DTD : DOCUMENT TYPE DEFINITION

- C’est une grammaire dont les phrases sont des documents XML (instances) qui :
  - ✓ permet de définir le «vocabulaire» (définit le jeux de balises utilisables ainsi que leurs attributs)
  - ✓ et la structure qui sera utilisée dans le document XML (l’imbrication des balises)
  - ✓ possibilité de décrire si les balises sont obligatoires ou optionnelles
- La DTD peut être référencé par un URI ou incluse directement dans le document XML
- Il existe d’autres types de grammaires comme XML Schema (XSD), Relax NG, etc.
  - ✓ leur puissance sémantique est plus élevée (on peut exprimer plus de contraintes)
  - ✓ Relax NG offre le meilleur rapport puissance/facilité
  - ✓ DTD est le plus répandu
  - ✓ XML Schema le plus souvent utilisé pour formaliser des langages "webservices", par ex. SOAP

DÉCLARATION ÉLÉMENT SIMPLE

- <!ELEMENT balise (définition) >
  - le paramètre définition représente
    - soit un type de données prédéfini,
    - soit un type de données composé constitué lui-même d'éléments
- Types prédéfinis
  - ANY: l'élément peut contenir tout type de donnée
  - EMPTY: l'élément ne contient pas de données spécifiques (élément vide)
  - #PCDATA: l'élément doit contenir une chaîne de caractères dans l'encodage courant (non interprétée par XML)
- Exemple
  - <!ELEMENT Nom (#PCDATA)>
  - <Nom>Victor Hugo</Nom>

ÉLÉMENTS COMPOSÉS

- Elément composé d'une séquence ou d'un choix d'éléments
- Syntaxe spécifique avec opérateurs de composition d'éléments:

  <!ELEMENT balise (composition) >

A et B	Explication	Exemples
A?	A (un seul) est une option, (donc: A ou rien)	<!ELEMENT personne (nom, email?)
A+	Il faut un ou plusieurs A	<!ELEMENT personne (nom, email+)
A*	A est une option, il faut 0, 1 ou plusieurs A	<!ELEMENT personne (nom, email*)
A | B	Il faut A ou B, mais pas les deux	<!ELEMENT personne (email|fax)
A , B	Il faut A suivi de B (dans l'ordre)	<!ELEMENT personne (nom, email ?)
(A, B)+	Les parenthèses regroupent. Ici, un ou plusieurs (A suivi de B)	<!ELEMENT liste (nom, email)+

- Semi-structuré car on peut avoir un *mixed content*
  - ✓(#PCDATA | e1 | … | en)

EXEMPLE

- DTD
  - <!ELEMENT personne (nom, prenom+, tel?, email, adresse >
  - <!ELEMENT nom (#PCDATA) >
  - <!ELEMENT prenom (#PCDATA) >
  - <!ELEMENT tel (#PCDATA) >
  - <!ELEMENT email (#PCDATA) >
  - <!ELEMENT Adresse (ANY) >
- Document associé
  - <personne>
  - <nom>Hugo</nom>
  - <prenom>Victor</prenom>
  - <prenom>Charles</prenom>
  - <tel>0383000000</tel>
  - <adresse><rue/><ville>Paris</ville></adresse>
  - </personne>

ATTRIBUTS

- <! ATTLIST *balise attribut type mode* >
  - ✓ *balise* spécifie l'élément auquel est attaché l'attribut
  - ✓ *attribut* est le nom de l'attribut déclaré
  - ✓ *type* définit le type de donnée de l'attribut choisi parmi:
    - CDATA: chaînes de caractères entre guillemets ("aa") non analysées
    - Enumération: liste de valeurs séparées par |

      <! ATTLIST balise Attribut (Valeur1 | Valeur2 | …) >
    - NMTOKEN: un seul mot
    - ID et IDREF: clé et référence à clé
  - ✓ *mode* précise le caractère obligatoire ou non de l'attribut
    - #REQUIRED: obligatoire (l'attribut doit être valué)
    - #IMPLIED: optionnel (l'attribut peut être valué)
    - #FIXED valeur: attribut avec valeur fixe (valeur fixée dans la DTD)

EXEMPLE

```
<!ELEMENT personne (nom, prenom+, tel?, adresse >

<!ELEMENT nom (#PCDATA) >

<!ELEMENT prenom (#PCDATA) >

<!ELEMENT tel (#PCDATA) >

<!ELEMENT email (#PCDATA) >

<!ELEMENT Adresse (ANY) >

<! ATTLIST personne
    num ID
    age CDATA
    genre (Masculin | Feminin ) >

<!ELEMENT auteur (#PCDATA) >

<!ATTLIST auteur
    genre (Masculin | Feminin ) #REQUIRED
    ville CDATA #IMPLIED>

<!ELEMENT editeur (#PCDATA) >

<!ATTLIST editeur

ville CDATA #FIXED "Paris">
```

ATTRIBUTS VS. ÉLÉMENTS

- Il s'agit ici une FAQ classique sans réponse précise…
- Il faut plutôt utiliser un élément
  - ✓ lorsque l'ordre est important (l'ordre des attributs est au hasard)
  - ✓ lorsqu'on veut réutiliser un élément plusieurs fois (avec le même parent)
  - ✓ lorsqu'on veut (dans le futur) avoir des descendants / une structure interne
  - ✓ pour représenter un type de données (objet) plutôt que son usage, autrement dit: une "chose" est un élément et ses propriétés sont des "attributs"
- Il faut plutôt utiliser un attribut
  - ✓ lorsqu'on désire faire référence à un autre élément
    - <compagnon genre="giraffe"> fait référence à <animal cat="giraffe">
  - ✓ pour indiquer l'usage/type/etc. d'un élément
    - <adresse usage="prof"> ... </adresse>
  - ✓ lorsque vous voulez imposer des valeurs par défaut dans la DTD
  - ✓ lorsque vous voulez un type de données (pas grand chose dans la DTD)

EXEMPLE DE DTD

```
<!ELEMENT carnetAdresses (personne)+>
<!ELEMENT personne (nom,email*)>
<!ELEMENT nom (famille,prenom)>
<!ELEMENT famille (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

```
<carnetAdresses>
  <personne>
    <nom><famille>Perrin</famille><prenom>Olivier</prenom></nom>
    <email>Olivier.Perrin@loria.fr</email>
  </personne>
  <personne>
    <nom><famille>Lagrange</famille><prenom>Anne</prenom></nom>
    <email>Anne.Lagrange@mozilla.org</email>
  </personne>
</carnetAdresses>
```

ID ET IDREF

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE Document [
    <!ELEMENT Document (personne*)>
    <!ELEMENT personne (#PCDATA)>
    <!ATTLIST personne PNum ID #REQUIRED>
    <!ATTLIST personne Mere IDREF #IMPLIED>
    <!ATTLIST personne Pere  IDREF #IMPLIED>
]>
< Document >
    < personne PNum = "P1">Marie</personne>
    < personne PNum = "P2">Jean</personne>
    < personne PNum = "P3" Mere ="P1" Pere ="P2">Pierre</personne>
    < personne PNum = "P4" Mere ="P1" Pere ="P2">Julie</personne>
</Document >
```

ASSOCIATION DTD/DOCUMENT XML

- Il existe 4 façons d'utiliser une DTD
  - ✓ 1. On déclare pas de DTD (dans ce cas le fichier est juste "bien formé")
  - ✓ 2. On déclare la DTD et on y ajouter les défintions dans le fichier (DTD interne)
  - ✓ 3. On déclare la DTD en tant que DTD "privé": la DTD se trouve quelque part dans votre système ou sur Internet (répandu pour les DTDs "faits maison")
  - ✓ 4. On délare une DTD "public". On utilise un nom officiel pour la DTD. Cela présuppose que votre éditeur et votre client conaissent cette DTD (répandu pour les DTDs connues comme XHTML, SVG, MathML, etc.)
- Lieu de la déclaration
  - ✓ la DTD est déclaré entre la déclaration de XML et le document lui-même
  - ✓ la déclaration de XML et celle de la DTD font parti du prologue (qui peut contenir d'autres éléments comme les *processing instructions*)
  - ✓ attention: l'encodage de la DTD doit correspondre à celui des fichiers XML !

ASSOCIATION DTD/DOCUMENT XML (2)

- Syntaxe de la déclaration
  - ✓ chaque déclaration de la DTD commence par <!DOCTYPE ... et fini par >
  - ✓ la racine de l'arbre XML (ici: <hello>) doit être indiquée après <!DOCTYPE
  - ✓ syntaxe pour définir une DTD interne (seulement !)
    - la DTD sera insérée entre [...]

```
<!DOCTYPE hello [
    <!ELEMENT hello (#PCDATA)>
]>
```

  - ✓ syntaxe pour définir une DTD privée externe:
    - la DTD est dans l'URL indiqué après le mot clef "SYSTEM"

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

    - déclaration de la DTD dans le fichier XML et PAS dans le fichier *.dtd.
- Définition de la racine de l'arbre
  - ✓ le mot "hello" après le mot clef DOCTYPE indique que "hello" est l'élément racine de l'arbre XML

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

ASSOCIATION DTD/DOCUMENT XML (3)

- Hello XML sans DTD

```
<?xml version="1.0"standalone="yes"?>
<hello> Hello XML et hello cher lecteur ! </hello>
```

- Hello XML avec DTD interne

```
<?xml version="1.0"standalone="yes"?><!DOCTYPE hello [
   <!ELEMENT hello (#PCDATA)>              ]>
<hello> Hello XML et hello cher lecteur ! </hello>
```

- Hello XML avec DTD externe

```
<?xml version="1.0" encoding="ISO-8859-1" ?><!DOCTYPE hello SYSTEM "hello.dtd">
<hello> Hello XML et hello cher lecteur ! </hello>
```

- Un fichier RSS (DTD externe public)

```
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE rss PUBLIC "-//Netscape  Communications//DTD RSS 0.91//EN"
"http://my.netscape.com/publish/formats/

rss-0.91.dtd">
<rss version="0.91"> <channel>…</channel></rss>
```

CONCEPTS ADDITIONNELS (NON DÉVELOPPÉS ICI)

- Validation (par ex. http://validator.w3.org/)
  - ✓ analyse le document en entrée (en particulier s'il est *well formed*)
  - ✓ vérifie l'élément racine
  - ✓ pour chaque élément, vérifie le contenu et les attributs
  - ✓ vérifie l'unicité et les contraintes référentielles (attributs ID/IDREF(S))
- Entité
  - ✓ une "entity" est un bout d'information stocké quelque part
  - ✓ les entités d'un document sont remplacées par le contenu référencé (macro)
  - ✓ distinction entre entités générales et entités paramétrées
- Espaces de noms
  - ✓ on peut dans un même document de mélanger plusieurs grammaires (si l'application le permet), par exemple: XHtml + Svg + MathML + XLink
  - ✓ pour éviter qu'il y ait confusion entre différentes balises, on définit un "namespace" pour chaque grammaire et on l'utilise pour les éléments (soit pour tout l'arbre, soit pour une partie seulement)

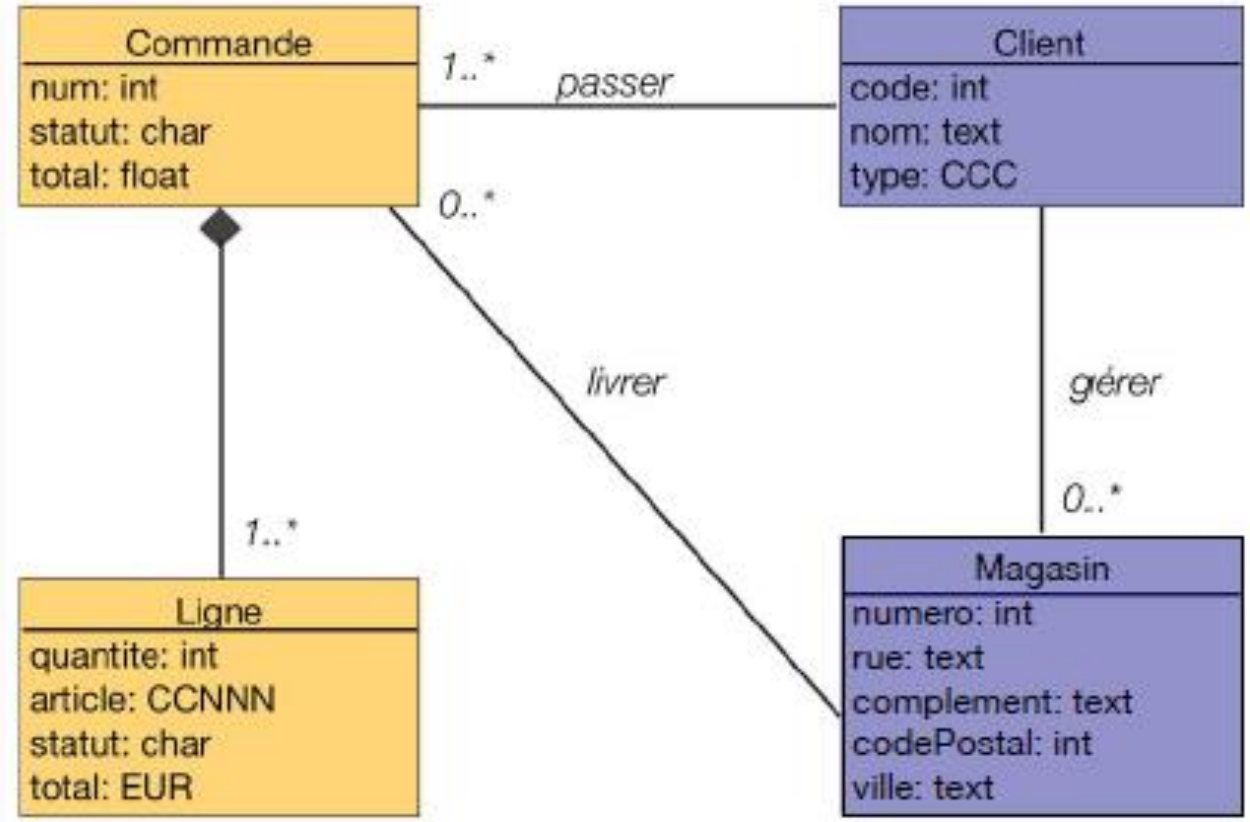
EXERCICE

- Un carnet d'adresses plus complet
  - ✓ la personne possède un identifiant unique (obligatoire), un nom, un prénom
  - ✓ on veut connaître le sexe de la personne (attribut optionnel)
  - ✓ on veut connaître son email (optionnel)
  - ✓ on veut connaître le lien entre le chef et ses employés

EXERCICE

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT carnetAdresses (personne)+>
<!ELEMENT personne (nom,email*)>
    <!ATTLIST personne id ID #REQUIRED>
    <!ATTLIST personne sexe (masculin|feminin) #IMPLIED>
<!ELEMENT nom (#PCDATA|famille|prenom)*>
<!ELEMENT famille (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT lien EMPTY>
     <!ATTLIST lien chef IDREF #IMPLIED employe IDREF #IMPLIED>
```

PASSAGE UML VERS DTD



PASSAGE UML VERS DTD (2)

```
<!-- Types de base-->
<!ENTITY % int "(#PCDATA)">
<!ENTITY % float "(#PCDATA)">
<!ENTITY % char "(#PCDATA)">
<!ENTITY % string "(#PCDATA)">
<!-- Classe Commande -->
<!ELEMENT Commande (cstatut, ctotal, Ligne+)>
<!ATTLIST Commande NUM ID #REQUIRED>
<!ELEMENT cstatut %char;>
<!ELEMENT ctotal %float;>
<!-- Classe Ligne -->
<!ELEMENT Ligne (article, quantite, statut?, total?)>
<!ELEMENT article %string;>
<!ELEMENT quantite %int;>
<!ELEMENT lstatut %char;>
<!ELEMENT ltotal %float;>
```

LE DOCUMENT XML ASSOCIÉ

```
<?xml version=”1.0” ?>
<message>
   <Commande num=”1”>
      <cstatut>A</cstatut>
      <ctotal>1000</ctotal>
      <Ligne>
         <article>212</article>
         <quantite>100</quantite>
      </Ligne>
   </Commande>
   <Commande num=”2”>
      <cstatut>B</cstatut>
      <ctotal>1000</ctotal>
      <Ligne>
         <article>212</article>
         <quantite>300</quantite>
      </Ligne>
      <Ligne>
         <article>312</article>
         <quantite>400</quantite>
      </Ligne>
   </Commande>
</message>
```

FAIBLESSE DES DTDS

- Pas de types de données
  - ✓difficile à interpréter par le récepteur (indépendant du contexte)
  - ✓pas d'expressions rationnelles pour les valeurs
  - ✓spécification des valeurs d'attributs simpliste
  - ✓support pour la modularité et la réutilisation limité
  - ✓difficile à traduire en schéma objets
- Pas en XML
  - ✓langage spécifique
- Propositions de compléments
  - ✓XML Schema Definition du W3C
  - ✓Relax NG

XML SCHEMA

- Un schéma d'un document définit:
  - ✓ les éléments possibles dans le document
  - ✓ les attributs associés àces éléments
  - ✓ la structure du document
  - ✓ les types de données
- Le schéma est spécifié en XML
  - ✓ pas de nouveau langage
  - ✓ balisage de déclaration
  - ✓ espace de nom spécifique xsd: ou xmlns:
- Présente de nombreux avantages
  - ✓ structures de données avec types de données (expressivité accrue)
  - ✓ extensibilité par héritage et ouverture
  - ✓ analysable à partir d'un parseur XML standard (c'est du XML)

OBJECTIFS

- Reprendre les acquis des DTD
  - ✓plus riche et complet que les DTD
- Permettre de typer les données
  - ✓éléments simples et complexes
  - ✓attributs simples
- Permettre de définir des contraintes
  - ✓existence, obligatoire, optionnel
  - ✓domaines, cardinalités, références
  - ✓patterns, ...
- S'intégrer à la galaxie XML
  - ✓espace de noms
  - ✓infoset (structure d'arbre logique)

MODÈLE DES SCHÉMAS

- Déclaration des éléments et attributs
  - ✓ nom
  - ✓ typage similaire à l'objet
- Spécification de types simples
  - ✓ grande variété de types
- Génération de types complexes
  - ✓ *sequence*
  - ✓ *choice*
  - ✓ *all*

TYPES SIMPLES

- string, normalizedString, token
- byte, unsignedByte
- base64Binary, hexBinary
- integer, positiveInteger, negativeInteger, nonNegativeInteger, nonPositiveInteger, int,
- unsignedInt
- long, unsignedLong
- short, unsignedShort

TYPES SIMPLES(2)

- decimal, float, double
- boolean
- time, dateTime, duration, date, gMonth, gYear, gYearMonth, gDay, gMonthDay
- Name, QName, NCName, anyURI
- language
- ID, IDREF, IDREFS
- ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS

COMMANDES DE BASE

- element: association d'un type à une balise
  - Attributs name, type, ref, minOccurs, maxOccurs,…
- attribute: association d'un type à un attribut
  - attributs name, type
- type simple: les multiples types de base
  - entier, réel, string, time, date, ID, IDREF,…
  - extensibles par des contraintes
- type complexe: une composition de types
  - définit une agrégation d'éléments typés

EXEMPLES DE TYPES SIMPLES

- Exemple

```
<simpleType name="entier0_a_100">
 <restriction base="integer">
   <minInclusive value="0"/>
   <maxInclusive value="100"/>
 </restriction>
</simpleType>
```

- Exemple 2

```
<simpleType name="listeEntier">
 <list itemType="integer"/>
</simpleType>
```

TYPES COMPLEXES

- Définition d'objets complexes
  - ✓ <*sequence*>: collection ordonnée d'éléments typés (concaténation)
  - ✓ <*all*>: collection non ordonnée d'éléments typés
  - ✓ <*choice*>: choix entre éléments typés (union)
  - ✓ <*any* …>: joker

TYPES COMPLEXES

- Exemple

```
<xsd:complexType name="AddresseFR">
 <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="codepostal" type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="pays" type="xsd:NMTOKEN" fixed="FR"/>
</xsd:complexType>
```

HÉRITAGE

- Définition de sous-types par héritage
  - ✓ par extension: ajout d'informations
  - ✓ par restriction: ajout de contraintes
- Possibilité de contraindre la dérivation
- Exemple :

```
<complexType name="AdressePays">
 <complexContent>
   <extension base="Adresse">
     <sequence>
       <element name="pays" type="string"/>
     </sequence>
   </extension>
 </complexContent>
</complexType>
```

PATTERNS

- Contraintes sur type simple prédéfini
- Utilisation d'expression rationnelles (regular expressions)
  - ✓similaires à celles de Perl
- Exemple

```
<xsd:simpleType name="pourcentage">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="([0-9]|[1-9][0-9]|100)%"/>
  </xsd:restriction>
</xsd:simpleType>
```

CONCLUSION

- Schémas flexibles et irréguliers
  - ✓optionnels, avec ou sans DTD
- Données auto-descriptives
  - ✓balises et attributs
- Modèle de type hypertexte
  - ✓support des références
- Éléments atomiques ou complexes
  - ✓composition par agrégation
- Types de données variés et extensibles
  - ✓textes, numériques, …, types utilisateur