

# Services Web

*Les services web : première approche*

*Le langage XML*

*Les services web : vue d'ensemble*

# Services Web

## Présentation

# *Les services web : première approche*

*Le langage XML*

*Les services web : vue d'ensemble*

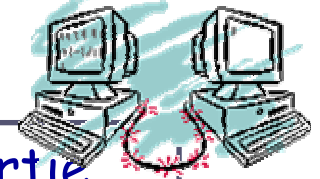


*Génèse : architecture réparties*

*Historique*

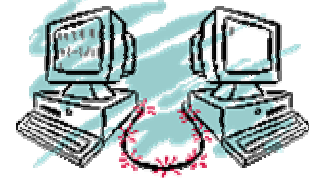
*Principes de base*

# Génèse : architectures réparties



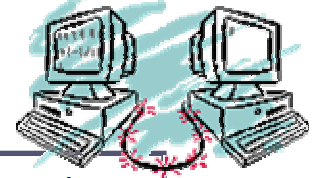
- Aujourd'hui, l'informatique d'entreprise est résolument répartie
- La technologie « client/serveur » rend directement accessibles aux consommateurs, les informations détenues par les fournisseurs
- Par exemple, une banque de dépôt offre à ses clients, la possibilité de consulter leurs comptes par divers moyens :
  - ▶ Mobile (Wap-Wml)
  - ▶ Internet (Http-Html)
  - ▶ Minitel
- Cela implique une architecture adaptative et ouverte
  - ▶ Une même application de traitement (i.e. gestion du compte)
  - ▶ Plusieurs applications de mise en forme suivant le canal de communication (i.e. affichage en WML, en HTML, ...)

# *Génèse : architectures réparties*



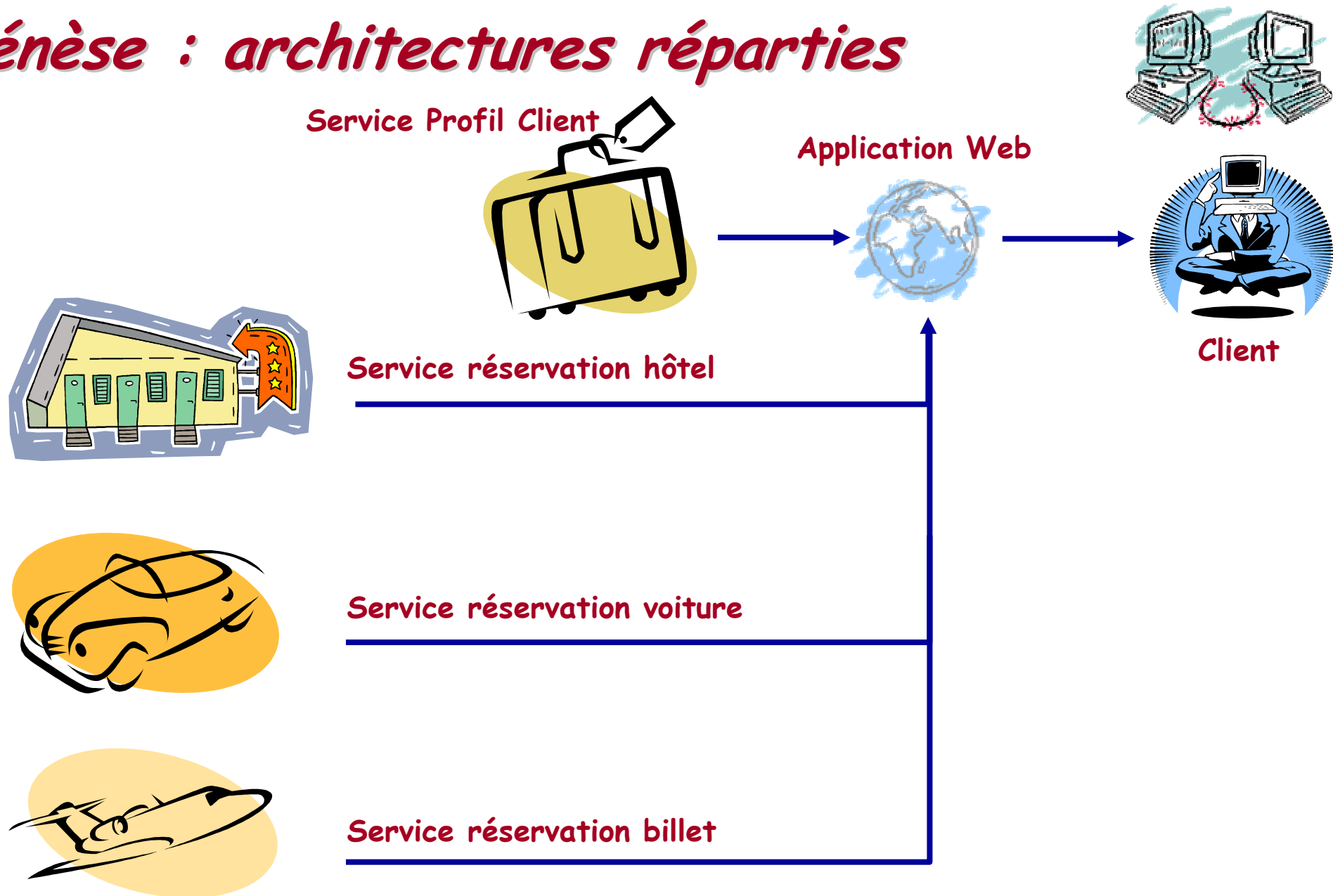
- De la même façon, certaines informations ou données peuvent être publiées pour d'éventuels partenaires afin de commercialiser de nouveaux produits financiers
- Cette nouvelle application à vocation « B2B » s'appuie sur les applications de traitement existantes
- On parle alors d'agrégation applicative qui repose sur l'usage, la combinaison d'applications existantes.

# Génèse : architectures réparties

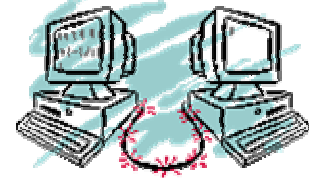


- L'exemple le plus illustre d'architecture agrégée nous est fourni par l'agence de voyage.
- En effet, un produit « voyage » est en réalité une combinaison de plusieurs produits :
  - ▶ Gestion de réservation des billets de transport
  - ▶ Gestion de réservation des hôtels
  - ▶ Gestion de réservation des voitures de location
  - ▶ ...
- L'élaboration d'un produit « voyage » est bien le résultat d'informations récupérées auprès de différents fournisseurs :
  - ▶ Compagnies aériennes
  - ▶ Chaînes hotellières
  - ▶ Loueurs de véhicules
  - ▶ ...

# Génèse : architectures réparties



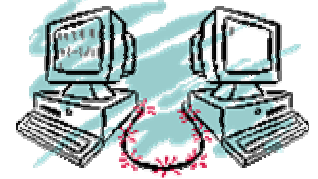
# Génèse : architectures réparties



- Sur le schéma précédent, l'application consultée par le client met en œuvre des applications réparties pour satisfaire sa demande
- Dans cette mise en œuvre applicative, on parle :
  - ▶ De transformation, lorsqu'il s'agit d'adapter le dialogue en fonction du profil utilisateur
  - ▶ D'agrégation, lorsqu'il s'agit de faire appel à des applications proposées par des partenaires ou fournisseurs
- Que ce soit de l'agrégation ou de la transformation d'informations, les applications réalisant ces tâches de façon complètement automatique et opaque s'appellent des « **services web** ».



# Génèse : architectures réparties



- Une application web communicante résulte alors d'un assemblage de services web.
- Certains sont internes et d'autres externes et fournis par divers partenaires et fournisseurs
- Les deux extrémités de cet assemblage sont :
  - ▶ Le tout interne : entièrement composé de services internes, on parle alors d'intégration d'applications d'entreprise ou EAI
  - ▶ Le tout externe : entièrement composé de services externes, on parle alors de portail d'entreprises.

# Historique



- Mais qu'est-ce qui différencie un service web d'une application distante traditionnelle ?
- L'histoire des architectures réparties est ponctuée d'évolutions liées aux :
  - ▶ protocoles d'échanges et d'accès (Corba, DCOM, RMI)
  - ▶ langages d'implémentation (C++, Java, Smalltalk, etc.)
  - ▶ interfaces (d'interaction et de présentation)
- Ces évolutions sont toujours plus riches, plus sophistiquées, et souvent plus complexes
- Elles imposent de fait, des configurations clientes suffisamment robustes aptes à recevoir de telles architectures

# Historique



- A contrario, le web impose des moyens de communication beaucoup plus sommaires :
  - ▶ débit plus faible
  - ▶ caractéristiques protocolaires de HTTP bien inférieures à Corba ou RMI
  - ▶ interaction et présentation bien moins riches (html)
- Les configurations clients sont très légères (un simple navigateur)
- Aujourd'hui, le protocole du web est universellement mis en œuvre.
- Le protocole du web devient le moyen d'échange incontournable dès lors que l'on désire interagir avec autrui (tant en interne qu'en externe)

# Historique



- La question essentielle est de savoir si l'on peut combiner :
  - ▶ Les caractéristiques des architectures distribuées comme corba ou rmi.
    - Ce type d'architecture est devenue indispensable pour répondre aux besoins en terme complexité des applications d'entreprise
  - ▶ Les contraintes imposées par le web, à savoir :
    - client léger et souvent réduit à un simple navigateur
    - mise en œuvre du protocole HTTP



*Les services web permettent d'atteindre cet objectif*

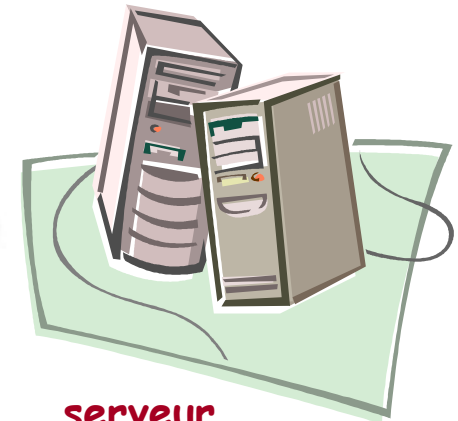
# Principes de base



- Le modèle des services web repose sur le transport d'une demande de service entre un client et un serveur.
- Le transport est assuré par la suite de protocoles « internet/web » constituée de « TCP/IP et HTTP »



client



serveur

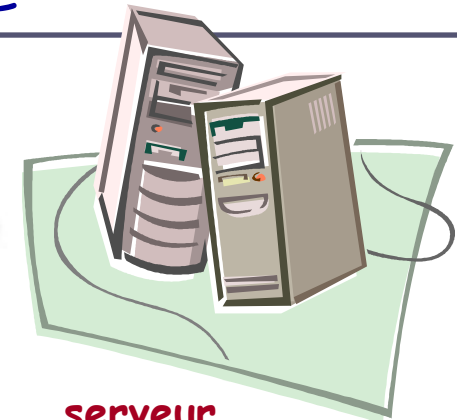
# Principes de base



- HTTP ne sait transporter que du texte (les pages HTML)
- Les échanges (requêtes et réponses) entre le client et le serveur sont donc au format texte
- Le format texte (représentation de base portable sur toute plateforme, au codage de caractères près) de représentation des informations est XML
- Les messages transportés sont donc au format XML



client

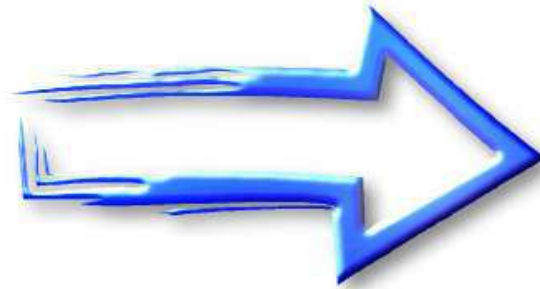


serveur

# Principes de base



- Les services web matérialise deux perceptions dans l'évolution des architectures.
  - ▶ *Passage du web client au web machines*
  - ▶ Adaptation des architectures distribuées au monde Web



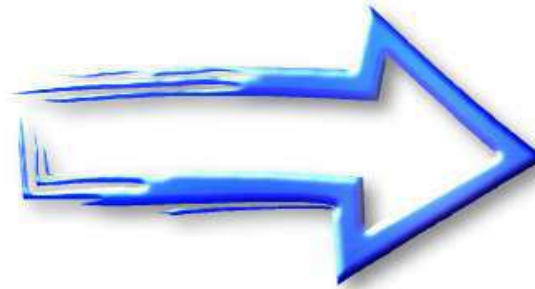
**Web Utilisateur caractérisé par :**  
**Navigateur HTML**  
**Contenu dynamique sur le serveur**

**Web machine caractérisé par :**  
**Communication directe**  
**entre applications**

# Principes de base



- Les services web matérialise deux perceptions dans l'évolution des architectures.
  - ▶ Passage du web client au web machines
  - ▶ *Adaptation des architectures distribuées au monde Web*



Architectures distribuées, à savoir :  
RPC, RMI  
Corba, DCOM, JRMP

Nouveau contexte, à savoir :  
Web (HTTP)  
XML



*Les services web : première approche*

*Le langage XML*

*Les services web : vue d'ensemble*



*Limites de HTML*

*Objectifs de XML*

*Les domaines de noms - Namespaces XML*

*Les schémas XML*

# Limites de HTML



- En fait HTML est très bien adapté à la création de sites simples. La facilité d'apprentissage de HTML a grandement participé à son développement auprès d'un public plus préoccupé par la forme que par le fond. Cependant, la demande nécessitant de plus en plus d'interactivité et de dynamisme, HTML a vite affiché des limites difficilement acceptables, principalement :
  - ▶ un balisage prédéfini dans la norme (n'autorisant pas la construction de ses propres balises et donc de ses propres outils),
  - ▶ Une gestion des liens trop simples (juste vers une autre page ou partie)
  - ▶ par un jeu de balisage dédié à la mise en forme, une dépendance très forte du contenu et de la forme, ce qui a pour conséquences :
    - une interopérabilité limitée (échange du contenu très difficile entre applications, cf. point précédent)
    - des recherches lourdes et des réponses très nombreuses (pas de notion sémantique dans la définition des contenus)

# Limites de HTML



➤ Pour illustrer certaines des limites de HTML, voici un exemple :



```
<H2>Bibliographie XML</H2>
  <UL>
<LI> Jean-Christophe Bernadac et François Knab, <I>Construire une application
XML</I>, Paris, Eyrolles, 1999</LI>
  <LI> Alain Michard, <I>XML, Langage et Applications</I>, Paris, Eyrolles,
1998 </LI>
  <LI> William J. Pardi, <I>XML en Action</I>, Paris, Microsoft Press, 1999,
adapté de l'anglais par James Guerin</LI>
</UL>
```

# Limites de HTML



- Nous constatons dans cet exemple que les balises HTML fournissent des indications qui vont permettre au butineur de mettre en page (ou en écran) le texte qu'elles encadrent:
  - ▶ `<H2>` et `</H2>` signalent un titre de niveau 2,
  - ▶ `<UL>` et `</UL>` une liste non numérotée,
  - ▶ `<LI>` et `</LI>` un élément de cette liste,
  - ▶ `<I>` et `</I>` du texte en italique.

# Limites de HTML



- Si on effectue l'affichage du texte précédent dans un navigateur HTML, on obtient le résultat suivant:



## ➤ Bibliographie XML

- ▶ Jean-Christophe Bernadac et François Knab, *Construire une application XML*, Paris, Eyrolles, 1999
- ▶ Alain Michard, *XML, Langage et Applications*, Paris, Eyrolles, 1998
- ▶ William J. Pardi, *XML en Action*, Paris, Microsoft Press, 1999, adapté de l'anglais par James Guerin

# Limites de HTML



- Toutes les balises d'une page HTML sont donc relatives à sa présentation finale et à rien d'autre. Rien ne permet à un logiciel de connaître le sens (la *sémantique*) du texte.
- Autrement dit, dans notre exemple, il n'est pas simple de de savoir que Alain Michard est l'**auteur** d'un **livre** intitulé *XML, Langage et Applications*, qui est paru en 1998, dont l'**éditeur**, Eyrolles, est situé à Paris.

# Objectifs de XML



- L'objectif majeur de XML est d'étendre les fonctionnalités de HTML afin de faciliter les échanges d'informations (arbitrairement complexes depuis l'apparition des schémas).
- Pour cela, on retrouve dans XML une généralisation des idées contenues dans HTML et surtout dans SGML (Standard Generalized Markup Language) tout en étant plus simple que ce dernier.
- Les principales évolutions vis-à-vis de HTML sont :
  - ▶ la définition libre de nouvelles balises et de nouveaux attributs,
  - ▶ le support de structure complexe de documents,
  - ▶ la vérification de la structure d'un document par rapport à une grammaire type défini dans un document appelé DTD (Document Type Descriptor).
  - ▶ la séparation entre le contenu (document XML) et la mise en page (feuille de style CSS ou XSL).

*XML signifie eXtensible Markup Language*

*XML peut être défini comme étant un langage de description et d'échanges de données structurés*

# Objectifs de XML



- XML a été conçu pour représenter des documents (des données structurées) suivant les cinq grands principes suivants :
  - ▶ lisibilité par l'homme et la machine\*
  - ▶ définition non ambiguë du contenu d'un document
  - ▶ définition non ambiguë de la structure d'un document
  - ▶ séparation entre documents et relations entre documents
  - ▶ séparation entre structure du document et présentation du document

\* Contrairement aux applications qui gèrent des formats de données binaires, comme Word, Excel par exemple, XML ne manipule que du texte, en fait des suites de caractères. C'est ce principe élémentaire qui le rend portable, puisque toutes les plateformes comprennent sans exception ce format de représentation. Le texte a toujours été le format de communication entre les applications (IOR Corba sous forme Texte) ou entre programme et utilisateur (toString de Java ou Smalltalk pour la représentation externe sous forme de chaînes de caractères)



# Objectifs de XML



➤ Voici ce que donnerait la même page si elle était codée en XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<BIBLIO SUBJECT="XML">
  <BOOK ISBN="9782212090819" LANG="fr" SUBJECT="applications">
    <AUTHOR>
      <FIRSTNAME>Jean-Christophe</FIRSTNAME>
      <LASTNAME>Bernadac</LASTNAME>
    </AUTHOR>
    <AUTHOR>
      <FIRSTNAME>François</FIRSTNAME>
      <LASTNAME>Knab</LASTNAME>
    </AUTHOR>
    <TITLE>Construire une application XML</TITLE>
    <PUBLISHER>
      <NAME>Eyrolles</NAME>
      <PLACE>Paris</PLACE>
    </PUBLISHER>
    <DATEPUB>1999</DATEPUB>
  </BOOK>
```

# Objectifs de XML



➤ Voici ce que donnerait la même page si elle était codée en XML (suite)

```
<BOOK ISBN="9782212090529" LANG="fr" SUBJECT="général">
  <AUTHOR>
    <FIRSTNAME>Alain</FIRSTNAME>
    <LASTNAME>Michard</LASTNAME>
  </AUTHOR>
  <TITLE>XML, Langage et Applications</TITLE>
  <PUBLISHER>
    <NAME>Eyrolles</NAME>
    <PLACE>Paris</PLACE>
  </PUBLISHER>
  <DATEPUB>1998</DATEPUB>
</BOOK>
<BOOK ISBN="9782840825685" LANG="fr" SUBJECT="applications">
  <AUTHOR>
    <FIRSTNAME>William J.</FIRSTNAME>
    <LASTNAME>Pardi</LASTNAME>
  </AUTHOR>
  <TRANSLATOR PREFIX="adapté de l'anglais par">
    <FIRSTNAME>James</FIRSTNAME>
    <LASTNAME>Guerin</LASTNAME>
  </TRANSLATOR>
  <TITLE>XML en Action</TITLE>
  <PUBLISHER>
    <NAME>Microsoft Press</NAME>
    <PLACE>Paris</PLACE>
  </PUBLISHER>
  <DATEPUB>1999</DATEPUB>
</BOOK>
</BIBLIO>
```

# Objectifs de XML



- Ici aucune des balises ne décrit la présentation finale. On remarque, en revanche, que maintenant les balises ont un sens et une hiérarchie.
- Par exemple, l'élément "author" comprend le prénom (balise "firstname") et le nom (balise "lastname") de l'auteur.
- On remarque aussi que des informations supplémentaires (langue, sujet, n° ISBN), ont pu être ajoutées sous forme d'"attributs" situés à l'intérieur même des balises.

# Objectifs de XML



- Le même fichier XML affiché par Microsoft Internet Explorer 5 (en l'absence d'une feuille de style) sous forme d'arborescence dépliable et repliable (le signe "-" indique une branche dépliée, le signe "+" une branche repliée)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <BIBLIO SUBJECT="XML">
+ <BOOK ISBN="9782212090819" LANG="fr" SUBJECT="applications">
+ <BOOK ISBN="9782212090529" LANG="fr" SUBJECT="général">
- <BOOK ISBN="9782840825685" LANG="fr" SUBJECT="applications">
  - <AUTHOR>
    <FIRSTNAME>William J.</FIRSTNAME>
    <LASTNAME>Pardi</LASTNAME>
  </AUTHOR>
  - <TRANSLATOR PREFIX="adapté de l'anglais par">
    <FIRSTNAME>James</FIRSTNAME>
    <LASTNAME>Guerin</LASTNAME>
  </TRANSLATOR>
  <TITLE>XML en Action</TITLE>
  - <PUBLISHER>
    <NAME>Microsoft Press</NAME>
    <PLACE>Paris</PLACE>
  </PUBLISHER>
  <DATEPUB>1999</DATEPUB>
</BOOK>
</BIBLIO>
```

# *Les domaines de noms (namespaces) XML*

- L'interopérabilité de XML suppose qu'il soit possible que des documents contiennent des balises réutilisables constituant des vocabulaires, des grammaires de différentes spécialités (mathématiques, biologie, ...).
- Il est donc courant de mettre en oeuvre plusieurs DTD (provenant de plusieurs domaines de spécialité) pour construire un document XML.
- Des conflits de noms peuvent alors survenir si deux éléments portent des noms identiques dans deux DTD différentes (par exemple adresse peut représenter une adresse d'expédition ou une adresse mail).
- Les domaines de noms (XML namespaces) sont une recommandation du W3C (rapidement adoptée après XML 1.0) pour résoudre le problème de conflits de noms dans un document XML. Le principe consiste à préfixer chaque nom d'élément par un nom « unique » qui identifie le domaine auquel il fait référence.

# *Les domaines de noms (namespaces) XML*

## ➤ Les avantages de cette structuration :

- ▶ Les espaces de noms permettent de **lever les ambiguïtés** sur des éléments de DTD différentes qui porteraient le même nom (nom de produit, nom de personne, ..)
- ▶ Les espaces de noms favorisent **la modularité** des documents XML et les sources multiples de ces derniers
- ▶ Les espaces de nom permettent, indirectement, de **rendre plus lisible** un document XML contenant des informations de sources diverses

# Les domaines de noms (namespaces) XML

- Les Namespaces ont un statut de recommandation du 14 Janvier 1999, dont les spécifications sont disponibles à :
  - ▶ <http://www.w3.org/TR/1999/REC-xml-names-19990114>
- Un espace de noms est déclaré à l'aide de l'attribut `xmlns`:
  - ▶ Soit en déclarant l'espace de nom dans l'élément
    - `<ElementDuDocument xmlns='UriDTDaImporter'>`
    - Exemple d'utilisation sans préfixe  
`<seminaire xmlns="http://www.unilog.fr/seminaire">`
  - ▶ Soit en associant un préfixe pour une utilisation plus fin
    - `<ElementDuDocument xmlns:Préfixe='UriDTDaImporter'>`
    - Exemple d'utilisation avec préfixe  
`<seminaire xmlns:sem="http://www.unilog.fr/seminaire">`  
....  
`<sem:date/>`

*Pour assurer l'unicité des domaines, on a recours aux URI (Uniform Ressourc Identifier), le moyen standard pour accéder à des ressources internet.*

# *Les schémas XML*

- XML, originellement conçu pour la publication de documents, a connu une forte mutation pour permettre l'échange de données quelconques.
- La DTD, le langage de schéma actuel, ne définit que la structure du documents XML
- XML schema a pour vocation de décrire tout type de structure de données, depuis les modèles des bases de données relationnelles au modèle objet des langages à objets comme Java, en passant par la description des structures des documents XML



# Les schémas XML

- Le modèle des DTD présentent les faiblesses suivantes :
  - ▶ Les DTD ne sont pas écrites en XML, ce qui signifie que les technologies existantes pour manipuler des documents XML telles que DOM ou SAX ne peuvent être utilisées pour « parser » des schémas de documents.
  - ▶ Les DTD ne supportent pas les espaces de nom ce qui rend impossible l'import de schémas externes afin de réutiliser du code existant.
  - ▶ Les DTD n'offrent qu'un typage très limité des données.

# Les schémas XML

- Conscient de ces fâcheuses limitations, le W3C a proposé un nouveau langage de définition de schéma, XML Schema.
- Conçu pour pallier les faiblesses des DTD, XML Schema propose, en plus des fonctionnalités fournies par les DTD, des nouveautés
  - ▶ Un grand nombre de types de données intégrées comme les booléens, les entiers, les intervalles de temps, etc. De plus, il est possible de créer de nouveaux types par ajout de contraintes sur un type existant.
  - ▶ Des types de données utilisateurs qui vous permettent de créer votre propre type de données nommé.
  - ▶ La notion d'héritage. Les éléments peuvent hériter du contenu et des attributs d'un autre élément. C'est sans aucun doute l'innovation la plus intéressante de XML Schema.
  - ▶ Le support des espaces de nom.
  - ▶ Les indicateurs d'occurrences des éléments peuvent être tout nombre non négatif.
  - ▶ Une grande facilité de conception modulaire de schémas

# *Les schémas XML*

- XML Schema a un statut de recommandation du 2 Mai 2001, dont les spécifications sont disponibles à :
  - ▶ <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/> (Part 0: Primer)
  - ▶ <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/> (Part 1: Structures)
  - ▶ <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/> (Part 2: Datatypes)

# Les schémas XML

➤ Exemple de DTD pour représenter un document « Livre »



```
<!ELEMENT Livre (Titre, Auteur, Date, ISBN, Editeur)>  
<!ELEMENT Titre (#PCDATA)>  
<!ELEMENT Auteur (#PCDATA)>  
<!ELEMENT Date (#PCDATA)>  
<!ELEMENT ISBN (#PCDATA)>  
<!ELEMENT Editeur (#PCDATA)>
```

# Les schémas XML

## ➤ Schéma associé

```
<xsd:element name="Livre">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Titre" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="Auteur" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="Editeur" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Titre" type="xsd:string"/>
<xsd:element name="Auteur" type="xsd:string"/>
<xsd:element name="Date" type="xsd:string"/>
<xsd:element name="ISBN" type="xsd:string"/>
<xsd:element name="Editeur" type="xsd:string"/>
```

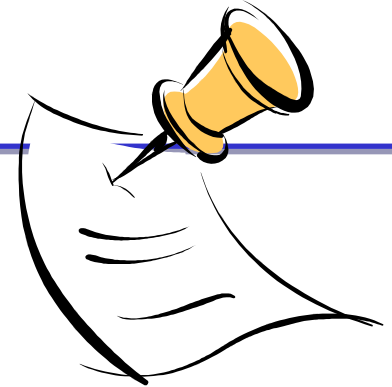
# Les schémas XML

- Les schemas XML permettent donc de représenter des méta-données pour décrire des structures arbitrairement complexes
- Les parseurs basés sur les schemas XML sont désormais disponibles, allégeant ainsi le travail des applications manipulant les documents
- L'accès à la description complète des structures de données favorise l'automatisation des échanges entre applications
- Par exemple, les schemas XML sont fortement utilisés :
  - ▶ Dans les gestionnaires de bases relationnelles pour importer ou exporter les méta-données
  - ▶ Dans les serveurs d'applications pour décrire précisément les déploiements, les configurations des composants

*Les services web : première approche*

*Le langage XML*

*Les services web : vue d'ensemble*



*Les services web, c'est quoi ?*

*Les services web, pourquoi*

*Les usages*

*Les acteurs (rôles)*

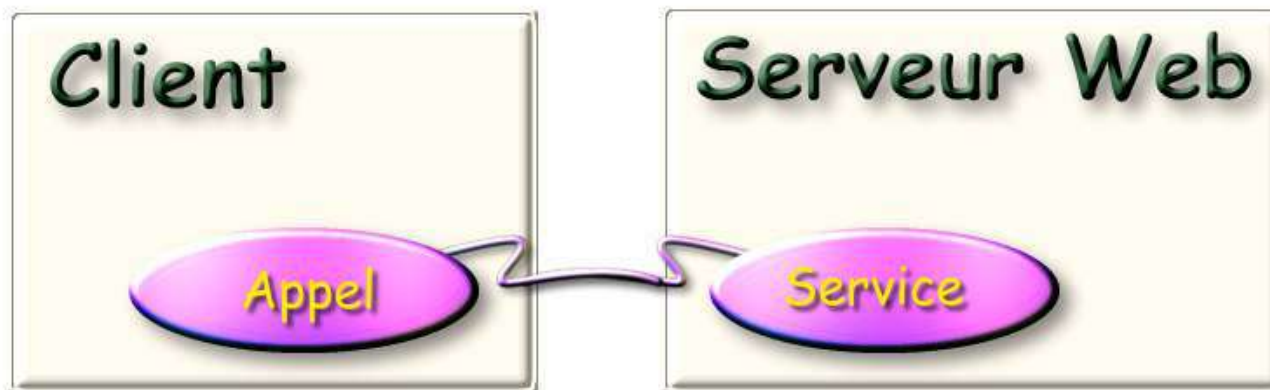
*Le scénario complet*

*Les technologies*



# *Les Services Web, c'est quoi ?*

- Avant toute autre chose, la technologie des services web affiche les mêmes intentions que les architectures les plus anciennes en terme d'accès distant, comme les moniteurs TP.
- C'est la possibilité d'invoquer une fonction distante. En l'occurrence, sur un serveur web distant puisque le protocole de base est HTTP
- On dispose d'une infrastructure souple basée sur XML pour les systèmes distribués hétérogènes





# *Les Services Web, c'est quoi ?*

- Ils sont accessibles via le web par des protocoles bien connus
- Ils sont décrits à partir de XML
- Ils interagissent via XML
- Ils sont localisables à partir de registres
- Ils sont entièrement transversaux aux plates-formes et très faiblement couplés

# *Les Services Web, c'est quoi ?*

- Ils introduisent un nouveau modèle de développement basé sur ce que l'on appelle les architectures orientées services
- Une architecture orientée services se focalise sur une décomposition plus abstraite dans la résolution des problèmes. On parle de résolution dirigée par les services.
- Un service résout un problème donné
- Les services peuvent être combinés pour résoudre des problèmes de plus en plus complexes

# *Les Services Web, c'est quoi ?*

- Les tâches associées à la manipulation des services web sont :
  - ▶ Interroger un annuaire : qui fournit des choses dont on ne connaît pas forcément la nature, le rôle ou le contenu
  - ▶ Négocier avec les fournisseurs potentiels de ces choses pour connaître :
    - Nature exacte du service fourni
    - Qualité/coût/etc.
  - ▶ Interagir avec le service du fournisseur choisi pour :
    - Connaître les modalités d'interaction
    - Introduire le service dans ma chaîne de traitements
  - ▶ Eventuellement composer des services
  - ▶ Eventuellement publier mes propres services

# *Les Services Web, c'est quoi ?*

- L'avantage essentiel des services web concerne le fait que le client consommateur n'a pas besoin de connaître l'identité du fournisseur du service
- Le client doit simplement exprimer son besoin
- Face à un besoin, plusieurs fournisseurs de services peuvent exister
- Chacun ayant des caractéristiques de coût, de performance, de fiabilité, etc.
- Le client choisit le fournisseur (i.e. le service) correspondant le mieux à ses besoins.

# *Les Services Web, pourquoi ?*

- Lorsque l'on a besoin d'interopérabilité dans des environnements applicatifs distribués
  - ▶ Les services peuvent communiquer entre eux, et cela depuis des environnements applicatifs distants
- Lorsque l'on veut accéder aux applications à travers les pare-feux
  - ▶ Les services web sont définis et accédés avec XML sur des protocoles standards comme HTTP et SMTP. Ils peuvent alors être invoqués à travers un pare-feux.
- Lorsque l'on veut profiter de différents environnements et langages de développement
  - ▶ Par une description et une invocation XML, les services web sont très flexibles et indépendants des langages et des systèmes.

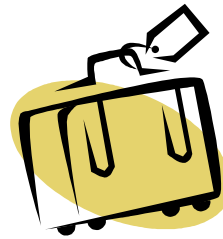
# Les usages

- Les services web pour représenter des applications sophistiquées bien délimitées et sans forte interactivité.
  - ▶ Par exemple, une application qui donne les conditions du temps peut être idéalement représentée par un service.
- Les services web sont adaptés pour l'assemblage de composants faiblement couplés.
  - ▶ Ils sont définis de façon indépendante, mais peuvent interagir.
- Les services web sont adaptés à la représentation d'applications orientées messages.
  - ▶ Les mécanismes d'invocation asynchrone des applications orientées messages sont en font de bonnes candidates aux services web

# Les usages

➤ Les services web servent de base à l'intégration d'applications faiblement couplées

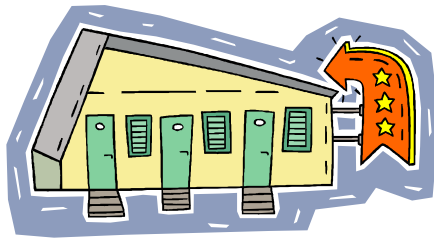
Service Profil Client



Application Web



Client



Service réservation hôtel



Service réservation voiture



Service réservation billet

# *Les acteurs (rôles)*

- Les principaux acteurs dans la technologie des services web sont :
  - ▶ Le client : celui qui utilise, invoque le service web
  - ▶ Le fournisseur : celui qui fournit le service web
  - ▶ L'annuaire : celui qui détient les informations du service web



# *Les acteurs (rôles)*

- Le client et le fournisseur sont les éléments principaux dans l'architecture des services web
- Un fournisseur est représenté par un serveur d'application (J2EE par exemple)
- Le fournisseur détient un ou plusieurs services qui sont représentés par des EJBs ou des servlets et qui sont enveloppés d'une couche « service »
- Le fournisseur peut être le client d'un autre fournisseur (interopérabilité)
- Une fois le service définit, il peut être déclaré dans un annuaire, on parle alors de publication du service afin de le rendre accessible aux clients

# *Le scenario complet*

- **Etape 1 : définition, description du service**
  - ▶ On doit décrire d'un point de vue informatique ce que fait le service, la solution qu'il propose, ...
  - ▶ La définition est faite au WSDL au sein du fournisseur de services (i.e. le serveur d'applications)
  
- **Etape 2 : publication du service**
  - ▶ Une fois le service défini et décrit en termes de mise en oeuvre, il peut être déclaré dans un annuaire, on parle alors de publication du service afin de le rendre accessible aux clients.
  - ▶ Comme on le verra, la publication sera effectuée au sein d'un annuaire dédié UDDI.
  
- **Etape 3 : recherche du service**
  - ▶ Le client se connecte, sur un annuaire (UDDI) pour effectuer une recherche de service.

# *Le scenario complet*

- **Etape 4 : enregistrement au service web**
  - ▶ Une fois le service trouvé par le client, ce dernier doit s'enregistrer auprès du fournisseur associé au service. Cet enregistrement indique au fournisseur l'intention du client d'utiliser le service suivant les conditions décrites dans la publication.
  
- **Etape 5 : mise en oeuvre du service**
  - ▶ Le client peut invoquer le service suivant les conditions inscrites au sein de l'annuaire lors de la publication du service web (étape 2)
  
- **Etape 6 : composition**
  - ▶ C'est la possibilité de combiner plusieurs services. En fait, un service peut devenir le client d'un autre service.

# Le scenario complet

## Application cliente

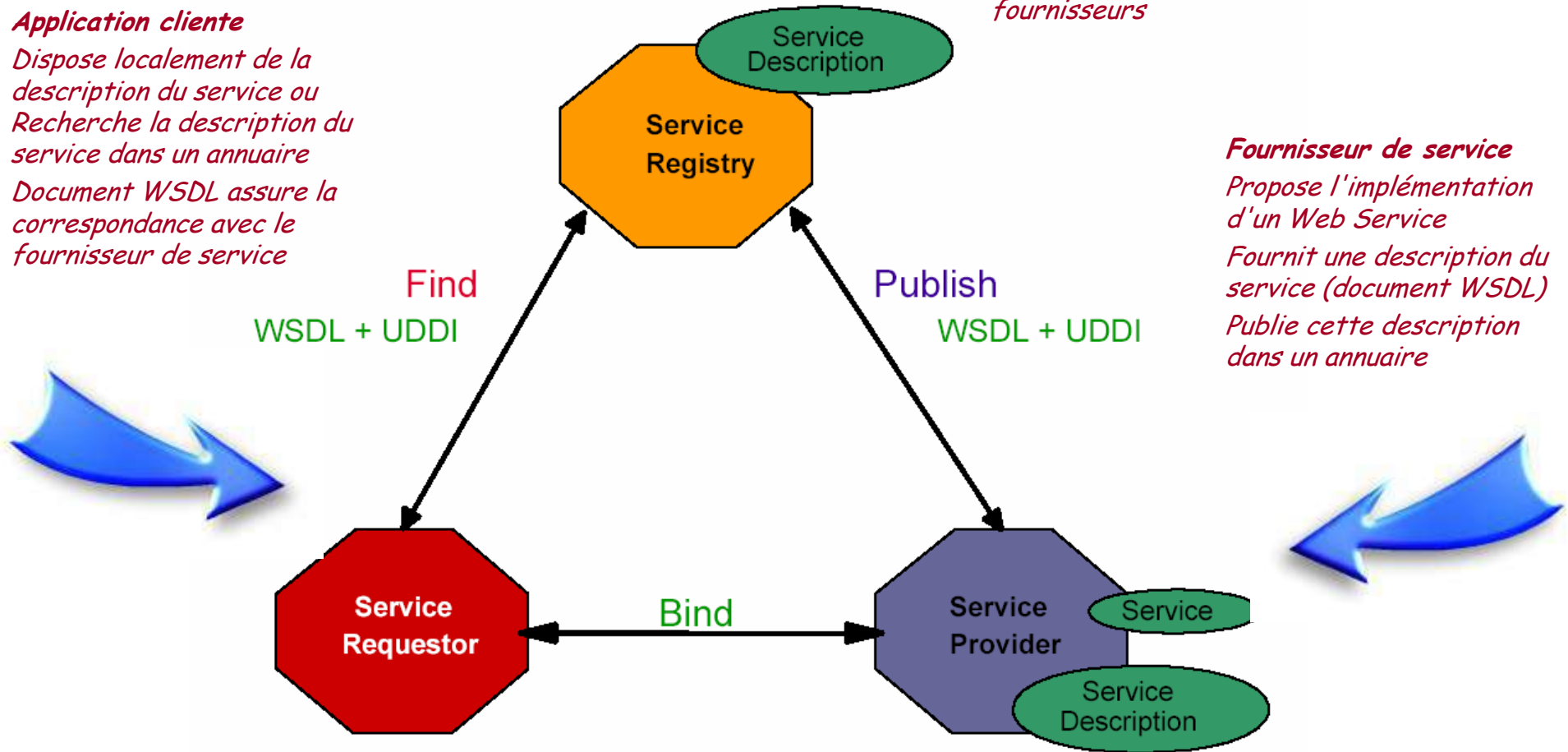
Dispose localement de la description du service ou Recherche la description du service dans un annuaire  
Document WSDL assure la correspondance avec le fournisseur de service

## Annuaire de publication

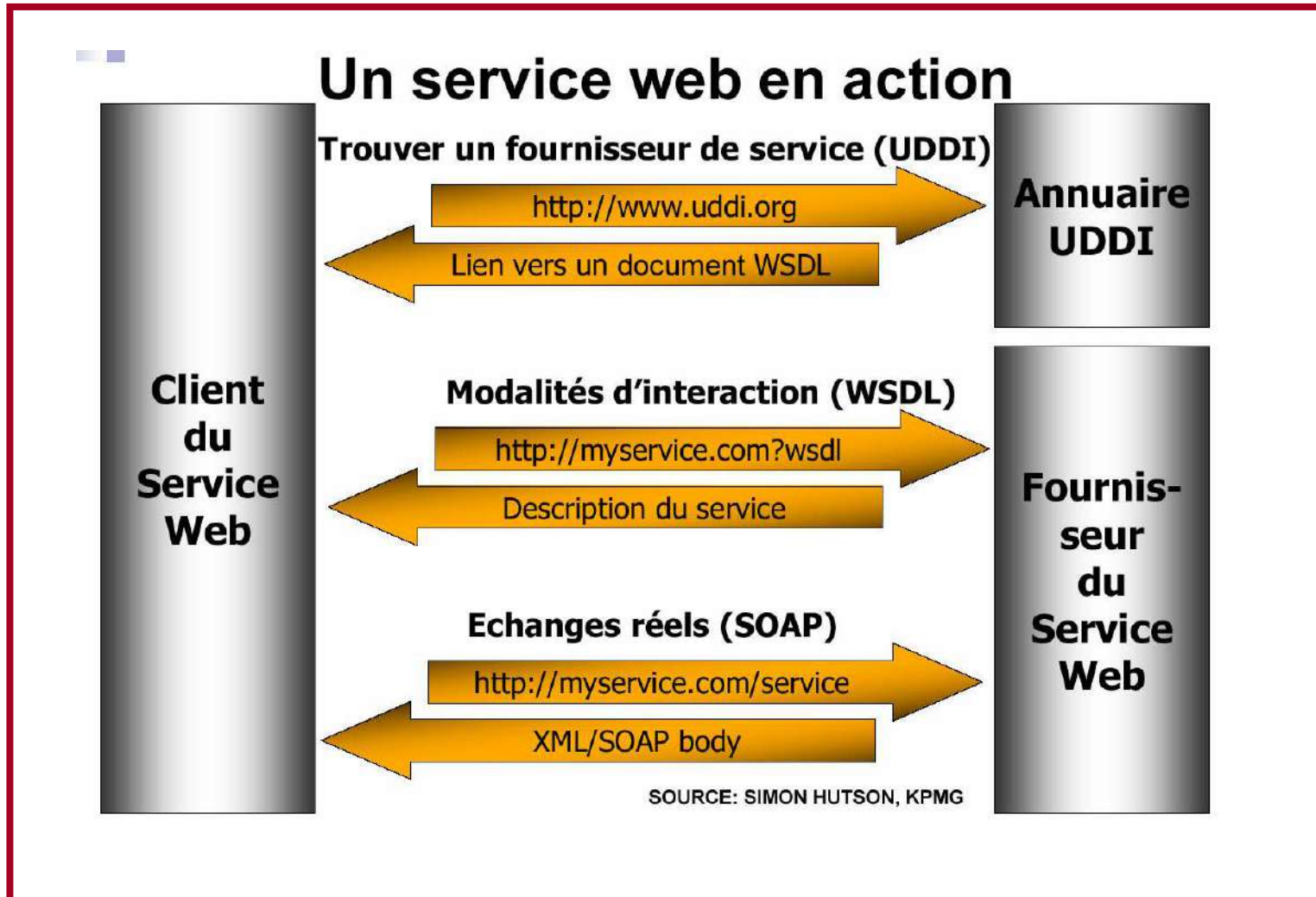
Dispose de la description des services ainsi que de leur localisation, c'est-à-dire des différents fournisseurs

## Fournisseur de service

Propose l'implémentation d'un Web Service  
Fournit une description du service (document WSDL)  
Publie cette description dans un annuaire

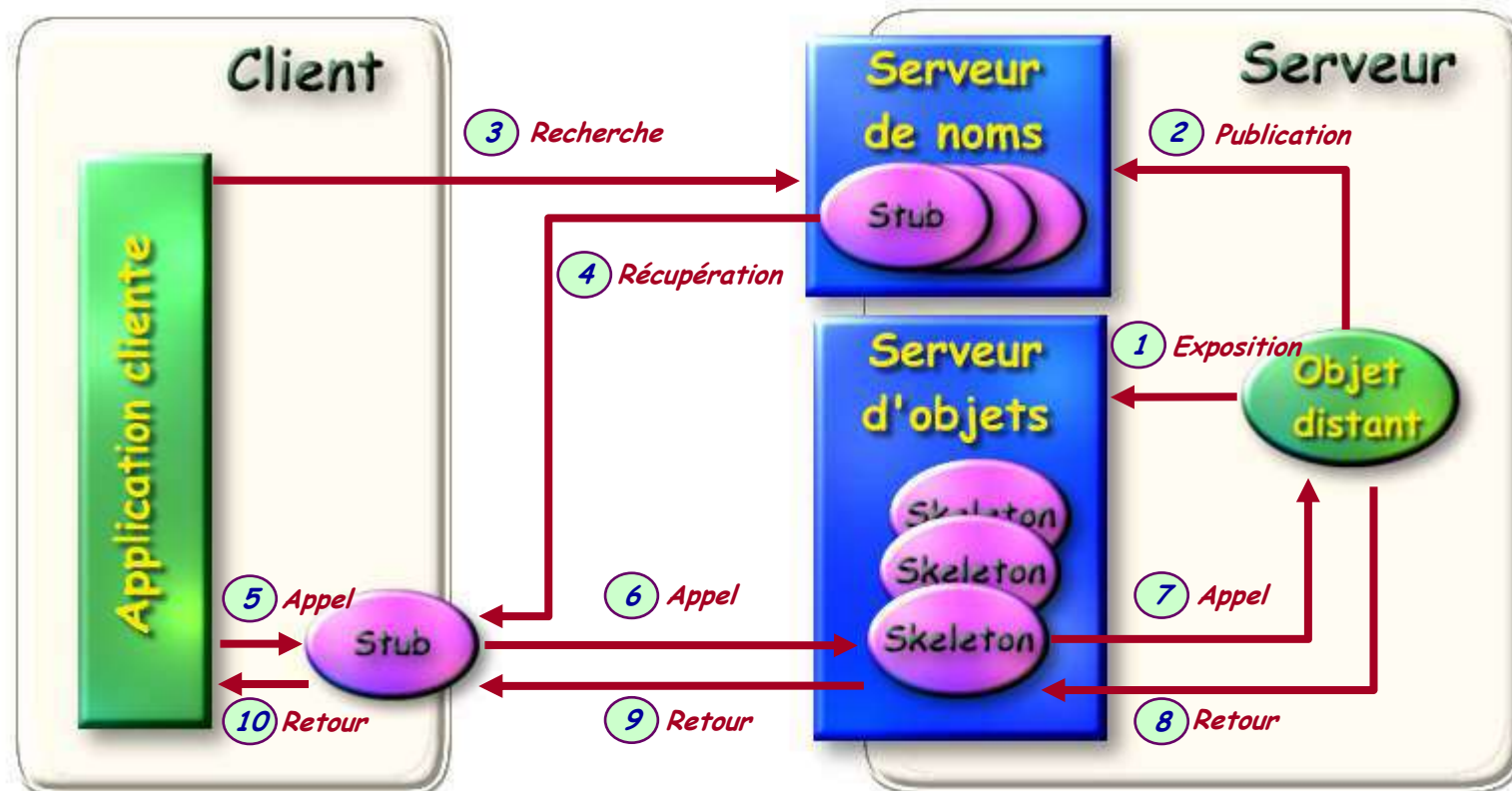


# Le scenario complet



# Une architecture déjà connue

- Les architectures Corba, EJB ou RMI manipulent ces concepts depuis fort longtemps



# Les technologies

- Les éléments techniques utilisés sont différents puisque imposés par le Web et XML
- L'architecture des Web Services repose essentiellement sur les technologies suivantes :
  - ▶ **SOAP - Simple Object Access Protocol**  
Protocole pour la communication entre Web Services  
IIOP pour Corba ou RMI-IIOP pour les EJBs
  - ▶ **WSDL - Web Service Description Language**  
Langage de description de l'interface du Web Service  
IDL pour Corba ou Interface Java pour les EJBs
  - ▶ **UDDI - Universal Description, Discovery and Integration**  
Annuaire pour le référencement du Web Service  
CosNaming pour Corba ou JNDI pour les EJBs

# *Les technologies*

- Un Web Service est une application déployée sur un serveur Web (serveur d'objets).
- Supporter des Web Services apporte une interopérabilité certaine et une grande flexibilité puisqu'il s'agit de coopération entre objets distants par le biais du Web (TCP/IP - HTTP).



# Les technologies

## Pile de protocole

Document XML décrivant le service afin de rendre la solution des Web Services générique	<b>WSDL</b>
Protocole basé sur le standard XML pour l'échange de données structurées entre des applications réseaux	<b>SOAP</b>
Couche réseau (HyperText Transfer Protocol)	<b>HTTP</b>

Couches de base permettant l'interopérabilité des Web Services

## *Les technologies*

- Afin d'être découvert, un service doit être publié. Au dessus de ces trois couches de base viennent se greffer deux couches UDDI :

Découverte de services	UDDI
Publication de services	UDDI

- On publie notre service via le document WSDL auprès de notre annuaire UDDI.
- Une application cliente peut découvrir et accéder à notre service lors de son exécution via un annuaire UDDI.

# *Le protocole SOAP*

## ➤ Rôle

- ▶ Assure les appels de procédures à distance au dessus d'un protocole de transport

## ➤ Fonctionnement côté Client

- ▶ Ouverture d'une connexion HTTP
- ▶ Requête SOAP est un document XML décrivant :
  - une méthode à invoquer sur une machine distante
  - les paramètres de la méthode

## ➤ Fonctionnement côté Serveur SOAP

- ▶ Récupère la requête
- ▶ Exécute la méthode avec les paramètres
- ▶ Renvoie une réponse SOAP (document XML) au client

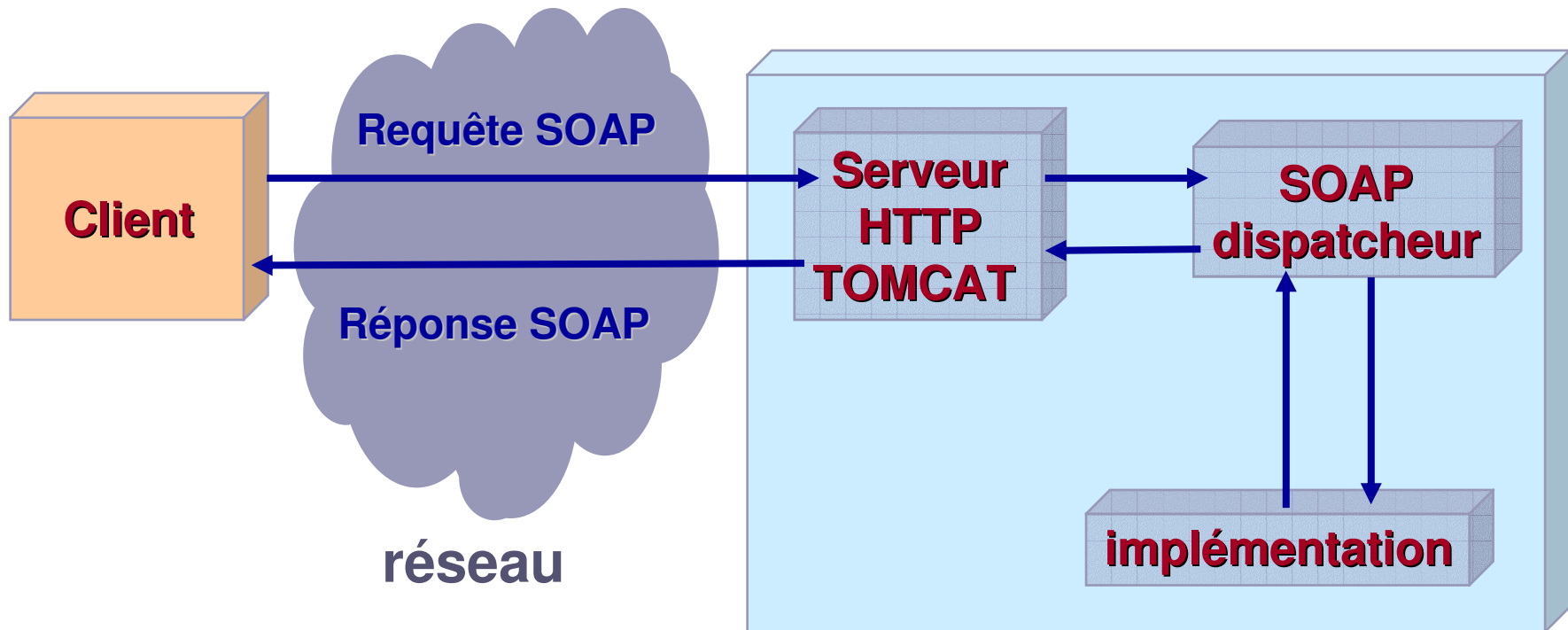
# Le protocole SOAP

## Service Requestor

*Demandeur de service*

## Service Provider

*Fournisseur de service*



# *Le langage WSDL*

- Une interface qui cache le détail de l'implémentation du service, permettant une utilisation indépendante :
  - ▶ de la plate-forme utilisée
  - ▶ du langage utilisé
  
- Le fichier WSDL est au format XML et regroupe toutes les informations nécessaires pour interagir avec le Web Service :
  - ▶ les méthodes
  - ▶ les paramètres et valeurs retournées
  - ▶ le protocole de transport utilisé
  - ▶ la localisation du service

*Documents WSDL, générés par les outils de développement favorisent une intégration rapide du service*

# Le langage WSDL

- 2 types de documents WSDL :
  - ▶ le document WSDL décrivant l'interface du service
  - ▶ le document WSDL décrivant l'implémentation du service
- Documents indispensables au déploiement de Web Services
- Publication et recherche de services au sein de l'annuaire UDDI se font via ces 2 types de document WSDL

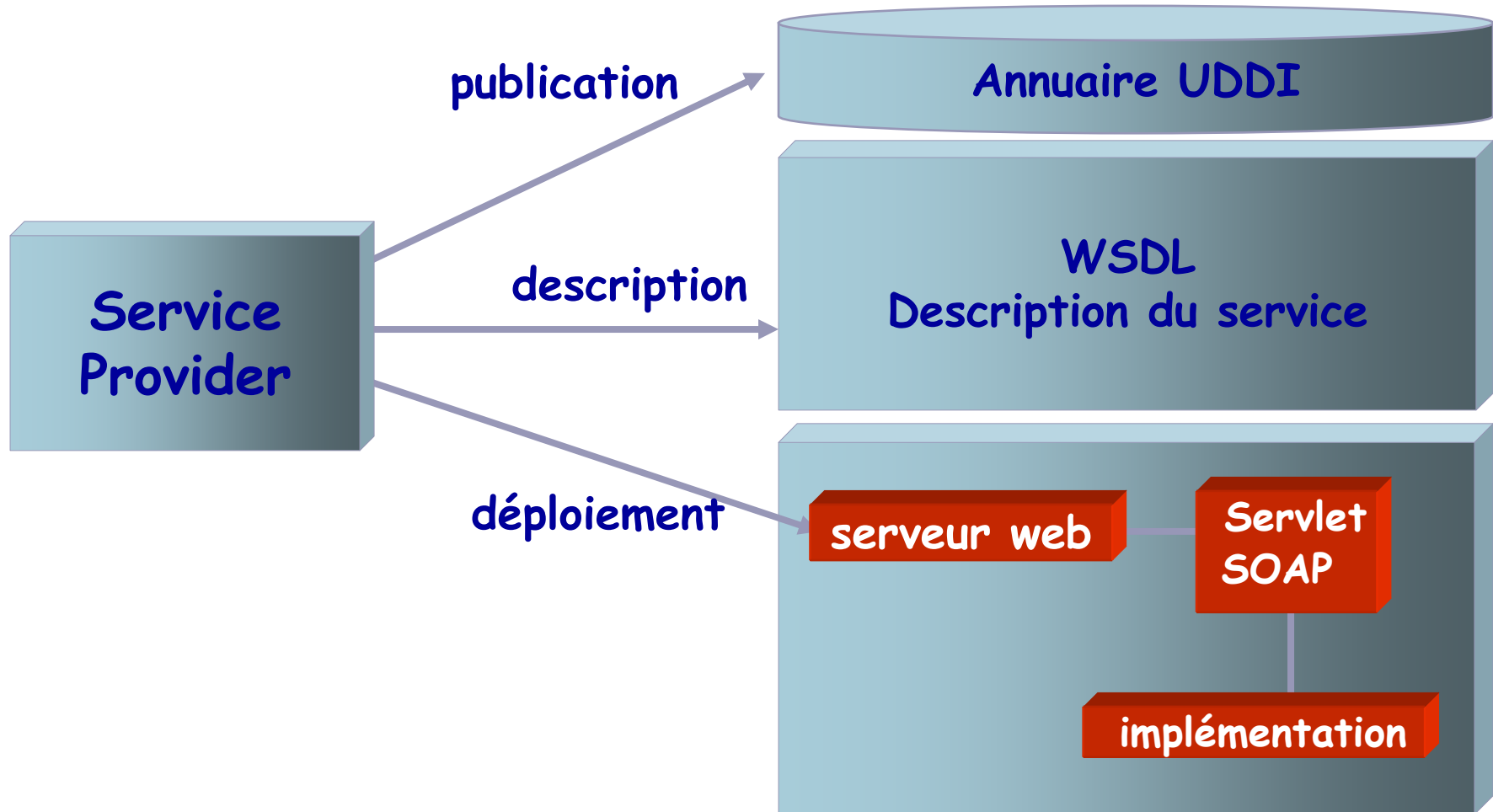
*Pour l'accès à un service particulier, un client se voit retourné l'URL du fichier WSDL décrivant l'implémentation du service. Seul l'emplacement de ce fichier WSDL est indiqué puisque ce dernier référence l'autre document WSDL décrivant l'interface de mise en œuvre du service.*

# Annuaire UDDI

## ➤ Rôle

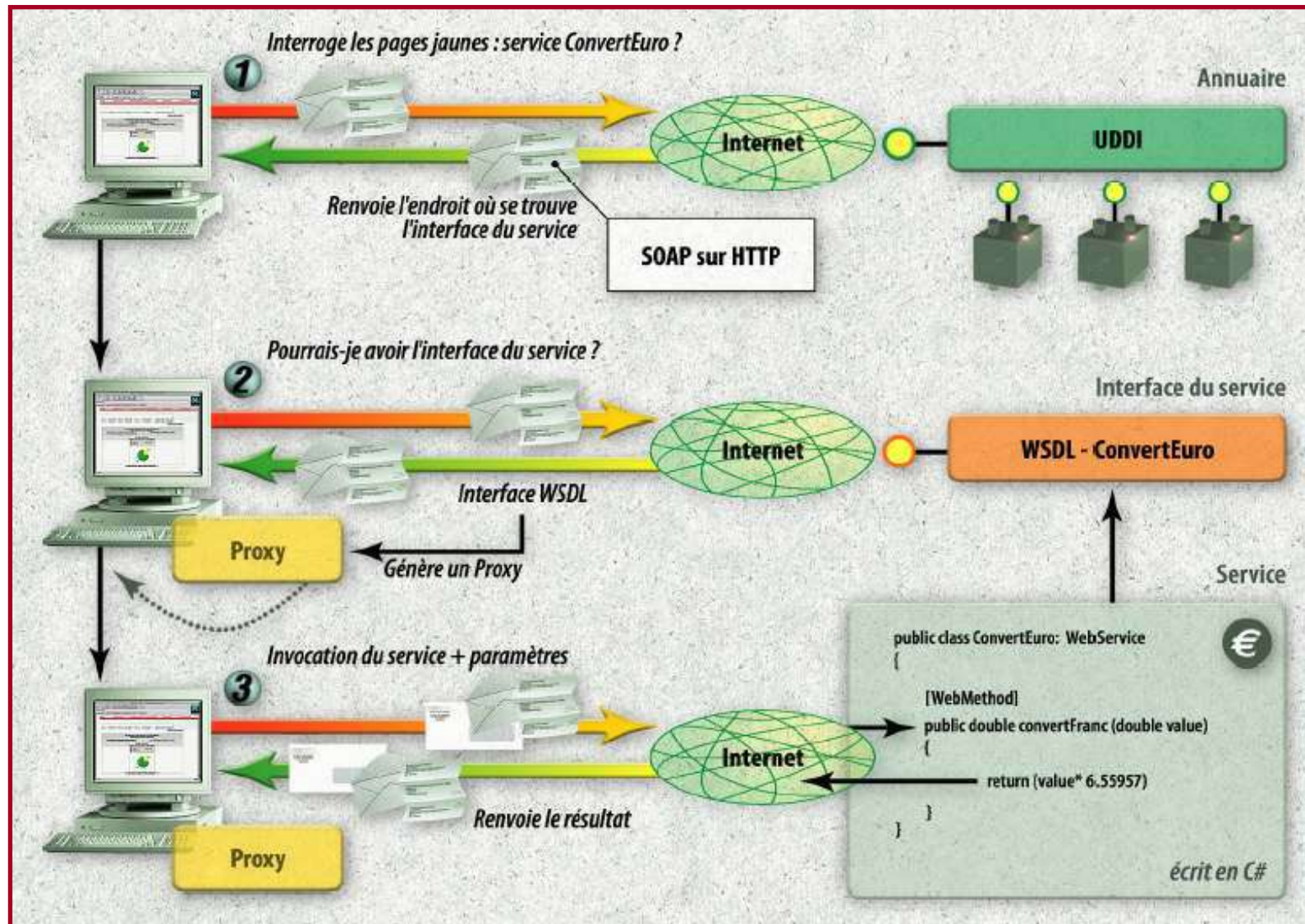
- ▶ Spécification pour la définition d'un service de registre
- ▶ Fournisseur
  - › Déclaration du fournisseur
  - › Enregistrement de ses Web Services disponibles
- ▶ Client
  - › Requête de recherche de Web Services (SOAP)
  - › Mise en relation avec le Web Service d'un fournisseur

# *Le fournisseur de services*





# Schéma fonctionnel



*Le protocole SOAP*  
*Le langage WSDL*  
*L'annuaire UDDI*

# Services Web

# Technologies associées

*Le protocole SOAP*

*Le langage WSDL*

*L'annuaire UDDI*



*SOAP, c'est quoi ?*

*Les principes*

*Historique*

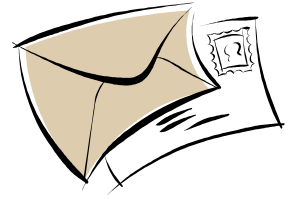
*Les messages SOAP*

*Architecture technique côté client*

*Architecture technique côté serveur*

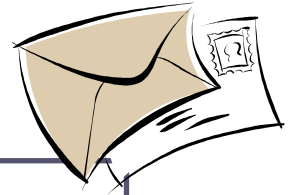


# *SOAP, c'est quoi ?*



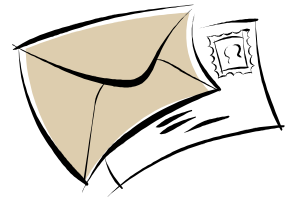
- SOAP est un protocole de transmission de messages.
- Il définit un ensemble de règles pour structurer des messages principalement pour exécuter des dialogues requête-réponse de type RPC (Remote Procedure Call).
- Il n'est pas lié à un protocole particulier mais HTTP est populaire.
- Il n'est pas non plus lié à un système d'exploitation ni à un langage de programmation, donc, théoriquement, les clients et serveurs de ces dialogues peuvent tourner sur n'importe quelle plate-forme et être écrits dans n'importe quel langage du moment qu'ils puissent formuler et comprendre des messages SOAP.

# SOAP, c'est quoi ?



- Pour comprendre, imaginons une base de données très simple d'une entreprise, comprenant une table spécifiant :
  - ▶ le numéro de référence,
  - ▶ le nom,
  - ▶ le numéro de téléphone des employés.
  
- On désire mettre en place un service qui permet à d'autres systèmes de la compagnie de consulter ces données.
  
- Le service retourne un nom et un numéro de téléphone (un tableau de chaînes de caractères à deux éléments) pour un numéro de référence d'employé donné (un entier).
  
- Voici la signature Java de ce service :
  - ▶ `String[] getEmployeeDetails ( int employeeNumber );`

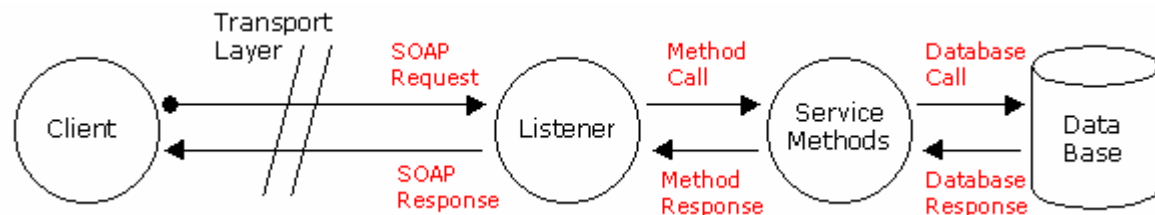
# SOAP, c'est quoi ?



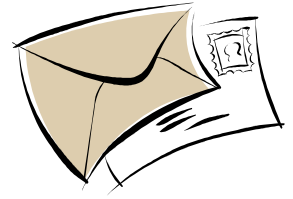
## ➤ L'approche SOAP consiste :

- ▶ à encapsuler la logique de requête de la base de données du service (i.e. implémentation), dans une méthode Java par exemple
- ▶ puis démarrer un thread qui écoute les requêtes adressées à ce service (un écouteur ou listener), ces requêtes étant formulées dans un format SOAP et contenant le nom du service et les paramètres requis.
- ▶ Le listener décode la requête SOAP entrante et la transforme en un appel de la méthode vers l'objet concerné.
- ▶ Il récupère le résultat de l'appel de la méthode, l'encode dans un message SOAP (la réponse) et le renvoie au demandeur.

## ➤ Cela donne:

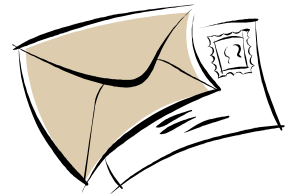


# SOAP, c'est quoi ?

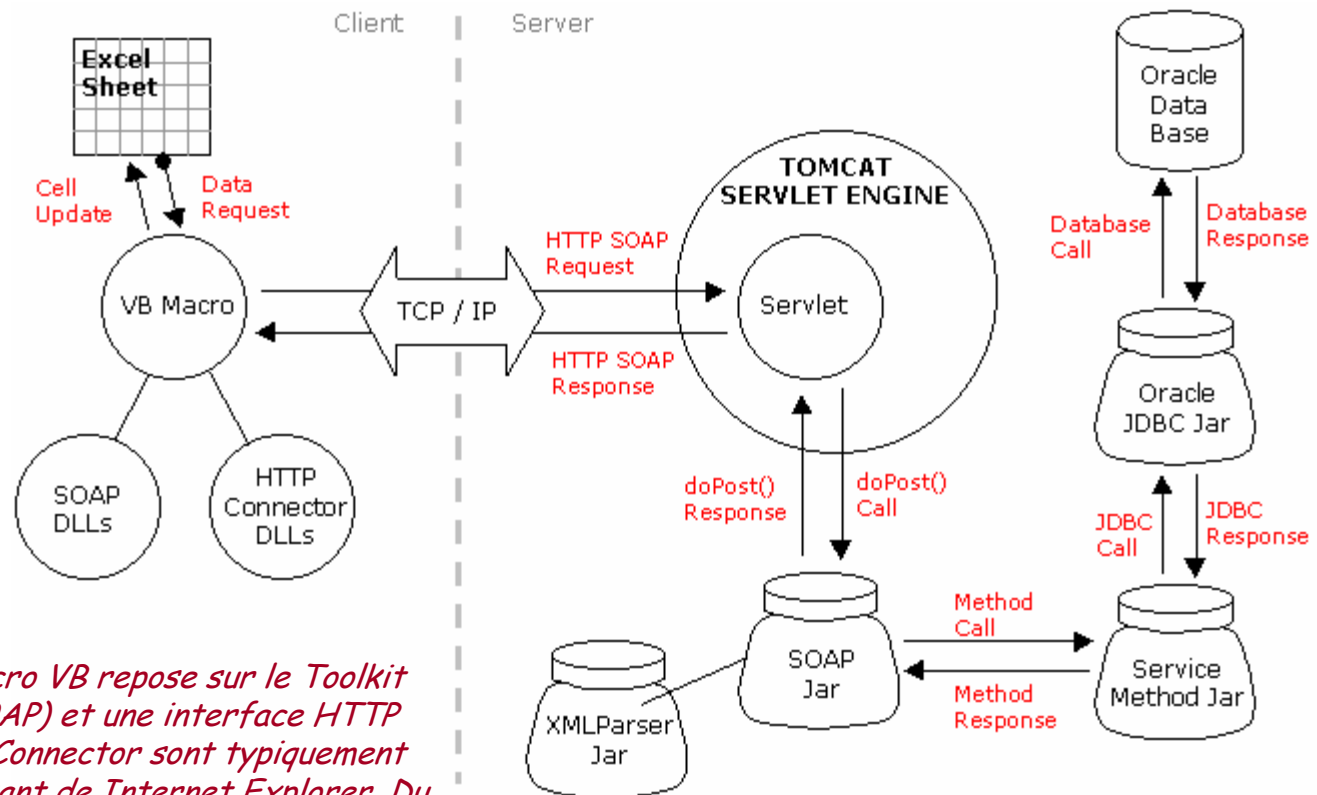


- L'architecture technique permettant cette mise en œuvre peut-être par exemple :
  - ▶ Une base de données « Oracle ».
  - ▶ La méthode du service est écrite en Java et se connecte à la base de données via un pilote « JDBC » pour Oracle.
  - ▶ Le listener est une Servlet Java tournant dans un moteur de servlet comme Tomcat.
  - ▶ La servlet a accès à des classes Java capables d'encoder et de décoder des messages SOAP (tel que Apache SOAP for Java) et écoute ces messages en tant que HTTP POST. Le transport est HTTP par TCP/IP. Le client est un tableau excel. Il utilise une macro VB qui, à son tour, exploite le Toolkit Microsoft SOAP pour encoder une requête SOAP et décoder la réponse reçue.

# SOAP, c'est quoi ?



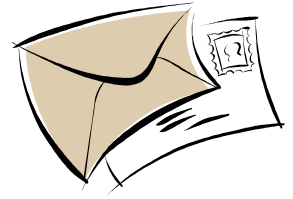
➤ L'architecture technique est illustrée par la figure suivante :



*Notez que du côté client, la macro VB repose sur le Toolkit Microsoft SOAP (les DLL's SOAP) et une interface HTTP Connector. De tels DLLs HTTP Connector sont typiquement déjà installés en tant que composant de Internet Explorer. Du côté serveur, à noter que le package SOAP repose sur un Parser XML qui analyse les messages SOAP. Dans le cas de SOAP Apache pour Java, ce sera Xerces.*

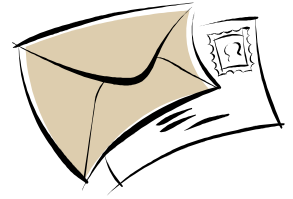


# SOAP, c'est quoi ?



- Les intérêts d'une telle architecture sont multiples vis-à-vis d'un modèle client/serveur de données avec procédures stockées par exemple :
  - ▶ L'envoi ou la réception de données structurées et arbitrairement complexes
  - ▶ Séparation des définitions métiers, des traitements, des données accédées
  - ▶ L'utilisation de sources de données Java « DataSources » (ce sont des pilotes RMI, middleware d'accès aux données distantes) impose l'utilisation du langage Java

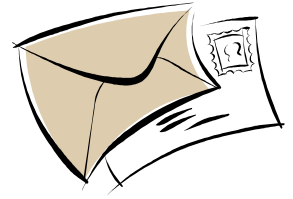
# SOAP, c'est quoi ?



- CORBA autorise, comme SOAP d'avoir :
  - ▶ des types complexes de données
  - ▶ des clients et serveurs dans tous langages et plate-formes
  - ▶ une séparation de la couche métier
  
- SOAP est plus adapté à la notion de services « web », car :
  - ▶ Corba exige de compiler et distribuer des stubs client pour chaque type de clients.
  - ▶ L'accès aux objets distants Corba par IIOP (le protocole de Corba) n'est pas particulièrement adapté aux firewall, ce qui pose problèmes dans une visibilité « internet »

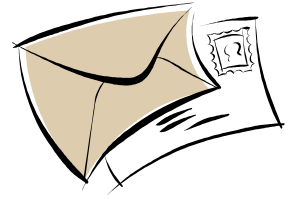
*Il est primordial de comprendre que SOAP est un protocole basé sur XML et en conséquence, particulièrement verbeux. CORBA, au travers de IIOP, est plus performant car les opérations de conversion et de déconversion (marshalling et demarshalling) sont plus efficaces puisque les données en format binaire (CDR Common Data Representation).*

# Historique



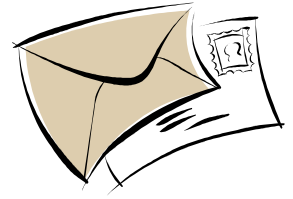
- **Années 80 : CORBA et DCOM**
  - ▶ Spécification d'interface en IDL
  - ▶ Fortement orientés objet
  - ▶ Couplage fort entre les systèmes
  
- **XML-RPC (1999)**
  - ▶ Messages XML
  - ▶ Envoi de formulaire (HTTP-POST),
  - ▶ Pas extensible, types de données limités.

# *Les points forts*



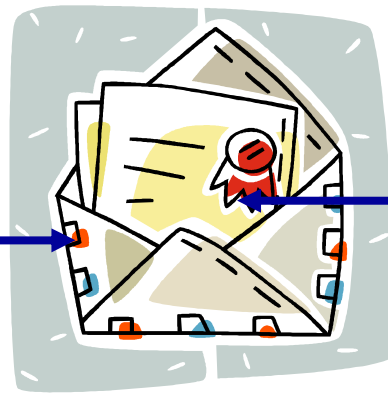
- Protocole de communication entre applications
- Basé sur XML et les namespaces.
- Communication par le Web (HTTP / SMTP / ...)
- Indépendant de la plateforme (windows, unix, mac, ...)
- Simple et extensible
- Bientôt un standard w3c (SOAP 1.2).

# Les principes



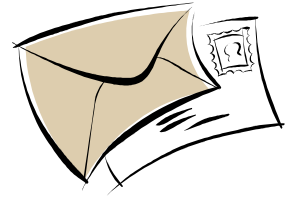
- Permet d'envoyer des messages XML entre deux machines.
- Les messages sont « emballés » dans une enveloppe SOAP
- L'enveloppe SOAP est une grammaire prédéfinie
- La grammaire du message dépend de l'application

Enveloppe repose sur  
une grammaire SOAP  
identique pour tous les  
messages



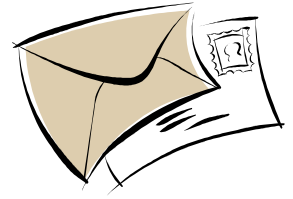
Le message est  
dépendant de  
l'application puisqu'il  
indique la méthode et  
les paramètres

# Les principes



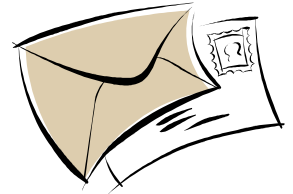
- **Modèle pour le RPC :**
  - ▶ Message Request invoque une méthode d'un objet distant
  - ▶ Message Response renvoie le résultat de son exécution
  
- **Encodage de types de données des langages de programmation, comme :**
  - ▶ Tableaux,
  - ▶ Enregistrements,
  - ▶ ...
  
- **A noter, encore une fois, que cela est hérité du modèle des schémas XML qui permettent la représentation de structures de données arbitrairement complexes (arbres, pointeurs, ...)**

# Les principes



- Utilisable avec des protocoles :
  - ▶ Synchrones : HTTP,
  - ▶ Asynchrones : SMTP, JMS, ...
  
- Gestion des erreurs (SOAP Fault)
  
- Entêtes (SOAP Header) :
  - ▶ utilisation de méta-données pour un ou plusieurs destinataires du message qui peuvent chacun modifier les entêtes (audit, suivi, etc.).
  
- Une spec du w3c en cours de finalisation (SOAP 1.2)
  - ▶ Vers plus de modularité
  - ▶ mais aussi plus de complexité.

# *Les messages SOAP : présentation*



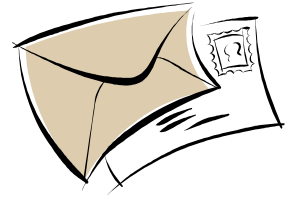
```
<soap:Envelope
  xmlns:soap=http://www.w3.org/2001/12/soap-envelope
  soap:encodingStyle=http://www.w3.org/2001/12/soap-encoding>

  <soap:Header>
    ...
  </soap:Header>

  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

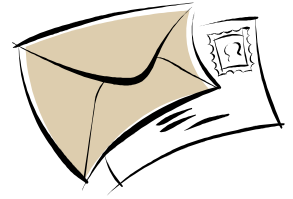


# Les messages SOAP : présentation



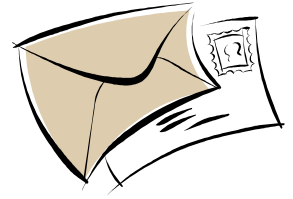
- Un message SOAP valide est un document XML correctement formé.
- Le prologue XML peut être présent, mais dans ce cas, ne doit contenir qu'une déclaration XML (c-à-d. qu'il ne doit contenir ni référence à un DTD, ni instruction XML).
- Le message doit utiliser l'enveloppe SOAP et les namespaces d'encodage SOAP, et doit avoir la forme suivante:
  - ▶ Une déclaration XML (optionnelle)
  - ▶ Une Enveloppe SOAP (l'élément racine) qui est composée de :
    - Une En-tête SOAP (optionnel)
    - Un Corps SOAP

# Les messages SOAP : exemple



- Un dialogue RPC encodé par SOAP contient un message de requête et un message de réponse.
- Considérons la méthode d'un service simple qui double la valeur d'un entier donné dont voici la signature Java :
  - ▶ `int doubleAnInteger ( int numberToDouble );`

# Les messages SOAP : exemple

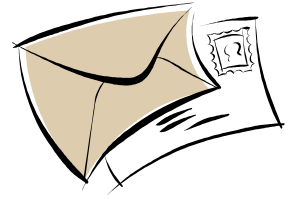


- Voici un exemple de requête SOAP sur un service définissant la méthode décrite précédemment :




```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doubleAnInteger
      xmlns:ns1="urn:MySoapServices">
      <param1 xsi:type="xsd:int">123</param1>
    </ns1:doubleAnInteger>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# Les messages SOAP : exemple

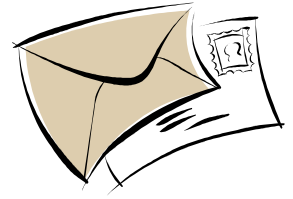


➤ Voici un exemple de réponse SOAP sur un service définissant la méthode décrite précédemment :



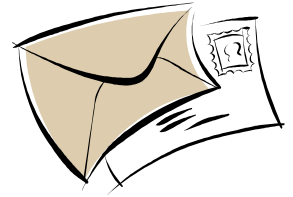
```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doubleAnIntegerResponse
      xmlns:ns1="urn:MySoapServices"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:int">246</return>
    </ns1:doubleAnIntegerResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# *Les messages SOAP : le prologue*



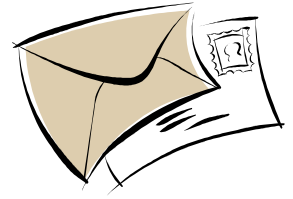
- Le prologue XML contient seulement une déclaration XML `<?xml version="1.0" encoding="UTF-8" ?>` spécifiant la version de XML et l'encodage des caractères du message XML.

# Les messages SOAP : l'enveloppe



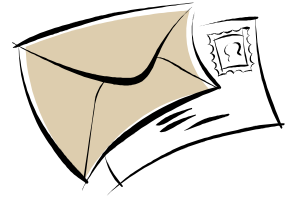
- La balise de l'Enveloppe SOAP <SOAP-ENV:Envelope ... > dans le message de requête spécifie tout d'abord le style d'encodage (défini dans <http://schemas.xmlsoap.org/soap/encoding/>). Cette balise est optionnelle comme c'est le cas dans le message de réponse.
- L'Enveloppe SOAP contient également des définitions de namespaces. Les identifiants des namespaces sont standards et la spécification SOAP demande à ce que ces namespaces soient définis correctement ou pas du tout (c-à-d qu'un message SOAP dans lequel manquent des définitions de namespaces est correct et peut être exploité mais un message contenant des définitions incorrectes, c-à-d non standards, est mauvais et refusé). Notez que la définition du namespace SOAP-ENC est absent du message de réponse mais cela ne signifie pas que le message est invalide.

# Les messages SOAP : l'entête



- Il n'y a pas de tag header (en-tête) SOAP dans cet exemple.
- Les en-têtes SOAP sont optionnelles et sont typiquement utilisées pour transmettre des données d'authentification ou de gestion de session.
- A noter que l'authentification et la gestion de session sont en dehors du cadre du protocole SOAP, même si les concepteurs de SOAP autorisent une certaine flexibilité dans la transmission de messages SOAP, de telle façon que les personnes qui les implémentent puissent inclure de telles informations.

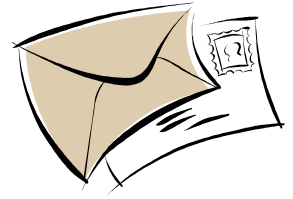
# Les messages SOAP : le corps



- La balise SOAP Body (le corps) `<SOAP-ENV:Body>` qui n'a rien de remarquable en elle-même mais encapsule une unique balise de méthode qui porte le nom de la méthode elle-même `<ns1:doubleAnInteger ... >` (ou, le même nom suivi du mot "Response" dans le cas du message de réponse).
- La balise de la méthode reçoit typiquement le namespace correspondant au nom du service, dans notre cas `urn:MySoapServices` pour assurer l'unicité (un service web, qui peut contenir n'importe quel nombre de méthodes nommées différemment, a un nom de service unique à l'URL sur laquelle il est accessible

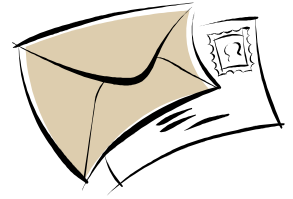


# Les messages SOAP : le corps



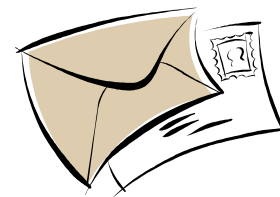
- La balise de méthode encapsule à son tour n'importe quel nombre de paramètres comme par exemple la balise <param1 ... >
- Les noms des balises de paramètres peuvent être de n'importe quel nom qui peut être autogénérés ou défini explicitement
- Dans le message de réponse, il n'y a qu'une seule balise de paramètre (représentant la valeur de retour de la méthode). Elle porte le nom <return>

# Les messages SOAP : compléments



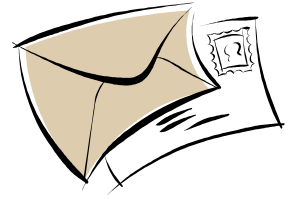
- L'une des caractéristiques les plus intéressantes du protocole SOAP est sa capacité à gérer des paramètres de tout niveau de complexité.
- Cette capacité est directement déduite du modèle des schémas XML et consiste à traiter des types primitifs (entier, chaîne de caractères etc...), des tableaux et structures et toutes combinaisons de ceux-ci.
- Voyons pour finir, à quoi ressemblent les dialogues SOAP pour des méthodes avec des types de paramètres et de retour complexes.

# Les messages SOAP : compléments



- Dans les transparents suivants, nous donnons le dialogue résultant de l'appel de la version initiale de `getEmployeeDetails` comme décrite précédemment.
- Dans cette version, le client envoie un entier (l'identification de l'employé) et reçoit un tableau de chaînes de caractères à deux éléments contenant :
  - ▶ le nom de l'employé
  - ▶ et le numéro de téléphone
- Dans la balise `<return>` du message de réponse, nous avons le type de la structure complexe (le tableau de chaînes), à savoir :
  - ▶ `xsi:type="ns2:Array" ns2:arrayType="xsd:string[2]"`

# Les messages SOAP : compléments

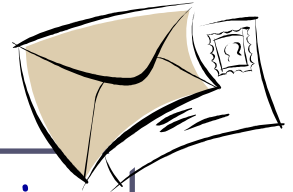


➤ Voici un exemple de requête SOAP pour notre nouveau service :



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:getEmployeeDetails
      xmlns:ns1="urn:MySoapServices">
      <param1 xsi:type="xsd:int">1016577</param1>
    </ns1:getEmployeeDetails>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

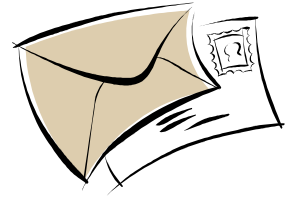
# Les messages SOAP : compléments



➤ Voici un exemple de requête SOAP pour notre nouveau service :

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ns1:getEmployeeDetailsResponse
      xmlns:ns1="urn:MySoapServices"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return
        xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
        xsi:type="ns2:Array"
        ns2:arrayType="xsd:string[2]">
        <item xsi:type="xsd:string">Bill Posters</item>
        <item xsi:type="xsd:string">+1-212-7370194</item>
      </return>
    </ns1:getEmployeeDetailsResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# Style RPC ou DOC



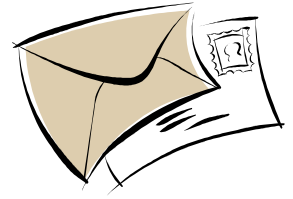
## ➤ Style RPC

- ▶ Appels de procédures distants
- ▶ Paramètres proches des types des langages de programmation
- ▶ Degré élevé d'automatisation

## ➤ Style DOC

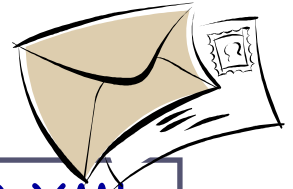
- ▶ Echanges de messages conformes à des schémas arbitraires (Ex: Demande d'achat).
- ▶ Plus d'expressivité
- ▶ Encouragé par .Net

# *Synchrone ou Asynchrone*



- Aujourd'hui : beaucoup de services synchrones,
  - ▶ au dessus d'HTTP.
  - ▶ Appels bloquants
  - ▶ Pas de garanties (timeout ?)
  
- Plus robuste : Echanges asynchrones
  - ▶ SMTP, JMS, ...
  - ▶ Non-bloquants
  - ▶ Garanties de service (Exactly Once)

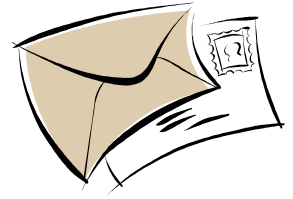
# Architecture technique côté client



- Les messages envoyés au serveur seront des requêtes SOAP-XML enveloppées dans des requêtes HTTP.
- De même, les réponses du serveur sont des réponses HTTP qui renferment des réponses SOAP-XML.
- Du côté client, pour ne pas prendre en charge la sérialisation SOAP et l'encodage HTTP, nous utilisons un package SOAP spécifique.
- Nous invoquons ensuite le service, simplement en invoquant la méthode appropriée du package SOAP (typiquement en spécifiant l'URL du service, le nom du service et tous les paramètres requis). Le premier travail d'un package est de sérialiser l'invocation de ce service en requête SOAP. Il doit ensuite encoder ce message dans une requête HTTP et l'envoyer à l'URL spécifiée.

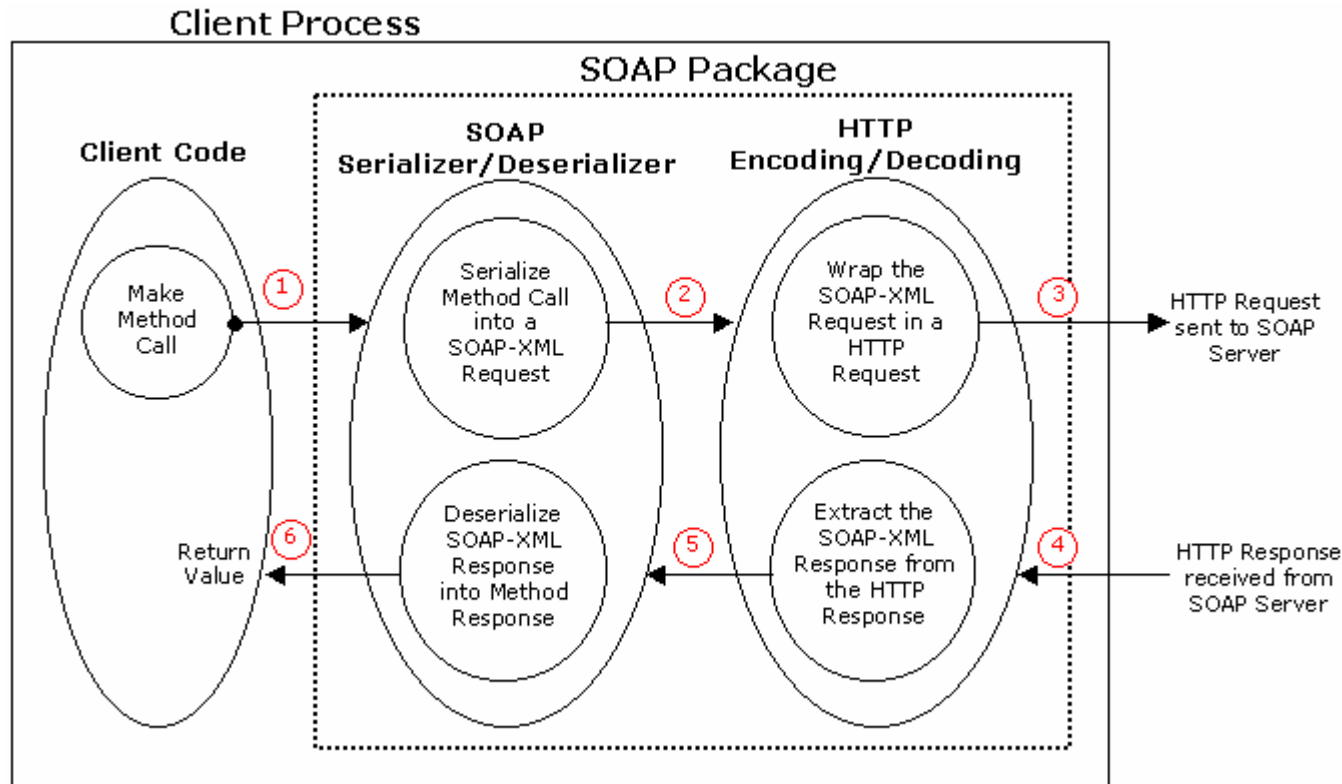
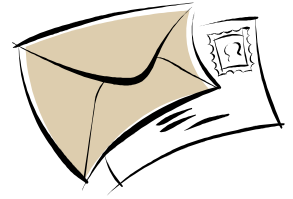


# Architecture technique côté client



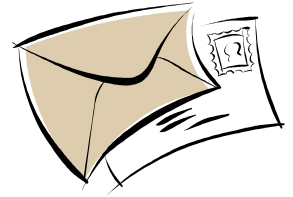
- Nous verrons la façon dont le serveur traite la requête, mais pour l'heure, il nous renvoie le message encodé HTTP contenant la réponse SOAP.
- Nous nous reposons sur le même package SOAP pour exécuter l'inverse de ce qui fut fait au stade de la requête, c'est-à-dire que le package décode le message HTTP et extrait le message SOAP, ensuite désérialise le message SOAP et obtient la valeur de retour de l'appel de la méthode. Cette valeur de retour trouvée est ensuite passée comme valeur de retour à l'invocation originale de la méthode par le code client.

# Architecture technique côté client



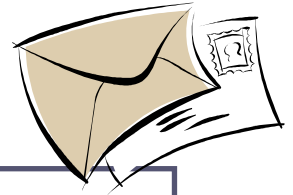
*Le code client crée un appel de service en invoquant la méthode appropriée du package SOAP (1). Le serialiseur SOAP du package SOAP convertit cette invocation en requête SOAP et l'envoie à l'encodeur HTTP (2). L'encodeur HTTP enveloppe le message SOAP dans une requête HTTP et l'envoie au serveur SOAP (3). La réponse est reçue par le module d'encodage/décodage HTTP du serveur SOAP(4); ce module décode et extrait la réponse SOAP qui la remet au deserialiseur SOAP (5). Le deserialiseur SOAP deserialise le message et passe le résultat au code client (6) comme valeur de retour de l'invocation originale (1).*

# Architecture technique côté serveur



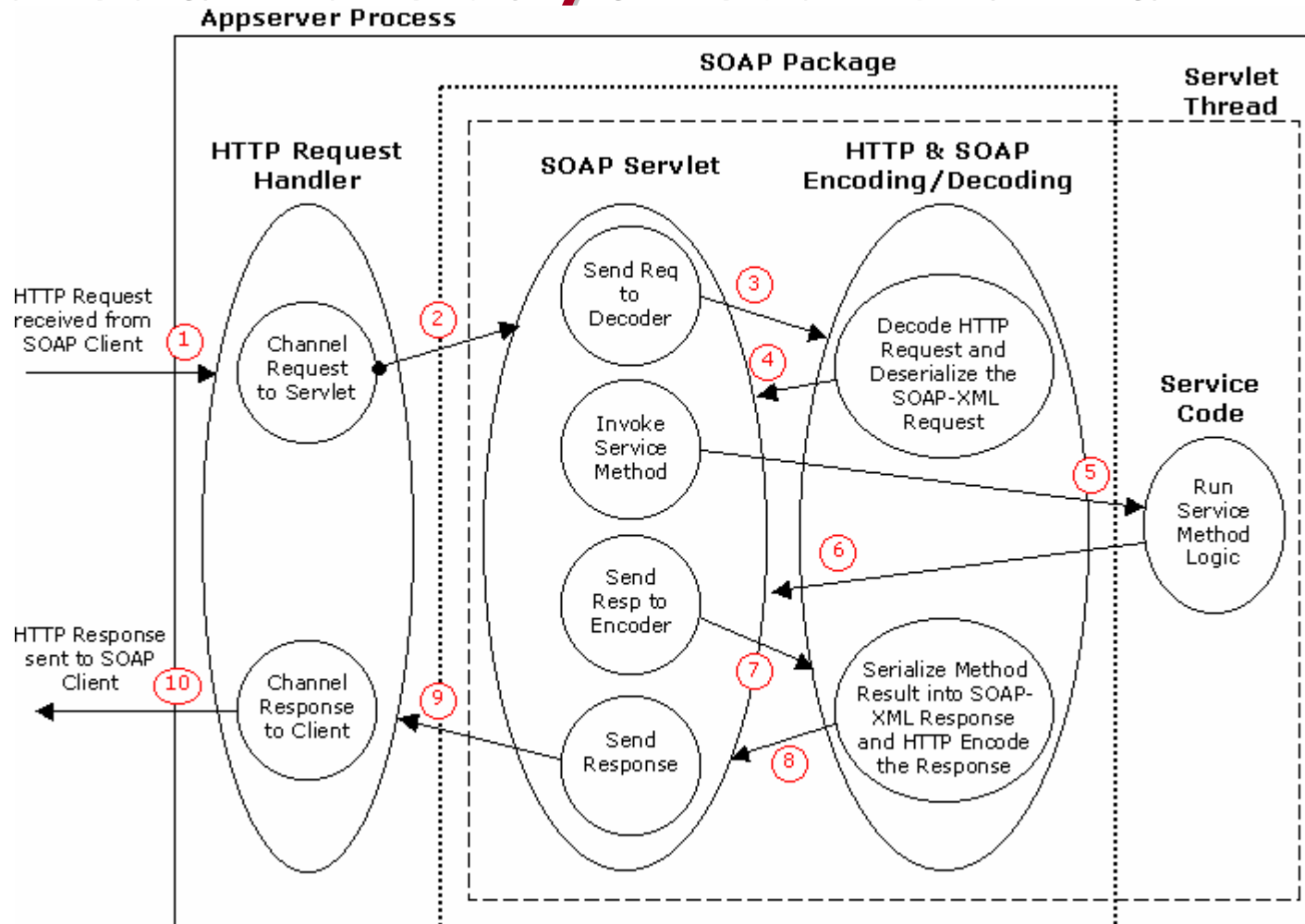
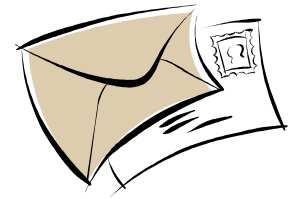
- Il est important de se rappeler que le choix du langage, de plate-forme et de package SOAP pour consommer des services web du côté client est entièrement indépendant du langage, de la plate-forme et du package SOAP utilisés par le côté serveur pour fournir des services web.
- De cette façon, le même service web basé sur SOAP (déployé par exemple sur UNIX, écrit en Java et exploitant Apache SOAP pour Java) peut être consommé par tout type de client écrit pour n'importe quelle plate-forme, dans n'importe quel langage, exploitant n'importe quel package SOAP applicable à cette combinaison langage/plate-forme. C'est l'une des grandes forces de la technologie SOAP.

# Architecture technique côté serveur



- C'est légèrement plus complexe puisque nous avons besoin d'un process "listener" (Le Listener est le process serveur qui est en attente de connexion client). Nous avons également besoin d'une implémentation du service lui-même. A part cela, nous nous reposons sur un package SOAP de la même façon que du côté client.
- Le listener est souvent implémenté au travers d'une servlet qui s'exécute comme une application web sur un appserver (comme c'est le cas lorsque nous utilisons Apache SOAP du côté serveur). Le serveur sera configuré pour passer toutes les requêtes destinées à une certaine URL (l'URL du service SOAP) à une servlet particulier (appelons-la servlet SOAP).
- Le travail de la servlet SOAP est d'extraire le message XML-SOAP de la requête HTTP, de le désérialiser (de ce fait, séparer le nom de la méthode et les paramètres fournis), et d'invoquer la méthode du service en conséquence. Le résultat de la méthode est alors sérialisé, encodé HTTP et renvoyé au demandeur.

# Architecture technique côté serveur



Le serveur d'application web reçoit une requête HTTP du Client SOAP destinée à l'URL de service SOAP(1) et, en conséquence de quoi, le passe à la servlet SOAP (2). La servlet SOAP utilise les fonctionnalités de décodage SOAP et HTTP incluses dans le package pour extraire les détails de l'appel des services (3 et 4), c'est-à-dire le nom et les paramètres de la méthode. Une fois muni de ceux-ci, la servlet SOAP peut invoquer la méthode (5 et 6), encoder la réponse (7 et 8) et fournir la réponse HTTP au handler de requêtes HTTP (9), qui à son tour, répond au client SOAP (10)

*Le protocole SOAP*  
*Le langage WSDL*  
*L'annuaire UDDI*



*Présentation*

*Structure*

*Exemple*

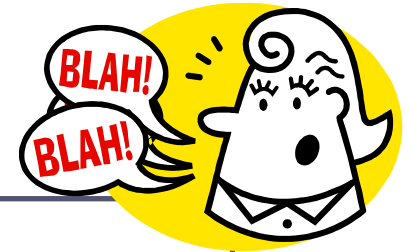
# Présentation



- Une interface qui cache le détail de l'implémentation du service, permettant une utilisation indépendante :
  - ▶ de la plate-forme utilisée
  - ▶ du langage utilisé
- Le fichier WSDL est au format XML et regroupe toutes les informations nécessaires pour interagir avec le Web Service :
  - ▶ les méthodes
  - ▶ les paramètres et valeurs retournées
  - ▶ le protocole de transport utilisé
  - ▶ la localisation du service

*Documents WSDL, générés par les outils de développement favorisent une intégration rapide du service*

# Présentation



- 2 types de documents WSDL :
  - ▶ le document WSDL décrivant l'interface du service
  - ▶ le document WSDL décrivant l'implémentation du service
- Documents indispensables au déploiement de Web Services
- Publication et recherche de services au sein de l'annuaire UDDI se font via ces 2 types de document WSDL

*Pour l'accès à un service particulier, un client se voit retourné l'URL du fichier WSDL décrivant l'implémentation du service. Seul l'emplacement de ce fichier WSDL est indiqué puisque ce dernier référence l'autre document WSDL décrivant l'interface de mise en œuvre du service.*

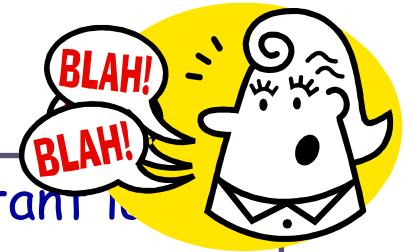


# Structure



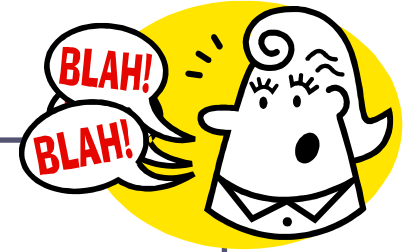
- Un document WSDL est constitué de plusieurs parties permettant la plus grande abstraction possible dans la définition des services.
- Ces différents éléments permettent de séparer les briques habituellement mises en jeu dans l'utilisation des services
- Ces briques sont pour l'essentiel :
  - ▶ L'adresse où est situé le service
  - ▶ Le protocole associé à l'utilisation du service
  - ▶ L'ensemble des opérations accessibles
  - ▶ Les types de données à véhiculer dans les opérations
  - ▶ ...

# Structure



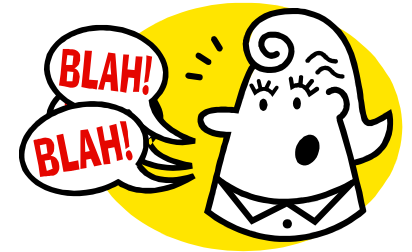
- Dans un document WSDL, on trouvera 6 parties permettant la séparation dont nous venons de parler.
- Les quatre premières parties décrivent des informations abstraites indépendantes d'un contexte de mise en œuvre. On y trouve :
  - ▶ les types de données envoyées et reçues
  - ▶ les opérations utilisables
  - ▶ le protocole qui sera utilisé,
- Les deux dernières parties décrivent des informations liées à un usage contextuel du service. On y trouve :
  - ▶ l'adresse du fournisseur implémentant le service,
  - ▶ le service qui est représenté par les adresses des fournisseurs

# Structure



- ▶ Les quatre premières parties, décrivant les informations abstraites indépendantes, sont les suivantes :
  - ▶ **Partie 1 - Les types**  
Un type décrit la structure de donnée transmise dans un message.
  - ▶ **Partie 2 - Les messages**  
Un message décrit l'ensemble des données transmises au cours de l'opération (ce peut être une requête ou une réponse)
  - ▶ **Partie 3 - Les types de ports**  
Un type de port est composé de l'ensemble des opérations abstraites applicables au service.
  - ▶ **Partie 4 - Les liaisons**  
La liaison décrit la façon dont un ensemble d'opérations abstraites, appelé « type de port », est lié à un port selon un protocole réel

# Structure



► Les deux dernières parties, décrivant des informations liées à un usage contexte du service, sont les suivantes :

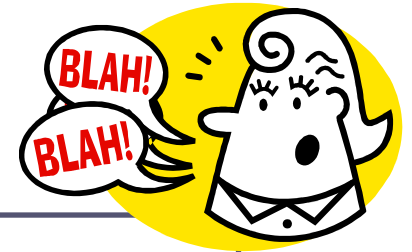
► **Partie 5 - Les ports**

Un port spécifie une adresse qui est associée à une « liaison » définissant un simple point de terminaison

► **Partie 6 - Le service**

Un service est décrit comme un ensemble de points finaux du réseau appelés « ports »

# Structure



## ➤ *Exemple : la gestion de compte*

Pour illustrer le rôle des différentes parties, nous allons prendre comme support, une application très simple de gestion de compte, dont voici la définition de l'interface Java :



```
import java.util.*;

public interface CompteInterface {

    public void depotDe(int montant);

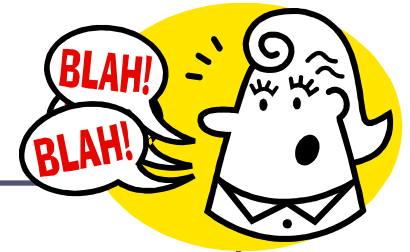
    public boolean retraitDe(int montant);

    public int valeurDuSolde();

    public Vector listeMouvements();

}
```

# Structure



## ➤ *Partie 1 - Les types*

Un type décrit la structure de données transmise dans un message. Par exemple, dans la gestion de compte, la méthode `listeMouvements` retourne un `Vector`. Nous aurons alors la description de ce type, comme illustré ici :

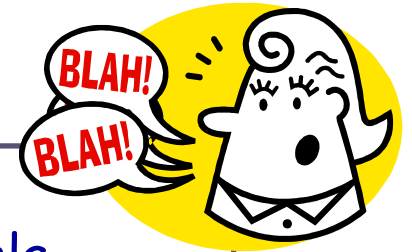
```
<wsdl:types>
  <schema targetNamespace="http://xml.apache.org/xml-soap"
    xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    <complexType name="Vector"> ← Le type défini
      <sequence>
        <element maxOccurs="unbounded" minOccurs="0" name="item"
          type="xsd:anyType" />
      </sequence> ← Le type est une séquence WSDL
    </complexType>
  </schema>
</wsdl:types>
```

← Une liste d'éléments de taille quelconque et dont les éléments sont de type quelconque



*A noter que dans cet exemple, nous n'aurons que cette déclaration de type. Les autres sont des types primitifs connus de WSDL.*


# Structure



## ➤ *Partie 2 - Les messages*

Un message décrit tous les cas d'usage d'une opération (cela recouvre l'appel, i.e. la requête HTTP, et le retour, i.e. la réponse HTTP).

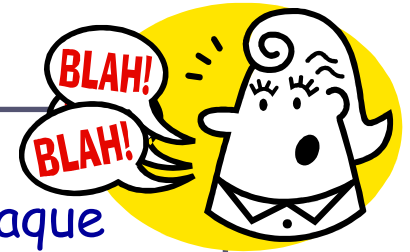
Par exemple, dans la gestion de compte, la méthode `listeMouvements` disposera de deux messages (un pour l'appel et un pour la réponse). Nous aurons alors la description de ces deux messages, comme illustré ici :



```
<wsdl:message name="listeMouvementsRequest" />

<wsdl:message name="listeMouvementsResponse">
  <wsdl:part name="listeMouvementsReturn" type="apachesoap:Vector" />
</wsdl:message>
```

# Structure



## ➤ *Partie 2 - Les messages*

Ainsi, comme expliqué précédemment, nous aurons pour chaque méthode déclarée dans notre interface, et considérée comme accessible par le client, deux messages.

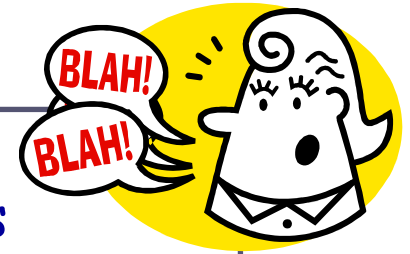
Dans notre exemple, nous aurons :



```
<wsdl:message name="listeMouvementsRequest" />
+ <wsdl:message name="depotDeRequest">
  <wsdl:part name="in0" type="xsd:int" />
</wsdl:message>
+ <wsdl:message name="listeMouvementsResponse">
  <wsdl:part name="listeMouvementsReturn" type="apachesoap:Vector" />
</wsdl:message>
+ <wsdl:message name="valeurDuSoldeResponse">
  <wsdl:part name="valeurDuSoldeReturn" type="xsd:int" />
</wsdl:message>
<wsdl:message name="depotDeResponse" />
<wsdl:message name="valeurDuSoldeRequest" />
+ <wsdl:message name="retraitDeResponse">
  <wsdl:part name="retraitDeReturn" type="xsd:boolean" />
</wsdl:message>
+ <wsdl:message name="retraitDeRequest">
  <wsdl:part name="in0" type="xsd:int" />
</wsdl:message>
```




# Structure



## ➤ *Partie 3 - Les types de ports*

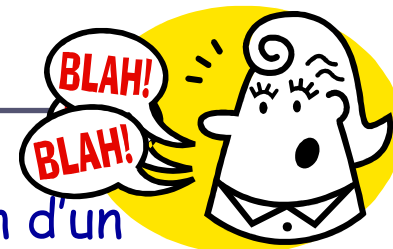
Un type de port est composé de l'ensemble des opérations abstraites applicables au service. On entend par opération abstraite, la signature de la méthode (i.e. la déclaration dans une interface Java ou dans un fichier WSDL). Une opération est composée d'une séquence de messages (en fait un pour l'appel et un pour le retour) à un « mode » d'invocation particulier du service. Par exemple, dans un contexte composants distribués, on entend par mode d'invocation, une méthode distante applicable au service.

En reprenant notre exemple, pour la méthode `listeMouvements`, nous aurons alors les deux messages associés :



```
<wsdl:operation name="listeMouvements">
  <wsdl:input message="impl:listeMouvementsRequest"
    name="listeMouvementsRequest" />
  <wsdl:output message="impl:listeMouvementsResponse"
    name="listeMouvementsResponse" />
</wsdl:operation>
```

# Structure



## > Partie 3 - Les types de ports

Pour notre exemple complet, nous aurons alors la définition d'un seul type de port, celui correspondant au service « Compte », avec les 4 opérations abstraites, correspondant aux 4 déclarations de méthodes dans l'interface Java :

```
<wsdl:portType name="Compte"> ← Le nom du type de port
+ <wsdl:operation name="depotDe" parameterOrder="in0">
  <wsdl:input message="impl:depotDeRequest" name="depotDeRequest" />
  <wsdl:output message="impl:depotDeResponse" name="depotDeResponse" />
</wsdl:operation>
+ <wsdl:operation name="retraitDe" parameterOrder="in0"> ← Un paramètre en entrée
  <wsdl:input message="impl:retraitDeRequest" name="retraitDeRequest" />
  <wsdl:output message="impl:retraitDeResponse" name="retraitDeResponse" />
</wsdl:operation>
+ <wsdl:operation name="valeurDuSolde">
  <wsdl:input message="impl:valeurDuSoldeRequest" name="valeurDuSoldeRequest" />
  <wsdl:output message="impl:valeurDuSoldeResponse" name="valeurDuSoldeResponse" />
</wsdl:operation>
- <wsdl:operation name="listeMouvements">
  <wsdl:input message="impl:listeMouvementsRequest" name="listeMouvementsRequest" />
  <wsdl:output message="impl:listeMouvementsResponse" name="listeMouvementsResponse" />
</wsdl:operation>
</wsdl:portType>
```

Le nom de l'opération

La séquence des messages  
la constituant

# Structure



## ➤ *Partie 4 - Les liaisons*

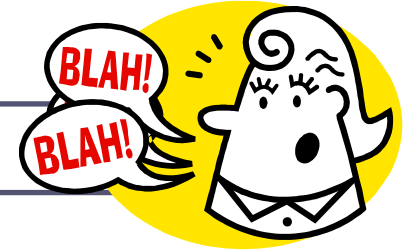
Une liaison décrit la façon dont un type de port (en d'autres termes l'abstraction du service, i.e. ses opérations abstraites) est mis en œuvre pour un protocole particulier (HTTP par exemple) et un mode d'invocation (RPC par exemple). Cette description est faite pour un ensemble donné d'opérations abstraites.

Pour un type de port, on peut avoir plusieurs liaisons, pour différencier les modes d'invocation (RPC ou autres) ou de transport (HTTP ou autre) des différentes opérations.

Comme précédemment, nous allons décrire les liaisons sur notre exemple de gestion de compte.

Pour des raisons de taille (le langage WSDL est très verbeux), nous ne décrivons la liaison que d'une seule opération abstraite.

# Structure



## > Partie 4 - Les liaisons

Le nom de la liaison

Le type de port concerné

Le mode d'invocation

```
<wsdl:binding name="CompteServiceBobSoapBinding" type="impl:Compte">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="depotDe">
    <wsdlsoap:operation soapAction="" />
    <wsdl:input name="depotDeRequest">
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost:8080/axis/services/CompteServiceBob"
        use="encoded" />
    </wsdl:input>
    <wsdl:output name="depotDeResponse">
      <wsdlsoap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://localhost:8080/axis/services/CompteServiceBob"
        use="encoded" />
    </wsdl:output>
  </wsdl:operation>
  ...
</wsdl:operation>
```

Le nom de l'opération dans le type de port

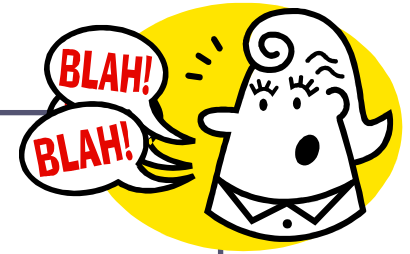
Le protocole

La représentation du message request

La représentation du message response

Pour un même port (Compte par exemple), on peut avoir plusieurs liaisons qui correspondent à des mises en œuvre protocolaires différentes des opérations du service. On ne retrouve pas dans les liaisons différentes, les mêmes opérations du service. On peut imaginer que l'opération « listeMouvements » soit asynchrone.

# Structure



## ➤ *Partie 5 - Les ports*

Un port spécifie une adresse URL qui correspond à l'implémentation du service par un fournisseur.

Le port est associée à une « liaison » définissant ainsi un simple point de terminaison



Le nom de la liaison associée

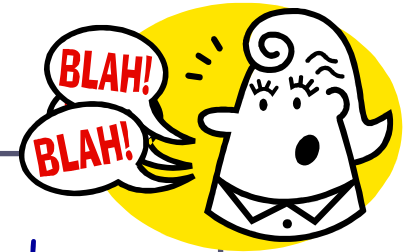
Le nom du port

```
<wsdl:port binding="impl:CompteServiceBobSoapBinding" name="CompteServiceBob">
  <wsdlsoap:address location="http://localhost:8080/axis/services/CompteServiceBob" />
</wsdl:port>
```

Comme indiqué précédemment, pour un même port (Compte par exemple), on peut avoir plusieurs liaisons qui correspondent à des mises en œuvre protocolaires différentes des opérations du service. On retrouve donc ici, les ports associés aux différentes liaisons définies.

L'adresse définissant le « endPoint »

# Structure



## > *Partie 6 - Le service*

Un service est décrit comme un ensemble de points finaux du réseau appelés « ports »

Le nom du service qui encapsule les ports

Le nom de la liaison associée au port

Le nom du port

```
<wsdl:service name="CompteService">
  <wsdl:port binding="impl:CompteServiceBobSoapBinding" name="CompteServiceBob">
    <wsdlsoap:address location="http://localhost:8080/axis/services/CompteServiceBob" />
  </wsdl:port>
</wsdl:service>
```

L'adresse définissant le « endPoint »

*Le protocole SOAP*  
*Le langage WSDL*  
*L'annuaire UDDI*



*UDDI, présentation*

*UDDI, description*

*Informations métier de l'entreprise*

*Informations sur le service métier*

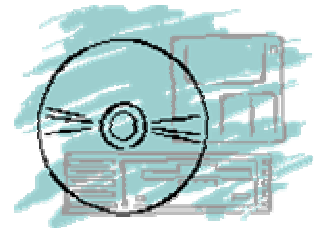
*Informations de liaison*

*Informations techniques*

*Utilisation de UDDI*



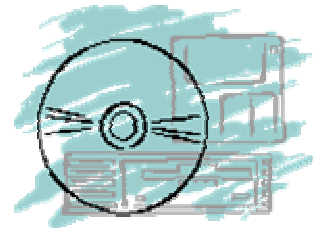
# UDDI, présentation



- UDDI (Universal Description, Discovery and Integration), standard né sous l'initiation de Microsoft, IBM en tête et d'un grand nombre d'industriels dont Sun, Oracle, Compaq, HP, Intel, SAP, etc.
- Standard pour faciliter la collaboration entre partenaires dans le cadre d'échanges commerciaux
- Le cœur de UDDI est un annuaire qui contient des informations techniques et administratives sur les entreprises et les services qu'elles publient

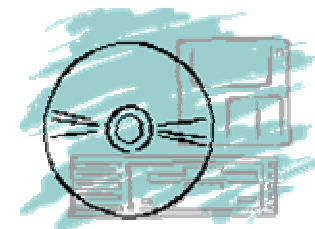


# UDDI, présentation



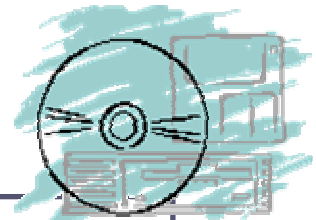
- L'annuaire UDDI permet de :
  - ▶ publier des informations sur une entreprise et ses services
  - ▶ découvrir des informations sur une entreprise et ses services
- L'inscription sur UDDI permet à une entreprise de se présenter ainsi que ses services
- L'adoption de UDDI facilite le développement des échanges de type « B2B »
- L'enregistrement des services dans un annuaire s'effectue après d'un opérateur (Microsoft ou IBM actuellement) à travers son site

# UDDI, présentation

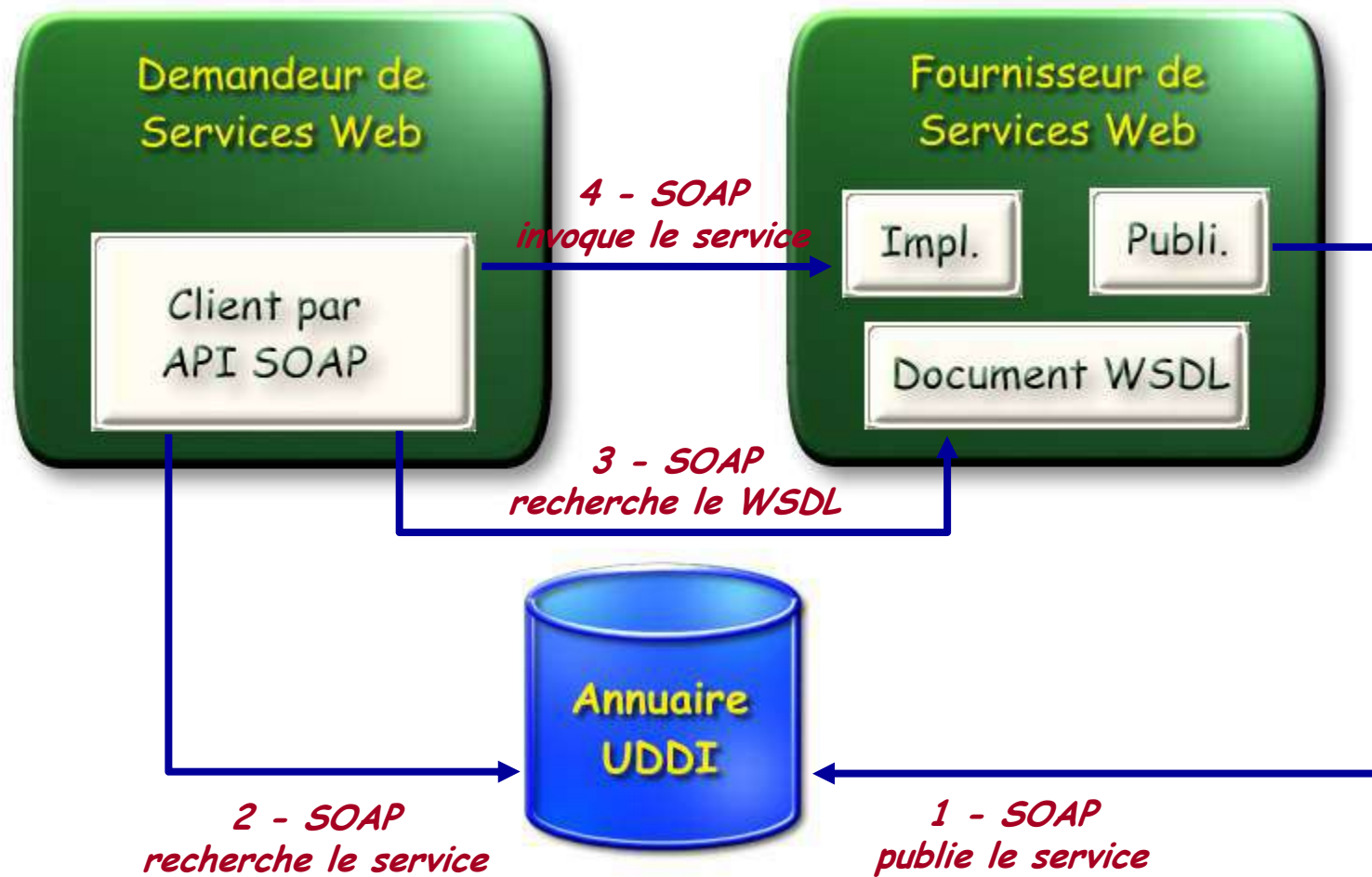


- Diverses recherches peuvent être effectuées, comme :
  - ▶ recherche par catégorie d'entreprise en utilisant des taxonomies ou des identifications d'entreprise
  - ▶ recherche d'un service particulier
- Par une API SOAP, on peut interagir avec UDDI au moment de la conception et de l'exécution des applications afin de découvrir des données techniques sur les entreprises et leurs services.
- Les services ainsi découverts peuvent ensuite être utilisés

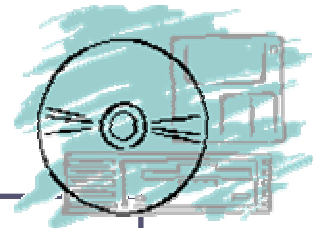
# UDDI, présentation



- UDDI repose sur le protocole SOAP et de ce fait, les requêtes et réponses sur l'annuaire sont des messages SOAP



# UDDI, présentation



➤ L'annuaire UDDI comportent plusieurs catégories de données :

▶ **Pages blanches**

Elles recensent les entreprises et contiennent donc des informations telles que :

- › le nom de l'entreprise
- › ses coordonnées
- › des descriptions accessibles aux clients
- › des identifiants permettant de la retrouver par recherche

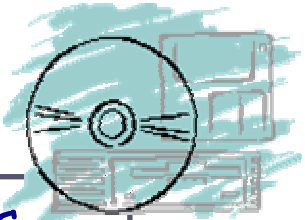
▶ **Pages jaunes**

Elles contiennent, au format WSDL, la description des services déployés par l'entreprise. Les services sont répertoriés par catégorie.

▶ **Pages bleues**

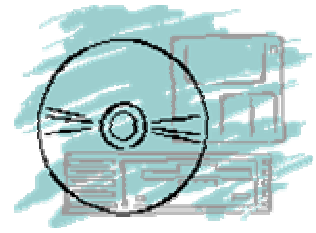
Elles fournissent des informations techniques détaillées sur les services

# UDDI, description



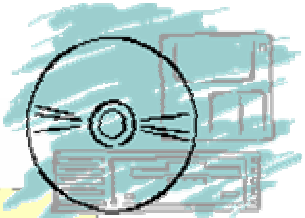
- ▶ Le modèle UDDI comporte 5 structures de données principales décrites sous forme de schémas XML :
  - ▶ **BusinessEntity** : ensemble d'informations sur l'entreprise qui publie les services dans l'annuaire
  - ▶ **BusinessService** : ensemble d'informations sur les services publiés par l'entreprise
  - ▶ **BindingTemplate** : ensemble d'informations concernant le lieu d'hébergement du service (i.e. adresse du fournisseur)
  - ▶ **tModel** : ensemble d'informations concernant le mode d'accès du service (définitions WSDL)
  - ▶ **publisherAssertion** : ensemble d'informations contractuelles entre partenaires dans le cadre d'échanges commerciaux

# Informations métier de l'entreprise

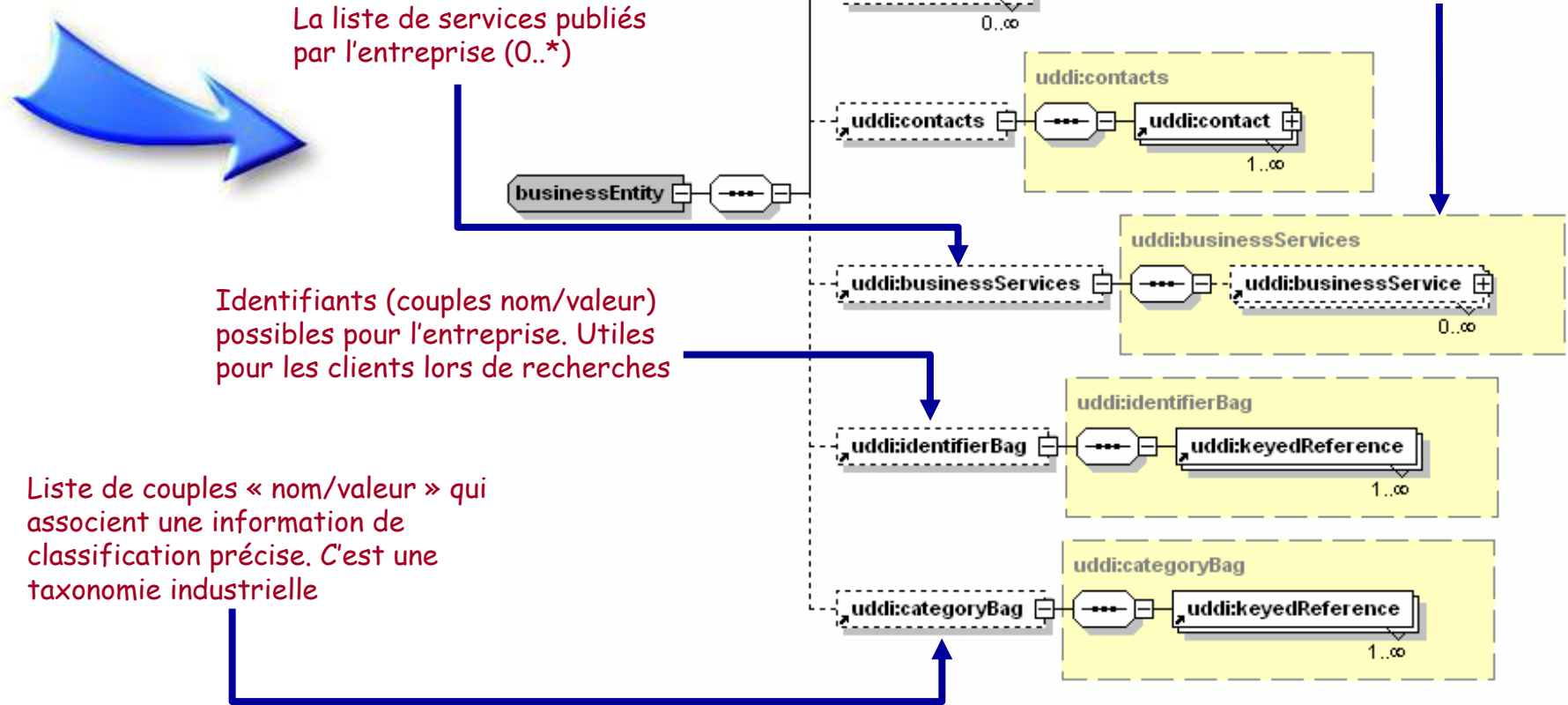


- BusinessEntity contient un ensemble d'informations sur le métier de l'entreprise. On y trouve, par exemple :
  - ▶ le type d'affaires que traite l'entreprise
  - ▶ le type de services qu'elle propose
  - ▶ son nom
  - ▶ sa description
  - ▶ ses coordonnées

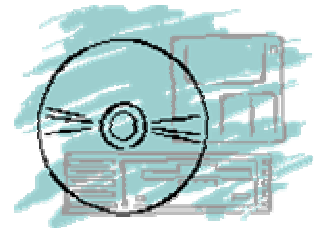
# Informations métier de l'entreprise



➤ Le schéma XML de la structure « BusinessEntity » est le suivant :



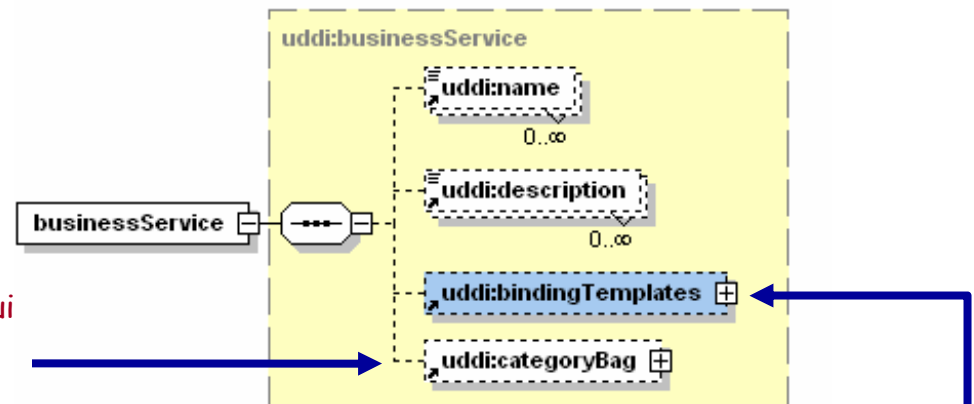
# Informations sur le service métier



- BusinessService contient un ensemble d'informations sur le service. On y trouve, par exemple :
  - ▶ le nom du service
  - ▶ sa description
  - ▶ son code



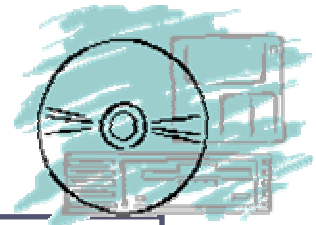
Liste de couples « nom/valeur » qui associe une information de classification précise.



Les modèles de liaison contenant la description technique du service

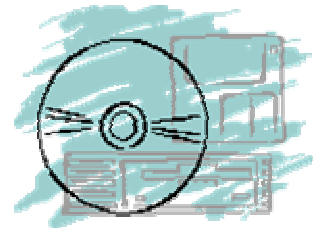


# Informations de liaison

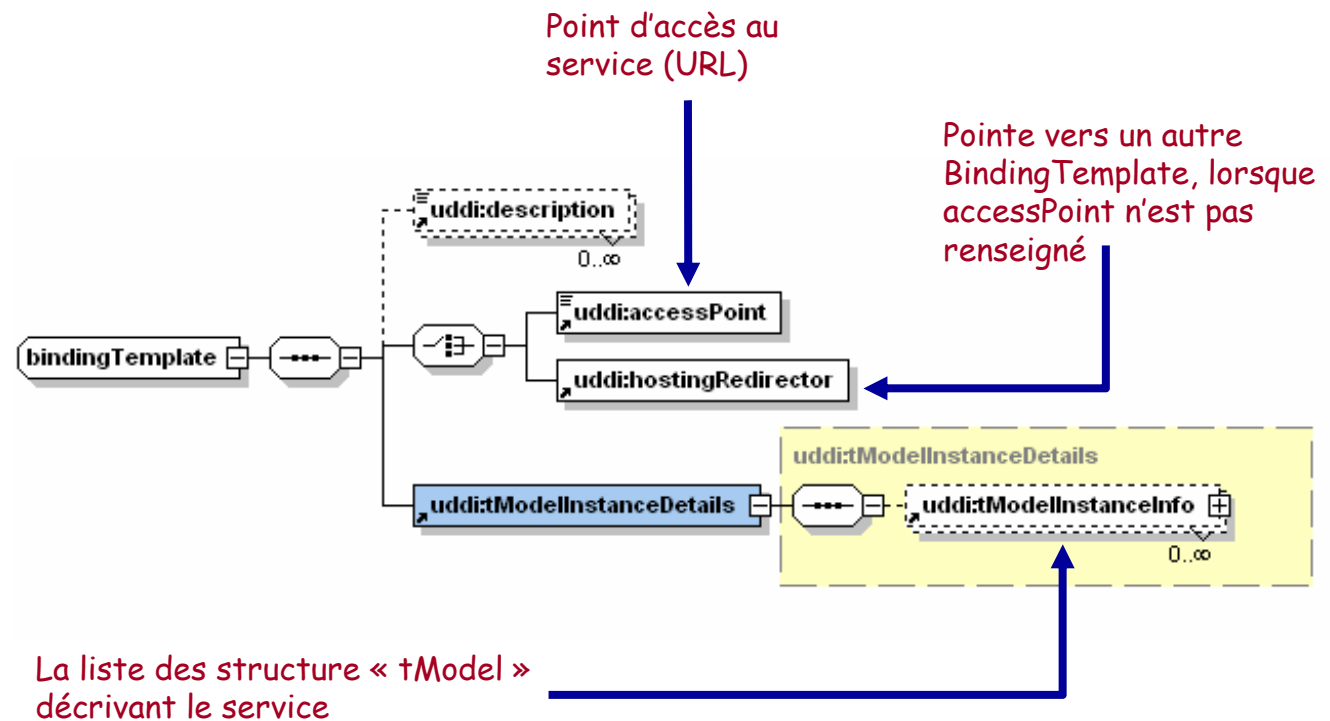


- BindingTemplate contient les informations de liaison pour accéder aux informations techniques sur le service.
- Il s'agit principalement :
  - ▶ des références aux « tModels » qui désignent les interfaces de mise en œuvres du service
  - ▶ du point de terminaison (endpoint sous forme d'une adresse Internet) du service
- Un service peut dispose de plusieurs modèles de liaison

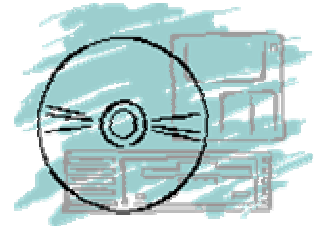
# Informations de liaison



➤ Le schéma XML de la structure « BindingTemplate » est le suivant :

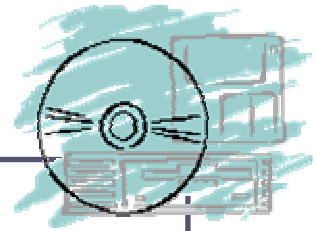


# Informations techniques



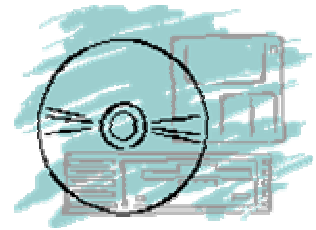
- Un tModel est un modèle type de mise en œuvre du service.
- Un tModel est généralement un fichier WDSL dont l'objectif est de définir :
  - ▶ les types utilisés par le service
  - ▶ les opérations applicables au service
- A partir d'un tModel, il est possible de connaître les opérations applicables et comment y accéder.
- C'est la représentation IDL de Corba ou de l'interface Java

# Informations techniques



- Avec les tModels, la spécification UDDI permet d'établir une distinction entre abstraction et implémentation :
  - ▶ La structure tModel représente la signature, les interfaces et les types de description techniques des métadonnées.
  - ▶ Les modèles BindingTemplate, associés au tModel, en représentent les implémentations.
  - ▶ On a une séparation entre interface et implémentation, au même titre que dans les APIs Java
  - ▶ Par exemple, un organisme peut décrire les spécifications d'une technologie sous forme de tModels
  - ▶ Plusieurs industriels peuvent ensuite en donner des implémentations sous forme de BindingTemplates qui référencent le même tModel

# Informations techniques

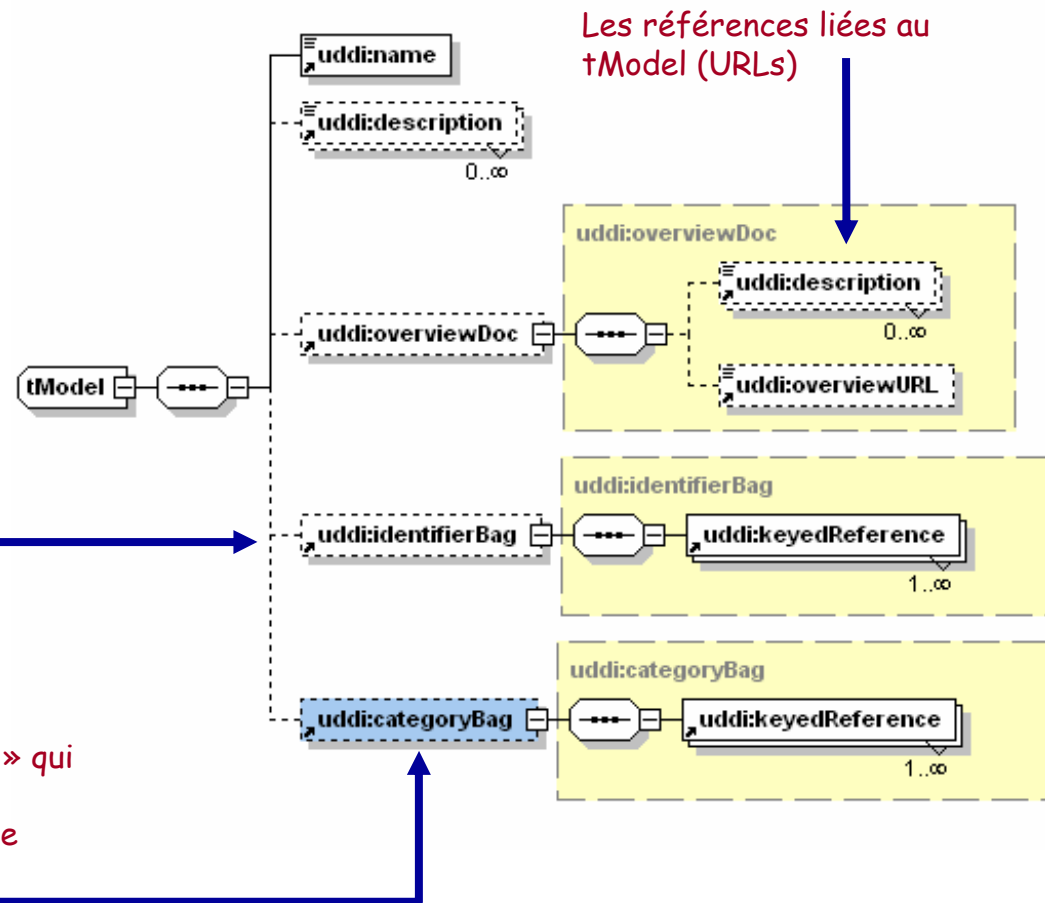


➤ Le schéma XML de la structure « tModel » est le suivant :



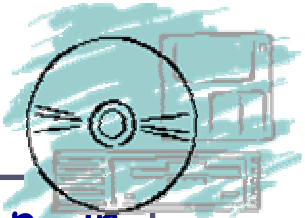
Identifiants (couples nom/valeur) possibles pour l'entreprise. Utiles pour les clients lors de recherches

Liste de couples « nom/valeur » qui associent une information de classification précise. C'est une taxonomie industrielle.



Les références liées au tModel (URLs)

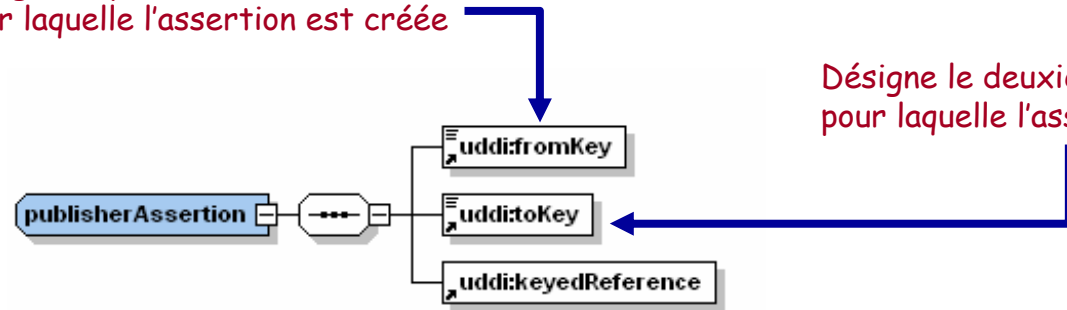
# Informations contractuelles



- PublisherAssertion représente des informations contractuelles pour les services publiés
- Les assertions représentent les règles de mise en œuvre sous la forme d'un protocole entre deux partenaires métier.
- Chaque rôle est défini par les assertions



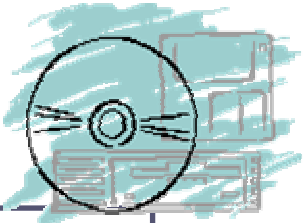
Désigne le première entité métier pour laquelle l'assertion est créée



Désigne le deuxième entité métier pour laquelle l'assertion est créée

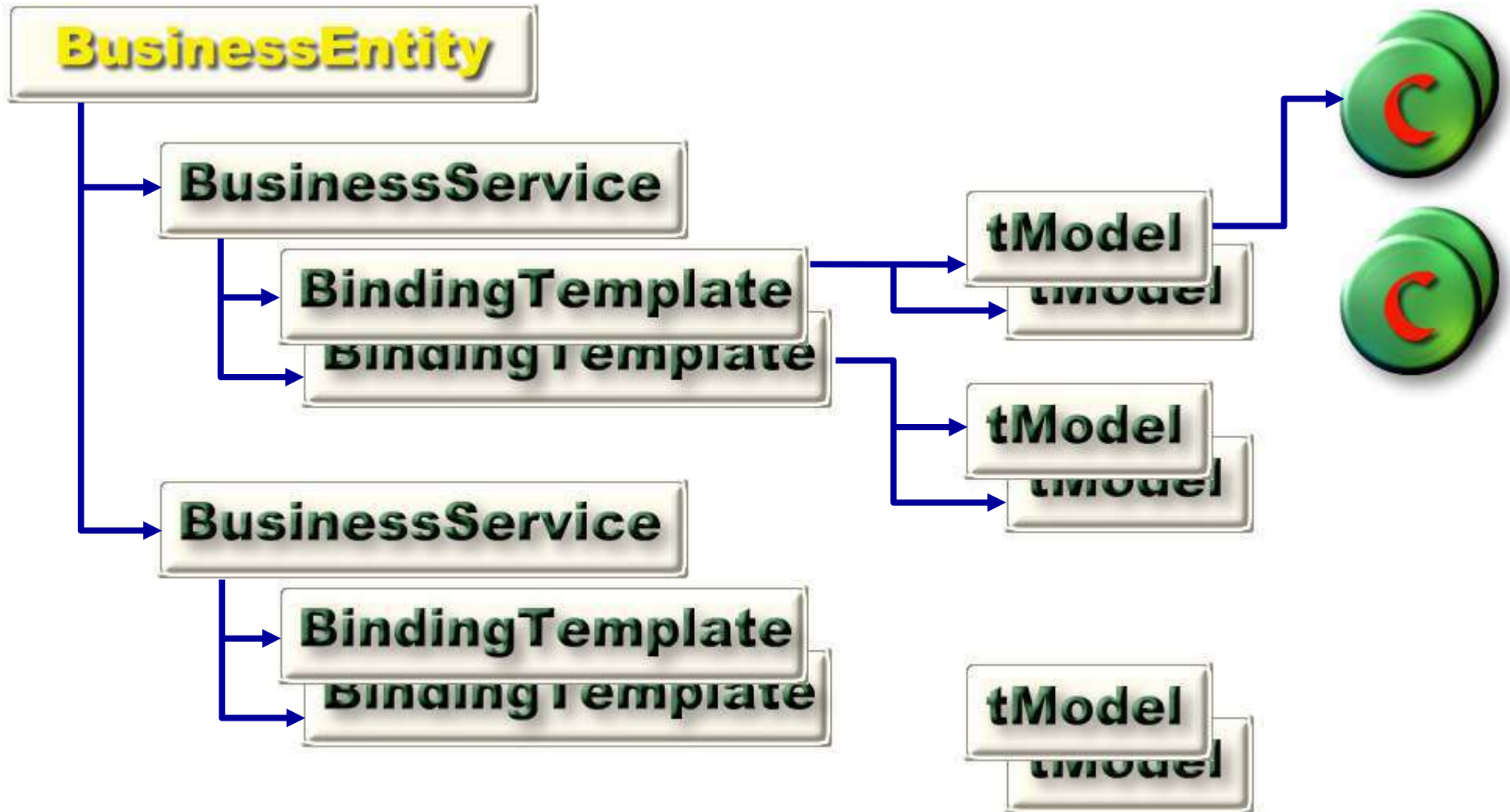
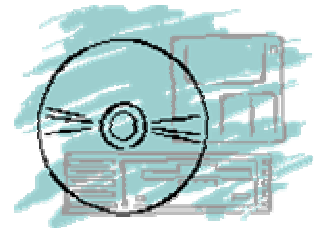
Désigne le type de relation

# Synthèse sur les informations



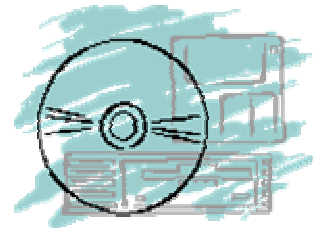
- Au cours de la modélisation des données de description de services dans l'annuaire, une entité métier **businessEntity** représente une information métier qui regroupe un ensemble d'entités services de type **BusinessService** qui vont être publiées.
- Une entité **BusinessService** contient des informations techniques et descriptives sur le service publié. Pour cela, il détient une collection d'entités de type **BindingTemplate**.
- Une entité **BindingTemplate** décrit l'accès au service et la façon de mettre en œuvre les spécifications techniques du service.
- Une entité **tModel** représente une spécification technique pouvant elle-même modéliser plusieurs aspects conceptuels

# Synthèse sur les informations



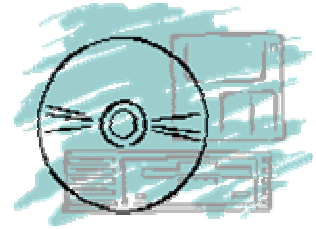


# Utilisation de UDDI



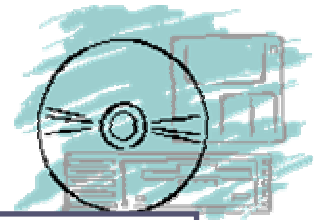
- L'annuaire UDDI permet de :
  - ▶ de publier les services web
  - ▶ de décrire les services web
  - ▶ de découvrir les services web

# Publication de services



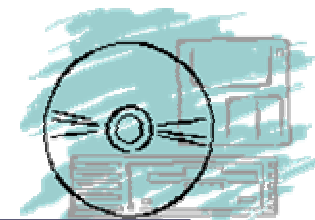
- La publication de services inclus aussi la création métier des services (l'implémentation et la description)
- Pour réaliser la publication et la description, il existe différentes possibilités :
  - ▶ manuellement
  - ▶ automatiquement (partiellement ou complètement suivant les cas)
- La publication la plus élémentaire consiste à envoyer directement au demandeur de service, la description WSDL correspondante (par email par exemple)

# Publication de services



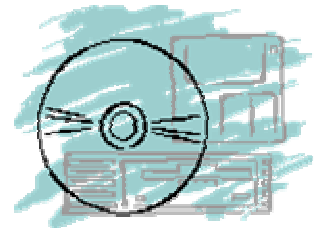
- Pour une publication dans un annuaire UDDI, il existe plusieurs possibilités, suivant la visibilité voulue par le fournisseur :
  - ▶ **Nœud interne UDDI** pour une application interne.  
Ils sont protégés par un firewall et sont utilisés en intranet
  - ▶ **Nœud catalogue du partenaire avec UDDI**  
Les services sont publiés sur un nœud catalogue d'un partenaire choisi avec les autorisations d'accès correspondantes
  - ▶ **Nœud portail UDDI** pour une application extranet  
Les services sont publiés pour que les partenaires externes puissent les utiliser
  - ▶ **Nœud place de marché UDDI** pour une relation inter-entreprises  
Les services sont largement accessibles pour un partage des ressources entre systèmes d'information

# Découverte de services



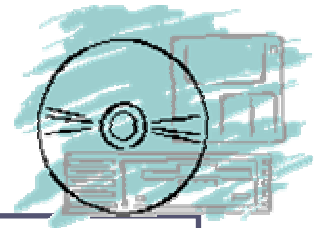
- La découverte de service recouvre :
  - ▶ L'acquisition des descriptions des services
  - ▶ La mise en œuvre, l'utilisation, des services
  
- L'acquisition est effectuée au cours de la modélisation pour obtenir des descriptions des services. La description peut être obtenue de plusieurs façons :
  - ▶ Mémorisée en cache (par programme par exemple) pour une utilisation ultérieure simplifiée
  - ▶ Dans un fichier local
  - ▶ Dans un annuaire local
  
- Au cours de l'exécution, les services dont on dispose de la description sont mis en œuvre.

# Description de services



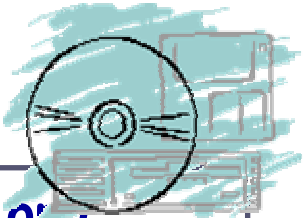
- Après en avoir obtenue la description, le client peut invoquer le service.
- Pour cela, il utilise la description, représentée par des descripteurs WSDL, pour générer les requêtes SOAP (ou les appels par programme depuis un langage quelconque)
- Les APIs SOAP disponibles au niveau du poste client permettent de cacher tous les détails liés au codage et au décodage des messages SOAP

# API de programmation UDDI



- Il existe deux APIs de programmation pour dialoguer avec un annuaire UDDI
- API d'interrogation dans un annuaire
  - ▶ Cela consiste à effectuer des interrogations, des recherches d'informations, sur les entrées d'un annuaire UDDI. Tout utilisateur a la possibilité d'effectuer des recherches et donc de mettre en œuvre cette API
- API de publication dans un annuaire
  - ▶ Pour permettre la publication ou la suppression de services au sein d'un annuaire. Cette API impose des autorisations d'accès et est destinée aux fournisseurs de services et aux entreprises désirant publier des informations les concernant

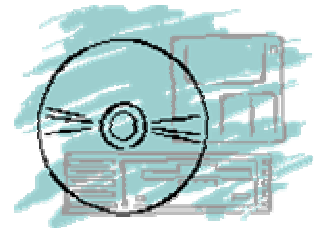
# API d'interrogation



Tirés directement des spécifications disponibles sur "[www.uddi.org](http://www.uddi.org)"

- ***find\_binding***: Used to locate specific bindings within a registered businessService. Returns a bindingDetail message.
- ***find\_business***: Used to locate information about one or more businesses. Returns a businessList message.
- ***find\_relatedBusinesses***: Used to locate information about businessEntity registrations that are related to a specific business entity whose key is passed in the inquiry. The Related Businesses feature is used to manage registration of business units and subsequently relate them based on organizational hierarchies or business partner relationships. Returns a relatedBusinessesList message.
- ***find\_service***: Used to locate specific services within a registered businessEntity. Returns a serviceList message.
- ***find\_tModel***: Used to locate one or more tModel information structures. Returns a tModelList structure.
- ***get\_bindingDetail***: Used to get full bindingTemplate information suitable for making one or more service requests. Returns a bindingDetail message.
- ***get\_businessDetail***: Used to get the full businessEntity information for one or more businesses or organizations. Returns a businessDetail message.
- ***get\_businessDetailExt***: Used to get extended businessEntity information. Returns a businessDetailExt message.
- ***get\_serviceDetail***: Used to get full details for a given set of registered businessService data. Returns a serviceDetail message.
- ***get\_tModelDetail***: Used to get full details for a given set of registered tModel data. Returns a tModelDetail message.

# API de publication

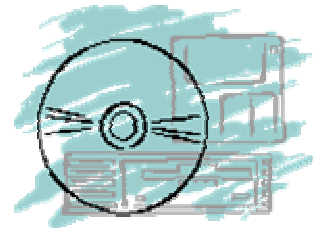


## Tirés directement des spécifications disponibles sur "www.uddi.org"

- **add\_publisherAssertions**: Used to add relationship assertions to the existing set of assertions.
- **delete\_binding**: Used to remove an existing bindingTemplate from the bindingTemplates collection that is part of a specified businessService structure.
- **delete\_business**: Used to delete registered businessEntity information from the registry.
- **delete\_publisherAssertions**: Used to delete specific publisher assertions from the assertion collection controlled by a particular publisher account. Deleting assertions from the assertioncollection will affect the visibility of business relationships. Deleting an assertion will cause any relationships based on that assertion to be invalidated.
- **delete\_service**: Used to delete an existing businessService from the businessServices collection that is part of a specified businessEntity.
- **delete\_tModel**: Used to hide registered information about a tModel. Any tModel hidden in this way is still usable for reference purposes and accessible via the get\_tModelDetail message, but is simply hidden from find\_tModel result sets. There is no way to actually cause a tModel to be deleted, except by administrative petition.
- **discard\_authToken**: Used to inform an Operator Site that a previously provided authentication token is no longer valid and should be considered invalid if used after this message is received and until such time as an authToken value is recycled or reactivated at an operator's discretion. See get\_authToken.



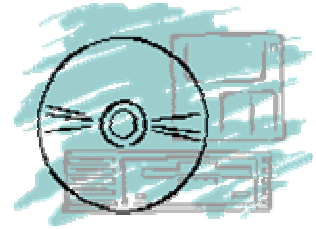
# API de publication



## Tirés directement des spécifications disponibles sur "www.uddi.org"

- **get\_assertionStatusReport**: Used to get a status report containing publisher assertions and status information. This report is useful to help an administrator manage active and tentative publisher assertions. Publisher assertions are used in UDDI to manage publicly visible relationships between businessEntity structures. Relationships are a feature introduced in generic 2.0 that help manage complex business structures that require more than one businessEntity or more than one publisher account to manage parts of a businessEntity. Returns an assertionStatusReport that includes the status of all assertions made involving any businessEntity controlled by the requesting publisher account.
- **get\_authToken**: Used to request an authentication token from an Operator Site. Authentication tokens are required when using all other API's defined in the publishers API. This function serves as the program's equivalent of a login request.
- **get\_publisherAssertions**: Used to get a list of active publisher assertions that are controlled by an individual publisher account. Returns a publisherAssertions message containing all publisher assertions associated with a specific publisher account. Publisher assertions are used to control publicly visible business relationships.
- **get\_registeredInfo**: Used to request an abbreviated synopsis of all information currently managed by a given individual.

# API de publication



## Tirés directement des spécifications disponibles sur "www.uddi.org"

- **save\_binding**: Used to register new bindingTemplate information or update existing bindingTemplate information. Use this to control information about technical capabilities exposed by a registered business.
- **save\_business**: Used to register new businessEntity information or update existing businessEntity information. Use this to control the overall information about the entire business. Of the save\_xx API's this one has the broadest effect. In UDDI V2, a feature is introduced where save\_business can be used to reference a businessService that is parented by another businessEntity.
- **save\_service**: Used to register or update complete information about a businessService exposed by a specified businessEntity.
- **save\_tModel**: Used to register or update complete information about a tModel.
- **set\_publisherAssertions**: (UDDI V2 and later) - used to save the complete set of publisher assertions for an individual publisher account. Replaces any existing assertions, and causes any old assertions that are not reasserted to be removed from the registry. Publisher assertions are used to control publicly visible business relationships.

*La sécurité et les services web*

# Services Web

## La sécurité

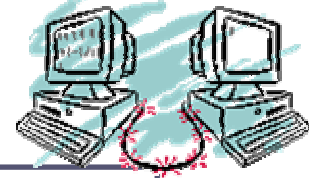
# *La sécurité et les services web*



*Introduction à la sécurité*

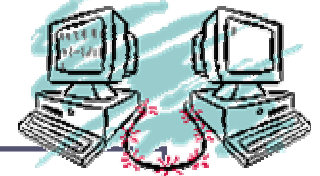
*Les modèles de sécurité des services*

# Introduction à la sécurité



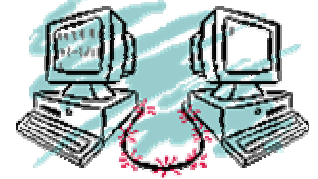
- Comme nous eu tout le loisir de le dire, le concept de services Web s'apparente à d'autres technologies autorisant des interactions en réseau, entre autres des technologies propriétaires telles que DCOM, CORBA et l'EDI, et à des tentatives plus récentes, RosettaNet et ebXML notamment, pour normaliser les interactions e-business sécurisées.
- Les aspects sécuritaires n'échappent pas à cette influence, c'est pourquoi, nous commencerons par présenter les concepts généraux liés à la sécurité.
- Ensuite, nous nous intéresserons aux aspects sécurité spécifiques des services web, en principalement les normes XML dans le domaine.

# Introduction à la sécurité



- Dans le cadre d'une transaction entre entreprises, plusieurs éléments doivent entrer en jeu pour tenter de recréer la confidentialité, l'intégrité, la garantie et la responsabilité d'une transaction entre personnes. Ces éléments sont :
  - ▶ Identification et authentification
  - ▶ Autorisation
  - ▶ Confidentialité et intégrité
  - ▶ Responsabilité
  - ▶ Définition des niveaux de sécurité nécessaires

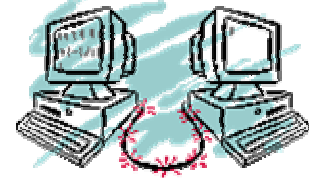
# Introduction à la sécurité



## Identification et authentification

- Il faut tout d'abord confirmer l'identité de l'entité avec laquelle s'établit l'interaction. Lors d'une transaction entre des personnes, l'individu, s'il n'est pas déjà connu du commerçant, peut être recommandé par un tiers de confiance ou justifier de son identité en présentant par exemple son passeport ou son permis de conduire. Dans l'environnement Internet sans visage et ouvert, le mode d'authentification le plus répandu est celui des *certificats numériques* que délivrent et valident des *autorités de certification tierces*.

# Introduction à la sécurité

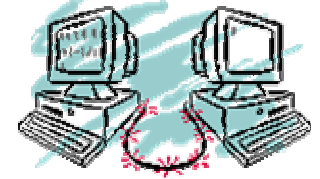


## Autorisation

- Avant d'effectuer la transaction, il est indispensable de savoir si la partie avec laquelle s'établit l'interaction est autorisée à faire ou à accepter l'offre. Dans un scénario entre personnes, ce sont les parties à la transaction ou, le cas échéant, leurs supérieurs hiérarchiques, qui y veillent. Le chef du service des achats peut, par exemple, devoir autoriser une transaction si la facture dépasse un certain montant. Dans le cas des transactions numériques, cette vérification est assurée par un *systeme d'autorisation en ligne*.



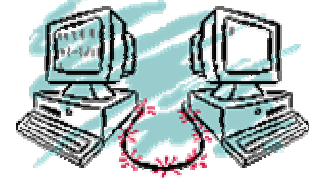
# Introduction à la sécurité



## Confidentialité et intégrité

- Une transaction entre personnes s'effectue généralement dans un environnement privé choisi pour sa discrétion. Les parties prenantes vérifient en outre que tous les documents écrits consignent rigoureusement les conditions convenues lors de la négociation. Dans l'univers numérique, la confidentialité et l'intégrité sont assurées par *le chiffrement et les signatures numériques*. Le canal de communication entre les parties est *chiffré* pour préserver la confidentialité et les documents de la transaction sont signés à l'aide de *certificats d'identité numériques* des parties concernées.

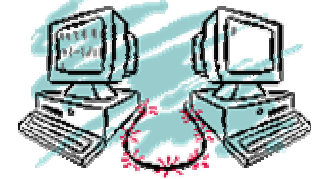
# Introduction à la sécurité



## Définition des niveaux de sécurité nécessaires

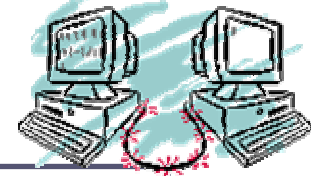
- Ces conditions de sécurité de base sont essentielles pour la bonne intégration aux environnements réseau tels ceux rendus possibles par les services Web. Pour autant, toutes ces conditions ne sont pas systématiquement indispensables. Néanmoins, il est bien plus efficace de partir de l'hypothèse que toutes sont nécessaires, puis de procéder par élimination au moment de la définition des besoins pour un processus métier spécifique que d'essayer d'ajouter des fonctionnalités de sécurité après coup.

# Les modèles de sécurité des services



- Les principaux standards associés à la sécurité des services web sont :
  - ▶ **XML Signature** pour la représentation des signatures digitales dans un document XML
  - ▶ **XML Encryption** pour les mécanismes d'encryptage et de décryptage des documents XML
  - ▶ **SAML-XML** (Security Assertions Markup Language) spécifie une procédure d'authentification
  - ▶ **XACML** (Extensible Access Control Markup Language) spécifie les stratégies de contrôle d'accès à travers le réseau
  - ▶ **XKMS** (XML Key Management Specification) décrit l'infrastructure à clé publique

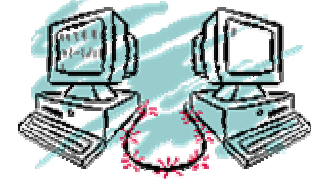
# Les modèles de sécurité des services



## XML Signature (Signature XML)

- En définissant un langage XML de représentation des signatures numériques, XML Signature remplit trois des principales conditions de sécurisation des transactions électroniques :
  - ▶ l'authentification, l'intégrité du message et la non répudiation.
- XML Signature est la seule norme de sécurité des services Web à être à ce jour suffisamment mûre pour un déploiement commercial - les autres normes ne trouveront pas d'utilisation commerciale avant 6 à 18 mois. Une signature XML peut être générée pour tout type de contenu numérique. Pour les documents XML, elle peut même être jointe directement au document « signé » et protège l'intégrité de tout ou partie d'un document XML. XML Signature est devenue une recommandation officielle du W3C en février 2002.

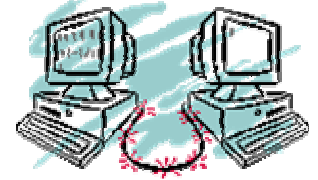
# *Les modèles de sécurité des services*



## **XML Encryption (Chiffrement XML)**

- Technologie complémentaire à XML Signature, le chiffrement XML permet le chiffrement et le déchiffrement sélectifs d'éléments XML ou de documents XML entiers.
- XML Encryption Syntax and Processing (syntaxe et traitement du chiffrement XML), le premier document du groupe de travail sur le chiffrement XML, est « candidat à recommandation » auprès du W3C.

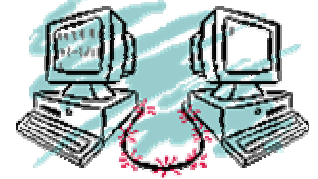
# *Les modèles de sécurité des services*



## **SAML (Security Assertions Markup Language)**

- SAML permet aux entités distribuées d'échanger des informations d'authentification et d'autorisation.
- Conçu pour répondre aux besoins de sécurité des applications en réseau, ce langage présente plusieurs avantages, en évitant par exemple de répéter les procédures d'authentification d'un utilisateur pour chaque application.

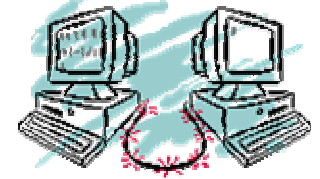
# Les modèles de sécurité des services



## XACML (Extensible Access Control Markup Language)

- OASIS soutient le développement de la norme XACML de contrôle d'accès de forte granularité.
- XACML est censé gérer le contrôle de granularité fine des activités autorisées, l'impact des caractéristiques du demandeur d'accès, le protocole utilisé pour la requête, l'autorisation en fonction des types d'activités et l'inspection du contenu (c'est-à-dire l'autorisation fondée à la fois sur le demandeur et les valeurs d'attribut de la cible lorsque les valeurs des attributs ne sont pas connues de l'auteur des règles).

# Les modèles de sécurité des services



## XKMS (XML Key Management)

- La saisie, la validation et la gestion des certificats à clé publique ou des paires de clés publiques/privées sont des besoins communs à tous les efforts de normalisation énoncés avant.
- Les groupes W3C et IETF soutiennent le groupe de travail XKMS qui travaille sur ces problèmes.
- XKMS et ses méthodologies secondaires X-RSS et X-KISS fournissent une infrastructure pour un service de certification s'appuyant sur Internet et intranet auquel les fournisseurs de services Web peuvent déléguer la complexité d'une infrastructure à clé publique. Ce service prend en charge l'enregistrement, la recherche, la validation, la révocation et, en option, la récupération des paires de clés publiques/privées utilisées pour des transactions e-business certifiées.



*Le modèle .NET*  
*Le modèle J2EE*

# Services Web

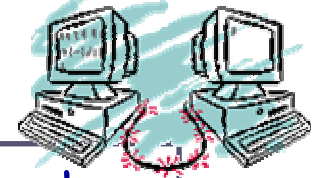
## Les plates-formes

*Le modèle .NET*  
*Le modèle J2EE*



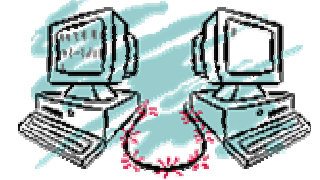
*Présentation*  
*Architecture*  
*Commentaires*

# Présentation



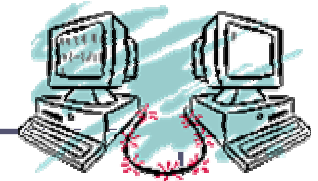
- Les services Web chez Microsoft sont implémentés en utilisant toute les possibilités « .NET ». Ainsi :
  - ▶ Les sessions sont prises en charge par « ASP.NET »
  - ▶ La communication HTTP est géré par IIS
  - ▶ Les infrastructures système comme la montée en charge, les transactions sont traitées par le modèle COM+
  - ▶ Le client peut être un navigateur ou une application Windows (WinForms) et les échanges avec le serveur sont traités par l'implémentation SOAP de Microsoft
  - ▶ L'implémentation et l'intégration de SOAP dans l'environnement natif de Windows a des impacts sur les performances
  - ▶ La base de toute la gestion des Services Web est IIS qui gère l'ensemble des aspects Services Web. Cela va de la gestion des requêtes et réponses jusqu'à la configuration de la sécurité et des autorisations d'accès

# Présentation

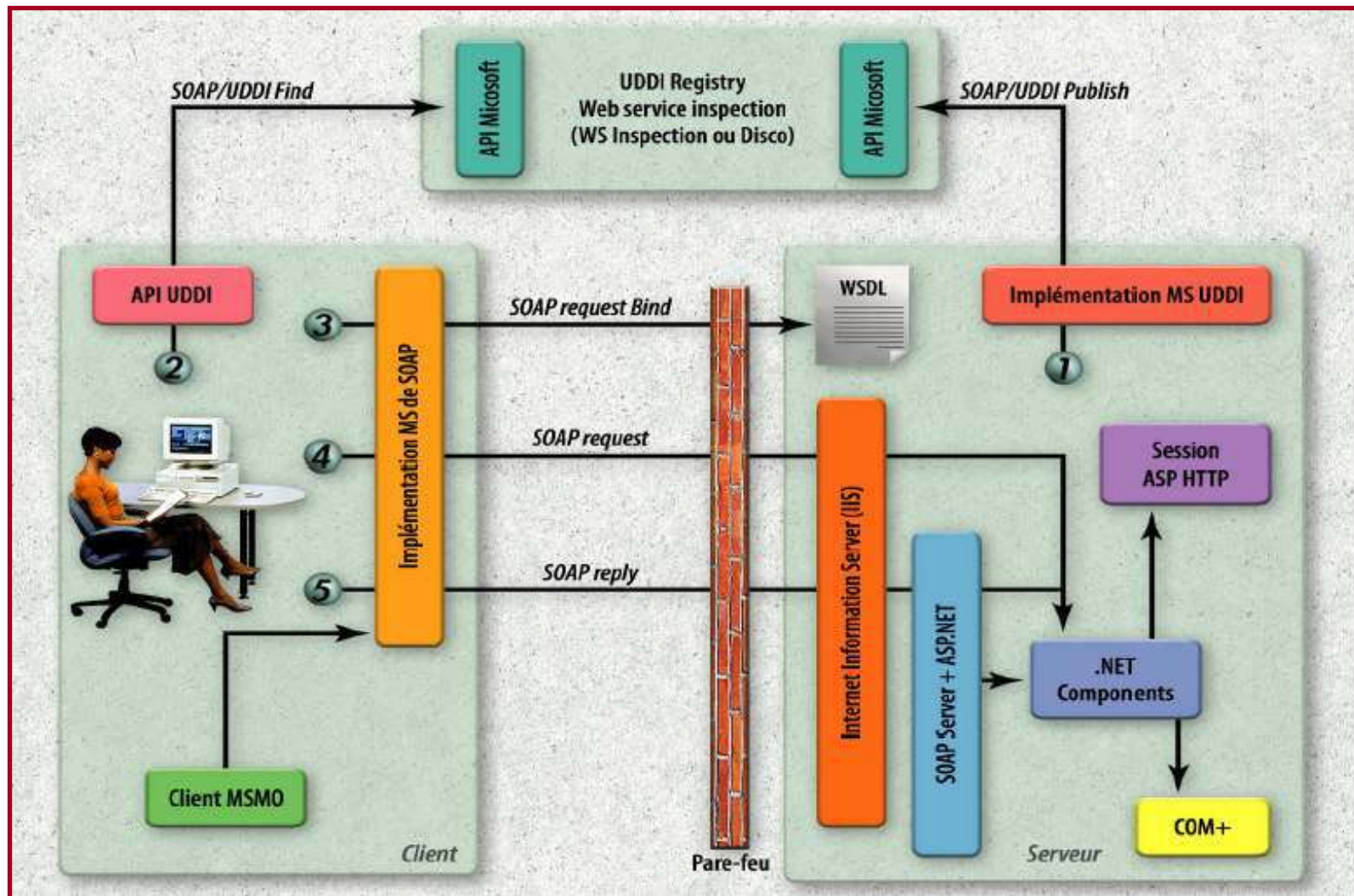


- L'outil Visual Studio permet de créer graphiquement, à l'aide d'une pléthore d'assistants, des Services Web de tout nature.
- La coopération des services web par proxy (par programme en s'appuyant sur des API spécialisées) est prise en charge par l'environnement
- L'implémentation d'un service en C# suit les mêmes principes que pour un autre langage, tel VB.NET ou Cobol.NET

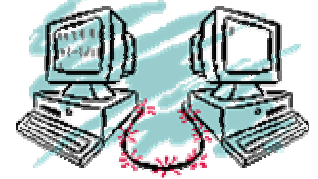
# Architecture



➤ L'architecture des Services Web dans .NET:



# Commentaires



- L'architecture des Services Web dans .NET est totalement intégrée ce qui présente les avantages suivants :
  - ▶ Cohérence et homogénéité
  - ▶ Performances optimales
  - ▶ Intégration des environnements de développement
  
- Les inconvénients sont ceux habituellement lancés à l'encontre du géant Microsoft :
  - ▶ Nécessité d'utiliser la gamme des produits de la firme.
  - ▶ Un bémol cependant, puisque le modèle des Services Web autorise une interopérabilité totale quelque soit les plates-formes et langages.

# *Le modèle .NET*

## *Le modèle J2EE*

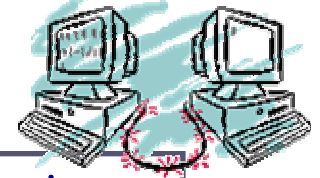


*Présentation*

*Architecture*

*Commentaires*

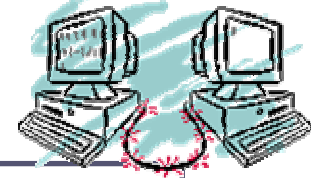
# Présentation



- Les services Web dans la plate-forme J2EE consiste à offrir la gestion interne de l'infrastructure à divers éditeurs (comme à l'usage de Sun)
- Pour ce faire, la plate-forme J2EE a été enrichie des principales API suivantes :
  - ▶ **JAXP**: Java API for XML Processins, est une implémentation complète des standards XML, Namespaces, SAX, DOM et XSLT
  - ▶ **JAXB**: Java Architecture for XML Binding, permet la traduction automatique d'objets Java en XML et inversement
  - ▶ **JAXM**: Java API for XML Messaging, est une implémentation en Java de SOAP 1.1 avec pièces jointes
  - ▶ **JAXRPC**: Java API for XML RPC, est aussi une implémentation en Java de SOAP, mais avec une orientation RPC
  - ▶ **JAXR**: Java API for XML Registries, est une abstraction d'accès et de dialogue avec différents types d'annuaires XML (UDDI, ebXML, ...)

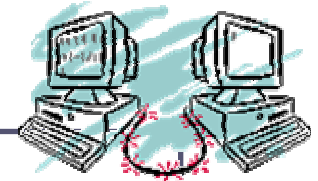


# Présentation

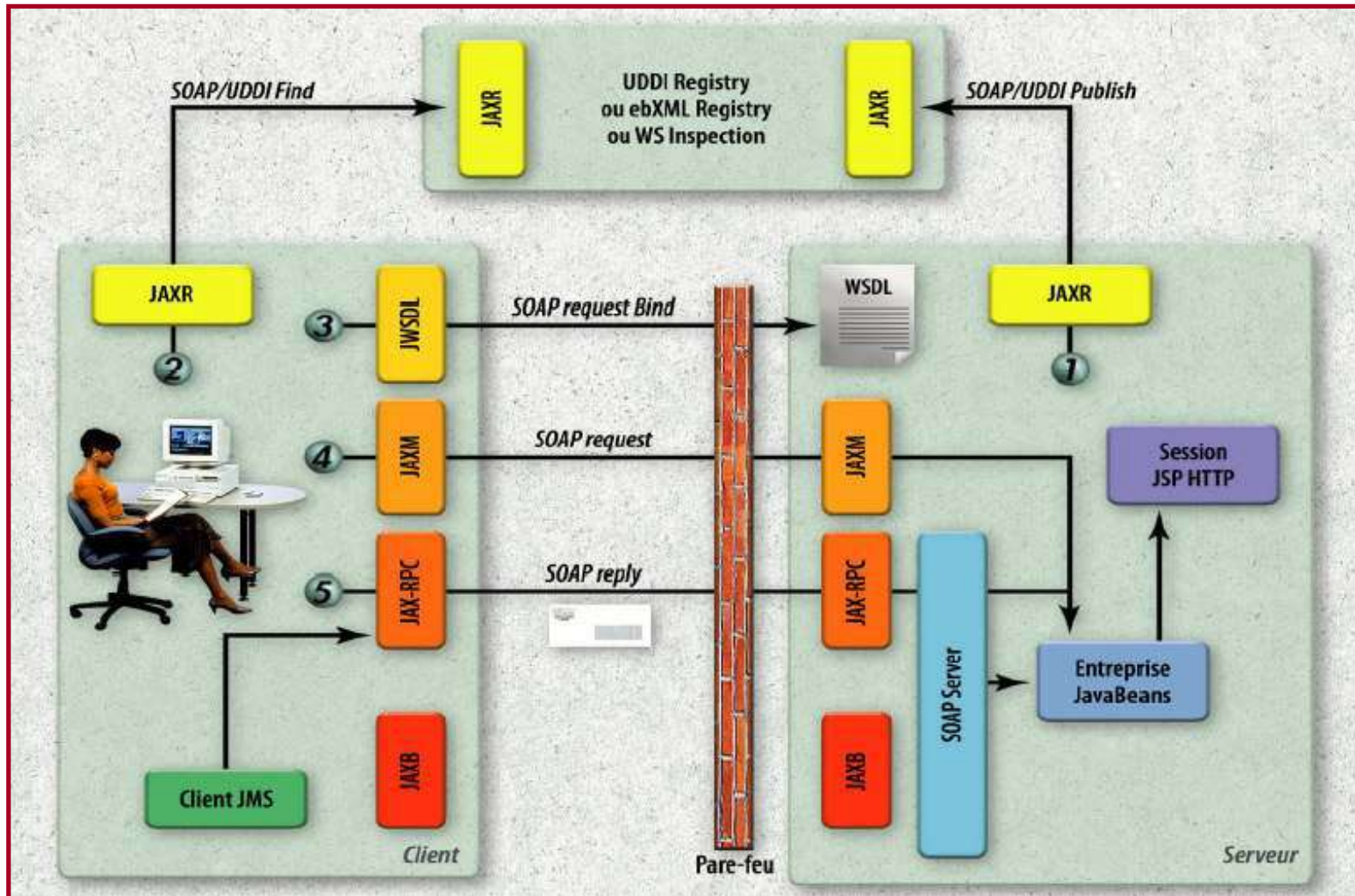


- Plusieurs environnements de développement basés sur la plateforme J2EE, permettent de créer graphiquement, à l'aide d'une pléthore d'assistants, des Services Web de tout nature.
- La coopération des services web par proxy (par programme en s'appuyant sur des API spécialisées) est prise en charge par les environnements, comme pour VisualStudio
- L'implémentation d'un service est en Java, mais peut être obtenue de différentes façons :
  - ▶ En partant de rien
  - ▶ Depuis une description WSDL existante
  - ▶ En définissant un JavaBean
  - ▶ En définissant un EJB Session stateless
  - ▶ En définissant une EJB M-bean

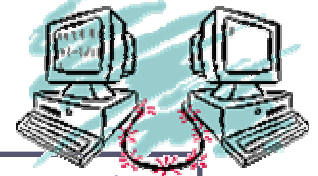
# Architecture



➤ L'architecture des Services Web dans J2EE :



# Commentaires



- L'architecture des Services Web dans J2EE diffère de l'approche Microsoft dans la mesure où elle ne se matérialise pas par une implémentation intégrée dans le système, mais par un ensemble d'abstractions destinées à être implémentées par divers éditeurs.
- Les avantages d'une telle approche sont ceux habituellement énoncés au modèle des abstractions Java, à savoir :
  - ▶ Indépendance plate-forme avec le langage lui-même
  - ▶ Innovation technologique (d'implémentation) puisque l'implémentation est libre ce qui favorise la compétition entre les éditeurs.
- Les principaux éditeurs sont :
  - ▶ IBM avec WebSphere et son environnement WebSphere Studio basé sur Eclipse
  - ▶ BEA avec Weblogic et son environnement Workshop
  - ▶ SUN avec SunOne et son environnement