# XML et les objets (Objectif XML)

**Jérôme Euzenat, Amedeo Napoli, Jean-François Baget**

*INRIA Rhône-Alpes & LORIA*
*655 avenue de l'Europe, 38330 Montbonnot, France*
*Jerome.Euzenat@inrialpes.fr,*
*Amedeo.Napoli@loria.fr,*
*Jean-Francois.Baget@inrialpes.fr*

RÉSUMÉ. *Le langage XML et les objets ont en commun la perspective de partage et de réutilisation de leur contenu grace à une plus grande structuration de celui-ci. On présente la galaxie XML□la base de XML (XML, espaces de noms, DTD et représentations internes), une structuration plus proche des modèles à objets (XMI, XML-Schema et Xquery) et des outils de modélisation apparentés aux représentations de connaissances (RDF, RDF-Schema, cartes topiques et OWL). Chaque langage présenté est mis en relation avec les efforts analogues au sein des objets.*

ABSTRACT. *The XML language and the object approach both aim at reusing and sharing their content through its improved structure. The XML galaxy is presented beginning with the XML basis (XML, namespaces, DTD and internal encoding), continuing with object-inspired structures (XMI, XML-Schema and Xquery) and modelling tools close to knowledge representation (RDF, RDF-Schema, Topic maps and OWL). Each language is presented in relation with corresponding efforts in the object world.*

MOTS-CLÉS : *XML, Objets.*

KEY WORDS: *XML, Objects.*

## 1. Introduction

Le Web, outre qu'il a accéléré l'informatisation d'une proportion importante de la société, a radicalement modifié la manière d'envisager l'informatique. Les contraintes de fonctionnement du Web privilégient la modularité et l'autonomie des éléments manipulés. De ce point de vue, les technologies objets sont souvent utilisées pour répondre à ces contraintes. Qu'ils se trouvent mobilisés pour encoder les documents, représenter leur contenu, ou implémenter des serveurs, qu'ils soient mobiles ou réduits à l'état de balises, les objets sont naturellement présents au sein du Web.
Par ailleurs, le souci de plus en plus prégnant de contrôler les informations, documents ou données, sur la base d'une sémantique qui leur serait associée, renvoie aux problématiques de la représentation des connaissances dans lesquelles les approches objets sont très présentes. C'est pourquoi, alors que se déploient d'importants projets sur le Web sémantique [Berners-Lee&2001], les objets pourraient se faire encore plus indispensables, à la fois en tant qu'éléments de représentation des connaissances, en tant que support de programmation et d'échange, et en tant qu'unité de déploiement modulaire de services.

Il est important et significatif d'analyser ce que trente ans de maturation de la technologie des objets, qui a investi toute l'informatique, apporte à la jeune technologie du Web qui a investi les façons de vivre et d'envisager les échanges. Le langage XML partage avec les objets l'objectif de réutilisation. Il s'agit effectivement d'un effort pour permettre de réutiliser l'information disponible. Mais là où les objets promettent la réutilisation par l'encapsulation, XML la permet par l'ouverture.
Cette introduction à « XML et objets » est conçue à la fois comme une introduction à XML et à sa relation avec les formalismes objets tels que nous les connaissons. Elle permettra de mettre en perspective les problèmes considérés dans les articles suivants.
Nous commençons par présenter le modèle de données de XML (§2), c'est-à-dire la structure des documents XML et la manière de la spécifier qui a peu à voir avec les propriétés des objets. Nous évoquons ensuite les efforts permettant de rapprocher ces spécifications des formalismes des langages à objets (§3). C'est le cas de XML-Schema pour spécifier les documents à l'aide de spécialisation et de restrictions de contraintes, XMI pour transporter les modèles UML et Xquery pour manipuler la structure abstraite d'XML. Enfin, nous présenterons les travaux qui, autour du web sémantique, conduisent à construire, au dessus d'XML, un langage de représentation de connaissance par objets (§4).

## 2. Le modèle de données XML

Le premier niveau de prise en compte d'XML est celui d'un format de données très général et très simple pouvant être utilisé dans de nombreux types d'applications.

On présente ci-dessous XML lui-même, des moyens élémentaires de contraindre la structure des documents XML□ DTD et RELAX-NG, et deux représentation manipulables en machine: DOM et SAX.

XML

XML [Bray&□1998a] est un langage de□alisage 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 [Bosak&□1999] et une description plus complète en français se trouve dans [Michard□1998]. XML est délibérément intermédiaire entre HTML et SGML. Il est plus simple et moins contraignant que SGML□; 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□

$$\texttt{<TAG att}_1\texttt{="v}_1\texttt{"… att}_n\texttt{="v}_n\texttt{"/>}$$

ou

$$\texttt{<TAG att}_1\texttt{="v}_1\texttt{"… att}_n\texttt{="v}_n\texttt{">contenu</TAG>}$$

où `contenu` est une suite d'expressions XML ou une chaîne de caractères, $\texttt{att}_i$ est un identificateur d'attribut et $\texttt{v}_i$ la valeur de cet attribut. Le `TAG` est appelé élément. Un document XML peut ainsi être vu comme un arbre dont la racine est le document et les fils d'un nœud sont les éléments de son contenu. Contrairement à la notion d'objet, elle aussi très générale, et relativement simple, XML peut parfaitement se passer de définition préalable (ceci la rapproche des langages de «□rames□» [Masini&□1989]).

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□». Cet exemple utilise des espaces de noms (`go` pour Gene Ontology and `fb` pour FlyBase) qui ne sont ici qu'à titre illustratifs et ne préjugent en aucun cas de l'existence ou de la conformité de ressources définies par ces organisations.
Voici l'exemple sous la forme d'un document XML (les textes XML sont donnés en police courrier et les DTD avec une barre verticale à gauche)□

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

<go:PROTEIN go:name="BICOID" go:length="422"
      xmlns:go="http://www.go.org">
   <go:GENE go:name="Bicoid"/>
   <go:INTERACTION>
```

```
      <go:PROTEIN go:name="BICOID"/>
      <go:GENE go:name="Hunchback"/>
   </go:INTERACTION>
</go:PROTEIN>
```

Ce document représente la protéine avec deux attributs — son nom, BICOID, et sa longueur, 422 — et 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 qui peut aisément être représenté dans un langage à objets.

Le modèle de XML peut être complété par la recommandation concernant les espaces de noms [Bray&□1999]. Les noms des éléments peuvent être décomposés en deux parties : le nom — PROTEIN — proprement dit et l'espace de noms auquel il est rattaché. Ce dernier est identifié par un URI ("Uniform Resource Identifier» [Berners-Lee&□1998]) — ici http://www.go.org identifié par le préfixe go. Cela permet de faire coexister des ensembles d'éléments ayant des noms en commun et surtout de développer des vocabulaires sans se soucier des interactions non désirées avec d'autres.
Plus qu'un langage, XML est une nébuleuse d'outils relativement bien organisés. Outre ceux que l'on évoquera par la suite parce qu'ils rapprochent XML des objets, on peut disposer de :

— Xpath, Xlink et Xpointers pour faire des références à des fragments de documents XML ;
— XSLT pour transformer un document XML en un autre document XML ;
— XML Normalisation, Signature et Encryption pour réaliser les opérations correspondantes de manière standard ;
— XML-RPC (ou SOAP) pour transmettre des messages encodés en XML.
— La "sérialisation" d'objets de programmation (c'est-à-dire leur encodage sous une forme indépendante de la mémoire, sans pointeurs, pour pouvoir les transporter) en XML.

Contrairement aux objets qui peuvent difficilement être exploités sans leurs classes, les documents XML doivent être appréhendables sans classes ce qui permet de leur appliquer un certain nombre de services. C'est pourquoi il est justifié de présenter le document avant sa spécification. Cependant, le besoin de définir les types d'éléments que l'on doit pouvoir trouver dans un document et leurs contenu se fait sentir. XML dispose pour cela des DTD héritées de SGML.

### DTD et RELAX-NG

Une DTD («□Définition de Type de Document») est une grammaire qui a pour but de définir chaque élément en précisant :

— son contenu comme une expression régulière utilisant 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.

L'exemple ci-dessus est défini par la DTD suivante :

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

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

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

<!ELEMENT go:INTERACTION (go:PROTEIN,go: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 obligatoires (#REQUIRED), ils pourraient être optionnels (comme `length` avec #IMPLIED).
Certains attributs peuvent être déclarés comme des identifiants (ID) permettant de faire référence à l'élément portant cet attribut sans crainte d'ambiguité.

À l'instar des DTD, RELAX-NG (lui-même fusion de deux langages : RELAX et TREX) [Clark& 2001] est un formalisme basé sur les grammaires. Il est légèrement plus puissant que les DTD car il permet de typer les attributs et les champs texte, il manipule nativement les espaces de noms et permet de déclarer des contenus dont l'ordre est indifférent (`interleave`). Mais il a surtout l'avantage d'être déclaré en XML. La DTD précédente s'exprimerait ainsi en RELAX-NG :

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

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:go="http://www.go.org">
<start>
   <ref name="protein"/>
</start>

<define name="protein">
   <element name="go:PROTEIN">
      <attribute name="go:name">
         <data type="string"/>
      </attribute>
      <optional>
         <attribute name="go:length">
            <data type="positiveInteger"/>
         </attribute>
      </optional>
      <interleave>
         <optional>
            <ref name="gene"/>
         </optional>
         <zeroOrMore>
```

```
            <ref name="interaction"/>
         </zeroOrMore>
      </interleave>
   </element>
</define>

<define name="gene">
   <element name="go:GENE">
      <attribute name="go:name">
         <data type="string"/>
      </attribute>
      <empty/>
   </element>
</define>

<define name="interaction">
   <element name="go:INTERACTION">
      <ref name="protein"/>
      <ref name="gene"/>
   </element>
</define>
</grammar>
```

La syntaxe de RELAX-NG n'est pas très éloignée de celle des DTD. On aimerait pouvoir y ajouter un peu plus des objets, à savoir, la possibilité de définir des spécialisations des éléments définis.

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 parenthésée et les éléments ouverts sont bien fermés, sans entrelacement).
- X-Valide ("valid") s'il satisfait les contraintes exprimées dans un schéma X (DTD RELAX-NG ou XML-Schema). Ci-dessous un tel document sera qualifié d'XML-valide.

L'exemple ci-dessus est DTD-valide vis-à-vis de la DTD et RELAX-valide vis-à-vis de la spécification RELAX-NG.
Il existe d'autres langages pour exprimer la structure des documents XML. On peut encore citer Schematron qui est un langage d'assertions qui doivent être satisfaites par des documents (pour être valides). Il tire parti des spécifications citées puisqu'il propose l'expression de contraintes en Xpath dans une syntaxe XML. Le schéma est transformé en XSLT et s'exécute comme une transformation qui signale les contraintes non satisfaites.

## Manipulation : DOM & SAX

Au plus bas niveau, la manipulation de documents XML se fait à l'aide de deux dispositifs : SAX et DOM.
SAX ("Simple API for XML" [Brownell 2002]) est un mécanisme de gestion d'évènements dédié à XML. Les évènements correspondent aux points d'accès de l'analyse d'un document XML : début de document, début d'élément, contenu textuel, fin d'élément, fin de document. Un gestionnaire ("handler") SAX est une interface qui reçoit un tel événement. Programmer avec SAX se résume donc à

développer de tels gestionnaires qui réagissent en fonction des évènements qui leur sont communiqués. Deux processus SAX peuvent communiquer en appelant leurs gestionnaires respectifs avec des évènements SAX. C'est ce que fait un analyseur SAX qui appelle un gestionnaire avec les évènements qu'il créé au fur et à mesure de l'analyse.

DOM ("Document Object Model", [Apparao&□1998]) propose une structure de données capable d'accueillir un document XML. Cette structure est un arbre dont les nœuds correspondent aux concepts d' XML (élément, attribut, texte…). Il offre une interface de programmation permettant de parcourir l'arbre (fils, père, attributs…) ou de le modifier (supprimer attribut, insérer élément, normaliser…). Il propose aussi un mécanisme d'évènements permettant d'être notifié des modifications réalisées par d'autres.
Ces formats sont encore très marqués par l'approche documentaire□ ils ne sont pas typés par les éléments des documents XML mais par les seuls types élément, attribut, texte… Dans ce contexte, on comprend l'intérêt d'un logiciel comme SmartTools présenté dans la suite qui est capable de manipuler les éléments de documents XML en fonction de leurs types (tirés des DTD dans l'article présenté).

XML n'est pas un format de stockage ou de manipulation, c'est principalement un format d'échange. DOM n'est qu'une interface bien pratique pour implémenter des traitements très généraux. 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). On peut donc dire qu'XML est un langage déclaratif□ il ne défini pas ses procédures d'exploitation. Même si — presque — personne ne songe à programmer en XML, la puissance d'un langage de transformation de documents comme 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.
Ni DOM, ni les DTD ne sont de véritables modèles à objets, XML a donc une position médiane entre document et objets [Al□Hulou&□2000]. On évoque ci-dessous des initiatives qui rapprochent beaucoup plus XML des langages à objets.

## 3. Objectif XML

Si XML est au départ un format d'échanges ou de balisage de documents, il a pris une place suffisamment importante pour que beaucoup cherchent à mieux spécifier ce qui doit être présent dans ce format. Malgré quelques ressemblances, XML est, en lui même, assez éloigné d'un langage à objets. 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).

C'est pourquoi plusieurs initiatives visent à donner plus de solidité à la spécification de ces formats. Quelques propositions permettent ainsi de définir une structure plus proche des objets.

C'est le cas d'XMI développé par l'Object management group qui vise à permettre l'échange de modèles et de données UML à l'aide d'XML. C'est encore le cas d'XML-Schema proposé par le W3C pour remplacer les DTD. Il permet de décrire les documents XML valides avec une finesse très proche de celle des langages de description d'objets. Par ailleurs, afin de déterminer la sémantique des requêtes, le groupe de travail Xquery a défini un modèle de données plus formel pour XML (et XML-Schema). Ce modèle permet de définir la sémantique d'un document XML et donc la réponse à une requête sur ce document.

Les deux premières propositions s'approchent de la définition de langage à objets en XML. Elles sont encore rapprochées par l'utilisation d'XML-Schema pour décrire les modèles UML et la possibilité d'intégrer des modèles correspondant aux Schémas XML présentés dans l'article d'Adbalhmed Maran et Dominique Marcadet.

## XMI

L'"Object Management Group" a défini XMI ("XML Metadata Interchange" [OMG 2002]) comme langage de sérialisation des spécifications UML. XMI est initialement conçu pour décrire et transporter des méta-données. En fait, il s'applique à n'importe quel modèle dont la structure est décrite à l'aide du MetaObject Facility (MOF [OMG 2000]) de l'Object Management Group. C'est le cas d'UML. XMI est défini par deux jeux de règles pour transformer respectivement un métamodèle décrit à l'aide du MOF en une DTD et un modèle décrit à l'aide de ce métamodèle en un document XML valide pour la DTD. Nous présentons ci-dessous deux modèles encodés en XML à l'aide du second jeu de règles : l'un est un modèle UML (dont le métamodèle est le métamodèle UML), l'autre est une instance de ce modèle. Le second ensemble de règles étant réversible, il est possible de recréer le modèle UML correspondant à partir de la sérialisation en XML.

Comme on peut s'en rendre compte, cette méthode astucieuse engendre un document XML plus verbeux que les DTD. Une part de cet accroissement est cependant due à la puissance d'UML qui permet des associations entre plus de deux objets.

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

```
<XMI version="1.1" xmlns:UML="org.omg/UML1.3">
   <XMI.header>
      <XMI.model xmi.name=""/>
      <XMI.metamodel xmi.name="UML" href="UML.xml"/>
   </XMI.header>
   <XMI.content>
      <UML:Class name="PROTEIN" xmi.id="PROTEIN">
         <UML:ModelElement.ownedElements>
            <UML:Attribute name="name" type="string"/>
            <UML:Attribute name="length" type="int"/>
         </UML:ModelElement.ownedElements>
      </UML:Class>
      <UML:Class name="GENE" xmi.id="GENE">
         <UML:ModelElement.ownedElements>
            <UML:Attribute name="name" type="string"/>
         </UML:ModelElement.ownedElements>
      </UML:Class>
      <UML:Class name="INTERACTION" xmi.id="INTER"/>
      <UML:Association>
         <UML:Association.connection>
            <UML:AssociationEnd name="protein" type="PROTEIN">
<UML:AssociationEnd.multiplicity>1..1</UML:AssociationEnd.multiplicity
>
            </UML:AssociationEnd>
            <UML:AssociationEnd name="gene" type="GENE">
<UML:AssociationEnd.multiplicity>0..1</UML:AssociationEnd.multiplicity
>
            </UML:AssociationEnd>
         </UML:Association.connection>
      </UML:Association>
      <UML:Association>
         <UML:Association.connection>
            <UML:AssociationEnd name="protein" type="PROTEIN">
<UML:AssociationEnd.multiplicity>1..1</UML:AssociationEnd.multiplicity
>
            </UML:AssociationEnd>
            <UML:AssociationEnd name="interactions" type="INTERACTION">
<UML:AssociationEnd.multiplicity>0..*</UML:AssociationEnd.multiplicity
>
            </UML:AssociationEnd>
         </UML:Association.connection>
      </UML:Association>
      <UML:Association>
         <UML:Association.connection>
            <UML:AssociationEnd name="interaction" type="INTERACTION">
<UML:AssociationEnd.multiplicity>1..1</UML:AssociationEnd.multiplicity
>
            </UML:AssociationEnd>
            <UML:AssociationEnd name="protein" type="PROTEIN">
<UML:AssociationEnd.multiplicity>1..1</UML:AssociationEnd.multiplicity
>
            </UML:AssociationEnd>
         </UML:Association.connection>
      </UML:Association>
      <UML:Association>
         <UML:Association.connection>
            <UML:AssociationEnd name="interaction" type="INTERACTION">
<UML:AssociationEnd.multiplicity>1..1</UML:AssociationEnd.multiplicity
>
            </UML:AssociationEnd>
            <UML:AssociationEnd name="gene" type="GENE">
<UML:AssociationEnd.multiplicity>1..1</UML:AssociationEnd.multiplicity
>
            </UML:AssociationEnd>
         </UML:Association.connection>
      </UML:Association>
   </XMI.content>
```

```
| </XMI>


<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<XMI version="1.1" xmlns:go="">
   <XMI.header>
      <XMI.model xmi.name="BICOID" href="BICOID.xml"/>
      <XMI.metamodel xmi.name="" href=""/>
   </XMI.header>
   <XMI.content>
      <go:PROTEIN name="BICOID" length="422">
         <go:PROTEIN.gene>
            <go:GENE name="Bicoid"/>
         </go:PROTEIN.gene>
         <go:PROTEIN.interactions>
            <go:INTERACTION>
               <go:INTERACTION.protein>
                  <go:PROTEIN name="BICOID"/>
               </go:INTERACTION.protein>
               <go:INTERACTION.gene>
                  <go:GENE name="Hunchback"/>
               </go:INTERACTION.gene>
            </go:INTERACTION>
         </go:PROTEIN.interactions>
      </go:PROTEIN>
   </XMI.content>
</XMI>
```

XMI constitue une première approche pour introduire des objets dans XML. S'il permet pratiquement l'échange de modèles UML son but n'est pas de manipuler le document XML comme un objet. La seule opération envisagée est la validation qui se fait (pour l'instant) uniquement à l'aide de DTD.

## XML-Schema

Dés la création d'XML, d'autres langages ont été développés afin de répondre aux limites des DTD et proposer une approche donnée à la place de l'approche documentaire des formalismes fondés sur les grammaires. Ces langages, SOX [Davidson& 1999] ou DCD [Bray& 1998b], ont été abandonnés au profit de XML-Schema.

XML-Schema est une paire de recommandations du W3C concernant le typage des documents XML :

— Schema-Structure [Thomson& 2001] propose un langage pour exprimer la structure des éléments XML en fonction d'autres éléments et de types de données;

— Schema-Datatypes [Biron& 2001] propose un ensemble de types de données de base (par exemple les dates) et la manière de les restreindre.

Plusieurs avantages sont espérés de ces extensions dont la simplification des DTD grâce à l'utilisation d'attributs XML typés et, plus généralement, une plus grande expressivité permettant aux analyseurs XML un contrôle de type plus précis sur des types plus étendus que ceux d'XML. Actuellement, les schémas permettent d'exprimer trois sortes d'entités : types simples (chaînes, entiers, dates…), types d'éléments (nommés types complexes) et éléments. Les types simples peuvent être raffinés (entiers positifs pairs non nuls) mais leur ensemble ne peut pas être étendu (impossible d'ajouter des séquences d'amino-acides par exemple). Les types

d'éléments sont organisés par une relation de sous-typage et les problèmes posés par la spécialisation de contenu ont été codifiés. Il est par ailleurs possible de poser des contraintes de cardinalité sur les séquences d'éléments. Les éléments, curieusement, ne peuvent se spécialiser les uns les autres, cette faculté étant réservée à leurs types. Voici un exemple de schéma qui définit deux types simples (`name` et `proteinName` — qui s'écrit en majuscule) l'un étant dérivé de l'autre, deux types d'éléments (`NamedElement` et `NamedSizedElement`) l'un étant dérivé de l'autre et trois sortes d'éléments (`GENE`, `PROTEIN` et `INTERACTION`).

```
<?xml version="1.0" ?>
<schema  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns="http://www.w3.org/2001/XMLSchema"
         xmlns:go="http://www.go.org"
         targetNamespace="http://www.go.org">

<!-- datatypes -->

<simpleType name="name">
  <restriction base="xsd:string"><maxLength value="10"/></restriction>
</simpleType>

<simpleType name="ProteinName">
  <restriction base="go:name">
    <pattern value="[A-Z]{1-10}"/>
  </restriction>
</simpleType>

<!-- types -->

<complexType name="namedEntity">
  <attribute name="name" type="go:name" use="required"/>
</complexType>

<complexType name="namedSizedEntity">
   <complexContent>
      <extension base="go:namedEntity"/>
      <attribute name="length" type="xsd:integer" default="0"/>
    </extension>
  </complexContent>
</complexType>

<!-- elements -->

<element name="GENE" type="go:namedEntity"/>

<element name="PROTEIN">
  <complexType>
    <complexContent>
      <extension>
        <complexType>
          <complexContent>
            <restriction base="namedSizedEntity">
               <attribute name="name" type="go:ProteinName"/>
               <attribute name="length" type="xsd:integer"
                         default="0"/>
            </restriction>
          </complexContent>
        </complexType>
        <sequence>
          <element ref="go:GENE" minOccurs="0"/>
          <element ref="go:INTERACTION" minOccurs="0"
                                        maxOccurs="unbounded"/>
```

```
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>

<element name="INTERACTION">
  <complexType>
    <sequence>
      <element ref="go:PROTEIN"/>
      <element ref="go:GENE"/>
    </sequence>
  </complexType>
</element>

</schema>
```

Le document correspondant à l'exemple initial décrit dans le schéma ci-dessus est inchangé. À l'instar de la transformation évoquée dans la section précédente, il est possible d'exporter des objets vers XML-Schema. Un document Schema-valide, à la différence d'un document simplement DTD-valide, n'est plus seulement bien structuré mais 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 un langage à objets :

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

— 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 [Ducournau& 1998] ou RELAX-NG qui ne disposent de pas de types primitifs.

## Xquery et son système de typage

XML devenant un modèle de description de données quasi-universel, il faut lui associer un langage de requêtes adapté, à l'image de tout modèle de description de données. Plusieurs propositions ont été faites qui ont contribué à l'émergence du langage Xquery [Boag& 2003]. Ce langage a été conçu pour prendre en compte l'hétérogénéité des données et offrir la possibilité d'exprimer des requêtes concises et intelligibles sur ces données.

La spécification actuelle de XQuery doit devenir une recommandation du W3C dans un futur proche. Globalement, XQuery peut être vu comme un langage fonctionnel aux nombreuses possibilités destinées à la manipulation des expressions arborescentes de XML. XQuery suit la recommandation selon laquelle un langage de requêtes pour XML doit avoir une syntaxe XML pour bénéficier de tous les avantages liés à XML dans la manipulation de requêtes (analyse syntaxique, génération, interrogation par le contenu). Il doit aussi avoir une syntaxe facilement accessible et intelligible pour un utilisateur. XQuery s'articule donc autour de trois langages : la

syntaxe externe (de surface), la syntaxe XML (XqueryX [Malhotra&□2003]) et l'algèbre formelle (relationnelle) associée [Fernández&□2003].
XQuery est un langage où le bloc de base est l'expression. Les expressions sont construites avec des constructeurs et peuvent se composer (s'imbriquer à la façon d'un langage fonctionnel), contiennent des variables et font appel à une riche bibliothèque de fonctions prédéfinies. Voici un exemple d'expression simple□

```
let $target := //go:INTERACTION/go:GENE
    return <TARGETGENES> {$target} </TARGETGENES>
```

qui sélectionne la liste des gènes sujets d'une interaction (`//go:INTERACTION/go:GENE`), via la variable `$target`, et la retourne dans l'élément XML `<TARGETGENES>`.
Le composant clé de XQuery est XPath, qui est un langage de description de chemins d'accès à des documents et leur contenu (éléments dans une structure XML). Ainsi, `//go:INTERACTION/go:GENE` qui désigne un chemin d'accès peut s'interpréter comme une requête valide en XQuery. En réalité, XQuery (version 1.0) est une extension de XPath (version 2.0) [Berglund&□2003], et toute expression syntaxiquement valide retourne le même résultat dans les deux environnements, le résultat étant lui-même en XML (une séquence de sous-arbres résultats et non un ensemble de nœuds comme en XPath 1.0). La valeur d'une expression est donc toujours une séquence d'un ou plusieurs éléments (valeur atomique ou composite par rapport aux types de XML-Schema).
Ces séquences peuvent être utilisées telles quelles comme dans l'exemple ci-dessus ou au contraire parcourues comme dans l'exemple, plus complexe, ci-dessous□

```
<results>
   {  for $protein in //go:PROTEIN
      let $intlist := $protein/go:INTERACTION
      return <protein
               name = {$protein/@go:name}
               nbint = {count($intlist)}>
               {for $int in $intlist
                  return <gene> {$int/go:GENE/@go:name} </gene> }
            </protein> }
</results>
```

qui retourne, sur le document cité en exemple, une liste d'élements de la forme□

```
<results>
   <protein name="BICOID" nbint="1">
      <gene>Hunchback</gene>
   </protein>
   …
</results>
```

Cet exemple fait appel à des structures de contrôle (`for`), de gestion d'environnement (`let`) ainsi que des fonctions prédéfinies (`count`).
XQuery est un langage fortement typé avec un système de types s'appuyant sur XML-Schema, qui comprend les types primitifs de XML-Schema, et des types composés. Le contrôle de type se fait selon une phase d'analyse et une phase

d'évaluation. La première dépend de la requête et du contexte statique (chaque expression lors de l'analyse se voit affecter un type statique). Les sémantiques statique et dynamique de XQuery sont définies formellement [Draper& 2003].
Si les données dépendent d'un schéma de XML-Schema, alors un processeur peut interpréter ce schéma et associer au moteur de requêtes un PSVI («post-schema validation infoset») pour en faire un type statique à la compilation.
La phase d'évaluation des expressions suit la phase d'analyse qui doit être satisfaite, et qui dépend des données d'entrée, et d'un contexte d'évaluation (un type dynamique est affecté à chaque expression lors de l'évaluation, type qui peut être plus spécifique que le type statique).
Les possibilités de XQuery sont nombreuses et plutôt proches du modèle de données de l'ODMG [Cattell 1994]. De plus, la spécification est ouverte et autorise les variations, ce qui permet un large éventail d'implantations possibles, tout en reposant toutefois sur un modèle rigoureux avec une sémantique bien définie.

## 4. Les langages du web sémantique

Les possibilités offertes par XML ouvrent la voie à l'utilisation d'XML pour la modélisation. Dans [Euzenat 1999], nous examinions différentes manières d'intégrer une représentation des connaissances en XML. Mais il a été jugé nécessaire, pour exprimer la «sémantique» des données du Web, de définir un nouveau langage qui permette d'annoter les documents XML de manière à les rendre intelligibles. C'est ainsi qu'est né RDF.
Après un développement rapide, RDF a été l'objet d'âpres débats avant d'être progressivement formalisé au point qu'il dispose maintenant d'une sémantique en théorie des modèles. Il a aussi été enrichi d'un langage permettant de spécifier les types de données RDF et fait l'objet d'extensions afin de construire des classes de plus en plus complexes.
On est dans la représentation des connaissances et, qui plus est, dans la représentation des connaissances structurée, celle des graphes conceptuels, des logiques de descriptions et des représentations des connaissances par objets. Quel chemin parcouru depuis le simple langage de balisage!

RDF

Le W3C développe RDF ("Resource description framework" [Lassila& 1999, Champin 2000, Klyne& 2003]) en tant que cadre général d'annotation des documents XML. Il n'est pas considéré comme une extension d'XML vers les objets, car les relations sont les constructeurs principaux qui sont appliqués aux classes (RDF est donc plutôt proche des graphes conceptuels). RDF permet de décrire les relations binaires entre ressources (qui peuvent être des documents ou parties de documents ou de chaînes de caractères). Les ressources comme les relations sont identifiées par des URI (on simplifiera cependant les relations en utilisant un simple nom). Le fait qu'une ressource puisse être la valeur d'une relation fait que les annotations RDF constituent un immense graphe à l'échelle du monde (grâce aux

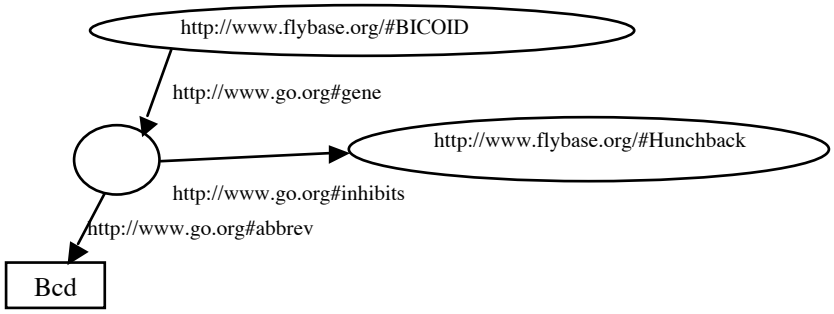
URI). RDF peut-être utilisé pour annoter des documents écrits dans des langages non structurés, ou comme une interface pour des documents écrits dans des langages ayant une structure similaire (des bases de données, par exemple). RDF est muni d'une syntaxe, et d'une sémantique. Bien que la sémantique — introduite récemment — suffise à définir quelles inférences sont valides, aucun mécanisme permettant de calculer celles-ci n'est proposé dans les recommandations.

Un document RDF est un ensemble de triplets de la forme *<sujet, prédicat, objet>*. Ce document sera codé en machine par un document RDF/XML [Beckett 2003] ou N3. Voici l'exemple original exprimé en RDF/XML.

```
<RDF  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:go="http://www.go.org">
   <go:PROTEIN ID="BICOID"/>
   <go:GENE ID="Bicoid"/>
   <go:INTERACTION ID="IN001"/>

   <Description about="#BICOID">
      <go:name>BICOID</go:name>
      <go:length>422</go:length>
      <go:gene resource="#Bicoid">
      <go:interactions>
         <sequence>
            <li resource="#IN001"/>
         </sequence>
      </go:interactions>
   </Description>
   <Description about="#Bicoid">
      <go:name>Bicoid</go:name>
   </Description>
   <Description about="#IN001">
      <go:source resource="#BICOID"/>
      <go:cible resource="#Hunchback"/>
   </Description>
</RDF>
```

Les éléments de ces triplets peuvent être des URIs, des littéraux ou des variables. Un tel ensemble peut être représenté de façon naturelle par un graphe (plus précisément un multi-graphe orienté étiqueté), où les éléments apparaissant comme sujet ou objet sont les sommets, et chaque triplet est représenté par un arc dont l'origine est son sujet et la destination son objet (voir figure).




La figure précédente présente une partie d'un document RDF. Les URIs identifient des ressources de manière unique. Notons que certaines ressources sont spécifiques à la biologie moléculaire (`go`), et que d'autres sont issues d'une ontologie dédiée à un organisme (`flybase`). Les objets d'un triplet qui sont des littéraux sont représentés dans un rectangle (ici, Bcd). Le sommet non étiqueté représente une variable. Intuitivement, ce graphe peut se comprendre comme « BICOID a pour gène un gène abrégé en Bcd qui inhibe l'objet Hunchback ». Cette sémantique « intuitive » ne suffisant pas à un traitement automatique, il faut munir les documents RDF d'une sémantique formelle.

La sémantique d'un document RDF est exprimée en théorie des modèles [Hayes 2003]. L'objectif est de donner des contraintes sur les mondes qui peuvent être décrits par un document RDF. L'utilisation de la théorie des ensembles pour décrire ces modèles a deux intérêts : la généricité de la notion d'ensemble (fondement des mathématiques) et son universalité (culture commune pour ceux qui vont s'intéresser à cette sémantique). Parallèlement à la validité d'un document par rapport à un schéma, la consistance d'un document par rapport à la sémantique du langage correspond à l'existence d'un modèle.
Un document RDF peut aussi être traduit en une formule de la logique positive (sans négation), conjonctive (sans disjonction), existentielle (sans quantificateur universel) du premier ordre sans symboles fonctionnels, dont les modèles sont identiques à ceux définis par la sémantique directe en théorie des modèles. À chaque triplet $<s, p, o>$ on associe la formule atomique $p(s, o)$, où $p$ est un nom de prédicat, et $s$ et $o$ sont des constantes si ces éléments sont des URIs ou des littéraux dans le triplet, et des variables sinon. Le document RDF se traduit par une formule qui est la fermeture existentielle de la conjonction des formules atomiques associées à ses triplets. Ainsi, le document RDF utilisé précédemment en exemple se traduit par la formule :

$$\exists x \, (gene(BICOID, x) \wedge abbrev(x, Bdc) \wedge inhibits(x, Hunchback))$$

L'information contenue dans un document RDF $R_1$ est déjà présente dans le document RDF $R_2$ si et seulement si la formule associée à $R_1$ est conséquence de celle

associée à $R_2$. Cette « traduction logique » de RDF permet de l'identifier à de nombreux autres paradigmes de raisonnement : la logique, bien sûr, mais aussi les bases de données (Datalog positif) ou les graphes conceptuels.

Bien qu'un mécanisme d'inférence correct et complet par rapport à la sémantique (on ne trouve que des conséquences et toutes les conséquences) soit évoqué dans les propositions du W3C, ceci n'entre pas dans la standardisation. L'objectif est de laisser la plus grande liberté à ceux qui vont implémenter des outils. Le rapprochement avec les graphes conceptuels simples permet cependant de préciser un tel mécanisme de raisonnement. Il s'agit d'un homomorphisme de graphes étiquetés, pour lequel des algorithmes efficaces (bien qu'il s'agisse d'un problème NP-complet) ont été développés [Baget 2003].
RDF propose aussi certains mots-clés réservés, qui permettent de donner une sémantique particulière à des ressources. Ainsi, on peut représenter des ensembles d'objets (*bag*), des listes (*sequence*), des relations d'arité quelconque (*value*)… Ce ne sont cependant pas de réelles extensions du langage présenté ci-dessus, puisqu'une transformation (la réification) permet d'exprimer cette « sémantique étendue » dans le langage de base : $R_1$ est une conséquence (sémantique étendue) de $R_2$ si et seulement si *réif*($R_1$) est une conséquence (au sens précédent) de *réif*($R_2$).

## RDF-Schema

RDFS (pour RDF-Schema [Brickley & 1999, Brickley & 2003]) a pour but d'étendre le langage RDF en décrivant plus précisément les ressources utilisées pour étiqueter les graphes. Pour cela, il fournit un mécanisme permettant de spécifier les classes dont les ressources seront des instances. RDFS s'écrit avec des triplets RDF, en définissant la sémantique de nouveaux mots-clés, comme:

- <*go:Protein rdf:type rdfs:Class*> la ressource *go:Protein* a pour type *rdfs:Class* et est donc une classe ;
- <*fb:BICOID rdf:type go:Protein*> la ressource *fb:BICOID* est une instance de la classe *go:Protein* ;
- <*go:Protein rdfs:subClassOf go:Sequence*> la classe *go:Protein* est une sous-classe de *go:Sequence*, toutes les instances de *go:Protein* sont donc des instances de *go:Sequence;*
- <*go:gene rdf:type rdfs:Property*> affirme que <*go:gene*> est une relation (une ressource utilisable pour étiqueter les arcs) ;
- <*go:gene rdfs:range go:Gene*> affirme que toute ressource utilisée comme objet (ou co-domaine) d'un arc étiqueté par *go:gene* sera une instance de la classe *go:Gene*.

Un fragment de schéma pour le document présenté précédemment est :

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

<RDF  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

   <rdfs:Class rdf:ID="PROTEIN"/>
```

```
   <rdfs:Class rdf:ID="GENE"/>
   <rdfs:Class rdf:ID="INTERACTION"/>

   <rdfs:Property rdf:ID="name">
      <rdfs:domain rdf:resource="rdf-syntax-ns#Resource"/>
      <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/>

</RDF>
```

RDF-Schema dote aussi RDF d'un méta-modèle proche de celui de SmallTalk-76 (voir [Masini&□1989]) dans lequel Relation et Class sont des sous-classes de Resource qui est elle-même une classe dont Relation et Class sont des instances. Les classes et relations sont ainsi utilisables comme n'importe quelle ressource (elles peuvent être en relation, etc.).

La proximité entre RDF-Schema et les graphes conceptuels conduit à réutiliser les outils développés dans ce contexte. C'est ainsi que CORESE, dont une extension est présentée dans un article suivant, constitue un système de requêtes pour RDF fonctionnant par traduction de RDF vers les graphes conceptuels, raisonnement sur les graphes et traduction des réponses en RDF.

La continuité entre le monde XML et RDF est loin d'être idéale. Des efforts ont été développés pour appliquer une sémantique RDF sur une syntaxe XML (ou plutôt sur le modèle de données de Xquery) permettant simultanément des traitements syntaxiques (transformations, validité) et des traitements sémantiques (déduction) [Patel-Schneider&□2002]. Qui plus est cette sémantique peut aussi être appliquée partiellement à XML-Schema ce qui permet de réduire l'ampleur des spécifications à décrire.

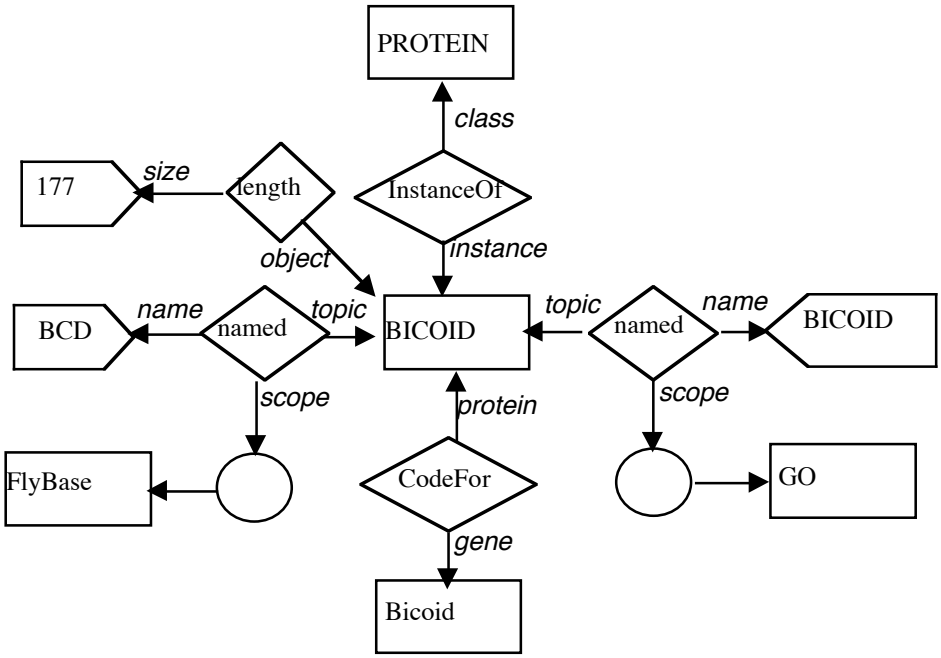
## Cartes topiques

Le modèle des cartes topiques (“Topic maps” [Biezunski&□1999]) est un standard ISO provenant de HyTime dont le but était d’annoter les documents multimédia. Issu de SGML, elles se sont vu récemment attribuer une syntaxe XML (XTM). Par ailleurs, un groupe de l’ISO s’occupe de définir un langage de requêtes pour les cartes topiques (TMQL).

Les cartes topiques sont bâties autour de quatre notions primitives (nous faisons ici abstraction des sujets)□

- Les concepts (ou “topics”) que l’on peut comprendre comme des individus des langages de représentation des connaissances.
- Les noms donnés aux concepts□ l’une des originalités des cartes topiques est la séparation des concepts et de leurs noms. Cela permet d’avoir plusieurs noms pour le même concept (et donc d’avoir des cartes topiques multilingues) et des noms partagés par plusieurs concepts.
- Les occurrences sont des “proxys” d’entités externes qui peuvent ainsi être indexés par les concepts (où les entités littérales lorsque celles-ci sont représentables).
- Les portées, qui sont parfois vues comme une quatrième dimension, permettent de spécifier le contexte dans lequel une relation est valide.

Par exemple, le concept de Protéine (PROTEIN) est instancié par BICOID, il a pour nom «□BICOID□ dans la portée de «□GeneOntology□ et «□BCD□ dans celle de «□FlyBase□.



Si ces quatre dimensions sont spécifiées de manière indépendante, elles sont en réalité interdépendantes : les concepts et les noms ont des portées, les concepts ont des noms, les portées sont des ensembles de concepts…

Dans la nouvelle syntaxe des cartes topiques, celles-ci sont représentées par des graphes comprenant 3 types de nœuds (concept, association, portée) et un certain nombre de types d'arcs (instance, occurrence, portée, nom). Les relations sont représentées par des nœuds dont les arcs sortants portent des étiquettes identifiant leur rôle. Par ailleurs, différentes interprétations sont données à ces primitives suivant les étiquettes placées sur les arcs et les nœuds. Autant dire que les cartes topiques ne disposent pas d'une sémantique claire et que, au contraire, leurs concepteurs ont tendance à considérer que la richesse du langage tient dans les interprétations multiples que l'on peut en faire.
Ceci ne le rend pas un candidat très souhaitable pour le web sémantique malgré des qualités indéniables. Il existe cependant des outils permettant de tirer parti de manière utile des cartes topiques qui sont utilisés dans un certain nombre d'applications [Le Grand 2002].

## Le langage d'ontologies pour le web OWL

RDF, langage dédié à l'expression d'assertions sur les relations entre objets, s'est heurté à la nécessité de définir les propriétés des classes dont ces objets sont instances. Cependant, l'extension à RDFS ne fournit que des mécanismes très primitifs pour spécifier ces classes. Le langage OWL [Dean & 2003], quant à lui, est dédié aux définitions de classes et de types de propriétés, et donc à la définition d'ontologies. Inspiré des logiques de descriptions il fait suite à de nombreuses initiatives (OIL et DAML-ONT ayant donné DAML+OIL [van Harmelen & 2001] ou DLML [Euzenat 2001]). OWL fournit un grand nombre de constructeurs permettant d'exprimer de façon très fine les propriétés des classes définies. La rançon de cette expressivité est l'indécidabilité du langage obtenu en considérant l'ensemble de ces constructeurs. C'est pour cela que OWL a été fractionné en trois langages distincts :

- OWL Lite ne contient qu'un sous-ensemble réduit des constructeurs disponibles, mais son utilisation assure que la comparaison de types pourra être calculée (un problème de NP, donc « simple » en représentation des connaissances) ;
- OWL DL contient l'ensemble des constructeurs, mais avec des contraintes particulières sur leur utilisation qui assurent la décidabilité de la comparaison de types. Par contre, la grande complexité de ce langage (un de ses fragments est PSPACE-complet) semble rendre nécessaire une approche heuristique ;
- OWL Full, sans aucune contrainte, pour lequel le problème de comparaison de types est vraisemblablement indécidable.

La syntaxe d'un document OWL est donnée par celle des différents constructeurs utilisés dans ce document. Elle est le plus souvent donnée sous la forme de triplets

RDF. À chaque constructeur est associée une sémantique en théorie des modèles. Elle est directement issue des logiques de descriptions [Patel-Schneider&□2003]. La sémantique associée aux mots-clés de OWL est plus précise que celle associée au document RDF représentant une ontologie OWL (elle permet plus de déductions). Ainsi, les documents précédents peuvent être régis par l'ontologie OWL-DL suivante:

```
<Ontology   xmlns="http://www.w3.org/2002/07/owl#"
            xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
            xmlns:xsd ="http://www.w3.org/2001/XMLSchema#"

<Class rdf:ID="GENE">
   <rdfs:subClassOf rdf:resource="http://www.go.org#sequence"/>
   <rdfs:subClassOf>
      <Restriction>
         <onProperty rdf:resource="#name"/>
         <maxCardinality>1</maxCardinality>
      </Restriction>
      <Restriction>
         <onProperty rdf:resource="#name"/>
         <rdfs:domain rdf:resource="xsd:string"/>
      </Restriction>
   </rdfs:subClassOf>
</Class>

<Class rdf:ID="PROTEIN">
   <rdfs:subClassOf rdf:resource="http://www.go.org#sequence"/>
   <rdfs:subClassOf>
      <Restriction>
         <onProperty rdf:resource="#domain"/>
         <rdfs:domain><owl:Class rdf:about="#GENE"/></rdfs:domain>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <Restriction>
         <onProperty rdf:resource="#interactions"/>
         <rdfs:domain><Class rdf:about="#INTERACTION"/></rdfs:domain>
      </Restriction>
   </rdfs:subClassOf>
</Class>

<Class rdf:ID="INTERACTION">
   <rdfs:subClassOf rdf:resource="http://www.go.org#process"/>
   <rdfs:subClassOf>
      <Restriction>
         <onProperty rdf:resource="#target"/>
         <rdfs:domain><Class rdf:about="#GENE"/></rdfs:domain>
      </Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <Restriction>
         <onProperty rdf:resource="#source"/>
         <rdfs:domain><Class rdf:about="#PROTEIN"/></rdfs:domain>
      </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <Restriction>
         <onProperty rdf:resource="#target"/>
         <cardinality>1</cardinality>
      </Restriction>
   </rdfs:subClassOf>
  </rdfs:subClassOf>
```

```
  ...
</Class>
```

Nous donnons ci-dessous un panorama des constructeurs utilisés dans OWL, dans une syntaxe simplifiée (les mots-clés réservés de OWL sont préfixés de OWL)□

OWL Lite

- reprend tous les constructeurs de RDF (c'est-à-dire fournit des mécanismes permettant de définir un individu comme instance d'une classe et de mettre des individus en relation),
- utilise les mots-clés de RDFS (*rdfs:subClassOf, rdfs:Property, rdfs:subPropertyOf, rdfs:range, rdfs:domain*), avec la même sémantique,
- permet de définir une nouvelle classe (*owl:Class*) comme étant plus spécifique ou équivalente à une intersection d'autres classes,
- *owl:sameIndividualAs* et *owl:differentIndividualFrom* permettent d'affirmer que deux individus sont égaux ou différents,
- des mots-clés permettent d'exprimer les caractéristiques des propriétés□ *owl:inverseOf* sert à affirmer qu'une propriété *p* est l'inverse de *p'* (dans ce cas, le triplet *<s p o>* a pour conséquence *<o p' s>*)□ d'autres caractéristiques sont la transitivité (*owl:TransitiveProperty*), la symétrie (*owl:SymmetricProperty*),
- *owl:allValuesFrom* associe une classe *C* à une propriété *P*. Ceci définit la classe des objets *x* tels que si *<x P y>* est une relation, alors la classe de *y* est *C* (quantification universelle de rôle en logique de descriptions). *owl:someValuesFrom* encode la quantification existentielle de rôle,
- *owl:minCardinality* (resp. *owl:maxCardinality*) associe une classe *C*, une propriété *P*, et un nombre entier *n*. Ceci définit la classe des objets *x* tels qu'il existe au moins (resp. au plus) *n* instances différentes *y* de *C* avec *<x P y>*. Pour des raisons d'efficacité algorithmique, OWL Lite ne permet d'utiliser que des entiers égaux à 0 ou 1. Cette restriction est levée dans OWL DL.

OWL DL

- reprend tous les constructeurs d'OWL Lite,
- permet tout entier positif dans les contraintes de cardinalité,
- *owl:oneOf* permet de décrire une classe en extension par la liste de ses instances,
- *owl:hasValue* affirme qu'une propriété doit avoir comme objet un certain individu,
- *owl:disjointWith* permet d'affirmer que deux classes n'ont aucune instance commune,
- *owl:unionOf* et *owl:complementOf* permettent de définir une classe comme l'union de deux classes, ou le complémentaire d'une autre classe.

OWL Full

- reprend tous les constructeurs d'OWL DL,

- reprend tout RDF-Schema,
- permet d'utiliser une classe en position d'individu dans les constructeurs.

Nous n'avons pas cité ici certains constructeurs, qui peuvent être trivialement implémentés grâce à ceux que nous avons évoqués (par exemple *owl:sameClassAs,* servant à affirmer que deux classes sont identiques, peut être écrit grâce à deux *rdfs:subClassOf*).

Des moteurs d'inférence ont déjà été implémentés pour des sous-ensembles significatifs de OWL DL (dans le cadre des logiques de descriptions) et sont utilisés dans divers outils (OilEd, Protégé…).

## 5. Conclusion

Ce bref tour d'horizon de technologies très changeantes et se nourrissant de toutes les influences disponibles et en particulier des objets n'a fait qu'effleurer les problèmes posés par les développements actuels du Web.
Comme décrit dans [Euzenat 2000], les efforts déployés pour ces développements intègrent progressivement des techniques et des tâches provenant des objets. Ce qui peut se résumer par le tableau suivant:

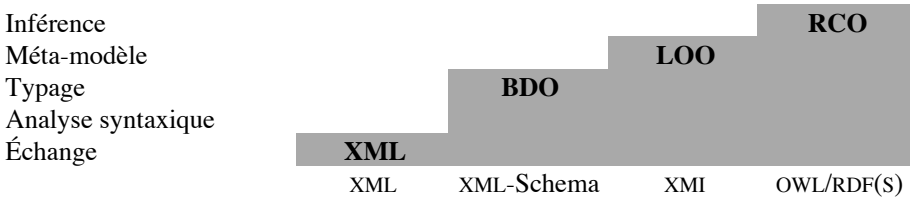



**Tableau 1.** *Répartition des tâches entre* XML *et objets dans les extensions présentées ci-dessus.*

Ainsi, sur les aspects syntaxiques, l'évolution d'XML le rapproche d'un langage de représentation des connaissances 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 les deux colonnes du milieu, même s'il peut être critiqué n'est pas à négliger. Il permet effectivement de décrire les documents dans un langage qui comprend un sous-ensemble important de la syntaxe des langages à objets.

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).

## 6. Remerciements

Merci à Raphaël Troncy pour sa relecture attentive et pour nous avoir fait partager sa connaissance d'XML-Schema.

## 7. Références

[Al Hulou & 2000] Rim Al Hulou, Amedeo Napoli, Emmanuel Nauer, XML : un formalisme de représentation intermédiaire entre données semi-structurées et représentations par objets, Actes 6e journées langages et modèles à objets, Mont-Saint-Hilaire (CA), pp75-90, 2000

[Apparao & 1998] Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, Lauren Wood (eds.), Document Object Model Level 1, 1998, http://www.w3.org/TR/REC-DOM-Level-1

[Baget 2003] Jean-François Baget, Simple conceptual graphs revisited: hypergraphs and conjunctive types for efficient projection algorithms, Actes International Conference on Conceptual Structures (ICCS), Dresden (DE), 2003 à paraître

[Beckett 2003] Dave Beckett (éd., RDF/XML Syntax Specification (Revised), W3C Working Draft, 2003, http://www.w3.org/TR/rdf-syntax-grammar

[Berners-Lee & 2001] Tim Berners-Lee, Jim Hendler, Ora Lassila, The semantic web, *Scientific american* 284(5):34-43, 2001 http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html

[Berners-Lee & 1998] Tim Berners-Lee, Roy Fielding, Larry Masinter, Uniform Resource Identifiers (URI): Generic Syntax, Request for Comments 2396, IETF, 1998 http://www.ietf.org/rfc/rfc2396.txt

[Berglund & 2003] Anders Berglund, Scott Boag, Mary Fernández, Michael Kay, Jérôme Siméon (éds.), XML Path Language (XPath) 2.0, Working Draft, W3C, 2003 http://www.w3.org/TR/xpath20/

[Biezunski & 1999] Michel Biezunski, Martin Bryan, Steven Newcomb (eds.), ISO/IEC 13250:2000 Topic Maps: Information Technology — Document Description and Markup Languages, 1999

[Biron & 2001] Paul Biron, Ashok Malhotra (éds.), XML Schema part 2: datatypes, Recommendation, W3C, 2001 http://www.w3.org/TR/XMLschema-2/

[Boag & 2003] Scott Boag, Don Chamberlin, Mary Fernández, Daniela Florescu, Johnathan Robie, Jérôme Siméon, Philip Wadler (éds.), XQuery 1.0: An XML Query Language, Working Draft, W3C, 2003 http://www.w3.org/TR/xquery/

[Bosak & 1999] Jon Bosak, Tim Bray, Le XML, Pour la science 261, 1999. http://www.pourlascience.com/numeros/pls-261/art-4.htm

[Bray & 1998a] Tim Bray, Jean Paoli, Michael Sperberg-McQueen (éds.), Extensible Markup Language (XML) 1.0, Recommendation, W3C, 1998 http://www.w3.org/TR/REC-XML

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

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

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

[Brickley&□2003] Dan Brickley, Ramanathan Guha (éds.), RDF Vocabulary description language 1.0: RDF Schema, Working draft, W3C, 2003 http://www.w3.org/rdf-schema

[Brownell□2002] David Brownell, SAX2, O'Reilly, Sebastopol (CA US), 2002

[Cattel□1994] Rick Cattell, The Object Database Standard: ODMG-93, Morhan-Kaufmann, San Francisco (CA US), 1994

[Champin□2000] Pierre-Antoine Champin, RDF tutorial, 2000 http://www710.univ-lyon1.fr/~champin/rdf-tutorial/

[Clark&□2001] James Clark, Makoto Murata, RELAX NG Specification, Oasis commitee specification, 2001 http://www.oasis-open.org/committees/relax-ng/spec-20011203.html

[Davidson&□1999] Andrew Davidson, Matthew Fuchs, Mette Hedin, Mudita Jain, Jari Koistinen, Chris Lloyd, Murray Maloney, Kelly Schwarzhof, Schema for object-oriented XML, Note, W3C, 1999 http://www.w3.org/1999/07/NOTE-SOX-19990730

[Dean&□2003] Mike Dean, Guus Schreiber (éds.), OWL Web Ontology Language: Reference, W3C Working Draft, 2003 http://www.w3.org/TR/owl-ref/

[Draper&□2003] Denise Draper, Peter Fankhauser, Mary Fernández, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, Philip Wadler (éds.), XQuery 1.0 and XPath 2.0 Formal Semantics, Working Draft, W3C, 2003 http://www.w3.org/TR/xquery-semantics/

[Ducournau&□1998] Roland Ducounau, Jérôme Euzenat, Gérald masini, Amedeo Napoli (éds.), Langages et modèles à objets□ état des recherches et perspectives, INRIA, Rocquencourt (FR), 1998

[Euzenat□1999]□Jérôme Euzenat, La représentation de connaissance est-elle soluble dans le Web?, *Document numérique* 3(3-4):351-367, 1999

[Euzenat□2000]□Jérôme Euzenat, XML est-il le langage de représentation de connaissance de l'an 2000?, Actes 6° journées langages et modèles à objets, Mont-Saint-Hilaire (CA), pp59-74, 2000

[Euzenat□2001]□Jérôme Euzenat, Preserving modularity in XML encoding of description logics, Deborah McGuinness, Peter Patel-Schneider, Carole Goble, Ralph Möller (éds.), Proc. 14th workshop on description logics (DL), Stanford (CA US), pp20-29, 2001 http://ceur-ws.org/Vol-49/Euzenat-20start.ps

[Fernández& 2003] Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, Norman Walsh (éds.), XQuery 1.0 and XPath 2.0 Data Model, Working Draft, W3C, 2003 http://www.w3.org/TR/xpath-datamodel/

[Hayes 2003] Patrick Hayes (éd.), RDF Semantics, W3C Working Draft, 2003 http://www.w3.org/TR/rdf-mt/

[Klyne& 2003] Graham Klyne, Jeremy Carroll (éds.), Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Working Draft, 2003 http://www.w3.org/TR/rdf-concepts/

[Le Grand 2002] Bénédicte Le Grand, Extraction d'information et visualisation de systèmes complexes sémantiquement structurés, Thèse d'informatique, Université Pierre et Marie Curie, Paris (FR), 2002 http://www-rp.lip6.fr/~blegrand/these.pdf

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

[Malhotra& 2003] Ashok Malhotra, Jonathan Robie, Michael Rys (éds.), XML Syntax for XQuery 1.0 (XQueryX), Working Draft, W3C, 2003 http://www.w3.org/TR/xqueryx

[Masini& 1989] Gérald Masini, Amedeo Napoli, Dominique Colnet, Daniel Léonard, Karl Tombre, Les langages à objets, InterÉditions, Paris (FR), 1989

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

[OMG 2000] Object Management group, XML Meta Object Facility (MOF) specification 1.3, OMG, 2000 http://www.omg.org/cgi-bin/doc?formal/00-04-03

[OMG 2002] Object Management group, XML Metadata Interchange (XMI) 1.2, OMG, 2002 http://http.omg.org/technology/documents/formal/xmi.htm

[Patel-Schneider& 2002] Peter Patel-Schneider, Jérôme Siméon, Building the Semantic Web on XML, *Lecture notes in computer science* 2342:147-161, 2002

[Patel-Schneider& 2003] Peter Patel-Schneider, Patrick Hayes, Ian Horrocks (éds.), OWL Web Ontology Language: Abstract Syntax and Semantics, W3C Working Draft, 2003 http://www.w3.org/TR/owl-semantics/

[Thompson& 2001] Henry Thompson, David Beech, Murray Maloney, Noah Mendelsohn, (éds.), XML Schema part 1: structures, Recomendation, W3C, 2001 http://www.w3.org/TR/XMLschema-1/

[van Harmelen& 2001] Frank van Harmelen, Peter Patel-Schneider, Ian Horrocks (éds.), Reference description of the DAML+OIL ontology markup language, W3C, 2001 http://www.daml.org/2001/03/reference.html