D'unicode au web sémantique : les dernières technologies du web au service de la terminologie

Yannis Haralambous

D'UNICODE AU WEB SÉMANTIQUE : LES DERNIÈRES TECHNOLOGIES DU WEB AU SERVICE DE LA TERMINOLOGIE

Yannis Haralambous
Département Informatique
Télécom Bretagne – Institut Télécom
29238 Brest Cedex 3
yannis.haralambous@telecom-bretagne.eu

Abstract

In the first part of this paper we present the Unicode character encoding and we discuss the manner in which human writing has been modelized by the computer. In the second part, inspired by Tim Berners-Lee's "semantic cake", we present the different layers of his vision of the (future) Web. These layers are added upon Unicode so that, in some years, we will obtain a coherent set of logical interacting layers. We will talk about namespaces, XML, RDF, ontologies. This paper is an introduction to these issues, intended for non-specialists.

Keywords: Unicode, Web technologies, semantic cake, XML, RDF, ontologies.

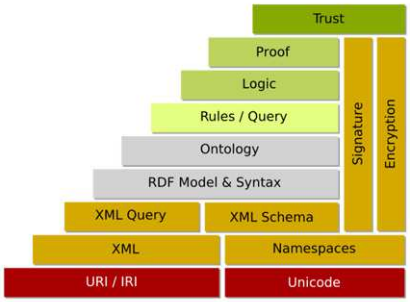
Introduction, plan de l'article

Há um destino igual, porque é abstracto, para os homens e para as coisas
— uma designação igualmente indiferente na álgebra do mistério.

E assim, contempladores iguais das montanhas e das estátuas, gozando os dias como os livros, sonhando tudo, sobretudo, para o converter na nossa íntima substância, faremos também descrições e análises, que, uma vez feitas, passarão a ser coisas alheias, que podemos gozar como se viessem na tarde.

FERNANDO PESSOA, *Livro do desassossego*

Voici comment Tim Berners-Lee, inventeur du Web dans les années 1990, a décrit en 2000 [1] sa vision du Web futur :



Chaque boîte correspond à une technologie spécifique (Unicode, XML, espaces de nommage, etc.) ou à une méthode générale (règles, logique, preuve).

Dans la première partie de cet article, nous nous étalerons sur la partie droite de la couche de base : la norme Unicode. Puis, dans la deuxième partie, nous aborderons progressivement toutes les couches allant du bas vers le haut (des données vers la connaissance).

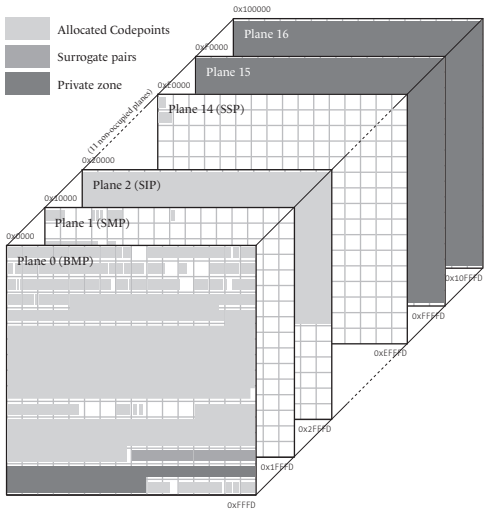
1 La norme Unicode

La norme Unicode (1991) [16, 11, 10] se donne comme but de coder le texte, dans toutes les écritures et avec tous les symboles nécessaires à tout type de texte. Elle constitue une réponse unique aux problèmes résultant de la multiplicité des codages (nationaux, commerciaux, sauvages, ...).

1.1 Les concepts de base d'Unicode

Pour la transmission et le stockage de données textuelles, Unicode décompose le texte écrit en *caractères*, ceci constitue le niveau atomique de subdivision du texte.

Unicode est divisé en 17 plans de 65 536 positions, on a donc, au total, 1 114 111 « cases ». À peine un dixième de celles-ci est occupé par des caractères.



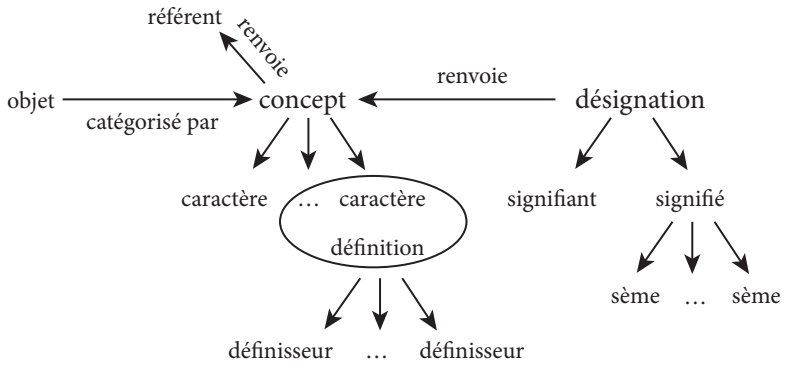
Un texte est une suite de caractères concaténés dans la mémoire de la machine (on les appelle *chaînes de caractères*). L'ordre des caractères est « logique » : c'est l'ordre attribué aux cellules mémoire qui les contiennent.

À un moment donné de sa vie, le texte est lu par l'homme, les caractères Unicode sont alors représentés par des images. L'image d'un caractère est appelée *glyphe*. L'image d'une chaîne de caractères est une chaîne de glyphes.

Dans de telles chaînes, les glyphes *interagissent* (cf. section 1.3). Mis à part l'interaction des glyphes, Unicode définit également un certain nombre d'*opérations* sur les caractères (cf. section 1.6).

1.2 Des analogies avec les outils utilisés en terminologie

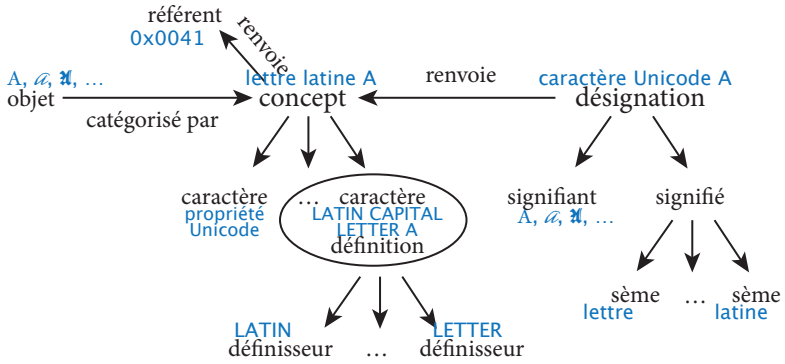
Dans *Entre signe et concept* [7], Loïc Depecker décrit les notions de base de la terminologie et leurs relations. Voici un schéma inspiré de ce texte :



Si nous en parlons ici, c'est parce que nous avons constaté des analogies avec les notions qui interviennent dans la norme Unicode. En effet, pour chaque caractère Unicode on dispose :

- d'une serie d'*objets* (les occurrences du caractère dans le monde réel) ;
- d'un *concept* linguistique, qui constitue, d'une certaine manière, l'« essence » du caractère ;
- d'une définition « officielle » du caractère par Unicode, qui est une parmi ses nombreuses propriétés (que l'on peut comparer aux *définisseurs* de Depecker) ;
- d'une valeur numérique dans la table Unicode, qui joue le rôle de *référent* (et même de *taxon*, puisque cette valeur appartient à une colonne de table Unicode, qui appartient à un bloc, qui appartient à un plan) ;
- du caractère Unicode « abstrait » que l'on peut qualifier de *désignation* du concept linguistique ;
- des différents glyphes représentant le caractère (qui sont les *signifiants*) ;
- de l'information véhiculée par le caractère, qui en est le *signifié*.

Voici un schéma qui superpose les notions liées à la norme Unicode à celles décrites par Depecker :



Ainsi donc, dans Unicode,

- un caractère est identifié par sa *description* (← définition) ;
- outre la description, un caractère possède des *propriétés* (← caractères du concept) ;
- les propriétés peuvent être *normatives* ou *informatives* (← caractères essentiels ou non) ;
- au niveau du stockage et de la transmission, un caractère est représenté par un nombre, appelé sa *position* (← référent) ;
- ce référent est même un taxon, puisque Unicode est subdivisé en *plans*, *tables*, *colonnes*, *positions* (en tout : 1 114 111 positions) ;
- l'image d'un caractère est appelée un *glyphe* (← signifiant).

1.3 Interaction des glyphes

Comme toute société, celle des glyphes a ses conventions. Les glyphes détestent la promiscuité. Ils se placent à une distance qui dépend de leurs formes respectives, l'opération qui consiste à les éloigner ou à les rapprocher dans ce but, est appelée *crénage*. Quand le crénage ne suffit pas pour gérer les rapports entre deux glyphes, dans un souci d'optimisation de la lecture, il arrive que certains glyphes s'« accouplent », par groupes de 2, 3, ou plus, on appelle cela des *ligatures* (exemple : 'f' suivi de 'i' donne 'fi'). Un accouplement historique de glyphes peut finir par être promu en caractère (exemples : ß, œ, ij, ce dernier étant un *digraphe*).

Dans certains cas, la morphologie d'une langue agit sur le comportement des glyphes. Ainsi, la langue allemande proscrit les ligatures entre les composantes d'un mot : comparer « Auflage » et « Pflege ».

1.4 Opérations sur les caractères

1.4.1 Diacritisation

Certains caractères ont la faculté de modifier le glyphe du caractère qui les précède. On les appelle *caractères combinatoires* (exemples : les accents, les voyelles courtes sémitiques, etc.).

Les caractères combinatoires peuvent être combinés entre eux et appliqués, à plusieurs, à un même caractère de base : ils agissent alors de l'intérieur vers l'extérieur.

Comme un caractère combinatoire agit toujours sur le caractère le précédant dans la chaîne, il faut garantir l'existence de celui-ci. Ainsi, une chaîne de caractères Unicode qui commence par un caractère combinatoire est *interdite*.

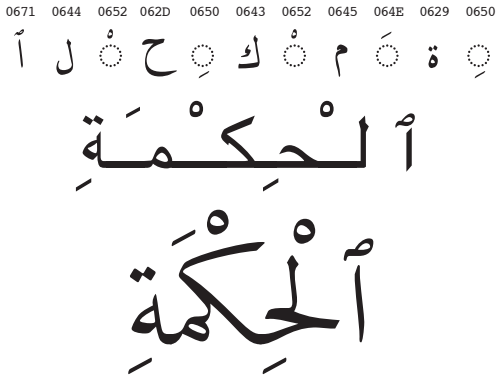
Il arrive, pour des raisons historiques, que certains combinaisons de lettre et d'accent peuvent être obtenues de deux manières différentes : en tant que caractères Unicode individuels, ou en tant que chaînes de deux caractères, dont le deuxième est combinatoire et représente l'accent. Il en résulte que l'on a un problème de multiplicité de représentation :

tiếng est-ce tiếng, tiê◌́ng ou tie◌̂◌́ng ?

Dans cet exemple, les caractères munis d'un cercle en pointillés sont combinatoires. La même lettre viêtnamienne peut donc être obtenue par un, deux ou trois caractères Unicode. Ces trois chaînes sont *canoniquement équivalentes* et on a recours à des mécanismes de normalisation pour n'en garder qu'une seule dans un texte.

1.5 Le passage des caractères aux glyphes

Pour pouvoir être lus par l'humain, les caractères doivent être rendus par des glyphes. Cette opération est plus ou moins simple, selon l'écriture. Ainsi, dans le cas de l'écriture arabe, on doit passer par deux étapes : une analyse contextuelle des caractères pour déterminer la forme du glyphe (isolé, initial, médian, final), et un assemblage des glyphes standard en ligatures (selon le type d'écriture : naskhi, coufique, nastaliq, diwaan, riqaa, etc.).



Cette opération doit conserver l'information sur le caractère original qui est à la source de chaque glyphe (et de chaque ligature de glyphes), puisqu'une interface utilisateur interactive doit permettre la sélection de glyphes et le copier-coller : sont alors copiés les caractères. Lorsqu'on copie une partie de mot, et que l'on la colle à un autre endroit, le système analyse de nouveau le contexte et repasse par les étapes décrites ci-dessus.

Un autre type de phénomène se rencontre dans les écritures de l'Inde et et du Sud-Est asiatique. Ici les voyelles (courtes) sont des caractères combinatoires qui se placent à gauche ou à droite de la consonne. l'absence de voyelle est signalée par un caractère Unicode spécifique (le virama), mais souvent deux consonnes sans voyelle intermédiaire forment une ligature. Techniquement, une consonne porte toujours une voyelle courte par défaut (le 'a') et c'est la présence du virama entre deux consonnes qui déclenche la formation de la ligature. Voici le mot « hindi » écrit dans l'écriture dévanagari :

Lors de la première étape le 'n' est suivi du 'd' et entre les deux on distingue le virama sous forme de petit trait. Sur la ligne du dessous, les deux lettres forment une ligature (la barre verticale du 'n' disparaît et sa partie horizontale est collée au 'd'), le virama n'est alors plus noté.

1.6 Opérations sur les chaînes de caractères

Diverses opérations sont définies sur les chaînes de caractères.

– Le *pli de casse* est la correspondance entre lettres bas-de-casse et capitales. Notons certaines correspondances spéciales : ß qui devient SS ou SZ en allemand (MASSE = la masse, MASZE = les mesures, pluriel de « Maß »), le 'i' qui devient 'İ' en turc, alors que c'est la lettre 'ı' (sans point) qui produit un 'I', les accents grecs qui disparaissent dans une écriture en capitales, le sigma final grec dont la forme capitale est celle du sigma médian, etc.

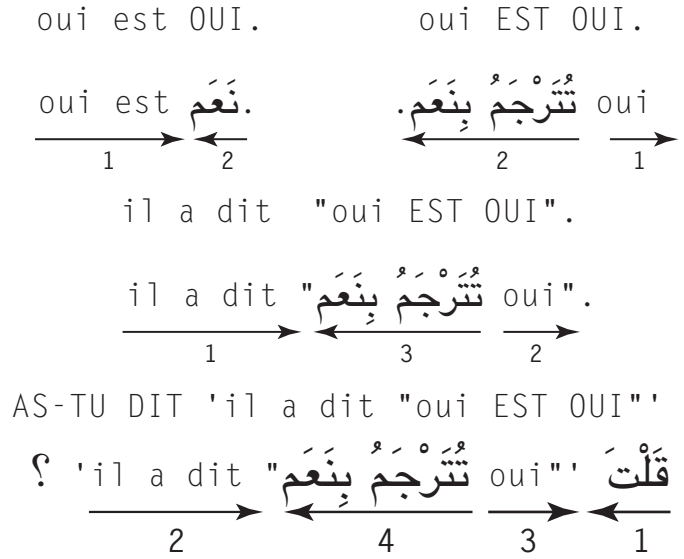
– Le *tri alphabétique* (UTS 10 [15]) dépendant de la langue, sur plusieurs niveaux (les lettres, les diacritiques, la ponctuation, les enrichissements). L'exemple classique de tri alphabétique selon les règles françaises est

$$\text{cote} < \text{côte} < \text{coté} < \text{côté}$$

– La *ségmentation* du texte (UAX 14 & 29 [17, 18]) : trouver les limites des « mots », « phrases », « paragraphes ».
– La *césure* (hors Unicode [12]) : on peut utiliser la technique des *motifs* pour la césure standard. Notons certains cas particuliers : backen → bak-ken (allemand), paral·lel → paral-lel (catalan), eighteen → eight-teen et record → rec-ord (nom) ou re-cord (verbe) (anglais), asszonnyal → asz-szony-nyal (hongrois), etc.

1.6.1 Cas particulier d'opération : l'algorithme bidirectionnel

L'arabe, l'hébreu, le syriaque, le thaâna, et quelques autres écritures s'écrivent horizontalement, de droite à gauche. Dans un texte multilingue il arrive donc que l'on ait à combiner des segments gauche-à-droite avec des droite-à-gauche. Se pose alors la question de la disposition de ses segments sur la page. La solution proposée par Unicode est l'*algorithme bidirectionnel* [19, 11]. Il s'agit d'un algorithme qui détermine, selon un certain nombre de critères, l'ordre graphique des glyphes d'une chaîne de caractères multidirectionnelle. L'exemple ci-dessous montre des cas de plus en plus complexes d'imbrication sémantique et les rapports entre ordre graphique et ordre de lecture (symbolisé par les flèches et les numéros) qui en découlent :



Dans la première phrase (« oui est *naam* »), le français (et donc le gauche-à-droite) est la langue du contexte, le mot arabe s'imbrique. La deuxième phrase (« oui *tatardzamou binaam* » = 'oui' se traduit par '*naam*') présente le cas inverse : l'arabe est la langue du contexte, le mot français s'y imbrique, il est donc écrit à droite, même s'il est prononcé en premier.

Les deux autres exemples présentent deux niveaux supplémentaires d'imbrication : « il a dit "oui *tatardzamou binaam*" », et « *qalta* "il a dit "oui *tatardzamou binaam*"" ? ». C'est la présence des guillemets qui permet à l'algorithme de détecter l'imbrication sémantique.

Dans certains cas, l'algorithme est incapable de détecter les relations entre les différents segments de texte. On dispose alors d'un certain nombre de caractères de contrôle qui nous permettent d'expliciter l'imbrication sémantique des segments.

1.7 Écritures idéographiques CJKV : l'unification

L'écriture idéographique a été inventée en Chine et puis, exportée au Japon, en Corée et au Viêtnam (dans ce dernier cas, elle a été surplantée par une écriture latine fortement diacritée). Une norme ISO (ISO 10646 Draft 1) qui était en phase de discussion lorsqu'Unicode a été développé, proposait de coder les caractères idéographiques chinois, japonais et coréens dans des zones distinctes du codage. Cela avait l'avantage d'éviter toute confusion entre ces trois langues, mais était considéré à l'époque comme plutôt gourmand en terme d'utilisation mémoire.

Vis-à-vis de cette norme, Unicode a innové en « unifiant » les idéogrammes chinois, japonais et coréens, selon les règles suivantes [11] :

1. le principe de *séparation des sources* : si deux idéogrammes occupent des positions distinctes dans le même codage, ils ne sont pas unifiés dans Unicode. Exemple : 剣劍劒劔劍釼, tous « épée » mais distincts dans JIS X 0208 ;
2. le principe de *distinction étymologique* : si deux idéogrammes sont étymologiquement différents, c'est-à-dire s'ils sont les dérivés historiques d'idéogrammes distincts, ils ne sont pas unifiés dans Unicode. Exemple typique : 土 (terre) et 士 (samouraï) ;
3. si deux idéogrammes qui ne satisfont pas aux conditions (1) et (2) ont des *formes abstraites identiques*, alors ils sont unifiés. Exemples : 周/周, 雪/雪, 酉/酉, 垂/垂, 朱/朱, 父/父, 八 / 八, 說/説.

Cette unification a permis de limiter le nombre de bits nécessaires pour représenter un caractère Unicode à 16 (ISO 10646 Draft 1 en exigeait 31), un argument commercial qui a su convaincre les industriel. Néanmoins, cet argument a été abandonné quelques années plus tard, quand Unicode s'est étendu à 21 bits.

1.7.1 Distinction entre caractère / radical / radical graphique

Les idéogrammes chinois sont formés à partir de 214 radicaux (comportant entre 1 et 17 traits). Unicode distingue parmi les radicaux en soi, les caractères formés des radicaux, et les radicaux dont les glyphes sont adaptés à l'utilisation à l'intérieur d'un caractère. Ainsi :

- `0x706b` 火 est un caractère idéographique qui signifie « feu » mais aussi « mardi », « mars », « flamme », « chaleur » ;
- `0x2f55` RADICAL CHINOIS FEU 火 est le radical n° 86, « feu ». Le glyphe de ce caractère est en tout point identique à celui du caractère « feu », mais il ne s'agit pas ici du caractère mais du radical ;
- `0x2ea3` RADICAL CJC FEU 灬 est la forme que prend ce radical lorqu'il est combiné avec d'autres radicaux.

1.7.2 En cas d'absence de caractère

Les caractères idéographiques forment un système organique. De nouveaux caractères sont inventés tous les jours (en tant que combinaisons des 214 radicaux) et attribués, par exemple, à des enfants nouveaux-nés ou à des chevaux de course prometteurs. Ces caractères accompagnent les personnes et les chevaux qu'ils désignent et obtiennent un degré de notoriété proportionnel à celui de leurs hôtes. Quelques centaines parmi eux se trouvent ainsi ajoutés à Unicode tous les ans. Comment gérer un système d'écriture qui change tous les jours ? Que faire quand un caractère manque à l'appel ? Face à ce problème nous avons constaté quatre attitudes possibles [11] :

– l'indignation : le signe (japonais) *geta* `0x3013` 〓 indique une absence de caractère sans donner aucune information sur celui-ci ;
– l'imagination : l'indicateur de variante idéographique `0x303e` 〾 (qui peut être interprêté comme suit : « ne vous étonnez pas si ce que vous lisez n'a aucun sens, voici à quoi ressemble le caractère manquant, et si vous n'êtes pas démesurément bête (ou étranger) le contexte vous permettra de le trouver »), il s'agit donc d'une approximation graphique du caractère manquant par un autre caractère, en faisant abstraction de leurs sémantiques ;
– l'assemblage (virtuel) : les *caractères de description idéographique* `0x2ff0-fb` : ⿰ ⿱ ⿲ ⿳ ⿴ ⿵ ⿶ ⿷ ⿸ ⿹ ⿺ ⿻ indiquent les assemblages à effectuer pour obtenir le caractère manquant à partir des caractères présents. Exemples :
  – ⿰女壬 (femme + 9e mois) signifie que l'on souhaite combiner horizontalement les deux caractères en question pour obtenir 妊 (grossesse) ;
  – ⿱宀女 (toit + femme) doit nous faire penser à 安 (tranquillité) ;
  – ⿴口大 (boîte + grand) à 因 (cause)
  – et ⿰女⿱女女 (femme + femme + femme) à 姦 (bruit).

  Notons que la transformation est mentale, l'interface logicielle n'étant pas censée faire l'assemblage visuel. On se contente juste de le suggérer, de la même manière que fonctionnent les émoticones comme `:-)`
– la précision : CDL [3] est un langage de description de caractères idéographiques basé d'une part sur les 39 traits fondamentaux de la calligraphie chinoise et d'autre part sur l'utilisation d'un système de coordonnées muni d'opérations d'homothétie et la réutilisation de caractères existants.

1.8 Les limites d'Unicode

Devant des écritures comme l'idéographique, on se rend vite compte des limitations d'Unicode, dues principalement au fait que l'informatique a été inventée à l'Occident, et se base sur des principes analytiques, dans la tradition de Démocrite. Les écritures occidentales procèdent d'abord à un découpage en parties atomiques (les lettres et autres signes), pour les réassembler ensuite pour former des nouvelles unités de sens. Le nombre de parties atomiques (appelé aussi alphabet) étant fini et bien délimité, l'infinie créativité de la langue se porte sur les groupements de lettres. Unicode s'intéresse à ces éléments atomiques et arrive ainsi à coder sans faille leurs groupements (les mots).

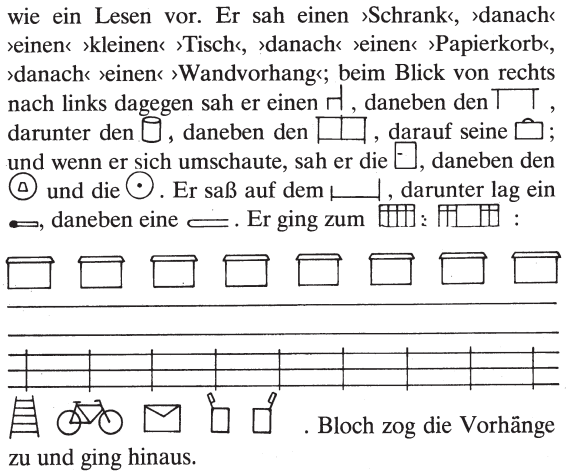
La situation est toute autre pour l'écriture idéographique où les caractères (au sens d'Unicode) sont déjà l'équivalent des mots occidentaux. Sans parler du fait que l'atomicité des idéogrammes en tant que caractères Unicode est également contestée puisque des variantes graphiques sont parfois utilisés pour les noms propres et acquièrent ainsi un poids sémantique.

Mais pour contester l'atomicité des caractères Unicode, nul besoin d'aller jusqu'en Extrême-Orient. La photo ci-dessous a été prise par l'auteur à Athènes [11] :

L'auteur de cette inscription a voulu écrire le mot PARKING (qui fait partie du vocabulaire néohellénique). Ce mot a la particularité de comporter des lettres explicitement grecques (ou latines) en début et en fin de mot, et des lettres communes aux deux alphabets à son milieu (K, I et N). Il a donc commencé par l'écrire en caractères grecs, puis est arrivé dans cette zone de pénombre bi-alphabétique, pour en sortir en alphabet latin. Ce fait divers pittoresque n'a rien d'étonnant dans un pays où les deux écritures se côtoient en permanence.

Mais comment coder ces caractères en Unicode ? K, I et N, sont ils grecs ou latins ? Ou peut-être intermédiaires avec un certain pourcentage d'appartenance à l'un ou l'autre alphabet ?

Autre exemple, qui pousse la réflexion un peu plus loin : l'extrait ci-dessous provient de l'ouvrage *L'angoisse du gardien de but au moment du penalty* de Peter Handke [9] :



L'auteur essaie de décrire les processus de pensée de son serial killer psychopathe. Les mots du texte sont d'abord placés entre guillemets pour être ensuite remplacés par des pictogrammes. Le texte devient alors une série de pictogrammes, arrêtée abruptement quand « Bloch tire les rideaux et sort ».

D'un point de vue technique, ces pictogrammes entrent dans le champ d'application d'Unicode : ils sont bel et bien utilisés dans un texte (et même dans un texte célèbres, ce qui n'est pas le cas de bon nombre de caractères Unicode !). Mais comment coder ces caractères, et comment les relier aux sémantiques correspondant aux intentions de l'auteur ?

2 Les autres couches du gâteau sémantique

2.1 Couche 1 : URI, URL, URN (les autres éléments atomiques)

Si un texte est formé de caractères, le Web, lui, est formé de documents, appelés des *ressources*. Une URI [2] est un *identifiant uniforme de ressource*. Il y a (au moins) deux manières d'identifier (et donc de désigner) une ressource :

1. par son emplacement sur le Web et le protocole d'accès : on a alors une URL (un *identifiant uniforme de localisation*) :

   `http://www.site.com/chemin/vers/fichier.php#mot-cle`
   `mailto:yannis.haralambous@telecom-bretagne.eu`

   etc.

2. ou alors, par son contenu : on a alors une URN (un *identifiant uniforme de nom*) :

   `urn:isbn:978-2-841-77273-5`

L'URN ci-dessus se réfère à un livre, il ne s'agit donc plus d'une ressource du Web, mais d'un document physique. On aurait pu s'attendre à ce que les URN s'étendent à plusieurs domaines pour identifier des objets et des concepts de plus en plus divers. En réalité, il n'en est rien. On verra par la suite que c'est la notion d'URL qui a été utilisée à cet effet, même si, en principe, son champ d'application se limite aux ressources informatiques sur le Web.

2.2 Couche 2 : Les documents XML

Cette couche nous donne les moyens de structurer les données (texte ou URIs) pour former des documents. Bien avant Tim Berners Lee, une autre grande figure de l'informatique, Charles F. Goldfarb, inventait la notion de document SGML [14], où la structure des données est celle d'une arborescence de blocs imbriqués et nommés. Le Consortium Web a pris cette norme et l'a simplifiée pour réduire la charge calculatoire, le résultat est XML [20]. Ainsi,

- un *document XML* est un ensemble de données, structuré en blocs (appelés *éléments*). Les éléments, ainsi que les autres ingrédients du document XML, munis de la relation d'inclusion, forment un arbre typé ;
- les principaux types de nœud sont : *élément*, *attribut*, *texte*, *document*, *commentaire*, *espace de nommage* ;
- les éléments et attributs ont des *noms*, formés par des caractères Unicode ;
- les nœuds de texte ont un contenu, formé par des caractères Unicode.

On peut se demander quel est le rôle de la langue dans un document XML. Grâce à l'attribut prédéfini `xml:lang` et aux normes ISO 639 (langues) et ISO 3166 (pays), on peut attribuer à chaque nœud (textuel ou structurel) une langue, ainsi que la donnée d'un pays. Cet attribut est générique (en jargon informatique : un attribut de classe), il est donc disponible pour toutes les instances XML. Selon l'utilisation que l'on fait de cet attribut, on peut considérer que sa valeur est héritée, ou non, par ses descendants.

2.2.1 Sérialisation de document XML

La définition de document XML que nous venons de donner est bien abstraite : des données subdivisées en blocs imbriqués. Il nous faut maintenant adapter cette définition à la linéarité logique de la mémoire de l'ordinateur.

Pour stocker ou transmettre un document XML, il faut le *sérialiser*, c'est-à-dire représenter sa structure dans un mode linéaire, sous forme d'une (potentiellement longue) chaîne de caractères.

On se pose donc la question : comment séparer les données décrivant la structure, de celles du contenu textuel (qui, *a priori*, peut être arbitraire) ? Autrement dit, comment rendre explicite la structure parmi les données ?

On utilise pour cela la notion de *balise*. Les balises d'un élément portent le nom de celui-ci et sont distinguées du texte par la présence des caractères spéciaux < et >. Ce sont ces caractères qui font ressortir la balise des données environnantes et la rendent identifiable sans ambiguïté.

La branche de l'arbre qui se trouve sous un nœud de type élément est entourée (dans la sérialisation) par des balises *d'ouverture* et *de fermeture* : `<body> ... </body>`

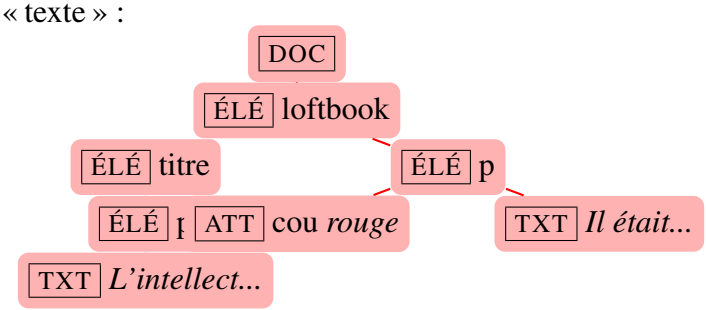
En ce qui concerne les attributs, ils sont notés à l'intérieur des balises d'ouverture :
`<img src="toto.gif"></img>`

Pour distinguer les balises du contenu textuel on utilise donc trois caractères dédiés :

– `<`, caractère de *début de balise*,
– `>`, caractère de *fin de balise*,
– `&`, caractère utilisé pour les *entités* ;

En tant qu'entités, on n'utilise plus aujourd'hui que les cinq *entités prédéfinies* `&lt;` (<) `&gt;` (>) `&amp;` (&) `&quot;` (") `&apos;` (').
Le rôle de celles-ci est de nous permettre d'obtenir, dans le contenu textuel d'un document, les trois caractères spéciaux <, > et & (les caractères " et ' protègent les valeurs d'attribut, ils ne sont pas spéciaux en dehors de ce contexte).

On dispose donc de deux manières de représenter un document XML : sous forme de tracé d'arbre ou en le sérialisant. Voici un exemple de document XML avec des nœuds de type « document » (lien non sérialisable qui se situe à la racine de l'arborescence), « élément », « attribut » et « texte » :



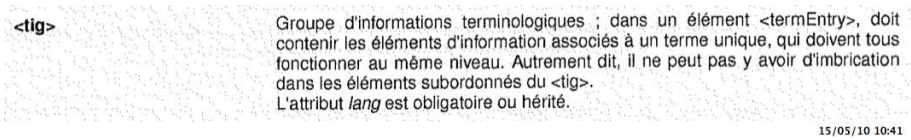
Et voici la sérialisation de ce document. On a essayé de rendre apparente l'imbrication des éléments en utilisant des retraits, mais cela n'est nullement nécessaire.

```
<loftbook>
   <titre>
      <p>L'intellect dans ma vie</p>
   </titre>
   <p cou="rouge">Il était une fois...</p>
</loftbook>
```

Et voici un exemple de document XML conforme à la norme ISO 12200 (MARTIF : stockage et échange de données terminologiques [13]) :

```
<martif type="MSC" lang="en">
   <text>
      <body>
         <termEntry id="ID67">
            <descrip type="subjectField">manufacturing</descrip>
            <descrip type="definition">A value between 0 and 1
            used in ... </descrip>
            <langSet lang="en">
               <tig>
                  <term>alpha smoothing factor</term>
                  <termNote type="termType">fullForm </termNote>
               </tig>
            </langSet>
            <langSet lang="fr">
               <tig>
                  <term>facteur lissant alpha</term>
               </tig>
            </langSet>
         </termEntry>
      </body>
   </text>
</martif>
```

C'est donc cette norme qui définit la sémantique de chaque balise. Exemple :



2.3 Couche 2 : Espaces de nommage

Devant le document XML donné en exemple ci-dessus, on peut se poser les questions suivantes :

1. Comment savoir qu'il s'agit bel et bien de cette norme ?
2. Comment savoir si la structure requise par cette norme est respectée ?
3. Et si on mélangeait des balises provenant de différentes normes ?

Les questions (1) et (3) se refèrent à la sémantique des balises, la question (2) à la structure de l'arbre du document XML. Pour répondre à (1) et (3) on se sert des *espaces de nommage* (couche 2), pour (2) on se sert des *schémas XML* (couche 3, cf. section 2.4).

Un *espace de nommage* [4] est une chaîne de caractères qui définit de manière non-ambiguë une norme ou simplement la sémantique d'une ou plusieurs balises.

C'est sans doute un choix discutable, mais pour écrire les espaces de nommage, on se sert d'URLs (alors que les espaces de nommage ne sont pas des véritables ressources du Web, mais simplement des chaînes de caractères, jouant le rôle de « signature »).

En ce qui concerne la sérialisation d'un document, pour attacher un espace de nommage à une balise d'élément ou d'attribut, on se sert d'un *préfixe* et du caractère `:` pour séparer le préfixe du nom de balise. Pour définir les préfixes on dispose du (méta-)attribut spécial `xmlns`.

Exemple :

```
<page xmlns="http://omega.enstb.org/exemple_xml"
      xmlns:isbn="http://loc.gov/books">
<p>Voici un numéro ISBN :
<isbn:number>978-2-84177-273-5</isbn:number></p>
</page>
```

Ici on déclare d'abord un espace de nommage *ad hoc*, sans préfixe : il est utilisé pour les balises `page` et `p`. Ensuite, on déclare un espace de nommage de préfixe `isbn`, utilisé pour la balise `number`.

C'est en attachant des espaces de nommage différents à différentes balises, que l'on arrive à mélanger plusieurs normes dans un même document (ex. : HTML et SVG, MARTIF et MML, etc.).

L'exemple ci-dessous est un document XML de la norme MathML [24], décrivant la formule mathématique $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Deux espaces de nommage sont utilisés : celui de XHTML (sans préfixe) et celui de MathML (avec préfixe `m`), nous avons affiché sur fond jaune la partie du document qui relève de l'espace de nommage de MathML.

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:m="http://www.w3.org/1998/Math/MathML" xml:lang="en">
  <head>
    <title>A Math Example</title>
  </head>
  <body>
    <p>The following is MathML markup:</p>
    <m:math>
      <m:mrow>
        <m:mi>x</m:mi>
        <m:mo>=</m:mo>
        <m:mfrac>
          <m:mrow>
            <m:mrow><m:mo>-</m:mo><m:mi>b</m:mi></m:mrow>
            <m:mo>±</m:mo>
            <m:msqrt>
              <m:mrow><m:msup><m:mi>b</m:mi><m:mn>2</m:mn></m:msup>
                <m:mo>-</m:mo>
                <m:mrow>
                  <m:mn>4</m:mn>
                  <m:mo>&InvisibleTimes;</m:mo><m:mi>a</m:mi>
                  <m:mo>&InvisibleTimes;</m:mo><m:mi>c</m:mi>
                </m:mrow></m:mrow>
            </m:msqrt>
          </m:mrow>
          <m:mrow><m:mn>2</m:mn><m:mo>&InvisibleTimes;</m:mo><m:mi>a</m:mi></m:mrow>
        </m:mfrac>
      </m:mrow>
    </m:math>
  </body>
</html>
```

2.4 Couche 3 : Schémas XML

Si la couche 1 est celle des données (textuelles) et la couche 2 celle de leur structuration (sous forme de document XML), on peut dire que la couche 3 est celle de la « structuration de la structure ».

On se pose la question : comment définir une structure de document et vérifier qu'un document est conforme à celle-ci ? Cela revient à modéliser une famille de documents, en se basant

sur des contraintes structurelles. La modélisation consiste en un typage des contenus textuels et une grammaire formelle des imbrications autorisées d'éléments.

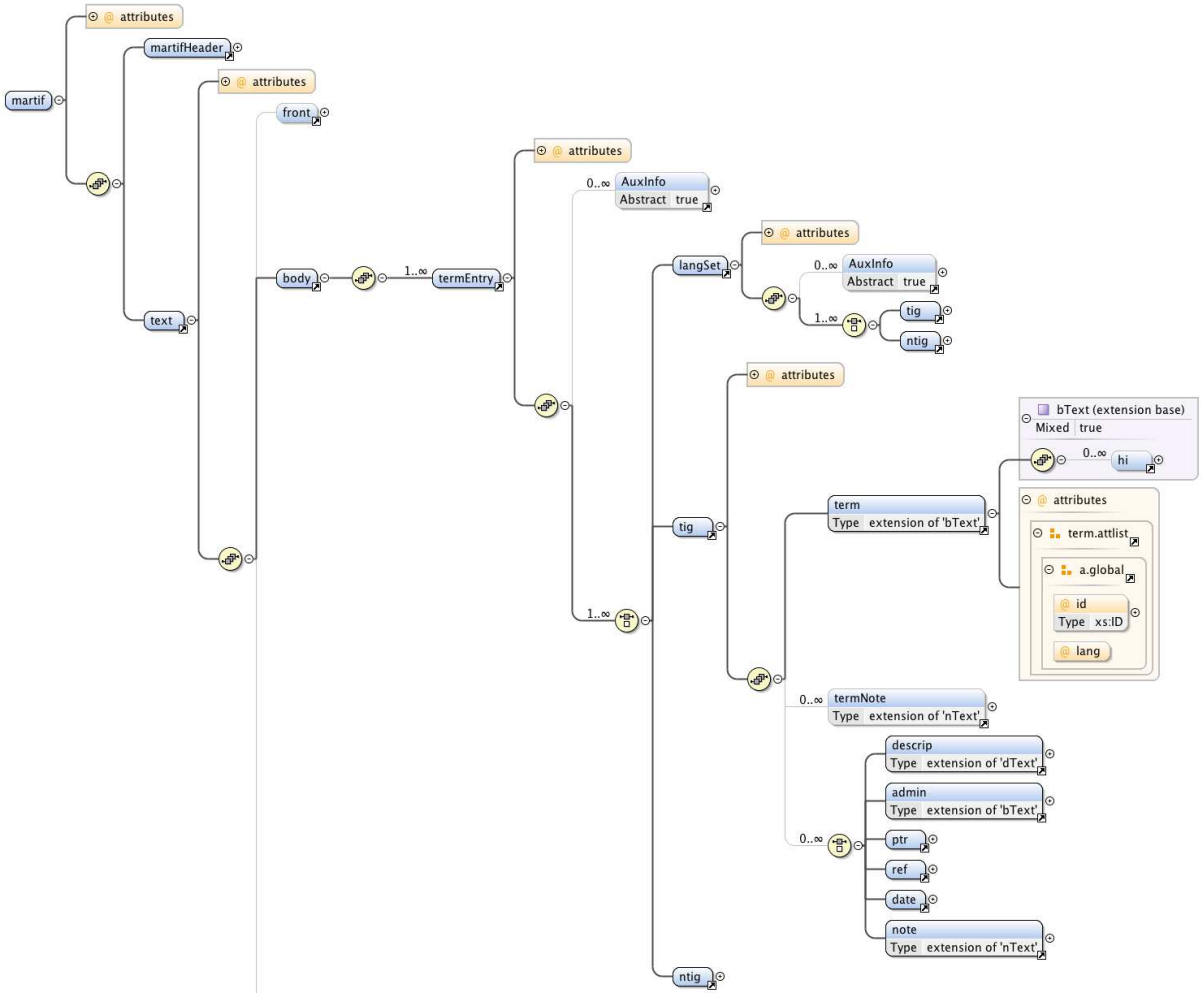
Une telle grammaire formelle s'appelle un *schéma* [25] et l'opération qui consiste à vérifier que le document est un mot du langage décrit par la grammaire, est appelée *validation*.

Un même document peut être valide vis-à-vis de plusieurs schémas (plus ou moins strictes), et, dans la plupart des cas, un même schéma peut servir à valider une infinité de documents différents.

Être valide est une garantie de bon déroulement des traitements appliqués aux données, du moment que ces traitements se limitent au niveau syntaxique.

Il y a du sens attaché aux balises (à travers les espaces de nommage), et il y a un sens implicite découlant de la structure arborescente du document. Mais c'est à l'humain de se servir de ceux-ci pour faire des inférences.

Voici, en guise d'exemple, le schéma de la norme MARTIF (cf. exemple de document XML, p. 252).



Le diagramme se lit de gauche à droite : chaque trait dénote une imbrication, dans certains cas celle-ci est accompagnée d'un quantificateur (par exemple : `body` peut contenir entre un et un nombre illimité (noté par ∞) de sous-éléments `termEntry`).

2.5 Couche 3 : XML Query / XSLT

Puisque structure il y a dans les documents XML, il est tout à fait naturel que l'on s'organise pour profiter de celle-ci en extrayant de l'information, ou pour sélectionner judicieusement certaines parties du document, ou même pour transformer le document et l'adapter à d'autres applications.

Ainsi, de la même manière que SQL pour les bases de données, XQuery [23] permet d'extraire des données d'un document XML, en tenant compte de sa structure. Une requête écrite dans le langage XQuery, appliquée à un document, retourne un ensemble de nœuds (textuels et/ou structurels).

Si les schémas permettent de spécifier et de valider une structure, XML Query (et le langage de programmation afférent XSLT [22]) permet de transformer la structure des documents pour l'adapter aux besoins : ainsi, les mêmes données peuvent, par exemple, être structurées en XHTML pour être lues sur un navigateur Web ou en RSS pour être transmises aux clients RSS.

De même, en tenant compte de la valeur de l'attribut `xml:lang`, la structure obtenue après une transformation XSLT peut être rendue dépendante de la langue de chaque nœud.

On constate alors que l'ensemble « document XML + règles de transformation » devient ainsi un « méta-document » qui va réagir selon le contexte culturel/linguistique. (Exemple-type : les sites Web multi(lingues | culturels) dynamiques.)

Notons néanmoins que ces transformations servent à une meilleure présentation ou à un meilleur accès aux données, mais se limitent, néanmoins, au niveau de la structure.

2.6 Couche 4 : Les triplets RDF

Entre les couches 3 et 4, on assiste à un saut qualitatif important (les médias parlent de passage au Web 3). Les couches 4–9 traitent du sens. On peut se poser la question de *ce qu'est le sens pour le Web sémantique*.

À la base, on trouve les *triplets RDF* :

$$\text{sujet} \xrightarrow{\text{prédicat}} \text{objet}$$

ou, en utilisant la notation de la logique de premier ordre :

$$\exists\,\text{sujet}\,\exists\,\text{objet}\;\text{prédicat}(\text{sujet},\text{objet}).$$

Pour définir les notions de « sujet », « prédicat » et « objet », on revient à la couche 1 et on se sert :

1. soit d'URIs pour identifier ces « ressources » (qui ne le sont pas forcément, puisqu'on peut aussi utiliser des espaces de nommage),
2. soit de chaînes de caractères Unicode (les *litéraux*) ;

Voici un exemple : nous allons exprimer le fait que l'auteur de cet article connaît un individu appelé José-Manuel Abreu. Dans ce triplet

« Yannis Haralambous » (sujet) « connaît » (prédicat) « José-Manuel Abreu » (objet),

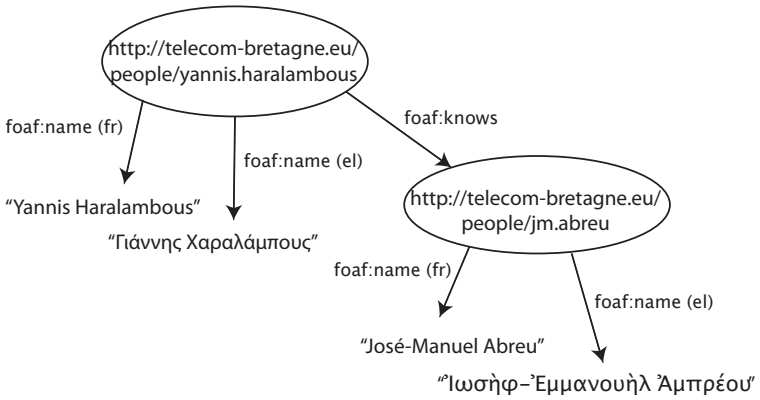
il nous faut un cadre qui définisse ce qu'est une personne, et ce qu'est l'action de « connaître ». Ce cadre s'appelle un *vocabulaire*, et il en existe un grand nombre, adaptés à diverses situations.

Nous allons ici employer le vocabulaire FOAF (acronyme de *Friend of a friend* [5]). On va donc utiliser des URIs pour les deux personnes qui apparaissent dans ce triplet RDF (même si les personnes ne sont aucunement des ressources du Web). Comment choisir ces URI ? Puisque les deux personnes en question sont des membres de l'institution connue sous le nom de « Télécom Bretagne », institution qui possède un nom de domaine Web (`telecom-bretagne.eu`), on peut se servir de celui-ci. Et puisque ces personnes ont reçu de leur institution des identifiants informatiques uniques (`yannis.haralambous` et `jm.abreu`), se servir de celles-ci permet justement d'éviter les ambiguïtés.

Il faut bien comprendre que le sens n'est pas donné par l'URI elle-même. Celle-ci ne sert que de point d'attache aux différentes informations que l'on va ajouter. Dans l'exemple ci-dessous, l'unique information donnée est le nom (dans deux langues différentes), mais on peut imaginer d'autres bribes d'information comme le numéro INSEE, ou la date de naissance ou l'ADN. C'est l'ensemble de ces informations qui donne un sens à l'URI.

En ce qui concerne le sens de l'acte de « connaître » (et pour le distinguer, par exemple, du sens biblique du verbe « connaître » qui serait tout à fait inapproprié dans ce contexte), on se sert du vocabulaire FOAF. En effet, cette norme décrit, dans ses spécifications, ce qu'elle entend par « X connaît Y ».

Voici donc un diagramme où chaque flèche est un prédicat, chaque « patate » une URI et chaque chaîne de caractères délimitée par des guillemets informatiques, un litéral :



Dans ce diagramme nous avons mis entre parenthèses la valeur de l'attribut `xml:lang`, ainsi les noms de ces deux personnes sont écrits en français et en grec. Si on avait voulu écrire ces noms selon la tradition espagnole, cela aurait donné « Yannis Haralambous Rysmanowski » et « José-Manuel Abreu Garcia », puisque le noms de jeune fille des mères des ces deux personnes sont, respectivement, Rysmanowski et Garcia.

Voici le code XML de cet ensemble de triplets RDF :

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rdf:Description rdf:about="http://telecom-bretagne.eu/people/yannis.haralambous">
    <foaf:knows>
      <rdf:Description rdf:about="http://telecom-bretagne.eu/people/jm.abreu">
        <foaf:name xml:lang="fr">José-Manuel Abreu</foaf:name>
        <foaf:name xml:lang="en">Ἰωσὴφ-Ἐμμανουὴλ Ἀμπρέου</foaf:name>
      </rdf:Description>
    </foaf:knows>
    <foaf:name xml:lang="fr">Yannis Haralambous</foaf:name>
    <foaf:name xml:lang="en">Γιάννης Χαραλάμπους</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

On constate, dans cet extrait de code, que les éléments `RDF` et `Description` utilisent l'espace de nommage de la norme RDF, alors que les éléments `knows` et `name` utilisent celui de FOAF. Autre point intéressant : pour montrer la relation sujet-prédicat-objet on utilise deux fois l'élément `Description` : la première fois (l'objet), il contient l'élément `knows`, la deuxième fois (le sujet), il est contenu dans ce dernier. C'est donc l'imbrication (syntaxique) des éléments qui attribue aux éléments leur rôle sémantique.

2.7 Couche 6 : SPARQL

Nous allons sauter une couche pour parler de SPARQL [21] : un langage qui permet de former des requêtes sur des triplets RDF. Il s'agit donc de formuler des méta-triplets RDF avec des inconnues, dont le moteur SPARQL retourne les valeurs, après avoir parcouru le réseau sémantique (dont les nœuds sont les sujets/objets et les flèches, les prédicats).

Voici un exemple, on demande les noms des amis de José-Manuel Abreu.

```
PREFIX foaf:<http://rdf.freebase.com/ns/>
PREFIX tb:<http://telecom-bretagne.eu/people/>
SELECT ?friend
WHERE {
tb:jm.abreu foaf:knows ?who .
?who foaf:name ?friend .
}
```

Cette requête, appliquée aux triplets RDF de la section précédente, aurait donné comme seul résultat le litéral `Yannis Haralambous`. Remarquons qu'il y a deux méta-triplets :

```
tb:jm.abreu foaf:knows ?who .
```

où la variable `?who` joue le rôle d'objet du prédicat « connaître » (au sens du vocabulaire FOAF) et

```
?who foaf:name ?friend
```

où la variable `?friend` est l'objet du prédicat « s'appeler » (encore une fois dans le cadre de FOAF). Si on n'avait utilisé que le premier triplet on aurait eu, en guise de réponse, l'URI de la personne. Le deuxième triplet nous permet d'avoir le nom qui correspond à l'URI de la personne.

2.8 Couche 5 : Les ontologies

Une *ontologie* est une représentation formelle d'un domaine spécifique [1] de connaissances à travers un ensemble de concepts et de relations entre concepts.

Il y a différents langages pour décrire les ontologies, avec plus ou moins de fonctionnalités. Un langage assez répandu est OWL (Web Ontology Language).

Dans OWL on a des *classes d'objets* que l'on peut *instancier*. La classe la plus haute est `owl:Thing`.

1. Il existe des ontologies qui s'intéressent à des concepts beaucoup plus généraux et qui s'appliquent à tous les domaines de la vie humaine. Ces ontologies, beaucoup plus proches du domaine de la philosophie, s'appellent *ontologies formelles* (un exemple : BFO [8]).

On peut définir des *sous-classes*, qui héritent les propriétés des classes mères.

On dispose de *propriétés* entre des instances et d'autres instances (`ObjectProperty`), ou entre des instances et des litéraux (`DatatypeProperty`).

Les propriétés ont un *domaine* (`rdfs:domain`) et une *cible* (`rdfs:range`). Elles peuvent, elles aussi, avoir des propriétés. Elles peuvent être

– *transitives* : $P(x,y) \wedge P(y,z) \Rightarrow P(x,z)$,
– *symétriques* : $P(x,y) \Leftrightarrow P(y,x)$,
– *fonctionnelles* : $P(x,y) \wedge P(x,z) \Rightarrow y = z$,
– *inversement fonctionnelles* : $P(y,x) \wedge P(z,x) \Rightarrow y = z$,
– *inverses* entre elles : $P_1(x,y) \Leftrightarrow P_2(y,x)$.

Il est possible de poser des restrictions sur les propriétés :

– `allValuesFrom` : toutes les valeurs (cibles) de la propriété sont d'un certain type,
– `someValuesFrom` : au moins une valeur (cible) de la propriété est d'un certain type,
– des restrictions de cardinalité ;

On peut également définir des équivalences entre classes ou entre propriétés, ainsi qu'indiquer l'identité (`sameAs`) ou la différence (`differentFrom`) entre instances.

Dans une version plus puissante d'OWL appelée OWL DL (DL comme *description logic*) on peut, en plus :

– définir, par intension, des *classes complexes* qui sont : des *intersections*, *unions* ou *complémentaires* d'ensembles de classes ;
– définir, par extension, des *classes énumérées*, par énumération conjonctive ou disjonctive de leurs instances.

Dans une version encore plus puissante d'OWL appelée OWL Full, on a diverses propriétés supplémentaires, comme par exemple le fait qu'une classe peut être l'instance d'une autre classe, la possibilité de spécifier des identités entre classes ou entre propriétés, etc.

2.8.1 Exemple d'ontologie sur les appareils photographiques

Voici, en guise d'exemple, une mini-ontologie (à but éducatif) sur les appareils photographiques.

On commence par les espaces de nommage utilisés et l'identité de l'auteur :

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns="http://www.xfront.com/owl/ontologies/camera/#"
         xmlns:camera="http://www.xfront.com/owl/ontologies/camera/#"
         xml:base="http://www.xfront.com/owl/ontologies/camera/">

    <owl:Ontology rdf:about="">
        <rdfs:comment>
        Camera OWL Ontology
      Author: Roger L. Costello
        </rdfs:comment>
    </owl:Ontology>
```

Ensuite, on exprime le fait qu'il existe une chose qui s'appelle l'« argent », et que cette chose a une propriété qui est la « devise » dont la cible est une chaîne de caractères.

```
<owl:Class rdf:ID="Money">
     <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="currency">
     <rdfs:domain rdf:resource="#Money"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

Il existe également une chose appelée « étendue », et cette chose a trois propriétés : min, max (nombres flottants), unité (chaîne).

```
<owl:Class rdf:ID="Range">
     <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<owl:DatatypeProperty rdf:ID="min">
     <rdfs:domain rdf:resource="#Range"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="max">
     <rdfs:domain rdf:resource="#Range"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="units">
     <rdfs:domain rdf:resource="#Range"/>
     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

Il existe des choses qui s'appellent « fenêtre » et « viseur ». On a deux instances de fenêtres : `ThroughTheLens` et `WindowOnTopOfCamera` (qui correspondent à deux types d'appareil photographique : les reflex de type Nikon F et les non-reflex de type Leica). On exprime également le fait qu'un viseur est une fenêtre de l'un de ces types.

```
<owl:Class rdf:ID="Window">
     <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<camera:Window rdf:ID="ThroughTheLens"/>
<camera:Window rdf:ID="WindowOnTopOfCamera"/>

<owl:Class rdf:ID="Viewer">
    <owl:oneOf rdf:parseType="Collection">
          <camera:Window rdf:about="#ThroughTheLens"/>
          <camera:Window rdf:about="#WindowOnTopOfCamera"/>
     </owl:oneOf>
</owl:Class>
```

Il existe une chose qui s'appelle « objet achetable », elle a une propriété appelée « coût », dont la cible est de l'argent.

```
<owl:Class rdf:ID="PurchaseableItem">
     <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="cost">
     <rdfs:domain rdf:resource="#PurchaseableItem"/>
     <rdfs:range rdf:resource="#Money"/>
</owl:ObjectProperty>
```

Il existe une chose qui s'appelle le « boîtier », qui a une propriété appelée « temps d'exposition ». Il existe une autre chose appelée « boîtier à temps d'exposition non ajustable » qui est un boîtier dont la cible de la propriété « temps d'exposition » a une cardinalité égale à 1, et donc n'a qu'une seule valeur.

```
<owl:Class rdf:ID="Body">
     <rdfs:subClassOf rdf:resource="#PurchaseableItem"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="shutter-speed">
     <rdfs:domain rdf:resource="#Body"/>
     <rdfs:range rdf:resource="#Range"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="BodyWithNonAdjustableShutterSpeed">
     <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Body"/>
          <owl:Restriction>
                <owl:onProperty rdf:resource="#shutter-speed"/>
                <owl:cardinality>1</owl:cardinality>
          </owl:Restriction>
     </owl:intersectionOf>
</owl:Class>
```

Nous allons arrêter ici la description de cet exemple. Le lecteur intéressé trouvera la totalité de l'exemple dans [6].

2.8.2 Considérations générales

On constate qu'un concept est défini à travers les relations qu'il détient avec les concepts voisins du même domaine. Ceux-ci deviennent donc ses définisseurs, dans un réseau sémantique où chaque concept participe à la définition des concepts environnants.

Un outil de TAL va parcourir ce réseau pour désambiguer, traduire, etc.

Qu'en est-il du multiculturalisme ? On peut toujours utiliser l'attribut `xml:lang` pour ventiler les éléments XML d'une ontologie selon les langues et les pays, mais OWL ne prévoit pas la re-définition d'un concept donné, selon la langue.

2.9 Couches 6–9 : Règles, logique, preuve, confiance

Le langage OWL est incapable de coder des règles du type *si <telle chose> alors <telle chose>*. Le codage de telles règles fait l'objet de langages spécifiques, en cours d'élaboration (RIF, SWRL). Exemple :

$$\text{aParent}(x_1, x_2) \wedge \text{aFrère}(x_2, x_3) \Rightarrow \text{aOncle}(x_1, x_3).$$

Dans cet exemple on fournit la règle qui dit que si x_1 et x_2 sont liés par le prédicat « x_2 est le parent de x_1 », et x_2 et x_3 sont liés par le prédicat « x_3 est le frère de x_2 », alors, automatiquement, x_1 et x_3 sont liés par le prédicat « x_3 est l'oncle de x_1 ». Cette information n'est pas explicitement contenue dans l'ensemble des triplets « être parent de » et « être le frère de » et ne peut donc pas être obtenue par une requête SPARQL. L'utilisation d'une règle est nécessaire pour que cette inférence soit possible.

Arrivé à ce stade, on pourra utiliser différentes logiques (par exemple, la logique du premier ordre), ainsi que les méthodes de preuve *ad hoc*, pour garantir algorithmiquement la cohérence et intelligibilité des données. Exemple (*modus ponens*) :

$$\forall x\,(P(x) \Rightarrow Q(x)) \wedge P(y) \implies Q(y).$$

Autre exemple (double négation) :

$$P(x) \implies \neg\neg P(x).$$

Enfin, cérise sur le gâteau (sémantique), si l'on dispose de données sémantiques formellement prouvées, et si l'on communique à travers des canaux cryptés et en servant de signatures numériques pour authentifier les ressources, alors la confiance régnera, et c'est cela le sens de la neuvième couche, appelée, tout simplement, « Confiance ».

Références

[1] Tim Berners-Lee. `http://www.w3.org/2000/Talks/1206-xml2k-tbl/`.

[2] Tim Berners-Lee et al. Uniform Resource Identifier (URI) : Generic Syntax. RFC 3986. `http://tools.ietf.org/html/rfc3986`.

[3] Tom Bishop and Richard Cook. Character description language cdl : The set of basic CJK unified stroke types. Technical report, Wenlin, 2004.

[4] Tim Bray et al. Namespaces in XML 1.0. `http://www.w3.org/TR/2006/REC-xml-names-20060816/`.

[5] Dan Brickley and Libby Miller. The Friend of a Friend (FOAF) project. `http://www.foaf-project.org/`.

[6] Roger L. Costello. OWL Example. `http://jmvidal.cse.sc.edu/talks/xmlrdfdaml/owlexample.html`.

[7] Loïc Depecker. *Entre signe et concept*. Presses Sorbonne Nouvelle, 2002.

[8] Pierre Grenon. BFO in a Nutshell : A Bi-categorial Axiomatization of BFO and Comparison with DOLCE. Technical report, Université de Leipzig, 2003. IFOMIS Report 06/2003 `http://www.ifomis.org/Research/IFOMISReports/IFOMIS%20Report%2006_2003.%pdf`.

[9] Peter Handke. *Die Angst des Tormanns beim Elfmeter*. Suhrkamp, 1998.

[10] Yannis Haralambous. Unicode et typographie : un amour impossible. *Document Numérique*, 6(3-4) :105–137, 2002.

[11] Yannis Haralambous. *Fontes & codages*. O'Reilly France, 2004.

[12] Yannis Haralambous. New hyphenation techniques in Ω_2. *TUGboat*, 27(1) :98–103, 2006.

[13] ISO. Applications informatiques en terminologie – Format de transfert de données terminologiques exploitables par la machine (MARTIF) – Transfert négocié. ISO 12200 : 1999.

[14] ISO. Traitement de l'information – Systèmes bureautiques – Langage normalisé de balisage généralisé (SGML). ISO 8879 : 1986.

[15] Unicode Consortium. Technical Standard 10. Unicode Collation Algorithm. `http://www.unicode.org/reports/tr10/`.

[16] Unicode Consortium. The Unicode Encoding. `http://www.unicode.org`.

[17] Unicode Consortium. Unicode Standard Annex 14. Unicode Line Breaking Algorithm. `http://www.unicode.org/reports/tr14/`.

[18] Unicode Consortium. Unicode Standard Annex 29. Unicode Text Segmentation. `http://www.unicode.org/reports/tr29/`.

[19] Unicode Consortium. Unicode Standard Annex 9. Unicode Bidirectional Algorithm. `http://www.unicode.org/reports/tr9/`.

[20] W3C. Extensible Markup Language (XML). `http://www.w3.org/XML/`.

[21] W3C. SPARQL Working Group Wiki. `http://www.w3.org/2009/sparql/wiki/Main_Page`.

[22] W3C. The Extensible Stylesheet Language Family (XSL). `http://www.w3.org/Style/XSL/`.

[23] W3C. W3C XML Query (XQuery). `http://www.w3.org/XML/Query/`.

[24] W3C. X3C Math Home. `http://www.w3.org/Math/`.

[25] W3C. XML Schema. `http://www.w3.org/XML/Schema`.