



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
- Architecture
- SOAP
- WSDL
- UDDI
- Implémentations
 - Les APIs Java (JAXP, JAX-RPC, JAXM, JAXR, JAXB)
 - Implémentation avec JAX-RPC
 - Apache SOAP, Apache Axis
- Conclusion



Les services Web

Généralités



Un Service Web, c'est quoi ?

- Un service Web est une « **unité logique applicative** » accessible en utilisant les **protocoles standard d'Internet**
- Une «**librairie**» fournissant des données et des services à d'autres applications.
- Un **objet métier** qui peut être déployé et combiné sur **Internet** avec une faible dépendance vis-à-vis des **technologies** et des **protocoles**.
- Combine les meilleurs aspects du développement à base de **composants** et du **Web**.



Un Service Web, c'est quoi ?

- Caractéristiques:

- Réutilisable

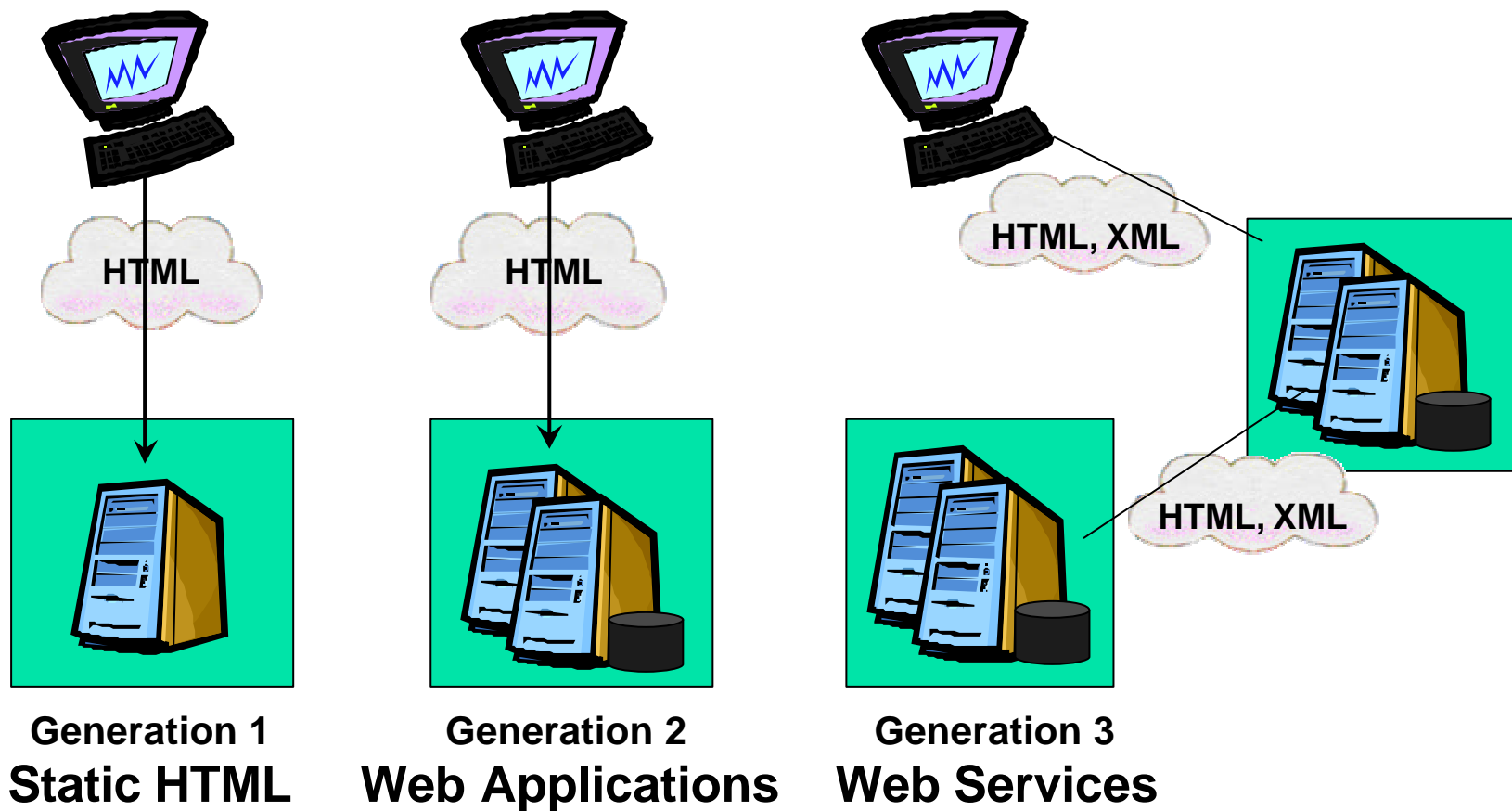
- Indépendamment de

- la plate-forme (UNIX, Windows, ...)

- l'implémentation (VB, C#, Java, ...)

- l'architecture sous-jacente (.NET, J2EE, Axis...)

Evolution du Web



Exemple

The image shows a Microsoft Development Environment (MDE) window titled "HTMLPage1*" in design view. The "Client Objects & Events" pane displays the following HTML code:

```
<a href="/investor/quotes/compare-industry/0-9970-1047-502-7">  
<a href="/investor/brokeragecenter/reports-single-company/0-  
</td>  
<font face="arial,helvetica" size="-1"><b>  
66.27  
</b>&nbsp;</font></td>  
<td><font face="arial,helvetica" size="-1">  
<font color=#cc0000>-2.24</font>  
&nbsp;</font></td>
```

Below the MDE window, a web browser displays the rendered page at the URL: <http://news.cnet.com/investor/quotes/quote-fast/0-9970-1041-0-MSFT.html?tag=qbox>. The page features a navigation bar with links: [Market Update](#), [My Portfolio](#), [Brokerage Center](#), [IPO Center](#), [Splits](#), and [Messages](#). Below this is a section for "Free Real-Time Quotes NEW!".

Symbol	Company	Last	Change	% Change	Volume
* MSFT	MICROSOFT CORP News , Chart , Compare , Broker Reports , Messages , RTSQ	66.27	-2.24	-3.27%	22,983,100

More info: [Detailed Quote](#) | [Profile](#) | [Splits](#) | [Upgrades](#) | [Downgrades](#) | [Buy/Hold/Sell](#) | [Momentum Rating](#) | [SEC Filings](#) **NEW!** | [More...](#) | [Add MSFT to My Portfolio](#)

[Email this quote to a friend](#) | Quotes delayed 15 minutes for Nasdaq, 20 minutes otherwise; NYSE data now reflects extended hours trading.

Le Web 3ème génération

Aujourd'hui

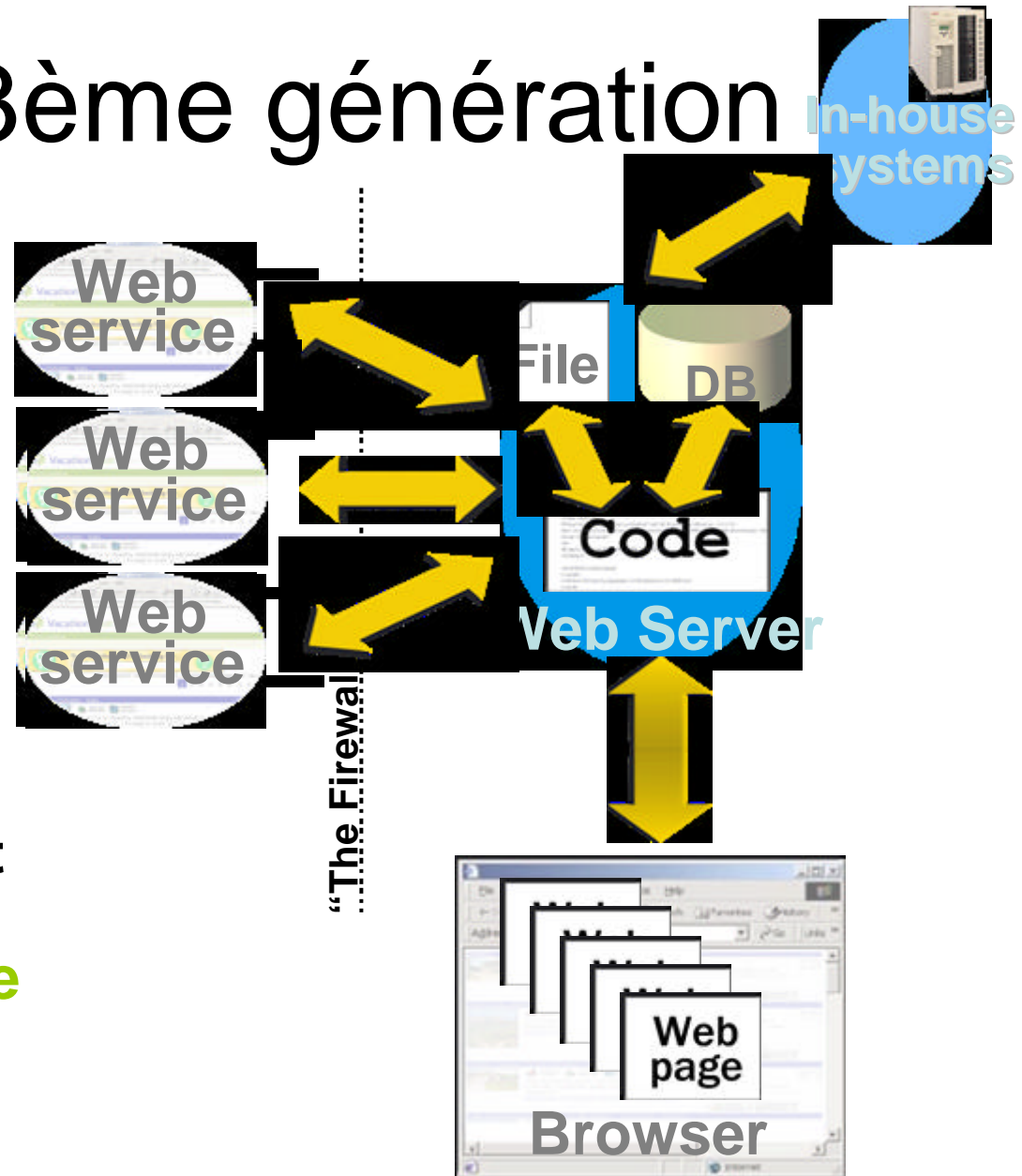
Un site Web fournit des pages HTML

- pas de structure
- impossible à fusionner avec d'autres pages

Demain

Un site Web est un composant fournissant des services en XML

- structure / sémantique
- fusion possible



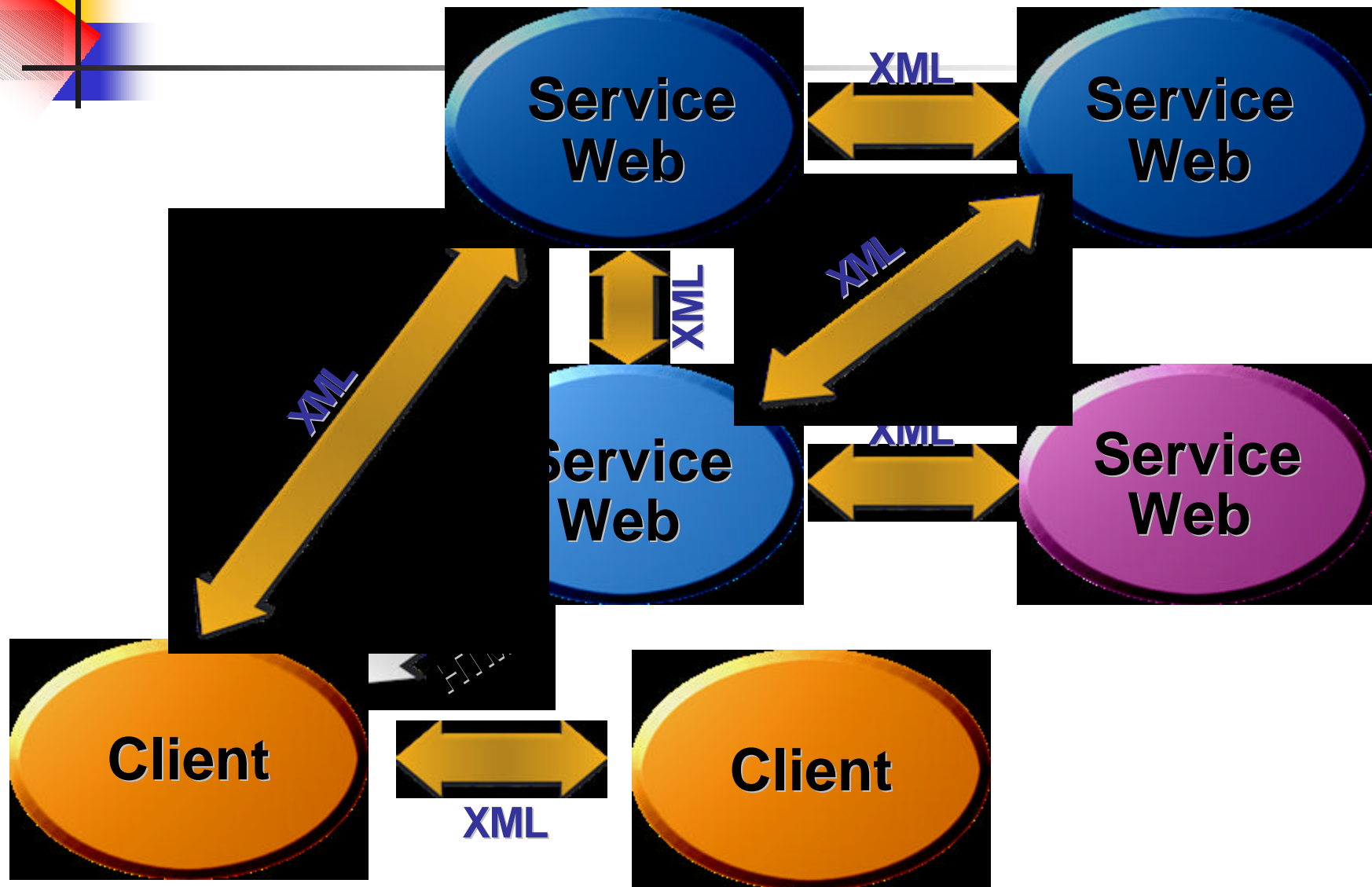
“Dynamic Pages”⁸



Pourquoi faire ?

- Les services Web permettent d'interconnecter :
 - Différentes entreprises
 - Différents matériels
 - Différentes applications
 - Différents clients
 - Pas uniquement des butineurs
- Distribuer et intégrer des logiques métiers
- Vers le Web sémantique
 - Pas uniquement le Web purement interactif
- Les services Web sont faiblement couplés

Modèle client / serveur





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 ?

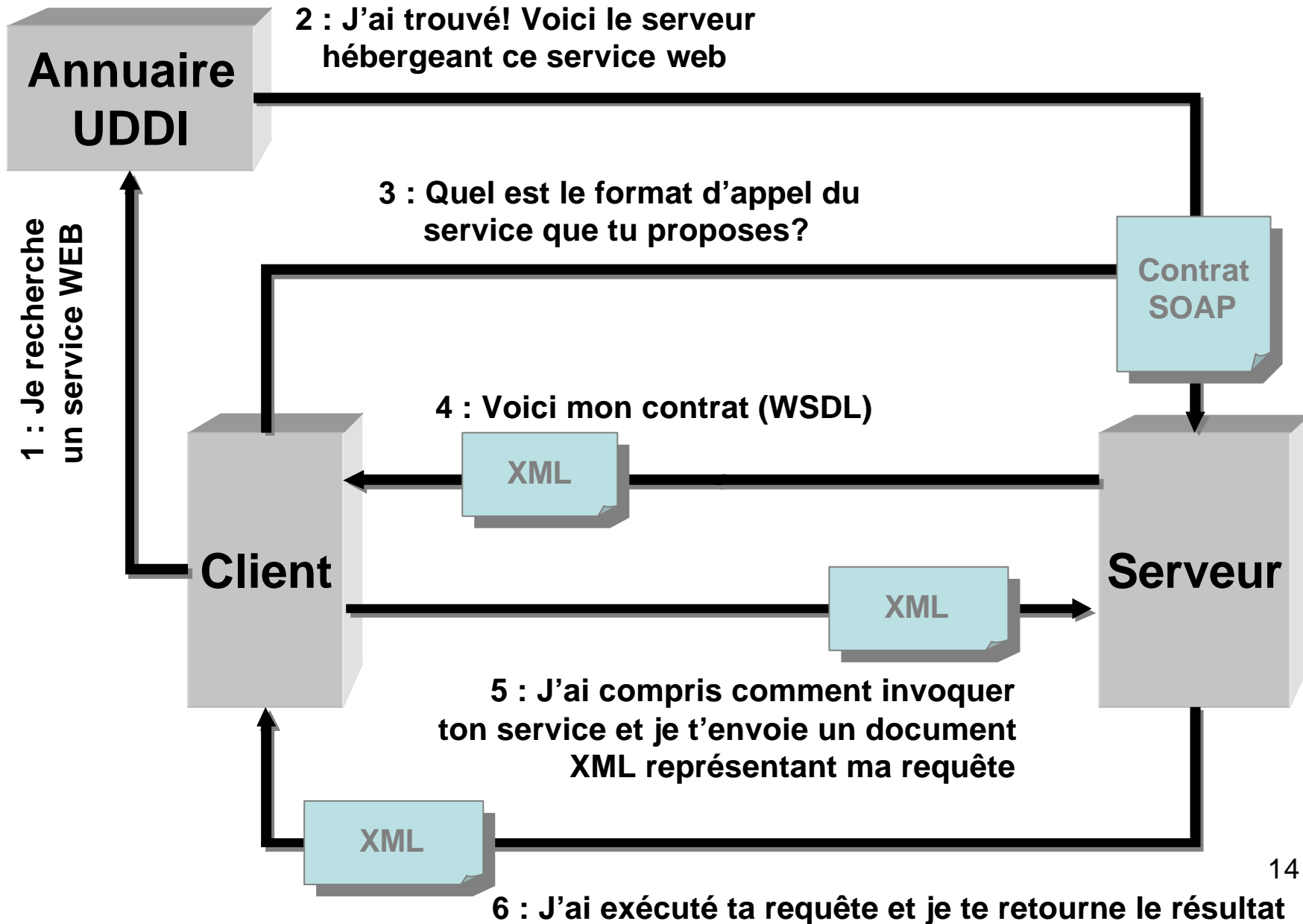
- Une nouvelle technologie des objets distribués ?
 - Invocation distante des services Web : **SOAP** (~IIOP)
 - Description des services Web : **WSDL** (~IDL)
 - Enregistrement et découverte de services Web : **UDDI** (~NameService)
- Basés sur des standards XML
 - Standards du W3C : XML, SOAP, WSDL
 - Standards industriels : UDDI, ebXML
 - Propriétaires : DISCO, WSDD, WSFL, ASMX, ...
- Implémentations actuelles :
 - Microsoft .Net
 - Sun JavaONE : J2EE + Web services (WSDP = JAXP, JAX-RPC, JAXM...)
 - Apache XSOAP / Axis, IBM WSTK
 - Oracle, Bea, Iona, Enhydra ...



Les services Web

Architecture

Cycle de vie d'utilisation

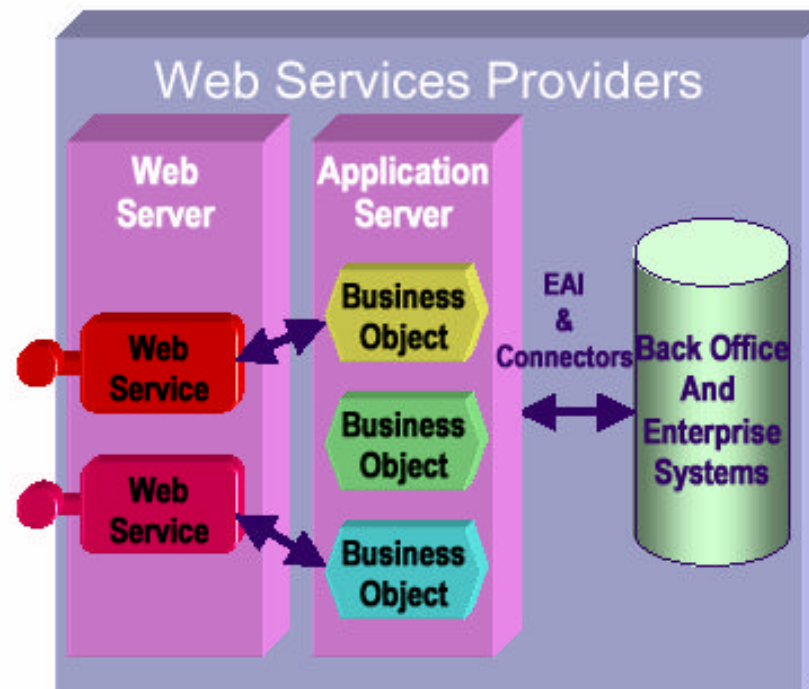




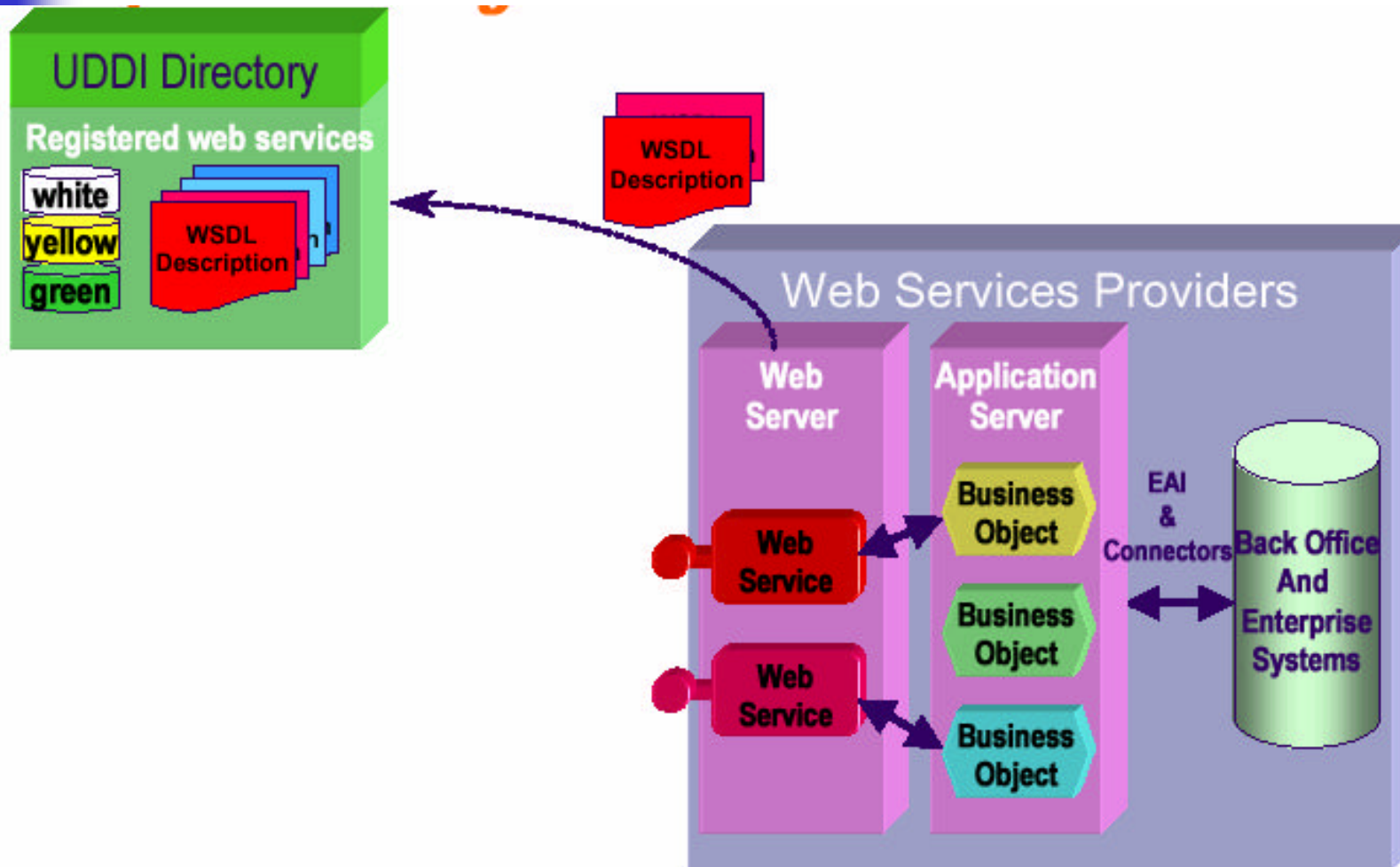
Cycle de vie complet

- 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

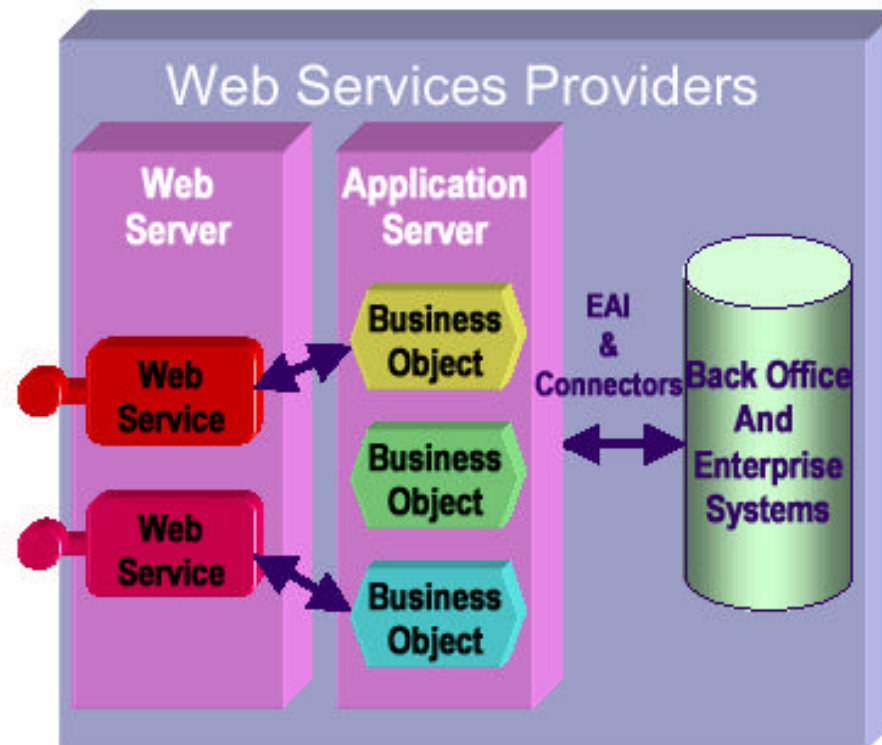
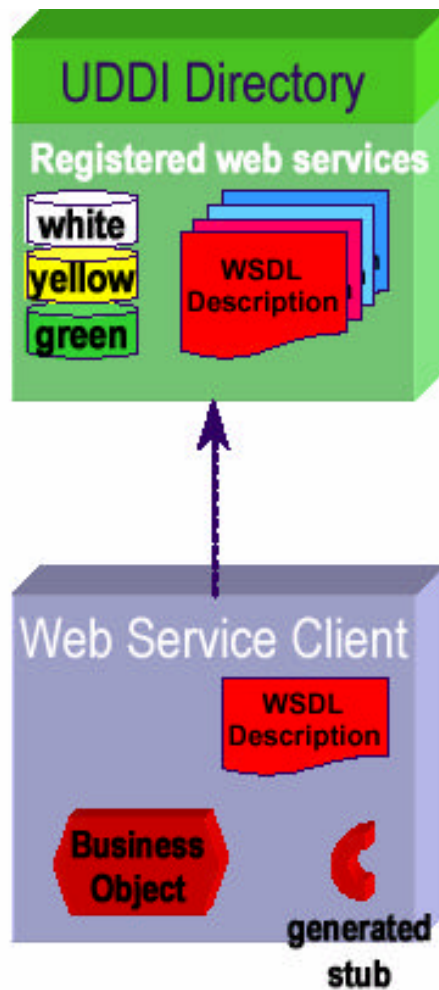
1: Déploiement du WS



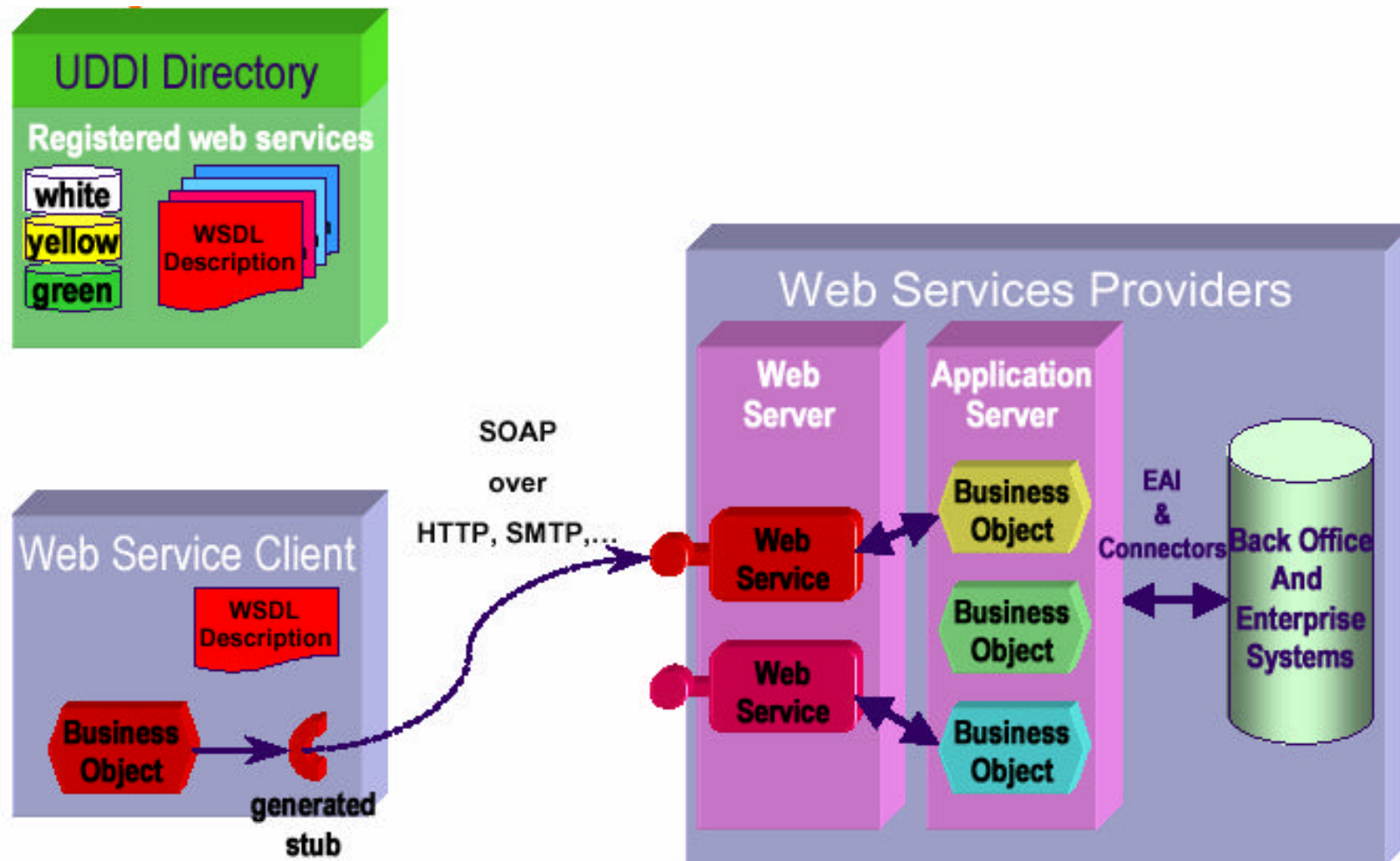
2: Enregistrement du WS



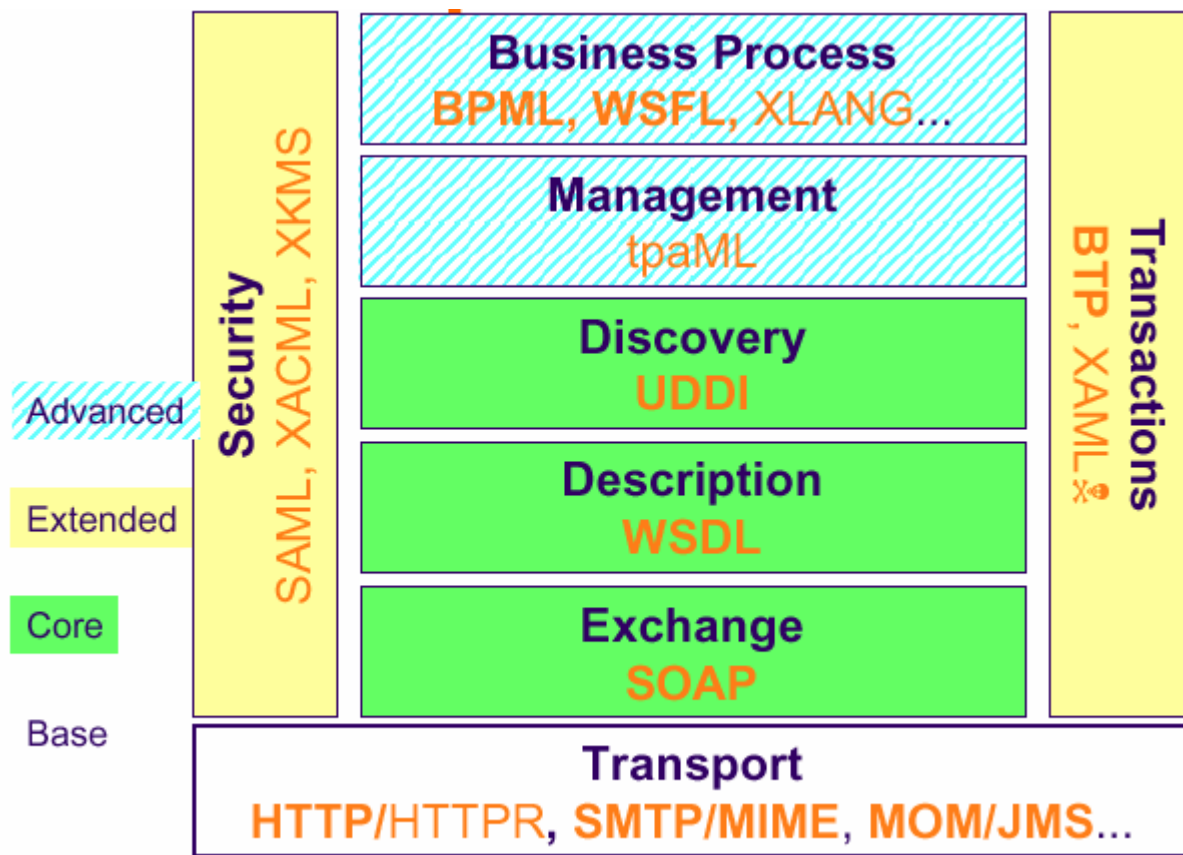
3: Découverte du WS



4: Invocation du WS



Architecture globale



D'après M. Pontacq, Evidian



Les services Web

SOAP : Simple Object Access Protocol

Merci à Michel Riveill et Didier Donsez



Le Web et le client serveur

- Proposition Web actuelle insuffisante
- Autres plates-formes client / serveur
 - Java RMI
 - mono-langage : Java, multi-plateforme (JVM), SUN
 - Pas réaliste pour une application industrielle (performance, sécurité, ...)
 - 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 version 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 »
 - ATTENTION : on perd le contrôle d'accès à faible granularité
 - 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
 - Vous pouvez écrire votre 1^{er} appel SOAP en moins d'une heure !!
 - Il vous a fallu combien de temps en CORBA, RMI, DCOM ?

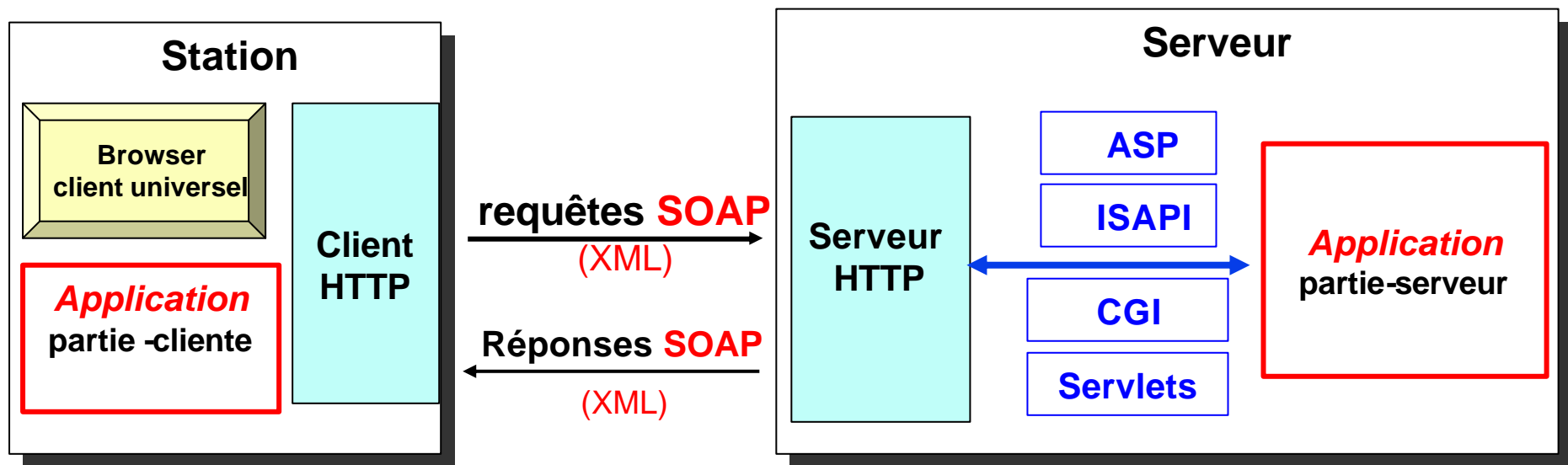


Les 3 aspects d'un appel SOAP

- 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
- Ces vues ne sont pas imposées par le protocole

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 offre un niveau de sécurité simple et effectif
- 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: 13

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)
 - ATTENTION : le texte est globalement peu lisible et vite complexe pour un humain
- 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

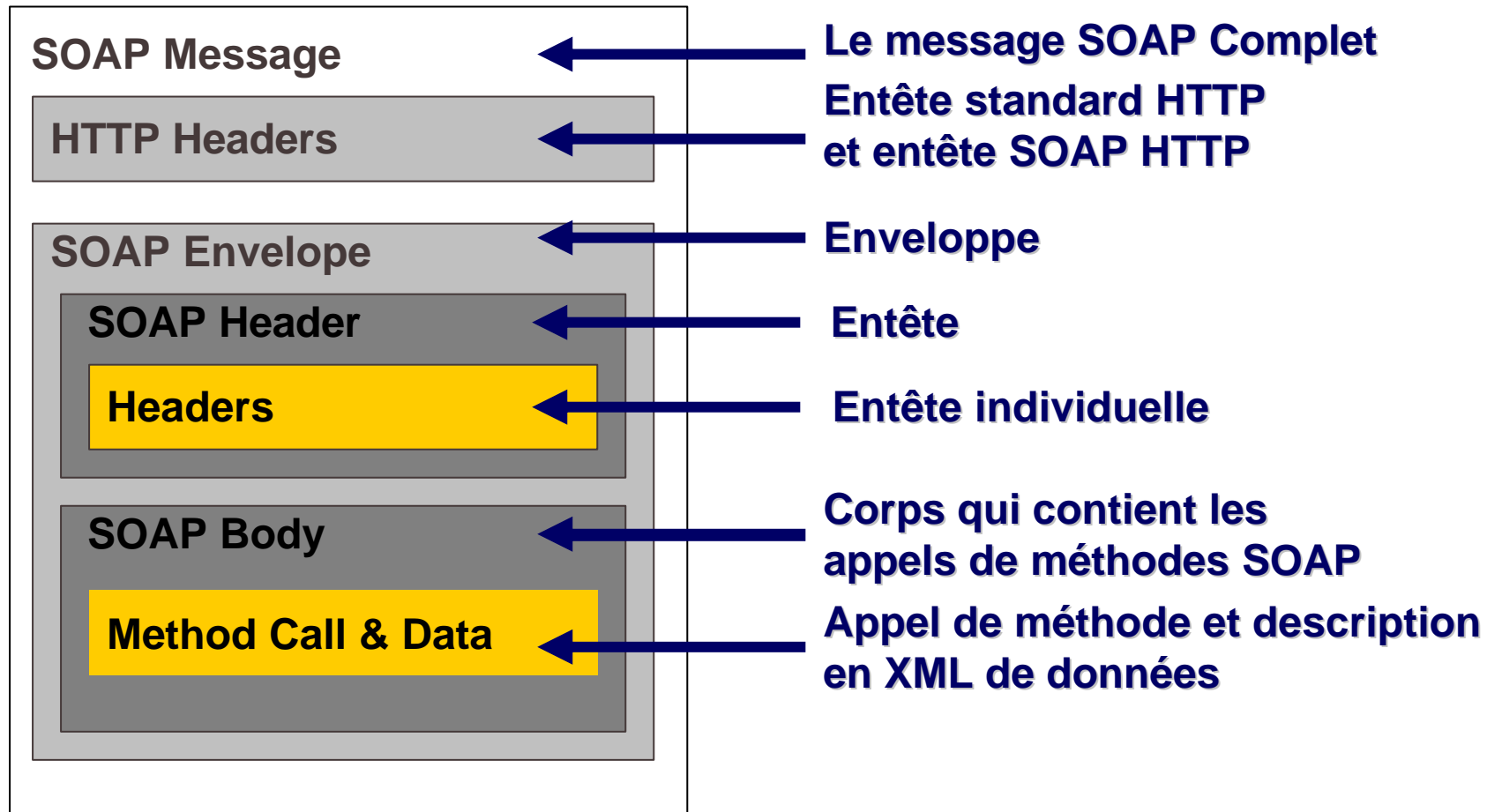
```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/" />
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse
      xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>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



Éléments 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





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-ENV <http://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)



Entête d'un Message

- Contient des entrées non applicatives
 - Transactions, sessions, ...
- L'attribut mustUnderstand
 - Si absent ou = 0
 - l'élément est optionnel pour l'application réceptrice
 - Si =1
 - l'élément doit être compris de l'application réceptrice
 - Si ce n'est pas le cas, le traitement du message par le récepteur doit échouer

- Exemple

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



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

- Décrit dans la section 5 pour les types de base
 - Règles par défaut
 - L'encodage est de type : element-normal-form
 - Un élément par champ, le nom de l'élément est celui du champ
- Les règles d'encodage définissent un système de type
 - Il est compatible avec le langage SDL de XML (XML Schema Definition language)
 - Les types SOAP peuvent être décrit en utilisant XSD
 - SOAP utilise les conventions XSD pour associer les instances aux types

```
<foo  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
      xsi:type="timeInstant">
      1999-11-12T09:43
</foo>
```
 - 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>
```

- **Énumération**

```
<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>
```



Règles d'encodage

- **Tableaux**

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

- **Tableaux d'octet**

```
<picture xsi:type="SOAP-ENC:base64">  
  aG93IG5vDyBicm73biBjb3cNCg==  
</picture>
```

- **Tableaux creux**

```
<SOAP-ENC:Array id="array-1"  
  SOAP-ENC:arrayType="xsd:string[10,10]">  
  <item SOAP-ENC:position="[2,2]">Third row, third col</item>  
  <item SOAP-ENC:position="[7,2]">Eighth row, third col</item>  
</SOAP-ENC:Array>
```



Exemple d'encodage (1/3)

```
<element name="Person" base="tns:Person"/>
  <complexType name="Person">
    <sequence minOccurs="0" maxOccurs="1">
      <element name="name" type="xsd:string"/>
      <element name="address" type="tns:Address"/>
    </sequence>
    <attribute name="href" type="uriReference"/>
    <attribute name="id" type="ID"/>
    <anyAttribute namespace="##other"/>
  </complexType>
```



Exemple d'encodage (2/3)

```
<element name="Address" base="tns:Address"/>
  <complexType name="Address">
    <sequence minOccurs="0" maxOccurs="1">
      <element name="street" type="xsd:string"/>
      <element name="city" type="xsd:string"/>
      <element name="state" type="xsd:string"/>
    </sequence>
    <attribute name="href" type="uriReference"/>
    <attribute name="id" type="ID"/>
    <anyAttribute namespace="##other"/>
  </complexType>
```



Exemple d'encodage (3/3)

```
<element name="Book" type="tns:Book"/>
  <complexType name="Book">

    <sequence minOccurs="0" maxOccurs="1">
      <element name="title" type="xsd:string"/>
      <element name="firstauthor"
        type="tns:Person"/>
      <element name="secondauthor"
        type="tns:Person"/>
    </sequence>
    <attribute name="href"
      type="uriReference"/>
    <attribute name="id" type="ID"/>
    <anyAttribute namespace="##other"/>
  </complexType>
```



Valeurs empaquetées et valeurs indépendantes

- SOAP permet d'encoder les données de différentes manières
 - Par exemple dans `foo.doit(obj1, obj1)`
 - On peut vouloir préciser que les deux paramètres effectifs sont identiques
 - Passage par référence dans les langages de programmation
 - Utilisation de valeurs indépendantes
- Définition d'une valeur indépendante
 - Elle apparaît dans l'entête (soap:Header) ou dans le corps (soap:Body)
 - Elle a un attribut unique (soap:id)
 - Elle est généralement encodée comme un élément typé
- Utilisation d'une valeur indépendante
 - Utilise l'attribut précédemment défini (soap:href)



Valeurs empaquetées et valeurs indépendantes

```
<soap:Envelope xmlns:soap='uri for soap'>
  <soap:Body>
    <calculateArea xmlns='interfaceURI'>
      <origin><x>23</x><y>34</y></origin>
      <corner><x>23</x><y>34</y></corner>
    </calculateArea>
  </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope xmlns:soap='uri for soap'>
  <soap:Body>
    <calculateArea xmlns='interfaceURI'>
      <origin soap:href='#id1' />
      <corner soap:href='#id1' />
    </calculateArea>
    <Point soap:id='id1' xmlns='someURI'>
      <x>23</x><y>34</y>
    </Point>
  </soap:Body>
</soap:Envelope>
```



Le retour d'erreurs (faults)

- 4 éléments
 - Faultcode (obligatoire)
 - Code d'erreur utilisé par le logiciel (switch(faultcode) { case ...
 - Faultstring (obligatoire)
 - Explication lisible d'un humain
 - faultactor (optionel)
 - Erreur en cours de cheminement du message (firewall, proxy, MOM)
 - Detail
 - Détail de l'erreur non lié au Body du message
 - Autres
 - D'autres éléments qualifiés par un namespace peuvent être ajoutés
- Faultcode
 - 4 groupes de code d'erreur
 - Client, Server, MustUnderstand, VersionMismatch
 - Ex: `Client.Authentication`



Le retour d'erreurs (faults)

- MustUnderstand

HTTP/1.1 500 Internal Server Error

Content-Type: text/xml; charset="utf-8"

Content-Length: nnnn

```
<SOAP-ENV:Envelope
  xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:MustUnderstand</faultcode>
      <faultstring>SOAP Must Understand
                          Error</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



Le retour d'erreurs (faults)

- Erreur sur le corps

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
```

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>Server Error</faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="Some-URI">
          <message> My application didn't
                        work </message>
          <errorcode>1001</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



SOAP sur HTTP

- Utilise le modèle POST Requête/Réponse
- Requête
 - Type MIME : `text/xml`
 - Champs d'entête supplémentaire de la requête
 - SOAPAction : URI
 - SOAPAction:** `"http://electrocommerce.org/abc#MyMessage"`
 - SOAPAction:** `"myapp.sdl"`
 - SOAPAction:** `" "`
 - SOAPAction:**
 - Envelope SOAP
- Réponse
 - Status
 - 2xx : le récepteur a correctement reçu, compris et accepté le message inclus
 - 500 (Internal Server Error): le récepteur n'accepte pas le message
 - Envelope SOAP
 - La réponse
 - Le détail des erreurs



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>
```



```
<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
- Il est possible d'utiliser des règles d'encodage spécifiques

```
<foo soap:encodingStyle='myuri' >
    Here is some random XML!!
</foo>
```



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
- Chaque développeur, choisi de rendre visible telle ou telle méthode
- 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



Un peu d'histoire

- **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*



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
- On peut construire des composants acceptant des requêtes SOAP
 - Brique de base
 - Canal de communication (channel)
 - sérialisation / dé-sérialisation (serializer)
 - Aiguilleur (dispatcher)

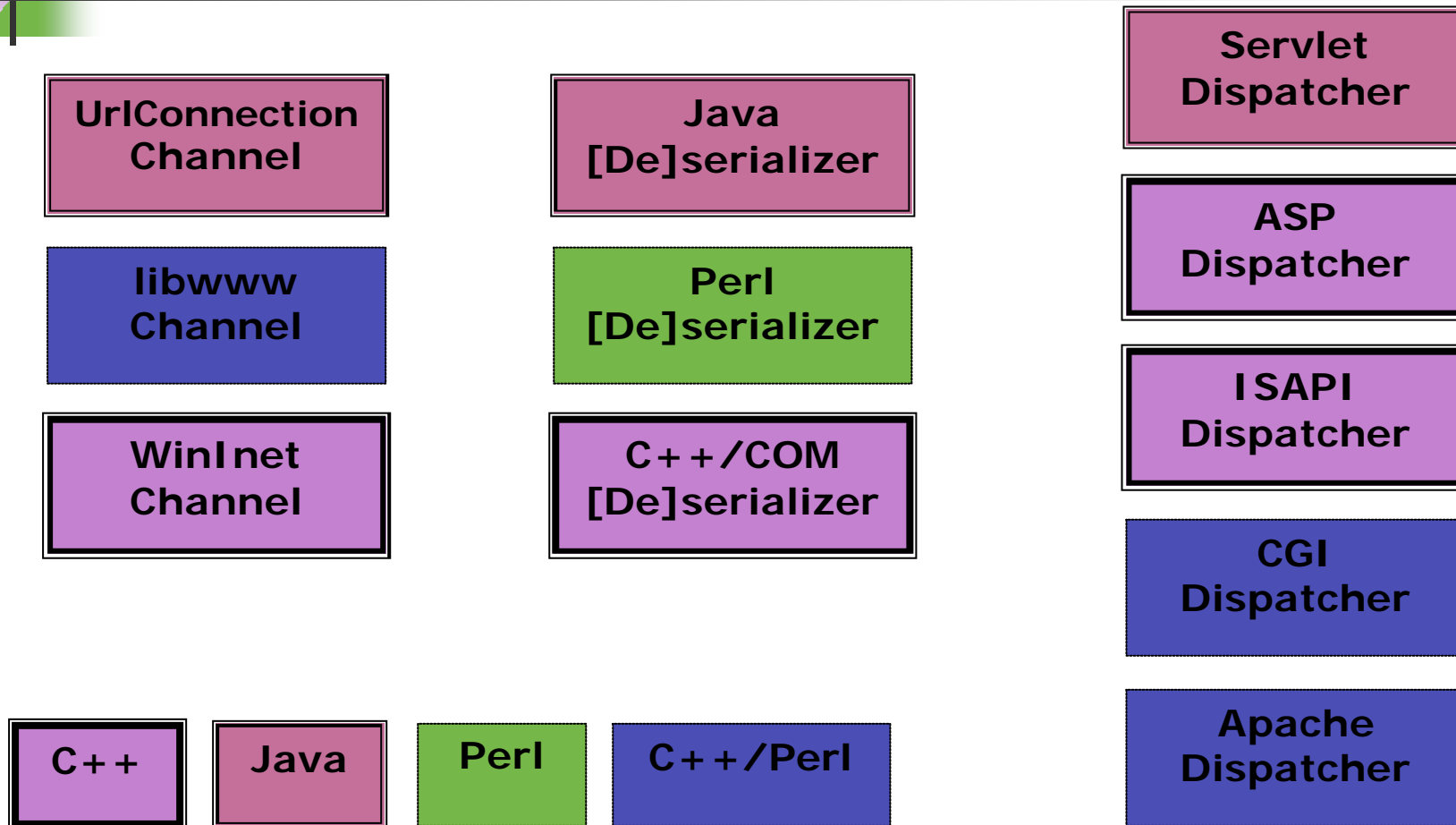


Implémentations de SOAP

See <http://www.Soapware.org>

- JAXM, JAX-RPC RI
- Apache SOAP 2.2
- Apache AXIS
- SoapRMI
- SOAPDirect
- InstantXML
- .Net
- ...

soap implementation techniques





Comparaison

	RMI	RPC	DCOM	CORBA	SOAP
Qui	SUN	SUN/OSF	MicroSoft	OMG	W3C
Plate-formes	Multi	Multi	Win32	Multi	Multi
Langages de Programmation	Java	C, C++, ...	C++, VB, VJ, OPascal, ...	Multi	Multi
Langages de Définition de Service	Java	RPCGEN	ODL	IDL	XML
Réseau	TCP, HTTP, IIOP customisable	TCP, UDP	IP/IPX	GIOP, IIOP, Pluggable Transport Layer	RPC, HTTP, SNMP
Firewall	Tunneling HTTP			HTTP Tunneling CORBA Firewall	HTTP
Nommage	RMI, JNDI, JINI	IP+Port	IP+Nom	COS Naming COS Trader	IP+Port, URL
Transaction	Non	Non	MTS	OTS, XA	Extension applicative dans le header
Extra	Chargement dynamique des classes			Services Communs Services Sectoriels	



Open issues with SOAP

- **Lack** of semantic definitions of Business processes
- **Insufficient** message transfer functionality that would be required for business transactions
 - Security and Reliability
 - ebXML Message Service **complements** SOAP
- Low performance



SOAP 1.2

- Nouvelle spécification pratiquement achevée
- Basé sur les Infoset
 - indépendant de XML 1.0
- Avec les Attachments
- Codes d'erreurs hiérarchiques
- Nouveaux namespaces
- ...



Les services Web

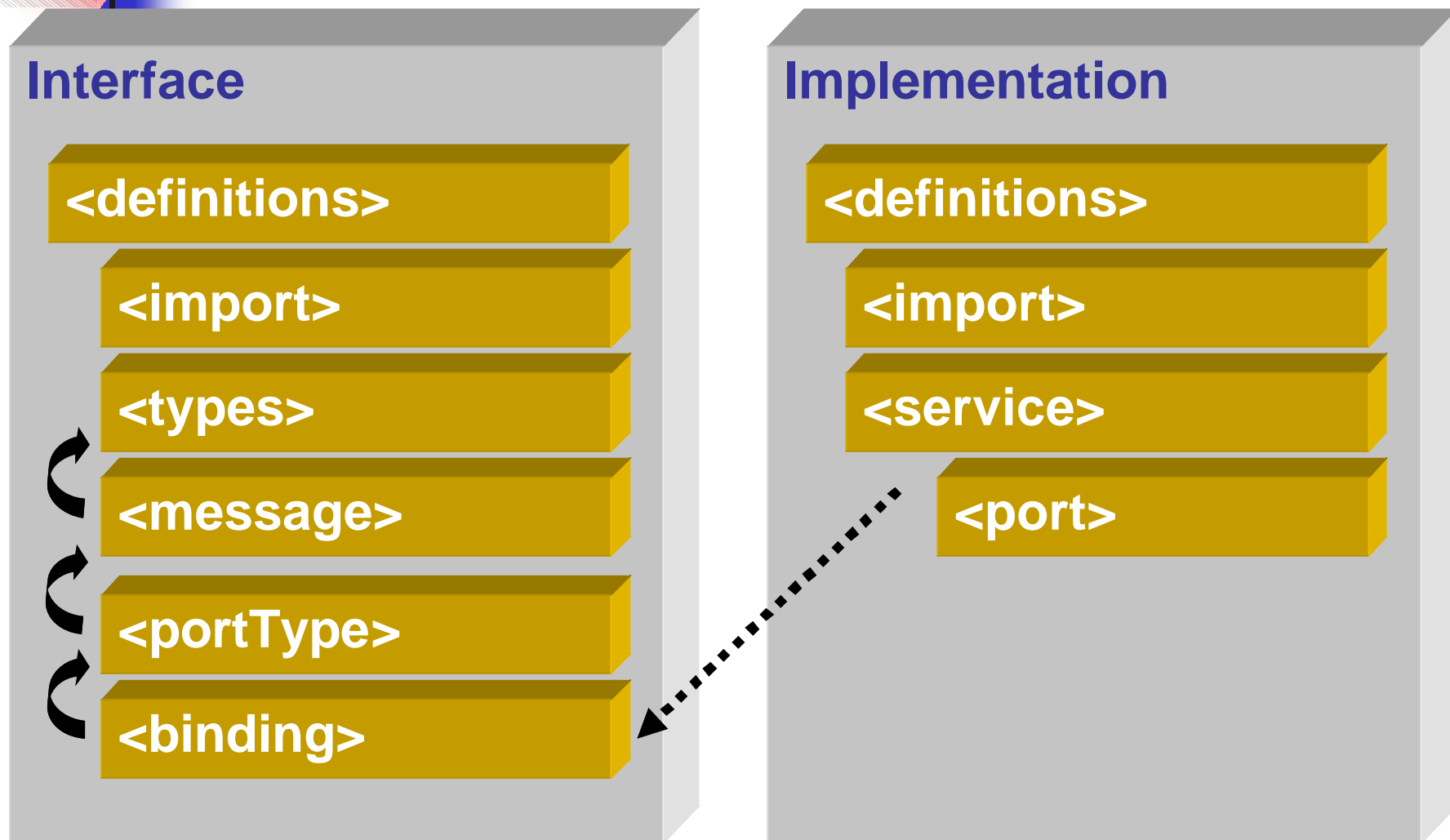
WSDL : Web Services
Description Language

Merci à Didier Donsez



- 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

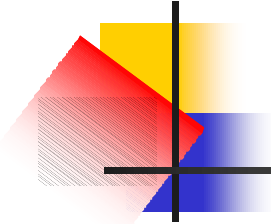
WSDL





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 définitions 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>
```




Élément <message>

- Décrit les noms et types d'un ensemble de champs à transmettre
 - Paramètres d'une invocation, valeur du retour, ...
- Exemple

```
<!-- message declns -->  
<message name="AddEntryRequest">  
  <part name="name" type="xsd:string"/>  
  <part name="address" type="typens:address"/>  
</message>
```

```
<message name="GetAddressFromNameRequest">  
  <part name="name" type="xsd:string"/>  
</message>
```

```
<message name="GetAddressFromNameResponse">  
  <part name="address" type="typens:address"/>  
</message>
```



Élément <porttype>

- Décrit un ensemble d'opérations.
- Plusieurs types d'opérations
 - **One-way**
 - Le point d'entrée reçoit un message (<input>).
 - **Request-response**
 - Le point d'entrée reçoit un message (<input>) et retourne un message corrélé (<output>) ou un ou plusieurs messages de faute (<fault>).
 - **Solicit-response**
 - Le point d'entrée envoie un message (<output>) et recoit un message corrélé (<input>) ou un ou plusieurs messages de faute (<fault>).
 - Binding HTTP : 2 requêtes HTTP par exemple
 - **Notification**
 - Le point d'entrée envoie un message de notification (<output>)
- Paramètres
 - Les champs des messages constituent les paramètres (in,out, inout) des opérations



Élément <porttype>

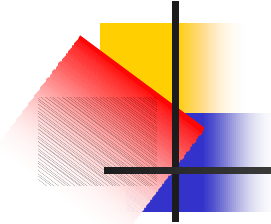
■ Exemple

```
<!-- port type declns -->
<portType name="AddressBook">

  <!-- One way operation -->
  <operation name="addEntry">
    <input message="AddEntryRequest"/>
  </operation>

  <!-- Request-Response operation -->
  <operation name="getAddressFromName">
    <input message="GetAddressFromNameRequest"/>
    <output message="GetAddressFromNameResponse"/>
  </operation>

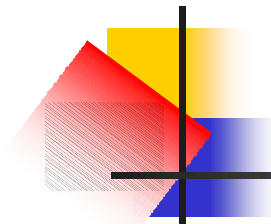
</portType>
```



Élément <binding>

- Spécifie une liaison d'un <porttype> à un protocole concret (SOAP1.1, HTTP GET/POST, MIME, ...).
 - Un porttype peut avoir plusieurs liaisons !
- Exemple de binding sur SOAP et HTTP

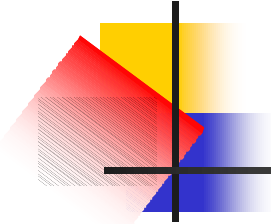
```
<!-- binding declns -->
<binding name="AddressBookSOAPBinding" type="AddressBook">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="addEntry">
    <soap:operation soapAction=""/>
    <input> <soap:body use="encoded" namespace="urn:AddressFetcher2"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" /> </input>
    <output> <soap:body use="encoded" namespace="urn:AddressFetcher2"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" /> </output>
  </operation>
  <operation name="getAddressFromName">
    <soap:operation soapAction=""/>
    <input> <soap:body use="encoded" namespace="urn:AddressFetcher2"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" /> </input>
    <output> <soap:body use="encoded" namespace="urn:AddressFetcher2"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" /> </output>
  </operation>
</binding>
```



Élément <binding>

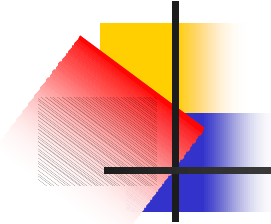
- Exemple de binding avec SOAP et SMTP

```
<definitions ...>
  <types>
    <schema targetNamespace="http://stockquote.com/stockquote.xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="SubscribeToQuotes">
        <complexType><all><element name="tickerSymbol" type="string"/></all></complexType>
      </element>
      <element name="SubscriptionHeader" type="uriReference"/>
    </schema>
  </types>
  <message name="SubscribeToQuotes">
    <part name="body" element="xsd1:SubscribeToQuotes"/>
    <part name="subscribeheader" element="xsd1:SubscriptionHeader"/>
  </message>
  <portType name="StockQuotePortType">
    <operation name="SubscribeToQuotes">
      <input message="tns:SubscribeToQuotes"/>
    </operation>
  </portType>
  ...
```



Élément <binding>

```
...
<binding name="StockQuoteSoap" type="tns:StockQuotePortType">
  <soap:binding style="document" transport="http://stockquote.com/smtp"/>
  <operation name="SubscribeToQuotes">
    <input message="tns:SubscribeToQuotes">
      <soap:body parts="body" use="literal"/>
      <soap:header message="tns:SubscribeToQuotes" part="subscribeheader"
        use="literal"/>
    </input>
  </operation>
</binding>
<service name="StockQuoteService">
  <port name="StockQuotePort" binding="tns:StockQuoteSoap">
    <soap:address location="mailto:subscribe@stockquote.com"/>
  </port>
</service>
</definitions>
```



Élément <service>

- Une collection de points d'entrée (endpoint) relatifs
- Exemple

```
<?xml version="1.0" ?>
<definitions name="urn:AddressFetcher"
    targetNamespace="urn:AddressFetcher2"
    xmlns:typens="urn:xml-soap-address-demo"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    ...
    <!-- service decln -->
    <service name="AddressBookService">
        <port name="AddressBook" binding="AddressBookSOAPBinding">
            <soap:address location="http://www.mycomp.com/soap/servlet/rpcrouter"/>
        </port>
    </service>
</definitions>
```



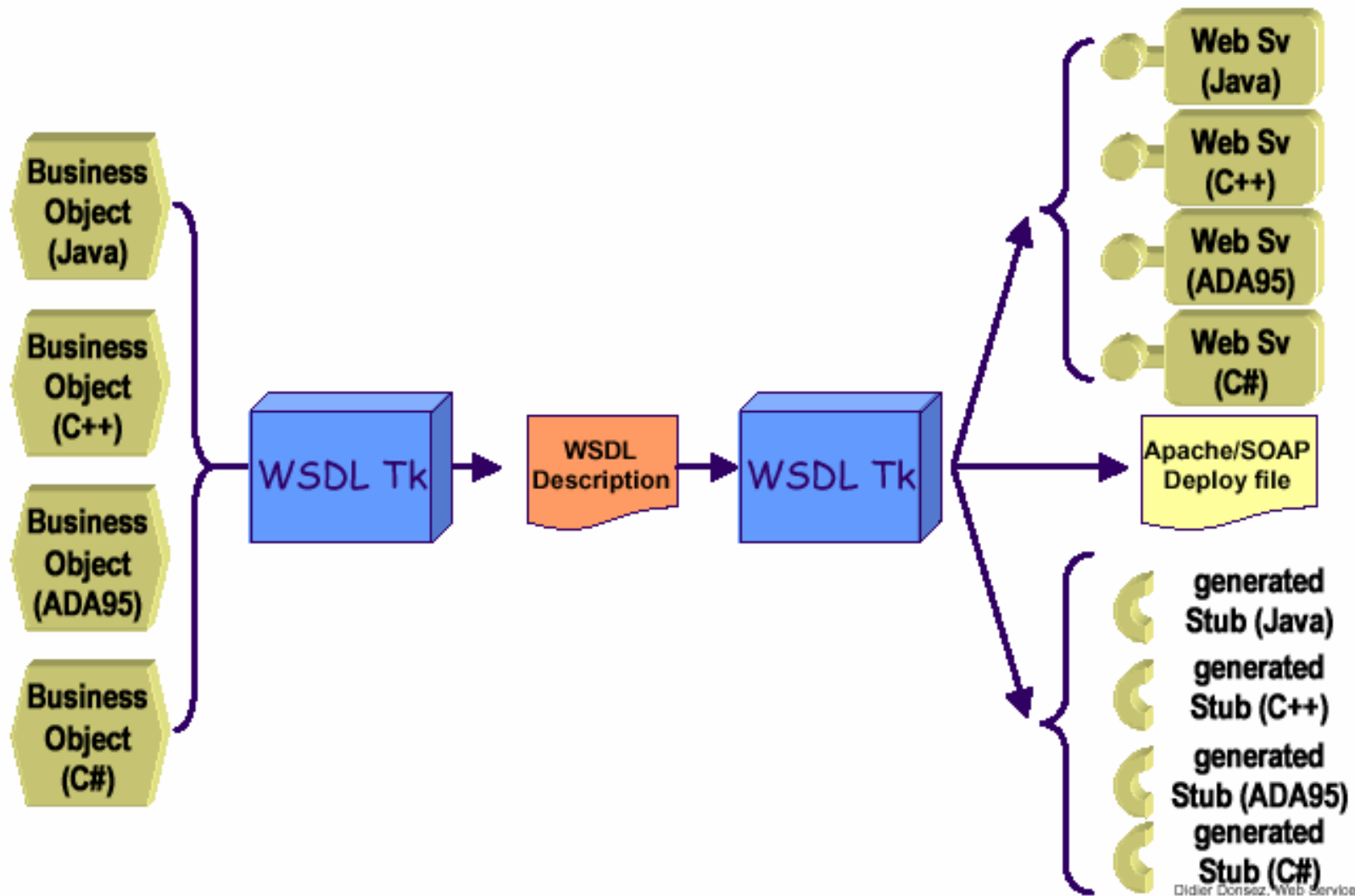
- Open – allows for other namespaces and thus highly extensible
- Ability to import other schemas & WSDL
- Provides “recipe” for Web Services
- Provides both interface and implementation details
- Allows for separation of the two



Outils

- Générateur WSDL à partir de déploiement SOAP ou EJB, ...
- Générateur de proxy SOAP à partir de WSDL
- Toolkits (WsdI2Java / Java2WsdI, ...)
 - Propriétaires (non normalisés)

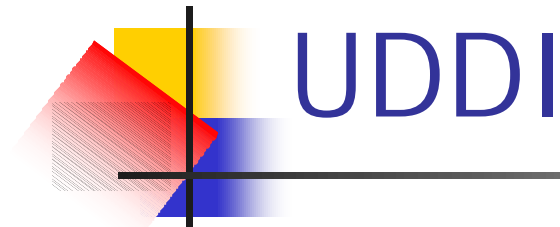
Génération WSDL



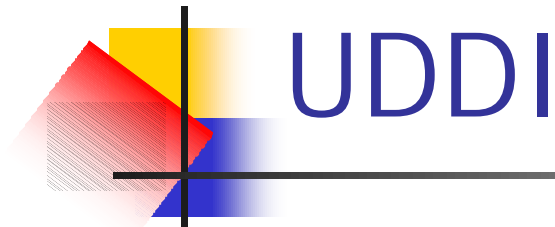


Les services Web

UDDI :
Universal Description, Discovery
and Integration



- Spécification (09/2000)
 - Ariba, IBM, Microsoft +260 autres sociétés
- Objectifs
 - annuaire mondial d'entreprises pour permettre d'automatiser les communications entre prestataires, clients, etc.
 - plusieurs entrées indexées : nom, carte d'identité des sociétés, description des produits, services applicatifs invocables à distance (références des connexions)
 - Indexation des catalogues propriétaires (ebXML, RosettaNet, Ariba, Commerce One, etc.)
- Grammaire XML (schéma XML)
 - Soumission/interrogation basées sur SOAP et WSDL



- **Worldwide directory of companies, services, products...**
 - White pages, Yellow pages, Green pages
- **Green pages**
 - Namespace to describe how to use the service, etc...
 - Identifier of who published the service
 - Unique identifier (tModelKey) of this service for registration
- **Accessing web services**
 - Bindings declared in directory entries:
 - for example, (tModelKey, URL) associations
- **UDDI directories, search engines**
 - xmethods.net, soapware.org, salcentral.com, soap-wrc.com, ...

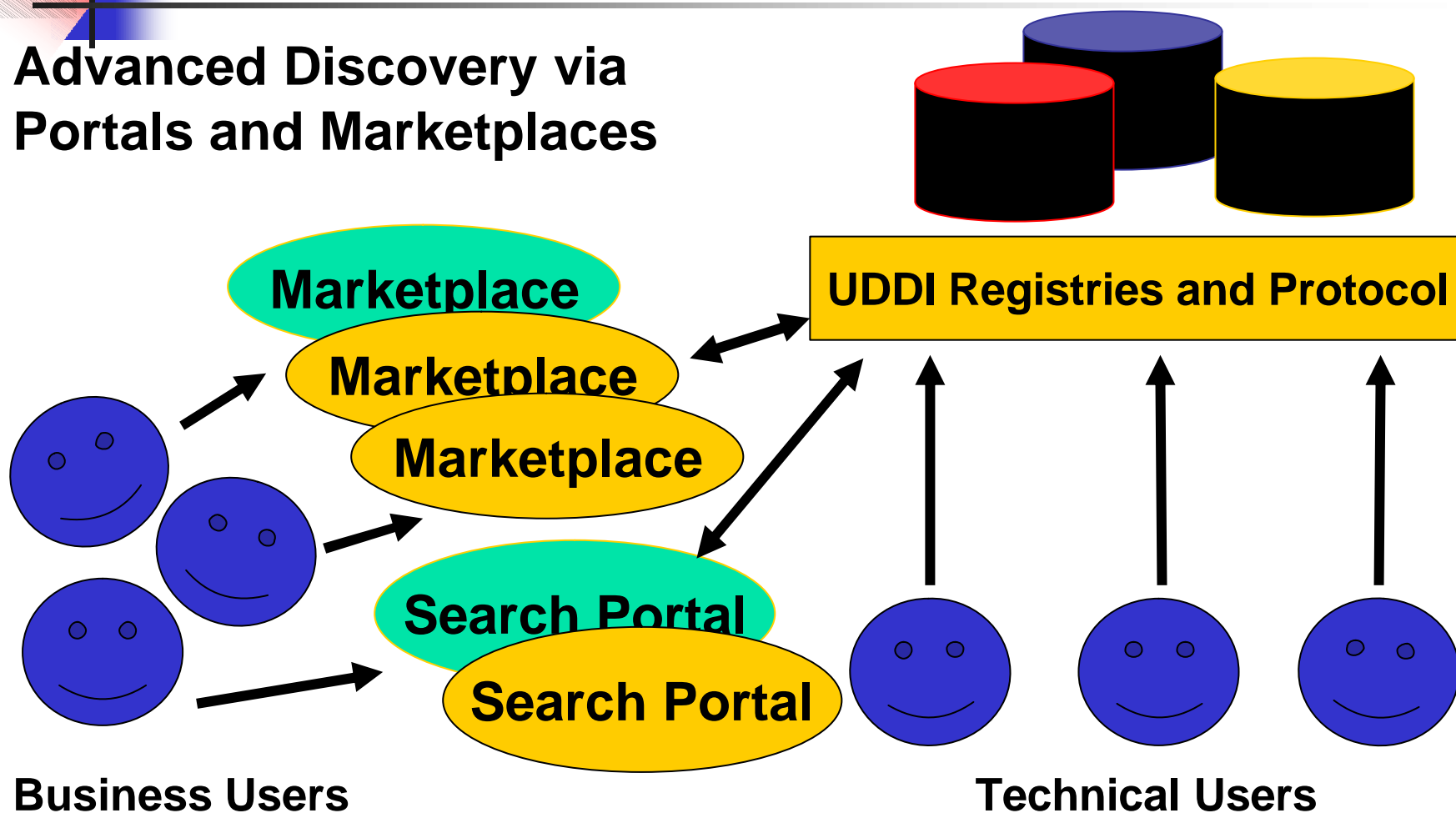


White, yellow and green pages

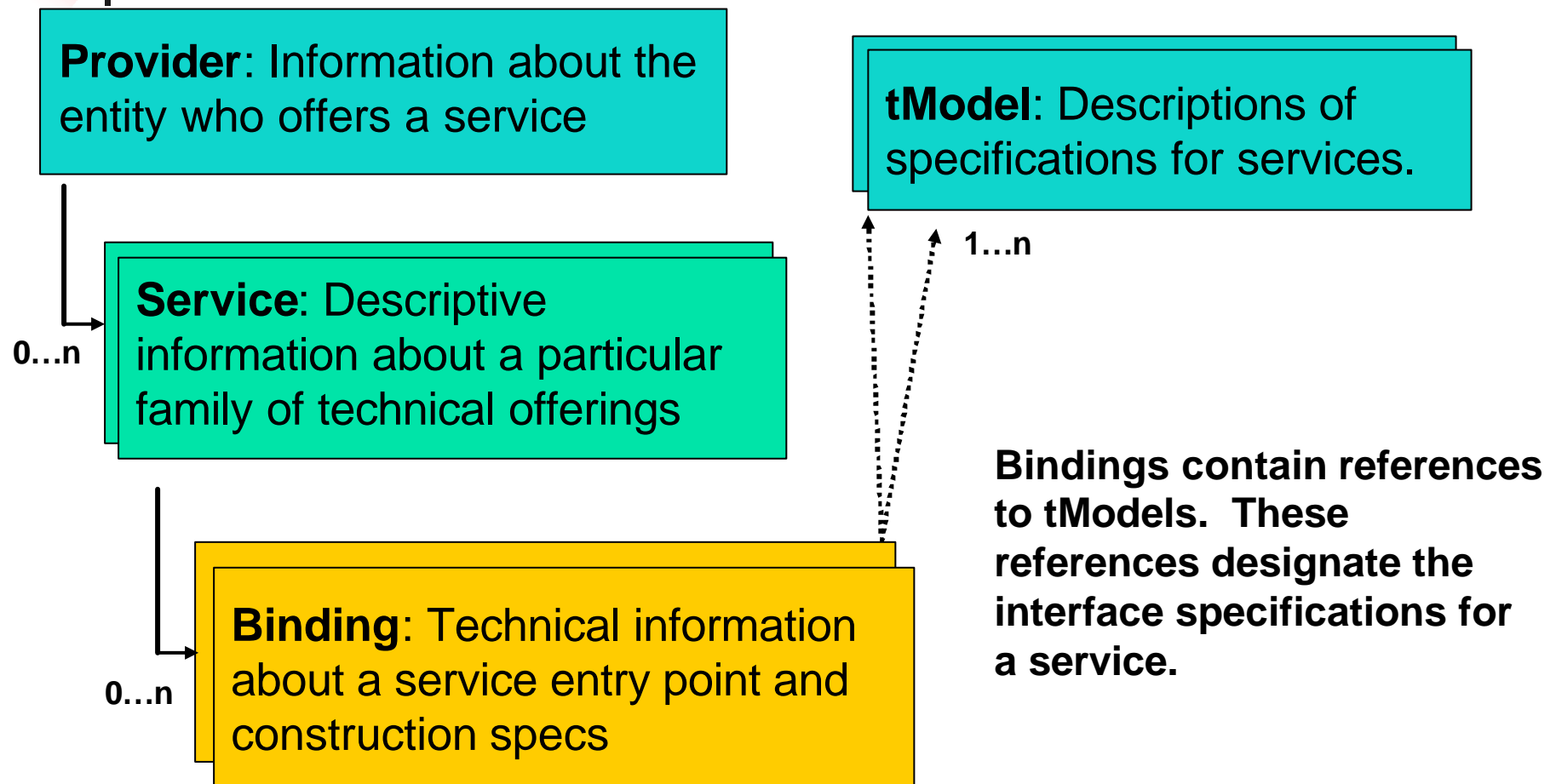
- « White pages »
 - address, contact, and known identifiers
- « Yellow pages »
 - industrial categorizations
 - Industry: NAICS (Industry codes - US Govt.)
 - Product/Services: UN/SPSC (ECMA)
 - Location: Geographical taxonomy
- « Green pages »
 - technical information about services

UDDI : The vision

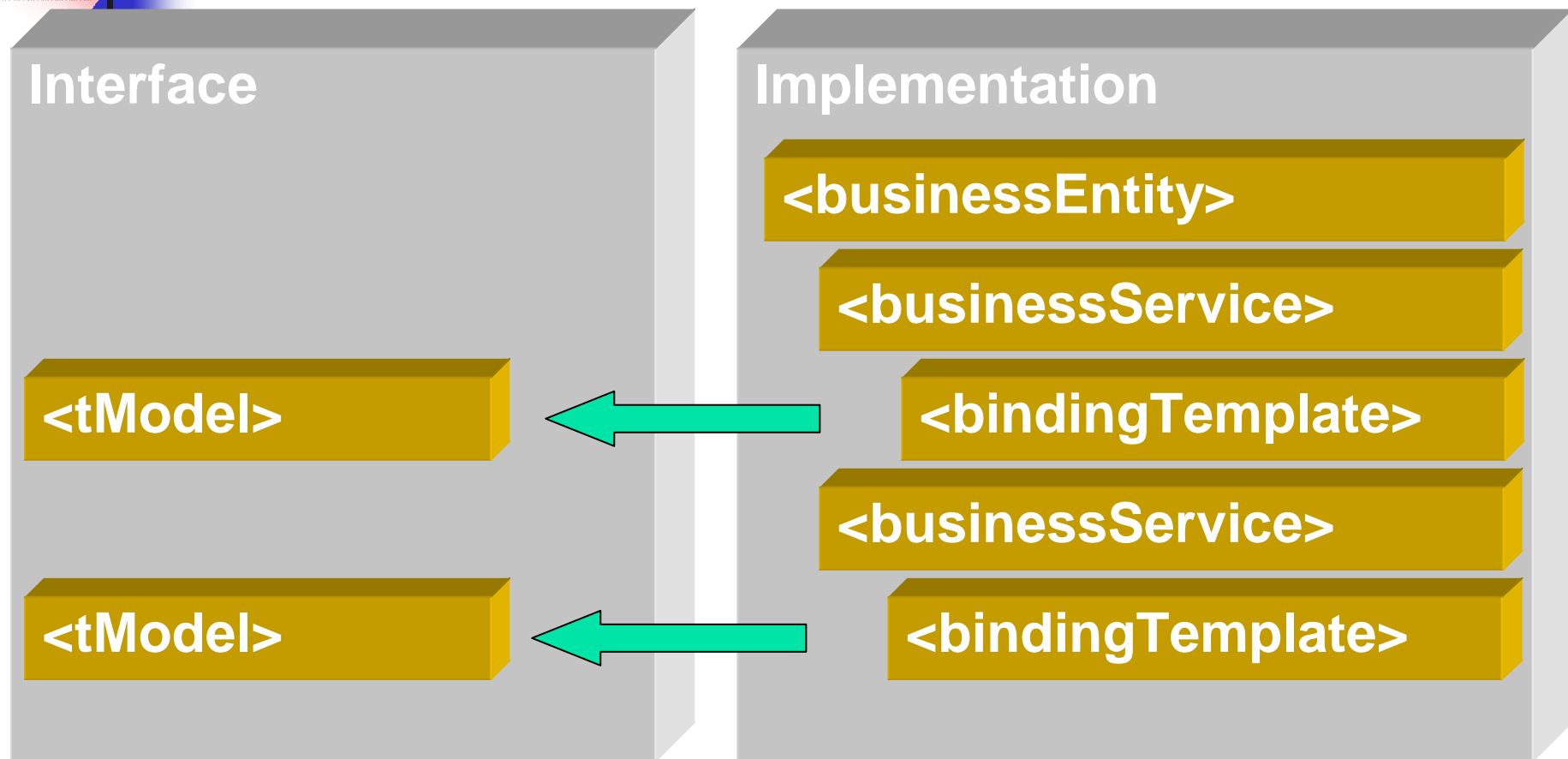
**Advanced Discovery via
Portals and Marketplaces**



UDDI Information Model



UDDI Schema





How UDDI Works: tModel

- tModel = Technology Model
- Generic meta-data structure to uniquely represent any concept or construct
- Also includes interface protocol definitions
- Powerful abstraction modeling system
- Examples: WSDL files, XML schema, namespaces, categorization schemes



UDDI : <tModel>

- <tModel> represents meta-data and interfaces

```
<tModel xmlns="urn:uddi-org:api" tModelKey="UUID:AAAAAAA-AAAA-
AAAA-AAAA-AAAAAAAAAAAA">
  <name>microsoft-com:creditcheck</name>
  <description xml:lang="en">Check credit limits</description>
  <overviewDoc>
    <overviewURL>http://schema.com/creditcheck.wsdl
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference
      tModelKey="UUID:CD153257-086A-4237-B336-6BDCBDCC6634"
      keyName="Consumer credit gathering or reporting services"
      keyValue="84.14.16.01.00"/>
    <keyedReference
      tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
      keyName="types" keyValue="wsdlSpec"/>
  </categoryBag>
</tModel>
```



Providers, Services And Bindings

- Providers
 - Examples: Accounting Department, Corporate Application Server
 - Name, Description, Contact Information
 - Categorization and Identification Information
- Services
 - Examples: Purchase Order services, Payroll services
 - Name, Description(s)
 - Categorization Information
- Bindings
 - Description(s), access points, parameters
 - Examples: Access Point (<http://...>) for Web Service



<bindingTemplate>

- <bindingTemplate> represents data and implementation details

```
<bindingTemplate serviceKey="33c3d124-e967-4ab1-  
8f51-d93d95fac91a" bindingKey="48f2bc6b-a6de-4be8-  
9f2b-2342aeafaaac">  
  <accessPoint URLType="http">  
    http://localhost/HelloWorld/Service1.asmx  
  </accessPoint>  
  <tModelInstanceDetails>  
    <tModelInstanceInfo tModelKey="uuid:64c756d1-  
3374-4e00-ae83-ee12e38fae63"/>  
  </tModelInstanceDetails>  
</bindingTemplate>
```



Important UDDI Features

- Neutral in terms of protocols – as a registry, it can contain pointers to *anything*
- Can search by business, service, Web Service (tModel), binding
- Usage of Globally Unique Identifiers (GUIDs)
- Specification allows public and private nodes
- Delineation between interface and implementation



Les services Web

Implémentations :

Les APIs Java :

JAXP, JAX-RPC, JAXM, JAXR, JAXB



Les APIs Java pour les SW

- Service Description (WSDL)
 - JSR 110 (Java API for WSDL)
 - JAX-RPC (Java TM API for XML-based RPC)
- Service Registration and Discovery (UDDI, ebXML Reg/Rep)
 - JAXR (Java TM API for XML Registry)
- Service Invocation (SOAP, ebXML Message Service)
 - JAXM (Java TM API for XML Messaging), JAX-RPC



Les APIs Java pour les SW

- Electronic business framework
 - JAXM with ebXML Message Service (TR&P) profile
 - JAXR with ebXML Registry/Repository
 - JSR-157: Java API for ebXML CPP/CPA
- J2EE TM Web Services Framework
 - JSR 151: J2EE 1.4
 - JSR 109: (Enterprise Web services)
 - JAXM, JAX-RPC



Les APIs Java pour les SW

- XML document management
 - JAXP (Java API for XML processing)
 - JAXB (Java API for XML data-binding)
- Security
 - XML Encryption (JSR 106)
 - XML Digital Signature (JSR 105)
 - XML Trust Service (JSR 104)
 - Web services security assertions (JSR 155)



Les services Web

Implémentation avec JAX-RPC

From Sun ...



JAX-RPC : cycle de développement

- 1. Service endpoint definition
- 2. Service endpoint implementation
- 3. Service endpoint deployment
- 4. Service description
- 5. Service publication and discovery



1. Service Endpoint Definition

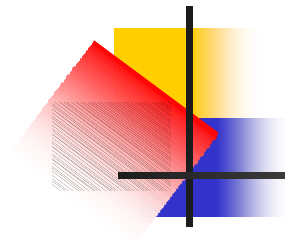
- Specified in **Service Definition Interface**
 - Could be generated from WSDL document using a tool or
 - Could be written in Java programming language directly
- No assumption about type of client that would use this service definition interface
- Exemple : **HelloIF.java**

```
package hello;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface HelloIF extends Remote {
    public String sayHello(String s) throws RemoteException;
}
```



2. Service Implementation

- Service implementation class is an ordinary Java class
- Invocation done inside the servlet container
 - JAX-RPC defines **Servlet-based Endpoint model**
 - Other endpoint models (e.g. stateless session beans) will be defined in J2EE 1.4, JSR 109, EJB 2.1
- Optional **ServiceLifecycle** interface for initialization and destruction callbacks



Example: Service Implementation

HelloImpl.java :

```
package hello;

public class HelloImpl implements HelloIF {
    public String message = new String("Hello ");

    public String sayHello(String s) {
        return new String(message + s);
    }
}
```



3. Service Deployment

- Deployed on JAX-RPC runtime system
- Generates of **container specific artifacts (skeleton or tie class)** based on the service definition interface
- Configures one or more **protocol bindings** for the JAX-RPC service
- Creates one or more **service ports** (or endpoints) for this JAX-RPC service



4. Generate Service Description

- JAX-RPC service endpoint mapped to WSDL service description
- WSDL enables export of a JAX-RPC service endpoint across heterogeneous environments
- JAX-RPC specifies the standard mapping between WSDL and Java:
 - Mapping between service endpoint interface and WSDL definitions
 - Binding to a specific protocol and transport



5. Service Publication

- Publish the JAX-RPC service to a registry
 - WSDL document of a service along with meta data



Simple Client

```
package hello;
public class HelloClient {
    public static void main(String[] args) {
        try {
            // Get Stub instance
            HelloIF_Stub stub = (HelloIF_Stub)
                (new HelloWorld_Impl().getHelloIF());
            // Set endpoint address
            stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
                args[0]);
            // Make method invocation
            System.out.println(stub.sayHello("Duke!"));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```



A few things on this simple client

- Instead of getting service implementation "HelloWorld_Impl()" object directly, it could get Service interface "HelloIF" object from Service object, which in turn can be obtained from ServiceFactory
- Instead of using Stub "HelloIF_Stub" for invoking a remote method, it could use Service interface "HelloIF"

```
Service service = ServiceFactory.newInstance().createService(...);  
HelloIF hello= (HelloIF)service.getPort(HelloIF.class);  
hello.sayHello();
```



Client: DII Invocation Model

```
ServiceFactory factory = ServiceFactory.newInstance();
Service service = factory.createService(new QName(qnameService));
QName port = new QName(qnamePort);
Call call = service.createCall();
call.setPortTypeName(port);
call.setTargetEndpointAddress(endpoint);
call.setProperty(Call.SOAPACTION_USE_PROPERTY, new Boolean(true));
call.setProperty(Call.SOAPACTION_URI_PROPERTY, "");
call.setProperty(ENCODING_STYLE_PROPERTY, URI_ENCODING);
QName QNAME_TYPE_STRING = new QName(NS_XSD, "string");
call.setReturnType(QNAME_TYPE_STRING);
call.setOperationName(new QName(BODY_NAMESPACE_VALUE, "sayHello"));
call.addParameter("String_1", QNAME_TYPE_STRING,
    ParameterMode.PARAM_MODE_IN);
String[] params = { new String("Duke!") };
String result = (String)call.invoke(params); System.out.println(result);
```

The logo consists of a stylized 'X' formed by overlapping colored squares (yellow, red, blue) and a black crosshair.

Apache XSOAP

- Open-source implementation of the **SOAP v1.1** and **SOAP Messages with Attachments**
- Not JAXM and JAX-RPC compatible yet
 - Apache SOAP node should interoperate with JAXM node
- Latest version is v2.2

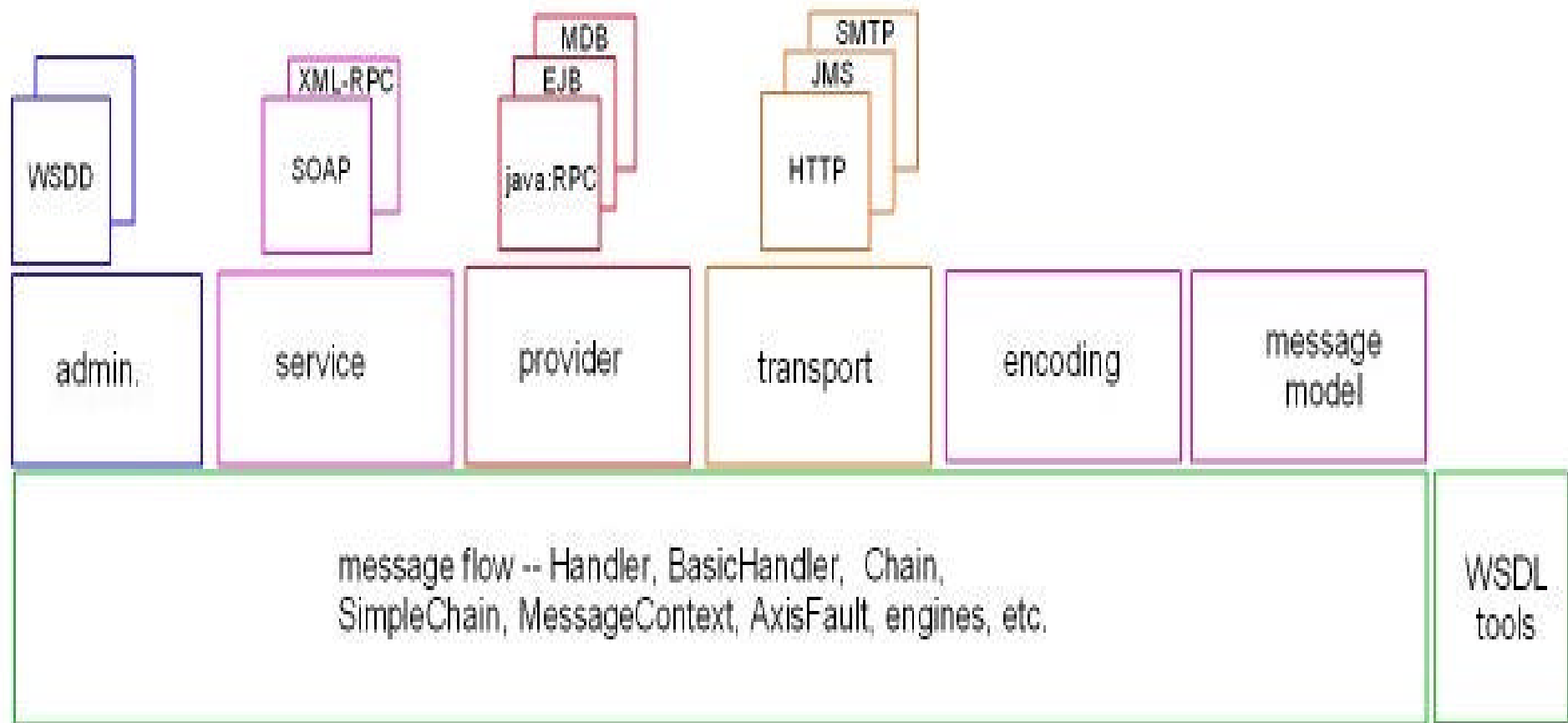
The logo for Apache Axis, featuring a stylized 'A' composed of overlapping red, yellow, and blue squares, with a black crosshair intersecting at the center.

Apache Axis

- Ground-up re-architecture and implementation of SOAP 1.1 and W3C XMLP (SOAP 1.2) effort
- Based on configurable **chains** of **message handlers**
 - Flexible, Modular, User-extensible
- SAX-based XML processing
 - Fast
- Transport framework
- JAX-RPC compliant (Technology Compatibility Kit)
- Axis 1.0 available (7 October 2002)



L'architecture d'Axis

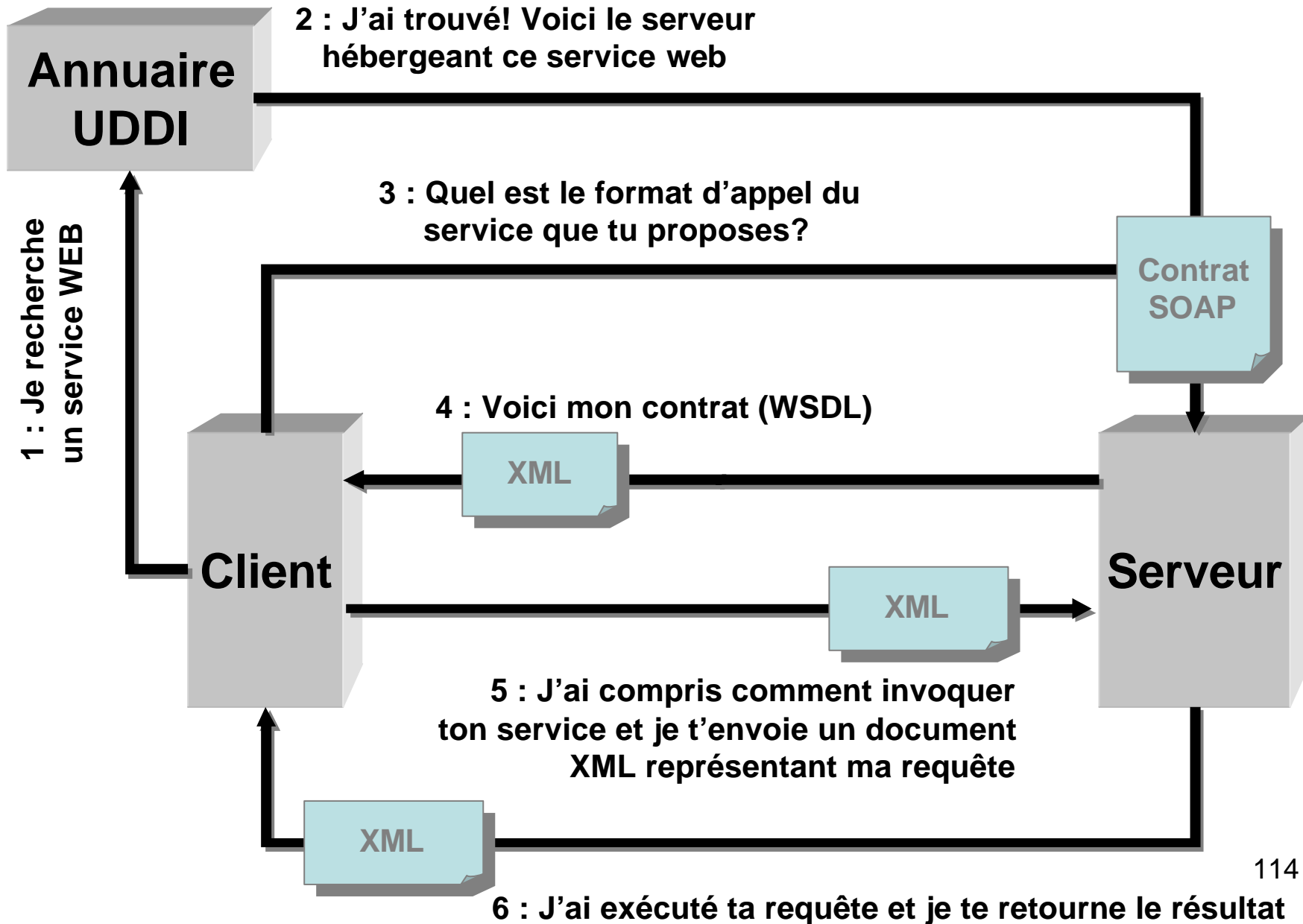




Les services Web

Conclusion

Et on boucle la boucle !





Les services Web

- La 3ème génération du Web
- Technologies Standards du Web
 - SOAP 1.1 (puis SOAP 1.2)
 - WSDL
- Technologies non standardisées
 - UDDI, DISCO
 - GXA (Global XML Architecture)
 - WSDD, WSFL, ASMX, ...



Références

- Services Web :
 - Spécifications officielles : <http://www.w3c.org>
 - Documentations et exemples en ligne : <http://www.xmlbus.com>
- SOAP :
 - Spécification SOAP 1.1 : <http://www.w3c.org/TR/SOAP>
 - Spécification SOAP 1.2 : <http://www.w3c.org/TR/soap12>
 - Implémentations : <http://www.soapware.org>
 - Exemples en ligne : <http://soapclient.com/soaptest.html>
- UDDI :
 - Spécification : <http://www.uddi.org>
 - Serveur UDDI : <http://uddi.microsoft.com> et <http://uddi.ibm.com>
- Cours :
 - Didier Donsez, Web Services
 - Michel Riveill, SOAP