HUGO ST-LOUIS

CRÉATION D'UNE MÉMOIRE COLLECTIVE

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en informatique
pour l'obtention du grade de Maître ès Sciences (M.Sc.)

DÉPARTEMENT D'INFORMATIQUE ET DE GÉNIE LOGICIEL
FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

AVRIL 2008

Résumé

Avec l'amélioration et la prolifération de la téléinformatique, les systèmes informatiques distribués connaissent un véritable essor dans l'industrie du logiciel. Un système distribué est un système pour lequel chaque composante est indépendante et souvent située à un endroit éloigné des autres composantes. Dans un tel système, les composantes communiquent entre elles pour échanger des informations, coordonner leurs actions, prendre des décisions collectives, etc. Cependant, dans ces systèmes, certaines problématiques subsistent ; comment représenter les informations, comment transmettre les informations et comment gérer la mise à jour de ces informations? Ce mémoire s'intéresse principalement à ces problèmes pour les systèmes distribués où les informations sont centralisées sur un serveur. Pour le cas qui nous intéresse, une partie seulement de ces informations est dupliquée dans les composantes et cette duplication peut ne pas être exacte.

Ce mémoire porte sur le partage de connaissances entre les différentes composantes d'un tel système distribué. Nous détaillerons plus particulièrement le mode de représentation des connaissances, basé sur les graphes conceptuels, ainsi qu'une méthode d'indexation appliquée à ce mode de représentation des informations. Nous proposons une technique originale de gestion des mises à jour. Cette technique, basée sur une analyse statistique des mises à jour, permet d'appréhender et de maintenir à jour les informations entre les composantes du système et le serveur. Avec ces techniques, nous espérons

diminuer l'utilisation de la bande passante ainsi que l'incohérence momentanée des informations chez les composantes du système.

Abstract

With the improvement and the proliferation of teleinformatics, the distributed systems are widely used in the software industry. A distributed system is a system for which each component is independent and often located in a place away from the other components. In such system, components communicate with each other to exchange information, to coordinate their actions, to make group decisions, etc. However, in these systems, some problematic remain; how to represent information, how to transmit information and how to manage the update information? This study is mainly interested in these problematic of distributed systems where information is centralized on a server. For the case which interests us here, only a part of information is duplicated in the components and it may not be exact.

This study is related to the way of sharing knowledge between components of such distributed system. We will more particularly detail the knowledge representation, based on conceptual graphs, as well as a method of indexing this knowledge representation. We proposed an original technique of management of the updates. This technique, based on a statistical analysis of the updates, makes it possible to perceive and maintain up to date the information between the components of the system and the server. With these techniques, we hope to decrease the use of the bandwidth as well as the temporary inconsistency of information between the components and the server.

Avant-propos

Ce mémoire est un carnet de voyage, récit d'une odyssée qui s'inscrit dans ma quête de connaissances. J'arrive à destination plus sage qu'à mon départ, mais tout autant avide de découvertes. J'ai descendu le fleuve; maintenant, l'océan est devant moi.

Je fais une halte pour exprimer ma gratitude à mes codirecteurs, messieurs Guy Mineau et François Laviolette. Ils m'ont livré un bagage de connaissances que, désormais, je transporte avec moi. Je remercie également les évaluateurs qui ont exploré mon mémoire : Luc Lamontagne et Mario Marchand.

En chemin, j'ai croisé des personnes qui m'ont donné le nécessaire pour continuer ma route. Ma famille, mes amis, mes enseignants et mes paires ont été des compagnons de voyage exceptionnels à qui je garde une place bien spéciale dans mon baluchon.

Enfin, cette aventure n'aurait pas été possible sans le soutien financier des Québécois qui, chaque année, permettent à des milliers d'étudiants d'atteindre les sommets.

À ma compagne de voyage,
Caroline (Belonde) pour ses encouragements
et son indéfectible amour.
À mes parents, Manon et Clermond, pour
leur amour et leur réconfort que j'emporte
avec moi pour me sentir près d'eux.

Le voyage pour moi, ce n'est pas arriver,
c'est partir. C'est l'imprévu de la prochaine
escale, c'est le désir jamais comblé de
connaître sans cesse autre chose, c'est
demain, éternellement demain.
[Roland Dorgelès]

Table des matières

Résumé....i
Abstract.... iii
Avant-propos iv
Table des matières vi
Liste des tableaux.... viii
Liste des figures ix
Table des algorithmes x
Chapitre 1 Introduction....1
1.1 Problématique 1
1.2 Motivations 5
1.2.1 Redondance et incohérence des connaissances 7
1.2.2 Les applications 8
1.3 Objectifs....10
1.4 Approche proposée 12
1.5 Organisation du mémoire....15
Chapitre 2 Approche proposée pour la représentation et l'accès aux connaissances 16
2.1 Introduction....16
2.2 Représentation et accès aux connaissances 17
2.2.1 L'essentiel des graphes conceptuels 18
2.2.2 La manipulation des connaissances 27
2.2.3 Recomposition d'un graphe conceptuel 34
2.2.4 Recherche structurelle à travers un ensemble de briques 36
2.2.5 Extension sémantique d'une brique....41
2.3 Conclusion 49
Chapitre 3 La recherche d'informations....50
3.1 Introduction....50
3.2 Reconstitution d'un ensemble de briques 51
3.3 Recherche de connaissances 54
3.3.1 Vue d'ensemble sur l'algorithme de recherche....54
3.3.2 Discussion sur l'algorithme de projection 58
3.4 Expérimentation de l'algorithme de recherche versus un algorithme de projection équivalent....59
3.5 Conclusion 64
Chapitre 4 Mise à jour des connaissances 66
4.1 Introduction....66
4.2 Informations transmises par le serveur 68
4.2.1 Niveau de confiance....70
4.2.2 Probabilité de changement....71
4.2.3 Valeur de fluctuation du niveau de confiance 72
4.2.4 Représentation d'une brique 75
4.3 Divers exemples de calcul de dégradation de valeur de vérité 76
4.3.1 Exemple 1 : dégradation rapide 77
4.3.2 Exemple 2 : Accroissement du niveau de confiance 78

4.3.3 Exemple 3 : croissance du niveau de confiance 80
4.4 Fonction d'utilité 81
4.5 Conclusion 84
Chapitre 5 Mécanisme d'évaluation des valeurs de vérité associées aux connaissances 85
5.1 Introduction 85
5.2 Modèle statistique 86
5.3 Mécanisme d'évaluation de la valeur de vérité associée à une connaissance 88
5.4 Exemple 98
5.5 Analyse de la technique proposée 101
5.6 Contexte d'application 102
5.7 Conclusion 105
Chapitre 6 Techniques de mises à jour des connaissances 107
6.1 Introduction 107
6.2 La méthode synchrone 109
6.2.1 Validation à deux phases 109
6.3 La méthode asynchrone 110
6.3.1 Serveur paresseux 111
6.4 La méthode hybride 114
6.4.1 Technique par vote 114
6.5 Mise à jour des connaissances dans un système multiagent 115
6.6 Conclusion 117
Chapitre 7 Conclusion 119
7.1 Retour sur les objectifs 121
7.2 Travaux futurs 123
Bibliographie 124
Annexe A PRÉSENTATION DU MODE REPRÉSENTATION DES CONNAISSANCES 130
Introduction 130
Pourquoi les graphes conceptuels? 133
Notions de base 136
Indexation des graphes conceptuels 146
Une approche relationnelle pour les graphes conceptuels 150
Le canon 150
L'ensemble de types T 151
L'ensemble de référents I 152
La relation de conformité « :: » 153
La base canonique B 154
Les graphes conceptuels 156
Extension sémantique des graphes conceptuels 156
Conclusion 160
Annexe B 162

Liste des tableaux

Tableau 5-1 Valeur de l'évaluation des briques 1 et 2. 100
Tableau 6-1 Technique de propagation de mises à jour utilisée en industrie. 113

Liste des figures

Figure 1-1 Architecture du modèle client-serveur. 2
Figure 2-1 Un graphe qui représente : " Tom, un chat, poursuit Jerry, une souris ". 19
Figure 2-2 Le treillis de type T. 21
Figure 2-3 La projection de *h* sur *g*. *h* peut être interprétée comme suit : il existe une activité et un sport localisés sur le même lieu possédant un lac". 25
Figure 2-4 Représentation d'une brique. 30
Figure 2-5 Application de la fonction de translation s(g) sur un graphe conceptuel. 33
Figure 2-6 L'ensemble {g1, g2, g3 } constitue une couverture de g. 38
Figure 2-7 Le chat, Axel, couché sur un tapis. 45
Figure 2-8 Ensemble de briques 46
Figure 2-9 Mise en correspondance des briques. 47
Figure 2-10 Mise en correspondance des briques. 48
Figure 3-1 Description sommaire du fonctionnement de l'algorithme de recherche. 57
Figure 3-2 Comparaison entre les deux techniques de recherche telles que la taille de la base de données est variable alors que la taille de la requête est fixe. 62
Figure 3-3 Comparaison entre les deux techniques de recherche tels que la taille de la base de données est fixe alors que la taille de la requête est variable. 63
Figure 4-1 Estimation du niveau de confiance au temps t_3 74
Figure 4-2 Estimation du niveau de confiance au temps t_3 78
Figure 4-3 Estimation du niveau de confiance au temps t_3 79
Figure 4-4 Estimation du niveau de confiance au temps t_3 81
Figure 5-1 Exemple d'évaluation des briques d'une requête. 99
Figure 7-1 Représentation des connaissances. 131
Figure 7-2 Un graphe qui représente : " Tom poursuit Jerry ". 137
Figure 7-3 Le treillis de type T. 138
Figure 7-4 La copie et la simplification. 140
Figure 7-5 Restriction et élargissement. 140
Figure 7-6 Jointure et séparation. 141
Figure 7-7 La projection de *h* sur *g*. *h* peut être interprétée comme suit: « il existe une activité et un sport localisés sur le même lieu possédant un lac ». 142

Table des algorithmes

Algorithme 2-1 Fonction de translation.....32
Algorithme 2-2 Couverture d'un graphe 40
Algorithme 2-3 Rechercher les référents d'un graphe dans un ensemble de briques.....45
Algorithme 3-1 Recomposer un ensemble de briques.....53

Chapitre 1 Introduction

1.1 Problématique

Avec l'amélioration et la prolifération de la téléinformatique, les systèmes informatiques distribués connaissent un véritable essor dans l'industrie du logiciel. Un système distribué est un système pour lequel chaque composante est indépendante de toute autre et souvent située à un endroit géographique éloigné des autres composantes. De nos jours, les systèmes distribués sont même apparus en développement de logiciels dits « intelligents ». Dans un tel système, les composantes communiquent entre elles pour échanger des informations, coordonner leurs actions, prendre des décisions collectives, etc. Ces composantes essaient d'atteindre un but commun en s'aidant mutuellement. Le but ultime d'un tel système est de coordonner les actions des participants de manière à simuler un seul organisme même si les participants sont indépendants et possiblement géographiquement distribués [SKORN2004], [SKORN2005]. Généralement, dans ces systèmes, la coordination des participants se fait à l'aide d'une communication efficace permettant un échange d'informations et de connaissances nécessaires à l'accomplissement des tâches de chacun. Ce mémoire s'attarde donc principalement aux systèmes distribués intelligents ayant une mission coopérative. Il porte précisément sur le partage de connaissances entre les différentes composantes d'un tel système.

Pour que la communication soit efficace entre les participants d'un système distribué coopératif, plusieurs types d'architectures peuvent être envisagés. Donnons en

exemple l'architecture utilisée par les essaims de robots [SWISI2006], [SWISR2006], [SSWARM2006]. Cette architecture répartie les connaissances à travers l'ensemble des participants, c'est-à-dire que chaque participant possède une partie de l'ensemble des connaissances. Dans cette architecture chaque participant interroge les autres participants pour avoir les connaissances nécessaires à sa tâche. Un autre exemple d'architecture de communication est l'architecture des *tableaux noirs* [CUTK1993], [GENE1994], [DECK1997], [DCOR2003] et [SYC2003]. Dans cette architecture, les connaissances sont centralisées dans quelques participants qui ajoutent les connaissances inférées par les autres participants. L'accès aux connaissances se fait en accédant à la mémoire de ces participants sélectionnés pour contenir les connaissances de tout le système.

Ici, nous nous intéressons à une architecture où l'échange d'informations se fait selon un modèle appelé *client-serveur*. Dans ce modèle, nous appelons « serveur » un ordinateur ou une composante logicielle qui répond aux requêtes d'un ou de plusieurs clients. Par exemple, un serveur de fichiers fournit des données ou des fichiers à ses clients. Lorsqu'un client demande au serveur une donnée ou un fichier, le serveur lui en transfère une copie. La Figure 1-1 montre un exemple d'architecture client-serveur.

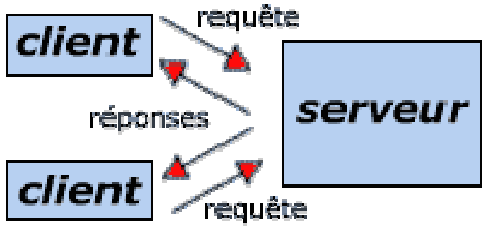

Figure 1-1 Architecture du modèle client-serveur.

Le modèle client-serveur est particulièrement recommandé pour des partages d'informations nécessitant un grand niveau de fiabilité. Ses principaux atouts sont :

- des ressources centralisées : le serveur est au centre du processus de communication, il peut gérer des ressources communes à tous les participants afin d'éviter les problèmes de redondance et de contradiction
- une meilleure sécurité : le nombre de points d'entrée permettant l'accès aux données est moins important. Le serveur peut contrôler les informations transmises aux participants
- une administration unique : les participants ont peu d'importance dans ce modèle. Ils ont moins besoin d'être administrés. L'administration des connaissances se fait presque entièrement sur le serveur.
- un réseau évolutif : grâce à cette architecture, il est possible de supprimer ou d'ajouter des clients sans perturber le fonctionnement du réseau et sans modification majeure.
- une meilleure fiabilité : en cas de panne, seul le serveur fait l'objet d'une réparation, pas le client.

L'architecture client-serveur a tout de même quelques lacunes parmi lesquelles on retrouve :

- la mise à jour : en cas de mise à jour des connaissances, le serveur doit informer les clients que les connaissances que ceux-ci possèdent sont peut-être périmées.
- le maillon faible : le serveur est le seul maillon faible du réseau client-serveur, étant donné que tout le réseau est architecturé autour de lui.

Un système distribué intelligent progresse généralement dans un environnement qui évolue lui aussi en temps réel. Par conséquent, les valeurs de vérité associées aux connaissances qui y sont manipulées peuvent également fluctuer dans le temps. Sans mécanisme de contrôle des mises à jour, les participants doivent exécuter une requête à chaque fois qu'ils utilisent une connaissance pour s'assurer que les informations qu'ils possèdent sont justes. Or, si le nombre de participants est élevé, le trafic généré par ce genre d'approche est catastrophique. Le serveur doit donc tenir compte de l'évolution des connaissances, de leur péremption et de la nécessité de les mettre à jour afin d'alimenter les clients avec de l'information (connaissance) toujours valide.

Ainsi, les systèmes qui utilisent l'architecture client-serveur doivent partager des informations malgré les trois problèmes ci-dessous, qui sont propres à ce genre d'architecture :

1. délai de transmission;
2. espace disponible dans toutes les composantes pour stocker la même information;
3. incohérence momentanée du système face à l'évolution de ces connaissances.

Pour pallier ces problèmes, nous proposons de créer une *mémoire collective* qui permet à un serveur de distribuer des connaissances à l'ensemble de ses clients en évitant les problèmes énumérés précédemment. Dans la prochaine section, nous présentons les principales motivations qui ont conduit à effectuer ce travail.

1.2 Motivations

Plusieurs motivations ont guidé le travail décrit dans ce mémoire. La principale motivation réside dans les applications éventuelles du modèle d'une telle mémoire collective. En effet, le modèle réalisé peut être mis en pratique dans un grand nombre d'applications distribuées dont le fonctionnement serait basé sur l'exploitation de connaissances qui fluctuent dans le temps.

Par exemple, les systèmes multiagents[1] [BDRIEU2001] où la mémoire collective permettrait d'homogénéiser les connaissances et de gérer la fluctuation des valeurs de vérité associées aux connaissances constituent un premier domaine d'application. Ceci permettrait aux systèmes multiagents de simuler, avec plus d'exactitude, une entité collective.

Aussi, dans le cas des moteurs de recherches Web où la mémoire collective permettrait de gérer les mises à jour des contenus des sites Internet référés en créant un mécanisme de gestion de la fluctuation des valeurs de vérité associées aux connaissances. Dans ce cas, on parle de serveur de connaissances supportant l'activité du Web sémantique [W3C2007].

Une autre source de motivation est l'élimination de deux contraintes généralement présentes dans les systèmes distribués coopératifs : 1) l'incohérence momentanée du

1 Relatif à une entité dans laquelle évolue un ensemble d'agents capables de coopérer et de communiquer entre eux, et susceptibles de modifier l'état de cette entité.

système et 2) la redondance de la connaissance dans l'ensemble des clients. Ainsi, nous espérons :

1) que l'application du modèle d'une mémoire collective permette une amélioration du délai de transmission des connaissances à l'ensemble des participants (clients);
2) que les informations transmises par le biais d'une mémoire collective soient uniformes pour l'ensemble des participants (clients) et éviter ainsi les problèmes d'incohérence momentanée et d'espace disponible chez les clients pour stocker ces informations;
3) que l'implantation d'une mémoire collective dans un système distribué intelligent permette autant aux clients qu'au serveur de gérer efficacement la mise à jour des connaissances dans un environnement temps-réel tout en minimisant le trafic généré.

Plus précisément, la fluctuation des valeurs de vérité associées aux connaissances ne doit pas générer de confusion lors de la réalisation de la tâche du client. Ainsi, ce mémoire vise à proposer une architecture client-serveur pour un système distribué qui œuvre dans un domaine où l'information à échanger est en fait la connaissance et où celle-ci est en constante évolution. La méthode proposée vise à minimiser les trois problèmes énumérés ci-haut.

1.2.1 Redondance et incohérence des connaissances

Dans les systèmes distribués coopératifs utilisant un modèle client-serveur, deux problèmes sont généralement présents : la redondance des connaissances à travers l'ensemble des clients et l'incohérence momentanée du système. Si ces deux problèmes pouvaient être résolus, une amélioration de la performance et une réduction du risque d'erreur de tout le système pourraient être réalisées.

Dans le modèle client-serveur, les informations ont tendance à être dupliquées dans la mémoire de chacun des clients. La duplication des informations survient lorsque le client et le serveur possèdent chacun leur propre mémoire. Si les clients et le serveur partageaient la même mémoire, on pourrait minimiser la mémoire nécessaire à chacun des clients pour emmagasiner les connaissances. Du même coup, on minimiserait les coûts de mise à jour et le risque d'incohérence momentanée du système. En effet, lors de la mise à jour des informations sur le serveur, les informations que possèdent les clients sont momentanément incohérentes avec celles du serveur; une mise à jour des informations de chacun des clients doit être effectuée pour assurer la conformité des informations. Encore une fois, si le serveur partageait la même mémoire que l'ensemble des clients, la mise à jour des informations se ferait simultanément dans le serveur et les clients. Toutefois, si le serveur et le client partageaient la même mémoire, il y aurait beaucoup plus de trafic sur le réseau de communication qui les lie. Ainsi, nous proposons donc une instanciation d'une mémoire collective pour laquelle nous minimisons l'utilisation de la bande passante utilisée par la synchronisation des connaissances entre les clients.

1.2.2 Les applications

Les applications possibles d'une mémoire collective sont présentes dans tous les systèmes intelligents qui utilisent une architecture client-serveur pour lesquels cette connaissance varie dans le temps (c.-à-d., est instable). Dans ce qui suit, nous présentons succinctement les applications citées ci-dessus, soit les systèmes multiagents et les moteurs de recherches Web.

1.2.2.1 Les systèmes multiagents

Dans les systèmes multiagents, les agents communiquent entre eux à l'aide de messages plus ou moins sophistiqués, dans le but d'assurer la coopération, voire la coordination du groupe. Ils peuvent s'échanger de l'information sur l'environnement dans le but d'augmenter leurs perceptions individuelles ou se transmettent leurs intentions pour que les agents puissent avoir une idée de ce que tout un chacun fait ou a l'intention de faire. Un système multiagent se distingue d'une collection d'agents indépendants par le fait que les agents interagissent en vue de réaliser conjointement une tâche ou d'atteindre conjointement un but particulier. Les agents doivent bien se coordonner entre eux, sinon la coopération risque de ne pas porter fruit.

La compétition de soccer entre robots, la RoboCup [SROBO2006], est un exemple de système multiagent. Pour marquer des points, les robots doivent se coordonner pour réaliser des stratégies d'équipe. Depuis 1998, les principales évolutions de la RoboCup ont touché les aspects de la coopération et de la représentation des connaissances [BDRIEU2001]. L'architecture que nous proposons permettrait d'améliorer[2] la qualité des inférences et l'accessibilité des connaissances au sein de la mémoire collective, ce qui accélérerait le processus de raisonnement et donc, le jeu d'équipe des robots.

1.2.2.2 Les moteurs de recherches Web

Le Web est un système hypertexte public fonctionnant sur Internet et permettant de consulter, avec un fureteur, des pages Web mises en ligne. Les pages Web sont stockées sur des serveurs Web, c'est-à-dire des machines connectées à Internet en permanence et chargées de fournir les pages Web demandées. Chacune des pages Web, et plus généralement toute ressource en ligne, est repérée par une adresse unique appelée URL (Uniform Resource Locator). Pour trouver l'URL correspondant à l'information qu'on cherche, on utilise un moteur de recherche [REWISI2006]. L'utilisateur formule une requête au moteur de recherche et celui-ci regarde, dans sa banque d'URL, celles qui correspondent le mieux à ce que l'utilisateur recherche. Le résultat du moteur de recherche est l'ensemble des sites Web dont le contenu correspond le mieux à la requête de l'utilisateur. L'ensemble des sites Internet résultant de la requête est ensuite organisé selon

[2] Sous certaines conditions qui seront énumérées dans les chapitres 2 et 3 de ce mémoire.

un certain algorithme de classification. Pour que le résultat du moteur de recherche soit pertinent, il faut que la banque d'URL du moteur de recherche soit constamment mise à jour. En effet, non seulement il y a de nouveaux sites Web qui s'ajoutent à toutes les secondes, mais, en plus, le contenu des sites déjà présents dans la banque évolue lui aussi. Ainsi, les sites Web qui correspondent le mieux à la requête de l'utilisateur évoluent avec le temps. Le moteur de recherche doit donc sans cesse mettre à jour sa banque d'URL pour s'assurer de présenter des résultats pertinents à l'utilisateur.

Dans cette application, on reconnaît l'architecture client-serveur. Le Web représente le serveur alors que les moteurs de recherches représentent les clients. Le modèle que nous proposons permettrait d'améliorer la fraicheur de l'information pour les moteurs de recherches et du même coup la pertinence des résultats fournis par ceux-ci.

1.3 Objectifs

La section 1.1 a présenté la problématique à laquelle ce mémoire s'intéresse. Par la suite, nous avons présenté, dans la section 1.2, les motivations qui nous ont poussés à réaliser ce travail. Au cours de cette section, nous présentons les objectifs de ce mémoire. Enfin, dans les prochaines sections, nous allons expliquer l'approche que nous avons préconisée pour atteindre ces objectifs.

Rappelons-le, le présent mémoire s'intéresse particulièrement aux systèmes distribués coopératifs ayant une architecture client-serveur et dont le fonctionnement est basé sur le traitement de connaissances qui évoluent dans le temps. La création d'une mémoire collective au service des composantes de tels systèmes constitue le principal objectif de cette maîtrise. La réalisation de cet objectif vise à minimiser les contraintes de l'architecture client-serveur expliquées dans la problématique (voir la section 1.1).

Pour concevoir une mémoire collective, nous avons identifié trois aspects qui doivent être réalisés : la représentation des connaissances utilisée par le système, l'accès à ces connaissances et la mise à jour des connaissances**.** Pour chacun des aspects énumérés précédemment, nous avons identifié des contraintes qui ont guidé les choix faits lors de l'élaboration de la solution que nous présentons dans ce mémoire. Ces contraintes délimitent le type de systèmes pour lequel notre solution s'appliquera. Les prochains chapitres discuteront des avantages et inconvénients d'avoir effectué ces choix.

1. Le mode de représentation des connaissances utilisé doit être expressif, permettant au processus de recherche de gagner en efficacité et en précision (voir le Chapitre 2), réduisant ainsi le nombre de réponses inutiles. Le but est donc de représenter les connaissances à l'aide d'un modèle qui permet une diffusion adéquate des connaissances.
2. Il faut définir une structure d'accès qui permette un accès simultané à la connaissance par l'ensemble des clients, sans engorger le serveur. La requête d'un client ne doit pas monopoliser le serveur.

3. Il faut définir un mécanisme de gestion des mises à jour des valeurs de vérité associé aux connaissances. Ce mécanisme doit permettre de faire une mise à jour continuelle de ces valeurs de vérité étant donné que certaines applications visées se déroulent en temps réel dans un contexte évolutif.

Au cours de la prochaine section, nous présentons l'approche proposée pour la réalisation de ce mémoire.

1.4 Approche proposée

L'objectif de cette maîtrise est de modéliser une mémoire collective. Pour atteindre cet objectif, nous avons défini les aspects importants à modéliser. Il faut définir un mode de représentation des connaissances qui respecte les sous-objectifs spécifiques énoncés précédemment à la section 1.3. Ensuite, il faut prévoir un moyen d'accéder à cette connaissance et de la mettre à jour dans un environnement où elles évoluent dans le temps. Voici donc les approches utilisées pour réaliser chacun des sous-objectifs :

1. *L'accès à la connaissance* est un aspect important d'une mémoire collective. Dans le modèle proposé, les connaissances sont centralisées sur le serveur. La mémoire centrale doit être disponible simultanément à l'ensemble des clients sans qu'il y ait d'engorgement dans le réseau. Le modèle proposé prévoit donc un moyen de diffuser simultanément aux clients les connaissances du système.

Le principe adopté est semblable à la télédiffusion : les informations sont transmises sur plusieurs canaux et l'utilisateur sélectionne le canal qui lui convient en fonction des informations qu'il désire recevoir. Les informations diffusées sur le réseau correspondent à la mémoire des clients. Lorsque les clients ont besoin de leur mémoire, ils accèdent directement à la mémoire collective présente sur le réseau. La connaissance est la même partout puisque les clients partagent une mémoire collective. Ils ont donc accès à la même version de l'information. Ainsi, la mémoire dite collective est disponible simultanément à l'ensemble des clients sans que le serveur ait besoin de tenir compte de l'espace disponible dans toutes les composantes pour stocker les informations.

2. *Le mode de représentation des connaissances* doit être suffisamment riche. Pour être riche, ce mode de représentation des connaissances doit :
   a. offrir de grandes possibilités pour exprimer la sémantique des connaissances. L'expressivité du mode de représentation des connaissances doit permettre de mieux définir les connaissances et, par conséquent, améliorer la qualité des réponses lors de l'interrogation des connaissances. On veut ainsi minimiser les transmissions d'informations inutiles au client;
   b. permettre de représenter et raisonner sur des connaissances complexes et structurées;
   c. faciliter la distribution des connaissances à travers un réseau;
   d. être facile à implémenter;

e. permettre d'adapter facilement le modèle à une nouvelle situation. Le mode de représentation des connaissances doit être extensible dans le but d'ajouter éventuellement de nouvelles fonctionnalités sans devoir restructurer la mémoire collective.

Ainsi, nous voulons que le mode de représentation des connaissances soit très expressif et adapté à nos besoins d'implantation. Le chapitre 2 présente le mode de représentation des connaissances que nous avons choisi.

3. *La mise à jour des connaissances* entre les clients et le serveur. Puisque le système distribué évolue dans un environnement temps-réel, la mémoire collective doit prendre en compte le fait que les connaissances qu'elle possède peuvent ne pas être stables. Ainsi, des altérations des valeurs de vérité attribuables aux connaissances peuvent survenir en tout temps puisque le système est évolutif. Le modèle proposé prévoit un moyen de mettre à jour les connaissances dans un environnement temps-réel de façon à minimiser le trafic généré. Ainsi, notre modèle propose que le serveur attribue des valeurs de vérité continues aux connaissances diffusées et de faire fluctuer celles-ci dans le temps pour simuler l'instabilité des connaissances. Pour éviter que le client ait à faire continuellement des mises à jour de ces valeurs de vérité le modèle propose un mécanisme qui lui permet de décider quand rafraîchir ses connaissances (tel que présenté dans le Chapitre 4 et le Chapitre 5). Pour accomplir cette tâche, nous proposerons (voir le Chapitre 5) d'intégrer des valeurs statistiques qui permettent d'évaluer le moment de péremption des connaissances.

1.5 Organisation du mémoire

Ce mémoire est organisé comme suit. Tout d'abord le Chapitre 2 présente le mode de représentation des connaissances utilisé ainsi que les adaptations que nous avons apporté à celui-ci pour mieux répondre à nos besoins. Dans ce chapitre, nous définissons un index sur le mode de représentation des connaissances choisi. Dans le chapitre suivant (Chapitre 3), nous présentons une technique de recherche à travers le mode de représentation des connaissances que nous avons défini au chapitre précédent. Cette technique permet de faire le lien entre la requête d'un client et les éléments dans la base de connaissances. Dans ce chapitre, les résultats d'une expérimentation de cette technique comparativement à une technique existante seront présentés.

Dans le Chapitre 4, nous présentons une technique qui permet de maintenir à jour les connaissances des clients tout en minimisant le transfert de connaissances. Cette technique, basée sur une analyse stochastique des connaissances, permet de déterminer les moments où les connaissances subiront une altération de leurs valeurs de vérité et ainsi prévoir les mises à jour. Enfin, le Chapitre 5 présente un algorithme qui implémente la méthode proposée au chapitre précédent. Dans ce chapitre, nous proposons un exemple d'utilisation de notre méthode ainsi que les critères nécessaires à l'implantation de cette méthode. Finalement, le Chapitre 7 conclut ce mémoire et présente quelques avenues possibles pour la suite du projet.

Chapitre 2 Approche proposée pour la représentation et l'accès aux connaissances

2.1 Introduction

Au chapitre précédent, nous avons présenté les prémisses de réalisation de ce mémoire. Nous avons détaillé les objectifs qui sous-tendent cette maîtrise, soit la représentation des connaissances, l'accès à la connaissance et la mise à jour des connaissances en vue de construire et de rendre disponible une mémoire collective. Dans ce chapitre, nous allons expliquer l'approche utilisée pour accéder et représenter les connaissances tout en respectant les prémisses formulées au Chapitre 1. Par la suite, le Chapitre 3 détaille l'implémentation du mode de représentation des connaissances utilisé par notre modèle. De plus, le Chapitre 3 présente une expérimentation qui compare la rapidité d'exécution de notre méthode avec celle d'un logiciel existant.

La section 2.2 de ce chapitre est dédiée au mode de représentation des connaissances utilisées pour implémenter la mémoire collective. Dans cette section, nous montrons comment, à partir d'une représentation sémantique, nous pouvons implémenter le modèle de mémoire collective qui atteint les objectifs d'accès et de représentation des connaissances décrits au Chapitre 1. Enfin, la section 2.3 présente une brève conclusion de la méthodologie proposée.

2.2 Représentation et accès aux connaissances

Afin de faciliter ou permettre certaines recherches et manipulations de connaissances par un système, les connaissances de ce système doivent avoir été représentées puis indexées. Durant nos recherches, le premier choix auquel nous avons été confrontés a été de sélectionner un langage de représentation des connaissances en fonction de nos objectifs. Pour les raisons énumérées à l'Annexe A, nous avons adopté le formalisme des *graphes conceptuels*. Brièvement, les avantages de ce formalisme de représentation des connaissances sont la généralité, l'existence de standards, l'expressivité, la rapidité de traitements de certaines inférences et les potentialités d'interaction personne-machine. Par contre, les contraintes imposées au choix d'un tel langage demeuraient présentes peu importe le choix du langage; elles sont aussi décrites dans l'Annexe A. En bref, les contraintes de ce formalisme sont la lourdeur du modèle et la complexité algorithmique de l'opération de recherche.

La prochaine section (section 2.2.1) présente sommairement les éléments importants des graphes conceptuels qui seront nécessaires à la compréhension du reste du chapitre. Pour plus d'informations concernant les graphes conceptuels, le lecteur est invité à consulter l'Annexe A. La section 2.2.2 présente une codification des graphes conceptuels pour que ceux-ci soient plus facilement transmissibles sur un réseau de communication. La section 2.2.3 montre qu'il est possible de passer de cette codification à la représentation structurelle des graphes conceptuels. De cette façon, les raisonnements peuvent être basés

sur les graphes conceptuels et l'accès à la connaissance peut être fait à partir de cette nouvelle forme présentée à la section 2.2.2. La section 2.2.4 présente brièvement la technique de recherche utilisée dans la mémoire collective. La section 2.2.5 présente l'interprétation du résultat d'une recherche dans la mémoire collective afin de déterminer la réponse à la requête d'un client. Dans cette section, nous définissons les conditions sous lesquelles le résultat de la requête d'un participant peut être considéré comme vrai ou faux.

2.2.1 L'essentiel des graphes conceptuels

Les graphes conceptuels sont présentés comme un modèle *général* d'écriture de réseaux sémantiques pour la représentation des connaissances [SOW1984]. Cet aspect de généralité est au cœur des cinq avantages du formalisme que la communauté des graphes conceptuels se plaît à rappeler : généralité, existence de standards, expressivité, rapidité de traitements et le potentiel d'interaction personne-machine.

Tout d'abord, nous définissons un graphe conceptuel comme étant un graphe fini, orienté et biparti, non nécessairement connexe[3]. Les deux types de nœuds composants le graphe conceptuel sont les nœuds *concepts* et les nœuds *relations*. Les nœuds concepts représentent les entités, les attributs, les états et les événements alors que les nœuds relations représentent les liens qui unissent ces concepts. Les concepts sont composés de deux parties : le *type* et le *référent*. Les relations, quant à elle, possèdent seulement un type.

[3] Historiquement, les graphes conceptuels étaient connexes [Sowa1984].

Le type représente la classe à laquelle se rattache le concept (relation) exprimé par rapport à un domaine d'application, tandis que le référent est vu comme une instanciation du type de concept. Les relations sont rattachées à un nombre *n* de concepts où *n* représente l'arité de la relation. Pour s'assurer que les concepts rattachés à la relation sont bien conformes à la syntaxe de celle-ci, une *signature de relation* est définie. Une signature de relation de type *t* et d'arité *j* est un tuple de *j* concepts. Chaque élément *i* du tuple représente le type de concept auquel le $i^{ème}$ concept rattaché à la relation doit se conformer.

Un exemple de graphe conceptuel est donné dans la Figure 2-1 suivante :

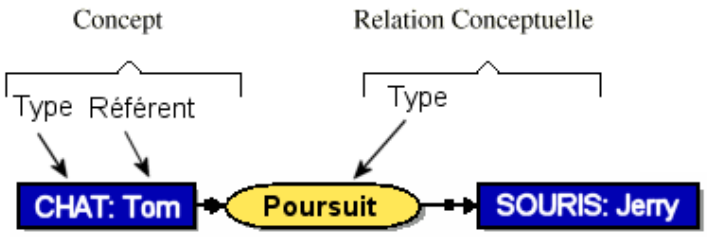

Figure 2-1 Un graphe qui représente : " Tom, un chat, poursuit Jerry, une souris ".

L'ensemble des référents *I* est défini par $I = M \cup X \cup \{\forall\}$. L'ensemble *I* représente les instances des types de concepts. Lorsque le référent universel ($\forall$) est utilisé dans un nœud concept, ce référent fait référence à toutes les instances conformes à ce type de concept. Contrairement au référent universel ($\forall$), les éléments des ensembles *M* et *X* représentent un seul référent d'un type de concept. Ainsi, l'ensemble *M* représente les référents constants du système, soit les éléments connus à l'exécution, alors que l'ensemble *X* représente les référents variables qui ne sont pas connus lors de l'exécution, mais que l'on

sait existants. Ainsi, nous disons que l'ensemble des référents conformes à un type de concept forme *l'extension* de ce dernier ou sa *dénotation*. De plus, les référents ont une portée globale par rapport au système, c'est-à-dire qu'un référent défini pour représenter une instance ne peut être redéfini pour en représenter une autre.

Les types de concepts T_c sont organisés selon une hiérarchie des types qui constitue un treillis fini. Ce treillis est muni d'une relation d'ordre partiel ≤, définie de la façon suivante: si la dénotation d'un type de concept c_1 est incluse dans la dénotation du type de concept c_2 alors $c_2 \leq c_1$ dans T_c. De plus, la relation d'ordre partiel ≤ vérifie les propriétés de réflexivité et de transitivité. Par exemple, Homme ≤ Personne. Nous disons que Homme est une restriction (sous-type) de Personne, et que Personne est une généralisation de Homme. Les types de relations, dénotés par l'ensemble T_r, sont aussi classés dans une structure de treillis. Dans ce cas, ce sont les signatures des relations qui sont organisées dans T_r. Par exemple, la Figure 2-2 présente un treillis (incomplet) des types. La partie de treillis encerclée en rouge représente les types de concepts alors que la partie encerclée en bleu représente les relations binaires (ayant deux éléments dans la signature). Les relations d'arité différentes seraient représentées dans un autre cercle à droite du cercle bleu, de telle sorte qu'elles ne sont pas comparables avec les relations d'arités différentes.

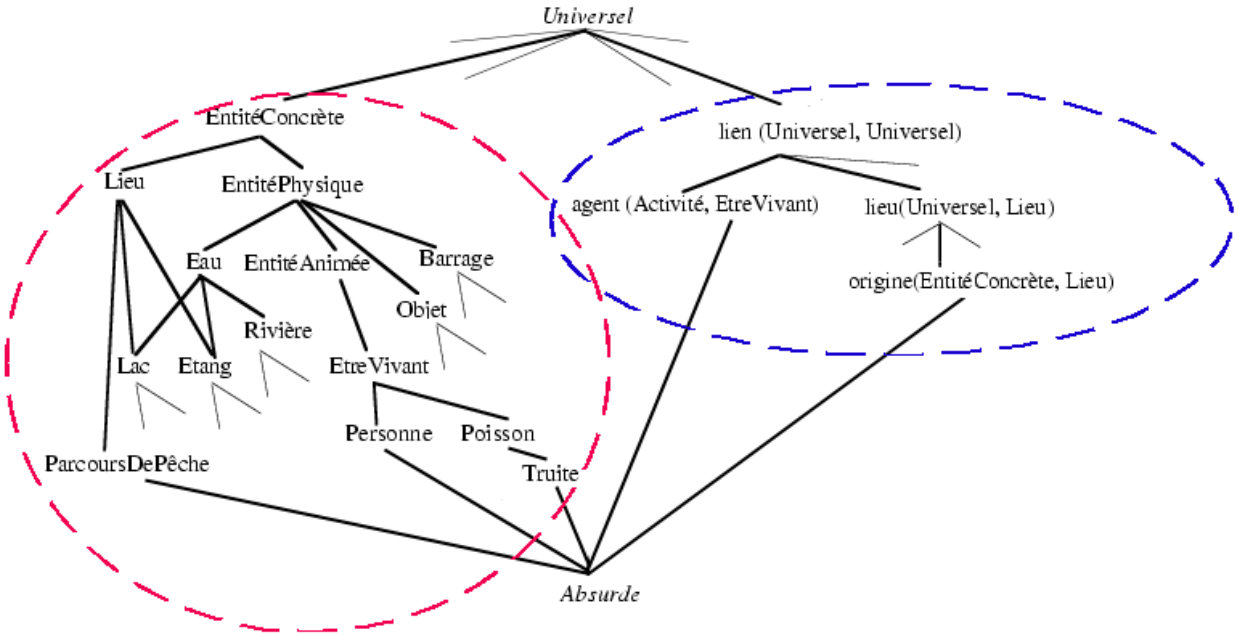

Figure 2-2 Le treillis de type T.

Ces éléments de base étant posés, Sowa introduit la notion de graphe conceptuel:

> **Définition 2-1** (*Graphe conceptuel*) Un graphe conceptuel est formé d'un ensemble de concepts et d'un ensemble de relations tel que chaque relation est reliée à un ensemble de concepts conformes à sa signature, en nombre égal à son arité. Chaque concept possède des référents qui appartiennent au type du concept. Un concept seul est aussi considéré comme un graphe conceptuel.

Ainsi, nous représentons un graphe conceptuel de la façon suivante :

Définition 2-2 (*Représentation d'un graphe conceptuel*) Un graphe conceptuel g est une structure représentée sous la forme d'un 4-uplet < C, R, E,l > [MIN2000] composé de :

- Un ensemble C de concepts.
- Un ensemble R de relations.
- Un ensemble E d'arcs qui relient les relations et les concepts tels que $E \subseteq C \times R$.
- Une fonction d'étiquetage l tel que : chaque concept $c \in C$ est étiqueté avec un tuple $l(c) = (type(c), ref(c)) \in T_c \times I$ appelé type et référent. Chaque relation $r \in R$ est étiquetée avec un type de relation $l(r) = type(r) \in T_r$.

Dans notre modèle, pour qu'un graphe soit valide, il faut, en plus qu'il respecte une structure où les référents sont uniques, appelée la *forme normale* [MUG1993b]. La forme normale d'un graphe est obtenue en fusionnant les nœuds concepts ayant le même marqueur individuel $\in M \cup X$ ou ayant $\forall$ comme référent. 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.

Définition 2-3 (*Forme normale)* Un graphe conceptuel $g = <C, R, E, l>$ est en forme normale si pour n'importe quel concept c_1 et $c_2 \in C$ où $ref(c_1) \neq \forall$, $ref(c_1) \neq ref(c_2)$ et si $ref(c_1) = \forall$, $\exists\, c_2$ où $ref(c_2) \neq \forall$ et $type(c_1) \neq type(c_2)$.

Pour réaliser la forme normale, nous utilisons l'opérateur de jointure maximale. Une jointure maximale correspond à une série de jointures, non pas sur un seul nœud, mais sur un sous-graphe commun. Cette opération s'exprime par une composition de jointures et de simplifications. Le résultat de cette opération forme un graphe normalisé.

Définition 2-4 (*Jointure maximale)* Soit le graphe conceptuel $g = <C_1, R_1, E_1, l_1>$. L'opération de jointure maximale

1. joint tous les nœuds compatibles, qui sont toutes paires de nœuds concept $c_1 \in C_1$ et $c_2 \in C_1$ tel que $ref(c_1) = ref(c_2)$. $\neq \forall$ ou $ref(c_1) = ref(c_2)$. $= \forall$ et $type(c_1) = type(c_2)$,
2. simplifie g en éliminant les relations de même type connectées dans le même ordre, aux mêmes concepts.

Pour assurer que la forme normale soit respectée, l'opérateur de jointure est appliqué sur chacun des graphes. La forme normale évite donc des ambiguïtés sémantiques et logiques dans les graphes conceptuels. Sans la forme normale, deux graphes peuvent avoir la même interprétation logique sans avoir le même graphe qui le représente. Dans le cas de la mémoire collective, nous voulons éviter les doublons de graphes pour minimiser la recherche et homogénéiser les connaissances. C'est pour cette raison que nous avons besoin de la forme normale.

Pour structurer les graphes conceptuels, Sowa introduit alors un ordre partiel ≤ sur les graphes (relation à ne pas confondre avec la relation d'ordre sur les types). Cet ordre

définit une relation d'inférence sur les graphes permettant de déterminer si les informations représentées par un graphe induisent les informations représentées 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. Cette relation peut aussi être définie par l'opérateur de projection, considéré comme un morphisme de graphes bipartis (les nœuds concepts sur les nœuds concepts, les nœuds relations sur les nœuds relations):

Définition 2-5 (*Projection*) Il existe une projection π d'un graphe h sur un graphe g seulement si :

- pour chaque concept c de h, $\pi(c)$ est soit une spécialisation de c, soit identique à c.
- pour chaque relation r de h, $\pi(r)$ 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 $\pi(r)$ doit être lié à $\pi(c)$ dans g.

Le résultat de la projection n'est pas toujours unique: le graphe g peut avoir plusieurs sous-graphes différents vérifiant la condition de projection. La Figure 2-3 montre un exemple de projection. On dit également que g est une spécialisation de h ou que h est une généralisation de g.

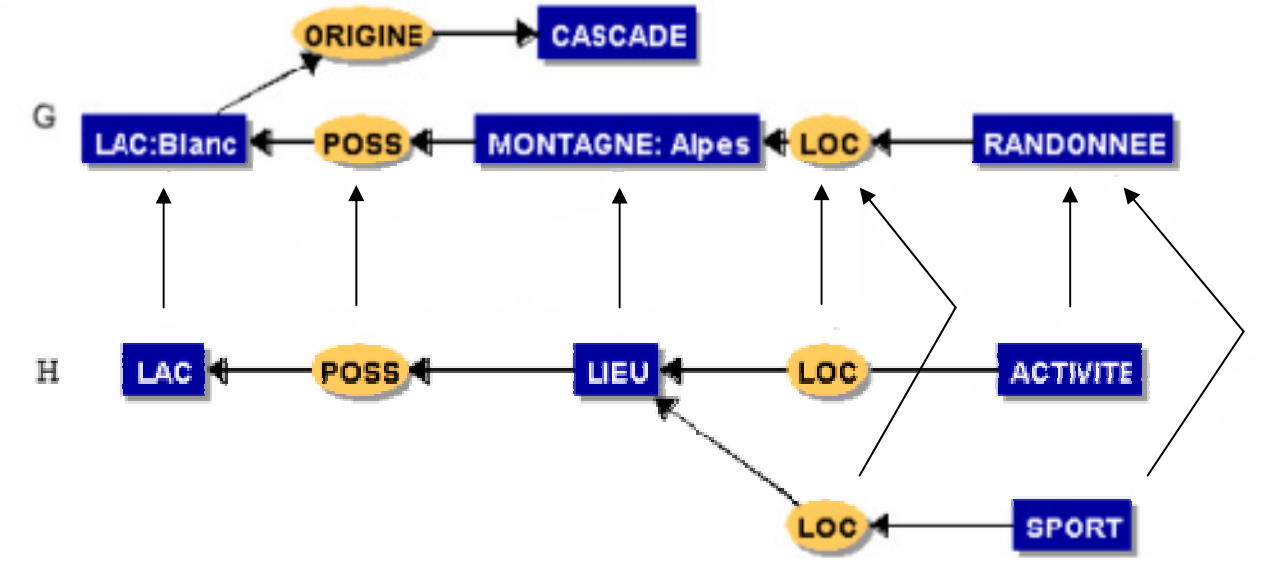

Figure 2-3 La projection de *h* sur *g*. *h* peut être interprétée comme suit : il existe une activité et un sport localisés sur le même lieu possédant un lac".

Sowa prouve que si un graphe *g* est une spécialisation d'un graphe *h*, alors il existe une projection de *h* sur *g*. De plus dans [CM1992], il est démontré que, s'il existe une projection de *h* sur *g*, alors $g \leq h$. Malgré de nombreux travaux ayant trait à l'étude de la complexité de la projection dans le formalisme des graphes conceptuels [MUG1993a], [MUG1993b], [PC1995], [GUI1996], la classe de complexité de cette dernière n'est pas encore exactement connue. Tout au plus, certains algorithmes polynomiaux ont été proposés pour des graphes conceptuels de type particulier (par exemple le cas où les graphes correspondent à des arbres [MUG1993a]). Cependant, à l'heure actuelle, il n'existe pas d'algorithme polynomial traitant de la projection d'une façon générale sur n'importe quel type de graphe conceptuel.

Comme on le verra plus tard, cet opérateur de projection est très important dans le processus de recherche fondé sur les graphes conceptuels. La projection est en fait la

méthode de mise en correspondance entre la requête formulée par l'utilisateur et la base de connaissances. L'existence d'une projection d'un graphe vers un autre signifie que ce graphe est l'une des réponses à la question formulée par l'utilisateur.

Dans un système de graphes conceptuels, l'existence même d'un graphe implique qu'il existe au moins un arrangement de référents pour lequel ce graphe est vrai. La véracité d'un graphe est donc dépendante de l'existence d'instances qui réalisent la sémantique du graphe. Pour déterminer les référents affectés à un graphe conceptuel *u*, nous définissons une fonction $\delta(u)$ qui retourne tous les tuples de référents pour lesquels *u* s'évalue à vrai lorsque ses concepts sont instanciés par ces référents. Ainsi, la recherche d'informations passe par cet opérateur qui est défini comme suit :

Définition 2-6 (*Support d'un graphe*) Le support d'un graphe conceptuel $g = <C_1, R_1, E_1, l_1>$, noté $\delta(g)$, est l'ensemble de tous les tuples $\{<i_{1,1}, i_{1,2}, \ldots, i_{1,|C1|}>, \ldots, <i_{2,1}, i_{2,2}, \ldots, i_{2,|C1|}>, \ldots <i_{n,1}, i_{n,2}, \ldots, i_{n,|C1|}>\}$ avec $|\delta(g)| = n$ où chaque $i_{x,j}$ $\forall x \in [1,n]$ instancie le concept $c_j \in C_1$ de *g* et où l'instanciation de tous les concepts par le tuple $<i_{x,1}, i_{x,2}, \ldots, i_{x,|C1|}>$ forme un graphe qui existe dans la base de connaissances $\forall x \in [1,n]$.

Pour qu'un graphe conceptuel soit vrai dans la base, il faut qu'il existe au moins un tuple de référents résultant de $\delta(g)$ pour lequel *u* est vrai. Nous disons donc que l'*extension sémantique* d'un graphe conceptuel est définie comme suit :

Définition 2-7 (Évaluation d'un graphe) Formellement, la valeur de vérité d'un graphe conceptuel *g* est défini par la fonction valeur(g) := ($\delta(g) \neq \varnothing$).

Dans cette section, nous avons présenté une brève description du formalisme des graphes conceptuels. Nous allons maintenant voir comment nous avons appliqué ce modèle de représentation des connaissances à notre proposition de modèle de mémoire collective.

2.2.2 La manipulation des connaissances

Le modèle de la mémoire collective évolue dans un environnement distribué où l'on tente de minimiser le plus possible les temps de recherche et de transmission. Or, le temps de recherche et de transmission des connaissances est considérable si l'on doit : 1) rechercher les connaissances séquentiellement à travers l'ensemble des connaissances; 2) télécharger la totalité des connaissances pour avoir la réponse à une requête qui peut ne concerner qu'une partie de la base. Ces deux points sont en opposition avec un de nos objectifs de départ; c'est-à-dire que le mode de représentation des connaissances choisi permet une recherche efficace et facilite la transmission des résultats. Pour ces raisons, il vaudrait mieux indexer les structures de connaissances emmagasinées dans la mémoire collective. Il convient donc de définir une fonction de translation (d'indexation) entre les structures de connaissances et le type d'index approprié aux services subséquents que doit rendre une mémoire collective.

La fonction de translation que nous avons envisagée subdivise une structure de connaissances en un ensemble de sous-structures. Cet ensemble de sous-structures contient les mêmes connaissances que la structure d'origine, mais subdivisée de façon à en faciliter l'indexation et la manipulation. Étant donné qu'on désire que chaque sous-structure soit directement indexable, chaque élément résultant de la fonction de translation représentera la plus petite unité d'information qui soit, appelée *brique*. Inversement, nous devrons avoir une fonction de translation inverse qui permet de reconstituer la structure d'origine à partir d'un ensemble de sous-structure. De cette façon, la décomposition d'une requête en sous-structure permettra, grâce à l'index, d'identifier les sous-structures de la mémoire collective directement concernées par la requête, et uniquement celles-ci.

Dans ce mémoire, tel que mentionné plus haut, nous appelons une *brique* une sous-structure qui représente la plus petite unité d'information composant une structure de connaissance. Elle correspond à une structure sémantique contenant : a) soit un concept seul qui représente un élément du système b) soit deux concepts reliés par une relation sémantique. Pour unifier la représentation des briques, un concept seul est automatiquement relié à une relation binaire de type universel. Ainsi, grâce à la fonction de translation, toutes les connaissances qui peuvent être exprimées par le système sont représentées par un ensemble de briques ainsi définies. Nous définissons donc formellement la fonction de translation de la façon suivante :

> **Définition 2-8** (*Fonction de translation*) Pour toute structure de connaissance g, une fonction de translation $s(g)$ est définie tel que $s(g) = \{P \mid P \subseteq \wp(G^1)\}$ où G^1 est l'ensemble de toutes les briques

possibles. Le choix de l'ensemble P parmi l'ensemble $\wp(G^l)$ est défini plus bas (Définition 2-9).

Inversement, la fonction s^{-1} fait correspondre l'ensemble $\wp$ résultant de $s(g)$ vers une structure de connaissance g' équivalente à g, c'est-à-dire, si $s(g) = P \subseteq \wp(G^l)$, alors $s^{-1}(P) = g' = s^{-1}(s(g))$. En terme de la théorie des graphes conceptuels, on dira que g' est le résultat de la jointure maximale de P.

Tel que mentionné plus haut, la fonction de translation relie un graphe vers un ensemble de briques $P \in \wp(G^l)$. Cet ensemble de briques P qui caractérise un graphe g est défini comme suit :

Définition 2-9 *(Éclatement)* L'éclatement d'un graphe conceptuel $g = <C_1, R_1, E_1, l_1>$ donne un ensemble de briques $P \in \wp(G^l)$ tel que P est une collection de briques. Ainsi, pour chaque relation $r \in R_1$, si le concept $c \in C_1$ rattaché à r possède un *degré(c)* supérieur à un alors on duplicate le concept c vers c' en conservant tous les arcs qui y sont connectés. On élimine ensuite l'arc *(r,c)* et on crée l'arc *(r,c')*. Dans le cas où il existe une composante connexe de g qui n'a pas de relation, on crée une relation de type universel (T) que l'on relie au concept tel que montré par la Figure 2-4.

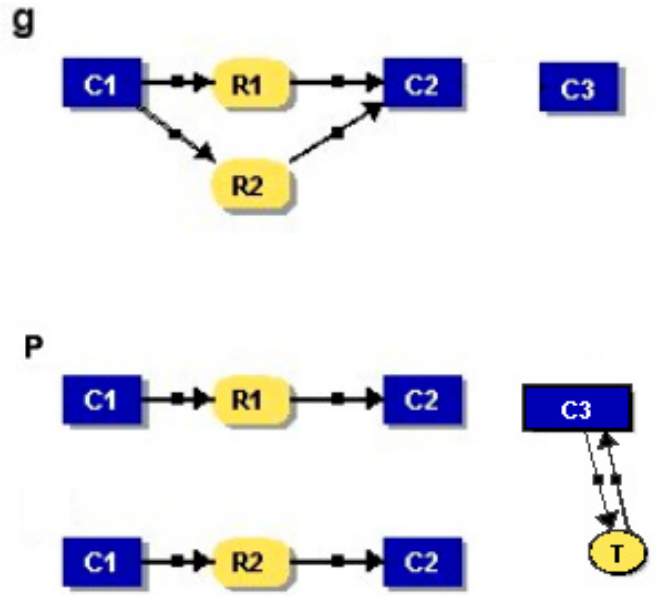

Figure 2-4 Représentation d'une brique.

L'idée principale de la décomposition d'une connaissance *k* est de pouvoir : *1) reproduire une forme normale de k, 2) de la décomposer afin de pouvoir régénérer la même sémantique qu'encodée à l'origine, 3) de l'encoder de façon à ce que certaines propriétés de manipulation soient respectées, même facilitées* :

- $k_1 \equiv k_2 \Rightarrow s(k_1) \equiv s(k_2)$
- $k_1 \subseteq k_2 \Rightarrow s(k_1) \subseteq s(k_2)$
- $((k_1 \cup k_2) \equiv k_3) \Rightarrow ((s(k_1) \cup s(k_2)) \equiv s(k_3))$
- $((k_1 \cap k_2) \equiv k_3) \Rightarrow ((s(k_1) \cap s(k_2)) \equiv s(k_3))$

Si la fonction de translation présentée dans la Définition 2-8 respecte ces propriétés, alors la manipulation des connaissances conserve la même sémantique dans les deux ensembles de représentation : l'univers des structures de connaissances et l'univers des briques. Les preuves de l'obtention de ces propriétés sont présentées à l'Annexe A.

D’un point de vu pratique (algorithmique), la fonction de translation fait l’éclatement d’une structure en un ensemble de sous-structures (briques). En terme de graphes conceptuels, l'éclatement de n’importe quel graphe conceptuel $g = <C_1, R_1, E_1, l_1>$ donne un ensemble de sous-graphes conceptuels connexes *P*. Pour implanter cette translation, nous utilisons Algorithme 2-1 présenté ci-dessous. Cet algorithme crée un nouveau graphe conceptuel *h* pour chacune des relations de *g*. Chaque graphe *h* possède au moins une relation et les concepts rattachés à cette relation dans *g*. S’il existe une composante connexe dans *g* qui possède un seul concept, alors le graphe *h* qui lui correspond est composé de ce concept plus une relation binaire de type universel qui lui est reliée.

```
Algorithme 1 (Fonction de translation)
s(un graphe conceptuel g)
// Soit g = <C1, R1, E1, L1> un graphe conceptuel sous forme normale
DÉBUT
      P = ∅
      Pour toutes composantes connexes h = <C2, R2, E2, L2> ∈ g FAIRE
      Début
         SI |R2| = 0 ET |C2| = 1 ALORS
         Début
            Soit g' = <C', R', E', L'> un nouveau graphe
            tel que C' =  R'= E'= L'= ∅
            Soit c' ∈ C2 et r' une nouvelle relation
            C' = c'
            L'(c) := (Type(c'), ref(c'))
            R' =  r'
            L(r') = (⊤)
            E = <r',c'> ∪ <c', r'>
            P = P ∪ g'
         Fin
         Sinon
               POUR TOUS ri ∈ R2 FAIRE
               DÉBUT
                  Soit g' = <C', R', E', L'> un nouveau graphe
                  tel que C' =  R'= E'= L'= ∅
                  Soit Cri les concepts rattachés à la relation ri dans h
                  R' = ri
                  L'(ri) = (Type(ri))
                  POUR TOUS ci ∈ Cri FAIRE
                  Début
                        C' = C' ∪ ci
                        L'(ci) = (Type(ci), Ref(ci))
                        SI <ri,ci> ∈ E2 ALORS
                              E' = E' ∪  <ri,ci>
                        SINON
                              E' = E' ∪  <ci,ri>
                  Fin
                  P = P ∪ g'
               Fin
      Fin
      Retourner P
Fin
```

Algorithme 2-1 Fonction de translation

Par exemple, l'Algorithme 2-1 (fonction de translation) appliqué au graphe de la Figure 2-1 consiste à isoler, pour chacune des composantes connexes, les sous-graphes ayant une seule relation. Dans l'exemple ci-dessous (Figure 2-5), le graphe conceptuel est subdivisé en deux briques ayant une seule relation et ses concepts rattachés.

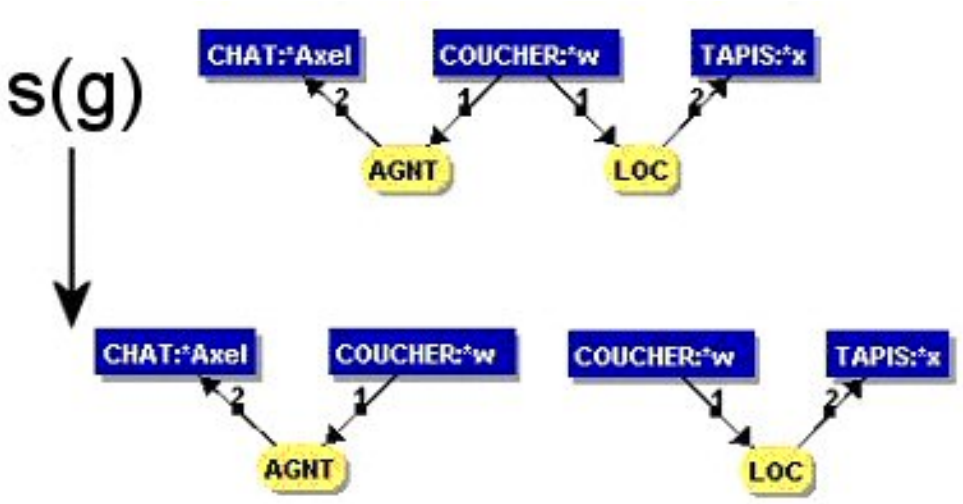

Figure 2-5 Application de la fonction de translation s(g) sur un graphe conceptuel.

Évidemment, après avoir exécuté cette opération, le graphe conceptuel ne respecte plus la forme normale. Cependant, si le graphe de départ *g* était en forme normale, l'exécution de la fonction de translation sur un graphe donne des sous-graphes du graphe de départ. Par conséquent, la jointure maximale basée sur les étiquettes identiques d'un ensemble de briques joindra nécessairement les concepts ayant été éclatés. Ainsi, la jointure maximale de cet ensemble de briques basé sur les étiquettes semblables redonne le graphe d'origine.

La fonction de translation présentée dans cette section sert principalement à caractériser un graphe conceptuel comme étant une collection de sous-graphes. Ainsi, la recherche d'informations nécessite seulement un sous-ensemble du graphe conceptuel pour valider l'existence d'une projection dans un graphe. De cette façon, on minimise les accès à la mémoire collective et on réduit le nombre d'éléments candidats à la projection. Dans la

prochaine section, nous verrons donc comment recomposer *g'* à partir de *P*, si *P = s(g')* où *g'* est la forme normale de *g*.

2.2.3 Recomposition d'un graphe conceptuel

À la section précédente, nous avons défini ce qu'est une brique ainsi que la fonction de translation d'un graphe conceptuel en un ensemble de briques. Le résultat de la fonction de translation appliqué sur un graphe conceptuel est un ensemble de graphes conceptuels connexes, où chacun de ces graphes conceptuels est un graphe conceptuel minimal qui possède exactement une seule relation. Dans cette section, nous définissons formellement la représentation d'un graphe conceptuel par rapport à un ensemble de briques. Nous verrons comment recomposer un graphe conceptuel originel à partir d'un tel ensemble de briques.

Nous définissons alors formellement un ensemble de briques constituant un graphe conceptuel de la façon suivante :

> **Définition 2-10** *(Un ensemble de briques)* Un ensemble de briques qui représentent un graphe conceptuel $g = <C_1, R_1, E_1, l_1>$ qui a subi un éclatement par l'application de la fonction de translation *s* est caractérisé par l'ensemble $G = \{g_1, g_2, \ldots, g_{|R1|}\}$ où chaque g_i représente une brique, tel que $G \subseteq \wp(G^l)$. Chaque brique $g_i \in G$ est une copie d'un sous-graphe de *u* ayant au plus une relation (et ses concepts rattachés). Ainsi, la

jointure maximale basée sur les étiquettes de tous les éléments de g_i forme un graphe isomorphe à *g*.

Les briques qui représentent un graphe conceptuel sont indépendantes dans leur représentation et uniques pour un graphe conceptuel en forme normale[4]. Par conséquent, l'application de la jointure maximale basée sur les étiquettes recompose le graphe conceptuel originel à partir d'un ensemble de briques :

Définition 2-11 (*Reconstruction*) Si un graphe conceptuel *g* en forme normale est décomposé en un ensemble de briques, il n'existe qu'une seule solution possible pour reconstruire *g* à partir de l'ensemble de briques; si deux étiquettes sont égales, les concepts (relations) associés sont joints.

En effet, si *g* est en forme normale au départ, chaque concept est représenté une seule fois dans le graphe pour un même référent. La fonction de translation d'un graphe conceptuel vers un ensemble de briques duplique nécessairement les concepts reliés à plus d'une relation dans le graphe originel. Puisque *g* est en forme normale, nécessairement chaque concept dupliqué qui possède une étiquette semblable, origine du même concept. Par conséquent, la jointure maximale de l'ensemble des briques a pour effet de fusionner les concepts de même identifiant, et ainsi, reproduire le graphe *g* du départ. Il est important, ici, de remarquer que la fusion des éléments se fond sur les éléments qui possèdent la même étiquette et non les concepts compatibles à la fusion. La reconstruction présentée dans ce

[4] Considérant qu'un graphe conceptuel du système est sous la forme normale, il n'existe donc pas de sous-graphe identique à un autre sous-graphe. Par conséquent, une brique est donc unique pour un graphe conceptuel donné.

chapitre est faite sur des briques qui proviennent du même graphe. Puisque les graphes conceptuels ne sont pas nécessairement sous la forme normale entre eux, nous détaillerons, dans le Chapitre 3, la technique utilisée pour reconstruire les graphes d'origine à partir d'un ensemble de briques qui proviennent de plusieurs graphes conceptuels différents.

La technique de représentation (et éventuellement de recherche) des éléments est pratiquement identique au paradigme des bases de données relationnelles. Dans les bases de données relationnelles, la représentation (recherche) d'information se fait à partir de clé primaire et étrangère. L'adaptation que nous proposons fait une recherche sur les briques en fusionnant les étiquettes (clés primaires et étrangères) des briques pour recomposer le graphe d'origine, ou du moins, un sous-graphe du graphe d'origine. Ainsi, on implante les graphes conceptuels dans la base de données relationnelle en suivant ce modèle. Au cours de la section suivante, nous allons voir cette technique.

2.2.4 Recherche structurelle à travers un ensemble de briques

Dans cette section, nous verrons comment rechercher un graphe à partir d'un ensemble de briques selon la structure du graphe de requête. Cette section a pour but de présenter la technique de recherche structurelle qui est utilisée dans le modèle de la mémoire collective. La méthode de recherche d'informations que nous proposons est séparée en deux parties. La première partie vérifie l'existence d'une structure similaire à la requête dans la base de connaissances. Nous vérifions s'il existe un ensemble de briques

qui puissent potentiellement se projeter sur la requête. Si tel est le cas, la seconde partie vérifie l'existence de référents qui instancient les nœuds concepts retrouvés dans la première partie. Ainsi, la première partie filtre les requêtes dans le but de minimiser les téléchargements en identifiant tôt les requêtes infructueuses ou trop volumineuses. De plus, ce procédé a pour but de réduire le champ de recherche à travers l'ensemble des graphes conceptuels.

Pour déterminer si une connaissance fait partie d'un ensemble de briques, nous définissons une opération de *couverture de graphe*. Cette opération de couverture permet donc de filtrer la requête de façon à éliminer les structures qui apporteront des résultats infructueux. Dans le cadre du formalisme des graphes conceptuels, elle a été déjà introduite dans [MC93b]. Cette opération recherche un graphe conceptuel à partir d'un ensemble de briques. Plus formellement, la définition suivante est introduite :

> **Définition 2-12** (*Couverture de graphe*) un ensemble de briques $G = \{g_1, g_2, ..., g_n\}$ constitue une couverture d'un graphe conceptuel g si, et seulement si, le graphe $<\cup_i C_i, \cup_i R_i, \cup_i E_i, L_i>_{1\leq i\leq n}$ est équivalent à g, où C_i, R_i et E_i dénotent respectivement les nœuds concepts, les nœuds relations et les arcs de g_i.

Considérons la Figure 2-6 suivante qui présente un graphe g et un ensemble des quatre briques g_1, g_2, g_3 et g_4. La couverture de l'ensemble de briques sur g est constituée des briques g_1, g_2 et g_3.

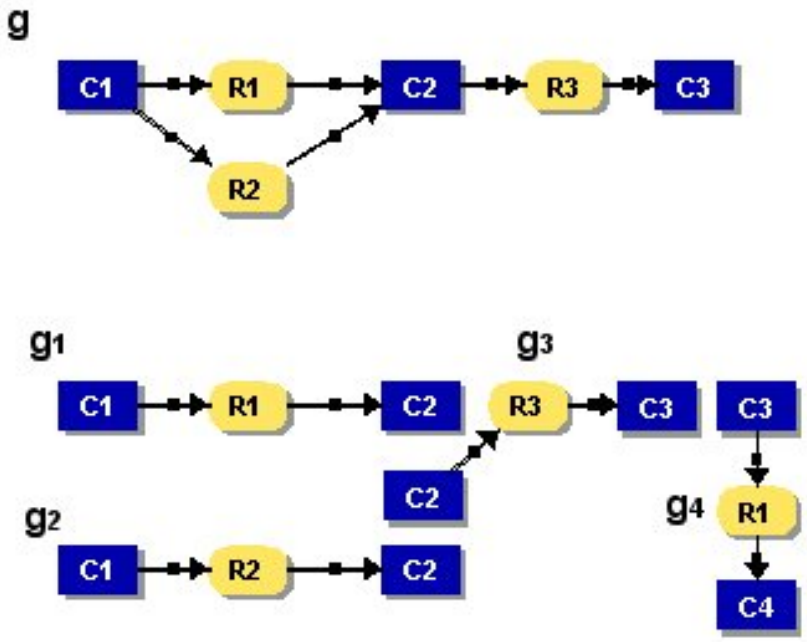

Figure 2-6 L'ensemble {g1, g2, g3 } constitue une couverture de g.

S'il existe plusieurs graphes dans l'ensemble de briques, il peut exister plusieurs couvertures possibles. En effet, selon les briques choisies pour couvrir les nœuds concepts et les nœuds relations du graphe de départ, il peut exister plusieurs façons de le couvrir. Au cours du Chapitre 3, nous verrons comment il est possible de trouver exactement le graphe d'où origine le résultat de la couverture.

Afin de trouver les couvertures possibles d'un graphe conceptuel, nous utilisons l'Algorithme 2-2 qui est une adaptation de l'algorithme présenté dans [MC1993b], [OUN1998] et [OUN2000]. L'algorithme donne pour un graphe conceptuel $g = <C, R, E, L>$ les couvertures possibles à partir d'un ensemble de briques P. La démarche consiste à projeter chaque sous-graphe de g sur les briques de P. La fonction s(g) éclate le graphe g en un ensemble de briques. Si la fonction de translation ne retourne

aucune brique, alors l'ensemble vide ∅ est retourné. Sinon, on projette les sous-graphes de *g* sur le sous-ensemble de briques de *P*. On sélectionne les briques à l'aide de la fonction SELECTIONNER(g_i, *P*). Cette fonction retourne toutes les briques de *P* dont la relation possède un type égal ou plus spécialisé à la relation de g_i. Cette fonction permet de sélectionner seulement les sous-graphes qui peuvent contribuer au succès d'une projection. On vise à minimiser les transferts entre le client et le serveur diminuant ainsi l'utilisation de la bande passante et le nombre de briques à traiter lors de la projection. Ensuite, on vérifie pour chacune des briques du graphe de requête celles qui peuvent se projeter sur les briques candidates. Soit la fonction PROJECTION (p_i, g_i) qui vérifie s'il existe une projection du sous-graphe g_i sur la brique p_i. Cette fonction retourne *faux* quand il n'est pas possible d'appliquer la projection de g_i sur p_i, les divers résultats possibles[5] de la projection étant tous des éléments de la couverture de g.

[5] Nous rappelons qu'il peut exister plusieurs résultats possibles à l'opérateur de projection.

```
Algorithme 2 (Couverture d'un Graphe)
Couverture(un graphe conceptuel g, un ensemble de brique P)
DÉBUT
      COUVERTURE = ∅
      ResEclatement = S(g)
      POUR TOUT gi ∈ ResEclatement FAIRE
      DÉBUT
            ResProj = ∅
            G = SELECTIONNER(gi,P)
            POUR TOUT pi ∈ G FAIRE
            DÉBUT
                  SI PROJECTION(pi , gi) = Vrai ALORS
                        ResProj = ResProj ∪ pi
            FIN
            SI ResProj ≠ ∅ ALORS
                  COUVERTURE = COUVERTURE ∪ ResProj
            SINON
                  Retourner ∅
      FIN
      Retourner COUVERTURE
Fin
```

Algorithme 2-2 Couverture d'un graphe

La complexité de cet algorithme dépend de la fonction PROJECTION(p_i; g). En effet, la complexité de cet algorithme est |S(g)| × |P| × O(PROJECTION), où |S(g)| désigne la cardinalité de l'ensemble de briques du graphe de requête *g*, |P| est le nombre de brique dans la base de graphes conceptuels et O(PROJECTION) désigne la complexité de la fonction PROJECTION.

Malgré de nombreux travaux ayant trait à l'étude de la complexité de la projection dans le formalisme des graphes conceptuels [MUG1993b], [MC1993a], [PC1995], [GUI1996], la classe de complexité de cette dernière n'est pas encore exactement connue. Tout au plus, certains algorithmes polynomiaux ont été proposés pour des graphes conceptuels de type particulier (par exemple le cas où les graphes correspondent à des arbres [MUG1993b]). Cependant, à l'heure actuelle, il n'existe pas d'algorithme polynomial

traitant de la projection d'une façon générale sur n'importe quel type de graphe conceptuel. La complexité de la recherche de la couverture d'un graphe n'est donc pas polynomiale. Toutefois, chaque fois que l'algorithme de projection est polynomial, la fonction COUVERTURE l'est également. Dans l'algorithme présenté dans cette section, la fonction de projection est seulement utilisée pour comparer deux briques. Dans ce cas, la fonction de projection est polynomiale puisqu'on compare les nœuds un à un.

Ainsi, lorsqu'une couverture *g'* est trouvée pour un graphe *g*, l'extension sémantique de cette couverture, c'est-à-dire $\delta(g')$, peut être trouvée. Nous verrons dans la prochaine section comment l'extension sémantique est calculée à l'aide d'un système de base de données relationnelle utilisé pour réaliser un tel calcul.

2.2.5 Extension sémantique d'une brique

Comme il a été dit précédemment, l'extension sémantique d'un graphe conceptuel représente les conditions sous lesquelles un graphe est vrai. Selon Mineau [MIN2000], la véracité d'un graphe conceptuel *g* est dépendante de l'existence d'instances qui réalisent la sémantique du graphe, d'où la définition de $\delta(g)$ présentée précédemment. La vérification de l'extension sémantique d'un graphe conceptuel consiste donc à vérifier s'il existe un arrangement possible de référents qui peuvent être associés à chaque concept d'un graphe. Dans son article, [MIN2000] propose de calculer l'extension sémantique des graphes

conceptuels à l'aide d'un système de gestion de base de données relationnelle, car cette technologie est éprouvée, efficace et connue des programmeurs.

Pour déterminer les référents correspondant à l'extension sémantique d'un graphe conceptuel *g en forme normale*, nous avons défini (voir section 2.2.1) une fonction δ(g) qui retourne tous les tuples de référents pour lesquels *g* s'évalue à vrai[6]. Si l'on transpose ce problème dans une optique de base de données relationnelle, l'opération consiste à faire une sélection, avec les contraintes du graphe conceptuel, sur la base de connaissances.

Dans le modèle de la mémoire collective, pour évaluer l'extension sémantique de plusieurs briques, il faut évaluer chaque brique et joindre ces briques sur leurs concepts communs. Les concepts communs de deux briques sont utilisés comme lien référentiel dans la requête. La définition suivante nous permet d'évaluer l'extension sémantique associée à un graphe.

> **Définition 2-13** (*Évaluation d'un graphe*) Soit $g = <C_1, R_1, E_1, l_1>$ un graphe conceptuel connexe, $G = \{g_1, g_2, ..., g_{|R1|}\}$ l'ensemble des briques qui lui correspond et l'opérateur $\delta(g_i)$ qui retourne les référents d'une brique. Le graphe *g* s'évalue à vrai seulement s'il existe au moins une jointure non vide *g'* des référents de $\{\delta(g_1), \delta(g_2), ..., \delta(g_{|R1|})\}$ différent de l'ensemble vide ∅ de sorte que $\delta(g') \neq \varnothing$. Dans le cas contraire, il s'évalue à faux. La méthode de jointure des référents sera expliquée dans le Chapitre 3.

[6] Les graphes sont en forme normale, ce qui permet de trier les nœuds du graphe avec un ordre lexicographique pour associer les bons référents aux bons concepts.

La projection π telle qu'elle a été définie par Sowa n'est pas nécessairement injective. Si x_1 et x_2 sont deux concepts ou deux relations tels que $x_1 \neq x_2$, il est possible d'avoir $\pi(x_1) = \pi(x_2)$. Le résultat de la projection n'est pas toujours unique. Le graphe *g* peut avoir plusieurs sous-graphes différents vérifiant la condition de la projection. C'est pour cette raison que l'opérateur de projection est si complexe du point de vu algorithmique. L'algorithme de la projection doit vérifier s'il existe un recouvrement de graphe lorsque le graphe est replié sur lui-même. Pour réduire la complexité de l'opérateur de projection, nous procédons en deux phases : la recherche de couverture (que nous avons vue à la section précédente) et la recherche de l'extension sémantique. La recherche d'une couverture vérifie s'il existe une structure qui est semblable à notre requête. La recherche de l'extension sémantique vérifie s'il existe des référents qui instancient la sémantique de notre requête. Ainsi, la couverture s'intéresse à la structure alors que l'extension sémantique s'intéresse au contenu du graphe conceptuel. Lors d'une opération de recherche, pour éviter des recherches infructueuses, nous vérifions au préalable l'existence d'une couverture semblable au graphe de requête. Par la suite, s'il existe une structure semblable au graphe de requête, l'extension sémantique est calculée pour trouver quels référents instancient les briques retrouvées.

L'Algorithme 2-3 présente l'implémentation de la technique de recherche de la mémoire collective. Tout d'abord, l'opération de couverture recherche, à travers un ensemble de briques, toutes les briques candidates pour former le graphe de requête. Les briques candidates retrouvées sont les briques qui peuvent se projeter dans le graphe de

requête. Cependant, l'union de ces briques ne donne pas nécessairement un graphe qui compose la sémantique du graphe de requête. Il est possible que la jointure des briques candidates donne un graphe conceptuel isomorphe au graphe de requête, mais que ces briques ne fussent pas jointes dans le graphe conceptuel originel. Ainsi, avant d'interroger la base de données, nous réalisons un algorithme d'épuration de briques qui permet de retrouver les briques qui peuvent être jointes et qui représentent la sémantique du graphe de requête.

La fonction `Composer_n_briques(g, P)` permet de vérifier s'il existe un ensemble de briques tel que la jointure maximale forme une couverture d'une composante connexe du graphe de requête. S'il existe un tel ensemble de briques pour chacune des composantes connexes du graphe de requête, alors cet ensemble est retourné. Sinon, un ensemble vide ∅ est retourné. Cette fonction valide que les briques retrouvées proviennent toutes du même graphe d'origine. Nous exprimons alors des conditions de recomposition des réponses partielles obtenues, afin de ne générer que des réponses de projection exactes. Les conditions de recomposition sont basées sur la structure de la requête. Cette fonction sera détaillée dans le Chapitre 3.

Enfin, la fonction `Interroger_SGBD(Resultat)` recherche, dans la base de données relationnelle, les référents qui correspondent aux concepts des briques passées en paramètre. Cette fonction interroge la base de données relationnelle avec une requête où les paramètres sont les briques. Le calcul de l'intersection entre les tuples résultant de l'extension sémantique des briques fait la jointure des référents ayant la même étiquette.

Enfin, les référents correspondant à l'extension sémantique du graphe de requête sont retournés. Ainsi, les référents donnent la réponse exacte à la projection du graphe de requête.

```
Algorithme 3 (Rechercher les référents d'un graphe dans un
              ensemble de briques)
Rechercher(un graphe conceptuel g, un ensemble de brique P)
DÉBUT
      Resultat = Couverture(g, P)
      SI Resultat ≠ ∅ ALORS
      DÉBUT
            Resultat = COMPOSER_N_BRIQUES(g, P)
            SI Resultat ≠ ∅ ALORS
            DÉBUT
                  EnsRef = Interroger_SGBD(Resultat)
                  Retourner EnsRef
            FIN
      FIN
      Retourner ∅
Fin
```

Algorithme 2-3 Rechercher les référents d'un graphe dans un ensemble de briques

Pour mieux illustrer l'algorithme de recherche, utilisons le graphe de requête présenté par la Figure 2-7. La couverture du graphe de requête sur l'ensemble de briques de la Figure 2-8 est représentée par les briques A_1 et A_4. Seules ces briques peuvent être projetées dans le graphe de requête.

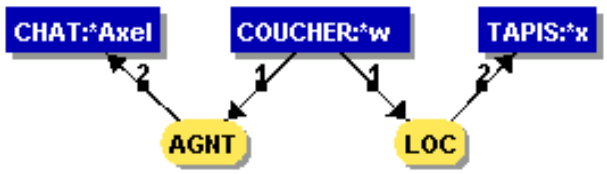

Figure 2-7 Le chat, Axel, couché sur un tapis.

Ensuite, le raffinement du résultat de la couverture ne modifie pas l'ensemble de briques trouvé. Ici deux cas peuvent survenir, Les briques A1, A2 et A_1, A_4 peuvent être jointes respectivement sur le concept [COUCHER :*w2] et [ACTION :*x1] et [COUCHER :*w2] et [ACTION :*z1]. La jointure de ces deux concepts donne un graphe conceptuel ayant le même nombre de relations que le graphe de requête.

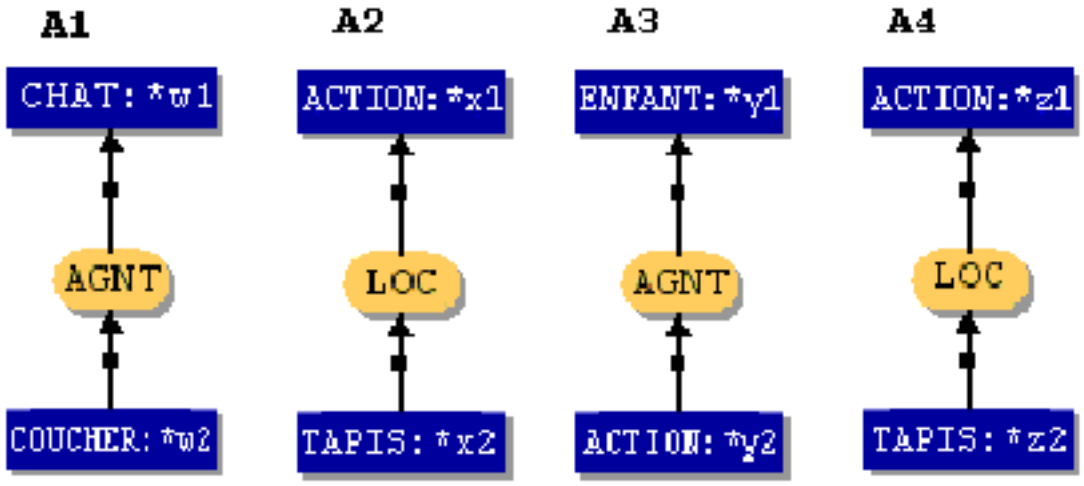

Figure 2-8 Ensemble de briques

Enfin, il ne reste plus qu'à interroger le système de gestion de base de données relationnelle pour déterminer les référents qui instancient ce graphe. Dans le premier cas, on essais de joindre sur le concept commun les référents des briques retournés par les *opérateurs* $\delta(A_1)$ et $\delta(A_2)$. Cependant, tel que présenté à la Figure 2-9, il n'existe pas d'arrangement possible qui puisse être joint entre les deux briques A_1 et A_2, c'est-à-dire

qu'il n'existe pas de variables correspondantes au concept [COUCHER :*w2] dans A_1 qui soit égale à au moins une variable correspondante au concept [ACTION :*x1] dans A_2.

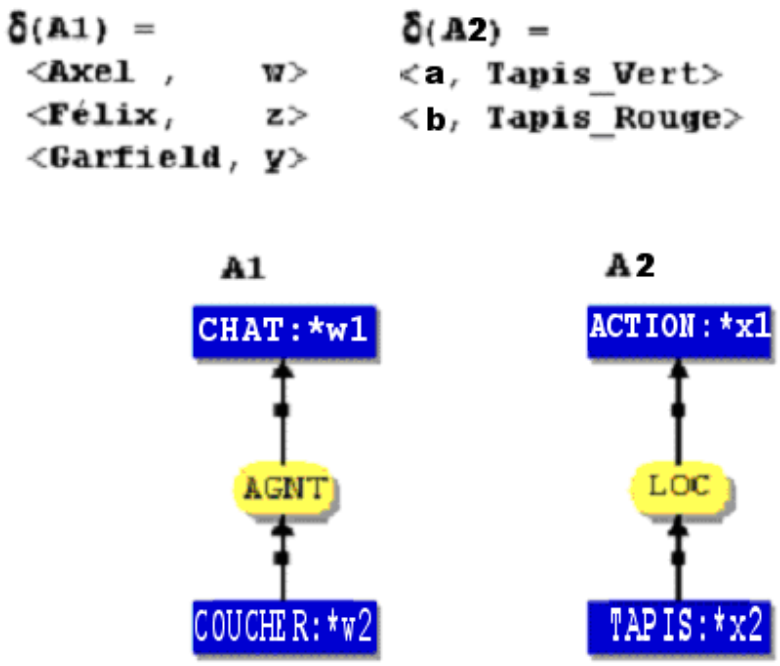

Figure 2-9 Mise en correspondance des briques.

Dans le second cas, on essaie de joindre les briques A_1 et A_4. La Figure 2-10 montre le résultat de l'exécution de l'opérateur δ sur les briques A_1 et A_4. *L'opérateur* δ(A_1) δ(A_4) retourne les référents qui instancient les concepts de ces briques. On remarque que les points de jonction sur les briques sont les variables qui possèdent le même identifiant. Dans le cas présent, tel que présenté dans la Figure 2-10, l'intersection des tuples de référents fusionne le référent *w* des deux briques. Intuitivement, il est possible de faire une telle jointure puisque deux référents ayant un même identifiant sont nécessairement du même type puisqu'à l'origine les graphes étaient en forme normale.

Après avoir fait la jointure de $\delta(A_1)$ et $\delta(A_4)$, on conclut que seul le tuple *<Axel, w, TAPIS_VERT>* est valable pour cet ensemble de briques. Le résultat de la recherche nous dit qu'il existe une situation conforme à notre graphe conceptuel de requête. De plus, nous savons qu'il existe un graphe dont la sémantique dit : « Un chat, Axel, couché sur le tapis vert. ». Par conséquent, la définition que nous avons donnée de l'évaluation de la véracité nous indique que la valeur de vérité qui correspond à cet ensemble de briques est vraie.

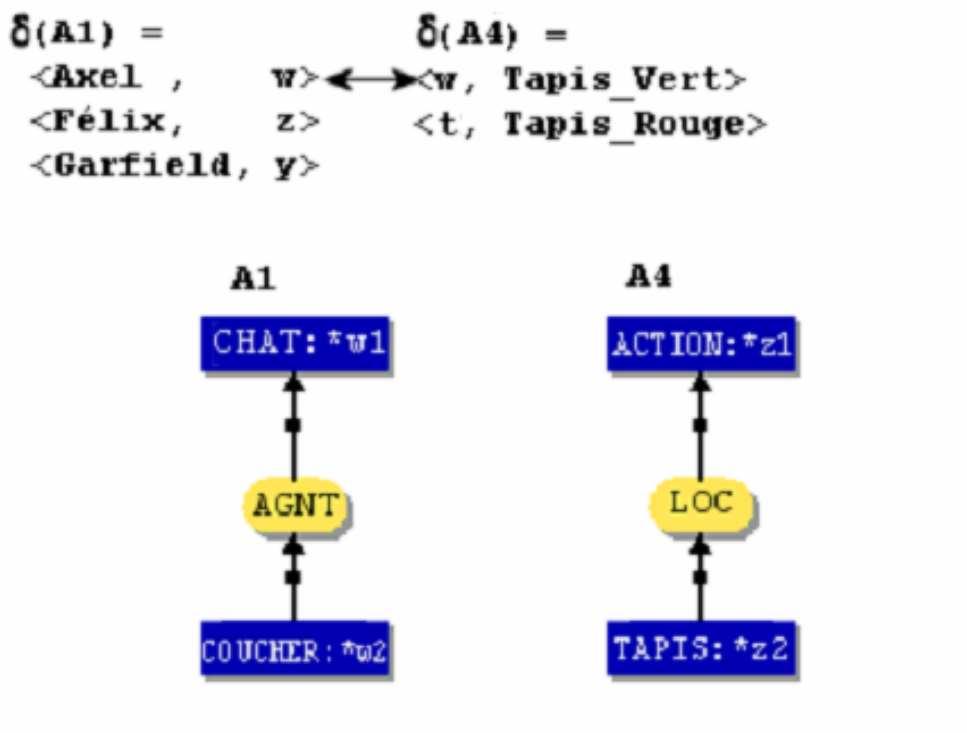

Figure 2-10 Mise en correspondance des briques.

2.3 Conclusion

Au cours de ce chapitre, nous avons présenté le langage de représentation des connaissances utilisé dans la mémoire collective : les graphes conceptuels. Ces graphes permettent de représenter les connaissances nécessaires à la réalisation des tâches des systèmes intelligents dans un environnement distribué. Les graphes conceptuels sont décomposés en des structures de graphes minimaux que nous appelons *briques*. Nous avons vu comment recomposer un graphe à partir d'un ensemble de briques, et comment la valeur de vérité de ce graphe recomposé pouvait être évaluée. Cette décomposition en brique est faite dans le but de faciliter le transfert, la manipulation et l'indexation des connaissances. Ces briques sont diffusées sur l'ensemble du réseau de façon à ce que chaque client puisse accéder comme bon lui semble aux connaissances sans redondance de la connaissance.

Les éléments définis dans ce chapitre font un survol des aspects de la mémoire collective. Au cours du Chapitre 3, nous allons voir l'implémentation de la technique de recherche sur les briques. Nous allons comparer notre méthodologie de recherche avec un logiciel existant de recherche et d'interrogation de système à base de graphes conceptuels.

Chapitre 3 La recherche d'informations

3.1 Introduction

Il existe aujourd'hui de nombreux systèmes fondés sur les graphes conceptuels intégrant un module de recherche. Citons par exemple, le système KALIPSOS d'IBM [BDFL1988], [FAR1989], [NOG1990] destiné à la compréhension des textes écrits en français ou le système MENELAS [BZ1992], [ZA1993], [MEN1994], [ZWEIL1994], [ZWEIL1994] dédié à la recherche médicale et le système de recherche de texte de Myaeng [MYA1992a], [MYA1992b], [ML1993], [MKL1994]. Cependant, la plupart de ces systèmes ne se situent pas dans le cadre particulier de la recherche d'informations, mais sont plutôt destinés à la représentation et à la gestion des connaissances. Contrairement aux exemples cités précédemment, le modèle de la mémoire collective nécessite un mode de représentation qui soit performant lors de la recherche.

Nous avons vu au Chapitre 2 que nous proposons de décomposer les graphes conceptuels en un ensemble de briques. Puis, une opération de couverture permet de retrouver les briques candidates à une projection. Ensuite, on recompose les briques candidates et le résultat de la projection est trouvé. Cependant, jusqu'à présent, plusieurs questions restent sans réponses : Comment recomposer les briques ? Est-ce plus rapide en terme de temps d'exécution que la projection traditionnelle ? Le Chapitre 3 répondra à ces questions. La section 3.2, présente la technique qui est utilisée pour s'assurer que l'assemblage d'un

ensemble de briques résultant de la couverture forme un graphe tel que toutes les briques utilisées pour l'assemblage proviennent du même graphe.

Ensuite, dans la section 3.3 nous terminerons de détailler l'algorithme de projection que nous avons réalisé. Nous présenterons l'algorithme de recherche dans son ensemble. De plus, la complexité de notre algorithme sera analysée. Dans la section 3.4, nous montrerons les résultats de la recherche sur un ensemble de graphes conceptuels. Dans cette section, nous exposerons un comparatif avec notre opérateur de recherche versus un opérateur de recherche de graphe conceptuel de l'API (Application Programming Interface) Notio développée par F. Southey [SOUT1999]. Enfin, la dernière section (section 3.5) fera une brève conclusion sur l'opérateur de recherche présenté dans ce chapitre.

Conséquemment, ce chapitre vise à démontrer l'utilité de la technique d'indexation de graphes conceptuels présentée au Chapitre 2 et à en évaluer son efficacité.

3.2 Reconstitution d'un ensemble de briques

Afin de trouver l'unique reconstruction correspondant à un graphe de requête, nous adoptons l'Algorithme 3-1. Cet algorithme, présenté brièvement dans le chapitre 2, donne pour un graphe conceptuel connexe *g*, l'ensemble de briques *B* qui forme sa couverture. La démarche consiste à former des graphes connexes à partir de l'ensemble de brique *B*. Soit la fonction `TROUVER_BRIQUES(C1,B)` permet de trouver toutes les briques de *B* qui possèdent un

nœud dont l'étiquette est égale à l'étiquette de c_1. À l'aide de cette fonction, les briques sont jointes sur les nœuds qui possèdent la même étiquette. Si deux nœuds ont la même étiquette, la fonction `JOINTURE(`c_1`,` d_k`,` b_i`)` fait une jointure externe sur les deux sous-graphes d_k et b_i sur le concept c_1. Ainsi, on joint les briques qui étaient jointes à l'origine dans la mémoire collective.

Ensuite, pour chacune des composantes connexes retrouvées à partir de l'ensemble de brique *B*, on vérifie s'il existe une couverture pour chacune des composantes connexes du graphe de requête. Pour réaliser cette tâche, on fait appel à la fonction `COUVERTURE(g, Ens_Brique)`. Cette fonction prend en paramètre un graphe *g* et un ensemble de briques *Ens_Brique*. Cette fonction retourne la couverture des briques candidates à la projection, s'il en existe. Sinon un ensemble vide est retourné. Dans l'algorithme présenté précédemment, s'il existe une couverture pour chacune des composantes connexes du graphe de requête, alors cet ensemble de briques est retourné, sinon c'est un ensemble vide qui est retourné.

```
Algorithme 4 (Recomposer un ensemble de briques)
Composer_n_briques(un graphe conceptuel connexe g,
                   un ensemble de briques B)
DÉBUT
      //Jointure maximale de toutes les briques par rapport à leurs
      //étiquette.
      POUR TOUTES les briques bi ∈ B FAIRE
      DÉBUT
            POUR TOUS les nœuds concepts cj ∈ bi FAIRE
            DÉBUT
                  C = TROUVER_BRIQUES(cj,B)
                  POUR TOUTES les sous-graphes dk ∈ C FAIRE
                  DÉBUT
                        B = JOINTURE(cj,dk, bi)
                  FIN
            FIN
      FIN

      //Vérifier quelles composantes connexes couvre le graphe de requête
      //g.
      POUR TOUTE composante connexe ci ∈ g FAIRE
      DÉBUT
            POUR TOUTES les composantes connexes bj ∈ B FAIRE
            DÉBUT
                  Ens_Brique = ÉCLATEMENT(bj)
                  Ens_Couv = COUVERTURE(ci, Ens_Brique)
                  SI Ens_Couv ≠ ∅ ALORS
                        Ens_Retour(i) = Ens_Couv
            FIN
            Si Ens_Retour(i) = ∅ ALORS
                  Retourner ∅
      FIN
      Retourner Ens_Retour
Fin
```

Algorithme 3-1 Recomposer un ensemble de briques

Grâce à cet algorithme, la recherche d'une projection à travers l'ensemble de briques retourne une réponse unique et conforme aux connaissances encodées dans la mémoire collective. Dans la section suivante, nous donnons une vue d'ensemble de l'algorithme de recherche de la mémoire collective. Ses fonctionnalités ainsi que ses caractéristiques calculatoires seront sommairement présentées.

3.3 Recherche de connaissances

3.3.1 Vue d'ensemble sur l'algorithme de recherche

Au départ, nous avions fixé des objectifs quant à l'efficacité de l'algorithme de recherche. L'algorithme de recherche devait permettre une recherche précise, rapide et qui n'engorge pas le serveur. Sachant que l'opérateur de recherche traditionnelle des graphes conceptuels possède une complexité non-polynomiale (NP-Complet), il était nécessaire de vouloir réduire cette complexité, notamment par l'utilisation d'index. Ainsi, nous avons proposé, au Chapitre 2, une fonction de translation qui permet de recodifier un graphe conceptuel en élément plus petit et plus facilement indexable. Notre but était de proposer une fonction de translation qui vérifie deux conditions :

- La première est que la procédure de construction doit être réversible, c'est-à-dire que nous devons être capables de reproduire les graphes de départ à partir de l'ensemble de connaissances recodifiées. Cet ensemble de connaissance est composé des parties élémentaires des graphes, à savoir les *briques*. Pour reconstruire l'ensemble de départ, nous utilisons la technique des étiquettes uniques (forme normale) telle que présentée précédemment. Cette technique permet de reconstruire exactement le même graphe à partir de graphes(briques) provenant de son éclatement.

- La deuxième condition est que cet ensemble de connaissances doit permettre une opération de projection π équivalente à celle définie dans [SOW1984]. La sémantique de recherche des graphes conceptuels doit être conservée tout en améliorant les performances de recherche.

La décomposition étant faite, l'algorithme de recherche repose sur la structure de la requête pour établir les décisions de pertinence. Rappelons les éléments importants de cet algorithme :

- Tout d'abord, nous cherchons la couverture du graphe de requête par rapport à la base de connaissances (l'ensemble de briques pertinentes). Cette recherche d'une couverture se fait selon l'Algorithme 2-2. Cette fonction recherche pour chaque sous-graphe les sous-graphes qui s'y apparentent dans l'ensemble de briques. La complexité de cette fonction est donc l'ordre de $O(n^2)$ tel que n représente le nombre de briques dans la base de connaissances.
- Ensuite, nous réalisons une jointure maximale pour chacune des briques trouvées à l'étape précédente. Cette jointure est basée sur les étiquettes. Cette fonction fait la fusion des étiquettes identiques pour chacune des briques. La complexité de cette recherche est encore une fois de l'ordre de $O(n^2)$ tel que n représente le nombre de briques dans la base de connaissances.
- On valide ensuite le résultat de la jointure maximale en vérifiant s'il existe une couverture pour chacune des composantes connexes du graphe de requête et les briques jointes à l'étape précédente. Cette recherche d'une couverture se fait

selon l'Algorithme 3-1. La complexité de cette fonction est donc l'ordre de $O(n^4)$ tel que n représente le nombre de briques dans la base de connaissances.

- Enfin, le SGBR est interrogé pour connaître les référents qui instancient les concepts des briques retrouvées.

Les opérations exécutées par notre algorithme de recherche sont données à la Figure 3-1.

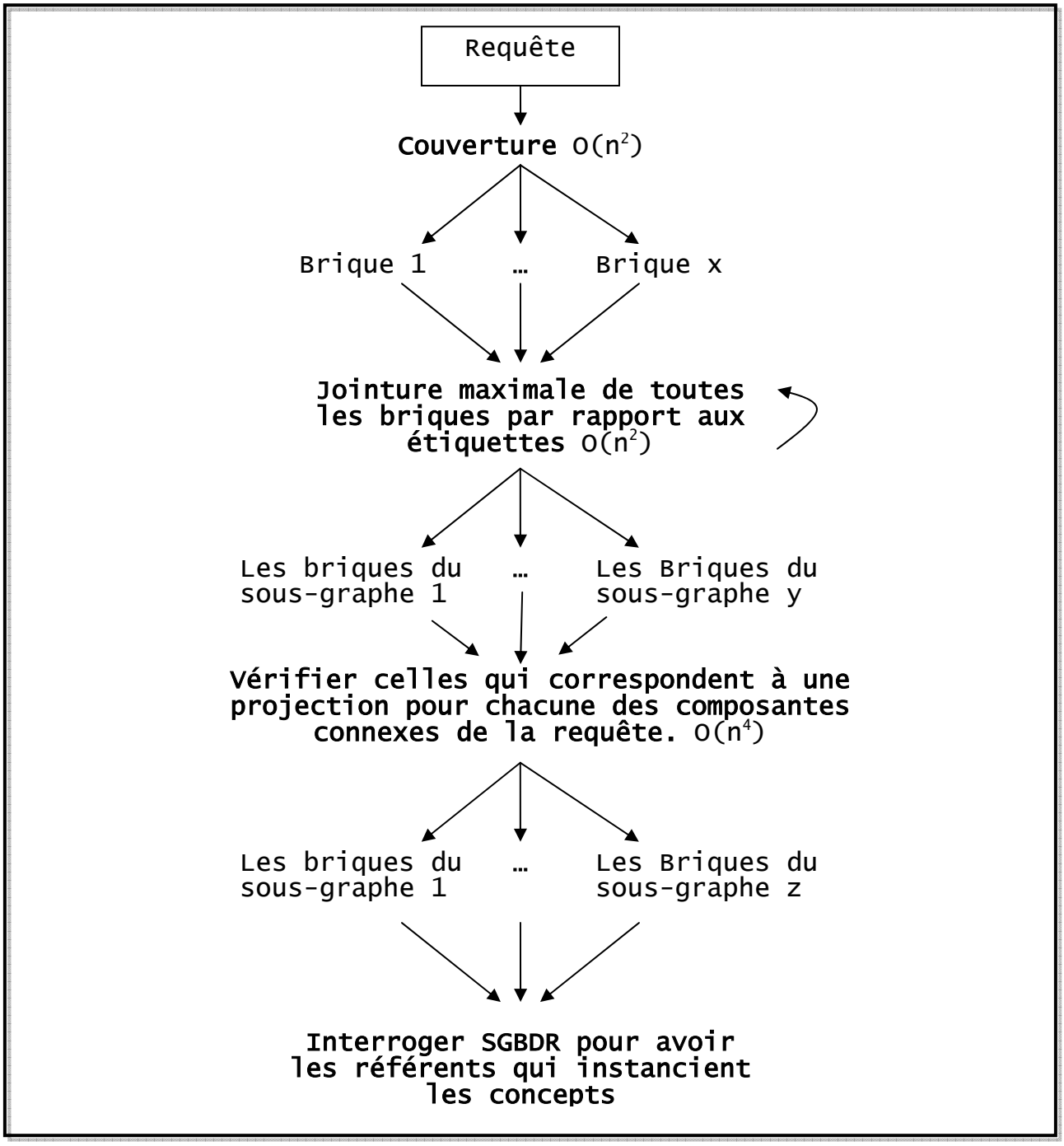

Figure 3-1 Description sommaire du fonctionnement de l'algorithme de recherche.

À la lumière de la Figure 3-1, nous pouvons déjà constater que l'algorithme de projection n'est autre qu'une combinaison d'unions et d'intersections ensemblistes. Ce qui

lui confère sa complexité polynomiale. Cette complexité est donnée tel que n représente le nombre de briques dans la base de connaissances et est donnée par:

$$O(n^2) + O(n^2) + O(n^4) + O(\text{Interrogation SGBDR})$$

La complexité des deux premières fonctions est négligeable comparée à celle de la fonction de vérification d'une couverture à partir des briques connexes $O(n^4)$. Nous pouvons déjà constater la complexité polynomiale de l'algorithme de projection proposé.

Comme il a été mentionné auparavant, la complexité de notre algorithme est fonction de la complexité des opérations d'intersection et d'union. Si aucune optimisation n'est appliquée à ces opérateurs (il peut s'agir de trier les ensembles manipulés), la complexité de notre algorithme est de l'ordre de $O(n^4)$. Par contre, nous savons que plusieurs techniques simples d'optimisation de l'union et de l'intersection d'ensemble permettent d'en réduire la complexité.

3.3.2 Discussion sur l'algorithme de projection

Plusieurs approches dans la littérature proposent un algorithme de projection basé sur une décomposition de la requête en sous-requêtes, puis un mécanisme de recomposition des réponses selon les résultats obtenus. Citons comme exemple l'algorithme de M. Mechkour [MEC1995], de B. Carbonneill [CAR1996] ou encore de Guinaldo et coll.

[GUI1996]. Dans ces algorithmes, il est toujours question d'appliquer une projection de chaque sous-requête sur toute la base de connaissances.

L'objectif étant de retrouver les graphes potentiellement pertinents. Cette opération est couteuse en temps de calcul. Dans notre algorithme, pour chaque sous-requête, nous filtrons directement les sous-requêtes qui lui sont pertinentes. La recherche à travers la base de connaissances est ainsi réduite par rapport aux connaissances transmises.

Afin d'illustrer la performance de notre approche, nous présentons, dans la prochaine section, un comparatif d'exécution par rapport à la projection dans l'API de Notio [SOUT1999]. Cet API a été créée et développée en java pour représenter et manipuler les graphes conceptuels. L'opération de projection est implémentée, mais permet seulement une projection injective, c'est-à-dire qu'on ne peut pas projeter de fusionner deux concepts différents sur un même concept.

3.4 Expérimentation de l'algorithme de recherche versus un algorithme de projection équivalent

Dans cette section, nous proposons une comparaison entre la méthode de recherche présentée dans ce mémoire et l'opérateur de projection de l'API Notio. Ces tests ont été réalisés sur un AMD Athlon™ XP 2000 à 1,67 GHz avec 256 Mo de RAM utilisant le système d'exploitation Windows Xp. Le support de la base de connaissances est constitué d'un treillis de 60 types de concepts et d'un treillis de 23 types de relations générés

aléatoirement de telle sorte que chaque nœud (concepts et relations) possède moins un parent et un enfant. Ces treillis sont conservés tout au long de l'expérimentation.

L'expérimentation que nous avons faite vérifie la recherche structurelle entre le graphe de requête et la base de connaissances. L'opérateur de projection de Notio vérifie s'il existe une projection injective du graphe de recherche dans la base de connaissances. C'est-à-dire que l'API Notio vérifie si l'on retrouve le graphe de requête comme sous-graphe d'un graphe de la base de connaissances ou pas. Ainsi, pour comparer les deux méthodes, nous observons la moyenne de temps d'exécution pour plusieurs graphes de requêtes et de base de connaissances engendrés aléatoirement. Pour engendrer aléatoirement des graphes, nous créons un certain nombre de concepts et de relations de type aléatoire que nous relions aléatoirement ensemble de telle sorte que les relations soient toujours conformes à leurs signatures.

L'expérimentation que nous avons faite traite les deux aspects suivants :

- une base de données de taille variable alors que la taille de la requête est fixe,
- une base de données de taille fixe alors que la taille de la requête est variable.

La façon dont nous avons procédé pour comparer les deux techniques pour une base de données de taille fixe et une taille de requête variable est la suivante :

1. Nous avons généré aléatoirement huit bases de connaissances contenant chacune un nombre égal de graphes qui ne sont pas nécessairement connexes. Dans ces

bases de connaissances, les nœuds, les arcs, les référents sont tous générés aléatoirement. Dans cette expérimentation, le nombre de briques dans ces huit bases de connaissances augmente à chaque itération.

2. Ensuite, nous avons dupliqué ces bases de connaissances et nous avons exécuté la fonction de translation, présentée au Chapitre 2, sur huit d'entre elles. Ainsi, la projection de l'API de Notio sera exécutée sur les bases de connaissances sans altération alors que la méthode proposée dans ce mémoire sera exécutée sur les bases de connaissances traduites sous la forme d'un ensemble de briques.
3. Pour chacune des bases de connaissances, nous avons comparé cent graphes de requêtes générées aléatoirement ayant chacun cent concepts et cent-cinquante relations reliés ensemble aléatoirement.
4. Pour chaque projection, nous avons vérifié le temps d'exécution des deux méthodes. Ensuite, nous avons calculé la moyenne des temps d'exécution pour une requête.
5. Enfin, nous avons réalisé les quatre étapes précédentes en augmentant la taille des bases de connaissances de deux cents briques.

Les résultats obtenus par cette expérimentation sont présentés dans la Figure 3-2.

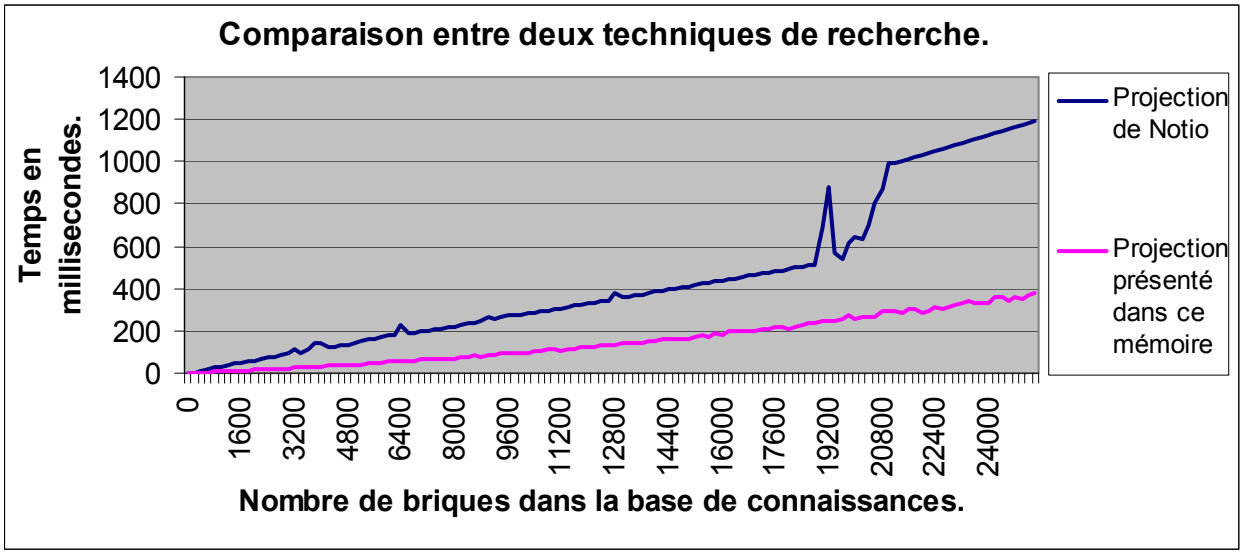

Figure 3-2 Comparaison entre les deux techniques de recherche telles que la taille de la base de données est variable alors que la taille de la requête est fixe.

Ensuite, nous avons comparé les deux techniques pour une base de données fixe et une taille de requête variable de la façon suivante :

1. Nous avons généré aléatoirement huit bases de connaissances de trois mille concepts et six mille relations. Les arcs qui relient ces concepts sont aussi générés aléatoirement. Dans cette expérimentation, les bases de connaissances sont fixes alors que la taille des requêtes est variable.
2. Nous avons dupliqué ces huit bases de connaissances et nous avons exécuté la fonction de translation, présentée au Chapitre 2, sur huit d'entre elles. Ainsi, la projection de l'API de Notio sera exécutée sur les bases de connaissances sans altération alors que la méthode proposée dans ce mémoire sera exécutée sur les bases de connaissances traduites sous la forme d'un ensemble de briques.

3. Pour chacune des bases de connaissances, nous avons comparé cent graphes de requêtes ayant chacun des nombres variables de briques générés aléatoirement.
4. Pour chaque projection, nous avons vérifié le temps d'exécutions des deux méthodes. Ensuite, nous calculons la moyenne des temps d'exécution pour une requête.
5. Enfin, nous avons réalisé les quatre premières étapes en augmentant la taille des requêtes de cent briques.

Les résultats obtenus par cette expérimentation sont présentés dans la Figure 3-3.

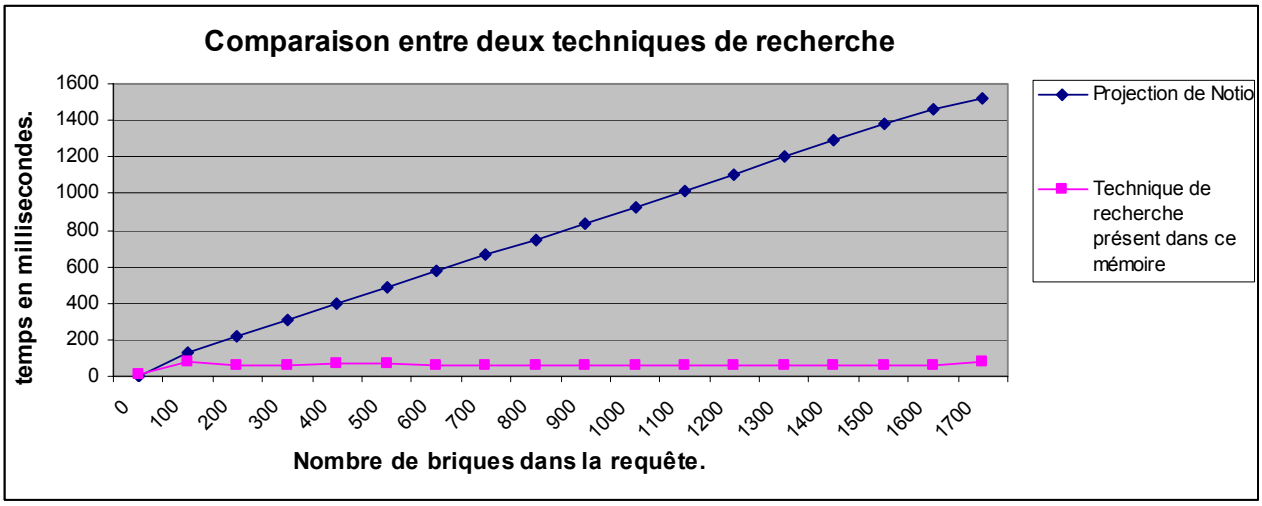

Figure 3-3 Comparaison entre les deux techniques de recherche tels que la taille de la base de données est fixe alors que la taille de la requête est variable.

Les résultats de ces expérimentations montrent qu'en moyenne, la taille de la base de connaissances ou de la requête influence moins notre méthode que la projection de Notio. Ainsi, lorsque nous avons affaire à une grande quantité d'information, notre méthode s'avère avantageuse comparativement à la recherche effectuée par l'API Notio.

Cependant, avec de petites bases de connaissances, la recherche à l'aide de l'opérateur de projection de l'API de Notio était plus performante que celle présentée dans ce mémoire. En fait, nous n'avons aucune idée des heuristiques utilisées dans Notio pour accélérer le traitement relié à la projection. Toutefois, nous avons divisé notre algorithme de projection en deux phases, dont la première identifie les sous-graphes potentiellement pertinents grâce à un index. Une partie des gains en rapidité sont probablement reliés à cet aspect.

3.5 Conclusion

Dans le chapitre précédent, nous avons présenté une méthode pour structurer les graphes conceptuels et les transmettre sur un réseau de communication. Nous avons aussi présenté une technique de recherche de connaissances recodifiées comme un ensemble de *briques*. Cependant, plusieurs questions ont été soulevées : Comment vérifier que les briques retrouvées font partie du même graphe à l'origine? Comment implémenter l'algorithme de recherche? Ce chapitre avait donc comme objectif de répondre à ces questions formulées au chapitre précédent. Grâce à la forme normale, nous avons vu qu'il est possible de joindre les concepts et les relations des briques qui étaient jointes dans un même graphe d'origine. Au cours de ce chapitre, nous avons aussi présenté un algorithme de recomposition de graphe. Cet algorithme permet de déterminer s'il existe une couverture de graphe pour laquelle les briques retrouvées forment une structure isomorphe au graphe de requête. Cet algorithme complète l'algorithme de recherche présenté au Chapitre 2. De

plus, nous avons présenté une analyse de la complexité de l'algorithme de recherche. Cet algorithme possède une complexité de l'ordre de $O(n^4)$ dans le pire cas.

Au départ, nous avions fixé comme objectif de trouver un mode de représentation des connaissances qui soit suffisamment expressif et qui permette une diffusion adéquate des connaissances. Le mode de représentation des connaissances utilisé favorise la transmission des connaissances puisqu'il permet une bonne précision des réponses et évite du même coup les requêtes inutiles du client. La totalité des connaissances ne doit pas être transmise pour résulter en une information qui intéresse le client puisque le client et le serveur engorgeraient le réseau. De plus, le mode de représentation des connaissances doit permettre une recherche efficace à travers l'ensemble des connaissances. Nous croyons avoir atteint ces objectifs. Dans le prochain chapitre, nous verrons quel est le mécanisme utilisé pour la mise à jour des valeurs de vérité associées aux connaissances dans un système de connaissances temps réel.

Chapitre 4 Mise à jour des connaissances

4.1 Introduction

Par définition, une *mémoire collective* est une mémoire qui est commune à l'ensemble de ses souscripteurs, appelés *clients*. Les clients peuvent accéder et réfléchir simultanément à partir des connaissances emmagasinées dans cette mémoire. Évidemment, pour réfléchir sur les informations reçues, les clients doivent emmagasiner les connaissances dans une mémoire temporaire. La mémoire du client permet un accès plus rapide aux données le plus souvent utilisées, un peu comme l'antémémoire d'un ordinateur. Cependant, la mémoire collective, qui évolue dans un environnement temps réel, doit prendre en considération que les connaissances qu'elle possède peuvent ne pas être stables. Des altérations des valeurs de vérité attribuables aux connaissances peuvent survenir en tout temps. Il y a donc un risque que les informations reçues par un client ne soient plus valides après un certain laps de temps. Considérant que le nombre de clients peut être très élevé, il est impensable que chaque client accède à sa guise au serveur. À cause de cette contrainte, les clients ne peuvent pas continuellement vérifier la validité des connaissances qu'ils possèdent. Nous devons donc minimiser le plus possible le nombre d'accès à la mémoire collective. Évidemment, cette problématique est connue et par conséquent une revue de littérature est présentée au Chapitre 6.

Ce qu'on souhaite faire ici, c'est d'associer à chaque brique b et chaque temps t_0 une fonction de péremption f_{b,t_0} où pour tout $t > t_0$, on a que $f_{b,t_0}(t)$ est la probabilité que la valeur de vérité de la brique b ait changée entre le temps t_0 et le temps t. En plus, on souhaite que cette fonction f_{b,t_0} soit complètement définie par des paramètres simples. Car on veut que la mémoire collective n'ait qu'à associer à chacune de ses briques les valeurs de ces paramètres pour que chaque client soit capable de calculer la fonction f_{b,t_0} pour les briques B qui les intéressent. Comme ce problème est presque impossible à résoudre tel quel, tu as donc fait des hypothèses simplificatrices. Les voici:

1. Le temps est discret, séparé en top d'horloge (laps de temps) fondamentale, que nous notons Δt;
2. Il est possible de quantifier un *niveau de confiance* quant à la validité d'une connaissance. Il est aussi possible de calculer pour chaque top d'horloge une distribution de probabilité quant à la variation de ce niveau de confiance envers la validité d'une connaissance;
3. La distribution de probabilité est la même pour chaque top d'horloge et elle est de la forme « le niveau de confiance d'une brique b subit une variation de $\varphi(b)$ dans la direction de la valeur 0 » avec probabilité $\eta(b)$ et « le niveau de confiance d'une brique b subit une variation de $\varphi(b)$ dans la direction opposée de la valeur 0 » avec probabilité $1 - \eta(b)$.

Ainsi, grâce à ces hypothèses simplificatrices, nous pouvons proposer que la mémoire collective distribue ses connaissances avec une indication de la volatilité de celles-ci. Les connaissances recueillies par le client peuvent être conservées jusqu'à ce qu'elles nécessitent une revalidation. Nous calculerons donc une date de péremption pour chacune des connaissances acquises par un client. L'objectif de ce chapitre est donc de présenter l'interprétation que nous faisons d'une mise à jour ainsi que l'approche que nous voulons utiliser pour gérer les mises à jour.

La section 4.2 présente les informations que doit fournir le serveur en vue de déterminer une date de péremption pour les connaissances. Nous y expliquerons l'interprétation accordée à chacune des informations transmises par le serveur. La section 4.3 présente des exemples d'utilisation des paramètres transmis par le serveur. La section 4.4 présente les fonctions d'utilité, nécessaires aux clients pour le calcul d'une date de péremption. Enfin, la dernière section (section 4.5) fait une brève conclusion de la technique de mise à jour des connaissances proposées.

4.2 Informations transmises par le serveur

Pour calculer une date de péremption, nous proposons une technique qui nécessite trois paramètres : $\lambda(b, t_0)$, $\eta(b)$, $\varphi(b)$, et qui représente respectivement le *niveau de confiance*, la *probabilité de changement* et une *valeur de fluctuation* d'une brique *b*. Tout d'abord, lors de la réception d'une brique, une valeur $\lambda(b, t_0)$, appelée *niveau de confiance*,

donne sa valeur de vérité $\in$ [-1,1]. Cette même valeur peut être interprétée comme un niveau de croyance envers la valeur de vérité de la brique au moment présent. Cette valeur sera présentée dans la section 4.2.1. Pour prévoir les mises à jour futures, nous utilisons un autre paramètre, en accord avec nos trois hypothèses, nous donne la probabilité que le *niveau de confiance* diminue ou augmente d'une certaine constante après un certain laps de temps constant. Cette autre valeur, appelée *probabilité de changement* $\in$ [0,1], permet de calculer la résistance du niveau de confiance par rapport au temps. Cette valeur sera présentée dans la section 4.2.2. Enfin, la valeur de fluctuation, présentée dans la section 4.2.3, représente l'ajout ou le retrait d'une portion (maximale) du niveau de confiance à chaque top d'horloge.

À l'aide de ces paramètres et nos hypothèses simplificatrices, il sera possible pour les clients de calculer une date approximative de péremption ces connaissances. En effet, la date de péremption donne le moment approximatif où une connaissance aura subi une fluctuation importante de sa valeur de vérité. De plus, le client pourra valider seulement les parties de ses connaissances qui sont périmées ou qu'il juge plus importantes ou critiques. Les connaissances que l'on sait stables n'ont pas besoin, quant à elles, d'être revalidées. On atteint donc un des objectifs de la mémoire collective, qui est d'assurer une cohérence entre les connaissances des clients et celle du serveur. On minimise du même coup les accès à la mémoire collective.

4.2.1 Niveau de confiance

Le niveau de confiance($\lambda(b, t_0)$) peut être interprété de deux façons: a) la valeur de vérité d'une brique ou b) le niveau de confiance que l'on peut porter envers la valeur de vérité. Similairement à [MASA1989], nous modélisons la fiabilité quant à la valeur de vérité à l'aide de trois catégories de valeurs: vrai(1), faux(-1) et indéterminé(0). L'état indéterminé signifie qu'il n'est pas possible pour le client de conclure quoi que ce soit à propos de la valeur de vérité associée à cette connaissance. Plus la valeur du niveau de confiance est loin de zéro et plus la croyance que l'on porte envers la valeur de vérité est forte. Le niveau de confiance d'une brique permet donc de déterminer la croyance que l'on porte, au moment présent, à la valeur de vérité associé à une brique. Cette valeur appartient donc à l'intervalle [-1,1].

Pour représenter le niveau de confiance que le client accorde envers la valeur de vérité d'une brique, nous utilisons une fonction de *niveau de confiance*. Cette fonction continue exprime toute une gamme de valeurs intermédiaires allant, de manière graduelle, du vrai au faux ou l'inverse. Nous définissons cette fonction de la façon suivante :

> **Définition 4-1** (*Niveau de confiance d'une brique*) Soit U, l'ensemble de toutes les briques constituant la base de connaissances de la mémoire collective et $g \in U$ une brique de l'ensemble. La fonction $\lambda : U \times temps \rightarrow \Re$ définit pour chaque brique g de U, un niveau de confiance à l'instant présent. Si $\lambda > 0$ alors

le client croit que l'information représentée par la brique est vraie. Si $\lambda < 0$ alors le client croit que l'information représentée par la brique est fausse. Autrement, Si $\lambda = 0$ alors le niveau de confiance est indéterminé.

4.2.2 Probabilité de changement

Comme nous l'avons déjà dit, nous avons émis l'hypothèse que le niveau de confiance fluctue par bons de tailles fixes selon une certaine probabilité de changement qui n'évolue pas dans le temps dans le temps. À chaque laps de temps *t* fixe préétabli, le niveau de confiance est mis à jour, localement chez le client, en fonction d'une probabilité de changement. Cette probabilité de changement nous donne la probabilité que le niveau de confiance s'est rapproché de zéro d'une certaine constante. Pour déterminer la tendance de l'évolution temporelle du niveau de confiance, nous définissons la fonction suivante :

Définition 4-2 (*Probabilité de changement*) Soit *U*, l'ensemble de toutes les briques constituant la base de connaissances de la mémoire collective et $g \in U$ une brique de l'ensemble. La fonction $\eta : U \rightarrow [0,1]$ défini pour chaque brique *g* de *U,* la probabilité que le niveau de confiance ait fluctué d'une certaine constante dans la direction qui, par rapport à la valeur de vérité $\lambda(g, t_0)$, mène à un changement de cette valeur de vérité.

Cette fonction permet de déterminer l'évolution du niveau de confiance et ainsi de synchroniser les mises à jour des connaissances acquises par les clients. Cette valeur permet de déterminer, pour un laps de temps *t*, la probabilité que le niveau de confiance subisse une augmentation/diminution. Inversement, *1-p* est la probabilité d'obtenir une diminution/augmentation du niveau de confiance après une période de temps *t*. Cette fonction permet donc de modéliser la dégradation, la restauration ou la stabilité du niveau de confiance envers la valeur de vérité associée à une brique. Ainsi, à chaque période de temps *t*, le niveau de confiance d'une brique est mis à jour suivant une probabilité *p* de fluctuer.

4.2.3 Valeur de fluctuation du niveau de confiance

Aux sections précédentes, nous avons défini formellement un niveau de confiance et une probabilité de changement. Cependant, nous n'avons pas défini l'influence qu'a la probabilité de changement sur le niveau de confiance. Quelles sont les valeurs que peut prendre le niveau de confiance selon la probabilité de changement? Pour déterminer la valeur de fluctuation du niveau de confiance selon une certaine propriété après un laps de temps t, nous définissons la fonction suivante :

> **Définition 4-3** (*valeur de fluctuation*) Soit *U*, l'ensemble de toutes les briques constituant la base de connaissances de la mémoire collective et $g \in U$ une

brique de l'ensemble. La fonction $\varphi : U \rightarrow \Re$ défini pour chaque brique *g* de *U* la valeur de fluctuation tel que cette valeur est entre 0 et $+\infty$.Selon nos hypothèses, nous supposons que pour une brique donnée, cette valeur est constante dans le temps

Par exemple, la Figure 4-1 montre le calcul effectué pour évaluer la confiance accordée à la valeur de vérité d'une brique. Lors de l'acquisition de cette brique, les trois fonctions $\lambda(g, t_0)$, $\eta(g)$ et $\varphi(g)$ retourne respectivement les valeurs de 0.2, 0.6 et 0.1. Le niveau de confiance de la brique observée possède donc au temps t_0, une valeur de niveau de confiance de 0.2 et après chaque intervalle de temps delta t, il y a une probabilité de 0.6 que cette valeur varie de -0.1[7]. Au départ, le niveau de confiance est supérieur à zéro, ceci nous indique que la valeur de vérité associée à la brique est vraie. Pour calculer la date de péremption de la valeur de vérité de cette brique, le client calcule les valeurs possibles que peut prendre le niveau de confiance en fonction de la probabilité de changement. Dans notre exemple, nous avons fixé arbitrairement ce temps à trois intervalles de temps. L'observation de la brique débute au temps t0 et se continue pendant trois unités de temps de longueur Δt (jusqu'à t_3). L'évaluation du niveau de confiance en la véracité de la brique porte à croire qu'il y a de fortes chances que la brique ne subisse pas d'altération de sa valeur de vérité (passant de vrai à faux) avant trois unités de temps. En effet, si on cumule les probabilités dont le niveau de confiance est supérieur à zéro, nous avons une probabilité de ~0.784 que la valeur de vérité ne change pas. Donc, de t_0 à t_3, le client peut avoir

[7] Comme la valeur de $\lambda(g, t_0)$ est positive, c'est par des fluctuations négatives qu'éventuellement la valeur de vérité changera et une probabilité de 0.4 qu'elle varie de +0.1.

confiance en la valeur de vérité (vrai) de la brique. Il n'a donc pas besoin de mettre à jour cette connaissance. Ainsi, à moins que l'information contenue dans la brique g soit jugée "extrêmement critique", il n'y aura pas besoin de mettre à jour cette connaissance.

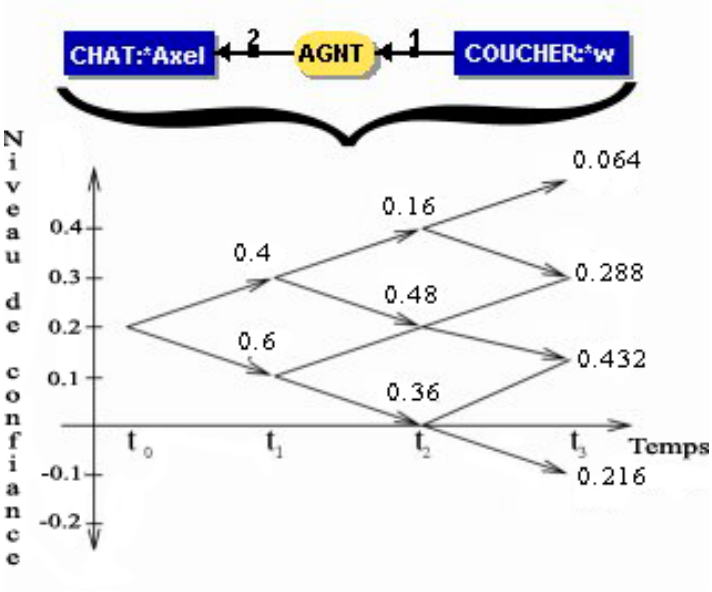

Figure 4-1 Estimation du niveau de confiance au temps t_3.

Le modèle proposé dans cette section permet de prédire si une connaissance est fiable et pour combien de tops d'horloge. Ce mécanisme permet de déterminer les moments propices de mise à jour des connaissances du client. Le client peut donc planifier les tâches à réaliser en fonction du niveau de confiance des valeurs de vérité de ses connaissances.

4.2.4 Représentation d'une brique

Pour que chaque brique puisse avoir un moment de mise à jour différent, le niveau de confiance, la probabilité de changement et la valeur de fluctuation sont encodés pour chaque brique. C'est le serveur qui est responsable de maintenir à jour ces valeurs pour chacune des briques. Ces informations sont envoyées vers les clients avec les briques résultant de l'interrogation de la mémoire collective. Ainsi, nous représentons cette information de la façon suivante :

Définition 4-4 (*Représentation d'une brique lors de la transmission*) Soit U, l'ensemble de toutes les briques constituant la base de connaissances de la mémoire collective et g une brique de l'ensemble U. On représente l'information envoyée vers un client, pour chaque brique concernée g, par un quintuplet $<g, t_0, \lambda(g, t_0), \eta(g), \varphi(g)>$ où $\lambda(g, t_0) \in \Re$ représente le niveau de confiance envers la valeur de vérité de la brique, $\eta(g) \in [0,1]$ représente la probabilité de changement d'état d'une brique pour un laps de temps fixe Δt et $\varphi(g) \in [0,+\infty[$ représente la valeur de fluctuation. Afin de différencier ces quintuplets des briques, on les notera $\hat{g}$ à la place de g et on notera par $\hat{U}$, l'ensemble de tous ces quintuplets.

Les paramètres transmis avec la brique permettent de déterminer les moments où les mises à jour doivent être effectuées. Dans la prochaine section, nous présentons trois

exemples d'utilisation des paramètres transmis par le serveur pour calculer les valeurs possibles que peut prendre le niveau de confiance.

4.3 Divers exemples de calcul de dégradation de valeur de vérité

Au cours de la section 4.2, nous avons présenté les éléments nécessaires à la technique que nous proposons pour la gestion des mises à jour des connaissances dans la mémoire collective. Pour mieux comprendre les notions présentées à la section 4.2, cette section présente trois exemples de calcul de dégradation de la valeur de vérité. Le premier exemple présente une brique dont la valeur de vérité se dégrade rapidement. Dans cet exemple, la brique possède une valeur de vérité qui est instable. Le client ne pourra donc pas avoir confiance envers sa valeur de vérité pour une longue période de temps. Le deuxième exemple présente une brique dont le niveau s'accroit tranquillement. Le client pourra donc avoir confiance envers la valeur de vérité pendant une longue période de temps. Enfin, le troisième exemple présente une brique où le niveau de confiance croit avec le temps. Le client pourra avoir confiance envers la valeur de vérité associée à la brique pendant une longue période de temps.

4.3.1 Exemple 1 : dégradation rapide

Par exemple, la Figure 4-2 montre le calcul effectué pour évaluer la confiance accordée à la valeur de vérité d'une brique dans le cas ou cette valeur se dégrade rapidement. Lors de l'acquisition de cette brique, les trois fonctions $\lambda(g, t_0)$, $\eta(g)$ et $\varphi(g)$ retournent respectivement les valeurs de -1, 0.9 et 1 donc une valeur de -1 et après chaque intervalle de temps Δt passé t_0, la brique a une probabilité de 0.9 d'avoir une variation de 1 et une probabilité de 0.1 d'avoir une variation de -1. Au départ, le niveau de confiance est inférieur à zéro, ceci nous indique que la valeur de vérité associée à la brique est fausse. L'observation de la brique débute au temps t0 et se continue pendant trois unités de temps (jusqu'à t3).

L'évaluation du niveau de confiance en la véracité de la brique porte à croire qu'il y a de fortes chances(0.972) que la brique subisse une altération de sa valeur de vérité (passant de faux à vrai) avant trois unités de temps. Même après deux unités de temps, le niveau de confiance nous indique qu'il y a de fortes chances que la valeur de vérité associée à la brique ait changée. Le client doit donc rafraichir cette brique pour voir si elle a été mise à jour.

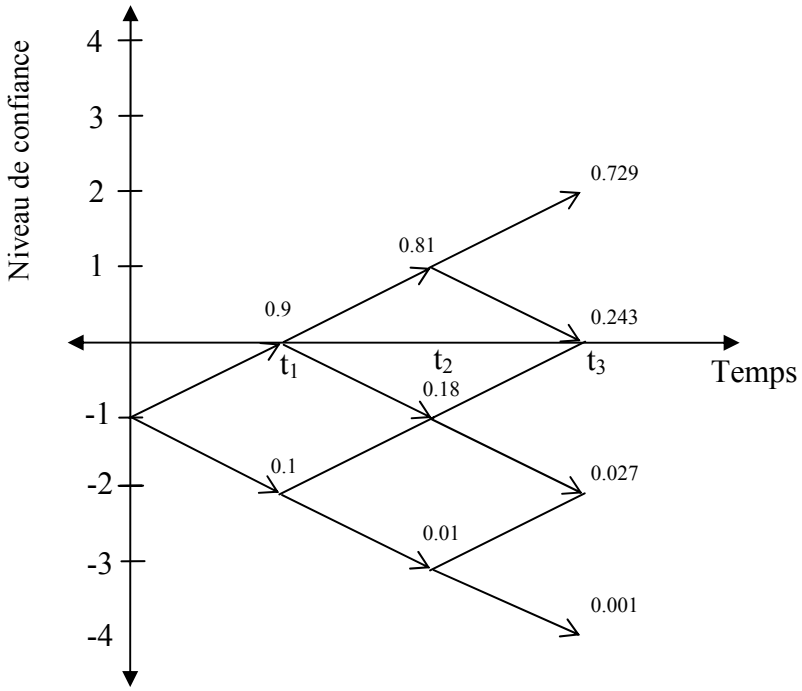

Figure 4-2 Estimation du niveau de confiance au temps t_3.

4.3.2 Exemple 2 : Accroissement du niveau de confiance

Par exemple, la Figure 4-3 montre le calcul effectué pour évaluer la confiance accordée à la valeur de vérité d'une brique dans le cas où cette valeur est stable. Lors de l'acquisition de cette brique, les trois fonctions $\lambda(g, t_0)$, $\eta(g)$ et $\varphi(g)$ retournent respectivement les valeurs de 1, 0 et 0.1. Le niveau de confiance de la brique observée possède donc une valeur de 1, une probabilité de changement de 0 et une valeur de fluctuation de 0.1. Au départ, le niveau de confiance est supérieur à zéro, ceci nous indique

que la valeur de vérité associée à la brique est vraie. L'observation de la brique débute au temps t_0 et se continue pendant trois unités de temps (jusqu'à t_3).

L'évaluation du niveau de confiance en la véracité de la brique présente une information constante du système. La probabilité que la brique subisse une altération de sa valeur de vérité est nulle; cette valeur ne se dégrade pas à travers le temps. Par conséquent, cette valeur n'aura jamais besoin d'être mise à jour.

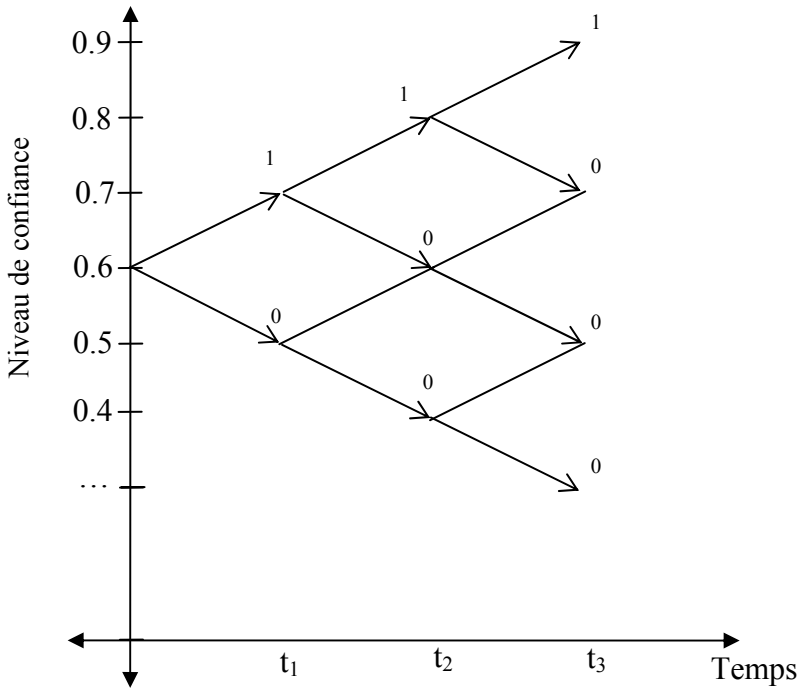

Figure 4-3 Estimation du niveau de confiance au temps t_3.

4.3.3 Exemple 3 : croissance du niveau de confiance

Par exemple, la Figure 4-4 montre le calcul effectué pour évaluer la confiance accordée à la valeur de vérité d'une brique dans le cas où, avec forte probabilité, cette valeur s'accroit tranquillement à chaque unité de temps. Lors de l'acquisition de cette brique, les trois fonctions $\lambda(g, t_0)$, $\eta(g)$ et $\varphi(g)$ retournent respectivement les valeurs de 0.6, 0.1 et 0.1. Le niveau de confiance de la brique observée possède donc une valeur de 0.6, une probabilité de changement de 0.1 et une valeur de fluctuation de 0.1. Au départ, le niveau de confiance est supérieur à zéro. Ceci nous indique que la valeur de vérité associée à la brique est vraie. L'observation de la brique débute au temps t_0 et se continue pendant trois unités de temps (jusqu'à t_3).

L'évaluation du niveau de confiance en la véracité de la brique porte à croire que les chances que la brique subisse une altération de sa valeur de vérité (passant de vrai à faux) est faible. Cependant, comme on peut le constater, éventuellement, il y aura une infime possibilité que la brique ait été mise à jour (altération de sa valeur de vérité). Plus le temps passera et plus la probabilité que le niveau de confiance soit négatif grandira. Le client peut donc attendre une longue période de temps avant de devoir rafraichir cette brique.

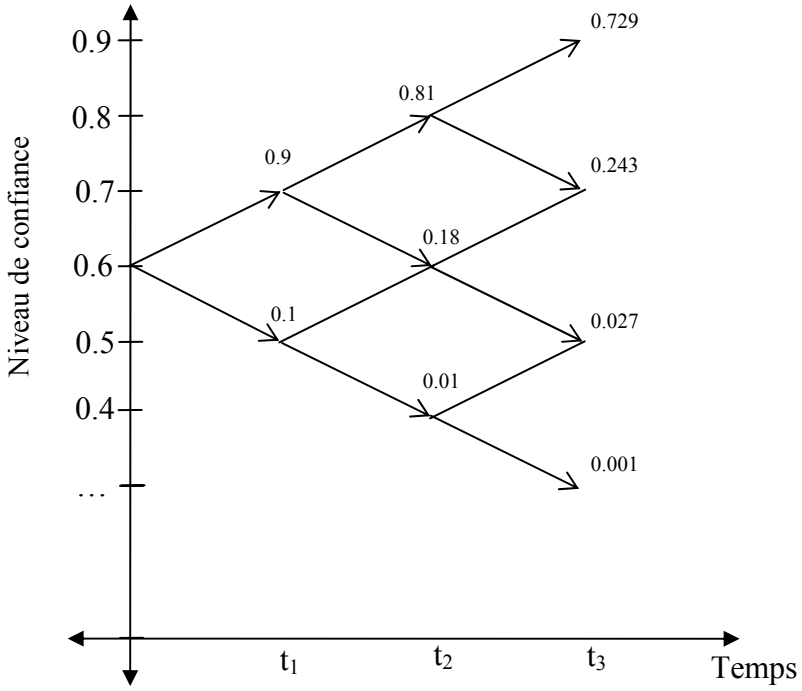

Figure 4-4 Estimation du niveau de confiance au temps t_3.

4.4 Fonction d'utilité

À l'aide des paramètres définis précédemment, on peut définir des fonctions d'utilités qui permettront aux clients de calculer une date de péremption. Dans les exemples précédents, la date de péremption correspondait au laps de temps pour lequel la probabilité que le niveau de confiance change ne soit plus acceptable. Chacune des fonctions d'utilité présentée dans cette section sera détaillée au Chapitre 5.

Nous définissons donc une fonction qui permet de déterminer le niveau de confiance que l'on porte envers une connaissance après *k* laps de temps *Δt*. Cette fonction d'utilité permet de déterminer si les connaissances que le client utilise n'ont pas à être mise à jour avant un certain moment. Ainsi, la fonction d'utilité γ_p calcule la probabilité que le niveau de confiance envers la valeur de vérité d'une brique après *k* laps de temps *Δt* n'est pas changé pour une certaine brique *g* :

Définition 4-5 (*Fonction d'utilité*) Soit U, l'ensemble de toutes les briques constituant la base de connaissances de la mémoire collective et soit $\hat{U}$ l'ensemble de tous les représentants des briques encodées de *U*. La fonction $\gamma_p : \hat{U}, \aleph \rightarrow [0,1] \cup \{-1\}$ définit pour chaque brique g de *U* la probabilité qu'une brique n'ait pas changé d'état dans après *k* laps de temps Δt. Si la fonction n'est pas capable de déterminer la probabilité du changement d'état, la valeur -1 est retournée. Le dernier cas se produit que pour des situations dégénérées comme les cas où $\lambda(g, t_0) = 0$.

De la même façon, nous définissons une autre fonction qui permet de déterminer le moment où le niveau de confiance envers une brique ne sera plus suffisant pour qu'on ait confiance en sa valeur de vérité. Cette fonction calcule, pour un *degré minimal d'assurance* α (défini à priori par chaque client pour chaque brique), le moment où nous ne pourrons plus avoir confiance en la valeur de vérité d'une brique. Le *degré minimal d'assurance* α détermine pour chaque brique notre « niveau de tolérance à l'erreur ». Ainsi, nous définissons cette fonction de la façon suivante :

Définition 4-6 (*Fonction d'utilité*) Soit $\hat{U}$, l'ensemble de tous les quintuplets qui représentent les briques constituant la base de connaissances de la mémoire collective, $\hat{g}$ un quintuplet de $\hat{U}$ et $\alpha \in [0,1]$ le degré minimal d'assurance que notre évaluation est correcte. La fonction $\gamma_t : \hat{U} \times [0,1] \rightarrow \aleph \cup \{-1, +\infty\}$ où γ_t représente le nombre d'intervalles de temps Δt nécessaires pour faire baisser sous la valeur α, la probabilité que la brique g ait la même valeur de vérité qu'au temps t_0. Si cette probabilité reste toujours au-dessus de α, on donne à γ_t la valeur $+\infty$, alors que la valeur -1 est retournée si on ne peut calculer cette probabilité. Autrement dit, pour un degré d'assurance d'au moins α, $\gamma_t(\hat{g}, \alpha)$ donne la date de péremption de la valeur de vérité donnée au temps t_0 à la brique.

Cette fonction permet donc de déterminer le moment où la valeur de vérité d'une brique ne sera plus fiable avec un degré minimal d'assurance α. L'utilisation de cette fonction permet aux clients de déterminer le moment de mise à jour d'une brique. De plus, le client peut choisir le seuil selon lequel il est tolérant, au risque que la connaissance véhiculée par la brique soit erronée. Encore une fois, la méthode de calcul de cette fonction sera expliquée au Chapitre 5.

L'évaluation que nous proposons dans cette section permet de déterminer si on peut avoir confiance ou non en la valeur de vérité initiale et si oui, pendant combien de temps. Ce mécanisme nous permet de calculer les moments propices de mise à jour de certaines briques visant ainsi à minimiser les accès à la mémoire collective.

4.5 Conclusion

Le modèle proposé dans cette section permet de calculer, pour un certain intervalle de confiance, les valeurs de vérité possibles que peut prendre une brique à travers le temps. Grâce à ce calcul, les clients peuvent avoir une bonne idée de la tendance de la valeur de vérité d'une connaissance et ainsi prévoir les mises à jour de leurs connaissances. Ainsi, ce mécanisme permet aux clients d'éviter l'incohérence momentanée de leur connaissance en leur indiquant quelles connaissances ne sont plus fiables et lesquels le sont. Lorsque le client juge que certaines connaissances ne sont plus fiables, il peut les mettre à jour périodiquement en accédant à la mémoire collective qu'à ces moments-là. Au Chapitre 5, la méthode permettant d'évaluer le niveau de confiance attribué à une brique ainsi que la technique de calcul de la péremption des valeurs de vérité sera discutée.

Chapitre 5 Mécanisme d'évaluation des valeurs de vérité associées aux connaissances

5.1 Introduction

L'objectif du Chapitre 4 était de formaliser le mécanisme qui permet de maintenir à jour les valeurs de vérité associées aux connaissances de la mémoire collective. Ce mécanisme permettait aux clients d'éviter l'incohérence momentanée des connaissances (un problème de concurrence) en ayant un niveau de confiance quant à la valeur de vérité des connaissances acquises. Comme nous l'avons expliqué au Chapitre 4, le serveur transmet les connaissances en incluant un niveau de confiance, une probabilité de changement et une valeur de fluctuation associée à chaque partie de connaissance. Dans le Chapitre 5, nous présentons les algorithmes relatifs au mécanisme de gestion des mises à jour qui permettent de calculer une date de péremption sur les connaissances.

Ce chapitre est donc consacré à l'évaluation du niveau de confiance futur des connaissances. Dans la section 5.2, le modèle probabiliste utilisé pour évaluer la date de péremption est présenté. Dans la section 5.3, le modèle probabiliste présenté dans la section précédente est appliqué au modèle de la mémoire collective. Les algorithmes de calcul de la date de péremption et du calcul de la probabilité de véracité sont détaillés. La section 5.4 présente un exemple d'utilisation du mécanisme. Cette section montre comment utiliser les

algorithmes présentés précédemment. La section 5.5, montre les avantages et inconvénients d'utiliser ce mécanisme comparativement à une méthode standard de gestion des mises à jour. La section 5.6, présente les conditions régissant le contexte d'application. Enfin, la section 5.7 fait un bref retour sur les notions vues dans ce chapitre.

5.2 Modèle statistique

Dans un système distribué, une mise à jour des connaissances peut être faite à n'importe quel moment. Lors de l'acquisition d'une connaissance par le client, la valeur de vérité reçue risque éventuellement d'être erronée si celle-ci évolue dans le temps. Par souci de simplification des calculs, nous considérons, dans ce qui suit, qu'une mise à jour peut subvenir à tous les intervalles de temps Δt. Ainsi, après chaque intervalle de temps Δt il y a quatre cas possibles quant à l'évolution des valeurs de vérité d'une brique (connaissance élémentaire):

1) une brique qui est vraie devient fausse;
2) une brique qui est fausse devient vraie;
3) une brique possède une valeur de vérité qui ne change pas;
4) la valeur de vérité d'une brique est indéterminée. Nous ne savons rien quant à sa valeur de vérité.

La solution que nous avons envisagée consiste à donner au client les outils nécessaires pour planifier ses mises à jour à l'aide d'une date de péremption pour chacune

des connaissances transmises. Pour arriver à déterminer une date de péremption, nous utilisons une méthode probabiliste qui permet d'évaluer le risque de mise à jour d'une connaissance. L'interprétation que nous portons à une mise à jour revient à calculer la probabilité que le niveau de confiance d'une connaissance subisse, pour une période de k unités de temps Δt, un certain nombre de diminutions (succès) du niveau de confiance d'une valeur. Le calcul donne la probabilité de réalisation de chacun des cas possibles. Ainsi, si on se base sur notre Définition 4-2 du paramètre η, on peut interpréter ce calcul comme étant une série de succès et d'échecs que subit le niveau de confiance tel que chaque succès représente une diminution[8] du niveau de confiance de ε alors que les échecs représentent une augmentation du niveau de confiance de ε. Avec cette interprétation, le modèle devient simple, il suffit d'utiliser la loi binomiale. La loi binomiale, telle que présentée dans [HM1990], est définie comme suit :

> **Définition 5-1** (*Loi binomiale*) Soit une série de n épreuves successives et indépendantes dont l'issue de chacune est un succès ou un échec, avec probabilités respectives de p et $q = 1 - p$ et soit X, la variable aléatoire qui représente le nombre de succès dans ces n épreuves, on dit que X est une variable aléatoire binomiale de paramètre n et p. De plus, la probabilité d'obtenir k succès en n tentatives est donnée par :

[8] Notons que lorsque nous parlons de diminution du niveau de confiance, nous faisons un abus de langage. Il serait préférable de parler de variation du niveau de confiance en la valeur de vérité de la brique dans le sens opposé au signe de $\varphi(g)$. En effet, le paramètre $\eta(g)$ donne la probabilité qu'après un intervalle de temps Δt, il ait une variation d'une valeur de $-\varphi(g)$ de la valeur de vérité $\lambda(g, t_0)$ de la brique g. Ainsi, si par exemple, $((g, t_0) < 0$ (c'est-à-dire que la valeur de vérité est fausse) et si $\varphi(g) > 0$, on aura avec probabilité η (g), une augmentation de la valeur de $\lambda(g, t_0)$, que nous interprétons donc comme une diminution de notre niveau de confiance en la valeur de vérité que la brique avait au temps t_0.

$$P(X=k) = P(Bin(n, p)=k) = \binom{n}{k} p^k q^{n-k}$$

Par exemple, prenons une brique ayant un niveau de confiance de 0.4, une probabilité de changement de 0.2 et une valeur de fluctuation de 0.2. Après 3 unités de temps Δt, la probabilité que le niveau de confiance ait subi exactement 1 diminution (et donc exactement 3-1 augmentations) est de :

$$P(X=1) = P(Bin(3, 0.2)=1) = \binom{3}{2} 0.2^1 0.8^{(3-1)} = 0.384$$

L'outil statistique que nous cherchons à implanter doit produire la probabilité pour que le niveau de confiance d'une valeur de vérité soit supérieur (ou inférieur) à un certain seuil. Dans la prochaine section, nous verrons comment la binomiale est utilisée pour calculer une date de péremption (c'est-à-dire, la fonction d'utilité γ_t présenté à la Définition **4-6**) à l'aide des paramètres d'une brique.

5.3 Mécanisme d'évaluation de la valeur de vérité associée à une connaissance

Dans la section précédente, nous avons vu un outil qui nous permet d'évaluer la distribution des probabilités d'un événement aléatoire. Nous avons vu qu'il était possible de calculer la probabilité de succès d'événements parmi un certain nombre de tentatives. Dans

cette section-ci, nous présentons comment appliquer ce modèle probabiliste au mécanisme de gestion des mises à jour de la mémoire collective. De plus, nous verrons comment calculer les fonctions d'utilités γ_p et γ_t présentées au Chapitre 4, lesquelles servent à déterminer la probabilité de changement après k unités de temps et à déterminer la date de péremption d'une connaissance.

Évaluer un niveau de confiance au temps initial nous intéresse peu puisque nous connaissons la valeur de vérité de la brique lors de l'acquisition. Nous voulons évaluer le niveau de confiance à un temps futur. Le mécanisme proposé calcule la probabilité que le niveau de confiance associé à une brique n'ait pas changé de signe après un certain laps de temps. Pour un temps futur, il faut calculer les probabilités de baisse et de hausse du niveau de confiance à l'aide de la loi binomiale et des paramètres d'une brique (niveau de confiance, probabilité de changement et la valeur de fluctuation). Le lemme suivant sera un outil pour cet objectif.

Lemme 5-1 (*Limite de l'observation*) **:** Soit g une brique de U et soit $\hat{g} = \langle g, t_0, \lambda(g,t_0), \eta(g), \varphi(g) \rangle$, un quintuplet de $\hat{U}$ associé à g tel que $\lambda(g,t_0) \neq 0$ et $\varphi(g) \neq 0$. Alors, la valeur de vérité de la brique g ne sera pas la même au temps $t_0 + \Delta t$ qu'au temps t_0 si et seulement si le nombre de fois qu'il y a eu diminution du niveau de confiance est strictement plus grand que :

$$\frac{|\lambda(g,t_0)|}{2 \times \varphi(g)} + \frac{k}{2}$$

Démonstration : Soit $k \in \aleph$ un nombre d'intervalles de temps fondamentaux Δt et soit $j \in \aleph$ le nombre de fois entre le temps t_0 et $t_0 + \Delta t$ où il y a eu diminution du niveau de confiance (c'est-à-dire, le nombre de fois où la variation qu'a subi λ était de signe opposé à sa valeur initiale $\lambda(g, t_0)$).

Alors nous devons montrer que, la valeur de vérité de la brique g au temps $t + k\Delta t$ est différente de sa valeur au temps t_0 si et seulement si :

$$j > \frac{|\lambda(g,t_0)|}{2 \times \varphi(g)} + \frac{k}{2} \qquad (1)$$

Notons que par de simples calculs, la condition (1) se ramène à :

$$0 > |\lambda(g,t_0)| + (k - 2j) - \varphi(g) \qquad (2)$$

Maintenant, il y a quatre cas à considérer :

CAS 1 : $\lambda(g,t_0) > 0$ et $\varphi(g) > 0$

Alors $\lambda(g,t_0+k\Delta t) = \lambda(g,t_0) + j \times (-\varphi(g)) + (k - j) \times \varphi(g)$

$= \lambda(g,t_0) + (k - 2j) \times \varphi(g)$

Nous avons donc que la valeur de vérité aura changé si et seulement si

$$0 > \lambda(g,t_0) + (k - 2j) \times \varphi(g)$$

Le **cas 1** est donc démontré. Le cas 2, 3 et 4 qui sont respectivement ($\lambda(g,t_0) > 0$ et $\varphi(g) < 0$), ($\lambda(g,t_0) < 0$ et $\varphi(g) > 0$) et ($\lambda(g,t_0) < 0$ et $\varphi(g) < 0$) se montre de façon similaire.■

Ainsi, pour calculer la valeur de $\gamma_p(\hat{g},k)$, c'est-à-dire la probabilité qu'il n'y ait pas de changement de signe du niveau de confiance se produise, entre les temps t_0 *et* $t_0+\Delta t$, nous avons le résultat suivant :

Proposition 5-1 (*Probabilité qu'il n'y ait pas de changement de signe du niveau de confiance*) La probabilité qu'il n'y ait pas de changement de signe du niveau de confiance d'une brique *g* est donné entre le temps t_0 et $t_0 + \Delta t$ est tel que :

$$\gamma_p(\hat{g}, k) = \mathrm{P}(Bin(\mathrm{k}, \eta(\mathrm{g})) \geq \mathrm{k\text{-}j})$$

$$= \sum_{i=k-j}^{k} \binom{k}{i} \eta(g)^i (1-\eta(g))^{k-i}$$

où j est le grand entier plus petit ou égale à $\frac{|\lambda(g,t_0)|}{2\times\varphi(g)} + \frac{k}{2}$

et où $j \leq k$.

Preuve : Conséquence directe du lemme 5.1 et des hypothèses 1, 2 et 3.

Cette définition permet de calculer la probabilité qu'une brique n'ait pas changé d'état après *k* unités de temps Δt. On peut donc calculer, pour n'importe quel intervalle de temps, la probabilité que le niveau de confiance nous indique la bonne valeur de vérité. Ainsi, l'algorithme suivant permet de déterminer la probabilité que la valeur de vérité d'une brique *g* n'ait pas changée.

```
Algorithme 5 (Intervalle de confiance d'une brique après k intervalle
       de temps)
γp (une brique g, un intervalle de temps k)
//λ(g, t0) la fonction qui retourne le niveau de confiance d'une brique
//          g.
//η(g) la fonction qui retourne la probabilité de changement d'une
//     brique g.
//φ(g) la fonction qui retourne la valeur de fluctuation d'une
//     brique g.
//Bin(n, p) variable aléatoire binomiale de nombre de tentative n et de
//           probabilité de succès d'une tentative p.
DÉBUT
   SI λ(g, t0)) = 0 ALORS
      Retourner -1

   prob = 0
   POUR TOUT i ∈ [0, k] FAIRE
   DÉBUT
     Si (k - i) ≤ Abs(λ(g)) / (2 × φ(g)) + k / 2) ALORS
        prob = prob + P(Bin(k, η(g)) = i))
   FIN

   Retourner prob
Fin
```

Cet algorithme permet de déterminer la probabilité qu'une brique ait la même valeur de vérité après laps de temps de k unités de temps. Par exemple, si on prend une brique g dont la valeur de vérité est vraie au temps t_0, la probabilité que nous croyions que cette brique est encore vraie après k laps de temps est de $\gamma_p(\hat{g}, k)$. Cet algorithme permet donc de prévoir avec quelle proportion une connaissance risque d'être encore valide après un laps de temps k.

De plus, la fonction γ_p permet de calculer les dates de péremption des briques (la fonction γ_t). La proposition suivante précise comment.

Proposition 5-2 (*Calcul de la date de péremption d'une brique*) Soit *U*, soit g une brique de U et $\hat{g} = \langle g, t_0, \lambda(g,t_0), \eta(g), \varphi(g) \rangle$ un quintuplet de Û associé à la brique g telle que $\lambda(g,t_0)$ et $\varphi(g)$ sont nuls. Soit $\alpha \in [0,1]$ le degré minimal d'assurance associé à la brique *g*. Alors la date de péremption de $\hat{g}$ est donné par la formule :

$$\gamma_{t(}\hat{g}, \alpha_{)} = \text{Sup}\{ k : \gamma_{p}(\hat{g}, k) < \alpha \}$$

Preuve : une conséquence directe de la proposition 5.2 et de la définition de γ_{t} *et* $\gamma_{p.}$ ■

De la même façon, nous définissons un autre algorithme qui permet de déterminer le moment où la probabilité ne sera plus suffisante pour que l'on ait confiance en la valeur de vérité d'une brique. Cet algorithme calcule, pour un degré minimal d'assurance α, le moment où nous ne pourrons plus avoir confiance en la valeur de vérité d'une brique. Cet algorithme détermine donc la date de péremption pour la valeur de vérité d'une brique selon l'importance que lui accorde le client. Ainsi, nous définissons cet algorithme de la façon suivante :

```
Algorithme 6 (Calcul d'une date de péremption)
γt (un quintuplet ĝ de Û associé à une brique g envoyé au client en un temps t0,
   un degré minimal d'assurance α)
//Δt représente un intervalle de temps constant
DÉBUT
      t = t0
      SI γp (ĝ ,t) ≠ -1 ALORS
      DÉBUT
            SI γp (ĝ , t) > γp (ĝ , t+ Δt) ALORS
            DÉBUT
                  TANT QUE γp (ĝ ,t) ≥ α FAIRE
                        t = t + 1
                  Retourner t
            FIN
            SINON
                  Retourner INFINI
      FIN
      SINON
            Retourner -1
Fin
```

La proposition suivante montre que si l'algorithme retourne « INFINI » (c'est-à-dire si $\gamma_p(\hat{g},t) \geq \gamma_p(\hat{g},t+\Delta t)$), c'est que $\gamma_t(\hat{g},\alpha)$ est, comme il se doit, « INFINI».

Proposition 5-3 : Soit $t \geq t_0$, $\alpha \in [0,1]$ et g une brique de U. Soit également $\hat{g} = \langle g, t_0, \lambda(g,t_0), \eta(g), \varphi(g) \rangle$, un quintuplet de $\hat{U}$ associé à g tel que $\lambda(g, t_0) \neq 0$ et $\gamma_p(\hat{g}, t) \neq -1$. Alors,

$$\gamma_p(\hat{g},t) \geq \gamma_p(\hat{g},t+\Delta t) \Rightarrow \gamma_p(\hat{g},\alpha) = INFINI$$

Démonstration : Rappelons que selon les définitions 4.3 et 4.3, $\eta(g) \in [0,1]$ est la probabilité que le niveau de confiance varie de $-\varphi(g)$ et rappelons également que

$\forall t > t_0 \quad \gamma_p(\hat{g}, t)$ est la probabilité que la brique *g* n'ait pas la même valeur de vérité au temps *t* qu'au temps initial t_0.

Ainsi, s'il existe une valeur $t_1 > t_0$ tel que $\gamma_p(\hat{g}, t_1) > \gamma_p(\hat{g}, t_1 + \Delta t)$, c'est que φ(g) < 0. On aura donc que $\forall t_2 > t_0 \quad \gamma_p(\hat{g}, t_2) > \gamma_p(\hat{g}, t_2 + \Delta t)$, ce qui implique que $\gamma_t(\hat{g}, \alpha) = INFINI$.

D'autre part, s'il existe une valeur $t_1 > t_0$ tel que $\gamma_p(\hat{g}, t_1) > \gamma_p(\hat{g}, t_1 + \Delta t)$ c'est que $\varphi(g) = 0$ ou $\eta(g) = 0$. Dans chaque cas, nous avons que $\gamma_p(\hat{g}, t_2) > \gamma_p(\hat{g}, t_2 + \Delta t)$ pour tout $t_2 > t_0$. Nous avons donc que $\gamma_t(\hat{g}, \alpha) = INFINI$.■

L'algorithme présenté ci-dessus permet de calculer, pour un certain seuil, la date de péremption d'une brique. Ce calcul se fait sur les briques indépendamment les unes des autres. Ainsi, il est possible que, pour un degré minimal d'assurance α, une brique ne soit plus vraie alors que le reste du graphe conceptuel qui représente la connaissance soit encore vrai. Dans ce cas, le client juge s'il est nécessaire de mettre à jour les connaissances qui lui font défaut. Une connaissance peut donc ne pas être entièrement vraie sans pour autant que le client mette à jour ses connaissances.

En plus de la date de péremption, on peut aussi définir la date d'activation de la brique. Autrement dit, on peut déterminer le moment où l'on peut être certain, selon un certain seuil, que la valeur de vérité de la brique a changé :

Définition 5-2 (*Calcul de la date d'activation d'une brique*) Soit *U*, l'ensemble de toutes les briques constituant la base de connaissances de la mémoire collective, $\hat{g} = \langle g, t_0, \lambda(g,t_0), \eta(g), \varphi(g) \rangle$, un quintuplet de $\hat{U}$ et $\alpha \in [0,1]$ le degré minimal d'assurance que notre évaluation est correcte. La probabilité d'un changement de signe du niveau de confiance d'une brique *g* est donné pour une période d'évaluation $k \times \Delta t$ à l'aide de la fonction $1 - \gamma_p(g, k \times \Delta t)$. Grâce à la dernière fonction, on peut calculer le moment où le niveau de confiance d'une brique *g* de *U* aura changé de signe selon un degré minimal d'assurance α. Cette valeur est le moment correspondant au plus petit entier *k* pour lequel on a une probabilité plus grande que α que la valeur de vérité de *g* au moment $t_0 + k \times \Delta t$ soit de signe inverse à sa valeur de vérité au moment t_0. Autrement dit :

$$\underset{k}{MIN} \left\{ t_0 + \Delta t : \; 1 - \gamma_p(\hat{g}, t_0 + \Delta t) \; \geq \alpha \right\}$$

Comme nous l'avons dit dans le premier chapitre de ce mémoire, la mise à jour d'une brique occasionne un coût pour le client. Le client doit donc accepter d'utiliser des connaissances qui sont plus ou moins fiables dans le but d'accélérer son traitement. Par exemple, dans la compétition de soccer entre robots, la *RoboCup*, la position des autres robots change à tout moment ; ceux-ci ne peuvent donc pas s'arrêter et constamment mettre à jour leur connaissance. Les robots doivent se déplacer et faire des passes en risquant de perdre le ballon si le receveur de passe a modifié sa position. Ainsi, le calcul de la date de

péremption à l'aide d'un seuil de tolérance est sans contredit bénéfique dans ce genre de situation.

Dans notre modèle de la mémoire collective, nous conseillons d'utiliser la plus petite date de péremption des briques pour déterminer la date de péremption d'un graphe entier. Cependant, il peut en être autrement dépendamment de l'importance accordée à chacune des briques.

Le modèle de mise à jour est donc basé sur des intervalles de confiance pour déterminer le risque qu'une connaissance soit altérée. Cependant, l'interprétation que le client porte à au niveau de confiance est en fonction du domaine d'application. Cet aspect de notre travail sera examiné lors de travaux futurs. À la prochaine section, nous verrons, par un exemple, comment appliquer le modèle expliqué dans cette section.

5.4 Exemple

Nous proposons maintenant d'étudier un exemple d'évaluation du niveau de confiance en nous basant sur le modèle de calcul présenté précédemment. Pour notre exemple, nous utilisons le graphe de requête présenté à la Figure 5-1. Le client interroge la mémoire collective pour retrouver un ensemble de briques qui couvre le graphe de requête et qui représente une projection telle que présentée au Chapitre 4. Lors de l'acquisition des

briques, les paramètres qui permettent l'évaluation du niveau de confiance ($\lambda(g)$,$\eta(g)$ et $\varphi(g)$) sont fournis pour chacune de ces briques.

Lors de l'interrogation de la base de connaissances, nous avons retrouvé deux briques compatibles. Ces deux briques sont présentées avec leurs paramètres au bas de la Figure 5-1.

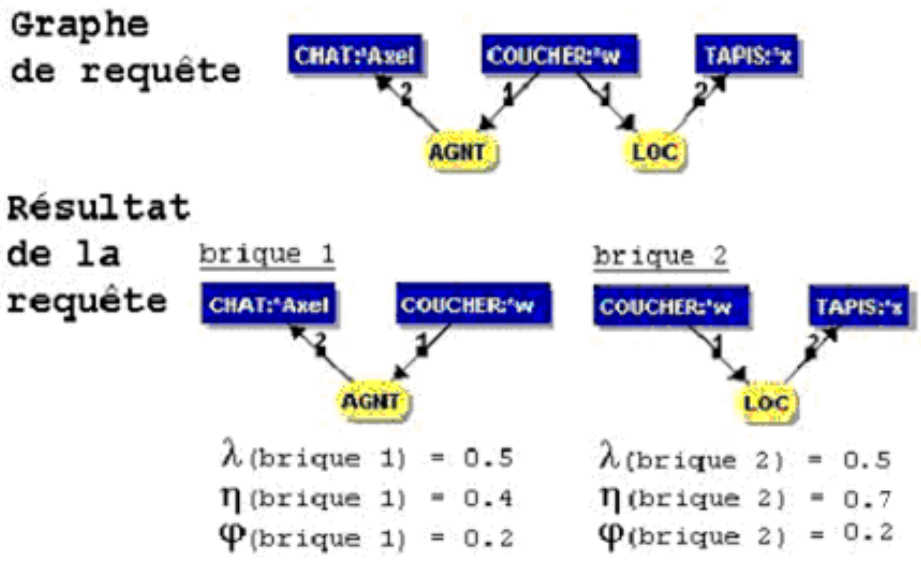

Figure 5-1 Exemple d'évaluation des briques d'une requête.

L'évaluation de l'intervalle de confiance se fait sur chacune des briques indépendamment les unes des autres. Pour interpréter les résultats, le client doit choisir une valeur seuil α pour chaque brique qu'il évalue. Dans cet exemple, nous avons décidé de fixer le paramètre α à 75 pour cent pour les deux briques.

Pour bien comprendre le processus d'évaluation, nous présentons le résultat de chacune des itérations du calcul de l'évaluation de la date de péremption. Ainsi, nous déterminerons le moment où le niveau de confiance ne sera plus suffisant pour que nous ayons confiance en la valeur de vérité de la brique. Pour chacune des itérations de l'algorithme γ_t , le résultat est présenté dans le Tableau 5-1. L'intervalle de confiance γ_p est calculé pour l'intervalle de temps Δt.

Intervalle de temps Δt	**Brique 1**	**Brique 2**
-	γ_p(brique 1, Δt)	γ_p(brique 2, Δt)
0Δt	1	1
1Δt	1	1
2Δt	1	1
3Δt	0,936	0,657

Tableau 5-1 Valeur de l'évaluation des briques 1 et 2.

Lors de la troisième évaluation, la *brique 2* possède un seuil inférieur à α (0,657 < 0.75). Ainsi, on peut en conclure qu'il est fort probable que la *brique 2* ait changé d'état après $3 \times \Delta t$ laps de temps. Dans ce cas, nous ne pouvons plus nous fier à la valeur de vérité de la brique 2.

5.5 Analyse de la technique proposée

Dans cette section nous comparons la technique que nous proposons avec une approche plus traditionnelle de gestion des mises à jour telle qu'utilisée dans un système de recherche d'informations basé sur les graphes conceptuels. Dans ce mécanisme, l'utilisateur fait une requête au serveur pour s'assurer de toujours avoir une information cohérente avant d'utiliser cette information. Cette méthode est utilisée dans beaucoup de systèmes, même les systèmes utilisant des graphes conceptuels comme COGITANT [COGIT2007].

Imaginons l'exemple suivant : une connaissance est utilisée à toutes les secondes alors qu'elle est mise à jour à toutes les cinq secondes sur le serveur. Avec notre mécanisme de gestion des valeurs de vérité des associées aux connaissances, si les paramètres transmis avec la brique représentent bien le phénomène de mise à jour, le client ne fera qu'une seule requête pour avoir une réponse de qualité. Avec la méthode qui met à jour les connaissances à chaque utilisation, quatre mises à jour seront faites sans que la valeur n'ait changé. Ainsi, selon la méthode que nous proposons, le trafic généré par la mise à jour des connaissances est grandement diminué. Cependant, le problème réside dans la validité des paramètres transmis avec chacune des briques. Conséquemment, les problèmes suivants demeurent :

1. Les paramètres reflètent-ils bien la réalité ?
2. Comment déterminer les valeurs initiales des paramètres ?
3. Le serveur a l'obligation de maintenir à jour pour chacune des connaissances qu'il a à gérer. Comment cela peut-il se faire efficacement ?
4. Pour les domaines en très grande évolution, le modèle de calcul n'est-il pas plus lourd que de tout mettre à jour à chaque top d'horloge?
5. Quels sont les impacts du domaine d'application sur ce modèle de calcul ?

Dans ce qui suit, nous présenterons le contexte d'application qui justifierait ou non l'utilisation du modèle de calcul proposé. Nous tenterons de répondre, bien que partiellement, à ces cinq questions.

5.6 Contexte d'application

Bien que le mécanisme de gestion des mises à jour présenté dans ce mémoire puisse s'appliquer à n'importe quel système, il n'en reste pas moins qu'il soit plus approprié dans certains cas que dans d'autres. Certains domaines d'application permettent de mieux caractériser les mises à jour des connaissances à l'aide d'une évaluation statistique.

Ainsi, sachant que l'efficacité de ce mécanisme repose sur la qualité du choix des paramètres pour représenter la fluctuation des valeurs de vérité des connaissances par rapport au domaine, un réglage judicieux des paramètres s'impose. Cependant, il n'existe pas de formule magique qui permette de déterminer avec certitude les valeurs de ces paramètres. Les valeurs que doivent prendre les paramètres des briques sont propres à chaque domaine d'application, voire chaque connaissance. Il revient donc aux concepteurs du système de vérifier la tendance de fluctuation des connaissances pour paramétrer leur système de la façon la plus représentative possible, à savoir, est-ce que les connaissances sont stables, prévisibles, etc.

Évidemment, le modèle présenté dans ce mémoire s'applique mieux aux domaines où les connaissances sont mises à jour à des intervalles réguliers, ou du moins où les mises à jour sont prévisibles à l'aide d'une analyse statistique. Si un domaine impose des mises à jour à tous les tops d'horloge, le modèle présenté dans ce mémoire n'est d'aucune utilité. Il faut donc qu'il y ait une période de temps raisonnable entre chacune des mises à jour. Si le domaine d'application subit des mises à jour à des intervalles de temps plus ou moins réguliers, notre modèle est adéquat.

Les paramètres transmis doivent donner une bonne idée des temps pour lesquels les valeurs de vérité associées aux briques n'ont pas changé. Cependant, il n'est pas certain que ces paramètres reflètent bien la vérité. Nous conseillons cependant aux concepteurs de

système d'utiliser une approche plus conservatrice pour les paramètres à utiliser de façon à diminuer le risque d'erreur, quitte à augmenter le nombre de requêtes. Dans un tel cas, il revient au client de déterminer son seuil de tolérance à l'erreur. Comme nous l'avons dit dans le Chapitre 4, le client doit évaluer la date de péremption des briques en fonction d'un degré minimal d'assurance α, où cette valeur représente la tolérance au risque qu'une connaissance ne soit pas à jour. Le client peut choisir le seuil selon lequel il est tolérant, au risque que la connaissance véhiculée par la brique soit erronée.

Pour déterminer les valeurs des paramètres, nous proposons une solution. Le système pourrait initialiser les paramètres pour chacune des briques de façon à ce qu'il y ait une mise à jour à chaque top d'horloge. Ensuite, à chaque intervalle de temps, les valeurs des paramètres sont mises à jour en fonction de la fluctuation des valeurs de vérité associées aux connaissances sur le serveur. Une analyse statistique des fluctuations permettrait de faire converger les valeurs de paramètres de façon à ce que ces paramètres représentent bien le phénomène des mises à jour des connaissances. Une fois que la stabilité des valeurs des paramètres est atteinte, le serveur peut cesser de mettre à jour ces valeurs tant qu'aucune modification significative n'apparaît quant à la fluctuation des valeurs de vérité. Ainsi, au départ il sera difficile pour le serveur de maintenir à jour les paramètres des connaissances, mais dès que ces paramètres convergeront vers une valeur stable, le serveur n'aura qu'à vérifier la validité de ces valeurs par rapport au phénomène des mises à jour des connaissances.

5.7 Conclusion

Dans la mémoire collective, les connaissances évoluent en temps réel. Des mises à jour des connaissances peuvent donc subvenir à tout moment. Nous avons donc dû imaginer un mécanisme qui permet aux clients de calculer le degré de fiabilité de chacune des connaissances afin d'éviter une incohérence momentanée des connaissances.

Dans ce chapitre, nous avons présenté la méthode utilisée pour gérer les mises à jour des valeurs de vérité associés aux connaissances de la mémoire collective. La méthode que nous avons proposée permet au client d'éviter une incohérence momentanée des connaissances en calculant une date de péremption pour chacune des parties de connaissance. Le client n'a pas besoin de constamment mettre à jour ses connaissances pour s'assurer de la validité de celles-ci. De plus, le client met seulement à jour les parties de connaissances qu'il juge ne plus être suffisamment fiables, malgré que toujours utiles à ses tâches, réduisant ainsi le nombre de mises à jour et la taille des mises à jour.

Enfin, nous avons présenté les algorithmes des fonctions d'utilité présentées au Chapitre 4. Ces algorithmes nous permettent d'évaluer, selon un certain intervalle de confiance, la probabilité qu'une connaissance nécessite une mise à jour. Le client peut calculer une date de péremption pour chacune des connaissances qu'il possède. Il utilise les paramètres transmis avec les briques acquises pour évaluer leur date de péremption. Ce

calcul permet de déterminer la confiance qu'on peut avoir envers la valeur de vérité d'une brique.

Au cours du prochain chapitre, nous verrons quels sont les mécanismes existants pour la gestion des mises à jour des connaissances dans les systèmes existants. Les avantages et les inconvénients de chacun seront présentés.

Chapitre 6 Techniques de mises à jour des connaissances

6.1 Introduction

Dans la plupart des systèmes distribués, les données ont besoin d'être répliquées entre le serveur et les clients pour améliorer la disponibilité des données et les temps de réponse des requêtes. Dans de telles circonstances, la mise à jour des connaissances devient problématique; on ne peut pas réaliser une mise à jour sur le serveur et les clients en même temps. Ainsi, ce chapitre présentera l'état de l'art des méthodes existantes de propagation des mises à jour dans un système distribué. L'objectif de ce chapitre est donc de situer notre approche(Chapitre 4 et Chapitre 5) par rapport à celles existantes. Pour chacune des méthodes présentées, nous verrons les avantages et inconvénients de chacune ainsi que le type d'application auxquels elles s'adressent.

Dans ce qui suit, nous présentons une vue d'ensemble des méthodes de propagation des mises à jour les plus connues. Nous classifierons les méthodes de propagation des mises à jour en trois catégories, selon les principes [CHKS1994], [CHKS1995] suivants: *synchrone*, *asynchrone* ou *hybride*.

Comme son nom l'indique, la méthode *synchrone* synchronise les mises à jour entre le serveur et les clients. Tant qu'une mise à jour n'est pas validée pour chacun des éléments

du système (clients et serveur), la mise à jour n'est pas effective. Ainsi, si une erreur survient lors du rafraichissement des connaissances, cette modification est défaite et la base de connaissances redevient comme elle était avant la mise à jour. La méthode synchrone a l'avantage de garantir en tout temps la cohérence des connaissances. Dans ce cas, la cohérence mutuelle des connaissances est garantie.

La méthode a*synchrone*, quant à elle, rend effective la mise à jour sur le serveur et propage par la suite cette mise à jour avec les clients dans une transaction différente. La demande de propagation est basée sur un événement qui peut être initié tant par le serveur que le client. Par exemple, une demande de propagation peut être faite après une mise à jour sur le serveur, après une certaine période de temps, à la demande d'un client, etc. Dans le cas où une erreur survient lors de la propagation d'une mise à jour, la modification n'est pas défaite sur le serveur. Le serveur possède alors une version différente de ses connaissances. Un mécanisme doit donc être utilisé pour gérer cette incohérence. La méthode asynchrone a l'avantage d'être plus rapide que la méthode synchrone. Cependant, la cohérence mutuelle des connaissances n'est pas garantie.

Dans la section 6.2, nous présentons une technique de propagation des mises à jour de type synchrone : la technique de *validation à deux phases*. Dans la section 6.3, nous décrivons une technique de propagation des mises à jour de type asynchrone : la technique du *serveur paresseux*. La section 6.4 présente une technique hybride entre les deux techniques présentées dans les sections précédentes. Enfin, dans la section 6.5, nous faisons un bref survol des techniques de propagation des mises à jour dans un système multiagents.

6.2 La méthode synchrone

6.2.1 Validation à deux phases

La technique de *validation à deux phases* permet de coordonner les mises à jour effectuées avec tous clients [GRA1979], [HT1988], [BHG1987]. Le principe consiste ici à diviser la transaction en deux phases. La première phase envoie un message qui demande aux clients de se préparer pour la mise à jour des éléments modifiés sur le serveur. La seconde phase, réalisée seulement en cas de succès de la première phase, valide la mise à jour des connaissances.

Lors la première étape, le serveur demande aux clients s'ils sont prêts à commettre leurs mises à jour. Si tous les participants répondent positivement, tous les clients effectuent leur validation. Si un participant n'est pas prêt et répond négativement, le serveur demande à tous les autres clients de défaire la transaction.

Cette technique assure qu'il n'y a pas d'incohérence momentanée des connaissances. Cette technique de mise à jour est efficace seulement pour les applications

où le nombre de clients est réduit et où l'accès au réseau est rapide et peu couteux. De plus, lorsqu'un message de mise à jour est transmis aux clients, ceux-ci doivent attendre la fin du transfert de l'information pour continuer à exécuter leur tâche. Les clients deviennent donc dépendants des mises à jour du serveur.

Ce genre de techniques est particulièrement bien adapté pour des applications où l'exactitude des informations est primordiale, par exemple pour des institutions financières. Le serveur gère les mises à jour des connaissances et s'assure que celle-ci soit valable et effective avant de l'appliquer à l'ensemble des clients.

6.3 La méthode asynchrone

Une alternative à la méthode synchrone est l'utilisation des techniques de réplication asynchrone. Dans ce cas, la cohérence mutuelle n'est plus garantie et le concept de fraicheur sert à mesurer les différences existantes entre les répliques de données. Dans ce qui suit, nous présenterons une technique de propagation des mises à jour des connaissances de façon asynchrone.

6.3.1 Serveur paresseux

La technique du *serveur paresseux* est une technique asynchrone très populaire en industrie [GH0S1996], [LAD1990]. Cette technique est adaptée pour les systèmes qui ont une tolérance aux erreurs et qui nécessitent plus de performance qu'en offre la technique synchrone. Avec cette technique, les mises à jour sont d'abord sauvegardées sur le serveur. Les clients mettent à jour, de façon asynchrone, leurs connaissances avec celles du serveur dans une transaction séparée de celle du serveur en fonction d'un certain événement : demande du client, laps de temps, etc.

Avec cette technique, la demande de propagation de la mise à jour peut être initiée soit par le serveur ou soit par le client. Si la demande de propagation est initiée par le serveur, celui-ci peut rapidement transmettre aux clients la nouvelle version des mises à jour. Cette technique de propagation basée sur un événement (déclencheur) sera présentée dans la prochaine section. Par contre, si la demande de propagation est initiée par le client, la détection d'une mise à jour doit être faite à l'aide de requêtes provenant du client. Dans ce cas, il est plus difficile pour le client d'être informé des moments où les mises à jour sont réalisées. L'avantage de cette approche est qu'elle est extensible et elle réduit le trafic sur réseau. Seules les mises à jour nécessaires à la réalisation de la tâche des clients leur sont transmises; contrairement à l'approche synchrone où toutes les mises à jour sont transmises aux clients. Cette technique a aussi l'avantage de décentraliser le traitement relatif à la

propagation des mises à jour. Pour déterminer si une mise à jour faite sur le serveur doit être propagée, plusieurs techniques peuvent être retenues. Parmi celles-ci, deux techniques sont généralement utilisées: *la technique du journal* et *la technique du déclencheur*.

Dans le premier cas (*technique du journal*), un journal conserve l'historique des mises à jour. Les clients ou le serveur peuvent consulter le journal et être informés du besoin en propagation de ces mises à jour. L'avantage de cette technique est qu'elle est indépendante du processus de sauvegarde des mises à jour, le journal ne fait que répertorier le statut des connaissances sur le serveur. Si cette technique est utilisée dans un mode où ce sont les clients qui initient la propagation des mises à jour, cette technique permet aux clients de vérifier eux même l'état de leurs connaissances et de prendre des décisions en conséquence.

L'autre technique (technique du déclencheur) consiste à transmettre les nouvelles connaissances aux clients concernés dès qu'un événement se produit. Cet événement peut aussi bien être la mise à jour d'une connaissance, le dépassement d'un laps de temps, etc. Oracle et Ingres sont des exemples de système de gestion de base de données qui utilisent cette technique [PACI199]. Par exemple, dans une base de données Oracle, il est possible de régler la synchronisation des connaissances à toutes les heures. L'utilisation de cette technique est généralement plus facile et plus rapide à implémenter que la technique du journal. Le système de gestion de base de données ne requiert aucune modification; la propagation utilise seulement des événements qui sont inclus avec ces systèmes. Avec cette technique, l'événement peut être généré tant sur le serveur que sur le client. De plus, si on

utilise cette technique dans un mode où le serveur initie la propagation et qu'on choisit de mettre à jour les clients dès qu'une transaction a lieu, on conserve une fraicheur des connaissances près du temps réel.

La technique du serveur paresseux a été largement implantée dans les systèmes de gestion de base de données actuels. Le Tableau 6-1 présente globalement, pour des systèmes de gestion de base de données actuels, les techniques utilisées pour détecter les mises à jour ainsi que l'acteur qui initie la propagation des mises à jour [STA1995], [SMI1995], [BOB1996], [DAV1994a], [DAV1994b], [DAV1994c].

Systèmes	Détecter une mise à jour	Qui initie propagation de la mise à jour?
Oracle	Déclencheur	Client et Serveur
Sybase	Journal	Serveur
Informix	Journal	Serveur
Ingres	Déclencheur	Serveur
IBM	Journal	Client

Tableau 6-1 Technique de propagation de mises à jour utilisée en industrie.

6.4 La méthode hybride

6.4.1 Technique par vote

Enfin, dans cette section, nous présentons une méthode hybride entre les deux méthodes présentées précédemment. Cette technique permet aux clients de détecter une nouvelle mise à jour en consultant les autres clients. Dans la technique *par vote* [THO1979], [GIF1979], chaque connaissance possède un numéro de version. Lors d'une mise à jour sur le serveur, le numéro de version de cette connaissance augmente. Cette nouvelle connaissance et son numéro de version sont transmis à une majorité de clients. Si une erreur survient lors du rafraichissement des connaissances pour cette majorité de clients, cette modification est défaite et la base de connaissances redevient comme elle était avant la mise à jour.

Lorsqu'un client a besoin d'utiliser une connaissance, il compare le numéro de la version qu'il possède avec le numéro de version des autres clients. Si le client trouve un client qui possède un numéro de version supérieur au numéro de version qu'il possède, alors le client rafraichit cette connaissance. Ainsi, avant chaque utilisation d'une connaissance, les clients recherchent la version la plus à jour chez les autres clients. S'il ne possède pas la version la plus à jour, il met à jour celle-ci. De cette façon, la mise à jour se

fait petit à petit à travers l'ensemble des clients en fonction du besoin envers cette connaissance.

Cette technique a l'avantage de diminuer le trafic entre les clients et le serveur en ayant toujours les connaissances les plus à jour. De plus, en cas de panne, il est possible de reconstituer la base de connaissances à partir de l'ensemble des connaissances des clients en comparant les versions de mises à jour. Cependant, l'inconvénient de cette technique est qu'elle nécessite beaucoup de communication entre les clients. Les clients doivent valider leurs connaissances avec celles des autres. Donc, la lecture d'une information pour un client impose la lecture d'informations pour plusieurs autres clients. Cette technique de propagation des mises à jour est plus adaptée pour une approche multiagent où la communication et la lecture sont peu couteuses, ou peu fréquentes.

6.5 Mise à jour des connaissances dans un système multiagent

Dans les systèmes multiagents, les agents communiquent entre eux à l'aide de messages plus ou moins sophistiqués, dans le but d'assurer la coopération, voire la coordination du groupe. Ils peuvent s'échanger de l'information sur l'environnement pour augmenter leurs perceptions individuelles ou se transmettent leurs intentions pour que les agents puissent avoir une idée de ce que tout à chacun fait ou a l'intention de faire. Dans ce genre de système, la connaissance est décentralisée, chaque agent possède sa propre connaissance. La propagation des mises à jour se fait donc entre les agents. Généralement,

deux méthodes sont utilisées pour transmettre les informations dans un système multiagent: soit les agents communiquent entre eux les nouvelles informations ou bien que chaque agent centralise les nouvelles informations vers un agent déterminé et celui-ci se charge de transmettre ces nouvelles informations. Dans ce genre de système, la propagation des mises à jour se fait généralement de façon asynchrone. Les mises à jour deviennent effectives chez un agent et il doit informer les autres agents des nouvelles informations. Il n'est donc pas garanti que les connaissances que chaque agent possède soient valides.

Dans le cas où la communication est décentralisée, chaque agent communique avec les autres agents pour être informé des mises à jour. Chaque agent a donc le devoir d'informer les autres agents des mises à jour de sa base de connaissances. Donnons en exemple les essaims de robots [SWISI2006], [SWISR2006], [SSWARM2006]. Cette architecture répartit les connaissances à travers l'ensemble des participants, c'est-à-dire que chaque participant possède une partie de l'ensemble des connaissances. Dans cette architecture chaque participant interroge les autres participants pour avoir les connaissances nécessaires à leur tâche.

Dans le cas où la communication est centralisée, les agents transmettent les mises à jour à un client qui se charge de les gérer. Cette architecture revient à une architecture client-serveur en mode asynchrone. Par exemple, les *tableaux noirs* sont un exemple d'architecture centralisé pour la propagation des mises à jour. Un tableau noir est une mémoire qui permet de partager des connaissances à un ensemble d'agents dans un système multiagent. Un tableau noir est représenté par un agent qui assume le rôle de la mémoire

centrale du système multiagent. Il a été démontré que cette architecture évite la redondance des informations et améliore les performances lors du raisonnement des agents [CUTK1993], [GENE1994], [DECK1997] et [SYC2003].

6.6 Conclusion

Dans ce chapitre, nous avons présenté un aperçu général des techniques de propagation des mises à jour existantes dans les systèmes distribués actuels. Nous avons vu les avantages et les inconvénients de chacun ainsi que les applications auxquelles ces techniques sont destinées.

Le modèle de la mémoire collective que nous proposons dans ce mémoire n'est pas tout à fait adapté pour les techniques existantes de propagation des mises à jour. Nous visons des applications qui se déroulent en temps réel dans un contexte évolutif où la bande passante est limitée. L'implantation d'une mémoire collective dans un système distribué intelligent doit permettre autant aux clients qu'au serveur de gérer efficacement la mise à jour des connaissances dans un environnement temps-réel tout en minimisant le trafic généré. Nous avons donc besoin de performance tout en minimisant l'incohérence momentanée des connaissances avec une contrainte d'accès au réseau.

Ainsi, nous avons opté pour une approche asynchrone qui minimise le trafic sur le réseau. Plutôt que d'utiliser le journal, nous avons utilisé une technique qui s'apparente au

déclencheur, mais dont l'événement est géré par le client basé sur des valeurs statistiques. De cette façon, le traitement généré est divisé entre le serveur et le client.

Chapitre 7 Conclusion

Au terme de ce mémoire, nous avons présenté une approche originale pour modéliser une mémoire collective basée sur la problématique définie au Chapitre 1. Il convient donc de revenir sommairement sur ce qui a été présenté, de valider l'atteinte des objectifs et de voir les travaux qui pourront découler de ce mémoire.

Au Chapitre 1, nous avions formulé la problématique qui consistait à minimiser l'utilisation de la bande passante d'un système intelligent évolutif qui utilise une architecture client-serveur tout en gérant l'incohérence momentanée des connaissances. Pour résoudre cette problématique, nous avions déterminé, au Chapitre 1, trois éléments sur lesquels nous baserions nos recherches pour la réalisation de ce mémoire, soit :

1. *L'accès à la connaissance* dans une architecture où les connaissances sont centralisées sur un serveur. La mémoire dite collective doit être disponible simultanément à l'ensemble des clients sans que le serveur n'ait besoin de tenir compte de l'espace disponible dans toutes les composantes pour stocker les informations.
2. *Le mode de représentation des connaissances* qui soit suffisamment riche pour améliorer la qualité des réponses du serveur aux clients de façon à minimiser le nombre de requêtes invalides

3. *La mise à jour des connaissances* entre les clients et le serveur. La mémoire collective doit prendre en compte que les connaissances qu'elle possède peuvent ne pas être stables. Le modèle doit prévoir un moyen de mettre à jour les connaissances dans un environnement temps-réel de façon à minimiser le trafic généré.

Au cours du Chapitre 2, nous avons présenté le mode de représentation des connaissances utilisé ainsi que les adaptations que nous avons apportées à celui-ci pour mieux répondre à nos besoins. Dans ce chapitre, nous avons présenté le paradigme des graphes conceptuels décomposés en un ensemble d'éléments, que nous avons appelé *ensemble de briques*. De cette façon, nous avons réussi à utiliser un mode de représentation des connaissances qui soit très riche et à créer un index sur ses structures. Dans le chapitre suivant (Chapitre 3), nous avons présenté une technique de recherche à travers les briques qui ont découlé des graphes conceptuels. Cette technique permettait de faire le lien entre la requête d'un client et les éléments dans la base de connaissance. De cette façon, seules les connaissances nécessaires sont transmises aux clients. On minimise ainsi la transmission d'éléments sur le réseau, ce qui répond à notre objectif de représentation des connaissances et d'accès à la connaissance.

Dans le Chapitre 3, nous avons présenté un algorithme de recherche ainsi qu'un comparatif avec un système de recherche des graphes conceptuels existant. Nous avons montré que notre méthode de recherche est meilleure selon les critères de recherche définie

dans ce chapitre. Cependant, comme il a été présenté dans le chapitre 3, nous n'avons pas réussi à créer un algorithme de recherche qui soit d'ordre polynomial.

Enfin, dans le Chapitre 4 et le Chapitre 5, nous avons présenté une technique qui permet de maintenir à jour les connaissances des clients tout en minimisant le transfert de connaissances entre le serveur et les clients. Cette technique, basée sur une analyse probabiliste de la péremption des valeurs de vérité des connaissances, permet de déterminer les moments où les connaissances subiront une altération de leurs valeurs de vérité. Ainsi, on modélise la fluctuation des valeurs de vérité et on rafraîchit les connaissances des clients, lorsque jugé nécessaire. Évidemment, le bon fonctionnement de cette technique repose sur un calibrage adéquat des paramètres qui représentent le phénomène de mise à jour des valeurs de vérité associées aux connaissances de la base de connaissance.

7.1 Retour sur les objectifs

En dépit des différentes limitations, les techniques que nous avons élaborées nous ont permis d'atteindre les objectifs que nous avions fixés à la section 1.3, à savoir :

1. Définir un mode de représentation des connaissances qui soit expressif pour permettre au processus de recherche de gagner en efficacité et en précision, pour

ainsi réduire le nombre de réponses inutiles et, par le fait même, l'utilisation de la bande passante.

2. Définir une structure d'accès qui permet un accès simultané à la connaissance par l'ensemble des clients, sans engorger le serveur. Toute requête d'un client ne doit pas monopoliser le serveur.
3. Définir un mécanisme de gestion des mises à jour des valeurs de vérité associées aux connaissances. Ce mécanisme doit permettre de faire une mise à jour continuelle, mais selon les besoins de l'application, de ces valeurs de vérité étant donné que certaines applications visées se déroulent en temps réel dans un contexte évolutif.

Comme nous venons d'en discuter dans la section précédente, nous avons construit un modèle capable de distribuer les connaissances à l'ensemble des clients d'un système en minimisant les transferts d'informations et en contrôlant le processus de mise à jour des connaissances. Ainsi, nous croyons que le modèle proposé permet d'atteindre nos objectifs.

Dans ce mémoire, nous avons apporté une solution originale pour minimiser l'utilisation de la bande passante d'un système intelligent évolutif qui utilise une architecture client-serveur. Nous avons apportés une contribution plus particulière à deux éléments de la problématique, soit la représentation des connaissances et la gestion des mises à jour. D'une part, nous proposons une technique de modélisation des graphes conceptuels qui permet une recherche efficace des connaissances en minimisant les

résultats d'une requête. D'autre part, nous avons présenté une technique originale pour préserver la cohérence des connaissances appliquées aux modèles des graphes conceptuels.

7.2 Travaux futurs

Parvenus au stade final de notre recherche, nous avons identifié les avenues de recherche suivantes pour des travaux ultérieurs :

1. évaluer empiriquement les performances du mécanisme de recherche et de gestion des mises à jour que nous avons proposé;
2. améliorer la représentation conceptuelle des connaissances en prenant en compte des phénomènes comme les liens de coréférence, des graphes inclus dans des graphes, etc.;
3. établir un cadre général pour déterminer les valeurs initiales que doivent prendre les paramètres d'évaluation des mises à jour;
4. établir un protocole de communication entre le serveur et les clients qui soit adapté à notre problématique;
5. améliorer l'utilisation de la bande passante utilisée dans la mémoire collective.

Bibliographie

[BDFL1988] A. Berard-Dugourd, J. Fargues, and M.C. Landau. *Natural language analysis using conceptual graphs*. In Proceedings of the International Computer Science Conference, ICSC'88, 1988.

[BDRIEU2001] Benjamin Drieu. *L'intelligence artificielle distribuée appliquée aux jeux d'équipe situés dans un milieu dynamique : l'exemple de la Robocup*. Mémoire de maîtrise, Université Paris 8, 2001.

[BHG1987] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[BOB1996] S. Bobrowski. *Oracle 7 server concepts, release 7.3*. Oracle Corporation,Redwood City, CA, 1996.

[BZ1992] J. Bouaud and P. Zweigenbaum. *A reconstruction of Conceptual Graphs on top of a production system*. In Proceedings of the 7th annual workshop on Conceptual Graphs, 1992.

[CAR1996] B. Carbonneill. *Vers un système de représentation de connaissances et de raisonnement fondé sur les graphes conceptuels*. PhD thesis, Janvier 1996.

[CHKS1994] S. Ceri, M.A.W. Houtsma, A.M. Keller, and P. Samarati. *A classification of update methods for replicated databases*. Technical report, Standford University, 1994.

[CHKS1995] S. Ceri, M.A.W. Houstma, A.M. Keller, and P. Samarati. *Independent updates and incremental agreement in replicated databases*. Distributed and Parallel Databases, 3(3):225-246, Juillet 1995.

[CM1992] M. Chein et M.L. Mugnier. *Conceptual graphs: fundamental notions*. Revue d'Intelligence Artificielle, 6(4):365-406, 1992.

[CM2001] Robert Côté et Hervé Morin. *Méthodes statistiques*. Université Laval, Septembre 2001.

[COGIT2007] COGITANT 5.1.8 . "Bibliothèque de manipulation de graphes conceptuels". Dans en.SourceForge.net, [En ligne]. http://cogitant.sourceforge.net/cogitant_html/index.html (Page consultée le 8 avril 2007)

[CUTK1993] M. R. Cutkosky, R. S. Engelmore, R. E. Fikes, M. R. Genesereth, T. R. Gruber, W. S. Mark, J. M. Tenenbaum, and J. C. Weber. *PACT: An experiment in integrating concurrent engineering systems*. Computer, 26(1):28–38, Janvier 1993.

[DAV1994a] J. Davis. *Data replication. Distributed Computing Monitor*, 1994.

[DAV1994b] J. Davis. *Oracle delivers. Open Information Systems*, Juilllet1994.

[DAV1994c] J. Davis. *Sybase system 10. Open Information Systems*, Mars 1994.

[DCOR2003] Daniel D Corkill, *Collaborating Software: Blackboard and Multi-Agent Systems & the Future*, Proceedings of the International Lisp Conference, Octobre 2003.

[DECK1997] K. Decker, K. Sycara, and M. Williamson. *Middleagents for the Internet*. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan, Août. 1997.

[ELL1993] G. Ellis. *Efficient retrieval from hierarchies of objects using lattice operations*. In Proceedings of the 1st International Conference on Conceptual Structures, volume 699 of Lecture Notes in Artificial Intelligence, pages 274–293, 1993.

[ELL1994] G. Ellis and F. Lehmann. *Exploiting the induced order on type labeled graphs for fast knowledge retrieval*. In Proceedings of the 2nd International Conference on Conceptual Structures, volume 835,1994.

[FAR1989] J. Fargues. *CG information retrieval using linear resolution, generalization and splitting*. In Proceedings of the 4th annual workshop on conceptual structures, 1989.

[FUH1992] N. Fuhr. *Probabilistic models in information retrieval*. The Computer Journal, 35(3):243-255, 1992.

[GAI1993] B.R. Gaines. *Representation, discourse, logic and truth : situating knowledge technology*. In G. Mineau, W. Moulin, and J.F. Sowa, editors, Proceedings of the third International Conference on Conceptual Structures, ICCS'93, volume 699 of Lecture Notes in Artificial Intelligence, pages 36-63, Quebec City, Canada, Août 1992.

[GENE1994] M. R. Genesereth and S. P. Ketchpel. *Software agents*. Communications of the ACM, Special Issue on Intelligent Agents, 37(7):48–53, Juillet 1994.

[GIF1979] D.K. Giford. *Weighted voting for replicated data*. ACM-SIGOPS Symp. on Operating Systems Principles, Paci c Grove, December 1979.

[GRA1979] J. Gray. *Notes on database operating systems*. Lecture Notes in Computer Science, 1979.

[GUI1996] O. Guinaldo. *Conceptual graphs isomorphism: Algorithm and use*. In P.W. Eklund, G. Ellis, and G. Mann, editors, Proceedings of the 4th International Conference on Conceptual Structures, ICCS'96, volume 1115 of Lecture Notes in Artificial Intelligence, pages 160-174, Sydney, Août 1996.

[GUI1998] Ollivier Guinaldo, Olivier et Haemmerlé. *Knowledge querying in the conceptual graph model : The rap module*. In M. Chein (Eds.) M.-L. Mugnier, editor, Conceptual Structures : Theory, Tools and Applications : 6th International Conference on Conceptual Structures, volume 1453, page p. 287, Montpellier, Août 1998. Springer-Verlag GmbH.

[HM1990] W.W. Hines and D.C. Montgomery. *Probability and Statistics in Engineering and Management Science*. John Wiley and Sons, inc, 1990.

[HT1988] V. Hadzilacos and S. Toueg. *A theory of reliability in database systems*. Journal of ACM, 35(1), Janvier 1988.

[LAD1990] R. Ladin. *Lazy replication: Exploiting the semantics of distributed services*. Int. Workshop on Management of Replicated Data, Houston, November 1990.

[LB1985] H.J. Levesque and R.J. Brachman. *A fundamental tradeoffs in knowledge representation and reasoning*, pages 42-70. Morgan Kaufmann Publishers, Los Atlos, California, 1985.

[LB1987] H.J. Levesque and R.J. Brachman. *Expressiveness and tractability in knowledge representation and reasoning*. Computational Intelligence, 3(2):78-93, Mai 1987.

[LEV1994] R. Levinson. *UDS: A Universal Data Structure*. In Proceedings of the 2nd International Conference on Conceptual Structures, volume 835,1994.

[MASA1989] Masao Mukaidono, Zuliang Shen, and Liya Ding. *Fundamentals of fuzzy Prolog*. International Journal of Approximate Reasoning, 3(2) :179-193, 1989.

[MC1993b] M.L. Mugnier and M. Chein. *Characterization and algorithmic recognition of canonical conceptual graphs*. In Mineau et al., pages 294-311.

[MC1993a] M. L. Mugnier and M. Chein. *Polynomial algorithms for projection and matching*. In Pfeifer and Nagle, pages 239-251.

[MEC1995] M. Mechkour. EMIR2. An extended model for image representation and retrieval. Dans DEXA'95. Database and EXpert system Applications, London., pages 395-404, Septembre 1995.

[MEN1994] Consortium MENELAS. *Menelas: accès à des comptes-rendus d'hospitalisation en langage naturel*. In Actes des 5èmes journées francophones d'Informatique Médicale, 1994.

[MIN2000] Guy W. Mineau. *The extensional semantics of the conceptual graph formalism*. In ICCS '00 : Proceedings of the Linguistic on Conceptual Structures, pages 221-234, 2000.

[MKL1994] S.H. Myaeng, C. Khoo, and M. Li. *Linguistic processing of text for a large-scale conceptual information retrieval system*. In Tepfenhart et al. [TDS94], pages 69-82, 1994.

[ML1993] S.H. Myaeng and E. Liddy. *Information retrieval with semantic representation of texts*. In Proceedings of Symposium on Document Analysis and Information Retrieval, Las Vegas, USA, Avril 1993.

[MUG1993a] M. L. Mugnier and M. Chein. *Polynomial algorithms for projection and matching*. Lecture Notes in Computer Science, 754 :239-??,1993.

[MUG1993b] Marie-Laure Mugnier. *Contributions Algorithmiques pour les Graphes d'Héritage et les Graphes Conceptuels*. PhD thesis, Université Montpellier II, Octobre 1993.

[MYA1992a] S.H. Myaeng. *Conceptual graphs as a framework for text retrieval*. In T.E. Nagle, J.A. Nagle, L.L. Gerholz, and P.W. Eklund, editors, Conceptual Structures: current Research and Practice, England, Ellis Horwood Workshops, 1992.

[MYA1992b] S.H. Myaeng. *Using conceptual graphs for information retrieval: A framework for adequate representation and flexible inferencing*. In Proc. of Symposium on Document Analysis and Information Retrieval, 1992.

[NOG1990] Nogier, J. F. *Un système de production de langage fonde sur le modèle des graphes conceptuels*. Ph.D. thesis, Universit~ de Paris VII, 1990.

[OUN1998] Iadh Ounis, *Un modèle d'indexation relationnel pour les graphes conceptuels fondé sur une interprétation logique*, thèse de doctorat, Université Joseph Fourier, 1998.

[OUN2000] Iadh Ounis. *Organizing conceptual graphs for fast knowledge retrieval*, Octobre 2000.

[PACI199] Esther PACITTI, *Improving Data Freshness in Replicated Databases*, Research repport, Unit of reasearch INRIA Rocquencourt, Février 1999.

[PC1995] Jonathan Poole and J. A. Campbell. *A Novel Algorithm for Matching Conceptual and Related Graphs*. In Gerard Ellis, Robert Levinson, William Rich, and John

Sowa, editors, LNAI 954, Conceptual Structures : Applications, Implementation and Theory, pages 293-307. Springer-Verlag, Berlin, 14-18, Proceedings of the 3rd Int. Conference on Conceptual Structures, (ICCS'95), Santa Cruz, CA, USA, August 1995.

[PFV2005] Fournier-Viger, Philippe, *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. 2005.

[REWISI2006] Wikipedia. « Moteur de recherche ». Dans en.Wikipedia.org, [En ligne]. http://fr.wikipedia.org/wiki/Moteur_de_recherche (Page consultée le 19 février 2006)

[ROB1977] S.E. Robertson. *The probability ranking principle in IR*. Journal of Documentation, 33:294-304, 1977.

[SKORN2004] S. Kornienko, O. Kornienko, P. Levi. *About nature of emergent behavior in micro-systems*. In Proc. of the Int. Conference on informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal, 2004.

[SKORN2005] S. Kornienko, O. Kornienko, P. Levi.: *Collective AI: Context-Awareness via Communication*. In: Proceedings of the 19th European Conference on Artificial Intelligence (IJCAI), 2005.

[SM1983] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* Mcgraw Hill Book Company, New York, 1983.

[SMI1995] G. Smith. *Oracle7 Symmetric Replication*. Oracle Corporation, 1995.

[SOUT1999] Southey, F. and Linders, J. *Notio - A Java API for Developing CG Tools*. In and W. R. Cyre, W. M. T., editor, Conceptual Structures : Standards and Practices, 7th International Conference on Conceptual Structures Proceedings, volume 1640 of Lecture Notes in Computer Science, pages 262.271. Springer. 1999.

[SOW1984] J. F. Sowa. *Conceptual Structures : Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[SOW1993] J.F. Sowa. *Relating diagrams to logic*. In First International Conference on Conceptual Structures, ICCS'93, volume 699 of Lecture Notes in Artificial Intelligence, Subseries of Lecture Notes in Computer Science, Montreal, Canada, Springer-Verlag., pages 1-35, Août 1992.

[SROBO2006] RoboCup. « RoboCup ». [En ligne]. http://www.robocup.org (Page consultée le 06 janvier 2006)

[SSWARM2006] Swarm-bots. « Swarm-bots ». [En ligne]. http://www.swarm-bots.org (Page consultée le 06 janvier 2006)

[STA1995] D. Stacey. *Replication: Db2, Oracle, or Sybase ?* ACM Sigmod Record, 24(4), 1995.

[SWISI2006] Wikipedia. "Swarm Intelligence". Dans en.Wikipedia.org, [En ligne]. http://en.wikipedia.org/wiki/Swarm_intelligence (Page consultée le 06 janvier 2006)

[SWISR2006] Wikipedia. "Swarm Robotics". Dans en.Wikipedia.org, [En ligne]. http://en.wikipedia.org/wiki/Swarm_robotics (Page consultée le 06 janvier 2006)

[SYC2003] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. *The RETSINA MAS infrastructure*. Special Joint Issue of Autonomous Agents and MAS, 7(1 and 2), Juillet 2003.

[THO1979] R.H. Thomas. *A majority consensus approach to concurrency control for multiple copy databases*. ACM Transactions on Database Systems, 4(2), Juin 1979.

[VH2001A] V. Haarslev and R. Möller. Description of the RACER System and its Applications. In Proceedings International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August, pages 131–141, 2001.

[VH2001B] V. Haarslev and R. Möller. RACER System Description. In R. Goré, A. Leitsch, and T. Nipkow, editors, International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy, pages 701–705. Springer-Verlag, 2001.

[VR1979] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[W3C2007] W3C Semantic Web Activitity. Dans en.w3.org, [En ligne]. http://www.w3.org/2001/sw/ (Page consultée le 25 février 2007)

[ZA1993] P. Zweigenbaum and al. *Linguistic and medical knowledge bases: An access system for medical records using natural langage*. Technical report, MENELAS: deliverable 9, AIM Project A2023, 1993.

[ZWEIL1994] Zweigenbaum P et Consortium MENELAS. *MENELAS: an access system for medical records using natural language*. Comput Methods Programs Biomed, 1994.

Annexe A PRÉSENTATION DU MODE REPRÉSENTATION DES CONNAISSANCES

Introduction

Un mode de représentation des connaissances dans un ordinateur est un système formel qui vise la codification de la connaissance humaine à des fins de traitement automatique. Habituellement, dans un mode de représentation des connaissances, l'accès aux connaissances est dirigé par la sémantique des connaissances : l'utilisateur exprime ses besoins en informations en indiquant le contenu de la connaissance qu'il recherche. Pour ce faire, les systèmes sont fondés sur un modèle formel de correspondance entre une requête et un mode de représentation des connaissances. La Figure 7-1 est une adaptation d'un classique repris dans de nombreux textes traitant de la recherche d'informations et de la codification de l'information [SM1983] et [VR1979]. Dans la Figure 7-1, trois défis sont présentés : la représentation des connaissances, l'indexation des connaissances et la mise en correspondance des connaissances. Au cours de cette annexe, nous allons définir les éléments clés présentés sur la Figure 7-1 en rapport avec la problématique présentée dans le Chapitre 1.

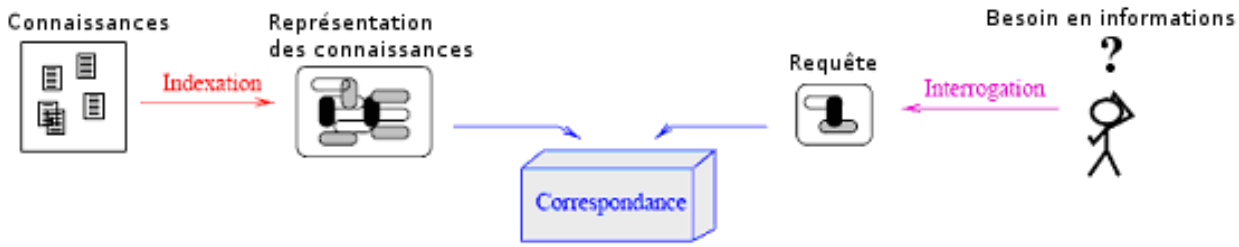

Figure 7-1 Représentation des connaissances.

Le modèle de la mémoire collective que nous proposons doit utiliser un mode de représentation des connaissances qui puisse représenter un volume important d'informations dans un environnement distribué. Dans ce contexte, la technique de mise en correspondance doit donc offrir une bonne rapidité d'exécution.

Cependant, si la recherche n'offre pas des réponses de bonne qualité, il n'est pas envisageable d'utiliser ce mode de représentation des connaissances, indépendamment de la rapidité de la recherche. Si les réponses ne sont pas de qualité, la tâche du client devient très ardue puisqu'il doit retrouver l'information qui l'intéresse dans une masse trop importante d'informations non pertinentes. Dans de telles circonstances, les accès aux réseaux sont multipliés inutilement par l'envoi de requêtes de plus en plus précises. De surcroît, plus le volume d'informations est élevé et plus la qualité est importante car le nombre de réponse inapproprié risque d'augmenter. L'accroissement du volume d'informations impose donc que la conception de tels systèmes soit orientée vers la précision des réponses. Nous allons donc devoir concilier rapidité de traitement et précision des réponses. Pour arriver à ce but, nous allons utiliser un mode de représentation des connaissances qui soit très expressif. De cette façon, la mise en correspondance pourra

correctement interpréter la sémantique de la requête de l'utilisateur et ainsi améliorer la précision des réponses.

La technique de mise en correspondance idéale permettrait un compromis entre la richesse d'expression et l'efficacité des traitements. Plusieurs travaux théoriques [LB1985], [LB1987] ont abouti à la conclusion qu'il existait une opposition entre richesse d'expression et efficacité des calculs. C'est le cas par exemple des logiques terminologiques (logique descriptive) SHIQ [PFV2005], [VH2001A], [VH2001B], connues pour leur sémantique clairement définie et leur puissance d'expression, mais dont la complexité des inférences constitue un obstacle pour toute utilisation efficace. À l'inverse, les systèmes basés sur les modèles classiques les plus connus et qui sont réputés pour leur rapidité comme le modèle Booléen, le modèle vectoriel [SAL1971], [SM1983], ou le modèle probabiliste [VR1979], [ROB1977], [FUH1992], semblent atteindre leur potentiel maximum en termes de qualité des réponses. Notre choix d'un mode de représentation des connaissances est confronté à un double défi. D'une part, l'utilisation d'un mode de représentation des connaissances expressif s'avère nécessaire pour l'amélioration de la précision des réponses aux requêtes. D'autre part, le coût d'évaluation des requêtes devra être raisonnable, c'est-à-dire que les fonctions de correspondance utilisées devront se baser sur des algorithmes polynomiaux.

Après avoir regardé plusieurs modes de représentations des connaissances, nous avons retenu le formalisme des graphes conceptuels de Sowa [SOW1984]. Les raisons de ce choix trouvent leurs origines aussi bien dans des considérations théoriques que dans un but pragmatique. Dans la section 0, nous discutons des éléments intéressants qui justifient

le choix de ce formalisme. La section 0 se veut une introduction rapide du formalisme des graphes conceptuels. Dans cette section, nous présentons les notions de base de ce formalisme telles qu'elles sont présentées dans [SOW1984]. Dans la section 0, nous verrons quelles méthodes ont été utilisées par le passé pour indexer les graphes conceptuels dans le cadre de projets en recherche d'informations. Ensuite, dans la section 0, nous présentons une technique d'implémentation de ce formalisme comme un langage d'indexation relationnelle. Nous verrons comment il est possible d'implanter un système de graphes conceptuels en utilisant le paradigme des bases de données relationnelles. Il s'agit d'étendre le modèle de base à l'aide de notions dont nous avons besoin pour qu'une manipulation de ces graphes, selon une approche basée sur l'algèbre relationnelle, puisse être possible [MIN2000]. L'implantation d'une approche relationnelle des graphes conceptuels permet de manipuler des bases de connaissances très volumineuses avec une technologie éprouvée. Nous verrons également, dans la section 0, comment calculer l'extension sémantique d'un graphe conceptuel, c'est-à-dire, nous verrons comment évaluer la valeur d'une requête réalisée par un client.

Pourquoi les graphes conceptuels?

Dans son livre sur les structures conceptuelles [SOW1984], Sowa introduit, à travers un formalisme logique convivial, un nouveau modèle de représentation de connaissances. Ce formalisme, appelé *graphes conceptuels*, fut présenté comme un

compromis entre les langages formels et les langues naturelles. Les fondements logiques des graphes conceptuels reposent sur la définition de l'opérateur Φ (défini dans la prochaine section), qui associe à tout graphe conceptuel une formule logique de premier ordre, et la définition d'un certain nombre de règles de réécriture. L'aspect graphique du formalisme procure une facilité de lecture fort appréciable en représentation de connaissances, permettant ainsi la construction d'interfaces personne-machine utiles dans tout processus interactif.

Un des atouts de ce formalisme est justement le fait qu'il constitue une représentation graphique de la logique, s'inspirant ainsi de l'idée défendue par C.S. Peirce [SOW1993], à savoir qu'un graphe est plus facilement déchiffré qu'une formule logique. À travers les opérateurs algébriques de graphes qu'il offre, le formalisme des graphes conceptuel simule un raisonnement logique. L'intérêt d'une telle approche est que ces opérateurs algébriques peuvent bénéficier des multiples algorithmes efficaces développés en théorie des graphes, autorisant ainsi des implantations rapides. Les graphes conceptuels offrent donc un bon compromis entre les approches algébriques et les démonstrateurs de preuves ; plutôt que de considérer les graphes conceptuels comme une logique et réaliser les calculs à l'aide d'un démonstrateur logique, l'approche adoptée consiste à exécuter exclusivement des opérations algébriques sur les graphes et à leur associer une interprétation logique par le biais de l'opérateur Φ.

De plus, comme cela a été mentionné par Gaines [GAI1993], les graphes conceptuels peuvent servir de base à un système opérationnel de représentation de

connaissances pouvant manier aussi bien la langue naturelle, la logique et les langages formels que les raisonnements qui leur sont associés. Le formalisme des graphes conceptuels est assez expressif pour représenter et raisonner sur des connaissances complexes et structurées. Ainsi, il est possible d'utiliser ce formalisme pour une large gamme de domaines d'application. De plus, ce formalisme permet une représentation précise des connaissances améliorant ainsi l'efficacité de l'opérateur de mise en correspondance.

Avec tous ces arguments, notre choix de mode de représentation des connaissances s'est arrêté sur le formalisme des graphes conceptuels. Le formalisme des graphes conceptuels est donc un mode de représentation des connaissances très intéressant pour réaliser nos premiers objectifs en rapport avec le mode de représentation des connaissances. C'est-à-dire, nous cherchions un mode de représentation qui :

1. sois expressif (pour maximiser la qualité des réponses aux requêtes);
2. minimise les accès aux réseaux (minimise le nombre de réponses lors d'une requête);
3. facilite l'implantation de la mémoire collective;
4. soit adaptable à de nouvelles situations (ceci sera présenté dans la section 0).

Au cours des prochaines sections, nous allons voir les notions de base pour l'implantation d'un système de graphes conceptuels.

Notions de base

Les graphes conceptuels sont présentés comme un modèle général d'écriture de réseaux sémantiques pour la représentation des connaissances. 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 cœur 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 personne-machine. La représentation des connaissances se fait à travers la description de concepts et des relations entre ces concepts.

Tout d'abord, nous définissons un graphe conceptuel comme étant un graphe étiqueté fini, orienté et biparti, non nécessairement connexe[9]. Les deux types de nœuds composant le graphe conceptuel sont les nœuds *concepts* et les nœuds *relations*. Les nœuds concepts représentent les entités, les attributs, les états et les événements alors que les nœuds relations représentent les liens qui unissent ces concepts. Les étiquettes des nœuds concepts sont composées de deux parties : le *type* et le *référent*. Les étiquettes des nœuds relations, quant à elle, possèdent seulement un type. Le type représente la classe à laquelle se rattache le concept (relation) exprimé par rapport à un domaine d'application, tandis que le référent est vu comme une instanciation du type de concept. L'ensemble des référents

[9] Historiquement, les graphes conceptuels étaient défini comme connexes [Sowa1984].

conformes à un type de concept forme l'extension de ce dernier, ou sa *dénotation*. Les relations sont rattachées à un nombre *n* de concepts où *n* représente l'arité de la relation. Pour s'assurer que les concepts rattachés à la relation sont bien conformes à la syntaxe de la relation, une *signature de relation* est définie. Une signature de relation de type *t* et d'arité *j* est un tuple de *j* concepts. Chaque élément *i* du tuple représente le type de concept auquel le $i^{ème}$ concept rattaché à la relation doit se conformer. Un exemple de graphe conceptuel est donné dans la Figure 7-2 suivante :

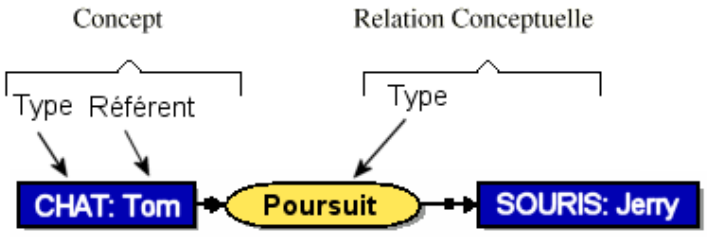

Figure 7-2 Un graphe qui représente : " Tom poursuit Jerry ".

Les types de concepts sont organisés selon une hiérarchie des types qui constitue un treillis fini appelé T_c [Sowa1984]. Ce treillis est muni de la relation d'ordre partiel ≤, définie de la façon suivante: si la dénotation d'un type de concept c_1 est incluse dans la dénotation du type de concept c_2 alors $c_1 \leq c_2$ dans T_c. Prenons par exemple, Homme ≤ Personne. Nous disons que Homme est une restriction (sous-type) de Personne, et que Personne est une généralisation de Homme. Les types de relations dénotés par l'ensemble T_r sont aussi classés dans une structure de treillis. Dans ce cas, ce sont les signatures des relations qui sont organisées dans T_r. Par exemple, la Figure 7-3 présente un treillis (incomplet) des types. L'encerclé en rouge représente les types de concepts alors que l'encerclé en bleu

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

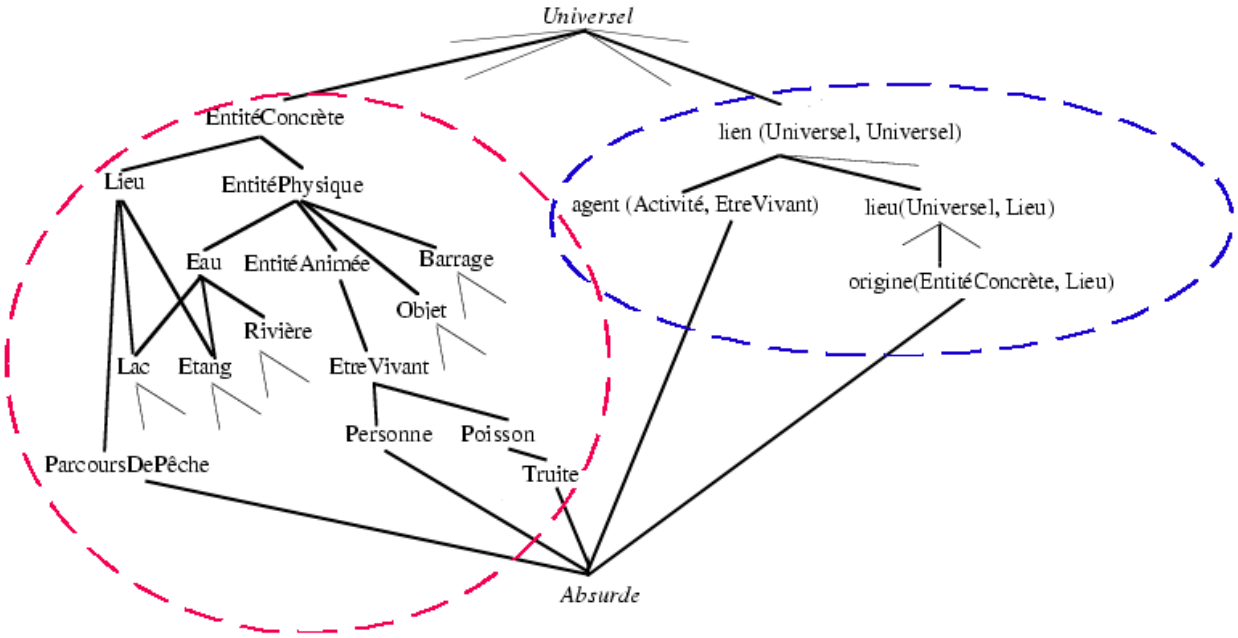

Figure 7-3 Le treillis de type T.

Ces éléments de base étant posés, Sowa introduit la notion de graphe simple:

> **Définition 7-1** (*Graphe Simple*) Un graphe simple est formé d'un ensemble de concepts et un ensemble de relations tel que chaque relation est relié à un ensemble de concepts conforme à sa signature et à son arité. Chaque concept possède des référents qui appartiennent au type du concept. Un concept seul est aussi considéré comme un graphe simple.

Le formalisme des graphes conceptuels ne définit pas uniquement des structures de données, elle définit également un certain nombre d'opérations permettant de les manipuler. Toutes les opérations valides pouvant être effectuées sur les graphes conceptuels sont une combinaison des six règles de base suivantes (appelées règles canoniques) :

- les règles d'équivalence : la copie et la simplification ;
- les règles de spécialisation : la restriction et la jointure ;
- les règles de généralisation : l'élargissement et la fragmentation.

Ces règles de base forment trois paires de transformations réciproques. La Figure 7-4 fournit un exemple de la copie et la simplification qui permettent respectivement de créer et d'éliminer des sous-graphes redondants. La Figure 7-5 illustre le fonctionnement de la restriction et de l'élargissement qui consiste à spécialiser ou généraliser un nœud du graphe conceptuel. Les relations ne peuvent être spécialisées qu'en modifiant leur type tandis que les concepts peuvent de surcroît être spécialisés en modifiant leur référent ; par exemple, en spécifiant un référent particulier comme c'est le cas dans la Figure 7-5 pour le concept [CHAT].

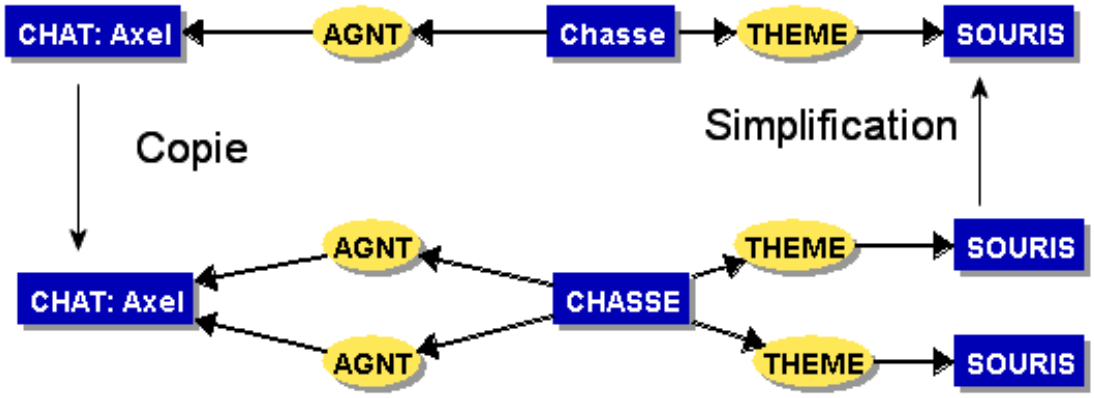

Figure 7-4 La copie et la simplification.

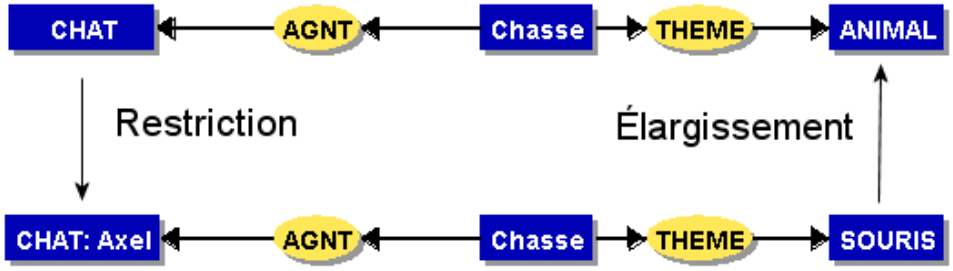

Figure 7-5 Restriction et élargissement.

La Figure 7-6 représente l'opération de jointure et sa transformation réciproque : la séparation. La jointure consiste simplement à assembler deux graphes en fusionnant certains de leurs nœuds. Cette transformation permet, par exemple, d'intégrer dans une seule structure plusieurs sources de connaissance traitant de concepts communs. À partir de ces règles de base, il est possible de créer des opérations plus complexes comme la jointure maximale qui joint chacun des concepts communs de deux graphes. Cette dernière sera formalisée plus tard dans ce chapitre.

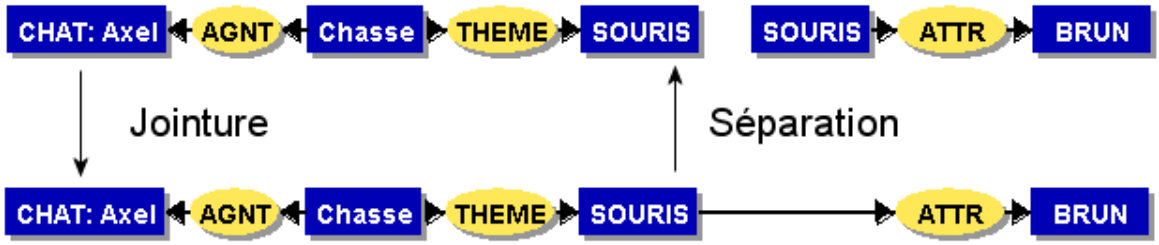

Figure 7-6 Jointure et séparation.

Seuls les graphes construits à partir de ces quatre opérateurs sont des expressions du langage des graphes conceptuels. Sowa introduit alors un ordre partiel ≤ sur les graphes (relation à ne pas confondre avec la relation d'ordre sur les types). Cet ordre définit une relation d'inférence sur les graphes permettant de déterminer si les informations représentées par un graphe induisent les informations représentées 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. Cette relation peut aussi être définie par l'opérateur de projection, considérée comme un morphisme de graphes bipartis (les nœuds concepts sur les nœuds concepts, les nœuds relations sur les nœuds relations):

Définition 7-2 (*Projection*) Une application π des nœuds d'un graphe *h* vers les nœuds d'un graphe *g* est appelé *projection* si :

- pour chaque concept *c* de *h*, π(*c*) est soit une spécialisation de *c*, soit identique à *c*.

- pour chaque relation *r* de *h*, π (*r*) 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. La Figure 7-7 montre un exemple de projection.

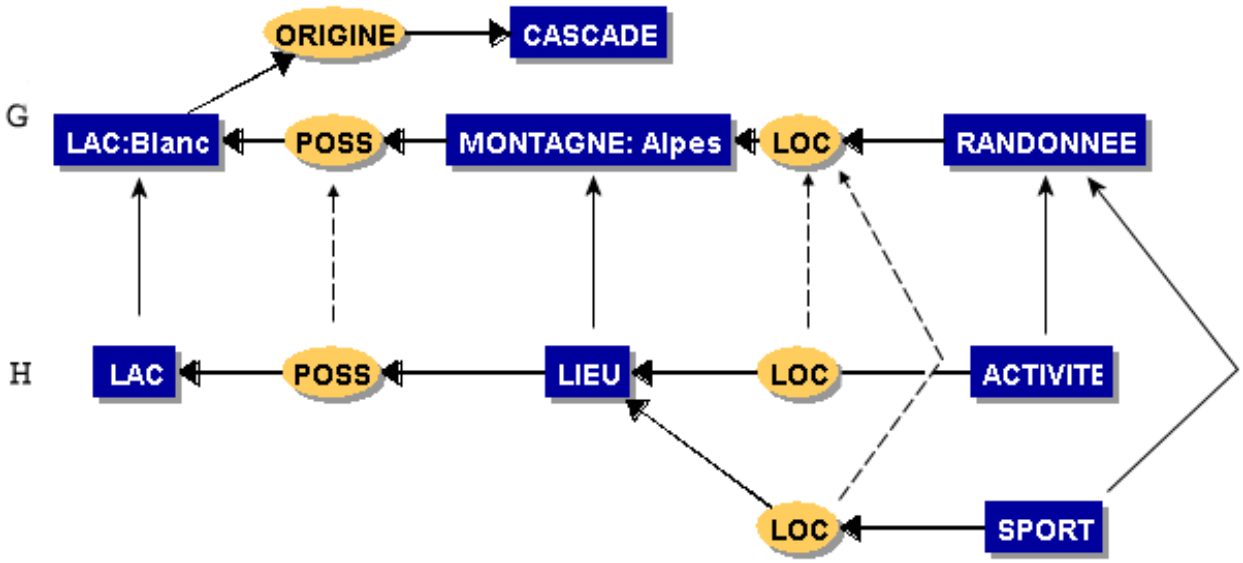

Figure 7-7 La projection de *h* sur *g*. *h* peut être interprétée comme suit: « il existe une activité et un sport localisés sur le même lieu possédant un lac ».

On dit également que *g* est une spécialisation de *h* ou que *h* est une généralisation de *g* s'il existe une projection de un vers l'autre. Cavalièrement, nous pouvons dire qu'un graphe *h* est une spécialisation d'un graphe *g* s'il existe au moins autant d'information dans *h* que dans *g*.

Sowa prouve que si un graphe *g* est une spécialisation d'un graphe *h*, alors il existe une projection de h sur *g*. De plus dans [CM92], il est démontré que s'il existe une projection de *h* sur *g*, alors $g \leq h$. Malgré de nombreux travaux ayant trait à l'étude de la complexité de la projection dans le formalisme des graphes conceptuels [MUG1993a], [MUG1993b], [PC1995], [GUI1996] et [GUI1998], la classe de complexité de cette dernière n'est pas encore totalement connue. Tout au plus, certains algorithmes polynomiaux ont été proposés pour des graphes conceptuels de type particulier (par exemple le cas où les graphes correspondent à des arbres [MUG1993a]). Cependant, à l'heure actuelle, il n'existe pas d'algorithme polynomial qui permet de vérifier l'existence d'une projection d'une façon générale sur n'importe quel type de graphe conceptuel.

Théorème 7-1 Soient *g* et *h* deux graphes conceptuels, *g* est une spécialisation de *h* (noté $g \leq h$) seulement s'il existe une projection π de h dans *g*.

Comme on le verra plus tard, cet opérateur est très important dans un processus de recherche fondé sur les graphes conceptuels. La projection est en fait la méthode de mise en correspondance entre la requête formulée par l'utilisateur et la base de connaissances. L'existence d'une projection d'un graphe vers un autre signifie que ce graphe est l'une des réponses à la question formulée par l'utilisateur. En résumé, cet opérateur est la technique de mise en correspondance du mode de représentation des connaissances.

Une caractéristique importante du modèle de Sowa est que les raisonnements présentés précédemment sur les graphes peuvent être faits tout en conservant un lien avec la logique de premier ordre. Sowa définit un opérateur d'interprétation logique, noté Φ, qui transforme chaque élément du modèle des graphes conceptuels simples en un élément de la logique du premier ordre. L'opérateur Φ associe à tout type de concept un prédicat unaire et à tout type de relation, un prédicat de même arité que le type. À un marqueur individuel est associé une constante.

Définition 7-3 (*L'opérateur Φ*) L'association d'une formule logique Φ(u) à un graphe conceptuel u est réalisé de la façon suivante:

1. Pour chaque concept [c], on associe un prédicat $T_c(p)$ dont le nom est égal au type de concept T_c de [c], et dont le paramètre p est soit une constante dans le cas où [c] est muni d'un marqueur individuel, soit une variable indépendante et quantifiée si [c] est muni du marqueur générique. Par exemple, au concept [Personne : #Hugo] est associé le prédicat Personne(Hugo) alors qu'au concept générique [Personne : *x] est associé la formule logique ∃x Personne(x).

2. À chaque relation d'arité *j* est associé un prédicat ayant *j* paramètres dont le nom est égal au type de la relation T_r, et dont les paramètres sont égaux aux paramètres des prédicats correspondant aux concepts liés par cette relation. L'ordre des paramètres est convenu pour chaque type de relation.

3. Si u est un graphe conceptuel, Φ(u) est une formule logique de premier ordre obtenue par conjonction des prédicats associés aux composants de u dans 1) et 2).

Par exemple, au graphe

[CHAT : *Tom*] → (Poursuit) → [SOURIS]

est associé la formule logique suivante:

$$\exists x \;\; \text{CHAT}(\; \textit{Tom}\;) \wedge \text{Poursuit}(\; \text{Tom}; x\;) \wedge \text{SOURIS}(\; x\;)$$

Une propriété importante de l'opérateur Φ est qu'il vérifie le théorème suivant [CM1992]:

Théorème 7-2 Pour tout couple de graphes conceptuels u et v, si $u \leq v$, alors $\Phi(u) \supseteq \Phi(v)$.

En d'autres termes, si u est dérivable de v par les opérateurs de formation de graphes (copie, restriction, simplification, jointure, etc.) alors $\Phi(u) \supseteq \Phi(v)$. La relation d'ordre partiel, qui est à la base du raisonnement dans le modèle des graphes conceptuels, peut être définie par l'existence d'une projection d'un graphe sur un graphe spécialisé. On en déduit des deux théorèmes précédents et de la définition de la projection, la propriété suivante:

Théorème 7-3 S'il existe une projection d'un graphe h sur un graphe g, alors $g \leq h$ et $\Phi(g) \supseteq \Phi(h)$.

Ainsi, la projection est un opérateur algébrique qui simule une inférence logique. Elle peut être vue comme la combinaison de plusieurs opérateurs de formation de graphes. La projection résume plusieurs pas de dérivation de graphes en une seule opération: si la projection d'un graphe *h* sur un autre graphe *g* est possible, alors *g* peut être dérivé de *h*.

Les derniers théorèmes sont très importants dans le cadre de ce projet de maîtrise. Ceux-ci nous indiquent qu'il est possible de définir un graphe conceptuel comme étant une conjonction d'éléments. Nous pouvons donc caractériser un graphe conceptuel comme étant une collection d'éléments individuels pour la transmission des connaissances, la mise en correspondances des connaissances et la recherche de sous-graphes dans la mémoire collective.

Voici donc une brève description du formalisme des graphes conceptuels. Nous allons maintenant voir un survol des techniques d'indexations utilisées pour améliorer la recherche d'informations dans une base de graphes conceptuels.

Indexation des graphes conceptuels

La représentation des connaissances n'est pas efficace si elle ne permet pas une méthode de recherche adéquate. Une façon d'accélérer la recherche est de créer un index sur les informations permettant ainsi un accès direct aux connaissances. Cependant, comment organiser cet index de façon à ce qu'il soit rapide et permette une mise à jour

facile des connaissances ? Plusieurs travaux ont été faits en ce sens en priorisant certains aspects de leur problématique. Cependant, peu ont utilisé la contrainte d'accès à cet index. Dans le cas d'un système distribué où l'accès au réseau a un coût, l'accès à l'index doit aussi être minimisé le plus possible. Nous allons voir les techniques qui ont été utilisés pour indexer les graphes conceptuels tout en gardant à l'esprit que nous cherchons une méthode d'indexation qui respecte les contraintes d'accès au réseau.

Dans leurs articles, G. Ellis et R. Levinson [ELL1993], [ELL1994], [LEV1994] ont utilisé une structure hiérarchique pour organiser la base de graphes conceptuels. Cette structure hiérarchique, organisée sous la forme d'un treillis semblable à celle de Sowa, utilisait une relation de subsomption pour agencer les graphes conceptuels. Les graphes étaient classés du plus général vers le plus spécifique. Ainsi, lors de la recherche d'informations, il suffisait de parcourir la hiérarchie en trouvant les graphes qui étaient suffisamment généraux sans être trop spécifique pour représenter le graphe de requête. Ce genre de structure apporte un cadre rigide à l'indexation et ne permet pas beaucoup de souplesse lors de la mise à jour des connaissances. Un recalcul de la structure est nécessaire lors de l'ajout ou le retrait d'informations. De plus, cette méthode n'est pas très efficace lorsque les connaissances sont très hétérogènes. La structure hiérarchique se retrouve avec un seul niveau qui possède tous les graphes conceptuels.

La bibliothèque CoGITaNT [COGIT2007] est un ensemble de classes *C++* permettant de manipuler facilement des graphes conceptuels ainsi que les autres objets du modèle. Pour indexer les graphes conceptuels, CoGITaNT offre un tableau de zones de

graphes et des fonctions permettant de construire, détruire et rechercher des graphes dans des zones de numéros données. Ils ont tenté d'utiliser ce tableau de zones pour implémenter la gestion d'une indexation des graphes en fonction de leurs méta-informations, certaines zones étant alors destinées à recevoir des graphes possédant telle méta-information ou tel groupe de méta-informations. Cependant avec ce type d'indexation, étant donné qu'un graphe ne peut appartenir qu'à une seule zone, l'utilisation d'une nouvelle méta-information entraînait la réorganisation de l'indexation. Dans la version ultérieure, l'indexation de graphes se fait à l'aide d'une table de hachage.[10] Dans ce cas, l'ajout est simplifié, cependant l'efficacité de la recherche est diminuée dans le cas d'une immense base de connaissances.

Dans son article, [MIN2000] présente une technique d'implantation de graphes conceptuels à partir des outils disponibles dans une base de données relationnelle. Cette technique permet de conserver la sémantique de manipulation et d'interprétation des graphes conceptuels en adoptant une organisation relationnelle. Les opérations de recherche et de manipulation sont déléguées au système de gestion de bases de données relationnelles (SGBDR). Cette façon de faire permet donc d'accélérer l'implantation du système à base de graphes conceptuels en profitant de plateformes de SGBDR optimisée pour la recherche et l'organisation des données. L'indexation des graphes conceptuels ainsi que la recherche est

[10] Une table de hachage est une structure de données qui permet une association clé-élément, c'est-à-dire une implémentation du type abstrait table de symboles. On accède à chaque élément de la table via sa clé. Il s'agit d'un tableau ne comportant pas d'ordre (un tableau est indexé par des entiers). L'accès à un élément se fait en transformant la clé en une valeur de hachage (ou simplement hachage) par l'intermédiaire d'une fonction de hachage. Les tables de hachage permettent en accès en O(1) en moyenne, quel que soit le nombre d'éléments dans la table.

déléguée aux SGBDR qui sont des logiciels éprouvés dans ce domaine et qui sont bien connus des programmeurs.

À la prochaine section, nous allons voir la technique qui a été utilisée pour maximiser les avantages de ces techniques tout en réduisant leurs désavantages. Nous avons principalement utilisé la technique présentée dans [MIN2000] tout en l'adaptant pour notre problématique. Ainsi, dans la suite de ce chapitre, nous allons voir la technique de représentation des graphes conceptuels présentés dans [MIN2000].

Une approche relationnelle pour les graphes conceptuels

Dans son article [MIN2000], G. Mineau expose une façon de représenter les connaissances selon un paradigme relationnel. La modélisation permet non seulement de représenter les connaissances, mais aussi d'exécuter des manipulations et des inférences sur ces connaissances. L'application au modèle relationnel permet de gérer une grande quantité d'information en utilisant des outils éprouvés, mais aussi d'élaborer une implantation plus rapide du système. Dans un tel système, les connaissances sont centralisées dans une base de données et les clients interrogent la base de données. Ce modèle permet donc d'implanter facilement une architecture client/serveur. Pour cette raison, nous avons utilisé ce paradigme d'implantation pour la mémoire collective.

Le canon

Lorsque nous avons présenté les éléments de bases des graphes conceptuels, nous avons parlé de la définition de concepts, de référents, de relations et de règles d'écriture (signature de relation). Ces éléments sont propres à un domaine et doivent être redéfinis pour chacun des systèmes. Ces éléments, dits éléments ontologiques, sont encodés dans une

structure que l'on appelle le *canon*. Ainsi, un graphe conceptuel n'a de sens que relativement à un canon donné.

Dans l'approche relationnelle présentée par [MIN2000], le canon est composé d'un 4-uplet < T, I, ::, B >. *T* est l'ensemble des types de concepts et de relations. *I* est l'ensemble de tous les référents existants identifiables ou non dans le domaine. La relation de conformité *::* est un opérateur qui vérifie l'appartenance d'un référent à un type de concept. Enfin, *B* est la base canonique du système de graphes conceptuels, c'est-à-dire l'ensemble des contraintes syntaxiques sur les relations incluant les signatures de relations.

L'ensemble de types T

L'ensemble de types *T* est défini par $T = T_c \cup T_r$. L'ensemble *T* est donc composé d'un ensemble de types de concepts (T_c) et d'un ensemble de types relations (T_r) qui sont relatifs au domaine *O*. Les éléments de *T* sont ordonnés en une structure de treillis fini notée (T, $\leq$) avec un élément universel $\top$ (le type le plus général) et un type absurde $\bot$ (le type le plus spécifique). Une relation de subsomption établit l'ordre entre les éléments de *T* qui sont comparables entre eux. Cependant, les types de concepts et de relations sont incomparables entre eux; les types de relations de différentes arités sont aussi incomparables entre eux.

Définition 7-4 (*Ensemble de types T*) Soit (T_c, $<_c$) un ensemble partiellement ordonné de types de concepts et (T_r, $<_r$) un ensemble partiellement ordonné de types de relations tels que:

- $T_c \cap T_r = \{\top,\bot\}$;
- $\bot <_c \top$ et $\bot <_r \top$;
- $\forall\, t \in T_c \setminus \{\top,\bot\}: \bot <_c t <_c \top$ et $\forall\, t \in T_r \setminus \{\top,\bot\}: \bot <_r t <_r \top$;
- $\forall\ t_1, t_2 \in T_r \setminus \{\top,\bot\}$: *arité*$(t_1) \neq$ *arité*(t_2) implique que t_1 et t_2 sont incomparables;

L'ensemble de référents I

L'ensemble des référents I est défini par $I = M \cup X \cup \{\forall\}$. L'ensemble I représente les instances des types de concepts (T_c). Lorsque le référent universel ($\forall$) est utilisé dans un nœud concept, ce référent fait référence à toutes les instances conformes à ce type de concept. Contrairement au référent universel ($\forall$), les éléments des ensembles *M* et *X* représentent un seul référent d'un type de concept. Ainsi, l'ensemble *M* représente les référents constants du système, soit les éléments connus à l'exécution, alors que l'ensemble *X* représente les référents variables qui ne sont pas connus lors de l'exécution, mais que l'on sait existants.

Définition 7-5 (*Ensemble de référents I)* M est l'ensemble des référents constants tel que $\forall \notin M$, et X est l'ensemble des référents variables tel que $\forall \notin X$ de sorte que $M \cap X = \varnothing$. Alors, l'ensemble I est défini par $I = M \cup X \cup \{\forall\}$.

Un référent sert à faire intervenir une instance précise du domaine d'application *O*. Ainsi, on définit une relation α entre les éléments de *I* et les éléments du domaine *O*, c'est-à-dire $\alpha : I \setminus \{\forall\} \rightarrow O$. Cette relation faite la mise en correspondance entre les éléments de *I* et de *O*.

La relation de conformité « :: »

La relation de conformité, notée « :: », définit des contraintes d'association entre un type de concept (T_c) et un référent dans $I \setminus \{\forall\}$. Cette relation vérifie si le référent est conforme au type de concept auquel il est affecté par rapport à une certaine interprétation. On définit donc une relation β qui fait le lien entre le paramètre de la relation et l'interprétation du paramètre dans le domaine noté *O*. Lorsqu'on utilise un type de concept $t \in T_c$ comme paramètre à la relation β, cette relation retourne le sous-ensemble d'éléments de *O* caractérisant ce type, $\beta(t) \subseteq O$. Toutefois, lorsque β utilise un type de relation $t \in T_r$ d'*arité*$(t) = n$ comme paramètre, la relation $\beta(t)$ retourne un ensemble de tuples de *n*

éléments de *O* pouvant instancier les concepts reliés à la relation, $\beta(t) \subseteq O^n$, en conformité avec sa signature.

Ainsi, on définit la relation de conformité *::* à l'aide de la relation α et β de la façon suivante :

Définition 7-6 (*Relation de conformité ::*) La relation de conformité *::* décide si oui ou non un référent est un élément de la classe d'un certain type en fonction de l'interprétation β de ce type. Formellement, nous disons que :

- $\forall\ t \in T_c$ et $i \in I \setminus \{\forall\}$, t::i = Vrai si $\alpha(i) \in \beta(t)$;
- $\forall\ t \in T_r$ tel que arité(t)=n et tous les tuples $(i_1, i_2, \ldots, i_n) \in (I \setminus \{\forall\})^n$, *t*::i = *Vrai*

  si $(\alpha(i_1), \alpha(i_2), \ldots, \alpha(i_n)) \in \beta(t)$;
- $\forall\ t \in T_c$, t:: $\forall$ si $\beta(t) \neq \varnothing$;

La base canonique B

La base canonique *B* contient les *graphes canoniques* d'un système de graphes conceptuels. Les graphes canoniques sont des graphes conceptuels à partir desquels sont dérivés les graphes composant le système. Ainsi, c'est à partir de ces graphes que l'on appliquera successivement les opérations canoniques pour former de nouveaux graphes

conceptuels. Dans le canon, les graphes canoniques représentent les signatures des relations sous la forme d'un graphe conceptuel. Le graphe conceptuel qui représente une telle signature est un graphe possédant seulement une relation *r* et *arité*(*r*) concepts rattachés à cette relation. Ainsi, un graphe conceptuel *h* est correctement formé par rapport à sa signature *g* s'il existe une *projection* (Définition 7-2) de *g* dans *h*. De cette façon, la base canonique *B* encode les contraintes syntaxiques sur les relations, en évitant qu'elles ne soient utilisées en surgénéralisation.

Définition 7-7 (*Base canonique B*) Définissons une fonction de signature de relation *sign*(*t*) qui fait la mise en correspondance entre chaque type de relation $t \in T_r \setminus \{\top,\bot\}$ avec sa signature $sign(t) \in (T_c \setminus \{\bot\})^{arité(t)}$. Définissons une fonction *comp*(*i*,*t*) qui retourne le $i^{ème}$ composant de la signature de la relation de type *t*, c'est-à-dire $comp(i,t) = t_i$ si $1 \leq i \leq arité(t)$ et $sign(t) = (t_1, t_2, \ldots, t_i, \ldots, t_{arité(t)})$. La fonction de signature *s* est conforme à l'ensemble T_r si pour tous $t_1,t_2 \in T_r \setminus \{\top,\bot\}$ de même arité et $t_1 \leq_r t_2$, alors pour tout $1 \leq i \leq arité(t_1)$, $comp(i,t_1) \leq_c comp(i,t_2)$. Pour une fonction de signature *s* conforme à l'ensemble des types de relations T_r, la base canonique *B* est définie par $B = \{<t, sign(t)> \mid t \in T_r \setminus \{\top,\bot\}\}$.

Les graphes conceptuels

Chaque graphe conceptuel doit se soumettre au canon auquel il est rattaché. Mais avant de continuer, il convient de formaliser la structure d'un *graphe conceptuel*. Nous les représentons sous la forme d'un 4-uplet $< C, R, E, l >$ composé de :

- Un ensemble C de concepts.
- Un ensemble R de relations.
- Un ensemble E d'arcs qui relient les relations et les concepts tels que $E \subseteq C \times R$.
- Une fonction d'étiquetage l tel que : chaque concept $c \in C$ est étiqueté avec un tuple $l(c) = (type(c), ref(c)) \in T_c \times I$ appelé type et référent; chaque relation $r \in R$ est étiquetée avec un type de relation $l(r) = type(r) \in T_r$; enfin chaque arc qui est incident à la même relation est étiqueté avec un nombre entier $\in [1,n]$ où n est l'arité de la relation. Par convention, l'arc possédant l'étiquette égale à n est l'arc sortant de la relation alors que tous les autres sont entrants.

Extension sémantique des graphes conceptuels

Sous l'hypothèse d'un monde fermé (*Closed World Assumption*), l'existence même d'un graphe dans un système implique qu'il existe au moins un arrangement de référents pour lequel ce graphe est vrai. La véracité d'un graphe est dépendante de l'existence

d'instances qui réalisent la sémantique du graphe. Pour déterminer les référents affectés à un graphe conceptuel *u*, nous définissons une fonction $\delta(u)$ qui retourne tous les tuples de référents pour lesquels *u* s'évalue à vraie lorsque ses concepts sont instanciés par ces référents. Ainsi, la recherche d'informations passe par cet opérateur qui est défini comme suit :

> **Définition 7-8** (Support d'un graphe) Le support d'un graphe conceptuel $u = <C_1, R_1, E_1, l_1>$, noté $\delta(u)$, est l'ensemble de tous les tuples $<i_1, i_2, \ldots, i_{|C1|}>$ qui instancient les concepts $i_j \in C_1$ de *u* et tel que la fonction $\alpha(i_j)$ fait correspondre au moins un élément de *O* pour chaque concept $i_j \in C_1$.

Pour qu'un graphe conceptuel soit vrai dans la base, il faut qu'il respecte les contraintes de l'ensemble *B* du canon et qu'il existe au moins un tuple de référents résultant de $\delta(u)$ pour lequel *u* est vrai :

> **Définition 7-9** (Évaluation d'un graphe) Formellement, l'extension sémantique d'un graphe conceptuel *u* est défini par la fonction valeur(*u*) := $(\delta(u) \neq \varnothing)$.

Nous utilisons une fonction d'ordre total pour indexer les concepts du graphe *u*. La fonction utilisée dans ce document est la fonction lexicographique $<_l$ appliquée sur les étiquettes des concepts. Ainsi, pour une paire de concepts distincts $c_1, c_2 \in C$ d'un graphe *u* nous avons que $l(c_1) <_l l(c_2)$ ou que $l(c_2) <_l l(c_1)$. Nous utilisons donc la fonction $<_l$ pour ordonner les concepts de la façon suivante

Définition 7-10 Pour tout concept $c \in C$ de u, on définit une fonction $pred(c_1,u) = \{c_2 \in C \mid c_1 \neq c_2$ et $c_2 <_l c_1\}$. Par la suite, il est facile de définir une fonction d'indexation pour classer les concepts : $index(c_1,u) = |pred(c_1,u)| + 1$. Également, on peut définir une fonction inverse $index^{-1}(i,u) = \{c \in C$ tel que $index(c,u) = i\}$.

Toutefois, cela n'est pas suffisant pour être complètement opérationnel car il faut, en plus, que l'assignation de l'index soit déterministe (chaque concept doit avoir un index unique). Il faut donc que les graphes respectent une structure où les référents sont uniques, appelée la *forme normale*. La forme normale d'un graphe est obtenue en fusionnant les nœuds concepts ayant le même marqueur individuel $\in M \cup X$. Ainsi, lorsqu'on utilise la forme normale, les éléments de *M* pointent tous vers des éléments distincts de *I*, c'est-à-dire que α est injective: il ne peut exister deux appellations différentes pour référer à une même instance d'un certain type.

Définition 7-11 (*Forme normale)* Un graphe conceptuel $u = <C, R, E, l>$ est en forme normale si pour n'importe quel concept c_1 et $c_2 \in C$, $ref(c_1) \neq ref(c_2)$.

Pour réaliser la forme normale, nous utilisons l'opérateur de jointure maximale sur l'ensemble de la base de connaissances. Une jointure maximale correspond à une série de jointures, non pas sur un seul nœud, mais sur un sous-graphe commun. Cette opération

s'exprime par une composition de jointures et de simplifications. Le résultat de cette opération forme un graphe normalisé.

Définition 7-12 (*Jointure maximale)* Soit les graphes conceptuels $g = <C_1, R_1, E_1, l_1>$ et $h = <C_2, R_2, E_2, l_2>$. L'opération de jointure maximale joint tous les nœuds compatibles de telle sorte que:

- pour n'importe quel nœud concept $c_1 \in C_1$ et $c_2 \in C_2$, $ref(c_1) = ref(c_2)$.
- pour n'importe quel nœud relation $r_1 \in R_1$ et $r_2 \in R_2$, $c_i \in C_1$ est relié à r_1 et $c_j \in C_2$ *est relié à* r_2 *et que* $ref(c_i) = ref(c_j)$ où i = j.

Pour assurer que la forme normale soit respectée, les variables doivent être globales au système. Donc, si la forme normale de *u* est respectée, nous savons que $|index^1(i, u)| = 1 \; \forall \; i \in [1, |C_u|]$ où $|C_u|$ représente le nombre de concept dans *u*. De plus, la forme normale évite des ambiguïtés sémantiques et logiques dans les graphes conceptuels. Sans la forme normale, deux graphes peuvent avoir la même interprétation logique sans avoir le même graphe qui le représente. Dans le cas de la mémoire collective, nous voulons éviter les doublons de graphes pour minimiser la recherche et homogénéiser les connaissances. C'est pour cette raison que nous avons besoin de la forme normale.

De plus, grâce aux définitions précédentes, on peut définir le support de *w*, un sous graphe de *u*, en fonction de $\delta(u)$. Le support d'un sous-graphe *w* de *u* est défini comme étant un sous-ensemble des référents de $\delta(u)$ pour lesquelles les concepts de *w* sont les mêmes que ceux de *u* :

Définition 7-13 (Support d'un sous-graphe) Si $w = <C', R', E', l'>$ est un sous-graphe de $u = <C_1, R_1, E_1, l_1>$ tel que $C' \subseteq C_1$, $R' \subseteq R_1$, $E' \subseteq E_1$, $l' \subseteq l_1$, $m = |C'|$ et $n = |C_1|$, alors on peut construire son support à partir de celui de u, notée $\delta_s(w,u)$. Nous avons que $\delta_s(w,u) := \{<i_{k1}, i_{k2}, \ldots, i_{km}>\}$ où $k_p \in [1,n]$ pour tout $p \in [1,m]$, et où pour chaque i_k il existe un j tel que $i_{kp} = i_j$ dans $<i_1, i_2, \ldots, i_n> \in \delta(u)$, où $k_p \neq k_q$ si $p \neq q$ pour tout $p, q \in [1,m]$ }.

La structure présentée dans cette section propose clairement une vision relationnelle des graphes conceptuels. Cette méthode ne change aucunement le formalisme des graphes conceptuels, mais l'adapte simplement pour une implantation plus simple. Plus de détails sur cette représentation sont présentés dans l'article [MIN2000].

Conclusion

Au cours de cette annexe, nous avons présenté le choix que nous avons fait quant au mode de représentation des connaissances : les graphes conceptuels. Nous avons présenté les avantages de ce formalisme de représentation des connaissances. Nous avons ensuite présenté brièvement les éléments importants du formalisme. Par la suite, nous avons présenté plusieurs techniques qui avaient été utilisées par le passé pour indexer les graphes conceptuels. Nous avons vu les techniques qui sont généralement utilisés pour ordonner les

graphes conceptuels de façon à accélérer l'accès aux connaissances. Enfin, nous avons vu une façon d'implémenter les graphes conceptuels à l'aide d'un système de gestion de base de données relationnelle. Nous avons vu qu'il est possible d'implémenter un système de graphes conceptuels en utilisant le paradigme des bases de données relationnelles. Cette approche permet de manipuler des bases de connaissances très volumineuses avec une technologie éprouvée.

En regard de notre problématique, nous avons choisi d'utiliser ce formalisme de représentation des connaissances pour sa capacité d'expressivité et sa facilité d'implantation. Pour appliquer ce modèle, nous avons choisi d'utiliser la technique d'implémentation présentée dans cette annexe. Cette technique facilite l'implémentation et a été éprouvée ce qui, selon nous, s'applique bien à notre problématique.

Annexe B

À la section 2.2.2, lors de la description de l'approche proposée pour représenter les connaissances, il a été mentionné que certaines propriétés d'encodage avaient été réalisées. Nous voulions produire une décomposition des connaissances qui respecte certaines propriétés de manipulation. En guise de rappel, nous voulions que la connaissance *k* soit *1) encodée de façon unique à un isomorphe près, 2) décomposée afin de générer la même sémantique qu'encodée à l'origine* $s^{-1}(s(k)) = k$ *et 3) encodée de façon à ce que certaines propriétés de manipulations soient respectées* :

P1) $k_1 \equiv k_2 \Rightarrow s(k_1) \equiv s(k_2)$

P2) $k_1 \subseteq k_2 \Rightarrow s(k_1) \subseteq s(k_2)$

P3) $((k_1 \cup k_2) \equiv k_3) \Rightarrow ((s(k_1) \cup s(k_2)) \equiv s(k_3))$

P4) $((k_1 \cap k_2) \equiv k_3) \Rightarrow ((s(k_1) \cap s(k_2)) \equiv s(k_3))$

En guise de rappel, voici la définition de la fonction de signature $s(k)$ présentée au Chapitre 2 :

> **Définition** (*Fonction de translation*) : Pour toute structure de connaissance *g*, une fonction de translation $s(g)$ est définie tel que $s(g) = \{P \mid P \subseteq \wp(G^1)\}$ où G^1 est l'ensemble de toutes les briques

possibles. Le choix de l'ensemble P parmi l'ensemble $\wp(G^1)$ est défini plus bas.

Inversement, la fonction s^{-1} fait correspondre l'ensemble $\wp$ résultant de $s(g)$, *à* une structure de connaissance g' équivalente à g, c'est-à-dire, si $s(g) = P \subseteq \wp(G^1)$, alors $s^{-1}(P) = g' = s^{-1}(s(g))$ et $g \equiv g'$. En terme de la théorie des graphes conceptuels, on dira que g' est le résultat de la jointure maximale de P puisque g' est un graphe qui résulte de la fusion de tous les nœuds ayant la même étiquette dans l'ensemble P.

Tel que mentionné plus haut, la fonction de translation relie un graphe vers un ensemble de briques $P \in \wp(G^1)$. Cet ensemble de briques P qui caractérise un graphe g est défini comme suit :

Définition *(Éclatement) :* L'éclatement d'un graphe conceptuel $g = <C_1, R_1, E_1, l_1>$ donne un ensemble de briques $P \in \wp(G^1)$. Ainsi, pour chaque relation $r \in R_1$, si le concept $c \in C_1$ rattaché à r possède un *degré(c)* supérieur à un, alors on duplicate le concept c par c' en conservant tous les arcs qui y sont connectés. On élimine ensuite l'arc *(r,c)* et on crée l'arc *(r,c')*. Dans le cas où il existe une composante connexe de g qui n'a pas de relation, on crée une relation de type universel (T) qu'on relie au concept.

En bref, s(g) est défini par construction tel que :

- $s(g) = \{g\}$ si g est un graphe constitué que d'une brique.
- $\{g_i\} \subseteq s(g)$ selon la définition de l'éclatement ci-haut, c'est-à-dire $\forall g_i \in P$
- $s(g) = s(g') \cup \{g_1\}$ où $g' = g$ et où l'on a retire le sous-graphe (brique) g_1 de g'.

P1) $k_1 \equiv k_2 \Rightarrow s(k_1) \equiv s(k_2)$

Démonstration 1 :

Soit $s(k_1)$ et $s(k_2)$ deux translations provenant respectivement de k_1 et k_2. Nous avons la fonction $s(k_1)$ qui donne un ensemble de sous-graphes minimaux de k_1 tel que :

$$k_1 \Rightarrow s(k_1) \Rightarrow \{\text{sous-graphes minimaux}\ ^{11}\text{de } k_1\} = A$$

De la même façon, nous avons que :

$$k_2 \Rightarrow s(k_2) \Rightarrow \{\text{sous-graphes minimaux de } k_2\} = B$$

Or, si $A \setminus B = \varnothing$ et que $B \setminus A = \varnothing$, il est impossible qu'il existe un sous-graphe $sg_1 \in k_i$ ($i \in [1,2]$) tel que $sg_1 \notin k_j$ ($j = 3 - i$) car cela voudrait dire que k_i porte plus d'information que k_j. Ainsi, si $k_1 \equiv k_2$ alors la fonction de translation donne une translation identique $s(k_1) \equiv s(k_2)$. ■

[11] Nous définissons un sous-graphe minimal d'un graphe comme étant l'une de ses briques, c'est-à-dire, appartenant à $P = s(g)$.

P2) $k_1 \subseteq k_2 \Rightarrow s(k_1) \subseteq s(k_2)$

Démonstration 2 ($\Rightarrow$) par contradiction

Soit $s(k_1)$ et $s(k_2)$ deux translations provenant respectivement de k_1 et k_2. Si $k_1 \subseteq k_2 \Rightarrow s(k_1) \subseteq s(k_2)$ cela signifie que k_1 est contenu dans k_2 et que $s(k_1) \subseteq s(k_1) \cup \{sg_1, sg_2 \ldots, sg_x\}$ tel que chaque sg_i correspond à une brique qui n'est pas dans $s(k_1)$. Il est donc impossible que $s(k_1) \supset s(k_2)$ car toutes les briques de $s(k_1)$ sont aussi contenues dans $s(k_2)$. Nous avons donc que $k_1 \subseteq k_2 \Rightarrow s(k_1) \subseteq s(k_2)$ ■

P3) $((k_1 \cup k_2) \equiv k_3) \Rightarrow ((s(k_1) \cup s(k_2)) \equiv s(k_3))$

Démonstration 3 ($\Rightarrow$)

Puisque nous avons déjà démontré que $k_1 \equiv k_2 \Rightarrow s(k_1) \equiv s(k_2)$ nous pouvons réutiliser cette preuve pour démontrer la propriété 3. Il ne nous reste qu'à démontrer que $s(k_1 \cup k_2) = s(k_1) \cup s(k_2)$. Nous allons donc démontrer cette propriété par construction. Ainsi, sachant que $s(\varnothing) = \varnothing$ et que pour n'importe quelle brique sg la fonction $s(sg) = \{sg\}$ voici les cas de la démonstration :

- **Cas 1 :**

  Soit la fonction $s(g \cup \varnothing)$, nous avons que $s(g \cup \varnothing) = s(g) \cup s(\varnothing) = s(g)$.

  Ce qui nous donne que $s(g \cup q) = s(g) \cup s(q)$ si $q = \varnothing$.

- **Cas 2 :**

  Soit un graphe g et une brique sg. La définition de la fonction de translation unit toutes les briques résultantes de l'éclatement pour composer la signature. Nous avons donc:

  $$s(g \cup sg) = s(g) \cup s(sg)$$
  $$= s(g) \cup s(q) \text{ tel que } q = sg$$

  ce qui nous donne que $s(g \cup q) = s(g) \cup s(q)$ si $q = sg$.

- **Cas 3 :**

  Soit les graphes g et q tel que $g \cap q = \varnothing$. Ce cas est identique au cas précédent, excepté que q est représenté par plusieurs briques. Nous avons donc:

  $$s(g \cup q) = s(g \cup \{sg_1, sg_2 \ldots sg_x\})$$
  $$= s(g) \cup s(sg_1) \cup s(sg_2) \cup \ldots \cup s(sg_x)$$
  $$= s(g) \cup s(q)$$

  ce qui nous donne que $s(g \cup q) = s(g) \cup s(q)$ si $g \cap q = \varnothing$.

- **Cas 4**:

  Soit les graphes g et q tel que $g \cap q \neq \varnothing$, $g = q \cup r$ et que $q \cap r \neq \varnothing$ nous avons:

  $$s(g \cup q) = s(q \cup r \cup q)$$

$$= s(r \cup q)$$

sachant que $q \cap r \neq \varnothing$ nous pouvons utiliser le cas précédent

$$= s(r) \cup s(q)$$

$$= s(q) \cup s(r) \cup s(q)$$

$$= s(q \cup r) \cup s(q)$$

$$= s(g) \cup s(q)$$

ce qui nous donne $s(g \cup q) = s(g) \cup s(q)$ si $g \cap q \neq \varnothing$.

Ce qui présente tous les cas possibles pour démontrer que $k_1 \cup k_2 \Rightarrow s(k_1) \cup s(k_2)$, ainsi nous pouvons dire, grâce à la démonstration 1, que $((k_1 \cup k_2) \equiv k_3) \Rightarrow ((s(k_1) \cup s(k_2)) \equiv s(k_3))$■

P4) $((k_1 \cap k_2) \equiv k_3) \Rightarrow ((s(k_1) \cap s(k_2)) \equiv s(k_3))$

Démonstration 4 (⇒)

Encore une fois, grâce à la démonstration 1, il ne nous reste qu'à démontrer que $s(k_1 \cap k_2) \Rightarrow s(k_1) \cap s(k_2)$. Nous allons donc démontrer cette propriété par construction.

- **Cas 1 :**

  Soit les graphes g et q tel que $g \cap q = \varnothing$. Nous avons donc:

  $$s(g \cap q) = s(\varnothing) = \varnothing$$

  ce qui est équivalent à $s(g_1 \cap g_2) \Rightarrow s(g_1) \cap s(g_2) = \varnothing$ avec $g \equiv g_1$ et $g \equiv g_2$.

- **Cas 2**:

  Soit les graphes g et q tels que :

  $$g \cap q \neq \varnothing, g \cap q = p$$

  $$g = p \cup r \text{ où } p \cap r = \varnothing,$$

  $$q = p \cup s \text{ où } p \cap s = \varnothing,$$

  $$r \cap s = \varnothing$$

  Ainsi, avec les propriétés définies à la démonstration 3, cas 3, nous avons:

  $$s(g) = s(p \cup r) = s(p) \cup s(\mathrm{r})$$

  $$s(q) = s(p \cup s) = s(p) \cup s(\mathrm{s})$$

  $$g \cap q = (p \cup r) \cap (p \cup \mathrm{s})$$

  $$= (p \cap p) \cup (p \cap \mathrm{s}) \cup (r \cap p) \cup (r \cap s)$$

  $$= p$$

  Nous avons donc que :

  $$s(g \cap q) = s(p \cap p) \cup s(p \cap \mathrm{s}) \cup s(r \cap p)$$

  $$\cup s(r \cap s) = s(p)$$

Ainsi, nous avons que si $((k_1 \cap k_2) \equiv k_3)$ alors la fonction de translation donne une translation identique $((s(k_1) \cap s(k_2)) \equiv s(k_3))$. ■