Université de Montréal

Réutilisation de méthodes de résolution de problèmes dans les systèmes à base de connaissances

par
Eliana de Mattos Pinto Coelho

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en informatique

Juillet 1997

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée:

Réutilisation de méthodes de résolution de problèmes dans les systèmes à base de connaissances

présentée par:

Eliana de Mattos Pinto Coelho

a été évaluée par un jury composé des personnes suivantes:

Esma Aimeur	(président-rapporteur)
Guy Lapalme	(directeur de recherche)
Vimla L. Patel	(codirecteur)
Jean Vaucher	(membre du jury)
Joost Breuker	(examinateur externe)

Thèse acceptée le _____

Sommaire

Dans cette thèse, nous nous sommes concentrée sur la question de la réutilisation de méthodes de résolution de problèmes. À l'aide d'une approche empirique, nous avons modélisé et implanté deux méthodes: *Proposer et Réviser* (P&R) et *Couvrir et Différencier* (C&D). Nous avons abordé trois aspects de ces méthodes: leur niveau de description, la spécificité de la structure des connaissances utilisée par ces méthodes, c'est-à-dire l'organisation des *rôles des connaissances*, et leur réutilisation.

Pour la description des méthodes, nous avons proposé un niveau intermédiaire entre le niveau conceptuel et le niveau d'implantation, avec la définition du langage K-Loom. Ce langage présente une syntaxe déclarative basée sur la logique terminologique, tout en étant un langage opérationnel. K-Loom permet de définir les méthodes de résolution de problèmes selon le modèle de l'expertise de la méthodologie KADS: les couches du domaine, des inférences et de la tâche. Contrairement à d'autres langages de KADS, K-Loom permet de représenter dans la couche des inférences la structure particulière des rôles des connaissances, au moyen d'une *ontologie de la méthode*. Cette ontologie sert aussi pour la définition des inférences et de la tâche, en favorisant la compréhension et la réutilisation de ces méthodes.

Nous proposons un compromis entre deux visions contradictoires par rapport à la réutilisation de connaissances. Une première vision, selon laquelle les connaissances ne peuvent être représentées qu'en considérant un problème spécifique (problème de l'interaction) et une deuxième qui propose la représentation des connaissances indépendamment des applications, de façon en favoriser la réutilisation. Nous supposons l'existence d'ontologies du domaine générales et réutilisables pour différentes tâches

et nous proposons la construction des ontologies de la méthode où les connaissances sont représentées en fonction de leur utilisation par la méthode. Le dilemme est résolu par l'établissement des correspondances entre ces deux ontologies. Nous avons proposé des opérateurs qui permettent des opérations syntaxiques, semblables aux opérations des bases de données, pour rendre une ontologie du domaine conforme à l'ontologie de la méthode.

Nous avons dégagé les caractéristiques qui limitent la réutilisation de ces méthodes. P&R, par exemple, n'est pas une méthode générale de configuration et ses rôles des connaissances ne sont pas complètement décrits, parce que la méthode s'appuie sur les connaissances heuristiques du domaine. C&D, par contre, est une méthode plus générale, dont nous montrons les limites pour une tâche complexe comme le diagnostic médical.

Enfin, dans notre travail, nous avons progressé vers la compréhension et la réutilisation de méthodes de résolution de problèmes. La définition de la meilleure approche pour décrire des méthodes réutilisables reste encore un problème ouvert. Nous soulignons, cependant, l'importance de décrire les rôles des connaissances au moyen de l'ontologie de la méthode pour faciliter l'indexation, la compréhension et la réutilisation de ces méthodes.

Table des matières

Table des figures

Liste des tableaux

Remerciements

Malgré toutes les occasions où je me suis plaint de la solitude du doctorat, sans l'aide de certaines personnes je ne l'aurais jamais terminé. Je tiens donc à remercier:

Mon directeur Guy Lapalme, pour avoir accepté de m'orienter dans cette thèse et fait tout son possible pour m'aider à la finir. Sa disponibilité, sa générosité et son intégrité vont me servir de modèle pour toute ma carrière future.

Vimla L. Patel, professeur au Département de Psychologie et de Médecine de l'Université McGill, pour avoir accepté d'assumer la codirection de cette recherche.

Joost Breuker, pour m'avoir fait l'honneur d'être l'examinateur externe. Je lui suis aussi reconnaissante pour sa disponibilité et pour avoir eu la gentillesse de s'intéresser à mon travail. Sa lecture attentive et ses commentaires ont beaucoup enrichi le contenu final de cette thèse.

Marco Ramoni, pour m'avoir beaucoup motivée au début avec des discussions et dont les suggestions bibliographiques m'ont aidée à démarrer ce projet.

Enrico Motta, pour m'avoir offert les données du système de configuration de l'ascenseur. Cela m'a permis de vivre une vraie expérience de réutilisation de connaissances et m'a aussi épargné beaucoup de travail.

Philippe Laublet, pour avoir lu et critiqué une partie de mon travail, en plus de ses suggestions bibliographiques.

André Valente, pour s'être intéressé à mon travail et avoir généreusement offert sa collaboration.

Tous mes collègues du laboratoire Incognito, pour leur amitié et leur appui. Merci surtout à Nicolas Anquetil, qui m'a beaucoup aidée (tout est très bien enregistré dans

mon fichier "notas") sur les questions techniques (et arides) de l'informatique.

Le CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), institut de subvention à la recherche du gouvernement brésilien, pour le support financier, ainsi que l'Université de Montréal.

Ma famille et mes amis, pour leurs encouragements et pour n'avoir jamais douté que j'arriverais à la fin, malgré toutes mes argumentations dans le sens contraire.

Michel Gagnon, pour m'avoir accompagnée presque tout au long de ce périple, en me soutenant avec son amour.

À Michel et Tomás

Science

Je commence à voir dans l'obscurité
un nouveau ton
d'obscurité.

Je commence à voir le vu
et je m'inclus
dans le mur.

Je commence à distinguer
un soupçon, à peine,
de ride.

Et à polir le plaisir
de la vie dans sa
fuite.

Carlos Drummond de Andrade

Chapitre 1

Introduction

Dans cette thèse nous abordons la réutilisation des connaissances dans les systèmes à base de connaissances (SBCs). Un système à base de connaissances est un programme conçu pour représenter et appliquer un ensemble de connaissances afin de résoudre des problèmes [Hayes-Roth et al., 1983]. Ces programmes sont développés pour des domaines divers tels que: la chimie, le génie, la médecine, les affaires, etc. [Waterman, 1986]. L'évolution de ces systèmes est telle qu'aujourd'hui on distingue deux générations de SBCs [Steels, 1984, David et al., 1993].

Dans la première génération, la construction de SBCs est réalisée à partir de rien et sans profiter de bases de connaissances déjà définies. Cela rend la création de nouveaux systèmes plus difficile et plus coûteuse. Par contre, dans la deuxième génération, l'ingénierie de connaissances évolue vers des méthodologies formelles pour la spécification des SBCs et la constitution de librairies de connaissances réutilisables.

On peut faire un rapprochement entre l'évolution de l'ingénierie de connaissances et celle du génie logiciel. Ainsi, en génie logiciel, les premiers programmes ont été développés sans aucune méthodologie. Le programmeur écrivait des blocs compacts de code en langages de bas niveau, les testait et les épurait jusqu'à ce qu'ils fonctionnent. Ces programmes étaient difficiles à comprendre et à modifier. De plus, il était ardu de les transposer dans un autre langage ou dans un autre environnement de travail. D'autre part, ces programmes n'étaient pas vérifiés, ni validés.

Pour résoudre ces problèmes, des méthodologies formelles ont été développées pour aider la construction de programmes. Ces méthodologies abordent le développement par l'analyse du problème à des niveaux d'abstraction différents, jusqu'à la conception. Ces méthodologies préconisent la modularité et favorisent la réutilisation de parties des programmes. Par conséquent, les systèmes développés sont plus faciles à comprendre et à modifier. De plus, des outils de vérification et de validation de programme ont été développés.

Si on analyse maintenant les systèmes à base de connaissances, on voit qu'on y retrouve les mêmes problèmes. Le cogniticien, avec l'aide d'experts, codifiait les connaissances sans aucune approche systématique et directement dans un formalisme de représentation de connaissances. Ces systèmes étaient difficiles à modifier, les explications données étaient ardues à comprendre et ils n'étaient pas réutilisables ou transférables à d'autres environnements.

Comme pour le génie logiciel, des méthodologies formelles [Alexander et al., 1986, Steels, 1990, Schreiber et al., 1993b] sont développées pour une construction plus systématique et bien fondée des SBCs. Ces méthodologies offrent plusieurs niveaux d'analyse des connaissances à des degrés d'abstraction différents. De plus, ces méthodologies préconisent la réutilisation des connaissances et favorisent la vérification et la validation des systèmes construits.

1.1 Évolution des SBCs

Pour mieux comprendre le contexte de notre travail, nous présentons dans cette section les principaux aspects de l'évolution des SBCs par rapport à la réutilisation des connaissances.

1.1.1 Modélisation de l'expertise

Dans la première génération, les connaissances sont des règles heuristiques obtenues auprès des experts. Dans cette perspective, la construction d'un SBC est

considérée comme une *tâche de transfert d'expertise*, c'est-à-dire un transfert de la connaissance des experts vers des bases de connaissances.

Les connaissances des experts utilisées pour la résolution de problèmes sont de type *compilé*. La métaphore "connaissances compilées" vient de l'informatique, où un programme écrit dans un langage de haut niveau comme Pascal est traduit dans un langage de plus bas niveau, de façon à être exécuté plus efficacement. Les connaissances humaines sont dites "compilées" quand elles sont organisées, indexées et conservées de façon à être plus facilement accédées. Les connaissances compilées, accumulées par l'expérience, sont des procédures spécifiques et des heuristiques qui permettent à l'expert de découper l'espace de recherche du problème [Harmon, 1987].

L'approche *transfert de l'expertise* a pour désavantage que le raisonnement de l'expert contient souvent des "sauts" qui sont le résultat de la "compilation" des connaissances. Ces sauts correspondent à des heuristiques acquises par l'expérience et l'automatisation d'une tâche réalisée plusieurs fois. Dans cette perspective, si les connaissances des experts sont codifiées telles quelles dans les systèmes, ces connaissances vont être compilées, ce qui rend difficile leur compréhension et leur maintenance. De même, ces systèmes ne fourniront pas d'explications compréhensibles. Finalement, ces systèmes ne peuvent pas résoudre des problèmes inattendus, parce que les principes généraux n'ont pas été codifiés. Seulement les heuristiques qui traitent des cas ordinaires sont codifiées.

Dans la deuxième génération, la construction d'un SBC est plutôt vue comme une *tâche de modélisation d'expertise*. Les bases de connaissances sont des modèles qui constituent une reconstruction rationnelle de la résolution d'un problème. Un modèle est une abstraction faite en considérant un but précis, qui permet de réduire la complexité en se concentrant sur certains aspects [Karbach et al., 1990]. Dans cette perspective, les connaissances des experts sont un point de départ pour la construction des SBCs et les experts aident à construire ces modèles. La structure des connaissances dans ces SBC peut être donc différente de celle des experts. Dans cette approche, l'acquisition de connaissances est vue comme une activité constructive et même créatrice: de l'interaction entre l'expert et le cogniticien résulte un modèle qui représente une

compréhension structurée des entités et des processus qui contribuent à la solution du problème [Linster, 1993].

L'avantage de l'approche de modélisation de l'expertise est que l'expert est obligé d'expliciter ses connaissances. Le modèle résultant est sujet à des modifications et des validations. De plus, les systèmes développés sont plus robustes parce qu'en construisant le modèle, les "sauts" de raisonnement des experts sont reconstruits et des problèmes non prévus peuvent être résolus. Les modèles générés guident l'acquisition de connaissances et permettent aussi aux systèmes d'offrir des explications plus compréhensibles, parce que toutes les étapes de la résolution du problème sont rendues explicites.

1.1.2 Niveau d'abstraction

Un autre aspect qui caractérise la première génération est le développement par prototypage rapide et par reformulation incrémentale. Les connaissances sont ajoutées, enlevées ou modifiées jusqu'à ce que le système atteigne un degré de performance acceptable. Les connaissances obtenues auprès des experts sont codifiées directement dans un formalisme de représentation de connaissances, tels que les règles de production, les réseaux sémantiques, les "frames", la logique de prédicats, etc. Les connaissances représentées dans ces formalismes correspondent au *niveau symbolique* d'un système expert. Le niveau symbolique est le niveau d'implantation d'un SBC.

Quand les connaissances sont codifiées directement dans un formalisme de représentation de connaissances, plusieurs aspects du problème sont négligés, parce que l'expert ou le cogniticien est trop centré sur les détails d'implantation et les contraintes syntaxiques du formalisme. Cette situation est provoquée par le manque d'abstraction du niveau symbolique utilisé pour spécifier les connaissances.

Dans la deuxième génération, les modèles sont construits en utilisant un niveau d'abstraction indépendant du formalisme de représentation de connaissances. Ce niveau d'abstraction plus élevé évite des décisions prématurées de conception, permet une modélisation plus complète du problème et facilite la communication entre les

experts et les cogniticiens. Newell [Newell, 1982] a proposé d'ajouter un niveau d'analyse de connaissances indépendant du niveau symbolique; *le niveau des connaissances* où les connaissances sont représentées indépendamment du formalisme utilisé pour l'implantation du SBC[1].

Par analogie avec les systèmes conventionnels, le niveau des connaissances correspond aux spécifications d'un programme, et les formalismes de représentation des connaissances (niveau symbolique) aux langages de programmation. Le niveau des connaissances offre des spécifications sémantiques, tandis que le niveau symbolique offre des explications plutôt syntaxiques. Construits selon le niveau des connaissances, les modèles facilitent l'acquisition de connaissances et contribuent à une meilleure conception, validation et maintenance des SBCs.

Dans la deuxième génération des SBCs, parce que les spécifications des connaissances sont indépendantes du formalisme de représentation, les connaissances sont plus réutilisables. Avant d'être représentées dans un formalisme implanté, les connaissances sont spécifiées en langue naturelle, en langages semi-formels et formels. Ainsi, les modèles construits selon le niveau des connaissances rendent les systèmes plus compréhensibles et plus adaptables pour différents langages de représentation des connaissances. Plusieurs méthodologies formelles offrant ce niveau d'analyse des connaissances ont été développées [Domingue et al., 1993, Serres, 1995, Van Heijst et al., 1994, Aussenac-Gilles et Matta, 1994].

L'autre avantage du niveau des connaissances est de distinguer entre "connaissance" et "formalisme de représentation". L'idée sous-jacente est d'avoir d'abord les connaissances spécifiées indépendamment du niveau d'implantation et de choisir ensuite le formalisme de représentation de connaissances le plus adéquat pour résoudre

1. L'usage du terme *niveau des connaissances* dans une grande partie de la littérature (et aussi l'usage dans cette thèse) s'est éloigné du sens original donné par Newell [Van de Velde, 1993]. Selon Newell, les connaissances à ce niveau devraient être décrites sans aucune structure. Ainsi, ces connaissances devraient être caractérisées fonctionnellement et non structurellement. En d'autres mots, les systèmes devraient être décrits en fonction de leur comportement simplement par la définition de leurs entrées et leurs sorties. Cependant, la plupart des méthodologies pour le développement des SBCs (chapitre 3) donnent une interprétation plus pragmatique, en considérant que les connaissances à ce niveau présentent déjà une certaine organisation.

le problème. Cela rend aussi possible la modélisation des facettes différentes d'un même problème (par exemple, un modèle causal, ou structural, ou comportamental) au niveau des connaissances et l'utilisation de différents formalismes de représentation au niveau symbolique. Ainsi, les modèles peuvent être décrits au niveau des connaissances et encodés avec différents langages de représentation. Cela permet au cogniticien de déterminer plus facilement, quand un problème survient, s'il est dû aux connaissances ou au formalisme de représentation utilisé. Pour en savoir plus à ce sujet, nous référons au survol fait par Simmons et Davis [Simmons et Davis, 1993], qui présentent des exemples de systèmes construits en utilisant différentes combinaisons de connaissances et de formalismes de représentation et qui montrent les avantages et les difficultés pour les combiner.

1.1.3 Connaissances du domaine et de contrôle

Deux types de connaissances peuvent être distingués pour la résolution de problèmes: les connaissances du domaine et les connaissances de contrôle. Les *connaissances du domaine* sont l'ensemble de faits, d'entités et de relations entre ces entités qui correspond à une représentation conceptuelle du domaine. Les *connaissances de contrôle* sont des opérations qui utilisent les connaissances du domaine et la stratégie d'application de ces opérations ("reasoning process") dans le but de résoudre un problème [Clancey, 1992]. Dans ce sens, les connaissances de contrôle contiennent deux types de connaissances : les opérations élémentaires sur les connaissances du domaine et la *structure de contrôle* sur ces opérations (l'ordre, le nombre de fois qu'une opération est exécutée et les conditions pour l'exécution de ces opérations).

Comme nous l'avons vu dans la section 1.1.1, dans la première génération, les connaissances obtenues des experts étaient du type compilé. Dans les connaissances compilées, les connaissances du domaine et de contrôle sont confondues, c'est-à-dire que les connaissances du domaine sont incorporées dans des programmes (les connaissances de contrôle) pour la résolution d'un problème particulier.

MYCIN [Buchanan et Shortliffe, 1984] est un système expert de première géné-

ration qui pose des diagnostics pour les maladies infectieuses. Ce système présente, en général, de bonnes performances. Cependant, dans sa tentative d'utiliser la base de connaissances du système MYCIN pour l'enseignement, Clancey [Clancey, 1987] a remarqué que les explications fournies étaient difficiles à comprendre à cause de la nature compilée des connaissances. La stratégie du raisonnement est implicite dans la structuration des règles. Il en découle que plusieurs pas du raisonnement et de la structure du domaine sont occultés. Certaines règles, ou l'ordre des règles, ou encore l'ordre des clauses dans les règles jouent un rôle important dans le processus de raisonnement qui reste caché dans l'organisation de la base de connaissances.

Pour résoudre les limitations du système MYCIN, Clancey a construit un nouveau système: NEOMYCIN [Clancey, 1986]. Ce nouveau système contient la base de connaissances de MYCIN réorganisée. NEOMYCIN est une décompilation de MYCIN dans le sens où la base de connaissances est restructurée de façon à ce que les connaissances de contrôle (dans ce cas, la stratégie de diagnostic) soient séparées des connaissances du domaine (figure 1.1). L'expérience de Clancey est un exemple de modélisation de l'expertise, où le nouveau système NEOMYCIN est une reconstruction rationnelle de la stratégie de résolution du problème qui était implicite dans MYCIN.

L'importance du travail de Clancey a été de montrer les avantages apportés par la distinction entre les connaissances du domaine et les connaissances de contrôle. Cette distinction favorise la réutilisation des deux types de connaissances: les connaissances du domaine réutilisées pour différents problèmes et les connaissances de contrôle réutilisées dans différents domaines. Par exemple, les connaissances du domaine de NEOMYCIN sont des connaissances médicales, comme l'association entre les symptômes et maladies. Les connaissances de contrôle sont l'ensemble des procédures et stratégies pour résoudre un problème de diagnostic. En étant séparées, les connaissances médicales peuvent être utilisées pour résoudre d'autres problèmes, comme la planification de thérapies; et les connaissances de contrôle peuvent être utilisées pour diagnostiquer dans d'autres domaines, comme la mécanique automobile.

Dans la prochaine section, nous concentrons notre attention sur la réutilisation

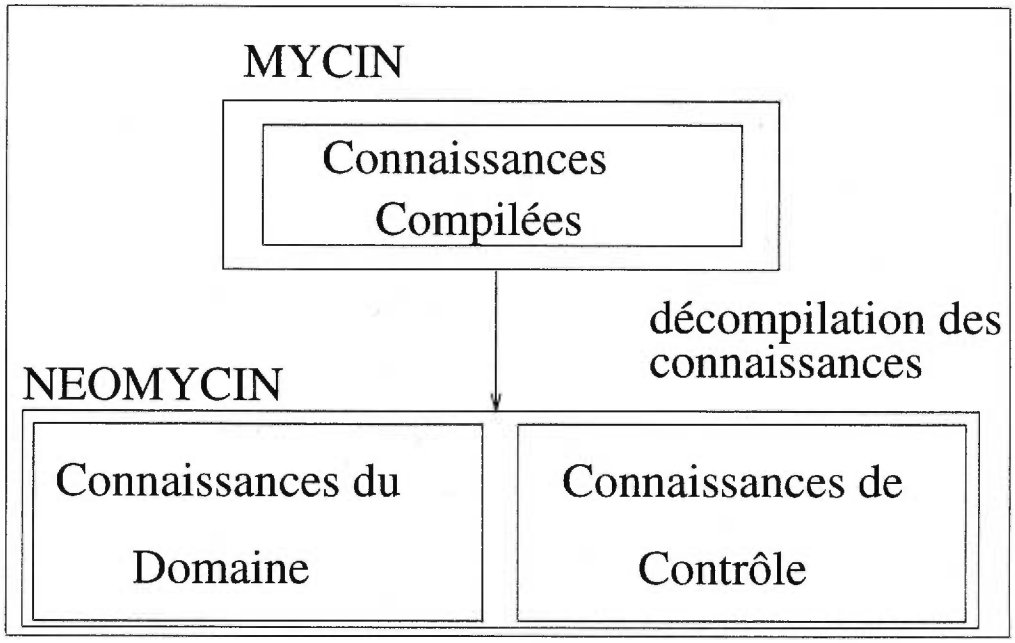

FIG. 1.1 - *Décompilation des connaissances dans le système MYCIN. MYCIN a donné naissance à NEOMYCIN. Les connaissances du domaine contiennent les connaissances médicales. Les connaissances de contrôle contiennent les procédures pour accomplir la tâche de diagnostic.*

des connaissances du domaine à partir de la construction d'ontologies.

1.1.4 Ontologies

Dans la section précédente, nous avons vu que la séparation des connaissances du domaine des connaissances de contrôle contribue à la réutilisation de ces deux types de connaissances. Dans cette section, nous introduisons le concept d'ontologie et nous montrons comment les ontologies contribuent directement à la réutilisation des connaissances du domaine.

À l'origine, le terme "ontologie" vient de la philosophie, où une "Ontologie" (avec "O" majuscule) est une discipline qui traite de la nature et de l'organisation de la réalité [Guarino et Giaretta, 1995]. En ingénierie de connaissances et dans cette thèse, nous utilisons le terme pour désigner une spécification explicite d'une conceptualisation [Gruber, 1993b].

Une conceptualisation correspond à un choix des objets et de relations entre eux pour représenter une partie du monde [Genesereth et Nilsson, 1987]. Par rapport à une tâche ou à un domaine, plusieurs conceptualisations sont possibles. Une conceptualisation est préférable à une autre si elle rend la résolution d'un problème plus facile ou si elle permet de comprendre plus facilement un domaine. Ces conceptualisations distinctes correspondent à diverses visions d'une même partie de la réalité, permettant de résoudre différents problèmes ou de résoudre un même problème plus facilement qu'un autre.

Toutes les caractéristiques d'un domaine ne pouvant être modélisées, un modèle est une simplification de la réalité. Il faut donc choisir un ensemble d'objets à modéliser et déterminer quelles caractéristiques et quelles relations entre les objets sont importantes. Pour représenter un ascenseur afin de résoudre un problème de configuration, il est fort probable qu'on veuille représenter les composantes de l'ascenseur, leur interconnexion et leur caractéristiques telles que poids, longueurs, etc, mais on ne va probablement pas représenter la couleur ou l'odeur de ces composants.

Une *ontologie* explicite une conceptualisation et est constituée d'un ensemble de

définitions de termes avec la description de leurs caractéristiques et des relations entre eux. Autrement dit, une ontologie exprime un ensemble de règles qui restreignent la structure d'une partie de la réalité représentée, en isolant et en organisant les objets et les relations pertinentes.

Ces restrictions expriment les *engagements ontologiques* [Gruber, 1993b] de la modélisation. Les engagements ontologiques sont les choix faits au moment d'une conceptualisation. Ils représentent un ensemble de contraintes caractérisant la conceptualisation de façon à restreindre la signification des termes. Ainsi, une ontologie n'est pas seulement une terminologie, mais aussi l'interprétation sémantique de ces termes [Van Heijst et al., 1997]. En d'autres mots, quand, par rapport à un domaine ou une tâche nous choisissons un ensemble d'objets, de relations et leur structure, nous sommes engagés ontologiquement dans une certaine vision du domaine au détriment d'une autre qui n'a pas été choisie [Valente, 1995].

Dans ce sens là, une ontologie n'est pas nécessairement une base de connaissances implantée. Une ontologie exprime explicitement ce qui est parfois implicite dans les bases de connaissances. Une autre différence est que les ontologies sont au niveau des connaissances, tandis que les bases de connaissances sont au niveau symbolique. Pour Wielinga et Schreiber [Wielinga et Schreiber, 1993], les ontologies sont des méta-modèles de bases de connaissances, parce qu'elles décrivent leur vocabulaire et leur structure. Ainsi, les ontologies ne contiennent pas les instances et les faits des bases de connaissances, mais seulement leur définition. Vues sous cette perspective, les ontologies représentent les intensions, tandis que les bases de connaissances représentent les extensions d'une modélisation.

L'idée de base de création des ontologies est de représenter de grandes quantités de connaissances pour qu'elles puissent être réutilisables. L'avantage des ontologies est que les connaissances peuvent être formalisées et validées une seule fois, et réutilisées plusieurs fois. Cela permet de construire plus rapidement des SBCs dont les résultats sont plus fiables.

Pour être partagées, les ontologies doivent exprimer un consensus et servir de référence dans une communauté de personnes ou d'agents.

De plus, pour être réutilisées, les ontologies doivent être construites sans tenir compte d'une application particulière, en étant le plus général possible. Pour maximiser la réutilisation des ontologies, on cherche à minimiser les engagements ontologiques. Plus précisément, on cherche à définir des ontologies indépendantes de leur utilisation ou d'une vision particulière. Après leur formalisation, ces connaissances, ou une partie de celles-ci, peuvent être partagées ou réutilisées dans différentes applications.

Le rôle des ontologies [Van Heijst et al., 1997] est d'aider l'acquisition de connaissances et aussi de supporter la conception informatique du système. Les ontologies servent aussi de médiateurs entre les connaissances telles que comprises par l'expert du domaine et leur représentation au niveau symbolique dans les bases de connaissances.

Ainsi, les ontologies, en explicitant les conceptualisations implicites des bases de connaissances, contribuent à la réutilisation de connaissances du domaine. Le travail de Clancey décrit dans la section précédente avance dans ce sens. Cependant, les ontologies représentent un pas de plus, en explicitant les engagements ontologiques des systèmes au niveau des connaissances.

Dans la prochaine section, nous nous attardons sur la réutilisation des connaissances de contrôle à partir de l'identification des méthodes de résolution de problèmes spécialisées dans l'exécution d'un problème particulier.

1.1.5 Méthodes de résolution de problèmes

La distinction entre les connaissances du domaine et les connaissances de contrôle contribue à deux types de réutilisation: par les connaissances du domaine et par les connaissances de contrôle. Dans la section précédente, nous avons abordé la réutilisation à partir de connaissances du domaine, au moyen de la construction d'ontologies. Ici, nous verrons l'autre type de réutilisation: la réutilisation de connaissances de contrôle à partir de l'identification de méthodes de résolution de problèmes.

Les systèmes experts de première génération sont constitués d'une base de connais-

sances et d'un moteur d'inférence. Différents moteurs d'inférences ont été développés. Ils se distinguent par l'ordre d'exécution des règles qui définit la stratégie de contrôle. Les stratégies de contrôle les plus communes sont le chaînage avant et le chaînage arrière. La séparation du moteur d'inférence de la base de connaissances permet à un même moteur d'inférence d'être réutilisé avec différentes bases de connaissances, constituant des coquilles de systèmes expert. EMYCIN [Van Melle et al., 1984] est la coquille de MYCIN et implante une stratégie de contrôle par chaînage arrière.

Dans la deuxième génération, la distinction de types de connaissances permet d'analyser les connaissances de contrôle qui sont implicites dans les SBCs de première génération. Plusieurs travaux ont été faits dans le but d'identifier, de décrire et de comparer différents types de connaissances de contrôle [Chandrasekaran, 1983, Clancey, 1985, McDermott, 1988]. Ces démarches tendent à créer des librairies de connaissances de contrôle réutilisables. Les connaissances de contrôle identifiées sont appelées *méthodes de résolution de problèmes.* Une méthode de résolution de problèmes est constituée de connaissances de contrôle spécifiques pour la résolution d'un problème particulier. La Classification Heuristique [Clancey, 1985], par exemple, est une méthode pour résoudre un problème de classification, tels le diagnostic médical ou la sélection dans un catalogue. L'identification de ces méthodes facilite la construction des SBCs: étant donné un problème, il faut associer ce problème à la méthode la plus adéquate pour le résoudre.

Dans la deuxième génération des SBCs, on trouve un nouveau niveau de réutilisation de connaissances. Dans la première génération, les moteurs d'inférence agissent directement sur les bases de connaissances, en établissant l'ordre d'application des règles. Dans les SBCs de deuxième génération, les moteurs d'inférence agissent sur les connaissances de contrôle qui, à leur tour, agissent sur les connaissances du domaine. Dans cette perspective, nous pouvons avoir différents niveaux de réutilisation: des moteurs d'inférence différents appliqués sur différentes connaissances de contrôle, et des connaissances de contrôle différentes appliquées sur différentes connaissances du domaine. La figure 1.2 illustre l'ajout de cet autre niveau de réutilisation.

Ainsi, dans les SBCs de la deuxième génération, les outils ont évolué des coquilles

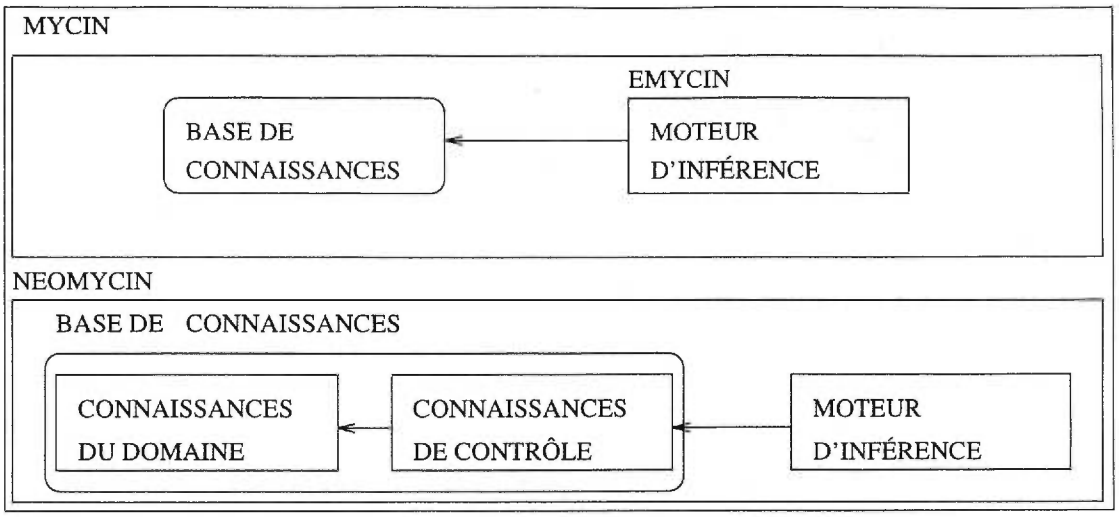

FIG. 1.2 - *Architecture des SBCs de première et deuxième génération. Dans MYCIN, le moteur d'inférence interprète la base de connaissances où se trouvent les connaissances du domaine médical et les connaissances pour la tâche de diagnostic. Dans NEOMYCIN, le moteur d'inférence interprète les connaissances de contrôle qui ont été explicitées et ces connaissances de contrôle opèrent sur les connaissances du domaine médical.*

génériques vers des outils qui exécutent des tâches particulières comme le diagnostic, la conception, l'assemblage, etc (figure 1.3). À la section 1.1.2, nous avons vu que ce qui différencie les architectures des deux générations de SBCs est leur niveau d'abstraction: tandis que EMYCIN est défini en termes d'une implantation spécifique (un formalisme de représentation basé sur les règles de production avec chaînage arrière), les connaissances de contrôle de la deuxième génération sont définies en termes d'une classe abstraite de méthodes de résolution de problèmes [Wenger, 1987]. La conséquence de ce changement d'architecture se situe au niveau de la réutilisation: dans la première génération, la réutilisation est en termes de coquilles génériques; dans la deuxième génération, la réutilisation est en termes de méthodes de résolution spécialisées dans la résolution d'un problème spécifique.

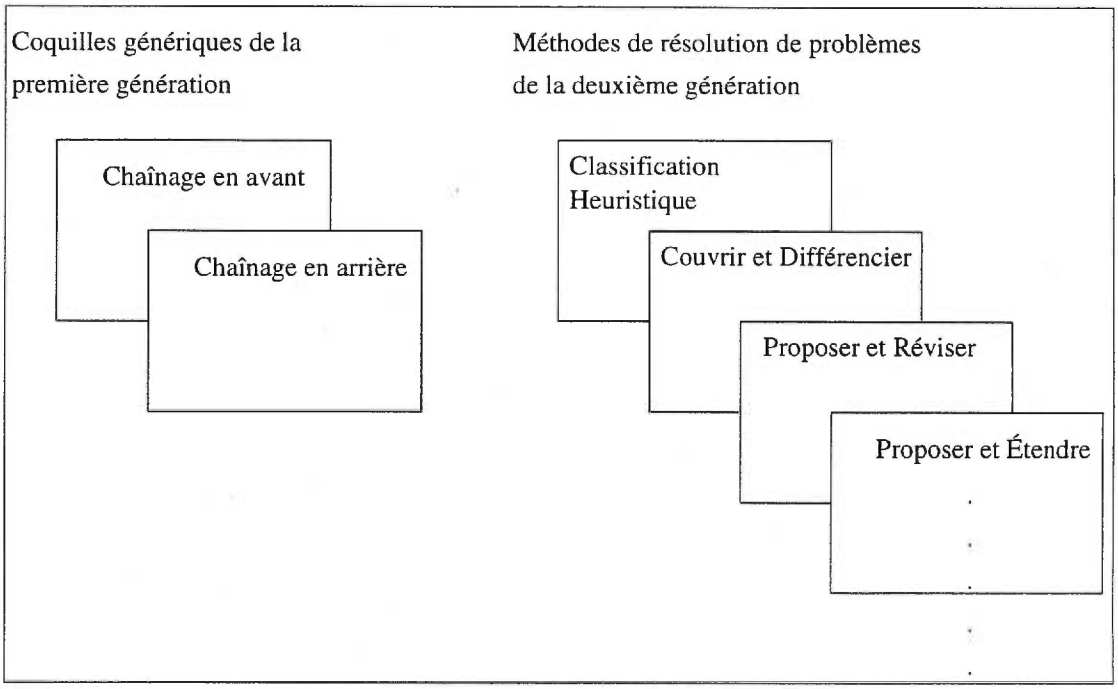

FIG. 1.3 - *Coquilles des SBCs: la première génération versus la deuxième génération. Dans la première génération, la réutilisation est définie en termes du niveau symbolique. Dans la deuxième génération, la réutilisation est définie en termes du niveau des connaissances.*

1.2 Présentation de cette thèse

Dans notre travail nous traitons de l'acquisition de connaissances dans les SBCs, plus particulièrement leur modélisation[2]. Nous nous concentrons surtout sur la question de la réutilisation des méthodes de résolution de problèmes, donc des connaissances de contrôle. Nous analysons ces méthodes pour trouver une manière de les décrire et de mieux les exprimer de façon à en faciliter la réutilisation.

1.2.1 Approche

Nous suivons une approche empirique en modélisant et en implantant les méthodes. Nous nous limitons à deux méthodes: *Proposer et Réviser* (P&R)[3] et *Couvrir et Différencier* (C&D)[4]. Ces méthodes sont appelées "méthodes à limitation de rôles" [McDermott, 1988], parce qu'elles sont basées sur la définition des rôles joués par les connaissances dans la solution du problème.

Le choix de ces deux méthodes est motivé par notre intérêt pour la définition de la structure des rôles des connaissances des méthodes. L'autre raison qui motive ce choix est que nous disposons d'exemples de domaines d'application pour ces méthodes: un exemple de problème de configuration d'ascenseur pour P&R; un exemple de problème de diagnostic médical pour C&D.

Dans cette thèse, nous définissons une méthode de résolution de problèmes comme étant une description de l'ordre d'exécution d'un ensemble d'opérations sur une structure de connaissances particulière. Cette structure définit les *rôles* joués par les connaissances dans la résolution du problème. Les rôles sont des "placeholders" pour les connaissances du domaine et leur nom est un indicatif du rôle de ces connaissances dans le processus de résolution du problème. Ces rôles peuvent être instanciés pour différents domaines. Dans la figure 1.4, nous illustrons une vision simplifiée de la méthode de résolution de problèmes C&D avec ses opérations et ses rôles des connais-

2. Un autre aspect de l'acquisition de connaissances concerne l'explicitation de connaissances auprès des experts. Cet aspect n'est pas considéré dans le cadre de cette thèse.

3. *Propose and Revise*

4. *Cover and Differentiate*

sances. Cette méthode peut être utilisée pour accomplir une tâche de diagnostic, où les rôles sont instanciés pour différents domaines. Par exemple, si le diagnostic est médical, l'**explication finale** sera une maladie ou, si le diagnostic est dans le domaine électronique, l'**explication finale** sera un composant d'un circuit.

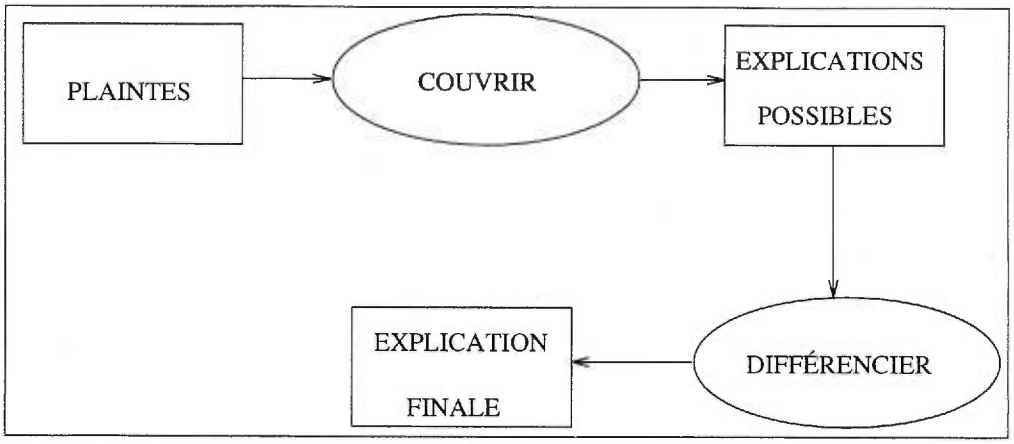

FIG. 1.4 - *Exemple d'une méthode de résolution de problèmes avec ses opérations et ses rôles des connaissances. Les rectangles représentent les rôles des connaissances et les ellipses représentent les opérations.* ***Couvrir*** *génère des* **explications possibles** *à partir des* **plaintes** *fournies par l'utilisateur.* ***Différencier*** *génère une l'***explication finale** *à partir des* **explications possibles**.

Pour décrire les méthodes de résolution de problèmes, nous utilisons les éléments suivants:

1. Une vision globale et abstraite des opérations et des rôles des connaissances.
2. Les rôles joués par les connaissances dans la résolution du problème.
3. Les opérations en fonction des rôles.
4. La structure de contrôle de la méthode, c'est-à-dire l'ordre d'exécution des opérations.

Pour définir les méthodes, nous utilisons comme cadre le modèle de l'expertise de KADS [Wielinga et al., 1993], une méthodologie pour le développement des SBCs. KADS distingue trois catégories de connaissances: domaine, inférence et tâche. Ces

trois catégories de connaissances sont organisées hiérarchiquement avec une interaction limitée, ce qui permet leur réutilisation pour différents domaines et problèmes.

Pour spécifier les méthodes, nous définissons un langage. Ce langage, appelé K-Loom (KADS-Loom), est une une extension de Loom. Loom [MacGregor, 1988, MacGregor et Burstein, 1991, Brill, 1993] est un langage de représentation de connaissances dérivé de KL-ONE [Brachman et Schmolze, 1985]. Il offre la capacité de raisonnement terminologique qui consiste à représenter les connaissances en termes de caractéristiques définissant les concepts et les relations.

1.2.2 Problèmes et contributions

Nous résumons les problèmes liés à la réutilisation des méthodes de résolution de problèmes traités dans cette thèse, ainsi que nos contributions pour les résoudre. Plus particulièrement, les problèmes résident dans les aspects suivants concernant les méthodes: le niveau de description, la spécificité des rôles des connaissances et la réutilisabilité des méthodes.

Le niveau de description

Les méthodes à limitation de rôles sont décrites soit au niveau d'implantation (symbolique), soit à un niveau trop abstrait. Ces deux niveaux de description empêchent souvent la compréhension des méthodes et, par conséquent, leur réutilisation. Le niveau abstrait permet d'avoir une compréhension globale de la méthode, mais il manque de précision. Le niveau d'implantation offre les informations plus précises sur la structure de connaissances, mais il est trop difficile à comprendre par la surcharge des détails.

Dans C&D, par exemple, les rôles sont identifiés par **plaintes** et **explications** (figure 1.4). L'idée de base est que ces rôles peuvent être remplis par des connaissances de différents domaines pour résoudre un problème de classification. Cependant, si un cogniticien veut réutiliser C&D, il ne sait pas exactement ce que la méthode fait ou quels sont les préalables pour l'utiliser.

Pour résoudre le problème du niveau de description des méthodes à limitation de rôles, nous définissons un niveau intermédiaire qui spécifie la structure des rôles des connaissances par la construction d'une *ontologie de la méthode*. Cette ontologie dévoile les caractéristiques des rôles des connaissances utilisés par la méthode ainsi que les restrictions qui s'appliquent à celle-ci. Nous utilisons le terme "ontologie de la méthode", dans le même sens que dans l'environnement de développement des SBCs PROTÉGÉ-II [Musen et al., 1994]. Ici, l'ontologie de la méthode représente les rôles des connaissances au moyen de concepts et de relations entre eux.

De plus, les opérations (inférences) sont définies en fonction de l'ontologie de la méthode. Cela va permettre une description plus précise des inférences. En spécifiant les inférences par rapport à l'ontologie de la méthode, nous faisons un parallèle avec la typologie d'inférences de KADS. Nous montrons comment les inférences de méthodes peuvent être vues comme des instances de cette typologie.

Nous définissons ensuite la structure de contrôle de la méthode qui spécifie l'ordre d'exécution des inférences, par exemple, le nombre de fois qu'une inférence est exécutée ou des tests qui conditionnent l'exécution d'une inférence. Une fois les inférences définies, il est plus facile de configurer différentes structures de contrôle en adaptant la méthode pour des problèmes légèrement modifiés. Cela permet d'évaluer l'efficacité de la méthode pour des structures de contrôle distinctes.

Finalement, nous montrons les avantages de ces définitions avec K-Loom qui tire profit des caractéristiques terminologiques et de classification de Loom pour représenter les couches du domaine, des inférences et de la tâche de KADS; nous disposons ainsi d'un langage de haut niveau d'abstraction, mais opérationnel.

La spécificité des rôles des connaissances

Comme nous l'avons vu, nous partons du principe que les méthodes opèrent sur une structure des connaissances particulière. La nature des rôles des connaissances est donc spécifique à la méthode. Les méthodes de résolution de problèmes définissent une organisation des connaissances qui les rend moins réutilisables pour les ontologies

générales.

Donc, il est difficile de réutiliser les méthodes pour différents domaines qui présentent une structure des connaissances précise qui ne coïncide pas toujours avec la structure de connaissances de la méthode. Comme nous l'avons vu dans la section 1.1.4, la réutilisation des connaissances du domaine implique de disposer d'ontologies définies avec le minimum d'engagement ontologique par rapport à une application. Il y a donc apparemment une contradiction entre ces deux objectifs: pour décrire une méthode, il faut définir la structure des connaissances spécifique utilisée par la méthode; par contre, les ontologies du domaine existantes sont générales afin d'être réutilisables pour plusieurs applications. Cette contradiction en ingénierie des connaissances est exprimée par le dilemme de la *réutilisabilité* versus l'*utilisabilité* des composants:

- Un composant est réutilisable s'il est suffisamment général de façon à être réutilisé dans différents domaines et problèmes.

- Un composant est utilisable s'il est suffisamment spécifique pour être utilisé dans un domaine et un problème particuliers.

Ainsi, pour en maximiser la réutilisabilité, les ontologies doivent être construites indépendamment de leur utilisation par une méthode; pour en maximiser l'utilisabilité, les rôles des connaissances de la méthode doivent être définis en dévoilant leur spécificité.

Le problème de l'utilisabilité et de la réutilisabilité des connaissances est abordé dans les environnements de développement de SBCs Spark-Burn-Firefighter (SBF) [Klinker et al., 1991] et PROTÉGÉ-II [Gennari et al., 1993]. Dans notre solution, nous nous inspirons de l'approche PROTÉGÉ-II, où à partir des méthodes de résolution de problèmes réutilisables, le dilemme est résolu par la définition d'opérateurs de conversion entre différentes ontologies.

Ainsi, nous définissons quelques types d'opérateurs de correspondance pour rendre viable l'adaptation des ontologies générales du domaine aux ontologies associées aux méthodes de résolution de problèmes. Ces types d'opérateurs étendent les opérations

présentes dans PROTÉGÉ-II. Ils facilitent le développement de deux types d'ontologie (de la méthode et du domaine), en contribuant à la réutilisation des connaissances du domaine et des méthodes de résolution de problèmes. Les avantages des opérations de correspondance sont l'indépendance entre les ontologies de méthode et les ontologies du domaine, et le fait qu'une même ontologie du domaine peut être considérée sous différents points de vue.

La réutilisabilité des méthodes

Les méthodes sont dites réutilisables, mais il n'est pas facile de définir dans quelle mesure ces méthodes sont vraiment applicables pour différentes tâches et domaines.

La plupart des approches pour la construction de SBCs distinguent les concepts de méthode et de tâche [Karbach et al., 1990]. Une méthode est définie comme une procédure qui résout un problème. Une tâche comprend la définition du problème et la méthode choisie pour le résoudre [Breuker, 1994]. Il peut y avoir plusieurs méthodes pour accomplir une tâche et, inversement, une méthode peut résoudre différentes tâches. Par exemple, une tâche de diagnostic peut être résolue par les méthodes Classification Heuristique, C&D, ou GDE ("general diagnostic engine") [de Kleer et Williams, 1987]. De même, une méthode comme la Classification Heuristique peut être utilisée pour accomplir différentes tâches telles que la classification, la sélection dans un catalogue, le diagnostic médical ou de problèmes de voiture, etc. Dans ce cas, il faut faire une analyse des paires *problème/tâche* pour déterminer la méthode la plus adéquate pour résoudre le problème.

Cependant, le degré de réutilisabilité des méthodes de résolution de problèmes n'est pas toujours clair:

- Sont-elles vraiment réutilisables pour différents domaines?
- Est-il facile de les trouver et de les associer à différentes applications?
- Les rôles des connaissances explicités simplifient-ils le travail du cogniticien en évitant d'avoir à ajouter des connaissances spécifiques à son application?

Nous analysons ces méthodes par rapport à l'étendue de leur réutilisation. En nous servant de résultats obtenus à propos de P&R, C&D et d'autres systèmes qui résolvent les mêmes types de problèmes, nous concluons que la réutilisabilité de ces méthodes est restreinte à cause des caractéristiques des rôles des connaissances. Nous montrons qu'il serait souhaitable que les rôles des connaissances explicitent les distinctions conceptuelles des connaissances, au lieu de laisser ces distinctions implicites.

Les méthodes étudiées représentent un progrès par rapport aux systèmes de première génération. Cependant, nous concluons qu'il y a encore beaucoup de difficultés à surmonter avant de rendre les méthodes vraiment réutilisables. Nous croyons à la nécessité d'une description plus précise des rôles des connaissances de ces méthodes pour en favoriser la compréhension et la réutilisabilité.

1.2.3 Organisation de la thèse

Le chapitre 2 expose les deux approches pour la réutilisation de connaissances: à partir des connaissances de contrôle, et à partir des connaissances du domaine. Nous concluons le chapitre en présentant une troisième approche qui combine ces deux approches.

Dans le chapitre 3, nous présentons KADS, en particulier son modèle de l'expertise pour décrire les méthodes de résolution de problèmes. Nous présentons également des aspects de la modélisation en KADS tels que les langages, la bibliothèque de composants réutilisables et la typologie d'inférences.

Le chapitre 4 définit notre langage pour modéliser les méthodes de résolution de problèmes, K-Loom. Ce langage est défini selon le modèle de l'expertise de KADS; il peut être vu comme une coquille de KADS sur Loom et fournit une interface en Loom pour développer des modèles de l'expertise.

Le cinquième chapitre introduit le concept d'ontologie de la méthode que nous utilisons pour décrire les inférences et la structure de contrôle des méthodes de résolution de problèmes. L'intérêt d'une telle description est de se situer à un niveau intermédiaire entre le niveau conceptuel et le niveau d'implantation, tout en dévoilant

la structure des rôles des connaissances des méthodes de résolution de problèmes.

Dans le chapitre 6, nous définissons un ensemble de types d'opérations de correspondance entre les ontologies. Ces opérations permettent de résoudre partiellement l'incompatibilité entre la généralité des ontologies du domaine et la spécificité des ontologies des méthodes. Nous montrons que même avec une équivalence sémantique entre les ontologies du domaine et de la méthode, il faut expliciter des opérations de correspondance de façon à établir une correspondance syntaxique entre ces deux ontologies.

Le chapitre 7 discute de la réutilisabilité de nos méthodes, en les comparant avec d'autres. Le dernier chapitre résume nos contributions et présente des perspectives pour des travaux futurs.

L'appendice A détaille l'implantation de P&R appliquée à la tâche de configuration d'ascenseur, en présentant une trace de l'exécution du système. L'appendice B illustre l'exécution de C&D appliquée à une tâche de diagnostic médical.

Chapitre 2

Approches pour la réutilisation de connaissances

Nous avons vu au chapitre 1 qu'un des aspects de l'évolution des SBCs est la distinction entre les connaissances du domaine et les connaissances de contrôle. Cette séparation a donné lieu à deux approches pour la réutilisation des connaissances:

- La réutilisation par les connaissances de contrôle, c'est-à-dire par les méthodes de résolution de problèmes.
- La réutilisation par les connaissances du domaine, c'est-à-dire par les ontologies.

Dans plusieurs travaux en ingénierie des connaissances, ces deux approches ont été considérées incompatibles. Selon l'approche de réutilisation par les connaissances de contrôle, les connaissances du domaine ne peuvent être représentées qu'en considérant un but ou un problème spécifique, alors que selon l'approche de réutilisation par les connaissances du domaine, elles doivent être représentées indépendamment de leur utilisation. Nous présentons dans ce chapitre les idées et les avantages de chaque approche. Ensuite nous montrons que la tendance actuelle est de combiner ces deux approches. Cette troisième approche conciliatrice est celle adoptée dans notre thèse.

2.1 Réutilisation par les connaissances de contrôle

Dans la réutilisation par les connaissances de contrôle, l'idée principale est de définir un ensemble de méthodes de résolution de problèmes. Cette approche présente les avantages suivants:

- Certaines classes de problèmes typiques se répètent d'un domaine et d'une application à l'autre. Ces classes sont restreintes et les méthodes pour les résoudre peuvent être caractérisées par leur forme (structure), plutôt que leur contenu. Des classes de problèmes typiques sont, par exemple: classification, planification, configuration, contrôle ("monitoring"), évaluation, etc[1].

- Certaines tâches complexes peuvent être accomplies en combinant plusieurs méthodes. Par exemple, pour la tâche de conception, on retrouve des problèmes de contrôle, d'évaluation et de diagnostic, en plus du problème central de conception [Breuker et Van de Velde, 1994].

- Les méthodes comportent des composants qui sont réutilisables d'une méthode à l'autre. Par exemple, la typologie d'inférences canoniques de KADS (section 3.2.3).

Comme nous l'avons dit plus tôt, selon le point de vue de la réutilisation par les connaissances de contrôle, les connaissances d'un domaine ne peuvent être représentées qu'en considérant un but spécifique. Cette perspective en ingénierie des connaissances est appelée *problème de l'interaction* [Bylander et Chandrasekaran, 1987], parce que les connaissances de contrôle interagissent avec les connaissances du domaine, de façon à déterminer comment ces dernières doivent être représentées:

> "La représentation des connaissances dans le but de résoudre un problème est fortement affectée par la nature du problème et par la stratégie d'inférence devant être appliquée aux connaissances."

1. Il manque cependant un critère mieux défini pour classifier les types de problèmes. Certains travaux se concentrent sur cette classification, notamment [Clancey, 1985, Breuker, 1994].

Selon les défenseurs de cette vision, le monde est trop compliqué pour être modélisé sans but spécifique [Mizoguchi et al., 1995]. Ainsi, la représentation des connaissances dépend de leur utilisation. Autrement dit, les connaissances ne peuvent pas être représentées sans une explicitation des *suppositions* sur la manière dont elles vont être utilisées.

Ces suppositions définissent les *rôles* joués par les connaissances dans la résolution du problème. Ces rôles sont utilisés pour connecter différents domaines avec une méthode. Ils sont des "placeholders" pour des connaissances d'un domaine particulier.

Ainsi, les méthodes de résolution de problèmes sont définies en fonction des rôles des connaissances et chaque méthode spécifie les caractéristiques particulières de ceux-ci. Ces caractéristiques permettent au cogniticien de vérifier les conditions préalables d'utilisation de la méthode, avant de la choisir pour résoudre un problème spécifique.

La définition des rôles des connaissances et l'explicitation de leur caractéristiques apportent les avantages suivants:

- Les rôles des connaissances peuvent être instanciés pour différents domaines.
- La définition des rôles permet de délimiter quelles sont les parties des connaissances du domaine à représenter. Cela aide l'acquisition des connaissances, parce que la méthode détermine les connaissances nécessaires pour résoudre le problème ("model-driven knowledge acquisition").
- Des outils d'acquisition de connaissances peuvent être développés pour automatiser l'obtention des connaissances dans différents domaines. Cela peut simplifier la construction, la vérification et la validation des bases de connaissances.
- Les rôles définissent les connaissances de la méthode en fonction de leur utilisation, mais ces connaissances sont quand même représentées séparément des connaissances de contrôle (des inférences et de la tâche).
- L'explicitation des suppositions à propos des rôles des connaissances permet de définir les caractéristiques des connaissances de la méthode et peut aider à trouver la méthode adéquate pour résoudre un problème.

- Les rôles permettent d'identifier les modèles de connaissances sous-jacents aux méthodes. Par exemple, les modèles hiérarchiques, causaux, fonctionnels, etc, qui explicitent l'utilisation des connaissances dans le processus de raisonnement.

Dans les sections suivantes, nous montrons des travaux en ingénierie des connaissances qui adoptent la réutilisation par les connaissances de contrôle, en définissant les rôles joués par les connaissances dans la résolution du problème: les tâches génériques, la Classification Heuristique et les méthodes à limitation de rôles.

2.1.1 Tâches génériques

Chandrasekaran et ses collègues [Chandrasekaran, 1983, Chandrasekaran, 1986, Bylander et Chandrasekaran, 1987] ont identifié et classifié quelques *tâches génériques* devant être utilisées comme des composants des SBCs. Elles sont considérées élémentaires parce que plusieurs systèmes peuvent être construits en utilisant des combinaisons de ces tâches. Une tâche générique identifie un type de problème, une méthode pour le résoudre et les types de connaissances utilisées par la méthode. Chaque tâche générique est donc caractérisée par les informations suivantes:

- **Le type de problème.** Quelle est la fonction de la tâche générique?
- **La méthode.** Comment la méthode opère-t-elle sur les connaissances pour accomplir la fonction de la tâche générique?
- **Les caractéristiques des rôles des connaissances.** Comment les connaissances sont-elles organisées pour accomplir la fonction de la tâche générique?

Dans cette perspective, le but de l'acquisition de connaissances est d'associer à un problème un groupe de tâches génériques qui peuvent accomplir la fonction désirée. Deux exemples de tâches génériques sont les suivants: la classification hiérarchique et la synthèse d'un objet par sélection d'un plan.

La classification hiérarchique

- **Le type de problème.** Étant donné une description d'une situation, déterminer quelles catégories ou hypothèses s'appliquent à la situation.
- **La méthode.** Quand une hypothèse est confirmée ou probable, ses sous-hypothèses doivent être considérées. Si une hypothèse est refusée, alors ses sous-hypothèses doivent être refusées aussi.
- **Les caractéristiques des rôles des connaissances.** Les hypothèses sont organisées en une hiérarchie de classification dans laquelle les nœuds enfants représentent les sous-hypothèses du nœud père.

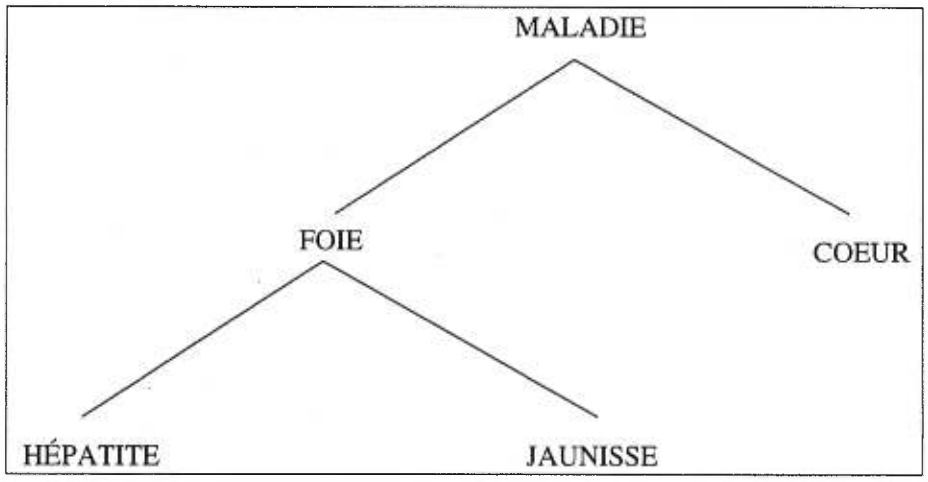

FIG. 2.1 - *Fragment d'une hiérarchie de maladies pour la classification hiérarchique (adapté de [Chandrasekaran, 1986]). Dans le haut de la hiérarchie se trouvent les maladies plus générales. La racine est "maladie" indiquant que la méthode a établi que les observations obtenues du patient indiquent qu'il est malade. En descendant la hiérarchie, les maladies sont classifiées en différents types, comme une maladie du foie ou du cœur. Par la hiérarchie, nous pouvons remarquer que la méthode a établi que le patient a une maladie du foie, plutôt qu'une maladie du cœur. Puis, la maladie du foie est précisée en maladies plus spécifiques, comme l'hépatite et le jaunisse.*

Le diagnostic médical peut être réalisé par une application de la classification hiérarchique, où la description des symptômes d'un patient est associée à un nœud

dans une hiérarchie de maladies. La stratégie de contrôle de la méthode est descendante ("top-down"), où les concepts plus généraux sont en haut de la hiérarchie. Chaque concept de la hiérarchie est capable de décider lui-même, s'il est accepté ou refusé. La méthode est nommée "establish-refine", parce que chaque concept essaie de s'établir lui-même. Si le concept réussit à s'établir, c'est-à-dire, à couvrir les observations, le processus de raffinement est invoqué. Ce processus consiste à vérifier quels nœuds descendants peuvent s'établir eux-mêmes, jusqu'à ce que des nœuds feuilles couvrent tous les symptômes. La figure 2.1 illustre la hiérarchie de maladies pour la classification hiérarchique.

La synthèse d'un objet par sélection d'un plan

- **Le type de problème.** Conception d'un objet en satisfaisant des spécifications.
- **La méthode.** Pour concevoir un objet, le choix d'un plan prescrit la sélection des sous-objets dans un certain ordre.
- **Les caractéristiques des rôles des connaissances.** L'objet est représenté par une hiérarchie des composants dans laquelle les nœuds enfants représentent les composants du nœud père.

Un exemple de tâche générique de synthèse est la conception routinière, un type de problème où la structure de l'objet à concevoir est déjà connue et ne doit pas être inventée. Dans ce cas, la tâche de conception consiste à associer les dimensions et les matériaux à l'objet. Des plans de conception sont disponibles pour aider à faire le choix des composants et sous-composants. Ces connaissances sont organisées dans une hiérarchie de plans qui reflètent la hiérarchie de la structure de l'objet.

Structure de la tâche

La *structure de la tâche* [Chandrasekaran et al., 1992] permet d'associer une tâche et différentes méthodes d'une façon modulaire et flexible, en rendant facile différentes combinaisons de méthodes pour accomplir une tâche. Une tâche est décomposée en

méthodes alternatives qui, à leur tour, sont décomposées en sous-tâches. Cette structure est décomposée récursivement jusqu'à ce que dans les feuilles de la hiérarchie se trouvent des méthodes qui ne sont plus décomposables. La figure 2.2 illustre l'organisation d'une structure de tâche.

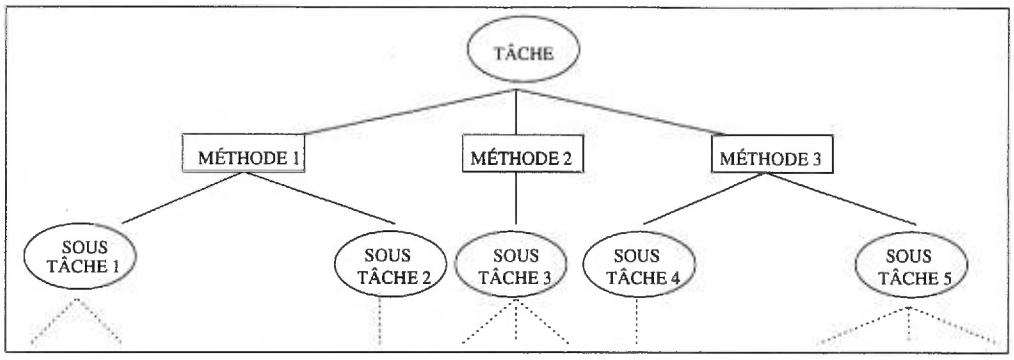

FIG. 2.2 - *Structure de la tâche. Les cercles représentent les tâches, les rectangles représentent les méthodes. Une tâche peut être accomplie par différentes méthodes, donc la relation entre tâche et méthode est du type "ou". Une méthode consiste en plusieurs sous-tâches, donc la relation entre une méthode et ses sous-tâches est du type "et".*

L'avantage de la structure de la tâche est d'avoir plus de flexibilité et de modularité dans la définition d'une tâche et, par conséquent, dans la réutilisation de méthodes. Cependant, le problème de cette approche est le niveau de description des méthodes et de la structure de la tâche. À chaque méthode est associée une représentation particulière de rôles des connaissances, mais cette représentation n'est pas explicitée d'une façon plus détaillée.

Dans la prochaine section, nous montrons la Classification Heuristique qui associe également une méthode à une organisation particulière des connaissances.

2.1.2 Classification Heuristique

Comme nous l'avons vu au chapitre 1, Clancey a construit NEOMYCIN à partir de MYCIN. En décrivant en termes abstraits la stratégie de raisonnement de NEOMYCIN, Clancey a remarqué que, non seulement ce système, mais plusieurs autres

SBCs possédaient la même structure d'inférence. Cette structure d'inférence peut être utilisée pour exécuter les tâches de diagnostic, de sélection de catalogues ou de planification.

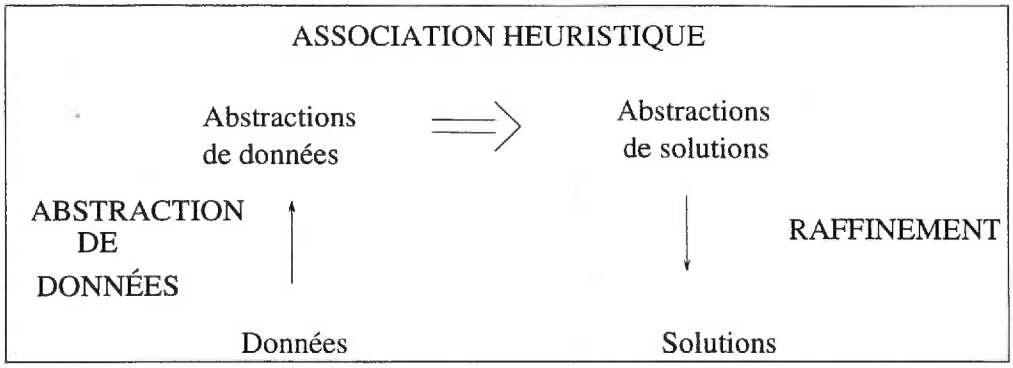

FIG. 2.3 - *Classification Heuristique (adapté de [Clancey, 1985]). Le dessin représente un modèle d'inférence selon un haut niveau d'abstraction. Ce modèle est sous-jacent à plusieurs systèmes à base de connaissances.*

La généralisation de cette stratégie est la méthode Classification Heuristique [Clancey, 1985] qui comprend trois phases (figure 2.3):

1. Abstraction des données dans une première hiérarchie.
2. Association heuristique avec une deuxième hiérarchie de solutions pré-énumérées.
3. Raffinement de cette deuxième hiérarchie.

L'abstraction des données dans la première hiérarchie filtre les données d'entrée de façon à générer des données de sortie plus simples, dans le but d'obtenir les caractéristiques plus pertinentes du problème. NEOMYCIN, par exemple, fait abstraction des observations sur un patient dans une classification hiérarchique des symptômes. Dans la figure 2.4, de l'abstraction de la donnée d'entrée "Le patient est un adulte et le taux de globules blancs est inférieur à 2.5", on obtient la donnée "Le taux de globule blancs du patient est bas".

Le nom "Classification Heuristique" est dû à la deuxième phase de la méthode. L'association entre les deux hiérarchies est une relation heuristique. Une relation

heuristique caractérise un type de connaissance dérivée de l'expérience, empirique et incertaine. Ces relations heuristiques correspondent aux règles heuristiques trouvées dans la plupart des SBCs. Les abstractions de la première hiérarchie sont associées à une autre classification hiérarchique des maladies. Dans l'exemple de la figure 2.4, la relation heuristique "Hôte compromis" => "Infection Gram Négative" fait l'association entre les deux hiérarchies.

Le raffinement dans la deuxième hiérarchie fournit une caractérisation plus précise de la solution. Dans notre exemple de la figure 2.4, la solution "'Infection Gram Négative" est raffinée de façon à caractériser plus précisément la maladie "Infection E.coli".

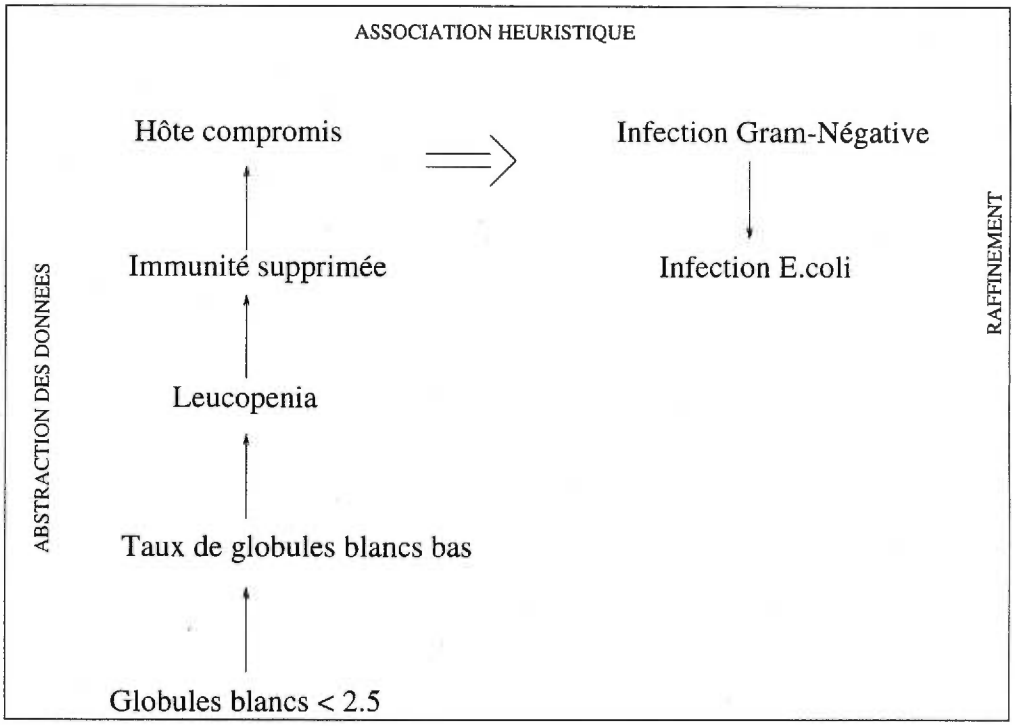

FIG. 2.4 - *Exemple de la structure d'inférence NEOMYCIN (adapté de [Clancey, 1985]). La première phase abstrait les caractéristiques générales des observations sur le patient. La deuxième phase associe les observations abstraites à une maladie générale. La troisième phase raffine la solution générale, en obtenant un diagnostic plus spécifique.*

L'idée de la Classification Heuristique est similaire à l'approche de tâches géné-

riques vue dans la section précédente: définir une méthode de résolution de problèmes réutilisable, en définissant pour chaque méthode les rôles des connaissances dans le processus d'inférence. Dans le cas de la Classification Heuristique, les trois phases de la méthode sont associées à trois structures de connaissances respectives: la phase d'abstraction exige une hiérarchie dans laquelle les données seront abstraites; la phase d'association heuristique exige des relations causales entre les données et les explications pour les données; la phase de raffinement exige une deuxième hiérarchie pour préciser la solution. Ces différentes structures de connaissances définissent les différents rôles joués par les connaissances dans la résolution du problème.

La différence par rapport aux tâches génériques se situe au niveau de granularité de la définition. Par exemple, la partie d'abstraction de la Classification Heuristique correspond à une seule tâche générique. Cela veut dire que les tâches génériques constituent des composants réutilisables plus petits, tandis que la Classification Heuristique est un composant réutilisable plus complexe constitué de plusieurs tâches génériques.

L'importance de la Classification Heuristique est d'avoir montré qu'il y a une structure sous-jacente aux règles de production des SBCs de première génération. L'explicitation de cette structure est importante en ce qui concerne la maintenance, l'explication et l'acquisition de connaissances. De plus, la structure d'inférence permet de comparer divers systèmes, en dévoilant leurs similarités, même si les domaines et les applications sont différents. Enfin, en faisant abstraction des détails des connaissances du domaine, la Classification Heuristique fait ressortir les rôles des connaissances dans le processus d'inférence.

L'inconvénient de la Classification Heuristique est le même que celui des tâches génériques, soit le niveau d'abstraction de sa description. La méthode est décrite en termes trop abstraits (graphiques et informels) ou en fonction du code implanté (les règles de production des SBCs). Ainsi, la Classification Heuristique ne distingue pas les rôles des connaissances ou les détails sur les étapes d'inférence. Il serait donc intéressant de mieux préciser la méthode.

Dans la section suivante, nous analysons une autre approche d'identification des méthodes de résolution de problèmes: les méthodes à limitation de rôles.

2.1.3 Méthodes à limitation de rôles

Le travail de McDermott [McDermott, 1988] consiste à identifier et à décrire les méthodes de résolution de problèmes les plus communes trouvées dans les SBCs, dans le but de créer une taxonomie de ces méthodes. Les méthodes identifiées par McDermott sont appelées *méthodes à limitation de rôles* parce qu'elles clarifient les rôles joués par les connaissances dans la recherche de la solution.

L'idée sous-jacente de cette approche est qu'une délimitation plus précise du problème simplifie l'identification des connaissances nécessaires pour accomplir la tâche. La compréhension des rôles des connaissances facilite l'interrogation des experts et l'identification des connaissances manquantes dans la base de connaissances. Comme dans les tâches génériques et la Classification Heuristique, les méthodes définissent les rôles joués par les connaissances dans la résolution du problème et caractérisent ces rôles dans une représentation particulière.

Ce qui est particulièrement intéressant dans ce travail est la représentation explicite des méthodes de résolution de problèmes dans des outils d'acquisition de connaissances. Ces outils dirigent la collecte et la codification des connaissances: ils explicitent l'information que la méthode requiert pour résoudre le problème et ils génèrent la base de connaissances. Les outils d'acquisition de connaissances spécialisés par méthode de résolution de problèmes facilitent l'obtention des connaissances: les questions posées sont adaptées au problème et non au formalisme de représentation des connaissances, comme dans les systèmes de première génération. De cette façon, le cogniticien et l'expert peuvent s'abstraire des détails d'implantation et se concentrer sur les détails du problème.

Ici, nous nous restreignons à décrire deux méthodes à limitation de rôles: *Couvrir et Différencier* et *Proposer et Réviser.*

***Couvrir et Différencier* (C&D)** [Eshelman, 1988] est une méthode pour résoudre un problème de classification. Les tâches qui peuvent être accomplies par

C&D doivent satisfaire les suppositions suivantes:

- Il existe un groupe identifiable de symptômes, chacun pouvant être expliqué.
- Il existe un ensemble d'explications possibles qui justifient ("couvrent") les symptômes.
- Il existe des informations pour différencier les explications possibles pour chaque symptôme.
- Il existe une seule explication pour chaque symptôme.

Cette méthode comprend deux phases: couvrir et différencier. En distinguant les connaissances de ces deux phases, la base de connaissances est plus facilement construite et raffinée, parce que les connaissances nécessaires pour la résolution du problème sont bien délimitées. Ainsi, dans C&D, les rôles joués par les connaissances sont de deux types:

- Des explications candidates pour les symptômes.
- Des informations pour différencier entre les explications candidates.

Dans un système qui utilise la méthode C&D, l'utilisateur doit fournir les symptômes. Ensuite, le système sélectionne les explications possibles pour les symptômes et il pose des questions à l'utilisateur de façon à différencier ces explications. La différenciation des connaissances est réalisée par la recherche d'informations de façon à éliminer ou confirmer chaque explication. Le système répète les étapes de génération d'explications ("couvrir") et de distinction d' explications ("différencier") jusqu'à l'obtention d'une seule explication. C&D utilise deux principes de base: l'exhaustivité (chaque symptôme doit être expliqué) et l'exclusivité (une seule explication est préférée).

Les rôles des connaissances de C&D sont définis par une représentation de connaissances particulière: un réseau causal, où les arcs représentent des relations d'explication entre les nœuds. Ce réseau permet de couvrir les symptômes observés, en générant

des explications possibles pour ces symptômes et de différencier ces explications dans le but de trouver une explication finale.

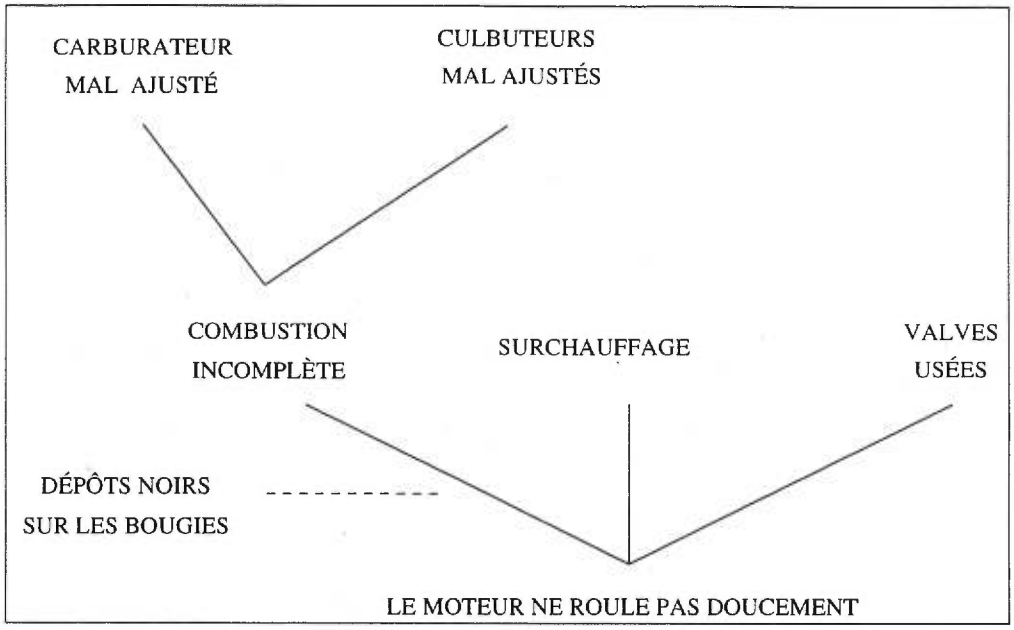

FIG. 2.5 - *Représentation des connaissances du domaine pour la méthode Couvrir et Différencier (adapté de [McDermott, 1988]). Ici, le réseau causal de C&D est instancié avec le domaine mécanique et la méthode est utilisée pour diagnostiquer des problèmes de mécanique automobile. Les lignes continues représentent les relations d'explication: les nœuds inférieurs sont expliqués par les nœuds supérieurs. Les lignes pointillées représentent les informations additionnelles utilisées pour différencier les explications possibles associées à un nœud.*

La figure 2.5 illustre une partie des connaissances du domaine pour diagnostiquer des problèmes de mécanique automobile en utilisant C&D. Dans ce réseau, les nœuds intérieurs représentent des explications intermédiaires. Ils expliquent les nœuds inférieurs et ils sont expliqués par les nœuds supérieurs. Initialement, le système demande à l'utilisateur les symptômes du mauvais fonctionnement de la voiture et l'utilisateur répond "le moteur ne roule pas doucement". Ensuite, le système couvre les explications possibles pour ce symptôme, par exemple, "combustion incomplète", "surchauffage" et "valves usées". Puis, le système pose des questions à l'utilisateur dans le but de différencier ces explications. Par exemple, "combustion incomplète" est signalée

par "dépôts noirs sur les bougies", qui permet de différencier l'explication "combustion incomplète" comme celle la plus probable pour l'état "le moteur ne roule pas doucement". Ainsi, si la "combustion incomplète" est l'explication trouvée pour le symptôme "le moteur ne roule pas doucement", des explications pour la "combustion incomplète" sont sélectionnées: "carburateur mal ajusté" et "culbuteurs mal ajustés". Puis le système pose des questions pour différencier ces explications et ainsi de suite, jusqu'à ce qu'il trouve une seule explication pour le symptôme.

Une méthode de classification comme C&D peut être utilisée pour résoudre des problèmes d'évaluation, de conception, de planification, etc, en plus du problème de diagnostic. Nous avons modélisé et implanté *Couvrir et Différencier* et nous présentons les résultats de ce travail dans le chapitre 6.

***Proposer et Réviser* (P&R)** [Marcus, 1988] est une méthode pour résoudre un problème de configuration. Les tâches qui peuvent être accomplies par P&R doivent satisfaire les suppositions suivantes:

- Il existe un groupe de paramètres auxquels il faut associer une valeur.
- Il existe, pour chaque paramètre, une formule qui permet de calculer sa valeur initiale.
- Il existe des contraintes qui limitent les valeurs de certains paramètres.
- Il existe des procédures de réparation qui permettent de recalculer les valeurs des paramètres dans le cas de violations des contraintes.

Dans P&R, les rôles joués par les connaissances sont associés aux trois types de connaissances suivants:

- Des procédures pour attribuer des valeurs aux paramètres.
- Des contraintes pour restreindre les valeurs des paramètres.
- Des procédures pour corriger les valeurs de paramètres quand des contraintes sont violées.

On remarque ici que les descriptions des suppositions préalables pour l'utilisation de la méthode et les caractéristiques des rôles des connaissances sont décrites informellement en langage naturel. On ne sait pas combien de procédures peuvent être associées à chaque paramètres ou s'il y a plus d'une procédure de réparation associée à chaque contrainte.

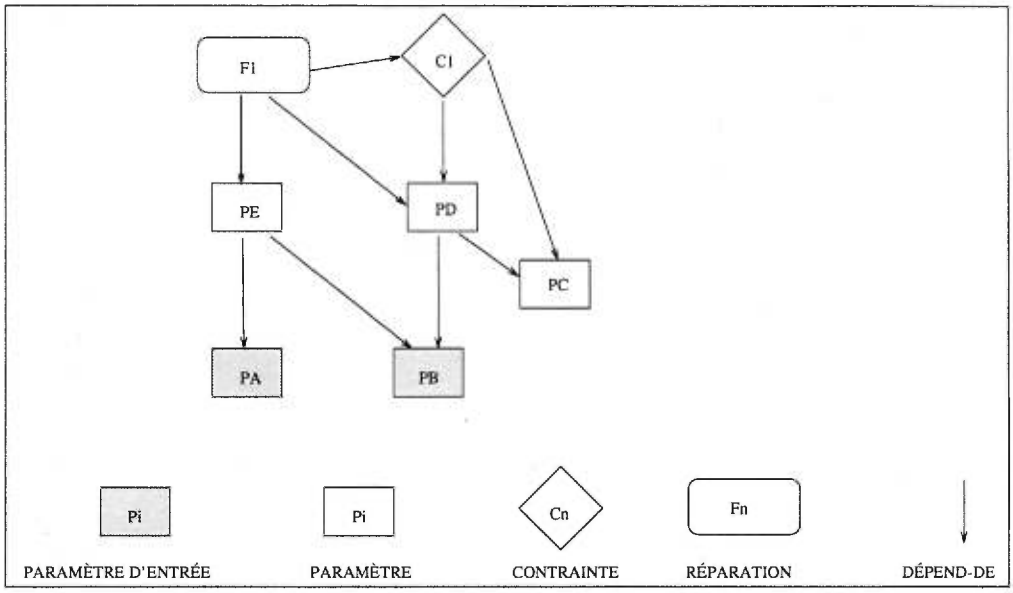

FIG. 2.6 - *Réseau de dépendances de Proposer et Réviser. Ici, les paramètres PA et PB sont des paramètres dont les valeurs sont fournies par l'utilisateur. Le paramètre PE est calculé en utilisant les valeur des paramètres PA et PB. Le paramètre PD est calculé en utilisant les valeurs de paramètres PB et PC. La contrainte C1 est évaluée en utilisant les valeurs de paramètres PD et PC. La réparation F1 est appliquée en utilisant les valeurs des paramètres PE et PD pour corriger la violation de la contrainte C1.*

Comme ceux de C&D, les rôles des connaissances de P&R sont définis en termes d'une représentation des connaissances particulière: un réseau de dépendances. La figure 2.6 illustre un réseau de dépendances de P&R. Il y a trois types de nœuds: les paramètres, les contraintes et les réparations. Parmi les paramètres, on distingue les paramètres d'entrée des autres paramètres. Les paramètres d'entrée sont les paramètres dont les valeurs initiales sont proposées d'avance par l'utilisateur et sont utilisées comme point de départ pour le calcul d'autres paramètres. Les arcs indiquent

les relations de dépendances entre les noœuds.

P&R a été implantée pour la tâche de conception d'un ascenseur, tâche connue par le nom de VT ("vertical transportation"). Ce problème est bien connu dans le domaine de l'acquisition de connaissances, parce qu'il a été utilisé comme test pour évaluer plusieurs systèmes et méthodologies de développement des SBCs [Gaines, 1996]. Dans cette thèse, nous avons étudié aussi ce problème et nous en parlons au chapitre 5.

Le travail de McDermott tente d'établir une taxonomie de méthodes de résolution de problèmes. La délimitation de rôles des connaissances optimise la construction des bases de connaissances en identifiant exactement quelles sont les connaissances nécessaires pour résoudre un problème spécifique. Ce qui est particulièrement intéressant dans cette approche est la construction d'outils d'acquisition de connaissances spécialisés par méthode. Cela rend possible la réutilisation des méthodes dans plusieurs SBCs pour des domaines différents.

Ce travail est loin d'être une classification complète des méthodes de résolution de problèmes. Il serait souhaitable aussi d'avoir une description plus formelle de chaque méthode, de façon à les comparer et à éliminer les ambiguïtés d'une description informelle. Par exemple, dans [Eshelman, 1988], la méthode C&D a été considérée comme un cas particulier de Classification Heuristique. Pourtant, dans [Schreiber et al., 1993a], une description plus formelle de cette méthode laisse un certaine doute sur cette affirmation: C&D ne contient pas la première étape d'abstraction de la Classification Heuristique.

Une autre conséquence d'une description informelle est l'impossibilité d'identifier des composants communs entre les méthodes. Les méthodes ne peuvent résoudre que des problèmes spécifiques d'une façon très rigide. Il est difficile de les adapter à d'autres problèmes ou d'identifier des composants réutilisables de façon à faciliter la description des nouvelles méthodes. Ainsi, les méthodes à limitation de rôles n'offrent pas de composants modulaires qui permettraient de les configurer pour des problèmes légèrement différents.

La description des conditions préalables (suppositions) d'utilisation associées aux méthodes et la description des rôles des connaissances offrent une vision générale de

la méthode, mais elles ne précisent pas en détail ces conditions et ces caractéristiques. Pour en savoir plus, le cogniticien est obligé de fouiller le code implanté et vérifier si la méthode est applicable à son problème. Par conséquent, il est difficile de décider si une application spécifique peut utiliser une méthode. Ainsi, les méthodes à limitation de rôles présentent les mêmes désavantages que les tâches génériques et que la Classification Heuristique en ce qui concerne la description de rôles des connaissances.

2.2 Réutilisation par les connaissances du domaine

Pour favoriser la réutilisation des connaissances du domaine, l'idéal est de construire des ontologies indépendamment de leur utilisation. Dans ce cas, il est hors de question de représenter des connaissances en fonction des méthodes de résolution des problèmes qui les utilise, selon l'approche décrite à la section précédente. Ainsi, la construction d'ontologies serait incompatible avec la réutilisation par les connaissances de contrôle.

Dans cette section, nous traitons le problème de réutilisation par une voie différente, en montrant les avantages des ontologies. Dans la section 1.1.4, nous avons introduit le concept d'ontologie comme une spécification d'une conceptualisation. Ainsi, une ontologie est un ensemble de définitions de termes avec la description de leur caractéristiques structurales et les relations entre ces termes; Les engagements ontologiques sont des contraintes caractérisant l'ontologie de façon à restreindre la signification des termes. Ils expriment les suppositions qui doivent être vraies pour utiliser une ontologie.

Une ontologie diffère d'une base de connaissances construite dans un but spécifique dans le cadre d'une application, et prenant en compte les détails de l'application et l'efficacité pour l'obtention des résultats. Une ontologie représente plutôt le vocabulaire et la structure de cette base de connaissance. Ainsi, une ontologie est associée à une base de connaissances, de la même façon que les schémas conceptuels sont associés aux bases de données.

Les ontologies offrent les bénéfices suivants:

- Donner accès à de grandes quantités de connaissances.
- Formaliser et valider les connaissances une seule fois, mais les réutiliser plusieurs fois.
- Construire plus rapidement des systèmes fiables et robustes.
- Servir comme dépôts d'information qui seront utilisés comme référence pour représenter un consensus par rapport à l'utilisation des termes d'un domaine.
- Avoir un niveau d'analyse des conceptualisations et des engagements ontologiques des bases de connaissances.
- Aider l'acquisition de connaissances et supporter la conception du système informatique [Van Heijst, 1995].

Les ontologies pourront donc être utiles pour:

- Des applications en intelligence artificielle. La plupart de recherches dans ce domaine utilisent des exemples "jouets" qui ne considèrent pas des problèmes complexes en termes d'espace et de temps. Des ontologies contenant de grandes quantités de connaissances pourront être utilisées pour valider ces recherches.
- Des applications qui ont besoin de larges connaissances dans des domaines scientifiques. Quelques disciplines scientifiques sont en train de ramasser une quantité énorme de connaissances dont la difficulté d'accès devient de plus en plus un problème. Les ontologies pourront servir à organiser et rendre ces informations disponibles.
- Des applications qui nécessitent des connaissances générales sur le monde ("background knowledge, common sense knowledge"). Ces connaissances sont applicables à la majorité des domaines et des problèmes, parce qu'elles sont d'utilité

commune. Il y a des applications qui nécessitent de grandes quantités de connaissances de ce type, comme les systèmes de traitement de langue naturelle. Le projet Cyc [Lenat et Guha, 1990], par exemple, a comme but la construction de bases de connaissances avec de grandes quantités de connaissances générales.

- Des applications qui ont besoin de conceptualisations différentes de connaissances dans un même domaine ou dans plusieurs domaines. Sans la réutilisation de connaissances, chaque base de connaissances doit être construite séparément. De plus, ces systèmes n'ont que des connaissances pour un seul domaine ou une seule vision de ce domaine. Si ces systèmes ont besoin de connaissances pour résoudre des problèmes qui sont en dehors du domaine d'application ou un peu plus complexes, la performance de ces systèmes décroît. L'utilisation de plusieurs ontologies peut rendre ces systèmes plus robustes.

- Des applications qui ont besoin d'établir un consensus dans une communauté de personnes ou d'agents. Par exemple, plusieurs entreprises tentent de se constituer une "mémoire" ("corporate memory"). Les ontologies pourraient leur servir, en intégrant des connaissances qui sont dispersées.

Les définitions d'une ontologie expriment différents types de conceptualisation. Ainsi, les ontologies peuvent être classifiées selon le sujet de leur conceptualisation [Van Heijst et al., 1997]:

- Les *ontologies de représentation ou de base* explicitent les conceptualisations des formalismes de représentation de connaissances. Elles sont utilisées pour définir les ontologies des autres types. Nous verrons dans cette thèse le Frame Ontology (section 2.2.1) et l'ontologie de base de KADS (section 3.1.1), basée sur les primitives épistémologiques de KL-ONE.

- Les *ontologies du domaine* explicitent les conceptualisations d'un domaine spécifique. Par exemple, une ontologie pour le domaine médical ou pour des données bibliographiques.

- Les *ontologies génériques* explicitent les conceptualisations des modèles qui peuvent être réutilisées dans différentes domaines et tâches. Par exemple, des ontologies pour un modèle causal, structural ou une représentation temporelle des événements.

- Les *ontologies d'application* modélisent les connaissances d'une application particulière. Elles sont en général obtenues à partir des ontologies du domaine et des ontologies génériques. En général, ces ontologies ne sont pas réutilisables. Cette dénomination est utilisée par PROTÉGÉ-II (section 2.3), pour les ontologies construites pour une application spécifique.

Dans cette thèse, nous ajoutons à cette classification les *ontologies de méthodes* qui explicitent les conceptualisations sous-jacentes aux méthodes de résolution de problèmes. Nous approfondissons les ontologies de méthodes dans le chapitre 5.

Les langages pour décrire des ontologies doivent être assez expressifs pour exprimer les engagements ontologiques. CML est le langage de la méthodologie KADS pour la définition d'ontologies. Dans nos travaux, nous avons choisi de représenter nos ontologies avec Loom (chapitre 4). Dans la section suivante, nous présentons Ontolingua, un système créé pour définir des ontologies réutilisables au niveau des connaissances.

2.2.1 Ontolingua

Le projet DARPA-ROME [Neches et al., 1991] exploite la représentation d'ontologies indépendantes des programmes d'application. Ontolingua [Gruber, 1992] est un outil développé dans le cadre de ce projet pour construire des ontologies réutilisables et portables dans différents langages.

L'idée de base d'Ontolingua est de définir une bibliothèque d'ontologies contenant des connaissances déclaratives définies selon le niveau des connaissances; ainsi les ontologies sont traduites en différents formalismes de représentation de connaissances, tels que les "frames", la logique de prédicats, les langages terminologiques, etc. Le traducteur Ontolingua est illustré dans la figure 2.7.

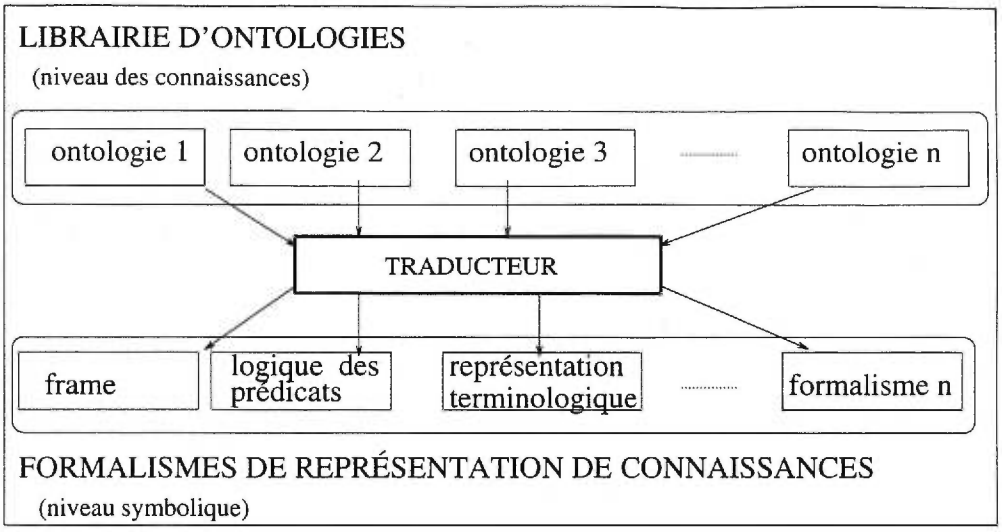

FIG. 2.7 - *Traduction des ontologies en différents formalismes de représentation de connaissances. Une même ontologie peut être utilisée dans différentes applications avec différents formalismes de représentation de connaissances.*

Comme langage de représentation, Ontolingua utilise KIF et Frame Ontology pour définir les ontologies. KIF est une interlingua et Frame Ontology est une ontologie de représentation.

KIF

La syntaxe et la sémantique des définitions en Ontolingua sont écrites en KIF ("Knowledge Interchange Format") [Genesereth et Fikes, 1992]. KIF est une interlingua, c'est-à-dire un langage pour la publication et communication des connaissances. Le pouvoir d'expression de KIF est considérable, mais KIF n'est pas une implantation d'un formalisme de représentation.

KIF est une extension du calcul de prédicat de premier ordre avec une notation dans un style LISP. L'idée de KIF est d'avoir une syntaxe commune pour représenter des expressions en logique de premier ordre, ainsi que quelques extensions de cette logique telles que le raisonnement par défaut, les méta-connaissances, etc. KIF présente une sémantique déclarative.

Les phrases en KIF sont formées par de listes dont le premier élément est une

relation et les autres sont des termes ou des opérations logiques sur les listes. Les variables quantifiées universellement ou existentiellement sont marquées par le préfixe "?". Dans la Figure 2.8, nous montrons la notation KIF correspondant à la phrase "`Tous les écrivains sont mal compris par un certain lecteur`".

```
(forall ?W  (=> (ecrivain ?W)
                (exists (?R ?D)
                        (and (lecteur ?R)
                             (document ?D)
                             (ecrire ?W  ?D)
                             (lire ?R ?D)
                             (not (comprendre ?R ?D))))))
```

FIG. 2.8 - *Représentation en KIF de la phrase: "`Tous les écrivains sont mal compris par un certain lecteur`" (adapté de [Gruber, 1993a]).*

Exemple d'une définition en Ontolingua

Les ontologies sont constituées d'un ensemble de définitions de concepts, de relations, d'objets et d'axiomes. En Ontolingua, par exemple, les concepts sont définis par leur nom et leurs caractéristiques. La figure 2.9 présente un exemple d'une définition de classe en Ontolingua pour une ontologie d'information bibliographique. La phrase `:def` établit les conditions nécessaires pour appartenir à la classe *écrivain*: (1) les écrivains doivent être des personnes; (2) les écrivains doivent avoir un seul nom défini; (3) le nom de l'écrivain doit être une instance de la classe `biblio-nom`; (4) l'écrivain doit avoir écrit au moins un document; et (5) le nom de l'écrivain est son vrai nom et non un pseudonyme.

Frame Ontology

Comme nous avons vu, KIF est un langage qui permet de représenter des connaissances selon la logique de premier ordre. Cependant, KIF ne permet pas l'organisation des termes en une hiérarchie de classes. Cela est offert par le Frame Ontology.

En Ontolingua, le Frame Ontology est un ensemble de primitives pour la définition

```
(define-class Ecrivain (?ecrivain)
  ''Un ecrivain est une personne. Il a un seul nom. Il  doit avoir
    ecrit au moins un document. Son nom est du type biblio-nom.
    Un ecrivain, dans cette ontologie, est connu par son vrai nom.''
  :def (and (personne ?ecrivain)
            (= (value-cardinality ?ecrivain ECRIVAIN.NOM) 1)
            (value-type ?ecrivain ECRIVAIN.NOM biblio-nom)
            (>= (value-cardinality ?ecrivain ECRIVAIN.DOCUMENTS) 1)
            (<=> (ecrivain.nom ?ecrivain ?nom)
                 (personne.nom ?ecrivain ?nom))))
```

FIG. 2.9 - *Exemple d'une définition en Ontolingua (adapté de [Gruber, 1993a]). Le concept est défini par son nom, une variable d'instance (`?ecrivain`), un commentaire qui explique le concept et, après le mot clé `:def`, une phrase KIF restreint l'utilisation du terme en déterminant sa sémantique.*

de classes et de relations et pour l'organisation des connaissances en hiérarchies avec héritage. Cette ontologie fonctionne comme une extension de KIF dont le but est de saisir les caractéristiques communes des formalismes du type "frames" et orientés objets. Frame Ontology est une ontologie du type de base ou de représentation, car elle est utilisée pour définir les autres ontologies.

Frame Ontology offre un ensemble de relations de deuxième ordre pour spécifier les définitions. Dans la figure 2.9, l'écrivain a été défini comme une sous-classe de personne par l'utilisation d'une variable d'instance:

```
(personne ?x)
```

Une autre possibilité est de faire la même définition avec des relations de deuxième ordre de Frame Ontology:

```
(subclass-of ecrivain personne)
```

La fonction "value-cardinality" utilisée dans la définition d'auteur est un autre exemple d'une relation de Frame Ontology.

Avec l'unification des représentations du type orientée objets (Frame Ontology) et des représentations en calcul du prédicat (KIF), Ontolingua offre un cadre de base

pour la définition d'ontologies. Le couple KIF et Frame Ontology peut être considéré comme un langage de représentation de connaissances spécialisé.

Ce qui est particulièrement intéressant avec Ontolingua est la tentative d'expliciter des ontologies selon le niveau des connaissances pour en faciliter la réutilisation. Cependant, Ontolingua n'aide pas la conception d'ontologies, en décidant quels concepts, relations et axiomes doivent être inclus. Ontolingua n'offre qu'un cadre descriptif pour la représentation de connaissances. De plus, Ontolingua n'établit aucun principe ou structure pour l'organisation de la bibliothèque d'ontologies. Ce problème est fondamental pour la réutilisation des ontologies, en considérant les divers types d'ontologies, des domaines et les grandes quantités de connaissances de chaque domaine. Pour en savoir plus à ce sujet, nous référons aux travaux de [Van Heijst, 1995] et de [Valente et Breuker, 1996].

2.3 Réutilisation par les connaissances du domaine et de contrôle

Nous avons présenté deux approches pour la réutilisation de connaissances, en montrant les avantages respectifs. Comme nous l'avons souligné, ces approches sont contradictoires. L'une revendique que les connaissances doivent être représentées en accord avec leur utilisation. L'autre défend l'idée que les connaissances doivent être représentées indépendamment des applications spécifiques.

Dans cette section, nous présentons une autre approche qui tente de concilier les deux approches en permettant la réutilisation par les connaissances de contrôle et du domaine. Cette approche est l'approche adoptée dans notre thèse.

L'idée est de résoudre le dilemme de la généralité et de la spécificité des connaissances par un compromis entre ces deux approches. Cette solution obtient, potentiellement, le meilleur des deux mondes, en considérant à la fois la définition des connaissances en fonction de la méthode utilisée (connaissances spécifiques) et l'indépendance des connaissances du domaine d'application par rapport à l'application

(connaissances générales).

Cette approche est adoptée par l'environnement PROTÉGÉ-II pour le développement des SBCs à partir des méthodes de résolution de problèmes réutilisables. PROTÉGÉ-II est un environnement inspirée de l'approche de méthodes à limitation de rôles, en générant des outils d'acquisition de connaissances pour les méthodes de résolution de problèmes. La différence par rapport à l'approche de méthodes à limitation de rôles réside dans le fait que les outils générés par PROTÉGÉ-II ne sont pas définis en termes de la méthode, mais en termes de l'application. PROTÉGÉ-II offre aussi plus de flexibilité en ce qui concerne l'instantiation et la configuration des méthodes.

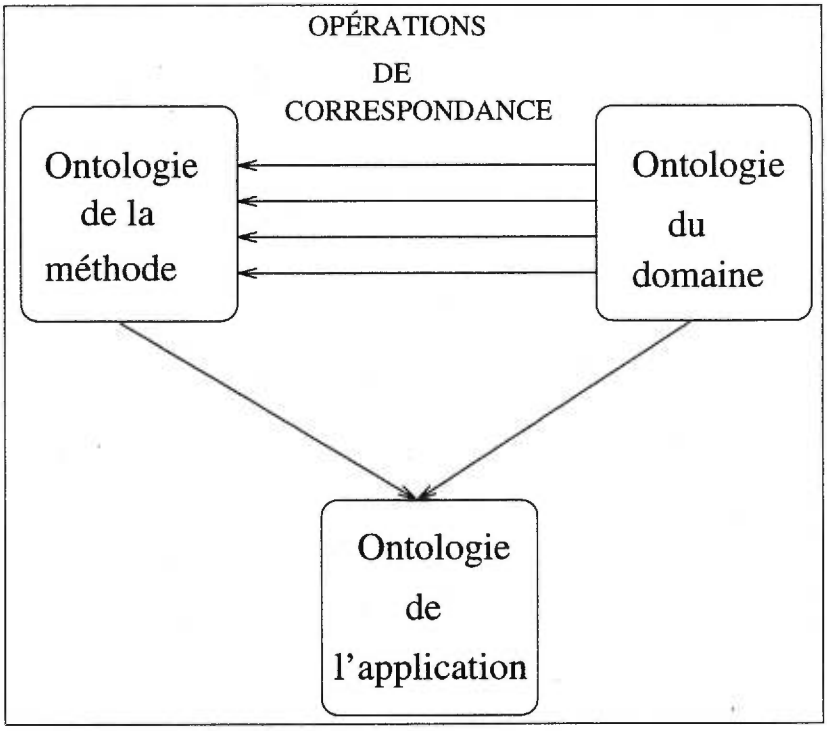

FIG. 2.10 - *Approche de PROTÉGÉ-II. L'ontologie de la méthode décrit les rôles des connaissances en termes de concepts et de relations manipulés par la méthode. L'ontologie du domaine contient les concepts et les relations spécifiques au domaine d'application représentés indépendamment de la méthode. L'ontologie de l'application contient le résultat des opérations de correspondance qui associent les termes de l'ontologie du domaine à l'ontologie de la méthode.*

C'est qui est intéressant dans PROTÉGÉ-II, c'est la possibilité d'avoir à la fois les rôles des connaissances spécifiques de la méthode et les connaissances générales du domaine. La définition d'une méthode et sa configuration pour une application spécifique PROTÉGÉ-II consiste des étapes suivantes: (1) la définition pour chaque méthode d'une *ontologie de la méthode* qui exprime les suppositions sur l'utilisation des connaissances par la méthode, en décrivant les rôles des connaissances; (2) l'existence d'une ontologie du domaine où les connaissances du domaine sont représentées indépendamment de la méthode; (3) l'application des *opérateurs de correspondance* qui adaptent l'ontologie du domaine à l'ontologie de la méthode; (4) la génération d'une troisième ontologie, *l'ontologie de l'application*, contenant le résultat des correspondances entre les deux ontologies.

Ainsi, cette approche permet la réutilisation par les deux types de connaissances: du domaine et de contrôle. La figure 2.10 illustre l'approche de PROTÉGÉ-II. Cette solution représente, à notre avis, un compromis par rapport aux deux approches précédentes:

- Le *problème de l'interaction* est traité par la définition de l'ontologie de la méthode.
- L'indépendance des connaissances du domaine est préservée par l'existence de l'ontologie du domaine.
- Les opérateurs de correspondance permettent d'intégrer les deux ontologies.

Dans le chapitre 5, nous détaillons notre utilisation de l'ontologie de la méthode pour décrire la structure de rôles des connaissances de la méthode et, dans le chapitre 6, nous présentons l'utilisation des opérateurs de correspondance pour adapter une ontologie du domaine à une ontologie de la méthode. De plus, nous y soulignons les différences de notre travail par rapport à l'approche de PROTÉGÉ-II.

2.4 Résumé

Dans ce chapitre, nous avons présenté une revue des principaux travaux qui touchent la réutilisation de connaissances. Nous avons comparé et critiqué ces travaux. Nous avons abordé trois perspectives différentes: la réutilisation à partir des connaissances de contrôle, la réutilisation à partir de connaissances du domaine et la réutilisation par les connaissances de contrôle et du domaine. Nous avons souligné l'incompatibilité entre les deux premières perspectives et nous avons montré que la troisième perspective représente une conciliation par rapport aux deux antérieures. Cette troisième perspective est l'approche adoptée dans notre thèse.

Dans le prochain chapitre, nous présentons KADS, une méthodologie pour le développement des SBCs. Nous nous attardons sur le modèle de l'expertise de KADS, utilisé dans cette thèse pour représenter les méthodes de résolution de problèmes.

Chapitre 3

KADS

Nous avons vu au chapitre précédent des approches offrant des méthodes de résolution de problèmes prédéfinies qui peuvent être réutilisées en autant qu'une application particulière se conforme aux rôles des connaissances de la méthode. Dans ce cas, il suffit d'instancier les rôles des connaissances avec les connaissances de l'application.

Dans ce chapitre, nous présentons la méthodologie KADS pour le développent des SBCs, qui permet la modélisation de différentes méthodes et différents domaines. Le grand avantage de KADS est sa flexibilité pour la modélisation au niveau conceptuel, sans modèles opérationnels préexistants, comme dans les travaux décrits au chapitre 2.

KADS est le fruit de plusieurs travaux dans le cadre de projets Esprit. Un résultat est la méthodologie KADS-I [Schreiber et al., 1993b] qui porte sur les aspects de la modélisation et de l'acquisition de connaissances. Un autre aspect du projet est la méthodologie CommonKADS [Van de Velde, 1994] qui synthétise d'autres méthodologies existantes, principalement KADS-I et *Composants de l'expertise* [Steels, 1990]. Dans CommonKADS on s'est concentré sur la réutilisation de connaissances et les aspects de génie logiciel. KADS est un point de référence pour le développement des SBCs partout au monde et un standard en Europe.

Dans KADS, plusieurs modèles ont été définis pour traiter la complexité du processus d'ingénierie de connaissances, en appliquant la stratégie "diviser pour régner".

Ces modèles permettent, par une abstraction des détails, d'exprimer certains aspects du système à construire. Ces modèles sont: le modèle de l'organisation, le modèle de la tâche, le modèle de l'agent, le modèle de communication, le modèle de l'expertise et le modèle de conception (figure 3.1).

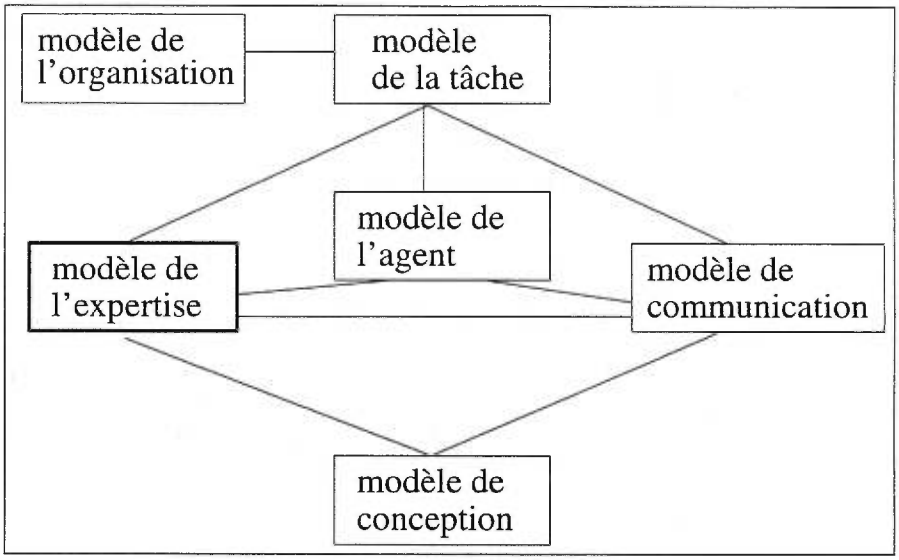

FIG. 3.1 - *Modèles de KADS. Le modèle de l'expertise est un modèle parmi d'autres pour la modélisation d'un SBC. Les lignes dans le dessin indiquent les dépendances directes entre les éléments des modèles.*

Les modèles de l'organisation, de la tâche, de l'agent et de communication tiennent compte du contexte de l'organisation dans lequel le système à base de connaissances est développé. Le modèle de l'expertise s'occupe des connaissances et de la stratégie de raisonnement pour accomplir la tâche. Le modèle de conception décrit le système informatique qui réalise la tâche. Dans cette thèse, nous présentons seulement le modèle de l'expertise. Le modèle de l'expertise est celui qui distingue le développement d'un système à base de connaissances d'un système conventionnel. Nous utilisons ce modèle comme cadre pour la modélisation des méthodes réutilisables de résolution de problèmes.

3.1 Modèle de l'expertise

Le modèle de l'expertise de KADS est basé sur deux principes: (1) le principe de niveau des connaissances (section 1.1.2), selon lequel les SBCs doivent être spécifiés indépendamment du niveau d'implantation; (2) le principe de séparation de types de connaissances (section 1.1.3), selon lequel différentes catégories de connaissances jouent différents rôles dans le processus de résolution du problème.

En plus de distinguer les connaissances du domaine des connaissances de contrôle, le modèle de l'expertise sépare les connaissances de contrôle en deux catégories: inférence et tâche. Ces trois types de connaissances définissent les différents types de connaissances dans le processus de résolution du problème[1]:

- Les connaissances du domaine définissent déclarativement les connaissances pertinentes pour l'application.
- Les inférences définissent les étapes primitives dans le processus d'inférence et les rôles joués par les connaissances du domaine dans le processus d'inférence.
- Les connaissances de tâche spécifient la décomposition des tâches en sous-tâches ou en inférences par la définition de la structure de contrôle qui détermine la combinaison des sous-tâches et des inférences pour accomplir une tâche.

La figure 3.2 illustre le modèle conceptuel de KADS, où les différentes catégories de connaissances sont organisées hiérarchiquement. Dans KADS-I, l'interaction entre les diverses couches était beaucoup plus restreinte que dans CommonKADS, parce que la définition de la couche du domaine était neutre par rapport aux deux autres couches. KADS-I utilisait le *principe de l'interaction limitée*, selon lequel la couche du domaine peut être définie indépendamment des connaissances de contrôle. Contrairement à

1. Dans KADS-I, il y avait une quatrième catégorie de connaissance représentée par la couche de la stratégie. Dans CommonKADS, cette couche est disparue et l'intégration dynamique des trois catégories de connaissances dans le cadre d'une application spécifique est appelée *connaissances stratégiques*. Les connaissances stratégiques décrivent comment les connaissances sont sélectionnées et appliquées dans le but de construire dynamiquement le modèle d'une application. Ces connaissances sont du type téléologique, parce que liées à une application particulière, tandis que les connaissances du domaine, des inférences et de la tâche sont du type épistémologique.

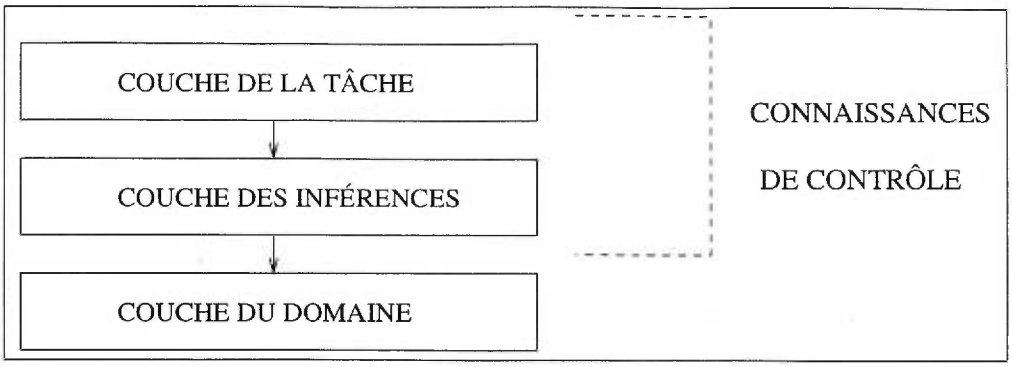

FIG. 3.2 - *Modèle conceptuel de KADS représenté en termes de couches. La couche du domaine représente les connaissances du domaine. Dans la couche des inférences, les inférences doivent être représentées avec les rôles des connaissances associés. La couche de la tâche détermine la séquence d'exécution des sous-tâches et inférences en termes de conditions et d'itérations.*

KADS-I, CommonKADS tient compte du problème de l'interaction (section 2.1), en considérant que les connaissances du domaine et de contrôle sont interdépendantes, c'est-à-dire, les connaissances du domaine ne peuvent pas être définies sans considérer la tâche à accomplir. Cela est défini par le principe de *l'interaction relative* et, dans ce sens, les couches des inférences et de la tâche supposent une organisation particulière des connaissances du domaine.

Dans les prochaines sections, nous détaillons le contenu et les caractéristiques de chaque couche.

3.1.1 Couche du domaine

La couche du domaine est subdivisée en trois types de connaissances: l'ontologie du domaine, un modèle du domaine et un modèle du cas[2].

2. Le modèle du domaine et du cas sont des concepts hérités de la méthodologie *Composants de l'Expertise.*

Ontologie du domaine

L'ontologie du domaine décrit le vocabulaire du domaine par la définition de termes, de leur types et de leur taxonomie. Pour décrire l'ontologie du domaine, KADS utilise un ensemble des primitives basé sur les primitives du langage KL-ONE [Brachman et Schmolze, 1985]. Ainsi, l'ontologie de base de KADS regroupe des primitives telles que les concepts, les instances, les attributs, les valeurs des attributs, les relations, les structures, etc[3]. Nous illustrons plus loin l'utilisation de ces primitives pour définir les connaissances associées au domaine des ascenseurs.

- **Concept.** Les concepts décrivent un ensemble d'objets du domaine. Un concept est identifié par son nom. Par exemple le concept **paramètre** regroupe tous les objets du domaine qui sont des paramètres.

- **Instance.** Les instances sont les objets du domaine. Une instance est une réalisation d'un concept. Par exemple, **type-d'ouverture-de-la-porte-de-l'ascenseur** est une instance du concept **paramètre**.

- **Attribut.** Les concepts ont des attributs. Les attributs définissent les propriétés ou les caractéristiques des concepts. Par exemple, un **paramètre** a l'attribut **paramètre-valeur**.

- **Valeur d'un attribut.** Les attributs ont des valeurs associées. La valeur de l'attribut **paramètre-valeur** de l'instance **type-d'ouverture-de-la-porte-de-l'ascenseur** est **ouverture-laterale**.

- **Relation.** Les relations expriment des associations entre les concepts ou les instances. Par exemple, la contrainte **poids-maximal** restreint la valeur du paramètre **nombre-de-passagers**.

- **Relation de valeur sur les attributs.** Les relations de valeur sur les attributs restreignent les valeurs des attributs. Par exemple, le paramètre **modèle-de-la-**

3. KADS offre seulement un ensemble de primitives pour définir la couche du domaine. Il ne met pas à la disposition des ontologies prédéfinies comme Ontolingua.

machine peut avoir comme valeur **18**, **28** ou **38**.

- **Structure** Les structures représentent des objets complexes. Par exemple, un système d'ascenseur peut être vu comme une structure constituée de plusieurs composants et des relations entre eux.

Modèle du domaine

Le modèle du domaine définit une vision particulière de l'ontologie du domaine. Dans ce sens, le modèle du domaine est un sous-ensemble de l'ontologie du domaine plus quelques définitions spécifiques à cette vision. Par exemple, un modèle causal, un modèle structural (division hiérarchique des composants et sous-composants), etc. C'est donc le modèle du domaine qui tient compte de l'interaction entre la couche du domaine et les autres couches, par la définition d'une interprétation particulière de l'ontologie du domaine. Le modèle du domaine restreint les connaissances du domaine à une utilisation spécifique.

Dans le système d'ascenseur, le modèle du domaine consiste, par exemple, à établir un réseau de dépendances entre les paramètres. Cette relation de dépendance entre les paramètres définit l'ordre de calcul des paramètres.

Modèle du cas

Le modèle du cas exprime les connaissances spécifiques à une situation. Ce modèle dynamique est changé pendant le processus de résolution du problème, tandis que l'ontologie du domaine et le modèle du domaine sont statiques. Le concept de modèle du cas de KADS ressemble au concept de modèle spécifique de situation (SSM - "situation-specific model") [Clancey, 1992], selon lequel un modèle est créé pour les données spécifiques à chaque problème.

Pour le cas de l'ascenseur, le modèle du cas correspond aux valeurs associées aux paramètres dans le cadre d'une situation spécifique. Ces valeurs peuvent être fournies par l'utilisateur, être affectées par le système ou peuvent être modifiées au cours du processus d'inférence pour éviter les violations des contraintes associées à ces valeurs.

C'est dans la définition du modèle du domaine que l'on retrouve le principe d'interaction relative: c'est lui qui donne une vision particulière aux connaissances du domaine. Ceci est notre principale critique du modèle de l'expertise de KADS, car selon nous, cette interprétation de l'ontologie du domaine devrait être plutôt définie dans la couche des inférences. En effet, cette interprétation impose une structure aux connaissances du domaine qui restreint sa réutilisation. La définition de cette vision spécifique des connaissances devrait correspondre plutôt à la définition des rôles des connaissances dans la couche des inférences. Cela parce que ce sont les rôles des connaissances qui présentent une structure adaptée à la résolution d'un problème particulier.

3.1.2 Couche des inférences

Dans la deuxième couche, les connaissances de contrôle sont abstraites et elles constituent un ensemble d'inférence définissant les opérations sur les connaissances du domaine en termes de rôles. Ces inférences peuvent être vues isolément ou globalement, à l'intérieur de la *structure d'inférence*. La figure 3.3 montre un exemple d'inférence en KADS. Chaque inférence peut être considérée sous deux aspects:

L'action de l'inférence. Une inférence est caractérisée par une action sur une ou plusieurs données d'entrée et produit une nouvelle donnée de sortie. Son nom doit indiquer le type d'opération exécutée (description opérationnelle) ou le but de l'inférence (description téléologique).

Les rôles des connaissances Les rôles des connaissances représentent les données d'entrée et de sortie d'une inférence. Ils sont des "placeholders" pour les objets du domaine et leur nom est un indicatif du rôle de ces objets dans le processus de résolution du problème[4].

L'idée de rôles des connaissances vient originalement des travaux montrés dans le chapitre antérieur (la Classification Heuristique, les méthodes à limitation de rôles),

4. KADS fait la distinction entre différents types de rôles: dynamiques (d'entrée et sortie) et statiques. Ici, nous ne considérons pas cette distinction.

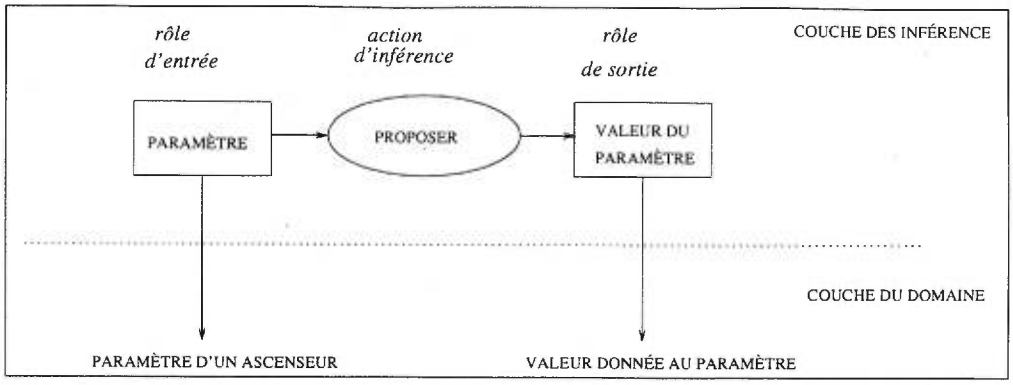

FIG. 3.3 - *Exemple d'une inférence primitive de la couche des inférences dans KADS. L'inférence* **proposer** *utilise comme rôle d'entrée* **paramètre**, *pour générer comme rôle de sortie* **valeur du paramètre**. *Les rôles peuvent être instanciés avec diverses connaissances du domaine. Ici, on voit que les rôles sont instanciés avec des connaissances du domaine des ascenseurs. Le rôle* **paramètre** *est instancié avec les paramètres d'un ascenseur (par exemple,* **capacité maximal***), et le rôle* **valeur du paramètre** *est instancié avec des valeurs possibles associées au paramètre (par exemple,* **10 personnes***). L'ellipse est utilisée pour représenter l'action de l'inférence, les rectangles représentent les rôles des connaissances et les flèches indiquent les dépendances des entrées-sorties des rôles des connaissances.*

où nous avons explicité leurs avantages (section 2.1). Dans KADS, cette idée est exprimée par le *principe de limitation de rôle* [Wielinga et al., 1993], selon lequel une structure est imposée aux connaissances dans le but de résoudre un problème. Les différentes parties de cette structure jouent des rôles spécialisés dans le processus de résolution du problème.

Structure d'inférence

En plus de définir les inférences individuellement, la couche des inférences décrit les inférences à l'aide d'une structure d'inférence montrant la connexion conceptuelle des inférences par les liens avec les rôles des connaissances. Elle permet d'avoir une vision globale de la méthode de résolution de problèmes, mais elle ne spécifie pas l'ordre d'exécution des inférences qui est défini dans la couche de la tâche. Dans le chapitre 2 (section 2.1.2), nous avons souligné les avantages des structures d'inférence. La figure 3.4 illustre un schéma générique pour la structure d'inférence.

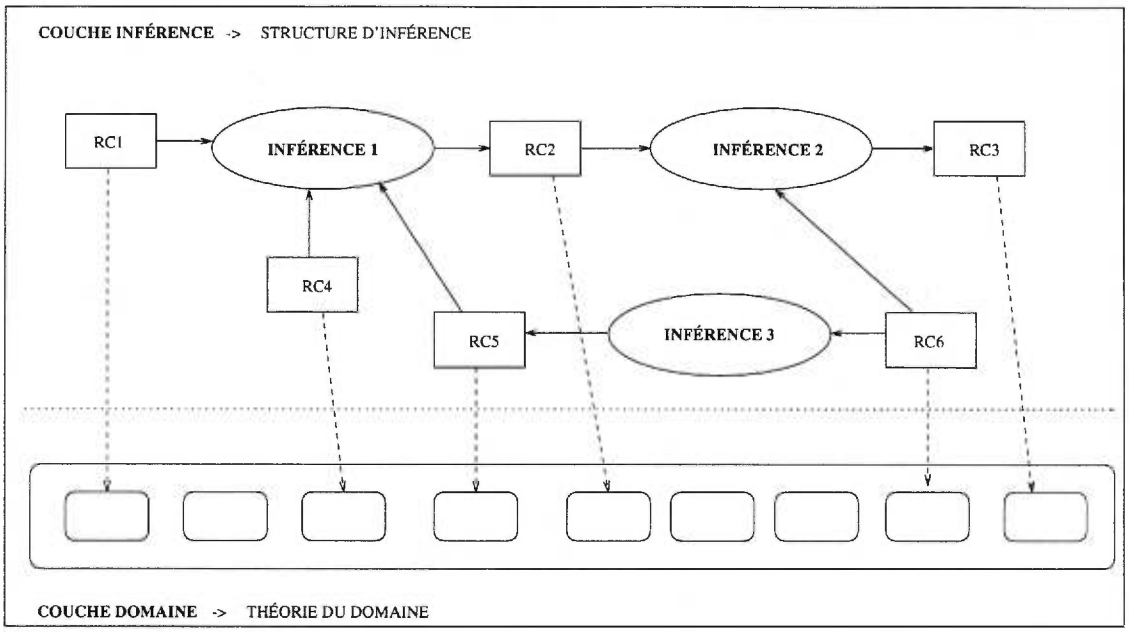

FIG. 3.4 - *Une structure d'inférence générique (adapté de* [Aben, 1995]*). Les étiquettes dans les ellipses indiquent ce que l'inférence accomplit. Les rôles des connaissances sont des "placeholders" pour les connaissances du domaine. Dans cette représentation, la structure des connaissances est définie dans la couche du domaine, laquelle contient une théorie qui établit l'organisation inhérente des concepts et relations du domaine.*

Cependant, le problème avec la structure d'inférence est que les étiquettes qui spécifient les rôles des connaissances sont abstraites et ne clarifient pas l'organisation des rôles des connaissances. La structure des connaissances est spécifiée seulement dans la couche du domaine qui peut ne pas correspondre à la structure des connaissances utilisée par les inférences.

En effet, comme nous avons montré dans la présentation de la couche du domaine, le modèle du domaine présente déjà une vision particulière d'organisation des connaissances. Causse-Gobinet [Causse-Gobinet, 1994] souligne que le modèle du domaine fait un peu un double emploi avec la notion de rôles de connaissances, car on a tendance a nommer un modèle du domaine en fonction des rôles qu'il assume dans le raisonnement. Cela rend difficile la distinction entre modèle du domaine et rôles des connaissances. Pour nous, cette définition d'une organisation particulière de connaissances en fonction de son utilisation devrait être définie seulement dans la couche des inférences, pour ne pas restreindre la réutilisation de la couche du domaine.

3.1.3 Couche de la tâche

La troisième couche du modèle de l'expertise décrit une décomposition récursive d'une tâche en sous-tâches, à leur tour définies comme des tâches. Cette décomposition récursive termine par les inférences primitives[5]. La couche de la tâche détermine quelles sous-tâches et inférences seront exécutées, leur ordre et leur fréquence. Pour faire cela, une tâche est spécifiée en termes de la *définition de la tâche* et du *corps de la tâche*. La définition de la tâche spécifie ce qui est accompli. Le corps de la tâche spécifie "comment" elle est accomplie. Une tâche est donc définie par les éléments suivants (figure 3.5):

Goal Le but que la tâche doit accomplir.

Input Les données d'entrée pour accomplir la tâche.

5. Il n'est pas possible parfois de décider à priori si un élément de la décomposition est une tâche ou une inférence. Il faut attendre la fin de la modélisation pour faire cette distinction.

Output Les données de sortie obtenues après l'exécution de la tâche.

Task Structure La structure de la tâche est une description de la décomposition de la tâche en sous-tâches et en inférences primitives de la couche des inférences et, les dépendances de contrôle entre eux.

```
task Proposer-Valeurs
    goal
        assigner des valeurs à tous les paramètres d'un système
    input
        les informations fournies par l'utilisateur
    output
        les valeurs des paramètres de sortie
    task-structure
        REPEAT
            seléctionner paramètre
            proposer
        UNTIL (les valeurs de tous les paramètres sont calculées)
```

FIG. 3.5 - *Définition de la tâche Proposer-Valeurs en KADS. La définition de la tâche (goal, input et output) est exprimée en langue naturelle. La structure de la tâche est définie en termes algorithmiques: une boucle principale effectue l'exécution des inférences* **sélectionne paramètre** *et* **proposer** *jusqu'à ce que toutes les valeurs de paramètres soient calculées.*

La définition de la tâche fournit déclarativement le but de la tâche par les arguments *goal*, *input* et *output*. Le corps de la tâche décrit la structure de contrôle en termes algorithmiques par l'argument *task-structure*.

3.2 Modélisation en KADS

Dans la section précédente, nous avons présenté le modèle de l'expertise de KADS. Dans cette section, nous présentons d'autres aspects de la méthodologie de modélisation, tels que les langages de modélisation, la bibliothèque de composants réutilisables, ainsi que la typologie d'inférences primitives.

3.2.1 Langages KADS

KADS offre deux langages pour la description des modèles d'expertise: CML [Schreiber et al., 1994], un langage semi-informel, et $(ML)^2$ [Van Harmelen et Balder, 1993], un langage formel. CML ("Conceptual Modelling Language") présente des définitions textuelles et une notation graphique qui permet de représenter les modèles par des diagrammes. CML est un langage structuré, mais encore à un niveau informel. $(ML)^2$ est utilisé pour définir des spécifications formelles qui pourront être validées.

En plus des langages propres à KADS, plusieurs autres langages ont été développés pour décrire le modèle de l'expertise. Ces langages peuvent être séparés en trois classes: les langages opérationnels, les langages formels et les langages à la fois opérationnels et formels.

Les langages opérationnels présentent des primitives exécutables pour définir le modèle conceptuel de KADS. Ils fournissent un feed-back rapide du modèle créé, en permettant de le modifier directement. Les avantages des langages formels sont la précision et la possibilité de vérification des modèles générés ([Fensel et Van Harmelen, 1994] comparent ces différents langages).

Avec les langages au niveau conceptuel et formel, KADS permet plus de flexibilité pour la modélisation des SBCs que les approches qui nous avons présentées, soit les tâches génériques, les méthodes à limitation de rôles et PROTÉGÉ-II. Toutefois, cette flexibilité est aussi à l'origine de certaines critiques sur le fait que KADS n'offre pas beaucoup de support pour rendre les modèles opérationnels. De plus, il y a une grande distance entre le modèle conceptuel et le système implanté. Les langages de modélisation opérationnels visent à surmonter ces problèmes.

Dans notre thèse, nous avons développé un langage, K-Loom, pour modéliser et rendre opérationnel le modèle conceptuel de KADS; K-Loom sera présenté au prochain chapitre.

3.2.2 Bibliothèque KADS

En plus de permettre la modélisation des SBCs à partir de rien, KADS offre une bibliothèque de composants et de méthodes de résolution de problèmes réutilisables. La bibliothèque de CommonKADS [Breuker et Van de Velde, 1994] remplace la bibliothèque de modèles d'interprétation de KADS-I, en unifiant diverses approches, telles que celles des tâches génériques et des méthodes à limitation de rôle.

Dans KADS-I, cette librairie de méthodes de résolution de problèmes réutilisables était constituée de *modèles d'interprétation* organisés selon une taxonomie de tâches. Ces modèles d'interprétation présentaient une structure trop rigide, et ne considéraient pas l'organisation de rôles des connaissances inhérentes aux méthodes.

Dans CommonKADS, la bibliothèque est utilisée d'une façon descendante ("top-down") et elle permet deux approches différentes de modélisation, en fonction du niveau de granularité de ses composants:

1. Les *modèles génériques*, qu'il suffit d'instancier avec les termes du domaine et qui constituent des modèles complets de l'expertise. Ils sont des cadres qui représentent un modèle de l'expertise abstrait qui peut être rempli par des composants de base. Autrement dit, un modèle générique est un cadre de modèle de l'expertise complet dans sa forme, mais non nécessairement dans son contenu.

2. Les *composants de base*, à partir desquels il faut construire un nouveau modèle. Ces composants sont les inférences, les structures d'inférence, les modèles du domaine, les modèles de cas, les ontologies du domaine et les méthodes de résolution de problèmes. Les composants de base présentent un niveau de granularité adéquat pour créer des modèles flexibles. Cependant, ils n'offrent aucune aide pour assembler les composants. Cela est fourni par les modèles génériques. De plus, pour construire un nouveau modèle, la bibliothèque met à la disposition des opérateurs de modélisation (spécialisation, généralisation, etc) qui permettent la transformation des composants de façon à les adapter à une application spécifique.

La bibliothèque CommonKADS permet une modélisation beaucoup plus flexible que celle de KADS-I, en offrant plusieurs niveaux de granularité de ses composants[6]. De plus, la bibliothèque associe à chaque composant des caractéristiques ("features") permettant de vérifier les conditions pour la sélection d'un composant. Ces caractéristiques tiennent compte du principe de l'interaction relative des composants et de la notion de rôles des connaissances. Les caractéristiques explicitent les suppositions sur les restrictions et hypothèses associées à une méthode et elles expriment les contraintes pragmatiques et épistémologiques associées aux composants[7].

Nous avons présenté ici une description générale et succincte de la bibliothèque de KADS. Dans la prochaine section, nous présentons plus en détail la typologie d'inférences de KADS contenue dans la bibliothèque.

3.2.3 Typologie d'inférences

Comme nous avons vu à la section 3.1.2, selon KADS [Aben, 1995], une inférence définit une relation entre les rôles d'entrée et de sortie, de telle sorte que les inférences ne fassent pas référence directement aux connaissances du domaine. Une inférence est définie par son nom et par ses rôles des connaissances.

Le nom de l'inférence peut être caractérisé en fonction d'une opération, dans une perspective opérationnelle ("operator view") ou, en fonction d'un but, dans une perspective téléologique ("teleological view"). Par exemple, une inférence appelée ***sélectionner*** est définie dans une perspective opérationnelle, parce que son nom définit ce que l'inférence va réaliser. Par contre, une inférence appelée ***prendre-décision*** est définie dans une perspective téléologique parce que son nom définit le but de l'inférence. En effet, les deux inférences peuvent effectuer la même fonction; c'est la perspective qui change.

KADS offre une typologie d'inférences [Breuker et Van de Velde, 1994] de façon

6. Les composants sont décrits en CML, mais ils peuvent être traduits semi-automatiquement en ML^2.

7. Les notions de caractéristiques épistémologiques et pragmatiques sont héritées de l'approche *Composants de l'Expertise.*

Catégorie	Description
Concept	Description caractérisant un objet
Instance	Réalisation spécifique d'un concept
Attribut	Propriété ou caractéristique d'un concept
Valeur d'un attribut	Valeur associée à un attribut
Relation	Relations entre les concepts ou instances
Relation de valeur	Expression qui restreint la valeur d'un attribut
Structure	Objet complexe constitué de concepts et relations

TAB. 3.1 - *Catégories épistémologiques inspirées de KL-ONE. Ces catégories délimitent la capacité d'expression des connaissances du domaine et, par conséquent, des inférences.*

à permettre la combinaison et le raffinement des composants réutilisables. Cette typologie a été définie en termes d'opérations sur l'ontologie de base (section 2.2) de KADS, en utilisant ainsi une perspective opérationnelle pour définir les inférences. Les inférences sont caractérisées en termes des opérations sur les catégories ontologiques inspirées de KL-ONE et présentées dans la section 3.1.1. La table 3.1 résume ces catégories considérées pour la définition de la typologie d'inférences.

La table 3.2 montre la classification des inférences de KADS. Les problèmes avec cette caractérisation sont les suivants:

- L'ontologie de base n'est pas nécessairement complète.
- Différentes approches de modélisation utilisent différentes ontologies de base.
- La typologie des inférences n'est pas complète.
- Les noms des inférences ne sont pas précis.

Par exemple, l'inférence ***abstraire*** supprime un attribut d'un concept, en rendant le concept moins spécifique. Cependant, si on compare cette définition de l'inférence ***abstract*** avec celle *d'abstraction qualitative*, selon Clancey [Clancey, 1985], elle ne fonctionne pas. L'abstraction qualitative associe une description quantitative à une description qualitative. Par exemple, si chez un adulte, le taux de globules blancs est inférieur à 2500, alors on peut considérer que le taux de globules blancs est bas. Ainsi, dans ce cas d'abstraction, il n'y a pas moins d'attributs que dans le concept original.

Type d'inférence	Inférence	Opération
Manipulation d'attribut et de concept	Abstraire	Enlève des attributs d'un concept
	Spécialiser	Ajoute des attributs à un concept
	Généraliser	Obtient un concept à partir d'un ensemble d'instances
	Spécialiser	Obtient des instances à partir d'un concept
	Instancier	Obtient une instance à partir d'un concept
	Classifier	Associe un concept à une instance
Manipulation d' attribut et de valeur	Affecter	Donne une valeur à un attribut d'un concept
	Enlever valeur	Enlève une valeur d'un attribut d'un concept
	Calculer	Calcule la valeur d'un attribut d'un concept
	Comparer	Compare les valeurs des attributs de concepts
	Apparier	Compare des structures
Manipulation d'ensemble	Sélectionner	Sélectionne une instance à partir d'un ensemble d'instances
	Sélectionner sous-ensemble	Sélectionne un sous-ensemble d'instances à partir d'un ensemble
	Ordonner	Ordonne un ensemble d'instances
	Unifier	Unifie des ensembles d'instances
Manipulation de structure	Assembler	Compose un ensemble d'instances en une structure
	Décomposer	Décompose une structure dans un ensemble d'instances
	Transformer	Transforme une structure en une autre structure
	Sélectionner instance	Sélectionne une instance à partir d'une structure
	Sélectionner instance de relation	Sélectionne une instance d'une relation à partir d'une structure

TAB. 3.2 - *Typologie d'inférences de KADS. Cette classification est définie en termes d'opérations sur les catégories ontologiques de l'ontologie de base, qui sont à leur tour, inspirées des catégories épistémologique du langage KL-ONE.*

De même, le concept d'abstraction est très général et pourrait être associé à différents types d'inférences.

Ainsi, la classification des inférences selon l'ontologie de base est un peu restreinte et peut mener à des ambiguïtés. Afin de préciser ces inférences et de faciliter leur sélection, Aben [Aben, 1995] a formalisé les catégories ontologiques de l'ontologie de base, ainsi que les inférences elles-mêmes.

La typologie de KADS sert de référence pour la définition des inférences, au même titre qu'un type abstrait de données dans un langage de programmation. En ce sens, les inférences spécifiées dans le cadre d'une application sont associées à cette typologie, comme les variables sont associées à des types abstraits. Il faut, selon cette perspective, faire correspondre la typologie de KADS et les inférences dans le contexte d'une application spécifique.

Au chapitre 5, nous utilisons l'approche de KADS pour définir les inférences des méthodes de résolution de problèmes en termes d'une ontologie. Alors que KADS utilise l'ontologie de base, nous utilisons l'ontologie de la méthode pour définir les inférences. Nous montrons que cette approche a l'avantage de dévoiler l'organisation des connaissances associées aux rôles des connaissances; les rôles des connaissances seront plus détaillés, au lieu d'être seulement définis par une étiquette. De plus, les inférences définies selon l'ontologie de la méthode peuvent être considérées comme des réalisations de la typologie d'inférences de KADS.

3.3 Résumé

Dans ce chapitre, nous avons présenté la méthodologie KADS. Nous avons décrit le modèle de l'expertise: les couches du domaine, des inférences et de la tâche. Nous avons montré les constituants de ces couches et nous avons critiqué les aspects liés à la description des rôles des connaissances dans la couche des inférences. D'autre part, nous avons présenté quelques aspects de la modélisation en KADS, tels que les langages de modélisation, la bibliothèque de modèles génériques et de composants de base et, finalement, nous avons présenté la typologie d'inférences. Cette typologie sera

utilisée dans le chapitre 5, pour comparer les inférences définies selon KADS avec les inférences telles que nous les avons définies.

Dans le prochaine chapitre, nous présentons K-Loom, le langage que nous avons développé pour définir des méthodes de résolution de problèmes, selon le modèle de l'expertise de KADS.

Chapitre 4

K-Loom

Dans le chapitre précédent, nous avons présenté le modèle conceptuel de KADS que nous utilisons pour représenter les méthodes de résolution de problèmes. Toutefois, au lieu d'utiliser CML et $(ML)^2$, les langages semi-formel et formel de KADS, nous avons choisi de définir un autre langage pour représenter les connaissances du modèle de l'expertise. Dans ce chapitre, nous décrivons ce langage, appelé K-Loom (KADS-Loom).

K-Loom est adéquat pour représenter le modèle de l'expertise de KADS parce qu'il présente des primitives similaires à celles de CML, en plus d'être un langage opérationnel. En effet, K-Loom est une extension de Loom, un langage de représentation de connaissances du type terminologique [MacGregor, 1988, MacGregor et Burstein, 1991] dérivé du langage KL-ONE. Notre idée est de construire un langage KADS sur Loom de façon à pouvoir définir et tester les modèles conceptuels de méthodes de résolution de problèmes. Nous avons choisi Loom parce qu'il est un langage déjà existent, stable et bien développé. Nous présentons dans ce chapitre les caractéristiques des langages terminologiques en général et la syntaxe spécifique de Loom. De plus, nous présentons K-Loom, notre extension de Loom proposée pour représenter les couches des inférences et de la tâche de KADS.

4.1 Loom

Loom combine les caractéristiques d'un langage déclaratif avec plusieurs paradigmes de programmation comme la programmation logique, les langages à objets, les systèmes à base de règles de production, etc. Loom offre la capacité de raisonnement terminologique qui permet de décrire, de vérifier et d'organiser les termes génériques d'un domaine. De plus, le langage rend possible la création d'instances réelles et de faire des assertions par rapport à ces instances. Ici, nous nous limitons à décrire les aspects terminologiques de Loom. Cette section donne des notions de base à ceux qui ne sont pas familiers avec Loom ou d'autres langages terminologiques.

Les langages terminologiques peuvent être vus comme une élaboration formelle d'idées sous-jacentes aux réseaux sémantiques [Sowa, 1991]. KL-ONE, un de ces premiers langages, tente de traiter les inconsistances et les ambiguïtés de ces réseaux. Ces langages sont centrés sur la notion *d'objet* qui, dans ce contexte, est appelé *concept* ou *terme.*

Tous les langages terminologiques offrent deux formalismes: un formalisme pour la description de termes, appelé TBox ("terminological box") et un autre pour les assertions sur les termes, appelé ABox ("assertional box"). Le langage pour les descriptions, TBox, permet de former des termes à partir d'autres termes et en utilisant un ensemble d'opérateurs. D'où l'appellation de *logique terminologique* ou *logique de description.* Le langage pour les assertions, ABox, utilise les termes prédéfinis pour faire des assertions.

4.1.1 Classificateur

Dans le TBox, le classificateur ("classifier") organise et indexe les descriptions dans une hiérarchie. Ainsi, toutes les descriptions introduites dans le système sont insérées dans une taxonomie, avec les descriptions plus générales placées avant les plus spécifiques. Autrement dit, le classificateur infère et maintient un treillis taxonomique consistant en relations de *subsomption* entre les descriptions. De cette façon, une description est placée dans la hiérarchie au-dessous de toutes les descriptions qui la

subsument et au-dessus de toutes les descriptions subsumées par elle.

Une description subsume une autre description si elle est plus générale. Selon cette perspective, si un concept A subsume un concept B, tous les instances du type B peuvent être décrites comme du type A. De même, toutes les restrictions associées à A sont héritées par B. La figure 4.1 illustre une hiérarchie de concepts créée et entretenue par le classificateur.

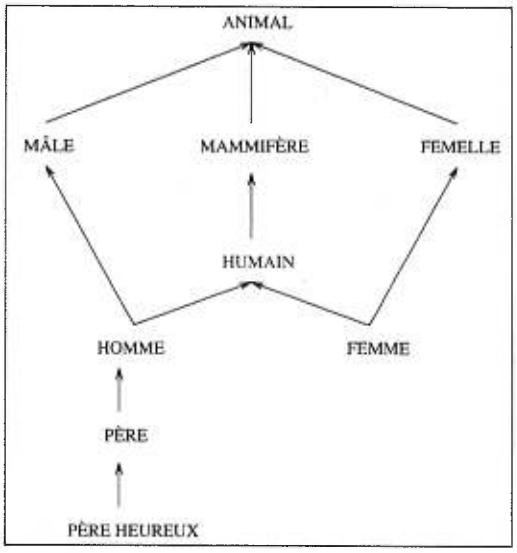

FIG. 4.1 - *Hiérarchie de concepts. Les flèches représentent les relations de subsomption. Un concept peut avoir plus d'un super-concept, comme dans le cas de* **Homme** *qui a les super-concepts* **Humain** *et* **Mâle**. *Ainsi,* **Homme** *hérite toutes les restrictions associées à* **Humain** *et* **Mâle**.

4.1.2 Descriptions

Concepts

Le TBox offre des opérateurs pour décrire les concepts. Ces opérateurs permettent d'expliciter les caractéristiques associées à un concept et aussi de définir un concept en termes d'autres concepts. L'opérateur `defconcept` associe un symbole à une description d'un terme. Par exemple:

```
(defconcept Animal)
(defconcept Humain)
(defconcept Homme :is (:and Humain Male))
(defconcept Femme :is (:and Humain Femelle))
```

Relations

En plus de la notion de concept, les langages terminologiques présentent la notion de *rôle*. Pour éviter une confusion entre le concept de *rôles des connaissances* des méthodes de résolution de problèmes et le concept de *rôle* dans les langages terminologiques, nous utilisons le terme *relation*, au lieu de *rôle* pour les langages terminologiques. Une relation est liée à un concept et décrit un attribut qui établit des associations entre les instances du type du concept et d'autres instances. Les relations ont des *remplisseurs* ("fillers") potentiels auxquels on associe des restrictions de cardinalité et de valeur. Les restrictions sont exprimées par les opérateurs `:at-least`, `:at-most`, `:exactly`, `:all` et `:the`.

Par exemple, dans les définitions suivantes, au concept **Père** est associée la relation **Enfant** avec la restriction de cardinalité selon laquelle, pour appartenir à la classe **Père**, il faut avoir au moins un **Enfant**. En plus des restrictions de cardinalité, une description peut restreindre les valeurs des remplisseurs des relations. Par exemple, le concept **Père-Heureux** est défini avec la restriction selon laquelle tous les remplisseurs du type **Fille** sont complétés par des instances appartenant au concept **Mariée**.

```
(defconcept Pere :is (:and Male (:at-least 1 Enfant)))
(defconcept Pere-Heureux :is (:and Pere (:all Fille Mariee)))
```

L'héritage de restriction entre les relations est similaire à la relation de subsomption entre les concepts, en ce sens que les restrictions sont héritées par les relations plus spécifiques. Prenons par exemple, un concept A avec une relation RA qui subsume un autre concept B; si à B est associée une relation RB qui restreint RA, alors les remplisseurs de RB respectent les restrictions de RA et RB. Dans les définitions suivantes, les relations **Fils** et **Fille** héritent de la restriction de la relation **Enfant**.

Dans la définition de **Enfant**, le premier élément de la relation ("domain") est restreint à être de la classe **Humain** et le deuxième élément de la relation ("range") doit être un **Humain**. De plus, il faut que les remplisseurs des relations **Fils** et **Fille** respectent les restrictions ajoutées dans leurs propres définitions. Ainsi, **Fils** doit être **Humain** et **Male** et **Fille** doit être **Humain** et **Femelle**.

```
(defrelation Enfant :domain Humain :range Humain)
(defrelation Fils :is (:and Enfant (:range Male)))
(defrelation Fille :is (:and Enfant (:range Femelle)))
```

Opérateur :satisfies

Dans le TBox, on peut aussi décrire des concepts avec l'opérateur :satisfies qui définit un concept en associant toutes les instances qui satisfont la requête spécifiée dans la définition. Le même opérateur peut être utilisé pour définir une relation. L'exemple suivant illustre l'utilisation de cet opérateur, où le concept **Père-Fatigué** définit les pères dont tous les enfants sont malades:

```
(defconcept Pere-Fatigue :is
  (:satisfies ?x (:and (Pere ?x)
                       (:for-all ?y (:and (Enfant ?x ?y)
                                          (Malade ?y))))))
```

Descriptions primitives et définies

Dans les langages terminologiques, il est important de comprendre la distinction entre les descriptions primitives et définies. Les descriptions *primitives* contiennent le terme :is-primitive dans leurs définitions. Elles sont associées aux termes qui ne sont pas définis complètement. En d'autres mots, les descriptions primitives ne sont pas définies par toutes les caractéristiques (nécessaires et suffisantes) pour les spécifier. Il y a donc des informations qui distinguent les concepts primitifs d'autres concepts que ne sont pas explicitées. Par exemple, dans le domaine de la géométrie, les concepts **Point**, **Ligne** et **Polygone** sont considérés comme primitives:

```
(defconcept Point :is-primitive)
```

```
(defconcept Ligne :is-primitive)
(defconcept Polygone :is-primitive)
```

Les descriptions *définies* sont spécifiées à partir des descriptions primitives. La description de **Triangle** est un type de description définie:

```
(defconcept Triangle :is (:and Polygone (:the  Nombre-Cotes 3)))
```

Cette définition de triangle donne les conditions nécessaires et suffisantes pour être un triangle. Ainsi, étant donné, un polygone de 3 côtés, il sera automatiquement classifié comme triangle. Les définitions de **Quadrilatère** et **Rectangle** suivantes illustrent la différence entre les concepts primitifs et définis.

```
(defconcept Quadrilatere :is (:and Polygone (:the  Nombre-Cotes 4)))
(defconcept Rectangle:is-primitif (:and Polygone (:the Nombre-Cotes 4)))
```

Dans ce cas, étant donné, un polygone dont le nombre de côtés est égal à 4, ce polygone sera classifié comme **Quadrilatère**, mais non comme **Rectangle**.

4.1.3 Assertions

Les instances représentent les objets dans l'univers du discours. Le langage d'assertion permet de compléter les descriptions et de définir des instances associées aux concepts. Le ABox associe les instances aux concepts ("recognizer") et déduit des informations en raisonnant à partir de la hiérarchie de concepts. À chaque modification de la base de connaissances, l'association entre instances et concepts est maintenue à jour. Les instances de concepts peuvent être reconnues seulement par leurs descriptions en termes d'autres concepts. La commande `tell` permet de faire les assertions:

```
(tell (Homme Jean))
(tell (Humain Marie) (Femelle Marie))
(tell (Polygone T1) (Nombre-Cotes T1 3))
(tell (Polygone R1) (Nombre-Cotes R1 4))
(tell (Polygone R2) (Nombre-Cotes R2 4))
```

Une autre manière de complémenter la description d'un concept est par l'utilisation de la commande `implies` qui fait partie du langage d'assertions du ABox. Cette

commande permet d'ajouter des caractéristiques ou des restrictions à un concept déjà défini. Dans l'exemple suivant, le concept **Mère-Malheureuse** est d'abord défini sans aucune restriction. La commande `implies` ajoute des restrictions additionnelles au concept prédéfini.

```
(defconcept Mere-Malheureuse :is Mere)
(implies Mere-Malheureuse (:all Enfant Mauvais-Eleve))
```

4.1.4 Requêtes

Loom offre des facilités pour faire des requêtes à propos du contenu de la base de connaissances. La commande `Ask`, permet de tester si un fait existe ou peut être déduit de la base de connaissances (la valeur `T` signifie que le résultat de la requête est vrai; la valeur `NIL` signifie que le résultat de la requête est faux):

```
(ask (Femme Marie)) -> T
(ask (Homme Paul)) -> NIL
(ask (Triangle T1)) -> T
(ask (Rectangle R1)) -> T
(ask (Rectangle R2)) -> NIL
```

La commande `retrieve` permet d'obtenir toutes les instances qui satisfont les conditions définies. Dans ces requêtes, les variables débutent par un point d'interrogation ("?") et des quantificateurs universels ou existentiels peuvent être associés aux variables. Dans les requêtes suivantes, la première obtient toutes les instances appartenant au concept **Père** qui ont au moins une fille mariée. La deuxième requête obtient tous les instances appartenant au concept **Père**, dont toutes les filles ne sont pas mariées.

```
(retrieve ?x (:and (Pere ?x)
                   (:for-some ?y (:and (Fille ?x ?y) (Mariee ?y)))))
(retrieve ?x (:and (Pere ?x)
                   (:for-all ?y (:implies (Fille ?x ?y)
                                          (:not (Mariee ?y))))))
```

Nous ne détaillons pas les opérateurs de formation des descriptions en Loom ou le langage d'assertions et de requêtes. Nous n'avons montré que les notions de base

de Loom, pour faciliter la compréhension de la suite de la thèse. Pour plus de détail, il faudra consulter le manuel de Loom [Brill, 1993].

4.2 K-Loom

Notre idée en définissant K-Loom est de profiter des caractéristiques terminologiques et de classification de Loom pour représenter les couches du domaine, des inférences et de la tâche de KADS.

Un langage similaire à K-Loom est Model/KADS [Barbuceanu, 1993], créé dans un environnement de modélisation de connaissances intégré (IKME - "Integrated Knowledge Modeling Environments"). Cet environnement offre un cadre pour la construction des SBCs, en facilitant l'acquisition de connaissances, l'utilisation et l'évolution de ces systèmes. L'idée intéressante dans IKME est d'offrir un environnement pour la modélisation des SBCs et la transformation des modèles en systèmes opérationnels. Cet environnement offre un langage général de modélisation de type terminologique appelé MODEL. À partir de ce langage, plusieurs méthodologies de développement des SBCs peuvent être utilisées. L'idée principale est d'avoir ce langage terminologique comme cadre et de construire des interfaces pour différentes approches de modélisation de connaissances comme KADS, les tâches génériques, etc. Model/KADS est l'interface développée pour faire une modélisation à la KADS.

Notre langage K-Loom utilise le même principe que Model/KADS, car notre modélisation est basée sur une représentation terminologique des connaissances. Cependant, nous avons développé K-Loom dans un contexte plus restreint, parce que nous développons une interface de Loom seulement pour KADS et non pour différentes approches de modélisation.

Dans les sections suivantes, nous présentons les extensions de K-Loom pour représenter le modèle de l'expertise de KADS. Comme nous avons montré les principales notions de Loom dans cette section, nous ne décrirons pas les primitives de modélisation pour la couche du domaine où K-Loom est semblable à Loom. Nous nous restreindrons à présenter les primitives de modélisation pour les couches des infé-

rences et de la tâche de KADS. Un exemple détaillé de modélisation en K-Loom est présenté au chapitre 5.

4.2.1 Couche des inférences

Les inférences spécifient la capacité de déduction d'une méthode de résolution de problèmes. En K-Loom, nous proposons une extension de Loom pour représenter les inférences en utilisant trois arguments: (1) une condition, (2) un corps, et (3) un résultat:

```
(definference  <NOM-INFERENCE>
  :cond <REQUETE-LOOM>
  :body <FONCTION-INFERENCE>
  :result <REQUETE-LOOM>)
```

Le `<NOM-INFERENCE>` contient le nom de l'inférence. L'argument `:cond` définit les conditions à satisfaire pour l'application de l'inférence. Cette condition est exprimée en termes d'une requête Loom, faisant référence aux rôles des connaissances d'entrée. L'argument `:body` définit une fonction qui réalise l'inférence et utilise les variables données dans la condition. L'argument `:result` exprime les faits qui doivent être vrais après la réalisation de l'inférence en termes des rôles des connaissances de sortie.

L'argument `:body` représente un appel à une fonction qui accomplit l'inférence. En ne montrant pas le corps de l'inférence, nous voulons faire ressortir que l'implantation de l'inférence n'est pas nécessaire à la compréhension d'une méthode de résolution de problèmes.

4.2.2 Couche de la tâche

La couche de la tâche contrôle l'ordre et la fréquence d'activation des inférences définies dans la couche des inférences. En K-Loom, nous représentons une tâche de la façon suivante:

```
(deftask  <NOM-TACHE>
  :input <REQUETE-LOOM>
  :output <REQUETE-LOOM>)
  :task-structure  <LISTE-COMMANDES>
```

NOM-TACHE contient le nom de la tâche. L'argument :input définit les conditions à vérifier avant l'exécution de la tâche. Cette condition est une requête Loom faisant référence aux rôles des connaissances. L'argument :output exprime les faits à vérifier après l'exécution de la tâche. L'argument :task-structure définit la séquence d'exécution des inférences en termes de conditions et itérations. La liste de commandes de la structure de la tâche peut être constituée d'une seule commande ou d'une liste de commandes qui reprennent les structures de contrôle classiques des langages de programmation.

Pour définir des itérations en K-Loom, nous avons les commandes suivantes:

```
(while <REQUETE-LOOM>
  <COMMANDE-1>
  <COMMANDE-2>
        :
        :
  <COMMANDE-n>)


(repeat-until <REQUETE-LOOM>
  <COMMANDE-1>
  <COMMANDE-2>
        :
        :
  <COMMANDE-n>)


(if-cond <REQUETE-LOOM>
    <LISTE-COMMANDES-1>
    <LISTE-COMMANDES-2>)
```

La commande **while** exécute les commandes tant que la condition est vraie. La commande **repeat-until** exécute les commandes jusqu'à ce que la condition soit vraie. La commande **if-cond** définit l'exécution conditionnel d'une inférence. Dans ce cas, si la condition est vraie, la LISTE-DE-COMMANDES-1 va être exécutée, sinon la LISTE-DE-COMMANDES-2 sera exécutée.

Ceci termine notre description de K-Loom, un langage qui étend Loom pour spécifier les connaissances selon le modèle de l'expertise de KADS. Ces extensions

sont simples, mais permettent une représentation précise des connaissances et une opérationalisation de cette représentation.

Comme nous l'avons mentionné au chapitre précédent, plusieurs langages ont été développés pour représenter le modèle conceptuel de KADS. Ces langages sont classifiés en trois types: formels, opérationnels et une combinaison de formels et opérationnels. K-Loom est un langage opérationnel dont la modélisation est basée sur l'utilisation d'un langage terminologique. Les avantages de K-Loom sont:

- Une syntaxe déclarative qui contribue à la clarté et la précision des connaissances.

- Un langage opérationnel; en formalisant les connaissances en Loom, nous pouvons tester le modèle et le modifier d'après le feed-back obtenu à partir du prototype.

- La possibilité de représenter les éléments génériques (les concepts, les relations - TBox) et les éléments spécifiques (les faits - ABox) d'un domaine.

- Un classificateur qui compare un concept avec d'autres et trouve sa place dans la taxonomie; de plus, les définitions terminologiques et leurs conséquences logiques sont propagées pour les assertions. Ainsi, les instances ont les mêmes propriétés que les concepts.

- Un langage de description expressif, associé à la capacité de raisonnement, en plus d'un langage de requêtes basé sur la logique de premier ordre. C'est le principal avantage de Loom par rapport à Ontolingua qui n'offre ni la capacité déductive de Loom, ni la possibilité de faire des requêtes.

- Le même langage pour représenter les connaissances du domaine (concepts et relations) et les connaissances de contrôle (inférences et tâches), tout en gardant leur indépendance. Cela facilite l'intégration des composants en différentes couches, ainsi que les opérations de correspondance entre les ontologies tel que

nous le voyons au chapitre 6[1].

- La facilité de générer des outils d'acquisition de connaissances spécifiques par domaine ou par méthode de résolution de problèmes à partir des définitions terminologiques.

- La distinction entre les composants réutilisables et les composants non réutilisables par la séparation entre les composants génériques (TBox) et les composants factuels (ABox).

4.3 Résumé

La création des langages pour la modélisation en KADS a été et continue d'être un champ de recherche très important. Dans ce chapitre, nous avons décrit K-Loom, notre langage pour modéliser KADS en utilisant Loom. Nous avons étendu Loom pour représenter la couche des inférences et la couche de la tâche, alors que nous utilisons Loom tel quel pour la couche du domaine. K-Loom est opérationnel, tout en ayant une syntaxe déclarative. K-Loom modélise les trois couches en utilisant le même langage, ce qui facilite l'intégration des trois types de connaissances.

Dans le prochain chapitre, nous modélisons en détail la méthode de résolution de problèmes *Proposer et Réviser* en explicitant les rôles des connaissances, les inférences et la tâche à l'aide de descriptions et requêtes terminologiques de K-Loom.

1. Nous avons tenté d'étendre Ontolingua pour représenter les connaissances de contrôle [Coelho et Lapalme, 1995]. Cependant, comme nous avons dit, Ontolingua n'offre pas les avantages déductifs et la puissance des requêtes de Loom

Chapitre 5

Ontologies de méthode: la modélisation de *Proposer et Réviser*

Comme[1] nous l'avons vu au chapitre 2, plusieurs approches en ingénierie de connaissances traitent du problème de l'identification et de la description des méthodes de résolution de problèmes réutilisables. Cependant, nous avons vu que les descriptions de méthodes de résolution sont soit à un niveau trop abstrait, soit au niveau d'implantation, qui ne dévoilent pas la structure réelle des rôles des connaissances et limitent la compréhension et la réutilisabilité des méthodes. De plus, il est difficile de modifier ces méthodes pour des problèmes légèrement différents.

Ici, nous considérons qu'une méthode de résolution de problèmes définit l'ordre d'exécution d'un ensemble d'opérations (inférences) sur une structure de connaissances particulière. Cette structure décrit les rôles joués par les connaissances dans la résolution du problème. Rappelons que les rôles sont des "placeholders" pour les connaissances du domaine et, peuvent être instanciés pour différentes applications.

Dans ce chapitre, nous proposons une approche pour spécifier des méthodes de

1. Ce chapitre est une version modifiée et approfondie de l'article "Describing Reusable Problem-Solving Methods with a Method Ontology" présenté au workshop KAW'96 [Coelho et Lapalme, 1996].

résolution de problèmes réutilisables utilisant les éléments suivants: la structure d'inférence, l'ontologie de la méthode, les inférences et la structure de contrôle.

La structure d'inférence donne une vision globale et abstraite des inférences et des rôles des connaissances. L'ontologie de la méthode décrit l'organisation des rôles des connaissances. Les inférences décrivent les opérations sur les rôles des connaissances définies selon l'ontologie de méthode. La structure de contrôle définit l'ordre et la fréquence d'exécution des inférences.

Pour définir les méthodes, nous séparons les connaissances selon les catégories du modèle de l'expertise de KADS (chapitre 3) et nous définissons l'ontologie de la méthode, les inférences et la tâche en utilisant K-Loom (chapitre 4). Notre idée est d'offrir un niveau intermédiaire de description des méthodes de résolution de problèmes, en montrant les avantages de faire cette description au moyen d'un langage terminologique. Nous illustrons notre approche par la modélisation de la méthode *Proposer et Réviser*, appliquée à la tâche VT [Yost, 1994] qui configure un système d'ascenseur.

5.1 Ontologie de Méthode

Les recherches actuelles [Fensel, 1995a, Benjamins et Pierret-Golbreich, 1996] tentent d'expliciter les suppositions sous-jacentes à la structure des connaissances utilisée par une méthode. Nous partageons cette perspective selon laquelle une méthode de résolution de problèmes est un ensemble d'opérations sur une structure particulière de connaissances que nous devons clarifier pour comprendre et réutiliser la méthode. La structure de connaissances définit l'organisation des rôles joués par les connaissances dans la résolution du problème, tandis que les opérations sont définies par des inférences.

Pour définir les rôles des connaissances, nous avons choisi de construire une ontologie qui formalise les rôles en termes des concepts et de relations. Comme nous l'avons vu dans la section 1.1.4, une ontologie permet de définir des connaissances partagées et réutilisables. Selon cette vision, les ontologies doivent être construites

indépendamment d'une application spécifique.

Toutefois, dans notre thèse, nous adoptons une autre orientation. Nous voulons définir des ontologies qui sont spécifiques aux méthodes de résolution de problèmes. Nous appelons ces ontologies *ontologies de méthode*, comme dans l'approche PROTÉGÉ-II [Musen et al., 1994], parce qu'elles expriment la structure des rôles des connaissances utilisée par une méthode. Notre objectif est d'expliciter les engagements ontologiques de la méthode (section 1.1.4), sans en spécifier la réalisation au niveau de l'implantation.

Dans KADS, les rôles des connaissances sont définis comme des étiquettes qui ne précisent pas beaucoup la structure de rôles des connaissances. En effet, c'est dans la couche du domaine, par le modèle du domaine, que les connaissances sont modélisées en fonction de leur utilisation. La construction de l'ontologie de la méthode pour décrire les rôles des connaissances est, à notre avis, plus adéquate, parce que les rôles explicitent la structure particulière des connaissances utilisée par la méthode qui peut ne pas coïncider avec le modèle de la couche du domaine.

La base de notre approche est illustrée à la figure 5.1, où sont définies des implantations de coquilles de méthodes de résolution de problèmes réutilisables, dont l'interface est l'ontologie de la méthode. Une ontologie de la méthode décrit l'organisation et les suppositions sur les connaissances utilisées par les inférences de la méthode. Avant de sélectionner une méthode pour une application spécifique, le cogniticien vérifiera si le domaine d'application est conforme ou adaptable à l'ontologie de la méthode. De cette façon, l'ontologie sert comme une spécification de la méthode et elle peut ainsi contribuer à leur réutilisation dans différents domaines.

Des travaux récents [Linster, 1992, Gennari et al., 1993] en ingénierie des connaissances soulignent l'importance des correspondances ("mappings") entre ontologies. Au prochain chapitre, nous allons définir les types d'opérateurs de correspondance offerts au cogniticien. Ces opérateurs permettent l'adaptation des connaissances d'un domaine particulier aux rôles des connaissances de la méthode. Lorsqu'il est impossible d'établir des correspondances, la méthode est rejetée. Cependant, cette analyse aidera à chercher une autre méthode ou même à construire une nouvelle méthode.

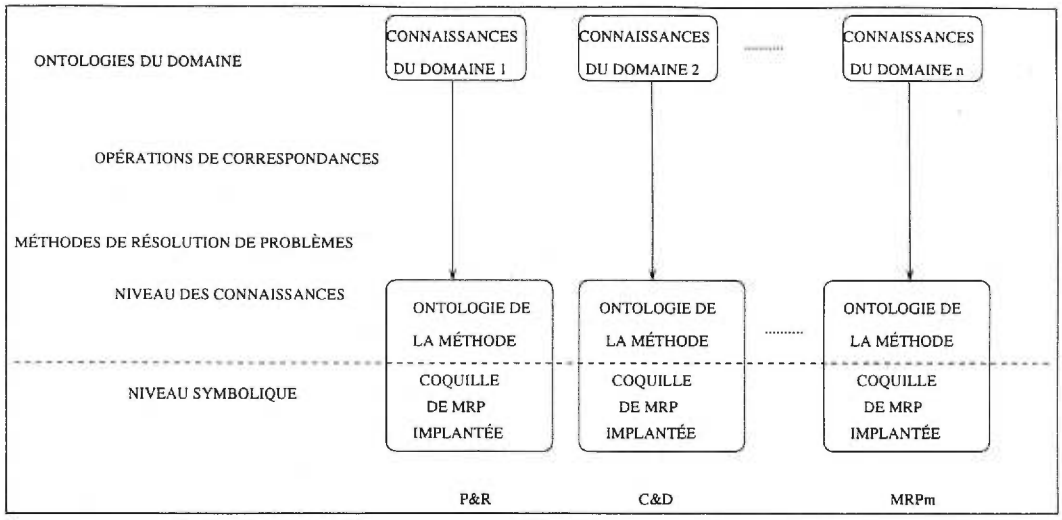

FIG. 5.1 - *Réutilisation des méthodes de résolution de problèmes. Les coquilles des méthodes sont décrites via l'ontologie de la méthode. Pour réutiliser une méthode, le cogniticien doit analyser si le domaine d'application correspond aux rôles des connaissances de la méthode. Si c'est le cas, le cogniticien doit faire correspondre le domaine d'application aux rôles des connaissances en utilisant des opérations de correspondance.*

5.2 Inférences

Nous considérons qu'il y a une forte interdépendance entre la définition des inférences et celles de rôles des connaissances [Le Roux, 1994]. Pour la définition des inférences, nous avons les mêmes problèmes que pour la description des rôles des connaissances: les inférences sont soit définies à un niveau abstrait, soit au niveau d'implantation.

Dans la structure d'inférence selon KADS, par exemple, les inférences sont décrites par des étiquettes qui ne dévoilent pas clairement la relation entre les inférences et les rôles des connaissances. Par contre, dans des approches telles que PROTÉGÉ-II [Musen et al., 1994] et Spark-Burn-Firefighter [Klinker et al., 1991], les inférences sont définies en termes de *mécanismes*. Aux mécanismes correspondent une implantation, mais non une spécification déclarative.

Dans la typologie d'inférences de KADS (section 3.2.3), les inférences ont été formalisées [Aben, 1995] et définies selon une ontologie de base (section 2.2). Cette

définition des inférences est beaucoup plus précise que dans la structure d'inférence. Cependant, les inférences sont classifiées en fonction des opérations sur les catégories épistémologiques de cette ontologie de base. Néanmoins, la description des instances des inférences demeure informelle.

Pour définir les inférences des méthodes, nous avons adopté l'approche de KADS pour définir la typologie d'inférences: nous considérons que les inférences opèrent sur une ontologie prédéfinie. Dans notre cas, l'ontologie considérée est l'ontologie de la méthode qui décrit les rôles des connaissances. Les inférences sont définies en fonction de l'ontologie spécifique à la méthode. Cette définition des inférences est déclarative et fait abstraction des détails d'implantation, en permettant de spécifier plus clairement les relations entre les rôles des connaissances et les inférences. Ainsi, les inférences des méthodes de résolution de problèmes sont des opérations sur les catégories ontologiques des rôles des connaissances.

La différence par rapport à la typologie d'inférences de KADS se situe au niveau d'abstraction. Dans KADS, la typologie fait référence à une ontologie de base et constituent, ainsi, une *méta-description* des inférences. Dans notre cas, les inférences font référence à une ontologie spécifique et elles offrent une définition précise des inférences de la méthode. Les inférences classifiées selon KADS peuvent être vues comme des types abstraits des inférences des méthodes de résolution de problèmes, telles que nous les avons définies. Nous comparons plus en détail les inférences de KADS avec les nôtres (section 5.4.1)

5.3 Modélisation de P&R

Proposer et Réviser (P&R) est une méthode à limitation des rôles (section 2.1.3), classifiée comme une méthode de configuration paramétrique, parce qu'elle configure un système défini par ses paramètres, en assignant une valeur à chaque paramètre. La méthode consiste à faire la configuration d'un système en proposant une valeur pour chaque paramètre et en vérifiant si chacun satisfait les contraintes qui lui sont associées. Nous avons modélisé P&R pour la tâche VT (de "vertical transportation"),

selon la description faite par Yost [Yost, 1994]. La tâche VT réalise un problème de conception dont l'objectif est de configurer un ascenseur. De plus, cette tâche est devenue un exemple très intéressant pour comprendre, comparer et évaluer des nombreuses approches en ingénierie des connaissances. À ce sujet on peut se référer aux modélisations et implantations de VT présentées dans [Gaines, 1996].

5.3.1 Structure d'inférence de P&R

La figure 5.2 illustre la structure d'inférence de P&R, abstraite du domaine VT. Nous avons décomposé la méthode en cinq inférences[2]:

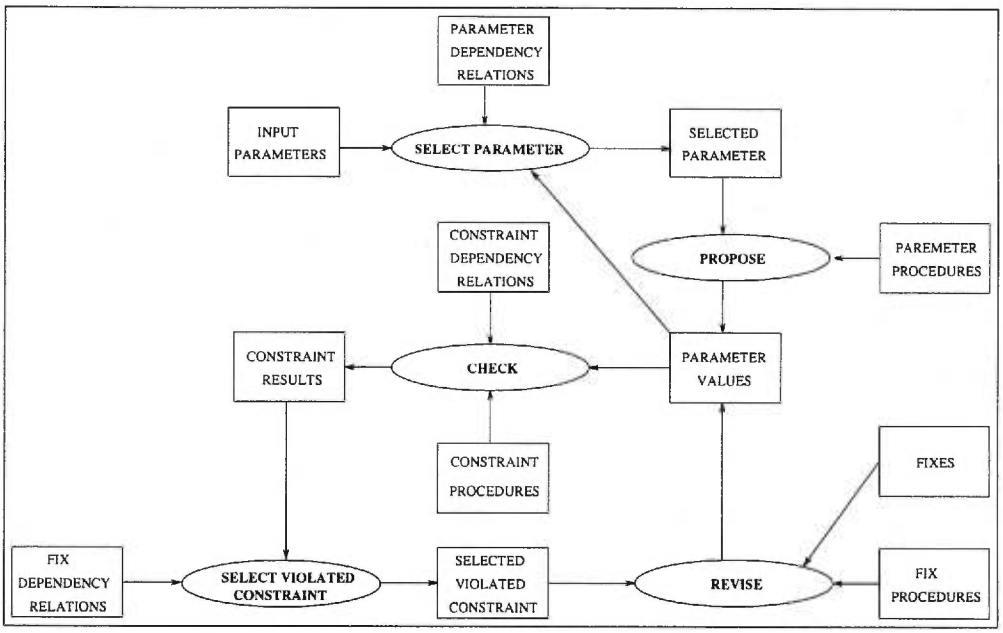

FIG. 5.2 - *Notre représentation de la structure d'inférence pour P&R. La structure d'inférence spécifie les inférences indépendamment de l'ordre d'application. Les ellipses représentent l'action de l'inférence, les rectangles représentent les rôles des connaissances et les flèches indiquent les dépendances des entrées-sorties des rôles des connaissances.*

- L'inférence ***select parameter*** sélectionne un paramètre dont la valeur va être calculée. L'inférence utilise les paramètres d'entrée (**input parameters**), les

2. Les noms des inférences et des rôles des connaissances sont restés en anglais pour correspondre au code implanté.

valeurs des paramètres (**parameter values**) déjà calculées et les relations de dépendance entre les paramètres (**parameter dependency relations**) pour obtenir le paramètre sélectionné (**selected parameter**).

- L'inférence ***propose*** propose une valeur au paramètre sélectionné. L'inférence utilise le paramètre sélectionné (**selected parameter**) et les procédures associées aux paramètres (**parameter procedures**) pour calculer les valeurs des paramètres (**parameter values**).

- L'inférence ***check*** vérifie les contraintes après le calcul des nouvelles valeurs de paramètres. L'inférence utilise les valeurs des paramètres (**parameter values**), les procédures de vérification de contraintes (**constraint procedures**), et les relations de dépendance entre les contraintes (**constraint dependency relations**) pour calculer les résultats de vérification de contraintes (**constraint results**).

- L'inférence ***select violated constraint*** choisit une contrainte violée. Elle utilise les résultats de vérification de contraintes (**constraint results**) et les relations de dépendance entre les réparations (**fix dependency relations**) pour générer une contrainte violée sélectionnée (**selected violated constraint**).

- L'inférence ***revise*** traite une contrainte violée. L'inférence utilise la contrainte violée sélectionnée (**selected violated constraint**), les réparations (**fixes**) et les procédures de réparation (**fix procedures**) pour réparer la violation de contrainte, en proposant d'autres valeurs des paramètres (**parameter values**).

La structure d'inférence présente une vision générale d'une méthode de résolution de problèmes et les connaissances utilisées par les inférences. Cependant, cette description est trop abstraite et imprécise. L'information sur la structure des rôles des connaissances n'est pas dévoilée. Par exemple, les paramètres sont liés entre eux par des relations de dépendance, mais il n'y a pas d'information sur l'organisation de ces relations. De plus, nous ne savons pas si un paramètre a plus d'une contrainte, ou si plus d'une réparation est associée à une contrainte.

Dans la prochaine section, nous proposons une approche pour décrire les rôles des connaissances d'une méthode en termes d'une ontologie de la méthode et des inférences. L'ontologie précise les suppositions sur la structure des rôles des connaissances demandée par la méthode. Les inférences sont définies en termes de cette ontologie. L'avantage de cette description est d'avoir une meilleure compréhension de la méthode, en faisant abstraction des détails d'implantation.

5.3.2 Ontologie de méthode et inférences de P&R

Dans cette section, nous représentons l'ontologie de la méthode et les inférences de P&R en K-Loom. Nous décrivons les rôles des connaissances de la structure d'inférence, en explicitant les suppositions sur les connaissances utilisées par la méthode. Après avoir décrit les rôles des connaissances, nous décrirons les inférences selon les rôles des connaissances définis.

L'inférence *select parameter*

L'inférence ***select parameter*** choisit parmi les paramètres un paramètre dont on veut calculer la valeur. La figure 5.2 illustre cette inférence et les rôles des connaissances: les paramètres d'entrée (**input parameters**), les relations de dépendance entre les paramètres (**parameter dependency relations**) et le paramètre sélectionné (**selected parameter**).

Les paramètres d'entrée doivent être fournis par l'utilisateur au départ. Dans la tâche VT, les valeurs d'entrée donnent les caractéristiques de base du système d'ascenseur, comme **door opening type** (`center`, pour les portes qui s'ouvrent au centre; `side`, pour les portes qui s'ouvrent d'un côté de la cabine), **car capacity** (le poids maximal des occupants que le système peut supporter), etc. Ces valeurs sont utilisées pour le calcul des autres paramètres.

Parmi les relations de dépendance entre les paramètres, on retrouve, par exemple, que **sling underbeam**[3] est égal à **car cab height** plus **sling underbeam space**. Cette

3. distance entre la plataforme et la partie supérieure de la cabine de l'ascenseur

formule exprime que **sling underbeam** dépend de **car cab height** et de **sling underbeam space**. Les relations de dépendance entre les paramètres constituent un aspect fondamental de la structure de connaissances pour P&R. Dans ce cas, au lieu d'une structure hiérarchique de composants et sous-composants (section 2.1.1), les paramètres sont organisés en un réseau qui modélise leurs interdépendances. Ainsi, pour utiliser P&R, le domaine d'application doit présenter ce genre d'organisation ou doit y être adapté. Nous faisons alors quelques suppositions sur les relations de dépendance entre les paramètres:

- Un paramètre p_i dépend d'un paramètre p_j, si p_j est utilisé pour calculer la valeur de p_i.
- Un paramètre p_i dépend indirectement d'un paramètre p_k, si p_i dépend de p_j et p_j dépend directement ou indirectement de p_k.
- Un paramètre d'entrée p_i ne dépend pas d'autres paramètres.

De plus, il ne peut exister de cycles dans les relations de dépendance:

- Si un paramètre p_i dépend d'un paramètre p_j, alors p_j ne dépend pas de p_i.
- Si un paramètre p_i dépend indirectement de p_k, alors p_i ne dépend pas de p_k.

Si les valeurs des paramètres ne sont pas fournies par l'utilisateur, elles sont calculées par la méthode. Dans P&R, chaque paramètre a une valeur unique. Un paramètre sélectionné n'a pas encore sa valeur calculée, mais tous les paramètres desquels il dépend sont déjà calculés.

La figure 5.3 présente une définition partielle de l'ontologie de la méthode de P&R et de l'inférence ***select parameter***. (1)[4] définit la classe des paramètres. Chaque paramètre a une valeur unique et au maximum une procédure associée. Les autres attributs tels quels `Max-Parameter-Value`, `Min-Parameter-Value` et `List-Upgrade-Values` seront utilisés dans le cadre de l'inférence ***revise*** (section 5.3.2). (2) définit le concept

4. Les chiffres sont mis pour référence seulement et ils ne font pas partie de la représentation.

```
(1) (defconcept Parameter :is-primitive (:and
     (:exactly 1 Has-Value)
     (:exactly 1 Parameter-Procedure)
     (:exactly 1 Max-Parameter-Value)
     (:exactly 1 Min-Parameter-Value)
     (:exactly 1 List-Upgrade-Values)))
(2) (defconcept Parameter-Value)
(3) (defrelation Has-Value
      :domain Parameter
      :range  Parameter-Value)
(4) (defconcept Input-Parameter:is-primitive Parameter)
(5) (defrelation Parameter-Depends-On
      :domain Parameter
      :range  Parameter)
(6) (defrelation Asymmetric-Depends-On :is (:satisfies (?p1 ?p2)
      (:and (Parameter-Depends-On ?p1 ?p2)
            (:not (Parameter-Depends-On ?p2 ?p1)))))
(7) (defrelation Weak-Transitive-Depends-On :is (:satisfies (?p1 ?p2)
      (:for-some ?p3 (:and (Parameter-Depends-On ?p1 ?p3)
                           (Parameter-Depends-On ?p3 ?p2)
                           (:not (:same-as ?p1 ?p2))
                           (:not (Parameter-Depends-On ?p2 ?p1))))))
(8) (defconcept Selected-Parameter :is-primitive Parameter)
(9) (definference Select-Parameter
     :cond
       (:for-some ?p1
          (:and (Parameter ?p1)
                (:not (:for-some ?v1 (Has-Value ?p1 ?v1)))
                (:for-all ?p2 (:implies (Parameter-Depends-On ?p1 ?p2)
                                        (:for-some ?v2 (Has-Value ?p2 ?v2))))))
     :body (Select-Parameter ?p1)
     :result (Selected-Parameter ?p1))
```

FIG. 5.3 - *Définition de l'inférence* ***select parameter*** *et ses rôles des connaissances.*

de valeur de paramètre et (3) définit la relation qui associe une valeur unique à chaque paramètre. (4) définit le concept de paramètre d'entrée. (5) exprime les relations de dépendance entre les paramètres. (6) et (7) décrivent les dépendances directe et indirecte entre les paramètres. La relation ASYMMETRIC-DEPENDS-ON signifie que si $?p1$ dépend de $?p2$, alors $?p2$ ne dépend pas de $?p1$, de sorte qu'il n'existe pas de cycles dans les relations de dépendance directe. Pour définir les relations de dépendance indirecte entre les paramètres, la relation WEAK-TRANSITIVE-DEPENDS-ON signifie que, si $?p1$ dépend de $?p3$, et que $?p3$ dépend de $?p2$, alors $?p1$ dépend de $?p2$. La relation est appelée transitive "faible" parce que $?p1$ doit être différent de $?p2$. Le fait que $?p1$ ne dépend pas de $?p2$ garantit qu'il n'existe pas de cycles dans les relations de dépendance indirecte. (8) définit le rôle de sortie: le paramètre sélectionné. (9) présente l'inférence ***select parameter*** qui choisit un paramètre dont la valeur n'a pas encore été calculée, mais dont tous les paramètres desquels il dépend sont déjà calculés.

L'inférence *propose*

P&R donne une valeur initiale à tous les paramètres sélectionnés. Cette valeur peut être ensuite modifiée pour résoudre des violations de contraintes. L'inférence ***propose*** calcule la valeur du paramètre sélectionné. La figure 5.2 illustre cette inférence avec ses rôles des connaissances: le paramètre sélectionné (**selected parameter**), les procédures associées aux paramètres (**parameter procedures**) et les valeurs des paramètres (**parameter values**).

Le paramètre sélectionné et les valeurs des paramètres sont les rôles des connaissances décrits dans la section précédente. Les procédures associées aux paramètres calculent les valeurs initiales des paramètres. Tous les paramètres ont une seule procédure, sauf les paramètres d'entrée, qui n'en ont aucune. Dans l'application VT, par exemple, la procédure pour calculer la valeur de **sling-underbeam** est égale à la valeur de **car cab height** plus la valeur de **sling underbeam space**.

La figure 5.4 présente la définition de l'inférence ***propose*** et ses rôles des connaissances. (10) définit la classe de procédures. (11) définit la relation qui associe une

```
(10) (defconcept Procedure)
(11) (defrelation Parameter-Procedure
       :domain Parameter
       :range  Procedure)
(12) (definference Propose
      :cond (:for-some ?p (Selected-Parameter ?p))
      :body (Propose ?p)
      :result (:for-some ?v (Has-Value ?p ?v)))
```

FIG. 5.4 - *Définition de l'inférence* ***propose*** *et de ses rôles des connaissances.*

procédure à chaque paramètre. (12) définit l'inférence ***propose*** qui calcule la valeur du paramètre sélectionné.

L'inférence *check*

L'inférence ***check*** examine les contraintes dont les paramètres ont été déjà calculés, et dont tous les paramètres desquels ils dépendent ont été déjà calculés. La figure 5.2 illustre l'inférence et ses rôles des connaissances: les valeurs des paramètres (**parameter values**), les procédures de vérification de contraintes (**constraint procedures**, les relations de dépendance entre les contraintes (**constraint dependency relations**) et les résultats de vérification de contraintes (**constraint results**).

Les contraintes limitent les valeurs de paramètres. Tous les paramètres n'ont pas nécessairement des contraintes, et certains peuvent même avoir plus d'une contrainte. Dans la tâche VT, le paramètre **car overtravel** a deux contraintes: une contrainte qui limite sa valeur maximale, et une autre qui limite sa valeur minimale.

Les procédures de vérification de contraintes déterminent si une contrainte est satisfaite. Les contraintes établissent des relations de dépendance entre les paramètres. Chaque procédure contient une expression qui calcule les résultats de vérification de contraintes, qui peuvent être "non violée ("not-violated") (si la contrainte est satisfaite) ou "violée ("violated") (si la contrainte n'est pas satisfaite). Dans l'exemple VT, le paramètre **car overtravel** a la contrainte **minimum car overtravel**, laquelle a une procédure qui évalue si la valeur de **car overtravel** est au moins la valeur

de **counterweight runby** plus 1.5 fois la valeur de **counterweight buffer stroke** plus 24 pouces. Cette contrainte dépend de la valeur des paramètres **car overtravel**, **counterweight runby** et **counterweight buffer stroke**.

```
(13) (defconcept Constraint :is-primitive (:and
       (:exactly 1 Constraint-Result)
       (:exactly 1 Constraint-Procedure)
       (:exactly 1 Constraint-Priority)))
(14) (defrelation Constraint-Result
       :domain Constraint
       :range  (:one-of 'VIOLATED 'NOT-VIOLATED))
(15) (defrelation Constraint-Priority
      :domain Constraint
      :range  (:through 1 10))
(16) (defrelation Has-Constraint
       :domain Parameter
       :range Constraint)
(17) (defrelation Constraint-Depends-On
       :domain Constraint
       :range  Parameter)
(18) (definference Check
      :cond (:for-some ?ct
              (:and (Constraint ?ct)
                    (:not (:for-some ?v (Constraint-Result ?ct ?v)))
                    (:for-all ?p (:implies (Constraint-Depends-On ?ct ?p)
                                           (:for-some ?v (Has-Value ?p ?v))))))
      :body (Check ?ct)
      :result (:for-some ?v (Constraint-Result ?ct ?v)))
```

FIG. 5.5 - *Définition de l'inférence* ***check*** *et ses rôles des connaissances.*

La figure 5.5 montre la définition de l'inférence ***check*** et ses rôles des connaissances. (13)-(15) spécifient le concept **Constraint** et ses attributs uniques **Constraint-Result**, **Constraint-Procedure** et **Constraint-Priority** (utilisé dans le cadre de l'inférence ***select violated constraint***). (16) définit la relation qui associe les paramètres et contraintes. (17) définit la relation de dépendance entre contraintes et paramètres. (18) définit l'inférence ***check*** qui vérifie la contrainte dont tous les paramètres desquels la contrainte dépendent sont déjà calculés.

L'inférence *select violated constraint*

Si plusieurs contraintes sont violées au même moment, l'inférence ***select violated constraint*** choisit une contrainte à réviser. La figure 5.2 montre cette inférence avec ses rôles des connaissances: les résultats de vérification de contraintes (**constraint results**), les relations de dépendance entre les réparations (**fix dependency relations**), et la contrainte violée sélectionnée (**selected violated constraint**).

Les contraintes ont des mécanismes associés pour réparer leur violation. Une contrainte peut avoir plus d'une réparation, chacune modifiant la valeur d'un paramètre dans le but de réparer la violation. En plus du paramètre modifié, la réparation peut faire référence à d'autres paramètres pour accomplir la modification. Ainsi, les relations de dépendance entre les réparations établissent les dépendances entre les réparations et les contraintes. Dans la tâche VT, par exemple, la contrainte **minimum platform to hoistway left** exprime que **platform to hoistway left** doit être au moins huit pouces. Si ce n'est pas le cas, deux réparations sont associées à la contrainte: (1) augmenter la valeur de **opening to hoistway left** en y ajoutant l'écart entre la valeur normale et la valeur de violation, et (2) diminuer la valeur de **car return left** en y enlevant l'écart entre la valeur normale et la valeur de violation. Dans ce cas, les réparations dépendent des valeurs des paramètres **opening to hoistway left**, **car return left** et **platform to hoistway left**.

La contrainte violée sélectionnée est une contrainte dont toutes les réparations peuvent être appliquées. Une réparation peut être appliquée quand toutes les valeurs dont dépendent les réparations ont déjà été calculées. Comme il peut y avoir plus d'une contrainte violée qui satisfait ce critère, l'inférence ***select violated constraint*** doit sélectionner seulement une contrainte. Dans notre modélisation de P&R, la contrainte avec la priorité la plus haute est choisie. Dans la tâche VT, il n'existe pas de priorité associée aux contraintes: Si plusieurs peuvent être traitées en même temps, une contrainte est choisie arbitrairement. La seule exception est lorsque les contraintes associées aux paramètres **machine groove pressure** et **hoist cable traction ration** sont violées en même temps. Dans ce cas, la contrainte associée à **machine groove**

```
(19) (defconcept Fix :is-primitive (:and
       (:exactly 1 Modified-Parameter)
       (:exactly 1 Fix-Procedure)
       (:exactly 1 Fix-Cost)
       (:exactly 1 Fix-Type)
       (:exactly 1 Fix-Step)))
(20) (defrelation Modified-Parameter
       :domain Fix
       :range  Parameter)
(21) (defrelation Fixes-Constraint
       :domain Fix
       :range  Constraint)
(22) (defrelation Fix-Depends-On
       :domain Fix
       :range  Parameter)
(23) (defrelation Constraint-Priority
       :domain Constraint
       :range  INTEGER)
(24) (defconcept Selected-Violated-Constraint :is-primitive Constraint)
(25) (definference Select-Violated-Constraint
      :cond
       (:for-some ?ct1
         (:for-some ?pr1
           (:and (Constraint-Result ?ct1 'VIOLATED)
                 (Constraint-Priority ?ct1 ?pr1)
                 (:for-all ?f (:implies
                    (Fixes-Constraint ?f ?ct1)
                    (:for-all ?p (:implies (Fix-Depends-On ?f ?p)
                                           (:for-some ?v (Has-Value ?p ?v))))))
                 (:for-all ?ct2 (:implies
                     (:and (Constraint-Result ?ct2 'VIOLATED)
                           (:not (:same-as ?ct1 ?ct2)))
                     (:for-some ?pr2
                        (:and (Constraint-Priority ?ct2 ?pr2)
                              (:or (> ?pr1 ?pr2) (= ?pr1 ?pr2)))))))))
      :body (Select-Violated-Constraint ?ct1)
      :result (Selected-Violated-Constraint ?ct1))
```

FIG. 5.6 - *Définition de l'inférence* ***select violated constraint*** *et ses rôles des connaissances.*

pressure doit être révisée en premier. Ici, nous avons choisi d'assigner une priorité à chaque contrainte, parce cela nous permet de définir l'inférence ***select violated constraint*** indépendamment de la tâche VT et, ainsi, généraliser P&R pour d'autres domaines.

La figure 5.6 illustre la définition de l'inférence ***select violated constraint*** et de ses rôles des connaissances. (19) définit la réparation et ses attributs. Les attributs `Fix-Procedure`, `Fix-Cost`, `Fix-Type` et `Fix-Step` sont utilisés dans la définition de l'inférence ***revise***, expliqué dans la prochaine section. (20)-(23) représentent les rôles des connaissances d'entrée. (24) définit le rôle de connaissance de sortie. (25) signifie que l'inférence ***select violated constraint*** sélectionne une contrainte violée dont toutes les réparations ont les valeurs de paramètres dont elles dépendent déjà calculées. De plus, si plus d'une contrainte satisfait cette condition, l'inférence choisit la contrainte de plus haute priorité.

L'inférence *revise*

Cette inférence est la plus complexe de la méthode. Quelques modélisations de P&R [Motta et al., 1994, Fensel, 1995a] ont décomposé cette inférence en une autre structure d'inférence. Ici, nous avons choisi de maintenir l'inférence ***revise*** comme élémentaire, sans la décomposer.

L'inférence ***revise*** répare une violation de contrainte. La figure 5.2 illustre l'inférence avec ses rôles des connaissances: la contrainte violée sélectionnée (**selected violated constraint**), les réparations (**fixes**), les procédures de réparation (**fix procedures**), et les valeurs des paramètres (**parameter values**).

L'inférence ***revise*** répare la violation d'une contrainte par l'application des réparations séparément ou par leur combinaison, jusqu'à ce que la contrainte ne soit plus violée. Les procédures de réparation accomplissent les modifications des valeurs des paramètres. Toutes les réparations ont une seule procédure qui est représentée par une expression. L'ordre d'application des réparations est basé sur leur coût. Dans l'application VT, la contrainte **maximum hoist cable traction ratio** a trois réparations

associées avec trois coûts différents:

1. Diminuer la distance de **counterweight to platform rear** par décrément d'un demi pouce (coût 3).

2. Augmenter le **car supplement weight** par incrément de 100 pouces (coût 4).

3. Modifier le **compensation cable model** selon le prochain modèle supérieur (coût 6).

Pour essayer de réparer la violation de la contrainte **maximum hoist cable traction ratio**, les réparations seront appliqués dans l'ordre suivant: 1, 2, 1 2, 3, 1 3, 2 3, 1 2 3.

En plus de son coût, chaque réparation a un type qui caractérise le type de modification effectuée. Il y a quatre types de réparations: (1) réparations du type `change-value` qui changent la valeur d'un paramètre une seule fois; (2) réparations du type `increment` qui augmentent graduellement la valeur d'un paramètre; (3) réparation du type `decrement` qui diminuent graduellement la valeur d'un paramètre; (4) réparations du type `upgrade` qui modifient le modèle d'un paramètre selon une liste ordonnée des valeur prédéfinies. Les réparations du type `increment`, `decrement` et `upgrade` sont des réparations *récurrentes*, parce qu'elles peuvent être appliquées plusieurs fois. Dans la tâche VT, nous retrouvons les types de réparations suivants:

- Soustraire à **car return left** la valeur de l'écart entre la valeur normale et la valeur de violation (réparation du type `change-value`).

- Augmenter **counterweight buffer blocking height** par incrément d'un pouce (réparation du type `increment`, pas = 1).

- Diminuer **counterweight plate depth** par décrément d'un demi pouce (réparation du type `decrement`, pas = 0.5).

- Modifier **compensation cable model** (réparation du type `upgrade`).

```
(26) (defrelation Fix-Procedure
       :domain Fix
       :range  Procedure)
(27) (defrelation Fix-Cost
       :domain Fix
       :range  (:through 1 10))
(28) (defset Type-Fix :is
       (:one-of :Change-Value :Increment :Decrement :Upgrade))
(29) (defconcept Fix-Type
       :domain Fix
       :range  Type-Fix)
(30) (defconcept Fix-Step
       :domain Fix
       :range  Number)
(31) (defrelation Max-Parameter-Value
       :domain Parameter
       :range  Parameter-Value)
(32) (defrelation Min-Parameter-Value
       :domain Parameter
       :range  Parameter-Value)
(33) (defrelation List-Possible-Parameter-Values :is
       :domain Parameter
       :range  List)
(34) (definference Revise
      :cond (:for-some ?ct (Selected-Violated-Constraint ?ct))
      :body (Revise ?ct)
      :result (Constraint-Result ?ct 'NOT-VIOLATED))
```

FIG. 5.7 - *Définition de l'inférence* **revise** *et ses rôles des connaissances.*

Lorsqu'une combinaison de réparations contient une ou plusieurs réparations qui doivent être appliquées plus d'une fois, toutes les combinaisons doivent être essayées avant de passer à une autre. Pour limiter le nombre de fois que chaque modification doit être appliquée, on établit une valeur maximale et une valeur minimale pour les modifications du type `increment` et `decrement` et une liste de tous les modèles possibles pour les modifications du type `upgrade`. L'association de ces valeurs aux paramètres à modifier évite l'application infinie de ces réparations. Dans la description de Yost, pour limiter le nombre de fois que les réparations récurrentes sont appliquées, des contraintes spéciales sont définies pour les paramètres modifiés des réparations récurrentes. Ces contraintes sont spéciales, parce qu'aucune réparation ne lui est associée. Ici, nous proposons d'associer des valeurs délimitantes aux paramètres modifiées de manière à éclairer cette distinction.

La figure 5.7 représente la définition de l'inférence ***revise*** et de ses rôles des connaissances. Les définitions (26)-(33) dépeignent les rôles des connaissances. Dans (34), l'inférence ***revise*** répare la contrainte violée sélectionnée.

5.3.3 Définition de la tâche de P&R

Après avoir décrit l'ontologie de la méthode et les inférences, nous allons maintenant définir la tâche qui impose un contrôle sur l'activation des inférences.

Nous allons voir ici que les inférences étant précisées, la définition de la tâche est relativement facile. En général, différents contrôles peuvent être associés à une méthode. K-Loom permet de tester l'exécution des structures de tâches et de les évaluer en termes des résultats et d'efficacité. En analysant les résultats, nous pouvons décider quel contrôle obtient les meilleures solutions. D'un autre côté, si nous analysons l'efficacité, nous pouvons déterminer le contrôle qui obtient les résultats le plus rapidement. Suivant le domaine, on choisira l'un ou l'autre ou un compromis entre les deux.

Pour la méthode P&R, nous présentons dans la figure 5.8 une définition de la structure de la tâche faite selon KADS. Le but de la tâche est d'assigner des valeurs

aux paramètres d'un système. L'entrée comprend les informations fournies par l'utilisateur. La sortie donne les valeurs des paramètres calculées. Le flux de contrôle de la structure de la tâche indique que les inférences ***select parameter*** et ***propose*** retournent un seul paramètre et une seule valeur à la fois. Ensuite, toutes les contraintes qui peuvent être traitées sont vérifiées par l'inférence ***check***. Finalement, chacune à son tour, les contraintes violées sont sélectionnées et réparées par les inférences ***select violated constraint*** et ***revise***.

```
task Propose-and-Revise
    goal
        assigne des valeurs à tous les paramètres d'un système
    input
        les informations fournies par l'utilisateur
    output
        les valeurs de paramètres de sortie
    task-structure
        REPEAT
            select-parameter
            propose
            REPEAT
                check
            UNTIL (toutes les contraintes qui peuvent être vérifiées le soient)
            REPEAT
                select-violated-constraint
                revise
            UNTIL (il n'y aient plus de contraintes violées)
        UNTIL (les valeurs de tous les paramètres sont calculées)
```

FIG. 5.8 - *Définition de la tâche P&R selon KADS. Dans cette structure de contrôle les inférences* ***propose*** *et* ***revise*** *s'alternent.*

Dans la figure 5.9, nous montrons la représentation en K-Loom qui correspond au modèle conceptuel de P&R en KADS. Cette représentation est aussi une opérationalisation du modèle conceptuel. Au lieu de phrases en langue naturelle, nous avons les requêtes Loom qui précisent le sens de la représentation informelle. Ainsi, l'argument : `input` définit que tous les paramètres d'entrée doivent avoir une valeur associée avant l'exécution de la tâche. L'argument : `output` définit qu'après l'exécu-

```
(deftask Propose-and-Revise
   :input (:for-all ?p (:implies (Input-Parameter ?p)
                                 (:for-some ?v (Has-Value ?p ?v))))
   :output (:and (:for-all ?p  (:implies (Parameter ?p)
                                         (:for-some ?v (Has-Value  ?p ?v))))
                 (:for-all ?ct (:implies (Constraint ?ct)
                                         (Constraint-Result ?ct 'NOT-VIOLATED))))
   :task-structure
    ((while (:for-some ?p (:and (Parameter ?p)
                                (:not (:for-some ?v (Has-Value ?p ?v)))))
       (Select-Parameter)
       (Propose)
       (while (:for-some ?ct
                (:and (Constraint ?ct)
                      (:not (:for-some ?v (Constraint-Result ?ct ?v)))
                      (:for-all ?p
                        (:implies (Constraint-Depends-On ?ct ?p)
                                  (:for-some ?v (Has-Value ?p ?v))))))
         (Check))
       (while (:for-some ?ct
                (:and (Constraint-Result ?ct 'VIOLATED)
                      (:for-all ?f
                        (:implies (Fixes-Constraint ?f ?ct)
                                  (:for-all ?p
                                    (:implies (Fix-Depends-On ?f ?p)
                                              (:for-some ?v
                                                (Has-Value ?p ?v))))))))
         (Select-Violated-Constraint)
         (Revise)))))
```

FIG. 5.9 - *Définition de la tâche P&R en K-Loom. Cette représentation correspond à la définition en KADS montrée dans la figure 5.8. Ici, nous avons une représentation plus précise des conditions d'exécution des inférences en termes de rôles des connaissances définis dans l'ontologie de la méthode. Les inférences sont en italiques et les rôles des connaissances sont en gras.*

tion de la tâche tous les paramètres doivent avoir une valeur et aucune contrainte ne doit être violée. Le flux de contrôle défini par l'argument :`task-structure` décrit une boucle principale où les commandes sont répétées jusqu'à ce que les tous les paramètres aient une valeur associée. Dans la boucle principale, les paramètres sont tour à tour sélectionnés et leurs valeurs calculées. Ensuite, une autre boucle vérifie toutes les contraintes qui doivent être vérifiées après le calcul du paramètre sélectionné. Une contrainte doit être vérifiée quand elle n'a pas encore de résultat associé et quand toutes les valeurs dont elle dépend sont déjà calculées. Finalement, une autre boucle sélectionne et révise, chacune à son tour, les contraintes violées qui peuvent être révisées. Une contrainte violée peut être révisée quand tous les réparations associées peuvent être appliquées. Une réparation peut être appliquée quand tous les valeurs dont elle dépend sont déjà été calculées.

Ce qu'il faut noter, c'est que notre représentation en K-Loom est très semblable à la représentation conceptuelle en KADS. Nous avons essayé de minimiser l'écart entre le langage conceptuel de KADS et K-Loom. De plus, nous avons l'avantage d'une représentation à la fois précise et opérationnelle. On remarquera aussi que les conditions sont exprimées en Loom et qu'elles font référence aux rôles des connaissances définis dans l'ontologie de la méthode. De cette façon, nous avons l'indépendance entre les couches, mais en même temps, une couche peut toujours faire référence aux connaissances définies dans les autres couches. La couche de la tâche fait référence aux rôles des connaissances (ontologie de la méthode) et aux inférences définies dans la couche des inférences. Nous avons donc l'autonomie de chaque couche et, à la fois, une cohérence entre leurs représentations.

Une définition alternative de la tâche pour P&R

Maintenant, nous allons illustrer la définition d'un flux de contrôle différent pour la méthode P&R. Notre but n'est pas de faire une étude approfondie de P&R. Cela a été déjà fait par d'autres groupes de recherche. Par exemple, [Fensel, 1995a] et [Motta et al., 1994] proposent des flux de contrôle alternatifs pour la méthode P&R

appliquée à la tâche VT. Une analyse profonde du domaine a été faite pour dégager des structures de contrôle plus efficaces, en considérant les relations entre paramètres, contraintes et réparations. Ici, nous allons nous contenter d'illustrer comment la modélisation en K-Loom facilite le changement d'ordre d'exécution des inférences et permet la modification et l'adaptation d'une méthode existante à différents problèmes et domaines.

Ainsi, avec les mêmes spécifications d'entrée et de sortie, la figure 5.10 illustre un flux de contrôle alternatif pour la méthode P&R en K-Loom. Nous avons supprimé les arguments `:input` et `:output` parce qu'ils sont identiques à ceux de la figure 5.9. La structure de la tâche, par contre, est différente: tous les paramètres sont calculés d'abord; ensuite, toutes les contraintes sont vérifiées; finalement, toutes les contraintes violées sont révisées.

Cette structure de contrôle est appelée [Motta et al., 1994] *Complete-Model-then-Revise* (CMR), par opposition à la structure de contrôle proposée dans le document qui décrit la tâche VT, présentée à la section 5.3.3, et appelée *Extend-Model-then-Revise* (EMR). D'après cette étude, l'implantation selon CMR exploite les interdépendances entre contraintes et entre contraintes et réparations de manière à minimiser le nombre restant de violations de contraintes. En plus, le modèle CMR trouve des solutions qui ne pourraient pas être obtenues à partir d'EMR. Nous n'avons pas analysé les différentes implantations de la structure de la tâche pour P&R, mais nous voulons seulement souligner la facilité d'apporter des modifications aux flux de contrôle en utilisant K-Loom. La structure de contrôle de la figure 5.10 est à la fois une définition précise et une opérationalisation de CMR. Ainsi, cette variation de P&R peut être facilement testée et validée pour la tâche VT ou pour d'autres domaines.

Il est également facile de configurer la méthode en changeant les inférences, de façon à modifier la méthode pour un nouveau problème de configuration. Imaginons une autre application où toutes les inférences définies peuvent être réutilisées (***select parameter***, ***propose***, ***check*** et ***select violated constraint***), sauf l'inférence ***revise***, parce qu'elle est trop spécifique à la tâche VT. Dans cette perspective, il sera possible de définir et implanter une nouvelle inférence ***revise*** et garder les autres

```
(deftask Propose-and-Revise-2
   task-structure:
   ((while (:for-some ?p (:and (Parameter ?p)
                               (:not (:for-some ?v (Has-Value ?p ?v)))))
      (Select-Parameter)
      (Propose))
    (while (:for-some ?ct
                  (:and (Constraint ?ct)
                        (:not (:for-some ?v (Constraint-Result ?ct ?v)))))
      (Check))
    (while (:for-some ?ct (Constraint-Result ?ct 'VIOLATED))
      (Select-Violated-Constraint)
      (Revise))))
```

FIG. 5.10 - *Définition alternative de P&R en K-Loom. Dans cette structure de contrôle tous les paramètres sont calculés avant la vérification et les réparations des contraintes. Les inférences sont en italiques et les rôles des connaissances sont en gras.*

inférences telles quelles. On peut aussi changer l'inférence ***select parameter*** de façon à associer des priorités aux paramètres. Ainsi, les paramètres seront sélectionnés non seulement parce qu'ils peuvent être calculés, mais parce qu'ils sont prioritaires. La priorité permet d'associer des connaissances heuristiques pour faire le choix du prochain paramètre à être sélectionné.

Ainsi, la spécification des inférences en fonction de l'ontologie de la méthode permet une vision plus modulaire de la méthode et rend plus flexible sa configuration pour différents tâches et domaines. Ce qui, à l'origine, n'était pas possible dans les méthodes à limitation de rôles.

Pour représenter la tâche VT dans notre système, nous avons réutilisé une ontologie du domaine de VT représentée en OCML, VT-OCML. OCML est un langage de modélisation selon le niveau des connaissances utilisé dans la méthodologie VITAL [Domingue et al., 1993] pour la construction des systèmes à base de connaissances. Dans l'annexe A, nous montrons une trace de l'exécution de P&R en K-Loom appliquée à la tâche VT, en réutilisant les données de l'ontologie VT-OCML.

Dans le cas de P&R appliquée au domaine VT, il y a une correspondance directe

entre l'ontologie VT-OCML et notre ontologie de la méthode, où il suffit de renommer les concepts de l'ontologie VT-OCML en termes de l'ontologie de la méthode. Dans le prochain chapitre, nous montrons un autre exemple. où une ontologie du domaine ne correspond pas exactement à l'ontologie de méthode. Nous introduisons des opérateurs de correspondance pour associer des connaissances du domaine à une ontologie de méthode.

5.4 Discussion

Nous avons illustré notre approche pour décrire les méthodes de résolution de problèmes à l'aide de l'ontologie de la méthode. Pour y arriver, nous avons montré une définition originale des rôles des connaissances et des inférences de la méthode P&R, en proposant un niveau intermédiaire de description, entre le niveau conceptuel et le niveau d'implantation.

L'ontologie de la méthode pour P&R est formée par toutes les définitions de classes et de relations présentées dans les sections précédentes. Cette ontologie n'est pas une ontologie définitive; elle reste encore ouverte à des extensions et à des modifications. Nous verrons dans la prochaine section, où nous comparons notre travail avec d'autres travaux, que dans notre définition de la méthode P&R, il manque une caractérisation plus précise de la solution obtenue par P&R.

La définition de l'ontologie de la méthode donne à la couche des inférences de KADS le même pouvoir d'expression pour décrire les rôles des connaissances que celui dans la couche du domaine pour décrire les connaissances du domaine. Dans la définition originale de KADS, les rôles étaient décrits comme des étiquettes qui ne spécifient pas la structure inhérente aux connaissances utilisées par la méthode. C'était le modèle du domaine, dans la couche du domaine, qui était responsable pour la description des connaissances en fonction de leur utilisation. L'avantage d'avoir l'ontologie de la méthode dans la couche des inférences et le modèle du domaine dans la couche du domaine est de permettre d'exprimer, dans chaque couche, les particularités associées soit à la méthode, soit au domaine.

En utilisant K-Loom pour définir une méthode, nous avons les avantages d'un langage terminologique, ce qui permet d'enlever les ambiguïtés des spécifications en langue naturelle ou même dans un langage conceptuel; le langage terminologique offre une richesse d'expression qui permet de dévoiler la structure inhérente des connaissances utilisées par la méthode. D'une part, un langage opérationnel offre un feedback rapide du modèle créé, en rendant possible sa modification directement. D'autre part, il est intéressant d'avoir les trois catégories de connaissances représentées avec le même langage, tout en gardant leur indépendance.

Dans les langages terminologiques, les définitions et leurs conséquences logiques sont propagées aux instances, de manière à que les instances aient les mêmes propriétés que les concepts (chapitre 4). Cela permet la vérification et la validation quand différents composants sont assemblés. En K-Loom, quand l'ontologie de la méthode est instanciée avec un domaine d'application, toutes les contraintes de définitions doivent être vérifiées avant l'exécution des inférences. Cela permet une vérification automatique si le domaine d'application correspond aux rôles des connaissances de la méthode.

De même, nous avons montré comment il est facile de configurer différents flux de contrôle pour la méthode par la définition de la tâche, en changeant l'ordre d'exécution des inférences. Cela est possible parce que les inférences sont bien spécifiées. Par conséquent, on peut imaginer de configurer la méthode pour différents problèmes en changeant quelques inférences ou en les remplaçant par d'autres.

Toutefois, l'ontologie de la méthode et la définition des inférences n'est qu'un aspect parmi d'autres constituant la spécification d'une méthode de résolution de problèmes. La méthode P&R est une méthode très spécifique, ce qui restreint beaucoup sa réutilisation, comme nous le verrons dans les analyses faites sur cette méthode. La figure 5.1 présente une vue très simplifiée de la procédure d'indexation et de récupération des méthodes réutilisables. Dans cette thèse, nous ne nous attardons pas sur le problème de l'organisation et de l'indexation d'une bibliothèque de méthodes ou de composants réutilisables. Il mérite à lui seul une recherche approfondie (cf. [Breuker et Van de Velde, 1994]).

5.4.1 Comparaison avec des travaux connexes

Analyses des connaissances de VT

Plusieurs analyses [Motta et al., 1994, Fensel, 1995a] ont montré que la méthode P&R présente des caractéristiques très particulières qui limitent sa généralisation à d'autres problèmes de configuration. Cela est dû premièrement au fait que cette méthode a été conçue pour résoudre une application spécifique, la configuration VT. Ce domaine présente des connaissances heuristiques très particulières qui empêchent les distinctions conceptuelles claires et une généralisation de la méthode pour d'autres applications.

La spécification de P&R a été construite à partir d'une description faite par un expert du domaine [Yost, 1994]. L'expert a inclus des heuristiques obtenues à partir de l'expérience qui donnent un aspect "compilé" aux connaissances de VT. Un exemple de connaissances heuristiques dans VT sont les procédures qui donnent les valeurs initiales aux paramètres. Ces procédures génèrent des valeurs initiales de façon à donner la "bonne" solution du premier coup, en minimisant le nombre de contraintes violées [Fensel et Straatman, 1996]. Un autre exemple sont les combinaisons de réparations utilisées pour modifier la valeur de plusieurs paramètres, quand l'application d'une seule procédure de réparation ne peut pas traiter la violation de contrainte [Motta et Zdrahal, 1996].

L'avantage de ces connaissances heuristiques est l'efficacité de la méthode. Les connaissances spécifiques du domaine guident et limitent l'espace de recherche. Toutefois, P&R manque de flexibilité pour être une méthode générale de configuration. De plus, la définition des rôles des connaissances n'est pas complète, parce qu'elle ne montre pas toutes les distinctions conceptuelles nécessaires à la compréhension de la méthode.

Ici, nous résumons quelques résultats d'analyses sur P&R qui montrent des aspects de cette méthode:

- L'interaction entre les contraintes et les procédures de réparation n'est pas explicitée. Il faut "croire" que les contraintes n'interagissent pas entre elles. Comme

les étapes *propose* et *revise* alternent, s'il y avait beaucoup d'interaction entre les contraintes, il faudrait toujours recalculer les valeurs des paramètres à chaque violation et réparation de contrainte [Fensel et Straatman, 1996].

- La caractérisation de l'ensemble de contraintes n'est pas explicitée. On ne sait pas si l'ensemble de contraintes est complet, c'est-à-dire si toutes les solutions générées sont valides [Motta et Zdrahal, 1995].

- Le type de solution obtenue n'est pas défini. On ne sait pas si la méthode génère une solution optimale ou non, ou si une solution est meilleure qu'une autre. La seule information disponible est que la solution ne viole pas les contraintes définies [Fensel, 1995a]. Dans [Motta et Zdrahal, 1995], P&R est caractérisée comme une méthode qui obtient une solution optimale.

- On ne sait pas ce qui arrive quand la méthode ne peut pas réparer une violation de contrainte. Il n'est pas possible de faire un retour en arrière et donner de nouvelles valeurs aux paramètres dans un ordre différent pour essayer d'obtenir une nouvelle solution [Fensel, 1995a]. Ainsi, l'ordre d'application des réparations est très important parce qu'il peut empêcher de trouver une solution ou peut trouver une solution non optimale [Motta et al., 1994].

- La notion de coût dans P&R n'est pas claire. Une étude montre que le coût dans P&R est associé à la distance entre la solution obtenue et la solution idéale [Motta et Zdrahal, 1996]. Cette distance est mesurée par la somme de toutes les réparations requises pour produire la solution. Ainsi, la notion de coût de P&R est très complexe, parce qu'elle mesure le coût de la solution et le coût de maintenance. Dans ce cas, le coût n'est pas simplement la somme des coûts individuels de chaque composant de la solution.

Toutes ces informations ne sont pas explicites dans la description faite par l'expert. Ces informations implicites ou manquantes montrent la nature compilée des connaissances utilisées par P&R. Cela a été confirmé par la comparaison entre P&R

et d'autres méthodes plus générales de configuration. Par exemple, lorsque P&R est comparée avec la méthode *Générer et Tester* (G&T) [Fensel et Straatman, 1996], on peut remarquer comment dans G&T les distinctions conceptuelles des rôles des connaissances sont beaucoup plus claires que dans P&R (section 7.1). Un travail récent [Motta et Zdrahal, 1996] décrit un modèle plus général pour ce type de problème, à partir d'études comparatives avec P&R et avec d'autres méthodes de configuration. Ce modèle générique identifie et détaille les éléments conceptuels qui spécifient une méthode pour la configuration paramétrique.

Ainsi, ces travaux ont démontré que l'utilisation des connaissances heuristiques de VT empêche l'explicitation de toutes ces informations. On peut comparer ces analyses à un travail de décompilation de connaissances, semblable sous certains aspects à la décompilation des connaissances des systèmes de première génération. Les connaissances heuristiques spécifiques au domaine de P&R limitent la réutilisation de P&R pour d'autres tâches de configuration. Nous discutons la réutilisation des méthodes de résolution de problèmes dans le chapitre 7.

Typologie de suppositions

Pour vérifier si un problème peut être résolu par une méthode, il faut analyser si les conditions préalables pour utiliser la méthode sont satisfaites par l'application en question. Des travaux récents en ingénierie de connaissances revendiquent l'importance d'expliciter les conditions requises pour utiliser une méthode. Ces conditions sont appelées de différentes manières selon les approches: *caractéristiques* ("features") [Steels, 1990, Breuker et Van de Velde, 1994], *suppositions* ("assumptions") [Akkermans et al., 1993, Fensel, 1995a, Benjamins et al., 1996] et *ontologie de la méthode* [Musen et al., 1994], le terme choisi dans cette thèse.

Benjamins et Pierret-Golbreich [Benjamins et Pierret-Golbreich, 1996] proposent une organisation plus systématique des conditions d'application des méthodes de résolution de problèmes, appelées de *suppositions*. Ces suppositions sont organisées en termes d'une typologie qui caractérise les différents facettes d'une méthode. En

d'autres mots, cette typologie constitue un cadre conceptuel pour classifier les différents types de conditions d'applicabilité associées à une méthode. Selon cette organisation, une méthode peut être choisie si le problème considéré satisfait les conditions d'applicabilité explicitées par les différents types de suppositions. Vu sous cet angle, une méthode pourrait être indexée par ces suppositions. Ce travail de classification de suppositions est très important. Cela permet d'avoir un cadre conceptuel de façon à vérifier pour une méthode les divers types de suppositions exprimés. Ici, nous allons confronter notre définition d'une méthode de résolution de problèmes avec cette classification.

La typologie des suppositions est constituée de trois types: les suppositions épistémologiques, pragmatiques et téléologiques. Les suppositions épistémologiques définissent les connaissances exigées par une méthode. Ces suppositions sont raffinées en deux types. D'abord, les suppositions de disponibilité, qui définissent les connaissances utilisées par la méthode. Ensuite, les suppositions de propriétés, qui définissent les caractéristiques de ces connaissances. Par exemple, une supposition de disponibilité définit que les connaissances sont organisées selon un réseau causal et une supposition de propriété définit que ce réseau est organisé comme un graphe sans cycles. D'autre part, les suppositions pragmatiques définissent les exigences physiques ou l'environnement externe où la méthode va être appliquée, telles que le temps requis d'exécution, l'espace de mémoire disponible et la fiabilité des observations. Finalement, les suppositions téléologiques définissent le but à atteindre par la méthode. Un exemple de supposition téléologique est d'expliciter si une méthode pour le diagnostic trouve un seul problème ("single-fault model") ou plusieurs problèmes ("multiple-fault model") comme solution finale.

Si on analyse nos définitions de méthodes à la lumière de cette classification, on verra que les suppositions épistémologiques se retrouvent dans la définition de la méthode, que nous ne considérons pas les suppositions pragmatiques, et que les suppositions téléologiques sont partiellement définies dans la définition de la tâche.

Ainsi, les deux types de suppositions épistémologiques, de disponibilité et de propriété, se trouvent dans les définitions de l'ontologie de la méthode. Dans P&R, par

exemple, les paramètres doivent être organisés en termes d'un réseau de dépendances (supposition de disponibilité) sans cycles (supposition de propriété).

En ce qui concerne les suppositions pragmatiques, elles ne sont pas du tout considérées dans nos méthodes. Toutefois, il ne nous semble pas évident que ce type de supposition soit nécessaire pour exprimer les conditions d'applicabilité d'une méthode. Nous croyons que les suppositions pragmatiques ne contribuent pas vraiment à comprendre la méthode. À notre avis, essayer d'expliciter ces contraintes pragmatiques, telles que la disponibilité de tests de laboratoire, ou l'existence des pièces de remplacement pour un dispositif, n'est pas nécessaire, parce que ces distinctions peuvent contribuer à surcharger les descriptions de méthodes avec des détails inutiles.

D'un autre côté, les suppositions téléologiques sont spécifiées partiellement dans la définition de la tâche (les arguments `:input` et `:output`). Pour P&R, le but indique que la méthode doit associer à chaque paramètre une valeur sans qu'il y ait des contraintes violées. Cependant, dans la définition de la tâche, on ne trouve pas d'informations sur le type de solution trouvée. Par exemple, il n'est pas possible de savoir si la solution obtenue par P&R est la meilleure (le coût plus bas) ou si la solution est celle obtenue le plus rapidement. Nous croyons qu'il serait intéressant d'avoir ce type d'information associé aux méthodes réutilisables, mais nous n'avons pas considéré cet aspect, pour nous concentrer sur la définition des rôles de connaissances. En effet, il s'agit d'un problème complexe qui nécessite d'une étude approfondie.

Inférences de KADS

Comme nous l'avons mentionné à la section 5.2, nous utilisons la même approche que KADS pour définir les inférences: une classification des inférences basée sur une ontologie prédéfinie. Cependant, les inférences de KADS sont spécifiées en fonction des catégories épistémologiques de son ontologie de base, tandis que nos inférences sont spécifiées en fonction des structures ontologiques de l'ontologie de la méthode.

Il ne faut pas perdre de vue que les deux ontologies sont basées sur une représentation terminologique. La différence est que KADS classifie en fonction des primitives

épistémologiques des langages terminologiques tels que concept, attribut, etc. Tandis que nous classifions nos inférences en termes non des primitives, mais des instances de ces catégories épistémologiques. Ainsi, nous avons pris littéralement l'approche KADS, en l'appliquant à une ontologie existante, l'ontologie de la méthode.

Vu sous cet angle, les inférences KADS sont dans un méta-niveau par rapport à nos inférences. Ceci implique que la typologie des inférences selon KADS peut être vue comme des types abstraits de nos inférences. Par exemple, l'inférence *sélectionner* de KADS choisit une instance à partir d'un ensemble, en se basant sur une relation prédéfinie. L'inférence ***select parameter*** (section 5.3.2) de P&R sélectionne un paramètre (instance) parmi tous les paramètres (ensemble), en considérant les dépendances (relation) entre les paramètres. De même, l'inférence *affecter* de KADS peut être vue comme un type abstrait de l'inférence ***propose*** de P&R. *Affecter* dérive des valeurs pour les attributs des concepts et affecte ces valeurs aux attributs. L'inférence ***propose*** (section 5.3.2) de P&R génère des valeurs aux attributs **has-value** des paramètres. Ainsi, l'inférence ***select parameter*** est une instance de l'inférence du type *sélectionner* de KADS, et ***propose*** est une instance l'inférence du type *affecter* de KADS.

On remarquera que les inférences de KADS sont plus *réutilisables* que les inférences de la méthode, parce que plus générales. Cependant, les inférences de la méthode sont plus *utilisables*, parce plus spécifiques et adaptées à la méthode. De plus, les inférences décrites selon l'ontologie de la méthode favorisent la compréhension de la méthode et peuvent, dans un certain sens, contribuer à sa réutilisation. En comprenant ce que la méthode fait, il est plus facile de la modifier et de l'intégrer dans différentes applications. Ainsi, nous avons décrit les instances concrètes des inférences et en fonction des rôles des connaissances.

De plus, la classification de KADS n'est pas exhaustive: par exemple, il est difficile de classifier l'inférence ***check*** de P&R (section 5.3.2) en fonction de la typologie de KADS. L'inférence ***check*** est, dans un premier temps, classifiée comme du type *comparer* de KADS. *Comparer* définit que les valeurs des attributs entre deux concepts sont comparés. Pour ***check***, ce ne sont pas les valeurs des attributs entre concepts

qui sont comparés, mais la valeur d'un attribut d'un concept avec une autre valeur. De cette manière, la typologie de KADS ne définit pas toutes les opérations possibles sur les catégories épistémologiques de base; par conséquent, les instances concrètes des inférences ne peuvent pas être toutes classifiées selon cette typologie.

VT-Ontolingua

Ontolingua a été développé pour construire des ontologies qui minimisent les engagements ontologiques et qui sont indépendantes d'une tâche ou d'une méthode spécifique (section 2.2.1). Par conséquent, l'ontologie pour VT (VT-Ontolingua) représentée en Ontolingua [Gruber et al., 1994] ne modélise pas tous les rôles des connaissances utilisés par la méthode. L'analyse de Motta et Zdrahal [Motta et Zdrahal, 1995] montre que VT-Ontolingua manque et interprète mal les rôles des connaissances utilisés par P&R. Par exemple, les procédures de proposition des valeurs initiales des paramètres ne sont pas distinguées des contraintes. De plus, les réparations ne sont pas représentées dans l'ontologie, étant considérées comme des connaissances de contrôle. Par conséquent, VT-Ontolingua rend la méthode moins réutilisable, parce que cette ontologie n'explicite ni adéquatement ni complètement les types de connaissances utilisés par P&R.

Nous nous intéressons à la construction des ontologies qui dévoilent le plus possible la structure des connaissances utilisée par les méthodes, comme l'ontologie de P&R. Les ontologies de méthode sont très spécifiques, en maximisant les engagements ontologiques. Par conséquent, ces ontologies sont utiles pour comprendre une méthode et pour contribuer à sa réutilisation.

PROTÉGÉ-II

Le concept d'ontologie de la méthode de PROTÉGÉ-II, dont nous avons emprunté le terme, est similaire au concept utilisé dans cette thèse. Dans les deux cas, l'ontologie de la méthode spécifie les types de connaissances requises pour une méthode particulière.

L'avantage de PROTÉGÉ-II est la génération automatique d'outils d'acquisition de connaissances spécifiques par méthode et adaptables pour des différentes applications. Nous n'avons pas traité de cet aspect, mais nous croyons que notre ontologie de la méthode pourrait servir de spécification pour l'élaboration d'outils de ce type.

Notre approche diffère de celle de PROTÉGÉ-II, en ce qui concerne la définition des inférences. Dans PROTÉGÉ-II, les inférences sont appelées *mécanismes*, auxquels correspondent un code implanté, mais non une spécification déclarative. Dans notre cas, les inférences sont déclarées et à cette définition correspond le code implanté, mais il n'est pas nécessaire pour la compréhension de la méthode.

Dans un travail récent, le langage KARL [Fensel, 1995b] a été utilisé pour formaliser les rôles des connaissances et les inférences des méthodes dans PROTÉGÉ-II [Fensel et al., 1996]. KARL est un langage formel et opérationnel construit selon le modèle conceptuel de KADS. Ce travail utilise la même perspective que la nôtre, en spécifiant les rôles des connaissances avec la partie terminologique de KARL. La différence par rapport à notre modélisation concerne les inférences; celles-là sont exprimées en KARL par des clauses de Horn, tandis que nous utilisons les phrases en Loom, qui ont le pouvoir d'expression de la logique de premier ordre.

Objets du raisonnement

Dans sa thèse de doctorat, Causse-Gobinet [Causse-Gobinet, 1994] propose également la définition des rôles des connaissances de la méthode en termes d'une ontologie de la méthode. La différence par rapport à notre travail est que les rôles de l'ontologie de la méthode de Causse-Gobinet sont définis comme des *objets du raisonnement* et que les inférences sont associées à ces objets, selon le même principe que des méthodes sont associés aux structures de données dans l'approche orientée objet dans les langages de programmation. Dans notre définition des méthodes de résolution de problèmes, les inférences sont définies en termes des rôles des connaissances, mais elles ne sont pas vues comme des opérateurs liés intrinsèquement aux rôles. Nous maintenons la même structure du modèle de l'expertise de KADS, en gardant l'indépendance

entre les rôles et les inférences.

5.5 Résumé

Dans ce chapitre, nous avons montré notre approche pour décrire les méthodes de résolution de problèmes. D'abord, nous avons utilisé comme point de départ la structure d'inférence de la méthode qui donne une vision globale des inférences et des rôles des connaissances. Ensuite, nous avons construit une ontologie de la méthode qui spécifie les rôles des connaissances. À partir de cette ontologie, nous avons défini les inférences de la méthode. Finalement, nous avons spécifié la tâche par la définition de l'entrée-sortie et par la structure de contrôle. Nous avons utilisé la méthode P&R appliquée à la tâche VT pour illustrer notre démarche. Les avantages de notre spécification de la méthode sont les suivants:

- Permettre une meilleure compréhension de la méthode, sans avoir à fouiller le code implanté.
- Permettre de créer des nouvelles méthodes par la modification, la combinaison et l'extension de méthodes déjà existantes.
- Fournir une spécification de la méthode, pour en faciliter la vérification et la validation.
- Utiliser cette spécification pour analyser si une tâche peut être accomplie par la méthode en question.
- Générer et tester plus facilement des méthodes par l'utilisation d'un langage opérationnel, ce qui donne un feed-back rapide dans les premières phases de modélisation.

Finalement, nous avons comparé notre travail avec d'autres travaux connexes. Nous avons présenté des études à propos de P&R qui montrent que la nature compilée des connaissances de VT, à partir desquelles la méthode a été définie, complique

la réutilisation de cette méthode. Ensuite, nous avons confronté notre définition des méthodes avec une typologie existante de suppositions qui spécifie les conditions d'applicabilité de la méthode. Puis, nous avons analysé nos inférences à la lumière des inférences des KADS. Nous avons aussi comparé notre ontologie pour P&R avec l'ontologie pour VT construite en utilisant Ontolingua et nous avons aussi comparé notre approche avec celle de PROTÉGÉ-II. Finalement, nous avons confronté notre description des rôles des connaissances avec la description en termes d'objets du raisonnement, une perspective similaire à l'approche orientée objet.

Dans le prochain chapitre, nous définissons quelques types d'opérateurs de correspondance entre une ontologie de la méthode et les connaissances du domaine.

Chapitre 6

Opérateurs de correspondance: application à *Couvrir et Différencier*

Comme[1] nous l'avons montré au chapitre précédent, une méthode de résolution de problèmes est définie en fonction de la structure des connaissances utilisée par la méthode. C'est-à-dire que la méthode est définie non seulement en termes des inférences et des rôles des connaissances abstraits, mais en termes d'une ontologie qui précise comment les connaissances doivent être organisées pour comprendre et réutiliser la méthode.

Toutefois, nous avons vu aussi que pour réutiliser des bases des connaissances existantes, les connaissances ne doivent pas être destinées à une utilisation spécifique. Ceci nous amène à deux objectifs incompatibles:

- Bases de connaissances pour être réutilisées → les ontologies du domaine.
- Bases de connaissances pour être utilisées par les méthodes de résolution de problèmes → les ontologies de méthode.

1. Ce chapitre est une version modifiée et approfondie de l'article "From KADS Models to Operational Problem-Solving Methods: Perspectives in the Domain View" présenté au workshop KEML'96 [Coelho et al., 1996].

Nous avons parlé au chapitre 1 que cette incompatibilité est exprimée par le dilemme de la *réutilisabilité* versus l'*utilisabilité*:

- Un composant est réutilisable s'il est suffisamment général, de façon à être réutilisé dans différentes domaines et problèmes.
- Un composant est utilisable s'il est suffisamment spécifique, de façon à être utilisé dans un domaine et un problème particuliers.

La figure 6.1 illustre le dilemme de la *réutilisabilité* versus l'*utilisabilité*: des ontologies du domaine réutilisables ne correspondent pas aux ontologies de méthodes utilisables (parce qu'elles sont adaptées à la méthode).

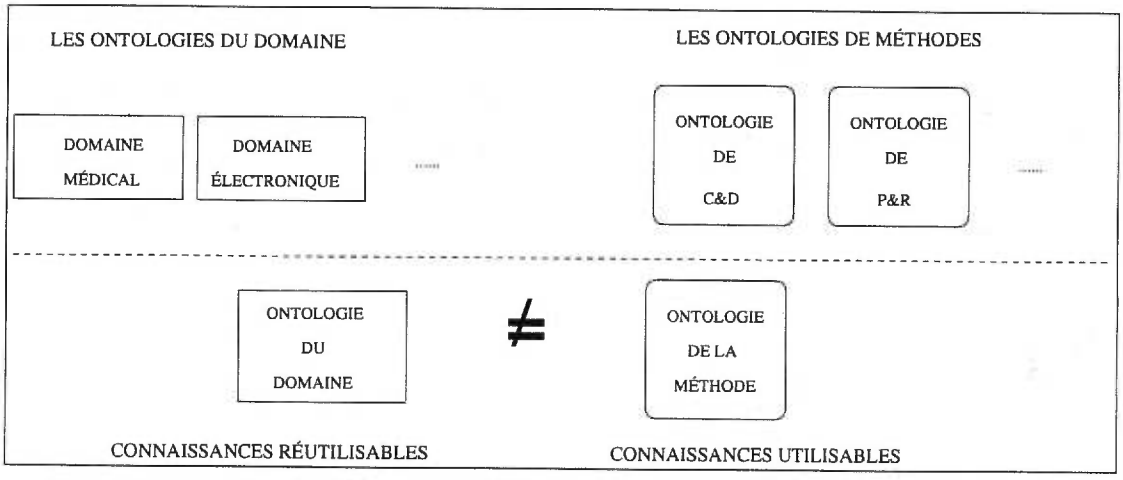

FIG. 6.1 - *Dilemme de la réutilisabilité versus l'utilisabilité. Cette situation arrive lorsqu'on veut réutiliser une ontologie du domaine appliquée à une méthode. Dans ce cas, il est peu probable que l'ontologie du domaine soit exactement pareille à l'ontologie de la méthode. Autrement dit, les connaissances réutilisables (générales) du domaine ne correspondent pas aux connaissances utilisables (spécifiques) de la méthode.*

Dans notre thèse, nous essayons de surmonter ce problème, en faisant un compromis pour obtenir les avantages de la réutilisabilité et l'utilisabilité des connaissances:

- Maximiser l'utilisabilité des méthodes de résolution de problèmes par la définition des ontologies de méthode.

- Maximiser la réutilisabilité des connaissances du domaine par l'utilisation des ontologies du domaine les plus indépendantes possible des applications.

Dans le chapitre précédent, nous avons esquissé notre architecture pour la réutilisation des méthodes de résolution de problèmes en utilisant les ontologies de méthode comme interface. Nous avons dit que pour utiliser des ontologies du domaine déjà existantes, il faut définir des opérations de correspondance ("mapping relations") [Linster, 1992] entre les ontologies de méthodes et les ontologies du domaine. Ces opérations de correspondance permettent d'avoir le meilleur des deux mondes: l'utilisabilité de la méthode et la réutilisabilité des ontologies du domaine. Dans ce chapitre nous allons détailler ces opérations de correspondance.

Dans le modèle conceptuel de KADS (chapitre 3), aux rôles des connaissances de la couche des inférences doivent être associés les connaissances du domaine de la couche du domaine. Comme nous avons vu dans la définition originale de KADS, les rôles des connaissances sont représentés par des étiquettes et n'explicitent pas la structure inhérente aux connaissances utilisées par la méthode. Ainsi, c'est dans le modèle du domaine qu'une structure des connaissances est définie selon son utilisation. Dans le chapitre précédent, nous avons ajouté à la couche des inférences l'ontologie de la méthode qui précise les rôles des connaissances de la méthode.

En se basant sur le modèle conceptuel de KADS, plusieurs groupes de recherche ont développé des langages pour décrire et comparer des méthodes de résolution de problèmes [Schreiber et al., 1993a, Fensel et Van Harmelen, 1994]. Ces langages peuvent modéliser les connaissances du domaine et les connaissances de contrôle des méthodes, en facilitant la compréhension et la modification de ces méthodes. Cependant, ces langages manquent de primitives pour représenter les opérations de correspondance entre les rôles des connaissances de la méthode et les connaissances du domaine. Les primitives existantes sont très simples, parce que ces langages considèrent les rôles comme de simples "placeholders" pour les connaissances du domaine, plutôt des entités ayant une structure particulière qui peut différer de celle des connaissances du domaine.

Dans ce chapitre, nous définissons des types d'opérateurs primitifs pour réaliser les opérations de correspondance entre les connaissances du domaine et les ontologies de la méthode. Ici, il faut considérer que pour réutiliser les connaissances et les méthodes de résolution de problèmes, le coût de compréhension et d'adaptation doit être inférieur au coût de construction du système à partir de rien. Autrement dit, les opérations de correspondance doivent être assez simples pour mériter une réutilisation d'une méthode existante dans une application spécifique. Notre idée de base est de formaliser et de rendre opérationnel les opérations de correspondance. Nous allons définir des types d'opérations de correspondance qui sont à la fois simples et puissants.

Pour illustrer nos idées, nous modélisons la méthode de résolution de problèmes *Couvrir et Différencier* (C&D) en utilisant le modèle conceptuel de KADS. Nous utilisons comme référence une description formelle de C&D [Van Harmelen et Balder, 1993] faite en $(ML)^2$. Après avoir défini C&D, nous montrons comment la méthode peut être réutilisée pour le domaine médical en appliquant des opérations de correspondance pour adapter les connaissances du domaine aux rôles des connaissances de la méthode.

6.1 Modélisation de C&D

Couvrir et Différencier [Eshelman, 1988] est une méthode pour résoudre un problème de classification (section 2.1.3). C&D peut être utilisée pour le diagnostic, en proposant des solutions candidates pour expliquer ("couvrir") les plaintes spécifiées par l'utilisateur et en cherchant des informations pour différencier les solutions candidates.

C&D contient deux phases: la phase "couvrir" et la phase "différencier". Pour chacune de ces phases, des rôles des connaissances spécifiques sont associés (figure 6.2). En suivant l'idée de base des méthodes à limitation de rôles, C&D précise les rôles joués par les connaissances dans la méthode de telle sorte que ces rôles peuvent être remplis pour différents domaines.

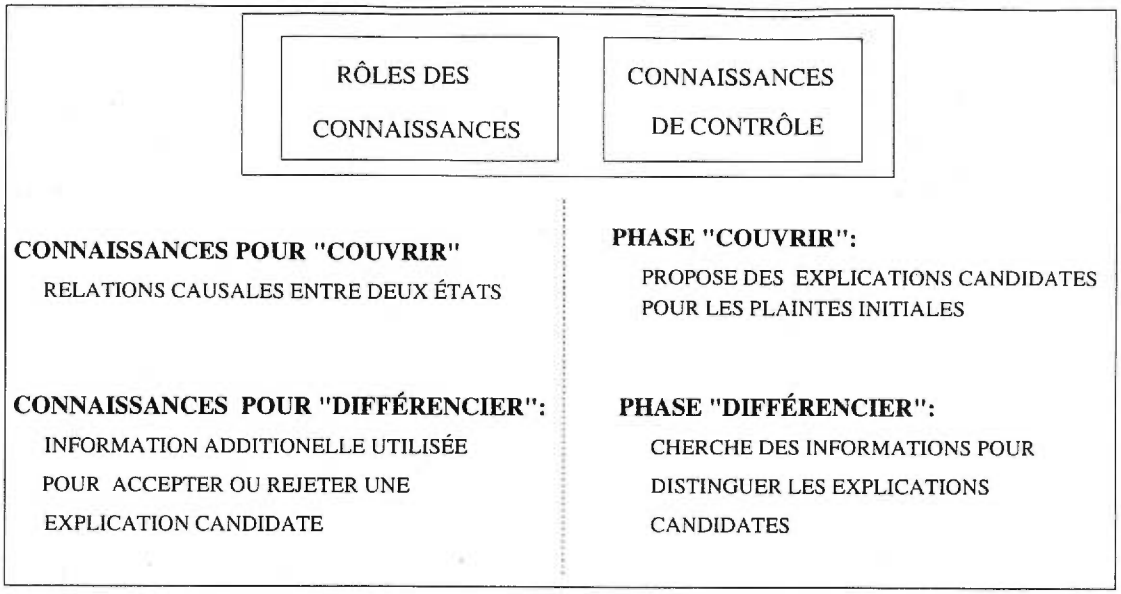

FIG. 6.2 - *Méthode C&D et ses rôles des connaissances. Les rôles des connaissances bien définis peuvent être remplis par différents domaines.*

6.1.1 Structure d'inférence de C&D

La figure[2] 6.3 montre la structure d'inférence de C&D. Dans cette modélisation, C&D est décomposée en huit inférences. Cette vision générale de la méthode nous montre qu'une plainte est d'abord choisie comme focus pour être expliquée (l'inférence ***establish focus***). Ensuite, dans la phase "couvrir", les explications candidates sont générées par le focus (l'inférence ***cover***). Finalement, dans la phase "différencier" les explications candidates vont être distinguées en explications acceptées (les inférences ***prefer*** et ***anticipate-1***) ou refusées (les inférences ***rule out*** et ***anticipate-2***). De plus, la phase différencier est accomplie par l'application de deux principes:

- Le principe d'exhaustivité selon lequel toutes les plaintes doivent être expliquées. L'idée ici est que si une plainte a seulement une explication, cette explication doit être acceptée (l'inférence ***exhaustivity***).

2. Les noms des inférences et des rôles des connaissances sont en anglais pour correspondre au code implanté.

- Le principe d'exclusivité selon lequel un nombre d'explications plus petit est préféré à un nombre d'explications plus grande. C'est le principe de parcimonie ou du rasoir d'Occam[3]. Dans ce cas, s'il existe deux explications pour un état, on va tenter de n'en conserver qu'une seule. Par le principe de parcimonie, on choisit celle qui explique déjà un autre état (l'inférence ***exclusivity***).

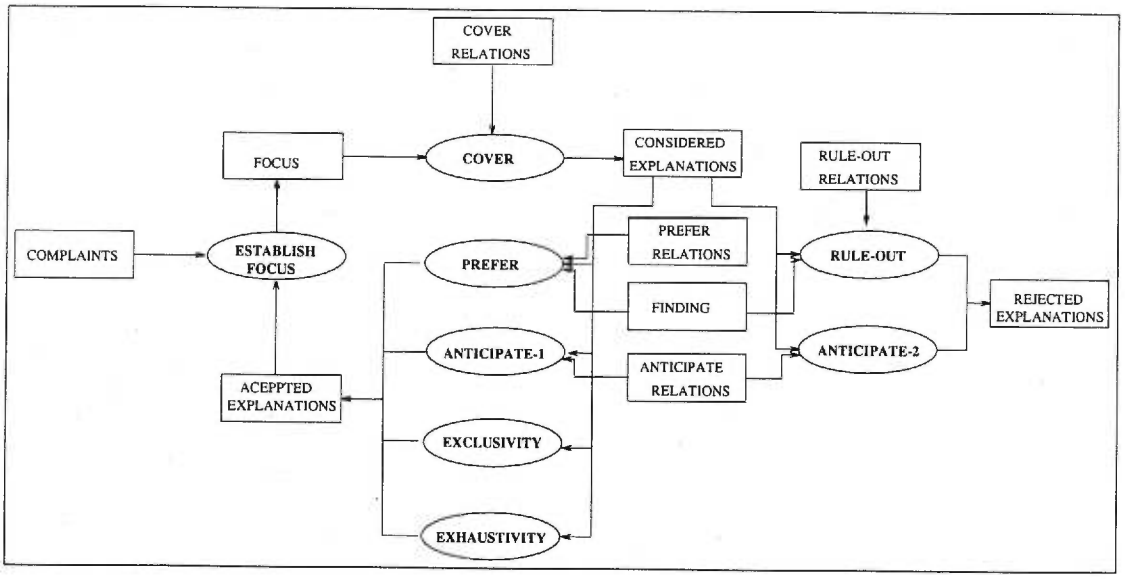

FIG. 6.3 - *Structure d'inférence pour C&D. Les ellipses représentent l'action de l'inférence, les rectangles représentent les rôles des connaissances et les flèches indiquent les dépendances des entrées-sorties des rôles des connaissances.*

Dans la prochaine section, nous décrivons l'ontologie de la méthode et les inférences pour C&D.

6.1.2 Ontologie de méthode et inférences de C&D

Pour construire l'ontologie de méthode pour C&D, nous allons faire de la même façon que pour P&R dans le chapitre précédent, c'est-à-dire, nous allons considérer

3. Guillaume d'Occam est un philosophe anglais de la première moitié du XIVe siècle. Son rasoir est une lame qui tranche au plus juste et supprime tout ce qui est inutile. Cette image signifie que, pour expliquer un phénomène, on doit s'efforcer d'en trouver la ou les causes dans d'autres phénomènes que nous connaissons déjà [Lercher, 1985].

chaque inférence à son tour, en expliquant les rôles des connaissances et ensuite en définissant l'inférence.

L'inférence *establish focus*

C&D peut être utilisée pour accomplir une tâche de diagnostic dont le but est de trouver une explication à un ensemble initial de plaintes. À partir de cet ensemble de plaintes, des explications possibles vont être générées (la phase "couvrir") et analysées de façon à choisir une seule explication pour tous les plaintes initiales (la phase "différencier").

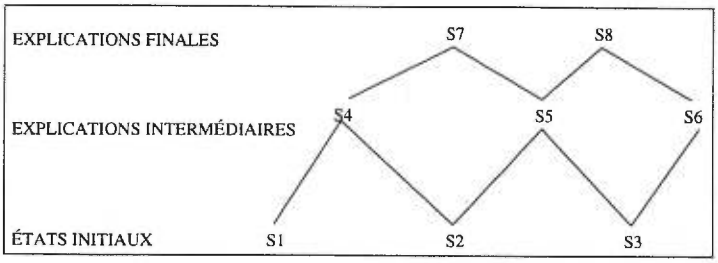

FIG. 6.4 - *Réseau d'explications de C&D. Les nœuds inférieurs représentent les états ou plaintes initiaux. Les nœuds supérieurs représentent les explications finales. Les autres nœuds fournissent des explications intermédiaires pour les plaintes.*

Dans le cas de C&D, les connaissances sont organisées en un réseau causal qui établit des relations entre les plaintes et les explications possibles pour ces plaintes (figure 6.4). Dans ce réseau, les nœuds inférieurs sont expliqués par les nœuds supérieurs.

En effet, les connaissances de C&D sont définies par de deux réseaux. Un réseau statique, qui contient toutes les connaissances du domaine, et un réseau dynamique, qui contient les connaissances spécifiques à un cas particulier[4].

Le réseau statique est généré une seule fois et il représente toutes les relations causales du domaine. La figure 6.5 montre une partie des définitions de l'ontologie de la méthode de C&D qui correspond aux définitions du réseau statique de C&D.

4. Cette distinction en KADS est faite par le modèle du domaine (connaissances statiques) et le modèle du cas (connaissances dynamiques).

```
(1) (defconcept State)
(2) (defrelation Cover-R :is-primitive
      (:and (:domain State) (:range State)))
(3) (defconcept Initial-State :is (:satisfies (?s1)
      (:and (State ?s1)
            (:not (:for-some (?s2) (Cover-R ?s1 ?s2)))))
         Initial-State)
(4) (defconcept Final-State :is (:satisfies (?s1)
      (:and (State ?s1)
            (:not (:for-some (?s2) (Cover-R ?s2 ?s1))))))
```

FIG. 6.5 - *Partie du réseau statique de C&D. (1) définit les nœuds du réseau. (2) définit les liens d'explications par la relation Cover-R. (3) définit les états initiaux. Un nœud est un état initial s'il n'explique aucun autre nœud. (4) définit les états finaux. Un état est final s'il n'est pas expliqué pour un autre état.*

```
(5) (defconcept Dynamic-State :is-primitive State)
(6) (defconcept Complaint :is-primitive  Dynamic-State)
(7) (defrelation Considered-Explanation
       :arity 2
       :domain State
       :range State)
(8) (defrelation Accepted-Explanation
       :arity 2
       :domain Dynamic-State
       :range Dynamic-State)
(9) (defrelation Rejected-Explanation
       :arity 2
       :domain State
       :range  State)
(10) (defconcept Focus :is-primitive Dynamic-State)
```

FIG. 6.6 - *Partie du réseau dynamique de C&D. (5) définit les nœuds du réseau dynamique. (6) définit les plaintes comme des nœuds du réseau dynamique. (7) définit les explications candidates comme un sous-ensemble des relations d'explication du réseau statique. (8) et (9) définissent les explications différenciées comme un sous-ensemble du réseau statique. (10) définit qu'un focus est un nœud qui fait partie du réseau dynamique. Pour les définitions des relations (7), (8) et (9), le premier argument explique le deuxième.*

D'un autre côté, le réseau dynamique peut être vu comme un sous-ensemble du réseau statique et il est construit en utilisant les données spécifiques d'une situation de diagnostic. Le réseau dynamique est construit en utilisant les données initiales (**complaints**) et le réseau statique pour générer des explications candidates (**considered explanations**) pour ces plaintes jusqu'à ce que les explications soient différenciées (**accepted explanations** et **rejected explanations**) et une seule explication soit trouvée pour chaque plainte. La figure 6.6 montre les définitions associées au réseau dynamique.

```
(11) (definference Establish-Focus
       :cond
        (:for-some ?f
          (:and (:or (Complaint ?f)
                     (:for-some  ?s1 (Accepted-Explanation ?f ?s1)))
                (:not (:or (:for-some ?s2 (Considered-Explanation ?s2 ?f)
                           (:for-some ?s2 (Accepted-Explanation ?s2 ?f)))))))
       :body (Establish-Focus ?f)
       :result (Focus ?f))
```

FIG. 6.7 - *Inférence* ***establish focus***.

La figure 6.7 présente une formalisation de l'inférence ***establish focus*** en considérant les rôles des connaissances définis par la définition du réseau statique et dynamique. L'inférence ***establish focus*** choisit un **focus** à partir duquel des explications candidates seront générées. L'inférence utilise les plaintes (**complaints**) ou les explications acceptées (**accepted explanations**) déjà calculées pour sélectionner un focus (**focus**).

L'inférence *cover*

Cette inférence génère des explications candidates. L'inférence utilise les relations de couverture (**cover relations**) pour générer des explications candidates (**considered explanations**) pour le focus (**focus**). La figure 6.8 montre la définition de l'inférence ***cover***.

```
(12) (definference Cover
       :cond (:for-some ?f (:and (Focus ?f)
                                 (:for-some ?s1 (Cover-R  ?s1 ?f))))
       :body (Consider-Explanation ?f)
       :result (:for-some ?s1 (Considered-Explanation ?s1 ?f)))
```

FIG. 6.8 - *Inférence* ***cover***.

Les inférences *prefer* et *rule out*

Ces inférences font partie de la phase "différencier" dans laquelle les explications candidates (**considered explanations**) sont distinguées en explications acceptées et explications refusées (**accepted explanations** et **rejected explanations**). Pour faire cela, d'autres informations sont nécessaires.

```
(13) (defconcept Qualifier)
(14) (defrelation Prefer-State-R
       :arity 2
       :domain Qualifier
       :range State)
(15) (defrelation Prefer-Connection-R
       :arity 3
       :domains (Qualifier State)
       :range State)
(16) (defrelation Rule-Out-State-R
       :arity 2
       :domain Qualifier
       :range State)
(17) (defrelation Rule-Out-Connection-R
       :arity 3
       :domains (Qualifier State)
       :range State)
```

FIG. 6.9 - *Définition des qualificateurs et leur association aux états et les relations causales font partie du réseau statique de C&D. (13) spécifie le concept* ***Qualifier****. (14) définit la relation* ***Prefer-State-R*** *qui indique qu'un qualificateur confirme un état du réseau. (15) définit la relation* ***Prefer-Connection-R*** *qui indique qu'un qualificateur confirme une relation causale dans le réseau. (16) et (17) définissent qu'un qualificateur disqualifie un état ou une relation causale dans le réseau.*

Ces informations sont définies comme des qualificateurs et sont associées au ré-

seau statique de C&D. Un qualificateur est une observation ou une constatation, concernant l'acceptation ou le rejet d'un état ou d'une relation indépendamment des plaintes. En d'autres termes, un qualificateur confirme ou disqualifie un état ou une relation d'explication. Considérons l'exemple montré en [Eshelman, 1988] expliquant la raison de la mort d'une personne. Si une des explications candidates était une piqûre d'abeille, cette hypothèse aurait pu être éliminée si l'événement avait eu lieu en hiver, puisqu'il il n'y a pas d'abeilles en hiver. Dans ce cas, le qualificateur est associé à un état et il disqualifie cet état, c'est-à-dire, si le qualificateur est présent, l'état peut être déconsidéré. Une autre possibilité pour refuser cette hypothèse serait que l'individu en question n'était pas allergique aux abeilles. Dans ce cas, le qualificateur est associé à une relation causale et il la disqualifie, c'est-à-dire que s'il est présent, la relation peut être éliminée.

```
(18) (defconcept Finding :is-primitive Qualifier)
(19) (definference Prefer
      :cond (:for-some (?s1 ?s2)
                  (Considered-Explanation ?s1 ?s2)
                  (:for-some ?q
                    (:and (Finding ?q)
                          (:or (Prefer-State-R ?q ?s1)
                               (Prefer-Connection-R ?q ?s1 ?s2)))))
      :body (Accept-Explanation ?s1 ?s2)
      :result (Accepted-Explanation ?s1 ?s2))
(20) (definference Rule-Out
      :cond (:for-some (?s1 ?s2)
              (:and (Considered-Explanation ?s1 ?s2))
                  (:for-some ?q
                    (:and (Finding ?q)
                          (:or (Rule-Out-State-R ?q ?s1)
                               (Rule-Out-Connection-R ?q ?s1 ?s2)))))
      :body (Reject-Explanation ?s1 ?s2)
      :result (Rejected-Explanation ?s1 ?s2))
```

FIG. 6.10 - *Inférences* ***prefer*** *et* ***rule out.***

La définition de ces qualificateurs et leur association aux états ou relations causales du réseau sont montrés dans la figure 6.9. Les qualificateurs qui confirment un état ou une relation sont définis par les relations "prefer". Les qualificateurs qui

disqualifient un état ou relation sont définis par les relations "rule-out".

Les qualificateurs spécifiques à la situation de diagnostic appartenant au réseau dynamique sont appelés **findings** et ils sont fournis au début par l'utilisateur ou demandés par le système. La figure 6.10 montre la définition des inférences ***prefer*** et ***rule out*** ainsi que du rôle **findings**. Ces inférences permettent d'accepter ou de refuser des explications candidates en se basant sur la présence des qualificateurs.

Les inférences *anticipate-1* et *anticipate-2*

En plus des qualificateurs, C&D utilise un autre type d'information pour pouvoir distinguer les explications candidates. Cette distinction peut être faite au moyen d'une différenciation entre les liens causaux du réseau statique, en les séparant en liens possibles et liens obligatoires. Dans cette perspective, une relation possible indique qu'un état $s1$ peut être la cause d'un état $s2$, mais que ce n'est pas toujours le cas. Une relation d'anticipation exprime que si un état $s1$ se produit, il provoque toujours l'état $s2$. Une relation d'explication dépend de la "force" de la connexion causale entre deux états. Dans une relation possible, la connexion causale entre deux états peut être ténue.

La figure 6.11 montre les définitions des relations d'anticipation et les inférences ***anticipate-1*** et ***anticipate-2***. Dans ce cas, les inférences considèrent que lorsqu'il existe une relation causale possible entre un état $s1$ et $s2$, et une relation causale d'anticipation entre $s1$ et un autre état $s3$. Si $s3$ fait partie du réseau dynamique, l'explication est acceptée (***anticipate-1***), sinon l'explication est refusée (***anticipate-2***). Donc, ***anticipate-1*** génère des explications acceptées basées sur la présence d'un nœud du réseau dynamique; ***anticipate-2*** génère des explications refusées basées sur l'absence d'un nœud du réseau dynamique.

Les inférences *exclusivity* et *exhaustivity*

L'inférence ***exclusivity*** est utilisée pour appliquer le principe de parcimonie. Supposons qu'il existe deux explications $s1$ et $s2$ pour un même état $s3$. Supposons

```
(21) (defrelation Anticipate-R :is-primitive
      (:and (:domain State) (:range State)))
(22) (definference Anticipate-1
      :cond (:for-some (?s1 ?s2 ?s3)
              (:and (Considered-Explanation ?s1 ?s2)
                    (:not (Accepted-Explanation ?s1 ?s2))
                    (Anticipate-R ?s1 ?s3)
                    (:not (:same-as ?s2 ?s3))
                    (Dynamic-State ?s3)))
      :body (Accept-Explanations ?s1 ?s2 ?s3)
      :result (:and (Accepted-Explanation ?s1 ?s2)
                    (Accepted-Explanation ?s1 ?s3))
(23) (definference Anticipate-2
      :cond (:for-some (?s1 ?s2)
              (:and (Considered-Explanation ?s1 ?s2))
                    (:for-some ?s3 (:and (Anticipate-R ?s1 ?s3)
                                         (:not (:same-as ?s2 ?s3))
                                         (:not (Dynamic-State ?s3)))))
      :body (Reject-Explanation ?s1 ?s2)
      :result (Rejected-Explanation ?s1 ?s2))
```

FIG. 6.11 - *Inférences* ***anticipate-1*** *et* ***anticipate-2***.

que $s1$ explique aussi un autre état $s4$ et que $s2$ n'explique aucun autre état. Dans ce cas, l'explication préférée va être celle qui explique plus d'un état, c'est-à-dire $s1$ (figure 6.12). Cette inférence permet d'avoir des explications plus simples en forçant le minimum d'explications.

L'inférence ***exhaustivity*** applique le principe selon lequel au moins une explication existe pour chaque plainte. De cette façon, s'il y a seulement une explication pour un état, cette explication doit être acceptée. La figure 6.13 montre les définitions des inférences ***exclusivity*** et ***exhaustivity***.

6.1.3 Définition de la tâche de C&D

Pour la méthode C&D, nous présentons dans la figure 6.14 une définition de la tâche construite selon le modèle conceptuel de KADS. Le but de la tâche est de diagnostiquer un problème. L'entrée est l'ensemble des plaintes initiales fournies par l'utilisateur. La sortie donne le diagnostic. Le flux de contrôle indique, par la boucle

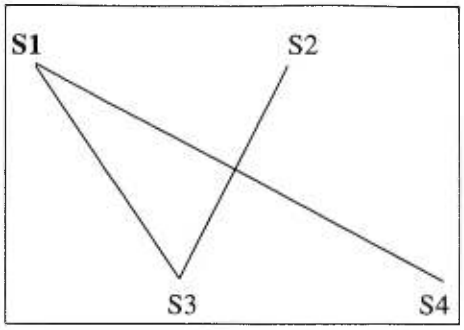

FIG. 6.12 - *Principe de l'exclusivité. L'explication s1 est préférée à l'explication s2, parce que s1 explique les plaintes s3 et s4, tandis que s2 explique seulement s3.*

```
(24) (definference Exclusivity
      :cond
        (:for-some (?s1 ?s3)
          (:and (Considered-Explanation ?s1 ?s3)
                (:not (Accepted-Explanation ?s1 ?s3))
                (:for-some ?s2
                  (:and (:not (:same-as ?s1 ?s2))
                        (Considered-Explanation ?s2 ?s3)
                        (:not (:for-some ?s5
                                (:and (:not (:same-as ?s3 ?s5))
                                      (Considered-Explanation ?s2 ?s5))))))
                (:for-some ?s4 (:and (:not (:same-as ?s1 ?s4))
                                     (Considered-Explanation ?s1 ?s4)))))
     :body (Accept-Explanation ?s1 ?s3)
     :result (Accepted-Explanation ?s1 ?s3))
(25) (definference Exhaustivity
      :cond (:for-some (?s1 ?s2)
              (:and (Considered-Explanation ?s1 ?s2)
                    (:not (Accepted-Explanation ?s1 ?s2))
                    (:not (:for-some ?s3
                            (:and (Considered-Explanation ?s3 ?s2)
                                  (:not (:same-as ?s3 ?s1))))))))
     :body (Accept-Explanation ?s1 ?s2)
     :result (Accepted-Explanation ?s1 ?s2))
```

FIG. 6.13 - *Inférences* ***exclusivity*** *et* ***exhaustivity***.

principale, que la tâche est accomplie quand tous les états sont expliqués. Dans la première boucle interne (la phase "couvrir"), les inférences ***establish focus*** et ***cover*** dérivent des explications candidates jusqu'à ce que tous les états aient des explications candidates ou acceptées. Dans la deuxième boucle interne (la phase "différencier"), les inférences ***prefer***, ***rule out***, ***anticipate-1***, ***anticipate-2***, ***exclusivity*** et ***exhaustivity*** distinguent les explications candidates jusqu'à ce que elles soient toutes différenciées.

```
task Cover-and-Differentiate
   goal
        diagnostiquer un problème
   input
        les plaintes fournies par l'utilisateur
   output
        le diagnostic obtenu
   task-structure
        REPEAT
            REPEAT
                establish-focus
                cover
            UNTIL (tous les états aient ou des explications candidates ou acceptées)
            REPEAT
                prefer
                rule-out
                anticipate-1
                anticipate-2
                exclusivity
                exhaustivity
            UNTIL (tous les explications candidates soient différenciées)
        UNTIL (tous les états soient expliqués ou finaux)
```

FIG. 6.14 - *Définition de la tâche C&D selon KADS.*

Dans la figure 6.15, nous montrons la représentation en K-Loom qui correspond au modèle conceptuel de C&D en KADS. Ainsi, l'argument : `input` établit que toutes les plaintes initiales doivent être un nœud inférieur du réseau statique, sinon il restera des états non expliqués. L'argument : `output` spécifie qu'après l'exécution de la tâche tous les états doivent avoir une seule explication. Cela va préserver le principe d'ex-

```
(deftask Cover-and-Differentiate
   :input
     (:for-all ?s (:implies (Complaint ?s)
                            (Initial-State ?s)))
   :output
     (:for-all ?s (:implies
                    (:and (Dynamic-State ?s) (:not (Final-State ?s)))
                    (:for-some ?s1
                      (:and (Accepted-Explanation ?s1 ?s)
                            (:not
                             (:for-some ?s2
                                (:and (Accepted-Explanation ?s2 ?s)
                                      (:not (:same-as ?s1 ?s2)))))))))
  :task-structure
     ((repeat-until (:for-all ?s1 (:implies
                       (:and (Dynamic-State ?s1)
                             (:not (:for-some ?s2 (Accepted-Explanation ?s2 ?s1))))
                       (Final-State ?s1)))
      (while (:for-some ?f (:and
                 (:or (Complaint ?f)
                      (:for-some ?s1 (Accepted-Explanation ?f ?s1)))
                 (:not (:or (:for-some ?s2 (Accepted-Explanation ?s2 ?f))
                            (:for-some ?s2 (Considered-Explanation ?s2 ?f))))))
        (Establish-Focus)
        (Cover))
      (while (:for-some (?s1 ?s2) (Considered-Explanation ?s1 ?s2))
        (Prefer)
        (Rule-Out)
        (Anticipate-1)
        (Anticipate-2)
        (Exclusivity)
        (Exhaustivity)))))
```

FIG. 6.15 - *Définition de la tâche C&D en K-Loom. Les inférences sont en italiques et les rôles des connaissances sont en gras.*

clusivité. Par conséquent, le graphe généré des explications acceptées est un arbre. Le flux de contrôle donné par l'argument :`task-structure` définit une boucle principale où les commandes sont répétées jusqu'à ce que les tous les états aient des explications acceptées ou soient des états finaux. Une boucle interne génère des explications candidates pour les états non encore expliqués jusqu'à ce que tous les états aient des explications candidates ou acceptées. Finalement, la deuxième boucle à l'intérieur distingue les explications candidates en acceptées ou rejetées jusqu'à ce qu'il y n'ait plus d'explications candidates.

De la même façon que pour la méthode P&R montrée à la section 5.3.3, nous pourrions imaginer des flux de contrôle alternatifs pour la méthode C&D, où les règles de différenciation sont appliquées dans un ordre différent. Par exemple, les inférences ***antecipate-1*** et ***antecipate-2*** pourraient être appliquées avant les inférences ***prefer*** et ***rule-out***. De même, pour d'autres applications, d'autres règles de différenciation peuvent être ajoutées.

Dans l'annexe B, nous montrons une trace de l'exécution de C&D en K-Loom en considérant le cas de diagnostic médical présenté dans la prochaine section.

6.2 Diagnostic médical

Dans cette section, nous illustrons la réutilisation de C&D pour accomplir une tâche de diagnostic médical. Nous montrons comment les données médicales déjà définies peuvent être associées aux rôles des connaissances de C&D. Nous faisons cette association en utilisant des opérations de correspondance.

Les données médicales utilisées sont basés sur les protocoles obtenus en examinant les aspects du raisonnement médical. Dans ce cas, un diagnostic médical est représenté par un réseau d'explications, dont les nœuds inférieurs représentent des symptômes d'un patient et les autres nœuds des explications possibles pour les symptômes. Dans cette perspective, le réseau causal établit des relations entre les symptômes et les explications possibles (les maladies) pour les symptômes. Dans la figure 6.16, nous illustrons un réseau d'explications pour un cas d'endocrinologie. Le réseau a été adapté

de [Patel et al., 1989] et il est basé sur les explications des médecins pour un cas clinique réel.

Nous pouvons constater que ce réseau est semblable au réseau statique utilisé par C&D: les arcs établissent des relations d'explication entre les nœuds. Les nœuds inférieurs sont expliqués par les nœuds supérieurs. Les nœuds les plus bas représentent des plaintes initiales, et les nœuds supérieurs représentent les explications finales. Les autres nœuds fournissent des explications intermédiaires pour les plaintes initiales.

De plus, il existe deux types de relations d'explication: des relations causales possibles, représentées par les lignes fines, et les relations causales nécessaires, représentées par les lignes épaisses. Ainsi, nous retrouvons les deux types de relation de causalité de C&D, les relations possibles ("Cover-R") et les relations nécessaires ("Anticipate-R"). Dans la figure 6.16, nous avons la relation nécessaire entre **paralysie hypokalémique** (hypokalemic paralysis) et **commencement subit de faiblesse du muscle** (sudden onset of MW), qui signifie que si un patient a **paralysie hypokalémique**, il aura certainement un **commencement subit de faiblesse du muscle**. Les autres relations sont des relations possibles. Par exemple, si le patient a un **état hypermétabolique**, c'est possible qu'il y ait **perte de poids**, mais ce n'est pas toujours le cas.

Nous trouvons aussi des qualificateurs dans le réseau médical. Les lignes pointillées associent un qualificateur à un nœud ou à une relation. Dans la figure 6.16, **ethnie orientale** (oriental ethnic background) est un qualificateur d'état qui confirme l'explication **paralysie périodique hypokalémique avec thyrotoxicosis**; **après l'ingestion de carbohydrates** (following ingestion of carbohydrates) est un qualificateur de connexion qui confirme la relation causale entre **paralysie hypokalémique** et **faiblesse du muscle**. Donc, un qualificateur est une observation ou une constatation qui n'est pas un symptôme et ils ont le même sens que les qualificateurs définis dans la méthode C&D.

Le cas clinique de la maladie musculaire offre un bon exemple pour démontrer C&D. Supposons, cependant, que le réseau médical est déjà organisé dans une base de connaissances selon les définitions de la figure 6.17 qui illustre la structure des

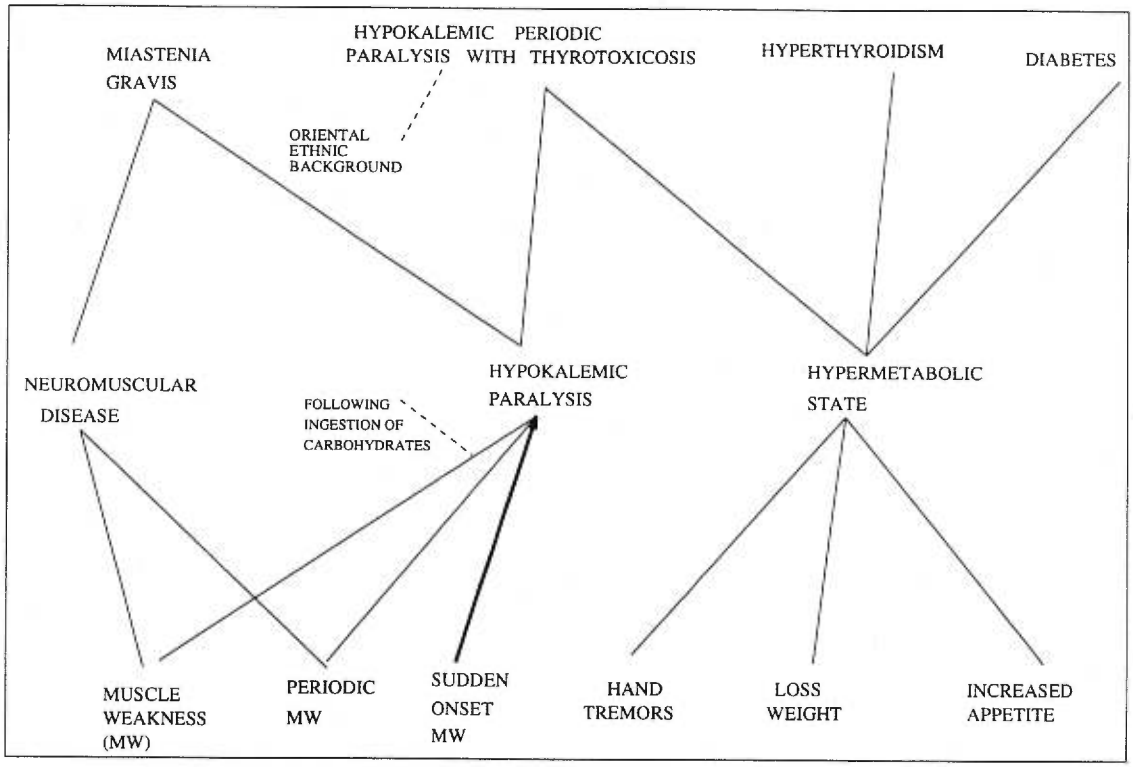

FIG. 6.16 - *Réseau d'explication pour un cas endocrinologique (adapté de* [Patel et al., 1989]*). Le réseau modélise un cas d'un homme oriental qui présente une* **faiblesse périodique du muscle des jambes** *("periodic leg muscle weaknesses"). Le diagnostic correct est* **paralysie périodique hypokalémique associé à thyro-toxicosis** *("hypokalemic periodic paralysis associated with thyrotoxicosis"). La* **paralysie périodique hypokalémique** *("hypokalemic periodic paralysis") est une maladie musculaire rare qui apparaît exclusivement* **chez les orientaux** *("oriental ethnic background"). Les épisodes de paralysie sont fréquemment déclenchés* **après un repas riche en carbohydrates** *("following ingestions of carbohydrates") ou par des activités physiques vigoureuses. La* **thyrotoxicosis** *est forme aiguë de* **hyperthyroïdisme** *("hyperthyroidism") qui résulte en un état* **hypermétabolique** *(hypermetabolic state) qui provoque des symptômes tels quel* **tremblement de mains** *("hand tremors"),* **perte de poids** *("weight loss") et* **augmentation de l'appétit** *("increased appetite"). La complexité du cas est telle que quelques médecins ont conclu des groupes de diagnostiques différents comme* **myasthenia grave** *(myasthenia gravis),* **hypethyroïdisme** *(hypethyroidism) et* **diabète** *(diabetes).*

connaissances correspondant au réseau causal de la figure 6.16. Il faut noter ici que, même si le réseau médical est apparemment équivalent au réseau de C&D, la définition formelle de ce réseau ne correspond pas aux définitions de C&D, comme nous pouvons vérifier en comparant les définitions de la figure 6.5 avec la figure 6.17.

```
(26) (defset Modal-Operator :is (:one-of 'Possibly 'Necessarily))
(27) (defset Qualifier-Operator :is (:one-of 'Positively 'Negatively))
(28) (defconcept Patient-State)
(29) (defconcept Qualifier)
(30) (defrelation Cause
       :arity 3
       :domains (Modal-Operator Patient-State)
       :range Patient-State)
(31) (defrelation State-Qualifying
       :arity 3
       :domains (Qualifier-Operator Qualifier)
       :range Patient-State)
(32) (defrelation Connection-Qualifying
       :arity 4
       :domains (Qualifier-Operator Qualifier Patient-State)
       :range Patient-State)
```

FIG. 6.17 - *Structure des connaissances de notre exemple du domaine médical. (26) définit les opérateurs modaux qui déterminent le type de relation causale entre deux états:* ***Possibly*** *ou* ***Necessarily****. (27) définit les opérateurs de qualification qui déterminent le type de qualificateur pour l'état ou relation:* ***Positively*** *ou* ***Negatively****. (28) et (29) spécifient les concepts* ***Patient-State*** *et* ***Qualifier****. (30) définit la relation* ***Cause*** *qui établit une relation entre deux états. Le premier état explique le deuxième état, et le* ***Modal-Operator*** *détermine la force de la relation. Les relations* ***State-Qualifying*** *(31) et* ***Connection-Qualifying*** *(32) établissent les relations entre les qualificateurs et les nœuds ou les arcs dans le réseau.*

Examinons maintenant comment les connaissances du domaine médical, en étant définies d'une façon particulière, peuvent être adaptées pour être utilisées par la méthode C&D.

6.3 Opérations de correspondance

Dans cette section, nous montrons comment les opérations de correspondance permettent d'augmenter la réutilisabilité et l'utilisabilité des connaissances. Ici, la réutilisabilité est obtenue en gardant les connaissances du domaine générales de façon à être réutilisées par différentes méthodes. L'utilisabilité est obtenue en établissant une correspondance entre les connaissances du domaine médical et les rôles des connaissances de la méthode.

Nous allons illustrer nos idées par la définition des opérations de correspondance qui adaptent les connaissances du domaine, telles qu'elles ont été définies dans la section précédente, de façon à les utiliser dans le cadre de la méthode C&D. L'idée de base est de réarranger les connaissances du domaine pour être réutilisées par les inférences.

Avec notre exemple de réseau de connaissances médicales, il est facile d'établir une correspondance avec le réseau de C&D, parce que la différence entre les deux se situe au niveau de la définition formelle, c'est-à-dire au niveau syntaxique. Ce que nous voulons montrer avec cet exemple est que les opérations de correspondance entre les ontologies de méthodes et les ontologies du domaine doivent être assez simples. En effet, nous remarquons que les deux ontologies doivent être potentiellement équivalentes et, dans ce cas, l'ontologie de la méthode correspond à une *vision* de l'ontologie du domaine. Ceci est motivé par le fait que si les ontologies sont complètement différentes, l'effort demandé pour faire la correspondance entre les deux est trop grand et, en ce sens, il est mieux de recommencer à zéro. En d'autres termes, si après avoir analysé les ontologies de la méthode et du domaine, les deux ontologies sont très différentes, il peut être préférable de construire une nouvelle méthode ou une nouvelle ontologie du domaine qui correspond à l'ontologie de la méthode.

Analysons maintenant les ontologies du domaine et de la méthode. Commençons par les nœuds du réseau causal qui, dans l'ontologie médicale sont appelés, dans le réseau statique, `Patient State` (6.17). Dans l'ontologie de C&D, ces nœuds sont appelés `State`. Dans ce cas, il faut renommer le concept de l'ontologie médicale par le

concept de l'ontologie de la méthode. Considérons, d'autre part, les relations causales qui, dans le domaine médical, sont définies par la relation `Cause`. Pour cette définition, nous distinguons entre des relations causales "fortes" ou "faibles" par l'utilisation d'un opérateur modal. En ce qui concerne les relations causales de C&D, il y a les deux types de relations, mais au lieu d'utiliser un opérateur modal, la distinction est faite par le nom de la relation: `Cover-R` et `Anticipate-R`. Donc, nous pouvons identifier facilement la correspondance entre l'ontologie médicale et l'ontologie de C&D: les relations `Cause` du domaine médical doivent être associées aux relations `Cover-R` et `Anticipate-R` de C&D. Ces associations peuvent être exprimées en Loom, comme le montre la figure 6.18.

```
(33) (implies Patient-State State)
(34) (implies (:satisfies (?s1 ?s2) (:for-some (?o) (Cause ?o ?s1 ?s2)))
              Cover-R)
(35) (implies (:satisfies (?s1 ?s2)
              (:for-some (?o) (:and (Cause ?o ?s1 ?s2)
                                    (:same-as ?o 'Necessarily))))
         Anticipate-R)
```

FIG. 6.18 - *Correspondances entre les relations causales de C&D et le domaine médical de la figure 6.17. (33) est une opération du type* ***alias****. (34) est une opération du type* ***projection****. (35) est une opération du type* ***projection*** *et* ***restriction****.*

Cela explicite les associations entre les relations de deux ontologies. Les instances de notre exemple médical sont associées aux relations de l'ontologie de la méthode. Nous pouvons caractériser ces associations comme des opérations d'**alias**, de **projection** et de **restriction**. L'opération d'**alias** est accomplie par l'assertion (33): le concept `Patient-State` de l'ontologie du domaine est rénommé en `State`, le concept de l'ontologie de la méthode. L'opération de **projection** est accomplie lorsque seulement quelques attributs d'une relation sont associés à une autre relation; par exemple en (34), seulement les attributs $?s1$ et $?s2$ de la relation `Cause` de l'ontologie du domaine sont associés à la relation `Cover-R` de l'ontologie de la méthode. L'opération de **restriction** est accomplie quand seulement des valeurs spécifiques d'une relation sont associés à une autre relation; par exemple en (35) , seulement les instances `Cause`

'Necessarily de l'ontologie du domaine sont associés à la relation Anticipate-R de l'ontologie de la méthode; l'assertion (35) correspond, en effet, à deux opérations en même temps, **restriction** et **projection**.

```
(36) (implies (:satisfies (?o ?s) (State-Qualifying 'Positively ?o ?s))
              Prefer-State-R)
(37) (implies (:satisfies (?o ?s) (State-Qualifying 'Negatively ?o ?s))
              Rule-Out-State-R)
(38) (implies (:satisfies (?o ?s1 ?s2)
                    (Connection-Qualifying 'Positively ?o ?s1 ?s2))
              Prefer-Connection-R)
(39) (implies (:satisfies (?0 ?s1 ?s2)
                    (Connection-Qualifying 'Negatively ?o ?s1 ?s2))
              Rule-Out-Connection-R)
```

FIG. 6.19 - *Correspondances entre les relations de qualification de C&D et le domaine médical. (36) à (39) sont des opérations de* ***restriction*** *et de* ***projection***.

Continuons à faire les correspondances entre les deux ontologies. Les relations qui associent un qualificateur à un état ou relation dans le domaine médical utilisent encore un opérateur modal qui distingue les qualifications favorables ou défavorables. Dans C&D, cette distinction est faite par le nom de la relation, et non par l'opérateur modal. Donc, nous pouvons exprimer la correspondance entre les deux ontologies par les définitions de la figure 6.19. En (36)-(39), des opérations de **restriction** et **projection** permettent d'associer les relations de qualifications de l'ontologie du domaine aux relations de qualification de l'ontologie de la méthode.

En faisant cela, nous avons explicité les associations entre les deux ontologies. Ces opérations sont très puissantes et, en même temps, très simples. Ici, nous avons tout le pouvoir d'expression de la logique pour exprimer les correspondances entre les deux ontologies. Les avantages de ces opérations sont de permettre d'adapter les connaissances d'un domaine aux connaissances de la méthode tout en conservant leur indépendance. La table 6.1 synthétise les opérations accomplies et leur type, dans ce cas, les opérations du type **alias**, **restriction** et **projection** ont été définies.

Concepts-Relations de l'ontologie du domaine	Concepts-Relations de l'ontologie de la Méthode	Type d'opération de Correspondance
Patient-State	State	Alias
Cause	Cover-R	Projection
	Anticipate	Restriction et Projection
State-Qualifying	Prefer-State-R	
	Rule-Out-State-R	
Connection-Qualifying	Prefer-Connection-R	
	Rule-Out-Connection-R	

TAB. 6.1 - *Opérations de correspondance entre l'ontologie médicale et l'ontologie de C&D. Les opérations de correspondance réalisés sont du type* ***alias***, ***projection*** *et* ***restriction***.

6.4 Types d'opérations de correspondance

Dans la section précédente, nous avons explicité les opérations de correspondance entre les ontologies de notre exemple du domaine médical et de C&D. Nous avons montré comment la définition de ces opérations peuvent contribuer à l'utilisabilité et réutilisabilité des connaissances du domaine et des méthodes de résolution de problèmes. Dans cette section, nous allons analyser des perspectives sur la définition d'un ensemble de types d'opérateurs qui peuvent être considérés primitifs pour établir les correspondances entre ontologies.

Dans notre exemple, les opérations de correspondance ont été définies par des assertions en Loom. L'opérateur `implies` de Loom permet d'exprimer ces opérations, en associant un concept ou une relation à un autre concept ou relation. Cela a permis de définir des instances d'une relation en fonction d'une autre relation. Nous avons dit que ces opérations étaient du type **alias**, **projection** et **restriction**. En considérant ces aspects, nous pouvons imaginer d'autres types d'opérateurs qui pourront réarranger l'ontologie du domaine en l'adaptant à l'ontologie de la méthode. De cette façon, nous suggérons quelques types d'opérateurs de correspondance primitifs pour la définition des correspondances:

- **Alias** → renommer un concept ou relation.

- **Projection** → définir une relation avec quelques attributs d'une autre relation.

- **Restriction** → définir une relation avec quelques valeurs d'une autre relation.
- **Inversion** → définir une relation comme l'inverse d'une autre relation.
- **Composition** → définir une relation comme une composition d'autres relations.
- **Union** → obtenir un concept à partir de l'union des instances de deux autres concepts.
- **Intersection** → obtenir un concept à partir de l'intersection des instances de deux autres concepts.
- **Complément** → obtenir un concept à partir du complément des instances de deux autres concepts. Le concept résultant contient les instances qui appartiennent au premier concept, mais non au deuxième.

Ces opérateurs sont semblables à ceux que l'on retrouve en mathématiques ou en bases de données [Date, 1986]. Ce sont des opérations simples et elles sont faciles à définir en Loom. Nous montrons quelques exemples de ces opérations dans la figure 6.20[5].

Les opérations de correspondance sont simples, mais en même temps puissantes de façon à adapter les connaissances d'une ontologie à autre. Cet ensemble de types d'opérations ne se veut ni minimal, ni exhaustif. À ce propos, nous croyons que définir et fournir ces types d'opérateurs de correspondance comme des opérateurs primitifs peut aider à la tâche d'adaptation d'une ontologie du domaine à une ontologie de la méthode.

6.5 Discussion

Dans ce chapitre, nous avons présenté des opérateurs pour faire la correspondance entre différents domaines et différentes méthodes, de manière à pouvoir réutiliser les

5. En Loom, quelques uns de ces opérateurs sont déjà implicites dans le langage, comme par exemple, les opérations `:inverse` et `:compose`.

```
(40) (implies (:satisfies  (?y ?x) (Cause ?x ?y))
              Caused-by)
(41) (implies (:satisfies (?x ?z) (:for-some ?y
                                      (:and (A ?x ?y) (B ?y ?z))))
              C)
(42) (implies (:satisfies (?x) (:or (A ?x) (B ?x)))
              C)
(43) (implies (:satisfies (?x) (:and (A ?x) (B ?x)))
              C)
(44) (implies (:satisfies (?x) (:and (A ?x) (:not (B ?x))))
              C)
```

FIG. 6.20 - *Exemples d'opérations de correspondance. (40) définit une opération du type* ***inversion****; la relation* ***Caused-by*** *est définie comme l'inverse de la relation* ***Cause****. (41) définit la relation* ***C*** *comme la composition des relations* ***A*** *et* ***B****. (42) définit le concept* ***C*** *comme l'union des concepts* ***A*** *et* ***B****. (43) définit le concept* ***C*** *comme l'intersection des concepts* ***A*** *et* ***B****. (44) définit le concept* ***C*** *comme le complément entre les concepts* ***A*** *et* ***B****. En général, les opérations utilisent le quantificateur universel suivi d'une implication logique, où dans l'antécédent on trouve les termes de l'ontologie du domaine, et dans le conséquent on trouve les termes de l'ontologie de la méthode.*

deux. Avec les opérateurs de correspondance, nous envisageons d'obtenir un compromis entre l'approche qui revendique une interaction forte entre les connaissances et la méthode, et l'approche qui défend l'idée de construire des ontologiques génériques et "universelles".

Nous nous sommes concentrée surtout sur l'étude et la définition des méthodes de résolution de problèmes, et non sur des ontologies du domaine réutilisables. La possibilité de construire et d'utiliser ces ontologies universelles reste encore un sujet de recherche. D'abord, il va falloir classifier ces ontologies par rapport aux domaines, ce qui est encore un autre problème ouvert: "qu'est-ce qu'un domaine?". Ensuite, la quantité de connaissances est, en général, énorme pour chaque domaine: "comment séparer les connaissances de façon à les stocker et les retrouver par la suite?". Ces questions ne sont pas simples.

Par exemple, il est difficile de construire une ontologie du domaine médical où plusieurs domaines se superposent, comme la biologie, la chimie, etc. Des travaux

récents [Van Heijst, 1995, Van Heijst et al., 1997] proposent la construction d'une *bibliothèque centrale* ("core library"), où les principes et les connaissances de base d'un domaine seraient définis dans des ontologies. Les connaissances spécifiques ou qui constituent des sous-domaines seraient organisées dans d'autres ontologies. La bibliothèque d'ontologies centrales et les ontologies de sous-domaines seraient organisées dans une hiérarchie débutant avec les ontologies de la bibliothèque centrale dont héritent les autres ontologies de sous-domaines. [Valente et Breuker, 1996] propose des principes pour guider la construction de ces ontologies centrales; ces principes sont illustrés par la construction d'une ontologie dans le domaine de droit.

Ici, nous avons insisté sur la définition des méthodes de résolution de problèmes réutilisables. Ce problème est, dans un certain sens, plus "facile" que celui de définir les ontologies universelles. D'abord, parce qu'il a été beaucoup plus exploré. Ensuite, parce qu'il est un problème plus "traitable", si on considère que le nombre de types de problèmes typiques est limité (8 types, selon la classification de [Breuker, 1994]) et, que le nombre de méthodes pour résoudre ces problèmes typiques est restreint. De plus, ces méthodes ont une structure abstraite qui les rend compréhensibles et réutilisables pour différentes applications.

Nous avons défini les opérateurs de correspondance de façon à favoriser la réutilisation de ces méthodes. Cependant, nous exigeons qu'il y ait une correspondance sémantique entre les connaissances du domaine et les rôles des connaissances de la méthode, sinon l'établissement de cette correspondance s'avère compliqué. Il nous faut aussi tester et valider ces opérateurs pour d'autres méthodes et pour d'autres domaines.

6.5.1 Comparaison avec des travaux connexes

L'idée d'utiliser des opérations de correspondance entre les connaissances du domaine et les rôles des connaissances d'une méthode a déjà été exploitée. Dans cette section, nous comparons notre travail avec d'autres travaux.

OMOS et d'autres langages pour KADS

Le langage OMOS [Linster, 1992] est basé sur le modèle conceptuel de KADS dont l'idée de base est de permettre la construction du modèle de l'expertise à partir d'un langage opérationnel.

OMOS permet la description d'un système à partir du point de vue de la méthode, *modèle de la méthode*, et à partir du point de vue de la structure du domaine, *modèle du domaine*. Les opérations de correspondance sont établies entre le modèle du domaine et le modèle de la méthode, en générant le *modèle de la tâche*. Cette organisation permet l'indépendance des connaissances du domaine par rapport aux connaissances de la méthode, considérées comme des connaissances procédurales. Il n'y a pas de concept d'ontologie de la méthode définissant la structure des connaissances de la méthode. En effet, le modèle du domaine est créé spécifiquement pour l'application en question. Le modèle de la méthode représente les inférences. Donc, il n'y a pas vraiment de différence syntaxique entre les rôles de la couche des inférences et les connaissances de la couche du domaine. Il s'agit seulement d'associer des rôles aux connaissances définies dans le modèle du domaine en désignant directement les connaissances du domaine en termes des rôles des connaissances.

La limitation d'OMOS est de ne pas considérer le fait que les modèles du domaine auraient pu être construits indépendamment et, dans ce cas, les opérations de correspondances devraient être plus élaborées et non seulement des opérations du type *renommer*, comme c'est le cas.

Les autres langages KADS [Fensel et Van Harmelen, 1994] existants ne présentent pas non plus d'opérateurs de correspondance tels que nous les avons définis. La plupart offrent des primitives pour associer les rôles des connaissances de la couche des inférences aux connaissances du domaine. Cependant, ces primitives sont très simples, parce que ces langages ne considèrent pas le fait que les rôles des connaissances ont une structure particulière qui peut être différente de la structure de connaissances du domaine.

PROTÉGÉ-II

PROTÉGÉ-II (section 2.3) offre aussi les opérations suivantes pour établir les correspondances entre les connaissances d'une application et les rôles des connaissances de la méthode:

- *Renommer* ("renaming"). Les rôles de la méthode et les concepts du domaine sont syntaxiquement équivalents, mais il faut les renommer.

- *Filtrage* ("filtering"). Les rôles de la méthode sont remplis par le filtrage des informations à partir des instances du domaine.

- *Correspondance entre concepts* ("class mapping"). Les rôles de la méthode sont remplis par les définitions de concepts du domaine, au lieu des instances (comme c'est le cas avec la correspondance par filtrage présentée auparavant).

Les opérations de PROTÉGÉ-II sont similaires aux nôtres. Par exemple, l'opération de filtrage de PROTÉGÉ-II correspond à l'opération de restriction dans notre approche. La différence est que les opérations de PROTÉGÉ-II sont limitées à trois types. Ainsi, PROTÉGÉ-II ne considère pas et d'autres possibilités de réarrangement des connaissances. En effet, nous avons proposé ici une extension de ces opérations, tout en gardant leur simplicité.

Une autre différence par rapport à notre approche est que PROTÉGÉ-II considère la possibilité d'établir des correspondances entre l'ontologie du domaine et l'ontologie de la méthode quand il y a des différences sémantiques entre ces deux ontologies [Eriksson et al., 1995]. Cependant, nous sommes plus sceptiques à propos de cette possibilité. Dans notre cas, nous considérons seulement les opérations de correspondance quand il n'y a qu'une différence syntaxique entre les ontologies. Cela parce qu'à notre avis, le coût pourrait ne pas valoir la peine pour adapter des connaissances du domaine dont la sémantique est différente de la sémantique de rôles des connaissances de la méthode. Il pourrait s'avérer préférable de trouver une autre méthode ou concevoir une méthode spécifique pour le problème en question.

SBF

Spark-Burn-Firefighter (SBF) [Klinker et al., 1991] est un environnement pour le développement des SBCs. Dans cet environnement un groupe de composants de base opérationnels, appelés *mécanismes*, est offert pour permettre la construction de nouveaux systèmes. L'approche de SBF est d'aider à l'utilisation des mécanismes par des non programmeurs.

Dans SBF, le *Active Glossary* [Klinker et al., 1993] est responsable pour l'association entre les mécanismes et la description de la tâche ("workplace activities"). Le *Active Glossary* intègre les termes organisés dans le glossaire, de façon à être un médiateur entre la perspective de la tâche et la perspective des mécanismes. Autrement dit, le glossaire associe les termes de la tâche avec les termes définis dans les mécanismes.

La différence par rapport à notre approche est que nous utilisons les opérateurs de correspondance pour exploiter les différences syntaxiques entre les définitions des ontologies de la méthode et du domaine, tandis que le *Active Glossary* utilise des heuristiques pour aider à faire l'association entre les connaissances spécifiques à une tâche et les termes utilisés par les mécanismes. SBF ne classifie pas les types d'opérations existants et n'indique pas comment le faire. Par exemple, si dans le voculaire utilisé dans le cadre de l'application il y a l'opération "Créer Rapport Final Hebdomadaire", le *Active Glossary* va associer ce terme avec le mécanisme déjà existent "Créer Rapport Résumé". Pour faire cette association, le *Active Glossary* utilise une heuristique qui trouve le terme dans le glossaire le plus similaire au terme de l'application.

Automatisation des opérations de correspondance

Le travail de Beys et ses collègues [Beys et al., 1996] propose d'automatiser partiellement le processus d'adaptation des connaissances du domaine aux rôles des connaissances de la méthode. Ici, les méthodes sont appelées *méthodes neutres par rapport à la tâche*, parce qu'elle sont définies sans aucune connotation associée à une tâche particulière (section 7.2). Cela est fait par l'association entre les caractéris-

tiques syntaxiques des définitions des connaissances du domaine et les suppositions de disponibilité et de propriété de la méthode (section 5.4.1). Cette vérification est automatique, mais elle peut aussi se faire avec l'aide de l'expert.

D'abord, un algorithme essaie de faire la correspondance entre les relations de l'ontologie du domaine et les suppositions de disponibilité de la méthode. Ainsi, plusieurs associations possibles peuvent être trouvées. Ensuite, les suppositions de propriétés sont utilisées pour vérifier si la correspondance est vraiment valide. Si l'ontologie du domaine n'explicite pas toutes les propriétés de ses relations, l'expert ou cogniticien doit intervenir pour terminer ce processus d'association entre la méthode et les connaissances du domaine.

Dans notre approche, nous ne proposons pas l'automatisation des opérations de correspondance entre une méthode et un domaine. À notre avis, le travail d'adaptation doit d'abord être fait par le cogniticien après une analyse profonde des caractéristiques de l'ontologie du domaine et de l'ontologie de la méthode. Après cette analyse, le cogniticien définit les opérations de correspondance. Finalement, les correspondances pourraient être accomplies automatiquement, une fois les opérations définies.

6.6 Résumé

Dans ce chapitre, nous avons suggéré un ensemble de types d'opérations de correspondance entre ontologies. Ces types sont: restriction, projection, alias, inversion, composition, union, intersection et complément. Nous avons exprimé ces opérations en Loom. Ces opérations ne doivent être considérées ni comme un ensemble minimal, ni comme un ensemble exhaustif, mais plutôt comme un cadre de base à partir duquel le cogniticien pourra faire les correspondances entre ontologies.

Notre principal contribution a été de définir et rendre opérationnel les opérations de correspondance entre une ontologie du domaine et une ontologie de la méthode. Ces opérations ne sont pas offertes présentement pour les langages KADS existants.

Les avantages des opérations de correspondance:

- L'indépendance entre les ontologies de méthode et les ontologies du domaine.
- Une même ontologie du domaine peut être vue sous différents points de vue.

Nous avons utilisé comme exemple une adaptation d'une ontologie du domaine médical pour la méthode C&D. Notre représentation de C&D est basée sur la description formelle de cette méthode dans le langage $(ML)^2$. Avec cet exemple, nous avons montré que même avec une équivalence sémantique entre les deux ontologies, il faut expliciter des opérations de correspondance de façon à établir une correspondance syntaxique entre les définitions formelles des ontologies.

Finalement, nous avons discuté les limites de notre approche et nous l'avons comparée avec d'autres travaux connexes: les opérations de correspondance dans les langages pour KADS; les opérations de correspondance dans PROTÉGÉ-II; le glossaire de l'environnement SBF; et l'automatisation des opérations de correspondances dans l'approche des méthodes neutres par rapport à la tâche.

Dans le prochain chapitre, nous analysons les méthodes P&R et C&D en ce qui concerne leur réutilisabilité.

Chapitre 7

Réutilisation de méthodes de résolution de problèmes

Dans les chapitres précédents, nous avons décrit deux méthodes de résolution de problèmes, P&R et C&D, en utilisant des ontologies de méthode. Ces ontologies sont employées pour spécifier les inférences et la tâche. Nous avons montré comment cette description dévoile la structure des rôles des connaissances spécifique aux méthodes et facilite leur compréhension. Nous avons aussi défini des opérateurs de correspondance pour réutiliser ces méthodes dans différentes applications. Dans ce chapitre, nous analysons notre travail, en considérant les limitations et les perspectives par rapport à la réutilisation de ces méthodes.

Dans la première partie, nous faisons une analyse de P&R. Nous avons déjà parlé de la nature compilée des connaissances de cette méthode dans la section 5.4.1. Nous reproduisons ici une analyse [Fensel et Straatman, 1996] qui compare P&R avec une méthode très générale de configuration, appelée *Générer et Tester* (G&T).

Dans la deuxième partie, nous analysons C&D par rapport à la tâche de diagnostic médical [Patil, 1988], en la comparant avec d'autres systèmes du même type. Ensuite, nous comparons C&D avec une approche qui présente une description plus neutre par rapport à la tâche [Van Heijst et Anjewierden, 1996].

Finalement, dans la troisième partie du chapitre, nous discutons des problèmes

et des limitations de notre travail.

7.1 Analyse de P&R

Nous avons déjà souligné dans la section 5.4.1 que, comme P&R a été définie initialement pour résoudre la tâche VT, la méthode est donc très spécifique et, par conséquent, moins réutilisable pour d'autres problèmes de configuration.

Dans cette section, nous montrons une autre analyse [Fensel et Straatman, 1996] qui justifie la nature compilée de rôles des connaissances de P&R pour des raisons d'efficacité. Pour démontrer cela, une nouvelle méthode pour une tâche configuration est présentée, *Générer et Tester* (G&T), où les connaissances sont décompilées. Cette analyse montre que G&T, à cause de la nature décompilée des connaissances, est une méthode inefficace.

7.1.1 G&T

Dans la figure 7.1, la structure d'inférence de la méthode G&T est présentée. L'inférence ***generate*** produit toutes les configurations possibles (**possible designs**). L'inférence ***requirement test*** filtre, parmi les configurations possibles, les solutions désirées (**desired designs**), en considérant les conditions requises par l'utilisateur (**requirements**). L'inférence ***constraint test*** filtre, parmi les configurations possibles, les solutions valides (**valid designs**), en considérant les contraintes (**constraints**). L'inférence ***intersect solutions*** sélectionne les solutions communes à l'ensemble de solutions désirées et valides (**selected solutions**). L'inférence ***select solution*** choisit la meilleure solution (**optimal solution**), en se basant sur un critère de préférence (**preference**).

On voit que toutes les différences conceptuelles des rôles des connaissances sont explicitées: (1) un rôle pour toutes les configurations possibles; (2) un rôle pour toutes les configurations valides; (3) un rôle pour toutes les configurations désirées; et (4) un rôle pour la meilleure configuration.

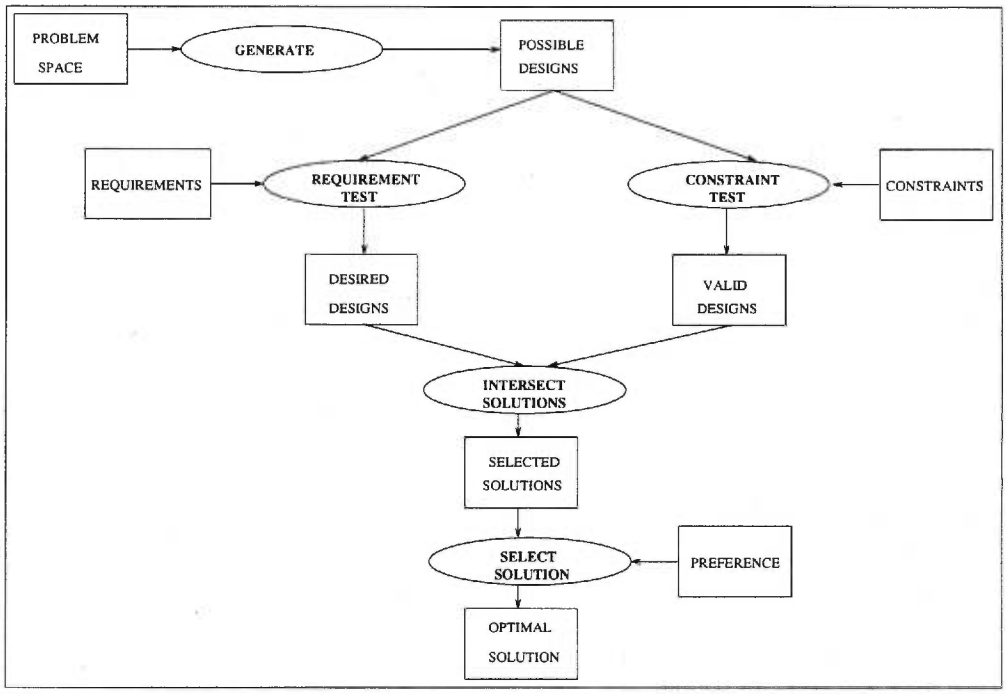

FIG. 7.1 - *Structure d'inférence de Générer et Tester (adapté de [Fensel et Straatman, 1996]).*

Le problème avec cette méthode est qu'elle très inefficace, parce que l'espace de recherche est trop large. Même pour un nombre fini de paramètres, l'espace de recherche croît exponentiellement par la production de toutes les combinaisons de paramètres et leurs valeurs, rendant le problème intraitable.

Pour résoudre ce problème, G&T est modifiée de façon, à trouver une solution acceptable au lieu de la meilleure solution. Pour y arriver, il y a un changement dans les inférences: ***generate***, au lieu de générer toutes les solutions possibles, génère m solutions à la fois; ***select solution***, au lieu de trouver la meilleure solution, va trouver une solution acceptable selon un seuil prédéfini.

Avec cette restriction, la méthode est plus spécifique, donc plus efficace, par l'affaiblissement du critère de solution: une solution acceptable au lieu de la meilleure solution. Cependant, même avec cette nouvelle restriction, la méthode reste encore inefficace par rapport à P&R.

7.1.2 P&R

Ici, nous analysons la différence entre les rôles des connaissances de G&T et de P&R. Toujours, selon l'analyse trouvée en [Fensel et Straatman, 1996], nous constatons que dans le cas de P&R, les distinctions conceptuelles sont obscures, parce que les connaissances sont de nature compilée.

Par exemple, l'inférence ***propose*** de P&R (figure 5.2) génère déjà une solution acceptable, selon les conditions requises par l'utilisateur. Ainsi, les rôles des connaissances pour ***propose*** contiennent implicitement des heuristiques pour trouver, à la fois, une solution désirée par l'utilisateur et une solution acceptable. Donc, les rôles des connaissances des inférences ***generate*** et ***requirement test*** de G&T sont compilés dans les rôles des connaissances pour l'inférence ***propose*** de P&R. Cette compilation de connaissances permet de représenter des heuristiques connues par l'expert qui diminuent l'espace de recherche, mais au prix d'une perte de clarté dans la description des rôles des connaissances.

Dans l'inférence ***revise***, les rôles des connaissances correspondant aux procédures

de réparations responsables pour remédier aux violations de contraintes vont obtenir une solution acceptable. Cela veut dire que les procédures de réparation contiennent implicitement une solution acceptable selon un seuil préétabli.

En effet, les connaissances heuristiques obtenues de l'expert permettent à la méthode P&R d'obtenir la solution d'une façon efficace. L'expert, avec son expérience, sait déjà quelles connaissances génèrent des solutions désirées et possibles. Ce ne sont donc pas toutes les solutions possibles qui sont générées comme dans le cas de G&T, mais seulement les solutions désirées. Les connaissances heuristiques de la méthode rendent possible l'obtention d'une solution acceptable dans un temps raisonnable, ce qui n'est pas le cas pour G&T. Cela veut dire que l'ontologie de la méthode construite pour P&R dévoile la structure des connaissances utilisée par la méthode, mais d'autres distinctions conceptuelles des connaissances sont obscurcies par l'utilisation des heuristiques obtenues de l'expert.

Toutefois, si P&R gagne en efficacité par l'utilisation de connaissances heuristiques, elle perd en réutilisabilité. La difficulté d'exprimer les distinctions conceptuelles des rôles des connaissances complique la réutilisation de P&R pour d'autres applications que la tâche VT.

Les rôles des connaissances de G&T sont mieux définis que les rôles de P&R. Dans ce sens, la méthode G&T est plus réutilisable que la méthode P&R, parce que les définitions des rôles des connaissances de G&T ne supposent pas des heuristiques du domaine, comme c'est le cas pour P&R.

En résumant, les rôles des connaissances de P&R peuvent être vus comme une "compilation" des rôles de G&T. Dans la définition des rôles de P&R, il y a des connaissances heuristiques définies implicitement, dans le but de réduire l'espace de recherche. Ainsi, les distinctions conceptuelles de G&T sont perdues en P&R. D'un autre côté, la compilation des connaissances de P&R rend la méthode efficace. Autrement dit, pour rendre un problème intraitable traitable, il faut ajouter des heuristiques. Ainsi, pour ne pas explorer tout l'espace de recherche existant (problème intraitable), des heuristiques sont appliquées pour trouver la solution plus rapidement (problème traitable).

Cela nous ramène à la dichotomie de l'adéquation épistémologique et pragmatique des représentations de connaissances [Lenat et Guha, 1990]. Une représentation est adéquate épistémologiquement, si elle représente tous les aspects du domaine nécessaires à la résolution du problème. Une représentation est adéquate pragmatiquement, si elle rend le problème soluble d'une manière efficace. Ces objectifs sont parfois incompatibles. Dans P&R par exemple, plusieurs distinctions conceptuelles sont perdues ou ne sont pas claires.

Cependant, si on est intéressé par la réutilisabilité de méthodes de résolution de problèmes, ce qu'il faut envisager, c'est l'adéquation épistémologique. Cela permet de comprendre les rôles des connaissances et, éventuellement, dans le contexte d'une application, d'ajouter des connaissances heuristiques spécifiques à l'application. Autrement dit, dans le cadre d'une application, il est plus facile d'ajouter des heuristiques dans une méthode générale que d'essayer de réutiliser une méthode où l'utilisation des heuristiques est considérée d'avance.

Ainsi, si le but est de promouvoir la réutilisabilité, nous croyons que la définition de méthodes générales comme G&T ou comme la méthode générale pour configuration paramétrique [Motta et Zdrahal, 1996] est plus intéressante que des méthodes qui s'appuient sur l'utilisation des connaissances heuristiques du domaine comme P&R. Comme les méthodes générales décrivent plus clairement les distinctions conceptuelles de rôles des connaissances, il est plus facile de les comprendre, de les adapter et de les rendre plus efficaces pour des applications spécifiques.

7.2 Analyse de C&D

Dans cette section, nous analysons la méthode C&D (chapitre 6) par rapport à sa réutilisabilité, où nous confrontons deux visions différentes.

Nous comparons d'abord C&D avec d'autres systèmes de diagnostic médical. L'idée est d'illustrer plusieurs distinctions conceptuelles de rôles des connaissances spécifiques à la tâche de diagnostic médical que C&D ne présente pas.

Nous présentons ensuite un travail qui favorise des *méthodes neutres par rapport*

à la tâche ("task-neutral problem-solving methods"). Selon cette perspective, C&D est très biaisée par rapport à la tâche de diagnostic et restreint sa réutilisation.

Nous avons donc deux argumentations contradictoires: (1) avoir des méthodes plus spécifiques par tâche pour rendre la réutilisation pour cette tâche particulière plus facile; (2) avoir des méthodes plus neutres par tâche pour rendre la réutilisation plus étendue. Nous concluons la section en prenant le parti de la première vision et en donnant nos arguments pour appuyer cette perspective.

7.2.1 D'autres systèmes médicaux

Dans cette section, nous utilisons la revue sur les SBCs pour les systèmes médicaux [Patil, 1988] pour illustrer des distinctions conceptuelles de rôles qui rendent le processus de diagnostic plus efficace. Nous analysons trois distinctions conceptuelles de rôles des connaissances souhaitables pour le diagnostic médical qui ne sont pas explicitées dans C&D.

Réduction du nombre d'hypothèses

Dans les systèmes pour le diagnostic médical, un problème très connu est celui de réduire l'ensemble initial d'hypothèses afin d'éviter au système d'avoir à évaluer continuellement un grand nombre de maladies. Dans notre exemple du chapitre 6, le cas de diagnostic est simple et C&D se montre très efficace pour le résoudre. Cependant, si on imagine un réseau plus gros, le nombre d'hypothèses initiales peut augmenter rapidement.

Une technique appliquée pour restreindre le nombre d'hypothèses est d'avoir un *ensemble d'hypothèses actives*. Cet ensemble peut être géré de différentes manières. Une première manière est d'associer un point à chaque symptôme. À chaque hypothèse est associée une valeur qui correspond à l'addition des points de tous les symptômes observés qui soutiennent l'hypothèse. Ainsi, pour prévenir l'augmentation du nombre d'hypothèses, celles dont la valeur est sous une limite préétablie sont enlevées de l'ensemble d'hypothèses actives.

Toutefois, cette stratégie n'évite pas toujours l'augmentation de l'ensemble d'hypothèses à cause des symptômes non spécifiques, c'est-à-dire qui se trouvent dans plusieurs maladies, comme la fièvre, la toux, etc. Ces symptômes renforcent plusieurs hypothèses à la fois et ne permettent pas la diminution du nombre d'hypothèses actives.

Une manière de traiter ce problème est de séparer les symptômes en deux types: "déclencheurs" et "non-déclencheurs", ces derniers étaient considérés seulement après l'activation de l'hypothèse afin de la soutenir.

Parmi les distinctions conceptuelles pour réduire le nombre d'hypothèses à évaluer, nous n'en trouvons aucune dans les rôles des connaissances de l'ontologie de méthode de C&D. Le concept d'ensemble d'hypothèses actives n'existe pas, parce que toutes les hypothèses sont évaluées. De plus, tous les symptômes observés sont considérés indistinctement pour appuyer les hypothèses.

Organisation de la hiérarchie de maladies

Une autre technique pour diminuer le nombre d'hypothèses est d'organiser les maladies dans une hiérarchie. Dans ce cas, chaque hypothèse correspond à un groupe de maladies, et non à une seule maladie. Les avantages de regrouper les maladies, en les traitant en groupe sont les suivants:

- Le processus de différenciation est appliqué à plusieurs maladies à la fois, ce qui évite de s'impliquer très tôt dans une hypothèse spécifique.

- Le rejet d'une hypothèse, lors du processus de différenciation, implique que plusieurs maladies sont éliminées à la fois, au lieu d'une seule.

- L'organisation hiérarchique des maladies rend la construction, l'entretien et la compréhension de la base de connaissances plus facile.

Patil donne deux exemples des hiérarchies pour les maladies de reins (figure 7.2). Une première différencie les maladies de reins par les distinctions anatomiques; une

deuxième hiérarchie différencie les maladies par l'aspect temporel, c'est-à-dire chronique ou aiguë.

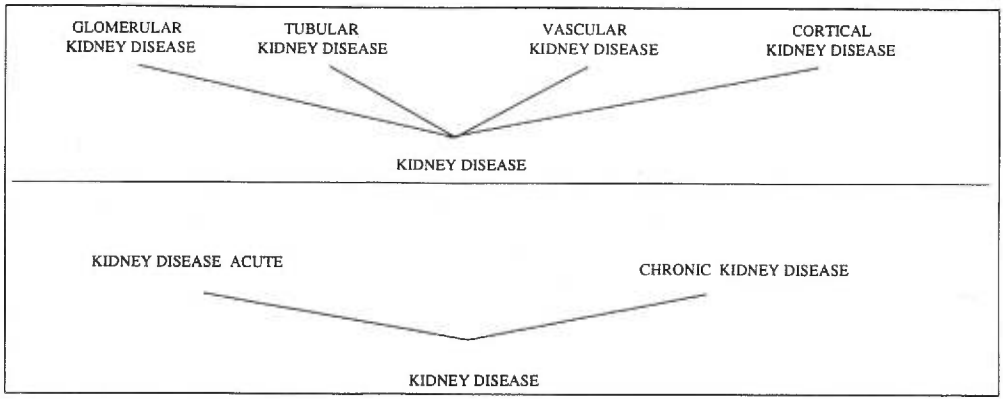

FIG. 7.2 - *Différentes alternatives pour l'organisation de la hiérarchie de maladies (adapté de [Patil, 1988]). La première hiérarchie fait une différenciation par l'aspect temporel (chronique ou aiguë) et la deuxième, par l'aspect anatomique (glomerulaire, tubulaire, vasculaire, cortical).*

Dans le réseau causal de C&D, l'organisation hiérarchique des maladies est implicite, mais le principe qui l'oriente n'est pas clair. Nous ne savons pas si elle est organisée en termes de différences anatomiques ou temporelles et, si c'est le cas, dans quel ordre ces différences sont considérées. Dans le réseau médical présenté à la section 6.2, l'organisation causale représente plutôt l'espace d'hypothèses dans la tête de l'expert qu'une organisation bien fondée des connaissances du domaine. Ces connaissances sont compilées et représentent des heuristiques pour réduire l'espace de recherche du problème.

Dans C&D, le principe d'organisation de la hiérarchie de maladies est laissé au cogniticien; ce principe peut ne pas être clair, comme nous avons vu pour l'exemple médical du chapitre précédent.

Hiérarchies multiples

L'organisation hiérarchique de maladies n'est pas claire car certaines maladies peuvent affecter plusieurs organes à la fois; elles partagent des symptômes dispersés

dans la hiérarchie. Des principes différents d'organisation de la hiérarchie de maladies donnent des résultats distincts ou peuvent être plus ou moins efficaces selon la situation. Par exemple, si dans la figure 7.2, le système différencie les maladies par l'aspect anatomique, la première hiérarchie est adéquate. Cependant s'il différencie plutôt par l'aspect temporel, la deuxième serait plus adéquate.

Les hiérarchies multiples caractérisent les maladies selon des dimensions différentes et peuvent caractériser une même maladie selon son étiologie (l'association entre les maladies et leurs causes), son anatomie (l'association entre les maladies et les organes affectés), ses symptômes (l'association entre les maladies et les disfonctions produites par les maladies), etc.

Ces dimensions multiples permettent d'aborder le problème avec différentes directions. Pour réduire le nombre d'hypothèses après avoir obtenu les différentes caractérisations des maladies, il faut utiliser la technique *heuristique d'intersection.* Par exemple, si l'étiologie de la maladie est infectieuse et l'organe affecté (caractérisation anatomique) est le rein, l'heuristique d'intersection donnera comme ensemble d'hypothèses les maladies infectieuses de reins.

L'utilisation des hiérarchies multiples permet de faire une caractérisation plus riche des maladies et d'orienter l'obtention du diagnostic selon plusieurs facettes du problème. Cependant, cette technique suppose que le patient souffre d'une seule maladie à la fois, ce qui n'est pas pas toujours vrai. Il arrive souvent qu'un malade présente plusieurs maladies à la fois.

C&D est constitué d'un seul réseau causal qui considère les maladies seulement par les symptômes et les observations. C&D ne va probablement pas obtenir de bons résultats pour des cas plus complexes différenciés seulement selon diverses dimensions.

Nous avons montré que la méthode C&D ne fait pas de distinction conceptuelle dans ses rôles des connaissances pour optimiser le diagnostic. C'est au cogniticien d'ajouter ces distinctions à C&D ou de mettre ces distinctions implicitement à l'aide de connaissances heuristiques.

7.2.2 Méthodes neutres par rapport à la tâche

Dans cette section, nous présentons un autre travail qui soutient que les méthodes de résolution de problèmes sont trop engagées par rapport à la tâche et que cela les rend moins réutilisables. Cette approche [Van Heijst et Anjewierden, 1996, Beys et al., 1996], appelée "task neutral", propose la définition de *méthodes de résolution de problèmes neutres par rapport à la tâche*, où les rôles des connaissances de méthodes sont spécifiés indépendamment de la tâche. Une méthode seraient rendue spécifique pour une tâche seulement lorsqu'elle est associée à la définition de la tâche dans le modèle de l'expertise.

D'après cette vision, une spécification de C&D où les rôles sont appelés "observations" et "hypothèses" biaise la méthode vers une tâche spécifique, dans ce cas le diagnostic. Elle limite inutilement l'applicabilité de la méthode.

Pour résoudre le problème, il faut spécifier une méthode en termes neutres et laisser le cogniticien associer les termes de la tâche aux rôles des connaissances de la méthode. Dans ce genre de définition neutre, les termes "observations" et "hypothèses" sont remplacés par les termes "feuilles" ("leaves") et "couvertures" ("covers"). Le réseau causal est donc vu comme un graphe dirigé.

Cependant, nous ne sommes pas d'accord avec cette approche car elle implique un retour au niveau symbolique qui amène deux problèmes: l'indexation des méthodes et l'ajout de connaissances spécifiques pour une application.

Le problème associé à l'indexation consiste à trouver une méthode parmi d'autres pour accomplir une tâche. C'est au cogniticien de trouver, parmi différentes méthodes, une méthode neutre pour résoudre son problème dans le cadre d'une application. S'il y a plusieurs méthodes, il serait plus facile d'avoir dans la description de la méthode des indications pour aider à l'associer à cette tâche spécifique.

L'autre problème des méthodes neutres est la surcharge de travail laissé au cogniticien qui va même à l'encontre des objectifs initiaux des méthodes de délimitation de rôles. Selon McDermott [McDermott, 1988], ces méthodes sont limitées en étendue, mais elles fournissent une aide substantielle à l'automatisation des tâches car elles sou-

lagent le cogniticien d'avoir à ajouter des connaissances spécifiques par tâche, parce qu'elles ont été définies par la méthode.

En résumé, les méthodes qui décrivent des tâches spécifiques et complexes comme le diagnostic médical s'avèrent plus intéressantes que des méthodes neutres. C&D décrite en termes neutres peut potentiellement être utilisée dans différentes applications, mais elle implique un coût élevé de reconfiguration pour une application spécifique.

7.3 Discussion

Dans les sections précédentes, nous avons analysé les deux méthodes étudiées dans cette thèse. On remarque que, si ces méthodes représentent un progrès significatif par rapport aux systèmes de première génération, elles présentent encore beaucoup de restrictions qui en empêchent la réutilisation et la compréhension.

P&R, par exemple, est une méthode spécifique qui compte sur une grande quantité de connaissances heuristiques associées au domaine pour la rendre efficace, mais cela la rend moins réutilisable pour d'autres applications. D'un autre côté, C&D est une méthode plus générale que P&R, réutilisable pour différentes applications, mais qui, dans le contexte d'une tâche complexe comme le diagnostic médical, manque de connaissances qui pourraient l'optimiser et faciliter le travail du cogniticien pour l'adapter.

Cette analyse nous laisse encore beaucoup questions à propos de méthodes de résolution de problèmes:

1. Est-il préférable d'avoir des méthodes générales, mais inefficaces? Ou d'avoir de méthodes spécifiques qui supposent une utilisation massive des connaissances heuristiques associées au domaine, mais sans clarifier les distinctions conceptuelles des rôles des connaissances?

2. Est-il préférable d'avoir des méthodes spécifiques par tâches complexes, mais qui perdent en étendue de réutilisation? Ou d'avoir des méthodes générales, décrites sans être biaisées par une application particulière, mais qui sont difficiles

à trouver et qui demandent beaucoup de connaissances additionnelles pour être utilisées par une tâche spécifique?

Nous avons esquissé les réponses suivantes à ces questions:

1. Il est préférable d'avoir des méthodes générales et inefficaces, mais à partir desquelles on peut ajouter des restrictions sur le type de solutions obtenues pour les rendre moins générales, et d'ajouter des connaissances heuristiques spécifiques au domaine pour les rendre plus efficaces.

2. On cherche à avoir des méthodes pour des tâches complexes, avec une réutilisation plus restreinte, mais qui sont plus faciles à identifier pour l'application en question et qui demandent moins d'effort d'adaptation par le cogniticien.

Notre première réponse se rapproche d'un processus de raffinement, où les suppositions sont ajoutées pour configurer la méthode pour une application spécifique [Akkermans et al., 1993]. Cette approche suggère que le processus de construction d'une méthode consiste en une succession de raffinements dans le but d'expliciter les suppositions par rapport à la compétence de la méthode pour résoudre un problème et aux connaissances du domaine requises. L'idée est de commencer avec une méthode générale (par exemple, G&T) et d'y ajouter des suppositions pour rendre la méthode plus spécifique et efficace.

En ce qui concerne notre deuxième réponse, nous avons vu qu'il est difficile d'associer C&D à une tâche, si C&D est définie en termes neutres. Dans ce cas, la *distance cognitive* [Eriksson et al., 1995] entre la méthode et la tâche sont grandes. Cette distance cognitive mesure le degré de "neutralité" de la méthode par rapport à la tâche. Plus la distance cognitive est grande, plus il est difficile de réutiliser la méthode pour la tâche. Nous croyons donc que pour aider la modélisation des SBCs, en accord avec le niveau des connaissances, il est plus intéressant d'avoir des méthodes plus spécifiques par tâche pour simplifier leur indexation et configuration.

L'analyse de ces méthodes peut aussi être formulée en termes de modèles *profonds* et de *surface* ("deep knowledge *versus* surface knowledge") [Steels, 1990]. Le premier

explicite les distinctions conceptuelles de rôles des connaissances, alors que le second laisse implicite ces distinctions.

Ce qui caractérise les systèmes de surface sont les règles heuristiques qui associent les observations à leurs causes possibles. Ces règles sont aussi appelées connaissances associationnelles; elles représentent des raccourcis ("shortcuts") sautant des étapes intermédiaires du raisonnement pour accélérer la solution du problème. De plus, le raisonnement heuristique ne fait pas référence au fonctionnement interne du système, mais il fait des associations entre les observations externes et les conclusions plausibles.

Ce qui caractérise les systèmes du type profond, c'est d'avoir des modèles qui représentent un domaine d'une manière complète et systématique. Pour le diagnostic, par exemple, il y a des systèmes basés sur les modèles de la structure et du comportement d'un dispositif [Davis et Hamscher, 1988]. Le modèle de la structure représente comment les pièces d'un dispositif sont interconnectées. Cette modélisation est, en général, hiérarchique (composants et sous-composants) et isomorphique, dans le sens où elle doit avoir des objets et leur interconnexions correspondant à ceux du dispositif. Le modèle du comportement est décrit par des expressions qui capturent les interconnexions entre les valeurs des objets du dispositif. Ces systèmes constituent une modélisation plus complète et claire que les systèmes basés sur des heuristiques, mais ils sont, en général, inefficaces. Ils ont été définis pour surmonter les problèmes des systèmes avec des connaissances heuristiques, mais, dans leur forme pure, ils rendent le problème intraitable. Il faut ajouter des suppositions qui restreignent la compétence de ces méthodes, mais les rendent plus efficaces.

D'autres solutions proposées dans la littérature représentent un compromis par rapport à ces approches divergentes. L'idée est de n'avoir ni modèles généraux ou profonds, ni modèles spécifiques ou de surface, mais des modèles qui identifient les rôles des connaissances de façon à amener à une solution acceptable et efficace, dans un nombre limité de situations [Van de Velde, 1993]. Dans cette perspective, il y a des systèmes qui combinent différents types de modèles de façon à obtenir les avantages de chacun: l'efficacité de systèmes de surface et la robustesse des systèmes profonds. Dans ces systèmes, par exemple, il y aura un modèle plus profond, pour rendre le

système robuste et un autre, en surface, pour des raisons d'efficacité.

Steels [Steels, 1984], par exemple, a proposé de construire les SBCs avec les deux types de modèles: un modèle de surface, pour les problèmes routiniers, et un modèle profond, pour les problèmes plus complexes et inattendus. Dans la plupart des cas, on utiliserait le modèle de surface parce qu'il est plus efficace. Le modèle profond serait utilisé beaucoup moins souvent, pour les problèmes complexes. De plus, le modèle profond pourrait être utilisé aussi pour donner des explications plus claires et bien fondées.

À l'instar du travail de Steels, le système Gorgius [Simmons, 1992] est un exemple d'utilisation des multiples modèles des connaissances. Cependant, dans Gorgius, au lieu d'utiliser le modèle de surface pour des problèmes routiniers et le modèle profond pour des problèmes complexes, les différents modèles sont utilisés dans différentes phases de la résolution du problème. Gorgius est un système qui utilise la méthode *Générer, Tester et Épurer* (G&T&D - "generate-test-debug") pour résoudre des problèmes d'interprétation des données géologiques. Dans Gorgius, les connaissances sont de deux types: associationnelles (heuristiques) et causales (basées sur modèles). Les connaissances associationnelles sont utilisées pour générer des hypothèses initiales, et les connaissances causales sont utilisées pour tester et épurer les hypothèses. Ainsi, la méthode G&T&D définit deux types de rôles des connaissances: un rôle associationnel et un rôle causal. Le connaissances associationneles sont du type surface qui rendent la méthode efficace et les causales sont du type profond qui la rendent plus robuste.

Ainsi, nous avons vu qu'il est difficile de trouver le bon compromis entre généralité et spécificité, et entre efficacité et clarté des rôles des connaissances. Nous avons tenté de répondre à ces questions, mais nous ne prétendons pas avoir réglé la situation. Notre analyse peut être vue comme une tentative de résoudre des problèmes très complexes sans avoir encore de solutions définitives. Ce que nous pouvons affirmer avec un peu plus de fermeté est que l'explicitation des rôles des connaissances à l'aide des ontologies de méthodes est une étape importante dans la compréhension et la réutilisation de méthodes de résolution de problèmes. De plus, cette ontologie peut servir à choisir une méthode pour accomplir une tâche spécifique.

7.4 Résumé

Nous avons, dans ce chapitre, montré les caractéristiques des rôles des connaissances de méthodes P&R et C&D qui limitent leur réutilisation. Au sujet de P&R, nous avons vu que la difficulté de réutiliser la méthode est due à la nature compilée des connaissances, ce qui complique les distinctions conceptuelles claires entre les rôles des connaissances. Nous avons vu aussi que cela est justifiable pour des raisons d'efficacité. Cependant, si on cherche à avoir des méthodes réutilisables, il est préférable d'avoir des méthodes moins efficaces, mais avec de rôles des connaissances où les distinctions conceptuelles sont claires. La méthode réutilisable et non efficace pourra être adaptée pour une application spécifique. Dans le contexte de cette application, il est fort probable que des heuristiques soient ajoutées pour rendre la méthode plus efficace. Toutefois, il est important d'avoir, au moins au départ, une description de la méthode où les rôles des connaissances sont explicités pour en faciliter la réutilisation.

Pour C&D, nous avons vu que, malgré sa généralité, pour des tâches complexes comme le diagnostic médical, cette méthode révèle encore beaucoup de lacunes par rapport à la définition des rôles des connaissances. À notre avis, il serait donc plus intéressant d'avoir des méthodes plus spécifiques par tâche, avec une étendue d'utilisation plus restreinte, mais avec plus de support pour cette tâche spécifique. Cela, au lieu d'avoir des méthodes neutres par rapport à la tâche, potentiellement réutilisables pour différentes tâches, mais qui donnent beaucoup de travail au cogniticien pour y ajouter des connaissances nécessaires pour une tâche spécifique.

L'identification de la meilleure approche pour décrire des méthodes de résolution de problèmes réutilisables reste encore un problème ouvert. Nous soulignons, cependant, l'importance de décrire les rôles des connaissances au moyen de l'ontologie de la méthode pour faciliter l'indexation, la compréhension et la réutilisation de ces méthodes.

Chapitre 8

Conclusion

Dans cette thèse, nous avons étudié le problème de la réutilisation des méthodes de résolution de problèmes. Dans le présent chapitre, nous faisons un rappel des contributions de notre travail et nous présentons des perspectives pour des recherches futures.

8.1 Contributions de la thèse

La principale hypothèse de notre travail est que les méthodes de résolution de problèmes supposent une organisation particulière de rôles des connaissances. Au chapitre 3, nous avons critiqué KADS, qui considère les rôles comme des étiquettes abstraites sans structure sous-jacente. En effet, KADS fait la description des connaissances dans la couche du domaine et suppose une structure "plate" des rôles des connaissances. Ainsi, nous avons reproché l'interaction très générale des rôles des connaissances et des inférences en KADS.

Pour décrire les méthodes, nous avons développé le langage K-Loom décrit au chapitre 4 qui représente les connaissances associées aux méthodes selon le modèle de l'expertise de KADS et basé sur une représentation terminologique des connaissances. K-Loom présente une syntaxe déclarative pour représenter les connaissances, mais il est aussi un langage opérationnel qui offre un feed-back rapide du modèle. L'avantage

de K-Loom par rapport aux autres langages basés sur KADS est l'intégration de la description des couches du domaine, des inférences et de la tâche, en plus des opérations pour faire la correspondance entre les connaissances de la couche du domaine et les rôles des connaissances de la couche des inférences. De plus, K-Loom offre des primitives similaires à celles de CML, le langage semi-formel de KADS, ce qui rend facile une modélisation à la KADS en K-Loom.

Au chapitre 5, nous avons précisé les rôles des connaissances de la méthode au moyen d'une ontologie qui définit ces rôles en termes des concepts et de leurs relations. La définition de l'ontologie de la méthode donne à la couche des inférences de KADS le même pouvoir d'expression pour décrire les rôles des connaissances que celui dans la couche du domaine pour décrire les connaissances du domaine. Dans KADS, c'est la couche du domaine qui est responsable pour la description des connaissances en fonction de leur utilisation. L'avantage d'avoir l'ontologie de la méthode dans la couche des inférences et les connaissances du domaine dans la couche du domaine est de pouvoir exprimer dans chaque couche les particularités associées soit à la méthode, soit au domaine.

Le concept d'ontologie de la méthode n'est pas original à notre thèse. Ce qui l'est, toutefois, est son utilisation pour préciser les inférences de la méthode. PROTÉGÉ-II, qui utilise aussi ce concept, ne fait pas de distinction précise entre le modèle conceptuel et l'implantation. Ainsi, nous avons défini déclarativement les instances concrètes des inférences en fonction des rôles des connaissances. Cela permet une meilleure compréhension de la méthode sans avoir à fouiller le code implanté.

Pour définir les inférences, nous avons adopté la même approche que KADS pour définir la typologie d'inférences canoniques: nous considérons que les inférences opèrent sur une ontologie prédéfinie (section 5.4.1), dans notre cas l'ontologie de la méthode. On a remarqué que les inférences canoniques de KADS sont plus *réutilisables* que les inférences de la méthode, parce que plus générales. Cependant, les inférences de la méthode sont plus *utilisables*, parce que plus spécifiques et adaptées à la méthode. De plus, les inférences décrites selon l'ontologie de la méthode favorisent la compréhension de la méthode et peuvent, dans un certain sens, contribuer à sa

réutilisation. Cela parce qu'en comprenant ce que la méthode fait, il est plus facile de la modifier et de l'intégrer dans différentes applications.

Toujours en considérant que la structure de rôles des connaissances ne correspond pas nécessairement à la structure des connaissances du domaine qui va être appliquée à la méthode, nous proposons un ensemble d'opérateurs pour faire la correspondance entre la couche du domaine (ontologie du domaine) et la couche des inférences (ontologie de la méthode). La définition de ces opérateurs est un pas important vers la réutilisation de méthodes de résolution de problèmes. Avec cet ensemble d'opérations, nous proposons une solution qui représente un compromis entre les deux visions opposées à propos de la réutilisation de connaissances (chapitre 6). Une première propose que les connaissances peuvent être représentées seulement en considérant une application spécifique (le problème de l'interaction), tandis que l'autre vision soutient que les connaissances doivent être représentées de la façon la plus générale possible pour en augmenter la réutilisabilité. Nous surmontons partiellement l'incompatibilité entre ces deux visions par l'établissement d'opérations de correspondance entre l'ontologie de la méthode, qui représente les connaissances en fonction de leur utilisation par la méthode et les connaissances du domaine définies indépendamment de la méthode.

Cette solution est une réponse partielle à ce problème, parce que nous ne considérons que le cas de différences syntaxiques entre les représentations de l'ontologie de la méthode et de l'ontologie du domaine. Nous supposons une correspondance sémantique entre les connaissances du domaine et les rôles des connaissances de la méthode. S'il n'y a pas une correspondance sémantique entre les deux ontologies, nous considérons que l'ontologie du domaine n'est pas réutilisable pour la méthode.

Évidemment, notre description des méthodes n'est pas complète. Nous n'avons pas considéré certains aspects de la méthode, comme une description plus précise du type de solution obtenu, par exemple les suppositions téléologiques présentées à la section 5.4.1.

Au chapitre 7, nous avons montré les limites de réutilisabilité des méthodes de résolution de problèmes. D'abord, les faiblesses des méthodes à limitation de rôles qui présentent des caractéristiques restreignant leur réutilisation. P&R, par exemple,

n'est pas une méthode générale de configuration et ses rôles des connaissances ne sont pas complètement décrits, parce que la méthode s'appuie sur les connaissances heuristiques du domaine. C&D, par contre, est une méthode plus générale, dont nous avons vu les limites pour une tâche complexe comme le diagnostic médical.

Nous avons aussi discuté des limites des méthodes de résolution de problèmes en général. En effet, nous ne croyons pas à une réutilisation immédiate et directe des méthodes. Dans le contexte d'une application spécifique, le cogniticien doit analyser les méthodes existentes, potentiellement applicables à son problème. À partir de cette analyse, il va pouvoir confronter son problème avec la méthode. Cette analyse va d'abord l'aider à être plus conscient des particularités de son application. Ensuite, il pourra configurer la méthode pour l'application en question.

Enfin, dans notre travail, nous avons progressé dans la compréhension et la réutilisation de méthodes de résolution de problèmes et de l'importance de la description de rôles de connaissances pour atteindre ce but. Nous avons vu que ces méthodes et leur niveau de description représentent un progrès par rapport aux systèmes de première génération, mais qu'il est difficile de trouver un bon équilibre en ce qui concerne la réutilisabilité et l'efficacité d'un côté, et la généralité et la spécificité de ces méthodes d'un autre côté.

8.2 Perspectives et travaux futurs

Une première suite de notre travail consiste à construire des outils pour constituer un environnement de modélisation, de conception et de validation des méthodes de résolution de problèmes développés en K-Loom. Ces outils devraient disposer d'interfaces graphiques pour aider la conception du modèle, pour l'épurer dans la phase d'implantation et pour valider et vérifier le système implanté. De plus, des interfaces graphiques permettront de visualiser le processus d'inférence, de façon à contribuer à l'étape d'explication du système. De même, nous pourrons explorer la construction des outils d'acquisition de connaissances spécifiques par méthode, comme dans les méthodes à limitation de rôles [Marcus, 1988, Eshelman, 1988], mais générés auto-

matiquement à partir de l'ontologie de la méthode et les configurer spécifiquement pour le domaine d'application en question, comme dans l'approche de PROTÉGÉ-II [Musen et al., 1994].

Nous n'avons pas travaillé dans cette thèse sur les aspects d'organisation d'une bibliothèque de méthodes ou de composants réutilisables et nous n'avons pas non plus traité du problème de l'indexation d'une bibliothèque de ce type. Ainsi, nous pensons qu'il sera important d'étudier la possibilité d'intégrer l'ontologie de la méthode, les inférences et les méthodes de résolution de problèmes développés en K-Loom, dans une bibliothèque existente. Autrement dit, il sera intéressant d'analyser la possibilité d'établir une correspondance entre les définitions en K-Loom et les composants et modèles génériques de la bibliothèque conceptuelle de CommonKADS, décrits semi-formellement et à un haut niveau d'abstraction.

Un autre aspect pragmatique à considérer sera de tester et valider nos idées avec d'autres méthodes. Après cette expérience avec les méthodes à limitation de rôles, nous croyons qu'il serait approprié d'étudier des méthodes de résolution de problèmes plus générales [de Kleer et Williams, 1987, Motta et Zdrahal, 1996]. Avec ces méthodes plus générales, nous pourrons analyser et vérifier la possibilité de les spécialiser et les intégrer pour différentes applications, en utilisant les opérateurs de correspondance que nous avons définis et en évaluant le coût d'utilisation de ces opérateurs pour adapter différentes applications aux méthodes.

La démarche proposée tout au long de cette thèse s'est surtout concentrée sur la définition des méthodes de résolution de problèmes. Cependant, il reste beaucoup de travail, si nous considérons l'autre aspect du problème de la réutilisation, celui de la construction d'ontologies du domaine réutilisables. Les problèmes concernant l'identification de ce qu'est un domaine, l'énorme quantité de connaissances associées à un domaine, la redondance des connaissances d'un domaine à l'autre, sont des sujets de recherche encore à poursuivre. Des travaux récents [Van Heijst et al., 1997, Valente et Breuker, 1996] vont également dans cette direction, mais il reste encore plusieurs questions ouvertes, qu'il sera intéressant d'explorer.

Finalement, une autre direction de recherche est la définition et la formalisation

des conditions d'applicabilité associées à une méthode. Nous avons remarqué que l'ontologie de la méthode et la définition des inférences est un aspect parmi d'autres constituant la spécification d'une méthode de résolution de problèmes. Dans la bibliothèque CommonKADS, le type du problème est l'index principal utilisé pour sélectionner les méthodes. Dans cette bibliothèque, les caractéristiques de rôles des connaissances de la méthode sont considérées comme un index secondaire pour choisir un modèle générique. Les recherches menées par [Benjamins et Pierret-Golbreich, 1996, Benjamins et al., 1996, Fensel et Benjamins, 1996] vont également dans cette direction. Dans ces travaux, les auteurs proposent une organisation plus systématique des conditions d'application des méthodes de résolution de problèmes, appelées de *suppositions*. Cependant, les problèmes d'expression de ces suppositions et de leur vérification sont encore sans solution.

Appendice A

Exécution en K-Loom de P&R appliquée à la tâche VT

Dans cet appendice, nous illustrons l'exécution de la méthode *Proposer et Réviser* appliquée à la tâche VT. L'ontologie de la méthode et les inférences ont été définies dans la section 5.3.2. La tâche a été définie dans la section 5.3.3 et la structure de contrôle correspond à celle de la figure 5.9, la version de P&R appelée EMR.

A.1 Données VT

Les données de la tâche VT sont décrites en détail dans le document écrit par Yost [Yost, 1994], appelé ici VT-Yost qui décrit la résolution d'un problème de configuration d'un ascenseur fait par un expert. Ce document est disponible sur l'Internet [1] et contient:

- Les paramètres d'entrée dont les valeurs doivent être fournies par l'utilisateur.
- Les procédures associées aux autres paramètres qui permettent de calculer leurs valeurs à partir des données d'entrée.
- Les contraintes associées aux paramètres.

1. http://camis.stanford.edu/projects/protege/sisyphus-2/s2-0.html

- Les modifications suggérées pour réparer les éventuelles violations de contraintes.
- Les paramètres de sortie dont les valeurs doivent être fournies à l'utilisateur.

Dans VT-Yost, toutes ces informations sont décrites en langue naturelle. Pour nous épargner le travail d'entrer toutes ces données dans notre système, nous avons considéré la possibilité de réutiliser une représentation déjà existante de ces données. Après avoir analysé les formalisations de VT en Ontolingua (VT-Ontolingua) et en OCML (VT-OCML) nous avons opté pour la formalisation en OCML.

VT-Ontolingua [Gruber et al., 1994] ne contient pas toutes les informations contenues en VT-Yost, comme cela a été montré en détail dans [Motta et Zdrahal, 1995] (section 5.4.1).

En raison des ces limitations de VT-Ontolingua et pour avoir accès aux données VT telles que sont décrites dans VT-Yost, nous avons préféré la modélisation VT-OCML (section 5.3.3). VT-OCML est une représentation modulaire et complète des données trouvées en VT-Yost. Nous avons traduit facilement les données VT-OCML en instances de notre ontologie de la méthode de P&R, en obtenant une représentation complète des données pour P&R.

A.2 Entrée des données

Pour accomplir la tâche VT dans notre système, il faut d'abord, dans Loom, charger les fichiers de K-Loom, de l'ontologie de la méthode, des inférences, de la définition de la structure de la tâche et des instances VT. Le fichier K-Loom contient les macros qui expriment les inférences et les tâches selon les définitions du chapitre 3, section 4. Le fichier de l'ontologie de la méthode contient les définitions de concepts et de relations pour utiliser P&R et le fichier des inférences contient la définition des inférences de P&R (section 5.3.2). Le fichier de la tâche contient la définition de la structure de la tâche de P&R (section 5.3.3). Le fichier de l'ontologie du domaine contient les définitions du domaine de l'ascenseur obtenues à partir du VT-OCML. En effet, ce fichier contient un ensemble de macros qui traduisent la représentation

VT-OCML en instances de l'ontologie de la méthode de P&R. Dans ce cas, comme les instances VT correspondent exactement aux définitions de l'ontologie de la méthode de P&R, on n'a pas besoin d'un fichier d'opérations de correspondance, où seraient définies les opérations pour faire l'association entre les connaissances du domaine et de la méthode.

```
(load "k-loom")
(load "ontologie-methode-PR")
(load "inferences-PR")
(load "task-PR")
(load "ontologie-VT")
```

A.3 Méthodes pour exécuter les inférences

Pour définir les inférences, Loom permet la déclaration d'un ensemble de *méthodes* associées à des *actions* génériques. La déclaration d'une action introduit une opération générique qui peut être appelée pour exécuter une opération spécifique. Chaque méthode spécifie une implantation d'une action du même nom. Si plus d'une méthode est applicable, Loom sélectionnera la méthode la plus adéquate à la situation. Ce type de stratégie de contrôle est une généralisation des techniques utilisées en programmation objet.

Ici, les méthodes sont responsables de la réalisation des inférences. Pour nous, l'avantage d'avoir utilisé les méthodes de Loom est de spécifier séparément la réalisation des inférences au niveau symbolique par la définition des méthodes. Comme les méthodes sont déclarées indépendamment des règles d'inférence, la lisibilité du code est augmentée et cela permet de comprendre la méthode sans se soucier des détails d'implantation. De plus, nous pouvons changer le niveau implantation sans avoir à changer la définition formelle.

Voici, la définition des actions auxquelles correspondent des méthodes pour accomplir les inférences:

```
(defaction Select-Parameter (?p))
(defaction Propose (?p))
(defaction Check (?ct))
```

```
(defaction Select-Violated-Constraint (?ct))
(defaction Revise (?ct))
```

Cette organisation en termes d'actions génériques et de méthodes spécifiques convient tout à fait à la tâche VT où nous avons, pour chaque paramètre, une procédure spécifique pour calculer sa valeur et, de la même façon, pour chaque contrainte, une procédure spécifique pour vérifier si la contrainte est violée.

Ainsi, l'inférence ***propose*** utilise une méthode spécifique pour calculer la valeur du paramètre sélectionné. La méthode pour calculer la valeur du paramètre **safety edge weight** est la suivante: si la valeur du paramètre **door opening type** est **side**, la valeur de **safety edge weight** doit être 7; si la valeur du paramètre **door opening type** est **centre**, la valeur de **safety edge weight** doit être 13.

```
(defmethod Propose (?p)
:title "Calcul of SAFETY-EDGE-WEIGHT"
:situation  (:same-as ?p SAFETY-EDGE-WEIGHT)
:response ((set-value 'SAFETY-EDGE-WEIGHT 'Has-Value
                (cond ((equal (Parameter-Value DOOR-OPENING-TYPE)
                              'SIDE)
                       7.0)
                      ((equal (Parameter-Value DOOR-OPENING-TYPE)
                              'CENTRE)
                       13.0)))))
```

De la même façon, l'inférence ***check*** est une méthode spécifique pour chaque contrainte à être vérifiée. La méthode pour vérifier si la contrainte **min cwt buffer quantity** est violée consiste en une vérifiaction de la valeur du paramètre **cwt buffer quantity** qui doit être supérieure à 1.

```
(defmethod Check (?ct)
  :title "Verify violation of  MIN-CWT-BUFFER-QUANTITY"
  :situation (:same-as ?ct MIN-CWT-BUFFER-QUANTITY)
  :response
          ((if (>= (Parameter-Value CWT-BUFFER-QUANTITY) 1)
               (set-value ?ct 'CONSTRAINT-RESULT 'NOT-VIOLATED)
             (set-value ?ct 'CONSTRAINT-RESULT 'VIOLATED))))
```

A.4 Inférence *revise*

Dans cette section, nous détaillons comment nous avons implanté l'inférence ***revise***. Voici le code correspondant à la méthode qui implante l'inférence:

```
(defmethod Revise (?ct)
  :title "Revise the value of parameter because violation of a constraint"
  :situation (:and (Constraint ?ct)
                   (:same-as (Constraint-Result ?ct) 'VIOLATED))
  :response ((progn
              (Save-Values)
              (setq Applicable-Fixes (Get-Fixes ?ct))
              (loop
                (setq Combination (Get-Next-Combination Applicable-Fixes))
                (if Combination
                    (Apply-Combination-Incrementally Combination ?ct)
                  (return))
                (if (not (or (Constraint-Result ?ct 'VIOLATED)
                             (Other-Related-Violations)))
                    (return))
                (Restore-Values))
                (Make-Changed-Values-Permanent)
              )
             ))
```

En résumé, la méthode effectue les actions suivantes:

- Toutes les valeurs de variables sont enregistrées avant d'appliquer les modifications pour essayer de réparer la contrainte violée (`Save-Values`).
- Les réparations associées à la contrainte violée sont récupérées (`Get-Fixes`).
- Après avoir récupéré les réparations applicables, celles-ci doivent être appliquées dans l'ordre suivant: les réparations simples de coût inférieur d'abord; ensuite,

les combinaisons de réparations au coût courant et inférieurs, en passant au prochain niveau de coût seulement quand toutes les combinaisons aux niveaux inférieurs n'ont pas réussi à réparer la contrainte (`Get-Next-Combination`).

- Quelques réparations spécifient qu'une valeur doit être augmentée graduellement selon une dimension, par exemple, les réparations du type `:increment`, `:decrement` or `:upgrade`. Quand une combinaison de réparation générée selon l'ordre décrit contient une ou plusieurs réparations de ce type, toutes les combinaisons possibles pour cette réparation doivent être essayées avant d'appliquer la prochaine combinaison de réparation (`Apply-Combination-Incrementally`).

- Après avoir appliqué la réparation ou la combinaison de réparations, un test est fait pour vérifier si la contrainte originale n'est plus violée, et si la réparation n'a pas violé d'autres contraintes liées aux modifications (des contraintes sur les valeurs modifiées sont violées soit parce qu'elles sont devenues violées comme résultat de l'application de la réparation, soit parce qu'elles étaient déjà violées et la réparation n'a pas éliminé la violation) Dans les deux cas, la réparation doit être rejetée, et les modifications sont défaites (`Restore-Values`) et la prochaine réparation applicable sera essayée. Sinon, la réparation est acceptée, et les modifications sont rendues permanentes (`Make-Changed-Values-Permanent`).

A.5 Trace

Dans cette section, nous présentons la trace de l'exécution de notre implantation de P&R appliquée à la tâche VT. Nous avons utilisé le cas de test présenté en VT-Yost et nous avons obtenu le même résultat final. Voici les contraintes violées et les réparations utilisées pour les satisfaire:

- La contrainte **min platform to hoistway left** est satisfaite par l'application de la réparation **inc opening to hoistway left** une fois.

- La contrainte **legal value motor model** est satisfaite par l'application de la réparation **inc machine model** une fois.
- La contrainte **max traction ratio** est satisfaite par l'application de la combinaison de réparation (**dec cwt to platform rear, inc car supplement weight, inc comp cable model**), 7, 4 et 1 fois, respectivement.
- La contrainte **max machine groove pressure** est satisfaite par l'application de la réparation **inc hoist cable quantity** 1 fois.
- La contrainte **max vertical rail force** est satisfaite par l'application de la réparation **inc car rail unit weight** 1 fois.
- La contrainte **min hoist cable safety factor** est satisfaite par l'application de la réparation **inc hoist cable quantity** 1 fois.
- La contrainte **min machine beam section modulus** est satisfaite par l'application de la réparation **inc machine beam model** 1 fois.
- La contrainte **max traction ratio** est encore violée et elle est satisfaite par l'application de la réparation **inc car supplement weight** 1 fois.

La trace suivante montre l'exécution de P&R pour l'obtention du résultat final. Ici, nous pouvons remarquer que les inférences ***propose*** et ***check*** alternent jusqu'à ce qu'une contrainte soit violée. Dans ce cas, les inférences ***select violated constraint*** et ***revise*** sont exécutées. Après la réparation de la contrainte violée, un nouveau cycle recommence.

```
Method: Propose |i|BRACKET-SPACING
Method: Check |i|MAX-CAB-HEIGHT
Method: Check |i|MIN-CAB-HEIGHT
Method: Check |i|MAX-OPENING-HEIGHT
Method: Check |i|MIN-OPENING-HEIGHT
Method: Check |i|MAX-OPENING-WIDTH
Method: Check |i|MAX-OVERHEAD
Method: Check |i|MAX-PIT-DEPTH
Method: Check |i|MIN-PIT-DEPTH
Method: Check |i|MIN-PLATFORM-WIDTH
```

```
Method: Check |i|MAX-BRACKET-SPACING
Method: Propose |i|MACHINE-BEAM-BEARING-PLATE-THICKNESS
Method: Propose |i|MACHINE-ROOM-FLOOR-TO-UNDERSIDE-MACHINE-BEAM
Method: Propose |i|TOP-LANDING-TO-UNDERSIDE-MACHINE-BEAM
Method: Propose |i|UNDERBEAM-SPACE
Method: Check |i|MIN-UNDERBEAM-SPACE
Method: Propose |i|SLING-UNDERBEAM
Method: Propose |i|CWT-FRAME-HEIGHT
Method: Check |i|MAX-CWT-FRAME-HEIGHT
Method: Check |i|MIN-CWT-FRAME-HEIGHT
Method: Propose |i|CWT-PLATE-THICKNESS
{...}
Method: Propose |i|PLATFORM-TO-HOISTWAY-FRONT
Method: Propose |i|CWT-SPACE
Method: Propose |i|PLATFORM-TO-HOISTWAY-LEFT
Method: Check |i|MIN-PLATFORM-TO-HOISTWAY-LEFT
Method: Select-Violated-Constraint |i|MIN-PLATFORM-TO-HOISTWAY-LEFT
Method: Revise |i|MIN-PLATFORM-TO-HOISTWAY-LEFT

Combination (|i|INC-OPENING-TO-HOISTWAY-LEFT) Times (1)
OPENING-TO-HOISTWAY-LEFT set to 33
Recomputing....

Constraint |i|MIN-PLATFORM-TO-HOISTWAY-LEFT repaired.

Method: Propose |i|CWT-U-BRACKET-PROTRUSION
Method: Propose |i|CWT-TO-PLATFORM-REAR
Method: Propose |i|CWT-TO-HOISTWAY-REAR
Method: Check |i|MIN-CWT-TO-HOISTWAY-REAR
Method: Propose |i|CAR-CABLE-HITCH-OFFSET
Method: Propose |i|MACHINE-BEAM-LENGTH
{...}
Method: Propose |i|MOTOR-MODEL
Method: Check |i|LEGAL-VALUE-MOTOR-MODEL
Method: Select-Violated-Constraint |i|LEGAL-VALUE-MOTOR-MODEL
Method: Revise |i|LEGAL-VALUE-MOTOR-MODEL

Combination (|i|INC-MACHINE-MODEL-5-14-9) Times (1)
MACHINE-MODEL set to 28
Recomputing....

Constraint |i|LEGAL-VALUE-MOTOR-MODEL repaired.

Method: Propose |i|MOTOR-WEIGHT
Method: Propose |i|MACHINE-TOTAL-WEIGHT
{...}
```

```
Method: Propose |i|TRACTION-RATIO
Method: Check |i|MAX-TRACTION-RATIO
Method: Select-Violated-Constraint |i|MAX-TRACTION-RATIO
Method: Revise |i|MAX-TRACTION-RATIO

Combination (|i|DEC-CWT-TO-PLATFORM-REAR-7-2) Times (1)
CWT-TO-PLATFORM-REAR set to 4.75
Recomputing....
{...}
Combination (|i|DEC-CWT-TO-PLATFORM-REAR-7-2 |i|INC-CAR-SUPPLEMENT-WEIGHT
             |i|INC-COMP-CABLE-MODEL) Times (7 4 1)
CWT-TO-PLATFORM-REAR set to 4.75
CWT-TO-PLATFORM-REAR set to 4.25
CWT-TO-PLATFORM-REAR set to 3.75
CWT-TO-PLATFORM-REAR set to 3.25
CWT-TO-PLATFORM-REAR set to 2.75
CWT-TO-PLATFORM-REAR set to 2.25
CWT-TO-PLATFORM-REAR set to 1.75
CAR-SUPPLEMENT-WEIGHT set to 100.0
CAR-SUPPLEMENT-WEIGHT set to 200.0
CAR-SUPPLEMENT-WEIGHT set to 300.0
CAR-SUPPLEMENT-WEIGHT set to 400.0
COMP-CABLE-MODEL set to "3/16-CHAIN"
Recomputing....

Constraint |i|MAX-TRACTION-RATIO repaired.

Method: Select-Violated-Constraint |i|MAX-MACHINE-GROOVE-PRESSURE
Method: Revise |i|MAX-MACHINE-GROOVE-PRESSURE

Combination (|i|INC-HOIST-CABLE-QUANTITY) Times (1)
HOIST-CABLE-QUANTITY set to 4
Recomputing....
{...}
Constraint |i|MAX-MACHINE-GROOVE-PRESSURE repaired.

Method: Select-Violated-Constraint |i|MAX-VERTICAL-RAIL-FORCE
Method: Revise |i|MAX-VERTICAL-RAIL-FORCE

Combination (|i|INC-CAR-RAIL-UNIT-WEIGHT) Times (1)
CAR-RAIL-UNIT-WEIGHT set to 11
Recomputing....
{...}
Constraint |i|MAX-VERTICAL-RAIL-FORCE repaired.

Method: Select-Violated-Constraint |i|MIN-HOIST-CABLE-SAFETY-FACTOR
Method: Revise |i|MIN-HOIST-CABLE-SAFETY-FACTOR
```

```
Combination (|i|INC-HOIST-CABLE-QUANTITY) Times (1)
{...}
HOIST-CABLE-QUANTITY set to 5
Recomputing....
Constraint |i|MIN-HOIST-CABLE-SAFETY-FACTOR repaired.

Method: Select-Violated-Constraint |i|MIN-MACHINE-BEAM-SECTION-MODULUS
Method: Revise |i|MIN-MACHINE-BEAM-SECTION-MODULUS

Combination (|i|INC-MACHINE-BEAM-MODEL) Times (1)
MACHINE-BEAM-MODEL set to "S10x35.0"
Recomputing....
{...}
Constraint |i|MIN-MACHINE-BEAM-SECTION-MODULUS repaired.

Method: Select-Violated-Constraint |i|MAX-TRACTION-RATIO
Method: Revise |i|MAX-TRACTION-RATIO

Combination (|i|DEC-CWT-TO-PLATFORM-REAR-7-2) Times (1)
CWT-TO-PLATFORM-REAR set to 1.25
Recomputing....
{...}
Combination (|i|INC-CAR-SUPPLEMENT-WEIGHT) Times (1)
CAR-SUPPLEMENT-WEIGHT set to 500.0
Recomputing....
{...}

Constraint |i|MAX-TRACTION-RATIO repaired.
```

À la fin, les valeurs des paramètres réparés sont:

```
[3] VT(53): (retrieve ?v (has-value MACHINE-MODEL ?v))
(28)
[3] VT(54): (retrieve ?v (has-value OPENING-TO-HOISTWAY-LEFT ?v))
(33)
[3] VT(55): (retrieve ?v (has-value CAR-RAIL-UNIT-WEIGHT ?v))
(11)
[3] VT(56): (retrieve ?v (has-value HOIST-CABLE-QUANTITY ?v))
(5)
[3] VT(57): (retrieve ?v (has-value CWT-TO-PLATFORM-REAR ?v))
(1.75)
[3] VT(58): (retrieve ?v (has-value COMP-CABLE-MODEL ?v))
("3/16-CHAIN")
[3] VT(59): (retrieve ?v (has-value MACHINE-BEAM-MODEL ?v))
("S10x35.0")
```

Nous avons ainsi présenté dans cet appendice l'exécution dans notre système de la méthode *Proposer et Réviser* appliquée à la tâche VT qui configure un ascenseur.

Appendice B

Exécution en K-Loom de C&D appliquée au domaine médical

Dans cet appendice, nous illustrons l'exécution de la méthode *Couvrir et Différencier* réutilisée pour le domaine médical. L'ontologie de la méthode et les inférences ont été définies dans la section 6.1.2 et la structure de la tâche a été définie dans la section 6.1.3. Les données médicales ont été définies dans la section 6.2.

B.1 Entrée des données

Dans Loom, il faut d'abord charger les fichiers correspondants à K-Loom, à l'ontologie de la méthode, aux inférences à l'ontologie du domaine et aux opérations de correspondance. Le fichier K-Loom contient les macros qui permettent d'exprimer les inférences et les tâches selon les définitions du chapitre 4. Le fichier de l'ontologie de la méthode contient les définitions de concepts et de relations pour utiliser C&D et le fichier des inférences contient la définition des inférences de C&D (section 6.1.2). Le fichier de l'ontologie du domaine contient les définitions du domaine médical (section 6.2) et le fichier d'opérations de correspondance (section 6.3) contient les opérations qui associent les connaissances du domaine médical aux connaissances de la méthode. Ainsi, dans Loom, les fichiers suivants doivent être chargés:

```
(load "k-loom")
(load "ontologie-methode-CD")
(load "inferences-CD")
(load "ontologie-medical")
(load "operations-CD")
```

Ensuite, l'instanciation des réseaux statique et dynamique de C&D avec le domaine médical est réalisée. Pour instancier le réseau statique de C&D, il faut entrer les données qui correspondent aux connaissances médicales constituant le réseau médical de la figure 6.16. L'assertion de ces instances du domaine médical est faite en Loom par les commandes suivantes:

```
(tellm (Patient-State Muscle-Weakeness)
       (Patient-State Sudden-Onset-MW)
       (Patient-State Periodic-MW)
       (Patient-State Hand-Tremors)
       (Patient-State Loss-Weight)
       (Patient-State Incresead-Appetite)
       (Patient-State Neuromuscular-Disease)
       (Patient-State Hypokalemic-Paralysis)
       (Patient-State Hypermetabolic-State)
       (Patient-State Miastenia-Gravis)
       (Patient-State Hypokalemic-Periodic-Paralysis-With-Thyrotoxicosis)
       (Patient-State Hyperthyroidism)
       (Patient-State Diabetes))

(tellm (Qualifier Following-Ingestion-of-Carbohydrates)
       (Qualifier Oriental-Ethnic-Background))

(tellm (Cause 'Possibly Neuromuscular-Disease Sudden-Onset-MW)
       (Cause 'Possibly Hypokalemic-Paralysis Muscle-Weakeness)
       (Cause 'Necessarily Hypokalemic-Paralysis Sudden-Onset-MW)
       (Cause 'Possibly Neuromuscular-Disease Periodic-MW)
       (Cause 'Possibly Hypokalemic-Paralysis Periodic-MW)
       (Cause 'Possibly Hypermetabolic-State Hand-Tremors)
       (Cause 'Possibly Hypermetabolic-State Loss-Weight)
       (Cause 'Possibly Hypermetabolic-State Incresead-Appetite)
       (Cause 'Possibly Miastenia-Gravis Neuromuscular-Disease)
       (Cause 'Possibly Miastenia-Gravis Hypokalemic-Paralysis)
       (Cause 'Possibly Hypokalemic-Periodic-Paralysis-With-Thyrotoxicosis
              Hypokalemic-Paralysis)
       (Cause 'Possibly Hypokalemic-Periodic-Paralysis-With-Thyrotoxicosis
              Hypermetabolic-State)
       (Cause 'Possibly Hyperthyroidism Hypermetabolic-State)
       (Cause 'Possibly Diabetes Hypermetabolic-State))
```

```
(tellm (State-Qualifying 'Positively Oriental-Ethnic-Background
                         Hypokalemic-Periodic-Paralysis-With-Thyrotoxicosis))

(tellm (Connection-Qualifying 'Positively Following-Ingestion-of-Carbohydrates
                              Hypokalemic-Paralysis Muscle-Weakeness))
```

Les assertions qui représentent une situation spécifique de diagnostic correspondent au réseau dynamique de C&D. Ces assertions donnent les symptômes du patient et sont les données d'entrée. C'est à partir de ces données que le diagnostic va pouvoir être établi. Dans notre cas, les symptômes du patient constituent tous les nœuds inférieurs du réseau statique. En plus des symptômes initiaux, il faut aussi fournir les observations qui ne sont pas des symptômes, c'est-à-dire les qualificateurs spécifiques à la situation, appelés **findings**. Les assertions qui permettent d'entrer ces données sont les suivantes:

```
(tellm (Symptom Muscle-Weakeness)
       (Symptom Sudden-Onset-MW)
       (Symptom Periodic-MW)
       (Symptom Hand-Tremors)
       (Symptom Loss-Weight)
       (Symptom Incresead-Appetite))


(tellm (Finding Oriental-Ethnic-Background)
       (Finding Following-Ingestion-of-Carbohydrates))
```

B.2 Méthodes pour exécuter les inférences

Dans notre implantation de C&D, il y a une seule méthode associée à chaque action, contrairement à la méthode P&R, où il y a plusieurs méthodes pour la même action générique (section A.3). Les actions auxquelles correspondent des méthodes pour accomplir les inférences en C&D sont:

```
(defaction Establish-Focus (?new-focus))
(defaction Consider-Explanation (?s2))
(defaction Accept-Explanation (?s1 ?s2))
```

```
(defaction Accept-Explanations (?s1 ?s2 ?s3))
(defaction Reject-Explanation (?s1 ?s2))
```

Voyons maintenant quelques exemples de méthodes de C&D:

- La méthode qui insère dans la base de connaissances le nouveau **focus**.

```
(defmethod Establish-Focus (?new-focus)
  :response ((tellm (Focus ?new-focus))))
```

- La méthode qui génère des explications candidates pour un état:

```
(defmethod Consider-Explanation (?s2)
  :response ((forget (Focus ?s2))
             (do-retrieve (?s1)
               (Cover-R ?s1 ?s2)
               (tell (Considered-Explanation ?s1 ?s2)))
             (tellm)))
```

- La méthode qui accepte une explication pour un état et refuse toutes les autres explications pour ce même état:

```
(defmethod Accept-Explanation (?s1 ?s2)
  :response ((tell (Accepted-Explanation ?s1 ?s2))
             (do-retrieve ?s3 (:and (Considered-Explanation ?s3 ?s2)
                                    (:not (:same-as ?s3 ?s1)))
                          (forget (Considered-Explanation ?s3 ?s2))
                          (tell (Rejected-Explanation ?s3 ?s2))
                          )
             (tellm)))
```

B.3 Trace

Dans cette section, nous allons suivre l'exécution de C&D instanciée pour le domaine médical. La trace suivante montre l'activation des règles d'inférences et leurs réalisations par les méthodes. Quand l'inférence n'est pas activée (parce que sa condition n'est pas satisfaite), seul le nom de l'inférence apparaît et il n'y a donc pas de méthode associée.

Dans C&D, la phase "couvrir" alterne avec la phase "différencier". Dans la phase "couvrir", l'inférence ***establish focus*** déclenche l'exécution de l'inférence ***cover***.

Ensuite, c'est la phase "différencier" qui commence. Dans la phase "différencier", une ou plusieurs inférences peuvent être exécutées dans le but de générer des explications acceptées ou refusées.

À partir des symptômes initiaux fournis par l'utilisateur, la phase "couvrir" exécute les inférences ***establish focus*** et ***cover***. ***Establish focus*** choisit d'abord comme focus (**focus**) les plaintes initiales du patient. Puis, ***cover*** génère des explications candidates pour les symptômes initiaux. Voyons maintenant l'exécution de ces inférences. Les inférences ***establish focus*** et ***cover*** alternent jusqu'à ce que toutes les explications possibles soient générées pour les focus. La figure B.1 montre les explications candidates générées par la phase "couvrir". Les explications candidates (**considered explanations**) sont représentées par les lignes pointillées et les étiquettes sur les arcs indiquent l'inférence responsable par la génération de l'explication [1]

```
[1] C&D(12): (Cover&Differentiate)
Task: COVER-AND-DIFFERENTIATE
Inference: ESTABLISH-FOCUS
Method:Establish-Focus |I|MUSCLE-WEAKENESS
Inference: COVER
Method:Consider-Explanation |I|MUSCLE-WEAKENESS
Inference: ESTABLISH-FOCUS
Method:Establish-Focus |I|SUDDEN-ONSET-MW
Inference: COVER
Method:Consider-Explanation |I|SUDDEN-ONSET-MW
Inference: ESTABLISH-FOCUS
Method:Establish-Focus |I|PERIODIC-MW
Inference: COVER
Method:Consider-Explanation |I|PERIODIC-MW
Inference: ESTABLISH-FOCUS
Method:Establish-Focus |I|HAND-TREMORS
Inference: COVER
Method:Consider-Explanation |I|HAND-TREMORS
Inference: ESTABLISH-FOCUS
```

1. Cette représentation du processus d'inférence en termes d'un graphe est analogue à la représentation du SSM de Clancey [Clancey, 1992]. Ici aussi les inférences peuvent être vues comme des opérateurs qui ajoutent et enlèvent les nœuds et les arcs d'un graphe. Cependant, nos inférences sont mieux définies que les opérateurs de Clancey. À la base de cette représentation, chaque inférence est définie déclarativement, ce qui permet de comprendre intuitivement la méthode (par le graphe) et ensuite, plus en détail, par la définition en K-Loom des inférences.

```
Method:Establish-Focus |I|LOSS-WEIGHT
Inference: COVER
Method:Consider-Explanation |I|LOSS-WEIGHT
Inference: ESTABLISH-FOCUS
Method:Establish-Focus |I|INCRESEAD-APPETITE
Inference: COVER
Method:Consider-Explanation |I|INCRESEAD-APPETITE
```

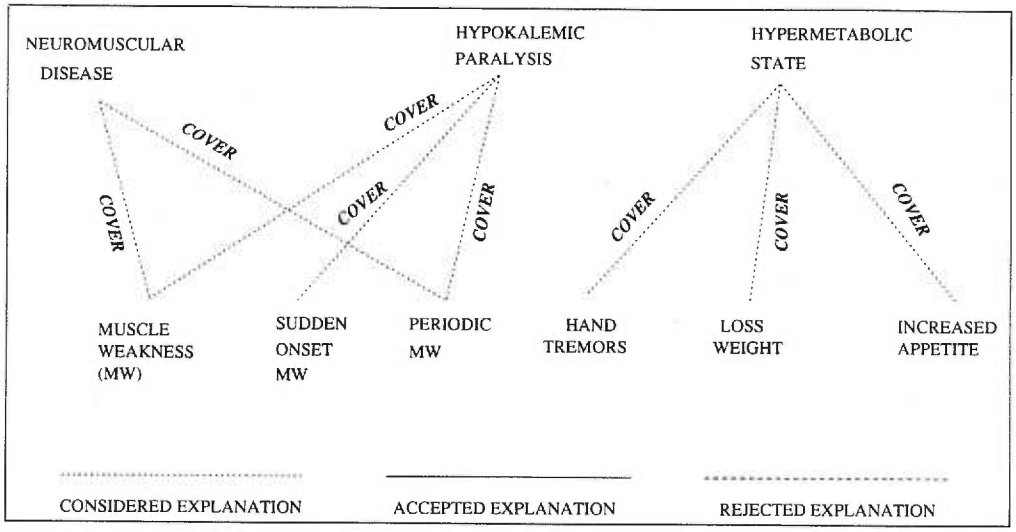

FIG. B.1 - *Explications candidates générées à partir des symptômes initiaux.*

La phase suivante consiste à différencier les explications candidates pour chaque état. L'inférence ***prefer*** est exécutée, en utilisant le qualificateur **following ingestion of carbohydrates**, pour générer l'explication **hypokalemic paralysis** pour **muscle weakeness**. Ensuite, l'inférence ***anticipate-1*** est exécutée, en générant **hypokalemic paralysis** comme explication acceptée pour les états **periodic muscle weakness** et **sudden muscle weakness onset**. Pour les états **increased appetite**, **loss weight** et **hand tremors** il existe une seule explication possible. Ainsi, l'inférence ***exhaustivity*** est exécutée, générant pour ces trois états l'explication acceptée **hypermetabolic state**. Lorsqu'une explication est acceptée pour un état, les autres explications candidates sont rejetées. Ainsi, les inférences qui génèrent les explications acceptées sont les mêmes qui génèrent les explications rejetées.

La trace de l'étape "différencier" et la figure B.2 illustrent les explications accep-

tées, représentées par les lignes solides, et les explications refusées, représentées par les lignes pointillées, ainsi que les inférences associées à chaque arc.

```
Inference: PREFER
Method:Accept-Explanation |I|HYPOKALEMIC-PARALYSIS |I|MUSCLE-WEAKENESS
Inference: RULE-OUT
Inference: ANTICIPATE-1
Method:Accept-Explanations |I|HYPOKALEMIC-PARALYSIS |I|PERIODIC-MW
                           |I|SUDDEN-ONSET-MW
Inference: ANTICIPATE-2
Inference: EXCLUSIVITY
Inference: EXHAUSTIVITY
Method:Accept-Explanation |I|HYPERMETABOLIC-STATE |I|HAND-TREMORS
Inference: PREFER
{...}
Inference: EXHAUSTIVITY
Method:Accept-Explanation |I|HYPERMETABOLIC-STATE |I|LOSS-WEIGHT
Inference: PREFER
{...}
Inference: EXHAUSTIVITY
Method:Accept-Explanation |I|HYPERMETABOLIC-STATE |I|INCRESEAD-APPETITE
```

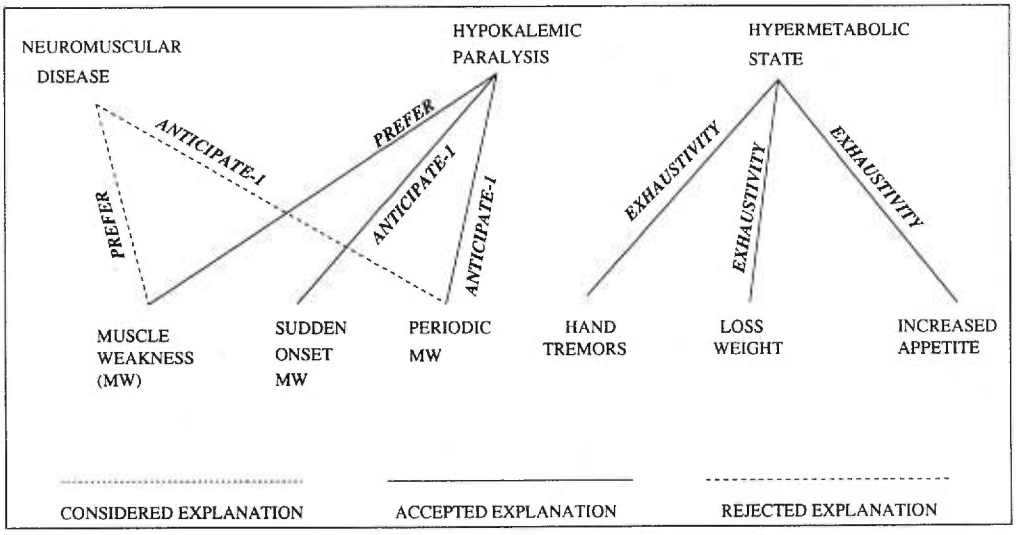

FIG. B.2 - *Explications acceptées et refusées générées par la phase "différencier".*

C&D est un processus itératif. Les états utilisés pour expliquer les nœuds inférieurs doivent, à leur tour, être expliqués par des états au niveau supérieur. Ainsi, la génération des explications acceptées (**accepted explanations**) ira déclencher

encore un nouveau cycle de "couvrir et différencier". Les explications candidates seront différenciées en explications acceptées et refusées. Voyons maintenant l'exécution des inférences et de leurs méthodes respectives. Les états **hypokalemic paralysis** et **hypermetabolic state** doivent être expliqués et différenciés. L'inférence ***prefer*** distingue les explications candidates par la présence du qualificateur **oriental ethnic background**. La figure B.3 montre les nouvelles explications acceptées et refusées générées par ce nouveau cycle de "couvrir et différencier", ainsi que les inférences utilisées dans la phase "différencier".

```
Inference: ESTABLISH-FOCUS
Method:Establish-Focus |I|HYPOKALEMIC-PARALYSIS
Inference: COVER
Method:Consider-Explanation |I|HYPOKALEMIC-PARALYSIS
Inference: ESTABLISH-FOCUS
Method:Establish-Focus |I|HYPERMETABOLIC-STATE
Inference: COVER
Method:Consider-Explanation |I|HYPERMETABOLIC-STATE
Inference: PREFER
Method:Accept-Explanation |I|HYPOKALEMIC-PERIODIC-PARALYSIS-WITH-THYROTOXICOSIS
                          |I |HYPOKALEMIC-PARALYSIS
{...}
Inference: PREFER
Method:Accept-Explanation |I|HYPOKALEMIC-PERIODIC-PARALYSIS-WITH-THYROTOXICOSIS
                          |I|HYPERMETABOLIC-STATE
{...}
Inference: EXHAUSTIVITY
```

Nous arrivons à la fin de l'exécution de C&D, en trouvant comme explication un nœud qui n'a pas d'explications supérieures, ce qui caractérise un diagnostic final. La requête suivante montre les explications acceptées générées par le système:

```
[1] C&D(14): (pprint (retrieve (?s1 ?s2) (Accepted-Explanation ?s1 ?s2)))
((HYPOKALEMIC-PARALYSIS MUSCLE-WEAKENESS)
 (HYPOKALEMIC-PARALYSIS SUDDEN-ONSET-MW)
 (HYPOKALEMIC-PARALYSIS PERIODIC-MW)
 (HYPERMETABOLIC-STATE INCRESEAD-APPETITE)
 (HYPERMETABOLIC-STATE LOSS-WEIGHT)
 (HYPERMETABOLIC-STATE HAND-TREMORS)
 (HYPOKALEMIC-PERIODIC-PARALYSIS-WITH-THYROTOXICOSIS
  HYPERMETABOLIC-STATE)
 (HYPOKALEMIC-PERIODIC-PARALYSIS-WITH-THYROTOXICOSIS
  HYPOKALEMIC-PARALYSIS))
```

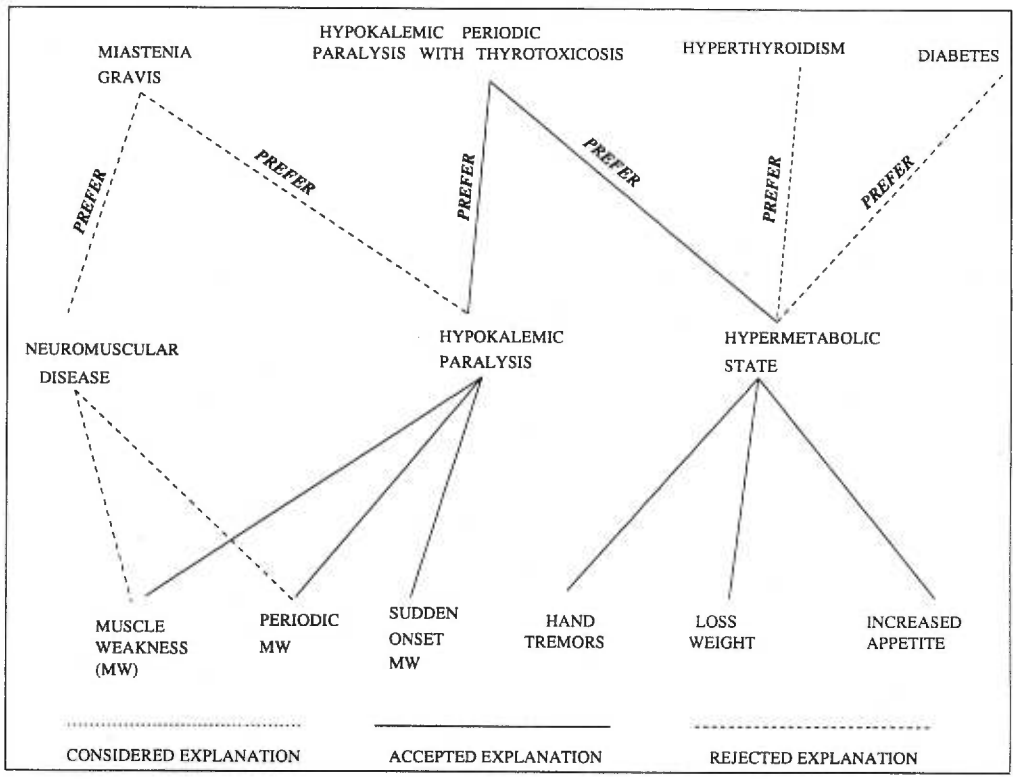

FIG. B.3 - *Explications différenciées générées par le nouveau cycle de C&D.*

Finalement, dans la figure B.4, nous montrons le réseau d'explications acceptées obtenu pour le cas endocrinologique. Nous pouvons noter que les explications convergent vers une seule explication finale.

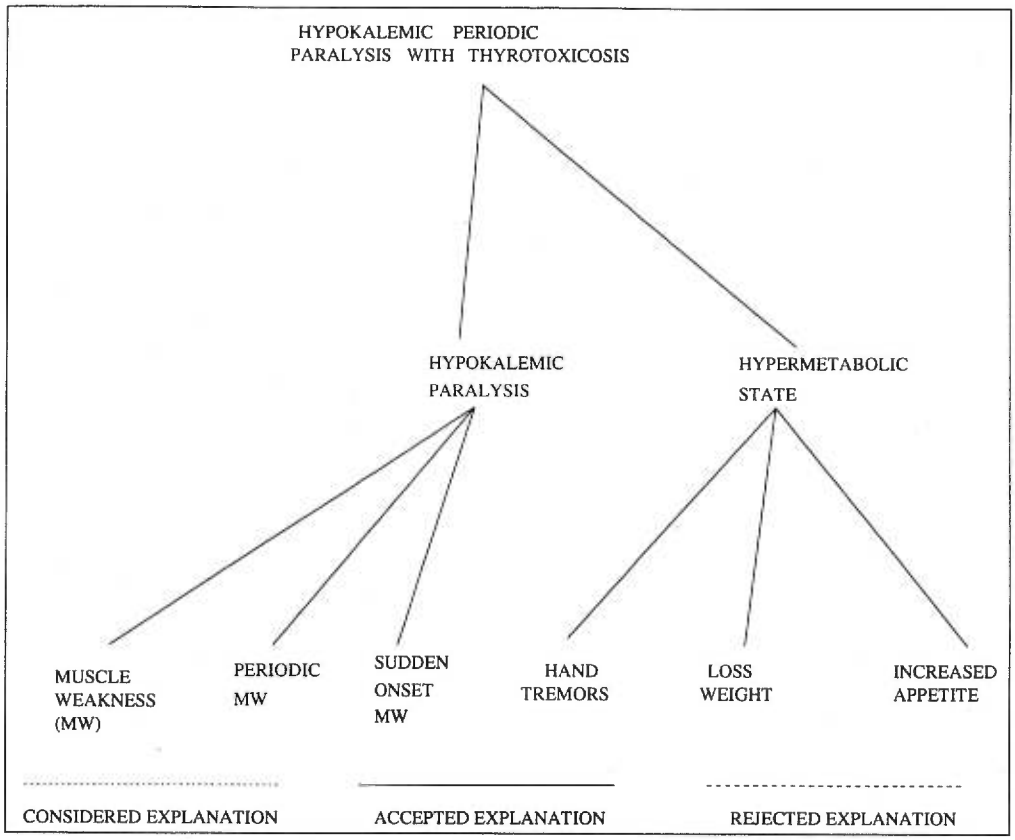

FIG. B.4 - *Réseau d'explications acceptées pour le diagnostic médical.*

Bibliographie

[Aben, 1995] Aben, M. (1995). *Formal Methods in Knowledge Engineering.* Thèse de doctorat, SWI, University of Amsterdam.

[Akkermans et al., 1993] Akkermans, H., Wielinga, B. et Schreiber, G. (1993). Steps in Constructing Problem Solving Methods. Dans Aussenac, N., Boy, G., Gaines, B., Linster, M., Ganascia, J.-G. et Kodratoff, Y., éditeurs, *Knowledge Acquisition for Knowledge-Based Systems*, numéro 723 dans Lecture Notes in AI, pages 45–65. Springer-Verlag.

[Alexander et al., 1986] Alexander, J., Freiling, M., Shulman, S., Staley, J., Rehfuss, S. et Messick, S. (1986). Knowledge Level Engineering: Ontological Analysis. *Proceedings AAAI-86*, pages 963–968.

[Aussenac-Gilles et Matta, 1994] Aussenac-Gilles, N. et Matta, N. (1994). Making a Method of Problem Solving Explicit with MACAO. *International Journal of Human-Computer Studies*, 40(2):193–219.

[Barbuceanu, 1993] Barbuceanu, M. (1993). Models: Toward Integrated Knowledge Modeling Environments. *Knowledge Acquisition*, 5:245–304.

[Benjamins et al., 1996] Benjamins, R., Fensel, D. et Straatman, R. (1996). Assumptions of Problem-Solving Mehtods and their Role in Knowledge Engineering. Dans Wahlster, W., éditeur, *Proceedings of 12th European Conference on Artificial Intelligence, ECAI'96*, pages 408–412. John Wiley & Sons, Ltd.

[Benjamins et Pierret-Golbreich, 1996] Benjamins, R. et Pierret-Golbreich, C. (1996). Assumptions of Problem-Solving Methods. Dans Shadbolt, N., O'Hara, K. et Schreiber, G., éditeurs, *Advances in Knowledge Acquisition, 9th European Knowledge Acquisition Workshop, EKAW'96*, numéro 1076 dans Lecture Notes in Artificial Intelligence, pages 1–16. Springer-Verlag.

[Beys et al., 1996] Beys, P., Benjamins, R. et Van Heijst, G. (1996). Automated Mapping Generation: Remedying the Reusability-Usability Tradeoff. Dans Gaines, B. et Musen, M., éditeurs, *10th Knowledge Acquisition Workshop (KAW'96)*, volume 1.

[Brachman et Schmolze, 1985] Brachman, R. et Schmolze, J. (1985). An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216.

[Breuker, 1994] Breuker, J. (1994). A Suite of Problems. Dans Breuker, J. et Van de Velde, W., éditeurs, *CommonKADS Library of Expertise Models - Reusable Problem Solving Components*, pages 57–87. Springer-Verlag.

[Breuker et Van de Velde, 1994] Breuker, J. et Van de Velde, W. (1994). *CommonKADS Library of Expertise Models - Reusable Problem Solving Components.* Springer-Verlag.

[Brill, 1993] Brill, D. (1993). *Loom Reference Manual, Version 2.0.* University of Southern California.

[Buchanan et Shortliffe, 1984] Buchanan, B. et Shortliffe, E. (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley.

[Bylander et Chandrasekaran, 1987] Bylander, T. et Chandrasekaran, B. (1987). Generic Tasks in Knowledge-Based Reasoning: the Right Level of Abstraction for Knowledge Acquisition. *International Journal Man-Machine Studies*, 1(26):231–243.

[Causse-Gobinet, 1994] Causse-Gobinet, K. (1994). *MCC, vers l'acquisition de connaissances de contrôle.* Thèse de doctorat, Université Paris-Sud.

[Chandrasekaran, 1983] Chandrasekaran, B. (1983). Towards a Taxonomy of Problem Solving Types. *AI Magazine*, 4(1):9–17.

[Chandrasekaran, 1986] Chandrasekaran, B. (1986). Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE EXPERT*, 1:23–30.

[Chandrasekaran et al., 1992] Chandrasekaran, B., Johnson, T. et Simith, J. (1992). Task-Structure Analysis for Knowledge Modeling. *Communications of the ACM*, 35:124–137.

[Clancey, 1985] Clancey, W. (1985). Heuristic Classification. *Artificial Intelligence*, 1(27):289–350.

[Clancey, 1986] Clancey, W. (1986). From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ONR final report 1979-1985. *AI Magazine*, 7(3):40–60.

[Clancey, 1987] Clancey, W. (1987). *Knowledge-Based Tutoring - The GUIDON Program.* MIT Press.

[Clancey, 1992] Clancey, W. (1992). Model Construction Operators. *Artificial Intelligence*, 7(53):1–115.

[Coelho et Lapalme, 1995] Coelho, E. et Lapalme, G. (1995). Extending Ontolingua for Representing Control Knowledge. Dans *IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing.*

[Coelho et Lapalme, 1996] Coelho, E. et Lapalme, G. (1996). Describing Reusable Problem-Solving Methods with a Method Ontology. Dans Gaines, B. et Musen, M., éditeurs, *Proceedings of 10th Banff Knowledge-Acquisition for Knowledge-Based Systems Workshop*, volume 1, pages 3.1–3.20.

[Coelho et al., 1996] Coelho, E., Lapalme, G. et Patel, V. (1996). From KADS Models to Operational Problem-Solving Methods: Perspectives in the Domain View. Dans *6th Workshop on Knowledge Engineering: Methods & Languages.*

[Date, 1986] Date, C. (1986). *An Introduction to Database Systems*, volume I de *The Systems Programming Series*, chapitre Relational Algebra, pages 257–280. Addison-Wesley Publishing Company, édition 4.

[David et al., 1993] David, J.-M., Krivine, J.-P. et Simmons, R. (1993). *Second Generation Expert Systems.* Springer-Verlag.

[Davis et Hamscher, 1988] Davis, R. et Hamscher, W. (1988). Model-Based Reasoning: Troubleshooting. Dans Shrobe, H. E., éditeur, *Exploring AI: Survey Talks from the National Conference on AI.* Morgan Kaufman, San Mateo, CA.

[de Kleer et Williams, 1987] de Kleer, J. et Williams, B. (1987). Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1):97–130.

[Domingue et al., 1993] Domingue, J., Motta, E. et Watt, S. (1993). The Emerging VITAL Workbench. Dans Aussenac, N., Boy, G., Gaines, B., Linster, M., Ganascia, J.-G. et Kodratoff, Y., éditeurs, *Knowledge Acquisition for Knowledge-Based Systems: Proceedings of the 1993 European Knowledge Acquisition Workshop.* Springer-Verlag.

[Eriksson et al., 1995] Eriksson, H., Shahar, Y., Tu, S., Puerta, A. R. et Musen, M. (1995). Task Modeling with Reusable Problem-Solving Methods. *Artificial Intelligence*, 79:293–326.

[Eshelman, 1988] Eshelman, L. (1988). MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. Dans Marcus, S., éditeur, *Automating Knowledge Acquisition for Expert Systems*, chapitre 3, pages 37–79. Boston: Kluwer Academic Publishers.

[Fensel, 1995a] Fensel, D. (1995a). A Case Study: Assumptions and Limitations of a Problem-Solving Method. *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW'95).*

[Fensel, 1995b] Fensel, D. (1995b). *The Knowledge Acquisition and Representation Language KARL.* Kluwer Academic.

[Fensel et Benjamins, 1996] Fensel, D. et Benjamins, R. (1996). Dans Gaines, B. et Musen, M., éditeurs, *Proceedings of 10th Banff Knowledge-Acquisition for Knowledge-Based Systems Workshop*, volume 1.

[Fensel et al., 1996] Fensel, D., Eriksson, H., Musen, M. et Studer, R. (1996). Conceptual and Formal Specifications of Problem-Solving Methods. *International Journal of Expert Systems - Research & Applications*, 9(4).

[Fensel et Straatman, 1996] Fensel, D. et Straatman, R. (1996). Problem-Solving Methods: Making Assumptions for Efficiency Reasons. Dans Shadbolt, N., O'Hara, K. et Schreiber, G., éditeurs, *Advances in Knowledge Acquisition, 9th European Knowledge Acquisition Workshop, EKAW'96*, numéro 1076 dans Lecture Notes in Artificial Intelligence, pages 17–32. Springer-Verlag.

[Fensel et Van Harmelen, 1994] Fensel, D. et Van Harmelen, F. (1994). A Comparison of Languages Which Operationalize and Formalize KADS Models of Expertise. *Knowledge Engineering Review*, 9(2):105–146.

[Gaines, 1996] Gaines, B., éditeur (1996). *International Journal of Human-Computer Studies*, volume 44.

[Genesereth et Fikes, 1992] Genesereth, M. et Fikes, R. (1992). *Knowledge Interchange Format, Version 3.0*, édition Computer Science Department, Stanford University.

[Genesereth et Nilsson, 1987] Genesereth, M. R. et Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann.

[Gennari et al., 1993] Gennari, J., Tu, S. W., Rothenfluh, T. E. et Musen, M. A. (1993). Mapping Domains to Methods in Support of Reuse. Dans *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-Based Workshop.*

[Gruber, 1992] Gruber, T. (1992). Ontolingua: A Mechanism to Support Portable Ontologies. Rapport technique, Technical Report KSL91-66, Stanford University, Knowledge Systems Laboratory.

[Gruber, 1993a] Gruber, T. (1993a). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220.

[Gruber, 1993b] Gruber, T. (1993b). Towards Principles for the Design of Ontologies Used for Knowledge Sharing. Rapport technique, Technical Report KSL93-04, Stanford University, Knowledge Systems Laboratory.

[Gruber et al., 1994] Gruber, T., Runkel, J. T. et Olsen, G. (1994). VT Domain Ontology. ftp://ksl.stanford.edu/pub/knowledge-sharing/ontologies/.

[Guarino et Giaretta, 1995] Guarino, N. et Giaretta, P. (1995). Ontologies and Knowledge bases: Towards Terminological Clarification. Dans Mars, N., éditeur, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25–32. Amsterdam: IOS Press.

[Harmon, 1987] Harmon, P. (1987). *Intelligent Job Aids: How AI Will Change Training in the Next Five Years*, chapitre Artificial Intelligence and Instruction - Applications and Methods, pages 165–190. Kearsley, G.

[Hayes-Roth et al., 1983] Hayes-Roth, F., Waterman, D. et Lenat, D. (1983). *Building Expert Systems.* Addison Wesley.

[Karbach et al., 1990] Karbach, W., Linster, M. et Voss, A. (1990). Models, Methods, Roles and Tasks: Many Labels - One Idea? *Knowledge Acquisition*, 2(4):279–299.

[Klinker et al., 1991] Klinker, G., Bhola, C., Dallemagne, G. Marques, D. et McDermott, J. (1991). Usable and Reusable Programming Constructs. *Knowledge Acquisition*, 3(2):117–135.

[Klinker et al., 1993] Klinker, G., Marques, D. et McDermott, J. (1993). The Active Glossary: Taking Integration Seriously. *Knowledge Acquisition*, 5(2):173–197.

[Le Roux, 1994] Le Roux, B. (1994). *Éléments d'une approche constructive de la modélisation et de la réutilisation des connaissances.* Thèse de doctorat, Université Paris VI.

[Lenat et Guha, 1990] Lenat, D. et Guha, K. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project.* Addison-Wesley Pub. Co.

[Lercher, 1985] Lercher, A. (1985). *Les mots de la philosophie.* Belin.

[Linster, 1992] Linster, M. (1992). Linking *Modeling to Make Sense* and *Modeling to Implement Systems* in an Operational Modeling Environment. Dans Wetter, T., Althoff, K.-D., Boose, J., Gaines, B., Linster, M. et Schmalhofer, F., éditeurs, *Current Developments in Knowledge Acquisition - EKAW'92*, Lecture Notes in Artificial Intelligence, 599, pages 55–74. Springer-Verlag.

[Linster, 1993] Linster, M. (1993). Integrating Conceptual and Operational Modeling: A Case Study. *Knowledge Acquisition*, 5(2):143–171.

[MacGregor, 1988] MacGregor, R. (1988). A Deductive Pattern Matcher. Dans *Proceeding of AAAI-88*, pages 403–408. The National Conference on Artificial Intelligence.

[MacGregor et Burstein, 1991] MacGregor, R. et Burstein, M. (1991). Using a Description Classifier to Enhance Knowledge Representation. *IEEE Expert*, 6(3):41–46.

[Marcus, 1988] Marcus, S. (1988). SALT: A Knowledge-Acquisition Tool for Propose-and-Revise Systems. Dans Marcus, S., éditeur, *Automating Knowledge Acquisition for Expert Systems*, pages 81–123. Boston: Kluwer Academic Publishers.

[McDermott, 1988] McDermott, J. (1988). Preliminary Steps Toward a Taxonomy of Problem-Solving Methods. Dans Marcus, S., éditeur, *Automating Knowledge Acquisition for Expert Systems*, chapitre 8, pages 225–256. Boston: Kluwer Academic Publishers.

[Mizoguchi et al., 1995] Mizoguchi, R., Takaoka, Y., Vanwelkenhuysen, J. et Ikeda, M. (1995). Ontology for Modeling the World from Problem Solving Perspectives. Dans *IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing.*

[Motta et al., 1994] Motta, E., O'Hara, K., Shadbolt, N., Stutt, A. et Zdrahal, Z. (1994). A VITAL Solution to the Sisyphus II Elevator Design Problem. Dans *8th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW'94).*

[Motta et Zdrahal, 1995] Motta, E. et Zdrahal, Z. (1995). The Problem with What: Issues in Method-Independent Task Specifications. Dans *9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW'95).*

[Motta et Zdrahal, 1996] Motta, E. et Zdrahal, Z. (1996). Parametric Design Problem Solving. Dans Gaines, B. et Musen, M., éditeurs, *10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW'96)*, volume 1.

[Musen et al., 1994] Musen, M., Gennari, J., Eriksson, H., Tu, S. et Puerta, A. (1994). PROTÉGÉ-II: Computer Support for Development of Intelligent Systems From Libraries of Components (Tech. Rep. Nr. KSL-94-60). Rapport technique, Stanford University, Knowledge Systems Laboratory.

[Neches et al., 1991] Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T. et Swartout, W. R. (1991). Enabling Technology for Knowledge Sharing. *AI Magazine*, 12(3):16–36.

[Newell, 1982] Newell, A. (1982). The Knowledge Level. *Artificial Intelligence*, 1(18):87–127.

[Patel et al., 1989] Patel, V., Evans, D. A. et Kaufman, D. R. (1989). A Cognitive Framework for Doctor-Patient Interaction. Dans Evans, D. A. et Patel, V., éditeurs, *Cognitive Science in Medicine: Biomedical Modeling*, chapitre 7, pages 253–308. MIT Press: Cambridge, MA.

[Patil, 1988] Patil, R. S. (1988). Artificial Intelligence Techniques for Diagnostic Reasoning in Medicine. Dans Shrobe, H. E., éditeur, *Exploring AI: Survey Talks from the National Conference on AI*. Morgan Kaufman, San Mateo, CA.

[Schreiber et al., 1993a] Schreiber, G., Wielinga, B. et Akkermans, H. (1993a). Using KADS to Analyze Problem-Solving Methods. Dans Schreiber, G., Wielinga, B. et Breuker, J., éditeurs, *KADS - A Principled Approach to Knowledge-Based System Development*, chapitre 18, pages 415–429. Academic Press Inc.

[Schreiber et al., 1994] Schreiber, G., Wielinga, B., Akkermans, H., Vand de Velde, W. et Anjewierden, A. (1994). CML: The CommonKADS Conceptual Modelling Language. Dans Steels, L., Schreiber, G. et Van de Velde, W., éditeurs, *A Future for Knowledge Acquisition*, numéro 867 dans Lecture Notes in Artificial Intelligence (LNAI), pages 2–25. Springer-Verlag, Berlin.

[Schreiber et al., 1993b] Schreiber, G., Wielinga, B. et Breuker, J. (1993b). Introduction and Overview. Dans Schreiber, G., Wielinga, B. et Breuker, J., éditeurs, *KADS - A Principled Approach to Knowledge-Based System Development*, chapitre 1, pages 1–17. Academic Press Inc.

[Serres, 1995] Serres, C. (1995). *Modèles et modules pour les systèmes à base de connaissances: l'approche CERISE.* Thèse de doctorat, Université Paris VI.

[Simmons, 1992] Simmons, R. (1992). The Roles of Associational and Causal Reasoning in Problem Solving. *Artificial Intelligence*, 53(2-3):159–208.

[Simmons et Davis, 1993] Simmons, R. et Davis, R. (1993). The Roles of Knowledge and Representation in Problem Solving. Dans David, J.-M., Krivine, J.-P. et Simmons, R., éditeurs, *Second Generation Expert Systems*, pages 27–45. Springer-Verlag.

[Sowa, 1991] Sowa, J. (1991). *Principles of Semantic Networks.* Morgan Kaufmann Publishers, Inc.

[Steels, 1984] Steels, L. (1984). Second Generation Expert Systems. *International Journal on Future Generation Computer*, 1:213–221.

[Steels, 1990] Steels, L. (1990). Components of Expertise. *AI Magazine*, 11(2):28–49.

[Valente, 1995] Valente, A. (1995). *Legal Knowledge Engineering*, chapitre Ontologies: Theoretical Foundations for Legal Knowledge Engineering. IOS Press.

[Valente et Breuker, 1996] Valente, A. et Breuker, J. (1996). Towards Principled Core Ontologies. Dans Gaines, B. et Musen, M., éditeurs, *10th Knowledge Acquisition Workshop (KAW'96)*, volume 1, pages 33.1–33.19.

[Van de Velde, 1993] Van de Velde, W. (1993). Issues in Knowledge Level Modelling. Dans David, J.-M., Krivine, J.-P. et Simmons, R., éditeurs, *Second Generation Expert Systems*, pages 211–231. Springer-Verlag.

[Van de Velde, 1994] Van de Velde, W. (1994). An Overview of CommonKADS. Dans Breuker, J. et Van de Velde, W., éditeurs, *Common KADS Library of Expertise Models - Reusable Problem Solving Components*, pages 9–29. Springer-Verlag.

[Van Harmelen et Balder, 1993] Van Harmelen, F. et Balder, J. (1993). $(ML)^2$: A Formal Language for KADS Models of Expertise. Dans Schreiber, G., Wielinga, B. et Breuker, J., éditeurs, *KADS - A Principled Approach to Knowledge-Based System Development*, chapitre 8, pages 169–202. Academic Press Inc.

[Van Heijst, 1995] Van Heijst, G. (1995). *The Role of Ontologies in Knowledge Engineering.* Thèse de doctorat, SWI, University of Amsterdam.

[Van Heijst et Anjewierden, 1996] Van Heijst, G. et Anjewierden, A. (1996). Four Propositions Concerning the Specification of Problem-Solving Methods. Dans *9th European Knowledge Acquisition Workshop, EKAW'96.*

[Van Heijst et al., 1997] Van Heijst, G., Schreiber, A. T. et Wielinga, B. J. (1997). Using Explicit Ontologies in KBS Development. *International Journal of Human-Computer Studies*, 46(2/3).

[Van Heijst et al., 1994] Van Heijst, G., Schreiber, G., Lanzola, G. et Stephanelli, M. (1994). Fondations for a Methodology for Medical KBS Development. *Knowledge Acquisition*, 6(4):395–434.

[Van Melle et al., 1984] Van Melle, W., Buchanan, B. et Shortliffe, E. (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, chapitre EMYCIN: A Knowledge Engineer's Tool for Constructing Rule-Based Expert Systems, pages 302–313. Buchanan, B. and Shortliffe, E.H.

[Waterman, 1986] Waterman, D. A. (1986). *A Guide to Expert Systems.* Addisson-Wesley Publishing Company.

[Wenger, 1987] Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems.* Morgan Kaufmann Publisher, Inc.

[Wielinga et al., 1993] Wielinga, B., Van de Velde, W., Schreiber, G. et Akkermans, H. (1993). Towards a Unification of Knowledge Modelling Approaches. Dans David, J.-M., Krivine, J.-P. et Simmons, R., éditeurs, *Second Generation Expert Systems*, pages 299–335. Springer-Verlag.

[Wielinga et Schreiber, 1993] Wielinga, B. J. et Schreiber, A. T. (1993). Reusable and Shareable Knowledge Bases: A European Perspective. Dans *Proceedings of International Conference on Building and Sharing of Very Large-Scaled Knowledge Bases*, pages 103–115.

[Yost, 1994] Yost, G. R. (1994). Configuring Elevator Systems. Rapport technique, Knowledge Systems Laboratory, Stanford University. Edité et modifié par T. E. Rothenfluh en http://camis.stanford.edu/projects/protege/sisyphus-2/s2-0.html.