République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderahmane Mira de Béjaïa

Faculté des sciences exactes

Département d'Informatique

École Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère

En Informatique

Option

Réseaux et Systèmes Distribués

Thème

Gestion distribuée de compétences

Présenté par

Badrina GASMI

Devant le jury composé de :

Président	Djamil AISSANI	Professeur	U. Béjaïa, Algérie.
Rapporteur	Nacer BOUDJLIDA	Professeur	U. Nancy, France.
Examinateur	Ahmed NACER	Professeur	USTHB, Algérie.
Examinateur	Zizette BOUFAIDA	Professeur	U. Constantine, Algérie.
Invité	Abdelkamel TARI	MCB	U. Béjaïa, Algérie.
Invitée	Hassina TALANTIKITE	MCB	U. Béjaïa, Algérie.

Promotion 2007-2008

A mes parents,

A mes fréres,

A toute ma famille,

A mes amis et collègues,

A tous ceux que j'aime,

Et à tous ceux qui m'aiment.

Remerciements

Au terme de ce travail, je remercie le BON DIEU tout puissant qui m'a donné la force et la volonté d'achever ce travail et nous lui rendons grâce.

Mes très vifs remerciements vont à l'encontre de mon enseignant et mon promoteur Mr. Nacer Boudjlida professeur à l'université de Nancy 1, pour avoir accepté de m'encadrer. Je le remercie également pour ses nombreux conseils, son aide précieuse et sa compréhension durant l'élaboration de ce mémoire.

Je remercie ma co-promotrice, Mme Talantikite Hassina, maitre de conférences à l'université de Bejaïa, pour avoir co-encadré ce travail et d'avoir insisté pour le choix du sujet. Je la remercie également pour sa serviabilité, et soutien constant.

J'adresse évidemment mes sincères remerciements à l'ensemble du jury pour m'avoir honoré en consentant à juger mon travail.

je remercie vivement les enseignants et responsables de l'Ecole Doctorale d'Informatique de Béjaia pour tous les efforts qu'ils ont fournis pour la réussite et l'épanouissement de l'ecole.

Je souhaite également saluer tous mes amis et camarades en post-graduation.

A tous ceux qui d'une façon ou d'une autre, de près ou de loin, m'ont apporté leurs contributions et leur soutien dans l'élaboration de ce travail ; qu'ils trouvent ici l'expression de ma profonde gratitude.

Je remercie de tout c oeur toute ma famille pour la confiance, le soutien et l'aide qu'ils m'ont apporté durant toute la durée de ce mémoire. Spécialement mes frères : Azzedine, A-Elhafid, Adel que j'adore et dont je suis très fière.

Je ne terminerais pas avant d'adresser un immense merci à mes parents pour tous les sacrifices qu'ils ont consentis pour ma réussite et pour le soutien qu'ils m'ont apporté durant toutes mes études, et à qui je dois tout.

Table des matières

Introduction 1

PARTIE I : Etat de l'art 5

Chapitre 1: Gestion de compétences 5

1.1 Introduction 5
1.2 Donnée, information, connaissance et compétence 5
1.2.1 Donnée 5
1.2.2 Information 6
1.2.3 Connaissance 6
1.2.3.1 Définition de la connaissance 6
1.2.3.2 La nature de la connaissance 6
1.2.4 Compétence 7
1.3 Gestion de compétences 8
1.3.1 Définition 8
1.3.2 La représentation de connaissances 9
1.3.2.1 Le problème de la représentation de connaissances 9
1.3.2.2 Classification des formalismes de représentation de connaissances 9
1.3.2.3 Quelques langages de représentation de connaissances 10
1.4 Découverte des offreurs de compétences 16
1.5 L'ontologie dans le cadre de la gestion de compétences 17

1.6 Conclusion 18

Chapitre 2: La médiation 19

2.1 Introduction 19
2.2 Description générale de la médiation 19
2.3 Les services offerts par une architecture de médiation 20
2.4 Les approches de médiation 21
2.4.1 L'approche centralisée 21
2.4.2 L'approche distribuée 21
2.4.2.1 Les limites de la médiation centralisée 21
2.4.2.2 Définition de la fédération de médiateurs 23
2.4.2.3 Architecture conceptuelle pour les médiateurs fédérés dans le cadre de la gestion de compétences 23
2.4.2.4 La satisfaction de la recherche des offreurs de compétences dans une fédération 24
2.4.2.5 La propagation de requêtes dans une fédération 24
2.5 Conclusion 25

PARTIE II : Proposition 27

Chapitre 3: Les graphes conceptuels pour la gestion de compétences 27

3.1 Introduction 27
3.2 Notions de base 27
3.2.1 Rappel du formalisme des graphes conceptuels 27
3.2.2 La projection 28
3.2.3 La normalisation 29
3.2.4 La somme disjointe de deux graphes conceptuels 30
3.2.5 Les graphes irredondants 30
3.2.6 Headed graph 31
3.2.7 Les règles de graphes 31

3.3 Les graphes conceptuels pour la gestion de compétences 32
3.3.1 Gestion de compétences 33
3.3.1.1 La représentation de compétences 33
3.3.1.2 L'organisation de la base de compétences et de connaissances 36
3.3.2 La découverte des offreurs de compétences 37
3.3.2.1 Représentation de la requête 38
3.3.2.2 Satisfaction locale de la requête 38
3.3.2.3 Recherche des réponses composites 39
3.3.2.4 Satisfaction distribuée de la requête 41
3.4 Comparaison avec des travaux similaires 46
3.5 Conclusion 48

Chapitre 4: Mise en oeuvre 49

4.1 Introduction 49
4.2 Les outils de développement utilisés 49
4.2.1 CoGui 50
4.2.2 CoGITaNT 50
4.2.2.1 Présentation de la bibliothèque 50
4.2.2.2 Les formats de représentation supportés 50
4.2.2.3 Représentation d'un graphe conceptuel 51
4.2.3 Cygwin 51
4.3 Architecture du système 52
4.4 Communication entre clients et serveurs 53
4.5 La base de compétences et de connaissances 53
4.6 Les services offerts par le serveur de médiation 53
4.6.1 Service Exportation 54
4.6.2 Service Importation 54
4.6.3 Service Coopération 55
4.7 Scénario d'exécution 56
4.7.1 Présentation de la population du domaine 56
4.7.2 Création et exploitation d'une base de compétences 59

4.8 Conclusion 63

Conclusion et perspectives 65

Bibliographie 67

Annexe A : Les outils disponibles pour la manipulation des graphes conceptuels 72

Annexe B : La bibliothèque cogitant 74

Annexe C : CoGUI et le format CoGXML 79

Table des figures

1.1 Lien entre donnée, information, connaissance et compétence 8
1.2 Les catégories des langages de représentation de connaissances 9
1.3 Exemple d'une base de connaissances en logiques de description 11
1.4 Exemple de graphe conceptuel 12
1.5 Exemple de support 14

2.1 Les services d'une architecture de médiation 21
2.2 Architecture de médiation 22
2.3 Une architecture à base des médiateurs fédérés pour la gestion de compétences 24

3.1 Exemple de projection de graphes conceptuels 29
3.2 Exemple de normalisation d'un graphe conceptuel 30
3.3 Exemple de somme disjointe de deux graphes conceptuels 30
3.4 Exemple de graphes redondants et de graphes irredondants. 31
3.5 Exemple de headed graph 31
3.6 Exemple de règles de graphes 32
3.7 Exemple de support 34
3.8 Procédure de publication de compétences 37
3.9 Procédure de la satisfaction locale d'une requête 39
3.10 Procédure de recherche des réponses composites 40
3.11 Procédure de satisfaction d'une sous requête 41
3.12 Procédure de construction d'une requête pour la fédération 44
3.13 Exemple de construction d'une requête 44
3.14 Traitement de la requête de fédération 46

4.1 Architecture de notre système 52

4.2 Diagramme de séquence du processus d'exportation 54
4.3 Diagramme de séquence du processus d'importation 55
4.4 Diagramme de séquence du processus de coopération 56
4.5 La hiérarchie de types de concepts 57
4.6 Les hiérarchies de relations 58
4.7 Exemple1 de publication de compétence 59
4.8 Exemple2 de publication après la somme disjointe 60
4.9 Exemple2 de publication après la normalisation 60
4.10 Exemple2 de publication après l'application des règles 61
4.11 Exemple1 de recherche des offreurs de compétences 61
4.12 Base de compétences du médiateur M2. 62
4.13 Résultat d'exécution de la requête2 sur le médiateur M1. 62
4.14 Résultat d'exécution de la requête2 sur le médiateur M2. 62
4.15 Exemple3 de requêtes 63
4.16 Exemple3 de découverte des offreurs de compétences 64
4.17 L'interface principale de Cogui 79

Liste des tableaux

4.1 Les types de concepts définis avec les règles les définissant 57
4.2 Les types de relations définies avec les règles les définissant 58
4.3 Les méta-connaissances sur les relations 59

Introduction

La gestion de compétences est une méthode permettant aux organisations de mettre au maximum les compétences des collaborateurs à profit. Elle s'inspire du concept que le facteur clé de la réussite de l'organisation repose sur les individus qui en font partie.

Les concepts de compétence et de la gestion de compétences sont très souvent liés au domaine de la gestion des ressources humaines. Elles sont depuis quelques années l'un des principaux chantiers des directeurs des ressources humaines (DRH). Cependant, divers domaines d'application de ces concepts sont aussi possibles tel que la programmation par composants et les affaires électroniques (e-business). Cette diversité des domaines conduit à des définitions très diverses du terme compétence, chacune est liée à son domaine d'application.

D'une façon générale, la gestion de compétences est la façon avec laquelle les organisations gèrent les compétences des individus dans un domaine particulier, elle a l'objectif principal de définir et de maintenir continuellement ces compétences, selon les objectifs visés [12]. La gestion de compétences est un processus complexe, nous le traitons dans ce travail en trois étapes : l'identification ou la représentation de compétences, leurs organisation, et leurs utilisation . L'identification consiste à mettre les compétences sous une représentation formelle qui puisse être exploitée par les machines, celle-ci est l'un des sous domaines de la représentation de connaissances qui essaye d'adapter les langages de représentation de connaissances pour décrire les compétences. Une fois identifiées, les compétences doivent être organisées pour être exploitées ensuite dans la phase d'utilisation.

Dans ce travail, nous nous intéressons au problème de la gestion de compétences mais sans se limiter à un domaine particulier. Notre objectif est de trouver une méthode générale qui soit instanciable dans divers domaines. Nous adoptons à cet effet une définition générale du terme compétence comme étant les capacités (ou services) d'une entité quelconque qui peut être physique ou non physique (un objet, un composant logiciel, un partenaire industriel, un service web, etc.) [26][16][15].

Notre motivation principale est d'apporter une contribution à la satisfaction de besoins en recherche des entités (offreurs de compétences) pouvant satisfaire un besoin, il s'agit d'exploiter l'ensembles des compétences acquises dans un domaine particulier lors de la recherche de ces entités. En effet, les réponses ne sont pas du type Oui/Non, mais elles peuvent être coopératives dans le sens où si aucune entité ne satisfait à elle toute seule une demande, on essaye de déterminer quel ensemble d'entités réunies la satisfont. Ainsi, dans le domaine de la programmation par composants, il s'agit de rechercher les composants logiciels offrant certains services (fonctions). De même, dans le domaine du e-business, il s'agit de rechercher des partenaires ayant les compétences requises pour satisfaire les exigences d'un projet donné par exemple.

D'une façon générale, notre but est de trouver une méthode pour procéder à la construction d'un système permettant à une entité de se décrire, on dira de publier ou d'expliciter ces compétences pour qu'elles puissent être explorées par d'autres entités. Cet objectif nous a naturellement orienté vers un modèle à base des médiateurs (traders) distribués et coopératifs.

Les médiateurs sont définis comme étant des modules qui occupent une couche explicite et active entre les applications et leurs ressources [61]. Dans une architecture de médiation une entité appelée exportateur publie ses compétences auprès d'un ou de plusieurs médiateurs. Des entités appelées importateurs adressent des demandes de recherche d'exportateurs dotés de certaines compétences à l'un des médiateurs à charge pour lui d'essayer de trouver des exportateurs satisfaisant la demande. La coopération de médiateurs peut être requise lorsqu'une application complexe veut accéder à des données par le biais de divers médiateurs [26].

Pour répondre à notre objectif, nous nous somme retrouvés face à plusieurs questions auxquelles il faut répondre.

1. Quel langage de représentation formelle de compétences ?
2. Quelle organisation de ces compétences ?
3. Quel algorithme de découverte ?
4. Et quel protocole de fédération sur lequel les médiateurs s'appuient pour coopérer entre eux ?

Suite à notre problème, nous structurons la présentation de notre travail en deux parties :

La première partie est consacrée à l'état de l'art, cette partie est composée de deux chapitres. **Le premier chapitre** présente le problème de la gestion de compétences,

il introduit également les différents formalismes de représentation de connaissances qui ont été (ou qui peuvent être) utilisés pour la description de compétences. **Le deuxième chapitre** introduit la notion de médiateurs distribués et leur fédération dans le cadre de la gestion de compétences.

La seconde partie présente notre proposition : cette partie est également composée de deux chapitres. **Le troisième chapitre** décrit le formalisme de représentation de connaissances adopté, à savoir, une représentation fondée sur les graphes conceptuels [28] ainsi que l'utilisation que nous faisons de ce formalisme pour la gestion de compétences, et enfin **le quatrième chapitre** présente une expérimentation de notre modèle.

Enfin, ce rapport s'achève par une conclusion générale dans laquelle nous présentons un rappel des principaux résultats obtenus et quelques perspectives.

Etat de l'art

Chapitre 1

Gestion de compétences

1.1 Introduction

La gestion de compétences est un problème crucial dans plusieurs types d'applications et plus particulièrement dans des applications distribuées coopératives ou des applications de type commerce électronique (e-commerce). Il s'agit d'un ensemble de processus permettant de représenter, d'organiser et d'utiliser les compétences pour assurer les fonctions d'un domaine particulier.

Le sujet de la gestion de compétences a donné lieu à de nombreux travaux et concepts, qui n'ont pas toujours conduit au développement de méthodes suffisamment opérationnelles pour qu'elles soient réellement adoptés dans les domaines d'études. L'une des raisons de cette difficulté est le manque d'une définition de la compétence qui soit largement adoptée [24]. Il s'avère alors nécessaire d'éclaircir cette notion ainsi que les notions liées. Cela est l'objectif de la section suivante.

1.2 Donnée, information, connaissance et compétence

Les tentatives de définition du terme compétence sont nombreuses, celles-ci varient significativement selon les auteurs. Cependant, toutes ces définitions s'accordent au moins sur le fait que cette notion n'est pas indépendante de celle de la connaissance qui est elle-même le fruit d'un cycle [11] : données - informations - connaissances.

1.2.1 Donnée

Les données sont définies en tant qu'éléments donnés, accordés, ou admis par tous ; des éléments sur lesquels quelque chose peut être discutée, elles sont des représentations dont les significations dépendent du système de représentation (i.e. symboles, langage,

etc.) utilisé [43]. Une donnée est définie aussi dans [35] comme le résultat d'observations. Une donnée peut être aussi définie comme toute représentation à laquelle une signification peut être attachée, elle peut être qualitative (les chiffres) ou quantitative (les couleurs), et elle n'a pas de sens en elle-même [43].

1.2.2 Information

L'information est une collection de données organisée pour donner forme à un message le plus souvent sous forme visible, imagée, écrite ou orale, pour réduire une incertitude et transmettre quelque chose qui déclenche une action [43]. Elle est définie aussi dans [49] comme toutes les données externes aux personnes, communiquées oralement par d'autres ou médiatisées dans des matériels sur divers supports numériques, imprimés ou analogiques.

1.2.3 Connaissance

1.2.3.1 Définition de la connaissance

D'une façon générale, la connaissance, est ce grâce à quoi nous élaborons des réponses et des solutions à des problèmes nouveaux [5]. La connaissance est le résultat de toute construction mentale effectuée par un individu à partir d'informations ou d'autres stimuli [49]. Dans [50], la connaissance est décrite comme les éléments permettant de construire de nouveaux faits ou permettant de déterminer de nouvelles actions à entreprendre et non pas comme des éléments descriptifs. Les questions liées à la connaissance sont celles de son acquisition (apprentissage), de sa représentation et de son utilisation (raisonnement)[17].

1.2.3.2 La nature de la connaissance

Les connaissances se répartissent en deux catégories : les connaissances explicites et les connaissances tacites.

1. **La connaissance explicite :** C'est la connaissance qui peut être codifiée. Elle est stockée et représentée dans les documents, des bases de données, des référentiels, etc. Il s'agit donc d'une connaissance consultable et transférable [7].
2. **La connaissance tacite :** La connaissance tacite telle que définie dans [51], est la connaissance personnelle cachée, même de la conscience du connaisseur, qui ne peut être saisie que par la démonstration de notre connaissance exprimable et par nos actes. La connaissance tacite est difficilement transférable, il s'agit d'une connaissance qui ne s'acquière pas de la même façon que la connaissance explicite

car elle ne réside pas dans les répertoires, mais dans l'expérience, le jugement, l'intuition, et l'inconscient [7]. Les connaissances tacites ont un aspect personnel qui les rend difficiles à formaliser et à communiquer [34].

1.2.4 Compétence

A ces concepts, nous pouvons ajouter la notion de compétence. Selon le domaine considéré, plusieurs définitions ont été données à ce terme, on trouve :

- Dans [39], la compétence est considérée comme un ensemble d'aptitudes et de talents, de traits de personnalités et de connaissances acquises pour mener à bien les tâches assignées à un individu. Dans cette perspective, la connaissance est un sous-élément de la compétence, au même titre que les attitudes et les aptitudes.
- Une compétence est aussi définie dans [5] comme étant l'état ou qualité d'être bien qualifié pour réaliser un rôle spécifique.
- Dans [13], une compétence est la manière de mettre en pratique les connaissances, savoir-faire et aussi les attitudes, dans un contexte spécifique.
- Dans le domaine multi-agents, la compétence représente la connaissance par l'agent (ou les agents) d'une pratique relative à l'exécution de l'action [23]. Ainsi, dire que " Ali est compétent pour peindre " sous-entend que Ali sait ce qu'il faut faire pour peindre (i.e. il connaît la succession de gestes à faire pour peindre).
- Dans [26][16][15], le terme compétence est défini d'une façon générale indépendamment du domaine d'application comme étant "la capacité d'une entité à réaliser des actions". De ce fait, le terme capacité a été utilisé à la place de compétence vu que ce dernier est très souvent associé au domaine de la gestion des ressources humaines.

La figure 1.1 [49] situe les quatres concepts les uns par rapport aux autres.

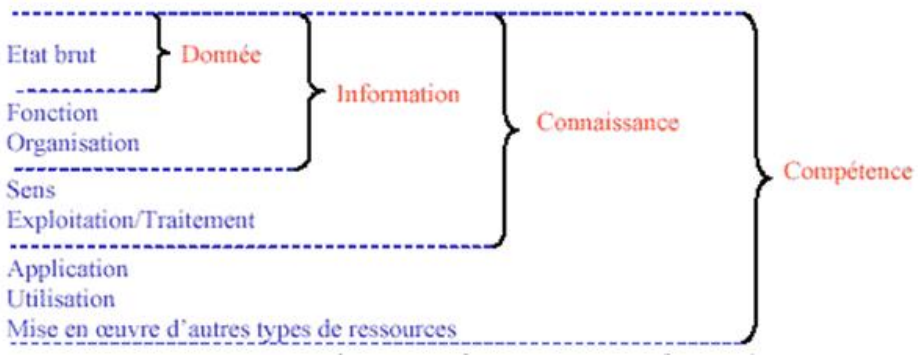


FIG. 1.1 – Lien entre donnée, information, connaissance et compétence

Pour bien clarifier ces quatre concepts, considérons l'exemple [11] ci-dessous :

1. **Données :**
   - **Donnée 1 :** 10°
   - **Donnée 2 :** Paris
   - **Donnée 3 :** Moi
2. **Informations :**Organisation des données.
   - **Information1 :** La température est de 10° à Paris aujourdhui.
   - **Information2 :** Je suis à paris.
3. **Connaissance :** Exploitation des informations.
   - Je suis à Paris aujourdhui⇒ je m'habille chaudement.
4. **Compétence :**Utilisation des connaissances.
   - je mets un manteau.

1.3 Gestion de compétences

1.3.1 Définition

Il s'agit de trouver des méthodes pour la capture et l'organisation des compétences acquises pour assurer les activités du domaine d'étude, ainsi que l'identification et le classement des objets et leurs compétences selon un mécanisme d'inférence intelligent.

D'une manière générale, le processus de la gestion de compétences inclut deux étapes [26] à savoir, (1)la représentation ainsi que l'organisation des compétences et (2) le raisonnement sur les compétences.

La représentation de compétences est considérée comme l'un des sous domaines de la représentation de connaissances en intelligence artificielle. Il s'agit d'adapter les

langages de représentation de connaissances pour pouvoir représenter les compétences et leurs caractéristiques. Le but de la représentation est évidemment à des fins de raisonnement.

Dans ce qui va suivre, on présentera une brève introduction aux formalismes de représentation de connaissances qui peuvent être utilisés pour la représentation de compétences.

1.3.2 La représentation de connaissances

1.3.2.1 Le problème de la représentation de connaissances

Le problème de la représentation de connaissances est un problème central de l'intelligence artificielle : comment représenter dans un langage informatique des connaissances humaines de toutes sortes, de telle façon que cette représentation puisse être utilisée par un ordinateur pour effectuer des raisonnements ? Les applications modernes de l'informatique ont conduit à un usage généralisé des représentations de connaissances dans des contextes variés (recherche d'information, description des ontologies du web sémantique[14], etc.).

Typiquement, un système à base de connaissances se compose d'une base de connaissances construite à partir de la modélisation d'un domaine d'application et de mécanismes d'inférence. Les connaissances sont décrites dans un langage de représentation de connaissances, qui détermine à la fois les descriptions que l'on peut effectuer et les mécanismes de raisonnement associés.

1.3.2.2 Classification des formalismes de représentation de connaissances

Les formalismes de représentation de connaissances peuvent être groupés en deux catégories ; à savoir les formalismes logiques, et les formalismes non logiques (voir figure 1.2 [33]).

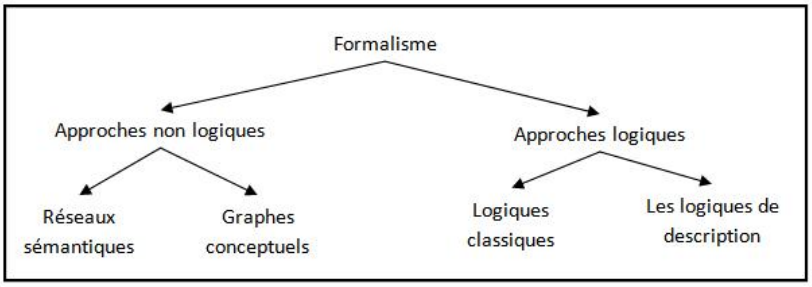


FIG. 1.2 – Les catégories des langages de représentation de connaissances

Les caractéristiques des approches non logiques sont [33] :

1. Le sens intrinsèque attribué aux structures,
2. L'existence de procédures adaptées et souvent efficaces et
3. Une plus grande facilité de révision.

Les caractéristiques des approches logiques sont [33] :

1. Le sens est externe (interprétation),
2. Des procédures génériques plus ou moins adaptées selon le choix de la logique.

Le choix du formalisme à utiliser dépend à la fois du domaine d'application, des opérations à mettre en oeuvre sur ces connaissances et de la culture du modélisateur.

1.3.2.3 Quelques langages de représentation de connaissances

1. Les logiques de description :

Les logiques de description forment une famille de langages de représentation de connaissances. Ils ont été étudiés et utilisés dans plusieurs systèmes à base de connaissances tel que les système de gestion de compétences. L'accent au sein des logiques de descriptions est mis sur les services de raisonnement et plus particulièrement sur ceux pour la prise de décision : l'objectif principal des logiques de description consiste à pouvoir raisonner efficacement pour minimiser les temps de réponse.

Les logiques de description utilisent une approche ontologique, c'est-à-dire que pour décrire les individus d'un domaine, elles requièrent tout d'abord la définition des catégories générales d'individus et les relations logiques que les individus ou catégories peuvent entretenir entre eux. Cette approche ontologique est naturelle pour le raisonnement puisque même si la majorité des interactions se déroulent au niveau des individus, la plus grande partie du raisonnement se produit au niveau des catégories.

La modélisation des connaissances d'un domaine avec les logiques de description se réalise en deux niveaux. Le premier, le niveau terminologique ou TBox, décrit les connaissances générales d'un domaine alors que le second, le niveau factuel ou ABox, représente une configuration précise.

Un exemple d'une base de connaissances en logique de description est présenté dans la figure 1.3 [27]. Le coté gauche de cette figure représente la TBox, tandis que la partie droite représente la ABox.

Une TBox comprend la définition des concepts et des rôles. Les concepts modélise des classes d'individus (ensemble d'individus) : aussi un concept correspond à une

TBox	ABox
$Femelle \sqsubseteq \top \sqcap \neg Mâle$	$Humain(Anne)$
$Mâle \sqsubseteq \top \sqcap \neg Femelle$	$Femelle(Anne)$
$Animal \equiv Mâle \sqcup Femelle$	$Femme(Sophie)$
$Humain \sqsubseteq Animal$	$Humain(Robert)$
$Femme \equiv Humain \sqcap Femelle$	$\neg Femelle(Robert)$
$Homme \equiv Humain \sqcap \neg Femelle$	$Homme(David)$
$Mère \equiv Femme \sqcap \exists relationParentEnfant$	$relationParentEnfant(Sophie, Anne)$
$Père \equiv Homme \sqcap \exists relationParentEnfant$	$relationParentEnfant(Robert, David)$
$MèreSansFille \equiv Mère \sqcap$	
$\forall relationParentEnfant.\neg Femme$	
$relationParentEnfant \sqsubseteq \top_R$	

FIG. 1.3 – Exemple d'une base de connaissances en logiques de description

entité générique d'un domaine d'application et un individu à une entité particulière, instance d'un concept. Les rôles modélisent les relations entre les classes (une relation binaire entre individus). Un concept possède une description structurée qui se construit à l'aide d'un ensemble de constructeurs introduisant les rôles associés aux concepts et les restrictions attachées à ces rôles.

Les concepts et rôles peuvent être atomiques ou composés. Les concepts et rôles atomiques constituent les entités élémentaires d'une TBox. Les noms débutants par une lettre majuscule désignent les concepts, alors que ceux débutant par une lettre minuscule dénomment les rôles (dans l'exemple de la figure 1.3 Femelle, Mâle, Homme et Femme sont des concepts, et relationParentEnfant est un rôle).

Les logiques de description prédéfinissent minimalement quatre concepts atomiques : le concept T et le rôle Tr, les plus généraux de leurs catégories respectives, et le concept $\perp$ ainsi que $\perp$r les plus spécifiques.

Les concepts et rôles atomiques peuvent être combinés au moyen de constructeurs pour former respectivement des concepts et des rôles composés. Par exemple, le concept composé Mâle $\bigcap$ Femelle résulte de l'application du constructeur $\bigcap$ aux concepts atomiques Mâle et Femelle. Les différentes logiques de description se distinguent par les constructeurs qu'elles proposent. Plus les logiques de description sont expressives, plus les chances sont grandes que les problèmes d'inférence soient non décidables ou de complexité très élevée.

Une ABox décrit les individus en les nommant et en spécifiant en termes de concepts et de rôles, des assertions qui portent sur ces individus nommés. Plusieurs ABox peuvent être associés à une même TBox ; chacune représente une configuration constituée d'individus, et utilise les concepts et rôles de la TBox pour l'exprimer.

Une ABox contient un ensemble d'assertions sur les individus : (1) des assertions

d'appartenance (par exemple : Humain(Anne)), et (2) des assertions de rôles (Par exemple : relationParentEnfant(Sophie,Anne)).

2. les graphes conceptuels :

Les graphes conceptuels[1] est un formalisme de représentation de connaissances de type réseaux sémantiques introduit pour la première fois par Sowa [28] en 1984. L'accent a été mis sur le fait que toutes les autres formes de représentation pourraient être exprimées sous la forme de graphes conceptuels. Cet aspect de généralité est au c oeur des cinq avantages du formalisme que la communauté des graphes conceptuels se plaît à rappeler : la généralité, l'existence de standards, l'expressivité, la rapidité de traitements et enfin, les potentialités d'interaction homme-machine.

Un graphe conceptuel est défini comme étant un graphe [18] :

- **Orienté :** c'est-à-dire qu'un lien entre deux n oeuds u et v du graphe relie soit u vers v, soit v vers u.
- **Fini :** tout graphe dans une mémoire d'un ordinateur ne peut avoir qu'un nombre fini de n oeuds,
- **biparti :** il ne possède que deux sortes de n oeuds - les concepts et les relations conceptuelles- chaque arc ne relie que deux n oeuds de types différents.
- **Etiqueté :** les n oeuds sont étiquetés
- **Non nécessairement connexe :** Un graphe connexe veut dire que si deux parties n'étaient pas connectées entre elles on aurait deux graphes conceptuels.

Le but du formalisme des graphes conceptuels est de représenter la sémantique dans une forme qui soit précise, lisible et utilisable par un ordinateur.

Les graphes conceptuels sont très proches de la langue naturelle comme l'illustrent l'exemple de la figure 1.4 qui exprime la phrase "L'homme h est le père de la personne p".

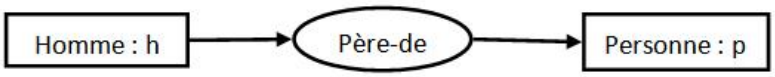


FIG. 1.4 – Exemple de graphe conceptuel

D'une manière formelle, un graphe conceptuel est composé de deux sortes de n oeuds

[1]Nous parlons ici des graphes conceptuels simples. Pour une étude plus approfondie des graphes conceptuels il est intéressant de lire avec intérêt les deux livres de John Sowa [57] et [28].

(concepts et relations conceptuelles) étiquetés à partir d'un support et d'arcs.

Définitions [32] :

1. Un concept est la représentation d'un objet de l'univers du discours. Il est l'assemblage de deux parties : un référent qui identifie l'objet représenté et un type qui classifie l'objet représenté.

2. Une relation conceptuelle représente une association entre un ou plusieurs concepts. Une relation conceptuelle est composée d'un libellé de relation qui identifie le type de la relation, et d'arcs qui lient la relation aux concepts associés.

3. Un arc lie une relation à un concept dans un graphe conceptuel. Un arc est dit appartenir à la relation et attaché au concept.

Dans l'exemple de la figure 1.4, l'homme h est représentée par le concept "Homme : h", la personne p est représenté par le concept "Personne : p" et la relation de parenté qui existe entre l'homme h et la personne p est représentée par un arc joignant les deux concepts et porte le type de la relation, ici "père-de".

Le support d'un graphe conceptuel est composé de [19] :

1. Une hiérarchie de types de concepts organisée autour de la relation de spécialisation/généralisation.

2. Un ensemble de relations. Cet ensemble peut être organisé autour de la relation de spécialisation/généralisation en plusieurs hiérarchies, chaque hiérarchie organise les types de relations de même arité[2] (ou valence).

3. Un ensemble de marqueurs ou référents qui représentent les entités du domaine que l'on cherche à modéliser.

4. Une relation de conformité qui associe un marqueur au(x) type(s) de concept dont il relève.

5. Et, une signature de relations qui représente un ensemble des graphes qui expriment les contraintes liées à chaque relation, elle définie le nombre et le type de ses arguments. Les graphes de signature constituent des graphes élémentaires à partir desquels ont peut construire des graphes plus complexes.

Un exemple incomplet de support est donné dans la figure 1.5. L'encerclé en rouge représente les types de concepts alors que l'encerclé en bleu représente les relations

[2] l'arité d'une relation représente le nombre de ses arguments

binaires (ayant deux éléments dans la signature). Les relations d'arités différentes seraient représentées dans un autre cercle à droite du cercle bleu de telle sorte qu'elles ne soient pas comparables avec les relations d'arités différentes.

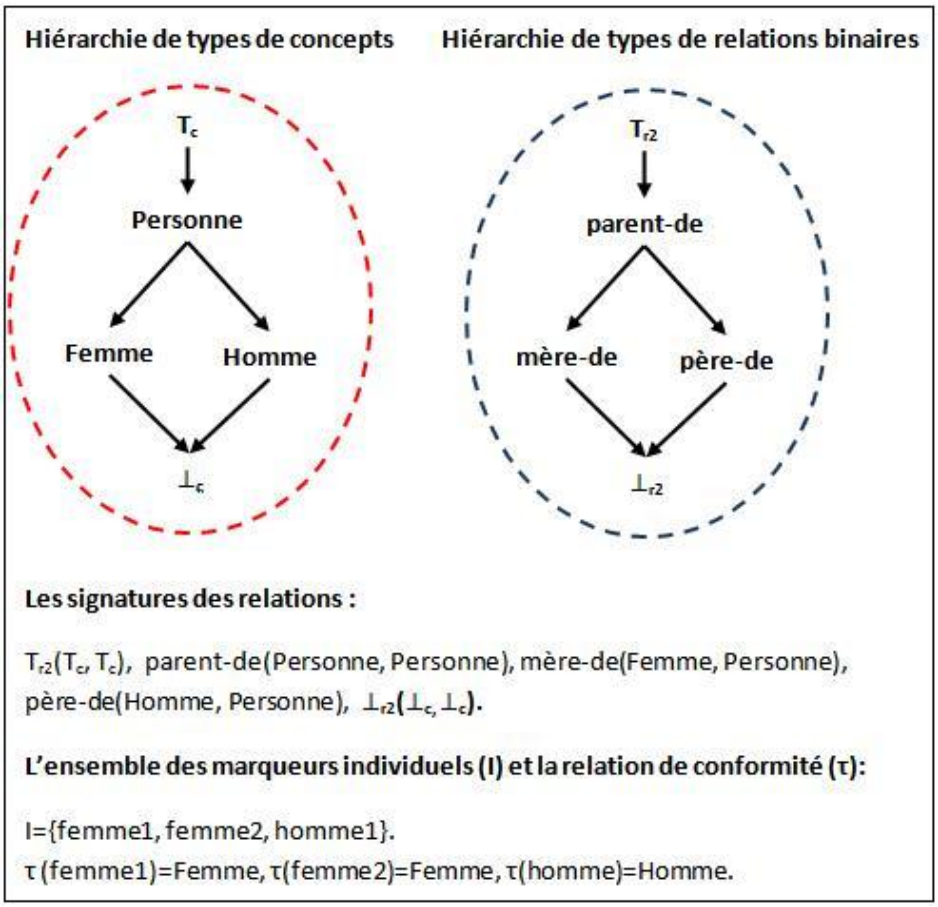


FIG. 1.5 – Exemple de support

Plusieurs représentations des graphes conceptuels sont possibles, nous citons les trois notations les plus connues.

1. **La notation graphique (Display Form) :** Un graphe conceptuel représente une formule logique les noms et les arguments sont représentés par des n oeuds. Les arcs du graphe relient les noms des prédicats à leurs arguments. On utilise des rectangles pour les concepts (arguments) et des cercles pour les relations (prédicats). Cette représentation est dite représentation graphique d'un graphe conceptuel. La figure 1.4 montre un exemple d'un graphe conceptuels sous sa forme graphique .

2. **La représentation linéaire (Linear Form) :** Il existe une forme textuelle dite

linéaire pour la représentation des graphes conceptuels. Dans la forme linéaire les concepts sont représentés entre crochets et les relations entre parenthèses, les arcs étant représentés par des flèches orientés. Le graphe de la figure 1.4 s'écrit de la manière suivante en forme linéaire :

[Homme : h]->(père-de)->[Personne : p]

3. **La notation CGIF :** Il existe d'autres possibilités avec les formats d'échange de graphes conceptuels entre machines tel que CGIF[3]. Voici le même exemple sous sa représentation CGIF : père-de (Homme : h, Personne : p).

Le formalisme des graphes conceptuels ne définit pas uniquement des structures de données, mais il définit également un certain nombre d'opérations permettant de les manipuler. Nous rappelons ici l'ensemble des opérations de combinaison de graphes qui permettent de dériver un graphe à partir d'un ou de plusieurs graphes élémentaires.

Soient w le graphe dérivé de deux autres graphes élémentaires u et v (u et v pouvant être identiques). Il y a quatre opérations de dérivation, et seuls les graphes construits à partir de ces quatre opérations sont des expressions du langage des graphes conceptuels :

1. **Simplification :** Si deux relations conceptuelles sont identiques, c'est à dire si elles ont la même arité, même ordre des concepts liés et même type, une des deux relations peut être supprimée ainsi que les arcs la reliant aux concepts qu'elle associe.

2. **Restriction de concept :** Un concept de u peut être restreint en remplaçant son type par un type plus spécialisé (sous-type) ou en remplaçant un concept générique représentant une entité anonyme (inconnue) par un concept individuel représentant une entité identifiée (connue).

3. **Restriction de relation :** Le type d'une relation de u peut être remplacé par un sous-type.

4. **Jointure :** Si un concept c de u est identique à un concept d de v, w est le graphe résultant de l'union de u et v en supprimant d et en reliant les arcs arrivant et partant de d à c.

En se basant sur ces opérations, Sowa [28] introduit un ordre partiel sur les graphes . Cet ordre définit une relation d'inférence sur les graphes permettant de déterminer si les informations représentées par un graphe induisent les informations représentées

[3]Conceptual Graph Interchange Format

par un autre graphe. En d'autres termes, Sowa étend la relation d'ordre partiel sur les types de concepts et de relations aux graphes conceptuels. C'est de cette façon que Sowa construit pour la première fois un index sur les graphes conceptuels. Il classe (indexe) les graphes conceptuels selon une relation de subsomption.

Un graphe est subsumé par un autre s'il est le fruit de l'application de l'une des quatre opérations citées sous dessus.

La relation de subsomption (spécialisation/généralisation) qu'on vient d'évoquer peut aussi être définie par l'opérateur de projection considéré comme un morphisme de graphes bipartis (les n oeuds concepts sur les n oeuds concepts, les n oeuds relations sur les n oeuds relations).

Les graphes conceptuels étant un système logique. En effet, un graphe conceptuel peut être traduit par une fonction notée φ en une formule bien formée de la logique des prédicats du 1er ordre. Par exemple, le graphe conceptuel de la figure 1.4 peut être traduit en la formule du premier ordre suivante :
Homme(h)$\wedge$Personne(p)$\wedge$père-de(h,p).
Cette formule peut être ainsi lue : h est un homme, p est une personne et h est le père-de p.

1.4 Découverte des offreurs de compétences

Après avoir défini le langage de représentation de compétences, il s'agit en second lieu de définir un mécanisme de raisonnement en vu de découvrir des entités (offreurs de compétences) possédant certaines compétences requises.

Les requêtes de découverte sont adressées à des bases de connaissances écrites dans des langages de représentation de connaissances donnés. Dans plusieurs situations, ces bases de connaissances sont souvent de nature distribuées et hétérogènes, par conséquent, la découverte des offreurs de compétences peut occuper trois niveaux [26] :

1. La requête et la base de connaissances sont exprimées dans le même langage de représentation de connaissances. Dans ce cas la base de connaissances est localisée dans un seul serveur. Ce niveau se focalise dans le calcul du matching et le calcul des réponses composites. Ce niveau est appelé une satisfaction locale homogène.

2. La requête et la base de connaissances sont exprimées dans le même langage, mais la base de connaissances est distribuée sur plusieurs serveurs. Cette situation se focalise sur les solutions aux problèmes des calculs distribués. Ce niveau est appelé la satisfaction homogène distribuée.

3. Enfin, Le dernier niveau est appelé satisfaction hétérogène distribuée. La requête et la base de connaissances sont exprimées dans des langages différents, et la base de connaissances est distribuée sur plusieurs serveurs et exprimée avec des langages de représentation de connaissances différents. Ce niveau se focalise sur les problèmes de communication et d'interopération entre les différents langages de représentation de connaissances.

1.5 L'ontologie dans le cadre de la gestion de compétences

À l'origine, l'ontologie est une branche de la philosophie dans laquelle les philosophes ont tenté de rendre compte de l'existant de façon formelle [41].

En informatique, une ontologie est un système conceptuel qui permet de partager et de réutiliser des concepts d'un domaine particulier, elle est comprise comme un système de concepts fondamentaux qui sont représentés sous une forme compréhensible par un ordinateur.

Le domaine des ontologies attire l'attention parce qu'une ontologie fournit [41] :

1. Une structure conceptuelle de base à partir de laquelle il est possible de développer des systèmes à base de connaissances qui soient partageables et réutilisables,

2. L'interopérabilité entre les sources d'information et de connaissances.

Aujourd'hui, les ontologies sont devenues cruciales dans différents domaines, tels que le web sémantique[14] pour améliorer la recherche d'information ou dans l'ingénierie des connaissances afin d'organiser et partager ces informations. Elle sont cruciales aussi dans le cadre de la gestion de compétences comme dans le cas de e-recrutement [6], puisqu'elles permettent aux fournisseurs et aux chercheurs de compétences de partager un référentiel commun pour décrire leurs compétences d'une façon non ambiguë, précise, sémantique et formelle. Un tel référentiel facilitera sans doute la tâche d'annotation et permettra un enrichissement avec de nouvelles connaissances, inspirées des concepts et des relations de cette ontologie. L'apport de la formalisation réside dans la possibilité d'assurer un raisonnement automatique pour le rapprochement entre offre et demande de compétences.

La notion d'ontologie existait dès les années 1970, sans être nommée et de façon transversale, dans les différents systèmes de représentation de connaissances [29] : c'est la TBox des logiques de description, où l'on décrit les types de termes qui existent dans

notre représentation et leurs caractéristiques ; c'est le support des graphes conceptuels, où l'on décrit des hiérarchies de multi-héritage entre des types de concepts ou des types de relations ; ce sont enfin les schémas des " Frames " et les classes des langages de représentation par objets.

1.6 Conclusion

Ce chapitre s'est porté sur la gestion de compétences. Ce problème est abordé selon deux angles à savoir la représentation de compétences et le raisonnement pour la recherche des offreurs de compétences. Dans ce chapitre, nous avons évoqué les langages de représentation de connaissances vu que la représentation de compétences est un sous domaine de la représentation de connaissances. Ces derniers fournissent une représentation syntaxique et sémantique, ce qui rend évident leur utilisation dans le domaine de la gestion de compétences.

Le chapitre prochain a pour but d'introduire les architectures de médiation et la fédération de médiateurs.

Chapitre 2

La médiation

2.1 Introduction

L'intégration d'informations est un domaine de recherche en informatique qui a pour but de proposer des solutions pour la construction de systèmes de questions/réponses sur des informations distribuées et hétérogènes provenant de multiples sources de données pouvant être disparates. L'une de ces solutions est la médiation.

La médiation est née dans les années quatre-vingts lorsque le développement des réseaux donna accès à de nombreuses sources d'information, et suscita l'envie de pouvoir les interroger de façon uniforme.

Le présent chapitre vise à décrire les principes de cette solution.

2.2 Description générale de la médiation

Les médiateurs sont définis comme des modules qui occupent une couche explicite et active entre les applications et leurs ressources [26]. Le terme médiateur est défini aussi de manière générale comme un composant logiciel assurant la médiation entre un utilisateur et un ensemble de ressources hétérogènes et distribuées [38]. Une ressource peut être dans notre esprit un fichier ou un ensemble de fichiers de données ou de code programme, une base de données, un programme, etc.

L'architecture de médiation fut conçue en 1992 par Gio Wiederhold dans l'article fondateur "Mediator in the architecture of future information systems" [61]. Wiederhold y développe une nouvelle vision de l'architecture du traitement de l'information en entreprise. Il tente de régler la problématique de l'accès et de l'intégration de l'information en introduisant la notion de médiateur comme suit : "*A mediator is a software module that exploits encoded knowledge about some sets or subsets of data to create*

information for a higher layer of applications."

Dans un contexte de médiation, les ressources pré-existent, avec leurs données mais aussi leurs propres métadonnées, et le but d'un médiateur est de réconcilier certaines de ces métadonnées de façon à donner l'illusion à un utilisateur qu'il interroge (ou navigue dans) un système d'informations homogène [38]. La médiation consiste à rendre le plus transparent possible l'échange requête-réponse avec le système médiateur. L'utilisateur n'a qu'à se préoccuper de formuler la requête de son choix (navigation ou interrogation), le système doit lui retourner un résultat intelligible. La médiation présente donc l'intérêt de pouvoir construire un système d'interrogation de sources de données sans toucher aux données qui restent stockées dans leurs sources d'origine [52].

2.3 Les services offerts par une architecture de médiation

Une architecture de médiation offre deux services principaux (figure 2.1 [42]) ; ces deux services sont entre autre :

- **L'exportation de données sources :** En fait, comme il vient d'être expliqué le médiateur ne dispose pas des données réelles mais seulement de leur description abstraite sous la forme de formules logiques appelées des vues.
- **L'importation de données :** Cela consiste à évaluer par le médiateur une requête décrite aussi en terme de vues pour fournir des réponses. Afin de permettre cette évaluation, le médiateur s'appuie sur la caractérisation des données sources, et ainsi garantir par exemple de ne produire que des réponses certaines, ou toutes les réponses certaines, selon les critères de qualité exigés ou désirés.

Dans le cadre de la gestion de compétences, il s'agit d'exploiter les services offerts par une telle architecture de médiation pour :

- **L'exportation de compétences :** Offrir à une entité qu'on appelle exportateur des possibilités à expliciter et à décrire ces compétences par le langage de représentation de compétences adopté par le système de médiation.
- **L'importation de compétences :** Permettre à d'autres entités appelées importateurs d'exploiter les compétences exportées.

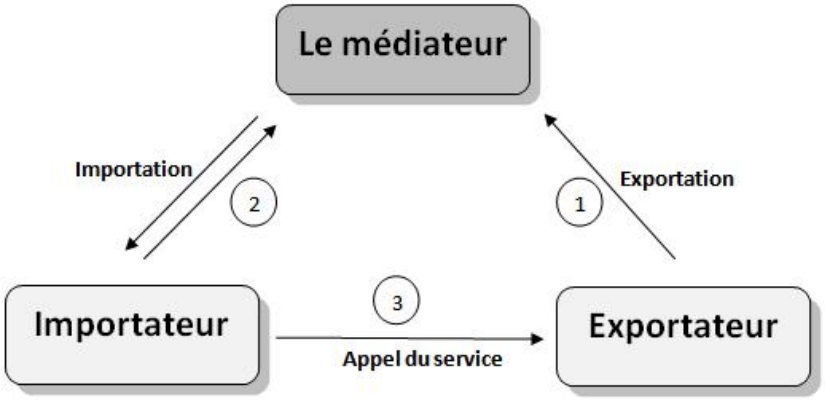


FIG. 2.1 – Les services d'une architecture de médiation

2.4 Les approches de médiation

2.4.1 L'approche centralisée

Un médiateur est spécifique à un domaine d'application donné. Il comprend un schéma global, ou ontologie, dont le rôle est central. Il fournit un vocabulaire structuré servant de support à l'expression des requêtes. Par ailleurs, il établit une connexion entre les différentes sources accessibles.

Dans l'approche de médiation centralisée, l'intégration d'information est fondée sur l'exploitation de vues abstraites décrivant de façon homogène et uniforme le contenu des sources d'information dans les termes de l'ontologie. Ces vues sont définies par des modules spéciaux appelés des adaptateurs. Les sources d'information pertinentes pour répondre à une requête, sont calculées par réécriture de la requête en termes de ces vues.

Le principe d'une telle architecture est présenté dans la figure 2.2 [53].

2.4.2 L'approche distribuée

2.4.2.1 Les limites de la médiation centralisée

L'arrivée massive sur les réseaux de nouveaux utilisateurs et de leurs données fait apparaître de nouveaux besoins d'intégration d'informations. Le plus médiatique actuellement est l'échange de fichiers multimédia (musique, vidéos, etc.) entre internautes [52]. Pour ces nouvelles applications, la médiation centralisée n'est pas adaptée car la centralisation de l'accès à un tel volume de données offre des performances faibles. Les médiateurs centralisés peuvent intégrer efficacement les données de quelques grandes

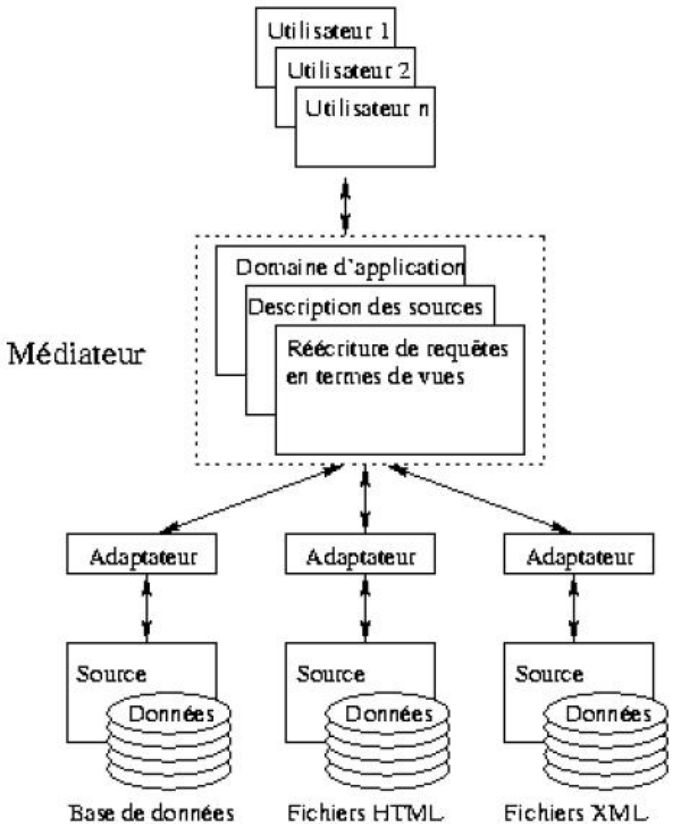


FIG. 2.2 – Architecture de médiation

entreprises, mais pas celles de millions d'utilisateurs. Pour ces besoins nouveaux, une fédération de médiateur[1] est en passe d'émerger avec l'apparition des systèmes pair-à-pair (peer-to-peer).

Les systèmes pair-à-pair sont composés des n oeuds sur un réseau P2P[2], ces n oeuds sont appelés des pairs[30]. Chaque pair peut jouer indifféremment le rôle de serveur de données ou de médiateur avec les autres pairs, pour participer de manière distribuée et collaborative au calcul de réponses à des requêtes. Une requête est posée à un pair qui y répond à l'aide des données qui lui sont directement accessibles ou dont il sait qu'elles sont accessibles via un autre serveur.

L'intérêt d'étudier le calcul des réponses à une requête dans les systèmes pair-à-pair permettra de faire passer à l'échelle la taille des données pouvant être intégrées dans les applications contraintes qualitativement, aujourd'hui limitées à la capacité d'intégration des médiateurs (systèmes d'information, sites de e-commerce ou encore les portails internet/intranet)[53].

[1]La fédération de médiateurs est appelée dans certains documents la médiation pair-à-pair ou une médiation distribuée.

[2]P2P est l'abréviation donnée à peer to peer.

2.4.2.2 Définition de la fédération de médiateurs

La fédération de médiateurs désigne un groupe de médiateurs qui acceptent d'être considérés comme une entité unique par d'autres médiateurs quand ceux-ci demandent un service à la fédération [26]. En d'autres termes, un importateur peut envoyer une demande à un médiateur de la fédération et celle-ci peut être transmise à n'importe lequel ou à chaque médiateur de la fédération. La fédération fera coopérer les serveurs de médiation indépendants, traduire les langages de représentation des médiateurs et intégrer les résultats en une réponse unique homogène.

2.4.2.3 Architecture conceptuelle pour les médiateurs fédérés dans le cadre de la gestion de compétences

Une architecture à base des médiateurs fédérés peut être utilisée pour la gestion de compétences (figure 2.3 [16]). Le principe de cette architecture de médiation est le suivant [16] [15] :

1. Une entité appelée exportateur publie ses compétences auprès d'un ou de plusieurs médiateurs (arc (a) sur la figure 1).

2. Des entités appelées importateurs adressent au médiateurs des demandes de recherche d'exportateurs dotés de certaines compétences, à charge pour lui d'essayer de trouver les exportateurs satisfaisant la demande (arc (b) et (c) de la figure 2.3).

3. Pour fournir une réponse à cette demande, le médiateur se fond sur les descriptions des sources qui lui ont été adressées ainsi que sur les relations entre elles, des relations qu'il aura lui-même établi. En cas de la satisfaction de la demande, le médiateur adresse à l'importateur les références des exportateurs la satisfaisant. Dans le cas échéant, un médiateur peut entamer un processus de coopération avec les autres médiateurs de la fédération afin de tenter de satisfaire la demande (arc (d) de la figure 2.3).

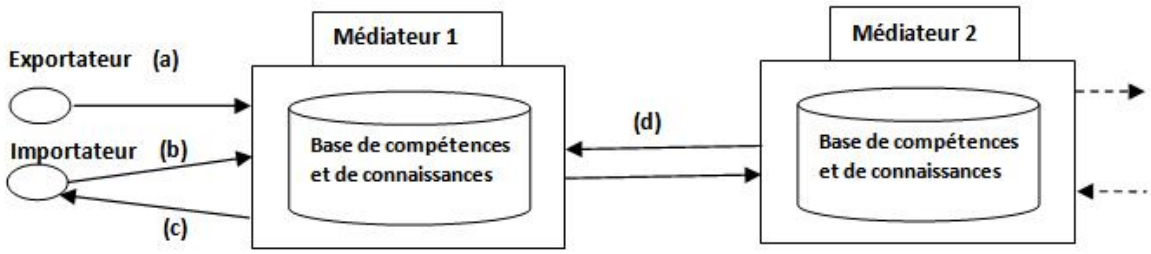


FIG. 2.3 – Une architecture à base des médiateurs fédérés pour la gestion de compétences

2.4.2.4 La satisfaction de la recherche des offreurs de compétences dans une fédération

La satisfaction d'une demande présente plusieurs cas de figure [16] [15] :

1. Il existe des exportateurs satisfaisant totalement la demande.

2. Il existe des exportateurs satisfaisant partiellement la demande mais en combinant les compétences de plusieurs exportateurs, la demande peut être totalement satisfaite.

3. Il n'existe pas d'exportateurs ni plusieurs exportateurs qui satisfont totalement la demande. Dans ce dernier cas, une coopération entre les médiateurs est requise. La profondeur de la fédération peut être contrôlée et contrainte par la demande. Par exemple, si la requête consiste à rechercher au plus deux individus satisfaisant un ensemble de compétences, dans ce cas la réponse ne peut inclure que de la composition de deux exportateurs s'il n'existe pas un seul qui la satisfait totalement.

2.4.2.5 La propagation de requêtes dans une fédération

Dans le cadre de la fédération, si une requête n'est pas satisfaite localement, un médiateur peut propager la partie non satisfaite de la requête à un autre médiateur, cela peut être fait selon deux critères [16] [15] :

1. Selon un critère sémantique : en confrontant la requête au contexte qu'il gère, le médiateur identifie les médiateurs les plus appropriés pour satisfaire la requête.

2. Selon un critère de proximité : la requête est propagée vers des médiateurs déjà identifiés.

Un algorithme de propagation peut évidemment combiner les deux types de critères.

2.5 Conclusion

Dans ce chapitre, nous avons présenté l'une des approches d'intégration des sources de données distribuées et hétérogènes qui est la médiation dans le cadre de la gestion de compétences. Deux approches de médiation ont été discutées, à savoir, l'approche centralisée et l'approche distribuée.

De notre part, nous avons opté pour l'approche distribuée en utilisant une architecture à la base des médiateurs fédérés pour mettre en oeuvre notre solution au problème de la gestion de compétences. Ce choix est justifié par le fait que nous estimons que nos propositions peuvent trouver leurs applications dans multiples domaines, et plus particulièrement dans des applications à grand échelle comme le web où la quantité d'information ne cesse à augmenter. Dans ce type d'applications la médiation centralisée n'est pas adaptée, une fédération des médiateurs semble être une première voie de solution pour le passage à l'échelle.

Le prochain chapitre est réservé à la présentation de notre proposition.

Proposition : Les graphes conceptuels pour la gestion de compétences

Chapitre 3

Les graphes conceptuels pour la gestion de compétences

3.1 Introduction

Dans notre proposition, nous nous intéressons aux formalismes basés sur les graphes. Par formalismes basés sur les graphes nous entendons les formalismes dans lesquels non seulement les connaissances sont représentées par des graphes mais les mécanismes de raisonnement sont également réalisés par des opérations sur les graphes. Nous nous inspirons évidemment du formalisme de représentation de connaissances des graphes conceptuels qu'on va adapter pour la gestion de compétences.

Le présent chapitre détaille notre contribution. Avant de la décrire, nous commençons tout d'abord par une présentation de quelques notions de base sur le formalisme des graphes conceptuels nécessaires à sa compréhension.

3.2 Notions de base

3.2.1 Rappel du formalisme des graphes conceptuels

Les graphes conceptuels sont un modèle de représentation de connaissances du type réseaux sémantiques qui a donné lieu à un certain nombre de travaux depuis son introduction par John F. Sowa en 1984 [28].

L'une des particularités de ce modèle est de permettre de représenter des connaissances sous forme graphique. Plus précisément, un graphe conceptuel est un graphe biparti étiqueté, les deux classes de sommets étant étiquetés respectivement par des noms de " concepts " et des noms de " relations conceptuelles" entre ces concepts.

Une telle représentation graphique des connaissances permet à des utilisateurs de comprendre, créer ou modifier directement des objets de ce type, de façon beaucoup plus simple qu'avec une représentation sous forme de formules logiques.

La facilité d'utilisation du modèle est renforcée par une séparation explicite de différents types de connaissances. Ces derniers sont en effet représentés de différentes façons par des objets distincts et cette structuration entraîne une plus grande clarté lors de l'utilisation de ce modèle pour la représentation de connaissances. Le vocabulaire permettant de représenter l'ontologie du domaine est notamment structuré dans un objet du modèle appelé "support" qui permet de représenter de façon simple des liens de spécialisation/généralisation.

L'autre intérêt du modèle vient du fait que des raisonnements peuvent être effectués sur les connaissances représentées. Ces raisonnements peuvent être vus comme des opérations de graphes et peuvent ainsi se baser sur les travaux d'algorithmique de graphes, puisque les graphes conceptuels sont des graphes étiquetés. Enfin, le modèle est muni d'une sémantique en logique du premier ordre qui est adéquate[1] et complète[2] par rapport à la déduction [48].

Plusieurs extensions et opérations ont été définies sur les graphes conceptuels [8] [21] [22] [10]. Nous citons dans les sections suivantes celles qui sont indispensables pour la compréhension de la suite de ce rapport. Nous proposons pour chacune de ces notions un exemple approprié pour simplifier leur compréhension.

3.2.2 La projection

La projection [28] [19] est l'opération de base sur les graphes conceptuels, elle permet de déterminer si les informations représentées par un graphe induisent les informations représentées par un autre graphe. La projection est adéquate [28] est complète [46] par rapport à la déduction en logique du premier ordre.

Formellement, la projection est définie comme une application "π" des n oeuds d'un graphe "H" vers les n oeuds d'un graphe "G" tel que [58] :

– pour chaque concept "c" de "H", π(c)[3] est soit une spécialisation de c, soit iden-

[1]Soit K une base de connaissances et f une application de K dans un ensemble de formules d'une certaine logique. Soit P une procédure (ou algorithme, ou ensemble de règles, ...) permettant de faire des inférences. P est adéquat (par rapport à f), si toute inférence faite par P correspond à une déduction, i.e. "si une expression i est inférée de K, alors f(i) est déductible de f(K)".

[2]P est complet (par rapport à f) si à toute déduction dans le langage logique cible correspond une inférence de P, i.e. "si, pour une expression i, f(i) est déductible de f(K), alors i peut être inférée de K "

[3]π(c)est appelé projection ou image du concept c.

tique à c.

- pour chaque relation r de H, "π(r)"[4] est soit une spécialisation de r, soit identique à r.
- si le i ème arc de "r" est lié à un concept "c" dans "H", alors le i ème arc de "π(r)" doit être lié à "π(c)" dans "G".

Une projection d'un graphe "H" sur un graphe "G" n'est pas toujours unique : le graphe "G" peut avoir plusieurs sous-graphes dont il existe une projection de "H" sur ce sous-graphe vérifiant la condition de la projection.

On dit que "G" est une spécialisation de "H" ou que "H" est une généralisation de "G" s'il existe une projection de "H" vers "G". De même, nous pouvons dire qu'un graphe "G" est une spécialisation d'un graphe "H" s'il existe au moins autant d'information dans "H" que dans "G".

La Figure 3.1 montre un exemple de projection d'un graphe "H" sur un autre graphe "G".

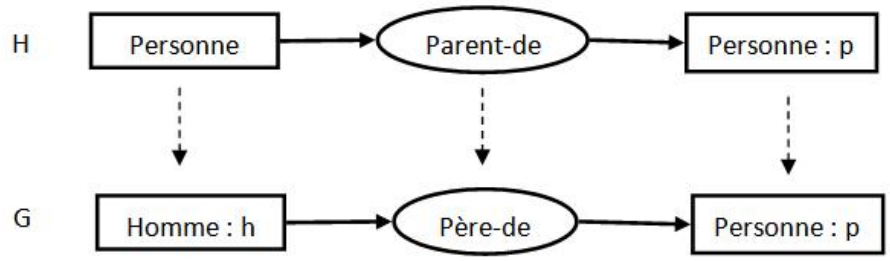


FIG. 3.1 – Exemple de projection de graphes conceptuels

3.2.3 La normalisation

La normalisation [44] est l'opération qui rend un graphe sous forme normale.

Un graphe sous forme normale respecte une structure où les référents sont uniques : elle est obtenue en fusionnant les n oeuds concepts ayant le même marqueur individuel. Ainsi, lorsqu'on utilise la forme normale, il ne peut exister deux appellations différentes pour référer à une même instance d'un certain concept dans un même graphe [25].

La forme normale d'un graphe évite des ambiguïtés sémantiques et logiques dans les graphes conceptuels. Sans la forme normale, deux graphes peuvent avoir le même sens et la même interprétation logique sans avoir le même graphe qui les représente [58].

[4] "π(r)" est appelé projection ou image de la relation "r".

D'une manière formelle, considérons "H" un graphe conceptuel, et "C" l'ensemble de ces concepts. "H" est en forme normale si pour n'importe quel couple de concepts "c1" et "c2" $\in$ "C", référent(c1)$\neq$référent(c2).

Un exemple de normalisation est donné dans la figure 3.2. Dans cet exemple nous fusionnons les deux concepts ayant le même marqueur individuel (h) dans le grahe "H" pour obtenir le graphe normalisé "G".

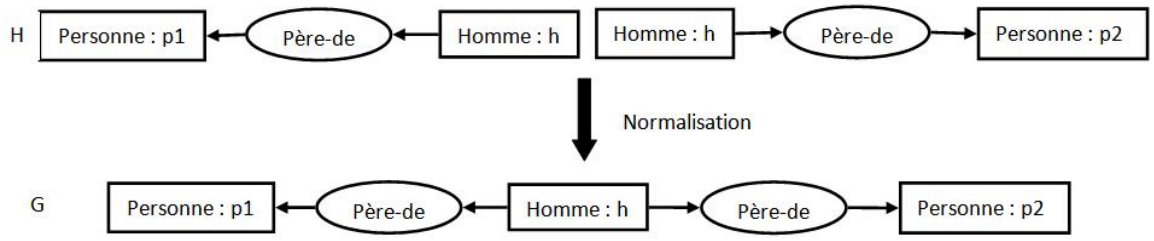


FIG. 3.2 – Exemple de normalisation d'un graphe conceptuel

3.2.4 La somme disjointe de deux graphes conceptuels

La somme disjointe de deux graphes est le graphe obtenu en mettant deux copies des deux graphes d'origines en juxtaposition [20].

Un exemple d'une somme disjointe de deux graphes est présenté dans la figure 3.3.

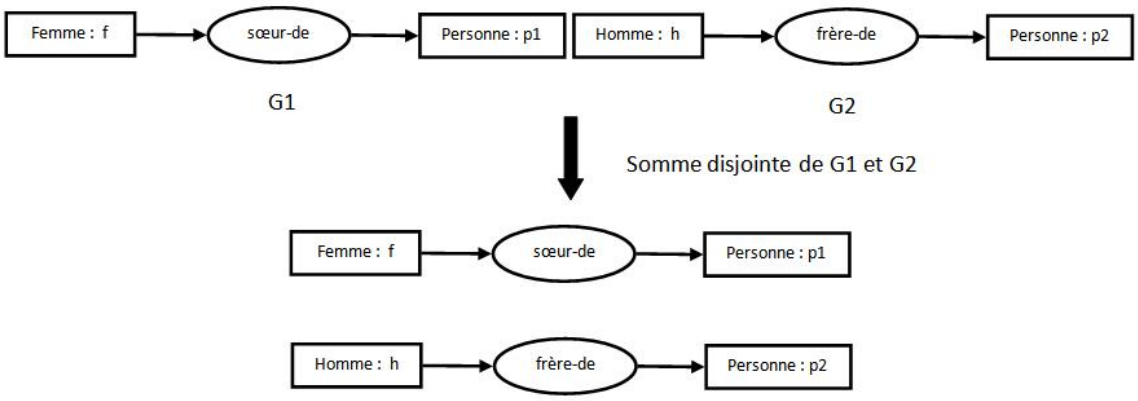


FIG. 3.3 – Exemple de somme disjointe de deux graphes conceptuels

3.2.5 Les graphes irredondants

D'une manière simple, un graphe irredondant est un graphe qui ne contient pas de l'information superflue. Il est défini selon les auteurs des travaux [19] [45] comme suit :

"Un graphe conceptuel est irredondant si et seulement s'il ne se projette pas dans l'un de ses sous graphes stricts [5]".

Exemple : Le premier graphe de la figure 3.4 est redondant. En effet la présence de l'information " Le père p est un père d'une certaine personne" est suffisante, et l'information "Le père p est un parent de la même personne" est additionnelle.

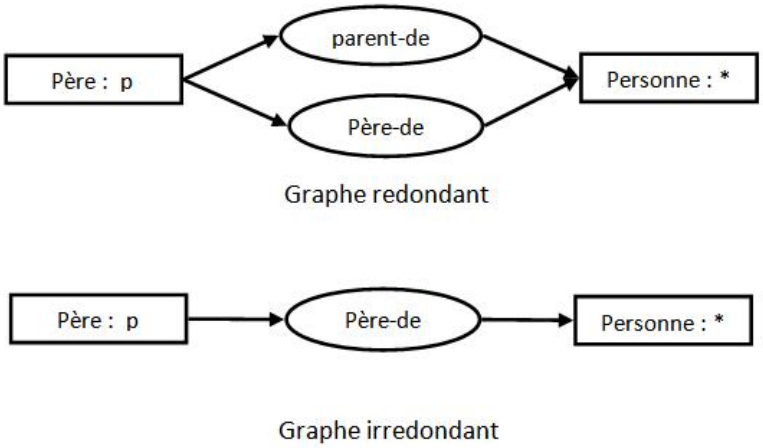


FIG. 3.4 – Exemple de graphes redondants et de graphes irredondants.

3.2.6 Headed graph

Un graphe conceptuel est de type headed graph s'il dispose d'un certain n oeud choisi comme une tête sémantique [48].

Dans l'exemple 3.5, la tête du graphe est dessiné en rouge.

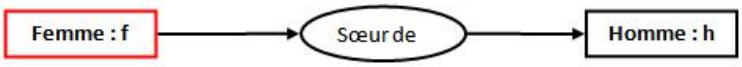


FIG. 3.5 – Exemple de headed graph

3.2.7 Les règles de graphes

Les règles de graphes sont introduit pour la première fois par E.Salvat [54] comme une extension des graphes conceptuels, proposée pour exprimer des connaissances de type "Si G1 alors G2" ou "G1⇒G2" tel que "G1" et "G2" sont des graphes conceptuels.

[5]Un sous-graphe strict d'un graphe conceptuel "G" est un sous-graphe de "G" qui n'est pas isomorphe à "G".

Formellement, une règle de graphes est constituée d'un graphe hypothèse "G1", d'un graphe conclusion "G2", et d'un ensemble de points d'attache correspondant à des liens de connexion entre "G1" et "G2".

La figure 3.6 représente un exemple de règle de graphes exprimant ceci : "Si une personne est le parent d'un homme x et d'une personne y alors x est le frère de y".

Dans cet exemple les marqueurs *x et *y sont des définitions des variables tandis que ?x et ?y représentent les références aux variables définies. Ces variables sont définies pour représenter les liens de connexion entre les deux graphes hypothèse et conclusion.

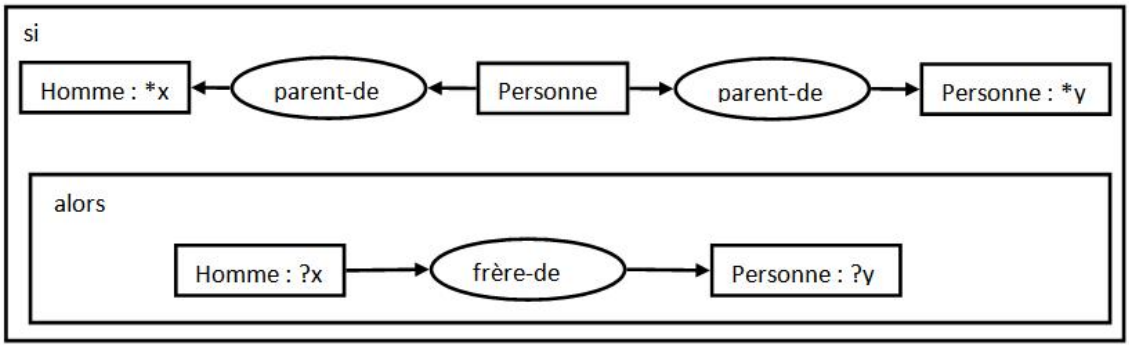


FIG. 3.6 – Exemple de règles de graphes

Des opérations de graphes permettant d'utiliser des règles de graphes conceptuels sont également définies dans des mécanismes de chaînage avant et de chaînage arrière. Dans les deux cas, la déduction est basée sur l'opération de projection et elle est prouvée adéquate et complète par rapport à la déduction en logique du premier ordre [55].

3.3 Les graphes conceptuels pour la gestion de compétences

Dans cette section nous présentons l'utilisation que nous faisons des graphes conceptuels pour la résolution de notre problème. Par compétence, nous entendons ici les capacités (savoir-faire ou services) d'une entité (un objet, un composant logiciel, un partenaire industriel, un service web, etc.).

3.3.1 Gestion de compétences

3.3.1.1 La représentation de compétences

La représentation des compétences est fondée sur un modèle des graphes conceptuels. Les offreurs de compétences sont représentés par des concepts et les compétences sont représentées par les relations attachées à ces concepts. D'autres informations sur les compétences peuvent être exprimées en terme d'autres concepts et d'autres relations entre ces concepts.

Exemple : Dire qu'un programmeur appelé x a la compétence pour programmer en un langage de programmation appelé JAVA, peut être exprimé en terme de graphes conceptuels (sous sa notation linéaire) de la façon suivante :

[Programmeur : x]->(programmer)->[Langage-programmation :JAVA].

Cependant, le modèle des graphes conceptuels simples ne permet pas une représentation adéquate des entités et des relations d'un domaine. En effet la représentation des types dans le modèle de base des graphes conceptuels est pauvre et manque de l'expressivité afin de décrire des information génériques sur ces types [37].

Exemple : Considérons les hiérarchies des types de concepts et de relations présentées dans la figure (3.7).

Dans cet exemple, le sens du type de concept "Mère" est donné juste par rapport à sa place dans la hiérarchie de type de concept, la seule information qu'on peut déduire est le fait que mère est une spécialisation du type de concept "Femme", mais aucune information n'indique par exemple qu'une femme ayant la compétence pour être mère d'une personne est une mère. De même, le sens de la relation être grand-parent est donné juste par rapport à sa position dans la hiérarchie des relations, et aucune information n'indique que la composition des deux relations "parent-de" est équivalente à une relation "grand-parent-de".

Pour remédier à ce problème, nous proposons dans ce travail d'utiliser les règles de graphe[31] comme il est expliqué dans les sections suivantes.

1. **Les règles de graphes pour la définition des types de concepts :**

   Nous définissons un type de concept par les "conditions nécessaires et suffisantes" ou les "conditions nécessaires" pour appartenir à ce type.

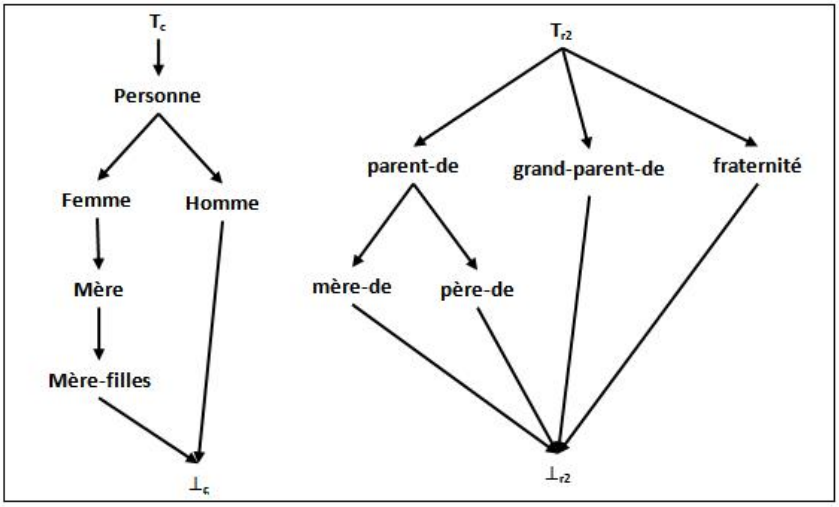


FIG. 3.7 – Exemple de support

Exemple : Définition du type de concept "Mère" par les deux règles :

R1 : [Mère :*x]⇒[Femme : ?x]->(mère-de)->[Personne]
R2 : [Femme :*x]->(mère-de)->[Personne]⇒[Mère : ?x].

Exemple 2 : Définition de type de concept "Mère-filles" par les règles suivantes :

R1 : [Mère-filles :*x] ⇒ [Mère : ?x]
R2 : [Mère-filles :*x]->(mère-de)->[Personne :*y] ⇒[Femme : ?y]

Bien que ce dernier exemple ne permet que de représenter la partie "conditions nécessaires" et non la partie inverse (i.e. conditions suffisantes pour être une Mère-filles), cette représentation nous aide à déduire certaines informations.

Exemple : Grâce à la définition du concept "Mère-fille" par les deux règles de l'exemple précédent, en disposant du graphe suivant :
[Mère-filles :x]->(mère-de)->[Personne :y], nous pouvons déduire le graphe : [Femme :y] (c.à.d. que y est un concept de type Femme).

2. **Les règles de graphes pour la définition des types de relations :**
   De la même façon qu'un type de concept, nous définissons un type de relation par les "conditions nécessaires et suffisantes" ou les "conditions nécessaires" pour appartenir à ce type.

Exemple : Définition de type de relation "grand-parent-de" par les deux règles suivantes :

R1 : [Personne :*x]->(grand-parent-de)->[Personne :*y]⇒
[Personne : ?x]->(parent-de)->[Personne :*]->(parent-de)->[Personne : ?y].

R2 : [Personne :*x]->(parent-de)->[Personne :*]->(parent-de)->[Personne :*y]⇒
[Personne : ?x]->(grand-parent-de)->[Personne : ?y].

3. **Les méta-connaissances sur les relations :**

Nous désignons par les méta-connaissances sur les relations, certaines propriétés des relations tel que les propriétés algébriques. Pour exprimer ces propriétés, nous utilisons toujours le même mécanisme de règles de graphes.

Exemples :

(a) Représentation du fait que la relation de "fraternité" est transitive :

R1 : [Personne :*x]->(fraternité)->[Personne :*]->(fraternité)->[Personne :*y]⇒
[Femme : ?x]->(fraternité)->[Femme : ?y]

(b) Représentation du fait que les deux relations "parent-de" et "enfant-de" sont symétrique entre elles :

R1 : [Personne :*x]->(parent-de)->[Personne :*y]⇒
[Personne : ?y]->(enfant-de)->[Personne : ?x]

R2 : [Personne :*x]->(enfant-de)->[Personne :*y]⇒
[Personne : ?y]->(parent-de)->[Personne : ?x]

3.3.1.2 L'organisation de la base de compétences et de connaissances

La base de connaissances et de compétences suivant notre modèle occupe deux niveaux de représentation ; (1)le niveau ontologique, qui comprend la définition du vocabulaire conceptuel du domaine visé et des connaissances générales sur ce domaine en terme de règles, et (2)le niveau factuel (ou assertionnel), qui comprend des graphes étiquetés définis sur le support et dédiés à la représentation des compétences acquises.

1. Le niveau ontologique :

- **La hiérarchie de types de concepts :** Elle est composée des types atomiques ainsi que des types définis. Ces types de concepts sont organisés suivant la relation de spécialisation/généralisation. Pour les types atomiques la relation de spécialisation/généralisation est simplement donné par l'utilisateur qui est supposé être un expert de domaine de représentation. Pour les types définis, un test de subsomption doit être fait afin de décider de leurs emplacement dans la hiérarchie.

- **Les hiérarchies de types de relations :** Le support contient aussi une ou plusieurs hiérarchies organisant les types de relations selon la relation de spécialisation/généralisation. Les types de relations peuvent être atomiques ou définies. Comme pour les concepts atomiques, la relation de spécialisation/généralisation entre les types de relations atomiques est donnée par l'utilisateur, par contre un test de subsomption doit être fait pour organiser les types de relations définies.
- **La base de règles :** Elle est constituée des règles utilisées pour exprimer les méta-connaissances sur les relations ainsi que celles utilisées pour définir les types.

Remarque : Ici on ne traite pas l'inférence dans le niveau ontologique, par contre on traite l'utilisation qu'on fait de cette structure pour la découverte des offreurs de compétences.

2. Le niveau factuel :

Une fois que le support est établi, on peut construire un graphe conceptuel cohérent par rapport à ce support qui décrit les entités du domaine ainsi que leurs compétences publiées. Ce graphe est appelé ici la base de compétences et est noté par "BC".

On rappelle qu'en point de vue architecture système, nous avons opté pour une architecture à base de médiateurs. Donc, la base "BC" est construite et mise à jour suite

à des publications succéssives. Les étapes de publication d'un ensemble de compétences décrit par un graphe conceptuel sont les suivantes :

1. **La première étape** consiste à ajouter les compétences publiées à la base de compétences. Nous utilisons à cet effet l'opération de la somme disjointe du graphe BC et du graphe à publier. Le graphe résultant représente la nouvelle base de compétences.

2. **La deuxième étape** est la normalisation du graphe résultant. Dans le cas de la découverte, nous voulons éviter les doublons pour minimiser la recherche et homogénéiser les connaissances. C'est pour cette raison que nous avons besoin de la forme normale.

3. Enfin, **le dernier traitement** est le processus d'application des règles.

Notons "Gp" le graphe représentant les compétences à publier. Les étapes de publication de "Gp" sont présentées dans la figure 3.8 avec un diagramme d'activités.

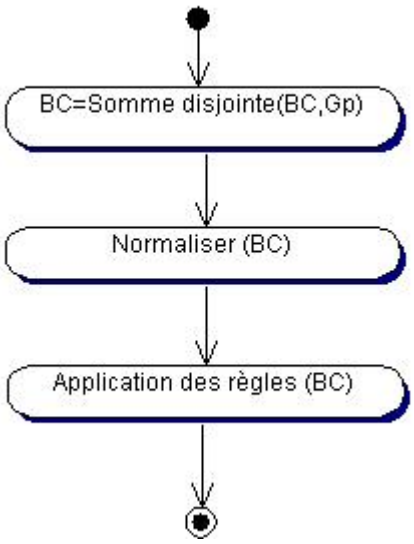


FIG. 3.8 – Procédure de publication de compétences

3.3.2 La découverte des offreurs de compétences

Il s'agit dans cette section de définir un mécanisme de recherche d'entités dotées de certaines compétences requises. Ce dernier doit pouvoir trouver des réponses composites s'il n y a plus des réponses exactes.

3.3.2.1 Représentation de la requête

La requête (R) est représentée par un graphe conceptuel de type "headed graph", tel que :

1. Le type des entités recherchées est représenté par le type de la tête (t) de R. Nous introduisons dans ce travail un marqueur spécial noté " ?" équivalent logiquement au marqueur "*" pour désigner cette tête de la requête.
2. L'ensembles des compétences requises sont représentées par les relations qui sont directement liées à t.
3. Le reste de R représente les conditions sur les compétences recherchées.

Exemple :

Une requête de recherche des programmeurs ayant la compétence pour programmer en un des langages de programmation, tel que ce langage supporte la notion de classe, peut être représenté comme suit :

[Programmeur : ?]->(programmer)->[Langage-programmation :∗]->(supporte)->[classe].

3.3.2.2 Satisfaction locale de la requête

Le processus de la satisfaction locale de R (voir la figure 3.9) suit les étapes suivantes :

1. Normalisation de R pour minimiser sa taille, et pour enlever les ambiguïtés sémantiques et logiques.
2. Supprimer de R toutes les composantes connexes qui ne contiennent pas la tête "t", parce qu'elles sont indépendantes des entités recherchées.
3. Projection de R sur BC.
4. Si au moin une projection est trouvée alors sélectionner les projections de t.

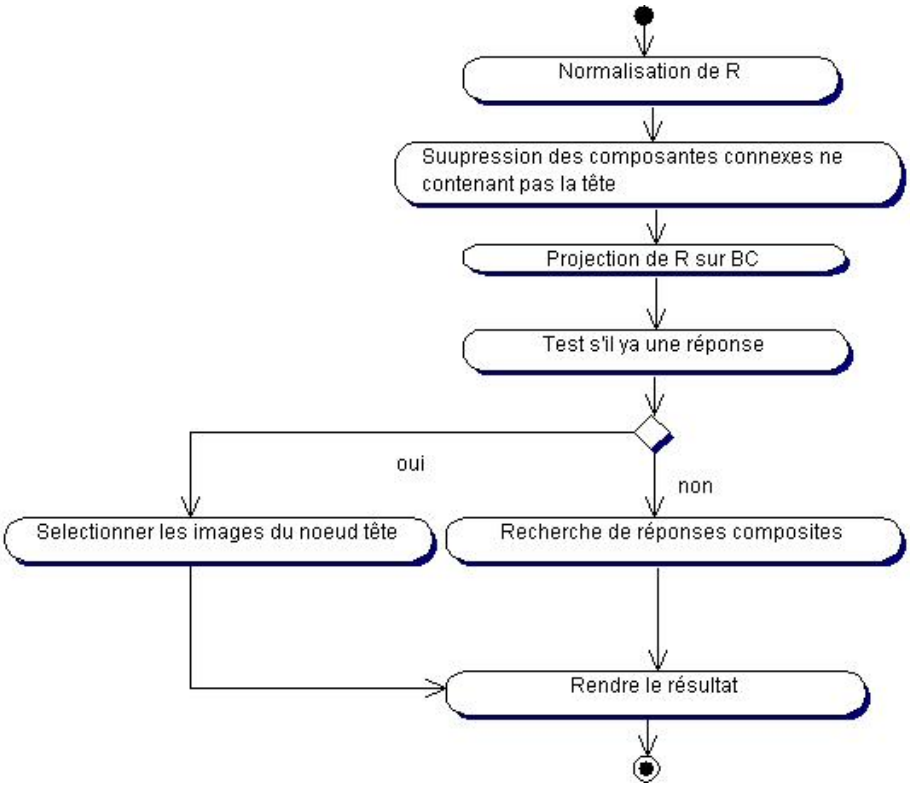


FIG. 3.9 – Procédure de la satisfaction locale d'une requête

3.3.2.3 Recherche des réponses composites

Si R n'est pas satisfaite localement, on procède à la recherche des réponses composites, i.e. de rechercher un ensemble d'entités dont l'ensembles des compétences acquises inclue l'ensemble des compétences requises. Pour ce faire, on suit les étapes suivantes (voir figure 3.10) :

1. Eclater R en n sous requêtes Ri (i=1—n), chacune d'entre elle représente une requête de recherche d'entités ayant une seule compétence. Dans cette étape, nous supposons qu'il n y a plus aucun arc liant le graphe conditions d'une des compétences avec les graphes conditions des autres compétences.
2. Satisfaire chacune des sous requêtes Ri (i=1—n).
3. Si toutes les sous requêtes Ri sont satisfaites, les réponses seront les combinaisons des réponses des sous requêtes Ri.

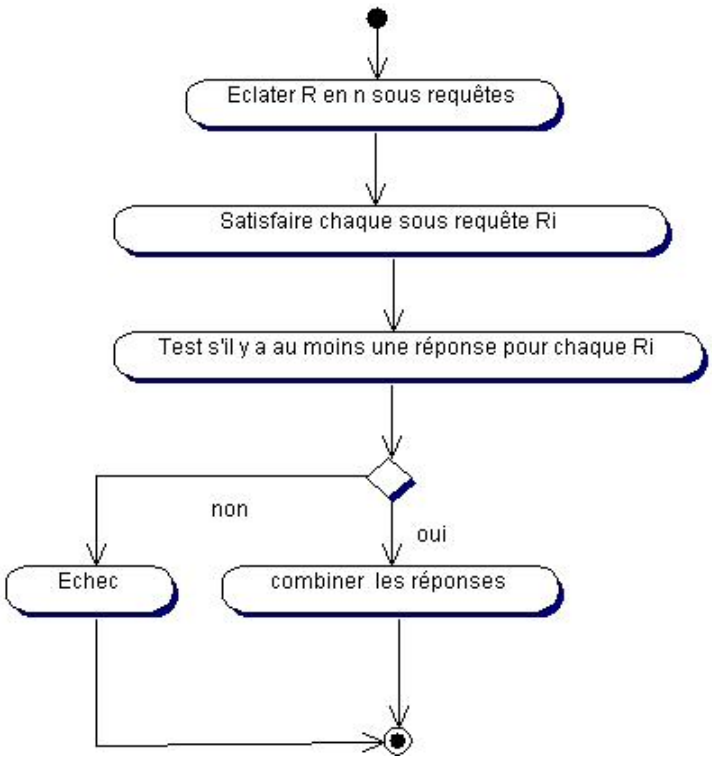


FIG. 3.10 – Procédure de recherche des réponses composites

Exemple :

Soit une requête R de recherche des personnes ayant les compétences pour programmer en JAVA et en C++. Celle-ci est formalisée comme suit :

[Personne : ?]->(programmer)->[Langage-programmation :JAVA]
——>(Programmer)->[Langage-programmation :C++]

Pour trouver les réponses composites, R est éclatée en deux sous requêtes qui sont :

R1 : [Personne : ?]->(programmer)->[Langage-programmation :JAVA]

R2 : [Personne : ?]->(programmer)->[Langage-programmation :C++]

La satisfaction d'une sous requête Ri suit le processus décrit dans la figure 3.11.

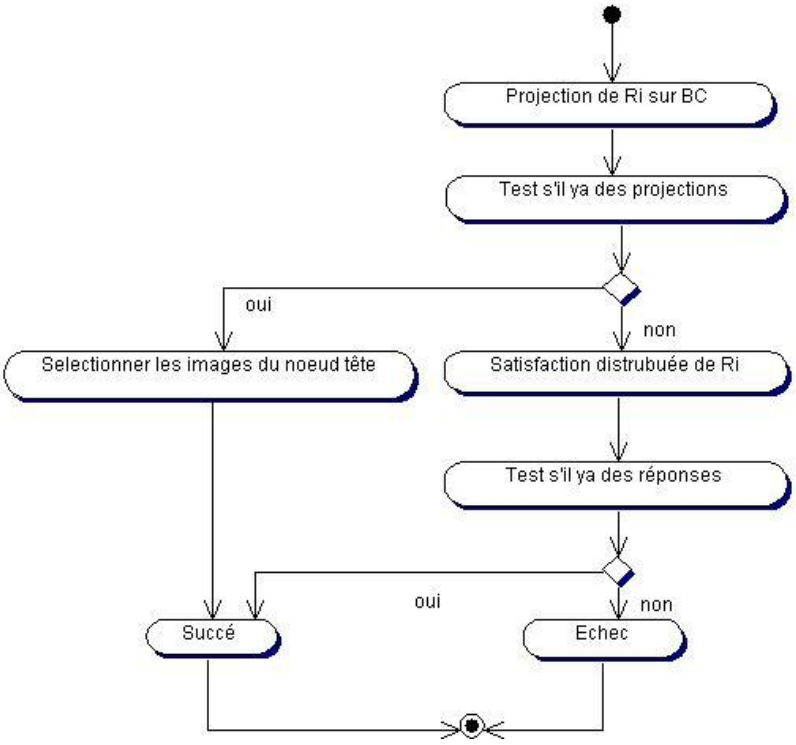


FIG. 3.11 – Procédure de satisfaction d'une sous requête

3.3.2.4 Satisfaction distribuée de la requête

Si une sous requête "Ri" n'est pas satisfaite localement au niveau d'un médiateur "M", ce dernier devient un initiateur de la fédération ; il construit alors une requête de fédération et il l'envoi au médiateur suivant.

Construction de la requête de fédération

La base de compétences et de connaissances étant distribuée, donc pour pouvoir construire une requête de fédération, il faut pouvoir distinguer la partie satisfaite et la partie non satisfaite de la sous requête Ri. Pour cela, nous proposons de :

1. Eclater le graphe requête en blocs de bases.
   Un bloc de base est un graphe conceptuel contenant une et une seule relation.
2. Garder les liens de jointures entre chaque couple de blocs de base.
   Un lien de jointure est un couple de concepts appartenant chacun à un graphe différent mais qui représentent la même entité. Les deux graphes auquels appartiennent ces deux entités peuvent être joints sur ces derniers.
3. Projeter chacun des blocs de base sur BC.

4. Chaque bloc de base auquel au moins une projection est trouvée peut représenter une partie satisfaite, et la partie non satisfaite sont tous les autres blocs dont les projections ne respectent pas les liens de jointures.

   Définition : Un ensemble de projections respectent un ensemble de liens de jointures, si et seulement si toutes ces projections peuvent être joints sur ces liens de jointures.

Exemple : Reprenons la requête suivante :

[Personne : ?]->(programmer)->[Langage-programmation :∗]->(supporte)->[classe]

Cette requête va être éclatée en deux blocs de base de la façon suivante :

g1 : [Personne : ?]->(programmer)->[Langage-programmation :∗]
g2 : [Langage-programmation :∗]->(supporte)->[classe]

Tout en gardant le lien de jointure ([Langage-programmation :*]de g1, [Langage-programmation :*] de g2).

La requête peut être affinée et la recherche peut être approfondie en remplaçant chaque bloc de base par un graphe équivalent, cette opération est appelée ici expansion.

Soit B un bloc de base. Pour trouver les graphes équivalents à B, nous procédons comme suit :

1. Chercher dans la liste des règles une règle "R" dont la conclusion est un bloc de base qui peut être projeté sur B.
2. Remplacer B par la partie hypothèse de R.

Exemple : Soit le graphe G, et la règle de graphe R suivants :

G : [Femme : f]->(grand-mère-de)->[Personne :p].
R : [Femme : *x]->(mère-de)->[Personne :*]->(parent-de)->[Personne :*y]
⇒ [Femme : ?x]->(grand-mère-de)->[Personne : ?y].

Ayant le graphe G comme bloc de base, et ayant la règle R, l'expansion de G, consiste à le remplacer par le graphe $\grave{G}$ suivant :
$\grave{G}$: [Femme : f]-(mère-de)->[Personne :*]-(parent-de)->[Personne :p].

La structure de la requête de fédération

Une requête de fédération suit une structure par niveaux, elle est construite de la manière suivante :

1. Le premier niveau d'ordre 0 contient la requête elle même.
2. Le niveau 1 contient les blocs de base de la requête accompagnés de leurs projections sur la base locale si elle est constituée de plusieurs blocs de base, sinon elle contient les blocs de base issus de la première expansion ainsi que leurs projections sur la base locale.
3. Le niveau n (n>1) contient la liste des blocs de base issus de la n-1^{ime} expansion si elle est constituée de plusieurs blocs de base ou la liste des blocs de base issus de la n^{ime} expansion dans le cas contraire ; accompagnés de leurs projections sur la base locale.

Le nombre de niveaux est fixé par un paramètre appelé profondeur, ce paramètre limite le nombre maximal d'expansions possibles pour un bloc de base de la requête.

La procédure de construction d'une requête pour la fédération est donnée dans la figure 3.12 tel que "p" représente le paramètre profondeur.

Exemple :

Un exemple de construction de requête pour la fédération est montré dans la figure 3.13 dans laquelle on représente les liens de jointures par des traits discontinus de couleur rouge.

La requête initiale est :
[Femme : ?]->(grand-mère)->[Homme :*]->(frère-de)->[Homme :h].

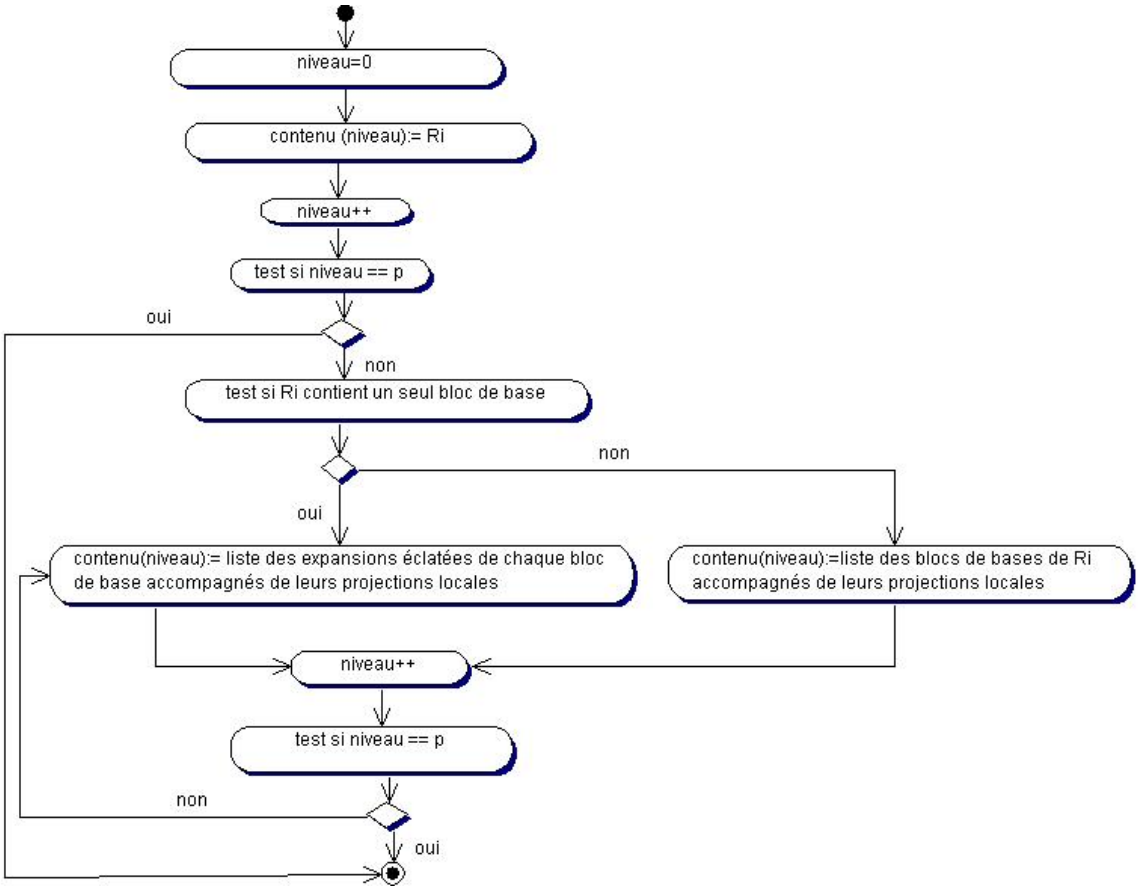


FIG. 3.12 – Procédure de construction d'une requête pour la fédération

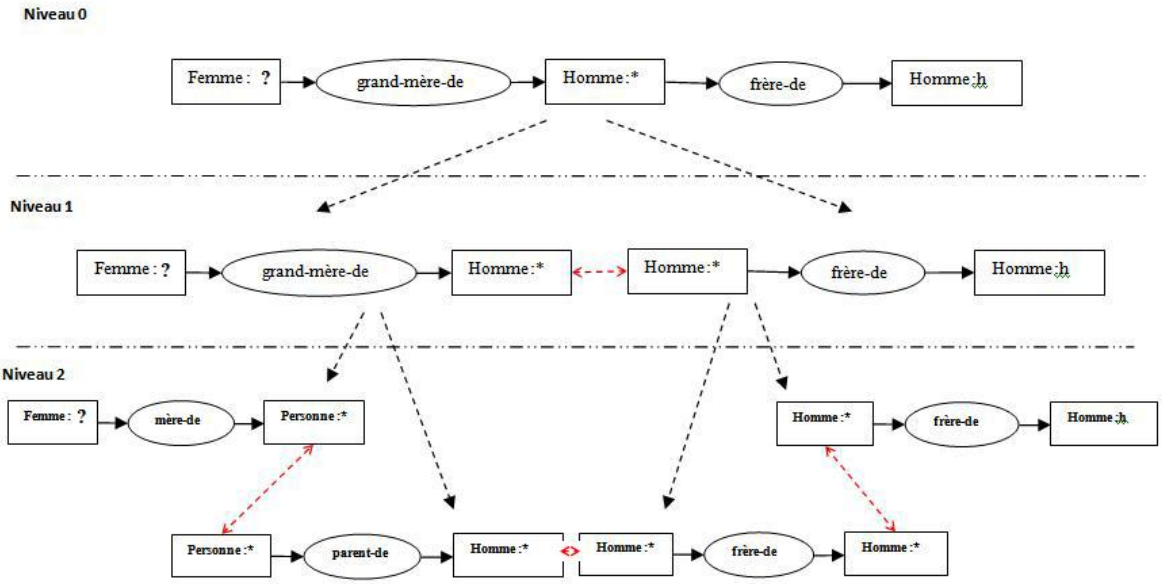


FIG. 3.13 – Exemple de construction d'une requête

Remarque

Il faut noter que suite à l'application des règles dans la phase de publication, le graphe BC peut être redondant, par conséquent les résultats de projections de blocs de base peuvent aussi être redondantes.

Définition : Les résultats redondants sont des blocs de base dont chaque couple de relations (r1, r2) appartenant chacune à un bloc de base différent vérifie les propriétés suivantes :

- Soit r1 et r2 sont identiques, soit l'une des deux est une spécialisation de l'autre.
- Ces relation relient les mêmes concepts avec le même ordre.

Pour minimiser la recherche, nous envisageons à supprimer tous les résultats de projections redondantes pour chaque bloc de base et ne laisser qu'une seule qui est la plus spécifique.

Traitement de la requête de fédération

La requête de fédération est traitée par ordre croissant de niveaux ; un niveau n'est traité que si la requête du niveau n-1 n'est pas satisfaite.

Le traitement d'un niveau n suit les étapes suivantes :

1. Projeter chacun des blocs de base sur la base locale.
2. Rajouter les résultats à la requête.
3. Vérifier si les liens de jointures sont respectés.

La procédure de traitement d'une requête de la fédération par un médiateur non initiateur est présentée par le diagramme d'activités de la figure 3.14 tel que n représente le niveau et p représente le paramètre profondeur.

La propagation de la requête dans une fédération

Dans ce travail, nous avons opté pour une propagation selon un critère de proximité, c.à.d. le médiateur suivant est fixé au départ d'une manière statique.

Le principe de résolution de requêtes est le suivant : un client soumet sa requête à un médiateur quelconque de la fédération, celui-ci essaye de répondre à la requête localement, s'il n'arrive pas à la satisfaire localement, il lance une procédure de satisfaction distribuée comme il a été expliqué dans les sections précédentes.

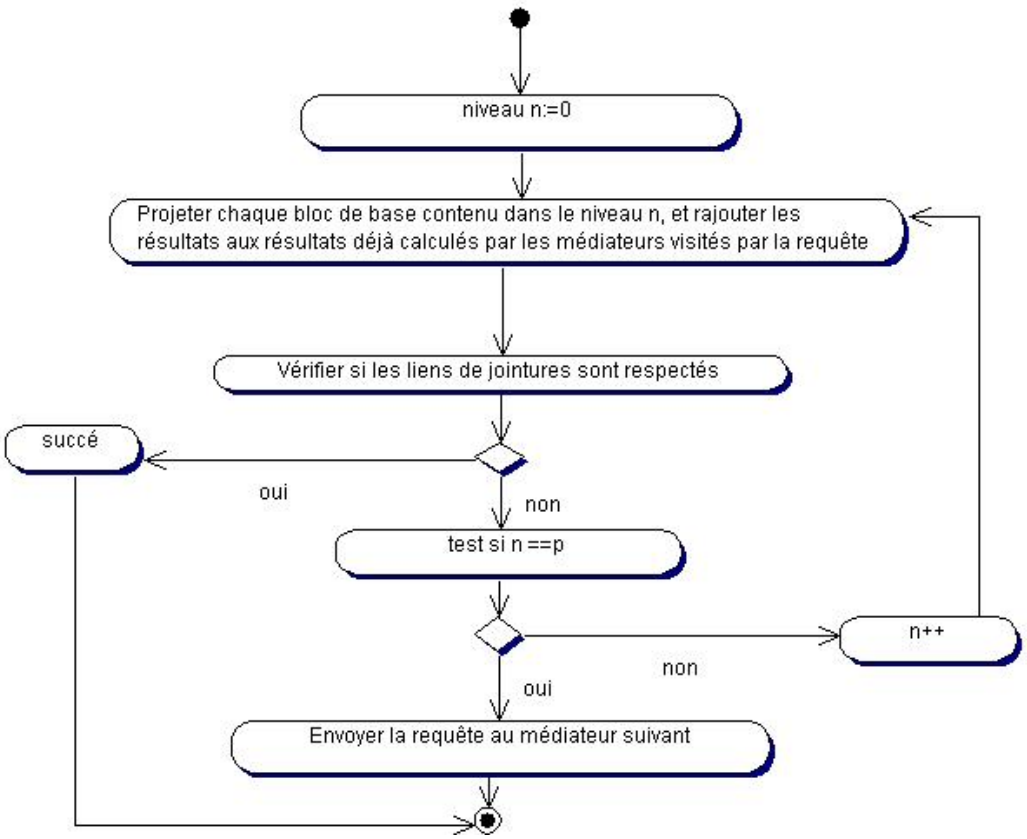


FIG. 3.14 – Traitement de la requête de fédération

3.4 Comparaison avec des travaux similaires

Dans ce travail nous nous somme focalisé sur le problème de la gestion de compétences. Notre objectif était de représenter formellement ces compétences et de les exploiter lors de la recherche d'entités dotées de certaines compétences requises.

Le sujet de la gestion de compétences est un axe qui émerge. Durant nos recherches sur ce sujet, on a trouvé des travaux qui traitent de la gestion de compétences des ressources humaines dans les entreprises [13], de la gestion de compétences des partenaires d'un domaine de recherche scientifique [59][60], de la découverte et composition de web services [47][40], etc.

Cependant, peu de travaux traitent ce problème indépendamment du domaine d'application, i.e. de se contenter de trouver une méthode plus générale possible qui soit instanciable pour divers domaines.

Dans cette dernière catégorie dont notre travail fait partie, on trouve les travaux [26][15][16] suivant une approche par objet et utilisant les logiques de description pour décrire les compétences. Le travail le plus avancé sur cette problématique dans cette

catégorie est celui de C.Dong [26].

Dans ce travail nous avons proposé une solution au problème de la gestion de compétences indépendamment du domaine d'application en utilisant le formalisme des graphes conceptuels :

1. En tèrme de représentation de compétences :
   - Dans le travail de C.Dong , la représentation des compétences est fondée sur les logiques de description. La compétence est représentée par les relations en terme de ce langage, et pour exprimer les relations de spécialisation/généralisation entre compétences, une extension des logiques de description par la hiérarchie de relations est définie.
   - Dans notre travail, nous avons adopté une représentation de compétences à base du formalisme des graphes conceptuels. Ce dernier en tant que formalisme de représentation de compétences présente quelques avantages par rapport aux logiques de description :
     (a) En terme de facilité d'utilisation, un graphe édité à l'écran par le biais d'une interface conviviale n'a pas, auprès d'utilisateurs non informaticiens, le côté rebutant que peuvent avoir les formules logiques des logiques de description.
     (b) La présence d'une hiérarchie de types de relations se prête bien à la représentation des relations de spécialisation/généralisation entre les compétences.
     (c) La distinction entre les relations de différentes arités nous permet de mieux gérer des compétences ayant des nombres d'arguments différents sans avoir besoin à incorporer d'autres méthodes ou mécanismes, à la différence des logiques de description qui ne traitent que des relations binaires.
2. En tèrme d'expressivité
   - En tèrme d'expressivité, le langage utilisé dans notre travail est peu expressif par rapport à celui présenté dans le travail de C.Dong. En effet, on ne peut pas exprimer la négation. Par exemple on ne peut pas exprimer le fait que le type de concept "Femme"' et la négation du type de concept "Homme". Notre langage n'exprime pas aussi d'une manière exacte l'opérateur $\forall$, il n'exprime que la partie nécessaire et non pas la partie suffisante (voir la section 3.3.1.1). D'un autre coté, notre travail permet de représenter des relations de différentes

arités à la différence du travail de C.Dong qui ne permet de représenter que des relations binaires.

3. En tèrme de découverte
   - Le travail de C.Dong s'appuie sur le processus de classification permettant de découvrir les relations de subsomption entre les concepts de la Tbox. Dans ce contexte, une requête de recherche de compétences est représentée par un concept Q, et satisfaire la requête revient à classifier Q dans la hiérarchie. Les réponses sont alors toutes les instances des concepts subsumés par Q qui sont nécessairement des instances de Q.
   - Dans notre travail, nous nous somme basé sur l'opération de projection. La requête est représentée par un graphe conceptuel. Satisfaire la requête revient à trouver ces projections sur le graphe base de compétences, et les réponses sont les images de la tête de la requête.

     Il faut noter qu'à la différence du travail de C.Dong, dans notre travail, plusieurs inférences sont faites lors de la publication, en effet, grâce à l'application des règles lors de la publication, notre base de compétences contient toutes les compétences qui sont explicitement publiées et aussi celle qui sont implicitement publiées. Cela peut minimiser énormément le temps de la recherche lors de la phase de découverte.

3.5 Conclusion

Dans ce chapitre, nous avons présenté l'approche proposée pour la résolution de notre problème.

Dans la première section nous avons fait un rappel du formalisme des graphes conceptuels. Ce formalisme décrit les connaissances d'une manière graphique expressive, aisée à manipuler et lisible.

Dans la deuxième section, nous avons décrit l'utilisation que nous faisons des graphes conceptuels pour résoudre le problème de la gestion de compétences. Nous avons adapté ce formalisme pour la représentation et le raisonnement sur les compétences. Du point de vue découverte, nous nous somme appuyé sur la notion d'éclatement de graphes et sur l'opération de projection ; la satisfaction d'une requête peut être simple ou composite, locale ou distribuée. Dans ce dernier cas, nous avons défini une structure par niveaux de requête pour la fédération.

Le prochain chapitre est réservé à la validation expérimentale de nos propositions.

Chapitre 4

Mise en oeuvre

4.1 Introduction

Pour mettre en oeuvre les idées présentées dans le chapitre précédent, nous avons implémenté un premier prototype conformément à une architecture client/serveur[1], nous permettant d'effectuer quelques expérimentations dont nous présentons les résultats dans ce chapitre.

4.2 Les outils de développement utilisés

Il existe un certain nombre d'outils qui implémentent les graphes conceptuels notamment pour une utilisation orientée recherche et extraction d'informations, nous citons : CoGITaNT[2], CoGui[3], Amine[36], CharGer[1], Notio[56] et bien d'autres. Cependant, très peu de ces outils offrent un environnement logiciel complet pour l'utilisation la plus large possible du modèle, le stockage et la manipulation d'un grand nombre de graphes.

Pour ces raisons, nous avons choisi de bâtir notre solution en utilisant le logiciel de création graphique de graphes conceptuels CoGui[3] et une bibliothèque de raisonnement permettant de faire des opérations de graphes CoGITaNT[2], sous le système d'exploitation linux. Et pour tester notre application sous windows, nous avons utilisé également une collection de logiciels appelée Cygwin[4] permettant au système windows d'émuler l'environnement unix.

[1] L'architecture client/serveur désigne un mode de communication entre plusieurs ordinateurs d'un réseau qui distingue un ou plusieurs clients du serveur.

4.2.1 CoGui

CoGui[2] est un outil visuel pour construire des bases de connaissances de graphes conceptuelles. Il permet de créer une base de connaissances, éditer sa structure et son contenu et de la contrôler. La base de connaissances peut être sérialisée en format XML[3] appelé CoGXML[4]. De plus, un mécanisme basique de question/réponse a été développé pour des buts éducatifs ; c'est-à-dire afin d'illustrer comment une base de connaissances peut être requise et exploitée. Pour les mécanismes de question/réponse complexes et d'autres tâches de raisonnement, CoGITaNT doit être utilisé.

4.2.2 CoGITaNT

4.2.2.1 Présentation de la bibliothèque

L'open source CoGITaNT[5] est un ensemble d'outils logiciels permettant le développement d'applications basées sur le modèle des graphes conceptuels, développé par l'équipe «Représentation de connaissances par des graphe » de LIRMM[6].

La bibliothèque CoGITaNT est un ensemble de classes C++ permettant une gestion facile des graphes conceptuels ainsi que les objets du modèle comme le support et les règles. Pour chaque objet du modèle est associé une classe dans CoGITaNT, et les structures de données utilisées sont des implémentations de base des objets du modèle (par exemple un graphe est un ensemble de n oeuds et d'arcs), de cette façon, il est possible à l'utilisateur connaissant le modèle des graphes conceptuels à comprendre la structure de CoGITaNT, et les extensions sont simplifiées.

4.2.2.2 Les formats de représentation supportés

CoGITaNT est l'outil accompagnant CoGui. Il propose plusieurs formats qui lui permettent de manipuler les graphes en lecture et en écriture. Les fichiers peuvent être au format propre BCGCT[7] ou au format CoGXML propre à CoGui.

1. Le format BCGCT est le format natif de CoGITaNT. Il permet de représenter sous une forme lisible les différents objets manipulés par la bibliothèque :

[2]Conceptual Graphs user interface.
[3]Extensible Markup Language
[4]Les graphes conceptuels au format XML
[5]Conceptual Graphs Integrated Tools allowing Nested Typed graphs
[6]Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier
[7]Base de Connaissances Graphes Conceptuels Textuelle

- Un support, formé d'un ensemble partiellement ordonné des types de concepts, des types de relations, des types d'emboitements[8], des signatures des types de relation et de la relation de conformité.
- Des graphes conceptuels (connexes ou non connexes, simples ou emboités).
- Et des règles.

2. Le format CoGXML permet de représenter des graphes conceptuels sous la forme de documents XML. Ce format standard permet ainsi d'échanger avec d'autres applications de manière assez aisée. En effet, le langage XML est un langage de représentation de données adapté pour décrire des données structurées. La DTD[10] qui a été définie permet la représentation des supports, des graphes et des règles sous une forme qui peut facilement être interprétée par des hommes et par des programmes.

4.2.2.3 Représentation d'un graphe conceptuel

Un graphe conceptuel est représenté comme un ensemble de n oeuds **(cogitant : :GraphObject)**, et un ensemble d'arcs **(cogitant : :Edge)**. Chaque objet du graphe est identifié d'une manière unique par un entier appelé le iSet **(cogitant : :iSet)**.

Les graphes conceptuels ainsi que les autres structures tel que les règles sont également identifiées par des identificateurs (iSet) uniques facilitant ainsi leurs manipulations.

4.2.3 Cygwin

Cygwin est une collection de logiciels libres à l'origine développés par Cygnus Solutions[11] permettant à différentes versions de Windows de Microsoft d'émuler un système Unix. Cygwin vise principalement l'adaptation à Windows de logiciels qui fonctionnent sur des systèmes POSIX[12] (tels que les systèmes GNU[13]/Linux et Unix), il tente de créer un environnement Unix sous Windows, rendant possible l'exécution de ces logiciels après une simple compilation.

[8]Les types emboités sont une extension des graphes conceptuels simples, cette notion n'est pas utilisée dans ce travail. [9]

[10]Définition de Type de Document.

[11]Cygnus Solutions, initialement Cygnus Support est une entreprise fondée en 1989 par John Gilmore, Michael Tiemann et David Henkel-Wallace pour fournir du support commercial aux logiciels libres.

[12]POSIX est le nom d'une famille de standards définie depuis 1988 par l'IEEE et formellement désignée IEEE 1003.

[13]GNU est un projet de système d'exploitation composé exclusivement de logiciels libres.

4.3 Architecture du système

L'architecture de notre système (figure 4.1) est basée sur l'architecture client/serveur de CoGITaNT. Dans cette architecture un médiateur est un serveur qui offre des services (fonctionnalités) à ses clients. Un client peut être un exportateur ou un importateur de compétences.

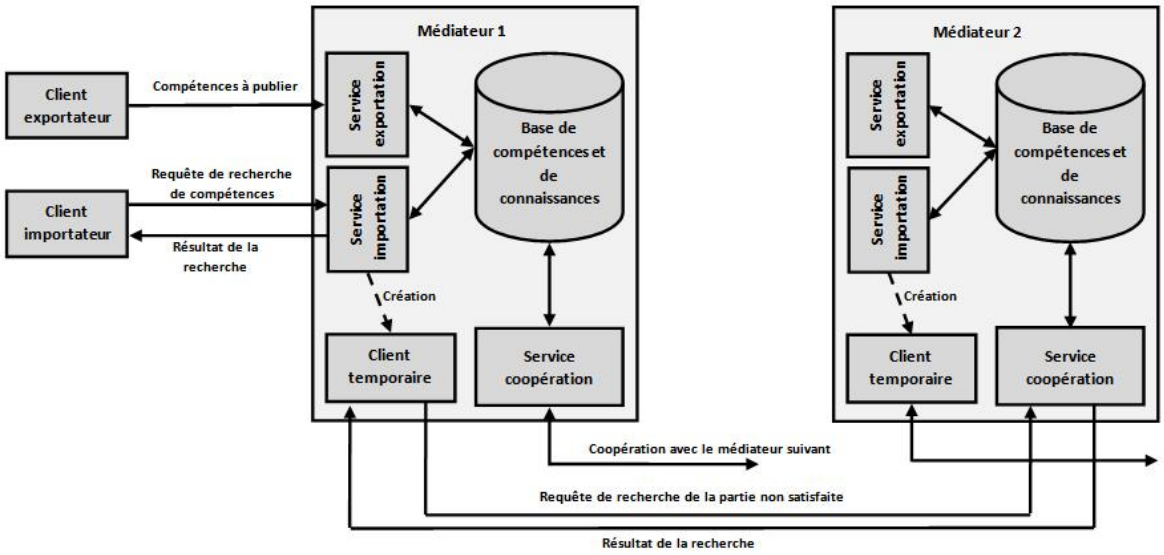


FIG. 4.1 – Architecture de notre système

Les composants de cette architecture sont décrits brièvement ci_dessus :

1. **Le client exportateur :** C'est l'entité voulant publier ces compétences.

2. **Le client importateur :** C'est l'entité qui adresse une requête de recherche des offreurs de compétences.

3. **La base de connaissances et de compétences :** Elle est constituée du support et des règles de graphes décrivant le domaine d'étude ainsi que les compétences publiées.

4. **Le client temporaire :** C'est le composant demandeur de la coopération en cas de nécessité.

5. **Les services offerts :**

   – **Le service exportation :** C'est le service qui s'occupe de la publication de compétences.

- **Le service importation :** C'est le service qui traite la requête du client.
- **Le service coopération :** C'est le service qui s'occupe de la fédération : il traite la requête issue d'un autre médiateur de la fédération.

4.4 Communication entre clients et serveurs

La communication est l'une des caractéristiques principales du système de médiation. CoGITaNT offre des mécanismes de communication entre les clients et les serveurs basés sur une connexion TCP[14] ou HTTP[15]. L'utilisation du protocole HTTP est recommandée si le serveur de CoGITaNT doit être disponible via l'internet.

Le client envoie sa requête écrite sous format XML au serveur qui exécutera le service demandé, et répondra de la même façon. Le mécanisme général de communication et les opérations utilisées sont décrites dans l'Annexe B.

4.5 La base de compétences et de connaissances

Elle est composée du support, des règles, et du graphe "base de compétences". Ces composants sont édités en se servant du logiciel CoGui et enregistrés sous un format CoGXML (voir l'annexe C) pour qu'ils puissent être manipulés par CoGITaNT dans les phases de publication et de découverte.

4.6 Les services offerts par le serveur de médiation

Le serveur de médiation contient un module spécial appelé raisonneur. Il est considéré comme un gestionnaire des tâches ; il se place en attente d'une requête, et selon le service demandé il exécute l'opération concernée. Il assure les fonctions suivantes :

1. Lecture des requêtes transportées sous format XML.
2. Extraction du nom du service demandé.
3. Extraction des paramètres du service demandé.
4. Exécution de l'opération concernée.

[14]Transfer Control Protocol.
[15]HyperText Transfer Protocol.

4.6.1 Service Exportation

Le premier service offert par notre système de médiation est le service d'export spécialisé dans la publication de compétences. Pour mettre en oeuvre ce service nous nous sommes basé complètement sur la bibliothèque CoGITaNT. La classe de base de l'opération d'importation est **cogitantcs : :OpeExportation** qui hérite de la classe **cogitantcs : :OpeServer** fournie par CoGITaNT. Cette classe implémente les étapes de publication de compétences décrites dans la section 3.3.1.2. Après avoir défini ce service, nous devons le rendre accessible par le serveur, nous utilisons à cet effet la méthode **Server : :addOperation()** ayant comme paramètre le nom de l'opération.

Du coté client, nous avons défini un programme client, qui invoque ce service par appel à l'opération d'exportation avec les paramètres correspondants.

Le processus d'exportation de compétences est décrit par le diagramme de séquence de la figure 4.2.

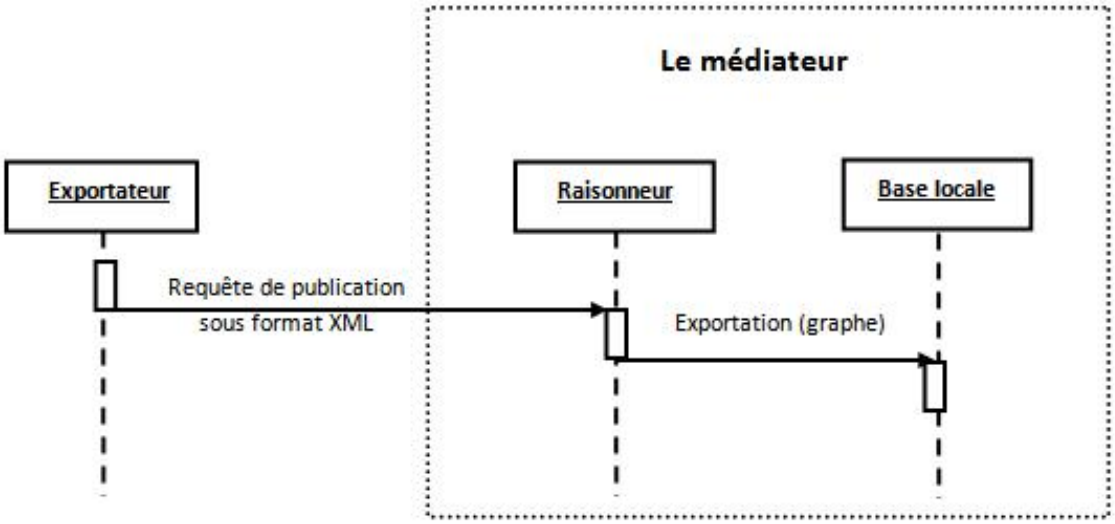


FIG. 4.2 – Diagramme de séquence du processus d'exportation

4.6.2 Service Importation

Le service d'import est le plus important et le plus complexe dans un environnement de fédération de médiateurs puisque une requête de découverte peut nécessiter un accès à plusieurs médiateurs, le résultat final devant être réorganisée dans une seule réponse composite pour la rendre au client.

Pour mettre en oeuvre ce service, nous procédons de la même manière que le service

d'export : nous avons défini à cet effet une classe **cogitantcs : :OpeImportation** qui hérite de la classe **cogitantcs : :OpeServer** de CoGITaNT. Cette classe implémente les étapes de satisfaction d'une requête de recherche de compétences, en suivant les étapes décrites par les diagrammes d'activités présentés dans la section 3.3.2, puis nous l'avons rendu accessible par le serveur.

D'une manière générale, le processus d'importation est décrit par le diagramme de séquence UML de la figure 4.3.

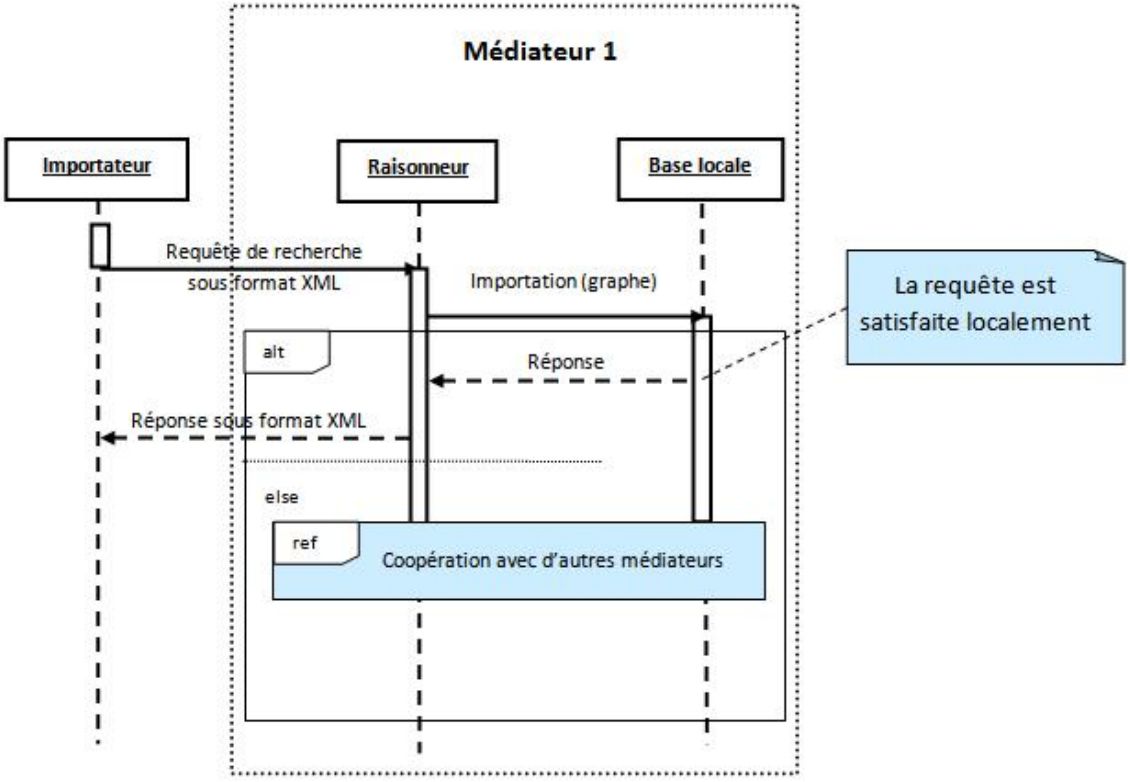


FIG. 4.3 – Diagramme de séquence du processus d'importation

4.6.3 Service Coopération

Le service coopération s'occupe du traitement de la requête de recherche de compétences issue de l'un des médiateurs de la fédération. Pour mettre en oeuvre ce service, nous procédons de la même manière que les services précédents : nous avons alors défini une classe **cogitantcs : :OpeCooperation** qui hérite de la classe **cogitantcs : :OpeServer** de CoGITaNT, et qui implémente les étapes de traitement d'une requête de fédération décrites dans le chapitre précédent.

Les étapes de fédération de deux médiateurs est décrit par le diagramme de séquence

de la figure 4.4.

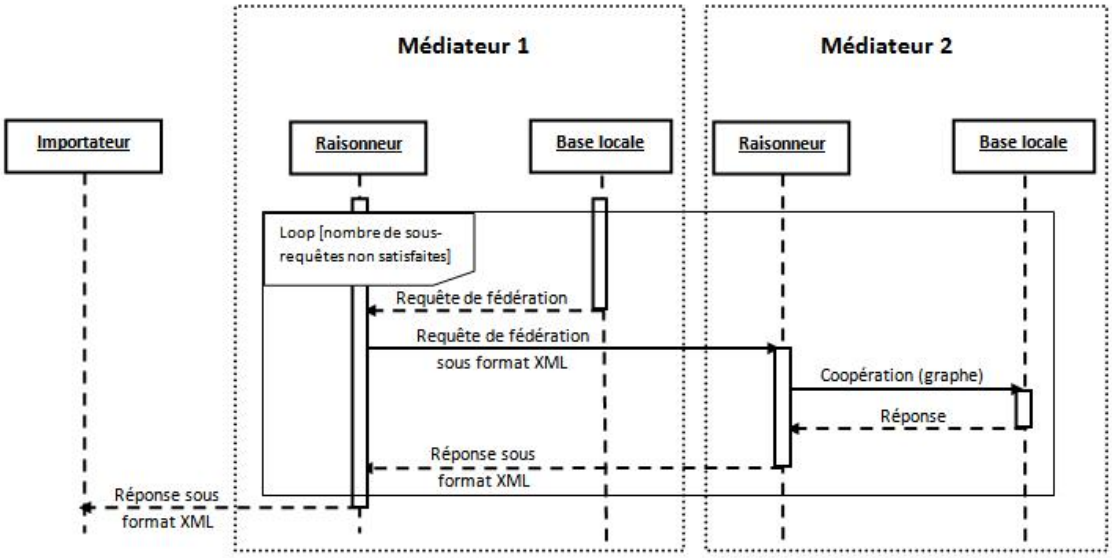


FIG. 4.4 – Diagramme de séquence du processus de coopération

4.7 Scénario d'exécution

Nous illustrons dans cette section un petit exemple de gestion de compétences d'un échantillon d'humains. Nous présentons tout d'abord la population en terme des hiérarchies de concepts et de relations accompagnés des règles de graphes les définissant. Puis, nous présentons des exemples de création et d'exploitation d'une base de compétences.

4.7.1 Présentation de la population du domaine

La population dont on souhaite étudier est présentée par les composants suivants :

1. La hiérarchie de types de concepts : Elle est constituée des concepts primitifs et des concepts définis (voir la figure 4.5). Les concepts primitifs sont : Humain, Femme, Homme, Payé, Ville, Véhicule, Route, Permet-de-conduire, tandis que les types définis sont : Mère, Père, Grand-mère.

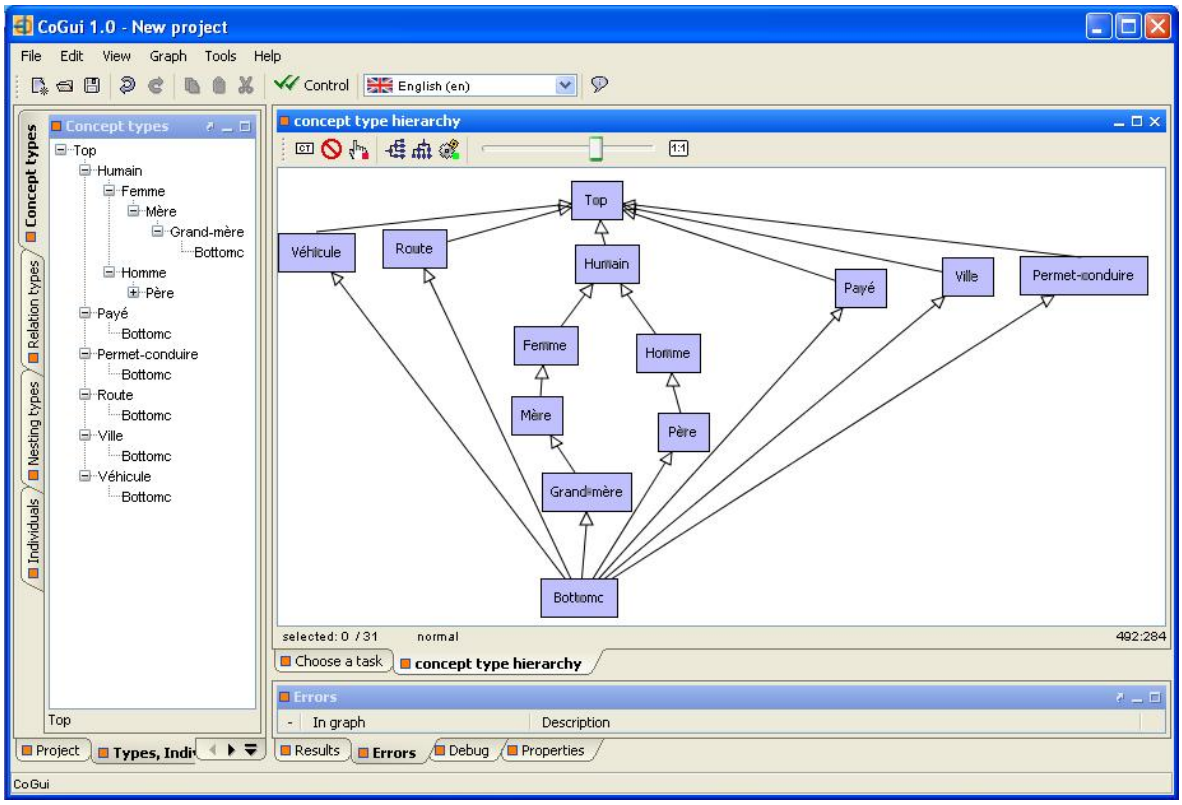


FIG. 4.5 – La hiérarchie de types de concepts

Les concepts définis ainsi que les règles les définissant sont donnés dans le tableau 4.1.

Le type de concept	Les règles de définition
Mère	[Mère :*x]==>[Femme : ?x]->(mère-de)->[Humain :*] [Femme :*x]->(mère-de)->[Humain :*]==>[Mère : ?x]
Père	[Père :*x]==>[Homme : ?x]->(père-de)->[Humain :*] [Homme :*x]->(père-de)->[Humain :*]==>[Père : ?x]
Grand-mère	[Grand-mère :*x]==>[Femme : ?x]->(grand-mère-de)->[Humain :*] [Femme :*x]->(grand-mère-de)->[Humain :*]==>[Grand-mère : ?x]

TAB. 4.1 – Les types de concepts définis avec les règles les définissant

2. Les hiérarchies de relations :

Les relations peuvent aussi être primitives ou définies (figure 4.6). Dans cet exemple, les relations unaires et ternaires sont toutes primitives. Les relations binaires primitives sont : enfant-de, parent-de, possède, peut-conduire et se-trouve.

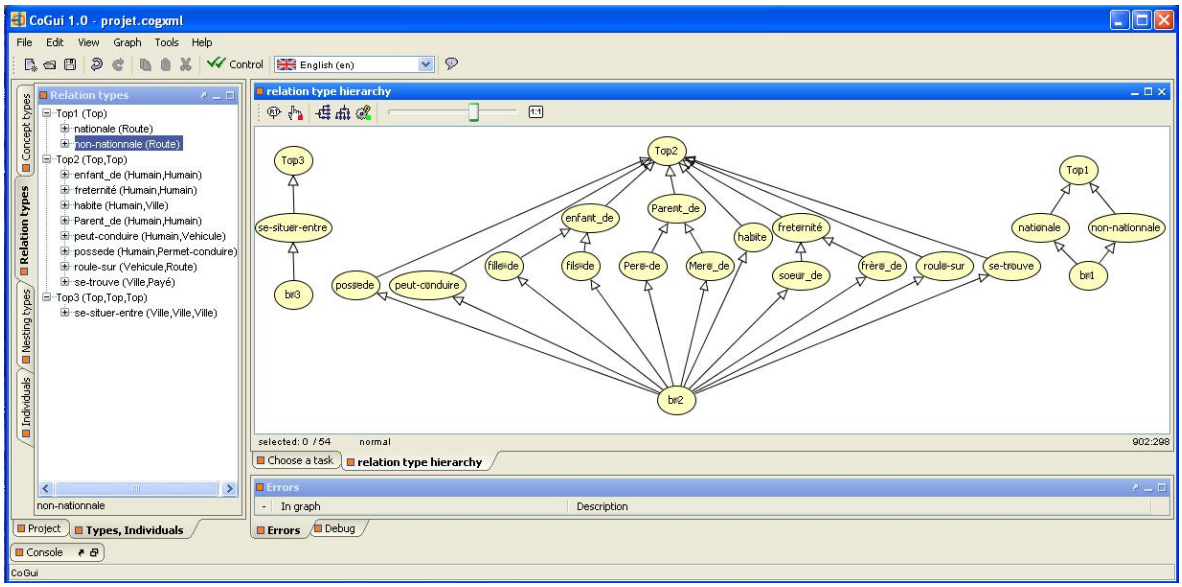


FIG. 4.6 – Les hiérarchies de relations

Les relations binaires définies accompagnées des règles de définition sont présentées dans le tableau 4.2.

Le type de relation	Les règles de définition
fraternité	[Humain :*x]->(fraternité)->[Humain :*y]==> [Humain : ?x]<-(parent-de)<-[Humain :*]->(parent-de)->[Humain : ?y] [Humain :*x]<-(parent-de)<-[Humain :*]->(parent-de)->[Humain :*y]==> [Humain : ?x]->(fraternité)->[Humain : ?y]
frère-de	[Homme :*x]->(frère-de)->[Humain :*y]==>[Homme : ?x]->(fraternité)->[Humain : ?y] [Homme :*x]->(fraternité)->[Humain :*y]==>[Homme : ?x]->(frère-de)->[Humain : ?y]
soeur-de	[Femme :*x]->(soeur-de)->[Humain :*y]==>[Femme : ?x]->(fraternité)->[Humain : ?y] [Femme :*x]->(fraternité)->[Humain :*y]==>[Femme : ?x]->(soeur-de)->[Humain : ?y]
fils-de	[Homme :*x]->(fils-de)->[Humain :*y]==>[Homme : ?x]->(enfant–de)->[Humain : ?y] [Homme :*x]->(enfant-de)->[Humain :*y]==>[Homme : ?x]->(fils-de)->[Humain : ?y]
fille-de	[Femme :*x]->(fille-de)->[Humain :*y]==>[Femme : ?x]->(enfant-de)->[Humain : ?y] [Femme :*x]->(enfant-de)->[Humain :*y]==>[Femme : ?x]->(fille-de)->[Humain : ?y]
père-de	[Homme :*x]->(père-de)->[Humain :*y]==>[Homme : ?x]->(parent-de)->[Humain : ?y] [Homme :*x]->(parent-de)->[Humain :*y]==>[Homme : ?x]->(père-de)->[Humain : ?y]
mère-de	[Femme :*x]->(mère-de)->[Humain :*y]==>[Femme : ?x]->(parent-de)->[Humain : ?y] [Femme :*x]->(parent-de)->[Humain :*y]==>[Femme : ?x]->(mère-de)->[Humain : ?y]

TAB. 4.2 – Les types de relations définies avec les règles les définissant

3. La base des règles : Elle est constituée des règles définissant les types, à lesquelles on ajoute les méta-connaissances sur les relations présentées dans le tableau 4.3.

Les propriétés	Les règles
Transitivité de la relation de fraternité	[Humain :*x]->(fraternité)->[Humain :*]->(fraternité)->[Humain :*y]==> [Humain : ?x]->(fraternité)->[Humain : ?y]
Symétrie des deux relations parent-de et enfant-de	[Humain :*x]->(enfant-de)->[Humain :*y]==> [Humain : ?y]->(parent-de)->[Humain : ?x] [Humain :*x]->(parent-de)->[Humain :*y]==> [Humain : ?y]->(enfant-de)->[Humain : ?x]

TAB. 4.3 – Les méta-connaissances sur les relations

4.7.2 Création et exploitation d'une base de compétences

Nous donnons dans cette section quelques exemples de publication et de traitement de requêtes de recherche des offreurs de compétences. Ces exemples sont testés sur un système composé de deux médiateurs que nous avons appelé M1 et M2 et en limitant le paramètre "profondeur" à 2.

Exemple1 de publication :

Supposons qu'une femme **f1** publie auprès du médiateur M1 le graphe suivant : "[Femme :f1]->(fille-de)->[Homme :h1]".

Après l'exécution du service Exportation, nous avons obtenu la base de compétences BC1 (figure 4.7), visualisée par l'éditeur Cogui.

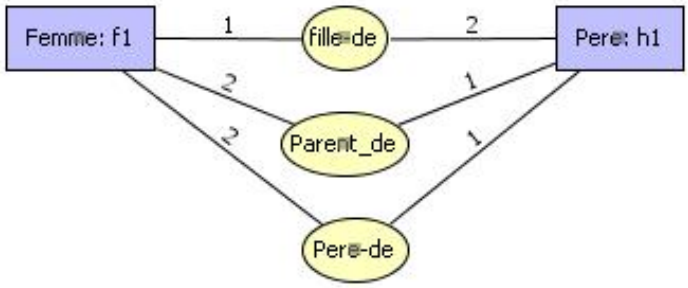


FIG. 4.7 – Exemple1 de publication de compétence

Exemple2 de publication :

Supposons que l'homme **h1** publie auprès du même médiateur M1 le graphe suivant : "[Homme :h1]->(père-de)->[Homme :h2]".

La base de compétences BC1 est mise à jour pour obtenir la base BC2. Nous présentons ici les résultats intermédiaires de l'exécution du service "Exportation", visualisés par l'outil Cogui :

1. Après la somme disjointe (voir figure 4.8).

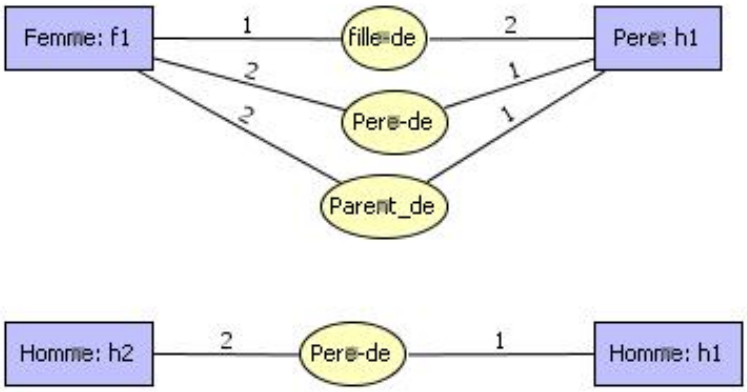


FIG. 4.8 – Exemple2 de publication après la somme disjointe

2. Après normalisation (voir figure 4.9).

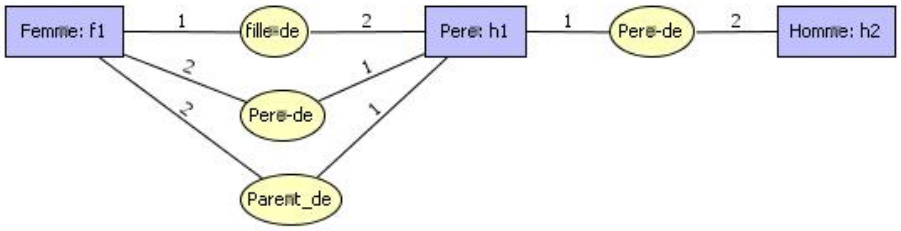


FIG. 4.9 – Exemple2 de publication après la normalisation

3. Après application des règles (voir figure 4.10).

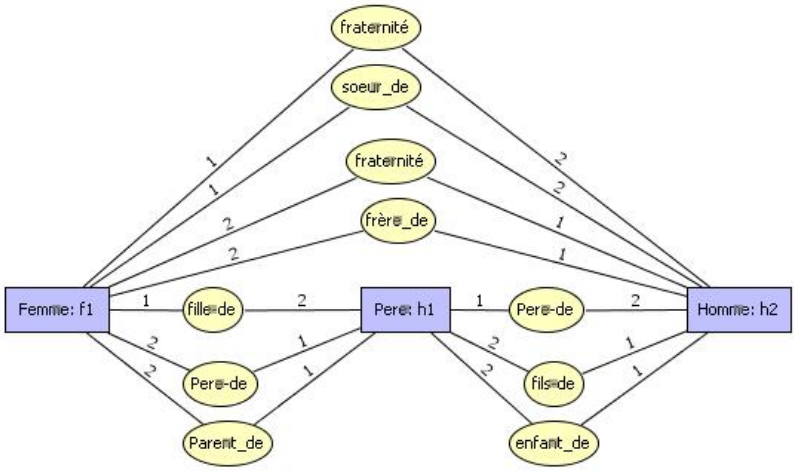


FIG. 4.10 – Exemple2 de publication après l'application des règles

Exemple1 de traitement de requête de recherche des offreurs de compétences :

Supposons qu'un importateur adresse au médiateur M1 la requête suivante : "[Femme : ?]->(s oeur-de)->[Homme : h2]".

Le résultat de l'exécution du service "Importation" est donné dans la figure 4.11.

```
==>>>>>>>>>Traitement de la requête
-----------------------------------------------------------------
_graph3:
        [Homme:h2]<-(soeur_de)<-[Femme:?].
-----------------------------------------------------------------

==>>>>>>>>>La requête est satisfaite localement

==>>>>>>>>>La liste de ces satisfactions
-----------------------------------------------------------------
graph30:
        [Femme:f1]->(soeur_de)->[Homme:h2].
-----------------------------------------------------------------
```

FIG. 4.11 – Exemple1 de recherche des offreurs de compétences

Exemple2 de traitement de requête de recherche des offreurs de compétences :

Nous supposons cette fois-ci que le deuxième graphe a été publié auprès du médiateur M2. Dans ce cas les bases des deux médiateurs M1 et M2 sont donnés dans les figures 4.7 et 4.12 respectivement.

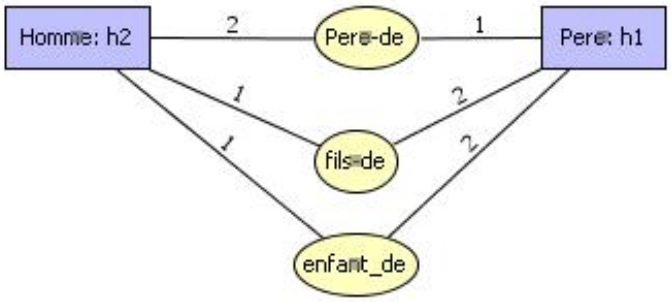


FIG. 4.12 – Base de compétences du médiateur M2.

On adresse la même requête de recherche de compétence de l'exemple1 au médiateur M1 ayant l'@ IP 192.168.0.2.

La requête sera satisfaite grâce à la coopération des deux médiateurs M1 et M2 ayant L'@ IP 192.168.0.1.

Le résultat de recherche sur les deux médiateurs M1 et M2 sont donnés respectivement dans les figures 4.13 et 4.14.

```
==>>>>>>>>>>Traitement de la requête
-----------------------------------------------------------------
_graph3:
        [Homme:h2]<-(soeur_de)<-[Femme:?].
-----------------------------------------------------------------
==>>>>>>>>>>Cette requête non satisfaite localement
==>>>>>>>>>>Essai de satisfaction distribuée
==>>>>>>>>>>Construction de requête et envoi au médiateur ayant l'@ 192.168.0.1
```

FIG. 4.13 – Résultat d'exécution de la requête2 sur le médiateur M1.

```
==>>>>>>>>>>Reception d'une requête ayant 2 niveaux de la part du médiateur ayant l'@ 192.168.0.2

==>>>>>>>>>>La requête est:
-------------------------------------------------------------------------------
graph31:
        [Femme]->(soeur_de)->[Homme:h2].

-------------------------------------------------------------------------------
=>>La requête n'est pas satisfaite au premier niveau, passage au deuxième niveau

=>>Succé, on a trouvé une réponse au deuxième niveau
=>>Les satisfactions sont:
graph9:
        [Homme:h2]->(fils-de)->[Pere:h1].

graph1:
        [Pere:h1]<-(fille-de)<-[Femme:f1].

-------------------------------------------------------------------------------
```

FIG. 4.14 – Résultat d'exécution de la requête2 sur le médiateur M2.

Exemple3 de traitement de requête de recherche des offreurs de compétences :

Nous proposons dans cet exemple l'exécution de la requête de la figure 4.15, après avoir publié les graphes suivants auprès du médiateur M1 dont le contenu de la base locale est présentée dans la figure 4.7 :

1. -[Homme :h1]->(possède)->[Permet-conduire]
2. -[Homme :h1]->(habite)->[Ville :v4]
3. -[Femme :f1]->(habite)->[Ville :v3]
4. -[Ville :v3]->(se-situe-entre)<-[Ville :v1]
   ->[Ville :v2]
5. -[Ville :v4]->(se-situe-entre)<-[Ville :v5]
   ->[Ville :v1]

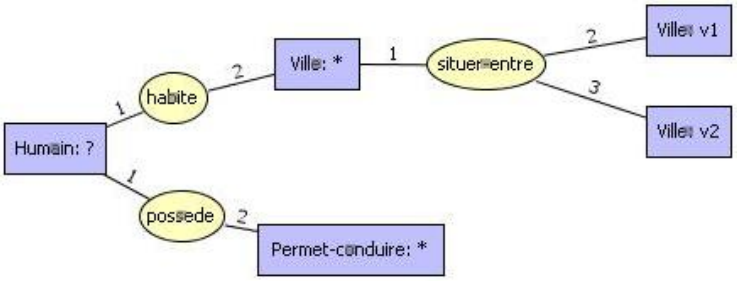


FIG. 4.15 – Exemple3 de requêtes

Le résultat de recherche est présenté dans la figure 4.16.

4.8 Conclusion

Ce chapitre nous a permis de présenter les outils utilisés pour l'implémentation de notre prototype de médiation fédérée à savoir l'éditeur CoGui et la bibliothèque CoGITaNT.

La réalisation de ce prototype est basée sur l'architecture client serveur de CoGITaNT. Un client peut être un exportateur ou un importateur de compétences, tandis que le serveur représente le médiateur.

Un médiateur offre trois services ; le service d'export, le service d'import et le service de coopération avec d'autres médiateurs de la fédération. Ces trois services ainsi que quelques tests prenant en considération différents scénarios sont également présentés dans le présent chapitre.

```
==>>>>>>>>>Traitement de la requête
-----------------------------------------------------------------
_graph4:
        [Ville:v2]<-(se-situe-entre)-
                {
                1<-[Ville]<-(habite)<-[Humain:?]->(possede)->[Permet-conduire];
                2<-[Ville:v1];
                }.
-----------------------------------------------------------------
==>>>>>>>>> La requete n'est pas satisfaite d'une manière exacte

==>>>>>>>>> Recherche de compositions
-----------------------------------------------------------------
 ==>>>>>>>>>La sous requête numéro 0 est:
graph32:
        [Humain:?]->(possede)->[Permet-conduire].
=>>Cette sous requête est satisfaite localement
=>>La liste de ses satisfactions est:
graph30:
        [Pere:h1]->(possede)->[Permet-conduire].

-----------------------------------------------------------------
 ==>>>>>>>>>La sous requête numéro 1 est:
graph35:
        [Humain:?]->(habite)->[Ville]->(se-situe-entre)-
                {
                2<-[Ville:v1];
                3<-[Ville:v2];
                }.
=>>Cette sous requête est satisfaite localement
=>>La liste de ses satisfactions est:
graph36:
        [Femme:f1]->(habite)->[Ville:v3]->(se-situe-entre)-
                {
                2<-[Ville:v1];
                3<-[Ville:v2];
                }.


-----------------------------------------------------------------
==>>>>>>>>> L'ensemble d'entités de la composition est: {[Pere:h1],[Femme:f1]}
```

FIG. 4.16 – Exemple3 de découverte des offreurs de compétences

Pour tester notre système, nous avons pu faire coopérer quatre médiateurs. Ces médiateurs communiquent en échangeant des messages XML et en utilisant le protocole TCP.

Concernant la partie ontologique présentée par le support, celle_ci a été dupliquée sur les quatre médiateurs. Cela avait comme objectif de pouvoir simuler l'existence d'une seule ontologie partagée par tous les médiateurs de la fédération. Dans le cas contraire, on aurait été obligé d'introduire un raisonneur lexical permettant de trouver pour chaque concept ou relation dans un support donné son équivalent le plus proche dans un autre support d'un autre médiateur.

Conclusion et perspectives

Dans ce travail, nous avons traité le problème de la gestion de compétences. Ce problème préoccupe de plus en plus les professionnels de la formation que ceux de la gestion des ressources humaines dans les entreprises ou les administrations dans le but de mesurer et de reconnaître réellement les compétences de chacun, de fixer des objectifs d'évolution pouvant bénéficier à l'amélioration de la productivité.

Notre objectif global consistait à trouver une méthode pour la représentation formelle de compétences ainsi que leur exploitation lors de la recherche d'individus répondant à un besoin donné.

Pour répondre à cet objectif, nous avons débuté notre travail par une étude bibliographique consacrée essentiellement à la gestion de compétences et l'étude des systèmes de médiation ainsi que leur fédération. Cette étude nous a aidé à envisager une solution basée sur les graphes conceptuels et nous avons obtenu à cet effet plusieurs résultats.

1. Nous avons adapté le formalisme de représentation de connaissances des graphes conceptuels pour la représentation et le raisonnement sur les compétences.

2. Du point de vue découverte, nous nous somme appuyés sur la notion d'éclatement de graphes et sur l'opération de projection. La satisfaction peut être locale ou distribuée, simple ou composite.

3. Du point de vue architecture système nous avons implémenté un prototype de médiation fédérée qui permet la composition de réponses lors de la recherche des offreurs de compétences. Ce prototype de médiation fédérée est décentralisé vu qu'un importateur peut adresser sa requête à n'importe quel médiateur de la fédération, autonome vu que chaque médiateur offre des services pour l'importation ainsi que l'exportation de compétences à ces clients.

4. Et enfin, nous avons défini une structure par niveaux de requête de fédération qui peut minimiser enormément le nombre de messages échangées lors d'une fédération.

Plusieurs autres extensions de ce travail sont possibles. En termes de perspectives possibles :

1. En premier lieu, il sera intéressant de continuer à travailler pour améliorer l'expressivité du langage afin de pouvoir représenter quelques concepts et relations. L'une des voies de recherche consiste à considérer la négation et exprimer d'une manière complète l'opérateur $\forall$.

2. Dans ce travail, nous avons opté pour une propagation de requêtes selon un critère de proximité. Néanmoins, pour introduire de l'efficacité dans la propagation des requêtes, il est toutefois préférable d'adopter une propagation selon un critère sémantique : il faut donc de la connaissance sur les autres médiateurs. Mais maintenir cette connaissance a un coût. Toute la difficulté est de trouver un bon équilibre entre la qualité des informations disponibles et les coûts de maintenance. Des études expérimentales peuvent être nécessaires pour bien calibrer ce compromis.

3. Enfin, et en terme d'hétérogénéité, nous n'avons traité dans ce travail que la fédération entre les médiateurs homogènes qui adoptent tous le même langage de représentation de compétences. Cependant, les médiateurs sont très souvent de nature hétérogènes, i.e. ils utilisent des langages différents pour représenter les compétences, il est alors très intéressant d'adapter notre approche pour fonctionner dans des environnements hétérogènes.

Bibliographie

[1] « CharGer : editeur de graphes conceptuels ». Disponible sur : `http://sourceforge.net/projects/charger/`. (Dernière consultation : Avril 2009).

[2] « CoGITaNT : Conceptual Graphs Integrated Tools allowing Nested Typed graphs ». Disponible sur : `http://cogitant.sourceforge.net`. (Dernière consultation : Décembre 2009).

[3] « Cogui : outils de création graphique de graphes conceptuels ». Disponible sur : `http://www.lirmm.fr/cogui`. (Dernière consultation : Septembre 2009).

[4] « CYGWIN ». Disponible sur : `www.cygwin.com`. (Dernière consultation : Décembre 2009).

[5] « Wikipedia ». Disponible sur : `http//www.wikipedia.org/`. (Dernière consultation : Novembre 2009).

[6] Fouzia Amourache, Zizette Boufaïda, et leila Yahiaoui. « Construction d'une ontologie basée compétence pour l'annotation des CVs/Offres d'emploi ». Dans *actes 10th Conference on Software Engineering and Artificial Intelligence (MCSEAI),Maghrebian Conference on Information Technologies (MCIT)*, pages 1–7, Algérie, 2008.

[7] Faical Azouaou. « *Modèles et outils d'annotations pour une mémoire personnelle de l'enseignant* ». Thèse de doctorat, Université Joseph Fourier-Grenoble I, Octobre 2006.

[8] Jean-François Baget. « A Simulation of co-identity with rules in simple and nested graphs ». Dans *actes 9th International Conference on Conceptual Structures (ICCS)*, pages 442–455, 1999.

[9] Jean-François Baget. « *Propositions pour l'extension du modèle des graphes conceptuels - Emboitement, co-référence et règles de réécriture* ». Thèse de master, Université Montpellier II, France, Juin 1998.

[10] Jean-François Baget, Marie-Laure Mugnier, Jean-François Baget, et Baget-liimm Fi. « Extensions of simple conceptual graphs : the complexity of rules

and constraints ». *Journal of Artificial Intelligence Research (JAIR)*, 16 :425–465, 2001.

[11] Gilles Balmisse. « Le marché des outils de KM ». Intervention réalisée dans le cadre du KM Forum 2004. Disponible sur : `http://www.gillesbalmisse.com`. (Dernière consultation : Septembre 2009).

[12] Giuseppe Berio et Mounira Harzallah. « Knowledge Management for Competence Management ». *Journal of Universal Knowledge Management*, 0(1) :21–28, 2005.

[13] Giuseppe Berio et Mounira Harzallah. « Towards an integrating architecture for competence management ». *Computers in Industry*, 58(2) :199–209, 2007.

[14] Tim Berners-Lee, James Hendler, et Ora Lassila. « The semantic web ». *Scientific American*, 284(5) :34–43, 2001.

[15] Nacer Boudjlida. « A Mediator-based architecture for capability management ». Dans *actes 6th IASTED International Conference Software Engineering and Applications - SEA*, pages 45–50, USA, 2002.

[16] Nacer Boudjlida et Rabai Benferroudj. « Une architecture à base de médiateurs pour la gestion de compétences ». Dans *actes 7th Maghrebian Conference on Software Engineering and Artificial Intelligence - MCSEAI*, pages 275–284, Algérie, 2002.

[17] Stephane Brunel, Marc Zolghadri, et Philippe Girard. « Triple instrumentation : une approche de la chaine de valeur ajoutée de la création des connaissances ». Rapport Technique, INRIA a CCSD electronic archive server based on P.A.O.L [http ://hal.inria.fr/oai/oai.php] (France), 2009.

[18] Mohamad Chawk. « Modèle de graphes conceptuels et représentation sémantique du langage naturel ». Rapport de recherche, ERSICO, Université Jean Moulin, Lyon 3, Mars 2000.

[19] Michel Chein et Marie-Laure Mugnier. « Conceptual graphs : fundamental notions ». *Revue d'Intelligence Artificielle*, 6 :365–406, 1992.

[20] Michel Chein et Marie-Laure Mugnier. « Conceptual graphs are also graphs ». Rapport Technique, LIRMM (CNRS and Universite de Montpellier II), 161, rue Ada, 34392 Montpellier Cedex 5, 1995.

[21] Michel Chein et Marie-Laure Mugnier. « Positive nested conceptual graphs ». Dans *actes 7th International Conference on Conceptual Structures (ICCS)*, pages 95–109, USA, 1997.

[22] Michel CHEIN et Marie-Laure MUGNIER. « Concept type and coreference in simple conceptual graphs ». Dans *actes 12h International Conference on Conceptual Structures (ICCS)*, pages 303–318, USA, 2004.

[23] Laurence CHOLVY, Christophe GARION, et Claire SAUREL. « Notion de capacité dans un contexte multi-agents : une modélisation dans lecalcul des situations ». Dans *actes 3eme Conférence Francophone sur les Modèles Formels de l'Interaction (MFI)*, pages 25–27, France, 2005.

[24] Held DANIEL. « La gestion des compétences ». *Revue Economique et Sociale*, 1995.

[25] Juliette DIBIE-BARTHÉLEMY, Ollivier HAEMMERLÉ, et Eric SALVAT. « A semantic validation of conceptual graphs ». *Knowledge Based Systems*, 19(7) :498–510, 2006.

[26] Cheng DONG. « *Gestion et découverte de compétences dans des environnements hétérogènes* ». Thèse de doctorat, Université de Henri Poincaré-Nancy1, Octobre 2008.

[27] Philippe FOURNIER-VIGER. « *Un modèle de représentation des connaissances à trois niveaux de sémantique pour les systèmes tutoriels intelligents* ». Mémoire de maîtrise (m.sc.), Université de Sherbrooke, Sherbrooke, Canada, Décembre 2005.

[28] John F.SOWA. *Conceptual structures : information processing in mind and machine*. Addison-Wesley Publishing Company, USA, 1984.

[29] Fabien GANDON. « Ontologies informatiques ». Disponible sur : `http://interstices.info/ontologie`. (Dernière consultation : Octobre 2009).

[30] Georges GARDARIN. « P2P ». Disponible sur : `http://georges.gardarin.free.fr/Cours_Total/X9-P2P2.ppt`. (Dernière consultation : Octobre 2009).

[31] Badrina GASMI, Nacer BOUDJLIDA, et hassina NACER TALANTIKITE. « Conceptual graphs for competence management ». Dans *actes 3eme Conférence internationale sur les Systèmes d'Information et Intelligence Economique (SIIE)*, pages 505–514, Tunisie, 2010.

[32] Olivier GERBÉ, Guy W. MINEAU, et Rudolf K. KELLER. « Un métamodèle des graphes conceptuels ». *Revue d'Intelligence Artificielle*, 21(2) :255–284, 2007.

[33] Serge HADDAD. « Réseaux sémantiques et graphes conceptuels ». Cours de logique pour l'IA. Disponible sur : `http://julien.chauveau.online.fr`. (Dernière consultation : Octobre 2009).

[34] Stephane HAEFLIGER. « La gestion des connaissances ». Disponible sur : `http://www.stephanehaefliger.com`. (Dernière consultation : Septembre 2009).

[35] Daniel KAYSER. *La représentation des connaissances.* Paris, Editions Hermès, 1997.

[36] Adil KEBBAJ. « Amine : une plate-forme pour le développement de systèmes et d'agents intelligents ». Disponible sur : `http://sourceforge.net/projects/amine-platform`. (Dernière consultation : Avril 2009).

[37] Michel LECLÈRE. « Reasoning with type definitions ». Dans *actes 5th International Conference on Conceptual Structures (ICCS)*, pages 401–415, UK, 1997.

[38] Thérèse LIBOUREL. « Médiation via les métadonnées ». Disponible sur : `www.lirmm.fr`. (Dernière consultation : Septembre 2009).

[39] Claude LÉVY-LEBOYER. *Le bilan de compétences.* Paris, Les Éditions d'Organisation, 1993.

[40] Brahim MEDJAHED, Athman BOUGUETTAYA, et Ahmed K. ELMAGARMID. « Semantic web enabled composition of web services ». *Very Large Data Bases (VLDB journal)*, 12(4) :333–351, 2003.

[41] Riichiro MIZOGUCHI et Jacqueline BOURDEAU. « Le rôle de l'ingénierie ontologique dans le domaine des EIAH ». *Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation*, 11 :231–246, 2004.

[42] Stefan MÜLLER, Kay MÜLLER-JONES, Winfried LAMERSDORF, et Tuan TU. « Global trader cooperation in open service markets ». Dans *actes International Workshop Trends in Distributes Systems (IWTDS)*, USA, 1996.

[43] Mahmoud MORADI, Stephane BRUNEL, et Bruno VALLESPIR. « Approche sémantique de la chaîne de valeur ajoutée de la création des connaissances : de la donnée a la capabilité ». *International Journal of Information Technology and Decision Making (ISDM)*, 7 :33–594, 2008.

[44] Marie-Laure MUGNIER. « *Contributions algorithmiques pour les araphes d'heritage et les graphes conceptuels* ». Thèse de doctorat, Université Montpellier II, France, Octobre 1993.

[45] Marie-Laure MUGNIER. « On Specialization/generalization for conceptual graphs ». *Journal of Experimental and Theoretical Artificial Intelligence*, 7 :325–344, 1993.

[46] Marie-Laure MUGNIER et Michel CHEIN. « Représenter des connaissances et raisonner avec des graphes ». *Revue d'Intelligence Artificielle*, 7(1) :7–56, 1996.

[47] Hassina NACER TALANTIKITE, Djamil AISSANI, et Nacer BOUDJLIDA. « Semantic annotations for web services discovery and composition ». *Computer Standards and Interfaces*, 31(6) :1108–1117, 2009.

[48] Nicolas NICOLOV, Chris MELLISH, et Graeme RITCHIE. « Sentence generation from conceptual graphs ». Dans *actes 3rd International Conference on Conceptual Structures (ICCS)*, pages 74–88, UK, 1995.

[49] Gilbert PAQUETTE. *Modélisation des connaissances et des compétences, pour concevoir et apprendre.* Presses de l'Université du Québec (PUQ), 2001.

[50] Jacques PITRAT. *Métaconnaissance. Futur de l'intelligence artificielle.* Paris, Editions Hermès, 1990.

[51] Michael POLANYI. *The tacit dimension.* Doubleday, USA, 1966.

[52] Chantal REYNAUD et Gloria GIRALDO. « Mediation de services sur le Web ». Dans *actes des Journees Francophones de la Toile (JFT)*, pages 59–68, France, 2003.

[53] MC. ROUSSET et ALL. « Construction de Médiateurs pour Integrer des Sources d'information multiples et heterogenes ». *Revue I3 (Information - Interaction - Intelligence)*, 2(1) :9–59, 2002.

[54] Eric SALVAT. « *Raisonner avec des opérations de graphes : graphes conceptuels et règles d'inférence* ». Thèse de doctorat, Université Montpellier II, Décembre 1997.

[55] Eric SALVAT et Marie-Laure MUGNIER. « Sound and complete forward and backward chainings of graph rules ». Dans *actes 4th International Conference on Conceptual Structures (ICCS)*, pages 248–262, Australie, 1996.

[56] Finnegan SOUTHEY et James G. LINDERS. « Notio - A Java API for developing CG tools ». Dans *actes 7th International Conference on Conceptual Structures (ICCS)*, pages 262–271, USA, 1999.

[57] John F. SOWA. *Knowledge representation : logical, philosophical and computational foundations.* Brooks/Cole Publishing Co., USA, 2000.

[58] Hugo ST-LOUIS. « *Création d'une mémoire collective* ». Mémoire de maîtrise (m.sc.), Université Laval, Avril 2008.

[59] Paola VELARDI, Alessandro CUCCHIARELLI, et Michael PETIT. « A semantically enriched competency management system to support the analysis of a web-based research network ». Dans *actes Web Intelligence*, pages 41–47, USA, 2007.

[60] Paola VELARDI, Roberto NAVIGLI, et Michael PETIT. « Semantic indexing of a competence map to support scientific collaboration in a research community ». Dans *actes International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 2897–2902, Inde, 2007.

[61] Gio WIEDERHOLD. « Mediators in the architecture of future information systems ». *IEEE Computer*, 25(3) :38–49, 1992.

Annexe A :Les outils disponibles pour la manipulation des graphes conceptuels

Un certain nombre d'outils logiciels manipulant les graphes conceptuels ont été mis en oeuvre par des équipes de recherches. Les plus connus sont CoGITaNT, CoGUI, la plateforme Amine, CharGer, Notio, Prolog+CGs. Une brève description des quatre derniers outils est présentée dans cette annexe. CoGITaNT et CoGUI sont présentés plus en détail dans l'annexe B et C respectivement puisqu'ils sont utilisés dans notre travail.

La plateforme Amine

Amine est une plate-forme Java open-source et multicouches, elle a été créé dans I.N.S.E.A (Rabat, le Maroc) par le docteur Adil Kabbaj, elle est dédiée au développement de systèmes intelligents et au développement d'agents. La plate-forme Amine est composée de quatre couches : une couche noyau qui permet la création, l'édition, la mise à jour et la manipulation d'ontologies multilingues, une couche algébrique qui offre un ensemble de classes (structure + opérations) ; une couche de programmation et une quatrième couche permettant le développement d'agents et de systèmes multi-agents. La couche de programmation fournit trois paradigmes de programmation : un paradigme de programmation basée sur la mémoire, un paradigme de programmation à base de règles incorporé dans le langage PROLOG+CG et un paradigme de programmation visuelle à base d'activation et de propagation, incorporé dans le langage SYNERGY.

Prolog+CG est un langage de programmation logique qui intègre PROLOG, les Graphes Conceptuels, la programmation Orientée objet et JAVA. Il a été créé dans I.N.S.E.A (Rabat, le Maroc) par le docteur Adil Kabbaj. Il est en fait un moteur Prolog écrit en Java avec des extensions orientées objet et l'appui incorporé pour les

graphes conceptuels. Il supporte les graphes conceptuels simples et la coréférence et fournit un ensemble d'opérations sur la hiérarchie de types et sur les graphes conceptuels (la subsomption, la jointure maximale, etc.). Un programme Prolog+CG peut être organisé en objets et ces objets peuvent être organisés dans une hiérarchie d'héritage. Finalement, il y a deux façons de connectivité avec Java, en effet un Programme Java peut appeler un programme Prolog+CG et poser des requêtes, et un programme Prolog+CG peut accéder aux classes Java et définir/utiliser des objets globaux.

CharGer

C'est un prototype d'un éditeur des graphes conceptuels créé à l'université Alabama à Huntsville par Harry Delugach. Il supporte la création de concepts, des relations, les liens de coréférence, des hiérarchies de types de concepts et de relation, des contextes (incluant des emboîtements), la jointure des graphes et le matching. Il peut enregistrer et lire des graphes dans un format texte propre, il est capable également d'importer, exporter et manipuler les graphes conceptuels en format CGIF. En plus il peut translater les graphes conceptuels dans une langue anglaise contrôlée. Le GharGer est gratuit pour un usage non commercial et fonctionnant sous Java.

Notio

Notio est une spécification d'API pour un ensemble de classes Java conçues pour fournir une implementation-independent interface pour manipuler les graphes conceptuels. Une implémentation de référence de l'API Notio a été construite pour l'évaluation, l'expérimentation et pour servir comme une base pour les implémentations futures de l'API. L'implémentation est dans une étape alpha et elle est limité dans les fonctionnalités.

Annexe B :La bibliothèque CoGITaNT

Présentation de CoGITaNT

CoGITaNT (Conceptual Graphs Integrated Tools allowing Nested Typed graphs)) est une bibliothèque de classes C++ permettant le développement des logiciels basés sur le formalisme des graphes conceptuels. CoGITaNT est une extension de la plateforme CoGITo développée en 1994 par l'équipe de représentation des connaissances et d'inferences avec des graphes (LIRMM).

CoGITaNT fournit un grand nombre de classes et de méthodes élémentaires pour les graphes conceptuels. Ces dernières inclut la définition des types de concepts et de relations, la définition des règles de graphes, les graphes conceptuels emboités sont aussi supportés. La bibliothèque fournit aussi un certain nombre d'opérations pour manipuler les graphes conceptuels, la plus importante est l'opération de projection. Les bases de connaissances (supports ou graphes) peuvent êtres sauvegardés pour un usage ultérieur vu que la bibliothèque fournit des opérations d'entrée/sortie.

Une description détaillée de la bibliothèque est donnée dans les sections suivantes.

La hiérarchie des objets CoGITaNT

CoGITaNT suit une approche orienté objet, pour cela toutes ces fonctionnalités sont implémentées avec des classes.

La classe principale de CoGITaNT est Environment. Cette classe fournit des méthodes pour manipuler le support, exécuter des opérations de projection, appliquer les règles de graphes, écrire/lire la base de connaissances dans des fichiers sur disque et afficher tous les éléments de la base sur l'écran.

Une autre classe importante est la classe Graph. Cette classe gère les graphes qui sont représentés comme un ensemble d'instances de la classe GraphObject qui sont connectés avec un ensemble d'arcs (instances de la classe Edge). Évidemment, un noeud

peut être un concept (la classe Concept) ou une relation (la classe Relation), mais pour représenter les graphes emboités, la classe Nesting peut être utilisée.

La classe Rule est utilisée pour représenter les règles. L'hypothèse et la conclusion d'une règle doivent être des graphes simples (et non pas emboîtés) et ne doivent pas avoir des liens de coréférence. La création des règles est tout à fait simple, puisque la classe fournit des méthodes pour la création des parties hypothèse et conclusion, aussi bien que pour définir des points de connexion possibles.

Une des classes de la bibliothèque CoGITaNT les plus fréquemment utilisées est l'iSet. Cette classe offre un identificateur pour chaque instance d'objet créé. Par exemple, le support contient un ensemble de types de concepts, l'environnement contient un ensemble de graphes et un ensemble de règles, chaque graphe contient un ensemble de concepts et un ensemble de relations, etc. La plupart des méthodes disponibles utilisent une instance de la classe iSet pour accéder à ces éléments.

Une partie importante de la hiérarchie de classes de CoGITaNT, chacune des classes est suivie par une courte description, est donnée ci_dessus.

- **cogitant : :CogitantObject :** C'est la classe abstraite mère de n'importe qu'elle autre classe de la bibliothèque.
- **cogitant : :Environment :** L'environnement est l'objet qui contient le support, les graphes, et les règles définies sur ce support. En plus, cette classe fournit des méthodes pour utiliser facilement les opérations principales comme la projection, les opérations d'entrée/sortie (newGraph(), newRule(), graphs(), rules(), projections(), projectionsImage(), ruleApplications(), ruleApply(), disjoinSum(), readSupport(), writeSupport(), readGraph(), writeGraph()).
- **cogitant : :EnvironmentObject :** C'est la classe abstraite mère de chaque objet défini sur un environnement.
- **cogitant : :Graph :** Cette classe représente les graphes conceptuels en mémoire. Elle fournit des méthodes permettant la gestion des graphes (newGenericConcept(), newIndividualConcept(), newRelation(), link(), join()).
- **cogitant : :Rule :** Cette classe décrit une règle avec un graphe hypothèse, un graphe conclusion et lie entre les noeuds concepts de ces deux parties. L'hypothèse et la conclusion doivent être des graphes simples (hypothesis(), conclusion(), connectionPoints(), addConnectionPoint()).
- **cogitant : :GraphObject :** C'est la classe abstraite mère de chaque objet dans le graphe.

- **cogitant : :Concept :** Cette classe décrit les noeuds concepts dans un graphe.
- **cogitant : :Relation :** Cette classe décrit les n oeuds relations dans un graphe.
- **cogitant : :ResultOpeProjection :** Cette classe représente le résultat de l'opération de projection (projections()) .
- **cogitant : :Support :** Cette classe représente le support sur lequel les graphes conceptuels sont construits (newConceptType(), newRelationType(), newIndividual()).
- **cogitant : :SupportObject :** C'est la classe abstraite mère de chaque objet défini dans le support.
- **cogitant : :Individual :** Cette classe est dédiée aux marqueurs individuels.
- **cogitant : :SupportType :** C'est la classe abstraite mère de chaque type dans le support.
- **cogitant : :ConceptType :** Cette classe permet la définition de types de concepts.
- **cogitant : :RelationType :** Cette classe permet la définition de types de relations.
- **cogitant : :Edge :** Cette classe représente un arc dans le graphe.
- **cogitant : :PartialOrder :** C'est une classe abstraite pour représenter un ordre partiel sur les instances de la classe iSet.

L'architecture client/serveur de cogitant

L'architecture client/serveur est mise en oeuvre avec d'autres classes définies dans le namespace cogitantcs. Ce namespace a été mis en oeuvre à partir de la version CoGITaNT 5.0.90 (bêta). Les classes principales sont la classe Client et la classe Server et leurs sous-classe Client_TCP, Client_HTTP et Server_TCP. Ici, nous expliquerons seulement l'approche de TCP vu que c'est l'approche utilisée dans ce travail. Vous pouvez vous référez à la documentation pour la communication HTTP.

La communication entre le client et le serveur est effectuée par des sockets TCP ou par le HTTP via un Serveur Web. Les messages sont décrits dans un format XML et leurs structures sont définies par une DTD. Il est possible d'étendre le protocole en ajoutant des opérations.

Le serveur

Premièrement , le serveur doit être initialisé par le constructeur : Server_TCP : :Server_TCP(Port).

Le paramètre port définie le port TCP sur lequel le serveur va écouter. En fait c'est pas obligatoire, et le port par défaut est 4246. Après initialisation, les opérations doivent être ajoutées dans le serveur pour pouvoir être effectuées. La méthode utilisée pour ajouter une opération est : Server : :addOperation(operation). Le paramètre "operation" est une instance d'une classe héritant de la classe OpeServer.

Pour ajouter les opérations standards, la méthode à utiliser est : Server : :addStdOperations(). Quand les opérations sont insérées, un environnement doit être crié dans le serveur par la méthode : Server : :addEnvironment (environment, name). Tel que environment est un pointeur vers une instance Environment, et name est le nom de l'environnement dans le client.

Enfin, la boucle principale est lancé par la méthode : Server : :mainLoop().

Les opérations sur le serveur

Une opération sur le serveur est une classe qui hérite de la classe OpeServer. L'utilisateur doit redéfinir les méthodes :

- **OpeServer : : name() :** Cette méthode rend le nom de l'opération. Ce nom est utilisé par le serveur pour décoder le document XML pour lancer l'opération appropriée.
- **OpeServer : : run () :** Cette méthode contient l'opération elle-même. Cette méthode peut obtenir les paramètres de la demande avec OpeServer : : m_query et OpeServer : : m_querynode. Le champ OpeServer : : m_server permet à la méthode d'avoir accès au serveur et par conséquent, ses environnements. Finalement, la méthode peut remplir le champ OpeServer : : m_answer par les résultats de l'opération. Ce message XML sera envoyé au client comme une réponse.

Le client

Quand le serveur est lancé, le programme client peut être exécuté. Le client est initialisé par le constructeur (pour un accès de TCP) : Client_TCP : :Client_TCP(host, port, localsupport, localobjects). tel que host est le nom de l'hôte ou l'adresse IP du serveur éloigné distant, port est le port TCP sur lequel le client écoutera. Le port par defaut est 4246. Localsupport et localobjects sont des booléens et sont utilisé pour acti-

ver (faux) ou desactivate (vrai) la connexion entre le client et le serveur. La connexion elle-même est créé dans le constructeur et détruit dans le destructeur du Client.

Quand la connexion est établie, la première méthode à appeler est Client : :setServerEnvironment(environement), avec environnement est l'identificateur d'un environnement dans le serveur ou le nom de l'environnement d'un serveur. Après cela, la méthode Client : :environment() peut être utilisé pour avoir accès à l'environnement.

Les méthodes principales du client sont :

- **Client : :answer() :** Cette méthode retourne un document XML décrivant la réponse du serveur après appel de la méthode Client : :addPendingQuery().
- **Client : :commitEnvironmentObject (IsetLocalObject, IsetRemoteObject, imm) :** Cette méthode met à jour des objets sur le serveur à partir du client. IsetLocalObject est le iSet du client sur le client, IsetRemoteObject est le iSet sur le serveur.
- **Client : :newEnvironment(imm) :** Cette méthode crie un nouveau environnement au serveur et retourne son iSet.
- **Client : :newEnvironmentObject(Type, imm) :** Cette méthode crie un nouveau objet dans le serveur. Type est soit EnvironmentObject : :OT_GRAPH pour créer un graphe ou EnvironmentObject : :OT_RULE pour créer une règle.
- **Client : :deleteEnvironmentObject(Object, imm) :** Cette méthode supprime un objet du serveur. Object représente l'iSet de l'objet dans le serveur.
- Client : :refreshEnvironment(imm) : Cette méthode met à jour l'environnement du client à partir du serveur, qui envoi le support et le nombre des objets dans l'environnement. La méthode doit être appelée après newEnvironmentObject().

Toutes les méthodes ont le paramètre imm. S'il est met à true(la valeur par défaut), la requête est immédiatement envoyée au serveur. Dans le cas contraire, la requête va être envoyée lors de l'appel à Client : :executePendingQueries() .

Annexe C :CoGUI et le format CoGXML

Présentation de CoGUI

CoGUI est un outil visuel gratuit basé sur les graphes, développé en java pour construire des bases de connaissances à base des graphes représentée sous format CoGXML.

CoGUI est doté d'une interface simple à utiliser, elle sert à l'édition du support, des graphes conceptuels, des règles de graphes et des graphes emboités. L'interface principale de Cogui est donnée dans la figure 4.17.

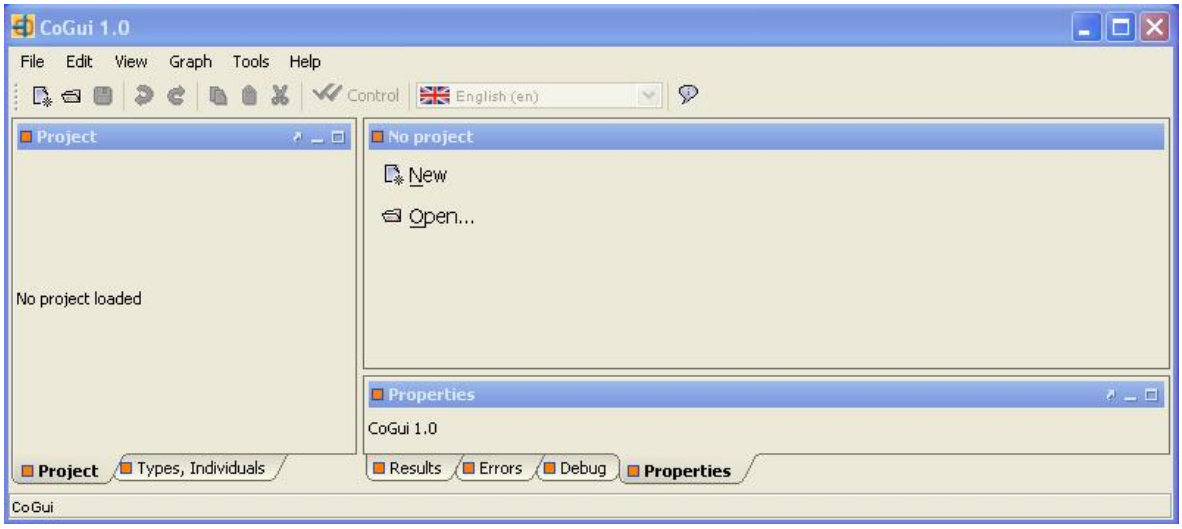


FIG. 4.17 – L'interface principale de Cogui

Le format CoGXML

Le langage XML permet la représentation de données structurées. Un document XML est une représentation d'objets sous un format semblable à un arbre. On peut

considérer cela comme valable et formé. Valable signifie qu'il est construit en accordance à une DTD (la Définition de Type de Document). Une DTD est utilisée pour définir la structure et le type des balises d'un document XML. Pour être valable, le document XML doit respecter les définitions fondées dans sa DTD.

Le format de CoGXML permet la représentation de graphes conceptuels dans le format XML. LA DTD permet une représentation du support, la représentation de graphes (simples et emboités) et la représentation de règles.

Ce qu'a été appelé CGXML dans les versions précédentes de CoGITaNT, est maintenant appelé CoGXML. À l'origine, ils étaient deux formats nommés CGXML, les deux permettent la représentation des graphes conceptuels en format XML.

Une vue générale du document CoGXML

Comme n'importe quel fichier XML, un fichier CoGXML contient une en-tête XML spécifiant la version XML et la DTD associée. Suivi de la balise <CoGXML>. Cette balise contient les déclarations de support, des graphes et des règles. Le fichier de GoGXML est terminé par une balise </CoGXML>.

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

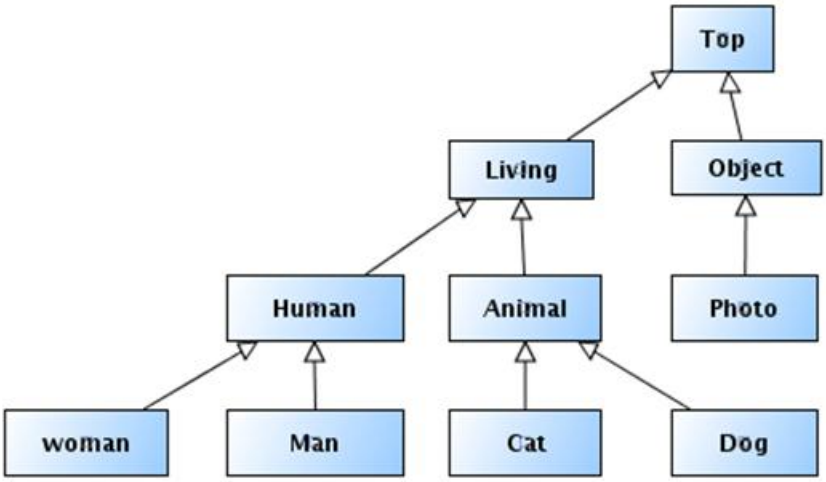
Une vue du support CoGXML

Le support a un identifiant et se compose de la hiérarchie des types de concepts, des hiérarchies des types de relations, d'emboitements et des marqueurs individuels.

```
<support id="support">
        <conceptTypes>
        </conceptTypes>

        <relationTypes>
        </relationTypes>

        <nestingTypes>
        </nestingTypes>

        <individuals>
        </individuals>
</support>
```

Hiérarchies de types de concepts et d'emboitements

Les hiérarchies de types de concepts et d'emboitements sont construites de manière identique.

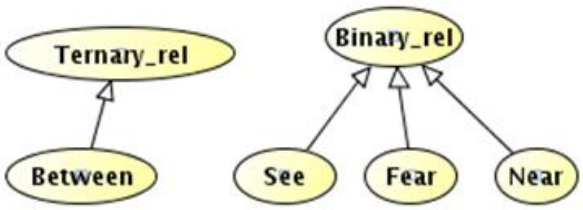


Dans sa forme la plus simple, la hiérarchie de types de concepts se compose des sommets de type de concepts (vocabulary) ainsi que de la relation de sous-typage (subtypeOfRelation). On peut préciser la racine de la hiérarchie en spécifiant un attribut root sur la relation de sous-typage.

```
<conceptTypes>
        <vocabulary>
                <conceptType id="Top"/>
                <conceptType id="Living"/>
                <conceptType id="Human"/>
                <conceptType id="Woman"/>
                <conceptType id="Man"/>
                <conceptType id="Animal"/>
                <conceptType id="Dog"/>
                <conceptType id="Cat"/>
                <conceptType id="Object"/>
                <conceptType id="Photo"/>
        </vocabulary>

        <subtypeOfRelation root="Top">
                <edge from="Living" to="Top"/>
                <edge from="Human" to="Living"/>
                <edge from="Woman" to="Human"/>
                <edge from="Man" to="Human"/>
                <edge from="Animal" to="Living"/>
                <edge from="Cat" to="Animal"/>
                <edge from="Dog" to="Animal"/>
                <edge from="Object" to="Top"/>
                <edge from="Photo" to="Object"/>
        </subtypeOfRelation>
</conceptTypes>
```

Hiérarchie de types de relations

La hiérarchie de types de relation est pratiquement identique à celles vues au dessus. Elle diffère essentiellement sur la signature des types de relations.



```
<relationTypes>
        <vocabulary>
                <relationType id="Binary_rel" arity="2"/>
                <relationType id="Near" arity="2"/>
                <relationType id="See" arity="2"/>
                <relationType id="Fear" arity="2"/>
                <relationType id="Ternary_rel" arity="3"/>
                <relationType id="Between" arity="3"/>
        </vocabulary>

        <subtypeOfRelation>
                <edge from="Near" to="Binary_rel"/>
                <edge from="See" to="Binary_rel"/>
                <edge from="Fear" to="Binary_rel"/>
                <edge from="Between" to="Ternary_rel"/>
        </subtypeOfRelation>


        <signatures>
                <signature ref="Fear">
                        <parameter>
                                <type ref="Living"/>
                        </parameter>
                        <parameter>
                                <type ref="Living"/>
                        </parameter>
                </signature>
                <signature ref="See">
                        <parameter>
                                <type ref="Living"/>
                        </parameter>
                        <parameter/>
                </signature>
        </signatures>
</relationTypes>
```

On indique l'arité de chaque type de relation grâce à l'attribut arity. Il n'y a pas d'attribut root comme c'était le cas dans la hiérarchie de types de concepts.

Les signatures des relations sont définies dans un bloc séparé, chaque paramètre de la signature peut être un type simple, conjonctif, ou désigner implicitement la racine.

Ici, on déclare que la relation Fear a comme signature deux êtres vivants (Living), et que la relation See implique un être vivant et la racine de la hiérarchie de types de concepts.

Les relations dont la signature n'est pas explicitement désignée ont une signature composée de n fois la racine des types de concepts, n étant son arité.

Marqueurs individuels

Les marqueurs individuels se définissent en explicitement, ils sont déclarés comme étant du type de la racine des types de concepts.

Les marqueurs individuels se définissent en explicitement, ils sont déclarés comme étant du type de la racine des types de concepts.

```
<individuals>
        <vocabulary>
                <marker id="Marc"/>
                <marker id="Hector"/>
                <marker id="Quasimodo"/>
        </vocabulary>
        <conformity>
                <marker ref="Marc">
                        <type ref="Man"/>
                </marker>
                <marker ref="Quasimodo">
                        <type ref="Man"/>
                        <type ref="Animal"/>
                </marker>
                <marker ref="Hector">
                        <type ref="Dog"/>
                </marker>
        </conformity>
</individuals>
```

On déclare ici trois individus : Marc qui est un homme, Hector le chien, et Quasimodo qui est à la fois un homme et un animal.

Graphes

Graphe simple

Un graphe est composé d'un ensemble de sommets concepts et relations.

```
<graph>
        <concepts>
        ...
        </concepts>
        <relations>
        ...
        </relations>
</graph>
```

Concepts

Chaque sommet concept à un identifiant, il peut avoir un référent, un type et des liens de coréférences

```
<concepts>
        <concept id="c123">
                <referents>
                        <marker ref="Quasimodo"/>
                </referents>
                <type ref="Man"/>
                <coreference ref="c98"/>
                <coreference ref="c76"/>
        </concept>
        ...
</concepts>
```

Un concept peut n'avoir aucun référent, dans ce cas son marqueur est le marqueur générique ("*"), il peut avoir un ou plusieurs marqueurs individuels.

Faits et requêtes

Les faits et les requêtes se composent d'un graphe simple vu précédemment

```
<fact>
        <graph>
        ...
        </graph>
</fact>
<query>
        <graph>
        ...
        </graph>
</query>
```

Règles

Les règles se composent de deux parties : une hypothèse, une conclusion et un ensemble de points de connexion.

```
<rule>
        <hypothesis>
                <graph>
                ...
                </graph>
        </hypothesis>
        <conclusion>
                <graph>
                ...
                </graph>
        </conclusion>
</rule>
```

RESUME

Dans le cadre de ce mémoire, nous nous intéressons au problème de la gestion de compétences. Par compétence, nous entendons les capacités (ou services) d'une entité physique ou pas (un objet, un composant logiciel, un partenaire industriel, un service web, etc.).

La gestion de compétences est un processus complexe et nous le traitons dans ce travail en trois étapes : la représentation de compétences, leurs organisation et leur utilisation lors de la recherche d'entités (ou offreurs de compétences) pouvant satisfaire un besoin. Une des originalités de notre travail est la nature des réponses rendues lors d'une requête de recherche des offreurs de compétences. En effet les réponses ne sont pas du type oui/non mais elles peuvent être de type coopératives. Dans ce travail, nous avons opté pour une approche basée sur les graphes conceptuels tant pour la représentation et l'organisation de compétences que pour leur utilisation lors de la découverte des offreurs de compétences.

Mots clés : Gestion de compétences, Découverte, Réponses composites, Représentation de connaissances, Médiation, Graphes conceptuels .

ABSTRACT

This work adresses the problem of competence management. By competence, we mean the capability or services of a physical or a non physical entity (an object, a softeware component, a possible industrial partner, a web service, etc.).

The competence management is a complex process that we treat in this work into three steps : competence representation, competence organization and competence use while searching for entities (or compétence providers) that can satisfy a need. One of the originalities of our work is the nature of the responses made during a request of competence provider search. Indeed, the answers are not yes/no answers but they may be cooperative answers. In this work, we opted for a conceptual graphs based approach both for the competence representation and organization and for competence provider discovery.

Keywords : Competence management, Discovery, Composite answers, Knowledge representation, Mediation, Conceptual graphs.