MASTER EN SCIENCES INFORMATIQUES

Recherche de services bioinformatiques dans une ontologie investigation de la relation part-whole

Schuurman, Amandine

Award date:
2005

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique

Recherche de services bioinformatiques dans une ontologie

Investigation de la relation part-whole

Amandine Schuurman

Mémoire présenté en vue de l'obtention
du grade de Maître en Informatique

Année Académique 2004-2005

Résumé

La bioinformatique regroupe l'ensemble des programmes informatiques appliqués à la biologie. Le biologiste, qui fait appel à ces nombreux outils, se limite principalement aux programmes qu'il connaît, même s'ils ne sont pas le choix optimal par rapport à l'analyse qu'il doit mener.

Constatant cela, nous souhaitons aider cet utilisateur en intégrant une partie de l'expertise qui lui était jusqu'ici requise. La description actuelle des services est insuffisante et nous allons donc travailler sur base d'une ontologie qui va l'enrichir.

Bigre est un projet qui vise à introduire un système distribué dans le cadre de la bioinformatique, il est co-développé par les FUNDP et l'ULB. Ce mémoire s'inscrit dans le cadre de ce projet. Nous tentons de mettre en évidence l'aspect *recherche de services* sur base de l'ontologie proposée dans Bigre. Nous proposons ensuite d'enrichir cette dernière de relations sémantiquement plus riches, les relations *part-whole*.

Mots-clé : ontologie, inférence, logique descriptive, relation part-whole, représentation de la connaissance.

Abstract

Bioinformatics brings together all the data-processing programs applied to biology. The biologist, who makes use of these many tools, limits himself mainly to the programs he knows, even if they are not the best choice in respect of the analysis that he needs to carry out.

Having noted this, we wish to help this user by integrating a part of the expertise that was up until now required of him. The current description of the services is insufficient and we shall therefore work on the basis of an ontology that will enhance it.

Bigre is a project the aim of which is to introduce a distributed system within the framework of bioinformatics ; it has been jointly developed by the University of Namur and the University of Brussels. This thesis comes within the scope of this project. We try to highlight the research aspect of services on the basis of the ontology proposed in Bigre. We then propose to improve this ontology with semantically richer relations, the part-whole relations.

Keywords : ontology, inference, description logic, part-whole relation, knowledge representation.

Je tiens à remercier mon promoteur, le Professeur Englebert, pour ses bonnes idées et ses conseils avisés, pour sa patience et sa gentillesse durant mon stage et pendant l'élaboration de ce mémoire.

Je remercie l'équipe Bigre, et en particulier Pierre et Quentin pour toutes leurs explications, surtout concernant l'ontologie de Bigre.

Merci à Marc Colet pour son accueil à l'IBMM, ainsi qu'à Olivier, Joseph, et les autres. Je remercie Peter Rice de l'EBI pour mes quelques semaines de stage passées là-bas. Merci à Valérie et Morgane pour m'avoir relue sur la partie bioinformatique.

Merci à Philippe et Anthony pour leurs relectures et corrections.

Je remercie ma famille et mes amis pour leur soutien et leurs encouragements.

Table des matières

Table des figures

Liste des tableaux

Introduction

Les biologistes utilisent les ordinateurs pour étudier des problèmes qui leurs sont spécifiques. Ces tâches spécialisées forment ce que nous appelons la bioinformatique, que nous étudions dans le premier chapitre : comparer des séquences, modéliser les propriétés d'un système biologique, visualiser des structures tridimensionnelles, procéder à des simulations, etc. Toutes ces actions sont exécutées grâce à des programmes, ce sont les services bioinformatiques. Ils sont nombreux et disponibles sur Internet, de même que les banques de données qui regroupent un grand nombre d'informations.

Les services sont proposés par différents fournisseurs et sont présentés sur les pages Web de chacun. Il y a donc une forte hétérogénéité, tant au niveau de leur présentation qu'au niveau des services eux-mêmes (chaque fournisseur peut proposer sa propre classification des services). Pour cela et pour d'autres raisons, telles que la sécurité par exemple, les biologistes se limiteront dans les programmes utilisés. Les avantages procurés par un système fédéré, qui est un cas particulier de système distribué où les différents agents collaborent, sont nombreux : sécurité, espace de travail, présentation unique et cohérente, classification, etc. Nous allons nous intéresser à de tels systèmes, et en particulier à Bigre, présenté dans le deuxième chapitre.

Il existe, parmi le large champ de la bioinformatique, différents profils d'utilisateurs, allant du plus spécialisé au novice. Le biologiste, qu'il soit un utilisateur normal ou occasionnel, fait appel aux différents outils disponibles. Mais il va se limiter principalement à ceux qu'il connaît, soit parce qu'il les a déjà utilisés, soit parce qu'il en a entendu parler lorsqu'il a pris le temps de se renseigner sur ceux-ci.

Chaque service présente une description textuelle qui indique la ou les fonctionnalités qu'il remplit. Si l'utilisateur souhaite découvrir de nouveaux services, il va utiliser un moteur de recherche dans lequel il indiquera un ou plusieurs mots-clé. Le moteur va donc lui renvoyer en réponse tous les services qui comprennent le mot-clé introduit dans leur description textuelle : ce n'est pas une solution optimale.

Voilà pourquoi l'utilisation d'une ontologie sera un plus pour la présentation des services et des données manipulées par ces services. Une ontologie va permettre d'en

indiquer de manière plus pointue les caractéristiques, elle va au delà de la simple description textuelle. Nous présentons les ontologies dans le chapitre 4. Mais avant cela, le chapitre 3 introduira les bases du Web sémantique et du langage d'ontologie OWL.

De plus, lorsque l'utilisateur a un nouveau besoin en termes de programmes bioinformatiques, cela lui demandera du temps et un certain niveau d'expertise pour trouver ce qui lui convient, car aucun outil ne l'aide dans cette tâche. De ce fait, s'il est un utilisateur occasionnel, il ne souhaitera pas découvrir de nouveaux services, même si ces derniers seraient plus appropriés par rapport à l'analyse qu'il doit mener. Nous nous intéressons à la problématique de la recherche de services, et nous voyons comment fournir une liste de services intéressante sur base de l'ontologie étudiée, dans le chapitre 5.

Dans les deux derniers chapitres, nous allons encore plus loin dans le détail qui pourrait être apporté à l'ontologie, par l'utilisation des relations *part-whole* que nous allons étudier en profondeur. En effet, elles permettront d'enrichir la représentation de la connaissance, et d'affiner un peu plus le résultat à des recherches de services.

Chapitre 1

La bioinformatique

Dans ce chapitre, nous allons commencer par définir la bioinformatique (section 1.1). Afin de bien en comprendre l'utilité (section 1.3, [Gib01] et [CN03]), nous devrons acquérir des notions de biologie moléculaire (section 1.2, [Eti95]). Nous nous limiterons à la biologie moléculaire, bien que la bioinformatique s'intéresse également à d'autres branches telles que la biologie cellulaire, ou encore la biologie structurale. Nous définirons ensuite les banques de données et services bioinformatiques (sections 1.3.2, [Nod]). Enfin, nous nous intéresserons aux différents profils d'utilisateurs (section 1.4).

1.1 Qu'est-ce que la bioinformatique ?

Les biologistes utilisent les ordinateurs pour étudier des problèmes qui leur sont spécifiques. Ces tâches spécialisées forment, dans leur ensemble, le champ de la bioinformatique. Nous pouvons définir la bioinformatique comme étant la branche informatique de la biologie moléculaire.

Avant l'ère bioinformatique, seules deux façons de procéder à des expériences biologiques étaient disponibles : *in vivo* (dans un organisme vivant) et *in vitro* (dans un environnement artificiel). Par analogie, la bioinformatique est en fait la biologie *in silico*.

1.2 Notions de biologie moléculaire

1.2.1 Les protéines

Les protéines sont les constituants essentiels de toute cellule vivante. Leurs rôles au sein des organismes vivants sont multiples : catalyse enzymatique, stockage et transport (fer, oxygène, etc.), transmission des influx nerveux, anticorps, etc. Dès 1925,

les chimistes avaient mis en évidence que les protéines étaient constituées de longues chaînes formées par assemblage d'éléments plus simples : les *acides aminés*. Mais ce n'est qu'en 1951 qu'un biochimiste britannique, F. Sanger déterminera pour la première fois la séquence complète d'acides aminés d'une protéine : l'insuline[1]. Ce travail lui vaudra d'ailleurs l'attribution du prix Nobel de chimie.

Les acides aminés sont des molécules organiques complexes. Elles portent[2] à la fois un groupement acide organique et un groupement amine[3], basique[4]. Un atome de carbone peut participer à quatre liaisons dont deux sont définies par les groupements acides et amines. Il reste donc deux liaisons disponibles. Parmi la multitude de possibilités pour les deux groupements restants, la nature a effectué un tri sévère, puisque le troisième substituant est systématiquement un hydrogène, tandis que le quatrième est choisi dans une liste de vingt groupements seulement. On ne rencontre donc que 20 acides aminés dits « naturels », parmi lesquels on peut citer l'alanine, la glycine, la tyrosine, la glutamine, etc. La figure 1.1 illustre la forme générale d'un acide aminé.

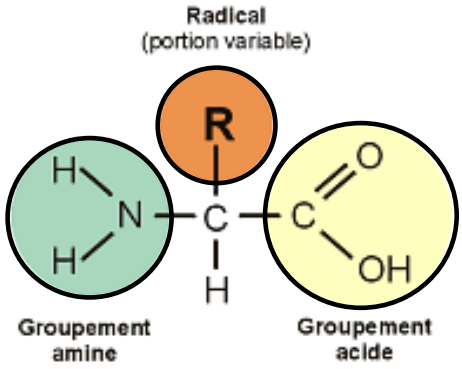

FIG. 1.1: Un acide aminé : atome de carbone avec groupement amine, groupement acide, hydrogène et radical R (20 groupements R possibles).
Source : `http://ici.cegep-ste-foy.qc.ca/profs/gbourbonnais/pascal/fya/chimcell/notesmolecules/imagesmolecules/acidesamine2.gif`

[1]Hormone sécrétée par le pancréas lors des apports de sucre. Elle diminue le taux de sucre dans le sang et permet son utilisation par les organes. L'insuline sert au traitement de certains types de diabète.

[2]On dit d'une molécule qu'elle *porte*, c'est-à-dire comporte, un certain nombre de groupements.

[3]Une amine est une molécule dérivée de l'ammoniac dont certains hydrogènes ont été remplacés par un groupement carboné

[4]Une base est une molécule chimique capable de céder une paire d'électrons. Elle est généralement définie par ses réactions avec un autre type de composé chimique complémentaire, les acides.

Des codes à une ou trois lettres ont été établis pour chacun de ces 20 acides aminés, ce sont les codes IUPAC[5], du nom du comité qui les ont conçus. Ils permettent une écriture plus facile des *chaînes protéiques*. La chaîne (ou séquence) protéique est donc une *représentation* de la protéine réelle.

Les acides aminés peuvent s'associer les uns aux autres, le groupement acide d'un acide aminé peut réagir avec le groupement amine d'un autre, avec élimination d'une molécule d'eau pour former une liaison covalente nommée liaison peptidique. La figure 1.2 représente la formation de la chaîne protéique, appelée aussi *chaîne peptidique* qui est ainsi constituée d'un squelette formé d'atomes de carbone asymétriques (porteurs de quatre substituants différents) reliés par des liaisons peptidiques. Ce squelette porte des groupements correspondants à la chaîne latérale de l'acide aminé d'origine sur chacun des carbones asymétriques.

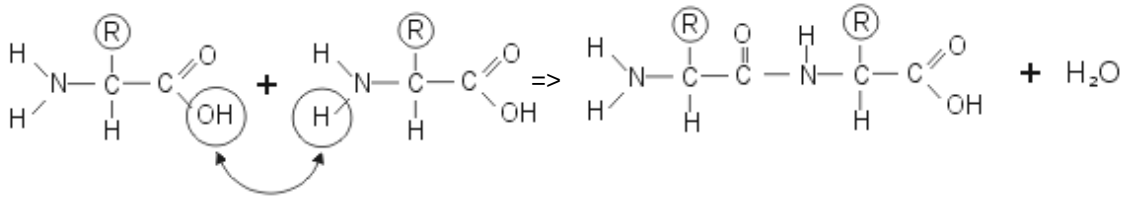

FIG. 1.2: Liaison peptidique par élimination d'une molécule d'eau.
Source : `http://ici.cegep-ste-foy.qc.ca/profs/gbourbonnais/pascal/fya/chimcell/notesmolecules/imagesmolecules/acidesamines.gif`

Les protéines sont donc de grosses molécules constituées d'une ou plusieurs chaînes peptidiques. La plupart des protéines sont formées de l'union de 100 à 200 acides aminés. La formule globale d'une protéine peut donc ressembler à $C_{1200}H_{2400}O_{600}N_{300}S_{100}$. Un type donné de protéine contient toujours un nombre total précis d'acides aminés, appelés les résidus, dans des proportions identiques. La formule de la protéine d'insuline se présente donc comme [30 glycine + 44 alanine + 5 tyrosine + 14 glutamine + ...]

Toutefois, l'identité réelle d'une protéine n'est pas seulement dérivée de sa composition mais aussi de l'ordre précis de ses acides aminés constitutifs. Ainsi, c'est de la séquence des 110 résidus de l'insuline humaine que découlent toutes ses propriétés biologiques. La séquence (ou l'ordre) des acides aminés de la protéine définit ce qu'on appelle sa **structure primaire**.

Les propriétés de la protéine, par exemple sa capacité à digérer le sucre ou à devenir une partie d'une fibre musculaire, ne résultent pas directement de sa structure primaire, mais plutôt de la forme tridimensionnelle que le ruban d'acides aminés adopte dans

[5]International Union of Pure and Applied Chemistry ou Union Internationale de Chimie Pure et Appliquée

son environnement. Cette forme est dictée par le type des acides aminés de la séquence et des interactions entre les groupements dont ils sont porteurs : leur caractère hydrophile (qui aime l'eau) ou hydrophobe, ou encore leur éventuelle charge électrique va amener les groupements à s'associer, à se rassembler en certaines régions, provoquant ainsi le repliement de la chaîne. Une protéine fonctionnelle peut être constituée de plusieurs chaînes peptidiques. La figure 1.3 montre les quatre grands types d'interactions intervenant dans le repliement de la chaîne.

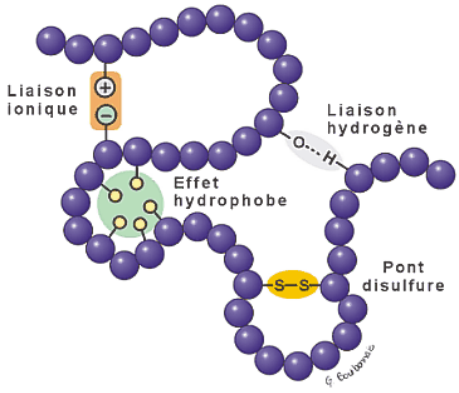

FIG. 1.3: Quatre grands types d'interactions interviennent dans le repliement de la chaîne.
Source : `http://ici.cegep-ste-foy.qc.ca/profs/gbourbonnais/pascal/fya/chimcell/notesmolecules/imagesmolecules/protrepli.gif`

Ces interactions font apparaître sur divers segments de la chaîne peptidique une structure régulière appelée **structure secondaire**. On reconnaît deux grands types de structure secondaire :

- **l'hélice alpha** : dans la structure dite en hélice alpha, la chaîne d'acides aminés prend la forme d'un tire-bouchon. Les différentes spires sont stabilisées par des liaisons hydrogène ;
- **le feuillet bêta** : dans un feuillet bêta, il se forme des liaisons hydrogène entre certains segments de la chaîne disposés parallèlement les uns par rapport aux autres. L'ensemble forme comme une membrane plissée ;

Une protéine est donc faite d'hélices alpha et de feuillets bêta reliés par des segments qui n'ont pas de structure secondaire définie. La forme finale de la chaîne d'acides aminés, c'est à dire la structure tridimensionnelle finale qu'adopte la chaîne d'acides aminés, constitue la **structure tertiaire** de la protéine. La figure 1.4 représente une protéine dans sa forme finale, formée d'hélices alpha et de feuillets bêta reliés par des segments sans structure définie.

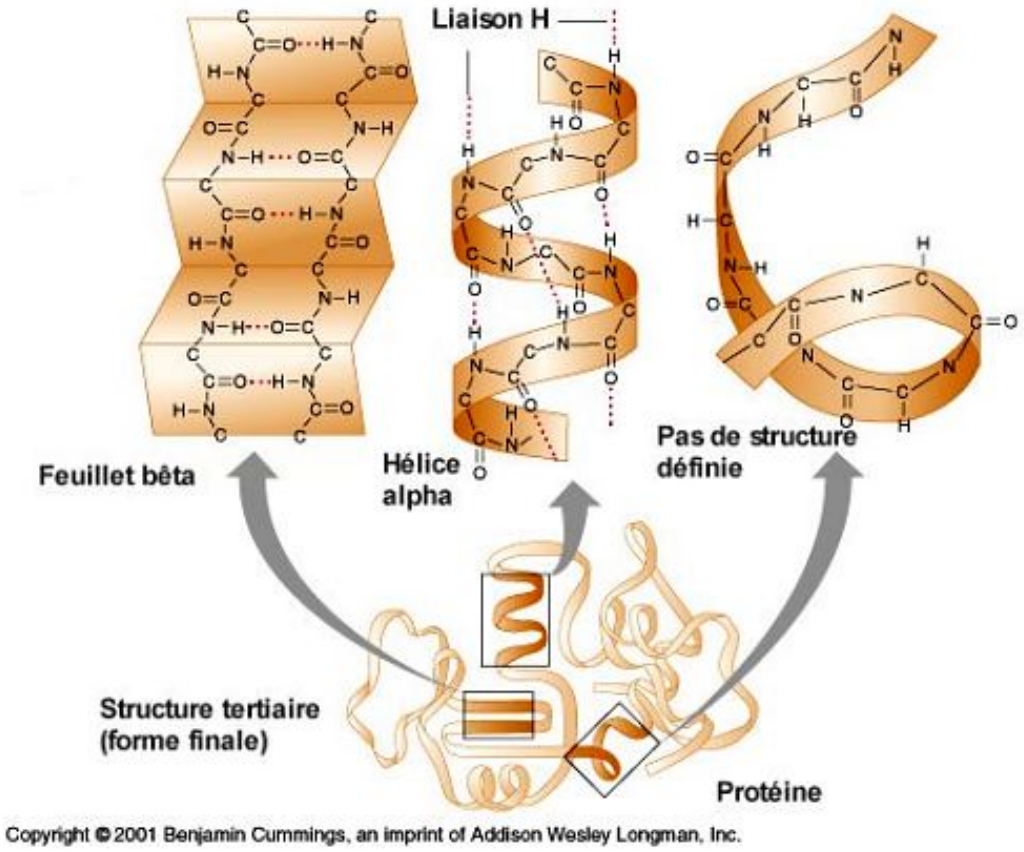

FIG. 1.4: Protéine dans sa forme finale, composée d'hélices alpha et de feuillets bêta

1.2.2 L'ADN

L'ADN (acide désoxyribonucléique) est le support de l'hérédité, car il a la faculté de se dupliquer et d'être transmis aux descendants lors des processus de reproduction des organismes vivants. Il est à la base de processus biologiques importants aboutissant à la production des protéines.

Complémentarité

L'ADN est une macromolécule formée par deux chaînes complémentaires qui s'emboîtent tout en s'enroulant l'une autour de l'autre pour former une double hélice. Chaque chaîne est constituée d'un squelette formé de phosphodiesters et de sucres (le désoxyribose) en alternance (P et D sur la figure 1.5). Chaque sucre D porte un groupe chimique appelé *base azotée*, ce sont l'adénine A, la thymine T, la guanine G et la cytosine C, comme illustré à la figure 1.5. La complémentarité des deux chaînes vient du fait que A-T et G-C peuvent s'apparier entre elles (former des paires), on parle alors de *paires de bases*. L'information génétique est codée par la succession des bases azotées A, T, G et C.

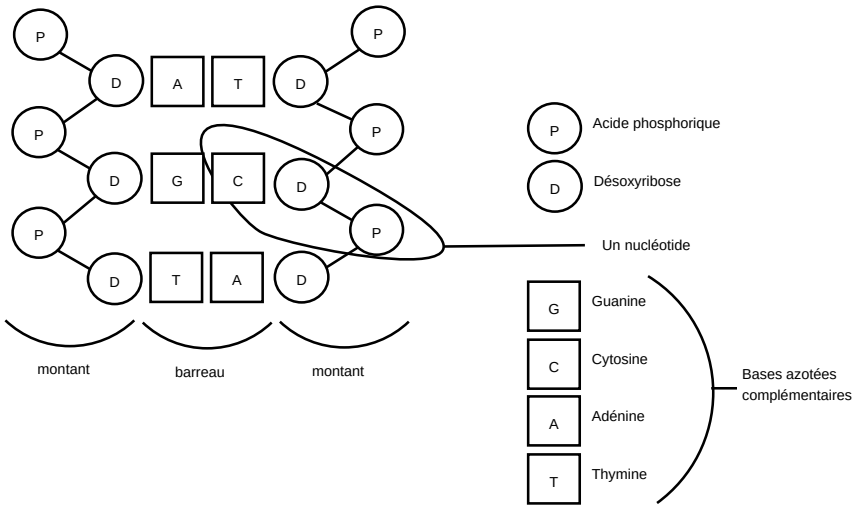

Fig. 1.5: Structure de l'ADN

Grâce à la complémentarité des chaînes, chaque brin de l'hélice d'ADN va, en se séparant de l'autre, servir de gabarit à la machinerie biologique qui va recréer (synthétiser) le brin complémentaire. En partant d'une hélice composé de deux brins d'ADN, on obtient deux hélices identiques, formées chacune d'un brin d'origine et d'un brin nouvellement synthétisé, c'est la **réplication** de l'ADN, illustrée à la figure 1.6. C'est l'ADN polymerase qui va ouvrir l'hélice d'ADN (désolidariser les deux brins), une fourche de

réplication se forme pour donner deux brins distincts qui vont, par le biais de la complémentarité, édifier deux nouvelles molécules d'ADN. Chaque molécule d'ADN fille hérite d'un brin de l'ADN mère, la réplication est dite semi-conservatrice.

La synthèse des protéines

Les fragments d'ADN qui contiennent l'information nécessaire à la production des protéines sont appelés des gènes. Ils sont formés d'une région codant pour la protéine et d'une région régulatrice qui contrôle l'expression de la partie codante du gène.

Le processus de production s'appelle la synthèse des protéines et se déroule en trois étapes : la **transcription** de l'ADN en ARN simple brin, la **maturation** de cet ARN en ARN messager et enfin, la **traduction** de l'ARN messager en protéine. Ces étapes sont représentées à la figure 1.7.

L'ARN (acide rybonucléique) est une copie de l'ADN, à quelques différences près :
- Le sucre désoxyribose est remplacé par un ribose.
- La base thymine T est remplacée par un uracile U car il est moins coûteux à produire pour les organismes vivants, et comme beaucoup d'ARN est produit par la cellule mais n'est pas conservé, c'est le coût énergétique qui prime sur la stabilité de l'information (l'ADN).
- Il n'est généralement qu'un simple brin, alors que l'ADN a une structure en double hélice.
- Il est court : de 50 à 5000 nucléotides (des millions pour l'ADN).

Lors de la **transcription** d'un gène, l'ADN va se dérouler pour se séparer en deux brins, afin de créer un brin complémentaire d'ARN. Ensuite, l'ADN se réassemble et l'ARN est libéré de la chaîne d'ADN.

Lors de la **maturation** de l'ARN, celui-ci subit des transformations dans le noyau avant de devenir l'ARN messager qui va être exporté pour la traduction. Une opération importante de la maturation est l'épissage de l'ARN : il est composé d'exons, ce sont les parties codantes qui seront traduites en protéine, et d'introns qui ne codent pas pour une protéine et sont et sont excisés (enlevés) lors de l'épissage. Les exons sont ligaturés entre eux pour former l'ARN messager mature qui code pour la protéine complète.
Ceci permet un phénomène très intéressant qui est l'épissage alternatif : selon les protéines réalisant l'épissage, on peut avoir différents épissages de l'ARN en choisissant parmi plusieurs exons et obtenir une protéine dont une région peut être de plusieurs sortes différentes. Un gène permet ainsi de coder pour un plus grand nombre de protéines différentes[6].

La **traduction** de l'ARN messager en protéine se fait dans le cytoplasme. L'ARN se fixe à un ribosome qui va assembler une séquence d'acides aminés selon les instructions du code génétique.

[6]Source : `http://www.bio.espci.fr/scolarite/c_BIO/mol/mol3.htm`

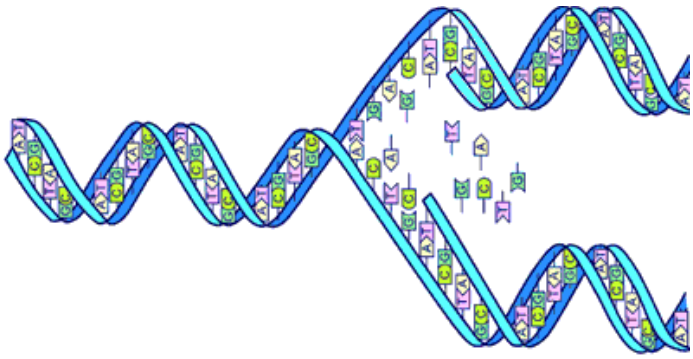

FIG. 1.6: La réplication
Source : `http://fr.wikipedia.org/wiki/Image:Dna-split.png`

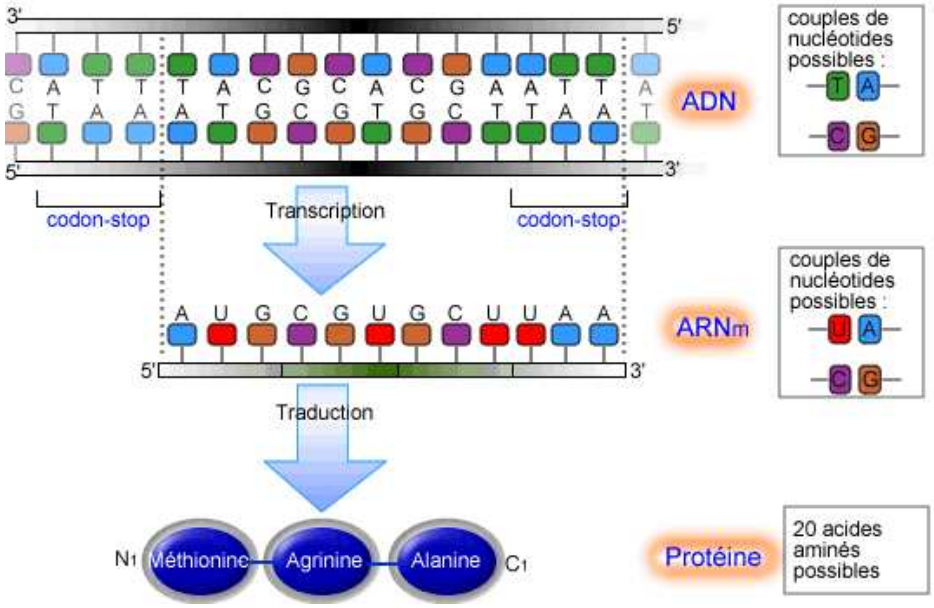

FIG. 1.7: Transcription et Traduction
Source : `http://fr.wikipedia.org/wiki/Image:Prot%C3%A9ines.png`

1.3 Utilité de la bioinformatique

L'information héréditaire et fonctionnelle d'un organisme est contenue dans ses macro-molécules d'ADN, d'ARN et de protéines. Elles sont construites à partir d'un alphabet bien connu (*A-T-C-G* pour l'ADN, *A-U-C-G* pour l'ARN et les 20 acides aminés pour les protéines), elles peuvent donc être représentées par des séquences de symboles. Il est possible de comparer ces séquences afin de trouver des similarités qui suggèrent que les molécules sont apparentées, de par leur forme ou leur fonction. La comparaison est l'un des outils informatiques les plus utiles pour les biologistes moléculaires. Par exemple, le *Blast* est un algorithme qui compare une séquence à la totalité des séquences d'une banque de données publique (les bases de données sont présentées à la section suivante), la consultation de celle-ci étant rendue possible grâce à l'Internet (voir chapitre suivant).

La bioinformatique va au-delà de la création et la maintenance de bases de données, les bioinformaticiens s'inspirent de différents domaines (les statistiques, la physique, l'informatique et l'ingénierie) afin de permettre le traitement des données biologiques à tous les niveaux : que ce soient les données issues de l'analyse d'une séquence ou de la structure d'une protéine ; la modélisation des métabolismes et l'analyse quantitative des populations et de l'écologie ; ou encore l'abstraction des propriétés d'un système biologique en un modèle mathématique ou physique par exemple.

L'ordinateur permet également l'utilisation d'outils de visualisation 3D des protéines. En effet, la fonction de la protéine au sein d'un organisme est la conséquence directe de sa structure tridimensionnelle. Le lien logique entre structure et fonction est un concept central de la biologie moléculaire et il est évidemment bien plus aisé de manipuler des modèles de structure sur un écran d'ordinateur que de procéder aux tests « à la main » (imaginez devoir jouer avec des images de protéines, telles un puzzle). Une proportion croissante de la bioinformatique est donc dédiée au développement d'outils de navigation entre séquences d'acides aminés et structures 3D.

En résumé, nous pouvons citer quelques domaines dans lesquels l'informatique joue un rôle de premier plan :

- utilisation des bases de données publiques ;
- recherche et alignement de séquences ;
- prédiction de gènes ;
- analyse de séquences protéiques ;
- prédiction de structures protéiques ;
- analyse des propriétés structurales d'une protéine ;
- simulation biochimique ;
- etc.

1.3.1 Les services

Dans les domaines cités, on fait appel aux outils informatiques, que nous appelons services bioinformatiques, ou encore les applications. L'algorithme *Blast*, présenté plus haut, est un exemple de service.

EMBOSS (*European Molecular Biology Open Software Suite*) est une suite complète de services bioinformatiques pour l'analyse de séquences(plus de 150 outils dans sa version de base). Son développement est mené par Peter Rice et Alan Bleasby. Les outils de cette suite ont été classés par catégories. Par exemple, parmi tous les services d'alignement, certains sont dédiés à l'alignement multiple, à l'alignement global ou local, etc. Il faut également des outils d'édition, d'affichage, d'information, de création et d'indexation de bases de données, etc.

1.3.2 Les banques de données

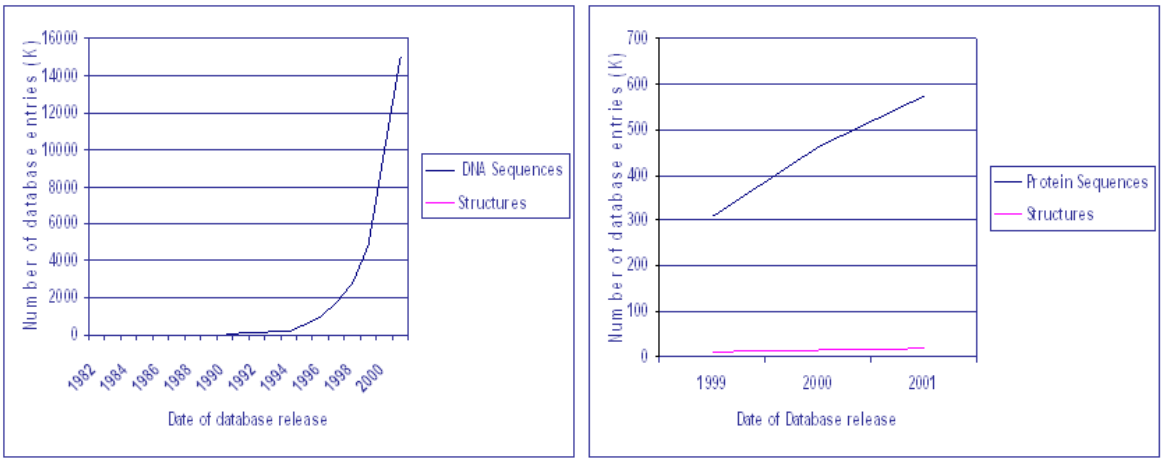

FIG. 1.8: Croissance exponentielle des bases de données nucléiques et linéaire des bases de données protéiques. Source : [Nod]

Il existe différentes banques (ou bases) de données, chacune stocke un certain type d'informations. Il y a les bases de données nucléiques, protéiques, ou composites.

- Les trois grandes bases de données nucléiques sont *EMBL* (*European Bioinformatics Institute* - UK, Europe), *GENBANK* (*USA*) et *DDJB* (*National Institute of Genetics*, Japon).
- Les bases de données protéiques sont *UNIPROT/SWISSPROT*, *UNIPROT/TREMBL*, *GENPEPT* (USA).
- *RefSeq* (*The Reference Sequence*) est une base de données composite qui vise à fournir un ensemble de séquences compréhensible et non redondant de séquences d'ADN et d'ARN et de séquences protéiques.

La croissance des banques de données de séquences nucléiques a été exponentielle ces dernières années, comme illustré à la figure 1.8. Cela est du, entre autre, aux différents projets de séquençage de génomes complets. La croissance des bases de données protéiques a quant à elle été linéaire.

Vu la richesse d'information contenue dans ces différentes banques de données, la première nécessité pour le biologiste est d'apprendre à utiliser des outils de recherche d'informations disponibles sur Internet, et notamment ceux dédiés à la recherche bibliographique. En effet, il existe des bases de données qui permettent la recherche dans des dizaines de journaux scientifiques.

1.4 Profil des utilisateurs

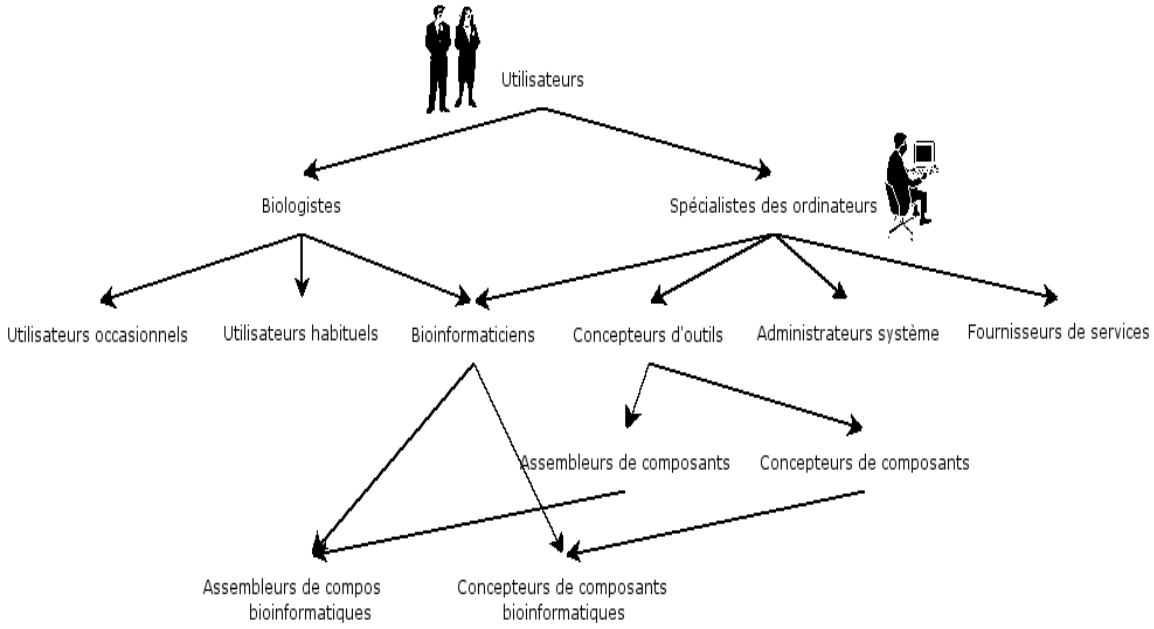

Fig. 1.9: Classification des utilisateurs

MyGrid [Ric] est une solution middleware qui exploite la technologie Grid[7]. L'équipe de MyGrid a étudié le profil de ses utilisateurs directs et indirects, et en propose une classification (figure 1.9). Cette étude a été menée car le développement de solutions telles que MyGrid ou Bigre (voir chapitre suivant) nécessite de collecter les exigences

[7]Il s'agit d'une infrastructure qui permet la collaboration de ressources. L'informatique distribuée est présentée au chapitre suivant.

par rapport à la façon dont les bioinformaticiens travaillent, afin de connaître la manière dont les différents services seront utilisés.

Comme illustré à la figure 1.9, le public d'utilisateurs est large. Les différents groupes d'utilisateurs comprennent les administrateurs système, qui peuvent installer et lancer un serveur d'applications ; les différents types de biologistes, qui peuvent utiliser des outils qui tournent sur un serveur ; et les concepteurs d'outils de différents types, qui peuvent utiliser des outils déjà existants dans leurs tâches de programmation.

1.5 Conclusion

La bioinformatique génère des outils mais cela n'est pas une fin en soi, l'objectif essentiel reste d'aider à découvrir comment le vivant fonctionne. L'accès aux différents services, ainsi qu'aux banques de données, se fait grâce à Internet. Nous abordons ce sujet dans le chapitre suivant.

Dans ce chapitre, nous avons étudié les notions de biologie moléculaire et nous avons compris ce qu'est la bioinformatique. Nous avons mis en évidence quelle en est l'utilité et nous avons vu qu'il existe de nombreux services ainsi que de très grandes banques de données.

De plus, nous avons appris qu'il existe différents profils d'utilisateurs de la bioinformatique. Parmi eux se trouvent les utilisateurs habituels et occasionnels. Ces derniers ne savent pas toujours quel service conviendra le mieux à ce dont ils ont besoin par rapport à l'analyse qu'il est en train de mener. Nous souhaitons aider l'utilisateur dans sa recherche, sur base d'un catalogue de services disponibles.

Chapitre 2

Les nouveaux besoins en bioinformatique

Ce chapitre présente la manière dont sont utilisés les outils bioinformatiques, en termes d'informatique distribuée, et l'intérêt qu'un système distribué présenterait pour répondre à certains manques. Nous étudierons le projet Bigre, qui s'inscrit dans cette voie.

2.1 La situation actuelle

L'utilisation d'outils bioinformatique se fait très peu en local, à savoir sur un seul ordinateur. L'utilisateur accède à des outils de calcul principalement via Internet. Il préférera les sites des laboratoires très connus. Ceux-ci sont fortement sollicités, les requêtes s'en trouvent doncoui ralenties. Il existe toutefois des sites miroirs qui sont présents pour désengorger le trafic et améliorer ainsi la disponibilité des outils, mais l'utilisateur ne les connaît pas ou bien il n'en retient pas l'adresse.

La manière de procéder de l'utilisateur est la suivante : il fait appel aux services consécutifs dont il a besoin (sur un même ou plusieurs sites) et il procède à des copier-coller ou à des transformations manuelles des données (le format peut varier selon le fournisseur). Parallèlement, il gère ses données, dans un éditeur de texte par exemple. Dans cette optique, Bigre proposera un environnement de travail adéquat, en intégrant tous les outils bioinformatiques accessibles au moyen d'une même et unique interface graphique.

Un grand problème se pose également au niveau de la sécurité, avec pour conséquence que le bioinformaticien préférera travailler dans un intranet sécurisé, plutôt que d'aller découvrir et utiliser de nouveaux services qui pourraient mieux lui convenir.

La figure 2.1 présente l'architecture actuelle du mode d'utilisation de la bioinformatique.

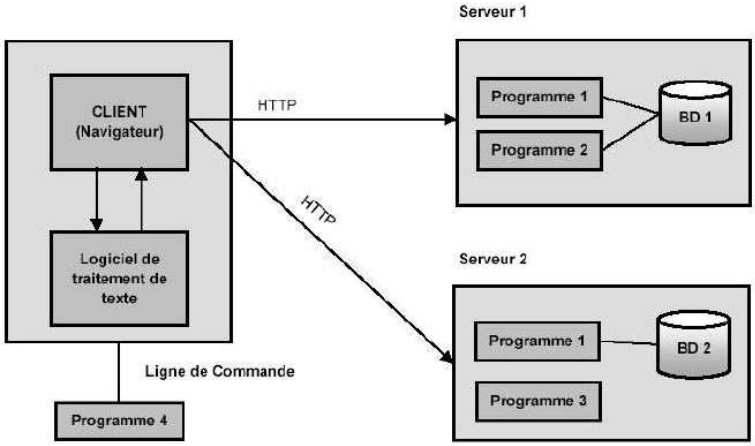

FIG. 2.1: Architecture actuelle
Source : [DDS03]

2.2 Les systèmes distribués

Actuellement, les bases de données et les services bioinformatiques sont principalement appelés via Internet. Chaque fournisseur propose un accès à ses outils via ses pages Web. Nous venons de voir quelle la situation actuelle concernant l'utilisation des services. Par rapport aux problèmes présentés (sécurité, hétérogénéité, disponibilité, environnement de travail), un système fédéré procurera certains avantages, que nous allons étudier.

> « *Un système distribué est une collection d'hôtes autonomes qui sont connectés par l'intermédiaire d'un réseau. Chaque hôte héberge et exécute un ou plusieurs composants tout en supportant un middleware, qui permet aux composants du système de coordonner leurs activités d'une telle façon que les utilisateurs perçoivent le système comme une capacité de traitement unique et intégrée. On oppose généralement système distribué à système centralisé.* » [Eng04]

Le projet Bigre propose une solution distribuée développée dans le cadre du métier mais il est toutefois générique dans sa conception et donc adaptable par la suite à d'autres domaines, par changement d'ontologie (la partie spécifique est l'ontologie de données, nous étudions les ontologies dans le chapitre 4).

Un système distribué permet de répondre à de nouvelles exigences [Eng04] :
- Évolution quantitative : Montée en charge du système, par exemple en multipliant ou répliquant les hôtes et les composants.
- Évolution qualitative : Évolution du système suite au changement du langage de programmation (hier en Cobol, aujourd'hui en Java, et demain en ? ?) ou au changement de la conception de l'architecture (déplacement des composants)
- Hétérogénéité : Les ressources peuvent être pré-existantes dans le système (*legacy*), nouvellement et entièrement développées (*from-scratch*) ou achetées (provenant de tiers).
- Partage et accès aux ressources (hardware, software et données) : Permet l'accès simultané et concurrent aux ressources (les composants). Qui dit partage dit sécurité et confidentialité.
- Tolérance aux pannes : On active un nouvel hôte lorsqu'un composant est en panne, on réplique les données pour se protéger des pannes de réseau, de bases de donnée ou d'ordinateur.

2.3 Le projet Bigre

Bigre (Bioinformatic Grid Ressources and Environment) est un exemple d'application distribuée en bioinformatique. Il s'agit d'un projet qui a débuté le 1er octobre 2003. Il est co-développé par l'Institut d'informatique des Facultés Universitaires Notre Dame de la Paix à Namur (FUNDP), par l'Unité de Recherche de Bioinformatique de l'Institut de Biologie et de Médecine Moléculaires, ULB Wallonie à Gosselies (IBMM) et par l'Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle de l'Université Libre de Bruxelles (IRIDIA).

Son architecture, au premier octobre 2004 [EBD+04], est le fruit de deux mémoires, celui de Laurent Debaisieux et Fernando De Souza [DDS03] et celui de Pierre Buyle et Quentin Dallons [BD03], en 2003. Le premier a étudié le partage de ressources bioinformatiques hétérogènes, et le second l'a enrichi et y a apporté la partie conception et implémentation d'une fédération de médiateurs. D'autres mémoires s'inscrivent dans le cadre du projet Bigre : [Den04], [San04].

2.3.1 Architecture générale

Trois sous systèmes

L'architecture générale de Bigre se présente sous forme de trois sous-systèmes et est représentée à la figure 2.2 [1].

[1] Note : les différentes figures présentées dans ce chapitre sont reprises du rapport [EBD+04].

- Les **services** sont proposés par les fournisseurs. Ils peuvent être natifs, c'est-à-dire développés pour être directement intégrés à Bigre, ou bien interfacés grâce à des « wrappers »[2], qui sont les composants façade intégrant des bases de données ou des outils existants.
- Les **médiateurs** se situent entre l'utilisateur et le fournisseur. Ils assurent différents rôles : la découverte dynamique de services, la sécurisation des communications par une infrastructure à clefs publiques, la comptabilité des utilisations (pour les services payants) et enfin le suivi de la qualité de service, le tout avec une intégration sémantique des services sur base d'une ontologie commune.
- Le **client** est l'application utilisateur qui va se connecter à un médiateur et assurer toute l'interaction avec les services, d'une manière homogène et cohérente. L'interface graphique proposée est indépendante des services et adaptée au profil de l'utilisateur (la personne débutante ou au contraire l'expert qui requiert un grand nombre d'options supplémentaires), cela résout le problème des interfaces changeantes selon les pages Web accédées.

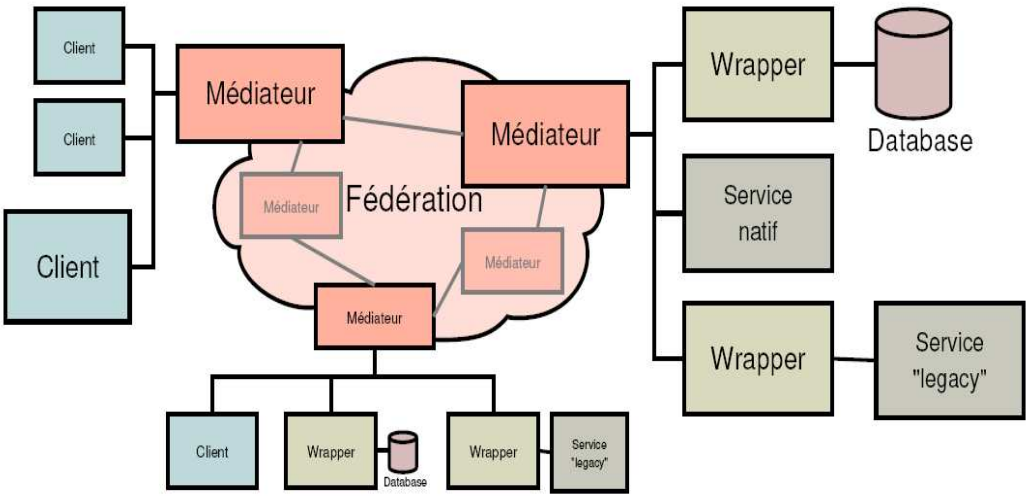

FIG. 2.2: Architecture générale de Bigre

Les médiateurs s'organisent en une **fédération**, ils forment donc un système distribué, avec les avantages qui en découlent. La gestion de ce système est non centralisée mais plutôt autogérée : les médiateurs assurent eux-même l'ouverture du système et la gestion de la fédération. C'est une organisation type Peer-to-Peer[3] où chaque média-

[2]Signifie « emballage »

[3]Les éléments d'un réseau informatique P2P sont à la fois clients et serveurs, contrairement aux systèmes client-serveur classiques où les machines clientes faisant partie du réseau contactent un serveur, une machine généralement très puissante en terme de capacités d'entrée-sortie, qui leur fournit des services. Ces services sont des programmes fournissant des données.

teur connaît un nombre restreint de voisins. Tout ceci est transparent aux clients et aux services, ils ne communiquent qu'avec un médiateur, par exemple le médiateur de leur université ou de leur entreprise.

Dynamique

La figure 2.3 permet de mieux comprendre les interactions entre les trois sous-systèmes. Les médiateurs sont représentés comme un ensemble « fédération ». Le client *c1* commence par rechercher des services et obtient la réponse *s1, s2, ...* ; il choisit d'utiliser *s1*, et en demande, s'il ne la possède pas déjà, la clé publique, ce afin de procéder de manière sécurisée. Il peut alors appeler *s1*, qui se procure la clé de *c1*, en vérifie les droits, et procède au traitement. Chacun fournit ensuite un rapport à la fédération.

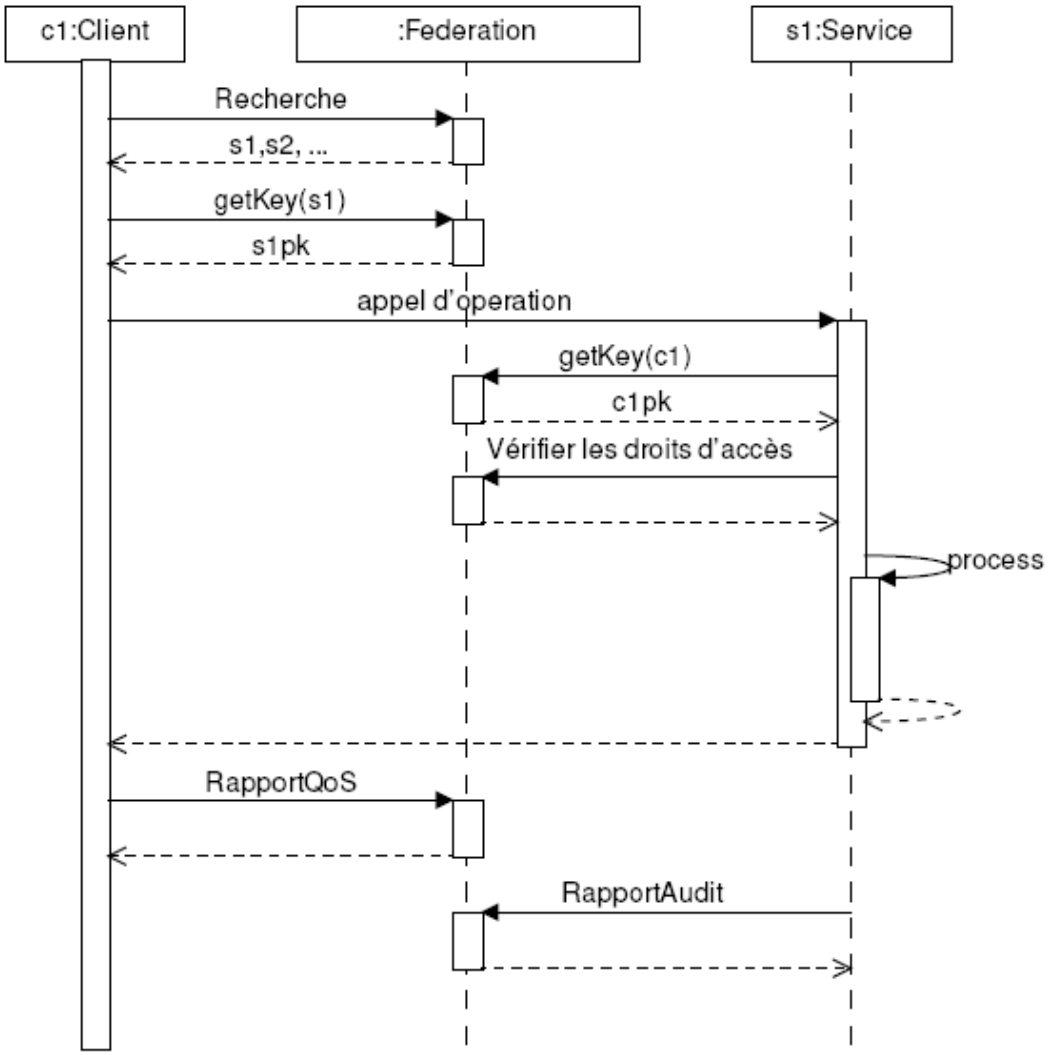

FIG. 2.3: Diagramme de séquencement général pour l'appel à un service

2.3.2 Architecture des médiateurs

Les médiateurs remplissent plusieurs fonctions.

- Le **répertoire de services** est un annuaire des services présents dans la fédération. Il permet d'un part de gérer l'information contenue dans l'annuaire : ajout ou modification de données sur les services ; et d'autre part de consulter ces informations et rechercher des services. C'est cet aspect « recherche » que nous étudierons.
- Le **répertoire d'ontologies** assure le maintien et la propagation d'ontologies dans la fédération, ainsi que l'accès aux ontologies connues et acceptées par le médiateur. C'est dans ces ontologies que s'effectueront les recherches de services. Nous allons donc étudier ces ontologies attentivement.
- Le **gestionnaire de sécurité** permet de garantir des communications sécurisées et se charge des droits d'accès aux services. Le médiateur assure donc un rôle d'autorité de certification.
- Le **gestionnaire de comptabilité** s'occupera de la facturation de l'utilisation des services payants. C'est le sous-système côté fournisseur qui fournira les informations d'utilisation de ces services.
- L'**évaluateur de la qualité de service (QoS)** maintient à jour l'information de qualité attendue pour un service, au moyen de rapports faits par l'application cliente. Il calcule un score basé sur plusieurs critères (le prix, la vitesse de traitement, la disponibilité, etc.) pour chaque service.
- Le **messager de fédération** propose aux composants d'un médiateur des moyens pour communiquer avec la fédération quand il en a besoin.

Dynamique

La figure 2.4 permet de mieux comprendre les interactions entre les trois sous-systèmes, avec mise en évidence du rôle de chaque composant du médiateur. Le scénario est le même que pour la figure précédente.

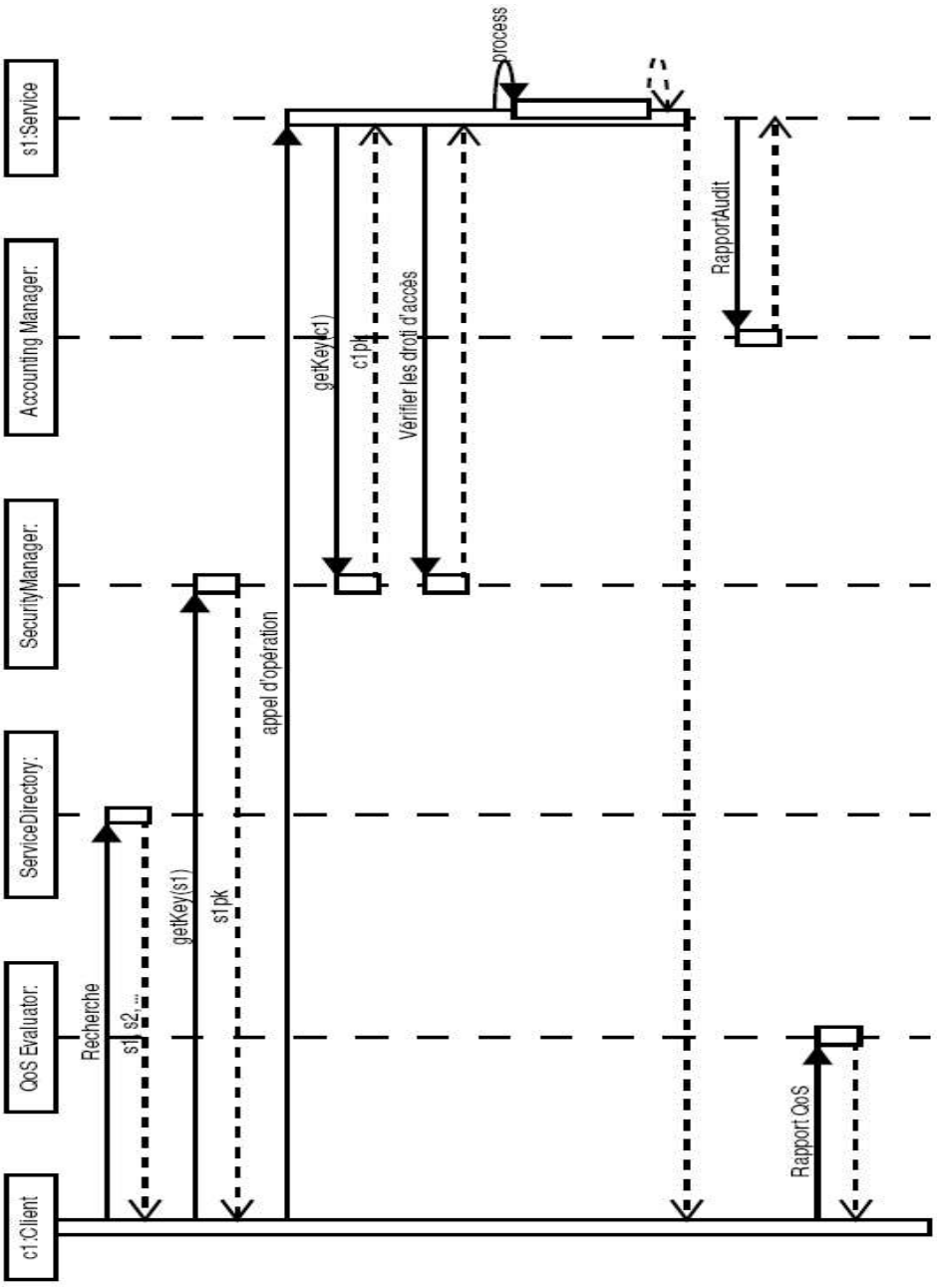

FIG. 2.4: Diagramme de séquencement général pour l'appel à un service - Mise en évidence des composants du médiateur

2.4 La recherche de services

Dans le chapitre précédent, nous avons vu que les services bioinformatiques sont nombreux. Entre autres, la suite EMBOSS qui regroupe plus de 150 outils dans sa version de base. Ces outils ont été classifiés selon leurs fonctions, chacun possède un nom et une courte description textuelle. Par exemple, le service *genemark* (qui est classé dans la catégorie *nucléic*, sous-catégorie *gene finding*) est décrit comme « *finds potential genes using a species specific HMM* ». Il est donc possible de faire une recherche soit sur le nom du service, soit à l'aide de mots-clé dans le texte de description. Ces mots-clé peuvent également être combinés sous la forme *et/ou* utilisée dans les moteurs de recherche.

Si l'on entre le mot-clé *translation*, la recherche fournit 86 résultats car *translation* est compris dans la présentation textuelle de 86 services bioinformatiques de la suite EMBOSS. Ce n'est donc pas une recherche suffisamment pertinente. L'intérêt serait d'enrichir la recherche de services, de manière à tenir compte de nombreux paramètres tels les propriétés (qu'un service soit payant ou non, disponible 24h sur 24, etc.), les données prises en entrée ou produites en sortie du service, les services qui en spécialisent d'autres, etc.

L'architecture de Bigre prend en compte cette recherche de services. Dans la figure 2.4, le client *c1* commence par rechercher des services auprès du *Service Directory* (le répertoire de services), qui lui renvoie la liste *s1, s2,* ...

Nous souhaitons aider l'utilisateur à la découverte de nouveaux outils bioinformatiques. Cette recherche de services pourra se faire grâce à l'annuaire dans lequel le fournisseur a inscrit ses services, pour lesquels il fournit une description complète qui reprend toute information nécessaire à sa découverte et à son utilisation par les utilisateurs d'outils bioinformatiques.

Afin que ces informations soient échangeables et surtout traitables par les différents composants logiciels du système distribué, il faudra que sa sémantique soit définie de manière non ambiguë. Nous allons étudier, en particulier dans le chapitre suivant, les langages prévus à cet effet (RDF, RDFS et OWL).

2.5 Conclusion

Dans ce chapitre, nous apprenons quelle est l'architecture actuelle en bioinformatique et quels avantages seraient procurés par un système distribué. Nous étudions une solution distribuée : Bigre. À partir de là, nous comprenons que la découverte de nouveaux services bioinformatiques est importante, c'est à ce niveau que notre étude se situe.

Chapitre 3

Le Web sémantique

OWL est le langage d'ontologie utilisé dans Bigre, nous allons donc aborder la question du Web sémantique pour en comprendre les fondements. Nous définirons le Web sémantique et en étudierons les trois langages de base : RDF, RDFS et OWL (chapitres 3 et 4 de [SS04]).

3.1 Présentation

Le Web sémantique est une extension du Web[1] permettant de publier, de consulter et, tout particulièrement, d'automatiser le traitement de connaissances précisément formalisées. Ce projet vise à introduire dans la toile certaines indications de sens afin de faciliter la recherche de documents et leur traitement automatisé.

3.1.1 Principe général

Le Web sémantique est entièrement fondé sur le Web « classique » et s'appuie sur sa fonction de base : un moyen de publier et consulter des documents. Seulement, les documents traités contiennent non pas des textes en langage naturel (français, anglais, etc.) mais des informations formalisées pour être traitées automatiquement. Ces documents sont générés, traités et échangés par des logiciels.

Il rend possible la compréhension de l'information aux ordinateurs. Le partage des connaissances se fait dans des langages essentiellement humains, le Web sémantique permet aux individus de s'exprimer dans des termes que les ordinateurs pourront également interpréter et échanger, et ce afin de résoudre des problèmes qui nous semblent fastidieux.

[1] Pour rappel, le Web (World Wide Web ou W3) est la méthode d'exploration d'Internet (Interconnected Networks ou Réseaux Interconnectés), par usage de l'hypertexte.

3.1.2 Principes techniques

Le Web sémantique est fondé sur les protocoles et langages standards du Web : le protocole HTTP et les URI (Uniform Ressource Identifier ou Identificateur Uniforme de Ressource). À ces standards s'ajoutent ceux qui sont propres au Web sémantique :

- *RDF* : modèle conceptuel permettant de décrire toute donnée ;
- *RDF Schema* : langage permettant de créer des vocabulaires, ensembles de termes utilisés pour décrire des choses ;
- *OWL* : langage permettant de créer des vocabulaires complexes à des fins de traitements logiques poussés (inférences).

Ces trois standards sont ouverts et issus du W3C[2]. Ils forment l'ossature du Web sémantique.

3.2 RDF

RDF (Resource Description Framework ou Cadre/Structure pour la Description de Ressources) est assez simple à comprendre, il s'agit d'une collection de constructions {*sujet, prédicat, objet*} appelées les déclarations. Ces triplets peuvent se voir comme étant un arc *prédicat* orienté du noeud source *sujet* au noeud destination *objet*. Par exemple, le sujet peut être un document à commenter, le prédicat une propriété de ce document (comme son titre) et l'objet sera la valeur de cette propriété.

RDF est donc un modèle de graphe qui décrit les données et permet un certain traitement automatique de celles-ci. Il a besoin d'être « sérialisé » (c'est-à-dire qu'il faut donner à RDF une syntaxe). Pour ce faire, le format le plus utilisé est RDF/XML[3] (fort prisé, car présenté sous une forme XML qui est un langage bien connu par la majorité), il en existe d'autres, notamment N3[4].

Comme le Web sémantique est, par définition, distribué, RDF se doit de permettre la collecte de informations à partir de sources distribuées. Cela se fait au moyen des URI qui permettent d'identifier les objets dans des systèmes décentralisés. De cette manière, les trois éléments d'une déclaration sont identifiés et peuvent être réutilisés dans un même ou dans plusieurs modèles.

Un *littéral* est un objet qui n'est pas une URI mais bien un contenu réel, une valeur. Il peut également être typé, le type étant donné entre crochets derrière sa valeur.

La *réification* est le fait qu'un triplet puisse être un noeud origine d'un autre triplet, à savoir une construction du type

[2]World Wide Web Consortium - `www.w3.org` : c'est un groupement fondé en 1994 qui crée des standards pour le Web, notamment le XML au niveau de l'architecture, le HTML pour l'interaction, le Web sémantique, etc.

[3]`http://www.w3.org/TR/rdf-syntax-grammar/`

[4]`http://www.w3.org/DesignIssues/Notation3.html`

$$[sujet \text{ — } prédicat \rightarrow objet] \text{ — } prédicat \rightarrow objet$$

Par exemple,

$$[cette\ section \text{ — } traite\ de \rightarrow RDF] \text{ — } introduit \rightarrow OWL$$

3.2.1 Un exemple RDF

FIG. 3.1: Une illustration du RDF
Source : `http://www.w3.org/TR/rdf-primer/`

Un exemple est proposé en figure 3.1. La personne Eric Miller
- est identifiée par l'URI `http://www.w3.org/People/EM/contact#me`
- est de type (`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`) `Personne` (`http://www.w3.org/2000/10/swap/pim/contact#Person`)
- a un nom complet (`http://www.w3.org/2000/10/swap/pim/contact#fullName`) `Eric Miller`
- a une adresse électronique (`http://www.w3.org/2000/10/swap/pim/contact#mailbox`) `em@w3.org`
- et a un titre (`http://www.w3.org/2000/10/swap/pim/contact#personalTitle`) `Dr.`

Les valeurs `Eric Miller` et `Dr.` sont encadrées d'un rectangle, car contrairement aux autres éléments de ce graphe RDF, ce ne sont pas des URI, mais bien des litéraux.

3.2.2 Une note sur le XML

Avant de présenter le format d'échange RDF/XML, nous devons comprendre le langage XML.

Le XML (eXtensible Markup Language ou Langage d'Annotation eXtensible) est la norme d'échange de documents informatisés qui intègre l'idée de métadonnée [5] et permet de définir les balises que l'on veut en fonction de ses besoins. L'idée est de produire des documents vraiment structurés, grâce à un langage qu'on peut définir en fonction des circonstances, mais selon une syntaxe très stricte.

En XML, les informations doivent être

- soit encadrées par des balises ouvrantes `<ELEMENT>` et fermantes `</ELEMENT>`. Les éléments doivent s'imbriquer proprement les uns dans les autres, aucun chevauchement n'est autorisé. Les éléments vides sont permis, selon le format `<ELEMENTVIDE/>`.
- soit incluses à l'intérieur même des balises, on parle alors d'attributs : `<ELEMENT ATTRIBUT="VALEUR">`.
- soit définies sous forme d'entités. Les entités sont des abréviations. Par exemple, si *Extensible Markup Language* est déclaré comme une entité associée à la notation *xml*, alors cette chaîne de caractères pourra être abrégée en *&xml;* dans tout le fichier XML. Une entité peut aussi représenter un fichier XML externe tout entier. Ainsi un même fichier XML pourra être utilisé par plusieurs pages XML différentes

Les *namespaces* (espaces de nommage) sont un mécanisme destiné à lever les ambiguïtés éventuelles des intitulés de balise. Par exemple `titre` peut aussi bien désigner le titre d'un ouvrage que celui d'une personne. Pour lever ces ambiguïtés, le mécanisme des namespaces consiste à utiliser des préfixes, par ex. `perso:titre` et `biblio:titre`. Ces préfixes doivent être déclarés comme associés à une URL qui peut être fictive, mais qui le plus souvent fera référence à l'organisme garant du vocabulaire en question. Nous retrouvons cette notion d'espaces de nommage dans les syntaxes basées sur le XML, à savoir XML/RDF, RDFS et OWL.

3.2.3 Le format RDF/XML

RDF/XML est une manière d'encoder un modèle RDF sous une forme XML. Les noeuds deviennent des éléments. Les classes ont une majuscule (`rdf:Description`) et les propriété une minuscule (`dc:title`). L'exemple de la figure 3.1 devient :

[5] Méta est un mot grec qui signifie « au milieu de, avec, après », il exprime la succession, le changement, la participation. Le *smiley* est une méta-donnée du texte, car il lui apporte une information d'humeur, d'intonation.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>

</rdf:RDF>
```

3.3 RDF Schema

RDF Schema (ou RDFS) est un vocabulaire qui permet de décrire des vocabulaires simples. Comment ? En se dotant d'un nombre minimum de concepts, nécessaires à la définition d'un vocabulaire. Il définit :

- La notion de **classe** qui est un ensemble de plusieurs objets.
- La propriété particulière **est une sous-classe de** qui permet de définir qu'une classe est un sous-ensemble d'une autre classe.
- La classe des **ressources** qui est la classe mère de toutes choses. Tout est ressource dans le web sémantique, sauf la notion de *litéral*. Toute classe est une sous-classe de la classe des ressources.
- La notion de **litéral** qui est une valeur comme une chaîne de caractère ou des chiffres : ces choses ne sont pas des concepts et ne peuvent être manipulés comme tels.
- La propriété **s'applique à la classe** (le rang) permettant ainsi de spécifier le champ d'application d'une propriété.
- La propriété **est l'objet de la propriété** (domaine) permettant ainsi de spécifier quelles sont les classes auxquelles ont peut affecter telle ou telle propriété

RDF est à la base du Web sémantique, mais est un système insuffisant. RDF Schema est utile pour créer des classifications et groupements de concepts et pour écrire des déclarations, mais ne suffit pas pour les besoins du monde réel . On peut reprocher à RDF et RDFS leurs insuffisances dues à la seule définition des relations entre objets par des assertions. Il s'agit d'apporter une meilleure intégration, une évolution, un partage et une inférence plus faciles des ontologies. Voilà pourquoi OWL a été créé.

3.4 OWL

Le W3C a mis au point OWL (Web Ontology Language ou Langage d'Ontologie Web) [owl], conçu pour étendre RDF et préciser les ontologies. Il enrichit le modèle des RDF Schemas en définissant un vocabulaire riche pour la description d'ontologies Web structurées (et complexes).

Une ontologie décrit un domaine de connaissances spécifiques en précisant les relations qui existent entre les différents termes. Par exemple, une ontologie s'intéressant à la zoologie spécifiera qu'il existe plusieurs sortes d'animaux, que les chats sont des mammifères, que les oiseaux pondent des oeufs et possèdent des ailes mais ne savent pas forcément voler, etc. Les ontologies font l'objet du chapitre 4.

Le site `www.schemaweb.info` est un site Web qui regroupe diverses ontologies, il est intéressant d'en sélectionner quelques unes et des les étudier à l'aide d'un outil (Protégé par exemple, présenté à la section 4.3), de manière à bien comprendre le concept d'ontologie. Ce site propose une ontologie du vin qui nous servira d'exemple par la suite.

OWL et RDF Schema permettent donc de définir des vocabulaires. RDF Schema définit le plus petit nombre de notions et de propriétés nécessaires à la définition d'un vocabulaire simple, essentiellement les notions de classe, ressources, littéral ainsi que les propriétés de sous-classe, de sous-propriété, de champ de valeur, de domaine d'application. OWL est un langage beaucoup plus riche qui, aux notions définies par RDF Schema, ajoute les propriétés de classe équivalente, de propriété équivalente, d'identité de deux ressources, de différence de deux ressources, de contraire, de symétrie, de transitivité, de cardinalité, etc., permettant de définir des rapports complexes entre des ressources.

3.4.1 Les trois sous-langages OWL

Le langage OWL est assez complexe, voilà pourquoi il se compose de trois sous langages qui proposent une expressivité croissante, chacun conçu pour des communautés de développeurs et des utilisateurs spécifiques : OWL Lite, OWL DL, OWL Full. Chacun est une extension par rapport à son prédecesseur plus simple.

Le langage **OWL Lite** répond à des besoins de hiérarchie de classification et de fonctionnalités de contrainte simples de cardinalité 0 ou 1. Une cardinalité 0 ou 1 correspond à des relations fonctionnelles, par exemple, une personne a une adresse. Toutefois, cette personne peut avoir un ou plusieurs prénoms, OWL Lite ne suffit donc pas pour cette situation.

Le langage **OWL DL** concerne les utilisateurs qui souhaitent une expressivité maximum couplée à la complétude du calcul (cela signifie que toutes les inférences[6] seront

[6]Voir chapitre 5

assurées d'être prises en compte) et la décidabilité (c'est-à-dire que tous les calculs seront terminés dans un intervalle de temps fini) du système de raisonnement. Ce langage inclut toutes les structures OWL avec certaines restrictions, comme la séparation des types : une classe ne peut pas aussi être un individu ou une propriété. Il est nommé *DL* car il correspond à la *logique descriptive* (présentée au chapitre 5, c'est un champ de la recherche qui a étudié un fragment décidable particulier de la logique de premier ordre).

Le langage **OWL Full** se destine aux personnes souhaitant une expressivité maximale, ainsi que la liberté syntaxique de RDF, mais sans garantie de calcul. En OWL Full, une classe peut également être un individu, il n'y a pas de séparation des types. C'est pour cela que la calculabilité ne peut être garantie.

3.4.2 Nécessité de l'OWL Full

Quel pourrait être l'intérêt de l'OWL Full si ce langage est non calculable ? C'est-à-dire que tous les calculs ne seront pas garantis de se terminer en un temps fini.

Parfois, la distinction entre une classe et un individus n'est pas simple : la classe est un nom et une collection de propriétés, elle sert à décrire un ensemble d'individus ; individus qui, appartenant à cette classe, en possèdent les propriétés. Ainsi, une classe correspondrait aux ensembles de choses qui apparaissent naturellement dans un domaine d'étude, et les individus seraient les entités réelles que l'ont peut regrouper dans ces classes. Cette distinction classe / instances peut s'embrouiller de deux manières, l'explication qui suit est directement reprise de `www.yoyodesign.org/doc/w3c/`[7] :

- *Les niveaux de représentation* : Dans certains cas, une chose qui est une classe évidente peut elle-même être considérée comme une instance de quelque chose d'autre. Par exemple, dans une ontologie du vin, on a la notion d'une classe *Grape* dont l'objet est de représenter l'ensemble de tous les cépages. Un exemple d'instance de cette classe est *CabernetSauvignonGrape*, qui représente dans ce cas le cépage Cabernet Sauvignon. Néanmoins, on peut considérer cette instance *CabernetSauvignonGrape* comme une classe à part entière, c'est-à-dire l'ensemble de tous les cépages Cabernet Sauvignon réels.
- *La sous-classe comparée à l'instance* : Il est très facile de confondre la relation instance de et la relation sous-classe de. Par exemple, il peut sembler arbitraire de préférer faire de *CabernetSauvignonGrape* un individu qui soit une instance de la classe *Grape* plutôt qu'une sous-classe de *Grape*. Ce n'est pas une décision arbitraire. La classe *Grape* représente l'ensemble de tous les cépages, de ce fait, toute sous-classe de *Grape* devrait représenter un sous-ensemble de ces cépages. Ainsi, on devrait considérer *CabernetSauvignonGrape* comme une instance de la

[7] Il s'agit de la traduction d'un document produit par le W3C

classe *Grape*, et non pas une sous-classe de celle-ci. Ce n'est pas la description d'un sous-ensemble des cépages, c'est un cépage.

La même distinction apparaît dans le traitement de la classe *Wine*. La classe *Wine* représente en réalité l'ensemble de toutes les variétés de vin (ou appellations), et non l'ensemble des bouteilles réelles qu'une personne achète. Dans une ontologie de remplacement, chaque instance de la classe *Wine* de l'ontologie courante pourrait désigner à la place une classe consistant en toutes les bouteilles de vin de ce type. Il est facile d'imaginer un système d'information, tel un système d'inventaire pour un marchand de vin, qui doive tenir compte individuellement de chaque bouteille de vin. L'ontologie sur le vin nécessiterait de pouvoir traiter les classes comme étant des instances pour mettre en oeuvre une telle interprétation. Le langage OWL Full autorise une telle expressivité, en permettant de traiter simultanément une instance de cépage comme étant une classe dont les instances sont des bouteilles de vin.

3.5 Conclusion

Dans ce chapitre, nous avons appris ce qu'est le Web Sémantique et nous avons étudié les langages RDF, RDF Schéma et OWL. OWL est le langage d'ontologie Web, il est sérialisé sous un format XML et convient donc bien à un système distribué pour le partage de connaissance entre les différents composants, voilà pourquoi il a été choisi pour être utilisé dans Bigre. Nous pouvons à présent nous intéresser aux ontologies et étudier le modèle Bigre à partir duquel nous pourrons rechercher de nouveaux services.

Chapitre 4

Les ontologies

Nous avons déjà abordé le concept d'ontologie dans le chapitre précédent. Nous allons l'approfondir et étudier l'ontologie de Bigre.

4.1 Définition

En philosophie, l'ontologie est l'étude de l'être en tant qu'être, c'est-à-dire l'étude des propriétés générales de ce qui existe.

En informatique, une ontologie est un ensemble structuré de concepts. Les concepts sont organisés dans un graphe dont les relations peuvent être des relations sémantiques ou des relations de composition et d'héritage (au sens objet). La structuration des concepts dans une ontologie permet de définir des termes les uns par rapport aux autres, chaque terme étant la représentation textuelle d'un concept.

L'ontologie est donc la définition d'un domaine, une sorte de dictionnaire non linéaire et complexe. A la lecture d'une ontologie, la personne est censée comprendre tout le domaine concerné.

La définition la plus connue d'une ontologie a été donnée par Gruber [Gru93] :

« *Une ontologie est une spécification explicite d'une conceptualisation.* »

Dans cette définition, la conceptualisation signifie un modèle abstrait d'un certain aspect du monde, prenant la forme d'un algorithme qui détermine des relations d'instances : l'individu i est une instance de la description de concept C si et seulement si i est toujours interprété comme un élément de C. L'algorithme de consistance détermine si une base de connaissance (qui consiste en un ensemble d'assertions et un ensemble d'axiomes terminologiques) est non contradictoire [SS04].

4.2 L'ontologie de Bigre

Dans le chapitre 2, nous avons étudié le projet Bigre et mis en évidence son architecture, où les clients contactaient les fournisseur via la fédération de médiateurs. Afin de les présenter au reste du système, le fournisseur avait inscrit dans l'annuaire ses services, pour lesquels il fournissait une description complète reprenant toute information nécessaire à sa découverte et à son utilisation par les utilisateurs d'outils bioinformatiques.

Comme ces descriptions seront échangées et surtout traitées par les composants des sous-systèmes fournisseur, médiateur et client, il est important que leur sémantique soit décrite d'une manière non ambigüe. Nous avons étudié les langages prévus à cet effet et nous abordons maintenant le modèle de descriptions proposé par Bigre.

L'ontologie de Bigre se compose de deux grandes parties : l'ontologie de données et l'ontologie de services.

4.2.1 L'ontologie de services

Nous devons bien comprendre le modèle entité-association proposé à la figure 4.1 :

Un **service** est, comme nous l'avons vu, soit un outil natif, soit une base de données ou un programme interfacé par un wrapper. Tout service possède une URI, qui l'identifie.

Un **type de service** est un regroupement de services, il reprend les éléments communs de leurs descriptions. Concrètement, *Blast* est un type de service et est instancié par *Blast EBI* et *Blast NCBI*, qui sont deux outils Blast situés à des URI différentes. Un service est toujours associé à un et un seul type de service.

Le **traitement** définit ce qu'effectuent les services : nom, description et documentation. Un traitement peut en spécialiser un autre, ou bien le généraliser. Il y a donc ici une hiérarchie de traitements, ce qui pourra donner des suggestions intéressantes dans notre étude. Un traitement prend des inputs et produit des outputs.

L'**interface** présentée par les services permet de définir leurs propriétés et les opérations qu'ils effectuent.

Une **propriété** est une caractéristique des services. Par exemple le prix, la disponibilité, la version, etc.

L'**instance de propriété** donne une valeur à une propriété pour un service.

Une **opération** définit une action que l'utilisateur peut effectuer auprès du service concerné. L'opération prend des données en entrée et en renvoie en sortie.

Une **donnée** peut être une propriété, une entrée d'opération, ou une sortie d'opération. Les entrées et sorties de traitement sont les entrées de la première opération de ce traitement (organisée par le scénario) et les sorties de la dernière opération.

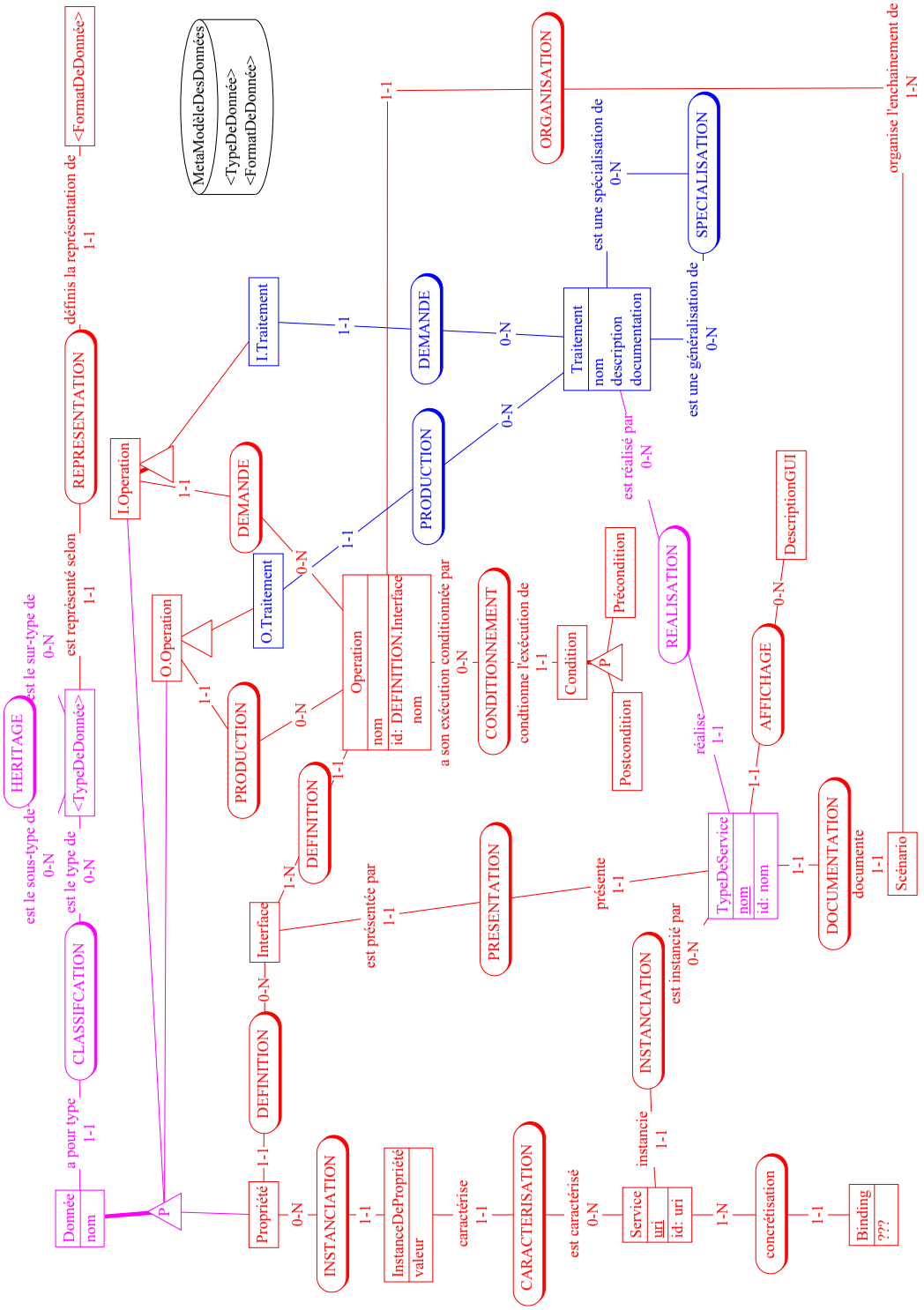

FIG. 4.1: Modèle entité-association de l'ontologie Bigre

Les **préconditions** et **postconditions** s'appliquent à une opération.

Le **scénario** documente un type de service, afin d'organiser l'enchaînement des opérations pour réaliser un traitement.

La **description GUI** permet l'affichage d'un type de service à l'écran.

Le **binding** concrétise le service, c'est le lien vers une couche middleware : RMI, Corba, etc.

Les données (propriétés, inputs, outputs) sont classifiées dans un **type de données**. Il y a une relation d'héritage de données, de la même manière que pour les traitements.

La partie du modèle {*donnée, type de données, héritage, format de données*} correspond à l'ontologie de données.

4.2.2 L'ontologie de données

Modéliser l'ontologie de données est quelque chose de très délicat, car il faut réfléchir à la bonne manière de représenter le domaine.

L'équipe Bigre présente son étude à ce niveau dans son rapport [EBD+04]. Afin de bien décrire les données, elle a étudié différentes ontologies déjà existantes dans le domaine et les a évaluées. Les ontologies étudiées sont TaO, Gene Ontology, SONG, MBO, RiboWeb, EcoCyc, et MyGrid.

En fait, deux ontologies ont été créés. L'une est entièrement consacrée aux données manipulées par l'algorithme d'alignement de séquences *Blast*, et l'autre est une ontologie plus générale.

Présentation de l'outil *Blast* :

> « *L'algorithme BLAST (Basic Local Alignment Search Tool) permet de réaliser une recherche de similarité entre une séquence et une collection de séquences. Cet algorithme est accessible via le Web sur des serveurs de différents instituts de recherche. Le plus connu est le NCBI qui gère parallèlement une banque de séquences (GenBank) mise à jour quotidiennement. L'accès au service BLAST du NCBI permet donc de comparer une séquence d'ADN ou de protéine avec la collection la plus large et la plus à jour.* (...) » *Olivier Dugas*, [EBD+04]

La seconde ontologie est plus générale, trois grands concepts forment cette ontologie :

– Les concepts purement liés aux entités biologiques et à leurs relations. On y trouve les molécules et leurs caractéristiques biologiques ou physicochimiques, il ne figure pas de caractéristiques relatives à la représentation de ces molécules.

– Les concepts de représentation mentales des entités biologique. Cette classe reprend les caractéristiques des entités biologiques tel que les représentations des molécules, les formules chimiques, etc.
– Les concepts liés à la définition des processus et analyses bioinformatiques. Le concept est définit par une relation entre une à plusieurs entités biologiques et une à plusieurs entités ou caractéristiques d'entités biologiques ainsi qu'un score qui qualifie cette relation.

4.3 Passage du modèle en OWL

Dans leur rapport scientifique [EBD+04], les chercheurs présentent la méthodologie adoptée pour transformer leur modèle en un modèle OWL.

Par rapport au modèle conceptuel, les types d'entités deviennent des classes, les types de relations deviennent des propriétés objets et les attributs des propriétés données.

Les contraintes de cardinalité sont conservées par l'ajout de restrictions : un rôle devient une propriété dans chacune des deux classes (qui étaient les types d'entités). Certaines de ces propriétés ont ensuite été éliminées, car elles ne leur semblaient pas nécessaires.

La figure 4.2 illustre l'ontologie OWL dans Protégé. Protégé est un outil qui permet aux utilisateur de construire des ontologies, il supporte les fichiers OWL.

L'annexe B présente l'ontologie de Bigre dans son format OWL.

Il faut noter que le modèle Bigre, dans sa version actuelle, est OWL Full, car elle possède une propriété dont le *range* est `owl:Class`, il s'agit de la propriété *hasDataType* concernant la classe *Data*. Si l'on supprime ce *range*, alors l'ontologie est DL. Nous avons vu la différence entre l'OWL DL et l'OWL Full au point 3.4 page 30.

4.4 Conclusion

Dans ce chapitre, nous avons étudié l'ontologie de données et de services Bigre. Nous souhaitons enrichir le spectre de la réponse proposée à l'utilisateur lorsqu'il recherche des service. Pour cela, par rapport aux types d'entités (ou aux classes OWL) *type de données* et *traitement* qui forment des hiérarchies, il nous faut disposer de mécanismes d'inférence. Nous allons donc étudier l'inférence et l'appliquer à l'ontologie de Bigre pour la découverte de nouveaux services.

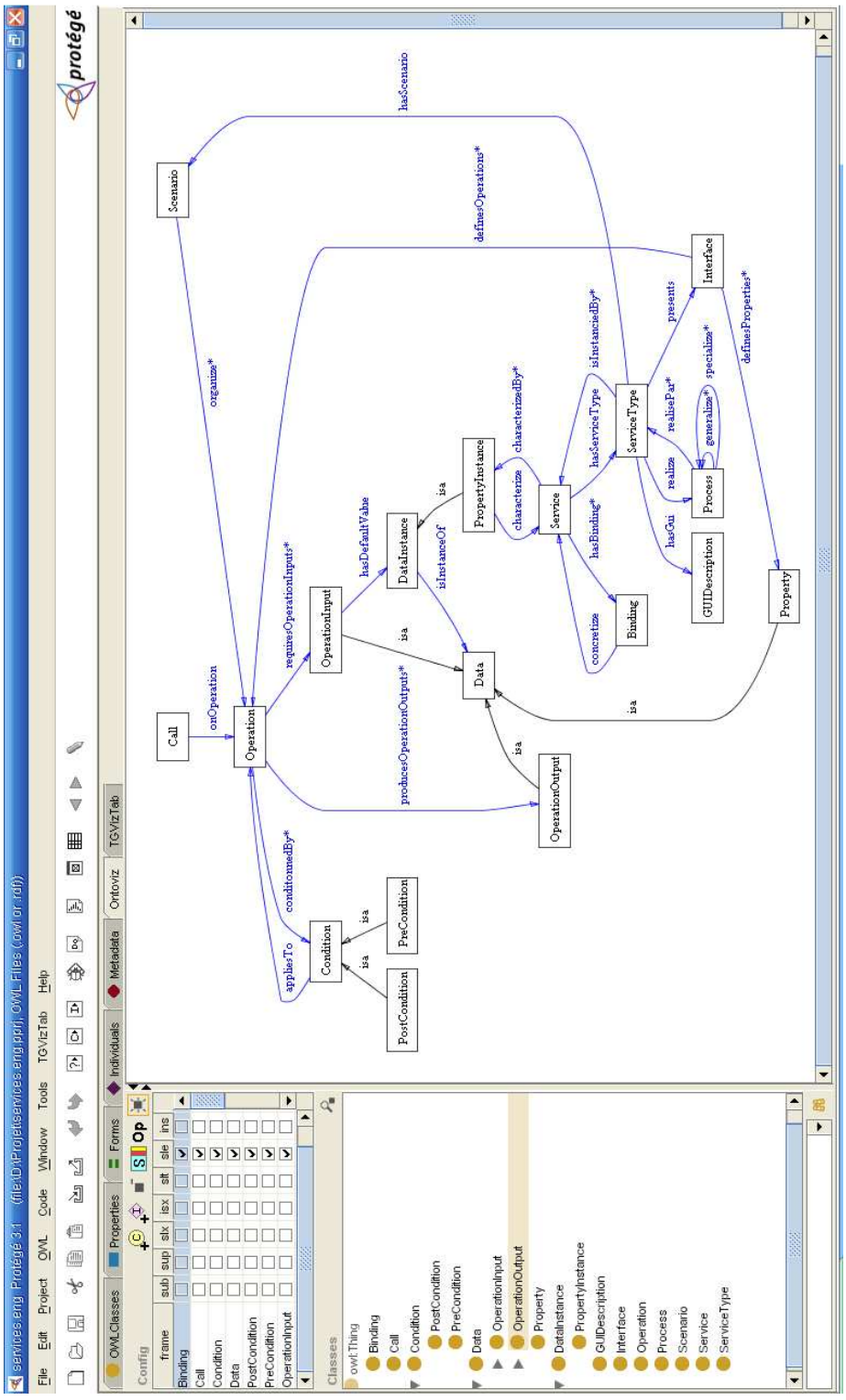

FIG. 4.2: L'outil Protégé et Visualisation du modèle en OWL

Chapitre 5

L'inférence

Ce chapitre a plusieurs objectifs : comprendre ce qu'est l'inférence afin de voir en quoi elle sera utile à notre recherche de services, étudier la logique descriptive qui est à la base des langages d'ontologies[1], choisir l'outil qui nous permettra de raisonner dans l'ontologie, et enfin, construire les premières questions sur le modèle.

5.1 Définition

L'inférence est l'opération qui consiste à admettre une proposition en raison de son lien avec une proposition préalable tenue pour vraie. C'est un terme général dont les mots raisonnement, déduction, induction, etc., sont des cas spéciaux. On voit donc en l'inférence la réalisation d'une déduction logique, utilisée par un système expert pour paraître intelligent. L'inférence est illustrée à la figure 5.1.

Le moteur d'inférence est le programme qui réalise les déductions logiques d'un système expert à partir d'une base de connaissances (faits) et d'une base de règles. Les règles sont utilisées pour manipuler les connaissances et aboutir à des conclusions qu'on espère lumineuses.

Nous souhaitons donc profiter de ce mécanisme d'inférence afin de naviguer dans l'ontologie pour pouvoir proposer de l'information pertinente à l'utilisateur.

[1] Voir `http://www-db.research.bell-labs.com/user/pfps/talks/history/all.html`

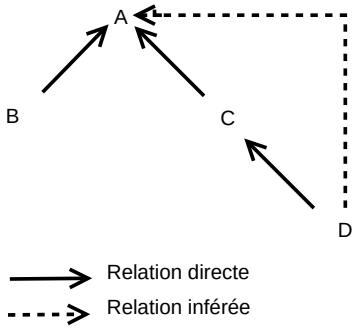

FIG. 5.1: Relation inférée

5.2 La logique descriptive

5.2.1 Introduction à la logique descriptive

La logique descriptive (DL – [SS04], [BCM+03]) est une famille de formalismes de représentation de la connaissance basées sur la logique, conçue pour représenter et raisonner sur la connaissance d'un domaine d'application d'une manière structurée et bien comprise. Elle descend des réseaux sémantiques et des frames [Jac04].

Les notions de base sont les *concepts* (prédicats unaires) et les *rôles* (relations binaires). La DL est caractérisée par les *constructeurs* qu'elle fournit pour former des concepts et rôles complexes à partir des concepts et rôles atomiques (les prédicats unaires et les relations binaires). Les deux constructeurs *conjonction* ($\sqcap$) et *disjonction* ($\sqcup$) sont respectivement l'intersection et l'union de concepts. La *restriction existencielle* ($\exists R.C$ où R est un rôle et C un concept) et la *restriction de valeur* ($\forall R.C$) sont telles que :

- pour qu'un objet x soit l'instance de l'ensemble $\exists R.C$, il doit exister un objet, soit y, qui est une instance de C et est lié via R à x.
- x est une instance de l'ensemble $\forall R.C$ si tous les objets liés à x via R sont des instances de C.

La *TBox* est le composant terminologique de la logique descriptive, elle est utilisée pour introduire des noms (abbréviations) pour les concepts complexes. Elle permet la déclaration de *general concepts inclusion axioms* (GCI - axiomes généraux d'inclusion de concepts).

La logique descriptive offre une capacité de raisonnement qui déduit de la connaissance implicite à partir de celle donnée explicitement dans la TBox. L'algorithme de *subsumption*[2] détermine les relations de sous et super-concepts : un concept C est englobé par un concept D si chaque instance de C est aussi une instance de D, c'est-à-dire

[2] *To subsume* signifie *englober*.

si chaque modèle de la TBox interprète C comme un sous ensemble de l'interprétation de D. Cet algorithme est utilisé pour calculer la *taxonomie* de la TBox : c'est la hiérarchie de *subsumption* de tous les concepts introduits dans la TBox. L'algorithme de *satisfiability* (consistance) teste si un concept donné peut effectivement être instancié.

Plus le pouvoir d'expressivité de la logique descriptive est élevé, plus la complexité de la *subsumption* et de la *satisfiability* sera élevée. La DL expressif $\mathcal{SHIQ}$ est à la base de la conception du langage d'ontologies Web OWL, même s'il a une syntaxe RDF Schema. Les développeurs ont essayé de trouver un bon compromis entre l'expressivité et la complexité de raisonnement. Raisonner en $\mathcal{SHIQ}$ est calculable, mais a toutefois une complexité ExpTime élevée (c'est-à-dire que raisonner en $\mathcal{SHIQ}$ prendra un temps de calcul exponentiel [Dev04]). Néanmoins, des raisonneurs $\mathcal{SHIQ}$ hautement optimisés tels FaCT et Racer se comportent assez bien en pratique.

Nous allons étudier $\mathcal{AL}$ qui est le langage DL minimal présentant un intérêt pratique.

5.2.2 La syntaxe $\mathcal{AL}$

Les descriptions de concepts sont formées selon les règles syntaxiques suivantes : Soient A et B des concepts atomiques, R un rôle atomique et C et D des descriptions de concepts. $\mathcal{AL}$ est un langage descriptif minimal :

$C,\ D$	$\rightarrow$	$A \mid$	(concept atomique)
		$\mathrm{T} \mid$	(concept universel : $\mathrm{A} \sqcup \neg\ \mathrm{A}$)
		$\bot \mid$	(concept bottom : $\neg\mathrm{T}$)
		$\neg A \mid$	(négation de concept atomique)
		$C \sqcap D$	(intersection)
		$\forall R.C \mid$	(restriction universelle)
		$\exists R.T$	(quantification existentielle limitée)

Voici un exemple pouvant être exprimé en $\mathcal{AL}$:

- Soient *Personne* et *Féminin* des concepts atomiques. Alors *Personne* $\sqcap$ *Féminin* et *Personne* $\sqcap \neg$*Féminin* (les personnes qui sont ou non féminin) sont des concepts $\mathcal{AL}$.
- Soit *a-enfant* un rôle atomique, nous pouvons former les concepts :
  1. *Personne* $\sqcap\exists$*a-enfant.*T (les personnes qui on un enfant),
  2. *Personne* $\sqcap\forall$*a-enfant.Féminin* (toutes les personnes dont les enfants sont des filles)
  3. et *Personne* $\sqcap\forall$*a-enfant.*$\bot$ (les personnes qui n'ont pas d'enfant).

5.2.3 La syntaxe $\mathcal{SHIQ}$

Le langage $\mathcal{AL}$ ne remplit pas les exigences de restrictions de cardinalité sur les rôles ou les exigences de types (entiers, strings, etc.). $\mathcal{SHIQ}$ est un langage DL au pouvoir d'expressivité supérieur. Il ajoute la négation de concepts aribtraires, les restrictions (qualifiées) de cardinalité, les hiérarchies de rôle, l'inverse de rôles, la transitivité de rôles et les types de données. Les nouveaux constructeurs sont :

$\exists R.C$	(quantification existentielle complète)
$\neg C$	(négation de concepts arbitraires)
$\alpha\ n\ R$	(restriction de cardinalité β)
$\alpha\ n\ R.C$	(restriction de cardinalité β qualifiée)
$\alpha_n\ R$	(restriction concrète de domaine β)
	où α est : $\leq$, $\geq$, $=$
	et β est : *au plus*, *au moins*, *exactement*

Dans l'exemple,
- si *Femme* $\equiv$ *Personne* $\sqcap$ *Féminin*, alors *Femme* $\sqcap \exists$*a-enfant.Personne* dénote les mères et $\neg$*Femme* dénote les individus qui ne sont pas des femmes ;
- *Mère* $\sqcap \geq 3$*a-enfant* sont les mères avec plus de trois enfants et
- *Mère* $\sqcap =3$*a-enfant.Féminin* sont les mères ayant exactement trois filles ;
- *Personne*$\sqcap \geq_{18}$*a-âge* dénote les adultes (les personnes ayant plus de 18 ans).

5.2.4 Sémantique DL

Un axiome général d'inclusion de concept (GCI) est de la forme $C \sqsubseteq D$. Une TBox est un ensemble fini de GCIs.

L'*interprétation* $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ consiste en un ensemble non vide $\Delta^{\mathcal{I}}$ appelé le *domaine* de $\mathcal{I}$, et une fonction $.^{\mathcal{I}}$ qui associe pour chaque nom de concept A un ensemble $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ et pour chaque nom de rôle R une relation binaire $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

L'interprétation de concepts complexes est définie comme suit :
$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$,
$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$,
$\neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$,
$(\exists\ R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid$ *il existe un* $e \in \Delta^{\mathcal{I}}$ *tel que* $\langle d, e\rangle \in R^{\mathcal{I}}$ *et* $e \in C^{\mathcal{I}}\}$,
$(\forall R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid$ *pour tout* $e \in \Delta^{\mathcal{I}}$ *si* $\langle d, e\rangle \in R^{\mathcal{I}}$ *alors* $e \in C^{\mathcal{I}}\}$.

Une interprétation $\mathcal{I}$ satisfait le GCI $C \sqsubseteq D$ *si* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ satisfait la TBox $\mathcal{T}$ si $\mathcal{I}$ satisfait tous les GCI dans $\mathcal{T}$. Dans ce cas, $\mathcal{I}$ est appelé *modèle* de $\mathcal{T}$. Un élément $d \in C^{\mathcal{I}}$ est appelé une *instance de C*. Si $\langle d, e\rangle \in R^{\mathcal{I}}$, alors e est appelé un *R-successeur de d*.

Un concept C est *satisfiable* par rapport à une TBox $\mathcal{T}$ si il y a un modèle $\mathcal{I}$ *de* $\mathcal{T}$ tel que $C^{\mathcal{I}} \neq \oslash$ Un concept C est *subsumed* par un concept D ($C \sqsubseteq_{\mathcal{T}} D$) si pour chaque modèle $\mathcal{I}$ de $\mathcal{T}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Deux concepts sont *équivalents* si ils se *subsument* mutuellement ($C = D$ si $C \sqsubseteq D$ et $D \sqsubseteq C$).

La ABox est l'ensemble des assertions. Par exemple, si la TBox contient le concept *Personne*, alors la ABox peut contenir *Jean* qui en est une instance.

5.3 Evaluation d'outils d'inférence

Nous souhaitons profiter de l'inférence dans l'ontologie Bigre. Nous avons choisi trois raisonneurs que nous présentons et comparons ensuite dans un tableau récapitulatif : $\mathcal{F}$lora-*2*, Jena, et Racer.

5.3.1 Flora

$\mathcal{F}$lora-*2* [YKZ03] est un langage orienté objet de base de connaissance et un environnement de développement d'applications. Il est un dialecte de *F-logic* avec des extensions incluant la meta-programmation dans le style de *HiLog* et de la logique dans le style de *Transaction Logic*. Il a été conçu pour être extensible et flexible, il fournit donc un support pour la conception modulaire grâce à ses modules dynamiques. Il est aussi un compilateur qui traduit la meta-programmation vers du code Prolog unifié.

Les applications de $\mathcal{F}$lora-*2* incluent les agents intelligents, le Web sémantique, la gestion d'ontologie et l'intégration d'information. Il se base sur le moteur d'inférence d'XSB [3] qui est un système de programmation logique et de bases de données déductives.

$\mathcal{F}$lora-*2* appartient à la famille de la programmation logique, il raisonne donc à l'aide d'un arbre de décision [Jac04]. Il a l'avantage de supporter l'inférence grâce à l'opérateur `::` sur les classes.

Il est cependant trop isolé par rapport à l'ontologie développée en OWL. Il faudrait développer et maintenir des outils de conversion entre les deux langages. Il existe F-OWL[4] qui est un moteur d'inférence d'ontologie pour OWL dont le mécanisme est implémenté dans $\mathcal{F}$lora-*2*. Il permet de raisonner sur un modèle d'ontologie défini dans le langage standard OWL selon les recommandations du W3C.

F-OWL valide son développement selon les *tests cases* définis par le W3C. Le projet F-OWL n'est plus maintenu depuis septembre 2003. Il n'est pas intéressant de retenir un raisonneur qui ne sera plus corrigé, alors que la communauté OWL évolue chaque jour.

[3] `http://xsb.sourceforge.net`

[4] `http://fowl.sourceforge.net/`

5.3.2 Jena

Jena [W3Ca] est un cadre de travail Java open source permettant de construire des applications de Web sémantique. Elle fournit un environnement de programmation pour RDF, RDFS et OWL, ainsi qu'un moteur d'inférence basé sur les règles. Le cadre de travail inclut :

- API RDF ;
- lecture et écriture RDF en RDF/XML, N3, et N-Triples ;
- API OWL ;
- stockage en mémoire et persistant ;
- RDQL - un langage d'interrogation du RDF.

Le sous-système d'inférence de Jena est conçu pour permettre à certains moteurs d'inférence ou raisonneurs d'être connectés à Jena. Le terme inférence est utilisé pour se référer au processus abstrait de dérivation d'informations supplémentaires, et le terme raisonneur pour faire référence à un code objet spécifique qui effectue cette tâche.

La structure de la machinerie d'inférence est illustrée à la figure 5.2. La *ModelFactory* associe l'ensemble de données avec le raisonneur pour créer un nouveau modèle qui contiendra les déclarations supplémentaires qui ont été dérivées de l'ensemble de données, en utilisant les règles et mécanismes d'inférence implémentés dans le raisonneur. La distribution Jena inclut des raisonneurs prédéfinis :

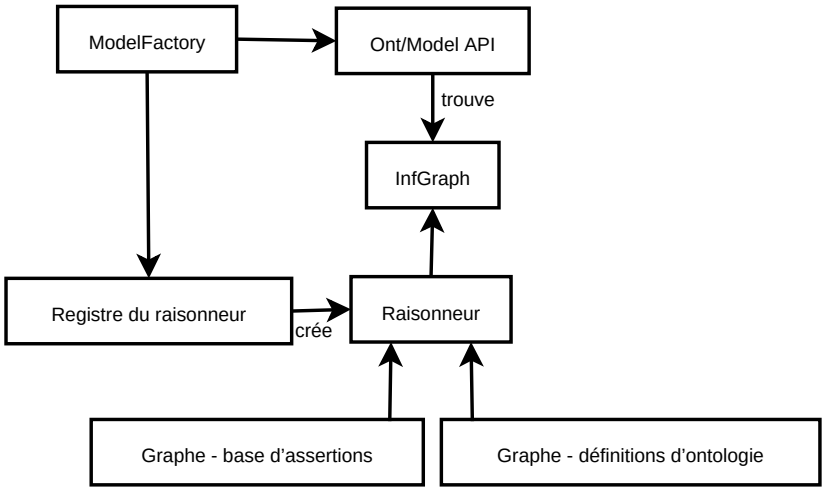

Fig. 5.2: Machinerie d'inférence

- un raisonneur transitif qui implémente les propriétés transitives et symétriques de `rdfs:subPropertyOf` et `rdfs:subClassOf` ;
- un raisonneur de règles RDFS ;

– des raisonneurs pour OWL, OWL Mini et OWL Micro qui sont des implémentations incomplètes du langage OWL Full, ils couvrent le sous-ensemble OWL Lite ;
– un raisonneur générique basé sur les règles définie par l'utilisateur.

Au moment de notre analyse, le raisonneur OWL supportait l'OWL Lite et une partie de l'OWL Full, en ne supportant toutefois que les cardinalités 0 ou 1. Ce raisonneur convient donc pour des ontologies simples et régulières.

L'avantage est que nous travaillons directement sur le modèle OWL. La programmation se fait en Java qui est le langage utilisé dans le projet Bigre.

L'ontologie Bigre va au delà des cardinalités 0 ou 1, et donc les raisonneurs proposés dans la distribution de Jena ne conviennent pas.

Il est possible de connecter d'autres raisonneurs. Nous avons choisi Racer qui implémente directement le langage $\mathcal{SHIQ}$ de la logique descriptive.

5.3.3 Racer

Le système Racer (RenamedABox and Concept Expression Reasoner ou Raisonneur d'expression de concepts et de ABox renommées) [MH04] est un système de représentation de la connaissance pour le calcul DL. Il implémente $\mathcal{SHIQ}$.

Racer se présente sous la forme d'un serveur qui peut être accédé par le protocole TCP ou HTTP (voir figure 5.3) :.

– JRacer est l'interface TCP Java qui peut être utilisé depuis des programmes clients qui souhaitent accéder au serveur ;
– pour un accès HTTP, on utilisera DIG qui est l'interface HTTP ;
– Rice (Racer Interactive Client Environment) est le client graphique de Racer, il utilise JRacer ;
– Racer peut être utilisé dans Protégé (outil présenté à la section 4.3).

En DL, une base de connaissance consiste en une TBox et une ABox. La TBox représente la connaissance conceptuelle ; et la connaissance sur les instances et le domaine sont représentées dans la ABox.

Conventions de nommage : C(concept), *CN* (nom de concept), *IN* (nom d'individu), *ON* (nom d'objet), R (rôle), *RN* (nom de rôle), *AN* (nom d'attribut), *ABN* (nom d'ABox), *TBN* (nom de TBox), *KBN* (nom de base de connaissance), *name* (nom de toute sorte), n (un nombre naturel), *[]* (optionnel).

Les tableaux 5.1, 5.2, 5.3 et 5.4 présentent la syntaxe Racer par rapport à la notation DL pour les constructeurs, les termes booléens, les restrictions qualifiées et les restrictions de nombre.

Le GC1 $C_1 \sqsubseteq C_2$ se note *(implies* C_1 C_2*)* ; L'équivalence de concept $C_1 = C_2$ se note *(equivalent* C_1 C_2*)* ; La disjonction de concept (les concepts n'ont pas d'instances communes) se note *(disjoint* $C_1 \ldots C_n$*)*.

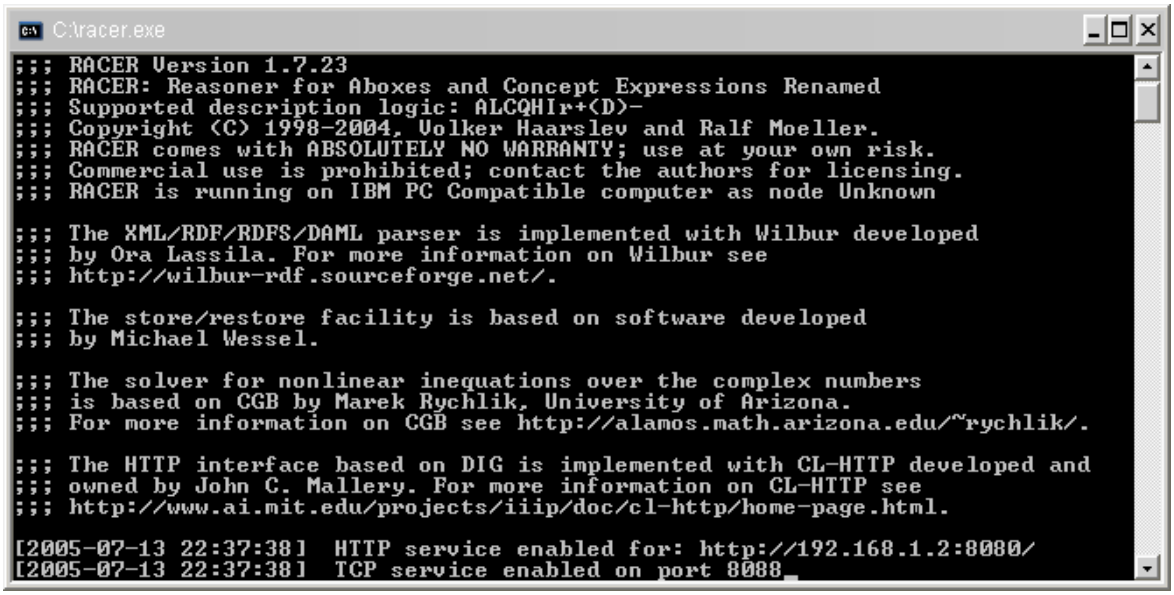

FIG. 5.3: Capture d'écran du serveur Racer lancé sous Windows

$$
\begin{array}{rl}
C \rightarrow & \text{CN} \mid \\
& \textit{*top*} \mid \\
& \textit{*bottom*} \mid \\
& \textit{(not } C\textit{)} \mid \\
& \textit{(and } C_1 \ldots C_n\textit{)} \mid \\
& \textit{(or } C_1 \ldots C_n\textit{)} \mid \\
& \textit{(some } R\ C\textit{)} \mid \\
& \textit{(all } R\ C\textit{)} \mid \\
& \textit{(at-least } n\ R\ [C]\textit{)} \mid \\
& \textit{(at-most } n\ R\ [C]\textit{)} \mid \\
& \textit{(exactly } n\ R\ [C]\textit{)} \mid \\
& \textit{(a AN)} \\
\text{et } R \rightarrow & \textit{RN} \mid \\
& \textit{(inv RN)}
\end{array}
$$

TAB. 5.1: Constructeurs DL dans Racer

	Notation DL	Syntaxe Racer
Négation	$\neg C$	$(not\ C)$
Conjonction	$C_1 \sqcap \ldots \sqcap C_n$	$(and\ C_1 \ldots C_n)$
Disjonction	$C_1 \sqcup \ldots \sqcup C_n$	$(or\ C_1 \ldots C_n)$

TAB. 5.2: Termes booléens

	Notation DL	Syntaxe Racer
Restriction d'existence	$\exists R.C$	$(some\ R\ C)$
Restriction de valeur	$\forall R.C$	$(all\ R\ C)$

TAB. 5.3: Restrictions qualifiées

	Notation DL	Syntaxe Racer
Restriction au plus	$\leq n\ R$	(at-most n R)
Restriction au minimum	$\geq n\ R$	(at-least n R)
Restriction exacte	$= n\ R$	(exactly n R)
Restriction au plus qualifiée	$\leq n\ R\ C$	(at-most n R C)
Restriction au minimum qualifiée	$\geq n\ R\ C$	(at-least n R C)
Restriction exacte qualifiée	$= n\ R\ C$	(exactly n R C)

TAB. 5.4: Rappel des restrictions de cardinalités

Les *features* (ou attributs, notés $\mathcal{F}$) restreignent un rôle $\mathcal{R}$ à être fonctionnel, à savoir que chaque individu peut avoir au plus une valeur pour ce rôle.
Les *rôles transitifs* ($\mathcal{R}^+$) : si les deux paires IN_1 et IN_2 ; et IN_2 et IN_3 sont reliées par le rôle transitif R, alors IN_1 et IN_3 sont également liés par R.
La *hiérarchie de rôles* définit les relations sous et super-rôles. Si R_1 est un super-rôle de R_2, alors toute paire d'individus liée par R_2 l'est également par R_1. Les propriétés du super-rôle ne sont pas dans tous les cas héritées par le sous-rôle, car la hiérarchie de rôles ne peut pas être cyclique, comme présenté au tableau 5.5 qu'il faut lire ainsi : si un rôle RN_1 est déclaré comme un simple rôle $\mathcal{R}$ et qu'il a un *feature* $\mathcal{F}$ RN_2 comme super-rôle, alors RN_1 sera également un *feature*.

		Super-rôle $RN_2 \in$		
		$\mathcal{R}$	$\mathcal{R}^+$	$\mathcal{F}$
Sous-rôle RN_1	$\mathcal{R}$	$\mathcal{R}$	$\mathcal{R}$	$\mathcal{F}$
déclaré comme	$\mathcal{R}^+$	$\mathcal{R}^+$	$\mathcal{R}^+$	-
élément de	$\mathcal{F}$	$\mathcal{F}$	$\mathcal{F}$	$\mathcal{F}$

TAB. 5.5: Héritage des propriétés de rôles

Un *feature* ne peut être transitif. Les rôles transitifs ou ayant des sous-rôles transitifs ne peuvent pas être utilisés dans les restrictions de nombres. Un rôle peut être déclaré avec des restrictions de domaine et de rang.

Modèle de fichier Racer :

```
(in-knowledge-base la-tbox la-abox)

(signature
:atomic-concepts(  un-concept-atomique
                   un-autre)
:roles(           (un-role :domain un-concept-atomique :range un-autre))
:transitive-roles((son-parent)
                  (un-rôle-transitif :parents son-parent))
:individuals(      un-individu
                   un-autre-individu)
)

(related un-individu un-autre-individu  un-role)
```

Dans la signature, on peut définir les concepts atomiques, les rôles, les rôles transitifs, les features, les individus et les objets. Il y a une limitation dans l'implémentation de la version actuelle de Racer : les contraintes qui sont utilisées pour définir des attributs sur les individus ne fonctionnent que si la signature ne contient pas de section *:individuals*.

RQL (Racer Query Language) est le langage d'interrogation pour Racer Une requête est de la forme

```
(retrieve <liste des objets> <corps de requête>)
```

où `<liste des objets>` est de la forme `(?x ?y)`
et `<corps de requête>` serait par exemple `(?x ?y rôle)` : « *Donnez-moi tous les couples (x, y) liés par rôle* ».

Les requêtes sont présentées dans une syntaxe S-expression. Une S-expression (ou Expression Symbolique) est une convention pour la représentation de données ou d'expressions d'un programme sous forme textuelle. Les S-expressions sont utilisées dans la famille de langages Lisp, la propriété la plus commune est l'utilisation de la notation préfixée parenthésée (connue sous le nom de Notation polonaise de Cambridge).

DIG (Description Logic Implementation Group) [Bec03] est un interface pour les systèmes DL tels que Racer. C'est un schéma XML avec des fonctionnalité d'interrogation et de déclaration, un exemple de déclaration est donné à la figure 5.4.

5.3.4 Récapitulatif

Nous résumons les caractéristiques des trois technologies présentées au tableau 5.6.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tells
  xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dl.kr.org/dig/2003/02/lang
  http://dl-web.man.ac.uk/dig/2003/02/dig.xsd"
  uri="urn:uuid:abcdefgh-1234-1234-12345689ab">
    <defconcept name="driver"/>
    <equalc>
        <catom name="driver"/>
        <and>
            <catom name="person"/>
            <some>
                <ratom name="drives"/>
                <catom name="vehicle"/>
            </some>
        </and>
    </equalc>
    <defconcept name="person"/>
    <defconcept name="vehicle"/>
    <defrole name="drives"/>
</tells>
```

FIG. 5.4: Exemple DIG

	Flora-2	Le raisonneur OWL de Jena	Racer
Présentation	langage orienté objet qui est traduit vers du code Prolog	raisonneur distribué avec Jena	serveur interrogeable en TCP ou HTTP
Famille	programmation logique, arbre de décision	cadre de travail Java	description logique
Inférence	::	*subClassOf*	$\sqsubseteq$
Intégration	isolé : développer et maintenir des outils de conversion	travaille directement à partir du modèle OWL	lit les fichiers OWL
	solution : F-OWL, mais projet arrêté	mais ne supporte pas tout OWL Full	et gère OWL DL

TAB. 5.6: Récapitulatif : Flora – Jena – Racer

5.4 Transcription du modèle

Nous avons porté le modèle Bigre (page 35) sous Racer. Les types d'entité deviennent des concepts et les types d'association deviennent des rôles sur lesquels nous définissons le domaine et le rang. Une erreur dans la version de Racer utilisée se présente pour les contraintes quand la section *:individuals* de la signature est non vide, nous ne représentons donc pas les attributs et les cardinalités. Ce n'est pas un problème, le but étant par la suite que Racer lise un fichier OWL, celui-ci est déjà correct concernant ces contraintes. La figure 5.5 illustre le modèle dans Rice[5].

```
(in-knowledge-base modelacademique test-model)

(signature

:atomic-concepts( service
                  typedeservice
                  traitement
                  binding
                  instancepropriete
                  descriptiongui
                  scenario
                  interface
                  operation
                  propriete
                  condition
                  precondition
                  postcondition
                  donnee
                  inputoperation
                  outputoperation
                  inputtraitement
                  outputtraitement
                  type
                  formatdonnee
                )

:roles( (instancie :domain service :range typedeservice)
        (estinstancie :inverse realise)

        (realise :domain typedeservice :range traitement)
        (estrealise :inverse realise)

        (concretise :domain binding :range service)
        (estconcretise :inverse concretise)
```

[5]Rappel : Rice (Racer Interactive Client Environment) est le client graphique de Racer

```
        (caracterise :domain instancepropriete :range service)
        (estcaracterise :inverse caracterise)

        (affiche :domain descriptiongui :range typedeservice)
        (estaffiche :inverse affiche)

        (documente :domain scenario :range typedeservice)
        (estdocument :inverse documente)

        (presente :domain typedeservice :range interface)
        (estpresente :inverse presente)

        (organise :domain scenario :range operation)
        (estorganise :inverse organise)

        (definit :domain interface :range propriete)
        (estdefini :inverse definit)

        (propinstancie :domain instancepropriete :range propriete)
        (propestinstancie :inverse propinstancie)

        (opdefinit :domain interface :range operation)
        (opestdefini :inverse opdefinit)

        (conditionne :domain condition :range operation)
        (estconditione :inverse conditionne)

        (opdemande :domain operation :range inputoperation)
        (opestdemande :inverse opdemande)
        (opproduit :domain operation :range outputoperation)
        (opestproduit :inverse opproduit)

        (demande :domain traitement :range inputtraitement)
        (estdemande :inverse demande)
        (produit :domain traitement :range outputtraitement)
        (estproduit :inverse produit)

        (classifie :domain type :range donnee)
        (estclassifie :inverse classifie)

        (represente :domain formatdonnee :range type)
        (estrepresente :inverse represente)

      )

:transitive-roles(
        (specialise :domain traitement :range traitement)
        (herite :domain type :range type)
      )
```

```
:individuals( untype
              unautretype
              ...)
)
(related untype unautretype  herite)
...
```

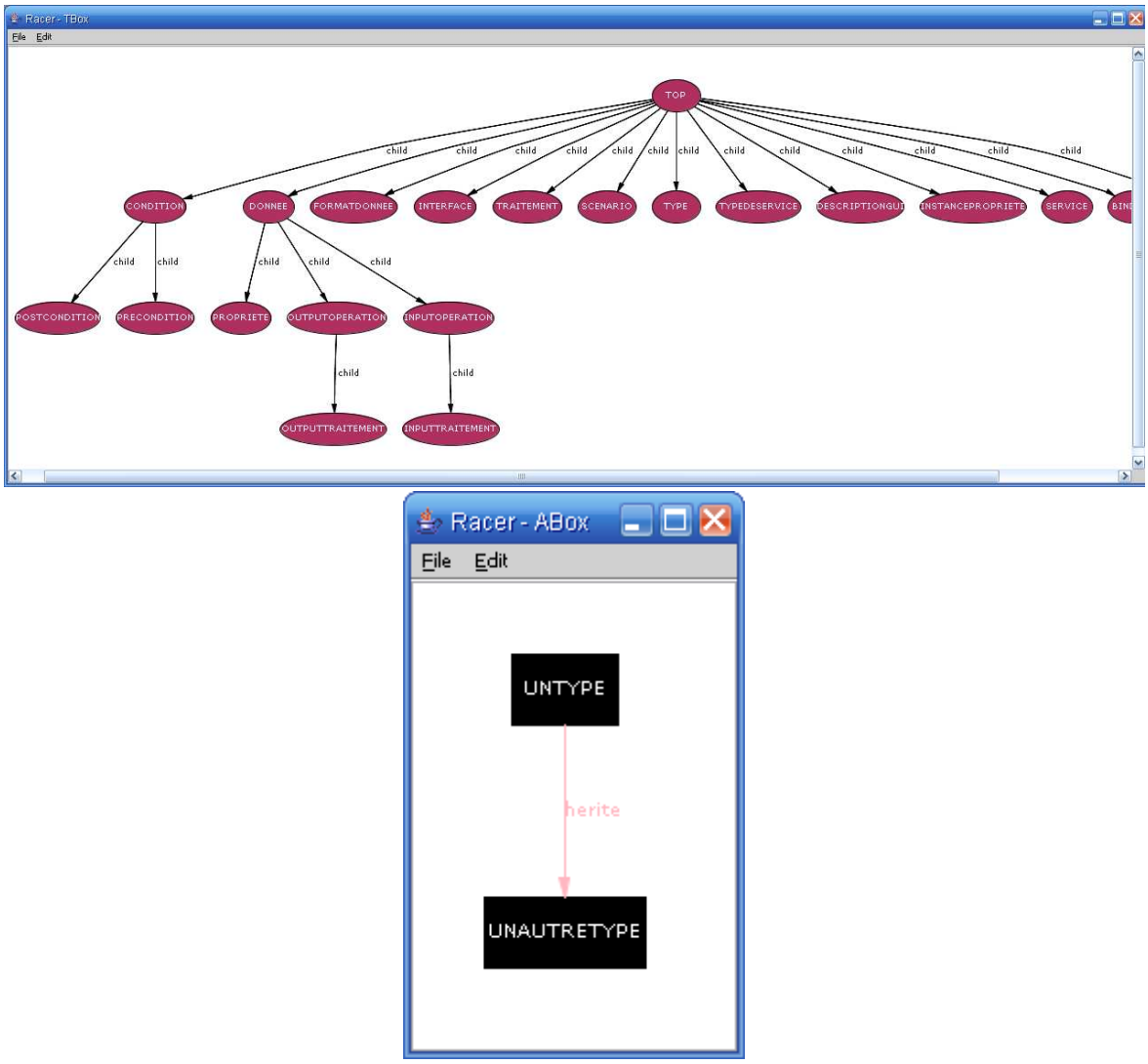

FIG. 5.5: Tbox et Abox

5.5 La recherche de services

Sur base de ce modèle, nous pouvons nous demander comment découvrir de nouveaux services. Lorsque l'utilisateur est à la recherche de ces services, nous souhaitons lui proposer des requêtes intéressantes.

Dans cette section, nous allons commencer par présenter quelques questions simples que l'on peut poser sur le modèle. Ensuite, nous verrons comment poser des questions en profitant des hiérarchies de données et de services.

Nous sommes à chaque fois à la recherche de services, nous nous demandons donc : « *Je recherche tous les services qui ...* ». Pour chaque demande, nous présentons la requête Racer correspondante.

5.5.1 Questions simples

Les phrases suivantes sont des questions qui sont susceptibles d'être posées :

« *Je recherche les services qui instancient tel type de service* » devient :

```
(retrieve (?services) (?services le-type-de-service instancie))
```

« *Je recherche les services dont le type réalise tel traitement* » devient :

```
(retrieve (?services) (and (?services ?tds instancie)
                           (?tds le-traitement realise)))
```

« *Je recherche les services dont le type réalise un traitement qui prend en entrée telle donnée* » devient :

```
(retrieve (?services) (and (?services ?tds instancie)
                           (?tds ?tt realise)
                           (?tt la-donnee demande)))
```

« *Je recherche les services dont le type réalise un traitement qui prend en entrée une donnée de tel type* » devient

```
(retrieve (?services) (and (?services ?tds instancie)
                      (?tds ?tt realise)
                      (?tt ?donnee demande)
                      (?donnee le-type-de-donnee classifie)))
```

5.5.2 Rôles transitifs

Un service prend des données en entrée et en renvoie en sortie, ce sont les inputs et les outputs. Il exécute un certain traitement de ces données. Voyons ce service comme étant une fonction *f*, qui nécessite certains arguments (les inputs) et procure un résultat (les outputs).

$$f(a,\ b,\ c,\ d,\ ...) \ \rightarrow\ r,\ s,\ t,\ u,\ v,\ ...$$

Hiérarchie d'inputs

Nous avons remarqué une hiérarchie au niveau des types de données, que nous avons traduite par une propriété de transitivité sur le rôle *herite*. Au niveau des inputs, l'utilisateur pourrait souhaiter connaître les services qui prennent une donnée d'un certain type en entrée, ou bien toute donnée d'un type dont il hérite. Si le type demandé est un sous-type de celui pris en input par le programme, alors il sera également accepté par le programme. En effet, supposons que dans l'ontologie, on ait précisé que le type entier spécialise le type réel. Si je suis à la recherche de programmes qui acceptent en entrée un nombre entier, je peux donc en déduire que tous les programmes travaillant sur nombres réels accepteront le nombre entier, car il en est un sous-type.

Illustration : c est un sur-type de y, et l'utilisateur pose une requête sur y. Nous pouvons donc proposer f à cet utilisateur.

$$f(a,\ b,\ \begin{array}{c} x \\ \uparrow \\ c \\ \uparrow \\ y \end{array}\ ,\ ...) \ \rightarrow\ r,\ s,\ t,\ ...$$

$$?\ \lambda(y)$$

où λ indique une recherche sur un argument en entrée

$$\Rightarrow\ \lambda = f$$

Traduction en une requête Racer :

```
(retrieve (?services) ( or (and (?services ?tds instancie)
                                (?tds ?tt realise)
                                (?tt ?donnee demande)
                                (?donnee le-type classifie)
```

```
                    )
                   (and (?services ?tds instancie)
                        (?tds ?tt realise)
                        (?tt ?donnee demande)
                        (?donnee ?surtype classifie)
                        (le-type ?surtype herite)
                    )
               )
)
```

Cette requête se comprend comme : *Donnez-moi tous les services qui prennent en entrée une donnée de type* le-type *ou tous les services qui prennent en entrée une donnée d'un super-type de* le-type.

Etant donné que le rôle *herite* est transitif, Racer infère sur tous les super-types de *le-type* et non seulement ses parents directs.

Hiérarchie d'outputs

De même, cette logique s'applique aux outputs. La hiérarchie se fait toutefois dans l'autre sens, à savoir que si l'utilisateur recherche un service renvoyant une donnée d'un certain type, il pourrait être intéressé par les services qui ont comme outputs des données d'un sous-type à ce qu'il recherche. Dans l'exemple entiers/réels, si il recherche les services qui renvoient des réels, alors un service renvoyant un entier pourrait répondre à ses attentes étant donné qu'un entier est un réel.

$$f(a,\ b,\ c,\ \ldots) \ \rightarrow\ r,\ \begin{array}{c} n \\ \uparrow \\ s \\ \uparrow \\ m \end{array}\ ,\ t,\ \ldots$$

$$?\ \kappa(n)$$

où κ indique une recherche sur un argument en sortie

$$\Rightarrow\ \kappa = f$$

Traduction en une requête Racer :

```
(retrieve (?services) ( or (and (?services ?tds instancie)
                                (?tds ?tt realise)
                                (?tt ?donnee demande)
```

```
                    (?donnee le-type classifie)
                )
               (and (?services ?tds instancie)
                    (?tds ?tt realise)
                    (?tt ?donnee demande)
                    (?donnee ?soustype classifie)
                    (?soustype le-type herite)
                )
           )
)
```

Cette requête se comprend comme : « *Donnez-moi tous les services qui renvoient en sortie une donnée de type* le-type *ou tous les services qui renvoient en sortie une donnée d'un sous-type de* le-type ».

Il est techniquement possible de proposer une recherche de services sur les sous-types en entrée ou bien les super-types en sortie. Dans ce cas, l'utilisateur tenterait de mettre en argument d'entrée un sur-type à celui attendu par le programme, il ne sera donc pas assez spécifique pour le programme. A l'inverse, s'il attend un certain type de données en sortie, et qu'il en reçoit un sur-type, alors ce type n'est encore une fois pas assez spécifique.

Hiérarchie de traitements

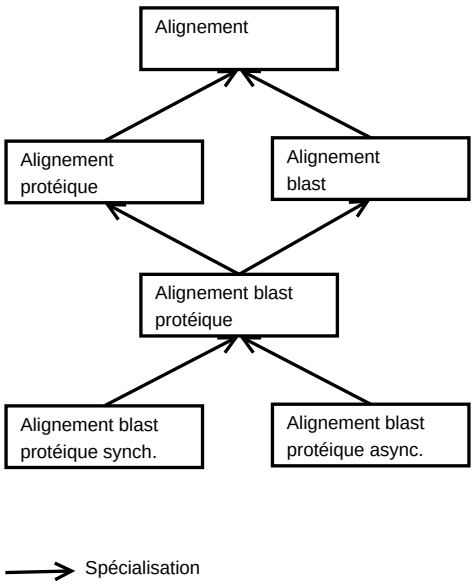

FIG. 5.6: Spécialisation de traitements

Illustration de la hiérarchie des traitements (voir figure 5.6) : le type de service *blast-p* réalise le traitement *alignement-blast-proteique* qui a un nom, une description et une documentation. *alignement-blast-p-synchrone* et *alignement-blast-p-asynchrone* le spécialisent. De plus, *alignement-blast-proteique* spécialise *alignement-blast* et *alignement-proteique* qui eux-mêmes spécialisent le traitement *alignement*. Un traitement ne doit pas forcément avoir un (ou des) service(s) associé(s). Ainsi, on peut hiérarchiser les traitements dans des concepts plus généraux, comme ici le traitement *alignement*.

L'utilisateur qui recherche les services dont le type réalise un certain traitement pourraient s'intéresser aux services d'un traitement que le traitement cible spécialise. Par exemple, s'il recherche les services associés à *Alignement blast protéique asynchrone*, on pourrait lui proposer les services implémentant *Alignement blast protéique*, *Alignement protéique*, *Alignement blast* et *Alignement*. La requête sera :

```
(retrieve (?services) ( or (and (?services ?tds instancie)
                                (?tds le-traitement realise)
                           )
                           (and (?services ?tds instancie)
                                (?tds ?sur-traitement realise)
                                (le-traitement ?sur-traitement  specialise)
                           )
                       )
)
```

A l'inverse, l'utilisateur voulant réaliser un *Alignement* peut se demander quels traitements le spécialisent. La requête sera :

```
(retrieve (?services) ( or (and (?services ?tds instancie)
                                (?tds le-traitement realise)
                           )
                           (and (?services ?tds instancie)
                                (?tds ?sous-traitement realise)
                                (?sous-traitement le-traitement  specialise)
                           )
                       )
)
```

5.5.3 Combinaisons

Nous pouvons supposer que l'utilisateur a plus d'un critère de recherche. Soient b et s ces critères supplémentaires.

$$f(a,\ b,\ \begin{matrix} x \\ \uparrow \\ c \\ \uparrow \\ y \end{matrix}\ ,\ \ldots) \ \rightarrow\ r,\ s,\ t,\ \ldots$$

$$? \ \lambda\kappa(b, y, s)$$

où $\lambda\kappa$ indique une recherche sur un argument indifféremment en entrée ou sortie

$$\Rightarrow \ \lambda\kappa = f$$

Cette optique de combinaison de critères est fort intéressante et permet d'étendre ou de restreindre le résultat grâce à des questions *et ou*, comme illustré à la figure 5.7.

5.6 IHM

Concernant la façon de présenter la recherche à l'utilisateur, nous avons envisagé différentes options. Nous mettons en évidence quelques moyens d'interaction :

1. **la recherche par mots clés** : c'est la méthode utilisée actuellement par les suites de programmes bioinformatiques, comme nous l'avons vu au point 2.4, page 23. Comme présenté dans l'exemple, les mots-clés recherchent l'information par rapport aux noms des services et à leur description textuelle. Ici, nous pouvons les appliquer à tous les champs de l'ontologie ;
2. **la recherche multi-critères** est celle présentée à la figure 5.7, l'utilisateur peut y spécifier les mots clés et les champs sur lesquels rechercher ces mots. Cette IHM peut être enrichie pour permettre une meilleure utilisation des mots par la précision *contient, ne contient pas, égal, n'est pas égal, supérieur, inférieur* ;
3. **voyage 2D ou 3D dans l'ontologie** : comme l'ontologie est un catalogue de services et de données, il est possible d'y voyager à l'aide d'une arborescence en deux ou trois dimensions. À chaque noeud sélectionné, on en développe les voisins. TouchGraph[6] est un outil graphique qui permet une visualisation 3D. On peut naviguer d'un noeud à l'autre par double clic et, de cette manière, l'arborescence évolue en fonction du noeud sélectionné pour afficher ses voisins directs et indirects, comme présenté en figure 5.8. Un tel utilitaire, adapté à une ontologie OWL, offre une belle manière de découvrir l'information. Il permet d'aborder la complexité des ontologies d'une manière simple et attractive. Dans l'instanciation de l'ontologie Bigre, un noeud *service* permettrait d'en voir les *bindings*, les *instances de propriétés*, et le *type de service* par exemple. Un passage vers ce *type de service* montrerait tous ses *services* associés, etc. TouchGraph est dorénavant intégré à l'outil Protégé.

[6] `www.touchgraph.com`

5.7 Conclusion

Dans ce chapitre, nous avons étudié l'inférence et la logique descriptive. Nous avons évalué trois raisonneurs et en avons choisi un pour inférer sur le modèle Bigre que nous avons traduit dans Racer. Sur cette base, nous avons réfléchi à certaines requêtes intéressantes pour la découverte de services. Ces requêtes exploitent la hiérarchie qui existe sur les données et sur les traitements.

Ces hiérarchies restent cependant assez simples vu qu'elles ne permettent que la généralisation/spécialisation. Nous souhaitons enrichir la sémantique qui existe dans l'ontologie en permettant de nouveaux types de relations sur les données. Pour cela, nous allons étudier le *part-whole* et nous verrons ensuite comment exploiter les relations *part-whole* dans la recherche de services.

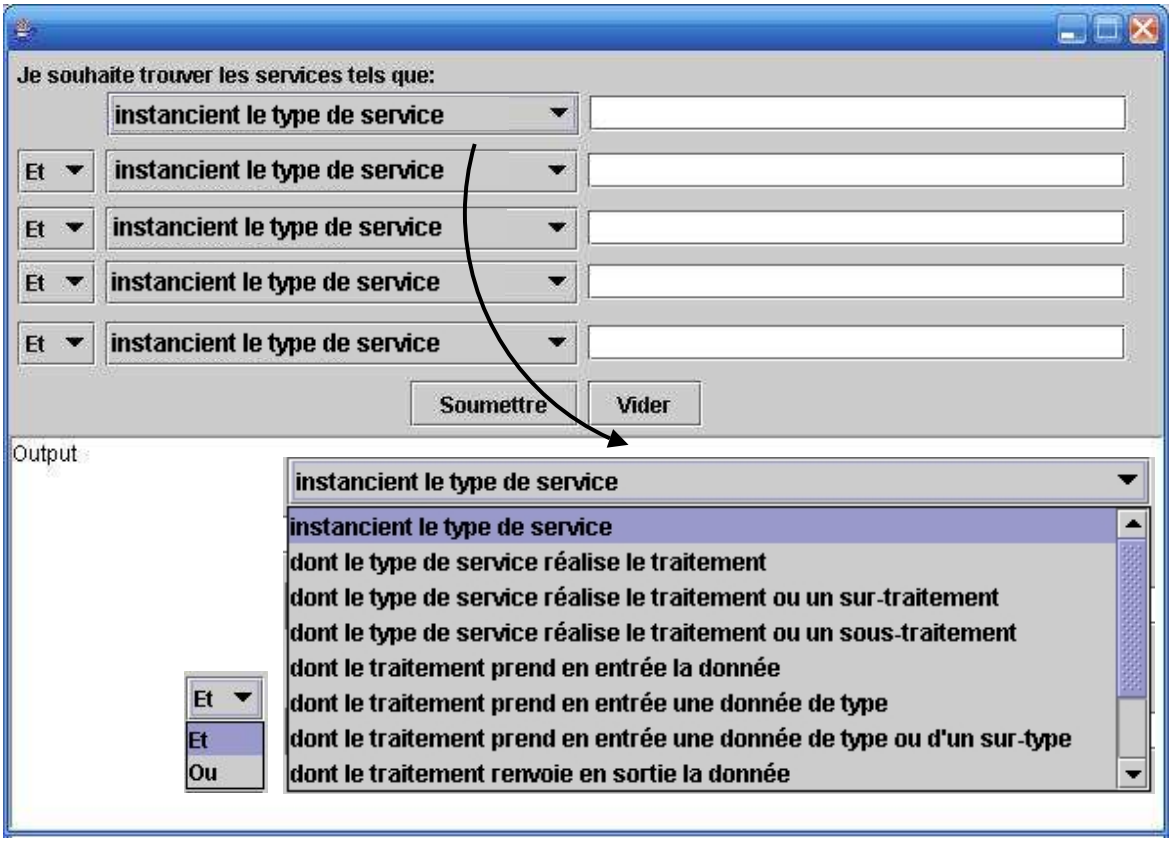

FIG. 5.7: Permettre une combinaison de critères de recherche

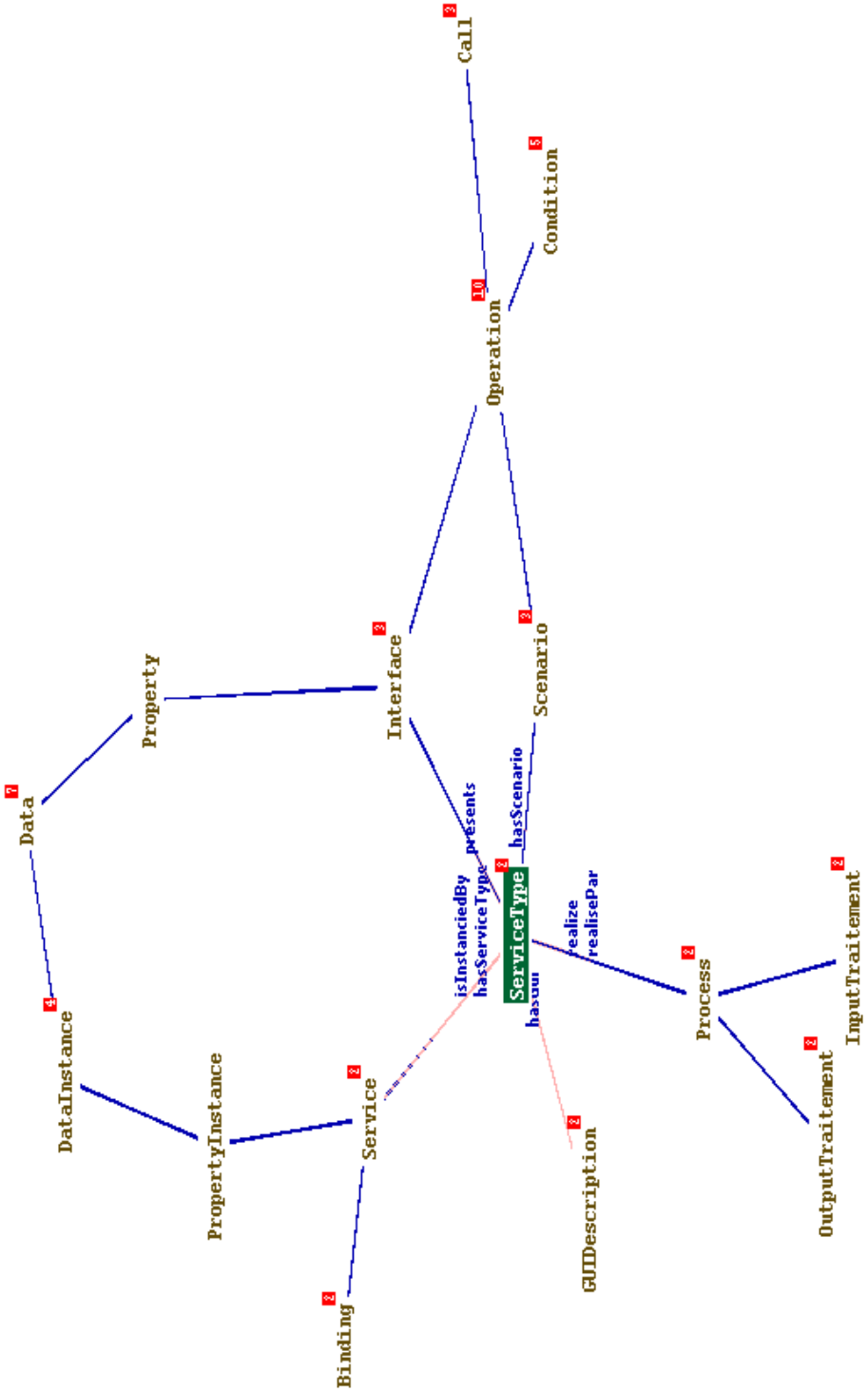

FIG. 5.8: TouchGraph utilisé dans Protégé

Chapitre 6

Le Part-Whole

Dans ce chapitre, nous allons légèrement nous écarter du domaine bioinformatique car, avec nos faibles connaissances en ce domaine, il nous est difficile de trouver des exemples qui pourraient bien convenir. Les bioinformaticiens pourront ensuite appliquer nos raisonnements et conclusions à leur domaine.

Après plusieurs lectures ([BCM+03], [Sat95], [GP96], [Ber95], [MARR03], [PL]), nous pouvons admettre que [BCM+03] propose l'étude la plus complète en matière de relation *part-whole*, la présentation théorique ci-après en est donc largement inspirée.

Le *part-whole* (également appelé relation de composition, ces deux termes seront utilisés indifféremment dans ce chapitre) est une relation qui lie les parties (les *part*) au tout (*le whole*). Par exemple, le moteur est un objet *part* de la voiture, qui est le *whole*. Le lien qui existe entre moteur et voiture s'appelle le lien *part-of* (*est la partie de*).

6.1 La relation part-whole

Un objet composite est un objet formé par l'agrégation d'un ensemble d'objets, appelés ses composants, qui décrivent chacun une partie de l'objet composite. La relation qui lie un objet composite à ses composants est appelée relation de composition. L'objet composite est le *whole*, ses composants sont les *part*.

De nombreuses questions se posent : Qu'est-ce qu'un objet composite ? Quelles propriétés peuvent être attachées à la relation de composition ? Comment définir et manipuler un objet composite ? Existe-t-il une inférence propre à la composition ? Quelles sont les contraintes liées aux langages pour la description explicite d'objets composites ? Quels domaines et applications sont concernés par les objets composites ? Qu'est-ce qu'un bon modèle d'objets composites ?

À la relation *part-whole*, ou relation de composition, sont associées différentes sémantiques. Il faut des modèles de représentation qui ont un pouvoir d'expressivité élevé. Le besoin est de manipuler un ensemble d'objets comme une seule entité logique (pour accroître l'expressivité et les performances du modèle).

6.1.1 Les sept types de relation part-whole

La relation **composant/objet** traduit le fait que les composants montrent certaines formes de relations structurelles ou fonctionnelles entre eux et le composite auquel ils appartiennent. Exemples : Les roues font partie des voitures, les chapitres font partie des livres, une poignée fait partie d'une valise. Cette relation est la plus utilisée dans les domaines de l'ingénierie de la connaissance.

Elément/collection : Les éléments d'une collection ne requièrent pas une fonction particulière ou un agencement structurel particulier par rapport aux autres éléments de la collection ou à la collection elle-même. Exemples : Un arbre fait partie d'une forêt, un avion fait partie d'une escadrille, un élève fait partie d'une classe.

Portion/masse : Les portions de masse sont des parties similaires entre elles mais aussi similaires à l'objet composite qu'elles composent. Exemples : Une part de tarte fait partie d'une tarte, les grains font partie du sel.

Substance/objet : Rapport qui existe entre un objet et ce dont il est fait, l'object perd son identité dès lors que l'un de ses constituants est supprimé. Exemples : L'hydrogène fait partie de l'eau, un vélo est partiellement composé d'acier.

Etape/événement : Cette relation définit une structure ou un agencement de composants prédéterminé. Les composants (étapes) interviennent à différents moments. Exemples : Une introduction fait partie d'une présentation, une matinée fait partie d'une journée, le débarquement de Normandie a fait partie de la Deuxième Guerre mondiale.

Phase/activité : Diffère de la relation précédente par le fait que les composants (phases) ne peuvent être séparés de l'activité. Exemples : Le paiement fait partie d'un achat, le banquet fait partie d'un mariage.

Lieu/espace : Traduit une composition spatiale et exprime la situation géographique d'un lieu. Les lieux ne sont pas séparables de l'espace dans lequl ils se trouvent. Exemples : Une oasis fait partie du désert, le mont Blanc fait partie des Alpes.

6.1.2 Identification des différents part-whole

Quatre critères sont utilisés pour différencier les sept types de relation *part-whole* :

La **fonctionnalité** exprime la dépendance fonctionnelle entre le composant et le composite. Ce critère indique que les parties sont dans une position spatiale ou temporelle spécifique qui est en relation avec leur fonction dans l'objet composite.

La **similarité** indique que les composants sont similaires entre eux et à l'objet composite auquel ils appartiennent.

La **séparabilité** s'intéresse à la séparation physique entre le composant et le composite.

La **simultanéité** indique que le composite a tous ses composants décrits en même temps.

Relations	Exemple	Fonctionnalité	Similarité	Séparabilité	Simultanéité
Composant/objet	roue/voiture	+	-	+	+
Elément/collection	arbre/forêt	-	-	+	+
Portion/masse	part/tarte	-	+	+	+
Substance/objet	hydrogène/eau	-	-	-	+
Etape/événement	analyse/cycle	+	-	+	-
Phase/activité	paiement/achat	+	-	-	-
Lieu/espace	oasis/désert	-	+	-	+

Tab. 6.1: Typologie des relations de composition

6.1.3 Transitivité de la relation part-whole

La propriété de transitivité d'une relation est importante dans tout processus d'inférence d'informations. Une relation de composition apparaît comme étant transitive : si A est une partie de B et B une partie de C, alors A est une partie de C. Cependant, en considérant les types de relations mis en évidence, nous pouvons en conclure que la relation est transitive lorsque les mêmes types de relation de composition sont impliquées dans les prémisses, et qu'il y a échec de la transitivé lorsque l'on compose deux relations de types différents, comme illustré à la figure 6.1

6.2 Application à notre modèle

6.2.1 Ajout des relations part-whole

Partant du modèle présenté dans le chapitre précédent (page 35), nous tenons à ajouter de l'information sémantique entre différents types de données. Nous définissons des rôles pour mettre en relation les termes, comme présenté à la figure 6.2. Pour satisfaire à la propriété de transitivité à l'intérieur d'un même type de *part-whole*, nous déclarons ces rôles comme étant transitifs. Ces sept rôles ont un super-rôle commun : *compose*. Le domaine et le rang sont *type*, comme pour le rôle non transitif *herite*.

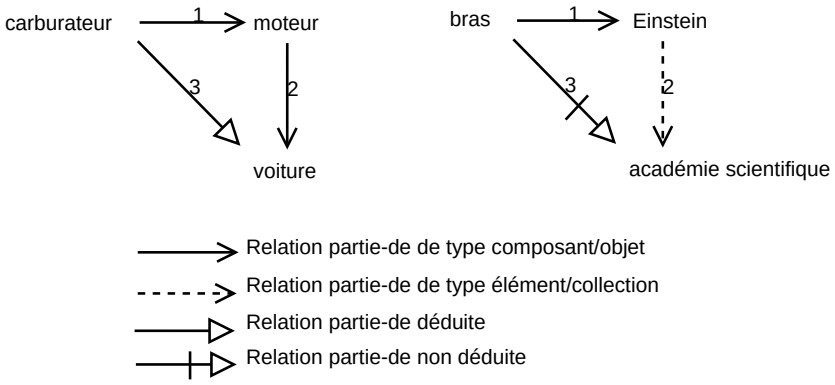

FIG. 6.1: Transitivité de la relation de composition. Les dessins se lisent de la façon suivante : partant des prémisses (1) et (2), est-il possible de conclure (3) ?

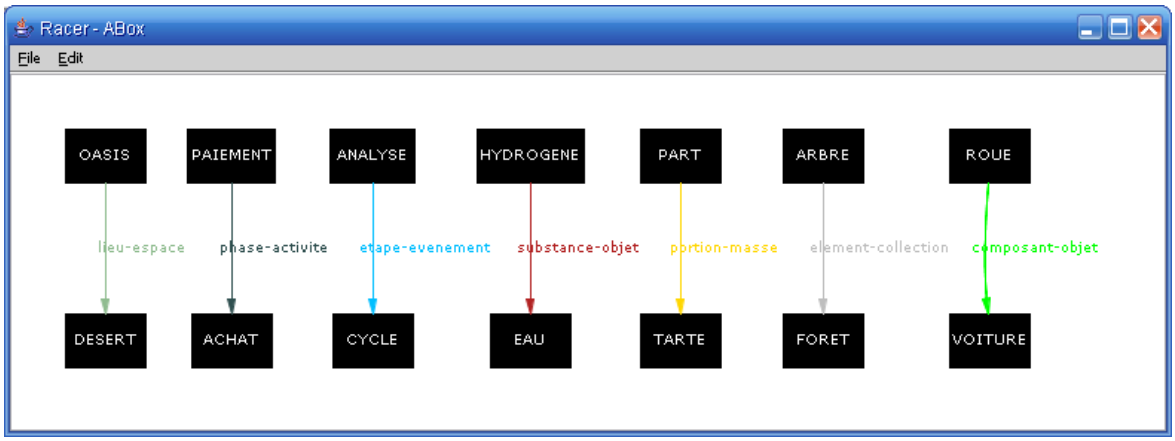

FIG. 6.2: Les types de part-whole

```
...
:roles( ...
  )
:transitive-roles(
        (compose :domain type :range type)
        (composant-objet :parents (compose))
        (element-collection :parents (compose))
        (portion-masse :parents (compose))
        (substance-objet :parents (compose))
        (etape-evenement :parents (compose))
        (phase-activite :parents (compose))
        (lieu-espace :parents (compose))
)
...
```

Exemples de requêtes

1. Pour savoir quels objets sont liés par une relation de composition, nous posons la question

```
(retrieve (?x ?y) (?x ?y compose))
```

et la réponse est :

```
((?X OASIS) (?Y DESERT)),
((?X PAIEMENT) (?Y ACHAT)),
((?X ANALYSE) (?Y CYCLE)),
((?X HYDROGENE) (?Y EAU)),
((?X PART) (?Y TARTE)),
((?X ARBRE) (?Y FORET)),
((?X ROUE) (?Y VOITURE))
```

2. La section 6.1.3 présente les deux conclusions importantes auxquelles les travaux précédents arrivent. Si nous décrivons les relations de composition d'une montre à la figure 6.3, alors à la question

```
(retrieve (?x) (?x montre composant-objet))
```

(quels sont les composants de l'objet montre), Racer répond

```
((?X BRACELET)),
((?X PETITE-AIGUILLE)),
((?X GRANDE-AIGUILLE)),
((?X CADRAN))
```

Ce qui respecte bien les deux principes : transitivité au sein d'un même type *part-whole* et non transitivité entre différents *part-whole*.

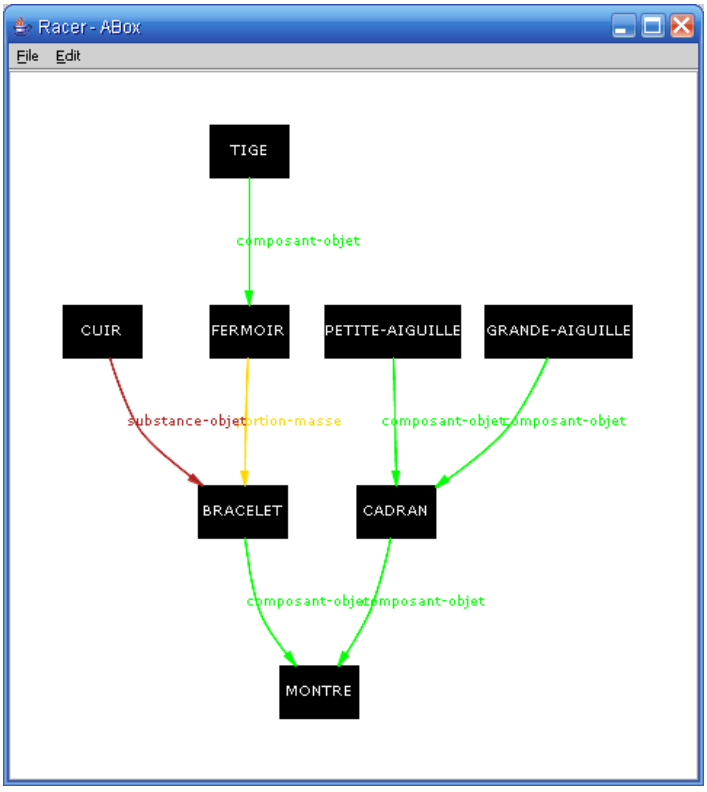

FIG. 6.3: Inférence sur rôle transitif

6.2.2 La question des relations directes

Si x est composé de y et y est composé de z, on peut en déduire que x est composé de z. Cependant, si le lien de composition de x vers z est explicitement indiqué, bien qu'il soit un lien de transitivité, il n'est pas considéré comme redondant. Il apporte l'information supplémentaire suivante : x est composé directement de z. Voilà pourquoi il est parfois intéressant d'obtenir les liens directs uniquement.

Cela n'est pas possible, étant donné la définition des *rôles transitifs* ($\mathcal{R}^+$) : si les deux paires IN_1 et IN_2 ; et IN_2 et IN_3 sont reliées par le rôle transitif R, alors IN_1 et IN_3 sont également liés par R.

Pour restreindre la réponse aux relations directes uniquement, il faut la *nettoyer* : par rapport à l'illustration proposée à la figure 6.4, il s'agirait de demander tous les descendants de B, C, et D lorsque nous avons obtenu les descendants de A, et ainsi, voyant que D est également un descendant de C, nous pouvons en conclure qu'il n'est pas directement lié à A. Seulement, quand la situation comme celle présentée à la figure 6.5 se présente, D sera considéré comme étant inféré alors qu'il ne l'est pas. Ce n'est donc pas une solution acceptable.

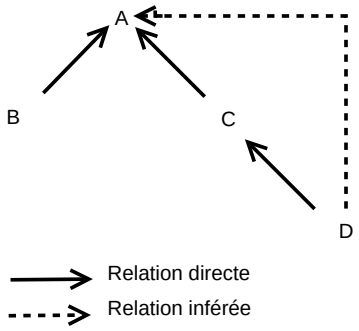

FIG. 6.4: Relation inférée

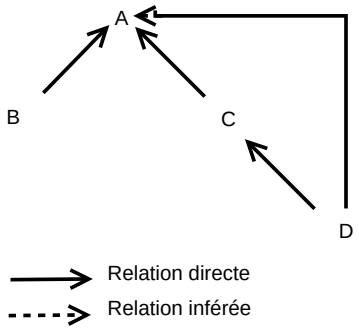

FIG. 6.5: Pas de relation inférée

En tenant compte du tableau 5.5 (page 47) d'héritage des propriétés de rôles, nous

constatons qu'il ne peut y avoir de relation *(super-rôle $\mathcal{R}$, sous-rôle $\mathcal{R}^+$)*. Par contre, un rôle transitif peut être le super-rôle d'un rôle normal. En redéfinissant notre modèle de la manière suivante, nous sommes à présent en mesure de répondre à des requêtes aussi bien sur des relations directes qu'inférées :

1. nous conservons les rôles transitifs déclarés (*composant-objet*, *etape-evenement*, etc.) ;
2. nous déclarons de nouveaux rôles (non transitifs) qui ont pour parent les rôles transitifs déclarés précédemment :

```
:roles(  ...
        (comp-obj-direct :parents (composant-objet))
        (elt-coll-direct :parents (element-collection))
        (port-mass-direct :parents (portion-masse))
        (subst-obj-direct :parents (substance-objet))
        (et-evt-direct :parents (etape-evenement))
        (ph-act-direct :parents (phase-activite))
        (l-esp-direct :parents (lieu-espace))
);
```

3. les relations entre deux instances sont les relations directes. Sur l'exemple de la montre, illustré à la figure 6.3, la relation *(related cadran montre composant-objet)* devient *(related cadran montre comp-obj-direct)* ;
4. ainsi, la requête *(retrieve (?x) (?x montre composant-objet))* renvoie toujours bien tous les composants de l'objet montre (à savoir *cadran*, *bracelet*, *petit-aiguille* et *grande-aiguille*), d'une manière transitive pour respecter la propriété de transitivité au sein d'un même type *part-of* ;
   tandis que la requête *(retrieve (?x) (?x montre comp-obj-direct))* répondra uniquement *cadran* et *bracelet*.

6.3 Contraintes sur les objets composites

Nous avons vu quels types *part-whole* existent et comment les identifier grâce aux quatre critères de fonctionnalité, similarité, simultanéite et séparabilité. En plus de cela, il existe certaines contraintes qui s'appliquent à la relation binaire entre composite et composant (c'est le lien de composition) : exclusivité/partage, dépendance/indépendance, prédominance/non prédominance et cardinalité.

Un lien de composition exprime qu'un objet composite est composé d'un autre objet composant (le mot objet désigne une instance). Le terme *lien de composition* désigne une relation binaire qui peut être soumise à certaines contraintes alors que le terme *relation de composition* est un terme générique qui désigne la réunion des différents liens de composition. La relation de composition est généralement définie comme étant irréflexive, antisymétrique et transitive.

6.3.1 Les propriétés des liens de composition

Des propriétés peuvent être attachées aux liens de composition, permettant ainsi d'enrichir la représentation des objets composites et de simplifier leur manipulation. Elles sont illustrées à la figure 6.6 page 71.

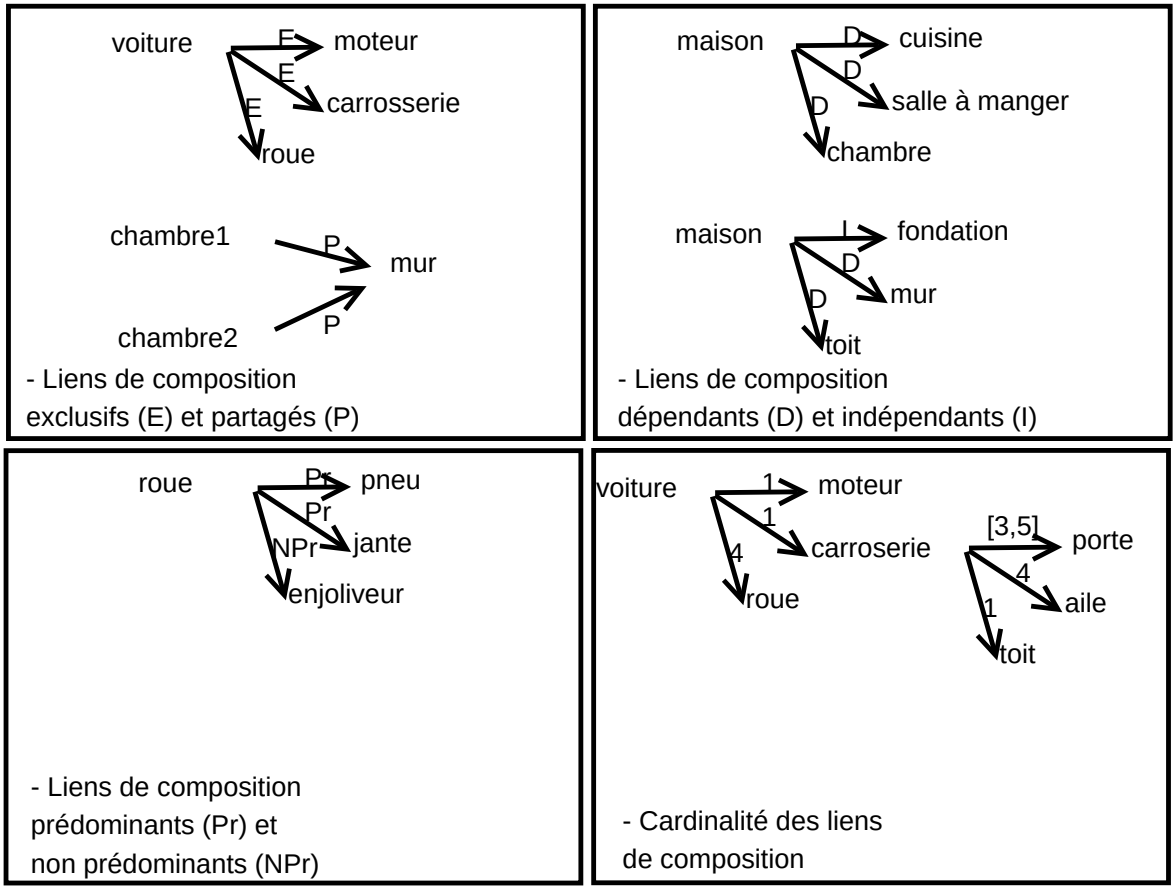

FIG. 6.6: Les différentes propriétés des liens part-whole

Exclusivité et partage

Un objet représenté par un lien de composition exclusif ne peut être référencé que par ce seul lien de composition et ne peut donc faire partie que d'un seul objet composite. Cette notion est souvent nécessaire pour la modélisation des objets physiques tels que le *moteur*, qui n'appartient qu'à une *voiture*.

À l'inverse, un composant référencé par un lien de composition partagé peut être référencé par un nombre quelconque de liens de composition partagés. Par exemple,

un même *mur* pour deux *chambres*. C'est une propriété utile pour la modélisation de hiérarchies de composition logiques.

Dépendance et indépendance

La dépendance permet de spécifier que l'existence d'un objet composant est liée à celle de l'objet composite. Dans ce cas, la destruction de l'objet composite entraîne automatiquement la destruction du composant.

Par exemple, les pièces *cuisine*, *salle à manger* ou encore *chambre* sont dépendantes de la *maison* mais les *fondations* en sont indépendantes.

Prédominance et non-prédominance

La prédominance spécifie que l'existence de l'objet composite est liée à celle du composant, c'est-à-dire que la destruction de l'objet composant entraîne la destruction de l'objet composite. Cette propriété traduit le fait que l'objet composite perd son identité lorsque le composant est détruit. Par exemple, *roue* a une relation prédominante avec *pneu* et *jante* mais pas avec *enjoliveur*.

Cardinalité

Il est possible d'associer une cardinalité à chaque lien de composition. La cardinalité d'un lien de composition indique le nombre d'objets référencés par ce lien de composition. Elle peut être fixe ou variable, par exemple une *voiture* a un *moteur*, une *carrosserie* et quatre *roues* ; tandis que la *carrosserie* peut avoir de trois à cinq *portes*.

Combinaison des propriétés

Par défaut, les liens de composition sont exclusifs, dépendants et non prédominants, cela correspond à des hiérarchies de composition physiques. Ils peuvent être partagés, indépendants et non prédominants afin de poser le minimum de contraintes. La cardinalité est égale à 1 par défaut.

6.3.2 Ajout des contraintes dans le modèle

Nous ajoutons des rôles pour définir les contraintes entre deux instances. Par exemple, entre *cadran* et *montre*, nous pouvons préciser qu'ils sont exclusifs, dépendants et non prédominants, comme illustré à la figure 6.7. Il est important de remarquer que là où les rôles de composition vont des *part* vers le *whole*, les propriétés s'appliquent

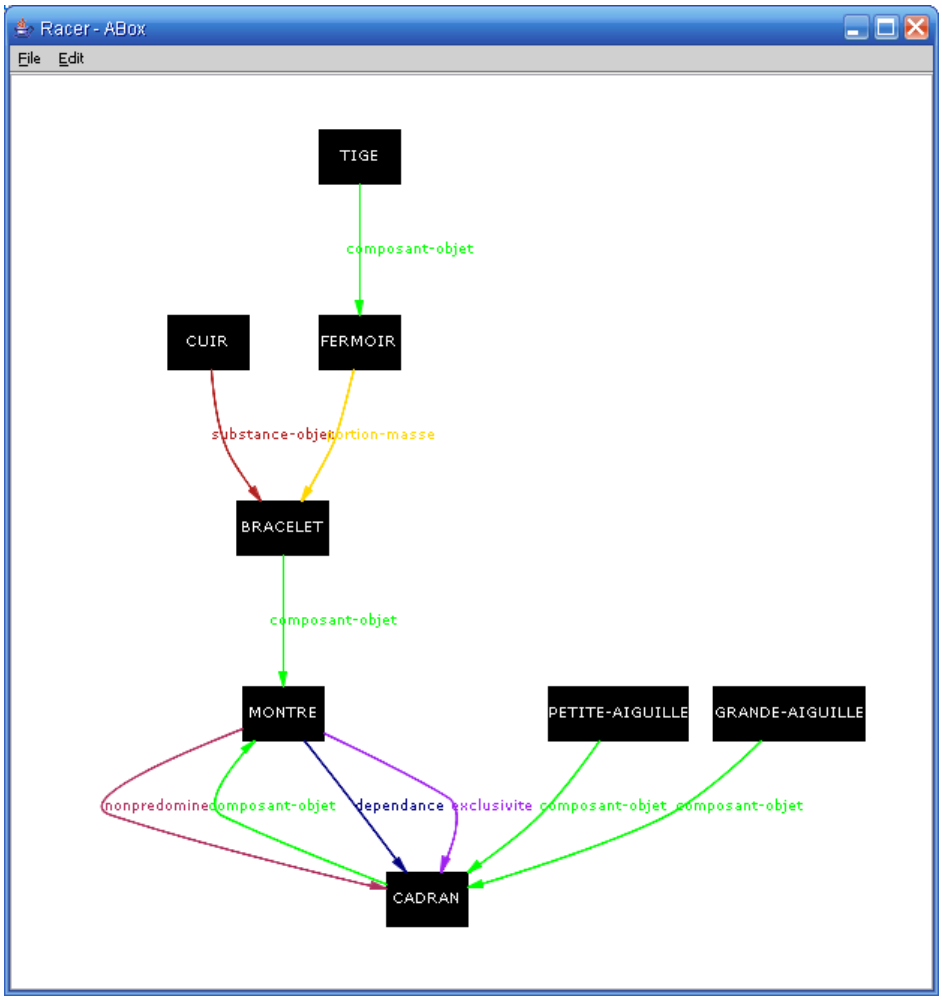

FIG. 6.7: Précision des propriétés

FIG. 6.8: Exemple de formulaire

en sens inverse. Pourquoi ? Afin de respecter le fait que c'est l'objet *whole* qui influe sur ses *part* (voir l'illustration 6.6 page 71).

Afin d'automatiser la définition des contraintes dans le modèle, la figure 6.8 suggère un formulaire et le pseudo-code qui permet la construction des commandes Racer est :

```
String requete;
if (choix-E-P = exclusivite)
 then requete = "(related "+noeud-source+" "+noeud-cible" exclusivite)"
 else requete = "(related "+noeud-source+" "+noeud-cible" partage)"
if (choix-D-I = dependance)
 then requete = requete+" (related "+noeud-source+" "+noeud-cible" dependance)"
 else requete = requete+" (related "+noeud-source+" "+noeud-cible" independance)"
if (choix-Pr-NPr = predominance)
 then requete = requete+" (related "+noeud-source+" "+noeud-cible" predominance)"
 else requete = requete+" (related "+noeud-source+" "+noeud-cible" nonpredomine)"
Send requete.
```

Nous conservons le mécanisme présenté à la section 6.2.2 afin d'obtenir les relations directes uniquement, ou les relations inférées également.

6.3.3 Utilité des contraintes

Exclusivité et partage

Avant de lier un composant à un composite, il faut vérifier qu'il n'est pas déjà le composant exclusif d'un autre composant :

```
(retrieve (?composite) (?composite le-composant exclusivite))
```

Si la réponse est `NIL` (ne signifie pas NON mais signifie que Racer ne trouve pas l'information, par opposition à la réponse `T` pour *true* ou à une réponse renvoyant d'autres instances), alors on peut lier le composant. Par exemple,

```
(retrieve (?composite) (?composite cadran exclusivite))
```

répond

```
((?COMPOSITE MONTRE))
```

donc nous ne pourrons pas lier *cadran* à un autre composite.

Dépendance et indépendance

Au moment de supprimer un objet, il faut s'assurer qu'il n'est pas le composite dont les composants sont dépendants, auquel cas il faudra les supprimer également :

```
(retrieve (?composant) (le-composite ?composant dependance))
```

Par exemple :

```
(retrieve (?composant) (montre ?composant dependance))
```

répond

```
((?COMPOSANT CADRAN))
```

donc il faut supprimer *cadran*.

Prédominance et non prédominance

Au moment de supprimer un objet, il faut s'assurer qu'il n'est pas le composant prédominant d'un objet composite, auquel cas il nous faudra le supprimer également :

```
(retrieve (?composite) (?composite le-composant predominance))
```

Par exemple, à supposer que `(related bracelet cuir predominance)`, la requête

```
(retrieve (?composite) (?composite cuir predominance))
```

renvoie

```
((?COMPOSITE BRACELET)),
```

bracelet est à supprimer.

6.3.4 La Cardinalité

La quatrième contrainte est la cardinalité. Par rapport à l'exemple de cardinalité présenté à la figure 6.6 page 71, nous souhaitons dire qu'une *voiture* est composée de quatre *roues* : *voiture* – {*cardinalité = 4*} →*roues*. Il faut une construction de la forme *cardinalité(voiture, roue, 4)*, or Racer ne permet pas les triplets. Pour les représenter, nous ajoutons un noeud intermédiaire auquel nous pouvons donner une valeur, comme illustré à la figure 6.9. *C* est un noeud unique à *A* et *B*, et le rôle *Lien* devient un triple rôles : *Lien-A-C*, *Lien-C-B* et *Lien-C-Valeur*.

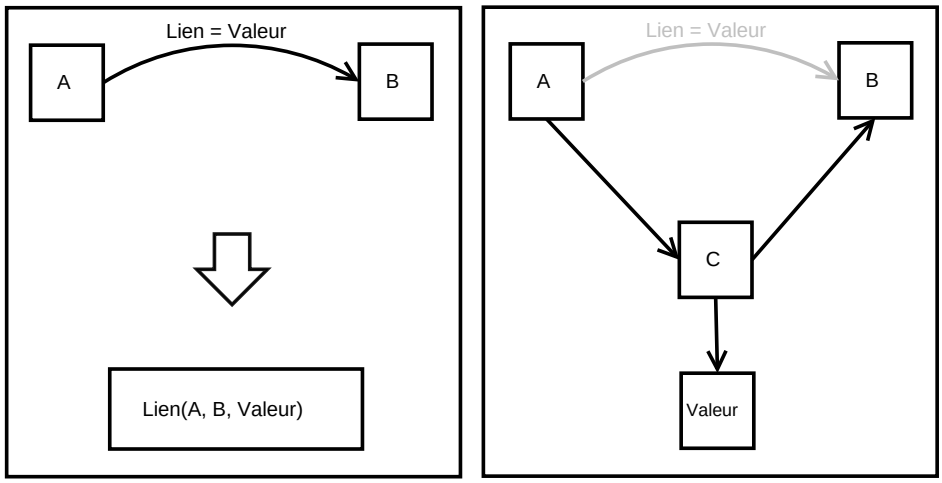

FIG. 6.9: Représenter un triplet

Dans Racer, il est possible de donner une valeur à C grâce aux attributs : on définit *Valeur* en tant qu'objet dans la signature *:objects(valeur)* et *Lien-C-Valeur* est un attribut *:attributes((integer lien-c-valeur))* ; on contraint l'individu C à *Valeur* par *(constrained c valeur lien-c-valeur)* et on peut attribuer la valeur, soit 4 : *(constraints (equal valeur 4))*.

Nous devons ajouter un noeud intermédiaire et ne pas simplement valuer A avec une règle du genre *Quand « Lien » alors regarder « Valeur »*, comme présenté à la figure 6.10. En effet, s'il devait y avoir plus d'un *Lien*, auquel s'attacherait la contrainte *Valeur* définie ?

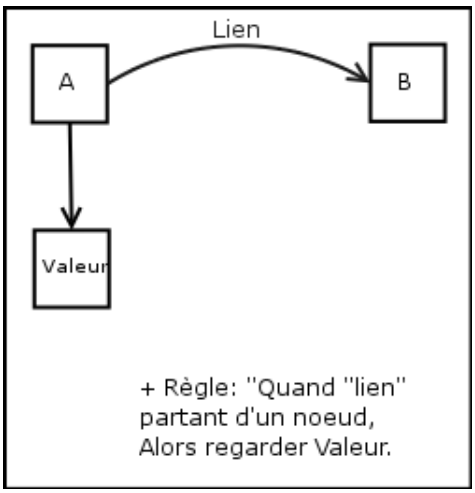

FIG. 6.10: Représenter un triplet - Alternative erronée

6.4 Comment poser des questions ?

6.4.1 Requêtes mono-critère

Input

« *Je recherche les services qui prennent en entrée une donnée de tel type.* »

Lorsque l'utilisateur pose cette question, une réponse pertinente et exhaustive doit contenir les éléments suivants :

1. la liste des services qui prennent directement en entrée une donnée de ce type, par la requête

```
retrieve (?services) (and (?services ?tds instancie)
                          (?tds ?tt realise)
                          (?tt la-donnee demande)));
```

2. la liste des autres services valides étant donné qu'ils prennent en entrée une donnée d'un type plus général (super-type) que le type demandé :

```
(retrieve (?services) (or (and (?services ?tds instancie)
                               (?tds ?tt realise)
                               (?tt ?donnee demande)
                               (?donnee le-type classifie))
                          (and (?services ?tds instancie)
                               (?tds ?tt realise)
                               (?tt ?donnee demande)
                               (?donnee ?surtype classifie)
                               (le-type ?surtype herite)));
```

3. si le type de données recherché est dans une relation de composition (`(retrieve (?x ?y) (?x ?y compose))`), alors apporter les informations supplémentaires suivantes :
   (a) s'il est un composant :
      i. indiquer le type de données qu'il compose,
      ii. préciser la nature de la relation *part-whole* (composant/objet, lieu/espace, etc.),
      iii. préciser les propriétés d'exclusivité, dépendance, prédominance et cardinalité,
      iv. donner les autres données composites ainsi que la nature du lien *part-of* et les propriétés pour chaque,
   (b) s'il est un composite :
      i. indiquer quels sont ses types de données composants,

ii. préciser la nature de chaque relation *part-whole* (composant/objet, lieu/espace, etc.),

iii. préciser les propriétés d'exclusivité, dépendance, prédominance et cardinalité pour chaque relation,

(c) toutes les informations s'obtiennent par la commande `(describe-individual <TYPE>)`. Par exemple, à l'illustration 6.7 page 73, si l'on demande `(describe-individual montre)`, nous obtiendrons les informations suivantes :

```
(MONTRE
:ASSERTIONS NIL
:ROLE-FILLERS (((INV COMPOSE) (BRACELET CADRAN))
                    ((INV COMPOSANT-OBJET) (BRACELET CADRAN))
               (PROPRIETE-PARTWHOLE (CADRAN)))
                    (NONPREDOMINE (CADRAN))
                    (DEPENDANCE (CADRAN))
                    (EXCLUSIVITE (CADRAN))
:TOLD-ATTRIBUTE-FILLERS NIL
:DIRECT-TYPES ((TYPE)));
```

(d) en accompagnement à toutes ces explications, en donner la signification et donner des indications sur comment relancer une recherche sur les nouveaux éléments obtenus.

Output

« *Je recherche les services qui renvoient en sortie une donnée de tel type.* »

Même procédure.

Traitement

« *Je recherche les services dont le type réalise tel traitement.* »

Lorsque l'utilisateur pose cette question, une réponse pertinente et exhaustive doit contenir les éléments suivants :

1. la liste des services dont le type réalise directement ce traitement ;
2. la liste des autres services valides étant donné que leur type réalise un traitement dont le traitement demandé est une spécialisation. Tous les super-traitements à celui demandé n'ont pas forcément de service associé. La requête est

```
(retrieve (?services) ( or (and (?services ?tds instancie)
                                (?tds le-traitement realise))
                           (and (?services ?tds instancie)
```

```
                                    (?tds ?sur-traitement realise)
                                    (le-traitement ?sur-traitement
                                     specialise)));
```

3. la liste des services pouvant l'intéresser étant donné que leur type réalise un traitement qui spécialise le traitement demandé, par la requête

```
(retrieve (?services) ( or (and (?services ?tds instancie)
                                (?tds le-traitement realise))
                           (and (?services ?tds instancie)
                                (?tds ?sous-traitement realise)
                                (?sous-traitement le-traitement
                                 specialise)));
```

4. si le traitement voulu est dans une relation de composition, alors même procédure que pour les inputs.

6.4.2 Requête multi-critères

Une requête multi-critères sera de la forme : « *Je souhaite les services implémentés par tel binding ou réalisant telle opération et prenant en entrée une donnée de tel type et réalisant tel traitement* »

Construction de la requête

Le système d'encodage de la requête multi-critères doit permettre de faire des constructions de la forme *et/ou*. Il offre en outre la possibilité de parenthésage, ou bien il doit être explicite sur la manière dont sera construite la requête finale envoyée à Racer concernant l'ordre d'évaluation des différents éléments.

Présentation de la réponse

1. la liste des services qui satisfont directement à la requête multi-critères ;
2. la liste des services qui satisfont à une requête enrichie :
   (a) on crée une seule requête enrichie qui fait directement appel aux sur-types des inputs demandés, aux sous-types des outputs demandés et aux sur-traitement des traitements demandés,
   (b) dans un souci de détail de l'information renvoyée à l'utilisateur, on créera plusieurs requêtes (autant qu'il n'y a de critères sur lesquels peut s'appliquer une hiérarchie ou une composition, à savoir sur les entrées, les sorties et les traitements) et pour chacune, la réponse contiendra les éléments présentés précédemment pour les requêtes mono-critère.

6.5 Conclusion

Dans ce chapitre, nous avons étudié la relation *part-whole* et avons appliqué les éléments mis en évidence à notre modèle. Cela concerne les sept types de relation *part-whole* auxquels s'appliquent quatre critères, ainsi que les contraintes sur les objets qui ajoutent des règles sur la gestion de l'ontologie (vérifier qu'un composant est effectivement exclusif à son objet, gérer la suppression d'objets et valider les cardinalités).

Nous avons vu comment traiter les relations directes uniquement, ou bien comment inférer de nouvelles relations grâce à la transitivité sur les rôles. Nous avons mis en évidence deux propriétés importantes auxquelles les différentes recherches aboutissent : il existe une transitivité au sein d'un même type de relation *part-whole* et il n'existe pas de transitivité entre deux types différents. Dans le prochain chapitre, nous allons passer outre cette dernière propriété et proposer une manière de composer les types entre eux.

Nous n'avons pas encore exploité les quatre critères de fonctionnalité, similarité, simultanéité et séparabilité qui définissent les sept relations *part-whole*. Ces critères seront de première importance dans le chapitre suivant.

Sur base de l'étude présentée dans ce chapitre, nous avons proposé une manière d'établir des réponses aux critères de recherche que l'utilisateur encode. Nous n'exploitons cependant pas encore entièrement la relation *part-whole* pour la découverte de services, cela sera fait dans le prochain chapitre.

Chapitre 7

Inférence supplémentaire

Ce chapitre propose une étude approfondie qui est en rapport avec les concepts étudiés dans les deux chapitres précédents. Par rapport au chapitre 5, nous allons continuer l'évaluation de la hiérarchie des données qui est à présent enrichie du *part-whole* étudié en détail dans le chapitre 6.

Après deux remarques préalables, nous allons composer les relations *part-whole* entre elles et voir comment en exploiter les résultats dans notre modèle. Nous proposons un tableau de composition ainsi qu'un algorithme qui permet de comprendre comment parcourir l'ontologie pour l'enrichir des nouvelles relations découvertes dans le tableau de composition.

Ensuite, nous enrichissons la réponse que nous avions précédemment étudiée aux sections 5.5 et 6.4 en nous demandant comment exploiter la relation *part-whole* afin d'élargir le spectre des services proposés à l'utilisateur. Pour cela, nous utilisons les quatre critères qui définissent les différents types *part-of*, ce sont la fonctionnalité, la similarité, la simultanéité et la séparabilité.

7.1 Remarques préalables

7.1.1 Le raisonnement sur instances

Dans la littérature ([AFGP95] et [Ber95]), les chercheurs redéfinissent une syntaxe et une sémantique supplémentaires à $\mathcal{AL}$ et $\mathcal{SHIQ}$, car ces langages ne permettent que la relation $\sqsubseteq$ (*is-a*) qui caractérise l'héritage. Par exemple, dans le modèle, *Precondition* $\sqsubseteq$ *Condition* signifie que toute instance de *Precondition* est également une instance de *Condition*. Au niveau DL, les chercheurs étudient donc la relation *part-whole* entre concepts (pour rappel, les concepts équivalent aux types d'entité du modèle entité-association, ou aux classes OWL) et y ajoutent souvent la relation $\preceq$ qui est le rôle

has-part. Par exemple, le langage $\mathcal{P}$ de Sattler qui définit différents types de relations *part-whole* et leur inverse.

Le modèle Bigre formalise le fonctionnement des services bioinformatiques (par exemple : les opérations s'organisent en scénarios et ont des pré/postconditions ; les inputs des traitements sont des inputs d'opérations ; les types de données classifient les données ; etc.) tout en restant général par rapport au domaine représenté. Une fois ce modèle instancié par les fournisseurs de services bioinformatiques (au moment de l'enregistrement de leurs services), l'ontologie sera complétée. Les types d'association *héritage* et *specialisation* traduisent de l'héritage *is-a* au sein des instances. De même, nous traduisons le lien *part-of* par rapport aux instances et non aux concepts.

Au niveau instances, le mécanisme d'inférence correspondant au $\sqsubseteq$ qui s'applique aux rôles se traduit par la transitivité des rôles. C'est le seul mécanisme dont nous pouvons profiter. Par exemple, nous ne pouvons pas profiter de la cardinalité comme nous pourrions le faire au niveau des concepts ; à la place, nous devons ajouter nos propres mécanismes tels que celui présenté au point 6.3.4 pour la cardinalité.

7.1.2 Représenter le part-whole en OWL

OWL ne contient pas de primitives spécifiques pour les relations *part-whole*, contrairement à la relation *subClass* (est une sous-classe de) par exemple. Il est toutefois possible de les représenter d'une manière similaire à celle présentée dans notre analyse[1].

Concernant la propriété de transitivité au sein d'un même type *part-of* que nous avons étudiée, il y a moyen de déclarer une propriété comme étant transitive en OWL. Si A compose B et B compose C, alors A compose C et les parties de C incluent A et B. On définira donc la transitivité des propriétés de la même manière que celle vue au point 6.2.2 : *partOf_directly* est une sous propriété non transitive de *partOf*.

En DL, lorsque l'on définit une relation entre a et b, alors on peut, en partant de b, retrouver a via l'inverse du rôle. Les rôles correspondent aux propriétés OWL, à la différence qu'il n'y a pas moyen de retrouver a à partir de b dans OWL. Il est possible de définir la propriété *hasPart* comme étant l'inverse de *partOf*. Cependant, les raisonneurs OWL ne se comportent pas bien pour de grandes hiérarchies *part-whole* connectées à la fois par *partOf* et *hasPart*. Il est conseillé de choisir un sens unique, *partOf* par exemple.

7.2 Transitivité inter-types

Supposons que l'utilisateur recherche les services de transport de matériel pour véhiculer une cargaison de Belgique en Espagne. Il souhaite que le contenu de sa cargaison

[1] `http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/`

soit assurée. Le raisonneur trouve un service de location de camions, sur lequel existe une relation d'assurance. Le composite est le *camion*, dont le *contenu* est un composant de type composant/objet (car il a pour fonction de contenir, il est séparable et simultané). De plus, il y a une relation entre *assurance* et *camion*. Est-ce que la propriété d'assurance se répercute sur le composant *contenu* de l'objet *camion* ?

Nous remarquons sur cet exemple que la propriété de transitivité au sein d'un même type de relation *part-whole* n'est pas suffisante pour permettre au raisonneur de faire des inférences intéressantes.

Nous allons donc étudier la transitivité entre relations *part-whole* différentes, cela nous permettra de modéliser de nouvelles inférences.

7.2.1 Récapitulatif des relations part-whole

Nous commençons par rappeler la définition de chaque relation *part-whole*, en mettant l'accent sur les spécificités qu'elles présentent. Pour ce faire, chaque définition est incrémentale par rapport à une précédente, et avec pour point de départ la relation composant/objet, car elle est la relation de composition la plus utilisée en ingénierie de la connaissance.

Ce récapitulatif est proposé en annexe libre A, afin que le lecteur puisse s'en servir tout au long de ce chapitre qui revient souvent sur les définitions et critères des différents types.

7.2.2 Le tableau de composition initial

Nous souhaitons construire un tableau de composition des types *part-whole* dans le but de modéliser de nouvelles inférences. Afin de valider notre investigation, nous sommes parti d'une étude similaire [Sat95] où l'auteur envisage un tel tableau. Toutefois, les différents types de relations *part-whole* repris dans son tableau ne sont pas tous semblables à ceux que nous étudions ici. Nous commençons donc par ramener ce tableau de composition à nos « normes ». Pour ce faire, nous avons comparé les définitions données par Sattler aux nôtres, et ramené ses concepts à ceux que nous avons définis. Il propose 6 types de relations *part-whole*, dont 4 sont facilement identifiables :

1. **composant/composite** où le composant a des frontières *part-whole* inhérentes et où le *whole* est non homogène. Ce type correspond à notre type **composant/objet** ;
2. **membre/collection**, pour lequel les *parts* sont définies à travers le *whole* et ne sont pas fixées localement, il s'agit de **élément/collection** ;
3. **quantité/masse** où le tout n'a pas de frontières ni de volume ; le *part* est quant à lui limité. Cela correspond au type **portion/masse** ;

4. **substance/objet** correspond au type du même nom.

Les deux autres types sont plus difficiles à rapprocher de nos définitions, il s'agit de :

1. **segment/entité** est un type où les frontières *part-whole* sont arbitraires ; par exemple l'avant de la voiture par rapport à la voiture ;
2. **ingrédient/objet** a pour définition que le *part* et le *whole* sont spatialement inséparables. Par exemple, 2 kilos d'acier par rapport au vélo.

Nous mettons ces deux types de côté et nous concentrons sur les 4 types pour lesquels il n'y a aucune ambiguïté de concept. Par rapport au tableau de référence, nous apportons de plus une correction quant à sa diagonale, car de nombreux travaux sont arrivés à la conclusion qu'il y a transitivité au sein d'un même type de relation *part-whole*. Parmi les 4 types, **portion/masse** n'est pas repris en prémisse de tableau, mais il intervient dans la composition de **segment/entité** et **ingrédient/objet** que nous n'étudions pas ici. Cela nous amène à un tableau 4×3 pour lequel les relations sont clairement définies. Sattler ne justifie pas la construction de son tableau, nous le validons donc par des exemples et nous devrons ensuite déterminer notre propre méthode afin de l'étendre.

où	a C/O $b \wedge$ $b\ \omega\ c \Rightarrow$	a E/C $b \wedge$ $b\ \omega\ c \Rightarrow$	a S/O $b \wedge$ $b\ \omega\ c \Rightarrow$
ω est C/O (composant/objet)	a C/O c	a C/O c	a S/O c
ω est E/C (élément/collection)	rien	a E/C c	a S/O c
ω est P/M (portion/Masse)	impossible	a E/C c	a S/O c
ω est S/O (substance/Objet)	impossible	rien	a S/O c

Tab. 7.1: Table de composition inter-types

Le tableau 7.1 présente la transitivité inter-types *part-whole*. Il faut le lire de la manière suivante :

- pour le premier élément, on a « a C/O $b \wedge b\ \omega\ c \Rightarrow a$ C/O c », c'est-à-dire que si a et b sont liés par une relation composant/objet et que b et c sont liés par ω, où ω est la relation composant/objet, alors a et c sont liés par la relation composant/objet ;
- « rien » signifie qu'on ne sait rien dire entre les éléments a et c lorsqu'ils sont chacun liés à l'élément commun b ;
- « impossible » signifie qu'il est impossible que b soit un élément commun à a et c car il ne peut exister de situation où la première relation soit prémisse de la seconde.

Nous étudions ce tableau colonne par colonne :

1. si a est le composant de l'objet b et que
   (a) b est le composant de l'objet c, alors a est le composant de l'objet c. Par exemple, un *boulon* du *moteur* est également un composant de la *voiture*,
   (b) b est l'élément de la collection c, alors nous ne pouvons rien dire entre a et c. En effet, le *bras* compose le *musicien* qui est un élément de l'*orchestre*, mais nous ne pouvons pas dire que le *bras* soit un composant ni même un élément de l'*orchestre*,
   (c) il est impossible d'avoir un objet composite b qui soit également une portion de la masse c. La masse est homogène, ses éléments sont similaires, tandis que l'objet est non homogène et surtout, le composant a une fonction pour cet objet là, mais aucune fonction en dehors,
   (d) il est impossible d'avoir un objet composite b qui soit également une substance de l'objet c. Il n'y a aucune frontière entre substance et objet car l'objet est fait de sa substance et perd son identité si la substance est supprimée, tandis que les frontières sont bien définies dans une relation composant/objet (critère de séparabilité) ;
2. si a est l'élément de la collection b et que
   (a) b est le composant de l'objet c, alors a en est aussi le composant. Par exemple, le *musicien* est un élément de la collection *orchestre*, et l'*orchestre* est un composant de l'objet *opéra* (il a une fonction dans l'*opéra*, il en est séparable et il est simultané), alors le *musicien* est également composant de l'*opéra*,
   (b) b est l'élément de la collection c, alors a est l'élément de la collection c. Ceci provient de la propriété de transitivité intra-type,
   (c) b est la portion de la masse c, alors a est un élément de la collection c. Une portion étant similaire à sa masse, alors l'élément de cette portion ne peut être qu'un élément de sa masse. La collection *orchestre* est une portion de la masse *ensemble des orchestres*, le *musicien* est bien un élément de l'*ensemble des orchestre*,
   (d) b est la substance de l'objet c, alors on ne peut rien conclure entre a et c. La relation n'est pas impossible même si a est séparable de b qui lui n'est pas séparable de c (sinon, c perd son identité). Par exemple, l'*arbre* est séparable de la *forêt* mais, à supposer que la *forêt* soit une substance du *paysage*, alors retirer un *arbre* ne modifiant pas le fait que ce soit une *forêt*, le *paysage* existe toujours, la relation n'est donc pas impossible. Et l'on ne sait rien dire entre l'*arbre* et le *paysage* ;

3. si a est la substance de l'objet b, alors a sera la substance de l'objet c quel que soit le type de relation *part-whole* entre b et c
   (a) quand b est un composant de c : le *cuir* est la substance du l'objet *siège* qui est un composant de la *voiture*, alors le *cuir* est une substance de la *voiture*. En effet, la relation composant/objet a les critères de fonctionnalité (le *siège* permet de s'assoir dans la *voiture*), de séparabilité (on peut le retirer) et de simultanéité (on n'utilise la *voiture* que si l'on a un *siège* pour pouvoir la conduire) ; la relation substance/objet a l'unique critère de simultanéité : supprimer le *cuir* fait perdre son identité au *siège*, et donc à la *voiture*, il en est donc bien une substance,
   (b) quand b est un membre de la collection c, comme l'*arbre* pour la *forêt*, avec le fait que le *bois* soit la substance de l'*arbre*, il est également la substance de la *forêt*. Sans *bois*, pas d'*arbres*, et donc pas de *forêt*,
   (c) quand b est la portion de la masse c, comme la *part* de la *tarte*, alors il est également évident que la substance de la *part*, *farine* par exemple, soit substance de la *tarte*,
   (d) quand b est substance de l'objet c, alors la transitivité inter-type est d'application.

7.2.3 Le tableau de composition par ses critères

Nous souhaitons élargir les résultats à toute combinaison des 7 types *part-whole* initiaux. Comme les 4 types du tableau précédent ont été ramenés sans ambiguïté possible à nos types, alors nous pouvons poser l'hypothèse que les critères de fonctionnalité, similarité, séparabilité et simultanéité s'appliquent également. Pouvons-nous exploiter ces critères ?

La liste qui suit est une traduction du tableau 7.1 et se comprend ainsi : nous lisons le tableau colonne par colonne et chaque relation (composant/objet, élément/collection, etc.) est représentée par ses critères positifs, [fonct,sépa,simu] par exemple. La forme globale est :

$$[\textit{relation entre a et b}] \wedge [\textit{relation entre b et c}] \Rightarrow [\text{relation entre a et c}].$$

1. [fonct,sépa,simu] $\wedge$
   (a) [fonct,sépa,simu] $\Rightarrow$ [fonct,sépa,simu]
   (b) [sépa,simu] $\Rightarrow$ [rien]
   (c) [simi,sépa,simu] $\Rightarrow$ [impossible]
   (d) [simu] $\Rightarrow$ [impossible]

2. [sépa,simu] ∧
   (a) [fonct,sépa,simu] ⇒ [fonct,sépa,simu]
   (b) [sépa,simu] ⇒ [sépa,simu]
   (c) [simi,sépa,simu] ⇒ [sépa,simu]
   (d) [simu] ⇒ [rien]
3. [simu] ∧
   (a) [fonct,sépa,simu] ⇒ [simu]
   (b) [sépa,simu] ⇒ [simu]
   (c) [simi,sépa,simu] ⇒ [simu]
   (d) [simu] ⇒ [simu]

Après avoir analysé ces 12 relations entre les critères, nous admettons qu'elles ne sont pas suffisantes pour en dériver des règles pour aider à la constitution d'un tableau de composition de tous les types (tableau 7 × 7) mais elles donnent certaines pistes. Nous choisissons donc une approche moins formelle en partant d'exemples étudiés sous leur aspect critères.

Il nous reste 3 types à composer avec les 4 étudiés ci-dessus. Contrairement à ceux-ci, étape/événement et phase/activité sont des relations qui s'appliquent à des concepts et non à des biens matériels. Si l'on prend l'exemple du *débarquement de Normandie* qui est une étape de l'événement *Deuxième Guerre mondiale*, il s'agit bien de quelque chose d'immatériel, à savoir qu'on ne peut pas prendre le *débarquement* et le manipuler par exemple. Peut-on composer ces relations à du matériel ? En prémisse (c'est-à-dire entre a et b), non, car il est difficilement concevable qu'un événement soit la substance d'un objet par exemple.

Pour étape/événement entre b et c, et pour les relations suivantes entre a et b :
- il est possible que l'étape *débarquement de Normandie* soit un « objet » composé de *fusils*. En effet, le *fusil* a une fonction dans le *débarquement*, il est séparable de l'événement et lui est simultané. La relation est donc bien composant/objet. Alors, les *fusils* composent également « l'objet » *Deuxième Guerre mondiale* ;
- il est impossible qu'une collection soit directement l'étape d'un événement, par exemple, le *soldat* n'est pas un élément de la collection *débarquement*, il est un élément de la collection *armée* mais une *armée* ne peut être une étape. Elle ne peut être que le composant de l'étape *débarquement* ;
- de même, il est impossible qu'une masse soit une étape ;
- c'est impossible également pour substance/objet ;
- pour la relation phase/activité entre a et b, alors a est aussi une étape de l'événement c. En effet, a n'est pas séparable de b qui lui-même a une fonction dans c mais en est séparable, donc a qui a une fonction dans b a également une fonction dans c et en est séparable.

Phase/activité se rapproche très fort de étape/événement, il ne peut non plus être prémisse d'un des 4 types déjà étudiés, et s'il lie b et c, alors :

- si a est la *carte de crédit* qui sert à l'objet *paiement* qui est une phase de l'*achat*, alors la *carte de crédit* est également le composant de l'objet *achat* ;
- il est impossible qu'une collection, une masse, ou un objet (objet de la relation substance/objet) soit une phase ;
- si a est une étape de l'événement b, alors a a une fonction dans b mais en est séparable, avec b qui n'est pas séparable de c, alors on ne sait rien dire entre a et c.

Concernant la relation portion/masse entre a et b,

- une masse peut très certainement être le composant d'un objet, mais quelle peut-être la relation entre a et c ? Ce ne peut être une portion car elle n'est pas similaire à l'objet. Raisonnons sur un exemple, la portion pourrait être le *grain de sable* qui appartient au *sable* qui est un composant du *bac à sable*. Alors, le *grain de sable* est également un composant du *bac à sable* ;
- si une masse est l'élément d'une collection, alors, comme la portion est similaire à sa masse, elle est également élément de la collection. Par exemple, la *tarte* est un élément de la collection *pâtisserie*, alors la *part* de la *tarte* est aussi élément de cette collection ;
- pour la composition de la relation élément/collection avec substance/objet, nous avons vu que l'on ne sait rien dire entre a et c. La relation portion/masse a pour unique différence par rapport à élément/collection que les portions sont similaires entre elles et par rapport à la masse. Il n'est donc pas plus possible de conclure pour la composition de portion/masse et substance/objet.

Il reste à étudier lieu/espace qui sert par définition à des compositions spatiales, afin d'exprimer une situation géographique. Pour une relation lieu/espace entre b et c :

- si a est le composant de l'objet b,b serait un objet qui est le lieu d'un espace c, or b et c sont similaires, non séparables et a est fonctionnel à b. Ceci est impossible ;
- si l'*arbre* est l'élément de la collection *forêt*, et que *forêt* est un lieu de l'espace *Amazonie*, alors l'*arbre* est aussi un élément de l'*Amazonie* car elle est similaire, simultanée et non séparable de la *forêt* ;
- une masse peut-elle être le lieu d'un espace ? Oui, l'*oasis*, qui est un lieu de l'espace désert, est la masse dont l'*eau* est une part ; comme *eau* est séparable de *oasis*, elle l'est également de *désert*, la relation est donc portion/masse ;
- la substance d'un objet qui est le lieu d'un espace est également la substance de cet espace ;
- un événement, ainsi qu'une phase ne peuvent être les lieux d'un espace.

Pour la relation lieu/espace en prémisse, c'est-à-dire entre a et b,

- si b est le composant de l'objet c, alors a est aussi le composant de l'objet c vu que a est simultané, similaire, non séparable et non fonctionnel par rapport à b ;
- si b est le membre de la collection c, alors on ne peut rien dire entre a et c. Par exemple, pour le lieu *racine* de l'espace *arbre* qui est un élément de la collection*forêt*, on ne sait rien dire entre *racine* et *forêt*, tout comme on ne savait rien dire entre le *bras* du *musicien* et l'*orchestre* pour la relation composant/objet ;
- quand b est la portion de la masse c, alors a est forcément portion de cette masse c également ;
- il est impossible qu'un espace soit la substance d'un objet c ;
- il est impossible qu'une relation lieu/espace soit la prémisse d'une relation étape/événement ou phase/activité.

Le tableau 7.2 synthétise les résultats obtenus et le tableau 7.3 le traduit sous sa forme *types de relations*.

↙		C/O [fonct,sépa,simu]	E/C [sépa,simu]
C/O	[fonct,sépa,simu]	[fonct,sépa,simu]	[fonct,sépa,simu]
E/C	[sépa,simu]	[rien]	[sépa,simu]
P/M	[simi,sépa,simu]	[impossible]	[sépa,simu]
S/O	[simu]	[impossible]	[rien]
E/E	[fonct,sépa]	[fonct,sépa,simu]	[impossible]
P/A	[fonct]	[fonct,sépa,simu]	[impossible]
L/E	[simi,simu]	[impossible]	[sépa,simu]

↙		P/M [simi,sépa,simu]	S/O [simu]
C/O	[fonct,sépa,simu]	[fonct,sépa,simu]	[simu]
E/C	[sépa,simu]	[sépa,simu]	[simu]
P/M	[simi,sépa,simu]	[simi,sépa,simu]	[simu]
S/O	[simu]	[rien]	[simu]
E/E	[fonct,sépa]	[impossible]	[impossible]
P/A	[fonct]	[impossible]	[impossible]
L/E	[simi,simu]	[simi,sépa,simu]	[simu]

↙		E/E [fonct,sépa]	P/A [fonct]
C/O	[fonct,sépa,simu]	[impossible]	[impossible]
E/C	[sépa,simu]	[impossible]	[impossible]
P/M	[simi,sépa,simu]	[impossible]	[impossible]
S/O	[simu]	[impossible]	[impossible]
E/E	[fonct,sépa]	[fonct,sépa]	[fonct,sépa]
P/A	[fonct]	[rien]	[fonct]
L/E	[simi,simu]	[impossible]	[impossible]

↙		L/E [simi,simu]
C/O	[fonct,sépa,simu]	[fonct,sépa,simu]
E/C	[sépa,simu]	[rien]
P/M	[simi,sépa,simu]	[simi,sépa,simu]
S/O	[simu]	[impossible]
E/E	[fonct,sépa]	[impossible]
P/A	[fonct]	[impossible]
L/E	[simi,simu]	[simi,simu]

Tab. 7.2: La table de composition des relations part-whole

où	a C/O b $b\ \omega\ c \Rightarrow$	a E/C b $b\ \omega\ c \Rightarrow$	a P/M b $b\ \omega\ c \Rightarrow$	a S/O b $b\ \omega\ c \Rightarrow$	a E/E b $b\ \omega\ c \Rightarrow$	a P/A b $b\ \omega\ c \Rightarrow$	a L/E b $b\ \omega\ c$
ω est C/O	C/O	C/O	C/O	S/O	imp.	imp.	C/O
ω est E/C	rien	E/C	E/C	S/O	imp.	imp.	rien
ω est P/M	imp.	E/C	P/M	S/O	imp.	imp.	P/M
ω est S/O	imp.	rien	rien	S/O	imp.	imp.	imp.
ω est E/E	C/O	imp.	imp.	imp.	E/E	E/E	imp.
ω est P/A	C/O	imp.	imp.	imp.	rien	P/A	imp.
ω est L/E	imp.	E/C	P/M	S/O	imp.	imp.	L/E

Tab. 7.3: Traduction de la table de composition en ses types

7.3 Application au modèle

Dans la section précédente, nous avons compris que la propriété de transitivité au sein d'un même type de relation *part-whole* ne permettait pas d'exploiter l'ontologie de manière suffisante. Nous avons donc déterminé comment composer les différentes relations *part-whole* entre elles. Maintenant, il nous faut mettre à profit ces résultats afin d'améliorer la recherche de services et d'enrichir la réponse renvoyée à l'utilisateur.

Dans un premier temps, nous devons savoir quels objets de l'ontologie nous pouvons composer pour ajouter la nouvelle information qui est la relation *part-whole* évaluée par rapport aux relations *part-whole* déclarées. Nous voyons ensuite de quelle manière nous pouvons procéder pour faire cet ajout d'information, nous proposons un algorithme pour cela. Nous nous demandons également quand est-il recommandé de procéder à cet ajout de relations dans l'ontologie.

7.3.1 Quels objets pouvons-nous composer ?

Les relations directes. Lorsque a compose directement b qui compose c, alors on peut regarder quelle relation *part-whole* existe entre a et c.

Les relations inférées. Lorsque a compose indirectement d (via b, relation inférée par la transitivité intra-type) qui compose e , alors on peut regarder quelle relation existe entre a et e.

Les relations évaluées. A partir du moment où nous avons défini la relation entre a et c et entre a et e, alors nous pouvons aussi regarder les relations qui partent de c et e afin de voir s'il est possible de les composer avec a.

Illustration. Nous illustrons ces différentes compositions à la figure 7.1.

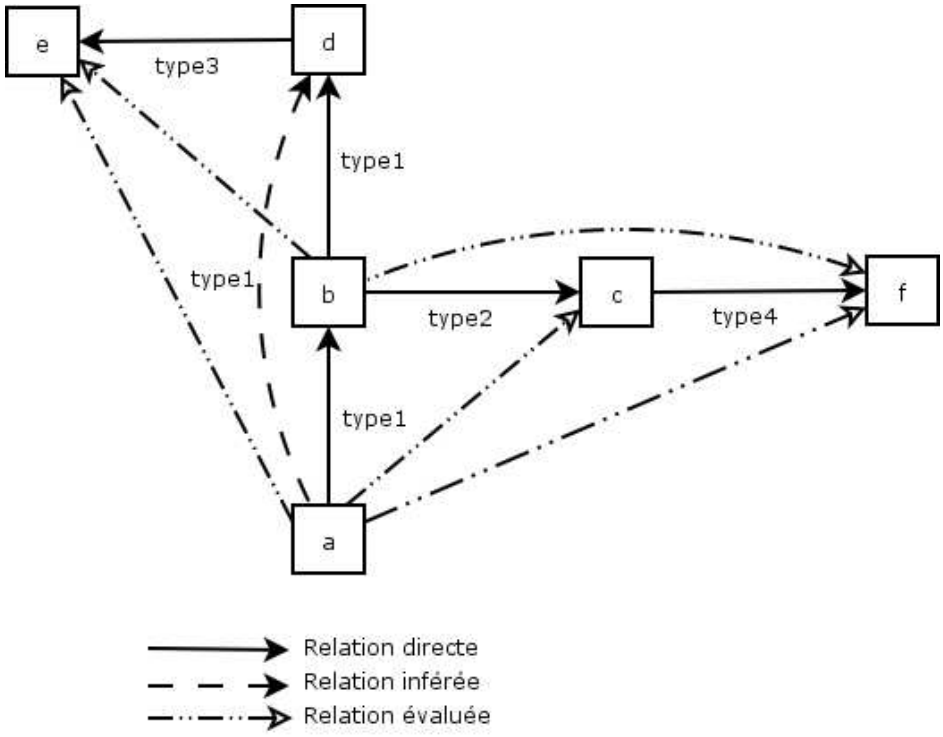

FIG. 7.1: Qui composer

7.3.2 Comment composer ? Proposition d'un algorithme

Sur la simple hiérarchie présentée à la figure 7.1, nous constatons que la complexité devient vite élevée concernant l'évaluation des relations. Il faut donc une méthode de parcours du graphe afin que toutes les relations soient assurées d'être étudiées. A cet effet, nous proposons un algorithme :

soient

$\mathcal{H}$ une hiérarchie de relations *part-whole* qui est un graphe orienté sans boucles,

$\mathcal{F}$ l'ensemble des feuilles de $\mathcal{H}$ (à l'initialisation),

$\mathcal{R}$ l'ensemble des racines de $\mathcal{H}$,

$\mathcal{P}n$ l'ensemble des précédents directs et inférés du noeud n,

$\mathcal{P}.\mathcal{IND}n$ l'ensemble des précédents directs, inférés et indirects du noeud n,

alors

pour tout élément c_i de $\mathcal{F}$, faire

initialiser une liste L = $\mathcal{H}$ afin d'y marquer les noeuds traités ;

pour tout élément b_j de $\mathcal{P}c_i$, faire

pour tout élément a_k de $\mathcal{P}b_j$, faire

- si a_k n'a pas encore été marqué comme traité pour le noeud c_i,

alors évaluer la relation $(a_k,\ c_i)$: la relation peut-être un des sept types *part-whole* ou *rien* ou *impossible*,

et marquer a_k comme traité pour le noeud c_i,

- si $(a_k,\ c_i) \neq$ *rien* ou *impossible*,

alors, tant que la relation est différente de *rien* ou *impossible*, si le noeud n'a pas déjà été traité, évaluer la relation $(d_l,\ c_i)$ pour tout $d_l \in \mathcal{P}.\mathcal{IND}a_k$ (jusqu'à $d_l \in \mathcal{R}$) et le marquer comme traité pour le noeud c_i ;

mettre à jour $\mathcal{F}$ qui est remplacé par $\mathcal{P}c_i$ pour tout c_i appartenant à $\mathcal{F}$;

recommencer jusqu'à ce que $\mathcal{F} \subseteq \mathcal{R}$.

Avec cet algorithme, nous pouvons réviser l'exemple donné à la figure 7.1 :
– les relations directes sont les couples *(a, b)*, *(b, c)*, *(b, d)*, *(c, f)* et *(d, e)* ;
– *(a, d)* est la relation inférée car le type qui lie *(a, b)* et *(b, d)* est le même ;
– nous pouvons appliquer l'algorithme :

$\mathcal{H}$={*a, b, c, d, e, f*},
$\mathcal{F}$={*e, f*},
$\mathcal{R}$={*a*},
pour *e* (avec $\mathcal{P}e$={*d*})
L = [*a=false, b=false, c=false, d=false, e=false, f=false*] ;
pour *d* (avec $\mathcal{P}d$={*a, b*})
pour *a*
- comme *a=false* dans L,
alors évaluer la relation *(a, e)*,
et marquer *a=true* dans L,
- comme *(a, e)*≠*vide* ou *impossible*,
alors rien car $a \in \mathcal{R}$;
pour *b*
- comme *b=false* dans L,
alors évaluer la relation *(b, e)*,
et marquer *b=true* dans L,
- comme *(b, e)*≠*vide* ou *impossible*,
alors, avec $\mathcal{P}.\mathcal{IND}$={*a*}, *a=true* dans L, alors rien ;
pour *f* (avec $\mathcal{P}f$={*c*})
L = [*a=false, b=false, c=false, d=false, e=false, f=false*] ;
pour *c* (avec $\mathcal{P}c$={*b*}),
pour *b*
- comme *b=false* dans L,
alors évaluer la relation *(b, e)*,
et marquer *b=true* dans L,
- comme *(b, e)*≠*vide* ou *impossible*,
alors, avec $\mathcal{P}.\mathcal{IND}$={*a*}, *a=false* dans L, alors évaluer *(a, f)* et marquer *a=true* dans L ;
mettre à jour $\mathcal{F}$ qui est remplacé par $\mathcal{P}e$ et $\mathcal{P}f$, $\mathcal{F}$ devient ={*d, c*} ;

$\mathcal{F}$ n'est pas $\subseteq \mathcal{R}$, alors recommencer ;

pour b (avec $\mathcal{P}b=\{a\}$),

L = [*a=false, b=false, c=false, d=false, e=false, f=false*] ;

pour a (avec $\mathcal{P}a=\{\}$),

rien (ensemble des précédents de a vide),

pour c (avec $\mathcal{P}c=\{b\}$),

L = [*a=false, b=false, c=false, d=false, e=false, f=false*] ;

pour b (avec $\mathcal{P}b=\{a\}$),

- comme *a=false* dans L,

alors évaluer la relation *(a, c)*,

et marquer *a=true* dans L,

- comme *(a, c)≠vide* ou *impossible*,

alors rien car $a \in \mathcal{R}$;

mettre à jour $\mathcal{F}$ qui est remplacé par $\mathcal{P}b$ et $\mathcal{P}c$, $\mathcal{F}$ devient $=\{a\}$;

$\mathcal{F} \subseteq \mathcal{R}$, alors fin.

Validation de l'algorithme et complexité

L'algorithme est assuré de se terminer :

- le graphe est sans circuit, cela est logique pour une hiérarchie *part-whole*. S'il contenait des circuits, l'algorithme devrait inclure des mécanismes de gestion de boucles ;
- dans une itération, on évalue les relations entre un noeud et tous ses précédents. On commence avec les noeuds feuilles. Comme le graphe ne contient pas de circuit, on évaluera au plus un noeud par rapport à tous les autres noeuds du graphe ;
- à chaque nouvelle itération, l'ensemble des noeuds à évaluer (les feuilles au départ) est remplacé par l'ensemble de leurs précédents, et ce jusqu'aux noeuds racines. L'algorithme est donc assuré de se terminer.

Comme chaque noeud peut au pire être comparé à tous les autres, et que tous les noeuds seront évalués, alors l'algorithme est de complexité n^2, avec n le nombre de noeuds.

7.3.3 Quand composer ?

A la création de l'ontologie. Lors de la mise en place de l'ontologie, une fois les relations *part-whole* définies, alors on peut exécuter l'algorithme. À toute nouvelle création de noeud, on met en place un mécanisme qui va le réévaluer par rapport à l'ontologie déjà en place. Comme l'algorithme proposé a une complexité n^2, si l'ontologie contient un grand nombre de noeuds, elle peut entraîner un temps d'évaluation important, mais cela n'est fait qu'une fois.

Lors de la requête. A l'opposé, quand l'utilisateur inclut un critère de recherche qui se trouve dans une relation de composition, alors on évalue le graphe par rapport à ce noeud précis pour en déduire de nouvelles relations. Par cette option, le temps de réponse s'en trouverait augmenté, et les relations devraient être évaluées à chaque requête. Cette solution est déconseillée si le temps de réponse est un critère important, ou si les requêtes de ce type sont fréquentes.

Combinaison. Lors de l'évaluation de la requête par rapport à un noeud précis, on déclare les nouvelles relations découvertes, afin d'enrichir l'ontologie de manière incrémentale.

7.3.4 La nouvelle réponse

La réponse, formée comme indiqué à la section 6.4, inclura, en plus des composants/composites directs et inférés, les composants/composites indirects qui ont été évalués par rapport à la table de composition inter-types de relations *part-whole*.

7.4 Réponse enrichie

Nous avons vu que pour la composition de données, nous indiquons uniquement à l'utilisateur vers quelles données se diriger pour refaire une nouvelle requête, selon que la donnée demandée initialement est un composant ou un composite.

Nous n'avons cependant pas encore mis en place de mécanisme qui enrichisse la liste des services effectivement proposés.

En fait, qu'une donnée en compose une autre ne suffit pas à en conclure que le service qui traite cette donnée conviendra. Nous l'illustrons à la figure 7.2. Pour bien comprendre, commençons par raisonner sur des exemples.

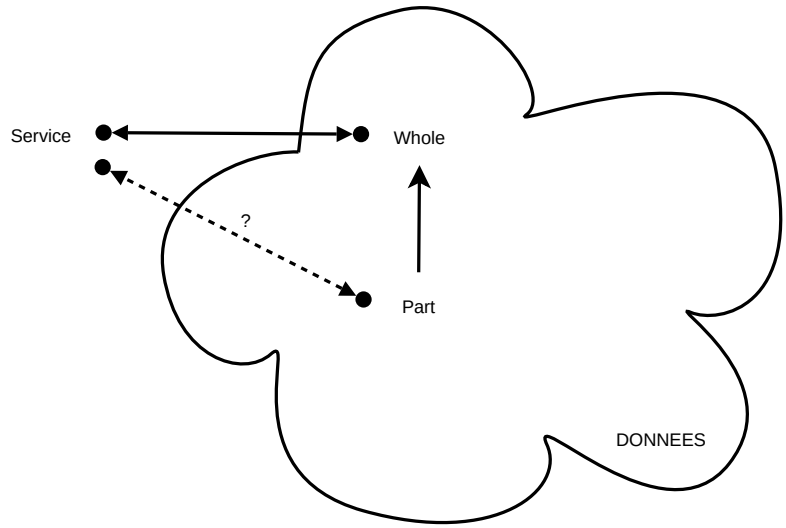

FIG. 7.2: Les services par rapport aux composants

7.4.1 Illustration du problème

Nous modifions l'exemple du *camion* et de l'*assurance* introduit à la page 82 : nous avons le *contenu* qui est un composant de l'objet *camion* et il existe un service *assurer* qui prend un objet et l'assure. La question est de savoir si *assurer* s'applique également au composant *contenu*.

Pour une relation élément/collection, soit *arbre/forêt*, il existe un service *protection* qui se charge de protéger la *forêt*. Est-ce que ce service protège alors les *arbres* de cette *forêt* ? A priori, oui, mais un service s'appliquant à une collection ne s'applique pas forcément à ses éléments. Par exemple, l'*orchestre* qui est engagé pour donner une représentation passe ensuite par le service *rémunération*, ce qui ne signifie pas que *rémunération* va verser l'argent à chaque musicien, car l'entité *orchestre* a son propre compte et l'argent gagné servira à l'*orchestre*, pour acheter de nouveaux costumes par exemple.

Nous constatons sur ces quelques exemples que le service est indépendant des règles de transitivité intra et inter-types de relations *part-whole*. De plus, il manque une information qui permettrait de déterminer si un service s'appliquant à un objet composite s'applique aussi à ses composants (ici, au *contenu*, à l'*arbre* et au *musicien*).

7.4.2 Illustration dans le domaine médical

Le domaine médical est une application importante des relations *part-whole* ([Ber95] et [MARR03]). Ce domaine offre donc une belle illustration, des exemples de données

sont :

- la tête de fémur *est un composant de l'objet* fémur ;
- la vertèbre *est un membre de la collection* colonne vertébrale ;
- l'échantillon de sang veineux *est une portion de la masse* sang veineux ;
- l'hémoglobine *est une substance de l'objet* erythrocyte ;
- le monitoring *est une phase de l'activité* anesthésie ;
- le segment de fémur *est un lieu de l'espace* fémur ;
- le pancréas est à la fois un *part* du système digestif et du système endocrine.

Souvent, en médecine, un phénomène, une maladie ou une procédure qui est liée à ou affecte une partie de l'objet anatomique dépend de la classe de phénomènes qui sont liés à un *whole* de l'objet anatomique en question :

- une douleur dans l'aire de la vertèbre lombaire est une sorte de douleur au dos ;
- la fracture du fémur diaphysis est une sorte de fracture du fémur.

Ce domaine nous fournit deux autres exemples de la relation qui peut exister entre un service et des données :

1. « *Je souhaite trouver un service de radiographie de la tête de fémur* ». Il n'y en a pas, mais il existe un service de radiographie du fémur dont la tête est un composant, donc il pourrait convenir ;
2. « *Je souhaite trouver un service de radiographie du fémur* ». Il n'y en a pas, mais il existe 3 services de radiographie de 3 segments du fémur. Appeler ces 3 services et combiner la réponse pourrait convenir.

Nous illustrons ces deux exemples à la figure 7.3.

7.5 Analyse

Comme nous pouvons le constater grâce aux différents exemples introduits ci-dessus, il y a deux grandes catégories à analyser :

1. lorsqu'un service s'applique à un composite, s'applique-t-il à ses composants ;
2. lorsqu'un service s'applique à un composant, comment s'applique-t-il au(x) composite(s) ?

Il faut étudier ces deux questions en rapport à chaque type de relation *part-whole* et selon que la donnée composante/composite est une entrée dont le service a besoin ou bien une sortie qu'il produit.

7.5.1 Services s'appliquant à des composites

La similarité

Une première certitude concerne le critère de similarité qui s'applique aux relations portion/masse et lieu/espace, lorsque l'input d'un service s'applique au *whole* qui est

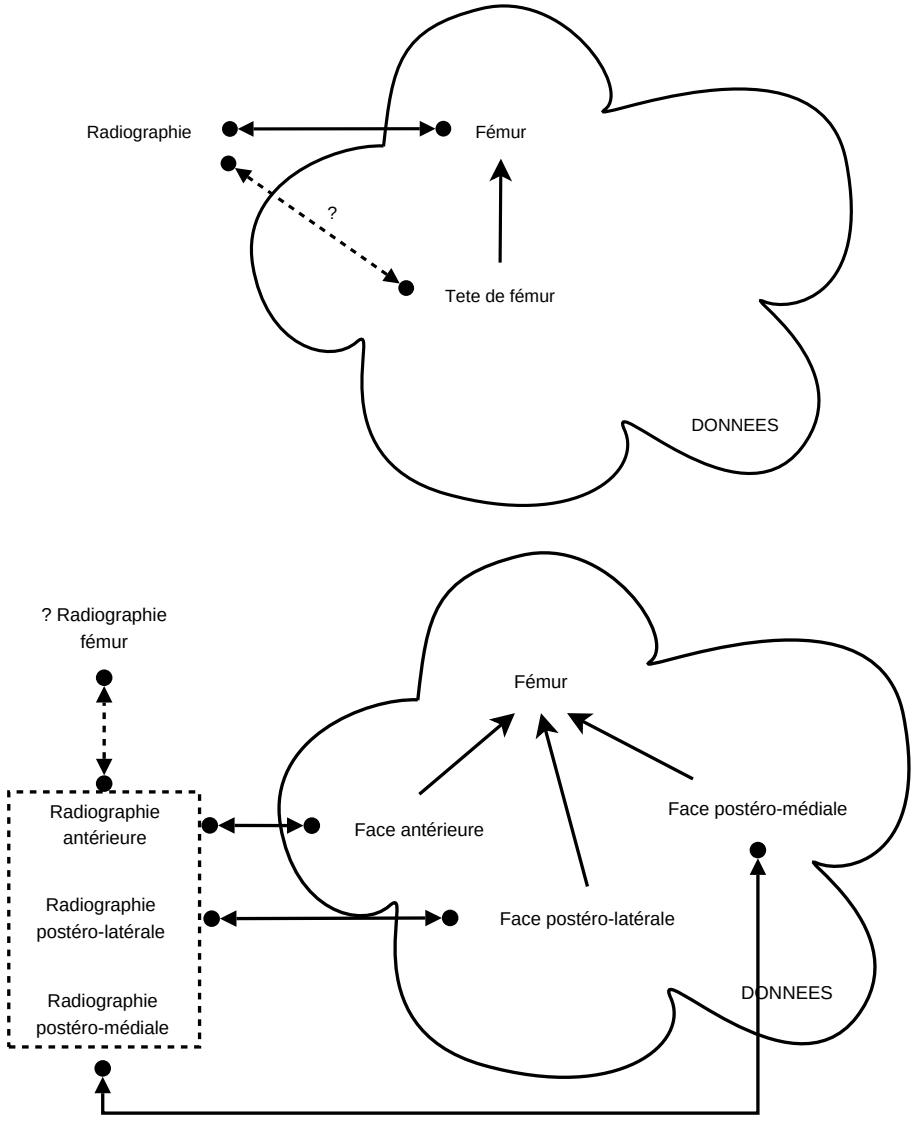

FIG. 7.3: Exemple (domaine médical)

composé du *part* pour lequel nous recherchons des informations. Les données étant similaires, alors nous pouvons en déduire que le service acceptera tout composant quand il s'applique au départ aux composites. Par exemple, le service *servir* qui agit sur la *tarte* fonctionnera également pour la *part de tarte*. Le service *saler* qui s'applique à une recette de cuisine requiert en input du *sel*, il accepte donc les *grains de sel*. Le service *survoler* peut s'appliquer aux *Alpes*, et donc au *mont Blanc* également.

Dans l'exemple médical illustré à la figure 7.3, nous cherchons tout d'abord un service de *radiographie* de la *tête de fémur*, or il existe un service de *radiographie* du *fémur*. Quelle relation lie le *fémur* et la *tête de fémur* ? Le composant est ici similaire au composite, il n'en est pas séparable, il lui est simultané et il n'a pas de fonctionnalité pour le composite *fémur* (bien qu'il aie une fonctionnalité par rapport à l'environnement du *fémur*, mais pas par rapport à l'objet lui-même). Il est donc un lieu de l'espace *fémur*. Comme nous raisonnons ici du composite vers le composant, concernant un input du service, alors le critère de similarité est suffisant pour en conclure que le service de radiographie du *fémur* convient parfaitement lorsque l'utilisateur est à la recherche d'un service de radiographie de la *tête de fémur*.

Etude selon les types de relation part-whole

Nous présentons les spécificités qui s'attachent à la relation d'un service par rapport à chaque type de relation :

1. **composant/objet** :
   - un service qui utilise un objet en utilise indirectement les composants car ils lui sont simultanés et ils ont une fonction dans l'objet. Étant séparables, il faudrait savoir si le service peut s'y appliquer d'une manière directe, lorsqu'ils sont pris isolément de leur objet. Cela semble difficile à déterminer si aucune information supplémentaire n'accompagne le service concerné, car chaque composant d'un objet y a sa fonction propre et il faudrait que le service indique à quelles fonctions il peut s'appliquer. Par exemple, le service *peinture* qui s'applique à une *voiture* s'applique également à *carrosserie*, mais pas à *moteur* ni à *roue*,
   - si on isole la fonctionnalité, donc si les composants et l'objet n'étaient pas distinguables du point de vue fonctionnel, y a-t-il alors moyen d'appliquer automatiquement le service aux composants ? Isoler la fonctionnalité revient à considérer la relation qui lie les composants et l'objet comme étant la relation élément/collection qui a les critères de séparabilité et de simultanéité :
2. **élément/collection** :
   - lorsqu'un service s'applique à une collection, nous avons vu par les deux exemples introduits à la section 7.4.1 qu'il n'a pas toujours un effet sur les éléments de la collection. Il manque une information sur le service,

- cette information est la distributivité, le service doit indiquer si il se distribue sur les composants de l'objet composite qu'il utilise. Pour la *rémunération* des *musiciens*, le service indiquera donc si il paie l'*orchestre* uniquement, ou ses *musiciens* en particulier. La distributivité sur les composants est une information simple à indiquer pour le service, contrairement à une information de fonction que réclamerait le critère de fonctionnalité ;

3. **portion/masse** : ce type a le critère de similarité, ce qui suffit à conclure que le service peut s'appliquer à une portion prise isolément ;
4. **substance/objet** :
   - l'unique critère positif de ce type est la simultanéité. Autrement dit, la substance est non fonctionnelle, non similaire et non séparable de l'objet,
   - la non séparabilité nous intéresse, elle implique que tout service s'appliquant à un *whole* composé de *parts* non séparables s'applique aussi à ses composants de manière indirecte, mais qu'il ne peut s'appliquer à eux isolément car ils sont inséparables de l'objet qui perdrait son identité si on les supprimait. Ce qui ne signifie pas que les composants sont exclusifs de l'objet, par exemple, l'*hydrogène* est substance d'autres objets que l'*eau*, ou encore l'*acier* ne compose pas uniquement le *vélo*. Seulement, pris hors de leur objet, il n'y a aucune raison que le service s'applique à eux. Supposons l'*hydrogène* qui est la substance non séparable de l'*eau*, le service *électrolyse* prend en entrée de l'*eau* et renvoie en sortie l'*hydrogène* et l'*oxygène* de cette *eau*, mais même dans cet exemple, le service ne s'applique pas à la substance uniquement,
   - contrairement à l'exemple de la *tête du fémur* qui est un lieu non séparable mais similaire du *fémur*, la non similarité fait que nous ne pouvons rien conclure ici,
   - la non fonctionnalité permet de conclure qu'il ne faut pas d'information supplémentaire sur la fonction que peut remplir le service,
   - et enfin, la simultanéité ne permet aucune hypothèse non plus, à part le fait qu'un service s'appliquant à un objet s'applique indirectement à ses substances qui en sont simultanées,
   - en conclusion, pour la relation substance/objet, aucune information, même de distributivité, n'est utile car il n'y a aucune raison qu'un service qui s'applique à un objet puisse également s'appliquer à sa substance uniquement, même comme pour l'exemple de l'*électrolyse*, il s'attache à changer cette substance ;
5. **étape/événement** : l'étape a une fonction par rapport à l'événement et en est séparable. Elle ne lui est ni similaire, ce qui signifie que le service ne s'y applique pas forcément, ni simultanée, c'est-à-dire qu'il ne s'y applique pas forcément de manière indirecte. Donc, sans information supplémentaire par rapport au service, rien ne peut être déduit de sa relation par rapport au composant. De plus, la présence du critère de fonctionnalité implique qu'il faudrait de l'information

supplémentaire qui précise les fonctions prises en charge par le service

6. **phase/activité** : à la différence de étape/événement, il n'y a pas de séparabilité. Cela ne permet pas de déduction supplémentaire concernant le comportement du service par rapport aux phases de l'activité qui le concerne ;
7. **lieu/espace** : un lieu est similaire et simultané à son espace, ce qui suffit pour dire que le service s'appliquera aussi bien sur le composite que ses composants.

Conclusion

Savoir si des services qui s'appliquent à un objet composite peuvent s'appliquer à ses composants revient à étudier la question de la hiérarchie de données en entrée, comme vu au point 5.5.2 à la page 54, où l'utilisateur qui recherche les services prenant en entrée une certaine donnée peut être intéressé par les services s'appliquant aux sur-types de cette donnée. Ici, l'objet composite équivaut au sur-type, et nous avons vu qu'il n'est pas possible pour tous les types de relations *part-whole* d'en déduire que le service s'applique aux composants de cet objet.

Concernant les critères qui caractérisent ces types, la similarité est suffisante, tandis que la fonctionnalité pose des limitations.

Une information supplémentaire de distributivité permet au service d'indiquer s'il accepte des composants quand il en prend l'objet composite en entrée.

7.5.2 Services s'appliquant à des composants

Raisonner des composants vers le composite équivaut à la recherche de services par rapport à ses outputs, comme vu au point 5.5.2, page 55. Lorsque l'utilisateur y cherchait les services qui produisent un output d'un certain type, alors il était intéressé par les services produisant une donnée d'un sous-type de celui recherché.

L'équivalence est qu'ici, il existe des services associés aux composants, lorsque l'on cherche l'ensemble des services qui s'apparentent à un objet composite. Voyons comment nous pouvons raisonner lorsqu'il s'agit de relations *part-whole*.

La similarité

Dans le second exemple illustré à la figure 7.3, si les trois lieux *face antérieure*, *face postéro-latérale* et *face postéro-médiale* de l'espace *fémur* ont chacun un service de *radiographie*, alors il est possible de conclure que, mis ensemble, ces trois *radiographies* peuvent convenir pour obtenir une *radiographie complète du fémur*. On devrait donc combiner les trois outputs qui sont les trois *radiographies* des différents lieux qui composent l'espace *fémur*.

Donc, quand il y a similarité, alors on peut proposer à l'utilisateur le groupement de tous les services qui s'appliquent aux composants de l'objet composite pour lequel il recherche un service, ici la *radiographie*.

Etude selon les types de relation part-whole

Nous présentons les spécificités qui s'attachent à la relation d'un service par rapport à chaque type de relation :

1. **composant/objet** :
   - Le service *réparation de moteur* s'applique au composant *moteur* de la *voiture*. On peut dire que ce service s'applique aussi à la *voiture*. En effet, si l'on applique un service *réparation de moteur* à la *voiture*, alors il va en réparer le composant *moteur*. De même, le service *peinture* qui s'applique au composant *carrosserie* est également un service de *peinture* de la *voiture*,
   - comme un composant est affecté par le service (dans l'exemple, le composant est *réparé* ou *peint*), et vu que ce composant a une fonction dans l'objet composite, alors l'objet composite se trouve également affecté par le service.
   - toutefois, la généralisation qui va dans le sens *composant vers composite* n'est pas automatique. Il faut un service intermédiaire, par exemple le service *ouvrir capot de voiture* avant de pouvoir *réparer* le *moteur*. Etant donné qu'il y a le critère de fonctionnalité, on ne peut définir un service intermédiaire générique qui permettrait de gérer la généralisation de manière automatique, contrairement à la relation élément/collection, comme nous le voyons au point suivant ;
2. **élément/collection** :
   - Le service *abattre* s'applique à l'*arbre* qui est un élément de la collection *forêt*. Peut-il s'appliquer à *forêt* ? Oui, à condition qu'une fonction intermédiaire soit ajoutée,
   - ici, la fonction intermédiaire est générique, elle se charge d'appeler x fois le service et à lui passer les x éléments de la collection, un à un. Dans l'exemple, elle donnera chaque *arbre* à *abattre*,
   - donc, dans le sens *composite vers composant* étudié précédemment, nous avons vu qu'il faut une information de distributivité. Ici, nous n'avons pas besoin d'information supplémentaire, mais bien d'un service prédéfini qui se charge de passer chaque élément de la collection un à un au service ;
3. **portion/masse** est similaire, ce qui suffit à dire que la généralisation est possible. Nous pouvons *manger* une *tarte* lorsque nous savons *manger* des *portions de tarte*. Pour ce faire, nous appelons à nouveau la fonction générique qui va donner toutes les portions de la masse au service. En effet, si l'on *mange* toutes les *parts* de la *tarte*, alors cela revient à *manger* la *tarte* entière ;

4. **substance/objet** : dans cette relation, nous avons des services qui s'appliquent à la substance d'un objet. La substance est non séparable de son objet, elle le compose. La détruire ou la modifier affecte directement l'objet. Par exemple *fondre* modifie l'*acier* et donc aussi le *vélo*. Dans le sens *composant vers composite* que nous étudions ici, nous nous intéressons aux services qui produisent ou modifient des données en sortie, or ici, une modification implique une altération de l'objet qui composé de la substance en question. Nous ne pouvons rien déduire, tout comme pour le sens *composite vers composant* étudié précédemment ;
5. **étape/événement** : le critère de fonctionnalité présent dans ce type de relation implique qu'une fonction générique n'est pas suffisante, il n'est pas possible de déterminer automatique comment se comporter face à un événement lorsque des services s'appliquent à ses étapes ;
6. **phase/activité** : à la différence de la relation précédente, la phase n'est pas séparable de l'activite. Cette caractéristique n'amène à aucune conclusion supplémentaire ;
7. **lieu/espace** : ce critère a été présenté dans l'exemple de la *radiographie* du *fémur* à partir des différentes *radiographies* des ses lieux. Les lieux sont similaires et non séparables de leur espace, on peut composer les outputs des différents services pour obtenir une réponse qui satisfera l'utilisateur.

La fonctionnalité

Pour mieux comprendre la problématique qui s'attache au critère de fonctionnalité, prenons un exemple mathématique : nous recherchons un service d'*addition* de *matrices*. Dans l'ontologie enrichie des relations *part-whole* sur laquelle nous souhaitons effectuer la recherche, nous avons défini une *matrice* comme étant composée de *vecteurs colonne* qui sont eux-même composés de *valeurs entières*. La relation qui lie les *valeurs* aux *vecteurs*, et les *vecteurs* à la *matrice* est une relation composant/objet. En effet, un *vecteur* a une fonction dans la *matrice* (il a une place bien déterminée et une signification), il lui est simultané (sans *vecteur*, pas de *matrice*) et séparable (on peut traiter un *vecteur* de manière isolée) :

$$
\begin{array}{ll}
matrice & \\
\uparrow & \textit{composant/objet} \\
vecteur & \\
\uparrow & \textit{composant/objet} \\
valeur &
\end{array}
$$

Retenons uniquement le critère de fonctionnalité. Il existe un service d'*addition* de *valeurs*. Nous pouvons dire qu'il faut appeler ce service pour chaque couple de *valeurs*

selon leur place dans les *vecteurs*, et donc dans les *matrices*. Ensuite, nous remettons chaque *valeur résultat* au bon endroit pour obtenir la *matrice résultat* additionnée :

$$\begin{pmatrix} 1 & 3 & 2 \\ 1 & 0 & 0 \\ 1 & 2 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \\ 2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1+0 & 3+0 & 2+5 \\ 1+7 & 0+5 & 0+0 \\ 1+2 & 2+1 & 2+1 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 7 \\ 8 & 5 & 0 \\ 3 & 3 & 3 \end{pmatrix}$$

Avec l'information présente dans l'ontologie, il n'est pas possible de déterminer l'agencement des opérations automatiquement, nous le comprenons mieux à la figure 7.4 : à part le fait d'indiquer que chaque élément est le composant d'un autre, aucune information ne détermine la fonction que cet élément occupe dans l'objet composite.

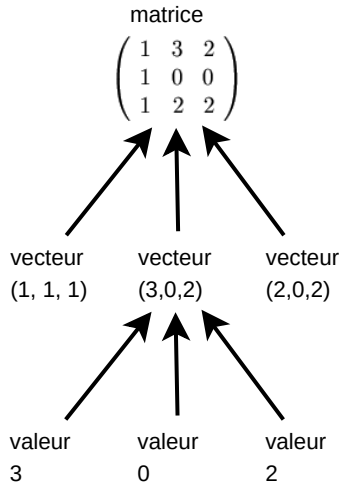

FIG. 7.4: Représentation de la matrice dans l'ontologie de données

La fonctionnalité est donc bien une information à ajouter de manière explicite dans un modèle, il n'est pas possible de l'étudier d'une manière généralisée. Le modèle de services et données Bigre est générique, il s'applique actuellement à la bioinformatique, mais pourrait assez simplement être porté sur un autre domaine. C'est son instanciation qui définira de manière spécifique la bioinformatique et l'agencement de ses services et données. De ce fait, il n'est pas possible d'envisager une manière d'ajouter de l'information de fonctionnalité à ce stade-ci.

L'ontologie obtenue sera donc précise et ciblée, telle qu'en est la définition de départ : « *une ontologie est une spécification explicite d'une conceptualisation* » (voir chapitre 4). Donc, toute application travaillant sur une certaine ontologie pourra y intégrer ses propres règles quant au critère de fonctionnalité. Par exemple, une ontologie de la mécanique enrichira la relation composant/objet afin d'y intégrer les fonctionnalités que remplissent les différents composants entre eux et par rapport à leur objet

composite. Une ontologie mathématique s'attardera à définir clairement les relations entre données et services.

Conclusion

Dans le sens *composant vers composite*, nous pouvons conclure qu'il est possible de proposer une réponse satisfaisante à l'utilisateur, au moyen d'une fonction intermédiaire qui se charge de regrouper soit les différents services associés à chaque composant, soit en appelant le service qui s'applique aux composants autant de fois qu'il n'y a de composants.

La fonctionnalité implique que nous fassions appel à une fonction supplémentaire qui ne peut à priori pas être définie d'une manière générique. La similarité garantit qu'une fonction générique suffit.

7.5.3 Synthèse

Nous avons mis en évidence le fait que certains critères permettent ou non des conclusions sur l'utilisation des services par rapport à la relation *part-whole* :

1. **la fonctionnalité** : elle est telle que nous ne pouvons apporter de conclusion si il n'existe pas d'information supplémentaire spécifique aux fonctions des données entre elles, et aux fonctions auxquels sont capables de répondre les services. La fonctionnalité s'applique aux relations composant/objet, étape/événement et phase/activité.
2. **la similarité** : lorsqu'une relation n'est pas fonctionnelle, nous pouvons regarder si elle est similaire. C'est le cas pour portion/masse et lieu/espace. Dans le sens *composite vers composant* que nous avons tout d'abord étudié, ce critère suffit à dire qu'un service s'appliquant à un objet composite offre une réponse pour ses composants. Dans le sens *composant vers composite* que nous avons abordé ensuite, il suffit d'une fonction générique qui séquence les données composantes, afin de proposer un groupement d'outputs produits par les services.
3. **la séparabilité** : si un type n'est ni fonctionnel, ni similaire, alors nous pouvons regarder s'il est séparable ou non. S'il l'est, comme pour la relation élément/collection, alors, lorsque nous avons affaire à un service qui a indiqué qu'il est distributif le long des composants, il peut s'appliquer aux composants de l'objet composite. A l'inverse, il suffit d'une fonction générique pour gérer la composition de services.
4. **la simultanéité** : le type non fonctionnel, non similaire, non séparable et simultané est substance/objet. Dans un sens ou dans l'autre, cette relation ne permet pas de décider comment les services peuvent se distribuer le long de la relation.

Dans Racer, lorsque l'utilisateur interroge le modèle afin de trouver certains services, s'il demande une recherche sur un certain type de donnée, alors :

on étudie l'instance concernée, par la commande
`(describe-individual <TYPE>)` ;

si le type est demandé en entrée de service, alors on regarde s'il est le composite d'un autre objet,

si oui, alors
- on ne s'intéresse pas à cet autre objet quand la relation est composant/objet, étape/événement, phase/activité, ou substance/objet,
- on renvoie à l'utilisateur les services associés à cet objet quand la relation est portion/masse ou lieu/espace,
- quand la relation est élément/collection, on regarde si les services associés à cet objet indiquent qu'ils sont distributifs. Si tel est le cas, alors on renvoie aussi ces nouveaux services à l'utilisateur ;

si non, alors il n'y a pas de service supplémentaire que nous pourrions proposer à l'utilisateur ;

si le type est demandé en sortie de service, alors on regarde s'il est le composant d'un autre objet,

si oui, alors
- on ne s'intéresse pas à cet autre objet quand la relation est composant/objet, étape/événement, phase/activité, ou substance/objet,
- quand la relation est portion/masse, lieu/espace ou élément/collection, alors nous faisons appel à une fonction intermédiaire présentée ci-dessous,

si non, alors il n'y a pas de service supplémentaire que nous pourrions proposer à l'utilisateur.

Concernant la fonction intermédiaire, il faut distinguer deux situations :
- il y a la situation illustrée dans la deuxième partie de la figure 7.3, page 99, où chaque lieu a un service qui lui est propre. Dans ce cas, la fonction intermédiaire se charge de rassembler tous ces services et propose un sur-service à l'utilisateur. Si l'utilisateur choisi ce sur-service, celui-ci se chargera d'appeler chaque service associé à chaque composite, et renverra la liste groupée des différentes réponses obtenues. Dans l'exemple, il donnera à l'utilisateur les trois radiographies composées, qui se ramènent à une radiographie complète ;
- un même service est associé à tous les types de part, de lieu, ou d'élément qui composent la masse, l'espace, ou la collection qu'est le type sur lequel nous travaillons (le type demandé par l'utilisateur). Dans ce cas, la fonction intermédiaire proposera un sur-service qui, s'il est appelé par l'utilisateur, va appeler x fois le

service concerné pour qu'il travaille sur les x types qui composent le type de départ.

7.6 Critiques

7.6.1 Le niveau de perception

Nous avons défini les différents types *part-whole* et développé notre analyse sur base de quatre critères. Il faut cependant noter que la frontière qui sépare un critère positif de son pendant négatif n'est pas clairement définie, elle intègre une dimension supplémentaire qui est le niveau de perception. Celui-ci varie en fonction du domaine, ou plutôt de l'expertise de la personne qui souhaite modéliser ce domaine.

Par exemple, nous avons statué que les éléments *arbres* de la collection *forêt* ne sont pas similaires. À un certain niveau, on peut considérer que dans une *forêt* d'*hêtres*, chaque *arbre* est similaire à un autre, vu qu'ils sont tous des *hêtres*. L'herbologiste va évidemment contredire le fait que deux *hêtres* soient similaires.

Plus subtil, pourquoi statuer qu'un *grain de sable* est similaire à un autre *grain de sable* et similaire au *sable* qu'il compose ? Les *grains* sont des petites particules de roche, et il existe différents types de roches, donc deux *grains* ne sont pas similaires selon cet angle de vision. Par exemple, en granulométrie, où l'on classe les *sables* selon la grosseur des *grains*, on ne considérera pas que les *sables* soient similaires. On distingue aussi le *sable* selon sa capacité à s'écouler ; il peut aussi être transporté par le vent ou par l'eau, etc.

Concernant le critère de fonctionnalité, nous avons défini la substance comme étant non fonctionnelle par rapport à son objet. L'*hydrogène* est la substance de l'*eau*, or un chimiste désapprouvera le fait que nous n'attachions pas de critère de fonctionnalité à cet *hydrogène* par rapport à l'*eau* qu'il compose.

Nous mettons donc en évidence le fait qu'il y a toujours une certaine interprétation des quatre critères présentés. Il n'y a évidemment pas de réponse absolue à ce problème.

7.6.2 La représentation de la connaissance

De même, le simple fait de mettre en place une ontologie relève quelque peu du défi. Il ne sera pas possible de mettre tout le monde d'accord concernant une unique manière de représenter la connaissance d'un domaine particulier ou d'une tâche donnée.

La question de la représentation de la connaissance couvre un domaine fort vaste que nous n'avons pas fini d'explorer.

Le *part-whole*, qui tente d'enrichir les ontologies, et donc d'améliorer la réprésentation de la connaissance, n'est pas non plus optimal. L'application la plus connue du

part-whole est le domaine médical, afin de représenter l'anatomie. Une partition y est la décomposition du corps ou d'une structure anatomique étant donné un contexte, à savoir selon un certain point de vue. L'article [MARR03] revisite la relation *part-whole* appliquée à l'anatomie afin d'en proposer leur propre taxinomie[2] basée sur de nouvelles règles. Ils introduisent une plus grande spécificité, afin de lever les ambiguïtés : il existe une règle de consistance de la dimensionalité, une règle de distinction *contenu/part*, etc. Nous constatons donc que le *part-whole* n'est pas encore suffisant pour la représentation de la connaissance, et nous pensons qu'il sera encore enrichi dans les prochaines années.

7.6.3 Autres Relations

Il existe d'autres types de relations que le *part-whole*. Par exemple, une relation qui mette en évidence le rapport qui existe entre deux objets, en indiquant s'il y a ou non équivalence sémantique.

Nous le voyons sur des exemples illustrés à la figure 7.5 :

1. nous avons deux *types de données* pour représenter la température : *celsius* et *fahrenheit*. Un utilisateur qui recherche les programmes travaillant sur des données du type *celsius* seront intéressés par des programmes travaillant sur des degrés *fahrenheit*, à la condition qu'il existe un programme de conversion de la donnée *celsius* fournie par l'utilisateur. Il faut donc une premier information qui concerne ce programme de conversion ;
2. nous avons deux *types de données* pour représenter une image : *bmp* et *jpg*. Nous faisons les mêmes remarques que pour l'exemple précédent, seulement ici, nous mettons en évidence le fait qu'il faille une information supplémentaire au programme de conversion : l'information sémantique. Cette information permettra d'indiquer que la conversion allant de *bmp* à *jpg* provoquera une perte d'information sur l'image donnée. Il existe également un programme de conversion du *jpg* vers le *bmp*, avec, dans ce sens-ci, équivalence sémantique. En effet, on ne peut pas *retrouver* de l'information dont le format bmp, qui est plus riche, disposait avant de passer à un format *jpg*, mais on ne perd pas d'information que le *jpg* ait et que n'aurait pas le *bmp* ;
3. en bioinformatique, il s'agirait, par exemple, de mettre en rapport les formats *fasta* et *raw format* qui permettent de stocker des séquences. Dans le format *fasta*, la première ligne est composée du symbole > suivie du nom de la séquence et d'une éventuelle description. Sur une seconde ligne, il y a la chaîne de caractères qui représente la séquence des nucléotides de l'ADN ou des acides aminés de

[2]La taxinomie est la partie de la biologie visant à établir une classification systématique des êtres vivants

la protéine. Le second format, *raw format*, équivaut au premier, mais sans la première ligne. Passer du format *fasta* au *raw format* provoque donc une perte sémantique, vu qu'on a supprimé deux informations : le nom de la séquence et la description. On peut repasser d'un *raw format* au format *fasta*, sans perte sémantique, par ajout d'une première ligne factice.

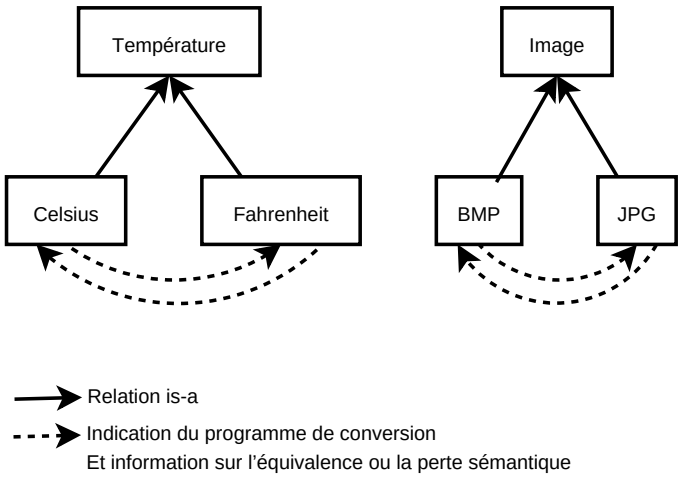

FIG. 7.5: Conversion de type et équivalence/perte sémantique

Un autre type de relation est la relation contenant/contenu, par exemple la *bouteille d'eau* et l'*eau*, ou bien, comme indiqué dans l'article [MARR03], la nouvelle règle *distinction contenu/part* qui contraint la relation *contient* à la classe *espace anatamique*, et son inverse *contenu-dans* à la classe *substance du corps*.

7.7 Conclusion

Dans ce chapitre, nous avons poussé notre investigation de la relation *part-whole* un cran plus loin. Nous avons abordé deux grandes parties.

Premièrement, nous avons tenté de combiner les différents types, afin de surmonter la propriété qui dit que les types ne sont pas transitifs entre eux. Nous avons donc justifié notre approche et proposé un tableau de composition inter-types. Pour ce faire, nous sommes parti d'un autre tableau, qui existait mais qui n'était pas validé, nous avons étudié cette approche par les critères, et nous avons apporté des exemples et contre-exemples pour justifier la construction de notre tableau.

Nous avons ensuite regardé comment appliquer nos nouvelles propositions de transitivité inter-type dans une ontologie existante. Pour cela, nous proposions un algorithme de parcours du graphe pour y ajouter ces nouvelles informations.

Dans un second temps, nous avons souhaité enrichir la liste des réponses proposés à l'utilisateur de manière automatique. Jusqu'ici, nous nous contentions de lui indiquer que le *type* recherché appartenait à une relation de composition, et nous l'invitions à se renseigner lui-même sur la pertinence des services associés. Nous avons donc étudié les différents cas qui pouvaient se présenter et, grâce aux quatre critères qui définissent les types de relations *part-whole*, nous avons pu tirer des conclusions sur la validité des services associés à des objet composés ou composant l'objet du *type* voulu.

Enfin, nous avons apporté différentes critiques et remarques par rapport à notre étude.

Conclusion

Les services bioinformatiques sont nombreux. Le biologiste, qui a besoin de ces outils de manière régulière ou occasionnelle, se limite souvent aux quelques outils qu'il connaît, alors que d'autres pourraient mieux lui convenir. La recherche de services, selon divers critères, est donc une étape préalable nécessaire à un bon usage de la bioinformatique.

Ce mémoire s'inscrit dans la lignée d'autre mémoires. Tout d'abord, [DDS03] et [BD03] ont permis la mise en place d'un système distribué qui serait l'intermédiaire entre les utilisateurs et les fournisseurs de services bioinformatiques. Ensuite, [Den04] a permis de mettre en évidence l'importance d'une modélisation des données.

Partant de l'ontologie développée pour ce système distribué, Bigre, nous avons mis en évidence certaines formes de questions qui pourraient être posées par l'utilisateur, avec pour avantage qu'il peut cibler sa recherche sur des critères bien plus riches que ce qu'il n'existe actuellement, à savoir une simple description textuelle du service.

Les données et les services sont organisés en hiérarchies, ce qui permet de naviguer le long de la généralisation/spécialisation, selon que la donnée est recherchée comme input du service, ou comme son output.

Nous avons ensuite constaté qu'il serait intéressant d'enrichir l'ontologie au delà du simple héritage *is-a*. Pour ce faire, nous avons étudié en profondeur une relation de composition qui s'appelle le *part-whole*. Cela nous a permis d'enrichir la réponse par rapport à une recherche sur les données prises ou produites par les services.

L'analyse que nous avons menée sur le *part-whole* est originale, dans le sens où aucun aucun article étudié n'a approfondi tous les points que nous avons abordés. Après avoir défini de manière précise les sept relations *part-whole*, chacune ayant quatre critères positifs ou négatifs, nous avons vu quelles contraintes s'y attachent et avons appliqué les concepts étudiés à notre modèle.

Nous avons établi un tableau de composition afin de pouvoir inférer de nouvelles relations par rapport aux relations *part-whole* explicitées dans le modèle, ce qui est

unique. Nous avons également proposé un algorithme pour évaluer l'ontologie existante par rapport aux nouvelles relations inter-types découvertes.

Les critères nous ont ensuite permis de tirer des conclusions quant à la manière d'élargir de manière automatique le spectre des services valides proposés à l'utilisateur. En conclusion de cette analyse, nous avons mis en évidence les rôles de chaque critère, ainsi que la priorité qu'ils ont entre eux.

Bibliographie

[AFGP95] Alessandro Artale, Enrico Franconi, Nicola Guarinon, and Luca Pazzi. Part-whole relations in object-centered systems : An overview. *Data & Knowledge Engineering*, 20 :347–383, 1995.

[BCM+03] Franz Badder, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The description logic handbook - Theory, Implementation and Applications.* 2003.

[BD03] Pierre Buyle and Quentin Dallons. Partage de ressources bioinformatiques hétérogènes - Conception et implémentation d'une fédération de médiateurs. Master's thesis, FUNDP, 2003.

[Bec03] Sean Bechhofer. *The DIG Description Logic Interface : DIG /1.1.* University of Manchester, février 2003.

[Ber95] Jochen Bernauer. Analysis of part-whole relation and subsumption in the medical domain. *Data & Knowledge Engineering*, 20 :405–415, 1995.

[CN03] Jean-Michel Claverie and Cédric Notredame. *Bioinformatics for Dummies.* Wiley Publishing, janvier 2003.

[DDS03] Laurent Debaisieux and Fernando De Souza. Partage de ressources bioinformatiques hétérogènes. Master's thesis, FUNDP, janvier 2003.

[Den04] Marie-Laetitia Denayer. Proposition de modélisation des services bioinformatiques dans le cadre d'une architecture fédérée. Master's thesis, FUNDP, 2004.

[Dev04] Yves Deville. *Théorie de la calculabilité et de la complexité*, 2004.

[EBD+04] Vincent Englebert, Pierre Buyle, Quentin Dallons, Marc Colet, Olivier Dugas, Hugues Mavor, Joseph Bersini, Amin Mantrach, and Tom Lenaerts. Rapport Scientifique et Technique Projet Bigre 1ère année. Technical report, FUNDP, IBMM et IRIDIA, octobre 2004.

[Eng04] Vincent Englebert. *Conception des systèmes d'information coopératifs.* FUNDP, 2004.

[Eti95] Jacqueline Etienne. *Biochimie génétique, Biologie moléculaire.* 1995.

[fow] F-OWl : An OWL Inference Engine in Flora-2. http ://fowl.sourceforge.net/ (vérifié le 24 août 2005).

[Gib01] Cynthia Gibas. *Introduction à la bioinformatique - Concepts fondamentaux et outils logiciels*. 2001.

[GP96] Peter Gerstl and Simone Pribbenow. A conceptual theory of part-whole relations and its applications. *Data & Knowledge Engineering*, 20 :305–322, 1996.

[Gru93] Thomas Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2) :199–220, 1993.

[Jac04] Jean-Marie Jacquet. *Techniques d'Intelligence Artificielle*. FUNDP, 2004.

[LH] Lei Li and Ian Horrocks. *A Software Framework For Matchmaking Based on Semantic Web Technology*.

[MARR03] José L.V. Mejino, Augusto V. Agoncillo, Kurt L. Rickard, and Cornelius Rosse. *Reprensenting Complexity in Part-Whole Relationships within the Founational Model of Anatomy*, 2003.

[MH04] Ralph Möller and Volker Haarslev. *RACER User's Guide and Reference Manual, version 1.7.19*, 26 avril 2004.

[MO97] Martine Magnan and Chabane Oussalah. *Ingénierie objet. Concepts et techniques. Objets et composition*, chapter 2, pages 61–92. 1997.

[Nod] Belgian EMBnet Node. *BIO-Computing with BEN*. ftp ://ben05.ulb.ac.be/pub/BEN-Course/.

[owl] Web Ontology Language OWL, W3C Semantic Web Activity. http ://www.w3.org/2004/OWL/ (mis à jour le 19 août 2005, vérifié le 24 août 2005).

[PL] Lin Padgham and Patrick Lambrix. *A Framework for Part-of Hierarchies in Terminological Logics*.

[Ric] Peter Rice. *Grid and E-Science R&D at the EBI*. EBI, Hinxton. http ://www.ebi.ac.uk/Groups/reports/current/rice.pdf (vérifié le 24 août 2005).

[San04] Yves Sangwo. Workflow intelligent. Master's thesis, Université Libre de Bruxelles - Faculté des Sciences - Département d'Informatique, 2004.

[Sat95] Ulrike Sattler. *A Concept Language for an engineering application with part-whole relations*, 1995.

[Sat03] Ulrike Sattler. *Description Logics for Ontologies*, Avril 2003.

[SS04] Steffen Staab and Rudi Studer. *Handbook on Ontologies*. 2004.

[W3Ca] W3C. Jena - A Semantic Web Framework for Java. http ://jena.sourceforge.net/ (mis à jour le 18 avril 2005, vérifié le 24 août 2005).

[W3Cb] W3C. Jena 2 Inference Support. http ://jena.sourceforge.net/inference/index.html (mis à jour le 12 avril 2005, vérifié le 24 août 2005).

[xsb] XSB. http ://xsb.sourceforge.net/ (mis à jour le 23 juin 2003, vérifié le 24 août 2005).

[YKZ03] Guizhen Yan, Michael Kifer, and Chang Zhao. *Flora-2 : User's Manual. Version 0.92*, juin 2003.

Annexe A

Récapitulatif des relations part-whole

Relations	Exemple	Fonctionnalité	Similarité	Séparabilité	Simultanéité
Composant/objet	roue/voiture	+	-	+	+
Elément/collection	arbre/forêt	-	-	+	+
Portion/masse	part/tarte	-	+	+	+
Substance/objet	hydrogène/eau	-	-	-	+
Etape/événement	analyse/cycle	+	-	+	-
Phase/activité	paiement/achat	+	-	-	-
Lieu/espace	oasis/désert	-	+	-	+

TAB. A.1: Typologie des relations de composition

Nous rappelons la définition de chaque relation *part-whole*, en mettant l'accent sur les spécificités qu'elles présentent. Chaque définition est incrémentale par rapport à une précédente, avec pour point de départ la relation **composant/objet** car elle est la relation de composition la plus utilisée en ingénierie de la connaissance.

1. **composant/objet**
   (a) relation structurelle ou fonctionnelle : 1) entre les différents composants et 2) entre chaque composant avec le composite. D'où critères de fonctionnalité,
   (b) les composants interviennent en même temps. D'où critère de simultanéité,
   (c) composant : Frontières inhérentes. D'où critère de séparabilité,
   (d) composite : Non homogène. D'où pas de critère de similarité,
   (e) critères : fonctionnalité, séparabilité, simultanéité,
   (f) exemple : roue/voiture ;

2. **élément/collection**
   (a) relation composant/objet mais pas de relation structurelle ou fonctionnelle particulière. D'où critère de fonctionnalité en moins,
   (b) critères : séparabilité, simultanéité,
   (c) exemple : arbre/forêt ;
3. **portion/masse**
   (a) relation élément/collection mais les composants *portion* sont des parties similaires entre elles et similaires au composite *masse*. D'où critère de similarité en plus,
   (b) critères : similarité, séparabilité, simultanéité,
   (c) exemple : part/tarte ;
4. **substance/objet**
   (a) rapport entre un objet et ce dont il est fait. Le composite perd sont identité quand un composant est supprimé. D'où unique critère de simultanéité,
   (b) critères : simultanéité,
   (c) exemple : hydrogène/eau ;
5. **étape/événement**
   (a) Relation composant/objet (rapport structurel entre les composants *étapes* et le composite *événement*) mais pas de simultanéité, les composants interviennent à différents moments,
   (b) critères : fonctionnalité, séparabilité,
   (c) exemples : analyse/cycle ;
6. **phase/activité**
   (a) relation étape/événement sans séparabilité. Le composant *phase* ne peut être séparé du composite *activité*,
   (b) critères : fonctionnalité,
   (c) exemple : paiement/achat ;
7. **lieu/espace**
   (a) relation portion/masse sans séparabilité. Le composant *lieu* ne peut être séparé du composite *espace*. Pour exprimer la situation géographique d'un lieu,
   (b) critères : similarité, simultanéité,
   (c) exemple : oasis/désert.

Annexe B

L'ontologie Bigre en OWL

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.info.fundp.ac.be/~bigre/ontologies/services.eng.owl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
  xml:base="http://www.info.fundp.ac.be/~bigre/ontologies/services.eng.owl">
  <owl:Ontology rdf:about="">
    <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >1.0 RC1</owl:versionInfo>
    <owl:imports rdf:resource="http://protege.stanford.edu/plugins/owl/protege"/>
  </owl:Ontology>
  <owl:Class rdf:ID="Service">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="hasServiceType"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Data">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="hasName"/>
        </owl:onProperty>
```

```
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasDataType"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="InputTraitement">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="OperationInput"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="DataInstance">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="isInstanceOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment xml:lang="fr">Un individus InstanceDeDonne est un individus
  qui est utilise comme une instance d'une Donnee. Il doit egalement faire
  partie de la classe instancieDonnee.typeePar.</rdfs:comment>
</owl:Class>
<owl:Class rdf:ID="GUIDescription"/>
<owl:Class rdf:ID="Scenario"/>
<owl:Class rdf:ID="PostCondition">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Condition"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="PropertyInstance">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Par rapport aux autres DataInstance, une PropertyInstance est egalement
  liee a un Service paritculier. On retrouve ce lien via la propriete characterise.
  </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="characterize"/>
```

```
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#DataInstance"/>
</owl:Class>
<owl:Class rdf:ID="OutputTraitement">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="OperationOutput"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Interface"/>
<owl:Class rdf:ID="PreCondition">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Condition"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Property">
  <rdfs:subClassOf rdf:resource="#Data"/>
</owl:Class>
<owl:Class rdf:ID="Operation"/>
<owl:Class rdf:ID="ServiceType">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasScenario"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:InverseFunctionalProperty rdf:ID="presents"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:InverseFunctionalProperty rdf:ID="hasGui"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
```

```
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="realize"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Condition">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="appliesTo"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Call">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Represente l'appel d'une operation</rdfs:comment>
  <owl:versionInfo rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >TODO: Proposition a valider</owl:versionInfo>
</owl:Class>
<owl:Class rdf:ID="Process"/>
<owl:Class rdf:ID="Binding">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="concretize"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#OperationInput">
  <rdfs:subClassOf rdf:resource="#Data"/>
</owl:Class>
<owl:Class rdf:about="#OperationOutput">
  <rdfs:subClassOf rdf:resource="#Data"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="generalize">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="specialize"/>
```

```
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Process"/>
    <rdfs:range rdf:resource="#Process"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="organize">
    <rdfs:domain rdf:resource="#Scenario"/>
    <rdfs:range rdf:resource="#Operation"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="isInstanciedBy">
    <rdfs:range rdf:resource="#Service"/>
    <owl:inverseOf>
      <owl:FunctionalProperty rdf:about="#hasServiceType"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#ServiceType"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="realisePar">
    <rdfs:range rdf:resource="#ServiceType"/>
    <owl:inverseOf>
      <owl:FunctionalProperty rdf:about="#realize"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Process"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="characterizedBy">
    <owl:inverseOf>
      <owl:FunctionalProperty rdf:about="#characterize"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#PropertyInstance"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
    <rdfs:domain rdf:resource="#Service"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#specialize">
    <rdfs:range rdf:resource="#Process"/>
    <owl:inverseOf rdf:resource="#generalize"/>
    <rdfs:domain rdf:resource="#Process"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="onOperation">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Reference l'operation sur laquelle porte l'appel</rdfs:comment>
    <rdfs:domain rdf:resource="#Call"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:range rdf:resource="#Operation"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="producesOperationOutputs">
    <rdfs:domain rdf:resource="#Operation"/>
    <rdfs:range rdf:resource="#OperationOutput"/>
  </owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="withParameters">
  <rdfs:domain rdf:resource="#Call"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:comment xml:lang="fr">Les individus qui representent les parametres
  de l'appel. Pour un Call, cette propriete prends autant de valeurs qu'il
  n'y a d'InputOperation definis pour la valeur prise par la propriete
  onOperation. Chacune de ces valeurs est un individus qui a pour classes
  DataInstance et la classe definie par la propriete hasDataType de la
  valeur prise par sa propriete isInstanceOf.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="requiresOperationInputs">
  <rdfs:domain rdf:resource="#Operation"/>
  <rdfs:range rdf:resource="#OperationInput"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasDataType">
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  <rdfs:domain rdf:resource="#Data"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="producesProcessOutputs">
  <rdfs:domain rdf:resource="#Process"/>
  <rdfs:range rdf:resource="#OutputTraitement"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasDefaultValue">
  <rdfs:domain rdf:resource="#OperationInput"/>
  <rdfs:range rdf:resource="#DataInstance"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="requiresProcessInputs">
  <rdfs:domain rdf:resource="#Process"/>
  <rdfs:range rdf:resource="#InputTraitement"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasScenario">
  <rdfs:domain rdf:resource="#ServiceType"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:range rdf:resource="#Scenario"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="definesProperties">
  <rdfs:domain rdf:resource="#Interface"/>
  <rdfs:range rdf:resource="#Property"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="#hasName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Data"/>
        <owl:Class rdf:about="#Operation"/>
        <owl:Class rdf:about="#ServiceType"/>
```

```
        <owl:Class rdf:about="#Process"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasTextualDescription">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Process"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="isMandatory">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdfs:domain rdf:resource="#OperationInput"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:DatatypeProperty>
<owl:FunctionalProperty rdf:about="#isInstanceOf">
  <rdfs:domain rdf:resource="#DataInstance"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Data"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#concretize">
  <rdfs:domain rdf:resource="#Binding"/>
  <rdfs:range rdf:resource="#Service"/>
  <owl:inverseOf>
    <owl:InverseFunctionalProperty rdf:ID="hasBinding"/>
  </owl:inverseOf>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasServiceType">
  <rdfs:range rdf:resource="#ServiceType"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <owl:inverseOf rdf:resource="#isInstanciedBy"/>
  <rdfs:domain rdf:resource="#Service"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#realize">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Process"/>
  <rdfs:domain rdf:resource="#ServiceType"/>
  <owl:inverseOf rdf:resource="#realisePar"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#characterize">
  <owl:inverseOf rdf:resource="#characterizedBy"/>
  <rdfs:domain rdf:resource="#PropertyInstance"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Service"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="isDocumentedBy">
  <rdfs:comment xml:lang="fr">Reference la documation de ce Traitement.
```

```
    </rdfs:comment>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
    <rdfs:domain rdf:resource="#Process"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="hasUri">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
    <rdfs:domain rdf:resource="#Service"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:about="#appliesTo">
    <owl:inverseOf>
      <owl:InverseFunctionalProperty rdf:ID="conditonnedBy"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Condition"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range rdf:resource="#Operation"/>
  </owl:FunctionalProperty>
  <owl:InverseFunctionalProperty rdf:ID="definesOperations">
    <rdfs:range rdf:resource="#Operation"/>
    <rdfs:domain rdf:resource="#Interface"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:InverseFunctionalProperty>
  <owl:InverseFunctionalProperty rdf:about="#hasGui">
    <rdfs:domain rdf:resource="#ServiceType"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range rdf:resource="#GUIDescription"/>
  </owl:InverseFunctionalProperty>
  <owl:InverseFunctionalProperty rdf:about="#hasBinding">
    <rdfs:domain rdf:resource="#Service"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:range rdf:resource="#Binding"/>
    <owl:inverseOf rdf:resource="#concretize"/>
  </owl:InverseFunctionalProperty>
  <owl:InverseFunctionalProperty rdf:about="#presents">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#ServiceType"/>
    <rdfs:range rdf:resource="#Interface"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  </owl:InverseFunctionalProperty>
  <owl:InverseFunctionalProperty rdf:about="#conditonnedBy">
    <owl:inverseOf rdf:resource="#appliesTo"/>
    <rdfs:range rdf:resource="#Condition"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#Operation"/>
  </owl:InverseFunctionalProperty>
</rdf:RDF>
```

```
<!-- Created with Protege (with OWL Plugin 2.1, Build 284)
http://protege.stanford.edu -->
```