XML et services web

Université de Cergy-Pontoise

Master Informatique SIIC M2

Remise à niveau en Bases de données

Dan VODISLAV

Dan.Vodislav@u-cergy.fr

Plan

- XML
  - Principes, syntaxe
  - DTD
  - Espaces de noms
  - XML Schema
  - DOM
- Services web
  - Principes
  - SOAP
  - WSDL
  - UDDI

XML: eXtensible Markup Language

- Langage de description de documents structurés
  - Utilisation de balises (balisage structurel)
- Standard W3C pour l'échange / publication de données sur le web
- Héritage:
  - HTML: documents publiés sur le web
  - SGML: documentation technique (documents structurés)
    - HTML est une grammaire spécifique de SGML
  - Données structurées: bases de données relationnelles, objet

Pourquoi XML?

- HTML: documents sur le web
  - Langage *de présentation* pour les documents du web
  - Ensemble de balises et grammaire fixes, mélange d'éléments de structure de document et de mise en page
    - Structure: titres, paragraphes, listes, images, etc.
    - Mise en page: type caractère (italique, gras, …), alignement, taille police, etc.

  → Difficile de déduire la signification du contenu

- Données structurées: bases de données
  - Décrivent le contenu, pas la présentation qu'on peut en faire
  - Structure régulière basée sur des types simples: string, int, boolean, …

  → Les documents du web mal adaptés à cette structuration rigide
    - Texte, structure variable

Conclusion: on a besoin de séparer le contenu de la présentation, mais à l'aide d'une structuration flexible, adaptée aux documents

Exemple

Bibliographie

- G. Gardarin, *XML : des bases de données aux services web*, Dunod, 2003
- S. Abiteboul, N. Polyzotis, *The Data Ring*, CIDR, 2007

HTML

```
<h1>Bibliographie</h1>
<ul><li>G. Gardarin, <i>XML : Des Bases de Données aux Services Web</i>, Dunod, 2003
    <li>S. Abiteboul, N. Polyzotis, <i>The Data Ring</i>, CIDR, 2007
</ul>
```

Base de données relationnelle « Bibliographie »

Auteur	Titre	Éditeur	Conférence	Année
G. Gardarin	XML : des bases de données aux services web	Dunod	NULL	2003
S. Abiteboul	The Data Ring	NULL	CIDR	2007
N. Polyzotis	The Data Ring	NULL	CIDR	2007

Exemple (suite)

- XML

```
<bibliographie>
   <ouvrage année="2003">
      <auteur>G. Gardarin</auteur>
      <titre>XML : Des Bases de Données aux Services Web</titre>
      <éditeur>Dunod</éditeur>
   </ouvrage>
   <ouvrage année="2007">
      <auteur>S. Abiteboul</auteur>
      <auteur>N. Polyzotis</auteur>
      <titre>The Data Ring</titre>
      <conférence>CIDR</conférence>
   </ouvrage>
</bibliographie>
```

Un document XML contient

- *Un prologue*: présence facultative, mais fortement conseillée
  - Informations sur le document

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

  - Informations sur un éventuel schéma (DTD)

```
<!DOCTYPE document SYSTEM "document.dtd">
```

- *Un arbre d'éléments*: obligatoire

```
<document>
    <salutation>Bonjour!</salutation>
</document>
```

- *Des commentaires et des instructions de traitement*: facultatifs

```
<!-- Ceci est un commentaire -->

<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

Les DTD

- DTD = grammaire pour la structure des documents
  - Interne ou externe au document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE chapitre [
  <!ELEMENT chapitre (titre,intro?,section+)>
  <!ATTLIST chapitre type CDATA #IMPLIED
                     ref ID #REQUIRED>
  <!ELEMENT titre (#PCDATA)>
  <!ELEMENT section (par|fig)*>
  <!ELEMENT par (#PCDATA|refch)*>
  <!ELEMENT refch EMPTY>
  <!ATTLIST refch cible IDREF #REQUIRED>
  ...
]>
<chapitre ref="ch1">
  <titre>Contexte de l'étude</titre>
  <section><par>Les travaux présentés chapitre <refch cible="ch2"> ... </par>
    ...
  </section>
  ...
</chapitre>
```

Documents bien formés et documents valides

- Document XML *bien formé* : document correct *sans DTD*
  - le prologue ne contient pas de déclaration de type de document (DTD)
  - contient un arbre d'éléments correct

```
<?xml version="1.0"?>
<document>
  <salutation>Bonjour!</salutation>
</document>
```

- Document XML *valide* : document correct *avec DTD*
  - son prologue contient une déclaration de type de document (DTD)
  - son arbre d'éléments respecte la structure définie par la déclaration de type

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE document [
  <!ELEMENT document (salutation)>
  <!ELEMENT salutation (#PCDATA)>
]>
<document>
  <salutation>Bonjour!</salutation>
</document>
```

Espaces de noms

- Espace de noms (« namespace »)
  - Collection de noms d'éléments ou noms d'attributs, identifiée par un URI
  - But: éviter les conflits de noms quand on veut intégrer des vocabulaires de balises
    - Liberté dans le choix des balises → possibilité de « collision »
    - Un même nom de balise peut être utilisé avec des significations et des compositions différentes

```
<document>
  <art:film xmlns:art='http://www.pariscope.fr/'>
    <com:acteur xmlns:com='http://www.comedie.fr/'
                com:nom='Juliette Binoche' />
  </art:film>
</document>
```

Nom qualifié et nom universel

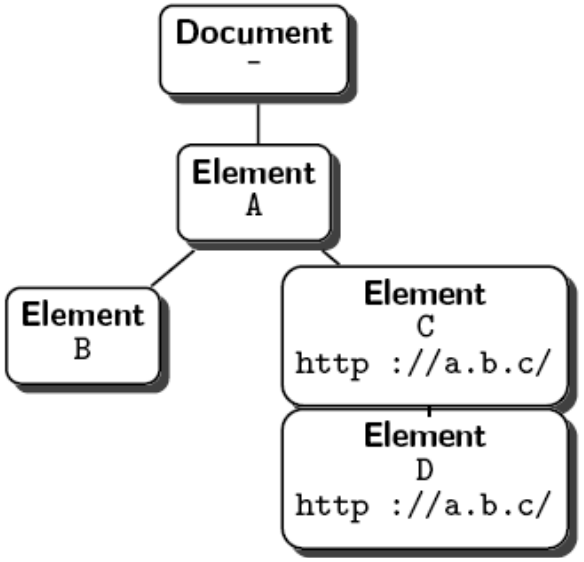
- Nom qualifié
  - Composé d'un préfixe (optionnel) et du type de l'élément (nom local)
- Nom universel
  - Composé d'une URI (optionnelle) et du type de l'élément (nom local)
  - Obtenu à partir du nom qualifié en remplaçant le préfixe par sa définition

```
<document>
  <art:film xmlns:art='http://www.pariscope.fr/'>
    <com:acteur xmlns:com='http://www.comedie.fr/'
                com:nom='Juliette Binoche' />
  </art:film>
</document>
```

Nom qualifié	Nom universel
art:film	http://www.pariscope.fr/:film
com:acteur	http://www.comedie.fr/:acteur

Espace de nom sans préfixe

```
<?xml version="1.0"?>
<A>
  <B/>
  <C xmlns="http://a.b.c/">
    <D/>
  </C>
</A>
```



- Noms qualifiés sans préfixe
  - Un élément sans préfixe appartient à l'espace de noms défini par *l'attribut `xmlns` de l'ancêtre le plus proche*. Si cet attribut n'existe pas, l'élément n'appartient à aucun espace de nom.
  - Un attribut sans préfixe n'appartient à aucun espace de noms

XML Schema

- Recommandation W3C pour le typage des documents XML
  - Alternative aux DTDs, beaucoup plus puissante
  - Richesse de types
    - Types simples, complexes, abstraits, anonymes
    - Sous-typage: extension, restriction
  - Séparation types – éléments
    - Définition séparée des types et des éléments
    - Possibilité de référencer et partager des types et des éléments
  - Contraintes d'intégrité: clés, clés étrangères
- Autres différences avec les DTD
  - Schéma exprimé en format XML
  - Types locaux et anonymes dans la définition des éléments
  - Éléments locaux dans la définition d'un type

Types simples

- Type simple: ensemble de valeurs (pas d'élément)
  - DTD: un seul type simple (#PCDATA), 10 types d'attributs
  - XML Schema: 43 types simples
    - xsd:string, xsd:byte, ...
    - xsd:integer, xsd:long, xsd:float, xsd:double, ...
    - xsd:boolean
    - xsd:anyType
    - xsd:time, xsd:timeDuration, xsd:Date, xsd:year, xsd:month, ...
    - xsd:language, xsd:uriReference
    - xsd:ID, xsd:IDREF, xsd:NMTOKEN, ...

Exemple XML Schema

DTD

```
<!ELEMENT chapitre (titre,intro?,section+)>
<!ATTLIST chapitre num CDATA #REQUIRED>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT section (par|fig)*>
...
```

XML Schema

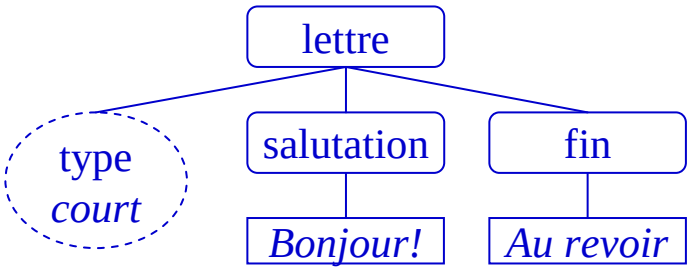
```
<xsd:element name='chapitre' type='chapitreType'/>
<xsd:complexType name='chapitreType'>
  <xsd:sequence>
    <xsd:element name='titre' type='xsd:string'/>
    <xsd:element name='intro' type=' xsd:string' minOccurs='0'/>
    <xsd:element name='section' type='sectionType'
                 minOccurs='1' maxOccurs='unbounded'/>
  </xsd:sequence>
  <xsd:attribute name='num' type='xsd:integer' optional='false'/>
</xsd:complexType>
<xsd:complexType name='sectionType'>
  <xsd:choice minOccurs='0' maxOccurs='unbounded'>
    <xsd:element name='par' type='parType'/>
    <xsd:element name='fig' type='figType'/>
  </xsd:choice>
</xsd:complexType>
```

Formes sérialisée et arborescente

- Forme sérialisée d'un document/élément
  - Chaîne de caractères (texte) incluant balises et contenu textuel

```
<lettre type='court'>
   <salutation>Bonjour!</salutation>
   <fin>Au revoir</fin>
</lettre>
```

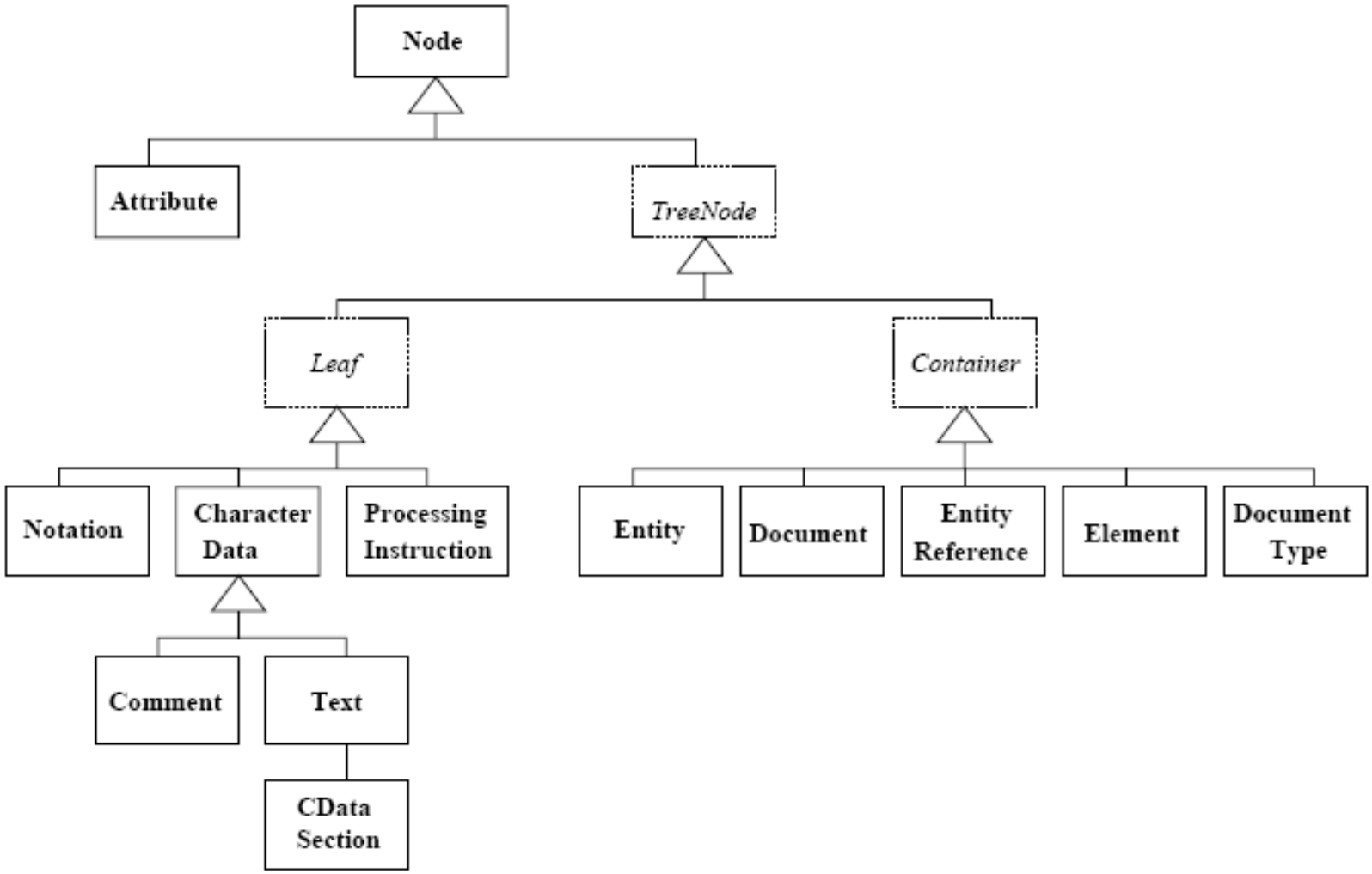
- Forme arborescente
  - Utilisée par les applications, modèle DOM (W3C)



DOM

- DOM = *Document Object Model*, norme W3C
- Un parseur DOM: document XML → **arbre** formé **d'objets**
  - chaque objet appartient à une sous-classe de *Node*
  - des opérations sur ces objets permettent de **créer, modifier, détruire** des noeuds, ou de **naviguer** dans le document
- Nœuds
  - Racine: type *Document*
  - Les autres nœuds: *Element*, *Attribute*, *Text*, *Comment*, etc.
  - Structure d'arbre
    - Nœuds containeurs (« Container »): nœuds internes de plusieurs types
    - Nœuds feuilles (« Leaf »)
    - Nœuds attribut (« Attribute »): similaires aux feuilles, avec quelques différences

Types de nœuds DOM



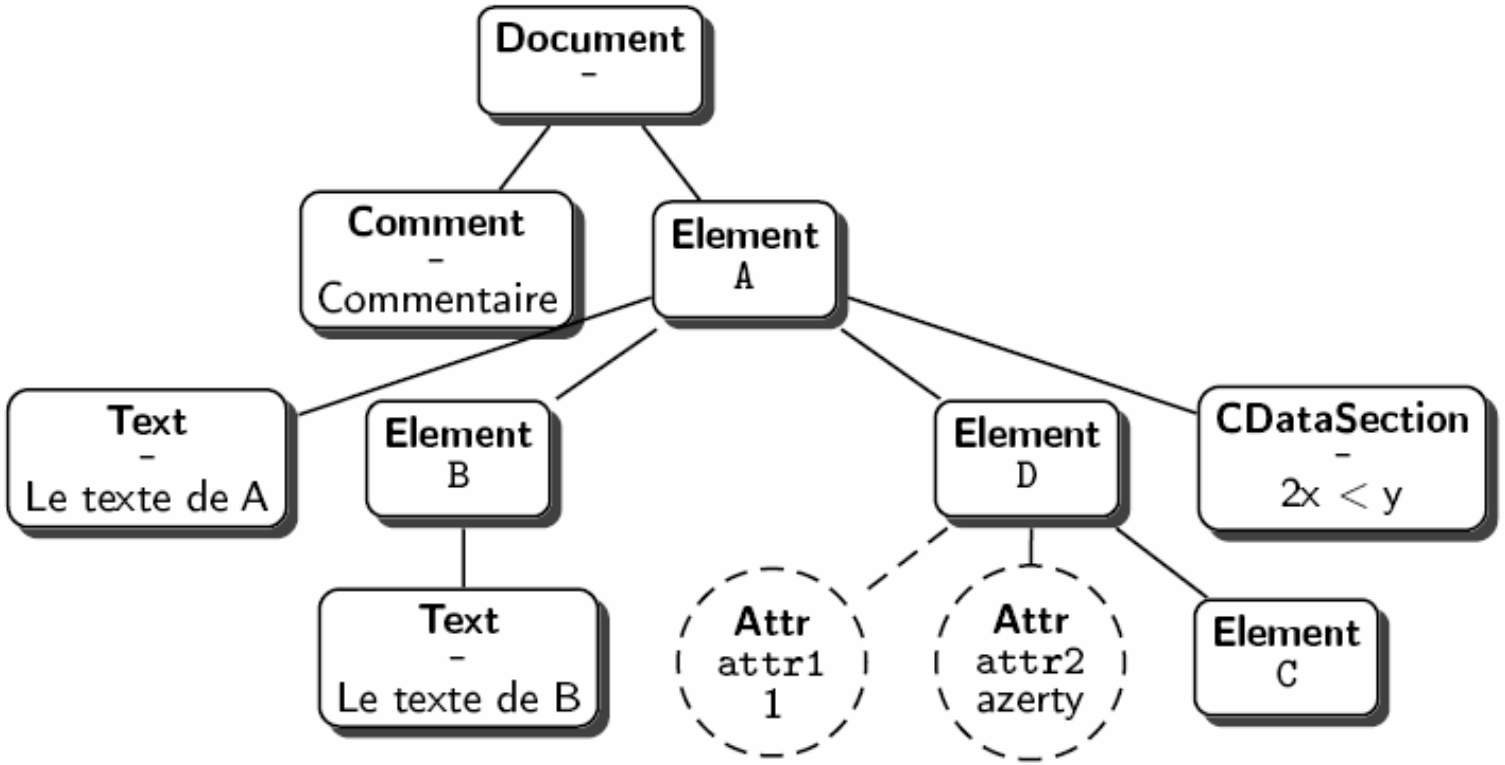
Représentation textuelle

- Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Commentaire -->
<A>Le texte de A
  <B>Le texte de B</B>
  <D attr1="1" attr2="azerty">
   <C/>
  </D>
  <![CDATA[2x < y]]>
</A>
```

Représentation DOM



DOM en tant que modèle orienté-objet

- *Node*: super-classe pour tous les objets dans un arbre DOM
  - Classes spécifiques pour chaque type de nœud (élément, texte, attribut, ...)
- Approche pragmatique dans le traitement de l'arbre
  - ***On voit tout comme des objets Node***
    - On parcourt l'arbre DOM comme un arbre d'objets *Node* et on réalise des actions en fonction du type du nœud courant
    - Le prix à payer: on met dans *Node* toutes les propriétés qui peuvent apparaître dans les différents types de nœud
    - Certains nœuds n'utilisent pas toutes ces propriétés
      - Ex: le nom est utilisé par *Element*, mais pas par *Text*

Principales propriétés de *Node*

Propriété	**Type**
nodeType	short (int)
nodeValue	String
firstChild	Node
childNodes	NodeList
nextSibling	Node

Propriété	**Type**
nodeName	String
parentNode	Node
lastChild	Node
previousSibling	Node
attributes	NamedNodeMap

NodeList: liste (tableau) de *Node*

- propriété ***length*** → *int*
- méthode ***item***(*int*) → *Node*

NamedNodeMap: table d'association par nom

- pareil que *NodeList* + méthode ***getNamedItem***(*String*) → *Node*
- aussi ***setNamedItem***, ***removeNamedItem***

Méthodes de *Node* par rapport à ses enfants

- ***insertBefore*** (*Node* nouveau, *Node* enfant)
  - Insère le nouveau nœud en tant que nouvel enfant, juste avant *enfant*
- ***appendChild*** (*Node* enfant)
  - Insère le nouveau nœud en tant que nouvel enfant, en dernière position
- ***replaceChild*** (*Node* nouveau, *Node* ancien)
  - Remplace l'ancien enfant avec le nouveau
- ***removeChild*** (*Node* enfant)
  - Supprime le nœud enfant en question
- *boolean* ***hasChildNodes*** ()
  - Vérifie si c'est un nœud feuille

Méthodes de *Document* pour la création de nœuds

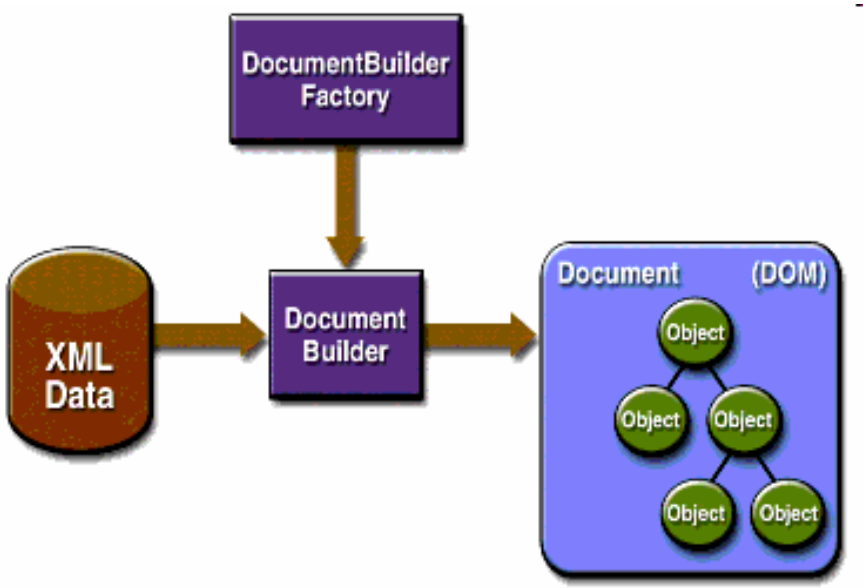
- Le nœud *Document*: le premier créé dans un arbre DOM
  - Joue le rôle de *fabrique* de nœuds pour l'arbre dont il est la racine
- Méthodes
  - ***createElement()*** : crée et retourne un nœud *Element*
  - ***createTextNode()*** : crée et retourne un nœud *Text*
  - ***createCommentNode()*** : crée et retourne un nœud *Comment*
  - …

Programmation DOM en Java

- JAPX: Java API for XML Processing
  - Incluse dans Java 1.5 - 1.6

```
import javax.xml.parsers.*
import org.w3c.dom.*
```

- Idée générale
  - *Parseur DOM* qui transforme un document XML en arbre DOM
  - Le parseur est produit par une *fabrique de parseurs*
  - L'arbre DOM est manipulé à l'aide des méthodes de parcours, modification, … de l'API



Méthodes Java pour *Node*

- Pour chaque propriété de *Node* → une méthode *get* qui retourne la valeur de la propriété
  - *getNodeType()*, *getNodeValue()*, *getFirstChild()*, …
- Des méthodes de modification de propriétés
  - *setNodeValue*
- Les méthodes concernant les enfants du nœud
  - *insertBefore*, *appendChild*, *replaceChild*, *removeChild*
- Autres méthodes

Obtenir un arbre DOM

- A mettre dans un try…catch

```
try{
  // création d'une fabrique de parseurs
  DocumentBuilderFactory fabrique =
      DocumentBuilderFactory.newInstance();
  fabrique.setValidating(true); //si l'on veut vérifier une DTD

  // création d'un parseur
  DocumentBuilder parseur = fabrique.newDocumentBuilder();

  // transformation d'un fichier XML en DOM
  File xml = new File("exemple.xml");
  Document document = parseur.parse(xml);
  ...
} catch(ParserConfigurationException pce){
     System.out.println("Erreur de configuration du parseur DOM");
} catch(SAXException se){
     System.out.println("Erreur lors du parsing du document");
} catch(IOException ioe){
     System.out.println("Erreur d'entrée/sortie");
}
```

Parcourir un arbre DOM

- Parcours récursif à partir de l'élément racine

```
Document document = parseur.parse(xml);
//obtenir l'élément racine
Node racine = document.getDocumentElement();
//explorer par appel récursif
TypeRésultat res = explorer(racine, ...);
```

- Le parcours peut:
  - Calculer un résultat: nombre de nœuds, d'éléments, représentations variées sous forme de chaîne de caractères, etc.
  - Modifier l'arbre DOM: ajout, suppression, modification de nœuds
  - Réaliser des actions: affichage, appels d'autres programmes

Exploration récursive

- Forme générale de la méthode d'exploration

```
static TypeRésultat explorer(Node noeud, ...){
  TypeRésultat resultat = ...;
  //traitement nœud courant
  if (noeud.getNodeType() == Node.TEXT_NODE){
       ... //action pour les nœuds texte
  } else if (noeud.getNodeType() == Node.ELEMENT_NODE){
        ... //action pour les nœuds élément
  } ...
  //parcours récursif
  if (noeud.hasChildNodes()){
       NodeList enfants = noeud.getChildNodes();
       for(int i=0; i<enfants.getLength(); i++){
          TypeRésultat resenf = explorer(enfants.item(i), ...);
          ... //combiner resenf avec resultat
       }
  }
  return resultat;
}
```

Exemple: modifier chaque nœud texte

- On rajoute la chaîne " (modifié)" à la fin de chaque texte

```
static void explorer(Node noeud){
  //traitement nœud courant seulement s'il est un texte
  if (noeud.getNodeType() == Node.TEXT_NODE){
      String modif = noeud.getNodeValue() + " (modifié)";
      noeud.setNodeValue(modif);
  }
  //parcours récursif
  if (noeud.hasChildNodes()){
      NodeList enfants = noeud.getChildrenNodes();
      for(int i=0; i<enfants.getLength(); i++){
         explorer(enfants.item(i));
      }
  }
}
```

Services web

- Infrastructure pour le développement d'applications distribuées sur le web
- Besoin d'applications distribuées
- Spécifique du web en tant que environnement distribué:
  - Contrôle limité sur les sites, débit faible
  - Utilisation des protocoles web (ex: HTTP), avec leurs limitations
  - Fonctionnalités, présentation moins riches (HTML)
  - Clients légers
- Objectif: réaliser des applications distribuées avec les contraintes imposées par le web

Caractéristiques des services web

- Demande de service adressée par un client à un serveur
  - Appel d'une fonction distante
  - Utilise les protocoles web: TCP/IP, HTTP
- Données transportées sur le web
  - Généralement du texte (HTTP: pages HTML)
  - Services web → texte en format XML
    - Format d'échange flexible
    - Standardisation
- Services web: évolution des architectures
  - Architectures distribuées classiques → web
    - RPC, RMI, CORBA, DCOM → HTTP, XML, services
  - Web « homme-machine » → web « machine-machine »

Avantages des services web

- Flexibilité
  - Indépendance du langage et du système
  - Données XML
- Interopérabilité dans des environnements distribués
  - Interfaces formalisées
  - Communication entre services, composition
  - Automatisation
- Adaptés à la communication sur le web
  - Protocoles web bien connus et acceptés (HTTP, SMTP, …)
  - Invocation à travers des pare-feux (à la différence de CORBA)

Scénario d'utilisation

1. Définition du service (fournisseur)
   - Description WSDL des entrées/sorties, des caractéristiques du service
2. Publication du service (fournisseur)
   - Publication de la description WSDL dans un annuaire (UDDI)
3. Recherche de service (client)
   - Recherche d'un service dans un annuaire → adresse du service choisi
4. Enregistrement au service web (client)
   - Enregistrement auprès du fournisseur pour accéder au service trouvé
5. Appel du service (client)
   - Exécution du service avec les paramètres fournis par le client
6. Composition (client, fournisseur)
   - Utilisation du résultat pour l'appel à d'autres services (client)
   - Appel d'un autre service lors de l'exécution du service appelé (fournisseur)

Scénario (suite)

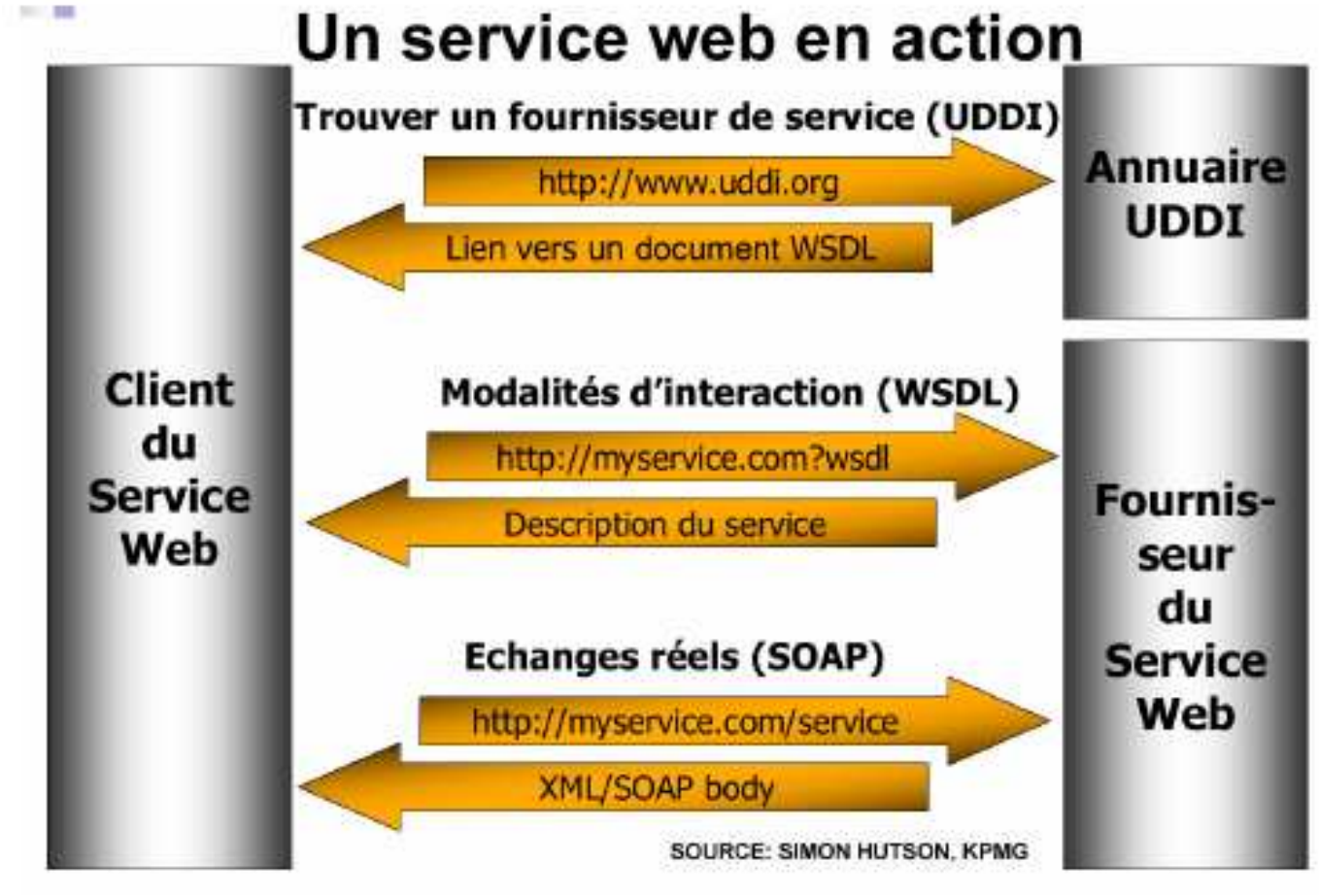
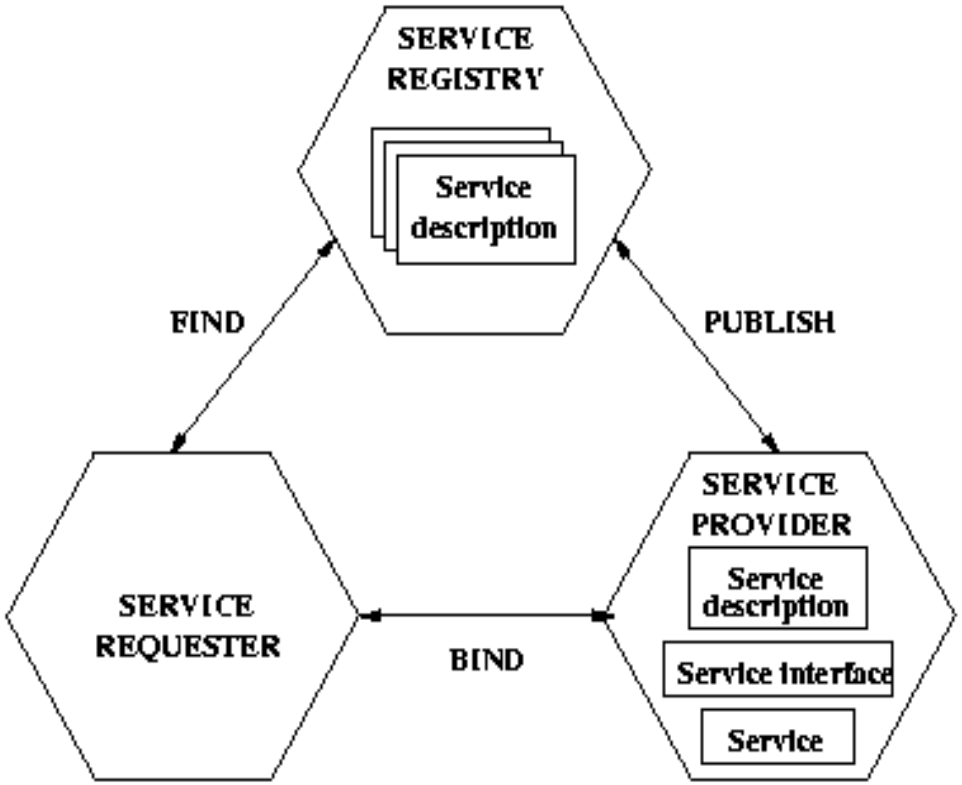


Schéma d'interaction

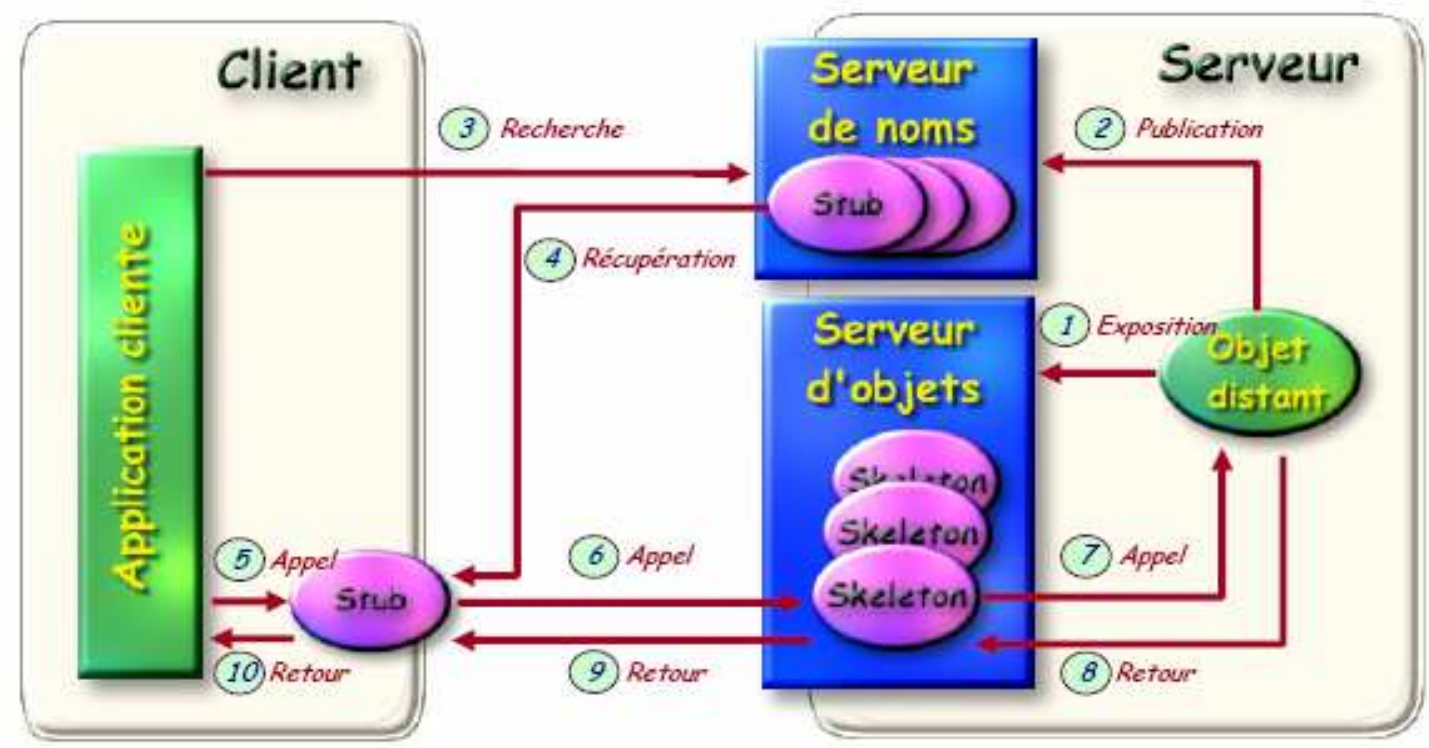


Source: G. Alonso

- Schéma proposé par IBM
  - Bas niveau, construction ascendante
  - D'autres architectures (ex. ebXML): haut niveau, construction descendante

Comparaison avec les solutions middleware

- Les mêmes notions existent dans CORBA, EJB, RMI



Source: R. Voyer

Technologies pour services web

- Technologies de base
  - SOAP : « Simple Object Access Protocol »
    - RPC par appel de service web
    - Protocole de communication à l'appel de services web
  - WSDL : « Web Service Description Language »
    - Langage de description de services web
    - Paramètres, type du résultat, opérations fournies par le service, points d'accès
  - UDDI : « Universal Description, Discovery and Integration »
    - Protocole de description et d'interaction avec des annuaires de services web
- Autres aspects: énormément de standards
  - Orchestration, composition: WSBPEL, WS-Coordination, WS-CDL
  - Sémantique: OWL-S, WSDL-S
  - Sécurité: WS-Security
  - …

SOAP

- Simple Object Access Protocol, norme W3C
  - SOAP 1.0: 1999, basé sur HTTP
  - SOAP 1.1: 2000, plus générique, autres protocoles
  - SOAP 1.2: recommandation W3C
- Couvre 4 aspects
  - Format XML des messages échangés
  - Comment un message SOAP est transporté sur le web par HTTP, SMTP.
  - Règles de traitement des messages SOAP
  - Conventions de transformation d'un appel RPC en SOAP et d'implémentation d'une communication RPC

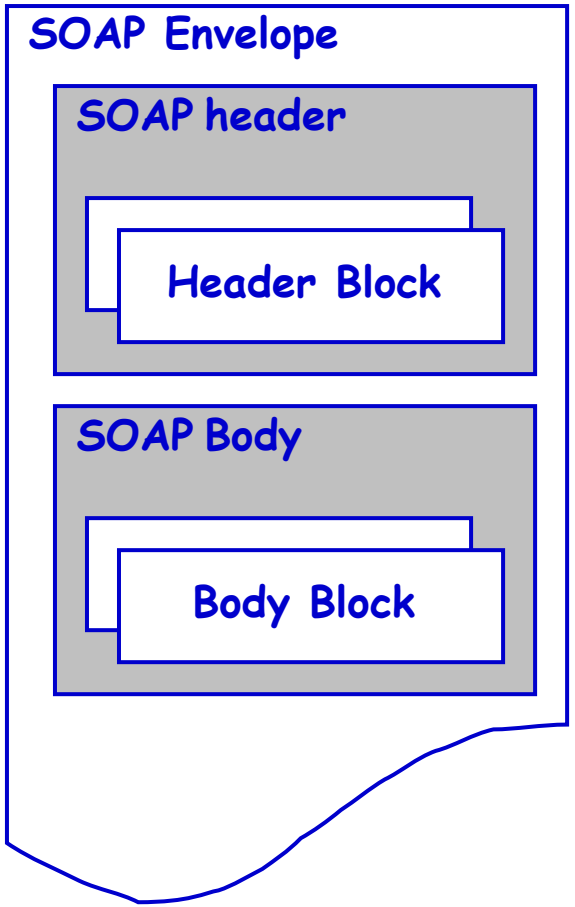
SOAP: objectifs

- But initial : infrastructure minimale pour faire du RPC sur le web
  - Utilisation de XML pour les échanges
  - Structure de messages très simple
  - Basé sur HTTP pour résoudre le problème des pare-feux
- Raisons pratiques
  - Éviter les problèmes de CORBA sur le web, qui doit en pratique s'appuyer de toute façon sur HTTP à cause des pare-feux
  - Disposer d'une couche facile à mettre en œuvre au-dessus des plateformes middleware pour une intégration à travers le web
  - Transformer SOAP par la suite en support générique d'échange de messages sur le web, au-delà de RPC et de HTTP.

Structure des messages SOAP

- Messages : « enveloppes » où l'application met les données à transmettre

- Structure
  - En-tête (optionnel)
    - Niveau infrastructure
  - Corps (obligatoire)
    - Niveau application



Exemple

```
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
         <SOAP-ENV:Header>
            <t:Transaction
               xmlns:t="some-URI"
               SOAP-ENV:mustUnderstand=“true">
                  5
            </t:Transaction>
         </SOAP-ENV:Header>

         <SOAP-ENV:Body>
            <m:GetLastTradePrice xmlns:m="Some-URI">
               <symbol>DEF</symbol>
            </m:GetLastTradePrice>
         </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

En-tête SOAP

- Conteneur pour information indépendante de l'application
  - Information de sécurité, de coordination, identificateurs de transaction, …
  - Ensemble de blocs (éléments XML)
  - Chaque nœud sur le parcours source → destination peu traiter des blocs
- Qui traite les blocs d'en-tête?
  - Attribut « role »: qui doit traiter le bloc
    - "none" : pas besoin de le traiter
    - "next" : prochain nœud
    - "ultimateReceiver" : seul le destinataire final (par défaut)
  - Attribut « mustUnderstand »: obligation ou non de traiter l'élément
    - "true" : traitement obligatoire
    - "false" : traitement optionnel
  - Attribut « relay » (SOAP 1.2): un bloc non traité doit être transmis au prochain nœud

Cheminement des messages SOAP

- Un message source → destination : chemin dans le réseau
  - Objectif: tenir compte de l'architecture du réseau (ex. middleware)
  - Attribut « role » dans chaque bloc d'en-tête (par défaut: "ultimateReceiver")
  - Chaque nœud dans le chemin regarde chaque bloc d'en-tête
    - Il traite (éventuellement) les blocs d'en-tête qui lui reviennent
      - Traitement obligatoire des parts avec « mustUnderstand » = "true"
    - Enlève ces blocs de l'en-tête sauf pour les blocs non traités où l'attribut « relay » est présent
    - Retransmet le message avec l'en-tête mise-à-jour vers le nœud suivant
    - Si erreur, arrêt du cheminement et retour d'un message SOAP d'erreur
- Les modifications en chemin: seulement pour l'en-tête
- Le corps du message est traité seulement par le destinataire
  - Similaire aux blocs d'en-tête avec « role » = "ultimateReceiver"
- Remarque: il existe aussi une notion d' « intermédiaire actif », qui peut modifier le message reçu d'une façon dépendante de l'application

Corps SOAP

- Conteneur pour données spécifiques à l'application
- Divisé en blocs
  - Un bloc : traité seulement par le récepteur final
  - Certains blocs ont une signification prédéfinie
    - Pour traduire un appel RPC en SOAP
    - Pour signaler des erreurs
- Bloc d'erreur (« Fault ») : rapport d'erreur dans la traitement du message
  - Code: catégorie d'erreur (version, mustUnderstand, client, server)
  - Texte: explication textuelle, à afficher
  - Acteur: nœud à l'origine de l'erreur
  - Détail: information dépendante de l'application
  - D'autres éléments rajoutés dans SOAP 1.2

Appel RPC en SOAP

- RPC: deux messages SOAP (appel + réponse)
  - Message d'appel:
    - Élément : nom de fonction à appeler
    - Sous-éléments: paramètres d'appel de la fonction
  - Message de retour: deux possibilités
    - Résultat de l'appel OU
    - Élément « Fault » pour signaler une erreur de traitement
- Généralement réalisé par HTTP (mais pas nécessairement)
  - Appel: GET/POST
  - Retour: réponse HTTP

Exemple

- Message d'appel

```
<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>
    <add soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
          <op1 xsi:type="xsd:int">2</op1>
          <op2 xsi:type="xsd:int">5</op2>
    </add>
  </soapenv:Body>
</soapenv:Envelope>
```

- Message réponse

```
<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>
    <addResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
          <addReturn xsi:type="xsd:int">7</addReturn>
    </addResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

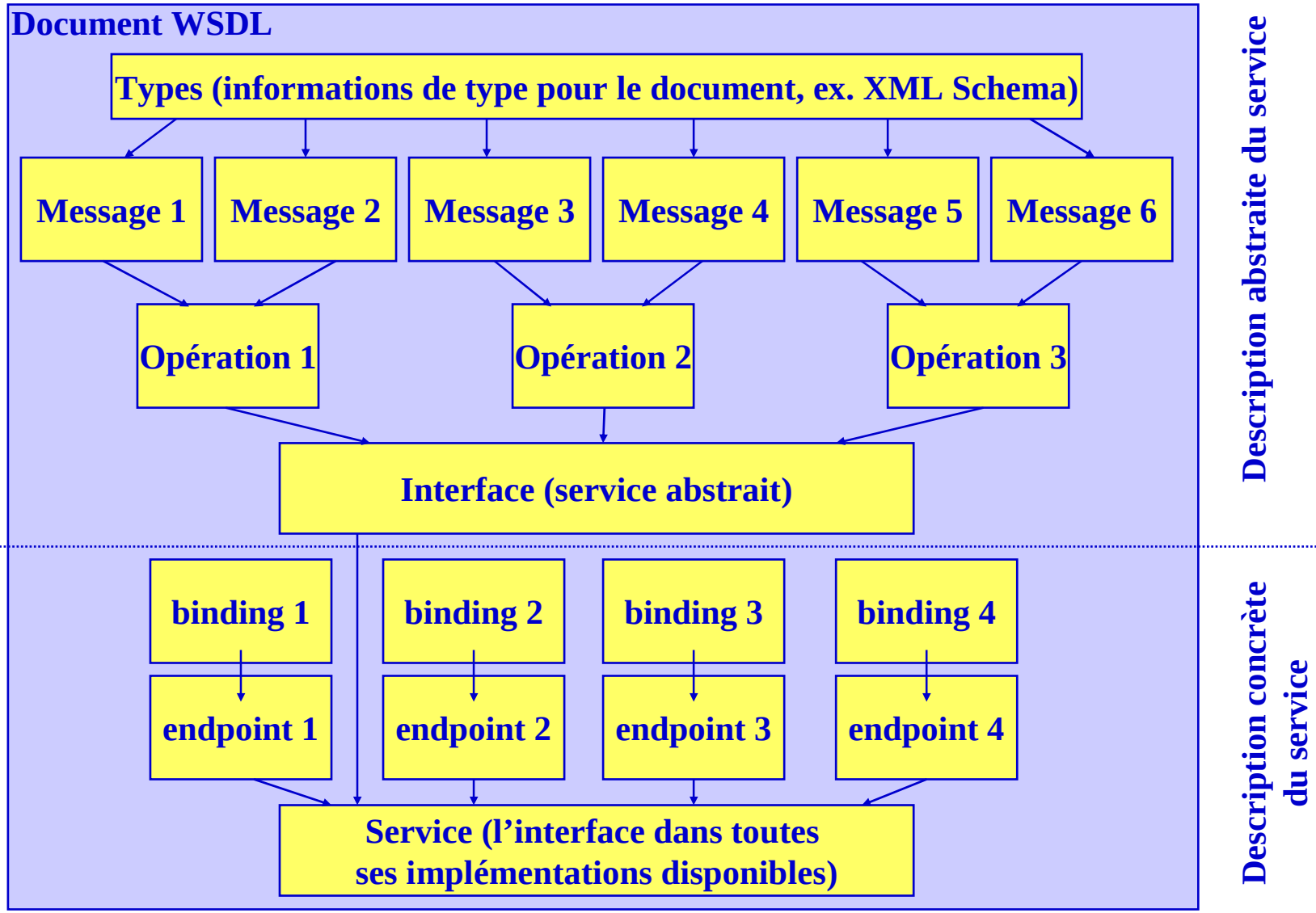
SOAP et HTTP

- « Binding » SOAP
  - Description de la façon dont les messages SOAP sont envoyés en utilisant un protocole de transport donné
  - « Binding » typique pour SOAP : HTTP
- SOAP dans HTTP: utilise GET ou POST
  - GET: l'appel n'est pas un message SOAP, seule la réponse l'est
  - POST (préféré): l'appel et la réponse sont des messages SOAP
- Inclusion dans les messages HTTP
  - Appel POST: enveloppe SOAP = contenu du message POST
  - Réponse: enveloppe SOAP = contenu de la réponse
  - SOAP utilise les mêmes codes d'erreur et d'état que HTTP → une réponse HTTP peut être directement interprétée par un module SOAP

WSDL

- Web Services Description Language
  - Description des différentes parties d'un service web
- Description à deux niveaux
  - Abstrait
    - Les types (XML Schema) des paramètres et des résultats des messages
    - Les messages manipulés dans le service
    - Les opérations individuelles: suite d'échange de messages
    - Une interface de service abstrait, qui groupe les opérations individuelles
  - Concret
    - Le « binding » de l'interface (des opérations) à un protocole de transport
    - Les points d'accès (adresses réseau) pour chaque opération
    - Service = ensemble de « bindings » avec leurs points d'accès

Éléments WSDL



Source: G. Alonso

Version courante: WSDL 2.0

- WSDL 2.0 (2006)
  - Points d'accès ("endpoints")
  - Interfaces
  - Héritage d'interfaces
  - Redéfinition d'opérations enlevée
  - Messages définis par des types
  - Opérations définies dans les interfaces
  - Points d'accès définis dans les "Bindings"
  - 8 motifs d'échange de messages
  - Quelques nouveaux éléments
- WSDL 1.1 (2001)
  - Ports
  - "PortTypes"
  - Redéfinition d'opérations
  - Messages composés de parts
  - 6 éléments de premier niveau: Messages, Opérations, « PortTypes », « Bindings », Ports et Services
  - 4 primitives de transmission : « One-way », « Request-Response », « Solicit-Response », « Notification »

Exemple : service de réservation

```
<description xmlns="http://www.w3.org/2006/01/wsdl"
   targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
   xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
   xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
   xmlns:wsoap= "http://www.w3.org/2006/01/wsdl/soap"
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
<documentation> This document describes … </documentation>
<types>
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://greath.example.com/2004/schemas/resSvc"
         xmlns="http://greath.example.com/2004/schemas/resSvc">
        <xs:element name="checkAvailability" type="tCheckAvailability"/>
        <xs:complexType name="tCheckAvailability">
          <xs:sequence>
            <xs:element name="checkInDate" type="xs:date"/>
            <xs:element name="checkOutDate" type="xs:date"/>
            <xs:element name="roomType" type="xs:string"/>
        </xs:sequence>
        </xs:complexType>
        <xs:element name="checkAvailabilityResponse" type="xs:double"/>
        <xs:element name="invalidDataError" type="xs:string"/>
      </xs:schema>
</types>
```

Exemple (suite)

```
<interface name = "reservationInterface" >
      <fault name = "invalidDataFault" element = "ghns:invalidDataError"/>
      <operation name="opCheckAvailability"
         pattern="http://www.w3.org/2006/01/wsdl/in-out"
         style="http://www.w3.org/2006/01/wsdl/style/iri">

         <input messageLabel="In" element="ghns:checkAvailability" />
         <output messageLabel="Out" element="ghns:checkAvailabilityResponse" />
         <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
      </operation>
</interface>
```

- Interface : définition abstraite du service
  - Composée d'opérations et d'erreurs
  - Peut hériter d'autres interfaces
- Opérations
  - Ensemble de messages et d'erreurs
  - Enchaînement des messages défini par un *motif* (« pattern »)
  - Style d'opération: contraintes sur l'opération et les messages
    - RPC, IRI (internationalized resource identifier), etc.

Motifs d'échange de messages

- Motifs de base
  - IN-ONLY : un seul message d'entrée, sans erreurs
  - ROBUST IN-ONLY : pareil, mais avec erreur possible
  - IN-OUT
    - Message d'entrée reçu en provenance d'un noeud N
    - Message de sortie envoyé au noeud N
    - Erreurs, qui si elles apparaissent, remplacent le message de sortie
- Motifs supplémentaires (en dehors de la norme)
  - IN-OPTIONAL-OUT : pareil, mais le message de sortie est optionnel
  - OUT-ONLY : un seul message de sortie, sans erreurs
  - ROBUST OUT-ONLY : pareil, mais avec erreur possible
  - OUT-IN
    - Message de sortie envoyé au noeud N
    - Message d'entrée reçu en provenance d'un noeud N
    - Erreurs, qui si elles apparaissent, remplacent le message d'entrée
  - OUT-OPTIONAL-IN : l'opposé de IN-OPTIONAL-OUT

Exemple (fin)

```
<binding name="reservationSOAPBinding"
    interface="tns:reservationInterface"
    type="http://www.w3.org/2006/01/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">

        <fault ref="tns:invalidDataFault" wsoap:code="soap:Sender"/>
        <operation ref="tns:opCheckAvailability"
            wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
</binding>

<service name="reservationService" interface="tns:reservationInterface">
        <endpoint name="reservationEndpoint"
            binding="tns:reservationSOAPBinding"
            address ="http://greath.example.com/2004/reservation"/>
 </service>

</description>
```

- « Binding »: format des messages et détails de protocole par opération
  - Une même opération peut avoir plusieurs bindings
- Service: ensemble de points d'accès = couples (binding, adresse réseau)

UDDI

- Universal Description, Discovery and Integration
- Historique
  - À l'origine: annuaire universel pour les services web (à la Google)
  - Aujourd'hui: vise plutôt les environnements privés, à petite échelle
  - Raisons: peu d'annuaires généraux UDDI (IBM, Microsoft, …), contenu pauvre et non fiable
    - Meilleure fiabilité en environnements contraints, privés (~EAI)
    - Élément d'infrastructure qui aide aussi à stocker des infos absentes en WSDL
- Versions
  - Version 1: les bases d'un annuaire de services
  - Version 2: adaptation à SOAP et WSDL
  - Version 3: redéfinition du rôle UDDI, accent sur les implémentations privées, sur l'interaction entre annuaires privés et publics

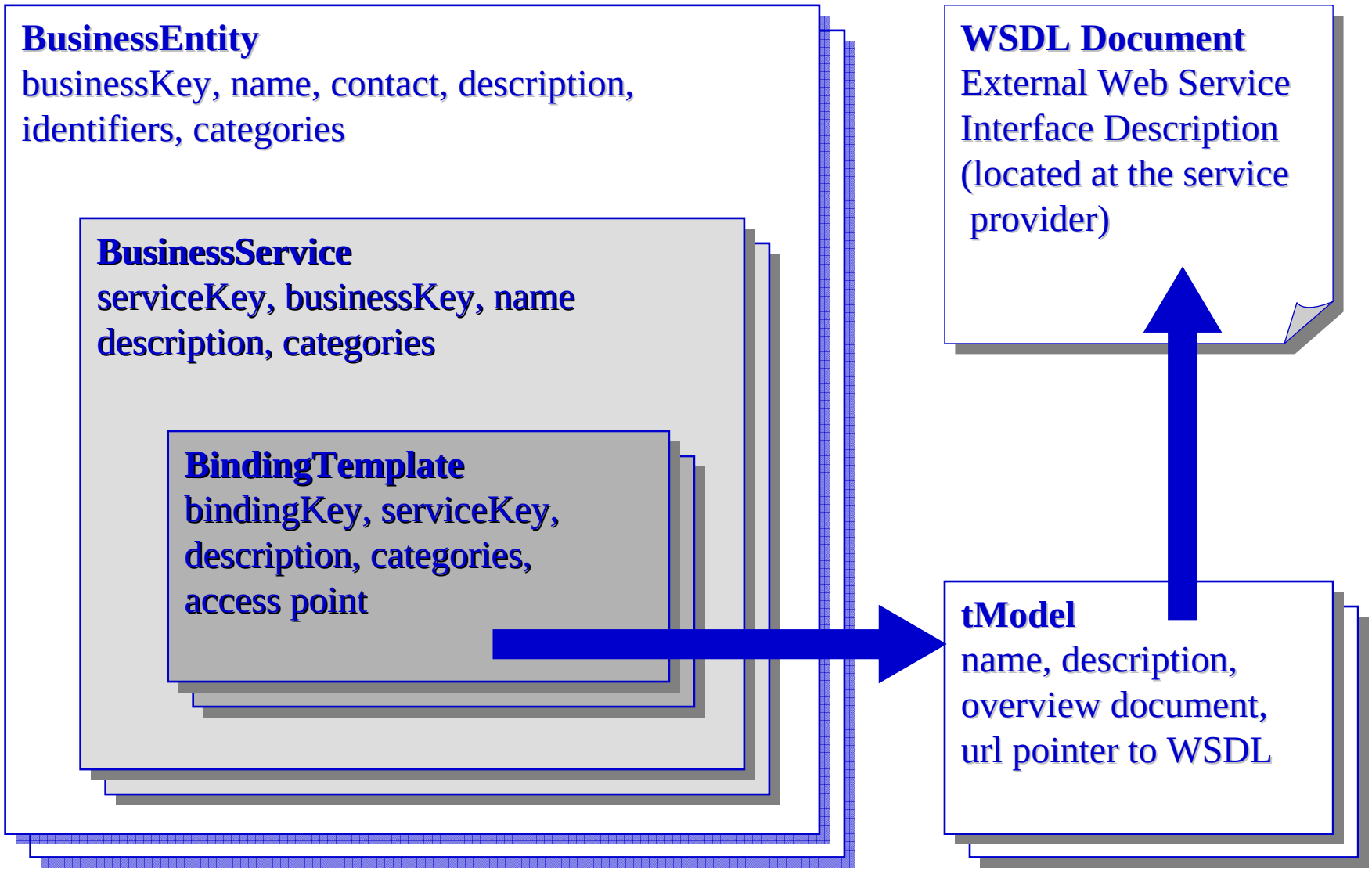
Modèle de données UDDI

- Entrée d'annuaire UDDI = document XML composé d'éléments
  - *businessEntity*: organisation qui offre le service
  - *businessService*: liste des services web offerts par l'organisation
  - *bindingTemplate* : aspects techniques du service offert
  - *tModel*: élément générique pour info supplémentaire sur le service
- Types d'information
  - Pages blanches: données sur le fournisseur du service (nom, adresse, …)
  - Pages jaunes: classification du type de service, basée sur des standards
  - Pages vertes: info technique sur l'utilisation du service
    - Pointeurs sur les descriptions WSDL, qui ne font pas partie de l'annuaire

Éléments UDDI

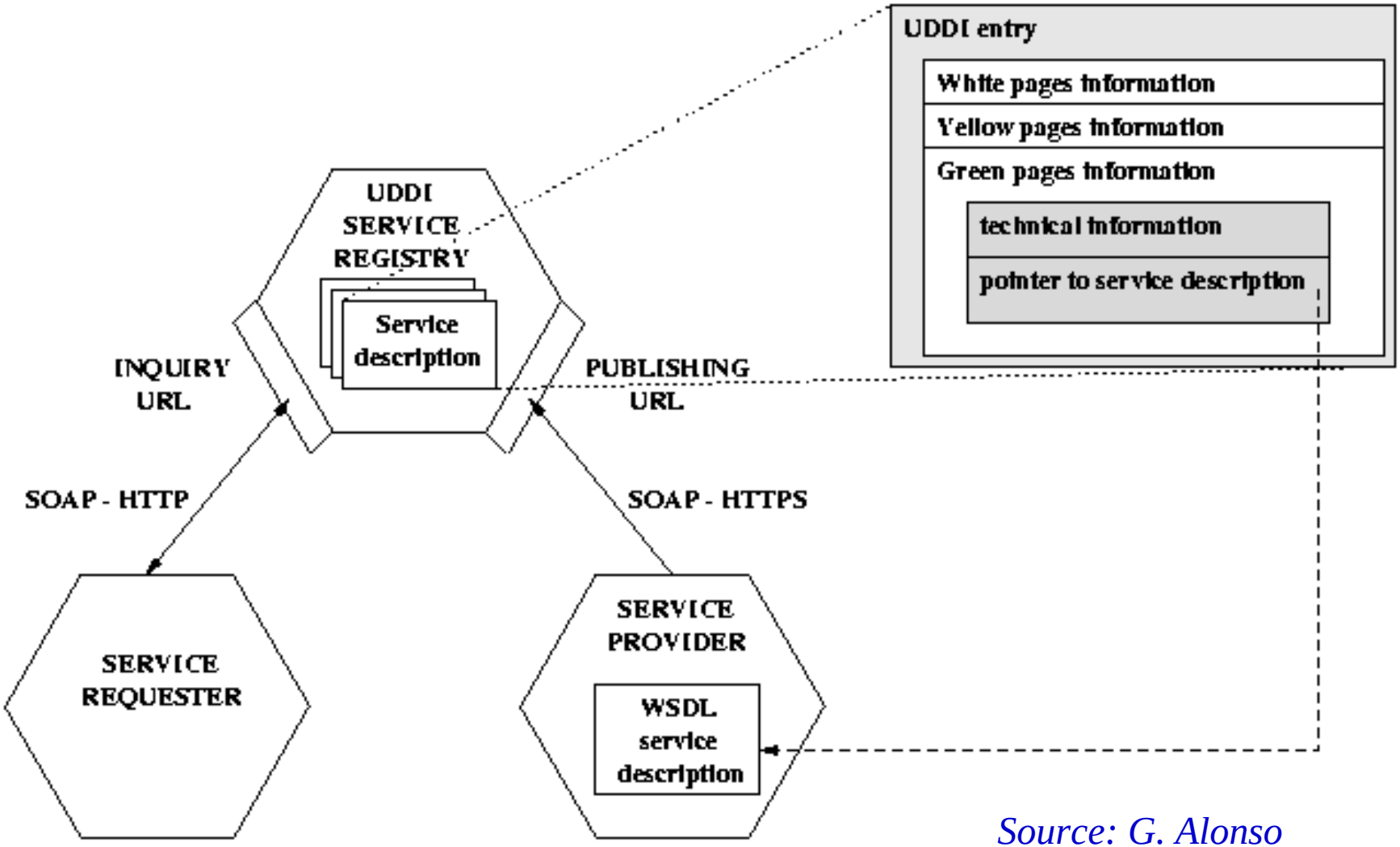
- Business entity
  - Infos génériques de pages blanches et jaunes
  - Clé entité, nom, description, contacts, catégories, etc.
- Business service
  - Description d'un ou plusieurs services offerts par un fournisseur
  - Inclus dans le « business entity » du fournisseur
  - Clé service, nom, description, liste de « binding templates »
- Binding template
  - Infos techniques pour un service
  - Clé binding, clé service, description, adresse réseau, liste tModels
- tModel
  - Info technique: pointeur vers la description WSDL, infos sur le protocole
  - Clé tModel, nom, description, catégories, liste d'autres tModels liés

Schéma du modèle de données UDDI



Source: G. Alonso

UDDI, WSDL et SOAP



Source: G. Alonso

Interaction avec UDDI

- APIs pour l'accès à UDDI
  - UDDI Inquiry: rechercher des entrées UDDI dans l'annuaire (mots clés)
  - UDDI Publication: publier et modifier des entrées UDDI dans l'annuaire
  - UDDI Security: contrôle d'accès à l'annuaire
  - UDDI Subscription: souscription à des modifications d'entrées UDDI
  - UDDI Replication: dupliquer des entrées sur plusieurs nœuds
  - UDDI Custody and Ownership transfer: modifier le propriétaire d'une entrée UDDI
  - UDDI Subscription Listener: pour le client qui souscrit aux modifications
  - UDDI Value Set: pour valider l'information à publier dans l'annuaire