XML : Extensible Markup Language

Olivier.Corby@sophia.inria.fr

http://www/inria.fr/acacia

Java

program once, *run everywhere*

XML

write once, *publish everywhere*

Plan

- Introduction
- XML
  - DTD (Document Type Definition)
  - XML Schema
- Liens et chemins
  - XPath : XML Path Language
  - XPointer : XML Pointer Language
  - XLink : XML Linking Language
- Processeurs
  - Parser : DOM vs SAX
  - XSLT : Extensible Stylesheet Language Transformation
  - XQuery : XML Query Language
  - DOM : Document Object Model
- Semantic Web
  - RDF : Resource Description Language
  - RDFS : RDF Schema
  - OWL : Ontology Web Language
  - Web Services

XML : Extensible Markup Language

http://www.w3.org/TR/xml

- Un format d'échange de documents et données structurés
- Un ensemble de langages complémentaires : liens, chemins, style, etc.
- Des outils logiciels
- Conçu pour le Web

Une boite à outils pour traiter les documents et les données structurés

XML et WWW

- XML élaboré par la communauté SGML: été 96
- Spécifié au sein du World Wide Web Consortium : W3C
- Recommandation du W3C, février 1998
- Nouvelle recommandation d'octobre 2000, pour corriger des erreurs
- XML 1.1 Février 2004
- http://www.w3.org/XML
- (Quasiment) toutes les recommandations du W3C sont faites en XML

XML, SGML & HTML

- Version simplifiée de SGML, *SGML Light pour le Web*
- SGML: Standard Generalized Markup Language, 1986
- Issu lui même de travaux d'IBM : Generalized Markup Language
- HTML, élaboré selon la norme SGML : 1992

XML : un métalangage

- **Générateur** de formats d'échange de
  - données structurées
  - documents structurés
- Séparer :
  - données structurées
  - présentation
- Un document XML contient la définition de sa structure
- Repose sur la modélisation des caractères Unicode
- Peut être lu par tout parser XML

Exemple

```
<slide>
<title>XML : un métalangage</title>
<ul>
        <lip>Générateur de formats d'échange de</lip>
```

```
        <ul>
                <lip>données structurées</lip>
                <lip>documents structurés</lip>
        </ul>
        <lip>Séparer : </lip>
        <ul>
                <lip>données structurées</lip>
                <lip>présentation</lip>
        </ul>
        <lip>Un document XML contient la définition de sa structure</lip>
        <lip>Repose sur la modélisation des caractères Unicode</lip>
        <lip>Peut être lu par tout parser XML</lip>
</ul>
</slide>
```

Exemple

```
<book>
<head>
        <title>The General Theory of Employment, Interest and Money</title>
        <author>J. M. Keynes</author>
</head>
<chapter>
        <title>Employment</title>
        <section>...</section>
</chapter>
<chapter>
        ...
</chapter>
</book>
```

Exemple

```
<defclass name="Car">
        <defattribute name="mark" type="string"/>
        <defattribute name="age" type="integer"/>
</defclass>

<defobject id='123'>
        <class>Car</class>
        <attribute name='mark'>Renault</attribute>
        <attribute name='age'>1992</attribute>
</defobject>
```

Exemple

```
<lettre>
        <date>17/10/2000</date>
        <objet>Réclamation</objet>
        <politesse>Madame</politesse>
        <para>En réponse à votre lettre du <date>15/12/1998</date>
```

```
           dans laquelle Monsieur <name>Ybroc</name> ...
        </para>
        <salutation/>
</lettre>
```

Exemple

```
<bibliography>

<book id='LCCN-36027176'>

<author>
<firstname>John Maynard</firstname>
<lastname>Keynes</lastname>
</author>

<title>The General Theory of Employment, Interest and Money</title>
<publisher>MacMillan</publisher>
<year>1936</year>
</book>

</bibliography>
```

Structure vs Contenu

Un document XML inclu la définition de sa structure sous forme d'arbre.

```
<bibliography>
<book id='LCCN-36027176'>
        <author><firstname>John Maynard</firstname>
                <lastname>Keynes</lastname>
        </author>
        <title>The General Theory of Employment, Interest and Money</title>
        <publisher>MacMillan</publisher>
        <year>1936</year>
</book>
</bibliography>
```

```
bibliography
   \n
   book id='LCCN-36027176'
      \n
      author
         firstname
            "John Maynard"
         \n
         lastname
            "Keynes"
         \n
      \n
      title
         "The General Theory of Employment, Interest and Money"
```

```
    \n
    publisher
       "MacMillan"
    \n
    year
       "1936"
    \n
 \n
```

XML vs HTML

- Définir ses propres balises
- Pas de limitation d'imbrication des entités XML
- Validation possible par rapport à un Schema ou une DTD
- Uniquement la structure du document, pas de format de présentation
- Amélioration du système de liens hypertextes

Avantages

- Réutiliser des informations
- Publier indépendamment des medias
- Imprimé et numérique à partir de la même source:
  - write once, publish everywhere
- Indépendant de formats propriétaires
- Indépendance par rapport aux vendeurs et aux plates-formes
- XML + XSL peut remplacer formats de traitement de texte et logiciel de publication
- Standard pour représenter les données: on peut changer d'outils

Avantages (suite)

- Améliorer la recherche d'information sur le Web:
  - recherche sur des balises et des schemas/DTD spécifiques
  - recherche sur le Web comme dans une base de données
- Une communauté peut définir un standard d'échange d'information:
  - se mettre d'accord sur un Schema
  - échanger des données XML, sans le Schema

Types d'Application

- Client Web médiateur de plusieurs sources hétérogènes
- Décharger le serveur, charger le client d'une partie du traitement:
  - présentation des données
  - calculs, transformations, etc.

- Présenter différentes vues des mêmes données, sans les recharger à chaque fois
- Adapter le format de présentation à l'utilisateur

La galaxie XML

- **XML, DTD, XML namespace**
- **XML Schema** : décrire la structure d'un document et le typage (remplace DTD)
- **XPath, XPointer** : langages de chemins pour naviguer dans les documents
- **XLink** : liens hypertextes
- **RDF** : Resource Description Framework, méta informations pour annoter des documents (Semantic Web)
- **RDF Schema** : Schema pour RDF

Traitement de XML

- **XML Events** : modèle d'évènements pour XML
- **DOM** : Document Object Model
  - API d'accès aux documents et aux données XML
- **SAX** : Simple API for XML: API dirigée par les événements (hors W3C)
- **XSL** , Extensible Stylesheet Language: feuilles de style
  - **XSLT** : XSL Transformation
  - **FO** : Formatting object
- **XQuery** : Langage d'interrogation pour XML

XML

XForms XLink
XML Schema XPointer XSLT XQuery RDF Schema
XPath RDF XHTML SMIL MathML
XML, DTD, Namespace

Formalismes XML

- XHTML (26 janvier 2000)
- MathML (W3C)
- SMIL: Synchronized Multimedia Integration Language (W3C)
- XMI: Metadata Interchange Format (OMG)
- XML/EDI: Electronic Data Interchange, Commerce électronique basé sur UN/EDIFACT
- TIM: Telecom Interchange Markup (ATIS)
- CML: Chemical Markup Language
- ThML: Theological Markup Language

Document XML

- Document bien formé: well formed. (syntaxe)
  - balises correctement emboîtées, pas de '<' dans le texte, etc.
- Document valide: vérifie des contraintes supplémentaires
  - avoir un schema/DTD et s'y conformer
- Document composé de:
  - Texte: Parsed Character data (PCDATA)
  - Balises: Markup

DTD: Document Type Definition

- DTD: description formelle de la structure du document
- Locale, importée ou les deux
- DTD contient:
  - element type
  - attribute list
  - entity
  - notation
- DTD est facultative
- Un document qui déclare une DTD doit s'y conformer

DTD

Document XML :

```
<?xml version="1.0"?>

<!DOCTYPE bibliography SYSTEM "http://somewhere.net/book.dtd">

<bibliography>
<book>
        <author>
                <firstname>John Maynard</firstname>
                <lastname>Keynes</lastname>
        </author>
        <title>The General Theory of Employment, Interest and Money</title>
        <publisher>Macmillan</publisher>
</book>
</bibliography>
```

DTD :

```
<!ELEMENT bibliography (book*)>

<!ELEMENT book (author, title, publisher, year?)>

<!ELEMENT author (firstname, lastname)>
```

```
<!ELEMENT title (#PCDATA)>

<!ELEMENT publisher (#PCDATA)>
```

Element

```
<!ELEMENT text ((para|figure)+)>

<!ELEMENT nihil EMPTY>

   <nihil/>

<!ELEMENT cool ANY>
```

Attribute list

```
<!ATTLIST book id ID #REQUIRED>

        <book id="V0-98/04">...</book>



<!ELEMENT reference(#PCDATA)>

<!ATTLIST reference value IDREF #IMPLIED>

       <reference value="V0-98/04">
          Un livre passionnant ...
       </reference>
```

Attribute list (2)

```
<!ATTLIST title
       form (bold|italic|normal) "normal" #IMPLIED>

        <title form="italic">A Theory of Justice</title>


<!ATTLIST author name CDATA #REQUIRED>

       <author name="Jean-Sol Partre"/>
```

Entity

- Entité définie, à laquelle est associé un nom

- parsed entity (XML)
- unparsed entity (image, etc.) : référencée par son nom dans un attribut de type ENTITY
- General parsed entity:

```
<!ENTITY cpr "Copyright INRIA 1998">

<text>du texte libre ... &cpr; </text>

<text>du texte libre ... Copyright INRIA 1998 </text>
```

Entity externe

Pour intégrer un document XML dans un autre :

```
<!DOCTYPE test [
   <!ENTITY include SYSTEM "file.xml">
]>
<test>
   &include;
</test>
```

Entity de DTD

- Parameter parsed entity: utilisée dans la DTD pour factoriser/customiser des portions de documents

Définir les deux attributs name et id :

```
<!ENTITY % attr-list "name CDATA #IMPLIED id IDREF #REQUIRED">
```

Utiliser cette définition dans l'élément Test :

```
<!ELEMENT Test (#PCDATA)>
<!ATTLIST Test %attr-list; >
```

Equivaut à :

```
<!ATTLIST Test name CDATA #IMPLIED id IDREF #REQUIRED>
```

Unparsed entity

- Pour intégrer des images, etc:

```
<!ENTITY icon SYSTEM "../icons/test.gif" NDATA gif>


<!ELEMENT figure EMPTY>
<!ATTLIST figure image ENTITY #REQUIRED>


<figure image="icon"/>
```

Notation

- Notation: pour déclarer un processeur pour des éléments non XML: image, audio, etc.

```
<!NOTATION gif SYSTEM "gifprocessor">
```

Processing instruction

Directive de traitement au processeur XML : navigateur, parser, etc.

```
<?xml-stylesheet type='text/xsl' href='talk.xsl'?>
```

Autres

- Commentaires : restitué dans l'arbre XML

```
<!-- comment -->
```

- CDATA : considérer comme du texte une portion de document contenant des balises :

```
<![CDATA[
      ici du texte avec des balises est reconnu comme du texte,
      exemple: ici <a> n'est pas une balise
]]>
```

Namespace

- Modulariser et réutiliser
- Utiliser des balises provenant de plusieurs DTD et éviter des conflits de nom
- Précéder les balises d'un identifiant unique, spécifique de la DTD : un URI
- Namespace : l'URI. e.g. : `http://www.inria.fr/acacia/corby`
- prefix : un nom local auquel est associé un namespace. e.g. `<s:title>`
- Le namespace est déclaré avec un attribut spécial `xmlns:s`

```
xmlns:s='http://www.inria.fr/acacia/corby'
```

Exemple namespace

```
<s:book
        xmlns:s='http://www.inria.fr/acacia/corby'

        xmlns:html='http://www.w3.org/TR/REC-html40'>

    <s:title>XML</s:title>

    <s:author>O. Corby</s:author>

    <html:p> ...</html:p>

</s:book>
```

Structure du document

Il y a 7 types de noeud dans un arbre XML

- élément : balise
- attribut
- namespace
- texte
- commentaire
- processing instruction
- entité

Le parser XML intègre au document

- tout le balisage
- tout le texte, y compris les fins de lignes sous forme d'éléments text

Eléments de méthodologie

- Utiliser les balises et les attributs pour structurer le document
- Les attributs de sont pas structurés : ils ne contiennent pas de balise
- Valeurs d'attributs entre " " ou entre ' '
- Ne pas définir de système de balisage ad hoc :

Eviter :

```
<name> [Olivier] [Corby] </name>
```

Préférer :

```
<firstname>Olivier</firstname>

<lastname>Corby</lastname>
```

Elément vs attribut ?

Préférer les éléments, plutôt que les attributs, pour exprimer le contenu du document

A ceci :

```
<book   author="J. M. Keynes"

        title="The General Theory of Employment, Interest and Money"/>
```

Préférer ceci :

```
<book>
        <author>J. M. Keynes</author>

        <title>The General Theory of Employment, Interest and Money</title>
</book>
```

Elément vs attribut ?

Utiliser les attributs pour :

Exprimer des relations entre éléments (e.g. référence, lien hypertexte)

```
<author id="oc">
        <firstname>Olivier</firstname>
        <lastname>Corby</lastname>
</author>

<paper ref="oc"/>
```

Exprimer des données *codées*

```
<country code="fr">France</country>
```

Compléter une valeur d'élément avec une unité, etc.

```
<length unit='m'>2.718</length>

<amount currency='euro'>3.14</amount>
```

Elément vs attribut ?

- Faire des choix uniformes dans un document
- Utiliser le même type de codage pour le même type d'information

XML Schema

Recommandation du W3C : Mai 2001

Un langage pour définir la structure et le typage des documents

Une syntaxe XML

- Destiné à remplacer la DTD
- Introduit des types simples : integer, float, string, énumération, date, etc.
- Définition de type par extension de types existant
- Une touche *objet* pour le typage : spécialisation

Recommandation

- XML Schema Part 0 : Primer
- XML Schema Part 1 : Structure
- XML Schema Part 2 : Datatypes

Type complexe

Déclaration d'un élément book :

```
<xsd:schema xmls:xsd="http://www.w3.org/2001/XMLSchema"
        xmls:s='http://www.inria.fr/acacia'>

        <xsd:element name="book" type="s:bookType"/>
```

<!-- Déclaration de son type bookType-->

```
        <xsd:complexType name='bookType'>
                <xsd:sequence>
                        <xsd:element name='author'      type='s:author'/>
                        <xsd:element name='title'       type='string'/>
                        <xsd:element name='publisher'   type='string'/>
                        <xsd:element name='year'        type='integer'/>
                </xsd:sequence>

                <xsd:attribute name='id'                type='string'/>
        </xsd:complexType>
<xsd:schema>
```

Elément book dans un document

```
<s:book id='...'>
        <s:author>
                <s:name>J. M. Keynes</s:name>
        </s:author>
        <s:title>General Theory ...</s:title>
        <s:publisher>...</s:publisher>
        <s:year>...</s:year>
<s:book>
```

Type complexe avec choix

```
<xsd:complexType name='bookType'>
    <xsd:sequence>
        <xsd:element name='author'      type='s:author'/>
        <xsd:element name='title'       type='string'/>
        <xsd:element name='publisher'   type='string'/>

        <xsd:choice>
                <xsd:element name='year'        type='gYear'/>
                <xsd:element name='date'        type='date'/>
        </xsd:choice>

    </xsd:sequence>

    <xsd:attribute name='id'            type='string'/>
</xsd:complexType>
```

Déclaration sans ordre

Tous les éléments doivent apparaitre zéro ou une fois, dans n'importe quel ordre

```
<xsd:complexType name="bookType">
   <xsd:all>
        <xsd:element name='author'      type='s:author'/>
        <xsd:element name='title'       type='string'/>
        <xsd:element name='publisher'   type='string'/>
        <xsd:element name='year'        type='gYear'/>
   </xsd:all>
   <xsd:attribute name='id'             type='string'/>
 </xsd:complexType>
```

Facettes d'occurrence

Pour un élément

```
<xsd:element name="date" type="integer"
        minOccurs="0" maxOccurs="1"
        fixed="2000" default="2000"/>
```

Pour un attribut

```
<xsd:attribute name="price" type="integer"
        use="required | optional | fixed | default"
        value="123"/>
```

Types simples

- string
- integer
- decimal
- float
- boolean
- time : 13:20:00.000
- timeInstant : 1999-05-31T13:20:00.000-05:00
- date
- Name
- QName
- ID IDREF
- NMTOKEN
- ENTITY
- ...

Type simple étendu

Spécification d'un motif à l'aide d'expression régulière

Date sous la forme : 16-10-1959

```
<xsd:simpleType name="Date">
   <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d{2}-\d{2}-\d{4}"/>
   </xsd:restriction>
</xsd:simpleType>
```

Code sous la forme : Catch-22

```
<xsd:simpleType name="code">
   <xsd:restriction base="xsd:string">
        <xsd:pattern value="[A-Za-z]{5}-\d{2}"/>
   </xsd:restriction>
</xsd:simpleType>
```

Type énuméré

```
<xsd:simpleType name="Region">
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="Alsace"/>
     <xsd:enumeration value="Centre"/>
     <xsd:enumeration value="Languedoc-Roussillon"/>
     <xsd:enumeration value="PACA"/>
     <!-- etc ... -->
   </xsd:restriction>
</xsd:simpleType>
```

Type liste

```
<xsd:simpleType name="ListeRegion">
  <xsd:list itemType="Region"/>
</xsd:simpleType>

<xsd:element name="regions" type="ListeRegion"/>
```

Elément de type liste de régions :

```
<regions>Alsace PACA</regions>
```

Type simple avec attribut

Déclaration d'un élément de type simple avec attribut

```
 <xsd:element name="price">
   <xsd:complexType>
    <xsd:simpleContent>
     <xsd:extension base="xsd:decimal">
      <xsd:attribute name="currency" type="xsd:string" />
     </xsd:extension>
    </xsd:simpleContent>
   </xsd:complexType>
 </xsd:element>
```

Elément

```
<price currency="EUR">423.46</>
```

Type dérivé par extension

Un type peut être dérivé d'un autre par extension

Ici on ajoute un élément URI à un type bookType

```
<xsd:complexType name='eBookType'>
    <xsd:complexContent>
        <xsd:extension base='bookType'>
                <xsd:sequence>
                        <xsd:element name='URI' type='uriReference'/>
                </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Utiliser un type dérivé dans un document

Déclaration d'un élément item de type bookType

```
<xsd:element name="item" type="bookType"/>
```

Elément item *standard* (i.e. de type bookType):

```
<s:item>
        <s:author> ...</s:author>
        ...
        <s:date>..</s:date>
</s:item>
```

Elément item de type eBookType :

```
<s:item xsi:type="eBookType">
        <s:author> ...</s:author>
        ...
        <s:date>..</s:date>
        <s:URI> ...</s:URI>
</s:item>
```

Si l'on utilise un type dérivé dans le document, à la place du type de base, il faut le signaler.

Contrainte d'intégrité : unicité

La valeur de l'attribut code des éléments regions/zip est unique

```
<xsd:unique name="dummy1">
       <xsd:selector xpath='regions/zip'/>
       <xsd:field xpath='@code'/>
```

```
</xsd:unique>
```

Contrainte d'intégrité : clé et référence

Toute clé référencée doit être définie. Exemple : référence bibliographique

```
<para>....<bibref code="Corby1999"/> ...</para>

<bibliography>
        <bibitem code="Corby1999">
                <author>Olivier Corby</author>
                ...
        </bibitem>
</biliograpgy>
```

Définition de la clé `bibitem/@code` et de la référence `bibref/@code`

```
<xsd:key name="bib">
        <xsd:selector xpath='bibitem'/>
        <xsd:field xpath='@code'/>
</xsd:key>

<xsd:keyref name="dummy" refer="bib">
        <xsd:selector xpath='bibref'/>
        <xsd:field xpath='@code'/>
</xsd:keyref>
```

Autres

Equivalence entre éléments :

```
<xsd:element name="comment"     type="string" />

<xsd:element name="shipComment" type="string"
                substitutionGroup="ipo:comment" />

<xsd:element name="customerComment" type="string"
                substitutionGroup="ipo:comment" />
```

Groupe d'éléments

```
<xsd:group name="shipAndBill">
        <xsd:sequence>
                <xsd:element name="shipTo" type="USAddress" />
```

```
                <xsd:element name="billTo" type="USAddress" />
        </xsd:sequence>
</xsd:group>

<xsd:group   ref="shipAndBill" />
```

Groupe d'attributs

```
<xsd:attributeGroup name="ItemDelivery">
              <xsd:attribute name="partNum" type="SKU"/>
              <xsd:attribute name="weightKg"  type="xsd:decimal"/>
</xsd:attributeGroup>

<xsd:attributeGroup ref="ItemDelivery"/>
```

Autres

Union type

```
<xsd:simpleType name="zipUnion">
     <xsd:union memberTypes="USState listOfMyIntType"/>
</xsd:simpleType>
```

Union of different kinds of types :

```
<xsd:simpleType>
    <xsd:union>
        <xsd:simpleType>
             <xsd:restriction base='nonNegativeInteger'/>
        </xsd:simpleType>
        <xsd:simpleType>
             <xsd:restriction base='string'>
                  <xsd:enumeration value='unbounded'/>
             </xsd:restriction>
        </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>
```

Autres

Mixed content :

```
<xsd:element name="letterBody">
      <xsd:complexType mixed="true">
          <xsd:element .../>
      </xsd:complexType>
```

```
</xsd:element>
```

Any type :

```
<xsd:element name='anything' type='xsd:anyType'>
<xsd:element name='anything'/>
```

Empty content type:

```
<xsd:element name='vacuum'>
        <xsd:complexType>
                <xsd:attribute .../>
        </xsd:complexType>
</xsd:element>
```

null content :

```
<xsd:element name="shipDate" type="xsd:date" nillable="true"/>

<shipDate xsi:nil="true"></shipDate>
```

Autres

Abstract : ne peut être utilisé dans un document :

```
<xsd:element name="comment" type="string" abstract="true"/>

<xsd:complexType name="Vehicle" abstract="true"/>
```

Limiter la dérivation de types :

```
<xsd:complexType name="Address" final="restriction|extension|#all">
```

Limiter l'utilisation d'un type dérivé à la place d'un type de base dans un document :

```
<xsd:complexType name="Address" block='restriction|extension|#all'>
```

Autres

targetNamespace

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace='http://www.inria.fr/acacia'>
...
```

```
</schema>
```

Référence au schema dans un document

```
<s:document xmls:xsi='http://www.w3.org/2001/XMLSchema-instance'
        xsi:schemaLocation='http://www.inria.fr/acacia
                            http://www.inria.fr/acacia/pub/schema.xsd'
        xmls:s='http://www.inria.fr/acacia'>

....

</s:document>
```

documentation

```
<xsd:annotation>
    <xsd:documentation>XML Schema is not de la tarte</xsd:documentation>
</xsd:annotation>
```

Autres

```
<xsd:any namespace='a namespace | ##targetNamespace |
                        ##any | ##local | ##other'
        processContents='skip | lax | strict'/>

<xsd:anyAttribute namespace='...'/>

length, minLength, maxLength

totalDigits
fractionDigits

prohibited

<xsd:include schemaLocation='http://...' />

<xsd:redefine schemaLocation='http://...'/>

elementFormDefault='qualified' for local elements only
attributeFormDefault='qualified' for local attributes only
form='qualified'
blockDefault='#all'


global definition and ref=''

multiple local definitions of element with same name and different types
```

Outils

Browser XML

- Documents structurés:
  - définir des feuilles de styles et le navigateur présente le document: CSS, XSL
- Objets structurés:
  - Ecrire des applets, par exemple en Java, qui font les traitements appropriés.

Outils

- Parser XML : JAXP (Sun), XML parser for Java (IBM), Xerces (www.apache.org), XP de James Clark
  - Analyse un document XML et construit un arbre
  - Document accessible via une API:
    - DOM, Document Object Model (W3C),
    - SAX: Simple API for XML (pas de construction d'arbre)
- XSL:
  - Intégré dans Java 1.4 !, Xalan (Apache), XT de James Clark : http://www.jclark.com
  - Lotus

Outils (2)

- Mozilla, Internet Explorer, msxml
- Document : ArborText, InterLeaf, etc.
- Base de données, Serveur: Xylème, Object Design eXcelon, Oracle8i, Euroclid XML Server (O2, Verity), BlueStone XML Server, etc.

Processeur XML

- DOM : W3C Document Object Model
- SAX : Simple API for XML

DOM : Document Object Model

- Recommandation W3C
- Modèle abstrait de manipulation de document XML
- Interface de programmation (API) permettant d'accéder à la structure et au contenu d'un document XML
- Définit une API en Java, ECMAScript et IDL
- Programmer une application avec l'API
- Parser ou navigateur XML supporte l'API DOM

- Modèle d’événements, parcours d’arbre XML, range

DOM

Interfaces pour :

- Node, Document, Element, Attribute,
- ProcessingInstruction, Entity, Notation, Comment, Text, CDATASection, etc.

Node

Interface racine des interfaces DOM

```
interface Node {ATTRIBUTE_NODE

        COMMENT_NODE

        DOCUMENT_NODE

        DOCUMENT_TYPE_NODE

        ELEMENT_NODE

        ENTITY_NODE

        NOTATION_NODE

        PROCESSING_INSTRUCTION_NODE

        TEXT_NODE

}
```

Node

```
interface Node {

        NodeList getChildNodes();

        NamedNodeMap getAttributes();

        String getNodeName();

        String getNodeValue();

        short getNodeType();
}
```

Document

Modélise le document XML

```
interface Document : Node {

        Element createElement(in DOMString tagName)
                raises(DOMException);

        NodeList getElementsByTagName(in DOMString tagname);
}
```

Element

Les noeuds de l'arbre de type élément (balise)

```
interface Element : Node {

        DOMString getAttribute(in DOMString name);

        void setAttribute(in DOMString name, in DOMString value)
        raises(DOMException);

}
```

Hérite `getChildNodes()` et `getAttributes()` de l'interface Node

Attribute

```
interface Attr : Node {

        String getName();
        String getValue();

}
```

DOM : Java API

Parser un document XML en Java

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

public class DOMParser {
```

```
    public static void main (String args[]) throws Exception  {

        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance()

        DocumentBuilder parser=factory.newDocumentBuilder();

        Document doc=parser.parse(args[0]);

    }

}
```

DOM : Exemple de parcours d'arbre

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

public class DOMParser {

    public static void main (String args[]) throws Exception  {
        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance()
        DocumentBuilder parser=factory.newDocumentBuilder();
        Document doc=parser.parse(args[0]);

        Node root=doc.getDocumentElement();

        new DOMParser().walk(root);
    }
```

Parcours récursif d'un arbre XML DOM

```
void walk(Node root){
        Node node;
        System.out.println(root.getNodeName() + " " + root.getNodeValue());
        NodeList list=root.getChildNodes();
        for (int i=0; i<list.getLength(); i++){
            node=list.item(i);
            walk(node);
        }
     }
}
```

SAX : Simple API for XML

Interface de programmation définie par la communauté XML, supportée par les parsers XML

Traitement dirigé par les événements (event driven)

Réalisé sous forme de callbacks appelées par le parser, traitées par un Handler

L'utilisateur crée son propre handler qui réalise les traitements appropriés :

```
startDocument

endDocument

startElement

endElement

characters

processingInstruction
```

SAX : Simple API for XML

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import javax.xml.parsers.*;

public class SaxParse {

public static void main (String args[]) throws Exception {

    SAXParserFactory factory=SAXParserFactory.newInstance();

    SAXParser parser=factory.newSAXParser();

    DefaultHandler handler = new MyHandler();

    try {
       parser.parse(args[0], handler);
    }
    catch (SAXException e){
        System.out.println(e.getMessage());
    }
}

}
```

SAX : Handler

```
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

public class MyHandler extends DefaultHandler {

    public void startDocument (){

    }

    public void startElement(String namespaceURI, String simpleName,
                             String qualifiedName, Attributes atts){
        atts.getValue("name");
```

```
        for(int i=0; i<atts.getLength(); i++){
            atts.getValue(i);
        }
    }

    public void characters (char buf [], int offset, int len){
        String s=new String(buf, offset, len);

    }

    public void endElement (String namespaceURI, String simpleName,
                            String qualifiedName){
    }


    public void endDocument (){

    }
}
```

Web sémantique

The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. Tim Berners-Lee

- Une extension du Web actuel
- L'information disponible est accompagnée d'une signification bien définie
- Permettre le traitement de l'information par des ordinateurs

Données et informations

- Le Web actuel distribue des documents à lire
- Le Web sémantique distribue des données, des informations que des programmes peuvent traiter
- Exemple : la préparation d'un voyage pourrait être automatisée

RDF : Resource Description Framework

- Pour exprimer des méta informations sémantiques sur le contenu des documents (pages Web, etc.)
- Aide pour de meilleurs moteurs de recherche
- Catalogage pour décrire le contenu
- Aider des agents intelligents pour faciliter le partage et l'échange de connaissances
- Vers un Web sémantique

Web Service

The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as Web services.

- Communiquer des informations entre programmes
- Echanger des données XML via une interface
- Indépendance vis à vis des plate-formes, des systèmes et des langages

SOAP

- Simple Object Access Protocol
- Protocole de type RPC avec interface et corps.
- Permet l'échange de données typées au format XML

WSDL

- Web Service Description Language
- Décrit des services de manière abstraite, sans se référer à une implémentation
- messages : description abstraite des données échangées
- port types : collection abstraite d'opérations

X Files : Sites de référence

- W3C: World Wide Web Consortium
  http://www.w3.org
- OASIS: Organization for the Advancement of Structured Information Standards
  http://www.oasis-open.org
- http://www.xml.org/ Site XML d'OASIS
- http://xmlfr.org/ XML francophone
- Semantic Web
  http://www.semanticweb.org
- Apache Software Foundation
  http://www.apache.org
- Cours XML
  http://www.inria.fr/acacia/cours/essi