Markup Languages

Marie Beurton-Aimar

July 30, 2004

Introduction

Pour communiquer des informations (*données*) à un ordinateur (*un programme informatique*) il est nécessaire de lui indiquer comment il doit interpréter la suite de caractères (*bits*) qu'il reçoit. Cette interprétation passe par le typage des données. Dans sa forme la plus simple on indiquera que les caractères numériques sont : des entiers, des flottants, des nombres complexes etc.
Pendant de nombreuses années les programmes ont majoritairement traité des nombres qui étaient alors organisés en vecteur, matrice ... pour être ensuite analysés par des logiciels de calcul. L'interprétation est obtenue en associant les lignes et les colonnes à des variables recueillies pour plusieurs individus (ou expériences). Chaque entête de colonne porte généralement le nom de la variable et chaque ligne un identifiant (ou numéro) d'échantillon.

La volonté d'élargir le champs d'application des programmes a conduit la communauté scientifique à associer une sémantique plus complexes aux données manipulées. La structuration des données au moyen de types de données complexes tel qu'elle est proposée par les modèles objets est un premier élément de réponse à cette problématique. L'architecture du modèle : composition, héritage, fournit alors l'interprétation des données. Cette architecture peut être communiquée par le code d'un programme mais cela suppose une dépendance à un langage de programmation spécifique, celui dans lequel le modèle est codé. Une autre solution est de communiquer un ensemble de documents réalisés par exemple avec un langage de modélisation comme UML. Dans ce cas, il faut soit disposer d'un programme qui implémente ce modèle soit en écrire un afin de traiter les données. La dernière solution est de fournir les données avec leur modèle au format *texte* suivant une convention pré-établie et clairement exprimée dans le texte. C'est cette dernière solution que nous allons maintenant examiner.
Remarque : cette solution ne dispense nullement de réaliser le modèle de conception (avec UML ou un autre langage).

1 Quelques éléments de base sur les Markup Languages

Les Markup Languages (ML) sont des outils qui permettent d'exprimer des modèles de structuration des données. Pour cela on dispose d'une liste de 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.
La suite de ce document présente les concepts de base du langage XML, quelques outils qui permettent de manipuler les fichiers respectant ce format et se termine par une brève présentation de langages de cette famille qui ont été développés pour la biologie.

2 Concepts de base

2.1 Les TAGS

Un TAG est un mot - ou mot clé - encadré par les signes < > . L'ensemble des TAGS *"valides"* définit le langage de description.

Exemple : TAG entête qui doit être au début de chaque fichier XML, on déclare ici la version du langage et l'encodage qui autorise l'utilisation des ccents français :

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

TAG commentaire :

```
<!--Description du réseau métabolique''-->
```

TAG décrivant un noeud de l'arbre :

```
<section> Début du noeud
</section> Fin du noeud
```

TAG avec attribut :

```
<Chapitre numéro="1">
```

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

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.

2.2 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. En XML, 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.
Dans l'exemple précédent `numéro` est un attribut du TAG `Chapitre`.

2.3 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*. Dans l'exemple ci-dessous le TAG racine est <`journal`>.

2.4 Visualiser un fichier XML

La plupart des navigateurs Web peuvent afficher correctement les données d'un fichier XML sous forme d'arbre.
Exemple :
Fichier XML :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<journal nom="Linux à Gogo">
  <!--Description du numéro''-->
  <Chapitre numero="1">
     <section>
        <nom>Le coin du mongueur </nom>
        <page>2</page>
```

```
    </section>
    <section>
       <nom>L'astuce du mois</nom>
       <page>4</page>
    </section>
  </Chapitre>
  <Chapitre numero="2">
    <section>
       <nom>Une nouvelle interface Web </nom>
       <page>7</page>
     </section>
  </Chapitre>
</journal>
```

Pour **visualiser** cet exemple le recopier dans un fichier appelé `exemple.xml`, puis l'ouvrir avec un navigateur classique (attention vérifier que la version est assez récente pour supporter XML). Il doit apparaitre sous la forme d'un arbre qu'il est possible de plier ou déplier en cliquant sur le petit `+/-` qui se trouve devant le TAG.

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.

3 Déclaration de la structure du document - DTD et Schémas

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. 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**.
Historiquement, les DTD sont plus anciens, les Schémas reprennent l'essentiel des DTD avec des ajouts tel que la possibilité d'exprimer des contraintes, de définir un espace de noms etc....

3.1 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 docuemnt à l'extérieur de celui-ci. Lorsque deux documents déclarent le même DTD il est possible de garantir qu'ils respectent la même syntaxe.
Un parser de XML (cf la section suivante) vérifie les règles du DTD lorsque celui-ci est présent.
Pour déclarer un fichier DTD, la ligne suivante est nécessaire :

```
<!DOCTYPE mesDTD ‘‘mesDTD.dtd’’>
```

Quelques éléments de syntaxe Les DTD ont leur propre langage :

- Déclarer un élément :

  ```
  <!ELEMENT nom du champs (type du champs)>
  ```

- Exemple:

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

  Nous venons de déclarer un TAG `journal` qui est une chaine de caractères. Le fichier XML correspondant contiendra :

```
<journal> Linux à Gogo </journal>
```

- Déclarer un élément composite :

```
<!ELEMENT journal (nom)>
<!ELEMENT nom (#PCDATA)>
```

Fichier XML correspondant :

```
<journal>
     <nom> Linux à Gogo </nom>
</journal>
```

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

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

- 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>
</journal>
```

- 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 à Gogo </nom>
</journal>
```

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

```
<!ENTITY MENS SYSTEM ‘‘entités.xml’’>
```

3.2 Les Schémas

4 Utiliser des fichiers XML

Les fichiers XML ne sont pas destinés à être analysés *"manuellement"*, entre autre à cause de leur verbosité. Il est donc indispensable de disposer d'outils de lecture, d'interrogation et de génération qui allègent le travail de l'utilisateur.

4.1 Interroger un fichier XML

Pour relire un fichier XML, le programme utilisé doit connaitre la syntaxe et les règles du langage XML et s'il existe, le DTD spécifique défini pour l'application. Il existe un certain nombre de programme déjà disponible qui vous permette d'analyser simplement un fichier XML.

DOM : Ce programme permet de relire un fichier XML et de construire l'arbre correspondant.

SAX :

XPATH :

XQUERY

5 XML pour la biologie

La communauté des biologistes a voulu définir une extension de XML pour la biologie. En fait une liste de TAGs et de règles de composition spécifique de ce domaine. On notera que le niveau très général de la syntaxe XML permet (et en quelques sortes favorise) ce type de développement.

5.1 Le modèle SBML

Les différents projets internationnaux de description et d'annotation de données biologiques ont produit des langages respectant la norme XML et incluant des TAGS spécifiques à la biologie. Le projet SBMl pour System Biology Markup Language est l'un d'eux. Vous trouverez à cette adresse : http://www.sbml.org le modèle de structuration de données et plusieurs exemples de modèles traduit en SBML.
Le fichier `Agrobacterium.xml` est un exemple *"simplifié"*, chargé dans un navigateur il vous donnera une liste de réactions qui ont été décrites par J.Zucker pour la base Biocyc.
L'intégralité du fichier est à l'adresse :
http://genome.dfci.harvard.edu/ zucker/BPHYS/biocyc-open/
SBML est soutenu par le projet E-Cell (www.e-cell.org)de description des processus cellulaires.

5.2 Cell-ML et la connexion à MathML

Le projet Cell-ML est une autre proposition de Markup Language pour la biologie. Il fournit un modèles de structuration des connaissances très proche des logiciels traditionnels de modélisation physiologique (par exemple Gepasi). On dispose de beaucoup moins de concepts - le plus souvent uniquement celui de variable - mais Cell-ML est plus riche en modélisation dse opérations mathématiques nécessaires à la simulation du métabolisme ou des processus physiologiques. Ceci est possible par une très bonne intégration de MathML qui permet de décrire les opérations mathématiques à l'aide de TAGS.

6 Bibliographie

Livres :

- Comprendre XSLT par Bernd Amann et Philippe Rigaux - O'Reilly
- Schémas XML de Jean-Jacques Thomasson - Eyrolles

Sites Web :

- Le site officiel : http://www.w3.org/XML
- Le site français : http://xmlfr.org
- Un tutoriel pour les débutants chez developpez.com : http://xml.developpez.com/cours/
- Tutoriel XPath : http://jerome.developpez.com/xmlxsl/xpath/