Rapport de stage de DEA I^3
– stage de fin d'études à Supélec

Étude de l'intégration d'informations symboliques et analogiques dans le cadre de l'aide au déplacement de personnes non voyantes

Étudiant : Christophe Jacquet
Responsables : Yacine Bellik, Jean-Paul Sansonnet, Yolaine Bourda

septembre 2003

Table des matières

Remerciements

Je tiens à remercier les personnes suivantes :

- **Yacine Bellik**, pour son suivi tout au long du stage, sa compétence et sa disponibilité ;
- **Jean-Paul Sansonnet**, pour ses conseils avisés ;
- **René Farcy**, pour sa maîtrise des aspects technologiques ;
- **Yolaine Bourda**, pour la liaison avec Supélec ;
- et toute l'équipe du LIMSI, permanents et stagiaires, pour leur bonne humeur et leur accueil chaleureux.

Introduction

Depuis quelques années, le LIMSI[1] et le Laboratoire Aimé Cotton développent un dispositif électronique d'aide au déplacement des personnes non voyantes, surnommé Télétact [Farcy 2002b, Farcy 2002a] (voir fig. 1). Ce dispositif, qui s'ajoute à la traditionnelle canne blanche, permet de mesurer la distance entre l'utilisateur et les obstacles à l'aide d'un télémètre à laser. Ces distances sont ensuite communiquées à la personne non voyante sous forme de vibrations tactiles ou de signaux sonores de fréquence variable.

Ce système est actuellement utilisé avec succès par quelques aveugles, dont il a permis d'augmenter l'autonomie. Cependant, il présente un inconvénient : il est incapable de nommer les objets désignés. Ainsi, si l'utilisateur s'égare au cours de son déplacement, l'appareil ne lui est d'aucun secours pour se re-stituer dans l'espace et retrouver des points de repère. L'utilisateur en est alors réduit à devoir compter sur la bonne volonté des passants.

C'est pourquoi il faudrait doter les systèmes Télétact de la capacité à indiquer à leur utilisateur leur localisation, et à donner des informations sur les objets pointés. Cet objectif nécessite la mise au point de plusieurs fonctionnalités. Entre autre, il faut :

– que le système soit capable de connaître en permanence ses propres coordonnées. Ceci sera réalisé par combinaison de données issues d'un récepteur GPS, d'une boussole trois axes, d'un accéléromètre trois axes, et d'un inclinomètre. Les données en provenance de tous ces capteurs seront combinées de façon à ce que le système puisse disposer en permanence des coordonnées géographiques de l'utilisateur avec une précision suffisante ;

[1] Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur.

FIG. 1 – Le dispositif Télétact actuel

– que soient modélisées au sein de l'appareil des informations associées aux objets rencontrés et aux lieux visités, de manière à pouvoir les restituer au moment opportun. Pour cela, il faut tout d'abord disposer d'un formalisme qui permette de représenter l'architecture des lieux parcourus. De plus, il faut être capable de lier des données sémantiques à ces objets purement topologiques.

Dans ce stage, nous nous sommes tout d'abord attachés à la définition d'un formalisme de représentation de la structure des bâtiments visités par les utilisateurs du Télétact, avant de nous pencher sur le problème de l'association de données sémantiques.

Première partie

Modélisation de la structure

Chapitre 1

Étude de l'existant

1.1 Langages de description 3D

De nombreuses applications informatiques font appel à la représentation de scènes tridimensionnelles. Entre autres, on peut citer toute la gamme des modélisations effectuées en CAO[1], ou bien les applications bien connues de réalité virtuelle.

Si l'on veut permettre une certaine interopérabilité entre différentes applications, éventuellement issues de constructeurs différents, il est nécessaire de disposer de formats de description des scènes relativement reconnus et normalisés.

Dans ce chapitre, nous allons dresser un rapide bilan de ce qui a été proposé jusqu'à aujourd'hui.

1.1.1 Formats liés à la CAO

En général, les applications de CAO utilisent des formats propriétaires, qui recouvrent exactement les possibilités qu'elles offrent. De plus, la marché de ces logiciels étant extrêmement convoité par les fabricants, ceux-ci cultivent ouvertement leur incompatibilité avec les concurrents, de façon à se créer un petit monopole...

Cependant, les utilisateurs de ces logiciels doivent parfois avoir recours à des environnements logiciels hétérogènes, d'où la nécessité de disposer de formats standards, ou à défaut, assez largement reconnus. Parmi ceux-ci, on peut citer :

STEP (Standard for the Exchange of Product model data) : C'est un standard émergent, mis au point au niveau international, mais qui n'est pas encore communément accepté. Il permet de modéliser à la fois les caractéristiques et la forme géométrique des produits. Ainsi, il permet de modéliser les tolérances et les caractéristiques électriques.

IGES (Initial Graphics Exchange Specification) : ce format a été développé par Boeing, General Electric, et le National Bureau of Standards américain. Il permet de représenter aussi bien des formes simples comme des lignes, des arcs, etc. que des solides. La version 2 de ce format date de septembre 1981. Elle a été normalisée

[1] Conception Assistée par Ordinateur ; en anglais : CAD, Computer-Aided Design.

par l'ANSI[2]. Les fichiers peuvent être aussi bien des fichiers binaires que des fichiers ASCII[3].

DXF (Data Exchange Format) : C'est un format introduit par le logiciel AutoCAD, mais qui a été repris par de nombreux logiciels concurrents. Il est basé sur des fichiers ASCII.

SET (Standard d'Échange et de Transfert) : cette norme a été proposée par l'AFNOR[4] en 1985, mais, même en France, elle est moins répandue qu'IGES. Cependant, elle possède des caractéristiques comparables.

On est donc loin d'un consensus dans le monde des formats de CAO. De plus, ces formats travaillent à un niveau géométrique assez "bas" : ils permettent de travailler à partir d'objets tels que droites, points, arcs, splines, etc. Les objets complexes modélisés sont donc constitués par assemblage de ces formes géométriques simples.

1.1.2 Formats liés à la Réalité Virtuelle

Au milieu des années 1990, certaines personnes pensaient qu'après le développement du Web sur Internet à partir de 1993, les pages textuelles statiques allaient rapidement être supplantées par des environnements tridimensionnels de "réalité virtuelle", qui étaient censés apporter aux utilisateurs des modes d'interaction plus "naturels". C'est pourquoi certaines sociétés ont commencé à définir le langage VRML[5].

L'objectif avoué était de définir un langage suffisamment riche pour définir des "mondes virtuels". Les navigateurs Web seraient alors munis de "plug-ins" pour naviguer dans ces mondes.

Cependant, ce langage ne s'est jamais répandu dans le grand public, pas plus d'ailleurs que la réalité virtuelle. Mais quoi qu'il en soit, le langage VRML et ses successeurs ont été développés, et ils présentent *a priori* une piste intéressante lorsqu'il s'agit de développer une modélisation d'environnements architecturaux.

VRML

VRML est une spécification très volumineuse. Elle permet de décrire des formes tridimensionnelles.

Cependant, en raison de son poids, elle a été peu implémentée, car le cahier des charges auquel doit se conformer un logiciel "compatible VRML" est trop contraignant. De plus, il est difficile de définir ses propres extensions au langage tout en gardant la compatibilité avec le standard.

Le format VRML n'est basé ni sur XML[6] ni sur SGML[7] : il s'agit d'un format de fichiers

[2] American National Standards Institute
[3] American Standard Code for Information Interchange
[4] Agence Française de Normalisation
[4] American Standard Code for Information Interchange, format de représentation des caractères de l'alphabet latin, des chiffres romains et de quelques symboles associés.
[5] Virtual Reality Modelling Language
[6] eXtensible Markup Language
[7] Standard Generalized Markup Language

binaires.

Pour toutes ces raisons, la diffusion de VRML est restée quelque peu confidentielle, se limitant à quelques démonstrations. La version de VRML qui a été diffusée se nomme VRML97.

X3D

Le W3C[8] et le Consortium Web3D[9] ont essayé de définir un langage basé sur XML, et qui reprendrait les principes de VRML. Il en a résulté la spécification du langage X3D (eXtensible 3D).

La spécification a été modularisée, sous forme de *profils*. Ainsi, un logiciel donné n'a pas besoin d'implémenter la totalité de la spécification : il peut se contenter d'être conforme à l'un seulement de ces *profils*. De plus, X3D est plus facilement extensible que VRML.

Cependant, X3D souffre de certains défauts : notamment, il est organisé sous la forme d'une DTD[10] monolithique, et ne fait pas usage des espaces de noms.

Cette DTD existe en deux versions : une version "verbeuse" (noms de balises assez longs), et une version "compacte" (noms de balises plus courts). Les niveaux d'avancement de ces deux DTD sont inégaux, ce qui ajoute à la confusion.

Les API[11] de X3D ne sont pas encore définies. Celles de VRML97 sont toujours restées floues. Il pourrait être envisagé de reprendre le DOM[12].

En pratique, la différence entre VRML et X3D est surtout une différence de forme. Le pouvoir expressif de ces deux langages est le même : tous deux permettent de décrire des scènes à l'aide de formes géométriques simples (sphères, cylindres, cubes, courbes, etc.), sur lesquelles il est possible d'appliquer des textures. De plus, il est possible de placer des sources de lumière et des ombrages de façon à obtenir une vision réaliste.

1.1.3 Autres pistes

Il est possible d'explorer d'autres pistes dans le cadre d'un langage de description d'environnements 3D.

VML

VML[13] est une application XML qui permet le codage de graphisme vectoriels pour affichage à l'écran. VML fonctionne sur les mêmes principes que HTML[14], qui a pour but le codage de données de type texte :

[8] Word Wide Web Consortium
[9] `http://www.web3d.org`
[10] Document Type Definition
[11] Application Programming Interface
[12] Document Object Model
[13] Vector Markup Language
[14] HyperText Markup Language

– utilisation d'un langage à balises (XML ou SGML) ;
– fichiers facilement éditables à la main.

VML est prévu pour fonctionner en 2D. Il est extensible, de façon à ne pas limiter les fonctionnalités des éditeurs.

SVG

Le langage VML a été proposé au W3C en 1998 par Microsoft, qui a été jusqu'à l'implémenter dans l'une des versions de son logiciel Internet Explorer. Cependant, VML n'est jamais passé au stade de recommandation.

En effet, le W3C a définit un autre format de graphismes vectoriels en 2D : SVG[15]. SVG reprend en gros les principes de VML, tout en étant plus complet. Il se présente lui aussi sous la forme d'un dialecte XML.

L'adoption de SVG semble en bonne voie : Adobe diffuse un plug-in pour les navigateurs Web, le navigateur Mozilla dispose d'une implémentation native expérimentale de SVG, et de nombreux logiciels proposent d'ores et déjà l'importation et l'exportation de graphismes au format SVG.

1.1.4 Conclusion

Quelques formats existent donc pour décrire des scènes de façon vectorielle (que ce soit en deux ou trois dimensions). Cependant, tous ces langages de description souffrent d'une absence de véritable normalisation et de consensus.

De plus, il faut bien garder en tête qu'ils s'attachent uniquement à la représentation de scènes d'une façon la plus proche possible de la réalité. Ces formalismes s'adressent à des personnes *voyantes*, auxquelles il s'agit de restituer l'*apparence visuelle* de la réalité.

1.2 Nos besoins

Pour notre problème, l'*apparence visuelle* des éléments architecturaux revêt une importance mineure. En revanche, la représentation de leur *sémantique* constitue un besoin majeur. Ainsi, pour décrire une porte, peu nous importe d'en donner une description fidèle (couleur, forme précise de la poignée, etc.). Par contre, nous souhaitons modéliser les renseignements utiles à une personne non-voyante : tout d'abord, le simple fait que nous avons affaire à une porte, mais aussi la position de l'éventuelle poignée, le sens d'ouverture, l'éventuelle ouverture automatique, etc.

La description devra également être structurée de telle façon qu'il soit par exemple aisé de déterminer quelles pièces sont mises en relation par une porte donnée. Nous devons donc choisir un modèle qui tienne compte de cette structure, c'est-à-dire de ces relations qui existent entre éléments architecturaux.

De plus, nous devrons être capable d'associer des *annotations sémantiques* aux

[15]Scalable Vector Graphics, voir `http://www.w3.org/Graphics/SVG/`

éléments d'architecture : à une pièce, il sera utile d'associer, par exemple, son numéro, ses occupants, sa fonction, voire son planning d'utilisation.

Voyons tout d'abord ce que doit nous permettre de représenter le modèle. Au plus haut niveau, il doit permettre de positionner des bâtiments dans l'espace, puis de décrire chacun de ces bâtiments, en termes d'objets, comme des étages, des murs, des surfaces de sol, des portes et fenêtres, des pièces, etc.

En outre, nous devons être capables de représenter des éléments non physiques. Par exemple, considérons une pièce divisée en une zone fumeurs et une zone non fumeurs. Ces deux zones sont clairement séparées, et il est important de modéliser cette séparation pour les utilisateurs non voyants. Nous représentons donc la limite entre ces deux zones par un "mur virtuel", c'est-à-dire un élément non physique, mais qui a une signification précise vis-à-vis de la sémantique de l'ensemble.

Enfin, le modèle doit nous permettre, étant donnée la position d'un point dans l'espace, de déterminer dans quel objet il est inclus.

Dans un premier temps, nous ne décrirons que quelques éléments d'architecture, de façon à tester les possibilités du modèle. De nouveaux éléments seront ajoutés par la suite.

Pour commencer, nous n'allons nous préoccuper que de la représentation d'un bâtiment. Nous ne traiterons ni du positionnement de plusieurs bâtiments sur un plan, ni de la description des éventuels objets qui se trouvent dans les pièces du bâtiment.

1.3 Inadéquation des modèles existants

Nous l'avons déjà vu lors de l'étude réalisée au paragraphe 1.1, la plupart des langages 3D existants sont orientés vers la modélisation de l'aspect visuel des scènes. Des modèles qui prennent en compte la sémantique des objets architecturaux commencent néanmoins à apparaître. En particulier, des architectes sont en train de définir bcXML [van Rees 2002], pour "Building Construction XML". Cependant, d'une part ce format est assez lourd par rapport à son embarquement dans un dispositif portable, et d'autre part ses spécifications ne sont pas encore définitivement arrêtées. C'est pourquoi nous ne l'avons pas retenu pour notre application.

Les roboticiens ont eux aussi besoin de modéliser l'architecture des lieux dans lesquels se déplacent leurs machines. Ainsi, certains décrivent l'architecture des pièces visitées à l'aide du modèle spécifié en UML [Chella 2002]. Cependant, il s'agit d'initiatives assez isolées : il n'existe pas de standard communément admis.

Nous pourrions alors penser à nous tourner vers un système inspiré des bases de données spatiales [Rigaux 2002, Hadzilacos 1997]. Cependant, nos besoins sont différents de ceux exprimés dans le cahier des charges d'une base de données. En effet, cinq opérations doivent êtres offertes par un Système de Gestion de Bases de données ([Rigaux 2002], p. 4) :

1. définition de la base de données ;
2. construction de la base de données ;
3. manipulation de la base de données ;
4. requêtes sur les données ;

5. mise à jour des données.

Dans notre cas, les opérations (1) et (2) sont faites une fois pour toutes ; (3) et (5) sont rarement effectuées. La seule opération que notre appareil sera amené à faire est la (4). Cependant, comme l'on dispose de données organisées (avec une structure bien spécifique), il sera certainement plus efficace de concevoir nos propres algorithmes de requêtes optimisés pour notre structure.

De plus, il n'est pas souhaitable d'avoir une base de données centralisée à laquelle tous les utilisateurs formuleraient des requêtes. Il paraît plus approprié de décrire les plans pour des zones de taille restreinte (par exemple, un bâtiment). Ces plans pourraient alors être téléchargés en intégralité et une fois pour toutes par le Télétact au moment où il en aurait besoin, ce qui éviterait d'avoir un goulot d'étranglement au niveau des liaisons réseau.

Enfin, on le verra par la suite, les données de description d'un bâtiment sont fortement structurées. En utilisant une base de données, il n'aurait pas été possible de rendre explicite cette structure.

Chapitre 2

Notre modèle

Les raisons indiquées au chapitre 1 nous ont amenés à développer notre propre modèle pour les descriptions d'architectures. Cependant, lorsque des standards ouverts auront atteint un niveau de maturité suffisant, nous pourrons aisément réaliser des conversions à partir et à destination de notre propre formalisme, afin de pouvoir importer des descriptions existantes ou d'exporter les nôtres.

2.1 Représentation en trois couches

Nous avons défini un modèle en trois couches pour décrire les architectures (voir fig. 2.1) :

Première couche : niveau lexical. Nous appelons "éléments lexicaux" les éléments architecturaux simples – c'est-à-dire élémentaires. En pratique, ce sont des murs, des portes, des volées de marches, etc.

Deuxième couche : niveau syntaxique. Les "éléments syntaxiques" sont complexes – c'est-à-dire composés. Ils sont constitués de l'arrangement de plusieurs "éléments lexicaux". Par exemple, une pièce est définie par ses murs.

Troisième couche : niveau d'agrégation. Les "éléments syntaxiques" sont alors rassemblés en "éléments d'agrégation". Ainsi, plusieurs bureaux peuvent être assemblés en un agrégat appelé, par exemple "service des ventes".

On peut faire un parallèle entre notre terminologie et la structure des textes en langage naturel. Au niveau le plus bas, les textes sont simplement écrits avec des mots : c'est le niveau lexical. Ces mots sont alors assemblés en phrases par rapport à une syntaxe définie. À leur tour, les phrases sont agrégées en diverses unités de sens telles que des paragraphes, des listes à puces, etc.

Au sein d'une description, les concepts décrits sont assemblés par deux types de relations :

– relations d'*inclusion* au sein d'une couche donnée (en traits pleins sur la fig. 2.1). Par exemple, une pièce est incluse dans un étage, lui-même inclus dans un bâtiment ;

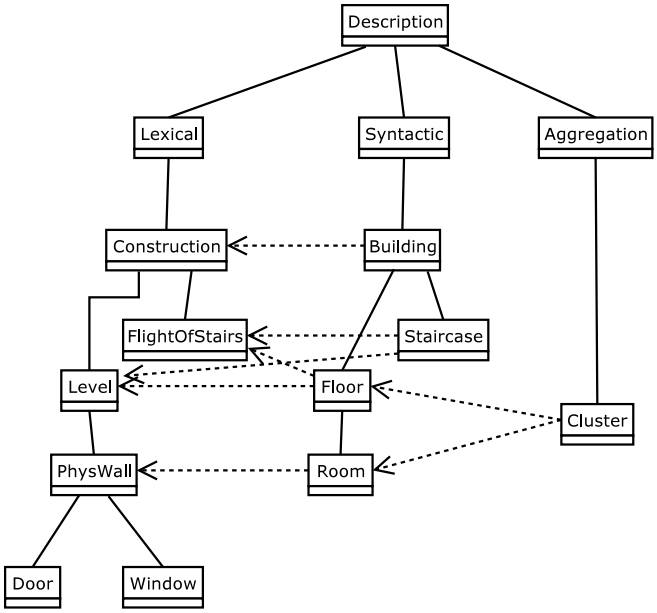


FIG. 2.1 – Constitution d'une description

– relations de *composition* entre les éléments de la couche n et ceux de la couche $n-1$ (en pointillés sur la fig. 2.1). Par exemple, une pièce est définie par la donnée de ses murs.

Les concepts mis en jeu dans notre modèle prennent place dans une hiérarchie, qui possède trois branches principales (voir fig. 2.2), correspondant à chacune des couches de notre modèle. Pour le moment, seuls quelques concepts ont été définis (ce sont ceux qui apparaissent sur la figure 2.1).

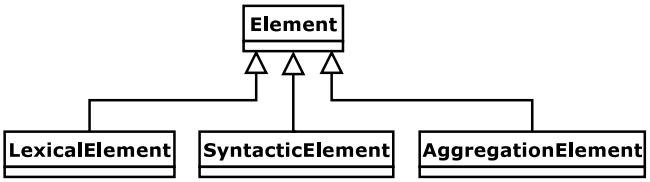


FIG. 2.2 – Sommet de la hiérarchie de classes

Éléments lexicaux. Ce sont des éléments physiques positionnés grâce à leurs coordonnées (voir fig. 2.3). Ils possèdent un repère cartésien dans lequel sont repérés leurs sous-éléments. En outre, ils peuvent être munis d'attributs qui décrivent d'autres aspects physiques.

La classe `Construction` permet de repérer le point géométrique à partir duquel sera décrit le reste du bâtiment.

La classe `FlightOfStairs` désigne une volée de marches toutes identiques. Cette classe entrera dans la composition d'un escalier, sachant qu'un escalier est formé d'une ou plusieurs volées de marches, éventuellement séparées par des paliers.

Les surfaces planes sont modélisées par la classe `Level`. De tels éléments permettent de composer des étages et des paliers.

Dans cette hiérarchie, nous distinguons deux classes abstraites. Ainsi, `Opening` modélise toute ouverture pratiquée dans un mur, et regroupe les caractéristiques

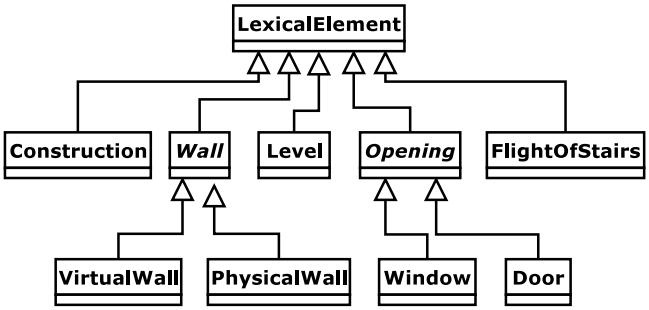


FIG. 2.3 – Éléments lexicaux

génériques communes aux classes `Window` et `Door`. De même, `Wall` représente un mur générique, sachant que les murs concrets peuvent être de deux types : `PhysicalWall` pour les murs classiques, et `VirtualWall` pour des murs "virtuels", c'est-à-dire non physiques, mais qui permettent d'introduire des limites de zones.

Éléments syntaxiques. Ils correspondent aux compositions possibles à partir des éléments lexicaux précédemment définis (voir fig. 2.4). Ce sont des concepts non pas physiques, mais abstraits, qui résultent de l'association de leurs éléments constitutifs.

Ainsi, au contraire des éléments lexicaux, les éléments syntaxiques ne possèdent pas leurs propres attributs physiques, mais un ensemble de références vers les éléments lexicaux qui les composent.

- un bâtiment (`Building`) est un édifice. Il est associé à une construction (`Construction`) ;
- un étage (`Floor`) peut être composé de plusieurs niveaux (`Level`). Ces niveaux peuvent avoir des altitudes légèrement différentes, et être reliés par des volées de marches, à condition que l'ensemble ait la signification d'un seul étage dans le contexte du bâtiment en cours de description ;
- une pièce (`Room`) est un espace entouré de murs (`Wall`). Il doit y avoir au moins trois murs pour former une pièce ;
- un escalier (`Staircase`) permet de relier deux étages (`Floor`). Il est composé de plusieurs volées de marches, et éventuellement de niveaux qui représentent des paliers.

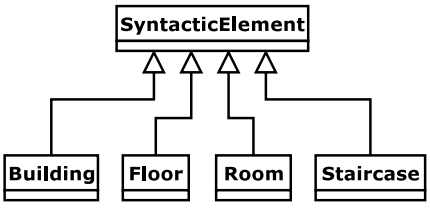


FIG. 2.4 – Éléments syntaxiques

Éléments d'agrégation. Pour l'instant, une seule classe a été définie dans cette hiérarchie (voir fig. 2.5). Il s'agit de la classe `Cluster` qui permet de modéliser des agrégats quelconques. Nous pourrons envisager d'introduire des spécialisations de cette classe : par exemple, des services d'une entreprise, des secteurs dans un musée, etc.

Ainsi, dans une entreprise, il serait possible de modéliser un "service des ventes" sous forme d'un agrégat, ledit service étant composé d'un certain nombre de pièces.

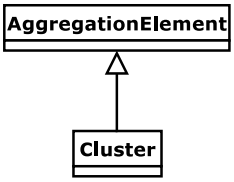


FIG. 2.5 – Éléments d'agrégation

2.2 Choix d'un formalisme

Le modèle défini pour décrire nos architectures est fortement structuré. En particulier, les relations d'inclusion dans notre modèle doivent pouvoir être représentées de manière explicite dans le formalisme choisi.

C'est pourquoi nous nous sommes tournés vers XML, qui présente entre autres les avantages suivants :

- il permet de représenter la structure de façon explicite (inclusion d'éléments) ;
- c'est un standard de fait ;
- il existe des bibliothèques de manipulation de fichiers XML pour la plupart des langages de programmation ;
- il est possible de transformer un fichier XML en un autre de façon automatique à l'aide d'une description déclarative de la transformation à effectuer (feuilles XSLT[1]).

Il faut alors déterminer le formalisme utilisé pour décrire notre application XML. Il existe au moins six méthodes pour le faire [Lee 2000], mais nous avons limité notre choix entre l'utilisation d'une DTD[2] ou d'un schéma XML, car ce sont les deux seuls formalismes communément admis et proposés par le W3C, un organisme de standardisation reconnu :

- les DTD sont simples à écrire, mais elles ne permettent ni de modéliser l'héritage entre objets, ni de spécifier finement les types de données utilisés ;
- les schémas[3] pallient ces deux défauts, et présentent un autre avantage important : un schéma XML est lui-même un fichier XML, donc il peut faire l'objet de traitements standards appliqués aux fichiers XML.

En conséquence, nous avons choisi d'utiliser les schémas XML, tout comme [van Leeuwen 2001]. Selon l'étude réalisée par [Lee 2000], nous privilégions ainsi l'expressivité des schémas, au détriment quelquefois de la simplicité des DTD.

2.3 Métamodèle

Ainsi, le schéma XML, dans le sens où il décrit la façon dont doivent être rédigées les descriptions, constitue une représentation informatique du modèle sous-jacent. Plus précisément, il constitue a priori l'*unique* représentation informatique de notre modèle.

Cet état de fait présente deux inconvénients :

- d'une part, le langage XML Schema est adapté à la description de classes de

[1] XML Stylesheets, voir `http://www.w3.org/Style/XSL/`
[2] Document Type Definition
[3] Voir `http://www.w3.org/XML/Schema`

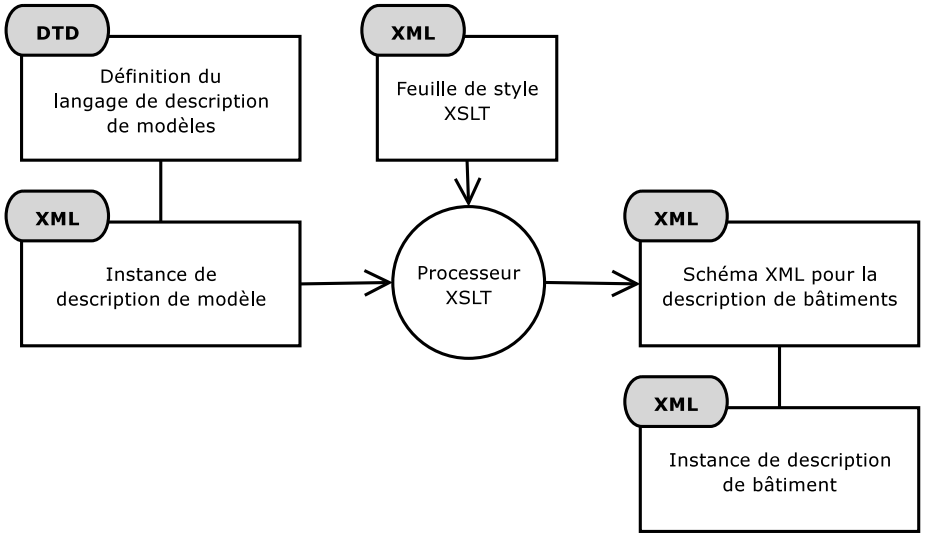


FIG. 2.6 – Architecture de notre système

documents XML, et non à la description d'un modèle conceptuel. En effet, il introduit un certain nombre de concepts qui ne se justifient que par rapport à la syntaxe XML, et qui sont donc superflus d'un point de vue théorique ;
– d'autre part, il est assez compliqué d'apporter au modèle des modifications, même mineures, si nous devons retoucher le schéma XML. En effet, certains passages du schéma sont reproduits en plusieurs exemplaires, du fait de la similarité de nos concepts. En conséquence, une modification du schéma entraîne souvent une recopie en entier d'un passage préexistant.

Comme notre modèle n'est pas encore définitif, et que nous serons amenés à le modifier assez fréquemment, nous souhaitons disposer d'un moyen de décrire notre modèle indépendamment du formalisme XML choisi pour nos descriptions.

C'est pourquoi nous avons défini un métamodèle, par rapport auquel est décrit notre modèle. Ce métamodèle est accompagné d'un langage de description de modèles, qui constitue lui aussi un dialecte XML.

De cette façon, notre modèle est décrit de façon théorique, sous la forme d'un fichier XML, ce qui nous permet de générer automatiquement le schéma XML à partir de cette description abstraite, par simple application d'une feuille de style XSLT (voir fig. 2.6).

2.4 Acquisition des descriptions

Nous envisageons d'acquérir nos descriptions de trois façons différentes :

– en les créant complètement. Pour le moment, cela revient à écrire à la main un fichier XML, mais à terme, l'aspect syntaxique pourra être caché derrière un éditeur graphique plus facile à utiliser ;
– en convertissant une description donnée issue d'un autre format. En fonction de la sémantique comprise dans cet autre format, cette conversion pourra être automatique ou semi-automatique, dans le cas où l'utilisateur doive donner des précisions supplémentaires non incluses dans le format de base ;

– en balayant les lieux à l'aide du Télétact et en nommant les objets au fur et à mesure. Cette méthode pourrait s'avérer extrêmement pratique ; l'on pourrait même envisager découvrir l'architecture d'un lieu donné au fur et à mesure, au même rythme que la personne non voyante. Cependant, il sera nécessaire d'étudier de façon précise la nature et les conditions d'une telle interaction.

2.5 Conclusion

Au début de ce chapitre, nous avons construit un modèle en trois couches pour la description de l'architecture de bâtiments.

Ce modèle est défini par rapport à un métamodèle, ce qui garantit une indépendance par rapport au formalisme de représentation choisi.

En pratique, nous définissons notre modèle dans un langage de description de modèles, qui est un dialecte XML. Grâce à l'application d'une feuille de style XSLT, nous pouvons alors générer un schéma XML par rapport auquel nous écrivons nos instances de descriptions de bâtiments.

Nous disposons ainsi d'un système complet et évolutif pour la description d'architectures.

Deuxième partie

Données sémantiques associées à la structure

Introduction

Jusqu'à présent, nous avons défini un modèle pour représenter la structure physique des bâtiments représentés. Nous qualifions cette structure de *physique*, car, même si les objets utilisés possèdent une sémantique de haut niveau, ils n'en décrivent pas moins des objets *physiques* (portes, fenêtres, pièces, etc.).

Si l'on s'en tenait là, nous obtiendrions un système capable de donner la nature des objets pointés. Par exemple, si l'utilisateur désignait une porte, le système pourrait dire "il y a une porte à trois mètres". Et éventuellement, "cette porte donne sur telle pièce".

Cela constitue déjà un net progrès par rapport au système actuel, qui n'est capable de fournir qu'une information de distance. Cependant, il est possible d'aller plus loin. Pour reprendre l'exemple précédent, il serait intéressant de pouvoir donner des informations sur la pièce : est-ce un bureau ou une salle de conférence ? Le cas échéant, à qui est attribué ce bureau, ou quel est le planning d'utilisation de cette salle de conférence ? Quelles sont les restrictions d'accès à cette pièce ?

Nous appelons *annotations sémantiques* l'ensemble de ces informations. Elles viennent s'ajouter aux descriptions *structurelles* décrites dans la partie précédente. Il faut bien voir qu'elles sont moins nécessaires que les informations de structure ; elles s'y ajoutent ; elles forment un niveau supplémentaire.

La description de structure étudiée en première partie permet de positionner des objets physiques "lexicaux" dans l'espace, et de les agréger pour former des objets physiques d'ordre supérieur.

Les annotations sémantiques vont nous permettre de relier ces objets physiques à tout un ensemble d'"objets organisationnels" présents dans le même lieu : personnes, activités, plannings, restrictions et autorisations d'accès, etc. Ainsi, chacun des objets présents dans la description de structure sera aussi présent dans le "monde sémantique", mais celui-ci sera peuplé de nombreux autres types d'objets (voir fig. 2.7).

Lors de l'interaction avec l'utilisateur, les annotations sémantiques viendront s'ajouter aux informations extraites de la structure seule du lieu ou de l'objet pointé. Le dispositif devra donc présenter à l'utilisateur une synthèse des deux types d'information.

Dans cette partie du rapport, nous allons décrire des méthodes permettant de modéliser ces annotations sémantiques. Ainsi, nous aborderons le thème général du *Web Sémantique* et des langages qui lui sont liés, en particulier RDF et ses langages d'ontologies. Enfin, nous proposerons une méthode pour la représentation des données sémantiques qui nous préoccupent présentement.

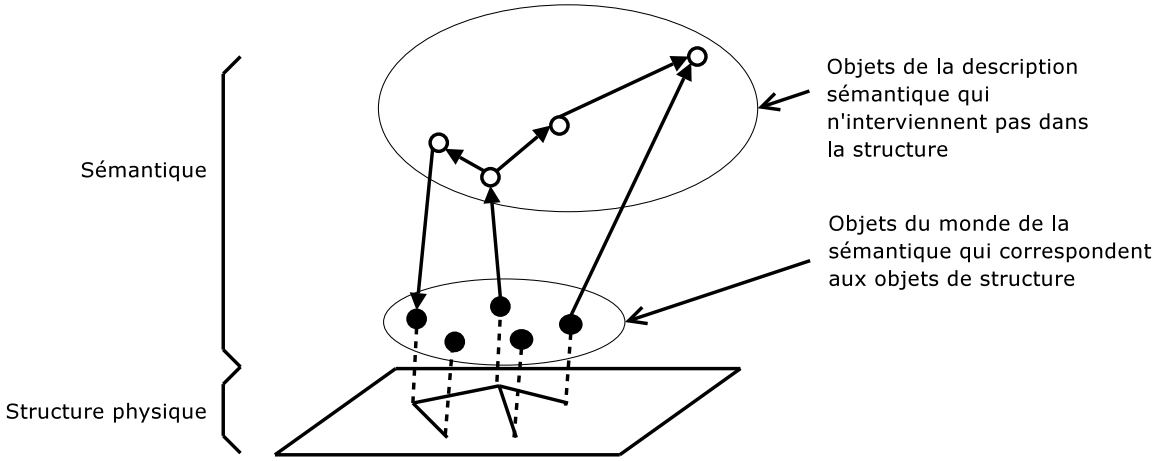


FIG. 2.7 – Liaison entre structure physique et objets sémantiques

Chapitre 3

Étude de l'existant

Depuis le début des années 1990, HTML, un dialecte de SGML, permet de décrire la structure des documents sur le Web. Apparu en 1998, XML permet de se concentrer sur la *structure* et la *sémantique* des documents, et non plus seulement sur leur formatage pour la présentation sur le Web.

Cependant, il est parfois nécessaire de travailler à un niveau de modélisation plus poussé, c'est à dire de disposer de la capacité à définir des relations diverses entre des objets. C'est ce genre de modèle que le W3C s'emploie actuellement à définir dans le cadre de ses activités liées au *Web sémantique*.

Notamment, le langage RDF permet de représenter des modèles très riches. Avant de voir précisément comment il est possible d'utiliser RDF pour notre projet, nous allons procéder à un tour d'horizon des théories et techniques liées au Web sémantique et à RDF.

3.1 Réseaux sémantiques

Les premières théories de représentation de données sémantiques s'appuient sur la notion de *réseau sémantique*. Dans un tel réseau, les sommets représentent des concepts, et ils sont mis en relation grâce à des arcs, qui modélisent les relations entre concepts.

Les types et les applications des réseaux sémantiques sont très divers : parmi eux, on trouve par exemple les réseaux de neurones à apprentissage. La notion de réseau sémantique est donc très générale.

En 1987, Peirce a introduit une notation de la logique du premier ordre sous forme de réseaux sémantiques : les *graphes existentiels* [Sowa 2002].

Par exemple, le graphe de la figure 3.1 modélise la phrase *"Jean va à Orsay en bus"*. Les boîtes représentent des *concepts*, et les flèches des *relations conceptuelles*.

On appelle ces graphes des *graphes existentiels* car les concepts sont quantifiés par le quantificateur existentiel ($\exists$). Ces graphes permettent de modéliser toute la logique du premier ordre et certaines de ses extensions, comme les logiques modales.

Les graphes existentiels peuvent être écrits sous une forme textuelle, connue sous le

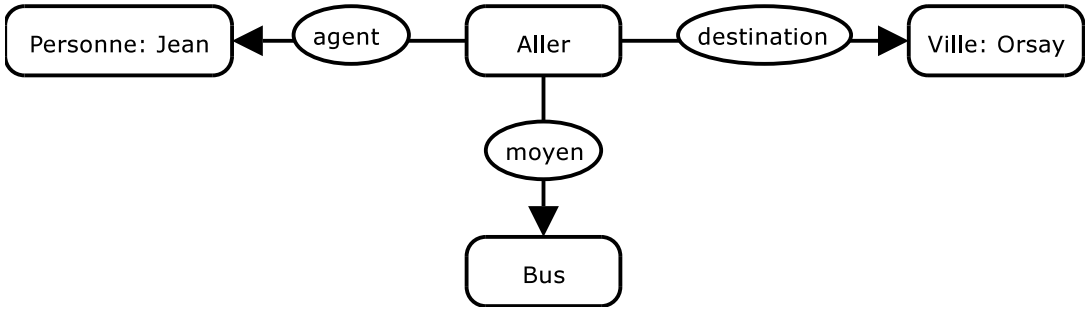


FIG. 3.1 – Exemple de graphe existentiel

nom de KIF[1]. Ainsi, le graphe de la figure 3.1 peut être écrit sous la forme :

```
(exists ((?x aller) (?y personne) (?z ville) (?w bus))
         (and (nom ?y Jean) (nom ?z Orsay) (agent ?x ?y)
              (destination ?x ?z) (moyen ?x ?w)))
```

Ceci correspond précisément à la logique du premier ordre, où cet exemple s'écrit avec une syntaxe à peine différente :

$$(\exists x)\,(\exists y)\,(\exists z)\,(\exists w)\,(\text{aller}(x) \wedge \text{personne}(y) \wedge \text{bus}(x) \wedge \\ \text{nom}(y, \text{Jean}) \wedge \text{nom}(z, \text{Orsay}) \wedge \text{agent}(x, y) \wedge \text{destination}(x, z) \wedge \text{moyen}(x, w))$$

John F. Sowa a proposé une extension des graphes existentiels, sous la forme des *graphes conceptuels* [Sowa 2001]. Sowa conserve la même expressivité que celle des graphes existentiels, mais il introduit une notation plus riche de façon à écrire les propositions de façon plus succincte. Avec ses principes d'expressivité, de simplicité et de lisibilité [Sowa 2003], il cherche à faciliter la liaison avec la langue naturelle.

3.2 Le Web sémantique

L'expression *Web sémantique* tire ses origines d'un article [Berners-Lee 2001] écrit en 2001 par Tim Berners-Lee, l'inventeur du Web en 1990.

Actuellement, beaucoup d'informations sont disponibles sur le Web, mais elles sont destinées aux êtres humains et pas aux machines car elles mélangent le fond et la forme. En conséquence, des agents autonomes (des logiciels qui essaient de rechercher et de s'échanger eux-mêmes des informations) ne peuvent pas en tirer parti de façon simple et fiable.

Le but du Web sémantique est donc de donner du sens aux informations disponibles sur le Web, un sens compréhensible par des agents autonomes. Ces agents pourront alors passer de page en page et effectuer des tâches complexes.

[1] Knowledge Interchange Format

Il ne s'agit pas de créer un *nouveau* Web, mais plutôt une *extension* du Web actuel, où les informations auraient un sens bien précis, de façon à permettre aux utilisateurs et aux machines de mieux coopérer.

Les résultats derrière le Web Sémantique existent déjà : ils sont issus des travaux en Représentation des Connaissances. Cependant, les technologies en sont à leurs balbutiements. Il est nécessaire de spécifier trois choses :

- un langage de représentation concret des données : XML est un très bon choix ;
- un modèle conceptuel pour la représentation des connaissances : le W3C est en train de spécifier RDF[2], un langage basé sur les théories évoquées au paragraphe 3.1 ;
- un moyen de partager des ontologies, c'est-à-dire du vocabulaire : plusieurs langages (basés sur RDF) ont été envisagés. On retiendra en particulier DAML+OIL et OWL. Les ontologies sont fondamentales pour le Web sémantique [Studer 2003] car elles permettent à des programmes différents de communiquer entre eux, dès le moment où ils partagent une ontologie (*i.e.* un vocabulaire commun).

Nous avons déjà évoqué XML, donc nous allons nous intéresser dans ce chapitre à une étude des langages de représentation de la sémantique.

3.3 RDF

3.3.1 Origine : métadonnées et Dublin Core

Dès la fin des années 1990, s'est fait sentir le besoin de donner des informations à propos des données du Web, c'est à dire d'associer des *métadonnées* aux pages Web. C'est pourquoi une initiative indépendante du W3C s'est donnée pour but de définir un vocabulaire de métadonnées pour la description des pages Web.

Ce projet a donné naissance au *Dublin Core*[3] [Weibel 2000]. Dans ce vocabulaire, on peut donner facilement le titre, l'auteur, ou encore le sujet d'un document. Il est également possible de définir des relations entre documents.

En pratique, le *Dublin Core* est utilisé pour associer des métadonnées aux pages Web (inclusion dans le code HTML), mais aussi pour d'autres applications qui font elles aussi appel aux notions de titre ou d'auteur, comme la description de morceaux de musique [Swartz 2002].

3.3.2 RDF

Parallèlement au Dublin Core, le W3C a lui aussi cherché à définir un format de méta-données. Cependant, au lieu de définir un vocabulaire précis adapté à un usage précis comme l'avait fait le Dublin Core, le W3C a défini un cadre général pour la définition de toutes sortes de méta-données. Il s'agit de RDF, langage de description d'informations sur le Web [Miller 1998].

[2] Resource Description Framework, voir `http://www.w3.org/RDF/`

[3] DCMI : Dublin Core Metadata Initiative, voir `http://www.dublincore.org/`

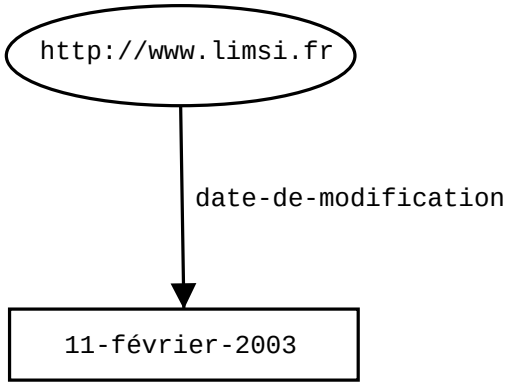


FIG. 3.2 – Exemple de triplet RDF représenté sous forme de graphe

RDF est proche est largement inspiré des graphes conceptuels[4], avec quelques différences : ainsi, les relations en RDF sont toujours binaires, alors qu'elles sont n-aires dans les graphes conceptuels.

Le modèle RDF définit trois types d'objets :

- des *ressources* : les ressources sont tous les objets *décrits* par RDF. Généralement, ces ressources peuvent être aussi bien des pages Web que tout objet ou personne du monde réel. Les ressources sont alors identifiées par leurs URI[5] ;
- des *propriétés* : une propriété est un attribut, un aspect, une caractéristique qui s'applique à une ressource. Il peut également s'agir d'une mise en relation avec une autre ressource ;
- des *valeurs* : les valeurs en question sont les valeurs particulières que prennent les propriétés.

Ces trois types d'objets peuvent être mis en relation par des *assertions*, c'est à dire des triplets (*ressource*, *propriété*, *valeur*), ou encore (*sujet*, *prédicat*, *objet*). Une description RDF est une suite d'assertions.

Voyons trois objets RDF :

- une ressource : `http://www.limsi.fr/` ;
- une propriété : `date-de-modification` ;
- une valeur : `11-février-2003`.

On peut alors construire l'assertion : `(http ://www.limsi.fr/, date-de-modification, 11-février-2003)`, qui signifie *"la date de modification de la page Web du* LIMSI *est le 11 février 2003"*.

Il est possible de représenter les descriptions RDF par des graphes. Par exemple, on voit sur la figure 3.2 le graphe correspondant à l'assertion précédente.

Dans un graphe RDF, on représente par des ellipses les ressources nommées (i.e. les objets qui possèdent des URI), et par des rectangles les littéraux (i.e. les constantes, qui ne possèdent pas d'URI).

À la base, RDF ne fait que définir ce modèle de représentation. La syntaxe n'est pas

[4]Voir l'analyse comparative de Tim Berners-Lee, `http://www.w3.org/DesignIssues/CG.html`

[5]Uniform Resource Identifier, identifiant unique donné à une et une seule ressource du Web ou du Web sémantique.

spécifiée : on pourrait par exemple décider de représenter toutes les descriptions sous forme de triplets entourés de parenthèses. Cependant, RDF étant issu du W3C, il est assez naturel de vouloir le représenter dans des documents XML. Ainsi, l'espace de noms XML `rdf` a été réservé pour l'inclusion de descriptions RDF dans des documents XML.

Voyons comment on peut représenter l'exemple précédent en XML :

```
<rdf:RDF>
  <rdf:Description about="http://www.limsi.fr/">
    <s:date-de-modification>2003-02-11</s:date-de-modification>
  </rdf:Description>
</rdf:RDF>
```

On a vu qu'il est possible de représenter graphiquement des assertions RDF. En fait, une description RDF sous-tend un graphe :

- les éléments noeuds de l'arbre XML (`rdf :Description`), c'est à dire les ressources, correspondent aux noeuds du graphe ;
- les éléments propriétés correspondent aux arcs entre les noeuds.

Dans l'exemple précédent, on utilise deux espaces de noms : `rdf` et `s`. On l'a vu, l'espace de nom `rdf` correspond aux balises utilisées pour décrire les descriptions RDF elles-mêmes. Quant à l'espace de noms `s`, il fait référence au *schéma* RDF utilisé.

En effet, jusqu'à présent, on n'a dit nulle part quels étaient les attributs autorisés, à quelles ressources ils s'appliquaient, quelles étaient leurs valeurs admises... C'est le schéma RDF qui précise tous ces points. Le schéma donne véritablement sa sémantique à la description RDF. Il permet de décrire ressources et propriétés dans un modèle objet : il existe des classes de ressources et des classes de propriétés. Il est possible de définir des relations entre ces différentes entités : relations de subsomption entre classes bien entendu, mais aussi définition de classes par opérations ensemblistes (union, intersection, différence symétrique, etc.). On peut aussi définir des sous-classes par restriction des valeurs des propriétés des super-classes.

En un mot, le schéma définit le vocabulaire utilisé dans une description RDF. On peut imaginer à loisir de nombreux schémas différents, adaptés chacun à un domaine ou à une application spécifique. Il faut noter que les schémas sont eux-mêmes écrits en RDF, en utilisant des balises de l'espace de nom du langage RDF Schema[6], souvent désigné par le préfixe `rdfs`.

En particulier, le Dublin Core peut être traité comme l'un de ces schémas RDF, un schéma adapté à la description de pages Web.

Un autre schéma RDF en vogue actuellement est RSS (RDF Site Summary), qui permet de donner des résumés de sites Web. En particulier, ce format est bien adapté aux sites qui présentent une liste de nouvelles : avec RSS, on peut présenter objectivement cette liste de nouvelles, de façon indépendante d'une quelconque présentation. Par exemple, la page d'accueil du W3C peut être obtenue sous forme de résumé RSS. Il est également possible de fusionner de multiples sources de données RSS en une seule présentation, ou encore de présenter sur un site Web une source d'information provenant d'un autre site.

[6] Voir `http://www.w3.org/TR/rdf-schema/`

Pour notre application, nous utiliserons RDF car ce langage connaît actuellement un essor rapide, si bien qu'il s'est déjà beaucoup répandu, et qu'il dispose des mêmes fonctionnalités que les autres modèles comme les graphes conceptuels.

3.4 Langages d'ontologies

On vient de voir qu'il est possible de construire des descriptions sous forme de graphes qui mettent en relation des concepts. Mais pour que le système puisse fonctionner, les classes auxquelles appartiennent ces concepts doivent être clairement définies et être partagées par tous les acteurs.

C'est le rôle des *langages d'ontologies* de définir les taxonomies de concepts, et les règles de raisonnement sur ces taxonomies. C'est ce qu'ébauche le langage RDF Schema, mais de façon trop limitée. Ainsi, tous les langages d'ontologie que nous allons décrire sont conçus comme des *extensions* du langage RDF Schema.

3.4.1 DAML+OIL

Le langage DAML[7] permet aux agents de partager de la sémantique.

DAML est associé à OIL[8], qui est un autre langage de description d'ontologies. OIL est un langage de description et d'inférence sur les ontologies, basé sur RDF. Il se base sur les logiques de description [Borgida 1995].

DAML et OIL ont donc été développés séparément, mais sont réunis au sein du couple DAML+OIL, qui repose sur RDF. Ainsi, DAML+OIL cherche à combiner toutes les caractéristiques de DAML, d'OIL, et de RDF Schema [Horrocks 2002b, Horrocks 2002a]. Il permet de modéliser les aspects suivants :

- définition de classes de propriétés ;
- définition de classes de ressources ;
- relations logiques entre classes (disjonction, union, équivalence, etc.) ;
- relations d'héritage entre classes ;
- restriction de propriétés (cardinalité, etc.) et typage ;
- prise en charge des collections (listes) ;
- instanciation de classes de propriétés et de ressources.

Voici un exemple d'extrait d'ontologie DAML+OIL :

```
<daml:Class rdf:ID="Homme">
  <rdfs:subClassOf rdf:resource="#Humain" />
</daml:Class>

<daml:Class rdf:ID="Femme">
  <rdfs:subClassOf rdf:resource="#Humain" />
  <daml:disjointWith rdf:resource="#Homme" />
</daml:Class>
```

[7] DARPA Agent Markup Language, voir http://www.daml.org/2000/10/daml-ont.html
[8] Ontology Inference Layer

On explique que la classe *Humain* a deux sous-classes disjointes : *Homme* et *Femme*.

```
<daml:Class rdf:ID="Père">
  <rdfs:sameClassAs>
    <daml:Restriction daml:minCardinality="1">
      <daml:onProperty rdf:resource="#possèdeEnfants"/>
      <daml:toClass rdf:resource="#Homme"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

On dit ici que la classe des *Pères* est précisément la classe des *Hommes* pour lesquels la cardinalité de l'attribut *possèdeEnfants* vaut au moins 1.

Comme application de DAML+OIL, l'on peut citer DAML-S, qui est une ontologie de description de services Web.

3.4.2 OWL

OWL[9] est un nouveau langage défini par le W3C : à la date de rédaction de ce document, les spécifications définitives ne sont pas encore parues. Le projet de spécification est passé le 18 août 2003 au stade de *Candidate Recommendation*. OWL est censé être le futur langage d'ontologie universel du Web sémantique. En ce sens, il devrait supplanter RDF Schema (qu'il étend) et DAML+OIL.

Comme les langages précédemment évoqués, OWL est construit au-dessus de RDF. Un document OWL est donc composé de triplets RDF, qui peuvent être écrits dans la syntaxe RDF de son choix. Au sein d'un document, les triplets RDF non définis dans la spécification OWL sont ignorés.

Tout comme DAML+OIL, mais contrairement à RDF Schema, OWL permet de faire des raisonnements, des inférences sur l'ontologie. Ainsi, OWL permettra de conclure sur des faits implicites, alors que RDF Schema ne sait traiter que l'explicite.

Il existe trois sortes d'OWL :

- OWL *Lite* : c'est une version d'OWL aux fonctionnalités réduites, mais qui restent quand même suffisantes pour bien des usages, comme la constitution de taxonomies ou de thésaurus ;
- OWL *DL* : il correspond exactement aux logiques de description. Ce langage est expressif, mais les procédures d'inférences sont complètes, et effectuables en un temps fini ;
- OWL *Full* : il donne à l'utilisateur une expressivité maximale, mais on n'a aucune garantie quant à la complétude et à la terminaison des procédures d'inférence.

Une ontologie OWL se compose des éléments suivants :

- en-têtes optionnels : commentaire, version, importation d'ontologie ;
- éléments de classe ;
- éléments de propriétés ;

[9] Web Ontology Language, voir `http://www.w3.org/TR/owl-features/`

– instances.

OWL sépare l'univers en deux :

– d'une part le domaine des *types de données* (types de données de schémas XML) ;
– d'autre part les *objets*, membres de classes OWL ou RDF.

En OWL, la définition des classes se fait à quelques différences près comme en DAML+OIL. Mais voyons ce qu'apporte ce langage pour la définition de propriétés, grâce à un exemple inspiré du guide d'OWL du W3C [Smith 2002][10] :

```
<owl:ObjectProperty rdf:ID="situéDans">
  <rdf:type rdf:resource="owl:TransitiveProperty" />
  <rdfs:domain rdf:resource="owl:Thing" />
  <rdfs:range rdf:resource="#Lieu" />
</owl:ObjectProperty>

<Region rdf:ID="Orsay">
  <locatedIn rdf:resource="#IleDeFrance" />
</Region>

<Region rdf:ID="LIMSI">
  <locatedIn rdf:resource="#Orsay" />
</Region>
```

On indique tout d'abord que la propriété `situéDans` est transitive. Après les deux assertions suivantes (*"Orsay est situé en Île de France"* et *"le* LIMSI *est situé à Orsay"*), on peut conclure sur : *"le* LIMSI *est situé en Île de France"*.

3.5 Difficultés dans la définition d'ontologies

En pratique, la définition et l'utilisation d'ontologies est assez difficile, pour plusieurs raisons.

3.5.1 Définition d'ontologie

Il est difficile de définir une fois pour toute une ontologie, ou un ensemble d'ontologies qui couvrent l'intégralité d'un champ d'application donné. Par exemple, on risque d'oublier des concepts, ou de faire des choix de classification qui pourront s'avérer mauvais par la suite.

L'une des solutions est de faire appel à des ontologies non figées : ainsi, l'ontologie évolue au cours du temps, de nouveaux concepts étant ajoutés par les utilisateurs au fur et à mesure des besoins. Ainsi, dans le système OntoShare [Davies 2003], les utilisateurs peuvent classifier des documents au sein d'une taxonomie préexistante. Mais s'ils ne

[10] Une version provisoire est disponible à l'adresse `http://www.w3.org/TR/owl-guide/`

trouvent pas de catégorie adaptée, ils ont la possibilité d'ajouter une nouvelle catégorie à la taxonomie.

Pour aider la création d'ontologies complexes, il est possible de fournir à l'utilisateur des outils spécifiques, qui permettent l'édition d'ontologies à travers une interface graphique, comme Hozo [Mizoguchi 2001].

3.5.2 Partage d'ontologie

Lors de l'utilisation pratique de systèmes basés sur des ontologies, deux cas peuvent se présenter :

1. soit le domaine d'intérêt possède une ontologie clairement définie et acceptée par tous : cela pourrait par exemple être le cas de l'indexation de documents de type "page Web". Dans ce cas, il est fort probable que tous les acteurs utiliseront cette ontologie consensuelle. Pour reprendre l'exemple, l'ontologie qui s'imposerait serait le Dublin Core ;
2. soit il n'y a pas d'ontologie consensuelle : dans ce cas, chaque concepteur de système va définir sa propre ontologie, et il va devenir assez difficile de faire coopérer les agents, étant donnée une certaine *hétérogénéité sémantique*.

Malheureusement, on va souvent se trouver dans la situation (2). Dans ce cas, il faudra trouver un moyen de mettre en correspondance différentes ontologies, comme dans [Goasdoué 2003]. Il existe des systèmes capables de mettre automatiquement en correspondance deux ontologies différentes [Doan 2002].

3.6 Conclusion

Actuellement, les moyens d'échange de l'information sémantique commencent à être normalisés. D'une part, un large consensus est en train de se former autour de RDF, une plate-forme de modélisation de sémantique inspirée de travaux antérieurs sur les réseaux sémantiques. D'autre part, l'adoption d'OWL permettra à tous les acteurs de s'échanger des ontologies.

Cependant, de gros progrès restent à accomplir dans le domaine des ontologies : il serait souhaitable de disposer d'ontologies communes pour les applications courantes. En l'absence d'une telle normalisation des ontologies, chacun en est réduit à définir sa propre ontologie, ce qui pose de gros problèmes d'interopérabilité des systèmes.

Chapitre 4

Proposition d'ontologie

Nous avons vu dans le chapitre précédent les fondations du Web sémantique, et la place qu'y trouvent les ontologies. Si ce n'était suffisant, [Mizoguchi 2001] nous rappelle ce que peut apporter une ontologie :

- tous les acteurs (humains ou logiciels) peuvent partager le même vocabulaire commun ;
- les suppositions et la conceptualisation sont explicitées, ce qui permet le partage et la réutilisation des connaissances ;
- la connaissance est systématisée et standardisée ;
- une ontologie est définie par rapport à un méta-modèle, donc il peut exister des outils qui travaillent au niveau de ce méta-modèle, et effectuent des traitements sur des ontologies.

Ces raisons nous poussent donc à utiliser une ontologie pour nos annotations sémantiques.

Cette ontologie doit recouvrir les aspects classiques de la vie dans des bâtiments (par exemple, les bâtiments d'une entreprise) : attributions de pièces, rôles dans une organisation, plannings, etc.

Aucune ontologie consensuelle n'existe dans ce domaine. Nous définirons donc notre propre ontologie, sachant qu'il serait relativement aisé de la mettre en correspondance avec d'autres ontologies équivalentes en cas de besoin.

Nous nous inspirons de l'ontologie CoBrA[1] définie pour le "pervasive computing" à l'université du Maryland [Chen 2003].

4.1 Classes

Nous répartissons les objets d'intérêt dans trois grandes catégories :

- les lieux ;
- les personnes ;
- les activités.

[1] Context Broker Architecture

Nous allons voir successivement les classes que nous définissons dans chacune de ces catégories.

4.1.1 Lieux

Un endroit classique, par exemple une pièce ou un couloir est modélisé par la classe `Place`. Les instances de `Place` correspondent en fait à des éléments syntaxiques ou d'agrégation dans la description de la structure.

La classe `Function` permet de créer des fonctions pour des pièces. Des exemples d'instances de la classe `Function` pourraient être par exemple `conferenceRoom`, `office` ou encore `machineRoom`.

Il est possible de créer des catégories de pièces (vis-à-vis de consignes de restriction d'accès, par exemple) à l'aide de la classe `PlaceCategory`. Par exemple, les zones accessibles aux visiteurs seraient représentées par une instance `publicArea`, les zones accessibles seulement au personnel par `privateArea`, et les zones de haute sécurité par `highSecurityArea`.

4.1.2 Personnes

Les classes qui modélisent les personnes sont construites sur le même schéma que celles qui modélisent les lieux.

Il existe déjà un projet qui a pour but de définir un ensemble de classes qui décrivent les personnes et leurs relations : c'est le projet FOAF[2]. FOAF définit entre autres la classe `foaf:Person` qui représente une personne physique, en donnant par exemple ses nom, prénom, surnom, etc. Nous utiliserons donc cette classe `foaf:Person` pour les personnes interagissant dans les bâtiments. Les classes qui interviennent dans FOAF sont définies en RDF/OWL[3].

La figure 4.1 montre comment on peut relier deux instances de la classe `foaf:Person` grâce à des propriétés FOAF. Dans notre cas, nous introduirons en plus nos propres propriétés pour relier les objets de type `foaf:Person` aux autres objets de nos données sémantiques (lieux, plannings, etc.).

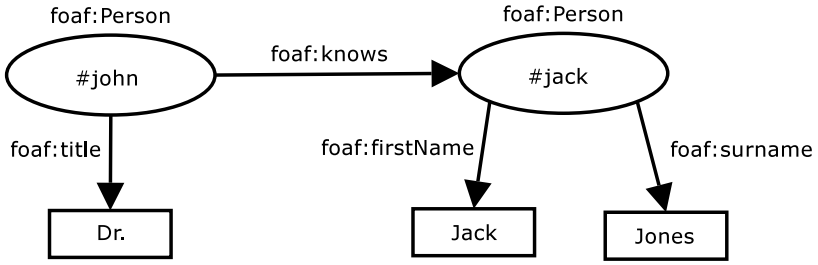


FIG. 4.1 – Exemple de graphe RDF utilisant l'ontologie FOAF

[2]Friend Of A Friend, voir `http://www.foaf-project.org/`

[3]Voir la description de l'ontologie FOAF en langue naturelle et en OWL à l'adresse `http://xmlns.com/foaf/0.1/`

De même que nous avions défini une classe pour représenter les fonctions des lieux, nous définissons une classe `Role` pour représenter les rôles des personnes par rapport à leur organisation. Des instances de `Role` peuvent être par exemple : `chiefExecutiveOfficer`, `professor`, `watchman`, etc.

Et, toujours par analogie avec les lieux, nous définissons une classe pour représenter des catégories de personnel : `PersonnelCategory`. Par exemple, nous pourrions avoir les catégories suivantes : `intern`, `junior`, `senior`, etc.

4.1.3 Activités

Les activités permettent de représenter ce que font les personnes dans les lieux d'intérêt.

La classe `Event` représente un événement qui se déroule pendant un laps de temps donné et en un lieu donné.

La classe `Planning` permet de regrouper plusieurs événements associés à un même lieu.

Nous venons donc de décrire un certain nombre de classes d'objets (ou encore : *ressources* pour reprendre la terminologie du modèle RDF) utilisés dans les descriptions sémantiques. Selon le modèle RDF, les ressources sont mises en relation par des propriétés. Nous allons donc décrire des propriétés qui nous permettent de construire des graphes à partir des classes définies ci-avant.

4.2 Propriétés

Pour répondre à nos besoins, nous avons défini un certain nombre de propriétés.

Pour chaque propriété, nous indiquons son nom, ses domaines et codomaines[4](sous la forme d'une liste d'expressions `domaine` → `codomaine`), et une courte description de sa sémantique.

Propriété :	hasCategory
Utilisation :	`Place` → `PlaceCategory`, `foaf:Person` → `PersonnelCategory`, `Role` → `PersonnelCategory`, `Function` → `PlaceCategory`
Description :	Indique qu'un lieu, une personne, une fonction ou un rôle appartient à une certaine catégorie (de lieux ou de personnes).
Propriété :	hasFunction
Utilisation :	`Place` → `Function`
Description :	Indique qu'un lieu a une certaine fonction.

[4]Nous utilisons les mots français *domaine* et *codomaine*, qui correspondent respectivement aux termes anglais *domain* et *range* utilisés dans les textes techniques consacrés à RDF.

Propriété : isAssignedTo
Utilisation : `Place` → `foaf:Person`
Description : Indique qu'un lieu est attribué à une certaine personne pour son usage propre.

Propriété : hasPlanning
Utilisation : `Place` → `Planning`
Description : Indique qu'un planning est associé au lieu courant.

Propriété : worksIn
Utilisation : `foaf:Person` → `Place`
Description : Indique que la personne courante travaille dans un lieu donné.

Propriété : hasRole
Utilisation : `foaf:Person` → `Role`
Description : Indique que la personne courante joue un rôle donné au sein de l'organisation

Propriété : hasAccessTo
Utilisation : `foaf:Person` → `Place`, `PersonnelCategory` → `PlaceCategory`
Description : Indique qu'une personne, ou qu'une catégorie de personnel, ont accès, respectivement, à un lieu ou à une catégorie de lieux.

Propriété : hasEvent
Utilisation : `Planning` → `Event`
Description : Indique qu'un planning contient un événement donné.

Propriété : date
Utilisation : `Event` → `rdfs:Literal`
Description : Date d'un événement.

Propriété : start
Utilisation : `Event` → `rdfs:Literal`
Description : Heure de début d'un événement.

Propriété : end
Utilisation : `Event` → `rdfs:Literal`
Description : Heure de fin d'un événement.

Propriété : end
Utilisation : `*` → `rdfs:Literal`
Description : Permet de donner un nom à une ressource. S'applique à toutes les ressources. C'est ce nom qui pourra être donné à l'utilisateur lors de l'exploitation de la description sémantique.

Nous envisageons de définir formellement notre ontologie à l'aide du langage OWL, car il s'agit du langage d'ontologie le plus complet, et il semble destiné à jouer un rôle important dans le cadre du Web sémantique.

4.3 Exemple

Afin d'illustrer notre concept d'annotation sémantique, nous proposons un exemple de description très simple, dont le graphe est donné en annexe 7.2 (p. 63).

Cette description a été éditée sous forme graphique à l'aide du logiciel IsaViz[5], développé par le W3C.

On rappelle la convention évoquée en 3.3.2 (p. 30) : les ressources sont représentées par des ellipses, les littéraux par des rectangles, et les propriétés par des flèches étiquetées par le nom de la propriété.

La classe des objets est indiqué par une propriété `rdf:Type` pointant sur la ressource de la classe correspondante.

Dans notre exemple, deux ressources sont liées aux objets physiques de la description de structure évoquée en première partie : les deux pièces `room210` et `room34`. Elles sont liées à leur fonction, à une personne (ressource `johnSmith`), ou encore à un planning (ressource anonyme de type `Planning`).

Sur ce graphe, on voit qu'il serait possible de se livrer à des inférences de façon à faire apparaître de nouvelles propriétés. Par exemple, étant donné d'une part que `johnSmith` appartient à la catégorie de personnel `seniorEmployee`, catégorie qui a accès aux lieux de la catégorie `openToPersonnel`, étant donné d'autre part que le lieu `room34` à une fonction de `conferenceRoom`, et que les lieux de fonction `conferenceRoom` appartiennent à la catégorie `openToPersonnel`, l'on pourrait raisonnablement en déduire que `johnSmith` a accès à `room34`.

Il est possible d'automatiser ce genre de déduction en insérant les règles d'inférence correspondantes dans la description d'ontologie exprimée en langage OWL.

4.4 Critiques

L'ontologie que nous venons de définir permet d'annoter sémantiquement nos descriptions de structure de bâtiments dans des cas relativement simples. Cependant, il est impossible d'imaginer toutes les annotations que nous pourrions vouloir représenter.

Par exemple, lors de la description d'un musée, nous pourrions vouloir annoter chacun des tableaux à l'aide d'une sous-ontologie bien précise : référence au peintre, année de réalisation, référence aux mécènes, historique du tableau, etc.

Avec RDF, il est tout à fait possible de faire appel à une nouvelle ontologie qui n'a pas été prévue au départ. Certes, le système pourra éventuellement raisonner (*i.e.* réaliser des inférences) sur ces informations, mais il ne sera pas capable de *présenter* ces informations sous une forme compréhensible par l'utilisateur.

Ceci constitue actuellement une limitation du système. Il faudrait systématiquement accompagner chaque morceau d'ontologie des instructions permettant la présentation à l'utilisateur d'informations exprimées par rapport à cette ontologie.

[5] `http://www.w3.org/2001/11/IsaViz/`

4.5 Représentation et liaison avec la description de structure

Les descriptions RDF pourraient être représentées de façon assez naturelle grâce au formalisme XML adapté à RDF. Cependant, ce formalisme possède l'inconvénient d'être assez "lourd" (fichiers généralement très longs, même pour des descriptions assez simples). Donc en cas de problèmes dûs à nos contraintes d'embarquabilité, nous pourrions envisager d'utiliser le formalisme N3, qui représente les descriptions RDF sous forme de suite de triplets, et qui possède l'avantage d'être un peu plus concis que le formalisme RDF/XML.

On l'a dit plus haut, les ressources qui représentent des lieux dans les graphes sémantiques sont en fait des objets qui appartiennent à la description de structure. Ce sont donc ces ressources qui assurent la liaison entre les descriptions de structure décrites en première partie (et représentées en XML), et les informations sémantiques associées dont nous venons de parler (et représentées en RDF).

Un moyen assez naturel serait de munir les éléments de la description de structure d'un attribut `id`, qui correspondrait au nom de la ressource associée dans la description RDF. Ainsi, la liaison entre les deux "mondes" serait assurée de façon simple et efficace.

4.6 Conclusion

L'ontologie que nous proposons constitue un point de départ pour la représentation de sémantique associée à des descriptions de structure. Nous avons décrit un mécanisme précis de liaison entre les éléments de structure et les annotations sémantiques associées.

Nous avons essayé de réutiliser du travail effectué par d'autres (avec l'ontologie FOAF), mais nous avons néanmoins dû recourir à la définition de nombreuses classes qui nous sont propres. Si à l'avenir un consensus émergeait dans certains de ces domaines, nous pourrions remplacer nos classes particulières par des morceaux de ces ontologies consensuelles.

Troisième partie

Algorithmes et implémentation

Chapitre 5

Algorithmes

De façon à exploiter les descriptions de structure décrites en première partie, nous avons mis au point des algorithmes de :

- localisation d'un point dans une description, étant données ses coordonnées spatiales ;
- détermination de l'information sémantique pertinente à fournir à l'utilisateur, c'est-à-dire du niveau de détail (*granularité*). En effet, à un élément particulier de l'architecture correspondent plusieurs informations sémantiques de niveaux de détail différents. Il convient alors d'effectuer une sélection pour déterminer le niveau de détail pertinent en fonction du contexte (positions de l'utilisateur et du point désigné à l'aide du laser, etc.)

5.1 Localisation d'un point

Cet algorithme est assez simple. Nous recherchons l'objet qui "englobe" un point donné. Donc nous nous intéressons au sous-arbre syntaxique. Nous partons de la racine (qui en toute logique doit englober le point), et au niveau de chaque nœud nous descendons, si possible, dans le sous-arbre qui englobe encore le point. Si aucun des sous-arbres n'englobe le point d'intérêt (ou si le nœud courant n'a pas de sous-arbre), on conclut que ce point d'intérêt se trouve au niveau du nœud courant.

En soi, cet algorithme est simple, mais nous avons besoin d'une procédure qui permette de tester *si un point donné est englobé dans un sous-arbre* (composé d'éléments syntaxiques).

En fait, ce problème se ramène à la question : *le point est-il à l'intérieur de l'élément syntaxique situé à la racine de l'arbre ?* Nous allons voir comment il est possible de répondre à cette question de façon simple.

Les points sont repérés par leurs trois coordonnées spatiales. Il est tout d'abord possible d'exploiter la coordonnée d'altitude de façon à déterminer une strate candidate (pour le moment, une strate correspond à un étage). Mais il faut alors déterminer dans quelle pièce, ou dans quel secteur se trouve le point, vis-à-vis de la strate déterminée.

Ces secteurs sont délimités par des structures linéaires : par exemple, des murs ou des murs virtuels. Ces structures linéaires (assimilables à des segments de droites) constituent

un découpage de la strate en un ensemble de polygones.

Il s'agit alors de déterminer dans lequel de ces polygones se trouve le point. Comme le nombre de polygones est réduit, le plus simple est de tester successivement la présence du point dans chacun des polygones.

On en arrive alors à la résolution du sous-problème suivant, qui permettra en fin de compte la détermination de la position d'un point arbitraire donné :

Problème de l'appartenance d'un point à un polygone

On utilise la méthode évoquée dans [Rigaux 2002] (chapitre 5, *Computational Geometry*, paragraphe 5.5.2, *Point in Polygon*, pp. 177–179).

Supposons que l'on souhaite déterminer si un point M appartient ou non à un polygone.

1. on choisit une demi-droite Δ quelconque, issue de M ;
2. on initialise le *nombre d'intersections* à 0 ;
3. on parcourt les côtés du polygone. Lorsqu'un côté coupe Δ, on ajoute 1 au *nombre d'intersections* ;
4. à la fin, si le *nombre d'intersections* est non impair, alors le point M est à l'intérieur du polygone. Sinon, il se trouve à l'extérieur.

En fait, des problèmes peuvent survenir si la demi-droite Δ choisie est parallèle à l'un des côtés du polygone. Nous ne voulons pas mettre de contrainte particulière sur Δ[1] il faut légèrement modifier l'étape (3), en précisant que si un côté du polygone est parallèle à Δ, alors *on ne compte pas d'intersection.*

De plus, il faut tester explicitement si le point M est l'un des sommets du polygone.

Quoi qu'il en soit, on obtient un algorithme rapide, car il est juste nécessaire de parcourir tous les côtés et tous les sommets du polygone au pire *une* fois, donc on obtient une complexité en $\Theta(n)$, où n est le nombre de sommets du polygone.

Voyons ce que donne cette méthode sur un exemple (voir fig. 5.1) :

Nous prenons pour Δ la demi-droite $[MP)$:

– dans le cas (1), Δ coupe deux côtés du polygone, donc M se situe à l'extérieur de ce polygone ;
– dans le cas (2), Δ coupe un seul côté du polygone, donc M se situe à l'intérieur de ce polygone.

5.2 Niveau de granularité

5.2.1 Quelques définitions

Soit E un ensemble appelé *ensemble des éléments syntaxiques*. Dans la suite, les éléments de E sont également appelés *nœuds*.

[1] Ainsi, un programme pourra par exemple choisir systématiquement Δ orientée selon l'un des axes du repère, ce qui simplifie grandement les calculs.

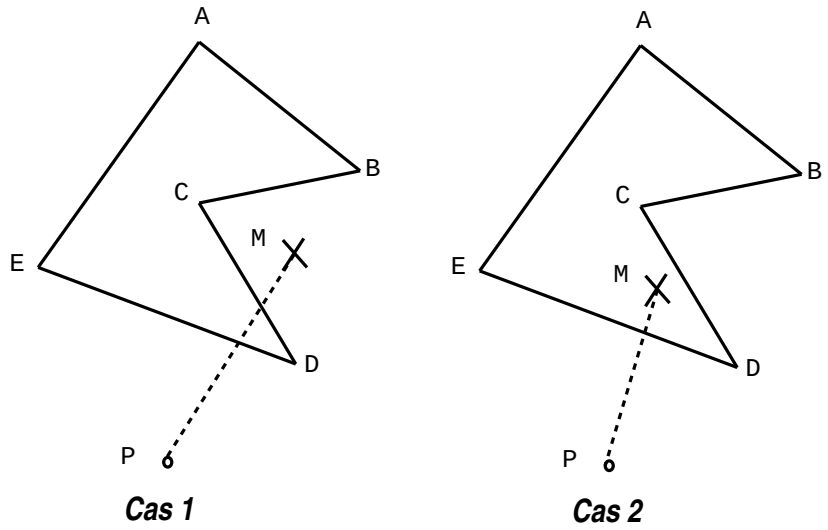


FIG. 5.1 – Détermination de l'appartenance d'un point à un polygone

On note $\mathcal{R}$ un élément distingué de E nommé *racine*.

Définition (arbre syntaxique)**.** On appelle *arbre syntaxique* une partie $\mathcal{A}$ de $E \times E$, qui possède les propriétés suivantes :

- $(\forall y \in E - \{\mathcal{R}\})\ \ (\exists! x \in E - \{y\})\ \ (x, y) \in \mathcal{A}$;
- $\{x \in E | (x, \mathcal{R}) \in \mathcal{A}\} = \emptyset$;
- $(\exists x \in E - \{\mathcal{R}\})\ \ (\mathcal{R}, x) \in \mathcal{A}$.

Autrement dit : les éléments de $\mathcal{A}$ sont appelés *arcs* (orientés). Un arc lie un nœud *père* à un nœud *fils*. Un nœud quelconque (sauf la racine) possède un et un seul père. La racine n'a pas de père.

Un arc $(x, y) \in \mathcal{A}$ est aussi noté : $x \to y$. x est le nœud *père* ; y est le nœud *fils*.

Définition (chemin)**.** On dit qu'il existe un *chemin* de $a \in E$ vers $b \in E$, et on note $a \xrightarrow{*} b$ s'il existe $N \in \mathbb{N}$ et des nœuds $\nu_0, \nu_1, \ldots, \nu_N$ avec $\nu_0 = a$ et $\nu_N = b$ tels que pour tout $i \in [0; n-1]$ il existe un arc $\nu_i \to \nu_{i+1}$.

Remarque. À propos des chemins :

- Notons qu'il est possible d'avoir $N = 0$, car pour tout nœud a on a : $a \xrightarrow{*} a$;
- Les chemins sont transitifs : on voit clairement sur la définition que si l'on a deux chemins $a \xrightarrow{*} b$ et $b \xrightarrow{*} c$, alors le chemin $a \xrightarrow{*} c$ existe ;
- On appelle *branche* un chemin qui part de la racine.

Définition. On définit sur E une relation d'ordre partiel, notée $\preceq$, et définie par :

$$(x \preceq y) \Leftrightarrow (\text{il existe un chemin } x \xrightarrow{*} y)$$

Démonstration. Vérifions qu'il s'agit bien d'une relation d'ordre :

- la relation est *réflexive* : on a remarqué qu'on a $a \xrightarrow{*} a$ pour tout a ;
- la relation est *transitive* : si $x \preceq y$ et $y \preceq z$, alors existent les chemins $x \xrightarrow{*} y$ et $y \xrightarrow{*} z$. D'après la propriété de transitivité des chemins, le chemin $x \xrightarrow{*} z$ existe, c'est à dire : $x \preceq y$;

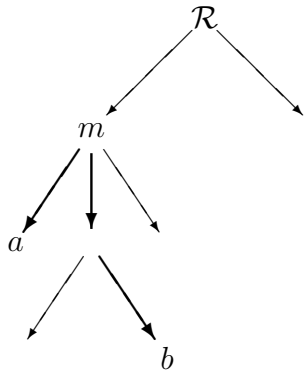


FIG. 5.2 – Un nœud a, un nœud b, et m leur plus grand ancêtre commun

– la relation est *antisymétrique* : si $x \preceq y$ et $y \preceq x$, alors existent les chemins $x \xrightarrow{*} y$ et $y \xrightarrow{*} x$. Si l'on avait $x \neq y$, il existerait une boucle dans l'arbre, ce qui est impossible.

□

Propriété. $(\forall x \in \mathcal{A})\ \ \mathcal{R} \preceq x$

Définition (ensemble des ancêtres de x)**.** $P(x) = \{y \in \mathcal{A} | y \preceq x\}$.

Propriété. $(\forall x \in \mathcal{A})\ \ \mathcal{R} \in P(x)$.

Propriété. Pour tout $x \in \mathcal{A}$, $P(x)$ est un ensemble totalement ordonné.

Définition (plus grand ancêtre commun de a et b)**.** Soit $\Omega = P(a) \cap P(b)$. $\Omega \subset P(a)$, donc Ω est totalement ordonné. Soit alors $m = \max(\Omega)$: m est appelé le *plus grand ancêtre commun de a et b*, et noté $\alpha(a, b)$. La figure 5.2 donne une illustration des positions relatives de a, b et $m = \alpha(a, b)$.

Définition (successeur)**.** Soit $(\Gamma, \trianglelefteq)$ un ensemble totalement ordonné. Soit $x \in \Gamma$: on nomme *successeur* de x l'élément minimal de l'ensemble $\{y \in \Gamma | x \triangleleft y\}$. On le note $\sigma_\Gamma(x)$, ou $\sigma(x)$ s'il n'y a pas d'ambiguïté sur l'ensemble ambiant.

5.2.2 Détermination du niveau de granularité

Illustrons notre propos par un exemple. Supposons que l'utilisateur soit situé dans le bâtiment du LIMSI, et qu'il pointe dans un bureau situé à l'intérieur de l'IDRIS, le bâtiment voisin (voir fig. 5.3).

Quelle information va-t-on lui donner ? Celle située au niveau de l'objet pointé ? de la pièce qui le contient ? du bâtiment IDRIS ? du campus ? de la ville ?...

Nous représentons la situation sous forme d'un arbre, où l'utilisateur est situé en un nœud u de l'arbre, et où l'objet pointé se trouve en un nœud p de l'arbre. On veut donc déterminer un nœud n de l'arbre qui contienne une information pertinente pour l'utilisateur.

Si $u = p$, alors on prendra de façon naturelle $n = u = p$. Dans la suite, nous supposons $u \neq p$.

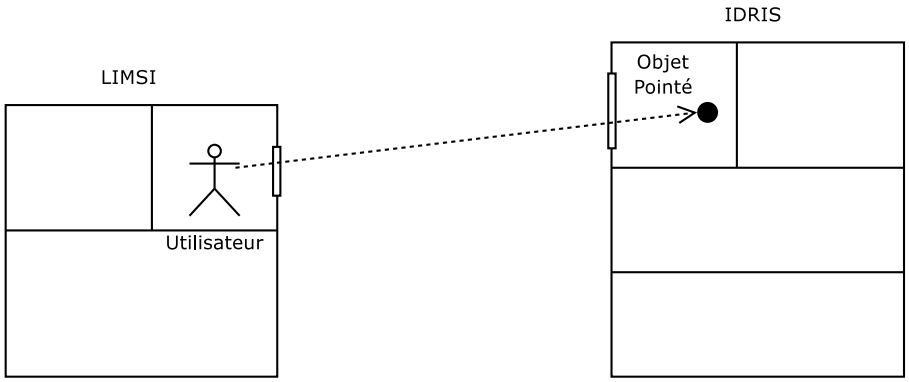


FIG. 5.3 – Cas où l'on cherche à déterminer le niveau de granularité

Soit c le plus grand ancêtre commun de u et p. $c \in P(p)$. Définissions alors n comme étant le successeur de p dans l'ensemble totalement ordonné $P(p)$.

Nous allons montrer en quoi ce choix de n est pertinent. Il peut se produire trois situations différentes :

u et p sont dans la même branche, avec $p \prec u$. Cette situation est impossible. En effet, l'utilisateur est *à l'intérieur* d'un certain nombre de "boîtes" encapsulées. Il n'est pas possible de viser l'une des boîtes englobantes, tout simplement car on est à un niveau de détail supérieur.

u et p sont dans la même branche, avec $p \succ u$ (voir fig. 5.4). Dans ce cas, $\alpha(u, p) = u$, et donc $n = \sigma(u)$. L'objet u correspond au milieu ambiant de l'utilisateur. Donc l'échelle de $\sigma(u)$ correspond aux objets présents dans le champ d'interaction direct de l'utilisateur. L'information extraite de $\sigma(u)$ se situera donc à un niveau de granularité pertinent.

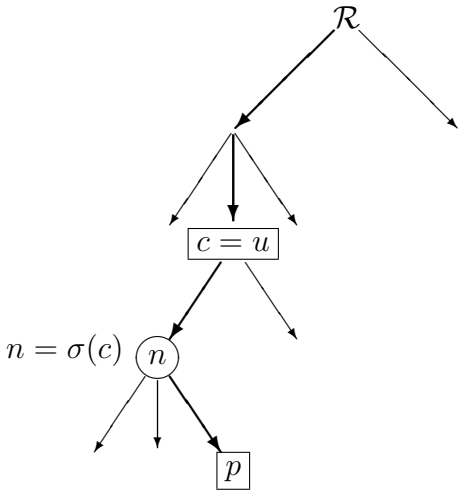


FIG. 5.4 – Cas où u et p sont dans la même branche

u et p ne sont pas dans la même branche (voir fig. 5.5). L'objet c possède deux sous-objets. u se trouve dans l'un de ces sous-objets, noté O_u, à une profondeur quelconque ; p se trouve dans l'autre, noté O_p, à une profondeur également quelconque. L'information pertinente (portée par l'objet n) doit se trouver dans

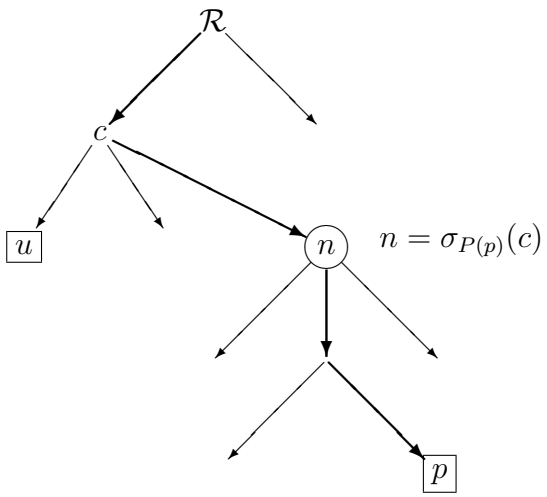


FIG. 5.5 – Cas où u et p sont dans des branches différentes

la branche $P(p)$. De plus, il faut que $n \succ c$, car dans le cas contraire l'information renvoyée pourrait tout aussi bien décrire l'endroit où se trouve l'utilisateur que l'endroit qu'il pointe : l'information n'aurait aucune pertinence car trop vague, trop peu détaillée. Mais par contre, il ne faut pas que l'information soit trop détaillée par rapport à l'objet O_p car l'utilisateur se trouve *à l'extérieur* d'O_p. Ces raisons nous poussent à choisir pour n un élément de la branche $P(p)$, tel que $n \succ c$ (donc dans l'objet O_p), mais qui soit le plus petit possible (au sens de $\preceq$). D'où le choix de $n = \sigma_{P(p)}(c)$, qui correspond précisément à ces critères.

Dans tous les cas, nous sommes capables de déterminer l'objet qui contiendra l'information pertinente vis-à-vis de ce que pointe l'utilisateur. L'information fournie sera contenue dans les annotations sémantiques associées à cet objet.

Si l'on reprend l'exemple évoqué en début de chapitre, l'arbre obtenu est celui donné en figure 5.6.

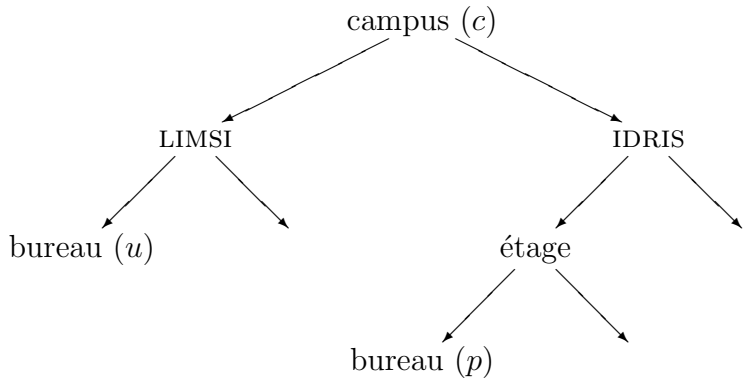


FIG. 5.6 – Arbre correspondant à l'exemple concret

Nous sommes dans le cas où u et p ne sont pas dans la même branche, donc nous renvoyons le fils de c situé dans la branche menant de la racine à p, c'est-à-dire l'information correspondant à l'IDRIS, ce qui semble raisonnable.

Chapitre 6

Réalisations pratiques

Dans ce chapitre, nous passons en revue les programmes développés dans le cadre de ce stage de DEA.

Ces programmes ont été réalisés en langage Java. Les traitements de fichiers XML ont été réalisés au travers de l'API DOM, dans son implémentation Xerces[1], issue du projet Apache.

6.1 Éditeur de modèle

Comme nous l'avons vu dans la première partie, notre modèle de bâtiment n'est pas figé, mais il est défini par rapport à un métamodèle. En pratique, un modèle est donc défini par un fichier XML (voir la fig. 2.6 p. 21).

Cependant, la saisie à la main d'un tel fichier XML est relativement fastidieuse, et sujette à de nombreuses erreurs. Nous avons donc développé un éditeur dédié qui nous permet de concevoir notre modèle au travers d'une interface graphique. Nous allons en détailler quelques aspects.

Les attributs des instances de notre modèle doivent se conformer à des types bien précis. Le langage XML Schema permet de spécifier ces types de données par l'intermédiaire d'expressions rationnelles du même format que celles utilisées par le langage Perl. Notre éditeur permet donc la définition de nos types de données, chacun étant défini par une expression rationnelle (voir fig. 6.1).

Lors de la définition du schéma d'une classe de documents, il arrive fréquemment qu'un certain nombre d'attributs soient souvent utilisés ensemble. Au lieu de spécifier tous leurs noms lors de la définition de chaque type de données qui les utilise, il est possible dans le langage XML Schema de définir des *groupes d'attributs*, c'est-à-dire de donner un nom à un ensemble d'attributs. Par la suite, il devient alors possible de se référer à l'ensemble des attributs ainsi définis par ce nom.

Notre langage de description de modèle dispose de la même fonctionnalité, et donc l'éditeur de modèle permet de spécifier des *groupes d'attributs* (voir fig. 6.2).

On définit donc des noms de groupes, et pour chaque groupe, on donne une liste

[1] Voir `http://xml.apache.org/xerces2-j/index.html`.

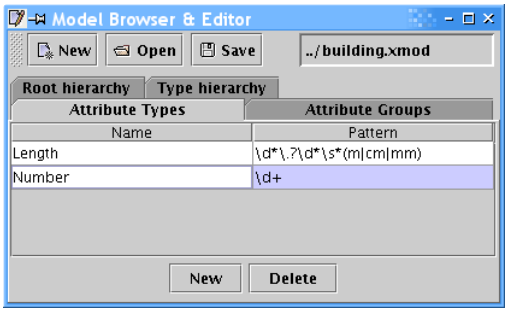


FIG. 6.1 – Éditeur de modèle — Types d'attributs

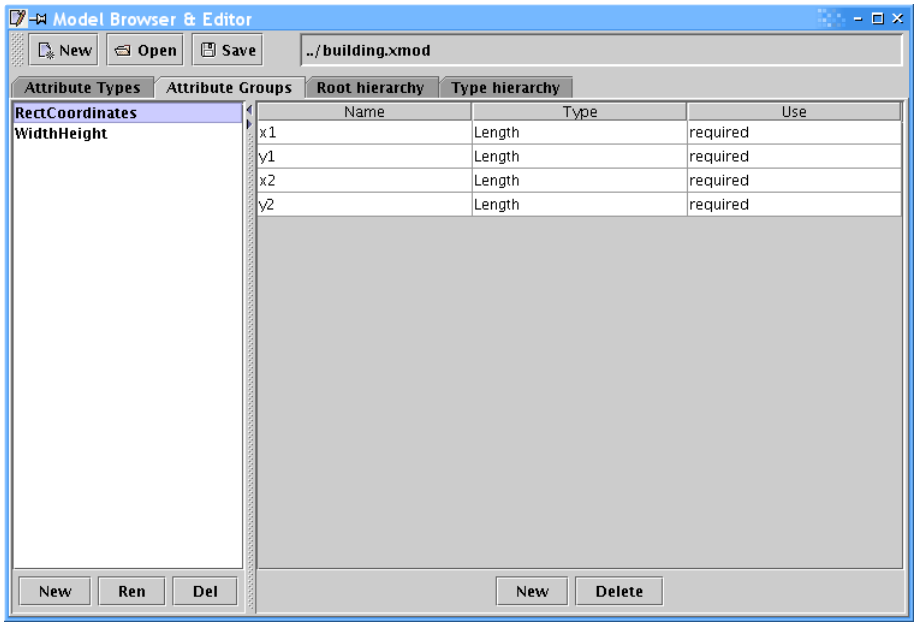


FIG. 6.2 – Éditeur de modèle — Groupes d'attributs

d'attributs, chaque attribut étant muni de son type et d'un attribut `required` qui indique si sa présence est obligatoire ou non.

Enfin, la fonctionnalité principale de l'éditeur est la création de la hiérarchie de types dans le modèle (voir fig. 6.3).

Grâce à la vue arborescente, on peut positionner chaque type de données à l'intérieur de la taxonomie.

Pour chaque type, on donne la liste des *sous-éléments*, c'est-à-dire des éléments qui peuvent être *inclus* dans les éléments du type donné.

On peut également spécifier les attributs propre à cet élément (en noir dans le tableau central), ainsi que choisir les groupes d'attributs dont il dispose (ils s'affichent en bleu dans le tableau). De plus, comme chaque type de données hérite des attributs de ses parents, ceux-ci s'affichent aussi dans le tableau (en vert).

Ainsi, le tableau central présente un récapitulatif complet de tous les attributs d'un type donné, ce qui confère une autre utilité à notre éditeur : on peut également l'utiliser pour naviguer dans le modèle et voir tout simplement quels concepts il contient, sans même chercher à utiliser les fonctionnalités d'édition.

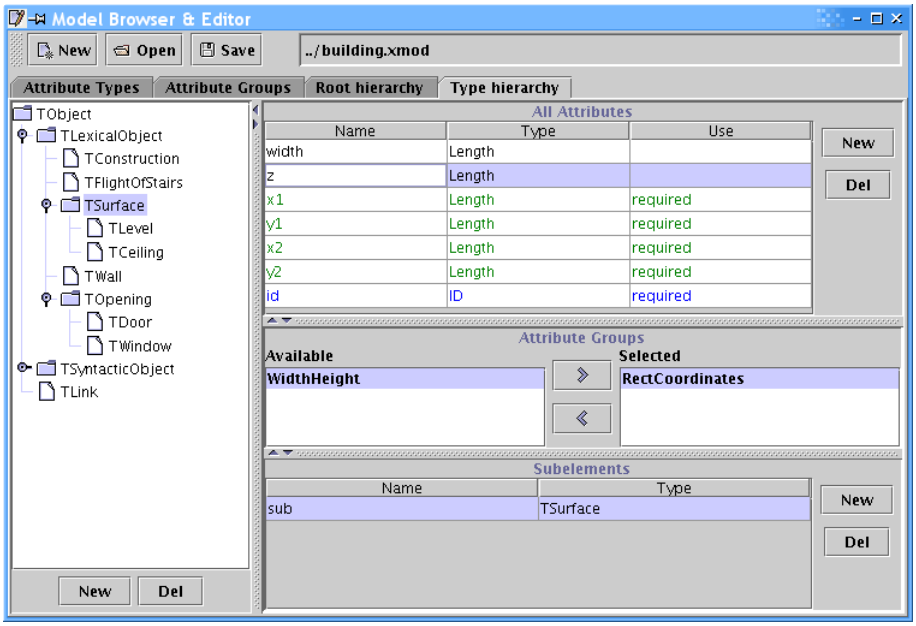


FIG. 6.3 – Éditeur de modèle — Hiérarchie de types

Cet éditeur charge et enregistre des descriptions de modèle au format XML. On peut alors produire le schéma XML correspondant par simple application d'une feuille de style XSLT à l'aide du processeur XSLT Xalan[2], issu tout comme Xerces du projet Apache.

6.2 Visualisateur d'instances

Afin de tester la pertinence de notre modèle, nous avons écrit des descriptions de bâtiments simples. Cependant, simplement écrire et valider un fichier XML a en soi peu d'utilité : il serait possible d'écrire des descriptions syntaxiquement valides, mais qui ne correspondraient raisonnablement à aucun bâtiment imaginable !

C'est pourquoi nous avons développé un simple visualisateur de descriptions de bâtiments.

Celui-ci charge en mémoire le fichier XML correspondant à une description de bâtiment, et affiche sa structure sous forme arborescente. En regard de cette structure, l'on a un aperçu du bâtiment sous la forme d'un plan qui s'approche d'un plan d'architecte.

Afin de tester la description, il est possible de sélectionner les éléments affichés dans l'arborescente. En guise de réponse, leur représentation correspondante passe en surbrillance dans la vue sous forme de plan.

Cette application de visualisation constituera une base de test afin d'essayer des implémentations d'algorithmes. À titre d'exemple, cette plate-forme nous a déjà permis de tester l'algorithme de recherche du plus grand ancêtre commun.

À l'avenir, il pourra être envisagé de transformer ce simple visualisateur en un éditeur. En effet, la saisie manuelle des descriptions d'architecture sous forme de fichier XML est extrêmement fastidieuse. Un éditeur graphique serait donc le bienvenu.

[2]Voir `http://xml.apache.org/xalan/`.

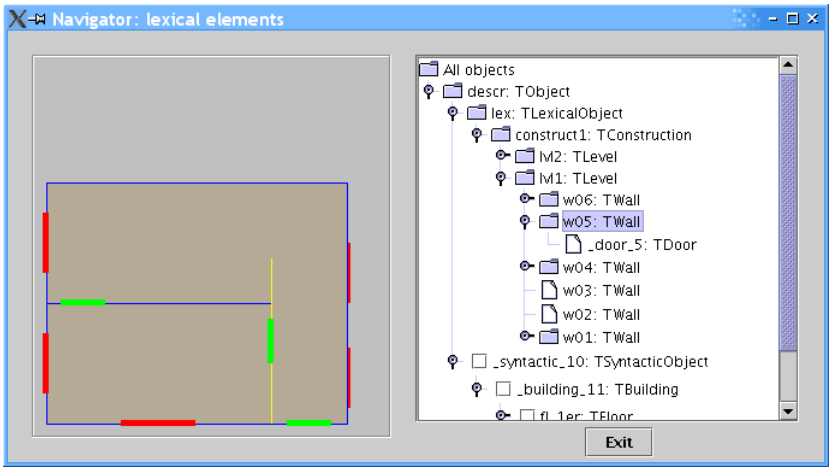


FIG. 6.4 – Visualisateur d'instances

Chapitre 7

Conclusion et perspectives

7.1 Bilan des résultats obtenus

Dans le domaine de la description de structure, nous avons défini un modèle et un métamodèle, ainsi qu'un langage de description de modèle et une feuille XSLT qui permet de générer des schémas à partir de descriptions de modèles.

De façon à exploiter ce modèle, nous avons mis au point des algorithmes de :

- localisation d'un point dans une description, étant données ses coordonnées spatiales ;
- détermination de l'information sémantique pertinente à fournir à l'utilisateur, c'est-à-dire du niveau de détail (*granularité*).

Dans le domaine de l'annotation sémantique de la structure, nous avons étudié les possibilités du modèle RDF et de ses langages de description d'ontologies associés, en particulier OWL.

Nous avons proposé une ontologie de base, s'appuyant sur des propositions en passe de devenir largement utilisées, de façon à exprimer nos informations sémantiques.

Nous avons évoqué une méthode de représentation pratique de nos annotations sémantiques en RDF, et de liaison avec nos descriptions de structure en XML.

Enfin, nous avons mis au point deux programmes en langage Java :

- un éditeur de modèle ;
- un visualisateur d'instances du modèle, qui permet de tester nos algorithmes.

7.2 Perspectives

Dans ce projet, nous définissons un modèle pour la description d'architectures dans le cadre de l'aide au déplacement des personnes non voyantes.

Cependant, le modèle et les algorithmes définis peuvent prendre place dans le cadre beaucoup plus général de la description d'environnement. Ils peuvent alors être utilisés dans des contextes divers :

- déplacement de robots autonomes dans des bâtiments ;
- aide à la navigation pour des personnes voyantes, par exemple dans des musées ;
- détermination d'itinéraires.

Le projet se situe à l'intersection de préoccupations liées à la représentation des connaissances (sémantique, structure, ontologies), à l'*ambient intelligence* [van Loenen 2003, Flury 2003] et au handicap.

En ce qui concerne la représentation de structure des bâtiments, le modèle défini contient à l'heure actuelle un certain nombre de types d'éléments architecturaux de base (murs, fenêtres, pièces, escaliers, etc.) Nous devrons encore ajouter de nouveaux concepts, de façon à ce qu'il soit utilisable pour décrire des environnements réels. En parallèle, il faudra décrire de vrais environnement (pour commencer, des bâtiments), de façon à tester la pertinence de notre modèle et la justesse de nos algorithmes.

En ce qui concerne les annotations sémantiques, il faudra là aussi enrichir l'ontologie définie, et éventuellement se rapprocher d'initiatives du même ordre, de façon à utiliser un vocabulaire non pas spécifique, mais qui bénéficie au contraire d'un consensus le plus large possible.

De plus, il faudra étudier et définir les méthodes d'interaction entre le dispositif et son utilisateur, de façon à disposer de moyens de présentation des annotations sémantiques. On peut ainsi imaginer que chaque ontologie soit accompagnée d'une sorte de *feuille de style* qui spécifie sa méthode de présentation.

Bibliographie

[Berners-Lee 2001] Tim Berners-Lee, James Hendler et Ora Lassila. *The Semantic Web*. Scientific American, mai 2001.

[Borgida 1995] Alexander Borgida. *Description Logics in Data Management*. IEEE Transactions on Knowledge and Data Engineering, vol. 7, no. 5, pages 671–682, octobre 1995.

[Chella 2002] Antonio Chella, Massimo Cossentino, Roberto Pirrone et Adrea Ruisi. *Modeling Ontologies for Robotic Environments*. In Proceedings of the Fourteenth Intemational Conference on Software Engineering and Knowledge Engineering, pages 77–80, Ischia, Italy, juillet 2002. ACM Press.

[Chen 2003] Harry Chen, Tim Finin et Anupam Joshi. *An ontology for a context aware pervasive computing environment, IJCAI workshop on ontologies and distributed systems*. In IJCAI'03, Acapulco, Mexico, août 2003.

[Davies 2003] John Davies, Alistair Duke et York Sure. *OntoShare – An Ontology-based Knowledge Sharing System for Virtual Communities Of Practice*. In I-KNOW 2003, Graz, Styria, 2003.

[Doan 2002] A. Doan, J. Madhavan, P. Domingos et A. Halevy. *Learning to Map between Ontologies on the Semantic Web*. In Proceedings of the World-Wide Web Conference (WWW-2002). ACM Press, 2002.

[Farcy 2002a] René Farcy et Yacine Bellik. *Comparison of Various Interface Modalities for a Locomotion Assistance Device*. In K. Miesenberger, J. Klaus et W. Zagler, éditeurs, Computers Helping People with Special Needs. Springer, 2002.

[Farcy 2002b] René Farcy et Yacine Bellik. *Locomotion Assistance for the Blind*. In S. Keates, P. Langdom, P.J. Clarkson et P. Robinson, éditeurs, Universal Access and Assistive Technology, pages 277–284. Springer, 2002.

[Flury 2003] Thibaud Flury et Gilles Privat. *An Infrastructure Template for Scalable Location-Based Services*. In Smart Objects Conference, pages 214–217, Grenoble, mai 2003.

[Goasdoué 2003] François Goasdoué et Marie-Christine Rousset. *Querying Distributed Data through Distributed Ontologies : a Simple but Scalable Approach*. In 18th International Joint Conference on Artificial Intelligence, 2003.

[Hadzilacos 1997] Thanasis Hadzilacos et Nectaria Tryfona. *An Extended Entity-Relationship Model for Geographic Applications*. SIGMOD Record, vol. 26, no. 3, pages 24–29, 1997.

[Horrocks 2002a] Ian Horrocks. *DAML+OIL : a Reason-able web ontology language*. In Christian S. Jensen, Keith G. Jeffery, Jaroslav Pokorný, Simonas Saltenis, Elisa Bertino, Klemens Böhm et Matthias Jarke, éditeurs, Proceedings of the Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, March 25-27, volume 2287 de *Lecture Notes in Computer Science*. Springer, 2002.

[Horrocks 2002b] Ian Horrocks, Peter F. Patel-Schneider et Frank van Harmelen. *Reviewing the Design of DAML+OIL : An Ontology Language for the Semantic Web*. In Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-02), pages 792–797, Menlo Parc, CA, USA, juillet 28– août 1 2002. AAAI Press.

[Lee 2000] Dongwon Lee et Wesley W. Chu. *Comparative analysis of six XML schema languages*. SIGMOD Record (ACM Special Interest Group on Management of Data), vol. 29, no. 3, pages 76–87, 2000.

[Miller 1998] Eric Miller. *An Introduction to the Resource Description Framework*. D-Lib Magazine, mai 1998.

[Mizoguchi 2001] Riichiro Mizoguchi. *Ontological Engineering : Foundation of the next generation knowledge processing*. In N.Zhong et al., éditeur, WI2001, volume 2198 de *Lecture Notes in Artificial Intelligence*, pages 44–57. Springer-Verlag, 2001.

[Rigaux 2002] Philippe Rigaux, Michel Scholl et Agnès Voisard. Spatial databases with application to gis. Academic Press, Morgan Kaufmann Publishers, 2002.

[Smith 2002] Michael K. Smith, Raphael Volz, Deborah McGuiness et Christopher Welty. *Web Ontology Language (OWL) Guide Version 1.0*. Rapport technique, W3C World Wide Web Concortium, 2002.

[Sowa 2001] John F. Sowa. *Conceptual Graphs Standard*. Rapport technique, ISO, 2001.

[Sowa 2002] John F. Sowa. *Architectures for Intelligent Systems*. IBM Systems Journal, vol. 41, no. 3, pages 331–349, 2002.

[Sowa 2003] John F. Sowa. Conceptual graphs : Bridging the gap between language and logic. Publié par l'auteur sur son site Web : `http://www.jfsowa.com/cg/cgonto.htm`, 2003.

[Studer 2003] Rudi Studer, Gerd Stumme, Siegfried Handschuh, Andreas Hotho et Boris Motik. *Building and Using the Semantic Web*, 2003.

[Swartz 2002] Aaron Swartz. *MusicBrainz : A Semantic Web Service*. IEEE Intelligent Systems, vol. 17, no. 1, pages 76–77, January–February 2002.

[van Leeuwen 2001] Jos P. van Leeuwen et A. J. Jessurun. *XML for Flexibility and Extensibility of Desing Information Models*. In Proceedings of CAADRIA 2001, Sydney, Australia, avril 2001.

[van Loenen 2003] Evert J. van Loenen. *On the Role of Graspable Objects in the Ambient Intelligence Paradigm.* In Smart Objects Conference, pages 3–7, Grenoble, mai 2003.

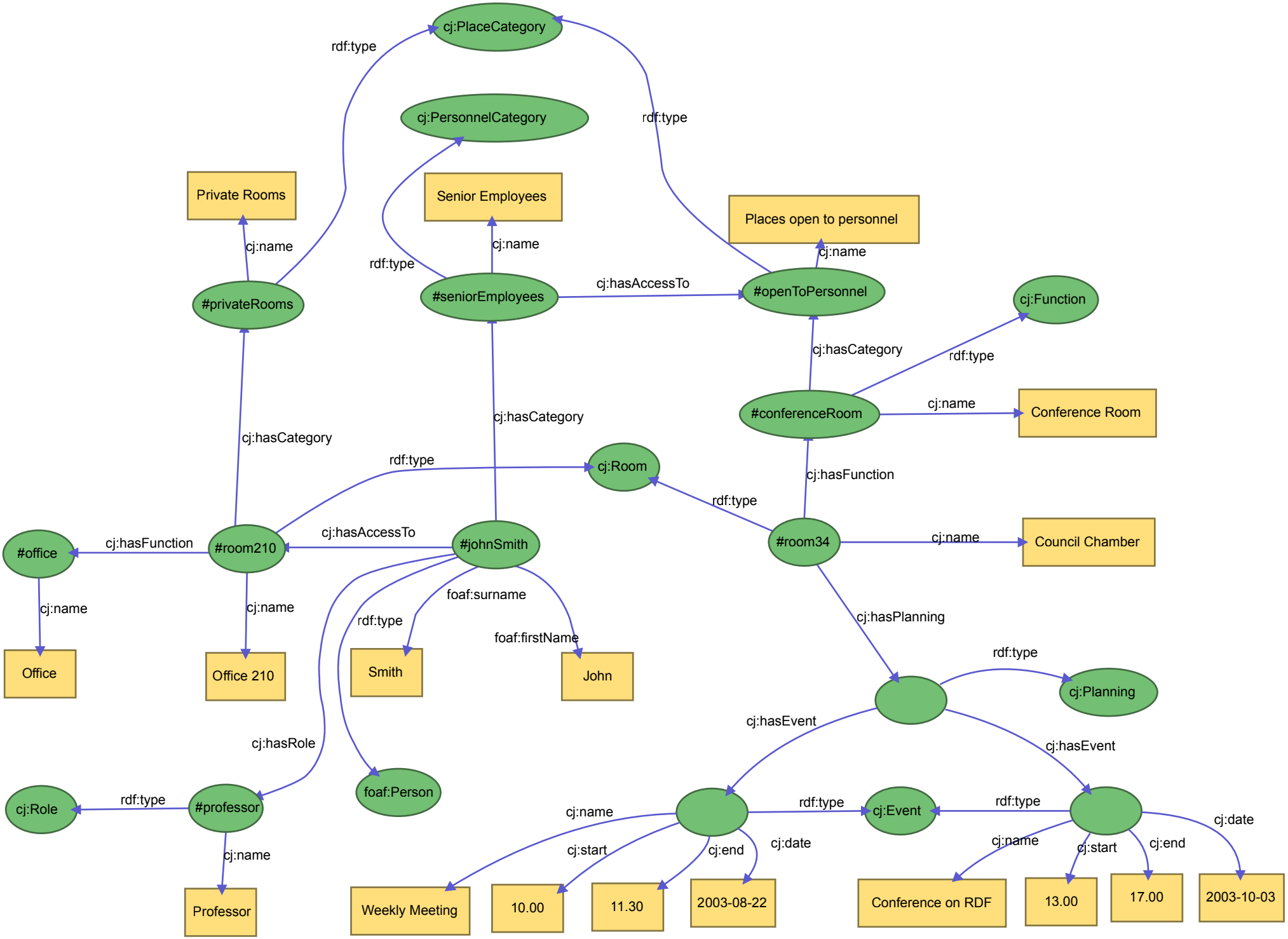
[van Rees 2002] Reinout van Rees, Frits Tolman et Reza Beheshti. *How bcXML Handles Construction Semantics.* In CIB W98 Workshop, Denmark, Århus, 2002.

[Weibel 2000] Stuart L. Weibel. *The Dublin Core Metadata Initiative.* D-Lib Magazine, vol. 6, no. 12, décembre 2000.

Table des figures

Graphe RDF

Nous donnons ici le graphe RDF servant à l'exemple du paragraphe 4.3 page 41.


cj:PlaceCategory
rdf:type
cj:PersonnelCategory
rdf:type
Private Rooms
Senior Employees
Places open to personnel
cj:name
cj:name
cj:name
rdf:type
#privateRooms
#seniorEmployees
cj:hasAccessTo
#openToPersonnel
cj:Function
cj:hasCategory
rdf:type
cj:hasCategory
cj:hasCategory
#conferenceRoom
cj:name
Conference Room
rdf:type
cj:Room
cj:hasFunction
rdf:type
cj:hasFunction
cj:hasAccessTo
#office
#room210
#johnSmith
#room34
cj:name
Council Chamber
cj:hasPlanning
cj:name
cj:name
rdf:type
foaf:surname
foaf:firstName
Office
Office 210
Smith
John
rdf:type
cj:Planning
cj:hasEvent
cj:hasEvent
cj:hasRole
foaf:Person
cj:Role
rdf:type
#professor
cj:name
rdf:type
cj:Event
rdf:type
cj:name
cj:start
cj:end
cj:date
cj:name
cj:start
cj:end
cj:date
Professor
Weekly Meeting
10.00
11.30
2003-08-22
Conference on RDF
13.00
17.00
2003-10-03