Structuration et échange d'informations sur le Web

Université Lyon 1
Master CCI

L. Médini, Janvier 2007

Résumé de l'épisode précédent

- Introduction
  - Définitions
  - Historique
  - Aperçu de quelques langages
- XML : principes de base
  - Hérité de SGML (beaucoup plus concis)
  - Méta-langage de description des données
  - Restrictions de syntaxe et non de contenu
  - Documents XML valides : les DTD
- HTML et XHTML
  - Langages de description des pages Web
  - Syntaxe
  - Langage de feuilles de style : CSS

Plan du cours 2 : Applications et programmation XML

- Applications XML
  - Notion d'espaces de noms XML
  - Retour sur la validation de documents : les schémas XML
  - Le langage de feuilles de style XSL
    - Xpath
    - XSLT
- Programmation XML
  - Les API existantes
  - Le Document Object Model (DOM)
  - Simple API for XML (SAX)

URI, URL et URN

- URI : Uniform Resource Identifier
  - But : identifier de façon unique une ressource sur le web
    - En disant où elle se trouve
      - Donner son URL (Uniform Resource Locator)
      - Format : protocole ":" chemin "/" nom de fichier "/" requête
      - `http://www/w3.org/2001/XMLSchema`
      - Permet d'accéder réellement à la ressource (tant qu'elle existe)
      - Enregistrement des DNS auprès de l'entité concernée
    - En disant comment elle s'appelle
      - Donner son URN (Uniform Resource Name)
      - Format : "URN:" NID (namespace identifier) ":" NSS (namespace specific string)
      - `URN:ISBN:0-395-36341-1`
      - Choix plus « libre », et correspondant mieux à la définition d'un espace de noms
      - Enregistrement des NID à l'IANA (Internet Assigned Numbers Authority)
  - Syntaxe générique

URI, URL et URN

- URI : Uniform Resource Identifier
  - But : identifier de façon unique une ressource sur le web
  - Syntaxe générique
    - « scheme » ":" autorité ":" chemin ":" requête ":" fragment
    - Avec le temps, on s'est mis à penser que « urn » peut aussi être un URI scheme
- D'un point de vue pratique, les URL sont plus sures afin d'éviter les conflits entre les espaces de noms
- ➔ Un URI est uniquement un identificateur, qui n'a pas de sens en soi
- ➔ Il ne signifie rien pour le processeur XML, qui le transmet tel quel à l'application

Espaces de noms XML

- Position du problème
  - Liberté de choix des noms de balises et des attributs XML
  - ⇒ Conflits et polysémie entre ces noms/attributs
  - Besoin d'associer plusieurs applications dans un même document
  - ⇒ « Préfixage » des noms de balises par l'URI de l'application concernée

Espaces de noms XML

- Noms qualifiés (qualified names)
  - Noms de balises appartenant à des espaces de noms
  - Syntaxe : PrefixeDEspaceDeNoms:PartieLocale
  - Exemple : `<xsl:stylesheet>`
  - Le préfixe fait référence à un URI
  - Les noms d'attributs peuvent également être préfixés
- Association d'un préfixe à un URI
  - Attribut `xmlns`
  - Exemple : `<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml">`
- Remarques
  - Portée : l'élément porteur de l'attribut `xmlns`
  - Bien entendu, un document XML peut contenir des éléments se référant à plusieurs espaces de noms
  - Le préfixe en lui-même n'a aucune signification
  - En interne, le parser passe à l'application des « noms pleinement qualifiés », où le préfixe est remplacé par la valeur de l'URI

Espaces de noms XML

- Espace de noms par défaut
  - Pas de préfixe d'espace de noms
  - Exemple : `<html xmlns="http://www.w3.org/1999/xhtml">`
- Annulation d'espaces de noms
  - Par valeur de l'attribut xmlns vide : `xmlns=""`
- Exemple de code

```
<?xml version="1.0"?>
<CV xmlns="http://www.univ-lyon1.fr/etds/CV/english"
    xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <personne>
    <civil_status>
      <title>Mr.</title>
    </civil-status>
    ...
  </personne>
  <xhtml:html>
    <xhtml:head>
      <xhtml:title>CV of a student</xhtml:title>
    </xhtml:head>
    <xhtml:body>
    ...
    </xhtml:body>
  </xhtml:html>
</CV>
```

Document XML valide : les schémas XML

- Comparaison DTD/Schémas

Caractéristique	DTD	Schémas
Syntaxe	Notation EBNF + pseudo-XML	XML 1.0
Outils	Outils SGML existants (chers et complexes)	Tous les outils XML existants et à venir
Supports DOM/SAX	Non	Oui (comme pour les fichiers XML).
Modèles de contenu	- Listes : ordonnées ou de choix - Cardinalité : 0, 1 ou plusieurs occurrences - Pas d'éléments nommés ou de groupes d'attributs.	- Listes : ordonnées et de choix (détails de contenus mixtes) - cardinalité : spécification d'un nombre exact d'occurrences possible - groupes de modèles nommés
Typage des données	Faible (chaînes, jetons nominaux, ID…)	Fort (nombres, chaînes, date/heure, booléen, structures…)
Héritage	Non	Oui
Extensibilité	Non (pas sans modification de la recommandation XML)	Oui (puisque fondés sur l'extensibilité de XML)
Contraintes légales	Compatibilité avec SGML	Aucune (simplement des « emprunts » aux DTD, comme pour les types de données)
Nombre de vocabulaires supportés	Une seule DTD par document	Autant que nécessaire (grâce aux espaces de noms)
Dynamicité	Aucune : les DTD sont en lecture seule	Peuvent être modifiés dynamiquement

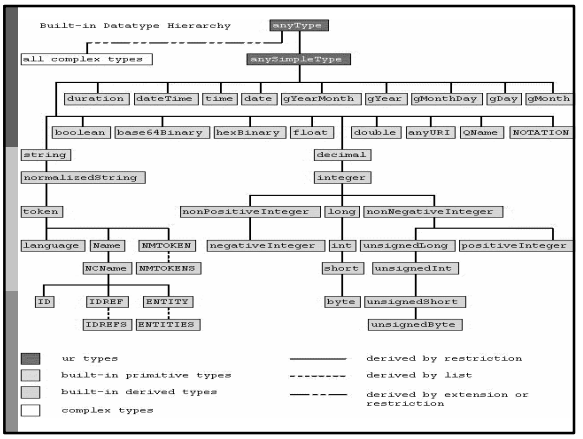
Document XML valide : les schémas XML

- Principes de base des schémas XML
  - Utilisation de la syntaxe et des outils XML
    - ⇒ Extensibilité
    - ⇒ Dynamicité
  - Possibilité de définir ses propres types de données et modèles de contenus
  - Un schéma définit une classe de documents dont chaque document est une instance
  - S'appuient sur les notions de
    - Types de données
    - Structures

Document XML valide : les schémas XML

- Les types de données : 3 dichotomies
  - Hiérarchie arborescente à partir d'un *ur-type*
    - Types primitifs : premier niveau de décomposition
    - Types dérivés : tous les niveaux suivants
  - La recommandation définit un ensemble de types
    - Types intégrés
    - Types dérivés par l'utilisateur
  - Atomicité
    - Types atomiques : dont les valeurs ne peuvent pas être décomposées
    - Types listes : ensembles de valeurs atomiques

  Remarques
    - Tous les types primitifs sont intégrés. La réciproque est fausse
    - string est un type atomique



Document XML valide : les schémas XML

- Types de données
  - Les types de données comportent 3 caractéristiques
    - Espace lexical : définit tous les caractères représentant les valeurs possibles
    - Espace de valeurs : ensemble des valeurs exprimé dans l'espace lexical
    - Facettes : propriétés définitionnelles de l'ensemble des valeurs
      - Facettes fondamentales : propriétés abstraites (égalité, bornes, ordre, cardinalité, numérique ou non)
      - Facettes de contraintes : limitent certaines propriétés (12 facettes : length, enumeration, minExclusive...)
  - Voir poly p. 60

Document XML valide : les schémas XML

- Les structures
  - Permettent de définir des types de données (contenus et attributs) selon deux méthodes
    - SimpleType : dérivation de types atomiques
      - Par restriction (par intension)
      - Par liste (par extension)
      - Par union (sur-ensemble de types existants)

```
<xsd:simpleType name ="myInteger">
 <xsd:restriction base="xs:integer">
  <xsd:minInclusive value="-2"/>
  <xsd:maxExclusive value="5"/>
 </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="myIntList">
 <xsd:list>
  <xsd:simpleType>
   <xsd:restriction base="xs:integer">
    <xsd:maxInclusive value="100"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:list>
</xsd:simpleType>
```

```
<xsd:simpleType name="intOrUndefined">
 <xsd:union>
  <xsd:simpleType>
   <xsd:restriction base="xs:integer"/>
  </xsd:simpleType>
  <xsd:simpleType>
   <xsd:restriction base="xs:NMTOKEN">
    <xsd:enumeration value="undefined"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:union>
</xsd:simpleType>
```

Document XML valide : les schémas XML

- Les structures
  - Permettent de définir des types de données (contenus et attributs) selon deux méthodes
    - ComplexType : autres types de dérivation
      - Dérivation
        - par restriction d'un type de base complexe,
        - par extension d'un type de base (simple ou complexe),
        - par restriction de l'ur-type definition
      - En pratique : la définition d'un type complexe est une composition
        - De séquences (ET)
          - ordonnées : `xsd:sequence`
          - non-ordonnées : `xsd:all`
        - De choix (OU) : `xsd:choice`

Document XML valide : les schémas XML

- Les structures
  - Définition d'un élément
    - Avec la balise `xsd:element`
    - En utilisant le type choisi (simple ou complexe)
  - Définition d'un attribut
    - Avec la balise `xsd:attribute`
    - En utilisant un type simple
  - Éléments de syntaxe : poly p. 61.
  - Pour aller plus loin : un cours très instructif

http://globalcomputing.epfl.ch/unifr/seance02-xml-schema-1/xml-schema-notes.pdf

```
<xs:element name="recette">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="titre" type="xs:string"/>
   <xs:element name="commentaire" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
   <xs:element name="item" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="entete" type="xs:string" minOccurs="0"/>
      <xs:choice maxOccurs="unbounded">
       <xs:element name="ingredient" type="xs:string"/>
       <xs:element name="preparation" type="xs:string"/>
      </xs:choice>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

```
<xs:element name="MusicDescription">
 <xs:complexType>
  <xs:all>
   <xs:element name="country" type="xs:string"/>
   <xs:element name="originalTitle" type="xs:string"/>
   <xs:element name="author" type="xs:string"/>
  </xs:all>
 </xs:complexType>
</xs:element>
```

```
<xs:element name="picture" minOccurs="0">
 <xs:complexType>
  <xs:attribute name="source" type="xs:anyURI"/>
 </xs:complexType>
</xs:element>
```

Document XML valide : les schémas XML

- Préambule d'un schéma
  - Avec gestion des espaces de noms

```
<xsd:schema xmlns:xsd="http://www/w3.org/2001/XMLSchema"
xsd:targetNamespace="http://www.monsite.com/monnamespace">
```

  - Sans gestion des espaces de noms

```
<xsd:schema xmlns:xsd="http://www/w3.org/2001/XMLSchema"
xsd:noTargetNamespace="noTargetNamespace">
```

ou simplement

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

Document XML valide : les schémas XML

- Association d'un document à un schéma
  - Avec gestion des espaces de noms

```
<ici:element
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://monsite.com/monnamespace
http://monsite.com/monnamespace/schema/MonSchema.xsd"
   xmlns:ici="http://monsite.com/monnamespace">
```

  - Sans gestion des espaces de noms

```
<element
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://monsite.com/monnames
   pace/schema/MonSchema.xsd">
```

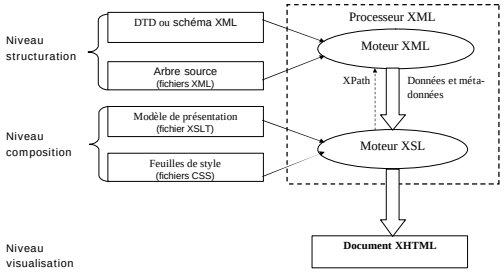
  - Dans tous les cas, il faut fournir une URI vers le schéma

Transformation d'arbres XML : XSL

- Caractéristiques de XSL
  - Officiellement : XML Stylesheet Language
  - En pratique, ça ne sert à rien d'appliquer des éléments de style à un document XML
  - Mais XSL fournit un mécanisme très puissant pour transformer un arbre XML
    - En un autre arbre XML (échange de données)
    - En un arbre XHTML (visualisation des données XML)
    - En un texte simple (fichier non destiné à une application utilisant un parser XML)
    - En un document papier formatté (XSL-FO)

Transformation d'arbres XML : XSL

- Utilisation la plus courante de XSL



Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XPath
    - Permet de pointer vers les données de l'arbre XML
      - pour le parcours de documents XML
      - pour le test de valeurs associées aux contenus ou aux attributs d'éléments
    - Ne respecte pas la syntaxe XML
      - pour ne pas « perturber » l'analyse des feuilles de style XSLT par le parser XML

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XPath
    - Nœud
      - Tout type de données (élément, attribut, PI)
      - Racine du document : '/'
      - Les éléments sont identifiés par leurs noms
      - Les attributs sont identifiés par '@' suivi du nom de l'attribut
    - Chemin de localisation
      - Absolu : à partir de la racine de l'arbre XPath
      - Relatif : à partir du nœud contextuel
      - Récursif : à partir du nœud contextuel, mais seulement « vers le bas »

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XPath
    - Axes de navigation
      - Déplacements complexes dans l'arbre XPath
      - Voir liste poly p. 75
    - Filtrage des nœuds
      - Permet de sélectionner des nœuds par leurs contenus
      - Définition des caractéristiques recherchées entre crochets
      - Opérateurs de comparaison simples
      - Fonctions XPath
        - Expression de caractéristiques de sélection complexes
        - Voir liste poly p. 74

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : principes de base
    - Description de l'arbre résultant (programmation déclarative)
    - Application XML définissant des « éléments de transformation »
      ⇒Référence à un espace de noms spécifique « `xsl:` »
    - Balises spécifiques interprétées par un processeur XSLT
    - Structuration par modèles (« templates ») de contenus
      - Définissant le traitement à appliquer à un élément repéré par une expression XPath
      - Imbriqués grâce à des balises d'application de templates

      ➔parallèle avec les fonctions en programmation déclarative

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : syntaxe
    - Élément racine

```
<xsl:stylesheet      version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

    - Éléments de premier niveau (cardinalité=0 ou 1)
      - `<xsl:output>` : définit le type d'arbre de sortie
        - Attribut `method` : 3 valeurs possibles (`text`, `html`, `xml`)
        - Autres attributs : `version`, `encoding`, `standalone`, `indent`…
      - `<xsl:include>` et `<xsl:import>` : permettent d'inclure d'autres feuilles de style
        - Attribut `href` : URI de la ressource à inclure
        - Différence entre les deux : règles de priorités

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : syntaxe
    - Éléments de premier niveau (cardinalité=0 ou 1)
      - `<xsl:strip-space>` et `<xsl:preserve-space>` : gestion des espaces dans l'arbre résultant (resp. suppression et conservation)
        - Attribut `elements` : noms des éléments concernés séparés par des espaces
      - `<xsl:template>` : modèle racine de l'arbre de sortie
        - Attribut `match` : désigne le nœud XPath concerné par le modèle (au premier niveau, toujours "/")
        - Contient la racine de la déclaration de l'arbre de sortie
      - Autres éléments (key, variable, attribute-set, param) : voir la recommandation

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les templates
    - Définition
      - Modèles simples : `<xsl:template match="noeud_XPath">`
        - L'expression XPath qui définit le nœud peut inclure un filtre
        - Ce nœud devient le nœud contextuel dans le template
      - Modèles nommés : `<xsl:template name="nom_tmplate">`
    - Appel
      - Modèles simples :

```
<xsl:apply-templates select="expr_XPath" />
```

        - L'expression XPath est un chemin de localisation qui désigne le nœud
      - Modèles nommés :

```
<xsl:call-template name="nom_tmplate" />
```

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les éléments
    - Génération de contenus XML
      - `<xsl:element name="p" namespace="xhtml">Contenu de l'élément (ici: un paragraphe XHTML)</xsl:element>`

        Remarque : `<xsl:element>` n'est nécessaire que lorsque le nom de l'élément à générer doit être calculé
      - `<xsl:attribute name="href" namespace="xhtml">Contenu de l'attribut (ici : référence XHTML)</xsl:attribute>`

        Remarque : `<xsl:attribute>` se place dans l'élément auquel il se rapporte
      - `<xsl:text>Contenu textuel quelconque.</xsl:text>`

        Remarque : `<xsl:text>` ne sert qu'au formatage du texte (gestion des espaces…)
      - Tout autre élément XML bien formé est accepté

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les éléments
    - Traitement de contenus de l'arbre XML source
      - `<xsl:value-of select="expr_XPath" />`
        - Permet d'obtenir la valeur d'un nœud (élément ou attribut)
        - L'expression XPath est un chemin de localisation
        - Elle désigne un nœud à partir du nœud contextuel
      - `<xsl:copy-of select="expr_XPath" />`
        - Permet de recopier dans l'arbre destination toute une partie de l'arbre source
        - L'expression XPath fonctionne comme précédemment
      - `<xsl:copy />`
        - Permet de copier uniquement un élément sans ses sous-éléments

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les éléments
    - Structures de contrôle
      - `<xsl:if test="expr_XPath">Contenu conditionnel</xsl:if>`
        - Le contenu conditionnel peut être composé d'autres éléments (`<xsl:value-of select="expr_XPath" />`)
      - `<xsl:for-each select="expr_XPath">Contenu répété</xsl:for-each>`
        - Cet élément est redondant avec `<xsl:apply-templates />` mais rend la feuille de style moins lisible

Transformation d'arbres XML : XSL

- Les deux composants de XSL
  - XSLT : les éléments
    - Structures de contrôle
      -
```
<xsl:choose>
  <xsl:when test="expr_XPath1">
        Contenu conditionnel 1
  </xsl:when>
  <xsl:when test="expr_XPath2">
        Contenu conditionnel 2
  </xsl:when>
  ...
  <xsl:otherwise>
        Contenu conditionnel n
  </xsl:otherwise>
</xsl:choose>
```

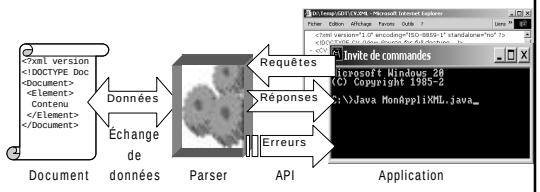
Outils de programmation avec XML

- Définitions
  - Qu'est-ce qu'un parser ?
    - « Un module logiciel [...] utilisé pour lire les documents XML et pour accéder à leur contenu et à leur structure. »
  - Qu'est-ce qu'une application ?
    - « On suppose qu'un processeur XML effectue son travail pour le compte d'un autre module, appelé l'application. »

http://babel.alis.com/web_ml/xml/REC-xml.fr.html#dt-xml-proc

Outils de programmation avec XML

- Communications entre parsers et applications
  - Rappel : Application Programming Interface
    - Outils
    - Protocole de communication
  - Schéma des échanges de données



XML et Java

- Standardisation des API
  - Nombreux parsers
  - API spécifiques

  ⇒ Le DOM (W3C)

  ⇒ SAX (xml-dev)
- Standardisation des accès aux parsers
  - Données conformes aux standards XML
  - Langage de programmation identique
  - Applications conformes aux API standards
  - Parsers implémentant ces API

  ?💣☠! Parsers différents pour faire la même chose

XML et Java

- JAXP : au départ
  - Java API for XML Parsing
  - Version 1.0

    Package Java additionnel au JDK 1.3
  - Couche intégration des parsers

    Instanciation du processeur transparente
  - Couche API

    Implémentation des API DOM et SAX

XML et Java

- JAXP : aujourd'hui
  - Java API for XML Processing
  - Version 1.2
    Package Java intégré au JDK 1.4
  - Couche intégration des parsers
    Instanciation du processeur transparente
  - Couche API
    - Implémentation des API DOM et SAX
    - Prise en charge des schémas XML
    - TrAX (Transformation API for XML)
      - XSLT
      - XSLTC

XML et Java

- JAXP : les packages java
  - Couche intégration des parsers
    `javax.xml.parsers`
  - Couche API
    - API DOM
      `org.w3c.dom`
    - API SAX
      `org.xml.sax`
      `org.xml.sax.helpers`
      `org.xml.sax.ext`
    - TrAX (Transformation API for XML)
      `javax.xml.transform`

XML et Java

- JAXP : l'intégration des parsers
  - Le package `javax.xml.parsers`
    - Les classes abstraites « factory »
      - Destinées à être instanciées
      - Possèdent une méthode `newInstance()`

      ⇒DOM : `DocumentBuilderFactory`
      - Possède une méthode `newDocumentBuilder()`

      ⇒SAX : `SAXParserFactory`
      - Possède une méthode `newSAXParser()`

XML et Java

- JAXP : l'intégration des parsers
  - Le package `javax.xml.parsers`
    - Les classes abstraites « parser »
      - Instanciées par les objets factory
      - Transparentes vis-à-vis du parser utilisé
      - Possèdent une méthode `parse()`

      ⇒DOM : `DocumentBuilder`

      ⇒SAX : `SAXParser`

XML et Java

- JAXP : l'intégration des parsers
  - Le package `javax.xml.parsers`
    - L'erreur `FactoryConfigurationError`
      - Erreur dans la configuration du parser par la classe factory
    - L'exception `ParserConfigurationException`
      - …

XML et Java

Exemple de code DOM

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
public class DomParsing{
  static Document doc;
  public static void main()
  {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    try {
       DocumentBuilder builder = dbf.newDocumentBuilder();
       doc = builder.parse( new File("CV.xml") );
    ...}
    catch (ParserConfigurationException pce) { // Peut-être généré par la méthode
    }                                          // newDocumentBuilder()
    catch (SAXException se) {    // Peut être générée par la méthode parse()
    ...}
    catch (IOException ioe) {     // Peut être générée par la méthode parse()
    ...}
    catch (IllegalArgumentException iae) {  // Peut être générée par la méthode parse()
    ...}
  } // main
```

XML et Java

Exemple de code SAX

```
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
public class SAXParsing extends DefaultHandler {
    public static void main()  {
        DefaultHandler dh = new SAXParser();
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            SAXParser sp = factory.newSAXParser();
            sp.parse( new File("CV.xml"), dh);
        } catch (Throwable t) {...}
    }
    public void startDocument()  throws SAXException { ... }
    public void endDocument()    throws SAXException { ... }
    public void startElement(String namespaceURI, String LocalName, String QualifiedName,
                    Attributes atts) throws SAXException { ... }
    public void endElement(String namespaceURI, String LocalName, String QualifiedName)
throws SAXException { ... }
    public void characters(char buf[], int offset, int len) throws SAXException { ... }
    ...
}
```

XML et Java

- JDOM : l'alternative à JAXP ?
  - Représentation arborescente d'un document XML
  - Plus « facile » à utiliser que DOM et SAX
  - Compatible avec DOM et SAX
    - ⇒ Surcouche de DOM et SAX
  - Pas incompatible avec JAXP
  - Packages
    - `org.jdom ; org.jdom.adapter ; org.jdom.input ; org.jdom.output ; org.jdom.transform ; org.jdom.xpath`
  - Site web
    - http://www.jdom.org

Les API standard

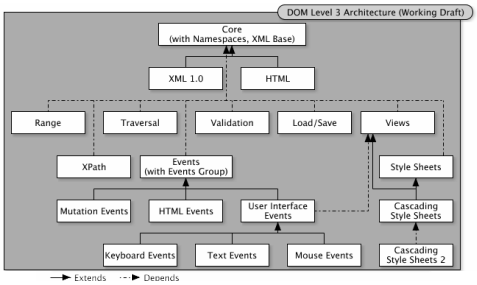
- Le DOM : généralités
  - Modèle objet de document
  - Motivations
    - Rendre les applications W3 dynamiques
    - Accéder aux documents HTML et XML depuis un langage de programmation
  - Utilisations courantes
    - Intégré aux navigateurs
    - Utilisé en programmation comme API XML
  - Origine : DOM working group (W3C)
    - Début : 1997 ; fin : …
    - Standardiser les tentatives existantes

Les API standard

- Le DOM : principes fondamentaux
  - Représentation arborescente d'un document
    - Tout le document est chargé en mémoire
    - Navigation dans la structure arborescente
    - Représentation des nœuds par des *interfaces*
      - Propriétés
      - Méthodes
  - Recommandations sous forme de niveaux
    - Niveau 0 : avant…
    - Niveau 1 : octobre 1998
    - Niveau 2 : depuis novembre 2000
    - Niveau 3 : depuis janvier 2004

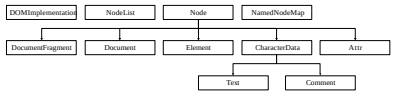
Les API standard

- Le DOM : fonctionnalités

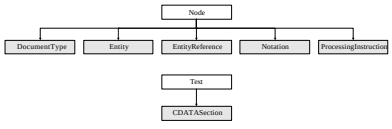


Les API standard

- Le DOM : modules
  - DOM Core :



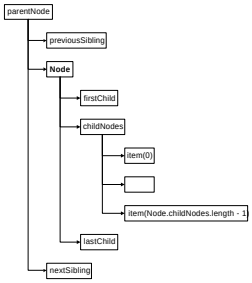
  - DOM XML :



  - Détail des interfaces : poly p. 93

Les API standard

- Le DOM : hiérarchisation des interfaces (module Core)



Les API standard

- Le DOM : utilisation en Java
  - Package JAXP `javax.xml.parsers`
  - Package spécifique `org.w3c.dom`
  - Liste des interfaces (Core + XML) : poly p. 113
  - Détail des interfaces : http://java.sun.com/j2se/1.4.2/docs/api

Les API standard

- SAX : principes fondamentaux
  - Simple API for XML
  - Issue d'une communauté de développeurs (liste xml-dev, sur http://www.xml.org)
  - Fondée sur la programmation événementielle
    - Pas de chargement de tout le document en mémoire
    - Pas de vision globale du document
  - À l'origine : développée en Java
  - Depuis : implémentations dans d'autres langages
  - 2 versions différentes

Les API standard

- SAX : principes fondamentaux
  - Des interfaces
    - Pour programmer des parsers compatibles SAX
    - Pour programmer des applications compatibles SAX
  - Des classes
    - Pour faciliter la programmation
    - Pour la gestion des erreurs
  - Des exceptions

Les API standard

- SAX : utilisation en Java
  - Procédure
    - Instanciation d'un parser
    - Lancement de l'analyse
    - Appel/implémentation de fonctions spécifiques
      - Interface `XMLReader` : `setDTDHandler()`
      - Interface `ContentHandler` : `startElement()`, `characters()`
      - Interface `Attributes` : `getLength()`, `getType`, `getValue()`
      - Interface `ErrorHandler` : `fatalError()`, `error()`, `warning()`

Les API standard

- SAX : utilisation en Java
  - Package JAXP `javax.xml.parsers`
  - Packages SAX `org.xml.sax`, `org.xml.sax.helpers`, `org.xml.sax.ext`
  - Présentation générale : poly p. 115
  - Détails http://java.sun.com/j2se/1.4.2/docs/api

Conclusion

- Dans ce cours, on a vu
  - Le contexte et l'historique
  - Les principes et langages de base
  - Les outils de traitement
- Ce qu'il faut retenir
  - La signification des acronymes
  - Les principes de base
  - Le schéma général d'articulation des langages et des outils
- Ce qu'il est autorisé d'oublier
  - Les détails de syntaxe des différents langages

Conclusion

- Ce qu'on n'a pas vu
  - Intégration XML / bases de données
    - Langage : XQuery
    - Connaissances nécessaires : XPath, XML Schémas, SQL
  - Services web et applications réparties sur le web
    - Langages : SOAP, WSDL, UDDI
    - Connaissances nécessaires : POO, objets répartis, protocoles web, serveurs d'applications
  - Web sémantique
    - Langages : RDF, RDF-S, OWL
    - Connaissances nécessaires : logiques de description, techniques de raisonnement, ingénierie des connaissances, intelligence artificielle…