Génie II — Tâche 3.3.1

Intégration de connaissances modélisées et de connaissances textuelles

Intégration objets-termes-textes via XML

3/09/1999

Génie II — Tâche 3.3.1
Intégration de connaissances modélisées et de connaissances textuelles
Intégration objets-termes-textes via XML

Jérôme Euzenat[*]
INRIA Rhône-Alpes

Contexte

Le but du présent document est de décrire les moyens de transférer des données entre un gestionnaire de lexique (XTERM) et un système de représentation de connaissance (TROEPS). Cette communication est destinée à rester très peu contrainte de telle sorte à pouvoir remplacer aisément un des éléments. Elle se fait par le biais du langage XML.
Dans ce document, on présente brièvement le langage XML (§1) avant de se pencher sur le codage d'une représentation de connaissance par objets en XML (§2) et les éléments qui manquent à XML pour pouvoir assurer pleinement ce codage (§3). Quelques éléments sur les DTD utilisées par TROEPS sont ensuite détaillées (§4). Le principe de communication, via XML, entre XTERM et TROEPS est alors présenté (§5). et l'implémentation mise en œuvre dans le cadre du projet est détaillée et analysée (§6).

1 Quelques éléments sur XML

XML est présenté ci-dessous tout d'abord par ses principes fondateurs (§1.1) puis par un exemple qui sera développé tout au long de l'article (§1.2) et, enfin, par le langage de transformation de documents XML (XML, §1.3).

1.1 Principe

XML [Bray 1998a] est un langage de balisage de document recommandé par le "Worldwide web consortium" (W3C). Un rapide point de départ est [Garshol 1999] et une description plus complète en français [Michard 1998]. Il est délibérément intermédiaire entre HTML et SGML. Il est plus simple et moins contraignant que SGML et plus complexe et plus contraignant qu'HTML. Contrairement à HTML et à l'instar de SGML, XML dispose d'un langage permettant de décrire des formats (DTD). HTML peut être redéfini comme une DTD XML.
XML 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. Les documents XML suivent une syntaxe relativement simple que l'on peut 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.

Une DTD a pour but définir chaque élément en précisant :
— son contenu comme une expression régulière fonction d'autres éléments (ou vide pour le premier) ;
— ses attributs en précisant le type de valeur prise et un ensemble d'autres paramètres.

[*] Jerome.Euzenat@inrialpes.fr

Des exemples de documents XML et de DTD sont donnés ci-dessous.

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`). On dira ci-dessous qu’un tel document est XML-valide.
— Schéma-valide (“Schema-valid”, non normatif) s’il satisfait les contraintes exprimées dans son Schéma (voir §3.2).

1.2 Un exemple

Par souci de complétude, le petit exemple présenté est complet (cependant, pour des raisons de simplicité, la spécification des espaces de nommage [Bray& 1999] n’est pas développée ici). Ces solutions ont été validées par les analyseurs adéquats (XML4J).

Voici un exemple de document bien formé XML : qui sera pris comme unique exemple dans la suite (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 une protéine ayant deux attributs — son nom, `BICOID`, et sa longueur, 422 — et contenant deux éléments : 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 que l’on peut aisément représenter dans un langage objet. Il est aussi bien formé vis-à-vis de 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)>
```

La DTD est encore plus simple que le texte XML. Elle définit les trois types d’éléments visibles dans le documents : `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 (blocks 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`) ou même hérités de l'élément englobant.
À l'exception de la différentiation entre contenu et attributs, on voit aisément pourquoi XML fait penser à un langage objet. Mais nous reviendrons sur ce sujet dans les sections suivantes.

1.3 XML et présentation (XSL)

XML exprime la structure d'un document et non sa présentation. Pour cela, on utilise XSL ("XML Stylesheet Language"). XSL est composé de deux parties :

— Un langage [Clark 1999] permettant d'exprimer des transformation d'un document XML en un autre document XML (ou autre). Ces transformation sont exprimées à l'aide d'un langage de règles de réécriture sur la structure du document considéré.

— Un langage d'objets de formatage [Deach 1999] permettant d'exprimer un formatage abstrait des documents (qui pourra être raffiné par un utilisateur ou le type de dispositif d'affichage).

L'idée principale est d'utiliser le premier pour obtenir un document au format du second. Cependant, les formateurs capables de lire le langage de formatage n'étant pas disponibles, on pourra engendrer du HTML. Ainsi, le document XML précédent peut produire le HTML suivant (analogue à ce que TROEPS produit pour représenter un objet) :

```
<HTML><HEAD>
	<TITLE>PROTEIN BICOID</TITLE>
</HEAD><BODY BGCOLOR="#ffffff">
<H1>PROTEIN BICOID</H1>
	<UL>
		<LI>size = 422</LI>
		<LI>gene = <A HREF="../genes/bicoid.html">Bicoid</A></LI>
		<LI>interactions = <UL>
			<LI><A HREF="BICOID-hunchback.html">BICOID-hunchback</A></LI>
</UL>
		</LI>
	</UL>
</BODY></HTML>
```

Les instructions XSLT correspondantes se trouvent ci-dessous :

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

<xsl:template match="/">
	<HTML><HEAD>
		<TITLE>PROTEIN <xsl:value-of select="PROTEIN/@name"></TITLE>
	</HEAD><BODY BGCOLOR="#ffffff">
		<xsl:apply-templates>
	</BODY></HTML>
</xsl:template>

<xsl:template match="PROTEIN">
	<H1>PROTEIN <xsl:value-of select="@name"></H1>
	<UL>
		<LI>size = <xsl:value-of select="@length"></LI>
		<LI>gene = <xsl:apply-templates select="GENE"/></LI>
		<xsl:if test="INTERACTION">
			<LI>interactions = <UL>
				<xsl:for-each select="INTERACTION">
					<LI><xsl:apply-templates select="INTERACTION"/></LI>
</UL></LI>
```

```
                </xsl:if>
        </UL>
</xsl:template>

<xsl:template match="GENE">
        <A HREF="{@name}.html"><xsl:value-of select="@name"/></A>
</xsl:template>

<xsl:template match="INTERACTION">
        <A HREF="{PROTEIN/@name}-{GENE/@name}.html"/>
                <xsl:value-of select="PROTEIN/@name"/>-
                <xsl:value-of select="GENE/@name"/></A>
</xsl:template>
```

Ce langage permet de spécifier comment réécrire un élément (`template`), comment sélectionner un sous-élément (`select`) et comment appliquer la réécriture sur ces sous-éléments (`apply-templates`).
On peut donc dire qu'XML est un langage déclaratif. Il est même possible d'interpréter le même document XML à l'aide de DTD différentes, et surtout de le manipuler à l'aide de feuilles de styles différentes (par exemple pour engendrer une interface d'édition). Même si — presque — personne ne songe à programmer en XML, la puissance d'XSLT permet de réaliser des manipulations qui dépassent amplement le simple formatage comme l'assemblage de sources de données dispersées ou la construction de tables des matières par exemple.

2 Décrire des objets en XML

L'exemple ci-dessus montre bien en quelle mesure, il est possible de décrire des objets dans un document XML. Mais, il est possible de remarquer d'emblée certaines lacunes par rapport aux constructeurs des représentations de connaissance par objets (comme l'absence de spécialisation, de types ou de collecteurs). Plutôt que d'énumérer les différences que l'on peut y voir a priori, voyons comment il est possible de transformer les éléments d'une représentation par objets en XML. Cet exercice mets en évidence les questions que chacun doit se poser avant de réaliser une telle transformation. Malgré la ressemblance entre les objets et les structures XML, plusieurs codages peuvent être pris en compte dans des contextes différents.
On présente d'abord (§2.1), très brièvement, le langage TROEPS, avant d'aborder les transformations mises en œuvre. Deux manières de coder les objets en XML sont introduites ci-dessous :

— La DTD définit les classes et les documents représentent alors les instances (§2.2). 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 (§2.3).

Une synthèse de leurs avantages et leurs inconvénients est alors proposée (§2.4) avant de s'intéresser aux traits non couverts par XML (§3).

2.1 TROEPS

TROEPS [Mariño& 1990, Sherpa 1995] est un système de représentation de connaissance par objets développé par nos soins. Il est 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 bases de connaissance et lexique). Pour cela diverses traductions ont été évaluées et le système est donc pris comme exemple ici.
Une base TROEPS se compose d'un ensemble d'objets qui sont répartis en concepts totalement étanches (mais pouvant se référer les uns aux autres par le biais des attributs comme ici la protéine se réfère au gène). La séparation en concept indépendant régnants sur leur propre espace de nom et la structure des objets, permet de proposer plusieurs taxonomies de classes sur un concept (qualifiées de points de vue). Les objets sont très typés en TROEPS : chaque attribut voit sa valeur

déterminée par un constructeur (liste, ensemble, élément) et un type qui est soit un concept, soit un type externe au modèle décrit par un type abstrait de donnée [Capponi 1995].
Un document XML sera dit RCO-valide, s'il est valide et si son contenu est considéré comme consistant par TROEPS (en particulier, il sera bien typé).

2.2 Solution 1

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 alors assez peu 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. Cependant, les types associés aux valeurs des 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. 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>
```

Ci-dessus, il est obligatoire de coder les attributs à l'aide d'éléments spéciaux car sinon, il ne serait pas possible de distinguer deux attributs connexes avec le même type (si deux ensembles d'interactions sont présentés à la suite, où se termine le premier ?). Les éléments commençant par `A` sont les attributs, ceux commençant par `R` sont des références, ceux commençant par `V` des valeurs de types externes et les autres sont les descriptions d'objets.
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`). Mais, il s'avère que les extensions présentées ici ne l'utilisent pas. Sans doute car il est plutôt adapté à un contexte ou les documents XML sont complets et où il y a une traduction directe du document aux objets.

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

<!ELEMENT BIOKB (PROTEIN|GENE|INTERACTION)*>
<!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 [Erdman& 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 en `A-X` et `X` résout le principal problème rencontré par DTDMaker. 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-GENE` ne sera pas uniquement `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 dynamique des classes ce qui est couramment fait en représentation de connaissance.
On remarque enfin que toute la vérification de type est déléguée à l’analyseur : un document XML-valide est aussi RCO-valide.

2.3 Solution 2

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.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE TRPKBDSC SYSTEM "http//co4.inrialpes.fr/xml/troeps.dtd">

<TRPKBDSC name="biologykb" xml:lang="FR-fr">
       <CONCEPTDSC name="proteine">
              <KEYDSC>
                     <KFIELDDSC name="name" nature="property" constructor="un">
                            <TYPEREF name="chaine"/>
                     </KFIELDDSC>
              </KEYDSC>
```

```
		<KFIELDDSC name="length" nature="property" constructor="un">
			<TYPEREF name="entier"/>
		</KFIELDDSC>
		<KFIELDDSC name="gene" nature="property" constructor="un">
			<CONCEPTREF name="gene"/>
		</KFIELDDSC>
		<KFIELDDSC name="interactions" nature="property"
				constructor="set">
			<CONCEPTREF name="interaction"/>
		</KFIELDDSC>
		<OBJDSC>
			<ATTVAL name="name"><VAL>BICOID</VAL></ATTVAL>
			<ATTVAL name="length"><VAL>422</VAL></ATTVAL>
			<ATTVAL name="gene">
				<OBJREF>
					<CONCEPTREF name="gene"/>
						<ATTVAL name="name"><VAL>Bicoid</VAL></ATTVAL>
				</OBJREF>
			</ATTVAL>
			<ATTVAL name="interactions">
				<OBJREF>
					<CONCEPTREF name="interaction"/>
					<ATTVAL name="source">
						<OBJREF>
							<CONCEPTREF name="proteine"/>
							<ATTVAL
name="name"><VAL>BICOID</VAL></ATTVAL>
						</OBJREF>
					</ATTVAL>
					<ATTVAL name="cible">
						<OBJREF>
							<CONCEPTREF name="gene"/>
							<ATTVAL
name="name"><VAL>Hunchback</VAL></ATTVAL>
						</OBJREF>
					</ATTVAL>
				</OBJREF>
			</ATTVAL>
		</OBJDSC>
	</CONCEPTDSC>
</TRPKBDSC>
```

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

<!ENTITY % naming "name NMTOKEN #REQUIRED">
<!ELEMENT TRPKBDSC (CONCEPTDSC*)>
<!ATTLIST TRPKBDSC %naming;>

<!-- External types and values -->
<!ELEMENT trp:TYPEREF EMPTY>
<!ATTLIST trp:TYPEREF %naming;>

<!ELEMENT CONCEPTREF EMPTY>
<!ATTLIST CONCEPTREF %naming;>
<!ELEMENT CONCEPTDSC (KEYDSC,KFIELDDSC*,VIEWDSC*,BRIDGEDSC*,OBJDSC*)>
<!ATTLIST CONCEPTDSC %naming;>

<!ELEMENT KEYDSC (KFIELDDSC+)>
```

```
<!ELEMENT KFIELDDSC (CONCEPTREF|trp:TYPEREF)>
<!ATTLIST KFIELDDSC %naming;
                    nature (property|component|link) "property"
                    constructor (un|set|list) "un">

<!ELEMENT VIEWDSC (CLASSDSC,CLASSDSC*)>
<!ATTLIST VIEWDSC %naming;>
<!ELEMENT VIEWREF (CONCEPTREF)>
<!ATTLIST VIEWREF %naming;>

<!ELEMENT CLASSDSC (CLASSREF,CFIELDDSC*)>
<!ATTLIST CLASSDSC %naming;>
<!ELEMENT CLASSREF (VIEWREF)>
<!ATTLIST CLASSREF %naming;>

<!ELEMENT CFIELDDSC (DESCRIPTORDSC*)>
<!ATTLIST CFIELDDSC %naming;>

<!ELEMENT DESCRIPTORDSC (VAL)>
<!ATTLIST DESCRIPTORDSC %naming;>

<!ELEMENT BRIDGEDSC (CLASSREF,CLASSREF+)>
<!ATTLIST BRIDGEDSC %naming;>

<!ELEMENT OBJDSC (CLASSREF*,ATTVAL+)>
<!ELEMENT OBJREF (CONCEPTREF,ATTVAL+)>

<!ELEMENT ATTVAL (VAL+|OBJREF+)>
<!ATTLIST ATTVAL %naming;>

<!ELEMENT VAL ANY>
```

Ici, le schéma de la base se trouve décrit dans le document XML et non dans la DTD. Ceci conduit à des documents qui seront plus volumineux. Cependant, notre expérience montre que la taille d'un schéma en TROEPS est négligeable par rapport à la taille des données.
Contrairement à ce que l'on pourrait penser il n'y a pas de typage (au niveau d'XML) dans cette proposition. Mais les éléments de typage présents dans TROEPS sont aussi présents dans le document si bien qu'un post-processeur connaissant leur sémantique pourra facilement vérifier la correction du document (la TROEPS-validité). C'est d'ailleurs ainsi que fonctionne l'analyseur syntaxique actuel de TROEPS (écrit en YACC) : il ne fait que vérifier une TROEPS-bonne-forme et passe la main à des routines qui vérifient le typage et les références.

2.4 Première synthèse

La seconde solution est celle à retenir pour une raison très importante : elle permet de récupérer les objets (sous un système de représentation de connaissance). Ce n'est en effet pas le cas de la première car les tags utilisés lui seront totalement inconnus et dépendant de la base de donnée considérée. En effet, il n'y a pas dans la première solution de classes de concepts et par conséquent d'instances de ce concept : le résultat correspondra à une base de connaissance compilée. Par contre, la seconde solution implique qu'il ne sera pas possible de déléguer à un simple éditeur XML validant l'édition ou même la vérification de types.
Le clivage que l'on rencontre 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 l'apparent succès de la traduction présenté 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. On ne parlera pas des 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).

Toutes ces caractéristiques se trouvent dans les langages de représentation de connaissance et il est assez naturel que leurs utilisateurs aient proposé leurs solutions. On présente donc ci-après, quelques propositions dont le but est d'inclure ces constructeurs dans XML permettant ainsi de définir une structure plus proche de celle des représentations de connaissance par objets (et d'en vérifier le typage).

3.1 Premières extensions

Il existe un certain nombre d'initiatives, liées au W3C, contribuant à rapprocher XML et les objets au sens où nous l'entendons.

DCD ("Document content description" [Bray& 1998b]) est une proposition jointe d'IBM et de Microsoft qui vise à étendre la définition des DTD avec la spécialisation, un contenu structuré sous la forme de divers types de groupes (ensemble, multi-ensembles, listes…), des contraintes de cardinalités sur ce contenu (0/01/+/*), 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. On voit ci-dessous la description DCD correspondant à l'exemple initial.

```
<DCD>

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

<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>
       <Group Occurs="Optional" RDF:Order="Seq">
              <ElementDef Type="Data" Model="Data" Datatype="int" Min="0"/>
              <Element>GENE</Element>
              <Group Occurs="ZeroOrMore" RDF:Order="Seq">
                     <Element>INTERACTION</Element>
              </Group>
       </Group>
</elementType>

<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" [Lassila& 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). Par ailleurs, les traits de spécialisation et de typage décrits dans les schémas RDF [Brickley& 1999] sont plus simple que ceux permis par les schémas XML.
Voici l'exemple original exprimé en RDF ainsi qu'un Schéma RDF lui correspondant.

```
<rdf:RDF>
        <Protein rdf:ID="BICOID"/>
        <Gene rdf:ID="Bicoid"/>
        <Interaction rdf:ID="IN001"/>

        <rdf:Description rdf:about="#BICOID">
                <name>BICOID</name>
                <length>422</length>
                <gene rdf:resource="#Bicoid">
                <interactions>
                        <Seq><rdf:li rdf:resource="#IN001"/></Seq>
                </interactions>
        </rdf:Description>
        <rdf:Description rdf:about="#Bicoid">
                <name>Bicoid</name>
        </rdf:Description>
        <rdf:Description rdf:about="#IN001">
                <source rdf:resource="#BICOID"/>
                <cible rdf:resource="#hunchback"/>
        </rdf:Description>
</rdf:RDF>
```

```
<?xml version="1.0" encoding="UTF-8" ?>

<rdfs:Class rdf:ID="Protein">
        <rdfs:subClassOf rdf:resource="rdf-syntax-ns#Class">
</rdfs:Class>
<rdfs:Class rdf:ID="Gene">
        <rdfs:subClassOf rdf:resource="rdf-syntax-ns#Class">
</rdfs:Class>
<rdfs:Class rdf:ID="Interaction">
        <rdfs:subClassOf rdf:resource="rdf-syntax-ns#Class">
</rdfs:Class>

<rdfs:Property rdf:ID="name">
        <rdfs:domain rdf:resource="PR-rdf-schema#Class"/>
        <rdfs:range rdf:resource="rdf-syntax-ns#Litteral"/>
</rdfs:Property>
```

```
<rdfs:Property rdf:ID="length">
       <rdfs:domain rdf:resource="#Protein"/>
       <rdfs:range rdf:resource="rdf-syntax-ns#Integer"/>
</rdfs:Property>

<rdfs:Property rdf:ID="gene">
       <rdfs:domain rdf:resource="#Protein"/>
       <rdfs:range rdf:resource="#Gene"/>
</rdfs:Property>

<rdfs:Property rdf:ID="interactions">
       <rdfs:domain rdf:resource="#Protein"/>
       <rdfs:range rdf:resource="#Gene"/>
</rdfs:Property>

<rdfs:Property rdf:ID="source">
       <rdfs:domain rdf:resource="#Interaction"/>
       <rdfs:range rdf:resource="#Protein"/>
</rdfs:Property>

<rdfs:Property rdf:ID="target">
       <rdfs:domain rdf:resource="#Interaction"/>
       <rdfs:range rdf:resource="#Gene"/>
</rdfs:Property>
```

Il existe ou a existé d'autres propositions non détaillées ici : DDML, SOX, XDR, XML-Data (cité dans [Malhotra& 1999]).

3.2 XML-Schéma

XML Schéma [Malhotra& 1999] est un groupe de travail du W3C qui s'occupe des extensions de XML concernant les types de donnés (c'est-à-dire les problèmes évoqués jusqu'à présent). Les résultats de ses travaux ont été développés dans deux propositions préliminaires :

— Schema-Structure [Beech& 1999] donne la définition d'un schéma permettant d'associer aux éléments des DTD des structures plus complexes dont le contenu et les attributs sont plus typés;
— Schema-Data [Biron& 1999] permettant de définir ses propres types de données à inclure dans les schémas.

On peut espérer plusieurs avantages 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 serré 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 archétypes (`NamedElement` et `NamedSizedElement`) l'un étant dérivé de l'autre et trois types d'entités (`GENE`, `PROTEIN` et `INTERACTION`).

```
<!DOCTYPE schema PUBLIC '-//W#C//DTD XML Schema Version 1.0//EN'
       SYSTEM "http://www.w3.org/1999/05/06-xmlschema-1/structures.dtd">

<schema name="protein.xsd">

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

```
<datatype name="proteinName">
	<basetype name="name">
	<lexicalRepresentatiom>
		<lexical>"[A-Z]{1-10}"</lexical>
	</lexicalRepresentation>
</datatype>

<archetype name="NamedElement" model="refinable">
	<attrDecl name="name" required="true">
		<datatypeRef name="name"/>
	</attrDecl>
</archetype>

<archetype name="NamedSizedElement" model="refinable">
	<refines><archetypeRef name="NamedElement"></refines>
	<attrDecl name="length">
		<datatypeRef name="integer">
			<default>0</default>
		</datatypeRef>
	</attrDecl>
</archetype>

<elementType name="GENE">
	<archetypeRef name="NamedElement">
</elementType>

<elementType name="PROTEIN">
	<archetypeRef name="NamedSizedElement">
	<attrDecl name="name"><datatypeRef name="ProteinName"></attrDecl>
	<choice>
		<empty/>
		<sequence>
			<elementTypeRef name="GENE"/>
			<all minOccur=0 maxOccur="*">
				<elementtypeRef name="INTERACTION"/>
			</all>
		</sequence>
	</choice>
</elementType>

<elementType name="INTERACTION">
	<sequence>
		<elementTypeRef name="PROTEIN"/>
		<elementTypeRef name="GENE"/>
	</sequence>
</elementType>

</schema>
```

Le document correspondant à l'exemple initial décrit dans le Schéma précédent est identique à celui de l'exemple initial (à ceci près qu'il faudra préciser qu'il est régit par ce schéma). À l'instar de la transformation TROEPS-XML, il est tout à fait possible d'exporter vers XML-Schema. La différence essentielle entre les deux est qu'un document Schéma-valide à la différence d'un document simplement valide n'est plus seulement bien structuré mais qu'il est 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 d'`ELEMENT`) ;
— les collecteurs (ensemble, liste) sont toujours absent,
— 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 [Capponi 1995] qui ne dispose pas de types primitifs. Pour cela il faudrait pouvoir définir un prédicat d'appartenance ou d'ordre par exemple. Ceci suppose de pouvoir donner une définition exécutable des types de données. Les concepteurs d'XML 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, CKML: la RCO n'est pas loin

Pour beaucoup, 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 champs du W3C.

SHOE ("Simple HTML Ontology Extensions" [Heflin& 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" [Kent 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. Son expressivité est donc très importante.

CKML ("Conceptual Knowledge Markup Language" [Kent 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 application (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 décrite plus haut ou l'effort de la communauté des graphes conceptuels pour définir un "Conceptual Graph Markup Language". À noter qu'à l'instar de la solution 2 retenue ci-dessus ces extensions d'XML sont des DTD et le schéma conceptuel (ou les ontologies) sont définies dans les documents XML eux-même.

3.4 Seconde synthèse

Il existe donc différentes initiatives à l'extérieur du W3C 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.

Si l'on considère l'évolution naturelle de ces efforts, ils vont dans le sens de permettre d'exprimer dans les DTD le plus possible d'information et par conséquent de déléguer à l'analyseur XML (ou plutôt à la partie générique du client) de plus en plus de tâches. Il en résulte que petit à petit les tâches d'un système de représentation de connaissance y seront intégrées. On peut décrire cette évolution ainsi :

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

Tableau 1 : Répartitions des tâches entre XML et représentation de connaissance dans les trois types d'extensions.

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 comprendre 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 objet. On ne peut pas en dire autant de la sémantique. C'est principalement pourquoi XML n'est pas un langage de représentation de connaissance.

4 TROEPS et XML

TROEPS est doté d'une interface XML, en entrée comme en sortie, fondée sur la seconde solution du §2. La DTD correspondante est bien plus complète que celle présentée ci-dessus. Elle peut être obtenue (ainsi que quelques exemples) en :

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

On présente ci-dessous brièvement quelques caractéristiques qui n'ont pas été discutées ci-dessus mais peuvent être utiles pour l'application.

4.1 Fonctionnalités

Les fonctionnalités demandées au schéma proposé peuvent être multiples :

— Fournir un format de lecture/écriture pour TROEPS/XTERM lisibles par d'autres instruments (browsers…) ; ceci n'est pas l'objectif principal et demandera l'utilisation d'HTML dans l'état actuel des technologies (pas de browser XML/XSL) ;
— Fournir un format d'échange entre les logiciels TROEPS et XTERM dans le cadre du projet ;
— Fournir un format permettant d'insérer du TROEPS dans d'autres documents (XML ou HTML) afin de les annoter.

Dans ce dernier cas on peut penser à utiliser les expressions TROEPS-XML au sein de descripteurs RDF. Cela est assez naturel et la spécification RDF indique comment insérer ces annotations au sein de pages HTML. On voit ici qu'XML est réellement un formalisme unificateur puisqu'il permet de d'exprimer les trois formats nécessaires : HTML, RDF et TROEPS-XML.

4.2 Opérations sur les objets

L'un des intérêts d'XML est de pouvoir importer des DTD (dans les DTD). On peut ainsi construire des extensions de la DTD TROEPS et, en particulier, une DTD manipulant les objets (comme elle implémente une partie de l'API de TROEPS, elle se nomme `trapi.dtd`).
Ainsi, le contenu du document XML ne sera plus une description de base de connaissance mais une séquence d'actions à appliquer à une base déjà existante. Cette particularité est utile dans le cadre de l'expérimentation GÉNIE car XTERM va dynamiquement invoquer TROEPS pour créer des taxonomies de classes et les décrire.
L'exemple suivant se borne à détruire une base existante :

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE trp:PGM SYSTEM "http//co4.inrialpes.fr/xml/trapi.dtd">
```

```
<trp:PGM name="biologykb">
	<trp:DEL>
		<trp:TRPKBREF name="biologykb"/>
	</trp:DEL>
</trp:PGM>
```

Un autre exemple est donné dans la section suivante.

4.3 Le lexique en XML

Comme TROEPS dispose lui-aussi d'un lexique, il est intéressant de considérer l'encodage de ce lexique et de ses éléments en XML. Ci-dessous on donne un exemple d'élément de lexique. Le terme « protéine » est défini comme une séquence d'acides aminés, hyponyme d'entité biologique et hyperonyme d'enzyme et d'hormone. Il indexe par ailleurs le concept `proteine` définit plus haut.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE LEXICON SYSTEM "http//co4.inrialpes.fr/xml/trlex.dtd">

<LEXICON>
	<LEXITEM term="proteine">
		<DEFINITION>Une sequence d'acides amines</DEFINITION>
		<HYPERONYM><LEXREF term="entite-biologique"/></HYPERONYM>
		<HYPONYMS>
			<LEXREF term="enzyme"/>
			<LEXREF term="hormone"/>
		</HYPONYMS>
		<TROEPSOBJ>
			<CONCEPTREF name="proteine"/>
		</TROEPSOBJ>
	</LEXITEM>
</LEXICON>
```

La DTD correspondante illustre l'import d'une DTD par une autre :

```
<?xml  version="1.0" encoding="UTF-8" ?>
<!-- trlex.dtd version="0.1" last-modification="18/04/1999"
               url="http://co4.inrialpes.fr/xml/trlex.dtd" -->

<!-- import troeps DTD -->
<!ENTITY % troepsDTD SYSTEM "troeps.dtd">
%troepsDTD;

<!-- Declare reference content -->
<!ENTITY % LEXREFT "term CDATA #REQUIRED">

<!ELEMENT LEXICON (LEXITEM*)>
<!ELEMENT LEXREF EMPTY>
<!ATTLIST LEXREF %LEXREFT;>
<!ELEMENT LEXITEM (DEFINITION?,SYNONYMS?,ANTONYMS?,HYPERONYM?,HYPONYMS?,TROEPSOBJ?)>
<!ATTLIST LEXITEM %LEXREFT;>
<!ELEMENT DEFINITION (#PCDATA)>
<!ELEMENT SYNONYMS (LEXREF*)>
<!ELEMENT ANTONYMS (LEXREF*)>
<!ELEMENT HYPERONYM (LEXREF?)>
<!ELEMENT HYPONYMS (LEXREF*)>
<!ELEMENT TROEPSOBJ (CONCEPTREF|VIEWREF|CLASSREF|KFIELDREF|CFIELDREF|BRIDGEREF|OBJREF)*>
```

4.4 Manipulation de base sous XML

La facilité de développement sous XSLT permet d'envisager aisément la reconstruction de l'interface graphique de TROEPS autour de transformateurs de la DTD TROEPS permettant d'engendrer du HTML (et sans doute de transformer le serveur TROEPS en servlet manipulant du XML à la demande) et vers FO (par exemple pour imprimer un document à partir d'une base de connaissance). La Figure 1 montre le fonctionnement du dispositif :

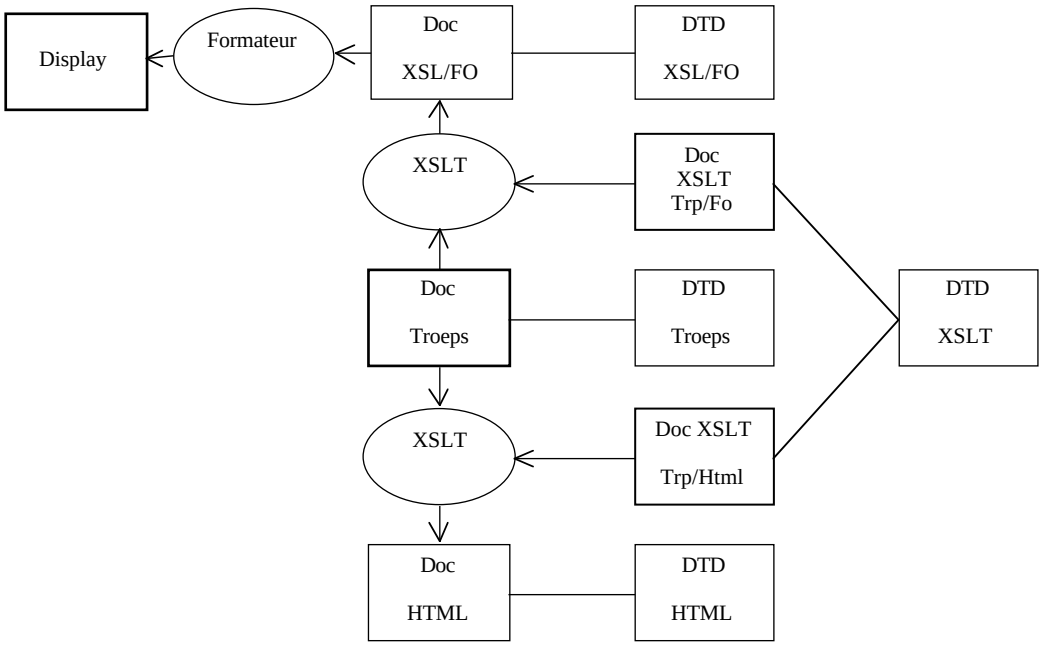

Figure 1 : Deux processus (XSLT et Formateur) permettent de formater un document. Toutes les données sont des fichiers XML (dont certain sont des DTD). Il suffit de décrire les règles de transformation de XML/TROEPS vers le langage cible (par exemple, HTML) pour obtenir le forma désiré.

5 Interface TROEPS-Lexique-XML

On présente ci-dessous le fonctionnement de l'interface entre TROEPS et XTERM via XML. Pour cela on présente tout d'abord le format d'échange entre les deux applications (en fait de XTERM à TROEPS) puis le schéma de communication dans lequel s'insère ce format.

5.1 Format d'échange

L'utilisateur d'XTERM a décidé de créer plusieurs classes et de les relier aux termes correspondants dans XTERM. On utilisera pour cela la DTD `trapi.dtd` mais il est possible détendre encore cette DTD pour les besoins de l'application si cela se fait sentir.
Ci-dessous est présenté en XML deux actions commandées par XTERM. La première (`ADD`) consiste à créer une classe à partir des éléments du lexique. La seconde `ANNOTATE` permet d'ajouter à cette classe sa définition supposée (qui apparaîtra à titre de commentaire dans l'interface de TROEPS).

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE trp:PGM SYSTEM "http//co4.inrialpes.fr/xml/trapi.dtd">

<trp:PGM name="Geniekb">
       <trp:ADD>     <!-- object created by the lexicon -->
              <trp:CLASS>
                     <trp:VIEWREF name="DocGenerated">
```

```
                        <trp:CONCEPTREF name="Plan"/>
                    </trp:VIEWREF>
                    <trp:CLASSDSC name="ModifiedPlan">
                        <trp:CLASSREF name="Plan">
                            <trp:VIEWREF name="DocGenerated">
                                <trp:CONCEPTREF name="Plan"/>

                            </trp:VIEWREF>
                        </trp:CLASSREF>
                        <trp:CFIELDDSC name="NewMission"/>
                    </trp:CLASSDSC>
                </trp:CLASS>
        </trp:ADD>
        <trp:ANNOTATE label="xterm"><!-- attach link to term representation -->
                <trp:CLASSREF name="ModifiedPlan">
                    <trp:VIEWREF name="DocGenerated">
                        <trp:CONCEPTREF name="Plan"/>
                    </trp:VIEWREF>
                </trp:CLASSREF>
                <trp:CONTENT>
A modified plan is a plan that has been changed either in its schedule or its instruments. It
retains the same goal as its initial plan.
                </trp:CONTENT>
        </trp:ANNOTATE>
</trp:PMG>
```

Voici la DTD `trapi.dtd` qui permet d'envisager de communiquer d'autres informations (création d'attributs, voire destruction des objets créés).

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- trapi.dtd version="0.2" last-modification="23/08/1999"
                url="http://co4.inrialpes.fr/xml/trapi.dtd" -->

<!-- import troeps DTD -->
<!ENTITY % troepsDTD SYSTEM "http://co4.inrialpes.fr/xml/troeps.dtd">
%troepsDTD;

<!ENTITY % basic "xml:lang NMTOKEN #IMPLIED
                  xmlns:trp CDATA #FIXED 'http://co4.inrialpes.fr/xml/troeps.dtd'">
<!ENTITY % val "trp:VAL|trp:OBJREF|trp:SET|trp:LIST">
<!ENTITY % aval "trp:VAL|trp:OBJREF">

<!ENTITY                                        %                                       trentity
"(trp:TRPKB|trp:CONCEPT|trp:KFIELD|trp:VIEW|trp:CLASS|trp:FILTER|trp:CFIELD|trp:BRIDGE|trp:OBJ
ECT)">
<!ENTITY                                         %                                        trref
"(trp:TRPKBREF|trp:CONCEPTREF|trp:KFIELDREF|trp:VIEWREF|trp:CLASSREF|trp:CFIELDREF|trp:BRIDGER
EF|trp:OBJREF)">

<!ENTITY                                        %                                       traction
"trp:ADD|trp:DEL|trp:ANNOTATE|trp:RENAME|trp:ATTACH|trp:SETVAL|trp:ADDVAL|trp:REMVAL|trp:UNSET
VAL">

<!ELEMENT trp:PGM ((%traction;)*)>
<!ATTLIST trp:PGM base NMTOKEN #REQUIRED
                  %basic;>

<!ELEMENT trp:ADD %trentity;>
```

```
<!ELEMENT trp:DEL %trref;>

<!ELEMENT trp:ANNOTATE (%trref;,trp:CONTENT)>
<!ATTLIST trp:ANNOTATE label NMTOKEN #REQUIRED>

<!ELEMENT trp:RENAME %trref;>
<!ATTLIST trp:RENAME name NMTOKEN #REQUIRED>

<!ELEMENT trp:ATTACH (trp:OBJREF,trp:CLASSREF)>

<!ELEMENT trp:SETVAL (trp:OBJREF,(%val;))>
<!ATTLIST trp:SETVAL fieldname NMTOKEN #REQUIRED>

<!ELEMENT trp:ADDVAL (trp:OBJREF,(%aval;))>
<!ATTLIST trp:ADDVAL fieldname NMTOKEN #REQUIRED>

<!ELEMENT trp:REMVAL (trp:OBJREF,(%aval;))>
<!ATTLIST trp:REMVAL fieldname NMTOKEN #REQUIRED>

<!ELEMENT trp:UNSETVAL (trp:OBJREF)>
<!ATTLIST trp:UNSETVAL fieldname NMTOKEN #REQUIRED>

<!ELEMENT trp:CONTENT ANY>
```

5.2 Utilisation

Dans le cadre de GÉNIE II 3.3.1, l'objectif premier est d'échanger les données entre XTERM et TROEPS, c'est-à-dire de permettre à XTERM d'engendrer des descriptions d'objets TROEPS au format XML et de les communiquer au serveur TROEPS.

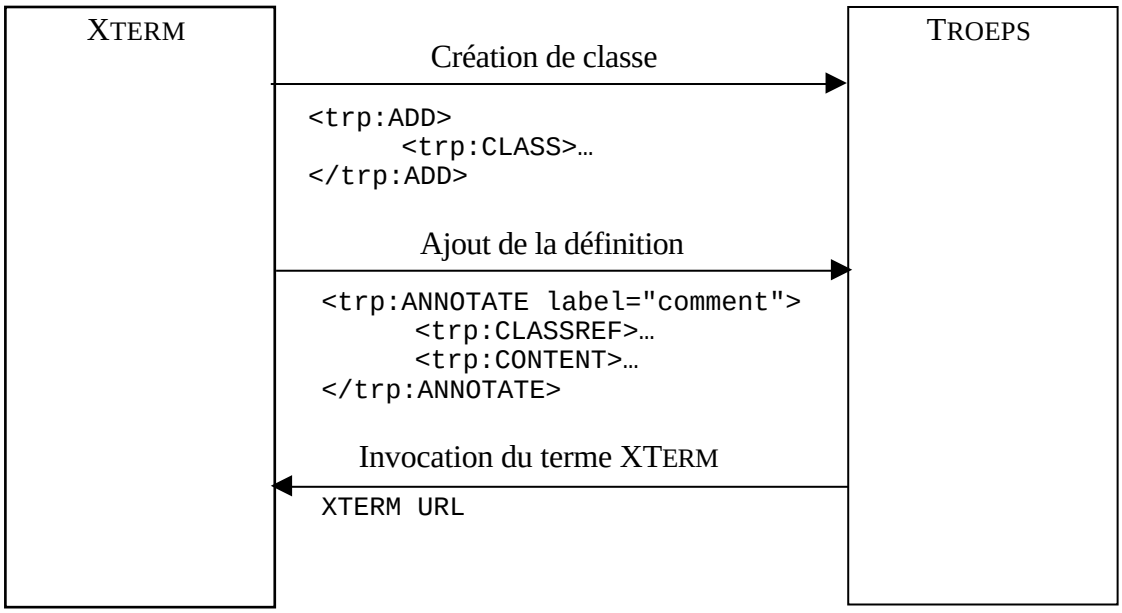

Figure 2 : Interaction entre XTERM et TROEPS via XML.

La réalisation pratique de cette fonctionnalité est décrite dans [Euzenat& 1999].
Ici TROEPS ne conserve aucune informations concernant la création des classes. Ainsi, il est possible d'invoquer XTERM sur un objet qu'il n'a pas créé. De cette manière, et comme décrit dans [Euzenat& 1999], on peut considérer toute base TROEPS comme implicitement indexée par XTERM.

Ceci est réalisé simplement en assignant à la variable `*xterm-prefix*` le début de l'URL d'XTERM. Par exemple :

```
? (setf (global *xterm-prefix*) "http://da.com/xterm/")
```

Lorsque l'utilisateur clique sur le bouton "Lexicon" de l'interface de TROEPS, XTERM est invoqué sur le terme correspondant (`http://da.com/xterm/ModifiedPlan`).

6 Implémentation

On présente ci-dessous les principes de l'analyse de documents XML et les logiciels disponibles (§6.1). L'insertion de ces principes dans le cadre de TROEPS est détaillée (§6.2) avant d'analyser les performances obtenues (§6.3) et les problèmes subsistant dans l'implémentation actuelle (§6.5).

6.1 Analyse syntaxique

Il existe des analyseurs validant et non validant suivant qu'ils prennent en compte ou non les DTD ou DOCTYPE. À partir de l'instant où nous voulons utiliser une DTD, il est bien entendu intéressant d'utiliser un analyseur validant. Ceci dit, les deux types d'analyseurs doivent rendre le même résultat sur des documents valides. Ainsi, si nous sommes assurés de disposer de documents valides, ils peuvent être analysés par n'importe quel type d'analyseur.

Il existe principalement deux modes de fonctionnement des analyseurs :

— Un fonctionnement par événement où chaque « règle syntaxique » reconnue déclenche l'appel d'un "call-back" permettant d'effectuer les actions appropriées. Ces analyseurs utilisent en général une interface de programmation (API) nommée SAX (Simple API for XML) qui appelle les "call-backs" à l'ouverture et à la fermeture des tags (correspondant aux éléments).

— Un fonctionnement par arbre d'analyse où l'analyseur retourne un arbre correspondant au document que l'application pourra utiliser. Ces analyseurs engendrent en général un arbre dans le format DOM (pour "document object model") qui pourra être manipulé par les primitives DOM.

Ces deux types d'interfaces ne sont pas incompatibles. Par exemple, l'utilitaire UNIX YACC autorise les deux types de manipulation et elles peuvent parfaitement être imbriquées. C'est ainsi que l'analyseur de TROEPS peut être considéré comme un analyseur réactif (il ne construit que localement quelques structures de données très vite libérées). Pour une application telle que TROEPS amenée à manipuler des données d'une taille conséquente il est en général irréaliste de vouloir construire un arbre contenant toutes les données et un analyseur réactif est la seule solution.

Les analyseurs validant disponibles sur le marché qui peuvent être réutilisés sont les suivants (seuls les analyseurs en JAVA et C disponibles sous UNIX sont retenus pour des raisons de portabilité) :

Nom	Langage	Origine	Diffusion	DOM	SAX	Valid	Sch.
Xmllink	JAVA	W3C ?					
RXP[2]	C	Edinburgh U.	Gnu GPL			•	
XP[3]	JAVA 1.1	James Clark			•		
Lark/Larval[4]	JAVA	Textuality			•	•	
XML4J[5]	JAVA 1.1	IBM Tokyo	registr.	•	•	•	•

[2] http://www.cogsci.ed.ac.uk/~richard/rxp.html
[3] http://www.jclark.com/xml/xp/
[4] http://www.textuality.com/Lark/
[5] http://www.alphaworks.ibm.com/formula/xml

SXP[6]	JAVA 1.1	INRIA (Sylphide)		•	•	•	
JAVA project X	JAVA 1.2	Sun	registr.	•	•	•	
Libxml[7]	C	Gnome (Veillard)	free	/			

Tableau 2 : Liste et caractéristiques des analyseurs XML.

6.2 Réalisation

La génération d'XML à partir des objets TROEPS est une simple affaire de reprogrammation de l'interface de sauvegarde (ou de génération de pages HTML).
L'analyse des objets XML pour les transformer en structure TROEPS ne pose pas de problème conceptuels. En fait, le langage de TROEPS est très proche d'XML (légèrement plus riche car il permet de distinguer simplement listes et ensembles). L'analyseur est un programme YACC qui ne reconnaît que la structure. Il fonctionne par événements et seuls quatre "call-backs" sont implémentés. Dans la figure 2, ils sont comparés aux "call-backs" de SAX.

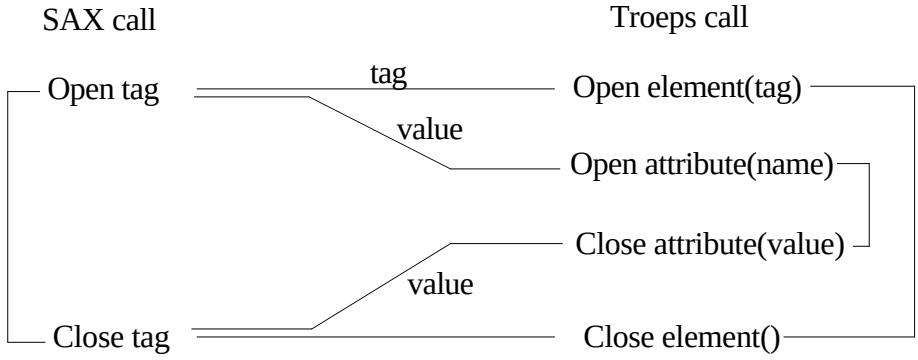

Figure 3: Primitives SAX et Primitives TROEPS.

Le choix de l'analyseur à utiliser dépend du type de connexion que l'on souhaite :

— Une connexion forte permet d'intégrer l'analyse à TROEPS et de réutiliser les appels de TROEPS. Il est raisonnable de la faire en C, l'édition de lien entre JAVA et TALK étant peut-être délicate.

— Une connexion faible devra communiquer avec TROEPS à l'aide d'une "socket" ou de fichier. Elle peut indifféremment s'effectuer en C ou en JAVA. Elle permet de ne pas modifier notablement le serveur TROEPS ni de l'alourdir.

Aucun analyseur SAX validant en C n'étant librement disponible, JAVA est le langage d'implémentation ce qui par rapport au type de connexion entraîne une connexion faible. On suivra cependant la mise à niveau d'autres analyseurs écrits en C.

6.3 Utilisation

Pour utiliser le dispositif mis en place, il est tout d'abord nécessaire d'avoir un serveur HTTP lancé (par exemple, à l'URL `http://da.com`) avec un catalogue de scripts CGI contenant le script `hytropes`. Il faut ensuite lancer le serveur TROEPS à l'aide de la commande :

```
$ $TRPDIT/$PORTNAME/bin/trpserver –b http://da.com/cgi-bin/hytropes \
                                  -n machine.da.com –p 5555 \
                                  -j http://co4.inrialpes.fr/java \
                                  -d http://co4.inrialpes.fr/help \
                                  htrp-config.t
```

[6] http://www.loria.fr/projets/Xsilfide/EN/sxp/
[7] http://rufus.w3.org/veillard/XML/xml.html

Où `machine.da.com` est l'adresse IP de la machine courante, `5555` est le port sur lequel le serveur sera connecté et `htrp-config.t` est un fichier TALK dans lequel le serveur est configuré (par exemple en chargeant une base ou en assignant la variable `*xterm-prefix*`).
Il faut ensuite lancer le serveur d'analyse XML (avec une machine virtuelle JAVA 1.2) par la commande :

```
$ java –cp .:xml4j.jar:xxt.jar xxt.xxtServer
```

Le serveur prend sa configuration (et en particulier l'adresse IP et le port du serveur TROEPS) dans un fichier ayant pour nom `./config`. Le résultat peut être testé sur le fichier à l'aide de la commande :

```
$ java –cp .:xml4j.jar:xxt.jar xxt.xxtAppli mon-fichier.xml
```

Si on désire utiliser le système non pas en serveur mais comme une simple application, il est possible de lancer :

```
$ java –cp .:xml4j.jar:xxt.jar xxt.xxtServer mon-fichier.xml
```

6.4 Performances

Comme il est intéressant de maîtriser cette implémentation, on compare ci-dessous les performances de TROEPS/XML avec celles de TROEPS. Cela donne une idée de l'intérêt du passage par XML et de l'implémentation réalisée en termes de performances. On compare celle-ci sur deux bases : l'une très simple real-estate, l'autre, knife beaucoup plus complexe intégrant toutes sortes de circularités (500 classes, 500 objets).
Une première mesure concerne la taille des fichiers engendrés par TROEPS pour décrire une base. Il existe pour cela trois formats : un format spécifique à TROEPS, un programme en langage TALK (LISP) qui permet de reconstruire la base et XML. Par mesure d'équité on a supprimé les indentations de tous les fichiers. Les chiffres donnés distinguent le nombre de lignes et le nombre de caractères puis le nombre de caractères une fois le fichier compressé (par zip) pour réduire les effets dus à la redondance.

	Real-estate	Knife
TROEPS	656 — 16137 — 2086	13495 — 283378 — 25651
TALK	654 — 32513 —2726	10404 — 661664 — 31762
XML	1748 — 53086 — 2634	36348 — 1199299 — 37729

Tableau 3 : Tailles comparées des fichiers (en lignes, caractères, caractères une fois compressés).

Les fichiers XML sont donc plus important à stocker (et sans doute à analyser) mais si l'on considère le temps de transmission avec compression, la différence n'est plus très importante.
Les résultats temporels constituent les autres mesures. Ils sont données en temps réel et temps CPU (temps CPU utilisateur + temps CPU système) avec la commande time du Bourne-shell (moyenne sur 10 essais le meilleur et moins bon temps exclus). Ils sont données sur une SparcStation 4 dotée de 96Mo de mémoire centrale.
La ligne Trpserver correspond au chargement des objets sous TROEPS (lancement de TROEPS, analyse des arguments inclus) et prise en compte des circularités. La ligne XML4J correspond au simple comptage de l'analyseur XML (classe saxCounter de démonstration donnée avec le système). Elle ne prend donc pas en compte la création des objets sous Troeps. La ligne XXT correspond à l'analyse du fichier XML et au chargement des objets sous TROEPS via l'architecture présentée ci-dessus, c'est-à-dire communication par "socket". Le temps réel est alors fortement pénalisée par l'architecture puisqu'il comprend l'encodage des actions et le temps mis par TROEPS pour les décoder et les créer. Mais le temps CPU ne prend en compte que l'encodage (il fait donc moins de travail que TROEPS).

	Real-estate	Knife
Trpserver	2.2 — 2.1	8:34 — 9.24 (!)
XML4J	6.8 — 6.5	17.7 — 15.2
XXT	1:03.3 — 14.2	1:00:31.9 — 1:43.

Tableau 4 : Temps comparés d'analyse d'une base (temps réel et CPU).

Si l'on se fonde uniquement sur les temps CPU, l'analyseur JAVA est bien plus lent (2 fois) que TROEPS qui crée le objets et gère les circularités. Lorsque l'analyseur doit, en plus, réaliser la reconnaissance des éléments (ligne XXT), les temps nécessaire sont nettement trop longs.
La conclusion de ces études est qu'il n'est pas pour l'instant viable de réaliser l'analyse en JAVA[8] (par rapport à TROEPS écrit en LISP qui gère les objets). Les deux analyseurs sont pourtant très comparable : ils sont dirigés par les évènements (de même type) et la partie écrite en YACC réalise à peu près le même travail que l'analyseur XML. Il faudrait confirmer ceci avec l'analyseur XP de James Clark dont l'ambition est d'être optimisé.

6.5 Problèmes connus

L'analyseur implémenté est actuellement partiel pour des raisons de démonstrations dans le cadre de GÉNIE. Les problèmes non traités pour l'instant sont les suivants :
— Pas de gestion des circularités.
— Pas de prise en compte des annotations.
— L'API TROEPS [Sherpa 1995], n'est pas complètement couverte.

7 Conclusion

L'intérêt principale de l'expérimentation est de montrer la faisabilité de l'approche et non de produire un système exploitable. Elle est aussi de produire un système tenant compte des potentialités actuelles.
On peut tirer deux conclusions de cette étude :
— le choix du langage XML pour communiquer entre XTERM et le système d'objets (TROEPS ou autre) est judicieux dans la mesure ou il peut s'appuyer sur des standards existants eux-mêmes implémentés dans des outils largement disponibles, ouverts et maintenus,
— le choix de l'architecture de communication entre les différents modules, si elle a permis de maquetter confortablement, n'est pas à retenir pour une application opérationnelle (à l'heure actuelle).

L'intérêt de la connexion entre TROEPS et XTERM pour assurer la traçabilité est l'objet de l'ensemble de la tâche 3.3.1.
Cette phase de maquetage aura aussi permis de s'initier à XML sur un système disponible. Le transfert de l'effort vers des outils industriels tels que Rational Rose devrait être possible dés lors que ceux-ci supporteront le format "XML Metadata Interchange" qui sera vraisemblablement retenu par l' OMG en tant que "Stream-based model Interchange Format". Alternativement, IBM a sorti en juillet 1999 son XMI Toolkit[9] permettant de traduire les modèles UML de Rose en XMI et vice-versa.

8 Documentation

David Beech, Scott Lawrence, Murray Maloney, Noah Mendelsohn, Henry Thompson (éds.)
XML Schema part 1: structures
Working draft, W3C, mai 1999 `http://www.w3.org/TR/xmlschema-1/`

[8] Deux remarques : — XXT va être légèrement optimisé de sorte que les chiffres vont baisser — Il faut que j'ajoute un test avec XP et sans doute un autre avec Expat ou libxml (deux analyseurs en C).
[9] http://www.alphaworks.ibm.com/tech/xmitoolkit

Paul Biron, Ashok Malhotra (éds.)
XML Schema part 2: datatypes
Working draft, W3C, mai 1999 http://www.w3.org/TR/xmlschema-2/

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen (éds.)
Extensible Markup Language (XML) 1.0
Recommendation, W3C, février 1998 http://www.w3.org/TR/REC-xml

Tim Bray, Charles Frankston, Ashok Malhotra (éds.)
Document Content Description for XML
Submission, W3C, juillet 1998 http://www.w3.org/TR/NOTE-dcd

Tim Bray, Dave Hollander, Andrew Layman (éds.)
Namespaces in XML
Recommandation, W3C, janvier 1999 http://www.w3.org/TR/REC-xml-names

Dan Brickley, R. Guha (éds.)
Resource description framework schema specification
Proposed recommandation, W3C, mars 1999 http://www.w3.org/TR/PR-rdf-schema

Cécile Capponi
Identification et exploitation des types dans un modèle de représentation des connaissances par objets
Thèse d'informatique, université Joseph-Fourier, Grenoble (FR), 1995

James Clark (éd.)
XSL transformations (XSLT) specification version 1.0
Working draft, W3C, août 1999 http://www.w3.org/TR/WD-xslt

Stephen Deach (éd.)
Extensible Stylesheet language (XSL) specification
Working draft, W3C, avril 1999 http://www.w3.org/TR/WD-xsl

Michael Erdman, Rudi Studer
Ontologies as conceptual models for XML documents
Actes 12th KAW, Banff (CA), 1999

Jérôme Euzenat, Farid Cerbah
Spécification d'un système d'aide à la pose de liens de traçabilité
Rapport interne, Génie II 3.3.1, mars 1999

Lars Garshol
Introduction to XML
http://www.stud.ifi.uio.no/~larsga/download/xml/xml-eng.html

Jeff Heflin, James Hendler, Sean Luke, Qin Zhendong
SHOE: a knowledge representation language for internet applications
submitted to Artificial intelligence, 1998 http://www.cs.umd.edu/projects/plus/SHOE/aij-shoe.ps

Robert Kent
Conceptual Knowledge Markup Language
Actes 12th KAW, Banff (CA), 1999

Ora Lassila, Ralph Swick (éds.)
Resource Description Framework (RDF) Model and syntax specification
Recommendation, W3C, février 1999 http://www.w3.org/TR/REC-rdf-syntax

Alain Michard
XML: langage et applications
Eyrolles, Paris (FR), 1998

Ashok Malhotra, Murray Maloney (éds.)
XML Schema requirements
Note, W3C, février 1999 http://www.w3.org/TR/NOTE-xml-schema-req

Olga Mariño, François Rechenmann, Patrice Uvietta
Multiple perspectives and classification mechanism in object-oriented representation
Actes 9th ECAI, Stockholm (SE), pp425-430, 1990

Projet SHERPA
TROEPS 1.2 reference manual
Rapport interne, INRIA Rhône-Alpes, Grenoble (FR), juillet 1998, 148p.
http://co4.inrialpes.fr/docs/tropes-manual.html
ftp://ftp.inrialpes.fr/pub/sherpa/rapports/tropes-manual.ps.gz

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