

Les Services Web SOAP

Abdelhak-Djamel Serial

serial@lirmm.fr

Basé sur les transparents du cours de Clémentine Nebut

Plan

- Définitions et généralités
- Architecture et technologies au cœur des services web
 - SOAP
 - WSDL
 - UDDI
- Services web et sécurité
- Les services web avec .NET et Java
- Les services REST (representational state transfer)

Définition du W3C

« A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols. »

<http://www.org/TR/ws-gloss/>

« Un service web est un système logiciel identifié par une **URI**, dont les interfaces publiques et les **associations** sont définies et décrites en XML. Sa définition peut être découverte par d'autres systèmes logiciels. Ces systèmes peuvent alors interagir avec le service web selon les modalités indiquées dans sa définition, en utilisant des messages XML transmis par des protocoles internet. »

Vocabulaire -1-

- URI : Uniform Resource Identifier
 - Une URI décrit :
 - Le mécanisme pour accéder à la ressource
 - La machine où se trouve la ressource
 - Le nom de la ressource sur la machine
 - EX:
 - URL : <http://www.w3.org>
 - mail : <mailto:toto@domain.com>
 - FTP : <ftp://ftp.computer.org>.

- Association (binding)
 - Association entre une interface, un protocole concret, et un format de données
 - Spécification du protocole et du format des données utilisés pour échanger des messages en vue d'une utilisation d'une interface

Définition opérationnelle

- W3C WS Description WG (mai 2003)
 - “A **resource** [on the Web] offers one or more interfaces. An **interface** is a collection of operations. An **operation** is an exchange of messages between the service provider and the requestor.
 - An **endpoint** is a binding of an interface to a particular protocol. It is identified by a URI.
 - A **service** is a collection of endpoints bound to the same interface, and therefore to the same resource.”

En plus simple ...

- Un service web est
 - un programme décrit en XML et identifié par une URI
 - Proposant des fonctionnalités que d'autres programmes peuvent
 - découvrir
 - et utiliser grâce à des protocoles décrits en XML, et basés sur l'échange de messages décrits en XML et transmis via des protocoles internet.

But des services web

- Remplacer les protocoles actuels (RPC, DCOM, RMI, ...)
- Faire interagir des composants hétérogènes, distants, et indépendants avec un protocole standard (SOAP).
- Dédiés aux applications B2B, EAI (Enterprise Application Integration), P2P.

L'exemple de l'agence de voyage

- Une agence de voyage web combine plusieurs services :
 - Réservation de billets d'avion, de train
 - Réservation de logement
 - Réservation de véhicules de location
- Et donc utilise les services proposés par la SNCF, Air France, RentACar, ...

Services web et applis distribuées

- Les applications distribuées ont évolué ...
 - Protocoles d'échange (RMI, CORBA, DCOM, ...)
 - Langages
 - Interfaces
- Applications très dépendantes des technologies
- Couplage assez fort

Services web et applis distribuées

- Le web impose :
 - des protocoles de communication légers avec services plus faibles (http, ftp, ...)
 - Un client léger (le navigateur)
 - Présentation html
- Couplage faible

Services web et applis distribuées

- Services web = combiner les caractéristiques des applis distribuées (à la corba) avec les contraintes du web
 - Transport assuré par http sur TCP/IP
 - Modèle client-serveur
 - Messages échangés en XML (http transporte du texte)
- Passage du web-client au web-machines

Quand utiliser des services web ?

- Besoin d'interopérabilité dans des environnements applicatifs distribués
- Accès à des applications à travers de pare-feux
- Possibilité d'utiliser différentes plateformes et différents langages

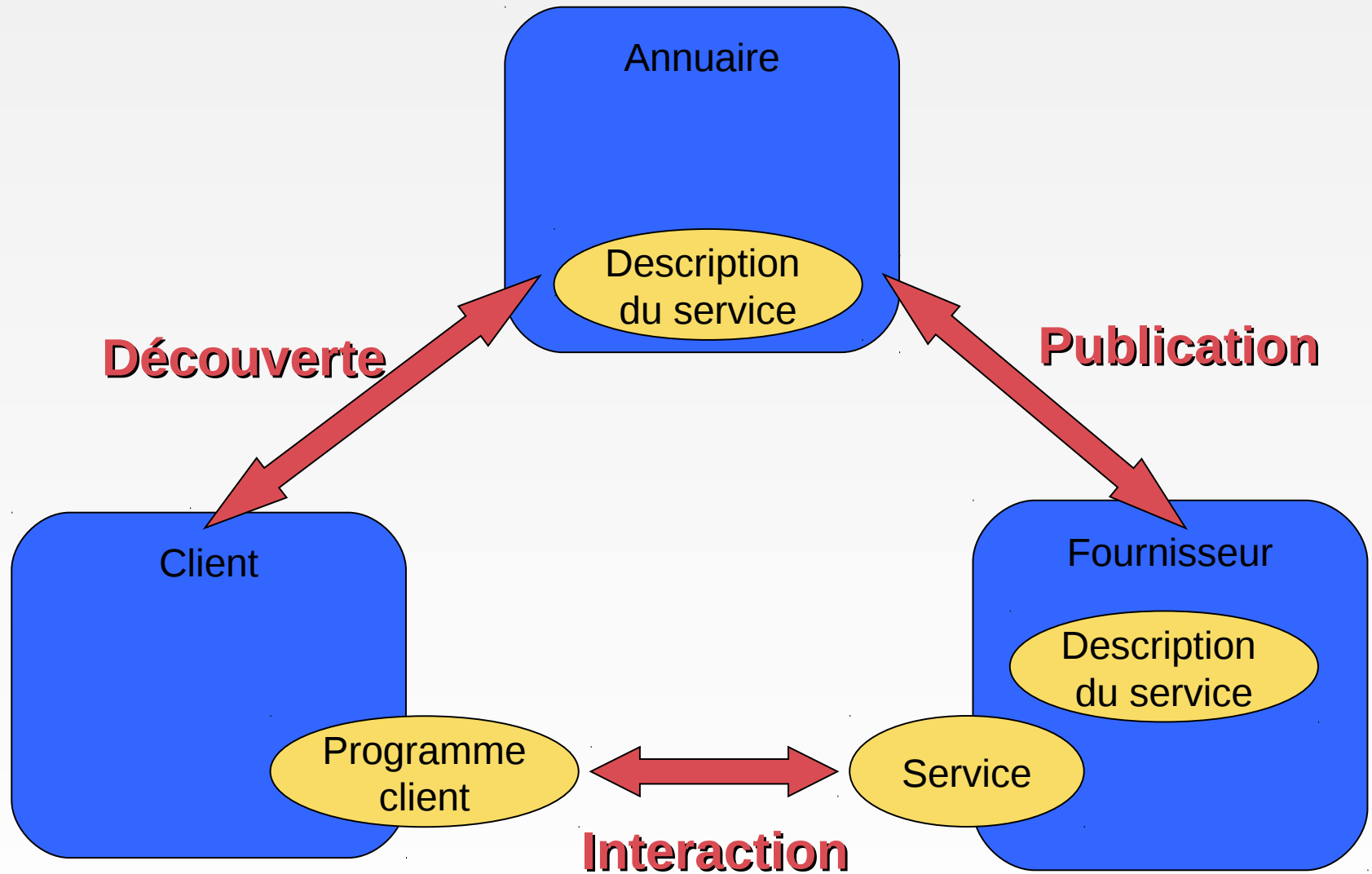
Utilisations classiques

- Applications bien délimitées et sans forte interactivité
 - Service météo, cotations boursières, google, ...
- Assemblage de composants faiblement couplés
 - Agence de voyage
- Applications orientées message

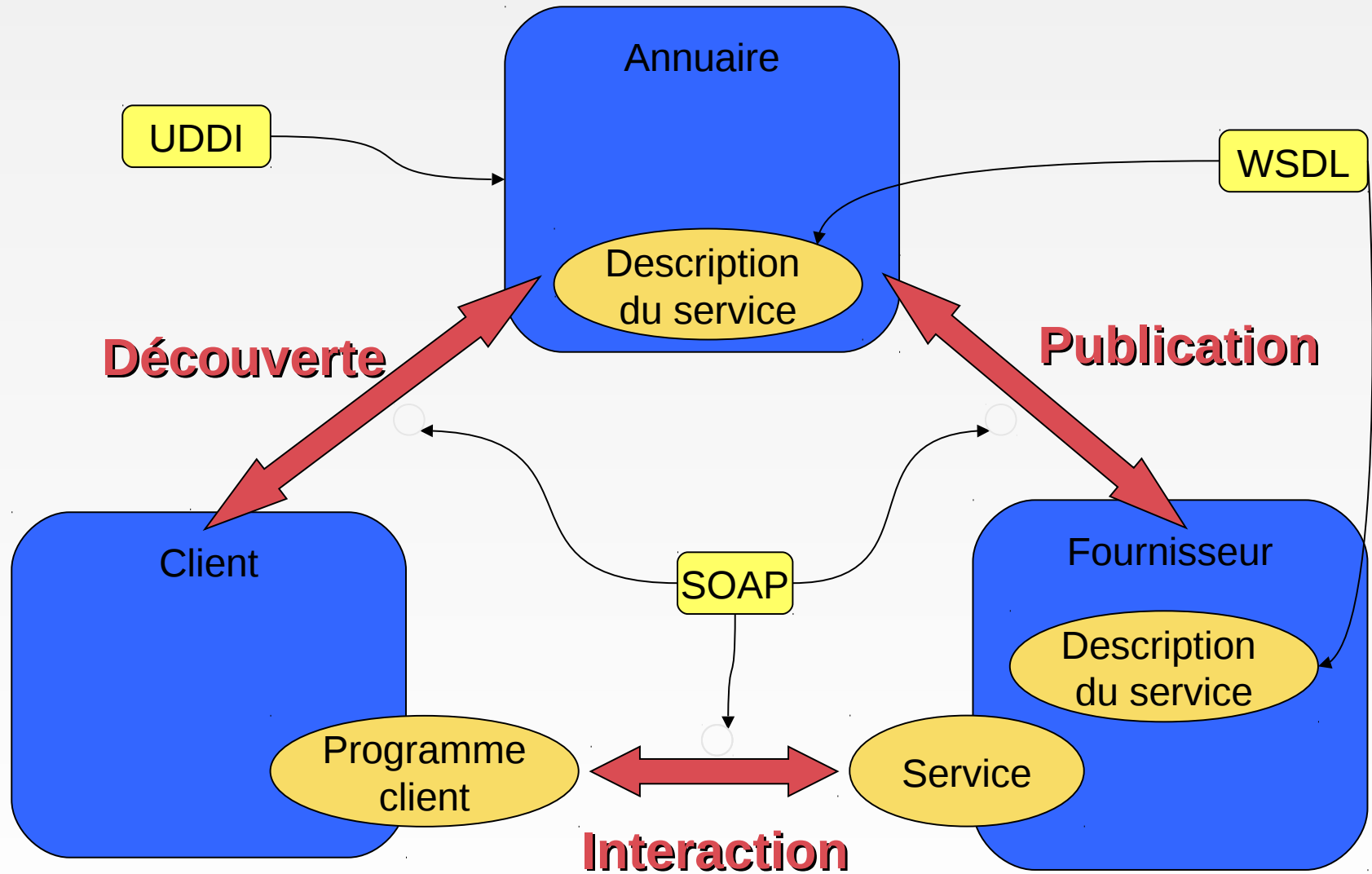
Acteurs impliqués

- Le client
 - Utilise, invoque des services web
- Le fournisseur
 - Fournit le service web
 - Est représenté par un serveur d'applications
- L'annuaire
 - Pour publier le service, pour le rendre accessible aux clients

Architecture de base



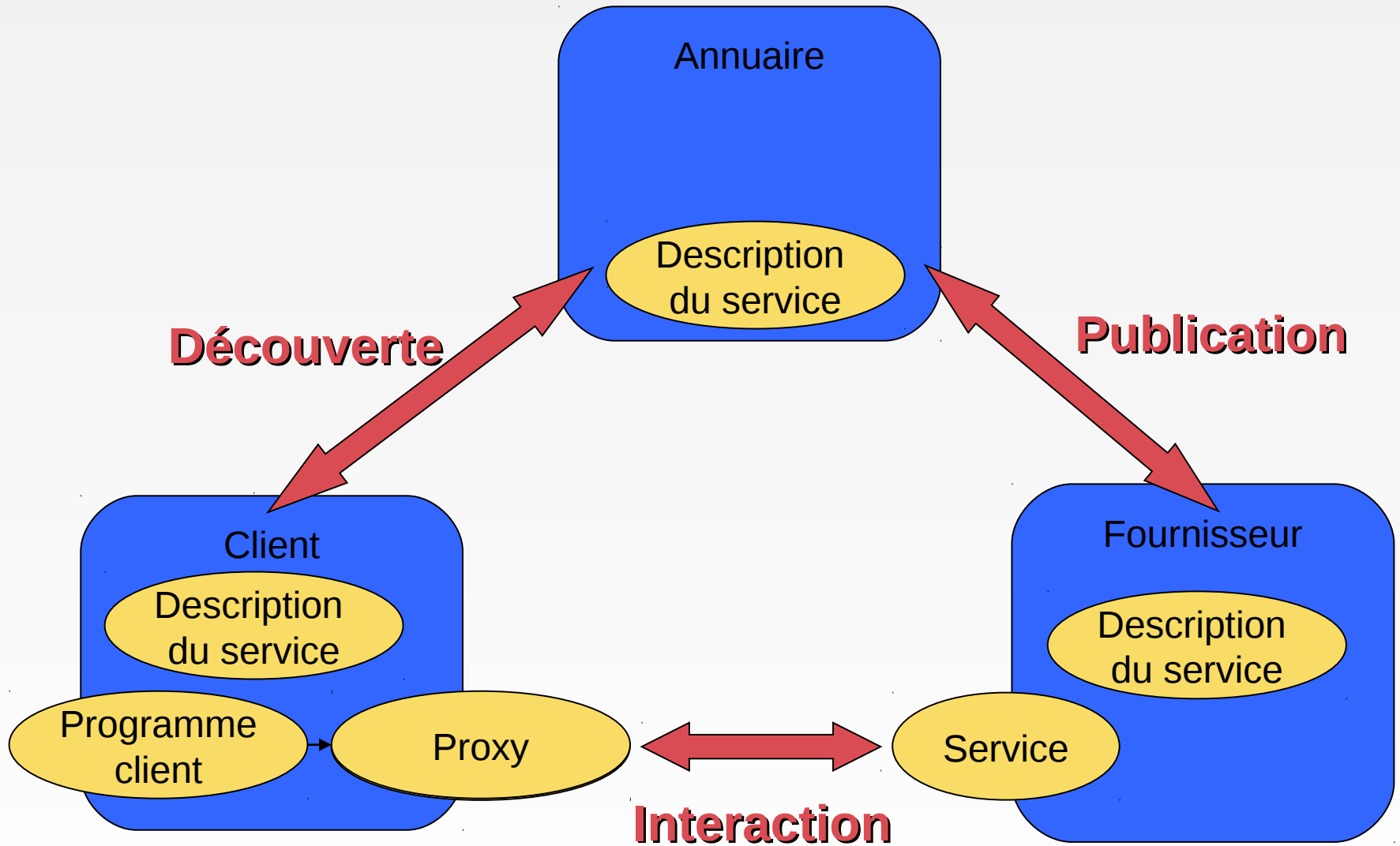
Technologies



Technologies

- WSDL
 - Web Service Description Language
 - Description des services
- SOAP
 - ~~Simple Object Access Protocol~~
 - Protocole de communication des services web
- UDDI
 - Universal Data Description Language
 - Annuaire de WS

Où l'on retrouve les proxies ...



L'intérêt des outils

- WSDL et SOAP sont des dialectes XML
- Il serait très long
 - de transcrire à la main le WSDL en un proxy
 - de générer à la main les requêtes SOAP nécessaires
 - de décortiquer à la main les messages SOAP
- Les outils s'en chargent ...

D'où vient l'interopérabilité ?

- De la standardisation ...
 - ce qui est échangé : du WSDL, standard W3C
 - comment c'est échangé : SOAP, standard W3C
- Donc
 - un WS C# peut très bien interopérer avec un WS java
 - et réciproquement

SOAP

SOAP, c'est quoi ?

- Codification d'une pratique existante
 - Utilisation conjointe de XML et HTTP
- Protocole minimal pour appeler des méthodes sur des serveurs, services, composants, objets
 - sans imposer un langage dédié
 - sans l'utilisation d'un ORB (CORBA, DCOM, ...) ou d'un serveur web particulier (Apache, IIS, ...)
 - sans imposer un modèle de programmation
 - sans réinventer la poudre
- Objectif : portage facile sur toutes les plates-formes et les technologies
- SOAP n'est plus l'acronyme de Simple Object Access Protocol

Les 3 facettes d'une requête SOAP

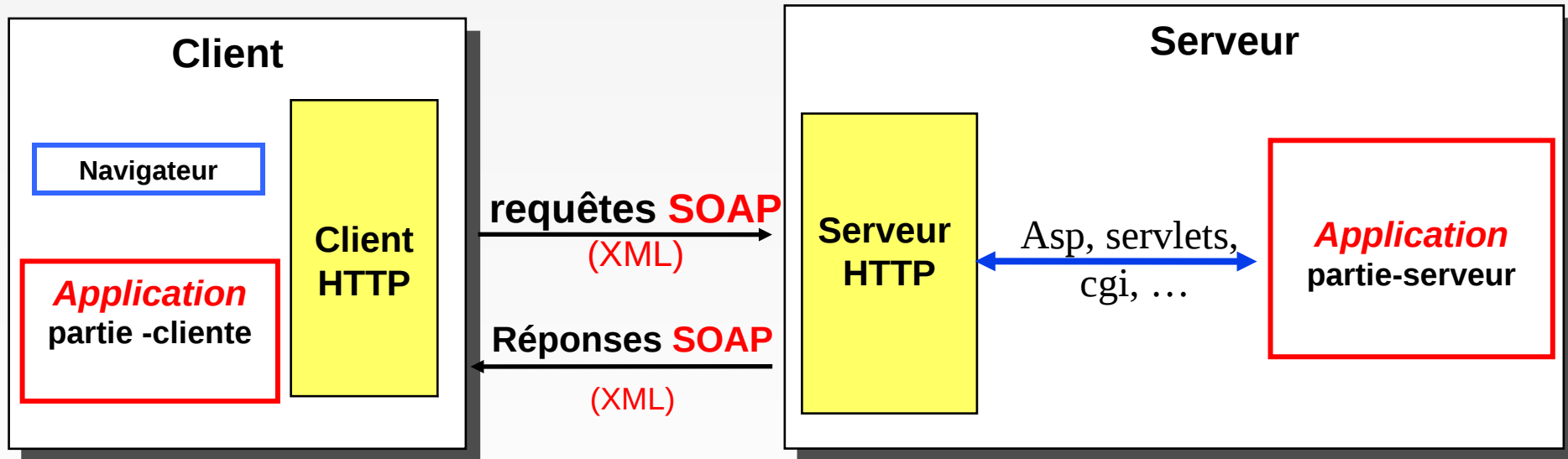
- un autre RPC Objets
 - Les requêtes contiennent les paramètres IN et INOUT
 - Les réponses contiennent les paramètres INOUT et OUT
- un protocole d'échange de "messages"
 - La requête contient un seul message (appel sérialisé d'une méthode sur un objet)
 - La réponse contient un seul message (retour sérialisé d'un appel de méthode sur un objet)
- un format d'échange de documents
 - La requête contient un document XML
 - Le serveur retourne une version transformée

Comment ça marche ? (en gros)

- Encapsulation d'un service dans une méthode (java par exemple)
- Démarrage d'un thread qui écoute les requêtes adressées à ce service
 - requêtes adressées dans le format SOAP, contiennent le nom du service et les paramètres requis).
 - ex : servlet java tournant dans Tomcat
- Le thread qui écoute décode la requête SOAP et la transforme en appel de méthode, récupère le résultat de l'appel et le transmet au demandeur

En résumé

- SOAP = HTTP + XML



Les messages SOAP

- Un message SOAP est un document XML qui contient :
 - Une déclaration XML facultative
 - Une enveloppe SOAP composée d' :
 - Une en-tête SOAP = HEADER
 - Un corps SOAP = BODY



Exemple

- StockQuote est un ensemble de services qui permet d'obtenir des informations sur des actions boursières.
 - GetLastTradePrice est le service qui permet de connaître la dernière valeur d'une action.
- Messages pour avoir la réponse à la question : Quelle est la valeur de l'action « DIS » ?

Exemple de requête

POST /StockQuote HTTP/1.1

Host: **www.stockquoteserver.com**

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

Content-Length: nnnn

SOAPAction: "Some-URI"

Propre au
portage sur
HTTP

Entête HTTP spécifique pour
qu'un serveur HTTP puisse
reconnaître la requête SOAP

<SOAP-ENV:Envelope

xmlns:SOAP-ENV=

"http://schemas.xmlsoap.org/soap/envelope/"

SOAP-ENV:encodingStyle=

"http://schemas.xmlsoap.org/soap/encoding/">

<SOAP-ENV:Body>

<m:GetLastTradePrice xmlns:m="Some-URI">

<symbol>DIS</symbol>

</m:GetLastTradePrice>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

namespace utilisateur

Exemple de réponse

HTTP/1.1 200 OK

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

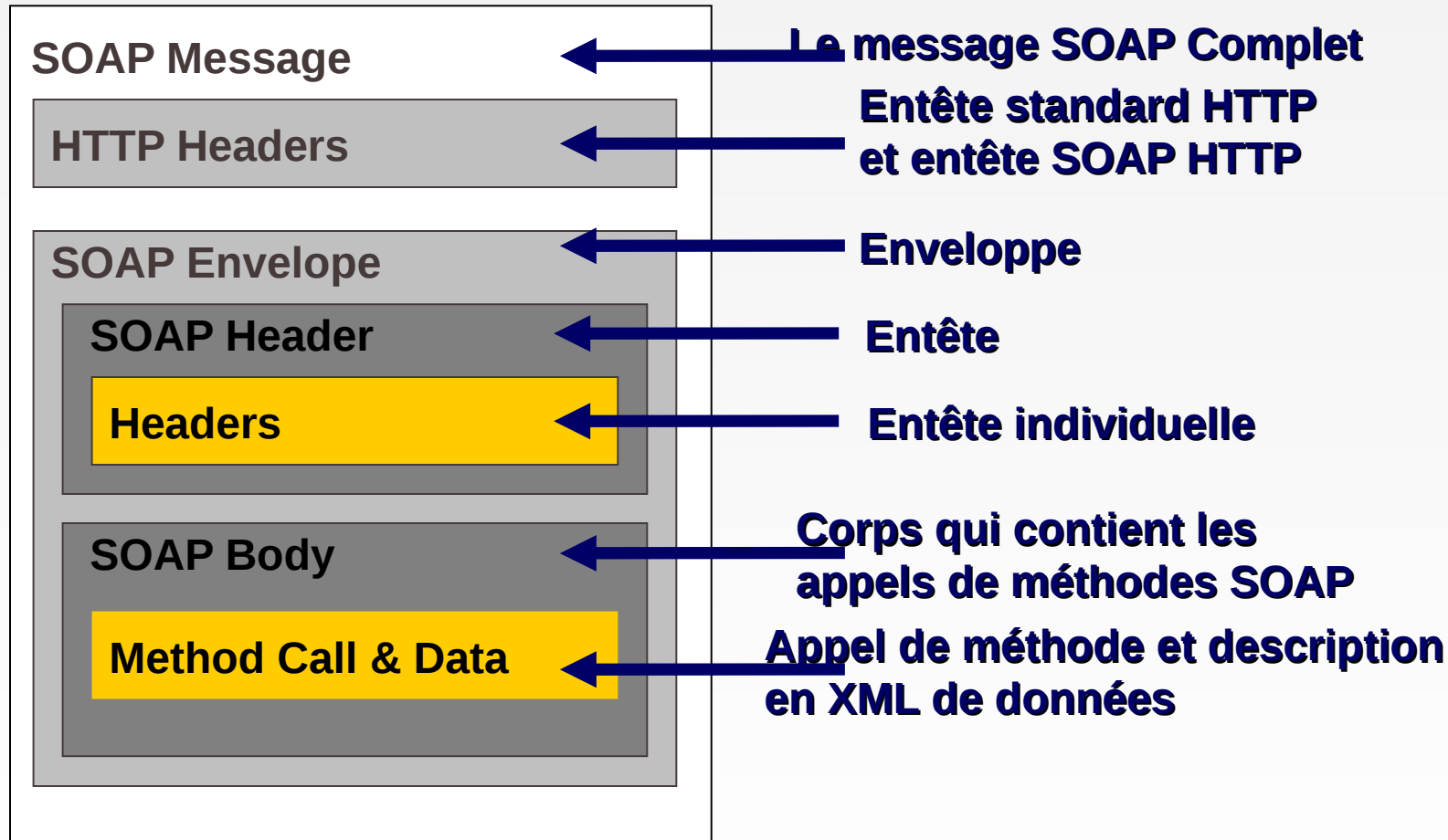
Content-Length: nnnn

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

Eléments de SOAP

- L'enveloppe (enveloppe)
 - Définit la structure du message
- Les règles d'encodage (encoding rules)
 - Définit le mécanisme de sérialisation permettant de construire le message pour chacun des types de données pouvant être échangés
- Fonctionnement en modèle client / serveur (RPC representation)
 - Définit comment sont représentés les appels de procédure et les réponses

Structure d'un message SOAP




Structure d'un message

- Enveloppe / Envelope
 - Élément racine
 - Namespace : SOAP-ENV <http://schemas.xmlsoap.org/soap/envelope/>
- Entête / Header
 - Élément optionnel
 - Contient des entrées non applicatives (Transactions, sessions, ...)
- Corps / Body
 - Contient les entrées du message
 - Nom d'une procédure, valeurs des paramètres, valeur de retour
 - Peut contenir les éléments « fault » (erreurs)

Exemple avec header

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```



Header

body

Types de message SOAP

- SOAP définit trois types de message
 - Appel (Call) - obligatoire
 - Réponse (Response) - optionnel
 - Erreur (Fault) - optionnel

Appel simple

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

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

Réponse

HTTP/1.1 200 OK

Content-Type: text/xml

Content-Length: nnnn

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

Erreur

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

Erreur - balise Fault

- Balise permettant de signaler des cas d'erreur.
- La balise Fault contient les balises suivantes
 - faultcode : un code permettant d'identifier le type d'erreur
 - Client, Server, VersionMismatch, MustUnderstand
 - faultstring : une explication en langage naturel
 - faultactor : une information identifiant l'initiateur de l'erreur
 - detail : Définition précise de l'erreur.

- Un message SOAP contient des données typées
=> nécessité de définir un moyen d'encoder ces données
- Vocabulaire SOAP :
 - Value (valeur d'une donnée)
 - Simple value (string, integer, etc)
 - Compound value (array, struct, ...)
 - Type (d'une value)
 - Simple Type
 - Compound Type

Encodage

- L'encodage c'est la représentation de valeurs sous forme XML.
- Le décodage c'est la construction de valeurs à partir d'XML
- L'XML qui représente les valeurs a une structure qui dépend du type des valeurs
- Il faut donc définir le type
 - Soit mécanisme définit par l'utilisateur
 - Soit utilisation de schémas XML (préconisé)

Simple Types

- Type (XML Schema)

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

Type XML Schema

Construction de Type XML Schema

- Valeurs

```
<age>45</age>  
<color>Blue</color>
```

Simple Types

- La définition d'un schéma XML pour tout type peut être fastidieuse
- SOAP a défini deux façons de préciser le type d'une valeur sans définir le Schéma XML:
 - `<SOAP-ENC:int>45</SOAP-ENC:int>`
 - `<cost xsi:type="xsd:float">29.5</cost>`

Compound Types

- Une structure est un type composé dans lequel les membres sont accessibles uniquement grâce à des noms différents.
- Un tableau est un type composé dans lequel les membres sont accessibles uniquement grâce à leur position.

Struct

- Type (XML Schéma)

```
<element name="Person">  
  <complexType>  
    <element name="name" type="xsd:string"/>  
    <element name="age" type="xsd:int"/>  
  </complexType>  
</element>
```

- Valeur

```
<Person>  
  <name>Jean</name>  
  <age>48</age>  
</Person>
```

Array

- Le type est directement précisé grâce aux balises SOAP:

```
<myFavoriteNumbers SOAP-ENC:arrayType="xsd:int[2] ">  
  <SOAP-ENC:int>3</SOAP-ENC:int>  
  <SOAP-ENC:int>4</SOAP-ENC:int>  
</myFavoriteNumbers>
```

Sécurité

- Basée sur la sécurité dans http
 - HTTPS
 - Certificats X.509
- Les Firewalls peuvent filtrer les messages facilement
- Pas de transfert de code applicatif
 - Uniquement des données
- Chaque développeur choisit de rendre visible telle ou telle méthode
- Les paramètres sont typés lors du transport

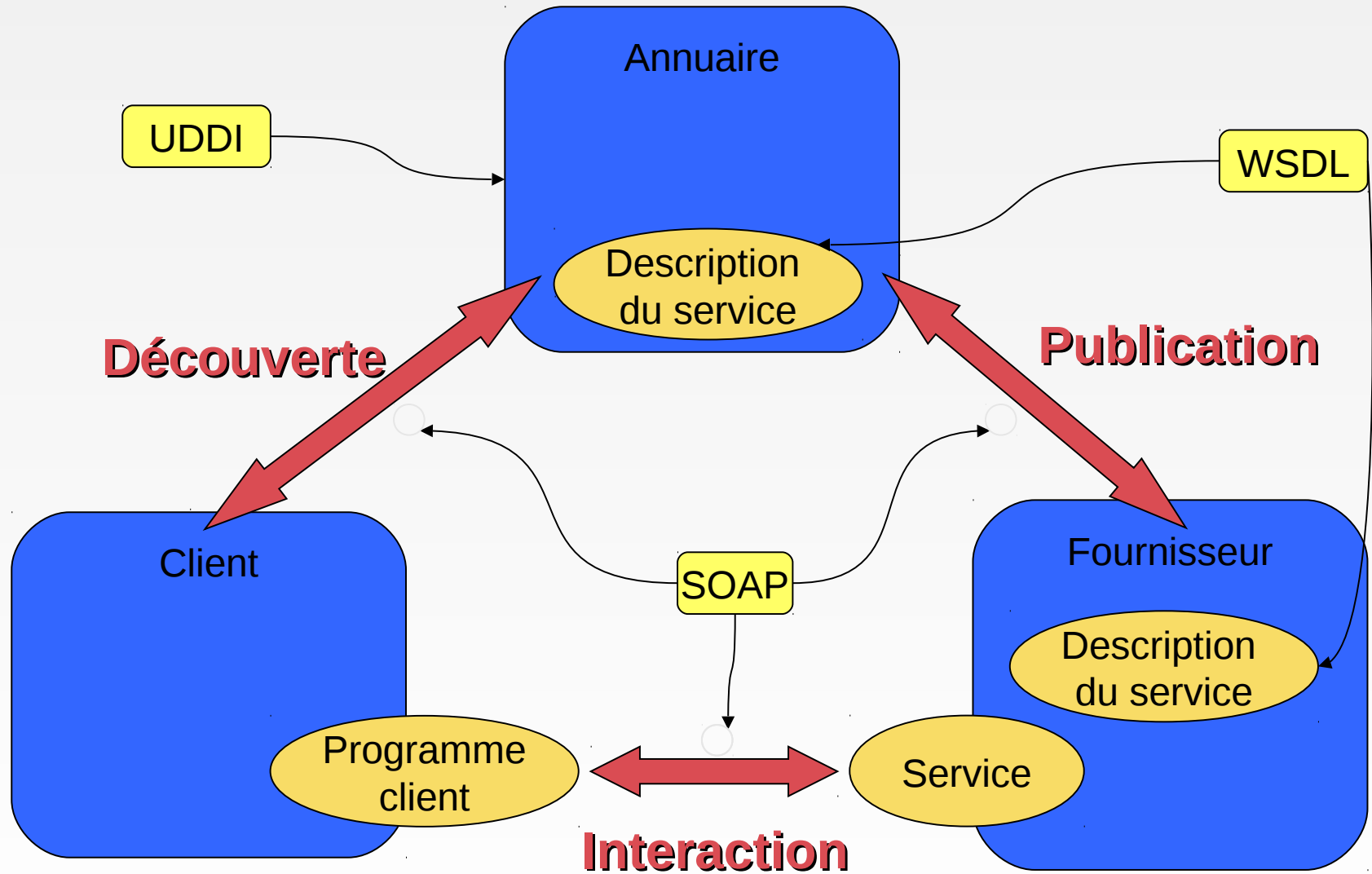
Portée de SOAP

- SOAP est simple et extensible
 - Il permet de réaliser des appels de méthode sur le Web
 - Indépendant des OS, des modèles objets, des langages
 - Transport des messages par HTTP + XML
 - Fonctionne avec l'infrastructure Internet existante
 - Permet l'interopérabilité entre OS, langages et modèles objets
- Ce n'est pas un système réparti à objets :
 - Pas de ramasse-miettes
 - Pas de contrôle de types
 - Pas de passage d'objets par référence

Conclusion (SOAP)

- Ni simple ni vraiment objet ...
- ... mais un protocole
 - assez léger
 - basé XML
 - s'appuyant principalement sur http
 - pour l'échange d'informations dans un environnement distribué
- SOAP n'est plus un acronyme depuis la version 1.2

Technologies



WSDL

Introduction

- But :
 - Décrire les services Web ...
 - ... comme un ensemble d'opérations et de messages abstraits associé à des protocoles et des serveurs réseaux
- WSDL est un langage
 - basé sur XML
 - utilisé pour
 - décrire les services offerts par une entreprise,
 - fournir un moyen aux particuliers ou à d'autres entreprises d'y accéder électroniquement
 - modulaire (import d'autres documents WSDL)

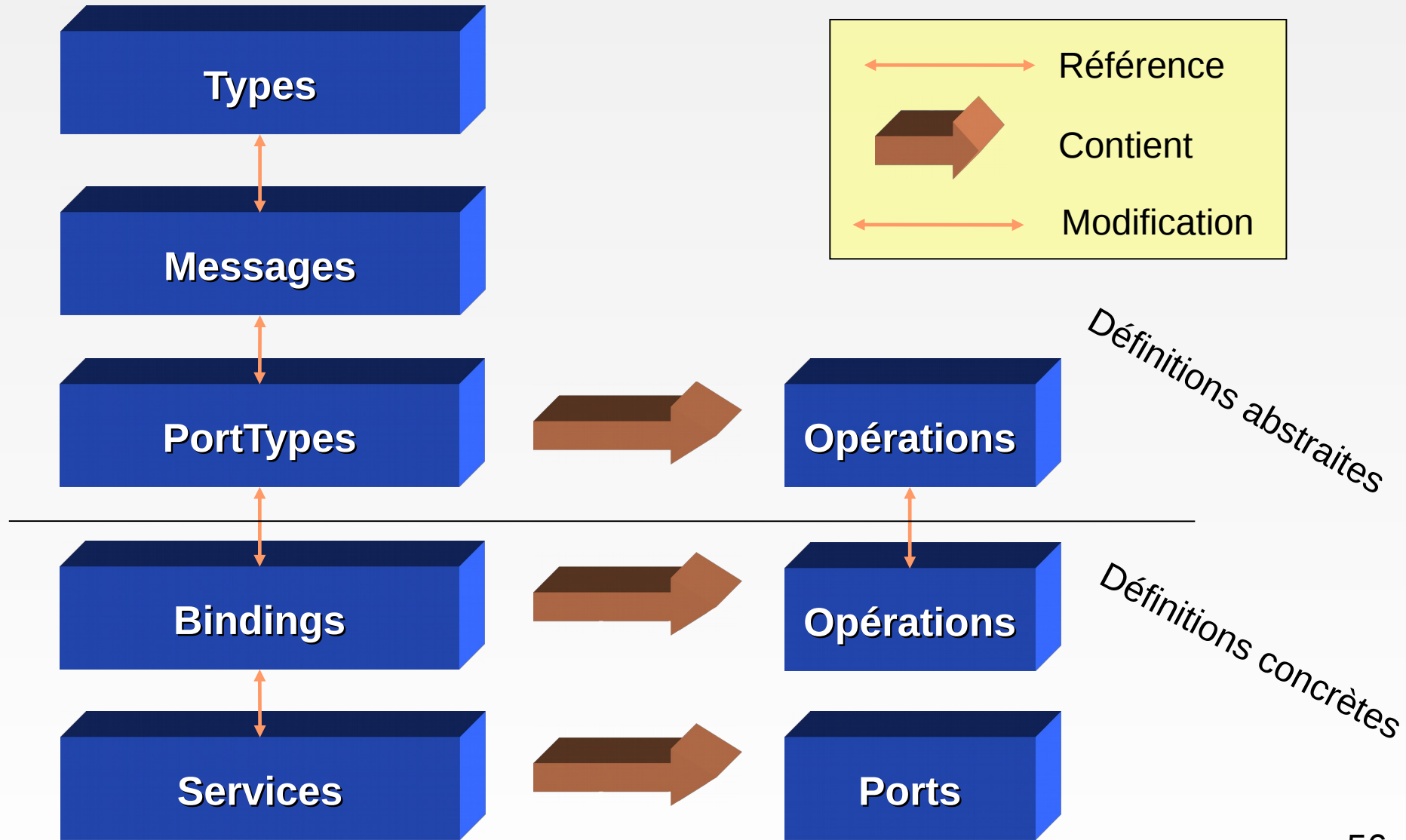
Partie Abstraite et Concrète

- Partie abstraite : Messages et données échangées de manière abstraite
 - Types de données
 - Messages élémentaires
 - Opérations (services élémentaires)
 - Les opérations sont regroupées au sein d'un portType
- Partie concrète : Manière dont les données sont encodées et quel protocole est utilisé
 - Encodage et protocole utilisé
 - Adresse du service

Éléments d'une définition WSDL

- **<types>**
 - Contient les définitions de types
- **<message>**
 - Décrit les noms et types d'un ensemble de champs à transmettre
 - Paramètres d'une invocation, valeur du retour, ...
- **<porttype>**
 - Décrit un ensemble d'opérations. Chaque opération a 0 ou 1 message en entrée, 0 ou plusieurs messages de sortie ou de fautes
- **<binding>**
 - Spécifie une liaison d'un <porttype> à un protocole concret (SOAP1.1, HTTP1.1, MIME, ...). Un <porttype> peut avoir plusieurs liaisons
- **<port>**
 - Spécifie un point d'entrée (endpoint) comme la combinaison d'un <binding> et d'une adresse réseau
- **<service>**
 - Une collection de points d'entrée (endpoint) relatifs

Définitions abstraites et concrètes



Premier Exemple

1 <?xml version="1.0" encoding="UTF-8" ?>
<definitions name="FooSample"
 targetNamespace="http://tempuri.org/wsdl/"
 xmlns:wsdl="http://tempuri.org/wsdl/"
 xmlns:typens="http://tempuri.org/xsd"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:stk="http://schemas.microsoft.com/soap-toolkit/wsdl-extension"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

2 <types>
 <schema targetNamespace="http://tempuri.org/xsd"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 elementFormDefault="qualified" >
 </schema>
</types>

3 <message name="Simple.foo">
 <part name="arg" type="xsd:int"/>
</message>

 <message name="Simple.fooResponse">
 <part name="result" type="xsd:int"/>
 </message>

4 <portType name="SimplePortType">
 <operation name="foo" parameterOrder="arg" >
 <input message="wsdl:Simple.foo"/>
 <output message="wsdl:Simple.fooResponse"/>
 </operation>
</portType>

 <binding name="SimpleBinding" type="wsdl:SimplePortType">
 <stk:binding preferredEncoding="UTF-8" />
 <soap:binding style="rpc"
 transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="foo">
 <soap:operation
 soapAction="http://tempuri.org/action/Simple.foo"/>
 <input>
 <soap:body use="encoded" namespace="http://tempuri.org/message/"
 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
 </input>
 <output>
 <soap:body use="encoded" namespace="http://tempuri.org/message/"
 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
 </output>
 </operation>
 </binding>

5

6 <service name="FOOSAMPLEService">
 <port name="SimplePort" binding="wsdl:SimpleBinding">
 <soap:address location="http://carlos:8080/FooSample/FooSample.asp"/>
 </port>
</service>
</definitions>

1 - Entête

`<?xml version="1.0" encoding="UTF-8" ?>`

Début des définitions de
l'exemple

`<definitions name="FooSample"`

`targetNamespace="http://tempuri.org/wsdl/"`

Espace de noms cible

`xmlns:wsdl="http://tempuri.org/wsdl/"`

`xmlns:typens="http://tempuri.org/xsd"`

`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

`xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"`

`xmlns:stk="http://schemas.microsoft.com/soap-toolkit/wsdl-extension"`

`xmlns="http://schemas.xmlsoap.org/wsdl/">`

Espace de noms par
défaut

- Définition des espaces de noms

2 - Types

```
<types>
  <schema targetNamespace="http://tempuri.org/xsd"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    elementFormDefault="qualified" >
    </schema>
</types>
```

- Ici, pas de types spécifiques à l'application
- Ex de type Google :

```
<xsd:complexType name="DirectoryCategory">
  <xsd:all>
    <xsd:element name="fullViewableName" type="xsd:string"/>
    <xsd:element name="specialEncoding" type="xsd:string"/>
  </xsd:all>
</xsd:complexType>
```

Cq elt du all apparait 0 ou 1 fois, dans n'importe quel ordre

3 - Messages

```
<message name="Simple.foo">  
  <part name="arg" type="xsd:int"/>  
</message>
```

Message correspondant
à un paramètre arg:int

```
<message name="Simple.fooResponse">  
  <part name="result" type="xsd:int"/>  
</message>
```

Message correspondant à
un paramètre de retour
result:int

- Permet la définition des messages à échanger pour les passages de paramètres
- Chaque paramètre a un nom et un type

4 - PortType

```
<portType name="SimplePortType">  
  <operation name="foo" parameterOrder="arg" >  
    <input message="wsdl:Simple.foo"/>  
    <output message="wsdl:Simple.fooResponse"/>  
  </operation>  
</portType>
```

- Définition des méthodes, regroupées dans des portType
- Operation
 - Définition des paramètres
 - Association entre l'opération foo et les messages à échanger (tels que définis en 3), ie associe les paramètres (et leur ordre)

5 - Binding

```
<binding name="SimpleBinding" type="wsdl:SimplePortType">
  <stk:binding preferredEncoding="UTF-8" />
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="foo">
    <soap:operation soapAction="http://tempuri.org/action/Simple.foo"/>
    <input>
      <soap:body use="encoded" namespace="http://tempuri.org/message/"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded" namespace="http://tempuri.org/message/"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
```

- Association avec le protocole, encodage
- Ici : encodage SOAP

6 - Port et service

```
<service name="FOOSAMPLEService">  
  <port name="SimplePort" binding="wsdl:SimpleBinding">  
    <soap:address location="http://carlos:8080/FooSample/FooSample.asp"/>  
  </port>  
</service>  
</definitions>
```

- Un service contient des ports
- Un port associe un binding avec une adresse

Élément <types>

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

- Exemple

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


Élément <message>

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

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

Élément <message>

- Les messages sont envoyés entre deux interlocuteurs
- Un message est composé de plusieurs **parts**
- Deux façons de définir des parts
 - Soit une part est un élément de type simple

```
<wsdl:message name="TelMsg">
```

```
  <wsdl:part name="areacode" type="xsd:int" />
```

```
  <wsdl:part name="number" type="xsd:string" />
```

```
</wsdl:message>
```

- Soit une part est un élément XML dont le type est défini dans un XML Schema

```
<wsdl:message name="TelMsg">
```

```
  <wsdl:part name="personne" element="phone" />
```

```
</wsdl:message>
```

Éléments <porttype> et <operation>

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

Éléments <porttype> et <operation>

- Exemple

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

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

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

</portType>
```

Élément <binding>

- WSDL permet de lier une description abstraite (portType) à un protocole.
- Chacune des opérations d'un portType pourra être liée de manière différente.
- Le protocole SOAP est un des protocoles qui peut être utilisé.
- D'autres binding sont standardisés par WSDL : HTTP et MIME.

Élément <binding>

- Un Binding :
 - peut être identifié par un nom : **name**
 - identifie le portType : **type**

```
<wsdl:binding name="binding_name"  
  type="nom du portType" >
```

...

```
</wsdl:binding>
```

Élément <binding> -- SOAP

- Pour préciser que le binding est de type SOAP, il faut inclure la balise suivante :

```
<soap:binding transport="uri" style="soap_style" />
```

- L'attribut *transport* définit le type de transport (<http://schemas.xmlsoap.org/soap/http> pour utiliser SOAP/HTTP)
- L'attribut *style* définit la façon dont sont créés les messages SOAP de toutes les opérations
 - *rpc* : encodage RPC défini par SOAP RPC
 - *document* : encodage sous forme d'élément XML
- Pour chaque opération du portType :
 - il faut préciser l'URI de l'opération : *soapAction*
 - on peut préciser la façon dont sont créés les messages SOAP : *style*
- Pour chaque message de chaque opération, il faut définir comment sera créé le message SOAP

Élément <binding> -- SOAP et HTTP

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


Élément <binding> -- SOAP et SMTP

```
<binding name="StockQuoteSoap" type="tns:StockQuotePortType">  
  <soap:binding style="document" transport="http://stockquote.com/smtp"/>  
  <operation name="SubscribeToQuotes">  
    <input message="tns:SubscribeToQuotes">  
      <soap:body parts="body" use="literal"/>  
      <soap:header message="tns:SubscribeToQuotes" part="subscribeheader"  
        use="literal"/>  
    </input>  
  </operation>  
</binding>
```

Élément <service>

- Un service est un ensemble de ports ie de points d'entrée
- Un port a un portType, et, dans le cadre de SOAP, un port a une adresse
- Exemple :

```
<!-- service decln -->  
<service name="AddressBookService">  
  <port name="AddressBook" binding="AddressBookSOAPBinding">  
    <soap:address  
      location="http://www.mycomp.com/soap/servlet/rpcrouter"/>  
  </port>  
</service>
```

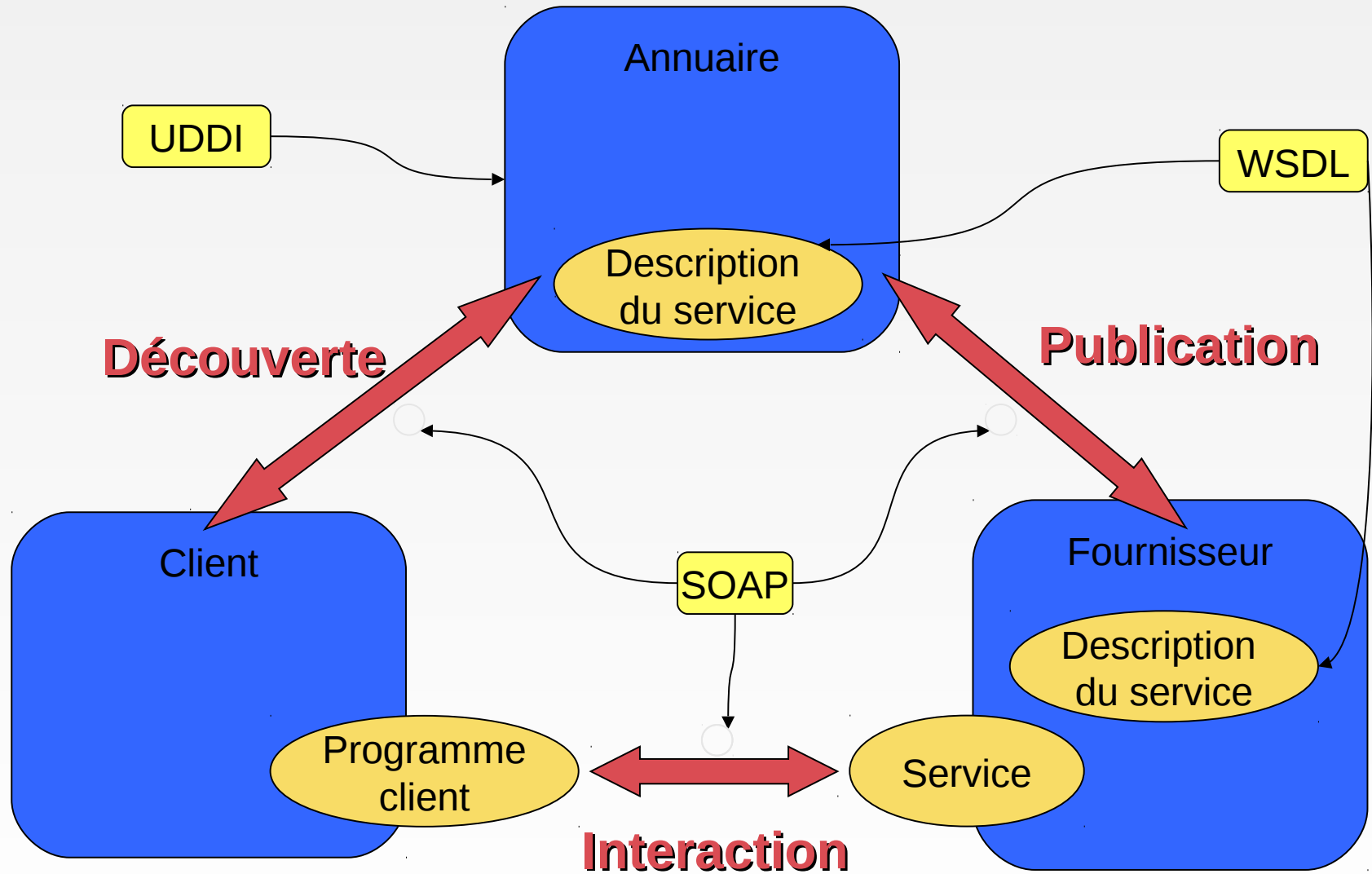
Outils

- Générateur WSDL à partir de déploiement SOAP ou EJB, ...
- Générateur de proxy SOAP à partir de WSDL
- Toolkits (Wsd2Java / Java2Wsd, ...)
 - Propriétaires (non normalisés)
- Outils de WSDL vers C# et réciproquement

Conclusion (WSDL)

- Un dialecte XML
- Permettant de décrire des services
 - De manière verbeuse (c'est du XML ...)
- Association avec un protocole (souvent SOAP)

Technologies

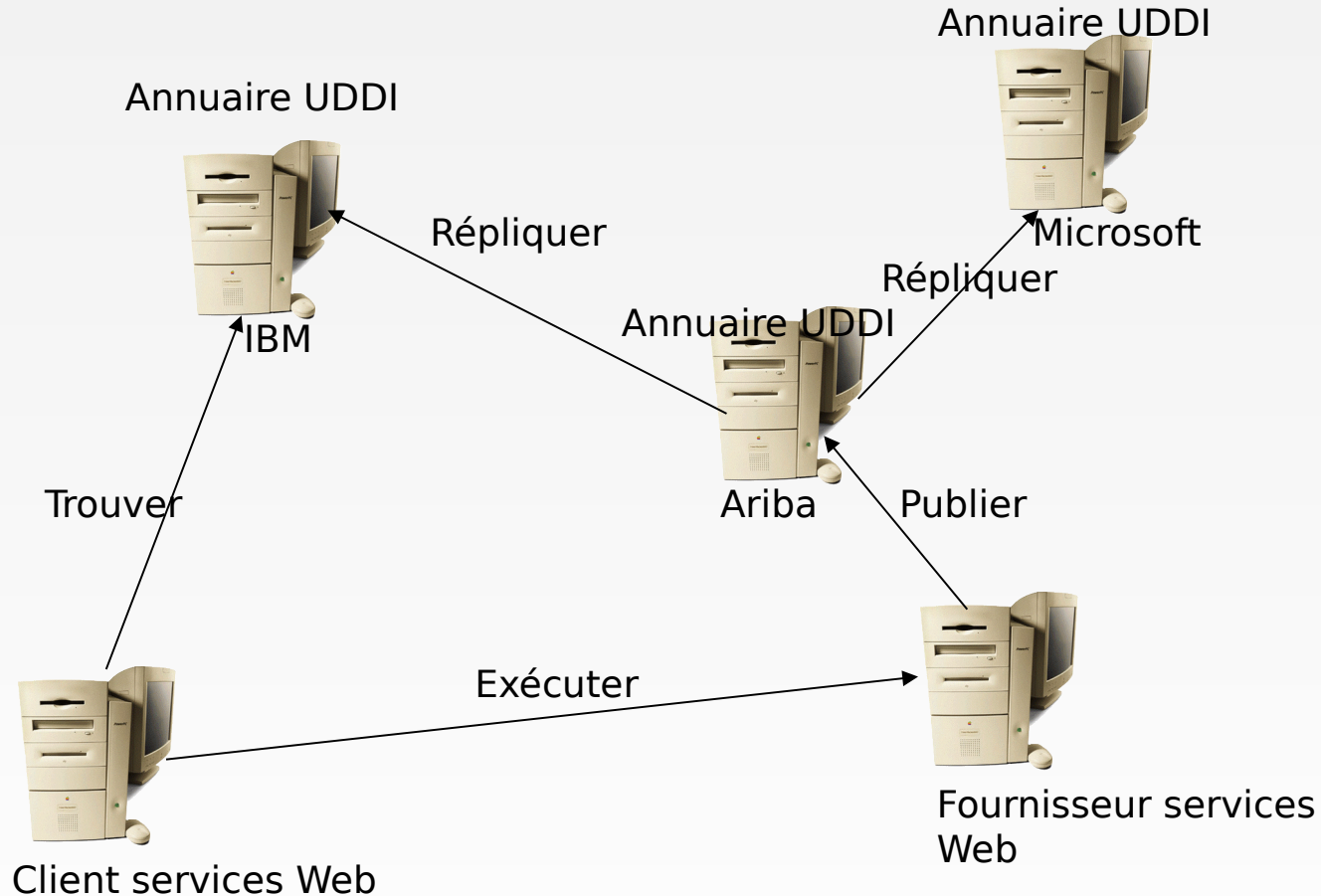


UDDI

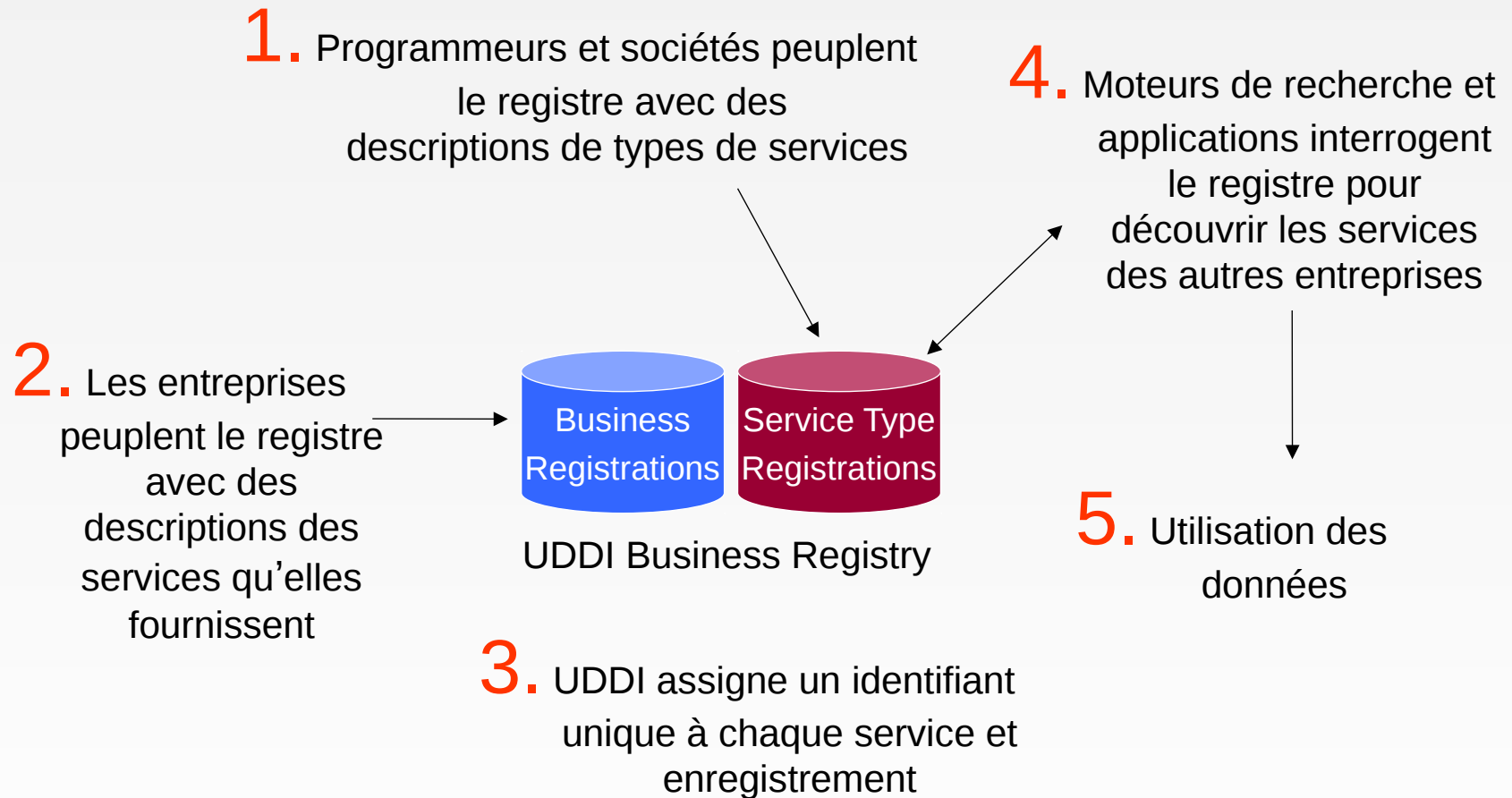
UDDI

- Origine : Besoin de découvrir les Web Services
 - Les concepteurs d'applications Web ont besoin de trouver les fournisseurs de services Web
 - Les outils ont besoin d'obtenir la description de ces services pour coder les appels nécessaires
 - Les applications elles-mêmes peuvent avoir besoin de trouver de nouveaux services Web en cours d'exécution
- UDDI a été créé pour répondre à ces besoins
 - par OASIS (SUN, Microsoft, Oracle, ...)
- UDDI est un modèle de découverte de services Web centralisé avec réplication

Annuaire UDDI



Fonctionnement UDDI



Registre UDDI

- Les entreprises fournissent des données publiques sur leur propre compte et sur leurs services
- Les organismes de normalisation, les développeurs et les entreprises fournissent des informations sur les types de services

Pages Blanches

Pages Jaunes

Pages Vertes

Service Type
Registrations
(tModels)

Pages Blanches UDDI

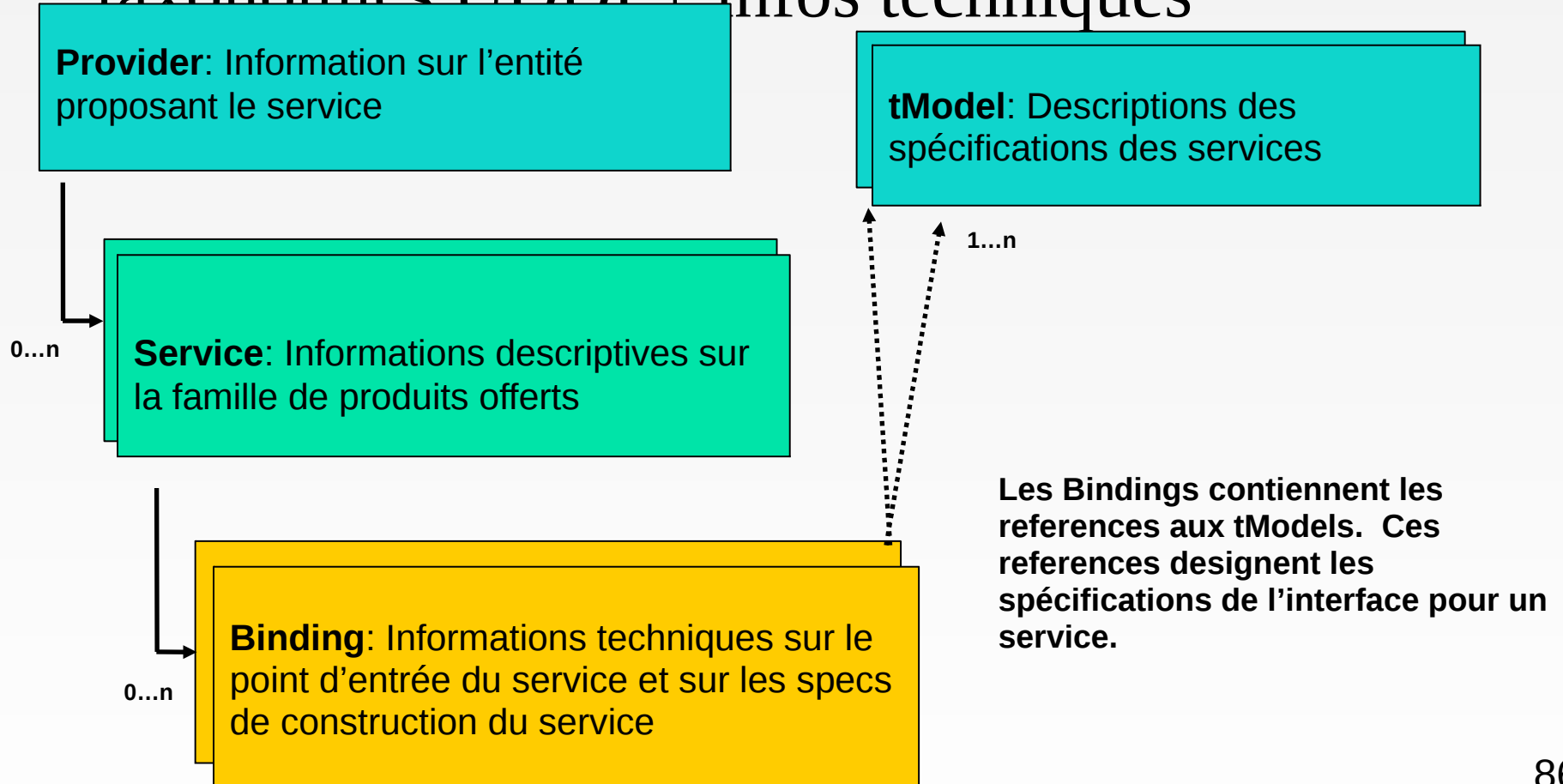
- Liste des entreprises, avec infos associées (businessEntity)
 - Nom entreprise
 - Description (texte)
 - Contact
 - Noms, n° tél., fax
 - Site Web, etc.
 - Identificateurs publics
 - N° SIRET, etc.

- Recensement des services web et description en termes métier
- BusinessService
- Pas de détails techniques

- Informations techniques précises sur les services fournis :
 - descriptions de services et d'information de liaison
 - processus métiers associés.
- bindingTemplates

tModels UDDI

- Structure de données pour décrire des taxonomies UDDI + infos techniques



Référentiels

- Type
 - Public :
 - Microsoft
 - IBM
 - ...
 - Privé ou d'entreprise
- Accès
 - Défini en WSDL
 - Plusieurs APIs d'interrogation d'UDDI

Problèmes d'UDDI

- Pas de modérateur d'annuaires
 - Risques d'entrées erronées, de doublons, de fraude
- Pas de QoS
 - Pas moyen d'indiquer tarifs ou QoS
 - Par ex. niveau sécurité assuré, fiabilité ou disponibilité, support des transactions
 - Pas moyen de vérifier, voire appliquer niveau QoS
- Centralisation excessive pour certains
- La plupart des gros annuaires ont « fermé »
-
- → Annuaires locaux

SOAP et Sécurité

Sécurité et WS : les besoins

- Authentification des utilisateurs
- Gestion des autorisations
- Intégrité des messages (signatures)
- Cryptage des messages

WS et sécurité : les solutions

- Au début des services web : rien !
- Puis, Microsoft, IBM et Verisign travaillent sur WS-security
- WS-security est finalement repris par le consortium Oasis, sous le nom de WSS
 - v1 : mars 2004
 - v2 : février 2006
- SOAP Message Security : intégrité et confidentialité au niveau des messages SOAP

- 3 mécanismes :
 - envoi de jetons (tokens) à l'intérieur des messages
 - intégrité des messages
 - confidentialité des messages
- Besoins de WSS :
 - des formats de jetons de sécurité
 - des domaines de confiance (trust domains)
 - des formats et des technologies pour le cryptage
 - Sécurité de bout en bout et pas seulement au niveau des messages SOAP

Terminologie (1)

- Confidentialité
 - propriété faisant que les données ne sont pas accessibles aux entités, processus ou individus non autorisés
- Signature digitale
 - valeur calculée par un algo cryptographique et liée aux données. Les destinataires des données peuvent utiliser la signature pour vérifier que les données n'ont pas été altérées. En général : calcul et vérification par algos symétriques, basés sur des clefs différentes.

Terminologie (2)

- Intégrité
 - Propriété : les données n'ont pas été modifiées
- Intégrité des messages
 - Propriété du message fournie par une signature digitale
- Confidentialité des messages
 - Propriété du message assurée par cryptage

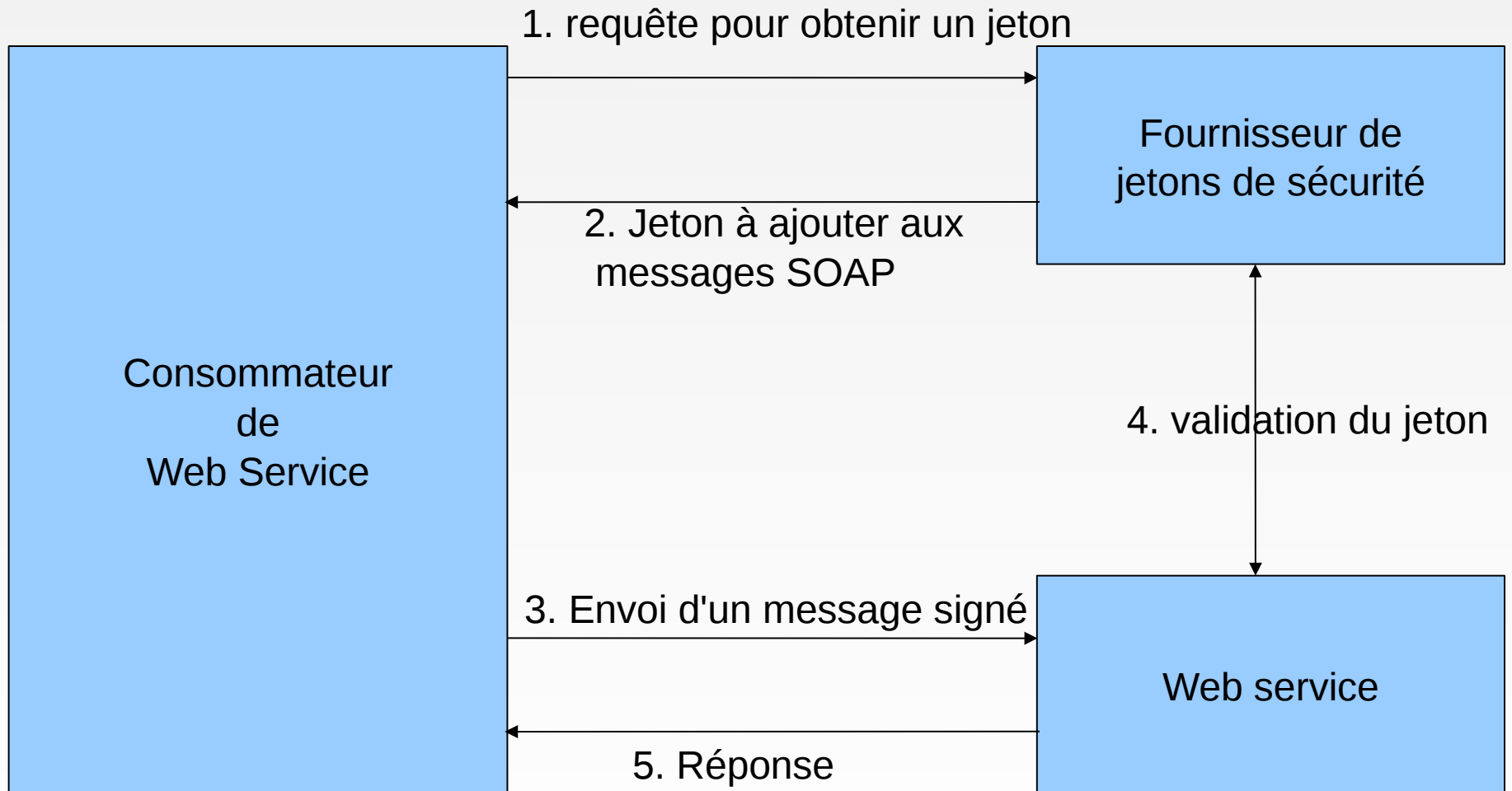
Terminologie (3)

- Jeton de sécurité (signé/pas signé)
 - collection d'affirmations (claims)
 - Affirmation (claim) : déclaration faite par une entité (nom, identité, privilèges, ...)
 - Confirmation d'affirmation : vérification qu'une affirmation s'applique à une entité
 - Si signé : vérifié et cryptographiquement signé par une autorité spécifique (certificat X.509 ou ticket Kerberos)

WSS : appliquer des concepts existants à SOAP

- Existant :
 - protocole d'authentification : Kerberos ou X.509
 - classique paire login / mot de passe
- WSS codifie cela dans SOAP
- Les messages SOAP vont transporter des jetons d'authentification
 - UserNameToken (mot de passe)
 - Binaire : Kerberos ou X.509

Schéma classique



Jetons de sécurité (security tokens)

- 3 types :
 - UsernameToken
 - certificats X.509
 - Tickets Kerberos

SOAP:actor en SOAP1.1
cible des infos de sécurité
optionnel

<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd>

```
<SOAP:Envelope xmlns:SOAP="...">
  <SOAP:Header>
    <wsse:Security SOAP:role="..."
                  SOAP:mustUnderstand="...">
      <wsse:UsernameToken>
        ..
      </wsse:UsernameToken>
      ...
    </wsse:Security>
  </SOAP:Header>
  <SOAP:Body Id="MsgBody">
    <!-- SOAP Body data -->
  </SOAP:Body>
</SOAP:Envelope>
```

Quelques éléments du UsernameToken

- /Username : nom d'utilisateur associé au jeton
- /Password : mot de passe pour l'utilisateur associé au jeton
- /Password/@Type : type de mot de passe fourni ; 2 types pré-définis :
 - PasswordText : mot de passe en clair
 - PasswordDigest : digest du mot de passe, hash-value encodée avec SHA1 du mot de passe (encodé en utf8)
- /Nonce :nonce (jeton (chaîne) unique de chiffrement généré de manière aléatoire)
- /Created : date et heure de création du token
- PasswordDigest=Base64(SHA-1(Nonce+Created+Password))

Username Token : transmission en clair

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="...">
  <S11:Header>
    ...
    <wsse:Security>
      <wsse:UsernameToken>
        <wsse:Username>Zoe</wsse:Username>
        <wsse:Password>IloveDogs</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
    ...
  </S11:Header>
  ...
</S11:Envelope>
```

soap 1.1

Username Token : transmission cryptée

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="...">
  <S11:Header>
    ...
    <wsse:Security>
      <wsse:UsernameToken>
        <wsse:Username>NNK</wsse:Username>
        <wsse:Password Type="PasswordDigest">
          weYl3nXd8LjMNVksCKFV8t3rgHh3Rw==
        </wsse:Password>
        <wsse:Nonce>WScqanjCEAC4mQoBE07sAQ==</wsse:Nonce>
        <wsu:Created>2003-07-16T01:24:32Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
    ...
  </S11:Header>
  ...
</S11:Envelope>
```

Scénario avec mot de passe haché

Message avec jeton J :

- username=c1
- passwordHaché
- nonce =bkg6876JHY6-;m
- created= ...

Client c1

mdp=azerty

Serveur

calcul de :

T.password(J.username)+
J.nonce+J.created

Hachache SHA-1 puis base64

Si résultat= J.passwordHaché

authentificationOK

sinon, authentificationKO


T

login	password
c1	azerty
c2	qwerty
...	...

Attaque possible et parade

- Le jeton peut être attrapé et réutilisé
- Parade 1
 - utiliser des timeout sur les messages

<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd>



```
<wsu:Timestamp wsu:Id="...">  
  <wsu:Created ValueType="...">...</wsu:Created>  
  <wsu:Expires ValueType="...">...</wsu:Expires>  
  ...  
</wsu:Timestamp>
```

- Parade 2
 - garder un historique des nonces reçus le temps que les timestamp expirent

- Attaques toutefois possible (blocage du message) 10

Utilisation de certificats X.509

- Chiffrement asymétrique
 - clef publique pour encoder
 - clef privée pour décoder
- Aucune garantie que la clé publique récupérée est celle du service auquel on s'adresse.
 - Si pirate remplace la PK par la sienne, il sera en mesure de décoder les messages !
- Certificat : associe une PK à une entité
- Délivré par un organisme de certification

Structure d'un certificat

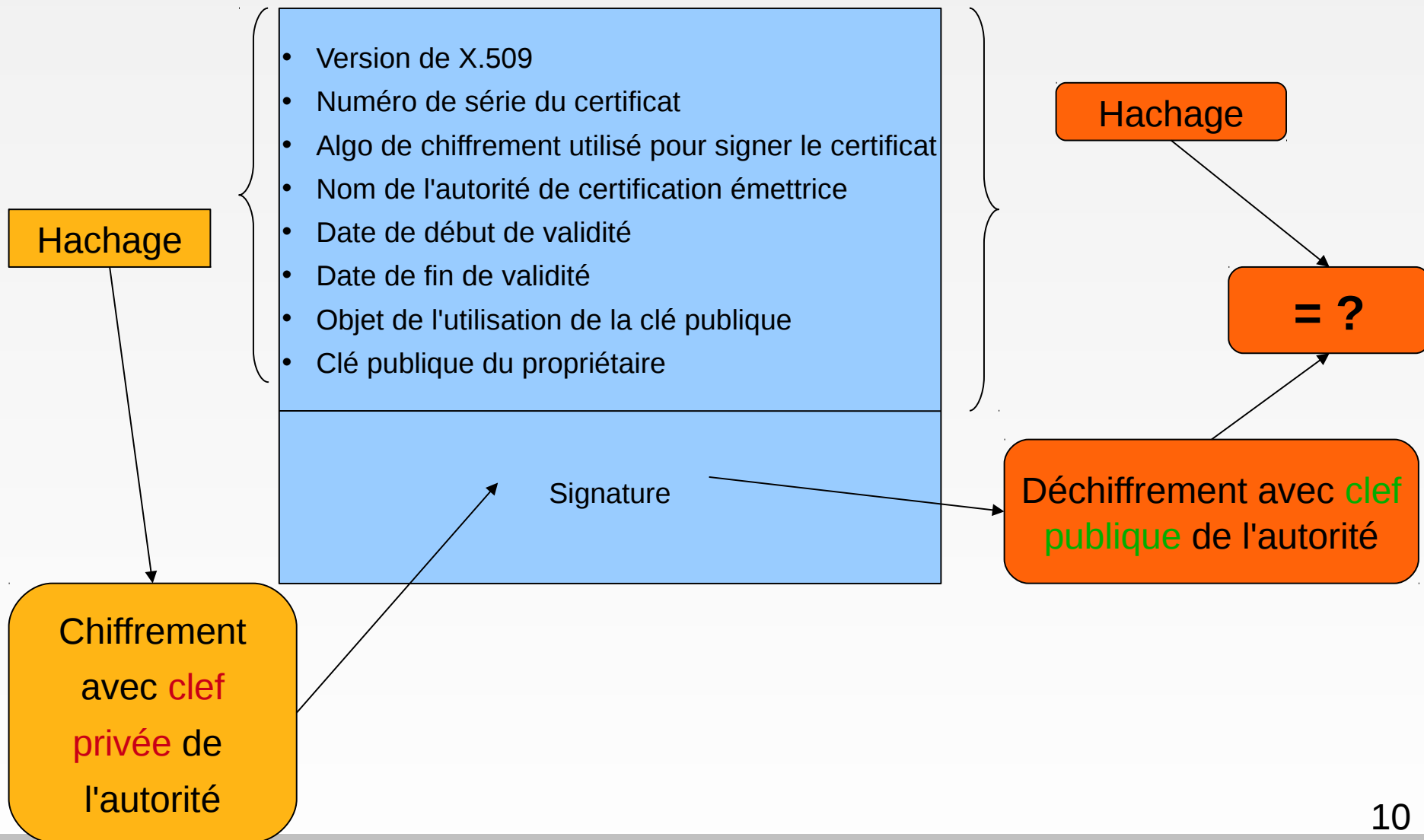
- Version de X.509
- Numéro de série du certificat
- Algo de chiffrement utilisé pour signer le certificat
- Nom de l'autorité de certification émettrice
- Date de début de validité
- Date de fin de validité
- Objet de l'utilisation de la clé publique
- Clé publique du propriétaire

Signature

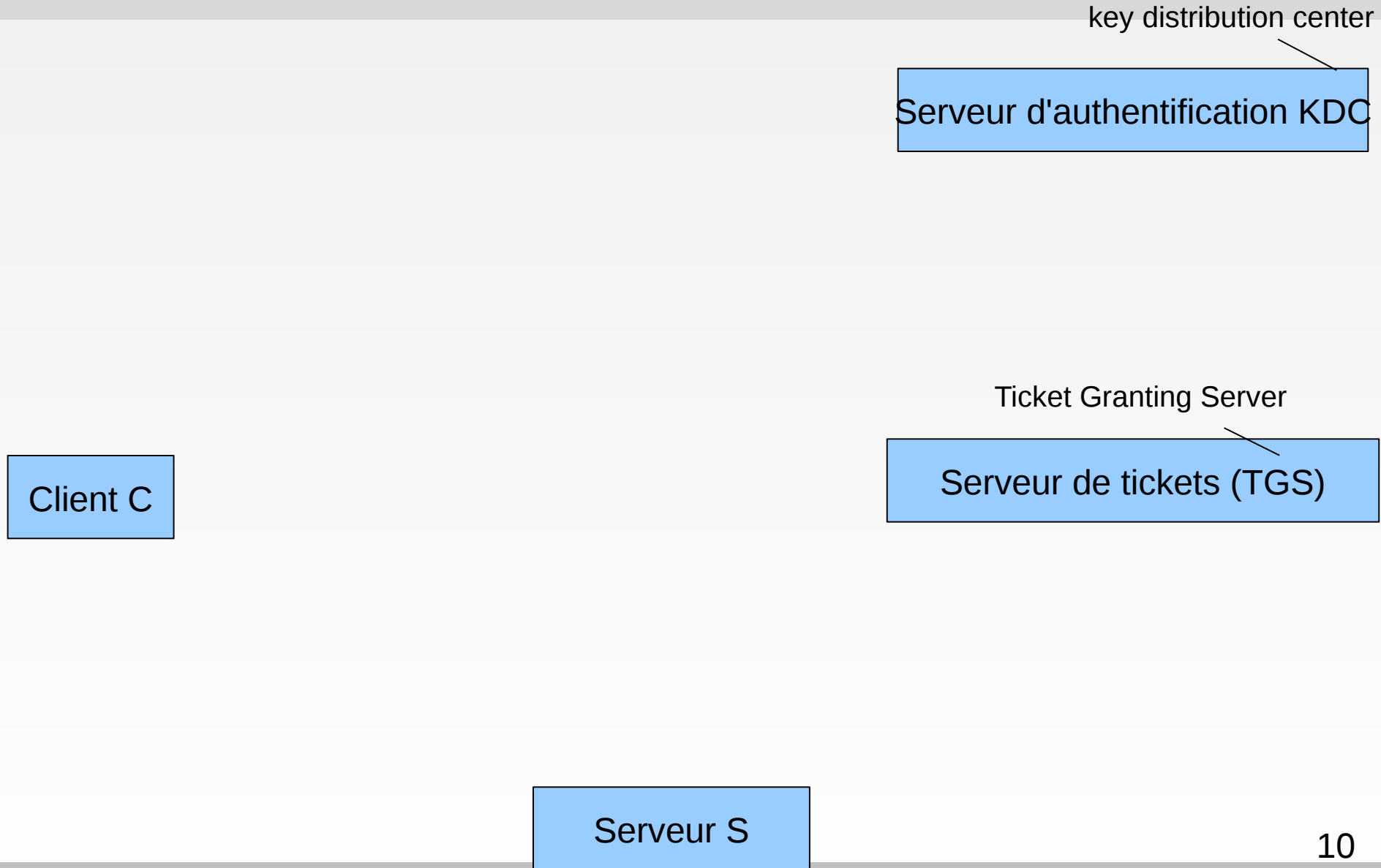
Hachage

Chiffrement avec clef
privée de l'autorité

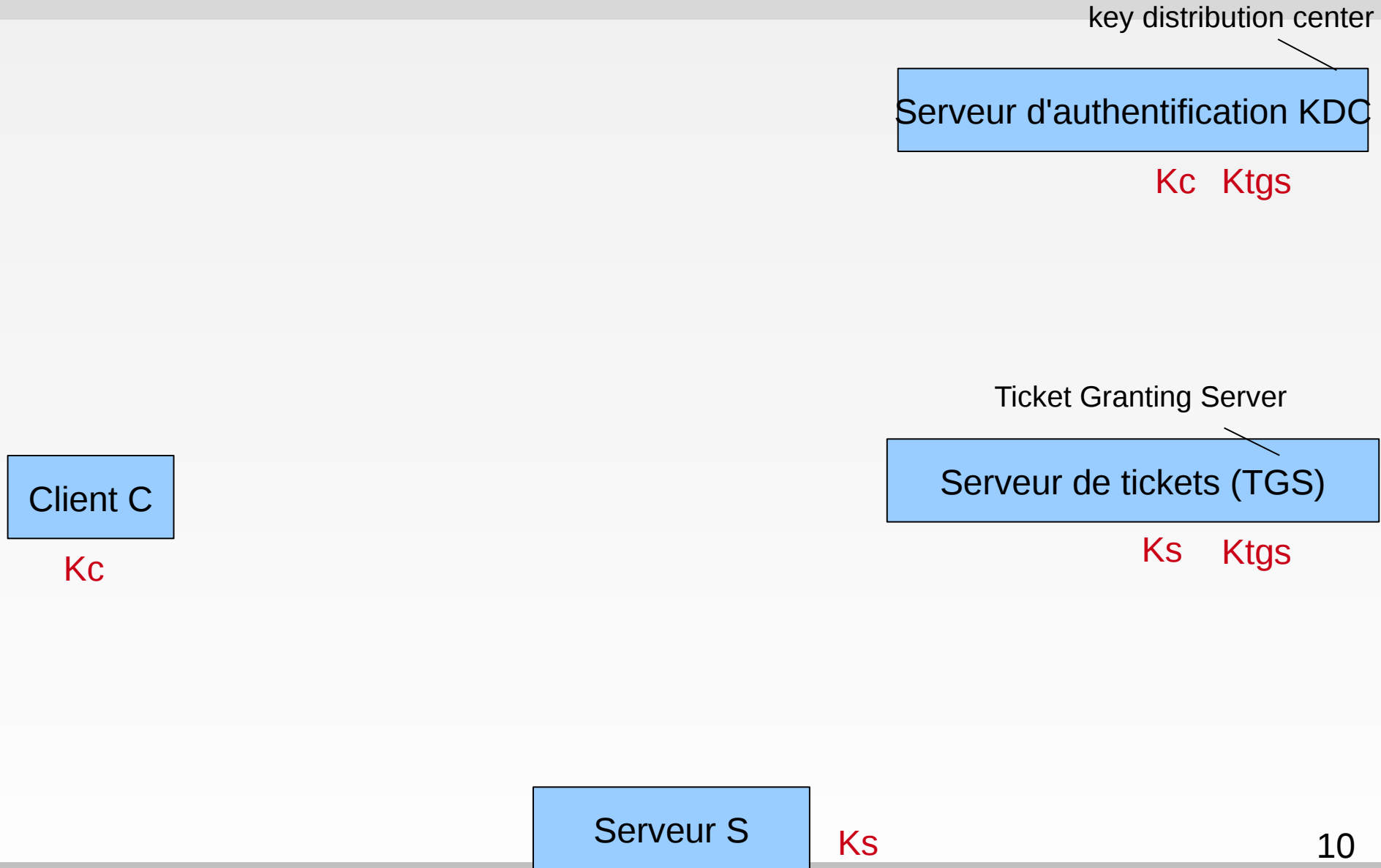
Côté client



Aperçu des tickets Kerberos : les acteurs



Aperçu des tickets Kerberos : les clefs initiales



Aperçu des tickets Kerberos

15. le client s'assure
que le timestamp
est ok

1. nom, serveur de tickets

key distribution center

Serveur d'authentification KDC

2. vérif
 K_c K_{tgs}

3. Clef de session $K_{c,tgs}$
chiffrée avec K_c
+ Ticket T_{tgs} chiffré avec K_{tgs}

4. déchiffre
 $K_{c,tgs}$

Ticket Granting Server

Serveur de tickets (TGS)

K_c

Client C

5. Demande de ticket (id+date émission)

chiffrée avec $K_{c,tgs}$ + ticket T_{tgs}

8. Ticket d'accès au serveur chiffré avec K_s
+ $K_{c,s}$ chiffrée avec $K_{c,tgs}$

6. déchiffre T_{tgs}
 K_s K_{tgs}

7. déchiffre l'identifiant
du client avec $K_{c,tgs}$

12. déchiffre ticket

13. fabrication acquittement avec
timestamp+1 chiffré avec $K_{c,s}$

10
9

K_s

Serveur S

11. id+ticket d'accès

14. acquittement

9. déchiffre
 $K_{c,s}$

10. génère id avec
timestamp chiffré

avec $K_{c,s}$

Principes

SOAP

WSDL

UDDI

Sécurité

.NET / Java

Certificats X.509, Tickets Kerberos et SOAP

```
<wsse:BinarySecurityToken
  ValueType="wsse:X509v3"
  EncodingType="wsse:Base64Binary"
  Id="SecurityToken-f49bd662-59a0-401a-ab23-1aa12764184f">
  MIIHdjCCB...
</wsse:BinarySecurityToken>
```

wsse:Base64Binary ou
wsse:HexBinary

wsse:X509v3 ou
wsse:Kerberosv5TGT ou
wsse:Kerberosv5ST

Implémentation WSS en .net

- Dans WSE dans les versions antérieures de .net
 - Web Service Enhancement
- Dans WCF actuellement
- Manipulation de certificats
- Encryption
- Manipulation des messages SOAP

Conclusion

- La 3ème génération du Web
- Technologies Standards du Web
 - SOAP 1.1 (puis SOAP 1.2)
 - WSDL
- Technologies non standardisées
 - UDDI, DISCO
- WS-*
 - interopérabilité, sécurité, orchestration, policy

WS et interopérabilité

- Un ensemble de règles (basic profile) à suivre pour l'interopérabilité
- Des outils qui vérifient les règles en monitorant les échanges
- Règles plus ou moins suivies par les outils qui génèrent les requêtes SOAP

WS-I : exemples de règles

- un message doit être sérialisé en UTF-8 ou UTF-16
- un receveur doit générer une faute soap:MustUnderstand quand un message contient un bloc de header obligatoire (ie avec un attribut soap:MustUnderstand à 1) que le receveur ne comprend pas
- un service ne doit pas exiger une acceptation des cookies côté client pour fonctionner correctement (utilisation pour optimisation)
- L'élément wsdl:documentation peut apparaître comme enfant d'un élément wsdl:part

Concevoir une application orientée services

- Principes
 - expliciter les frontières
 - fonctionnelles
 - sécurité
 - données et regroupement
 - faciliter la consommation
 - prévoir les cas d'erreurs dues à la distribution
 - ne pas transmettre au delà des frontières des détails internes

Concevoir une application orientée services

- Principes (suite)
 - Autonomie des services
 - gestion de version et déploiement autonome
 - ne pas modifier des schémas une fois publiés
 - Pas de partage d'objets
 - partage de contrat / de données
 - Réfléchir à la compatibilité des services

À ne pas faire ...

- Concevoir l'interface comme celle d'un objet
 - rajouter des [WebMethod] sur toutes les méthodes de l'interface
 - interface CRUD en web method
 - attention à la visibilité !
- Ecrire des méthodes très abstraites
 - `Public DataSet Query(String requete)`
 - `Public Boolean Execute(int commande)`
 - Rien dans le contrat !