2. Schémas XML

Ouvrages recommandés :

- XML in a nutshell – S. Means & E.R. Harold - Edition O'Reilly (Bibliothèque UMLV)
- XML Schéma – E. Van Der Vlist – Edition O'Reilly (Bibliothèque UMLV).

Introduction

- Comment échanger des données d'un langage XML à un autre.
  - Il faut décrire formellement les deux langages : des schémas
  - Réaliser une transformation en utilisant les schémas.
- Plusieurs solutions pour définir des schémas :
  - DTD (Document Type Definition)
  - XML Schema
  - RelaxNG

Document valide

- Un document '**valide**' est un document qui respecte la grammaire définie dans un schéma.
- Différent de 'bien formé', condition nécessaire pour un document XML, qui indique que le document respecte les règles standards d'écriture d'un doc. XML.
- Document 'valide' <=> structure OK
- Document 'bien formé' <=> syntaxe OK

DTD

-
- Structure le document
  - DTD=Classe / Document=instance
  - Contraint la syntaxe
  - Exemple:
    - Livre = Titre + Contenu
    - Titre = TEXT
    - Contenu = Chapitre(s) + References
    - Chapitre = Section(s) ...

DTD - élément

- Déclaration des **éléments**: <!ELEMENT ... >
  - <!ELEMENT UL (LI)+>
  - Déclaration implicite de leur imbrication => structure d'arbre
  - Syntaxe
    - X* : 0 ou plus occurrence(s) de X
    - X+ : 1 ou plus occurrence(s) de X
    - X? : X est présent une fois au plus
    - -X : pas d'occurences de X
    - X¦Y : X OU Y
    - X,Y : X ET Y (dans l'ordre)
    - X&Y : X ET Y (pas forcément dans l'ordre)
    - EMPTY, ANY : Vide ou quelconque
    - #PCDATA : contient du texte

DTD – élément (2)

- On peut définir des modèles de contenu autorisant le mélange de données et éléments.
- Syntaxe :

```
(#PCDATA | nomElement1 | ... | nomElementn)
```

- Exemple d'une déclaration

```
<!ELEMENT p (#PCDATA | em | exposant | indice)*>

<!ELEMENT em (#PCDATA |exposant|indice)*>

<!ELEMENT exposant (#PCDATA)>

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

- Exemple d'utilisation

```
<p> un paragraphe contenant du texte <em> mis en
  evidence</em> ou <exposant> exposant </exposant></p>
```

DTD - attribut

- Déclaration des **attributs**: <!ATTLIST ... >
  - <!ATTLIST TD
    valign (top|middle|bottom|baseline) #IMPLIED
    ... >
  - Déclaration de leur nécessité (syntaxe contrainte)
  - CDATA : caractères
  - ID : Lettres uniquement, et unique sur le document
  - IDREF: Reference sur ID
  - IDREFS : Plusieurs valeurs de ID
  - ENTITY, ENTITIES
  - NMTOKEN, NMTOKENS (utilisation avancée)

DTD – attribut (2)

- Déclaration des **attributs**: <!ATTLIST ... >
  - <!ATTLIST TD
    valign (top|middle|bottom|baseline) #XXX >
  - 4 valeurs possibles pour XXX :
    - La valeur par défaut de l'attribut
    - REQUIRED et chaque élément instance devra posséder cet attribut.
    - IMPLIED et la présence de cet attribut est facultative.
    - FIXED : fixe la valeur de cet attribut pour tout élément instance. Il n'est pas nécessaire de répéter cet attribut.

Exemple avec DTD

```
 <?xml version="1.0"?>
<!DOCTYPE famille SYSTEM "ele1.dtd">
<famille>
   <personne>
     <nom>Durand</nom>
     <prenom>Jean</prenom>
     <age>28 ans</age>
   </personne>
   <personne>
     <nom>Durand</nom>
     <prenom>Marie</prenom>
     <age>20 ans</age>
   </personne>
</famille>
```

Et sa DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT famille (personne+)>
<!ELEMENT personne (nom, prenom,age)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT age (#PCDATA)>
```

Le nom du fichier est ele1.dtd

Un autre exemple

```
<?xml version="1.0"?>
<!DOCTYPE famille SYSTEM "att1.dtd">
<famille>
<personne
   nom="Durand"
  prenom="Jean"
  age = "28 ans"/>
<personne
   nom="Cure"
  prenom="Marie"
  age = "20 ans"/>
</famille>
```

Et sa DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT famille (personne+)>
<!ELEMENT personne EMPTY>
<!ATTLIST personne
   nom CDATA #REQUIRED
   prenom CDATA #REQUIRED
   age CDATA #REQUIRED>
```

Le nom du fichier est att1.dtd

DTD - Doctype

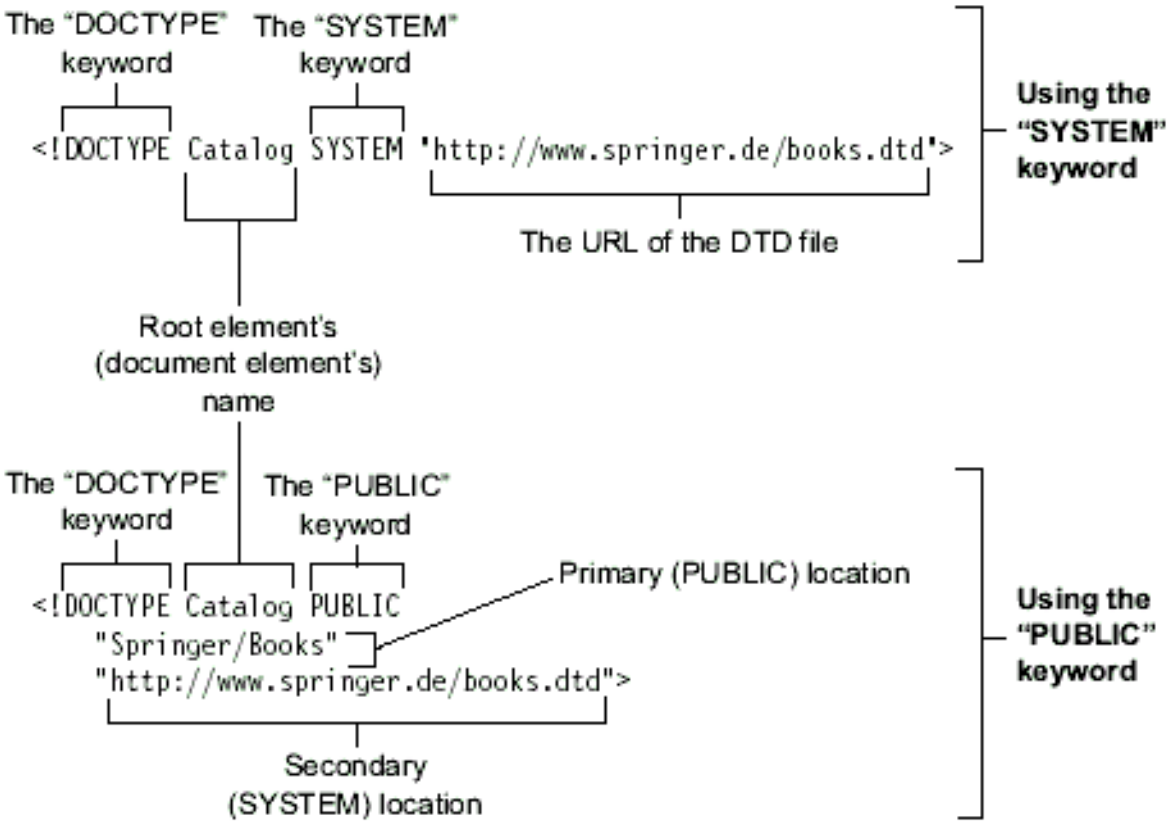
- Une PI (ou Processing Instruction) qui s'insère dans le prologue d'un document XML dans le but d'associer une DTD au document.
- Possibilité de déclarer la DTD en interne du document ou en externe.
- En externe, il existe 2 déclarations différentes :
  - Avec le mot réservé "SYSTEM"
  - Avec le mot réservé "PUBLIC"

DTD – Interne au document

```
<?xml version="1.0"?>
<!DOCTYPE articles [
    <!ELEMENT articles (article+)>
    <!ELEMENT article (nom)>
    <!ELEMENT nom (#PCDATA)>
    ]>
    <articles>
        <article>
            <nom>X1090AB</nom>
        </article>
        <article>
                <nom>Y9968AZ</nom>
        </article>
    </articles>
```

Le nom de l'élément racine du document

DTD – Externe au document



DTD – externe avec SYSTEM

- Syntaxe avec localisation absolue:

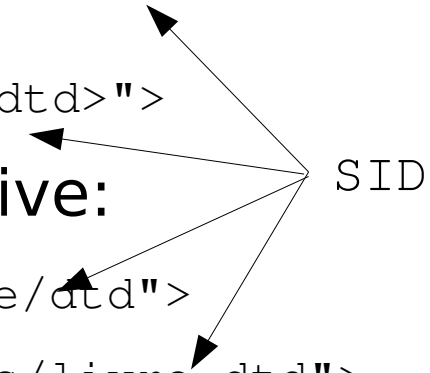
```
<!DOCTYPE livre SYSTEM "/home/user1/dtds/livre.dtd">

<!DOCTYPE livre SYSTEM "<
  http://www.toto.com/dtds/livre.dtd>">
```

- Syntaxe avec localisation relative:

```
<!DOCTYPE livre SYSTEM "dtds/livre/dtd">

<!DOCTYPE livre SYSTEM "../../dtds/livre.dtd">
```



- Un SID (System Identifier) est une URI permettant d'accéder à une DTD.

DTD – externe avec PUBLIC

- Méthode rarement utilisée, elle permet d'accéder via un identifiant officiellement déclaré à une DTD publique. Il est nécessaire de déclarer un tel identifiant auprès d'organismes : ISO, GCA (American Graphic Communication Association).
- La définition prévoit de définir un second identifiant qui correspond à un identifiant SYSTEM.

DTD – externe et interne

- On peut mélanger les sources (externe et interne) de DTD. Lecture de DTD interne avant l'externe.
- Exemple :

```
<?xml version="1.0">
<!DOCTYPE famille SYSTEM "dtds/famille.dtd"
   [ ... ] >
<famille>
...
</famille>
```

DTD – Entités

- Une séquence de caractères (nom de l'entité) utilisée pour représenter une autre séquence de caractères (contenu de l'entité).
- Une entité est déclaré dans une DTD.
- Déclaration d'une entité générale (ne peut s'appliquer dans la DTD).

```
<!ENTITY nom "contenu">
```

- Exemple (code Unicode pour copyright)

```
<!ENTITY copyright "&#xa9">
```

Dans le document : `<copyright> &copyright; 2004</copyright>`

DTD – Entités (2)

- Une entité paramètrée s'applique dans une DTD.
- Déclaration :

```
<!ENTITY % nom contenu>
```

- Exemple

```
<!ENTITY % pcdata "#PCDATA">

<!ELEMENT titre %pcdata;>
```

DTD – Entités externes

- Permet de copier le contenu d'un document XML dans le document courant.
- Déclaration

```
<!ENTITY contenudoc1 SYSTEM "
  http://www.toto.com/doc1.xml">

<document>

   &contenudoc1;

</document>
```

-

DTD – Entités non analysées

- Permet de copier le contenu d'un document quelconque dans le document courant.
- Déclaration

```
<!ENTITY image1 SYSTEM "
http://www.toto.com/image/photo1.gif" NDATA
GIF89a>
<image src="image1"/>
```

- En général l'attribut (ici src) est du type ENTITY ou ENTITIES.

Limites de DTD

- Syntaxe non XML
- Pas de support des espaces de noms
- Modularité limitée
- Modèle limité :
  - Contraintes de structures trop simplistes (nombre, modèle mixte, etc.).
  - Pas de contraintes sur les données
- Autres solutions :
  - XML Schema (W3C), RelaxNG (OASIS)

XML Schema

- Développé par le W3C
- Norme officielle
- Modèle riche et évolué
- Syntaxe XML
- Exploitation des espaces de noms
- Mais verbeux et complexe

XSD - Les types

- Notion centrale de type :
  - Chaque élément (attribut) possède un type.
  - Approche objet : il existe de types de base et on peut dériver de nouveaux types.
  - Catégories :
    - Types simples (simpleType)
      - Un élément de ce type ne peut contenir d'autres éléments. Un attribut présente un type simple.
    - Types complexes (complexType)
      - Les éléments qui contiennent d'autres éléments et/ou des attributs.

Structure d'un schéma XML

- C'est un document XML bien formé(arbre) avec :
  - Une racine `schema`,
  - Une suite de définitions de types, d'éléments et d'attributs

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema
   ">
...
</xsd:schema>
```

XSD : définition d'un élément

- Au niveau global
- Dans la définition d'un type complexe
- Avec un élément `element` :
  - Attribut `name` : nom de l'élément.
  - Attribut `type` : type de l'élément.
  - Attribut `fixed` : valeur de l'élément fixée ou à préciser.
  - Attribut `default` : valeur par défaut lorsque l'élément est présent mais avec un contenu vide.
  - Attribut `minOccurs, maxOccurs` : spécifier les cardinalités min et max.
- En donnant un type sans nom, on doit définir le type de l'élément directement comme descendant du noeud `element`.

XSD : définition d'un attribut

- Au niveau global
- Dans la définition d'un type complexe
- Avec un élément `attribut` :
  - Attribut `name` : nom de l'attribut.
  - Attribut `type` : type de l'attribut (forcément un type simple, éventuellement obtenu par dérivation).
  - Attribut `fixed` : valeur de l'élément fixée.
  - Attribut `default` : valeur par défaut.

XSD – types simples

- string,
- boolean,
- float, decimal, integer, nonPositiveInteger, negativeInteger, nonNegativeInteger, positiveInteger
- timeDuration, timeInstant, time, timePeriod, date, month, year, century
- UriReference
- ID, IDREF, ENTITY, NOTATION, QName, Name
- On peut restreindre chaque type à l'aide de facets et on obtient alors un type dérivé, toujours simple :
  - Valeur minimale ou/et maximale
  - Longueur (min, max, exacte)
  - Nombre de chiffres
  - Expression régulière
  - Énumération des valeurs possibles.
  - etc..

XSD – Exemple de dérivation de types simples

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

XSD – types complexes

- Construction à base de séquence(s) :
  - Élément `sequence`
  - Contenu indique les fils exigés par le type à l'aide des éléments `element` :
    - Attribut `ref` : permet d'utiliser un élément déjà défini
    - Attribut `minOccurs et maxOccurs` : définition des cardinalités
  - Contenu mixte : attribut `mixed="true"` pour l'élément complexType
  - On précise les attributs à l'aide d'`attribut` :
    - Attribut `ref` : permet d'utiliser un attribut déjà défini
    - Attribut `use` : précise si l'attribut est optional, prohibited ou required .

XSD – types complexes (2)

- L'attribut `content` de l'élément `complexType` peut prendre les valeurs suivantes :
  - `elementOnly`, la valeur par défaut, et le contenu de l'élément est composé d'un ou plusieurs élément(s).
  - `textOnly` pour un contenu du type string.
  - `empty` pour un contenu vide
  - `mixed` pour définir un contenu mixte.
- Exemple :

```
<xsd:element name="personne">
 <xsd:complexType content="empty">
  <xsd:attribute name="nom" type="xsd:string">
    ....
<personne nom="Durand"/>
```

XSD - Schema (exemple)

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<xsd:element name="article" >
    <xsd:complexType content="elementOnly">
      <xsd:element name="titre" type="xsd:string"
  minOccurs="1" />
      <xsd:element name="auteur" type="xsd:string"
  minOccurs="1" />
      <xsd:element name="date" type="xsd:string"
  minOccurs="1" />
      <xsd:element name="lieu" type="xsd:string"
  minOccurs="1" />
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Exemple avec schéma

```
<?xml version="1.0"?>
<famille
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="ele1.xsd">
   <personne>
     <nom>Durand</nom>
     <prenom>Jean</prenom>
     <age>28 ans</age>
   </personne>
   <personne>
     <nom>Durand</nom>
     <prenom>Marie</prenom>
     <age>20 ans</age>
   </personne>
</famille>
```

Et son schéma

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="famille">
   <xsd:complexType>
     <xsd:sequence>
        <xsd:element ref="personne" minOccurs="1" maxOccurs="unbounded"/>
     </xsd:sequence>
   </xsd:complexType>
</xsd:element>
<xsd:element name="personne">
   <xsd:complexType>
     <xsd:sequence>
        <xsd:element name="nom" type="xsd:string"/>
        <xsd:element name="prenom" type="xsd:string"/>
        <xsd:element name="age" type="xsd:string"/>
     </xsd:sequence>
   </xsd:complexType>
</xsd:element>
</xsd:schema>
```

XSD - Conclusion

- Bien plus complexe et riche que DTD
- Un vrai langage qui demande un étude des documents officiels (http://www.w3.org/XML/Schema) :
  - XML Schema Part 0 : Primer
  - XML Schema Part 1 : Structures
  - XML Schema Part 2 : Datatypes
- XML Schema 1.1. en cours de développement.

RelaxNG

- Développé par le consortium OASIS (1)
- Norme compact et simple
- Norme modulaire
- Syntaxe XML (Fusion de Trex et Relax)
- Exploitation des espaces de noms
- Mais n'est pas la norme du Web (W3C)
- Mais son contenu est moins riche que XML Schema.

1 : http://www.oasis-open.org

Relax NG

- Spécification de décembre 2001 (1)
- Buts :
  - Proposer une syntaxe XML
  - Avec support des espaces de noms
  - Proposant un modèle plus efficace que les DTD (modèle non ordonné)
  - Support de types de données complexes (non inclus par défaut).

1 : http://relaxng.org/spec-20011203.html

Document XML et son schéma Relax NG

```
<addressBook>
  <card>
    <name>John Smith</name>
    <email>js@example.com</email>
  </card>
  <card>
    <name>Fred Bloggs</name>
    <email>fb@example.net</email>
  </card>
</addressBook>
```

Schéma DTD

```
<!DOCTYPE addressBook [
<!ELEMENT addressBook (card*)>
<!ELEMENT card (name, email)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
]>
```

Schéma Relax NG

```
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="card">
      <element name="name">
        <text/>
      </element>
      <element name="email">
        <text/>
      </element>
    </element>
  </zeroOrMore>
</element>
```

Relax NG

- Les éléments Relax NG sont obligatoirement dans http://relaxng.org/ns/structure/1.0
- Les éléments servent à définir des patterns (modèles) qui seront comparés au document pour savoir si celui-ci est valide.
- Eléments :
  - Element : declaration d'un élément avec nom comme attribut
  - Gestion des répétitions : par défaut (une occurence), oneOrMore, zeroOrMore, optionnal (un au plus).
  - Text : du texte.

Schematron

- Un langage de schéma structuré pour documents XML.
- Fonctionne en effectuant des concordances avec des expressions Xpath. Exploite la syntaxe XSL.
- Aussi utilisé pour l'annotation de document XML en générant automatique des éléments.
- Détails : http://www.schematron.com

3. API Java et XML

Ouvrages recommandés :

- Java et XML 2nd édition– B. McLaughlin - Edition O'Reilly (Bibliothèque UMLV)
- XML in theory and practice – C. Bates – Edition Wiley

Pourquoi Java et XML

- Java : code portable (write once, run anywhere)
- XML : données portables
- On peut aussi utiliser d'autres langages :
  - Visual Basic : parser de Internet Explorer 6, spécifique à une plate-forme

Autres langages

- C++ : plusieurs parsers (Xercès, IE6, etc.), complexe
- Perl : librairies XML importantes
- Python : librairies XML importantes, exploitation marginale.
- Lisp et Scheme : librairies XML intéressantes, optimisation importante (parsing Scheme + rapide que C++)

Java et XML

- Java est populaire
- Java et XML évoluent dans des environnements similaires : applications réseaux, interopérables et systèmes hétérogènes (indépendance de la plate-forme).

Traiter du XML

- Parser : "a computer program that divides code up into functional components" *(hyperdictionary.com)*.
- Ecrire un parser est une tâche complexe (implémentation, optimisation, etc.). On profite donc des parsers disponibles (libres/ propriétaires) => compatibilité ascendante, évolution ?
- Nécessité de changer de parser sur la durée de vie d'une application.

JAXP (Java API for XML Parsing)

- Une initiative de Sun Microsystems pour créer une API indépendante de la structure des données et des traitements.
- JAXP est une API standard pour les traitements et transformations associés à XML.

JAXP (2)

- JAXP supporte DOM et SAX (les deux principales API).
- JAXP propose des méthodes qui peuvent être exploitées avec des API compatibles.
- Un parser est chargé lors de l'exécution par une classe *factory* (cf. Design pattern, décrit comment les classes peuvent être instanciées et chargées au moment de l'exécution).
- Exemple :

`javax.xml.parsers.SAXParserFactory` charge un parser SAX.

`javax.xml.parsers.DocumentBuilderFactory` charge un constructeur DOM

Xerces

- Xerces (Apache foundation) propose un parser SAX et DOM.
- Compatibilité avec les recommandations du W3C : XML 1.0, Namespaces, XML Schema 1.0, DOM level 2 avec SAX 2, JAXP 1.1
- Xerces 2 supporte XML Schema

API Java pour XML

- API de bas niveau (*low-level API*) : elle permet un traitement direct des documents XML
  - La sortie est du XML
  - Puissantes et efficaces
  - Nécéssitent une bonne connaissance de XML
  - Les solutions : SAX (Simple API for XML), DOM (Document Object Model), JDOM, JAXP.

API Java pour XML(2)

- API de haut niveau (*high-level API*) : elle exploite une API de bas niveau.
  - Divers formats en sortie
  - Développement rapide et facile mais traitement lourd
  - Temps d'apprentissage de l'API important
  - Solution : XML Data Binding (transforme un document XML et objets Java).

Parser XML

- Un parser est un programme qui va lire un document XML et analyser sa structure.
  - On peut aussi dire qu'un parser transforme un document XML en une structure de donnée qui peut être manipulée.
- Quelques fonctionnalités :
  - Assure que le document est bien formé.
  - Si un schéma existe, assure que le document est valide.

Parser XML (2)

- Arguments pour choisir un parser : efficacité, conformité aux normes (W3C).
- Solutions : Xercès (Apache), XP, XML4J (IBM), Crimson (Sun), MSXML (Microsoft), Oracle XML Parser.

Parser XML (3)

- En général, on utilise un parser de la manière suivante :
  - Création d'un objet parser
  - On pointe cet objet sur un document XML
  - On réalise des traitements (par exemple créer une page web ou bien générer un bon de commande en PDF).

SAX (Simple API for XML)

- SAX est une API XML événementielle (*event-driven*).
  - Elle ne construit pas une représentation du document XML en mémoire.
  - Le programme reçoit des notifications lorsque le parser reconnaît des parties spécifiques d'un document XML. Les notifications (*events*) sont gérées par des gestionnaires de notifications (*event handlers*). 3 types de event handlers :
    - DTDHandler : accès au contenu de la DTD
    - ErrorHandler : accès aux erreurs d'analyse
    - ContentHandler : accès au contenu du document.

Evénements avec SAX

Le document :

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

<personne>

    <prenom>Pierre</prenom>

    <nom>Durand</nom>

</personne>
```

Les événements :

```
start document
start element: personne
start element: prenom
characters: Pierre
end element: prenom
start element: nom
characters: Durand
end element: nom
end element: personne
end document
```

Le paquetage org.xml.sax

- org.xml.sax
  - Contient des interfaces et classes permettant la manipulation de documents.
  - `Interfaces`
    - `AttributeList, Attributes, ContentHandler, DocumentHandler, DTDHandler, EntityResolver, ErrorHandler, Locator, Parser, XMLFilter, XMLReader`
  - `Classes`
    - `HandlerBase, InputSource`

Interface ContentHandler

- Cette interface permet de notifier :
  - Le début et la fin d'un document.
  - Le début et la fin d'un élément.
  - Les données
  - Les espaces de noms
  - Les "processing instructions" (PI)
  - D'ignorer les 'espaces'.

Interface ContentHandler (2)

- Méthodes
  - characters : gère les noeuds textuels
  - startDocument et endDocument : gère le début et la fin d'un document.
  - startElement et endElement : gère le début et la fin d'un élément.
- CF. javaDoc

Interface XMLReader

- Le point d'entrée d'une application SAX est l'interface XMLReader qui contient les méthodes qui vont permettre :
  - De contrôler la manière dont va opérer le parser.
  - D'activer/ désactiver certaines fonctionnalités de SAX (gestion des espaces de noms).
  - D'instancier des objets pouvant recevoir des notifications.
  - D'initialiser le flux d'entrée du parser.

Un exemple (1)

```
import org.xml.sax.*;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXException;
import java.io.IOException;
class MonHandler extends DefaultHandler {
public void startDocument() throws SAXException {
System.out.println("Debut du document");
}
public void endDocument() throws SAXException {
System.out.println("Fin du document");
}
public void startElement(String namespaceURI, String localName, String
   qName, Attributes atts) {
System.out.println("Debut de balise");
System.out.println("       Nom local       :" +localName);
System.out.println("Length ="+atts.getLength());
for(int cpt=0;cpt<atts.getLength();cpt++)
System.out.println(atts.getLocalName(cpt)+" = "+ atts.getValue(cpt));
}
```

Un exemple (2)

```
public void endElement(String namespaceURI, String localName, String
    qName) {
System.out.println("Fin de balise :"); }
}
public class Ex2SAX {
private String docName;
    public Ex2SAX(String docName) {
    this.docName = docName;
    MonHandler monHandler = new MonHandler();
    try {
    XMLReader parser =
    XMLReaderFactory.createXMLReader("org.apache.xerces.parsers.SAXParser
    ");
    try {
    parser.setContentHandler(monHandler);
    parser.setFeature("http://xml.org/sax/features/validation", true);
                        parser.parse(docName); }
    catch (IOException ioexc) {
    System.out.println("PB acces a ele1.xml");}
    }
    catch (SAXException saxexc) {
    System.out.println("Pb SAX" + saxexc.getMessage());    }    }
```

Un exemple (3)

```
    public static void main(String[] args) {
    System.out.println("Exemple  - SAX");
    if(args.length==1)
    {
    Ex2SAX exSax = new Ex2SAX(args[0 ]);
    }
    else
    System.out.println("1 argument requis");
    }
}
```

Bilan SAX

- Avantages :
  - utilisation mémoire et performance de traitement sont indépendantes de la taille du document.
  - Traitement possible avec une lecture complète du document.
- Inconvénients :
  - Le développeur est responsable de la structure de donnée qui stocke les informations.
  - Pas adaptée à un accès aléatoire aux données du document.

DOM (Document Object Model)

- Une recommendation du W3C.
- N'est pas conçu spécifiquement pour le langage Java.
- Représente le contenu et le modèle d'un document quelque soit le langage de programmation.
- Architecture en couches
  - Le niveau 1 détaille les fonctionnalités et la navigation dans un document (pas nécessairement du XML).
  - Le niveau 2 ajoute des fonctionnalités pour XML, HTML et CSS.
  - Le niveau 3 (Load and Save) est une recommandation (avril 2004)

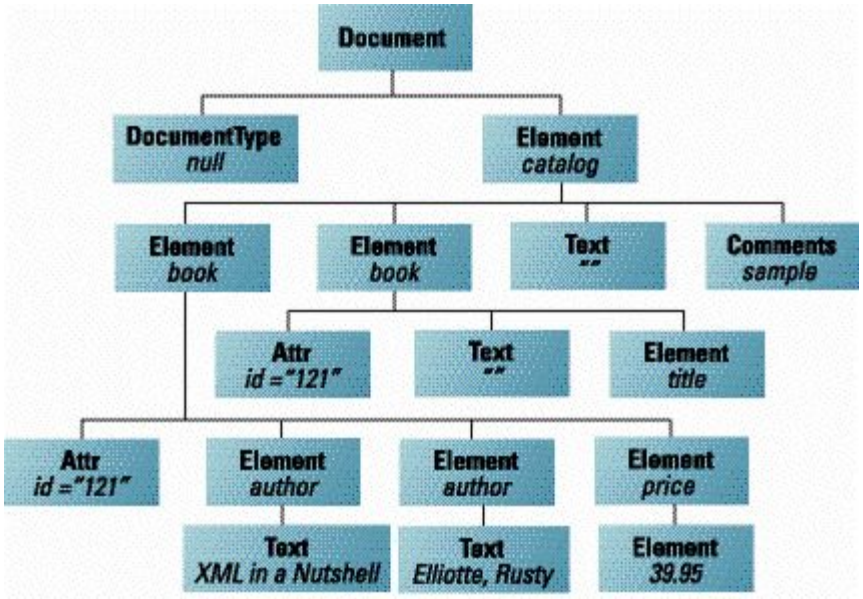
DOM (suite)

- En analysant un document XML avec DOM, on obtient un arbre qui représente le contenu du document (texte, éléments et attributs).
- DOM ne fournit pas d'information sur le codage du document (attention aux espaces).

Exemple d'arbre DOM

```
<catalog>
  <book id="101">
    <title>XML in a Nutshell</title>
    <author>Elliotte Rusty Harold, W.
   Scott Means</author>

    <price>39.95</price>
  </book>
  <book id="121">
    <title>Who Moved My Cheese</title>
    <author>Spencer, M.D. Johnson,
   Kenneth H. Blanchard</author>
    <price>19.95</price>
  </book>
</catalog>
```



Exemple DOM

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document; import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node; import org.w3c.dom.NodeList;
public class Ex1Dom {
private String uri; private Document doc;
private int docNd, eleNd, attNd, txtNd;
private static long debut, fin;
public Ex1Dom(String uri) {
this.uri = uri;
doc = null; docNd = 0; eleNd = 0; attNd = 0; txtNd = 0;
try {
  DocumentBuilderFactory dbf =
   DocumentBuilderFactory.newInstance();
  DocumentBuilder db = dbf.newDocumentBuilder();
  debut = System.currentTimeMillis(); doc = db.parse(uri);
  fin = System.currentTimeMillis();
  if (doc != null) { count(doc); printStats(); }
}
```

Exemple DOM (2)

```
catch (Exception e) {
  System.err.println("Erreur : " + e.getMessage());}
}
public void count(Node node) {
int type = node.getNodeType();
switch (type)
{
  case Node.DOCUMENT_NODE: {
  docNd++;   processChild(node);   break;
    }

  case Node.ELEMENT_NODE: {
  eleNd++;   System.out.println(node.getNodeName());
  NamedNodeMap atts = node.getAttributes();
  if(atts.getLength()>0)     attNd += atts.getLength();
  processChild(node);   break;
}
```

Exemple DOM (3)

```
    case Node.TEXT_NODE: {
   txtNd++;  break;}
     } // ferme le switch

} // ferme la méthode count
public void processChild(Node node) {
NodeList sousNd;
sousNd = node.getChildNodes();
for(int cpt=0;cpt<sousNd.getLength();cpt++)
  count(sousNd.item(cpt));
}
public void printStats() {
System.out.println("Document node = "+docNd);
System.out.println("Element nodes = "+eleNd);
System.out.println("Attribute nodes = "+attNd);
System.out.println("Text nodes = "+txtNd);
System.out.println("Duree parsing = "+Long.toString(fin-debut)
   +" ms");
}
```

Exemple DOM (4)

```
    public static void main(String[] args) {
        Ex1Dom ex1Dom = new Ex1Dom(args[0]);
    }
}// ferme la classe Ex1Dom
```

Bilan de DOM

- Structure arbre propose une bonne solution pour la navigation dans le document.
- Construction de l'arbre en un seul passage (pas d'analyse partiel du document).
- Construction en mémoire => attention aux documents volumineux.
- Avantages :
  - Accès aléatoire
  - Ecriture dans le document.

JDOM

- Une réponse à ceux qui trouvent SAX et DOM compliqués.
- Comme DOM, JDOM propose un arbre du contenu du document.
- JDOM inclut des adaptateurs qui utilisent des parsers SAX et DOM en tâche de fond.
- JDOM est spécifique à Java alors que DOM est neutre en terme d'exploitation d'un langage de programmation. JDOM exploite de nombreuses caractéristiques de Java (collections, surcharge de méthodes, etc..).
- JDOM est une API open-source : www.jdom.org

JDOM (2)

- JDOM n'est pas un parser, c'est une représentation en Java d'un document XML.
- L'exploitation de JDOM repose donc sur l'utilisation d'un parser pour lire les documents XML. JDOM accepte également les événements SAX et les arbres de DOM.
- Pour faciliter ce mécanisme, JDOM propose le package org.jdom.input. Ce package contient des classes *builder.* Nous utiliserons le plus souvent `SAXBuilder` et `DOMbuilder`.

Builders

```
SAXBuilder builder = new SAXBuilder( );

Document doc = builder.build(new
   FileInputStream("contents.xml"));
```

- Ou pour DOM:

```
DOMBuilder builder = new DOMBuilder( );

Document doc = builder.build(myDomDocumentObject);
```

- Attention, DOMBuilder exploite SAX pour construire l'arbre DOM puis une traduction intervient pour obtenir l'arbre JDOM. C'est moins rapide que d'exploiter SAX pour construire l'arbre JDOM.JDOMBuilder est adapté lorsque l'entrée est déjà un arbre DOM pas pour un flux texte XML.

Gérer la sortie

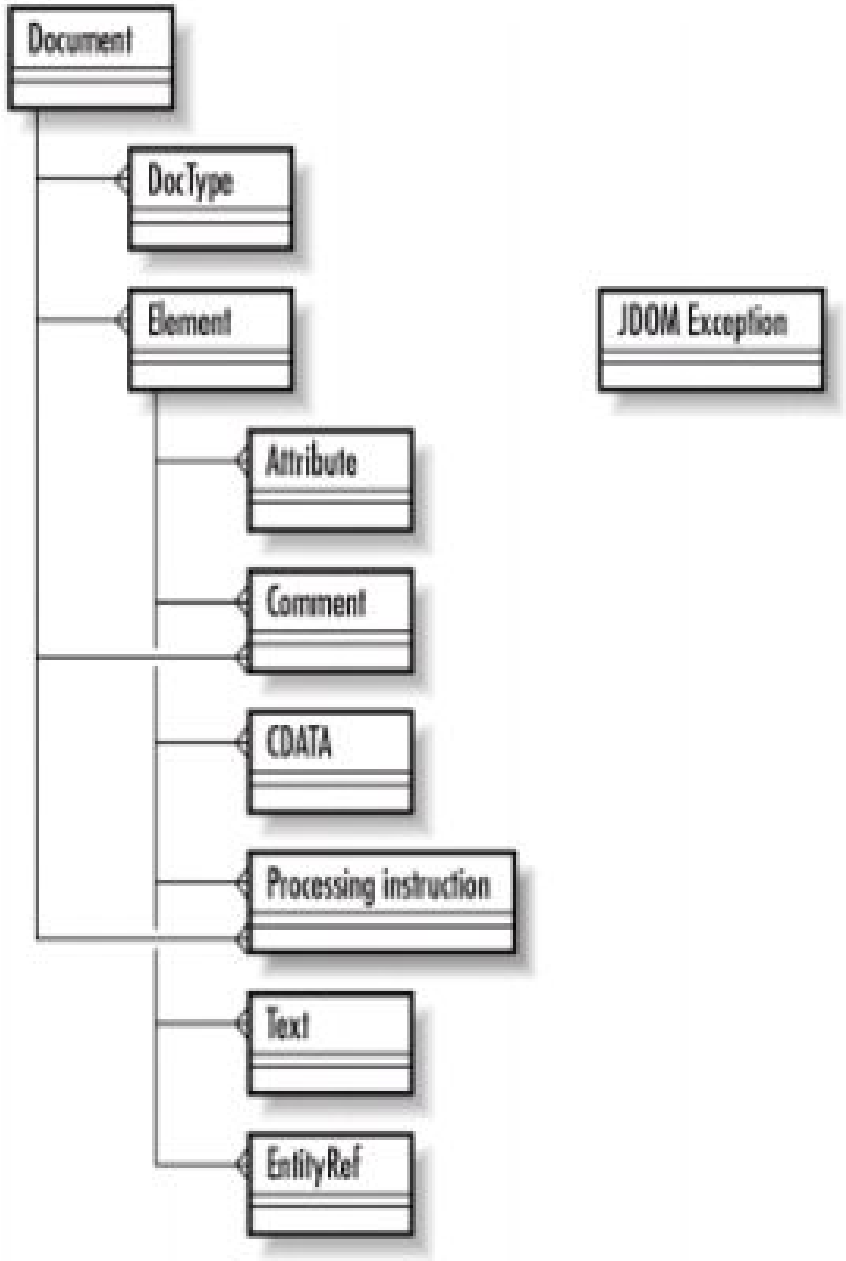
- Les classe XMLOutputter, mais aussi DOMOutputter et SAXOutputter sont disponibles.
- Exemple:

```
XMLOutputter outputter = new
  XMLOutputter();

outputter.output(jdomDocObject,"new
  FileOutputStream("resultat.xml"));
```

Modèle de JDOM

• La classe Document est la représentation d'un document XML et la super classe des classes de JDOM
•Element et Attribute représentent respectivement un élément et un attribut XML.



Element

- Avec DOM, le contenu d'une balise est un noeud.
- Avec JDOM, la classe Element propose une méthode getText().
  - Plus intuitif pour un développeur Java.

Création d'un document

```
Element elementRacine = new
  Element("racine");
Document document = new
  Document(elementRacine);
```

Exemple JDOM

```
import java.io.File; import org.jdom.Document;
import org.jdom.input.SAXBuilder; import org.jdom.output.XMLOutputter;
public class JDomOne
{
  public static void main(String[] argv)
  {
    if (argv.length == 0 || (argv.length == 1 && argv[0].equals("-
   help")))
    {
      System.out.println("1 argument necessaire (document xml).");
      System.exit(1);
    }
    try
    {
      SAXBuilder sb = new SAXBuilder();
      Document doc = sb.build(new File(argv[0]));
      XMLOutputter xo = new XMLOutputter();
      xo.output(doc, System.out);
    }
    catch (Exception e)
    {       e.printStackTrace();     }
  }
}
```

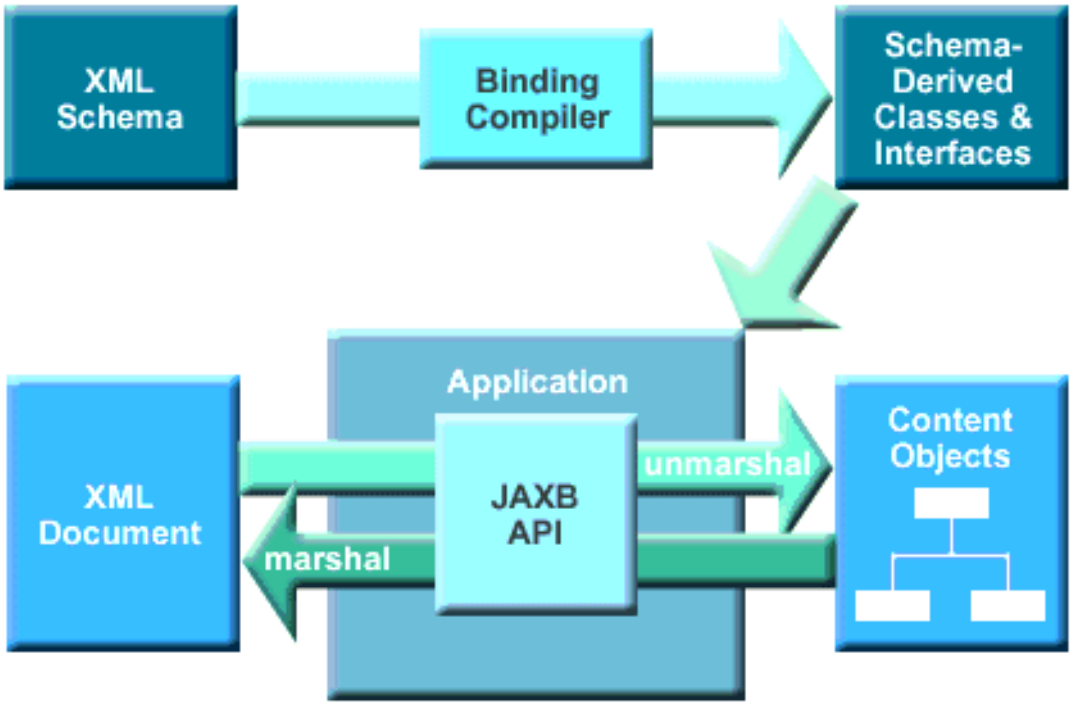
Data Binding

- Le data binding est le processus de représenter des bits de données (issues d'un fichier texte, d'une base de données, d'un document XML) sous une forme de données dans un langage de programmation (sur lesquelles il sera possible de réaliser des opérations).
- Du XML data binding en Java permet de transformer un document XML en des objets Java.

Lexique du data binding

- Marshalling : convertir des données en mémoire vers un média de stockage (BD, fichier texte, XML). Ex : convertir des objets Java en un document XML.
- Unmarshalling : convertir les données d'une structure de stockage vers des données en mémoire. Ex : convertir d'un document XML vers des objets Java.
- Round-tripping : terme exprimant un cycle complet de marshalling-unmarshalling.
- Une API efficace doit permettre une équivalence sémantique lors d'un cycle complet (round-tripping).

Architecture JAXB



Génération de classes

- Elle se base sur le schéma du document XML.
- Suivant l'API : DTD, XML schema ou RelaxNG.
- Les solutions :
  - Castor (Exolab) , Zeus (Enhydra) , JAXB (Sun), JaxMe (Apache).

Problèmes

- Problème de typage des données dans le langage de programmation.