Actes 6e journées « langages et modèles à objets », Mont Saint-Hilaire (CA), pp59-74, 2000

XML est-il le langage de représentation de connaissance de l'an 2000 ?

Jérôme Euzenat

INRIA Rhône-Alpes
355 avenue de l'Europe, 38330 Montbonnot, France
Jerome.Euzenat@inrialpes.fr

RÉSUMÉ. *De nombreuses applications (représentation du contenu, définition de vocabulaire) utilisent XML pour transcrire la connaissance et la communiquer telle quelle ou dans des contextes plus larges. Le langage XML est considéré comme un langage universel et sa similarité avec les systèmes à objets a été remarquée. XML va-t-il donc remplacer les langages de représentation de connaissance ? Un exemple concret permet de présenter quelques questions et problèmes posés par la transcription d'un formalisme de représentation de connaissance par objets en XML. Les solutions possibles de ces problèmes sont comparées. L'avantage et la lacune principale d'XML étant son absence de sémantique, une solution à ce problème est ébauchée.*

ABSTRACT. *Various applications (content representation, lexical definition) use XML for encoding knowledge and communicating it to a wider context. XML is seen as a universal language which shares many features with objects. Will XML replace knowledge representation languages as we know them? Some questions and problems raised by the transcription of an object-based representation language to XML are presented through an example. The possible solutions to these are compared. The main advantage and shortcoming of XML being its lack of semantics, a solution is briefly sketched.*

MOTS-CLÉS : *XML, Représentation de connaissance par objets, Sémantique*

KEY WORDS:*XML, Object-based knowledge representation, Semantics*

Le langage XML (“eXtensible Markup Language” [BRA 1998a]) est maintenant un passage obligé pour l’utilisation des objets, et principalement de la représentation de connaissance par objets, sur le Web. Les représentations de connaissance, transcrites en XML, sont particulièrement utilisées dans la représentation du contenu des documents [GAI 1999, EUZ 1999] (indexation d’offres de voyages [MAR 1999] ou de communiqués de presse [KEN 1999]). Mais XML est aussi très important en vue du simple échange entre représentations par objets. L’“Object Management Group” ne s’y est pas trompé qui a choisi XMI (“XML Metadata Interchange” [XMI 1999]) comme langage de sérialisation des spécifications UML.
Cet article est conçu à la fois comme une introduction à XML et aux langages permettant de l’étendre, une présentation de la conception de DTD pour les représentations de connaissance par objets et une réflexion sur le développement d’un langage d’échange entre représentations de connaissance. Afin de donner plus de consistance à la réflexion, la transcription des constructions d’un langage de représentation par objets en XML sera présentée.
La première section est une très rapide introduction à XML et à l’exemple qui sera développé tout au long de l’article. La seconde section se penche sur l’encodage en XML des représentations de connaissance. Il existe plusieurs solutions qui ne résolvent pas tous les problèmes. Les problèmes rencontrés et les limitations mises en évidence permettent d’introduire les différents efforts destinés à les corriger (§3). Ces efforts et leur état actuel sont présentés avant d’en critiquer certains aspects qui ont déjà trouvé une solution dans les langages de représentation par objets. S’il semble possible, à moyen terme, d’exprimer toute la syntaxe des formalismes de représentation de connaissance dans les documents XML, il n’en va pas de même de la sémantique qui a suscité de fructueuses recherches en représentations de connaissance ces dernières années. Une dernière section (§4) est dévolue à une proposition qui permettrait, sans bouleverser le langage XML, une meilleure interopérabilité des représentations de connaissance par objets.

1. Présentation d’XML

XML [BRA 1998a] est un langage de balisage de document recommandé par le “Worldwide web consortium” (W3C). Il a pour vocation d’être un format d’échange de documents entre diverses applications. Il n’est pas un format de stockage — mais risque de le devenir —, ni un format de présentation. Un rapide point de départ est [BOS 1999] et une description plus complète en français se trouve dans [MIC 1998]. XML est délibérément intermédiaire entre HTML et SGML. Il est plus simple et moins contraignant que SGML et plus complexe et plus contraignant qu’HTML (dont l’interprétation est très tolérante puisqu’elle ignore ce qu’elle ne comprend pas). Contrairement à HTML et à l’instar de SGML, XML dispose d’un langage permettant de décrire des formats (DTD pour description de type de documents). HTML peut être, à quelques détails près, redéfini comme une DTD XML.
Les documents XML suivent une syntaxe relativement simple qui peut se résumer par :

```
<TAG att1=v1…attn=vn/>
```

ou

```
<TAG att1=v1…attn=vn>contenu</TAG>
```

où `contenu` est une suite d'expressions XML ou une chaîne de caractères, `atti` est un identificateur d'attribut et `vi` la valeur de cet attribut. Le `TAG` est appelé élément. Un document XML est vu comme un arbre dont la racine est le document et les fils d'un nœud sont les éléments de son contenu.
Dans la suite, un unique exemple sera utilisé. Il représente une protéine (nommée « BICOID »), qui a pour longueur 422 acides aminés, qui code pour le gène « Bicoid » et qui interagit avec le gène « Hunchback ». Le voici sous la forme d'un document XML (les textes XML étant donnés en police courrier et les DTD avec une barre verticale à gauche) :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE PROTEIN SYSTEM "protein.dtd">

<PROTEIN name="BICOID" length="422">
   <GENE name="Bicoid"/>
   <INTERACTION>
      <PROTEIN name="BICOID"/>
      <GENE name="Hunchback"/>
   </INTERACTION>
</PROTEIN>
```

Il représente la protéine avec deux attributs — son nom, `BICOID`, et sa longueur, 422 — et deux éléments contenus : une structure de gène nommée `Bicoid` et une autre structure qui est une interaction entre la protéine `BICOID` et le gène `Hunchback`. Il s'agit d'un exemple simple qui peut aisément être représenté dans un langage objet. Il a été, ainsi que les exemples du §2, validé par un analyseur (XML4J).

Une DTD a pour but de définir chaque élément en précisant :
— son contenu comme une expression régulière introduisant la séquence (,) et l'alternative (|) fonction d'un nombre variable (1/?/+/*) d'autres éléments ;
— ses attributs en précisant le type de valeur prise et un ensemble d'autres paramètres.
Les documents XML sont destinés à être manipulés indépendamment de leur DTD contrairement aux objets qui peuvent difficilement être exploités sans leurs classes. C'est pourquoi il est justifié de présenter le document avant sa DTD. L'exemple ci-dessus est défini par la DTD suivante :

```
<?xml  version="1.0" encoding="UTF-8" ?>
<!-- protein.dtd -->

<!ELEMENT PROTEIN (GENE?,INTERACTION*)>
<!ATTLIST PROTEIN name     CDATA    #REQUIRED
                  length   CDATA    #IMPLIED>

<!ELEMENT GENE EMPTY>
<!ATTLIST GENE    name     CDATA    #REQUIRED>

<!ELEMENT INTERACTION (PROTEIN,GENE)>
```

Celle-ci définit les trois types d'éléments visibles dans le document : `PROTEIN`, `GENE` et `INTERACTION`. Chacun de ces éléments est défini par son contenu (à l'aide du mot clef `!ELEMENT`) et ses attributs (à l'aide du mot clef `!ATTLIST`, ce dernier étant

optionnel). Un troisième mot clef, `!ENTITY`, est utilisé pour définir des macros (blocs de texte réutilisables littéralement) mais il n'est pas utile à la discussion. Le contenu est spécifié en fonction d'autres éléments et de chaînes de caractères. Les attributs peuvent être plus complexes : ici les noms sont requis (`#REQUIRED`), ils pourraient être optionnels (comme `length`, `#IMPLIED`).
En XML un document est dit :
— Bien Formé ("well-formed") s'il correspond à la syntaxe XML (c'est-à-dire si l'imbrication des chevrons et guillemets forme une expression bien parenthèsée et les éléments ouverts sont bien fermés sans entrelacement).
— Valide ("valid") s'il satisfait les contraintes exprimées dans sa DTD (ou `DOCTYPE`). Ci-dessous un tel document sera qualifié d'XML-valide.
— Schéma-valide ("Schema-valid", non normatif) s'il satisfait les contraintes exprimées dans son Schéma (voir §3.2).
L'exemple ci-dessus est bien formé vis à vis de la DTD.

2. Décrire des objets en XML
Il est commun d'entendre la réflexion «XML, c'est des objets». À l'exception de la différentiation entre contenu et attributs, XML fait naturellement penser à un langage objet. En tant que métalangage sans sémantique, XML peut être qualifié de déclaratif : il décrit une structure qui est interprétée de l'extérieur. Il est donc possible de chercher à utiliser XML comme un langage de représentation de connaissance par objets. Mais, il a certaines lacunes comme l'absence de spécialisation, de types ou de collecteurs. Plutôt que d'énumérer les différences immédiatement perceptibles, il est intéressant de considérer la transformation des éléments d'une représentation par objets en XML. Cet exercice, outre qu'il illustre le propos, met en évidence les questions que chacun doit se poser avant de réaliser une telle transformation (à des fins d'interopérabilité par exemple). Malgré la ressemblance entre les objets et les structures XML, plusieurs codages peuvent être mis en œuvre dans des contextes différents.
Le langage de représentation utilisé dans l'exemple (de type TROEPS [SHE 1999]) définit des classes qui peuvent être spécialisations les unes des autres, des attributs qui peuvent être raffinés d'une classe à l'autre et typés à l'aide de constructeurs (élément, liste, ensemble) et de types concrets (chaînes, entiers) ou d'autres classes. Un document XML sera dit RCO-valide, s'il est valide et si son contenu est considéré comme consistant par le système de représentation de connaissance (en particulier, il sera bien typé).
Deux manières de coder les objets en XML sont introduites ci-dessous :
— La DTD définit les classes et les documents représentent alors les instances (§2.1). C'est le codage naturel de l'expression « XML = objets ».
— La DTD est unique pour un système de représentation donné et les documents contiennent classes et instances (§2.2).
Une synthèse de leurs avantages et leurs inconvénients est alors proposée (§2.3) avant de s'intéresser aux traits non couverts par XML (§3).

2.1. *Solution 1 : les classes constituent le type de document*

La vision selon laquelle un document XML est une base d'objets peut être plaquée directement sur une situation où la DTD correspond à la description de classes et les documents XML sont des instances. Il y a alors deux manières de voir un élément XML comme décrivant des objets :

— Les éléments sont des objets dont le contenu correspond aux attributs. Ces objets sont ne alors pas structurés sous une forme objet puisque c'est le langage des expressions régulières qui permet de les structurer ;

— Les éléments sont des objets dont les attributs sont naturellement les attributs XML. Cependant, les types associés aux valeurs de ces attributs sont trop pauvres pour permettre de représenter les objets de manière classique[1].

Seule la première solution est raisonnablement possible dans un contexte objet car elle permet d'avoir une structure quelconque de contenu et surtout de disposer d'(ensembles d')autres objets en valeurs d'attributs.

Par ailleurs, il est obligatoire de coder les attributs à l'aide d'éléments spéciaux car sinon, il ne serait pas possible de distinguer deux éléments connexes avec le même type (si deux ensembles d'interactions sont présentés à la suite, où se termine le premier ?). Ainsi, les éléments commençant par `A` sont les attributs, ceux commençant par `R` sont des références (à d'autres objets décrits par ailleurs), ceux commençant par `V` des valeurs de types externes et les autres sont les descriptions d'objets. Voici le résultat obtenu sur l'exemple cité à la section précédente :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE PROTEIN SYSTEM "biologykb.dtd">

<PROTEIN>
   <A-PROTEIN-NAME><V-CHAINE V="BICOID"/></A-PROTEIN-NAME>
   <A-PROTEIN-LENGTH><V-ENTIER V="422"/></A-PROTEIN-LENGTH>
   <A-PROTEIN-GENE>
      <R-GENE>
         <A-GENE-NAME><V-CHAINE V="Bicoid"/></A-GENE-NAME>
      </R-GENE>
   </A-PROTEIN-GENE>
   <A-PROTEIN-INTERACTIONS>
      <R-INTERACTION>
         <A-INTERACTION-SRC>
            <R-PROTEIN>
               <A-PROTEIN-NAME><V-CHAINE V="BICOID"/></A-PROTEIN-NAME>
            </R-PROTEIN>
         </A-INTERACTION-SRC>
         <A-INTERACTION-TARGET>
            <R-GENE>
               <A-GENE-NAME><V-CHAINE V="Hunchback"/></A-GENE-NAME>
            </R-GENE>
         </A-INTERACTION-TARGET>
      </R-INTERACTION>
   </A-PROTEIN-INTERACTIONS>
</PROTEIN>
```

La DTD définissant la base de connaissance est la suivante :

---

[1] Les références à un élément peuvent se faire indirectement grâce à un mécanisme de nommage (`ID` et `IDREF`). Il s'avère que les extensions présentées ici ne l'utilisent pas, sans doute parce qu'il est plutôt adapté à des documents XML complets avec une traduction directe du document aux objets.

```
<?xml  version="1.0" encoding="UTF-8" ?>
<!-- biologykb.dtd -->

<!ELEMENT PROTEIN (A-PROTEIN-NAME,A-PROTEIN-LENGTH?,A-PROTEIN-GENE?,A-
PROTEIN-INTERACTIONS?)>
<!ELEMENT R-PROTEIN (A-PROTEIN-NAME)>
<!ELEMENT A-PROTEIN-NAME (V-CHAINE)>
<!ELEMENT A-PROTEIN-LENGTH (V-ENTIER)>
<!ELEMENT A-PROTEIN-GENE (R-GENE)>
<!ELEMENT A-PROTEIN-INTERACTIONS (R-INTERACTION+)>

<!ELEMENT GENE (A-GENE-NAME,A-GENE-PROTEIN?)>
<!ELEMENT R-GENE (A-GENE-NAME)>
<!ELEMENT A-GENE-NAME (V-CHAINE)>
<!ELEMENT A-GENE-PROTEIN (R-PROTEIN)>

<!ELEMENT INTERACTION (A-INTERACTION-SRC,A-INTERACTION-TARGET)>
<!ELEMENT R-INTERACTION (A-INTERACTION-SRC,A-INTERACTION-TARGET)>
<!ELEMENT A-INTERACTION-SRC (R-PROTEIN)>
<!ELEMENT A-INTERACTION-TARGET (R-GENE)>

<!-- external types -->
<!ELEMENT V-CHAINE EMPTY>
<!ATTLIST V-CHAINE V CDATA #REQUIRED>
<!ELEMENT V-ENTIER EMPTY>
<!ATTLIST V-ENTIER V CDATA #REQUIRED>
```

Cette première solution a été retenue par DTDMaker [ERD 1999], qui transforme une « ontologie » (en F-Logic) en une DTD, et XMI [XMI 1998], qui transforme une description en "Meta Object Facility" (dont le métamodèle UML) en une DTD. La séparation entre attributs (A-PROTEIN-GENE) et objets (GENE) résout le principal problème rencontré par DTDMaker à savoir la délimitation entre différents attributs. En outre, ces deux propositions simulent l'héritage lié à la spécialisation en utilisant la seule ressource offerte par XML : la recopie littérale de ce qui est hérité (via !ENTITY). Ainsi, le type de l'attribut A-PROTEIN-GENE ne sera pas uniquement (R-)GENE mais toutes les sous-classes de celui-ci dans l'« ontologie ».
La notion d'« ontologie » suppose ici un consensus sur une définition figée des concepts décrits et il n'existe pas d'objets à traduire, ceux-ci seront décrits directement en XML en fonction de la DTD. Cette solution a pour principal défaut de ne pouvoir accepter de modifications dynamiques des classes ce qui est couramment fait en représentation de connaissance.
Il faut enfin remarquer que toute la vérification de type est déléguée à l'analyseur XML : un document XML-valide est aussi RCO-valide.

2.2. *Solution 2 : les classes sont dans les documents*

Dans la seconde solution, les documents contiennent classes et instances et les attributs font partie du contenu des éléments. Cette seconde approche peut être qualifiée de réification de l'« ontologie ». Ici, le schéma de la base se trouve décrit dans le document XML et non dans la DTD (qui est commune à l'ensemble des bases possibles). Ceci conduit à des documents qui seront plus volumineux. Cependant, l'expérience montre que la taille d'un schéma est négligeable par rapport à la taille des données.

Ci-dessous le document correspondant à la base décrite en introduction et la DTD correspondant au langage de représentation de connaissance sont présentés. L'ensemble de la base est défini dans le document. `GENE` et `PROTEIN` sont des sous-classes de `NamedObject` (qui définit leur champ `name` de type chaîne) et `INTERACTION` est sous-classe directe d'`Object`. Des descripteurs permettent de préciser les types (`length` est un entier supérieur ou égal à 1). Tous les attributs sont décrits par un élément `ATT` (défini, comme `FIELD` et `CLASS`, par la DTD).

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE RCOKB SYSTEM "rco.dtd">

<RCOKB name="biologykb">
   <CLASS name="NamedObject" specialises="Object">
      <FIELD name="name"><TYPE id="chaine"/></FIELD>
   </CLASS>
   <CLASS name="GENE" specialises="namedObject"/>
   <CLASS name="PROTEIN" specialises="namedObject">
      <FIELD name="length">
         <TYPE id="entier"/>
         <DESCRIPTOR name="valmin">1</DESCRIPTOR>
      </FIELD>
      <FIELD name="gene"><CLASSREF name="GENE"/></FIELD>
      <FIELD name="interactions" constructor="set">
         <CLASSREF name="INTERACTION"/>
      </FIELD>
   </CLASS>
   <CLASS name="INTERACTION" specialises="Object">
      <FIELD name="source"><CLASSREF name="PROTEIN"/></FIELD>
      <FIELD name="target"><CLASSREF name="GENE"/></FIELD>
   </CLASS>
   <OBJECT class="PROTEIN">
      <ATT name="name">BICOID</ATT>
      <ATT name="length">422</ATT>
      <ATT name="gene">
         <OBJREF class="GENE"><ATT name="name">Bicoid</ATT></OBJREF>
      </ATT>
      <ATT name="interactions">
         <OBJECT class="interaction">
            <ATT name="source">
               <OBJREF class="PROTEIN">
                  <ATT name="name">BICOID</ATT>
               </OBJREF>
            </ATT>
            <ATT name="target">
               <OBJREF class="GENE">
                  <ATT name="name">Hunchback</ATT>
               </OBJREF>
            </ATT>
         </OBJECT>
      </ATT>
   </OBJECT>
</RCOKB>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- rco.dtd -->

<!ENTITY % naming "name NMTOKEN #REQUIRED">
<!ELEMENT RCOKB ((CLASS|OBJECT)*)>
<!ATTLIST RCOKB %naming;>

<!ELEMENT TYPE EMPTY>
<!ATTLIST TYPE id NMTOKEN #REQUIRED>
```

```
<!ELEMENT CLASSREF EMPTY>
<!ATTLIST CLASSREF %naming;>
<!ELEMENT CLASS (FIELD*)>
<!ATTLIST CLASS %naming; specialises NMTOKEN #IMPLIED>

<!ELEMENT FIELD ((TYPE|CLASSREF)?,DESCRIPTOR*)>
<!ATTLIST FIELD %naming; constructor (elt|set|list) "elt">

<!ELEMENT DESCRIPTOR ANY>
<!ATTLIST DESCRIPTOR %naming;>

<!ELEMENT OBJECT (CLASSREF*,ATT+)>
<!ATTLIST OBJECT class  NMTOKEN  #REQUIRED>
<!ELEMENT OBJREF (ATT+)>
<!ATTLIST OBJREF class  NMTOKEN  #REQUIRED >

<!ELEMENT ATT ANY>
<!ATTLIST ATT %naming;>
```

Il n'y a pas de typage (au niveau d'XML) dans cette proposition. Mais les éléments de typage présents dans la représentation de connaissance sont aussi présents dans le document si bien qu'un post-processeur connaissant leur sémantique pourra vérifier la RCO-validité du document.

2.3. *Première synthèse*

La seconde solution est celle à retenir pour une raison très importante : elle permet de récupérer les objets (dans un système de représentation de connaissance). Ce n'est en effet pas le cas de la première car les balises utilisées lui seront totalement inconnues et dépendantes de la base considérée. En effet, il n'y a pas dans la première solution de classes et par conséquent d'instances de ces classes : cela correspond à une base de connaissance compilée. Par contre, la seconde solution implique qu'il ne sera pas possible de déléguer l'édition, ou même la vérification de types, à un simple éditeur XML validant.

Le clivage entre les deux solutions rejoint le clivage général entre bases de données et bases de connaissance : dans les premières les données sont massivement manipulées alors que dans les secondes c'est plutôt le schéma conceptuel qui est l'objet de manipulations.

3. Extensions vers la représentation de connaissance par objets

Malgré la faisabilité et l'intérêt de la traduction présentée ci-dessus, XML est, en lui même assez éloigné d'un langage de représentation de connaissance. Nombreux sont ceux qui voudraient y ajouter les caractéristiques suivantes :

— La relation de spécialisation (base de l'héritage) qui est absente (tout comme l'héritage) des définitions d'éléments. Sans évoquer les métaclasses.

— Les types de données qui sont restreints (chaînes et énumération). Le typage est à renforcer avec des types plus diversifiés (externes) et la possibilité de raffiner les valeurs possibles (un entier positif non nul par exemple).

— Les collecteurs : s'il est possible de spécifier qu'un contenu peut être composé d'un (ou plus) élément(s), il n'est ni possible de préciser sa cardinalité, ni de spécifier qu'il s'agit d'un ensemble ou d'une liste.

— La notion de référence entre objets (et non entre parties de documents comme le fait ID/IDREF).

Toutes ces caractéristiques se trouvent dans les langages de représentation de connaissance. Quelques propositions dont le but est d'inclure ces constructeurs dans XML sont présentées ci-dessous. Elles permettent ainsi de définir une structure plus proche des représentations de connaissance par objets (et d'en vérifier le typage).

3.1. *Premières extensions*

Il existe plusieurs initiatives, liées au W3C, contribuant à rapprocher XML et les objets au sens entendu ici.

DCD ("Document content description" [BRA 1998b]) est une proposition qui vise à étendre la définition des DTD avec la spécialisation, un contenu structuré sous la forme de divers types de groupes (ensembles, multi-ensembles, listes…), des contraintes de cardinalités sur ce contenu, des attributs structurés (types, contraintes d'intervalles, valeurs par défaut), l'introduction de types externes et les valeurs nulles. Cette proposition est assez bien pensée et extensible. Voici la description DCD correspondant à l'exemple initial. Elle introduit la spécialisation entre éléments (`GENE` et `NamedElement`) et des types de données (entier positif pour `length`).

```
<DCD>

<AttributeDef Name="name" Datatype="string" Global="true"/>

<ElementDef Type="NamedElement" Model="Empty" Content="Open">
   <Attribute>name</Attribute>
</ElementDef>

<ElementDef Type="GENE" Model="Empty" Content="Open">
   <Extends>NamedElement</Extends>
</ElementDef>

<ElementDef Type="PROTEIN" Model="Mixed" Content="Open">
   <Extends>NamedElement</Extends>
   <AttributeDef Name="length" Datatype="int" Global="false" Min="0"/>
   <Group Occurs="Optional" RDF:Order="Seq">
      <Element>GENE</Element>
      <Group Occurs="ZeroOrMore" RDF:Order="Seq">
         <Element>INTERACTION</Element>
      </Group>
   </Group>
</elementDef>

<ElementDef Type="INTERACTION" Model="Elements" Content="Open">
   <Group Occurs="Required" RDF:Order="Seq">
      <Element>PROTEIN</Element>
      <Element>GENE</Element>
   </Group>
</ElementDef>

</DCD>
```

Cette spécification devrait valider le document XML donné en introduction. Elle est plus proche d'une description de représentation de connaissance que les DTD de la section précédente.

RDF (“Resource description framework” [LAS 1999]) est une autre initiative dont le but est de décrire les relations entre ressources (qui peuvent être des documents ou parties de documents) à l’aide de propriétés nommées et valuées. Son but originel est de spécifier le format des méta-données à ajouter aux documents XML. Il n’est pas considéré comme une extension d’XML vers les objets car les attributs (ou relations) sont les constructeurs principaux qui sont appliqués aux classes (RDF est donc plutôt proche des graphes conceptuels).

3.2. *XML-Schéma*

XML Schéma est un groupe de travail du W3C qui s’occupe des extensions de XML concernant les types de données (c’est-à-dire les problèmes évoqués jusqu’à présent).

Les résultats de ses travaux ont été développés dans deux propositions préliminaires :

— Schema-Structure [THO 1999] donne la définition d’un schéma permettant d’associer aux éléments des DTD des structures plus complexes dont le type des attributs est plus précis;

— Schema-Data [BIR 1999] permet de définir ses propres types de données à inclure dans les schémas.

Plusieurs avantages sont espérés de ces extensions dont la simplification des DTD ci-dessus grâce à l’utilisation d’attributs XML typés et plus généralement une plus grande expressivité des DTD permettant aux analyseurs XML un contrôle de type plus précis sur des types plus étendus que ceux d’XML.

Voici un exemple de Schéma qui définit deux types de données (`name` et `proteinName` qui s’écrit en majuscule) l’un étant dérivé de l’autre, deux types d’éléments (nommés archétypes : `NamedElement` et `NamedSizedElement`) l’un étant dérivé de l’autre et trois sortes d’éléments (`GENE`, `PROTEIN` et `INTERACTION`).

```
<?xml version="1.0" ?>
<!DOCTYPE schema PUBLIC "-//W3C//DTD XMLSCHEMA 19991105//EN"
   "http://www.w3.org/TR/1999/WD-xmlschema-1-19991105/structures.dtd">

<schema>
<include schemaName="datatypes.xsd">

<!-- datatypes -->

<datatype name="name">
   <basetype name="string"
            schemaName="http://www.w3.org/xmlschemas/datatypes"/>
   <maxLength>10</maxLength></datatype>

<datatype name="ProteinName" type="name" pattern="[A-Z]{1-10}"/>

<!-- archetypes -->

<archetype name="namedEntity" model="refinable">
   <attribute name="name" required="true" type="name"/>
</archetype>
```

```
<archetype name="namedSizedEntity" model="refinable">
   <refines name="namedEntity"/>
   <attribute name="length" type="integer" default="0"/>
</archetype>

<!-- elements -->

<element name="GENE" archRef="namedEntity"/>

<element name="PROTEIN">
   <archetype>
      <refines name="namedSizedEntity"/>
      <attribute name="name" type="ProteinName"/>
      <group order="choice" minOccurs="0" maxOccurs="1">
         <group order="seq">
            <element ref="GENE"/>
            <group order="all" minOccurs="0" maxOccurs="*">
               <element ref="INTERACTION"/>
            </group>
         </group>
      </group>
   </archetype>
</element>

<element name="INTERACTION">
   <archetype order="seq">
      <element ref="PROTEIN"/>
      <element ref="GENE"/>
   </archetype>
</element>

</schema>
```

Le document correspondant à l'exemple initial décrit dans le schéma ci-dessus est inchangé (à ceci près qu'il faudra préciser qu'il est régit par ce schéma). À l'instar de la transformation de la section précédente, il est possible d'exporter des objets vers XML-Schema. Un document Schéma-valide à la différence d'un document simplement valide n'est plus seulement bien structuré mais il est aussi bien typé. Par ailleurs, les attributs peuvent avoir des valeurs par défaut. Par exemple, si `GENE` est déclaré comme `NamedSizedElement`, alors sa taille est 0.

Par contre il manque encore beaucoup à XML-Schema pour approcher une représentation de connaissance par objets :

— l'héritage est ici de la réutilisation dans les archétypes (mais il n'y a pas de spécialisation entre éléments) ;

— les collecteurs (ensemble, liste) sont toujours absents,

— il y a clairement dans XML-Schema la possibilité de raffiner les types existants en utilisant des propriétés abstraites de longueur ou d'ordre (et même la cardinalité des ensembles supports), mais il n'est pas possible d'ajouter de nouveaux types comme cela l'est dans un langage comme TROEPS qui ne dispose pas de types primitifs. Pour cela il faudrait pouvoir définir un prédicat d'appartenance ou d'ordre par exemple [CAP 1995]. Ceci suppose de pouvoir donner une définition exécutable des types de données. Les concepteurs d'XML-Schéma ont évité cela (de manière à ne pas faire allégeance à un langage particulier). C'est d'autant plus dommage que la "sérialisation" telle que promue par Java est un excellent point de départ pour l'intégration de types externes à XML.

3.3. *SHOE, OML… : la RCO n'est pas loin*

Ces extensions d'XML ne sont pas suffisantes pour la représentation de connaissance. D'autres DTD, plus spécifiques, permettent donc d'exprimer des objets sans forcément être considérées dans le champ du W3C.

SHOE ("Simple HTML Ontology Extensions" [HEF 1998]) bien que lié à HTML (comme son nom l'indique) est immédiatement transposable en XML. Il permet de définir des hiérarchies de classes munies d'attributs typés et des règles de déduction de type clauses de Horn. Son but est d'intégrer aux pages HTML une représentation formelle du contenu accessible à des agents. L'expressivité du langage est relativement réduite et la sémantique reste informelle.

OML ("Ontology Markup Language" [KEN 1999]) a pour but d'introduire les graphes conceptuels en XML. En fait, c'est aussi un encodage du calcul des prédicats du premier ordre. Il manipule les notions de classes, d'objets, de relations et de fonctions, mais aussi de quantificateurs et de connecteurs booléens. Son expressivité est donc très importante. CKML ("Conceptual Knowledge Markup Language" [KEN 1999]) est un sur-ensemble d'OML permettant la représentation des contextes, des treillis de concepts et de séquents. C'est une option maximaliste dont le but principal est d'échanger de l'information entre applications (en communiquant le contexte), de faire de la construction de taxonomies à base de treillis de Galois et de la recherche d'information.

Il existe évidement d'autres initiatives telles que la DTD de TROEPS ou celle de FaCT [BEC 1999]. À noter qu'à l'instar de la solution 2 retenue ci-dessus ces extensions d'XML sont des DTD et les schémas conceptuels (ou les « ontologies ») sont définis dans les documents XML eux-mêmes.

3.4. *Seconde synthèse*

Il existe donc différentes initiatives pour pousser plus loin l'identification entre XML et représentation par objets. Leur principal problème est de focaliser d'emblée sur un type de langage et de réduire l'ouverture inhérente à XML. En contrepartie, ils peuvent définir de manière très précise la sémantique des éléments.

Ces efforts contribuent à l'expression dans les DTD du maximum d'information et par conséquent à la délégation à l'analyseur XML de plus en plus de tâches. Il en résulte que, progressivement, les tâches d'un système de représentation de connaissance y seront intégrées. Cette évolution est présentée ainsi :

	XML	XML-Schéma	XML-OML
XML	Échange Analyse syntaxique	Échange Analyse syntaxique	Échange Analyse syntaxique
RCO	Typage Inférence	Typage Inférence	Typage Inférence

Tableau 1 : *Répartition des tâches entre XML et représentation de connaissance dans les extensions présentées ci-dessus.*

Ainsi, sur les aspects syntaxiques, l'évolution d'XML le rapproche d'un langage de représentation de connaissance par objets. Mais la dernière colonne du tableau ci-dessus n'offre plus réellement un métalangage permettant d'échanger entre systèmes différents, elle exige de connaître la sémantique des objets manipulés.
Le résultat obtenu par la colonne du milieu, même s'il peut être critiqué n'est pas à négliger. Il permet effectivement de définir des DTD qui comprennent un sous-ensemble important de la syntaxe des langages à objets. Ce n'est pas le cas de la sémantique. C'est principalement pourquoi XML n'est pas un langage de représentation de connaissance.

4. Décrire la sémantique pour l'exporter

La question de l'extensibilité d'XML vers les fonctions les plus avancées de représentation de connaissance mérite d'être posée. Pouvoir donner une sémantique à un document XML est d'autant plus important que l'avènement d'un « web sémantique » [BER 1998] est annoncé (et que chacune des extensions précitées promettent de mettre « plus de sémantique » dans XML).
Il existe donc une tension entre XML et représentation par objets : celle de l'ouverture face à la transparence sémantique. Cette tension semble irréconciliable. Cependant, il est imaginable de donner aux DTD XML une sémantique par le biais d'un langage de définition de celle-ci. En se restreignant à la théorie des modèles dans un domaine ensembliste, il semble possible de produire une description du type de ce qui suit :

$$I(\texttt{CLASS}) \subseteq I(D) \cap I(\texttt{@specialises}) \cap \bigcap_{f \in \texttt{FIELD}} \left\{ i \in D; I(f\texttt{@name})(i) \in I(f.\texttt{DESCRIPTOR}) \right\}$$

Cette description signifie que chaque classe est interprétée comme un sous-ensemble de l'intersection du domaine D avec l'interprétation de sa super-classe et de l'ensemble des individus pour lesquels l'application de l'interprétation de chacun des attributs retourne un élément satisfaisant l'interprétation des descripteurs de l'attribut dans la classe. Elle peut s'exprimer en XML dans un langage mêlant notations ensemblistes (MathML) et références au contenu (XPath).
L'échange des données, non seulement avec leur format, mais encore avec leur sémantique permettrait, pour l'ordinateur, de manipuler les données suivant la sémantique qui leur a été donnée par leur auteur. Par exemple, des transformations de langage à langage peuvent être appliquées en prenant en compte la sémantique (ne serait-ce qu'en vérifiant les transformations définies manuellement). Il est concevable d'importer en TROEPS des définitions de concepts de logiques terminologiques sachant qu'elles ne seront interprétées que comme des descriptions et inversement, d'importer dans un système terminologique des descriptions TROEPS sachant qu'elles doivent être interprétées comme primitives.
Il y a plusieurs langages d'échange entre systèmes de représentations de connaissance ; KIF ("Knowledge Interchange Format" [GEN 1992]) est sans doute le plus notable. Le défaut de ce type de langage est de contraindre non seulement à adopter une syntaxe mais aussi une sémantique particulière. Ainsi, traduire entre deux langages de faible expressivité nécessite le passage par un langage très

expressif. Le même type de problème se pose avec les langages de communication entre agents qui sont trop contraignants lorsqu'ils imposent une sémantique et pas assez lorsqu'ils n'en imposent pas. L'intérêt du dispositif présenté est que l'exportation se fait dans un format qui préserve la structure des données (et non seulement la sémantique). Ainsi, les expressions exportées pourront être réimportées sans perte d'information. Parallèlement, l'interopérabilité est facilitée par l'exportation de la sémantique qui permet de mettre en correspondance deux langages. Bien entendu, une telle interopérabilité est au prix du développement d'un langage de description de sémantique complexe (dont la manipulation de manière autonome nécessiterait pour le moins un démonstrateur en théorie des ensembles).

5. Conclusion

Le compromis entre expressivité et complexité est particulièrement développé dans le domaine de la représentation de connaissance. Il semble que la solution soit dans la modularité des formalismes permettant de supporter uniquement la complexité nécessaire aux expressions manipulées. XML se veut un métalangage d'échange. À ce titre, il est d'autant mieux armé pour jouer ce rôle entre les différents formalismes de représentation de connaissance que la modularité (et la possibilité d'importer) et l'adressage (évitant les conflits de noms) des DTD ont été bien pensés.

Il est donc important qu'XML reste un langage extensible dynamiquement de manière à ne pas le brider (en empêchant son extensibilité) ni le plomber (en imposant des contraintes inutiles aux implémentations).

Mais XML, comparé aux représentations de connaissance par objets, possède les défauts de ses qualités. En particulier, le typage, l'agrégation et la spécialisation en sont absents. De nombreuses initiatives visent à combler ces lacunes. Ainsi, sur l'aspect syntaxique, seul l'accès à des types de données externes n'est pas complètement couvert afin de préserver la portabilité du langage.

Il n'en va pas de même de l'aspect sémantique : l'ouverture d'XML lui interdit de définir une sémantique intrinsèque aux structures manipulées. Les initiatives pour associer une sémantique précise au langage se cantonnent à un sous-langage et nuisent à l'interopérabilité (car il y a peu de chances que cette sémantique devienne standard à l'échelle du Worldwide web). Cela a conduit à une ébauche de solution permettant de définir non seulement la syntaxe des documents XML mais aussi leur sémantique de la même manière que cela est fait en représentation de connaissance.

Ces premières réflexions sont issues de la conception de DTD pour le système TROEPS [SHE 1995], un système de représentation de connaissance par objets utilisé dans plusieurs applications dans lesquelles le transfert des objets en XML est utile (annotation de pages HTML pour la recherche d'information ou liaison entre bases de connaissance et lexique). Pour cela, il est doté d'une interface XML, en entrée comme en sortie, fondée sur la seconde solution. La DTD correspondante est bien plus complète que celles présentées ci-dessus. Elle peut être obtenue (ainsi que plusieurs exemples) en

`http://co4.inrialpes.fr/xml/`.

Pour résumer, cet article exhibe un contre-exemple à la transitivité de la relation « aime » puisque si X*m*L et L*m*O, cela n'entraîne pas X*m*O (pas plus que X=O).

Remerciements
Ce travail a été partiellement réalisé dans le cadre du programme GENIE II soutenu par le MENRT et la DGA/SPAé et a bénéficié du soutien de l'action de recherche coopérative ESCRIRE de l'INRIA/France Telecom. L'auteur remercie Pierre-Antoine Champin, Raphaël Troncy et Emmanuel Nauer qui ont bien voulu commenter une précédente version.

6. Références

[BEC 1999] BECHHOFER S., HORROCKS I., PATEL-SCHNEIDER P., TESSARIS S., A proposal for a description logic interface, Actes int. workshop on description logics, Linköping (SE), CEUR-WS-22, 1999. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-22/bechhofer.ps

[BER 1998] BERNERS-LEE T., Semantic web roadmap, 1998. http://www.w3.org/DesignIssues/Semantic.html

[BIR 1999] BIRON P., MALHOTRA A. (éds.), XML Schema part 2: datatypes, Working draft, W3C, 1999. http://www.w3.org/TR/XMLschema-2/

[BOS 1999] BOSAK J., BRAY T., Le XML, Pour la science 261, 1999. http://www.pourlascience.com/numeros/pls-261/art-4.htm

[BRA 1998a] BRAY T., PAOLI J., SPERBERG-MCQUEEN C. (éds.), Extensible Markup Language (XML) 1.0, Recommendation, W3C, 1998. http://www.w3.org/TR/REC-XML

[BRA 1998b] BRAY T., FRANKSTON C., MALHOTRA A. (éds.), Document Content Description for XML, Submission, W3C, 1998. http://www.w3.org/TR/NOTE-dcd

[CAP 1995] CAPPONI C., Identification et exploitation des types dans un modèle de représentation des connaissances par objets, Thèse d'informatique, Université Joseph-Fourier, Grenoble (FR), 1995.

[ERD 1999] ERDMAN M., STUDER R., Ontologies as conceptual models for XML documents, Actes 12th KAW, Banff (CA), 1999.

[EUZ 1999] EUZENAT J., La représentation de connaissance est-elle soluble dans le Web?, *Document numérique* 3, 1999 à paraître.

[GAI 1999] GAINES B., SHAW M., Embedding knowledge models in active documents, *Communication of the ACM* 42(1):55-63, 1999.

[GEN 1992] GENESERETH M., FIKES R., Knowledge interchange format, version 3.0 - reference manual, Rapport de recherche Logic-92-1, Stanford university, Stanford (CA US), 1992.

[HEF 1998] HEFLIN J., HENDLER J., LUKE S., ZHENDONG Q. SHOE: a knowledge representation language for internet applications, submitted 1998. http://www.cs.umd.edu/projects/plus/SHOE/aij-shoe.ps

[KEN 1999] KENT R., Conceptual Knowledge Markup Language, Actes 12th KAW, Banff (CA), 1999.

[LAS 1999] LASSILA O., SWICK R. (éds.), Resource Description Framework (RDF) Model and syntax specification, Recommendation, W3C, 1999. http://www.w3.org/TR/REC-rdf-syntax

[MAR 1999] MARTIN P., EKLUND P., Embedding Knowledge in Web Documents, Actes 8th International World Wide Web Conference, Toronto (CA), 1999. http://www8.org/w8-papers/3b-web-doc/embedding/embedding.html

[MIC 1998] MICHARD A., XML: langage et applications, Eyrolles, Paris (FR), 1998.

[THO 1999] THOMPSON H., BEECH D., MALONEY M., MENDELSOHN N. (éds.), XML Schema part 1: structures, Working draft, W3C, 1999. http://www.w3.org/TR/XMLschema-1/

[SHE 1995] PROJET SHERPA, TROEPS 1.0 reference manual, Rapport interne, INRIA Rhône-Alpes, Grenoble (FR), 1995 (rev. 1.3, 1999). http://co4.inrialpes.fr/docs/tropes-manual.html ftp://ftp.inrialpes.fr/pub/sherpa/rapports/tropes-manual.ps.gz

[XMI 1998] XMI Consortium, XML Metadata Interchange, submitted to OMG OA&DTF RFP3: Stream-based model interchage format, 1998. ftp://ftp.omg.org/pub/docs/ad/98-10-05.ps