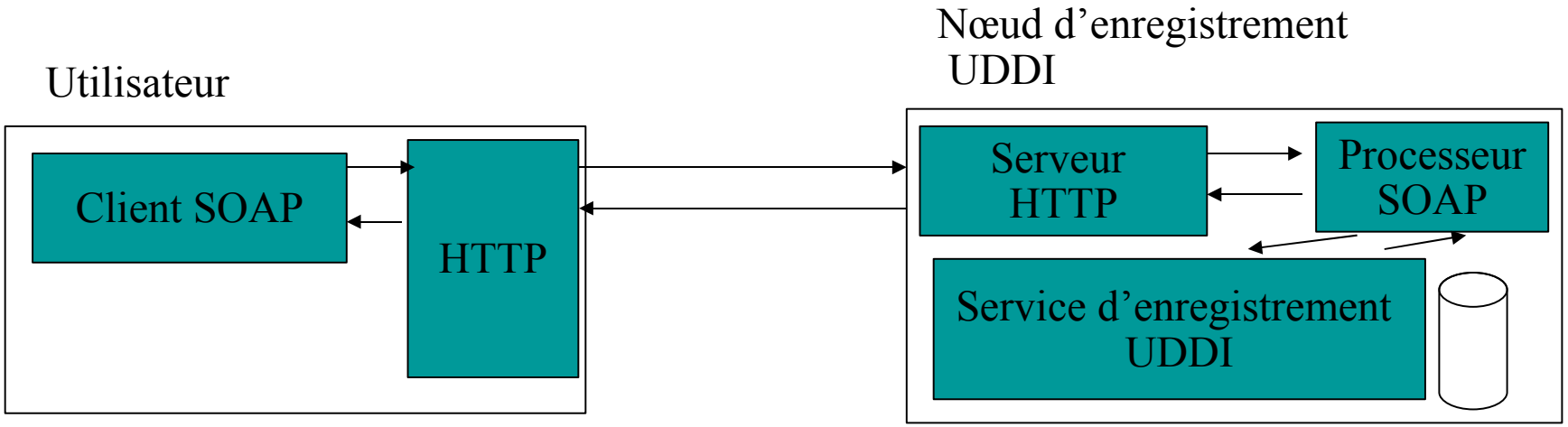
```
<tModel tModelKey="">
  <name>http://www.horloge.com/time-interface</name>
  <description xml:lang="en">Time service interface </description>
    <overviewDoc>
        <description xml:lang="en">WSDL document </description>
         <overviewURL>http://www.hotloge.com/services/time.wsdl
         </overviewURL>
    </overviewDoc>
    <categoryBag> ...
    </categoryBag>
</tModel>
```

Enregistrement sous UDDI

Deux approches

1/ interface utilisateur adaptée : formulaire accessible a partir d'une page Web

3/ Enregistrement automatique : interface SOAP API



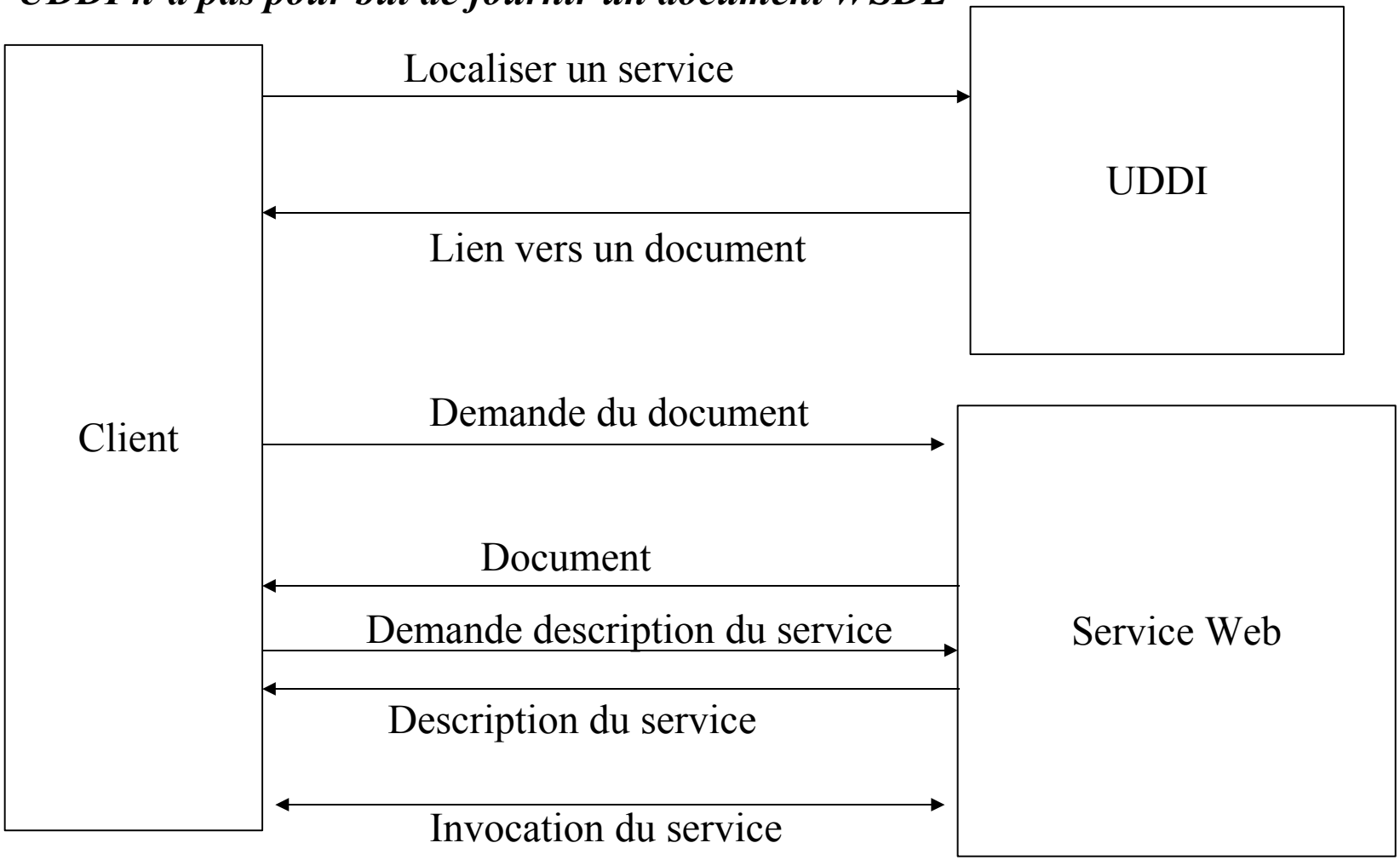
Etape suivante : découvrir les services

Découverte de services

- UDDI fournit une API de recherche des services Web permettant aux utilisateur de découvrir le service Web recherché
- Une requête UDDI peut porter sur
  - Une entité business (argument : nom business, catégorie business ..)
  - Un service (argument : clé du service ou nom du service, find_service())
- Après identification de l'entrée UDDI voulue, un ensemble d'API fournit plus de détail sur les paramètres enregistrées
  - get_businessDetail(), get_serviceDetail(), ...

Mécanisme de découverte

UDDI n'a pas pour but de fournir un document WSDL



UDDI- Bilan

- Une spécification technique permettant de publier et rechercher des services Web
- 3 versions successives d'UDDI
  - UDDI v1 (2000) : pose les fondations
  - UDDI v2 (2003) : aligne UDDI avec les évolutions des services Web
  - UDDI v3 (2005) : support accru à la sécurité (privé, publique, affilié)
- Eléments non gérés : prix, confidentialité, réputation, QOS
  - → Support accru pour une utilisation privée (v3)

UDDI - Limites

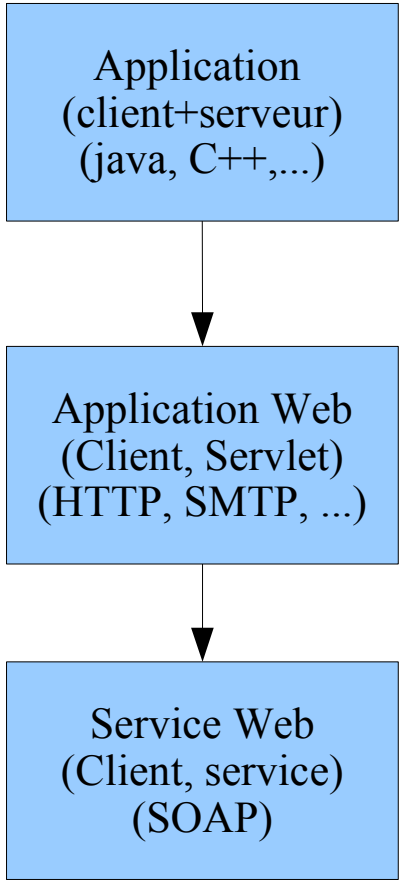
- Pas de découverte dynamique
  - Pas de gestion des documents périmés
  - Descriptions de service sont stockées à l'extérieur d'UDDI
  - Pas de vérification de la disponibilité des services
  - → WS-Discovery : définition d'un protocole multicast pour découverte des services Web
- Interrogation non standard d'un annuaire UDDI & langage d'interogation pauvre
  - → Interrogation améliorée (UDDI v2 et v3)

Outils

Un environnement pour les WS

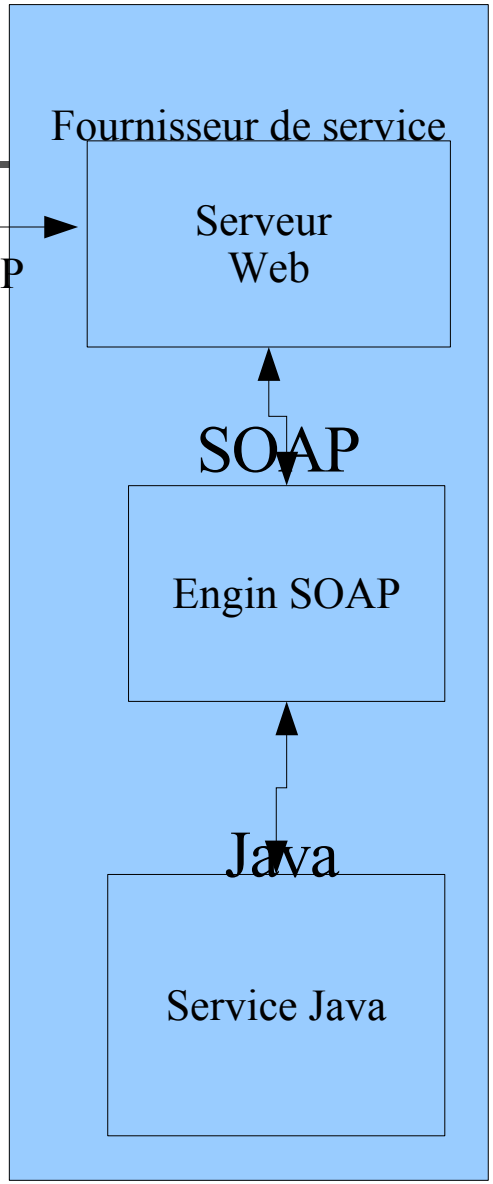
Avant de commencer, sélectionner :

- Un language de programmation (java, c++, ..) pour développer le client et le serveur → **application java**, C++, ...
- Un environnement de développement
- Une IDE (pour faciliter le développement)



Application Web versus Service Web

- Une application Web (WebApp)
  - retourne un document HTML
  - En entrée, prend des données GET ou POST
  - En sortie, fournit un résultat destiné à un humain (navigateur)
- Un service Web
  - Manipule (en entrée et en sortie) des documents XML



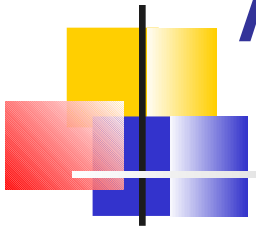
Framework (1/2)

- Une multitude de frameworks disponibles
  - BEA-Oracle JBos, IBM Websphere, .NET Windows
  - **Apache-AXIS2 (successeur de Axis): logiciel libre disponible sous windows, linux, …**
- Chacun de ces frameworks :
  - Fournit les outils nécessaires pour gérer les échanges entre le client et le service Web

Frameworks (2/2)

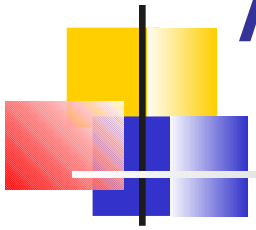
- Les dernière générations de frameworks
  - prennent en compte plus de normes WS-*
  - facilitent la création d'un service web et d'un client:
    - Génération automatique d'une souche client à partir d'un document WSDL
    - Génération d'un document WSDL à partir d'une classe java implémentant le service
    - Génération d'un squellette à partir d'un document WSDL

Axis 2



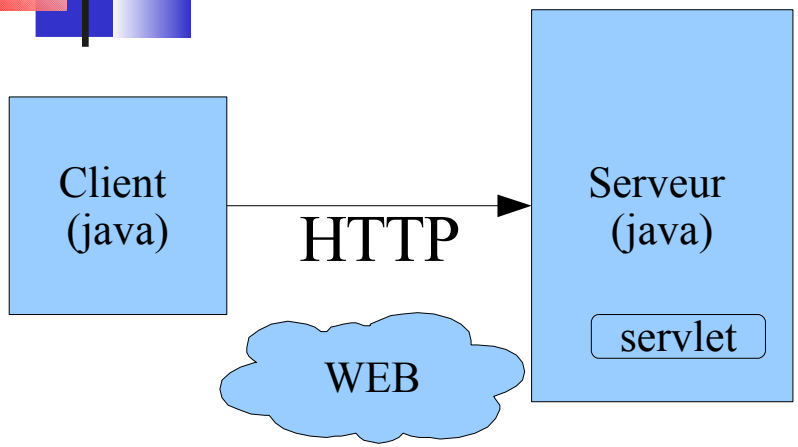
- Un engin SOAP pour développer des services Web
  - en lignes de commande unix/dos (par exemple pour générer un document WSDL à partir d'une implémentation java),
  - en utilisant une interface utilisateur, eclipse par ex.

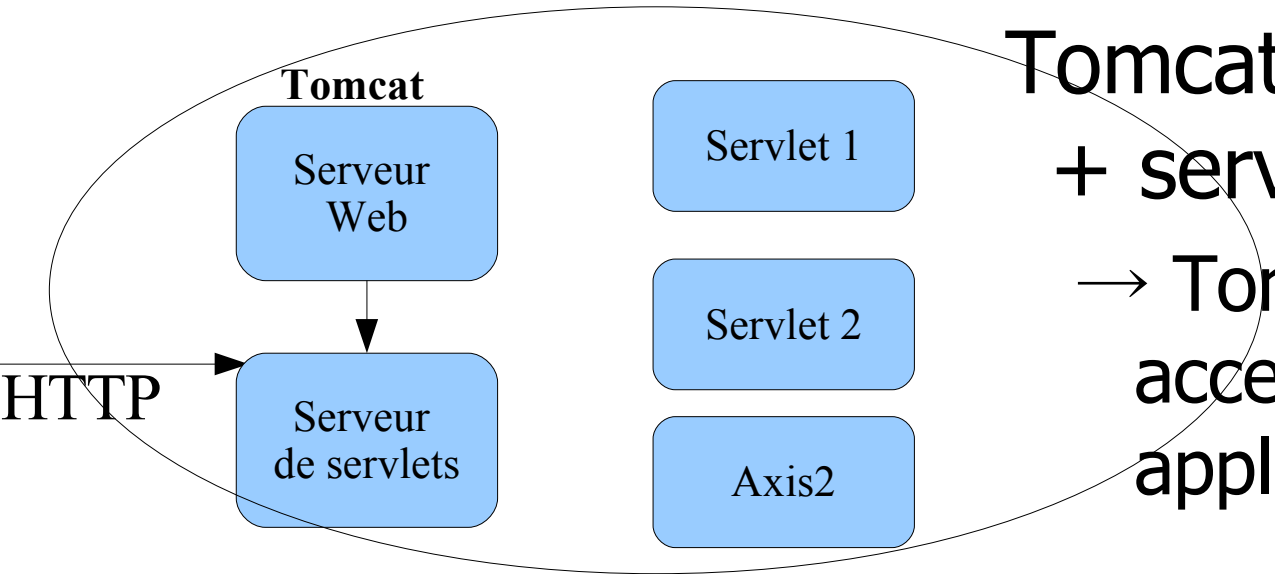
Axis 2 – Etapes



- Etape 1 : Installation du framework
  - Installation sdk (java)
  - Installation d'un serveur d'application Web (Tomcat)
  - Installation d'un engin SOAP (Axis2)
  - Installation d'une IDE (eclipse)
- Etape 2 : Développement du service Web
- Etape 3 : Déploiement du service Web (avec ou sans UDDI)

Etape 1 : Installation



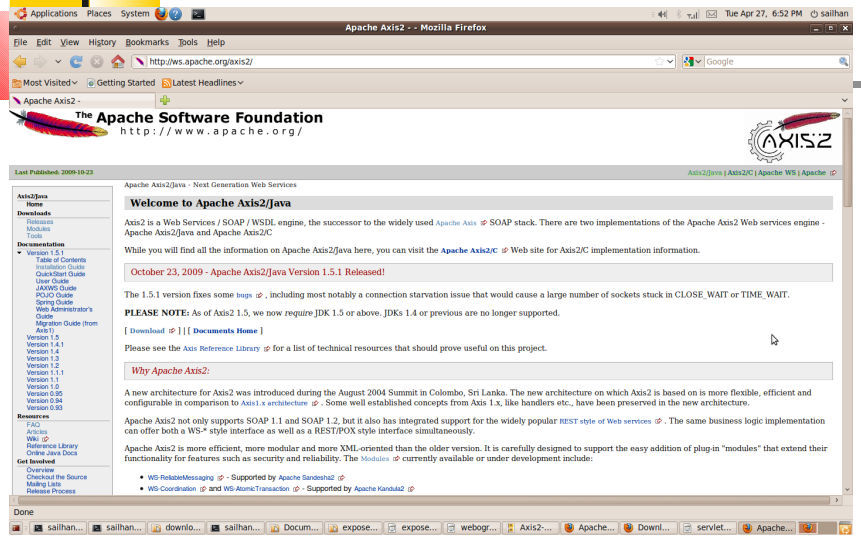


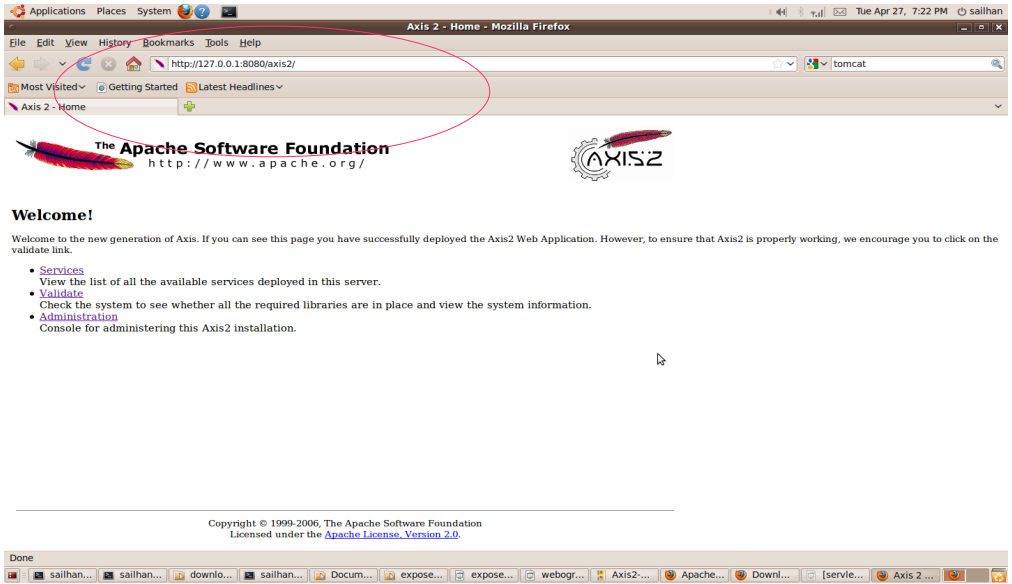
Etape 1. Télécharger un serveur d'application Web, par ex. logiciel libre Tomcat
http://tomcat.apache.org/

Tomcat : serveur Web + serveur de Servlet

→ Tomcat rend accessible des applications Web java

Etape 2 : télécharger Axis

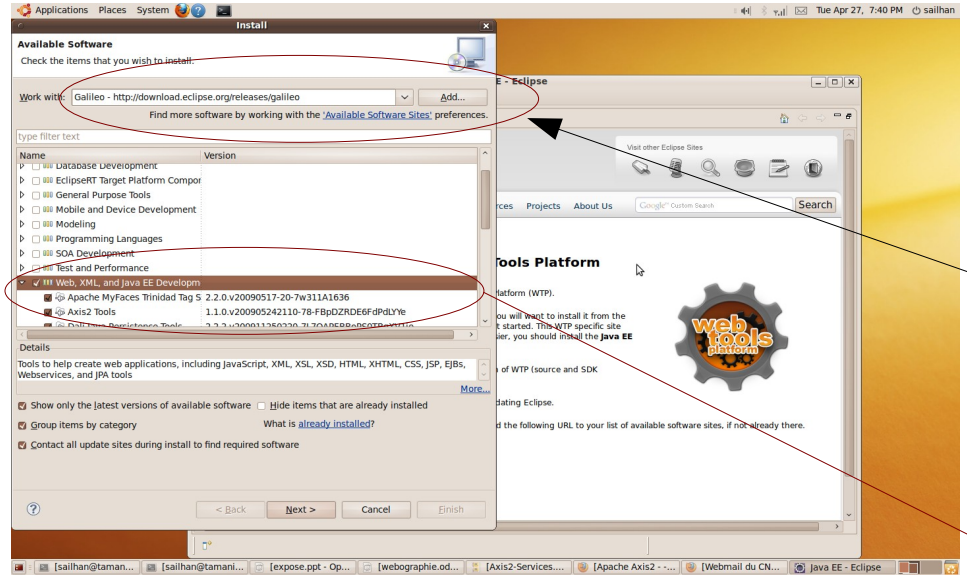




- ~~Télécharger la dernière~~ version binaire d'Axis2 (version pour java et non pas C)
  - http://ws.apache.org/a
  - Installer Axis en tant qu'application web
  - Axis accessible à http://127.0.0.1:8080/axis2

Etape 3 : installation d'eclipse

- Télécharger Eclipse EE
- Mettre à jour eclipse en incluant les plugins nécessaires au développement de services Web
  - Sélectionner dans Help > Update. Les plugins Web sont disponibles à l'URL



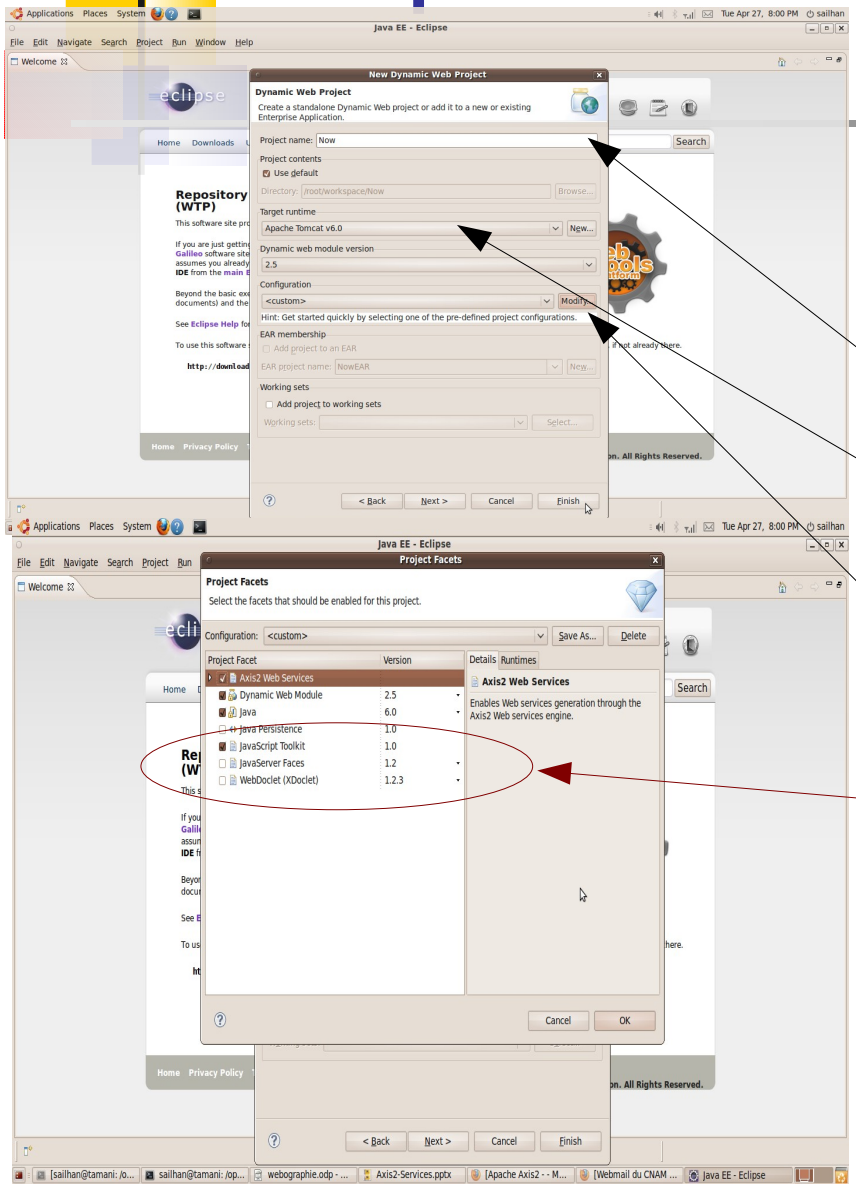
Etape 2 - Création d'un service

- Deux façons de créer un service Web
  - Approche bottom-up (bas vers le haut) : on développe une classe Java implémentant le service, on génère un document WSDL, on génère la souche client, on développe le client
  - Approche top-down (haut vers le bas), on développe un document WSDL à partir duquel on génère souche et le squelette et on implémente le code java

Etape 2 - Implémentation d'un service, approche bottom-up

- Développer une classe java (aucune interface particulière, classe parent, package ou nom de méthode ..)
  - Public class Now {
  - public Date getTime(){...}
  - }
- Exposer de façon automatique la méthode en tant que service Web
  - Now devient un service

Etape 2 – Création d'un projet



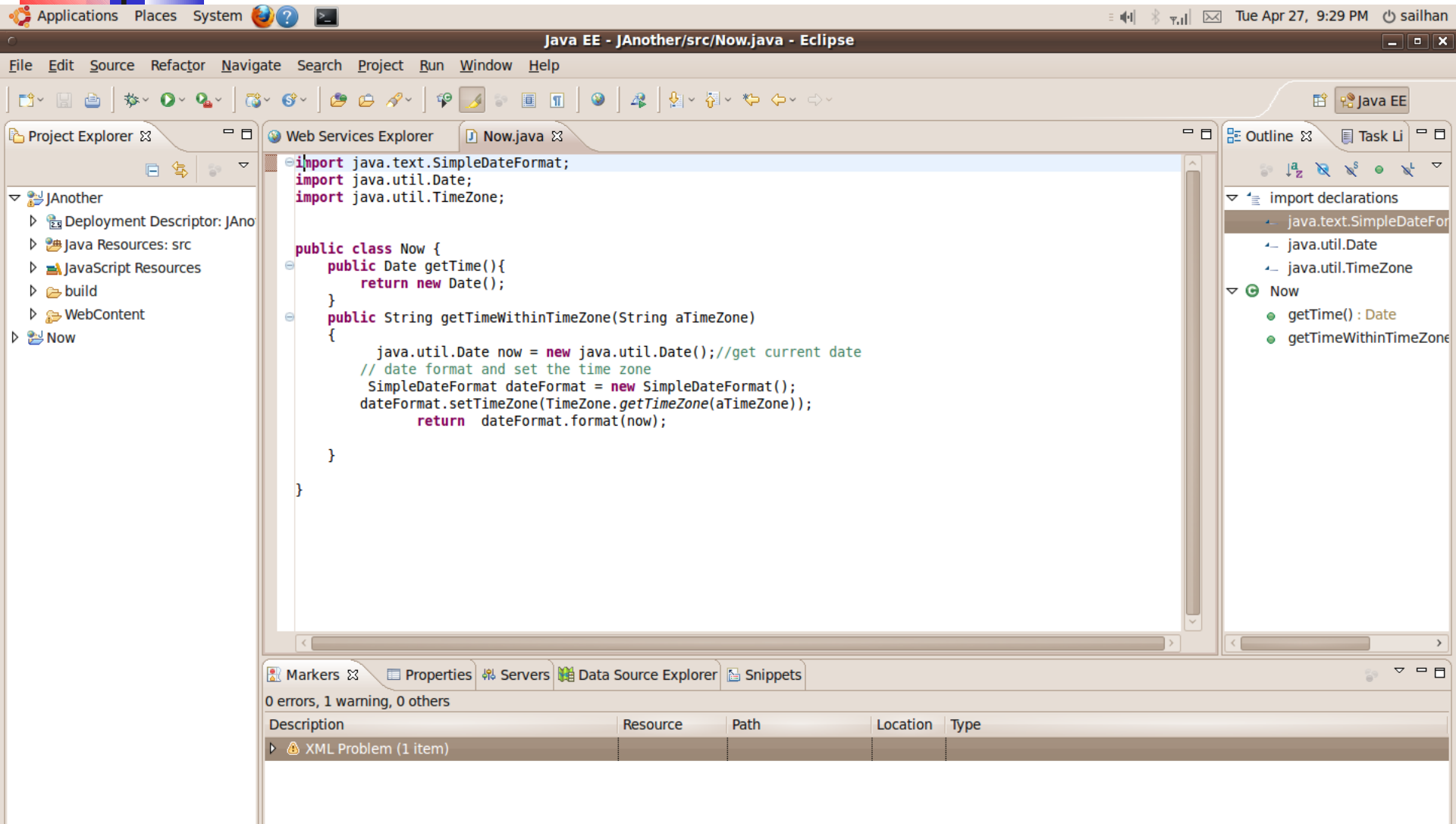
Etape 1 : créer un nouveau projet

File → New → Project → Dynamic Web Project

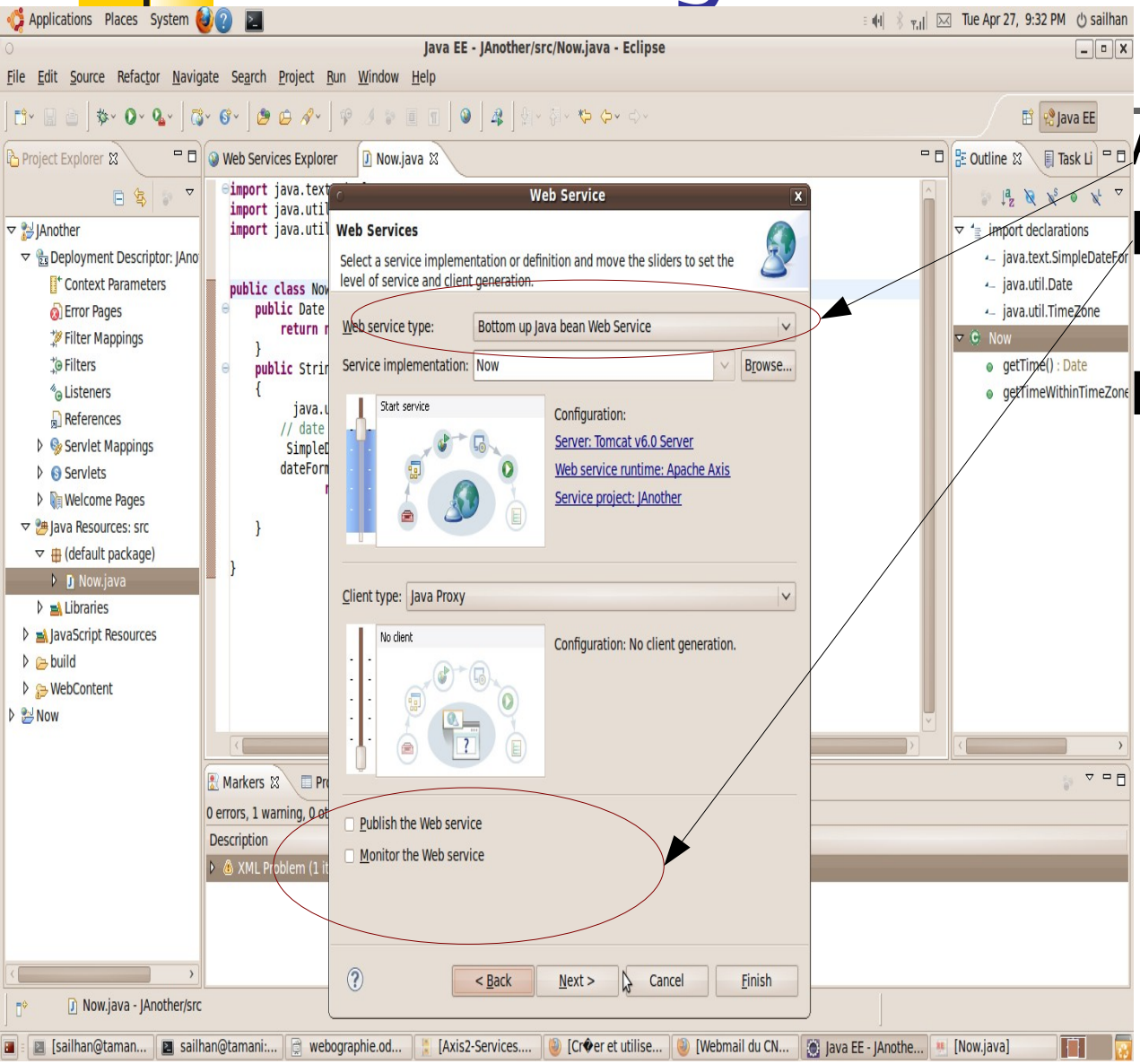
Nom de projet

Selectionner Tomcat comme serveur d'application

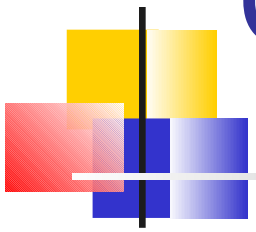
Etape 2 - Implémentation d'une classe



Mise en ligne du service



- Approche Bottom up
- Découverte de service (UDDI...)
- Finish



Compte-rendu

- Au niveau du fournisseur de service Web
  - La classe Now a été implémentée
  - Un fichier WSDL a été généré
  - L'application java a été déployée et est accessible en tant que Service Web (127.0.0.1:8080/...)
- Il reste :
  - À implémenter le client

Développement du client

Idée : en se basant sur le fichier WSDL qui a été généré, une souche client est générée

File → New → Other... → WebService → WebServiceClient

- Localisation du fichier WSDL
- Spécifier un nom de projet (NowClient par ex.)

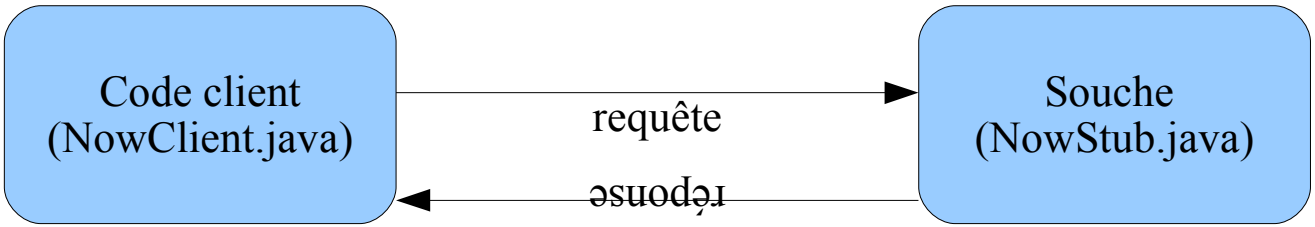
Résultat :

- La souche NowStub.java est générée automatiquement

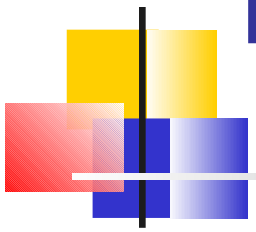
Interaction avec la souche

2 étapes :

- Obtention de la souche
- Envoie d'une requête à la souche
- Réception de la réponse (s'il y a une réponse)



Requête - Réponse



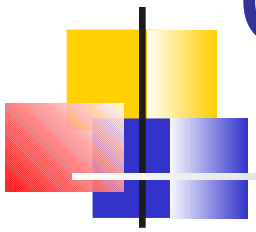
Obtention de la souche

```
String addressOfService = “
  http://localhost/NowService/”+”services/NowService”
XXXStub stub = new XXXStub(addressOfService);
```

Création d'un requête ayant pour paramètre le nom de la zone

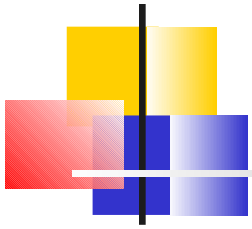
```
XXXStub.GetTime request = new
  XXXStub.getTime();
request.setaTimeZone(“HongKong”);
```

Conclusion



- Services Web : 3 mousquetaires : XML, SOAP, UDDI : description, découverte, accès
- D'Artagnan? : orchestration, conversation, transactions, sécurité, messages, gestion des événements, transactions, profiles, politiques, gestion du service, contexte, fiabilité
- Risque de dégradation des performances : parsing XML, échanges volumineux : XML-binary, compression, mise en cache

Bibliographie



http://www.w3schools.com : Un ensemble de turoriaux et d'exemples

http://www.w3.org/2002/ws/desc/ : page d'accueil du "Web Services Description Working Group"

http://ws.apache.org/axis2 : implémentation open source Axis2 pour construire des services web

www.xmlrpc.com : site officiel d'XML-RPC