XML et XML schéma : représentation des données dans les services Webs

Gérard Florin
Laboratoire CEDRIC
CNAM Paris

Introduction aux services webs

- **Le besoin :** des communications de programme à programme utilisant comme support la toile mondiale.
- **Service toile (service web)**: une application informatique quelconque (plutôt de gestion ou de commerce électronique).
  - modulaire,
  - basée sur la toile,
  - utilisant des services et protocoles standards.
- **Notion de fournisseur de services** sur la toile: (ASP 'Application Service Providers') au moyen des services webs.
- **Applications visées:** commerce électronique puis par extension tout service comportant un accès à des données (température, trafic, ...) ou une algorithmique plus ou moins complexe (exemple calcul intensif).

Organisation actuelle des standards

UDDI Services d'annuaire	Ws-Security Services de sécurité	WS-Transaction Services de transaction	Bpel4WS, .. Services de synchronisation

SOAP, WSDL : Interactions de communication

HTTP, SMTP, MOM(JMS) : Transmission effective

SOAP
(‘Simple Object Access Protocol’)

- Le protocole de **communication** de base (définissant la principale interaction de communication).
- **Une approche de mode message** mais aussi un protocole d’appel de procédure distante (**RPC**).
- **Utilise XML** pour représenter les données.
- **Utilise des protocoles applicatifs Internet** pour acheminer ses messages:
  - **HTTP** (pour sa généralité, son mode synchrone).
  - **SMTP** , **MOM** (plutôt pour le mode asynchrone).

WSDL
(‘Web Services Definition Language’)

- **Un standard de description d’interfaces (analogue de IDL).**
- Permet de décharger les utilisateurs des **détails techniques** de réalisation d’un appel.
- **Basé sur XML, XML Schéma**.
- Règles de **sérialisation des données** échangées.

UDDI ‘Universal Description, Discovery, and Integration’

- Le standard **d’annuaire réparti** pour la description des services Web.
- Très orienté **affaires** (ventes, prestations)
- **Accessible au moyen de Soap**.
- Permet d enregistrer des informations variées via la notion de **tmodel** (modèle technique).

Acteurs du domaine

- **Consortiums/organismes**
  - W3C World Wide Web Consortium
  - Oasis
  - WS-I Web Services - Interoperability
  - Nations Unies
  - EbXML
  - Rosetta net : Web services en électronique
- **Editeurs de logiciels**
  - Microsoft, IBM, BEA
- **Logiciel libre**
  - Apache Axis

Exemples de produits

- **Microsoft** .NET
- **IBM** Web Services Architecture (Websphere)
- **SUN** One
- **BEA** Web Services Architecture (Weblogic)
- **Iona**, **CapeClear**, **SilverStream**, **Systinet**

- Logiciel libre **Apache** Axis (WSIF ‘ Web Services Invocation Framework ’)

La représentation des données

Origines : les formats de documents SGML, HTML

SGML (‘Standard Generalized Markup Language’) - 1986

- Un langage de balisage **très général** pour définir des documents électroniques, **indépendants de la forme visualisée**.
- Définir le contenu d'un document (rapport technique, livre, courrier, ...).
- Un contenu est **transformé en autant de formats imprimés** que nécessaire**.**
- Définir par ailleurs la forme imprimée.
- Le but principal reste de **transmettre, stocker des documents à imprim**er avec économies de volume et souplesse de formatage.
- **SGML est complexe = > son usage est restreint à des spécialistes de la gestion documentaire.**

HTML (‘HyperText Markup Language’) - 1989

- **Développement de la documentation accessible en ligne**

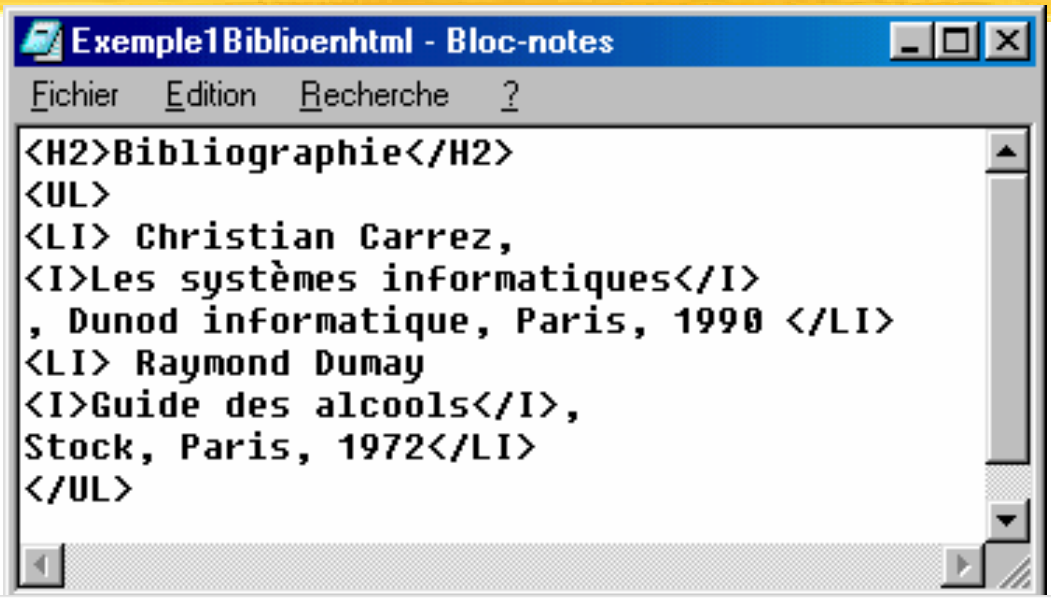
  => Avantage des liens **hypertextuels**.
- Première études sur **le langage de balises hypertextes HTML** 1989 conçu à partir de SGML comme une version très simplifiée.
- Disponibilité et **première normalisation** en 1992
- **Démonstration de la très grande capacité** d'accès à l'information rendue possible par le WEB (HTML + HTTP + URL).

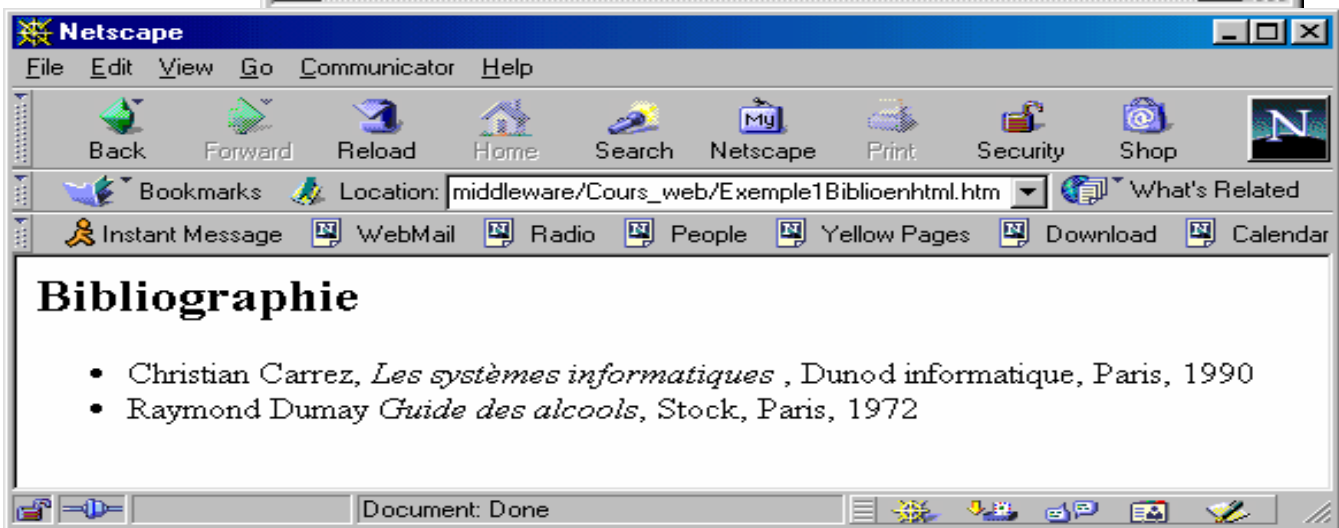
Quelques caractéristiques de HTML

- HTML est un **langage de balises** comme SGML :
  - Exemple <H2>
- Normalisation **d'un ensemble figé** de symboles encadrés par "<" et ">".
- Les balises HTML définissent **des directives de mise en page du text**e encadré par un couple balise ouvrante, balise fermante:
  - **<H2> xxxx </H2>** un titre de niveau 2,
  - **<UL> yyyy </UL>** une liste d'items (sans numéros d'item),
  - **<LI> zzzzzz </LI>** un item de la liste,
  - **<I> ttttttttt </I>** un texte en italique.
- **Pas de type de document** (pas de structuration) prédéfinie permettant de définir abstraitement une structure de document

Exemple d'une bibliographie en HTML:

- Une structuration de document complètement **orientée vers la présentation graphique.**





Quelques caractéristiques de HTML

- HTML est un **langage de balises** comme SGML :
  - Exemple <H2>
  - Balises: normalisation **d'un ensemble figé** de symboles encadrés par "<" et ">".
- Les balises HTML définissent **des directives de mise en page d'un texte** encadré par un couple balise ouvrante, balise fermante:
  - **<H2> xxxx </H2>** un titre de niveau 2,
  - **<UL> yyyy </UL>** une liste d'items (sans numéros d'item),
  - **<LI> zzzzzz </LI>** un item de la liste,
  - **<I> ttttttttt </I>** un texte en italique.
- **Pas de type de document** (pas de structuration) prédéfinie permettant de définir abstraitement une structure de document

Discussion des choix de HTML (1)

- **HTML: un langage trop graphique**
  - A l'origine HTML définissait des balises **indépendantes de la représentation** graphique des informations (Ex: UL définit une liste).
  - Logiquement, la présentation graphique d'une liste ne doit pas être toujours la même: elle **dépend d'une adaptation du navigateur à un contexte**.
    - Ex: présenter des informations en Braille pour des non voyants.
  - Au fil du temps les fabricants ont introduits pour leurs **besoins des éléments graphiques**. (Ex: I pour italique, FONT, CENTER, BGCOLOR. De même UL a pris le sens d'une indentation, pas d'une liste.
- **Le balisage HTML est devenu spécifique des différents navigateurs pour les besoins d'affichage**

  => HTML est en fait un langage graphique pour Explorer ou Mozilla Firefox.

Discussion des choix de HTML (2)

- **HTML: un langage non extensible**
  - Le langage HTML a été conçu pour être **assez simple** de sorte que le nombre et la signification des balises est **limité**.
    - Ex: Il n'existe pas de balisage pour la représentation des données en chimie (molécules, formules, valeurs numériques)
  - Le langage HTML est **figé**.
    - Toutes les balises utilisables sont définies au départ ce qui est intenable si l'on voulait prendre globalement en compte les besoins d'un grand nombre de métiers.
- **HTML est figé et assez simple : pas de possibilité pour un utilisateur de base d'introduire de nouvelles balises.**

Discussion des choix de HTML (3)

- **HTML: un langage de description de documents non structurés**
  - HTML permet de définir de façon beaucoup trop **limitée** la **structure** d'un document.
  - Il n'y a en fait pas de **vérification d'une structure** pour le document que l'on peut définir.
    - Ex: on peut créer un document commençant par une tête de chapitre H2 et poursuivant par une tête de chapitre H1.
- **HTML définit en fait un univers de documents plats.**
  - Une recherche doit considérer un document HTML comme **une chaîne de caractères**.
  - Pas de moyen de partager entre communicants **une structure de document** préétablie.
- **HTML ne type pas les documents.**

Conclusion : avenir de HTML

- **HTML peut rester très longtemps utilisé comme langage graphique pour les navigateurs WEB.**
- **HTML présente des limitations trop importantes pour rester à terme le support de la représentation des données échangées sur le WEB.**
  - Non **séparation** du document et de sa présentation graphique.
  - Non **extensibilité**.
  - Non **structuration**, **typage**.

XML et le typage par les DTD

Introduction XML : Héritage SGML/HTML

- **SGML** riche, lourd, mal adapté au Web.
- **HTML** adapté à la présentation graphique, mais limité par un ensemble de balises figé, non extensible et sans typage.
- **Groupe de travail XML** à partir de août 1996 qui est composé essentiellement de membres du groupe de travail SGML.
  - **Recherche d'un langage assez simple** mais présentant une richesse proche de SGML.
  - **XML : un sous-ensemble de SGML**, qui élimine des points trop ciblés sur certains besoins.
  - Un document XML est **conforme** SGML.

Conséquence sur XML

- **Extensibilité**: pouvoir définir de nouvelles balises.
- **Structuration** : pouvoir modéliser des données d'une complexité quelconque.
- **Validation** : pouvoir vérifier la conformité d'une donnée avec un type (un modèle de structure).
- **Indépendance du média**: pouvoir formater un contenu selon des représentations diverses.
- **Interopérabilité** : Pouvoir échanger et traiter une donnée en utilisant de nombreux types de logiciels.

Caractéristiques de XML (1) : un langage de niveau méta

- **XML n'est pas un langage de balises de plus** (comme HTML).
- **XML permet de définir de nouveaux langages de balises** (comme SGML).
  - **Notion de langage de niveau méta**
- Un langage qui permet de **définir des langages**.

Caractéristiques de XML (2) : Définition d'une syntaxe

- **En XML on définit une structure aussi variée que possible de balises mais qui n'est associée à aucun comportement (pas de sémantique).**
  - XML s'apparente **à la BNF 'Backus Naur Form'**
- **Le comportement doit venir d'ailleurs.**
  - Dans la publication de documents il s'agit de feuilles de styles ("style sheets" = **formes déclaratives** de présentation graphique**)**
  - Dans d'autres domaines il pourra s'agir d'une application spécifique définie par un composant logiciel jouant le rôle **d'interpréteur XML.**

Caractéristiques de XML (3) : un langage verbeux

Choix délibéré: XML définit tout en format caractère (lisible par un être humain).

- **Avantage :** pour le **développement** des codes, **la mise au point, l'entretien** et pour **l'interopérabilité** (choix des caractères UCS 'Universal Character Set' ISO Unicode).
- **Inconvénient:** Les données codées en XML sont **toujours plus encombrantes** que les mêmes données codées en binaires.
  - L'espace mémoire, l'espace disque et la bande passante sont devenus **moins chers**.
  - Les **techniques de compression** sont accessibles **facilement et gratuitement**.
  - Elles sont applicables **automatiquement** dans les communications par modems ou en HTTP/1.1 et **fonctionnent bien** avec XML.

Structure d'un document XML (1) : structure logique

Un document XML est composé de trois parties:

- **Un prologue** qui comporte:
  - Des déclarations diverses
  - Des instructions de traitement (optionnelles)
  - Une déclaration de type du document (optionnelle)
- **Un ensemble d'éléments** organisés en arbre (les balises).
- **Des commentaires** <!-- Un commentaire -->

Structure d'un document XML (2) : structure physique

- Un document est composé d'unités de source appelées **entités** (des blocs de caractères).
- Les **entités sont emboîtées** les unes dans les autres en commençant par l'entité racine (document) => l'ensemble forme un arbre.
- Une entité peut référencer une autre entité pour forcer **l'inclusion** à la place dans le document.
- Une entité peut être **nommée** (avoir un alias).
- Une entité peut être **interne** (son nom est alors un raccourci pour sa valeur).
- Une entité peut être **externe** et représenter un fichier local ou distant contenant du texte XML.

Corps d'un document XML

- **XML définit un cadre pour baliser n'importe quel document structuré en arbre (balises ou 'tags')**
- **Notion d'élément XML (associé à une balise)**: définit une donnée sous la forme d'une chaîne de caractères comme ayant un sens (message, de, pour, objet, texte)
- **Notion d'attribut XML (d'un élément)**: pour associer une donnée à une balise (localisation, codage).
- **Exemple simple:**

```
<message securite="secret défense">
        <de>Jean</de>
        <pour>Jacques</pour>
        <objet>Rappel</objet>
        <texte>On se voit demain à 10h</texte>
</message>
```

Exemple d'une bibliographie en XML

Exemple2BiblioenXML - Bloc-notes

Fichier Edition Recherche ?

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<BIBLIO SUBJECT="Un exemple">

<BOOK LANG="fr" SUBJECT="Informatique">
<AUTHOR> <FIRSTNAME>Christian</FIRSTNAME>
<LASTNAME>Carrez</LASTNAME> </AUTHOR>
<TITLE>Les systèmes informatiques</TITLE>
<PUBLISHER> <NAME>Dunod Informatique</NAME>
<PLACE>Paris</PLACE> </PUBLISHER>
<DATEPUB>1990</DATEPUB> </BOOK>

<BOOK LANG="fr" SUBJECT="Gastronomie">
<AUTHOR>  <FIRSTNAME>Raymond</FIRSTNAME>
<LASTNAME>Dumay</LASTNAME> </AUTHOR>
<TITLE>Guide des alcools</TITLE>
<PUBLISHER>  <NAME>Stock</NAME>
<PLACE>Paris</PLACE></PUBLISHER>
<DATEPUB>1972</DATEPUB> </BOOK>

</BIBLIO>
```

L’arbre associé à une bibliographie

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <BIBLIO SUBJECT="Un exemple">
  - <BOOK LANG="fr" SUBJECT="Informatique">
    - <AUTHOR>
        <FIRSTNAME>Christian</FIRSTNAME>
        <LASTNAME>Carrez</LASTNAME>
      </AUTHOR>
      <TITLE>Les systèmes informatiques</TITLE>
    - <PUBLISHER>
        <NAME>Dunod Informatique</NAME>
        <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1990</DATEPUB>
    </BOOK>
  - <BOOK LANG="fr" SUBJECT="Gastronomie">
    - <AUTHOR>
        <FIRSTNAME>Raymond</FIRSTNAME>
        <LASTNAME>Dumay</LASTNAME>
      </AUTHOR>
      <TITLE>Guide des alcools</TITLE>
    - <PUBLISHER>
        <NAME>Stock</NAME>
        <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1972</DATEPUB>
    </BOOK>
  </BIBLIO>
```

Documents bien formés

- **Si un document XML respecte les règles de la grammaire XML on dit qu'il est bien formé.**
- **Les règles d'un document bien formé**
  - Toute balise **ouverte** doit être **fermée**, Ex: <livre> </livre>
  - L'ensemble des balises est **correctement imbriqué**. Ex : Entrelacement mal formé   ....
  - Les valeurs d'attributs sont entre **guillemets**"
- Les **balises uniques correspondent à des documents vides** et sont notées: Ex : <IMG source="image.gif"/>
- Les caractères < & sont notés < &
- Un document commence par une **déclaration XML**
  - <?xml version="1.0" encoding ="iso-8859-1" standalone="yes"?>

Documents valides : DTD

- **Déclaration de type de document : DTD 'Document Type Definition** ’ une notion héritée de SGML permettant la définition formelle d’un type de document.
- Une information de niveau méta pour qu'un analyseur puisse vérifier la **conformité** d’un document à une spécification de type (notion de document **valide**).
  - **intitulé** des balises et de leurs **attributs**,
  - **imbrication** des balises (structure d’arbre du document),
  - **caractère obligatoire / facultatif** de certaines infos**.**
- Pour un document XML une DTD est **optionnelle**:
  - **En son absence**: on constate une structure.
  - **En sa présence**: on contraint la conformité d'un document XML.

DTD : Déclarations de types d'éléments de base (1)

- <!**ELEMENT** nom_élément **(**contenu_élément**)** >
  - **Le nom de l'élément** le nom de la balise dans un document.
  - Contenu: **un type de base** (chaîne de caractères avec des variantes) ou un type construit (en fait une séquence d'éléments).
    - **EMPTY : Chaîne vide**
      - <! ELEMENT elem (EMPTY)>
      - Eléments vides: support d'attributs
    - **#CDATA : Chaîne non analysée** La chaîne est supposée contenir des données qui n'ont pas à être parsées.
    - **#PCDATA : ('Parsed Character DATA')**
    - Chaîne de caractères devant être parsée.
    - **ANY : Une chaîne quelconque.**

DTD : Déclarations de types d'éléments construits (2)

- **Élément avec fils : Types d'éléments construits à partir d'autres types d'éléments (types article).**
- On peut préciser la structure (séquence) des fils au moyen **de constructeurs.**
  - Nom_élément — une seule occurrence
  - Nom_élément***** — 0, 1 ou n occurrences
  - Nom_élément**?** — 0 ou 1 occurrence (valeur optionnelle)
  - Nom1**|**Nom2 — alternative
  - **,** **( )** — séquence et parenthèses
- **Types éléments 'mixtes' :** des éléments et des chaînes
  - <!ELEMENT mémoire (titre, résumé, remerciements?, introduction, chapitre+, conclusion, bibliographie,#PCDATA) >

DTD : Déclarations de types d'attributs

- **Les listes d'attributs 'attlist'** définissent pour un élément la liste des attributs de cet élément avec leurs propriétés.
  - Nom de l'attribut, Type de l'attribut, Valeur par défaut.
    - <!ATTLIST nom_element
    - nom_attribut1 type1 défaut1
    - ...
    - nom_attributn typen défautn>
- **Le paramètre défaut**
    - **#REQUIRED**:attribut obligatoire à fournir.
    - **#IMPLIED** : attribut optionnel.
    - **#FIXED 'value'**: attribut de valeur invariable égale à 'value'.
    - **'value'**: une valeur pour l'attribut quand elle n'est pas donnée.

DTD : Typage des d'attributs (1)

- Trois catégories de types: chaînes, listes de chaînes, valeurs énumérées.
- **CDATA :** Les valeurs possibles sont des chaînes de caractères non analysées par le parseur.
  - Ex: <!ATTLIST fichier cheminaccès CDATA #REQUIRED>
- **ID** : Les valeurs sont des identifiants uniques.
  - Ex: <!ATTLIST dossier ident ID #REQUIRED nom CDATA #IMPLIED>
- **IDREF, IDREFS** : Les valeurs possibles sont des noms d'identifiants uniques (resp des listes).

DTD : Typage des d'attributs (2)

- **ENTITY, ENTITIES** : la valeur est un nom d'entité déclaré et existant (resp une liste de noms).
- **NMTOKEN, NMTOKENS** : la valeur est obligatoirement un mot clé valide de xml (resp une liste de mots clés).
- **NOTATION:** La valeur est le nom d'une notation.
- **Valeurs énumérées** : Soit des notations de valeurs déjà définies soit des valeurs.

  Séparation par des barres (val1|val2|...)

DTD : Exemple d'un rapport Les éléments

```
<!DOCTYPE REPORT [
<!ELEMENT REPORT (TITLE,(SECTION|SHORTSECT)+)>
<!ELEMENT SECTION (TITLE,%BODY;,SUBSECTION*)>
<!ELEMENT SUBSECTION (TITLE,%BODY;,SUBSECTION*)>
<!ELEMENT SHORTSECT (TITLE,%BODY;)>
<!ELEMENT TITLE %TEXT;>
<!ELEMENT PARA %TEXT;>
<!ELEMENT LIST (ITEM)+>
<!ELEMENT ITEM (%BLOCK;)>
<!ELEMENT CODE (#PCDATA)>
<!ELEMENT KEYWORD (#PCDATA)>
<!ELEMENT EXAMPLE (TITLE?,%BLOCK;)>
<!ELEMENT GRAPHIC EMPTY>
```

DTD : Exemple d'un rapport Les attributs

```
<!ATTLIST REPORT security (high | medium | low ) "low">
<!ATTLIST CODE type CDATA #IMPLIED>
<!ATTLIST GRAPHIC file ENTITY #REQUIRED>
<!ENTITY xml "Extensible Markup Language">
<!ENTITY sgml "Standard Generalized Markup Language">
<!ENTITY pxa "Professional XML Authoring">
<!ENTITY % TEXT "(#PCDATA|CODE|KEYWORD|QUOTATION)*">
<!ENTITY % BLOCK "(PARA|LIST)+">
<!ENTITY % BODY "(%BLOCK;|EXAMPLE|NOTE)+">
<!NOTATION GIF SYSTEM "">
<!NOTATION JPG SYSTEM "">
<!NOTATION BMP SYSTEM "">  ]>
```

Conclusion XML

- **Multiples avantages**
  - **Un langage effectivement simple**.
  - XML rend possible l'arrivée d'une nouvelle génération **d'outils logiciels** pour des **plate-formes hétérogènes**.
    - de manipulation,
    - de transmission,
    - de visualisation de données distribuées.
- **Les inconvénients**
  - La simplicité de base conduit à **énormément de compléments de toutes natures** => apparition de dialectes et d'outils très nombreux qui ajoutent des détails.

Avenir de XML

- **XML devrait permettre** pour des applications (qui ne posent pas de problèmes cruciaux de performances) de supporter dans un cadre unifié:
- **La présentation des réseaux**
  - Format des données échangées par messages ou par invocations de méthodes.
- **La définition des documents.**
  - Outils de bureautique, de documentation ..
- **La définition des données.**
  - SGBD, logiciels de gestion, Échanges de Données Informatisé (EDI), ...

Le typage XML avec ‘XML schéma’

Introduction
Les structures
Les types de données
Conclusion

Le typage avec les DTD

- **XML est un outil d'avenir** pour l'échange, le stockage et l'affichage des données sur Internet.
- **Un problème majeur de XML 1.0** (dans sa définition de base) concerne **son approche du typage => DTD** 'Document Type Definition'.
- **L'objectif n'est pas atteint** pour des applications informatiques: la définition des types par les DTD est trop orientée documents textuels.
- **Conséquence :** nombreuses propositions pour améliorer le typage en XML.

Objectifs généraux des schémas

- **Décision complexe au sein du consortium WEB** (W3C) pour arbitrer entre les différentes propositions.
- **Unification autour d'un projet de recommandation**, adopté définitivement en mai 2001.
- **XML schéma est l'analogue:**
  - **D'un langage de définition de données DDL** ('Data Definition Language') des bases de données et un document est l'analogue d'un article d 'une base de données.
  - **D'un langage de syntaxe abstraite des réseaux comme ASN1** (' Abstract Syntax Notation') et un document est l'analogue du contenu en syntaxe de transfert (BER ' Basic Encoding Rules').
  - **D'un langage évolué dans sa partie définition de données** comme C++ ou Java. Un schéma définit l'analogue d 'une classe et un document est une instance de cette classe.
- **Un document XML doit pouvoir être validé** relativement à son schéma.

Objectifs précis

- Structures–

- **Définir la structure et les contenus** des documents.
- **Définir des relations d'héritage.**

- Typage des données –

- **Fournir un ensemble de types primitifs.**
- **Définir un système de typage suffisamment riche.**
- **Distinguer les aspects reliés à la représentation lexicale** des données de ceux gouvernant les données.
- **Permettre de créer des types de données usagers dérivés** de types existants en contraignant certaines propriétés (domaine, précision, longueur, format).

Les structures

Principes généraux des schémas

- **Un schéma XML est un document XML.**
- <?xml version="1.0" encoding="ISO-8859-1"?>
- <xsd:schema
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <!-- Déclaration de deux types d ’éléments -->
- <xsd:element name="nom" type="xsd:string" />
- <xsd:element name= "prenom" type="xsd:string" />
- </xsd:schema>

Exemple d'adresse postale en XML: le document

```
<?xml version="1.0"?>
<Addresse_postale_France pays="France">
   <nom>Mr Jean Dupont</nom>
   <rue>rue Camille Desmoulins</rue>
   <ville>Paris</ville>
   <departement>Seine</departement>
   <code_postal>75600</code_postal>
</Addresse_postale_france >
```

DTD d'une adresse postale

```
<!DOCTYPE une_DTD_adresse
[   <!ELEMENT Addresse_postale_france
          (nom, rue, ville, département, code_postal)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT rue (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
<!ELEMENT département (#PCDATA)>
<!ELEMENT code_postal (#PCDATA)>
<!ATTLIST Addresse_postale_france
          pays  NMTOKEN #FIXED 'France' > ]>
```

Typage d'une adresse postale au moyen d'un schéma XML

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:complexType name="Addresse_postale_france" >
        <xsd:sequence>
            <xsd:element name="nom" type="xsd:string" />
            <xsd:element name="rue" type="xsd:string" />
            <xsd:element name="ville" type="xsd:string" />
            <xsd:element name="departement" type="xsd:string" />
            <xsd:element name="code_postal" type="xsd:decimal" />
        </xsd:sequence>
         <xsd:attribute name=  "pays" type="xsd:NMTOKEN"
                use="fixed" value="FR"/>
</xsd:complexType>       </xsd:schema>
```

Le langage XML schémas : Les composants primaires

- **Un schéma XML est construit par assemblage de différents composants** (13 sortes de composants rassemblés en différentes catégories).
- **Composants de définition de types**
  - Définition de types **simples (Simple type).**
  - Définition de types **complexes (Complex type).**
- **Composants de déclaration**
  - **Déclaration d'éléments.**
  - **Déclaration d'attributs.**

Déclaration des éléments

- **Un élément XML est déclaré par la balise ‘element’** de XML schéma qui a de nombreux attributs.
- Les deux principaux attributs sont:

  - **name** : Le nom de l’élément (de la balise associée).
  - **type** : Le type qui peut être simple ou complexe.

Exemple de base

```
<xsd:element name="code_postal" type="xsd:decimal"/>
```

Déclaration des attributs

- **Un attribut est une valeur nommée et typée associée à un élément.**
- **Le type d'un attribut défini en XML schéma est obligatoirement simple.**

```
<xsd:complexType name="TypeRapport">
   <xsd:attribute name= "Date_creation"
   type="xsd:date"/>
     ……
</xsd:complexType>
<xsd:element name="Rapport" type="TypeRapport"/>
```

Autres attributs

- **L'élément attribute de XML Schema peut avoir deux attributs optionnels** : **use** et **value**.
- **On peut ainsi définir des contraintes** de présence et de valeur.
- Selon ces deux attributs, la valeur peut:

  être obligatoire ou non

  être définie ou non par défaut.
- Exemple: <xsd:attribute name= "Date_peremption" type="xsd:date" use="default" value= "2005-12-31"/>

Valeurs possibles pour use

- **Use = required** : L'attribut doit apparaître et prendre la valeur fixée si elle est définie.
- **Use= prohibited** : L'attribut ne doit pas apparaître.
- **Use = optional** : L'attribut peut apparaître et prendre une valeur quelconque.
- **Use= default** : Si l'attribut à une valeur définie il la prend sinon il prend la valeur par défaut.
- **Use= fixed** : La valeur de l'attribut est obligatoirement la valeur définie.
- **Exemple :** <xsd:attribute name= "Date_creation" type="xsd:date" use="required"/>

Types simples

- **‘SimpleType’ permet de définir des éléments ou des attributs non structurés** : dérivés d’une chaîne, d’un entier etc
  - **Types simples prédéfinis** au sens de la norme XML Schémas ‘ datatypes’: string, integer, boolean ...

```
<xsd:element name="code_postal " type="xsd:integer"/>
```

  - **Types simples définis par dérivation d'un autre type simple**, au moyen de l’élément <xsd:simpleType ...>
- Exemple de type simple : dérivation par restriction.

```
<xsd:simpleType name=  "DeuxDecimales">
        <xsd:restriction base="xsd:decimal">
                <xsd:fractionDigits value="2" />
        </xsd:restriction>
</xsd:simpleType>
```

Types complexes

- **Déclarés au moyen de l'élément** <xsd:complexType name="...
- **Ils peuvent contenir d'autres éléments**, des attributs.
- **Exemple**

```
<xsd:complexType name= "TypePrix">
        <xsd:simpleContent>
            <xsd:extension base="DeuxDecimales">
                <xsd:attribute name="Unite" type= "FrancEuro" />
            </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
```

- **Trois façons de composer des éléments** dans un type complexe: sequence, choice, all.

Types complexes: Sequence

- Un type **sequence** est défini par une suite de sous-éléments qui doivent être présents dans l'ordre donné.
- Le nombre d'**occurences** de chaque sous-élément est défini par les attributs minOccurs et maxOccurs.

```
<xsd:complexType name= "Commande">
    <xsd:sequence>
        <xsd:element name= "Ad_livraison" type="Addresse"/>
        <xsd:element name= "Ad_facturation" type="Addresse"/>
       <xsd:element name=  "texte" type="xsd:string" minOccurs="1" />
        <xsd:element name="items" type="Items" maxOccurs= "30" />
     </xsd:sequence>
</xsd:complexType>
```

Types complexes: Choice

- Un seul des éléments listés doit être présent.
- Le nombre d'occurences possible est déterminé par les attributs minOccurs et maxOccurs de l'élément.

```
<xsd:complexType name=  "type_temps">
   <xsd:choice >
    <xsd:element name= "Noire"  type="Note" minOccurs="1"
    maxOccurs="1" />
    <xsd:element name= "Croche"  type="Note" minOccurs="2"
    maxOccurs="2" />
   </xsd:choice>
</xsd:complexType>
```

Types complexes: All

- C’est une composition de type ensembliste. Dans un document conforme, les éléments listés doivent être tous présents au plus une fois. Il peuvent apparaître dans n'importe quel ordre.

```
<xsd:complexType name= "Commande">
   <xsd:all>
      <xsd:element name= "Ad_livraison" type="Addresse"/>
      <xsd:element name= "Ad_facturation" type="Addresse"/>
      <xsd:element name= "texte" type="xsd:string" minOccurs="0" />
      <xsd:element name="items"  type="Items" maxOccurs= "30" />
   </xsd:all>
</xsd:complexType>
```

Les types de données XML schéma

Objectifs de la définition des types

- **Fournir des types primitifs** analogues à ceux qui existent en SQL ou en Java.
- **Définir un système de typage suffisamment riche** pour importer/exporter des données d'une base de données.
- **Distinguer les aspects reliés à la représentation lexicale des données** de ceux gouvernant les ensembles de données sous-jacents.
- **Permettre de créer des types de données usagers** dérivés de types existants en contraignant certaines propriétés (domaine, précision, longueur, format).

Système de typage des schémas

Trois composantes:

a) L'ensemble des valeurs du type ('value space')

Ex: type float.

b) L'ensemble des représentations lexicales possibles des valeurs ('lexical space ').

Ex: "10" ou "1.0E1"

c) L'ensemble des facettes (l'ensemble des propriétés) qui définit l'ensemble des valeurs (notion de facette fondamentale et de facette de contrainte).

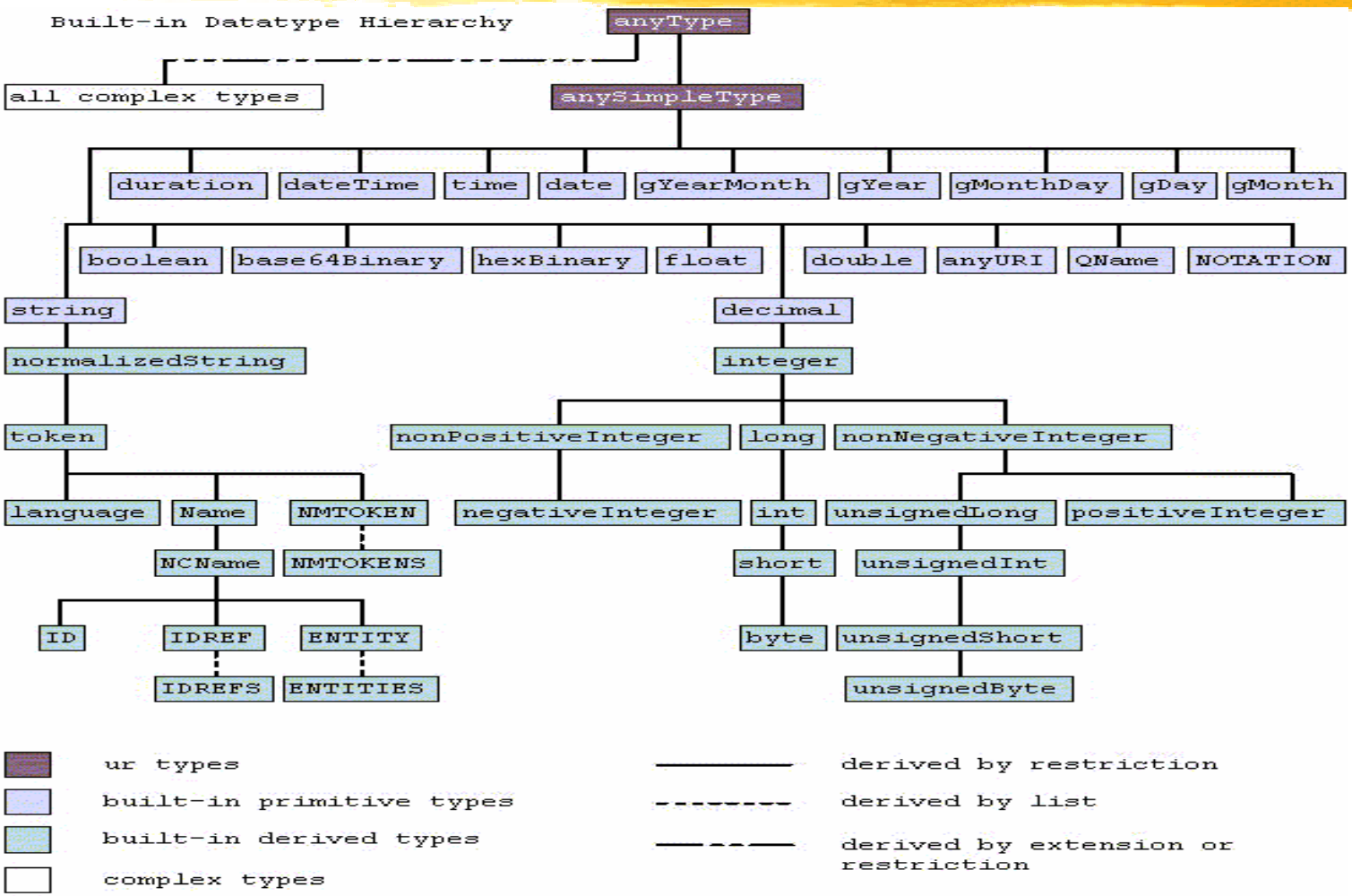
Ex: Le type float est défini par la norme IEEE 754-1985 (c'est un flottant simple précision sur 32-bit).

On peut dériver des types par contraintes.

Définitions relatives aux types

- **Types primitifs** (‘Primitive’) Non défini en référence à d’autres types..
- **Types dérivés** (‘Derived’) Définis par dérivation à partir d’autres types.
- **Types prédéfinis** (‘Built-in’) Définis dans le cadre de la spécification XML Schéma datatypes (primitif ou dérivé).
- **Types usagers** (‘User-derived’) Types construits par les utilisateurs.
- **Types atomiques** (‘Atomic’) Types indivisibles du point de vue de la spécification XML schéma.
- **Types listes** (‘List’) Types dont les valeurs sont des listes de valeurs de types atomiques.
- **Types unions** (‘Union’) Types dont les ensembles de valeur sont la réunion d’ensemble de valeurs d’autres types.

Hiérarchie des types prédéfinis



Quelques types prédéfinis

Type	Forme lexicale
String	**Bonjour**
boolean	{**true**, **false, 1, 0**}
float	**2345E3**
double	**23.456789E3**
decimal	**808.1**
dateTime	**1999-05-31T13:20:00-05:00.**
binary	**0100**
uriReference	**http://www.cnam.fr**
TOKEN	**un token selon definition des DTD**
....	

Dérivation de types simples
1- Dérivation par restriction

- **La dérivation par restriction restreint l'ensemble des valeurs** d'un type pré existant.
- La restriction est définie par des contraintes de facettes du type de base: valeur min, valeur max ...
- **Exemple :**

```
<xsd:simpleType name= "ChiffresOctaux">
        <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="0" />
                <xsd:maxInclusive value= 7" />
        </xsd:restriction>
</xsd:simpleType>
```

Les contraintes de facettes

- lenght : la longueur d'une donnée.
- minLenght: la longueur minimum.
- maxLenght: la longueur maximum.
- pattern: défini par une expression régulière.
- enumeration: un ensemble discret de valeurs.
- whitespace: contraintes de normalisation des chaînes relativement aux espaces (preserve, replace, collapse).
- maxInclusive: une valeur max comprise.
- maxExclusive: une valeur max exclue.
- minInclusive: une valeur min comprise.
- maxInclusive: une valeur min exclue.
- totalDigits: le nombre total de chiffres.
- fractionDigits: le nombre de chiffres dans la partie fractionnaire.

Exemple d’une énumération

```
<xsd:simpleType name= "Mois">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value= "Janvier"/>
                <xsd:enumeration value="Février"/>
                <xsd:enumeration value="Mars"/>
                <!-- ... -->
        </xsd:restriction>
</xsd:simpleType>
```

2 - Dérivation par extension

- Dériver un nouveau type par **extension** consiste à ajouter à un type existant des sous-éléments ou des attributs.
- On obtient inévitablement **un type complexe.**
- **Exemple**

```
<xsd:complexType name= "mesure">
   <xsd:simpleContent><xsd:extension base="xsd:Decimal">
        <xsd:attribute name="unite" type="xsd:NMTOKEN"/>
   </xsd:extension></xsd:simpleContent>
</xsd:complexType>
 <xsd:element name= "temperature" type= "mesure"/>
 <temperature unit="Kelvin">230</temperature>
```

3 - Dérivation par union

- Pour créer un nouveau type on effectue l'union ensembliste de toutes les valeurs possibles de différents types existants.
- Exemple:

```
<xsd:simpleType name="TransportFormatCaracteres">
<xsd:union memberTypes="base64Binary hexBinary"/>
</xsd:simpleType>
```

4 - Dérivation par liste

- Une **liste** permet de définir un nouveau type de sorte qu'une valeur du nouveau type est une liste de valeurs du type pré existant (valeurs séparées par espace).
- **Exemple**

```
<simpleType name= 'DebitsPossibles'>
        <list itemType='nonNegativeInteger'/>
</simpleType>
<debitsmodemV90 xsd:type='DebitsPossibles'>
33600 56000 </debitsmodemV90>
```

Conclusion typage avec XML schéma

Un standard très utile en réseaux/systèmes répartis

- **Indispensable comme outil d'interopérabilité en univers réparti.**
  - Entre des applications WEB.
  - Dans des approches objets répartis comme SOAP ('Simple Object Protocol')
  - WSDL ('Web Service Definition Language').
  - Entre des bases de données hétérogènes.
- **Quelques reproches**
  - Les performances.
  - Les imperfections à découvrir dans les choix de conception du système de typage.

Domaines d'utilisation

- Publication d'informations sur le WEB.
- Commerce électronique.
- Gestion de documents traditionnels.
- Assistance à la formulation et à l'optimisation des requêtes en bases de données.
- Transfert de données entre applications en réseaux.
- Contrôle de supervision et acquisition de données.
- Échange d'informations de niveau méta.

Bibliographie Normes

- ‘XML Schema Part 0: Primer’ W3C
  http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/
- ‘XML Schema Part 1: Structures’ W3C
  http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
- ‘XML Schema Part 2: Datatypes’ W3C
  http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
- XML Schema Requirements http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215 (Février 1999)
- Introduction aux schémas http://www.w3schools.com/schema
- Les schémas XML Gregory Chazalon, Joséphine Lemoine http://site.voila.fr/xmlschema
- W3C XML Schema, Eric Van Der Vlist, http://www.xml.com/pub/a/2000/11/29/schemas/