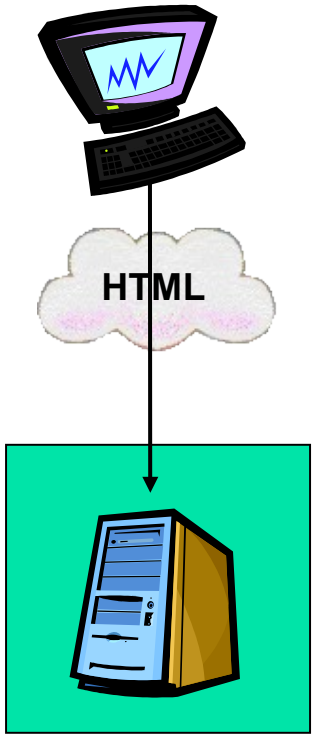
Les services Web vers l'interopérabilité des applications réparties sur internet

Tarek Melliti

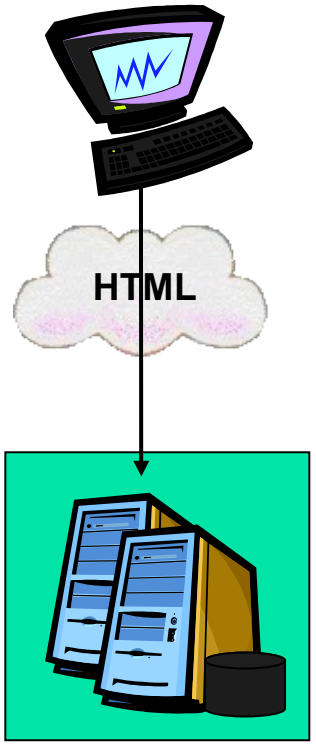
Plan de la présentation

- **Introduction aux services Web**
- **Approche conceptuel orienté service : SOA**
- **Les services Web une implementation de la SOA**
  - **SOAP**
  - **WSDL**
  - **UDDI**
- **Les services Web sous java:**
  - **Les API java pour les services web**
- **Axis**
- **Conclusion du cours**

Le Web hier

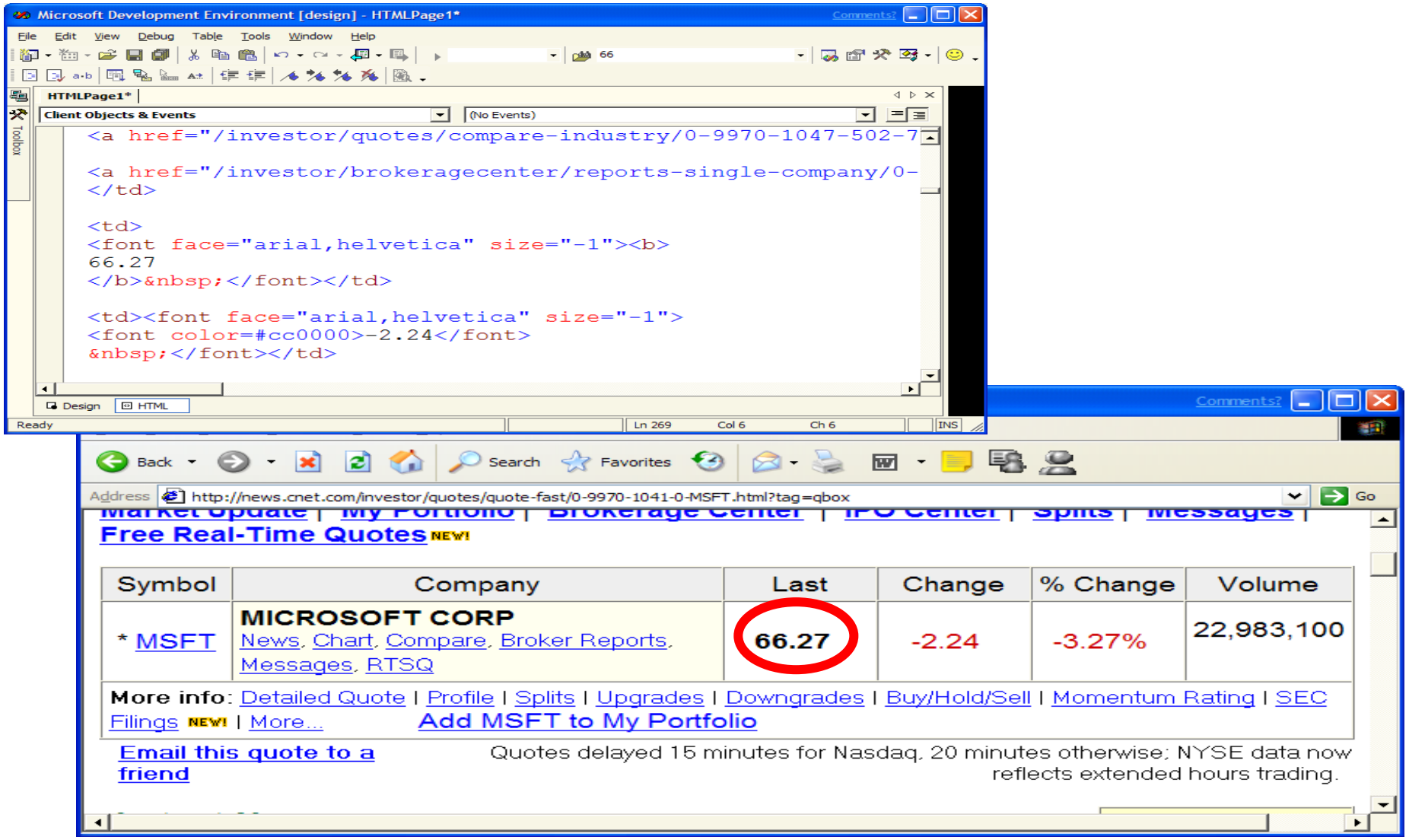


Generation 1
Static HTML



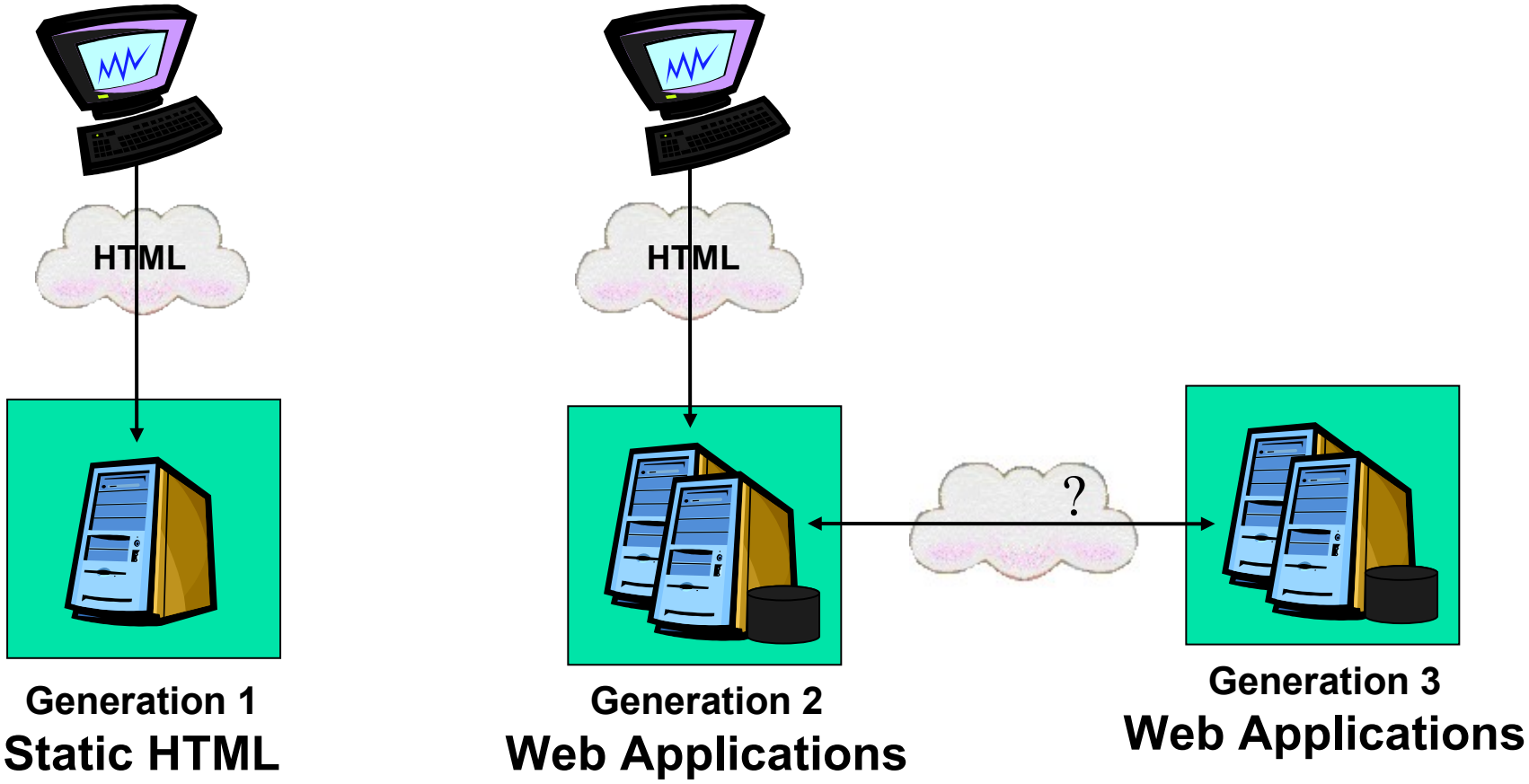
Generation 2
Web Applications

Le Web hier


Microsoft Development Environment [design] - HTMLPage1*
File Edit View Debug Table Tools Window Help
HTMLPage1*
Client Objects & Events
(No Events)
<a href="/investor/quotes/compare-industry/0-9970-1047-502-7
<a href="/investor/brokeragecenter/reports-single-company/0-
</td>
<td>

66.27
 </td>
<td>
-2.24
 </td>
Design HTML
Ready
Ln 269 Col 6 Ch 6
INS
Back Search Favorites
Address http://news.cnet.com/investor/quotes/quote-fast/0-9970-1041-0-MSFT.html?tag=qbox
Go
Free Real-Time Quotes NEW!
Symbol
Company
Last
Change
% Change
Volume
* MSFT
MICROSOFT CORP
News, Chart, Compare, Broker Reports, Messages, RTSQ
66.27
-2.24
-3.27%
22,983,100
More info: Detailed Quote | Profile | Splits | Upgrades | Downgrades | Buy/Hold/Sell | Momentum Rating | SEC Filings NEW! | More...
Add MSFT to My Portfolio
Email this quote to a friend
Quotes delayed 15 minutes for Nasdaq, 20 minutes otherwise; NYSE data now reflects extended hours trading.

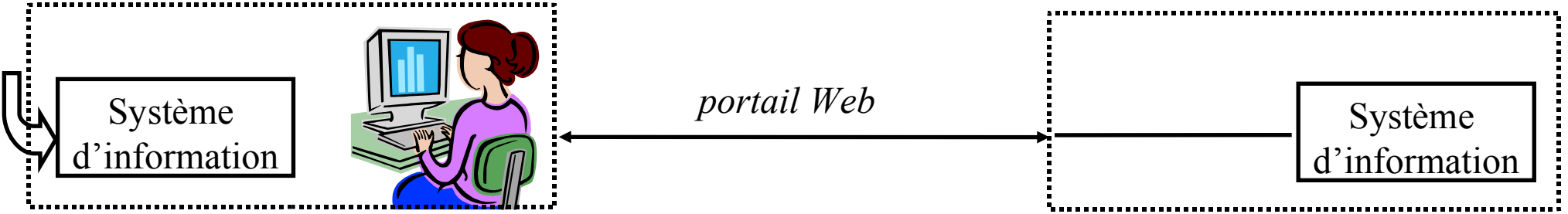
Le Web aujourd'hui



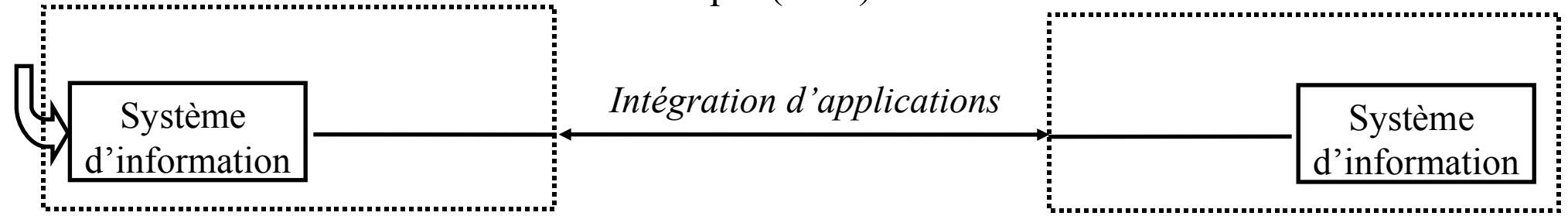
Intégration d'applications

Interaction client/fournisseur « manuelle »

Interaction client/fournisseur semi-automatique (B2C)

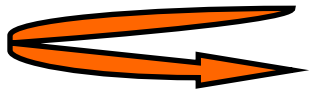


Interaction client/fournisseur automatique (B2B)



Les approches d'intégration

- Les architectures orientées objet
  - Approche conceptuelle par composant (CORBA, DCOM, J2EE)
  - Infrastructure (communication, description, localisation)
  - *mais ...*
    - nécessite des composants fortement couplés
    - rend difficile l'intégration de deux applications en cas d'hétérogénéité des mécanismes de description et/ou de publication
    - ne sont pas adaptés à l'Internet (par exemple ne franchissent pas les pare-feux)
- Les architectures orientées service (SOA)
  - Approche conceptuelle par service : un service est plus stable, plus cohérent, plus fonctionnel, …
  - Infrastructure de communication compatible avec l'Internet



Les services Web

C'est quoi les SOA?

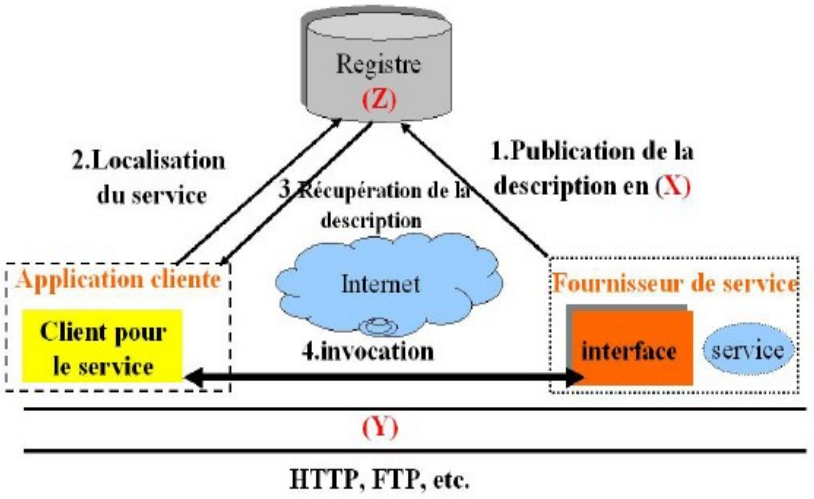
- Définit un modèle d'interaction applicative mettant en oeuvre des connexions en couplage faible entre divers composants logiciels (ou agents)

Definition

L'architecture SOA est un modèle (abstrait) qui définit un système par un ensemble d'agents logiciels distribués qui fonctionnent de concert afin de réaliser une fonctionnalité globale préalablement établie [Hea01].

Les acteurs de la SOA

- Fournisseur de service :
  - Application s'exécutant sur un serveur et comportant un module logiciel invocable de l'exterieur par envoi de message
  - Description de l'interface de l'application suivant une spécification (X)
- Distributeur de service :
  - Annuaire des services publiés par les fournisseurs
  - Géré sur un serveur de niveau application, entreprise ou mondial (destiné à un monde ouvert) (Z)



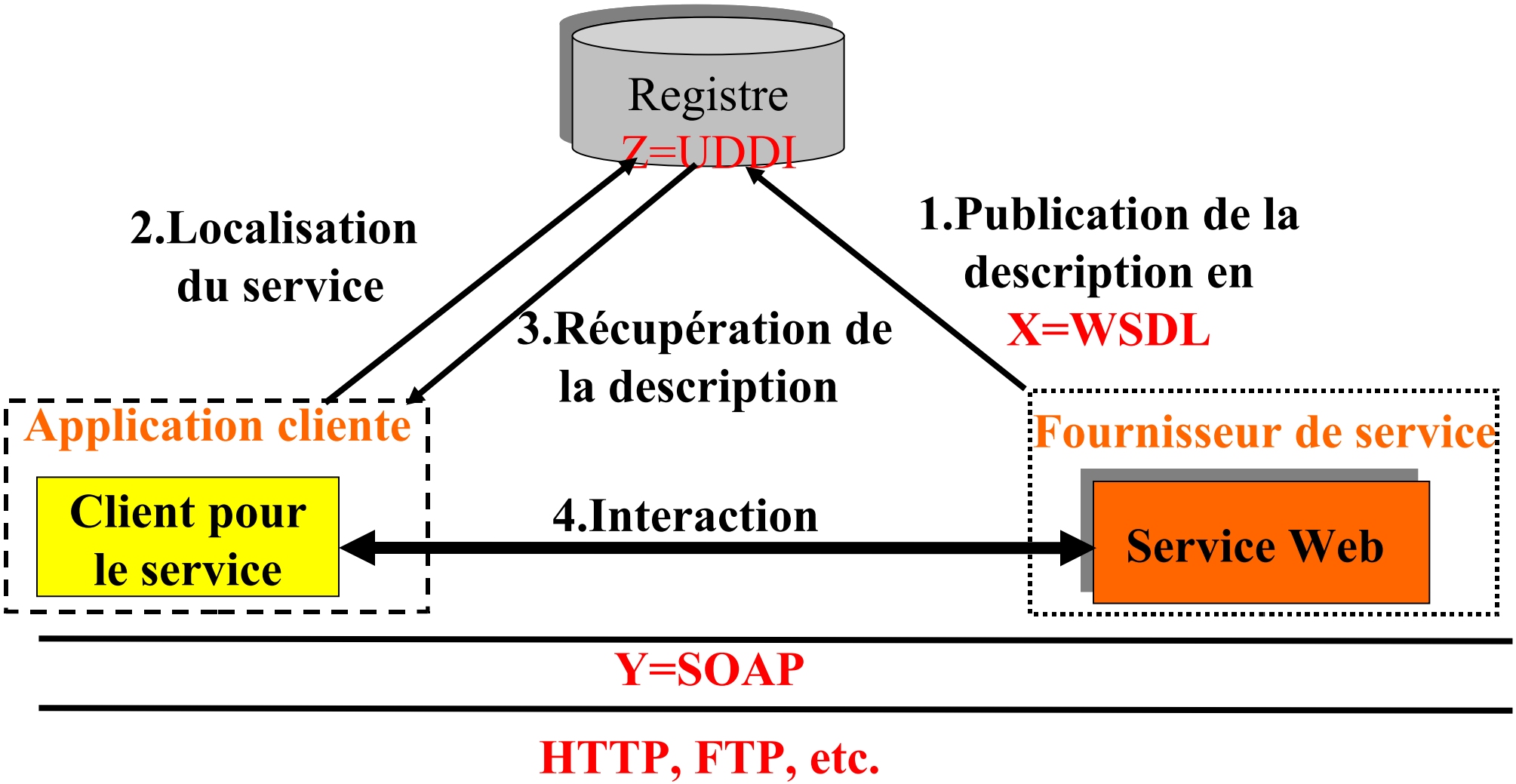
- Demandeur de service :
  - Application cliente désirant un service et invoquant ses fonctions
  - par échange de message suivant un format (Y)

- Définir un cadre technologique aux SOA sur Internet revient alors à définir X, Y et Z
  - support hétérogénéité
  - suffisamment expressive pour supporter le modèle de fonctionnement (auto-suffisant)
  - évolutif
  - = assurer le couplage faible

XML est au coeur des solutions d'intégration d'applications sur le Web.

  - Standardisation de XML (1998) :
  - Portabilité des documents
  - Structuration des documents (XMLSchema)
  - Typage de données (XSD)

Les services Web est une réalisation de la SOA sur Internet



Qu'est quoi un services Web alors?

Definition

Un service Web est une application accessible à partir du Web. Il utilise les protocoles Internet pour communiquer et utilise un language standard pour décrire son interface.

Pourquoi les services Web

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

Pour mieux comprendre les services Web

- Il faut
- Avoir en tête l'architecture SOA
- Et
- Voire en détail les standards
  - Description
  - Publication
  - Protocoles de communication

Le langage de description d'interface

WSDL

C'est quoi??

- C'est un langage de définition des interfaces des services (le contrat)
  - Donc d'une grande importance
- Il représente la définition d'un services Web vue par le fournisseur
- Il doit contenir toutes les information nécessaire au client pour consommer le service (auto-suffisant)
- Il joue exactement le même rôle que IDL sauf qu'il n'exprime pas des objet distant mais un service

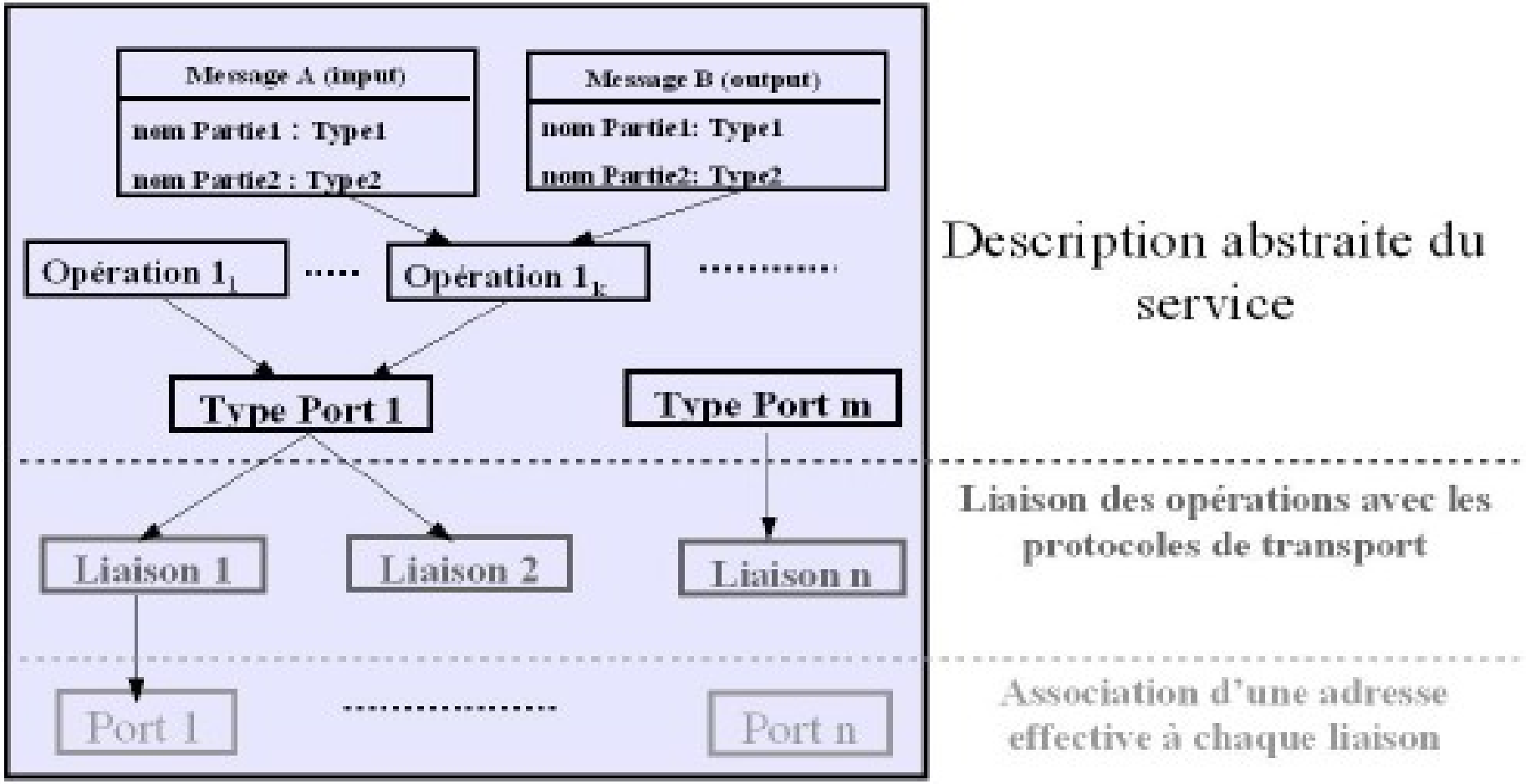
Un services selon WSDL

- Grammaire XML de description d'interface des services
- C'est quoi un service selon WSDL
  - Un ensemble d'opérations
  - Des formats des messages Typé nécessaire à chaque opération
- L'absence d'un environnement unique impose que
  - Un descriptif complet des mécanismes de
  - Formatage des messages
  - Liaison avec les protocoles de transport applicatifs
  - L'adresse d'écoute de traitement des requêtes

Pourquoi XML

- Utilise du texte (peut être lu et écrit directement)
  - ATTENTION : le texte est globalement peut lisible et vite complexe pour un humain
- Possibilité de définir des grammaire (test de validité)
- => structuré et possibilité d'une interprétation automatique
- Construire correctement du texte XML est simple
- XML permet une extensibilité aisée par l'utilisation d'espaces de nommage (namespaces et URIs)
- XML est aujourd'hui adopté par tous les acteurs de l'Internet : plates-formes, éditeurs, ...
- XML permet d'ajouter du typage et de la structure à des informations

Structure d'un fichier WSDL



Un exemple pour comprendre

- Un services de compagnie aérienne:
- Qui permet de
  - rajouter des vols
  - Consulter des vols
- on a besoin de ce que c'est
  - Vol
  - Date
  - Intervale de dates (départ et retour)
  - Liste de vols

Système de typage

- WSDL manipule des données typées
- Pour cela on a choisi le système de typage international comme (IDL sans les interface)
- Il utilise XSD de XML : largement utilisé et un mapping vers des environnement (java, C#) existe (voir votre cours sur XML)
- Toutefois des différences majeur avec IDL :
  - Le sujet de distribution (l'objet) est lui même un type dans IDL
  - Ici le services (sujet de répartition) de répartition n'est pas un type (pas de passage par référence ça n'a pas de sens).
  - Couplage faible ou lâche

L'élément <types>

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

```
- <wsdl:types>
  - <schema targetNamespace="http://vol.samples">
      <import namespace="http://localhost:8080/ch09/services/Employee"/>
      <import namespace="http://lang.java"/>
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
    - <complexType name="date">
      - <sequence>
          <element name="day" type="xsd:int"/>
          <element name="h" type="xsd:int"/>
          <element name="m" type="xsd:int"/>
          <element name="month" type="xsd:int"/>
          <element name="year" type="xsd:int"/>
        </sequence>
      </complexType>
    - <complexType name="Vol">
      - <sequence>
          <element name="ID" nillable="true" type="xsd:string"/>
          <element name="arr" nillable="true" type="xsd:string"/>
          <element name="dateD" nillable="true" type="tns1:date"/>
          <element name="dep" nillable="true" type="xsd:string"/>
          <element name="payed" type="xsd:boolean"/>
          <element name="reserved" type="xsd:boolean"/>
        </sequence>
      </complexType>
    - <complexType name="ArrayOf_Flights">
      - <complexContent>
        - <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:Vol[]"/>
          </restriction>
        </complexContent>
      </complexType>
    </schema>
  </wsdl:types>
```

Les messages

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

```
- <wsdl:message name="addVolRequest">
    <wsdl:part name="in" type="tns1:Vol"/>
  </wsdl:message>
- <wsdl:message name="addVolResponse">
    <wsdl:part name="response" type="soapenc:string"/>
  </wsdl:message>
```

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

- <wsdl:message name="getFlightResponse">
    <wsdl:part name="getFlight" type="tns3:ArrayOf_Flights"/>
  </wsdl:message>
```

WSDL : Les opérations

- Chaque opération est essentiellement définie par ses messages

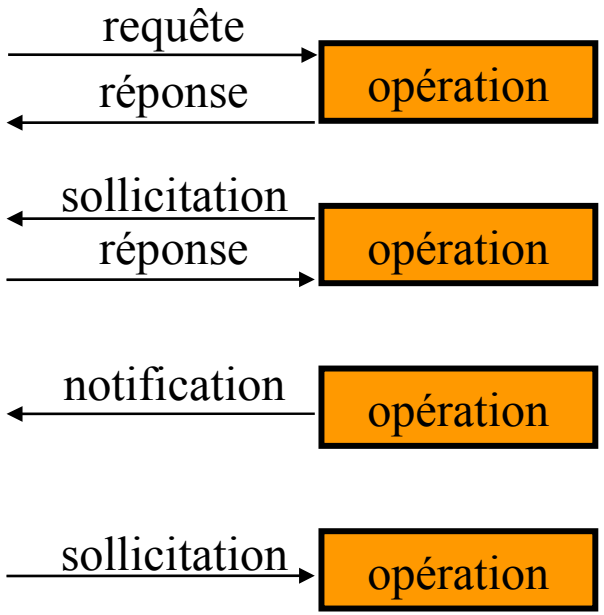
- Quatre types d'opération selon le séquencement et la nature des messages
  - Requête/réponse

  (ex: "Quel prix ?" ; "150€")
  - Sollicitation/réponse

  (ex: "Quel tarif ?" ; "famille nombreuse")
  - Notification (sortie)

  (ex: "Délai dépassé, transaction abandonnée")
  - Sollicitation (entrée)

  (ex: "Annulation de transaction")



Opérations et type de port

- Les <portType> est une façon de regrouper un ensemble d'opération
- Utilisé arbitraire peut être applicatif technique
- Exemple

```
- <wsdl:portType name="VolService">
  - <wsdl:operation name="addVol" parameterOrder="in">
      <wsdl:input message="impl:addVolRequest" name="addVolRequest"/>
      <wsdl:output message="impl:addVolResponse" name="addVolResponse"/>
      <wsdl:fault message="impl:NoSuchVolFault" name="NoSuchVolFault"/>
    </wsdl:operation>
  - <wsdl:operation name="getFlight">
      <wsdl:input message="impl:getFlightRequest" name="getFlightRequest"/>
      <wsdl:output message="impl:getFlightResponse" name="getFlightResponse"/>
      <wsdl:fault message="impl:DBIOException" name="DBIOException"/>
    </wsdl:operation>
  </wsdl:portType>
```

Les détails d'interaction<binding>

- Chaque <portType> peu avoir plusieurs <binding> ou liaison (encore mieux réalisation)
- C'est un élément qui définie les détails techniques nécessaire pour consommer le service:
  - Style de consommation des opérations
  - L'encodage des messages requête & réponse.
- Séparation entre la définition abstraite du service et la manière de le consommer

exemple

```
- <wsdl:binding name="volServicePTSoapBinding" type="impl:VolService">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  - <wsdl:operation name="addVol">
      <wsdlsoap:operation soapAction=""/>
    - <wsdl:input name="addVolRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://vol.samples" use="encoded"/>
      </wsdl:input>
    - <wsdl:output name="addVolResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="http://localhost:8080/axis/services/urn:serviceDemoCPAR" use="encoded"/>
      </wsdl:output>
```

Les adresses des services

- Une collection de points d'entrée (endpoint) relatifs (adresse)
- Chaque «binding» peut être associer a plusieurs adresse
- Exemple :

```
- <wsdl:service name="VolServiceService">
   - <wsdl:port binding="impl:volServicePTSoapBinding" name="volServicePT">
        <wsdlsoap:address location="http://localhost:8080/axis/services/urn:serviceDemoCPAR"/>
     </wsdl:port>
  </wsdl:service>
```

SOAP

Pourquoi un nouveau protocole

- Proposition Web actuelle insuffisante
- Autres plates-formes client / serveur
  - Java RMI
    - mono-langage : Java, multi-plateforme (JVM), SUN
    - Pas réaliste pour une application industrielle (performance, sécurité, …)
  - CORBA / IIOP
    - Multilangage, multi-plateforme, Multi-vendeurs, OMG
    - Installation « coûteuse » si on doit acheter un ORB
      - Mais les open-sources sont gratuits et souvent plus complet
        - www.objectweb.org
  - DCOM
    - multi-langages, plateforme Win32, Propriétaire Microsoft
    - protocole orienté connexion
      - Échange de nombreux paquets pour créer/maintenir une session
    - Faible diffusion
      - Pas disponible sur MacOS, NT3.51, Win95, WinCE2
      - Coûteux sur UNIX, MVS, VMS ou NT

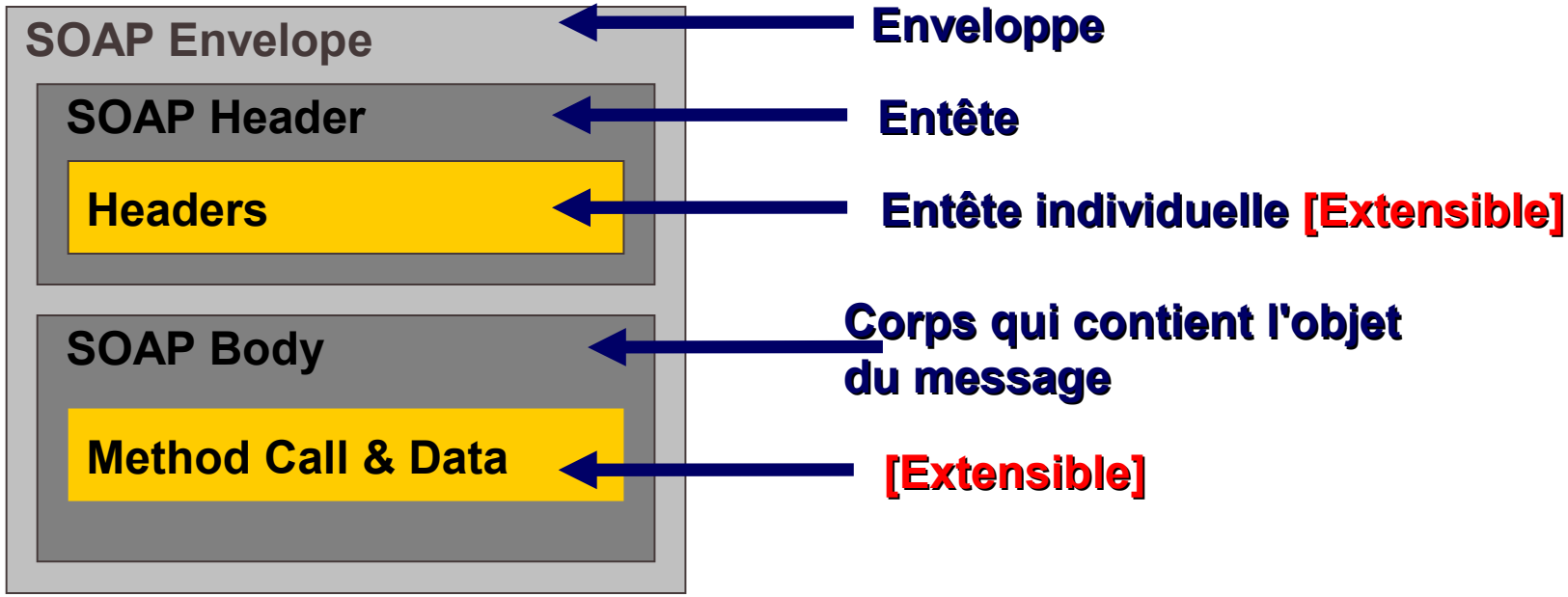
Le besoin

- Le Web a besoin d'un nouveau protocole
  - Multi-langages, multi-plateformes
  - Respectant les formats d'échanges du Web
    - Réponses et requêtes en XML
  - Facile à implémenter sur différents protocoles de transport
    - RPC, HTTP ou autre MOM
  - Permettant de franchir les « firewalls »
    - ATTENTION : on perd le contrôle d'accès à faible granularité
  - Avec une spécification non propriétaire garantie par un organisme indépendant
    - W3C
- La réponse : SOAP (Simple Object Access Protocol)

Simple Object Acess Protocol

- Contrairement à son nom rien d'objet dans SOAP
- C'est un protocole abstrait : spécification XML qui définie un message comme un document XML
- Il n'impose ni un modèle d'échange (eg RPC) ni un mécanisme de transport :
  - peut être transporter par HTTP ou
  - un pigeon ;-)
- Simple car c'est un document XML en cours de transite son interprétation est laisser au couche applicatif

Structure d'un message



Un Message SOAP

- Un message SOAP
- un document XML
- Un mécanisme de Transport
- Un ensemble de convention pour formater & interpréter :
  - Les entêtes spécifique à l'application
  - Les contenue du message

SOAP Extension pour RPC sur HTTP

- L'enveloppe (enveloppe) [existant]
  - Définit la structure du message
- Les règles d'encodage (encoding rules) [extension n°1]
  - Définit le mécanisme de sérialisation permettant de construire le message pour chacun des types de données pouvant être échangés
- Fonctionnement en modèle client / serveur (RPC representation) [extension n°2]
  - Définit comment sont représentés les appels de procédure et les réponses
- Proposer une mise en œuvre sur HTTP (HTTP Extension Framework) [extension n°3]
  - RFC 2774
  - Définir l'échange de message SOAP sur HTTP

Pourquoi HTTP

- HTTP (HyperText Transfer Protocol) est devenu de facto le protocole de communication de l'Internet
- HTTP est disponible sur ***toutes*** les plates-formes – très rapidement
- HTTP est un protocole simple, qui ne requière que peu de support pour fonctionner correctement
- HTTP est un protocole sans connexion
  - Peu de paquets sont nécessaires pour échanger des informations
- HTTP offre un niveau de sécurité simple et effectif
- HTTP est le seul protocole utilisable à travers des pare-feu

Pourquoi RPC

- C'est un modèle déjà connue
- Il est très adapté au services Web
- Comment?
  - Voir un service comme une abstraction d'une procédure.
  - Consommer un services revient à consommer une procédure
    - Nom du services => nom de la procédure
    - Requête => les paramètre + nom du services
    - Réponse => retour de la procédure
- Mais ATTENTION rien a voir avec le modèle d'implémentation du service

1. Encodage SOAP

- C'est une façon de réaliser un transfert de donnée typé
- C'est un ensemble de convention qui permette d'envoyer des paramètres typés
- Une règle par construction XSD (le système de typage XML)
- Offre aux consommateurs de services une façon d'encoder les paramètres des appels
- Aux fournisseur une garantie d'une interprétation ou reconstitution des paramètres.

1. Les règles de l'encodage (1)

- **Types primitifs**

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

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

- **Structures**

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

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

- **Enumération**

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

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

Les règles d'encodage (2)

- **Références**

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

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

2. Convention RPC (requête)

- Simple convention qui impose que si SOAP est une requête pour un services alors le corps de message SOAP doit être

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-
  org:soap.v1”>
    <SOAP:Body>
        <nm:NomDeOPeration
          xmlns:nm="Some-Namespace-URI”>
            encodage des parametres entrées
        </nm:NomDeOPeration>
    </SOAP:Body>
</SOAP:Envelope>
```

exemple

- Notre services addvol

```
<?xml version="1.0" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <tns1:addVol xmlns:tns1="http://vol.samples"
          xmlns:impl="http://localhost:8080/axis/services/urn:serviceDemoCPAR"
          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" ....>
            <in>
                <ID>12</ID>
                <arr>tunis</arr>
                <dateD>
                    <day>12</day>
                    <h>14</h>
                    <m>00</m>
                    <month>12</month>
                    <year>2007</year>
                </dateD>
                <dep>Paris</dep>
                <payed>false</payed>
                <reserved>false</reserved>
            </in>
        </tns1:addVol>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Convention RPC (réponse)

- Simple convention qui impose que si SOAP est une réponse (ou exception) pour un services alors le corps de message SOAP doit être

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-
  org:soap.v1”>
    <SOAP:Body>
        <nm:NomDeOPeration
          xmlns:nm="Some-Namespace-URI”>
            encodage du  retour de l'opération
        </nm:NomDeOPeration>
    </SOAP:Body>
</SOAP:Envelope>
```

Exemple

- La réponse a notre requête

```
<?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <soapenv:Body>
    - <ns1:addVolResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://vol.samples">
        <response xsi:type="xsd:string">ok</response>
      </ns1:addVolResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Convention RPC (erreur)

- Simple convention qui impose que si SOAP est une réponse (ou exception) pour un services alors le corps de message SOAP doit être

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

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

exemple

```
<in>
    <ID>1</ID>
    <arr>Paris</arr>
    <dateD>
        <day>12</day>
        <h>12</h>
        <m>12</m>
        <month>janvier</month>
        <year>2008</year>
    </dateD>
    <dep>tunis</dep>
    <payed>false</payed>
    <reserved/>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <soapenv:Body>
    - <soapenv:Fault>
        <faultcode>soapenv:Server.userException</faultcode>
        <faultstring>java.lang.NumberFormatException: For input string: "janvier"</faultstring>
      - <detail>
          <ns1:hostname xmlns:ns1="http://xml.apache.org/axis/">Melliti</ns1:hostname>
        </detail>
      </soapenv:Fault>
    </soapenv:Body>
  </soapenv:Envelope>
```

SOAP sur HTTP

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

Exemple les SOAP RPC/HTTP

- En cours de transite un SOAP/HTTP est :

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

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

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

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

En résumé ...

- Les services Web sont bâtis au-dessus de standards
  - de description (XML)
  - de transport applicatif (SOAP)
- Un service Web (WSDL) est une collection
  - de spécification d'opérations
  - de moyens de les invoquer
- Un service Web est référencé dans un annuaire (UDDI)
  - incluant des aspects opérationnels
  - et des aspects sémantiques

Déploiement et Appel aux web services Java avec Axis

Les Services Web sous java

- Besoin de quoi?
  - Axis 1.4
  - Apache Tomcat 5.X.
- Les API JAVA: livré avec axis
  - Service Description (WSDL)
    - JSR 110 (Java API for WSDL)
    - JAX-RPC (Java ™ API for XML-based RPC)
  - Service Registration and Discovery (UDDI, ebXML Reg/Rep)
    - JAXR (Java ™ API for XML Registry)
  - Service Invocation (SOAP, ebXML Message Service)
    - JAXM (Java ™ API for XML Messaging), JAX-RPC

Installation de tomcat & axis

- Installer tomcat et configurer votre serveur d'application (choisissez la bonne JVM)
- Pour axis
  - Décompressez le répertoir
  - Copier le répertoire axis sous Webapps vers %tomcat%\Webapps\
  - Copier l'ensemble des jar sous lib vers %tomcat%\common\lib

Les étapes

- Puisque java est un langage Objet on va utiliser les objets pour implémenter le service
- 1. écrivez l'interface de votre services.
- 2. génération de la description WSDL
  - Spécfication des espaces de nom et de l'adresse (endpoint)
  - La comande est:

```
java org.apache.axis.wsdl.Java2WSDL -o <nomfichier>.wsdl
  -l "http://localhost:8080/axis/.../..." -n
  "urn:nomespacedeNom" pakages.votreInterface
```

- 3. Génération des skeleton les stub et le container de l'implémentation
  - `java org.apache.axis.wsdl.WSDL2Java -o . -d Session -s -S -p <votrePackage> true <nomfichier>.wsdl`

Les classes générées

- Résultat :
  - <interface>SoapBindingImpl.java : une implmentation vide donc a remplir par le code de votre service
  - <Interface>.java: une version de votre interface qui hérite des interface necessaire “java.rmi.Remote” .
  - <Interface>Service.java: fichier java contenant l'interface du services coté client
  - <Interface>ServiceLocator.java: fichier Java contenant l'implmentation du service coté Client.
  - <Interface>SoapBindingSkeleton.java: skeleton une servelet.
  - <Interface>SoapBindingStub.java: stub coté client.
  - deploy.wsdd: un descripteur de déploiement
  - undeploy.wsdd: un descripteur de (un)déploiment
  - (data types): des classe Java implementant les type complexes manipulé (ça sera des beens) .

Les étapes

- Déploiement :
  - Copier vos classe %tomcat%\webapps\axis\WEB-INF\classes
  - Lancez tomcat
- java org.apache.axis.client.AdminClient deploy.wsdd
- Voilà ;-) le service est prêt à être consommer

Client

- Coté client vous avez seulement WSDL
- Compilez le fichier WSDL avec :
  - java org.apache.axis.wsdl.WSDL2Java -p <votrepackage>
  - Attention vous n'êtes pas obliger de suivre la même arborescence du serveur
- Ecrivez une classe qui obtient un stub puis consommez le service

Code client

```
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import  votre packageclient;
public class Client {
public static void main(String[] args) {
// Création du service depuis le endpoint
// SommerService correspond au nom du service dans le fichier
"wsdl"
// c'est la balise : wsdl:service name="sommerService"
<interface>Service service = new <interface>ServiceLocator();
try {
// Utilisation du service pour obtenir un stub qui implemente
le SDI
// (Service Definition Type ; i.e. PortType).
// Pour le typage, c'est la balise : wsdl:portType
//name="<interface>Portype"
// Pour le getsommer(), le sommer correspond à la balise :
// wsdl:port binding="impl:sommerSoapBinding" name="sommer"
Sommer port = service.ge<interface>portType();

try {
// Mise en oeuvre du service par application directe des
méthodes
port.<operation>(params);

} catch (RemoteException e1) {
e1.printStackTrace();
}
} catch (ServiceException e) {
e.printStackTrace();
}}
}
```

Remarque importante

- Bien que c'est des objets qui implémente le service ce n'est pas un objet distribué
- La diffrence c'est quoi???
  - Chaque invocation d'une opération va provoquer la création d'une nouvelle instance de votre classe servante
  - Contrairement aux objetx distribués où on distribue une instance de la classe.
  - Donc pas le même cycle de vie.

Merci de votre attention

fin du cours