
XML: Syntaxe, DOM/SAX et DTD

Bernd Amann et Philippe Rigaux

XML (eXtensible Markup Language)

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.

Notion de balisage structurel

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

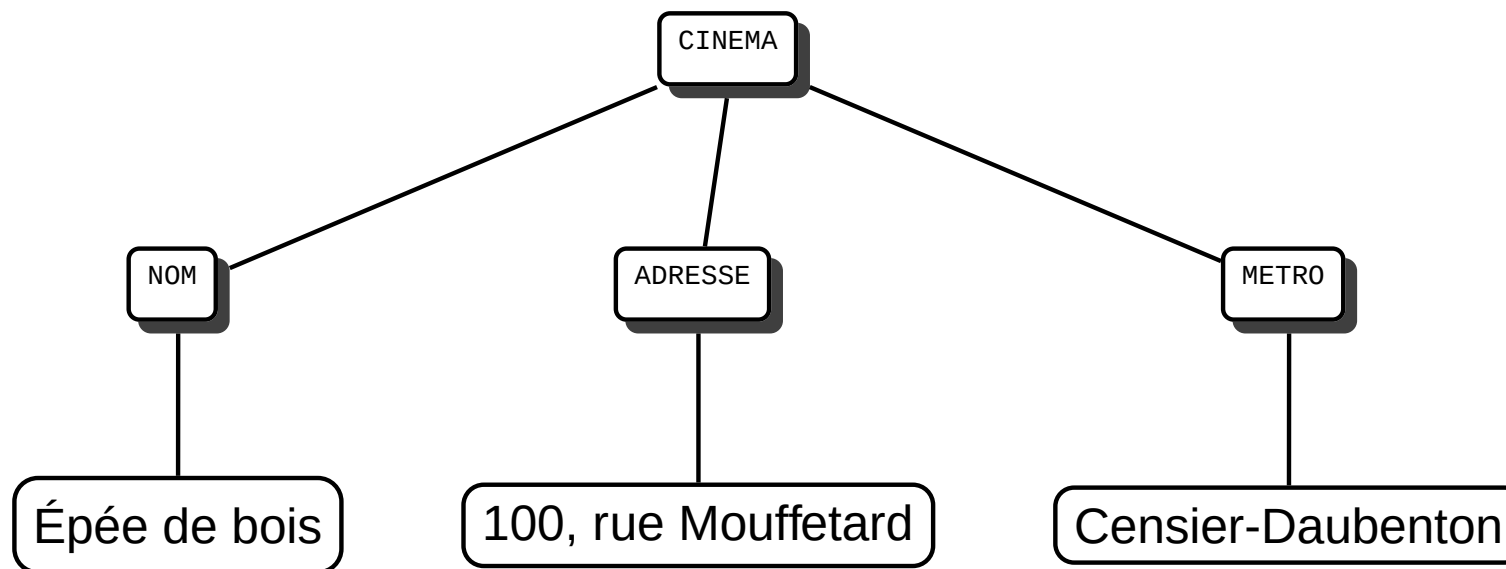
Forme sérialisée

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CINEMA>
  <NOM>
    Epée de Bois
  </NOM>
  <ADRESSE>
    100, rue Mouffetard
  </ADRESSE>
  <METRO>
    Censier-Daubenton
  </METRO>
</CINEMA>
```

Cette représentation facilite l'échange de documents.

Forme arborescente (DOM)

Un arbre, constitué de nœuds typés (éléments, commentaires, etc)



La structure des arbres XML est définie par le *modèle* (DOM).

Il est plus facile de raisonner sur la forme arborescente pour concevoir des traitements

Passage de la forme sérialisée à la représentation DOM

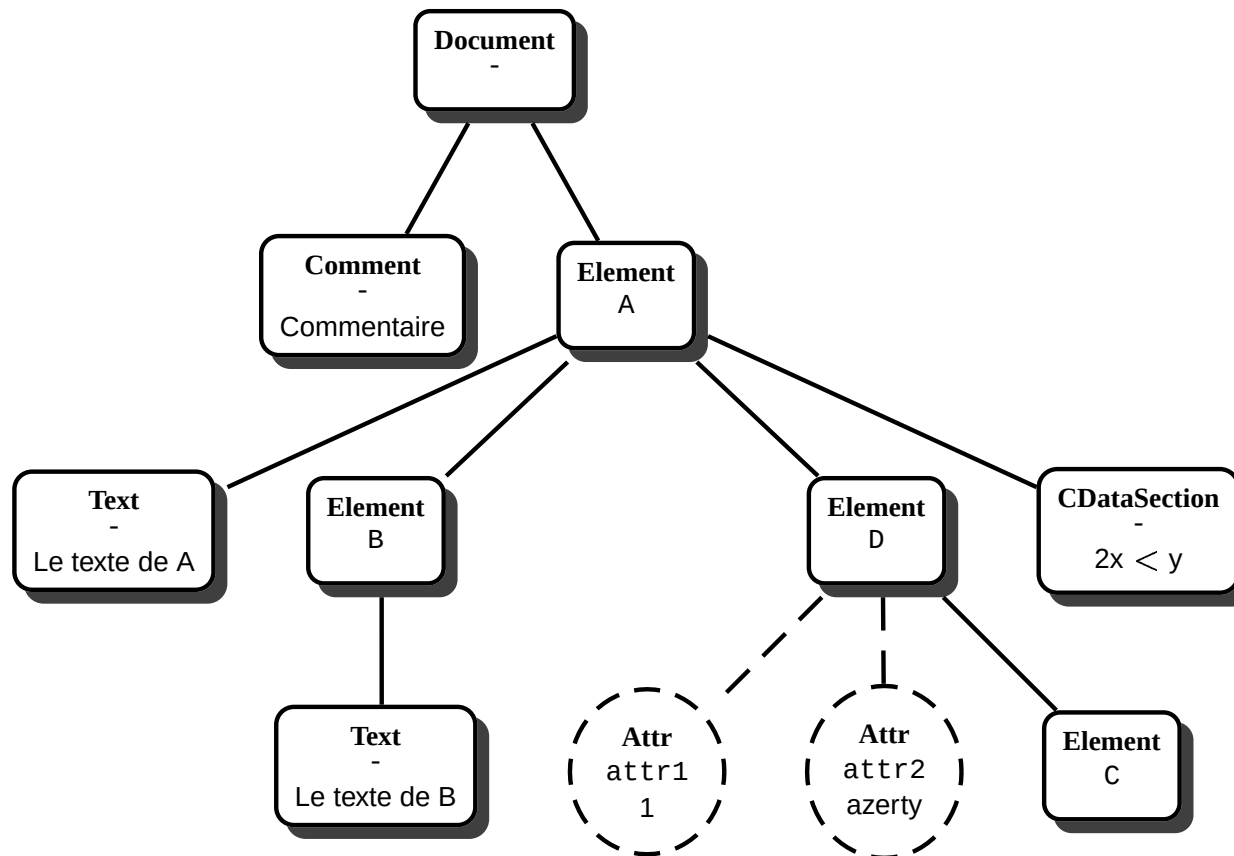
Le document sérialisé est analysé, et une représentation arborescente est créée :

- le nœud racine est de type **Document**
- les catégories syntaxiques (commentaires, balises, texte) se traduisent par différents types de nœuds (**Comment**, **Element**, **Text**)
- les nœuds constituent un arbre qui reflète l'imbrication des éléments dans la forme sérialisée

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>
```

La représentation DOM



La syntaxe XML

La syntaxe XML

La syntaxe XML permet la représentation sérialisée d'un arbre DOM :

- les éléments (et leurs attributs)
- les commentaires
- les instructions de traitement
- les sections de texte
- les sections littérales

Ainsi que quelques règles sur la structure d'un document

La déclaration XML

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

Déclaration de type

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 un source extérieure contenant la DTD
- *decLoc* sont des déclarations locales (pour les entités principalement)

Entités et références à des entités

Les *entités* servent à factoriser des parties du document.

```
<?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>
```

Entités caractères

Déclaration entité	Référence	Car.
<!ENTITY lt "<">	<	<
<!ENTITY gt ">">	>	>
<!ENTITY amp "&">	&	&
<!ENTITY apos "'">	'	'
<!ENTITY quot """>	"	"

Table 1:

Commentaires et instructions de traitement

- Les commentaires : à utiliser avec parcimonie :
`<!-- Ceci est un commentaire -->`
- Instructions de traitement : lié au système qui traite le document:
`<?xml-stylesheet href="prog.xslt" type="text/xslt">`

Éléments

Dans la forme sérialisée :

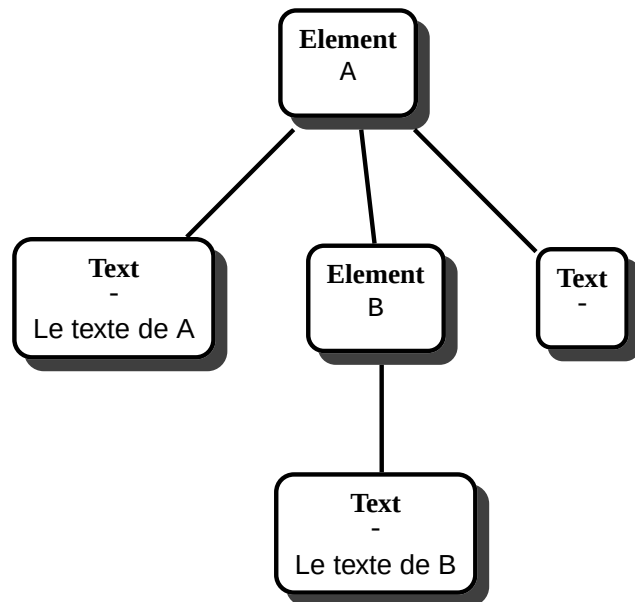
- C'est une **balise ouvrante** avec un nom, puis un **contenu**, puis une **balise fermante**

Dans la forme arborescente

- C'est un nœud avec un nom
- Le contenu est un arbre

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>
```



Quelques remarques sur les éléments

- Un nom d'élément ne contient pas de blanc, ni de caractère accentué
- Les majuscules sont distinguées des minuscules
- Il existe une forme abrégée pour les éléments sans contenu : `<C></C>` peut s'écrire `<C/>`.
- **Tout document comprend un et un seul élément racine**

Attributs

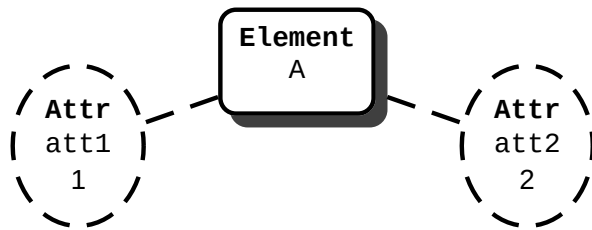
Les attributs constituent un **autre** moyen de représenter de l'information.

```
<A att1='1' att2='2'>
```

- l'ordre des attributs n'est pas important
- il doit toujours y avoir une valeur, encadrée par des guillemets (différent de HTML)
- il ne peut pas y avoir deux attributs avec le même nom dans un élément.

Attributs : exemples

- `<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)

Les sections littérales

À 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!

Solution : sections CDATA

Elles permettent d'inclure du texte qui n'est pas analysé par le parseur :

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

CDATASection

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

Résumé : structure d'un document XML

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.

Document Object Model (DOM)

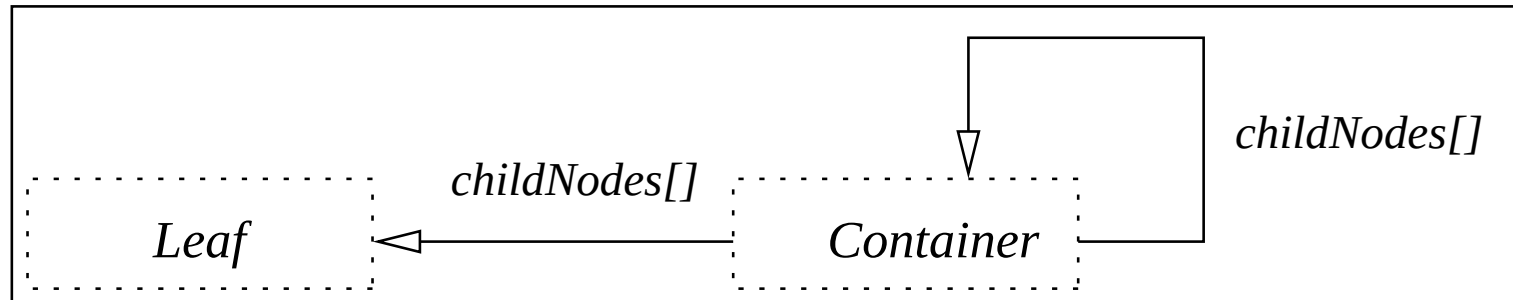
Le modèle DOM

Un parseur DOM prend en entrée un document XML et construit un **arbre** formé **d'objets** :

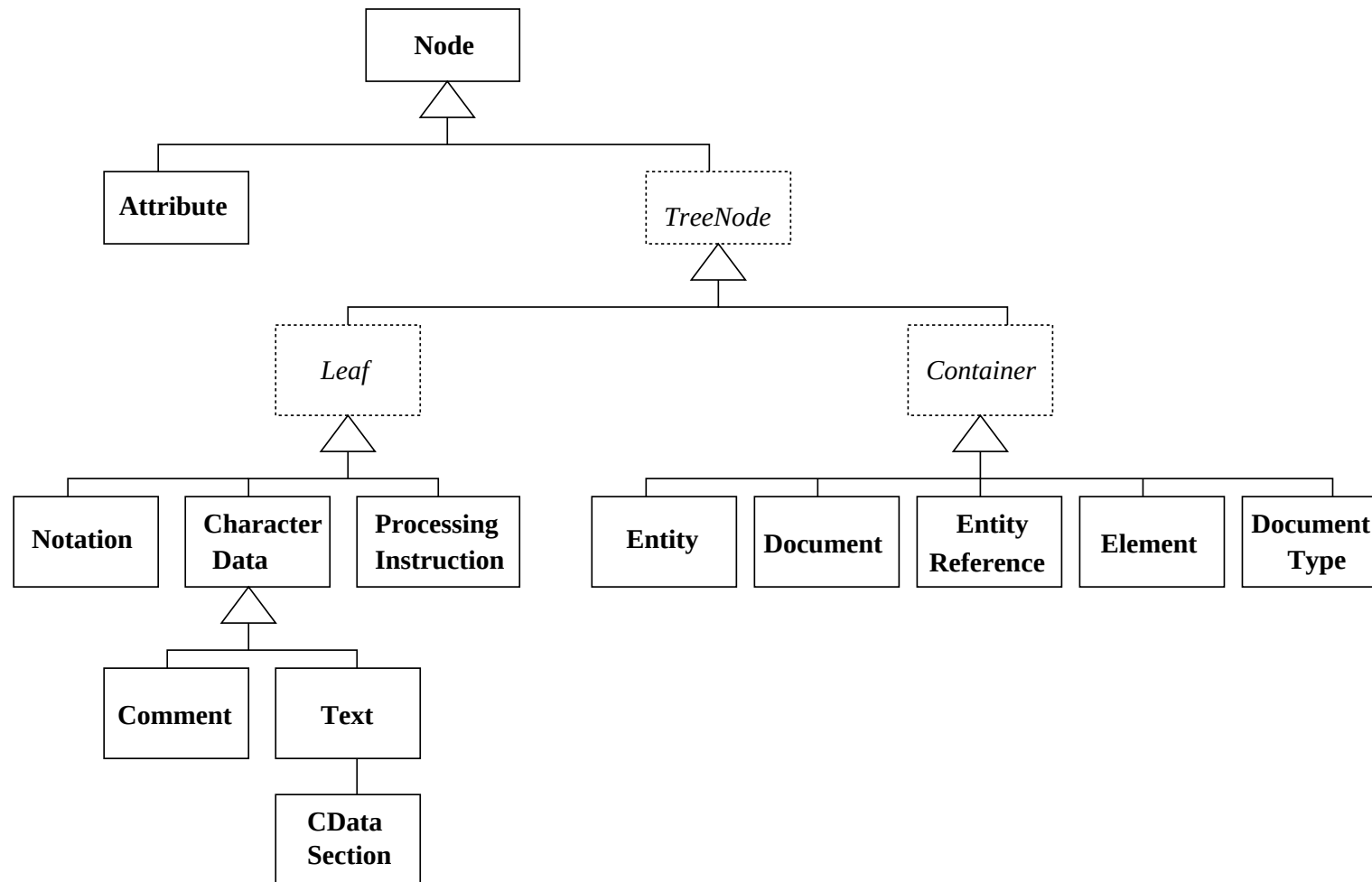
- chaque objet appartient à une sous-classe de Node
- des opérations sur ces objets permettent de **créer** de nouveaux nœuds, ou de **naviguer** dans le document

=> éditeurs XML, processeurs XSLT

Structure d'un arbre DOM



Les types de noeuds DOM



Propriétés d'un Noeud

Propriété	Type
<i>nodeType</i>	unsigned short
<i>nodeName</i>	DOMString
<i>nodeValue</i>	DOMString
<i>attributes</i>	NamedNodeMap

Table 2: Propriétés du type **Node**

Propriétés du type Node

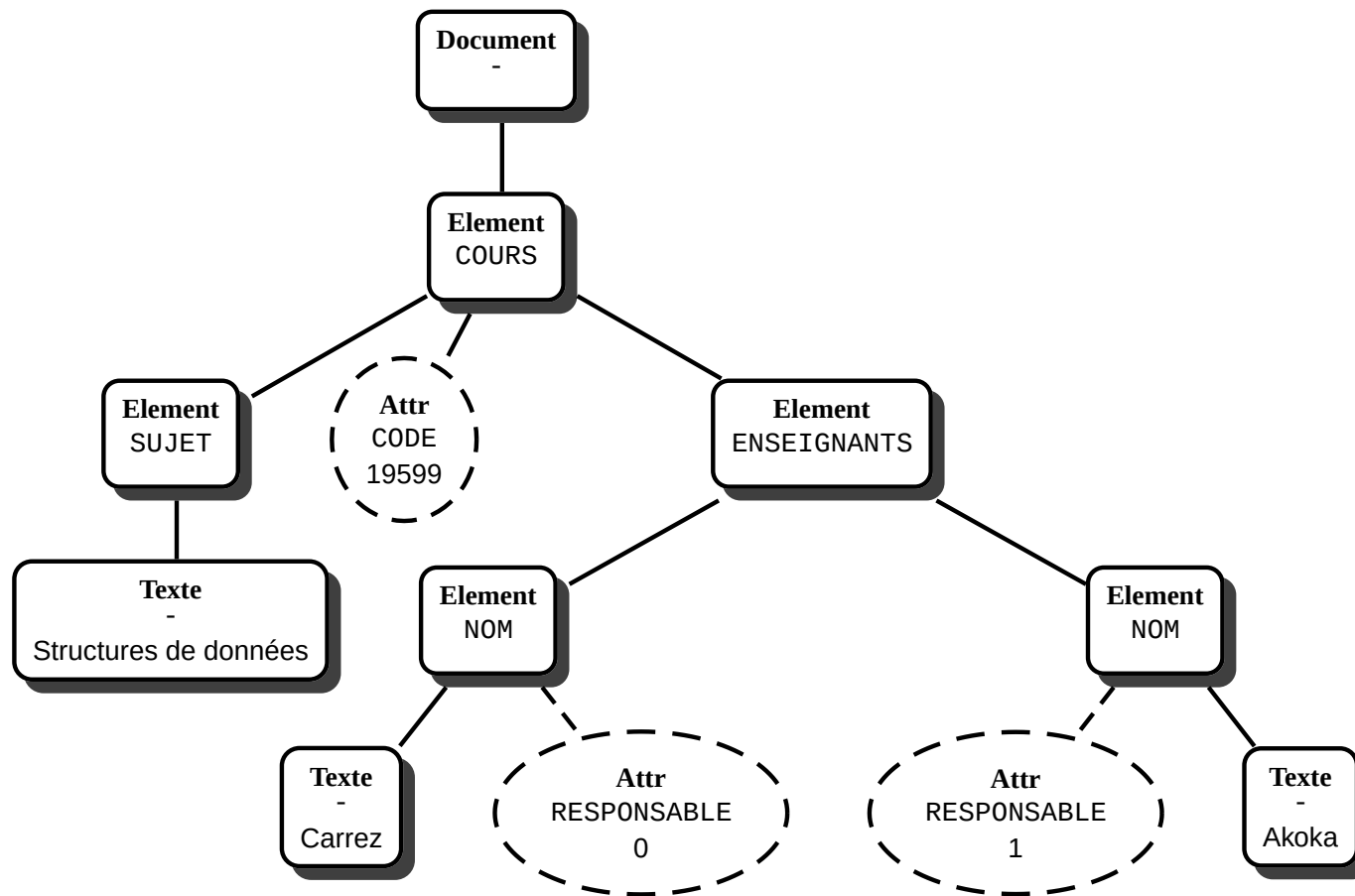
Propriété	Type
<i>parentNode</i>	Node
<i>firstChild</i>	Node
<i>lastChild</i>	Node
<i>childNodes</i>	NodeList
<i>previousSibling</i>	Node
<i>nextSibling</i>	Node

Table 3: Propriétés du type **Node**

Un exemple

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<COURS CODE="19599">
  <SUJET>Structures de données</SUJET>
  <ENSEIGNANTS>
    <NOM RESPONSABLE="0">Carrez</NOM>
    <NOM RESPONSABLE="1">Akoka</NOM>
  </ENSEIGNANTS>
</COURS>
```

L'arbre DOM



Opérations du type **Node**

Résultat	Méthode	Paramètres
Node	<i>insertBefore()</i>	Node <i>nouv</i> , Node <i>fil</i>
Node	<i>replaceChild()</i>	Node <i>nouv</i> , Node <i>anc</i>
Node	<i>removeChild()</i>	Node <i>fil</i>
Node	<i>appendChild()</i>	Node <i>nouv</i>
boolean	<i>hasChildNodes()</i>	
Node	<i>cloneNode()</i>	boolean <i>prof</i>

Table 4: Opérations du type **Node**

DOM : Exemple de code

Ajoute

```
<adventure type="epic">&adventure1;</adventure>
```

avant le 4e élément du document :

```
var racine = myDocument.documentElement;  
var enfants = racine.childNodes;
```

```
var el_nouv = createElement("adventure") ;  
var ent_ref = createEntityReference("adventure1");
```

```
el_nouv.setAttribute("type", "epic");  
el_nouv.appendChild(ent_ref)
```

```
insertBefore(el_nouv, enfants.item(3));
```

L'API SAX

Programmer avec XML

Les deux principales interfaces de programmation XML :

- DOM (*Document Object Model*), basé sur une représentation arborescente
- SaX (*Simple API for XML*), basé sur des *déclencheurs* “événement/action”

Les API XML

L'API DOM :

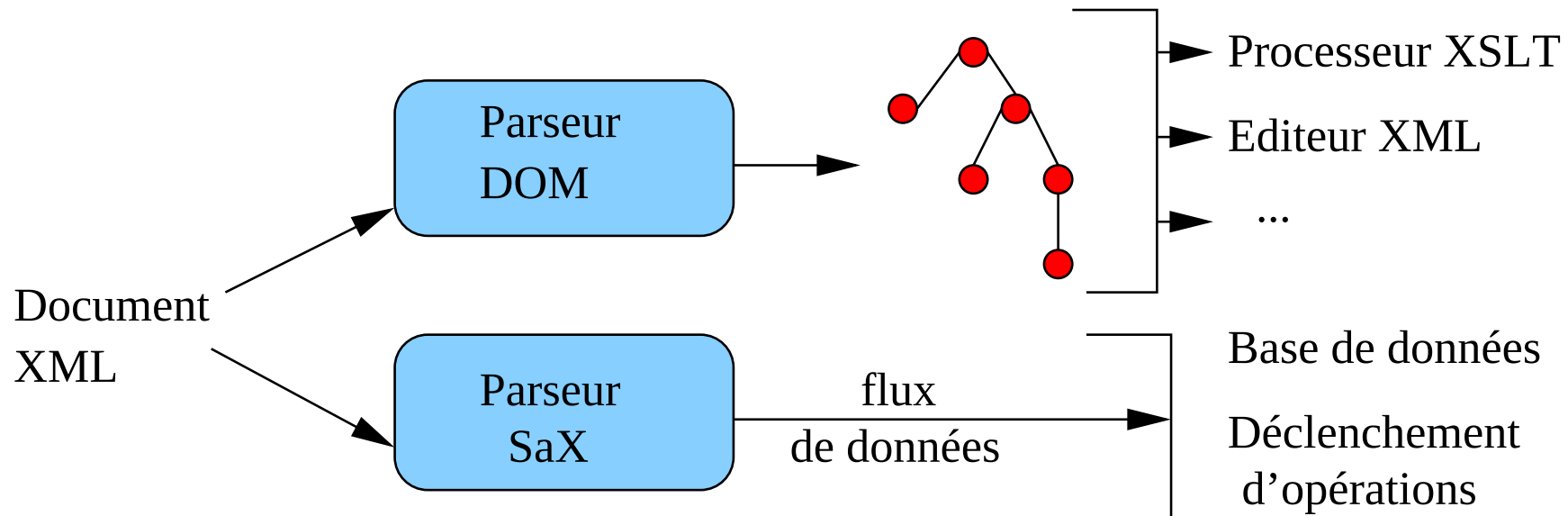
- Construit une représentation du document en mémoire sous forme d'arbre
- Adaptée aux applications qui modifient ou traitent dans leur globalité un document.

L'API SaX :

- Définit des *triggers* qui se déclenchent sur certaines balises.
- Adaptée aux applications qui extraient de l'information d'un document

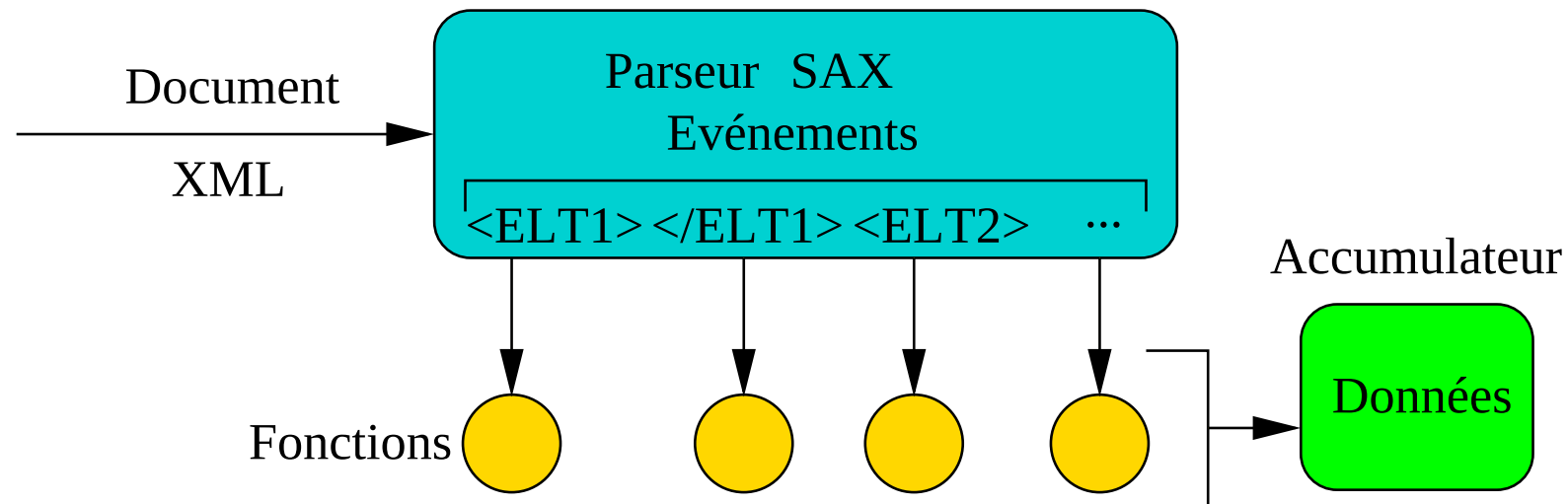
DOM et SaX

Toutes les applications XML passent par une phase préalable d'analyse



Architecture SAX

Associer des événements aux balises :



Les événements SAX

Un parseur SAX génère un événement à chaque fois qu'il rencontre

- le début et la fin de document,
- le début et la fin d'un élément,
- une instruction de traitements
- un commentaire, ...

Les fonctions s'exécutent indépendamment => il faut leur faire partager une zone mémoire (« accumulateur »)

Les événements SAX: exemple

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.apache.xerces.parsers.SAXParser;
public class Trace extends DefaultHandler {
    // début de document
    public void startDocument() {
        System.out.println("start document");
    }
    // fin de document
    public void endDocument() {
        System.out.println("end document");
    }
}
```

Suite...

```
// balise d'ouverture
public void startElement(String uri,
                        String localName, String qName,
                        Attributes attributes) {
    System.out.println("starting element: " +
                      qName);
}
// balise de fermeture
public void endElement(String uri,
                      String localName, String qName) {
    System.out.println("end element: " + qName);
}
```

Suite...

```
// espace blanc
public void ignorableWhitespace(char[] ch,
    int start, int length) {
    System.out.println("whitespace, length " +
        length);
}
// instruction de traitement
public void processingInstruction(String target,
    String data) {
    System.out.println("processing instruction: " +
        target);
}
```

Suite...

```
// texte
public void characters(char[] ch, int start,
    int length){
    System.out.println("character data, length " +
        length);
}
// procédure principale
public static void main(String[] args) {
    Trace t = new Trace();
    SAXParser p = new SAXParser();
    p.setContentHandler(t);
    try { p.parse(args[0]); }
    catch (Exception e) {e.printStackTrace();}
}
```

Exemple type : insertion dans une BD

À partir d'un flux XML contenant des films :

```
<FILMS>
  <FILM>
    <TITRE>Alien</TITRE>
    <ANNEE>1979</ANNEE>
    <AUTEUR>Ridley Scott</AUTEUR>
    <GENRE>Science-fiction</GENRE>
    <PAYS>USA</PAYS>
  </FILM>
</FILMS>
```

Les événements

- Début de document : connexion à la base
- Balise <FILM> : création d'un enregistrement *Film*
- Balise <TITRE> : on affecte *Film.titre*
- Balise <ANNEE> : on affecte *Film.annee*
- etc..
- Balise <FILM> : insertion de l'enregistrement dans la base
- Balise </FILM> : destruction de *Film*

Déclaration du Type de Document (DTD)

Documents XML valides et bien-formés

Document XML **bien-formé**:

- pas de DTD
- la structure est imbriquée (arborescence)

Document XML **valide**:

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

DTD: Exemple

```
1  <!ELEMENT Officiel (#PCDATA | cinéma | film)*>
2  <!ELEMENT cinéma (nom, adresse, (séance)*)>
3  <!ELEMENT nom (#PCDATA) >
4  <!ELEMENT adresse (ville, rue, (numéro)?)>
5  <!ELEMENT séance EMPTY>
6  <!ATTLIST séance heure NMTOKEN #REQUIRED
7      ref_film IDREF #REQUIRED>
8  <!ELEMENT film (titre, année>
9  <!ATTLIST film film_id ID #REQUIRED>
10      acteurs IDREFS #IMPLIED>
11 <!ELEMENT titre (#PCDATA) >
```

DTD: Utilisation

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: Pourquoi Validation?

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

ELEMENT : Déclaration du Type de 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 : $(\#PCDATA \mid A \mid B \dots)^*$

Expressions régulières

Expressions régulières sur un alphabet N :

- Chaque symbol $n \in N$ est une expression régulière (e.r.);
- Si e est une expression régulière, alors $(e)^*$, $(e)^+$ et $(e)^?$ sont des expressions régulières;
- Si e_1 et e_2 sont des expressions régulières, alors (e_1, e_2) et $(e_1|e_2)$ sont des expressions régulières;

Langages réguliers (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 $L(e_1)$ et dans $L(e_2)$ (union)

Langages réguliers (2)

- $L(e+) = \{m_0m_1 \dots m_n \mid m_i \in L(e)\}$: tous les mots composés de mots dans $L(e)$;
- $L(e*) = L(e+) \cup \{\epsilon\}$;

Exemples :

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

Exemples

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+)>*

ATTLIST : Déclaration des Attributs

<!ATTLIST élément nom type mode [default]>

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>
```

Types d'attributs

- 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

ATTLIST : Mode

Modes d'attributs:

- #REQUIRED : la valeur doit être définie
- #IMPLIED : la valeur est optionnelle
- #FIXED : la valeur est constante

```
<!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'>
```

Entités

```
<!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.

Entités externes

- Segmentation du document en plusieurs sous-documents
- Réutilisation de DTDs et de déclarations
- Références vers données non-XML (NOTATION)

Adressage:

URL: `<!ENTITY % autre SYSTEM 'http://pariscope.fr/ext.xml' >`

FPI (formal public identifier):

`<!ENTITY % autre PUBLIC '-//CNAM//Texte libre//FR'>`

Résumé sur les DTDs

- 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