Traitement de documents XML en JAVA

JAXP (SAX , DOM , TrAX) et JAXB , XQuery

Table des matières

I - Cadre XML

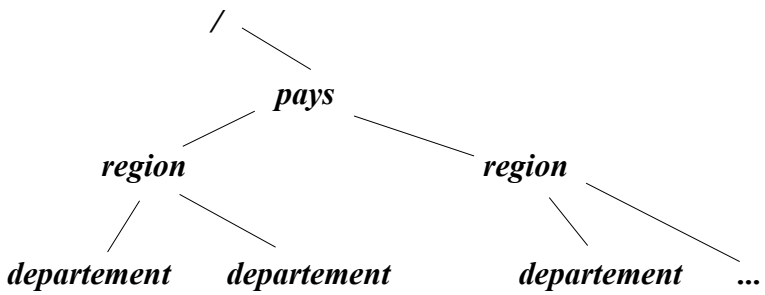
XML signifiant *eXtended Markup Language* est un **langage standardisé (w3c)** permettant d'**encoder des données structurées** de *manière hiérarchique* .
XML est **eXtensible** car il permet (contrairement à HTML) d'utiliser des noms de balises quelconques (ex: chapitre , region , ...) .

Exemple:

```
<?xml  version='1.0'  encoding='UTF-8'  standalone='yes' ?>
<?xml-stylesheet  type="text/xslt"  href='fic_style.xslt' ?>
<!-- commentaire -->
<pays  nom='France'  capitale='Paris' >
  <region nom='Haute-Normandie' >
     <departement  nom='Eure'          num='27'  prefecture='Evreux' />
    <departement  nom='Seine Maritime'  num='76'  prefecture='Rouen' />
  </region>
  <region nom='Picardie'>
     <departement  nom='Oise'    num='60'  prefecture='Beauvais' /> ...
  </region> ...
</pays>
```

Remarques:

Ce **document xml** comporte des **données** dont **les significations sont clairement indiquées par les attributs ou par les noms des balises englobantes**.
La **structure** globale de ce document s'apparente à celle de l'**arbre** suivant:



Aucune mise en forme (grs, italique , couleur , alignement, ...) n'apparaît directement dans le fichier xml. Par contre l'**application d'une feuille de style externe (CSS ou XSL)** permet d'obtenir facilement une présentation de ce genre:

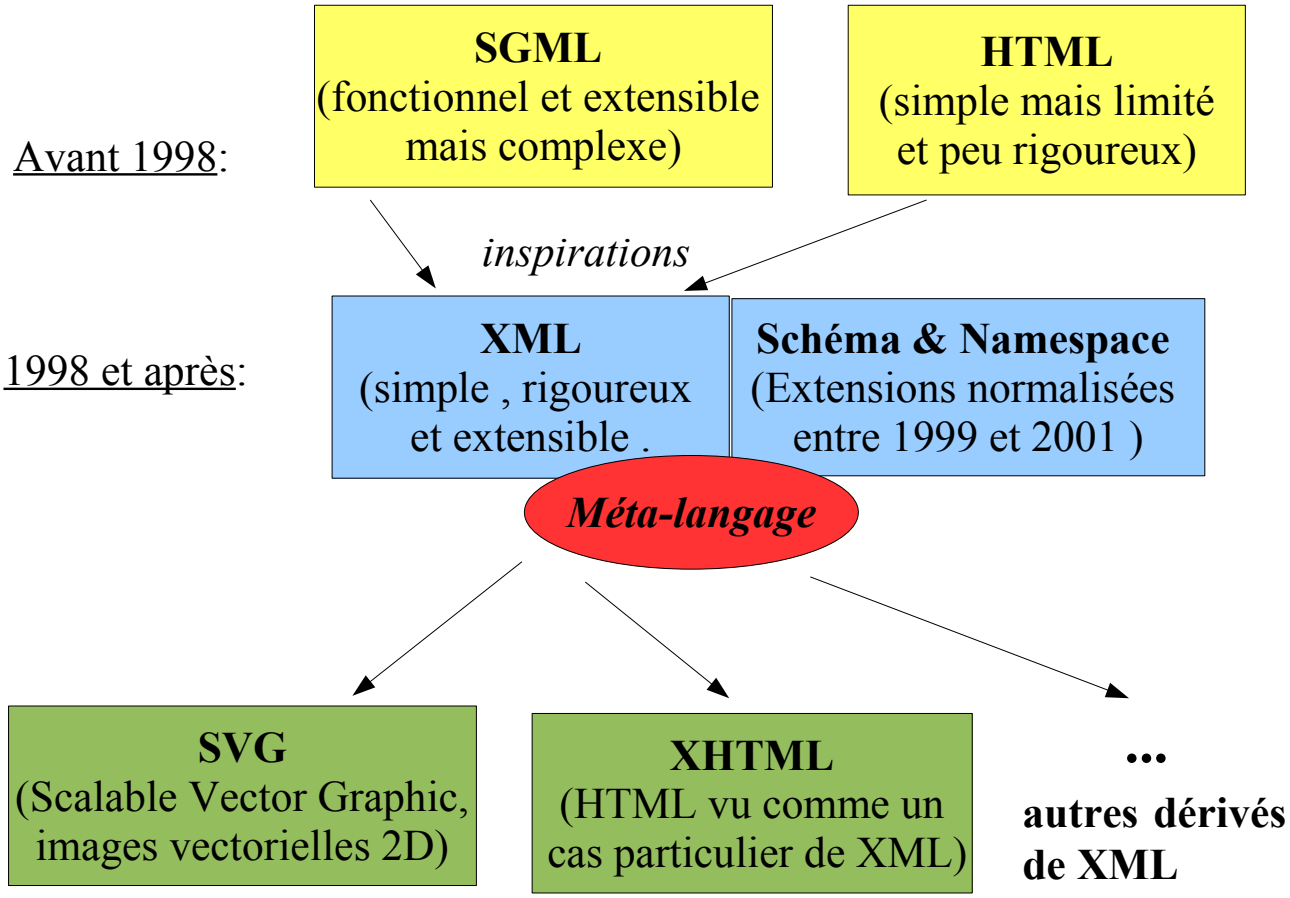
France

Haute-Normandie

num	departement	prefecture
27	Eure	Evreux
76	Seine Maritime	Rouen

1. Historique et évolution

XML – historique & évolution



Principales technologies :

DTD (Document Type Definition)	Dès 1998	Mécanisme permettant de **valider** un document en comparant la structure de celui par rapport à une **structure type** (arborescence imposée de balises).
Namespace XML	Depuis	**Identifiant de langage dérivé d'XML** (porté par l'attribut spécial **xmlns**). Les préfixes locaux associés permettent d'encoder des **documents composés** dont les différentes parties sont exprimées avec des langages différents.
Schéma XML (xsd)	Depuis 2001	**Mécanisme de validation plus évolués que les DTD**. Les 2 principaux apports sont : * syntaxe basée sur XML (==> homogène) * **typage fin des données** (string,integer, ...)
XHTML	Depuis	HTML vu comme un cas particulier de XML.

DTD (Document Type Definition)	Dès 1998	Mécanisme permettant de **valider** un document en comparant la structure de celui par rapport à une **structure type** (arborescence imposée de balises).
		XHTML 1.0 est le successeur officiel de HTML 4.

SVG (Scalable Vector Graphic)	Depuis	Langage permettant d'encoder des **images vectorielles** en XML (balises *line, circle , ...* avec *coordonnées*) .
XSLT	Depuis fin 1999	Partie **Transformation** des **feuilles de styles XSL** . Permet entre autre de tranformer un arbre XML (document sans mise en forme) en un arbre (X)HTML .

XML a été normalisé en 1998 par le *World Wide Web Consortium* (**W3C**) .
Le site web de référence (pour consulter les normes officielles) est www.w3.org

2. Principales caractéristiques d'XML

XML – principales caractéristiques

XML est un META-LANGAGE à partir duquel on peut *dériver* tout un tas de *langages particuliers (XHTML , SVG, MathML, …)*.
Les langages dérivés d'XML **(et normalisés) sont généralement *identifiés par des namespaces*** **(ex: http://www.w3.org/1999/xhtml) .**

• C'est un LANGAGE DE DESCRIPTION DE DONNEES comportant non seulement des valeurs textuelles mais également les *sémantiques* de celles-ci . Les *significations des données* sont renseignées au niveau des *noms de balises* d'encadrement (ou bien au niveau des noms de certains attributs).

• *Simple à encoder et à ré-interpréter* sur tout environnement *(Unix , Windows, ...)* et depuis n'importe-quel langage *(C/C++, Java, Perl/Php , ...)* , XML est une technologie clef permettant de garantir une bonne *interopérabilité* entre différents systèmes informatiques.

(X)HTML est un plutôt un langage de présentation (pour affichage à l'écran) .

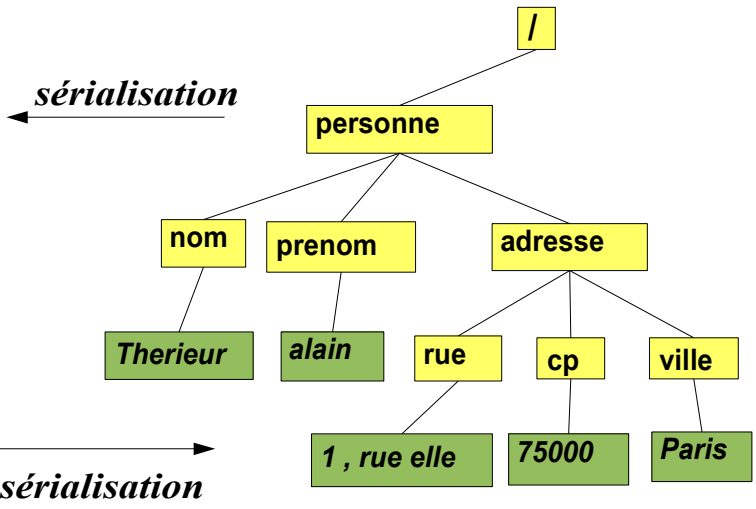
XML est plutôt un **langage de communication entre différentes applications** (des re-traitement sont presque toujours nécessaires).

XML est une **technologie transverse** que l'on retrouve partout (Présentation via feuilles de styles, Echange de données , Services WEB , B2C, B2B , Bureautique , ...).

Document XML = arbre de données

pers1.xml

```
<personne>
  <nom>Therieur</nom>
  <prenom>alain</prenom>
  <adresse>
     <rue>1, rue elle</rue>
     <cp>75000</cp>
     <ville>Paris</ville>
   </adresse>
</personne>
```



XML permet d'**encoder** une **structure de données arborescente** en une **suite de caractères séquentiels** *(paquet d'octets dans un fichier ou flux réseau)* .

3. Documents XML bien formés

XML – documents bien formés

Un document XML est dit « **bien formé** » s'il est **syntaxiquement correct** ; il doit pour cela vérifier les principales règles suivantes:

- **Toute balise ouverte doit être fermée**:
  ** ** paragraphe *mais* ** ** paragraphe ** **
  ligne **
** *mais* ligne **
</br>** *ou* ligne **
**
  sachant que **<elt_vide a1='v1' ></elt_vide> <==> <elt_vide a1='v1' />**

- **Une seule balise de premier niveau englobant toutes les autres**:
  <pers>....</pers> <pers>...</pers> *mais*
  <liste_pers><pers>....</pers> <pers>...</pers>**</liste_pers>**

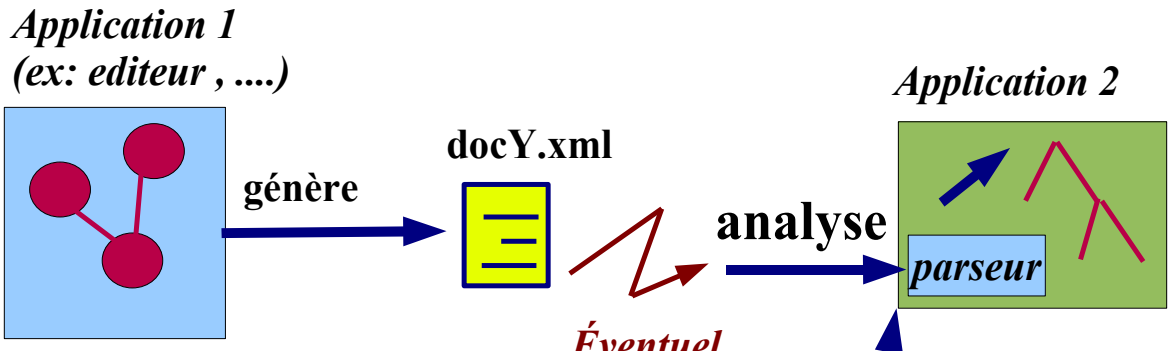
- **Pas de chevauchement de balises mais des imbrications claires**:
  <g>blabla</g> *mais* **<g>**blabla**</g>**

- **Les valeurs des attributs sont obligatoirement entre simple ou double quote**:
  <pers age=35>... *mais* **<pers age='35'>**... *ou* **<pers age=''35'' >**...

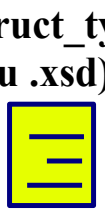
NB: Un document XML est par défaut encodé avec le jeu de caractères **UTF-8** .

4. Documents valides

XML – Validation (via DTD ou Schema)



Structure type précise imposant certaines imbrications de balises. Ceci contractualise le format des documents échangés. La sémantique des données qu'il faudra analyser est ainsi bien convenue.



Objectif : Rejeter tous les documents non conformes

4.1. DOCTYPE (DTD)

DOCTYPE (entités & validation)

personne.dtd

```
<!ELEMENT personne (titre?,nom,prenom+,adresse*) >
<!ATTLIST personne age CDATA #IMPLIED>
<!ELEMENT titre (#PCDATA) >
<!ELEMENT nom (#PCDATA) >
<!ELEMENT prenom (#PCDATA) >
...
```

VERIFICATION: conforme à la structure type attendue ?

personne.xml

```
<?xml version='1.0'?>
<!DOCTYPE personne SYSTEM "personne.dtd"
[
 <!ENTITY Mr "Monsieur" >
 <!ENTITY adr SYSTEM "./adresse.xml" >
 ]>
<personne age="35">
   <titre>&Mr;</titre><nom>Defrance</nom><prenom>Didier</prenom>
    &adr;
</personne>
```

ENTITY ==> *jeux de remplacements*

4.2. Schéma W3C

Schéma XML (.xsd) plus évolués que DTD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name='pays' type='PaysType' />
  <xsd:complexType name="PaysType">
          <xsd:sequence>
               <xsd:element name="region" type="RegionType" maxOccurs='22' />
          </xsd:sequence>
          <xsd:attribute name="nom" type="xsd:string"/>
          <xsd:attribute name="capitale" type="xsd:string"/>
 </xsd:complexType>
 <xsd:complexType name="RegionType">...</xsd:complexType>
 <xsd:complexType name="DepartementType">
          <xsd:attribute name="nom" type="xsd:string"/>
          <xsd:attribute name="num" type="xsd:positiveInteger"/> ....
 </xsd:complexType>
</xsd:schema>
```

france.xsd

Types précis

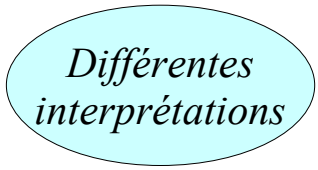
```
<pays xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="france.xsd"
        nom='France' capitale='Paris' >
   <region nom='Haute-Normandie' >   <departement nom='Eure' num='27' ... />
...</region> ... </pays>
```

5. Namespaces XML

Namespace XML et documents composés

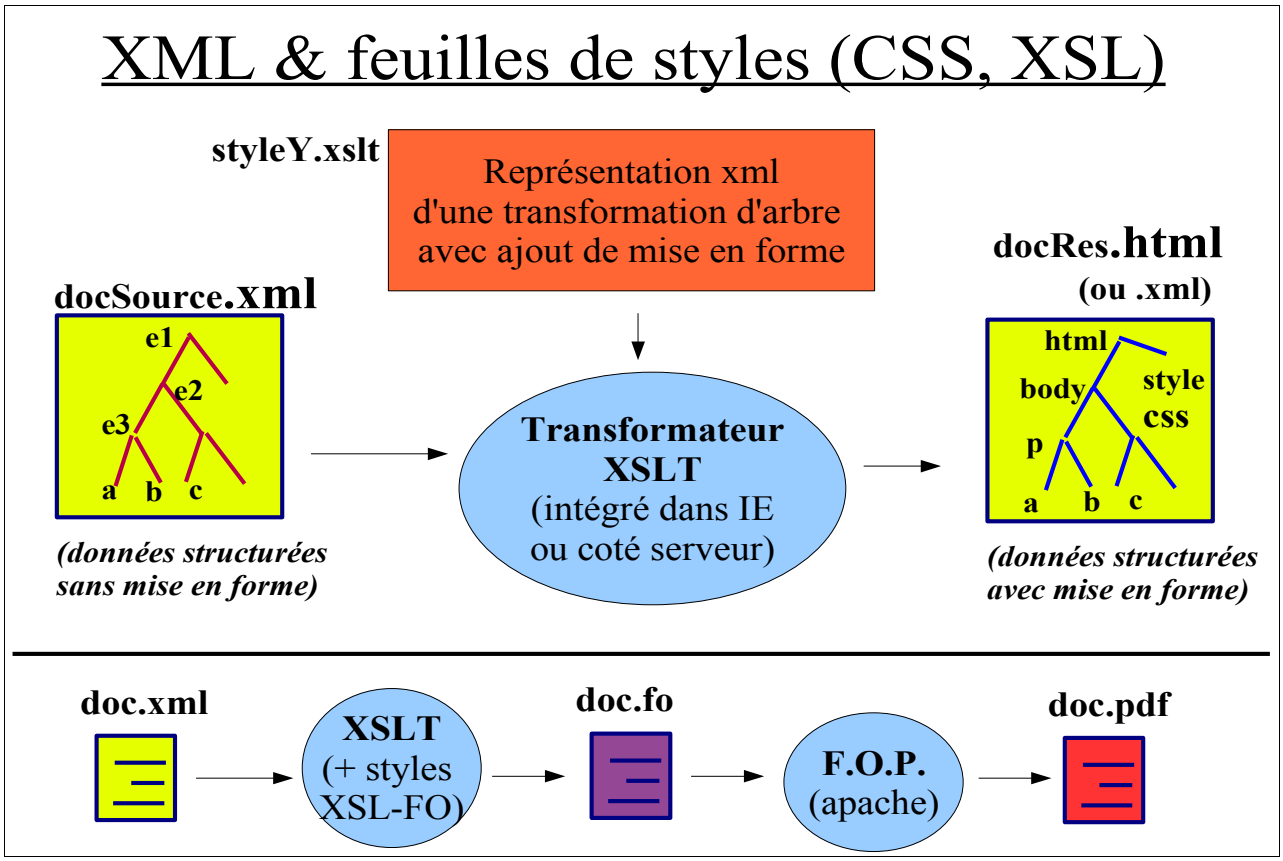
Un **namespace XML** est un **identificateur** de **langage dérivé d'XML**
Cet **ID** prend généralement la forme d'un **URI (*Exemple*: *'http://www.yyy.com/Année/Norme'*)** *[sans téléchargement]*.
Les **namespaces** permettent d'encoder des **documents XML composés** dont les différentes parties sont exprimées dans des langages distincts .

```
<h:html xmlns:h='http://www.w3.org/1999/xhtml'
        xmlns:b='http://www.yyy.com/2004/biblio' >
<h:head>
   <h:title> bibliographie sur XML  </h:title>
</h:head>
<h:body>
   <h:p>  <b:title> XML et XSLT </b:title> </h:p>
</h:body>
</h:html>
```



Syntaxe générale: **xmlns:***prefixe_local*='***nom_du_namespace***'

6. Feuilles de styles (CSS & XSL)



II - JAXP

1. Présentation de JAXP

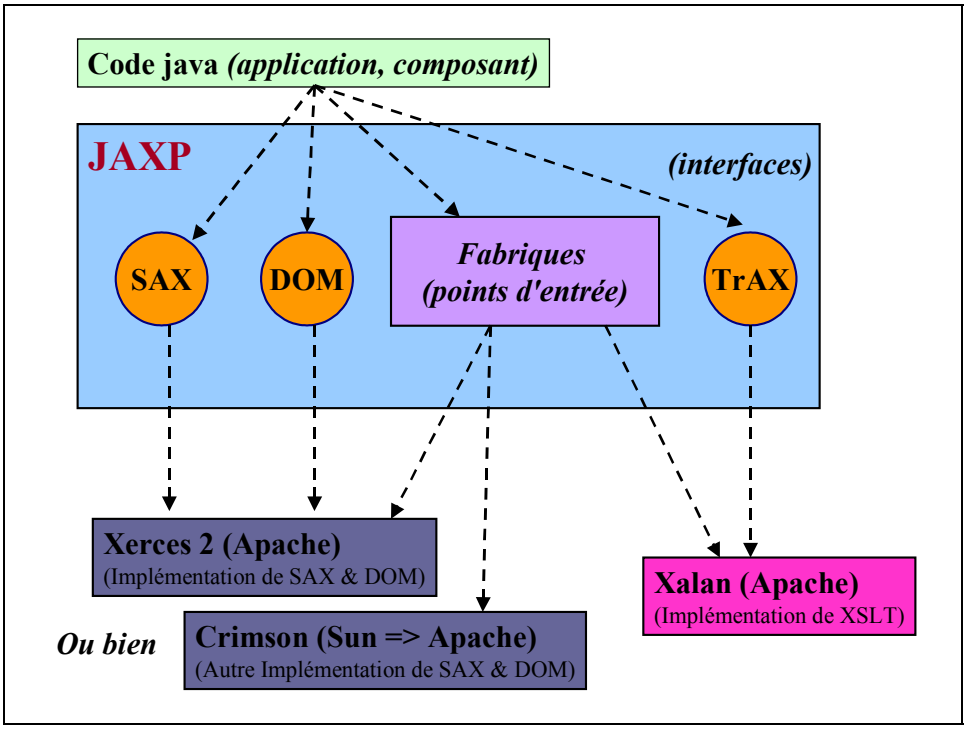
Via le site Internet de SUN/JAVASOFT (***www.javasoft.com***) , on peut télécharger l'api **JAXP** ***(Java Api for XML Processing)*** (url: ***http://java.sun.com/xml***).

JAXP est essentiellement un **jeu d'interfaces** se voulant "standard" dans le monde java.
JAXP s'appuie sur **TrAX** (Transformation Api for Xml) pour invoquer des transformations XSLT.

Historique des versions antérieures de Jaxp :

Jaxp 1.0	{ **jaxp.jar** (1.0) + **parser.jar** (sans NameSpace) }
Jaxp 1.1	{ **jaxp.jar** (1.1) + **crimson.jar** (avec NameSpace) } + **xalan.jar** (transf.)
Jaxp 1.2	{ **xerces.jar** (comportant jaxp) } + **xalan.jar** (version n+1) ***à ajouter au jdk1.3*** ou bien api **intégrée** dans le ***jdk 1.4*** ou bien version récente (***Jaxp 1.2.X***) à récupérer au sein de WSDP.

2. Structure de JAXP et implémentations

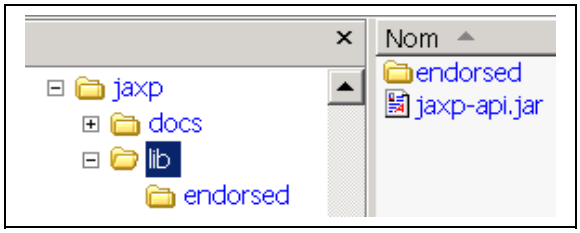


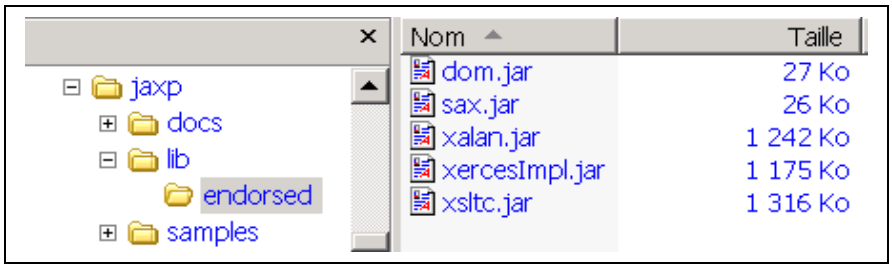
3. JAXP & JDK (versions)

Le jdk 1.3 est livré sans JAXP ==> beaucoup d'archives (.JAR) à ajouter au CLASSPATH.
Le jdk 1.4 est livré avec JAXP (on peut tout de même remplacer Crimson par Xerces 2 pour améliorer l'implémentation interne)

4. Archives java (.jar) correspondant à JAXP 1.2.3

Archive	*Package(s) incorporé(s)*	*Contenu*
jaxp-api.jar	***javax.xml.parsers*** **, ...** ***javax.xml.transform*** **, ...**	**TrAX + Fabriques de JAXP**
sax.jar	***org.xml.sax*** ***org.xml.sax.helpers***	**SAX (Api , interfaces)**
dom.jar	***org.w3c.dom***	**DOM (Api , interfaces)**
xercesImpl.jar	***org.apache.xerces*** ***org.apache.xerces.util*** **, ...**	**Implémentation** de SAX & DOM
xalan.jar	***org.apache.xalan.xslt***	**Implémentation** de **XSLT**





NB:

- JAXP 1.2.3 est livré avec le pack JWSDP de SUN
- Le répertoire endorsed est prévu pour être recopier globalement (avec son contenu) dans le répertoire **jre\lib** du ***JDK 1.4*** (==> plus de fonctionnalités , ex: validation via schéma W3C)
- Le fichier jaxp-api , sax.jar , xercesImpl.jar (ainsi qu'éventuellement dom.jar et xalan.jar) sont à ajouter au CLASSPATH d'un programme java interprété via le jdk 1.3 .
- Le jdk 1.5 (Java 5) comporte déjà Xerces en interne (rien a ajouter).

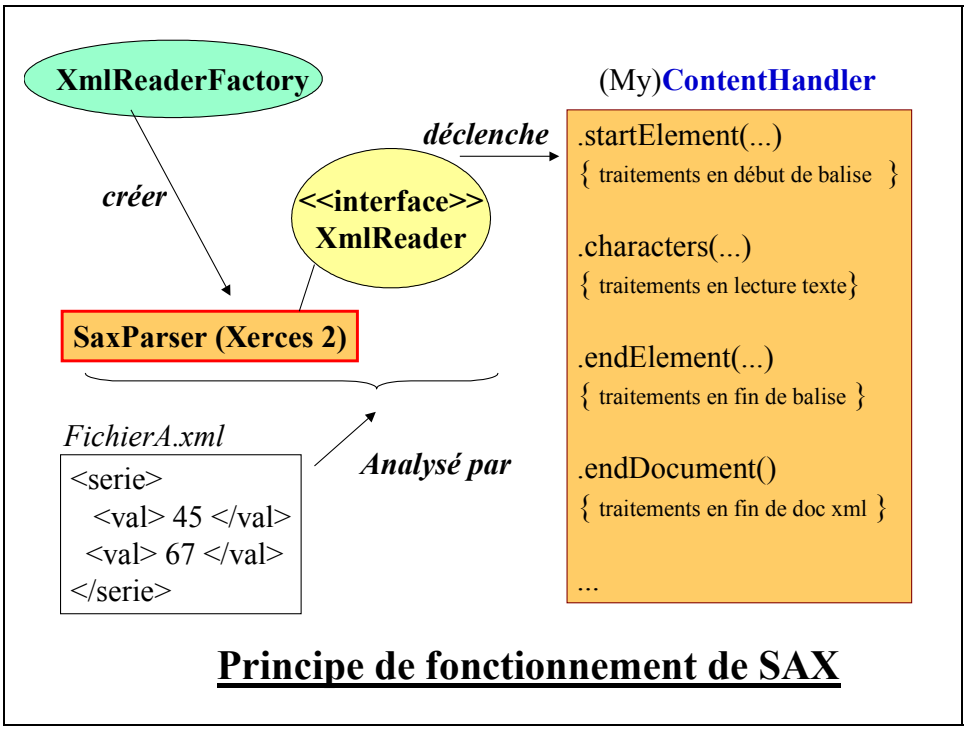
III - SAX

1. Présentation de SAX

SAX est une API simple et efficace provenant du monde Java qui effectue une **analyse au fil de l'eau** suivant le **modèle événementiel**. SAX est le fruit d'une concertation entre un grand nombre de développeurs qui ont communiqués via Internet. SAX est aujourd'hui un standard de fait.

Remarque: Microsoft a également incorporé SAX dans son propre parseur (MsXML) .

2. Modèle événementiel de SAX 2.0

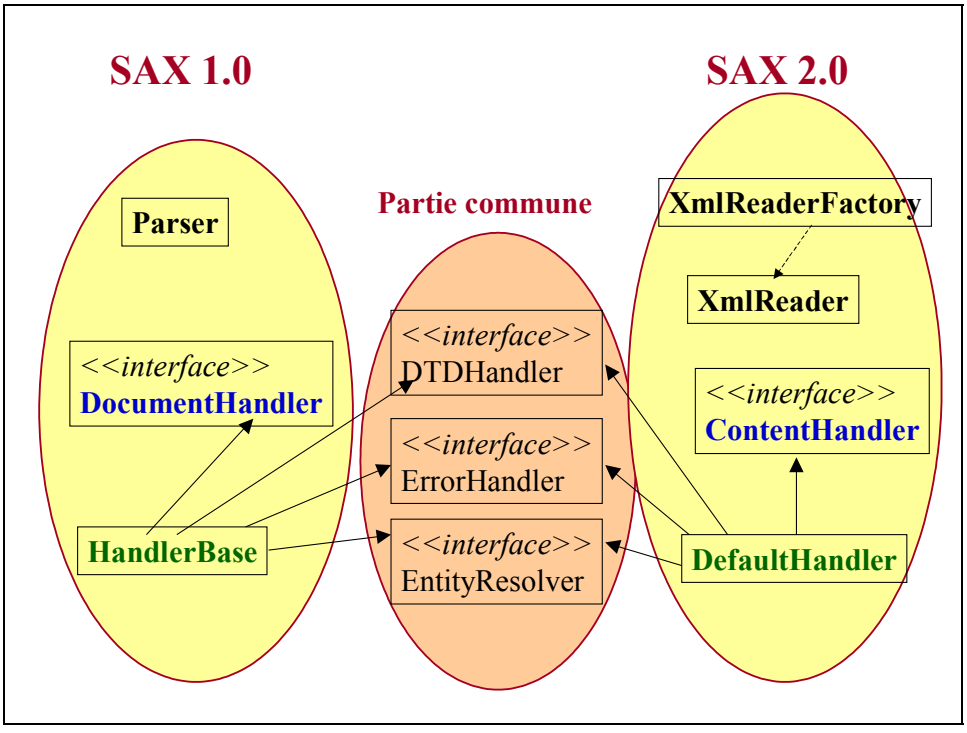


Principe de fonctionnement de SAX

Méthodes événementielles d'un gestionnaire SAX 2	
void	**characters**(char[] ch, int start, int length) *Receive notification of character data.*
void	**endDocument**() *Receive notification of the end of a document.*
void	**endElement**(java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName) *//Receive notification of the end of an element.*
void	**endPrefixMapping**(java.lang.String prefix) *End the scope of a prefix-URI mapping.*
void	**ignorableWhitespace**(char[] ch, int start, int length) *Receive notification of ignorable whitespace in element content.*
void	**processingInstruction**(java.lang.String target, java.lang.String data) *Receive notification of a processing instruction.*
void	**setDocumentLocator**(Locator locator) *Receive an object for locating the origin of SAX document events.*
void	**skippedEntity**(java.lang.String name) *Receive notification of a skipped entity.*
void	**startDocument**() *Receive notification of the beginning of a document.*
void	**startElement**(java.lang.String namespaceURI, java.lang.String localName, java.lang.String qName, Attributes atts)// *Receive notification of the beginning of an element.*
void	**startPrefixMapping**(java.lang.String prefix, java.lang.String uri) *Begin the scope of a prefix-URI Namespace mapping.*

3. Différences entre SAX 1.0 et SAX 2.0

SAX 1.0 ne supportait pas l'analyse les espaces de noms (namespace).
SAX 2.0 a introduit de nouvelles interfaces plus sophistiquées. Les principales améliorations se situent au niveau de la prise en compte des espaces de noms et de la validation via DTD.



Nb: pour coder rapidement un parseur basé sur SAX, on peut (au choix) utiliser une des deux méthodes de programmation suivantes:

- Reprogrammer entièrement une classe "gestionnaire" en y intégrant toutes les fonctions imposées par les interfaces **ContentHandler** (Sax 2.0) ou **DocumentHandler**(Sax 1.0).
- Programmer partiellement une classe "gestionnaire" en la faisant hériter de **DefaultHandler** (Sax 2.0) ou **HandlerBase** (Sax 1.0) et n'y intégrer que les méthodes événementielles ou du code spécifique est réellement utile.

Différence sur la gestion des attributs:

SAX 1.0 utilisait l' interface **AttributeList** pour parcourir la **liste des attributs d'une balise** tandis que SAX 2.0 utilise la nouvelle interface **Attributes**.

Ces deux interfaces comportent des fonctions de base communes:
getValue(String attrName), **getName**(int index) , getValue(int index) , **getLength**(), ...

L'interface **Attributes** (de SAX 2.0) comporte en plus certaines méthodes permettant de gérer les espaces de noms: **getLocalName**(int index) , **getQName**(int index) , ...

4. Déclenchement d'un "parsing" SAX 2.0 via XMLReaderFactory

NB: ***XMLReaderFactory*** est une fabrique du domaine public (*org.xml.sax.helpers*) qui n'est pas officiellement supporté par SUN/JAXP.

Cette fabrique tient compte de la propriété système ***org.xml.sax.driver*** **pour choisir la classe d'implémentation du parseur SAX (ex: Xerces 2).**

```
java -classpath .;../xerces.jar -Dorg.xml.sax.driver=org.apache.xerces.parsers.SAXParser tp.ValidApp
serieAvecDtd.xml    > ficRes.txt    2> fixErr.txt
```

Code de l'application qui déclenche l'analyse du fichier (en mode texte):

```
package tp;
import org.xml.sax.*;
import org.xml.sax.helpers.*; // XMLReaderFactory

public class ValidApp {
    public static void main(String[] args) {
        String xmlFileName = args[0];
        try {
                XMLReader myReader = XMLReaderFactory.createXMLReader();
                try {
                  myReader.setFeature("http://xml.org/sax/features/validation", true);
                    } catch (SAXException e) {
                            System.err.println("Cannot activate validation.");  }
                myReader.setContentHandler(    new AffDocHandler()  );
                myReader.parse(xmlFileName);
        } catch (Exception ex) {System.err.println(ex.getMessage());}
    }
}
```

5. Déclenchement d'un "parsing" SAX 2.0 via une fabrique de JAXP

```
package tp;
import org.xml.sax.*; // XMLReader , ...
import javax.xml.parsers.*; // JAXP (SAXParser, SAXParserFactory)

public class ValidApp {
    public static void main(String[] args) {
        String xmlFileName = args[0];
        try {
                SAXParserFactory factory = SAXParserFactory.newInstance();
                factory.setNamespaceAware(true);
                factory.setValidating(true);
                SAXParser saxParser = factory.newSAXParser();

                XMLReader myReader = saxParser.getXMLReader();

                myReader.setContentHandler(    new AffDocHandler()  );
                myReader.parse(xmlFileName);
        } catch (Exception ex) {System.err.println(ex.getMessage());}
    }
}
```

script de lancement du programme java (.bat):

```
set JAXP_1_2_2_LIBS=C:\...\lib\jaxp_1.2.3\endorsed\xercesImpl.jar
...
java -classpath %TP_BIN%;%JAXP_1_2_2_LIBS% tp.ValidApp serieAvecDtd.xml
```

6. Exemple de code - "ContentHandler" de SAX 2.0

Code de la classe correspondant au gestionnaire de contenu:

```
package tp;
import java.util.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*; // DefaultHandler

public class AffDocHandler extends DefaultHandler{
        private StringBuffer buf = new StringBuffer();
        private HashMap dico = new HashMap();

        public void startElement(String nameSpaceUri, String localName,
                                 String qName, Attributes atts)
        {
        String chTag = "[" + nameSpaceUri + "]" +localName;
        Integer cptObj = (Integer) dico.get(chTag);
        if(cptObj==null) cptObj = new Integer(1);
          else cptObj = new Integer(cptObj.intValue() + 1);
           dico.put(chTag,cptObj);
           System.out.print("<" +  qName + ">");
        }
```

```
        public void characters(char[] ch, int start,int length)
        {
        buf.append(ch,start,length);
        }

        public void endElement(String nameSpaceUri, String localName,String qName)
        {
        String chVal = buf.toString();
        if(chVal == null) chVal ="";
        System.out.println(chVal + "</" + qName + ">");
        buf.setLength(0);
        }

        public void endDocument()
        {
        System.out.println("***** Fin du document XML ******");
        System.out.println("Balises rencontrées:");
        Iterator it  = dico.keySet().iterator();
        while(it.hasNext())
            {
            String chTag = (String) it.next();
            Integer cptObj = (Integer) dico.get(chTag);
            System.out.println(chTag + " --- " + cptObj.toString() );
            }
        }

        public void startDocument()
        {
        System.out.println("***** Début du document XML ******");
        }
}
```

7. Comment se situer % au(x) fichier(s) analysés ?

L'interface **Locator** (du package org.xml.sax) comporte les méthodes suivantes:

- int **getLineNumber**() --> *retourne le numéro de ligne (lors de l'événement)*
- int **getColumnNumber**() --> *retourne le numéro de colonne (lors de l'événement)*
- String **getPublicId**() --> *retourne le FPI du fichier courant*
- String **getSystemId**() --> *retourne l'URL du fichier courant*

Portée (utilisation) de l'objet "Locator" :

```
class MyHandler extends DefaultHandler
{
Locator loc;

public void setDocumentLocator(Locator loc)
     { this.loc = loc; }

public void startElement(...)
    { System.out.println("Num Ligne = " + loc.getLineNumber() ); }
...
}
```

Remarques:

- **L'objet "Locator" doit être utilisé tout de suite** car ses valeurs sont régulièrement remises à jour (lors que chaque événement lié au parsing).
- Pour mémoriser (recopier) les valeurs d'un objet "Locator" on peut utiliser la classe **LocatorImpl** du package ***org.xml.sax.helper*** .
- La classe d'exception **SAXParseException** comporte également les méthodes **getLineNumber**() et **getColumnNumber**() .
- L'interface **EntityResolver** ne sert qu'a aider le programme à associer des noms logiques de Ressources publiques (FPI) avec des URL menant vers des fichiers physiques. Pour cela , la seule méthode ***resolveEntity***() de l'interface peut par exemple se baser sur une table d'association quelconque (fichier catalogue).

8. Gestion des erreurs

L'interface **ErrorHandler** (du package org.xml.sax) comporte les méthodes suivantes:

- void **error**(SAXParseException e) throws SAXException
  Cette méthode événementielle est déclenchée suite à l'analyse d'un document non valide (non conforme à la structure imposée par une DTD, ...).

- void **fatalError**(SAXParseException e) throws SAXException
  Cette méthode est déclenchée suite à une erreur de syntaxe xml découverte au niveau du fichier à analyser (document Xml qui n'est pas bien formé).

- void **warning**(SAXParseException e) throws SAXException
  Cette méthode est déclenchée suite à un petit problème de parsing qui n'entre pas dans les 2 catégories précédentes.

Nb1: SAXParseException est une sous classe de SAXException.

Nb2: Sans gestionnaire d'erreur personnalisé le parseur Sax 2.0 (avec le comportement par défaut de DefaultHandler) soulève une exception de type **SAXException** en cas d'erreur fatale ou bien en cas d'erreur de parsing (validation, ...).La grosse différence entre une ***fatalError*** et une ***Error*** réside dans le fait qu'une ***fatalError*** mais fin au parsing tandis qu'une ***Error*** (de validation) ne fait que remonter une exception de type SAXParseException que le programme appelant est libre de traiter ou d'ignorer.

Nb3: Les méthodes événementielles de l'interface ErrorHandler (que l'on programme souvent dans une sous classe de DefaultHandler) sont essentiellement utiles pour **écrire les messages d'erreur dans un fichier de log**.

```
try {...
                myReader.setErrorHandler(new MyErrorHandler());
                myReader.parse(xmlFileName);
        } catch (SAXException e) {          System.err.println(e.getMessage());  } ...
```

```
public class MyErrorHandler implements ErrorHandler {
...
public void error(SAXParseException exception) throws SAXException
```

```
            {
            //exception.printStackTrace();
            //System.exit(1);
            throw exception;
           }
}
```

9. Gestion des DTD depuis SAX 2.0

Depuis la version 2.0 de SAX , **l'activation de la validation** (via DTD) s'effectue en affectant la valeur **true** à la fonctionnalité "**http://xml.org/sax/features/validation**" comme le montre l'exemple suivant:

```
XMLReader myReader = XMLReaderFactory.createXMLReader();
            try {
              myReader.setFeature("http://xml.org/sax/features/validation", true);
               } catch (SAXException e) {
                        System.err.println("Cannot activate validation.");  }
```

ou bien

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
factory.setValidating(true);
```

Le parseur fonctionnera alors en mode **validant** et remontera des erreurs dans le cas où le document Xml n'est pas conforme à la structure type imposée par la DTD.

D'après le paragraphe précédent (sur le traitement des erreurs), il est conseillé de traiter convenablement les exceptions de type SAXParseException (héritant de SAXException).

Attention:

- En mode **non validant**, le Parseur SAX appelle **characters**() lorsqu'il rencontre le moindre **caractère blanc** dans le fichier xml source (à analysé).
- En **mode validant** (avec DTD), la rencontre d'un **caractère blanc inutile** déclenche l'appel automatique de la fonction **ignorableWhitespace**()

10. Validation via XMLSchema avec une version récente de JAXP

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
factory.setValidating(true);
SAXParser saxParser = factory.newSAXParser();
try { saxParser.setProperty(
       "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
       "http://www.w3.org/2001/XMLSchema");
    }
catch (SAXNotRecognizedException x) { x.printStackTrace();
  // Happens if the parser ( < Xerces2 ) does not support JAXP 1.2   ...
}
```

IV - DOM (Document Object Model)

1. API DOM – Modèle Objet

1.1. Présentation

DOM signifie ***Document Object Model*** .

Cette **API** normalisée par l'organisme **W3C** constitue un véritable **standard** .

1.1.a. Universalité - versions dans quasiment tout langage

Reposant sur un modèle orienté objet indépendant de tout langage de programmation, l' API **DOM** *(initialement spécifiée via le langage neutre IDL de Corba)* est utilisable dans un très grand nombre de langages informatiques:

Java , C++ , JavaScript , VBScript , PHP , ...

Seules diffèrent quelques adaptations liées à certains langages:

.**documentElement** en *JavaScript*

.**getDocumentElement**() en *Java*

1.1.b. Xml-DOM et DOM niveaux 1,2 et 3 pour XHTML

Le coeur de l'api DOM (**Xml-DOM**) permet de traiter n'importe quel fichier XML.

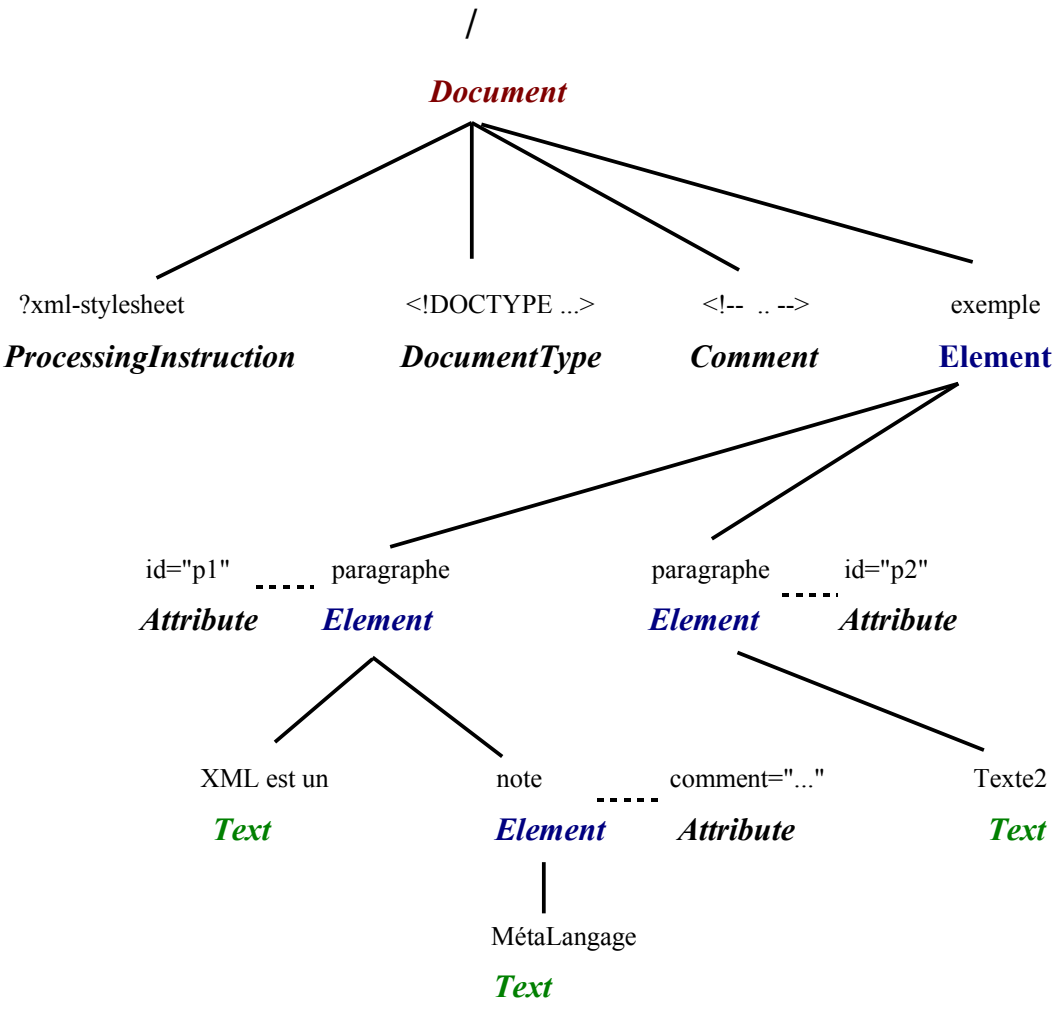
Des extensions liées au langage (X)HTML (considéré comme un cas particulier de XML) permettent d'écrire des petits morceaux de scripts *(ex: JavaScript)* qui vont pouvoir opérer sur la structure d'un document (ex: changement dynamique de style , développement / contraction d'une zone, permutation d'images , autres animations ou effets dynamiques, ...).

1.2. Modèle d'arbre en mémoire

Le fichier xml suivant

```
<?xml version="1.0" ?>
<?xml-stylesheet href="./style1.css" type="text/css" ?>
<!DOCTYPE  exemple
    [
      <!ENTITY  eacute  "&#x00E9;" >
    ]>
<!- - commentaire -->
<exemple>
    <paragraphe  id="p1">XML est un
                          <note comment="important"> M&eacute;taLangage</note>
    </paragraphe>
    <paragraphe  id="p2">texte2</paragraphe>
</exemple>
```

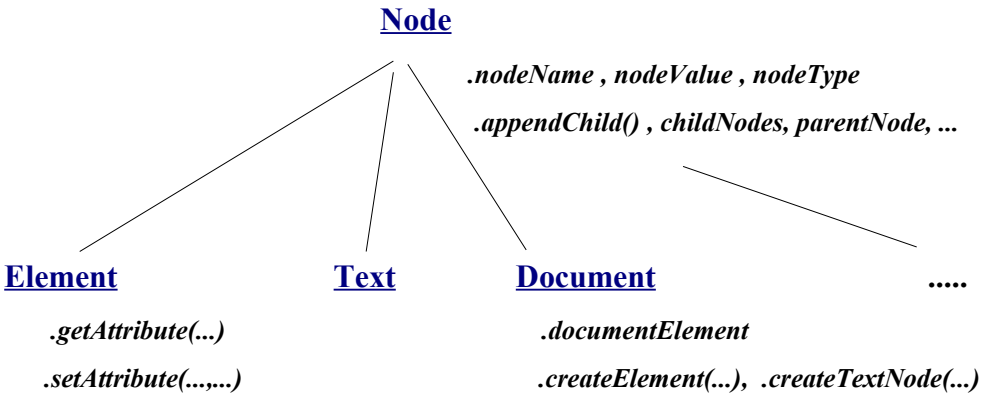
est représenté via un arbre DOM de ce type:



Dans l'arbre précédent, les noeuds ne sont pas tous du même type:

- Le noeud racine est de type **Document**
- Les noeuds liés aux balises sont de type **Element**
- Les noeuds comportant un morceau de texte sont de type **Text**

Cependant , ces différents types de noeuds héritent tous d'un type générique : "**Node**" .



Certaines fonctions (associées au type générique ***Node***) sont accessibles depuis n'importe quel noeud :

- ***nodeName*** retourne le nom d'une balise ou null .
- ***nodeValue*** retourne la valeur d'un texte ou null .
- ***nodeType*** retourne une constante indiquant le type de noeud (ELEMENT_NODE , TEXT_NODE, ...)
- **childNodes** retourne une liste de noeuds fils sous la forme d'un objet ensembliste de type **NodeList** .
- **parentNode** retourne une référence sur le noeud père

D'autres fonctions ne sont disponibles que sur certains types précis de noeuds:

- La fonction **getDocumentElement()** que l'on appelle sur le noeud racine du document retourne **l'unique noeud de type Element qui correspond à la balise de premier niveau .**
- Seul le noeud racine (de type *Document*) comporte des fonctions [***createElement***("nombalise") , ***createTextNode***("valeurTexte")] permettant de créer de nouveaux noeuds qui devront ultérieurement être accrochés sous un des noeuds de l'arbre via la méthode ***appendChild***() .
- Les méthodes ***setAttribute***("nomAttribut","valeur") et ***getAttribute***("nomAttribut") doivent être appelée sur un noeud de type *Element* .

1.3. L'interface centrale Node.

La plupart des autres interfaces de l'api héritent de ***Node*** (du package ***org.w3c.dom***) .

```
interface Node {
// valeurs pour nodeType (constantes):
public static final short      DOCUMENT_NODE                =1;
public static final short      ELEMENT_NODE                 =2;
public static final short      ATTRIBUTE_NODE               =3;
public static final short      PROCESSING_INSTRUCTION_NODE  =4;
public static final short      COMMENT_NODE                 =5;
public static final short      TEXT_NODE                    =6;
public static final short      CDATA_SECTION_NODE           =7;
public static final short      DOCUMENT_FRAGMENT_NODE       =8;
public static final short      ENTITY_NODE                  =9;
public static final short      ENTITY_REFERENCE_NODE        =10;
public static final short      DOCUMENT_TYPE        _NODE           =11;

// (propriétés):
public String          getNodeName();
public String          getNodeValue();  public void setNodeValue(String valeur);
public short           getNodeType();
public Node            getParentNode();
public NodeList        getChildNodes();
public Node            getFirstChild();
public Node            getLastChild();
public Node            getPreviousSibling();
public Node            getNextSibling();
public NameNodeMap     getAttributes();

//Méthodes "public":
Node          insertBefore(Node NewChild,Node refChild) throws DOMException;
Node          replaceChild(Node NewChild,Node oldChild) throws DOMException;
Node          removeChild(Node oldChild) throws DOMException;
Node          appendChild(Node newChild) throws DOMException;
boolean       hasChildNode();
Node          cloneNode(boolean deep);
boolean       equals(Node arg, boolean deep);
};
```

1.4. Description rapide des autres interfaces:

L'interface **Document** représente la racine d'un document complet. **Un noeud Document ne peut avoir qu'un seul fils de type Element (l'élément racine du document).**
Ce **noeud fils racine** peut être récupéré via la propriété get**DocumentElement**() .
L'interface Document comporte un ensemble de fonctions permettant de créer un nouveau noeud dans le contexte du document courant (**createElement**("*NomBalise*"), **createTextNode**("*Texte*"), ...).

Les interfaces **Element**, **DocumentType**, **EntityReference**, **Entity**, **Text**, **CDATASection** correspondent aux choses du document XML.

L'interface **Attr(ibute)** correspond à un attribut. *Les noeuds de type Attr(ibute) ne sont pas directement placés sous un noeud de type Element*. Les différents attributs d'un noeuds sont accessibles depuis la collection attributes d'un élément.

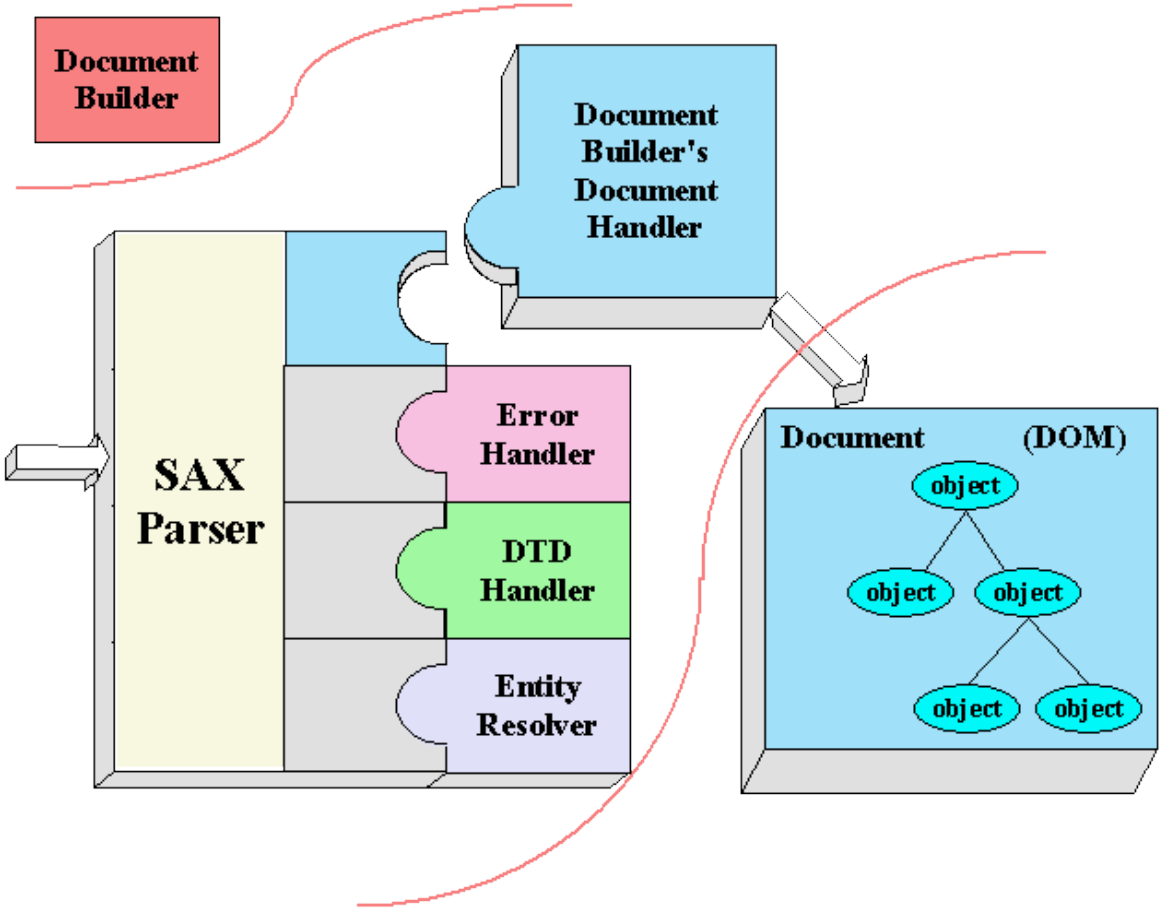
L'interface **NodeList** représente un ensemble ordonné de noeuds.
Sa propriété get**Length**() retourne le nombre d'éléments (pour boucle for de 0 à n-1).
Sa méthode **item(i)** retourne le i éme noeud de la liste.

L'interface **NamedNodeMap** correspond à une collection d'objets directement accessibles via leurs noms. En plus de length et item(i), cette interface comporte les méthodes **getNamedItem**(...), **setNamedItem**(...) et **removeNamedItem**(...) .

Finalement , l'interface **DocumentFragment** correspond à un sous arbre XML.
Cette interface peut s'avérer utile pour effectuer des "copier/coller" .

2. DOM & JAXP

NB: Le parser XML - DOM de Java s'appuie en interne sur SAX.



3. Déclenchement du parsing

3.1. Déclenchement du parsing avec Jaxp

java.io.**InputStream in = (InputStream) ...** *// flux de lecture depuis fichier ou (http)*
javax.xml.parsers.**DocumentBuilderFactory** docBuilderFactory =
DocumentBuilderFactory.newInstance();
javax.xml.parsers.**DocumentBuilder** docBuilder =
docBuilderFactory.newDocumentBuilder();

org.w3c.dom.**Document docXml = docBuilder.parse (in);**
// la méthode parse est surchargée , une version existe avec un "String fileName" en entrée.

4. Analyse de l'arbre généré

Analyser l'arbre généré revient souvent à parcourir l'arbre de façon à récupérer les informations

intéressantes.

Le point de départ du parcours correspond très souvent au noeud de type "Element" qui se trouve sous la racine de l'arbre DOM et qui correspond à l'unique balise de premier niveau (celle qui englobe tout le contenu du document Xml):
Pour accéder rapidement à ce noeud, il suffit de se référer à la propriété **documentElement** de l'arbre:

Node docElement = docXml.**getDocumentElement**(); *// avec Java / Jaxp*

Il faut ensuite parcourir les noeuds fils :

```
NodeList listeNoeudFils = docElement.getChildNodes();
for(int i=0;i<listeNoeudFils.getLength();i++)
    {
    Node childNode = listeNoeudFils.item(i);
     ...
    }
```

Pour chaque noeud fils rencontré, il est primordial de vérifier son type avant d'appeler dessus des méthodes quelquefois invalides:

```
if(childNode.getNodeType() == Node.ELEMENT_NODE  )
   {
   // ce noeud est de type ELEMENT
   if(childNode.getNodeName().equals("val"))
        {
     Node noeudFils = childNode.getFirstChild();
    if(noeudFils!=null)
             if(noeudFils.getNodeType()==Node.TEXT_NODE)
                chaine = noeudFils.getNodeValue();
     }
   }
```

Nb: La méthode **normalize**() que l'on peut éventuellement appeler sur la racine d'un sous arbre DOM permet d'être sûr de ne pas trouver des noeuds textes adjacents.

5. Fabrication de nouveaux nœuds

L'interface **Document** (qui correspond à la racine de l'arbre DOM) comporte un ensemble de fonctions permettant de créer un nouveau noeud dans le contexte du document courant:

1. Création d'un nouveau noeud (pas encore rattaché):
   *Node noeud = doc.**createElement**("NomBalise");*
   ou bien
   *Node noeud = doc.**createTextNode**("Texte");*

2. Rattachement du nouveau noeud en dessous d'un des noeuds de l'arbre:
   *noeudPere.**appendChild**(noeud);*

Nb: si l'arbre DOM n'existe pas encore, on peut en créer un nouveau (vide au début):

Document docXml = **docBuilder.newDocument();** ***// avec JAXP***

6. Génération d'un fichier XML (ou flux réseau).

6.1. Sérialisation (vers OutputStream) avec JAXP

- **JAXP 1.1** s'appuie en interne sur **xalan** (moteur de transformation XSLT du monde Apache) et sur **crimson** (parseur XML d'origine Sun et repris par Apache).
  Le package **org.apache.crimson.tree** de la version **1.1** correspond à l'ancien package **com.sun.xml.tree** de la version 1.0 :
  Ainsi via un simple casting (avec crimson) on peut manipuler un objet **Document** comme s'il s'agissait en fait d'une instance de **XmlDocument** de façon à pouvoir invoquer la méthode **write** (bien pratique pour écrire dans un flux le texte du document XML correspondant à l'arbre en mémoire):
  *XmlDocument wDoc = (XmlDocument) doc; wDoc.**write**(out);*

- **JAXP 1.2** s'appuie en interne sur Xalan et **Xerces 2 Java.**
  Dans cette implémentation, la classe **XmlDocument** n'existe plus. Il faut utiliser la classe ***org.apache.xml.serialize.*XMLSerializer** (spécifique au parseur Xerces 2) pour obtenir le même résultat:
  XMLSerializer xmlSer = new XMLSerializer();
  *xmlSer.**setOutputByteStream**(new FileOutputStream("f1.xml"));*
  *xmlSer.**serialize**(doc);*

Pour générer un fichier Xml à partir d'un arbre DOM en mémoire , la solution qui est la moins dépendante du parseur consiste à utiliser TrAx:

```
import javax.xml.transform.*;// TransformerFactory , Transformer
import javax.xml.transform.dom.DOMSource;
   import javax.xml.transform.stream.StreamResult;
   ...
   TransformerFactory trFactory = TransformerFactory.newInstance();
   Transformer tr = trFactory.newTransformer();
   tr.transform(new DOMSource(doc),
                new StreamResult(new FileOutputStream("f2.xml")));
```

7. Parseur validant

La méthode **setValidating(*true* /* *false par défaut*/)** de la classe **DocumentBuilderFactory** permet de **créer un parseur validant**.

On peut également spécifier son propre gestionnaire d'erreur (de type Sax):
builder.**setErrorHandler**(new MySpecificErrorHandler());

NB: en cas d'erreur de parsing, l'exception générée est de type **SAXException**.

==> voir le chapitre sur SAX pour approfondir la question.

8. Tenir compte des nameSpaces.

La méthode **setNamespaceAware(*true* /* *false par défaut* */)** de la classe **DocumentBuilderFactory** permet de créer un parseur prenant en compte les informations sur les espaces de noms.

Les interfaces **Node** et **Attr** (héritant bizarrement de Node) comportent les méthodes **getLocalName**() , **getNamespaceUri**() permettant de récupérer les informations liées aux espaces de noms.

9. Parcours rapides

9.1. Accès direct aux nœuds d'un certain type

NodeList listeNoeuds = rootElement.**getElementsByTagName**(String tagName);
+ *boucle for*

D'autre part , **si un noeud comporte un identificateur** (attribut de type **ID** ou **xsd:ID**) alors la méthode **getElementById**() permet de le retrouver directement:

```
<customer id="theClient">client</customer>
```

```
Node  n = xmlDoc.getElementById("theClient");
     if(n!=null) System.out.println("Le noeud dont l'id est theClient
                           est du type " + n.getNodeName());
```

10. Classe utilitaire spécifique à l'implémentation Xerces2

La classe **org.apache.xerces.util.DOMUtil** comporte une multitude de méthodes statiques qui peuvent s'avérer fort pratiques pour simplifier l'accès aux différentes parties de l'arbre DOM.

Exemples:

Element noeudBalise = **DOMUtil.getFirstChildElement**(noeudSommetBranche,"nom balise recherchée");

```
String texte = DOMUtil.getChildText(noeudBaliseComportantSousNoeudText);
...
```

11. Quelques exemples de code (DOM /JAXP)

11.1. Exemple1: génération d'un fichier xml

```
package tpdom;
import javax.xml.parsers.*;

// Utilisation de TrAX / Xalan pour écrire un arbre Dom dans un OutputStream:
import javax.xml.transform.*;// TransformerFactory , Transformer
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.*;
import java.io.*;

public class GenXml {
public static void main(String[] args) {
try {
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance ();
  DocumentBuilder db = dbf.newDocumentBuilder ();
  Document doc = null;
  doc = db.newDocument ();
  Element root = doc.createElement ("order");
  doc.appendChild(root);
  Element customer = doc.createElement("customer");
  customer.appendChild(doc.createTextNode("client"));
  root.appendChild(customer);
  Element item = doc.createElement("item");
  root.appendChild(item);
  Element ref = doc.createElement("ref");
  ref.appendChild(doc.createTextNode("1"));
  item.appendChild(ref);
  Element qty = doc.createElement("qty");
  qty.appendChild(doc.createTextNode("5"));
  item.appendChild(qty);

  //Ecriture d'un arbre DOM dans un flux (fichier)
  //avec Jaxp 1.2 et l'api normalisée TrAx (Xalan):
  TransformerFactory trFactory = TransformerFactory.newInstance();
  Transformer tr = trFactory.newTransformer();
  tr.transform(new DOMSource(doc),
            new StreamResult(
                new FileOutputStream("f2.xml")
             ));
}// end of try
catch(Exception ex) { ex.printStackTrace(); }
}// end of main()
}
```

11.2. Exemple2: parsing & analyse d'un arbre (Jaxp 1.2):

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

public class AnalyseXml {

        int quantite = 0;
        String clientName = "Inconnu";
        String refArticle = "?";

        public static void main(String[] args) {
        AnalyseXml obj = new AnalyseXml();
        obj.analyser(args[0]);
        }

        void analyser(String fileName)
        {
        try {
        /* Obtenir un accès à la factory: */
        DocumentBuilderFactory DocBuilderFactory =
                        DocumentBuilderFactory.newInstance();

        //DocBuilderFactory.setValidating(true);

        /* Fabriquer un parseur DOM: */
        DocumentBuilder docBuilder = DocBuilderFactory.newDocumentBuilder();

        /* Déclencher le parsing et récupérer une référence sur l'arbre DOM: */
        Document xmlDoc = docBuilder.parse(fileName);

        Element docElement = xmlDoc.getDocumentElement();

        NodeList listeNoeudFils = docElement.getChildNodes();
        if(listeNoeudFils!=null)
          for(int i=0;i<listeNoeudFils.getLength();i++)
            {
            Node childNode = listeNoeudFils.item(i);
            if(childNode.getNodeType()==Node.ELEMENT_NODE)
              {
              if(childNode.getNodeName().equals("customer"))
                  recupererNomDuClient(childNode); // sous fonction
               if(childNode.getNodeName().equals("item"))
                  recupererDetailsArticle(childNode); // sous fonction
              }
            }

        // Afficher le résultat de l'analyse:
        System.out.println("Nom du client : " + clientName);
```

```
        System.out.println("Ref de l'article: " + refArticle);
        System.out.println("Quantité : " + quantite);
        }
        catch(Exception ex)
        {
        ex.printStackTrace();
        }
        }


        void recupererNomDuClient(Node customerNode)
        {
        Node noeudFils = customerNode.getFirstChild();
        if(noeudFils!=null)
          if(noeudFils.getNodeType()==Node.TEXT_NODE)
             this.clientName = noeudFils.getNodeValue();
        }

        void recupererDetailsArticle(Node itemNode)
        {
        NodeList listeNoeudFils = itemNode.getChildNodes();
        if(listeNoeudFils!=null)
          for(int i=0;i<listeNoeudFils.getLength();i++)
            {
            Node childNode = listeNoeudFils.item(i);
            if(childNode.getNodeType()==Node.ELEMENT_NODE)
               {
               if(childNode.getNodeName().equals("ref"))
                   {
                   Node noeudFils = childNode.getFirstChild();
                          if(noeudFils!=null)
                                 if(noeudFils.getNodeType()==Node.TEXT_NODE)
                                        this.refArticle = noeudFils.getNodeValue();
                   }

                if(childNode.getNodeName().equals("qty"))
                   {
                   Node noeudFils = childNode.getFirstChild();
                   if(noeudFils!=null)
                     if(noeudFils.getNodeType()==Node.TEXT_NODE)
                          this.quantite = Integer.parseInt(noeudFils.getNodeValue());
                   }
               }
            }
        }
}
```

12. Validation d'un document xml via Xml-DOM et un schéma w3c

Il peut être astucieux de définir ces quelques constantes:

```
static final String JAXP_SCHEMA_LANGUAGE =
                    "http://java.sun.com/xml/jaxp/properties/schemaLanguage";

static final String W3C_XML_SCHEMA = "http://www.w3.org/2001/XMLSchema";
```

Il faut paramétrer l'usine à parseur de façon à ce quelle fabrique un parseur validant basé sur un schéma w3c .

```
...
  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance()
  factory.setNamespaceAware(true);
  factory.setValidating(true);
try {
  factory.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
}
catch (IllegalArgumentException x) {
  // Cas de figure où  le  parseur ne supporte pas les spécificités JAXP 1.2
  ...
}
```

Le document xml peut éventuellement faire référence au schéma permettant de le valider:

```
<documentRoot
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation='YourSchemaDefinition.xsd'
>
  ...
```

Si cette référence est inexistante, c'est au programme de préciser le schéma à utiliser pour effectuer la validation:

```
static final String schemaSource = "YourSchemaDefinition.xsd";
static final String JAXP_SCHEMA_SOURCE =
                                "http://java.sun.com/xml/jaxp/properties/schemaSource";
...
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance()
...
factory.setAttribute(JAXP_SCHEMA_SOURCE, new File(schemaSource));
```

Nb: La documentation complète sur jaxp montre comment éventuellement utiliser plusieurs schémas associés à plusieurs namespaces xml pour les documents composés (ex: xhtml + svg + mathml) . .

13. Présentation de JDOM

JDOM est une version (non encore normalisée) mais **optimisée pour JAVA** de l'api **DOM**.

JDOM sera peut être un jour une api officielle du monde java (suite à la demande de normalisation **JSR 102** datant de février 2001).

JDOM utilise directement les spécificités du langage Java (Collections, ...) et consiste en une api beaucoup plus simple et intuitive que DOM.

Contrairement à DOM, **JDOM** n'est pas qu'un simple jeu d'interfaces java. **Il s'agit en fait d'un vrai parseur complètement autonome se voulant léger (ayant une faible consommation mémoire) et efficace (relativement rapide)**.

Pour des raisons de compatibilité, l'api JDOM peut inter-opérer avec les API standards SAX et DOM. Néanmoins le parseur interne de JDOM ne peut pas être utilisé à la place de Xerces ou Crimson sous les interfaces de JAXP.

La version ***1.0 de JDOM*** est accessible depuis l'url suivante: **www.jdom.org** .

14. Présentation de DOM4J

Dom4J est une autre API qui se veut être une optimisation "Java" de l'api DOM .

Dom4J est aujourd'hui une api "**Open Source**".

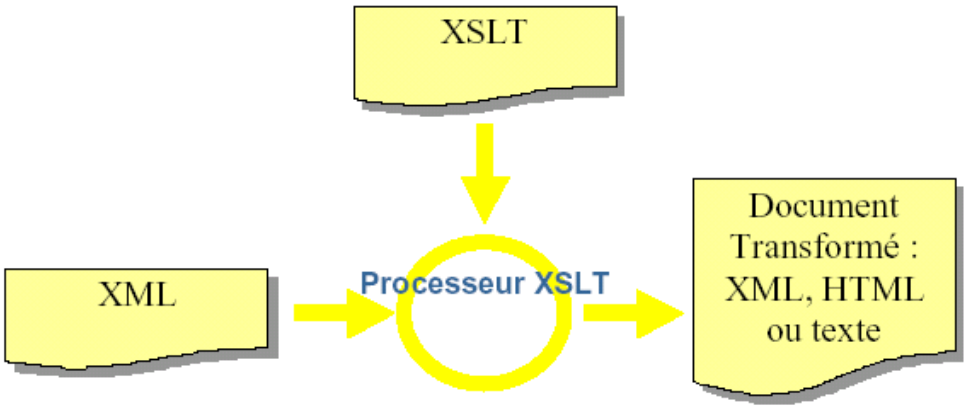
URL pour téléchargement: **`http://dom4j.org/`**

Quelques caractéristiques de **Dom4J** :

- Conçu pour la plate-forme Java avec un support complet des Collections (java.util)
- Support complet de JAXP, TrAX, SAX, DOM, and XSLT
- Intégration de XPath permettant une navigation simple à travers les documents XML.
- Comporte un mode de traitement basé sur des événements pour gérer des gros documents ou bien des flux continus
- Basé sur des interfaces Java pour offrir une flexibilité des implémentations.
- Validation possible via des Schémas XML .

==> Etudier les exemples de la partie "**Quick Start**" pour appréhender rapidement les caractéristiques de **Dom4j** .

V - TrAX



Seuls les navigateurs IE 5.5 et IE 6 sont actuellement capables de déclencher des transformations xslt coté client. De façon à ce que la mise en forme via XSLT puisse être vue par un navigateur internet quelconque (Netscape, Mozilla, Opera, ...) on a tout intérêt à déclencher cette transformation coté serveur.

1. Activation d'une transformation XSLT via Java et l'api xalan :

L'api **Xalan** peut être téléchargée depuis le site **www.apache.org** .

```
import org.xml.sax.SAXException;
import org.apache.xalan.xslt.*;

public class SimpleTransform
{
public static void main(String[] args) throws java.io.IOException,
java.net.MalformedURLException, org.xml.sax.SAXException
        {
XSLTProcessor processor = XSLTProcessorFactory.getProcessor();

processor.process(new XSLTInputSource("foo.xml"),
                  new XSLTInputSource("foo.xsl"),
                  new XSLTResultTarget(System.out));

/* NB: Ce Code est écrit avec une ancienne version de xalan */
        }
}
```

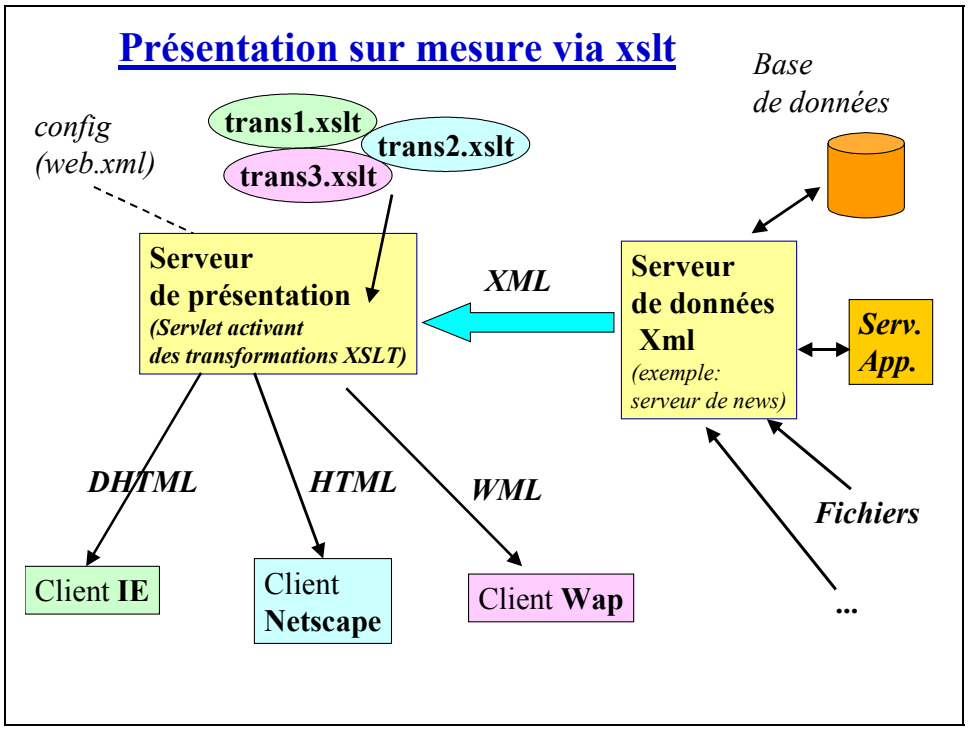
2. Transformation déclenchée via TrAX de JAXP

NB: Une transformation XSLT (effectuée par xalan) peut être indirectement déclenchée par **TrAX** de ***JAXP*** :

```
package tpxslt;

import javax.xml.transform.*;      // Imported TraX classes
import javax.xml.transform.stream.*;
import java.io.*;

public class TransformApp {

public static void main(String[] args) {

        if(args.length < 3)
                { System.err.println("usage:
                        java tpxslt.TransformApp ficSrc.xml ficXslt.xsl ficRes.html/xml ");
                 System.exit(1); }
        try{
                String ficSource=args[0];
                String ficXslt=args[1];
                String ficRes=args[2];

            TransformerFactory tFactory =TransformerFactory.newInstance();

            Transformer transformer = tFactory.newTransformer(
                                        new StreamSource( new FileInputStream(ficXslt)));

            transformer.transform(new StreamSource(new FileInputStream(ficSource)),
                                        new StreamResult(ficRes));

           } catch(Exception ex) { ex.printStackTrace(); }

        }
}
```

```
java -classpath %TP_BIN%
        tpxslt.TransformApp     news.xml      pres3.xslt      news.html
```

3. Dialogue entre serveur de présentation et serveur de données.



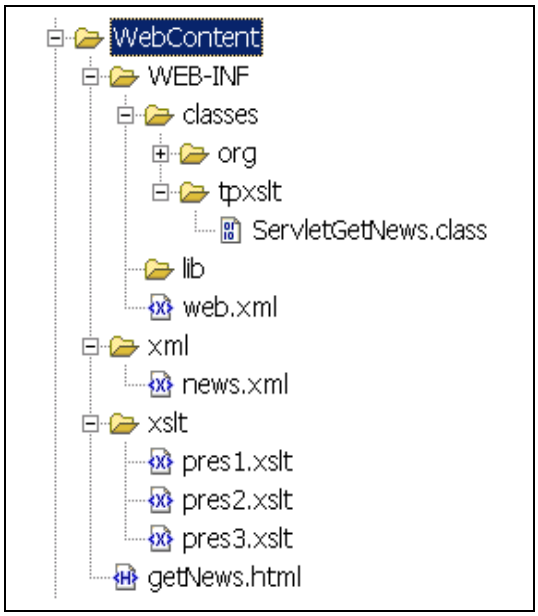
NB:

Attention aux performances *(3 arbres : Arbre_xslt , Arbre_Source, Arbre_Res)* !!!!

==>
Générer une bonne fois pour toute une page HTML via une transformation XSLT.
ou
Mettre en place un système de cache.

4. Mise en œuvre coté serveur (Servlet, pages JSP,...).

```
package tpxslt;
import javax.xml.transform.*;       // Imported TraX classes
import javax.xml.transform.stream.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletGetNews extends HttpServlet
{
     public void service(HttpServletRequest request,
                         HttpServletResponse response)
    throws ServletException, IOException
     {
     String chFicSource=
              getServletConfig().getInitParameter("source_xml");
     chFicSource=getServletContext().getRealPath(chFicSource);
     String chSubDirXslt=
              getServletConfig().getInitParameter("xslt_subdir");
     String chFicXslt=request.getParameter("style");
     chFicXslt=getServletContext().getRealPath(
                                          chSubDirXslt+"/"+chFicXslt );
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     try {
      TransformerFactory tFactory = TransformerFactory.newInstance();

       Transformer transformer =
           tFactory.newTransformer(new StreamSource(
                        new FileInputStream(chFicXslt)));

       transformer.transform(
                new StreamSource(new FileInputStream(chFicSource)),
               new StreamResult(out));

         } catch(Exception ex) { ex.printStackTrace(out); }
     }
}
```



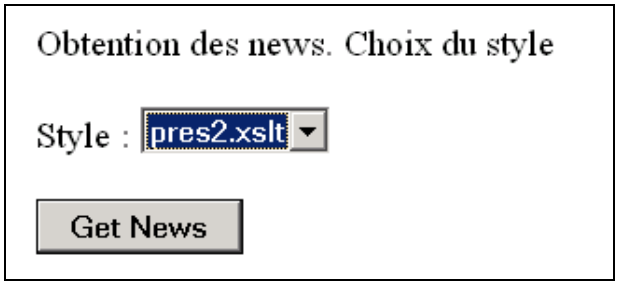
WEB-INF/web.xml

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

```
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
    <servlet>
        <servlet-name>ServletGetNews</servlet-name>
        <servlet-class>tpxslt.ServletGetNews</servlet-class>
        <init-param>
             <param-name>source_xml</param-name>
             <param-value>xml/news.xml</param-value>
     </init-param>
     <init-param>
         <param-name>xslt_subdir</param-name>
            <param-value>xslt</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>ServletGetNews</servlet-name>
        <url-pattern>/GetNewsServlet</url-pattern>
    </servlet-mapping>

</web-app>
```

getNews.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
   <HEAD>
      <META name="GENERATOR" content="Microsoft FrontPage 4.0">
      <TITLE>getNews.html</TITLE>
   </HEAD>
   <BODY>Obtention des news. Choix du style<BR>
<BR>
<FORM action="GetNewsServlet" method="get">
<P>Style : <select name="style" size="1" >
           <option>pres1.xslt</option>
           <option>pres2.xslt</option>
           <option>pres3.xslt</option>
         </select></P>
<P><input type="submit" value="Get News"/> </P>
</FORM>
</BODY>
</HTML>
```



news.xml

```
<?xml  version="1.0" ?>
<!-- <?xml-stylesheet  href="./pres1.xslt" type="text/xsl" ?> -->
```

```
<news-list>
  <news date="20/12/2002">
    <title>Bientot Noel !!!</title>
    <auteur>Père Noël</auteur>
    <text>cadeaux uniquement prévus pour les enfants sages.</text>
  </news>
  <news date="20/06/2003">
    <title>Attention , danger !</title>
    <auteur>OufADonf</auteur>
    <text>Des transformations XSLT erronées peuvent produire des
mutants.</text>
  </news>
</news-list>
```

pres2.xslt

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
    <html>
      <head>
      <title> Page HTML generee depuis XML via XSLT </title>
      <style>
          h3 {color=red; }
           td {color=black; }
           th {color=blue; }
      </style>
      </head>
      <body>
        <h3> liste des news </h3>
         <table border="3">
             <xsl:for-each select=".//news">
           <tr>
               <th> <xsl:value-of select="title" /> </th>
                 <td> <xsl:value-of select="@date" /> </td>
                 <td> <xsl:value-of select="auteur" /> </td>
                 <th> <xsl:value-of select="text" /> </th>
           </tr>
          </xsl:for-each>
         </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

L'url ***http://localhost:8080/serv_xslt/GetNewsServlet?style=pres2.xslt*** déclenche via TrAX la transformation de news.xml en une page HTML:

liste des news

Bientot Noel !!!	20/12/2002	Père Noël	**cadeaux uniquement prévus pour les enfants sages.**
Attention , danger !	20/06/2003	OufADonf	**Des transformations XSLT erronées peuvent produire des mutants.**

VI - JAXB (Binding XML <--> Java)

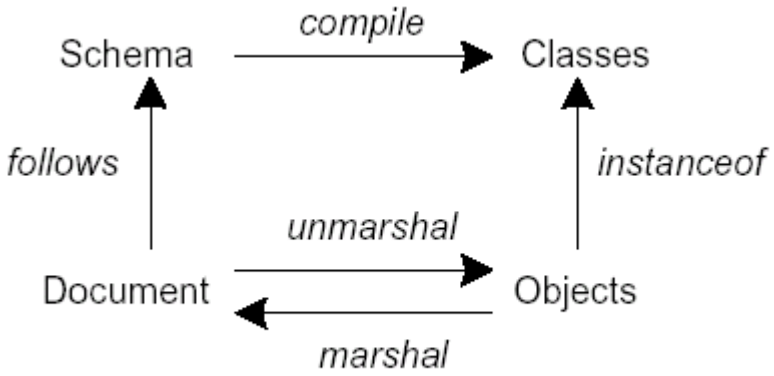
1. Principes

JAXB (**J**ava **A**pi for **X**ml **B**inding) est une api très récente permettant d'établir une **association (correspondance)** entre :

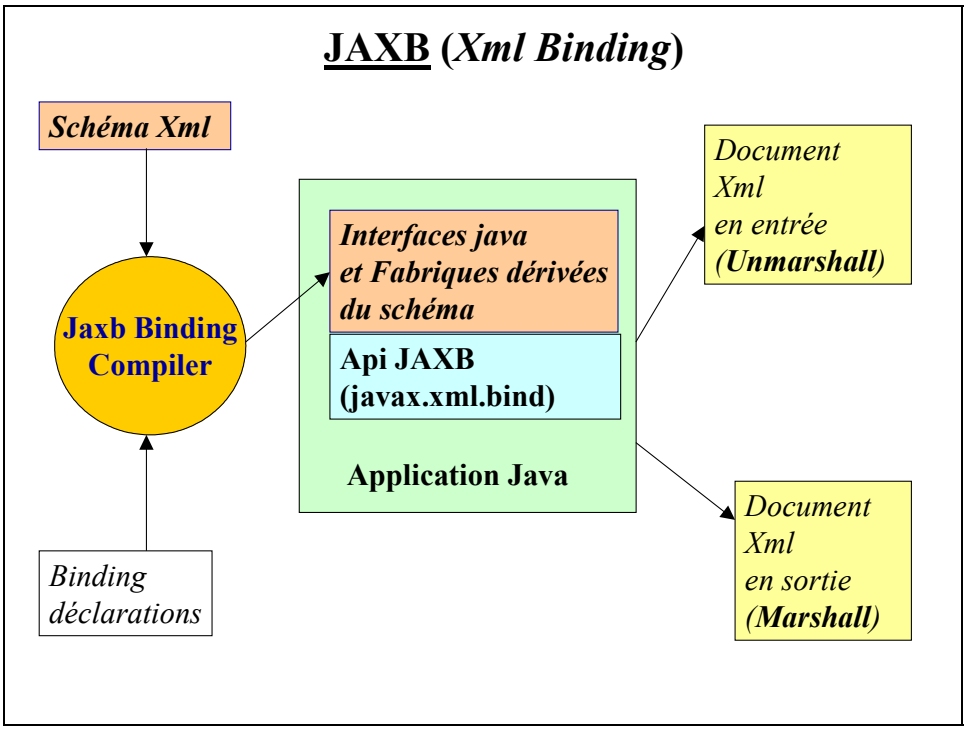
un **document XML** (dont le *type* est décrit par un *schéma XML*)

et

un **ensemble de classes java** (formant ensemble une structure arborescente) .



Une fois que les classes java ont été générées conformément au schéma , on peut alors effectuer très rapidement une "**sérialisation XML**" entre des instances des classes java (comportant certaines valeurs) et un fichier XML (instance du schéma) .



2. Mise en oeuvre

Fixer quelques variables d'environnement:

```
set JAXB_HOME=C:\Prog\jwsdp-1.1\jaxb-1.0
set JAXB_LIBS=%JAXB_HOME%\lib


set CLASSPATH=%CLASSPATH%;%JAXB_LIBS%\jaxb-api.jar;
%JAXB_LIBS%\jaxb-ri.jar;%JAXB_LIBS%\jaxb-xjc.jar;
%JAXB_LIBS%\jaxb-libs.jar;.
```

Invocation du compilateur de binding xml (générateur de code java en fonction d'un schéma xml):

```
%JAXB_HOME%\bin\xjc.bat -p pack1 po.xsd

  Usage: xjc [-options ...] <schema>
  Options:
    -nv       :  do not perform strict validation of the input schema(s)
    -d <dir>  :  generated files will go into this directory
    -p <pkg>  :  specifies the target package
    -readOnly :  generated files will be in read-only mode
    -help     :  display this help message
```

NB: ***xjc.bat*** lance en interne la commande java suivante:

```
java -jar %JAXB_HOME%/lib/jaxb-xjc.jar
```

3. Exemple de binding

Schéma XML (biblio.xsd):

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="bibliographie" type="BibliographieType"/>
<xsd:element name="comment" type="xsd:string"/>
<xsd:complexType name="BibliographieType">
  <xsd:sequence>
    <xsd:element name="titre" type="xsd:string"/>
    <xsd:element name="sujet" type="xsd:string"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="livre" type="Livre"  minOccurs="1"
maxOccurs="unbounded" />
  </xsd:sequence>
  <xsd:attribute name="date" type="xsd:date"/>
</xsd:complexType>

<xsd:complexType name="Livre">
        <xsd:sequence>
         <xsd:element name="titre" type="xsd:string"/>
         <xsd:element name="auteur" type="xsd:string"/>
         <xsd:element name="editeur" type="xsd:string"/>
        </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Document Xml (biblio.xml = instance possible du schéma)

```
<?xml version="1.0"?>
<bibliographie date="01/01/2002">
```

```
 <titre> Bibliographie sur Programmation Xml en java </titre>
 <sujet> Livres evoquant la programmation Xml en langage Java </sujet>
 <livre>
     <titre>XML, langage et applications</titre>
     <auteur>Alain Michard (INRIA)</auteur>
     <editeur>Eyrolles</editeur>
 </livre>
 <livre>
     <titre>Construire une application XML ...</titre>
     <auteur>J.C. Bernadac , F.Knab </auteur>
     <editeur>Eyrolles</editeur>
 </livre>
</bibliographie>
```

%JAXB_HOME%\bin\xjc.bat -p mybiblio biblio.xsd

==> Interfaces java et fabriques générées:

parsing a schema...
compiling a schema...
mybiblio\impl\BibliographieImpl.java
mybiblio\impl\BibliographieTypeImpl.java
mybiblio\impl\CommentImpl.java
mybiblio\impl\LivreImpl.java
mybiblio\Bibliographie.java
mybiblio\BibliographieType.java
mybiblio\Comment.java
mybiblio\Livre.java
mybiblio\ObjectFactory.java
mybiblio\jaxb.properties
mybiblio\bgm.ser

```
public interface BibliographieType {

    String getComment();
    void setComment(String value);

    java.util.Calendar getDate();
    void setDate(java.util.Calendar value);

    java.util.List getLivre();

    String getSujet();    void setSujet(String value);

    String getTitre();    void setTitre(String value);
}
```

```
public interface Bibliographie
    extends javax.xml.bind.Element, mybiblio.BibliographieType
{
}
```

```
public interface Livre {

    String getEditeur();   void setEditeur(String value);

    String getAuteur();     void setAuteur(String value);

    String getTitre();     void setTitre(String value);
}
```

```
public class ObjectFactory
    extends com.sun.xml.bind.DefaultJAXBContextImpl
{

    static {
        com.sun.xml.bind.ImplementationRegistry ir =
            com.sun.xml.bind.ImplementationRegistry.getInstance();
        ir.setDefaultImpl((mybiblio.Bibliographie.class),
                  (mybiblio.impl.BibliographieImpl.class)); ... ;
        ir.setDefaultImpl((mybiblio.Livre.class),
                        (mybiblio.impl.LivreImpl.class));
    }

    public ObjectFactory() {
        super(new mybiblio.ObjectFactory.GrammarInfoImpl());
    }

    public  Object newInstance(Class javaContentInterface)
        throws javax.xml.bind.JAXBException
    {...}

    public  mybiblio.Bibliographie createBibliographie()
        throws javax.xml.bind.JAXBException
    {
        return ((mybiblio.Bibliographie)
                  newInstance((mybiblio.Bibliographie.class)));
    }

    public  mybiblio.Xxx createXxx()
        throws javax.xml.bind.JAXBException
    {
        return ((mybiblio.Xxx) newInstance((mybiblio.Xxx.class)));
    }


    private static class GrammarInfoImpl
        extends com.sun.xml.bind.GrammarInfo
    {
        public Class getRootElement(String uri, String local) {
            if ("".equals(uri)&&"comment".equals(local)) {
                return (mybiblio.impl.CommentImpl.class);
            }
            if ("".equals(uri)&&"bibliographie".equals(local)) {
                return (mybiblio.impl.BibliographieImpl.class);
            }
            return null;
        }

        public String[] getProbePoints() {
            return new java.lang.String[] {"", "comment", "",
                                           "bibliographie"};
        }

    }

}
```

jaxb.properties

```
#Tue Nov 12 14:00:54 CET 2002
javax.xml.bind.context.factory=com.sun.xml.bind.ContextFactory
```

4. Structure de l'api JAXB

Le principal package de **JAXB** est **javax.xml.bind**

La classe abstraite **JAXBContext** est le point d'entrée de l'API.

```
JAXBContext jctx =
    JAXBContext.newInstance( "com.mycompany.pack1:com.mycompany.pack2" );
```

Le paramètre d'entrée de la méthode statique **newInstance** correspond à la liste de(s) package(s) de classes et d'interfaces java dérivées du schéma xml.

Une fois instancié, le contexte JAXB peut servir à créer les éléments suivants:

```
Unmarshaller um = jctx.createUnmarshaller(); // pour remonter des objets en mémoire
                                             // suite à la lecture d'un flux xml
Marshaller m = jctx.createMarshaller(); // pour générer un flux xml (fichier)
Validator v = jctx.createValidator(); // pour lancer une validation
```

Dé-sérialisation d'un flux xml:

```
XxxObject fooObj = (XxxObject)um.unmarshal( new File( "xxx.xml" ) );
```

Sérialisation vers un flux xml:

```
    m.marshal( XxxObj, System.out );
```

Création d'une nouvelle instance via la fabrique générée au sein du package (fruit de la compilation du schéma Xml):

```
com.mycompany.pack1.ObjectFactory  factory
                        = new com.mycompany.pack1.ObjectFactory();
com.mycompany.pack1.Xxx po =  factory.createXxx();
```

5. Exemple de code:

```
package myprog;

import javax.xml.bind.*; // JAXB
import mybiblio.*; // Result of XML Binding
import java.io.*;


/**
 * @author DDefrance
 *
 **/
```

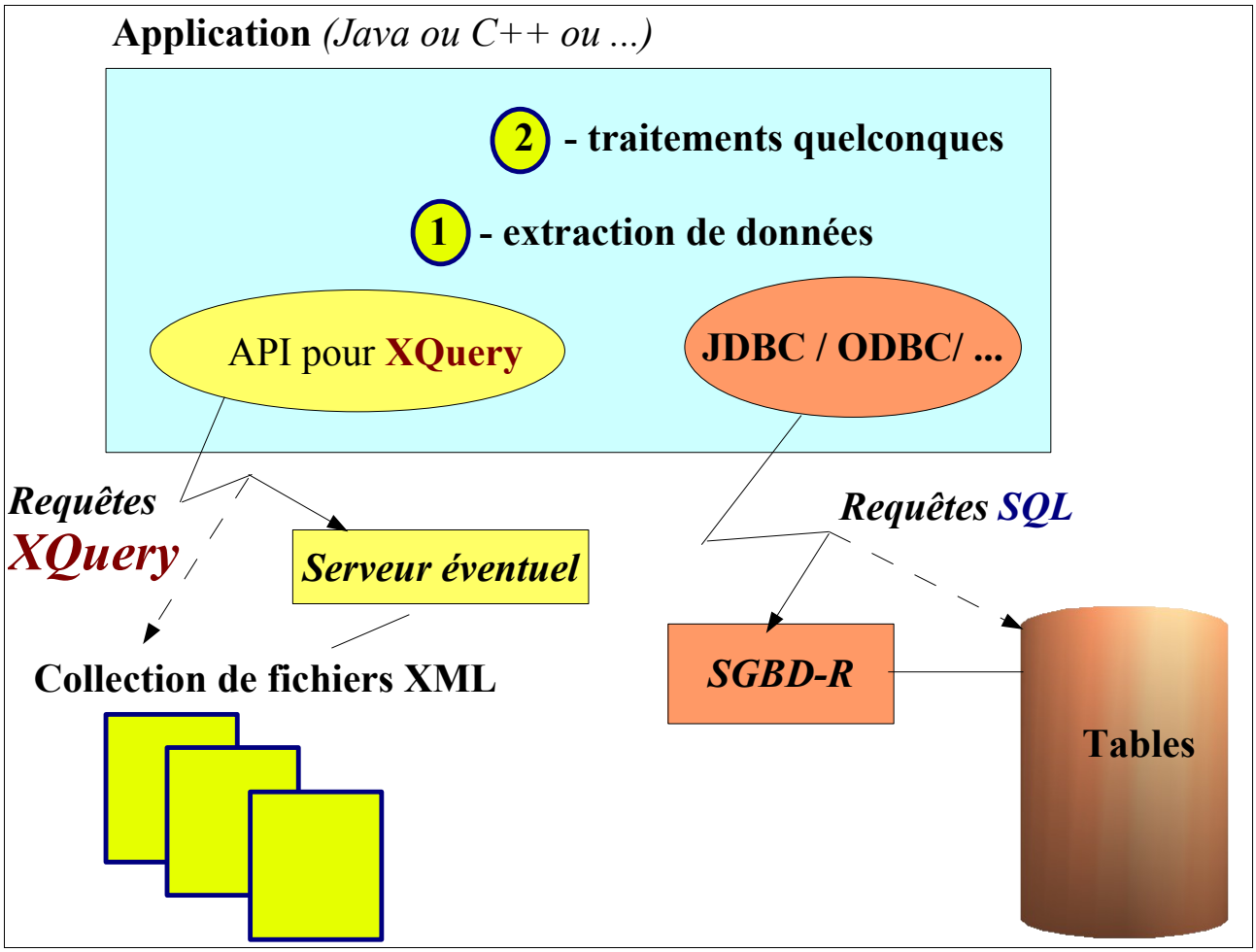
```
public class MyJaxbApp {

      public static void main(String[] args) {

      try {
      JAXBContext jctx = JAXBContext.newInstance("mybiblio");

     Unmarshaller um = jctx.createUnmarshaller(); // pour remonter des
       // objets en mémoire suite à la lecture d'un flux xml

      Marshaller m = jctx.createMarshaller();
       // pour générer un flux xml (fichier)



      Bibliographie biblioObj = (Bibliographie)
                  um.unmarshal( new File( "biblio.xml" ) );
      m.marshal( biblioObj, System.out );

      ObjectFactory objectFactory = new ObjectFactory();
      Bibliographie biblioObj3 = objectFactory.createBibliographie();
      java.util.Calendar d = java.util.Calendar.getInstance();
      biblioObj3.setDate(d);
      biblioObj3.setTitre("titre biblio");
      biblioObj3.setSujet("sujet qui va bien");

      biblioObj3.setComment("mon commentaire qui ne sert a rien");

      Livre l1 = objectFactory.createLivre();
      l1.setAuteur("Auteur qui ecrit bien");
      l1.setEditeur("Eyrolles");
      l1.setTitre("Titre du livre");

      Livre l2 = objectFactory.createLivre();
      l2.setAuteur("Auteur fou");
      l2.setEditeur("Masson");
      l2.setTitre("Titre2 du livre2");

      biblioObj3.getLivre().add(l1);
      biblioObj3.getLivre().add(l2);

            Validator v= jctx.createValidator();
            v.validate(biblioObj3);
            m.marshal( biblioObj3, new FileOutputStream("biblio3.xml") );


      }
      catch(Exception ex) { ex.printStackTrace(); }

      }
}
```

VII - XQuery

1.1. Présentation de XQuery

XQuery est un **langage d'extraction de données XML**.

D'un point de vue fonctionnel, ***XQuery*** est dans le monde XML, ce que la **partie "SELECT ..."** du langage **SQL** est dans le domaine des bases de données relationnelles .



Contrairement à XSLT , **XQuery** est un langage qui s'appuie sur des types de données très précis (*ex*: ***xs:string*** , ***xs:integer*** , ***element()***, ...) .
Ceci permet de faire en sorte que les données extraites soient directement transposables (sans aucune ambiguïté) dans des variables d'un langage fortement typé tel que Java , C++ ou C# .

D'un point de vue plus formel, **XQuery** est :

- **Un modèle de données *(avec types précis)* pour les documents XML.**
- **Un ensemble d'opérateurs (arithmétiques, logiques, ensemblistes, ...) basés sur ce modèle.**
- **Un langage d'interrogation basé sur ces opérateurs.**

1.2. Opérateurs et expressions de XQuery

1.2.a. Expressions de chemins (Path):

reprise de **XPath 1.0**

exemple: ***document("biblio.xml")//sujet[2]//livre[@titre = "java 5"]***

avec quelques suppléments:

- un opérateur permettant de suivre des références/liaisons (**->**)
- un prédicat de plage (*range*) pour exprimer certaines séquences: (1 **to** 4)

1.2.b. Opérateurs arithmétiques:

+ , - , * , / , ...

exemples: **3+4** **=> 7**

```
let $x := 5
let $y := 8
return $x+$y
```

```
=> 13
```

1.2.c. Séquences:

```
let $paire1 := (3,4)
let $paire_de_paire := ($paire1, $paire1)
let $element := 99
let $vide := ()
return (count($paire_de_paire) , count($paire1), count($element), count($vide))
```

⇨ génère `(4, 2, 1, 0)`.

1.2.d. Itérations:

```
for $x in (1 to 3) return ($x,5*$x)
```

⇨ **génère** 1,5,2,10,3,15

```
<html>{
let $sujet := document("biblio.xml")//sujet[1]
for $livre in $sujet/livre
return <h2>{$livre/titre/text()}</h2>
}</html>
```

=> génère :

```
<?xml version='1.0' encoding='UTF-8'?>
<html>
  <h2>Mieux développer en C++</h2>
  <h2>Numerical recipes in c++</h2>
</html>
```

1.2.e. Opérateurs logiques et ensemblistes:

Or, and, union, intersect, ...

1.2.f. Expressions conditionnelles :

exemples:

```
let $x :=5
return if ( $x mod  2 = 0) then "pair" else "impair"
```

```
for  $b in document("biblio.xml")//livre   order by ($b/titre/text())
return
<livre titre='{$b/titre/text()}'
        gamme_prix='{ if($b/@prix < 30) then "prix_vert" else "prix_orange" }' />
```

1.2.g. XQuery FLWR (For ... Let ... Where ... Return ...) Expression

La construction FLWR est très semblable à l'instruction SELECT du SQL.

La boucle **for** permet d'itérer sur un ensemble de valeurs ou de noeuds

pour chaque itération:

- l'instruction **let** peut éventuellement être utilisée pour calculer la valeur d'une variable qui sera utilisable au niveau des parties "**where**" et "**return**"
- la clause **where** permet d'effectuer un *filtrage*
- la partie **return** permet d'indiquer ce qu'il faut *générer*

Exemples:

```
for $b in document("biblio.xml")//livre
   where $b/editeur = "Eyrolles" and $b/@prix < 40
return $b/@titre
```

```
for  $e in distinct-values(document("biblio.xml")//editeur/text())
let  $moyenne := avg(document("biblio.xml")//livre[editeur = $e]/@prix)
return
<editeur>
     <nom> { $e} </nom>
     <prix_moyen> { $moyenne } </prix_moyen>
</editeur>
```

1.2.h. Fonctions

XQuery est un langage fortement typé qui nécessite la précision (via le mot clef "***as***") des types de données pour les paramètres et les valeurs de retour des fonctions .

exemples:

```
declare  function local:addition($x as xs:double , $y as xs:double) as xs:double
{
$x+$y
}

local:addition(2,3)
```

==> 5

```
define  function  local:liste_sujet($biblio as element() ) as element()
{
<html><body><ul>
 {
 for $s in $biblio/sujet
 return <li>{$s/titre/text()}</li>
 }
</ul></body></html>
```

```
}

local:liste_sujet(document("biblio.xml")/bibliographie)
```

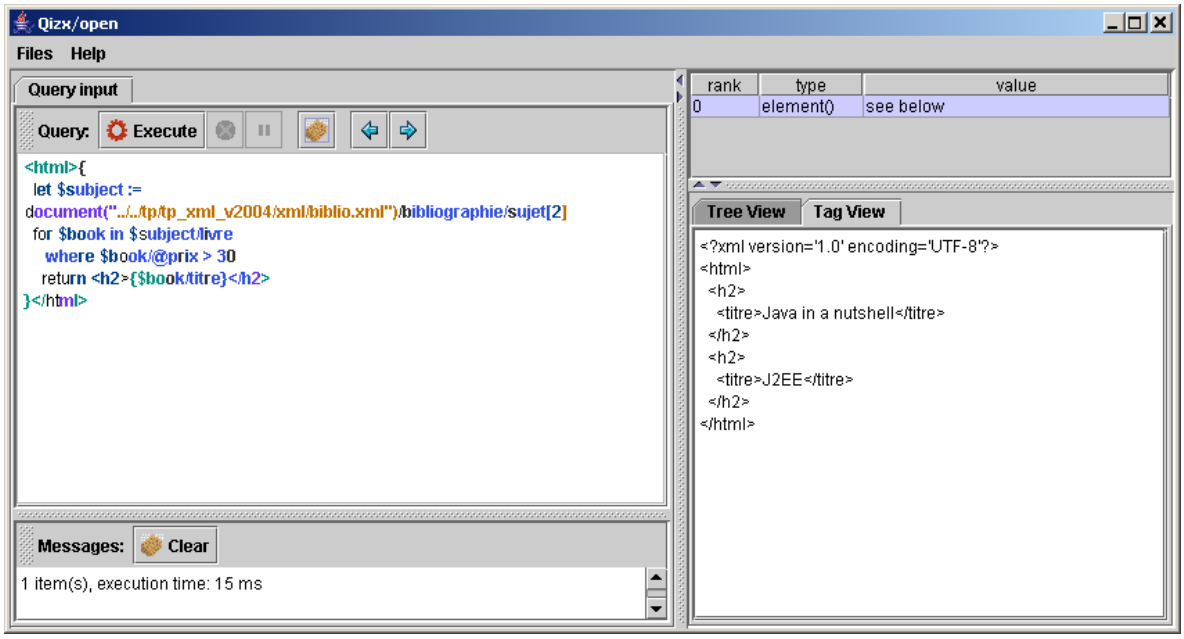
==>

```
<?xml version='1.0' encoding='UTF-8'?>
<html>
 <body>
      <ul>
         <li>C++</li>
        <li>Java</li>
        <li>XML</li>
      </ul>
</body>
</html>
```

1.3. Quelques produits "XQuery"

QizX/open	http://www.xfra.net/qizxopen/root.html	bonne documentation très simple à utiliser
Qexo / Kawa	GNU product	précurseur évoluant peu. latest release : 2003 !!!
eXist	http://exist.sourceforge.net/	respect des standards , ... base de données XML
...		
XML:DB API	http://xmldb-org.sourceforge.net/xapi/	api utilisé en interne par beaucoup de moteurs XQuery programmés en Java

1.4. QizX/open en mode GUI :


Qizx/open
Files Help
Query input
Query: Execute
<html>{
let $subject :=
document("../../tp/tp_xml_v2004/xml/biblio.xml")/bibliographie/sujet[2]
for $book in $subject/livre
where $book/@prix > 30
return <h2>{$book/titre}</h2>
}</html>
Messages: Clear
1 item(s), execution time: 15 ms
rank type value
0 element() see below
Tree View Tag View
<?xml version='1.0' encoding='UTF-8'?>
<html>
<h2>
<titre>Java in a nutshell</titre>
</h2>
<h2>
<titre>J2EE</titre>
</h2>
</html>