N° d'ordre : 31/012-M/INF

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie HOUARI BOUMEDIENNE

FACULTE D'ELECTRONIQUE ET D'INFORMATIQUE

MEMOIRE

Présenté pour l'obtention du diplôme de MAGISTER

EN: INFORMATIQUE

Spécialité : Système Intelligents et Ingénierie du Logiciel.

Par : BOURAÏ Fouzia

Sujet :

Opérationnalisation d'ontologies par les graphes conceptuels

Soutenu publiquement le 28 juin 2012, devant le jury composé de:

Mr A. GUESSOUM	Professeur à l'U.S.T.H.B	Président de jury
Mme Z. ALIMAZIGHI	Professeur à l'U.S.T.H.B	Directeur de mémoire
Mme L. MAHDAOUI	Docteur à l'U.S.T.H.B	Examinateur
Mr H. AZZOUNE	Docteur à l'U.S.T.H.B	Examinateur
Mme H. ALIANE	Chargée de recherche au C.E.R.I.S.T	Invitée

Remerciements

Merci au bon Dieu, le tout puissant.

Je tiens à remercier vivement ma directrice de mémoire Mme Zaia Alimazighui, professeur à l'USTHB, pour ses conseils et orientations. Merci d'avoir accepté d'être mon directeur de mémoire et me guider dans le monde si vaste de la recherche. Qu'elle trouve en ce mémoire l'expression de mon profond respect.

Je remercie tout particulièrement Mme Hassina Aliane, docteur au CERIST, de m'avoir orientée et corrigée mon travail. Qu'elle trouve en ce mémoire l'expression de ma profonde gratitude et mon respect infini.

Mes vifs remerciements et respects s'adressent aux membres du jury Mr Ahmed Guessoum, Mme Latifa Mahdaoui et Mr Hamid Azzoune pour m'avoir fait l'honneur d'accepter de juger ce modeste travail.

Mes vifs remerciements vont aussi à ma famille et plus particulièrement ma mère. Je les remercie pour leur soutien de tous les instants.

Mes chaleureux remerciements vont également à mes amies pour leurs soutiens et encouragements.

Sommaire

Table des Figures
Introduction générale i
Chapitre I : Ingénierie des ontologies et Modèles de connaissances 4
1.. Introduction 4
2.. Qu'est-ce que l'ontologie 5
3.. Evolution vers les ontologies 7
3.1. Composantes de l'ontologie 8
3.1.1. Le concept 8
3.1.2. La relation 9
3.1.3. Les individus 9
3.1.4. Les propriétés des concepts et des relations 9
4.. Cycle de vie d'une ontologie 11
5.. Processus de construction d'une ontologie 12
5.1. La conceptualisation 13
5.2. L'ontologisation 13
5.3. L'opérationnalisation 13
6.. Méthodologies de construction d'ontologies 14
6.1. La méthode de Uschold et King 14
6.2. La méthode TOVE 14
6.3. La méthode METHONTOLOGY 15
7.. Ontologies et modèles de connaissance 15
7.1. Les frames 16
7.2. Logiques de description 17
7.2.1. Sémantique des concepts et des rôles 18
7.2.2. L'inférence dans les logiques de description 18
7.3. Les graphes conceptuels 19
7.3.1. Le niveau terminologique 19
7.3.2. Le niveau assertionnel 21
7.3.3. Les opérations 22
7.3.4. Sémantique logique des graphes conceptuels 24
7.3.5. Les graphes emboîtés 25
7.3.6. La SG-family 26
7.3.7. Implémentation des graphes conceptuels sur machine 28
8.. Langages de représentation des ontologies 29
9.. Outils de développement des ontologies 31
10. Moteurs d'inférences 32
11. Conclusion 33
Chapitre II : L'opérationnalisation 34
1.. Introduction 34
2.. Les limites des ontologies en termes de raisonnement et d'inférence 34
3.. L'opérationnalisation et la représentation de connaissances 36
3.1. Le choix du modèle de représentation adéquat 36

3.2. L'opérationnalisation et les graphes conceptuels 37
4.. L'opérationnalisation des ontologies 37
4.1. Opérationnaliser une ontologie au sein d'un SBC 38
4.2. Opérationnaliser une ontologie dans le modèle des GCs 39
4.2.1. Scénarii d'usage et contextes d'usage 41
4.2.2. Etapes d'opérationnalisation de l'ontologie par les GCs 42
5.. Utilisation des ontologies opérationnelles pour raisonner 43
6.. Conclusion 44
Chapitre III : Une approche d'opérationnalisation d'ontologies par les graphes conceptuels ... 45
1.. Introduction 45
2.. Codification de l'ontologie en CoGXML 46
2.1. Forme générale d'un fichier BCGCT 46
2.1.1. Forme BCGCT d'un support 46
2.1.2. Forme BCGCT d'un graphe conceptuel 47
2.1.3. Forme BCGCT d'une règle 47
2.2. Structure d'un fichier CoGXML 48
3.. Une approche pour l'opérationnalisation d'ontologies fondée sur les graphes conceptuels 52
3.1. Création d'une ontologie dans le formalisme des graphes conceptuels 52
3.1.1. Phase de conceptualisation 53
3.1.2. Phase d'ontologisation 55
3.1.3. Phase d'opérationnalisation 56
3.1.3.1. Représentation d'une ontologie par les graphes conceptuels 56
3.1.3.2. Les ontologies comme bases de connaissances 58
3.2. Importation d'une ontologie existante 66
3.2.1. L'algorithme de transformation d'une ontologie OWL en CoGXML 66
3.2.2. Schema général de l'algorithme de transformation 69
4.. Utilisation des ontologies opérationnelles pour raisonner 71
4.1. Problème sous forme d'un graphe conceptuel 72
4.1.1. La règle (R) sous forme d'un graphe conceptuel 73
4.1.2. La confirmation (C) sous forme d'un graphe conceptuel 74
4.2. Mécanisme d'inférence 74
4.3. Solution sous forme d'un graphe conceptuel 76
4.3.1. Déduction de nouvelles connaissances à partir d'une règle 76
4.3.2. Confirmation de connaissances 78
5.. Portabilité et connexion avec les outils existants 80
6.. Implémentation 80
6.1. Choix technique 81
6.2. Description du prototype 81
7.. Conclusion 82
Conclusion & perspectives 83
Références Bibliographique 85
Annexe A : Grammaire du format BCGCT de CoGITaNT 5 92
Annexe B : Correspondance des éléments RDF et GCs 94

Table des Figures

Figure 1. Le spectre d'ontologie [Góm, 04] 8
Figure 2: Le cycle de vie d'une ontologie [Ban, 07] 12
Figure 3. Construction d'une ontologie opérationnelle d'après Fürst [Für, 04b] 13
Figure 4. Processus principal de modèle évolué par Uschold et King 14
Figure 5. Structure frame, slot et facette 16
Figure 6. Exemple de T-box et de A-box en LD 18
Figure 7. Exemple d'une hiérarchie de types de concepts [Für, 04b] 20
Figure 8. Exemple d'une hiérarchie de types de relations [Für, 04b] 21
Figure 9. Exemple d'un graphe conceptuel [Für, 04b] 22
Figure 10. Joint interne [Mug, 96] 23
Figure 11. Exemple d'un graphe emboité 26
Figure 12. Exemple d'un couple de lambda-abstractions 27
Figure 13. Exemple de règle [Für, 04b] 27
Figure 14. Contrainte négative [Für, 04b] 28
Figure 15. Langages d'ontologies basés Web [Corcho 03] 30
Figure 16. Schéma général d'opérationnalisation d'une ontologie en vue de son utilisation au sein d'un SBC 39
Figure 17. Schéma détaillé d'opérationnalisation d'une ontologie en vue de son utilisation au sein d'un SBC [Für, 04b] 41
Figure 18. Schéma d'application des axiomes 44
Figure 19. Étapes de construction d'une ontologie opérationnelle 53
Figure 20. La phase de conceptualisation 54
Figure 21. L'ontologie dans le modèle de graphe conceptuel 58
Figure 22. L'ontologie comme base de connaissances 58
Figure 23. La définition du concept Prof_d_anglais par un GC 61
Figure 24. Hiérarchie de type concept de l'être humain 62
Figure 25. La symétrie de la relation Frère-de 63
Figure 26. La transitivité de la relation Ancêtre-de 63
Figure 27. Définition de la relation Grand-Pére-De 64
Figure 28. Hiérarchie de type relation de l'être humain 64
Figure 29. Exemple d'un graphe conceptuel contenant des individus 65
Figure 30. Exemple des graphes de faits 66
Figure 31. La hiérarchie des éléments OWL et RDF/RDFS 67
Figure 32. La hiérarchie des éléments de graphes conceptuels et RDF/RDFS 69
Figure 33. Algorithme de transformation OWL/ CoGXML 71
Figure 34. L'inférence à base des ontologies 72
Figure 35. Exemple d'un graphe de règle 73
Figure 36. Exemple d'un graphe de confirmation 74
Figure 37. Exemple d'inférence 77
Figure 38. Algorithme de projection d'une règle (R) sur un graphe de fait (F) 77
Figure 39. Application d'une règle (R) sur un graphe de faits (F) 78
Figure 40. Exemple d'inférence 79

Figure 41. Algorithme de projection d'une requête (C) sur un graphe de faits (F) 79
Figure 42. Application d'une requête (C) sur un graphe de faits (F) 80
Figure 43. Page d'accueil d'OperOGC 82
Figure 60. Grammaire de BCGCT 93
Figure 61. Correspondance des éléments RDF et GCs 94

Résumé

L'objectif du web sémantique est de rendre les ressources du web interprétables par la machine, accessibles et utilisables par des programmes et des agents logiciels. En effet les documents traités par le web sémantique contiennent des informations formalisées pour être traitées automatiquement et non pas interprétées uniquement par les humains.

Les ontologies sont à l'heure actuelle considérées comme une solution intéressante au problème de la sémantique en Ingénierie des Connaissances. Leur développement croissant en Intelligence Artificielle vient de leur intérêt pour associer du sens à des ressources textuelles, afin de localiser et gérer des connaissances dans le Web.

Ce travail est une contribution à l'opérationnalisation des ontologies par l'utilisation des graphes conceptuels comme modèle de représentation de connaissances. Nous proposons une approche qui permet la création d'ontologies basées sur les graphes conceptuels ainsi que l'importation et la transformation d'ontologies existantes décrites en OWL.

L'ontologie devient une base de connaissances sur laquelle, on peut effectuer des raisonnements. Dans ce travail, nous nous intéressons en particulier au raisonnement déductif qui devient possible grâce aux opérateurs de projection du formalisme des graphes conceptuels. Par ailleurs, nous avons implémenté un prototype en utilisant la bibliothèque Cogitant qui permet de représenter des connaissances et raisonner avec des graphes conceptuels.

MOTS-CLÉS *: Opérationnalisation, Ingénierie Ontologique, Graphes Conceptuels, Intelligence Artificielle, Ingénierie des Connaissances, Raisonnement Déductif.*

Introduction générale

1. Contexte général

L'objectif du web sémantique est de rendre les ressources du web interprétables par la machine, accessibles et utilisables par des programmes et des agents logiciels. En effet les documents traités par le web sémantique contiennent des informations formalisées pour être traitées automatiquement et non pas interprétées uniquement par les humains.

Les ontologies sont à l'heure actuelle considérées comme une solution intéressante au problème de la sémantique en Ingénierie des Connaissances. Leur développement croissant en Intelligence Artificielle vient de leur intérêt pour associer du sens à des ressources textuelles, afin de localiser et gérer des connaissances dans le Web.

« Une ontologie est une spécification explicite d'une conceptualisation.» [Gru, 93]. Cette définition, qui reste la plus citée dans le domaine de l'informatique, permet d'une part, de voir l'ontologie comme une représentation abstraite des connaissances, et d'autre part, précise le fait que l'ontologie doit être formelle, exprimée sous forme d'une logique pouvant être manipulée par la machine.

La création d'ontologies regroupe un ensemble de phases [Für, 04b]. Une fois l'ontologie définie (*phase conceptualisation*) et formalisée (*phase ontologisation*) il reste à lui donner l'empreinte opérationnelle (*phase opérationnalisation*). Cette dernière phase consiste à outiller l'ontologie en utilisant des modèles de représentation logique pour l'introduire dans un système à base de connaissances dans le but d'effectuer des raisonnements. Cette opération est cruciale et nécessite un modèle de représentation de connaissances. Notre travail se situe dans ce contexte où nous nous intéressons à l'opérationnalisation des ontologies.

2. Problématique et objectifs

L'ingénierie des ontologies (IO) est née de la volonté de diversifier les applications des systèmes à base de connaissances, et de permettre une représentation des connaissances indépendante de ces diverses applications, de manière à assurer sa portabilité d'une application à l'autre. Les ontologies se veulent des représentations de connaissances bâties au niveau conceptuel et permettant l'utilisation de connaissances pour le raisonnement. Elles se doivent donc d'intégrer à la fois les connaissances **terminologiques** et **l'expression de la sémantique** d'un domaine, tout en conservant leur indépendance par rapport aux usages opérationnels des systèmes à base de connaissances.

Les systèmes **d'opérationnalisation des ontologies** existants [Mug, 92, Don, 05] se basent sur des modèles de représentation difficiles à manipuler p. ex. la logique du premier ordre et les logiques de description. En outre, ces modèles ne permettent pas d'introduire facilement les ontologies dans des systèmes à base de connaissances, ce qui rend par la suite leur manipulation opérationnelle très difficile [Für, 04b].

Les **graphes conceptuels**, vu la simplicité de leurs manipulations graphiques pour l'être humain et l'existence des bibliothèques gratuites et complètes facilitant ces manipulations, ont été utilisés comme modèle de représentation de connaissances, nous citons à titre d'exemple les travaux de *Mugnier* [Mug, 96] et de *Sowa* [Sow, 84, 91, 00b].

Nous nous inspirons de ces travaux et nous proposons une approche pour **l'opérationnalisation d'ontologies basées sur le modèle de graphes conceptuels**.

3. Contribution

Nous proposons une approche qui permet la création d'ontologies basées sur les graphes conceptuels ainsi que l'importation et la transformation d'ontologies existantes décrites en OWL. Cette approche est fondée sur un algorithme de transformation d'ontologies basé sur les graphes conceptuels. Une fois l'ontologie opérationnelle est obtenue, celle-ci peut être vue comme une base de connaissances à laquelle nous ajoutons un mécanisme d'inférence permettant un raisonnement déductif. Deux stratégies sont possibles:

- La confirmation de connaissances : cette stratégie part des données disponibles, et tente d'arriver vers le but : grâce à la projection de graphe de requête de confirmation sur un graphe de faits qui permet d'extraire ou de juger l'existence d'une connaissance ou de la nier;
- la déduction de nouvelles connaissances à l'aide de règles : grâce à la projection de graphe de règles sur un graphe de faits qui permet d'inférer de nouvelles connaissances.

Nous avons validé notre approche par le prototype *OPERationnalisation d'Ontologies par les Graphes Conceptuels* (*OperOGC)*, qui est fondé sur la bibliothèque COGITANT des graphes conceptuels. Cet outil est compatible avec d'autres outils existants p. ex. *Protégé2000.* Cette compatibilité permet d'importer des ontologies existantes sous format OWL.

4. Organisation du mémoire

Ce mémoire est structuré en trois grands chapitres :

Le premier chapitre porte sur l'ingénierie ontologique et différents formalismes de représentation de connaissances, nous commençons par les origines du terme 'ontologie', ses différentes définitions et nous présentons par la suite les principales méthodes de construction et les langages utilisés pour la représentation des ontologies.

Ce chapitre décrit les différents formalismes de représentation de connaissances (leurs fondements théoriques, spécificités, etc.), nous nous intéresserons en particulier aux graphes conceptuels qui permettent de représenter les ontologies.

Le deuxième chapitre, présente les limites des ontologies en termes de raisonnement et d'inférence qui ont amené à la création des ontologies opérationnelles. Dans ce chapitre nous présentons des définitions sur l'opérationnalisation d'ontologies.

Le troisième chapitre est consacré à la présentation de l'approche d'opérationnalisation que nous proposons. Nous décrivons les étapes de construction d'une ontologie opérationnelle basées sur les graphes conceptuels, et nous proposons un algorithme de transformation d'ontologie qui permet d'importer les ontologies existantes. Notre approche est complétée par un mécanisme d'inférence qui permet la déduction suivant deux stratégies. De là, un prototype d'édition et d'opérationnalisation d'ontologies a été mis en œuvre, en se basant sur l'approche proposée afin de valider notre approche.

En conclusion, nous revenons sur les points importants de ce travail et quelques perspectives qui découlent de cette étude.

Chapitre I : Ingénierie des ontologies et Modèles de connaissances

1. Introduction

Les **ontologies** sont apparues dans le domaine de l'Ingénierie des Connaissances (IC), et plus largement en Intelligence Artificielle (IA), comme une approche de modélisation et de représentation des connaissances. Elles sont introduites dans le cadre des démarches d'acquisition des connaissances pour les Systèmes à Base de Connaissances (SBC) et ont évolué vers la représentation des connaissances et leur organisation.

Durant la dernière décennie, l'utilisation des ontologies pour la gestion des connaissances s'est avérée avantageuse dans le domaine de la recherche en intelligence artificielle où la gestion des connaissances est basée sur la représentation des connaissances de façon à simuler le raisonnement humain afin de modéliser les connaissances d'une façon utilisable par la machine [Bar, 97].

Ainsi, les ontologies sont utiles pour construire des systèmes de gestion des connaissances. Elles permettent la représentation des connaissances et la modélisation des raisonnements qui sont des caractéristiques fondamentales des SBCs. Elles ont pour rôle de fournir un système de concepts fondamentaux du domaine, afin de construire une base de connaissances.

Le terme **ingénierie des ontologies** a été proposé par *Mizouguchi* [Miz, 97] pour désigner un nouveau champ de recherche ayant pour but la construction de systèmes informatiques tournés vers le contenu, et non vers les mécanismes de manipulation de l'information [Miz, 97]. Ce soubassement formel, de connaissances indépendante de leurs domaines d'applications, pouvant servir à la génération de modèles de représentation de connaissances et de modèles de raisonnement sur ces connaissances pour des SBC, assurant ainsi la portabilité des applications, leurs interopérabilités et leurs réutilisations [Bac, 00].

Cet engouement est motivé par le fait que les ontologies sont un moyen efficace pour la gestion et le partage des connaissances d'un domaine particulier entre personnes et\ou systèmes.

2. Qu'est-ce que l'ontologie

Ontologie est un terme philosophique qui signifie d'après le grec ancien la science ou la théorie de l'être dans notre univers « être pour ôn ou onton et étude ou science pour logos» [Enc, 00]. La première transition réelle de la philosophie vers l'intelligence artificielle était en 1980 par *John McCarthy* [Mcc, 80, Psy, 07]. Le terme a évolué, pendant la convergence de l'ingénierie des connaissances, la modélisation conceptuelle, et la modélisation du domaine. Par la suite, le terme a été introduit en intelligence artificielle au début des années 1990, à l'occasion de l'émergence de l'ingénierie des connaissances, comme réponses aux problématiques de représentation et de manipulation des connaissances au sein des systèmes informatiques.

Définitions

Dans la littérature on trouve plusieurs définitions des ontologies :

- *Neches* et ses collègues [Nec, 91] définissent l'ontologie comme : « *Une ontologie définit les termes et les relations de base du vocabulaire d'un domaine ainsi que les règles qui indiquent comment combiner les termes et les relations de façon à pouvoir étendre le vocabulaire* ». Cette définition permet d'élaborer une ontologie en repérant les termes de base et les relations entre eux, en identifiant les règles servant à les combiner et en fournissant des définitions de ces termes et de ces relations.
- La définition de *Gruber* [Gru, 93] est la plus citée dans la littérature : « *Une ontologie est une spécification* ***explicite*** *d'une* ***conceptualisation*** ». Le terme *'conceptualization'* réfère à un modèle abstrait d'un certain phénomène de la réalité et qui permet d'identifier les concepts pertinents de ce phénomène. Le terme *'explicit'* signifie que le type des concepts utilisés ainsi que les contraintes sur leur emploi, sont réellement définis d'une manière claire et précise.
- En 1997, *Borst* [Bor, 97] modifie légèrement la définition de *Gruber* [Gru, 93] en énonçant que les ontologies sont définies comme étant : « *Une ontologie est une spécification* ***formelle*** *d'une conceptualisation* ***partagée*** ». Cette définition précise d'une part, le fait que l'ontologie doit être formelle, exprimée sous forme d'une logique pouvant être manipulée sur machine, et d'autre part, le fait qu'elle doit être 'partagée'.

- La définition de *Studer* et ses collègues [Stu, 98] regroupe celle de *Borst* [Bor, 97] et *Gruber* [Gru, 93] en énonçant que les ontologies sont définies comme étant : « *Une ontologie est une spécification* ***formelle*** *et* ***explicite*** *d'une* ***conceptualisation partagée*** ». *Studer* explique les mots clés de cette définition ainsi :
  - ✓ Spécification explicite : signifie que le type des concepts et les contraintes sur leurs utilisations sont explicitement définies ;
  - ✓ Spécification formelle : se réfère au fait que la spécification (l'ontologie) doit être lisible et compréhensible par une machine ;
  - ✓ Partagé : signifie que les connaissances représentées sont partagées par une communauté (un groupe) ;
  - ✓ Conceptualisation: se réfère à un modèle abstrait d'un certain phénomène du monde basé sur l'identification des concepts pertinents de ce phénomène.
- En 1995, *Guarino* et *Giaretta* [Gua, 95] affinent la définition de *Gruber* en considérant les ontologies comme *«des spécifications partielles et formelles d'une conceptualisation »*. Pour
- Dans le même ordre d'idée, en 1997 *Swartout* et ses collègues [Swa, 97] au sein du projet SENSUS définissent l'ontologie par: « *Une ontologie est un ensemble de termes hiérarchiquement structurés, conçu afin de décrire un domaine qui peut être utilisé comme un squelette de base pour les bases de connaissances* ». Dans le même ordre d'idées, *Bernaras* [Ber, 96] propose la définition suivante: « *Une ontologie fournit le moyen pour décrire d'une manière explicite la conceptualisation des connaissances représentées dans les bases de connaissances* ». Cette définition propose l'extraction d'ontologie à partir d'une base de connaissances, ce qui reflète l'approche utilisée par les auteurs pour construire une ontologie.
- En 2000 *Aussenac-Gilles* et ses collègues [Aus, 00] énoncent que : *« Une ontologie organise dans un réseau des concepts représentant un domaine. Son contenu et son degré de formalisation sont choisis en fonction d'une application ».* Cette définition souligne la dépendance entre le degré de formalisation de l'ontologie et son contenu avec l'application dans laquelle elle va être utilisée.

Pour conclure cette section, nous pouvons dire que les ontologies visent à capturer les connaissances consensuelles de manière générique, et qu'elles peuvent être réutilisées et partagées au-delà des applications logicielles et des groupes de personnes.

3. Evolution vers les ontologies

Dans cette section nous définissons les différentes formes d'ontologie et les relations qui existent entre elles, afin de connaitre la nécessité qui a conduit au besoin d'utiliser les ontologies dans le domaine informatique. *Lassila* et *McGuinness* [Las, 01] proposent une classification des ontologies selon la richesse de leur structure interne.

- **Méta-données :** c'est une donnée servant à définir ou décrire une autre donnée.
- **Annotation :** une annotation est une note explicative accompagnant un texte, ou une quelconque marque de lecture portée sur un document.
- **Vocabulaire contrôlé :** c'est un ensemble de termes définis par une communauté afin de pouvoir labelliser des contenus p. ex. le glossaire d'un livre.
- **Taxonomie:** c'est la hiérarchisation simple du vocabulaire contrôlé. Cette hiérarchisation correspond souvent à une spécialisation. Il existe donc un lien précis entre un terme du vocabulaire et ses enfants. Ce lien donne un sens supplémentaire, une signification (d'un vocabulaire contrôlé, on passe à un vocabulaire organisé).
- **Thésaurus:** c'est une liste organisée de termes représentant les concepts d'un domaine de la connaissance, p. ex. les termes « *véhicule*, *navire »* sont reliés entre eux par des relations de synonymie (terme équivalent), de hiérarchie (terme générique et terme spécifique) et d'association (terme associé).
- **Ontologie :** une ontologie correspond à ensemble structuré des termes et concepts représentant le sens d'un champ d'informations, que ce soit par les métadonnées d'un espace de noms, ou les éléments d'un domaine de connaissances. L'ontologie constitue en soi un modèle de données représentatif d'un ensemble de concepts dans un domaine, ainsi que des relations entre ces concepts. Elle est employée pour raisonner à propos des objets du domaine concerné.

Dans [Usc, 96], *M. Uschold* considère que les ontologies varient suivant trois dimensions:

- Le **degré de formalisme** de la représentation, qui varie d'une façon continue depuis l'informel jusqu'au formel.
- L'**objectif opérationnel**, sous forme de communication entre utilisateurs, interopérabilité entre systèmes, application à un problème d'ingénierie comme la réutilisabilité de composants et la résolution de problèmes.
- Le **sujet**, représenté par le domaine de connaissance, connaissances de raisonnement et connaissances liées au modèle de représentation.

La figure 1 montre une échelle des degrés de formalisation et du niveau d'engagement sémantique en représentation de connaissances. Cette échelle va du vocabulaire contrôlé aux ontologies lourdes. Entre les deux, des glossaires aux ontologies légères, les représentations de connaissances offrent des possibilités croissantes d'intégrer la sémantique du domaine, en spécifiant toutefois toujours la terminologie.

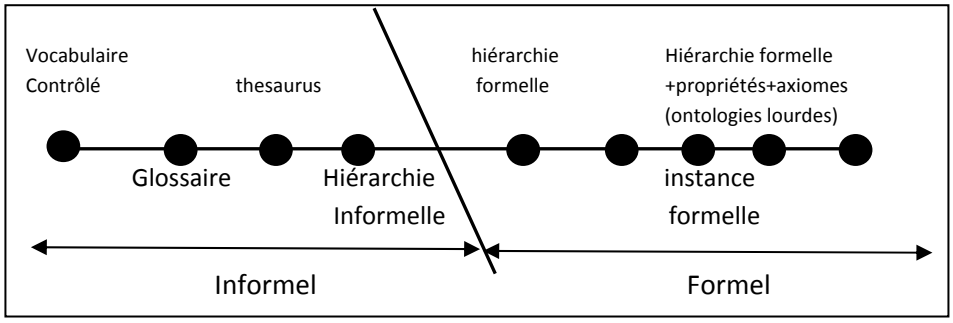

Figure 1. Le spectre d'ontologie [Góm, 04]

Les langages de représentation et les outils d'édition ontologique permettent la construction d'ontologies légères qui suffisent à représenter la terminologie d'un domaine et ses propriétés sémantiques minimales. Cependant, il s'avère nécessaire d'intégrer aux ontologies des connaissances pour préciser la façon dont les concepts sont manipulés dans le domaine. *A. Maedche* et *S. Staab* [Mae, 01] intègrent les axiomes aux ontologies comme des objets liés aux concepts et relations, de manière à apparaître comme des propriétés.

3.1. Composantes de l'ontologie

Une ontologie est une représentation de connaissances au niveau conceptuel [Bac, 00], l'ontologie fournit un vocabulaire commun d'un domaine et définit de façon plus ou moins formelle, le sens des termes et les relations entre ces derniers.

Les connaissances traduites par une ontologie sont à véhiculer à l'aide des éléments suivants: concepts, relations, instances (individus), fonctions et axiomes. Ces notions nécessitent cependant d'être explicitées [Psy, 03, Psy, 03a, Psy, 07].

3.1.1. Le concept

Le concept représente un ensemble d'objets, notions, idées et leurs propriétés communes. Il est appelé aussi class de l'ontologie dans certain travaux. Un concept peut être divisé en trois parties :

- Un **terme** (ou plusieurs), est un élément lexical qui exprime le concept en langue naturelle, il peut admettre des synonymes ;

- Une **notion**, également appelée intension du concept, contient la sémantique du concept, exprimée en termes de propriétés ;
- Un **ensemble d'objets** également appelé extension du concept, regroupe les objets (instances du concept) manipulés par le concept.

Ces concepts selon *Psyché* [Psy, 07] correspondent aux abstractions pertinentes d'un segment de la réalité, retenues en fonction des objectifs qu'on se donne et de l'application envisagée pour l'ontologie.

3.1.2. La relation

La relation représente le lien conceptuel pouvant exister entre les concepts. Une relation est caractérisée par un terme, une extension qui est l'ensemble des tuples d'instances liés par cette relation et une intension qui spécifie d'une part les concepts pouvant être liés par cette relation et d'autre part la sémantique d'usage de la relation [Für, 04a].
On peut dire aussi que les relations traduisent les associations (pertinentes) existant entre les concepts présents dans le segment analysé de la réalité [Psy, 03a, Psy, 07].
Les relations sont organisées de manière hiérarchisée à l'aide de la propriété de subsomption.

3.1.3. Les individus

Les individus constituent la définition extensionnelle de l'ontologie, elles représentent des instances de concepts.

L'ontologie admet d'autre composants supplémentaires comme :

- Les **rôles** selon *Sowa* [Sow, 84], une entité est caractérisée par un rôle qu'elle joue dans sa relation à une autre entité p. ex. les rôles de type *Humain* peuvent être mère, employé ou piéton.
- Les **fonctions** sont des cas particuliers de relations dans lesquelles le $n^{ième}$ élément de la relation est unique pour les n-1 éléments précédents p. ex. mère-de et carré [Góm, 99].
- Les **axiomes** constituent des assertions acceptées comme vraies.
- Les **propriétés** on différencie en général les attributs qui sont des propriétés simples (âge, nom, etc.) et les rôles (relations sémantiques).

3.1.4. Les propriétés des concepts et des relations

Dans la section précédente nous avons vu les composants d'une ontologie en général. Ce paragraphe permet de détailler les concepts, leurs propriétés ainsi que les relations.

1) Les propriétés intrinsèques d'un concept

Les propriétés intrinsèques d'un concept sont des propriétés internes qui permettent de donner une spécificité aux concepts.

- la *généricité* : un concept est générique s'il n'admet pas d'extension p. ex. la vérité.
- *l'identité* : un concept porte une propriété d'identité si cette propriété permet de conclure quant à l'identité de deux instances de ce concept. Cette propriété peut porter sur des attributs du concept ou sur d'autres concepts. Par exemple, le concept d'étudiant porte une propriété d'identité liée au numéro de l'étudiant, deux étudiants étant identiques s'ils ont le même numéro.
- la *rigidité* : un concept est rigide si toute instance de ce concept en reste instance dans tous les « mondes » possibles, c'est-à-dire que si une instance de ce concept cesse d'en faire partie, c'est qu'elle cesse d'exister. Par exemple, humain est un concept rigide, mais étudiant est un concept non rigide.
- *l'anti-rigidité* : un concept est anti-rigide si toute instance de ce concept est définie par son appartenance à l'extension d'un autre concept p. ex. *Etudiant* qui est avant tout un *Humain*.

2) Les propriétés portant entre les concepts

Les concepts manipulés dans un domaine de connaissances sont organisés au sein d'un réseau de concepts liés par des propriétés conceptuelles.

- *L'abstraction* : si toute instance de ce concept est aussi instance d'un de ses concepts fils p. ex. le concept *Homme* fils du concept *Humain*, le concept *Humain* est abstrait.
- La *subsomption* : un concept C_1 subsume C_2 si toute propriété sémantique de C_1 est aussi une propriété sémantique de C_2, c'est-à-dire C_1 est plus spécifique que C_2. La subsomption sert à la hiérarchisation de l'ensemble des concepts de l'ontologie [Für, 04b].
- *L'équivalence* : deux concepts sont équivalents s'ils ont une extension identique.
- La *disjonction* (incompatibilité) : deux concepts sont disjoints si leurs extensions sont disjointes p. ex. *Homme* et *Femme.*
- La *dépendance* : un concept C_1 est dépendant d'un concept C_2 si pour toute instance de C_1 il existe une instance de C_2 qui ne soit ni partie ni constituant de l'instance de C_2 p. ex. *Parent* est un concept dépendant de *Enfant*, car l'existence d'un parent suppose celle d'un enfant.

La formalisation de ces cinq propriétés permet leur traduction dans des langages de représentation d'ontologie (opérationnelles ou non). Mais ça n'empêche pas de définir des

notions de façon non formelle, à un niveau conceptuel, afin de valider l'aspect conceptuel de ces propriétés et pour pouvoir les utiliser auprès des experts lors de la conceptualisation.

3) Les propriétés intrinsèques d'une relation

Les propriétés intrinsèques d'une relation sont des propriétés internes, qui permettent de donner une spécificité aux relations.

- *Algébriques* : symétrie, réflexivité, transitivité, anti-symétrie, anti-réflexivité;
- *Cardinalité* : il s'agit du nombre possible de relations de ce type pouvant exister entre les mêmes concepts (instances de concept) p. ex. une *Pièce* a au moins une *Porte*.

4) Les propriétés liant deux relations

Les relations manipulées dans un domaine de connaissances sont organisés au sein d'un réseau de relations liées par des propriétés conceptuelles.

- *L'incompatibilité* : deux relations sont incompatibles si elles ne peuvent lier les mêmes instances de concepts p. ex. *fils-de* et *père-de* [Für, 04b];
- *L'inverse* : deux relations binaires sont inverses l'une de l'autre si, quand l'une lie deux instances I1 et I2, l'autre lie I2 et I1 p. ex. *fils-de* et *père-de* [Für, 04b];
- *L'exclusivité* : deux relations sont exclusives si quand l'une lie des instances de concepts, l'autre ne lie pas ces instances. L'exclusivité entraîne l'incompatibilité p. ex. l'appartenance et la non appartenance sont exclusifs [Für, 04b].

Ces propriétés des concepts et relations complètent la sémantique de l'ontologie, au sens où elles contribuent à préciser les liens et différences entre les primitives cognitives du domaine de connaissance [Für, 04b].

4. Cycle de vie d'une ontologie

Les ontologies étant destinées à être utilisées comme des composants logiciels dans des systèmes répondant à des objectifs opérationnels différents, selon *Fürst* [Für, 02] leur développement doit s'appuyer sur les mêmes principes que ceux appliqués en génie logiciel. En particulier, les ontologies doivent être considérées comme des objets techniques évolutifs et possédant un cycle de vie qui nécessite d'être précisé. Dans ce contexte, les activités liées aux ontologies sont d'une part des activités de gestion de projet (planification, contrôle, assurance qualité), et d'autre part des activités de développement (spécification, conceptualisation, formalisation), s'y ajoutent des activités transversales de support telles que l'évaluation, la documentation et la gestion de la

configuration [Bla, 98]. Un cycle de vie inspiré du génie logiciel est présenté dans [Für, 02] ainsi dans les travaux de *Psyché* [Psy, 07]. Il comprend une étape initiale de détection et de spécification des besoins qui permet notamment de cerner précisément le domaine de connaissances, une étape de conception qui se subdivise en trois phases, une étape de déploiement et de diffusion, une étape d'utilisation, une étape incontournable d'évaluation et enfin une sixième étape consacrée à l'évolution et à la maintenance du modèle. Après chaque utilisation significative, l'ontologie et les besoins doivent être réévalués et l'ontologie peut être étendue et si nécessaire en partie reconstruite. La validation du modèle de connaissances est au centre du processus et se fait de manière itérative [Ban, 07]. *Gómez-Pérez* et ses collègues [Góm, 97] insistent sur le fait que les activités de documentation et d'évaluation sont nécessaires à chaque étape du processus de construction.

L'évaluation précoce permettant de limiter la propagation d'erreurs. Le processus de construction peut être intégré au cycle de vie d'une ontologie comme l'indique la figure 2.

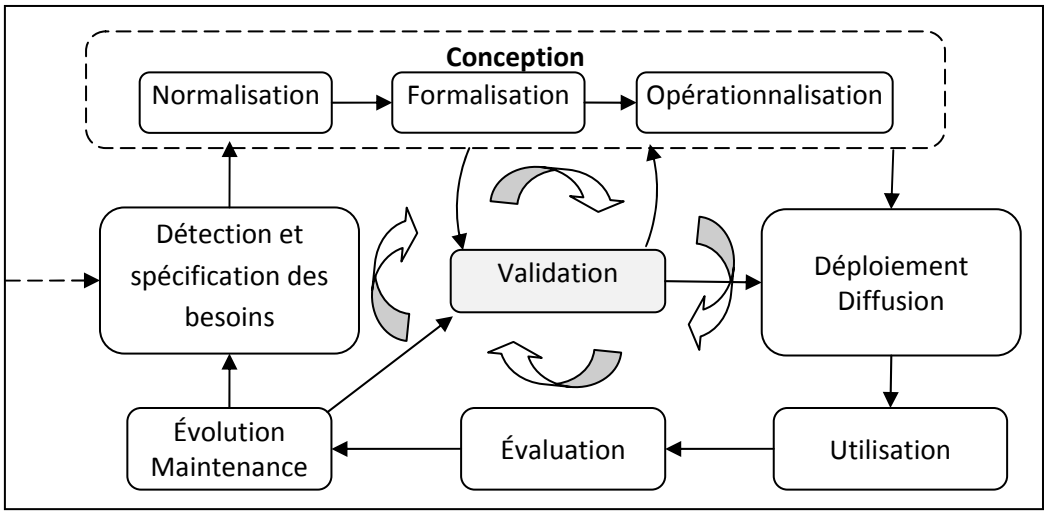

Figure 2: Le cycle de vie d'une ontologie [Ban, 07]

Loin d'une méthode concrète ou cadre méthodologique pour construire une ontologie, il en ressort que la phase de construction est un enchainement de trois étapes : conceptualisation, formalisation, opérationnalisation. L'étape de formalisation peut être complétée par une étape d'intégration au cours de laquelle une ou plusieurs ontologies vont être importées et réutilisées dans l'ontologie à construire.

5. Processus de construction d'une ontologie

Le processus de construction est une collaboration qui réunit des experts du domaine de connaissance, des ingénieurs de la connaissance et les futurs utilisateurs de l'ontologie. Cette

collaboration ne peut être fructueuse que si les objectifs du processus ont été clairement définis, ainsi que les besoins qui en découlent.

L'étape de construction est décomposée en trois (03) phases: conceptualisation, ontologisation et opérationnalisation [Für, 04b]. La figure 3 illustre l'enchaînement de trois phases permettant de passer des données brutes à l'ontologie opérationnelle.

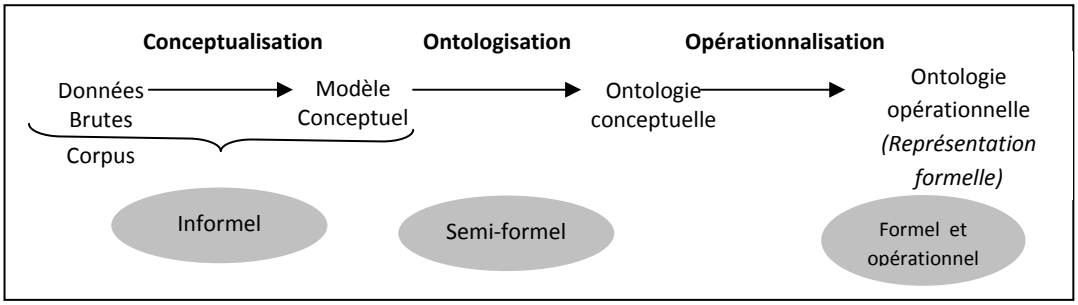

Figure 3. Construction d'une ontologie opérationnelle d'après Fürst [Für, 04b]

Les données brutes, constituant un corpus exprimé a priori en langage naturel, intégrant toutes les connaissances du domaine que l'on souhaite formaliser. Dans ce qui suit nous présenterons les trois phases du processus de construction d'une ontologie.

5.1. La conceptualisation

La conceptualisation consiste à identifier précisément, à partir du corpus, les objets conceptuels propres au domaine considéré, les relations pouvant lier ces concepts et la sémantique avec ces relations. Ce travail doit être mené par un expert du domaine (ou un groupe d'experts), assisté par un ingénieur de la connaissance, apportant son expertise des paradigmes de représentation des connaissances en machine pour aider à la structuration des connaissances. On obtient alors un modèle conceptuel informel car il est exprimé en langage naturel et potentiellement ambigu.

5.2. L'ontologisation

L'ontologisation correspond à la formalisation -autant que possible- du modèle conceptuel obtenu à l'étape précédente. Il s'agit donc de transcrire les connaissances dans un langage de représentation d'ontologie, ce langage doit être aussi générique que possible afin de conserver toutes les connaissances même celles qui ne peuvent être formalisées. Le caractère semi-formel d'une ontologie lui interdit d'être utilisée telle quelle dans un système à base de connaissances.

5.3. L'opérationnalisation

L'opérationnalisation consiste à formaliser complètement l'ontologie conceptuelle obtenue précédemment en la codant dans un langage formel et opérationnel de représentation de

connaissances, doté d'un mécanisme d'inférences. La formalisation totale des connaissances peut imposer à ce stade une perte d'information [Für, 01].
Une fois l'ontologie conceptuelle obtenue, l'opérationnalisation est assistée par des règles de traduction dans le langage cible, il existe également des traducteurs automatiques [Für, 04b].

6. Méthodologies de construction d'ontologies

Lors du processus de mise au point d'une ontologie, chaque équipe de développement suit habituellement ses propres principes, ses critères de conception et ses étapes d'élaboration. On entend par méthodologie, les procédures de travail, les étapes, qui décrivent le pourquoi et le comment de la conceptualisation puis de l'artefact construit. L'objectif de cette section est de présenter les principales méthodologies utilisées pour construire les ontologies.

6.1. La méthode de Uschold et King

C'est une méthode basée sur l'expérience de construction d'ontologies dans le domaine de la gestion des entreprises [Usc, 96]. La figure 4 représente le développement d'une ontologie via cette méthodologie qui repose sur les étapes suivantes [Góm, 04] :

- Identifier le rôle et la portée de l'ontologie.
- Créer l'ontologie les étapes de la construction sont: (1) l'identification des concepts et relations, (2) le codage de l'ontologie dans un langage adapté et (3) l'intégration de ces ontologies.
- Evaluer l'ontologie.
- Rédiger une documentation et une trace des actions réalisées lors des différentes phases.

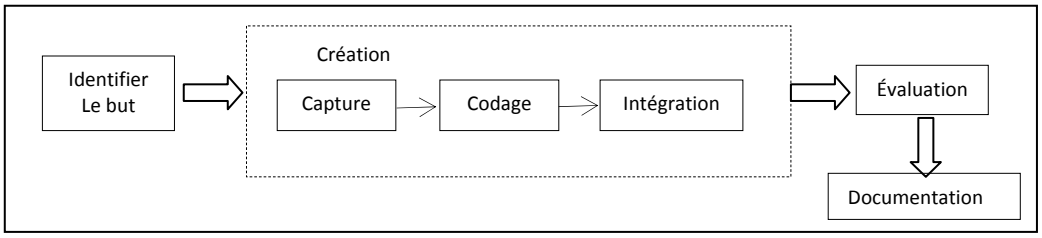

Figure 4. Processus principal de modèle évolué par Uschold et King

6.2. La méthode TOVE

C'est une méthode inspirée du développement des systèmes à base de connaissances en utilisant la logique du premier ordre [Gru, 95]. Elle repose sur les étapes suivantes :

- Capturer et identifier des scénarios de motivations pour clarifier le domaine que l'on investit et les différentes applications dans lesquelles l'ontologie sera employée.

- Formulation des questions de compétences informelles afin de déterminer la portée de l'ontologie. Ces questions et leurs réponses sont utilisées pour extraire les concepts principaux, leurs propriétés et les relations qui existent entre ces concepts.
- Spécifications de la terminologie de l'ontologie identifiée dans l'étape précédente, en utilisant le formalisme de la logique du premier ordre. Les concepts seront des constantes ou bien des variables et des propriétés, les relations seront des prédicats (fonctions).
- Evaluation de la complétude de l'ontologie.

La méthode TOVE, reste spécifiée de façon abstraite ; ni les différentes étapes ni les techniques ne sont décrites en détail.

6.3. La méthode METHONTOLOGY

Cette méthode vise à la construction d'une ontologie au niveau de connaissances [Góm, 97]. Cette méthodologie distingue les étapes suivantes :

- Le cadrage : cette étape consiste à cerner l'étendue de l'ontologie et le domaine à prendre en compte.
- La conceptualisation : le but de cette étape est d'identifier et de structurer les connaissances du domaine en utilisant un ensemble de représentations intermédiaires semi-formelles (des tables et des graphes), faciles à comprendre par les experts du domaine et qui sont indépendants du formalisme à utiliser pour représenter l'ontologie.
- L'implémentation : cette étape consiste à formaliser le modèle conceptuel obtenu dans l'étape précédente par un formalisme de représentation d'ontologie, puis coder l'ontologie dans un langage d'ontologie formel.

Les outils ODE [Bla, 98] et WeODE [Arp. 03] ont été construits pour donner un support technique à METHONTOLOGY. D'autres outils peuvent aussi être utilisés pour construire des ontologies en suivant cette méthodologie, par exemple, Protégé2000, OntoEdit, etc.

7. Ontologies et modèles de connaissance

Représenter des connaissances propres à un domaine consiste à décrire et à coder les éléments de ce domaine pour qu'une machine puisse les manipuler afin de raisonner [Kay, 97].

Des modèles de représentation de connaissances ont été proposés dans l'ingénierie ontologique pour l'exploitation et l'opérationnalisation des ontologies. Dans ce cadre, différents modèles de représentation d'ontologies sont proposés permettant ainsi de spécifier les concepts représentant

les objets d'un domaine, les relations entre ces concepts, et la sémantique de ces primitives de modélisation.

Nous nous intéresserons dans cette section aux modèles de représentation de connaissances liées à l'ingénierie des ontologies qui peuvent être regroupés selon les paradigmes conceptuels qu'ils réifient. Ils ont ainsi distingués :

- Les modèles à base de frames ;
- Les modèles basés sur les logiques de description ;
- Le modèle des graphes conceptuels.

7.1. Les frames

Le modèle des frames a été introduit par *Minsky* en 1975 et a été initialement proposé comme langage de représentation d'ontologies par *T. Gruber* [Gru, 93].

L'idée des Frames est de représenter des catégories d'objets par la description de leurs propriétés et des différents faits rattachés à leur utilisation dans des structures appelées frames, permettant ainsi par leur instanciation de construire les différents acteurs du monde modélisé.

Une ontologie dans le modèle de frame consiste en un ensemble de frames, slots, facettes (Cf. figure.5). Un frame est dans ce contexte un objet nommé, qui est utilisé pour représenter un certain concept dans un domaine donné. Les slots décrivent les propriétés (attributs) d'un concept. Une facette établit une contrainte pour la valeur de l'attribut (de type ou domaine des valeurs possibles) ou bien la valeur sûre ou par défaut (la plus probable) pour cet attribut. Entre les frames, il y a aussi la spécialisation qui donne l'héritage dans les concepts de frame, un frame F1 est plus spécifique qu'un frame F2 si toute instance de F1 est instance de F2.

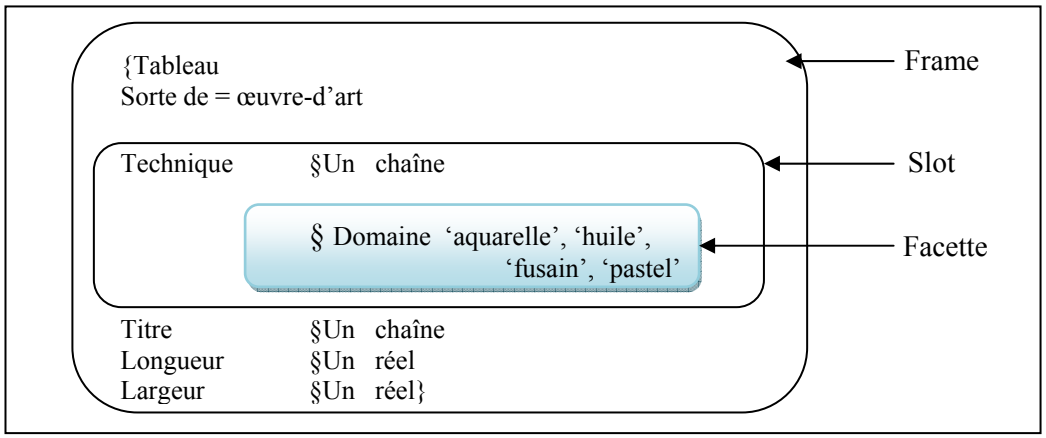

Figure 5. Structure frame, slot et facette

Les représentations par frames offrent deux types de raisonnement, (01) les mécanismes globaux (l'héritage, le filtrage et la classification) qui travaillent sur toute la base et (02) les mécanismes locaux (les réflexes de contrôle et les réflexes de calcul) qui font des inférences au niveau d'un frame ou d'un attribut d'un frame [Dre, 93].

7.2. Logiques de description

Les logiques de description (LDs), appelée aussi les logiques terminologiques, sont des langages formels conçus pour décrire et raisonner sur les connaissances d'un domaine. Elles ont été introduites par *Brachman* en 1979 [Bra, 79]. Elles combinent des représentations intentionnelles et extensionnelles des connaissances [Baa. 91].
Loom [Mac, 91] et CLASSIC [Bra, 91] sont deux exemples de modèles opérationnels implémentant ce modèle. Il est de plus utilisé dans les langages de représentation de connaissance développés pour le Web p. ex. OWL.

La logique de description définit et manipule les concepts, les rôles et les individus. Un concept permet de représenter un ensemble d'individus, tandis qu'un rôle représente une relation binaire entre individus. Un concept correspond à une entité générique d'un domaine d'application et un individu d'une entité particulière, instance d'un concept, à savoir :

- Un concept possède une description structurée qui se construit à l'aide d'un ensemble de constructeurs introduisant les rôles associés au concept et les restrictions attachées à ces rôles. Les restrictions portent généralement sur le co-domaine du rôle, qui est le concept avec lequel le rôle établit une relation, et la cardinalité du rôle, qui fixe le nombre minimal et maximal de valeurs élémentaires que peut prendre le rôle. Les valeurs élémentaires sont des instances de concepts ou bien des valeurs qui relèvent des types de bases.
- Les connaissances sont prises en compte selon deux niveaux (Cf. figure. 6) : la représentation des concepts et des rôles relève du niveau terminologique (**T_box**) ; la description des individus relève du niveau factuel ou niveau des assertions (**A_box**).
- La relation de **subsomption** (notée $\subseteq$) permet d'organiser concepts et rôles par niveaux de généralité : intuitivement, un concept C subsume un concept D si C est plus général que D au sens où l'ensemble d'individus représenté par C contient l'ensemble d'individus représenté par D.

Il existe de nombreux constructeurs permettant de former toute une famille de logiques de description, suivant les constructeurs retenus. La logique de description la plus simple est notée $\mathcal{FL}$ [Nap, 97]. Cette dernière, contient les constructeurs de conjonction de concepts notée

(C1∩C2) et de restriction universelle de concept notée (∀ R C) qui représente l'ensemble des éléments qui ne sont en relation par le rôle *R* qu'avec des éléments du concept *C*. Par ailleurs, la logique de description la plus expressive est notée $\mathcal{SHIQ}$[Hor, 02], qui est une extension de $\mathcal{FL}$ et regroupe un ensemble très riche de constructeurs. Citons par exemple : la cardinalité minimum (maximum) sur les rôles (≥ n *R.C*), (respectivement (≤ n *R.C*)), la disjonction de concepts notée (C1 ∪ C2), et la négation notée (¬C).

T-Box	Homme ⊆ Humain Femme := Humain ∩ (¬Homme) Parant := Humain ∩ (≥ 1 avoir-enfant .Humain)
A-Box	Mouhamed : Parant Avoir-enfant (Mouhamed, Samir)

Figure 6. Exemple de T-box et de A-box en LD

7.2.1. Sémantique des concepts et des rôles

La bonne formalisation des $\mathcal{LD}$ tient à leur sémantique. C'est cette sémantique qui permet d'avoir des mécanismes d'inférence bien fondés pour lesquels on peut démontrer des résultats de cohérence et de complétude. La sémantique des $\mathcal{LD}$ est donnée au moyen d'une interprétation **I**= ($\Delta_{\mathbf{I}}$, $^{\mathbf{I}}$) où $\Delta_{\mathbf{I}}$ est un ensemble d'individus, introduit en guise de domaine d'interprétation, et $^{\mathbf{I}}$ une fonction d'interprétation qui fait correspondre à une expression conceptuelle un sous-ensemble de $\Delta_{\mathbf{I}}$ et à un rôle un sous-ensemble de $\Delta_{\mathbf{I}} \times \Delta_{\mathbf{I}}$.

L'interprétation de la conjonction, de la restriction et de la cardinalité est donnée par :

$\mathbf{I}$ (C1∩ C2) = C1 $^{\mathbf{I}}$ ∩ C2 $^{\mathbf{I}}$

$\mathbf{I}$ ((∀r.C)) = {a ∈ $\Delta_{\mathbf{I}}$ /∀ b ∈ $\Delta_{\mathbf{I}}$ si (a,b) ∈ r $^{\mathbf{I}}$ alors b ∈ C $^{\mathbf{I}}$ }

$\mathbf{I}$ ((≥ n R.C)) = {a ∈ $\Delta_{\mathbf{I}}$ / #{∀b ∈ $\Delta_{\mathbf{I}}$ / (a,b) ∈ r $^{\mathbf{I}}$ et b ∈ C $^{\mathbf{I}}$ }≥ n}

7.2.2. L'inférence dans les logiques de description

Elle s'effectue au niveau terminologique ou assertionnel. Deux principaux problèmes se présentent que le moteur d'inférence essaie de résoudre.

- **L'inférence au niveau terminologique :** quatre principaux problèmes d'inférence se présentent au niveau terminologique :

  1. Satisfiabilité : trouver tous les concepts insatisfiables.
  2. Subsomption : calculer la hiérarchie de subsomption ou taxonomie des concepts.
  3. Trouver les concepts équivalents.

4. Trouver les concepts disjoints.

- **L'inférence au niveau assertionnel :** le niveau assertionnel comprend trois principaux problèmes d'inférence :

1. Cohérence de la ABox : les assertions définies restent cohérentes avec la TBox.
2. Vérification d'instance : vérifier que chaque instance respecte la définition de son concept.
3. Vérification de rôle : vérifier si les rôles de l'instance sont correctement utilisés.

7.3. Les graphes conceptuels

Le modèle des Graphes Conceptuels (GCs) est un formalisme de représentation de connaissances, fondé sur la définition de concepts et de relations entre concepts. Il a été introduit par *John F. SOWA* en 1984. Ce formalisme emploie la représentation graphique comme méthode pour modéliser les connaissances.

Les graphes conceptuels ont été initialement développés pour représenter facilement des connaissances exprimées en langage naturel [Kay, 97, Sow, 00a].

Le modèle des graphes conceptuels se décompose en deux niveaux [Kay, 97, Für, 04a] :

1. Le niveau **terminologique** où sont décrits les concepts, les relations et les instances de concepts, ainsi que les liens de subsomption entre concepts et entre relations.
2. Le niveau **assertionnel** dédié à la représentation des assertions du domaine de connaissances étudié. Les faits sont représentés sous la forme de graphes particuliers (les graphes conceptuels) qui sont construits à partir du vocabulaire conceptuel du niveau terminologique.

L'une des originalités des graphes conceptuels [Mug, 92, Bag, 02] est la représentation séparée des connaissances générales du domaine (concernant les type de concepts et de relations) qui seront regroupées au moyen d'un ensemble de graphes appelés *support* et des connaissances personnalisées des utilisateurs (définitions de faits, de contraintes d'intégrité, de règles d'inférence…) qui seront modélisées par la suite.

7.3.1. Le niveau terminologique

Le niveau terminologique (le support) du modèle comprend trois ensembles disjoints : l'ensemble des types de concepts, l'ensemble des types de relations et l'ensemble des marqueurs individuels.

1) Les types de concepts Tc

Un type de concepts encapsule les caractéristiques communes à plusieurs concepts. Les concepts sont des instances de leur type de concepts. Une relation de spécialisation, notée ≤ c, est définie sur Tc et correspond sémantiquement à la notion « is-a » : soient deux types *tc* et $t'c$ de Tc, si *tc* $\leq_c$ $t'c$ alors $t'c$ est dit sur-type de tc et tc est dit sous-type de $t'c$. Par exemple, le type de concepts Chat peut être vu comme un sous-type du type de concepts *Animal*.

Deux types de concepts particuliers sont distingués :

- Le type de concepts **universel**, noté T , encapsule tous les autres types. Il est donc sur-type de tous les types de concepts de Tc ;
- Le type de concepts **absurde**, noté ⊥ , représente le type de concepts le plus spécifique que tous les autres types de concepts (sous-type) et il ne possède pas d'instance.

La figure 7 présente un exemple de hiérarchie de types de concepts où les arêtes représentent la relation ≤c, les types de concepts les plus généraux étant placés au-dessus des types plus spécifiques.

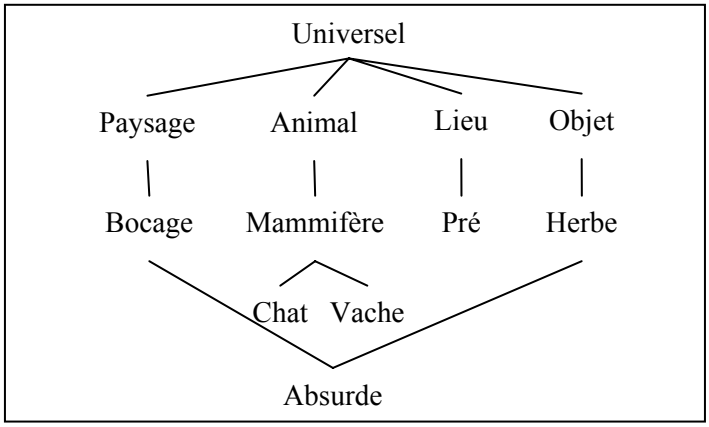

Figure 7. Exemple d'une hiérarchie de types de concepts [Für, 04b]

2) Les types de relations Tr

Un type de relations représente les relations pouvant exister entre les différents concepts, c'est la raison pour laquelle à chaque type de relations est associée une signature qui spécifie les types de concepts définis dans Tc. Formellement, on définit une application σ : $Tr \rightarrow T_c^n$ qui, à chaque type de relations, associe sa signature, c'est-à-dire le nuplet de types de concepts les plus généraux pouvant être liés par ce type de relations. L'arité de la signature est également l'arité du type de relations. Par exemple, le type de relations broute est d'arité 2 et sa signature est (Animal;

Herbe). Les types de relations sont également hiérarchisés par une relation de spécialisation notée $\leq r$. Deux types de relations d'arités différentes ne sont cependant pas comparables par $\leq r$.

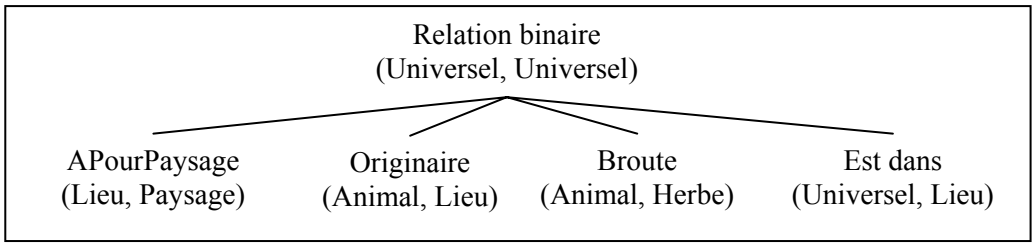

Figure 8. Exemple d'une hiérarchie de types de relations [Für, 04b]

La figure 8 présente un exemple de hiérarchie de relations, avec leurs signatures, basé sur la hiérarchie de types de concepts de la figure 7.

3) Les marqueurs individuels M

Les marqueurs permettent d'identifier les concepts (les instances) p. ex. la vache marguerite sera identifiée par le marqueur Marguerite associé au type de concepts *Vache*. Sur l'ensemble M est définie une application $\tau : M \rightarrow Tc$ qui à chaque marqueur associe le type le plus spécifique de l'instance qu'il désigne. Le type $\perp$ n'a bien évidemment aucun antécédent par τ .

Le **support** S = (Tc; Tr; M), les 3 ensembles Tc, Tr et M étant disjoints et les ensembles Tc et Tr étant partiellement ordonnés.

7.3.2. Le niveau assertionnel

Au niveau assertionnel, les faits sont représentés en utilisant le vocabulaire décrit dans le support. Ces connaissances sont présentées sous forme de graphes d'un type particulier appelés **graphes conceptuels**. Un graphe conceptuel est un multi-graphe fini, non-orienté et biparti, composé de:

- Un ensemble de *sommets concepts* (représentés par des rectangles) où chaque sommet concept possède un type de concepts et un marqueur individuel ;
- Un ensemble de *sommets relations* qui expriment la nature des liens entre les concepts (représentés par des ellipses) ;
- Un ensemble d'*arêtes* qui lient les sommets relations aux sommets concepts ;
- Une *étiquette* associée à chaque sommet et à chaque arête (un couple *(type de concept, marqueur)* pour un sommet concept, un type de relation pour un sommet relation, et le numéro d'ordre dans le voisinage d'un sommet relation donné pour une arête).

Un graphe conceptuel est forcément défini sur un support donné.

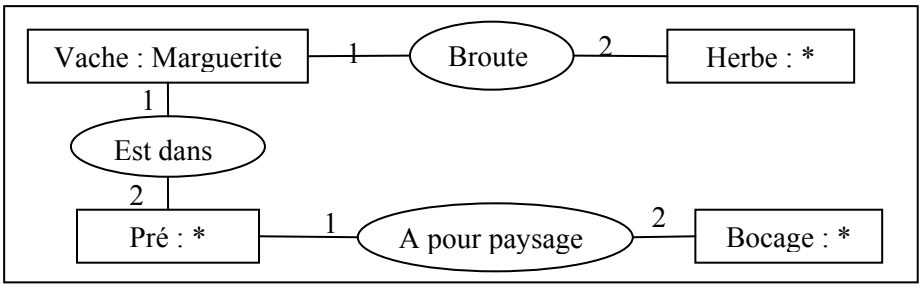

Figure 9. Exemple d'un graphe conceptuel [Für, 04b]

La figure 9 illustre un exemple d'un graphe conceptuel. Ce dernier peut être interprété comme : *« le mammifère de type vache Marguerite broute de l'herbe dans un pré d'une région de bocage»*.

7.3.3. Les opérations

La notion fondamentale pour tout raisonnement concernant les graphes conceptuels est la relation de **spécialisation** et **généralisation** sur l'ensemble des graphes conceptuels [Mug, 96]. La spécification est une opération de projection de graphe[1] utilisée pour savoir si une connaissance représentée par un graphe (ou une partie d'elle) est une variante plus spécifique qu'une autre représentée par un autre graphe (qui joue le rôle de modèle ou de validateur de la première) [Kay, 97], la généralisation est l'opération inverse.

On dit qu'un nœud c' : m' spécialise un autre c : m si et seulement si :

1. Le concept c est relié a c' dans le support par un chemin d'arcs Is-a (dans ce sens)
2. Le marqueur m et le marqueur existentiel * ou que m=m'

Plus formellement, on dit qu'un graphe conceptuel *H* spécialise *G* si et seulement si il existe deux applications f et g, f allant de l'ensemble des nœuds de *G* vers ceux de *H* ; et g de l'ensemble des arcs de G vers H telles que [Sow, 00a] :

- *f* associe à chaque nœud concept de *G* (étiqueté par le concept c et le marquer m), un nœud *f*(c, m) de *H* noté (c', m') tel que (c', m') spécialise (c, m).
- *f* associe à chaque nœud relation de *G* un nœud de relation de *H* de même type (marqué par la même étiquette).
- *g* associe à chaque arc (a) de *G* reliant un nœud de relation (R) et un nœud concept (c, m) un arc *g*(a)de *H* qui relie *f*(R) à *f*(c, m).

La généralisation étant la relation duale de la spécification, il suffit juste de trouver les applications inverses de f et g.

[1] Pour raison d'abréviation le terme **graphe** est utilisé pour indiquer un graphe conceptuel.

1) Opérations élémentaires de spécialisation

Les opérations élémentaires de spécialisation sont des opérations internes sur l'ensemble des graphes conceptuels définis sur un support donné [Mug, 96]. Les opérations élémentaires de spécialisation d'un graphe conceptuel G dans un autre graphe conceptuel H sont:

a. *Simplification de relation* : si *G* possède deux sommets relations de même type et jumeaux (c.-à-d. ayant exactement les mêmes voisins dans le même ordre), *H* s'obtient en supprimant l'un des jumeaux.
b. *Restriction de relation* : soit r un sommet relation de *G*. *H* s'obtient en diminuant l'étiquette de r : remplacer le type de r par un type plus petit $t_{r'}$ tel que : $t_{r'} \leq t_r$ en conservant les arêtes.
c. *Restriction de concept* : soit c un sommet concept de type c et d'identifiant *m* de *G*. *H* s'obtient en replaçant t_c par un sommet concept de type $t_{c'}$ et d'identifiant *m'* tel que $t'_c \leq t_c$ et soit m=* ou m=m'.
d. *Joint interne* : soient c_1 et c_2 deux sommets concepts de *G* ayant même étiquette. *H* s'obtient en fusionnant c1 et c2 (Cf. figure.10), où les sommets fusionnés appartiennent à la même composante connexe [Mug, 96].

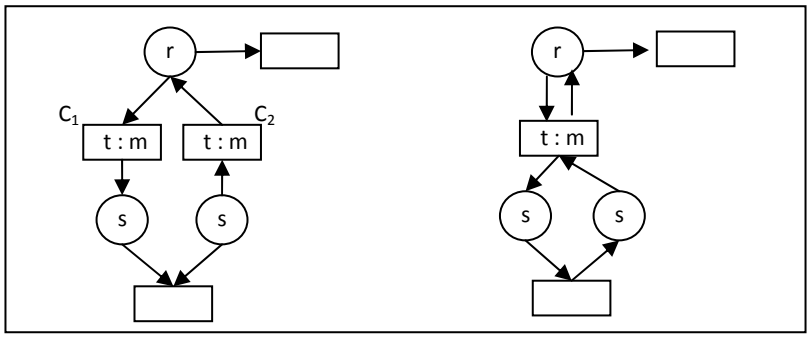

Figure 10. Joint interne [Mug, 96]

2) Opérations élémentaires de généralisation

La définition des règles élémentaires de généralisation est l'inverse des règles de spécialisation. Les opérations permettant d'obtenir la généralisation d'un graphe conceptuel *H* à partir d'un graphe conceptuel *G* sont [Mug, 96] :

a. *Duplication de relation* : soit *r* un sommet relation de *G.* On obtient *H* en ajoutant un sommet relation de même type que *r* et jumeau de *r*.
b. *Augmentation de relation* : soit *r* un sommet relation de *G. H* s'obtient en remplaçant r par r' tel que $t_r \leq t'_r$ (Il n'y a pas de contrainte sur cette opération).
c. *Augmentation de concept :* soit *c* un sommet concept de *G.* On obtient *H* en remplaçant c par c' tel que $t_c \leq t'_c$ et, soit m'=* ou m=m'.

d. *Eclatement*: Soit c un sommet concept de *G*. On obtient *H* en éclatant c en deux sommets c_1 et c_2, de même étiquette que c, tel que les ensembles d'arêtes adjacentes à c_1 et c_2 forment une bipartition (au sens large, l'un des deux ensembles peut être vide) de l'ensemble des arêtes adjacentes à c.
e. *Décomposition*: Supprimer certaines composantes connexes (pas toutes) de *G*.

Etant donnés deux graphes conceptuels *G* et *H*, on dit que *H* est une généralisation de *G* si *H* s'obtient à partir de G par une séquence finie d'opérations de généralisation. Une dérivation de *G* à *H* est un chemin, puisque les opérations de généralisation sont des opérations unaires. On a immédiatement la propriété : *G* est une spécialisation de *H* ($G \leq H$) ssi *H* est une généralisation de *G*. La relation ≤ sur les graphes conceptuels pourra donc être comprise indifféremment en termes de généralisation ou de spécialisation.

3) Projection

La projection est l'opération qui permet de déterminer si un graphe est plus spécialisé, ou plus général, qu'un autre [Für, 04b]. La projection d'un graphe *G* sur un graphe *H*, tous deux étant bien formés relativement à un support *S = (Tc, Tr, M)*, est la donnée de deux applications f_r, de l'ensemble des sommets relations de *G* dans l'ensemble des sommets relations de *H* et f_c, de l'ensemble des sommets concepts de *G* dans l'ensemble des sommets concepts de *H*, tel que :

- Pour tout sommet relation de *G*, son type est plus général que celui de son image par f_r ;
- Pour tout sommet concept de *G*, son type est plus général que celui de son image par f_c et, soit son identifiant est *, soit son identifiant est celui de son image par f_c ;
- Pour tout sommet relation r de *G* d'arité n, pour tout i de 1 à n, l'image par f_c du sommet concept numéro i lié à r est le sommet concept numéro i lié au sommet relation image par f_r de r.

Il est démontré dans [Sow, 84, Mug, 92] qu'il existe une projection d'un graphe conceptuel *G* dans un graphe conceptuel *H* si et seulement si *H* est une spécialisation de *G* ($H \leq G$).

7.3.4. Sémantique logique des graphes conceptuels

A tout graphe conceptuel G, on associe une formule de la logique du premier ordre, notée Φ(G) [Sow, 84]. A tout *type* de concept ou de relation, on associe un *prédicat* de même nom : un prédicat unaire pour les types de concepts, un prédicat ayant la même arité que le type pour les types de relations. A tout *marqueur individuel,* on associe une *constante* de même nom [Mug, 96].

L'interprétation logique d'un support *S* est un ensemble de formules Φ*(S)* traduisant les liens *sorte de* entre types de concepts et entre types de relations. Ainsi :

- Pour tous t_1 et t_2 de Tc (ensemble des types de concepts), tels que t_1 couvre t_2 dans *Tc* (c'est-à-dire $t_2 < t_1$ *et* il n'existe pas de type t_3, tel que $t_2 < t_3 < t_1$), on considère la formule : $\forall x\ t_2(x) \rightarrow t_1(x)$, qui correspond à l'interprétation des liens *Sorte de* entre types de concepts.
- Pour tous t_{r1} et t_{r2} de *TRp* (*p* étant l'arité) tels que t_{r1} couvre t_{r2} dans *Tr* (ensemble des types de relations), on considère la formule : $\forall x_1 \ldots \forall x_p\ t_{r2}(x_1 \ldots x_p) \rightarrow t_{r1}(x_1 \ldots x_p)$, qui correspond à l'interprétation des liens *Sorte de* entre types de relations.

Tout graphe conceptuel bien formé G peut être transformé en une formule bien formée Φ(*G*) de la logique des prédicats par l'application des règles suivantes :

1. à chaque concept générique est associé une variable distincte.
2. à chaque marqueur individuel m est associé la constante m.
3. à chaque concept est associé un prédicat unaire de nom celui du type du concept et d'argument la variable ou la constante associée à son marqueur.
4. à chaque relation d'arité n est associé un prédicat n-aire de nom celui du type de la relation et de ième argument la variable ou la constante associée au marqueur du ième sommet concept voisin de la relation.
5. on prend la conjonction de tous les prédicats ainsi construits.
6. on ferme existentiellement les variables de la formule.

Par exemple, le graphe de la figure 9 est logiquement équivalent à la formule :

$$\exists x \exists y \exists z \big(Vache(Marguerite) \wedge Herbe(x) \wedge Pre(y) \wedge Bocage(z) \wedge broute(Marguerite, x) \wedge estDans(Marguerite, y) \wedge aPourPaysage(y, z) \big).$$

7.3.5. Les graphes emboîtés

Afin de pouvoir représenter des connaissances plus ou moins précises, en raisonnant à un niveau de précision donné, le modèle des graphes conceptuels simples a été étendu par la notion des graphes emboîtés. Il s'agit de pouvoir faire des « zooms » sur certains sommets [Mug, 96]. Les graphes conceptuels permettent par le rajout de notations spéciales pour les ensembles et les nombres, de méthodes de référence vers des entités du monde (physiques ou logiques) ainsi que des mécanismes d'imbrication de graphes de faire croître leur expressivité. Ils permettent ainsi [Kay, 97, Sow, 00a] :

- de définir une sémantique plus poussée des concepts du domaine et de créer des concepts personnalisés.
- de représenter des connaissances modales p. ex. croyance.

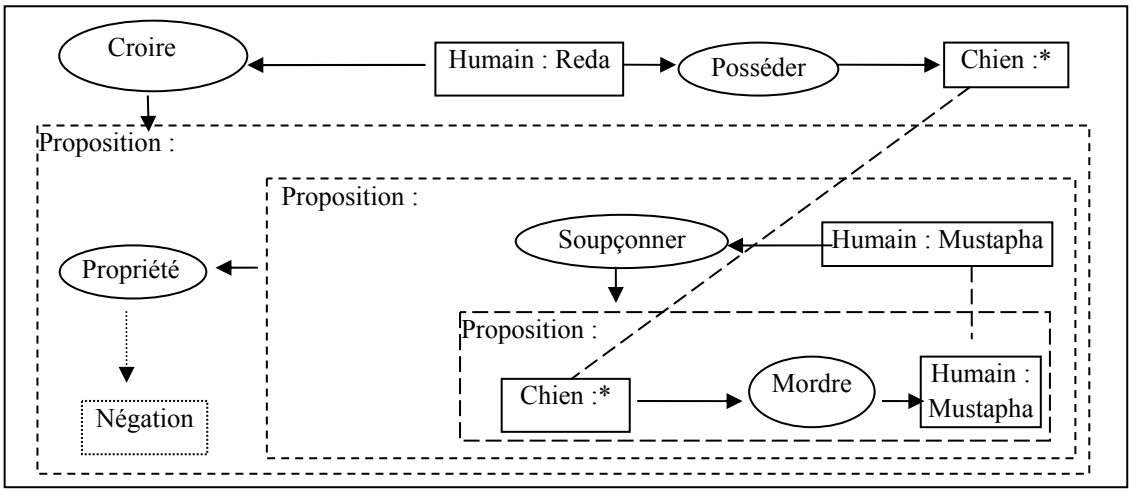

Figure 11. Exemple d'un graphe emboité

La figure 11 représente la situation : *Reda ne croit pas que Mustapha soupçonne que son chien (le chien de Reda) l'ait mordu*.

7.3.6. La SG-family

L'opération de projection permet de comparer des énoncés, de répondre à des questions portant sur les connaissances dont on dispose : on peut calculer si, dans une base de connaissances formalisée avec les graphes conceptuels, une information est présente, cette dernière étant elle aussi formalisée sous forme de graphes conceptuels. À partir de cette opération, un formalisme peut alors être défini pour représenter des règles et des contraintes [Bag, 02, Füsrt 04b].

Ces règles et contraintes de graphes conceptuels constituent la SG-family, une extension du modèle de base des GCs complètement formalisée par *Baget* et *Mugnier* [Bag, 99]. L'extension du modèle de base défini précédemment aux règles par l'utilisation de contraintes avait été préalablement étudiée par *Salvat* et *Dibie* [Sal, 97**,** Dib, 00].

Lambda-abstraction

Une lambda-abstraction n-aire est un graphe conceptuel *G* contenant n sommets concepts génériques repérés par n variables $x_1,\ldots,x_n$; on note cette lambda-abstraction par : $\lambda x_1,\ldots,x_n G$.
Une règle ou une contrainte est constituée d'un couple *(H, C)* de lambda-abstractions avec $H = \lambda x_1,\ldots, xnG_1$ et $C = \lambda x_1,\ldots, xnG_2$, liés par leurs sommets concepts $x_1,\ldots,x_n$ appelés *points de*

connexion à l'aide des liens de connexion appelés **liens de coréférence**. *H* constitue le graphe hypothèse et *C* le graphe conclusion [Für, 04b].

La présentation formelle des lambda-abstractions a été proposée originellement par *Salvat* [Sal, 98] avec l'introduction des règles avec des liens de connexion, et *Baget* [Bag, 99, Bag, 02] qui propose un formalisme qui permet de remplacer les liens de connexion par la notion de lambda-abstraction. La figure 12 montre un exemple de lambda-abstractions, où les pointillés indiquent des liens de coréférence entre les parties hypothèse et conclusion.

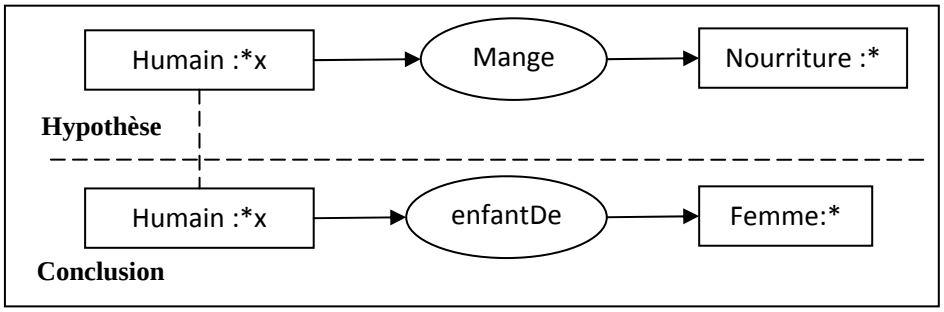

Figure 12. Exemple d'un couple de lambda-abstractions

Si les règles et contraintes ont la même forme, leurs sémantiques, bien entendu, diffèrent : les règles expriment la possibilité d'inférer des connaissances alors que les contraintes interdisent ou forcent l'expression de certaines connaissances [Für, 04b].

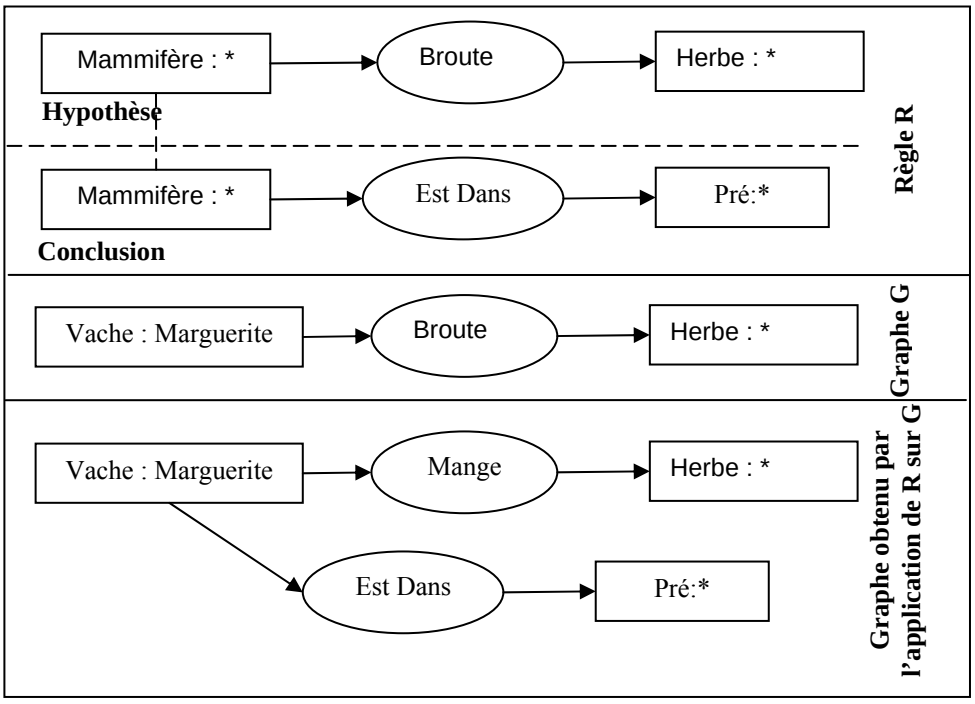

Figure 13. Exemple de règle [Für, 04b]

Les règles : avec un graphe hypothèse et un graphe conclusion dont la sémantique est : si l'hypothèse est présente dans un graphe, la conclusion peut être ajoutée au graphe.

Le mécanisme de projection est utilisé pour déterminer si la règle est applicable. La figure 13 montre la règle : *Si un mammifère broute de l'herbe alors il est dans un pré*.

Les contraintes : deux types de contraintes sont définis au sein de la SG-family :

1) les **contraintes positives** : avec un graphe hypothèse et un graphe conclusion dont la sémantique est : si l'hypothèse est présente dans un graphe, la conclusion doit être présente dans le graphe (sinon la contrainte n'est pas respectée et le graphe n'est pas valide dans le domaine modélisé).
2) les **contraintes négatives**, avec un graphe hypothèse et un graphe conclusion dont la sémantique est : si l'hypothèse est présente dans un graphe, la conclusion doit être absente du graphe (sinon la contrainte n'est pas respectée et le graphe n'est pas valide dans le domaine). La figure 14 présente la contrainte: *on ne peut avoir le beurre et l'argent du beurre*. Une contrainte négative est identifiée par un « - » dans un carré.

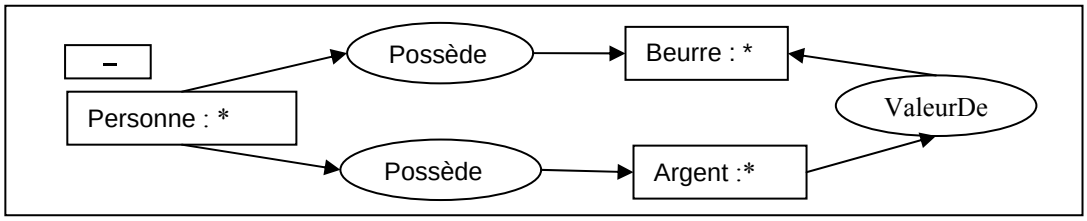

Figure 14. Contrainte négative [Für, 04b]

L'utilisation de ces trois types de connaissances par des moteurs d'inférence peut atteindre un certain degré d'automatisation.

7.3.7. Implémentation des graphes conceptuels sur machine

Plusieurs recherches visent à implémenter le modèle des graphes conceptuels en machine et plusieurs outils et bibliothèques ont été développés pour permettre cette implémentation, à savoir, Cogitant [Gen, 10] et Prolog+CG [Kab, 00] etc.

1) **Cogitant**[2] se place dans l'optique de l'utilisation du modèle de GCs comme un modèle formel disposant d'opérations de raisonnement internes au modèle (basées sur le morphisme de graphes), mais conservant les propriétés de consistance et complétude vis à vis de la logique des prédicats du premier ordre. Un deuxième point de vue consiste à considérer le langage comme une simple notation graphique de la logique des prédicats du premier ordre [Sow, 84a, Wer, 95, Für, 04b].

[2] http://cogitant.sourceforge.net/

Cogitant est une bibliothèque développée depuis 1994 au sein du LIRMM de Montpellier sous forme d'un ensemble de classes *C++* permettant de manipuler facilement des graphes conceptuels. Cogitant permet de représenter des connaissances et raisonner avec des graphes conceptuels. À chaque objet du modèle correspond une classe du programme, et les opérations sur les graphes sont implémentées, en particulier la projection et l'application d'une règle GC sur un graphe. Cogitant contient des classes dédiées aux opérations d'entrée/sortie pour stocker les supports, les graphes et les règles sous les formats BCGCT et CoGXML[3]. Le format BCGCT (Base de Connaissances Graphes Conceptuels Textuelle) est un format de stockage propre à Cogito [Hae, 95, Für, 04b]. Le format CoGXML est une transcription dans la syntaxe XML du format BCGCT.

2) **Prolog**[4] est un outil d'IA basé sur la programmation logique, actuellement ; il est devenu un standard, pour le langage programmation en IA. **Prolog-CG** est une extension orienté objet du langage Prolog intégrant les **graphes conceptuels** comme un modèle de représentation de connaissances [Kab, 00]. Prolog-CG permet, au travers d'un programme écrit en Java, de décrire une base de connaissances à travers le modèle des graphes conceptuels en utilisant une syntaxe Prolog. Une fois le support défini, ainsi que les liens de subsomption, on peut décrire des graphes, puis exécuter des requêtes sur la base via des prédicats Prolog prédéfinis.

Dans notre travail nous utilisons la plate-forme Cogitant qui intègre les mécanismes de raisonnement basés sur la projection, et non le système Prolog+CG qui n'exploite le modèle des graphes conceptuels que pour structurer les représentations de connaissance, et recourt au formalisme logique de Prolog pour raisonner.

8. Langages de représentation des ontologies

Une des principales décisions à prendre dans le procédé de développement d'ontologies consiste à choisir le langage dans lequel l'ontologie sera exprimée et utilisée.

Au début des années 1990, les ontologies étaient principalement construites en utilisant des techniques de représentation de l'intelligence artificielle. Ils sont basés principalement, selon *Gómez Pérez* [Góm, 04, Corcho 03], sur les formalismes de représentation de connaissances suivants :

- La logique du premier ordre tel que ***KIF*** ;
- Les frames combinés avec la logique du premier ordre tel qu'Ontolingua, OCML et ***Flogic*** ;

[3] Le format CoGXML était appelé CGXML dans les précédentes versions de Cogitant.
[4] http://www.insea.ac.ma/CGTools/PROLOG+CG.htm

- La logique de description telle que ***Loom***.

Ces langages sont considérés comme étant traditionnels (appelés aussi classiques) par rapport à ceux présentés dans la section suivante.

Le boom de l'Internet a mené à la création des langages d'implémentation des ontologies exploitant les caractéristiques du Web. Ils sont connus sous le nom des langages de balisage (*markup languages*) ou des langages d'ontologie dans le contexte du Web sémantique (web-based ontology language). Certains d'entre eux sont basés sur la syntaxe de XML tels que : ***XOL*** [Kar, 92] (Ontology Exchange Language), ***SHOE*** [Luk, 00] (Simple HTML Ontology Extension) qui a été précédemment basé sur le HTML, ***RDF*** [Las, 99] (Resource Description Framework) et ***RDF Schema*** [Bri, 02] qui est une extension de RDF. Les deux derniers sont des langages développés par des groupes de travail du World Wide Web Consortium (W3C). La combinaison de RDF et RDF *Schema* est connue sous le nom de ***RDF(S)***. Par la suite, trois autres langages ont été développés comme une extension de RDF(S) qui sont basés sur la logique de description : ***OIL*** [Fen, 00] (Ontology Inference Layer), ***DAML+OIL*** [Fen, 01] et ***OWL*** qui est le successeur de DAML+OIL. OWL est fractionné en trois langages distincts : ***OWL LITE***, ***OWL DL*** et ***OWL FULL***.

La figure 15 présente les langages de spécification d'ontologie qui sont récemment développés et les rapports principaux entre eux sous forme d'une pyramide des langages du Web sémantique.

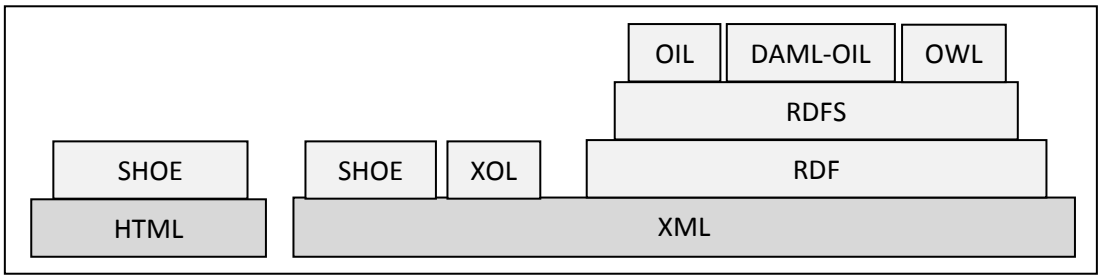

Figure 15. Langages d'ontologies basés Web [Corcho 03]

1) RDF[5] [Las, 01] (Ressource Description Framework) développé et recommandé par le W3C, permet de décrire les ressources du web sémantique qui sont l'élément de base de RDF. Chaque ressource est pourvue d'un identifiant URI (Uniform Resource Identifier).
   Tout document RDF est composé d'un ensemble de triplets (sujet, prédicat, objet) ou encore (ressource, propriété, valeur). Un ensemble de tels triplets est appelé un graphe RDF. Ceci peut être illustré par un diagramme composé de nœuds et d'arcs orientés, dans lequel chaque triplet est représenté par un lien nœud-arc-nœud (d'où le terme de "graphe").

[5] http://www.w3.org/RDF/

A ce modèle est associée une syntaxe écrite en XML et basée sur les triplets :

- Ressource (Sujet) : une entité d'informations pouvant être référencée par un identificateur. Cet identificateur doit être une URI.
- Propriété (prédicat) : l'attribut ou la relation utilisée (e) pour décrire une ressource.
- Valeur (objet) : la valeur d'une propriété associée à une ressource spécifique.

2) RDF Schema[6] a été construit par W3C comme extension de RDF comportant des primitives basées sur des frames. RDF Schema permet notamment de déclarer les propriétés des ressources ainsi que le type des ressources. La combinaison de RDF et RDF Schéma est connue sous le nom RDF(S). Bien que relativement limités dans la mesure où ils ne sont pas très expressifs, les langages RDF(S) peuvent cependant spécifier des concepts, des taxonomies et des relations binaires.

3) DAML-OIL a été proposé par le W3C pour représenter des méta-données et des ontologies. DAML a été transformé en DAML+OIL en intégrant certaines propriétés d'OIL. Il repose sur RDF et RDF Schema et fournit en plus des primitives plus riches issues de la logique de description.

4) OWL[7] (Web Ontology Language) est le standard actuellement recommandé par W3C pour représenter les ontologies. C'est une extension du vocabulaire de RDF(S). Il est dérivé du langage d'ontologie DAML+OIL.

   OWL a été fractionné en fait en trois sous-langages distincts offrant une complexité croissante et destinés à des communautés différentes d'utilisateurs. Chacun est une extension par rapport à son prédécesseur plus simple. Ils sont caractérisés par l'expressivité croissante (OWL Lite, OWL DL et OWL FULL) et la calculabilité des inférences décroissantes d'OWL Lite à OWL FULL.

9. Outils de développement des ontologies

Les outils qui aident à la construction et le développement d'ontologies qui existent sur le marché aujourd'hui sont divers et variés à bien des égards. On peut distinguer deux familles d'outils d'édition, selon leur évolution depuis le milieu des années 1990:

- Les outils de construction dépendants du formalisme de représentation, afin de permettre l'édition et la navigation sur des ontologies selon leur langage de développement, ainsi pour

[6] http://www.w3.org/TR/rdf-schema/
[7] http://www.w3.org/OWL/

permettre l'importation et l'exportation d'ontologies à partir de/vers d'autres langages, mais ils exigent que les utilisateurs aient connaissance du langage d'ontologie correspondent. Dans cette catégorie on peut citer les outils suivants : Ontolingua[8], OntoSaurus [Swa, 97], WebOnto et OilEd[9] [Bec, 01].

- Les outils de construction d'ontologies indépendants du formalisme de représentation, dont la principale caractéristique est qu'ils ont une architecture extensible, et dont le modèle de connaissances est souvent indépendant du langage de l'ontologie. Ces outils ont été construits pour supporter une large variété d'activités du processus de développement d'ontologies. Pour ce faire, ils ont une architecture extensible à base de composants, où de nouveaux modules peuvent facilement être ajoutés pour fournir plus de fonctions. Dans cette catégorie on peut citer les outils suivants : Protégé2000[10] [Noy, 00], ODE[11] et son adaptation pour le Web WEBODE[12] [Arp, 03] et OntoEdit[13] [Sur, 02].

10. Moteurs d'inférences

Un moteur d'inférence (raisonneur) permet de raisonner ou d'inférer sur une ontologie. De nombreux moteurs d'inférence utilisant des formalismes variés et offrant différentes fonctionnalités ont été développés. Tous ces outils offrent des supports pour le processus d'ontologisation, mais peuvent offrir une aide à la conceptualisation. Il y a d'autres qui nous intéressent plus qui supportent la phase d'opérationnalisation. Dans ce qui suit nous citons les raisonneurs les plus reconnus, à savoir :

- **Otter et SNARK**[14] sont des moteurs d'inférence basés sur des données RDF et le formalisme de la logique du 1er ordre [Sti, 94].
- **Racer et FaCT** sont des moteurs d'inférence basés sur la logique de description, déjà implantés dans plusieurs applications telles que Protégé2000 [Noy, 00].
- **Corese**[15] est un moteur RDF(S) basé sur les **graphes conceptuels**. Il permet de transcrire les métadonnées RDF des pages Web en graphes conceptuels en se basant sur l'analogie entre les deux modèles de représentation [Cor, 00, Cor, 02, Cor, 06]. Corese utilise la projection pour répondre à des requêtes (sous forme de graphes) concernant les pages Web. Les ontologies

[8] Ontolingua Home Page, http://www.ksl.stanford.edu/software/ontolingua/, 2004
[9] OIL Editor Home Page, http ://oiled.man.ac.uk/, 2004.
[10] Protege2000 Ontology Editor Home Page, http://protege.stanford.edu/, 2002.
[11] Ontology Design Environment Home Page, http://www.swi.psy.uva.nl/wondertools/html/ODE.html, 2004.
[12] WebODE Home Page, http ://delicias.dia.fi.upm.es/webODE/, 2004.
[13] http://www.semtalk.com/semnet_files/POntoEdit.htm
[14] http://www.ai.sri.com/~stickel/snark.html
[15] http://www-sop.inria.fr/edelweiss/software/corese/

servent à compléter les méta-données décrivant les pages Web avant d'appliquer les requêtes [Für, 04b].

- **Pellet**[16] est un Java open-source basé sur le raisonneur OWL-DL ; il travaille sur des ontologies décrites en RDF ou OWL.
- **TooCoM**[17] (a Tool to Operationalize an Ontology with the Conceptual Graphs Model) est un outil dédié à la transcription automatique d'une représentation ontologique des connaissances d'un domaine, dans une représentation opérationnelle adaptée à un scénario d'usage défini par l'utilisateur [Für, 03a, Für, 04b]. Il est basé sur les graphes conceptuels comme formalisme de représentation de connaissances.

À l'heure actuelle, Pellet et Racer sont les deux seuls moteurs d'inférence, permettant le raisonnement et exploitent des ontologies possédant un niveau d'expressivité en logique de description et acceptent en entrée des fichiers OWL.

11. Conclusion

À travers ce chapitre, nous avons remarqué que les ontologies ont suscité beaucoup d'intérêts et qu'elles ont donné lieu à des modèles théoriques significatifs. Cependant, ces modèles restent isolés et indépendants des approches de développement et d'utilisation traditionnelles. Afin de pouvoir intégrer les ontologies dans le processus de développement logiciel, les concepteurs se sont contentés de s'assurer du respect de la politique de création moyennant des techniques plus ou moins efficaces. Les travaux actuels visent à intégrer les modèles de représentation de connaissances dans une approche formelle de développement logiciels sûrs.

De plus, nous avons essayé d'éclaircir comment une ontologie peut être représentée dans les différents modèles de représentation de connaissances, à savoir les frames, les logiques de descriptions et les graphes conceptuels. Ces modèles de représentation de connaissances ont permis l'exploitation et l'opérationnalisation des ontologies.

En-outre, nous avons exposé le modèle des graphes conceptuels, et nous plaçons notre travail dans l'optique de l'utilisation de ce modèle et une partie de ses extensions comme un formalisme de représentation graphique de connaissances et langage opérationnel offrant des mécanismes internes de raisonnement. Ce choix a été fait vu que les graphes conceptuels peuvent être implémentés par des langages, en particulier des langages adaptés au Web et utilisant la syntaxe XML. Les langages implémentant ces modèles offrent souvent des mécanismes de raisonnement, et servent alors à représenter des ontologies opérationnelles.

[16] http://clarkparsia.com/pellet/
[17] http://prologpluscg.sourceforge.net/, http://sourceforge.net/projects/toocom/

Chapitre II : L'opérationnalisation

1. Introduction

Les ontologies sont actuellement au cœur de nombreuses applications de l'ingénierie des connaissances, en particulier le web sémantique, car elles permettent de raisonner sur des domaines de connaissances. Cependant, il apparait de plus en plus nécessaire de pouvoir raisonner sur les ontologies elles-mêmes, pour les évaluer. Dans ce cadre, nous présentons dans ce chapitre les limites des ontologies en termes de raisonnement et d'inférence qui ont mené à la création des ontologies opérationnelles. Dans ce cadre nous commençons par positionner le cadre d'opérationnalisation par rapport aux ontologies et par rapport aux représentations de connaissances présentées dans le chapitre précédent. Ce positionnement nous aide par la suite à définir : le processus d'opérationnalisation d'ontologies dans le cadre général, schéma général d'opérationnalisation d'une ontologie en vue de son utilisation au sein d'un système à base de connaissances et à la fin nous parlons de l'utilisation des ontologies opérationnelles pour raisonner.

2. Les limites des ontologies en termes de raisonnement et d'inférence

Dans le chapitre précédent nous avons présenté les recherches concernant l'ingénierie des ontologies, qui est née de la volonté de permettre une représentation des connaissances indépendantes de ses diverses applications, de manière à assurer sa portabilité d'une application à l'autre. L'expérimentation de ces recherches n'a pas abouti à des consensus, et de nombreux problèmes n'ont pas encore trouvé de solution, comme :

- *La spécification des propriétés de concepts et de relations* : si l'intégration de hiérarchies de concepts et relations, est admise par tous, la spécification des propriétés de ces concepts, qui expriment la sémantique du domaine, est moins consensuelle. La nécessité d'intégrer des propriétés conceptuelles aux ontologies est reconnue, mais les outils d'édition et les langages de représentation se bornent souvent à permettre la spécification de propriétés algébriques ou ensemblistes bien connues. Et les langages et outils permettant de représenter n'importe quel type d'axiome n'offre que des représentations opérationnelles dédiées à un type de raisonnement particulier, représentations inadaptées aux ontologies qui doivent être bâties au niveau conceptuel [Für, 04b].
- *L'utilisation d'ontologies pour la communication* : beaucoup d'ontologies ont pour but de servir de base à la communication entre machines, entre machine et utilisateur ou entre utilisateurs (systèmes multi-agents, systèmes médiateurs) ou de servir de support à la recherche d'information, en particulier sur le Web ou dans des systèmes de traitement de la langue. L'utilisation d'ontologies pour la communication repose essentiellement sur une terminologie commune et ne fait que peu appel aux axiomes. De même, la recherche d'information basée sur des ontologies n'utilise véritablement pour le moment que la spécification des liens de subsomption entre concepts [Für, 04b].
- *L'utilisation d'ontologies pour le raisonnement* : la représentation des connaissances par les ontologies peut s'accompagner des mécanismes de raisonnement qui concernent la manipulation des connaissances déjà acquises pour produire de nouvelles connaissances. Et pour que ces ontologies soient manipulables en machine, elles se doivent être formelles, avec une sémantique opérationnelle libre. La sémantique opérationnelle dépend du domaine de connaissance et est complètement déterminée à partir d'un corpus à l'issue des processus de conceptualisation et d'ontologisation. Cette sémantique dépend des connaissances du domaine et l'usage opérationnel ; elle est définie par les experts du domaine et les concepteurs du système intégrant l'ontologie. Son intégration dans un système à base de connaissances nécessite donc de préciser sa sémantique opérationnelle.

Les outils d'ingénierie ontologique, permettent d'utiliser les ontologies pour mener des raisonnements basés sur des propriétés conceptuelles autres que la subsomption [Für, 04b].

Donc, la construction des représentations de connaissance au niveau conceptuel, permet de donner un **raisonnement** opérationnel aux ontologies. De ce fait, le problème de l'opérationnalisation des ontologies, sera traduit en la mise en œuvre de ces ontologies d'une

façon opérationnelle dans un système à base de connaissances pour mener différents types de raisonnement.

3. L'opérationnalisation et la représentation de connaissances

Il existe généralement plusieurs façons d'appréhender les éléments de connaissances relatifs à un domaine, *Kayser* et *Dominé* [Kay, 97, Dominé 88] affirment qu'il n'y a pas de moyens concrets pour affirmer qu'un domaine est mieux représenté à l'aide d'un formalisme qu'un autre, car plusieurs caractéristiques souvent duales sont mieux prises en compte par certaines représentations que par d'autres comme:

- L'efficacité des méthodes de traitement et leur aptitude à produire des inférences plus poussées et robustes et offrant des mécanismes de preuve et d'explication de raisonnements.
- L'expressivité du langage de représentation permet la modélisation de plus de connaissances mais ceci au détriment du premier critère.
- La facilité de l'interprétation et de la formulation des connaissances par les utilisateurs qui grâce à une meilleure visualisation des concepts et relations pourront être mieux utilisés.

3.1. Le choix du modèle de représentation adéquat

Le choix du modèle de représentation est déterminant pour l'attente des objectifs d'inférence, pour cela les modèles doivent :

- Être plus intuitifs et accessibles pour faciliter leurs utilisations dans la construction des ontologies.
- Disposer des normes et des bibliothèques open-source, pour faciliter leurs utilisations dans la création des outils (coté programmation).
- Fournir une opérationnalisation, c.-à-d. l'existence d'un modèle opérationnel de représentation de connaissances compatible avec le modèle conceptuel.

La construction d'une ontologie requiert l'intervention d'experts du domaine qui doivent structurer leurs connaissances pour permettre leur représentation dans un modèle formel. Les experts du domaine n'étant pas a priori experts du modèle de représentation, il convient que celui-ci soit le plus intuitif, attrayant et accessible possible pour faciliter son utilisation. En-effet, un modèle de représentation d'ontologie doit pouvoir se prêter à une opérationnalisation, c'est-à-dire qu'il doit exister un modèle opérationnel de représentation de connaissances compatible avec le modèle conceptuel, en particulier du point de vue du paradigme conceptuel.

Or le modèle des *Frames* n'offre pas de représentation graphique des axiomes, et les outils implémentant ce modèle recourent d'ailleurs à la logique pour exprimer les axiomes, sous forme textuelle et en outre déjà opérationnelle. Les *Logiques de Description* ne permettent pas d'exprimer tous types d'axiome, et ceux exprimés le sont sous forme textuelle [Für, 04b]. Parmi les trois grands types de modèles de représentation présentés au chapitre précédent, le modèles des *Graphes Conceptuels* est le seul à présenter une syntaxe graphique, qui évite d'avoir à faire l'apprentissage d'un vocabulaire symbolique textuel et permet une présentation synthétique des connaissances, plus facilement manipulable qu'un ensemble de formules textuelles. De plus, l'aspect graphique des graphes conceptuels ne se limite pas à la représentation des connaissances terminologiques, mais s'étend à celle des connaissances de raisonnement à travers la SG-family.

Les graphes conceptuels constituent bien sûr un modèle opérationnel de représentation, car il est doté de mécanismes de raisonnement basés sur la projection, et de primitives de raisonnement (règles et contraintes) dont la sémantique opérationnelle est fixée. La syntaxe commune aux règles et contraintes peut cependant être reprise au niveau conceptuel pour représenter des axiomes [Für, 04b].

3.2. L'opérationnalisation et les graphes conceptuels

L'opération fondamentale du modèle des graphes conceptuels, la projection, est facilement compréhensible par un expert d'un domaine non spécialiste du langage, et permet donc de suivre, pas à pas, les raisonnements réalisés afin de valider la représentation adoptée [Mug, 00, Für, 04b]. L'utilisation du modèle des graphes conceptuels comme formalisme de représentation graphique de connaissances et un langage opérationnel offrent des mécanismes internes de raisonnement basés sur l'opérateur de projection. C'est pourquoi nous utilisons la plate-forme *Cogitant* intégrant les mécanismes de raisonnement basés sur la projection, et non le système Prolog+CG qui n'exploite le modèle des graphes conceptuels que pour structurer les représentations de connaissance, et recourt au formalisme logique de Prolog pour raisonner. L'utilisation des graphes conceptuels au niveau opérationnel permet de représenter des bases de connaissances et raisonner dessus [Für, 04b].

4. L'opérationnalisation des ontologies

Nous avons vu que les ontologies sont conçues pour rendre possible le traitement automatique de requêtes. Pour que les ontologies soient manipulables en machine, elles se doivent être formelles, avec une sémantique opérationnelle libre. Cependant, l'utilisation d'une représentation

conceptuelle dans un système informatique n'est possible que si le système peut exploiter les concepts à travers des opérations qui leur sont associées [Bac, 00, Für, 04b].

En outre, la sémantique formelle dépend du domaine de connaissance, elle est complètement déterminée à partir d'un corpus à l'issue des processus de conceptualisation et d'ontologisation [Mus, 92, Für, 04b]. Cette sémantique dépend des connaissances du domaine et de l'usage opérationnel, elle est donc définie par les experts du domaine et les concepteurs du système intégrant l'ontologie, son intégration dans un système de base de connaissances nécessite donc de préciser sa sémantique opérationnelle.

On peut définir l'opérationnalisation comme une opération qui permet de transcrire l'ontologie (du langage formel dans lequel elle est exprimée) dans un langage opérationnel (pour que le système manipule les connaissances).

> Une ontologie est dite *opérationnelle (computationnelle)*, si elle est exprimée dans un langage formel opérationnel offrant des mécanismes de raisonnements [Bou, 95] et dotée d'une sémantique opérationnelle [Für, 04b]

L'opérationnalisation d'une ontologie consiste donc en :

1. La spécification de la sémantique formelle, en ajoutant aux ontologies une sémantique qui permet de décrire les mécanismes de raisonnement [Für, 04b].
2. Transcrire la représentation conceptuelle de l'ontologie dans un langage opérationnel [Für, 04b].

4.1. Opérationnaliser une ontologie au sein d'un SBC

La possibilité d'utiliser l'ontologie dans un système à base de connaissances (SBC) est assurée à travers un processus d'opérationnalisation, qui consiste à introduire l'ontologie dans un langage opérationnel de représentation de connaissances suivant l'objectif opérationnel du système à base de connaissances.

La sémantique opérationnelle ne doit pas modifier la sémantique formelle, mais simplement la compléter, en précisant, dans le cadre interprétatif fixé par la sémantique formelle, les opérations de raisonnement que vont permettre les connaissances dans le système envisagé [Für, 04b].

La sémantique opérationnelle permet aux différentes applications l'usage opérationnel d'une même ontologie (Cf. figure**.** 16) [Cha, 03].

> Opérationnaliser une ontologie au sein d'un SBC, c'est donc la plonger dans un langage opérationnel de représentation de connaissances, conformément à l'objectif opérationnel du SBC [Für, 04b].

Opérationnaliser une ontologie suppose donc de choisir, parmi les opérations autorisées par la sémantique formelle de l'ontologie, celles qui vont être mises en œuvre dans un système opérationnel [Für, 04b].

Le principe consistant à opérationnaliser une représentation de connaissances pour l'adapter à un **objectif opérationnel** a déjà été mis en œuvre dans le domaine de la programmation logique sous le nom de compilation de connaissances (ou compilation de règles) [Cad, 97, Dar, 02].

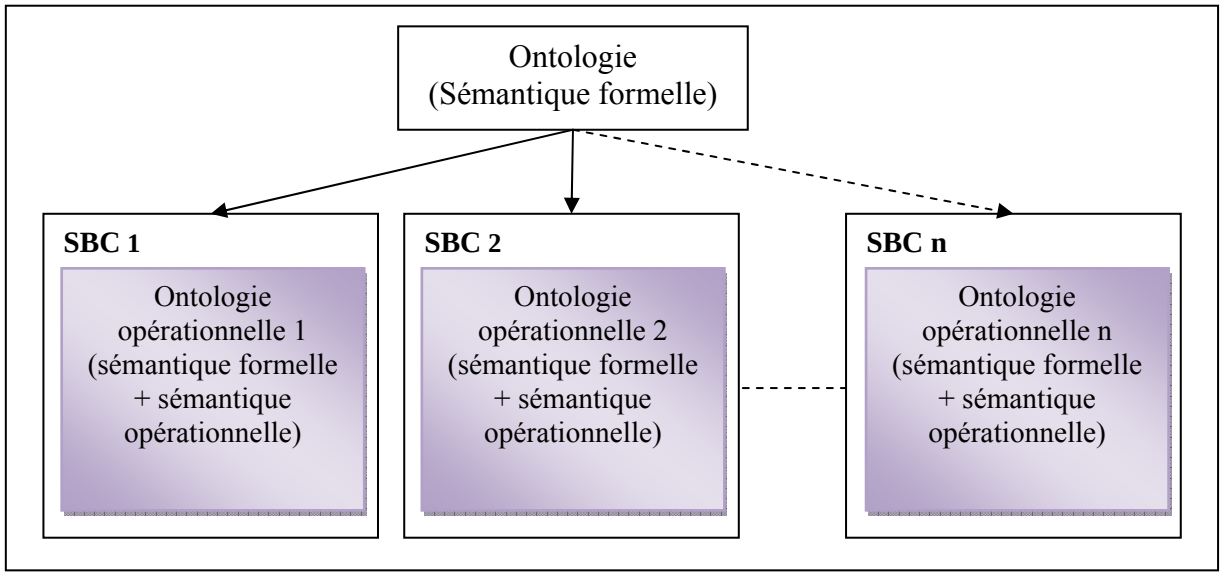

Figure 16. Schéma général d'opérationnalisation d'une ontologie en vue de son utilisation au sein d'un SBC

L'opérationnalisation consiste à transcrire une ontologie dans un langage opérationnel selon un objectif opérationnel donné. Le problème est alors de définir les mécanismes d'opérationnalisation permettant cette transcription [Für, 04]. *Fürst* propose une méthode d'opérationnalisation d'ontologie testée lors d'une expérience de construction et d'opérationnalisation d'une ontologie de la géométrie dans le cadre du modèle des graphes conceptuels [Für, 03, Für, 04b].

4.2. Opérationnaliser une ontologie dans le modèle des GCs

La construction d'une ontologie requiert une étude des connaissances humaines (*conceptualisation),* une définition dans un langage de représentation de connaissances

(*ontologisation)* et des systèmes pour la manipuler *(opérationnalisation)* [Cha, 01]. Donc, l'opérationnalisation nécessite un modèle de représentation de connaissances. Dans le modèle des graphes conceptuels, un certains nombre de schémas d'axiomes sont déjà représentés par des formes opérationnelles, c'est-à-dire des formes dotées d'une sémantique opérationnelle qui définit la façon dont ces propriétés sont utilisées pour raisonner. Plus précisément, la sémantique opérationnelle de certaines propriétés est déterminée par le mécanisme de projection de graphes, qui est à la base des mécanismes de raisonnement sur les graphes conceptuels. C'est le cas de la relation Sorte-de, des signatures des relations et des types des instances de concept.

Les schémas d'axiome et axiomes doivent être opérationnalisés à l'aide des primitives de raisonnement du modèle : règles graphe conceptuel et contraintes positives et négatives. Ces primitives peuvent de plus être utilisées dans un système à base de connaissances de manière automatique ou à la demande de l'utilisateur. Pour opérationnaliser les axiomes et les schémas d'axiomes autres que les relations sorte-de, les signatures et les types d'instances, nous disposons donc :

- De **règles implicites**, permettant la production automatique de connaissance.
- De **règles explicites**, permettant la production de connaissance à la demande de l'utilisateur.
- De **contraintes implicites** (négatives ou positives), permettant le test automatique d'un graphe.
- De **contraintes explicites** (négatives ou positives), permettant le test d'un graphe à la demande de l'utilisateur. [Für, 04]

Les schémas d'axiomes et axiomes étant prés-définis dans le modèle, et les primitives de raisonnement du langage opérationnel étant fixées, il reste à définir les règles de transcription permettant, en fonction des contextes d'usage, de passer des représentations ontologiques des propriétés à leurs possibles représentations opérationnelles. L'idée présidant à l'établissement des règles d'opérationnalisation est de permettre l'inférence autorisée par le schéma d'axiome, ou par l'axiome, s'il est utilisé dans un contexte inférentiel, tout en contrôlant qu'il n'est pas violé par la base de connaissances. Ce contrôle doit bien entendu également exister dans un contexte de validation. Pour chaque axiome, nous fournissons donc des règles d'opérationnalisation qui permettent de générer un ensemble de règles et contraintes constituant la forme opérationnelle de l'axiome. Dans certains cas, un axiome n'engendre au niveau opérationnel aucune contrainte GCs.

> Chaque ontologie exprimée dans le modèle de graphes conceptuels est dite ontologie opérationnelle (computationnelle).

Une *ontologie opérationnelle* (computationnelle) correspond finalement à une ontologie exprimée dans un langage opérationnel (computationnel) – dans notre cas les graphes conceptuels- permettant l'inférence ou le raisonnement.

4.2.1. Scénarii d'usage et contextes d'usage

Au niveau terminologique, la représentation d'un concept ou d'une relation est constituée par un terme, cette représentation ne varie pas en fonction de l'objectif opérationnel[18]. Seules les représentations de la sémantique du domaine doivent être adaptées à l'objectif de l'application envisagée. Décrire le scénario d'usage va donc consister à préciser la façon dont les axiomes et les schémas d'axiome vont servir au raisonnement [Für, 04b].

Spécifier un scénario d'usage, c'est donc spécifier pour chaque axiome (et schéma d'axiome) son rôle dans le système opérationnel qui va utiliser l'ontologie opérationnalisée.

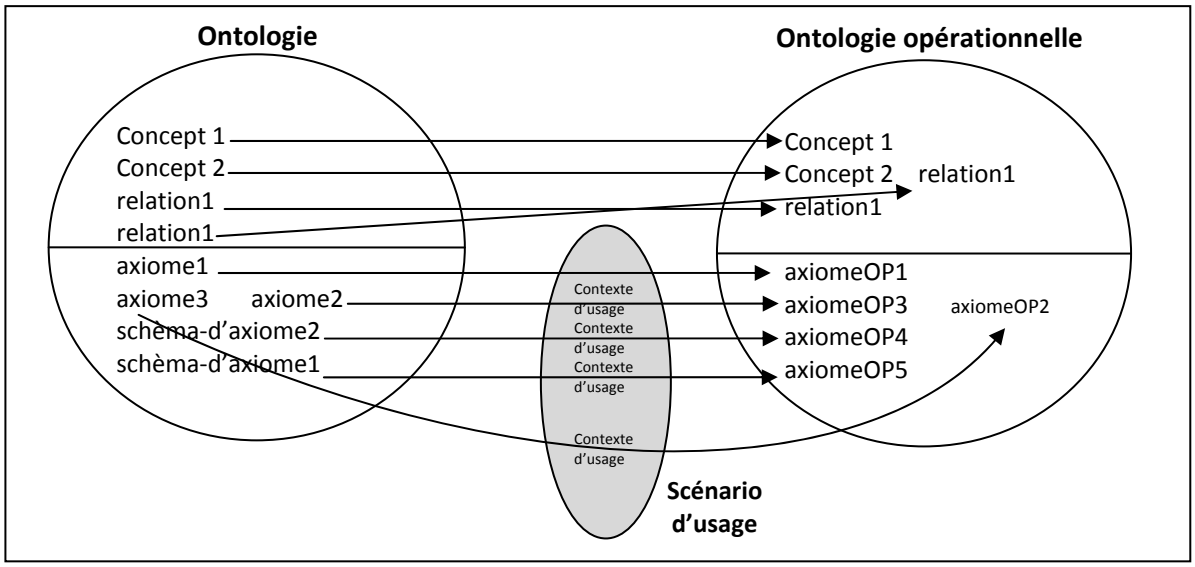

Figure 17. Schéma détaillé d'opérationnalisation d'une ontologie en vue de son utilisation au sein d'un SBC [Für, 04b]

Un **contexte d'usage** d'un axiome est la description du rôle qu'il va jouer dans les raisonnements mis en œuvre dans le système à base de connaissances. L'opérationnalisation doit définir, pour chaque axiome, un contexte d'usage, l'ensemble des contextes d'usage des axiomes et schémas d'axiome de l'ontologie formant le scénario d'usage de l'ontologie pour l'application envisagée.

[18] Les liens sorte-de structurant les hiérarchies de concepts et de relations sont considérés comme des schémas d'axiome et font partie du niveau axiomatique de l'ontologie.

Le niveau terminologique de l'ontologie et l'ensemble des formes opérationnelles des axiomes constituent l'ontologie opérationnelle comme l'illustre la figure 17.

Le contexte d'usage d'un axiome exprimant le rôle de l'axiome dans le système, il conditionne la forme opérationnelle de l'axiome, c'est-à-dire la représentation de l'axiome dans l'ontologie opérationnelle.

4.2.2. Etapes d'opérationnalisation de l'ontologie par les GCs

Les ontologies représentent un ensemble de concepts et relations exprimant les notions utilisées dans un domaine. Cet ensemble doit permettre d'exprimer les connaissances du domaine de manière à pouvoir les intégrer dans des systèmes à base de connaissances, pour l'usage d'inférence et de raisonnement [Bac, 08]. A cette fin, les ontologies sont des référentiels formalisés de manière à ce que leur syntaxe, appropriée à l'inférence, reflète une sémantique pertinente dans le domaine.

Pour modéliser les primitives ontologiques, il faut s'appuyer sur la réalité effective du domaine et la connaissance qu'il véhicule. On s'appuie pour cela sur une démarche en trois étapes:

1) **L'ontologie différentielle** : c'est la modélisation des termes contenus dans les expressions linguistiques (en s'appuyant sur la sémantique de la langue), cette sémantique est une sémantique différentielle et elle est mise en œuvre à travers des principes différentiels où chaque notion se voit définir de manière unique selon ces principes [Für, 04].
2) **L'ontologie référentielle** : c'est la formalisation d'ontologie différentielle, en associant une sémantique formelle référentielle aux concepts de l'ontologie [Für, 04].
3) **L'ontologie opérationnelle** : c'est l'ontologie exprimée dans un langage opérationnel (computationnel) permettant l'inférence ou le raisonnement [Für, 04].

Nous décrivons maintenant l'étape de l'opérationnalisation d'une ontologie par les graphes conceptuels, comme suit :

1) Représentation des connaissances de domaine par le modèle de graphes conceptuels, cette représentation permet de créer un *support* et une *base de faits*.
2) Intégration de l'ontologie opérationnelle dans un système à base de connaissances.
   a. Création de la partie terminologique, cette partie comporte le vocabulaire de l'ontologie (concepts, relation et individus) et définit logiquement les différentes primitives terminologiques tout en décrivant leurs interventions dans la spécification d'une base de connaissances. Les spécifications utilisées en ingénierie ontologique peuvent être

considérées comme des cas particuliers d'axiomes avec une forme prédéfinie, qui structurent l'ensemble de primitives conceptuelles de niveau terminologique, à savoir :

- **Relations entre Concepts** (la subsomption, Sorte-de, l'abstraction, la disjonction).
- **Relations entre Relations** (Sorte-de, la signature, l'incompatibilité, l'exclusivité, la symétrie, la transitivité, la réflexivité, l'anti-réflexivité, l'antisymétrie, l'asymétrie, l'équivalence, l'ordre, etc.).
- **Relations entre Individus** (Same-As, Different-From).

b. Création de la partie assertionelle, cette partie comporte toutes les informations assertionnelles.

3) Construction d'un ensemble de règles.

4) Utilisation de l'ontologie opérationnelle pour inférer de nouvelles connaissances, pour conclure des nouvelles connaissances en interrogeant les faits par des règles. Cette opération s'appuie sur l'utilisation de raisonnement des graphes conceptuels.

5. Utilisation des ontologies opérationnelles pour raisonner

Une fois l'ontologie opérationnelle conçue, on fait appel au raisonnement automatique. La mise en œuvre de cette ontologie opérationnelle dans un moteur d'inférence est possible à l'aide d'un cycle de raisonnement décrit par :

1. L'ajout des faits, où l'utilisateur peut ajouter des faits à la base de connaissance ;
2. L'inférence explicite, l'utilisateur déclenche une règle;
3. L'inférence automatique, le système déclenche et applique les règles implicitement ;
4. La validation semi-automatique ou automatique à l'aide des contraintes implicites et/ou explicites.

Après cela, on détermine le schéma (Cf. figure. 18) d'application des axiomes faisant ou non intervenir l'utilisateur [Für, 04b].

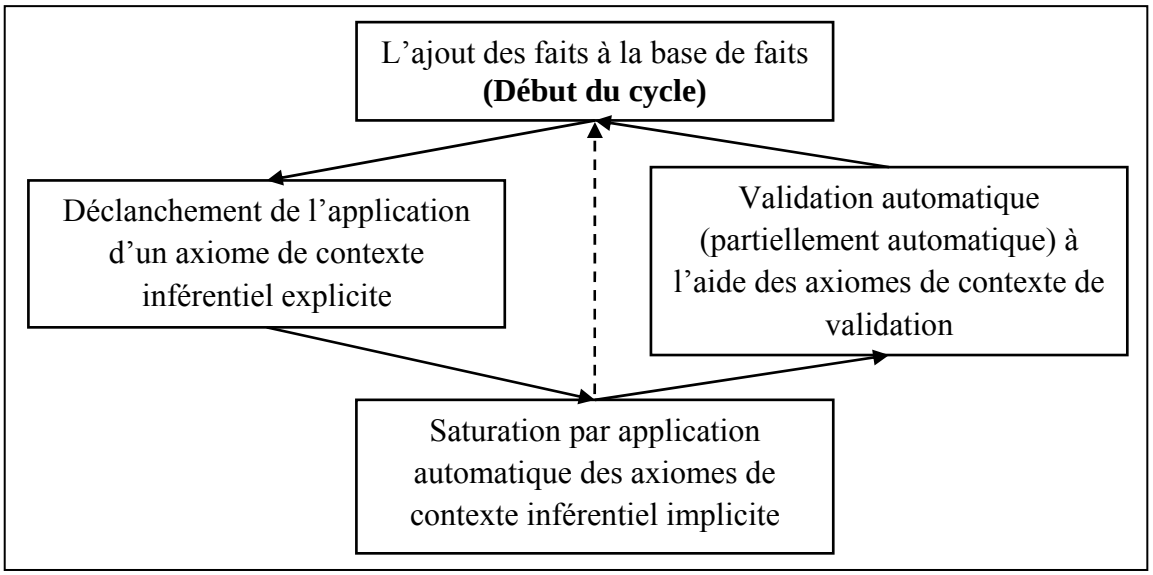

Figure 18. Schéma d'application des axiomes

Le mécanisme de raisonnement de nos ontologies opérationnelles est basé sur le mécanisme de raisonnement des graphes conceptuels.

6. Conclusion

Dans ce chapitre, nous avons exhibé les problèmes qui ont menés à l'opérationnalisation sémantique des connaissances à modéliser. Dans ce cadre nous avons positionné le cadre d'opérationnalisation par rapport aux ontologies et par rapport aux représentations de connaissances. Nous avons aussi présenté des définitions d'opérationnalisation d'ontologies en vue de leur utilisation au sein d'un système à base de connaissances et à la fin nous avons présenté l'utilisation des ontologies opérationnelles pour raisonner.

Ainsi, nous présentons dans le chapitre suivant notre approche, à savoir la construction d'ontologies opérationnelles et cela à l'aide des graphes conceptuels comme modèle de représentation de connaissances.

Chapitre III : Une approche d'opérationnalisation d'ontologies par les graphes conceptuels

1. Introduction

Nous présentons dans ce chapitre notre approche d'opérationnalisation d'ontologies à l'aide des graphes conceptuels. Pour ce faire plusieurs étapes sont proposées afin d'expliciter et de guider l'opérationnalisation de l'ontologie. Par ailleurs, la présentation des méthodologies, formalismes, langages et outils, dans le premier chapitre, est motivée par le fait que chacun d'entre eux est sollicité dans le processus d'opérationnalisation d'ontologies proposé, à une étape ou à une autre.

Les graphes conceptuels constituent le formalisme adopté pour l'expression de l'ontologie semi-formelle, résultat des phases de conceptualisation et d'ontologisation. L'utilisation des graphes conceptuels donne aux ontologies une empreinte opérationnelle, en intégrant l'ontologie dans une base de connaissances.

- Dans une première étape nous présentons une approche de création d'ontologies dans les graphes conceptuels en utilisant la bibliothèque Cogitant
- Dans une seconde étape nous proposons un algorithme de transformation d'une ontologie OWL déjà existante dans le formalisme des graphes conceptuels et plus précisément dans le langage CoGXML[19] de Cogitant.
- Enfin nous présentons le mécanisme de raisonnement déductif sur l'ontologie opérationnelle obtenue.

Pour arriver à des ontologies opérationnelles et comme nous l'avons précisé, nous avons choisi de situer notre démarche dans l'environnement opérationnel offert par Cogitant.

Dans Cogitant, le langage CoGXML permet de codifier une ontologie opérationnelle et de supporter des mécanismes de raisonnement déductif.

Donc, avant de présenter les différentes étapes de notre approche, nous commençons par présenter la codification d'une ontologie en CoGXML.

[19] Langage d'ontologie basé sur les graphes conceptuels et la syntaxe XML

2. Codification de l'ontologie en CoGXML

Notre approche fournit la possibilité d'importer des ontologies OWL et de les transformer en CoGXML (langage d'ontologies permettant de supporter les graphes conceptuels comme un modèle de représentation de connaissances vue dans le cadre du Web), à savoir que ce langage est basé sur le format BCGCT [Gen, 10] (Base de Connaissances Graphes Conceptuels Textuelle) qui permet la représentation du support et les graphes conceptuels et les graphes de règles.

Dans ce qui suit, nous présentons la forme générale d'un fichier et la structure d'un fichier CoGXML.

2.1. Forme générale d'un fichier BCGCT

BCGCT est un fichier structuré ; sa structure combine la description du support, des graphes et graphes de règle. Le format BCGCT permet de représenter sous une forme conjuguant lisibilité et efficacité un support (ensemble partiellement ordonné des types de concepts, des types de relations, signatures des types de relation et relation de conformité) ainsi que des graphes conceptuels et des règles [Gen, 10].

En-effet, un fichier BCGCT permet de définir le support et des graphes construits sur ce support. La structure d'un fichier BCGCT est la suivante : le support est décrit, suivi de graphes ou règles définis sur ce support. Il est toutefois possible et souvent préférable de séparer le stockage du support du stockage des graphes [Gen, 10].

2.1.1. Forme BCGCT d'un support

Un support est composé d'un ensemble partiellement ordonné des types de concepts, un ensemble partiellement ordonné des types de relations, un ensemble partiellement ordonné des signatures des types de relation et de relations de conformité [Gen, 10].

Le nom du support est éventuellement suivi d'une liste de 3 entiers représentant respectivement le nombre de types de concepts, types de relations et marqueurs individuels. Ces renseignements sont optionnels mais peuvent permettre d'accélérer le chargement du support.

- **Forme BCGCT de l'ensemble partiellement ordonné des types de concepts**

  L'ensemble partiellement ordonné des types de concepts est représenté sous la forme de deux listes de déclarations. La première fournit les identificateurs de tous les types de concepts présents dans l'ensemble. C'est dans la déclaration de cette liste que doivent être spécifié les

éventuelles propriétés des types (voir plus bas pour la description de la grammaire BCGCT). La seconde liste définit la relation d'ordre entre les types de concepts par spécification de couples dont le premier élément est inférieur au second. Cette liste est facultative : elle est absente dans le cas d'un ensemble dans lequel tous les éléments sont incomparables [Gen, 10].

- **Forme BCGCT de l'ensemble partiellement ordonné des types de relations**

L'ensemble partiellement ordonné des types de relations est représenté sous la forme de 2 listes de déclarations. La première fournit les identificateurs de tous les types de relations présents dans l'ensemble ainsi que leurs éventuelles propriétés et notamment leur arité et leur signature. Si la propriété Signature est absente, le type de relation n'a pas de signature et l'opération de vérification ne détectera aucune erreur sur des sommets relations utilisant ce type quel que soit leur nombre de voisins. Il est donc préférable de définir cette propriété pour tous les types de relations [Gen, 10].

La seconde liste permet de définir la relation d'ordre entre les types de relations par spécification de couples dont le premier élément est inférieur au second. Cette liste est facultative : elle est absente dans le cas d'un ensemble dans lequel tous les éléments sont incomparables [Gen, 10].

- **Forme BCGCT de la relation de conformité**

La relation de conformité est représentée par des couples *marqueur individuel, identificateur de type de concept* signifiant qu'un marqueur a pour type de concept minimum le type de concept spécifié [Gen, 10].

2.1.2. Forme BCGCT d'un graphe conceptuel

Le format BCGCT impose d'attribuer un identificateur à tout graphe conceptuel. Deux autres critères optionnels sont à disposition des concepteurs d'applications et peuvent être utilisés différemment selon les besoins: la nature (mot clef Nature) du graphe (qui peut être fact, query, ...). Le graphe est ensuite défini par trois listes : celle des sommets concepts, celle des sommets relations et celle des arêtes liant des sommets relations aux sommets concepts [Gen, 10].

2.1.3. Forme BCGCT d'une règle

La règle elle-même est définie en trois parties : le graphe hypothèse, le graphe conclusion et les liens entre ces deux graphes. L'hypothèse est repérée par le mot clef `Hypt`. Après ce mot-clef, le graphe hypothèse doit être donné au format BCGCT classique. La conclusion est repérée par le mot clef `Conc`, et doit être donnée sous la forme d'un graphe BCGCT [Gen, 10].

Enfin, la troisième partie de la description d'une règle concerne les liens entre les deux graphes `Hypt` et `Conc`. Il s'agit d'une liste de couples d'identificateurs de sommets (concepts génériques) dans lequel le premier identificateur repère un sommet du graphe hypothèse et le second un sommet du graphe conclusion. Ces couples de sommets repèrent les points d'attache de la règle.

La grammaire de BCGCT de CoGITaNT est représentée dans l'annexe A.

2.2. Structure d'un fichier CoGXML

Le sous-langage CoGXML est basé sur le langage de balisage sémantique qui permet d'exploiter et de partager des ontologies sur le World Wide Web. La syntaxe de ce sous-langage est fondée sur le langage XML profitant de l'universalité syntaxique de XML.

Afin que la syntaxe RDF/XML de langage CoGXML supporte l'ontologie opérationnelle basée sous les graphes conceptuels, nous nous somme basés dans notre étude sur la bibliothèque **Cogitant**, qui contient des classes dédiées aux opérations d'entrée/sortie pour stocker les supports.

Dans ce qui suit, nous présentons les déclarations des balises qui représentent l'ontologie opérationnelle selon la syntaxe RDF/XML :

1) Vue générale d'un fichier CoGXML

Un fichier CoGXML contient comme tout fichier XML une entête spécifiant la version XML et le DTD associé. La balise <CoGXML> contient la déclaration du support, graphes et règles ; la balise </CoGXML> permet de fermer le fichier.

Vue générale d'un fichier CoGXML

```
<?xml version="1.0"?>
<!DOCTYPE cogxml PUBLIC "-//COGITANT//CoGXML Format Specification
1.3//EN" "http://cogitant.sourceforge.net/cogxml.dtd">
<cogxml>
.../...
</cogxml>
```

2) Support

Le support est composé d'un ensemble ordonné de types concept, un ensemble ordonné de types relation, un ensemble d'individus et les signatures des types relation.

La balise <**Support** > permet de définir le vocabulaire elle comporte la définition de :

- <ConceptTypes> : balise utilisée pour définir un concept <ctype> en donnant un identificateur CONCEPT_ID « chaine de caractères », un label CONCEPT_LABEL « chaine de caractères » et une position CONCEPT_POSITION «couple réel : (x, y)».

La balise type concept permet de définir l'ordre entre des concepts à l'aide de la sous-balise <order>.

- <RelationTypes> : balise utilisée pour définir une relation <rtype> en donnant un identificateur RELATION_ID « chaine de caractères », un label LABEL « chaine de caractères », une signature IDSIGNATURE représentée par un couple de type concept « Concept_label Concept_label ».

La balise type relation permet de définir l'ordre entre des relations à l'aide de la sous-balise <order>.

- <Individus> : balise qui permet de correspondre les individus au concept type à l'aide de la balise MARKER qui a comme argument un ID « chaine de caractères », un label LABEL « chaine de caractères » et le type de concept correspondant IDTYPE « chaine de caractère ».
- < bannedTypes > : balise permet d'identifier les concepts disjoints.

L'exemple suivant permet la codification du domaine de l'être humain sous cogitant, le support est représenté à l'aide des balises <conceptTypes>, <relationTypes> et< Individus>.

Exemple d'un support sous CoGXML

```
<support name="vocabulary">
<conceptTypes>
<ctype id="_ct61" label="Femme" x="175" y="55"> </ctype>
…
<order id1="_ct61" id2="_ct63"/>
…
</conceptTypes>
<relationTypes>
<rtype id="_rt19" idSignature="_ct0 _ct0" label="enrelationavec"></rtype>
<rtype id="_rt24" idSignature="_ct62 _ct0" label="pèrede"></rtype>
……
<order id1="_rt14" id2="_rt19"/>
……
</relationTypes>
< Individus>
<marker   id="2fca54a692f8b112:1c568870:129c693f88e:-7fd9"   idType="_ct0"
label="F"/>
……
</ Individus>
<bannedTypes>
<bannedType> <type id="_ct58"/> <type id="_ct59"/> </bannedType>
</bannedTypes>
</support>
```

3) Graphes

La balise <**Graphe**> permet de définir les graphes conceptuels à savoir, graphes de faits et graphes de confirmation.

Chaque graphe est défini par la sous-balise <graph> qui a comme argument : un identificateur de graphe GRAPHE_ID, un graphe label GRAPHE_LABEL et la nature de graphe GRAPHE_NATURE= : « fait, règle ou confirmation ».

La sous-balise <graph> comporte elle aussi un ensemble de trois sous-balises qui sont :

- <Concept> permet de définir tous les concepts, cette balise a comme argument : Concept-id, Graphe-id-Marker qui correspond à un identificateur de graphe, id-Type et Concept-position.
- <Relation> permet de définir toutes les relations, cette balise a comme argument : Relation-id, Relation-Type et Relation-position donné par x et y.
- <Lien> permet de lier les concepts avec les relations cette balise a comme argument : Concept_id, label et Relation_id.

Exemple d'un graphe de faits sous CoGXML

```
<graph id="_graph1" label="example-simple-fait" nature="fait"
set="exemples-cas réel">
<concept id="_c10" idMarker="3c8268b8d99b6a7d:-29612915:129a7a4b7f8:-
8000" idType="_ct61" referent="individual" x="240" y="130"/>
…
<relation id="_r17" idType="_rt28" x="165" y="35"/>
<Lien cid="_c10" label="1" rid="_r16"/>
<Lien cid="_c11" label="2" rid="_r16"/>
…
</graph>
```

4) Règles

La balise <**Rule**> permet de définir un graphe de règles ; elle est un peu spécifique vue qu'elle comporte deux niveaux (hypothèse et conclusion). Cette balise a comme argument un identificateur ID qui permet d'identifier ce graphe.

L'hypothèse est définie grâce à la balise <**hypt**> qui contient comme sous-balise la balise <graphe> déjà définie, en spécifiant la nature du graphe par GRAPHE_NATURE = règle.

La conclusion est définie grâce à la balise <**conc**> qui contient comme sous-balise la balise <graphe> déjà définie dans 2.

Les liens de coréférences sont définis grâce à la balise <conPts> qui contient deux sous-balises identiques qui sont <couple>. Cette balise a comme arguments Concept_id_source (partie hypothèse) Concept_id_distination (partie conclusion).

Exemple d'un graphe de règle sous CoGXML

```
<rule id="_rule1">
<hypt>
<graph id="_rule1_hypt" label="R8(frerede)" nature="rule"
set="Liens familiaux ">
<concept id="_c44" idType="_ct63" x="220" y="70"/>
<concept id="_c45" idType="_ct0" x="215" y="230"/>
<relation id="_r27" idType="_rt15" x="220" y="155"/>
<Lien cid="_c44" label="1" rid="_r27"/>
<Lien cid="_c45" label="2" rid="_r27"/>
</graph>
</hypt>
<conc>
<graph id="_rule1_conc">
<concept id="_c48" idType="_ct63" x="385" y="70"/>
<concept id="_c47" idType="_ct0" x="385" y="225"/>
<relation id="_r28" idType="_rt26" x="385" y="155"/>
<Lien cid="_c48" label="1" rid="_r28"/>
<Lien cid="_c47" label="2" rid="_r28"/>
</graph>
</conc>
<conPts>
<couple idC1="_c44" idC2="_c48"/>
<couple idC1="_c45" idC2="_c47"/>
</conPts>
</rule>
```

5) CoGXML DTD

Le fichier CoGXML se réfère à une DTD conçue spécialement pour la bibliothèque Cogitant en utilisant la syntaxe suivante :

```
<!DOCTYPE CoGXML PUBLIC "-//COGITANT//CoGXML Format Specification 1.3//EN" "http://cogitant.sourceforge.net/cogxml.dtd">
```

La dernière version de la DTD se trouve dans le lien : http://cogitant.sourceforge.net/cogxml.dtd.

La DTD permet d'interpréter le fichier CoGXML.

3. Une approche pour l'opérationnalisation d'ontologies fondée sur les graphes conceptuels

Notre travail porte sur la définition d'un processus d'opérationnalisation des ontologies, le choix d'un modèle de représentation de connaissances, pour définir les mécanismes d'opérationnalisation et l'étude de la construction d'ontologies pour un support au raisonnement.

Nous avons choisi pour l'opérationnalisation des ontologies le formalisme de graphes conceptuels, qui permet de représenter les ontologies opérationnelles et de raisonner dessus.

Dans notre approche, l'opérationnalisation d'ontologie s'effectue dès la création, en se basant sur les graphes conceptuels ou lors d'importation d'ontologies existantes par transformation.

Nous abordons dans notre travail les deux cas suivants :

1. La création d'une ontologie dans le formalisme des graphes conceptuels.
2. L'importation d'une ontologie OWL existante.

Nous utilisons la bibliothèque cogitant comme support.

3.1. Création d'une ontologie dans le formalisme des graphes conceptuels

Notre approche se base essentiellement sur la méthodologie METHONTOLOGY pour la construction de l'ontologie opérationnelle. Nous créons une ontologie opérationnelle en démarrant de zéro, et nous formalisons à la sortie de la phase d'ontologisation le modèle conceptuel par le formalisme des graphes conceptuels.

Donc, il s'agit de transformer le modèle conceptuel d'un treillis formel avec ces définitions formelles de concepts à un treillis computationnel auquel nous rajoutons du code permettant l'opérationnalisation. Cette transformation consiste à proposer une traduction dans les graphes conceptuels des connecteurs de l'ontologie [Mug, 96].

Notre approche s'articule autour des phases conceptualisation, ontologisation et opérationnalisation présentées dans la figure 19.

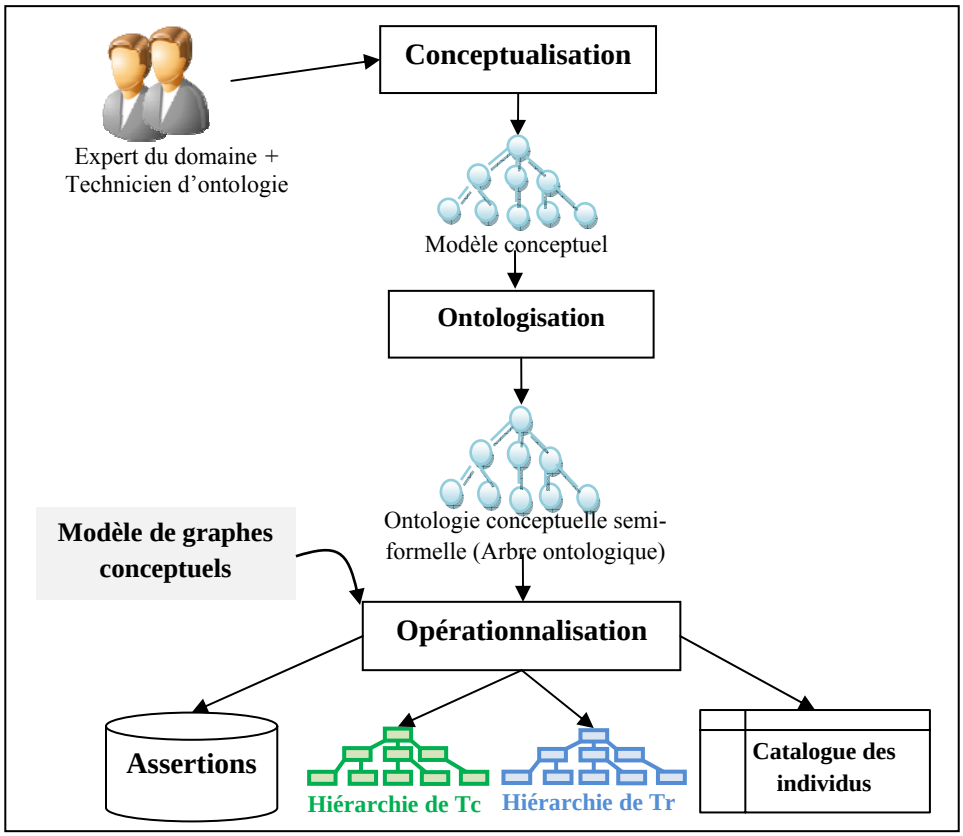

Figure 19. Étapes de construction d'une ontologie opérationnelle

3.1.1. Phase de conceptualisation

Cette phase (Cf. figure. 20) consiste à créer une ontologie conceptuelle en démarrant du zéro. Elle permet d'identifier précisément, à partir du corpus, les objets conceptuels propres au domaine considéré, les relations pouvant lier ces concepts et la sémantique de ces relations. Nous obtenons un modèle conceptuel, informel exprimé en langage naturel potentiellement ambigu.

Dans notre processus de construction cette phase se compose de trois étapes, à savoir :

Etape 1 : dans cette étape l'acquisition des connaissances se fait manuellement des ressources de données brutes. On fait appel à deux agents : un technicien d'ontologie qui est responsable de la création (c'est un connaisseur des étapes de la construction) et un expert de domaine afin de faciliter la tache du technicien qui n'est pas sensé connaitre le domaine sur lequel l'ontologie sera développée. La sortie de cette phase donne un corpus informel ou semi-formel qui n'est qu'un modèle pré-conceptuel.

Etape 2 : cette étape consiste à créer des nouvelles ontologies via les corpus sortants de la phase précédente ou la réutilisation des ontologies déjà misent dans la base de connaissances.

Le module de construction contient les méthodes qui permettent:

- La construction de nouvelles ontologies à partir de zéro.
- La réingénierie d'ontologies.

La fusion ou intégration d'ontologies.

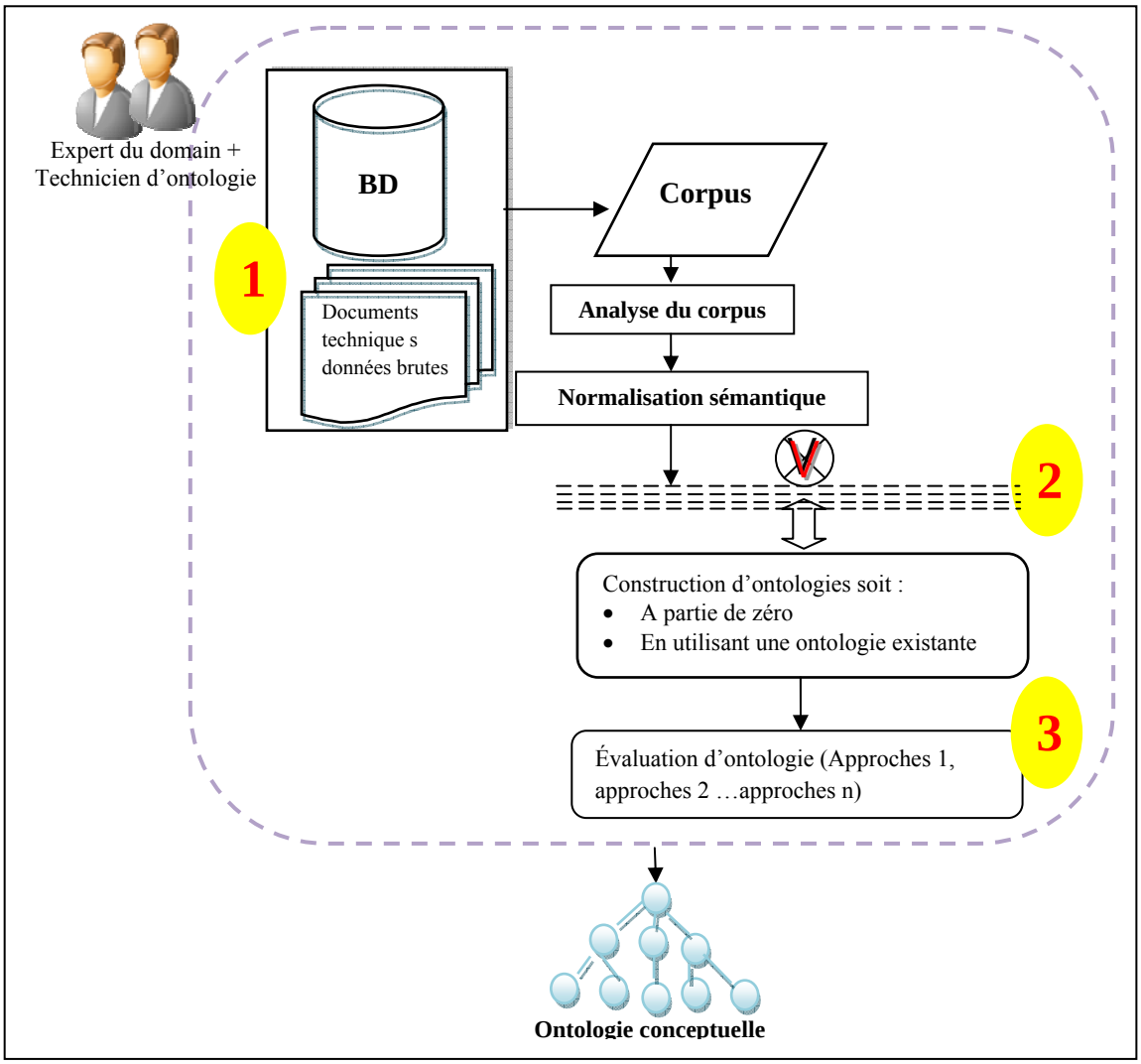

Figure 20. La phase de conceptualisation

Etape 3 : cette étape sert à émettre un jugement technique sur l'ontologie. Elle doit être utilisée dans un environnement, et associée à des documentations respectant le cadre de référence. Le cadre de référence peut être une spécification de besoins, questions de compétences.

La nouvelle ontologie résultante est sous forme d'un treillis qui contient des concepts reliés par des relations, autrement vue comme un arbre de concepts sémantique. Ce treillis décrit la sémantique en langage naturel en indiquant : les instances, les liens entre eux et leurs propriétés. Cette description se lie avec la partie corpus qui met en évidence sa sémantique.

Le processus de conceptualisation mène ainsi à la construction d'un **modèle conceptuel**, qui décrit les connaissances du domaine à l'aide des éléments terminologiques et sémantiques. Cependant, ce modèle n'est pas formel, il peut être complètement informel (exprimé en langage naturel) ou semi-formel (combinant langage naturel et propriétés formelles). Une fois le treillis conçu, il convient donc, pour l'utiliser dans une machine, de formaliser le modèle conceptuel obtenu. C'est l'objet du processus d'ontologisation.

3.1.2. Phase d'ontologisation

Cette phase représente un module d'ontologisation qui sert à l'acquisition et la modélisation des connaissances ontologiques par la transcription des connaissances dans un langage de représentation d'ontologie. Ce langage doit être aussi générique que possible. Le résultat de cette phase aboutit à une ontologie conceptuelle formelle.

L'ontologisation consiste à structurer et formaliser autant que possible la conceptualisation pour construire une ontologie spécifiant la terminologie et la sémantique du domaine à travers un modèle doté d'une sémantique formelle (mais non opérationnelle).

Comme entrée, nous avons l'ontologie conçue dans la 3^{éme} étape, et comme sortie, on aura un treillis de concepts semi-formels (des fois formels) qui lui fournit un libellé dont la sémantique se définit par une extension d'objets. Les concepts formels sont soit des concepts sémantiques dont on reprend le libellé linguistique et auquel on associe des référents conformément à l'engagement sémantique, soit de nouveaux concepts définis formellement par intersection de concepts formels déjà définis. Chaque concept formel est défini par un engagement ontologique qui spécifie quels objets doivent exister dans le domaine pour utiliser le concept conformément à sa signification formelle.

Chaque libellé linguistique qui caractérise le concept a une interprétation qui est contrainte par les principes différentiels. Ces principes correspondent à l'engagement sémantique qu'il faut respecter. Dans notre processus nous utilisons les quatre principes différentiels proposés par *Bachimont* [Bac, 01] suivants:

- Un concept partage l'intension de son concept père (**principe de similarité)**;
- L'intension d'un concept est différente de celle de son concept père, sinon il n'y aurait pas besoin de définir le concept fils (**principe de différence**);
- Une propriété est commune aux concepts frères issus du même concept père mais s'exprime différemment pour chaque frère (**principe de sémantique unique)**;

- Les frères doivent tous être incompatibles, sinon il n'y aurait pas besoin de les définir tous **(principe d'opposition)**.

En-outre, le processus d'ontologisation proposé se base sur les cinq critères de *Gruber* [Gru, 93] qui permettent de guider le processus d'ontologisation:

- La clarté et l'objectivité des définitions, qui doivent être indépendantes de tout choix d'implémentation;
- La cohérence (consistance logique) des axiomes ;
- L'extensibilité d'une ontologie, c'est-à-dire la possibilité de l'étendre sans modification ;
- La minimalité des postulats de formalisation, ce qui assure une bonne portabilité ;
- La minimalité du vocabulaire, c'est-à-dire l'expressivité maximum de chaque terme.

Après la structuration du modèle conceptuel, il faut l'exprimer dans un langage formel de représentation d'ontologies. La sémantique de la subsomption n'est cependant pas toujours conforme aux principes différentiels cités précédemment. L'étape d'ontologisation conduit ainsi à la construction d'une ontologie semi-formelle. Ce caractère (semi-formel) lui interdit d'être utilisée telle quelle dans un système à base de connaissances.

3.1.3. Phase d'opérationnalisation

Cette phase présente le niveau opérationnel, dont les concepts et relations sont définis par les opérations qu'il est possible de leur appliquer. Cette phase permet donc, la transformation d'une ontologie semi-formelle à l'aide des graphes conceptuels d'un treillis semi-formel (arbre avec des définitions formelles des concepts) à un treillis computationnel (treillis formel auquel on rajoute du code permettant l'opérationnalisation). En se basant sur les ressemblances des graphes conceptuels (l'aspect opérationnel) avec les ontologies, et l'ontologie sera découpée principalement en deux couches : vocabulaire et les assertions (base de faits).

3.1.3.1. Représentation d'une ontologie par les graphes conceptuels

Les ontologies et les graphes conceptuels présentent de nombreuses similarités, à savoir que les deux modèles distinguent entre les connaissances assertionnelles et connaissances terminologiques (Cf. figure 21).

Dans notre approche la correspondance des primitives de l'ontologie au support des graphes conceptuels est basée sur la définition de *Mugnier* [Mug, 96]. Rappelons que *Mugnier* a défini le support d'un graphe conceptuel par un quintuplet ontologique S= (Tc, Tr, σ, I,τ), avec :

- T_C est l'ensemble des types de concepts, plus le plus grand élément nommé universel (T) et un plus petit élément nommé absurde ($\perp$).
- T_R est un groupe d'ensembles des types de relations de même arité. $T_R = T_{Ri1} \cup \ldots \cup T_{Rip}$, où T_{Rij} est l'ensemble des types de relations d'arité ij, ij $\neq 0$.
- σ associe à tout type de relation le type maximal de chacun de ses arguments.
- I est l'ensemble des marqueurs individuels $\cup$ le marqueur générique (*).
- τ est une application de I dans $T_C \setminus \{\perp\}$, qui associe à tout marqueur individuel m un type de concept t.

Donc, une ontologie O est définie graphiquement et grâce au graphes conceptuels en un tuple $(Tc, Tr, I)_O$, avec Tc est un ensemble de types concept, Tr est un ensemble de types relation et I est l'ensemble des individus [Cor, 08].

Le support des graphes conceptuels définit le vocabulaire de base de l'ontologie avec lequel nous représentons des connaissances du domaine, il permet de représenter le niveau terminologique de l'ontologie par la description de :

- **La hiérarchie de types de concept** (Tc): c'est un treillis de types concepts tel que, s'il existe des types t1 et t2 (ontologique), alors il existe forcément un type correspondant à la conjonction (union) de t1 et t2 et un type correspondant à la disjonction (intersection) de t1 et t2. Ainsi, Tc est le treillis engendré par la fermeture d'un ensemble de types, par les opérations de conjonction (intersection) et disjonction (union).
- **La hiérarchie de types de relation** (Tr): détermine la relation entre deux types de concept, il explicite le rôle particulier d'un concept spécifique ou d'un ensemble de concepts dans son contexte conceptuel.
- **Les individus** (I): le support contient également un ensemble d'individus typés, à tout individu m on associe le plus petit type de concept t tel que m soit un t (m Est un t).

Le niveau assertionel des graphes conceptuels permet de représenter les faits à l'aide d'un vocabulaire qui décrit le support.

Il est à noter que la hiérarchie de types n'est pas forcement une taxonomie, car un type peut avoir plusieurs super-types.

Les deux ordres sur les ensembles de types concept et relation sont interprétés comme des liens **Sorte-de** des ontologies, et cela à l'aide de l'opération de la **subsomption** ($\leq$) des graphes conceptuels.

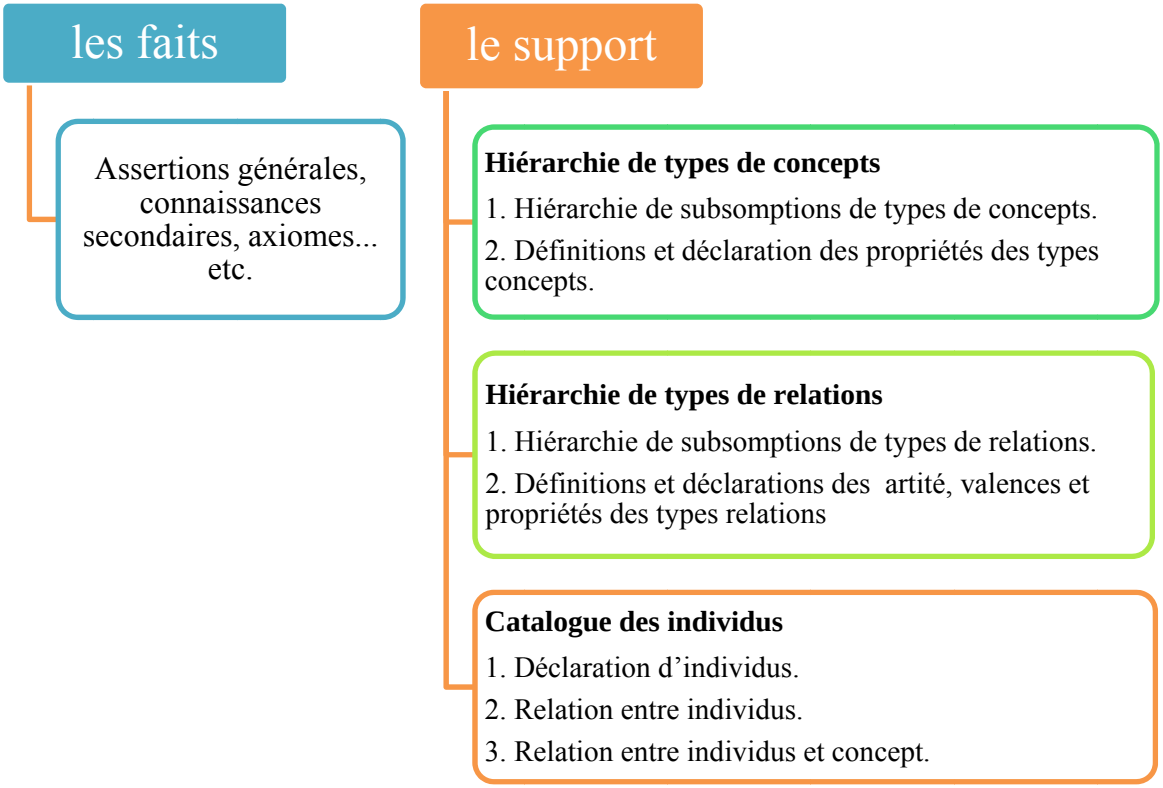

Figure 21. L'ontologie dans le modèle de graphe conceptuel

3.1.3.2. Les ontologies comme bases de connaissances

Une fois l'ontologie opérationnelle est définie, celle-ci devient une base de connaissances sur laquelle, on peut effectuer des raisonnements. La base de connaissances sera découpée en deux parties : un **vocabulaire** fourni par une ontologie respectant l'ordre de **subsomption** et des **assertions** qui permettent de représenter les faits à l'aide d'un vocabulaire.

Nous représentons la base de connaissances par (Cf. figure 22) :

```
["BaseDeConnaissances" :
   {
        ["Vocabulaire" :/* connaissances générales ou ontologie*/
          {
             ["HiérarchieTypeConcept " : GC]
             ["HiérarchieTypeRelation ": GC]
             ["CatalogeDesIndividue ": Table]
          }
        ["Assertion" : GC]
   }]
```

Figure 22. L'ontologie comme base de connaissances

Par la suite nous présentons syntaxiquement et sémantiquement les différents composants de la base de connaissances (le vocabulaire et les assertions), ainsi que la relation d'ordre qui permet l'organisation des hiérarchies.

I. La subsomption comme relation d'ordre

Les relations d'ordre utilisées dans la construction de différentes hiérarchisations de la base de connaissances à l'aide des graphes conceptuels ne possèdent pas une sémantique franche, ce qui donne la possibilité aux concepteurs et aux utilisateurs d'ajuster leur sens aux besoins de la modélisation souhaitée [Mug, 96]. La sémantique de la subsomption est exprimée par la formule logique de premier ordre suivante :

- Si un type de concept C subsume un autre type de concept C', alors toute instance de C' appartient à C : $((C > C') \Leftrightarrow (\forall x\, C'(x) \rightarrow C(x)))$.
- Si un vecteur d'individus vérifie une relation R', alors il vérifie aussi la relation R qui la subsume : $(R>R') \Leftrightarrow (\forall x_1,x_2 \ldots x_n\, (T_1(x_1) \wedge T_2(x_2) \wedge \ldots \wedge T_n(x_n) \wedge R'(x_1,x_2,\ldots..,x_n) \rightarrow R(x_1,x_2,\ldots..,x_n))$.

(C type concept, x individu, R type relation, n arité de R, $(T_1 \ldots T_n)$ signature de R.)

Il est à noter que deux relations R et R' de valences respectives n et m tel que $n \neq m$ n'admettent pas de relations d'ordre entre elles (R>R' et R'>R seraient axiomatiquement fausses).
Par conséquent nous considérons la relation de spécialisation notée < comme étant l'inverse de la subsomption, de même pour la relation d'équivalence notée = comme étant une relation par une double subsomption.

II. Vocabulaire

Le vocabulaire permet d'une part, l'organisation de la base de connaissances, sa gestion, et d'autre part, la construction des graphes. Les propriétés de vocabulaire définissent logiquement les différentes primitives terminologiques tout en décrivant leurs interventions dans la spécification d'une base de connaissances.

Le vocabulaire se compose de trois primitives «hiérarchie de types de concept, hiérarchie de types de relation et catalogue d'individus», définis comme suit :

1) Hiérarchie de types de concepts

La hiérarchie de types de concept correspond à un graphe dans lequel les nœuds représentent des types de concept respectant la relation d'ordre. Les **concepts** des graphes conceptuels par leur définition, permettent de désigner n'importe quelle entité (individus, ensembles,

collections…etc.) du domaine modélisé et d'y attribuer des notions existentielles, universelles, de cardinalités ou autres, grâce à la spécification de quantifieurs.

Nous décrivons les informations concernant les types de concepts qui seront présentées dans un contexte de type ConceptTypeHierarchy défini comme suit [Bouraï 11] :

```
"[" "ConceptTypeHierarchy"
  (TypeLabelOrdering |TypeDefinition|Typeproperty)* "]"
```

1. La construction de TypeLabelOrdering, se fait grâce aux relations d'équivalence (EQ) entre types de concept, de subsomption (GT) et de spécialisation de concept (LT).
2. La déclaration de types TypeDefinition, permet de lier les types et leurs définitions en lambda abstractions. L'expression des définitions de types pourrait aussi se faire par la combinaison des constructeurs suivants :

   - Une intersection : décrite par un contexte de type IntersectionOf ayant pour descripteur un ensemble de concepts singletons ayant pour type les types à unir et un quantifieur vide.
   - Une union : décrite par un contexte de type UnionOf ayant pour descripteur un ensemble de concepts singletons ayant pour type les types à unir et un quantifieur vide.
   - Un complément: décrit par un contexte de type ComplementOF ayant pour descripteur un singleton ayant pour type le type dont nous voulons définir le complément.
   - Une énumération : se fera par la spécification des marqueurs d'individus de l'ensemble à construire dans le remplisseur d'un contexte de type Enumeration à l'aide de singletons contenants les types associés aux marqueurs et le nom du marqueur.
   - Une restriction de propriétés : la restriction Hasvalue permet de construire un ensemble d'individus qui sont en relation avec les individus y_1, y_2…y_{n-1} par une relation R.
   - Une restriction de cardinalité : permet d'associer à chaque type de concept le nombre d'individus distincts pouvant être compatibles avec lui, elles prennent comme signature (%entier, %entier).
   - Les relations sup, inf et equal : de valence 2, elles prennent comme signature (%entier, %entier) ; et permettent de déclarer qu'un nombre est supérieur (pour la relation sup), inférieur (pour la relation inf) ou égal (pour la relation equal) par rapport à un autre.
3. Le Typeproperty permet de définir des schémas d'axiomes de mise en relation et de déclaration de propriétés sur les types de concepts:

   - Incompatibilité de types : pour exprimer le fait que deux types de concept C_1 et C_2 ne peuvent pas être compatibles, nous introduisons la relation Disjoint, qui prend en

paramètre deux types de concepts. Sa sémantique est donnée par la formule prédicative suivante :

$(C_1$ est disjoint avec $C_2) \Leftrightarrow (\forall x\ (C_1(x) \Rightarrow \neg C_2(x)) \wedge (C_2(x) \Rightarrow \neg C_1(x)))$.

- Abstraction de types : un type étiqueté comme étant abstrait ne peut pas avoir d'instance directe. Un tel type peut être utilisé pour l'organisation de la hiérarchie des types de concepts.

La syntaxe définie pour la manipulation d'un type de concept est:

(Def (ConceptType « lebelet »)][« définition de concept»]

La représentation graphique est un rectangle : **lebelet : <Individus∪ *>**

La figure 23 montre le graphe conceptuel de la définition du concept Prof_d_anglais, qui a comme syntaxe: (Def [ConceptType " Prof_d_anglais"] [Lambda (humain *x) (Enseigner ?x [Anglais])).

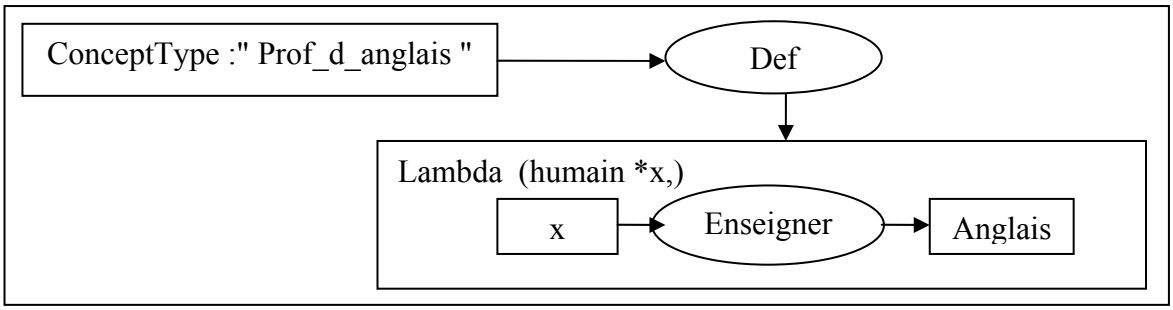

Figure 23. La définition du concept Prof_d_anglais par un GC

La représentation graphique de concept Prof_d_anglais est : Prof_d_anglais : *

La figure 24 montre la hiérarchie de type concept du domaine de l'être humain (les arcs représentent la relation de subsomption), qui a comme syntaxe :

(GT [ConceptType " Humain"] [ConceptType " Femelle"])
(GT [ConceptType " Humain"] [ConceptType " Male"])
(GT [ConceptType " Femelle "] [ConceptType " Fille "])
(GT [ConceptType " Femelle "] [ConceptType " Femme"])
(GT [ConceptType " Male"] [ConceptType " Garçon"])
(GT [ConceptType " Male"] [ConceptType " Homme"])
(GT [ConceptType " Femme "] [ConceptType "⊥"])
(GT [ConceptType " Fille "] [ConceptType "⊥"])
(GT [ConceptType " Garçon "] [ConceptType "⊥"])
(GT [ConceptType " Homme"] [ConceptType "⊥"])

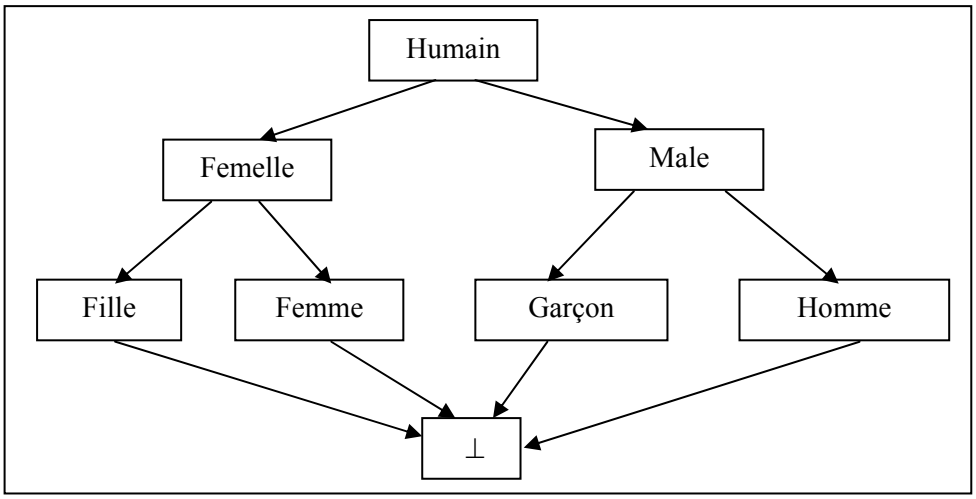

Figure 24. Hiérarchie de type concept de l'être humain

2) Hiérarchie de types de relations

La hiérarchie de types de relation correspond à un graphe représentant les types relations entre les concepts. Ces relations conceptuelles ayant comme signature (Domaine, Image), leur arité dépend de l'utilisation du domaine.

Nous définissons certaines informations concernant les types de relations de notre base. Ces informations permettent de créer des liens entre les relations d'une façon logique, présentés dans un contexte de type RelationTypeHierarchy défini comme suit [Bouraï 11]:

"[" "RelationTypeHierarchy" (RelationLabelOrdering|ValenceSpec |RelationDefinition|RelationProprety)*"]"

1. La construction de RelationLabelOrdering, s'effectue par les relations d'équivalence (EQ) entre types de relations, de subsomption (GT) et de spécialisation de types de relations(LT).
2. La définition de la valence se fait par la relation HAS qui associe à un type relations le nombre exact de types concepts qu'il comporte, et la RelationDefinition permet de lier les types relation et leurs définitions en lambda calcul.
3. RelationProprety permet de définir des schémas d'axiomes de mise en relation et de déclaration de propriétés sur les types de relations :

   - Symétriques SYM: set définie par la sémantique suivante : si une relation R de valence 2n (n entier positif) et de signature (T_1,T_2,T_n, T_1, T_2,…, T_n) est dite symétrique alors : $\forall x_1$, x_2,…x_n, x_{n+1}, x_{n+2},.., x_{2n} $T(x_1) \wedge T(x_2) \wedge..\wedge T(x_n) \wedge T(x_{n+1}) \wedge T(x_{n+2}) \wedge...\wedge T(x_{2n})(R(x_{1,x2},…x_n,x_{n+1},x_{n+2},..,x_{2n}) \Rightarrow R(x_{n+1},x_{n+2},..,x_{2n},x_1,x_2,…x_n))$.

La figure 24 montre la symétrie de la relation Frère-de.

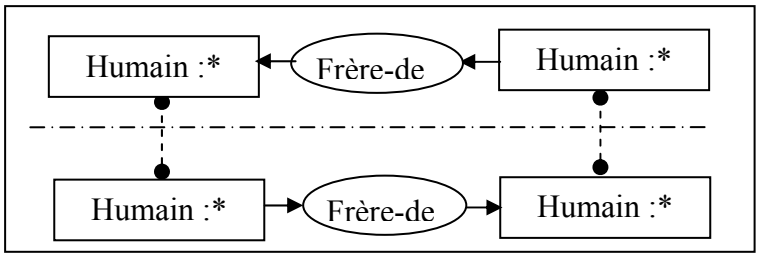

Figure 25. La symétrie de la relation Frère-de

- Fonctionnelles FONC@m: définie par la sémantique suivante : si une relation R de valence n et de signature (T_1, T_2,…, T_n) est étiquetée fonctionnelle de paramètre m (n>m >0) alors : $\forall x_1, x_2,\ldots x_n, y_1,y_2,..y_{n-m}\ T_1(x_1) \wedge T_2(x_2) \wedge..\wedge T_n(x_n)\wedge T_{m+1}(y_1) \wedge T_{m+2}(y_2) \wedge\ldots\wedge T_n(y_{n-m}) \wedge R(x_1,x_2,..,x_n)\wedge R(x_1,x_2,..,x_m ,y_{1,y2},..y_{n-m})\Rightarrow (=(x_{m+1},y_1)\wedge =(x_{m+2},y_2)\wedge\ldots\wedge=(x_n,y_{n-m})$.

- Transitives TRANS : si une relation R de valence 2n (n entier positif) et de signature (T_1, T_2, T_n, T_1, T_2,…,T_n) est étiquetée transitive alors : $\forall x_1,x_2,\ldots x_n,y_1,y_2,..,y_n,z_1,z_2,..,z_n\ T_1(x_1)\wedge T_2(x_2)\wedge..\wedge T_n(x_n)\wedge\ T_1(y_1)\wedge T_2(y_2)\wedge..\wedge T_n(y_n)\ \wedge\ T_1(z_1)\wedge T_2(z_2)\wedge..\ \wedge T_n(z_n)\ \wedge (R(x_1,x_2,\ldots x_n, ,y_1,y_2,..,y_n) \wedge R(,y_1,y_2,..,y_n,z_1,z_2,\ldots z_n))\Rightarrow R(x_1,x_2,\ldots x_n, ,z_1,z_2,..,z_n)$.

R est le prédicat correspondant à la relation conceptuelle R, et $T_1,T_2,\ldots T_n$ correspondent respectivement aux types de concepts $T_1,T_2,..T_n$.

La figure 26 montre la transitivité de la relation Ancêtre-de.

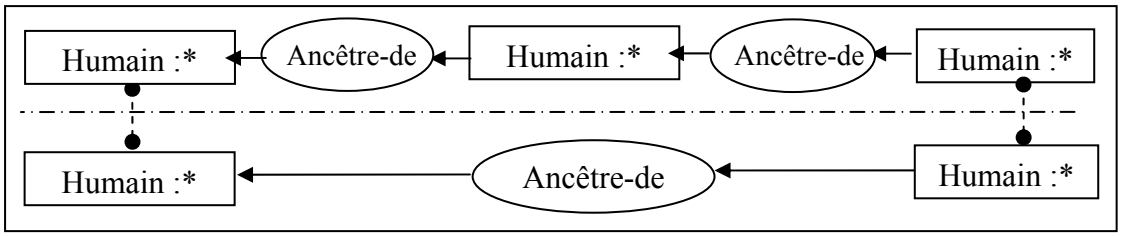

Figure 26. La transitivité de la relation Ancêtre-de

La syntaxe de manipulation est la suivante [Bou, 11] :

(Def (RelationType « lebelet » «areté »)][fonction ()]

La représentation graphique est une ellipse: **lebelet (arité$_1$, arité$_2$,…)**

La figure 27 montre la définition de la relation Grand-Pére-De.

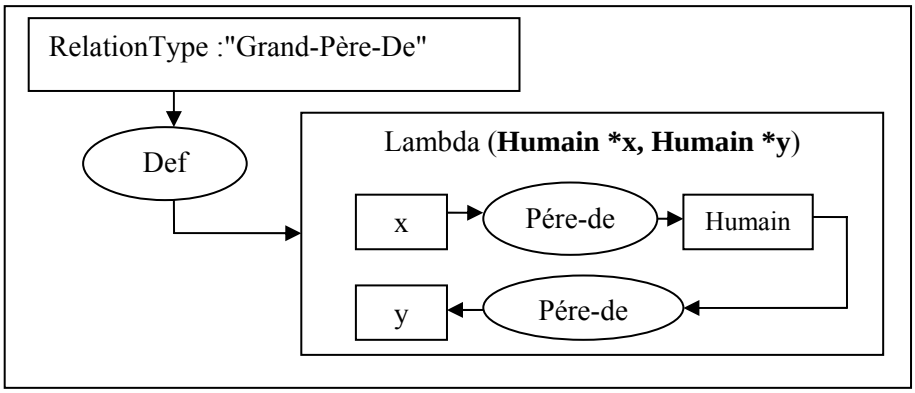

Figure 27. Définition de la relation Grand-Pére-De

La représentation graphique de la relation Grand-Père-De est : 

La figure 28 montre un exemple de la hiérarchie de type relations liés au domaine de l'être humain.

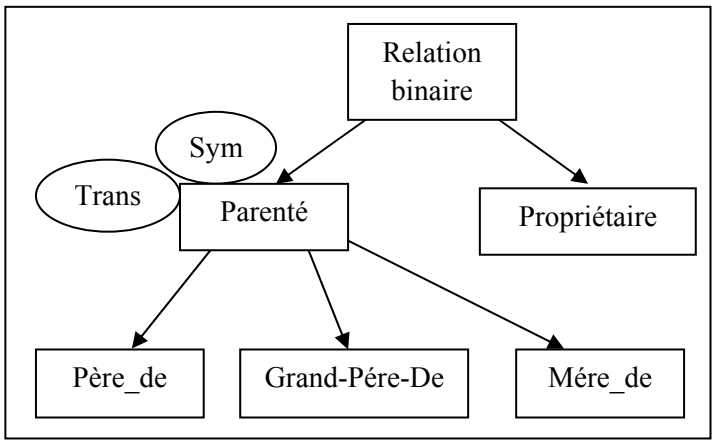

Figure 28. Hiérarchie de type relation de l'être humain

3) Catalogue d'individus

Le catalogue d'individu est un catalogue dans lequel on déclare tous les individus manipulés dans la base de connaissances et cela en spécifiant pour chaque individu : son nom et son type (privilège type).

La représentation graphique est un tableau organisé comme suit:

Nom	Privilège type
Chaine de caractères	Type concept

Nous allons maintenant spécifier quelques attributs récurrents dans la déclaration d'individus tels que : Same-As et Different-From:

- Same-As[20] : cette relation permet de déclarer que deux individus sont identiques et donc interchangeables entre eux. SameAs sera déclarée comme étant une relation de valence 2 et de signature (individu, individu).
- Different-From: la relation complémentaire de SameAs, pourrait être utilisée par exemple pour définir que deux marqueurs ne peuvent pas pointer le même individu.

La figure 29 montre un graphe conceptuel contenant des individus : {Ali, Maria, Mouhamed, Samia, Ahmed}.

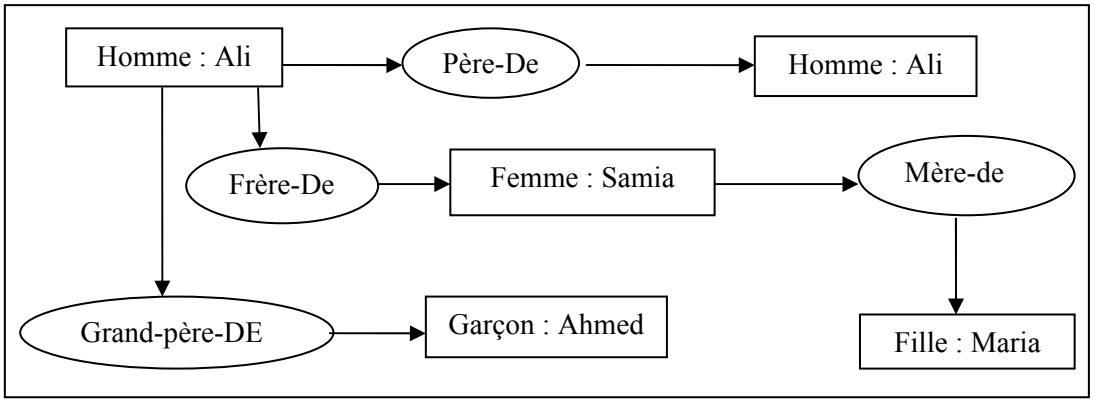

Figure 29. Exemple d'un graphe conceptuel contenant des individus

III. Les assertions

Les assertions nommées aussi base de faits sont des éléments de type déclaratif. Un fait constitue un atome de connaissance ; il est représenté sous forme d'un ensemble de graphes (Cf. figure 30) composés des nœuds de concepts présentés sous forme [type de concept,*] et des nœuds de relation qui relient les nœuds de concept. Il représente le domaine sous forme de graphes conceptuels.

Donc, la base de faits est composée de contextes de type assertion, contenant des graphes conceptuels définissant des faits généraux, des situations et autres connaissances, et cela en utilisant les primitives conceptuelles décrites grâce au support (vocabulaire). Donc, elle représente une base d'informations décrivant le domaine.

Ces graphes seront interrogés ultérieurement à l'aide des requêtes (représentant des problèmes), afin d'inférer de nouvelles connaissances ou de vérifier des connaissances existantes.

En résultat nous obtenons une ontologie opérationnelle codée en CoGXML.

[20] A ne pas confondre avec la notion de coréférence, qui remplace les nœuds d'un ensemble de coréférence par leur nœud dominant

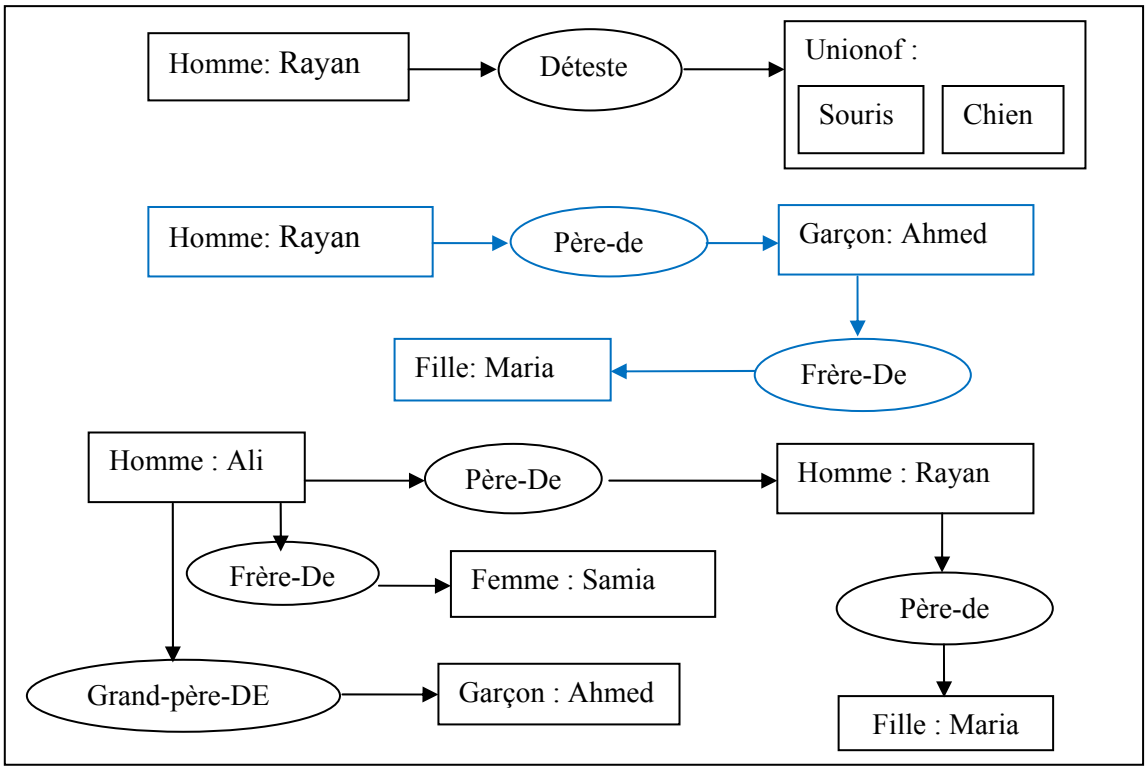

Figure 30. Exemple des graphes de faits

3.2. Importation d'une ontologie existante

Dans ce cas nous proposons un algorithme qui permet de transformer les ontologies OWL en CoGXML. En-effet, nous définissons des règles qui permettent d'exprimer les éléments d'une ontologie OWL[21] suivant de nouvelles classes qui expriment les éléments de graphes conceptuels. Ces règles permettent la création des triplets ontologiques par la correspondance de types concept et types relation avec les éléments RDFs.

3.2.1. L'algorithme de transformation d'une ontologie OWL en CoGXML

OWL (Ontology Web Language) permet de définir et de gérer des ontologies. Il permet de représenter et de manipuler les principaux composants ontologiques à savoir les classes, les individus qui sont des instances de ces classes, les propriétés (ou rôles) entre eux et les axiomes régissant la base de connaissances.

Ces composants permettent :

- La définition des classes et de relations entre elles.
- La définition des propriétés et des relations entre elles.
- La définition des instances de classes ou des propriétés et des relations entre elles.

[21] OWL est un langage de balisage sémantique qui permet d'exploiter et de partager des ontologies sur le World Wide Web. Il est développé comme une extension du vocabulaire de RDF (Resource Description Framework).

- La définition des informations de support.

OWL utilise la syntaxe de RDF et RDF Shema :

- Toutes les variétés d'OWL utilisent RDF dans leurs syntaxes.
- Les instances sont déclarées comme dans RDF, en utilisant les constructeurs RDF.
- Les constructeurs OWL comme owl:Class, owl:DatatypeProperty et owl:ObjectProperty sont tous des spécialisations de leurs homologues RDF.

La figure 31, montre la relation de sous classe entre quelques constructeurs OWL et RDF/RDFS.

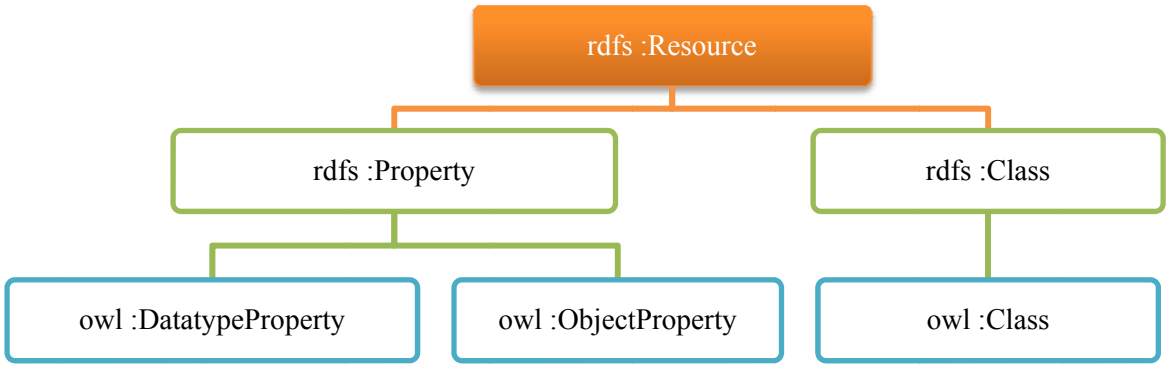

Figure 31. La hiérarchie des éléments OWL et RDF/RDFS

OWL définit trois types d'éléments pour la description des ontologies: les classes, les propriétés et les individus. Nous montrons ici la réécriture d'une ontologie OWL sous forme de balises CoGXML.

Nous proposons un algorithme de transformation qui parcourt les balises OWL et les transforme selon les graphes conceptuels.

Nous proposons des règles pour la transformation des classes et des propriétés de l'ontologie et nous décrivons par la suite le fonctionnement général de l'algorithme.

- **Règles de transformation d'une classe:** une classe fournit un mécanisme pour regrouper des ressources ayant les mêmes caractéristiques. Comme les classes RDF, toute classe OWL est associée à un ensemble d'individus, nommé l'extension de la classe. Les individus dans l'extension de la classe sont appelés les instances de la classe. Une classe a une signification intensionnelle (le concept sous-jacent) qui le relié mais pas égale à l'extension de la classe.
  Une classe OWL peut être définie en utilisant le constructeur ***owl : class*** [Ant, 04], en forme de graphes conceptuels une classe est définie par la balise <ConceptTypes>.

Les classes d'une ontologie peuvent être organisées par une hiérarchie de spécialisation, en utilisant l'axiome *subClassOf*, en forme de graphes conceptuels la balise type concept permet de définir l'ordre entre des concepts à l'aide de la sous-balise <order>.

Il existe dans toute ontologie OWL une superclasse, nommée ***Thing*** ; dans une hiérarchie de types concept GC la superclasse est représentée par T, dont toutes les autres classes sont des sous-classes. Il existe également une classe nommée ***NoThing*** qui est sous-classe de toutes les classes OWL ; dans une hiérarchie de types concept la sous-classe est représentée par ⊥. Cette classe ne peut avoir aucune instance.

Nous définissons la règle $\mathbf{R_{classe}}$ qui transforme les classes OWL vers la forme de graphes conceptuels, comme suit :

Régle$_{classe}$:

C concept type	←	*C owl : class*	{*C* unary predicate}
$C \leq D$	←	*C owl:subClassOf D*	$\{\forall x(C(x) \rightarrow D(x))\}$

- **Règles de transformation d'une propriété :** Les propriétés OWL donnent la capacité d'exprimer des faits au sujet de leurs classes et de leurs instances. Par exemple la couleur d'une voiture est une propriété d'une classe Voiture. OWL fait la distinction entre deux types de propriétés :
  - ✓ Les propriétés d'objet, qui permettent de relier des instances à d'autres instances
  - ✓ Les propriétés de type de donnée, qui permettent de relier des individus à des valeurs de données.

  Une propriété d'objet est une instance de la classe **owl :ObjectProperty**, une propriété de type de données est une instance de la classe **owl :DatatypeProperty**.

  Ces deux classes sont elles-mêmes sous-classes de la classe RDF **rdf :Property** [Lac, 06]. Dans forme de graphes conceptuels une propriété est une relation présentée par la balise <RelationTypes>.

  Nous définissons la règle $\mathbf{R_{relation}}$ qui transforme les propriétés OWL vers la forme de graphes conceptuels, comme suit :

Régle$_{relation}$:		
$R \leq S$	← R rdfs:subPropertyOf S	$\{ \forall x \forall y(R(x; y) \rightarrow S(x; y))\}$
$\sigma(R) = (C; –)$	← R rdfs:domain C	$\{\forall x \forall y(R(x; y) \rightarrow C(x))\}$
$\sigma(R) = (–; D)$	← R rdfs:range D	$\{\forall x \forall y(R(x; y) \rightarrow D(y))\}$
R binary relation type	← R rdf:type rdf:Property	$\{R$ binary predicate$\}$

- **Individus :** les individus sont des instances de classes, les propriétés peuvent relier un individu à un autre. OWL fournit trois constructeurs concernant l'identité des individus [Tra, 04]:
  - ✓ ***owl:sameAs*** est utilisé pour énoncer que deux références se rapportent à un même individu. Le constructeur ***owl:sameIndividualAs*** est un synonyme d'***owl:sameAs***.
  - ✓ ***owl:differentFrom*** est utilisé pour énoncer que deux références se rapportent à des individus différents.
  - ✓ ***owl:AllDifferent*** est utilisé pour énoncer qu'une liste d'individus sont tous différents.

Dans la forme de graphes conceptuels les individus sont représentés par la balise <Individus>.

La figure 32, montre la relation de sous classe entre quelques constructeurs OWL et CoGXML.

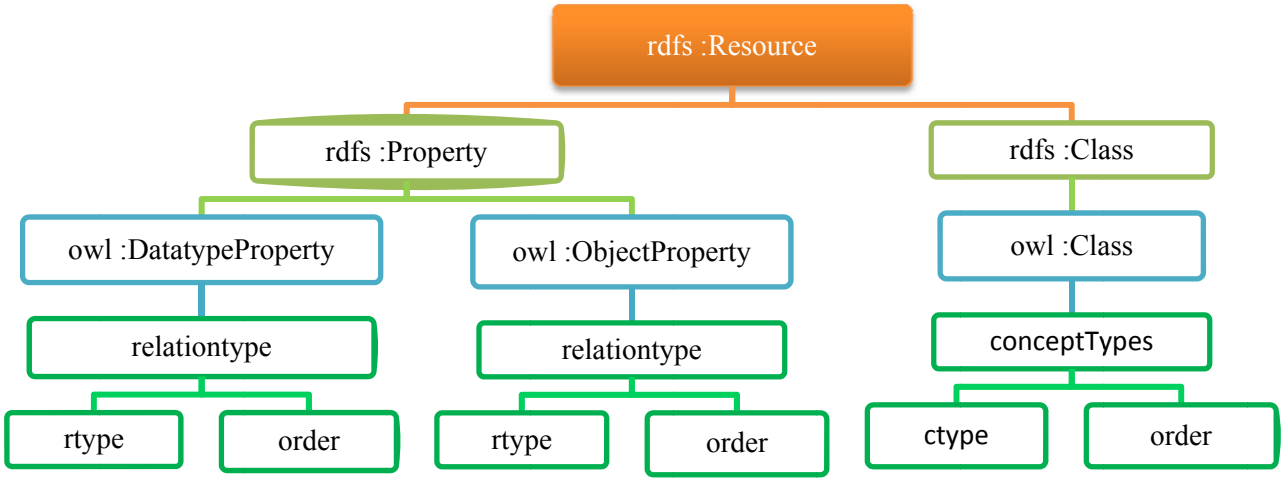

Figure 32. La hiérarchie des éléments de graphes conceptuels et RDF/RDFS

3.2.2. Schema général de l'algorithme de transformation

La figure 33 présente l'algorithme de transformation d'une ontologie OWL en CoGXML que nous proposons suivant : les règles Régle$_{classe}$, Régle$_{relation}$ et la transformation des triplets ontologiques RDF/RDFs en forme des éléments de graphes conceptuels.

```
DébutAlgo
{
Bc Kb ; /*déclaration de la base de connaissance*/
support support;
Tc hiérarchie de type concepts; Tr hiérarchie de type relations ;
I ensemble d'individus ;
Gr GCs ; /*graphe de règle */
Gf GCs ; /*graphe de faits*/
F fichierOWL ;
R fichierGC-XML ;
Pntr pointeur de balise de fichier owl ;
ImporterfichierOWL (F) ;
Parcourir (F) ;
Ouvrir (R)
/*création de la hiérarchie types concepts et relation*/
n :=1 ; m := 1 ;
R.Tc.n.m :=  rdfs :  resources ;  /*n=1  et  m=1  alors  Tc.1.1
représente le top de la hiérarchie*/
R.Tr.n.m :=rdf.Proprty ; /*n=1 et m=1 alors Tc.1.1 représente le
top de la hiérarchie*/
     Tantque non findefichier (F)
     Début
          Si (Pntr == <owl :class> alors
               n++ ; /*Création du 2eme niveau via subsomption*/
               R.Tc.n.m:=rdf.ID ;
               Pntr++;
               Si (Pntr == <owl subclass> et rdf : ressource
               <>vide) alors
                    m++;
                    R.Tc.n.m := rdf:ID.rdf :resource;
               Sinon Pntr ++;
                    Si Pntr == <owl:Restraction> alors
                         Pntr++ ;
                         R.Tr := <owl.proprty> ;
                    FSi
               FSi
          FSi
          n=1 ; m=1 ;
          j=1 ;
          I tableau des individus ;
          /*graphes de règles/
          Si (Pntr == <owl:ObjectProperty>) alors
               Pntr++ ;
               R.Tr := < rdf:ID> ; /*relation*/
               Si (<rdf:type> <> vide) alors
                    R.Gr := rdf:resource ;
               FSi
          FSi
     Fintanque.
     /*graphe de faits*/
```

```
      Tantque nonfinde R.Tr alors
      Relier chaque relation avec ces concepts ;
      Concaténation des concepts ;
      Gf ←relie chaque relation avec ses concepts,
      FinTantque
      /*création des individus*/
      Tantque nonfinde R.TC
            Si Pntr == <R.Tc.n.m> alors
                  I(j) := les sous balises de Pntr ; j++ ;
            FSi
      FinTantque
}
FinAlgo.
```

Figure 33. Algorithme de transformation OWL/ CoGXML

4. Utilisation des ontologies opérationnelles pour raisonner

Dans cette section nous présentons la structure que nous proposons pour la mise en œuvre d'un mécanisme d'inférence déductif pour la base de connaissances (Cf. figure 34). Ce mécanisme d'inférence basé sur des ontologies opérationnelles, à partir de faits initiaux, de graphes de règles et de graphes de confirmations.

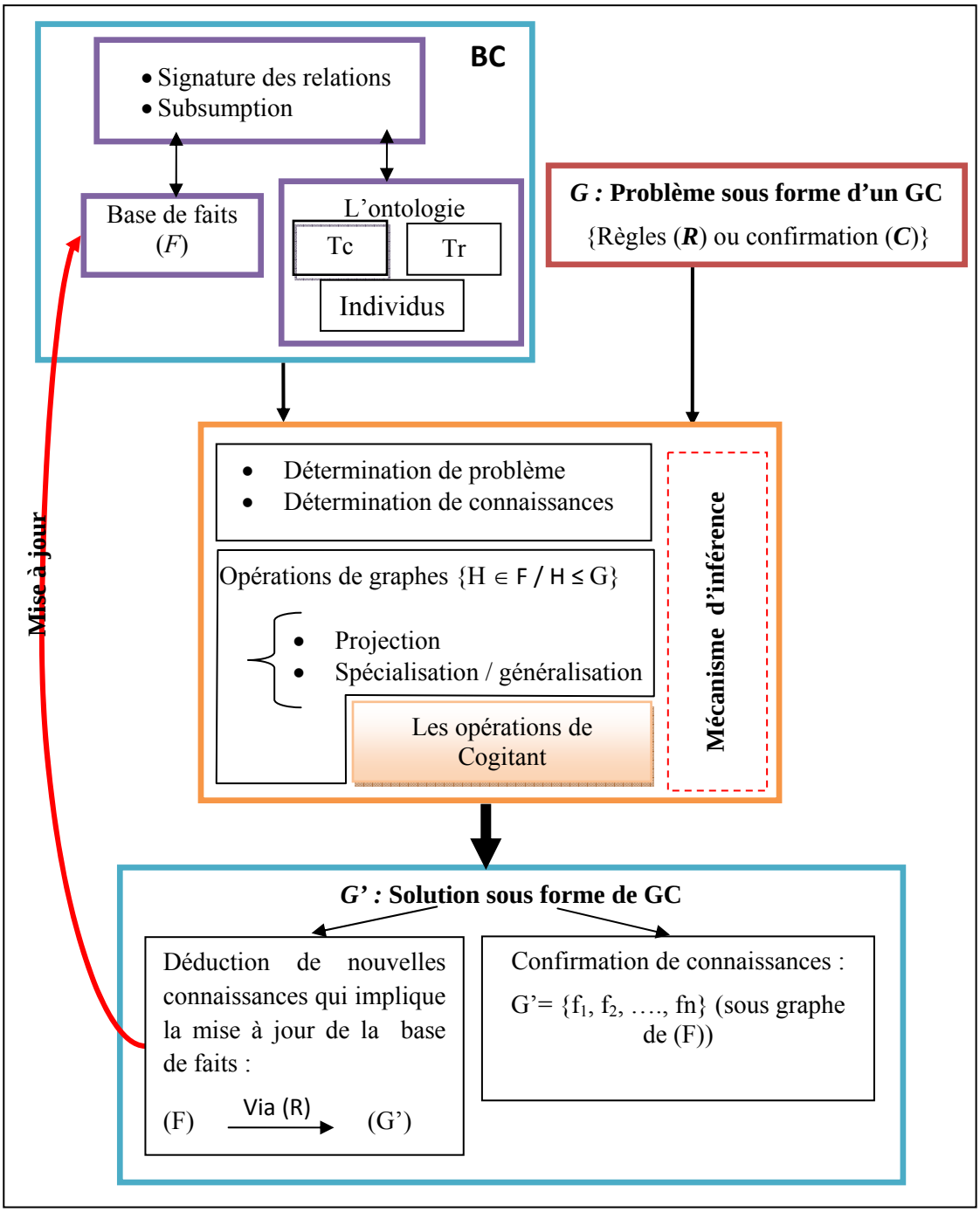

Figure 34. L'inférence à base des ontologies

4.1. Problème sous forme d'un graphe conceptuel

Le problème est représenté sous forme d'un ensemble de graphes conceptuels. Étant donné un ensemble de faits *F*, il s'agit de trouver les réponses à une question *G*, les faits et la question étant représentés par des graphes conceptuels.

Soit : $F = \{G_1, \ldots, G_n\}$ un ensemble de faits et *G* un problème sous forme de graphes conceptuels. Le problème est de trouver les réponses à *G* dans *F*. Deux cas sont possibles:

1) Un ensemble de graphe de faits ***F*** (un graphe de faits ou plusieurs) et un graphe de confirmation ***C*** auquel on cherche une solution dans ***F***. Ce type de requête sert à la confirmation de connaissances.

   La réponse d'une requête de confirmation est un élément de la base de connaissances qui implique la validation d'une connaissance ou à la prise de décision (confirmation de connaissances).

2) Un ensemble de graphes de faits ***F*** (un graphe de faits ou plusieurs) et un graphe de règles ***R*** auquel on cherche une solution dans ***F***. ce type de problème sert à inférer de nouvelles connaissances.

   La réponse d'une règle est un élément de la base de connaissances qui implique la règle ; c'est une réponse qui permet d'inférer de nouvelles connaissances.

La solution est le résultat de la projection de ***F*** et (***R*** ou ***C*)**, sous forme d'un graphe conceptuel.

4.1.1. La règle (R) sous forme d'un graphe conceptuel

L'ensemble de règles correspond aux règles logiques sous forme de graphes conceptuels, ce qui donne à la base de connaissances l'empreinte de raisonnement. Les graphes de règles n'ont pas une forme prédéfinie, une règle est une connaissance qui organise quelques faits entre eux, regroupés en un ensemble de faits regroupés en un ensemble appelé graphe de prémisses ou graphe d'hypothèses, et le restant est appelé graphe déduction ou graphe conclusion.

Par exemple, la règle : SI Rayan est père-de Maria et Ali est père-de Rayan alors Ali est le Grand-Père-De Maria. Cette règle est représentée par un graphe conceptuel comme illustré dans la figure 35 :

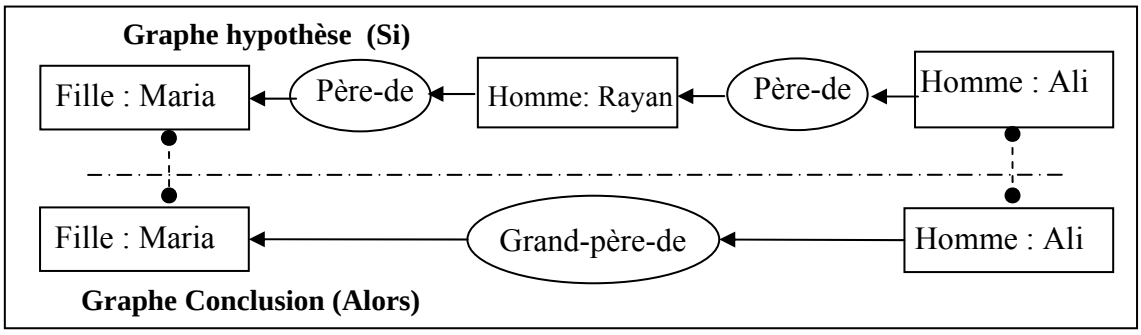

Figure 35. Exemple d'un graphe de règle

Les liaisons entre les concepts de la partie hypothèse et de la partie conclusion se fait à l'aide des liens appelés liens de coréférence. L'utilisation des règles se fait pour valider des connaissances ou pour inférer d'autres nouvelles.

4.1.2. La confirmation (C) sous forme d'un graphe conceptuel

Les graphes de confirmations permettent la manipulation des connaissances dans l'objectif de la résolution d'un problème. Ces graphes permettent donc, d'interroger la base de connaissances à l'aide des graphes de faits.

Un graphe de confirmation est un graphe conceptuel dont les éléments sont les éléments du support. La figure 36 est une représentation graphique de la confirmation « Tous les ancêtres-de la fille Maria ».

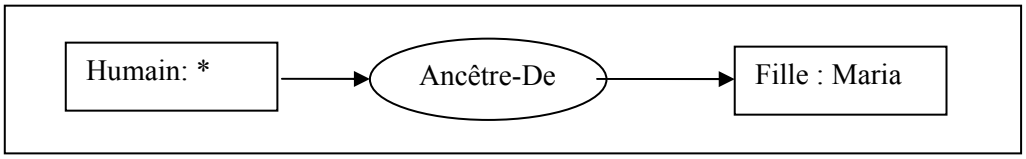

Figure 36. Exemple d'un graphe de confirmation

4.2. Mécanisme d'inférence

Dans cette section nous présentons le mécanisme d'inférence que nous proposons qui permet la résolution des problèmes sous forme de graphes conceptuels. Le but de notre processus opérationnel est d'inférer de nouvelles connaissances ou de confirmer (ou extraire) d'autres sous forme de graphes conceptuels qui allouera la possibilité de les manipuler par des diverses syntaxes en l'occurrence la syntaxe graphique, facilitant de ce fait la spécification des connaissances et des préférences de l'utilisateur dans les processus de validation et de raisonnement.

A la réception d'un problème, le système opérationnel pourra déterminer s'il s'agit d'une requête de confirmation ou de règle. Par la suite, il détermine s'il s'agit d'un concept ou d'une relation, en utilisant les informations du vocabulaire, et grâce à sa définition lambda, le système pourra déterminer sa valence ; même les signatures pourront être définies et d'autres informations pourront être vérifiées au fur et à mesure qu'elles parviennent au système. Nous citerons comme exemple :

- La compatibilité des types de concepts apparents dans le graphe avec les individus qu'ils typent et cela grâce aux informations du catalogue d'individus et des algorithmes de calcul de compatibilité de types ;
- La hiérarchie de concepts, nous permet de déduire les types qui sont des spécialisations d'autres types de concept et conformes à la signature de la relation validant de ce fait le graphe.

Une fois le système reçoit le problème le mécanisme commence par effectuer la projection entre le problème et la base de faits. La projection permet de déterminer si un graphe (graphe de problème) est plus spécialisé, ou plus généralisé qu'un autre (graphe de faits), deux cas sont possibles :

1) La projection d'un graphe conceptuel qui représente une requête de confirmation $C= (R_c, C_c, U_c, étiqc_q)$ dans un graphe conceptuel qui représente des faits $F = (R_f, C_f, U_f, étiq_f)$ est un couple d'applications $\Pi=(f,g), f : R_C \rightarrow R_F, g : C_C \rightarrow C_F,$ qui :

   - Conserve les arêtes et la numérotation des arêtes : pour toute arête *rc* de U_C, $f(r)g(r)$ est une arête de U_F. De plus, si $c = C_i(r)$, alors $g(r) = Fi\ (f(r))$.
   - Peut restreindre les étiquettes des sommets : pour tout sommet *r* de R_C, $étiq_F(f(r)) \leq étiq_C(r)$, pour tout sommet *c* de C_C, $étiq_F(g(r)) \leq étiq_C(r)$.

   Autrement dit :

   - Pour tout sommet relation de *F*, son type est plus général que celui de son image par f_r ;
   - Pour tout sommet concept de *F*, son type est plus général que celui de son image par f_c et, soit son identifiant est *, soit son identifiant est celui de son image par f_c ;
   - Pour tout sommet relation r de *F* d'arité n, pour tout i de 1 à n, l'image par f_c du sommet concept numéro i lié à r est le sommet concept numéro i lié au sommet relation image par f_r de r.

2) La projection d'un graphe conceptuel qui représente une règle $R= (R_r, C_r, U_r, étiq_r)$ dans un graphe conceptuel qui représente des faits $F = (R_f, C_f, U_f, étiq_f)$ est un couple d'applications $\Pi= (f,g), f : R_R \rightarrow R_F, g : C_R \rightarrow C_F,$ qui :

   - Conserve les arêtes et la numérotation des arêtes : pour toute arête *rc* de U_R, $f(r)g(r)$ est une arête de U_F. De plus, si $c = R_i(r)$, alors $g(r) = F_i\ (f(r))$.
   - Peut restreindre les étiquettes des sommets : pour tout sommet *r* de R_R, $étiq_F(f(r)) \leq étiq_R(r)$, pour tout sommet *c* de C_R, $étiq_F(g(r)) \leq étiq_R(r)$.

   Autrement dit :

   - Pour tout sommet relation de *F*, son type est plus général que celui de son image par f_r ;
   - Pour tout sommet concept de *F*, son type est plus général que celui de son image par f_c et, soit son identifiant est *, soit son identifiant est celui de son image par f_c ;

- Pour tout sommet relation r de F d'arité n, pour tout i de 1 à n, l'image par f_c du sommet concept numéro i lié à r est le sommet concept numéro i lié au sommet relation image par f_r de r.

Notre approche se base sur la bibliothèque **Cogitant** qui permet de représenter des connaissances et raisonner avec des graphes conceptuels. Cogitant utilise le modèle de graphes conceptuels comme un modèle formel disposant des opérations de raisonnement internes au modèle, et conservant ainsi les propriétés de consistance et complétude vis à vis de la logique des prédicats du premier ordre. À chaque objet du modèle correspond une classe du programme, et les opérations sur les graphes sont implémentées, en particulier la projection et l'application d'une règle sous forme d'un graphe conceptuel sur un graphe.

4.3. Solution sous forme d'un graphe conceptuel

Soit : $F = \{G_1, ..., G_n\}$ un ensemble de faits et G un problème sous forme de graphes conceptuels. La solution est le résultat de la projection de ***F*** et (***R*** ou ***C***), sous forme d'un graphe conceptuel, deux cas sont possibles:

4.3.1. Déduction de nouvelles connaissances à partir d'une règle

L'implémentation des règles d'inférence, se fait par la spécification de la sémantique et du rôle des différents composants dans le processus de déduction et cela comme étant des règles d'inférences avec un graphe hypothèse et un graphe conclusion dont la sémantique est : « ***Si l'hypothèse d'un graphe de règles est présente dans un graphe de faits, la conclusion peut être ajoutée au graphe de faits***».

L'utilisation de ce type de connaissances peut atteindre un certain degré d'automatisation déchargeant l'utilisateur au cours des processus d'opérationnalisation et de raisonnement.
Exemple : la projection de la règle (Cf. la figure 35) sur le graphe (Cf. la figure 30) permet d'ajouter la connaissance : Ali est le grand père de Maria, voir la figure 37.

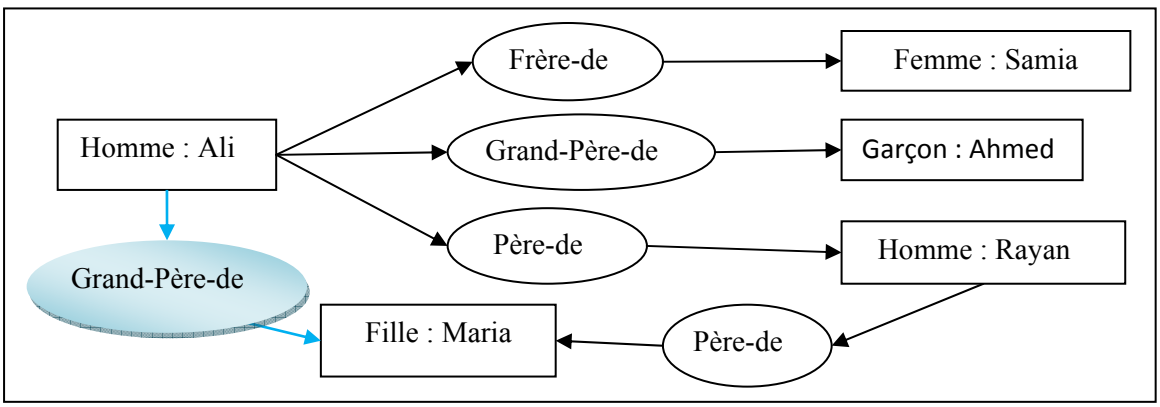

Figure 37. Exemple d'inférence

Donc, notre approche permet d'appliquer les graphes de règle *R* (une règle ou un ensemble de règles) sur un graphe de faits *F* qui représente un fait de domaine (ou un ensemble graphes de faits) cette opération se fait à l'aide des projections (Cf. chapitre. I). Cette opération permet d'inférer des nouvelles connaissances ou de raisonner sur des faits existants.

Dans le cas de plus d'une règle la sélection des règles se fait par le calcule de la fermeture d'un graphe par un ensemble de règles. La fermeture des graphes est supporté par Cogitant à l'aide de la class **`cogitant::Environment::rulesClosure`** [Gen, 10].

La figure 38 présente l'algorithme de projection d'une règle (R) sur un graphe de faits (F).

1. Début
2. Sélectionner la règle d'inférence à projeter R ∈ {GR} (si hypothèse alors conclusion);
3. Sélectionner un graphe de fait F ∈ {GF}, sur le quel on projette R ;
4. Identifier les ressemblances dans R, entre les concepts C de la partie hypothèse et la partie conclusion ;
5. Identifier les ressemblances de concept et relation entre le graphe de règles R (hypothèse) et le graphe de faits F ;
6. Si il y'a une ressemblance aller à 7 sinon arrêter et aller à 11.
7. Ajouter la relation (les relations) qui se trouve dans la partie conclusion de graphe de règles R au graphe de faits F ;
8. Ajouter le concept (les concepts) qui se trouve dans la partie conclusion de graphe de règles R et qui relie les relations de l'étape 4 au graphe de faits F ;
9. Relier les relations de l'étape 4 avec les concepts de graphe de faits F.
10. Répéter depuis 2.
11. Fin.

Figure 38. Algorithme de projection d'une règle (R) sur un graphe de fait (F)

Où {GF} est l'ensemble de graphes de faits et {GR} est l'ensemble de graphes de règles.

La figure 39, montre graphiquement la projection d'une règle suivant l'algorithme de projection d'une règle (R) sur un graphe de fait (F).

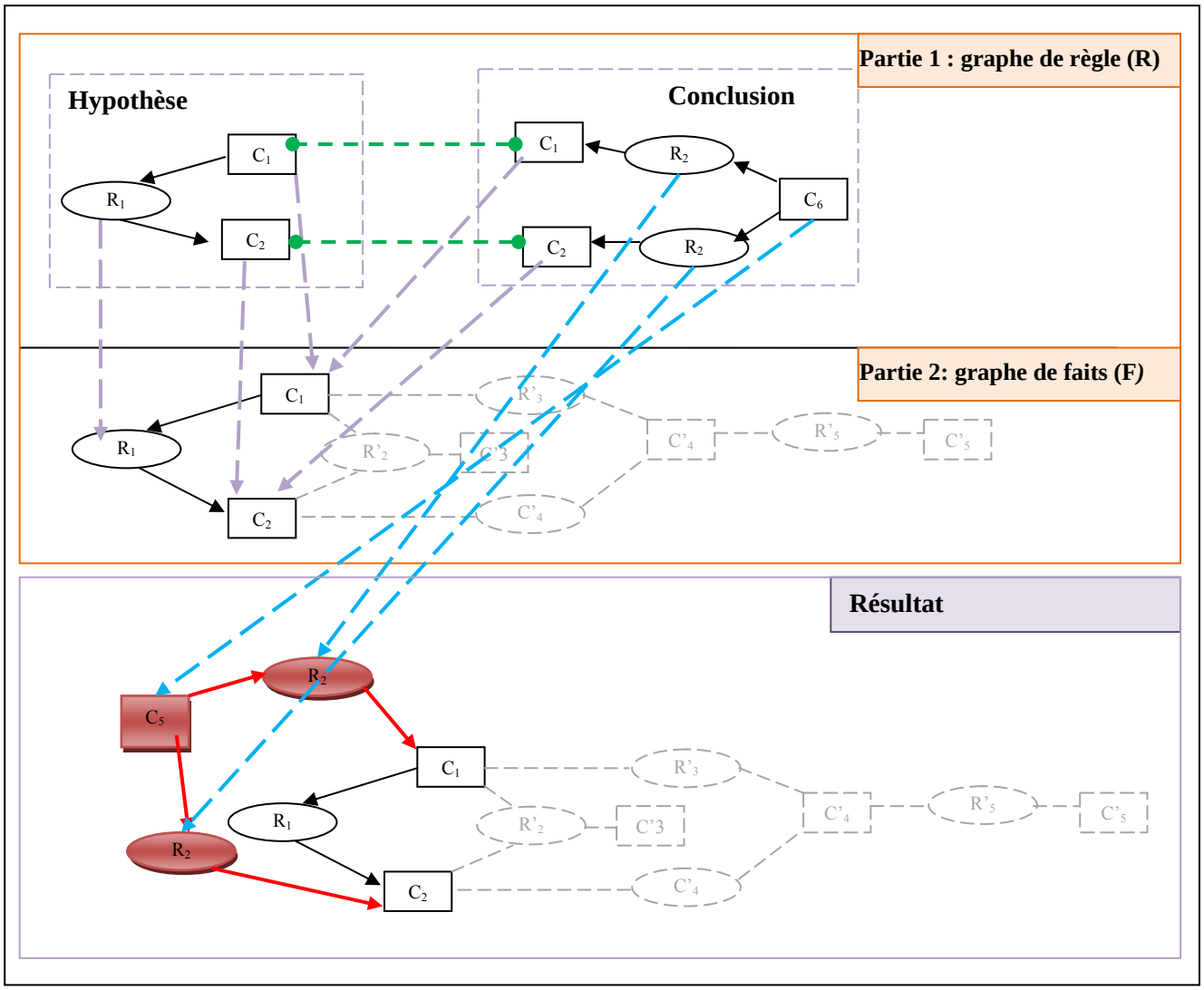

Figure 39. Application d'une règle (R) sur un graphe de faits (F)

4.3.2. Confirmation de connaissances

L'implémentation des requêtes pour la confirmation ou l'extraction de connaissances, se fait par la spécification de la sémantique et du rôle des différents composants dans le processus de déduction et cela comme étant des requêtes qui aident à la confirmation de connaissances avec un graphe dont la sémantique est : « ***Si le graphe de requête engendre une (des) partie(s) du graphe de faits, l'extraction de connaissances est l'ensemble des graphes engendrés par la requête***».

Exemple : la projection de la requête (Cf. la figure 36) sur le graphe de faits (Cf. la figure 30) permet d'extraire la connaissance : Ali est le grand père de Maria et Rayan est son père, voir la figure 40.

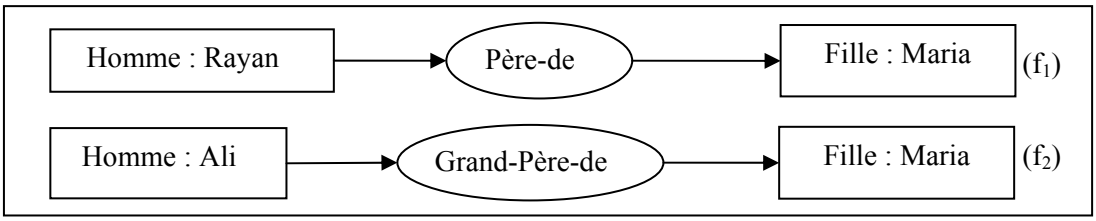

Figure 40. Exemple d'inférence

Notre approche permet d'appliquer les graphes de confirmation *C* (une requête ou un ensemble de requêtes) sur un graphe de faits *F* qui représente un fait de domaine (ou un ensemble de graphes de faits) cette opération se fait à l'aide des projections graphiques. Cette opération permet de confirmer ou d'extraire des informations du monde réel sous forme d'un graphe de faits existant à l'aide d'une requête.

La figure 41 présente l'algorithme de projection d'une requête de confirmation (C) sur un graphe de faits (F).

1. Début
2. Sélectionner la requête C ∈ {GC} à projeter ;
3. Sélectionner un graphe de fait F ∈ {GF} sur le quel on projette C ;
4. Identifier les ressemblances de concept et relation entre le graphe de requête C et le graphe de faits F pour cela :
   a. Si (le type de concept et le référent) de C sont identique avec celui de F alors
      Alors aller à b ;
      Sinon Si le référent de la requête est plus générique que celui de graphe de faits Alors aller à b ;
      Sinon arrêter et aller à 6 ;
   b. Si le type de relation de la requête est identique à celui de graphe de faits alors aller à 5 ;
      Sinon Si type de relation de la requête est plus générique que celui de graphe de fait alors aller à 5 ;
      Sinon arrêter et aller à 6 ;
5. Le résultat sera l'extraction d'une partie de graphe de fait, qui conforme la requête.
6. Répéter depuis 4.
7. Fin.

Figure 41. Algorithme de projection d'une requête (C) sur un graphe de faits (F)

Où {GF} est l'ensemble de graphes de faits et {GC} est l'ensemble de graphes de confirmation.

La figure 42, montre graphiquement la projection d'une requête de confirmation suivant l'algorithme de projection d'une requête de confirmation (C) sur un graphe de faits (F).

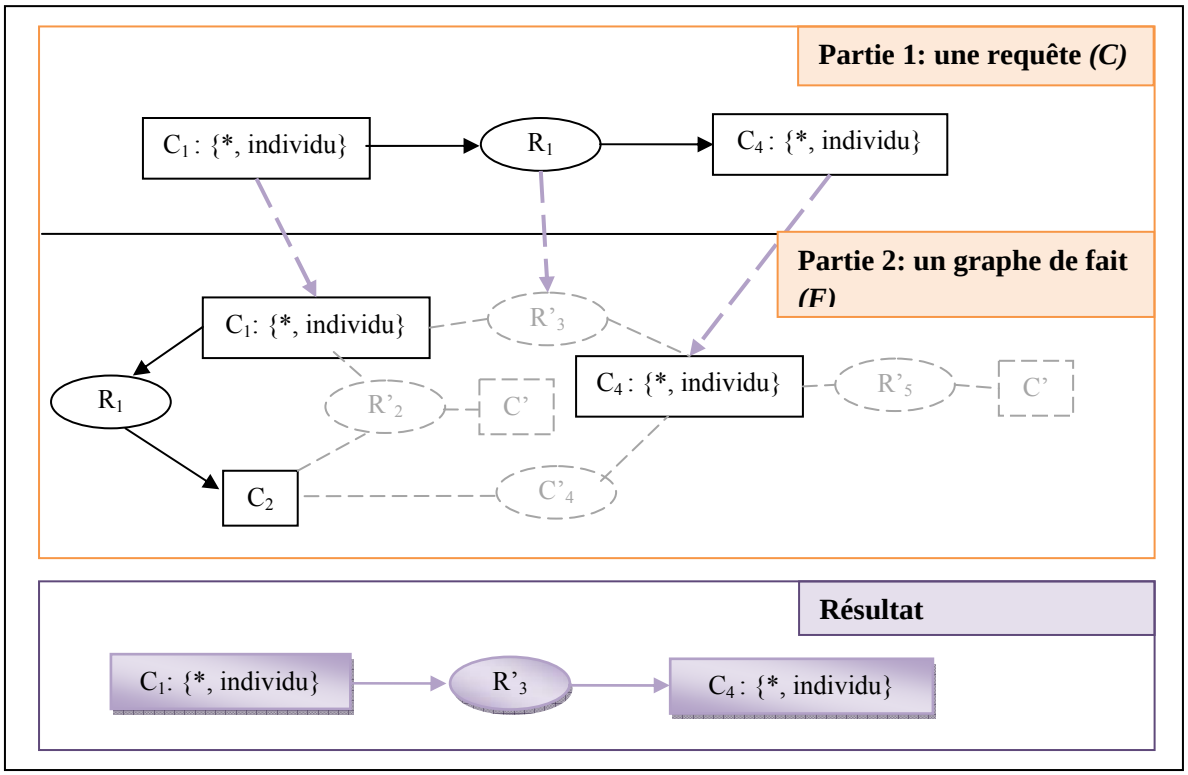

Figure 42. Application d'une requête (C) sur un graphe de faits (F)

5. Portabilité et connexion avec les outils existants

Notre approche permet la connexion avec d'autres outils existants qui permettent la création des ontologies (p. ex. l'outil Protégé-2000), grâce à l'algorithme de transformation d'ontologies (Cf. §. 2.4).

En-outre, le fichier de sortie CoGXML est bâti sous des normes universelles (syntaxe RDF/XML), ce qui permet à notre approche d'interagir avec n'importe quels outils bâtis sous ces normes (p. ex. importation ou exportation des fichiers sous format RDF ou OWL).

L'utilisation de la bibliothèque open-source **Cogitant** permet d'interagir avec d'autres outils qui utilisent cette bibliothèque.

Des recherches actuelles sont menées pour ajouter à OWL et à RDFS des représentations de règles p. ex. [Pan, 04]. Ce qui permettra à CoGXML la sauvegarde et le chargement d'ontologie à partir d'un formalisme standard, ce qui accroîtra son interopérabilité avec les autres éditeurs d'ontologie.

6. Implémentation

Pour montrer la faisabilité de notre approche d'opérationnalisation d'ontologies par les graphes conceptuels, un prototype d'édition et d'opérationnalisation d'ontologies a été mis en œuvre.

Dans cette section nous allons donner une idée de l'environnement de mise en œuvre du prototype et une description du prototype.

6.1. Choix technique

Pour l'implémentation de prototype, nous avons dû faire un choix concernant les langages de programmation utilisés pour l'implémentation des applications Web (JAVA, PHP, C++…) et les langages du Web sémantique (RDF, RDFS, et OWL).

- Nous avons utilisé le langage de programmation java pour développer notre éditeur d'ontologies opérationnelles, vu qu'il est portable et multiplateforme et possède une riche bibliothèque de classes. Nous avons réalisé notre application java en utilisant JDK 1.6, avec l'éditeur Eclipse.

- Nous avons utilisé la bibliothèque **Cogitant** (librairie de classes C++), qui permet la création des applications basées sur le modèle des graphes conceptuels*,* cette bibliothèque comporte les graphes emboités ainsi les liens de coréférences [Gen, 10]..

  Les principales fonctions offertes par la bibliothèque sont :

  - ✓ Gestion de la mémoire de graphes conceptuels.
  - ✓ Manipulation du support dans la mémoire (plusieurs supports peuvent coexister dans la mémoire, mais un graphe conceptuel est défini sur un seul support).
  - ✓ La manipulation des règles des graphes conceptuels, et les opérations portant sur les règles.

  La classe Java (partly) : permet aux applications développées en Java d'accéder aux fonctions Cogitant. On outre, il est possible d'accéder aux fonctions de Cogitant, à distance grâce à l'architecture client-serveur qui se base sur l'échange de messages sous format XML.

- Dans notre projet CMAKE permet de compiler la bibliothèque Cogitant sur laquelle nous nous basons dans le développement de notre prototype. CMAKE permet de rendre Cogitant une bibliothèque portable et dont le code source respecte les règles de portabilité.

6.2. Description du prototype

Notre prototype se compose de plusieurs fenêtres, la figure 43 présente la page d'accueil qui contient :

1. Une barre de menu (Fichier, Edition, Raisonnement, Contrôle et Aide)
2. Volet d'onglets (Vocabulaire et Graphes)

3. Menu principal dans lequel nous observons notre travail (création de vocabulaire, les requêtes, les assertions, etc.)
4. Afficheur des erreurs : permet d'afficher les erreurs de la construction.

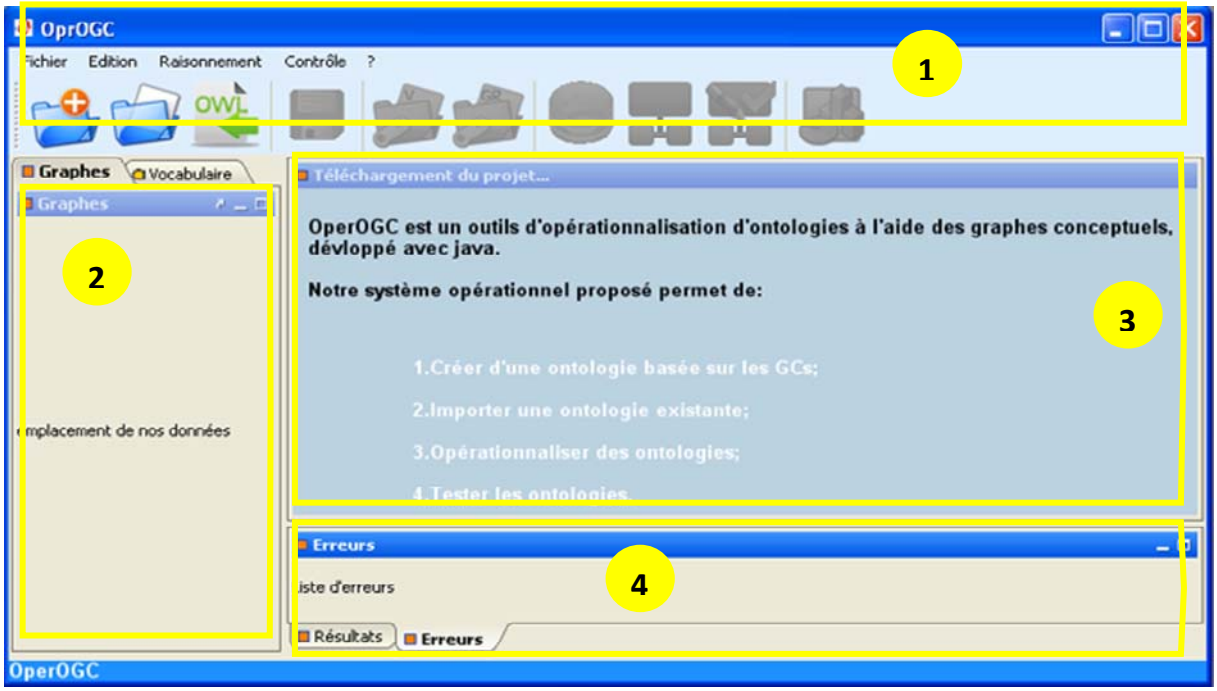

Figure 43. Page d'accueil d'OperOGC

7. Conclusion

Dans ce chapitre nous avons proposé une approche de construction d'ontologies opérationnelles, en passant par une étape de conceptualisation, d'ontologisation, d'opérationnalisation. Cette étape est suivie d'une étape de codification dans laquelle nous avons représenté l'ontologie opérationnelle sous un langage d'ontologie basé sur les graphes conceptuels et la syntaxe XML (CoGXML). Nous avons aussi proposé un algorithme de transformation d'ontologies OWL en CoGXML.

Un moteur d'inférence a été ajouté, permettant de mettre en œuvre les ontologies opérationnelles pour raisonner sur des bases de connaissances. Ce moteur d'inférence utilise le formalisme des graphes conceptuels, qui constitue le langage opérationnel cible, et les bases de connaissances sont constituées par des ensembles de graphes conceptuels. Basé sur la plateforme Cogitant, ce moteur offre toutes les fonctionnalités requises pour raisonner avec des graphes conceptuels, des règles graphe conceptuel et des confirmations graphe conceptuel.

A la fin, un prototype d'édition et d'opérationnalisation d'ontologies a été mis en œuvre, en se basant sur l'approche proposée et en utilisant le langage de programmation java.

Conclusion & perspectives

Ce travail nous a permis d'étudier l'opérationnalisation des ontologies par les graphes conceptuels, nous avons commencé par un état de l'art sur : l'ingénierie des ontologies et la représentation des connaissances dans le quel nous avons étudié les différentes définitions qui portent sur la notion d'ontologie, les langages utilisés pour leurs implémentations, ainsi que quelques méthodologies de construction des ontologies.

Nous avons ensuite montré les limites des ontologies en termes de raisonnement et capacité d'inférence ainsi que la nécessité de faire supporter l'ontologie par un formalisme de représentation de connaissances. Pour atteindre cet objectif, et nous avons choisi le formalisme des graphes conceptuels dans le cadre de notre travail.

Les avantages qu'apportent les graphes conceptuels dans le traitement des ontologies se résument essentiellement en des avantages structurels et représentatifs à savoir la possibilité de représenter des connaissances de types plus variés et cela d'une façon plus simple. Du point de vue de l'inférence, nous avons également vu que certains modes de raisonnement étaient applicables sur des systèmes à base de connaissances basés sur les graphes conceptuels.

Nous avons proposé une approche guidée pour la création d'ontologies opérationnelles par l'utilisation des graphes conceptuels comme modèle de représentation de connaissances. Cette approche regroupe trois phases : (1) *conceptualisation* qui permet de définir l'ontologie, (2) *ontologisation* qui sert à formaliser -autant que possible- le modèle conceptuel obtenu à l'étape précédente et (3) *opérationnalisation* qui consiste à formaliser complètement l'ontologie conceptuelle obtenue précédemment en la codant dans le modèle des graphes conceptuels.

Une fois l'ontologie opérationnelle est définie, celle-ci devient une base de connaissances dont les concepts de l'ontologie opérationnelle sont organisés en deux parties :

- Vocabulaire : hiérarchie de types concept, hiérarchie de types relation et les individus.
- Base de faits : graphe de faits dont la création se base sur le vocabulaire.

Nous avons proposé un algorithme qui permet de transformer les ontologies OWL vers CoGXML. Pour se faire, nous avons défini des règles qui permettent d'exprimer les éléments d'une ontologie OWL suivant de nouvelles classes qui expriment les éléments de graphes conceptuels. Ces règles permettent la création des triplets ontologiques par la correspondance de types concept et types relation avec les éléments RDFs.

Nous avons présenté le format BCGCT (Base de Connaissances Graphes Conceptuels Textuelle) fourni par Cogitant qui permet la représentation du support, des graphes et des règles. Ce format codifie l'ontologie à l'aide d'un fichier CoGXML basé sur la syntaxe XML et une DTD fournit par Cogitant.

Nous avons ensuite proposé une approche permettant de raisonner sur l'ontologie opérationnelle, celle-ci peut être vue comme une base de connaissances à laquelle nous ajoutons un mécanisme d'inférence permettant un raisonnement déductif. Le mécanisme de raisonnement que nous proposons se base sur le principe de projection des graphes conceptuels. Deux stratégies sont possibles:

- La confirmation de connaissances, cette stratégie part des données disponibles, et tente d'arriver vers le but : grâce à la projection de graphe de confirmation sur un graphe de faits qui permet d'extraire ou de juger l'existence d'une connaissance ou de la nier;
- La déduction de nouvelles connaissances à l'aide de règles : grâce à la projection de graphes de règles sur un graphe de faits qui permet d'inférer de nouvelles connaissances.

Nous avons validé notre approche par le prototype **OperOGC** *« OPERationnalisation d'Ontologies par les Graphes Conceptuels »*, qui est fondé sur la bibliothèque des graphes conceptuels Cogitant. OperOGC permet de créer des ontologies opérationnelles et d'importer des ontologies OWL existantes. L'ontologie opérationnelle devient une base de connaissances sur laquelle OperOGC va raisonner.

Notre travail permet de dresser de nombreuses perspectives. Dans cette section, nous présentons celles qui nous paraissent être les plus intéressantes.

- L'utilisation de l'approche pour des cas réels dans le cadre de projet de l'équipe notamment de raisonner sur des ontologies de domaine en langue Arabe.
- Nous avons implémenté dans ce travail un mécanisme de raisonnement déductif. Dans la suite de ce travail, nous nous intéresserons à d'autres types de raisonnement comme le raisonnement abductif et le raisonnement à base de cas.

Références Bibliographique

[Ant, 04] G. Antoniou. 'A Semantic Web Primer, MIT Press, Cambridge', MA, Van Harmelen, 2004.

[Arp, 03] J. C. Arpirez, O. Corcho, M. Fernandez-Lopez, A. Gomez- Perez. *WebODE in a nutshell. AI Magazine'*, 2003.

[Aus, 00] Aussenac-Gilles. N. Biébow. B. Szulman. **'**Revisiting Ontology Design: a method based on corpus analysis'. Dans: R. Dieng, O. Corby (Ed.), 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00). Pages: 172-188. Berlin, Germany: Springer-Verlag. 2000.

[Baa, 91] F. Baader, D.Mcguinness, D. Nardi, P. Patel-schneider. 'The Description Logic Handbook: Theory, Implementation and Applications'. Cambridge University Press. 1991.

[Bac, 00] B. Bachimont. 'Engagement sémantique et engagement ontologique : conception et réalisation d'ontologies en ingénierie des connaissances'. Dans : Ingénierie des connaissances. Évolution Récentes et nouveaux défis, J. Charlet. M. Zacklad. G. Kassel et D. Bourgault. Pages : 305-323. Paris : Eyrolles. 2000.

[Bac, 01] B. Bachimont. 'Modélisation linguistique et modélisation logique des ontologies : l'apport de l'ontologie formelle'. Dans : Actes des journées francophones d'Ingénierie des Connaissances (IC'2001). Presse Universitaire Grenobloise, 2001.

[Bac, 08] B. Bachimont, N. Nadah, A. Baneyx. 'Categorial Ontologies: Formalizing the Differential Ontology'. Journées Francophones sur les Ontologies **(**JFO'2008). Lyon, France Copyright 2008 ACM 978-1-60558-373-0/08. 2008.

[Bag, 99] J.F. Baget, D. GenesT, M.L. Mugnier. 'Knowledge Acquisition with a Pure Graph-Based Knowledge Representation Model. Application to the Sisyphus-I Case Study'. Dans: Proceedings de Knowledge Acquisition Workshop (KAW'1999), volume: 954. Springer-Verlag LNAI. 1999.

[Bag, 02] J.F. Baget, M.L. Mugnier. 'Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints'. Journal of Artificial Intelligence Research. Pages:425–465, 2002.

[Ban, 07] A. Baneyx. 'Construire Une Ontologie De La Pneumologie : Aspects Théoriques, Modèles Et Expérimentations'. Thèse de doctorat, Université Pierre Et Marie Curie. 2007.

[Bar, 97] J.P. Barthès. 'Capitalisation des connaissances et intelligence artificielle'. Journées Franco-Finlandaises de Tampere, Compiègne. 1997.

[Bec, 01] S. Bechhofer, I. Horrocks, C. Goble. R. Stevens. 'OilEd: a Reason-able Ontology Editor for the Semantic Web'. Dans: Proceedings of the Joint German/Austria Conference on Artificial Intelligence (KI'2001), volume: 2174. Pages: 396–408. Springer-Verlag LNAI. 2001.

[Bla, 98] M. Blazquez, M. Fernandez, J. Garcia-Pinar, A. Gomez-Perez. 'Building Ontologies at the Knowledge Level using the Ontology Design Environment'. Dans: Proceedings of the Banff Workshop on Knwoledge Acquisition for Knwoledge-based Systems, 1998.

[Ber, 96] A. Bernaras, I. Laresgoiti, J. Corera. 'Building and Reusing Ontologies for Electrical Network Applications'. Dans: Proceedings Of the 12th ECAI96. Pages: 298-302. 1996.

[Bor, 97] W.N. Borst. 'Construction of Engineering Ontologies'. Center for Telematica and Information Technology, University of Tweenty, Enschede, NL. 1997.

[Bou, 95] J. Bouaud, B. Bachimont, J. Charlet, P. Zweigenbaum. 'Methodological Principles for Structuring an Ontology'. Dans : Proceedings of IJCAI'95 Workshop: Basic Ontological Issues in Knowledge sharing. ACM Press, 1995.

[Bou, 11] F. Bourai, H. Aliane, Z. Alimazighi. 'Opérationnalisation d'ontologies par les graphes conceptuels'. Dans : 4èmes Journées Francophones sur les Ontologies, Montréal, Canada. Pages : 175-187. 2011.

[Bra, 79] R. Brachman. 'On the Epistemological Status of Semantic Networks'. Associative Networks: Representation and Use of Knowledge by Computers. Academic Press. Pages: 3-50. 1979.

[**Bra, 91**] R.J. Brachman, D.L. Mcguinness, P.F. Patel-schneider, L.A. Resnik, A. Borgida. 'Living with CLASSIC: When and How to Use a KL-ONE-Like Language. Principles of Semantic Networks'. Dans: the representation of knowledge. Pages: 401–456. 1991.

[**Bra, 04**] R.J. Brachman, H.J. Levesque. 'Knowledge representation And reasoning'. ISBN: 1-55860-932-6. 2004.

[Bri, 02] D. Brickley, R.V. Guha. 'RDF Vocabulary Description Language 1.0: RDF Schema (W3C Working Draft)'. 2002.

[Cad, 97] M. Cadoli, F.M. Donini. 'A Survey on Knowledge Compilation'. Dans: AI Communications-The European Journal for Artificial Intelligence, volume : 10. Pages: 137–150. 1997.

[Cha, 01] J. Charlet. 'L'ingénierie des connaissances : une science, un enseignement?', Dans : Actes des journées francophones d'Ingénierie des Connaissances (IC'2001). Pages 233–252. Presse Universitaire Grenobloise. 2001.

[Cha, 03] J. Charlet. 'L'ingénierie des connaissances : développements, résultats et perspectives pour la gestion des connaissances médicales'. Mémoire d'habilitation à diriger des recherches, Université Pierre et Marie Curie. 2003.

[Cor, 08] D.R. Corbett. 'Graph-Based Representation and Reasoning for Ontologies'. Computational Intelligence: A Compendium : Studies in Computational Intelligence. Volume : 115/2008, Pages : 351-379, DOI: 10.1007/978-3-540-78293-3_8. 2008.

[Cor, 00] O. Corby, R. Dieng, C. Hébert. 'A Conceptual Graph Model for W3C Resource Description Framework'. Dans: Proceedings of the 8th International Conference on Conceptual Structures (ICCS'2000), volume: 1867. Pages: 468–482. Springer-Verlag LNCS. 2000.

[Cor, 02] O. Corby, C. Faron-zucker. 'OCorese: a Corporate Semantic Web Engine'. Dans : Proceedings of the workshop on Real World, RDF and Semantic Web applications at the 11th International World Wide Web Conference (WWW'2002). 2002.

[Cor, 06] O. Corby. R. Dieng. C. Faron-Zucker. F. Gandon. A. Giboin. 'Le moteur de recherche sémantique Corese'. INRIA Sophia Antipolis, I3S Université de Nice, Sophia Antipolis. 2006.

[Dar, 02] A. Darwiche, P. Marquis. 'A Knowledge Compilation Map'. Dans : Journal of Artificial Intelligence Research, volume: 17. Pages : 229–264. 2002.

[Dib, 00] J. Dibie. 'Validation et réparation de Graphes Conceptuels'. Thèse de Doctorat, Université Paris- IX Dauphine. 2000.

[Don, 05] J. Dong, Y Feng, Y. Fang Li, J Sun. 'A Tools Environment for Developing and Reasoning about Ontologies'. Dans: Proceedings, APSEC'05. Taiwan. Pages : 465-472. 2005.

[Dre, 93] O.M. Drews. 'Raisonnement classificatoire dans une représentation à objets multipoints de vue'. Thèse de doctorat, Université Joseph Fourier, Grenoble. Pages : 30-51. 1993.

[Enc, 00] A. Michel. 'Dictionnaire de la philosophie'. Encyclopaedia. Pages : 2044. Universalis Paris : 2000.

[Fen, 00] D. Fensel, I.Horrocks, F.vanHarmelen, S. Decker, M. Erdmann, M. Klein. 'Knowledge Engineering and Knowledge Management'.Germany. Springer- Verlag. Pages: 1-16. 2000.

[Fen, 01] D. Fensel, F.van Harmelen, I. Horrocks, D.L. McGuinness. P.F. Patel-Schneider. 'OIL: An Ontology Infrastructure for the Semantic Web'. IEEE Intelligent Systems', volume: 16, n° 2. Pages: 38-45. 2001.

[Für, 01] F. Fürst, M. Leclère, F. Trichet, 'Construction d'une ontologie opérationnelle : un retour d'expérience ', Revue Extraction des Connaissances et Apprentissage (Actes des Journées Francophones Extraction et Gestion des Connaissances EGC2002), volume: 1, n° 4. Pages : 227-232. Hermès. 2001.

[Für, 02] F. Fürst. 'L'ingénierie ontologique', rapport de recherche n°02-07, Institut de Recherche en Informatique de Nantes IRIN, 2002.

[Für, 03] F. Fürst, M. Leclère, F. Trichet. 'Ontological Engineering and Mathematical Knowledge Management: a Formalization of Projective Geometry'. Annals of Mathematics and Artificial Intelligence, Kluwer Academic Publishers, volume: 38. Pages: 65–89, 2003.

[Für, 03a] F. Fürst. 'TooCoM: a Tool to Operationalize an Ontology with the Conceptual Graph Model'. Dans: Proceedings of the second Workshop on Evaluation of Ontology-Based Tools (EON'2003) at the International Semantic Web Conference (ISWC'2003). Pages 57-70, 2003.

[Für, 04] F. Fürst. 'Opérationalisation des ontologies : une méthodologie et son application au modèle des Graphes Conceptuels'. Dans : Journal Electronique d'Intelligence Artificielle (JEDAI). 2004.

[Für, 04a] F. Fürst. 'Opérationnalisation d'ontologie : une méthodologie et son application au modèle des Graphes conceptuels', article Institut de Recherche en Informatique de Nantes. 2004.

[Für, 04b] F. Fürst. 'Contribution à l'ingénierie des ontologies : une méthode et un outil d'opérationnalisation', Thèse de Doctorat, École Polytechnique de l'Université de Nantes (EPUN). 2004.

[Für, 05] F. Fürst. 'Axiom-based ontology matching: a method and an experiment'. Research report, volume: 05, n° 02, LINA, Nantes. 2005.

[Gen, 10] D.Genest. 'Cogitant: Reference Manual'. En ligne sur la page : http://cogitant.sourceforge.net . 2010.

[Góm, 97] A. Gómez-Pérez, M. Fernández, N. Juristo. 'METHONTOLOGY: From Ontological Art Towards Ontological Engineering'. Symposium on Ontological Engineering of AAAI. Stanford (California). Pages: 33–40. 1997.

[Góm, 99] Gómez-Pérez. 'Ontological Engineering: A state of the art. Expert Update', université Politécnica de Madrid, Espagne. 1999.

[Góm, 04] Gómez-Pérez, Feràndez-Lopez, Oscar Corcho. 'Ontological Engineerin'. Handbook on Ontologies-Springer. 2004.

[Gru, 93] T.R. Gruber. 'A Translation Approach to Portable Ontology Specifications'. Knowledge Acquisition, volume: 5, n° 2. Pages: 199-220. 1993.

[Gru, 95] T.R. Gruber. 'Toward principles for the design of ontologies used for knowledge sharing'. International Journal of Human Computer Studies. 1995.

[Gua, 95] N. Guarino, P. Giaretta. 'Ontologies and Knowledge Bases: Towards a Terminological Clarification'. Dans: Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. Pages: 25-32. Amsterdam: IOS Press. 1995.

[Hae, 95] O. Haemmerlé. 'CoGITo: une plateforme de développement de logiciel sur les Graphes Conceptuels'. Thèse de Doctorat, Université de Montpellier II. 1995.

[Hor, 02] I. Horrocks. 'Reasoning with Expressive Description Logics: Theory and Practice'. Dans: Proceedings of the 18th International Conference on Automated. 2002.

[Kab, 00] A. Kabbaj. 'From Prolog++ to Prolog+CG : a CG objet-oriented logic programming language'. Dans: Proceedings of the International Conference on Conceptual Structures (ICCS'00), volume: 1867. Pages: 540-554. Springer-Verlag LNAI. 2000.

[Kar, 92] P. Karp. 'The design space of frame knowledge representation systems', SRI International AI Center. 1992.

[Kay, 97] D. Kayser. 'La représentation des connaissances'. édition HERMES. 1997.

[Lac, 06] X. Lacot. 'Introduction à owl, un langage xml d'ontologies web enjeux, objectifs et mise en œuvre'. 2006.

[Las, 99] O. Lassila, R.R. Swick. 'Resource Description Framework (RDF) Model and Syntax Specification'. W3C Recommendation. 1999.

[Las, 01] O. Lassila, D. McGuinness. 'The Role of Frame-Based Representation on the Semantic Web'. Technical Report KSL-01-02, Knowledge Systems Laboratory.Stanford University, Stanford, California. 2001.

[Luk, 00] S. Luke, J. Heflin. 'SHOE 1.01 Proposed Specification', SHOE Project. 2000.

[Mac, 91] MacGregor R. 'Inside the LOOM Description Classifier'. SIGART Bulletin, vol 2, n° 3, p 88–92. 1991.

[Mcc, 80] J McCarthy. 'Circumscription—A Form of Non-Monotonic Reasoning'. Artificial Intelligence, volume: 13. Pages: 27-39. 1980.

[Mae, 01] A. Maedche, S. Staab. 'Ontology Learning for the Semantic Web'. Dans: IEEE Intelligent Systems, Special Issue on the Semantic Web, volume: 16, n°2. Pages: 72–79. 2001.

[Miz, 97] R. Mieguchi, M. IkedaA. 'Towards ontology engineering'. Dans: Proceedings of the Joint Pacific Asian Conference on Expert Systems. 1997.

[Mug, 92] M. Chein, M.L. Mugnier. 'Conceptual Graphs : Fundamental notions'. Revue d'intelligence artificielle, volume : 06, n°4. Pages: 365-406. 1992.

[Mug, 95] M. Gruninger, M. S. Fox. 'Methodology for the design and evaluation of ontologies'. Dans : Proceedings of the Workshop on Basic Ontological Issues on Knowledge Sharing of the IJCAI'95 conference. 1995.

[Mug, 96] M. Mugnier, M.L. Chein. 'Représenter des connaissances et raisonner avec des graphes', Article publié dans R.I.A, volume : 10, n°1. Pages : 7-56, LIRMM (CNRS Université Montpellier II). 1996.

[Mug, 00] M. L. Mugnier. 'Knowledge Representation and Reasonings Based on Graph Homomorphism'. Dans: Proceedings of the 8th International Conference on Conceptual Structures (ICCS'2000), volume: 1867. Pages: 172-192. Springer-Verlag LNAI. 2000.

[Mus, 92] M.A. Musen. 'Dimensions of knowledge sharing and reuse'. Computers and Biomedical Research, volume: 25. Pages: 435-467. 1992.

[Nap, 97] A.Napoli. '*Une introduction aux logiques de descriptions*". Rapport de recherche n° 3314. 1997.

[Nec, 91] R.E. Neches, T. Fikes, T. Finin, T. Gruber. '*Enabling technology for knowledge sharing*', AI Magazine, R. 1991.

[Noy, 00] N. Noy, R. W. Fergerson, M. A. Musen. 'The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility'. Dans: Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW'00). 2000.

[Pan, 04] J.Z. Pan, e. Franconi, S. Tessaris, G. Stamou. 'Specification of Coordination of Rule and Ontology Languages'. Rapport technique D2.5.1, Knowledge Web Project (FP6 Network of Excellence). 2004.

[Psy, 03] V. Psyché. R. Mizoguchi. B. Bourdeau. 'Ontology Development at the Conceptual Level for Theory-Aware ITS Authoring Systems.' Dans; Proceeding of Artificial Intelligence in Education (AIED03). 2003.

[Psy, 03a] V. Psyché. O. Mendes. J. Bourdeau. 'Apport de l'ingénierie ontologique aux environnements de formation à distance', Centre de recherche LICEF, Télé-université, Université Fédérale à Paraiba – Brésil. 2003.

[Psy, 07] V. Psyché. 'Rôle des ontologies en ingénierie des EIAH : cas d'un système d'assistance au design pédagogique'. Université du Québec a Montréal, Thèse Doctorat en informatique cognitive. 2007.

[Sal, 97] E. Salvat. 'Raisonner avec des opérations de graphes : graphes conceptuels et règles d'inférences'. Thèse de Doctorat, Université de Montpellier II. 1997.

[Sal, 98] E. Salvat, M.L. Mugnier. 'Sound and Complete Forward and Backward Chainings of Graph Rules'. Dans: Proceedings of the 4th International Conference on Conceptual Structures (ICCS'1996), volume: 1115. Pages: 248–262. Springer-Verlag LNCS. 1998.

[Sow, 84] '*Semantics of Conceptual Graphs'.* IBM Systems Research Institute. **J.F. Sowa. 1984.**

[Sow, 84a] J. F. Sowa. 'Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley. 1984.

[Sow, 91] J.F. Sowa. 'Principes of semantic networks: Exploration in the representation of Knowledge'. Morgan KAUFMANN publishers INC. 1991.

[Sow, 00a] J.F. Sowa. 'Recommandation ISO pour les graphes conceptuels'. disponible a l'adresse : www.jfsowa.com\CG\CGDPANSW.htm. 2000.

[Sow, 00b] J.F. Sowa. 'Support mathématique pour la manipulation des graphes conceptuels'. disponible à l'adresse : www.jfsowa.com\www.jfsowa.com\LOGIC\MATHW.htm. 2000.

[Sti, 94] M. Stickel. R. Waldinger. M. Lowry. T. Pressburger. I. Underwood. 'Deductive composition of astronomical software from subroutine libraries'. Dans: Proceedings of the Twelfth International Conference on Automated Deduction (CADE-12), Nancy, France. Pages: 341-355. 1994.

[Stu, 98] R. Studer, R. Benjamins, D. Fensel. 'Knowledge engineering: principles and methods', Data and Knowledge Engineering, volume: 25, n° 1-2. Pages: 161-197. 1998.

[Sur, 02] Y.Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, 'OntoEdit: Collaborative Ontology Engineering for the Semantic Web. Dans I. Horrocks, J. A. Hendler (Éd.)'. First International Semantic Web Conference (ISWC'02). Pages: 221–235. (Lecture Notes in Computer Science LNCS 2342) Springer-Verlag. 2002.

[Swa, 97] B. Swartout, P. Ramesh, K. Knight, T. Russ. 'Towards Distributed Use of Large-Scale Ontologies'. Dans: Spring Symposium on Ontological Engineering of AAAI. Pages: 138-148. Standford University, California. 1997.

[Tra, 04] A. Traoré, D. Hérin. **'**OWL et la description de ressources pédagogiques'. WorkShop 2 AS-WebLearn – Montpellier. 2004.

[Usc, 96] M. Uschold. M. Grüninger. 'Ontologies: Principles, Methos and Applications'. Journal of Knowledge Engineering Review, volume: 11, n° 2. Pages: 93-155. 1996.

[Wer, 95] M. Wermelinger. 'Conceptual Graphs and First Order Logic'. Dans: Proceedings of the International Conference on Conceptual Structures (ICCS'1995), volume: 954. Springer-Verlag LNAI. 1995.

Annexe A : Grammaire du format BCGCT de CoGITaNT 5

Les notations utilisées ci-dessous sont les suivantes [Gen, 10]:

- Un non terminal est représenté entre < >
- Un terminal est écrit normalement
- Une alternative est matérialisée par le symbole |
- Une séquence optionnelle (apparaissant 0 ou 1 fois) est représentée entre [ ]
- Une séquence apparaissant entre 0 et *n* fois est représentée entre { }*
- Une séquence apparaissant entre 1 et *n* fois est représentée entre { }+

```
<fichier> ::=  [<propriétés>]
        Begin
        [<support>]
        {<graphe>}<règle>*
        End

<support> ::= Support: [<ident_support>] [<propriétés>] [<tailles_support>]
[+] ;
          [<ensemble_TCon>]
          [<ensemble_TRel>]
          [<ensemble_TEmb>]
          [<conformité>]
          [<ensemble_TBan>]
          EndSupport;

<propriétés> ::= {{<entier>|<chaîne>:<entier>|<chaîne>;}*}

<tailles_support> ::= (<entier>,<entier>,<entier>,<entier>)

<ensemble_TCon> ::= EnsTCon:
                        ConceptTypes:
                        {<tConcept>;}+
                        EndConceptTypes;
                        [<ordre_TCon>]
                  EndTCon;

<tConcept> ::= <ident_TCon>[<propriétés>]

<ordre_TCon> ::= Order:
                        {<ident_TCon> < <ident_TCon>;}+
                  EndOrder;

<ensemble_TRel> ::= EnsTRel:
                        RelationTypes:
                        {<tRelation>;}+
                        EndRelationTypes;
                        [<ordre_TRel>]
                  EndTRel;

<tRelation> ::= <ident_TRel>[<propriétés>]
<ordre_TRel> ::= Order:
                        {<ident_TRel> < <ident_TRel>;}+
                  EndOrder;

<ensemble_TEmb> ::= EnsTNes:
```

```
                              NestingTypes:
                              {<tEmboîtement>;}+
                              EndNestingTypes;
                              [<ordre_TEmb>]
                        EndTNes;

<tEmboîtement> ::= <ident_TEmb>[<propriétés>]

<ordre_TEmb> ::= Order:
                 {<ident_TEmb> < <ident_TEmb>;}+
                 EndOrder;

<conformité> ::= Conf:
                 {<ident_marqueur>,<ident_TCon>[<propriétés>];}+
                 EndConf;

<ensemble_TBan> ::= BannedTypes:
                 {<ident_TCon>{,<ident_TCon>}+;}+
                 EndBannedTypes;

<graphe> ::= Graph: <ident_gc> [<propriétés>];
                 [Nature: <ident_nature>;]
                 [Set: <ident_ensemble>;]
                 {
                 Concepts:
                       {<concept>;}+
                 Relations:
                       {<relation>;}*
                 Edges:
                       {<arc>;}*
                 }*
                 [<propriétés>]
           EndGraph;

<concept>::=<ident_concept>=[<ident_TCon>{,<ident_TCon>}+[:<référent>[:<descr
iption>]][<propriétés>]]

<référent> ::= <ident_marqueur>|*|$<ident_variable>

<description> ::= {(<ident_TEmb>,<ident_graphe> [<propriétés>])}+|**

<relation> ::= <ident_relation>=(<ident_TRel>[<propriétés>])

<arête> ::= <ident_relation>,<ident_concept>,<entier>

<règle> ::= Rule:[<ident_regle>][<propriétés>];
                 Hypt:{<graphe>}*
                 Conc:{<graphe>}*
                 ConnectionPoints:
                       {<couple>;}+
           EndRule;

<ident_xxx> ::= <chaîne>

<entier> ::= {0..9}+
<chaîne> ::= a..zA..Z{a..zA..Z0..9}* | "{a..zA..Z0..9}+"
<couple> ::= (<ident_c1>,<ident_c2>[<propriétés>])
```

Figure 44. Grammaire de BCGCT

Annexe B : Correspondance des éléments RDF et GCs

La figure 61 présente la correspondance des éléments RDF/RDFs (triplets ontologiques) et les éléments de graphes conceptuels suivant les règles proposées par notre approche.

/* triple ontologiques */

(classes, rdf:type , rdfs:Class), (classes , rdfs:subClassOf, classes), (Propriétés, rdf:type , rdf:Property), (Propriétés, rdfs:subPropertyOf , Propriétés),(properties, rdfs:domain , classes)and(Propriétés, rdfs:range, classes);
(instances, rdf:type, classes), (" ", rdf:type, classes)

/* transformation */

/* Creation des types concept à l'aide de : rdfs:Resource et rdfs:Datatype ,ainsi les types relation à l'aide de : rdf:Property*/

/* Traitement des triplets ontologiques*/

- (classes, rdf:type, rdfs:Class) → l'ajout du concept typeclasses
- (classes1, rdfs:subClassOf, classes2) → l'ajout de types concept classes1 et classes2, avec : classes1≤ classes2
- (Propriétés, rdf:type, rdf:Property) → l'ajout de la relation typeproperties
- (properties1, rdfs:subPropertyOf, properties2) → l'ajout de la relation binaire properties1 and properties2 avec : properties1≤ properties2
- (Propriétés, rdfs:domain , classes)et (Propriétés, rdfs:range, classes) → l'ajout du type relation binaire Propriétés, de type concept classes, et du domaine.
- (instances, rdf:type, classes) → l'ajout de nœud concept [classes: instances], l'ajout dans le vocabulaire (marqueur de l'instance : classe)
- (" ", rdf:type, classes) → l'ajout de nœud concept [classes:*]

Figure 45. Correspondance des éléments RDF et GCs