Plan du cours eXtensible Markup Language

- Éléments de base du XML
- DTD : Document Type Description

S1

- XPath : XML Path Language
- XML Schema

S2

- XSLT : Transformations XML Style Language
- Programmation avec XML : SaX et DOM

S3

=> projet

Introduction au langage XML
1: XML & DTD

Introduction : W3C et XML

- Le World Wide Web Consortium (W3C)
  - URL: http://www.w3.org
  - 400 partenaires industriels, parmi lesquels les plus grand comme Oracle, IBM, Compaq, Xerox, Microsoft
  - Laboratoires de recherche: MIT pour les États Unis, INRIA pour l'Europe, université Keio (Japon) pour l'Asie
- XML : recommandation W3C pour
  - les documents Web (généralisation de HTML),
  - mais aussi pour l'échange, la transformation, l'intégration et
  - l'interrogation des données sur le Web.
- XML : pourquoi je l'utilise ?

Introduction : W3C et XML(2)

- SGML : Standard Generalized Markup Language
  - Trop compliqué, inadapté aus besoins du WWW
- HTML : HyperText Markup Language
  - Simple mais ayant beaucoup de problème
- XLM : eXtensible Markup Language (1997)
  - Adaptation de SGML aux besoins du World Wide Web
  - Objectifs :
    - Séparation des données et de leur mise en forme (XML, XSL)
    - Séparation des informations structurelles et des données (DTD)

Éléments de base : Principe

- Principe clé de *XML*: *séparer la structure d'un document de sa présentation*

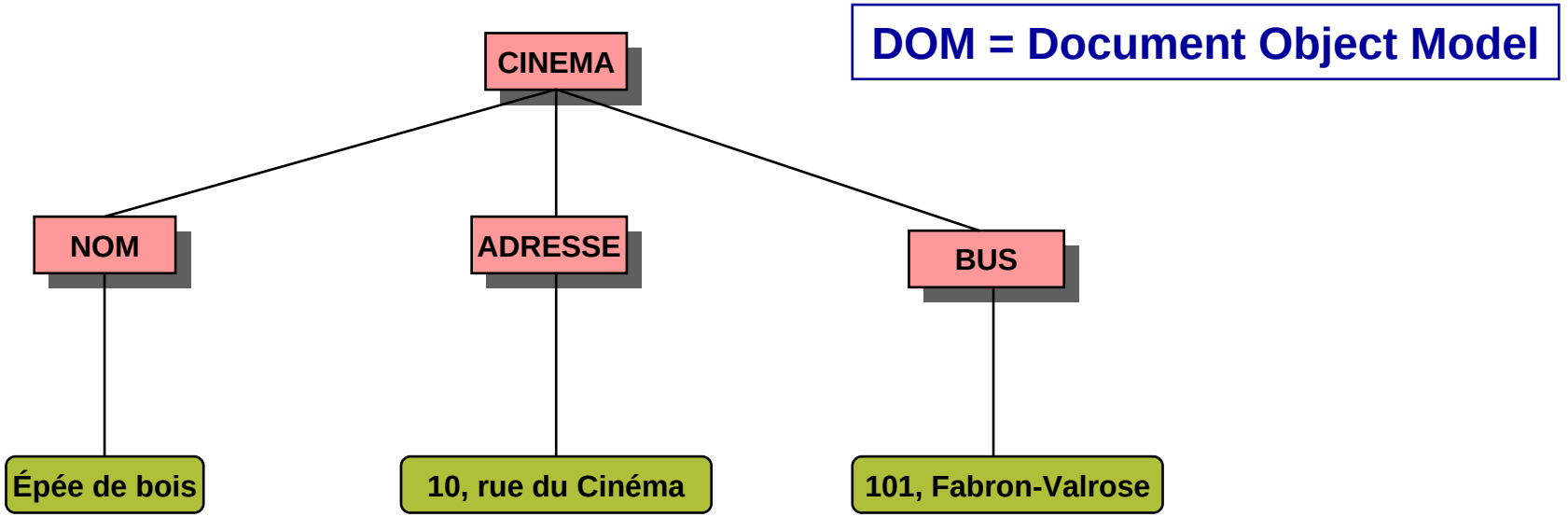
- Avantages:
  - indépendance entre les outils de présentation (browser) et les outils de gestion de l'information
  - différentes présentations sont possibles pour le même document (transformation)
  - interrogation (semi-)structurée de documents

Éléments de base : notion de balise

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CINEMA>
    <NOM>
        Epée de Bois
    </NOM>
    <ADRESSE>
        10, rue du Cinéma
    </ADRESSE>
    <BUS>
        101:Fabron-Valrose
    </BUS>
</CINEMA>
```

Éléments de base : DOM et Parseur

- Document a une forme arborescente :
  - DOM : Un arbre, constitué de noeuds typés (éléments, commentaires, valeurs, etc)



Éléments de base : DOM et Parseur

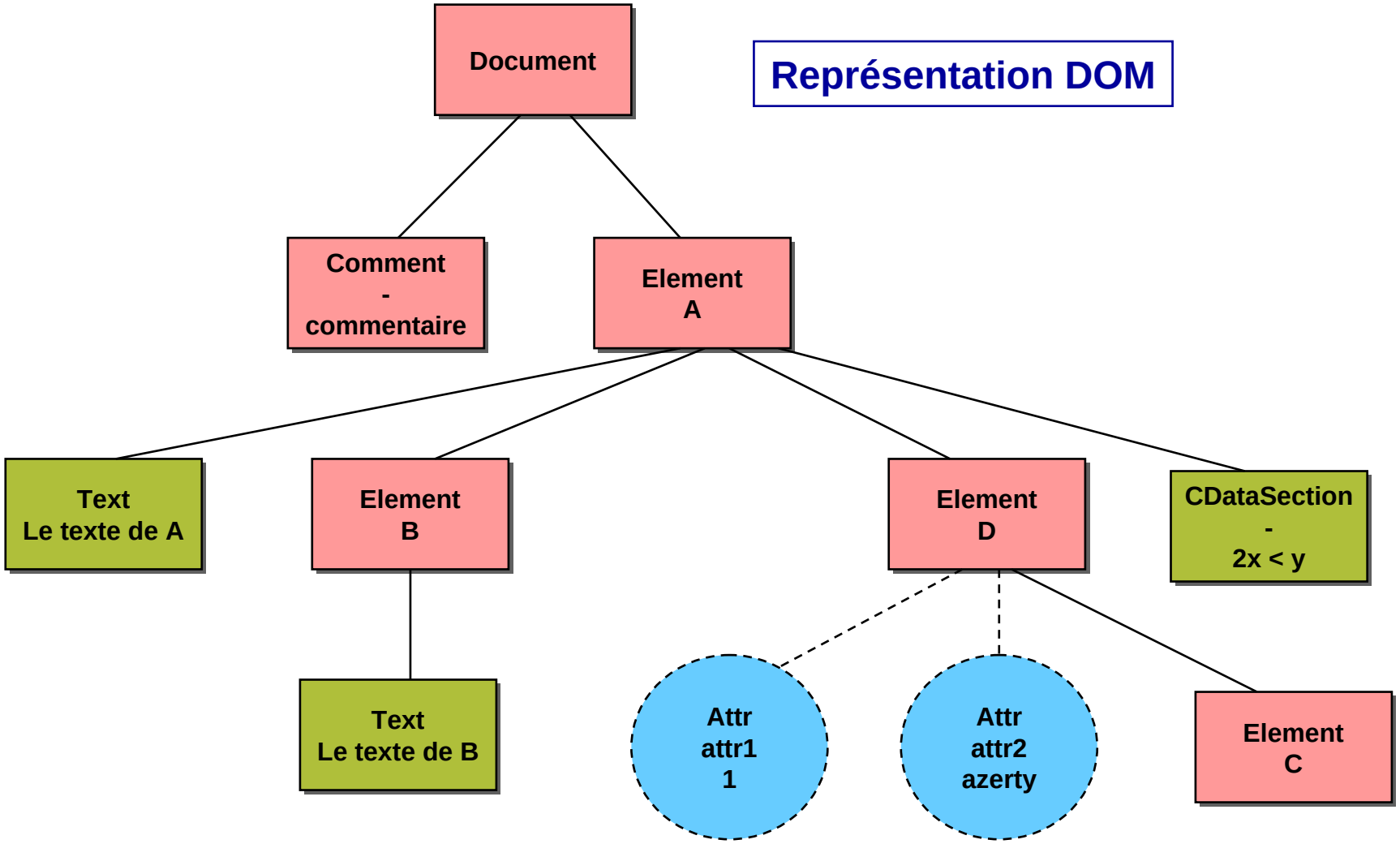
- Parseur : programme permettant le *passage de la forme sérialisée* à la *représentation arborescente de DOM*
- Le document sérialisé est analysé, et une représentation arborescente est créée :
  - le noeud racine est de type **Document**
  - les catégories syntaxiques (commentaires, balises, texte) se traduisent par différents types de noeuds (**Comment**, **Element**,**Text, …**)
  - les noeuds constituent un arbre qui reflète l'imbrication des éléments dans la forme sérialisée

Éléments de base : DOM et Parseur

Exemple : représentation sérialisée (textuelle)

```
<?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>
```

Éléments de base : DOM et Parseur



Éléments de base : structure d'un document

- Un document XML comprend trois parties :
  - le **prologue**, avec la déclaration XML, la DTD, des commentaires, des instructions de traitements (optionnels)
  - un **élément racine** avec son contenu
  - un **épilogue** avec des commentaires, ou des instructions de traitements (optionnels)
- Le contenu du document proprement dit est le contenu de l'élément racine.

Éléments de base : Prologue - Déclaration

- Tout document XML peut être précédé par une déclaration :

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

  - l'attribut « encoding » indique le jeu de caractères utilisé dans le document (norme iso10646)

  - l'attribut optionnel « standalone » indique si le document est composé de plusieurs entités.

Éléments de base : Types - DTD

DTD = Document Type Description

- On peut indiquer qu'un document est conforme à une *DTD*, et déclarer des *entités*.

<!DOCTYPE nom SYSTEM "sourceExt" [decLoc]>

  - *nom* est le type de l'élément racine
  - *sourceExt* est une source extérieure contenant la DTD
  - *decLoc* sont des déclarations locales (pour les entités principalement)

Éléments de base : Syntaxe du langage

- La syntaxe XML permet la représentation sérialisée d'un arbre DOM:
  - les éléments (et leurs attributs)
  - les entités
  - les commentaires
  - les instructions de traitement
  - les sections de texte
  - les sections littérales

  Ainsi que quelques règles sur la structure d'un document

Syntaxe : Balises ou éléments

- La balise (appelée également *élément*) est la notion principale pour définir le *contenu* d'un document XML
- Dans la forme sérialisée :
  - C'est une **balise ouvrante** avec un nom, puis
  - un **contenu**,
  - puis une **balise fermante**
- Dans la forme arborescente
  - C'est un noeud avec un nom
  - Le contenu est un arbre

Syntaxe : Balises ou éléments (2)

- Forme générale

  <nom_d_element> contenu </nom_d_element>

- Les noms sont libres. Ils obéissent à quelques règles :
  - 1er caractère ∈ { alpha, « - », « _ »}
  - Autres caractères ∈ { alpha, chiffre, « - », « _ », « : »}
  - Pas de blanc
  - Les majuscules sont distinguées des minuscules
  - 3 premiers caractères sont différents à «xml »
- La balise de fermeture est obligatoire

Syntaxe : Balises ou éléments (3)

- Le contenu d'un élement peut être
  - vide (<toc></toc> ou <toc/>)
  - du texte sauf « < » ou « > » ou « & »
  - un ou plusieurs éléments complets
  - une répétition de textes et d'éléments
  - Les blancs comptent :
    - <a> X est différent de <a>X
  - Les deux systèmes de codage des retours de lignes sont pris en charge

Syntaxe : arbre d'éléments

- Un document XML est un et un seul arbre d'éléments. C'est-à-dire pas de chevauchement d'éléments. La notation

  <list> ... <item> ... </list> ... </item>

  est invalide. Il faut la corriger comme suit

  <list> ... <item> ... </item> ... </list>

- Un document XML est compose d'un seul élément. La notation

  <?xml version="1.0" encoding="iso-8859-1" ?>
  <article> ... </article>
  <article> ... </article>

  est invalide. Il faut la corriger comme suit

  <?xml version="1.0" encoding="iso-8859-1" ?>
  <stock>
  <article> ... </article>
  <article> ... </article>
  </stock>

Syntaxe : Exemple des éléments

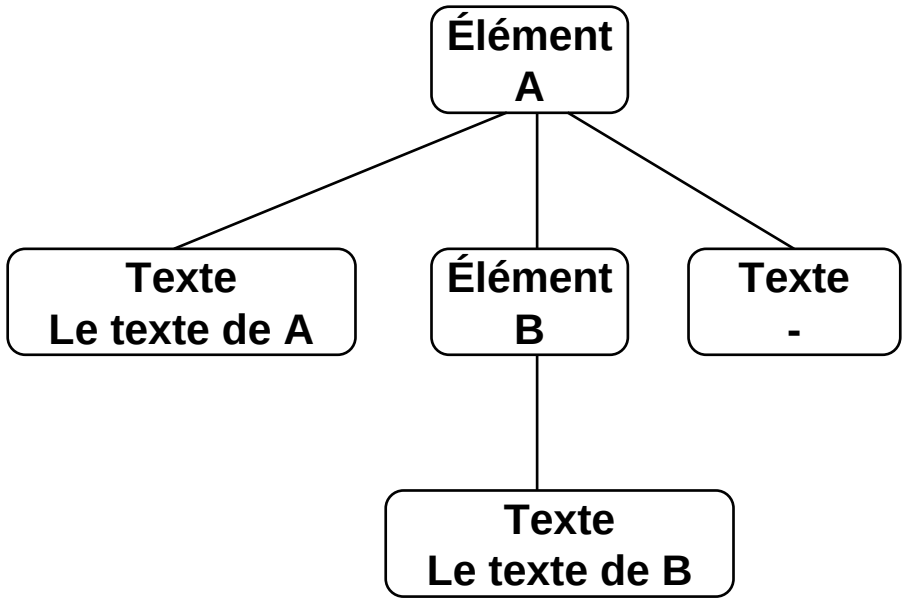
- *Exemple : un élément avec contenu*

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

  <A>Le texte de A

  <B>Le texte de B</B>

  </A>



Syntaxe : Les attributs

- Un élément ouvrant peut être enrichi par des couples de la forme attribut1=valeur1 comme dans l'exemple

  <A att1=’1’ att2=’2’>

  - l’ordre des attributs n’est pas important
  - il doit toujours y avoir une valeur (différent de HTML), encadrée par des guillemets ou des apostrophes
  - il ne peut pas y avoir deux attributs avec le même nom dans un élément. Le nom des attributs suit les mêmes règles syntaxiques que les noms d‘éléments.

Syntaxe : Les attributs (2)

- Exemple
  - <A att1=’1’ att2=’2’> est équivalent à
    <A att2=’2’ att1=’1’>
  - <A att=a> n’est pas bien formé: pas d’apostrophe
  - <A att1=’1’ att1=’2’/>: interdit
  - <A att1=’1’/> <B att1=’2’/>: autorisé (deux éléments différents)
  - <A att1=’ ″ v ″ ’/> <B att2= ″ ’2’″ />: autorisé

Syntaxe : Les attributs (3)

- Un choix de présentation : par balise ou par attribut ?
  - <produit nom="DVD" prix='200'>
  - <produit>
    <nom>DVD</nom>
    <prix>150</prix>
    </produit>
- Un critère : l'attribut doit changer l'interprétation des données :
  - <prix monnaie="Euro"> 150 </prix>

Syntaxe : Les attributs réservés

- xml:lang='langue'

  permet de définir la langue utilisée dans l‘ élément et tous les sous-éléments. La langue suit la norme ISO 3166 définie par la RFC 1766 (Request For Comment).

  Par exemple: fr ou en-US ou fr-FR

- xml:space='preserve' ou xml:space='default'

  permet de définir l'interprétation des espaces dans l‘élément et tous les sous-éléments.

Syntaxe : Les attributs réservés (2)

- xml:id='identificateur'

  permet d'associer une et une seule clef a un élément .

- xml:idref='identificateur'

  permet de faire référence a une clef.

- Exemple :

<section id='intro'> <titre>introduction a XML</titre> ... </section>	<section> <p> apres la section <xref idref='intro'>d'introduction</xref> nous allons passer au plat de résistance... </section>

Syntaxe : Entités et références à des entités

- Les *entités* servent à factoriser des parties du document. Ce sont des fragments de document XML définis dans la DTD
- La référence d'entité se note : &nom_de_l_entité;
- Il existe des entités prédéfinis :

& donne & < donne < > donne > " donne "	' donne ' &#nnn; donne le caractère de code décimal nnn, &#xnnn; donne le caractère de code hexadécimal nnn,

Syntaxe : Entités et références à des entités(2)

Exemple : utilisation des entités

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE A SYSTEM "minimal.dtd" [
    <!ENTITY monTexte "texte simple">
    <!ENTITY maSignature SYSTEM "signature.xml">
    ]>
<A>
    Du &monTexte;, sans caractères réservés:
    ni &lt; ni &gt; ni &amp; ni &apos; ni &quot;
    &maSignature;
</A>
```

Syntaxe: les sections littérales CDATA

- A priori, on n'a pas le droit de placer dans le contenu d'un document XML des caractères comme '<', '>', ou '&'.

<?xml version='1.0'?>

<PROGRAMME>

if ((i < 5) && (j > 6)) printf("error");

</PROGRAMME>

est incorrecte!

Syntaxe : les sections CDATA

- Avec les sections littérales Il est possible de stopper l'interprétation des caractères spéciaux. Elles permettent d’inclure du texte qui n’est pas analysé par le parseur :

La syntaxe est la suivante :

```
<?xml version=’1.0’?>
<PROGRAMME>
   <![CDATA[if ((i < 5) && (j > 6)) printf("error"); ]]>
</PROGRAMME>
```

Syntaxe : Les commentaires

- Les commentaires sont à utiliser avec parcimonie :

  <!-- Ceci est un commentaire -->
- Les contraintes d'utilisation sont
  - pas de double tirets dans le texte,
  - pas de commentaire dans un élément
- Exemple : Ceci est incorrect !

```
<produit
nom="DVD"
prix='100' <!-- en euros -->
/>
```

Syntaxe : Les instructions de traitement

- Les instructions de traitement introduisent un aspect procédural dans un document XML.
- Ils se notent comme suit dans un document XML

<?nomproc
attribut1='val1' ...
attributN='valeurN‘
?>

  - Cette instruction revient à appeler l'application "nomproc " en lui passant la valeur des attributs
  - Le sens des instructions de traitement depend de l'application qui traite les documents XML

Syntaxe : Les espaces de noms

- Un problème apparaît si on mélange deux textes XML dont les éléments ont le même nom.
- Pour régler ce problème on enrichit le nom de l'élément par l'identification de la source de données (URI) dans laquelle l'élément a défini.
- Le document importateur doit utiliser cette identification qui préfixe les éléments importés
- Les espaces de noms doivent être utilisés si le document XML rédigé est destiné à être mélange à d'autres sources.

Syntaxe : Les espaces de noms (2)

- Exemple :

<produit> <nom>...</nom> <desc>...</desc> </produit>	<dil:produit xmlns:i3s='http://www.i3s.unice.fr'> <i3s:nom>...</i3s:nom> <i3s:desc>...</i3s:desc> </i3s:produit>

- On peut fixer l'espace de noms par défaut avec la syntaxe :

<produit xmlns='http://www.i3s.unice.fr'>
<nom>...</nom> <desc>...</desc>
</produit>

DTD : Introduction

- Les DTD viennent de SGML
- Les DTD suivent les règles de réécriture des expressions régulières mais non pas une arborescence (pas de même structure que XML)
- Dans une DTD on trouve :
  - des déclarations d‘éléments,
  - des déclarations d'attributs,
  - des déclarations d'entités :
    - internes XML,
    - externes XML,
    - externes Non XML

DTD : Principe

- Une DTD est une description de *l'interface* entre le *producteurs* et les *consommateurs* des données/documents XML :
  - le producteur peut contrôler la qualité des données/documents produits
  - le consommateur peut séparer la vérification syntaxique des données/documents (parseur) de la logique de l'application
- DTD, pourquoi je l'utilise?

DTD : Document XML valide et bien formé

- Document XML **bien-formé**:
  - la structure est bien imbriquée (arborescence DOM)
  - pas de DTD

- Document XML **valide**:
  - Bien formé
  - respecte une DTD
  - respecte *l'intégrité référentielle* :
    - toutes les valeurs d'attributs de type ID sont distinctes
    - toutes les références sont valides

DTD : Déclaration dans le document XML

- La référence a la DTD doit être placée au début du fichier :

  <!DOCTYPE nom_er SYSTEM "test.dtd">

- On peut enrichier la DTD avec des déclarations locales :

  <!DOCTYPE nom_er SYSTEM "test.dtd" [declarations] >

- On peut se passer de DTD et définir toutes les balises dans le document XML local :

  <!DOCTYPE nom_er [declarations] >

DTD : un exemple

```
<!DOCTYPE Officiel [
<!ELEMENT Officiel (#PCDATA | cinéma | film)*>
<!ELEMENT cinéma (nom, adresse, (séance)*)>
<!ELEMENT nom (#PCDATA) >
<!ELEMENT adresse (ville, rue, (numéro)?)>
<!ELEMENT séance EMPTY>
<!ATTLIST séance heure NMTOKEN #REQUIRED
ref_film IDREF #REQUIRED>
<!ELEMENT film (titre, année>
<!ATTLIST film film_id ID #REQUIRED>
acteurs IDREFS #IMPLIED>
<!ELEMENT titre (#PCDATA) >
]>
```

DTD : utilisation dans XML

- On ajoute au début du document XML la clause DOCTYPE.
  - Définition locale:

    <!DOCTYPE Officiel [

    <!ELEMENT Officiel (#PCDATA|cinéma|film)>*

    <!ELEMENT cinéma (nom, adresse, (séance))>*

    ...]>
  - Définition externe :

    <!DOCTYPE Officiel SYSTEM "officiel.dtd">

DTD - déclaration du type d'élément

- Un élément est défini par un nom et un *modèle de contenu* :
  - *Expression régulière* sur
    - l'alphabet des noms d'éléments;
    - *EMPTY* = élément vide;
    - *ANY* = toute combinaison de tous les éléments;
    - *#PCDATA* = texte
  - Contenu mixte : *(#PCDATA | A | B ...)**

DTD - Expression régulière

- Expressions régulières sur un alphabet N :
  - tout symbole n ϵN est une expression régulière (e.r.);
  - Si e est une expression régulière, alors (e)*, (e)+, (e)? sont des expressions régulières;
  - Si e1 et e2 sont des expressions régulières, alors (e2,e2), (e1|e2) sont des expressions régulières;

DTD - Langage régulier (1)

Chaque expression régulière e sur un alphabet N définit un ensemble de mots (langages) $L(e)$ sur N :

- $L(a) = \{a\}$ pour tous les a dans N;
- $L(e?) = L(e) \cup \{\epsilon\}$: ϵ désigne le mot vide;
- $L(e_1, e_2) = \{m_1 m_2 \mid m_1 \in L(e_1) \wedge m_2 \in L(e_2)\}$: tous les mots composés d'un mot dans $L(e_1)$ suivi d'un mot dans $L(e_2)$
- $L(e_1|e_2) = L(e_1) \cup L(e_2)$: tous les mots dans $L(e_2)$ et dans $L(e_2)$ (union)

DTD - Langage régulier (2)

- $L(e+) = \{m_0 m_1 \ldots m_n \mid m_i \in L(e)\}$: tous les mots composés de mots dans $L(e)$;
- $L(e*) = L(e+) \cup \{\epsilon\}$;

Exemples :

- $L(a, b) = \{ab\}$, $L(a \mid b) = \{a, b\}$
- $L(a*, b) = \{b, ab, aab, aaab, \ldots\}$
- $L((a, b)*) = \{\epsilon, ab, abab, ababab, \ldots\}$
- $L((a|b)*) = \{\epsilon, a, b, ab, ba, aaa, aab, aba, \ldots\}$

DTD – exemple d'élément

- Un cinéma a
  - un nom, une adresse optionnelle et
  - une suite de séances.

  <!ELEMENT cinéma (nom,adresse?,(séance))>*
- Une personne a
  - un nom, plusieurs numéros de téléphone et
  - au moins une adresse email

  <!ELEMENT personne (nom,tel,email+)>*

DTD – Quelques modèles d'éléments

- Le modèle libre
  - <!ELEMENT doc ANY>
- Le modèle libre
  - <!ELEMENT toc EMPTY>
- Le modèle textuel (Parsed Character Data)
  - <!ELEMENT commentaire (#PCDATA)>

DTD – Quelques modèles d'éléments(2)

- Le modèle composé
  - <!ELEMENT produit (nom,prix,#PCDATA)>
  - <!ELEMENT nom (#PCDATA)>
  - <!ELEMENT prix (#PCDATA)>
- Exemple

```
<produit>
    <nom>XML par la pratique</nom>
    <prix>20</prix>
    Comment programmer des applications XML.
</produit>
```

DTD – Quelques modèles d'éléments(3)

- Une définition équivalente (de préférence)
  - <!ELEMENT produit (nom,prix,comment)>
  - <!ELEMENT nom (#PCDATA)>
  - <!ELEMENT prix (#PCDATA)>
  - <!ELEMENT comment (#PCDATA)>
- Exemple

```
<produit>
        <nom>XML par la pratique</nom>
        <prix>20</prix>
        <comment>Comment programmer des applications
    XML</comment>
</produit>
```

DTD – Quelques modèles d'éléments(4)

- Le modèle mixte

  <!ELEMENT reponse (#PCDATA|oui|non)>

  <!ELEMENT oui EMPTY>

  <!ELEMENT non EMPTY>

- Exemple :

Exemple	Contre-exemple
<reponse> <oui/> </reponse> <reponse> <non></non> </reponse> <reponse> peut-être </reponse> <reponse/>	<reponse> <oui/><non/> </reponse> <reponse> <oui/><oui/> </reponse>

DTD – Quelques modèles d'éléments(5)

- Le modèle combiné
  - <!ELEMENT personne (nom,(adr|email))>
- Exemple :

```
<personne>
    <nom>...</nom>
    <adr>49 Bd de la Revolution</adr>
</personne>
<personne>
    <nom>...</nom>
    <email>tartanpion@...</email>
</personne>
```

DTD – Quelques modèles d'éléments(6)

- Le modèle combiné : autres formes possibles
  - <!ELEMENT personne ((adr|email),nom)>
  - <!ELEMENT personne (((age,adr)|email),nom)>
  - <!ELEMENT personne (((age,adr)|(age,email)),nom)>
  - <!ELEMENT personne (age,(adr|email),nom)>

DTD – Quelques modèles d'éléments(7)

- Répétition de modèles :
  - modèle* zéro ou plusieurs occurrences,
  - modèle+ au moins une occurrence,
  - modèle? zéro ou une occurrence
- Exemple
  - <!ELEMENT chapitre (
    nom,date?,auteur*,intro?,
    (nom-de-section,corps-de-section)+
)>

DTD – Quelques modèles d'éléments(8)

- Document XML valide (par rapport à l'exemple)
  - <chapitre>
    <nom>Utiliser les DTD</nom>
    <date>10/11/2002</date> <!-- optionnel -->
    <!-- 0 a n fois -->
    <auteur>...</auteur><auteur>...</auteur>
    <!-- optionnel -->
    <intro>...</intro>
    <!-- 1 a n fois -->
    <nom-de-section>Preambule</nom-de-section>
    <corps-de-section>
    </chapitre>

DTD – Déclaration d'Attributs

- Syntaxe :

  <!ATTLIST nom_element
  nom_attribut_1 type_attribut_1 declar_de_defaut
  nom_attribut_2 type_attribut_2 declar_de_defaut
  ...
  >

- Déclarations de défaut
  - 'valeur' valeur par défaut,
  - #REQUIRED l'attribut doit être renseigné,
  - #IMPLIED l'attribut est facultatif,
  - #FIXED 'valeur' l'attribut a toujours la même valeur

DTD – Déclaration d'Attributs (2)

- *Exemple Cinéma*
  - Les éléments de type *séance* ont un attribut *heure* et un attribut *ref_film*:

    <!ATTLIST séance heure NMTOKEN #REQUIRED ref_film IDREF #REQUIRED>
  - Les éléments de type *film* ont un attribut *film_id* et un attribut *acteurs* :

    <!ATTLIST film film_id ID #REQUIRED acteurs IDREFS #IMPLIED>

DTD - Types d'attribut

- Chaînes de caractères : CDATA
- Énumérations : séquences de valeurs alternatives séparées par |
- ID, IDREF, IDREFS : identifiants et références
- ENTITY/ENTITIES : entité(s)
- NMTOKEN/NMTOKENS : chaîne(s) de caractères sans blancs
- NOTATION : notation

DTD - Types d'attribut (2)

- Les attributs de type ID et IDREF : La validation va vérifier (dans un seul document)
  - l'unicite des clefs,
  - les contraintes de référence.

Modèle	Document valide
<!ELEMENT p (#PCDATA)> <!ATTLIST p id ID #IMPLIED> <!ELEMENT xref (#PCDATA)> <!ATTLIST xref idref IDREF #REQUIRED>	Ceci est une intro... ... Ce detail est explique dans <xref idref="intro">l'introduction</xref>.

DTD - Types d'attribut (3)

- Les attributs de type ENTITY ou ENTITIES :
  - L'attribut prend comme valeur une entité externe non XML (par exemple une image)
- Les attributs de type NMTOKEN ou NMTOKENS :
  - L'attribut prend comme valeur un ou plusieurs tokens XML sépares par des blancs

Modèle	Document valide
<!ELEMENT exo (#PCDATA)> <!ATTLIST exo niveau NMTOKENS #IMPLIED>	<exo niveau="facile"> ... </exo> <exo niveau="difficile facultatif"> ... </exo>

DTD - Types d'attribut (4)

- Les attributs de type liste de valeurs :
- Exemple :
  - <!ELEMENT prix (#PCDATA)>
    <!ATTLIST prix monnaie (euros|francs) #REQUIRED>

  - Fragment d'un document XML valide :
    <prix monnaie="euros"> 10 </prix>
- Les valeurs possibles doivent être des tokens XML.
- Les listes de valeurs n'existent que pour les attributs

DTD - Modes d'ATTLIST

- Mode d'attributs (déclaration de défaut)
  - #REQUIRED : la valeur doit être définie
  - #IMPLIED : la valeur est optionnelle
  - #FIXED : la valeur est constante
- Exemple

<!ATTLIST séance heure NMTOKEN #REQUIRED
ref_film IDREF #REQUIRED>
<!ATTLIST film film_id ID #REQUIRED
acteurs IDREFS #IMPLIED
langue (AN|FR|AL|ES|IT) #IMPLIED>
<!ATTLIST adresse ville CDATA #IMPLIED 'Paris'>

DTD - Entités générales et paramètres

```
<!DOCTYPE Officiel [
   <!ENTITY copyright 'Copyright B. Amann'>
   <!ELEMENT Officiel (p, année) >
   <!ELEMENT p (#PCDATA) >
   <!ENTITY % text '#PCDATA'>
   <!ELEMENT année (%text;) >
]>
<Officiel>
   <p> %copyright; </p><année>2000</année>
</Officiel>
```

Les entités paramètres (*ENTITY %*) peuvent seulement être utilisées dans la DTD.

DTD - Entités internes

- Les entités internes sont des macros qui permettent de factoriser et de paramétrer les documents XML :
  - <!ENTITY nom_entite "valeur">
- les références se font par « &nom_entite; »
- les définitions circulaires sont interdites,
- le développement se fait à la demande,
- ils sont utilisables dans les documents XML.

DTD - Entités internes (2)

- Exemple
  - <!ENTITY dom.unice "unice.fr" >
  - <!ENTITY dom.iut « iut.&dom.unice;" >
- Fragment de document XML valide :
  - <email> nlt@&dom.iut; </email>
  - <!-- erreur -->
  - ...

DTD - Entités externes XML

- Les entités externes XML permettent d'inclure des documents XML identiés par une URL
- Exemple
  - <?xml version="1.0" encoding="iso-8859-1" ?>
    <!DOCTYPE livre SYSTEM "livre.dtd"[
    <!ENTITY chap1 SYSTEM "chapitre1.xml.inc">
    <!ENTITY chap2 SYSTEM "chapitre2.xml.inc">] >
    <livre>
    &chap1;
    &chap2;
    </livre>

DTD - Entités externes XML (2)

- Contraintes sur le document XML inclu (référencé par l'entité externe) :
  - il peut contenir un prologue (encodage),
  - pas de référence a une DTD,
  - il doit être bien forme, c'est a dire contenir un
  - élément complet

DTD - Entités externes non XML

- Une entité externe non XML peut contenir n'importe quoi (image, son, données, etc.)
- Nous devons donc, au préalable, définir un type

  <!NOTATION nom_du_type SYSTEM "url_associe_au_type">

Exemples

- <!NOTATION jpeg SYSTEM "/usr/bin/xv">
- <!NOTATION gif SYSTEM "/usr/bin/gv">
- <!NOTATION pdf SYSTEM "http://www.acrobat.com">

DTD - Entités non XML

- Utilisation:
  - déclaration du format (type = application) pour entités non-XML
  - référence à une entité de type notation seulement possible comme valeur d'attribut

```
<!DOCTYPE exemple [
    <!NOTATION gif SYSTEM '/usr/local/bin/xv' >
    <!ENTITY myphoto SYSTEM './moi.gif' NDATA gif >
    <!ELEMENT person EMPTY >
    <!ATTLIST person photo NOTATION (gif) #IMPLIED>
]>
<person photo='myphoto' >
```

(&myphoto; est IMPOSSIBLE!)

DTD : Clauses spéciales

- Les clauses INCLUDE et IGNORE
- Deux exemples :
  - <![IGNORE[
    ... partie de la DTD a ignorer ...
    ]]>
  - <![INCLUDE[
    ... partie de la DTD a traiter ...
    ]]>

DTD : Entités paramètres interne

- Les entités paramètres sont des macros qui permettent de factoriser et de paramétrer les DTD
- Syntaxe : <!ENTITY %nom_entité "valeur">
- Le référencement se note %nom_entité;
- Exemple

```
<!ENTITY % statut "statut (public|prive) #IMPLIED
   'public'">
<!ELEMENT article (#PCDATA)>
<!ATTLIST article date CDATA #IMPLIED %statut;>
```

DTD : Entités paramètres externe

- Les entités paramètres externes permettent d'inclure des DTD externes.
- Exemple

```
<!ENTITY % chapitre SYSTEM "chapitre.mod">
<!ENTITY % tableau SYSTEM
   "http://monserveur.fr/dtd/tableau.mod">
%chapitre;
%tableau;
```

DTD : partie interne / partie externe

Dans une définition

<!DOCTYPE livre SYSTEM "livre.dtd"[…partie interne ...]>

- La partie interne est traitée avant la partie externe (DTD externe).
- Il est donc possible de faire varier la DTD à partir des déclarations internes
- Les clauses IGNORE et INCLUDE ne sont pas utilisables dans la partie interne

DTD - Résumé sur DTD

- Une DTD décrit la structure d'un ensemble de documents XML valides ;
- Tous les parseurs XML permettent de valider un document XML par rapport à une DTD;
- Une DTD n'est pas un document XML;
- Il existent des langages plus riches pour la description d'un document XML : XML Schema, Relax NG => grammaires d'arbres régulières

Exemple d'application XML (de Rigaux)

- Problème : de base de données à XML
  - Prenons l'exemple (très simplifié) de la base de données d'un organisme de voyage. Soient quatre tables suivantes (avec les données)
    - Station (**nomStation**, capacité, lieu, région, tarif)
    - Activité (***nomStation***, **libellé**, prix)
    - Client (**id**, nom, prénom, ville, région, solde)
    - Séjour (***idClient***, **station**, **début**, nbPlaces)
  - Présenter cette base de données avec XML

Exemple d'application XML (2)

- Les données dans les tables :

id	nom	prénom	ville	région	solde
10	Fogg	Phileas	Londres	Europe	12465
20	Pascal	Blaise	Paris	Europe	6763
30	Kerouac	Jack	New York	Amérique	9812

La table *Client*

nomStation	capacité	lieu	région	tarif
Venusa	350	Guadeloupe	Antilles	1200
Farniente	200	Sicile	Europe	1500
Santalba		Martinique	Antilles	2000
Passac	400	Alpes	Europe	1000

La table *Station*

idClient	station	début	nbPlaces
10	Passac	2001-07-01	2
30	Santalba	2001-08-14	5
20	Santalba	2001-08-03	4
30	Passac	2001-08-15	3
30	Venusa	2001-08-03	3
20	Venusa	2001-08-03	6
30	Farniente	2002-06-24	5
10	Farniente	2002-09-05	3

La table *Séjour*

nomStation	libellé	prix
Venusa	Voile	150
Venusa	Plongée	
Farniente	Plongée	130
Passac	Ski	200
Passac	Piscine	20
Santalba	Kayac	50

La table *Activité*

Exemple d'application XML (3)

- Méthodologies
  - Garder la structure plate relationnelle : chaque table sera une structure dans XML. On définir ligne par ligne et table par table
    - Avantage : facilité de la traduction
    - Inconvénient : ne pas exploiter la capacité de XML dans la présentation de données hiérarchiques avec une structure dynamique
  - Étudier une nouvelle présentation prenant en compte la sémantique de données mais également les avantage de langage XML
  - Nous suivons la deuxième méthodologie

Exemple d'application XML (4)

- Un peu d'analyse :
  - Éléments ou attributs ?
  - Quel chemin d'accès principal ?
  - Quel est l'élément racine ?
- Choix :
  - les colonnes sont représentées par des attributs XML;
  - le chemin d'accès principal est la station ;
  - pour chaque station on trouve, imbriqués, les séjours de la station, et dans chaque séjour les clients qui ont séjourné dans la station ;
  - pour les besoins de la présentation, on va supposer que les activités de la station sont représentées par des éléments indépendants, avec un lien de navigation
  - Les documents auront un élément racine de type Stations, constitué de 0 ou plusieurs éléments de type Station et de 0 ou plusieurs éléments de type Activite.

Exemple d'application XML (5)

- Construction le DTD pas à pas :
  - Définition de l'élément racine Stations
    - <!ELEMENT Stations (Station*,Activite*)>
  - Définition de l'élément Station
    - <!ELEMENT Station (Sejour*)>
      <!ATTLIST Station
      nomStation ID #REQUIRED
      capacite CDATA #IMPLIED
      lieu CDATA #REQUIRED
      tarif CDATA #REQUIRED
      region (Océan_Indien|Antilles|Europe|Amérique|Asie) #REQUIRED
      >

Exemple d'application XML (6)

- Construction le DTD pas à pas :
  - Définition de l'élément Séjour et Client

```
<!ELEMENT Sejour (Client)>
<!ATTLIST Sejour
debut CDATA #REQUIRED
nbPlaces CDATA #REQUIRED
>
```

  - Définition de l'élément Activite

```
<!ELEMENT Activite EMPTY>
<!ATTLIST Activite
nomStation IDREF #REQUIRED
libelle CDATA #REQUIRED
prix CDATA #IMPLIED
>
```

Exemple d'application XML (7)

- La DTD finale :

```
<!-- DTD des documents exportés de la base
"Agence de voyages" -->
<!ELEMENT Stations (Station*,Activite*)>
<!ELEMENT Station (Sejour*)>
<!ATTLIST Station
nomStation ID #REQUIRED
capacite CDATA #IMPLIED
lieu CDATA #REQUIRED
tarif CDATA #REQUIRED
region (Océan_Indien|Antilles|Europe|Amérique|Asie)
#REQUIRED >
<!ELEMENT Sejour (Client)>
<!ATTLIST Sejour
debut CDATA #REQUIRED
nbPlaces CDATA #REQUIRED >
```

```
<!ELEMENT Client EMPTY>
<!ATTLIST Client
id ID #REQUIRED
nom CDATA #REQUIRED
prenom CDATA #REQUIRED
ville CDATA #REQUIRED
region CDATA #REQUIRED
solde CDATA #REQUIRED
>
<!ELEMENT Activite EMPTY>
<!ATTLIST Activite
nomStation IDREF #REQUIRED
libelle CDATA #REQUIRED
prix CDATA #IMPLIED
>
```

Exemple d'application XML (8)

- La base de données :

```
<?xml version='1.0' encoding='ISO-8859-1?>
<Stations>
<Station nomStation='Venusa'
    capacite='350'
    lieu='Guadeloupe'
    region='Antilles'
    tarif='1200.00'>
    <Activite libelle='Voile' prix='150.00'/>
    <Activite libelle='Plongee' prix='130.00'/>
<Sejour idClient='30'
    debut='2001-08-03'
    nbPlaces='3'/>
    <Sejour idClient='20'
    debut='2001-08-03'
    nbPlaces='6'/>
</Station>
```

```
<Station nomStation='Farniente'
    capacite='200'
    lieu='Seychelles'
    region='Océan Indien'
    tarif='1500.00'>
    <Activite libelle='Plongée' prix='130.00'/>
    <Sejour idClient='30'
    debut='2002-06-24'
    nbPlaces='5'/>
    <Sejour idClient='10'
    debut='2002-09-05'
    nbPlaces='3'/>
</Station>
```

Exemple d'application XML (9)

- La base de données (suite):

```
<Client id='10'
nom='Fogg'
prenom='Phileas'
ville='Londres'
region='Europe'
solde='12465.00'
/>
<Client id='20'
nom='Pascal'
prenom='Blaise'
ville='Paris'
region='Europe'
solde='6763.00'
/>
```

```
    <Client id='30'
nom='Kerouac'
prenom='Jack'
ville='New York'
region='Amérique'
solde='9812.00'
/>
</Stations>
```