Approche Objet - XML

Marie Beurton-Aimar

September 27, 2022

Méthodes de modélisation des connaissances

- Structuration des données en fonction des supports :
  - Base de données : SGBD
  - Fichiers : XML
- Langages de description de modèles : Merise, UML.

Les Markup Languages

- Les Markup Languages (ML) sont des outils qui permettent d'exprimer des modèles de structuration des données.
- Un ensemble d'outils permet de lier plus ou moins strictement la description et le fichier de données.
- **Remarque** : ceci ne dispense nullement de réaliser le modèle de conception (avec UML ou un autre langage).

Le Langage XML

- XML pour *eXtensible Markup Language*.
- Ancêtre : GML (Generalised Mark-up Language) de Charles Goldfarb, lui- même inspiré par William Tunicliffe (1967) qui décrit le premier la séparation du contenu d'un document de sa présentation.
- En 1986 le standard SGML est établi, 3 ans avant la création de HTML et du Web.
- XML est développé en 1996 par Jon Bosak.
- Historiquement, l'apparition de XML dans la suite de SGML et HTML fait que les présentations de XML et HTML sont souvent liées mais dans les faits, il n'existe pas d'obligation de considérer XML dans le contexte du Web.

Différences entre XML et HTML

- XML est un langage descriptif.
- Originaire du SGML, il s'illustre lui aussi à l'aide de balise.
- La différence avec HTML se situe au niveau de sa capacité à s'auto-structurer dans sa façon de décrire l'information. Alors que ce dernier se contente de formater une information pêle mêle.
- La balise XML décrit l'information qu'elle jalonne alors que le HTML détermine la façon de présenter l'information qu'il balise.

Format XML

- Un fichier XML est balisé par des TAGS ou mots clés qui fonctionnent comme un système de parenthésage mathématique.
- Ces mots clés peuvent être vus comme les éléments lexicaux et l'agencement ou composition que vous réalisez avec ces mots clés définit une sorte d'arbre syntaxique qui permet de reconstruire les instances des objets de votre programme.
- La notion d'arbre est ici essentielle, un document XML possède une racine, des branches de décomposition et des feuilles avec les valeurs des variables.

Les TAGS

- TAG décrivant un noeud de l'arbre :

  <section> Debut du noeud
  </section> Fin du noeud

- TAG avec attribut :

  <Chapitre numero="1">

  NB : le TAG se referme sans citer l'attribut.

Les TAGS

- Si XML définit lui-même un ensemble de TAGS, il doit surtout être considéré comme un **meta langage**, c'est à dire un langage qui permet d'en définir d'autres.
- Les **attributs** :
  - Chaque TAG peut supporter des spécifications qui sont alors indiqués comme les valeurs des attributs de ces TAGS.
  - Les attributs autorisés pour chaque TAG doivent être précisés lors de la définition du langage.
- Les TAGS et les attributs ne sont pas prédéfinis, ce qui signifie que l'utilisateur est libre de créer les TAGS et les attributs qui lui sont nécessaires.

Définir une arborescence

- La structure d'un document se décompose en un préambule (entête) et un corps.
- Ce corps commence par le TAG racine de l'arbre et se terminera par le même TAG dans sa position *fermée*.
- Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<journal nom="Linux à Gogo">
   <!--Description du numero''-->
   <Chapitre numero="1">
      ..................
   </Chapitre>
</journal>
```

Visualiser un fichier XML

- La plupart des navigateurs Web peuvent afficher correctement les données d'un fichier XML sous forme d'arbre.
- **Attention** : à l'heure actuelle le traitement des fichiers XML par les navigateurs n'est pas normalisé. Il est extrêmement fréquent d'obtenir des résultats différents suivant le navigateur utilisé.
- Exemple :

Déclaration de la structure du document

- Le fait que la définition de la structure du document et les données soient placées ensemble dans le fichier XML est un des reproches majeurs qui est fait à XML.
- DTD et Schémas
  - Il est possible de séparer ces deux informations et de spécifier la structure dans un *Document Type Definition* **DTD** ou bien dans un *Schéma* **xsd**.

Caractéristiques des DTD

- Un DTD décrit la structure logique du document.
- C'est un ensemble de règles ou contraintes que tout document qui déclare ce DTD doit respecter afin d'être considéré comme bien formé.
- **Remarque** : un document qui ne déclare pas de DTD est considéré bien formé par défaut.
- La présence d'un DTD permet d'exporter la structure d'un document à l'extérieur de celui-ci.
- Lorsque deux documents déclarent le même DTD il est possible de garantir qu'ils respectent la même syntaxe.

Caractéristiques des DTD

- Déclaration d'un fichier DTD :

```
<!DOCTYPE mesDTD ''mesDTD.dtd''>
```

- La vérification des règles du DTD se fait au moyen d'un *parser* XML.
- Un parser a un rôle assez semblable à celui d'un compilateur :
  - Analyse syntaxique du programme et signalement des erreurs.
  - Traduction du programme en langage machine - remplacement des entités par leur valeur.
  - Construction de l'arbre syntaxique - mais pas de traitement dans le cas du parser.

Caractéristiques des DTD

- Les DTD ont leur propre langage :
  - Déclarer un élément :

    <!**ELEMENT** nom du champs (type du champs)>
  - Exemple:

    <!**ELEMENT** journal (#**PCDATA**)>
  - Fichier XML de données correspondant :

    <journal> Linux a Gogo </journal>

Caractéristiques des DTD

- Il est possible de déclarer une liste d'éléments composites comme suit :

```
<!ELEMENT journal (nom, adresse, numero)>
```

- Liste d'attributs :

```
<!ATTLIST champs type_attribut valeur>
<!ATTLIST journal type_journal #PCDATA>
```

- Fichier XML correspondant :

```
<journal type_journal=''mensuel''>
    <nom> Linux à Gogo </nom>
        <adresse> Paris </adresse>
        <numero> 25 </numero>
</journal>
```

Caractéristiques des DTD

- Déclaration des entités : une entité est un type de données qui permet de faire référence à un autre élément du DTD en tant que type de l'élément du TAG.

```
<!ENTITY nom valeur>
<!ENTITY MENS ''Mensuel''>
<!ATTLIST journal type_journal ENTITY #REQUIRED>
```

- Fichier XML correspondant :

```
<journal type_journal=''MENS''>
        <nom> Linux a Gogo </nom>
</journal>
```

- Il est possible de déclarer des entités externes c.-à-d. décrites dans un autre fichier.

```
<!ENTITY MENS SYSTEM ''entites.xml''>
```

Limites et Inconvénients des DTD

- Les DTD sont écrits dans une syntaxe particulière différente de la syntaxe XML.
- Un seul type primitif PCDATA défini.
- Pas de possibilités de poser des contraintes sur les données ; nombre précis d'occurences (seule la forme * ou 0..n existe), format (date, longueur de chaine.
- Depuis mai 2001 le W3C préconise donc de remplacer les DTD par des **schémas** XML.

Les Schémas

- Contrairement aux DTD, les schémas permettent de décrire l'imbrication et l'ordre d'apparition des éléments et de leurs attributs soit une **grammaire** d'un langage particulier.
- Un **schéma** XML est un fichier écrit en XML.
- Si un **schéma** existe le fichier de données devra contenir la déclaration de ce *schéma*.

Exemple :

```
<journal
  xmlns:xsi=
     ''http://www.w3.org/2001/XMLSchema-instance''
  xsi:noNamespaceSchemaLocation=''journal.xsd''>
  <nom> Linux à Gogo </nom>
</journal>
```

Description d'un Schéma

- Les descriptions utilisent un nouveau TAG `xsd`.
- L'élément racine du *schéma* est de type `xsd:schema`.
- Pour décrire un élément on donne son *nom*, son *type*, et si nécessaire les liens ou contraintes qui s'appliquent.
- Pour décrire un élément on donne son *nom*, son *type*, et si nécessaire les liens ou contraintes qui s'appliquent.

Exemple

```
<xsd:element name=''journal''>
  <xsd:complexType>
   <xsd:all>
     <xsd:element name=''nom''
                  type=''xsd:string''/>
     <xsd:element name=''adresse''
                  type=''xsd:string''/>
     <xsd:element name=''numéro''
                  type=''xsd:positiveInteger''/>
     <xsd:element name=''type_journal''
                  type=''type_j''/>
   </xsd:all>
 </xsd:complexType>
</xsd:element>
```

Exemple

```
<xsd:simpleType name=''type_j''/>
  <xsd:enumeration value=''mensuel''>
  <xsd:enumeration value=''hebdomadaire''/>
  <xsd:enumeration value=''journalier''/>
</xsd:simpleType>
```

Les Markup Languages dédiés

- Echange d'Objets : XMI - XML Metadata Interchange Specification
  - Documentation : `http://www.omg.org/news/pr99/xmi_overview.html`
  - Ce format est utilisé pour décrire des objets - par exemple des diagrammes de classes UML.
  - Les logiciels ArgoUML et Poseidon peuvent utiliser le format XMI.
- Les graphiques : Precision Graphics Markup Language PGML et Scalable Vector Graphics - SVG

Utiliser des fichiers XML

- Les fichiers XML ne sont pas destinés à être analysés *“manuellement”*, entre autre à cause de leur verbosité.
- Pour relire un fichier XML, le programme utilisé doit connaitre la syntaxe et les règles du langage XML et s’il existe, le DTD ou le schéma spécifique défini pour l’application.
- Un `parser` est un programme qui sachant un grammaire interprète les différents éléments contenu dans un fichier de données.

XSL

- Standard du W3C - Version 1.0 - basé sur le standard DSSSL (Document Style Semantics and Specification Language), norme ISO 1996.
- XSL est un langage de transformation de document : trier un document, extraire uniquement certaines informations.
- 2 étapes : transformation par XSLT du document XML en un autre document XML, puis mise en forme.
- C'est dans ce 2ième temps que l'on peut choisir le mode de présentation, par exemple HTML.
- La transformation est opérée par un processeur **XSLT**

XSLT - XPath

- Version 3.0 .
- Les navigateurs implémentent plus ou moins XSl/XSLT.
- Il est nécessaire d'implémenter XPath pour utiliser XSLT.
- XPath est un langage de *pattern matching* qui travaille à partir de l'arbre syntaxique obtenu par la transformation XSLT du fichier..

DOM - SAX Parser

- DOM - lit tout le document et extrait l'arbre - peut être interrogé à partir de ces informations.
- SAX - traite les données séquentiellement - plus efficace que DOM. Permet de n'extraire que les données choisies.

Les outils :

- La classe SaxParser permet d'instancier un parser de type SaxParser.
- JavaXP fournit 2 parsers pour DOM et pour SAX.
- JavaXB le plus récent, permet de faire correspondre un document XML à un ensemble de classes et vice-versa au moyen d'opérations de sérialisation/désérialisation nommées marshalling/unmarshalling.
  see https://www.jmdoudoux.fr/java/dej/chap-jaxb.htm

Le format JSON

- Format né dans les années 2000. Le format est un dérivé de JavaScript.
- Créé par Douglas Crockford. Dernière spécification 2017.
- 2 types de données composées : les objets (ensemble clé-valeur) et les listes ordonnées de valeurs (tableau).

Le format JSON

```
[
    {
        "name":"John",
        "city":"Berlin",
        "cars":[
            "audi",
            "bmw"
        ],
        "job":"Teacher"
    },
    {
        "name":"Mark",
        "city":"Oslo",
        "cars":[
            "VW",
            "Toyata"
        ],
        "job":"Doctor"
    }
]
```

Java et JSON

- Des classes dédiées dans le package `org.json.*`
- Propose une classe `JSONParser`.

Java et JSON

```
public static void main(String[] args) {
    JSONParser parser = new JSONParser();
    try {
      Object obj = parser.parse(new FileReader("c:\\file

      JSONObject jsonObject =  (JSONObject) obj;

      String name = (String) jsonObject.get("name");
      System.out.println(name);

      String city = (String) jsonObject.get("city");
      System.out.println(city);

      String job = (String) jsonObject.get("job");
      System.out.println(job);
```

Java et JSON

```
// loop array
  JSONArray cars = (JSONArray) jsonObject.get("cars");
  Iterator<String> iterator = cars.iterator();
  while (iterator.hasNext()) {
      System.out.println(iterator.next());
   }
 } catch (FileNotFoundException e) {
            e.printStackTrace();
 } catch (IOException e) {
            e.printStackTrace();
 } catch (ParseException e) {
            e.printStackTrace();
  }
 }
}
```