Conception et exploitation d'une base de métadonnées de traitements informatiques, représentation opérationnelle des connaissances d'expert – Application au domaine géographique

Yann Abd-El-Kader

THÈSE

pour l'obtention du grade de
DOCTEUR de l'UNIVERSITÉ de CAEN
spécialité : informatique

Conception et exploitation d'une base de métadonnées de traitements informatiques, représentation opérationnelle des connaissances d'expert

Application au domaine géographique

par

Yann Abd-el-Kader

soutenue le 3 juillet 2006

Jury :

Rapporteurs	:	Jérôme	Euzenat
		Michel	Mainguenaud
Examinatrice	:	Chantal	Reynaud
Directrice de thèse	:	Marinette	Revenu
Encadrante IGN	:	Bénédicte	Bucher

Remerciements

L'aventure de la thèse s'achève. Au-delà de la satisfaction du travail réalisé, je garderai un excellent souvenir des trois années passées au laboratoire Cogit. Je voudrais ici exprimer ma reconnaissance envers les personnes qui m'ont aidé et soutenu durant cette période.

Je souhaite tout d'abord remercier Bénédicte Bucher qui m'a encadré avec patience durant ces trois ans, Marinette Revenu qui a assumé le rôle de directrice de thèse, et Anne Ruas qui m'a accueilli au sein du laboratoire Cogit qu'elle dirige. Leurs conseils m'ont été bien utiles, notamment pour la rédaction du mémoire. Plus généralement, leur soutien m'a permis de mener ma thèse dans de bonnes conditions et avec un degré de liberté précieux.

Je remercie également Jérôme Euzenat, Michel Mainguenaud et Chantal Reynaud pour avoir accepté de faire partie de mon jury de thèse. J'ai recueilli leurs remarques et leurs critiques avec intérêt.
Parmi les personnes qui m'ont aidé, je tiens à remercier tout particulièrement Antoine Isaac et Raphaël Troncy, experts du Web sémantique. Leurs conseils éclairés sur les principes de représentation des connaissances et sur les subtilités du langage OWL m'ont été éminemment utiles.

Je remercie aussi Christiane Muller, pour sa relecture de mon mémoire, et Nicolas Sabouret, pour avoir bien voulu me fait part, dès la première année, de ses appréciations indulgentes. Cela a été un encouragement important, à un moment où le chemin à parcourir est encore long.

Préparer une thèse est effectivement une tâche de longue haleine. Je me prends presque à regretter qu'elle soit maintenant achevée, car le laboratoire Cogit aura vraiment été un lieu de travail très agréable. Je tiens à remercier pour cela les différents collègues croisés au cours de ces trois années ; je m'excuse de ne pouvoir les citer tous ici. Je mentionnerai toutefois Benoît Poupeau, Élisabeth Chesneau, Éric Grosso et Nathalie Abadie qui ont partagé successivement mon bureau, ainsi qu'Olivier Bonin dont j'ai apprécié la grande science informatique et le regard toujours aiguisé sur le monde de la recherche. Plus largement au sein de l'IGN, j'ai été très heureux de pouvoir discuter avec des personnes aussi nécessaires qu'Olivier Delbeke et Vincent Beauce.

Je voudrais pour finir spécialement remercier les personnes à qui je dois d'avoir pu réaliser cette thèse : les professeurs qui ont contribué à me former au cours de ma scolarité, la communauté des développeurs qui participent au partage libre des connaissances sur Internet, et enfin mes parents qui m'ont encouragé sur la voie des études.

Table des matières

Table des figures

Table des extraits de code

Liste des tableaux

Introduction

Contexte et objectifs de la thèse

Aujourd'hui, beaucoup d'organisations utilisent et développent des traitements informatiques dans le cadre de leurs activités. Elles cherchent à faciliter en leur sein le partage de ces traitements informatiques, mais aussi – c'est plus délicat – le partage des connaissances sur lesdits traitements. À l'Institut Géographique National (IGN), où s'est durant trois ans déroulée notre thèse, les développeurs et les utilisateurs des traitements informatiques ont besoin d'aide pour rechercher, connaître et partager les bibliothèques de fonctions, logiciels, *plug-in*, programmes isolés et algorithmes disponibles. Ces traitements dessinent les cartes, analysent les photos aériennes et exploitent les bases de données. Certains besoins, tels le traitement d'image, la généralisation cartographique[1] ou l'appariement[2] de base de données, sont spécifiques. Ils ne peuvent être satisfaits par les logiciels commerciaux standards du domaine, les Système d'Information Géographiques (SIG). C'est pourquoi les équipes de production et les laboratoires de recherche de l'IGN conçoivent et implémentent leurs propres traitements, associés ou non aux SIG existants. Il est nécessaire de décrire ces traitements, de recueillir les savoir-faire des experts et de les rendre accessibles aux novices.

Les réponses aux questions "Que font les traitements ? Comment fonctionnent-ils ? Comment les utiliser ? En existe-t-il d'adaptés à un besoin donné ?" sont souvent difficiles à trouver dans les documentations existantes. La dispersion et l'hétérogénéité des formats de ces dernières rendent, en l'état, impossible la construction d'un système d'information unifié. Le niveau de détail n'est, de plus, pas toujours à la hauteur souhaitée par l'utilisateur. Surtout, les documentations existantes sont en général statiques : elles ne peuvent fournir des modes d'emploi adaptés aux contextes d'utilisation particuliers (caractéristiques des données, environnement, connaissances de l'utilisateur).

Dans ce mémoire, nous soutenons la thèse qu'une réponse aux besoins d'informations sur les traitements informatiques peut être fournie par un système basé sur l'association de métadonnées structurées à des connaissances d'expert représentées de façon opérationnelle.

Nous contribuons à la résolution du problème de la description des traitements de plusieurs façons. L'analyse des besoins que nous effectuons permet d'identifier les différents types de traitements de notre contexte et de cerner les connaissances tacites ou explicites que possède l'expert mais qui manquent au novice. L'état de l'art des métadonnées des traitements que nous dressons ensuite, bien que partiel, permet d'obtenir un aperçu des forces et faiblesses des principales documentations existantes ainsi que des caractéristiques de quelques standards de descriptions de traitements, notamment des services Web. Plusieurs modèles de métadonnées sont exclusive-

[1] Il s'agit de simplifier la représentation des cartes lorsque l'échelle décroît, sauvegardant ainsi leur lisibilité pour l'œil humain à la perception limitée (cf. fig.1.2 p. 7).

[2] L'appariement de données géographiques consiste à établir des correspondances entre différentes représentations d'une même réalité du monde.

ment dédiés à un type particulier de traitements. Ils ne répondent donc que partiellement à nos attentes. Nous visons en effet une certaine généricité. Nous souhaitons, par exemple, décrire à la fois les façons d'utiliser les interfaces utilisateur des logiciels, comment appeler les fonctions des librairies dans des programmes, ou comment invoquer un service Web. Une autre spécificité du modèle de métadonnées que nous souhaitons est d'être à la fois approprié à la spécificité des traitements du domaine géographique (description fine des propriétés des données avant et après traitements, illustrations) et propre au recueil des connaissances d'expert. Pour ces raisons, nous sommes amenés à définir notre propre modèle de métadonnées.

Un aspect particulier et important de notre travail réside dans l'objectif de l'opérationnalisation des connaissances d'expert. Nous montrons que cet objectif est nécessaire : construire, grâce à l'instanciation du modèle défini, une base de métadonnées, ne peut suffire à répondre à toutes les requêtes des utilisateurs. En effet, les réponses à ces dernières ne peuvent être toutes stockées à l'avance. Il faut pouvoir dériver de l'information à partir de celle explicitement présente dans la base de métadonnées. C'est ce que sait faire l'expert, lorsqu'il met par exemple en œuvre ses connaissances pour déterminer les instructions à suivre adaptées à un contexte d'utilisation particulier. Par conséquent, les connaissances d'expert doivent être représentées au moyen d'un langage doté d'une sémantique opérationnelle.

Ce constat nous mène à adopter une double approche : *documentaire* et *orientée représentation des connaissances*. L'approche documentaire apparaît nécessaire afin de contrôler la structure et le contenu des métadonnées, conformément au modèle défini. Sa mise en œuvre se traduit par la construction d'un *Système d'information (SI)*. L'approche orientée représentation des connaissances est complémentaire de l'approche documentaire. Les connaissances d'expert y sont représentées de façon opérationnelle au sein d'un *Système à Base de Connaissances (SBC)*. Un SI peut, certes, également être doté de fonctionnalités exploitant les métadonnées. Mais les connaissances mises en œuvre y sont figées dans des algorithmes, et leur application est restreinte à des cas particuliers. Au contraire, dans un SBC, les connaissances sont des ressources exprimées de façon déclarative au moyen de langages dont les constructeurs sont associées à des opérations permettant d'effectuer des inférences (on parle de *sémantique opérationnelle*).

Concernant le volet "représentation des connaissances" de notre travail, nous étudions les principes, langages et outils du très actuel Web Sémantique. Nous le faisons pour deux raisons principales. Premièrement, il nous faut décrire et organiser les concepts des domaines informatique et géographique. Or le Web sémantique fournit des langages de définition d'ontologies (au sens de la représentation des connaissances). Deuxièmement, certains traitements géographiques se présentent désormais sous la forme de services Web. Il ne semble d'ailleurs pas impossible que cette tendance se généralise à l'avenir ; certains auteurs le pensent [GM97]. Or, les services Web sont l'objet d'une attention particulière dans la communauté du Web sémantique. Le but poursuivi est de permettre à des agents logiciels de découvrir, invoquer et interopérer avec ces services Web. Bien que, comme nous l'avons dit, les services Web ne constituent qu'une partie seulement des traitements à décrire, et que dans notre contexte l'invocation automatique ne soit pas un objectif, les modèles de description du Web sémantique ne peuvent que nous intéresser.

Comme dans le Web sémantique dont nous sommes donc proches, deux voies distinctes sont possibles. L'une consiste à produire des métadonnées destinées aux humains, l'autre des métadonnées destinées aux machines[3]. Dans notre contexte, les destinataires des métadonnées sont avant tout des humains. Mais nous ne renonçons pas pour autant à l'objectif d'une opérationnalisation des connaissances. Ce dilemme reflète l'existence de deux branches du domaine de la représentation des connaissances : concevoir des systèmes qui raisonnent ou des systèmes qui aident à raisonner.

[3]Ce que Caussanel *et al.* nomment respectivement "Web cognitivement sémantique" et "Web computationnellement sémantique" [CCZC02].

Notre thèse traite donc du problème de la représentation des connaissances ; elle traite également dans une moindre mesure de celui de leur acquisition. Nous montrons comment notre modèle de métadonnées favorise le recueil des connaissances d'expert, en particulier celles tacites sous forme de règles.

Organisation du mémoire

Au chapitre 1, nous effectuons une analyse des besoins d'informations sur les traitements. Nous tâchons de recenser les traitements à décrire et de cerner leurs spécificités. Nous tentons d'identifier les types de connaissances relatives aux traitements. Nous spécifions les objectifs à atteindre concernant le modèle de métadonnées et l'application proposée aux utilisateurs.

Au chapitre 2, nous dressons d'abord un état de l'art partiel des métadonnées des traitements informatiques en général, des traitements informatiques géographiques plus précisément ensuite, des traitements informatiques géographiques à l'IGN enfin. Ceci effectué, nous proposons notre propre modèle conceptuel de métadonnées.

Au chapitre 3, nous nous intéressons aux raisonnements de l'expert à simuler. Nous détaillons une sélection d'exemples significatifs de cas où ces raisonnements interviennent. Nous en déduisons le niveau nécessaire d'expressivité des langages de représentations des connaissances à mettre en œuvre. Nous achevons alors de définir notre modèle conceptuel de métadonnées en y introduisant les nouveaux éléments, tels ceux utiles à la description du contexte de l'utilisateur, nécessaires à l'adaptation des modes d'emploi.

Au chapitre 4, nous effectuons les choix relatifs à l'implémentation de la base de métadonnées. Nous justifions le choix de l'architecture duale SI/SBC. Nous traitons les questions correspondant aux aspects "SI" de notre application d'abord, aux aspects "SBC" ensuite. Nous choisissons donc d'abord des langages pour construire des métadonnées structurées, conformes à notre modèle, persistantes et rendant aisée le développement de l'application Web présentée à l'utilisateur. Nous choisissons ensuite des langages de représentation des connaissances exploitables par des moteurs d'inférence.

Au chapitre 5, nous présentons l'architecture générale de notre application qui permet à l'utilisateur de rechercher, consulter, créer et modifier les métadonnées. Nous en décrivons en détail les aspects "SI" et "SBC". Nous montrons ensuite comment nous mettons en œuvre quelques-uns des exemples de raisonnement exposés au chapitre 3. Nous tentons enfin de cerner quelques-une des limites de notre l'application.

Au chapitre 6, nous présentons les parties de notre application dédiées à l'acquisition des métadonnées. Nous exposons la façon dont se déroule la saisie manuelle des métadonnées, puis nous détaillons comment s'effectue la saisie semi-automatique. Au-delà des simples questions relatives à l'interface de l'application, nous essayons de cerner les facteurs qui entravent l'acquisition des métadonnées. Cette réflexion est également un moyen d'évaluer l'adéquation de notre modèle aux connaissances que les experts souhaitent exprimer.

Chapitre 1

Les besoins d'informations sur les traitements

1.1 L'information géographique à l'IGN

Dans ce premier chapitre, nous précisons ce que recouvre pour nous la notion de traitement géographique. Nous présentons l'Institut Géographique National (IGN), cadre de notre travail (section 1.1). Les activités de l'IGN nécessitent l'emploi de traitements informatiques géographiques. Cela implique des besoins d'information. Notre but est d'y répondre. Section 1.2, nous exposons les différents types de traitements auxquels nous nous intéressons. Section 1.3, nous analysons les besoins d'informations concernant lesdits traitements. Nous esquissons section 1.4 les principales lignes de notre contribution au problème. L'objectif d'un modèle de métadonnées est justifié, celui d'une application suivant une approche relevant à la fois du domaine documentaire et de celui de la représentation des connaissances également.

1.1.1 L'information géographique et les missions de l'IGN

Denègre et Salgé [DS96] définissent l'information géographique comme un ensemble reliant :

- une information relative à un objet ou un phénomène du monde terrestre, décrit plus ou moins complètement par sa nature, son aspect et ses attributs (p. ex. un bâtiment décrit par sa hauteur, son nombre d'étages, sa fonction, etc.) ; cette description peut inclure des relations avec d'autres objets ou phénomènes (p. ex. ce bâtiment appartient à telle commune, etc.) ;
- et sa localisation sur la surface terrestre, décrite dans un système de référence explicite (p. ex. système de coordonnées ou adresse postale).

À cette définition peut être ajoutée la composante temporelle : on précise alors que la localisation de l'information est valable à un moment donné.

L'information géographique est au cœur des activités de l'Institut Géographique National. Le décret du 22 novembre 2004 indique en effet que la vocation de l'IGN est "de décrire, d'un point de vue géométrique et physique, la surface du territoire national et l'occupation de son sol, d'en faire toutes les représentations appropriées et de diffuser les informations correspondantes" [Off04]. L'IGN a ainsi pour missions principales d'assurer la production, la mise à jour, l'édition et la diffusion de données géographiques de référence en France. Cela suppose des capitaux humain et financier conséquents : en 2003, l'IGN employait plus de 1700 salariés et possédait un budget annuel d'environ 120 millions d'euros, financé à part égales par l'État et par les recettes commerciales [IGN04].

Derrière ces chiffres une réalité intéresse plus particulièrement notre travail de recherche. Il s'agit de ce que l'on pourrait appeler le *capital intellectuel* de l'IGN, à savoir la somme des "connaissances métier" des salariés anciens ou actuels, leurs compétences, leur savoir-faire. Les outils matériels et informatiques permettant d'acquérir, traiter et produire les photographies aériennes, images satellites, BD vecteur, cartes papiers, etc. constituent eux le *capital outil*.

Le capital intellectuel se manifeste par la réalisation de capital outil non volatile. Depuis la numérisation des données, il s'agit de traitements informatiques immatériels (au sens où leur existence ne dépend pas du type de support) qui sont développés, même si les outils situés en bout de chaîne du cycle de vie de l'information géographique (acquisition et, pour l'instant encore, restitution papier) restent nécessairement matériels : caméra numérique, scanner, imprimante laser, etc.

Le capital intellectuel n'est pas tout entier contenu dans les outils qu'il a permis de réaliser ; il réside également dans les connaissances *sur* ces outils. Les besoins nécessitant ces connaissances existent ; ils sont même très importants. Pour y répondre, il faut disposer de métadonnées. Cette problématique est le point de départ de notre travail de recherche.

Pour des organismes comme l'IGN, l'enjeu économique est de taille. Il concerne directement l'amélioration de la productivité, notamment par gain de temps, et la garantie de pérennité des traitements. L'existence de projets de "mémoire d'entreprise" dans plusieurs secteurs de l'industrie en est une autre illustration.

Une présentation succincte des laboratoires de recherche et des services de production de l'IGN se trouve en annexe E.

1.1.2 Cycle de vie de l'information géographique à l'IGN

Une illustration de la coordination des différents services de l'IGN est fournie par la figure 1.1. On y voit, à travers les étapes de la construction de la BD Pays, un exemple du cycle de vie de l'information géographique à l'IGN et des types de traitements qui lui sont appliqués.

De l'acquisition à partir de photos aériennes ou de cartes scannées à la constitution de la base de données topographique BD Pays au format vecteur, les processus mobilisent plusieurs corps de métiers. Les géodésiens, géomètres et photogrammètres interviennent lors de la saisie de l'information ; les cartographes et les géographes interviennent ensuite pour respectivement représenter et analyser l'espace géographique [Rua99].

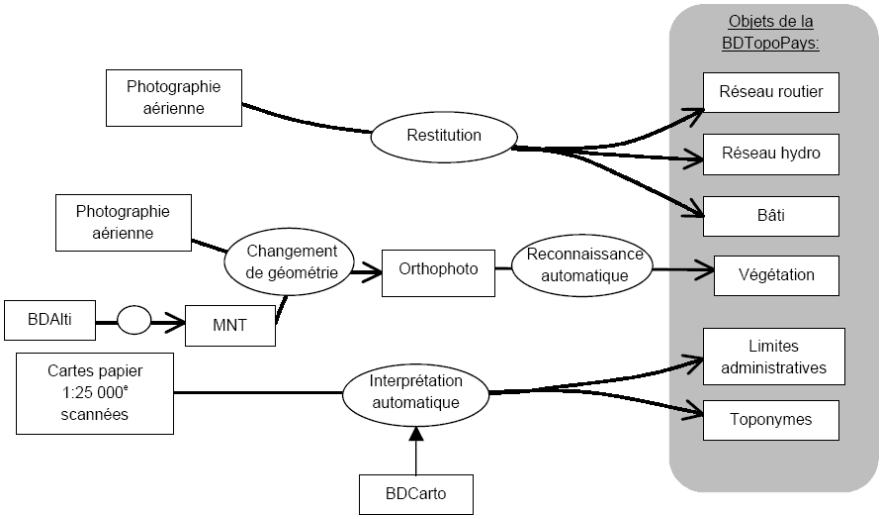


FIG. 1.1 – La construction de BD Pays (tiré de [Buc02])

Beaucoup d'exemples de ce mémoire sont liés aux traitements de généralisation cartographique. Il s'agit d'un des principaux domaines de recherche du laboratoire COGIT, dans lequel s'est déroulé notre thèse. La figure 1.2 montre en quoi consiste la généralisation cartographique, comparée à la simple réduction : la simplification de l'information permet de sauvegarder la lisibilité de la carte.

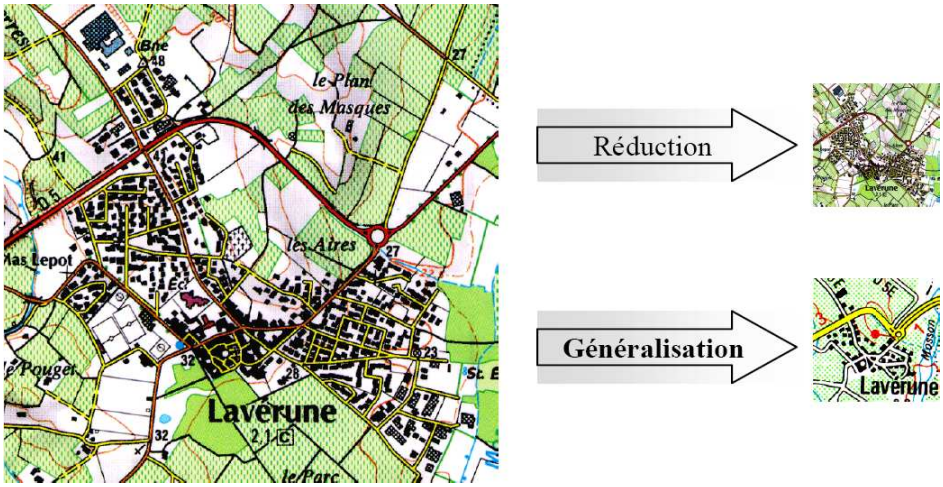


FIG. 1.2 – Réduction et généralisation de cartes du 1 : 25 000 au 1 : 100 000 (tiré de [Mus01])

Un autre exemple de traitements géographiques développé au laboratoire Cogit est celui des traitements d'appariement de bases de données géographiques (figure 1.3). Il s'agit d'établir la correspondance entre différents objets représentant la même réalité.

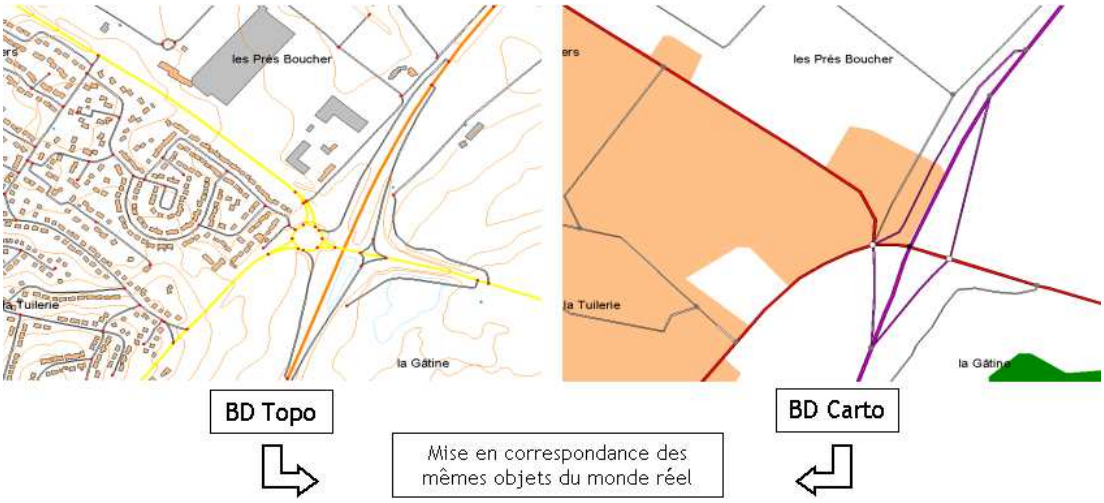


FIG. 1.3 – Appariement de bases de données géographiques (tiré de [She05])

1.2 Quelles ressources décrire ?

Deux types de ressources[1] apparaissent figure 1.1 : les données géographiques et les traitements qui leur sont appliqués. Notre travail a pour but de décrire les traitements. Ils possèdent deux caractéristiques saillantes. Ce sont des *traitements informatiques*, car les données manipulées sont numérisées. Ce sont des *traitements géographiques*, car les données manipulées sont géographiques.

Dans cette section nous détaillons ce qu'il en est exactement de ces traitements. Les *connaissances* d'expert sont également des ressources à décrire ; mais comme elles dépendent des besoins que l'on se propose de satisfaire nous nous y intéresserons section 1.3.

1.2.1 La notion de traitement

Avant de détailler les deux catégories de traitements mentionnées, définissons ce qu'on entend exactement par "traitement". Le terme mérite d'être précisé car il est communément employé dans plusieurs sens à la fois. Par exemple, l'expression "traitement de lissage d'une route" peut désigner indifféremment l'algorithme de lissage[2], le programme informatique qui l'implémente, ou le lissage lui-même.

Traitement : Dans ce mémoire, par convention, un "traitement" est un outil qui permet la transformation d'une information fournie en entrée et livrée en sortie. Comme nous choisissons de nous intéresser aux traitements informatiques, cette définition devrait s'appliquer aux ordinateurs dans leur ensemble. Nous considérerons en fait avant tout la partie logicielle (*software*) de ceux-ci. Les descriptions concernant la partie matérielle (*hardware*) n'interviendront que de façon marginale, lorsqu'elles seront susceptibles d'influer sur les aspects logiciels.

Par facilité de langage, nous dirons d'un traitement qu'il réalise une fonctionnalité. Pour être précis, puisque nous ne considérons que la partie logicielle de l'outil informatique, il faudrait plutôt dire qu'un traitement spécifie la façon de réaliser une fonctionnalité. Cette spécification peut être exprimée en langage informatique ou en pseudo-code.

Le point de vue qui vient d'être exposé exclu de notre objet d'étude une partie des instruments géographiques utilisés à l'IGN. Par exemple les scanners, les caméras numériques et les imprimantes laser sont en effet principalement constitués par leur dispositif matériel. Nous ne nous interdisons pas de les décrire, mais cela n'est pas notre objectif premier. On ne rencontrera ainsi pas de champs de description matérielle (poids et taille p. ex.) dans le modèle de métadonnées que nous allons définir. En revanche la partie logicielle de ces instruments pourra être décrite.

A priori, nous excluons également de notre étude les instruments non informatiques tels que loupes, les règles graduées, les sextants, les stéréo-restituteurs, etc.

1.2.2 Les traitements informatiques

L'informatique est souvent définie comme la science du traitement automatique de l'information par les ordinateurs. Les processeurs de ces derniers exécutent ce que nous appelons de façon

[1] Le terme "ressource" désigne dans ce mémoire tout ce qui peut se voir attribuer un identifiant. On retrouve cette idée dans la notion d'URI (Uniform Resource Identifier) [BFIM98]. Une ressource peut donc être n'importe quoi. Mais concrètement, dans notre contexte, si par exemple les programmes se verront attribuer un identifiant, ce ne sera pas le cas de leur date de création.

[2] Le lissage ("smoothing") est un traitement de généralisation cartographique qui a pour but de donner un aspect lisse aux objets à la géométrie linéaire auxquels il s'applique. Les virages peuvent être arrondis, par exemple.

générique les "traitements informatiques". Il en existe plusieurs variétés. Les noms des plus courantes sont : *programme, logiciel, application, script, macro, plug-in, add-on, fonction, procédure, template, méthode, classe, composant, package, librairie, module, plateforme, framework, service Web, opération, servlet, applet*, etc. À cette liste nous ajoutons les *algorithmes*, suite d'instructions non interprétable en l'état par une machine, mais pouvant l'être après implémentation dans un langage informatique. Les différents termes évoqués seront définis section 2.3 lors de la présentation de notre modèle de métadonnées ; nous ne voulons ici que donner un aperçu des ressources à décrire. Dans ce but, cherchons à caractériser les traitements. Beaucoup de points de vue sont possibles. Nous n'en présentons ci-après que quelques uns.

Taille et complexité des traitements

Certains traitements informatiques sont élémentaires. C'est le cas des opérations arithmétiques ou logiques de base et des interactions simples avec les périphériques (afficher un pixel sur le moniteur, lire ou écrire sur le disque, envoyer un flux d'octets sur un port, etc.). Par le passé, développer ou utiliser des traitements informatiques nécessitait de programmer en assembleur et donc de manipuler des fonctions de bas niveau. Ce n'est plus guère le cas aujourd'hui. Les langages de programmation "de haut niveau" fournissent des jeux d'instructions qui masquent les caractéristiques liées au matériel. Les nouveaux paradigmes de programmation, procéduraux, orientés objets, maintenant par composants et orientés aspects, tendent toujours plus à abstraire les traitements informatiques de la machine pour se rapprocher de la conception humaine de l'activité de programmation. La complexité des traitements est de plus en plus masquée à l'utilisateur. Par exemple, l'opération de jointure entre deux tables d'une base de données relationnelle peut être considérée, du point de vue de l'utilisateur, comme un traitement simple ; le placement automatique des toponymes d'une carte ou le détramage[3] comme des traitements complexes.

La complexité est habituellement définie comme la fonction qui lie la taille des données d'entrée avec le nombre moyen d'opérations élémentaires exécutées lors du traitement. La complexité peut alors être linéaire, logarithmique, polynomiale, exponentielle, factorielle, ... Cette acceptation du terme complexité n'est pas forcément liée au nombre de lignes de code qui constituent le traitement. Le temps processeur d'exécution est un critère classique de complexité de traitement, la quantité de mémoire nécessaire également. Avec l'avènement annoncé des services Web, d'autres critères devront peut-être être imaginés : le randonneur qui demande un calcul d'itinéraire à son système de navigation embarqué sollicite potentiellement un nombre insoupçonné de services Web (calcul de géopositionnement, requêtes à des bases de données géographiques et thématiques, calcul de graphe, programme de cryptage pour l'authentification, etc.).

Façon d'invoquer les traitements

Comme nous le verrons par la suite, les traitements s'adressent à des publics divers. Leur utilisation requiert des niveaux d'expertise variables. En particulier, certains s'invoquent via des interfaces graphiques ou des lignes de commandes (c'est le cas des logiciels) ; d'autres nécessitent l'écriture de programmes (c'est en général le cas des librairies de fonctions).

Dépendance des traitements à des environnements de travail

Un traitement informatique donné ne peut s'utiliser dans n'importe quel contexte. Les obstacles sont nombreux. Il faut disposer d'un système d'exploitation adéquat, parfois en plus de logiciels ou de librairies particuliers. Il faut aussi disposer de bons formats de données. Ces

[3]Les images imprimées sur papier sont formées de petits points de couleur, que l'on nomme la trame. Pour numériser une image provenant d'un journal, il faut demander au scanner de la détramer, c'est-à-dire de reconnaître globalement ces points comme des zones de couleur.

restrictions induisent, de fait, autant de catégories de traitements.

Question de licences

Du point de vue du droit, les traitements peuvent être commerciaux, propriétaires, libres, du domaine public, *copyleft*, *open source*, etc. Ces différents statuts – et d'autres, la liste n'est pas exhaustive – sont décrits sur le site du projet GNU[4].

Diverses licences précisent les conditions d'utilisation d'un point de vue légal. Les plus connues sont GPL (General Public License), LGPL (Lesser GPL), BSD (Berkeley Software Distribution), MPL (Mozilla Public License), etc. Le critère essentiel qui les distingue concerne la liberté plus ou moins grande d'intégrer les projets sous licence à des projets commerciaux [ATI02].

1.2.3 Les traitements informatiques géographiques

Les traitements informatiques auxquels nous nous intéressons manipulent des données géographiques. Ils sont principalement réunis dans des *Systèmes d'Information Géographique* (SIG), mais on peut également les trouver sous d'autres formes que nous allons répertorier. Nous incluons dans notre revue les logiciels de graphisme car ils sont utilisés, notamment, pour la cartographie ou la visualisation 3D de MNT. Ils ne peuvent néanmoins être qualifiés de géographiques, puisqu'en général ils n'exploitent pas l'aspect localisé des données qu'ils manipulent[5].

Les différentes formes des traitements informatiques géographiques

Les différentes formes des traitements informatiques géographiques que nous considérons sont : les *SIG*, *les logiciels de graphisme*, les *libraires de programmes et de fonctions*, les *services Web géographiques*, et les *algorithmes*.

Les SIG

Un SIG est un ensemble informatique de matériels, de logiciels, et de processus conçus pour permettre la collecte, la gestion, la manipulation, l'analyse, la modélisation et l'affichage de données à référence spatiale [Ele04].

Historiquement, les SIG sont apparus après l'informatisation de la production cartographique, qui a commencé à la fin des années 60. Cette informatisation a donné lieu à l'avènement d'une nouvelle technologie, nommée *géomatique*, définie comme l'ensemble des techniques de traitement informatique des données géographiques[Fra94] et dont un des buts premiers était l'automatisation de la cartographie.

Un SIG sert à répondre à cinq questions principales :

- *Où ?* recherche d'objets selon leurs caractéristiques (données attributaires)
- *Quoi ?* recherche de caractéristiques d'objets
- *Comment ?* comment sont spatialement répartis des objets
- *Quand ?* recherche de changements intervenus sur les données
- *Et si ?* définir en fonction de certaines hypothèses l'évolution du terrain, étude d'impact.

[4] `http://www.gnu.org/philosophy/categories.fr.html` (GNU est l'acronyme récursif de Gnu's Not Unix).

[5] Par exemple, la retouche d'une carte sous Photoshop ou la visualisation d'un fichier VRML ne contenant que la forme d'un objet et non son géoréférencement ne correspondent pas à proprement parler à des traitements géographiques (cf. définition de l'information géographique p. 5). Ainsi les spécifications des bases de données de l'IGN distinguent le niveau géodésique (positionnement sur la Terre à partir des coordonnées), et le niveau géométrique (localisation par des coordonnées) [IGN05].

Voici des exemples de ces questions :

- *"Où se trouvent les forêts d'Ile-de-France de plus de trente ans ?"*
- *"Quelle est la surface totale des forêts d'Ile-de-France ?"*
- *"Quel arrondissement de Paris est à égale distance du Bois de Vincennes, du Parc de la Courneuve, et du Bois de Boulogne ?"*
- *"Quand les tramways ont-ils investi les boulevards des maréchaux ?"*
- *"Quel impact aurait l'interdiction de la circulation automobile dans Paris sur le temps moyen de transport professionnel des franciliens ?"*

Les fonctionnalités offertes par les SIG sont couramment regroupées en cinq catégories nommées les "5 A" [DS96] (tab.1.1). La catégorie "Analyser" se décompose elle-même en plusieurs sous-catégories (tab.1.2).

Acquérir	Consiste à alimenter le SIG en données : numérisation, restitution, analyse spatiale, etc. Les fonctions d'acquisition consistent à entrer d'une part la forme des objets géographiques et d'autre part leurs attributs et relations. *Ex. : digitalisation, jointure de cartes bord à bord, reformatage.*
Archiver	Consiste à transférer les données de l'espace de travail vers l'espace d'archivage (disque dur). *Ex. : gestion, stockage, sécurité.*
Abstraire	Revient à concevoir un modèle qui organise les données par composants géométriques et par attributs descriptifs ainsi qu'à établir des relations entre les objets.
Analyser	Répondre aux requêtes de l'utilisateur et modifier les données. *voir tableau 1.2 pour le détail*
Afficher	Visualiser les données, percevoir les relations spatiales entre les objets, produire des cartes de façon automatique. *Ex. : Zoom (échelle +/-), généralisation cartographique, superposition de couches.*

Tab. 1.1 – Classification "5A" des fonctionnalités des SIG

Analyse spatiale	Interpolation d'information manquante, analyse multi-variables, obtention du centre d'un objet, analyse de la forme.
Recherche	Par thème Par région Par type (BD)
Analyse d'endroit	Buffer, corridor, overlay, polygones de Thiessen
Analyse de terrain (3D)	Pente, aspect, analyse d'écoulement des eaux, calcul des zones de visibilité
Distribution / voisinage des objets	Etendue (calcul d'aire) Proximité (calcul de distance) Calcul du plus proche voisin (calcul de graphe)
Mesure	Tout ce qui peut caractériser un objet géographique : direction, distance, aire, volume, poids, etc.

Tab. 1.2 – Classification "5A" des fonctionnalités des SIG – détail de la catégorie "Analyser" du tableau 1.1

Il est possible de distinguer trois types de SIG [Val04] :

- *Les SIG généralistes bureautiques* ont pour vocation essentielle l'import de données externes et leur analyse pour donner des cartes à insérer dans des rapports ou des présentations. Ils permettent bien sûr la modification de données géométriques ou descriptives mais ils ne disposent pas d'outils d'assurance qualité perfectionnés pour saisir des Bases

de Données complètes. Ils disposent d'outils de développement pour s'adapter à tout type d'application.
- *Les SIG généralistes de gestion* disposent des mêmes capacités que les SIG bureautiques, sont fréquemment moins conviviaux, mais disposent d'outils de modélisation beaucoup plus puissants, qui vont imposer des contraintes à la saisie et donc assurer une certaine qualité des données. Ces SIG vont également disposer de capacités client/serveur permettant à plusieurs personnes en réseau de travailler sur la même base de données à partir de postes informatiques distants. Ils disposent d'outils de développement pour s'adapter à tout type d'application.
- *Les SIG "métiers"* sont, dès le départ, très spécialisés, destinés à des métiers particuliers. Leur champ d'application est réduit mais ils sont souvent les seuls ou les meilleurs dans leur domaine. Ce sont néanmoins des SIG car ils possèdent les fonctionnalités "5 A" qui définissent les SIG.

Des librairies de programmes complémentaires peuvent être importées (les *plug-in* ou *add-on*) ou développées dans les langages éventuellement propriétaires des SIG. Par exemple, le plug-in CadReader permet d'importer des données DXF [6] dans le SIG Arcview .

Nous venons de décrire ce que font les SIG. Mais nous n'avons pas encore dit précisément ce qu'ils sont, à quelle famille de traitements informatiques ils appartiennent, et en quoi leur spécificité géographique les rend particuliers.

Comme les données géographiques sont stockées dans des bases de données, on peut être tenté de dire que les SIG sont une sorte de SGBD. En effet les SGBD (Système de Gestion de Base de Données) sont des logiciels qui permettent la gestion et l'accès à une base de données (BD), une BD étant un ensemble structuré et organisé de données permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour, interrogation)[7].

En fait, plusieurs types de systèmes sont à distinguer : les SGBD classiques, les SGBD spatiaux, et les SIG.

Un SGBD spatial est un SGBD doté d'une extension spatiale. Il permet de répondre à une requête telle que : *"quelles sont les rues qui intersectent la rue de Hayeps ?"*. Cette requête est spécifique car elle demande un calcul où intervient un *opérateur spatial* (à moins, bien sûr, que la BD interrogée contienne déjà les informations sur les intersections ; auquel cas les opérateurs de l'algèbre relationnelle classique suffisent). Il existe en fait de nombreuses relations spatiales implicites entre objets géographiques (cf. tab. 1.4). Il faut pouvoir les calculer. C'est ce que font les SGBD spatiaux. Cependant un SGBD spatial n'est pas un SIG ; pour le devenir des sur-couches logicielles doivent lui être adjointes afin de proposer les fonctionnalités "5 A".

Le tableau comparatif 1.3 montre que les SGBD relationnels classiques et les SGBD spatiaux se distinguent par le type d'objets manipulés, par les opérations qu'on leur applique lors des requêtes, et par la façon dont on y accède. Les techniques d'indexation, notamment, diffèrent car il est difficile de définir un critère d'ordre qui traduise la proximité spatiale [Sch02]. Ainsi les index B-tree[8] ne sont pas adaptés car leur mise en œuvre repose sur l'existence d'un ordre

[6] Drawing eXchange Format, format d'échange pour logiciels de CAO / DAO créé par Autodesk pour son logiciel Autocad.

[7] Définitions d'après `http://fr.wikipedia.org`. On pourra néanmoins objecter que les SIG ne possèdent en général pas toutes les fonctionnalités classiques des SGBD, par exemple les aspects partagés comprenant la gestion des accès concurrents.

[8] Une démonstration graphique du fonctionnement peut être trouvée sur `http://slady.net/java/bt/`.

total sur le domaine de la clé (2 objets dont les clés d'accès sont proches dans l'espace des clés sont proches en mémoire). C'est pourquoi les index spatiaux reposent sur d'autres types d'arbres (R-tree, quad-tree).

	SGBD relationnel classique	**SGBD spatial**
Données	entier, réel, texte	point, ligne, région
Prédicats et calculs	=, >, contient, ...	prédicats et calculs géométriques et topologiques
Manipulation	opérateurs d'algèbre relationnelle : sélection, projection, jointure. Fonctions : count, sum, ..	manipulation mono ou inter-thèmes
Liens entre objets	par jointures sur les clés	relations spatiales (souvent) implicites
Méthodes d'accès	index B-tree, hachage	index R-tree, quad-tree, grid-file, ..

TAB. 1.3 – Parallèle entre les SGBD relationnels classiques et les SGBD spatiaux (d'après [ZY00], cité par [Buc02] p. 28)

M. Scholl propose une classification des opérations spatiales en six catégories, combinaisons du nombre d'arguments et du type de la sortie [Sch01] :

sortie / **nb.arg.**	**booléen**	**scalaire**	**spatial**
1	test de l'existence d'une propriété spatiale d'un objet *convexité*	*calcul de longueur*	transformation topologique
2	prédicats spatiaux *intersecte, contient, adjacent*	*calcul de distance*	opérations ensemblistes

TAB. 1.4 – Classification des opérations spatiales (d'après [Sch01])

Au laboratoire COGIT les bases de données vecteurs telles que la BD Topo[9] sont stockées dans un SGBD Oracle 9[10]. Il lui est adjoint l'extension "Spatial analyst"[11], qui permet d'utiliser dans les requêtes des opérateurs spatiaux[12]. La possibilité de stocker et d'interroger des BD géographiques avec des opérateurs spatiaux ne constituent qu'une partie des fonctionnalités "5 A", ce n'est donc pas suffisant pour prétendre disposer d'un SIG. Nous n'obtenons ce dernier qu'en adjoignant une couche logicielle comprenant une interface graphique pour visualiser les données. Nous avons alors un ensemble de programmes qui se rapproche des SIG que l'on peut trouver dans le commerce.

Il existe différents types d'architectures parmi les SIG existants[13], variant selon le mode de représentation et de gestion de l'aspect géométrique (dimensions, coordonnées) des données. Scholl montre par exemple que tous les SIG ne couplent pas de la même façon données géométriques et données attributaires [Sch01]. Moyennant quelques nuances, on peut globalement considérer que les SIG sont des SGBD dotés d'une extension spatiale. La raison pour laquelle il peut

[9] La BD TOPO, ou Base de Données Topographiques, contient la description métrique 3D du territoire et de ses infrastructures.

[10] Les aspects payant et contraignant d'Oracle ont motivé l'étude du passage pour le COGIT à un SGBD gratuit et libre : PostgreSQL/PostGIS.

[11] développée par ESRI.

[12] SDO_DISTANCE, SDO_DISTANCE, SDO_POINTONSURFACE, SDO_CONVEXHULL, etc.

[13] Selon R.Vallée, on en recense en 2004 plus d'une soixantaine sur le marché [Val04].

paraître abusif de qualifier les SIG de SGBD est qu'ils n'en possèdent en fait pas toutes les fonctionnalités (par exemple la gestion des accès concurrents).

Les logiciels de graphisme

Les SIG permettent d'afficher les données géographiques. Ils comportent des fonctionnalités plus ou moins sophistiquées de symbolisation, de placement des toponymes, de gestion des contraintes graphiques, généralisation, etc. Néanmoins, pour la dernière étape de finition des cartes géographiques avant impression, des logiciels de graphismes sont utilisés. On peut citer Adobe Illustrator et Autocad Map, mais il en existe beaucoup d'autres. Ce ne sont pas des purs logiciels de dessin car ils manipulent des données vecteurs. C'est pourquoi ils sont rangés sous la catégorie CAO (Conception Assistée par Ordinateur) plutôt que DAO (Dessin Assisté par Ordinateur).

L'évolution des SIG, qui est d'offrir toujours plus de fonctionnalités cartographiques, pourrait un jour rendre inutile le passage à un logiciel de CAO. C'est en tout cas l'ambition affichée, par exemple, par la société ESRI qui présentait en 2005 à la Conférence Internationale de Cartographie la version 9 de son SIG Arcview.

En attendant la convergence possible entre SIG et logiciels de CAO, empruntons à J.Perice le tableau synthétisant leurs différences [Per95].

description \ type logiciel	**priorités**	**volume de données**	**unité de travail**	**mode de représentation des objets**	**lien graphiques / attributs**	**utilisation des attributs**	**outil**	**modélisation des données**
CAO/DAO	représentation graphique des données, fonctionnalités de dessin et de calculs techniques	petit volume	plan, carte	fixée au moment de leur description une seule façon de visualiser	type hiérarchique	liste d'objets et quantités, bordereaux de prix-métrés et avant-métrés	aide à la conception	pas indispensable
SIG	organisation informatique et structuration géométrique des données, fonctionnalités de gestion et d'analyse	gros volume	territoire	pas de lien entre description et représentation graphique. plusieurs possibilités de représentation	type relationnel	requêtes multi-critères-visualisation d'attributs-analyse spatiale	aide à la décision	requise au niveau géométrique (topologie) et sémantique (classe et couches d'objets)

Tab. 1.5 – Comparaison des SIG et des logiciels de CAO/DAO (d'après [Per95])

À l'IGN des logiciels de PAO (Publication Assistée par Ordinateur) sont également utilisés. Par exemple le SPI[14] utilise XPress sous MacIntosh.

Les librairies de programmes et de fonctions

Pour leurs besoins spécifiques, les laboratoires et les services de productions développent leurs propres programmes, liés ou non à des SIG. Dans un but de modularité et réutilisation, des librairies de fonctions sont utilisées. Souvent les librairies de bas niveau ont déjà été développées hors de l'IGN, par des organisations commerciales ou non, sous licences libres ou non. Dans tous les cas, plus une librairie est utilisée, plus elle est éprouvée et donc plus on est fondé à lui accorder confiance[15].

Au COGIT, la plupart des fonctions géométriques 2D de bas niveau (calcul de longueur, d'aire, d'intersection, de buffer, etc.) utilisent une bibliothèque open-source codée en Java, ap-

[14] Service de Photogravure et d'Impression, cf. p. 242.

[15] On peut faire le parallèle avec le critère de scientificité énoncé par Popper : plus une théorie est testée sans être réfutée, plus on est fondé à lui accorder confiance. Cela suppose que ladite théorie soit réfutable (l'adverbe falsifiable est également employé) [Cha87].

pelée JTS[16]. Pour la 3D, c'est l'API Java 3D qui est utilisée. Elle repose elle-même sur les API d'OpenGL ou de DirectX.

Les services Web géographiques

Dans le monde de l'information géographique, il est actuellement beaucoup question d'utilisation de composants accessibles via des services Web. Günter et Müller soutenaient par exemple en 1998 que l'avenir des SIG passait par leur transformation de "logiciel client" en "client léger", *i.e.* du déplacement des grosses applications SIG des machines clientes vers des serveurs qui ne fourniraient alors que les fonctionnalités demandées à un instant donné [GM97]. Sept ans plus tard, l'intérêt pour les services Web semble bien se confirmer si l'on en juge par le nombre d'articles consacrés au sujet dans les conférences. On lira par exemple les propositions de Vogele pour améliorer les infrastructures de données géographiques à l'aide des services Web [Vog04].

La tendance au développement des services Web n'est pas spécifique au domaine géographique, comme l'illustre cet extrait d'article paru dans la presse :

> "Microsoft risque de se faire défier, dans son cœur de métier, par les géants venus de l'Internet, tels Yahoo ! et surtout Google. (...) Ces portails veulent devenir des fournisseurs de "services logiciels" : les services (traitement de texte, tableur, messagerie, moteur de recherche, comparateur mais aussi gestion de fichiers musicaux, photo, vidéo, etc.) seraient accessibles en ligne, gratuitement ou par abonnement, au lieu d'être achetés et stockés sur les ordinateurs. D'un usage plus souple, n'exigeant plus des machines très puissantes ni de lourdes procédures de remise à jour, ces logiciels sont accessibles [par] tout appareil connecté et permettent de court-circuiter Microsoft. Un vrai danger pour cette entreprise qui tire toute sa puissance de son monopole sur les logiciels de base, équipant presque tous les ordinateurs de la planète." (G. Macke, "Microsoft, la vie au-delà du PC", Le Monde daté du 03/12/2005)

Pour la géographie spécifiquement, une explication de l'intérêt pour les services Web réside dans l'essor de l'informatique nomade. L'objectif est de répondre à des requêtes telles que "trouver le restaurant le plus proche". La réponse peut être apportée par un seul service Web mais peut aussi l'être par des composants séparés (opération de localisation, base de données des commerces, opération de sélection, calcul de plus court chemin) [LdB02].

Ce type de scénarii suppose l'interopérabilité des services Web, que l'ISO TC211[17] et Open GIS espèrent faciliter grâce à l'établissement de normes.

À l'heure actuelle il n'existe quasiment pas de services Web disponibles à l'IGN, mais des chercheurs y travaillent. Sur le Web mondial, dans le domaine géographique, les services Web qui existent sont pour l'instant avant tout des fournisseurs de données : cartes, images satellites, etc.

Les algorithmes

Dans notre contexte les traitements sous formes d'algorithmes sont courants. Au sein du laboratoire COGIT par exemple, plusieurs algorithmes ont été développés dans le cadre du projet européen Agent pour la généralisation cartographique. De façon plus générale, beaucoup de rapports de stages et de projets, de mémoires de thèse et d'articles de conférences présentent des algorithmes. Il est important de les indexer et de les décrire, au même titre que les traitements implémentés.

[16] Java Topology Suite `http://www.geotools.org/`.

[17] International Organization for Standardization, Technical Committee 211 (groupe de travail pour l'information géographique), `http://www.isotc211.org`.

Classification des traitements informatiques géographiques

Nous venons de donner un aperçu des diverses formes de traitements à décrire. Nous pouvons maintenant proposer quelques-unes des autres façons possibles de classifier les traitements.

Classification des traitements en fonction du type de données manipulées

Il existe deux grands types de données utilisés pour représenter l'information géographique : les types *raster* et *vecteur*.

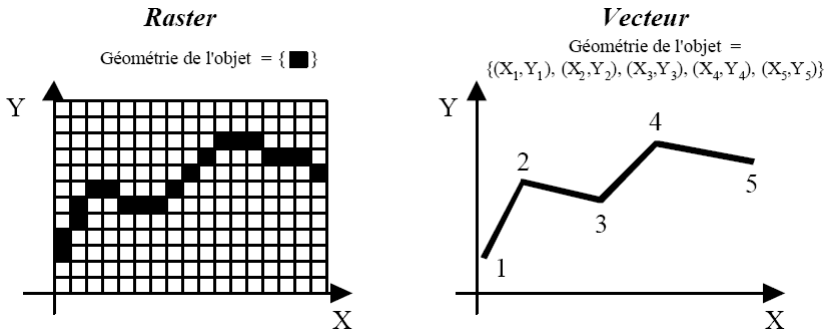


FIG. 1.4 – Types de données vecteur et raster (tiré de [Mus01])

Tous les traitements utilisés dans le domaine géographique ne manipulent pas des données raster ou vecteur. Certains par exemple manipulent des données purement numériques. La dichotomie raster / vecteur reste cependant un des critères principaux pour distinguer les traitements géographiques.

Classification des traitements en fonction des points de vue "bases de données" ou de "analyse spatiale"

Une autre façon de distinguer les traitements géographiques est relevée par J-P. Cheylan. Il considère qu'il existe deux points de vue différents concernant les traitement géographiques : celui des spécialistes des bases de données qui considèrent que les fonctionnalités géographiques sont des opérations spécifiques de BD (car les données sur lesquelles elles opèrent sont souvent contenues dans des BD) ; et celui des spécialistes de l'analyse spatiale qui ne s'intéressent qu'aux fonctionnalités rentrant dans le cadre de leur domaine [Che92].

Classification des traitements en fonction des types de fonctionnalités réalisées

Certains auteurs proposent des classifications des traitements géographiques, et plus particulièrement les SIG, reposant sur [Bor02] :

- *les thèmes des données manipulées* : agriculture, aménagement, défense, environnement, etc. Comme exemples de besoins on peut citer : l'identification des parcelles nécessitant de l'engrais, le développement urbain, la cartographie des voies en fonction de leur largeur, la cartographie du bruit à Paris, etc.
- *les territoires des données* : commune, département, région, pays, continent. Des exemples de besoins sont l'étude de la démographie, des variables météorologiques, etc.
- *les usages des données* : inventaire, observatoire, étude, aide à la décision, etc. Un exemple de besoins est le choix du lieu d'implantation d'une nouvelle pharmacie compte tenu de la répartition de celles déjà existantes.

1.2.4 Les traitements informatiques géographiques à l'IGN

L'IGN doit construire, entretenir et diffuser des produits d'information géographiques, à savoir, principalement, des bases de données vecteurs, des bases de données images et des cartes papiers. Pour cela des traitements informatiques sont développés dans les laboratoires de recherche et dans les services de production. Certains traitements sont aussi directement achetés dans le commerce.

L'exemple du laboratoire COGIT, illustration de la diversité des traitements

Dans les laboratoires, il s'agit de répondre à des objectifs de recherche en amont de la production. Nous prenons ici l'exemple du laboratoire COGIT pour deux raisons. D'abord parce que c'est en son sein que s'est déroulé pendant trois ans mon travail de thèse ; ensuite parce que le COGIT fournit une bonne illustration des problèmes propres aux traitements informatiques du domaine géographique évoqués section 1.2.3.

En effet on voit cohabiter au COGIT deux plateformes de développement (Lamps2 et GeOxygene), associées à divers SGBD (Gothic, Oracle et PostgreSQL/PostGIS). Les modalités de chargements de données sont parfois complexes, notamment lorsque le passage d'une modélisation relationnelle à une modélisation objets doit être réalisé. Plusieurs systèmes d'exploitation cohabitent également (Linux, Windows, OS9), tout comme les langages (Java, Lull[18], C, ADA) et les formats de données (shape, GML, SVG, ..). De plus, à ces problèmes purement informatiques s'ajoutent des considérations sur les aspects gratuit et/ou libre des logiciels et librairies utilisés.

Le couple Lamps2/Gothic sert au COGIT au développement des traitements de généralisation, dans la continuité du projet AGENT. Ainsi par exemple des traitements de lissage des routes sont développés : on simplifie leur représentation en ne conservant que les virages les plus prononcés, les portions ne contenant que des faibles courbes devenant droites. Des traitements de simplification des formes des bâtiments sont également développés.

L'autre principal environnement de développement utilisé au COGIT repose sur la plateforme GeOxygene. Elle est prévue pour accueillir les principaux développements du COGIT autres que la généralisation. Par exemple, les traitements d'appariement établissent des correspondances entre différentes bases de données. Des objets géographiques distincts peuvent en effet représenter la même partie du monde réel. Par exemple, un échangeur d'autoroute peut être représenté par un nœud dans une base et par l'ensemble de ses bretelles dans une autre base. Des traitements s'appuyant sur la géométrie, la sémantique et la topologie permettent d'établir certaines de ces correspondances.

Voici, selon ses créateurs Thierry Badard et Arnaud Braun, ce qu'est GeOxygene [BB03] :

> "GeOxygene vise à fournir un cadre ouvert de développement, compatible avec les spécifications édictées par l'Open Geospatial Consortium (OGC) et l'ISO, pour la conception et le déploiement d'applications s'appuyant sur des données géographiques (SIG). Il s'agit d'une contribution "open source" du laboratoire COGIT (`http://recherche.ign.fr`) de l'IGN (Institut Géographique National, `http://www.ign.fr`). GeOxygene est diffusé selon les termes de la licence LGPL (GNU Lesser General Public License).
>
> GeOxygene est basé sur Java et les technologies open source. Il met à disposition des utilisateurs un modèle de données objet et extensible (permettant la modélisation des objets géographiques, de la géométrie, de la topologie et des métadonnées) compatible avec les spécifications de l'OGC et les standards de l'ISO concernant l'information géographique. Il est prévu dans un futur proche que GeOxygene implémente les interfaces Java développées par le projet GeoAPI (`http://geoapi.sourceforge.net`).
>
> Les données sont stockées dans un SGBD relationnel pour permettre un accès rapide et sûr au système. Néanmoins, les utilisateurs n'ont pas à manipuler les données au travers de SQL : Ils modélisent leurs applications en UML et les codent en Java. Le "mapping"

[18] Laser-Scan User Language.

> entre les environnements objet et relationnel est assuré par un composant open source, OJB. Les fichiers de "mapping" pour le stockage flexible des objets géographiques dans Oracle et PostGIS sont fournis."

Aperçu sommaire du fonctionnement du laboratoire MATIS

Le laboratoire MATIS travaille sur des données raster : photos aériennes et satellites, cartes scannées, etc. Les traitements d'image développés peuvent consister par exemple à distinguer automatiquement les routes du paysage photographié. Des bibliothèques de fonctions C et C++ "noyau" sont disponibles. Les chercheurs peuvent les utiliser et les faire évoluer de façon collaborative grâce à CVS (Concurrent Versioning System), logiciel de gestion de versions de projets informatiques utilisés dans plusieurs parties de l'IGN (notamment, également, au COGIT).

Le MATIS fournit un exemple parmi d'autres de l'utilisation de traitements *open source*. Cela illustre l'apparition de communautés de développeurs rendues possibles grâce à Internet. Néanmoins partager les traitements informatiques est une chose ; partager les connaissances nécessaires à leur réutilisation en est une autre. À cet égard, les laboratoires sont un contexte favorable au partage de traitements : les chercheurs se connaissent, ils se parlent.

Aperçu sommaire du fonctionnement de traitements au sein des services de production

Dans les services de production de l'IGN, les traitements informatiques peuvent être développés à la demande d'un service commercial ou de la direction technique, qui décident de l'opportunité de la réalisation et mettent au besoin en place un projet de développement spécifique. Le résultat produit peut être un projet dont le résultat est un cahier de spécifications d'un produit, ou du code se présentant sous forme de librairies de fonctions ou de classes objets, associées ou non à un logiciel préexistant. Ces développements répondent à des besoins qui ne peuvent être satisfaits par aucun SIG, ni par les programmes disponibles sur le Web, ou bien lorsque les SIG adéquats existent mais que l'on désire ne pas en être dépendant (licence, format de données propriétaire).

Les développements peuvent prendre diverses formes, indépendantes ou non des logiciels existants. Notons que certains ne comportent pas de traitements géographiques : par exemple, un développement peut consister en une interface graphique destinée à faciliter l'accès à des fonctions existantes. Ainsi, par exemple, une bibliothèque d'*add-on* a été écrite en 2001 pour le logiciel Datadraw3. Les traitements réalisés simplifient la forme des bâtiments dans le cadre du processus de généralisation (*add-on* pour la dilatation de formes, rectangulation d'angles, etc.).

1.3 Quels besoins ?

Nous venons d'effectuer un tour d'horizon des traitements dont la description est l'objet de notre travail. Voyons à présent quels sont les besoins que nous nous proposons de satisfaire. Nous commençons par préciser à quel public notre travail est destiné. Nous décrivons ensuite les types de besoins auxquels ledit public est confronté. Il apparaît alors nécessaire de représenter différents types de *connaissances*. Nous les détaillons. Des exemples de besoins d'utilisateurs permettent de se former une idée concrète du problème.

Concernant notre méthode d'investigation, signalons que nous avons diffusé deux questionnaires aux utilisateurs de notre application d'accès aux métadonnées. Les questionnaires, reproduits en annexe p. 231, ont été diffusés en juin 2003 et avril 2004 au sein du laboratoire COGIT.

1.3.1 Les utilisateurs et les développeurs

Les personnes qui travaillent dans les laboratoires de recherche et les services de production de l'IGN se trouvent en général amenées à jouer alternativement les rôles d'utilisateur et de

développeur de traitements. Dans d'autres parties de l'IGN comme le service commercial ou l'ENSG, on trouvera essentiellement des utilisateurs de traitements, plus ou moins experts.

En sus des catégories utilisateur et développeur, la charte logicielle de l'IGN définit celle d'opérateur [IGN90]. Le but de ces distinctions est de fournir des documentations adaptées aux trois publics auxquels elles sont destinées :

- un *utilisateur* est censé être simplement intéressé par le mode de fonctionnement général et la connaissance de l'existence des fonctionnalités des logiciels ;
- un *opérateur* est appelé à utiliser les fonctionnalités des logiciels ;
- un *développeur* est concerné par la programmation des logiciels.

La distinction entre utilisateur et opérateur peut au premier abord ne pas sembler très claire. En fait, un utilisateur peut, par exemple, être un responsable qui a simplement besoin de savoir ce que peut faire un SIG pour assigner des tâches aux opérateurs, qui, eux, se serviront effectivement dudit SIG.

Parmi les utilisateurs de traitements géographiques, on peut aussi distinguer les utilisateurs de données et les producteurs de données. Il y a là clairement deux publics dont les besoins en terme de traitements diffèrent. On rencontre ces deux types de publics au sein de l'IGN.

Les profils des lecteurs de métadonnées

D'après la définition initiale de notre sujet de thèse, l'application attendue du travail de recherche consiste en un serveur de métadonnées accessible via l'intranet de l'IGN. Nous pouvons donc raisonnablement supposer que tout lecteur de métadonnées possède un niveau de connaissance géographique et informatique minimum. Un contexte de métadonnées pour l'enseignement à des débutants complets, par exemple, aurait impliqué d'autres choix pour le contenu de l'aide à apporter.

Le niveau de connaissance des destinataires de métadonnées est cependant très variable. Il constitue pour nous un des éléments du contexte d'utilisation d'un traitement. Nous verrons qu'un de nos objectifs est d'apporter une aide adaptée à ce contexte.

Du point de vue des connaissances possédées, l'ensemble des lecteurs potentiels de métadonnées est grand. Nous aurions pu définir des partitions au sein de cet ensemble, en vue de "personnaliser" les métadonnées ou les modalités de leur consultation. La charte IGN considère les trois catégories sus-citées ; le clivage entre géomètres, spécialistes du traitement d'image, cartographes, informaticiens, etc. aurait aussi pu être établi. Nous avons en fait choisi de ne considérer de façon spécifique qu'une seule frontière : celle qui sépare

- les *développeurs* capables de programmer,
- et les *utilisateurs* qui en sont incapables (ou non disposés à le faire)[19].

Cette séparation se concrétisera par une différenciation des types de modes d'emploi.

Dans la suite de ce mémoire, le terme *utilisateur* pourra être employé pour désigner les personnes qui utilisent l'application de consultation des métadonnées. Ces personnes ont en effet besoin de rechercher, connaître et utiliser les traitements[20].

[19] Les *utilisateurs* de notre définition englobent donc les *utilisateurs* et les *opérateurs* de la nomenclature de la charte IGN.

[20] Pour leur part, [Gro00] (cité par [Buc02] p.34) distinguent trois niveaux d'utilisation de métadonnées :
- *découverte* : savoir si un traitement existe, s'il est accessible,
- *exploration* : savoir si un traitement convient à un besoin,
- *exploitation* : savoir comment utiliser un traitement.

Dans la phase de découverte, l'utilisateur est à la recherche d'un traitement ; dans les phases exploration et exploitation, il en a trouvé un et il cherche à mieux le connaître.

1.3.2 Rechercher les traitements

Un utilisateur cherche un ou plusieurs traitements. Ce peut être en vue de satisfaire un besoin précis et bien défini, ou au contraire vague. Voici des exemples de requêtes.

- *Où sont disponibles les programmes d'appariement ?*
- *Quels algorithmes de détection de contour ont été développés à l'IGN en 1995 ?*
- *Quels SIG possèdent la fonctionnalité "buffer" ?*
- *Quels sont les algorithmes de calcul de flux sur un réseau ?*
- *Quels sont les traitements utilisables sur les objets "bâtiments" ?*
- *Quels sont les programmes du COGIT qui utilisent l'API GeOxygene ?*

On devine que certaines requêtes sont plus complexes que d'autres. Plutôt qu'une simple sélection dans la base de métadonnées, ces requêtes vont demander de mener des *raisonnements*. En effet leurs réponses ne vont pas toujours être explicitement présentes dans la base de métadonnées. Mais elles pourront en être dérivées. Cette considération ne concerne pas l'utilisateur, c'est pourquoi nous reviendrons plus tard sur la question de la façon d'apporter les réponses. Contentons-nous pour le moment de répertorier quelques requêtes *a priori* complexes.

- *Quel SIG est le plus adapté pour une étude sur la démographie ?*
- *J'ai des données de la BD GeoRoute, que puis-je en faire ?*
- *Quelles sont les classes Java qui utilisent des objets de la classe* `GeomPrim` *(ou d'une de ses sous-classes).*
- *Je cherche un logiciel gratuit de visualisation de données VRML qui fonctionne sous Linux et puisse traiter un fichier de 5000 objets (sachant que mon PC est équipé d'une mémoire vive de 256 Mo).*
- *Je désire avoir un aperçu comparatif des possibilités des SIG Arcview et Geoconcept.*

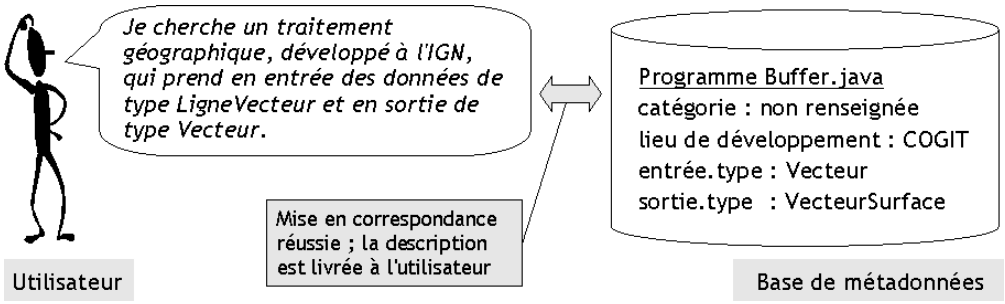


FIG. 1.5 – ER 1 : Recherche de traitements – mise en correspondance entre requête utilisateur et description de traitement

Parmi les requêtes nécessitant *a priori* un raisonnement (cela dépend en fait du contenu de la base de métadonnées), choisissons-en une en particulier qui nous servira d'exemple type dans la suite du mémoire. Nous la nommons ER 1 (pour Exemple de Raisonnement numéro 1). La figure 1.5 la présente.

1.3.3 Connaître les traitements

L'analyse des requêtes typiques exprimées par les utilisateurs montre que pour un traitement donné les besoins d'information portent sur cinq thèmes principaux : les métadonnées qui l'identifient (nom, date, auteur, etc.), "ce qu'il fait", "comment il fonctionne", "comment l'utiliser" et "quelle évaluation en est faite".

Voici un exemple de requêtes pour chacun des cinq thèmes en question.

- *Quel chercheur a développé le plus de traitements au laboratoire COGIT en 2005 ?*

- *Quels sont les requêtes topologiques permises par Arcview 8 ?*
- *Sur quelle théorie mathématique repose l'algorithme Accordéon ?*
- *Comment faire un changement de projection Lambert 2 sous Geoconcept ?*
- *Le programme de détramage "planches méres" est-il rapide ?*

Outre l'identification des cinq thèmes susdits, l'enquête auprès des utilisateurs a permis de révéler des besoins spécifiques au domaine géographique : par exemple pour comprendre ce que fait un traitement, il est utile de fournir des illustrations graphiques sous forme d'échantillons des données, ainsi qu'une description de l'évolution des propriétés des objets géographiques avant et après traitement. Une partie de ces besoins avaient déjà été identifiés par P. Michaux lors d'un stage de DESS au laboratoire COGIT sur le catalogage des traitements de généralisation [Mic03]. Un besoin important est également celui concernant sur les modes d'emploi ; nous allons y prêter une attention toute particulière.

1.3.4 Utiliser les traitements

Une des questions les plus couramment posées est probablement *"comment utiliser ce traitement ?"*. C'est également une des questions auxquelles il est le plus difficile de répondre. D'abord parce qu'elle mobilise de nombreuses connaissances, souvent tacites et non liées directement au traitement, ensuite parce que la réponse dépend du contexte d'utilisation. La figure 1.6 montre un demande d'information de mode d'emploi d'un traitement dont l'utilisation dépend du contexte.

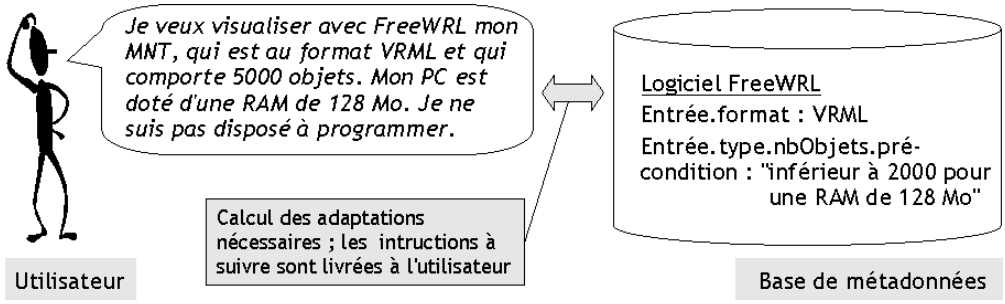


FIG. 1.6 – ER 3 : Adaptation de mode d'emploi – mise en correspondance entre contexte utilisateur et contexte de traitement

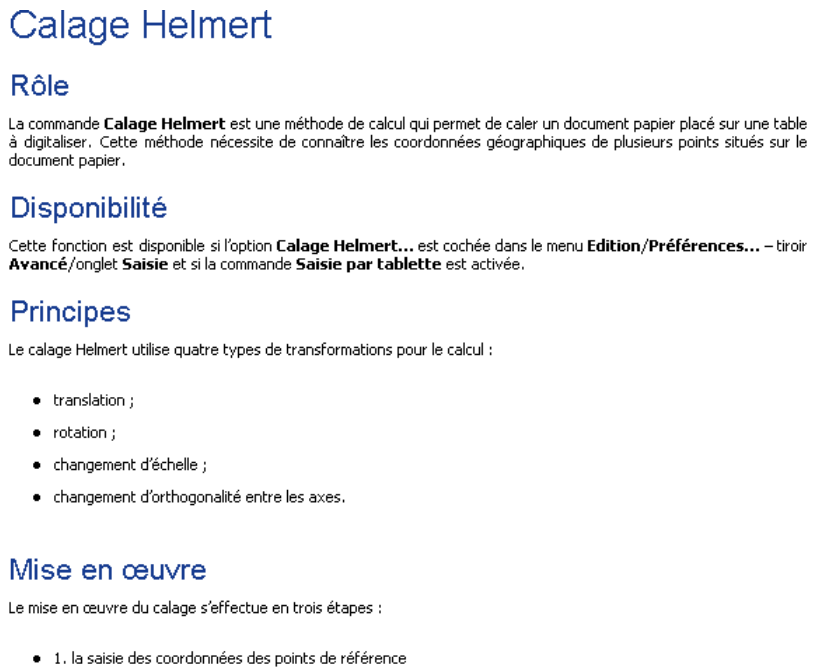
Parfois l'utilisateur invoque les traitements via l'interface graphique d'un logiciel ; d'autres fois il passe par le biais d'un programme qu'il doit alors développer. La figure 1.9 p. 26 montre un exemple de besoin où les deux types d'invocation sont possibles. De façon générale, il apparaît nécessaire de distinguer deux types de modes d'emploi : ceux destinés aux utilisateurs non-développeurs, et ceux destinés aux développeurs.

Les modes d'emploi destinés aux utilisateurs non développeurs

Les modes d'emploi des logiciels destinés aux utilisateurs existent. Plusieurs sortes de documentations les décrivent[21]. Pour être en mesure de concevoir une amélioration éventuelle de l'aide qu'elles apportent déjà, essayons de cerner quelques-unes des difficultés qui posent problème à l'utilisateur.

La première difficulté réside dans le rôle joué par les *connaissances tacites* sur lesquelles nous allons revenir. Une autre difficulté tient aux préconditions entre les actions à effectuer. Comme les connaissances tacites, les préconditions sont délicates à décrire (cf. par exemple figures 1.7 et 1.8 les nombreuses instructions préalables à l'invocation de la commande "Calage Helmert).

[21] Nous en discutons au chapitre 2, p. 49.



Calage Helmert

Rôle

La commande **Calage Helmert** est une méthode de calcul qui permet de caler un document papier placé sur une table à digitaliser. Cette méthode nécessite de connaître les coordonnées géographiques de plusieurs points situés sur le document papier.

Disponibilité

Cette fonction est disponible si l'option **Calage Helmert...** est cochée dans le menu **Edition/Préférences...** – tiroir **Avancé**/onglet **Saisie** et si la commande **Saisie par tablette** est activée.

Principes

Le calage Helmert utilise quatre types de transformations pour le calcul :

- translation ;
- rotation ;
- changement d'échelle ;
- changement d'orthogonalité entre les axes.

Mise en œuvre

Le mise en œuvre du calage s'effectue en trois étapes :

- 1. la saisie des coordonnées des points de référence

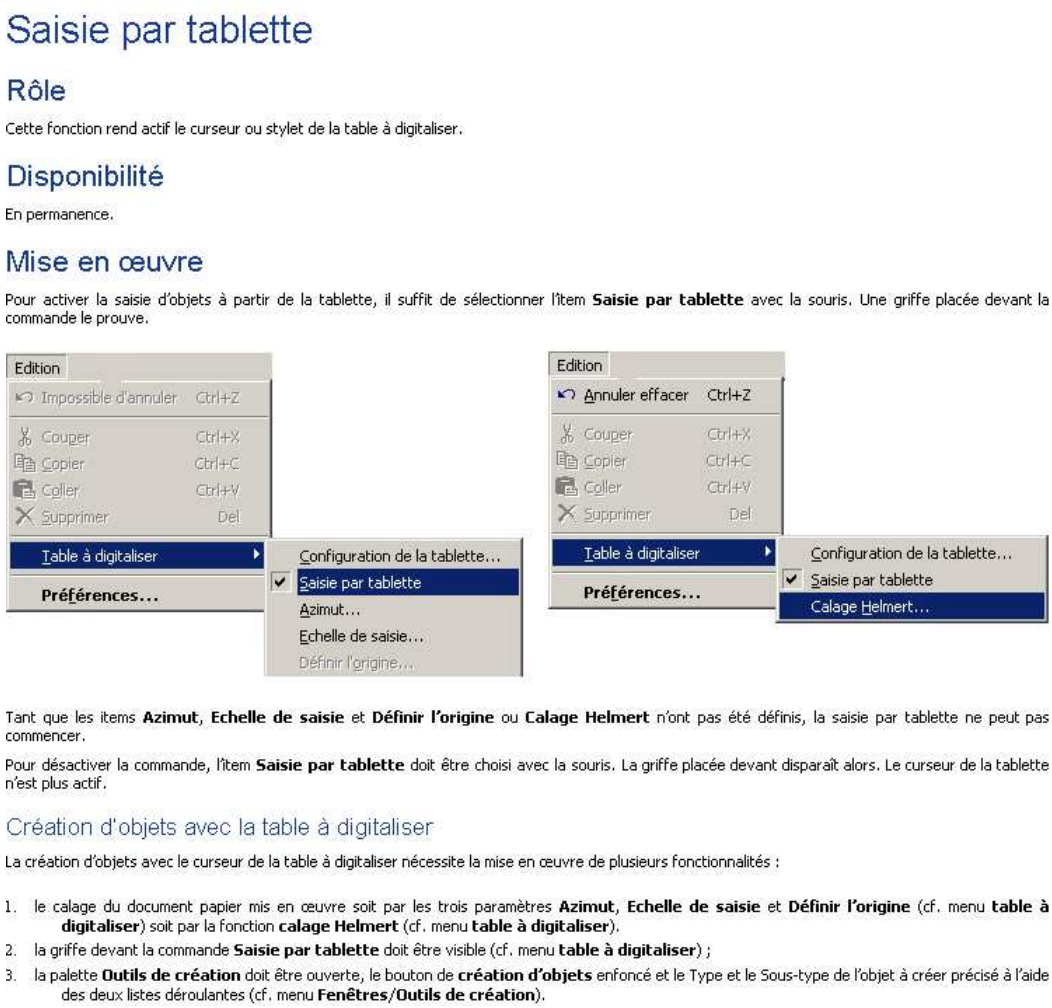
FIG. 1.7 – Aide de Geoconcept 5.0 – Calage Helmert (extrait partiel de la page d'aide)

Peut-être davantage que pour les autres types de logiciels, utiliser un SIG nécessite souvent de la patience. Une fonction quelconque peut en effet rarement s'exécuter avant un nombre important d'étapes préliminaires : conversion de format, ouverture de projet, importation des données dans le SIG, projection, sélection des objets à traiter, actions parfois subtiles provoquant l'apparition des menus contextuels désirés. Les figures 1.7 et 1.8 montrent des extraits de deux pages d'aide du SIG Geoconcept 5.0. On voit que l'obtention des menus contextuels adéquats et l'activation de leurs entrées grisées nécessitent toute une suite d'actions. Cela demande une certaine habitude de la part de l'utilisateur – ou une certaine dose d'intuition (qui est une forme de connaissances tacites).

Par ailleurs, une des préconditions d'utilisation des SIG les plus courantes est la conversion de format de données. Il existe un grand nombre de ces formats, quasiment un par éditeur de SIG. Cette multiplicité s'explique souvent par des raisons de concurrence et de stratégie commerciale, et non pour des raisons techniques. Les utilisateurs se trouvent souvent captifs d'un SIG particulier. Ils se heurtent à des problèmes de compatibilité. Pour leur part, les fournisseurs de données sont contraints de décliner leur catalogue suivant les divers formats. L'IGN propose ainsi les jeux de données de la BD Topo en pas moins de cinq formats : SHP/SHX/DBF 2D et 3D (SHaPefile, SHapefile indeX et DataBase File pour les SIG de la famille Arcview), MIF/MID (pour MapInfo), GCM/GCR (GeoConcept Model/Ressource, pour Geoconcept), DXF 2D et 3D (Drawing eXchange Format, format d'échange pour logiciels de CAO / DAO créé par Autodesk pour son logiciel Autocad), Édigéo (norme d'échange de données géographiques définie par l'AFNOR, et utilisée entre autres par les services de l'État tels l'IGN et la direction générale des impôts.) [IGN03].

Face à cette situation, des problèmes de décision se posent aux utilisateurs. Les connaissances nécessaires pour y répondre concernent plusieurs SIG à la fois. C'est pourquoi elles sont davantage fournies par l'expérience de la pratique plutôt que par les modes d'emploi existants, généralement centrés sur un SIG particulier.

Outre l'expression des préconditions d'utilisation, un autre besoin d'information difficile



Saisie par tablette

Rôle

Cette fonction rend actif le curseur ou stylet de la table à digitaliser.

Disponibilité

En permanence.

Mise en œuvre

Pour activer la saisie d'objets à partir de la tablette, il suffit de sélectionner l'item **Saisie par tablette** avec la souris. Une griffe placée devant la commande le prouve.

Tant que les items **Azimut**, **Echelle de saisie** et **Définir l'origine** ou **Calage Helmert** n'ont pas été définis, la saisie par tablette ne peut pas commencer.

Pour désactiver la commande, l'item **Saisie par tablette** doit être choisi avec la souris. La griffe placée devant disparaît alors. Le curseur de la tablette n'est plus actif.

Création d'objets avec la table à digitaliser

La création d'objets avec le curseur de la table à digitaliser nécessite la mise en œuvre de plusieurs fonctionnalités :

1. le calage du document papier mis en œuvre soit par les trois paramètres **Azimut**, **Echelle de saisie** et **Définir l'origine** (cf. menu **table à digitaliser**) soit par la fonction **calage Helmert** (cf. menu **table à digitaliser**).
2. la griffe devant la commande **Saisie par tablette** doit être visible (cf. menu **table à digitaliser**) ;
3. la palette **Outils de création** doit être ouverte, le bouton de **création d'objets** enfoncé et le Type et le Sous-type de l'objet à créer précisé à l'aide des deux listes déroulantes (cf. menu **Fenêtres/Outils de création**).

FIG. 1.8 – Aide de Geoconcept 5.0 – Saisie par tablette (précondition au Calage Helmert FIG. 1.7)

à exprimer concerne la description de l'agencement des étapes d'utilisation. On relève dans les modes d'emploi beaucoup de listes énumérées spécifiant explicitement des séquencements d'étapes ; en revanche, les alternatives et les instructions conditionnelles et itératives (si ... alors, tant que...) sont généralement indiquées en langue naturelle. Cela exige une lecture forcément plus attentive de la part de l'utilisateur.

Complexité du paramétrage et spécificité géographique

La géographie est un domaine où les traitements sont souvent difficiles à paramétrer. On peut penser que cela est lié au fait que les algorithmes dépendent de beaucoup de facteurs.

C'est effectivement ce à quoi on peut être confronté lorsque l'on tente de représenter le monde réel (phénomènes physiques ou, en l'occurrence dans notre contexte, espace géographique), potentiellement plus complexe que des mondes "artificiels" tels que, par exemple, celui des données bancaires (pour prendre un domaine où les traitements informatiques sont nombreux).

Outre le nombre de facteurs, une autre source de complexité et de difficultés dans la tâche de paramétrage est celle des cas où des heuristiques sont nécessaires pour guider les traitements. Cette fois, les traitements de mondes "artificiels" peuvent être concernés.

La complexité du paramétrage des traitements de généralisation cartographique tient probablement au nombre des facteurs en jeu et, de façon liée, au rôle important de l'heuristique. Le comportement des algorithmes ne peut être prévu à l'avance de façon exacte. Ce comportement doit donc être guidé par autre chose que des instructions pré-écrites. Pour ce type de problèmes des approches basées sur les agents (objets guidés par des buts et des contraintes) ont été développées en intelligence artificielle. Les paramètres portent alors sur les critères de satisfaction des buts et des contraintes qui déterminent le déroulement du traitement.

Les modes d'emploi destinés aux développeurs

Les modes d'emploi des librairies de fonctions ne différent pas fondamentalement de ceux des logiciels, sinon que davantage de choix de mise en œuvre sont laissés à la discrétion de l'utilisateur-développeur. À l'IGN, et particulièrement dans les laboratoires de recherche, la spécificité des besoins implique souvent de se placer dans un contexte de programmation. Plus généralement, dans le domaine de l'information géographique, le besoin de développer de nouveaux traitements et de ré-utiliser les anciens est constant. En effet les progrès technologiques permettent l'acquisition de nouveaux types de données[22].

Parallèlement au renouveau constant des applications, leur conception tend à devenir de plus en plus modulaire. Aujourd'hui, développer un traitement, c'est avant tout assembler du code déjà écrit, l'apport du développeur se limitant même parfois à produire du code "colle" pour adapter les différents composants de son application. Le guide de l'utilisateur de l'IDE Eclipse va même jusqu'à pronostiquer que les développeurs vont désormais passer plus de temps à lire du code qu'à en écrire. Dans ces conditions, on comprend bien l'importance de la documentation et plus généralement des métadonnées des traitements.

Bien sûr, la (ré-)utilisation des traitements n'est possible que si les auteurs les rendent disponibles. C'est pourquoi la tendance que nous venons d'évoquer est très liée au succès de l'*open source* et d'Internet.

Certains problèmes sont spécifiques au domaine géographique. A. Braun, ingénieur de l'IGN qui a participé au développement de la plateforme géographique GeOxygene, pose ce diagnostic

[22] Par exemple, l'apparition de nouveaux capteurs aéroportés et satellitaires a conduit à l'obtention d'images numériques à des résolutions submétriques, qui permettent d'envisager de nouveaux modes d'étude des environnements naturels et humains [Mar05].

([Bra03])[23] :

> "Le développement d'applications liées à l'information géographique se heurte à de nombreux problèmes :
> – L'absence d'interopérabilité entre les modèles de données des différents SIG commerciaux, malgré les efforts de standardisation de l'ISO[24] et de l'OpenGIS[25]. Une application développée avec un modèle non standard peut ne pas être réutilisable.
> – Les langages de programmation liés aux SIG commerciaux sont bien souvent des langages propriétaires ; ainsi le partage de code entre les différents SIG est impossible, et les utilisateurs sont très dépendants des évolutions technologiques de l'éditeur.
> – Sans des compléments onéreux, les SIG commerciaux ne sont pas ouverts sur le Web. Et si de tels compléments permettent l'accès aux données à distance, l'appel de processus à distance n'est quant à lui pas toujours possible (notion de service Web de traitement).
> – Les SIG ne sont pas des SGBD purs, et des problèmes résolus par les SGBD ne sont pas toujours adressés par les SIG (accès concurrent, sécurité, etc.).
>
> Pour surmonter ces problèmes, des technologies ont émergé en génie logiciel : des langages orienté-objets ouverts sur le Web (comme Java), des techniques d'analyse et de conception orienté-objets, basées sur la réutilisabilité des composants (comme UML), des SGBD relationnels intégrant des fonctionnalités objets, et permettant le stockage de l'information géographique (comme Oracle et [PostgreSQL]), des langages structurés pour l'échange d'information sur les réseaux (comme XML), et des technologies de service Web permettant la description et l'appel de procédures à distance dans des environnements informatiques hétérogènes et distribués (comme SOAP et WSDL)".

Concrètement, si l'on prend l'exemple du mode d'emploi de la plateforme GeOxygene – justement disponible en Open Source[26] –, on se rend compte que la tâche pour l'utilisateur est ardue, ne serait-ce que pour saisir l'architecture globale qu'il est nécessaire de mettre en place avant de manipuler effectivement les objets Java représentant les données géographiques (la difficulté de la chose réside essentiellement dans la réalisation du *mapping* objet-relationnel).

Cet exemple confirme un besoin déjà pressenti : celui de représenter et de relier des connaissances de niveaux de généralité différents.

[23] Les organismes ISO et OpenGIS ainsi que les langages UML, SOAP et WSDL mentionnés dans cette citation sont décrits au chapitre 2, pages 44, 55, 61 et 60.

[24] L'ISO (International Organization for Standardization) est l'organisme international pour la normalisation. Il comporte un comité technique dédié à l'information géographique (TC 211).

[25] L'OpenGIS Consortium (OGC) : groupement d'organismes actifs dans les technologies de l'information géographique, visant à rendre interopérables les systèmes géoinformatiques, via des interfaces communes définies dans des spécifications techniques.

[26] `http://geoapi.sourceforge.net`

1.3.5 Accéder aux connaissances de l'expert

La recherche, le développement et l'utilisation de traitements informatiques géographiques nécessitent la mobilisation d'ensembles étendus de connaissances. Des connaissances informatiques ou géographiques, contextuelles ou générales, théoriques ou empiriques, explicites ou tacites... toutes ces connaissances que possède l'expert et qui manquent au novice. Illustrons cela avec l'exemple d'un utilisateur désirant développer un programme de visualisation de MNT[27](cf. fig. 1.9).

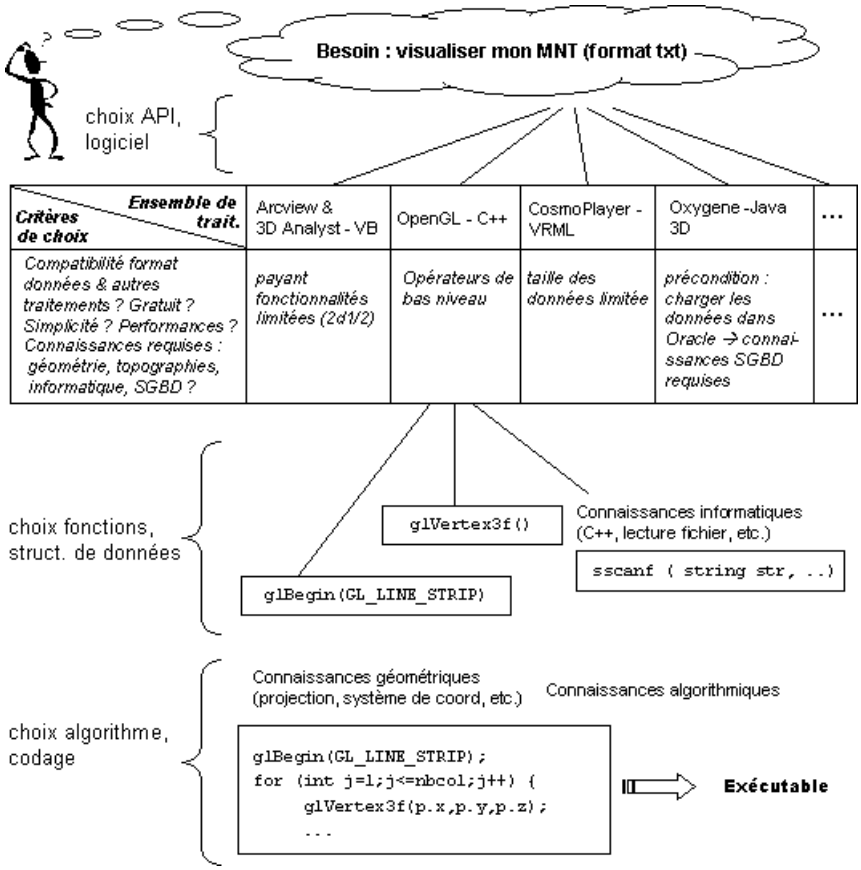


FIG. 1.9 – Connaissances requises pour le développement d'un visualisateur de MNT

L'utilisateur novice qui ne disposerait que d'un simple catalogue de descriptions de traitements ne pourrait prendre la chaîne de décisions qui aboutit à la réalisation de son besoin. La raison en est que, prises isolément, les informations contenues dans les descriptions de traitements ne sont pas des connaissances. Elles le deviennent seulement lorsqu'elles sont reliées à d'autres connaissances qui permettent de décider de l'action à effectuer pour réaliser le besoin considéré.

B. Bucher, dans sa thèse traitant d'une problématique proche de la nôtre – l'aide aux utilisateurs pour l'accès aux données géographique [Buc02] –, cite [SAA+00] pour soutenir l'idée que les informations sont des connaissances lorsque l'on sait quelle action y associer. Cette idée est également présente, par exemple, dans [Dup99] :

"connaître, c'est effectuer, sur des représentations, des manipulations réglées"

[27] Un Modèle Numérique de Terrain est une représentation numérique du relief sous forme d'un ensemble de données d'échantillonnage (points cotés, courbes de niveau, lignes directrices) et d'une fonction d'interpolation permettant d'obtenir une altitude en n'importe quel point [Rou04].

et dans [Kay97] :

> *"il n'y a présomption de connaissance que si la faculté d'utiliser des informations à bon escient est attestée" ; "les connaissances sont des données qui influencent le déroulement de processus".*

En amont des connaissances et des informations, les données "brutes", c'est-à-dire non interprétées, constituent pour [SAA+00] une troisième catégorie à distinguer (fig. 1.10).

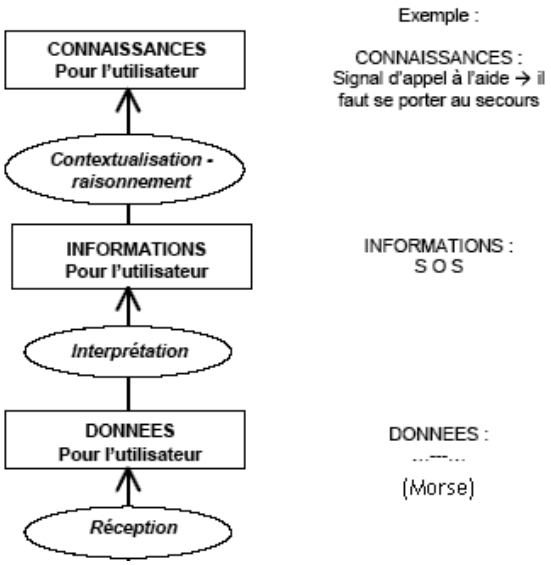


FIG. 1.10 – Données, informations et connaissances (d'après [SAA+00], cité par [Buc02])

Les auteurs de la figure 1.10 nomment "interprétation" le passage de *données* à *informations*. Nous verrons section 3.2.3 que nous attribuons un autre sens à ce terme. Pour nous le passage en question mériterait plutôt d'être désigné par "décodage asémantique".

La notion de connaissance est au carrefour de nombreux domaines : philosophie, psychologie, sciences cognitives, I.A. (dont l'ingénierie des connaissances est une branche). Cela explique que l'on trouve de nombreuses classifications de types de connaissances dans la littérature. Nous en retenons ici trois.

Connaissances opératoires et connaissances factuelles

La première classification retenue, mentionnée entre autres par [Kor03], distingue six formes de connaissance (tab.1.6). Elles ont pour particularité d'être potentiellement opératoires : elles peuvent donner lieu à un calcul ou une inférence. Précisons ce qu'on entend par là. D'après la définition adoptée, le propre d'une connaissance est de pouvoir être impliquée dans un raisonnement menant à une action ou à une nouvelle connaissance. Les formes de connaissances listées par [Kor03], par exemple, portent en elles la spécification d'un calcul. Elles sont, si l'on peut dire, dotées nativement d'une sémantique opérationnelle. Nous verrons par la suite que les connaissances de forme 1, 2 et 3 sont représentables dans les langages d'ontologies basés sur les logiques de description, tandis que les formes 5 et 6 nécessitent des langages à base de règles. L'opérationnalisation des connaissances de la 4ème forme requiert, elle, un calcul qui pourrait s'effectuer par un langage de programmation procédural traditionnel.

1– Structure[28]	Un thème est un ensemble d'objets géographiques. Le package `spatial.geomprim` est une partie de la plate-forme GeOxygene.
2– Classification[29]	La fonctionnalité de lissage est une sorte de fonctionnalité de généralisation. La classe `GM_Curve` est une sous-classe de `GM_Primitive`.
3– Définition	Des données vecteur "spaghettis" sont des données où la topologie n'est pas représentée.
4– Loi, axiome	La distance euclidienne d entre deux points $P_1(x_1, y_1)$ et $P_2(x_2, y_2)$ est $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
5– Règle d'expertise	Si les données ne s'affichent pas convenablement dans un SIG, alors suspecter une erreur dans la configuration de la projection ou du géoréférencement.
6– Stratégie	Si les bâtiments à généraliser se trouvent en zone urbaine, alors commencer par un traitement d'élimination.

TAB. 1.6 – Une classification des connaissances (d'après [Kor03])

La classification qui suit montre qu'on a aussi besoin de logique modale, floue, non monotone (révision de connaissance).

Connaissances universelles, évolutives, incertaines, vagues, typiques, sous-déterminées

Nous avons affaire à des connaissances (tab. 1.7) :

1– universelles éternelles	Un quadrilatère est un polygone à quatre côtés.
2– évolutives	Le COGIT utilise le SGBD Oracle pour stocker les données vecteurs.
3– incertaines	Les données 3D utilisées au COGIT proviennent souvent de la BD Topo.
4– vagues	Avec un PC équipé d'un processeur 1 GHz, l'exécution du programme DétectionCarrefours prend environ 30 secondes sur un jeu de données 1000 objets.
5– typiques	Un moyen d'ouvrir un fichier est de double-cliquer sur son nom. Les données aux formats *shape* ne possèdent pas d'informations sur la topologie.
6– sous-déterminées	Les carrefours de 50m de diamètre sont représentés dans la BD Topo par des objets surfaciques[30].

TAB. 1.7 – Une classification des connaissances proposée par D. Kayser [Kay97]

Les connaissances typiques se caractérisent par l'existence de contre exemples. En l'occurrence pour le premier exemple (tab.1.7, l. 5), il existe au moins trois cas de contre-exemples : quand on visualise une page HTML un clic suffit ; le système d'exploitation peut être configuré pour qu'un clic suffise ; si l'on double-clique sur le nom du fichier et non sur l'icône qui le symbolise, et que l'intervalle de temps entre les deux clics dépasse le délai prédéfini (≈ 500 ms par

[28] Organisation des concepts selon la relation de méréologie (partie-tout).

[29] Organisation des concepts selon la relation de subsumption (généralisation-spécialisation). Mais une classification peut aussi se contenter d'énumérer des concepts, comme c'est le cas dans le présent tableau.

[30] Exemple tiré de la thèse de N. Gesbert au COGIT sur les spécifications des BD géographiques [Ges05]

défaut), alors le fichier ne s'ouvre pas mais son nom se met en sur-brillance pour être renommé. Ces contre-exemples fournissent de très bons exemples de connaissances tacites, catégorie sur laquelle nous allons revenir.

Pour le deuxième exemple de connaissance typique, les données au format shape peuvent être accompagnées de tables dans lesquelles figurent des relations topologiques. Par exemple, un réseau routier pourra comporter une table "intersection".

Connaissances "imparfaites"

Les connaissances peuvent également être "imparfaites", au sens où elles peuvent être (tab.1.8) :

1– incomplètes	On connaît le nom des paramètres du programme détramage.exe, mais pas leur rôle.
2– imprécises	Détramage a été développé entre 2001 et 2005.
3– vagues	Il faut environ 3 jours à un informaticien moyen connaissant le C++ pour développer un visualisateur de MNT avec la librairie OpenGL.
4– incertaines	Il est peut-être possible de faire tenir sur une disquette de 1.4 Mo l'extrait de la BD Géoroute du XIème arrondissement de Paris.
5– inconsistantes	Le programme Buffer.java implémente un algorithme dont les types de données d'entrée ne correspondent pas.

TAB. 1.8 – Une classification des connaissances "imparfaites"

Connaissances explicites et connaissances tacites

Les notions de connaissances tacites et connaissances explicites ont fait l'objet de plusieurs définitions dans le domaine de la gestion de connaissances (*knowledge management*) [Erm03][Ban00] ; elles résistent néanmoins à une définition nette. Selon I. Dostaler, dont l'objectif des travaux est d'explorer les applications pratiques des connaissances tacites dans les organisations : "malgré leur caractère très empirique, les connaissances tacites apparaissent le plus souvent comme un savoir caché, mal défini, presque inaccessible, voire comme une simple hypothèse de recherche". "Tacite" vient d'ailleurs du latin *tacitum*, qui signifie ce qui est secret, caché ou mystérieux [BD00]. De façon concordante, M. Polanyi considère que les connaissances explicites sont issues de l'observation empirique, et sont verbalisables. Au contraire, les connaissances tacites sont difficiles à exprimer et relèvent davantage de l'intuition ou du savoir-faire, comme le montre la figure 1.11 empruntée à M. Grundstein [Gru95].

Nous dirons donc d'une connaissance qu'elle est explicite si son existence est identifiée, et s'il existe un support quelconque qui en permet la transmission ; sinon elle est tacite. Cette définition est hautement subjective. En effet une même connaissance pourra être qualifiée de l'une ou l'autre des façons, suivant que la représentation considérée aura, ou non, su la mettre à jour.

Il est toujours intéressant de remonter *a posteriori* le fil des raisonnements menés inconsciemment et de recenser les connaissances tacites qui ont été implicitement utilisées. Notre objectif d'acquérir les connaissances d'expert pour l'utilisation des traitements demande que l'on s'attelle à ces deux tâches. Plusieurs travaux de représentation des connaissances traitent de la question.

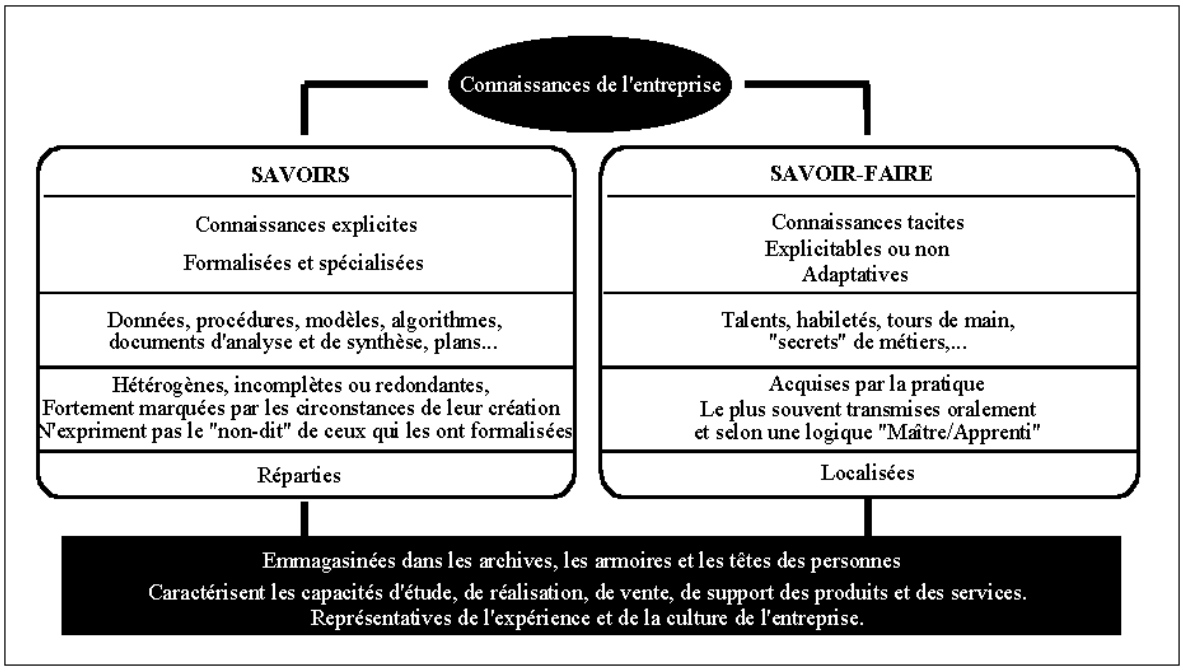


FIG. 1.11 – Connaissances tacites et connaissances explicites (extrait de [Gru95])

D. Kayser a exhibé un échantillon de connaissances nécessaires à la compréhension de la vie scolaire (*"à l'école, les élèves sont regroupés en classe", "en général, un élève reste un an dans la même classe", "pour chaque période d'enseignement, les élèves d'une classe se trouvent généralement dans une même salle, etc."* [Kay97]). D. Kayser a choisi ces exemples pour montrer le caractère incomplet des connaissances présentes dans les dictionnaires, et par conséquent l'importance du rôle des "méta-connaissances" implicitement supposées chez le lecteur.

P. Boyer, dans son essai sur les origines des religions, prend lui l'exemple concret de raisonnement nécessaire à la compréhension d'un récit très simple impliquant un enfant, un chien, un voleur et un policier. Un enfant joue avec un chien. Au cours du jeu le chien heurte le voleur. Le policier assiste à la scène et intervient. Le voleur se méprend sur le motif de l'intervention et, se croyant démasqué, se rend [Boy02]. Nous comprenons l'histoire parce que nous possédons des connaissances sur la psychologie des protagonistes. Ces connaissances sont tacites, elles relèvent selon l'auteur de la *psychologie intuitive*. Certaines connaissances de *physique intuitive* sont également mobilisées : par exemple, le heurt du chien et du voleur provoque la chute de ce dernier. Sans la connaissance tacite de ce lien de causalité, on ne peut comprendre le récit. En l'occurrence, il s'agit de savoir que deux objets matériels solides ne peuvent se traverser. On trouve dans [Pia70] beaucoup d'exemples de ce type de connaissances. L'auteur, J. Piaget, mène diverses expériences avec des enfants pour tester le caractère éventuellement inné des notions de physique intuitive : conservation des propriétés (longueur, poids, nombre) d'objets que l'on déplace ou transforme, notions de temps et d'espace, ou encore connaissances logico-mathématiques (transitivité des relations d'ordre ou d'égalité, par exemple).

Les connaissances des exemples qui viennent d'être cités présentent un caractère très général. Il peut sembler inenvisageable de les recenser de façon absolument exhaustive[31] (à supposer que cela présente un intérêt). Le cas des traitements informatiques, et plus particulièrement celui de leur utilisation, est peut-être différent. En effet on a affaire là à des artefacts. Les machines qui exécutent les programmes sont des automates finis. Cela signifie que le nombre d'interactions

[31] Une telle tentative a cependant été effectuée dans le milieu des années 80 avec le projet CYC de D. Lenat `http://www.cyc.com/cyc/opencyc/overview`.

possibles avec l'utilisateur est *a priori* limité, donc que les connaissances qu'un système d'aide doit représenter est également limité.

On peut également faire remarquer, si l'on s'en tient strictement au critère de l'existence de leur description, qu'aucune de ces connaissances[32] n'est vraiment tacite : elles sont en effet inscrites dans le code des programmes. Afin de diminuer l'effort d'apprentissage demandé à l'utilisateur, les éditeurs de logiciels tentent de concevoir des interfaces graphiques "intuitives". Les paradigmes de fenêtre, de clic de souris, de menu, de *drag and drop*, etc. sont ainsi apparus.

Notre modèle de métadonnées doit permettre de décrire ces connaissances de base. Ceci dit, nous ne nous sommes pas spécialement préoccupé de les recenser, car nous les avons supposées acquises par le public visé dans notre contexte de travail. Nous avons en revanche apporté davantage d'attention aux connaissances requises par les traitements spécifiquement géographiques. Nous verrons plus loin des exemples précis d'utilisation de traitements mettant en jeu des connaissances explicites. Regardons pour l'instant l'interface graphique d'un SIG (figure 1.12), en l'occurrence celle d'Arcview 3.1 (les données géographiques visualisées représentent l'espérance de vie des populations de quelques pays). Cet exemple nous semble intéressant car la compréhension de cette interface requiert à la fois la connaissance des paradigmes d'IHM généralistes, de connaissances relatives aux bases de données, et de connaissances géographiques.

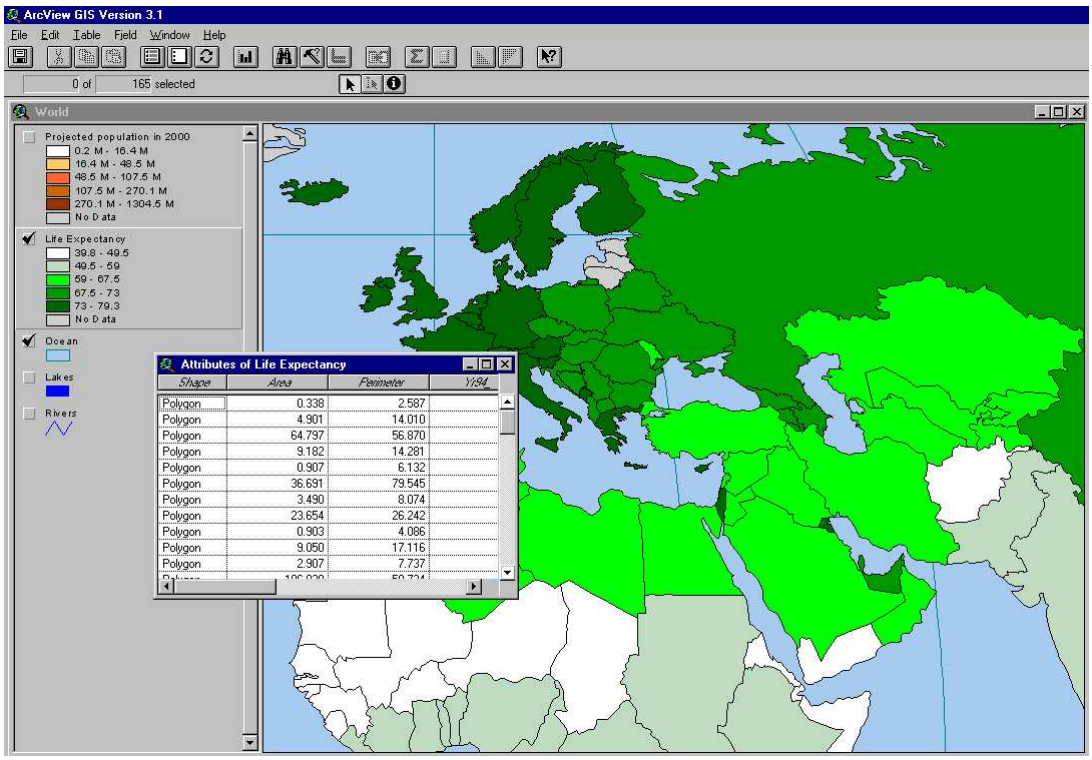


FIG. 1.12 – Interface graphique d'Arcview 3.1 – vues cartographique et tabulaire des données de quelques pays symbolisés en fonction de l'espérance de vie des populations

La liste des connaissances du tableau 1.9 n'est pas complète ; notre objectif est simplement d'initier le travail d'acquisition. Il incombera ensuite aux experts de l'IGN d'enrichir la base de métadonnées. Pour cela ils pourront utiliser l'application que nous avons développée.

Dans l'optique d'une aide à l'utilisateur efficace, lister les connaissances ne suffit pas. Il faut

[32] Quelles sont les actions possibles sur l'interface d'un logiciel, quels sont leurs effets (cf. la première catégorie de connaissances du tableau 1.9).

Catégorie	Concept	Description et explication
Connaissances générales sur les IHM des logiciels	layer	Dans un contexte de dessin : couche d'objets graphiques, qui peut être affichée ou masquée, être sélectionné ou non, etc.
	menu contextuel	Les menus et leurs entrées dépendent de l'espace de travail courant ou des objets sélectionnés.
	surbrillance	Indique la sélection.
	verrouillage	Parfois, dans la présentation tabulaire de données, l'italique indique le verrouillage des données.
	zone cliquable	Boutons, liens, menus, zones de la carte, etc. dont le clic ou le double-clic déclenche une fonction du logiciel.
Connaissances sur les bases de données relationnelles	table	Collection d'enregistrements.
	enregistrement	Ensemble d'informations concernant un même sujet. Un enregistrement correspond à une ligne d'une table.
	attribut, champ	Une information d'un enregistrement. A une colonne d'une table correspond un type d'attribut.
	sélection	Opération qui permet de sélectionner une partie des enregistrements d'une table.
Connaissances sur la cartographie	projection cartographique	Transposition d'une portion de l'ellipsoïde de référence géodésique représentant la surface terrestre, sur une surface plane, à l'aide d'un modèle mathématique.
	échelle et niveau de détail	Rapport entre les distances réelles et les distances apparentes sur une carte. Cependant, comme les données ont une représentation géométrique plus ou moins *détaillée*, elles ne sont adaptées qu'à une certaine plage d'échelles. On parle donc dans le cas des BD géographiques de niveau de détail géométrique[Ges05].
	sémiologie des couleurs	Traditionnellement, les couleurs foncées traduisent des valeurs hautes (pour un attribut donné).
Connaissances sur les SIG	thème	Ensemble des informations attributaires et géométriques relatives à une portion de l'espace géographique, plus éventuellement la description de la symbolisation cartographique associée (en vue cartographique d'un SIG, un thème est une sorte de layer).
	fonctionnalités	"5 A" (cf. tab. 1.1 p. 11)
Connaissances spécifiques à Arcview	thème Arcview	Informations géométriques : SHP (shape, ou forme) ; attributaires : DBF (database file), index géométriques : SHX (SHP index), légende : AVL (ArcView Legend).
	zone cliquable Arcview	Pour éditer les légendes, double-cliquer sur les palettes de couleurs qui les symbolisent (sorte de zone cliquable).

Tab. 1.9 – Quelques connaissances requises pour la compréhension de l'interface du SIG Arcview 3.1. Les connaissances sont classées par catégories de spécificité croissante, de haut en bas.

aussi les organiser, les hiérarchiser, voire, si l'on souhaite simuler une partie du raisonnement de l'expert, les opérationnaliser.

Le tableau 1.9 montre que la compréhension d'une interface de SIG nécessite des connaissances relevant de plusieurs domaines à la fois. Ne pouvant toutes les représenter, les manuels qui accompagnent les SIG laissent nécessairement dans l'ombre certaines explications ; le lecteur est supposé pouvoir s'en passer ou être capable d'aller les chercher ailleurs. Les documentations existantes sont donc le fruit d'un inévitable compromis entre précision et concision, compromis dépendant du niveau d'expertise supposé du public visé. Ceci dit, la représentation des connaissances n'est pas qu'une question de quantité et de niveau de détail. Pour preuve, on observe que souvent l'expert est capable de faire comprendre des notions au novice en lui fournissant des explications absentes des documentations. Prenons un exemple. Le thème d'un jeu de données géographique est similaire aux couches de données graphiques que l'on trouve dans divers logiciels dotés de fonctions de dessin, comme Photoshop ou PowerPoint. L'expert bon pédagogue fait remarquer cette similarité au novice. De façon tacite, ce dernier déduit alors, sans qu'il soit besoin de les lui énumérer, quelles sont les propriétés des thèmes (superposition, ordre, etc.)[33].

Les manuels utilisateurs, dont nous étudierons les caractéristiques au chapitre 2, recourent parfois à l'analogie. Mais ils ne le font pas de façon systématique. Surtout, ils ne peuvent le faire *à la demande*. Face à ce constat, un objectif d'un système d'aide basé que les métadonnées peut être l'exhibition de *prototypes* pour chaque famille de concepts. En l'occurrence, la description d'un SIG – Arcview p.ex. – peut être reliée à celle de la famille des logiciels utilisant des couches de données graphiques dont un prototype est MS PowerPoint. Allons plus loin : si l'on sait que l'utilisateur travaille sur une station Linux, le prototype exhibé pourra être le logiciel Grass[34]. L'idée de prototype est intéressante. Haton *et al.* en discutent, et évoquent des travaux qui la mettent en œuvre ([HBF+91], p.321).

Nous venons de prendre des exemples mettant en jeu un SIG. Nous aurions pu en prendre d'autres impliquant des tâches de programmation et non seulement d'utilisation d'interface (nous aurions pu entre autres développer l'exemple exposé fig.1.9 p. 26). La problématique et les objectifs sont les mêmes.

Connaissances tacites "triviales" nécessaires à une simulation informatique du raisonnement de l'expert

Signalons enfin un cas particulier, celui des connaissances tacites parce qu'évidentes pour les humains. S'il est inutile de les faire figurer dans les descriptions de traitement destinées à la consultation, il faut en revanche les représenter explicitement dans celles destinées à servir de support aux programmes de simulation du raisonnement de l'expert. Par exemple, supposons que dans la description de la fonction de fusion de thèmes d'Arcview figure simplement l'information "appartient à Arcview", sans aucune indication sur les conditions d'utilisation. L'utilisateur qui consulte cette description sait de façon tacite que s'il veut utiliser ladite fonction, il doit disposer du logiciel Arcview. En effet ce type de logiciel forme généralement un tout indissociable[35], au contraire de certaines API qui peuvent être composées de fonctions

[33] On peut considérer que les thèmes héritent des propriétés et valeurs de propriétés d'un concept plus général "ensemble d'objets graphiques". Les *frames* de Minsky reposent sur cette idée, que l'on retrouve également dans les *noèmes* de Husserl permettant de guider la perception que l'on a d'un objet. Le noème d' "ensemble d'objets graphiques" est associé à des croyances, il suscite une attente sur les perceptions futures. Cela peut provoquer des erreurs. Par exemple le novice peut avoir l'intention de modifier les objets d'un thème – comme il le fait avec les logiciels de dessin –, mais ne pas pouvoir mettre en œuvre la procédure habituelle. Selon Bachimont, "le noème peut (...) être compris comme la structure cognitive analogue aux *frames* de Minsky" ([Bac92], p.252).

[34] Le lecteur pourra objecter que ce dernier exemple est mal choisi : dans la pratique les utilisateurs de Linux ont un niveau de compétence qui rend inutile une explication sur un concept aussi trivial que les couches de données graphiques.

[35] pour être exact, précisons qu'en fait les fonctions des logiciels sont quasiment toujours définies dans des

indépendantes. Par conséquent, il nous faut représenter explicitement la règle *"si l'utilisateur a besoin d'une fonction d'un logiciel, alors il a besoin du logiciel"*.

1.3.6 Recevoir l'aide de l'expert

Les requêtes exposées figures 1.5 et 1.6 montrent le besoin de recevoir une aide de l'expert.

Le problème apparaît lorsque les termes des requêtes ne correspondent pas exactement au contenu de la base de métadonnées. Pour des requêtes comme celles qui suivent, l'utilisateur a besoin de recevoir l'aide de l'expert :

- *Quels sont les avantages de Lamps2 par rapport à Geoconcept 5.0 ?*
- *Pourquoi ne puis-je pas effectuer tel traitement sur cette donnée ?*
- *Quels sont les traitements utilisables sur les objets "bâtiments" ?*

- *Comment calculer l'enveloppe d'une triangulation ?*
- *Quel est le processus actuel de mise à jour des cartes régionales ?*
- *Comment calculer le plus court chemin routier entre Paris et Caen ?*
- *Je connais Arcview et je veux calculer la distance entre deux codes postaux.*
- *Je veux cartographier mon jeu de données routières. Je dispose d'une heure et ne connais pas Arcview.*
- *Quels sont les traitements qui permettent de sélectionner les rues qui intersectent rue Hayeps ?*

1.4 Quelles réponses ?

Nous avons exposé les besoins d'information des développeurs et utilisateurs de traitements informatiques géographiques. Nous proposons d'y répondre au moyen de métadonnées. Cela implique plusieurs objectifs. La construction d'une base de métadonnées suppose la définition d'un modèle et son instanciation. L'accès à cette base par l'utilisateur demande la construction d'une application permettant la consultation et l'acquisition selon diverses modalités.

1.4.1 Les métadonnées, pourquoi ?

Une métadonnée est littéralement "une donnée sur une donnée" (préfixe méta : du grec *meta*, ce qui dépasse, englobe un objet, une science[36]). Plus spécifiquement, c'est selon [Pec05a] "un ensemble structuré d'informations décrivant une ressource quelconque". Pourquoi les métadonnées sont-elles indispensables ?

Une personne cherche ou souhaite obtenir des informations sur un livre, un document audiovisuel, une œuvre d'art, une personne ou un programme informatique. La solution qui consisterait à parcourir physiquement les rayonnages des bibliothèques puis à lire l'intégralité du livre, ou visionner le film, se déplacer au musée, rencontrer la personne, utiliser le programme, etc. n'est pas envisageable, ne serait-ce que faute de temps. Il faut donc passer par un media intermédiaire. Ce média, c'est les métadonnées, qui jouent le rôle de substitut aux ressources originales.

Nous obtenons donc cette nouvelle définition : "Les métadonnées sont des données relatives à d'autres données et destinées à supporter des traitements impliquant ces autres données" [Rol99].

fichiers séparés. Par exemple, le fichier exécutable d'Arcview 3.1 s'accompagne d'une centaine de dll (dynamic link library). On pourrait imaginer isoler et utiliser une partie seulement de ces librairies, mais dans la majeure partie des cas ce type de logiciel n'est pas conçu pour être dépecé.

[36] Définition du Robert.

Les métadonnées existent sous différentes formes et sont destinées à différents publics pour différents usages. Dans les bibliothèques, par exemple, des documentalistes constituent des catalogues et rédigent des notices à l'intention de lecteurs humains. Au contraire, dans le domaine du Web sémantique les auteurs de documents ou développeurs de services créent eux-mêmes des descriptions destinées à des machines. De façon moins formelle, les critiques de films ou les manuels d'utilisation d'imprimante sont également des métadonnées.

Différents usages, différents domaines : les normes, recommandations et initiatives ayant trait aux métadonnées sont très nombreuses. Nous tenterons au chapitre 2 de dresser un état de l'art de celles qui existent pour les traitements informatiques géographiques, bien sûr, mais également de celles dont la vocation généraliste couvre nos besoins. On étudiera ainsi les métadonnées du domaine informatique en général, et celles destinées encore plus largement à tous types de ressources (Dublin Core et LOM notamment).

Lors de l'analyse des besoins d'informations sur les traitements[37], nous avons vu s'esquisser une façon d'organiser les connaissances. Il est intéressant de compléter notre point de vue en considérant différentes classifications de métadonnées. Par exemple, F. Role distingue[38] :

- les métadonnées dépendantes du contenu (p.ex. : le langage informatique dans lequel le traitement est implémenté),
- les métadonnées descriptives du contenu (p.ex : la fonctionnalité réalisée par le traitement),
- les métadonnées indépendantes du contenu (p.ex : la date de création du traitement).

De nombreux auteurs distinguent également les métadonnées internes ou externes aux ressources, dédiées davantage à l'indexation ou à la description, au type de public, au multilinguisme, à l'échange, à la sécurité, à l'authentification, aux droits, aux aspects administratifs, etc. Le domaine est vaste ; nous n'approfondirons pas son exploration au-delà de la limite estimée des besoins de notre contexte.

1.4.2 La nécessité d'un modèle de métadonnées

Nous avons établi la nécessité des métadonnées comme intermédiaire entre l'utilisateur et les traitements. Par ailleurs, l'analyse des besoins a permis de cerner les aspects des traitements à décrire. Il est donc possible de créer "librement" des métadonnées en langue naturelle, de les stocker sur un support quelconque et de les rendre accessibles aux utilisateurs. Ce type de pratique perdure encore partiellement ; il tend à disparaître grâce à l'édiction de normes de métadonnées.

Il est clair en effet que l'exploitation des métadonnées est facilitée lorsque celles-ci respectent des contraintes de structure et de contenu. En particulier, l'exploitation informatique demande l'adoption de normes. Cela était vrai pour des applications locales ; cela l'est d'autant plus avec les possibilités d'échange qu'offre Internet.

Comme le note D. Hillmann [Hil01] :

> "Le concept de métadonnées est antérieur à Internet et au Web. Toutefois, c'est avec l'augmentation de l'édition électronique et des bibliothèques numériques que l'intérêt mondial pour les pratiques et standards de métadonnées a véritablement explosé. La surabondance d'information (...), résultant de vastes quantités de données numériques non différenciées disponibles en ligne, explique cet intérêt soudain (...). L'adoption à grande échelle de normes descriptives et de nouvelles pratiques pour les ressources électroniques va améliorer la possibilité de trouver des ressources pertinentes dans Internet."

L'adoption d'un modèle de métadonnées permet de définir un ensemble d'éléments de descriptions, leur organisation, leurs relations, leur type et leurs valeurs possibles. Nous verrons

[37] *Connaître les traitements*, sous-section 1.3.3 p. 20.
[38] [Rol99] p.4, cité par [Rom01] p. 44)

qu'il peut également être doté d'une sémantique formelle. Le modèle considéré indépendamment des questions d'implémentation[39] sera appelé *modèle conceptuel* (chapitres 2 et 3). La traduction du modèle conceptuel dans un langage informatique sera appelée *modèle d'implémentation* (chapitre 4).

Ces deux formes du modèle de métadonnées permettent à l'humain et à la machine de manipuler *une représentation* des traitements décrits. Ainsi, par exemple, on ne comptera pas les programmes ADA disponibles au laboratoire COGIT en cherchant un par un tous les fichiers portant l'extension ADA, mais en effectuant une requête sur la base de métadonnées indexant lesdits programmes.

La distinction entre réalité et représentation de celle-ci est fondamentale en science. C'est pourquoi elle a fait l'objet de beaucoup de réflexions qui ont abouti à la notion de modèle[40].

Un modèle est une abstraction de la réalité. Compte tenu de notre contexte, l'exemple de la carte géographique s'impose. Il illustre bien le fait que l'opération d'abstraction implique des choix sélectifs : c'est une simplification de la réalité. Seuls les aspects utiles à un but donné sont retenus ; sans informations parasites le raisonnement est plus aisé. Un modèle est donc une vue abstraite, partielle, mais utile de la réalité. Dans ce qui suit nous poursuivons l'exemple de la carte géographique pour illustrer notre réflexion sur les modèles, et nous établissons le parallèle avec les besoins d'informations sur les traitements.

Pour un même lieu géographique, un urbaniste choisira une carte cadastrale tandis que l'analyste des réseaux de transports optera pour une carte topographique. Différents besoins impliquent différents thèmes ou différentes facettes de description d'une même réalité. Un de nos objectifs est donc de créer un modèle multi-points de vues. Nous avons indiqué par ailleurs que nous essaierons *d'adapter les modes d'emploi au contexte de l'utilisateur.*

Considérons cette fois des besoins de même nature mais de granularité ou complexité différentes. Un randonneur choisit une carte routière d'échelle 1/25 000 ème, un cycliste d'échelle 1/100 000 ème. Le niveau de détail est différent. De même, un utilisateur à la recherche d'un SIG pour cartographier grossièrement un jeu de donnée désirera consulter des descriptions ne contenant que les informations minimales (disponibilité, système d'exploitation, formats de données acceptées). Au contraire un cartographe professionnel exigera des informations détaillées sur les fonctionnalités offertes, les possibilités d'extension, les tests effectués, etc.

Nous sommes là face à un dilemme. Si l'enseignant humain peut adapter son cours au niveau de sa classe, construire un modèle et une application qui permette une telle chose paraît au premier abord délicat. Simples, les descriptions seraient incomplètes ; complètes, elles seraient trop complexes[41].

Comme nous l'avons déjà évoqué lors de l'étude des connaissances tacites, deux de nos objectifs sont donc d'autoriser des descriptions de complexité variable et de permettre la progressivité dans la présentation des informations.

Un modèle est conçu dans un but. En sciences physiques il s'agit de prédire les comportements du système modélisé, par exemple la chute des corps. Le modèle dans ce cas est la traduction d'une théorie qui comporte des lois et qui permet d'interpréter des faits.

Notre modèle de métadonnées aurait pu poursuivre un but du même type. Répondre à la requête *"Comment se comporte l'algorithme A sur tel type de données"* suppose de pouvoir simuler un aspect du comportement du traitement. Autre exemple, la fonction qui décrit le lien

[39] du moins en théorie. Nous discutons de ce point sous-section 2.2.1.

[40] parmi les sources traitant de la notion de modèle dont nous nous sommes inspirés, nous pouvons notamment citer [Pie00].

[41] variante de "Ce qui est simple est faux, ce qui est compliqué est inutilisable", sentence attribuée à Paul Valéry qui aurait par ailleurs affirmé "on ne raisonne que sur des modèles" [Moi87].

entre la taille et la qualité d'une image raster compressée en JPEG est connue [CVM03]. Elle pourrait être exploitable dans le cadre de notre application. Le calcul prédictif de la complexité des traitements – donc de leurs temps d'exécution machine – peut également être intéressant, par exemple si l'on souhaite comparer l'efficacité de deux algorithmes en fonction d'un jeu de données particulier.

Un autre type de but est poursuivi dans le domaine mathématique. Les modèles servent là à démontrer formellement des théorèmes, étant donné un ensemble d'axiomes et de règles d'une logique formelle. L'équivalent existe dans le domaine informatique. Le but est d'apporter des preuves de programmes (finitude, complexité, obtention du résultat escompté), au moyen de *méthodes formelles*.

Notre modèle n'a vocation ni à prédire ou à simuler le comportement de traitements, ni à apporter des preuves de programmes. Les buts que nous fixons à notre modèle sont ceux révélés par l'analyse de besoins, c'est-à-dire permettre la construction de métadonnées pour rechercher, connaître et utiliser les traitements. Ces métadonnées doivent posséder deux qualités difficilement conciliables : elles doivent être à la fois *expressives* et *opérationnalisables*. Elles doivent en effet être lisibles par l'homme (notamment avec du texte en langue naturelle et des images) mais aussi dotées d'une sémantique formelle qui permette la mise en œuvre de raisonnements (cf. les exemples ER 1 et ER 3 p. 20 et 21). Face à ce dilemme, vers quel type de solution allons nous nous orienter ?

1.4.3 Système d'Information ou Système à Base de Connaissance ?

Nos métadonnées doivent pouvoir, en partie, remplacer l'expert humain. Ce but appartient clairement au champ de l'Intelligence Artificielle. On peut considérer que l'IA construit deux types de systèmes : les systèmes qui raisonnent et les systèmes qui aident l'humain à raisonner. Historiquement, il semble que les systèmes qui raisonnent sont plus anciens que les systèmes qui aident à raisonner[42]. Les projets de systèmes intelligents cybernétiques apparus dans les années cinquante, puis plus tard les premiers systèmes experts à base de règles, par exemple, ont en effet précédé les travaux de représentations des connaissances aidant l'humain à raisonner (avec p.ex. la représentation orientée objet, les réseaux sémantiques et les principes de génie logiciel).

La distinction entre les deux grands types de systèmes de l'IA que l'on vient d'évoquer se retrouve de façon plus ou moins explicite dans les propos de plusieurs auteurs de la communauté des représentations des connaissances. B. Bachimont considère ainsi que "l'ingénierie des connaissances comprend deux modalités essentielles : l'ingénierie des représentations formelles de connaissances et l'ingénierie des inscriptions documentaires de connaissances" [Bac04]. De façon similaire, J. Caussanel et E. Chouraqui proposent une typologie des systèmes de gestion de connaissances où apparaissent deux branches : les *Systèmes d'Information (SI)* et les *Systèmes à Base de Connaissance (SBC)* [CC99]. Laquelle des deux orientations devons-nous suivre ? Voyons plus précisément ce qu'il en est des SI et des SBC, et quel choix est le plus approprié à notre but.

"Un SI est un ensemble organisé de ressources (matériel, logiciel, personnel, données, procédures, ...) permettant d'acquérir, de stocker, de transformer et de communiquer des informations sous forme de textes, images, sons, ou de données codées dans des organisations"[43]. Un système de gestion de base de données indexant les ouvrages d'une bibliothèque, par exemple, constitue un SI. Certes, il existe souvent des programmes informatiques pour traiter les informations

[42] On pourra trouver, entre autres, dans [Teu00] une chronologie sommaire des systèmes d'informations et des travaux d'intelligence artificielle.

[43] Encyclopédie Wikipedia. `http://fr.wikipedia.org/wiki/Système_d'information`

du SI ; par exemple, dans le cas d'un système d'information bibliothécaire, ces programmes permettent d'effectuer des recherches, de gérer des emprunts, etc. Mais les traitements effectués font appel à des connaissances figées dans le code des programmes et clairement séparées des informations du SI.

Au contraire, un SBC est un systèmes où les connaissances font partie des données et non des procédures. Selon, J. Pomian, un SBC est le résultat de la capitalisation des connaissances, démarche qui vise à identifier, recueillir et rendre exploitable, quels que soient le contexte, l'expérience acquise par une organisation [Pom96]. Aussi appelés systèmes experts de seconde génération [ALR96], les SBC contiennent les connaissances destinées à être opérationnalisées. Certains auteurs considèrent que les mémoires d'entreprises sont des SBC où les connaissances sont destinées aux humains, et non opérationnalisables informatiquement [CC99]. Nous pourrions décrire les connaissances d'utilisation des traitements suivant cette voie. Mais nous ne pourrions alors pas simuler une partie du raisonnement de l'expert. Notre modèle de métadonnées doit donc permettre la construction d'un SBC opérationnalisable informatiquement. Pour cela une approche possible est de formaliser les connaissances des domaines des traitements dans des ontologies formelles et de représenter en complément les règles de l'expert dans un langage de logique à l'expressivité adaptée[44].

En conclusion, notre objectif est double. D'abord nous allons construire un SI. Ensuite, par la représentation logico-formelle et l'opérationnalisation d'une partie des connaissances du SI, nous mettrons en place un SBC répondant à une gamme de besoins dont plusieurs exemples seront donnés chapitre 3.

1.4.4 L'acquisition et la consultation des métadonnées

La figure 1.13 montre les principaux cas d'utilisation de l'application à construire. Une même personne peut endosser alternativement les rôles d'utilisateur qui consulte la base de métadonnées et d'auteur de métadonnées qui en saisit de nouvelles. Nous n'exposons pas ici tous les cas d'utilisation. Par exemple, les différentes modalités de recherche dans la base de métadonnées (par mot-clés, par soumission de formulaires de requêtes, par sélection dans des index de navigations) ne sont pas détaillées.

1.4.5 Définition des objectifs à atteindre

Nous définissons deux types d'objectifs : relatifs au modèle de métadonnées, et relatifs à l'application qui permet l'accès à la base de métadonnées qui instancie le modèle. Les principes listés ci-dessous sont bien connus. Ils sont prescrits, entre autres, par [SB01] (cité par [Tao02]).

Objectifs relatifs au modèle de métadonnées

On a vu se dessiner plusieurs objectifs. Ils vont conditionner notre façon de modéliser les connaissances sur les traitements. Premièrement, il faut organiser les connaissances de telle sorte qu'une représentation progressive soit possible ; deuxièmement, il faut les décrire systématiquement au niveau le plus général possible afin de factoriser les aspects communs de leur description ; troisièmement, il faut introduire la notion de prototype afin de permettre l'exhibition d'exemples basés sur l'analogie.

O1 : Universalité. Le modèle doit permettre de décrire tous les traitements informatiques

[44] Nous verrons que ces deux approches complémentaires correspondent à deux des couches du *"layer cake"* du Web sémantique proposé par T. Berner-Lee (cf. fig.4.2, p. 152).

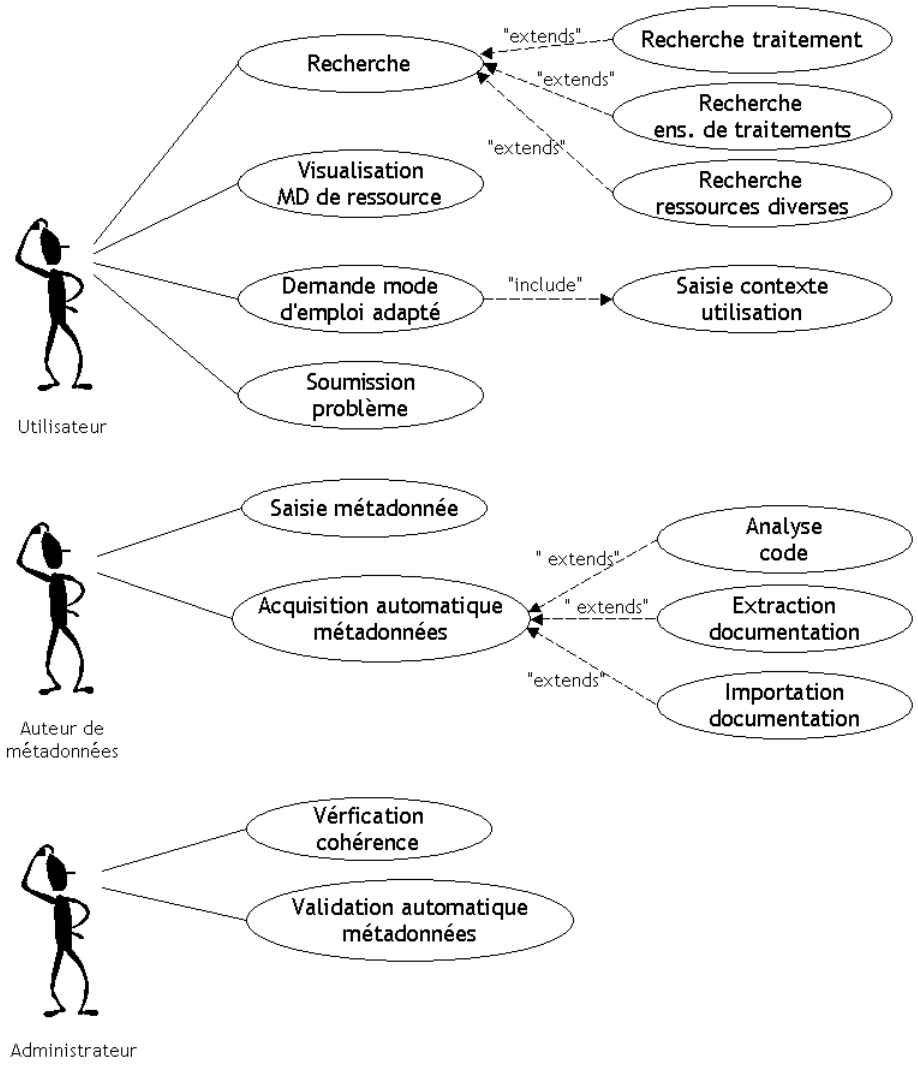


FIG. 1.13 – Application d'accès au métadonnées des traitements – principaux cas d'utilisation

géographiques[45].

O2 : Homogénéité. Les traitements doivent être décrits de façon homogène. La structure et le contenu des métadonnées doivent être contraints. En théorie, pour un niveau de détail donné, il ne doit exister qu'une seule description de traitement conforme au modèle[46]. Autant que possible, les standards existants doivent être utilisés.

O3 : Intégralité. Les éléments de description du modèle doivent tendre à refléter tous les points de vue des traitements pertinents pour l'utilisateur. Ceci dit, il est toujours possible de détailler plus précisément certains aspects des descriptions, d'où l'importance de l'objectif O4 suivant.

O4 : Extensibilité. Le modèle doit être extensible. L'ajout d'éléments de descriptions ou de relations entre ceux-ci doit être possible. La possibilité de référencer ou d'inclure des sources de métadonnées existantes doit être offerte.

O5 : Intelligibilité. Le modèle doit autant que possible être simple, compréhensible, et non ambigu. Il doit organiser les connaissances de façon logique. Il doit permettre une vision progressive des connaissances ; c'est là un point important.

O6 : Opérationnabilité. Une partie des connaissances représentables à l'aide du modèle doit être opérationnalisable. Un de nos objectifs est en effet la simulation de certains raisonnements de l'expert. Le modèle doit capturer les connaissances sous une forme déclarative, mais suffisamment formelle pour permettre le passage à une forme procédurale. Pour certaines connaissances ces deux formes apparaîtront inconciliables ; nous renoncerons alors à l'objectif d'opérationnalisation mais non à celui de l'acquisition.

Objectifs relatifs à l'application d'accès aux métadonnées

O7 : Acquisition.

O7a : L'application doit permettre l'instanciation du modèle, c'est-à-dire l'acquisition des métadonnées sur les traitements et les connaissances sur les domaines associés.

O7b : L'acquisition doit être en partie automatisée (analyse de codes et de documentations existantes, déduction à partir d'informations saisies).

O7c : La structure et le contenu des métadonnées doivent être contrôlés. La définition formelle des concepts mobilisés doit, de plus, permettre de contrôler l'interprétation d'une partie des métadonnées : pour le lecteur qui vérifie le sens des termes employés ; pour l'application qui tire les conséquences logiques des faits explicitement spécifiés et infère de nouveaux faits. L'acquisition des connaissances d'experts comprend ainsi celle des définitions des concepts des domaines impliqués.

O7d : Dans un souci d'incitation à l'acquisition auprès des auteurs des traitements, les descriptions incomplètes doivent être dans une certaine mesure tolérées.

O8 : Consultation.

O8a : L'application doit permettre la soumission de requêtes via des formulaires.

O8b : L'application doit permettre la recherche par mots-clé.

O8c : L'application doit permettre la navigation par le parcours des relations entre métadonnées.

[45] Si l'on considère notre modèle comme une fonction qui à un traitement associe une description, l'universalité correspond à la propriété de surjection.

[46] Relativement à la fonction de la note précédente, l'homogénéité correspond à la propriété d'injection.

O8d : Dans leur présentation à l'utilisateur, les descriptions doivent posséder une forme qui répond à la fois aux spécificités du domaine géographique et aux limites identifiées dans les descriptions existantes (cf. chapitre 2). De plus, diverses fonctionnalités classiques de consultation seront offertes, telles le tri des résultats des requêtes, la possibilité de croiser les critères de recherche, l'offre d'une interface ergonomique et conviviale, l'exhibition de diagrammes statistiques.

O9 : Exploitation.

O9a : Sans préjuger de la forme qu'elle doit prendre, l'implémentation du modèle doit suivre un certain nombre de recommandations prescrites notamment pour la conception de schémas de base de données relationnelles : présence d'attributs identifiant de façon unique les ressources, absence ou limitations des redondances par l'usage de références, etc.

O9b : Des mécanismes d'inférences doivent être mis en œuvre pour répondre aux besoins qui seront exposés en détail au chapitre 3.

O9c : Les choix techniques adoptés doivent prendre divers critères de faisabilité tels le passage à l'échelle, la mise à jour de la base sans l'intervention d'un gestionnaire humain, la modularité de l'architecture (séparation présentation, traitements des requêtes, base de métadonnées), la gestion des accès concurrents, etc.

O9d : L'application doit tendre à l'indépendance vis-à-vis du modèle. Modifier le modèle doit impliquer un minimum de répercussion sur le code de l'application.

Objectifs que l'on choisit de ne pas poursuivre

La compréhension du besoin de l'utilisateur est un des verrous des systèmes d'aide. Les techniques de traitement automatique de la langue naturelle (TALN) permettent de faciliter le dialogue entre utilisateurs et applications. Cette piste de recherche est intéressante, mais nous ne l'explorons pas.

Nous ne poursuivons pas non plus la piste qui aurait consisté, pour décrire les effets des traitements, à créer des descriptions capables de simuler le comportement des traitements, ou de présenter les résultats de méthodes statistiques telles que les plans d'expérience. Si des descriptions liées à ces techniques existent, elles pourront être référencées dans les descriptions (objectif O4) mais ne seront pas validées par le modèle.

Nous n'essayons pas de concurrencer les outils de génie logiciel tels les IDE (Integrated Development Environment, ou environnement de développement intégré) pour l'aide au développement de traitements informatiques. Notre système d'accès aux métadonnées a un rôle complémentaire plutôt que concurrent de celui des IDE, dont nous verrons d'ailleurs au chapitre 2 quelques-unes des fonctionnalités. Un objectif intéressant aurait été de coupler notre système avec un IDE. Nous n'avons pas poursuivi cet objectif. Globalement, notre approche se situe aussi à un niveau de description plus abstrait que le niveau code, même si nous considérons bien sûr aussi ce dernier. Nous verrons néanmoins au chapitre 6 que nous avons été amenés à utiliser certains outils d'analyse de code pour automatiser l'acquisition d'une partie des métadonnées.

Concernant la simulation du raisonnement de l'expert, notre objectif n'est pas l'opérationnalisation totale des connaissances, mais la meilleure présentation de celles-ci à l'utilisateur, sous une forme qui, par ailleurs, permet la mise en œuvre d'un certain nombre de mécanismes d'inférence. Ce faisant, nous nous situons dans la lignée des travaux d'intelligence artificielle qui parient davantage sur une coopération agent logiciel – agent humain que sur les capacités des seuls agents logiciels. G. Boy formule ainsi ce point de vue : "l'optimum pour le système ne

se situe pas au degré d'automatisation maximum, mais à un point d'automatisation qui optimise le résultat de son utilisation par l'agent-humain" (cité par [Aus89]).

1.5 Conclusion

Nous avons présenté le contexte de notre travail : les traitements informatiques géographiques à l'IGN. Nous avons identifié ce qui nous semblait en être les principales catégories, en tentant notamment de cerner les spécificités liées au domaine géographique.

Nous avons mené l'analyse des besoins. Il faut permettre aux utilisateurs et aux développeurs de l'IGN de rechercher, connaître et utiliser les traitements géographiques. Il faut leur donner accès aux connaissances de l'expert, et simuler une partie des raisonnements de ce dernier pour répondre aux requêtes qui nécessitent de dériver de l'information non explicitement présente dans la base de métadonnées. En particulier, nous souhaitons fournir des modes d'emploi adaptés au contexte d'utilisation.

Il s'ensuit que nous nous fixons pour objectifs de définir un modèle de métadonnées, puis de construire une application couplant un *système d'information* et un *système à base de connaissance*.

Chapitre 2

Proposition d'un modèle conceptuel de métadonnées

Ce chapitre a pour but la définition de notre modèle conceptuel de métadonnées.

Nous commençons par dresser un état de l'art des métadonnées de traitements (section 2.1). L'exhaustivité n'est pas visée. Elle serait de toutes façons difficile à atteindre compte tenu de l'étendue des travaux liés aux métadonnées des traitements informatiques. Il existe notamment de nombreux projets cherchant à permettre la description de services Web. Après nous être intéressés aux métadonnées des traitements informatiques en général, nous porterons notre attention sur celles concernant plus spécifiquement le domaine géographique.

Notre état de l'art recense principalement des modèles de métadonnées, proposés par exemple par les normes d'organismes reconnus. Mais il recense aussi des métadonnées comme les manuels d'utilisateur. Dans ce dernier cas le modèle est implicite ; nous cherchons alors à en identifier les éléments. L'aperçu des quelques normes et langages de métadonnées de ressources généralistes est également utile à nos buts.

Ce tour d'horizon effectué, et après avoir précisé quelques points relatifs à notre démarche et à nos choix de modélisation (section 2.2), nous présentons les diagrammes de classe de notre modèle conceptuel (section 2.3).

2.1 État de l'art des métadonnées des traitements

L'état de l'art proposé ici regroupe une sélection de descriptions existantes, de modèles de description (standards, normes et langages) et d'outils utiles à la connaissance des traitements[1]. Nous avons cherché à recueillir les éléments de description et les idées les plus intéressants en vue de la construction de notre propre modèle. Nous abordons ainsi des modèles de métadonnées et des langages aux vocations diverses : formels ou non, destinés aux humains ou aux machines, à la conception ou à l'utilisation, etc. Tous sont potentiellement porteurs d'enseignements utiles à notre but.

2.1.1 Les producteurs de modèles de métadonnées

Dès lors qu'une activité nécessite l'échange d'information entre agents humains ou logiciels, des protocoles de communication apparaissent. Des langages sont créés, des normes et des

[1]Parce que toutes informations sur les traitements ne sont pas à proprement parler des descriptions de ceux-ci, nous préférons parler d'état de l'art des métadonnées plutôt que d'état de l'art de descriptions. Par exemple, nous faisons figurer dans notre état de l'art la fonctionnalité d'un IDE comme Eclipse qui permet de savoir où est utilisée une classe Java (un IDE – *Integrated Development Environment*, environnement de développement intégré en français – est un logiciel réunissant les outils nécessaires à la création d'applications).

standards de métadonnées sont définis pour structurer les documents supports de l'information échangée. Dans le domaine de l'informatique des consortiums d'industriels se constituent, des communautés se dotent d'organismes dont elles reconnaissent l'autorité. Lorsque les intérêts en jeu le justifient, les normes émanent de l'État.

Normes et standards

Les termes norme et standard se traduisent tous deux en anglais par standard. En français il existe cependant une nuance. Elle se situe, selon G. Chartron [Cha00]

> "au niveau des acteurs en jeu et des procédures de consensus attachées :
> – La norme fait référence surtout à l'ISO et à ses instances nationales telles que l'AFNOR *[Association Française de NORmalisation]* en France avec des processus de validation assez lourds.
> – Le standard est plus assimilé à un processus réactif de consensus du monde économique ou du monde technique. Pour des organismes comme le W3C, l'enjeu est un accord consensuel pour le développement rapide du commerce ; pour l'IETF *[Internet Engineering Task Force]*, l'enjeu est le développement ou l'extension rapide de protocoles techniques. Les méthodes de travail adoptées alors pour l'élaboration de ce type de standard répondent à une exigence de vitesse : le courrier électronique et les forums sont des outils de travail majeurs".

Alors qu'un standard est un "ensemble de recommandations développées et préconisées par un groupe représentatif d'utilisateurs", une norme est, selon la définition de l'ISO, "un document établi par un consensus et approuvé par un organisme reconnu, qui fournit, pour des usages communs et repérés, des règles, des lignes directrices ou des caractéristiques, pour des activités ou leurs résultats, garantissant un niveau d'ordre optimal dans un contexte donné".

Principaux organismes émetteurs de normes de métadonnées des traitements

Les organismes qui établissent des normes – notamment des normes de métadonnées – sont très nombreux. Nous ne citons ci-après que les principaux dont nous avons été amenés à étudier les propositions. Les normes mentionnées dans cette section seront présentées dans la suite du mémoire.

Le premier producteur mondial de normes internationales est l'ISO (International Organization for Standardization) [ISO05b]. Les 190 comités techniques[2] de l'ISO travaillent sur des domaines très divers. Celui qui nous intéresse tout particulièrement est le comité pour l'information géographique et la géomatique, le TC 211. Il définit les normes ISO 19119 et 19115 respectivement pour les métadonnées des services Web géographiques et les métadonnées des données et traitements géographiques. D'autres normes ISO, portant sur les métadonnées de façon plus générale, sont également utiles à nos besoins. L'ISO n'est pas toujours à l'initiative de leur création, elle se contente parfois d'entériner des standards existants, tels que Dublin Core, SGML, LOM et TopicMaps.

Une autre organisation importante de standardisation est l'IEEE (Institute of Electrical and Electronics Engineers). Même si elle est surtout connue pour l'édiction de normes informatiques "bas niveau" (télécommunications), on lui doit le modèle de métadonnées des objets d'enseignement LOM.

[2]Chiffre en 2005 (`http://www.iso.org/iso/fr/aboutiso/isoinfigures/January2005-p1.html`).

OASIS (Organization for the Advancement of Structured Information Standards), "consortium d'industriels visant à promouvoir l'utilisation de standards ouverts et auteur de nombreuses spécifications liées à XML"[3], a, pour sa part, défini UDDI, standard pour les annuaires des services Web. Dans le monde du Web, justement, l'organisation incontournable est le W3C (World Wide Web Consortium). Son but est de standardiser les langages du Web. Nous verrons que la partie "implémentation" de notre travail repose entièrement sur des langages W3C. Ces langages ne sont pas à proprement parler des langages de métadonnées, ce sont principalement des langages de définition de formats de contenus. Ce sont donc, en quelque sorte, des normes de méta-langages applicables aux métadonnées. Nous les présenterons aux chapitres 4 et 5. Pour l'heure, les normes W3C qui nous intéressent sont celles qui peuvent être vues comme définissant directement des métadonnées. C'est le cas de WSDL et de SOAP pour les services Web, de MathML pour les notations mathématiques. Le W3C comporte un groupe de travail lié au domaine géographique[4], mais ses travaux ne nous sont pas utiles. Ils ont en effet pour but de fournir un vocabulaire décrivant les informations de localisation spatiale.

Dans le domaine du génie logiciel, l'OMG (Object Management Group) a proposé des normes célèbres comme UML, MOF, CORBA et IDL. Nous allons évoquer UML dans ce chapitre, bien qu'il s'agisse plutôt d'un méta-langage de conception logicielle orienté objet que d'un langage de métadonnées des traitements à proprement parler.

Dans le domaine spécifiquement géographique, l'organisme le plus connu semble être l'OGC (Open Geospatial Consortium – anciennement Open GIS Consortium, avec GIS pour Geographic Information Systems). L'OGC regroupe plus de 200 membres dans le monde entier. Alors que le comité technique TC 211 de l'ISO édicte des normes sous forme de modèles conceptuels, l'OGC s'attache à fournir des solutions techniques sous forme de définitions d'interfaces de services Web (WFS, WMS et WCS pour Web Feature/Map/Coverage Service notamment, cf. p. 70), sous forme de formats XML de descriptions de services (implémentation d'ISO 19119), et sous forme de format XML de données (GML, Geography Markup Language).

Les normes de métadonnées géographiques existantes concernent principalement les données, beaucoup plus que les traitements. De fait, on trouve sur les sites d'organismes comme le FGDC (Federal Geographic Data Commitee[5]), avant tout des normes de métadonnées de données. L'examen de ces normes n'est pas sans intérêt si l'on veut décrire ce que font les traitements, comment ils affectent les données. C'est pourquoi nous avons par la force des choses été amenés à examiner, par exemple, les normes de métadonnées ISO 19115 et CSDGM (Content Standard for Digital Geospatial Metadata, élaborée par le FGDC)[6]. S'il nous arrivera de les évoquer dans la suite de ce mémoire, nous ne les détaillerons pas pour autant ; cela nous aurait éloigné de notre sujet principal.

Tout comme les logiciels, les modèles de métadonnées sont le fruit d'un travail. Tout comme eux, ils relèvent de réglementations liées à la propriété intellectuelle. L'accès aux normes n'est donc pas toujours libre. Il en coûte par exemple 224 francs suisses à qui veut consulter la spécification du standard SGML – ISO 8879[7]. En revanche, tous les standards du W3C sont libres de droits. Les politiques de développement sont différentes ; il est en tous cas clair que plus un travail – modèle, langage ou logiciel – s'offre aux regards et aux critiques, plus ses défauts ont des chances d'être détectés et corrigés. Il existe plusieurs types de licences. Le W3C

[3] `http://xmlfr.org/index/org/oasis/`

[4] `http://www.w3.org/2003/01/geo/`

[5] Créé par le gouvernement américain pour coordonner le développement de la NSDI (National Spatial Data Infrastructure).

[6] `http://www.fgdc.gov/metadata/csdgm/`

[7] Prix indiqué en 2005 sur le site de l'ISO `http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387`.

a adopté le système RF, l'ISO, OASIS et l'OMG, le système RAND[8].

Un bon modèle de métadonnées participe de façon importante à l'efficacité d'un système d'information et de gestion des connaissances. En cela c'est un facteur de compétitivité. Pour cette raison, dans les secteurs d'activités régis par la loi de la concurrence, les entreprises n'ont pas intérêt à partager les modèles fruits de leur expérience. L'espionnage industriel étant un délit, l'état de l'art établi dans ce mémoire est forcément incomplet. Ceci dit, on peut penser que s'il existe peut-être des modèles de métadonnées intéressants mais confidentiels, il est probable que, même si nous y avions eu accès, des adaptations à la spécificité du contexte de l'IGN auraient été nécessaires.

De façon générale, on peut penser que des besoins spécifiques nécessitent presque toujours la création de modèles *ad hoc*. Les normes et standards qui ambitionnent de toucher un public large ne peuvent être à la fois simples et adaptés à tous les besoins. Un nombre important de travaux présentés dans les conférences informatiques et géographiques consistent justement à présenter l'application d'une norme à un contexte particulier, et à en déduire les adaptations nécessaires.

Des modèles de métadonnées peuvent être définis de façon locale dans le cadre de projets. Nous avons pu nous inspirer des grilles de descriptions de l'OEEPE (Organisation Européenne d'Etudes Photogrammétriques Expérimentales) utilisées par les développeurs du projet Agent destiné à l'automatisation de la généralisation cartographique, projet auquel a participé activement le laboratoire COGIT.

[8]RF (Royalty Free) : les auteurs renoncent aux droits sur les brevets qui pourraient être créés à partir de leur travail ; RAND (Reasonable And Non Discriminatory) : les auteurs s'accordent pour être "raisonnables" et ne pas s'attaquer entre eux, par contre ils se réservent le droit de faire payer les usages extérieurs faits de leur travail par des personnes extérieures. Un article expliquant pourquoi le W3C a choisi RF plutôt que RAND comme politique en matière de licence est disponible sur `http://www.uzine.net/article1401.html`.

2.1.2 Métadonnées des traitements informatiques

Sommaire

1) Métadonnées des fichiers informatiques

Les tables d'allocations et les informations internes

Les traitements informatiques se présentent sous forme de fichiers. Leur nom nous renseigne déjà, leur extension (".com", ".exe.", ".dll", etc.) également.

Pour en savoir plus, demandons à voir les propriétés des fichiers, par les commandes adéquates d'une console ou via l'interface de l'explorateur de fichiers (fig. 2.1).

Nom	Taille	Type	Modifié le
Attributs	Commentaire	Créé le	Dernier accès le
Propriétaire	Titre	Objet	Catégorie
Copyright	Description du module	Version du module	Nom du produit
Version du produit			

TAB. 2.1 – Quelques-unes des informations disponibles avec les tables NTFS

Le numéro de version des logiciels, que l'on retrouve également dans la fenêtre "À propos..." classiquement accessible à partir du menu "?", est une information importante.

> "La notion de version est importante en informatique. Nous avons affaire à des logiciels (et maintenant à des documents électroniques) évolutifs. Le nom d'un logiciel, ou bien le

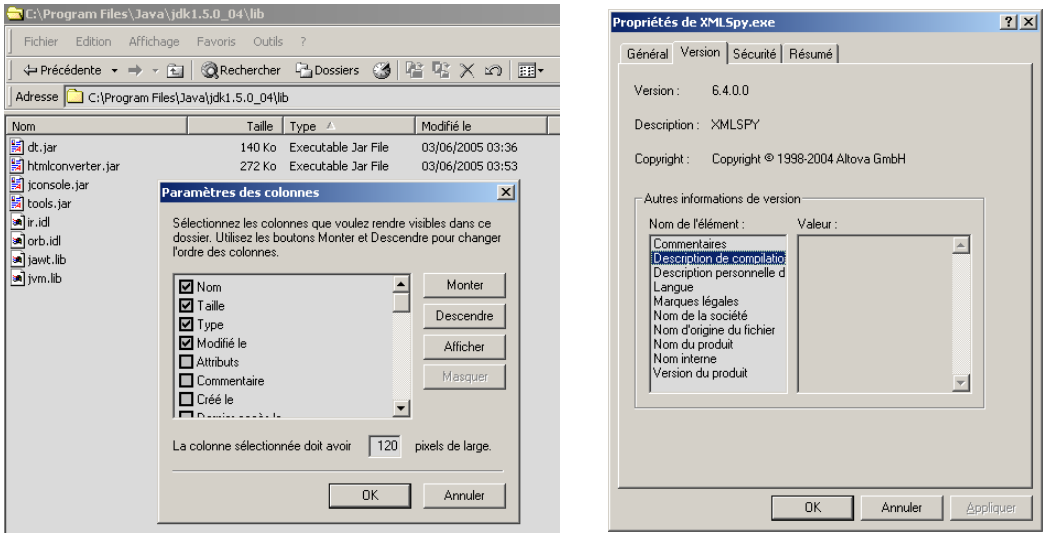


FIG. 2.1 – Visualisation des propriétés des fichiers sous Windows

titre d'un document placé sur l'internet, ne suffisent pas toujours à le définir. Il convient d'ajouter le "numéro de version". En général, une version "de premier développement", c'est à dire qui n'est pas encore fiable, possède un numéro commençant par 0. (...). Chaque "mise à jour" du logiciel voit progresser son numéro de version. On considère que les versions dont seul le chiffre décimal progresse sont des versions qui améliorent un logiciel sans apporter des changements profonds dans l'apparence ou dans les fonctionnalités. Régulièrement des pas importants sont franchis, et les logiciels passent "à la version supérieure", en modifiant le chiffre des unités.(...) Entre deux versions existent des versions "bêta", c'est-à-dire des versions qui sont diffusées pour que les spécialistes puissent repérer les erreurs et les signaler à l'auteur (ou à la société de production)" [Cro98].

On peut également ajouter que pour les logiciels commerciaux l'évolution des numéros de versions répond souvent à une logique *marketing*.

Où sont stockées les informations ?

Les métadonnées telles que celles du tableau 2.1 proviennent des tables des systèmes de fichiers : FAT, ou NTFS et des *streams*.

"En NTFS, un fichier consiste en plusieurs data streams qui sont un peu la généralisation du concept de fork pour les fichiers Macintosh. Un stream particulier contient les informations de sécurité (droits d'accès, etc.) et un autre stream appelé standard contient les données habituelles, celles qui sont normalement accessibles par les applications. Quand on examine avec l'Explorateur de Windows un fichier "normal" nommé MonFichier, c'est le stream appelé en interne MonFichier::DATA qui est affiché. Mais il peut exister aussi d'autres streams (alternate streams) liés au stream standard et contenant [d'autres] métadonnées" [Pec05b].

Les métadonnées qui viennent d'être évoquées ne sont pas *a priori* réservées à un public particulier. En revanche, il est bien demandé au concepteur de traitement informatique de produire deux types de documents distincts : les "documentations utilisateurs" et les "documentations programmeurs".

2) Les documentations destinées aux utilisateurs

La recommandation en forme de boutade "Quand tout a échoué, lisez le manuel" insinue la paresse des utilisateurs. Cette assertion n'est probablement pas totalement fausse. Mais on pourrait aussi mettre en cause la forme des documentations proposées. Elles existent essentiellement sous trois formes :

- documents papiers (livres, manuels, revues),
- fichiers d'aide et programmes d'assistance qui accompagnent sous forme électronique le traitement – on parle d'aide *helpdesk*,
- documents disponibles sur Internet (cours, tutoriels, sites spécialisés, articles, etc.).

Ces trois formes de documentation ne sont bien sûr pas exclusives : on peut imprimer des fichiers d'aide ou trouver sur le Web des manuels au format électronique. Néanmoins, elles possèdent chacune des caractéristiques propres. Elles traduisent également l'évolution des pratiques d'utilisation. Au début des années 90, les logiciels s'accompagnaient de manuels papiers volumineux. Ils ont progressivement été remplacés au profit de documentations au format électronique, qui aujourd'hui sont désormais de plus en plus souvent accessibles "en ligne", c'est-à-dire directement sur le Web. Entre autres avantages, ces évolutions successives ont eu pour conséquence la résolution des problèmes de mises à jour. Cependant, si le recours au format électronique a bel et bien entraîné une révolution dans les usages pour les besoins d'informations ponctuels, la forme "livre papier" reste encore la plus pratique dès que le volume de connaissances à acquérir est important. On peut donc pronostiquer qu'il va subsister pour quelque temps encore des rayons "informatique" dans les bibliothèques.

Les documents papiers

Une vision pessimiste de la documentation "papier" pourrait pousser à définir, de façon quelque peu triviale, l'informatique comme "20% de hardware et de software et 80% de footware dans les harmwares"[9]. Il est vrai que les protocoles de gestion de projet imposent la rédaction de documentations souvent volumineuses. La lecture des spécifications de normes informatiques peut également paraître rebutante.

La première leçon à tirer de ce constat est qu'une documentation de traitement doit être proche du besoin de l'utilisateur, et, en particulier, adaptée au niveau de détail qu'il attend. Certains ouvrages tentent bien de proposer plusieurs niveaux de lecture (on trouve ainsi dans les manuels d'utilisation des chapitres "démarrage rapide" pour les lecteurs pressés, ou des résumés à chaque début ou fin de chapitre), mais la nature figée du texte limite fatalement les possibilités. De ce point de vue, l'apparition de l'hypertexte présente une réelle valeur ajoutée.

Une deuxième leçon peut être tirée de l'étude des documentations papiers. Chaque type (manuel utilisateur, cahier des charges fonctionnel, cahier des charges technique, article de conférence présentant un traitement, thèse, etc.) respecte des conventions en termes de structure et de contenu. Un article suit en général le plan *résumé, contexte, analyse des besoins, état de l'art, expérimentations, conclusion* ; un manuel comporte la plupart du temps un sommaire, un index, un glossaire, et des conventions typographiques particulières (définition, astuce, erreur classique, point technique)[10]. Cela signifie pour notre modèle de métadonnées autant d'éléments de description potentiels à intégrer.

[9] Dictionnaire terminologie informatique de F. de Solliers, cité par `http://www.linux-france.org/prj/jargonf/D/documentation.html`.

[10] Une des raisons du succès de la collection *"Pour les nuls"*, destinée à ses débuts à la vulgarisation informatique avant d'être déclinée dans d'autres domaines, est probablement d'avoir systématisé le recours aux icônes aidant le lecteur à se repérer. Chaque page comporte ainsi une part de texte et une part d'annotation, c'est-à-dire de métadonnées.

Les fichiers et aides "helpdesk"

Les fichiers *Readme* (ou *LisezMoi*) et *HowTo* qui accompagnent presque systématiquement les logiciels ne respectent pas de format particulier. Les rubriques qu'on y trouve contiennent généralement des informations sur le matériel requis, les problèmes connus, les évolutions par rapport aux versions antérieures.

Un peu plus élaborées dans leur mise en forme que les simples fichiers textes, les pages d'aide aux formats HTML, CHM ou HLP[11] décrivent les fonctionnalités des logiciels et les instructions pour les invoquer. Nous examinerons spécifiquement les éléments de descriptions des aides des SIG dans la partie consacrées aux traitements informatiques géographiques (p. 77).

Les systèmes d'aide à l'utilisateur souvent associés aux pages d'aide proposent classiquement un sommaire, une indexation par mots-clés et une fonctionnalité de recherche plein-texte. Plus sophistiqués, les agents "intelligents" tels les *compagnons Office* en environnement Microsoft tendent à permettre à l'utilisateur d'exprimer librement ses questions. L'intelligence réside, en l'occurrence, en la capacité desdits agents à établir des relations de synonymies et à faire le lien entre les termes du besoin spontanément utilisés par l'utilisateur et ceux décrivant les fonctionnalités correspondantes. Ces outils restent cependant pour l'instant encore assez frustres.

Une caractéristique intéressante de certains systèmes d'aide est de fournir une aide contextuelle. Cela signifie que les pages d'aide présentées à l'utilisateur dépendent du logiciel qu'il est en train d'utiliser, voire de l'action qu'il est en train de réaliser. Cette prise en compte du contexte permet de faire gagner du temps de recherche à l'utilisateur, mais ne donne lieu à aucune adaptation du contenu des pages d'aide.

Pour réellement obtenir une aide adaptée, il nous faut nous tourner non pas vers les documentations évoquées ci-dessus, mais vers les programmes usuellement appelés *wizards*. Les wizards guident l'utilisateur dans la réalisation des tâches comme le paramétrage.

Le couplage entre pages d'aide et code des logiciels est certainement une tendance appelée à se développer. En environnement Microsoft, les pages d'aide permettent ainsi désormais de visionner les sélections à effectuer au sein des différents menus. L'intérêt de cette fonctionnalité est illustré par les deux premières phrases de la maxime :

Tell me, and I forget. Show me, and I remember. Let me do, and I understand.

Les documentations sur Internet

Parallèlement à la publication des documentations officielles, des communautés d'utilisateurs se forment sur Internet. Les forums de discussion (*newsgroups*) sont accessibles publiquement sur le Web. L'ensemble des personnes qui y participent forme le *Usenet*[12]. Par exemple, tous les sujets généraux relatifs aux SIG sont discutés sur `comp.infosystems.gis`[13] ; les sujets concernant plus spécifiquement les SIG Esri sont discutés sur `comp.soft-sys.gis.esri`.

Les utilisateurs novices posent souvent les mêmes questions. Afin d'éviter de polluer les forums, les modérateurs créent des FAQ (*Frequently Asked Question*)[14].

Autre façon d'échanger des messages au sein d'une communauté sur Internet, l'abonnement par mail à des listes de diffusion est également très répandu. Les utilisateurs du SIG Geoconcept

[11] cf. p. 52 pour une description de ces formats.

[12] `http://usenet-fr.news.eu.org/fr.usenet.reponses/usenet/Qu-est-ce-que-Usenet.html`

[13] `http://groups.google.fr/group/comp.infosystems.gis/about`

[14] Par exemple une FAQ sur les SIG se trouve sur `http://www.faqs.org/faqs/geography/infosystems-faq/`.

peuvent ainsi soumettre leurs problèmes et faire partager leurs expériences sur la Geoliste (`geo.liste@geoconcept.com`). Il existe également au sein de l'intranet de l'IGN plusieurs listes de diffusion. Les messages que l'on y trouve témoignent des besoins d'aide identifiés au chapitre 1.

Pour l'utilisateur qui ne fait que consulter leurs archives, le principal défaut des forums de discussion, FAQ et listes de diffusions tient parfois à la difficulté d'y mener une recherche. L'utilisateur qui publie sa question est tributaire de la disponibilité d'un humain ; le problème se déplace hors du cadre fixé par nos hypothèses de travail.

Nous pouvons néanmoins retenir deux leçons de l'étude de ce type d'aide. La première est que les utilisateurs sont confrontés à des **problèmes**. Ils doivent être indexés dans notre base de métadonnées. La seconde leçon est que certains de ces problèmes nécessitent parfois l'établissement d'un dialogue avec l'expert, et que l'exploitation des précisions apportées demande la mobilisation de connaissances pour s'adapter à des contextes imprévus dans les documentations officielles des traitements.

3) Les documentations destinées aux programmeurs

Les documentations API

Une API (*Application Programming Interface*) est une librairie de fonctions et de structures de données servant à programmer des applications. Par opposition aux programmes dont les fonctions internes sont réputées inaccessibles, il est possible d'invoquer les fonctions d'une API. En général, les API proposent des fonctions de bas niveau permettant de programmer des applications de "haut niveau". La spécification de l'interface consiste en la donnée de la liste des fonctions et de leurs signatures. Les langages de programmation n'ont pas tous la même syntaxe de définition d'interface. Mais la volonté d'interopérabilité, liée en particulier au développement de la programmation par composant, a conduit au besoin de disposer d'un langage standard de définition d'interface. L'OMG a ainsi défini en 2002 le langage IDL (Interface Definition Language). Il est utilisé dans le cadre d'applications basées sur CORBA (Common Object Request Broker Architecture) ou sur COM (Component Object Model).

Les fichiers IDL ne constituent pas des documentations idéales. Les documentations API classiques sous forme de fichiers HTML sont plus adaptées à la consultation[15]. Générées automatiquement à partir du code, ces documentations utilisent les commentaires inclus par le développeur. Des balises réservées sont définies 2.2. On remarque notamment le mot-clé `since` qui permet de spécifier la compatibilité de la classe Java décrite. Par exemple, `@since JDK 1.3` permet de signifier l'incompatibilité de la classe avec les versions antérieures de Java. Potentiellement, un système d'aide à l'utilisateur devrait pouvoir exploiter cette information, qui mériterait alors d'être qualifiée de *connaissance*.

`author`	`version`	`see`	`since`
`serial`	`serialField`	`serialData`	`deprecated`
`link`	`return`	`param`	`throws`
`exception`			

TAB. 2.2 – Mots-clés utilisés par l'outil de génération de documentation Javadoc [Sun04]

A l'IGN comme ailleurs, les documentations API au format HTML sont appréciées des développeurs. Plus exactement, elles sont appréciées des développeurs experts qui savent à l'avance de quelle API ils ont besoin et qui en connaissent les principes généraux d'utilisation.

[15] À titre d'illustration, voir la copie d'écran figure 5.8, p. 187.

L'information manquante se réduit alors à la façon exacte d'invoquer les fonctions de l'API, *i.e.* à trouver leur nom et leur signature. Dans ce type de cas, effectivement, les documentations API classiques remplissent bien leur rôle. On peut même penser qu'elles le remplissent alors de façon optimale, puisque l'information pertinente y est représentée, et seulement celle-ci. De fait, certains développeurs de l'IGN que nous avons interrogés se sont montrés sceptiques quant à l'apport potentiel de toute autre forme de documentation. Les développeurs ayant émis cette opinion comptaient parmi les plus experts. Dans le milieu informatique, certaines pratiques sont parfois solidement ancrées. On peut ainsi encore rencontrer, par exemple, des développeurs réfractaires aux facilités des IHM, préférant l'usage des lignes de commande et les éditeurs de textes sommaires comme l'antique vi[16]. L'intérêt de toute nouvelle proposition tendant à faire évoluer les pratiques n'est pas d'avance forcément invalidée pour autant. Il existe plusieurs types de publics et plusieurs types de besoins. Peut-être parfois le chercheur, davantage conscient des enjeux de certains besoins, doit-il devancer les attentes du public. Ce faisant il prend un risque ; c'est alors seulement la pratique qui, après coup, livre son verdict sur le bien fondé du travail.

Les documentations API classiques peuvent faire l'objet de quelques critiques.

Leur manque le plus important est probablement l'**absence d'indexation sémantique**, *i.e.* d'indexation avec des termes d'un vocabulaire contrôlé. Autrement dit, les métadonnées dont nous avons besoin sont aux documentations API classiques ce que le Web sémantique est au Web actuel.

Une deuxième critique est l'**absence de centralisation inter-langages** et *inter-domaines de fonctionnalités*. Elle est illustrée figure 1.9 avec l'exemple du besoin d'aide pour le choix d'une API permettant le développement d'un programme de visualisation de MNT.

Diverses initiatives ont vu le jour pour pallier l'absence de centralisation inter-domaines de fonctionnalités. Par exemple L. Perron, de la communauté organisée autour du site `www.developpez.com`, a créé un moteur de recherche basé sur l'ensemble des documentations produites par l'outil javadoc. Il justifie ainsi son initiative :

> "Comme tout programmeur en Java, j'ai régulièrement besoin de me plonger dans la documentation. La javadoc est bien faite et plutôt claire. Par contre, il n'existe aucun moteur de recherche simple permettant de trouver facilement les classes ou les méthodes recherchées. Lassé d'utiliser le "CTRL+F" de mon explorateur pour trouver le bon mot dans la liste des 2700 classes, j'ai décidé d'indexer la totalité de la documentation java (plus de 200 Mo de fichiers html !) pour créer un moteur de recherche rapide et simple." (`http://javasearch.developpez.com/`)

Dans le même ordre idée, Franck Allimant propose sur son site[17] l'ensemble de la documentation de l'environnement standard Java (J2SE) aux formats hlp (WinHelp) et chm (Compiled Html Module)[18]. C'est également à ce format que l'on peut trouver la documentation du MSDN (Microsoft Development Network)[19] réunissant la totalité des documentations relatives au développement en environnement Microsoft.

Dans un contexte classique de programmation, ce type de mode de recherche présente un bon rapport bruit / silence : on trouve ce que l'on cherche et l'on subit moins de parasites

[16] Les éditeurs vi (pour *vi*sual) fonctionnent sur les systèmes de type Unix. L'ergonomie est limitée puisque toutes les actions s'effectuent en ligne de commande.

[17] `http://www.allimant.org/javadoc/jdk14e.html`

[18] À titre expérimental, nous avons construit un fichier d'aide au format chm avec une partie des pages de descriptions de notre application (à l'aide du logiciel Html Help Workshop 4.74). Entre autres raisons, la difficulté de mise à jour en contexte C/S et l'obligation de spécifier les mots de chaque page écartait d'emblée l'éventualité d'adoption de cette solution.

[19] Version Web sur `http://msdn.microsoft.com/library/`. "The MSDN Library is an essential resource for developers using Microsoft tools, products, and technologies. It contains a bounty of technical programming information, including sample code, documentation, technical articles, and reference guides."

qu'une recherche sur le Web entier. On dit que le *rappel* et la *précision*[20] sont satisfaisants.

Une dernière critique que l'on peut formuler à l'encontre des documentations API classiques concerne leur inadéquation partielle à certains besoins pour lesquels il y a trop ou trop peu d'informations. Par exemple l'utilisateur qui cherche simplement à se former une idée des principales fonctionnalités réalisées par une classe ou un package Java ne désire pas parcourir la liste de toutes les méthodes sans distinction de niveau de complexité ou de vocation (réalisation effective d'un traitement ou simples instructions "utilitaires" d'initialisations, de connexion à une base de données, de lecture de fichier, etc.[21]). Typiquement, les accesseurs[22] ne réalisent pas de fonctionnalités à proprement parler, ce sont des méthodes de bas niveaux. Ceci dit, on peut considérer ce problème marginal, compte tenu de l'existence souvent évocatrice des noms de méthodes, de la présence de commentaires de code[23], et, en théorie, de l'organisation thématique des packages.

Plus délicat est peut-être le problème de l'insuffisance de l'information portée par les signatures de méthodes. Pour la méthode de l'exemple du tableau 2.3, la notion de signature devrait être étendue si l'on veut pouvoir répondre aux requêtes telles que "quelles sont les méthodes qui donnent des mesures de surfaces comme résultat ?".

`void afficheAire() {`
`  system.out.println(this.aire);`
`}`
	La documentation générée par javadoc pour cette méthode ne permet pas, sur la base de sa signature, de connaître les entrées et sorties "effectives". L'utilisateur peut donc parfois avoir besoin d'une description complémentaire.

TAB. 2.3 – Signature de méthode : une description parfois insuffisante

En dépit des réflexions critiques qui viennent d'être exposées, les documentations API classiques restent incontournables. Nous retenons de leur étude la souplesse de navigation permise par la présence systématique d'hyperliens et d'index `Overview` et `Tree`. En particulier, le principe objet d'héritage se reflète dans l'organisation des pages et permet une prise de connaissance **progressive** du contenu des API (cf. fig. 5.7, p. 187).

Les fichiers log

Les fichiers *log* sont des sources d'informations sur le déroulement des traitements. Ce sont des journaux d'événements utiles surtout en cas de problème, pour le *débuggage*. Parfois les fichiers log ne sont générés que lorsque surviennent des erreurs d'exécutions, d'autres fois lorsque l'utilisateur en fait la demande[24]. Les fichiers log sont en général au simple format texte. Ils ne respectent pas de structure particulière. D'une façon générale, les informations qu'on y trouve concernent l'évaluation des traitements : par exemple la durée d'exécution, les statistiques liées

[20] Pour une requête donnée : rappel = nombre de documents pertinents trouvés / nombre total de documents pertinents ; précision = nombre de documents pertinents trouvés / nombre total de documents trouvés.

[21] Ces méthodes sont souvent privées. L'option `- public` de la commande javadoc permet de ne pas les faire apparaître dans la documentation générée. D'autres moyens existent pour configurer le comportement de javadoc, notamment les doclet (cf. 219).

[22] Simples méthodes de lecture et d'écriture des propriétés d'une classe, par exemple `x = getX() ;` et `setX(value) ;`

[23] Au niveau des packages Java, les auteurs de traitement sont invités à créer un fichier `package.html`.

[24] Par exemple, la commande Java possède l'option `-verbose`. En programmation C et C++, les développeurs prévoient également un mode "debug" que l'on choisit ou non lors de la compilation pour activer les instructions qui génèrent les traces d'exécution.

à l'identification des utilisateurs et aux fichiers ouverts, les programmes et librairies utilisés en arrière-plan, etc.

4) Outils utilisés par les programmeurs

Certains programmeurs pourraient prétendre de façon un peu provocatrice que le code d'un traitement constitue sa meilleure description. De fait, il existe beaucoup d'outils pour faciliter la lecture ou la gestion de code. Incidemment, la pratique de ces outils suggère de nouveaux éléments de description des traitements.

Logiciels et programmes d'inspection de code

L'inspection de code peut être utile pour comprendre ce que fait un traitement, comment il fonctionne... et pourquoi il ne fonctionne pas (activité de *débuggage*). Une des questions les plus fréquentes concerne la localisation d'instructions particulières de code. Certains utilitaires comme `grep` (*General Regular Expression Parser*, sélectionne toutes les lignes de code qui satisfont une expression régulière) permettent la recherche plein texte. D'autres outils exploitent la signification du code pour retrouver les appels entre traitements. Par exemple, le logiciel Dependency Walker permet de voir les dépendances entre librairies DLL sous Windows.

Certains logiciels comme Windasm permettent de désassembler les codes exécutables en langage machine, d'y retrouver les chaînes de caractères présentes et les instructions qui font référence à ces dernières. Les informations de ce type sont *a priori* de trop bas niveau pour figurer dans les métadonnées que nous souhaitons construire. Néanmoins les dépendances entre traitements, en tant que préconditions d'utilisation, doivent être décrites.

Lorsque l'on veut obtenir des informations sur les appels de fonctions effectuées par un programme ou sur l'algorithme sur lequel il repose, mais qu'on ne dispose pas de son code source et que l'analyse de son code exécutable, traduit en langage assembleur, ne s'avère pas instructif, on peut recourir à des outils de trace d'exécution comme SoftIce. Dans ce type de scénarios l'obtention des métadonnées sur les traitements demande une participation active du programmeur. Cela déborde donc un peu les objectifs que nous nous sommes fixés. Il est toutefois intéressant de noter que les compétences de *debuggage* reposent grandement sur des connaissances tacites fournies avant tout par l'expérience, connaissances qu'il sera utile de représenter.

Plus proches de nos besoins de description, des outils d'inspection de code comme par exemple JavInspector [ZCG+03] génèrent des descriptions comme celle exposée code 2.1.

Concernant l'évaluation des traitements, les informations fournies par divers logiciels au sujet des pourcentages d'utilisation du processeur et de mémoire vive utilisés constituent également des éléments de description intéressants.

Les logiciels de versioning

Dès qu'un projet informatique atteint une taille conséquente, la capacité à obtenir des informations sur l'évolution entre les différentes versions du code est cruciale. Des logiciels sont spécifiquement dédiés au suivi de versions de code : ce sont les logiciels dits de *versioning*. Ils permettent le travail collaboratif entre développeurs : pour chaque partie du projet, il est possible de savoir qui la développe, qui y a participé, quand, pour quelle version du projet. Les développeurs sont censés décrire chaque nouvelle version d'une partie de code en indiquant en langue naturelle quelles sont les modifications par rapport à la version précédente. Les logiciels de *versioning* MKS (Mortice Kern Systems) sont associés à des utilitaires de comparaison de fichier texte tels Visual Difference et Examdiff. Aux laboratoires COGIT et MATIS, le logiciel de *versioning* utilisé est CVS (Concurrent Versions System).

Les informations permettant la comparaison entre traitements constituent des métadonnées utiles non seulement pour les développeurs, mais pour tous les types d'utilisateurs. La grande

```
<class>
  <name>GM_Point</name>
  <modifier>1</modifier>
  <superClasses>
    <class>
      <name>GM_Primitive</name>
      <packageName>spatial.geomprim</packageName>
    </class>
  </superClasses>
  <methods>
    <method>
      <name>setPosition</name>
      <returnType>void</returnType>
      <params>
        <param>
          <type>DirectPosition</type>
        </param>
        <!-- ... -->
```

Extrait de code 2.1: XML – Description générée par JavInspector pour la classe GM_Point

difficulté est l'acquisition de ce type d'informations. Les logiciels de *versioning* apportent une solution dans le contexte particulier des développements de projets dotés de charte de bonne conduite.

5) Langages et modèles de conception en génie logiciel

UML

L'Unified Modeling Language est un langage normalisé par l'OMG[25] début 1997, permettant de décrire une application en fonction des méthodes objet avec lesquelles elle a été construite. Graphiquement, ces modèles sont des diagrammes : de cas d'utilisation, de classes, d'interaction (collaboration et séquence), d'état, d'activités. Le but est de fournir des spécifications claires et rigoureuses des traitements. Si le diagramme de classes prédestine plutôt UML à la description des applications développées selon une philosophie orientée objet, les autres diagrammes peuvent être utilisés de façon plus générale.

Comme le terme "application" recouvre davantage pour nous la notion d' "ensemble de traitements" que de celle de "traitement", il n'est pas sûr que les diagrammes autres que celui de classes trouvent une utilité dans le cadre de nos besoins de consultation. Le diagramme de classe, lui, peut être adapté aux traitements individuels, mais l'information représentée est alors celle déjà présente sous une autre forme dans les documentations API.

UML est une notation. UML ne fournit pas de vocabulaire pour la description des traitements, mais permet de représenter synthétiquement l'ensemble des objets manipulés par un traitement, les dialogues, les scénarios de fonctionnements. A l'IGN, un langage voisin est encore beaucoup utilisé : SADT (Structured Analysis and Design Technics).

Les réseaux de Pétri

Dans la lignée d'UML, les réseaux de Pétri sont destinés à exprimer des spécifications formelles. Leur avantage principal est l'absence d'ambiguïtés syntaxique et sémantique. Leur difficulté principale réside dans le niveau d'expertise nécessaire pour les manipuler. On associe

[25]Object Management Group. Association de professionnels de l'informatique orientée objet ayant défini la norme CORBA (Common Object Request Broker Achitecture, standard de gestion d'objets distribués rivalisant avec COM de Microsoft), ainsi que l'OMA et les ORB.

des méthodes d'analyse à ces spécifications qui permettent d'établir formellement la preuve des propriétés.

Les réseaux de Pétri sont particulièrement adaptés à l'expression du contrôle entre différents processus.

Un réseau de Pétri est composé :

- de places associées aux états du système et aux ressources,
- de transitions modélisant les différents traitements,
- d'arcs, reliant places et transitions, exprimant les dépendances entre ces types d'objets.

De places en places circulent des jetons qui représentent les instances des ressources sur lesquelles le système opère. Les descriptions sous forme de réseaux de Pétri, comme par ailleurs les descriptions reposant sur les *méthodes formelles* permettant d'établir la preuve des programmes, pourraient figurer dans les parties "fonctionnement" et "évaluation" de nos métadonnées. Mais comme leur vocation est de servir plutôt au moment de la conception des traitements qu'à celui de leur utilisation, nous n'y ferons en fait pas appel.

Les design-patterns

Les *design-patterns*, ou patrons de conception, sont des descriptions de connaissances utiles à la conception de logiciels. Les *design-patterns* sont en fait constitués de trois descriptions : celle du problème, celle de la solution et celle du contexte. Le but des *design-patterns* est en quelque sorte de capitaliser l'expérience pour des problèmes récurrents de conception. Il ne nous semble pas qu'il y ait d'éléments de description particuliers associés aux *design-patterns* qui puissent nous être utiles.

Qualité et fiabilité des traitements informatiques

La sûreté de fonctionnement d'un traitement informatique est une information utile. Plusieurs déclinaisons de la notion de sûreté de fonctionnement sont possibles [Kor99] :

- disponibilité : capacité à être prêt à délivrer le service,
- fiabilité : capacité à maintenir la continuité du service,
- maintenabilité : aptitude aux réparations et aux évolutions,
- sécurité et innocuité : absence de défaillances catastrophiques,
- confidentialité : absence de divulgation non autorisée.

La question de savoir comment établir ces propriétés n'est pas triviale et varie fortement selon les traitements et leur capacité à se prêter à des jeux de tests. Ainsi, par exemple, l'évaluation de différentes bibliothèques de programmes géométriques destinées à être intégrée dans la plateforme GeOxygene du COGIT a dû faire l'objet d'un stage de DEA [Pel03]. Les critères d'évaluation retenus y étaient la rapidité d'exécution, la qualité des résultats, la stabilité des algorithmes. A ce propos les informations telles que "fiabilité et efficacité sont contradictoires car tests et gestion des exceptions ralentissent le code" doivent être stockées dans la base de MDT. Elles font partie des méta-connaissances nécessaires à la compréhension et à l'utilisation des traitements (cf. 1.3.1).

Une difficulté de l'évaluation des traitements est la variation de leur comportement en fonction des données auxquelles ils s'appliquent. Cet aspect ne semble pas être fréquemment abordé dans les descriptions que l'on a pu rencontrer. Les méthodes formelles vues précédemment sont une solution mais leur formalisation rigoureuse ne se prête probablement pas aux traitements géographiques complexes.

6) Quelques descriptions et langages de description de traitements en IA

Langage VDL

N. Sabouret a réalisé une thèse sur un langage capable de représenter le fonctionnement de ce qu'il appelle des composants actifs [Sab02]. Il s'agit par exemple de services Web ou de robots type photocopieur. Ce langage, nommé VDL (View Design Language), unifie le code des composants actifs sous une forme à la fois exécutable (après compilation) et descriptive de son fonctionnement. Il est à la fois le formalisme de représentation interne des connaissances, le langage de programmation des composants, le modèle dans lequel leur exécution s'effectue et le support du raisonnement sur les connaissances et le fonctionnement de ces composants.

Le VDL est doté d'une expressivité permettant, en théorie, de décrire n'importe quel service. Le VDL est conçu pour servir de support aux questions générales de l'utilisateur sur le fonctionnement des services.

Au premier abord les descriptions VDL sont assez déroutantes. Elles ne comportent pas d'opérateurs usuels de programmation *if, while*, etc. On y trouve à la place des listes de variables associées à des éléments prédéfinis, le tout englobé dans l'élément racine *view* correspondant au concept grâce auquel il est possible d'accéder aux descriptions. Les éléments prédéfinis correspondent à des événements (*start, stop, slower, faster, ..*), des actions (*action*), des instructions (précondition *guard, put, get*, opération arithmétiques, ..).

Le VDL a été conçu en fonction d'un certain nombre de principes. Il nous paraît utile de retenir deux d'entre eux que nous présentons brièvement.

D'abord les éléments des descriptions VDL sont conçus pour servir de support aux mécanismes explicatifs. L'existence de chacun de ces éléments est justifié :

- soit d'un point de vue opérationnel (représentation des fonctionnalités du traitement)
- soit du point de vue de l'explication (représentation d'une information utile au raisonnement mais qui n'appartient pas à la description proprement dite du traitement)

Le VDL est un langage procédural puisque les descriptions sont clairement façonnées en vue de leur exploitation, par opposition aux descriptions déclaratives écrites indépendamment des considérations sur leur exploitation automatique future. Leur expressivité est alors meilleure.

L'objectif principal du VDL est de décrire le fonctionnement d'un traitement au cours du déroulement de son exécution. Ce dernier aspect implique une nette différence d'orientation entre les descriptions VDL et les nôtres. Certes, notre désir de rendre "intelligents" les outils de consultation tend à donner un caractère "opérationnalisable" à nos descriptions. Mais nous restons néanmoins au niveau "métadonnées", au contraire de VDL qui unifie description et code des traitements. Nous ne voulons pas atteindre le "niveau code", ce que la complexité des traitements géographiques nous interdirait probablement de toutes façons. C'est pourquoi l'état de l'art des modèles de description de fonctionnement établi par [Sab02] ne semble pas pouvoir nous être utile. Les travaux exposés (*Qualitative Process Theory*, algèbres évoluantes, logiques de réécriture, etc.) semblent surtout appropriés à la description du code et incluent des notions de temporalité ou de changement d'états éloignées de nos besoins de consultation.

En dépit de l'importance de la différence d'objectifs, certains principes de langage VDL peuvent néanmoins nous être utiles. Celui du caractère nécessairement procédural des descriptions en est un. Au moment d'aborder l'implémentation, il nous sera également profitable de nous inspirer des modalités de la communication homme-machine et du langage de requêtes proposés par VDL.

Nous retenons enfin un des principes ayant présidé à la conception du VDL. Il est couramment rencontré. Il s'agit de la décomposition successive d'actions en sous-actions, jusqu'au niveau des actions atomiques de base.

Réalisation d'un système d'aide à la conception d'applications de Traitement d'Images

Dans le domaine du TI (traitement des images), réaliser une tâche peut nécessiter d'enchaîner plusieurs dizaines d'opérateurs (programmes qui effectuent des opérations de base sur les images). La modélisation du raisonnement permettant l'enchaînement et le paramétrage des opérateurs a fait l'objet d'une thèse [FC99], dont nous reprenons ici certaines idées. Le problème de description des traitements y est abordé à travers celui de l'ordonnancement des tâches. L'optique est donc différente de la nôtre (la catégorie de requête RT définie en 1.3.5 mise à part).

L'obtention des plans de tâches s'appuie principalement sur deux idées :

- décomposition hiérarchique du problème posé en problèmes plus simples. Chaque problème ou sous-problème est associé à une tâche de TI, qui, suivant son niveau dans le plan, exprime le but recherché, la technique à employer ou l'algorithme à appliquer.
- représentation et mémorisation des connaissances d'experts grâce au raisonnement à partir de cas. Il s'agit d'une forme de raisonnement par analogie qui consiste à raisonner à partir d'expériences ou de cas déjà rencontrés pour résoudre de nouveaux problèmes. Un cas est composé de deux parties : la description du problème et la description de la solution. Une solution est modélisée sous forme d'un arbre Tâche-Méthode-Outil, qui est repéré par sa tâche racine.

La notion de *Tâche-Méthode-Outil* sur laquelle Ficet s'appuie pour décrire les solutions est associée à des définitions particulières :

> "Une tâche représente un but ou un sous-but dans le système. Une méthode décrit un savoir-faire, elle spécifie comment une tâche peut être réalisée. Un outil est la réification d'un code informatique (opérateur de TI, fonction Lisp ou C) en termes conceptuels pour l'utilisateur avec un lien sur le code pour la mise en œuvre" [FCRP99].

Les concepts de Tâche et Méthode ont un rôle heuristique, il s'agit d'informations pour l'ordonnancement des Outils qui correspondent grosso modo à nos ressources Fonctionnalité et Programme réunies. Concrètement, les classes TMO possèdent les propriétés suivantes : entrées, paramètres, sorties, résultat, etc.

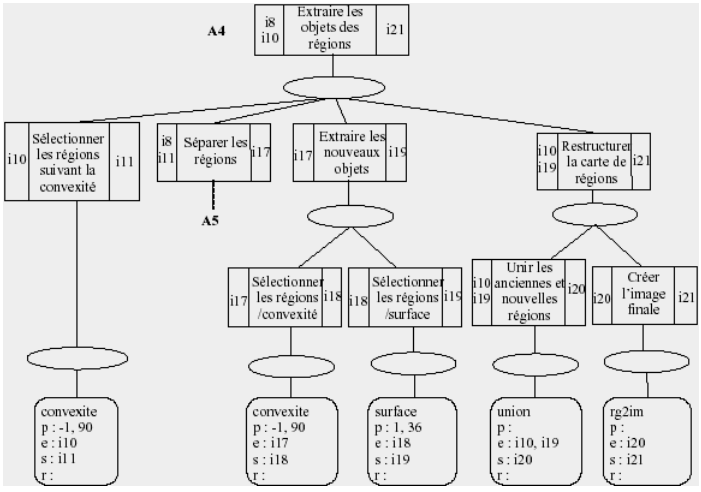


FIG. 2.2 – Architecture Tâche-Méthode-Outil (extrait de [FC99], p.157)

Dans la figure 2.2 :

- les rectangles aux coins droits représentent les tâches : ce sont les buts ou sous-buts, décrits en termes génériques indépendants du domaine de l'image (médical, surveillance optique, etc.) ;

– les ovales représentent les méthodes (*i.e.* le savoir-faire), qui décrivent l'utilisation des outils pour atteindre un objectif. Elles expriment l'expertise combinée d'experts en analyse d'image, en traitement d'image, et d'experts du domaine ;
– les rectangles aux coins arrondis représentent les outils, c'est-à-dire les algorithmes codés et paramétrables.

Concernant l'IHM du système réalisé, nous retenons le moyen par lequel l'utilisateur définit son problème. L'expression d'un besoin de traitement pourrait tout à fait employer le même procédé :

> "La définition du problème est composée d'un ensemble de mots clefs sélectionnés parmi trois listes prédéfinies : une liste de verbes décrivant les opérations effectuées par la tâche (détecter, classifier, binariser, lisser, ...), une liste de noms correspondant, soit aux objets sur lesquels l'action est effectuée (contours, régions, fond, ...), soit à la technique appliquée (variance, croissance, ...) et une liste d'adjectifs qualifiant, soit les objets sur lesquels l'action est réalisée (petit, local, ...), soit l'action elle-même (partiel, fort, ...)" [FCRP99]

Nous retenons l'idée bien connue de décomposition de buts en sous-buts, et surtout la notion d'outil qui permet de s'abstraire du code informatique et qui permet de réaliser une programmation au niveau connaissance. De façon sous-jacente nous retrouvons la nécessité de distinguer les approches "utilisateur" et "programmeur".

KADS

La méthode KADS (Knowledge Acquisition Design System) vise à modéliser les stratégies de raisonnement de l'expert d'un domaine de façon abstraite et de développer une bibliothèque d'actions génériques faisant intervenir une modélisation des connaissances stratégiques et des connaissances du domaine. KADS offre un référentiel d'organisation de la modélisation des connaissances, et une méthode permettant de développer des applications [Bar98].

Dans la philosophie KADS, une tâche est une partie d'un travail devant être accompli par un agent humain ou logiciel, et se rapproche de notre concept de traitement en ce qu'elle se caractérise par une entrée et une sortie. Par exemple la tâche de diagnostic prend la spécification d'un problème et fournit sa catégorisation et des propositions de solutions.

Dans KADS il existe deux types de tâches : les primitives et les composites. Une tâche composite est décrite d'une part par la spécification de ses entrées/sorties, d'autre part par une méthode de tâche qui spécifie sa décomposition en tâches primitives. Tâche et méthode de tâche correspondent respectivement au "quoi" (qu'est-ce qui est fait), et au "comment" (comment est-ce fait). Comme les tâches composites, les tâches primitives possèdent des entrées/sorties. Elles sont soit des inférences, c'est-à-dire des pas élémentaires de raisonnement, soit des fonctions de transfert, c'est-à-dire des fonctions d'échange d'information entre l'agent raisonnant et le monde extérieur [SAA+00]. Dans KADS, l'expression de préconditions est également prévue.

Les quelques principes évoqués, qui font de KADS une méthode connue d'acquisition des connaissances, pourront être adaptés à notre contexte particulier et utilisés pour la conception de notre modèle.

7) Langages et modèles de description de services Web

Pour permettre le dialogue entre deux programmes, s'exécutant éventuellement sur des machines distantes, de nombreux protocoles et langages informatiques ont été développés. Dans le cas d'un dialogue empruntant le réseau Internet, le programme qui joue le rôle de fournisseur de données est communément nommé *service Web*[26]. Cette appellation peut être jugée inexacte

[26]Définition W3C : "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL).

car elle repose sur la confusion entre Web et Internet[27]. En effet le Web n'est que la sous-partie d'Internet basée sur le protocole HTTP[28]. La messagerie e-mail, basée sur POP et SMTP, en est une autre, l'échange de fichier basé sur FTP une autre encore, etc. L'inexactitude du terme *service Web* vient par exemple de ce que les programmes des serveurs de messagerie répondent à la définition de service Web ci-dessus énoncée, mais ne reposent pas sur le protocole HTTP qui caractérise le Web.

Quoiqu'il en soit, les protocoles, langages et outils visant à permettre la mise en œuvre de services Web constituent un des principaux chantiers actuels dans le monde de l'informatique. Les raisons de sécurité constituent un des obstacles au développement de systèmes à l'architecture orientée service, mais également le manque de recul sur les langages de description. Nous allons brièvement en étudier quatre : **WSDL** qui décrit ce que font les services Web, **SOAP** qui décrit le format des messages échangés par les services Web, **UDDI** qui permet l'indexation des services Web dans des annuaires, et enfin **OWL-S** qui tend à compléter le trio WSDL-SOAP-UDDI.

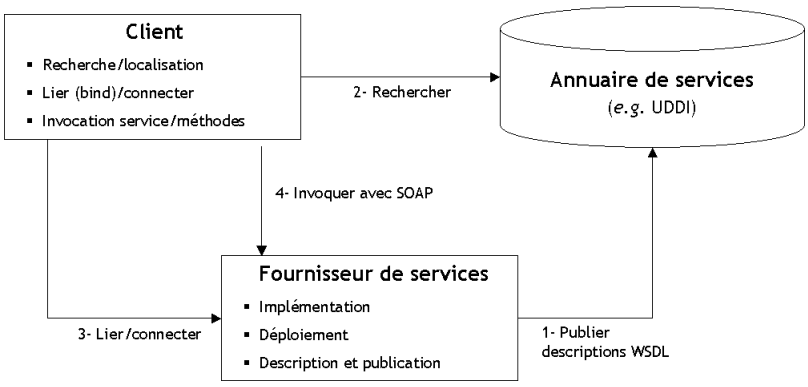


FIG. 2.3 – Architecture des services Web (adapté de [KT03])

WSDL

WSDL (Web Services Description Language) est, comme son nom l'indique, un langage de description de services Web. Il définit de manière abstraite et indépendante du langage l'ensemble des opérations et des messages qui peuvent être transmis vers et depuis un service Web donné. C'est actuellement un rapport technique W3C (statut précédant celui de recommandation) [W3C01].

WSDL décrit la façon d'accéder aux services. Tandis que OWL-S se situe plutôt à un niveau de description abstrait, WSDL décrit concrètement l'interface des fonctions publiques, le type de donnée des requêtes et réponses, la liaison sur le protocole de transport utilisé, l'adresse du

Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards" (`http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis`).

Dans un des ouvrages consacrés aux sevices Web on trouve également cette définition : "les services Web (...) interagissent via les technologies standard d'Internet" [ABC+03], p.16.

[27] "On the Net, you find computers – on the Web, you find document" (T. Berners-Lee, `http://www.w3.org/People/Berners-Lee/FAQ.html`)

[28] Créé en 1990 par Tim Berners-Lee, l'Hypertext Transfer Protocol est comme son nom l'indique initialement dédié au transfert de documents HTML (`http://www.commentcamarche.net/histoire/internet.php3`).

Il se trouve qu'il est maintenant détourné de son but initial puisqu'il est également utilisé pour l'échange de données via la méthode POST. Nous employons nous-mêmes ce procédé dans notre application pour permettre à l'utilisateur d'enregistrer des échantillons graphiques (cf. p. 215). La méthode POST est également utilisée de façon préférentielle par le protocole SOAP pour l'échange de messages entre services Web [SOA03].

service, etc. WSDL étant implémenté en XML, ses éléments sont décrits dans un schéma XSD[29].

WSDL se situe au niveau implémentation des interfaces de services. Cet aspect concerne une partie des besoins de consultation que l'on se propose de satisfaire, mais une partie seulement. Il sera utile d'associer WSDL à nos descriptions des services Web. Cela ne le sera pas pour les autres types de traitements.

SOAP

SOAP est l'acronyme de Simple Object Access Protocol (Protocole Simple d'Accès aux Objets). SOAP est une recommandation du W3C qui définit un protocole d'échanges de message entre clients et services Web [W3C03c].

Une des principales caractéristiques de SOAP par rapport aux autres protocoles au but similaire est d'être basé sur XML. Cela signifie que les messages échangés sont lisibles : ils sont constitués de texte et non de flux binaires[30].

```
<-- Requête -->
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:getFeatureType
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://essaiJGraph.essaiGraph.consul"/>
 </soapenv:Body>
</soapenv:Envelope>

<-- Extrait de la réponse -->
<soapenv:Envelope ...>
 <soapenv:Body>
  <ns1:getFeatureTypeResponse ... >
   <getFeatureTypeReturn href="#id0"/>
  </ns1:getFeatureTypeResponse>
  <multiRef id="id0" soapenc:root="0"
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      soapenc:arrayType="xsd:anyType[48]" xsi:type="soapenc:Array"
      ...>
   <multiRef xsi:type="soapenc:int">1</multiRef>
   <multiRef xsi:type="soapenc:string">tronçon de route</multiRef>
   <multiRef xsi:type="soapenc:string">portion connexe de route, de
chemin, de piste cyclable ou de sentier, homogène pour les relations
la mettant en jeu et pour les attributs qu'elle porte</multiRef>
```

Extrait de code 2.2: SOAP – Exemple de requête et de réponse lors de l'invocation de l'opération `getFeatureType` d'un service Web du COGIT pour la diffusion d'informations sur les schémas de base de données géographiques

[29] `http://schemas.xmlsoap.org/wsdl/2003-02-11.xsd`

[30] Avec certains protocoles le format des messages échangés est binaire. On peut citer par exemple IIOP (Inter-ORB Protocol, pour la communication entre objets JAVA et CORBA), ORPC (Object Remote Procedure Call) et JRMP (Java Remote Method Protocol utilisé par RMI – Remote Method Invocation pour les objets Java distribués).

Pour employer une métaphore avec le courrier postal, on peut dire que SOAP définit une partie de la façon dont les enveloppes de lettres doivent être formées[31]. En ce sens SOAP est un protocole utilisé au-dessus de l'ensemble des protocoles de transport de plus bas-niveau du modèle OSI[32].

SOAP présente un intérêt pour notre objectif de description des traitements pour deux raisons.

D'abord, parce que certains traitements géographiques – encore très peu actuellement dans le contexte de l'IGN, le code 2.2 étant un des rares exemples – sont des services Web invocables avec SOAP. Les descriptions de ces traitements doivent donc comporter les instructions nécessaires à l'utilisation du protocole. En l'occurrence, que faut-il savoir et de quoi faut- il disposer ? Il faut connaître le nom du service Web souhaité, le nom de l'opération à invoquer, ses paramètres, etc., toutes informations classiquement fournies par la description WSDL du service, et que nos descriptions doivent également comporter. Pour être exploitables par l'utilisateur, ces informations doivent être accompagnées de connaissances[33] sur SOAP : comment développer un programme client, quelles API utiliser, comment établir une connexion Internet, etc.

La deuxième raison pour laquelle l'étude de SOAP peut être utile à notre réflexion sur la description des traitements concerne les modes d'emplois en général. En effet, nous voulons décrire les modes d'emploi des traitements de façon plus formelle que la langue naturelle, même si nos descriptions s'adressent à des humains et n'ont pas pour vocation l'invocation automatique.

UDDI

Une fois qu'un service Web est développé, il faut publier sa description dans un catalogue UDDI (Universal Description, Discovery and Integration) afin que les utilisateurs potentiels puissent le trouver.

> "Le protocole UDDI est une plateforme destinée à stocker les descriptions des services Web disponibles, à la manière d'un annuaire de style "Pages Jaunes". Des recherches sur les services peuvent être effectuées à l'aide d'un système de mots-clés fournis par les organismes proposant les services. UDDI propose également un système de "Pages Blanches" (adresses, numéros de téléphone, identifiants...) permettant d'obtenir les coordonnées de ces organismes. Un troisième service, les "Pages Vertes", permet d'obtenir des informations techniques détaillées à propos des services et permettent de décrire comment interagir avec les services en pointant par la suite vers un PIP RosettaNet ou une description WSDL." [BCES04]

Comme le montre la figure 2.4, une description UDDI doit contenir quatre catégories de données [OAS02] :

- *businessEntity* (entité d'affaires) : information sur le fournisseur du service.
- *businessService* (service d'affaires) : information décrivant une famille particulière de services techniques.
- *bindingTemplate* (modèle de rattachement) : information technique sur les points d'accès aux services et leur implémentation.
- *tModel* : (modèle *t*echnique) description des spécifications des services.

Le modèle de description UDDI semble très orienté "commerce" (ventes, prestations), donc *a priori* éloigné des besoins de consultations de traitements géographiques. Si aucun élément de description UDDI en particulier ne nous est apparu pouvoir être réutilisé dans notre contexte,

[31] Le schéma XML de SOAP se trouve à l'adresse `http://schemas.xmlsoap.org/soap/envelope/`.
[32] cf. `http://fr.wikipedia.org/wiki/TCP/IP`
[33] Nous avons là une illustration de la distinction entre information et connaissance décrite p. 27.

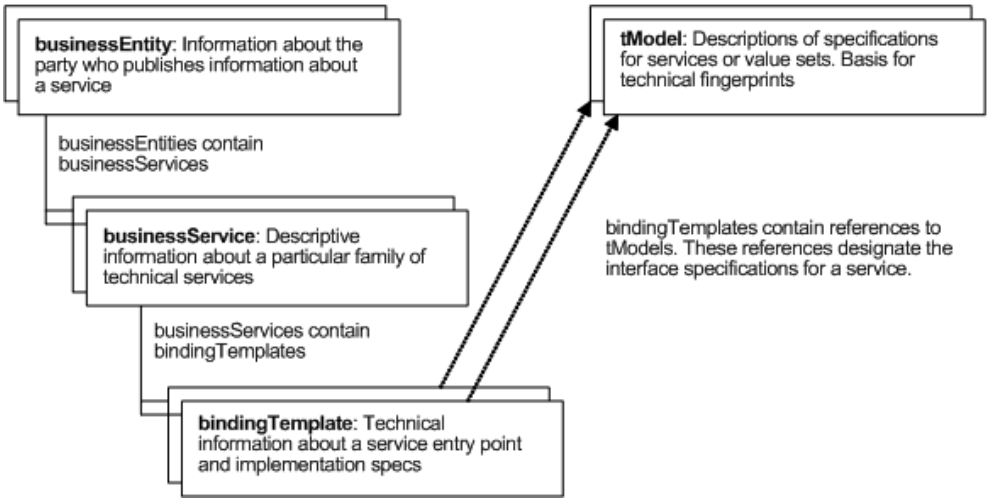


FIG. 2.4 – Structures de données "noyau" d'UDDI (extrait de [OAS02])

quelques principes, par ailleurs bien connus, peuvent en revanche être retenus.

L'attribution d'identifiants aux ressources (*businessKey*, *serviceKey*) facilite les tâches d'indexation et la recherche. Les *tModel* permettent par exemple d'identifier et de réutiliser des connaissances sur des domaines tels que les codes postaux américains et les produits industriels d'Amérique du Nord.

Les annuaires UDDI existent sous une forme XML. Le modèle de description UDDI est donc défini dans un schéma XML[34]. Des API sont disponibles pour interroger les annuaires UDDI et y référencer des services Web [OAS02].

OWL-S

OWL-S est l'acronyme de Ontology Web Langage – Services. OWL-S est une ontologie dédiée à la description des capacités et des propriétés des services Web[35]. Le but d'OWL-S est de permettre l'automatisation de la recherche, de la découverte, de l'invocation et de l'interconnexion des services Web. OWL-S fournit des éléments de descriptions et spécifie les relations entre ceux-ci. A ce titre, OWL-S peut être vu comme un modèle de description de services Web[36].

Le projet de cette ontologie des services a officiellement vu le jour au cours de l'année 2000[37]. L'ontologie portait alors le nom de DAML-S car elle était exprimée dans le langage DAML+OIL (Darpa[38] Agent Markup Language + Ontology Inference Layer).

L'organisation créatrice de l'ontologie, regroupement d'universités et d'industriels[39], s'appelait initialement "DAML-S coalition" ; elle a été renommée "OWL-S coalition".

OWL-S est complémentaire par rapport au trio WSDL-SOAP-UDDI. En effet, les des-

[34] `http://www.uddi.org/schema/uddi_v3.xsd`

[35] Nous préciserons au chapitre 3 le sens du terme *ontologie* dans le contexte particulier de la représentation des connaissances (section 3.2.1). Le langage OWL sera lui présenté au chapitre 4 (section 4.2.2).

[36] Ontologie et modèle de documents structurés ne sont néanmoins pas la même chose. Nous discuterons de la distinction, p. 128.

[37] `http://www.daml.org/2001/02/horus-daml/slide3-0.html`

[38] DARPA est lui-même l'acronyme de *Defense Advanced Research Projects Agency*. Il s'agit de l'agence du ministère de la Défense américain chargée des projets de recherche militaire, à qui l'on doit notamment l'Arpanet, ancêtre d'Internet, et le système de positionnement GPS (*Global Positioning System*).

[39] Leur liste peut être trouvée sur la première page de [Coa03].

criptions OWL-S sont plutôt portées sur le niveau sémantique abstrait des services Web (ce qu'ils font, comment ils fonctionnent). Par opposition, WSDL et SOAP sont liés au niveau implémentation des protocoles de communication. Pour ce qui est du rapport aux annuaires UDDI, Sycara *et al.* mentionnent l'existence de catalogues UDDI intégrant des descriptions OWL-S ([SPAS03], cité par [Lew04] diap. 9) En terme d'architecture, une telle décision implique une complexité croissante dans le processus de recherche (par rapport aux quatre étapes exposées fig. 2.3) puisqu'il faut y introduire une base de données contenant les ontologies auxquelles font référence les descriptions de services, et un serveur permettant de mener les raisonnements OWL.

C'est l'ambition de la communauté du Web sémantique que de mettre en place de telles architectures ; les travaux sont actuellement en cours. Il n'existe pas encore, à notre connaissance, d'outils disponibles complets pour qui veut profiter des potentialités d'OWL-S.

La classe parente de l'ontologie OWL-S s'appelle Service (fig. 2.5). Une instance de cette classe est décrite par trois concepts [Coa03].

ServiceProfile : qu'est-ce que le service fournit aux agents (ici des programmes clients) qui l'invoquent ? Qu'est-ce qu'il attend d'eux ?

Le *ServiceProfile* permet aux agents de découvrir et d'identifier un service. Il donne entre autre le nom du service, son niveau de qualité, le type de service rendu, mais également des préconditions à la fourniture de ce service, comme "avoir une carte bleue valide" ou "posséder des données géographiques avec la bonne projection" (cf. tab. 2.4 et 2.5)

ServiceModel : comment fonctionne-t-il ? Quel est son modèle d'exécution ?

Le *ServiceModel* permet aux agents de composer plusieurs services afin de résoudre un problème complexe, ou encore de surveiller le fonctionnement d'un service et d'établir des diagnostics en cas de défaillance. Le modèle d'exécution est décrit à l'aide de la classe *ProcessModel* qui fournit une ontologie des processus (cf. fig. 2.7).

ServiceGrounding : comment y accéder ? (descripteurs WSDL).

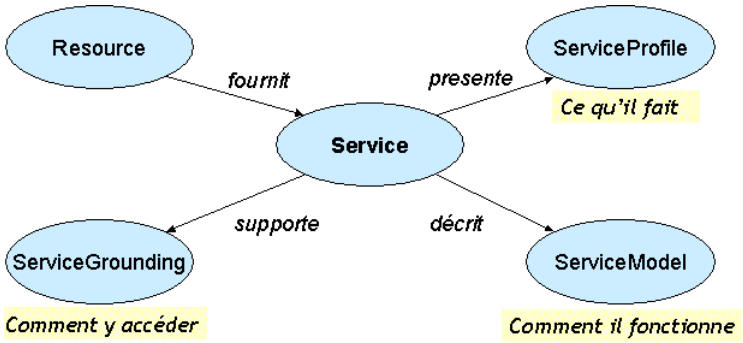


FIG. 2.5 – Niveau supérieur de l'ontologie OWL-S (d'après [Coa03])

nom élément	**description**
serviceName	nom du service, pouvant servir d'identifiant
textDescription	description du service, de ce qu'il offre et des fonctionnalités pouvant être invoquées
contactInformation	référence, vers les responsables du service (vers une description FOAF par exemple)

TAB. 2.4 – Service Profile – Informations générales (d'après [Coa03])

nom élément	description
hasInput	entrée
hasOutput	sortie
hasPrecondition	précondition
hasEffect	effet

TAB. 2.5 – Service Profile – Description "fonctionnelle" (d'après [Coa03])

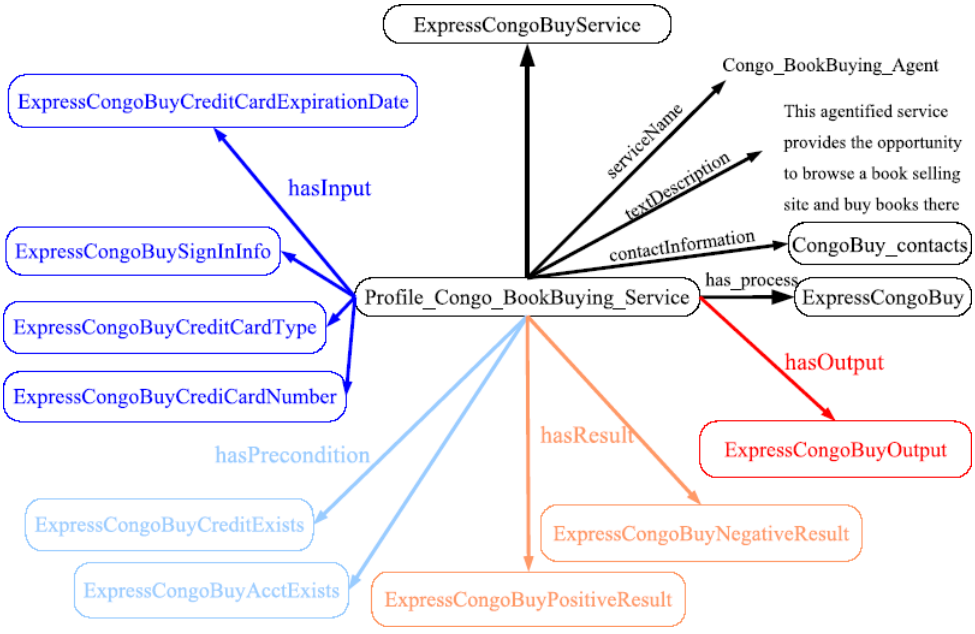


FIG. 2.6 – Description OWL-S d'un service Web d'achat de livre (tiré de [Cho05])

Voici un exemple de description de service d'achats de livres en ligne :

La figure 2.6 décrit le *profile* d'un service, *i.e.* ce qu'il fait. OWL-S permet également de décrire comment fonctionne un service : c'est le rôle de la classe *ProcessModel*, sous-classe de *ServiceModel*. Les services sont modélisés comme des traitements[40] qui peuvent être atomiques ou composites (cf. fig. 2.7).

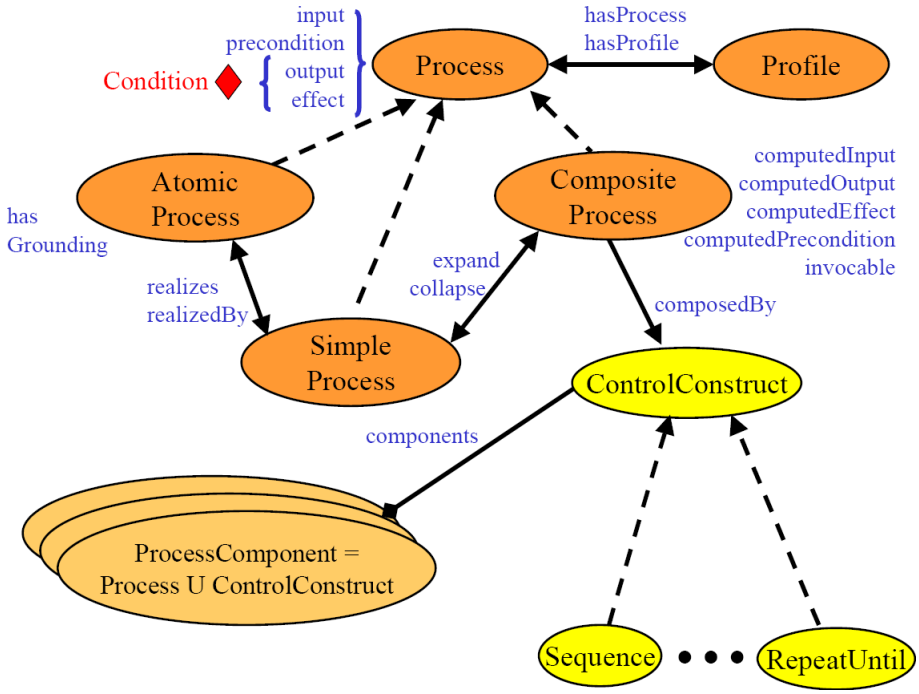


FIG. 2.7 – Description du *ProcessModel* OWL-S (tiré de [Coa03])

Les traitements atomiques (*AtomicProcess*) sont directement invocables au moyen des messages appropriés. Du point de vue du client, ils s'exécutent en une seule étape : envoi de requête puis réception de la réponse. Les traitements atomiques doivent fournir les informations sur les arguments attendus (service grounding).

Les traitements simples (*SimpleProcess*) ne sont pas soumis à cette dernière contrainte, mais ils sont également conçus pour s'exécuter en une seule étape.

Les traitements composites (*CompositeProcess*) sont, comme leur nom l'indique, eux-mêmes composés de traitements. Leur décomposition est définie à l'aide de constructeurs de contrôle présentés tableau 2.6.

sequence	split+join	if-then-else
concurrent	unordered	repeat-until
split	choice	iterate

TAB. 2.6 – Constructeurs de contrôle OWL-S

Pour illustrer l'utilisation de la classe *ProcessModel*, imaginons l'existence d'un service Web de généralisation de bâtiments (tab.2.7). Ce service Web serait composé de plusieurs traitements :

Pour décrire les services Web, la coalition OWL-S définit, en complément des classes qui viennent d'être exposées, deux ontologies dédiées respectivement aux ressources et au temps.

[40] Dans ce contexte, nous traduisons *process* par *traitement*.

process	généralisation d'un groupe de bâtiments
se décompose en :	
atomic process	déplacement d'un bâtiment individuel
atomic process	généralisation d'un bâtiment individuel
atomic process	élimination d'un bâtiment individuel
a pour structure de contrôle :	
controlConstruct	repeat(.., .., ..) until (satisfaction contraintes)

TAB. 2.7 – Exemple de description OWL-S Process : généralisation de groupe de bâtiments

Les principales propriétés de l'ontologie pour les ressources sont présentées tableau 2.8[41].

Dans l'ontologie OWL-S dédiée à l'expression des relations temporelles, on trouve, entre autres, les classes *Instant* et *Interval*, et les propriétés *start-of, end-of, inside, at-time, during*[42].

allocation Type	consumable	gone after it is allocated (carburant, argent, temps)
	reusable	available after released (bande passante, ligne de métro)
capacity Type	discrete	resource has notion of granularity (nombre de chaises occupées)
	continuous	allocate any amount (volume de carburant)

TAB. 2.8 – Ontologie pour les ressources

OWL-S : ce qu'on peut retenir

Au regard de nos besoins, OWL-S constitue la partie de l'état de l'art qui présente le plus d'intérêt : en tant que **source d'inspiration de notre modèle**, et en tant que **projet de représentation opérationnelle des connaissances**.

Concernant le modèle, nous reprenons à notre compte l'idée de réifier sous forme de classe les trois facettes de description *profile, grounding* et *model*.

Nous retenons également la façon de décrire la décomposition des étapes de fonctionnement à l'aide de structures de contrôle. Ceci posé, il est clair que notre modèle doit comporter d'autres éléments de descriptions que ceux proposés par OWL-S. En effet :

- Les services Web ne sont qu'une forme de traitement parmi d'autres. Ces autres formes possèdent des spécificités qui nécessitent des descripteurs particuliers (par exemple la façon dont on invoque les fonctions des logiciels ne se décrit pas avec WSDL et SOAP, mais avec des descriptions d'IHM).
- La vocation des descriptions OWL-S est l'automatisation de la recherche et de l'invocation de services Web ; ces descriptions sont donc avant tout conçues pour des machines. L'objectif de concevoir un système d'information destiné aux humains implique d'autres éléments de descriptions (par exemple l'inclusion d'images pour illustrer graphiquement ce que fait un traitement).
- Au delà de l'opposition humains/machines des destinataires des descriptions, l'analyse des besoins de notre contexte a révélé des besoins d'informations qui portent sur des aspects qui dépassent le cadre d'OWL-S (par exemple sur la description fine des effets des traitements ou sur l'évaluation de ces derniers).

Concernant les principes de représentation des connaissances sur lesquels repose OWL-S,

[41] `http://www.daml.org/services/daml-s/0.9/Resource.daml`
[42] `http://www.ai.sri.com/daml/ontologies/time/Time.daml`

nous retenons comme leçon essentielle qu'OWL est un langage adapté aux raisonnements nécessités par les besoins de recherche de traitements où expression de la requête et description de traitements ne correspondent pas exactement. D. Lewis explique ainsi pour montrer l'intérêt d'OWL-S que [Lew04] :

> "Generally a match requires :
>
> – Advertised outputs to be equivalent or more general than outputs of requested service ;
> – Requested inputs to be equivalent or more general than inputs of the advert."

C'est précisément en raison de ce type de besoins qu'au chapitre 1, nous nous sommes fixés pour objectif de construire non pas seulement un simple SI mais un SBC. OWL-S nous indique une voie possible : celle des ontologies formelles basées sur les logiques de description. Au-delà du choix de ce type de représentations des connaissances, dont nous discuterons au chapitre 3, ce qu'il faut noter c'est la proximité des problématiques de notre contexte et de celle d'OWL-S. Subséquemment, nous sommes amenés à adopter la même démarche que celle mise en œuvre dans le cadre du Web sémantique : concevoir des métadonnées sous une forme rendant possible l'automatisation de raisonnements.

2.1.3 Métadonnées des traitements informatiques géographiques

Sommaire

Dans le domaine de l'information géographique il existe des normes pour décrire les données, mais peu encore pour décrire les traitements.

Le comité technique de l'ISO dédié à l'information géographique et à la géomatique[43] propose un modèle, l'ISO 19119, mais l'ensemble des descripteurs que fournit ce dernier est trop sommaire pour nos besoins.

1) Normes et standard

ISO 19119

ISO 19119 est la norme produite par l'ISO/TC211 (comité technique TC 211 de l'ISO sur l'information géographique et la géomatique) pour la description des services. Elle est encore en cours d'élaboration. Son but est de fournir un cadre pour le développement des services géographiques. Être conforme ISO 19119, c'est fournir un service implémentant l'interface proposée et satisfaire deux types de contraintes portant sur l'architecture et la spécification du service [ISO01b].

ISO 19119 ne donne aucune indication sur la forme que peut prendre sa mise en œuvre ; le consortium OpenGIS (OGC) se charge de l'implémentation et de la définition des interfaces logicielles.

Les deux parties principales de la norme ISO 19119 sont constituées par les diagrammes de classes (schémas des métadonnées sous forme de diagrammes UML) et par le catalogue des objets (Data Dictionary). Ces deux parties sont spécifiées dans les annexes A et B de [ISO01b]. L'ensemble des éléments de descriptions ISO 19119 est assez succinct (cf. fig. 2.9) ; les règles régissant l'extension du modèle sont décrites dans l'annexe C de [ISO01b].

La structure d'une description d'un service comprend 3 classes principales :

- description générale du service *SV_ServiceIdentification*
- description des opérations *SV_OperationMetadata*
- description des entrées et sorties du service *MD_DataIdentification*

Ces trois classes correspondent respectivement aux niveaux de descriptions 1, 1+ et 0+ illustrés figure 2.8. Le diagramme de classe de la figure 2.9 montre les classes en question, ainsi que *SV_Parameter* et *SV_OperationMetadata*. La classe *MD_DataIdentification* (niveau 0+ figure 2.8) décrit les données ; elle appartient à ISO 19115.

[43] `http://www.isotc211.org/`

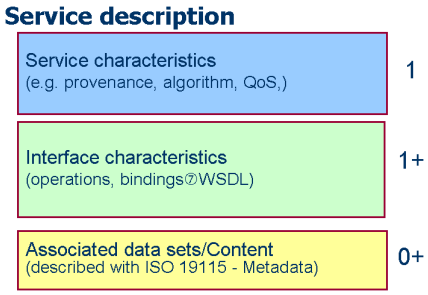


FIG. 2.8 – Trois niveaux de description des services Web selon ISO 19119 (extrait de [Per02])

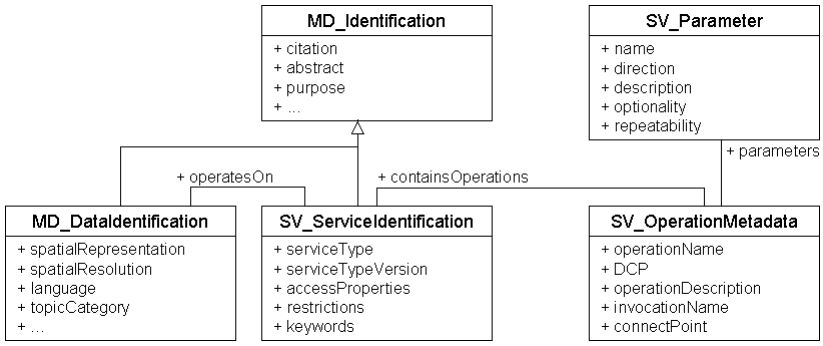


FIG. 2.9 – Les classes principales ISO 19119 pour la description d'un service (d'après [ISO01b])

Comme le montre le tableau 2.9, ISO 19119 prévoit que les consultations des descriptions de service puissent s'effectuer selon quatre points de vue différents : *Computational viewpoint, Information viewpoint, Engineering viewpoint et Technology viewpoint*. Ces points de vue correspondent à ceux définis par ISO/IEC 10746 (Reference Model – Open Distributed Processing).

L'OGC a réalisé une implémentation RDF (Resource Description Framework, cf. p. 153) de la taxinomie des services géographiques du point de vue "information" [Con01b] (cf. tab. 2.9). Cette démarche rejoint celle d'OWL-S : l'adoption de ce langage de représentation des connaissances ouvre des perspectives pour l'exploitation "intelligente" des descriptions.

OGC – Basic Service Model

À partir de la norme ISO 19119, l'OGC spécifie la façon d'implémenter les interfaces des services. Quatre principaux types de services d'accès à l'information géographique ont été définis [Con01a] :

- WMS (Web Map Server) fournit des cartes.
- WFS (Web Feature Service) donne accès à différents type de données : GML, *Web Feature Specification, Filter Encoding Specification, Feature Identifier Specification, et Transaction Encoding Specification*, etc. Pour accéder à ces différents types de données, WFS définit plusieurs opérations : *DescribeFeatureType, GetFeature, LockFeature, Transaction*, etc.
- WCS (Web Coverage Server) donne accès aux valeurs numériques associées aux données raster.
- WRS (Web Registry Server) est un catalogue de service Web.

Tous les services implémentés selon les principes de l'OGC doivent proposer au moins la méthode *GetCapabilites* qui envoie au client un document XML les décrivant. Cette méthode ressemble donc à celle nommée *QueryInterface* obligatoirement présente dans les objets COM.

Computational viewpoint	*Chaînage des services* : Defining Services with reusable interfaces Service Metadata (cf. fig. 2.9) Service/Data coupling Service Chaining
Information viewpoint	*Intéropérabilité sémantique* : Une taxinomie de services géographiques est définie : – éditeurs et *viewers* de données destinés aux humains, – simples fournisseurs de données (vecteur, raster ou texte), – services réalisant des traitements (spatiaux, thématiques, temporels, etc.), – services utiles à la communication (p.ex. encodage), – etc.
Engineering viewpoint	*Distribution des services* : Cette classification proposée par l'OGC des services par catégories recoupe celle du point de vue "information", mais du point de vue de la mise en œuvre : services avec IHM ou non, entre services effectuant des traitements ou délivrant des informations, etc.
Technology viewpoint	*Spécification des services et de leur plateforme* : protocoles pour l'interropérabilité des services (DCP – Distributed Computing Platform).

TAB. 2.9 – Description d'un service Web géographique selon les points de vue ISO/IEC 10746

Les réponses apportées aux requêtes *GetCapabilites* ne décrivent pas à proprement parler des traitements, mais les possibilités d'obtention de données proposées par les services WMS, WFS et WCS. Ainsi par exemple, la fonctionnalité *getMap* offerte par un WMS n'effectue pas réellement de transformation de l'information, et la façon dont elle est décrite présente un intérêt limité dans le cadre de notre travail.

Dans l'exemple ci-dessous (code 2.3) de description simplifiée d'un service WMS offrant des cartes de l'ozone, on peut notamment retenir l'indication des formats disponibles, le protocole de transmission, les différentes couches que comportera la carte demandée.

L'OGC propose des règles pour l'implémentation des catalogues de services Web géographiques (WRS). Le but est de standardiser les interfaces des services de catalogage de services. Cette initiative peut être comparée à celle d'UDDI.

ISO 19115

ISO 19115 définit un modèle de métadonnées des données géographiques. Décrire ces dernières, c'est, entre autres, décrire les traitements qui les ont produites. C'est-à-dire décrire leur généalogie (*lineage*). Le diagramme figure 2.10 montre les classes dévolues à cet aspect des métadonnées[44]. Nous ne nous intéressons ici qu'à la partie d'ISO 19115-2 qui étend ISO 19115.

On y voit que les traitements (LE_Processing) implémentent des algorithmes (LE_Algorithm) ; ce sont des ressources distinctes. Plus original est le choix de représenter ce qu'on appelle communément les fichiers *log* (LE_ProcessingReport) qui décrivent le déroulement des traitements et que l'on a déjà évoqués. La production des données s'effectue rarement en une seule étape, ceci explique la décomposition de la description de la généalogie en étapes

[44]Signification des préfixes employés : MD (Metadata), CI (Citation), DQ (Data quality), EX (Extent), LI (Lineage), MI (Metadata for imagery and gridded data) et LE (Lineage extended).

```
<WMS_Capabilities>
  <Capability>
    <Request>
      <GetMap>
        <Format>image/gif</Format>
        <Format>image/png</Format>
        <Format>image/jpeg</Format>
        <DCPType>
          <HTTP>
            <Get>
                <OnlineResource
                         xmlns:xlink="http://www.w3.org/1999/xlink"
                         xlink:type="simple"
                         xlink:href="http://hostname:port/path ?"/>
            </Get>
          </HTTP>
        </DCPType>
      </GetMap>
    </Request>
    <Layer>
      <Layer>
        <Name>ROADS_RIVERS</Name>
      </Layer>
      <Layer>
        <Title>Weather Forecast Data</Title>
      </Layer>
      <Layer>
        <Title>Global ozone distribution (1992)</Title>
      </Layer>
      <Layer>
        <Title>World population, annual</Title>
      </Layer>
    </Layer>
  </Capability>
</WMS_Capabilities>
```

Extrait de code 2.3: XML – Extrait d'une description de service WMS offrant des cartes de l'ozone [Con03]

(LI_ProcessStep). Le présent diagramme concerne les données dont la source est de type raster, d'où la spécification de la résolution (propriété de LE_Source).

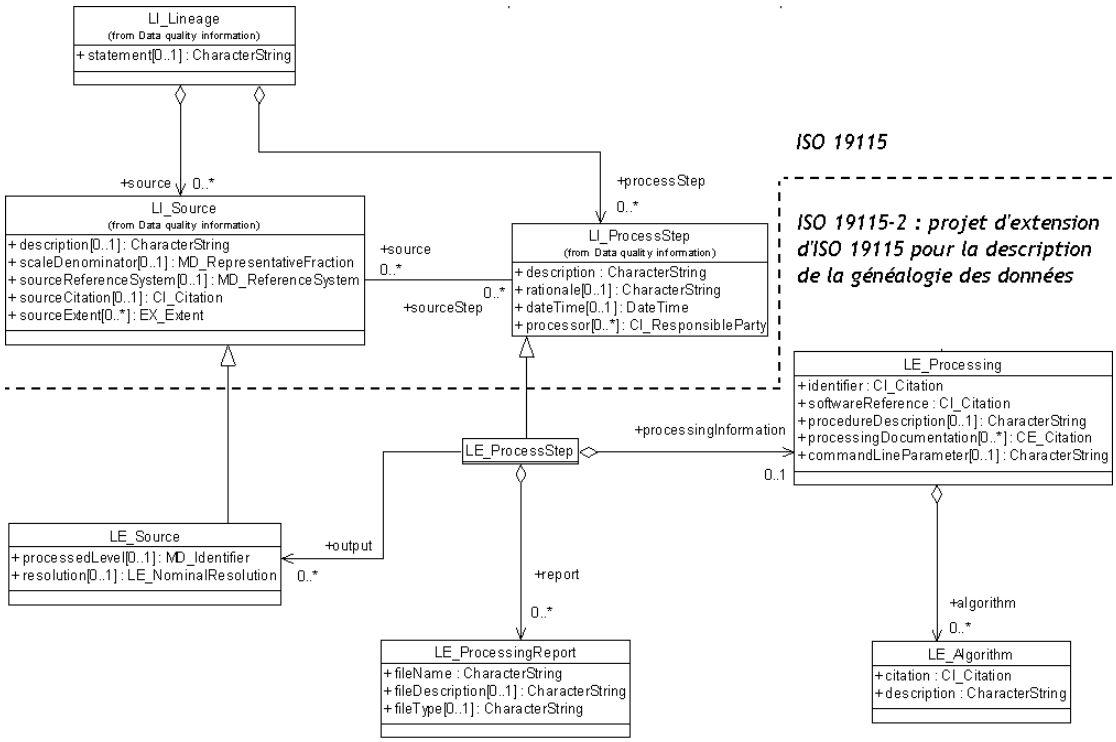


FIG. 2.10 – ISO 19115 – Description des traitements, indice de la qualité des données géographiques produites – Généalogie ("Data quality information – Lineage" [ISO05a])

2) Travaux institutionnels ou industriels

En marge des normes ISO et standards OGC qui viennent d'être présentés, de nombreux travaux institutionnels ou industriels sont consacrés à la description des services Web géographiques. Nous évoquons ici quelques-uns d'entre eux. Nous nous intéressons également aux travaux visant à décrire les traitements informatiques sous forme de SIG ou de programmes traditionnels.

Notre choix parmi les travaux existants est plus ou moins arbitraire ; nous tentons simplement de dégager les tendances actuelles du domaine, de nous positionner par rapport à elles, et le cas échéant, de nous en inspirer.

Travaux relatifs à la description des services Web géographiques

Les sites Web (Google Earth[45], Mappy[46], etc.) donnant accès aux données géographiques connaissent un grand succès. Les clients de ces sites utilisent des navigateurs Web standard ; ils ne dialoguent pas avec les services au moyen de protocoles comme SOAP. Les vrais services Web possèdent une interface WSDL ; leurs clients sont des programmes qui reçoivent autre chose que des pages HTML. Dans le domaine géographique ces services Web existent mais les architectures logicielles sont encore en gestation. Les nombreuses communications sur ce sujet dans les conférences le montrent.

[45] http://earth.google.com/
[46] http://www.mappy.com/

Par exemple, Neun et Burghardt proposent un service Web de généralisation [NB05]. L'utilisateur y accède au moyen d'un client SOAP qui se présente sous la forme d'un plug-in à intégrer à la plateforme JUMP[47]. Les données géographiques à généraliser sont converties au format GML et encapsulées dans les messages SOAP. On voit qu'isolément, les techniques pour invoquer les services sont au point.

Ce que relèvent divers auteurs comme [Tso02][LdB02], c'est la difficulté à mettre en œuvre des scénarios où interviennent de façon planifiée différents services.

Pour illustrer l'intérêt du système d'information qu'il propose[48], M-H. Tsou prend l'exemple d'une recherche d'épicerie située dans une zone résidentielle non-inondable du Colorado. Apporter une réponse à cette recherche nécessite d'interroger des bases de données distinctes, puis d'invoquer un service Web acceptant les différents formats de données et capable d'intégrer les données dans un même système de coordonnées (cf. fig. 2.11).

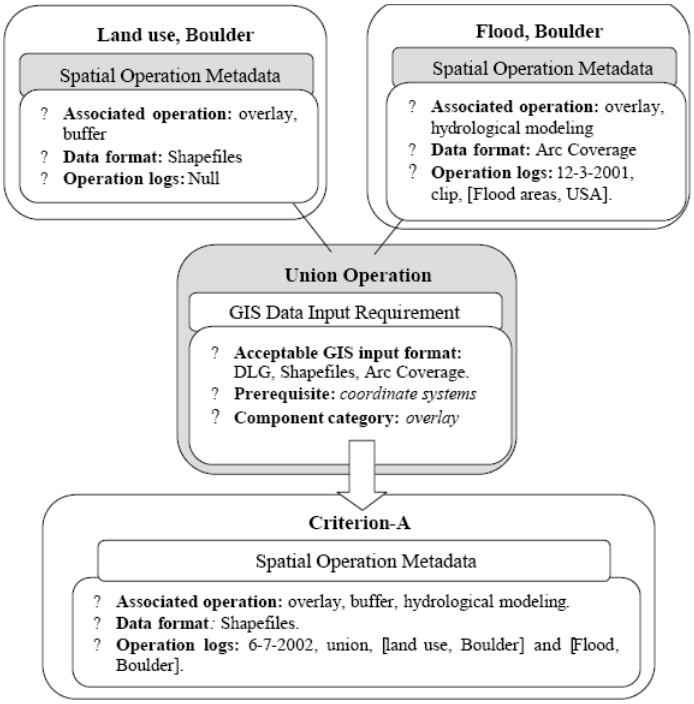


FIG. 2.11 – Exemple de besoin impliquant la coordination de deux traitements géographiques (extrait de [Tso02])

Une des principales propositions de M-H. Tsou pour améliorer non seulement les standards de métadonnées géographiques mais aussi les plateformes (*framework*) qui les exploitent est d'y intégrer ce qu'il appelle la notion de hiérarchie et qui en fait correspond aux problèmes de recherche de ressources dont s'occupe le Web sémantique. M-H. Tsou souligne ainsi la difficulté avec les métadonnées actuelles de trouver des informations routières sur San Diego lorsque l'indexation de celles-ci mentionne l'état de Californie.

Parallèlement à ce problème de fond, M-H. Tsou relève un certain nombre de caractéristiques des services Web qui montrent clairement les besoins d'*adaptation* au contexte de l'utilisateur. L'utilisation des services Web est ainsi soumise à des contraintes sur le débit de la connexion Internet ou sur la résolution des écrans affichant les données (notamment pour les applications sur client mobile).

[47] Java Unified Mapping Platform, permet de visualiser des données géographiques et de les manipuler via des objets Java `http://jump-project.org/`.

[48] *"An Operational Metadata Framework for Searching, Indexing, and Retrieving Distributed Geographic Information Services on the Internet" [Tso02].*

MMM – Un modèle pour les SIG "services Web"

Les auteurs de l'article "From GISystems to GIServices (...)" [GM97] partent du constat que les SIG monolithiques (*i.e.* non modulaires) sont coûteux et sous-utilisés. L'avenir appartiendrait donc aux services Web "à la carte" (*customisable*)[49].

Dans la même veine qu'UDDI ou que le catalogue de l'OGC (qui n'existaient pas à l'époque), une méthode pour créer un middleware (*i.e.* un serveur) permettant l'accès aux services géographiques est proposée. Il s'agit de MMM, acronyme de Middleware for Method Management.

Les premières des fonctionnalités supportées par MMM sont l'affichage, la recherche et le requêtage pour permettre l'accès aux services pertinents. Les buts de MMM rejoignent donc en partie les nôtres. Notons que MMM date de 1999 et est donc antérieur à WSDL et UUDI.

L'objet de l'article [GM97] est principalement de présenter les aspects techniques nécessaires à la mise en œuvre de MMM. Le modèle conceptuel de MMM nous intéresse davantage. L'article de Jacobsen *et al.* détaille l'implémentation de l'architecture MMM qui repose seulement sur trois classes représentant [JGR00] :

- les données : spécification des types, lieu de stockage, taille, fournisseur, conditions d'utilisation
- les services : spécification des fournisseurs, auteur, E/S, code source, fonctionnalité, etc.
- l'exécution des traitements : spécification des identifiants d'un service et des données correspondant à ses E/S pour une exécution donnée.

Ces trois classes héritent d'une classe mère abstraite contenant des méthodes pour la génération de documentation HTML, pour la recherche, l'accès et le stockage des instances des classes.

Le contenu des descriptions MMM les destine à être exploitées par des programmes et non à être consultées par des humains (cf. tab. 2.10). La langue naturelle est de fait quasiment absente. Le contenu est clairement plus procédural que déclaratif. Les utilisations nombreuses de références (par URL) déchargent MMM de certains aspects des descriptions, comme apparemment les types complexes de données.

Classe représentant les services	**Attributs**	**Détail des attributs**
Method Service Object	general	person, creation, right
	source	reference (url) ou code
	library	reference (url) ou code
	interface	language, input, output, precond, postcond
	domain	description, keyword, abstract

TAB. 2.10 – Représentation MMM des services (d'après la DTD de [JGR00])

Travaux relatifs à la description des SIG et programmes géographiques

L'article *"User interface for the integration of GIS components"* présente une méthode pour décrire en XML & RDF les composants SIG et leurs relations afin de permettre leur interopérabilité [LdB02]. Celle-ci dépend de la capacité à établir la correspondance entre le format des données disponibles et celui des données attendues par les traitements. Il existe

[49] La modularité des systèmes ne passe pas forcément par les architectures Web, mais c'est de ce contexte dont il est question ici.

donc un lien fort entre description des données et description des traitements.

Lemmens et de By présentent donc une méthode pour modéliser les relations entre données et traitements. Trois niveaux de connectivité entre données et traitements sont définis :

- niveau sémantique
- niveau structure de données
- niveau format de données

Pour chacun de ces niveaux, données et traitements doivent "parler" le même langage, c'est-à-dire utiliser les termes d'une même ontologie. C'est là l'idée forte de l'article. L'originalité du schéma de description des composants proposé tient à l'élément data_conditions qui exprime concrètement les conditions d'interopérabilité.

Elément	**Attributs**		
general	operation name, version, creation date, etc.		
detailled	data_format		
	parameters	name	
		type	
		order	
		description	
		direction	
		data_conditions	attribute
			comp (comparateur)
			condition ou parametre attribut
	prerequisites		
access	operation provider information		

TAB. 2.11 – Modèle de métadonnées de traitements géographiques proposé par Lemmens et de By [LdB02]

Par exemple, dans le cas d'une opération de projection, le système de coordonnées du fichier d'entrée doit être Geographic. Cela s'exprime ainsi :

```
<Data_Conditions>
   <Attribute>CoordSys</Attribute>
   <Comp>=</Comp>
   <Condition>Geographic</Condition>
</Data_Conditions>
```

Extrait de code 2.4: XML – Expression d'une condition dans le modèle de Lemmens et de By [LdB02]

Dans cet exemple, Geographic est un terme dont le sens n'est spécifié nulle part. Il faudrait qu'il le soit dans une ontologie afin de pouvoir être partagé, donc de servir de support à l'établissement de la connectivité. En pratique, les conditions ne peuvent donc être exprimées que si elles portent sur des formats de données connus et référencés. En l'occurrence, [LdB02] ne proposent malheureusement aucune ontologie des données géographiques (mais il suggère le format GML – Geographic Markup Language).

Le schéma de description proposé par Lemmens et de By a pour but l'interopérabilité des composants SIG. L'idée principale est de mettre en correspondance les données géographiques et les entrées/sorties attendues par les traitements. Les moyens employés – partage de termes et définition de niveau de connectivité définis dans des ontologies formalisées en RDF – nous serviront pour faire le lien non pas entre composants, mais entre besoins ou données d'un utilisateur et traitements recherchés.

Modèle de description des traitements informatiques géographiques proposé par l'ICA

Nous pouvons noter l'existence d'un autre modèle de description de traitements géographiques intéressant pour nos besoins. C'est celui que propose le groupe de travail de l'ICA (International Cartographic Association) pour la généralisation de cartes géographiques. Ce groupe a construit une plateforme pour le partage du code des outils géographiques et un schéma de description[50]. Les éléments de description associés sont listés tableau 2.12.

Author	Software Requirements	Comments by the author *(author's experiences, weakness and strength of the tool)*
Contact	Pseudo Code	
Status	Parameters	History
Language	Extended Description	References
Short Description	Samples (pictures)	Comments by other users

TAB. 2.12 – Éléments de description de code proposés par le groupe de travail de l'ICA de Zurich

Le schéma de description proposé couvre les besoins courants de consultation. De notre point de vue particulier, la faiblesse vient de la liberté de format laissée aux valeurs des rubriques. Seule celle des commentaires des auteurs possède l'ébauche d'une structuration. Les descriptions courtes et détaillées (short & extended) sont en langue naturelle.

3) Pages d'aide et interfaces des SIG

Les fonctions des SIG sont accessibles via des menus. Leurs descriptions sont contenues dans des pages d'aide. L'organisation des menus et des pages d'aide est le fruit d'une réflexion. Elle respecte une logique. Le critère essentiel est le regroupement thématique des fonctions. Examinons donc l'organisation choisie pour quelques uns des principaux SIG utilisés à l'IGN (tableau 2.13).

Nous tirons deux enseignements de l'examen du tableau 2.13. Premièrement, les pages d'aide et les IHM des SIG fournissent des classifications des fonctionnalités géographiques qui peuvent servir à décrire nos traitements. Deuxièmement, il n'y a pas de réelle unité entre les différents SIG et les pages d'aide sont faiblement et inégalement structurées. Ce constat nous conforte dans l'analyse des besoins à l'origine de notre travail ; la création de métadonnées structurées selon un modèle unifié comblera bien un manque.

2.1.4 Métadonnées des traitements informatiques géographiques à l'IGN

Sommaire

Il existe plusieurs types de description et modèles de description de traitements utilisés au sein de l'IGN. Dans beaucoup de cas, la spécificité géographique n'apparaît pas ou peu. Nous

[50] `http://www.geo.unizh.ch/ICA/docs/tools/code_template.html`

	ArcExplorer	**Arcview**	**Geoconcept**	**MapInfo**
Structure d'une page d'aide décrivant une fonctionnalité	Pas de structure récurrente	Pas de structure récurente	Rôle Disponibilité Principes Mise en œuvre des principes Intérêt Conseil de manipulation	Pas de structure réccurente, mais orienté instructions utilisateurs
Classification induite par les menus des IHM et par le sommaire de l'aide	Exploitation Outils d'intégration Web Obtention et utilisation des attributs des entités Echelle et représentation cartographique Symbolisation des données Appariement d'adresses	Types de données Créer carte Afficher carte Requêter carte Travailler avec les données tabulaires	Apparence Objets Données Topologie Géocodage GPS Itinéraire	Outils Objets Sélection Affichage Carte

TAB. 2.13 – Pages d'aide et interfaces des SIG

avons ainsi rencontré diverses documentations API générées par des outils comme Javadoc (pour les codes Java du COGIT) ou DOxygene (pour les codes C++ du MATIS), des informations relatives aux suivis de versions fournies par le logiciel CVS, des spécifications formelles utilisant des diagrammes UML et SADT (Structured Analysis and Design Technics). Toutes ces documentations sont informatiques avant d'être géographiques.

Nous avons néanmoins pu trouver trois modèles de métadonnées des traitements spécifiquement dédiés au domaine géographique (même si le troisième peut être adapté à un contexte plus générique) : le modèle OEEPE (Organisation Européenne d'Etudes Photogrammétriques Expérimentales), le modèle défini par P. Michaux, et le modèle défini par B. Bucher. Tous trois sont utilisés ou ont été conçus au laboratoire COGIT.

1) Grilles OEEPE

Les descriptions créées à partir de ce schéma ne sont destinées qu'à la lecture : elles sont contenues dans de simples documents Word (cf. tab. 2.14). Notons que les éléments *schema* et *example* accueillent des dessins très utiles à la compréhension.
Le schéma de description est très complet. Beaucoup des sous-catégories sont clairement dédiées aux algorithmes de généralisation. La plupart des catégories principales (colonne de gauche) sont, elles, génériques. Pour le besoin de consultation qui consiste à découvrir un algorithme, le schéma semble tout à fait adapté. Les descriptions déjà existantes pourront donc être récupérées. Par contre, pour les besoins de consultation qui nécessiteront de disposer de descriptions "opérationnalisables", il faudra contraindre les valeurs des éléments du schéma. Les éléments dont la valeur est en langue naturelle devront être décomposés ou complétés par une liste de mots-clés. Les autres éléments devront être typés : booléen, entier, réel, chaînes de caractères prédéfinies, et surtout types de données géographiques.

2) Modèle de P. Michaux

ALGO NAME

TEMPLATE FILLING

Author	
Date	

REFERENCES

Main reference	
Other references	

CLASSIFICATION **Main* ** Second*

Selection	Classification	Filtering	Smoothing	Caricature
Aggregation	**Structuration**	**CUSP**	**Enhancement**	**Others**

PRINCIPLE DESCRIPTION

Principle	Schema	Example	Foreseen Evolutions

MATHEMATICAL CONCEPT

DATA MODELING

Geometric Modelling	Type	Notes
Mode		
Dimension		
Modelling of input data		
Modelling of data in the algorithm		
Data enrichment		

Semantic Modelling	Y / N ?	Notes
Object by object?		
Use of attributes?		

Spatial Relations Modelling	Type	Notes
Topology		
Proximity		

Sample of the tested objects

GLOBAL EVALUATION

Advantages / Drawbacks / Comparison with other algorithms

APPLICATION AREA

On which semantic type of objects ?	
On how many objects ?	
For solving which conflicts ?	
Can be used on which kind of geometry (homogeneous, smooth...)?	
Can be used with which kind of environment ?	
Is there a necessary or usual pre and post treatment ?	
Kind of use (main, pre, post...) ?	
Used for which scales ?	

STABILITY OF COMPORTMENT

Continuous regarding...	Y / N	Notes
Geometrical deformation		
Parameters changes		
Environment changes		

Stable regarding...	Y / N	Notes
Segmentation, i.e. adding vertex without changing the geometry		
Splitting, i.e. splitting the object and applying the algo on both of them		
Orientation of the arcs. i.e. if the order of points is reversed		

PARAMETERS EVALUATION

	Y / N	Notes
Are the values related to known values?		
Do they have clearly distinct effects?		

Density		

CHANGES DUE TO THE USE OF THE ALGORITHM

Type of modifications	Y / N ?	Notes
Semantic		
Geometry		
Topology		

PARAMETERS

Name	Unit	Range of value	Meaning

ALGORITHMIC DESCRIPTION

Complexity	
Links with other algorithms	

DEVELOPMENT ENVIRONMENT

Institution	
Developers	
Manager	
Period of Development	
Language	
Use of rules or others development tools	
Operating System	
Graphic visualisation tool	
GIS	

EVALUATION

TESTS ON OBJECTS SET

Description of the objects set tested	
Description of the test	
Mean of evaluation of the test (use of measures?)	

In close conditions (ex. same type of geometry) are they used with close values		
Is it easy to choose them, interactively ...		
... and automatically		

FOR LINES ONLY, MORE ACCURATE QUALIFICATION OF ALGORITHM'S EFFECTS.

EFFECTS ON LINES CONFLICTS:

This part shows the systematic effects on conflicts (in the final line) if the algorithm as any.

Conflict	Effects
Presence of topologic conflicts in one line	
Presence of conflicts with environment (topologic or graphic)	
Presence of graphic conflicts in one line (ex. bend coalescence)	
Too angular lines	
Too detailed lines	
Too many points on lines	

EFFECTS ON LINES CONSTRAINTS

Comparison between original and final lines	Effects
Absolute planimetric precision	
Shape modifications	
Line length	

TAB. 2.14 – Modèle OEEPE de description d'algorithmes de généralisation de lignes utilisé au COGIT

P. Michaux a également développé une interface de consultation Topic Maps pour les traitements de généralisation au COGIT (cf. fig. 2.12). Etudiant en DESS Imagerie Electronique, il a réalisé un stage sur les métadonnées des traitements. Il s'agissait de réaliser une interface de consultation pour la soumettre à des utilisateurs pour faire ressortir leurs besoins [Mic03]. La fonctionnalité de base attendue dans le cadre du stage était de permettre à l'utilisateur de naviguer dans les index des descriptions. Dans ce but, P. Michaux a adopté le standard Topic Map, langage de représentation que nous étudierons chapitre 4. Deux modèles conceptuels de descriptions de traitements ont été créés : l'un générique et l'autre dédié à la généralisation.

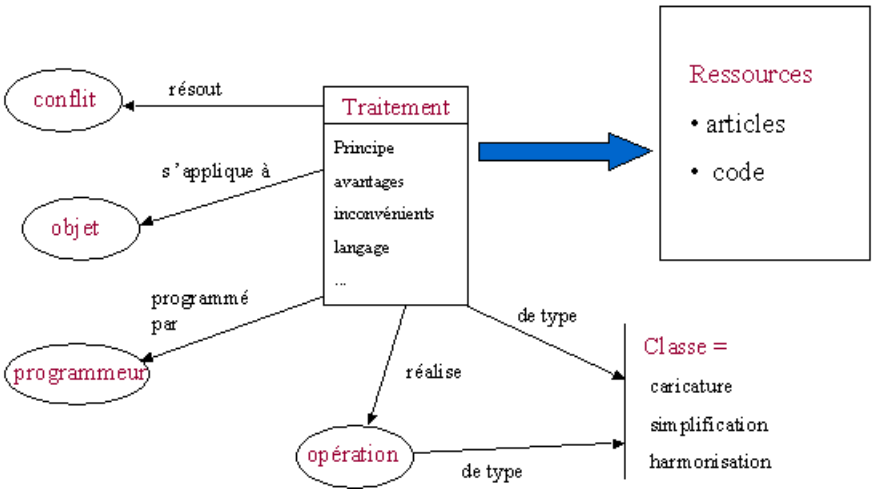


FIG. 2.12 – Modèle de métadonnées des traitements de généralisation proposé par P. Michaux [Mic03]

Notons la présence du topic "conflit", spécificité des traitements de généralisation. Les conflits surviennent lors des déplacements ou changements de forme d'objets géographiques imposés par la généralisation, par exemple quand un bâtiment se retrouve sur une route.

La figure 2.12 présente l'écran de l'interface utilisateur. Elle a été introduite suite au souhait exprimé par les utilisateurs de se situer visuellement au cours de leur navigation dans le modèle de descriptions.

Au niveau du modèle conceptuel de descriptions de traitements nous retenons notamment la propriété conflit spécifique à la généralisation ; et au niveau interface IHM la possibilité offerte à l'utilisateur de se situer dans le modèle de descriptions.

3) Modèle de tâches. L'aide à l'accès à l'information géographique : un environnement de conception coopérative d'utilisations de données géographiques – B. Bucher

B. Bucher a réalisé une thèse sur l'aide à l'accès à l'information géographique [Buc02]. Son but était de fournir des descriptions de données géographiques associées à leur mode d'emploi, relativement à un besoin exprimé par un utilisateur. Ce travail a donc ceci de commun avec le nôtre qu'il tente de relier l'expression d'un besoin à des métadonnées. Les deux travaux sont en fait complémentaires : il s'agit d'abord d'identifier les données nécessaires à un besoin, et de déterminer l'agencement de traitements qui le satisfasse (le travail de Bucher), puis de rechercher lesdits traitements (notre travail). Pour atteindre son but, Bucher distingue deux catégories de connaissances :

– les connaissances de description d'un domaine : le QUOI,
– les connaissances de manipulation des objets de ce domaine, le COMMENT.

Selon Bucher, le COMMENT doit être modélisé de telle sorte que puissent être apportés des réponses aux questions :

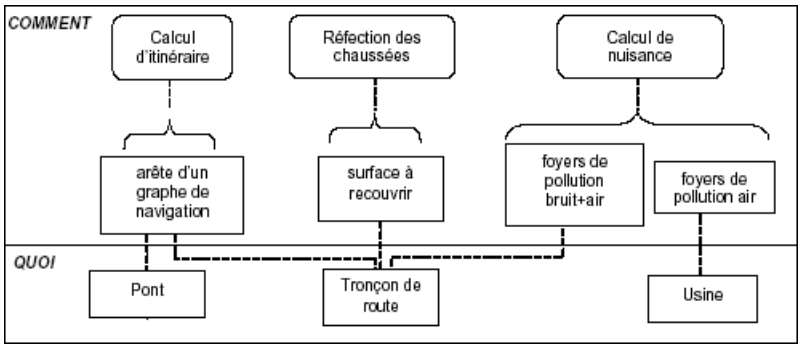


FIG. 2.13 – Le "quoi" et le "comment" de l'information géographique selon [Buc02]

– Pourquoi ? : quel est l'objectif poursuivi dans la manipulation ?
– Comment ? : quelles sont les stratégies et les méthodes pour manipuler les objets ?
– Avec quoi ? : quels sont les outils (*i.e.* les traitements) utilisés ?

On voit que la description du COMMENT comporte une partie touchant aux fonctionnalités des traitements, mais aussi les méta-connaissances nécessaires à l'élaboration de plans de tâches. Trois notions sont justement introduites pour modéliser le QUOI, le COMMENT, et les liens qui les unissent : les tâches, les méthodes et les rôles. Une tâche est un type de problème que l'on sait résoudre (l'exemple donné est celui d'aller d'un endroit de Paris à un autre en métro). La description d'une tâche se décompose ainsi :

– la description du problème posé (facette déclarative de la tâche),
– la description de la résolution du problème (facette opérationnelle de la tâche).

On distingue les tâches complexes, elles-mêmes constituées de tâches ; et les tâches primitives, qui ne sont pas décomposables. Au final, la solution d'un problème consiste donc en un plan de tâches primitives. Les connaissances heuristiques sont contenues dans les méthodes. Les rôles sont les termes servant à l'utilisateur pour décrire ses besoins et au système pour décrire les tâches. Les ressources décrites comme des tâches ou des traitements sont assez proches -manipulation d'une entrée pour produire une sortie-. Au COGIT, nous choisissons de faire les distinctions suivantes qui se situent au niveau des ressources décrites ou alors uniquement au niveau du mode de description.

> "Une tâche correspond à l'atteinte d'un objectif d'utilisateur, comme calculer un itinéraire ou apparier des jeux de données. La description de tâches se concentre sur le mécanisme de spécification d'une tâche générique dont la résolution peut comporter des alternatives pour décrire une tâche totalement spécifiée dont la résolution est figée. La description d'une tâche doit comporter des termes proches du niveau de compréhension de l'utilisateur, comme le concept de localisation.
>
> La description d'un traitement se concentre elle sur le niveau données et logiciel. Dans une telle description, la résolution est plus ou moins figée, hormis les fenêtres de dialogue et paramètres éventuels.
>
> Ainsi, une tâche totalement spécifiée peut être associée à une description de traitement."

2.1.5 Modèles et langages de description de connaissances générales

Nous serons amenés à parler en détail des langages de représentations des connaissances au moment du choix de l'implémentation (chapitre 4). Ici, nous nous intéressons à quelques modèles et langages de description de connaissances générales.

Sommaire

1 Métadonnées pour l'identification

Dublin Core

Dublin Core est un schéma de métadonnées générique qui permet de décrire une ressource numérique ou physique et d'établir des relations avec d'autres ressources. Il comprend 15 éléments de description listés tableau 2.15.

La signification de certains éléments peut être précisée à l'aide de raffinements. Un raffinement restreint la signification d'un élément, mais sans la changer fondamentalement. L'utilisation des raffinements est facultative.

Title	Language	Date
Creator	Type	Relation
Subject	Format	Coverage
Description	Identifier	Rights
Publisher	Source	Audience

TAB. 2.15 – Les quinze éléments du Dublin Core [Ini04]

FOAF

Les traitements sont décrits, développés et utilisés par des personnes. Par conséquent, les informations sur ces personnes sont, de façon annexe, des métadonnées pertinentes dans le cadre de nos besoins.

Dans l'esprit de ce qui allait devenir le Web sémantique[51], un projet a vu le jour en 1998 afin de décrire les personnes, leurs relations et leur activités. Ce projet, nommé FOAF (Friend Of A Friend[52]), vise à fournir un langage de métadonnées destiné aux machines, c'est-à-dire non seulement destiné à l'affichage par les navigateurs Web, mais aussi conçu pour permettre la mise en relation des personnes et les inférences basées sur la sémantique des termes FOAF. Pour cette

[51] Vision du Web de demain que nous présentons p. 151.
[52] `http://www.foaf-project.org/`

raison, le langage FOAF repose sur XML et RDF, standards du W3C que nous présenterons au chapitre 4.

Le langage FOAF est en passe de devenir un standard pour la description des personnes sur le Web. Dès lors qu'il s'avère adapté à une partie de nos besoins, il paraît opportun de chercher à être compatible avec lui. Effectivement, les éléments de description listés tableau 2.16 présentent bien un intérêt pour nous.

Classes	**Propriétés**	
Agent		
Person	knows	phone
	geekCode	mbox
	currentProject	publications
	homepage	
Organization	based_near	logo
Group	member	
Project	theme	made
Document	topic	

TAB. 2.16 – Quelques classes et propriétés FOAF [Pdc05]

2) Langage de notation mathématique

MathML

MathML (*Mathematical Markup Language*) est un langage dédié aux notations mathématiques. Apparu en 1998, c'est une recommandation W3C depuis 2001 [W3C03b].

Nous nous y intéressons car dans le domaine géographique, il est fréquent que les descriptions de traitements comportent des formules mathématiques. Or lorsque l'on édite du texte, la présentation des formules mathématiques est souvent un problème. Dans le monde du Web, HTML offre peu de possibilités ; dans le monde des éditeurs de texte généralistes, il n'y a que TEX et LATEX qui soient vraiment adaptés. Face à cette situation, MathML propose une façon d'encoder les formules mathématiques. Mais au contraire de LATEX, MathML n'a pas seulement pour but la représentation graphique des formules, mais également celle de leur sémantique, de leur sens mathématique (*"Encode both mathematical notation and mathematical meaning"* [W3C03b] – Chap.1, §*Design Goals of MathML*).

Nous avons indiqué section 1.4.2 n'avoir pas pour objectif de construire des métadonnées capables de simuler tout ou partie du comportement des traitements. Néanmoins, s'il se trouve que le formalisme choisi pour représenter graphiquement les formules mathématiques en préserve également le sens, ce peut être une bonne chose.

On peut ainsi imaginer que les descriptions formelles de la complexité des algorithmes pourraient être exploitées pour effectuer des comparaisons. Actuellement, seul l'expert est capable de prédire le temps d'exécution acceptable d'un programme en fonction du nombre d'objets d'entrée ou des paramètres de qualité attendue du résultat.

La complexité concerne l'évaluation des traitements. On a également besoin des formules mathématiques pour décrire leurs fonctions et leur fonctionnement. L'exemple qui suit est la traduction en MathML d'une description d'un programme ADA calculant le nombre de points d'inflexion d'une ligne. La description en question est issue de la documentation interne du COGIT. La formule y est notée en simple texte. Elle apparaît donc avec une lisibilité moindre que sur la figure 2.14 et, bien sûr, ne peut servir que dans le cadre d'une lecture humaine.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
      "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mn>1</mn>
  <mo>-</mo>
  <mrow>
    <munderover>
      <mo>&Sum;</mo>
      <mrow>
        <mi>i</mi>
        <mo>&le;</mo>
        <mi>taille</mi>
        <mo>(</mo>
        <mi>cptbis</mi>
        <mo>)</mo>
        <mo>&minus;</mo>
        <mn>1</mn>
      </mrow>
      <mrow>
        <mi>k</mi>
        <mo>&le;</mo>
        <mi>i</mi>
      </mrow>
    </munderover>
    <mi>cptbis[k]</mi>
  </mrow>
</math>
```

Extrait de code 2.5: MML – Formule pour le calcul du nombre de points d'inflexion d'une ligne

Le code MathML 2.5 est affiché de la façon souhaitée par le logiciel Amaya[53]. (figure 2.14). Nous ne connaissons pas suffisamment le langage pour être sûr que notre code est exempt d'erreur du point de vue sémantique. Il est du moins syntaxiquement correct puisqu'il est valide vis-à-vis de la DTD officielle MathML 2.0[54].

Si nous limitons notre ambition à l'affichage, il n'est pas indispensable de connaître le rôle de tous les éléments. Voici sommairement, à titre d'indication, quelques explications sur ceux utilisés dans notre code :

- l'élément mrow définit une ligne ;
- les éléments mi, mn et mo représentent respectivement les identifiants, les nombres et les opérateurs ;
- l'élément munderover permet de répartir ses trois éléments fils au dessus, sur et en-dessous de la ligne courante.

On voit à ces descriptions que le langage MathML est orienté présentation. Cela peut sembler gênant. La tendance générale du Web est justement de séparer contenu et présentation, comme le montre l'apparition du langage CSS et l'usage de plus en plus fréquent du couple XML/XSL[55] pour générer le HTML. Précisons d'ailleurs que MathML peut être vu comme une extension de

[53] `http://mozinet.free.fr/aut/amaya.html`).

[54] `http://www.w3.org/TR/MathML2/dtd/mathml2.dtd`. Dans cet exemple nous utilisons la DTD MathMl2 (et XML Spy 2004 pour la validation), mais au final nous verrons par la suite que c'est la version XML Schema que nous utilisons.

[55] XSL (eXtensible Stylesheet Language) est le langage de description de feuilles de style du W3C associé à XML [W3C99c]. XSL est composé de XSL-FO (Formatting Objects), un vocabulaire qui permet d'appliquer un style à un document XML, et de XSLT, un langage de transformation de document XML. Dans la suite de ce mémoire, conformément à un usage répandu, en l'absence de précision, XSL sera employé au sens XSLT.

HTML : il est possible d'intégrer du code MathML dans des documents HTML. Les navigateurs Web peuvent les interpréter à l'aide des plugins appropriés[56].

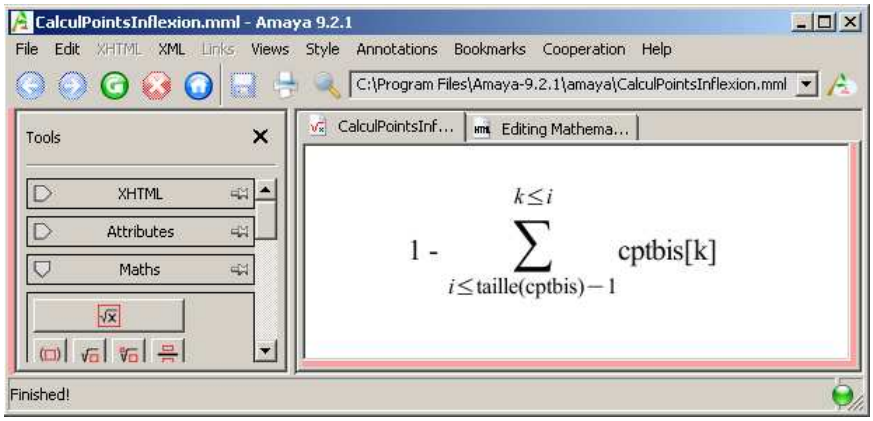


FIG. 2.14 – Visualisation avec Amaya 9.2.1 du code 2.2

3) Capitalisation des connaissances

MASK

Les principes généraux évoqués se traduisent dans des modèles particuliers de représentation de connaissances. Certains de ces modèles sont nécessairement très formels puisqu'ils servent de support direct à l'exploitation informatique (langages tels que par exemple UML pour la programmation objet). Le domaine est alors celui du génie logiciel ; or notre but en est plutôt éloigné. Nous sommes davantage intéressés par les modèles de gestion de connaissances (*knowledge management*) dont C.Bandza a fait une synthèse [Ban00] et parmi lesquels figure MASK.

La méthode MASK (*Method for Analysing and Structuring Knowledge*) est une méthode de gestion des connaissances qui permet de recueillir et capitaliser des savoirs tacites d'experts. L'ancêtre de MASK est MKSM (*Methodology for Knowledge System Management*), conçue par J.-L. Ermine afin de représenter les connaissances au sein du CEA (Commissariat à l'Energie Atomique). Depuis MKSM a été utilisée par des sociétés comme EDF, VIVENDI, PSA, Renault [Erm03][BA03].

MKSM a également été utilisée dans un but extrêmement proche du nôtre à l'INRIA Sophia Antipolis. Il s'agissait en effet de mettre en place une "gestion opérationnelle des connaissances sur les codes" [ME00]. Les auteurs de l'article présentant le projet, S. Moisan et J-L. Ermine, partent pour beaucoup du même constat que celui que nous avons effectué lors de notre analyse des besoins, notamment au sujet des connaissances tacites. "Dans la chaîne de moyens mis en place pour la gestion des codes, *[S. Moisan et J-L. Ermine ont]* identifié deux points faibles : le passage de la documentation scientifique et technique aux documents de développement (informatiques) et le passage de la documentation utilisateur à l'utilisation opérationnelle" *(ibid)*. Les auteurs proposent deux solutions : l'utilisation de *livres de connaissances MKSM* et des outils de pilotage de codes.

L'élaboration de la méthode MKSM a pour origine un constat, celui de la difficulté d'acquérir les connaissances tacites des experts parfois difficilement exprimables. D'où la

[56]Par exemple, MathPlayer pour Internet Explorer 6 et supérieur (`http://www.dessci.com/en/products/mathplayer/`).

nécessité de fournir un cadre afin de faciliter ce qui est avant tout un problème d'acquisition des connaissances.

Les objectifs de MASK sont multiples [EBS00] :

- Capitalisation de savoir d'experts quittant leur poste (mutation, départ à la retraite, etc.),
- Structuration de corpus d'informations et/ou de documents,
- Intégration de savoir-faire dans des procédés industriels ou des processus d'entreprise pour améliorer leur productivité et leur compétitivité,
- Diffusion des connaissances des experts à travers des *Livres de connaissances*.

Le Livre de connaissances, produit final de la méthode MASK, est une sorte d'encyclopédie "métier" qui peut se présenter sous un format papier ou sous un format électronique. Il est constitué par un ensemble de modèles formalisant la connaissance recueillie lors des interviews. Les modèles, d'apparence graphique à la façon des diagrammes UML, sont complétés par des fiches descriptives du domaine, des conseils, des retours d'expérience et des références bibliographiques. Les descriptions des six modèles MASK traduisant les différents points de vue sont données tableau 2.17.

Connaissances fondamentales	Les connaissances fondamentales, scientifiques, techniques ou autres, qui sont la base même du domaine, sont représentées par des **modèles de phénomène** ou **modèles du domaine**, qu'on cherche à maîtriser, soit pour les favoriser, soit pour les inhiber, dans une activité experte.
Activités	Les connaissances sur le déroulement de l'activité experte sont représentées par des **modèles d'activité**. C'est une décomposition en grandes phases (sous-activités) du métier considéré, ces grandes phases étant articulées entre elles par des échanges de données, de flux de matière, etc. Il s'agit d'une analyse de type "fonctionnelle" descendante, où chaque activité est décomposée hiérarchiquement en sous-activités de plus bas niveau.
Contexte historique	L'évolution du domaine de connaissance, replacé dans son contexte scientifique, technique, social, etc. est modélisé par des **modèles d'historique**. Il s'agit donc d'intégrer l'évolution d'une connaissance, d'un concept, d'un objet dans un système contextuel qui est explicatif de cette évolution, et permet d'appréhender globalement les lignes directrices qui ont amené la connaissance à l'état actuel.
Savoir-faire	Les savoir-faire particuliers et remarquables, rattachés aux activités sont représentés par des **modèles de tâches**. C'est une représentation de la stratégie mise en œuvre pour résoudre le ou les problèmes qui sont posés dans des cas précis concernant le système de connaissances considéré.
Concepts	Les concepts manipulés dans l'activité experte considérée sont modélisés par des **modèles de concepts**.
Historique des solutions et justifications	Un autre point de vue plus détaillé que l'historique pour appréhender l'évolution du domaine de connaissances à travers le temps est de reconstruire "l'arbre généalogique" des solutions qui ont été conçues dans le domaine, en indiquant les justifications qui ont amené à passer d'une génération à une autre. C'est le **modèle des lignées**.

Tab. 2.17 – Les modèles MASK (d'après [BA03] et [Erm03])

LOM

Enseigner des connaissances demande en général de les représenter. Ce constat pourrait nous inciter à nous intéresser au domaine de l'Enseignement Assisté par Ordinateur (EAO) ou, mieux,

à celui de l'Enseignement Intelligemment Assisté par Ordinateur (EIAO). Mais les systèmes existants sont souvent dédiés à des disciplines particulières, par exemple la géométrie en 4ème, et incluent la gestion de l'activité d'enseignement reliant la représentation de l'enseignant et celle de l'apprenant. Cela constitue un prolongement extrêmement intéressant de nos objectifs, mais qui, pour l'heure, les dépasse un peu.

Plus directement exploitables sont les travaux dédiés à l'enseignement à distance, aussi appelé *e-learning*. Ils ont été rendus possibles avec l'apparition d'Internet. Certes, les systèmes mettent parfois en place, comme pour les EIAO, des situations d'apprentissage comprenant des interactions avec l'apprenant. Mais il semble que l'apprentissage y est surtout étudié du point de vue de la mise en place des formations – *e.g.* conception, médiatisation, diffusion des supports [Tch02]. Or, comme ces systèmes sont par nature plus ouverts, des normes de métadonnées généralistes deviennent nécessaires pour la description, le partage, la réutilisation et l'évaluation des ressources pédagogiques.

C'est dans ce but qu'en 2002 l'IEEE a défini la recommandation LOM (Learning Object Metadata) [Com02].

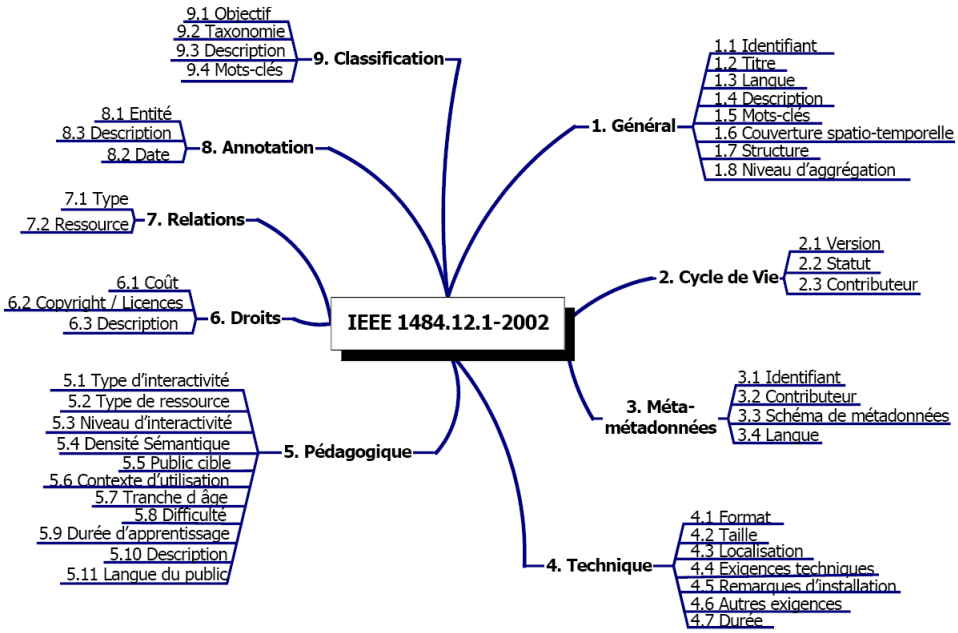


FIG. 2.15 – Aperçu général de LOM (diagramme extrait de [Gom04], d'après [Com02])

Que note-t-on dans le diagramme de la figure 2.15 ? D'abord que LOM reprend les descripteurs Dublin Core pour l'identification des ressources. On en retrouve notamment six sur huit dans la catégorie 1 *Général*. Ensuite, on note que beaucoup de descripteurs LOM se prêtent tout à fait à la description des connaissances d'utilisation des traitements. Par exemple, selon la complexité de son besoin, l'utilisateur d'un SIG devra préalablement savoir ce qu'est un thème, une table de base de données relationnelle, une représentation vecteur, etc. Les descripteurs LOM des catégories 4 et 5 (*Technique* et *Pédagogique*) seront utiles à la description des prérequis d'utilisation des traitements.

Les ressources pédagogiques comme les traitements géographiques sont protégés par des licences et peuvent être payants. La catégorie 6 *Droits* pourra donc également être intégrée à notre modèle.

Le diagramme de la figure 2.15 ne fournit qu'un aperçu de LOM. Certains descripteurs se décomposent plus finement. Il sera justement particulièrement intéressant pour nous de décrire

de façon fine les préconditions d'utilisation des traitements. On verra par exemple plus avant un exemple où doit être exprimée une contrainte sur la version d'un logiciel[57]. LOM a prévu ce type de cas : les descripteurs 4.4.1.3 et 4.4.1.4 de la catégorie 4.1 *Exigences techniques* portent sur les versions minimum et maximum des logiciels requis pour exploiter les ressources pédagogiques [Com02]. Cela rejoint exactement le besoin de l'exemple sus-cité.

2.1.6 Conclusion

En quête d'un modèle de métadonnées correspondant à nos besoins, nous nous sommes intéressés à ceux des domaines informatique, informatique géographique, et informatique géographique à l'IGN. De façon annexe, nous avons également étudié quelques modèles généralistes de description de ressources. Nous n'avons pas limité notre recherche aux stricts modèles de métadonnées. Nous avons aussi étudié les langages, méthodes, travaux et outils utiles à la description des traitements.

Une des principales raisons de l'inadéquation des modèles de métadonnées étudiés à nos besoins est la divergence dans la vocation desdits modèles. Alors que nous visons avant tout la création de métadonnées consultables par des lecteurs humains, les descriptions de services Web tels ISO 19119, SOAP, WDSL,UDDI et OWL-S sont plutôt dédiées aux machines. Parmi les buts des langages en question figurent certes la découverte de services Web, mais aussi surtout leur invocation automatique et leur interopérabilité. De même, les langages de description de composants, que nous avons peu évoqués, ne fournissent pas tous les éléments de description dont nous avons besoin. Il leur manque par exemple la description des effets des traitements sur les propriétés des données en termes pertinents pour l'utilisateur et non pas seulement du point de vue de l'implémentation. C'est justement une des spécificités des traitements géographiques que de nécessiter une description fine des entrées/sorties aux nombreuses propriétés.

Ce dernier reproche – relativement à nos besoins particuliers s'entend – peut être également adressé aux outils et langages de description du domaine du génie logiciel dont le but est de faciliter la phase conception de traitement et non l'utilisation *a posteriori*.

Dans le même ordre d'idée, parmi les documentations rencontrées certaines sont explicitement dédiées aux développeurs, d'autres aux utilisateurs.

Les inadéquations des modèles de métadonnées rencontrés lors de l'établissement de l'état de l'art ne sont pas toujours liées à des divergences de vocation. Les inadéquations sont parfois aussi simplement des niveaux de détail et de formalisation des modèles insuffisants. Certains modèles ne sont pas assez détaillés. ISO 19119, par exemple, propose relativement peu d'éléments de description. D'autres modèles de description en possèdent suffisamment, mais sont spécifiques à un domaine. Par exemple, le modèle défini dans le cadre du projet Agent est spécifique aux traitements de généralisation.

Par ailleurs, les éléments de descriptions des modèles ne sont pas toujours typés. Or notre objectif de contrôle des valeurs des métadonnées nous incline à limiter le recours à la langue naturelle. Nous avons vu en particulier que les documentations API classiques ne sont généralement indexées avec aucun vocabulaire contrôlé. La formalisation s'y limite à la signature des fonctions, ce qui n'est pour nous pas suffisant.

Néanmoins, les modèles étudiés comportent beaucoup d'éléments à réutiliser et d'idées à retenir :

Divers éléments de description utiles. Les fonctionnalités et la signature des entrées, sorties et paramètres se retrouvent dans quasiment tous les modèles. Nous retenons également

[57] Exemple donné p. 123.

les idées des facettes d'OWL-S, de la décomposition des traitements et l'utilisation de structures de contrôle (Moisan, Ficet, OWL-S), et de l'expression de préconditions (OWL-S, Lemmens et de By, Tsou). Concernant les modèles et langages de description de connaissances générales, les éléments des modèles Dublin Core, LOM, FOAF et MathML pourront être directement réutilisés. Par ailleurs, les travaux de capitalisation des connaissances comme MASK montrent comment décrire non seulement des ressources mais aussi les connaissances *sur* ces ressources. Il est ainsi intéressant de prévoir la description des concepts utilisés. Il est également utile de fournir un modèle capable de représenter les règles d'expert et de faciliter leur acquisition.

Représentation opérationnelle des connaissances. En tant que projet du Web sémantique, OWL-S met en œuvre des principes de représentation des connaissances adaptés à nos objectifs de simulation d'une partie du raisonnement de l'expert.

Traduction de la spécificité géographique des traitements. Signe des temps, les modèles de description des traitements proposés par la communauté du domaine géographique concernent essentiellement les services Web. Il semble en fait que la spécificité géographique ne se traduit pas dans les éléments de description mais dans ses valeurs. De nombreux travaux tentent ainsi de définir des classifications des fonctionnalités géographiques. On trouve aussi des classifications des types de données géographiques, des problèmes géographiques liés au processus d'acquisition des données géographiques, à la façon de les cartographier, etc. Ces classifications présentent clairement un intérêt pour la description des traitements géographiques. Mais elles ne transparaissent pas forcément dans le choix des éléments du modèle de métadonnées. C'est pourquoi il ne nous est pas apparu nécessaire de les évoquer dans ce chapitre.

Le tableau comparatif 2.18 liste les principaux critères définissant notre positionnement vis-à-vis de l'état de l'art établi. Il complète les paragraphes précédents et montre que les modèles de métadonnées et les documentations existants que nous avons recensé ne répondent pas à nos attentes. Nous allons donc définir un nouveau modèle. On peut se poser la question de l'apport qu'il constituera.

Dans la mesure où le modèle à construire réunira les caractéristiques de la colonne de droite du tableau 2.18, il constituera un apport par rapport aux modèles existants. Inversement, dans des contextes différents du nôtre, certains modèles existants pourront s'avérer plus adaptés. Cette évidence rappelle simplement qu'un modèle est conçu dans un but. Le nôtre est relativement générique par rapport à ceux dédiés, par exemple, à la planification de tâches, à l'invocation des services Web, ou à la description d'API pour des développeurs chevronnés.

En fait, les concurrents les plus directs des métadonnées que nous souhaitons sont les documentations pas ou peu formalisées, à savoir les commentaires des développeurs sur leurs propres traitements (commentaires de code ou fiches externes) et les aides classiques des logiciels commerciaux. Par rapports à ces concurrents, notre apport résidera dans une plus grande formalisation des métadonnées, et par les choix pertinents des éléments de descriptions que nous emprunterons pour partie aux divers modèles déjà existants.

L'intérêt d'une plus grande formalisation des informations manipulées dans le cadre de systèmes d'information est déjà bien connu. Dans le domaine du Web sémantique, c'est l'objectif d'une formalisation des connaissances qui est maintenant poursuivi. Nous accompagnons ce mouvement.

L'objectif de décrire l'agencement des étapes du fonctionnement et des modes d'emploi des traitements nous fait partager quelques points communs avec les modèles dédiés à la planification de tâches et à la résolution de problèmes en IA. Nous allons ainsi avoir besoin de recourir à des structures de contrôle (*si / alors, tant que, et, ou*, etc.).Pour autant, la planification n'est pas notre objectif. Notre modèle n'a pas pour vocation à concurrencer sur ce

terrain les modèles existants.

Comparativement aux modèles existants, l'apport attendu du modèle que nous allons définir est de permettre l'expression de connaissances actuellement peu formalisées. De façon sous-jacente, un apport de notre travail réside dans l'identification de ces connaissances et dans la démonstra-tion que leur formalisation permet une meilleure aide à l'utilisateur. Par exemple, comme nous l'avons vu au chapitre 1, décrire les actions nécessaires à l'utilisation des SIG nécessitent la description des IHM (menus contextuels, préconditions, problèmes réccurents). La compréhension des IHM des SIG et plus généralement des traitements informatiques nécessite également la description des caractéristiques communes de familles de traitements (les SIG, les logiciels Windows, etc.). Permettre de décrire des catégories de traitements et non seulement des traitements individuels est utile et nécessaire. Les modèles rencontrés n'offrent pas cette possibilité de façon formalisée, à l'exception, potentiellement, d'OWL-S mais uniquement pour le contexte des services.

	Manuels, fichiers d'aide, doc. API	**OWL-S & autres modèles pour services Web**	**ISO 19119, ISO 19115-2 & modèles géo.**	**Le modèle de métadonnées que nous attendons**
Degré de formalisation des métadonnées	Insuffisant car langue naturelle. Pas (manuels) ou peu (doc. API) de contrôle du vocabulaire.	Suffisant, car descriptions destinées aux machines (optique Web sémantique).	Suffisant pour les éléments de description, pas toujours pour leurs valeurs.	Contrôle des éléments de descriptions et de leurs valeurs. Uniformisation de la façon de décrire les traitements.
Niveau de détail, généricité, prise en compte spécificité géographique	Métadonnées inégalement détaillées, souvent faibles pour les effets des traitements. La plupart des éléments de description généralistes sont à réutiliser.	Pas de description fine des effets des traitements sur les propriétés des E/S Dédiés aux services Web. À spécialiser pour le domaine géographique.	Des éléments à réutiliser et à spécialiser ou compléter (grilles OEEPE en particulier).	Description des effets des traitements : évolution des propriétés des entrées et sorties, illustrations. Pour cela, description aussi des diverses ressources du domaine géographique (types de données, format, problèmes, concepts).
Planification Préconditions Support interopérabilité	Non.	OWL-S décrit les préconditions sur les E/S et la décomposition des traitements, permet les raisonnements de logiques de description, comporte les structures de contrôle classiques.	Oui en théorie (moins développé qu'OWL-S sur ce point).	Description des agencements des étapes de fonctionnement et des étapes de modes d'emploi : objectif poursuivi mais de façon plus simple que les travaux spécifiquement dédiés à la planification. Support aux raisonnements pour l'adaptation à prévoir (volet implémentation des connaissances d'expert).
Description du mode d'invocation des traitements	Bonnes descriptions des actions IHM des logiciels ou des instructions pour la programmation.	Oui, mais avant tout pour les machines,(WSDL-UDDI-SOAP) et seulement pour les services Web.	Oui (peuvent être liés aux standards pour services Web).	Invocation pour tous types de traitements attendue (donc, entre autres, description des IHM). Invocation automatique : objectif non poursuivi.
Recueil de connaissances tacites	Non (sauf MASK ou SBC spécifiques).	Non.	Non.	Plus spécifiquement à notre contexte, en proposant les concepts intervenant dans les règles d'expert, en réifiant des concepts abstraits (p.ex. familles de traitement), les connaissances tacites pourront être explicitées.

TAB. 2.18 – Tableau comparatif de quelques éléments de l'état de l'art établi et du modèle de métadonnées attendu

2.2 À propos des choix de modélisation

2.2.1 Comment notre modèle a-t-il été élaboré ?

Élaborer un modèle, c'est faire une succession de choix. Dans notre cas, la méthode de conception sciemment suivie a été principalement ascendante. L'état de l'art ayant abouti à la décision de créer notre propre modèle de métadonnées, nous sommes partis de l'examen des traitements à décrire. Cela a fourni le premier squelette du modèle (fig. 2.18 p. 96). Nous avons sélectionné des exemples de traitements – pour la majeure partie des traitements de généralisation développés au COGIT – et avons commencé à les décrire. Les éléments de description sont ainsi apparus progressivement. Des entretiens avec les utilisateurs ont été menés, des questionnaires diffusés (cf. p. 231). Chaque besoin d'information nouvellement identifié impliquait la création d'un élément de description supplémentaire du modèle.

Occasionnellement, nous avons procédé de façon descendante. Il n'avait ainsi pas été envisagé, initialement, de décrire l'environnement matériel des ordinateurs exécutant les traitements. C'est en songeant à ce thème général que nous avons introduit plusieurs éléments de description particuliers, éléments dont la présentation aux auteurs de traitements a rétrospectivement permis d'identifier de nouveaux exemples de besoins (cf. exemple p. 123).

S'il est difficile de dégager une référence particulière de l'état de l'art qui ait notablement influé sur le choix des éléments de description de base du modèle, on peut en revanche citer OWL-S comme source d'inspiration dans la façon de les organiser. Nous reprenons et étendons en effet l'idée des trois facettes de descriptions des traitements (ce que fait le traitement, comment il fonctionne, comment y accéder). Les classes qui réifient ces facettes sont d'un niveau d'abstraction supérieur à celui des classes correspondant aux ressources plus évidentes, "concrètes" pourrait-on dire, comme les algorithmes ou les formats de données.

On peut subodorer que, pour des besoins tels que les nôtres, l'intérêt et l'originalité d'un modèle résident surtout dans la présence de classes abstraites permettant une appréhension et des manipulations efficaces des métadonnées. De ce point de vue, l'état de l'art pertinent n'est pas tant celui des descriptions de traitements que celui des principes de représentation des connaissances. Les notions d'orienté-objet et de *frames* ont été sommairement évoquées dans les pages précédentes ; celles plus spécifiques aux systèmes à base de connaissances sont présentées au chapitre 3. La volonté d'exploiter informatiquement les connaissances d'expert a émergé à la fin de la première année de travail, notamment lorsque sont apparues les difficultés à répondre aux utilisateurs demandant une adaptation des modes d'emploi. La prise en compte de ce nouveau point de vue a alors rejailli sur le modèle, en particulier sur les classes présentées au chapitre 3.

Modèle conceptuel, modèle d'implémentation et application d'accès aux métadonnées

Un modèle conceptuel est une représentation d'un domaine exprimée dans un formalisme autant que possible neutre vis-à-vis des soucis d'implémentation. Un modèle d'implémentation est la traduction formelle d'un modèle conceptuel dans un langage informatique.

Dès le début de notre travail un langage d'implémentation a été choisi. En permanence, toute évolution du modèle conceptuel se traduisait immédiatement dans le modèle d'implémentation. Une application Web permettant de consulter la base de métadonnées naissante a été développée dès la première année de thèse. Cela a permis de recueillir très tôt les commentaires d'utilisateurs cobayes non seulement sur le modèle mais aussi sur l'interface de consultation. Ces deux types d'enseignements ont été mutuellement profitables. Nous verrons par exemple p. 104 comment le besoin de voir illustrées dans l'interface les données avant et après traitement a amené à enrichir

le modèle.

Le développement parallèle de l'application et du modèle a donc été profitable. Notre démarche a été incrémentale. En effet la succession de phases *"analyse – conception – implémentation – tests – validation"* [LJP98] a permis l'élaboration progressive d'un modèle dont la base de métadonnées instance supportait un nombre croissant de besoins d'information. Ce faisant, la faisabilité du projet répondant aux objectifs initialement définis a été contrôlée. Notamment, de façon récurrente, les utilisateurs cobayes nous ont mis en garde contre une complexité excessive du modèle.

Si construire tôt un modèle d'implémentation présente l'avantage de permettre des expérimentations précoces, cette démarche comporte néanmoins un risque, celui que la prise en compte des propriétés du langage adopté biaise notre façon d'appréhender le problème et parasite le modèle conceptuel. Soucieux d'éviter un tel travers, nous avons tenté de garder constamment claire la séparation entre les deux modèles, conceptuel et d'implémentation. La réalité de l'activité de modélisation n'est cependant pas si simple. D'abord parce que le formalisme du modèle conceptuel n'est pas neutre, ensuite parce que les caractéristiques des langages informatiques peuvent jouer le rôle de masque ou au contraire d'aide selon ce que l'on souhaite représenter.

2.2.2 Notre modèle de métadonnées est-il orienté objet ?

Parmi les spécifications qui accompagnaient notre sujet de thèse lors de notre arrivée au laboratoire COGIT, il figurait une demande particulière : le modèle conceptuel de métadonnées à définir devait être exprimé sous forme de diagramme de classes UML. Compte tenu de cet impératif, il nous parait utile de préciser notre position vis-à-vis des notions de l'orienté objet.

Une des raisons de la demande d'utilisation du formalisme UML était l'objectif d'interopérabilité avec une future plateforme[58] de métadonnées. L'idée qui présidait au sein de l'action de recherche Consul était que les différents modèles (de tâches, de métadonnées des traitements et de métadonnées des données) soient instanciés sous forme de classes Java.

Par ailleurs, et de façon liée, l'usage au sein du laboratoire COGIT est d'utiliser la notation UML comme support de communication[59]. Quant au langage Java, il est utilisé pour GeOxygene, l'une des deux plateformes de développement du COGIT.

Ce contexte étant posé, on peut se demander si notre modèle doit forcément être orienté objet, et si par conséquent UML est bien la notation la plus adaptée à ce que l'on souhaite exprimer (indépendamment du fait que le besoin d'un langage commun de communication est une raison suffisante pour l'adopter).

Il existe parfois une confusion entre l'idée de *représentation orientée objet* et celle de *programmation orientée objet*. Clairement, nous considérons que notre modèle relève de représentation OO mais non de la programmation OO. Nous allons voir que notre modèle fait appel à deux principes :

- celui de classe et d'instance,
- celui d'héritage.

Ces deux principes sont caractéristiques des langages de représentation OO. Mais prétendre faire de la programmation OO suppose de mettre en œuvre, en plus de ces principes, ceux d'*encapsulation* des propriétés et des méthodes, d'*abstraction* (visibilité variable des propriétés et des attributs entre les objets), de *polymorphisme* (mécanisme pour invoquer des méthodes

[58] Ensemble de structures de données, de logiciels et de librairies facilitant le développement ou l'exploitation de programmes.

[59] Néanmoins, au sein de l'IGN, il existe une tradition de formalisation en HBDS. Cette notation est en effet enseignée à l'ENSG par François Bouillé, qui en est le créateur.

désignées par un même nom, mais différentes selon le contexte) [LJP98]. Si l'on considère la programmation OO comme reposant avant tout sur l'encapsulation des propriétés et des méthodes, elle est un dépassement de la programmation procédurale. Dans notre contexte au contraire, l'exploitation envisagée de notre modèle dissocie clairement les données et les traitements qui leurs sont appliqués. Cette dissociation est d'ailleurs un principe que l'on retrouve dans diverses méthodes de conception de systèmes d'information : MERISE[60], par exemple, mais aussi plus récemment des méthodes utilisées notamment dans le domaine bancaire.

La notation UML est désormais le standard pour la modélisation en programmation OO. En revanche, elle n'est pas nécessairement la plus adaptée pour tous les modèles de représentation OO. On pourra préférer adopter d'autres notations pour certains modèles d'implémentation mettant en œuvre les notions de classes et d'héritage. On verra de plus qu'UML ne permet pas d'exprimer certaines contraintes de modélisation[61].

2.2.3 Réifier les familles de traitements

Les connaissances que nous voulons représenter portent non seulement sur des traitements ou ensembles de traitements (Buffer.java, Arcview, etc.), mais aussi sur des *familles de traitements* (SIG, logiciels Windows, programmes Java, etc.).

Prenons un exemple. Supposons qu'une nouvelle plateforme de développement *GéoEx* basée sur le langage Java doit être décrite. Les programmes de cette plateforme partagent un certain nombre de propriétés : sur les pré-requis pour les lancer, sur leur compatibilité, sur le domaine des fonctionnalités qu'ils réalisent, etc. Cette connaissance doit pouvoir s'exprimer grâce à notre modèle. Certes, il serait possible de proposer à l'expert de définir un ensemble de règles `"si appartientPlateforme(?prg, GéoEx) alors ..."`. Mais une telle solution n'est pas propice à un recueillement systématique et contrôlé des connaissances. L'abandon des premiers systèmes expert pour des systèmes basés sur les *frames* ou l'orienté objet le montre. Il est plus élégant de proposer à l'expert de décrire la famille de traitement *programmeGéoEx* comme un programme possédant les caractéristiques typiques attachées à ladite plateforme. Cela implique qu'il existe dans notre modèle une classe *FamilleTraitement*.

Les *FamilleTraitement* héritent les unes des autres : dans notre exemple, on peut ainsi imaginer que *programmeGéoEx* hérite de *programmeJava*.

Si les instances de *FamilleTraitement* étaient elles-mêmes des classes, *FamilleTraitement* serait une méta-classe. Ce n'est pas exactement le choix que nous faisons. En fait, notre modèle permet de dire, par exemple, qu'Arcview, instance de Logiciel, est lié aux deux instances de FamilleTraitement SIG et LogicielWindows par la relation[62] "type". Si l'on considère "type" comme une relation d'héritage, alors nous faisons là de l'héritage multiple.

"SIG" et "LogicielWindows", pour des raisons d'extensibilité et parce que ce sont selon notre point de vue des ressources à décrire, doivent figurer dans la base de métadonnées en tant qu'instances de classe et non dans le modèle en tant que classes. Si de nouvelles familles de traitements apparaissent, leur prise en compte ne doit pas affecter le modèle. Nous reviendrons sur la question au moment de présenter les diagrammes de classes.

[60] `http://www.commentcamarche.net/merise/concintro.php3`

[61] cf., entre autres, l'exemple sur l'association entre entrée et sortie p. 102, ou la restriction de propriétés fig. 2.16 et 2.17, p. 95. Pour permettre à ces contraintes d'apparaître dans les diagrammes de classes UML, le langage OCL (Object Constraint Language) a dû être créé.

[62] *relation* est un terme employé notamment dans le langage UML. Nous parlerons également de *propriété* des classes, qui sont des façons de représenter les relations. Dans les logiques de description, sur lesquelles reposent les ontologies que nous verrons plus avant, les relations sont appelées *rôles*, et les propriétés *attributs* ou *slots*.

La sémantique que nous attribuons à la relation “type” permet simplement de transmettre des valeurs de propriétés. Par exemple Arcview est de type LogicielWindows, donc son système d'exploitation est Windows.

Restrictions de propriétés

Nous avons besoin de contrôler les valeurs des éléments de nos descriptions. Cela commence par la restriction de l'ensemble des valeurs possibles pour une propriété. Les figures 2.16 et 2.17 donnent deux exemples de nos souhaits en la matière.

Un contrôle plus fin consiste à s'assurer de la cohérence entre différentes métadonnées. Par exemple, si un programme qui implémente un algorithme doit réaliser le même type de fonctionnalité que lui, les informations sur les types de données abstraits et implémentés doivent être cohérents. De même, les descriptions respectives d'une fonction et du logiciel à qui elle appartient ne sont pas sans relations. Les règles de cohérence ne pourront pas apparaître dans la définition du modèle. Elles seront donc exprimées en tant que méta-connaissances et utilisées par l'application exploitant la base de métadonnées.

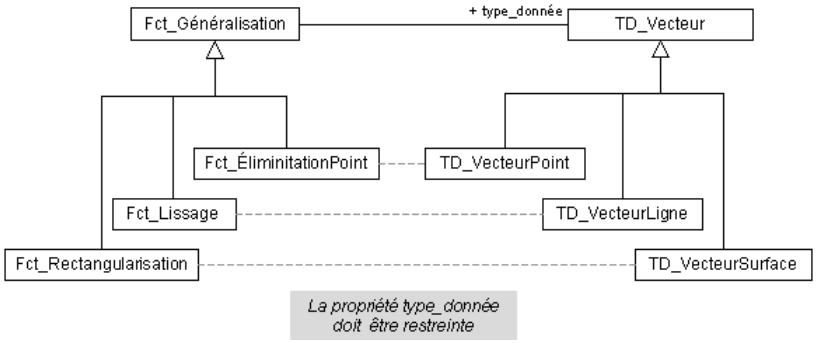


FIG. 2.16 – Restriction de l'ensemble des valeurs possibles pour la propriété type_donnée

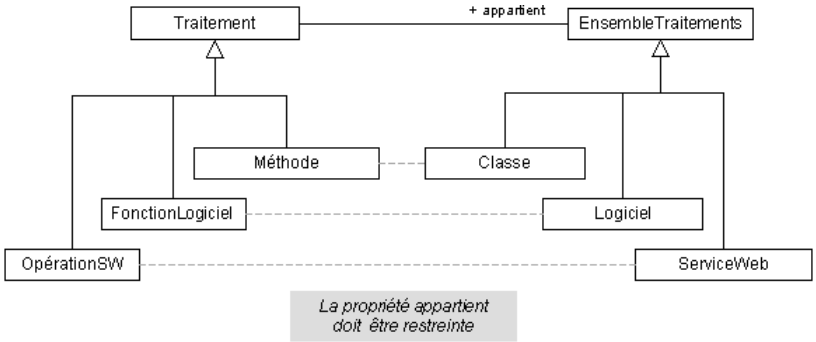


FIG. 2.17 – Restriction de l'ensemble des valeurs possibles pour la propriété appartient

2.3 Définition de notre modèle conceptuel de métadonnées

Les diagrammes de classes exposés dans cette section présentent notre modèle conceptuel de métadonnées. La clarté plutôt que l'exhaustivité a été recherchée, c'est pourquoi certaines relations et classes annexes ont été masquées. Le modèle d'implémentation sera lui présenté plus en détails. Par ailleurs, l'expressivité du modèle est illustrée par divers exemples. La traduction dans le modèle d'implémentation se trouvera dans la suite du mémoire, pour chacun d'eux. Enfin, accompagnant la présentation de certaines parties du modèle, des hypothèses expliquant les raisons de la spécificité des traitements géographiques ont été avancées.

Dans la suite du mémoire, les termes correspondant à des noms de classes ou de propriétés apparaissent en police sans serif. Nous considérons ces termes comme des noms propres, c'est pourquoi nous avons fait le choix de ne pas les accorder au pluriel. Par ailleurs, les classes dont le nom est noté en italique sont des classes abstraites.

2.3.1 Les traitements à décrire

Le diagramme figure 2.18 donne un aperçu général du modèle de métadonnées. On y voit que les différents types de traitements, désignés par le terme générique de RessourceTraitement, sont décrits selon cinq points de vue réifiés par les classes Identification, Fonction, Fonctionnement, ModeEmploi et Évaluation.

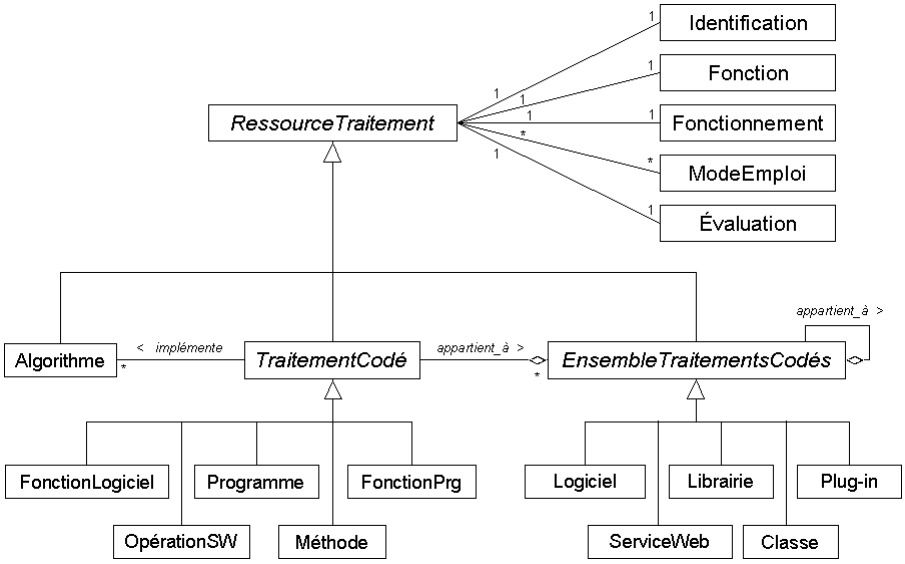


FIG. 2.18 – Classes principales du modèle de métadonnées

Certains des termes du modèle étant sujets à interprétations variables, précisons le sens recouvert, selon nous, par chacun d'eux.

RessourceTraitement : classe abstraite désignant tout traitement ou ensemble de traitements, ressources qui dans notre modèle ont en commun de pouvoir être décrites selon les cinq facettes de description définies. Comme nous l'avons précisé section 1.2.1 (p. 8), le terme "traitement" correspond dans notre contexte aux ressources qui réalisent ou décrivent comment réaliser une transformation d'information. Le cas particulier des traitements qui ne modifient ni ne créent de données est discuté p. 106.

Algorithme : "Suite finie d'opérations élémentaires constituant un schéma de calcul ou de résolution d'un problème" (Larousse)[63]. Plus particulièrement dans notre contexte, un algorithme est "une description de résolution de problème destinée à être implémentée sous

[63] Étymologiquement, le mot algorithme est dérivé du nom d'un mathématicien perse qui a vécu au IXème siècle,

forme de programme informatique" [Sed84], donc pour nous de TraitementCodé. Un algorithme est exprimé en langue naturelle ou en pseudo-code. Il n'est donc pas interprétable par une machine.

Exemples : Accordéon, Douglas & Peucker et Gauss sont des algorithmes de généralisation d'objets possédant une géométrie linéaire. Dans le cadre du projet Agent où de tels algorithmes sont développés, il est également question de mesures : *"a measure is a method that does not change the state of map objects, but is used to characterise it"* [Con99]. Dans notre modèle nous avons fait le choix de considérer les mesures comme des algorithmes, la distinction "transformation ou caractérisation" des entrées étant spécifiée dans la partie Fonction de la description. Par exemple une méthode de calcul de la distance de Hausdorf (plus petite distance entre deux objets) sera cataloguée dans notre modèle comme Algorithme.

TraitementCodé : classe abstraite désignant tout ensemble d'instructions d'un langage informatique, interprétable par une machine. Formellement, tout TraitementCodé implémente un Algorithme. Mais dans la base de métadonnées instanciant notre modèle, on trouvera des TraitementCodé dont l'Algorithme n'est pas catalogué (surtout si ce dernier est trivial). Un TraitementCodé peut appartenir à un EnsembleTraitementsCodés.

Nous décrivons ci-dessous les différents types de TraitementCodé.

Programme (sous-entendu, informatique) : ensemble d'instructions d'un langage informatique, interprétable par une machine, et possédant un point d'entrée qui permet de lancer son exécution sans passer par un autre programme. Un programme se présente soit sous forme de code source (instructions lisibles pour un programmeur, et exécutables dans le cas des langages interprétés comme le VisualBasic ou le Javascript), soit sous forme compilée (code binaire que l'on peut néanmoins visualiser sous forme d'instructions en assembleur, qui est le langage du processeur de l'ordinateur[64] ou, pour le langage Java, *byteCode*, intermédiaire entre code source et exécutable, destiné aux machines virtuelles Java). Le support de stockage d'un programme est en général le fichier informatique, mais le listing papier d'un code source sera également indexé, dans notre base, comme Programme. La caractéristique discriminante des Programme vis-à-vis des autres TraitementCodé, quand on dispose de sa forme compilée, est d'être invocable directement en ligne de commande (ce qui équivaut usuellement, dans les IHM des divers systèmes d'exploitations, à lancer le fichier exécutable). En effet un programme possède un point d'entrée, contrairement aux fonctions, procédures et classes Java dépourvues de méthode `main()`[65].

Une classe Java qui possède une méthode `main()` est donc cataloguée dans notre modèle comme Programme[66].

Exemples : programmes xls2tbl.ave qui permet la conversion de données Excel en vue d'une importation dans un SIG de la famille Arcview ; caricatures.ada qui réalise des caricatures de lignes vecteur.

FonctionProgramme : comme Programme, c'est un ensemble d'instructions d'un langage informatique, interprétable par une machine. La différence réside dans le fait qu'une Fonc-

Mohammed al-Khwârizmî (en latin Algorismus). Il a proposé un ensemble d'opérations élémentaires à exécuter séquentiellement, pour additionner, soustraire, multiplier et diviser des nombres décimaux [JJPJ04].

[64] Le langage assembleur varie selon les processeurs. Par exemple, le MIPS R3000 est un langage assembleur 32-bits développé par MIPS Technology, une filiale de la société Silicon Graphics. Par mesure de protection, certains auteurs cryptent parfois le code exécutable de leurs programmes. On ne peut alors pas les désassembler directement de façon lisible pour un humain.

[65] Le critère de distinction entre Classe et Programme est également retenu, entre autres, par L. Bodet, qui dresse un tableau des programmes et classes des packages Java du JDK ([Bod97], p.122)

[66] Ajoutons qu'en Java les méthodes `main()` sont obligatoirement `static()`, justement parce qu'elles doivent être invoquées en ligne de commande et non via une instruction de code. En effet, on invoque la méthode `Toto.main()` en exécutant la commande `java Toto` sans avoir au préalable créé d'objet par l'instruction `new Toto()`.

tionProgramme n'est pas invocable directement : pour être exécutée elle doit être appelée (par un Programme ou par une autre FonctionProgramme). Le terme de FonctionProgramme est utilisé, dans notre modèle, pour les langages de programmation non-orientés objet (*i.e.* procéduraux, fonctionnels, logiques ou autres). Pour ces langages, FonctionProgramme correspond à ce qui est usuellement appelé fonction ou procédure (procédure est une fonction qui ne renvoie pas de résultat[67]). Une FonctionProgramme appartient à une Librairie.

Exemple : `COGITgeom_line_false_semi_hausdorff` qui renvoie un entier.

Méthode : équivalent de FonctionProgramme pour les langages orientés objet. Outre les valeurs de la propriété langage, la distinction entre Méthode et FonctionProgramme apparaît dans les règles de cohérence des relations qu'ont ces objets avec les objets de la classe EnsembleTraitementsCodés.

Exemple : méthode `addSegment()` de la classe `GM_Curve`.

FonctionLogiciel : TraitementCodé qui présente la particularité d'appartenir à un Logiciel et d'être invocable via l'IHM de celui-ci. Deux cas de figures sont possibles.
Soit la FonctionLogiciel correspond à un Programme, une FonctionProgramme ou une Méthode dont l'invocation, outre via l'IHM, est possible pour l'utilisateur via la programmation ou l'exécution d'une ligne de commande. Dans ce cas le Programme, la FonctionProgramme ou la Méthode en question est référencé par la FonctionLogiciel, et lui transmet une partie de ses éléments de description.
Soit, et c'est le cas le plus courant, la FonctionLogiciel ne correspond à aucun TraitementCodé indexé car elle est encapsulée dans un logiciel "boîte noire".

Exemples : AddTheme est une FonctionLogiciel. Elle appartient et est accessible via le Logiciel Arcview.
ConversionMilesKm est une FonctionProgramme. Elle appartient à la Librairie maLib.c. Supposons qu'au moment de son indexation dans la base de MDT, elle n'est invocable que par l'écriture d'un Programme. Supposons maintenant que l'on dote l'IHM d'Arcview d'un nouveau menu permettant d'invoquer la FonctionProgramme ConversionMilesKm. Vis-à-vis de la base de métadonnées, on choisit alors d'indexer une nouvelle FonctionLogiciel ConversionMilesKm, qui porte une référence vers la FonctionProgramme ConversionMilesKm.

OpérationSW : TraitementCodé qui présente la particularité d'être invocable via le Web. Une OpérationSW appartient à, et est fournie par un ServiceWeb, dont la description – typiquement en WSDL[68] – spécifie les modalités d'invocation (protocole Web employé, port, format des messages, etc.).

De façon analogue aux FonctionLogiciel, une OpérationSW peut correspondre à un Programme, une FonctionProgramme ou une Méthode référencée dans la base de métadonnées instanciant notre modèle.
Le terme "opération" que nous utilisons est celui que l'on trouve dans la littérature. Le langage WSDL, notamment, y recours. Nous avons néanmoins décidé d'ajouter le suffixe "SW" (pour Service Web) afin d'éviter toute ambiguïté.

Exemples : xmlUpload.do est une OpérationSW du ServiceWeb crsClient. Elle permet la transformation de coordonnées de données géographiques codées selon une grammaire XML[69].

EnsembleTraitementsCodés : classe abstraite désignant un ensemble de TraitementCodé.

[67].. du moins, qui ne renvoie pas directement de résultat au code appelant.
[68]cf. section2.1.2, p. 60
[69]Ce service Web est développé par l'équipe projet "Diffusion des données numériques sur le réseau" de l'IGN.

Nous décrivons ci-dessous les différents types de EnsembleTraitementsCodés.

Classe : En programmation orientée objet, une classe est une structure de données comportant des propriétés et des méthodes pour un type donné d'objet, à partir duquel sont créés des objets concrets possédant des valeurs particulières. Dans notre modèle, Classe est à la fois un EnsembleTraitementsCodés et un TypeDonnéeImplémenté.

Exemple : `GM_LineSegment` est une classe de l'API GeOxygene[70] qui permet de représenter les segments de droites.

Logiciel : "Ensemble des programmes, des procédures et de la documentation et des données éventuellement associées." (selon l'ISO 12207[71]). Nous ajoutons que, en accord avec le sens commun (et de façon plus ou moins arbitraire), un logiciel doit être doté d'une IHM et posséder plusieurs FonctionLogiciel. Ce double critère nous permet de distinguer les Logiciel des Programme. Cette distinction est une convention nécessaire dans le cadre de notre travail. Néanmoins, globalement, dans la littérature, les logiciels sont qualifiés de programmes.

Exemple : Arcview est un SIG (Sytème d'Information Géographique) qui permet de réaliser plusieurs fonctionnalités et qui est doté d'une IHM. C'est donc un Logiciel.

Librairie : ensemble de Programme et/ou de FonctionProgramme. Nous avons choisi le terme "librairie" parcequ'il est communément utilisé dans la communauté des informaticiens[72]. Une Librairie n'est pas forcément *open source*. On n'a parfois accès qu'à son interface. Par exemple, en environnement Microsoft, on accède aux fonctions des objets COM via leur interface IDL(Interface Definition Language), mais on n'a souvent pas accès à leur code. Pour notre modèle de métadonnées, les objets COM sont des librairies, contenues dans d'autres librairies que sont les DLL (Dynamic Link Librairies). De façon équivalente, les packages Java (stockés dans les fichier JAR – *i.e.* Java Archive), sont des librairies qui contiennent des classes Java, lesquelles peuvent être des composants JavaBean. Les plate-formes et les API sont, dans notre modèle, des Librairie particulières qui sont composées d'autres Librairie. Telles des poupées russes, les traitements codés sont ainsi organisés par regroupements successifs en librairies de granularité croissante.

Exemples : GMT est une librairie gratuite de 60 fonctions Unix ou DOS pour la production de document au format EPS à partir de données 2D ou 3D[73].
COGIT_lib_geom.lull est une librairie d'une cinquantaine de fonctions Lull pour la manipulation et les calculs d'objets géométriques.

Plug-in : "extension à une application qui vient se loger dans l'application elle-même"[74]. Un Plug-in, contrairement à une Librairie, est toujours associé à un Logiciel. De plus, un Plug-in, lorsqu'il s'intègre à son Logiciel, provoque automatiquement l'enrichissement de l'interface de ce dernier. Les Librairie, elles, sont davantage associées à l'idée de programmation. Pour apparaître dans l'interface d'un Logiciel, le développement de code extérieur à la Librairie est en effet nécessaire.

Exemple : 3D Analyst est un plug-in du logiciel Arcview pour la visualisation des données 3D.

ServiceWeb : en accord avec la définition communément admise[75], un ServiceWeb est pour

[70] cf. section 1.2.4

[71] Définition de l'ISO citée par `http://www.alaide.com/dico.php?q=Logiciel&ix=1540`

[72] La raison de cet usage est peut-être à chercher dans l'identité phonétique avec le faux-ami "librairy" dont la traduction est en fait "bibliothèque").

[73] `http://gmt.soest.hawaii.edu/`

[74] Dictionnaire informatique Foldoc `http://www.linux-france.org/prj/jargonf/P/plug-in.html`

[75] Par exemple, "A WSDL document defines services as collections of network endpoints, or ports. (...) port types which are abstract collections of operations" [W3C01].

nous un ensemble d'OpérationSW. Un ServiceWeb permet l'invocation par un client de ses OpérationSW via un protocole Web défini, par exemple HTTP ou IIOP (Internet Inter-ORB Protocol).

Exemple : API GoogleMap.

Précisions et conventions à propos de la notation des diagrammes de classes

Les classes sont liées par des relations d'héritage, d'agrégation, de composition et d'association. Considérons deux classes quelconques liées par une relation d'association. Pour représenter cette dernière nous définissons, le plus souvent, une propriété dans une des deux classes. Il arrive également que nous réifions la relation sous la forme d'une troisième classe.

Parfois, il arrive que nous voulions juste indiquer l'existence d'une relation, sans préjuger de la façon dont cette existence se traduira dans le modèle d'implémentation. La notation adoptée est alors un arc orienté entre les rectangles symbolisant les deux classes, surmonté du nom en italique de la relation, accompagné éventuellement des informations de cardinalité.

2.3.2 Identification d'un traitement

Quel est le nom d'un traitement ? Qui l'a créé ? Où et quand ?... Pour répondre à ces questions classiques d'indexation, nous avons défini un ensemble de descripteurs. Un certain nombre correspond à ceux proposés par le Dublin Core (cf. p. 82).

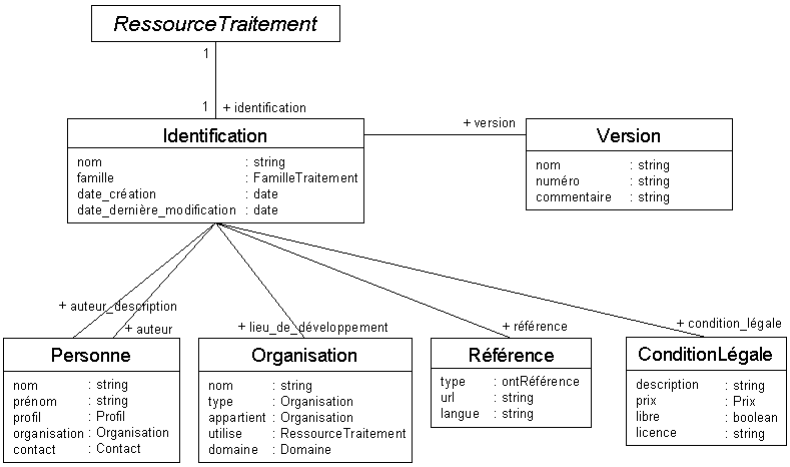


FIG. 2.19 – Identification d'un traitement

Apportons quelques commentaires sur les classes du diagramme figure 2.19.

La classe Version doit son existence au besoin d'ordonner les traitements en fonction de leur numéro de version. Le type simple "nombre réel" n'aurait pu convenir en raison de la présence fréquente de chaînes de caractères telles que "bêta", "NT", "XP", etc. La propriété nom vient donc compléter celle du numéro, qui est de la forme `(nombre(.nombre)*)` sur laquelle la relation d'ordre lexicographique peut être appliquée (par exemple $1.4.1 < 1.5.0$; malheureusement il arrive que pour des raisons de *marketing* le mode de numérotation change subitement, par exemple de 3.1 à 95 puis à 2000. On est alors contraint de se baser sur la date de création du traitement).

La propriété commentaire n'est destinée qu'à recueillir des informations propres à la version du traitement considéré. La description des différences entre version d'un même traitement, ou plus généralement entre deux traitements quelconques, s'effectue au moyen de la classe Différence (classe annexe non exposée).

Les classes Contact (tél, mail, etc.) et Prix (valeur, monnaie) liées respectivement à Personne et ConditionLégale ne sont pas détaillées. La classe Profil caractérisant Personne sera détaillée plus loin dans le mémoire (p. 139.).

Lorsque sa valeur est zéro, la propriété prix de ConditionLégale signifie la gratuité de la RessourceTraitement. Cette caractéristique n'est pas à confondre avec la propriété libre, qui pour nous est vraie si le code source est disponible, modifiable et redistribuable librement. Ainsi, le logiciel Acrobat Reader d'Adobe est gratuit mais n'est pas libre. De même, pour prendre un exemple de SIG, ArcExplorer[76] de la société ESRI est gratuit mais n'est pas libre.

Une définition plus complète que la nôtre du "libre" est proposée par l'Open Source Initative (OSI)[77].

D'une façon générale, les types contraints (comme ceux des propriétés numéro de version ou url), énumérés, ou prenant leur valeur dans des ontologies, sont qualifiés dans ce chapitre de "string". Ils seront détaillés dans le chapitre 4 présentant le modèle d'implémentation.

2.3.3 Décrire ce que fait un traitement

Dans notre modèle, la description de ce que fait un traitement consiste essentiellement en la donnée de trois informations (cf. fig. 2.20) :

1) la description des fonctionnalités réalisées,
2) la description des entrées, sorties, et paramètres,
3) la description de l'évolution des propriétés des données avant et après traitement.

Le terme "Fonction", que nous avons choisi pour désigner l'ensemble de ces informations n'est pas ici à confondre avec les formes de traitements que sont les FonctionProgramme, correspondant à des portions de code. Pour faire le parallèle avec un des modèles vus dans l'état de l'art, notre Fonction correspond au ServiceProfile d'OWL-S (cf. fig. 2.5 p. 64).

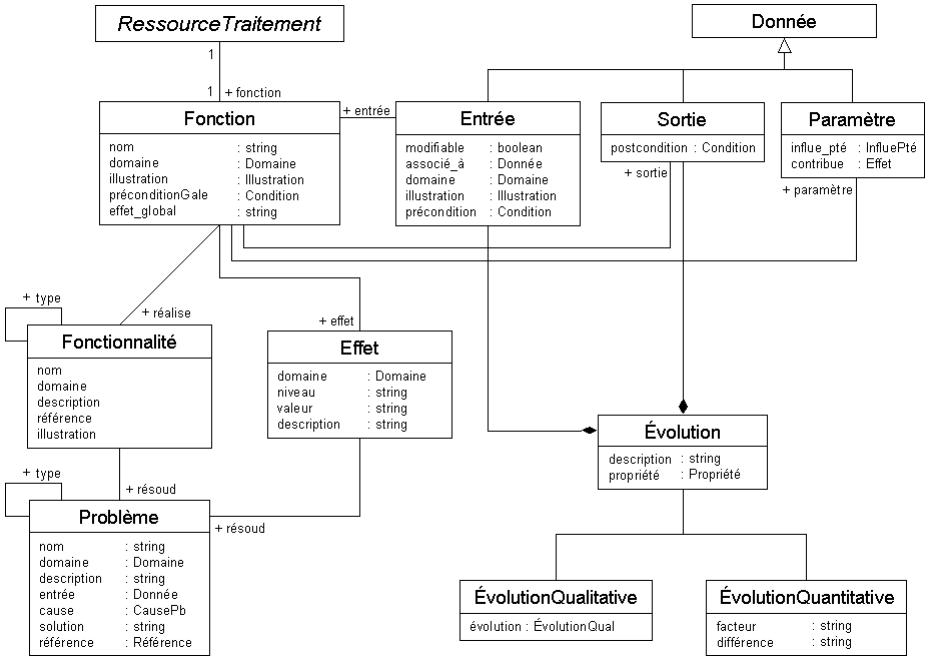


FIG. 2.20 – Fonction – description de ce que fait le traitement

[76] http://www.esri.com/software/arcexplorer/about/overview.html
[77] http://www.opensource.org/

Dans la suite de cette section nous allons détailler successivement les trois aspects de Fonction sus-cités.

1) Description des fonctionnalités réalisées

La classe Fonctionnalité représente le concept de fonctionnalité, qui est une abstraction pour désigner ce que font les traitements. Les fonctionnalités sont plus ou moins spécifiques. La propriété type représente la relation de spécialisation. Une taxinomie des fonctionnalités peut ainsi être décrite au niveau des instances du modèle. Par exemple *lissage* est une spécialisation de *simplification*, qui est une spécialisation de *généralisation*.

Outre son type, une Fonctionnalité est décrite par ses Entrée et Sortie. Il existe bien sûr des conditions de cohérence entre la description d'un traitement et celle de la fonctionnalité qu'il réalise.

2) Description des entrées, sorties et paramètres

Incidemment, décrire les traitements demande de décrire les données[78]. Les classes Entrée, Sortie et Paramètre (diagramme figure 2.20) sont des sous-classes de Donnée (diagramme figure 2.21). Détaillons quelques-unes de leurs propriétés à l'aide d'exemples.

Un traitement de rectangularisation prend en entrée une donnée contenant des objets à la géométrie surfacique, par exemple des bâtiments. L'entrée est modifiable : à moins que les objets ne soient déjà rectangulaires, leur géométrie est modifiée. L'entrée est associée à une sortie. Un traitement qui mesure l'aire d'objets surfaciques possède également une sortie (la valeur de l'aire), mais ne modifie pas l'entrée.

Une entrée modifiable est obligatoirement associée à une entrée. Ceci est un exemple de contrainte qui n'apparaît pas dans le diagramme.

Une entrée peut être associée à des sorties ; elle peut également l'être à d'autres entrées[79]. Par exemple le programme ContourV2[80] prend deux entrées : un fichier image TIFF (Tagged Image File Format) et un fichier de géoréférencement TFW (Tiff World File). Ces deux entrées sont associées. De même, les thèmes Arcview sont classiquement constitués de trois fichiers associés (SHP, DBF et SHX). Nous considérons dans de tels cas avoir affaire à trois entrées associées et non à une unique constituée de trois fichiers. Ce choix est justifié par l'existence de cas de traitements où ces fichiers sont pris isolément ; il est alors souhaitable d'unifier la façon de décrire les entrées en nous en tenant à la règle "un fichier – une entrée".

Une donnée est décrite par son format (p.ex. *shape*), son type abstrait (p.ex. surface en géométrie vecteur), son type "implémenté" (p.ex. classe `GM_Surface` de l'API GeOxygene) et ses propriétés (p.ex. origine : BD Topo).

Le diagramme de classe 2.21 montre comment nous modélisons les données.

Type de donnée abstrait et type de donnée implémenté

Considérons l'exemple du programme de lissage Lowe implémenté en Java sur la plateforme GeOxygene du laboratoire COGIT. À la question : *"quel est le type des données d'entrée de ce programme ?"*, l'expert répondra : *"il s'agit de données vecteur à la géométrie linéaire"*. Si

[78] Dans notre contexte une donnée est une "représentation conventionnelle d'une information sous une forme convenant à son traitement par ordinateur" (Larousse).

[79] Il existe un point de vue où, algorithmiquement parlant, seuls sont considérés les entrées, les sorties et les entrées-sorties. Ce point de vue est compatible avec le nôtre. Les "entrées modifiables" de notre modèle correspondent aux "entrées-sorties" du point de vue évoqué ; les "entrées non modifiables" et les "paramètres" de notre modèle correspondent aux "entrées" du point de vue évoqué ; les 'sorties" de notre modèle correspondent aux "sorties" du point de vue évoqué.

[80] Conçu en 2004 à l'IGN (service SBV) par S.Motet et J.Pêcheur, ce programme vectorise des images raster noir et blanc au format TIFF ; il livre en sortie des données vecteur aux formats Géoconcept ou SVG (Scalable Vector Graphics).

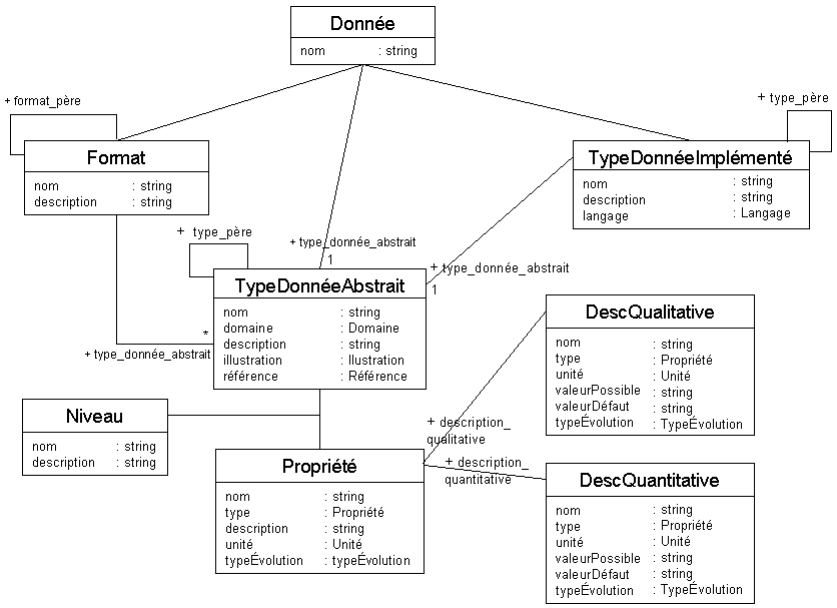


FIG. 2.21 – Données – description des entrées, sorties et paramètres

maintenant nous reposons la même question en précisant *"du point de vue du code Java"*, la réponse sera *"GM_LineString"*.

Dans notre modèle, une Donnée (Entrée, Sortie, ou Paramètre) est donc caractérisée par un TypeDonnéeAbstrait et un TypeDonnéeImplémenté. On retrouve là la dualité abstrait/implémenté qui distingue fonctionnalités abstraites (Fonctionnalité) et traitements codés (TraitementCodé).

Parmi les objets instances de TypeDonnéeImplémenté, on trouve les types simples (entier, réel, chaîne de caractères, etc.), les structures de données définies dans différents langages, et enfin les Classe[81]. Les Classe héritent donc dans notre modèle à la fois de TypeDonnéeImplémenté et de EnsembleTraitementsCodés. C'est la traduction de la conception objet encapsulant propriétés et méthodes.

Choix de la modélisation des types de données

Le standard ISO 19107 définit la hiérarchie des classes géométriques de base [ISO01a] (cf. fig. A.1, p. 229). Notre façon de modéliser les types de données se situe à un niveau d'abstraction supérieur. En effet, les classifications des types de données apparaissent au niveau des instances de notre modèle et non dans notre modèle lui-même.

Les propriétés des types de données

Chaque type de donnée, abstrait ou implémenté, peut posséder des propriétés. Par exemple une ligne vecteur est caractérisée par les propriétés longueur, direction, points d'inflexion, sa sinuosité, sa sémantique (route nationale p.ex.), etc.

Chaque propriété peut être décrite par des valeurs minimum, maximum, par défaut ; par des contraintes sur les valeurs autorisées ; par les types d'évolutions possibles d'un point de vue qualitatif et quantitatif.

Un type de donnée peut regrouper ses propriétés par niveau. Par exemple, dans le cas des traitements de généralisation des bâtiments, on utilisera les niveaux *micro, méso, macro*[82] du

[81] Le lien d'héritage n'est pas montré dans le diagramme fig. 2.21.

[82] Niveaux introduits dans le domaine de la généralisation par [Rua99].

TypeDonnéeAbstrait "ensemble de bâtiments[83]" (propriétés respectives *position, orientation, aspect, sémantique, forme, distribution, répartition sémantique, orientation spatiale ; proximité, topologie ; quantité objets, répartition sémantique*). Pour un traitement de généralisation de réseau routier, ce pourra être en revanche les niveaux *sémantique, géométrie, topologie*[84] du TypeDonnéeAbstrait *"ensemble de routes*[85] *"* qui seront jugés pertinents.

Le modèle de description des Entrée permet ainsi d'exprimer des préconditions précises de façon standardisée. Par exemple, *"le traitement de généralisation T ne marche que sur la BD Topo car il se base sur l'attribut direction des routes"*. On remarque avec ce dernier exemple que notre modèle doit nécessairement indexer les différentes BD géographiques afin de valuer les propriétés "origine" des types de données.

Voici un autre exemple de précondition exprimable, portant à la fois sur une propriété de l'entrée et sur l'environnement matériel de l'utilisateur : *"le logiciel FreeWRL de visualisation VRML a pour précondition : pour un environnement de travail où la mémoire vive est de 512 Mo et le système d'exploitation Linux SuSE*[86] *10.0, le nombre d'objets de l'entrée doit être inférieur à 5000"*[87].

La finesse dans la description des propriétés des données et des pré- et post-conditions des entrées/sorties peut être améliorée, afin de retarder le moment où il faut se résigner à recourir à la langue naturelle. On ne peut cependant augmenter indéfiniment le nombre de descripteurs, au risque de rendre le modèle inutilisable (en particulier en ce qui concerne la phase d'acquisition, souvent le maillon faible du cycle de vie des métadonnées). Ainsi, les deux exemples qui précédent se situent à la limite maximale de ce que le modèle permet d'exprimer de façon standardisée. L'analyse des besoins n'a pas, selon nous, fait apparaître de demande davantage poussée en terme de requête de l'utilisateur ou d'exploitation de la base de métadonnées.

La norme ISO 19115 de métadonnées des données géographiques propose plusieurs dizaines de propriétés pour décrire les données [ISO03]. Ces propriétés pourront apparaître comme instance de notre classe Propriété (cf. fig. A.2, p. 230). En procédant ainsi, nous nous assurons, en théorie, de la compatibilité entre nos métadonnées des traitements et les métadonnées existantes conformes à ISO 19115. Réutiliser directement les classes ISO 19115 était un choix possible. Nous ne l'avons pas fait car nous préférions nous situer au niveau "méta" supérieur. Cela nous a permis d'attribuer des propriétés (type, description, unité, typeÉvolution, description_qualitative, description_quantitative) à la classe Propriété.

3) Description des effets, évolution des propriétés des données avant et après traitements, illustrations graphiques

Des expérimentations ont été menées au cours de l'élaboration du modèle. Il s'agissait de recueillir les impressions d'utilisateurs face aux descriptions accessibles via l'application de consultation développée. Un des enseignements notables, qui a impliqué une évolution du modèle, porte sur la façon de décrire les données *avant* et *après* traitement. L'utilisateur apprécie que soit présentée l'évolution des valeurs des propriétés des données (d'un point de vue quantitatif et qualitatif) et que, en complément, les états avant/après soient illustrés graphiquement. Ce dernier point est spécialement utile pour les traitements géographiques qui présentent en effet souvent un aspect visuel. C'est pourquoi nous avons introduit les objets Échantillon, poursuivant ainsi l'idée qu'a mise en œuvre F.Hubert dans la thèse qu'il a effectué au COGIT en 2003, et

[83] Sous-type de ensemble de surfaces vecteur.
[84] Niveaux utilisés dans les grilles de descriptions OEEPE [Rua98].
[85] Sous-type de ensemble de surfaces vecteur.
[86] Software und SystemEntwicklung, développement logiciel et système.
[87] Estimation d'après les tests effectués au sein du laboratoire COGIT, dans le cadre de l'action de recherche travaillant sur les triangulations de MNT.

qui visait à aider l'utilisateur à paramétrer les traitements de généralisation [Hub03].

Les problèmes que résolvent les traitements

Un de nos soucis étant d'homogénéiser la façon de décrire les traitements[88] – donc également leurs effets –, la classe Problème a été créée. On indique ainsi par exemple, de façon contrôlée, qu'un traitement de caricature résout un problème d'empâtement.

Cela suppose qu'une typologie des problèmes géographiques résolubles par les traitements soit définie.

C'est peut-être une particularité de certains types de traitements géographiques que de résoudre des problèmes, quand d'autres traitements réalisent "simplement" des fonctionnalités. D'une façon générale, on se doute qu'effectuer des transformations de formats de données crée des "problèmes". Or, de ce point de vue, le cycle de vie de l'information géographique est une succession de transformations de format : de l'acquisition des photos aériennes à la cartographie issue des bases de données vecteurs, des traitements sont utilisés pour résoudre ce qu'on appelle des Problème. Par exemple, au laboratoire MATIS sont corrigés sur les photographies aériennes les problèmes d'absence d'orthogonalité des bâtiments (ils apparaissent parfois penchés), les problèmes d'ombres ; sur les cartes scannées divers artefacts, etc. Une fois les données vectorisées, l'utilisation des SIG est précédée de problèmes de projection à résoudre, et de formats à convertir parmi les nombreux existants. La mise en correspondance de jeux de données provenant de différentes BD [Ges05], voire des schémas de BD eux-mêmes [She05], est à nouveau la source de problèmes. Dans le domaine de la 3D, les différentes techniques de modélisation comme la triangulation produisent des artefacts (p.ex. crêtes plates ou rivières qui remontent) qui sont autant de problèmes à résoudre [Rou04]. Les traitements de généralisation, enfin, ont, par nature, vocation de résoudre des problèmes de lisibilité cartographique qui se produisent lorsqu'on passe d'une échelle à une autre.

Problème : ce terme qualifie, pour nous, tout obstacle à la réalisation d'une Fonctionnalité. Le plus souvent, il s'agira de caractéristiques non désirées des données : type, format, valeur de propriétés (origine, taille, etc.) ; il pourra s'agir plus généralement d'une "mauvaise" (au regard de l'usage attendu) caractéristique d'une ressource quelconque : langages incompatibles, erreur dans le code d'un programme, etc.

La classe Problème comporte huit propriétés. Parmi celles-ci, figure solution, de type chaîne de caractères, c'est à dire destinée à recueillir du texte en langue naturelle. Cet élément de description sert à fournir des explications, mais pas à référencer des Fonctionnalité. En effet le lien entre Problème et Fonctionnalité est porté par cette dernière. Par exemple, la Fonctionnalité "création de MNT optimisé" résout le Problème "triangles plats". En effet un problème existe indépendamment de l'existence d'une Fonctionnalité qui le résout, alors qu'en général la réciproque n'est pas vraie, puisque la raison d'être d'une Fonctionnalité peut être de résoudre un problème.

Néanmoins, la propriété solution a dû être définie pour les cas où les Problème ne sont pas résolus par des Fonctionnalité référencées, mais par des actions de l'utilisateur. Par exemple, le Problème "exécution d'un programme Java" qui se manifeste par le message d'erreur ``J2EE.Lang.UnsupportedClassVersionError" a pour solution "utiliser une machine virtuelle Java aussi récente que le compilateur".

La classe CausePb n'est pas détaillée figure 2.20. Ses propriétés sont : description, connaissance (lien vers une Connaissance relative au problème), fonctionnalité (qui peut être la cause du problème) et référence (vers un document).

L'effet des traitements sur les données et sur "l'environnement"

[88] Conformément à l'objectif O4

Les traitements informatiques ne font pas que modifier ou créer des données persistantes. Ils peuvent aussi avoir pour effet de modifier l'affichage de l'écran, de modifier le comportement d'un logiciel, d'interagir avec un périphérique de l'ordinateur. A-t-on alors encore affaire à des traitements ? Nous considérons que oui.

Prenons l'exemple de deux programmes de détection de carrefours[89]. Les deux prennent en entrée les tables d'une base de données contenant les objets un réseau routier (objets ponctuels et linéaires). En sortie, le premier programme livrera la table des objets ponctuels modifiée par l'ajout d'un attribut "carrefour", tandis que le second n'affichera qu'une fenêtre temporaire d'information pour indiquer les résultats à l'utilisateur.

Vis-à-vis de notre modèle, le premier programme modifie l'entrée fournie, au contraire du second qui ne modifie que ce que l'on pourrait appeler "l'environnement utilisateur". Pour décrire ce dernier cas, on crée le type de donnée abstrait "interaction périphérique", qui se spécialise en "message utilisateur", "changement affichage écran", "impression imprimante", etc[90].

Distinguer les traitements qui créent ou modifient les données de ceux qui ne le font pas peut parfois sembler délicat. En effet, du point de vue de l'ordinateur, toute instruction informatique a pour effet de modifier l'état de la mémoire (registres inclus) et éventuellement interagir avec les périphériques[91]. Néanmoins, pour faire la distinction dans notre modèle entre les traitements tels que ceux de notre exemple de détection de carrefour, nous introduisant le critère suivant : nous considérons qu'il y a création ou modification de données lorsque la durée de vie en mémoire des données du résultat excède celle du processus du traitement qui les a crée.

Exposé de l'évolution de leurs propriétés : une information parfois insuffisante pour la description des effets

Les effets de certains traitements n'ont pour ainsi dire pas à être décrits car la spécification de leur sortie parle d'elle-même. Par exemple, un appariement de deux jeux de données d'objets produit la liste des objets appariés ; une triangulation de MNT produit un ensemble de triangles. Comprendre l'effet de ces traitements nécessite bien sûr la connaissance des domaines respectifs et la qualité des résultats peut être interprétée, mais la description des sorties ne fait pas de mystère quant à la Fonctionnalité réalisée.

En revanche, les effets de certains traitements ne peuvent être saisis sur la simple base de l'exposé des modifications des propriétés des données. Par exemple, en généralisation, si plusieurs propriétés sont modifiées simultanément, l'effet global du traitement n'apparaît pas forcément. Il faut alors créer une propriété plus générale résultat de l'agrégation des propriétés modifiés. Cette problématique est montrée par [Bar04]. Ainsi, indiquer que des bâtiments ont vu leurs propriétés orientation, taille et position modifiées ne permet pas de comprendre que l'effet du traitement est de les aligner. On voit pour de tels cas la nécessité des illustrations graphiques et des descripteurs effet et problèmes résolus. On voit également l'intérêt de descriptions à la fois qualitatives et quantitatives, l'attribution de valeurs aux propriétés devant souvent, pour être signifiantes, relatives à des critères plutôt qu'absolues. Une base de métadonnées instanciant le modèle peut ainsi à la fois permettre des requêtes portant sur les effets sur des propriétés précises (*"quels sont les traitements qui diminuent la sinuosité des lignes vecteurs fournie entrée ?"*) ou générales (*"quels sont les traitements qui ont pour effet de modifier la géométrie et non la topologie des lignes vecteurs ?"*).

Effets des traitements et paramétrage

[89] Il existe au laboratoire COGIT un programme développé par Éric Grosso, qui détecte les carrefours en Y, en T, en croix et en étoile.

[90] Seule apparaît au niveau de notre modèle conceptuel la classe parente TypeDonnéesAbstrait.

[91] Même l'instruction assembleur `nop` (no operation), dont le rôle n'est que de consommer le temps d'un cycle processeur, ne fait pas exception : elle réalise bien une opération d'écriture dans la mémoire. Par exemple dans les processeurs MIPS R3000, le `nop` écrit la valeur 0 dans le registre R0 – ce qui équivaut à ne rien faire puisque ledit registre vaut toujours 0.

Les relations entre paramétrage et effets des traitements sont parfois difficiles à décrire. C'est le cas dans le domaine de la généralisation cartographique. Des travaux traitent spécifiquement du problème. Par exemple, F. Hubert a réalisé une thèse pour aider l'utilisateur à spécifier ses besoins de symbolisation cartographiques, le système informatique proposé se chargeant de traduire les informations recueillies en terme de paramétrage de traitements de généralisation [Hub03]. La description des effets des traitements en fonction du paramétrage est également complexe dans d'autres domaines géographiques tels l'appariement de bases de données ou le traitement d'image.

La modélisation que nous proposons figures 2.20 et 2.21 permet de décrire le rôle des paramètres de façon limitée mais générique. Il est possible d'indiquer pour chaque paramètre l'influence sur les propriétés des sorties en particulier et sur les effets du traitement en général. Ce niveau de formalisation donne déjà des descriptions de taille conséquente. C'est pourquoi les informations complémentaires seront données en langue naturelle. Pour aller plus loin dans la description des effets des paramètres, il faudrait enrichir notre modèle pour représenter les résultats des *plans d'expérience* et des *statistiques descriptives*. Ces techniques proposent des méthodes pour évaluer l'impact de la variation d'une variable ou d'un ensemble de variables sur les autres. Le traitement considéré est alors considéré comme une boite noire dont on n'observe que les entrées, paramètres et sorties. L'explosion combinatoire interdit une application manuelle de ces méthodes. Nous n'avons pas suivi la piste de travail constituée par les techniques évoquées, intéressantes mais paraissant spécifiques au regard de la généricité de nos besoins.

Idéalement, exhiber les effets du paramétrage devrait passer par la démonstration d'exécutions réelles des traitements. En effet, avec des exemples types et des images prémémorisés on ne peut illustrer qu'une plage limitée de paramétrage, et encore seulement pour une donnée d'entrée particulière.

Si cela était possible, les réponses aux questions des utilisateurs sur le paramétrage devraient s'appuyer sur le stockage massif des résultats de toute une gamme de paramétrage pour chacun des éléments d'un échantillon type de données[92]. Mais il serait plus élégant, et moins coûteux en mémoire, de réussir à simuler l'exécution des traitements au moyen d'une modélisation qui en capture les aspects comportementaux pertinents pour l'utilisateur, éventuellement en les approximant.

Il est clair que l'obtention de fonctions prédictives du comportement pose un problème énorme d'acquisition, et ce par ailleurs pour une utilité quand même incertaine. Néanmoins, en théorie, étudier comment ces fonctions peuvent être établies peut constituer un sujet de recherche intéressant.

2.3.4 Décrire comment fonctionne un traitement

Dans notre modèle, une RessourceTraitement peut être implémentée (TraitementCodé ou EnsembleTraitementsCodés) ou non (Algorithme). Cette alternative se traduit dans la description du Fonctionnement par l'existence des deux sous-classes FonctionnementTraitementCodé et FonctionnementAlgorithme.

Nous ne la montrons pas ici, mais une petite extension de notre modèle consiste à spécialiser la classe FonctionnementTraitementCodé en fonction des différents paradigmes de programmation. Cela est utile, par exemple, pour décrire le programme de généralisation cartographique développé au COGIT par C. Duchêne [Duc04]. Des agents sont mis en œuvre. Les éléments de description but et contrainte sont spécifiques à ce type de paradigme de programmation. Autre exemple, le programme d'appariement de schémas de base de données développé par David

[92] Idée qui a déjà été mise en œuvre par F.Hubert avec son outil d'aide au paramétrage de traitements de généralisation [Hub03].

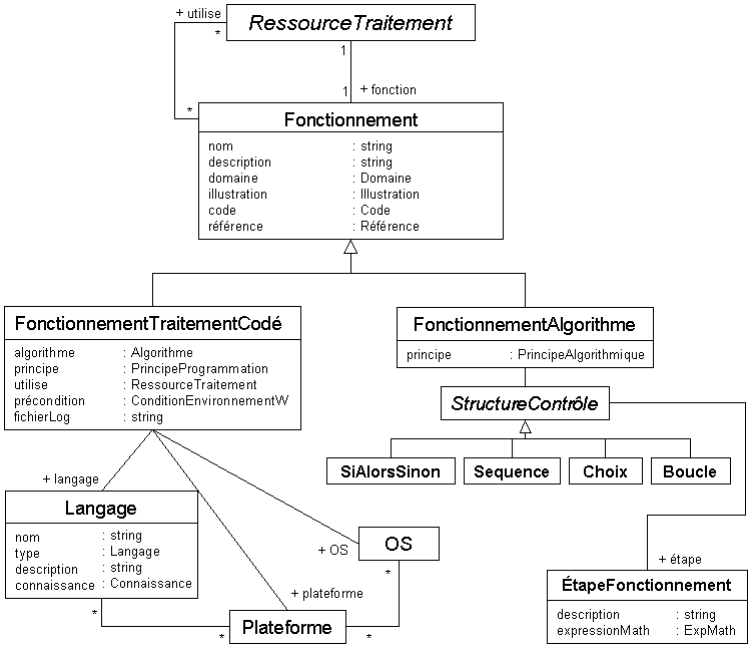


FIG. 2.22 – Fonctionnement d'un traitement

Sheeren, également au COGIT ([She05]). Il est basé sur le principe des systèmes experts ; sa description indique qu'il fonctionne avec la librairie Jess, a pour langage Java et CLIPS, et met en œuvre des règles de détection d'incohérence dans les résultats des appariements.

De façon générale cette fois, les algorithmes sont également décrits par leur complexité. La complexité d'un algorithme est liée à la quantité de ressources qui lui sont nécessaires pour s'exécuter, les ressources les plus couramment considérées étant le temps (que l'on mesure en nombre d'opérations élémentaires à effectuer en fonction de la taille des données d'entrée) et l'espace (que l'on mesure en quantité de mémoire à allouer).

2.3.5 Décrire comment utiliser un traitement

À l'utilisateur novice souhaitant utiliser un traitement, nous voulons fournir un mode d'emploi détaillant pas à pas les instructions à suivre. Ce mode d'emploi ne doit pas être stéréotypé mais au contraire adapté au contexte. Nous développons aussi dans cette section l'idée de spécialisation des connaissances d'utilisation. Le but est de permettre un accès progressif à la complexité des modes d'emploi, afin notamment de résoudre le dilemme classique de ces derniers : concis ils sont incomplets, complets ils sont trop volumineux.

Nous présentons ici la structure générale des modes d'emploi. L'adaptation au contexte sera développée chapitre 3.

1) Structuration des modes d'emploi

Chaque mode d'emploi est composé d'étapes. Les étapes ne se suivent pas toujours en séquence, et ne sont pas toutes à mettre au même niveau. Il n'est donc pas satisfaisant de les présenter sous forme de liste plate. C'est pourquoi nous nous dotons de structures de contrôles classiques (fig. 2.23), comme dans les process OWL-S[93], les langages de program-

[93] Les structures de contrôles utilisées par OWL-S pour l'agencement des services Web sont `for, while, split` (exécution simultanée), etc. Le détail des descripteurs de *processus* selon OWL-S peut être trouvé dans [Coa03].

mation procéduraux[94], les langages documentaires tels qu'XSD, ou comme encore de modèles de tâches.

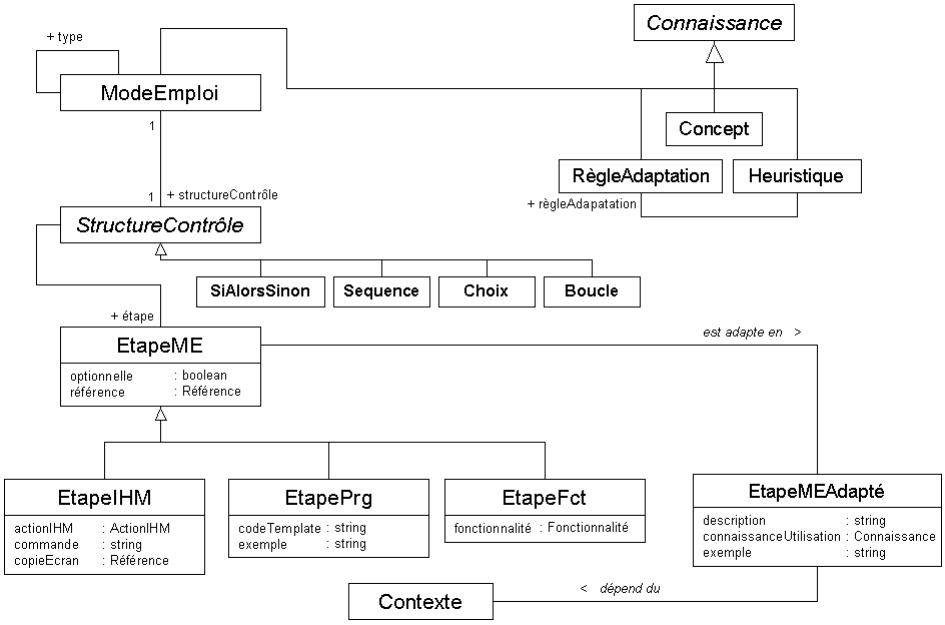


FIG. 2.23 – Mode d'emploi d'un traitement

Trois types d'étapes

Parce qu'elles se décrivent de façons différentes, il est apparu nécessaire de distinguer trois types d'étapes :

- les étapePgr qui demandent à l'utilisateur de programmer,
- les étapeIHM qui demandent simplement d'utiliser un traitement existant (la plupart du temps via l'IHM d'un logiciel, mais aussi via une ligne de commande),
- les étapeFct qui spécifient une fonctionnalité à réaliser, mais sans indiquer la façon de procéder.

Les modes d'emploi peuvent être mixtes, c'est-à-dire composés d'étapes de différents types. Les codeTemplate des étapesPrg permettent de décrire les instructions types et les modèles de code qui correspondent aux parties routinières des codes des programmes. En informatique géographique, c'est le cas par exemple des instructions qui permettent de faire des requêtes sur les bases de données géographiques, de charger des données dans les structures de données permettant de les manipuler, d'invoquer les méthodes des objets courants, etc.

Réalisation des étapes en fonction du contexte

Une même étape peut se réaliser de différentes façons suivant le contexte. Par exemple, "importer des données dans le SIG ou le programme considéré", se traduira par "appliquer tel traitement de conversion de format", puis "appliquer tel traitement de changement de projection", etc.

2) Les connaissances associées aux modes d'emploi

Au contraire des concepts et modes d'emploi destinés exclusivement à la lecture humaine (et non à l'invocation automatique), les règles d'adaptation et les heuristiques de choix de

[94]MIL, sorte de langage Pascal simplifié, cours de compilation de licence Perraut.

ces règles sont, dans notre modèle, des *connaissances opérationnalisables*. En effet elles sont destinées à être utilisées par l'application pour déterminer les réalisations des étapes. Elles devraient également être intelligibles pour l'utilisateur lambda, mais les aspects procédural et déclaratif sont difficiles à concilier. En l'état actuel de notre travail, les règles d'adaptations sont consultables et saisissables sous forme de règles *Si contexte Alors Adaptation*. C'est ce que nous allons détailler chapitre 3. Les heuristiques ne sont accessibles qu'au développeur de l'application de gestion des métadonnées.

Comme dans MASK, plusieurs types de connaissances sont représentés dans notre modèle. Nous en avons introduit quatre : Concept, ModeEmploi, Règle, et Heuristique.

Les concepts (par exemple "distance euclidienne") et les modes d'emploi (spécifiques à un traitement – p.ex. "faire une requête topologique avec Geoconcept" –, ou génériques – p.ex. "interfacer du code Java et du C"–, "améliorer un Modèle Numérique de Terrain"[95]) sont des connaissances destinées à la lecture humaine. C'est-à-dire qu'elles sont manipulables informatiquement (elles font l'objet de requêtes et sont affichées), mais leur signification n'est pas exploitée par l'application ; elles ne sont pas dotées de sémantique opérationnelle. Cela aurait été le cas si, pour reprendre nos exemples, le "concept distance" euclidienne avait été défini dans un langage formel qui aurait effectivement permis le calcul, ou si le mode d'emploi "convertir des données au format shape en MIF" avait été décrit dans un langage permettant l'invocation effective du service Web réalisant le changement de format.

3) Les actions IHM

Décrire le mode d'emploi d'un logiciel à un utilisateur – *i.e.* décrire des étapeIHM –, demande de décrire les actions à effectuer avec des interfaces homme-machine (IHM).

La figure 2.24 montre le diagramme de classes que nous avons défini pour décrire les actions IHM[96]. La figure 2.25 montre une autre façon de modéliser les actions IHM, plus classique mais qui ne convient pas à nos besoins de description plutôt que d'invocation des fonctions des logiciels.

Pour fixer les idées, disons que nous souhaitons permettre de représenter partiellement l'équivalent du code produit par les enregistreurs de macro Visual Basic des applications Microsoft Office. La description des actions utilisateurs y est formalisée de façon non ambiguë : sélection de portions de documents, activation de menu, ouverture de fenêtre, etc.

Nous pourrons ainsi atteindre l'objectif de décrire formellement les préconditions rencontrées dans les manuels des logiciels (cf. fig. 1.7 et 1.8 p. 22). Nous pourrons aussi décrire les règles d'expert correspondantes telles que *Si itemListe("Calage").état = grisé Alors explication = {commande("SaisieTablette").état = désactivé}*.

2.3.6 Évaluation d'un traitement

Un traitement peut être évalué selon plusieurs points de vue. Nous en avons retenus trois, associés respectivement à la qualité des résultats produits, aux performances et au comportement des traitements, et à leur utilisation (cf. diagramme de classe figure 2.26).

Les valeurs de certaines propriétés sont contraintes. La qualité d'un résultat du point de vue des données produites est ainsi décrite au moyen du vocabulaire contrôlé. Un traitement de généralisation de bâtiments, par exemple, sera décrit comme possédant une qualitéRésultat "bon" selon le critèreRésultat "conservation alignement".

[95] Les MNT décrivent la forme et la position de la surface du sol. Ils comportent souvent des défauts, des artefacts qu'il est possible de corriger, par exemple en évitant qu'une rivière remonte ou qu'une crête soit plate.

[96] Par souci de progressivité dans la présentation de nos diagrammes de classe – suivant en cela les principes de Bloch, un de des créateurs d'UML –, nous ne faisons apparaître qu'ici la propriété appartientIHM reliant Logiciel et ÉlémentIHM

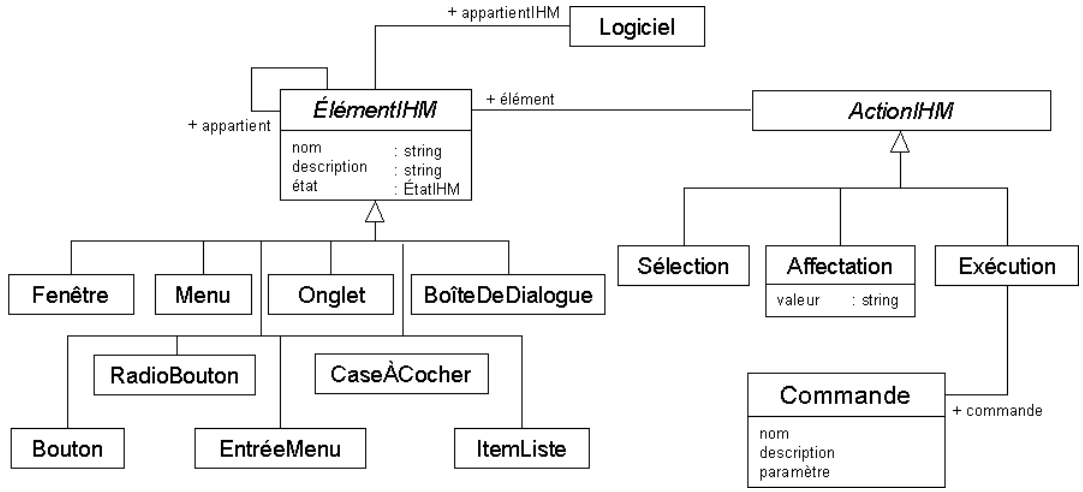


FIG. 2.24 – IHM d'un Logiciel

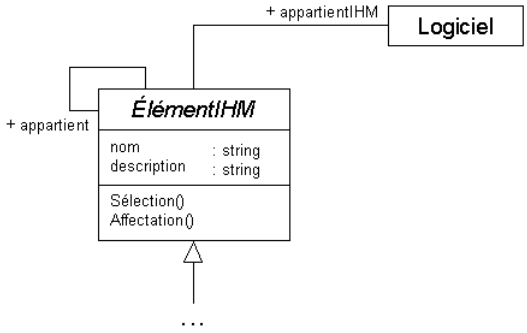


FIG. 2.25 – IHM d'un Logiciel (modélisation pour la programmation orienté objet, nous ne l'adoptons pas.)

Cet effort de formalisation a notamment pour but de rendre possible les comparaisons. Mais une telle exploitation des métadonnées semble difficile à réaliser. D'une part, il faudrait s'assurer au moment de l'acquisition que les auteurs font la même interprétation des termes de description. D'autre part, et surtout, les critères d'évaluation sont relatifs à des critères d'appréciation que notre modèle ne permet pas de décrire de façon rigoureuse. Par exemple, la stabilité de comportement d'un programme de visualisation de données géographiques pourra être jugée bonne dans le contexte d'une application d'un laboratoire de recherche et mauvaise dans le contexte d'un système de navigation embarqué.

Nous aurions pu enrichir notre modèle pour atteindre le niveau de finesse de description nécessaire à l'expression de l'exemple ci-dessus. Nous n'avons pas fait ce choix. Le prix à payer aurait été une trop grande complexité du modèle, non justifiée par les besoins identifiés. Or, les expériences d'acquisitions que nous avons menées tendent à montrer que pour la plupart des auteurs de métadonnées le nombre d'éléments de description retenus est déjà élevé. Nous prenons cet enseignement en compte.

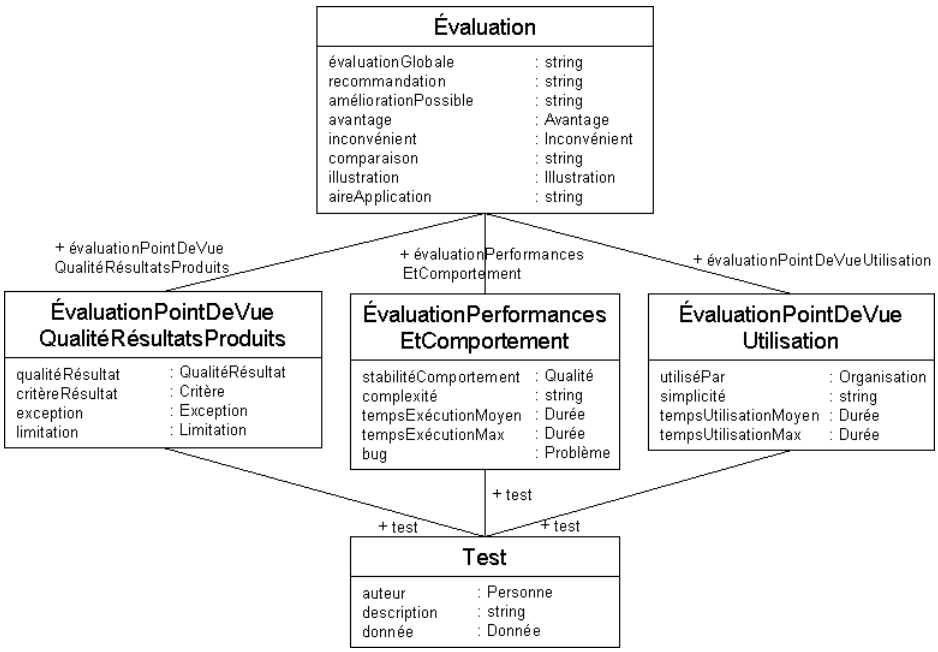


FIG. 2.26 – Évaluation d'un traitement

Notre souci pragmatique de ne pas oublier l'utilisateur se traduit également par l'introduction d'éléments de descriptions comme le *temps moyen d'utilisation* d'un traitement. Cette considération se démarque des descriptions rencontrées dans l'état de l'art où seul était décrit le *temps moyen d'exécution* machine des traitements. Peut-être est-ce là un héritage des temps où les puissances de calcul étaient encore faibles. Bien sûr le temps nécessaire pour l'utilisation d'un traitement dépend des compétences de l'utilisateur. C'est pourquoi l'estimation du tempsUtilisationMoyen peut s'accompagner d'une description du niveau d'expertise correspondant.

Enfin, la description des tests effectués vient compléter utilement les éléments qui composent la facette Évaluation des traitements.

2.3.7 Classes complémentaires

Les familles de traitements

Nous avons précédemment soulevé le problème de la représentation des connaissances relatives à des familles de RessourceTraitement et nous avons avancé l'idée de réifier ces dernières en

FamilleTraitement.

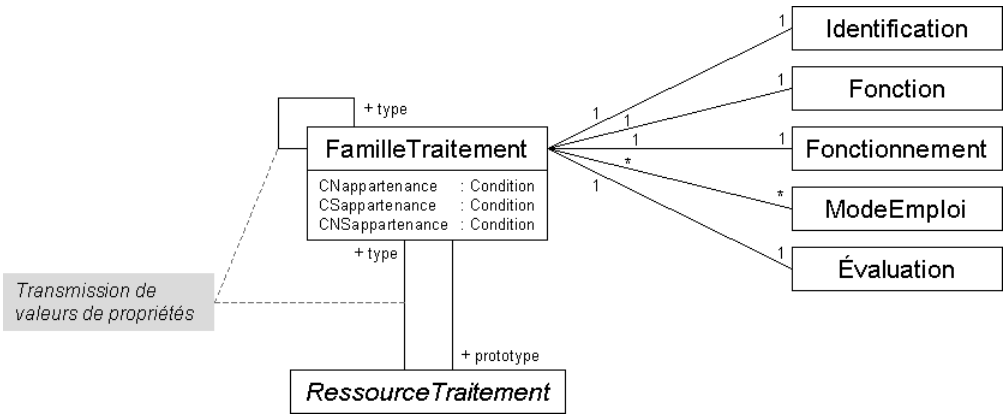


FIG. 2.27 – Famille de traitements

Les familles de traitements se spécialisent. Une taxinomie est constituée. Elle n'est pas figée, de nouvelles familles peuvent apparaîtrent au fur et à mesure de l'acquisition des connaissances d'expert. Pour cette raison, les instances de FamilleTraitement ne font pas partie du modèle mais de la base de métadonnées. Un nouveau projet pour la production d'une nouvelle série de carte ou pour la prise en compte de photos aériennes de résolution décimétrique voit le jour à l'IGN ? Aussitôt de nouvelles FamilleTraitement sont créées. Elles servent de réceptacles aux informations communes concernant le lieu de développement, le domaine de fonctionnalité, les types de données, les langages et les plateformes de développement utilisés, les environnements logiciel et matériel associés, etc.

Les attributs type de la figure 2.27 portent des relations de subsumption où sont héritées les valeurs de propriétés. Il s'agit là d'un type d'héritage différent de l'héritage classique de propriétés. Par exemple, Logiciel est subsumé par TraitementCodé. Il hérite donc des propriétés concernant les cinq facettes. Il s'agit là de la relation d'héritage de propriétés classique. En revanche, dans la relation de subsumption qui unit la FamilleTraitement SIG_ESRI à la FamilleTraitement SIG, ce sont des valeurs de propriétés qui sont héritées. À ce propos nous préférons le terme de *famille* à celui de *catégorie*, car il suggère davantage la notion de transmission de valeurs de propriétés.

Nous sommes convaincus que les FamilleTraitement sont des représentations propices à l'extraction des connaissances tacites de l'expert. Les simples règles le sont peut-être moins. Attendre de l'expert qu'il exprime spontanément des règles tels que *si lieu de développement = cogit alors type_donnée = vecteur* n'est pas une méthode optimale. Cela suppose qu'il réfléchisse à la fois à l'expression de la prémisse et à celle de la conclusion. Au contraire, si nous lui demandons dans un premier temps de songer aux familles pertinentes de traitements – *i.e.* en fait à la partie prémisse des règles, caractériser dans un second temps lesdites familles semble plus aisé. Cela a notamment l'avantage de regrouper les règles au sein des définitions de familles, et de mettre en place un mécanisme d'héritage.

On peut trouver une certaine similitude entre l'opposition des deux modes de représentation *règles isolées / caractérisation de familles* avec l'opposition *programmation procédurale / programmation orientée objet*. Dans les deux cas, il s'agit d'encapsuler au sein d'une même structure des connaissances de même thématique. Un ensemble de structure de données et de fonctions dispersées est plus difficile à appréhender qu'un ensemble d'objets ; un ensemble de règles dispersées est plus difficile à appréhender qu'un ensemble de concepts définis chacun par des conditions d'appartenance ("si un logiciel réalise les fonctionnalités 5A alors c'est un SIG" *versus* "Une CNS définissant le concept SIG est : un logiciel réalisant les fonctionnalités 5A").

Une FamilleTraitement possède un prototype, *i.e.* un élément typique qui peut être exhibé à l'utilisateur pour une meilleure compréhension (cf. p. 33). Haton *et al.* discutent de l'intérêt de l'utilisation des prototypes dans les SBC en invoquant certains résultats d'études en psychologie ([HBF+91], pp. 314-316). Ils citent notamment E. Rosh, selon qui le codage des catégories de concepts le plus économique sur le plan cognitif repose sur les prototypes. Les listes énumérant les individus membres d'une catégorie ou l'énonciation des conditions nécessaires et suffisantes d'appartenance seraient moins parlantes aux humains [Ros75].

Cependant, les représentations adaptées aux humains ne sont pas toujours adaptées aux machines (et réciproquement). C'est pourquoi, pour décrire les familles de traitements, nous proposons, en complément des prototypes, la définition de conditions nécessaires et/ou suffisantes d'appartenance. Nous verrons par la suite que ces dernières se prêtent mieux à notre objectif d'opérationnalisation des connaissances d'expert.

Classes annexes

Les diagrammes de classes exposés précédemment font référence à de nombreuses classes annexes. Nous ne les détaillons pas toutes. Certaines néanmoins sont définies figure 2.28. Notons que la classe Comparaison est inspirée des principes différentiels proposés par B. Bachimont pour la conception des ontologies [Bac00]. Nous y reviendrons au chapitre 6 (p. 223).

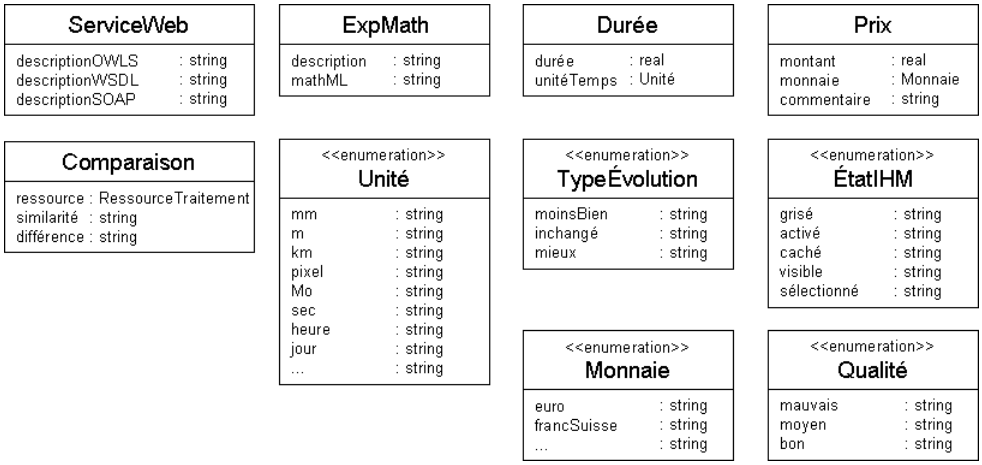


Fig. 2.28 – Classes annexes

2.4 Conclusion

Nous avons proposé un état de l'art des modèles de métadonnées des traitements informatiques, notamment de ceux du domaine géographique. De façon un peu plus large, nous avons considéré divers langages, méthodes, travaux et outils utiles à la description des traitements, bien que ne proposant pas tous à proprement parler de modèle de métadonnées.

De ce travail de prospection forcément partiel, nous avons tiré un bref bilan (sous-section 2.1.6). Il en ressort qu'aucun des modèles de métadonnées existants – parmi les plus connus du moins – ne répond totalement à nos attentes. L'ontologie OWL-S nous apparaît toutefois intéressante à plusieurs égards. Bien que son usage soit circonscrit à la description des services Web et non des traitements informatiques en général, les éléments de description qu'elle propose et les principes de représentation des connaissances sur lesquels elle repose nous intéressent fort. OWL-S est conçu pour supporter des requêtes similaires à celles que nous avons identifiées dans notre analyse des besoins fig. 1.5, p. 20. OWL-S s'inscrit dans le projet du Web sémantique. C'est naturellement que dans la suite de ce mémoire nous nous orienterons vers le domaine dudit Web sémantique.

Pour ce qui est du domaine géographique, nous nous inspirons en particulier du modèle de description de traitement de généralisation cartographique défini dans le cadre du projet Agent.

Une fois l'état de l'art effectué, nous avons défini notre modèle conceptuel de métadonnées au moyen du formalisme des diagrammes de classes UML. Auparavant nous avons discuté des principes de modélisation orientée objet sur lesquels nous nous basons.

La spécificité géographique de notre contexte transparaît peu dans notre modèle. Ce dernier est donc générique aux traitements informatiques en général[97]. Cela s'explique par notre choix de faire partiellement de notre modèle un méta-modèle. Ainsi, par exemple, la hiérarchie des fonctionnalités géographiques ne figure pas dans notre modèle. Elle figure dans la base de métadonnées qui instancie notre modèle.

En l'état actuel, notre modèle permet de décrire les traitements mais il ne permet de capturer qu'une partie des connaissances de l'expert. Si nous disposons donc d'une bonne base pour construire un SI, nous ne sommes pas encore en mesure de construire un SBC. Il nous reste notamment à modéliser les règles d'expert qui permettront d'adapter les modes d'emploi au contexte d'utilisation.

[97] Si les classes représentant les types de traitements n'appartiennent qu'au domaine informatique, en revanche les classes qui permettent la description des différentes facettes pourraient trouver une traduction dans des domaines variés. Un domaine auquel s'applique étonnamment bien notre modèle est par exemple celui de la médecine. Un traitement médical peut en effet se décrire ainsi : Identification (nom, date de création, auteur, version, lieu de développement, référence), Fonction (entrée : le patient, paramètre : posologie, effets curatifs et indésirables, problèmes résolus, préconditions : incompatibilité entre traitements, contre-indications), Fonctionnement (principes actifs, étapes), ModeEmploi (contexte : âge, allergies, antécédents familiaux, etc.), Évaluation (efficacité, temps d'exécution, avantage, inconvénients, etc.).

Chapitre 3

Vers une représentation opérationnelle des connaissances de l'expert

Au chapitre précédent, nous avons défini un modèle de métadonnées "documentaire". Nous entendons par-là que sa forme, selon nous, le destine à servir de base à un système d'information documentaire. C'est-à-dire à un système qui, s'il offre aux utilisateurs diverses fonctionnalités plus ou moins sophistiquées pour rechercher, consulter, créer, modifier des métadonnées, voire pour effectuer des procédures spécifiques, ne contient pas de connaissances permettant de dériver de l'information non explicitement présente dans la base de métadonnées. Cette caractéristique interdit la réponse aux requêtes ER 1 et ER 3 illustrées pages 20 et 21.

La solution pour répondre à ce type de requêtes réside, d'une part, dans le recours aux ontologies formelles et aux mécanismes d'inférences associés, et d'autre part, dans la mise en œuvre d'un système à base de règles permettant notamment l'adaptation des modes d'emploi des traitements au contexte d'utilisation. Ontologies formelles et règles sont deux formes complémentaires de représentation opérationnelle des connaissances d'expert.

Dans ce chapitre, nous détaillons une sélection d'exemples significatifs de cas où le raisonnement de l'expert intervient. Nous en déduisons le niveau nécessaire d'expressivité des langages de définition d'ontologie et de règles à mettre en œuvre. Nous achevons alors de définir notre modèle conceptuel de métadonnées en y introduisant les règles d'expert, le contexte de l'utilisateur et le contexte requis par les traitements. Comme nous le verrons à travers différents exemples, ces nouvelles classes nous permettent de calculer les instances de la classe ModeEmploiAdapté définie au chapitre 2.

3.1 Le raisonnement de l'expert

3.1.1 Le besoin de raisonner sur les métadonnées des traitements

Qu'est-ce que l'expert en traitements géographiques ? C'est celui qui possède et sait mettre en œuvre les connaissances liées aux traitements. Nous nous donnons pour objectif de simuler une partie de son raisonnement[1]. En effet, toute l'information sur les traitements ne peut être stockée explicitement dans la base de métadonnées, tous les cas d'utilisations associés aux contextes possibles des divers utilisateurs ne peuvent être prévus à l'avance. En revanche, l'information recherchée peut être dérivée de celles explicitement présentes. C'est précisément ce que sait faire l'expert, grâce à ses connaissances.

[1]Le terme de "raisonnement" fait l'objet de beaucoup de tentatives de définition en IA et en sciences cognitives. Dans ce mémoire, nous l'employons de façon générale pour désigner une suite d'inférences, qui en constituent les pas élémentaires.

Notre ambition est limitée, notre cadre de travail restreint ; nous sommes loin d'espérer construire un système d'aide assez puissant pour répondre aux requêtes des utilisateurs comme le ferait un expert humain, atteignant ainsi le Graal de l'IA que constitue le succès au test de Turing. Nous envisageons en fait trois cas typiques où un raisonnement est mis en œuvre.

Dans le premier cas, l'utilisateur cherche un traitement ; il exprime son besoin avec des mots-clés qui ne correspondent pas exactement à ceux des descriptions enregistrées dans la base de métadonnées. Le raisonnement permet de faire aboutir la recherche.

Dans le second cas, l'utilisateur a besoin d'un mode d'emploi adapté à son contexte d'utilisation, différent du contexte requis par le traitement. Le raisonnement consiste alors à identifier les différences entre les contextes – à poser le diagnostic en quelque sorte –, puis à indiquer à l'utilisateur les instructions à suivre.

Dans le troisième cas enfin, il s'agit d'enrichir la base de métadonnées par l'ajout de nouveaux faits inférés. Les mécanismes mis en œuvre sont les mêmes que lors d'une recherche de traitement, à ceci près que les inférences ne sont pas limitées au contexte d'une requête particulière.

Le système d'aide à l'utilisateur que nous voulons construire manipule des symboles. Or l'expert humain, pour raisonner, manipule des concepts. Le passage du niveau symbolique au niveau conceptuel sémantique est une des principales problématiques de l'IA.

Dans le modèle de métadonnées que nous avons défini au chapitre 2 figurent des éléments de description destinés à indiquer le sens des concepts aux humains au moyen de texte en langue naturelle. Cette forme les rend inaptes à servir notre objectif d'une modélisation opérationnelle des connaissances d'expert. Mais le modèle de métadonnées comporte également des éléments de description dont les types sont des identifiants de ressources, des nombres ou des booléens. Ces éléments représentent des relations auxquelles une sémantique peut être assignée. Par exemple la propriété appartient_à de la classe EnsembleTraitementsCodés est transitive. La chaîne de caractères appartient_à (niveau symbolique) est interprétée comme désignant la relation "appartient à" (niveau "sémantique").

Traduire notre modèle et ses instances sous forme d'ontologie formelle va nous permettre d'opérationnaliser une partie des connaissances d'expert, notamment une partie de celles dont nous avions identifié le besoin au chapitre 1 (tab. 1.6, p. 28, comportant les relations généralisation/spécialisation et partie/tout).

Une partie des connaissances sera ainsi représentée, mais une partie seulement. L'expressivité des ontologies est en effet limitée. Or il appert au vu des différents exemples de raisonnement qui vont suivre que les connaissances d'expert doivent également s'exprimer sous forme de règles. Dans ce but, nous allons nous donner un langage de logique.

Le sujet de ce chapitre, la simulation du raisonnement de l'expert, ne doit pas occulter notre objectif plus large de représentation des connaissances. Certaines sont opérationnalisables, d'autres pas. Notre modèle de métadonnées vise à permettre le recueil des deux catégories. L'expressivité des langages que nous utilisons sera parfois insuffisante[2] ; l'expert exprimera alors ces connaissances en langue naturelle. Notre démarche relève donc à la fois du domaine des SBC où les connaissances sont opérationnalisées et de celui de systèmes de gestion de connaissances comme MASK où les connaissances sont "simplement" recueillies.

Opérationnalisables ou non, nous devons permettre l'acquisition des règles d'expert. C'est pourquoi, en section 3.3, nous enrichissons notre modèle de métadonnées de nouvelles classes. Fidèle à notre démarche ascendante, nous avons tenté de partir d'exemples de règles exprimées par des experts lors d'entretiens. Nous avons rencontré plusieurs experts en généralisation et en

[2]Nous allons préciser nos hypothèses de travail concernant la simulation du raisonnement, et donner quelques exemples des renoncements qu'elles impliquent.

traitement d'image. Il ressort des entretiens menés que les obstacles potentiels à la représentation des règles dépendent principalement de la richesse du modèle. Nous reviendrons sur cette question et plus largement sur les problèmes d'acquisition au chapitre 6.

Nous avons également cherché des documents répertoriant des règles d'experts sur les traitements. L'un d'eux est un rapport de stage réalisé par M. Dadou et consacré à la réalisation d'une interface de saisie de connaissances d'expert pour l'automatisation de la généralisation cartographique [Dad05]. Les règles dont il est question décident, par exemple, du traitement à appliquer pour généraliser des groupes de bâtiments en fonction de leur taille et du nombre de leurs voisins (une petite maison en zone urbaine sera supprimée, une grande isolée en zone rurale sera simplifiée).

Dans le cadre de traitements d'appariement de base de données géographiques, des règles sont aussi parfois utilisées. Mais il s'agit de règles sur les données et non sur les traitements (par exemple, les spécifications de la base de données Géoroute indiquent que si un rond point possède un diamètre de plus de 30m, alors il doit être représenté par un objet de géométrie surfacique) [She05].

En fin de compte, c'est principalement en partant des besoins d'information des utilisateurs et en nous demandant quels raisonnements sont nécessaires pour y répondre que nous avons dressé la liste des règles à représenter.

3.1.2 Exemples de raisonnements de l'expert

Examinons à travers sept exemples les types de raisonnement que peut mener l'expert. Nous nommons ces exemples "ER" (Exemple de Raisonnement). Une partie d'entre eux nous serviront au chapitre 5 à illustrer concrètement la mise en œuvre de notre système. Le but n'est pas ici de décrire comment raisonne l'expert humain – à ce sujet, nous ne pouvons émettre que des hypothèses – mais de commencer à faire apparaître les moyens que nous allons devoir mettre en œuvre informatiquement. Pour cette raison, les commentaires des exemples font référence à la notion d'ontologie ; nous la présentons à la section suivante.

ER 1 : mise en correspondance entre requête et description de traitement – subsomption, méréologie et condition suffisante d'appartenance à une classe

L'exemple de raisonnement ER 1 a été donné p. 20. On y voit l'expert effectuer plusieurs inférences pour établir que le traitement "Buffer.java" répond à la requête de l'utilisateur. Par hypothèse, on ne considère pas ici les problèmes potentiellement liés à l'expression de la requête. En effet, on suppose que l'utilisateur définit les quatre critères de recherche via un formulaire qui le contraint à sélectionner des termes parmi ceux déjà référencés dans la base de métadonnées. On pourrait aussi supposer que l'utilisateur a été autorisé à faire usage de la langue naturelle, mais qu'un programme de TALN – étranger au cadre de notre étude – a permis la normalisation de la requête.

Des quatre critères de recherche définis, aucun ne correspond directement à la description de "Buffer.java". L'expert aboutit néanmoins à la bonne réponse car :

- Ses connaissances lui indiquent qu'une condition suffisante pour qu'un traitement appartienne à la catégorie "géographique" est qu'il soit développé au COGIT (il nous paraît acceptable de faire, ici, cette hypothèse). Le premier critère "catégorie" est donc satisfait.
- Il sait que le COGIT appartient au service de la recherche de l'IGN. Or le service de la recherche appartient à l'IGN. Donc que le COGIT appartient à l'IGN (appartient est une relation transitive). D'autre part, il existe une règle qui dit que si un traitement est développé dans une organisation O1 et que cette organisation O1 appartient à une organisation O2, alors le traitement est aussi développé dans l'organisation O2.
- Il sait que les types de données "TD_VecteurLigne" et "TD_VecteurSurface" sont deux spécialisations de "TD_Vecteur". Par ailleurs, il sait quelles interprétations faire de la propriété type, selon qu'elle caractérise les entrées ou les sorties des traitements.

Commentaires sur l'exemple ER 1

La première condition pour effectuer des raisonnements tels que ceux mis en œuvre dans l'exemple ER 1 est de désigner les ressources de façon non ambiguë, donc via des identifiants. C'est pourquoi, dans le cadre de notre application, l'expression de la requête par l'utilisateur d'une part et les descriptions de traitements d'autre part, doivent être contraintes, respectivement par l'interface d'interrogation et par le modèle de métadonnées.

Les ressources dont on manie les identifiants sont des traitements, des types de données, des concepts, etc. Elles ont été exposées dans les divers diagrammes de classes au chapitre précédent. Ce qu'il faut également, c'est représenter formellement les relations qui unissent les ressources, car c'est sur leur exploitation que reposent les inférences. Nous verrons chapitre 4

que les ontologies formelles ont vocation à représenter la plupart des relations impliquées dans l'exemple ER 1 ; d'autres devront se traduire dans des règles de la logique des prédicats.

ER 2 : mise en correspondance entre requête et description de traitement – subsomption et condition nécessaire d'appartenance à une classe

L'utilisateur cherche tous les logiciels capables d'afficher les données géographiques en général. Pour cet exemple, on suppose que la description du logiciel "Arcview 8" ne comporte qu'une seule indication, à savoir que sa famille est celle des SIG ESRI.

L'expert va déduire qu'Arcview 8 répond à la requête de l'utilisateur car :

- La famille de logiciel "SIG ESRI" est une spécialisation de la famille "SIG".
- Une condition nécessaire d'un SIG est qu'il réalise cinq fonctionnalités, parmi lesquelles figure "afficher des données géographiques" (cf. tableau 1.1 p. 11).

Commentaires sur l'exemple ER 2

Le type de raisonnement mis en œuvre présente des similarités avec celui de l'exemple ER 1. Comme dans ce dernier, il suppose l'existence de classifications hiérarchiques, en l'occurrence, ici, de familles de logiciels. Si l'on considère ces familles comme les classes d'une ontologie, alors pour chacune d'entres elles il peut être défini des conditions nécessaires et/ou suffisantes d'appartenance (les "axiomes de classes"). En sus de la simple exploitation de la relation de subsomption (SIG ESRI spécialise SIG), le raisonnement de l'exemple ER 2 se base sur la définition d'une condition nécessaire d'appartenance pour déduire qu'Arcview 8 est bien une réponse à la recherche de l'utilisateur. Dans l'exemple ER 1, c'était une condition suffisante d'appartenance qui était utilisée. Cela montre que les connaissances d'expert à représenter ne se limitent pas à la simple spécification d'une hiérarchie de concepts (ce que Fürst et Trichet, citant [GFLC03], appellent "ontologie légère" [FT05]). Il faut aussi décrire des règles qui apparaissent soit dans la définition même des concepts (conditions d'appartenance à une classe des ontologies lourdes (*ibid.*)), soit en tant que formalisation extérieure de connaissance comme dans les systèmes experts à base de règles.

ER 2 suite : recherche de ressources – classification des problèmes de lisibilité

Dans l'exemple ER 2 l'utilisateur cherchait un logiciel pour afficher des données géographiques ; l'expert lui a suggéré Arcview. Maintenant l'utilisateur veut cartographier son jeu de données en choisissant une autre symbolisation que celle proposée par défaut. Ceci effectué, l'utilisateur n'est pas satisfait : il trouve que ses données sont cartographiées de façon peu lisible. Il demande donc à l'expert de lui indiquer les traitements qui peuvent corriger ce problème.

Le but est ici de réussir à conseiller à l'utilisateur un traitement d'amplification des virages des routes. Un raisonnement est nécessaire car on suppose que dans la base de métadonnées :

- La description de la fonctionnalité "amplification" mentionne uniquement comme problème résolu : "problème d'empâtement".
- Le problème d'empâtement et le problème de lisibilité sont indexés mais il n'est pas indiqué que le premier est une sorte particulière du second. C'est cette information manquante que l'expert va déduire.
- La fonctionnalité "généralisation" est définie comme ayant pour condition nécessaire : résout un "problème de lisibilité" ;
- La fonctionnalité "caricature" est une sorte de "généralisation" ;
- La fonctionnalité "amplification" est une sorte de "caricature" ;
- La fonctionnalité "amplification" résout le problème d'empâtement (et seulement celui-ci).

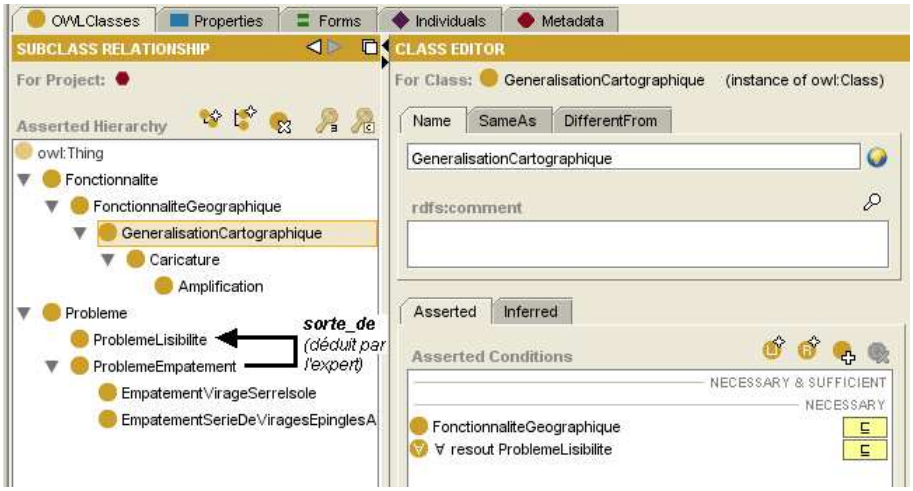


Fig. 3.1 – Visualisation des concepts de l'exemple ER 2 suite avec l'éditeur d'ontologie Protégé 3.1

L'expert déduit que l'empâtement est un problème de lisibilité. Il déduit que les traitements d'amplification répondent à la requête de l'utilisateur.

Commentaires sur l'exemple ER 2 suite

Dans l'exemple ci-dessus les problèmes et les fonctionnalités sont des concepts définis dans une ontologie par des conditions nécessaires et/ou suffisantes. En anticipant un peu sur la présentation de la façon dont nous allons opérationnaliser les ontologies, nous pouvons remarquer que, dans cet exemple, les valeurs de propriétés sont des concepts ("amplification" résout

"problème d'empâtement", ce dernier étant un concept pouvant être spécialisé). Nous verrons que pour pouvoir raisonner certains systèmes d'inférences demandent que les valeurs de propriétés soient non pas des concepts mais des instances de concepts.

ER 3 : adaptation de mode d'emploi – précondition sur la propriété d'une entrée

Reprenons l'exemple ER 3 donné p. 21. Un utilisateur désire visualiser avec FreeWRL un MNT au format VRML comprenant 5000 objets. Son ordinateur est doté d'une mémoire vive de 128 Mo, son système d'exploitation est Linux Suse.

L'expert, sollicité par l'utilisateur, ***pose à ce dernier des questions pertinentes*** sur son contexte d'utilisation. Ces informations obtenues, l'expert déduit que, compte tenu du contexte requis par FreeWRL, un problème d'insuffisance de mémoire vive se pose. Ce diagnostic étant posé, l'expert propose des solutions résolvant le problème, ou le résolvant moyennant quelques écarts potentiellement admissibles vis-à-vis du contexte initial de l'utilisateur. Enfin, l'expert livre quelques conseils relatifs à la cause du problème :

- L'expert demande si l'utilisateur travaille au laboratoire Cogit. Si oui, il lui indique les instructions à suivre pour utiliser FreeWRL sur une machine distante de mémoire vive suffisante via un ClientCitrix.
- L'expert suggère d'installer une barrette de mémoire vive supplémentaire sur la machine.
- L'expert sélectionne les autres logiciels de visualisation VRML pouvant être utilisés, pour peu que leurs fonctionnalités soient supérieures ou égales à celles de FreeWRL, et qu'ils soient par ailleurs compatibles avec le contexte spécifié par l'utilisateur (système d'exploitation, licence, etc.). Par exemple : CosmoPlayer.
- L'expert sélectionne également les RessourceTraitement répondant aux critères moyennant une adaptation. Par exemple, utiliser le programme de visualisation 3D développé au Cogit demande une étape supplémentaire de conversion de format de donnée (VRML vers les objet de l'API Java3D).
- Enfin, l'expert indique à l'utilisateur que d'une façon générale les problèmes de mémoire vive insuffisante peuvent être causés par un trop grand nombre d'applications ouvertes.

Commentaires sur l'exemple ER 3

Dans le scénario ci-dessus l'expert effectue trois raisonnements. Le premier sert à poser à l'utilisateur les questions ***pertinentes*** relativement au contexte. Le deuxième raisonnement sert à diagnostiquer le problème qui se pose. Pour cela l'expert a dû appliquer une règle générale concernant la mémoire vive des traitements au cas présent particulier. Une fois le diagnostic posé, l'expert mène un troisième raisonnement pour proposer des solutions. Certaines requièrent l'obtention de nouvelles précisions sur le contexte de l'utilisateur ; l'établissement d'un dialogue peut être nécessaire. Par ailleurs les conseils peuvent être ordonnés, cela suppose l'utilisation de méta-connaissances heuristiques.

ER 4 : adaptation de mode d'emploi – incompatibilité de traitements

Dans le cadre du développement d'un service Web de données géographiques au format GML, un utilisateur veut utiliser la classe Java ReqXML qui fournit des méthodes pour l'accès aux documents XML[3]. Comme il rencontre un problème empêchant la réalisation de son besoin, l'utilisateur demande à l'expert les instructions à suivre. Pour cela il indique :

– qu'il rencontre le problème : `"cannot find symbol : class XPathFactory"` à la compilation ;
– qu'il utilise le moteur de servlet Tomcat 4.2.

L'expert possède les connaissances suivantes :

– Tomcat requiert le JDK,
– le package XPath n'est disponible qu'à partir de la version 1.5 du JDK,
– les Tomcat de version inférieure à 5 sont incompatibles avec le JDK 1.5.

L'expert déduit que l'utilisateur doit installer le JDK version 1.5 et Tomcat version 5 ; il indique les instructions à suivre (copie de fichiers, affectation de variables d'environnement, inclusion de directives d'import, etc.).

Commentaires sur l'exemple ER 4

Outre le caractère un peu complexe du raisonnement effectué ici, on remarquera surtout qu'il met en jeu des connaissances générales non pas sur un traitement particulier, mais sur une famille de logiciels. Bien qu'on puisse éventuellement envisager de représenter l'assertion "Tomcat requiert le JDK" par une règle, il est plus élégant de recueillir cette connaissance d'expert au moyen d'une classe "méta" : c'est le rôle de la classe FamilleTraitement.

ER 5 : adaptation de mode d'emploi – sélection de traitements réalisant un ensemble de fonctionnalités

L'utilisateur a besoin de calculer le nombre de stades situés à moins de 2km du métro Saint-Mandé. Il demande à l'expert comment faire, précisant que ses données sont au format *shape* et qu'il ne veut pas programmer.

L'expert indique à l'utilisateur que le mode d'emploi "calcul de la population proche d'un fleuve" décrit dans la base de métadonnées correspond à son besoin. En effet ce mode d'emploi sert d'exemple type (de prototype pour reprendre la terminologie adoptée) au mode d'emploi générique "calcul du nombre d'éléments à proximité d'une zone géographique".

L'expert indique à l'utilisateur que le SIG Arcview correspond à son contexte d'utilisation et possède les fonctionnalités requises.

L'expert traduit les ÉtapeFct abstraites du mode d'emploi prototype (importer les différents thèmes de données dans un SIG, faire correspondre les projections respectives, effectuer un buffer autour du fleuve et effectuer une requête topologique à partir de la zone buffer obtenue) en ÉtapeIHM correspondantes d'Arcview.

[3]En effet le format GML de description d'objets géographiques possède une syntaxe XML.

Commentaires sur l'exemple ER 5

Dans cet exemple, l'expert ne construit pas le mode d'emploi prototype. Il se contente d'adapter celui qui existe déjà au SIG Arcview. La construction du mode d'emploi prototype doit être réalisée, en amont, par un expert.

ER 6 : adaptation de mode d'emploi – déductions au sujet des connaissances de l'utilisateur

L'utilisateur veut visualiser son MNT au format "point-virgule". Les solutions exposées figure 1.9 (p. 26) reposent sur différents langages de programmation : VisualBasic, C++ ou Java.

L'expert va aiguiller l'utilisateur vers la solution correspondant à ses connaissances. On suppose ici que l'utilisateur n'indique pas quels langages il connaît. En revanche, il indique qu'il utilise la plateforme GeOxygene.

L'expert en déduit que l'utilisateur connaît le langage de ladite plateforme, Java en l'occurrence. La solution prescrite est donc celle qui utilise l'API Java 3D.

Accessoirement, l'expert identifie l'utilisateur comme un programmeur Java, donc connaissant, par exemple, les principes de l'orienté objet.

Commentaires sur l'exemple ER 6

Le raisonnement de cet exemple suppose qu'une plateforme ne peut être implémentée que dans un langage. Cela n'est pas totalement exact car, en l'occurrence, du code C est parfois également utilisé (pour les triangulations de Delaunay qui sont des traitements appliqués aux données 3D). Toutefois, on peut considérer ici qu'une plateforme n'a bien qu'un seul langage principal, que l'utilisateur connaît forcément. La propriété a_pour_langage des plateformes est donc une *propriété fonctionnelle*.

ER 7 : adaptation de mode d'emploi – requête topologique

L'utilisateur veut un traitement permettant de trouver les rues qui intersectent une rue donnée. Un calcul d'intersection est donc nécessaire, à moins que l'information ne figure déjà dans les données de l'utilisateur. L'expert demande à ce dernier d'où proviennent ses données. L'utilisateur répond "de la BD Carto". L'expert en déduit que la topologie est peut-être déjà explicitement présente. L'expert sait que selon le format des données, la topologie est explicitement représentée ou non. L'expert demande donc dans quel format sont les données. L'utilisateur répond "ArcInfo". L'expert déduit que la topologie n'est pas explicite ; il conseille à l'utilisateur d'utiliser la fonction logicielle de ArcGIS "requête topologique".

Commentaires sur l'exemple ER 7

Dans ce scénario, un réel dialogue a lieu. Son déroulement doit être en partie pré-mémorisé : les règles doivent prévoir les cas de figure où l'origine des données n'est pas renseignée et où il faut demander spécifiquement cette information à l'utilisateur. Par ailleurs, cet exemple montre la nécessité de disposer de métadonnées sur les bases de données géographiques qui puissent être mises en relations avec les pré-requis des traitements.

3.1.3 Quelques travaux relatifs aux systèmes adaptatifs

Dans les exemples que nous venons d'exposer, les raisonnements ont essentiellement pour but l'adaptation des modes d'emploi des traitements au contexte de l'utilisateur. Avant de présenter comment nous proposons d'atteindre ce but, évoquons rapidement quelques travaux relatifs aux systèmes adaptatifs. L'intérêt est, en particulier, de souligner la nécessaire dérivation d'information non explicitement présente, le fait que pour cela les mécanismes d'inférences associés aux ontologies sont utilisés – nous en discutons section 3.2 –, le fait enfin que l'adaptation d'un système ou d'un document nécessite une modélisation du contexte de l'utilisateur – notre proposition à ce sujet se trouve en 3.3.

La notion de *système adaptatif* se rencontre dans beaucoup de domaines. Pour une tâche donnée, l'environnement, le contexte et le profil de l'utilisateur varient. De plus en plus, grâce à l'informatique, les systèmes matériels ou logiciels possèdent la faculté de s'adapter. Parmi les domaines concernés, deux sont particulièrement proches de notre problématique. Il s'agit du Web sémantique[4] et de l'enseignement intelligemment assisté par ordinateur (EIAO).

Dans leur article "Adaptation et personnalisation dans le Web sémantique", S. Garlatti et Y. Prié discutent de différents moyens pour adapter la recherche et le contenu des documents du Web à l'utilisateur [GP03]. Ils suggèrent notamment de décrire et prendre en compte le profil de l'utilisateur : ses connaissances, ses préférences et ses objectifs. Le but peut être de diminuer le bruit lors de la recherche de documents existants ; il peut être également de créer dynamiquement des documents sur mesure. Ces buts sont aussi les nôtres. Pour le second, S. Garlatti et Y. Prié emploient le terme de *document virtuel personnalisable* (DVP). L'exemple de l'adaptation de la forme d'un document à la taille de l'écran ou au débit de la connexion Internet de l'utilisateur s'applique bien au domaine géographique, les nouveaux usages de consultation de cartes sur PDA[5] et de systèmes d'aide à la navigation le montrent.

Les DVP sont virtuels au sens où, au moment de la réception des requêtes, ils n'existent pas sous la forme que va recevoir l'utilisateur. Cette proposition de définition des DVP mérite une précision. Certains DVP ne sont que le résultat de l'agrégation de documents, résultat pouvant être obtenu par de simples opérations de sélection sur une base de données. Typiquement, les serveurs de pages Web dynamiques couplés à une base de données répondent à la définition de DVP. En revanche, la construction de certains DVP nécessite réellement la mise en œuvre des connaissances dans le cadre de raisonnements. Ainsi, pour montrer l'intérêt du langage d'ontologie du Web sémantique OWL, ses auteurs donnent l'exemple non trivial d'un utilisateur à la recherche d'un vin adapté à ses goûts.

Cette idée d'utiliser les ontologies pour créer des DVP adaptés à l'utilisateur a été mise en œuvre dans le monde de l'entreprise, notamment, par Fortier et Kassel. Leur projet est l'élaboration d'un système adaptatif d'interrogation de mémoire d'entreprise, ce qu'ils appellent le *Web Sémantique d'Organisation* (WSO) [FK04].

Le domaine des EIAO fournit d'autres exemples de systèmes adaptatifs. Certains scénarii proposés sont construits autour du triangle constitué de l'apprenant, du domaine de connaissance, et de l'enseignant. En cela l'application que nous visons diffère des EAIO : l'enseignant n'est pas modélisé en tant que tel. Nous ne mettons pas en place de démarche pédagogique, et les dialogues entre l'utilisateur et le système sont réduits au minimum.

L'étude des EAIO peut néanmoins être instructive. Nous avons vu avec le modèle LOM comment modéliser des domaines de connaissances. L'idée de pré-requis, notamment, est intéressante. L'utilisateur décrit ses connaissances ; le système déduit l'ensemble des traitements

[4] Nous présenterons ce domaine au chapitre 4, p. 151.

[5] Personal Digital Assistant, petits ordinateurs portables dont la vocation première est de servir d'agendas électroniques. Ils tiennent généralement dans la poche et se présentent sous forme d'écran "ardoise".

dont les modes d'emploi sont accessibles, et l'ensemble des connaissances manquantes pour un mode d'emploi donné.

3.2 Ontologies et règles, réceptacles des connaissances pour le raisonnement

3.2.1 Les ontologies en représentation des connaissances

Le terme d'ontologie remonte à Socrate et Aristote (400-360 avant JC) pour désigner, en philosophie, l'étude "de ce qui est" [RNK+04]. Il est emprunté par la communauté de l'ingénierie des connaissances pour désigner "la spécification explicite d'une conceptualisation" [Gru93]. Plus précisément, dans notre contexte, une ontologie est constituée :

- de concepts,
- de relations entre concepts,
- de contraintes sur les relations et d'axiomes de classes,
- d'individus[6],
- de façon associée à l'ontologie, des fonctions d'interprétation sont définies pour les constructeurs du langage utilisé, permettant ainsi de déduire les conséquences logiques des connaissances représentées.

La vocation d'une ontologie est de définir un vocabulaire et une compréhension partagée. Certaines ontologies sont particulières : ce sont les ontologies formelles représentées dans un langage doté de fonctions d'interprétations. Les ontologies formelles peuvent servir de support au raisonnement automatique. Elles sont utilisées, en particulier, dans le cadre du Web Sémantique. Les principes sur lesquels reposent les ontologies formelles sont relativement anciens : les logiques de description et les langages de *frames* datent des années 70. Ces principes sont exposés dans des ouvrages comme Haton ou Kayser, sans qu'apparaisse le terme "ontologie" qui s'est maintenant imposé.

L'intérêt pour les ontologies formelles semble de plus en plus évident, comme en témoigne la variété des domaines où elles trouvent applications (médecine, industrie automobile, audio-visuel, e-learning, droit juridique, géographie, etc.). Cet intérêt n'est pas réductible à un effet de mode dans la mesure où les ontologies répondent effectivement à des besoins de représentation des connaissances auparavant insatisfaits. Cependant, il existe des formes et des exploitations variables d'ontologies. La figure 3.2 le montre.

Les catalogues, glossaires et dictionnaires fournissent un vocabulaire, mais n'organisent pas de façon systématique les concepts avec des relations déterminées. Les thesaurus[7] tendent à une telle organisation, principalement au moyen de la relation de subsomption sur laquelle repose les taxinomies et les systèmes de *frames*. Les ensembles d'axiomes logiques, ou axiomes de classe, permettent de définir des concepts en exprimant des conditions suffisantes d'appartenance. Les notions de *concepts primitifs* et *concepts définis* sont introduites.

"Les concepts primitifs dénotent de grandes catégories naturelles comme les personnes, les animaux, les plantes, etc., et servent à construire les concepts définis. (...) Un concept primitif est une description incomplètement spécifiée car il exprime des conditions nécessaires mais non suffisantes : il n'est pas possible de reconnaître un représentant d'un concept primitif au vu de ses seuls rôles." ([HBF+91] p.334-335.). Dans l'exemple ER 2 suite, Fonctionnalité est un concept

[6]Les individus instances de concepts sont parfois considérés comme ne faisant pas partie de l'ontologie. Cependant, lorsque la définition de concepts comporte une référence à des individus (par exemple, les LogicielESRI sont des Logiciel dont la propriété lieuDeDéveloppement a pour valeur l'individu ESRI instance du concept Organisation), concepts et individus sont nécessairement associés.

[7]Selon la définition de la norme ISO 5694-1, un thesaurus est un "vocabulaire d'un langage d'indexation contrôlé, organisé formellement de façon à expliciter les relations *a priori* entre les notions (par exemple les relations générique-spécifique)".

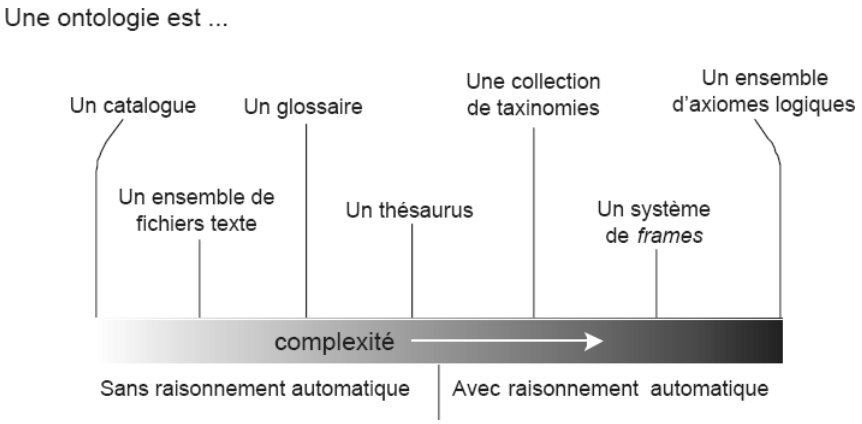


FIG. 3.2 – Différentes acceptations du terme "ontologie" (extrait de [Tro04], d'après [SW01])

primitif et GénéralisationCartographique est un concept défini par les conditions nécessaires "être une sorte de FonctionnalitéGéographique" et "résoudre un problème de lisibilité".

Nous avons mentionné, lors de la discussion de l'exemple ER 2, que la présence d'axiomes de classes distingue les ontologies légères des ontologies lourdes.

Lien entre modèle de métadonnées de traitements et ontologies

Les diagrammes de classes de notre modèle conceptuel de métadonnées présentés au chapitre 2 ne définissent pas la taxinomie des concepts utiles à la description des traitements. Par exemple, la hiérarchie des fonctionnalités géographiques n'apparaît pas dans le modèle conceptuel. Elle apparaît dans la base de métadonnées instance du modèle.

Pour répondre aux requêtes de l'utilisateur, il nous faut pourtant exploiter les relations de subsomption qui unissent entre elles les fonctionnalités, les types de données, les familles de traitements, les problèmes résolus par les traitements, les modes d'emploi, etc. Pour mener à bien cette tâche, il faut transposer le modèle et la base de métadonnées dans des ontologies. Il faut passer d'un SI à un SBC.

Plusieurs raisons motivent le choix de cette architecture duale ; nous y reviendrons par la suite. Notons cependant dès maintenant trois différences importantes entre ontologie et modèle de métadonnées. Les différences données ci-dessous s'appuient sur des citations où il est question de comparaison entre ontologies et schémas de BD. Notre modèle de métadonnées étant utilisé comme schéma de base de données dans le cadre d'un système d'information, les arguments s'appliquent parfaitement à notre contexte.

- Au contraire des schémas de BD, les ontologies sont en général destinées à évoluer. En particulier, elles doivent être étendues par l'ajout de nouveaux concepts et relations [Cos03][Ges05].
- Une ontologie peut être très volumineuse, pas un schéma de BD [Ges05].
- Les différents SGBD permettent certes d'obtenir des informations sur les schémas de BD, mais ils restent avant tout conçus pour répondre aux requêtes sur le contenu des BD. Dans une ontologie on requête en revanche aussi bien les concepts que les instances. Cela correspond, dans le langage de logiques de description qui implémentent les ontologies formelles, à deux types de requêtes : celles qui portent sur la terminologie (T-Box) et celles qui portent sur les assertions (A-Box).

Pour automatiser une partie du raisonnement nécessaire à nos besoins, nous établissons donc le pont entre modèle de métadonnées et base de métadonnées d'une part, et ontologies d'autre

part. Mais il n'y a pas équivalence entre les deux parties. L'ontologie n'est qu'une représentation formelle des connaissances contenues dans le modèle et la base de métadonnées. L'ontologie ne constitue pas non plus tout le SBC, ce n'en est qu'une partie. Comme le souligne J. Charlet, ontologie et base de connaissance sont bien à distinguer : "Une ontologie recense ce qui existe et le définit par ses propriétés essentielles. Elle ne rapporte pas tout ce qui arrive. La base de connaissances, elle, utilise les descripteurs ainsi fournis par l'ontologie, pour énoncer tout ce qu'il faut savoir sur le domaine" [Cha03]. Dans notre contexte le SBC peut être vu comme l'ensemble du modèle et de la base de métadonnées, des ontologies qui sont une représentation formelle des concepts utilisés, de la représentation opérationnelle des règles de l'expert et enfin de l'application qui exploite le tout.

Formalismes de représentation des connaissances appropriés aux ontologies

Afin d'utiliser les ontologies dans le cadre d'un SBC, R. Troncy, dans un contexte similaire au nôtre, a été amené à étudier en particulier deux formalismes de représentation des connaissances ([Tro04] pp. 63-71). Ces deux formalismes sont les logiques de description (LD) et les graphes conceptuels (GC). R. Troncy s'est inspiré des notes de cours de J. Euzenat donnés entre 1996 et 1999 à l'université J. Fourier de Grenoble [Euz99]. Nous nous référons également à ce document.

Les LD et les GC sont deux formalismes inspirés à la fois des réseaux sémantiques et des langages de *frames*. Ils sont tous deux dotés d'une sémantique formelle qui permet la mise en œuvre de mécanismes d'inférence sur les connaissances représentées. Bien que proches, leurs caractéristiques propres destinent les langages basés sur les LD à représenter la partie terminologique des ontologies, et les langages basés sur les GC la partie assertionnelle [Tro04].

Les logiques de description

La définition que nous avons donnée des ontologies formelles p. 127 se traduit bien avec le formalisme des logiques de description : nous avons affaire à des concepts, des rôles et des individus. Les concepts modélisent des classes d'individus. Ils peuvent être primitifs ou définis. Les rôles modélisent des relations entre classes. La relation de subsomption permet d'organiser les concepts et les rôles en hiérarchies ; la classification et l'instanciation sont les opérations qui sont alors à la base du raisonnement sur les descriptions (raisonnement terminologique). La classification permet de déterminer la position d'un concept et d'un rôle dans leurs hiérarchies respectives, tandis que l'instanciation permet de retrouver les concepts dont un individu est susceptible d'être une instance.

Les logiques de description constituent un ensemble restreint de formules de la logique du premier ordre. On parle de "logiques" au pluriel car il en existe plus d'une dizaine de types à l'expressivité variable, selon que l'on autorise ou non la définition de concepts avec les constructeurs "au moins", "au plus", avec des restrictions sur les rôles (*i.e.* les propriétés), etc.[8].

Les logiques de description ont en général de bonnes propriétés computationnelles, mais ne sont pas toujours décidables[9]. Par exemple, les logiques de description $\mathcal{U}$ et $\mathcal{R}$ sont indécidables pour le calcul de subsomption de concepts ([Euz99], p.39). P.F. Patel-Schneider indique

[8] Pour plus de détails, se reporter à [Euz99], pp. 37-42. Un diagramme montrant les différents types de logiques de descriptions, de $\mathcal{ALC}$ à $\mathcal{SHOIQ}$, se trouve également à l'adresse `http://www.cs.man.ac.uk/~ezolin/logic/navigator/add/alc_to_shoiq.ps`, et la page `http://www.cs.man.ac.uk/~ezolin/logic/complexity.html` propose un formulaire indiquant selon les besoins d'expressivité le type de logique de description nécessité.

[9] "Une logique est décidable s'il existe un procédé de calcul qui, pour tout formule, indique en un temps fini s'il s'agit ou non d'un théorème de cette logique" [Kay97] p.69.

également que le calcul de subsomption est indécidable pour le système NIKL[10] [Pat03].

La vérification de consistance (*est-ce qu'une classe peut avoir une instance ?*), la classification (A *est-elle une sous classe de* B *?*) et la classification d'instance (*à quelle classe appartient un individu ?*) sont des inférences permises par les systèmes de logiques de description [KMN04]. Ces inférences ne suffiront pas à exprimer toutes les connaissances d'expert dont nous avons besoin. C'est pourquoi nous allons voir plus loin que nous devons employer en complément un langage de règles. En effet, les logiques de description et les langages de règles comme ceux basés sur les clauses de Horn[11] sont deux sous-ensembles de la logique du premier ordre qui ne se recouvrent pas totalement :

- Les logiques de description ne permettent pas l'usage de variables pour définir les concepts. Par exemple on ne peut définir le concept "ami" par la condition suffisante "ennemi d'un ennemi". On peut par contre exprimer cela par la règle $ennemi(x, y) \wedge ennemi(y, z) \Rightarrow ami(x, z)$.
- Le langage de logique basé sur les règles de Horn nous interdit certaines inférences possibles avec les logiques de description. Par exemple, définissons un docteur comme quelqu'un ayant écrit une thèse. Un raisonneur de logique de description saura déduire que si Léon est un docteur, alors il a écrit une thèse, même si à aucune instance du concept *thèse* n'est présente dans la base de fait. Maintenant, ajoutons qu'un auteur comme est quelqu'un qui a écrit un document et qu'une thèse est un document. Le raisonneur de logique de description est capable de déduire qu'un docteur est un auteur[12]. Ce raisonnement n'est pas possible avec des règles où les variables du conséquent doivent apparaître dans l'antécédent.

Malgré ces différences, certaines connaissances peuvent être formulées de façon équivalente avec une LD ou avec un langage de règle. Par exemple la définition du concept "électeur" *a pour condition nécessaire "est(majeur)"* n'est qu'une autre formulation de la règle *"si x est un électeur alors x est majeur"*. La première façon de représenter les connaissances peut sembler plus "naturelle" [Euz99]. Nous la privilégions dans notre SBC.

Les graphes conceptuels

"Un réseau sémantique est un graphe étiqueté dans lequel les nœuds figurent des objets ou des concepts et les arcs étiquetés des relations" [HBF+91]. On peut vouloir préciser cette définition en ajoutant que le graphe est orienté [Kay97] ; sinon le sens des arcs étiquetés serait parfois ambigu (pour les relations non symétriques).

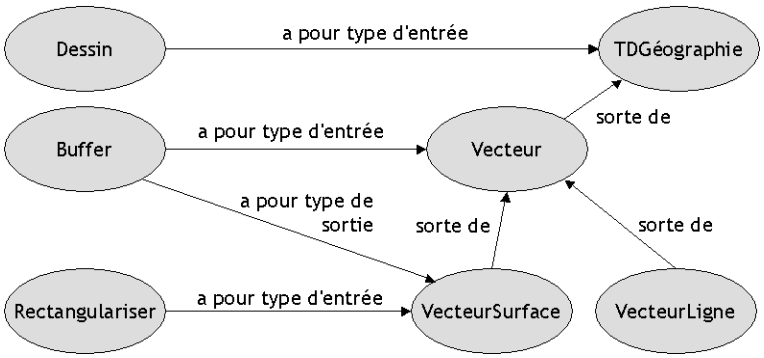
Un graphe conceptuel est une sorte particulière de réseau sémantique : c'est "un multi-graphe connexe biparti composé de deux types de nœuds : des concepts et des relations. Les nœuds relations possèdent un ou plusieurs arcs qui les lient aux nœuds concepts"[Tro04]. La notion de *support* permet de contraindre la construction des graphes : un vocabulaire et des schémas de relations sont définis. Une définition formelle de la notion de support peut être trouvée dans (*ibid*, pp. 65-66).

Il est possible de raisonner sur les graphes en introduisant la notion de conséquence : un graphe G' est conséquence d'un graphe G si toute information contenue dans G est aussi contenue dans G'. Par exemple, le graphe G' qui indique que l'algorithme "Gauss" réalise la fonctionnalité "simplification" est conséquence du graphe G qui indique que ledit algorithme réalise la fonctionnalité "lissage" (car un "lissage" est une sorte de "simplification").

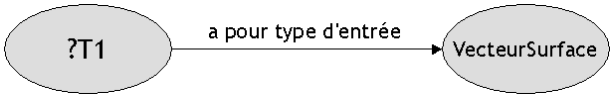
[10] New Implementation of KL-ONE, Knowledge Language-ONE étant un langage de représentation de connaissances dont sont issus les logiques de description ([Mug02], p. 5.).

[11] Une clause de Horn est une formule de la forme $p_1 \wedge p_2 \wedge ... \wedge p_n \Rightarrow q$

[12] Merci à Antoine Isaac pour m'avoir fourni cet exemple.



Le graphe ci-dessus est le graphe des faits, c'est-à-dire de la base de métadonnées. Les graphes ci-dessous sont des graphes requêtes. Pour obtenir les réponses des requêtes, on essaie d'appareiller les graphes requêtes au graphe des faits.



Réponse souhaitée : **D**essin, **B**uffer et **R**ectangulariser.
Expliquons ce souhait en prenant un exemple de code Java. Déclarons la classe `VecteurSurface` ainsi :

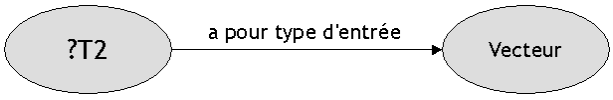
```
class VecteurSurface extends Vecteur{}
```

Déclarons ensuite la méthode `buffer` :

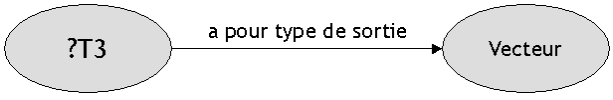
```
void buffer(Vecteur v){}
```

Ceci posé, un utilisateur demande : "quelle méthode Java puis-je appliquer à mon objet de type `VecteurSurface?`"

La réponse est buffer(). Cet exemple est intéressant car pour aboutir à la réponse, l'exploitation de la relation de subsomption est utilisée dans le *sens de généralisation*, soit le sens inverse de celui utilisé par l'opération classique d'appariement de graphes.



Réponse souhaitée : **D**essin et **B**uffer.
Rectangulariser ne convient pas car les données de l'utilisateur peuvent être de type **V**ecteurLigne. En l'absence de précision, les seuls traitements dont on est sûr qu'ils conviennent sont donc **D**essin et **B**uffer



Réponse souhaitée : **B**uffer.

TAB. 3.1 – Formalisation de type "graphe conceptuel" de l'exemple ER 1

Nous avons besoin de réaliser l'inférence classique reposant sur ce type de conséquence basée sur la relation de subsomption, mais pas uniquement. Nous avons aussi besoin d'inférences *ad hoc* où les relations de subsomption sont exploitées dans les deux sens. C'est ce que montre le tableau 3.1. Nous aurions peut-être pu chercher à utiliser des moteurs d'inférences spécifiquement basés sur les algorithmes d'appariement de graphes, mais nous verrons au chapitre 5 que nous avons choisi de recourir à l'expression de règles pour répondre aux requêtes comme ER 1.

3.2.2 Les langages de règles

Nous avons besoin d'un langage de règles pour exprimer les connaissances non supportées par les logiques de description. Plusieurs acteurs de la communauté du Web sémantique confrontés à ce même constat citent l'exemple typique de la transmission de valeur de propriété associée aux relations de méréologie : par exemple, *si une voiture est de couleur verte alors la portière, partie de la voiture, est verte* (cette règle ne peut être exprimée en LD car elle requière l'emploi de variables).

La question à laquelle nous allons répondre ici est de savoir de quels types de règles nous avons besoin et quelle interprétation doit pouvoir être faite de la manipulation de symboles de notre système formel.

Hypothèse de travail : des raisonnements basés sur la déduction

Les raisonnements permis par les logiques de description sont basés sur la déduction, ceux que l'on va permettre avec nos règles également. Pourtant, l'expert humain est capable de raisonnements d'autres types : induction, analogie, abduction[13], et peut-être même d'autres processus cognitifs non identifiés (notamment tous ceux liés à la perception ou à l'apprentissage). Haton *et al.* parlent aussi de raisonnement procédural lorsque la conduite du raisonnement est figée dans des algorithmes, ou de raisonnement géométrique dans le cadre de la perception visuelle [HBF+91].

L'analogie peut être mise en œuvre avec des systèmes de raisonnement à partir de cas (RàPC), l'induction avec les divers systèmes d'apprentissage développés en IA[14], notamment les réseaux de neurones. Nous n'explorons pas ces pistes de recherches. Par hypothèse de travail, nous limitons notre ambition à la mise en œuvre de raisonnements basés sur la déduction.

La logique des prédicats : un degré d'expressivité nécessaire

Nous avons besoin d'exprimer des règles portant non pas sur des éléments individuels de notre modèle de métadonnées, mais sur des classes d'éléments. Cela signifie que l'on ne peut se placer dans le cadre étroit de la logique propositionnelle (ordre 0 ou 0+), mais au moins dans celui de la logique des prédicats avec variables (ordre 1). Nous avons besoin d'écrire des règles générales comme :

SI formatEntréeFournieParUtilisateur(X)
ET formatEntréeAttendueTraitement(Y)
ET $\neg$*(sorte_de(X, Y)*
ALORS étapeAdaptation(X, Y).

[13] L'abduction peut être vue comme une méthode de construction d'hypothèse. Si les éléphants sont gris et que Clyde est gris, on infère par abduction que Clyde est un éléphant [HBF+91].
Le schéma d'inférence est le suivant : $\frac{\forall x(P(x) \Rightarrow Q(x)) \quad Q(a)}{P(a)}$.

[14] Un tel système a été mis en œuvre dans le domaine de la généralisation cartographique. Dans son travail de thèse, S. Mustière a utilisé l'algorithme d'apprentissage supervisé RIPPER pour acquérir des règles décidant des traitements de généralisation à appliquer en fonction des propriétés des données [Mus01]. Cet exemple montre l'intérêt de prévoir la spécialisation des descriptions du fonctionnement des traitements (évoquée p. 107).

L'expression de la règle ci-dessus appelle trois remarques.

Les symboles X et Y représentent des variables. De façon implicite intervient le quantificateur universel $\forall$.

La troisième prémisse est $ET \neg(sorte\_de(X,\ Y))$ est plus restrictive que $X \neq Y$. Le prédicat *sorte_de* permet de gagner en expressivité. Pour l'évaluer nous recourrons aux T-Box des logiques de description.

En marge de la question du type de logique employé, on voit se poser le problème du moyen employé pour faire référence à des éléments de la base de métadonnées. L'utilisation des prédicats *formatEntréeFournieParUtilisateur* et *formatEntréeAttendueTraitement* est possible mais peu raisonnable car guère pratique à manipuler. Cela supposerait de renoncer à la structure arborescente de nos métadonnées. La notation utilisée dans les langages de programmation orientés objets est plus élégante ; elle permettrait en l'occurrence d'exprimer la règle sous la forme : *SI utilisateur.entrée.format* $\neq$ *traitement.entrée.format ALORS ...* Nous reviendrons sur la question de l'adressage des éléments du modèle dans les règles au cours du chapitre dédié à l'implémentation.

Nous venons donc de voir que l'expressivité de la logique d'ordre 1 est nécessaire à nos besoins. Est-elle suffisante ?

Logique floue, logique modale, logique non-monotone : des pistes de recherches intéressantes mais que nous ne mettons pas en œuvre

Logique floue

Les besoins de l'utilisateur s'expriment parfois de façon floue (*"je veux un traitement de généralisation qui ne déplace* ***pas trop*** *les bâtiments"*). Néanmoins, la majorité des informations sur les traitements ne font pas intervenir de probabilités. La logique floue n'apparaît pas indispensable dans notre contexte. Nous avons donc choisi de ne pas y recourir. Dans l'hypothèse d'un choix contraire, l'objectif d'une exploitation des règles floues n'aurait pas été forcément irréalisable car des moteurs d'inférence existent.

Logique modale

Un utilisateur peut connaître de façon approfondie un traitement particulier, voire plusieurs traitements. Cela n'en fait pas pour autant un expert véritable. Pour prétendre à ce titre, il faut posséder une qualité supplémentaire, celle d'être capable de transformer son savoir en connaissances générales qui permettent de fournir aux questions des réponses telles que :

- Il est possible qu'il existe une version Macintosh de FreeWRL.
- Il est certain que MSPaint ne peut manipuler des données *vecteur*.
- Il est plausible que l'erreur d'exécution de ce programme d'appariement provienne non pas du programme mais des données.
- À l'IGN il est interdit d'utiliser des logiciels piratés.

On voit présents dans ces énoncés des opérateurs particuliers, les opérateurs modaux. Le tableau 3.2 montre les modes de trois logiques modales.

Les opérateurs modaux apportent de la connaissance sur la connaissance, donc de la méta-connaissance. Ils peuvent avoir un rôle heuristique : par exemple il est inutile de chercher un traitement dont on sait qu'il ne peut exister. Cette considération demande d'apporter une

[15] d'après [Kay97] et `http://fr.wikipedia.org/wiki/Logique_modale`

Modes classiques (ou aristotéliciens)	**Modes épistémiques** (relatifs à la connaissance)	**Modes déontiques** (moraux)
nécessaire	établi	obligatoire
contingent	contestable	interdit
possible	exclu	permis
impossible	plausible	facultatif

TAB. 3.2 – Les modes de trois types de logiques modales[15]

précision importante : celle du choix de nous placer dans un monde ouvert ou dans un monde clos.

L'hypothèse du monde ouvert

Raisonnons-nous en faisant l'hypothèse d'un monde ouvert ou d'un monde clos ? Autrement dit, les informations absentes de notre base de données sont-elles supposées fausses ou non ?

Prenons l'exemple de la description du SIG Geoconcept 5. Geoconcept 5 est indexé et décrit dans notre base de métadonnées. Nous devons tolérer les descriptions partielles, donc incomplètes (objectif O7d), car exiger une description exhaustive serait un obstacle à la tâche déjà difficile d'acquisition des métadonnées. Il se trouve en l'occurrence que la description de Geoconcept 5 omet de mentionner que la fonctionnalité de détection d'incohérence entre les objets linéaires est réalisée. C'est un manque. En revanche, il existe évidemment quantités de fonctionnalités que Geoconcept 5 ne réalise pas et qui, de fait, sont absentes de la description. Par exemple Geoconcept 5 ne permet pas d'effectuer de généralisation cartographique des données.

L'approche classique dans le monde des BD est d'adopter l'hypothèse du monde clos, au contraire des systèmes basés sur les logiques de description où c'est l'hypothèse du monde ouvert qui est postulée [CPSV03].

Nous choisissons de faire l'hypothèse du monde ouvert.

Logique non-monotone

Le tableau 1.7 (p. 28) montre que certaines connaissances à représenter peuvent être évolutives ou sujettes à exception. Les représenter impliquerait de devoir tolérer la présence de faits contradictoires dans notre base de métadonnées. Pour raisonner dans ces conditions, il existe des systèmes proposant des mécanismes de révision de connaissance, donc permettant une logique non monotone. Nous faisons le choix de ne pas nous situer dans ce cadre de travail. Nous supposerons donc que notre base de métadonnées est consistante et que les règles acquises auprès des experts ne souffrent pas d'exceptions[16].

3.2.3 Validité, complétude, consistance et décidabilité

Il paraît difficile d'échapper, lorsque l'on prétend raisonner, aux questions concernant la validité des inférences effectuées. L'enjeu n'est pas que théorique. En pratique, dans un contexte d'application plus large que le nôtre, le risque de fournir des instructions d'utilisation de traitements géographiques erronées pourrait ne pas être admissible. On imagine par exemple les conséquences potentielles d'une erreur de changement de projection cartographique dans des

[16]Il est possible de normaliser une règle avec exception : il suffit d'ajouter une condition "sauf si.."dans la partie prémisse. Chalmers explique que si l'ajout est *ad hoc* le procédé est mauvais ; il est en revanche bon s'il est l'expression d'une propriété qui permet de mieux comprendre le domaine modélisé[Cha87]. Par exemple pour la règle "Si oiseau(x) Alors vole(x)", l'ajout de "sauf si autruche" est une mauvaise restriction, alors que "sauf si surface aile petite" en est une bonne. On retrouve prescrits ces mêmes types de principes en programmation à propos du jugement des "verrues" de code chargées de gérer les exceptions.

domaines comme les transports aériens ou les conflits militaires. La question de la confiance dans les informations fournies est donc importante. Notre système formel de raisonnement est-il valide, est-il complet, est-il décidable ? Autrement dit, peut-on jurer que notre système calcule la vérité, toute la vérité, et rien que la vérité ? Qu'entend-t-on par ailleurs par "vérité" ?[17]

Interprétation des langages utilisés et théorie des modèles

Pour représenter les connaissances nous nous dotons de langages de logique de description et de logique basée sur les clauses de Horn. Ces langages sont constitués de symboles (de constantes ou de concepts, de prédicats ou de rôles, de connecteurs logiques, de parenthèses ; de variables et de quantificateur universel, en plus, pour les clauses de Horn). Ces langages respectent aussi chacun une syntaxe.

Pour établir le "pont" entre le niveau symbolique et le niveau sémantique, on définit des fonctions d'interprétations. C'est-à-dire que l'on définit les correspondances entre les symboles de constantes des langages et les objets du "monde réel" d'une part, entre les symboles de prédicats et leur signification d'autre part. Pour un langage $\mathcal{L}$, l'ensemble des symboles de constantes, de leur interprétation et des objets du "monde réel" forme ce qui est appelé une $\mathcal{L}$-structure. Considérant une $\mathcal{L}$-structure donnée, une proposition se verra attribuer une valeur de vérité "vrai" (1) ou "faux" (0). Par exemple, $|sorte\_de(l, s))|_{\mathbf{A}} = 1$ avec $\mathbf{A} = \langle$"lissage", "simplification", "sorte de", l, s, $sorte\_de\rangle$.

Une théorie est un ensemble de propositions. Par exemple on peut définir la théorie $\mathcal{T}$ comme l'ensemble {"lissage sorte généralisation", "simplification sorte généralisation"}. Si une $\mathcal{L}$-structure $\mathbf{S}$ satisfait tous les énoncés de la théorie $\mathcal{T}$, on dit que $\mathbf{S}$ est vraie dans $\mathcal{T}$, ou que $\mathbf{S}$ est un modèle de $\mathcal{T}$. On note cela $\mathbf{S} \models \mathcal{T}$.

Validité et complétude

L'existence d'un modèle pour une théorie équivaut à la non-contradiction de celle-ci [DRL00]. La théorie des modèles garantit la validité logique d'une preuve : une règle de déduction est valide lorsque tout modèle de ses prémisses est aussi un modèle de sa conclusion. Cela répond à la question de la correction des inférences effectuées.

La complétude du calcul de la logique du premier ordre a été prouvée par K. Gödel en 1929 [Del02]. Bien sûr, la complétude du système dépend des axiomes de base que l'on se donne. Pour poursuivre l'exemple précédent, si l'on omet de définir l'axiome *sorte_de(l, s)* plusieurs déductions au sujet de la hiérarchie des fonctionnalités géographiques deviennent impossibles. De la même façon, la géométrie euclidienne ne serait plus complète si était enlevé l'axiome des parallèles[18](*ibid.*).

Un système logique effectue, à partir d'un ensemble initial d'axiomes, des déductions. S'il effectue toutes les déductions possibles, il est complet. Cela signifie n'implique pas que l'ensemble de faits obtenus soit non contradictoire ; cela signifie en revanche que si le système est contradictoire on le saura.

Plus un langage est expressif, plus le risque de perdre la propriété de complétude est grand. Par exemple, l'arithmétique basée sur les cinq axiomes de Péano est incomplète ; c'est le prix de son expressivité.

[17] Pour la rédaction de la suite de cette section nous nous sommes principalement inspirés de [Del02], [DRL00] et [AS94].

[18] Si on pouvait le déduire des autres axiomes – ce qui n'est pas le cas –, alors son omission n'aurait pas affecté la complétude du système.

Calculabilité et décidabilité

Automatiser le raisonnement, cela signifie, en pratique, écrire des programmes informatiques qui le simulent. La question qui se pose est alors de savoir s'il existe des algorithmes de calcul pour le système de logique que nous nous sommes donné, donc si ce système est calculable. Il y a un peu plus d'un demi-siècle, plusieurs définitions équivalentes de la classe des fonctions calculables ont été proposées par Church, Gödel et Turing [HBF+91]. Les fonctions de certains systèmes logiques ne sont pas calculables : elles sont indécidables.

La logique du premier ordre est semi-décidable, c'est-à-dire qu' "il existe des procédures de calcul dont l'exécution pour une formule quelconque donnée se termine toujours si elle est contradictoire mais ne se termine éventuellement pas si elle n'est pas contradictoire " ([HBF+91]). Cette propriété de semi-décidabilité peut faire penser à la dissymétrie des critères de scientificité définis par K. Popper : il n'est en général pas possible de prouver la véracité d'un énoncé scientifique, mais on doit en revanche pouvoir prouver sa fausseté.

On a vu que certaines inférences des logiques de description expressives étaient indécidables. La logique du premier ordre réduite aux clauses de Horn, quant à elle, est décidable.

Concernant les langages d'implémentation que nous allons utiliser, les références sur la décidabilité sont [W3C04d] (§5.1) pour le langage de définition d'ontologie OWL-DL, et [Pan04] (p.2) pour le langage de règles SWRL Lite. Cependant, les propriétés théoriques de ces langages ne seront pleinement exploitées que si, en amont, les connaissances que l'on exprime sont effectivement valides, complètes et consistantes – objectif qui sera à la fois difficile à atteindre et à vérifier.

Nous venons de voir que les deux formalismes de représentation des connaissances sur lesquels nous comptons faire reposer notre système raisonnant sur les métadonnées sont les ontologies et les règles ; et que ces deux formalismes offrent des garanties quant à la validité des inférences rendues possibles. Il nous reste maintenant, concrètement, à achever de définir notre modèle de métadonnées afin de permettre la représentation des connaissances nécessaire à la mise en œuvre des exemples de raisonnements ER exposés en début de chapitre.

3.3 Proposition pour raisonner sur les métadonnées des traitements

3.3.1 Scénario d'une adaptation de mode d'emploi au contexte d'utilisation

La figure 3.3 montre le déroulement d'un scénario type mettant en jeu une adaptation de traitement. Le formalisme est inspiré des diagrammes de séquence UML. Les messages sont de type *synchrone*.

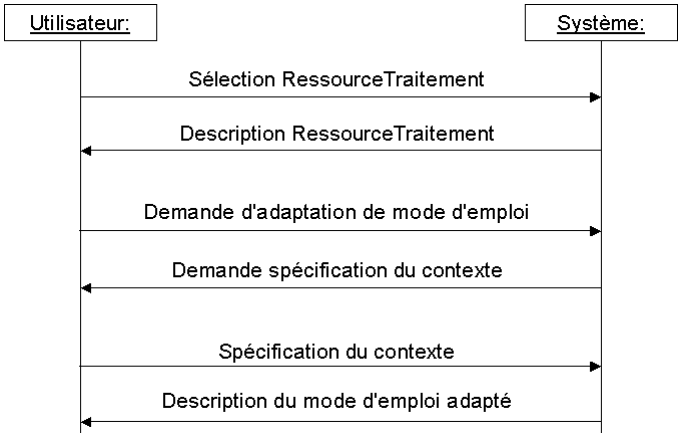


FIG. 3.3 – Diagramme de séquence UML utilisateur / système pour l'adaptation d'un mode d'emploi

3.3.2 Quatre types d'adaptation des modes d'emploi

Les modes d'emploi sont des agencements d'étapes. Quatre types d'adaptation sont possibles :

- Lorsque que des pré- ou post- conditions ne sont pas satisfaites, l'adaptation consiste à indiquer à l'utilisateur les instructions à effectuer. Par exemple, l'utilisateur veut des donnés MapInfo avec Arcview. Une conversion au format *shape* est nécessaire. La règle d'adaptation adéquate (cf. code A.2) est déclenchée : une étape est ajoutée au mode d'emploi.
- Lorsqu'une étape de mode d'emploi est de type ÉtapeFct, l'adaptation consiste à proposer des ÉtapePrg ou des ÉtapeIHM correspondantes. Parfois, le système doit chercher les RessourceTraitement qui réalisent les fonctionnalités nécessitées. Par exemple, "changer la projection d'un jeu de données" est une étape qui se réalise différemment selon le SIG utilisé. Dans ce cas, l'adaptation peut se limiter à indiquer à l'utilisateur quels menus sélectionner. Si la RessourceTraitement du contexte de l'utilisateur ne le permet pas, l'adaptation peut aussi nécessiter une recherche des plugins ou API complémentaires réalisant le changement de projection.
- Lorsqu'une étape de mode d'emploi est de type ÉtapePrg ou ÉtapeIHM et qu'elle n'est pas réalisable dans le contexte de l'utilisateur, l'adaptation consiste soit à proposer des conseils, soit à proposer une RessourceTraitement alternative. L'exemple ER 3 p. 123 montre de telles adaptations.
- Lorsque l'utilisateur soumet un problème d'utilisation, l'aide du système se traduit par l'indication d'un mode d'emploi adapté. Par exemple, le problème de qualité lié à l'impression d'une carte peut avoir des causes multiples auxquelles les règles de l'expert associent des solutions.

La figure 3.4 résume les différents types d'adaptation possibles. Notons que les premier et quatrième types d'adaptation ajoutent des étapes ; il se peut que le traitement considéré ne possède aucun mode d'emploi décrit dans la base de métadonnées.

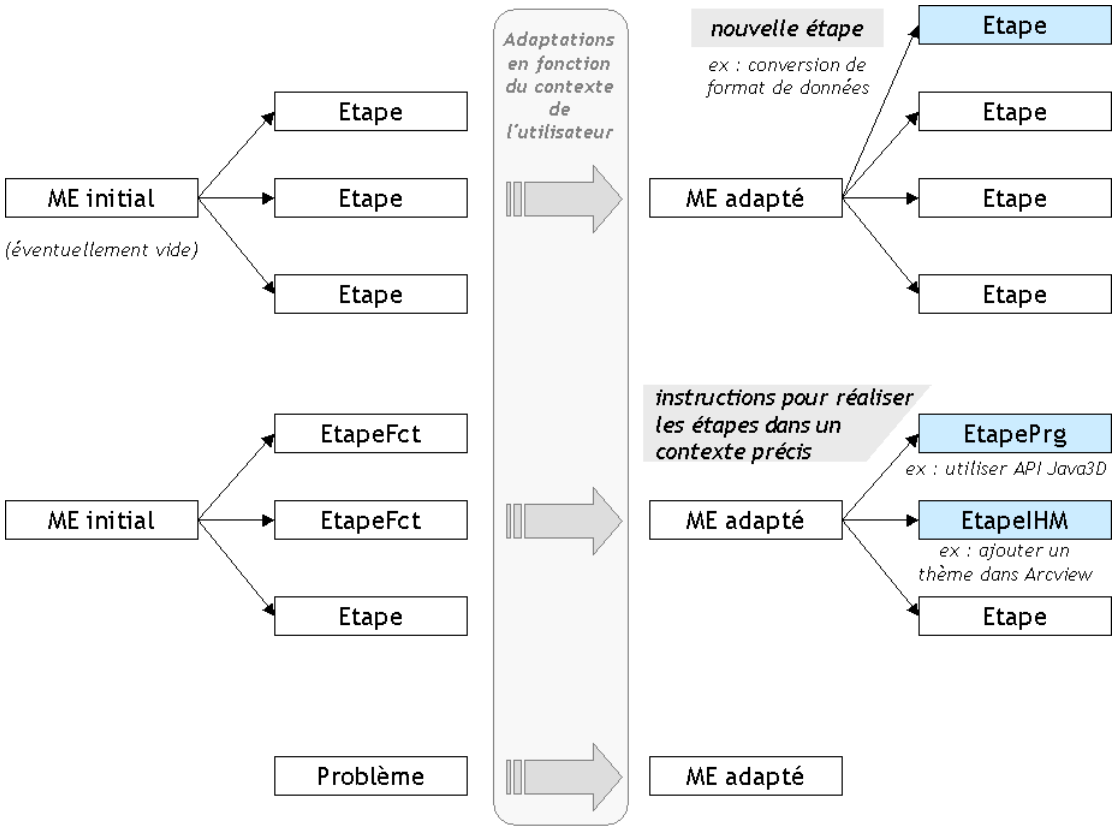


FIG. 3.4 – Les types d'adaptation des modes d'emploi

Pour effectuer les adaptations ci-dessus, nous considérons ici que l'expert tient deux types de raisonnement. D'abord il identifie les problèmes d'utilisation causés par l'écart entre le contexte de l'utilisateur et le contexte requis par le traitement ; ce faisant, il effectue des raisonnement de classifications (p.ex. pour déterminer si le format des données de l'utilisateur est une sorte du format attendu). À chaque mal, son remède : une fois le diagnostic posé, l'expert applique les règles déductives associant problèmes et adaptations de mode d'emploi. Comme souligné p. 130, les deux types de raisonnements mobilisés peuvent faire appel à deux sous-ensembles distincts de la logique du premier ordre.

Nous allons maintenant définir la façon dont nous proposons de modéliser les connaissances impliquées dans les raisonnements permettant l'adaptation des modes d'emploi.

3.3.3 Contexte de l'utilisateur et contexte requis par le traitement

Les règles d'adaptation sont composées de prémisses et de conclusions. Évaluer une prémisse, c'est évaluer la correspondance entre le contexte de l'utilisateur et le contexte du traitement. Les types des propriétés des classes ContexteUtilisateur et ContexteTraitement sont donc identiques.

Les instances des classes ProfilUtilisateur et RègleAdaptation sont stockées dans la base de métadonnées. Les instances de ContexteTraitement ne le sont qu'indirectement, puisqu'elles sont déduites dynamiquement à partir des facettes Fonction des RessourcesTraitement. Les instances de ContexteUtilisateur, quant à elles, sont construites au cours de l'utilisation de l'application grâce aux renseignements de l'utilisateur. Les instances de DeltaContexte sont calculées automatiquement, ainsi que les NouvelleEtape, EtapeMEAdapté et QuestionUtilisateur.

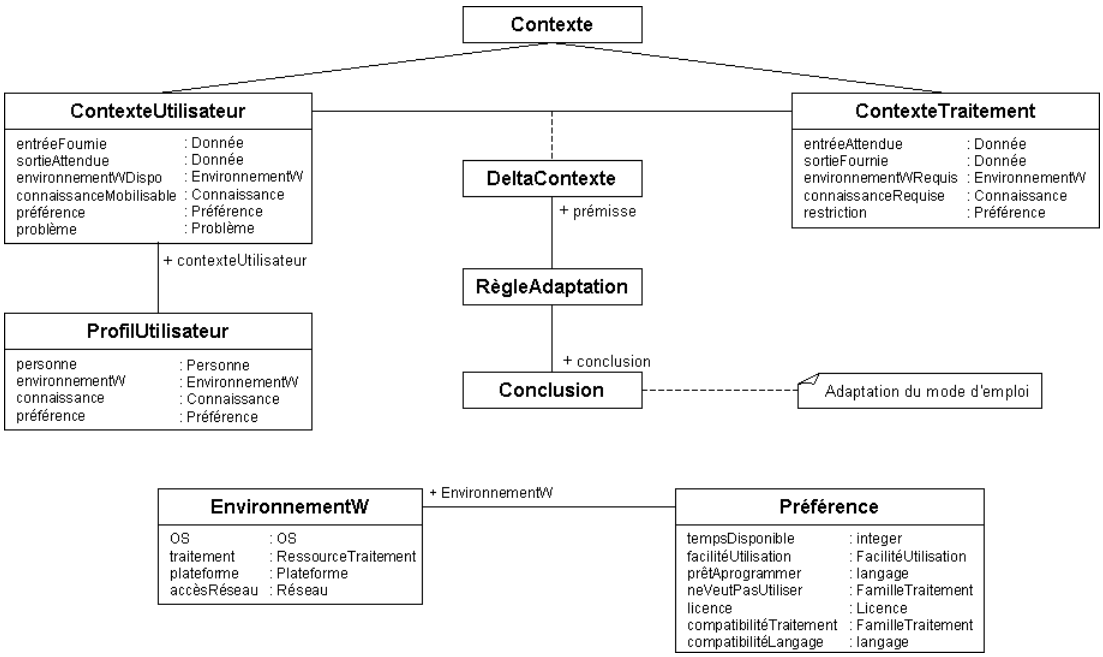


FIG. 3.5 – Contexte de l'utilisateur et contexte requis par le traitement

Le diagramme de classes ci-dessus est important. Il sert de base à l'adaptation automatique des modes d'emploi au contexte d'utilisation. Il sert aussi, sans préjuger de l'usage ultérieur, à recueillir de façon formalisée les connaissances relatives aux contextes d'utilisation des traitements. La modélisation proposée fournit un cadre utile à l'utilisateur devant spécifier son contexte. Elle fournit un cadre à l'auteur de traitement pour décrire les préconditions à l'utilisation de ce dernier. Elle fournit enfin un cadre à l'expert devant exprimer sa connaissance des actions à effectuer pour résoudre les problèmes d'inadéquation du contexte utilisateur au contexte requis par les traitements.

3.3.4 Règles de l'expert

Les règles de la figure 3.6 servent à recueillir et représenter toute une catégorie de connaissances que l'expert ne peut exprimer au moyen des éléments de descriptions des cinq facettes. L'expert indiquera par exemple que si une donnée est au format *shape*, alors la topologie n'y est pas représentée explicitement.

Exploitables informatiquement ou non, ces règles constituent des métadonnées utiles à décrire. L'expert les saisit si possible en utilisant les classes de notre modèle, en texte libre sinon. L'administrateur de l'application d'accès aux métadonnées décide ensuite ou non si elles doivent être traduites dans le langage opérationnel de règle. Si c'est le cas, la forme sous laquelle les règles ont été saisies rend en théorie possible l'automatisation complète du processus de traduction.

Les règles utiles à l'adaptation des modes d'emploi nous intéressent particulièrement. Certaines détectent les problèmes posés par le contexte de l'utilisateur ; d'autres permettent de déduire les instructions adaptées.

Il existe enfin une catégorie à part de règles qui ne sont pas destinées à être consultées, mais qui sont nécessaires au système pour répondre aux requêtes lors de la recherche de ressources ou lors de la mise en correspondance du contexte de l'utilisateur et du contexte du traitement. La règle *r_appartientLieuDeDev* décrite tableau 5.1 p. 196 en est un exemple.

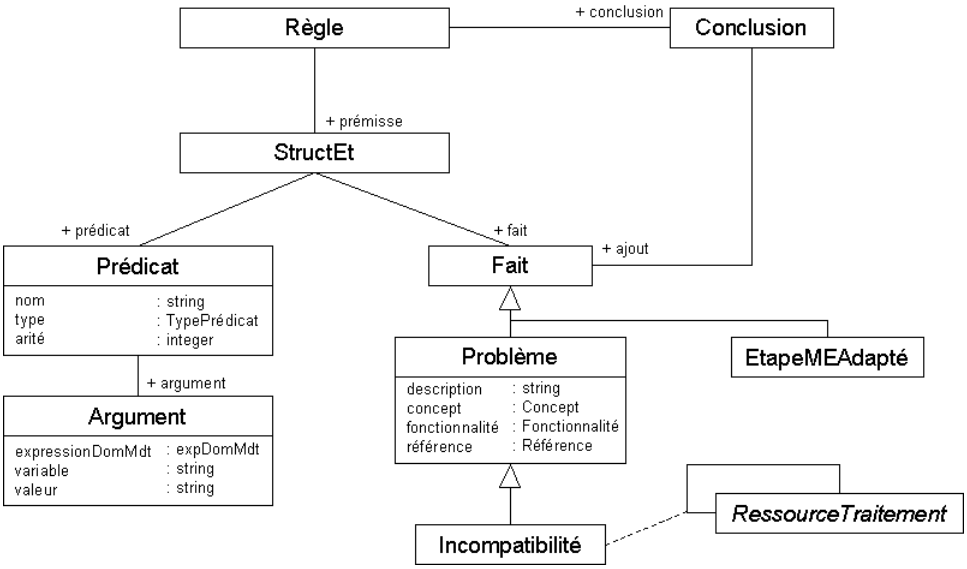


FIG. 3.6 – Règle d'adaptation de mode d'emploi

Nous n'abordons pas ici la modélisation de l'heuristique. Il s'agit d'un type de connaissance difficile à représenter ; nous nous résignons ici à recourir à la langue naturelle. Une heuristique étant une méta-règle, on peut néanmoins penser que le modèle de règle pourrait être utilisé si nous voulions contraindre le format des descriptions d'heuristiques.

Quant aux stratégies internes de déclenchement des règles (chaînage avant ou arrière, algorithme de graphes) ce sont là des questions d'implémentation ; elles seront abordées chapitre 5.

3.4 Conclusion

De l'idée d'un système d'information basé sur le modèle de métadonnées défini au chapitre 2, nous avons glissé vers celle d'un système à base de connaissances. Les exemples de besoins que nous avons sélectionnés nécessitent en effet la mise en œuvre de raisonnements. Deux types de langages ont été retenus pour représenter les connaissances d'expert de façon opérationnelle : celui des logiques de description pour définir les ontologies formelles et celui reposant sur les règles sous forme de clauses de Horn avec variables. Ces deux formalismes constituent des sous-ensembles presque toujours décidables de la logique du premier ordre – certaines LD expressives faisant exception. Le choix de ces langages de représentation des connaissances suppose d'effectuer un certain nombre d'hypothèses de travail que nous avons précisé : les raisonnements sont basés sur la déduction, sont monotones, et ne font appel ni à la logique floue ni aux opérateurs modaux. Nous avons également fait l'hypothèse d'un monde ouvert. Contrairement à certaines techniques d'IA tels que les réseaux de neurones ou aux algorithmes génétiques, les systèmes d'inférences que nous utiliserons ne sont pas des boîtes noires, au besoin la trace des raisonnements pourra être exhibée en guise d'explication[19].

Afin d'adapter les modes d'emploi des traitements, nous avons défini les classes permettant de représenter le contexte des utilisateurs et celui requis par les traitements. Le raisonnement consiste à détecter les adaptations nécessaires – *i.e.* à poser le diagnostic en quelque sorte –, puis à proposer à l'utilisateur un mode d'emploi contenant les instructions idoines. Dans un tel scénario, la complémentarité entre les ontologies et les règles apparaît : le raisonnement basé sur les ontologies permet l'évaluation des prémisses (par exemple en effectuant une classification qui détermine si le type de donnée de l'utilisateur correspond au type de donnée attendue par le traitement) ; le raisonnement reposant sur les règles permet de déduire les adaptations qui s'imposent.

Cette démarche est fort proche de celle du Web sémantique où doivent être rapprochés termes de requêtes et de description de ressources. C'est précisément pour cette raison que nous allons adopter les langages issus de ce domaine pour mettre en œuvre les principes qui viennent d'être exposés.

[19] Comme dans le système expert du domaine médical Mycin, où un module d'explication montre à la demande les règles utilisées pour poser le diagnostic.

Chapitre 4

Implémentation du modèle de métadonnées

Nous avons défini notre modèle conceptuel de métadonnées. Nous choisissons maintenant des langages informatiques pour encoder le modèle, et pour encoder les instances du modèle, *i.e.* les métadonnées elles-mêmes.

Reflet de la démarche générale suivie au cours de notre travail, le chapitre 4 traite des questions concernant l'implémentation de nos métadonnées selon le point de vue SI de l'application d'abord, selon le point de vue SBC de l'application ensuite.

Section 4.1, nous choisissons donc des langages pour construire des métadonnées structurées, conformes à notre modèle, persistantes et rendant aisée le développement de l'application Web présentée à l'utilisateur. Section 4.2, nous choisissons des langages de représentation des connaissances dotés d'une sémantique exploitable par des moteurs d'inférence. Nous justifions le choix de l'architecture duale SI/SBC caractérisant notre base de métadonnées section 4.3.

Nous montrons ensuite comment nous avons implémenté la base et le modèle de métadonnées : du point de vue SI section 4.4, du point de vue SBC section 4.5. Les sections 4.4 et 4.5 correspondent donc, respectivement, à la mise en œuvre des langages des sections 4.1 et 4.2.

4.1 Le choix de langages documentaires

Pour rendre persistantes les métadonnées de traitements, il nous faut nous doter d'un langage qui permette leur codage informatique. Ce langage est lui-même défini au moyen d'un langage de définition de langage, donc d'un méta-langage. C'est le couple formé par ce langage et ce méta-langage que nous allons maintenant choisir. Nous parlons pour cela de choix de langages documentaires car le but ici est de définir et contrôler la structure des métadonnées, pas de les doter d'une sémantique et d'en tirer les conséquences logiques.

Nous commençons par étudier le couple XML/XML Schema, basé sur le principe des langages à balises de la famille SGML. Le langage Topic maps, doté d'une syntaxe XML, apparaît comme un candidat pour représenter nos métadonnées ; nous en discutons. Les bases de données constituent une autre solution de stockage et de définition de structuration des données. Nous envisageons également une troisième solution : les langages de programmation permettent de définir des structures de données et offrent des mécanismes de stockage des instances.

4.1.1 Les langages à balises

GML/SGML et DTD

Le GML (Generalized Markup Language) est souvent cité comme un des premiers langages à balises. Apparu en 1969 chez IBM[1], le GML permet de définir des langages pour créer des documents structurés avec des balises. Il évolue peu de temps après en SGML (Standard GML). En 1986, SGML devient la norme ISO 8879. Ce sont ses descendants dédiés au monde du Web, HTML et XML principalement, qui connaîtront le succès le plus large.

Le principe de base du SGML est d'enrichir des documents textuels par des balises véhiculant des informations sur leur structure et leur contenu [Tro04]. Pour définir des modèles de documents, le SGML propose le méta-langage DTD (Document Type Definition). Les documents textuels balisés sont donc des instances de DTD.

Les DTD décrivent la structure des documents : la hiérarchie des balises, leur nom, leurs attributs. En revanche les DTD, entre autres limitations, ne permettent pas de contrôler finement le contenu des balises ni d'imposer des nombres quelconques d'occurrences de balises au sein d'autres balises. Les DTD ne permettent pas non plus l'importation de différents modèles de documents (car les espaces de noms ne sont pas supportés). Pour ces raisons, un autre langage de définition de modèle a été créé pour les documents structurés au format XML. Il s'agit de XML Schema que nous allons présenter et adopter.

XML et XML Schema

Le langage XML (eXtensible Markup Language) connaît depuis son apparition en 1996 un succès considérable. Recommandation W3C en 1998, XML est devenu un standard pour les documents structurés. Sans en exposer tous les détails, faisons-en une rapide présentation.

XML est un sous-ensemble de SGML. Un document XML est donc structuré en *éléments*. Les balises marquent le début et la fin de chaque élément. Les éléments peuvent contenir du texte et éventuellement d'autres éléments. L'ensemble des données du document XML est contenu dans un élément unique appelé *racine*, élément qui contient tous les autres éléments.

Le seul langage XML ne prescrit que la syntaxe minimale des documents (qui renvoie à la notion de document *bien formé*). En ce sens XML est un méta-langage servant de base aux langages[2] qui définissent des vocabulaires et des grammaires qui leur sont propres. Ainsi, par exemple, le langage XHTML définit les éléments légaux (head, title, body, etc.) et la façon dont ils peuvent s'agencer. Le langage HTML, qui est avec XML l'autre très célèbre descendant de SGML, offre une plus grande liberté dans l'utilisation des balises définies, ce qui rend certains documents HTML mal formés au regard de la syntaxe XML. Historiquement, la permissivité de la grammaire du HTML était en effet considérée, lors de sa création par Tim Berners-Lee, comme une condition du succès du Web et de sa "démocratisation" parmi les auteurs de pages non-informaticiens[3].

Le langage XML Schema est une recommandation du W3C depuis 1998 [W3C04f]. Au contraire des DTD, les schémas XML sont eux-mêmes des documents XML. Cette raison, en plus de celles évoquées précédemment, explique que les DTD tendent aujourd'hui à être remplacées par les schémas XML.

---

[1] GML correspond aux initiales de ses trois auteurs Charles Goldfarb, Edward Mosher et Raymond Lorie (`http://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language`).

[2] On parle aussi de "dialectes" XML ; c'est même un terme plus approprié puisqu'un dialecte est un sous-langage.

[3] Cette liberté a d'ailleurs eu des contreparties préjudiciables pour les navigateurs Web, d'où les différentes versions dont l'aboutissement est le XHTML, simple formulation de HTML 4 en XML 1.0 (du moins pour la version 1.0 du XHTML, puisque la XHTML 2.0 introduit de nouveaux éléments tels que XForms et XFrames).

Quelques-uns des avantages d'XML et XML Schema relativement à nos besoins...

Les principales raisons qui justifient le choix d'XML sont les suivantes :

- XML permet l'utilisation d'espaces de nommage (*namespace*). Un namespace est un URI, *i.e* une adresse. Un namespace pointe sur une collection de noms. Quand on utilise un nom dans un document, indiquer son namespace permet d'éviter les ambiguïtés avec d'éventuels homonymes [W3C04a].
- Il existe des langages et des outils rendant facile la manipulation des documents XML. XPath permet d'effectuer des requêtes. XSLT permet :
  - de produire le code HTML destiné à l'affichage sur le poste client,
  - de mettre en place des mécanismes d'importation et d'exportation entre la base de métadonnées conforme au schéma XML et la base de connaissance exprimée dans les langages que nous allons voir section 4.2,
  - de se réserver la possibilité d'importer ou d'exporter tous documents à la syntaxe XML (notamment ceux utilisant les langages RDF, OWL et SWRL que nous allons utiliser pour la partie SBC de notre application, ou aussi éventuellement les documents Topic Maps au format XTM).
- XML n'est lié à aucun système d'exploitation particulier. Les documents XML sont de simples fichiers texte lisibles pour un lecteur humain.

XML Schema est pour sa part un langage bien adapté à la traduction de notre modèle conceptuel :

- XML Schema intègre le principe d'héritage. Il fournit même des mécanismes plus riches que ceux habituellement rencontrés dans les langages à objets puisque les types d'éléments définis peuvent être étendus par extension ou par restriction.
- XML Schema permet l'expression de contraintes fines sur le contenu des éléments. Notamment, il est possible de définir des motifs à l'aide d'expressions régulières. Il est également possible de spécifier des contraintes non représentables en UML et plus simplement qu'avec OCL[4].
- La structure arborescente des éléments permet d'exprimer des inclusions implicites. Ce peut être un avantage par rapport aux bases de données relationnelles où pour une représentation équivalente plusieurs tables seraient nécessaires.
- XML Schema gère les espaces de nom et propose des mécanismes d'importation de schémas. Cela nous est notamment utile pour importer le schéma XML MathML2 définissant les éléments contenant les expressions mathématiques (cf. p. 168).
- Enfin, un dernier point important sur lequel nous reviendrons est la possibilité offerte par XML Schema d'exprimer l'ordre dans l'agencement des éléments. Cette notion d'ordre est absente d'autres langages de représentation basés sur les graphes comme RDF-S. Or il est clair, par exemple, que l'ordre des étapes de nos modes d'emploi est essentiel.

... et quelques inconvénients

La mise en œuvre de fonctionnalités telles que la gestion des accès concurrents est moins aisée dans le cas d'une base de données XML que dans celui d'une base de données relationnelle classique pour laquelle il existe de nombreux SGBD éprouvés. Cette situation est en train de changer avec l'apparition de ce qui est appelé les *bases de données XML natives*[5]. Ces dernières apportent une solution partielle à l'autre inconvénient des bases XML par rapport aux bases de données relationnelles : le temps et les ressources mémoires requis par le *parsing*. En effet, pour utiliser les langages XSL et XPath, il faut charger les documents XML dans des structures

[4]Les figures 2.16 et 2.17, p. 95 fournissent deux illustrations du problème. Le code 4.10 p. 166 montre une partie de notre modèle XML Schema où une contrainte lie le type d'un élément à la valeur d'un autre.

[5]`http://www.xmldb.org/projects.html`

de données DOM (Document Object Model) qui prennent en mémoire environ 10 fois la taille du document XML original. C'est beaucoup, mais non rédhibitoire dans notre contexte. Les documents XML peuvent être parsés de façon nettement plus économique avec la méthode SAX (Simple API for XML, dont le principe est un parcours séquentiel des documents, le programmeur spécifiant les opérations à effectuer en fonction des types d'éléments rencontrés), mais l'utilisation d'XPath et XSL est alors interdite.

Topic Maps

La norme Topic Maps (TM) est un modèle abstrait de métadonnées qui a pour but la représentation d'informations de type "index, thesaurus, table des matières et glossaire" [CCZC02]. L'origine du concept de TM remonte à 1993 [Top04]. Le modèle TM est devenu une norme ISO en 1999. En 2001 le consortium indépendant "TopicMaps.org"[6] a défini une représentation XML des TM ; le langage XTM était né.

Aujourd'hui ce langage connaît un certain succès dans la communauté du Web sémantique. Son intérêt a été étudié dans le cadre de plusieurs projets de capitalisation des connaissances en vue de satisfaire des besoins similaires aux nôtres. Par exemple [CCZC02] et [MABL03] ont fait le choix des Topic Maps pour représenter respectivement des catalogues de stages de formation informatique et des ressources pédagogiques pour l'enseignement de l'informatique (projet MEMORAe, *MEM*oire *OR*ganisationnelle *A*ppliquée au *e*-learning). En 2003, un stage au COGIT a eu pour sujet la création d'un catalogue de traitements de généralisation avec le langage XTM (cf. p. 78) : P. Michaux a créé un modèle TM et développé une application Web en permettant la consultation. Cette expérience concluante nous est utile pour situer les capacités des TM relativement aux besoins à présent plus larges qui sont les nôtres.

L'idée essentielle des TM est la suivante : étant donné un ensemble de ressources ou de description de ressources, on définit "par dessus" une couche d'indexation. Cette couche supérieure, c'est la carte des *topics* – un *topic* étant la réification d'un *sujet* qui est "toute chose qu'un homme peut concevoir" [Top04]. Autrement dit, les *topics* sont des termes d'indexation qui vont permettre la navigation au sein des TM.

"Les trois concepts clés des TM sont les *topics*, les *associations* et les *occurrences*" [Top04] :

- Les *topics* peuvent être vus comme des types de ressources. L'une des originalités des cartes topiques est la séparation des concepts et de leurs noms. Cela permet d'avoir plusieurs noms pour le même concept (et donc d'avoir des cartes topiques multilingues) et des noms partagés par plusieurs concepts [BCES04].
- Les *associations* entre *topics* décrivent leurs liens (ex : un traitement *réalise* une fonctionnalité).
- Les *occurrences d'un topic* sont des ressources contenant une information relative au dit *topic* (p.ex. le site Web d'une personne).

La figure 4.1 montre un exemple simple de TM appliqué à notre contexte ; le code 4.1 montre la façon dont cela se traduit en langage XTM.

La spécificité et l'originalité des TM résident dans des principes qui ne sont pas tous illustrés dans cet exemple. Avec les TM, il est notamment possible :

- d'exprimer la relation de subsumption (associations instances de `"supertype-subtype"` comme illustré dans [MABL03], §4.4.2 où le *topic* "book" spécialise celui de "document"),

[6] `http://www.topicmaps.org/`

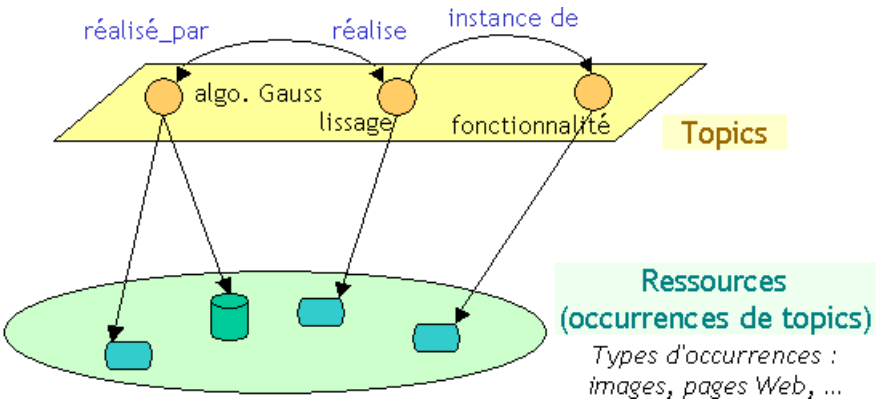


FIG. 4.1 – Exemple de Topic Maps pour la description de traitements

```
<topicMap xml:base="http://mycorp.com/xtm/mymap2.xtm"
          xmlns="http://www.topicmaps.org/xtm/1.0/"
          xmlns:xlink="http://www.w3.org/1999/xlink">

  <topic id="algorithmeGauss">
    <instanceOf>
      <topicRef xlink:type="simple" xlink:href="#algorithme"/>
    </instanceOf>
    <baseName>
      <baseNameString>algorithme de Gauss</baseNameString>
    </baseName>
    <occurrence>
      <resourceRef xlink:type="simple"
          xlink:href="http://www.ensg.ign.fr/CDOC/CDOC_PDF/Bull_art_12.pdf"/>
    </occurrence>
  </topic>

  <topic id="realise">
    <baseName>
      <baseNameString>réalise</baseNameString>
    </baseName>
  </topic>

  <association id="GaussRealiseLissage">
    <instanceOf>
      <topicRef xlink:href="#realise"/>
    </instanceOf>
    <member>
      <roleSpec>
        <topicRef xlink:href="#algorithme"/>
      </roleSpec>
      <topicRef xlink:href="algorithmeGauss"/>
    </member>
    <member>
      <roleSpec>
        <topicRef xlink:href="#fonctionnalite"/>
      </roleSpec><e
      <topicRef xlink:href="lissage"/>
    </member>
  </association>
</topicMap>
```

Extrait de code 4.1: XTM – Exemple de Topic Maps pour la description de traitements

– de typer les relations d'occurrences (une ressource relative à un *topic* peut être un livre, une image, une page Web, un document vidéo, etc.),
– de fusionner des TM de sources différentes.

La question de l'intérêt d'une adoption éventuelle des TM pour nos métadonnées mérite d'être posée. Elle concerne seulement le choix du langage de la partie SI de notre travail – ce que nous appelons le langage documentaire –, et non celui du langage de la partie SBC. En effet, si les TM peuvent être qualifiés de langage de représentation des connaissances, il ne n'agit pas de représentation *opérationnelle*. Un moteur d'inférence pourrait peut-être exploiter les relations de subsumption représentables avec les TM – et encore ce n'est sans doute pas là la vocation du langage –, mais les raisonnements possibles seraient de toutes façons très loin de ceux permis par les langages tels ceux basés sur les LD.

Ainsi, pour répondre à la question "Les Topic Maps sont-ils un bon candidat pour l'ingénierie du Web sémantique ?", Caussanel *et al.* ont été amenés à distinguer le *Web sémantique cognitif* et le *Web sémantique computationnel*

> "Il existe deux tendances en partie divergentes dans les approches du Web sémantique. La première, qui nous semble relever d'un "web computationnellement sémantique", vise essentiellement à automatiser la recherche d'information via des agents logiciels. Selon cette perspective, il est nécessaire que les modèles de connaissances représentant les documents, ou plus largement les domaines de connaissance, soient exprimés de la manière la plus formelle possible. Selon la seconde, qui nous semble relever d'un "web cognitivement sémantique", la structuration des contenus, si elle peut permettre une semi-automatisation de certaines tâches, vise tout autant à accroître l'intelligibilité du Web pour des utilisateurs humains engagés dans des pratiques de navigation et d'enrichissement des contenus." [CCZC02]

La conclusion de Caussanel *et al.* est que les TM répondent aux besoins du *Web sémantique cognitif* mais pas du *Web sémantique computationnel*. La question pour nous est donc de savoir si les TM répondent à nos objectifs liés à l'aspect "SI" de notre application.

Selon notre point de vue, choisir les TM pour exprimer nos métadonnées présenterait l'inconvénient de la lourdeur pour le développement de notre application. Cela est dû au manque de hiérarchie du langage. Il est bien possible de définir des *topics* correspondant aux classes de notre modèle d'une part, et de définir des *topics* et des occurrences correspondant à notre base de métadonnées d'autre part, mais sans possibilité, nous semble-t-il, de regrouper au niveau du code XTM les informations relatives à une même ressource au sein d'une même entité. Cette caractéristique, citée notamment par Le-Grand et Soto comme une limite des TM ([LGS02], diap. 11), peut être contournée et ne concerne *a priori* pas l'utilisateur final. Mais comparativement, une base de métadonnées au format XML *ad hoc* est plus aisément manipulable. De plus, du point de vue du contrôle de la structure et du contenu, le langage XTM n'offre pas les possibilités du langage XML Schema.

A contrario, les TM présentent l'avantage de permettre la construction d'applications offrant à l'utilisateur des qualités appréciables en terme de liberté de navigation. Il est par exemple intéressant de pouvoir considérer les associations entre *topics* comme des instances de *topics*[7] (dans notre exemple, l'association "algorithme Gauss réalise la fonctionnalité lissage" est instance du *topic* "réalise"). Ceci dit, les possibilités de navigation ou de requêtes ouvertes par cette propriété des TM ne nous semblent pas spécialement difficiles à mettre en œuvre avec un autre choix de langage. Nous verrons ainsi, par exemple, que notre application permet l'accès aux propriétés des entrées des traitements à la fois en tant que ressources et en tant qu'associations entre ressources (cf. fig. 5.5 p. 185). De même, la mise en relation de *topics* partageant de mêmes occurrences n'est pas l'apanage des TM. Ce qui fait la force de ces

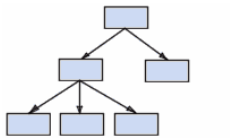
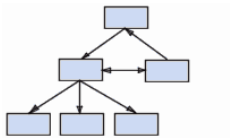
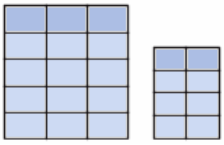
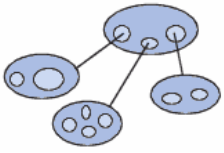
[7] Dans le même ordre d'idée, une assertion ("Accordéon réalise la fonctionnalité lissage") peut être une ressource en RDF.

derniers, c'est qu'une fois fixée la façon de décrire *topics* et occurrences, des outils standards peuvent être proposés. Omnigator[8], de la société Ontopia, est ainsi par exemple un logiciel de visualisation de TM assez agréable à utiliser. Mais nos besoins spécifiques de consultation nécessitent le développement de notre propre application (de notre progiciel, pourrait-on dire). Dans l'hypothèse du choix des TM, plus intéressant pour nous est l'existence d'un langage de requête comme TMQL (Topic Map Query Language). Mais comparativement, le langage XPath convient également à nos besoins.

Après étude, nous n'avons trouvé d'arguments décisifs ni en faveur des TM, ni en leur défaveur. Il nous semble néanmoins globalement que, dans notre contexte, les inconvénients l'emportent sur les avantages comparativement au choix du couple XML/XML Schema.

4.1.2 Les bases de données

Au sens large, une base de données est une collection de données structurées. Quand une base de données atteint une taille conséquente et doit être consultée ou modifiée par plusieurs utilisateurs, le recours à un Système de Gestion de Bases de Données (SGBD) devient nécessaire. Un SGBD comprend la base de données, le schéma de base de données, et les programmes qui permettent de manipuler ces derniers. Les premiers SGBD datent des années 60. Il existe plusieurs modèles de SGBD. Le tableau 4.1 montre les quatre principaux.

le **modèle hiérarchique** : les données sont classées hiérarchiquement, selon une arborescence descendante. Ce modèle utilise des pointeurs entre les différents enregistrements. Il s'agit du premier modèle de SGBD. 	le **modèle réseau** : comme le modèle hiérarchique ce modèle utilise des pointeurs vers des enregistrements. Toutefois la structure n'est plus forcément arborescente dans le sens descendant. 
le **modèle relationnel** : les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes). La manipulation de ces données se fait selon la théorie mathématique des relations. 	le **modèle objet** : les données sont stockées sous forme d'objets, c'est-à-dire de structures appelées classes présentant des données membres. Les [valeurs] des champs sont des instances de ces classes. 

TAB. 4.1 – Différents modèles de SGBD (d'après [Pil03][9])

L'avantage des SGBD sur les autres solutions de stockage est de permettre l'exécution de requêtes sur de très grandes quantités de données. Les inconvénients résident dans l'absence des qualités du langage XML évoquées plus haut. La piste des BD XML natives peut être une solution ; dans notre contexte il n'est pas apparu indispensable de la mettre en œuvre. De

[8] `http://www.ontopia.net/omnigator/`

[9] Ce document intitulé "Bases de données – Modèles de SGBD" issu de `http://www.commentcamarche.net` est mis à disposition sous les termes de la licence Creative Commons. Vous pouvez copier, modifier des copies de cette page, dans les conditions fixées par la licence, tant que cette note apparaît clairement.

façon intermédiaire, une solution simple pour concilier les avantages d'un SGBD et du format XML est d'éclater les métadonnées au format XML en plusieurs fichiers identifiés par leur *id*, et stockés dans les champs BLOB (Binary Large OBject) d'une base de données relationnelle. Pour préserver une partie de la puissance du requêtage XPath, des fichiers inversés XML sont générés avec XSL. Ce type d'architecture existe actuellement dans des SI du domaine bancaire. Nous ne l'avons pas mis en œuvre ici mais nous savons qu'il n'existe pas d'obstacle technique à le faire.

Même si, en fin de compte, la solution que nous avons adoptée ne recourt pas à l'usage d'un SGBD, il est intéressant d'en étudier les différents modèles. Nous appliquons en effet un certain nombre de principes qui en sont issus :

- Comme dans le modèle relationnel, nous regroupons les ressources principales de notre modèle dans des tables (cf. code 4.5 p. 161). Cela participe à la fois à la clarté de l'organisation et à l'efficacité des requêtes XPath, lesquelles peuvent ainsi porter sur des ensembles restreints d'éléments XML.
- Nous mettons également en œuvre la notion de référence entre éléments de métadonnées, notion similaire à celle des pointeurs du modèle réseau. Un des principes généraux des SGBD est d'éviter la redondance : on ne répète pas une information, on lui attribue un identifiant et on y fait référence.
- Comme dans le modèle objet, nos métadonnées font appel aux mécanismes d'héritages.

Un de nos objectifs étant la représentation des connaissances pour dériver de l'information non explicitement présente dans la base de données, il est naturel qu'en complément du tableau 4.1 nous évoquions les SGBD déductifs. Nous ne nous sommes pas tournés vers ce type de solution car pour représenter les ontologies et les règles d'expert nous avons choisi d'adopter les langages standards du W3C. C'est ce que nous allons voir section 4.2. Dès lors, nous nous écartons de l'hypothèse de l'utilisation d'un SGBD déductif qui n'aurait pas permis de mener certaines inférences propres aux logiques de description.

4.1.3 Les structures de données des langages de programmation

Les langages de programmation permettent en général de définir des structures de données, plus ou moins complexes. Certains langages offrent de plus la possibilité de rendre persistantes les instances desdites structures de données. En l'occurrence, puisque notre modèle conceptuel a été défini au moyen de diagrammes de classes UML, l'éventualité de le traduire en Java était à examiner[10]. Deux moyens courants de rendre persistants des objets Java sont l'emploi des mécanismes associés à l'interface `java.io.Serializable` et la réalisation d'un *mapping objet-relationnel*, donc le couplage avec un SGBD.

Chacune de ses deux solutions pose des difficultés : gestion des accès concurrents dans le cas de la sérialisation Java, lourdeur de l'opération de chargement dans le cas du *mapping* objet-relationnel. Mais, en fait, la question de l'adoption d'une de ses solutions ne se pose guère car l'exploitation escomptée de nos métadonnées – une exploitation documentaire, en particulier le requêtage et la génération de pages HTML – est éloignée de la vocation d'un langage de programmation comme Java. Il en aurait été autrement si, par exemple, nous avions voulu invoquer les traitements décrits. Mais cela ne faisait pas partie de nos objectifs. En comparaison d'une solution Java, faire reposer une application Web de type SI sur un principe d'échange de documents XML nous apparaît au final bien plus adaptée.

[10] Des outils de génie logiciel comme Eclipse permettent la génération automatique de canevas de classes Java à partir des diagrammes de classes UML.

4.1.4 Conclusion : le choix XML/XML Schema

Cette section avait pour but d'apporter réponse à deux questions liées : celle du choix d'un (méta)langage documentaire, et celle du choix d'une solution technique de stockage.

Nous faisons le choix du couple XML/XML Schema. Il est adapté à nos objectif O4 (extensibilité du modèle), O7c (contrôle de la structure de la base de métadonnées, et, dans une certaine mesure, du contenu), O7d (expression de la permissivité vis-à-vis des descriptions incomplètes grâce à la spécification des cardinalités minimum et maximum des éléments permise par XML Schema), O8a, O8b, O8c et O8d (existence de langage et d'outils permettant l'expression de requêtes et les manipulations aisées telles que la génération de HTML – ici XPath et XSL satisfont ces attentes).

Nous utilisons donc les langages recommandés par le W3C tant pour implémenter le modèle et la base de métadonnées que pour développer l'application qui en permet l'accès aux utilisateurs. Ces langages fortement liés au Web datent de moins d'une dizaine d'années. Ce sont maintenant des standards largement adoptés dans le monde de l'entreprise pour le développement de systèmes d'informations déployés en environnement Web client/serveur. Ils sont appelés à cohabiter avec les solutions basées sur les SGBD traditionnels qui restent précieux pour les questions de gestion d'accès concurrents et de gestion de gros volumes de données.

Nous disposons maintenant d'un langage documentaire qui va nous servir à créer une base de données instanciant notre modèle. Nous allons pouvoir construire un système d'information reposant sur ladite base.

4.2 Le choix de langages de représentation des connaissances

Le second volet de notre travail, construire un système à base de connaissances, reste à réaliser. Pour cela, il nous faut nous doter de langages de représentation des connaissances : langages pour définir les ontologies et exprimer des assertions d'abord (en 4.2.1), langage pour exprimer les règles ensuite (en 4.2.2). Ce faisant, nous progressons dans l'ascension du *semantic Web layer cake* que nous allons commencer par présenter.

4.2.1 Quelques mots sur le Web sémantique

Pour décrire efficacement les ressources disponibles sur le Web, il faut utiliser des langages de *représentation des connaissances*. C'est l'ambition du Web sémantique. On y trouve des problématiques similaires à celles que nous rencontrons dans le contexte des besoins de métadonnées sur les traitements géographiques. Au département STIC[11] du CNRS, une action de recherche est dédiée au Web sémantique. Empruntons à ses membres cette présentation de leur objet d'étude :

> "L'expression Web sémantique, attribuée à Tim Berners-Lee [BHL01] au sein du W3C, fait d'abord référence à la vision du Web de demain comme un vaste espace d'échange de ressources entre êtres humains et machines permettant une exploitation, qualitativement supérieure, de grands volumes d'informations et de services variés. Espace virtuel, il devrait voir, à la différence de celui que nous connaissons aujourd'hui, les utilisateurs déchargés d'une bonne partie de leurs tâches de recherche, de construction et de combinaison des résultats, grâce aux capacités accrues des machines à accéder aux contenus des ressources et à effectuer des raisonnements sur ceux-ci.
>
> Le Web sémantique, concrètement, est d'abord une infrastructure pour permettre l'utilisation de connaissances formalisées en plus du contenu informel actuel du Web, même si

[11] Sciences et Techniques de l'Information et de la Communication.

> aucun consensus n'existe sur jusqu'où cette formalisation doit aller. Cette infrastructure doit permettre d'abord de localiser, d'identifier et de transformer des ressources de manière robuste et saine tout en renforçant l'esprit d'ouverture du Web avec sa diversité d'utilisateurs. Elle doit s'appuyer sur un certain niveau de consensus portant, par exemple, sur les langages de représentation ou sur les ontologies utilisées" [LRC02].

Le diagramme qui résume les principes sur lesquels repose le Web sémantique est connu sous le nom de *semantic Web layer cake* (fig. 4.2).

Du bas vers le haut, on y voit la graduation progressive du niveau symbolique au niveau connaissance :

- Le standard **Unicode** définit des ensembles des caractères constituant des alphabets, par exemple l'ensemble UTF-8 où chaque caractère est codé sur 8 bits. Le protocole **URI** (Uniform Resource Identifier) permet d'identifier et d'adresser de façon unique les ressources[12].
- **XML**, **XML Schema** et le mécanisme de **namespace** permettent de construire des documents structurés et de définir leur syntaxe. Cette couche du *layer cake* permet d'implémenter des modèles de métadonnées. Les langages de cette couche sont des langages documentaires sur lesquels reposent – selon l'acception que nous avons du terme – les *systèmes d'informations*.
- À partir de la couche **RDF + rdfschema** apparaissent les langages de représentation des connaissances. Les notions de sémantique et de conséquence logique sont introduites.
- Au niveau de la couche **Ontology vocabulary**, les concepts de domaines de connaissances peuvent être représentés dans des ontologies formelles avec les langages OWL. Nous allons voir que nous utilisons OWL-DL.
- Les couches **Proof** (preuve) et **Trust** (confiance) constituent des objectifs non encore totalement atteints par la communauté du Web sémantique. Les moteurs d'inférence exploitant les capacités des langages des couches inférieures ne sont pas encore achevés et les règles permettant d'accorder un certain niveau de confiance dans les connaissances représentées restent pour partie à définir (questions d'authentification notamment, certains parlent de *Web of trust*).

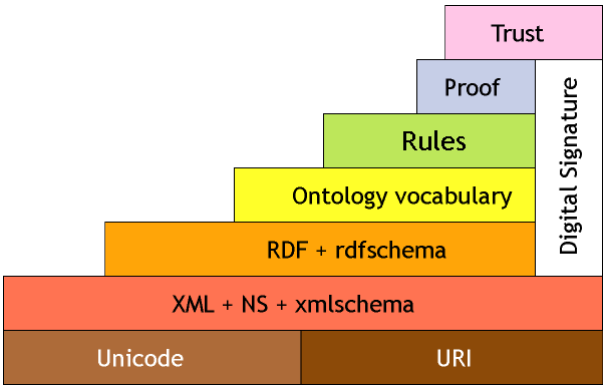


FIG. 4.2 – The Semantic Web "layer cake" (adapté de [BM02][13])

[12] Nous avons déjà évoqué le sujet des URI p. 8. Davantage d'informations se trouvent sur `http://www.w3.org/Addressing/`.

[13] Il existe plusieurs variantes légèrement différentes du *semantic Web layer cake*. Dans celle qui est présentée ici, nous avons pris la liberté d'apporter une modification minime en remplaçant l'intitulé de la couche "logic" par "rules" afin de mieux montrer le parallèle avec notre démarche (il se trouve d'ailleurs que dans une autre version répandue du *SW layer cake* les deux couches "rules" et "logic" apparaissent de façon distincte pour signifier qu'en fait la couche des règles repose sur celle de la logique).

4.2.2 Langages pour exprimer des assertions et définir des ontologies

Parmi les langages permettant d'exprimer des assertions et de définir des ontologies, certains semblent aujourd'hui faire à peu près consensus au sein de la communauté du Web sémantique. Ces langages sont recommandés par le W3C. C'est ceux-là que nous adoptons. Il s'agit de OWL (Ontology Web Language) et de RDF (Resource Description Framework). OWL permet de définir des ontologies. RDF permet d'exprimer des assertions, notamment en utilisant les concepts et relations des ontologies OWL.

RDF et RDFS

RDF est un modèle, associé à une syntaxe, dont le but est de permettre à une communauté d'utilisateurs de partager les mêmes métadonnées pour des ressources partagées. Il a été conçu initialement par le W3C pour permettre de décrire l'information accessible sur le Web.

RDF n'est pas particulièrement conçu pour permettre de stocker les métadonnées de documents mais plutôt pour permettre leur échange et leur traitement par des agents humains ou logiciels. Un des gros avantages de RDF est son extensibilité, à travers l'utilisation des schémas RDF qui peuvent s'intégrer et ne s'excluent pas mutuellement grâce à l'utilisation du concept d'espace de nom (*namespace*).

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns ="http://www.ign.fr/mdt">
  <rdf:Description about="prgAccordeon">
    <realise rdf:resource="#fctCaricature">
  <rdf:Description/>
  <! ... >
<rdf:RDF
```

Extrait de code 4.2: RDF – Exemple d'assertion

RDF permet d'exprimer des assertions, mais pas de définir un vocabulaire de termes et de relations. RDFS (RDF Schema) comble ce manque. RDFS permet de définir des classes (`rdfs:Class`), des sous-classes (`rdfs:subClassOf`), des sous-propriétés (`rdfs:subPropertyOf`), etc. [W3C04e]. RDFS apparaît néanmoins insuffisant pour définir des ontologies. En effet, en RDFS il n'y a pas de distinction entre les classes et les instances et on ne dispose pas de la possibilité d'indiquer des contraintes sur un domaine, pas plus que sur les cardinalités ou bien encore de préciser qu'une propriété est transitive, inverse ou symétrique. L'ensemble de ces manques fait de RDF Schéma un support insuffisant pour répondre aux exigences du Web Sémantique [Per04]. OWL, au contraire, possède les qualités requises ; c'est pourquoi il supplante RDFS.

OWL

OWL (Ontology Web Langage) est un langage de définition d'ontologies destiné, en particulier, à décrire les ressources du Web. Il a le statut de recommandation W3C depuis février 2004.

OWL permet de formaliser un domaine en définissant des classes et leurs propriétés. Comme RDF, OWL est doté d'une sémantique formelle en théorie des modèles [EB04]. La sémantique formelle indique comment déduire les conséquences logiques d'une ontologie, c'est-à-dire les faits qui ne sont pas littéralement présents dans l'ontologie mais peuvent être déduits par la sémantique formelle [W3C04c]. Concrètement, la sémantique formelle d'un langage comme

OWL peut être vue comme un ensemble de règles génériques de raisonnement (par exemple transitivité de la relation de subsumption). Il est ainsi possible de mener des raisonnements sur les classes et les individus : raisonnements terminologiques et assertionnels. R. Costello donne l'exemple des assertions *"Kamehameha est né à Hawaii"* et *"Kamehameha est né dans l'état d'Aloha"*. Un moteur d'inférence OWL peut alors déduire que Hawaii et l'état d'Aloha représentent le même lieu (en supposant que l'ontologie indique qu'une personne ne possède qu'un unique lieu de naissance ; l'exemple ici n'évoque pas la possibilité que les lieux soient liés par une relation d'inclusion) [Cos03].

OWL est le successeur de DAML+OIL, langage issu de la collision entre DAML et OIL[14] et s'inspirant donc des principes de *frames* et de logique de description. OWL peut-être vu comme une extension de RDFS, mais auquel on aurait enlevé des propriétés telle que la possibilité de traiter les assertions comme des ressources [Tro04].

OWL fournit un grand nombre de constructeurs permettant d'exprimer de façon très fine les propriétés des classes définies[15]. La rançon de cette expressivité est l'indécidabilité du langage obtenu en considérant l'ensemble de ces constructeurs. C'est pour cela que OWL a été fractionné en trois langages distincts, d'expressivité croissante :

- Le langage **OWL Lite** concerne les utilisateurs qui ont principalement besoin d'une hiérarchie de classifications et de fonctionnalités de contraintes simples. Par exemple, OWL Lite ne permet que des valeurs de cardinalité de 0 ou 1.
- Le langage **OWL DL** (Description Logic) concerne les utilisateurs qui souhaitent une expressivité maximum sans perdre la complétude du calcul (toutes les inférences sont assurées d'être prises en compte) et la décidabilité (tous les calculs seront terminés dans un intervalle de temps fini) des systèmes de raisonnement. Le langage OWL DL inclut toutes les structures de langage de OWL, avec des restrictions comme la séparation des types (une classe ne peut pas aussi être un individu ou une propriété, une propriété aussi être un individu ou une classe). OWL DL est ainsi nommé en raison de sa correspondance avec la logique descriptive possède des propriétés de calcul avantageuses pour les moteurs d'inférences.
- Le langage **OWL Full** est destiné aux utilisateurs qui souhaitent une expressivité maximale et la liberté syntaxique de RDF sans garantie de calcul. Par exemple, dans OWL Full, une classe peut se traiter simultanément comme une collection d'individus et comme un individu à part entière. Une autre différence significative par rapport à OWL DL réside dans la possibilité de marquer un objet `owl:DatatypeProperty` comme étant un objet `owl:InverseFunctionalProperty`. Il est peu probable qu'un système de raisonnement puisse mettre en œuvre toutes les caractéristiques de OWL Full.

Chacun de ces sous-langages représente une extension par rapport à son prédécesseur plus simple, à la fois par ce qu'on peut exprimer légalement et par ce qu'on peut conclure de manière valide. Pour notre part, nous utiliserons le langage OWL-DL.

D'autres variétés de langages OWL ont aussi été proposées par des auteurs cherchant des compromis particuliers entre expressivité et qualités calculatoires. Par exemple Grosof *et al.* ont définit OWL DLP en 2003 (plus simple qu'OWL Lite, OWL DLP peut être traduit en Datalog)[GHVD03], et ter Horst a défini OWL Horst en 2005 en introduisant l'usage de règles [tH05]. Une autre variété encore de OWL a été proposée par Comte *et al.*. Il s'agit de OWL-SG (OWL Simple Graphs). Le but est d'exploiter les qualités des graphes conceptuels perdues par l'approche logique de description, tels la possibilité de manipuler des nœuds anonymes (*blank*

[14] cf. la signification des acronymes page 63. Des détails sur DAML+OIL et la filiation avec OWL peuvent être trouvés dans [Tro04], p.76.

[15] Cette partie a été rédigée à partir de [EB04] et [W3C04c] §1.1

nodes) et la possibilité d'exprimer des requêtes conjonctives [CL05] (dont la définition du concept "ami" comme "ennemi d'un ennemi" pourrait être un exemple). Le langage de règles que nous allons utiliser, SWRL, peut aussi être considéré comme une extension de OWL.

Un des buts d'OWL est d'améliorer les performances des moteurs de recherche du Web qui ne reposent que sur le principe des mots-clés. On trouve ainsi dans la littérature du Web sémantique des exemples de requêtes semblables à celui que nous avons exposé pour illustrer le besoin de raisonnement ER 1. Par exemple, R. Costello montre comment franchir le fossé (*terminology gap*) qui sépare les termes de l'utilisateur et ceux de la description de l'appareil photo qu'il recherche [Cos03]. Le guide d'OWL publié par le W3C donne également l'exemple *"Dites-moi quels vins acheter pour accompagner chaque plat du menu suivant. Et, à propos, je n'aime pas le Sauterne"*. Appliqué à notre contexte, cela pourrait donner : *"Dites-moi quelles API utiliser pour développer un visualisateur de MNT. Et, à propos, je ne veux pas de problème de compatibilité avec GeOxygene"*. Un autre exemple d'application des inférences permises par OWL, dans le domaine de la détection du signal en pharmacologie cette fois, est la réorganisation automatique de concepts dont on cherche à établir la proximité sémantique [BCLJ04].

Les applications potentielles d'OWL sont en fait très nombreuses. Le W3C suggère plusieurs cas d'utilisation. Certains sont particulièrement proches de notre contexte. Ainsi, le cas d'utilisation *"administration d'un site Web d'entreprise"* est présenté comme suit :

> Un site Web mettant en oeuvre des ontologies [W3C04b] est susceptible d'intéresser :
> – (...)
> – Un technicien recherchant les poches d'expertise technique particulière et les détails de l'expérience acquise ;
> – Un chef de projet recherchant dans l'expérience acquise et les modèles passés afin de soutenir un projet à phases multiples complexe, à la fois pendant la phase de proposition et lors de l'exécution.
>
> Un problème habituel pour chacun de ces types d'utilisateur est celui selon lequel ils ne peuvent pas partager de terminologie avec les auteurs des contenus demandés. Le vendeur peut ignorer le terme technique pour une caractéristique souhaitée, ou des techniciens dans différents domaines de compétence peuvent employer des termes différents pour le même concept. Pour de tels problèmes, il serait utile que chaque classe d'utilisateurs dispose de différentes ontologies de termes, mais que chaque ontologie soit inter reliée, de sorte que des traductions puissent être effectuées automatiquement [W3C04c].

"La documentation d'un concept" et *"les agents et les services"* sont deux autres cas d'utilisation qui sont en rapport étroit avec l'exploitation que l'on souhaite faire de notre base de métadonnées de traitements. Concernant les services, nous avons souligné au chapitre 2 l'intérêt d'OWL-S pour notre projet. Dans le même esprit, le cas d'utilisation nommé *l'informatique omniprésente* :

> On utilisera OWL afin de décrire les caractéristiques des appareils, les moyens d'accéder à ces appareils, la politique d'usage d'un appareil établie par son propriétaire et les autres contraintes techniques et conditions qui régissent l'insertion d'un appareil dans un réseau de type informatique omniprésente [W3C04c].

4.2.3 Langages de règles

Nous devons maintenant choisir un langage opérationnel de logique du premier ordre pour exprimer les règles avec variables dont on a montré le besoin au chapitre 3. Un certain nombre des systèmes expert et systèmes à base de connaissances existant utilisent par exemple les langages CLIPS, Prolog et Datalog.

Avant d'envisager l'emploi d'un de ces langages, rappelons qu'une particularité de notre contexte est la nécessaire cohabitation des aspects SI et SBC de notre application. Les règles de l'expert sont avant tout des métadonnées comme les autres. Elles sont donc stockées

dans notre base de métadonnées XML sous une forme déclarative. Un critère à prendre en compte pour le choix du langage de règles est la facilité à passer du format déclaratif au format opérationnalisable. Concilier les formes déclarative et procédurale des règles est délicat ; c'est un problème classique représentation des connaissances. Nous allons essayer d'en proposer une solution dans le contexte de notre travail. Même si nous cherchons à l'éviter, nous n'écartons pas l'hypothèse d'une intervention humaine pour transformer les règles sous une forme opérationnalisable.

CLIPS et Prolog

Dans un contexte très similaire au nôtre, Gandon et Sadeh, à partir d'une "base de faits" constituée de descriptions RDF et d'une ontologie OWL d'une part, et d'une "base de règle" reposant sur les langages ROWL, WOWL, SOWL et QOWL[16] d'autre part, ont choisi de générer du CLIPS avec des feuilles XSLT. Le but était d'utiliser un moteur d'inférence JESS pour déduire de la base de faits les droits d'accès à l'information d'utilisateurs interrogeant des services Web fournissant des informations sur le personnel d'une université. Autrement dit, il s'agissait de mettre en place un système de gestion de la confidentialité [GS04].

CLIPS (C Language Integrated Production System)[17] est un langage créé en 1985 par la section IA de la Nasa ; le nom désigne aussi un environnement pour le développement de systèmes experts. Il permet d'assigner des priorités aux règles, ce qui pourrait être intéressant par exemple pour traduire les préférences dans les stratégies de réponse au besoin de l'utilisateur comme dans le raisonnement ER 3. Par contre dans le cas de raisonnements comme ER1, il s'agit juste de saturer la base de faits et l'ordre des règles n'importe pas. Quoiqu'il en soit, les connaissances heuristiques figurent bien dans notre modèle de métadonnées, mais nous avons renoncé à les rendre opérationnalisables.

Une autre caractéristique qui fait de CLIPS un langage potentiellement intéressant pour nous réside dans l'existence du moteur d'inférence très répandu Jess (Java Expert System Shell). Si nous décidions de charger nos métadonnées XML dans des classes Java, nous pourrions utiliser les objets Java dans les règles CLIPS. Une telle architecture a été mise en œuvre notamment par D. Shereen dans le cadre d'une base de règles pour l'évaluation de la cohérence entre bases de données géographiques [She05]. Un avantage non négligeable dans notre contexte de ce choix résiderait dans la facilité d'adressage des éléments de notre base de données. Nous reviendrons sur cette question d'adressage p. 173.

Prolog (*pro*grammation *log*ique) est un autre langage de règles très répandu. Il a été créé en 1972 par A. Colmerauer et P. Roussel. Les nombreux moteurs d'inférences des différentes versions de Prolog fonctionnent en chaînage arrière. Certains développeurs ont choisi, dans le cadre de leurs travaux, d'exporter leur base de faits et de règles au format XML en Prolog. Par exemple, Khayati *et al.* transforment avec XSLT leurs fichiers XMI[18] en formules Prolog. L'idée globale est de représenter les signatures des composants et leurs fonctionnalités avec des prédicats Prolog, puis de lancer les règles de production pour obtenir comme nouveaux faits les informations sur l'interopérabilité desdits composants [KFG05]. On retrouve les mêmes calculs d'héritage de types de données que ceux nécessités par notre exemple ER1.

CLIPS et Prolog font l'hypothèse du monde fermé : un fait absent de la base est supposé faux. En ce qui nous concerne nous avons au contraire fait l'hypothèse du monde ouvert (cf. p.

[16] Extensions de OWL utilisés par [GS04] pour exprimer différents types de règles : déduction de faits nouveaux, invocation de services Web – on remarque donc que dans ce contexte, contrairement au notre, le lien entre SBC et procédures externes est établi –, règles de confidentialité, et requêtes

[17] `http://www.ghg.net/clips/WhatIsCLIPS.html` (site officiel de CLIPS).

[18] XML Metadata Interchange (XMI) est un standard d'échange de données UML basé sur XML. XMI est un standard de l'OMG.

134). Ce n'est cependant pas pour ce motif que nous n'adoptons ni CLIPS ni Prolog, car nous n'avons pas mis en œuvre de cas où cette hypothèse intervenait (par exemple en signalant à l'utilisateur la nuance entre description incomplète d'un SIG ou négation explicite concernant ses propriétés).

L'élément déterminant dans notre choix du langage de règles est en fait notre volonté de coupler les raisonnements terminologiques propres aux logiques de description et les raisonnements propres aux règles avec prédicats et variables dites "de Horn". Or ni CLIPS ni Prolog ne permettent de mener les raisonnements propres aux logiques de description. C'est justement parce que l'expression des inférences de LD en logique de Horn n'est pas toujours possible que certains auteurs ont voulu considérer le sous-ensemble de la logique du premier ordre nommé DLP (Description Logic Programming), et défini comme l'intersection de DL et de Horn. Dans le prolongement des différences exhibées p. 130, Stoutenburg *et al.* énumèrent les raisons pour lesquelles DL et Horn ne permettent pas les mêmes types d'inférences (§4.3 de [SON+04]).

L'autre critère qui intervient dans notre choix du langage de règle est de continuer à utiliser les standards du Web sémantique. De façon liée, nous souhaitons adopter une syntaxe XML pour l'encodage de nos règles afin de faciliter les imports/exports entre SI et SBC, grâce à XSLT.

Pour les raisons évoquées ci-dessus, nous adoptons le langage de règle du Web sémantique SWRL.

SWRL

SWRL (Semantic Web Rule Language) est un langage de règles du Web sémantique combinant OWL et RuleML [HPSB+04]. Il est prévu pour supporter les raisonnements reposant sur les logiques de description et les règles de Horn. Le langage RuleML[19] (Rule Markup Language) permet, selon ses différentes versions, de représenter les règles de divers sous-ensembles de la logique du premier ordre.

La syntaxe de SWRL est totalement en XML. Le code 4.23 montre une règle SWRL de notre base de connaissance.

Corese

Corese est un autre langage de représentation de règles du premier ordre. Il a été développé à l'INRIA dans le cadre d'un projet de moteur de recherche sémantique basé sur le formalisme des graphes conceptuels [CDF04]. La syntaxe de Corese n'est pas complètement XML : le "sucre syntaxique" est allégé par rapport à SWRL.

Le code 4.3 montre l'expression d'une règle Corese indiquant que si une personne a écrit une thèse sur un sujet, alors c'est un expert de ce sujet.

4.2.4 Conclusion : le choix RDF/OWL/SWRL

Nous avons choisi d'utiliser conjointement trois langages de représentation des connaissances : OWL pour définir les ontologies, RDF pour "peupler" ces ontologies (*i.e.* définir des individus instances des concepts) et exprimer des assertions, SWRL pour définir des règles non exprimables en OWL. Ces trois langages sont des recommandations W3C, et déjà des standards du Web sémantique – du moins pour les deux premiers d'entre eux[21].

[19] `http://www.ruleml.org`

[20] Règle extraite de `http://www-sop.inria.fr/acacia/corese/querydoc/node51.html`.

[21] Les outils associés à SWRL étant encore en gestation. L'implémentation du *semantic Web layer cake* s'effectue progressivement, or SWRL correspond à une couche supérieure par rapport à RDF/OWL.

```
<cos:rule>
 <cos:if>
   ?p rdf:type s:Person
   ?p s:hasCreated ?doc
   ?doc rdf:type s:Thesis
   ?doc s:concern?s
 </cos:if>
 <cos:then>
   ?p s:isExpertIn ?s
 </cos:then>
</cos:rule>
```

Extrait de code 4.3: COR – Exemple de règle exprimée avec le langage Corese[20]

Nous adoptons ces trois langages car ils sont dotés d'une sémantique formelle[22]. Cela signifie que des conséquences logiques peuvent être tirées à partir des faits de notre base de connaissances. Cette propriété est absente des langages XML et XML Schema adoptés à la section précédente.

La possibilité de combiner les raisonnements associés respectivement aux LD et à la logique de Horn, ainsi que leur syntaxe XML[23] ont été les principales raisons de la préférence de RDF, OWL et SWRL par rapport à d'autres langages ; et ce, malgré des moteurs d'inférences moins éprouvés du fait de leur conception récente.

4.3 Discussion : pourquoi une architecture duale SI/SBC plutôt qu'un seul SBC ?

Si le besoin de recourir à un langage de représentation des connaissances apparaît bien clairement, inversement, la justification d'utiliser les langages documentaires XML/XML Schema mérite d'être précisée. On peut en effet se demander si la réciproque de l'affirmation de J. Euzenat "un schéma XML n'est pas une ontologie car son but est de valider un document, pas d'en définir les conséquences" [EB04] est vraie. C'est-à-dire, est-ce qu'il est également vrai qu' "une ontologie n'est pas un schéma de données car son but est de définir les conséquences logiques d'un document, pas de le valider" ? Cette interrogation est légitime si l'on considère par exemple que l'ontologie OWL-S est utilisée en tant que modèle de description des services Web. Cela peut sembler étonnant dans la mesure où dans la cadre de la gestion d'une base de description, il peut paraître souhaitable de mettre en place des mécanismes de validation non permis par OWL. Du reste, OWL-S intègre WSDL dans sa partie *grounding* et il existe bien un schéma XML pour WSDL[24].

Pour notre part, nous voyons deux raisons liées qui justifient d'utiliser XML/XML Schema pour la partie SI de notre base de métadonnées.

[22] La note de travail du W3C *"LBase : Semantics for Languages of the Semantic Web"* [W3C03a] présente le cadre dans lequel la sémantique formelle des langages RDF et OWL peut être définie en théorie des modèles. On y retrouve les notions que nous avons étudiées au chapitre 3, à savoir que les éléments d'un langage font référence aux objets d'un "monde", et qu'un monde particulier constitue une interprétation du langage. La théorie des modèles garantit alors la validité du point de vue sémantique des manipulations effectuées sur le langage. La théorie des modèles est également invoquée pour définir la sémantique de SWRL [HPSB+04]. L'idée de base est de définir des liens (*bindings*), extensions des interprétations OWL, qui prennent en compte les variables apparaissant dans les règles. Une règle SWRL est satisfaite par une interprétation si et seulement si tous les "liens-interprétations" qui satisfont l'antécédent de la règle satisfont aussi le conséquent.

[23] Il existe d'ailleurs différentes syntaxes XML plus ou moins verbeuses pour RDF, OWL et SWRL. Nous utilisons les plus concises.

[24] `http://schemas.xmlsoap.org/wsdl/2003-02-11.xsd`

D'un simple point de vue pratique, il est à l'usage nettement plus aisé de faire des requêtes et de manipuler une base XML dont on a défini le schéma que de faire la même chose sur une base RDF/OWL/SWRL. Cela s'explique pour deux raisons.

La première raison est que la syntaxe et la structure des documents RDF/OWL/SWRL est variable. Cela rend compliqué l'application directe de feuilles XSL pour les manipuler, en particulier pour générer des pages HTML. Pour manipuler les documents RDF/OWL/SWRL il faudrait donc passer par des API. Comment ferions-nous alors pour générer des pages HTML, pour prendre en exemple cet objectif central dans le cadre d'une application Web ? Plusieurs solutions seraient envisageables. Générer directement le source HTML comme du simple fichier texte est une mauvaise solution de par sa lourdeur en terme de maintenance et de modification. C'est justement l'attrait du couplage XML/XSL de générer le HTML de façon aisée, en séparant la question du contenu et celle de sa présentation. Ce type de solution est d'ailleurs à notre connaissance largement adopté aujourd'hui dans le monde professionnel des applications Web devant générer des pages dynamiques. Donc, à supposer que nous soyons initialement résolus à n'utiliser qu'une implémentation RDF/OWL/SWRL pour nos métadonnées, nous serions en fin de compte tout de même amenés à faire usage de documents intermédiaires XML.

La deuxième raison en faveur d'un format XML *ad hoc* est la possibilité pour nous de structurer la base en créant des éléments "conteneurs". Ils jouent le rôle de tables comme dans une BD relationnelle[25]. Par exemple, l'élément programmes contient des éléments programme. Cela a le mérite à la fois d'organiser d'un point de vue logique la base et d'optimiser l'exécution des requêtes grâce à une diminution du nombre d'éléments parcourus. On retrouve là des principes d'algorithmique des arbres[26].

D'un point de vue plus théorique, XML Schema permet d'exprimer des relations d'ordre entre les éléments, au contraire d'OWL. "Alors que XML (et XML Schema) sont basés sur un modèle d'arbre où les nœuds sont totalement ordonnés, OWL (et RDF) sont basés sur un modèle de graphes orientés où les arcs sont non ordonnés. En d'autres termes, la notion d'expression régulière utilisée pour contraindre la structure des documents XML est absente en OWL" ([Tro04], p.102). Dans le contexte du travail de R. Troncy, l'expression de la relation d'ordre est requise pour décrire l'agencement des séquences audiovisuelles indexées dans le système documentaire de l'Institut National de l'Audiovisuel (INA). En ce qui nous concerne, ce sont les étapes des modes d'emploi qui doivent être ordonnées. La mise en place d'un système de numérotation aurait été possible mais inélégante et inutilement lourde.

Un point central de la thèse de R. Troncy est de montrer la nécessité d'une architecture duale XML/XML Schema – RDF/OWL pour satisfaire ses besoins relevant à la fois de l'ingénierie documentaire et de l'ingénierie des connaissances. Nos besoins sont similaires à ceux de R. Troncy ; nous arrivons à la même conclusion que lui au sujet de l'architecture duale SI et SBC.

L'idée d'architecture duale est aussi présente dans les travaux de J-Y. Fortier et G. Kassel. Leur projet est de construire "une mémoire d'entreprise" reposant sur les technologies du Web sémantique. Il consiste "à développer des WSOs *[Web Sémantique d'Organisation]*[27] hybrides réalisant un couplage – fort – entre une Base de Connaissances (BC) et une Base de Documents (BDoc)" [FK04]. J-Y. Fortier et G. Kassel parlent de "couplage fort" car ils ne se contentent pas d'annoter les documents au moyen de termes d'ontologies à de "simples" fins d'indexation. Ils souhaitent de plus – si nous avons bien compris leur propos – représenter une partie des connaissances de leurs documents dans un langage de représentation des connaissances basé sur les ontologies. Cette question du couplage fort ou faible s'est également posée à nous. Prenons

[25] Pour éditer notre base de métadonnées en phase de développement, nous avons justement beaucoup utilisé la très pratique vue "table" (*table view*) du logiciel XML 2004.

[26] En cherchant à limiter le nombre maximum de fils des nœuds d'un arbre et en créant de nouvelles branches pour équilibrer ces branches, on tend à obtenir des durées semblables lors des recherches de feuilles (avec des requêtes de complexités comparables).

[27] Synonyme de mémoire d'entreprise basée sur les technologies du Web sémantique.

un exemple concret. Nous aurions pu nous contenter de définir un schéma XML indiquant que des traitements réalisent des fonctionnalités. Parallèlement, nous aurions défini une ontologie des fonctionnalités géographiques. Nous aurions alors réalisé un couplage faible en nous limitant à utiliser les termes de l'ontologie des fonctionnalités pour les valeurs de la propriété "réalise" des traitements. Mais seul alors une partie des raisonnements souhaités auraient été possibles. C'est pourquoi dans l'architecture duale pour laquelle nous avons opté les éléments de descriptions de notre schéma XML sont tous transposés en RDF/OWL.

4.4 Implémentation de la base de métadonnées – aspect "SI"

4.4.1 Principes et aperçu général

Le schéma de la base de métadonnées à vocation documentaire se présente sous la forme d'un ensemble de fichiers XML Schema. Ces fichiers portent l'extension *.xsd* (XML Schema Description). Chacun définit un ensemble de type d'élément XML. Ce sont les équivalents des classes de notre modèle conceptuel. Définir un type d'élément, c'est décrire quels sont ses attributs et quel est son contenu : chaîne de caractères ou éléments fils. Notre schéma XML complet définit un élément racine, 129 *types complexes* et 7 *types simples*. Les éléments de type complexe peuvent posséder des éléments fils, pas les éléments de type simple.

Le code 4.4 définit le type simple des identifiants des métadonnées. La convention que nous avons adoptée impose qu'ils possèdent au moins 3 caractères. Cette contrainte s'exprime au moyen d'une expression régulière. Les codes 4.10 et 4.12 sont des exemples de types complexes.

```
<xsd:simpleType name="mdt_idType">
  <xsd:restriction base="xsd:string">
     <!-- 3 caractères non-espace minimum (préfixe + id) -->
     <xsd:pattern value="[s]{3,}"/>
   <xsd:restriction>
<xsd:simpleType>
```

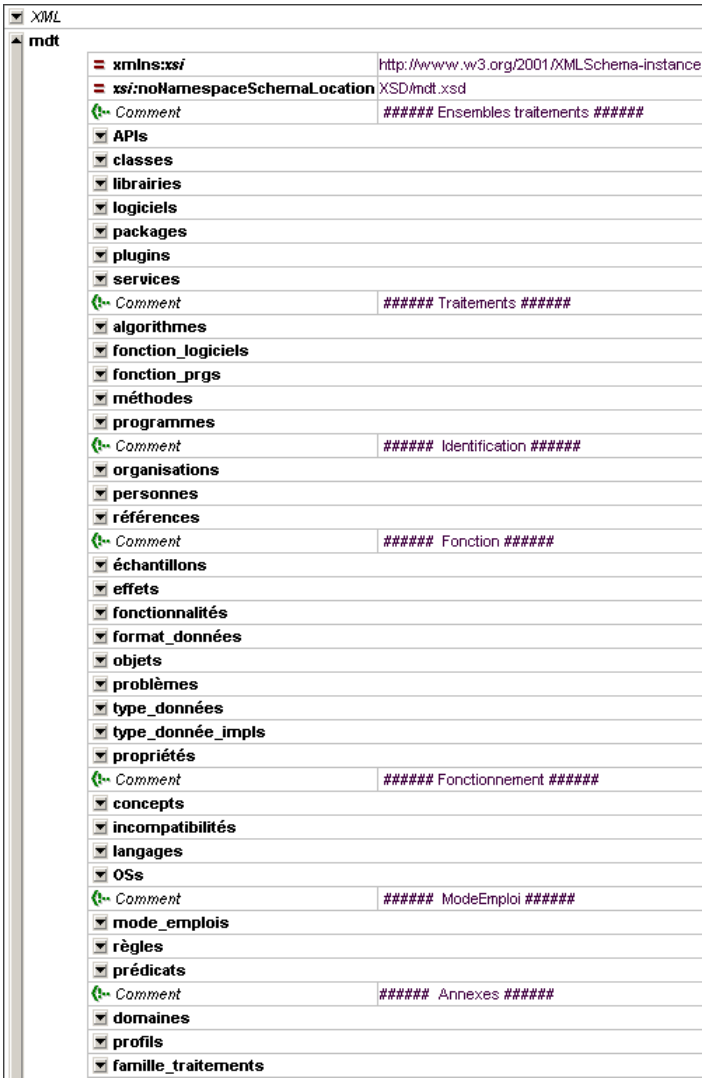
Extrait de code 4.4: XSD – Type simple mdt_idType

La dérivation par extension entre types complexes nous permet de traduire les relations d'héritage. Par exemple dans notre modèle conceptuel fig. 2.18, Algorithme et TraitementCodé héritent de RessourceTraitement. Dans notre schéma XML, traitementCodéType étend traitementCodé en ajoutant les nouveaux éléments de description langage, OS et implémente (code 4.12).

La base de métadonnées est stockée sous la forme d'un fichier XML unique dont l'élément racine est nommé mdt (*m*éta*d*onnées des *t*raitements). Cet élément racine contient une trentaine de collections d'éléments. On peut les voir comme les tables de bases de données relationnelles. Le code XML 4.5 en donne un aperçu général de la base.

On remarque que des cinq facettes de description des traitements définis dans notre modèle conceptuel, seule ModeEmploi figure en tant que ressource dans la base de métadonnées. Un mode d'emploi possède un identifiant. Il peut être consulté indépendamment des traitements auxquels il est éventuellement rattaché. Les quatre autres facettes ne possèdent pas d'identifiant. Identification et Evaluation ne peuvent apparaître qu'incluses au sein d'une RessourceTraitement ou d'une FamilleTraitement, Fonction et Fonctionnement ne sont pas réifiées en tant qu'éléments[28].

[28] Ce faisant, nous nous démarquons des descriptions OWL-S dont les trois facettes existent en tant que ressources RDF, donc en tant qu'éléments XML possédant un identifiant `rdf:ID`.



Extrait de code 4.5: XML – Aperçu général de la base de métadonnées (vue "grille" de XML Spy 2004)

Nous avons fait ce choix car nous tenons à distinguer la question de la présentation des métadonnées de celle leur implémentation. Or, selon nous, les facettes de description relèvent avant tout de la *présentation* des métadonnées. Leur présence se justifie donc dans le modèle conceptuel et dans l'interface graphique de l'utilisateur, mais pas obligatoirement dans le modèle d'implémentation : seulement dans la mesure où cela semble pertinent d'un point de vue logique et/ou efficace d'un point de vue de facilité de développement. En l'occurrence, créer par exemple un élément "Fonctionnement" destiné à regrouper les éléments de description relatifs à ce thème n'apporte aucune information ni aucun avantage du point de vue de l'exploitation informatique.

4.4.2 Identification d'un traitement

Le code 4.6 décrit l'identification du programme Accordéon. Ici l'élément de description nom est le seul dont la valeur est de type chaîne de caractère libre (sa longueur ne doit toutefois pas être nulle).

Les autres éléments sont soit de type numérique ou de type date, soit de type mdt_id (c'est-à-dire faisant référence à d'autres éléments de la base de métadonnées).

```
<programme id="prgAccordion2">
  <identification>
    <nom>Accordion v.2</nom>
    <version>
      <numéro>2</numéro>
    </version>
    <appartient>orgPlage</appartient>
    <auteur>persMauffrey</auteur>
    <auteur>persLecordix</auteur>
    <auteur_description>persMustière</auteur_description>
    <date_création>01/04/1998</date_création>
    <date_last_modification/>
    <lieu_de_développement>orgCOGIT</lieu_de_développement>
    <condition>conDevIGN</condition>
    <référence>refPlazanet96</référence>
  </identification>
   <!-- ... -->
```

Extrait de code 4.6: XML – Identification du programme Accordéon

4.4.3 Décrire ce que fait un traitement

Les codes 4.7 et 4.8 décrivent ce que fait le programme Accordéon. Il s'agit donc de la facette Fonction. La plupart des éléments ont pour valeur des identifiants de ressources. Ce n'est pas encore le cas pour tous. Cela devra le devenir si l'on souhaite augmenter le niveau de formalisation des descriptions. Pour cela, il faudra importer des ontologies de haut niveau. Par exemple, la description qualitative des propriétés fait appel à des concepts ou individus "peu", "beaucoup", "petit", "moyen", "grand", etc. Notre modèle permet de définir l'équivalent des types énumérés pour les types de données (cf. l'élément valeurs code 4.9). Mais ces descriptions appartiennent à la partie SI de notre application. Elles n'ont pas été traduites dans la partie SBC en RDF/OWL. La sémantique potentielle d'une partie des descriptions reste donc inexploitée. Par exemple, répondre à la requête "quel est le paramètre du traitement Accordéon qui influe *le plus* sur le déplacement des points d'inflexion ?" nécessiterait de représenter une relation d'ordre sur les valeurs des propriétés.

```
  <domaine>domAnalyseVecteur</domaine>
  <description>élargit un virage ou une série de virages afin de supprimer les
fusions de virages. Le point d'inflexion central de la ligne ne bouge pas, et tous
les autres points sont écartés de ce point central d'une distance epsilon, dans
la direction orthogonale de l'axe de chaque virage. La valeur epsilon est propre à
chaque virage.</description>
  <réalise>fctCaricaturer</réalise>
  <effets>
    <effet niveau="micro" ref="pteGénPosition">inchangée</effet>
    <effet niveau="micro" ref="pteGénOrientation">moins</effet>
    <effet niveau="micro" ref="pteGénAspect">inchangée</effet>
    <effet niveau="micro" ref="pteGénSémantique">inchangée</effet>
    <effet niveau="micro" ref="pteGénForme">inchangée</effet>
    <effet niveau="micro" ref="pteGénDistribution">inchangée</effet>
    <effet niveau="micro" ref="pteGénRépartitionSémantique">inchangée</effet>
    <effet niveau="micro" ref="pteGénOrientationSpatiale">inchangée</effet>
    <effet niveau="méso" ref="pteGénProximité">inchangée</effet>
    <effet niveau="méso" ref="pteGénTopologie">inchangée</effet>
    <effet niveau="macro" ref="ptGénQuantitéObjets">inchangée</effet>
    <effet niveau="macro" ref="pteGénRépartitionSémantique">inchangée</effet>
    <commentaire>Les extrémités de la ligne sont déplacées, l'aspect topologie du
réseau doit donc être recalculé.</commentaire>
  </effets>
  <entrées>
    <entrée>
      <nom>virage ou série de virages</nom>
      <type_donnée>tdVecteurLigne</type_donnée>
      <objet>objVirage</objet>
      <modifiable>oui</modifiable>
      <pte ref="pteSinuosité">
        <avant>
          <valeur_qual>moyen</valeur_qual>
        </avant>
        <après>
          <valeur_qual>petit</valeur_qual>
        </après>
        <évolution>diminution</évolution>
        <pb_résolu>pbLisibilitéCarte</pb_résolu>
      </pte>
      <pte ref="ptNbVirageKm">
        <évolution>diminution</évolution>
      </pte>
      <pte ref="pteAspect">
        <évolution>plus lisse</évolution>
        <pb_résolu>pbLisibilitéCarte</pb_résolu>
        <commentaire>on voit mieux l'écartement</commentaire>
      </pte>
```

Extrait de code 4.7: XML – Fonction du programme Accordéon 1/2

```
    <pte ref="ptePointInflexion">
      <avant>
        <valeur_quant>n</valeur_quant>
      </avant>
      <après>
        <valeur_quant>f(n, epsilon, param2)</valeur_quant>
      </après>
      <différence>n - f(n, epsilon, param2)</différence>
      <facteur>entre 1 et m</facteur>
      <commentaire>dépend du paramètre tau (cf. étape 1)</commentaire>
    </pte>
  </entrée>
</entrées>
<sorties>
  <sortie>
    <nom>le virage ou la sortie de virage élargis</nom>
    <correspond_entrée>1</correspond_entrée>
    <type_donnée>tdVecteurLigne</type_donnée>
    <objet>objVirage</objet>
  </sortie>
</sorties>
<paramètres>
  <paramètre>
    <nom>logma</nom>
    <type_donnée>tdInt</type_donnée>
    <influe_pte>
        <pté ref="ptePointInflexion"/>
       <influence/>
    </influe_pte>
    <min>5</min>
    <max>17</max>
  </paramètre>
</paramètres>
<illustration>accordéon_exemple.bmp</illustration>
<illustration_echantillon>
  <avant>echRouteMontagne1</avant>
  <après>echRouteMontagne2</après>
</illustration_echantillon>
```

Extrait de code 4.8: XML – Fonction du programme Accordéon 2/2

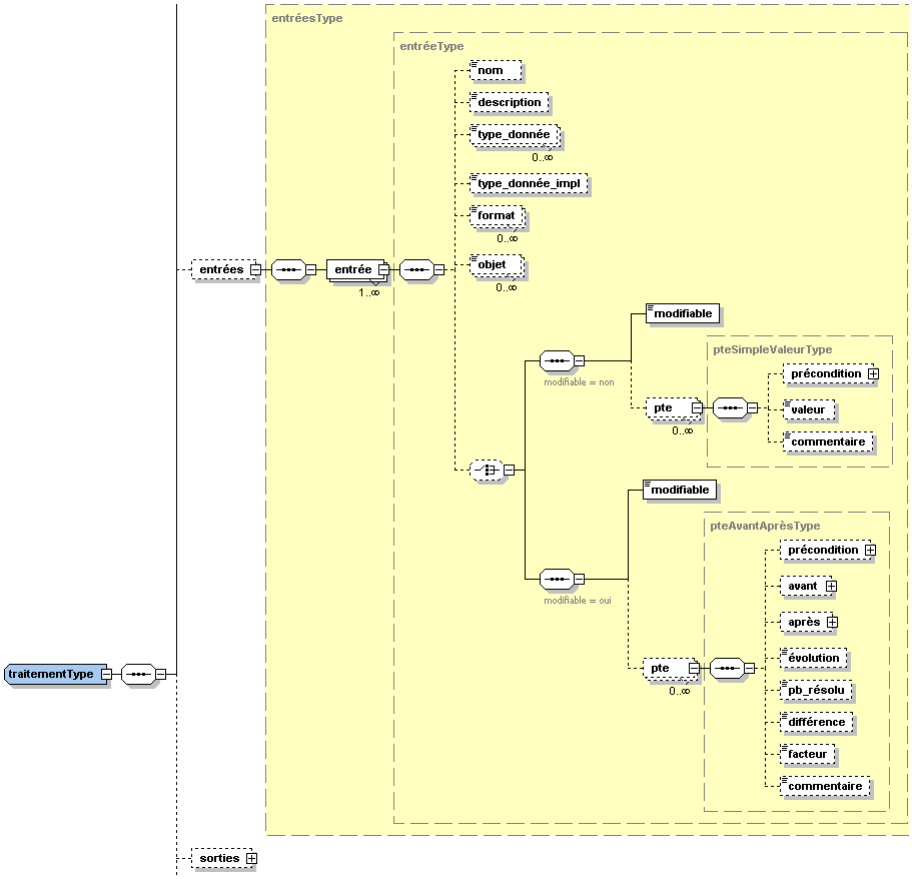
```
<type_données>
    <!-- ... -->
    <type_donnée id="tdVecteurLigne">
      <nom>ligne vecteur</nom>
      <domaine>domAnalyseVecteur</domaine>
      <description/>
      <type_donnée_pere>tdVecteur</type_donnée_pere>
      <propriétés>
        <niveau>géométrie</niveau>
        <propriété niveau="géométrie" ref="pteAspect"/>
        <propriété niveau="géométrie" ref="pteDimension"/>
        <propriété niveau="géométrie" ref="pteDirection"/>
        <propriété niveau="géométrie" ref="pteLongueur"/>
        <propriété niveau="géométrie" ref="ptePointInflexion"/>
        <propriété niveau="géométrie" ref="ptePosition"/>
        <propriété niveau="géométrie" ref="pteProvenance"/>
        <propriété niveau="géométrie" ref="pteSinuosité"/>
        <propriété niveau="géométrie" ref="pteTopologie"/>
      </propriétés>
    </type_donnée>
</type_données>

<propriétés>
    <!-- ... -->
    <propriété id="pteSinuosité">
      <nom>sinuosite</nom>
      <qualitatif>
        <valeurs>
          <valeur>petit</valeur>
          <valeur>moyen</valeur>
          <valeur>grand</valeur>
        </valeurs>
      </qualitatif>
      <quantitatif>
        <unités>
          <unité>
            <nom>virages/km</nom>
          </unité>
        </unités>
      </quantitatif>
    </propriété>
</propriétés>
```

Extrait de code 4.9: XML – Type de donnée ligne vecteur et propriété sinuosité

Détail d'un type d'élément : exemple du type traitementType

Nous avons vu des extraits de code de descriptions XML. Détaillons à présent un extrait de code de schéma XML. Certaines entrées de traitements sont modifiables. D'autres ne le sont pas. Elles ne se décrivent pas de la même façon.

Le code 4.10 montre comment s'exprime la dépendance entre la valeur de l'élément booléen modifiable et les éléments fils autorisés de l'élément entrée. La version texte du code 4.10 est donnée code A.1, p. 232.



Extrait de code 4.10: XSD – Type complexe traitementType (vue "schema" de XML Spy 2004)

4.4.4 Décrire comment fonctionne un traitement

Le code 4.11 décrit comment fonctionne le programme Accordéon.

```
<langage>langADA</langage>
<code>http://walrus:8080/Mdt/documentsConsultables/code/plage/
 ACCORDION_V2.ADA</code>
<projet>pjAGENT</projet>
<OS>osVMS</OS>
<implémente>algoAccordion2</implémente>
<concept>conCirconvolutionGaussienne</concept>
```

Extrait de code 4.11: XML – Fonctionnement du programme Accordéon

Un TraitementCodé comme Accordéon possède les mêmes propriétés qu'un Algorithme : celles héritées de RessourceTraitement. Un TraitementCodé possède toutefois quelques propriétés supplémentaires spécifiques à l'implémentation. Pour traduire cela, nous utilisons la dérivation par extension. C'est ce que montre le code 4.12 (le type XML Schema traitementType correspond à la classe RessourceTraitement du modèle conceptuel).

```
<xsd:complexType name="traitementCodéType">
  <xsd:complexContent>
    <xsd:extension base="traitementType">
      <xsd:sequence>
        <xsd:element name="langage" type="xsd:string"
                     minOccurs="0" maxOccurs="1"/>
        <xsd:element name="OS" type="xsd:string"
                     minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="implémente" type="xsd:string"
                     minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Extrait de code 4.12: XSD – Dérivation par extension du type complexe traitementType

Concernant les expressions mathématiques, nous importons le schéma XML de MathML2. Le code 4.13 montre comment est spécifiée l'importation[29] et comment il est fait référence à l'élément math. Le code 4.14 montre un extrait de description comportant une expression MathML2.

Un cas particulier, dont nous ne donnons pas d'exemple ici, est celui des services Web. Leur description comporte un élément supplémentaire permettant de référencer les descriptions WSDL éventuellement existantes. Afin d'éviter les informations redondantes, les principes généraux sur le fonctionnement des services Web et leurs modes d'emplois associés (p.ex. créer une requête SOAP, créer un client SOAP, etc.) sont centralisés dans la description de la FamilleTraitement "service Web".

4.4.5 Décrire comment utiliser un traitement

Pour illustrer la façon de décrire le fonctionnement d'un traitement nous n'avons pas choisi l'exemple du programme Accordéon. Nous avons choisi l'exemple partiel du mode d'emploi pour développer un visualisateur de MNT avec OpenGL (cf. fig. 1.9, p. 26). La description de ce mode d'emploi est donnée code 4.15. La façon dont notre application affiche cette description est montrée figure 5.6, p. 186.

Ce mode d'emploi a le mérite de comporter des étapes des types possibles : étape_fct, étape_ihm et étape_dév.

Les éléments séquence et choix permettent d'exprimer l'équivalent des arbres "et/ou" ordonnés.

Il est important de noter que les étapes font référence à différentes ressources, modes d'emploi ou traitements. C'est, notamment, ce qui va permettre à l'utilisateur de naviguer d'une description à l'autre via des hyperliens.

[29] Il est normal que l'attribut `schemaLocation` contienne une paire de valeurs : la première est l'espace de nom du schéma auquel se réfère le document ; la seconde est l'endroit (URI) où le processeur XML peut trouver ce schéma [W3C04f] – §5.6.

```
<xsd:schema elementFormDefault="qualified"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:mml="http://www.w3.org/1998/Math/MathML">

  <xsd:import namespace="http://www.w3.org/1998/Math/MathML"
            schemaLocation="mathml2/mathml2.xsd"/>


  <!-- ... -->

  <xsd:complexType name="étape_fonctionnementType">
    <xsd:sequence>
      <xsd:element name="description" type="xsd:string"
             minOccurs="0"/>
      <xsd:element ref="mml:math" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Extrait de code 4.13: XSD – Importation du schéma MathML2

```
<?xml version="1.0" encoding="iso-8859-1"?>
<mdt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="XSD/mdt.xsd"
         xsi:schemaLocation="http://www.w3.org/1998/Math/MathML
mathml2/mathml2.xsd"
         xmlns:mml="http://www.w3.org/1998/Math/MathML">
      <!-- ... -->
      <étape_fonctionnement>
        <description>Calcul du nombre de points d'inflexion d'une
ligne.</description>
        <mml:math>
          <mml:mn>1</mml:mn>
          <!-- ... -->
        </mml:math>
      </étape_fonctionnement>
<!-- ... -->
```

Extrait de code 4.14: XML – Extrait de description comportant une expression MathML2

```
<mode_emploi id="modVisualiserMNTavecOpenGL">
  <type>modUtiliserAPI</type>
  <nom>visualiser un MNT avec OpenGL</nom>
  <requis>pjOpenGL</requis>
  <séquence>
    <étape_ihm>
      <référence>modOuvrirEclipse</référence>
    </étape_ihm>
    <étape_ihm>
      <référence>modCreateNewFileC++</référence>
    </étape_ihm>
    <étape_dév>
      <code trait="pjOpenGL">#include "glut.h"</code>
    </étape_dév>
    <choix>
      <étape_dév>
        <code trait="fpSscanf">sscanf();</code>
      </étape_dév>
      <étape_fct>
        <description>lire le fichier texte du MNT</description>
      </étape_fct>
    </choix>
    <étape_dév>
      <code trait="fpGlVertex">glVertex3f(p.x,p.y,p.z);</code>
    </étape_dév>
  </séquence>
</mode_emploi>
```

Extrait de code 4.15: XML – Mode d'emploi pour créer un client de service Web

4.4.6 Évaluation d'un traitement

Le code 4.16 décrit l'évaluation du programme Accordéon. La plupart des éléments contiennent du texte en langue naturelle. Une amélioration envisageable du modèle pourrait consister à formaliser davantage ces descriptions par l'annotation avec, par exemple, les termes d'une ontologie des avantages et des inconvénients des traitements.

4.4.7 Décrire les familles de traitement

Le code 4.17 montre la description de la famille des SIG. Il y est indiqué que les SIG sont des logiciels qui réalisent cinq types de fonctionnalités. L'attribut condition permet d'indiquer si les propriétés des familles sont nécessaires (N) et/ou suffisantes (S). Cependant notre modèle XML Schema ne permet pas de définir les familles de façon aussi fine qu'en OWL. La traduction des éléments FamilleTraitement en tant que *concepts définis* de l'ontologie de notre SBC ne s'effectue donc pas de façon totalement automatique.

Il existe un mode d'emploi générique associé aux SIG ; il est référencé ici de façon centralisée. Par ce biais, tous les SIG voient figurer dans leur description les informations associées à leur famille. Ces informations ont été identifiées au chapitre 1 (tableau 1.9 p. 32). Une même RessourceTraitement peut appartenir à plusieurs FamilleTraitement : par exemple Arcview appartient aux familles "SIG" et "logiciel Windows". On voit d'ailleurs indiqué dans le code 4.17 qu'Arcview est le prototype de la famille des SIG.

```
   <évaluation>
     <avantage>
       <description>L'algorithme utilisé est bien adapté aux virages en
épingles à cheveux quand l'environnement n'est pas trop dense et quand
l'espace disponible autour des virage est suffisant. C'est un des rares
algorithmes qui déplace les extrémités de la ligne pour profiter de l'espace
disponible.</description>
     </avantage>
     <inconvénient>
       <description>Most of the time, it has to be used with a segmentation
algorithm to focus onto the bends series, a caricature algorithm that is able
to enlarge the bend extremities, and with a point displacement propagation
algorithm to reconnect the network.</description>
     </inconvénient>
     <amélioration_possible>La détermination de la direction du déplacement
par une méthode interactive pour obtenir epsilon ne marche pas à tous les
coups car l'angle qui donne la direction de l'axe du virage n'est pas orienté.
Quand les points d'inflexion sont proches du sommet du virage, l'approximation
de e dans la formule d'epsilon n'est pas entièrement valide. Le problème
pourrait être résolu en déterminant l'axe du virage et la tangente du virage
près du point d'inflexion. On obtiendrait alors la valeur réelle de la largeur
du virage.</amélioration_possible>
     <comparaison ref="algoAccordion">Contrary to the classical algorithm
of the accordion, this one is also adapted to bend series even badly
aligned and in a bend series, only the bends which need it are enlarged.
The enlargement direction and quantity is automatically chosen for each
bend according to the bend configuration and the symbol size. So there is
no parameter.</comparaison>
     <complexité>o(n)</complexité>
     <temps_exécution_max>S secondes pour V virages</temps_exécution_max>
     <test>
       <description>généralisation à petite échelle (1/100.000) sur
des extraits du réseau routier de la BD (10m. resolution) : R2 and
Route70</description>
       <objet>objRoute</objet>
       <format>forBDCarto</format>
     </test>
   </évaluation>
```

Extrait de code 4.16: XML – Évaluation du programme Accordéon

```
    <famille_traitement id="catSIG">
      <nom>SIG</nom>
      <type>catLogiciel</type>
      <description>Système d'Information Géographique</description>
      <prototype>logArcview</prototype>
      <réalise condition="N">fctAcquérir</réalise>
      <réalise condition="N">fctArchiver</réalise>
      <réalise condition="N">fctAbstraire</réalise>
      <réalise condition="N">fctAnalyser</réalise>
      <réalise condition="N">fctAfficher</réalise>
      <mode_emploi>modUtiliserSIG</mode_emploi>
    </famille_traitement>
```

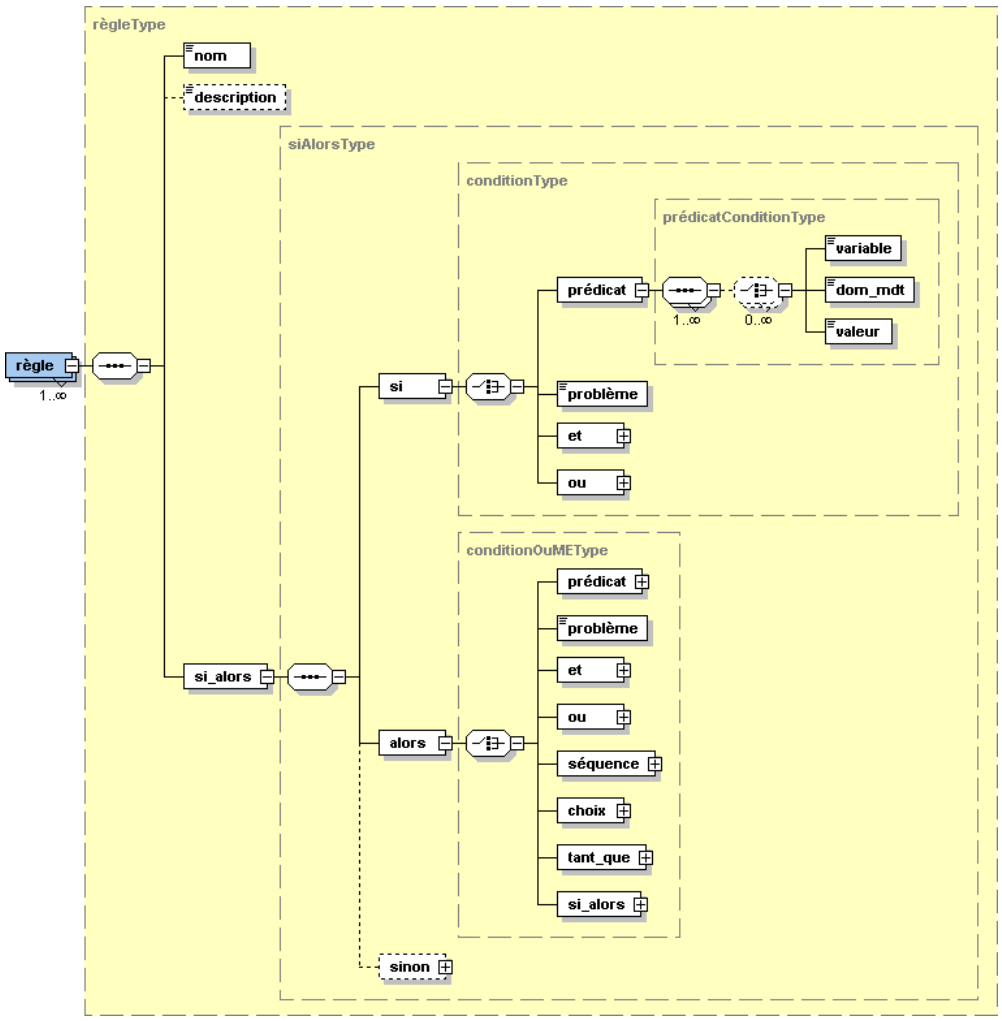
Extrait de code 4.17: XML – Famille de logiciels : les SIG

4.4.8 Décrire les règles de l'expert

Le code 4.18 est un extrait de notre schéma XML. On y voit le type complexe règleType qui définit la structure des règles. Le code 4.19 montre une instance de ce type. La règle en question indique que les données au format shape ne représentent pas les relations topologiques. Les prémisses et les conclusions des règles utilisent une notation DOM pour adresser les différents éléments de la base de métadonnées.

Un DOM (Document Object Model) est un modèle objet de documents. Il existe ainsi par exemple un DOM HTML pour adresser les éléments des pages HTML et un DOM Word pour adresser les éléments des documents Microsoft Word. Les principes DOM ont fait l'objet de plusieurs recommandations du W3C[30].

Nous utilisons la notation DOM pour l'expression de nos règles de la partie SI de nos métadonnées en raison de sa concision. Par exemple, le format de la première entrée du logiciel FreeWRL est désigné par l'expression DOM : `traitement['logFreeWRL'].entrée[1].format`. Cette expression a pour valeur `"forVRML"`.



Extrait de code 4.18: XSD – Type complexe règleType

[30] http://www.w3.org/DOM/

```
<règle id="rgSHPpasTopologie">
  <nom>shape absence topologie</nom>
  <si_alors>
    <si>
      <prédicat ref="predEgal">
        <dom_mdt>donnee.format</dom_mdt>
        <valeur>forSHP</valeur>
      </prédicat>
    </si>
    <alors>
      <prédicat ref="predEgal">
        <dom_mdt>donnee.pte('pteTopologie')</dom_mdt>
        <valeur>faux</valeur>
      </prédicat>
    </alors>
  </si_alors>
</règle>
```

Extrait de code 4.19: XML – Règle sur l'absence de topologie des données au format SHP

4.5 Implémentation de la base de métadonnées – aspect "SBC"

Nous allons maintenant traduire notre modèle de métadonnées et notre base de métadonnées dans les langages de représentation des connaissances OWL, RDF et SWRL. La figure 4.3 résume la façon dont on passe du SI au SBC.

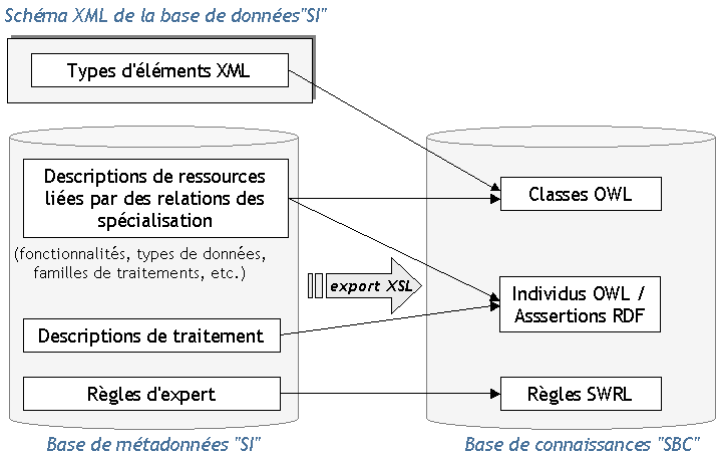


FIG. 4.3 – Correspondances SI/SBC

4.5.1 Ontologie OWL

Examinons deux extraits de notre base de connaissances OWL.

Le code 4.20 montre que la classe SIG est définie comme étant à la fois – la conjonction est implicite – sous-classe de la classe anonyme des logiciels réalisant cinq fonctionnalités, et sous-classe de la classe des familles de traitement.

Le code 4.21 montre comment exprimer que la fonctionnalité de lissage, sorte particulière de fonctionnalité de généralisation, s'applique à des données de type "ligne vecteur" et non à des données de type "vecteur" en général (cf. fig. 2.16 p. 95). La propriété "typeDonnée" est *ObjectProperty*, c'est-à-dire une propriété entre objets de la base de connaissance. La propriété

```
<owl:Class rdf:ID="SIG">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="realise"/>
          </owl:onProperty>
          <owl:hasValue>
            <Fonctionnalite rdf:ID="afficheDonnneeGeo"/>
          </owl:hasValue>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#realise"/>
          </owl:onProperty>
          <owl:hasValue>
            <Fonctionnalite rdf:ID="analyseDonnneeGeo"/>
          </owl:hasValue>
        </owl:Restriction>
        < !-- les trois autres fonctionnalités + catégorie Logiciel... -->
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#FamilleTraitement"/>
</owl:Class>
```

Extrait de code 4.20: OWL – Définition du concept SIG dans notre base de connaissances

"version" d'un traitement, par exemple, relie un objet et le type de données "nombre réel" ; c'est une *DatatypeProperty*.

4.5.2 Assertions RDF

Le code 4.22 montre des assertions RDF correspondant à un extrait de la description du logiciel FreeWRL. La syntaxe RDF utilisée ici est une syntaxe abrégée. C'est une syntaxe valide. D'autres sont possibles. La syntaxe dite "de sérialisation" est plus verbeuse. Les différentes façons de l'abréger sont exposées dans [W3C99a], §2.2.

4.5.3 Règles SWRL

La règle SWRL code 4.23 indique que si un traitement est développé dans une organisation qui appartient à une autre organisation, alors le traitement est aussi développé dans cette autre organisation. Cette règle simple apparaît assez verbeuse en SWRL. Le problème s'aggrave encore lorsqu'il faut adresser des éléments de la base de métadonnées situés à un grand niveau de profondeur de l'arborescence XML.

En effet, les expressions en notation DOM que nous utilisons dans la partie SI de l'application ne sont pas permises par SWRL. Elles ne font pas partie du langage. Il faut donc traduire les expressions DOM sous forme de conjonction de prédicats. De façon similaire, Baget *et. al*, notamment, évoquent la transformation de triplets RDF : "À chaque triplet $<s, p, o>$ on associe la formule atomique $p(o, s)$, où p est un nom de prédicat, et o et s sont des constantes si ces éléments sont des URIs ou des littéraux dans le triplet, et des variables sinon" [BCES04].

```
<owl:Class rdf:ID="Fct_Généralisation"/>

<owl:Class rdf:ID="Fct_Lissage">
  <rdfs:subClassOf rdf:resource="#Fct_Généralisation"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="typeDonnée">
  <rdfs:range rdf:resource="#TD_Vecteur"/>
  <rdfs:domain rdf:resource="#Fct_Généralisation"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="TD_VecteurLigne">
  <rdfs:subClassOf>
      <owl:Class rdf:ID="TD_Vecteur"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#typeDonnée"/>
      <owl:allValuesFrom rdf:resource="#TD_VecteurLigne"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Extrait de code 4.21: OWL – Restriction de propriété

```
<Logiciel rdf:ID="FreeWRL">
  <version
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.91</version>
  <realise rdf:resource="#afficheDonnee3D"/>
  <entree rdf:resource="#entFreeWRL"/>
</Logiciel>

<Donnee rdf:ID="entFreeWRL">
  <typeAbst rdf:resource="#tda_3D"/>
  <format rdf:resource="#forVRML"/>
  <aPropriete>
    <Propriete rdf:ID="nbObjetsEntFreeWRL">
      <type rdf:resource="#nbObjets"/>
      <valeurMaxInt
           rdf:datatype="http://www.w3.org/2001/XMLSchema#int">5000
     </valeurMaxInt>
    </Propriete>
  </aPropriete>
</Donnee>
```

Extrait de code 4.22: RDF – Description du logiciel FreeWRL

Cette transformation constitue un des points faibles de notre application. D'abord parce que l'automatisation théorique réalisée à l'aide d'expressions régulières ne fonctionne pas toujours en pratique. Davantage de développements seraient nécessaires, notamment en employant une API pour générer proprement les règles SWRL[31] plutôt qu'à partir de simples manipulations de chaînes de caractères comme actuellement.

Un autre problème des règles SWRL est celui de l'efficacité. Lorsque l'on utilise une expression de type DOM en programmation Java par exemple, on manipule en fait des adresses mémoire. L'évaluation d'une expression est alors très rapide, il n'y a pas de traitement à effectuer. Mais en "éclatant" ces expressions en conjonction de prédicats, on perd cette notion d'adresse : potentiellement, le parcours de toutes les assertions RDF de la base devient nécessaire pour effectuer les jointures entre les prédicats et instancier les variables qui y apparaissent. Sans doute les moteurs d'inférences SWRL peuvent-ils construire des tables stockant à l'avance des résultats pré-calculés pour optimiser l'exécution des règles, mais un calcul sera de toutes façons nécessaire. Ainsi, les tests que nous avons effectués avec notre base de règles SWRL prenaient assez vite des temps d'exécution de l'ordre de la dizaine de secondes.

[31] Par exemple `http://protege.stanford.edu/plugins/owl/swrl/SWRLFactory.html`.

```
<swrl:Imp rdf:ID="appartientLieuDeDev">
  <swrl:body>
    <swrl:AtomList>
      <rdf:first>
        <swrl:IndividualPropertyAtom>
          <swrl:propertyPredicate rdf:resource="#lieuDeDeveloppement"/>
          <swrl:argument2>
            <swrl:Variable rdf:ID="l1"/>
          </swrl:argument2>
          <swrl:argument1 rdf:resource="#t"/>
        </swrl:IndividualPropertyAtom>
      </rdf:first>
      <rdf:rest>
        <swrl:AtomList>
          <rdf:first>
            <swrl:IndividualPropertyAtom>
              <swrl:propertyPredicate rdf:resource=
"#appartientOrganisation"/>
              <swrl:argument1 rdf:resource="#l1"/>
              <swrl:argument2>
                <swrl:Variable rdf:ID="l2"/>
              </swrl:argument2>
            </swrl:IndividualPropertyAtom>
          </rdf:first>
          <rdf:rest rdf:resource=
          "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
        </swrl:AtomList>
      </rdf:rest>
    </swrl:AtomList>
  </swrl:body>
  <swrl:head>
    <swrl:AtomList>
      <rdf:rest rdf:resource=
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
      <rdf:first>
        <swrl:IndividualPropertyAtom>
          <swrl:propertyPredicate rdf:resource="#lieuDeDeveloppement"/>
          <swrl:argument1 rdf:resource="#t"/>
          <swrl:argument2 rdf:resource="#l2"/>
        </swrl:IndividualPropertyAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:head>
</swrl:Imp>
```

Extrait de code 4.23: SWRL – Règle pour déduire le lieu de développement d'un traitement, les relations entre organisation étant prises en compte (cette règle est décrite sous une forme plus lisible dans le tableau 5.1 p. 196)

4.6 Conclusion

Nous avons implémenté le modèle conceptuel défini aux chapitres 2 et 3 suivant une double approche : *documentaire*, et orientée *représentation des connaissances*.

Nous avons d'abord défini la structure des métadonnées sous une forme qui en permet le contrôle. L'objectif est alors la mise en œuvre de notre *système d'information (SI)* .

Nous avons ensuite décrit les traitements avec les termes d'ontologies qui constituent, avec les règles de logique du premier ordre, le socle de notre *système à base de connaissances (SBC)*.

Les choix des langages effectués, nous avons montré comment nous avons implémenté notre modèle conceptuel défini aux chapitres 2 et 3. Des extraits de code du modèle d'implémentation et de ses instances ont servi d'illustrations.

Il reste maintenant à développer l'application qui va permettre à l'utilisateur d'accéder aux métadonnées et qui va mettre en œuvre une partie du raisonnement de l'expert.

Chapitre 5

L'application Web permettant l'accès aux métadonnées

Les choix d'implémentation effectués au chapitre 4 nous permettent de construire une base de métadonnées "documentaire" et une base de connaissances. Il nous faut à présent permettre l'accès et l'exploitation de ces bases.

Après avoir présenté l'architecture générale de notre application d'accès aux métadonnées section 5.1, nous en décrivons plus particulièrement les aspects "SI" et "SBC" sections 5.2 et 5.3. Nous montrons ensuite section 5.4 comment nous mettons en œuvre quelques-uns des exemples de raisonnement ER vus au chapitre 3. Section 5.4, nous tentons de cerner quelques-unes des limites de l'application construite.

5.1 Architecture de l'application

Nous développons une application Web accessible depuis l'intranet de l'IGN. Nous adoptons une architecture Web classique "n-tiers" :

L'utilisateur dispose d'un ordinateur équipé d'un navigateur Web standard et connecté au réseau intranet de l'IGN. Via le protocole HTTP, l'utilisateur accède aux pages HTML de l'application, soumet de requêtes et transmet des données au serveur Web.

Le serveur Web reçoit les requêtes de l'utilisateur. Il les reformule et les transmet au serveur d'application. Le serveur d'application renvoie la réponse au format XML. Le serveur Web effectue la mise en forme en transformant le XML en HTML qui est alors envoyé à l'utilisateur. Le rôle du serveur Web se limite à générer du HTML à partir des réponses XML fournies par le serveur d'application.

Le serveur d'application effectue toutes les opérations sur les métadonnées autres que leur mise en forme : exécution des requêtes, adaptation des modes d'emploi, vérification de la conformité de la base de métadonnées au modèle, pont entre les formalismes SI et SBC. Le serveur d'application peut être découplé en plusieurs serveurs d'application distincts, éventuellement répartis sur des machines distantes.

Les serveurs de données stockent les bases de métadonnées "SI" et "SBC". Par choix de simplicité, nous n'avons en fait pas réellement mis en place de serveur, du moins pas qui utilise un protocole du Web. L'accès aux bases de métadonnées s'effectue en effet par simple chargement de fichiers. Cette solution préserve la philosophie de l'architecture souhaitée, à savoir que la base de données soit dissociée de l'application. Une nouvelle application doit-elle

accéder aux métadonnées ? Elle le peut, indépendamment des applications existantes.

La figure 5.1 illustre l'architecture que nous avons mise en place. Les flèches entre les parties du système signifient "interagit avec". L'*application Consul* apparaît en grisé car nous ne l'avons pas développée. Elle ne faisait pas partie de nos objectifs, mais nous devions prendre en compte son existence future. Il est en effet prévu que notre serveur de métadonnées des traitements soit intégré à une plateforme plus large dédiée également aux métadonnées sur les données géographiques et au calcul de tâches (projet de l'action de recherche Consul dans laquelle s'inscrit notre travail). Ceci, entre autres, explique pourquoi serveur d'application et serveur Web sont séparés : les métadonnées des traitements ne doivent pas seulement être fournies au format HTML mais aussi au format XML. Cette séparation rend ainsi aisée, par exemple, la construction d'un service Web SOAP dont les réponses encapsuleraient nos métadonnées au format XML.

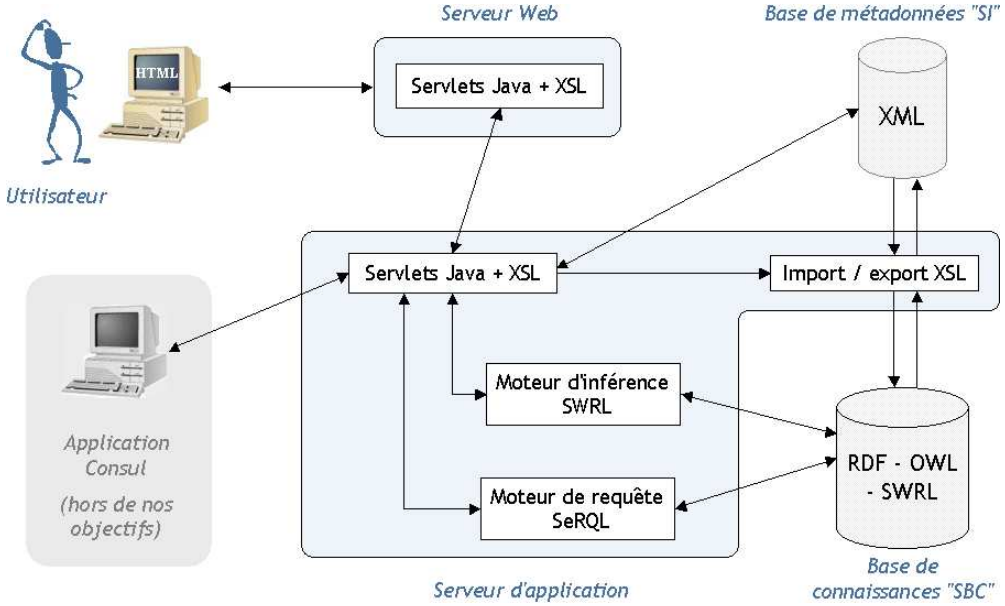


FIG. 5.1 – Architecture de l'application d'accès aux métadonnées

Aucune partie de notre application n'est liée à un système d'exploitation particulier. Les parties "SI" et "SBC" du serveur d'application reposent sur différentes API Java ; la machine virtuelle Java JRE 1.5.0 (*alias* Java 2 version 5.0) est utilisée. Les servlets Java développés fonctionnent avec Tomcat 5.5. Les pages Web statiques de l'application sont fournies par un serveur de pages Web Apache 1.3.27.

5.2 L'application d'accès aux métadonnées – aspect "SI"

Après avoir succinctement indiqué nos principaux choix techniques concernant l'aspect SI de notre application, nous présentons cette dernière en suivant le point de vue de l'utilisateur.

5.2.1 Choix d'implémentation – aspect "SI"

Pour manipuler les différents formats de données nous faisons appel à diverses API Java. En particulier, pour manipuler les documents XML nous utilisons l'API standard Jaxp (Java API for XML Processing). Cette API nous permet d'utiliser XSLT et XPath.

XSLT (eXtensible Stylesheet Language Transformation) est le langage standard de transformation de documents XML. XPath est le langage qui permet d'adresser les éléments du docu-

ment XML à transformer. XSLT et XPath sont des recommandations W3C dont les premières versions ont été rendues publiques en 1999 [W3C99b][W3C99c].

Nous nous servons d'XPath comme d'un langage de requêtes, de la même façon que nous pourrions utiliser SQL si nos métadonnées étaient stockées dans une base de données relationnelle.

XSLT est un langage fonctionnel. Il est bien adapté à l'implémentation d'algorithmes récursifs. Nous en mettons en œuvre dans de nombreux endroits de l'application, pour générer des fichiers d'index inversés[1], des fichiers retrouvant les types parents ou sous-types des ressources (cf. l'exemple des modes d'emplois code A.4 p. 234), des pages HTML représentant des arbres XML dont on ne connaît pas à l'avance la profondeur (cf. fig. 5.6, 5.12 et 6.8 p. 186, 199 et 220). D'une façon générale, les structures arborescentes se prêtent bien à la récursion.

Pour développer nos servlets nous utilisons le package `javax.servlet`. De façon annexe, nous nous assurons de la validité de notre base de métadonnées vis-à-vis du schéma XML avec XML Spy 2005, outil qui implémente toutes les spécifications W3C relatives aux éléments du langage que nous utilisons. Nous effectuons également des contrôles supplémentaires. Néanmoins, en théorie, la saisie *via* l'application ne permet pas d'enregistrer de description invalide.

5.2.2 Navigation et recherche dans la base de métadonnées

Notre application de consultation des métadonnées se présente à l'utilisateur sous la forme d'un site Web. L'utilisateur y accède avec un navigateur Web standard, *via* l'Intranet de l'IGN. La figure 5.2 est une copie d'écran de la page d'accueil. La barre de navigation, située à gauche, offre plusieurs fonctionnalités :

- Le lien *"Navigation dans les index"* mène au diagramme de la page d'accueil (dont les éléments sont cliquables).
- Le lien *"Soumettre une requête"* mène au formulaire montré fig. 5.9 p. 193. L'utilisateur ne fait que remplir ce dernier, il n'utilise aucun langage de requête (*i.e.* il ne saisit pas d'expression de langages comme SQL ou XPath).
- Le lien *"Statistiques"* mène à la page montrée fig. 5.4.
- Le lien *"Index de toutes les ressources"* mène à la liste alphabétique de toutes les ressources indexées dans la base de métadonnées.
- Le champ *"Rechercher"* permet d'effectuer une recherche plein-texte dans la base de métadonnées.

La partie *"Acquisition"* sera discutée au chapitre 6.

La figure 5.3 montre le résultat de la sélection de "Ensemble de traitements" puis de "Logiciel/SIG" : la liste de ces derniers est affichée. La liste peut être triée en prenant comme critère les propriétés affichées (les têtes de colonnes "nom", "(domaine de) fonctionnalités", etc. sont "cliquables") ou celles proposées dans la liste déroulante en haut de l'écran. Remarquons que les flèches situées à gauche des noms de ressources permettent d'en visualiser une courte description.

Les RessourcesTraitements mises à part, la plupart des ressources de notre base de métadonnées sont reliées entre elles par des relations de spécialisation. Lors de la navigation dans les index, il est possible de visualiser les taxinomies constituées sous forme arborescente. La figure 6.1 p. 214 montre un écran affichant la liste des fonctionnalités sous cette forme. Ce

[1] Le principe est simple : étant donné un ensemble d'index décrivant les relations entre ressources, on construit de nouveaux index décrivant les relations inverses. Typiquement, partant de pages Web contenant des listes de mots, on construit les index inverses qui décrivent pour chaque mot les pages Web qui les contiennent.

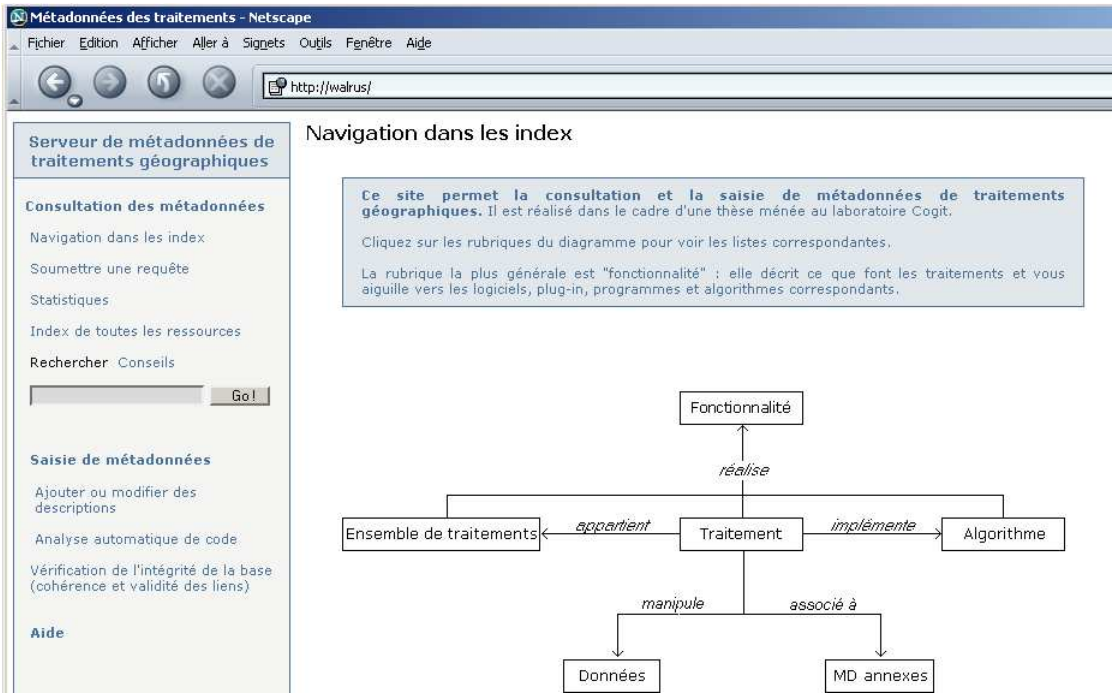


FIG. 5.2 – Page d'accueil de l'application

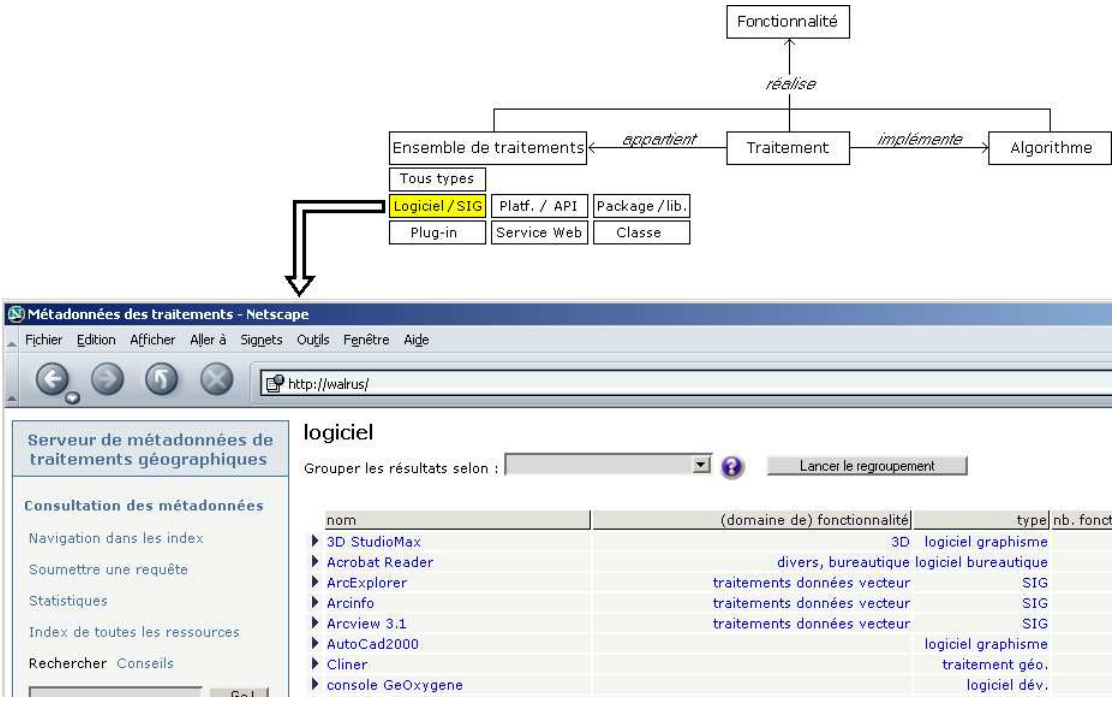


FIG. 5.3 – Visualisation de la liste des logiciels indexés dans la base de métadonnées

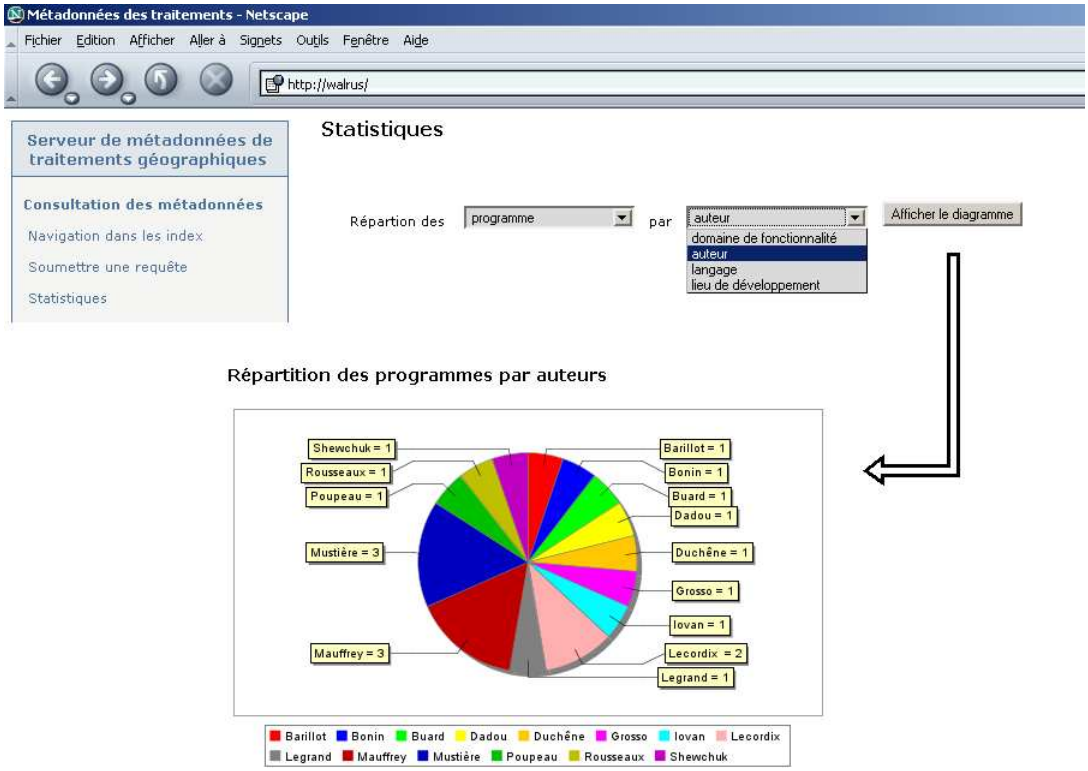


FIG. 5.4 – Affichage de statistiques – Génération dynamique de camemberts JChart

type d'écran est généré dynamiquement avec des feuilles XSL dont les *templates* parcourent récursivement la hiérarchie des ressources (chaque ressource n'indiquant que son parent direct).

L'utilisateur peut naviguer dans les index ; il peut aussi soumettre des requêtes. Celles qui s'effectuent *via* le formulaire proposé sont traitées par la partie SBC de l'application. Celles qui reposent sur la soumission de mots-clés sont plus simples, elles reposent sur la simple recherche plein-texte. Le résultat est la liste de toutes les ressources dont n'importe lequel des éléments de description contient la chaîne de caractère soumise par l'utilisateur. Un des obstacles de ce type de recherche réside dans les problèmes de synonymies ou de multilinguisme. Notre application permet de le surmonter en partie. Voici un procédé s'appliquant de façon générique. L'utilisateur pensant avoir un problème de vocabulaire pour utiliser les bons termes de recherche peut consulter la description des concepts décrits dans la base de métadonnées. Par exemple, l'utilisateur recherche un programme de détection de talwegs. S'il soumet le mot-clé "talweg" le résultat de la recherche ne comportera aucun traitement, mais comportera en revanche le concept "relief". L'utilisateur demande alors de visualiser la description de ce concept. La liste de toutes les ressources liées s'affiche. Parmi elles figure le programme "caractérisation des MNT", qui répond au besoin de l'utilisateur.

Dans notre contexte, la question du tri des résultats en fonction de la popularité des ressources est un aspect secondaire. Nous n'avons pas cherché à le traiter. Néanmoins, si les utilisateurs le souhaitaient, il n'y aurait aucune difficulté à proposer un tri des résultats en fonction du nombre de ressources dont la description "pointe" vers les ressources recherchées (*"popularité d'après les métadonnées"*), ou un tri en fonction du nombre de "clics" effectués depuis une période donnée par les utilisateurs, indice éventuellement pondéré par l'ancienneté de l'accès ou le profil des utilisateurs[2] (*"popularité d'après les utilisateurs"*).

[2]L'identité des utilisateurs est connue. Les servlets de notre application gardent une trace de leurs actions dans des fichiers *log*. Étant dans un contexte Intranet, nous avons stocké la table de correspondances entre les noms des

5.2.3 Visualisation des descriptions de traitements

À partir des listes de ressources obtenues par un des modes de recherche évoqués, l'utilisateur accède à la description d'une ressource particulière. La description du programme Accordéon v.2 telle qu'elle apparaît à l'utilisateur est montrée figure 5.5.

Les pages de descriptions de RessourceTraitement sont organisées selon les cinq facettes qui structurent notre modèle conceptuel. Chaque facette est signalée par une barre horizontale bleue. Les descriptions comportent une sixième barre nommée "Ressources liées", représentant la partie où sont listées toutes les ressources qui font référence à la ressource courante. Par exemple, à la fin d'une description d'une librairie se trouvent indiquées toutes les RessourceTraitement qui l'utilisent. Ce type d'information est simple mais précieux.

Pour plus de la moitié des éléments de description, les valeurs sont des références à des ressources. De façon systématique, ces valeurs apparaissent sous forme de lien hypertexte (en bleu, ou en violet pour les liens déjà visités)[3]. Les autres valeurs, de type simple (texte, nombres entiers et réels, date, booléens) apparaissent sous forme de texte simple (en noir). Les illustrations sont un cas particulier. Certaines ne sont que des images *raster* non indexées en tant que ressources. C'est le cas de l'image, figure 5.5, où les routes sont symbolisées en rouge. D'autres illustrations, au contraire, sont des ressources de type Echantillon.

Certains échantillons sont de simples images *raster* (au format bitmap, GIF, Jpeg ou PNG). Les autres échantillons – il est important de le souligner – sont des jeux de données réels au format shape. C'est le cas des deux échantillons qui illustrent la description du programme Accordéon 5.5. Le jeu de données avant traitement est issu de la version d'octobre 2002 de la BD Carto et représente des routes de la région de Nice. Il a été généralisé avec le module AGENT du SIG Lamps2 en février 2006 ; Accordéon est un des programmes qui a été appliqué. Les jeux de données sont stockés dans la base de métadonnées. Ils sont visualisés dans les pages HTML de notre application grâce à des *applets*[4] Java Geotools inclues dans des *frames HTML*.

utilisateurs et le nom ou l'IP fixe de leur machine, ces derniers étant récupérés par les servlets avec la méthode `getRemoteHost()` de l'objet `HttpServletRequest` créé à chaque accès à une page de l'application (pour les pages statiques comme la page d'accueil, l'utilisateur est au préalable redirigé automatiquement vers un servlet dédié au *log* grâce à une instruction Javascript `location.replace(page)`).

[3] Les propriétés des données sont des éléments de descriptions, mais ce sont aussi des ressources. Elles sont donc représentées par des liens hypertextes menant à leur description.

[4] *appl*ication widg*et*, un *widget* étant un élément graphique d'interface (contraction de *windows gadget*, mais néanmoins utile ici).

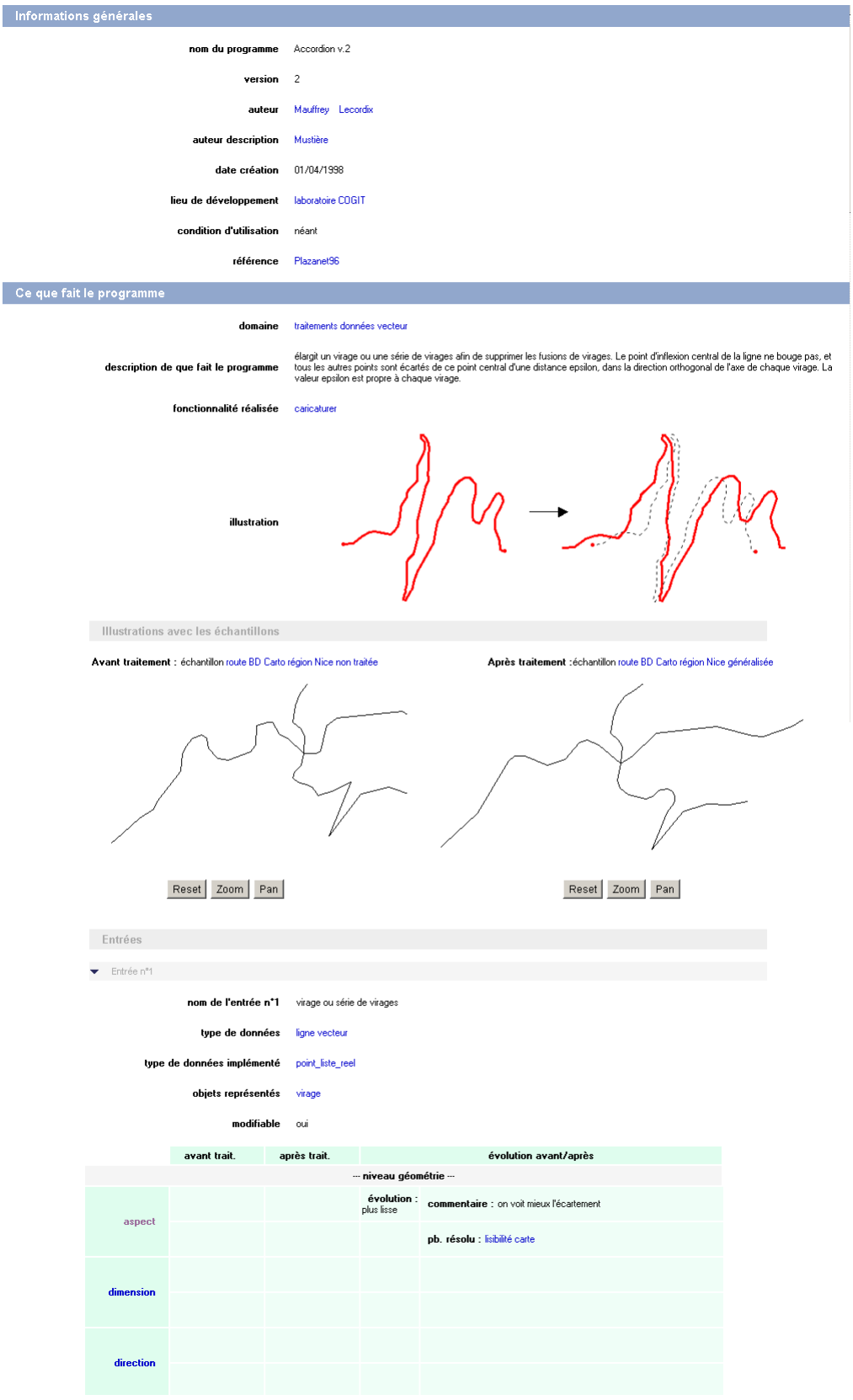


FIG. 5.5 – Description du programme Accordéon (haut de l'écran)

5.2.4 Visualisation des descriptions de modes d'emploi

Les descriptions des modes d'emploi sont visualisées de façon isolée (certaines informations sont néanmoins également affichées directement dans les descriptions de traitements). La figure 5.6 montre la description du mode d'emploi correspondant au besoin exposé figure 1.9 p. 26. Les étapes sont présentées sous forme arborescente. Dans le mode d'emploi figure 5.7, les étapes ne sont pas décrites suivant le formalisme de notre modèle : la page HTML du manuel de GéoConcept 5 a simplement été incluse automatiquement lors de la phase d'acquisition. En attendant une éventuelle traduction dans notre modèle, cette possibilité d'intégrer des documents existants est intéressante. Bien sûr, dans ce dernier cas, l'adaptation au contexte d'utilisation est interdite. L'adaptation est, en revanche, possible dans le cas "normal" ; nous verrons bientôt comment.

Les parties supérieures des écrans fig. 5.6 et 5.7 comportent des tableaux indiquant la "parenté" des modes d'emploi et les connaissances et pré-requis hérités. Ce choix de présentation est fortement inspiré des documentations API classiques dont la figure 5.8 montre un exemple. De la même façon qu'une classe hérite des propriétés de ses super-classes, un mode d'emploi hérite des concepts et pré-requis de ses modes d'emploi parents.

Le code qui permet la génération du tableau est montré p. 234.

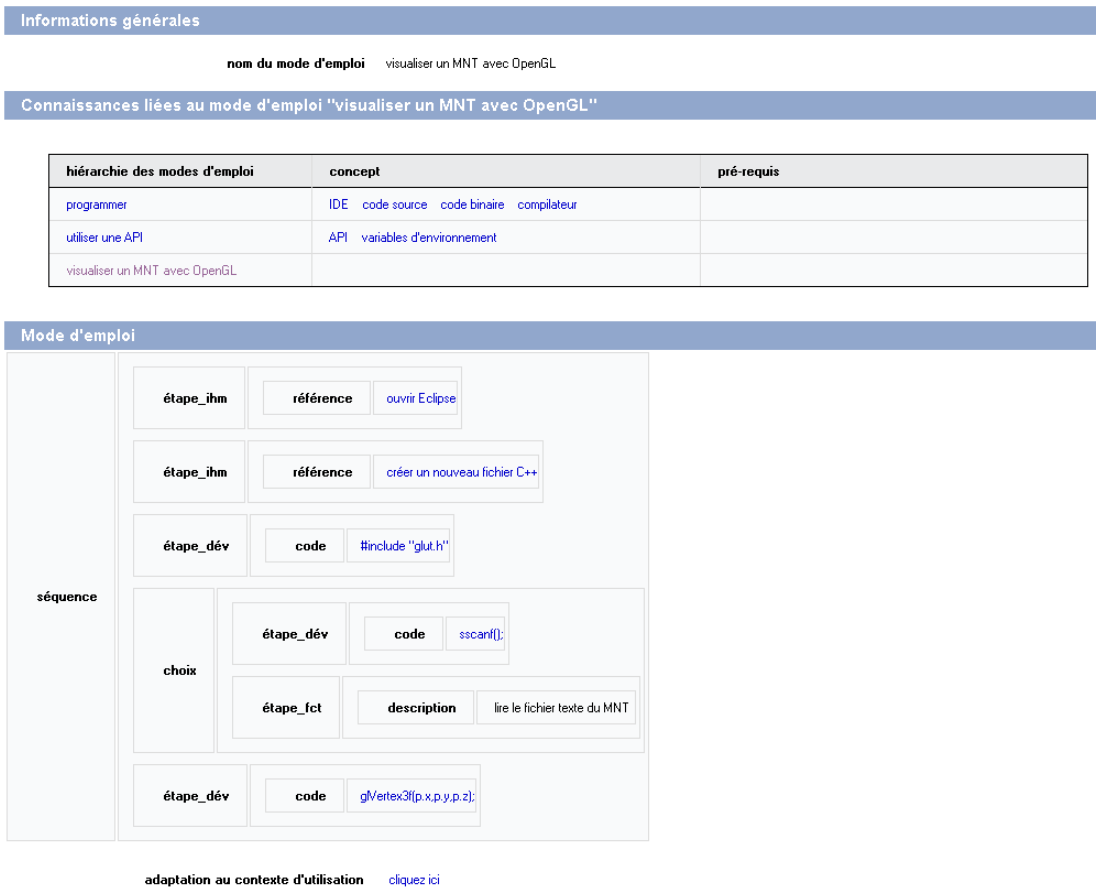


FIG. 5.6 – Mode d'emploi "Visualisation MNT avec OpenGL"

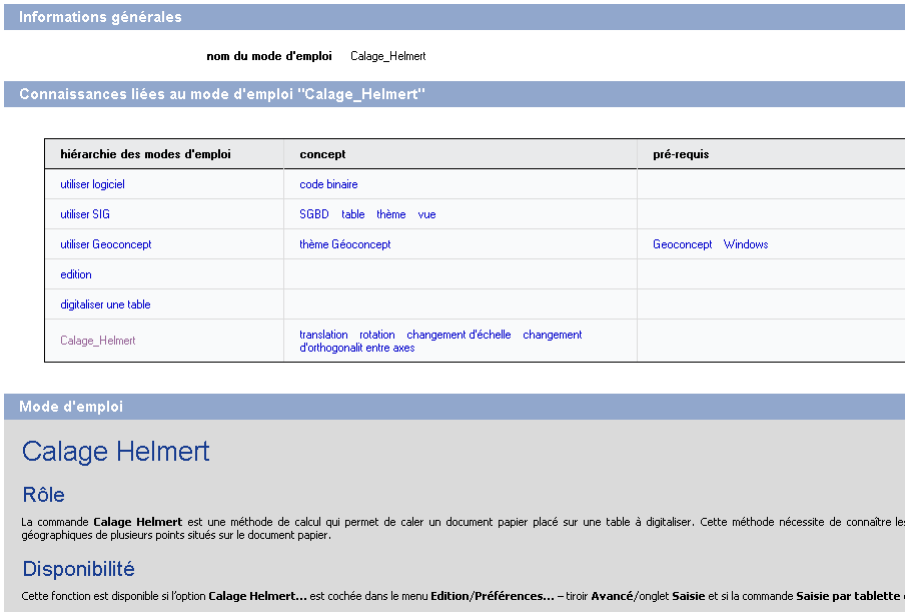


FIG. 5.7 – Mode d'emploi de la FonctionLogiciel "calage Helmert" du SIG Géoconcept 5.0

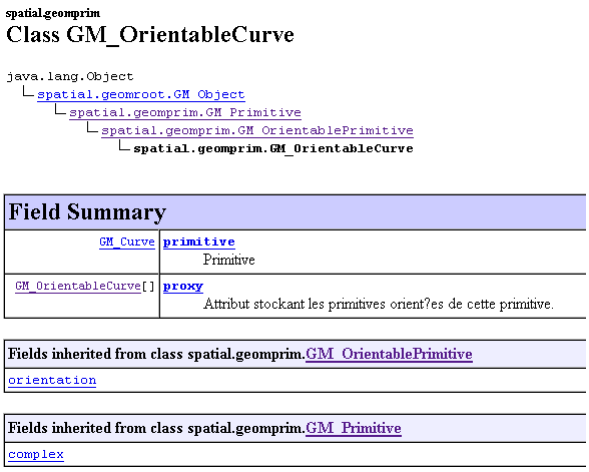


FIG. 5.8 – Extrait d'une page de "Javadoc" classique

5.2.5 Gestion des relations d'héritage dans la partie "SI" de l'application

À plusieurs endroits de l'application, nous avons besoin d'obtenir des informations sur la "généalogie" des ressources, *i.e.* la liste des ressources qui les spécialisent ou les subsument. Par exemple, l'utilisateur veut visualiser les échantillons de type vecteur. Il faut lui présenter les échantillons renseignés comme étant de ce type mais aussi de ses sous-types. Compte tenu du fait que, dans notre base de métadonnées, chaque description de ressource ne renseigne que sa parenté directe, trois solutions sont possibles pour obtenir les informations sur la généalogie. Nous adoptons la solution 3 pour les besoins simples de la partie "SI" de l'application ; la solution 2 est utilisée seulement pour la partie SBC.

Solution 1 : requêtes naïves avec jointures

La première solution est une mauvaise solution. Elle consiste à parcourir naïvement la généalogie des ressources au moyen du langage de requêtes choisi (en l'occurrence XPath, mais le problème serait exactement le même avec, par exemple, SQL si nous avions décidé de stocker nos métadonnées dans une base de données relationnelle. Pour plus de clarté pour le lecteur, nous adoptons d'ailleurs SQL pour l'exemple qui suit).

Ainsi, par exemple, pour retrouver les descendants de la fonctionnalité "TI" (traitement d'image) on pourrait essayer d'écrire une requête de la forme :

```
SELECT f1
FROM Fonctionnalité f1, f2, f3, f4
WHERE
   ( f1.type = f2 and f2.nom = "TI" )
or ( f1.type = f2 and f2.type = f3 and f3.nom = "TI" )
or ( f1.type = f2 and f2.type = f3 and f3.type = f4 and f4.nom = "TI"
)
```

Mais aussi loin que l'on développe les requêtes de ce type, elles ne pourront retrouver des descendants d'une profondeur quelconque. En effet, l'algèbre relationnelle n'a pas le pouvoir d'exprimer des règles de fermetures transitives de relations. Nous écartons donc la solution qui consisterait à écrire des requêtes telles que celles ci-dessus, même si, en pratique elles auraient pu localement convenir si l'on avait fait l'hypothèse réaliste mais peu satisfaisante d'un nombre borné de profondeurs dans les généalogies.

Solution 2 : couplage avec le SBC

La deuxième solution est la plus puissante. Elle consiste à faire appel aux capacités de la partie SBC de notre application. C'est la solution que nous employons pour calculer les requêtes soumises par l'utilisateur *via* le formulaire dédié et pour le calcul des adaptations des modes d'emploi. Pour les besoins courants de l'application les développements, le couplage est possible mais lourd. Concrètement, les feuilles XSL utilisées par les servlets Java de la partie SI ne peuvent invoquer directement le SBC[5], elles ne peuvent accéder aux résultats de ce dernier que *via* des fichiers XML qu'il faut faire générer par les servlets. Le processus est assez lourd lors du développement et assez peu efficace à l'exécution. Nous préférons donc adopter la solution 3, plus élégante.

[5]En fait, en 2001, la version 1.1 du langage XSLT proposée par un groupe de travail du W3C a introduit une nouveauté avec la possibilité d'utiliser des scripts dans divers langages (Javascript et VBscript notamment). Cette possibilité a été critiquée car elle remet en question le principe de neutralité de XSL vis-à-vis des plateformes [van01]. Néanmoins, par exemple, la version 3 de la librairie standard de Miscrosoft pour la manipulation de documents XML (msxml3.dll) permet l'exécution de scripts au sein de feuilles XSLT, et par ce biais l'invocation de librairies ou programmes externes. Le souci de sauvegarder la portabilité de notre application était une raison suffisante pour que nous nous interdisions de recourir aux scripts XSLT.

Solution 3 : génération de résultats pré-calculés

La troisième solution consiste à générer des fichiers XML décrivant pour chaque ressource les informations sur ses ascendants et descendants. Ces fichiers sont construits de façon simple et efficace au moyen de feuilles XSL parcourant récursivement les relations de spécialisation. En annexes, les codes A.4, A.5 et A.6 montrent respectivement la feuille XSL qui génère le fichier indiquant la hiérarchie des modes d'emploi, les extraits des descriptions initiales des modes d'emploi et les extraits du fichier généré.

5.2.6 Validation et contrôle d'intégrité dans la partie "SI" de l'application

L'administrateur de l'application doit pouvoir s'assurer de la validité de la base de métadonnées vis-à-vis du modèle. En plus de la spécification de la structure des éléments XML, le langage XML Schema permet de définir des contraintes d'intégrité sur l'unicité des identifiants et sur l'existence de ceux auxquels il est fait référence (codes 5.1 et 5.2).

Dans notre modèle XML Schema, chaque élément-ressource possède un attribut `id`. Le code 5.1 montre deux façons de s'assurer de leur unicité.

```
<xsd:unique name="uniciteId">
  <xsd:selector xpath="."/>
  <xsd:field xpath="@id"/>
</xsd:unique>

<!-- Alternative : utilisation du type prédéfini xsd:ID -->
<xsd:attribute name="id" type="xsd:ID"/>
```

Extrait de code 5.1: XSD – Contrainte d'unicité des identifiants

Le code 5.2 spécifie que les valeurs des éléments `réalise` sont des clés qui doivent être des identifiants de fonctionnalités (existence dans les éléments `fonctionnalité`, chemin @id).

```
<xsd:key name="identifiantFct">
  <xsd:selector xpath="fonctionnalité"/>
  <xsd:field xpath="@id"/>
</xsd:key>
<xsd:keyref name="existenceId" refer="identifiantFct">
  <xsd:selector xpath="réalise"/>
  <xsd:field xpath="."/>
</xsd:keyref>
```

Extrait de code 5.2: XSD – Contrainte d'existence des identifiants référencés

Les validateurs XML Schema n'implémentent pas toujours toutes les spécifications du W3C. Nous supposons que c'est pour cette raison que nous n'avons pas réussi à invalider des exemples erronés à dessein avec XML Spy 2005. D'autres validateurs existent, mais il nous a paru plus pratique de développer nous-mêmes des feuilles XSL détectant les violations des deux types de contraintes mentionnées ci-dessus (codes 5.3 et 5.4).

De façon similaire, diverses feuilles XSL peuvent être écrites afin de vérifier la cohérence entre les ressources. Par exemple, on pourrait vérifier que les fonctionnalités et les types de données des entrées et sorties des traitements codés sont cohérents avec ceux des algorithmes qu'ils implémentent.

```
<xsl:for-each select="//node()[@id]">
  <xsl:variable name="vId" select="@id"/>
  <xsl:variable name="vPosition" select="position()"/>
  <xsl:for-each select="//node()[@id]">
    <xsl:if test="position()!= $vPosition and @id=$vId">
    erreur : <xsl:value-of select="$Id"/> inconnu.<br/>
    </xsl:if>
  </xsl:for-each>
</xsl:for-each>
```

Extrait de code 5.3: XSL – Vérification de la contrainte d'unicité des identifiants

```
<h2>formats</h2>
<table>
  <xsl:for-each select="//text()[starts-with(., 'for')]">
    <xsl:variable name="strReference">
      <xsl:value-of select="."/>
    </xsl:variable>
    <xsl:if test="not(//@id= $strReference)">
      <tr>
        <td class="erreur">
          <xsl:value-of select="$strReference"/>
        </td>
      </tr>
    </xsl:if>
  </xsl:for-each>
</table>
```

Extrait de code 5.4: XSL – Vérification de la contrainte d'existence des identifiants référencés

5.3 L'application d'accès aux métadonnées – aspect "SBC"

Ce qui vient d'être évoqué concerne la partie "Système d'information" de notre application. La partie "SBC" repose sur d'autres principes et sur d'autres choix de mise en œuvre.

5.3.1 Choix d'implémentation – aspect "SBC"

Au début des années 2000, la question au sein de la communauté de l'ingénierie des connaissances (IC) était encore *"quel langage pour définir les ontologies du Web sémantique ?"*. La réponse est maintenant connue ; OWL semble adopté. Pour des raisons de compromis entre calculabilité et expressivité, de nouvelles distinctions de sous-familles OWL autres que Lite, DL et Full verront peut-être le jour mais cela ne remettra pas forcément en cause la base du langage.

À présent d'autres questions en suspens occupent la communauté de l'IC : *"quels langages pour effectuer des requêtes dans les ontologies OWL et les bases de connaissances RDF ? quels langages pour définir les règles ? quels outils pour exploiter les nouveaux langages en question et pour réaliser les inférences attendues ?"*.

La variété des réponses possibles montre que le Web sémantique est encore en gestation. Les couches du *layer cake* ne s'implémentent que de façon progressive.

Les besoins de standardisation du Web sémantique concernent les langages. L'implémentation des outils est *a priori* laissée libre, pourvu que les spécifications des langages soient respectées. Dans le cas des langages de requêtes et de règles, les outils définissent parfois leurs propres langages. Choix des langages et choix des outils sont alors liés. Nous présentons d'abord le choix des moteurs d'inférences OWL et SWRL utilisés dans notre application, puis le choix du langage de requête associé.

Choix du moteur d'inférence

Il existe beaucoup de raisonneurs, spécifiques ou non à OWL. Tous ne se valent pas. Certains ne raisonnent que sur les individus des ontologies, d'autres sur les concepts (capacités correspondant respectivement aux T-Box et aux A-Box des LD), et ce, de façon plus ou moins complète. Par exemple, FaCT++ (Fast Classification of Terminologies, implémenté en C++) raisonne sur les concepts, Pellet et Racer (Renamed ABox and Concept Expression Reasoner) permettent aussi de raisonner sur les instances de concepts. Selon le site de Mindswap, Pellet a été en 2003 le premier raisonneur OWL-DL sûr et complet (*"sound and complete"*) construit[6].

Pour nos premières expérimentations en 2005 nous avons utilisé Jena 2.2, développé par Hewlett Packard. Ce choix n'a pas été motivé par une raison particulière, sinon que la présentation sous forme d'API Java nous a paru simple à utiliser. Nous avons cherché un autre raisonneur lorsque nous avons voulu définir et exécuter des règles avec variables. En effet, Jena 2.2 permet bien l'usage de règles, mais il s'agit des règles JenaRules et non SWRL, langage que nous avions choisi d'utiliser. Nous sommes donc partis en quête d'un moteur d'inférence SWRL. Nous en avons trouvé très peu. Notre choix s'est arrêté sur un programme développé à l'université libre de Berlin [Mei05] et reposant sur le raisonneur Sesame 1.2.1 développé par la société Aduna [Ope05]. Le nom "Sesame" est une allusion au mot de passe ouvrant la grotte dans "les contes des 1001 nuits".

Sesame est une plateforme Java *open source*[7] pour stocker, effectuer des requêtes et raisonner sur une base de documents RDF et RDF Schema [Ope05].

[6] Site du projet Mindswap (Maryland Information and Network Dynamics Lab Semantic Web Agents Project), `http://www.mindswap.org/2003/pellet/`.

[7] Sesame est disponible sous licence LGPL, `http://www.openrdf.org/`

Ni Jena 2.2 ni Sesame 1.2.1 ne sont capables de saturer une base de connaissances OWL DL (*i.e.* de mener toutes les inférences possibles prévues par la sémantique OWL DL). Quant au moteur SWRL que nous utilisons, nous allons voir qu'il souffre malheureusement de limitations rédhibitoires pour nos besoins.

Depuis notre choix de moteur SWRL, d'autres sont apparus. Par exemple, en décembre 2005 est sortie la version 1.9 du raisonneur commercial RacerPro, commençant pour la première fois à implémenter un moteur SWRL ([KG05], §2.6). Le raisonneur KAON (*KA*rlsruhe *ON*tology and Semantic Web infrastructure) est également capable, dans sa version 2, d'exécuter des règles SWRL[8].

D'autres nouveaux outils du Web sémantique sont référencés, notamment sur le site du W3C[9].

Choix du langage d'interrogation

Le langage de requêtes que nous utilisons est SeRQL (Sesame RDF Query Language). Il a été développé par Aduna, en tant que composant de la plateforme Sesame. SeRQL combine les caractéristiques de plusieurs autres langages de requêtes tels que RQL, RDQL, N-Triples et N3 [BL05][10].

Ce langage est en phase de maturation. Nous avons commencé par utiliser la version 1.2, puis la version 2 bêta en décembre 2005. SPARQL (Sparql Protocol And RDF Query Language) est un autre langage de requêtes RDF. Il est, lui, une recommandation du W3C. Nous avons néanmoins utilisé SeRQL en raison de son association à Sesame.

Construction de la base de connaissances

Dès lors que le choix d'une architecture duale SI/SBC s'est imposé, un de nos objectifs a été d'automatiser les opérations d'importation et d'exportation. Pour cela nous avons écrit des feuilles XSL. Les taxinomies simples de concepts (ontologies légères) sont exportées facilement de XML vers OWL. La simple indication du type des individus s'effectue également sans problème : les instances XML des RessourceTraitement sont transformées en ressources RDF instances des classes OWL.

En revanche, les axiomes de classes et le détail des descriptions ont dû être saisis manuellement *via* l'éditeur Protégé, version 3.0 bêta puis 3.1. Les axiomes de classes, hormis certains pour les FamilleTraitement, sont absents de notre base de métadonnées SI : notre modèle XML Schéma n'est pas un langage de définition d'ontologies lourdes (ce qui supposerait des développements déraisonnables pour notre application Web, sachant que même une application aussi importante que Protégé 3.1 ne permet pas la création d'ontologies aussi complexes que celles normalement possibles en OWL[11]).

Concernant le détail des descriptions, le temps nous a manqué pour achever de développer les feuilles XSL permettant le passage SI vers SBC. La gestion des identifiants des ressources RDF ne rend pas la tâche insurmontable mais tout de même un peu compliquée. Inversement, pour le passage SBC vers SI, ce sont les différentes syntaxes RDF possibles qui compliquent la traduction (cf. les codes RDF équivalents 5.7 et A.7).

Concernant la traduction des règles de notre format XML en SWRL, l'obstacle essentiel concerne l'adressage des métadonnées avec le passage de la notation "DOM" à la notation "prédicats". Au moment de l'achèvement de la rédaction de ce mémoire nous n'avions pas surmonté cet obstacle, mais nous pensons que l'objectif n'est pas hors d'atteinte. Pour la saisie manuelle des règles SWRL nous avons utilisé le plug-in fourni avec Protégé 3.1.

[8] `http://kaon2.semanticweb.org/`

[9] `http://www.w3.org/2004/OWL/`, section "Tools, Projects and Applications".

[10] Le domaine évolue. On trouve les archives du groupe de travail du W3C "RDF Data Access Working Group" sur `http://www.w3.org/2001/sw/DataAccess/`

[11] Par exemple, l'interface de Protégé 3.1 ne permet pas de sélectionner des concepts pour affecter une valeur à une `ObjectProperty`. Pourtant, cela est normalement possible en OWL Full.

5.3.2 Simulation du raisonnement ER 1 – Recherche de traitements

Nous allons suivre pas à pas le déroulement de l'exemple de raisonnement ER 1. Nous l'avons déjà présenté pages 20 et 120.

L'utilisateur accède à la page d'accueil de l'application. Il sélectionne le lien "soumettre une requête" dans la *frame* de navigation. Il remplit le formulaire de saisie de requête qui s'affiche alors, en renseignant, à l'aide des listes déroulantes proposées, le lieu de développement, le type de l'entrée et le type de la sortie de la RessourceTraitement recherchée.

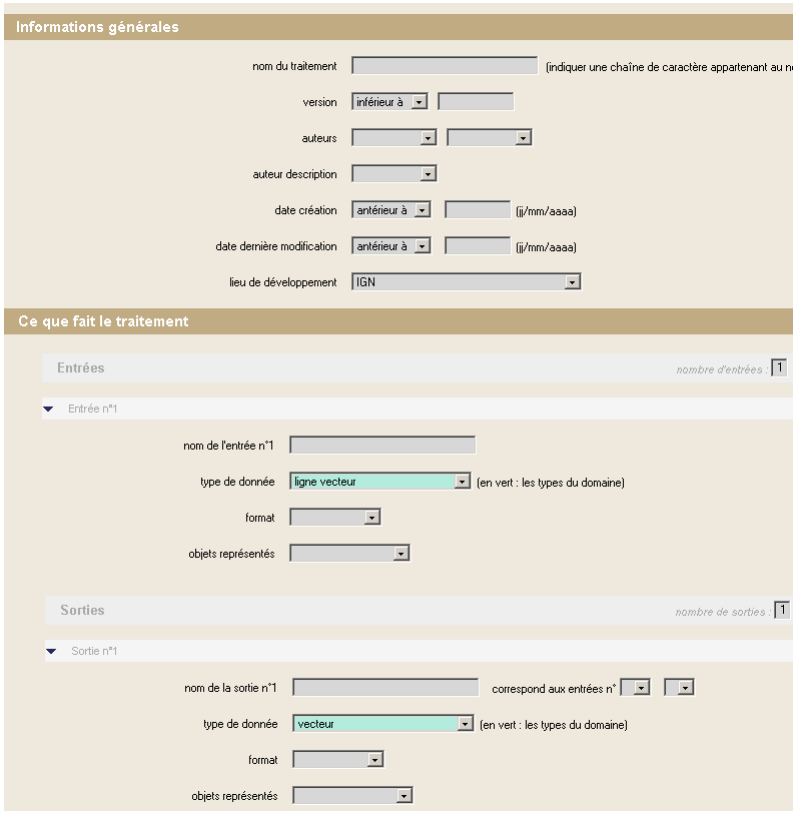


FIG. 5.9 – Formulaire d'expression de requêtes multi-critères

L'utilisateur soumet la requête ER 1 *via* le formulaire montré fig. 5.9. La requête en SeRQL est générée (code 5.5).

```
SELECT trait
FROM {trait}   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type {o},
     {trait}   <http://www.ign.fr#lieuDeDeveloppement>          {ldd},
     {trait}   <http://www.ign.fr#entree>                       {entree},
     {entree}  <http://www.ign.fr#typeAbst>                     {tdAbstE},
     {trait}   <http://www.ign.fr#sortie>                       {sortie},
     {sortie}  <http://www.ign.fr#typeAbst>                     {tdAbstS}

WHERE o       = <http://www.ign.fr#RessourceTraitement>
AND   ldd     = <http://www.ign.fr#IGN>
AND   tdAbstE = <http://www.ign.fr#vecteurLigne>
AND   tdAbstS = <http://www.ign.fr#vecteur>
```

Extrait de code 5.5: SeRQL – Requête ER 1

À ce stade du déroulement de l'exemple, si on exécute la requête SeRQL on n'obtient aucun résultat car la base de connaissances ne contient que les définitions OWL et assertions RDF exposées codes 5.6 et 5.7. La notation RDF adoptée est celle de triplets "à plat". Elle nous semble plus claire que la notation "arborescente" équivalente des triplets. À titre de comparaison, nous donnons en annexe la version "arbre" du code 5.7 page 236 (code A.7). La comparaison entre les deux notations est nécessaire pour décider comment écrire nos feuilles XSL d'importation et exportation SI/SBC. Elle est intéressante car elle montre pourquoi l'exploitation d'un fichier XML arborescent conforme à un schéma *ad hoc* est nettement plus aisée que celle de fichiers RDF à la structure de graphes et, qui plus est, à la syntaxe variable.

```
<!-- Remarque : les id des classes commencent par des majuscules -->
<owl:Class rdf:ID="Programme">
  <rdfs:subClassOf rdf:resource="#Traitement"/>
</owl:Class>
<owl:Class rdf:ID="Organisation"/>
<owl:Class rdf:about="#Vecteur">
  <rdfs:subClassOf rdf:resource="#TypeDonneeAbst"/>
</owl:Class>
<owl:Class rdf:ID="VecteurLigne">
  <rdfs:subClassOf rdf:resource="#Vecteur"/>
</owl:Class>
<owl:Class rdf:ID="VecteurSurface">
  <rdfs:subClassOf rdf:resource="#Vecteur"/>
</owl:Class>
<!-- ... -->
 <owl:Class rdf:about="#VecteurLigne">
   <owl:disjointWith rdf:resource="#VecteurSurface"/>
 </owl:Class>
<!-- ... -->
```

Extrait de code 5.6: OWL – Base de connaissances avant inférences (partie terminologique)

Pour inférer de nouveaux triplets RDF et les ajouter à notre base de connaissances, il faut appliquer les trois règles SWRL du tableau 5.1. Nous réalisons cela en exécutant les instructions du programme local.java développé à l'université de Berlin[12].

[12] http://www.inf.fu-berlin.de/inst/ag-nbi/research/swrlengine/local.java

```
  <Programme rdf:ID="buffer">
   <lieuDeDeveloppement rdf:resource="#COGIT"/>
   <entree rdf:resource="#entBuffer"/>
   <sortie rdf:resource="#sorBuffer"/>
 </Programme>

 <Organisation rdf:ID="IGN"/>
 <Organisation rdf:ID="ServiceRechercheIGN">
   <appartientOrganisation rdf:resource="#IGN"/>
 </Organisation>
 <Organisation rdf:ID="COGIT">
   <appartientOrganisation rdf:resource="#ServiceRechercheIGN"/>
 </Organisation>

 <Donnee rdf:ID="entBuffer">
   <typeAbst rdf:resource="#vecteur"/>
 </Donnee>
 <Donnee rdf:ID="sorBuffer">
   <typeAbst rdf:resource="#vecteurSurface"/>
 </Donnee>

 <!-- individus "uniques" de concepts créés pour remplacer ces
derniers lorsqu'on value la propriété "typeAbs". Ainsi on évite de
tomber dans OWL Full. -->
 <Vecteur rdf:id="vecteur"/>
 <VecteurLigne rdf:ID="vecteurLigne"/>
 <VecteurSurface rdf:ID="vecteurSurface"/>

  <swrl:Imp rdf:ID="r_appartientLieuDeDev">
    <swrl:head>
    <!-- ... -->
```

Extrait de code 5.7: RDF – Base de connaissances avant inférences (partie assertionnelle)

Définition de la hiérarchie des classes : on utilise l'élément du langage RDFS `subClassOf`. La transitivité de la relation est définie nativement. Il n'y a pas de fait à ajouter explicitement, le moteur de requête SeRQL est capable de retrouver la hiérarchie.

Transitivité de la propriété appartientOrganisation : on la spécifie en utilisant l'élément du langage OWL `transitiveProperty`. appartientOrganisation est une `objectProperty` pouvant relier deux Organisation.

Transitivité de la propriété sousType : on la spécifie en utilisant l'élément du langage OWL `transitiveProperty`. sousType est une `objectProperty` pouvant relier deux TypeAbst.
Pour les deux propriétés précédentes, il n'y a pas de fait à ajouter explicitement car le moteur de requête SeRQL est capable d'exploiter la transitivité.

Règle "r_appartientLieuDeDev" : si un traitement t est développé dans une organisation *o1* et que *o1* appartient à une organisation *o2*, alors le traitement t est développé dans *o2*. Le code SWRL est donné p. 176 (code 4.23). Voici une notation plus compacte :

$$lieuDeDeveloppement(?t, ?l1) \wedge appartientOrganisation(?l1, ?l2) \rightarrow lieuDeDeveloppement(?t, ?l2)$$

Règle "r_typeAbstEntree" : si un traitement t a pour entrée une donnée d, que d est de type abstrait *tda1*, et que *tda1* est un sous type de *tda2*, alors d est de type abstrait *tda2*.

$$entree(?t, ?d) \wedge typeAbst(?d, ?tda1) \wedge sousType(?tda1, ?tda2) \rightarrow typeAbst(?d, ?tda2)$$

Règle "r_typeAbstSortie" : la règle est identique à la règle précédente, à ceci près qu'elle porte sur les sorties et que les types sont inversés dans la 3ème prémisse.

$$sortie(?t, ?d) \wedge typeAbst(?d, ?tda1) \wedge sousType(?tda2, ?tda1) \rightarrow typeAbst(?d, ?tda2)$$

Tab. 5.1 – Propriétés RDFS/OWL et règles SWRL utilisées pour ER 1

Les faits suivants sont générés :

– *lieuDeDéveloppement(Buffer, IGN)* *(règle appartientLieuDeDev)*
– *typeAbst(entBuffer, Vecteur)* *(règle r_typeAbstEntree)*
– *typeAbst(sorBuffer, VecteurSurface)* *(règle r_typeAbstSortie)*

Le moteur SWRL utilise les capacités de Sesame pour mener toutes les inférences nécessaires. Il n'ajoute pas dans la base toutes les connaissances inférées – on ne le lui demande pas –, il ajoute seulement les conclusions des règles déclenchées. Par exemple, l'assertion "appartientOrganisation(COGIT, IGN)" a été correctement utilisée mais n'a pas été ajoutée à la base de connaissances.

La requête SeRQL ER 1 peut maintenant être soumise au moteur d'interrogation de Sesame (cf. code A.13 p.240). Les identifiants des RessourceTraitement résultats – le seul programme buffer.java, en l'occurrence – sont passés en paramètre de la feuille XSL qui récupère les descriptions dans la partie SI. Une dernière feuille XSL, enfin, génère le HTML qui est envoyé au client (fig. 5.10).

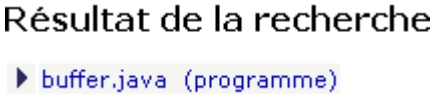


FIG. 5.10 – Résultat ER 1

Commentaires

Pour réussir à faire fonctionner ce type de requête nous avons été contraints de faire plusieurs choix de développement qui méritent quelques commentaires.

La contrainte la plus gênante à nos yeux a été de devoir créer l'`ObjectProperty` *sousType* pour les règles du tableau 5.1 et des individus représentant les concepts pour le code 5.7. Sans ces artifices, le moteur d'inférence que nous utilisons n'aurait pas fonctionné[13].

Nous aurions préféré n'utiliser que des concepts et le constructeur `rdfs:subClassOf`. Cela nous a été interdit car d'une part, donner à des propriétés des concepts pour valeur fait basculer dans OWL Full pour lequel il n'existe pas de raisonneurs complets, d'autre part, nous avons échoué à utiliser des prédicats "isInstanceOf" ou "isSubclassOf" dans les prémisses de nos règles SWRL.

Un autre point est potentiellement problématique. Il concerne la règle *r_typeAbstEntree* (tab. 5.1) : on pourrait craindre que des faits erronés soient générés. Par exemple, cette règle implique qu'une entrée de type *vecteur* est à la fois de type *vecteurPoint* et de type *vecteurLigne*. Or, les concepts *VecteurPoint* et *VecteurLigne* sont déclarés disjoints dans l'ontologie OWL. La base de faits devient donc inconsistante. En fait, on s'accommode de cet inconvénient car ici les inférences sont contextuelles à la requête. Elles ne sont pas utilisées en dehors. Cette notion d'interprétation contextuelle qui nous contraint à procéder de la sorte est illustrée tab. 3.1, p. 131.

[13] Les créateurs du moteur précisent : "OWL reasoning beyond RDF Schema inheritance is not considered in the current version of the engine, yet it is subject of future work." [Mei05].

Un dernier commentaire est à formuler au sujet de la démonstration de l'exemple ER 1. Notre application n'a pu générer automatiquement le code RDF et SWRL requis à partir de la base de métadonnées au format SI. La mise en œuvre de nouveaux exemples nécessitera l'intervention d'un administrateur de l'application pour écrire "à la main", ou *via* un éditeur comme Protégé, le code RDF & SWRL.

5.3.3 Simulation du raisonnement ER 2 suite – Classification de problème

Le raisonnement ER 2 suite a pour but de déterminer que les problèmes d'empâtement sont des problèmes de lisibilité (cf. p. 122).

Nous avons mis en œuvre ce raisonnement de classification en utilisant l'API Jena 2.2. Le moteur d'inférence de Sesame 1.2, qui implémente seulement la sémantique du langage RDF-S, n'est pas être capable d'effectuer les classifications reposant sur la sémantique que OWL ajoute à RDF-S. Lors de nos tests nous n'avons utilisé les capacités d'inférence RDF-S de Sesame que par l'intermédiaire du moteur SWRL. Les requêtes SeRQL, que nous avons exécutées *via* l'API de Sesame, ne mettent pas en œuvre d'inférences.

Les expérimentations avec Jena 2.2 ont été réalisées par le biais de programmes Java sans liens avec notre application. Les brancher pourra faire l'objet de développements futurs.

Nous introduisons ici une variante par rapport à l'énoncé du raisonnement donné p. 122 : nous créons un individu "problème empâtement virage serré isolé" instance du concept "empâtement" (cf. fig. 5.11). Les concepts de l'ontologie, notamment l'axiome de la classe "généralisation cartographique", sont inchangés.

Le code A.14 donné en annexe p. 240 effectue la classification et indique ce que nous attendions, à savoir que l'individu en question est un problème de lisibilité.

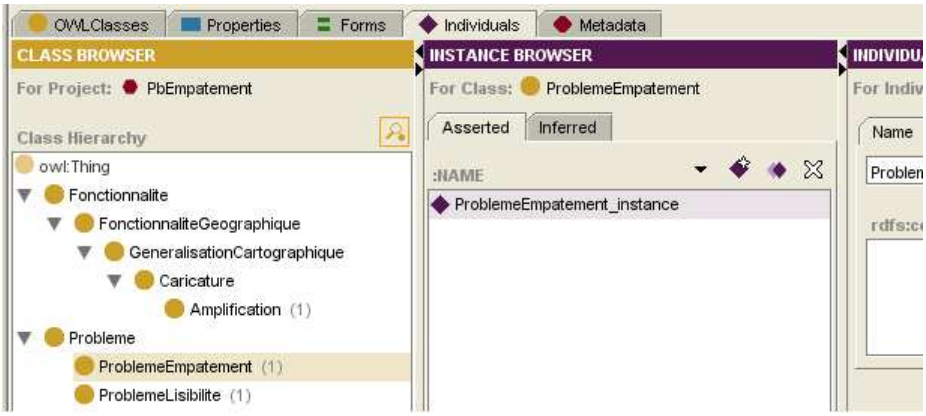


FIG. 5.11 – Exemple ER 2 suite – définition de l'individu à classifier (édition avec Protégé 3.1)

5.3.4 Simulation du raisonnement ER 3 – Adaptation de mode d'emploi

Nous allons maintenant suivre au pas à pas le déroulement de l'exemple de raisonnement ER 3. Nous l'avons déjà présenté pages 21 et 123. Le scénario correspond au diagramme de séquence défini page 137.

Du point de vue de l'utilisateur

Après avoir soumis une requête, navigué dans les index ou suivi un lien à partir d'une page quelconque de l'application, l'utilisateur accède à la description du logiciel FreeWRL (fig. 5.12). Il demande la description d'un mode d'emploi adapté à son contexte d'utilisation en sélectionnant le lien prévu à cet effet. Le formulaire de saisie du contexte d'utilisation s'affiche (fig. 5.13). L'utilisateur le remplit et le soumet. Le mode d'emploi adapté s'affiche (fig. 5.14).

Figure 5.12, le tableau jaune décrit les propriétés de l'entrée de FreeWRL. Le tableau ne contient pas de colonnes avant et après traitement car l'entrée n'est pas modifiable (cf. comparaison avec la fig. 5.5). Dans le cas présent, la seule propriété décrite est le nombre d'objets. Deux préconditions lui sont associées, selon la valeur de la RAM (Random Access Memory, ou mémoire vive) de l'environnement de travail de l'utilisateur.

Le lien "adaptation au contexte d'utilisation" mène à l'écran montré figure 5.13.

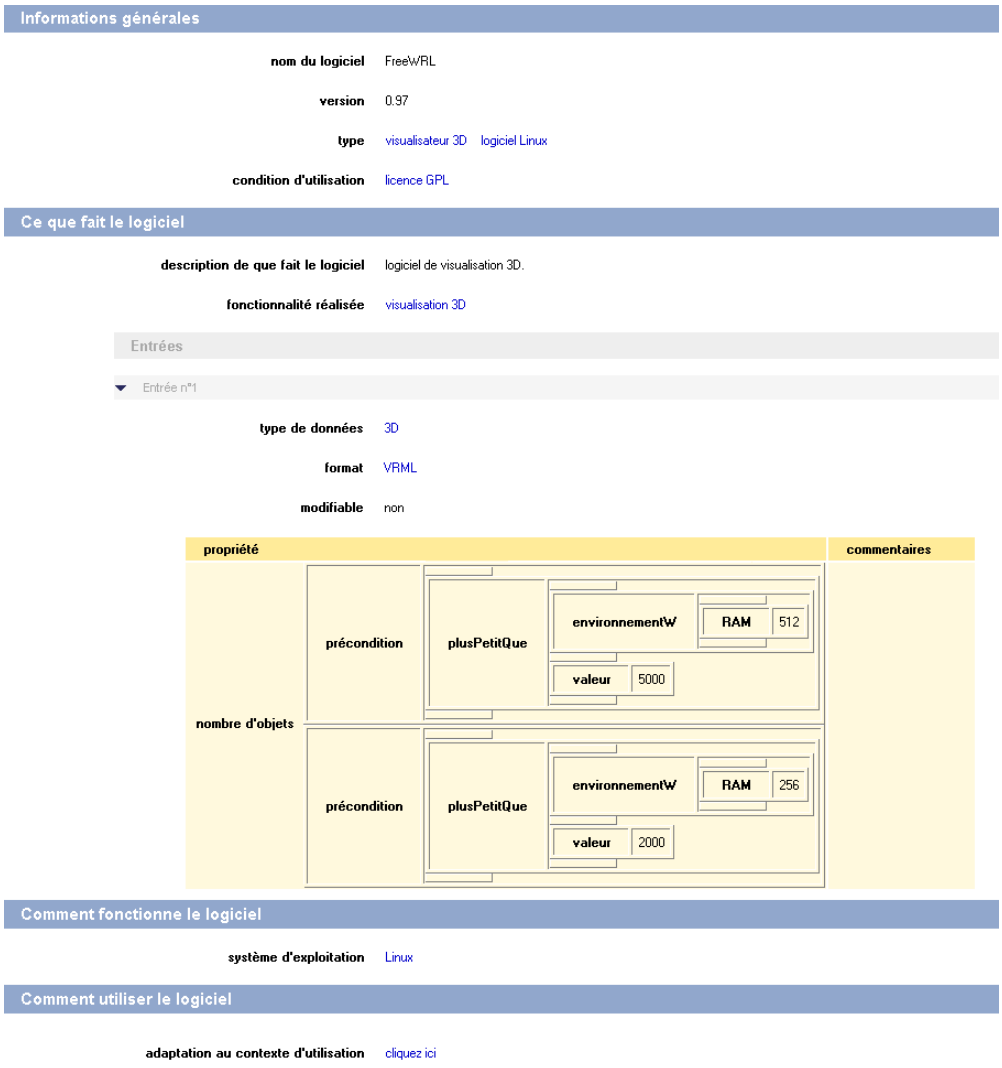


FIG. 5.12 – Description du logiciel FreeWRL

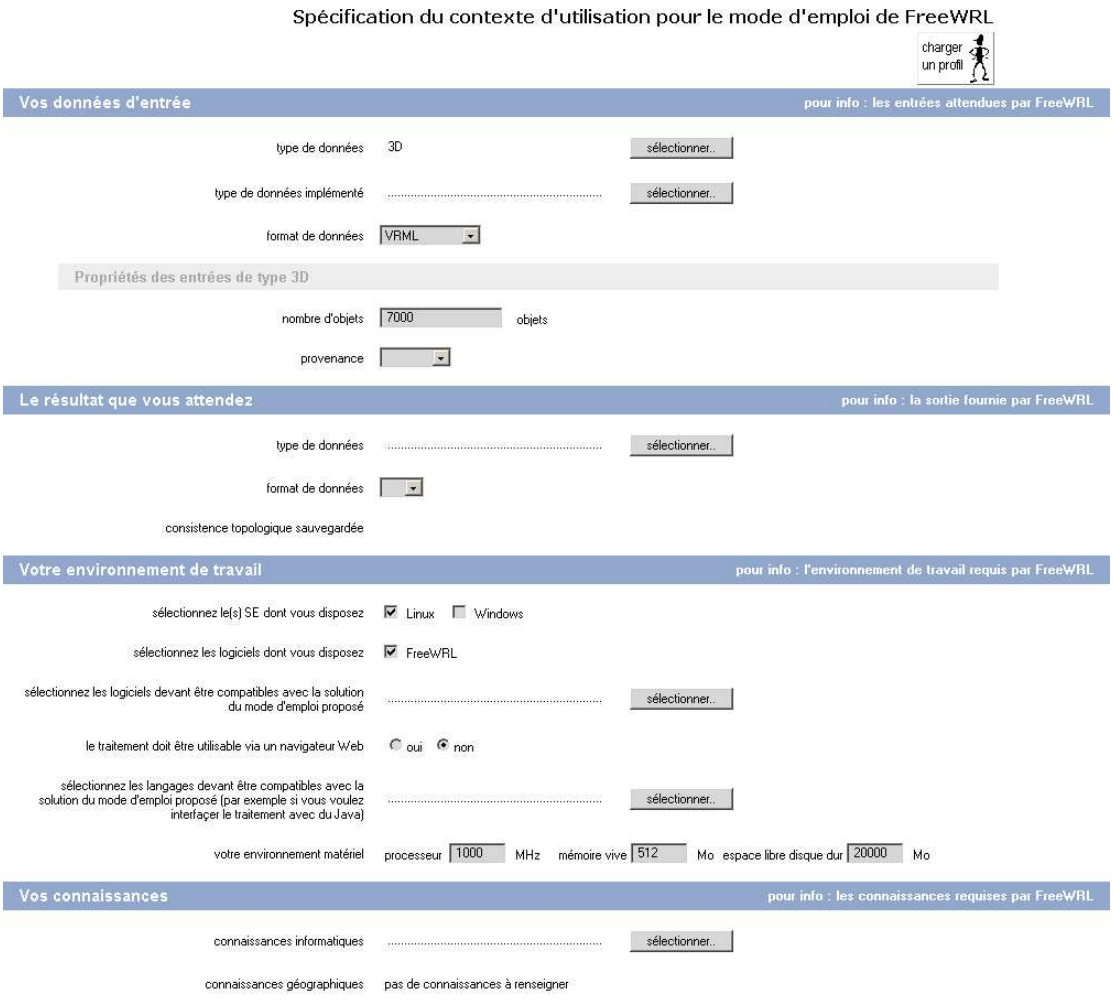


FIG. 5.13 – Formulaire de description du contexte de l'utilisateur

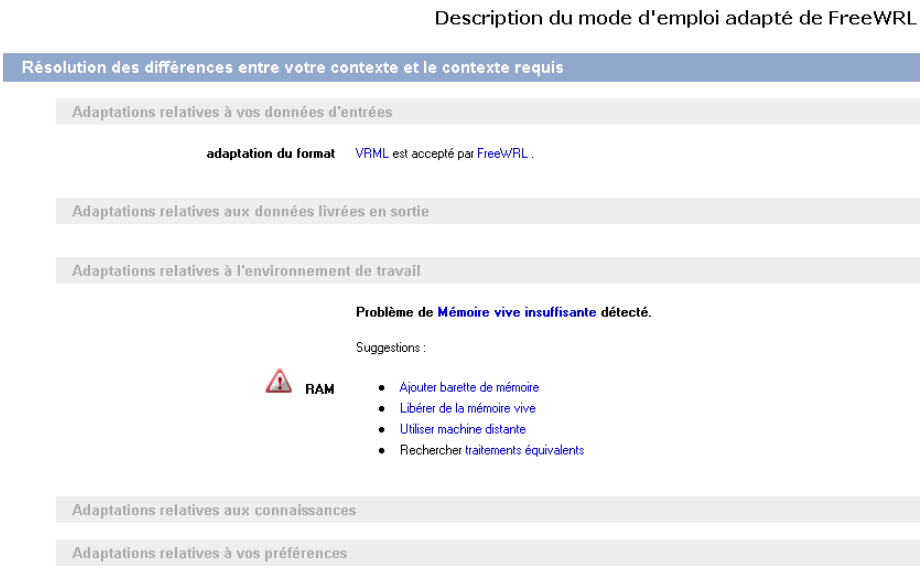


FIG. 5.14 – Mode d'emploi de FreeWRL adapté au contexte de ER 3

Du point de vue de l'application

Principe général

Hormis les pages d'accueil statiques, chaque écran de l'application est généré dynamiquement par un servlet. L'utilisateur clique sur les hyperliens, soumet des formulaires ; les requêtes HTTP sont envoyées au serveur (avec les méthodes GET ou POST) ; les servlets appelés effectuent diverses opérations puis envoient au client les pages HTML générées. Le tableau 5.2 décrit comment sont générées les trois pages de l'exemple ER 3.

Action utilisateur	**Page appelée**	**Opérations effectuées côté serveurs**
Sélection d'un traitement	DetailElement	1 – On applique **traitement.xsl** au DOM XML correspondant au traitement dont l'id est donné en paramètre. Le HTML produit est envoyé au client.
Demande d'adaptation du mode d'emploi au contexte	SpecifContexteUtilisation	1 – On génère le DOM askCTX.xml en appliquant la feuille XSL **genXML_askCTX.xsl** à modele.xml. **genXML_askCTX.xsl** contient des règles pour adapter le formulaire de saisie en fonction du traitement. Ces règles sont internes au système. On obtient un DOM XML conforme à askCTX.xsd. 2 – On applique **saisie_contexte.xsl** à askCTX.xml ; on obtient le HTML du formulaire de saisie. On l'envoie au client.
Saisie et soumission du formulaire de saisie du contexte	ModeEmploiAdapte	1 – On écrit contexte.xml conforme à contexte.xsd à partir des valeurs des champs de saisie. La partie requise par le traitement existe déjà ; on la récupère. 2 – On infère tout ce qu'on peut sur l'utilisateur. 3 – On calcule le delta(ctx_user_inféré, ctx_requis) en utilisant les inférences OWL pour établir les correspondances non évidentes. Grâce aux résulats de ces mises en correspondance on est en mesure d'évaluer les prémisses des règles d'adaptation, donc de savoir s'il faut déclencher ces dernières. On génère mode_emploi_adapte.xml conforme à mode_emploi_adapté.xsd avec la feuille **genere_mode_emploi_adapte.xsl** (appliquée à modele.xml et utilisant contexte.xml). 4 – On applique **mode_emploi_adapte.xsl** à mode_emploi_adapte.xml pour produire le HTML final.

TAB. 5.2 – Adaptation du mode d'emploi d'un traitement au pas à pas

Le principe général est de manipuler des documents XML[14] au moyen de feuilles XSL pour :

- extraire les parties de la base des métadonnées répondant aux requêtes XPath (générées par les servlets Java à partir des paramètres des requêtes HTTP),
- transformer et assembler les divers documents XML générés,
- générer les documents HTML finaux.

Le comportement des feuilles XSL dépend souvent de plusieurs paramètres comme l'id, le type de la ressource courante ou le profil de l'utilisateur. La plupart du temps nous incluons ces paramètres soit dans le fichier XML à transformer, soit dans des fichiers XML auxiliaires qui sont lus par les feuilles XSL. Mais parfois ce procédé est impossible – par exemple dans le cas de certaines expressions XPath – ou trop lourd ; nous générons alors dynamiquement le code

[14]Parfois les documents XML doivent être sauvegardés. Ils sont alors stockés sur le disque en tant que fichiers XML. Cela permet notamment de garder une trace précise des actions des utilisateurs. Mais la majorité du temps, les documents XML n'existent en fait que temporairement dans la mémoire vive sous forme d'objet DOM.

source des feuilles XSL dans les servlets Java[15]. Une solution que nous nous sommes interdit aurait pu être l'emploi de scripts `msxml` autorisés à figurer dans les feuilles XSL s'exécutant avec le parseur XML de Microsoft et capables notamment d'utiliser des objets COM, mais notre application n'aurait alors pu fonctionner que sur des systèmes d'exploitation Windows.

Détaillons l'implémentation de quelques-unes des étapes décrites par le tableau 5.2.

Génération du formulaire posant les questions "pertinentes" sur le contexte de l'utilisateur

Le code 5.8 montre un extrait de la feuille XSL qui génère le document XML à partir duquel l'écran de saisie du contexte de la figure 5.13 est généré.

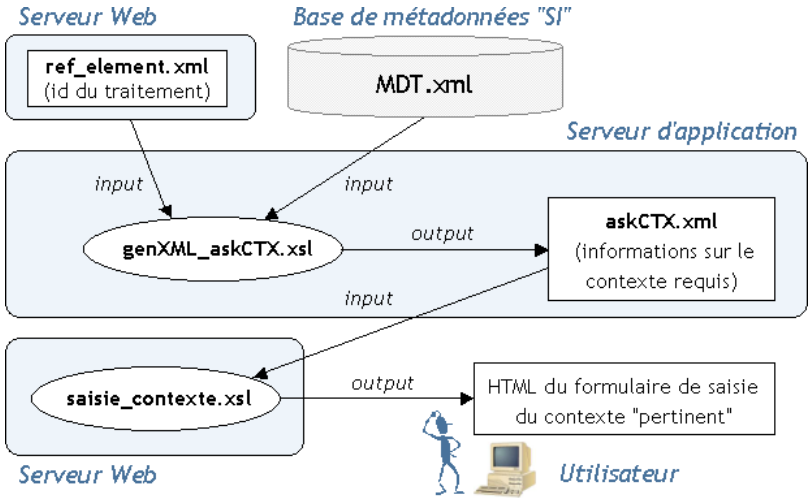


FIG. 5.15 – Opérations effectuées lors de l'appel du servlet SpecifContexteUtilisation

```
<xsl:for-each
   select="/mdt/mode_emplois/mode_emploi[@id=$vId_me]//traitement_requis">
  <xsl:choose>
    <xsl:when test="substring(., 1,3) ='cat' ">
      <!-- on va récupérer tous les logiciels de la catégorie -->
      <xsl:variable name="vCat" select="."/>
      <xsl:for-each select="/mdt/logiciels/logiciel[type=$SvCat]">
        <traitement_requis><xsl:value-of select="@id"/></traitement_requis>
      </xsl:for-each>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:for-each>
```

Extrait de code 5.8: XSL – Construction du formulaire de saisie "pertinent" (genXML_askCTX.xsl)

[15] Un exemple où l'on recourt à ce procédé est donné code A.12, page 239.

Calcul du mode d'emploi adapté

L'utilisateur a reçu le formulaire de saisie de son contexte. Il le remplit et le soumet. Le servlet ModeEmploiAdapte est appelé. À partir des champs reçus, d'une part, et de la description de FreeWRL présent dans la base de métadonnées, d'autre part, le document XML exposé code 5.9 est généré. Ce code correspond au diagramme de classe fig. 3.3, p. 137.

```
<contexte>
  <traitement>logFreeWRL</traitement>
  <contexte_traitement>
    <entrées>
      <entrée>
        <type_donnée>td3D</type_donnée>
        <format>forVRML</format>
        <modifiable>non</modifiable>
        <pte ref="pteNombreObjets">
          <précondition>
            <plusPetitQue>
              <environnementW>
                <RAM>512</RAM>
              </environnementW>
              <valeur>5000</valeur>
            </plusPetitQue>
          </précondition>
        </pte>
      </entrée>
    </entrées>
  </contexte_traitement>
  <contexte_utilisateur>
    <entrée_fournie>
      <format>forVRML</format>
      <type_donnée>td3D</type_donnée>
      <pte ref="pteNombreObjets">7000</pte>
    </entrée_fournie>
    <sortie_attendue/>
    <environnementW>
      <OS>null</OS>
      <logiciel>null</logiciel>
      <RAM>128</RAM>
      <DD/>
      <fréquenceProcesseur/>
    </environnementW>
    <connaissances_requises/>
    <préférence/>
  </contexte_utilisateur>
</contexte>
```

Extrait de code 5.9: XML – Contexte de l'exemple ER 3

L'utilisateur peut omettre des informations lors de la description de son contexte. Soit qu'il les ignore (p.ex. origine de ses données), soit qu'il néglige de les indiquer (p.ex. système d'exploitation, version des logiciels dont il dispose). Les règles stockées dans la base de connaissances doivent servir à déduire des informations du contexte. En l'occurrence, si l'utilisateur ne mentionne pas que ses données sont de type "3D", l'application le déduit grâce au format "VRML" indiqué. Nous avons implémenté quelques règles de ce type en XSL à des fins de tests. Cependant, la philosophie de notre application est de les implémenter en SWRL. En

l'occurrence, cette dernière voie ne pose pas de difficulté mais a pour inconvénient de demander un certain temps de développement (en raison du couplage en XSL et le moteur SWRL). Pour cette étape de l'adaptation des modes d'emploi nous n'avons pas eu le temps d'achever les développements comme il aurait convenu.

À partir de la description des contextes utilisateur et requis pas FreeWRL, le diagnostic du problème de RAM est posé grâce à la règle dont nous montrons ci-dessous l'expression dans trois formalismes : abrégé, au format de notre SI (code 5.10) et au format de notre SBC, *i.e.* en SWRL (code 5.11). Le code SWRL 5.11 s'applique à la description OWL schématisée fig. 5.16.

Le formalisme abrégé, en employant la notation DOM, est la suivante :

Si *contexte_traitement.entrée.propriété['pteNombreObjets'].precondition.valeur < contexte_utilisateur.entrée.propriété['pteNombreObjets']*
et *contexte_traitement.entrée.propriété['pteNombreObjets'].precondition.RAM = contexte_utilisateur.RAM*
Alors *detection(diagnostic, RAM_insuffisante)*

```
 <règle id="rgNbObjetsRAM">
   <nom>RAM insuffisante - nb. d'objets</nom>
   <description>détection de RAM insuffisante dûe au trop grand nombre
d'objets</description>
   <si_alors>
  <si>
    <et>
   <prédicat ref="Egal">
     <dom_mdt>contexte.contexte_utilisateur.entrée_fournie.pte.
     précondition.plusPetitQue.environnementW.RAM</dom_mdt>
     <dom_mdt>contexte.contexte_utilisateur.entrée_fournie.pte.
     précondition.environnementW.RAM</dom_mdt>
   </prédicat>
   <prédicat ref="plusPetitQue">
     <dom_mdt>contexte.contexte_traitement.entrée_attendue.pte.
     précondition.plusPetitQue.valeur</dom_mdt>
     <dom_mdt>contexte.contexte_utilisateur.environnementW.RAM</dom_mdt>
   </prédicat>
    </et>
  </si>
  <alors>
    <problème ref="pbRAMinsuffisante"/>
  </alors>
   </si_alors>
   <porte_sur>problème</porte_sur>
   <formalisme>horn</formalisme>
 </règle>
```

Extrait de code 5.10: XML – Règle pour la détection du problème de RAM insuffisante

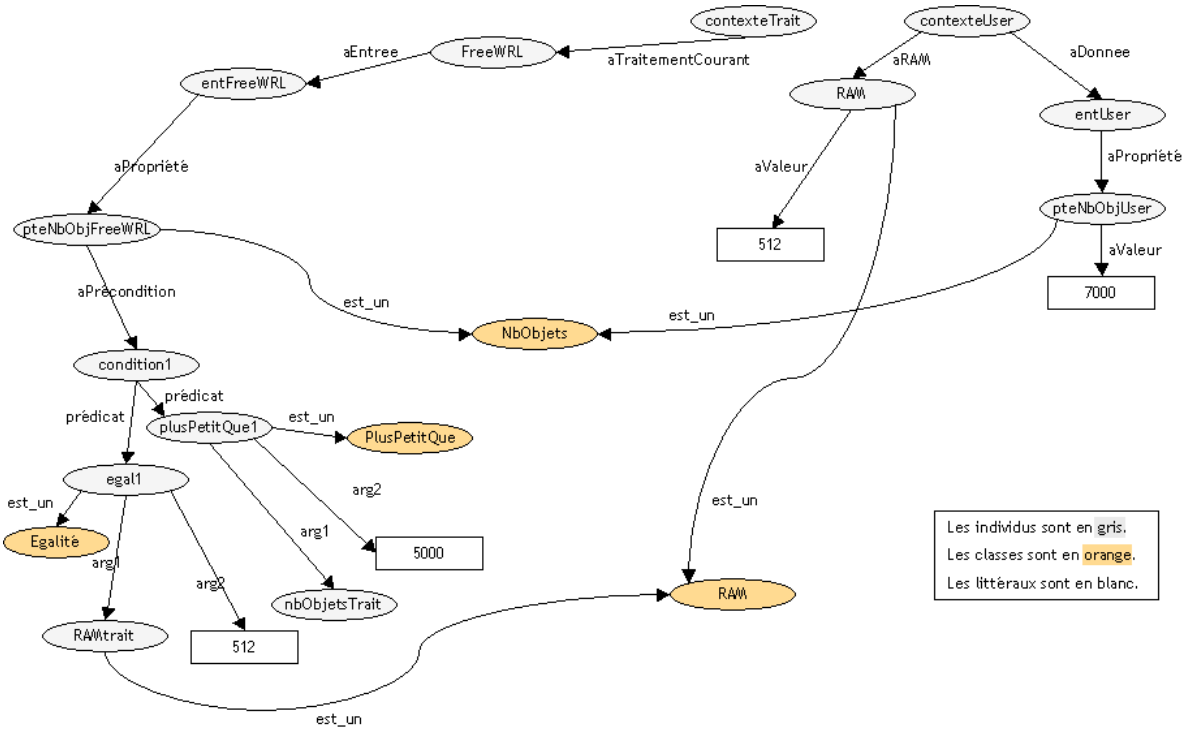


FIG. 5.16 – Classes et individus OWL du contexte de l'exemple ER 3

Nous n'avons pas pu exécuter la règle code 5.11 – traduite en syntaxe SWRL officielle par le plug-in de Protégé 3.1 – car le moteur que nous utilisons n'implémente pas encore les prédicats `swrlb:equal` et `swrlb:greaterThan`. Ces prédicats sont des prédicats *"built-in"* (d'où le 'b' du préfixe "swrlb"), c'est-à-dire des extensions au noyau du langage ([HPSB+04], §8).

De toutes façons, la règle 5.11 n'est en l'état pas capable de s'appliquer à tous les cas possibles. En effet, en supposant que les RAM ne peuvent prendre que certaines valeurs (des puissances de 2 la plupart du temps), il faudrait exprimer une condition par valeur. Or notre règle 5.11 ne permet, en l'état, de ne considérer qu'une précondition. Nous n'avons donc pas suffisamment poussé les développements pour réaliser l'exemple ER 3 de la façon espérée.

Nous voulions néanmoins réussir à atteindre au moins partiellement le but. Pour cela, nous avons implémenté la règle de détection du problème de RAM de façon *ad hoc* en XSL, la condition étant exprimée en XPath (code 5.12). La feuille XSL en question s'applique au code XML 5.9. C'est une entorse à la philosophie de notre application car le mécanisme de réponse ne repose plus sur un langage de représentation des connaissances mais sur un langage de programmation où les connaissances sont figées dans le code. C'est donc une solution "SI" et non une solution "SBC".

Précisément en raison des problèmes qu'il pose, l'exemple ER 3 est intéressant à plusieurs titres. Il permet de constater la lourdeur des règles SWRL dès lors que des structures de données un peu complexes doivent être adressées. Comparativement, le couple XSL/XPath est d'une expressivité plus faible mais d'une bien plus grande concision. En l'occurrence, une faible expressivité suffit puisque ER 3 ne requière pas de raisonnement de classification comme dans ER 1.

L'exemple ER 3 permet également de mesurer la difficulté qu'il y aurait à automatiser totalement la traduction des règles au format XML "SI" vers le format SWRL "SBC". Le premier doit être sauvegardé pour faciliter la lecture et la saisie par les utilisateurs, le deuxième

```
     aTraitementCourant(contexteTrait, ?trait)
∧    aEntree( ?trait, ?entreeTrait)
∧    aPropriete( ?entreeTrait, ?nbObjetsEntTrait)              propriété nb. objets
∧    estUn( ?nbObjetsEntTrait, NombreObjets)

∧    aPrecondition( ?nbObjetsEntTrait, ?condition)
∧    aPrédicat( ?condition, ?egal1)                            type de prédicat
∧    est_un( ?egal1, Egalité)

∧    aArgument1( ?egal1, ?RAMtrait)
∧    est_un( ?RAMtrait, RAM)                                   valeur de la RAM requise
∧    aValeur( ?RAMtrait, ?valeurRAMtrait)                      par le traitement

∧    aRAM(contexteUser, ?RAMuser)                              valeur de la RAM
∧    aValeur( ?RAMuser, ?valeurRAMuser)                        de l'utilisateur

∧    swrlb:equal( ?valeurRAMuser, ?valeurRAMtrait)             comparaison des RAM

∧    aPrédicat( ?condition, ?plusPetitQue1)
∧    est_un( ?plusPetitQue1, PlusPetitQue)                     type de prédicat
∧    aArgument1( ?plusPetitQue1, ?nbObjTrait)

∧    est_un( ?nbObjTrait, NombreObjets)                        valeur du nombre
∧    aValeur( ?nbObjTrait, ?valeurNbObjTrait)                  requis par le traitement

∧    aDonne(contexteUser, ?entUser)
∧    aPropriété( ?entUser, ?pteNbObjUser)                      valeur du nombre
∧    est_un( ?pteNbObjUser, NombreObjets)                      d'objet de l'utilisateur
∧    aValeur( ?pteNbObjUser, ?valeurNbObjUser)

∧    swrlb:greaterThan( ?valeurNbObjTrait, ?valeurNbObjUser)   comparaison des nb. obj.

⇒    detection(diagnostic, RAM_insuffisante)                   conclusion
```

Extrait de code 5.11: SWRL – Règle pour la détection du problème de RAM insuffisante

La syntaxe SWRL est simplifiée : c'est celle qu'accepte le plug-in SWRL de Protégé 3.1.
contexteTrait, contexteUser, diagnostic et RAM_insuffisante sont des individus définis dans la base de connaissances avant l'exécution de la règle.
RAM, NbObjets, Egalite et PlusPetitQue sont des classes.
Les noms des variables, préfixés par ' ?', sont quelconques ; nous ne les faisons correspondre avec les noms d'individus existants fig. 5.16 que pour une raison mnémotechnique.
Le prédicat binaire est_un(i, C) indique si l'individu i est instance de la classe C. Ce prédicat se note en fait en SWRL sous la forme d'un prédicat unaire C(i) ([HPSB+04], §2.1). Nous avons créé de toute pièce le prédicat est_un pour contourner l'incapacité du moteur SWRL que nous utilisons à calculer les prédicats C(i) : à chaque classe on associe un individu spécial qui la représente ; ces individus servent ensuite de 2ème argument de l'`objectProperty` OWL / prédicat SWRL est_un. Cette solution peut sembler artificielle mais elle fonctionne bien.

est nécessaire à l'opérationnalisation des règles ; nous sommes là confrontés au dilemme classique entre aspects déclaratif et procédural.

```
<xsl:if test="/contexte/contexte_utilisateur/environnementW/RAM =
  /contexte/contexte_traitement/entrées/entrée/pte/précondition/plusPetitQue/environnementW/RAM

and

  /contexte/contexte_utilisateur/entrée_fournie/pte[@ref = 'pteNombreObjets'] &gt ;
  /contexte/contexte_traitement/entrées/entrée/pte[@ref = 'pteNombreObjets']/
                                              précondition/plusPetitQue/valeur">
   <problème ref="pbRAMinsuffisante"/>
</xsl:if>
```

Extrait de code 5.12: XSL – Règle pour la détection du problème de RAM insuffisante[16]

5.3.5 Enrichissement de la base de métadonnées

Les résultats des inférences effectuées de façon locale, contextuelle et temporaire pour répondre à des requêtes précises, comme dans l'exemple ER 1, ne sont pas destinés à être stockés. D'autres inférences sont en revanche utiles pour compléter des descriptions de ressources. Par exemple, un auteur de métadonnées saisit la description du logiciel Geoconcept, décrit ses principales fonctionnalités, mais omet de spécifier qu'il s'agit d'un SIG ou, inversement, indique qu'il s'agit d'un SIG mais omet de décrire les fonctionnalités réalisées. Les classifications d'individus et de concepts OWL permettent de compléter les informations manquantes. Les connaissances OWL et RDF générées peuvent être exportées au format XML de la base de métadonnées de la partie SI.

Les inférences OWL et SWRL peuvent donc être utilisées pour enrichir la base de métadonnées. Nous n'avons pas totalement automatisé le processus mais nous avons fourni l'environnement logiciel pour qu'un administrateur puisse profiter des capacités d'inférences du SBC.

Certains moteurs d'inférence OWL permettent également de vérifier la consistance des ontologies. On peut aussi, par ailleurs, définir nos propres règles SWRL pour détecter les incohérences entre descriptions de ressources liées (traitement codé/algorithme, ressource traitement / fonctionnalité, etc.).

Une piste pour enrichir les descriptions incomplètes de logiciels ou de librairies est d'agréger les propriétés des traitements qui les composent. Supposons par exemple que le package java `appli.bdmul.carrefour`[17] est indexé dans la base de métadonnées sans que soit spécifié le type de fonctionnalités qu'il réalise. En revanche, il est indiqué que les classes qui composent ce package réalisent respectivement les fonctionnalités : "détection de carrefours simples", et "détection de carrefours en étoile", "détection de carrefours en T", "détection de carrefours en Y". Notre système pourrait alors déduire que le package `appli.bdmul.carrefour` réalise la plus proche fonctionnalité parente commune à ces fonctionnalités qui est "détection de carrefours" (fig. 5.17).

[16] Le symbole de prédicat "plus grand que" se note `&gt ;` dans l'expression XPath car > est un symbole réservé des documents XML, donc également aussi des feuilles XSL.

[17] Développé au laboratoire COGIT par É. Grosso.

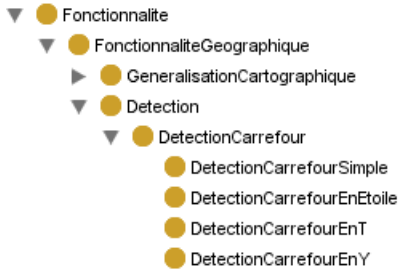


FIG. 5.17 – Classification des fonctionnalités de détection de carrefours

L'enrichissement de la base de métadonnées passe également par l'application de règles SWRL qui transmettent les valeurs de propriétés entre individus. Voici deux exemples typiques.

Certaines bases de données géographiques comportent des attributs représentant explicitement la topologie, d'autres pas. Les jeux de données extraits héritent de cette caractéristique :

Si estDonnée(d) et estBD (bd) et d.origine = bd
Alors d.pté("topologie") = bd.pté("topologie")

On peut ajouter des conditions supplémentaires afin de réaliser l'exemple ER 7.

Autre exemple de transmission de propriétés, une FonctionLogiciel hérite de beaucoup de propriétés du Logiciel auquel elle appartient (ici le système d'exploitation).

Si estFonctionLogiciel(fl) et estLogiciel(l) et appartient(fl,l)
Alors fl.OS = l.OS

De telles règles peuvent décharger les auteurs de descriptions d'opérations de saisie fastidieuses. Le choix de l'alternative entre enrichir la base de métadonnées, au prix d'une taille plus importante, ou de dériver dynamiquement l'information au moyen de règles appartient à l'administrateur de la base de métadonnées.

5.4 Discussion

Avant de clore la présentation de la partie de notre application dédiée à la consultation des métadonnées et à la simulation de certains raisonnements de l'expert, tentons d'en faire une première évaluation.

Évaluation de la partie SI de l'application construite

Les cas d'utilisation relatifs à la consultation des métadonnées (cf. p. 39) ont été réalisés. Nous ne pouvons pas pour autant affirmer que les besoins d'information des utilisateurs sont effectivement satisfaits. Cela, seule une validation par la pratique pourra l'indiquer. En attendant, des tests ont d'ores et déjà apporté plusieurs enseignements. Les tests que nous avons nous-mêmes menés ne peuvent bien sûr se voir accorder qu'une valeur limitée, compte tenu du biais méthodologique lié à notre forcément bonne connaissance du système. Les tests effectués par des utilisateurs témoins sont plus intéressants. Nous avons recueilli les impressions d'une dizaine d'entre eux, tous chercheurs ou stagiaires au laboratoire COGIT. Une étude sérieuse aurait nécessité la mise en place d'un protocole de test reproductible, avec des entretiens libres et non semi-dirigés comme cela a parfois été le cas ; le temps nous a manqué pour mettre cela

en place de façon rigoureuse.

Néanmoins, on peut affirmer que, globalement, les fonctionnalités de base que sont la recherche par mots-clés, la navigation dans les index et la visualisation des descriptions de ressources particulières sont utilisées de façon satisfaisante par les utilisateurs. Ces fonctionnalités, du reste, n'ont rien d'inédit. Elles sont déjà familières aux utilisateurs habitués à la navigation sur le Web et aux logiciels d'aide traditionnels, dont notre application ne fait que reprendre les principaux principes, notamment ergonomiques.

La réalisation de la partie SI de notre application, en fait, ne présentait pas de difficultés. C'est, en arrière plan, le modèle de métadonnées qui détermine l'adéquation aux besoins des utilisateurs et, bien sûr, la richesse du contenu de la base de métadonnées. Si nous en jugeons d'après les exemples de requêtes répertoriées lors des enquêtes et tests menés, notre modèle comporte bien les éléments de descriptions adéquats. Cela est naturel : il a précisément été construit à partir des résultats de l'analyse des besoins. Quant à la richesse de la base de métadonnées dont dépend *in fine* la satisfaction de la réponse aux utilisateurs, elle ne dépend plus de nous mais de la bonne volonté des auteurs de descriptions – en supposant toutefois que notre application soit dotée d'une interface accueillante pour l'acquisition, ce qui est le cas, comme nous allons le voir au chapitre 6.

En terme de temps d'accès, la partie de SI l'application présente des performances acceptables. Lors des tests effectués aucune page ne mettait plus de cinq secondes à s'afficher. Les facteurs responsables de cette durée ne sont ni les caractéristiques du poste client[18], ni le débit du réseau Intranet de l'IGN largement suffisant (réseau local Ethernet, 100 Mbits/sec. théorique), mais les caractéristiques de la machine hébergeant les serveurs Web et d'application. L'opération coûteuse en temps processeur et en mémoire vive est le chargement de la base de métadonnées XML dans des objets DOM et l'application des feuilles XSL. Lors de nos tests, la base de métadonnées XML occupait 600 ko sur le disque. La machine sur laquelle tournaient les servlets de l'application était équipée d'un Pentium IV cadencé à 2.4 GHz, d'une mémoire vive de 512 Mo, et avait Windows 2000 pour système d'exploitation. Le moteur de servlets, Tomcat 5, occupait jusqu'à 80 Mo en mémoire vive. Nous n'avons jamais sollicité l'application avec plusieurs accès simultanés ; cela pourrait dégrader fortement les temps de réponses. Sachant, de plus, qu'à terme la base de métadonnées devrait normalement voir sa taille multipliée, le recours à des fichiers index intermédiaires ou, mieux, l'adoption d'une base de données XML native sera nécessaire.

Évaluation de la partie SBC de l'application construite

La partie SBC de l'application, quant à elle, n'a pas été testée en conditions réelles. Pour son développement, les sept exemples de raisonnement ER nous ont servi de fil d'Ariane ; c'est à travers ces sept exemples que, dans un premier temps, nous entendons valider notre application.

La recherche "intelligente" de traitement, comme dans les exemples ER 1 et ER 2 suite, a été réalisée avec succès. L'adaptation des modes d'emploi au contexte d'utilisation, comme dans l'exemple ER 3, n'a été réalisée que partiellement par rapport à nos espoirs initiaux. Il est de toutes façons clair que la simulation du raisonnement de l'expert n'est pas un but que l'on peut atteindre, mais seulement un but vers lequel on peut tendre.

[18]Pour les pages les plus lourdes de l'application – 400 ko, sans les images et les données éventuelles –, on note toutefois un écart de l'ordre de la demi-seconde selon les navigateurs Web, Internet Explorer 6 possédant un moteur de rendu HTML un peu plus rapide que ses concurrents.

Notre système d'adaptation des modes d'emploi au contexte d'utilisation ne contient actuellement pas suffisamment de règles pour être réellement utilisé. Même si l'opérationnalisation de nouvelles règles demandera l'intervention d'un administrateur du système, leur acquisition en tant que simples métadonnées présente en soi un intérêt. Les utilisateurs devraient trouver intérêt à les consulter.

Nous nous sommes heurtés à la difficulté de traduire en SWRL les règles au format SI utilisant le formalisme DOM pour adresser les éléments de la base de métadonnées. Dès lors, la question de renoncer à notre format SI se pose. Nous pensons qu'il faut le garder, et poursuivre le développement de programmes effectuant la traduction automatiquement ou, à défaut, prévoir de l'effectuer manuellement. En effet, l'interface d'acquisition des règles, élément crucial pour inciter les experts à exprimer leurs connaissances, gagne beaucoup à employer le formalisme DOM plutôt qu'une notation à base de prédicats (cf. §6.6 p.217).

Ceci étant entendu, l'administrateur soucieux d'adopter les choix de développement les plus fiables, simples et rapides à mettre en œuvre pourra demander si en fin de compte il ne vaut pas mieux implémenter les règles sous forme de *templates* XSL *ad hoc*, quitte à recourir à des fichiers XML annexes générés par le SBC lorsque des raisonnements de classification sont nécessaires[19]. À cette question légitime, évoquée p. 188, nous répondons que la philosophie de la construction d'un SBC commande d'utiliser SWRL, qu'à long terme ce choix nous semble se justifier, notamment en terme de facilité de maintenance, les règles SWRL étant des ressources externes au moteur d'inférence, mais qu'à court terme effectivement le choix de XSL *ad hoc* apparaît le plus facile à mettre en œuvre.

Concernant l'automatisation de l'importation et de l'exportation avec le SI, la partie RDF de notre SBC, aux multiples syntaxes possibles, pose également des problèmes. Nous ne les avons pas résolus. L'intervention humaine est nécessaire.

Un autre problème, qui n'est pas de notre fait, est dû à la jeunesse des langages OWL et SWRL, qui implique un manque de maturité des moteurs d'inférence actuels. Cette faiblesse a limité nos expérimentations, notamment celles de l'exemple ER 3. L'apparition prochaine de nouveaux moteurs SWRL devrait résoudre ce problème.

Nous n'avons pas abordé les questions de stratégie d'application des règles SWRL, susceptibles d'influer sur les performances du SBC. Le moteur SWRL que nous utilisons fonctionne en chaînage avant. Contrairement au SI, les temps de réponse du SBC sont très importants. Ils dépassent souvent la dizaine de secondes alors que nous n'avons mené nos expérimentations que sur des bases de connaissances réduites (seulement une cinquantaine de concepts et une centaine d'individus).

[19] Pour les règles d'adaptations aux modes d'emploi, l'évaluation des prémisses requiert souvent un raisonnement de classification (p. ex. le type de la donnée de l'utilisateur est-il une sorte de celui attendu par le traitement ?), la conjonction des prémisses et le déclenchement de la conclusion requièrent une règle de la forme si - alors.

5.5 Conclusion

Nous avons présenté l'application développée sous ses deux aspects SI et SBC.

Les cas d'utilisation définis au chapitre 1 ont été réalisés. L'utilisateur recherche des traitements et des ressources liées. Il navigue dans la base de métadonnées. Des descriptions lui sont présentées. Leur forme met en œuvre, notamment, des principes d'héritage permettant un accès progressif aux connaissances.

L'objectif premier de notre travail est donc atteint : les principaux besoins d'information sur les traitements sont satisfaits – l'acquisition des métadonnées faisant l'objet du chapitre 6.

Nous avons montré plusieurs exemples de mise en œuvre de raisonnements. Dans le cadre d'une recherche de traitement, dans le cadre d'une adaptation de mode d'emploi au contexte d'utilisation, et dans le cadre d'un enrichissement du contenu de la base de métadonnées.

Chapitre 6

Acquisition des métadonnées

La saisie manuelle des descriptions de traitements ou de toutes autres ressources du modèle s'effectue *via* des formulaires HTML. La section 6.1 expose la façon dont se déroule le processus pour l'auteur de descriptions. Au-delà des simples questions relatives à l'interface de l'application, nous essayons de cerner les facteurs qui entravent l'acquisition des métadonnées. La section 6.2 montre comment certains champs de description peuvent être remplis automatiquement. La section 6.3 envisage les problèmes potentiellement posés par l'évolution future de la base de métadonnées, en particulier de la partie "ontologique".

6.1 Saisie manuelle des métadonnées

Nous exposons la façon dont se déroule une saisie de description de traitement à travers l'exemple du programme Accordéon. Nous montrons ensuite comme se passe la saisie d'une règle.

6.1.1 Saisie de description de traitement

Une personne veut décrire un traitement. Elle accède à la page d'accueil de l'application et sélectionne le lien "Ajouter ou modifier des descriptions". Avant de se voir proposer un formulaire de description, la personne doit préciser le type de ressource et le domaine de celle-ci. Dans l'exemple qui nous sert ici, la ressource est un programme du domaine "traitements données vecteur". Ces informations conditionnent le type des champs à remplir (p. ex. un fichier WSDL pour un service Web ou l'absence de partie "implémentation" pour un algorithme) etc.) et la valeur de ces champs (les types de données, les types d'effets généraux du traitement, etc.). Les deux informations préliminaires renseignées, le formulaire de description principal s'affiche (fig. 6.1).

Les valeurs de certains champs de descriptions sont des références à des ressources. Leur saisie s'effectue au moyen de listes déroulantes ou *via* des fenêtres *pop-up*. Les listes déroulantes sont adaptées aux ressources non hiérarchiques ou peu nombreuses comme les personnes. Les ressources comme les fonctionnalités, au contraire, s'effectue *via* des fenêtres *pop-up*. Apportons rapidement quelques précisions techniques au sujet de ces dernières.

La somme de toutes les références de ressources potentiellement sélectionnables dans le cadre d'une description de traitement est trop importante pour les faire figurer toutes dans une unique page Web. Découper le formulaire principal de description des traitements en plusieurs pages n'est une solution souhaitable ni pour l'utilisateur, à qui l'on doit demander le minimum d'effort cognitif, ni pour la simplicité de l'application côté serveur, les formulaires partiels devant alors être sauvegardés avant l'enregistrement définitif. Le recours aux *pop-up* apparaît comme une alternative élégante.

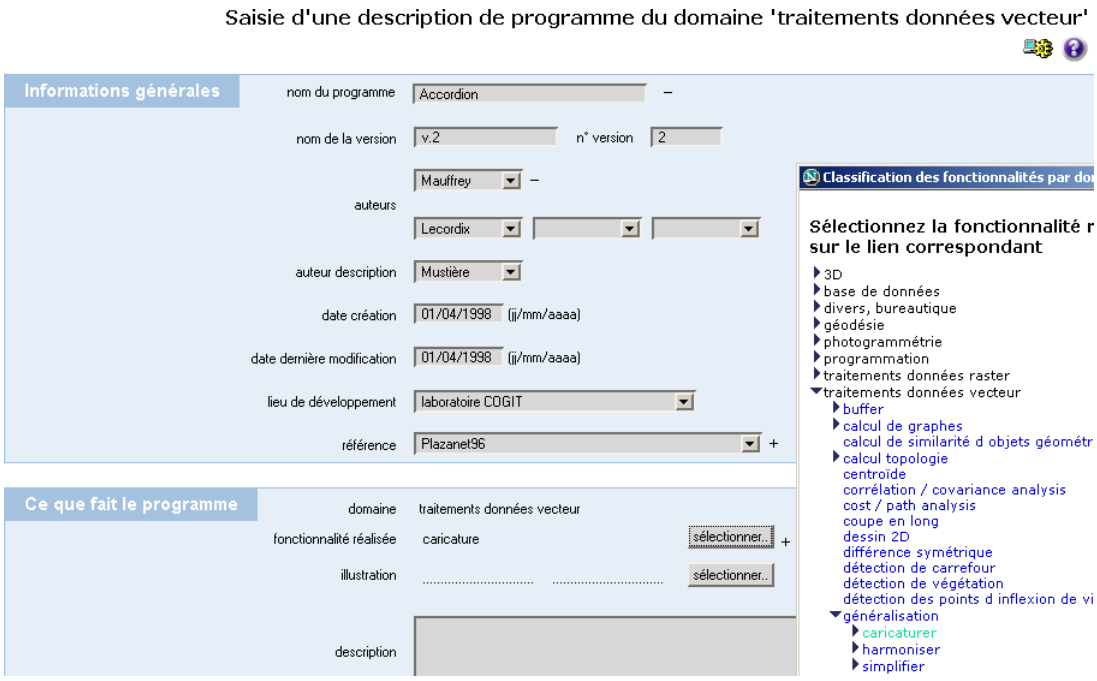


FIG. 6.1 – Sélection d'une fonctionnalité

Enregistrement d'un échantillon de données

La personne qui décrit le programme Accordéon veut inclure une illustration de jeu de données avant et après traitement. Elle indique que ses données sont de type "ligne vecteur" puis actionne le bouton "sélectionner ..." associé au champ illustration. La fenêtre *pop-up* montrée figure 6.2 apparaît.

La liste des échantillons référencés dans la base de métadonnées s'affiche. Si aucun ne convient, l'enregistrement de nouveaux échantillons est proposé *via* une nouvelle fenêtre *pop-up* (fig. 6.3). La personne sélectionne les fichiers de données présents sur le disque de sa machine ; ils sont envoyés au serveur au moment de la soumission du formulaire (cf. en annexe code 6.4 p. 240). De retour au premier *pop-up*, l'utilisateur visualise le nouvel échantillon ajouté à la liste. Il valide son choix et revient au formulaire principal de description du programme Accordéon.

Saisie des propriétés des données avant et après traitement

La figure 6.4 montre les différents champs pour la description de l'entrée. La valeur du type de données abstrait renseigné, ici "ensemble de ligne vecteur", conditionne les champs de description du *pop-up* qui apparaît lorsque le bouton de saisie des propriétés est actionné (fig. 6.5).

À chaque propriété peuvent être associées les informations prévues par le modèle. Pour saisir les préconditions l'utilisateur actionne le bouton "sélectionner" ; un nouveau *pop-up* apparaît, semblable à l'écran montré figure 5.13 p. 200, qui permet de décrire le contexte particulier nécessité.

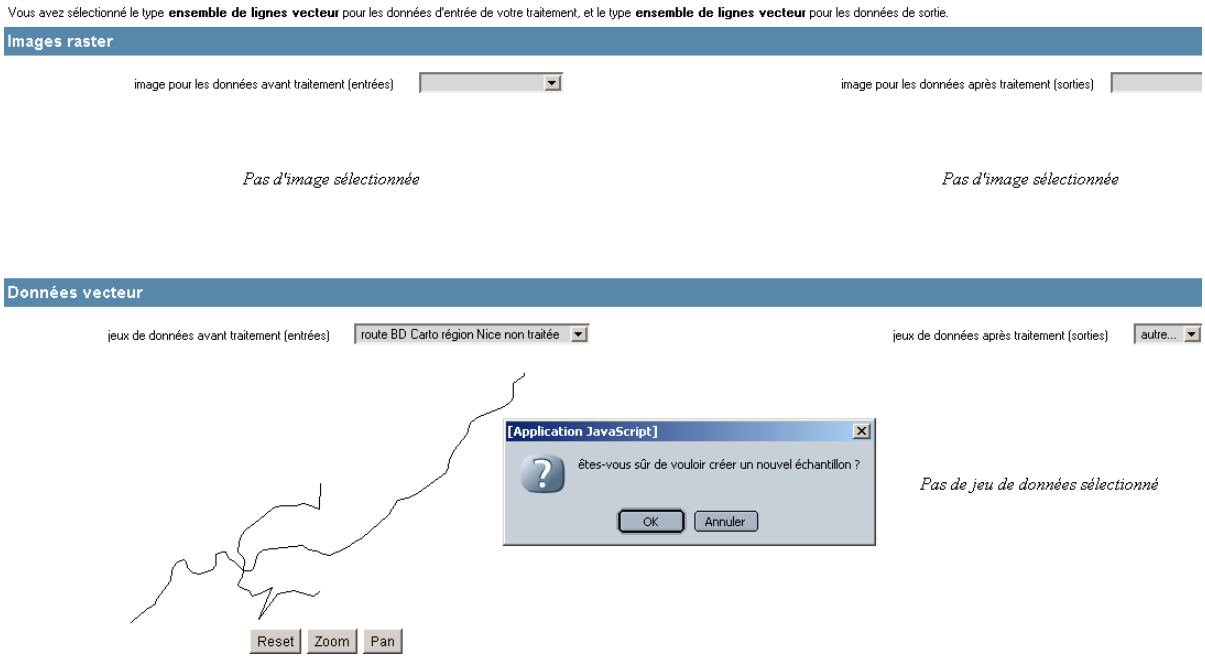


FIG. 6.2 – Sélection d'un échantillon de la base de métadonnées

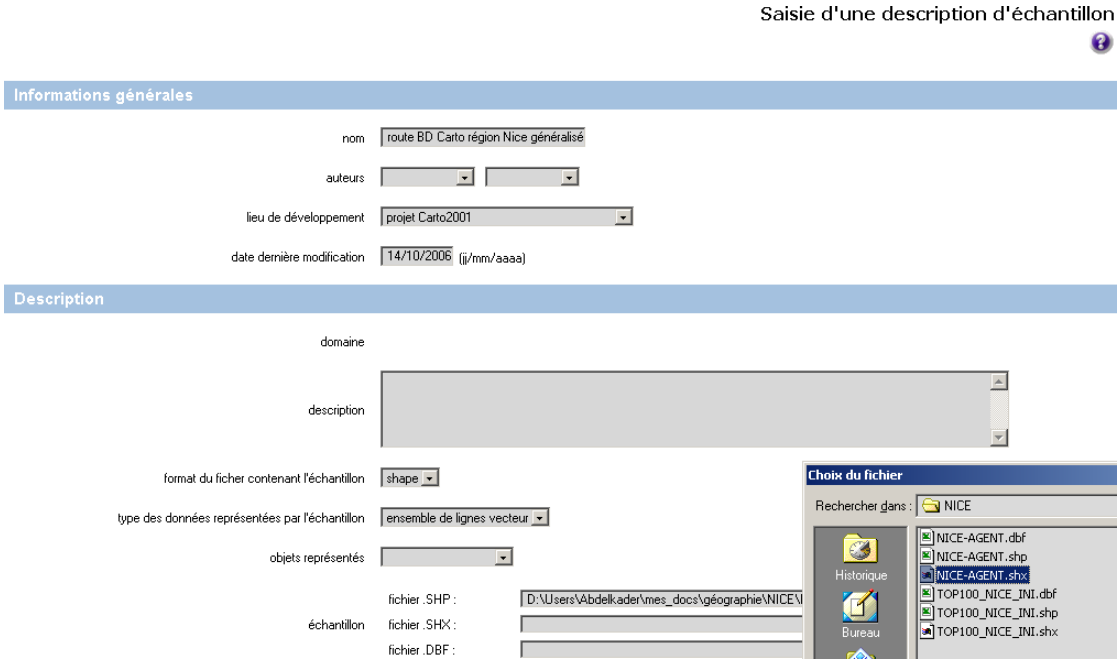


FIG. 6.3 – Enregistrement d'un échantillon de données au format *shape*

FIG. 6.4 – Saisie de la description d'une entrée

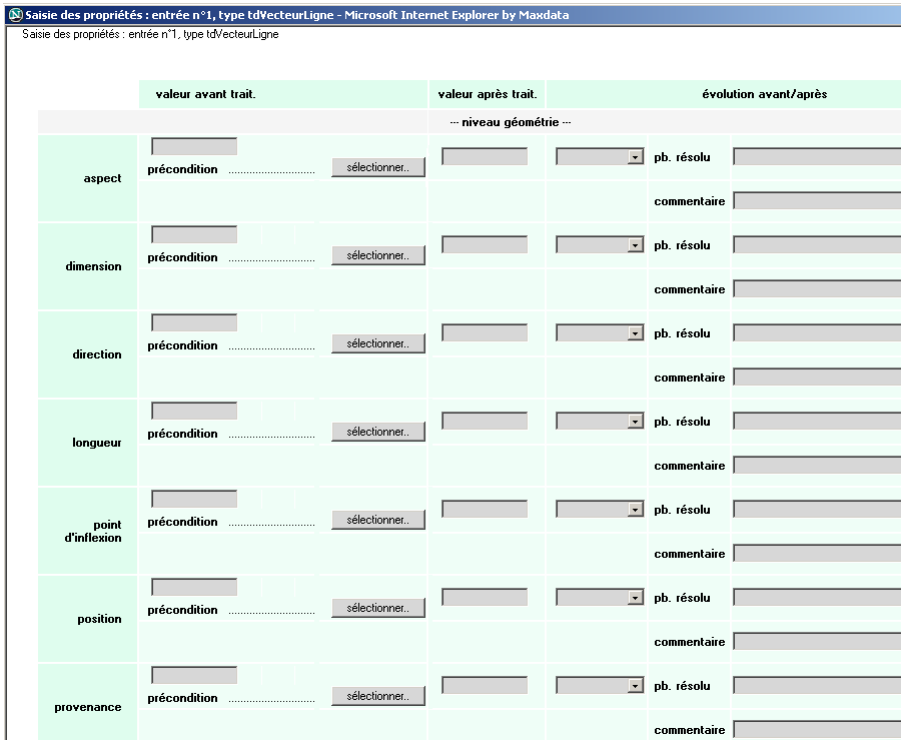


FIG. 6.5 – Saisie de la description d'une propriété

6.1.2 Saisie d'une règle

En cours de développement à l'heure de la rédaction du mémoire, l'interface d'acquisition des règles est composée de listes déroulantes permettant d'adresser n'importe quel élément du modèle (fig. 6.6). L'idée est de reproduire la fonctionnalité des IDE qui facilite la saisie d'expressions de type DOM (fig. 6.7). À chaque nœud de l'arbre représentant les structures de données manipulées, les choix possibles sont proposés à l'utilisateur sous forme de liste déroulante.

La figure 6.7 est une copie d'écran de l'IDE Eclipse ; les classes du modèle de métadonnées ont ici été traduites en Java pour l'exemple. Une interface conviviale de saisie des règles pourra s'inspirer du mécanisme ; à cette fin la réalisation d'une *applet* pourrait être envisagée.

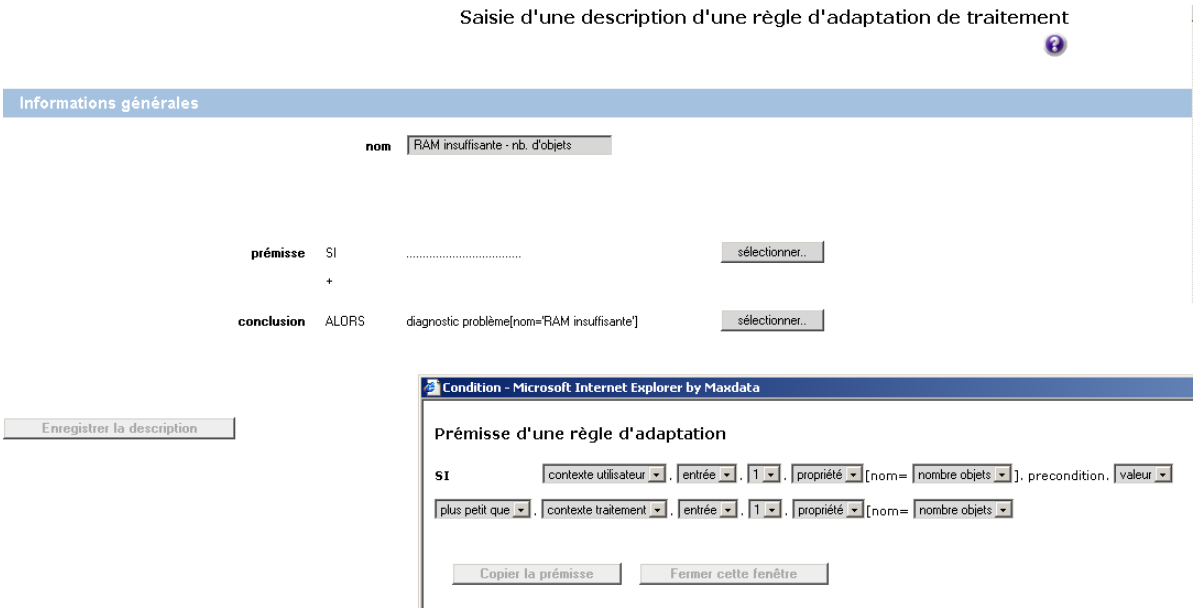


FIG. 6.6 – Saisie d'une règle

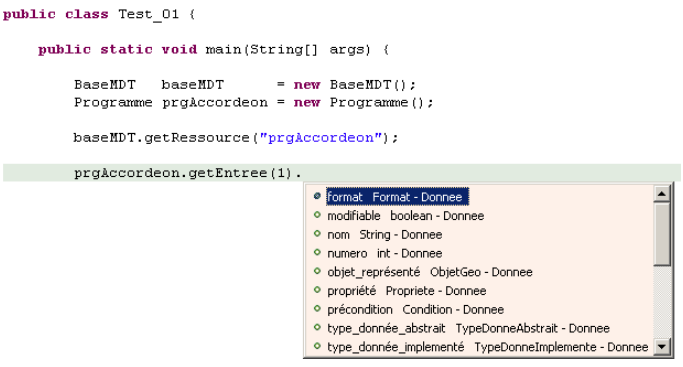


FIG. 6.7 – Saisie d'une expression en notation de type DOM avec Eclipse

6.1.3 Obstacles à l'acquisition

Lors du développement de SI, la phase d'acquisition des informations n'est pas forcément problématique. La phase d'acquisition des connaissances est, en revanche, connue pour être fréquemment un point faible du développement de SBC.

Dans le cadre de notre application, nous avons constaté des obstacles à l'acquisition de deux ordres. L'un dépend du niveau d'implication des auteurs de descriptions, l'autre tient aux limites de notre application.

Malgré nos efforts pour rendre simple l'interface de saisie des descriptions, plusieurs des utilisateurs témoins ont manifesté une certaine appréhension face au nombre de champs à remplir, perçu comme important. En réponse, nous avançons l'argument que le temps requis pour une saisie de description de traitement, négligeable rapporté au temps de développement de ce dernier, doit être considéré comme un investissement.

Il importe de convaincre les auteurs de traitements de l'intérêt qu'ils auront plus tard à utiliser la base de métadonnées. Un travail préalable d'explication est donc nécessaire. S'il ne s'avérait pas suffisant – il arrive qu'une certaine inertie dans les habitudes freine l'adoption d'un nouvel outil – l'incitation pourrait passer par l'édiction de chartes au sein des services de l'IGN concernés. Cela se pratique déjà dans certains projets de développement où obligation est faite, par exemple, de commenter et d'indenter le code, de respecter des conventions de nommage des variables, etc.

Nous avons toutefois tenté de prévenir les réticences éventuelles en modifiant l'interface de telle sorte que les champs de descriptions d'importance secondaire soient initialement masqués. Le formulaire de saisie apparaît donc de prime abord relativement court. La visibilité des champs secondaires est commandée, sur plusieurs niveaux de profondeurs, par les icônes $+/-$ et $\triangledown/\triangleright$.

Au cours des expérimentations menées nous avons également rencontré des obstacles à l'acquisition où étaient en cause l'expressivité du modèle et la richesse de la base de métadonnées. Certains de ces obstacles nécessitent simplement une légère familiarisation des utilisateurs avec les principes de l'application, d'autres ne peuvent être surmontés.

L'auteur de description veut référencer une ressource non indexée. Dans ce cas, il doit créer lui-même la description de la ressource à laquelle il veut faire référence. Nous avons ainsi vu précédemment, par exemple, comment un nouvel échantillon était ajouté à la base de métadonnées. Le risque existe cependant qu'une personne ne sache pas, par exemple, dans quelle catégorie classer une nouvelle fonctionnalité ou veuille créer une ressource qui existe déjà sous un autre nom. La constitution de taxinomies de ressources, voire d'ontologies si l'on introduit diverses relations, propriétés et contraintes entre concepts, demande parfois une réelle réflexion ; il est délicat d'attendre de tous les auteurs de descriptions une contribution correcte. En attendant que les principales ressources utilisées pour l'indexation soient toutes décrites, le contrôle de l'évolution de la base par un expert est nécessaire.

L'auteur de description veut décrire une donnée au moyen d'un critère non indexé. Ce cas de figure se ramène au précédent car, dans notre modèle, les propriétés et les unités sont des ressources. Par exemple, comme les types de données du domaine de traitement d'image ne sont actuellement que sommairement décrits, supposons que la propriété "résolution" ou l'unité TSL (teinte, saturation, luminescence) de la propriété "couleur" manquent pour décrire les images ; l'auteur de description peut créer ces nouvelles ressources. Si la compatibilité avec des normes de métadonnées existante est souhaitée, l'administrateur de la base de métadonnées devra créer les ressources correspondantes avant de permettre les descriptions

de traitements du domaine concerné. En cas de normes concurrentes, l'une d'elle devra être privilégiée ; les feuilles XSL établissant les correspondances devraient ensuite pouvoir être écrites.

L'auteur de description veut exprimer, entre des ressources ou des éléments de descriptions, des relations non prévues dans le modèle. Si les relations à exprimer sont des relations génériques standard de logique de description, l'auteur ne pourra les exprimer *via* l'interface de notre application. En effet, notre schéma XML ne permet de définir que des taxinomies de ressources, c'est-à-dire des ontologies légères (les FamilleTraitement faisant exception). Ce n'est pas un vrai langage de définition d'ontologies comme OWL. De plus, notre application n'a pas vocation à rivaliser avec de véritables éditeurs d'ontologies comme Protégé.

Si les relations à exprimer entre les ressources ou entre des éléments de descriptions ne sont pas prévues par le modèle, il existe deux solutions : les règles ou les champs de description en langue naturelle. Par exemple, un auteur veut spécifier que le bug d'un programme est lié à une propriété particulière de l'entrée. Il se trouve que le champ de description proposé "bug" est en langue naturelle car le besoin d'exploiter un référencement formel ne s'est pas fait sentir. Si, cependant, l'auteur de description tient à exprimer la relation autrement qu'en langue naturelle il a la liberté de créer une règle dont la prémisse comporte une expression DOM référençant n'importe quel élément de description – en l'occurrence la propriété de l'entrée incriminée.

L'auteur de description de traitements ou l'expert ne songent pas aux connaissances qu'ils pourraient exprimer. L'affichage de la liste des propriétés des traitements et du contexte utilisateur porte en elle la suggestion de règles. En cela, les écrans de saisie de l'application fournissent un cadre propice à l'explicitation des connaissances tacites. Les familles de traitements, créées spécialement pour servir de réceptacles aux connaissances générales sur les traitements, sont également un élément favorable à l'extraction du savoir de l'expert. Les règles de notre modèle étant des ressources comme les autres, elles peuvent être consultées et ainsi stimuler aussi l'imagination des experts.

6.2 Acquisition (semi-)automatique des métadonnées

6.2.1 Développement d'un *doclet*

Au laboratoire COGIT une grande partie des développements se font en Java. De nombreux outils sont associés à ce langage. Nous nous sommes servis de l'un d'eux, le programme standard javadoc, pour automatiser le remplissage d'une partie des champs du formulaire de saisie de description des classes et méthodes Java.

Pour réaliser cela nous avons développé un *doclet*. Proposée par Sun[1], la classe `com.sun.javadoc.Doclet` permet de modifier le comportement standard du programme javadoc. Ainsi, au lieu de produire la traditionnelle documentation au format HTML, le *doclet* que nous avons développé produit des métadonnées XML conformes à notre modèle (cf. codes A.9 et A.10 p. 237).

Les informations du code à documenter, auxquelles on accède *via* l'API Reflection[2], étant normalement déjà indexées dans la base de métadonnées, l' "indexation sémantique" automatique est en partie possible. Par exemple, si notre *doclet* indique qu'une variable est de type connu *GM_Object*, alors le programme chargé de faire le lien avec les métadonnées de la base déduit que le type de donnée abstrait est *vecteur*.

Ainsi, la personne désirant décrire une classe Java accède au formulaire de description, actionne le bouton "analyse de code" symbolisé par un écrou, sélectionne dans le *pop-up* qui s'affiche la classe à analyser (préalablement enregistrée sur le CVS du laboratoire COGIT, ce

[1] `http://java.sun.com/j2se/1.4.2/docs/tooldocs/javadoc/index.html`
[2] `http://java.sun.com/docs/books/tutorial/reflect/index.html`

qui constitue la démarche habituelle et donc n'est pas une contrainte liée à notre application), puis valide son choix.

Le serveur d'application, à partir de la description générée par le doclet, produit dynamiquement le code JavaScript que, côté client, le *pop-up* va exécuter, remplissant ainsi automatiquement les champs du formulaire (cf. code A.12, p. 239). L'auteur de description doit ensuite compléter les champs non remplis, ainsi que les descriptions des méthodes générés. Les descriptions de certaines méthodes, telles les accesseurs, présentent peu d'intérêt et pourront être supprimées.

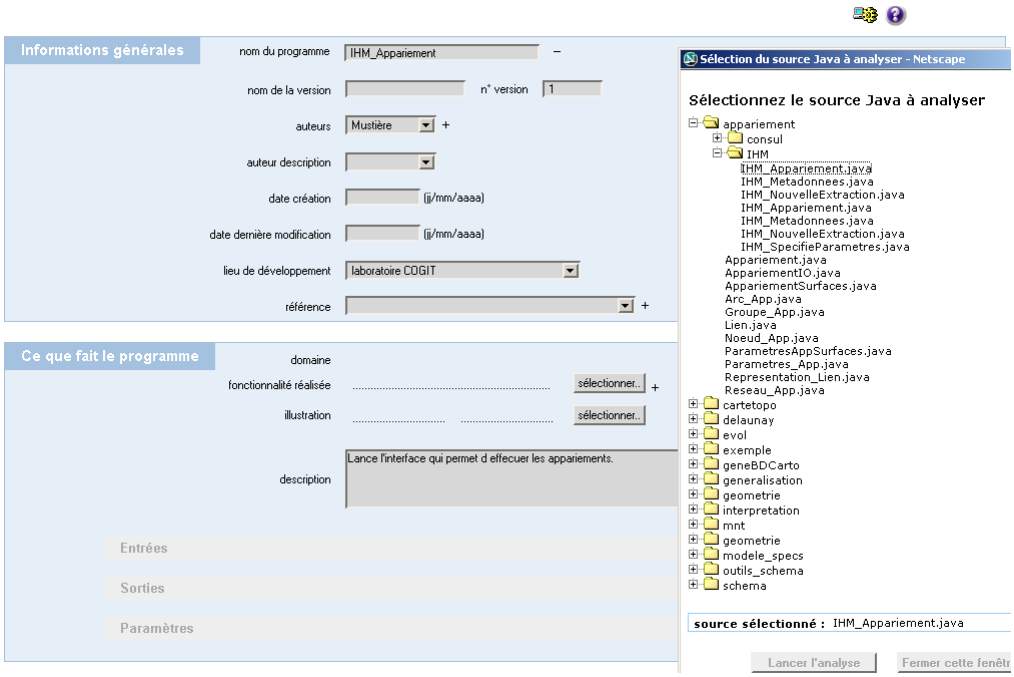


FIG. 6.8 – Analyse automatique de code Java

Afin de détecter automatiquement les fonctionnalités réalisées, nous avons développé un programme qui compte le nombre de mots communs aux commentaires de code extraits et aux descriptions de la base de métadonnées. Les résultats obtenus par ce procédé sont médiocres : outre qu'il faudrait utiliser en complément un dictionnaire de synonymes et un lemmatiseur (p. ex. convertir le verbe "détecte" en lexème "détection"), l'orthographe des commentaires de code est très fréquemment martyrisée et le français et l'anglais y sont allègrement mélangés. Nous n'avons pas poussé nos expérimentations. Les poursuivre demandera de recourir à des outils de TALN (Traitement Automatique de la Langue Naturelle) et recouper les résultats candidats à l'issue de l'analyse à un test de cohérence ; les capacités de classification du SBC seront alors utiles (p. ex. si la fonctionnalité "appariement" est définie dans l'ontologie comme ayant nécessairement des entrées de type "table de BD").

Pour extraire les commentaires et signatures de fonctions des langages autres que Java, comme Lull, nous avons développé des parseurs basés sur des expressions régulières (cf. code A.11).

6.2.2 Récupération et intégration de documentations existantes

Les documentations disponibles sous forme électronique peuvent être indexées et enregistrées dans notre base de métadonnées. Leur référencement dans les descriptions de traitements ne peut se faire que de façon manuelle, sauf à imaginer la mise en œuvre d'outils de fouille de données (*data mining*) et de TALN.

Une sorte particulière de documents est plus favorable à une intégration automatique dans les descriptions de traitements : c'est celle des fichiers d'aide dont on connaît le format. Nous avons ainsi, par exemple, pu indexer plus de 200 FonctionLogiciel du SIG Geoconcept 5 en décompilant le fichier d'aide livré par le fabriquant avec l'outil Html Help Workshop 4.74[3]. Le fichier en question – HelpG5.chm – est en effet au format CHM (Compiled Help Module), le format standard d'aide sous Windows. Sa décompilation produit des pages HTML, chacune correspondant à une FonctionLogiciel. Un programme simple que nous avons développé génère alors les descriptions de ces FonctionLogiciel comportant leur nom, leur logiciel d'appartenance et la référence vers les descriptions des ModeEmploi également créés. Le code A.5 p. 235 montre un extrait des descriptions de modes d'emploi (les champs pré-requis ayant été ajoutés manuellement). La figure 5.7 p. 187 montre un de ces modes d'emploi tel que le verra finalement l'utilisateur à travers notre application Web.

Ces techniques sont semi-automatiques. Elles seront grandement améliorées si, à l'image des documentations DocBook[4], les aides des logiciels adoptent une syntaxe XML. C'est le cas par exemple de l'IDE Eclipse : chaque plug-in est décrit par un fichier plugin.xml déclarant son contenu et la façon d'y accéder *via* l'interface[5]. La liaison avec notre modèle de métadonnées pourra alors être totalement automatisée, il suffira de spécifier les correspondances entre les éléments de descriptions dans des feuilles XSL.

6.2.3 Autres pistes non encore mises en œuvre

La seconde piste vise à automatiser la description des actionIHM de nos modes d'emploi. L'utilisateur expert ne saisirait plus les descriptions des modes d'emploi des FonctionLogiciel manuellement, mais effectuerait des démonstrations qui seraient enregistrées. L'utilisateur novice pourrait ensuite non seulement consulter les descriptions générées, mais aussi demander que lui soient reproduites les démonstrations. Cela existe déjà en environnement Windows : les applications de la suite Office permettent d'enregistrer des macros, et les pages HTML des aides CHM comportent des liens "Démonstration" (qui déclenchent la fonction VBScript `Showme()`, laquelle invoque les fonctions du logiciel concerné).

D'une façon plus générale, les programmes comportent souvent des options permettant d'activer un mode spécial destiné au *débuggage* et traçant les actions des utilisateurs. Les fichiers *log* générées constituent des sources de métadonnées à exploiter.

6.2.4 Bilan

Le tableau 6.1 dresse le bilan des éléments de descriptions dont l'obtention peut être automatisée, ou du moins dont on peut raisonnablement penser qu'elle puisse l'être à court terme.

[3] `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/hwMicrosoftHTMLHelpDownloads.asp`

[4] `http://www.docbook.org/`

[5] `http://wiki.eclipse.org/index.php/FAQ_What_is_the_plug-in_manifest_file_(plugin.xml)%3F`

Facette	Éléments de description	Moyen d'obtention
Identification	nom auteur date dernière modification date création	commentaires de code, propriétés des fichiers, ou informations de logiciels type CVS
	auteur description lieu de développement	IP de la machine cliente utilisée
Fonction	domaine fonctionnalité réalisée	commentaires de code + recoupement autres informations
	entrées / sorties	Javadoc et autres outils basés sur des compilateurs, voire simples parsers capables d'extraire les signatures des fonctions
Fonctionnement	utilise	détection des directives "d'import"
	langage	indiqué par l'extension du nom de programme ou par le logiciel de gestion de versions (type CVS)
	OS	déduits des autres éléments de description
Mode d'emploi	modes d'emploi communs à des familles de traitements	déduits après classification du traitement (d'après la fonctionnalités, l'OS, etc.)
	modes d'emploi spécifiques	enregistreurs de macros et/ou transformation de documentations existantes
Évaluation	bug	fichiers *log*

TAB. 6.1 – Informations dont l'obtention peut ou pourrait être automatisée

6.3 Évolution future de la base de métadonnées

La base de métadonnées construite comporte des descriptions de traitements, mais aussi des descriptions de ressources telles les fonctionnalités et les types de données. Ces ressources, organisées de façon hiérarchique et liées par diverses relations, forment des ontologies.

Parallèlement à la croissance de l'ensemble des descriptions de traitements, les ontologies sont aussi appelées à s'enrichir. Ces deux évolutions de la base de métadonnées n'ont pas les mêmes implications ; celle des ontologies comporte des risques qu'il faut prévenir.

6.3.1 Évolution de l'ensemble des descriptions de traitements

En son état actuel, la base de métadonnées indexe plusieurs centaines de traitements et ensembles de traitements. Seuls quelques dizaines d'entre eux possèdent des descriptions complètes. Les compléter et en saisir de nouvelles, *via* notre application, ne devrait pas poser de difficultés, les obstacles potentiels mentionnés précédemment se manifestant de façon marginale.

Le nombre de traitements indexés dans la base et la qualité des descriptions seront des clés essentielles de la popularité de l'application. Il est important de permettre l'enregistrement de descriptions incomplètes pour ne pas rebuter les auteurs de métadonnées, mais il est important de ne pas décevoir les utilisateurs par la pauvreté des informations offertes. Un compromis et à trouver. Actuellement la permissivité est maximale : seule les noms des ressources est nécessaire à leur indexation.

Une autre clé du succès de l'application de consultation des métadonnées sera la confiance dans les informations qui s'y trouvent. Les débats autour de l'encyclopédie libre Wikipedia[6] basée sur le principe de la coopération, et la circonspection quant à l'attitude parfois prônée à

[6]Wiki wiki signifie rapide en hawaiien ; pédia, du grec ancien *paideia*, éducation (`http://fr.wikipedia.org/wiki/Wikipedia`).

son égard, montrent l'importance de la question du contrôle et de l'autorité dans les systèmes d'information auquel chacun peut contribuer. Actuellement, n'importe quelle personne peut modifier directement le contenu de la base de métadonnées ; l'administrateur ayant toujours la possibilité de restaurer après coup les états antérieurs sauvegardés en cas de signalement d'erreurs

Néanmoins, les descriptions de traitements font davantage référence à des descriptions de ressources que l'inverse. Cela signifie qu'une description de traitement erronée a des répercussions limitées sur le reste de la base. Une description de fonctionnalité mal catégorisée, par exemple, est beaucoup gênante. Or justement, si les erreurs dans les descriptions de traitements ont a priori peu de raisons de se produire – d'autant que l'interface de saisie, en effectuant une première sélection dans les valeurs possibles, prévient une partie des risques –, les défauts de conceptions dans les ontologies sont beaucoup plus délicats à éviter.

6.3.2 Évolution des ontologies

Nous l'avons déjà souligné, notre application n'est pas un éditeur d'ontologie. Cela n'interdit cependant pas de conseiller certains principes sains de conception d'ontologies à l'intention des candidats à la création de nouvelles ressources-concepts.

Le premier critère de qualité d'une ontologie est sans doute sa validation par au moins un expert du domaine. Hormis pour certaines fonctionnalités géographiques, nous ne nous sommes pas assurés de cette validation. Selon F. Fürst, il faudrait faire suivre à nos ontologies un cycle de vie semblable à celui des logiciels : construction, diffusion, utilisation et évaluation. Notre but, en fait, était simplement d'amorcer le processus avant de laisser le soin aux experts de poursuivre la tâche.

Si notre application ne permet de définir des taxinomies de ressources de façon sommaire – bien loin de l'expressivité des constructeurs OWL , elle ne prescrit pas non plus de méthodes garantissant le respect d'autres critères de qualité particulier. Or l'approbation des experts ne suffit pas toujours. Par exemple, l'éditeur d'ontologie développé par R. Troncy et A. Isaac, baptisé DOE (Differential Ontology Editor) [TI02], est dédié à la mise en application de ces principes différentiels proposés par B. Bachimont [Bac00] :

- le principe de communauté d'un concept avec son père
- le principe de différence d'un concept avec son père
- le principe de différence d'un concept avec ses frères
- le principe de communauté d'un concept avec ses frères

La mise en application de tels principes et l'utilisation d'outils comme DOE les instrumentant compléteront utilement notre application (le format d'échange pourra s'effectuer en OWL)[7].

Les ontologies sont faites pour être partagées. On peut ainsi imaginer qu'une partie de nos métadonnées servent à d'autres usages que l'indexation des traitements informatiques.

Réciproquement, on peut imaginer importer des ontologies extérieures dans notre base de métadonnées. Des problèmes de fusion d'ontologies se poseront alors. Un problème lié est celui de la gestion des répercussions sur les descriptions de traitements lors de l'évolution des ontologies[8].

[7] Le lecteur trouvera un référencement des méthodes de conception d'ontologies et des outils les mettant en oeuvre dans [Für05] pp. 19 et 25-26, dans [Isa01] pp. 13-23 et dans [Isa05] pp. 123-135.

[8] Le versionnage des ontologies OWL est prévu ([W3C04c], §6) mais cela ne résout pas tous les problèmes causés par les révisions dans les indexations devenues obsolètes. La description des évolutions et les mises à jours seraient des pistes à creuser.

6.4 Conclusion

Nous avons présenté la façon dont les descriptions de ressources peuvent être saisies *via* l'interface de notre application. En particulier, nous avons montré comment nous avons su composer avec les contraintes inhérentes au contexte d'une application Web où l'utilisateur client ne dispose que d'un navigateur standard. Nous avons ébauché une solution pour rendre autant que faire se peu aisée la saisie des règles d'expert, objectif difficile comme nous l'avons constaté au chapitre 5 avec l'exemple ER 3. La suggestion des éléments du modèle lors de l'expression des prémisses des règles est sans doute un moyen propice à faire "parler" l'expert. Plus généralement, nous avons discuté des obstacles rencontrés par les utilisateurs lors de la saisie.

Dans ce chapitre nous avons également montré comment nous avions automatisé l'acquisition de certains éléments de description des traitements. En particulier nous avons profité de la possibilité offerte par l'environnement standard Java de personnaliser le comportement du programme `javadoc` au moyen d'un *doclet*.

Enfin, nous avons évoqué les problèmes potentiellement posés par l'évolution future de la base de métadonnées. Il apparaît nécessaire de mettre en place un contrôle minimum lors de l'acquisition, imposant notamment le respect de certains principes de conception d'ontologies afin d'en garantir la qualité, et par suite de permettre une exploitation optimale de la base de métadonnées.

Conclusion

Bilan de la recherche

Rechercher, connaître et utiliser les traitements informatiques du domaine géographique n'est pas toujours tâche facile. L'aide aux utilisateurs et développeurs de l'IGN est le besoin à l'origine de notre travail. De nombreuses connaissances, parfois tacites, manquent au novice confronté à différents types de traitements : les SIG possèdent leurs propres formats de données, interface et langage de programmation associés ; les librairies et programmes développés au sein de l'IGN requièrent des compétences spécifiques en programmation, en gestion des bases de données, en cartographie, en traitement d'images, etc. ; les traitements sous forme de services Web demandent, eux, des connaissances spécifiques sur les protocoles de communication.

Plusieurs types de documentations existent (manuels, documentations API, forums, etc.), mais leur dispersion, l'hétérogénéité de leur format et l'absence d'un système d'indexation commun (*i.e.* de l'utilisation d'un vocabulaire contrôlé commun) ne permettent pas de répondre aux besoins d'information identifiés que de façon imparfaite.

Face à ce constat, l'idée de départ de notre travail était de créer une base de métadonnées puis de construire un système en permettant la recherche, la consultation et l'enrichissement. L'analyse des besoins a cependant montré que pour répondre à certaines requêtes de l'utilisateur, une simple base de métadonnées, dont les informations explicitement présentes sont en nombre nécessairement limité, ne pouvait suffire. Il fallait donc mettre en place des mécanismes de dérivation de l'information s'appuyant sur une *représentation opérationnelle des connaissances d'expert*. En particulier, notre ambition était de fournir des modes d'emploi adaptés au contexte d'utilisation (caractéristiques des données, environnement logiciel, connaissances de l'utilisateur).

Nous avons soutenu la thèse qu'une réponse aux besoins évoqués peut être fournie par un système basé sur une double approche : *documentaire*, et orientée *représentation des connaissances*. L'idée est de construire, d'une part, un dans lequel la forme structurée des métadonnées, conformes à notre modèle, rende aisé le développement de l'application Web présentée à l'utilisateur, d'autre part, un *Système à Base de Connaissances* (SBC) doté des capacités d'inférences qui nous permettent de simuler une partie du raisonnement de l'expert.

Les principaux résultats obtenus peuvent se résumer en trois points.

Définition d'un modèle de métadonnées

L'état de l'art dressé au début de notre travail n'a permis de déceler aucun modèle de description des traitements qui réponde pleinement à nos attentes. Nous avons donc défini notre propre modèle, en nous inspirant toutefois, notamment, de l'ontologie OWL-S dédiée aux services Web. Organisé selon cinq facettes de description, notre modèle est générique ; il s'applique *a priori* aux traitements informatiques de n'importe quel domaine. Il prend cependant en compte des aspects spécifiquement adaptés au domaine géographique, tels que la description fine des

propriétés des données avant et après traitements et le recours aux illustrations cartographiques. De plus, si la partie *grounding* d'OWL-S décrit la façon d'accéder à des services Web, la partie mode d'emploi a vocation à décrire la façon d'accéder aux traitements informatiques en général.

Développement d'un SI

Nous avons implémenté notre modèle en XML Schema, créé une base de méta-données XML et développé une application Web qui permet la recherche, la consultation et la saisie des métadonnées. Plusieurs caractéristiques notables mises en œuvre peuvent être relevées. Elles correspondent aux objectifs identifiés lors de l'analyse des besoins et, parfois, tendent à dépasser certaines des limitations qui affectent les documentations classiques.

Notamment, nous avons tenté de permettre une description progressive des modes d'emploi, les plus spécifiques héritant des concepts et pré-requis des plus génériques.

Cherchant à permettre l'expression de connaissances générales, nous avons introduit dans notre modèle la notion de famille de traitement. L'expert humain recourt à des exemples pour clarifier ses explications ; l'association de prototypes aux familles de traitements pourra contribuer à rendre plus parlantes nos descriptions.

Les illustrations au moyen d'échantillons de données, dont l'intérêt a été montré dans le contexte de la généralisation cartographique par le travail au laboratoire COGIT de F. Hubert [Hub03], ont été intégrées à nos descriptions. Notre application en permet la visualisation et l'acquisition aux formats *image* courants mais aussi au format *vecteur shape*.

L'acquisition automatique d'une partie des descriptions est possible. Pour cela nous avons développé un *doclet* et des programmes basés sur des expressions régulières.

Développement d'un SBC

La recherche de traitement et l'adaptation des modes d'emploi au contexte d'utilisation nécessitent de simuler le raisonnement de l'expert. Les connaissances de ce dernier sont représentées de façon opérationnelle grâce à deux sous-ensembles de la logique du premier ordre : les logiques de description pour les ontologies et les clauses de Horn pour les règles avec variables. Les langages d'implémentation choisis sont ceux du Web sémantique : RDF, OWL et SWRL. Notre base de métadonnées documentaire, traduite dans ces langages, devient une base de connaissances. Pour effectuer sur celle-ci inférences et requêtes, notre système fait appel à la plateforme Sesame 1.2.1 et à un moteur SWRL développé à l'université libre de Berlin. En marge de l'application, nous avons également expérimenté Jena 2.2.

Apports et limites de notre travail

En proposant aux utilisateurs et développeurs de l'IGN l'application Web SI/SBC présentée dans ce mémoire, nous avons contribué à mieux répondre aux besoins d'informations mettant en jeu des connaissances parfois tacites. Le modèle que nous proposons favorise la mise à jour de ces connaissances.

Par rapport aux documentations existantes qui ne répondent que de façon imparfaite à nos besoins en terme de contenu, de précision ou de niveau de formalisation, notre apport réside dans l'expressivité du modèle de métadonnées défini et dans l'exploitation qui en est faite grâce à la double approche SI/SBC. Isolément, il existe plusieurs modèles de métadonnées et travaux qui répondent à une partie de nos objectifs ; notre travail a consisté à tenter de réunir les avantages de chacun au sein d'un même modèle adapté au contexte spécifique des besoins de l'IGN. Parmi les descriptions de traitements informatiques géographiques répertoriées lors de notre état de l'art, beaucoup étaient insuffisamment détaillées pour nos besoins, et celles qui l'était étaient insuffisamment formalisées. Nous avons en effet besoin de contrôler le vocabulaire

utilisé plutôt que de permettre l'usage de langue naturelle libre dans les descriptions. Des descriptions opérationnelles permettant la planification existent également, mais elles sont spécifiques à des catégories particulières de traitements (OWL-S pour les services Web par exemple). Enfin, des descriptions formelles d'interfaces utilisateurs de logiciels existent, mais sont destinées à un contexte de programmation plutôt que d'aide à l'utilisateur.

La proposition du modèle conceptuel de métadonnées des traitements constitue un des principaux apports de notre travail ; les exemples créés et les premières expérimentations menées montrent l'adéquation aux besoins d'information sur les traitements dans le contexte de l'IGN.

Un autre apport est d'avoir montré l'intérêt des principes de représentation des connaissances que sont les logiques de description (LD) et les règles de production en logique du premier ordre dans le but de la simulation d'une partie du raisonnement de l'expert pour rechercher les traitements et adapter leurs modes d'emploi au contexte de l'utilisateur. Les langages de LD permettent de définir des ontologies. Celles que nous avons proposées spécifiquement pour le domaine géographique (fonctionnalités, types de données, problèmes) demandent à être validées et enrichies par de véritables experts ; elles ont en fait pour fonction principale d'amorcer le processus de spécification formelle des concepts utilisés pour la description des traitements.

Un intérêt possible de notre travail réside dans la façon dont nous avons tiré profit des principes et langages du Web sémantique. La présentation imagée de ce dernier sous forme de *layer cake* illustre d'ailleurs fort bien la progression de notre démarche, de la définition de métadonnées structurées au contrôle des valeurs des éléments de description puis à l'exploitation de la sémantique des connaissances représentées. Dès lors, l'adoption des langages du Web sémantique RDF, OWL et SWRL pour la mise en œuvre de notre SBC était un choix naturel. Des enseignements peuvent être tirés des mises en œuvre effectuées, en particulier concernant les difficultés auxquelles nous nous sommes heurtés.

Cela nous amène à évoquer maintenant, après les apports, quelques unes des principales limites de notre travail.

Si l'expressivité des langages OWL et SWRL satisfait bien à nos besoins, les moteurs d'inférence existants, pas encore assez matures, n'ont pas permis d'exploiter pleinement les connaissances de notre SBC. Ce problème sera vraisemblablement résolu dans peu de temps avec l'apparition de nouveaux moteurs.

Plus délicate en revanche est la question de l'acquisition des connaissances dans le cadre d'un SBC destiné, comme dans notre cas, à être accessible depuis une application Web ouverte aux utilisateurs "lambda". En effet, notre modèle de métadonnées fournit le cadre dans lequel les connaissances doivent pouvoir s'exprimer ; or une des difficultés dans la conception d'une interface d'acquisition des règles est d'offrir un moyen simple de désigner les éléments de ce modèle, donc de faire référence à des ressources décrites. Il nous a semblé qu'une notation de type DOM était une bonne solution. La traduction automatique en SWRL des règles acquises sous cette forme s'est avérée quelque peu problématique, quoique non insurmontable. En raison du grand nombre de variables à manipuler, la saisie manuelle de règles SWRL, *via* un éditeur comme Protégé 3.1, est apparue lourde pour les exemples de complexité moyenne (comme celui de l'adaptation de mode d'emploi ER 3).

Au-delà du cas particulier des règles, l'automatisation de la conversion entre les versions SI et SBC de notre base de métadonnées constitue une difficulté, sinon une limite, de notre application. Le choix d'une architecture duale, également effectué, par exemple, par R. Troncy

dans le contexte de l'indexation des documents audiovisuels [Tro04], était nécessaire pour profiter à la fois des avantages d'un schéma XML documentaire ad hoc et des capacités d'inférences d'OWL. Pour autant, il est clair que notre système n'a pas vocation à permettre la conception d'ontologies pour lesquelles des éditeurs spécialisés et des méthodologies spécifiques devront être employés de façons complémentaires. Par ailleurs, nous avons souligné la difficulté qu'il y a à convertir dans notre format XML la partie assertionnelle des ontologies, exprimée en RDF, en raison des multiples syntaxes du langage. L'emploi d'API indépendantes de ces syntaxes, à envisager, impliquerait une certaine lourdeur dans les développements futurs.

Les expériences réalisées confirment l'adéquation des langages du Web sémantique aux besoins de notre contexte, mais pointent divers problèmes techniques qui se posent à l'usage et que nous avons exposés. Si nous ne les avons pas tous résolus, les développements réalisés incitent à penser qu'il est possible de le faire.

Perspectives

L'aide au paramétrage des traitements, complexe dans le domaine géographique, la simulation partielle de leur comportement à des fins de prédiction ou de démonstration, et l'opérationna-lisation de connaissances heuristiques pour mieux orienter l'utilisateur sont quelques-unes des pistes possibles pour poursuivre sur la voie d'un système d'aide "intelligent".

Les scénarios d'adaptation des modes d'emploi que nous avons mis en œuvre sont simples. Cependant, l'utilisateur est parfois confronté à des situations où l'existence de nombreux choix demanderait l'établissement d'un dialogue avec le système. La contrainte de ne sélectionner que des termes proposés, qui est un des principes de base de notre interface utilisateur, pourrait alors montrer des limites. Le recours à des outils de TALN pourrait permettre à l'utilisateur de s'exprimer de façon plus naturelle.

Chercher à simuler le raisonnement de l'expert et chercher à concevoir des métadonnées qui aident l'utilisateur à raisonner sont deux objectifs distincts que nous avons conciliés, les règles de notre SBC étant des ressources certes opérationnelles mais aussi consultables. Un système d'aide perfectionné nécessitera probablement de représenter des règles internes au système, non destinées à l'utilisateur. Le modèle des métadonnées présentées à ce dernier n'est pas forcément appelé à évoluer. Ce qui devra être amélioré, ce sont les ontologies dont les concepts et individus servent à valuer les éléments de description des traitements. Des méthodes de normalisation sémantique de ces ontologies devront être appliquées [Bac00], des consensus entre experts obtenus. Parallèlement, l'intégration et la fusion avec d'autres ontologies du domaine, en particulier celles qui ne vont pas manquer d'apparaître avec le développement annoncé de nombreux services Web géographique, constituent des perspectives de travail nécessaires et prometteuses.

Notre modèle de métadonnées est générique ; il pourra être utilisé pour décrire des traitements informatiques de domaines autres que géographiques. Les principes, langages et techniques mis en œuvre pour la conception du SI et du SBC sont également génériques ; ils sont issus du domaine de l'ingénierie des connaissances. Un prolongement de notre travail pourrait consister à effectuer la liaison avec le domaine voisin du génie logiciel, accompagnant ainsi la tendance de l'informatique de fournir à l'utilisateur/développeur une vision des traitements affranchie des considérations d'implémentation. Dans ce but et dans celui plus général de descriptions des traitements informatiques, le développement de métadonnées support de la connaissance est un objectif d'avenir.

Annexes

Annexe A Diagrammes de classes ISO 19107 et ISO 19115

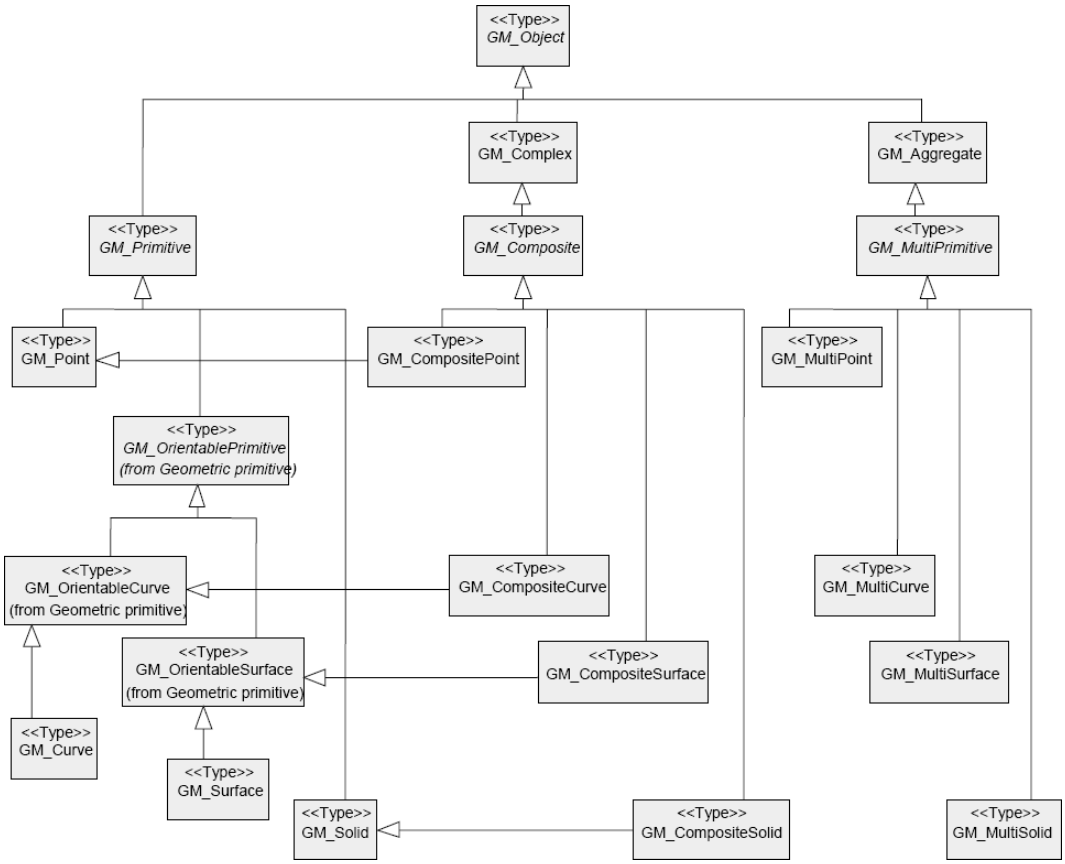


FIG. A.1 – Geometry basic classes with specialization relations (extrait de [ISO01a])

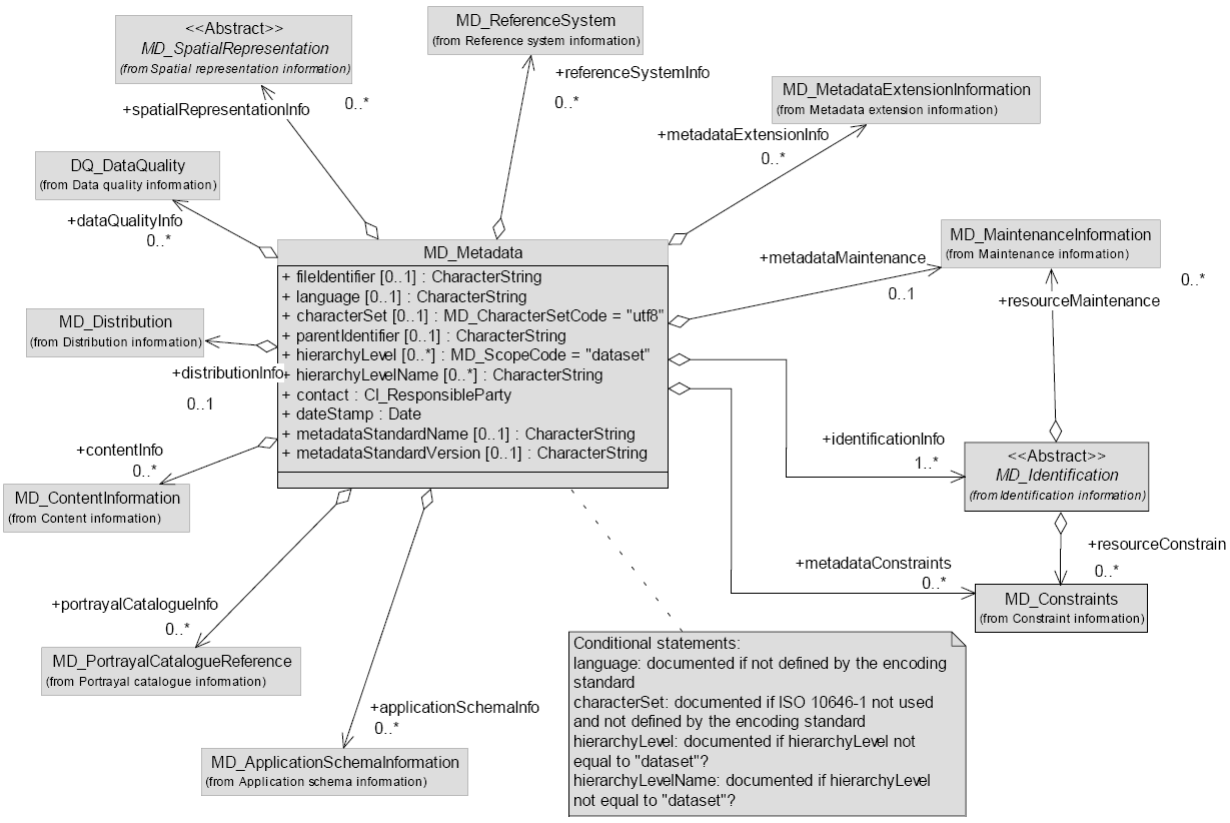


FIG. A.2 – ISO 19115 Metadata entity set information (extrait de [ISO03])

Annexe B Les questionnaires diffusés

Quelles requêtes pour un moteur de recherche de traitements géographiques ?

Bonjour !

Je viens de débuter au COGIT une thèse qui vise entre autres à réaliser un **moteur de recherche de traitements géographiques**, interne à l'IGN. J'aurais besoin de savoir quelles questions vous aimeriez poser à un tel moteur de recherche. Si vous avez quelques minutes à consacrer à ce questionnaire[1], vos réponses me seront très utiles. Merci !

Remarque : nous ne donnons pas ici de définition restreinte du terme "traitement" : c'est à vous de lui donner le sens qui vous est utile. Il peut s'agir par exemple de logiciels, de programmes, d'algorithmes, mais aussi plus abstraitement de savoir-faire de votre domaine.

Q1 – Qui êtes vous ?

☐ Utilisateur de SIG
☐ Développeur
☐ Autre (précisez) :

Q2 – Quelles questions poseriez-vous à un moteur de recherche sur les traitements ?

(exemples : "Quelles sont les différences fonctionnelles entre les deux SIG Arcview et Lamps2 ?", "Comment se comporte le programme Accordéon ?", "Quels algorithmes de détection de contour ont été développés à l'IGN en 1995 ?", "Est-ce qu'il existe un programme de généralisation adapté aux bâtiments de formes irrégulières ?")

Merci de retourner ce questionnaire à Yann Abd-el-Kader, laboratoire COGIT.

[1] Un précédent questionnaire sur les SIG a déjà été diffusé par le COGIT en avril 2002, ses résultats seront également exploités dans la base de données du moteur de recherche.

Une interface de saisie de description de traitements géographiques, phase de validation : décrivez un de vos programmes !

Bonjour !

Un des buts de la thèse que j'ai débutée au COGIT il y a un peu plus d'un an est de réaliser un moteur de recherche de traitements géographiques interne à l'IGN.

Une première application client/serveur permettant **la recherche, la consultation et la saisie | de descriptions de traitements** a été développée.

Le présent questionnaire a pour but de recueillir les remarques que vous aura inspiré l'expérience suivante :

1 – Saisissez la description d'un de vos programmes

Pour cela connectez-vous sur http://walrus (attention Internet Explorer 6, Netscape 7 ou navigateur équivalent requis), allez à la page du formulaire de saisie, remplissez-le et enregistrez-le[1].

Vos remarques (suggestions sur les champs proposés, difficultés rencontrés, etc.) :

2 – Recherchez un traitement

Toujours à partir de http://walrus, utilisez les liens proposés pour mener une recherche de votre choix.

Vous avez recherché :

Avez-vous réussi à exprimer votre requête ? ☐ oui ☐ non ☐ pas exactement

Indiquez comment vous avez procédé :

Remarques, suggestions et difficultés rencontrées :

Merci de retourner ce questionnaire à Yann Abd-el-Kader, laboratoire COGIT.

[1] en récompense vous pouvez constater qu'une portion de camembert vous a été attribuée en cliquant sur Statistiques-->Répartition des "programmes" par "auteur"--> Afficher le diagramme.

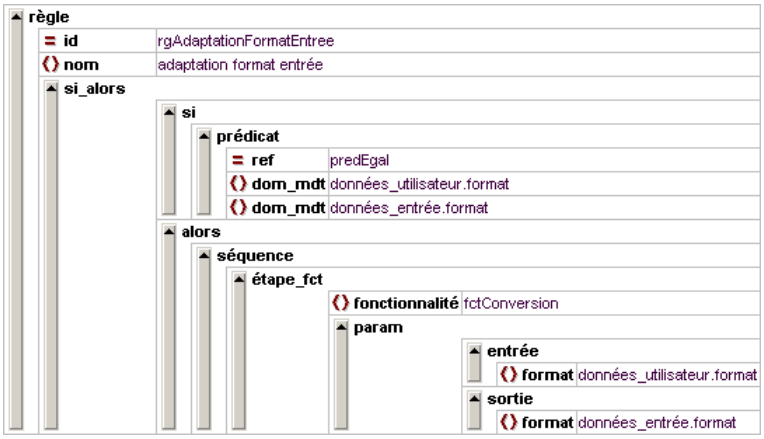
Annexe C Codes de l'application

C.1 Schéma XML – Entrées des traitements

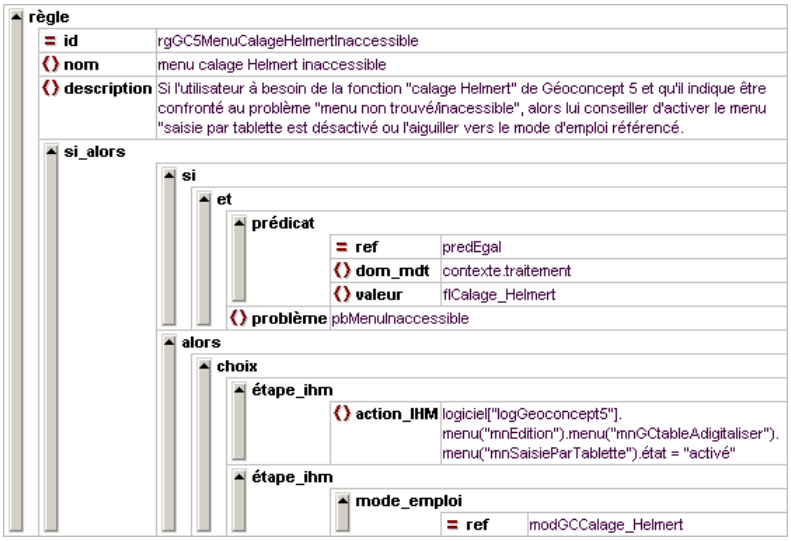
```
<xsd:complexType name="entréeType">
  <xsd:sequence>
    <xsd:element name="nom" type="xsd:string" minOccurs="0"/>
    <xsd:element name="description" type="xsd:string" minOccurs="0"/>
    <xsd:element name="type_donnée" type="xsd:string" minOccurs="0"
                 maxOccurs="unbounded"/>
    <xsd:element name="type_donnée_impl" type="xsd:string" minOccurs="0"/>
    <xsd:element name="format" type="xsd:string" minOccurs="0"
                 maxOccurs="unbounded"/>
    <xsd:element name="objet" type="xsd:string" minOccurs="0"
                 maxOccurs="unbounded"/>
    <xsd:choice minOccurs="0">
      <!-- entrée non modifiable -->
      <xsd:sequence>
        <xsd:annotation>
          <xsd:documentation>modifiable = non</xsd:documentation>
        </xsd:annotation>
        <xsd:element name="modifiable">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="non"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="pte" type="pteSimpleValeurType" minOccurs="0"
                     maxOccurs="unbounded"/>
      </xsd:sequence>
      <!-- entrée modifiable -->
      <xsd:sequence>
        <xsd:annotation>
          <xsd:documentation>modifiable = non</xsd:documentation>
        </xsd:annotation>
        <xsd:element name="modifiable">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="oui"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="pte" type="pteAvantAprèsType" minOccurs="0"
                     maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:choice>
  </xsd:sequence>
</xsd:complexType>
```

Extrait de code A.1: XSD – Le type des entrées en fonction de la valeur de la propriété "modifiable"

C.2 Base de métadonnées XML du SI – Règles pour l'adaptation des modes d'emploi



Extrait de code A.2: XML – Règle pour l'adaptation des format des entrées



Extrait de code A.3: XML – Règle "menu calage Helmert inaccessible" pour l'adaptation des modes d'emploi du SIG Géoconcept 5 (traduction d'une partie de l'exemple fig. 1.7 p. 22)

C.3 Génération d'index de la hiérarchie des modes d'emploi

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
               xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <xsl:template match="/">
      <root>
         <xsl:for-each select="/mdt/mode_emplois/mode_emploi">
            <me id="@id">
               <xsl:if test="substring(type, 1, 3) = 'mod'">
                  <xsl:call-template name="écritPère">
                     <xsl:with-param name="degré">1</xsl:with-param>
                     <xsl:with-param name="idPère" select="type"/>
                  </xsl:call-template>
               </xsl:if>
            </me>
         </xsl:for-each>
      </root>
   </xsl:template>
   <!-- ################################-->
   <xsl:template name="écritPère">
      <xsl:param name="degré"/>
      <xsl:param name="idPère"/>
      <père degré="$degré">
         <xsl:value-of select="$idPère"/>
      </père>
      <xsl:variable name="nouveauPère"
       select="/mdt/mode_emplois/mode_emploi[@id=$idPère]/type"/>
      <xsl:if test="substring($nouveauPère, 1, 3) = 'mod'
              and $degré != 20 ">
         <xsl:call-template name="écritPère">
            <xsl:with-param name="degré" select="($degré) + 1"/>
            <xsl:with-param name="idPère" select="$nouveauPère"/>
         </xsl:call-template>
      </xsl:if>
   </xsl:template>
</xsl:stylesheet>
```

Extrait de code A.4: XSL – Génération d'index de la hiérarchie de modes d'emplois

```
<mdt>
  <mode_emplois>
    <mode_emploi id="modGCAccrochage_au_Z.htm">
      <type>modUtiliserGeoconcept</type>
      <nom>Accrochage_au_Z</nom>
      <url>http://walrus:8080/Mdt/documentsConsultables/CHM/Geoconcept/
           HelpG5/Accrochage_au_Z.htm</url>
    </mode_emploi>
    <mode_emploi id="modGCAccrochage_aux_extrémités_des_linéaires.htm">
      <type>modUtiliserGeoconcept</type>
      <nom>Accrochage_aux_extrémités_des_linéaires</nom>
      <url>http://walrus:8080/Mdt/documentsConsultables/CHM/Geoconcept/
           HelpG5/Accrochage_aux_extrémités_des_linéaires.htm</url>
    </mode_emploi>
    <mode_emploi id="modGCAccrochage_aux_points_existants.htm">
      <type>modUtiliserGeoconcept</type>
      <nom>Accrochage_aux_points_existants</nom>
      <url>http://walrus:8080/Mdt/documentsConsultables/CHM/Geoconcept/
           HelpG5/Accrochage_aux_points_existants.htm</url>
    </mode_emploi>
    <mode_emploi id="modUtiliserGeoconcept">
      <type>modUtiliserSIG</type>
      <nom>utiliser Geoconcept</nom>
      <traitement>logGeoconcept5</traitement>
      <requis>logGeoconcept5</requis>
      <!-- ... -->
    </mode_emploi>
    <mode_emploi id="modUtiliserSIG">
      <type>modUtiliserLogiciel</type>
      <!-- ... -->
    </mode_emploi>
    <mode_emploi id="modUtiliserLogiciel">
      <!-- ... -->
    </mode_emploi>
    <!-- ... -->
  </mode_emplois>
</mdt>
```

Extrait de code A.5: XML – Descriptions initiales des modes d'emploi

```
<root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <me id="modGCAccrochage_au_Z.htm">
    <père degré="1">modUtiliserGeoconcept</père>
    <père degré="2">modUtiliserSIG</père>
    <père degré="3">modUtiliserLogiciel</père>
  </me>
  <me id="modGCAccrochage_aux_extrémités_des_linéaires.htm">
    <père degré="1">modUtiliserGeoconcept</père>
    <père degré="2">modUtiliserSIG</père>
    <père degré="3">modUtiliserLogiciel</père>
  </me>
  <me id="modGCAccrochage_aux_points_existants.htm">
    <père degré="1">modUtiliserGeoconcept</père>
    <père degré="2">modUtiliserSIG</père>
    <père degré="3">modUtiliserLogiciel</père>
  </me>
  <!-- ... -->
</root>
```

Extrait de code A.6: XML – Index généré de la hiérarchie de modes d'emplois

C.4 Document RDF de l'exemple ER 1

```
<Programme rdf:ID="buffer">
  <lieuDeDeveloppement>
    <Organisation rdf:ID="COGIT">
      <appartientOrganisation rdf:resource="#ServiceRechercheIGN"/>
    </Organisation>
  </lieuDeDeveloppement>
  <entree>
    <Donnee rdf:ID="entBuffer">
      <typeAbst>
        <Vecteur rdf:ID="vecteur"/>
      </typeAbst>
    </Donnee>
  </entree>
  <sortie>
    <Donnee rdf:ID="sorBuffer">
      <typeAbst>
        <VecteurSurface rdf:ID="vecteurSurface"/>
      </typeAbst>
    </Donnee>
  </sortie>
</Programme>
```

Extrait de code A.7: RDF – Base de connaissances ER 1 avant inférences (notation arborescente)

C.5 OWL

```
<owl:TransitiveProperty rdf:ID="appartientOrganisation">
  <rdfs:domain rdf:resource="#Organisation"/>
  <rdfs:range rdf:resource="#Organisation"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:TransitiveProperty>
```

Extrait de code A.8: OWL – Transitivé de la propriété appartientLieuDeDev

C.6 Génération de métadonnées au moyen d'un *doclet*

Le code A.9 montre comment fonctionne notre *doclet*. La méthode `start` est définie de façon standard par la classe `Doclet`. C'est en la surchargeant que l'on modifie le comportement du programme javadoc prenant en paramètre le *doclet* en question.

```
import com.sun.javadoc.*;
import java.util.*;

/** doclet qui génère les MDT XML et écrit dans la base */
public class Doclet_MDT extends Doclet {
   public static boolean start(RootDoc root) {
     Handle_MDT handle = new Handle_MDT();
     ClassDoc[] classes = root.classes();
     for (int i = 0; i < classes.length; ++i) {
       ClassDoc cd = classes[i];
       String classe_id = "cla" + cd.typeName();
       handle.ajoutClasse(classe_id, cd.typeName());
       MethodDoc methods[] = cd.methods();

       for (int j=0; j<methods.length; j++) {
           handle.ajoutMéthode( classe_id, methods[j] );
       }
       // on sauve et on ferme
       handle.saveAndClose();
     }
     return true;
   }
}
```

Extrait de code A.9: Java – Doclet pour la génération de métadonnées XML conformes à notre modèle

```
public class Handle_MDT{

   /* ... */

   public void ajoutMéthode( String classe_id, MethodDoc method) {
      try {
         NodeList list = nodeRoot.getElementsByTagName("méthodes");
         Node node = list.item(0);
         String strNewNum = "";
         // on crée le nouveau noeud
         Element newNode = document.createElement("méthode");
         newNode.setAttribute("id", "meth" + method.name() + strNewNum);
         node.appendChild(newNode);
         // INFORMATIONS GENERALES
         addChild(document, newNode, "nom", method.name());
         addChild(document, newNode, "appartient", classe_id);
         addChild(document, newNode, "modificateur", method.modifiers());
         addChild(document, newNode, "description", method.getRawCommentText());
         /* ... */
```

Extrait de code A.10: Java – Classe Handle_MDT utilisée par le *doclet* Doclet_MDT

```
import java.util.regex.*;
      /** ...**/
      /** ANALYSE DES COMMENTAIRES  **/
      if (strLangage.equals("langAvenue")) {
          limitComment = "'([^\n])";
      }
      else if (strLangage.equals("langC")) {
          limitComment = "[*]+ ([^*]*)";
      }
      else if (strLangage.equals("langCpp")) {
          limitComment = "[*]+ ([^*]*)";
      }
      else if (strLangage.equals("langLull")){
          limitComment = "[#*][^#]*#[^*]*([^-]*)";
      }
      // extraction des commentaires
      Pattern p = Pattern.compile(limitComment);
      Matcher m = p.matcher(wholeContent);
      while (m.find()){
         strDescription = strDescription + m.group(1);
      }
      // détection des fonctionnalités

   public void detecteFonctionnalite(String strCode, int indexTableFunct){
       Pattern pattern;
       Matcher matcher;
       String strFctClean;

       strFctClean = supprimeMotsVides(listOfFonctionnalites[indexTableFunct]);

       // pattern pour matcher les blancs
       Pattern p1 = Pattern.compile("[\\s]+");
       // split suivant les blancs
       String[] result = p1.split(strFctClean);

       int localNbTrouvé = 0;

       for (int i=0; i<result.length;i++){
         //on cherche le ième mot du nom de la fonctionnalité
         pattern = Pattern.compile(result[i], Pattern.CASE_INSENSITIVE);
         matcher = pattern.matcher(strCode);
         if (matcher.find()) {
            localNbTrouvé++;
         }
         i++;
       }
       // if (localNbTrouvé == i) System.out.println(motif + " trouvé!!!");
       if (localNbTrouvé > 0) tabTrouvé[indexTableFunct] = true;
       if (((localNbTrouvé == 1) && (result.length == 1)) || (result.length > 1
             && localNbTrouvé > 1)) tabTrouvéHigh[indexTableFunct] = true;
    }
```

Extrait de code A.11: Java – Analyse de code et liens avec les ressources indexées

C.6 Remplissage automatique des formulaire de saisie de traitements Java

```
/* ... */
import javax.xml.parsers.DocumentBuilder ;
import javax.xml.parsers.DocumentBuilderFactory ;
import javax.xml.xpath.* ;
import org.w3c.dom.Document ;

/* ... */
/** 2- RECUPERATION DES INFORMATIONS DE LA DESCRIPTION GENEREE PAR LE DOCLET */
try{
 DocumentBuilder builder = DocumentBuilderFactory.newInstance().newDocumentBuilder() ;
 Document document = builder.parse(new File("C:/Abdelkader/MDT/generationXML/outputMDTDoclet.xml")) ;
 //création du XPath
 XPathFactory fabrique = XPathFactory.newInstance() ;
 XPath xpath = fabrique.newXPath() ;
 //évaluation de l'expression XPath
 XPathExpression exp = xpath.compile("/mdt/classes/classe[@id='" + id + "']/nom") ;
 nom = exp.evaluate(document) ;

 /* ... */
}
/** 3- LA MISE A JOUR DU FORMULAIRE DE SAISIE */
out.println("<html><head><link href=\"http://walrus:8080/Mdt/css/coUA.css\" type=\"text/css\"
rel=\"stylesheet\"/>" +
"<script language=\"Javascript\">" +
// Informations générales nom du programme
"window.opener.document.forms['saisieForm'].elements['nom'].value='"     + nom + "';" +
"window.opener.document.forms['saisieForm'].elements['version'].value='" + version + "';" +
"window.opener.document.forms['saisieForm'].elements['date_modification'].value='"     +
date_modification + "';" +
"window.opener.document.forms['saisieForm'].elements['cb_auteur1'].options[" + pos_auteur1 +
"].selected='1';" +
"window.opener.document.forms['saisieForm'].elements['cb_auteur_description'].options[" +
pos_auteur_description + "].selected='1';" +
"window.opener.document.forms['saisieForm'].elements['cb_lieu_de_developpement'].options[" +
pos_lieu_de_developpement + "].selected='5';" +

// Ce que fait le programme
"window.opener.document.forms['saisieForm'].elements['description'].value='"   + description + "';"
+
"window.opener.document.forms['saisieForm'].elements['main_inp_fct1'].value='" + fonctionnalité +
"';" +
"window.opener.document.forms['saisieForm'].elements['main_fct1'].value='" + strRealise1 + "';" +
"window.opener.document.forms['saisieForm'].elements['ztnb_entree'].value='" + nb_entrées + "';" +
"window.opener.document.getElementById('entree2').style.display = 'block';" +
"window.opener.document.getElementById('detail_entree2').style.display = 'block';" +
"window.opener.document.forms['saisieForm'].elements['nomEntree1'].value='" + nomEntree1 + "';" +
"window.opener.document.forms['saisieForm'].elements['text_tdE1'].value='" + text_tdE1 + "';" +
"window.opener.document.forms['saisieForm'].elements['text_tdE1_impl'].value='" + text_tdE1_impl +
"';" +

/* ... */
```

Extrait de code A.12: Java – Génération du Javascript remplissant le formulaire de saisie

C.8 Exécution de requêtes SeRQL avec Sesame 1.2.1

```
import org.openrdf.model.Value ;
import org.openrdf.sesame.Sesame ;
   /* ... */
   java.net.URL sesameServerURL = new java.net.URL("http://localhost:8080/sesame/") ;
   SesameService service = Sesame.getService(sesameServerURL) ;
   service.login("testuser", "opensesame") ;
   SesameRepository myInRepository = service.getRepository("mem-rdfs-db") ;

   String query = "" ;
   /* Construction de la requête à partir des champs du formulaire soumis par l'utilisateur */
   /* ... */
   /* Lecture des résultats */
   QueryResultsTable resultsTable = myInRepository.performTableQuery(QueryLanguage.SERQL,
                                                                     query) ;
   /* Ecriture du fichier XML des résultats pour l'appli Web */
   /* ... */
```

Extrait de code A.13: Java – Exécution de requête SeRQL (d'après la section 7.2 de [Ope05])

C.9 Classification avec Jena 2.2

```
import com.hp.hpl.jena.rdf.model.* ;
import com.hp.hpl.jena.util.ModelLoader ;
import com.hp.hpl.jena.reasoner.* ;
import com.hp.hpl.jena.vocabulary.RDF ;

public class ClassifArcview {

  public static void main(String[] args) {

    Model schema = ModelLoader.loadModel("file:SBC/ER2suite.rdf") ;
    Model data = ModelLoader.loadModel("file:SBC/ER2suite.owl") ;

    Reasoner reasoner = ReasonerRegistry.getOWLReasoner() ;
    reasoner = reasoner.bindSchema(schema) ;
    InfModel infmodel = ModelFactory.createInfModel(reasoner, data) ;

    // individu empâtement virage sérré isolé
    Resource empatement = infmodel.getResource("urn :mdt/instEmpatementVirageSerreIsole") ;

    // instEmpatementVirageSerreIsole est-il un ProblemeLisibilite ?
    Resource pbLisibilite = infmodel.getResource("urn :mdt/ProblemeLisibilite") ;
    if (infmodel.contains(empatement, RDF.type, pbLisibilite)) {
      System.out.println("empâtement virage sérré isolé est un problème de lisibilité !") ;
    }
  }
}
```

Extrait de code A.14: Java – Classification de problème avec Jena 2.2 (utilisé pour ER2 suite)

Annexe D Logiciels, programmes et librairies utilisés

Application côté serveur		
Apache 1.3	Apache Software Foundation	Serveur HTTP
Tomcat 5.5.9	Apache Software Foundation	Moteur de servlet Java
JDK 1.5 programme javadoc com.sun.javadoc java.lang.reflect java.util.regex javax.xml ...	Sun Microsystems	 génération de documentation de code Java création de doclet reflection du code Java Moteur d'expressions régulières Parser XML, moteurs XSL et XPath
Sesame 1.2.1	Aduna	Plateforme et moteur RDF(-S)
com.oreilly.servlet.multipart	O'Reilly Media	Échange de données via protocole HTTP (images, shp, dbf, etc.)
Jchart	R. Piola	Génération dynamique de diagrammes statistiques
Application côté client		
uk.ac.leeds.ccg.geotools	Confluence	Visualisation des données au format SHP (applet Java)
Logiciels utilisés en marge de l'application		
Amaya 9.2.1	W3C & INRIA	Éditeur HTML et MathML
Html Help Workshop 4.74	Microsoft	Décompilateur de fichiers d'aide CHM
Jena 2.2	Hewlett Packard	Moteur d'inférence OWL
Lore's source converter	L. Haendel	Convertisseur de code source en pages HTML avec coloration syntaxique
Protégé 3.1	Stanford Medical Informatics	Éditeur d'ontologies OWL de règles SWRL
XML Spy 2004	Altova	Éditeur XML, XSL et XSD

Tab. A.1 – Logiciels, programmes et librairies utilisées

Annexe E Laboratoires de recherche et services de production à l'IGN

Il existe à l'IGN quatre laboratoires de recherche et une dizaine de services de production. La recherche de l'IGN participe aux évolutions techniques qui ont profondément modifié l'activité de l'établissement, avec l'émergence de l'informatique, de la télédétection, de la géodésie spatiale et des bases de données géographiques ainsi que le renouveau des instruments aéroportés et de la cartographie [IGN98]. Les résultats des laboratoires de recherche profitent ainsi aux services de production, le but étant d'aboutir à moyen ou long terme à un procédé, un produit, ou un outil de production.

En particulier, lorsque des besoins spécifiques tels que le détramage des cartes scannées ou la généralisation cartographique ne peuvent être satisfaits par les logiciels commerciaux standards, les laboratoires de recherche de l'IGN conçoivent et implémentent leurs propres programmes informatiques ; le passage en production s'effectuant ensuite en cas de succès.

La recherche, qui se situe donc en amont du processus de production, s'articule autour de quatre disciplines. À chaque discipline correspond un laboratoire :

Le **COGIT** (Conception Objet et Généralisation de l'Information Topographique) étudie

les problématiques liées à l'utilisation des données topographiques vectorielles. Ses compétences concernent les bases de données localisées et la cartographie. Les recherches portent sur la gestion, la dérivation et la diffusion de l'information géographique.

Le **LAREG** (LAboratoire de Recherche En Géodésie) est un laboratoire de l'ENSG (Ecole Nationale des Sciences Géographiques) couvrant plusieurs secteurs de la géodésie.

Le **LOEMI** (Laboratoire d'Optique, d'Electronique et de Micro-Informatique) est un laboratoire d'instrumentation. Les recherches visent à améliorer les prises de vue aériennes, par le développement de nouveaux capteurs et des techniques de trajectographie. Le LOEMI étudie aussi des instruments de métrologie.

Le **MATIS** (Méthodes d'Analyses et de Traitement d'Images pour la Stéréo-restitution) étudie la photogrammétrie et des méthodes d'analyse et de traitement d'images. Les recherches portent sur les photographies aériennes. Elles visent à détecter et restituer les éléments du paysage, généralement en 3 dimensions par stéréoscopie. Le MATIS travaille en outre sur les plans scannés.

Citons les principaux services de production de l'IGN :

Le **SAA** (Service des Activités Aériennes) prend en photo le territoire vu du ciel, hier en argentique, aujourd'hui en numérique.

Le **SBI** (Service des Bases de données Images) corrige (ombres, jointures, redressement des photos pour l'obtention d'ortho-images simulant la prise de vue à la verticale parfaite, etc.), et stocke les images.

Le **SBV** (Service des Bases de données Vecteurs) produit des données en mode vecteur à différentes échelles. Le produit qui mobilise le plus grand nombre de salariés est la composante topographique du référentiel à grande échelle (RGE) : la BD Topo. Il y a deux filières thématiques distinctes : la saisie de la végétation par un processus semi-automatique et une filière par restitution photogrammétrique et intégration de données existantes pour tous les autres thèmes. La deuxième mission concerne la mise à jour en continu des référentiels.

Le **SDC** (Service De la Cartographie) se charge de la production des cartes.

Le **SDOG** (Service de la DOcumentation Géographique) est principalement chargé de l'archivage et de la mise à disposition des données produites par l'IGN : données analogiques (cartes, photographies aériennes, ...) ou numériques (BD Topo, BD Ortho, ...). Ces activités sont partagées entre quatre unités : le serveur général, la photothèque nationale, la cartothèque, le centre de documentation.

Le **SGN** (Service de Géodésie et Nivellement) a actuellement pour activités principales les réseaux matérialisés de géodésie et de nivellement, les réseaux de stations GPS permanents, l'information géodésique.

Le **SPI** (Service de Photogravure et d'Impression) est le dernier maillon de la chaîne cartographique. Il est prestataire de service pour les autres unités de l'IGN. Il assure la partie Arts Graphiques de la réalisation de la carte.

Bibliographie

[ABC+03] D. Ayala, C. Browne, V. Chopra, P. Sarang, K. Apshankar, et T. McAllister. *Services Web Open Source*. collection Programmer to Programmer, trad. de l'anglais par E. Burr, V. Campillo et V. Warion, Campus Press, Paris, 2003.

[ALR96] N. Aussenac-Gilles, P. Laublet, et C. Reynaud. *L'acquisition des connaissances, une composante à part entière de l'informatique du futur*. Acquisition et ingénierie de la connaissance - Tendances actuelles, Cepadues, pp. 3-25., 1996.

[AS94] J-M. Alliot et T. Schiex. *Intelligence artificielle et informatique théorique*. Cépaduès, collection Intelligence Artificielle, 1994.

[ATI02] ATICA. *Guide de choix et d'usage des licences de logiciels libres pour les administrations - Annexe : Analyse détaillée des licences*. rapport au premier ministre de la république française, 2002. `http://www.adae.gouv.fr/upload/documents/analyse_detaillee.pdf`.

[Aus89] N. Aussenac. *Conception d'une méthodologie et d'un outil d'acquisition des connaissances expertes*. Thèse de doctorat d'informatique. Université Paul Sabatier de Toulouse, 1989.

[BA03] J-M. Bézard et S. Ariès. *La méthode MASK – Présentation pour la capitalisation des connaissances*. 2003. `http://perso.wanadoo.fr/serge.aries/presentation/MASKmet/frame.htm`.

[Bac92] B. Bachimont. *Le contrôle dans les systèmes à base de connaissances*. Hermès, 1992.

[Bac00] B. Bachimont. Engagement sémantique et engagement ontologique : conception et réalisation d'ontologies en ingénierie des connaissances. *In J. Charlet, M. Zacklad, G. Kassel et D. Bourigault,* Ingénierie des Connaissances : Évolutions récentes et nouveaux défis*, Eyrolles*, 2000.

[Bac04] B. Bachimont. *Ingénierie des connaissances*. page de présentation des travaux de recherche de B. Bachimont, hébergée sur le site de Université de Technologie de Compiègne, 2004. `http://www.utc.fr/~bachimon/Recherche.html` (accédé le 19 janvier 2006).

[Ban00] C. Bandza. *Des méthodes de formalisation des connaissances et de MKSM en particulier*. Thèse professionnelle du mastère MSIT, Management des Systèmes dÍnformations et des Technologies, HEC-Mines, 2000. `http://www.hec.ensmp.fr/Theses/Theses2000/Bandza.doc`.

[Bar98] J-P. Barthes. *Les systèmes à base de connaissances*. cours de l'Université de Technologie de Compiegne (UTC), 1998. `http://www.hds.utc.fr/~barthes/IA03/KADS.html`.

[Bar04] S. Bard. *Méthode d'évaluation de la qualité de données géographiques généralisées – Application aux données urbaines*. Thèse de doctorat d'informatique de l'Université de Paris 6, 2004.

[BB03] T. Badard et A. Braun. Oxygene – d'une plate-forme interopérable au déploiement de services web géographiques. *Les SIG sur le Web, revue internationale de Géomatique, vol. 13, n°3/2003, Hermès Sciences, Lavoisier, Paris, pp. 411-430*, 2003.

[BCES04] J-F. Baget, E. Canaud, J. Euzenat, et M. Saïd-Hacid. Les langages du Web Sémantique. *In [CLR03], chapitre 2, pp. 9-24*, 2004.

[BCLJ04] C. Bousquet, C.Henegar, A.Lillo-Le Louët, et M-Ch. Jaulent. Apport d'une modélisation ontologique pour la détection du signal en pharmacovigilance. *In Actes de la conférence IC'2004, 15èmes Journées francophones d'ingéniérie des connaissances*, Lyon, pp.187-198, 2004.

[BD00] O. Boiral et I. Dostaler. Mobiliser les connaissances tacites : l'exemple d'un atelier d'assemblage électronique. *IXème Conférence Internationale de Management Stratégique, AIMS*, Montpellier, 2000.

[BFIM98] T. Berners-Lee, R. Fielding, U.C. Irvine, et L. Masinter. *Uniform Resource Identifiers (URI) : Generic Syntax.* RFC 2396, IETF, 1998.

[BHL01] T. Berners-Lee, J. Hendler, et O. Lassila. The Semantic Web. Scientific American*, n°284, p.34-43*, 2001.

[BL05] Aduna B.V. et Sirma AI Ltd. *The SeRQL query language (revision 1.2).* 2005. `http://www.openrdf.org/doc/sesame/users/ch06.html`.

[BM02] T. Berners-Lee et E. Miller. The semantic web. *présentation W3C, diapositive 17* Enabling Standards & Technologies – Layer Cake, 2002. `http://www.w3.org/Talks/2002/01/10-video/slide17-0.html`.

[Bod97] L. Bodet. *Réalisation d'une machine virtuelle Java sous le système d'exploitation Plan9.* Mémoire de maîtrise d'informatique option micro-informatique / micro-électronique, Université Paris 8, p.122, 1997. `http://plan9.aichi-u.ac.jp/netlib/java/rapport.ps`.

[Bor02] P. Bordin. *SIG – concepts, outils et données.* Hermès science, Lavoisier, p. 103, 2002.

[Boy02] P. Boyer. *Et l'homme créa les dieux.* Folio essais n°414, pp.138-146, 2002.

[Bra03] A. Braun. *Manuel OXYGENE.* Document interne Cogit, 2003.

[Buc02] B. Bucher. *L'aide à l'accès à l'information géographique : un environnement de conception coopérative d'utilisations de données géographiques.* Thèse de doctorat d'informatique de l'Université de Paris 6, 2002.

[CC99] J. Caussanel et E. Chouraqui. Informations et connaissances : quelles implications pour les projets de capitalisation des connaissances. *In G. Dupoirier et J-L. Ermine, Gestion des documents et gestion des connaissances, Document numérique 3, n°3-4*, décembre 1999, Hermès, 2000. p.101-119, 1999.

[CCZC02] J. Caussanel, J-P. Cahier, M. Zacklad, et J. Charlet. Les Topic Maps sont-ils un bon candidat pour l'ingénierie du Web Sémantique ? *In Actes de la conférence IC 2002, Rouen*, 2002.

[CDF04] O. Corby, R. Dieng, et C. Faron. Querying the Semantic Web with the Corese Search Engine. *In Actes de la Conférence Européenne d'Intelligence Artificielle ECAI'2004 (dans le cadre de la conférence PAIS)*, 2004. `http://www-sop.inria.fr/acacia/pub/2004/corby-pais2004.pdf`.

[Cha87] A. F. Chalmers. *Qu'est-ce que la Science ?* Ed. la Découverte, Le livre de Poche n°4126, 1987.

[Cha00] G. Chartron. *Standards, normes, documents numériques.* Urfist de Paris, 2000. `http://www.ccr.jussieu.fr/urfist/presse/standard/coursintro.htm`.

[Cha03] J. Charlet. *L'ingénierie des connaissances – Développements, résultats et perspectives pour la gestion des connaissances médicales*. Mémoire d'Habilitation à diriger des recherches, Université Pierre et Marie Curie, p. 59., 2003.

[Che92] J-P. Cheylan. Classification des fonctions de traitements dans les SIG : éléments de synthèse. *communication à la conférence SIG-GIS*, 1992.

[Cho05] S. Chollet. Automatisation de la composition de web services. *LIMOS Clermont-Ferrand, séminaire équipe BD, 15 avril 2005*, 2005.

[CL05] F. Comte et M. Leclère. OWL-SG : un sous-langage pour la famille OWL. *Journée thématique : Raisonner sur le Web Sémantique avec des Graphes, plateforme AFIA, juin 2005, Nice*, 2005. `http://www.lirmm.fr/~leclere/recherche/rwsg/ComteLeclereMugnier.ps`.

[CLR03] J. Charlet, P. Laublet, et C. Reynaud. *Le Web sémantique*. rapport final de l'action spécifique 32 CNRS/STIC (version 3 de décembre 2003), publié chez Cépaduès (Hors-série de la collection Information interaction intelligence), 2003.

[Coa03] The OWL Service Coalition. *OWL-S : Semantic Markup for Web Services (Technical Overview – a white paper describing the key elements of OWL-S)*. 2003. `http://www.daml.org/services/owl-s/1.0/owl-s.pdf`.

[Com02] IEEE Learning Technology Standards Committee. *Draft Standard for Learning Object Metadata*. 2002. `http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf`, New York, USA.

[Con99] AGENT Consortium. *Selection of Basic Algorithms*. Rapport public du projet AGENT version 2.3, numéro de projet ESPRIT/LTR/24939, 1999. `http://agent.ign.fr/deliverable/DD2.pdf`.

[Con01a] Open GIS Consortium. *Basic Model Draft Candidate Implementation Specification 0.0.8*. 2001.

[Con01b] Open GIS Consortium. *Web Services – Service Registry (version 0.2)*. OpenGIS Project Document OGC 01-082 21-12-2001, 2001.

[Con03] Open GIS Consortium. *Geographic information – Web Map Service interface, (unofficial) ISO DIS 19128*. 2003.

[Cos03] R. Costello. *OWL Web Ontology Language*. The MITRE Corporation, 2003.

[CPSV03] N. Cullot, C. Parent, S. Spaccapietra, et C. Vangenot. Des SIG aux ontologies géographiques. *Les SIG sur le Web, revue internationale de Géomatique*, vol. 13, n°3/2003, Hermès Sciences, Lavoisier, Paris, pp. 285-306, 2003.

[Cro98] H. Le Crosnier. *Texte et informatique*. Cours d'informatique, version 0.91, Université de Caen, 1998. `http://ariane.mpl.ird.fr/textes/crosnier.pdf`.

[CVM03] Y. Caron, N. Vincent, et P. Makris. Mesure de la qualité de la compression par l'utilisation de la loi de Zipf. *Compression et Représentation de Signaux Audiovisuels - CORESA'03*, Lyon, 16-17 Janvier 2003, pp. 239-242, 2003.

[Dad05] M. Dadou. *Aide à la saisie de connaissances d'experts : conception d'outils de saisie et d'analyse*. rapport de stage effectué au laboratoire COGIT, IGN, pour un Master professionnel de sciences cognitives appliquées, Université Victor Segalen Bordeaux 2, 2005.

[Del02] J-P. Delahaye. *L'intelligence et le calcul, de Gödel aux ordinateurs quantiques*. Bellin – Pour la Science, p.152–156, 2002.

[DRL00] Y. Delmas-Rigoutsos et R. Lalement. *La logique ou l'art de raisonner*. coll. Quatre à quatre – Le Pommier, Fayard, 2000.

[DS96] J. Denègre et F. Salgé. *Les systèmes d'information géographique*. collection *Que sais-je ?*, PUF, Paris, p.12, 62, 1996.

[Duc04] C. Duchêne. *Généralisation par agents communicants : le modèle CARTACOM. Application aux données topographiques en zone rurale.* Thèse de doctorat d'informatique de l'Université de Paris 6, 2004.

[Dup99] J-P. Dupouy. *Aux origines des sciences cognitives.* nouvelle édition, Paris, La découverte/Poche sciences humaine et sociales n°69, p.22, 1999.

[EB04] J. Euzenat et J-F. Baget. *OWL : un langage d'ontologies pour le web – Une courte présentation en français.* INRIA Rhône-Alpes, 2004. `http://www.inrialpes.fr/exmo/cooperation/webont/owl.html`.

[EBS00] J. Erceau, P. Benhamou, et A. Seve. *ONERA - mission VIE, Projet Gestion des Connaissances.* 2000. `http://www.onera.fr/vie/mksm.html`.

[Ele04] D3E Electronique. *Guide de l'utilisateur ArcView 8.3.* 2004.

[Erm03] J-L. Ermine. *La Gestion des connaissances.* Hermès Science, Lavoisier, 2003.

[Euz99] J. Euzenat. *Sémantique des représentations des connaissances.* Notes de cours de DEA d'informatique, Université Joseph Fourier - Grenoble I, 1999.

[FC99] V. Ficet-Cauchard. *Réalisation d'un système d'aide à la conception d'applications de Traitements d'Image : une approche basée sur le Raisonnement à Parti de Cas.* Thèse de doctorat d'informatique de l'Université de Caen, 1999.

[FCRP99] V. Ficet-Cauchard, M. Revenu, et C. Porquet. Conception d'applications de traitement d'images par raisonnement à partir de cas : définition, utilisation et gestion de cas. *RàPC'99 Plate Forme AFIA, pp. 7-16, Palaiseau*, 1999.

[FK04] J-Y. Fortier et G. Kassel. Présentation 'sur mesure' de l'information. *In Présentation de l'information sur mesure, RSTI série RIA (Revue des Sciences et Technologies de l'Information – Revue d'intelligence artificielle)*, vol. 18 - n°4/2004, sous la direction de C. Paris et N. Colineau, Hermes-Lavoisier, pp. 515-547, 2004.

[Für05] F. Fürst. *L'ingénierie ontologique.* Rapport de recherche N°02-07, octobre 2002, Institut de Recherche en Informatique de Nantes, 2005. `http://www.sciences.univ-nantes.fr/info/perso/permanents/furst/papers/RR02-07.ps`.

[Fra94] République Française. *circulaire du 14 février 1994 relative à la diffusion des données publiques.* JO, 1994.

[FT05] F. Fürst et F. Trichet. Aligner les ontologies lourdes : une méthode basée sur les axiomes. *In Actes de la conférence IC'2005, 16èmes Journées francophones d'ingéniérie des connaissances*, Nice, 2005.

[Ges05] N. Gesbert. *Formalisation des spécifications de bases de données géographiques en vue de leur intégration.* Thèse de doctorat d'informatique de l'Université de Marne-la-Vallée, 2005.

[GFLC03] A. Gomez, M. Fernandez-Lopez, et O. Corcho. Ontological Ingeneering. *Springer, Advanced Information and Knowledge Processing*, 2003.

[GHVD03] B. Grosof, I. Horrocks, R. Volz, et S. Decker. Description Logic Programs : Combining Logic Programs with Description Logic. *In Actes de la conférence Word Wide Web 2003*, 2003. `http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/p117-grosof.pdf`.

[GM97] O. Günter et R. Müller. From GISystems to GIServices : Spatial computing on the Internet Marketplace. *In actes de la conférence Interopating Geographic Information Systems, Santa Barbara, CA, 3-4 décembre*, 1997.

[Gom04] R.M. Gomez de Regil. Normes et standards : un processus de normalisation en cours. *présentation aux Journées d'études sur l'indexation des ressources pédagogiques numériques. 16 novembre 2004*, Lyon, 2004.

[GP03] S. Garlatti et Y. Prié. Adaptation et personnalisation dans le Web sémantique. *In [CLR03], chapitre 6, pp. 79-91*, 2003.

[Gro00] GSDI Technical Working Group. *Developing Spatial Data Infrastructures : the SDI Cookbook, v.1.0*. Douglas Nebert Editions, 2000.

[Gru93] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition 5(2) :199-220*, 1993.

[Gru95] M. Grundstein. La capitalisation des connaissances de l'entreprise, système de production de connaissances. *In Actes du congrès* L'entreprise apprenante et les sciences de la complexité, Aix-en-Provence, 1995.

[GS04] F. Gandon et N. Sadeh. Gestion de connaissances personnelles et contextuelles, et respect de la vie privée. *In actes d'IC'2004, 15èmes Journées francophones d'Ingéniérie des Connaissances*, Lyon, 2004.

[HBF+91] J-P. Haton, N. Bouzid, F.Charpillet, M-C. Haton, B. Lâasri, H. Lâasri, P. Marquis, T. Mondot, et A. Napoli. *Le raisonnement en Intelligence Artificielle*. InterEditions, Paris, 1991.

[Hil01] D. Hillmann. *Using Dublin Core*. Dublin Core Metadata Initiative, 2001. `http://dublincore.org/documents/usageguide/` (traduction française de G. Teasdale sur `http://www.bibl.ulaval.ca/DublinCore/usageguide-20000716fr.htm`.

[HPSB+04] I. Horrocks, P. Patel-Schneider, H. Boley, S.Tabet, B. Grosof, et M. Dean. *SWRL : A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission 21 May 2004, 2004. `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`.

[Hub03] F. Hubert. *Modèle de Traduction des Besoins d'un Utilisateur pour la Dérivation de Données Géographiques et leur Symbolisation par le Web*. Thèse de doctorat informatique Université de Caen, 2003.

[IGN90] IGN. *Charte Logiciel*. IGN, Direction technique DT/133, 1990.

[IGN98] IGN. *La recherche en 1998, Bulletin d'information de l'IGN n°70*. service de la recherche de l'IGN, 1998.

[IGN03] IGN. *Descriptitif technique de la BD TopoPays – Édition 2.1*. 2003. `http://www.ign.fr/telechargement/MPro/produit/BD_TOPO/JT_Agglo/DT_BDTOPOPays_1_2.pdf`, p.6.

[IGN04] IGN. *Rapport d'activité 2003 de l'Institut Géographique National*. Direction commerciale de l'IGN, 2004.

[IGN05] IGN. *Spécification de contenu de la BD CARTO*. Version 3, Service de Base de données Vecteur, 2005.

[Ini04] Dublin Core Metadata Iniative. *Dublin Core Metadata Element Set, Version 1.1 : Reference Description*. http ://dublincore.org/documents/dces/, 2004.

[Isa01] A. Isaac. *Vers la mise en oeuvre informatique d'une méthode de conception d'ontologies*. Mémoire de DEA MIASH, stage effectué à l'INA (Institut National de l'Audiovisuel), Université Paris 4, 2001.

[Isa05] A. Isaac. *Conception et utilisation d'ontologies pour l'indexation de documents audiovisuels*. Thèse de doctorat d'informatique de l'Université Paris IV – Sorbonne, 2005.

[ISO01a] ISO. *ISO 19107, Geographic information – Spatial schema*. International Standard, 2001.

[ISO01b] ISO. *ISO 19119 : Geographic information – Services*. Draft International Standard, 2001. `http://www.ncits.org/ref-docs/DIS19119.PDF`.

[ISO03] ISO. *ISO 19115, Geographic information – Metadata*. International Standard, 2003.

[ISO05a] ISO. *Geographic information – Metadata – Part 2 : Extensions for imagery and gridded data*. ISO TC TC211/SC N – ISO/WD 19115-2.5, version du 20-05-2005 (statut du document : préparatoire), 2005.

[ISO05b] ISO. *ISO en bref*. 2005. `http://www.iso.org/iso/fr/aboutiso/isoinbrief/isoinbrief.html`.

[JGR00] H.A. Jacobsen, O. Günter, et G. Riessen. MMM, Component leasing on the WWW. *journal NETNOMICS*, vol.2, Baltzer Science Publishers, Pays-Bays, pp. 191-219, 2000.

[JJPJ04] J.Bovet, J-P.Ertz, et J.Hess. *Bases de Programmation – Introduction et présentation du cours*. 2004. `http://cours.eivd.ch/algo/pdfs/BaseDeProg_Chap0_Intro.pdf`.

[Kay97] D. Kayser. *La représentation des connaissances*. Hermès, 1997.

[KFG05] O. Khayati, A. Front, et J-P. Giraudin. Génération et appariement de spécifications formelles de diagrammes de classes pour la recherche de composants. *In Actes de la conférence INFORSID 2005*, Grenoble, pp. 235-250, 2005.

[KG05] Racer Systems GmbH & Co. KG. *RacerPro User's Guide Version 1.9*. 2005. `http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf`.

[KMN04] H. Knublauch, M.A. Musen, et N.F. Noy. *Creating Semantic Web (OWL) Ontologies with Protégé*. Stanford Medical Informatics, 2004. `http://protege.stanford.edu/plugins/owl/publications/2004-07-06-OWL-Tutorial.ppt`.

[Kor99] Kordon. *UML*. cours de génie logiciel – licence informatique Université Pierre et Marie Curie, Paris 6, 1999.

[Kor03] J. Korczak. *Systèmes Experts – CLIPS*. cours de d'informatique de l'Université Louis Pasteur Strasbourg, CNRS – Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection, 2003. `http://www-ipst.u-strasbg.fr/jjk/SI-IPSTSystemesExperts.pdf`.

[KT03] P. Kellert et F. Toumani. Les Web Services sémantiques. *In [CLR03], chapitre 7, pp. 93-106*, 2003.

[LdB02] R. Lemmens et R.A. de By. Distributed GIS and metadata - Methods for the description of interoperable GIS components. *In actes de la conférence International Workshop on Mobile and Internet GIS*, Wuhan, China, 15-16 août, 2002.

[Lew04] D. Lewis. *OWL-S Applications and Issues Support ontologies*. AI planing, 2004. `http://metadatos.cl/wiki/attach?page=RodrigoFrez%2Fowl-s-applications-and.pdf`.

[LGS02] B. Le-Grand et M. Soto. *TopicMaps et navigation "intelligente" sur le Web Sémantique*. présentation de l'AS2W – Action Spécifique Web Sémantique, 2002. `http://www.lalic.paris4.sorbonne.fr/stic/octobre/octobre/apr/LeGrand.pdf`.

[LJP98] N. Lopez, J.Migueis, et E. Pichon. *Intégrer UML dans vos projets*. Eyrolles Informatiques, pp. 25-41 et 81-114, 1998.

[LRC02] P. Laublet, C. Reynaud, et J. Charlet. Sur quelques aspects du Web sémantique. *In Actes des deuxièmes assises nationales du GdR I3 (information, interaction, intelligence), Cépaduès*, 2002.

[MABL03] C. Moulin, M-H. Abel, A. Benayache, et D. Lenne. Modélisation d'une Mémoire de Formation : le choix des Topic Maps. *In Actes de la conférence IC 2003*, Grenoble, 2003.

[Mar05] G. Martinoty. *Reconnaissance de matériaux sur des images aériennes en multirecouvrement, par identification de fonctions de réflectances bidirectionnelles.* Thèse de doctorat en méthodes physiques en télédétection de l'Université Paris 7 – Denis Diderot, 2005.

[ME00] S. Moisan et J-L. Ermine. Gestion opérationnelle des connaissances sur les codes. *In Actes de la conférence IC'2000, Journées francophones d'Ingénierie des Connaissances*, Toulouse, 2000.

[Mei05] J. Mei. *An Engine for SWRL rules in RDF graphs.* AG Netzbasierte Informationssysteme / Networkbased Informationsystems, Université libre de Berlin, 2005. `http://www.inf.fu-berlin.de/inst/ag-nbi/research/swrlengine/` (version du 27/06/2005).

[Mic03] P. Michaux. *Réalisation d'une interface de consultation pour les traitements de généralisation.* Rapport de stage de DESS Imagerie Electronique de l'Université de Paris VI, 2003.

[Moi87] J.L. Le Moigne. Qu'est-ce qu'un modèle ? *Les modèles expérimentaux et la clinique, Confrontations psychiatriques, numéro spécial consacré au modèles*, 1987.

[Mug02] M-L. Mugnier. *Candidature au diplôme d'habilitation à diriger des recherches – Document de synthèse.* 2002. `http://www.lirmm.fr/~mugnier/hab/MLMsynthese.doc`.

[Mus01] S. Mustière. *Apprentissage supervisé pour la généralisation cartographique.* Thèse de doctorat d'informatique de l'Université Paris VI, 2001.

[NB05] M. Neun et D. Burghardt. *Web Services for an Open Generalisation Research Platform.* 8th ICA WORKSHOP on Generalisation and Multiple Representation, La Corogne, 2005.

[OAS02] OASIS. *UDDI Version 3.0.* UDDI Spec Technical Committee Specification, 19 July 2002, 2002. `http://www.oasis-open.org/committees/uddi-spec/`.

[Off04] Journal Officiel. *Journal Officiel de la République Française, décret n° 2004-1246 du 22 Novembre 2004.* JO, 2004.

[Ope05] OpenRDF.org. *User Guide for Sesame – Updated for Sesame release 1.2.3.* Aduna B.V., Sirma AI Ltd., 2005. `http://www.openrdf.org/doc/sesame/users/ch01.html`.

[Pan04] J.Z. Pan. Requirements for a Semantic Web Rule Language. *`http://www.w3.org/2004/12/rules-ws/paper/51/`*, 2004.

[Pat03] P. F. Patel-Schneider. From KL-ONE to OWL : Description Logics in the Ivory Tower and the Semantic Web. *tutorial présenté aux conférences ISWC'2003 et ENC'2004*, 2003. `http://www-db.research.bell-labs.com/user/pfps/talks/history/all.html`.

[Pdc05] FOAF Project et RDF developer community. *FOAF Vocabulary Specification – Namespace Document 27 July 2005.* 2005. `http://xmlns.com/foaf/0.1/`.

[Pec05a] P. Peccatte. *Métadonnées : une initiation.* Soft Expérience, 2005. `http://peccatte.karefil.com/Software/Metadata.htm`.

[Pec05b] P. Peccatte. *Windows NT/2000/XP et les fichiers Macintosh.* Soft Experience, 2005. `http://peccatte.karefil.com/software/MacNT.htm`.

[Pel03] C. Pelé. *Développement d'une bibliothèque de géométrie algorithmique pour la plateforme OXYGENE.* Mémoire de fin de stage de DEA Sciences de l'information géographique, Université de Marne la Vallée, 2003.

[Per95] J. Perrice. *Les Sytèmes d'Information Géographique.* DESS ID 1994-1995, module GED, 1995.

[Per02] G. Percivall. ISO 19119 and OGC Geographic Information Service Architecture. *présentation au XXII congrès international FIG (Fédération Internationale des Géomètres), Washington, D.C. USA, 19-26 avril*, 2002.

[Per04] H. Perez. *Découverte dynamique de Web Services à travers la mise en œuvre des technologies du Web Sémantique.* rapport de stage de DESS Génie des logiciels applicatifs, centre R&D d'EDF de Clamart – Université Paris VI, 2004. `http://www.orchaid.com/ressources/pdf/rapportstage.pdf`.

[Pia70] J. Piaget. *Psychologie et épistémologie – Pour une théorie de la connaissance.* bibliothèque Médiations, ed. Denoël – Gonthier, 1970.

[Pie00] L. Piechocki. *Modéliser avec UML.* 2000. `http://uml.free.fr/cours/i-p6.html`.

[Pil03] J-F. Pillou. *Bases de données – Modèles de SGBD.* 2003. `http://www.commentcamarche.net/bdd/bddtypes.php3`.

[Pom96] J. Pomian. *Mémoire d'entreprise : techniques et outils de la gestion du savoir.* Sapientae, 1996.

[RNK+04] A. Rector, N. Noy, H. Knublauch, G. Schreiber, et M. Musen. Ontology Design Patterns and Problems : Practical Ontology Engineering using Protege-OWL. *Tutorial at the Third International Semantic Web Conference (ISWC 2004)*, 2004. `http://www.cs.man.ac.uk/~rector/tutorials/iswc-tutorial-2004/ISWC-Tutorial-Best-Practice.pdf`.

[Rol99] F. Role. *Panorama des travaux en cours sur les métadonnées.* Rapport de recherche INRIA n° 3628, 1999.

[Rom01] M. Ben Romdhane. *Navigation dans un espace textuel – Accès à l'information scientifique.* Thèse de doctorat d'informatique de l'Université Jean Moulin Lyon 3, 2001.

[Ros75] E. Rosh. Human categorization. *Journal of Experimental Psychology, vol. 104, pp. 192-233*, 1975.

[Rou04] F. Rousseaux. *Étude des modèles numériques de terrain pour améliorer la fiabilité des calculs d'aléas et de vulnérabilité.* Thèse de doctorat en sciences de l'information géographique de l'Université de Marne-la-Vallée,, 2004.

[Rua98] A. Ruas. *First results on the OEEPE test on generalisation.* OEEPE Newsletter, vol.1, pp.5-10, 1998.

[Rua99] A. Ruas. *Modèle de généralisation de données géographiques à base de contraintes et d'autonomie.* Thèse de doctorat en Sciences de l'Information Géographique de l'Université de Marne-la-Vallée, p.16 et pp.73-77, 1999.

[SAA+00] G. Shreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde, et B. Wielinga. *Knowledge Engineering and Management, The CommonKADS Methodology.* MIT Press, p. 22 et 112, 2000.

[Sab02] N. Sabouret. *Étude de modèles de représentations, de requêtes et de raisonnement sur le fonctionnement des composants actifs pour l'interaction homme-machine.* Thèse de doctorat informatique de l'Université Paris-Sud, 2002.

[SB01] R. Schneeberger et R. Bolliger. *Élaboration d'un modèle de métadonnées pour la Suisse compatible avec la norme ISO 19115.* Office fédéral de topographie COSIG, 2001.

[Sch01] M. Scholl. *Bases de données géographiques.* In Bases de données et internet, Modèles, langages et système, A. Doucet et G. Jomier, collection Informatique et systèmes d'information, Hermes-Lavoisier, pp. 186-187, 2001.

[Sch02] M. Scholl. *Indexation spatiale.* cours de DEA SIR Paris VI, module BD, 2002. `http://cedric.cnam.fr/vertigo/Cours/DEA-P6/indexSpat.pdf`.

[Sed84] R. Sedgewick. *Algorithms*. Addison-Wesley Publishing Company, Inc. (USA), 552 p., *cité par [Con99]*, 1984.

[She05] D. Sheeren. *Méthodologie d'évaluation de la cohérence inter-représentations pour l'intégration de bases de données spatiales*. Thèse de doctorat informatique de l'Université Pierre et Marie Curie (Paris VI), LIP6, 2005.

[SOA03] SOAPuser.com. *Bases SOAP : Qu'est-ce que SOAP ?* 2003. `http://www.soapuser.com/fr/basics1.html`.

[SON+04] S. Stoutenburg, L. Obrst, D. Nichols, J. Peterson, et A. Johnson. Toward a Standard Rule Language for Semantic Integration of the DoD Enterprise. *W3C Workshop on Rule Languages for Interoperability*, 27-28 April 2005, Washington D.C., USA, 2004. `http://www.w3.org/2004/12/rules-ws/paper/28/`.

[SPAS03] K. Sycara, M. Paolucci, A. Ankolekar, et N. Srinivasan. Automated discovery, integration and composition of semantic web services. *The Semantic Web Journal, December 2003, pp. 1-28*, 2003.

[Sun04] Sun. *How to Write Doc Comments for the Javadoc Tool*. 2004. `http://java.sun.com/j2se/javadoc/writingdoccomments/`.

[SW01] B. Smith et C. Welty. Ontology : Towards a new synthesis. *In Actes de la 2ème conférence internationnale FOIS'01 (Formal Ontology in Information Systems), Ogunquit, Maine, USA*, 2001.

[Tao02] A. Taouss. *Mise en place d'un serveur de métadonnées géographiques*. rapport de stage de DEA SIG de l'Université de Marnce la Vallée, p.7, 2002.

[Tch02] P. Tchounikine. Pour une Ingénierie des Environnements Informatiques pour l'Apprentissage Humain. *Revue Information Interaction Intelligence, volume 2 (n°1), p. 59-93.*, 2002.

[Teu00] R. Teulier. L'ingénierie des connaissances et les organisations. Quels enjeux pour la recherche ? *IC'2000, conférence tutorielle*, 2000.

[tH05] J.H. ter Horst. Combining RDF and Part of OWL with Rules : Semantics, Decidability, Complexity. *In Actes de la conférence ISWC 2005, Galway, Ireland, pp. 668-684*, 2005. `http://www-kasm.nii.ac.jp/~i2k/tmp/iswc2005/papers/3729/37290668.pdf`.

[TI02] R. Troncy et A. Isaac. DOE : une mise en oeuvre d'une méthode de structuration différentielle pour les ontologies. *In Actes d'IC'2002, Journées francophones d'Ingénierie des Connaissances*, Rouen, pp. 63-74, 2002.

[Top04] TopicMaps.Org. *XML Topic Maps (XTM 1.0), Specification*. http ://www.topicmaps.org/xtm/1.0/xtm1-20010806.html, 2004.

[Tro04] R. Troncy. *Formalisation des connaissances documentaires et des connaissances conceptuelles à l'aide d'ontologies : application à la description de documents audiovisuels*. Thèse de doctorat d'informatique de l'Université Joseph Fourrier - Grenoble I, 2004.

[Tso02] M-H. Tsou. An Operational Metadata Framework for Searching, Indexing, and Retrieving Distributed Geographic Information Services on the Internet. *GIScience '02 : Proceedings of the Second International Conference on Geographic Information Science, Lecture Notes In Computer Science, Vol. 2478, ed. Springer-Verlag, Londres, p. 313-332*, 2002.

[Val04] R. Vallée. *SIG et SGBD : Etude sur l'interconnexion et l'interopérabilité de Oracle 9i et de PostgreSQL*. rapport de fin d'études de DESS OPSIE, Université Lyon II, 2004.

[van01] E. van der Vlist. *Les experts XSLT partent en guerre contre xsl:script*. 2001. http://xmlfr.org/actualites/tech/010301-0001.

[Vog04] Vogele. Enhancing SDI with SW technologies. *In Actes de la conférence AGILE 2004*, Heraklion, Crête, Grêce, 2004.

[W3C99a] W3C. *Resource Description Framework (RDF) Model and Syntax Specification*. 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[W3C99b] W3C. *XML Path Language (XPath) Version 1.0*. W3C Recommendation 16 November 1999, 1999. http://www.w3.org/TR/xpath.

[W3C99c] W3C. *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation 16 November 1999, 1999. http://www.w3.org/TR/xslt.

[W3C01] W3C. *Web Services Description Language (WSDL) 1.1*. W3C Note 15 March 2001, 2001. http://www.w3.org/TR/wsdl.

[W3C03a] W3C. *LBase : Semantics for Languages of the Semantic Web*. W3C Working Group Note 10 October 2003, 2003. http://www.w3.org/TR/2003/NOTE-lbase-20031010/.

[W3C03b] W3C. *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*. W3C Recommendation 21 October 2003, 2003. http://www.w3.org/TR/2003/REC-MathML2-20031021/.

[W3C03c] W3C. *SOAP Version 1.2 Part 1 : Messaging Framework*. W3C Recommendation 24 June 2003, 2003. http://www.w3.org/TR/soap12-part1/.

[W3C04a] W3C. *Namespaces in XML 1.1 W3C – Recommendation 4 February 2004*. 2004. http://www.w3.org/TR/2004/REC-xml-names11-20040204/.

[W3C04b] W3C. *OWL Web Ontology Langage Use Cases and Requirements*. 2004. http://www.w3.org/TR/webont-req/, traduction française de J.J.Solari datant du 4 mai 2004 : http://www.yoyodesign.org/doc/w3c/webont-req-20040210/.

[W3C04c] W3C. *OWL Web Ontology Language Guide*. 2004. http://www.w3.org/TR/owl-guide/.

[W3C04d] W3C. *OWL Web Ontology Language Semantics and Abstract Syntax*. 2004. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/.

[W3C04e] W3C. *RDF Vocabulary Description Language 1.0 : RDF Schema – W3C Recommendation 10 February 2004*. 2004. http://www.w3.org/TR/rdf-schema/.

[W3C04f] W3C. *XML Schema Part 0 : Primer Second Edition*. W3C Recommendation 28 October 2004, 2004. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

[ZCG+03] G. Zerbib, L. Charbit, J. Gahide, X. Galbois, P. Crescenzo, M. Gautero, et P. Lahire. *JavInspector – Travail d'étude et de recherche*. Maîtrise d'informatique de l'Université de Nice Sophia Antipolis, 2003. http://www.i3s.unice.fr/~crescenz/publications/javinspector-minfo-rapport-2003-06.pdf.

[ZY00] K. Zeitouni et L. Yeh. Le Data Mining Spatial et les bases de données spatiales. *In Actes des Journées Data Mining Spatial et Analyse du Risque, Versailles*, 2000.

Publications

Y. Abd-el-Kader, Indexation sémantique des API du domaine géographique, apports du langage OWL, *actes de l'atelier MetSI de la conférence INFORSID 2005*, pp. 9-18, Grenoble, 24 Mai, 2005.

Y. Abd-el-Kader, Conception et exploitation d'une base de métadonnées de traitements géographiques – Description des connaissances d'utilisation, *IC'2005, 16èmes Journées francophones d'ingénierie des connaissances*, pp. 1-12, Nice, 30 Mai-3 Juin, 2005.

Y. Abd-el-Kader, Cataloguing Geographical Data Processing Tools, Conception and Exploitation of a Metadata Model, *ICC 2005 (International Cartographic Conference)*, La Corogne, Espagne, 9-16 Juillet, 2005.

Y. Abd-el-Kader, Catalogage de traitements informatiques du domaine géographique, conception et exploitation d'un modèle de métadonnées, *Le Monde des Cartes – Revue du Comité Français de Cartographie (CFC)*, n°186 consacré à la conférence ICC 2005, décembre 2005, pp. 22-29.

Y. Abd-el-Kader et B. Bucher, Cataloguing GI functions and software resources within IGN, *actes de la 9ème conférence internationale AGILE (article publié en appendum des actes)*, Visegrad, Hongrie, 2006.

Résumé

L'information géographique est construite, analysée et transformée par des traitements informatiques. À l'Institut Géographique National (IGN), les utilisateurs et les développeurs ont besoin d'aide pour rechercher, connaître et partager ces traitements.

Le but de notre travail est de fournir cette aide. Les documentations existantes ne permettent pas toujours de répondre de façon satisfaisante aux besoins identifiés : elles sont éparses, aux formats hétérogènes et ne décrivent pas les données avec toute la finesse souhaitée. Ces documentations sont également en général statiques : elles ne peuvent fournir des modes d'emploi adaptés aux contextes d'utilisation particuliers (caractéristiques des données, environnement, connaissances de l'utilisateur). Or, puisque toutes les réponses aux requêtes des utilisateurs ne peuvent être stockées à l'avance, il faut que des mécanismes de dérivation de l'information soient mis en œuvre.

Face à ce problème, nous soutenons la thèse qu'une solution peut être de recourir à des métadonnées à la structure et au contenu contrôlés, conformes à un modèle à la fois approprié à la spécificité des traitements géographiques (description fine des propriétés des données avant et après traitements, illustrations) et propre à une représentation opérationnelle des connaissances d'expert. Nous montrons l'intérêt de suivre une double approche en développant d'une part un *système d'information documentaire (SI)* dédié à la consultation et la saisie des métadonnées, d'autre part un *système à base de connaissances (SBC)* dédié à la simulation des raisonnements de l'expert et reposant sur les langages standard du Web Sémantique RDF, OWL et SWRL.

Mots-clés : métadonnées des traitements géographiques, connaissances expert, adaptation des modes d'emploi au contexte.

Abstract

Geographical information is built, analyzed, transformed by computing programs. At the French National Mapping Agency (IGN), developers and users need assistance to seek, know and share these computing programs.

The goal of our work is to provide this help, knowing that existing documentations are scattered, have heterogeneous formats and do not describe the data with all the desired smoothness. These documentations are also static : they cannot provide instructions adapted to the context of use (characteristics of the data, environment, user knowledge). However, since all the answers to the requests of the users cannot be stored in advance, it is necessary to implement mechanisms for derivate information.

Facing this problem, we propose to exploit metadata with controlled structure and contents, in conformity with a model at the same time appropriate to the specificity of the geographical computing programs (precise description of the data's properties before and after processing, illustrations) and designed for operational knowledge representation. We show the interest to follow a double approach. On one hand, we build a *documentary Information System (IS)* for metadata consultation and acquisition ; on the other hand, we build a *Knowledge-Based System (KBS)* dedicated to the simulation of the expert reasoning. The KBS is based on the standard languages of Semantic Web RDF, OWL and SWRL.

Keywords : metadata for geographical processing tools, expert knowledge, adaptation of instructions for use.