N^{o} d'ordre :

THÈSE

présentée à

L'UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

Ecole doctorale - Sciences pour l'ingénieur

pour obtenir le titre de

DOCTEUR EN SCIENCES

spécialité :

INFORMATIQUE

par

Philippe MARTIN

Sujet de la thèse :

Exploitation de graphes conceptuels et de documents structurés et hypertextes pour l'acquisition de connaissances et la recherche d'informations

Soutenue le 14 octobre 1996 à Sophia Antipolis devant le jury composé de :

MM.	**Paul FRANCHI-ZANNETTACCI**	Président
	Joost BREUKER **Michel CHEIN** **Francois RECHENMANN**	Rapporteurs
	Rose DIENG	Directeur
	Jean-Gabriel GANASCIA **Patrick ALBERT** **Olivier CORBY**	Examinateurs

Remerciements

Cette thèse s'est déroulée dans le projet ACACIA de l'Institut National de Recherche en Informatique et Automatique (INRIA), unité de Sophia Antipolis. Je voudrais à ce propos exprimer ma gratitude à Mme Rose Dieng, responsable scientifique du projet ACACIA, pour m'avoir permis d'effectuer cette thèse dans ce projet et pour l'avoir encadrée.

Je tiens à vivement remercier tous les membres du jury de cette thèse :

Paul Franchi-Zannettacci, directeur de l'Ecole Doctorale "Sciences Pour l'Ingénieur" de l'Université de Nice, pour m'avoir fait l'honneur de présider mon jury de thèse.

Joost Breuker, professeur à l'Université d'Amsterdam, *Michel Chein*, responsable de l'équipe "Graphes Conceptuels" au LIRMM (Université de Montpellier), et *François Rechenmann*, responsable du projet SHERPA à l'INRIA Rhône-Alpes, pour avoir accepté de rapporter ma thèse et pour leurs remarques sur mon travail.

Patrick Albert, directeur technique d'ILOG, *Jean-Gabriel Ganascia*, professeur à l'Université Pierre et Marie Curie, de faire partie de mon jury de thèse.

Rose Dieng, responsable de l'équipe ACACIA, et *Olivier Corby*, chercheur de l'équipe ACACIA, qui m'ont permis d'améliorer et d'éclaircir le contenu de mes différents travaux et de ce rapport.

Je suis reconnaissant à *Irène Vatton*, chercheur du projet OPÉRA, et à *Ollivier Haemmerlé*, auteur de CoGITo, pour l'aide qu'ils m'ont apportée dans l'exploitation de Thot et de CoGITo.

Je tiens également à remercier *Rose Dieng*, *Olivier Corby* et les autres membres d'ACACIA pour leur amitié et les nombreuses discussions scientifiques ou culturelles que j'ai eues avec eux : *Alain Giboin*, chercheur de l'équipe ACACIA, *Hortense Hammel*, secrétaire d'Acacia, *Laurence Alpay*, *Nada Matta*, *Johan Vanwelkenhusen* et *Roberto Sacile*, post-doctorants à ACACIA, *Krystel Amergé*, *Christophe Cointe*, *Sofiane Labidi*, *Stéphane Lapalut* et *Myriam Ribière*, doctorants à ACACIA.

Enfin, je tiens à exprimer ma gratitude aux services administratifs et de documentation de l'INRIA Sophia-Antipolis, et à mes anciens professeurs d'informatique Marc Gaetano et Marc Laporte.

Je dédicace ce rapport à tous ceux qui ont permis cette thèse ou bien l'ont rendue plus agréable, notamment :
♥ *Christelle D.*, *Krystel A.*, *Yohan B.R.* et ses fans du C.I.V. et *Jean-François P.* ;
♦ les membres de l'équipe ACACIA, ma famille, le personnel du C.I.V. et de Orly Restauration ;
♠ les membres de mon jury de thèse ;
♣ les contribuables de France.

Plan de la thèse

PARTIE II : LE TRAVAIL RÉALISÉ

Chapitre 1 Introduction

1 Sommaire

1.1 Acquisition de connaissances et recherche d'informations

Avant d'aborder nos objectifs généraux puis détaillés, nous présentons rapidement dans cette section des notions relatives à la représentation de connaissances, l'acquisition de connaissances et la recherche d'informations, ainsi que des inter-relations entre ces notions.

1.1.1 Informations et connaissances

Les mots "connaissance" et "information" sont des mots généraux dont nous devons préciser le sens. Nous adoptons la définition donnée par le dictionnaire de l'informatique (Computing Dictionary, 1996) à propos des connaissances[1] :

> **Connaissance(s)** : les ***objets***, ***concepts*** et ***relations*** qui sont supposés exister dans un certain domaine d'intérêt. Un ensemble de connaissances, représentées en utilisant un ***langage de représentation de connaissances***, est connu comme une ***base de connaissances***, et un programme qui étend et/ou effectue des recherches dans une base de connaissances est un ***système à base de connaissances***. Les connaissances diffèrent ***des données ou des informations*** dans le sens où de nouvelles connaissances peuvent être créées à partir de connaissances existantes par ***inférence logique***. Si une *information* est une *donnée* plus une *signification*, alors une *connaissance* est une *information* plus du *raisonnement*. Une forme courante de connaissances, e.g. dans un programme Prolog, est une collection de faits et de règles à propos d'un sujet. Par exemple, une base de connaissances à propos d'une famille peut contenir le fait que John est le fils de David, le fait que Tom est le fils de John, et la règle représentant que le fils du fils d'une personne est son petit-fils. A partir de cette connaissance, le programme peut inférer le fait que Tom est un petit-fils de David.

Ajoutons que pour nous, une connaissance est le résultat de l'interprétation d'une information. Par exemple, une personne peut lire une phrase dans un document technique, l'interpréter, puis représenter, avec un langage de représentation de connaissances, la connaissance qu'il a retirée de la lecture de la phrase et de son interprétation. Un analyseur de langage naturel peut également lire une

1. **Knowledge**: The ***objects***, ***concepts*** and ***relationships*** that are assumed to exist in some area of interest. A collection of knowledge, represented using some ***knowledge representation language*** is known as a ***knowledge base*** and a program for extending and/or querying a knowledge base is a ***knowledge-based system***. Knowledge differs from ***data or information*** in that new knowledge may be created from existing knowledge using ***logical inference***. If *information* is *data* plus *meaning* then *knowledge* is *information* plus *processing*. A common form of knowledge, e.g. in a Prolog program, is a collection of facts and rules about some subject. For example, a knowledge base about a family might contain the facts that John is David's son and Tom is John's son and the rule that the son of someone's son is their grandson. From this knowledge it could infer the new fact that Tom is David's grandson.

phrase dans un document technique, et l'interpréter de manière à générer une représentation du contenu de cette phrase qui soit exploitable par un système à base de connaissances. La représentation d'une information est une *représentation partielle* car 1) une personne, et a fortiori une machine, peut ne pas enregistrer, analyser ou comprendre tous le contenu de cette information (e.g. les détails dans une image ou encore les multiples sens d'une phrase), et surtout 2) il n'est pas possible de représenter tout le contenu d'une information avec un langage formel à moins que l'information soit très simple ou déjà formalisée. Pour représenter des connaissances, une certaine ***modélisation*** doit être effectuée, notamment celles des objets et des types d'objets impliqués dans ces connaissances, et leurs inter-relations. Cette collection de d'objets, de types d'objets et de leurs relations est parfois appelée une ***conceptualisation***. Voici la définition donnée par le dictionnaire de l'informatique (Computing Dictionary, 1996) à propos de cette conceptualisation[1].

> **Conceptualisation** : la collection des ***objets***, de ***concepts*** et des autres ***entités*** qui sont supposés exister dans un certain domaine d'intérêt, et les ***relations*** qui les relient. Une conceptualisation est une ***vue abstraite***, ***simplifiée*** du monde que l'on veut représenter. Par exemple, on peut conceptualiser une famille par un ensemble de noms, de sexes et de relations entre les membres de la famille. Le choix d'une conceptualisation est la *première étape de la* ***représentation de connaissances***. Chaque ***base de connaissances***, ***système à base de connaissances***, ou ***agent modélisé au niveau connaissance*** est, *explicitement* ou *implicitement*, relatif à une certaine conceptualisation.

Une représentation formelle de cette conceptualisation s'appelle une ***ontologie*** : "an ontology is an explicit specification of a conceptualisation" (Gruber, 1993). En pratique, une ontologie est une collection de termes formels généralement organisés hiérarchiquement ou munis de définitions formelles qui spécifient leur relations avec d'autres termes formels et posent des contraintes sur leur utilisation dans la représentation de connaissances.

Par abus de langage et pour alléger la lecture, nous utiliserons le terme de "connaissance" pour désigner une "représentation de connaissance". De même, nous utiliserons le mot "information" pour désigner le "support d'une information" : partie de document, image, son, film, etc.

1.1.2 Représenter des connaissances pour la recherche d'information

Afin d'être accessible et manipulable via un outil informatique, l'information doit être stockée dans des supports électroniques comme des bases de données ou des documents électroniques. Afin de faciliter ou permettre certaines ***recherches*** et ***manipulations*** par un logiciel, cette information doit avoir été *découpée* en divers blocs qui sont ***reliés*** entre eux ou ***indexés***. Ce découpage et la création de ces liens ou de ces index correspond à une ***représentation de certaines connaissances*** contenues dans les données stockées.

Le système de représentation de l'information peut être simple (e.g. références directes entre blocs, indexation par mots-clés) ou plus complexe (e.g. indexation de blocs par des noeuds d'un réseau sémantique). Plus la représentation des blocs est formelle ou explicite, détaillée et structurée, plus un outil logiciel peut exploiter cette représentation pour effectuer des recherches et des manipulations (e.g. copie, destruction, modification) sur ces blocs.

Cependant, plus la représentation des informations doit être détaillée et d'un haut niveau d'abstraction suffisant pour permettre des recherches à un niveau conceptuel, moins la création de des représentations peut être automatisée. L'analyse automatique du contenu sémantique d'un texte (et a fortiori d'un son ou d'une image) pose de nombreux problèmes non encore résolus ; aussi,

---

1. **Conceptualisation**: The collection of ***objects***, ***concepts*** and other ***entities*** that are assumed to exist in some area of interest and the ***relationships*** that hold among them. A conceptualisation is an ***abstract***, ***simplified view*** of the world that we wish to represent. For example, we may conceptualise a family as the set of names, sexes and the relationships of the family members. Choosing a conceptualisation is the *first stage of* ***knowledge representation***. Every ***knowledge base***, ***knowledge-based system***, or ***knowledge-level agent*** is committed to some conceptualisation, *explicitly* or *implicitly.*

n'existe-t-il pas à l'heure actuelle d'analyseur de langage naturel performant en dehors de domaines très limités. Par ailleurs, des représentations formelles et détaillées sont très longues à construire manuellement. C'est pourquoi les systèmes actuels de recherche d'informations n'utilisent généralement pas un langage de représentation très expressif (i.e. permettant de représenter finement la sémantique des informations) surtout si les informations à indexer sont nombreuses et si cette indexation est effectuée automatiquement. C'est typiquement le cas en recherche documentaire où les informations indexées sont généralement des documents entiers (et non par exemple des parties de ces documents) et l'indexation s'effectue souvent avec des mots-clés.

Dans ce rapport, la ***recherche d'informations*** (RI) ne désigne pas seulement la recherche de documents, mais la recherche de n'importe quel *élément de document (ED)*, qu'il soit structuré (e.g. une section, un graphique ou encore un document entier) ou non structuré (e.g. une portion de texte, un symbole, une image). Les documents et donc les EDs peuvent être multi-media. La RI peut se faire par requête ou bien par navigation (butinage ou "browsing" en anglais), c'est-à-dire l'exploration libre d'une structure telle qu'une hiérarchie ou un graphe (e.g. un réseau sémantique). Si un système permet la navigation sur des liens entre informations ou entre leurs indexations, nous appellerons ces liens des liens "hypertextes".

1.1.3 Rechercher des informations pour l'acquisition de connaissances

Un *système à base de connaissances* (SBC)[1] est un logiciel qui exploite une *base de connaissances* (BC) pour réaliser des tâches nécessitant une expertise, par exemple des tâches de résolution de problème ou d'aide à la décision.

L'***acquisition des connaissances*** (AC) consiste à construire un SBC (et donc une BC) ou bien uniquement une BC lorsque le but est uniquement de collecter, structurer et sauvegarder des connaissances d'experts (par exemple pour constituer une mémoire d'entreprise).
L'AC implique plusieurs phases, notamment :

1. ***l'élicitation*** de connaissances auprès d'un ou plusieurs experts et la *retranscription* des résultats dans des documents, ainsi que la ***collection*** de *documents sources d'expertise* déjà existants ;
2. ***la sélection, la modélisation et la représentation*** de certaines informations contenues dans ces documents (la représentation s'appuie sur une modélisation reposant sur une sélection ; par abus de langage, nous ne distinguerons désormais plus la ***modélisation*** de la représentation). La phase de modélisation d'une expertise conduit à la création d'un *modèle d'expertise* dont les composants peuvent être représentés dans des *langages formels, semi-formels ou informels*. La phase de modélisation des connaissances nécessaires à la création d'un SBC conduit à la création d'un *modèle conceptuel* contenant un ou plusieurs modèles d'expertise plus éventuellement d'autres modèles (e.g. modèles de coopération et de communication avec l'utilisateur du SBC) ;
3. ***l'implémentation ou l'opérationalisation*** du modèle conceptuel avec des *langages exécutables*, (cette étape est utile si le but de l'AC est de construire un SBC et non uniquement une BC).

Il est intéressant que la phase de ***modélisation*** conduise à la création d'un modèle conceptuel représenté avec des langages formels (ou semi-formels) plutôt qu'informels, car
a) les connaissances sont alors représentées de manière plus précise, explicite et exploitable par un outil informatique ; l'ensemble des modèles ainsi créés constitue déjà une BC sur laquelle des procédures de contrôle de cohérence et de couverture d'information peuvent être appliquées et des simulations d'exécution éventuellement effectuées ;
b) l'implémentation de la BC du SBC final est plus aisée et peut être en grande partie automatisable. En contrepartie, le travail de modélisation est plus long et difficile.
Nous appelons ***outil d'AC***, un outil qui permet, guide ou facilite la phase de modélisation et éventuellement celle d'opérationalisation des connaissances.

1. Nous listons à la fin de ce rapport, la liste des abréviations utilisées.

La phase de ***modélisation*** a pour but de *collecter*, *abstraire* et *structurer* les informations figurant dans les sources d'expertise (retranscriptions d'interviews, documents techniques, etc.). Les ***cogniticiens*** (ou *ingénieurs de la connaissance*) qui effectuent cette modélisation, ont donc de nombreuses recherches à effectuer au cours de cette phase :

1. ***recherches d'informations*** dans les documents sources d'expertise pour retrouver ou réunir les informations relatives à certains critères de sélection. Le cogniticien peut alors *comparer* ces informations et en *représenter* une partie ;
2. ***recherches de connaissances*** dans la BC correspondant au modèle conceptuel, pour retrouver ou réunir les connaissances relatives à certains critères de sélection. Le cogniticien peut alors *comparer* ces connaissances puis les modifier, les généraliser ou encore les restructurer. Si le langage de représentation le permet, l'outil d'AC peut effectuer ou faciliter ces tâches de recherche, de comparaison, de généralisation ou de restructuration.

Si lors de la représentation (automatique ou par un cogniticien) d'une information d'un document, le lien entre cette information et sa représentation dans la BC est sauvegardé par l'outil d'AC, cet outil peut permettre à ses utilisateurs de retrouver des connaissances et, depuis ces connaissances, aux informations qu'elles représentent. Certaines informations dans les documents étant *indexées* par des connaissances figurant dans la BC, l'utilisateur peut retrouver ces informations *via* des recherches de connaissances dans la BC. L'outil d'AC est alors également un outil de RI.

La RI via les connaissances de la BC, i.e. sur des critères conceptuels, ne peut néanmoins être effectuée que si des éléments de documents ont été représentés dans cette BC, de manière automatique ou manuelle. Aussi lorsque la modélisation des connaissances d'un document débute, à moins que certaines connaissances n'aient été extraites automatiquement, la RI doit faire appel à d'autres voies plus classiques, par exemple a) la recherche de chaînes de caractères ou b) des recherches par requête ou navigation sur la structure du document, i.e. sur les types de ses éléments ou sur les liens hypertextes qui les relient (mais pour cela, le document doit avoir été structuré[1], automatiquement ou manuellement).

Les liens entre les connaissances d'une BC et les éléments de documents qu'elles représentent, peuvent également être exploités dans d'autres buts que la RI, par exemple pour :
- documenter les connaissances de la BC (celle correspondant au modèle conceptuel ou celle du SBC final) et ainsi permettre à tout moment de remonter à leurs sources pour mieux les comprendre et éventuellement les corriger ou les raffiner ;
- générer des parties de documents en réutilisant les éléments de documents représentés par certaines connaissances (cela est particulièrement utile lors de la génération d'explications ou de documentation technique).

1. Dans un document structuré, tel un document SGML (Goldfarb, 990), les types des EDs et des liens hypertextes sont des types prédéfinis dans un modèle structurel. Structurer un document peut être vu comme une forme élémentaire de représentation de connaissances. Cependant, il semble inutile de représenter, en utilisant des types prédéfinis d'EDs et de liens hypertextes, des aspects conceptuels du contenu de certains EDs, si par ailleurs ces aspects sont représentés dans une BC via un véritable langage de représentation de connaissances. Un tel langage permet en effet de décrire et d'organiser plus finement des connaissances (et donc des informations), et de les rechercher. Aussi, si un tel langage est exploité, les types des EDs représentent plutôt les types de formats (e.g. Chapitre, Section, Paragraphe) ou de supports d'information (e.g. Texte, Image, Graphique) utilisés par les EDs.

1.2 Objectifs et approche

1.2.1 Combinaison de gestion d'informations et de gestion de connaissances

Outre la recherche d'informations, l'AC implique la gestion d'informations : construction de documents intermédiaires à partir des documents sources, génération de documents techniques, etc.

Les documents structurés hypertextes constituent actuellement une technique avancée de stockage et d'organisation d'informations : ils permettent de distinguer la structure logique d'un document (son contenu), de sa structure physique (sa présentation), et de retrouver des éléments de documents aussi bien sur leurs types (accès du type base de données) que sur leur structure.
L'éditeur de documents structurés Thot (Quint & Vatton, 1992)[1] est un outil adapté à la gestion d'informations puisque :
1) Il guide à la manière d'un éditeur syntaxique la construction de documents avec des éléments de documents (EDs) typés, structurés et conformes à leurs modèles structurels. Un modèle structurel définit pour un type d'ED, les éléments dont il peut être composé et les attributs qu'il peut porter, ce qui inclut ses liens hypertextes[2] avec d'autres EDs.
2) Un ED peut être partagé par divers autres EDs via le mécanisme "d'inclusion".
3) Thot peut gérer diverses "vues" sur un document et peut afficher de différentes manières un ED : différents modèles de présentation peuvent être associés au modèle structurel d'un ED.
4) Thot permet de rechercher des EDs par navigation sur les liens hypertextes ou structurels[3] qui relient ces EDs, ou bien par requête sur leurs types et/ou leurs attributs.
5) Thot dispose d'une interface fonctionnelle permettant à une application externe a) d'être avertie des opérations (création, modification, etc.) qu'effectue un utilisateur sur des EDs, et b) de consulter et de modifier la structure logique d'un document. Cela permet de créer des *documents actifs* (Quint & Vatton, 1994a, 1994b) et donc d'utiliser les documents Thot et les fonctions d'édition de Thot comme une interface graphique pour une application.

Aussi, compte-tenu de ce que nous noté dans les sections précédentes, un ***premier objectif*** de notre thèse est monter l'intérêt que peut constituer pour l'AC et la RI, la combinaison de Thot, un outil adapté à la gestion d'informations dans des documents, avec un outil permettant d'exploiter un formalisme adapté à la représentation et l'organisation de connaissances. Nous avons choisi pour cela, le formalisme des Graphes Conceptuels (Sowa, 1984) et la plate-forme de gestion de graphes conceptuels CoGITo (Haemmerlé, 1995a, 1995b). Nous discuterons ces choix plus loin. Nous avons nommé l'outil d'AC et de RI résultant de cette combinaison, CGKAT (Martin, 1995a, 1995b) (Martin & Alpay, 1996).

Notre ***approche*** est la suivante :

1. Le langage de définition des modèles structurels exploités par Thot est suffisamment complet pour permettre de définir la syntaxe de certains langages de représentation de connaissances. Deux expériences ont été effectuées en ce sens (Francou, 1993) (Schaar, 1994). Les EDs de Thot peuvent ainsi non seulement contenir des informations mais également des connaissances (plus exactement, des représentations de connaissances). Ces connaissances peuvent ainsi être connectées à d'autres EDs (contenant des informations ou des connaissances) par des liens structurels ou hypertextes. Elles peuvent alors être *recherchées* et *gérées* via les fonctions de Thot. Nous avons donc défini un modèle structurel et un modèle de présentation pour permettre de construire et de gérer dans Thot des EDs contenant des éléments représentés dans le formalisme des Graphes Conceptuels. De plus, nous avons défini dans un autre modèle structurel *différents*

1. Thot est le nouveau nom (depuis novembre 1995) de la version de l'éditeur de documents structurés Grif (Quint & Vatton, 1992) développé par le projet OPERA de l'INRIA.

2. Dans Thot, il existe deux sortes de liens hypertextes : les attributs de type Reference et les EDs de type Reference.

3. Nous appelons liens structurels, les liens de composition entre EDs. Par exemple, un ED (de type) Chapitre contient un ED TitreChapitre *et* une liste d'EDs Section. Un ED Section peut être défini comme contenant un ED TitreSection *et* une liste d'ED Sous-section *ou bien* une liste d'EDs Paragraphe.

types de liens hypertextes permettant différentes sortes d'*indexation* des informations par ces connaissances, et ce par *différents utilisateurs et selon différents points de vue*. La recherche d'informations ou de connaissances peut ainsi se faire par navigation sur les liens entre informations, entre connaissances, et entre informations et connaissances (dans Thot, les liens hypertextes sont bi-directionnels) : les recherches par navigation sur les informations peuvent être *combinées* avec les recherches par navigation sur les connaissances.

2. Grâce à l'interface fonctionnelle de Thot et à celle de CoGITo, lors de la *construction* de connaissances par l'utilisateur dans le formalisme des Graphes Conceptuels, et lors de l'ouverture d'un document contenant de telles connaissances, notre outil CGKAT construit automatiquement une *base de connaissances* dans l'outil CoGITo, c'est-à-dire dans un format permettant d'exploiter le formalisme dans lequel ces connaissances sont représentées.

3. Nous avons défini et implémenté dans CGKAT un *langage de requête* qui
   a) permet de rechercher des connaissances en exploitant le formalisme des Graphes Conceptuels,
   b) permet de rechercher des informations "via les connaissances" en exploitant les liens hypertextes d'indexation qui ont été posés dans les documents entre les informations et les connaissances.
   De plus, la présentation des résultats d'une requête, sous la forme de connaissances ou d'informations, s'effectue par *génération* d'une partie de document structuré, et en utilisant le mécanisme d'inclusion cité plus haut. Une *inclusion d'un ED* est une copie "vivante" de cet ED dans le sens où si l'ED source est modifiée, la copie est automatiquement modifiée par Thot. De plus, un lien hypertexte relie la copie à l'ED source. Ainsi dans CGKAT, une partie d'un document généré en réponse à une requête est une *vue* sur un ensemble d'informations ou de connaissances sélectionnées sur des critères conceptuels : les résultats de la requête sont modifiés si les EDs sources sont modifiés (nous définirons plus précisément cette notion de "vue" dans la section 2.3.2.1 et nous détaillerons les intérêts de son usage en RI et en AC dans la section 2.3.2.2.2).
   Les vues générées par CGKAT étant reliées par des liens hypertextes aux EDs qu'elles incluent, elles peuvent être le point de départ de recherches par navigation. La génération de ces vues permet donc a) de *focaliser* les recherches par navigation, b) de *combiner* les recherches par requête conceptuelle et les recherches par navigation hypertexte.

4. Enfin, nous avons complété notre langage de requête par des *commandes de manipulation de connaissances* dans la base de CoGITo, e.g. la "jointure maximale" sur des graphes conceptuels, et nous avons permis l'accès de ces commandes depuis le "shell", l'interpréteur standard de commandes d'Unix. Ainsi ces commandes peuvent être *combinées* via les instructions de shell (tests, boucles, "pipes", etc.), elles peuvent être combinées à d'autres commandes accessibles depuis le shell, et des scripts shell peuvent être écrits pour manipuler le contenu de la base de connaissances et générer des documents contenant des informations ou des connaissances. Dans un outil d'AC, de tels scripts peuvent être utiles pour faciliter la mise au point ou la constitution de documents techniques. Ils peuvent également faciliter la RI en étant par exemple destinés à répondre à des questions fréquemment posées.

Dans un contexte d'AC, cette approche est originale car les outils d'AC actuels, lorsqu'il ne s'agit pas d'outils hypertextes utilisés pour l'AC (e.g. Concorde (Hofmann & al., 1990) et MacWeb (Nanard & al., 1993a, 1993b), n'intègrent que peu de fonctions de gestion d'informations ou de recherche d'informations "par le contenu" : seuls des liens hypertextes entre certains types d'informations ou de connaissances sont généralement fournis ainsi que des index (e.g. dans la K-Station (Albert & Vogel, 1989) (ILOG, 1993a) et dans Kads-Tool (Albert & Jacques, 1993) (ILOG 1993b)).

Encore peu de systèmes hypertextes permettent comme Concorde et MacWeb de *représenter des connaissances* et donc d'être utilisés en AC ou dans la recherche d'informations via les connaissances. L'originalité de CGKAT par rapport à ces systèmes est de permettre l'exploitation 1) d'un langage de représentation de connaissances formel, assez général et néanmoins d'une utilisation relativement intuitive, et 2) d'un éditeur de documents structuré. Par contre, CGKAT n'intègre pas d'analyseur de langage naturel comme c'est le cas dans MacWeb. Cependant comme MacWeb,

CGKAT permet de générer des documents virtuels en réponse à des requêtes conceptuelles, c'est-à-dire des vues sur des informations ou des connaissances.

1.2.1.1 Choix du langage de représentation des connaissances

Afin de faciliter la représentation de connaissances à partir de documents, et donc la RI et l'AC, il est souhaitable qu'un langage de représentation de connaissances soit :
1) d'une grande puissance descriptive pour permettre de représenter des phrases, des images, des schémas (cela permet l'indexation d'EDs et dans un cadre d'AC, le passage progressif des documents sources d'expertise aux modèles d'expertise finaux) ;
2) facilement compréhensible (cela facilite le travail du cogniticien et la validation par les experts des connaissances) ;
3) formel pour permettre des contrôles de cohérence et guider le cogniticien vers une plus grande précision dans la représentation.
Il est enfin souhaitable qu'il permette des recherches à un niveau conceptuel sur les connaissances.

Nous avons choisi le formalisme des Graphes Conceptuels, car :
1) comme le souligne Chein (1994), ce formalisme partage les avantages des modèles de type réseau sémantique (i.e. les modèles utilisant des graphes étiquetés) qui sont d'un usage relativement intuitif (notamment grâce à leur aspect graphique) et qui ont de bonnes capacités descriptives (les étiquettes peuvent être complexes et des graphes peuvent être emboîtés dans d'autres graphes) ;
2) les "Graphes Conceptuels simples" (GCS) (cf. section 4.2.2) ont une interprétation consistante et complète en logique du premier ordre ;
3) une relation de spécialisation peut être calculée sur les GCSs. Les spécialisations d'un GCS "requête" peuvent ainsi être retrouvées dans la BC. La recherche de connaissances peut ainsi se faire de manière abstraite et "par le contenu" (i.e. en donnant une partie de la connaissance au lieu d'une référence exacte) comme le préconisent Acker & Porter (1994) pour faciliter la recherche de connaissances et la production d'explications.

Nous avons choisi CoGITo (Haemmerlé, 1995a, 1995b) car cet outil était une plate-forme intéressante de développement de base de graphes conceptuels notamment de par son interface fonctionnelle détaillée. D'autre part, ses sources nous étaient accessibles[1].

1.2.2 Une ontologie pour guider l'AC et la RI

Afin de guider les cogniticiens dans la modélisation d'expertise, les méthodologies et outils d'AC actuels proposent :

1. de collecter différents sortes de connaissances généralement relatives à des distinctions épistémologiques, par exemple :
   a) dans CommonKADS (Breuker & van de Velde, 1994), les connaissances relatives aux tâches, aux inférences de ces tâches, aux méthodes utilisées par ces inférences, et aux connaissances du domaines utilisées par ces méthodes,
   b) dans KOD (Vogel, 1988), les "actons" (ou processus), les "taxons" (entités nécessaires aux actons ou produites par les actons), les "schémas" (sorte de règles permettant notamment de déclencher, de terminer ou de restreindre l'usage de processus).
2. d'utiliser pour collecter et représenter de telles connaissances :
   a) certaines méthodes et tâches génériques de résolution de problèmes (ces tâches génériques étant à adapter et combiner), par exemple des tâches génériques de diagnostic et de planification (CommonKADS propose ainsi une bibliothèque de "modèles de tâches génériques") ;
   b) certains types de concepts ou de relations, par exemple KOD propose d'utiliser des relations de sous-parties et d'attributs sur les taxons et les relations "émetteur", "récepteur" et "ressources", "a_pour_restriction", "déclenche_action", "a_palliatif" sur les actons.

1. Dans le cadre du projet GRAFIA (GC PRC-GDR IA, 1994) du PRC-GDR IA, nous participons à la collaboration entre l'équipe ACACIA et l'équipe du LIRMM au sein de laquelle CoGITo a été développé.

Ces diverses propositions (distinctions épistémologiques, types de concepts, de relation, de méthode et de tâche générique) peuvent être vues comme un ensemble de distinctions conceptuelles utilisables pour interpréter, collecter, classer et représenter les informations des documents. Nous pensons que :

1. Il est possible de réunir et d'interclasser ces distinctions dans une unique ontologie composée d'une taxinomie de types de concepts et d'une taxinomie de types de relations[1]. Dans cette ontologie, nous supposons que des définitions et des schémas prototypiques sont associés aux types afin de représenter leurs inter-relations ou leurs contraintes d'utilisation. Comme nous le montrerons, des schémas prototypiques peuvent être utilisés pour représenter des "modèles de tâches génériques". Une ontologie peut ainsi stocker et indexer des modèles.
2. Il est possible de les compléter ou de les spécialiser en intégrant à cette ontologie, les distinctions établies par les spécialistes en représentation de connaissances ou en linguistique : nous pensons ici aux distinctions conceptuelles de haut niveau mais également aux milliers de types de concepts du langage naturel.
3. Une telle ontologie peut guider et accélèrer la recherche et la représentation des connaissances si elle peut être exploitée par un cogniticien via un outil de recherche et de visualisation adéquat. En effet :
   a) Le cogniticien *dispose* alors, sous une forme structurée, des distinctions conceptuelles issues de différents travaux en modélisation des connaissances, ainsi que des descriptions et contraintes qui sont associées à ces distinctions. Il peut ainsi appliquer différentes méthodes s'il le souhaite.
   b) Ces distinctions se *complètent* et leur interclassement permet leur comparaison automatique ou par le cogniticien (ce qui guide sa recherche et son *choix* des types à utiliser pour représenter des connaissances). De plus, les contraintes ou descriptions associées à un type sont *héritées* par ces sous-types (un système d'AC peut utiliser ces contraintes pour effectuer des contrôles sur les représentations).
   c) Le cogniticien peut construire l'ontologie de son application par extension de cette ontologie générale, c'est-à-dire en spécialisant certains types de cette ontologie, notamment les types de concepts du langage naturel. Les types de l'application sont ainsi des sous-types directs ou indirects de types de l'ontologie générale : cette ontologie générale interclasse les types de l'application et définit certaines de leurs propriétés. L'ontologie de l'application est ainsi *a priori* plus précise, extensible et réutilisable que si elle n'était pas une extension d'une ontologie générale.
   d) Si les types de l'application ont de *nombreux* supertypes, ces liens de spécialisation pourront être exploités par un mécanisme de requête pour permettre à l'utilisateur de retrouver ces types de l'application et les connaissances les utilisant, et ce à de *nombreux* niveaux d'abstraction (autant que de supertypes). Cela est particulièrement intéressant si les supertypes sont des types de concepts du langage naturel et que ceux-ci peuvent être retrouvés en utilisant des termes du langage naturel. Notons enfin que si la recherche de connaissances est facilitée, dans un système tel que CGKAT, la recherche d'informations l'est aussi. Toute distinction conceptuelle peut être utilisée comme un critère conceptuel de sélection de connaissances.

---

1. Dans les taxinomies auxquelles nous nous référons, les définitions de types sont soit des conditions nécessaires et/ou suffisantes d'appartenance à ce type, soit des schémas prototypiques. Ces définitions ne sont que des *descriptions de caractéristiques des objets de ces types*. Aussi, même si cela est difficile, il semble toujours possible d'interclasser dans une même taxinomie, des notions provenant de différentes personnes, méthodologies ou ontologies. Prenons par exemple deux types provenant deux ontologies différentes, soit ils dénotent la même notion et doivent être fusionnés sous un même nom dans la taxinomie, soit ils dénotent à des notions différentes et doivent être représentés sous deux noms différents dans la taxinomie et interclassés. Prenons un exemple plus concret, si les notions de "loisir" pour deux personnes différentes devaient être représentées et interclassées, il est vraisemblable qu'une des deux personnes soit en désaccord avec l'autre sur certains types d'activités que cette autre personne a classés comme "activité de loisir". Dans ce cas, les notions de "loisir" ou d'"activité de loisir" pour ces deux personnes sont différentes et doivent être représentées avec deux types différents afin de pouvoir être interclassées (l'un des types peut être supertype de l'autre ou bien ils ont un supertype en commun).

Aussi, un ***second objectif*** de cette thèse a été :

1. de *réunir et d'interclasser* dans une taxinomie de types de concepts et une taxinomie de types de relations, des distinctions conceptuelles provenant de différents travaux :
   - en représentation des connaissances, notamment Sowa (1992), Sowa (1995b), Tepfenhart (1992), Bateman (1990), Skuce (1995), Myaeng & Koo (1994) et Vilnat (1992),
   - en acquisition des connaissances, notamment KOD et CommonKADS (des modèles de tâches génériques ont été stockées sous forme de définitions de types),
   - en linguistique, notamment Mann & Thompson (1988), Schuler & Smith (1990) et WordNet (Miller & al., 1990) ; WordNet est une base générale de connaissances terminologique organisant 90.000 catégories conceptuelles et les reliant à 120.000 racines de mots anglais ;
2. de *compléter* cette ontologie par d'autres distinctions conceptuelles que nous avons estimées importantes pour la représentation de connaissances ;
3. de faire en sorte que CGKAT :
   - permette à ses utilisateurs de *compléter ou modifier* cette ontologie pour créer l'ontologie de leur application (celle-ci peut également être construite directement) ;
   - permette d'effectuer des *recherches par navigation ou requête* dans cette grande ontologie (un système "d'inclusion dynamique" permet d'exploiter la base WordNet) ;
   - offre pour ces recherches un système de *filtrage* de l'ontologie afin de permettre sa visualisation sous différents *points de vue* (expert, cogniticien, domaine, etc.) ;
   - exploite cette ontologie ou celle de l'application, et notamment les contraintes associées à ses types pour effectuer différents *contrôles* sur la représentation des connaissances (cette fonctionnalité est permise par le formalisme des Graphes Conceptuels qui est basé sur l'exploitation d'une ontologie, et un certain nombre de contrôles sont réalisés par CoGITo).

Cette ***approche*** se distingue de celles adoptées par les autres outils d'AC ou de RI sur deux points : 1) nous proposons une ontologie générale et 2) la représentation des connaissances par l'utilisateur de CGKAT implique la construction d'une ontologie. En effet, le formalisme des GCs n'inclut que peu de primitives épistémologiques mais il permet d'exploiter un treillis de types de concepts, une liste ou une hiérarchie de types de relations, et une liste de marqueurs conformes à des types de concepts.

1. Certains systèmes hypertextes permettent d'utiliser certains types de concepts ou de relations, notamment des types de relations d'argumentation pour faciliter la construction coopérative d'un réseau hypertexte (c'est le cas dans gIBIS (Conklin & Begeman, 1988), SEPIA (Streitz & al, 1992) et AAA (Schuler & Smith, 1990)), mais ces systèmes ne permettent généralement pas à l'utilisateur d'utiliser d'autres sortes de relations (ils n'exploitent pas une ontologie modifiable par l'utilisateur).
2. Dans les outils d'AC, une ontologie peut généralement être construite par l'utilisateur pour organiser les termes formels servant à construire des "connaissances du domaine".
   Cependant, pour représenter les tâches, les inférences et les méthodes, un formalisme dédié à cette représentation est généralement proposé, e.g. CommonKADS CML (Schreiber & al., 1992), et ce formalisme inclut en dur un certain nombre de distinctions conceptuelles. L'utilisateur ne peut donc adapter ou spécialiser ces distinctions conceptuelles.
   Les outils d'AC actuels offrent rarement une ontologie du domaine (et aucun n'est basé sur une base générale de connaissances terminologique) mais offrent plus souvent des types de tâches génériques accompagnés de modèles de tâches génériques.

1.2.3 Plan du rapport

En résumé, pour faciliter l'AC et la RI, notre approche consiste à :
1) combiner un outil adapté à la gestion d'information (Thot) avec un outil (CoGITo) basé sur un formalisme général de représentation de connaissances (le formalisme des Graphes Conceptuels) ;
2) faciliter les recherches dans une grande ontologie et de fournir une ontologie générale synthétisant différents travaux en acquisition, représentation des connaissances et linguistique.

La figure ci-dessous illustre l'architecture de CGKAT et montre la combinaison de Thot, de CoGITo et WordNet.

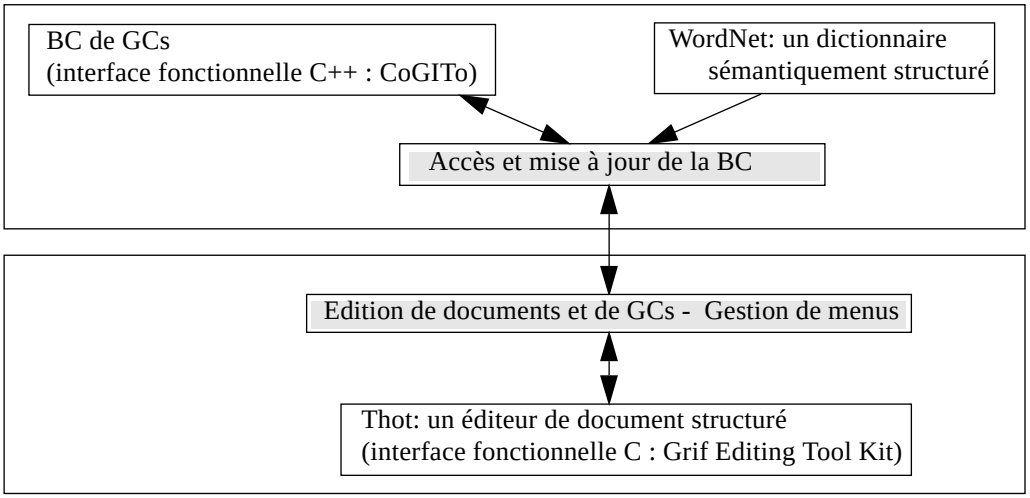


Figure 1.1: *Architecture de CGKAT* (les parties grisées représentent notre code).

Dans la première partie de ce rapport (chapitres 2, 3 et 4) nous effectuons un état de l'art sur l'acquisition de connaissances, sur la recherche et la gestion d'informations, et sur la représentation de connaissances via le formalisme des Graphes Conceptuels.

Dans la seconde partie (chapitres 5, 6 et 7) :

1. Dans le chapitre 5, nous présentons les techniques de visualisation, de recherche et de filtrage d'ontologie qui sont utilisées dans CGKAT, notre technique d'inclusion dynamique des types de WordNet dans l'ontologie de l'application, puis nous détaillons l'organisation des types de haut niveau de l'ontologie générale que nous proposons.
2. Dans le chapitre 6, nous précisons notre approche pour permettre à Thot d'être utilisé pour construire une base de graphes conceptuels. Puis nous montrons comment l'indexation des informations par les connaissances peut être réalisée, et comment notre langage de requête permet de rechercher des connaissances ainsi que les informations qu'elles indexent. Nous montrons l'intérêt de cette approche et la comparons à celles d'autres systèmes d'AC ou de RI. Enfin, nous présentons les aspects méthodologiques d'utilisation de CGKAT.
3. Dans le chapitre 7, nous précisons comment Thot, CoGITo et WordNet ont été exploités au niveau implémentation, et nous détaillons certaines fonctions de CGKAT.

Les annexes précisent un certain nombre de points.
En conclusion, nous présentons un bilan de notre travail et nous comparons les avantages et les inconvénients de notre outil par rapport aux autres systèmes d'AC et de RI. Enfin, nous évoquons des perspectives.

PARTIE I

ETAT DE L'ART

Chapitre 2 L'acquisition de connaissances

2 Sommaire

2.1 Introduction

L'***acquisition des connaissances*** (AC) a pour but soit de construire un ***système à base de connaissances*** (SBC) qui exécutera des ***tâches*** nécessitant une ***expertise*** (e.g. un système expert ou un système d'enseignement assisté par ordinateur), soit simplement de ***collecter et représenter*** de manière ***explicite*** et ***structurée*** une expertise (d'où son utilité par exemple pour créer une "mémoire d'entreprise").

Le premier cas inclut le second, car pour les méthodologies actuelles d'acquisition des connaissances, la création d'un SBC implique tout d'abord de représenter à un *niveau conceptuel* (ou "knowledge level" (Newell, 1982)) l'expertise nécessaire au SBC, c'est-à-dire de créer un ***modèle conceptuel*** de cette expertise. Ce modèle, formel ou semi-formel, doit se situer à un haut niveau d'abstraction et doit donc être indépendant des contraintes et techniques d'*implémentation*. Des choix techniques sont donc à effectuer pour son ***opérationalisation*** : techniques de représentation et de contrôles, e.g. réseau causal, règles, frames, blackboard, chaînage arrière, etc. Si le modèle est suffisamment formel, son opérationalisation peut être semi-automatique. Certains outils de développement de SBC sont dédiés à des tâches précises ou à des domaines très restreints. Dans ces cas, les choix d'implémentation peuvent être codés dans l'outil et la génération d'un système *exécutable* peut-être automatique.

Les premières approches de création de système expert n'incluaient pas ou peu d'étapes de modélisation mais procédaient par ***prototypage rapide*** (Hayes-Roth & al. (1983), Harmon &

King (1985)) : après quelques séances de recueil d'expertise, une maquette était créée et testée, puis soit raffinée grâce à d'autres séances de recueil, soit ré-implémentée en utilisant de nouvelles techniques de représentation (règles, frames, blackboard, etc.). C'est pourquoi de telles méthodes d'acquisition sont dites ***dirigées par l'implémentation***.

De nombreuses méthodes ***basées sur les modèles*** ont ensuite été proposées. Elles voient l'AC comme la construction d'un ou de plusieurs modèles, et proposent des squelettes de modèles, ou ***modèles génériques***[1], afin de guider le cogniticien dans la *sélection* d'informations pertinentes parmi les données de l'expertise et la *construction* du ou des modèles (voir figure 2.1). Par exemple dans KADS (Wielinga & al, 1992), les modèles à construire sont : le *modèle de l'organisation* (l'environnement social ou technique du futur SBC), le *modèle d'application* (fonctions du SBC, contraintes à respecter), le *modèle de tâches* (décomposition des tâches à exécuter et répartition de ces tâches entre le système et l'utilisateur), le ***modèle conceptuel*** (divisé en un *modèle d'expertise de résolution de problème*, un *modèle d'agent* et un *modèle de communication*) et le *modèle de conception* (choix des techniques pour opérationaliser le modèle conceptuel). A l'origine, pour KADS, les choix techniques et l'opérationalisation ne devaient pas engendrer de retour en arrière sur la phase de modélisation (Breuker & al., 1987). Dans la pratique, cela s'est avéré très difficile à respecter : les choix techniques imposent parfois de compléter la modélisation par de nouvelles informations, et la validation du modèle conceptuel impose d'effectuer des simulations et donc d'effectuer un certain prototypage par opérationalisation rapide ou partielle du modèle conceptuel. D'où le développement actuel des langages de représentation formels (et parfois partiellement exécutables), e.g. ML^2 (Harmelen & Balder, 1992) et celui de (Jonker & Spee, 1992).

Durant l'***implémentation*** (ou opérationalisation) du modèle conceptuel, il est utile de *conserver la structuration des connaissances modélisées*. Le système peut ainsi raisonner à divers niveaux d'abstraction et fournir des informations à différents niveaux d'abstraction sur ses connaissances et ses raisonnements (e.g. pour répondre à des questions du type "quoi", "comment", "pourquoi", "pourquoi pas", etc.). Cela est intéressant pour la génération d'explications ou de documentation, la réutilisation du code, la maintenance, et pour permettre la coopération du SBC avec d'autres systèmes ou, à l'intérieur du SBC, pour permettre la coopération d'un résolveur de problème avec d'autres résolveurs.

Inversement, et notamment pour la génération d'explications, de nombreuses informations doivent être *recueillies* (auprès d'un expert ou dans des documents) et représentées de manière *explicite* dans le modèle conceptuel même si elles ne sont pas directement utilisables pour la résolution de problèmes, e.g. des connaissances sur la nature, le fondement et l'utilité d'autres connaissances (du domaine ou de raisonnement), et sur les manières de communiquer ces diverses connaissances à différents utilisateurs. Par ailleurs, le recueil et la modélisation simultanées de toutes ces connaissances permet au cogniticien de mieux comprendre l'expertise, lui fournit plus de contraintes pour sa modélisation, et permet aux lecteurs de cette modélisation de mieux la comprendre.

L'idée directrice de l'activité de modélisation nous semble donc être ***expliciter*** et ***organiser*** : recueillir, expliciter et organiser les connaissances de résolution de problèmes, d'explication et de coopération avec l'utilisateur.

1. Pour guider la construction du modèle conceptuel, un modèle générique contient généralement la description d'une méthode pour réaliser une tâche générique, comme le diagnostic, la prédiction, la planification ou la conception. Les types de connaissances du domaine utilisés par ces méthodes sont également décrites.

2.2 Les grands types d'activités et de connaissances

Les méthodologies et outils actuels d'AC sont "basés sur les modèles" et une théorie générale émerge maintenant sur les différentes activités à réaliser au cours de l'AC (figures 2.1 et 2.2), et sur les composants du modèle conceptuel ou des modèles génériques qui guident sa construction (figure 2.3) (Heijst & al., 1996). Toutes ces activités ne sont pas guidées ou supportées par toutes les méthodologies et outils. Un cas extrême est la "mise à jour" de la base de modèles génériques (ajout ou modification) suite à la création du modèle conceptuel, qui selon (Heijst & al., 1996) est une activité supportée par aucun outil actuel. Cependant, toutes ces activités font partie de l'AC, implicitement ou explicitement, et de manière plus ou moins importante. Elles permettent donc de fournir un cadre pour comparer les outils et les méthodologies (Heijst & al., 1996).

Avant de détailler ces activités du cogniticien, nous allons détailler les grands types de connaissances utilisés en modélisation.

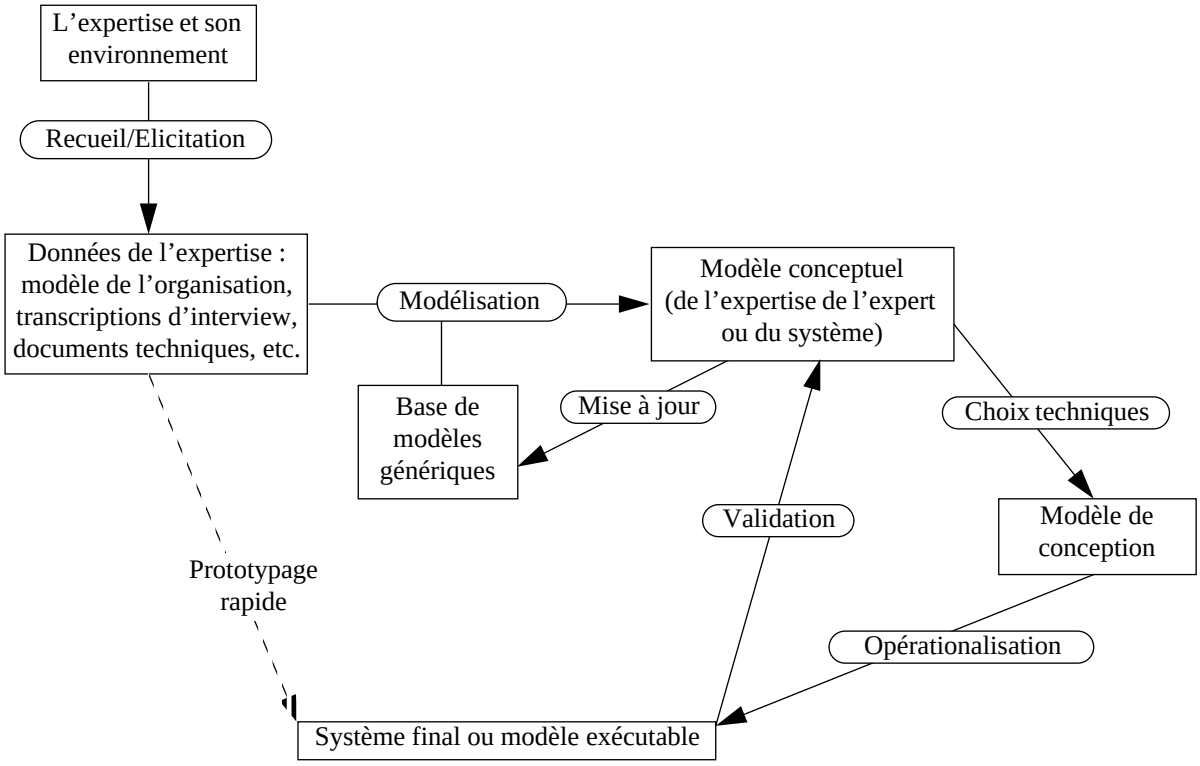


Figure 2.1: *Les principales activités de l'acquisition des connaissances (adapté de (Heijst & al., 1996)). L'approche par "modélisation" est maintenant préférée à l'approche par "prototypage rapide".*

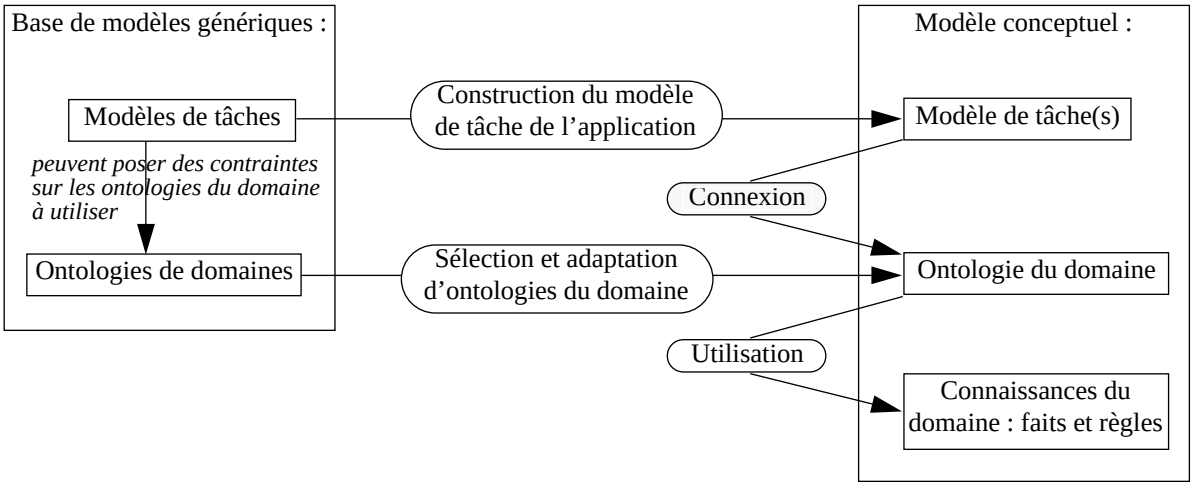


Figure 2.2: *Les principales activités de la modélisation des connaissances (adapté de (Heijst & al., 1996)) (toutes ces activités prennent bien sûr aussi en entrée les données de l'expertise).*

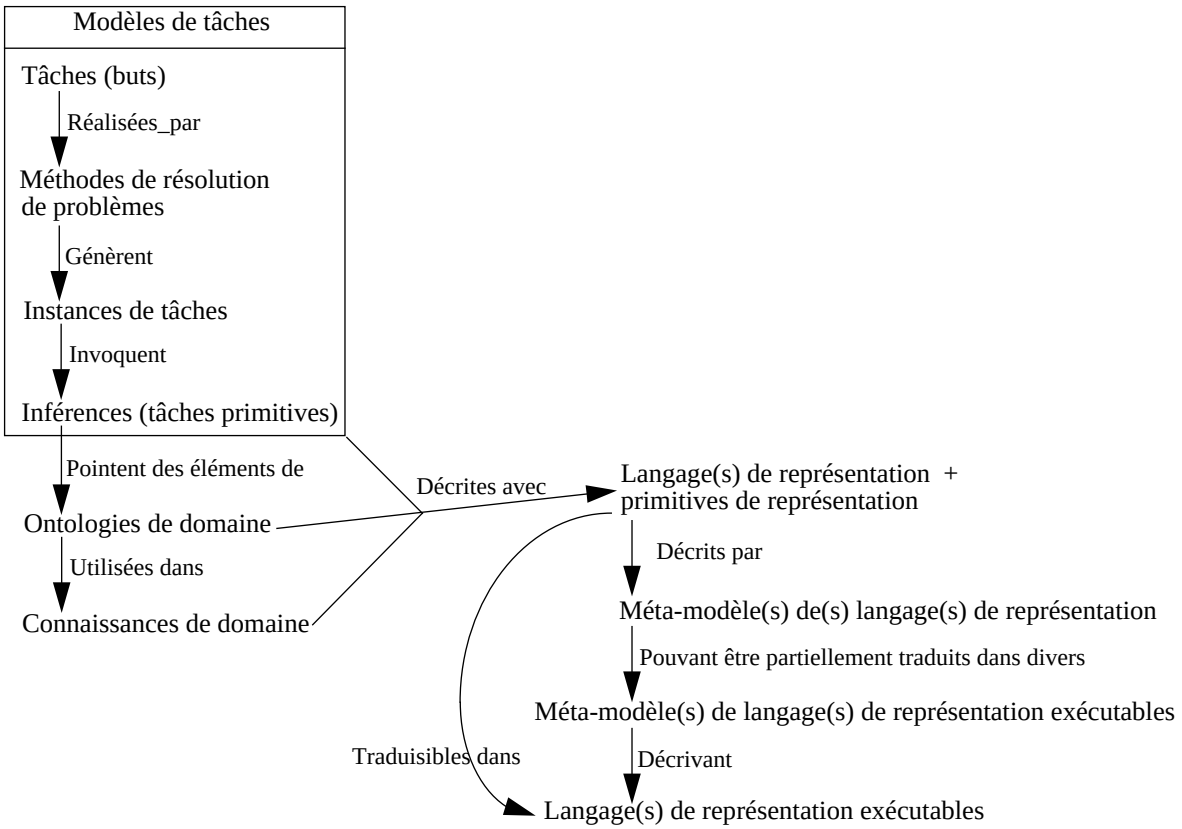


Figure 2.3: *Les liens entre les grands types d'éléments utilisables pour contruire un modèle conceptuel et l'opérationaliser (adapté de (Heijst & al., 1996)).*

2.2.1 Les grands types de connaissances utilisés en modélisation

Les connaissances du modèle conceptuel sont utilisées de différentes manières au cours de raisonnements, et les connaissances de résolution de problème sont relativement indépendantes des connaissances du domaine (connaissances statiques : faits, règles, données). Aussi, afin d'organiser les connaissances d'un modèle, et maximiser leur réutilisation, les méthodologies actuelles catégorisent ces connaissances suivant leur rôles au cours du raisonnement. Dans la littérature sur la modélisation de connaissances, notamment (Wielinga & al., 1992), (Musen & Schreiber, 1995), (Heijst & al., 1996), au moins cinq types de connaissances sont distingués (voir figure 2.3).

- Les *tâches* correspondent aux buts qui doivent être atteints durant la résolution de problème.
- Les *méthodes de résolution de problème* sont des manières d'atteindre les buts fixés. Pour cela, une méthode peut par exemple décomposer une tâche en sous-tâches. La décomposition totale ou partielle d'une tâche peut ainsi être dynamique ou bien définie statiquement. Dans ce dernier cas, le résultat est parfois appelé une *"instance de tâche"*. Comme nous emploierons souvent cette expression, et que nous désignerons également souvent aux "instances" de certains types ou classes ou types (les instances d'un type sont alors les individus (entités, objets) possédant les caractéristiques décrites par le type), notons qu'il ne faut pas confondre ces deux significations du terme "instance".
- Les *inférences* (ou *mécanismes*) décrivent les pas élémentaires d'un processus de résolution. Les inférences ont des entrées-sorties qui pointent sur des termes ou des connaissances du domaine et permettent donc de les utiliser au cours de l'exécution de la tâche à laquelle appartient l'inférence. Ces entrées-sorties sont souvent appelées des "rôles" car leurs noms (ou leurs types) représentent le rôle que jouent des termes ou des connaissances du domaine lors de l'exécution d'une inférence lorsque la tâche à laquelle elle appartient est exécutée. Un rôle est "dynamique" lorsqu'il représente des données pour une exécution ou des résultats intermédiaires, il est "statique" s'il représente des termes, des faits ou des règles du domaine. Des exemples de types de rôle dynamique sont : Symptôme, Observation, Paramètre, Hypothèse, Différence et Résultat_de_diagnostic. Des exemples de types de rôle statique sont : Modèle du système et Données_historiques. La structure décrivant le flot de données entre les diverses inférences d'une instance de tâche est appelée une *structure d'inférence*. Le contrôle sur ces inférences, e.g. leur ordonnancement, est décrit dans l'instance de tâche.
- Les *ontologies du domaine* décrivent les termes utilisables pour la modélisation des connaissances du domaine, et posent des contraintes sur ces utilisations.
- Les *connaissances du domaine* sont une collection d'assertions sur le domaine.

Les langages de représentation utilisés pour représenter ces différentes connaissances peuvent être semi-formels ou formels, et peuvent éventuellement être partiellement ou entièrement exécutables (i.e. un moteur d'inférence est associé au langage). Pour des raisons d'efficacité dans l'exécution, les langages exécutables nécessitent généralement de faire des choix d'implémentation, explicitement ou bien implicitement lorsque le langage impose des choix. Le cogniticien peut alors ne pas rester à un niveau conceptuel dans sa représentation. Par exemple, s'il utilise Prolog, il peut prendre en compte pour représenter un contrôle, le fait que l'ordre des règles a une importance pour l'exécution de ces règles : dans ce cas, le contrôle n'est pas explicitement représenté (comme c'est par exemple le cas avec un réseau de Petri) mais est implicite.

Les hypothèses ontologiques et épistémologiques faites par un langage de représentation peuvent être définies dans un méta-modèle de ce langage. En comparant les méta-modèles, des traducteurs peuvent être implémentés pour convertir dans un autre langage, des connaissances écrites dans un certain langage. Cette traduction peut être purement syntaxique, comme dans les traducteurs de KIF/ Ontolingua (Gruber, 1993) vers plusieurs langages exécutables (e.g. Loom (MacGregor, 1991) ou CYCL (Lenat & Guha, 1990a)), ou bien être plus conceptuelle en prenant en compte l'ontologie du domaine (Heijst & al., 1996).

2.2.1.1 Des ontologies pour les différents types de connaissances

La figure 2.3 résume ces divers éléments (types de connaissances, langages de représentation, etc.) et leur interactions. Pour chacun de ces types d'éléments, il est possible de définir une ou plusieurs ontologies, c'est-à-dire des "spécifications explicites de la conceptualisation" de ces éléments (Gruber, 1993), ou encore la collection des termes formels utilisables pour représenter ces éléments et la définition des inter-relations entre ces termes.

Une ontologie peut prendre la forme d'une *hiérarchie de termes* atomiques ou bien accompagnés de définitions, de *schémas conceptuels* spécifiant la structure de certaines connaissances (comme dans les bases de données), ou d'une *théorie*, i.e. d'un ensemble de règles logiques définissant un ensemble de termes et des contraintes sur leurs utilisations (comme dans Ontolingua). Cette liste n'est pas exhaustive. Les différents cas peuvent être combinés : des termes de différentes théories peuvent être liés par des liens de subsomption (comme dans Ontolingua). Notons enfin qu'une théorie peut inclure d'autre théories ("engagement ontologique d'une théorie à une autre ontologie") et donc être considérée comme plus "spécialisée" que celles-ci.

Il est possible de définir des ontologies pour chaque type d'éléments, et donc de distinguer des ontologies de tâches, de méthodes, d'instances de tâches, d'inférences, de domaines, de représentation et des ontologies de haut niveau (ou "génériques"). Ces dernières peuvent généraliser les autres ontologies (exceptées éventuellement les ontologies de représentation). Elles contiennent typiquement des concepts comme "Etat", "Evènement", "Processus", "Composant", etc.

Les ontologies de représentation spécifient les conceptualisations qui sous-tendent les formalismes de représentation de connaissances (Davis & al., 1993). Elles sont "*neutres"* vis-à-vis des entités du monde réel (Guarino & Boldrin, 1993). Les primitives qu'elles définissent permettent de décrire (représenter) les autres ontologies (y compris, partiellement, les ontologies génériques). Un exemple d'ontologie de représentation est la "Frame ontology" qui est utilisée dans Ontolingua et qui définit des termes comme "relation", "fonction", "classe", "instance-de", "toutes-les-instances-de", "arité", "même-valeur-que", "partition-de-classe", "relation-transitive", "relation-faiblement-transitive", "relation-N-1", "relation d'ordre". Cette ontologie est utilisée par Ontolingua pour traduire des connaissances exprimées en KIF (Genesereth & Fikes, 1992). vers des formalismes exécutables.

2.2.1.2 Les modèles génériques

De nombreux outils et méthodologies d'AC offrent des modèles génériques pour guider la recherche dans les données d'expertise d'informations pertinentes pour la résolution de problèmes, et pour guider la construction du modèle conceptuel. Comme le montre la figure 2.2, il peut s'agir de d'ontologies du domaine ou de modèles de tâches génériques. Ces derniers modèles peuvent également être vus comme des ontologies (ontologies de tâches, de méthodes, ...) mais cette terminologie n'est pas (encore) employée dans la littérature en AC. La structure des modèles de tâches génériques varie suivant la méthodologie. Par exemple, dans celle des "Tâches génériques" (Chandrasekaran, 1987), un modèle de tâche générique spécifie à la fois une méthode pour accomplir une tâche et la manière dont les connaissances du domaines utilisées par la méthode doivent être représentées. Par contre, les "modèles d'interprétation" de KADS-I (Wielinga & al., 1992) spécifient une méthode (avec une structure d'inférences et des propositions de contrôle) mais ne posent pas de contraintes sur la manière dont les connaissances du domaine utilisées doivent être représentées. Dans la méthodologie "Protégé" (Musen, 1989), un modèle de tâche générique spécifie à la fois une méthode et des classes d'objet à instancier pour décrire les connaissances du domaine.

2.2.2 Les activités en acquisition des connaissances

Nous résumons maintenant les principales activités de l'acquisition des connaissances, en nous inspirant principalement de la synthèse effectuée par Heijst & al. (1996). L'ordre donné ici n'indique pas que ces activités doivent être exécutées séquentiellement : le processus d'AC est un processus cyclique (Shadbolt & Wielinga, 1990). La table 2.1 compare divers outils d'AC sur les types de supports qu'ils offrent pour aider aux deux premières activités ci-dessous.

1) **Construction du modèle de tâche et de l'ontologie du domaine.** La première activité de modélisation est l'analyse de la tâche que le SBC doit réaliser ou dont l'expertise doit être sauvegardée, et sa décomposition en diverses tâches génériques. Pour chacune de celles-ci une méthode doit être choisie. La création du modèle de tâches peut se faire par sélection et adaptation (et combinaison) de modèle(s) de tâche générique, ou par création directe. Puis l'ontologie du domaine doit être créée. Cela est une opération compliquée qui doit généralement être faite par un cogniticien et non par un expert du domaine. Elle est facilitée si elle peut se faire par sélection et adaptation d'autres ontologies du domaine. Il faut ensuite définir les connexions entre le modèle de tâche et les termes et connaissances du domaines, ce qui se fait en connectant par des pointeurs ou des relations, les entrées-sorties des inférences (les rôles) à ces termes et connaissances du domaines.
Les outils d'AC peuvent aider à cette activité de trois manières (voir table 2.1) :
a) en offrant un ***éditeur*** (graphique) permettant de construire le modèle de tâche et l'ontologie du domaine, et pouvant effectuer des vérifications syntaxiques, sémantiques[1] et d'achèvement (i.e. tester si le modèle peut ou doit être complété) ; par exemple, Kads-Tool (Albert & Jacques, 1993), Krest (Steels, 1993) et Cokace (Corby & Dieng, 1996) incluent des éditeurs spécialisés qui effectuent de telles vérifications ;
b) en fournissant des ***bibliothèques de composants de modèle de tâche, d'ontologies du domaine, et des configurations typiques de ces composants*** ; par exemple, Kads-Tool, Krest, Kew (Anjewierden & al, 1992) et Dids (Runkel & Birmingham, 1994) offrent de telles bibliothèques ;
c) en ***guidant de manière active le processus d'assemblage de composants*** ; par exemple, Kew qui est basé sur la théorie GDM ("Generalized Directive Models" (Heijst & al., 1992)) fournit un support actif pour guider l'assemblage de composants de modèle de tâche.

2) **Création des connaissances du domaine** ***(cette création est souvent guidée par le modèle de tâche et l'ontologie du domaine, ou par des techniques d'interview)***. Cette création peut être effectuée par un cogniticien et validée par un expert du domaine ou éventuellement être directement effectuée par un expert. Plus le modèle de tâche et l'ontologie du domaine seront précis sur les connaissances à acquérir, plus l'extraction (i.e. recherche) et la modélisation de ces connaissances seront aisées et pourront être guidées ou contrôlées par un outil.
Les outils d'AC peuvent aider à cette activité de cinq manières (voir table 2.1) :
a) en ***vérifiant si les connaissances entrées sont cohérentes avec le modèle de tâche et l'ontologie du domaine*** (e.g. vérification syntaxique et sémantique par un éditeur ; comme le montre la table 2.1, la plupart des outils d'AC supportent cette activité) ;
b) en ***vérifiant l'achèvement de "l'instanciation" de ce modèle et de cette ontologi***e, c'est-à-dire en vérifiant si des connaissances ont été modélisées pour tous les types de connaissances contenues dans le modèle, et si pour chacun de ces types, toutes les connaissances nécessaires ont été modélisées (ceci n'est possible que si les différentes valeurs ou dimensions pour un type de connaissances ont été explicitées) ; de nombreux outils peuvent effectuer de telles vérifications, e.g. Kads-Tool (Albert & Jacques, 1993), Dids (Runkel & Birmingham, 1994) et Cue (Heijst & Schreiber, 1994);
c) en guidant ***l'instanciation des connaissances par des techniques de visualisation intuitives***, e.g. des langages semi-formels, des menus, des tableaux (e.g. dans Kads-Tool, Dids et Cue) ;
d) en engageant un ***dialogue*** structuré avec l'utilisateur (e.g. dans Dids et Cue) ;
e) en utilisant pour ces visualisations et dialogues, une ***terminologie*** adaptée au domaine (les éléments du modèle de tâche sont reformulés dans la terminologie du domaine), e.g. dans Cue.

1. Ici une vérification sémantique désigne un contrôle de types.

3) **Choix des techniques d'implémentation (langage, type de contrôle, etc.) et opérationalisation**. Un langage semi-formel est plus aisé à utiliser pour représenter les connaissances du domaine mais l'opérationalisation des connaissances représentées est plus complexe. Elle peut être entièrement automatique (e.g. Krest (Steels, 1993)) ou semi-automatique (e.g. Alto (Major & Reichgelt, 1990)). Dans le premier cas, le cogniticien ne peut contrôler l'implémentation qui peut être inefficace pour l'exécution, si le langage utilisé pour représenter les connaissances a une grande puissance descriptive. Dans le second cas, l'outil demande au cogniticien des informations supplémentaires lorsqu'il n'en dispose pas suffisamment pour effectuer des choix d'implémentation ou pour lever des ambiguïtés. Alternativement, un outil peut fournir des bibliothèques des procédures pour faciliter une opérationalisation manuelle du modèle conceptuel (c'est le cas dans Dids).

4) **Validation de l'opérationalisation et mise à jour du modèle conceptuel**. Les caractéristiques dynamiques du SBC sont testées sur un certain nombre de cas. Si des résultats s'avèrent incorrects, il faut corriger ou compléter le modèle conceptuel. Cela nécessite une "décompilation".
Les outils d'AC peuvent aider à cette activité de quatre manières :
a) en fournissant un mécanisme de trace pouvant montrer à différent niveaux d'abstraction les pas de raisonnement conduisant à une solution (e.g. Krest et Dids)
b) en inspectant les traces pour répondre à des questions du type "Pourquoi" et "Comment" (e.g. Kew) ;
c) en utilisant ces traces et/ou des raisonnements hypothétiques pour répondre à des questions du type "Pourquoi pas" et "Qu'est-ce qui se passe si" (e.g. Salt (Marcus & McDermott, 1989)) ;
d) en localisant dans le modèle conceptuel, les parties incomplètes ou erronées responsables de l'erreur détectée (e.g. Mole (Eshelman, 1988)).

5) M**ise à jour de la base de modèles génériques**. La construction du modèle conceptuel peut permettre de corriger ou de compléter certaines parties des modèles génériques utilisés (modèle de tâche ou ontologie du domaine). Des informations supplémentaires peuvent être apportées sur l'applicabilité de ces modèles et donc sur le système d'indexation de la base de modèles génériques si dans cette base les modèles sont indexés en fonction de leur condition d'application. Enfin, des parties du modèle conceptuel construit peuvent être ajoutées à cette base. Actuellement, aucun outil ne supporte cette activité (Heijst & al., 1996).

Heijst & al (1996) ont comparé différents outils d'AC suivant les activités qu'ils supportent (compte-tenu de la classification ci-dessus). La table 2.1. ci-après synthétise cette comparaison sur les activités de modélisation (activités 1 et 2 présentées ci-dessus) et positionne également CGKAT vis-à-vis des critères retenus.
Les premiers outils de construction de système experts, e.g. Emycin (Van Melle, 1979), supportaient principalement l'activité 4 décrite ci-dessus.
Les outils d'AC de la génération suivante permettaient de guider la création des connaissances du domaine (activité 4 ci-dessus) en s'appuyant sur des modèles de tâche particuliers codés en dur dans l'outil (e.g. Mole (Eshelman, 1988) et Salt (Marcus & McDermott, 1989)) ou sur un domaine particulier (e.g. Opal (Musen & al., 1988)), ou encore en guidant l'extraction et la modélisation des connaissances du domaine par des techniques d'interviews telles que les "grilles répertoires" (e.g. Aquinas (Boose & Bradshaw, 1988)) et les "laddering" (e.g. Alto (Major & Reichgelt, 1990)).
Les outils d'AC actuels supportent de plus la construction du modèle de tâche et de l'ontologie du domaine, généralement dans un langage non exécutable et avec des éditeurs spécialisés, et offrent souvent des bibliothèques de modèles de tâches génériques (e.g. Kads-Tool (Albert & Jacques, 1993), Kew (Anjewierden & al, 1992), Krest (Steels, 1993)). Ces modèles de tâches génériques ne sont plus codés en dur dans l'outil, ils peuvent être adaptés et combinés, et l'outil est parfois suffisamment générique pour pouvoir exploiter ces modèles de tâches (même adaptés par l'utilisateur) afin de guider l'extraction et la modélisation des connaissances du domaine, e.g. Protégé-II (Puerta & al, 1992) et Cue (Heijst & Schreiber, 1994).

Outil	**Construction du modèle de tâche**			**Construction de l'ontologie du domaine**			**Création des connaissances du domaine**				
	Edi-teur	Bibl.	Guide actif	Edi-teur	Bibl.	Guide actif	Vérif. cohér.	Vérif. achèv.	Instanc. tâches	Visual. intuitive	Dialogue
Emycin	-	-	-	o	-	-	o	-	-	-	-
Kas	-	-	-	-	-	-	o	o	-	-	-
Expert	-	-	-	-	-	-	o	-	-	-	-
Mole	-	-	-	-	-	-	+	+	-	-	+
Salt	-	-	-	-	-	-	+	+	-	-	+
Opal	-	-	-	-	-	-	+	+	+	+	-
Aquinas	-	-	-	o	-	-	+	+	+	+	o
Alto	-	-	-	o	-	-	+	+	-	+	-
Protégé	-	-	-	+	-	-	+	+	+	+	-
SBF	+	+	+	-	-	-	+	-	-	+	+
Keats	?	+	-	?	+	-	+	?	+	+	-
Shelley	+	+	-	-	-	-	+	-	-	+	-
Kads-Tool	+	+	-	+	-	-	+	+	-	+	-
Kew	+	+	+	-	-	-	+	+	-	+	-
Krest	+	+	-	+	+	-	+	?	?	?	?
Dids	?	+	-	?	+	-	+	+	-	+	+
Protégé-II	+	+	-	+	+	-	+	?	+	+	-
Cue	+	-	+	+	-	+	+	+	+	+	+
CGKAT	+	+	-	+	+	o	+	+	-	+	-

Table 2.1 Comparaison des supports offerts aux activités de modélisation par les outils d'acquisition de connaissances (extrait de (Heijst & al, 1996) + classement de CGKAT)
Pour chaque type de support, "+" signifie qu'il est fourni, "o" qu'il est fourni de façon limité, "-" qu'il n'est pas fourni, et "?" que les auteurs n'ont pas trouvé assez d'informations dans la littérature pour pouvoir répondre. Les termes suivants ont été abrégés : bibliothèque, vérification, cohérence, achèvement et visualisation. "Guide actif" désigne un guide plus ou moins actif du processus de modélisation. "Instanc. tâches" indique que la visualisation ou l'exploitation des modèles de tâche s'effectuent dans la terminologie du domaine. "Dialogue" indique un guide actif et interactif.

2.2.3 Extraction et modélisation ascendantes et descendantes

Deux approches sont possibles pour réaliser les activités d'extraction et de modélisation des données de l'expertise : l'approche ascendante (i.e. dirigée par les données) et l'approche descendante (i.e. dirigée par les modèles génériques).

Dans l'***approche ascendante***, les données de l'expertise sont analysées graduellement, et classées et structurées par le cogniticien qui pour cela a) peut appliquer des techniques générales de classification comme les "grilles répertoires" ou le "card sorting" et/ou b) peut être guidé par des modèles généraux qui peuvent être :

1. des *distinctions conceptuelles* fournies par une ontologie, une méthodologie, un langage ou encore un outil d'acquisition de connaissances. Ce sont par exemple les notions de tâches, d'inférences, de rôle, de méthode et leurs différentes inter-relations. La méthodologie KOD (Vogel, 1988), une méthodologie uniquement ascendante, préconise l'utilisation de sept sortes d'éléments et de onze sortes de relations (plus la relation de spécialisation). Les sept types d'éléments expriment les notions notions d'"acton" (ou processus), de "taxon" (entité nécessaire à un acton ou produite par un acton), de but, de méthode, de schéma causal, de schéma modal et de contrainte. Les onze sortes de relations expriment les rôles d'attribut, d'élément, de sous-tâche, de but, de méthode, de déclencheur, de palliatif, de précondition, de restriction, d'agent et de ressource.
2. des *modèles de méthodes de résolution générales*. Par exemple, un cogniticien qui suit le modèle de la méthode "Cover & Differentiate" utilisé dans Mole (Eshelmann, 1988) va classer les données dans les catégories suivantes : a) actions ou bien connaissances du domaine contribuant à trouver un ensemble d'hypothèses pour les états ou événements à expliquer ; b) actions ou bien connaissances du domaine permettant de choisir parmi les différentes hypothèse.

Ainsi, dans cette approche, une catégorie conceptuelle et sa description (son modèle) permet d'interpréter, de sélectionner et de regrouper diverses informations de même nature réparties dans un document. Ces informations peuvent alors être comparées et structurées. Les relations entre les connaissances de diverses catégories peuvent aussi être affinées. Un modèle de tâche peut alors être construit par assemblage des connaissances relatives à la résolution de problème.

Dans l'***approche descendante***, un modèle de tâche est d'abord construit par décomposition de la tâche à réaliser et généralement adaptation et assemblage de modèles de tâches génériques, puis la recherche (extraction) des données à modéliser et leur modélisation sont guidées par les attentes fixées par le modèle de tâche : celui-ci est "instancié". Ce sont principalement les entrées-sorties des inférences (les rôles) qui guident la recherche ou l'élicitation des données à modéliser (pour construire l'ontologie du domaine et les connaissances du domaine) et fixent des contraintes sur leur organisation.

Dans la pratique, ***ces deux approches doivent être combinées***.
En effet, avec une approche uniquement ascendante, il est difficile au cogniticien de savoir quelles sont les informations pertinentes à modéliser et les structurations pertinentes à utiliser pour cela.
Inversement, les connaissances du domaine ne peuvent être organisées uniquement avec une approche descendante, et pour reformuler un modèle de tâche générique dans les termes du domaine, un lexique du domaine doit avoir été créé (analyse ascendante). De plus, dans l'approche descendante, le choix des modèles de tâches génériques est déterminant et nécessitent préalablement une analyse ascendante du domaine pour savoir quels modèles appliquer. Le contenu et l'organisation du domaine doit être suffisamment connus pour ne pas se tromper de modèle et appliquer un modèle assez spécifique. Bien qu'une erreur à ce niveau aie de lourdes conséquences sur la suite de la modélisation, les méthodologies actuelles indiquent rarement comment effectuer cette analyse du domaine, ou comment combiner les approches ascendantes et descendantes (Rademakers & Vanwelkenhusen, 1992). On peut toutefois trouver dans (Nwana & al., 1991) quelques techniques pour effectuer une caractérisation du domaine.

2.2.3.1 L'extraction automatique de connaissances

L'approche ascendante peut être guidée ou partiellement réalisée par un outil d'analyse terminologique des documents, ou par une extraction et représentation partielles mais automatiques ou semi-automatiques du contenu sémantique des documents.

- *Les techniques d'analyse de données* : ce sont des statistiques sur les occurrences et co-occurrences d'éléments de documents dans de gros documents. Elles peuvent être utilisées pour créer des index sur des documents ou extraire la terminologie d'un domaine ; des thésaurus et des techniques d'analyse grammaticale sont généralement utilisés en parallèle pour structurer les termes sélectionnés par des relations sémantiques, e.g. dans LEXTER (Bourigault, 1995).
- *Les techniques d'analyse syntaxique et sémantique* : elles permettent d'extraire des représentations plus ou moins complexes ou précises d'éléments de documents et de leurs relations. Dans la tâche d'acquisition de connaissances à partir de documents, l'isolement de concepts importants dans des textes techniques est plus abordable : par exemple les outils décrits dans (Feng & al., 1994) trouvent des classes de mots ou de phrases équivalents, et exploitent pour cela le dictionnaire électronique Collins et la base générale de connaissances teminologique WordNet.

Nous n'adressons pas plus dans cette thèse, les techniques d'extraction automatique de connaissances à partir du langage naturel. Ces techniques sont complémentaires à celles que nous présentons et développons.

2.2.3.2 Exemples de modèles de tâches génériques pour l'approche descendante

Les figures 2.4 et 2.5 représentent respectivement les structures d'inférences (flots de données entre inférences) inclues dans le modèles de tâche "Cover & Establish" et de l'instance de tâche "Systematic diagnosis" (ces modèles appartiennent à la méthodologie KADS (Wielinga & al., 1992)). Rappelons qu'une instance de tâche est la décomposition d'une tâche par une méthode.
La figure 2.6 donne la description dans le langage CommonKADS CML (Schreiber & al., 1994) de la méthode "generate & test". Cette méthode peut être utilisée pour créer une instance de tâche pour la tâche "Cover & Establish". Wielinga & al. (1992) donne une description en CommonKADS CML pour une telle instance de tâche (cette description ressemble à celle de la méthode).

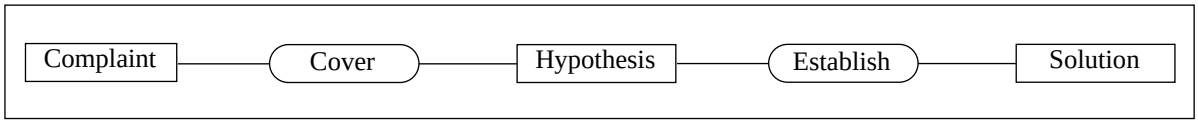


Figure 2.4: *La structure d'inférences de la tâche "Cover & Establish" (Wielinga & al., 1992).*

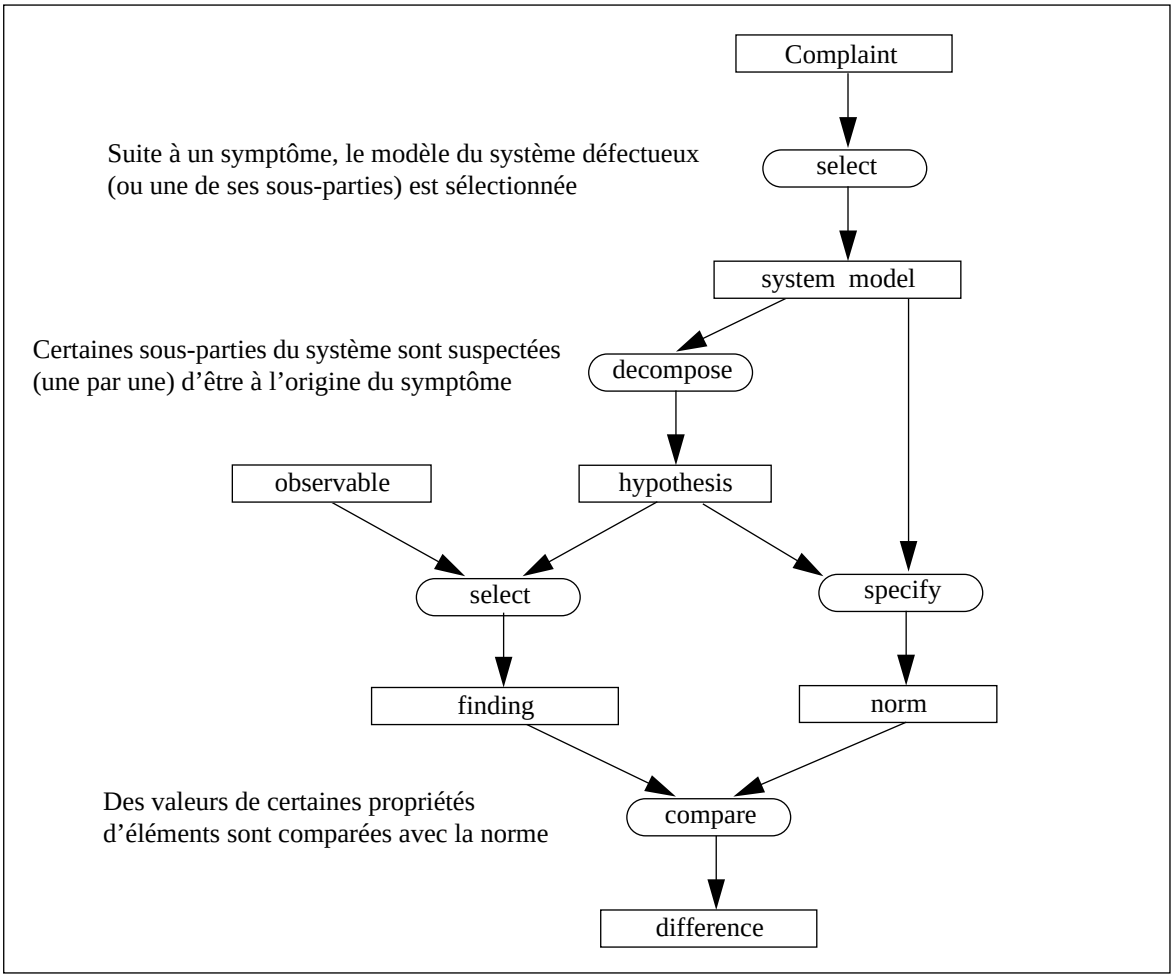


Figure 2.5: *La structure d'inférences de l'instance de tâche générique de "Diagnostic systématique" (extrait de Wielinga & al., 1992).*

problem-solving-method *generate-and-test*
goal G: Find (S: solution)
task-characterisation criterion1 (S) ∧ criterion2 (S) |- criterion1 (S)
control-roles C: complaint H: hypothesis
sub-tasks generate (complaint, hypothesis) test (hypothesis)
method-definition A1: ∀x solution (x) |- generate (x)
A2: ∀x generate (x) ∧ test(x) |- solution (x)
A3: ∀x generate (x) |- criterion1 (x)
A4: ∀x test (x) |- criterion2 (x)
A1 ∧ A2 ∧ A3 ∧ A4 |- <P1> ∃ s solution(s)
tasks-expression-schema P1
repeat generate (x)
until test(H)
result (H)

Figure 2.6: *La description de la méthode "Cover & Establish" dans le langage CML (Wielinga & al., 1992).*

La ***sélection par le cogniticien d'un modèle de tâche*** est généralement basée sur une *taxinomie de types de problèmes* et sur les *contraintes d'application* du modèle de tâche.

Pour les ***contraintes d'application***, les différentes approches semblent converger vers une classification en trois catégories (Steels, 1990) :
1) les contraintes sur la nature des connaissances utilisables par la tâche à réaliser (e.g. l'existence d'un modèle structurel, fonctionnel, comportemental, d'état, de faute, causal ou encore topologique) ;
2) les contraintes imposées par l'environnement de la tâche à réaliser (e.g. contraintes sur le temps de réponse) ;
3) l'adéquation de la tâche générique à certains types d'actions.
Par exemple, le modèle de tâche "Cover & Differentiate" peut être considéré comme applicable a) si la tâche à réaliser peut être vue comme une tâche de classification, b) s'il existe des connaissances permettant de générer des hypothèses, et d'autres permettant de les discriminer, et c) si tout symptôme est expliqué par une hypothèse et seulement une. Pour vérifier ces conditions, une analyse du domaine de l'application doit être effectuée.

S'il est possible de dresser une ***taxinomie*** d'*instances de tâches génériques* (i.e. décomposition d'une tâche générique par une méthode), cela est plus difficile pour *des tâches génériques*. En effet, généralement ces tâches génériques doivent être combinées pour résoudre un problème. Les tâches génériques, accompagnées de méthodes pour les décomposer, sont plus utiles pour le cogniticien que des instances de tâches car elles sont plus génériques et offrent donc un meilleur guide pour décomposer un problème.
C'est pourquoi les auteurs de la méthodologie KADS, qui ont d'abord fourni une bibliothèque d'instances de tâches génériques dans KADS-I (Breuker & al., 1987), fournissent maintenant dans CommonKADS (Breuker & van de Velde, 1994) une bibliothèque de tâches génériques et de méthodes pour les décomposer. Pour organiser/indexer ces instances de tâches, une *taxinomie* est donnée (voir figure 2.7). Pour les tâches génériques, c'est plutôt un *réseau de dépendance de données* qui est fourni (voir figure 2.8).

Pour guider le cogniticien dans l'abstraction des données et la création de modèles de tâches, la méthodologie KADS offre également des *typologies d'inférences et de rôles*.

analyse_de_système
- identification *(d'une catégorie)*
  - classification
    - classification_simple
    - diagnostic *(identification d'une catégorie de faute)*
      - diagnostic_faute_unique
        - diagnostic-par-classification-heuristique *(modèle du domaine : modèle de faute)*
        - diagnostic_systématique *(modèle du domaine : modèle de rectitude)*
          - localisation structurelle
          - localisation causale
      - diagnostic_fautes_multiples
    - évaluation *(de correspondance ou d'adéquation : identification d'une classe de décision)*
  - surveillance *(identification d'une incompatibilité entre une norme et une observation)*
- prédiction *(d'un état passé ou futur du système)*
  - prédiction_de_comportement
  - prédiction_de_valeurs

modification_de_système
- réparation
- remède
- contrôle
  - maintenance

synthèse_de_système
- transformation
- conception
  - conception_par_transformation
  - conception_par_raffinement
    - conception_par_raffinement_à_flot_unique
    - conception_par_raffinement_à_flots_multiples
  - configuration
- planification
- modélisation

Figure 2.7: *Une taxinomie de type de problèmes dont les feuilles sont modélisées par les instances de tâches génériques de la bibliothèque de KADS-I (Breuker & al., 1987).*

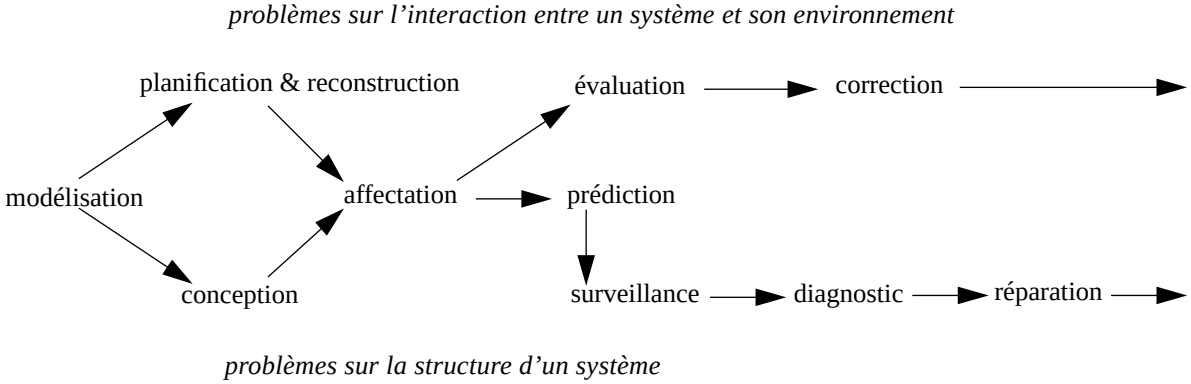


Figure 2.8: *Le réseau de dépendance de données entre les types de problèmes modélisés par les tâches génériques de la bibliothèque de CommonKADS (extrait de (Breuker, 1994)).*

2.3 La représentation et l'organisation des connaissances

2.3.1 Généricité du langage de représentation de connaissances

Nous avons noté en fin de section 2.2.2 que les outils d'AC actuels sont maintenant génériques vis-à-vis des modèles de tâches génériques (l'outil ne les intègre pas en dur mais exploite une bibliothèque de tâches génériques) et que certains outils sont suffisamment génériques pour pouvoir exploiter ces modèles de tâches, adaptés ou non par l'utilisateur, pour guider l'extraction et la modélisation des connaissances du domaine, e.g. Protégé-II (Puerta & al, 1992) et Cue (Heijst & Schreiber, 1994).

Cependant, pour permettre la construction par l'utilisateur de modèle de tâches ou de connaissances du domaines, les langages ou les outils de représentation de connaissances intègrent généralement des distinctions conceptuelles qui pourraient être définies dans une ontologie et exploitées via cette ontologie. Par exemple, la figure 2.6 illustre la syntaxe de CommonKADS CML pour la description de méthodes. On peut y relever sept mots-clés correspondant à sept distinctions conceptuelles : problem-solving-method, goal, task-characterisation, control-roles, sub-tasks, method-definition et tasks-expression-schema.

L'utilisateur est alors guidé par la syntaxe du langage ou par les menus des outils qui sont adaptés à la saisie de certaines caractéristiques de certains types d'éléments (ou connaissances), et l'outil peut aisément effectuer certains contrôles de *cohérence et d'achèvement* (cf. section 2.2.2).
En contrepartie, l'utilisateur ne peut ou peut difficilement adapter, spécialiser ou compléter ces types d'éléments ou leurs caractéristiques.

Voici deux autres exemples :

1. Dans l'outil K-Station (Albert & Vogel, 1989) (ILOG, 1993a) qui implémente la méthodologie KOD (Vogel, 1988), la représentation des connaissances consiste à représenter six sortes d'éléments qui expriment les notions d'"acton" (ou processus), de "taxon" (entité nécessaire à un acton ou produite par un acton), d'attributs (propriétés d'un acton), de schéma causal, de schéma modal et de contrainte. Des menus dédiés à chacun de ces éléments sont proposés pour les décrire. Par exemple, pour décrire un acton, l'utilisateur peut indiquer notamment une "version" (ou méthode), une "décomposition" (sous-processus reliés par des liens de successions) et les taxons nécessaires ou produits par un acton : l'"émetteur" (le taxon agent du processus), le "récepteur" (le taxon manipulé) et les autres "ressources" nécessaires au processus. Les différents éléments (taxons, actons, attributs, etc.) peuvent ainsi être reliés par une quinzaine de relations (dont une relation de spécialisation entre les taxons). En résumé, ces différentes sortes d'éléments et de relations sont proposées via des menus dédiés et non dans une ontologie où elles pourraient être spécialisées ou complétées. L'outil est dédié à la méthodologie KOD.
2. Similairement, dans l'outil Kads-Tool (Albert & Jacques, 1993) (ILOG 1993b) qui implémente la méthodologie KADS (Wielinga & al., 1992), la description des tâches est limitée : les propriétés associables aux tâches sont prédéfinies. En revanche, conformément à la méthodologie KADS, la description des connaissances du domaine est plus libre puisqu'un réseau sémantique de concepts lié par des relations peut être construit et que ces concepts peuvent être organisés dans des taxinomies.

Il peut donc être intéressant lorsque la généricité d'un outil d'AC est considérée comme un atout, que cet outil permette d'utiliser un formalisme de représentation des connaissances qui, tel celui des Graphes Conceptuels, intègre peu de distinctions conceptuelles mais soit basé sur l'exploitation d'une ontologie. Cet outil d'AC devra alors exploiter les définitions ou les modèles associés à ces distinctions conceptuelles pour guider et contrôler la modélisation des connaissances relatives à ces distinctions.

2.3.2 Encapsulation, vues et modules

La modélisation des connaissances suppose la représentation explicite de relations entre des objets : l'organisation des connaissances facilite la recherche de connaissances, les inférences et la production d'explications. Nous pensons que certaines formes d'*encapsulation* ou bien de *modularisation* proposées par des langages de représentation de connaissances, ne favorise pas la représentation explicite de relations entre objets.

2.3.2.1 Encapsulation

Par "encapsulation", nous désignons la construction d'un objet par composition d'autres objets. Il faut si possible éviter de représenter un objet comme une "boîte" *rassemblant et encapsulant* d'autres objets, si les relations entre ces objets et la boîte ne peuvent être également représentées.

Par exemple, supposons qu'en utilisant un langage à objets, un cogniticien définisse une classe "Voiture" par composition des classes "Moteur", "Roues", "Carrosserie", "Couleur", "Conducteur", et "Consommer de l'essence", par exemple de la façon suivante :

```
class Voiture
{ moteur : Moteur;
  roues : Roues;
  carosserie : Carosserie;
  couleur : Couleur;
  conducteur : Conducteur;
  consommer_essence : Consommer_essence;
}
```

Dans cet exemple, les noms des attributs n'apportent pas d'informations supplémentaires par rapport aux classes qu'ils contiennent : contrairement à la représentation ci-dessous, les relations sémantiques entre la classe Voiture et les autres classes ne sont pas représentées.

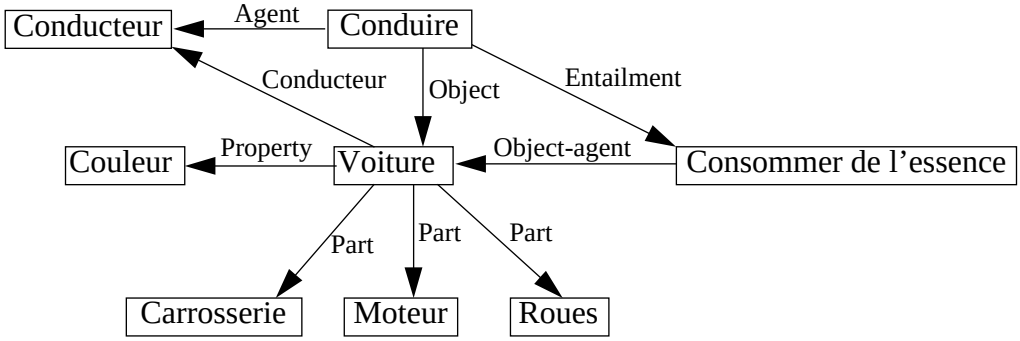


Figure 2.9: *Un réseau sémantique explicitant des relations entre une classe d'objet "Voiture" et d'autres classes.*

Il peut ne pas être aisé de représenter des relations sémantiques *en utilisant des attributs* dans un langage à objets. Par exemple, dans la figure ci-dessus, le fait que la classe Voiture soit reliée à plusieurs classes d'objets par la relation "Part"[1] peut poser problème dans certains langages. Cependant, différentes méthodes peuvent être utilisées dans les formalismes à objets pour représenter explicitement des relations entre des objets ou des classes d'objets. Dans les exemples

1. Nous présenterons au chapitre 5 une ontologie de types de base pour les relations et les concepts. Dans cette ontologie, nous avons repris les noms anglais sous lesquels ces types sont le plus connus. Aussi, dans nos exemples, nous rdésignons également ces types avec leurs noms anglais.

suivants, nous ne précisons pas si un objet désigne une classe d'objet ou bien une instance (notons à ce propos que le langage a une puissance descriptive plus grande si aussi bien des classes d'objet que des instances peuvent être connectées par des relations).
1) Représenter les relations sous forme d'objet, les attributs des objets servant alors à assurer les connexions entre les objets "concept" et les objets "relation". Par exemple, Othelo (Fornarino & Pinna, 1990) offre et gère de tels objets "relation".
2) Utiliser un formalisme objet permettant de décrire dans un "méta-modèle" les relations entre objets, comme par exemple SHOOD (Nguyen & al., 1992).
3) Utiliser un formalisme à objets permettant de décomposer un objet selon plusieurs points de vue, chaque point de vue correspondant à une relation (i.e. pour un point de vue, les composants d'un objet ont des relations de même type avec cet objet). C'est le cas par exemple du système TROPES (Marino, 1993).
4) Utiliser un formalisme de réseau sémantique après avoir décrit ses composants avec des classes d'objet et construit un interpréteur de ce formalisme qui crée et gère ces classes d'objets. C'est le cas par exemple de CoGITo pour le formalisme des Graphes Conceptuels.

L'encapsulation peut également exister dans les réseaux sémantiques. Il s'agit alors de noeuds qui incluent ou représentent un ensemble d'autres noeuds. Nous présenterons au chapitre 4, à propos des contextes dans les GCs, plusieurs interprétations pour de tels emboîtements.

L'encapsulation peut avoir de multiples fonctions et est indispensable par exemple pour contextualiser un ensemble de connaissances ou bien pour construire une "vue" sur des connaissances. Les vues ont de nombreux intérêts en AC comme dans d'autres domaines. Dans la suite de cette section, nous précisons ce que nous signifions par le terme "vue". Dans la section suivante, nous montrons certains intérêts des "vues" par rapport aux "modules". Puis nous montrerons les usages actuels de cette notion ou de notions proches.

2.3.2.2 Vues, points de vue et modules

***Définition d'une vue et d'un point de vue*.**
Nous appelons "***vue***", un objet qui réunit un ensemble d'objets[1] *sélectionnés* sur un même "***point de vue***" (ensemble de critères de sélection) et donc entretenant une même *relation* avec ce point de vue. Une vue n'est pas une contextualisation : les objets (concepts, relations, assertions, etc.) qu'elle contient ne sont pas locaux à la vue et doivent donc être cohérents avec les autres objets de la base (une classe d'objet dans un formalisme à objets n'est pas une vue puisqu'elle définit des attributs qui sont locaux à la classe). L'affichage d'une vue permet de *visualiser d'une certaine façon* un ensemble d'objets sélectionnés[2] sur un même point de vue. Si plusieurs vues incluent un même objet, une modification de cet objet dans une vue doit être répercutée dans les autres vues. La description d'un point de vue peut donc contenir 1) une description de *critères de sélection* avec une requête (une expression) ou une méthode de sélection (un programme), et/ou 2) un *modèle de présentation* des éléments sélectionnés.

La description d'un point de vue peut se faire avec une requête arbitrairement complexe ou une méthode de sélection. Une vue peut ainsi être le résultat de l'exécution d'une requête ou autre moyen de sélection. Une vue peut également être remplie ou complétée directement par un cogniticien (dans ce cas, la sélection est effectuée par le cogniticien). La figure suivante donne l'exemple de vues sur un réseau sémantique.
Dans une vue, les objets sont liés par *une* même *relation* à l'objet "vue" puisqu'ils ont été sélec-

1. Les objets sélectionnés peuvent être des représentations formelles ou informelles, e.g. un terme formel, un mot, une expression logique, une phrase, un GC, un graphique ou une image. Le support d'une vue peut donc être un document multi-media.

2. Eventuellement, le sous-ensemble d'objets sélectionnés peut être égal à l'ensemble des objets sur lequel la sélection va être faite.

tionnés sur un même point de vue. *Si cette relation est sauvegardée* (et donc connue du système de gestion de vues), comme l'illustre la figure 2.11, la représentation de connaissances avec une telle forme d'encapsulation n'entraîne pas de perte d'information par rapport à la représentation par réseau sémantique sur un seul niveau : *depuis l'une de ces représentations, l'autre peut être générée*. ***Si tel est le cas, un "système de gestion de vues" peut permettre à un cogniticien de rentrer des connaissances dans le format de représentation souhaité, le système pouvant afficher les connaissances dans d'autres formats si le cogniticien le désire.*** Cependant, nous ne connaissons aucun système qui propose un tel choix de format. Par contre dans certains systèmes hypertextes, comme MacWeb (Nanard & al., 1993a, 1993b) et CGKAT, il existe ce que nous appelons un ***"système de génération de vues"*** ; avec un tel système :
1) une vue peut être créée manuellement ou générée à l'aide d'une requête, et
2) la modification d'un élément dans une vue est automatiquement répercutée dans toutes les autres vues existantes qui incluent cet élément.

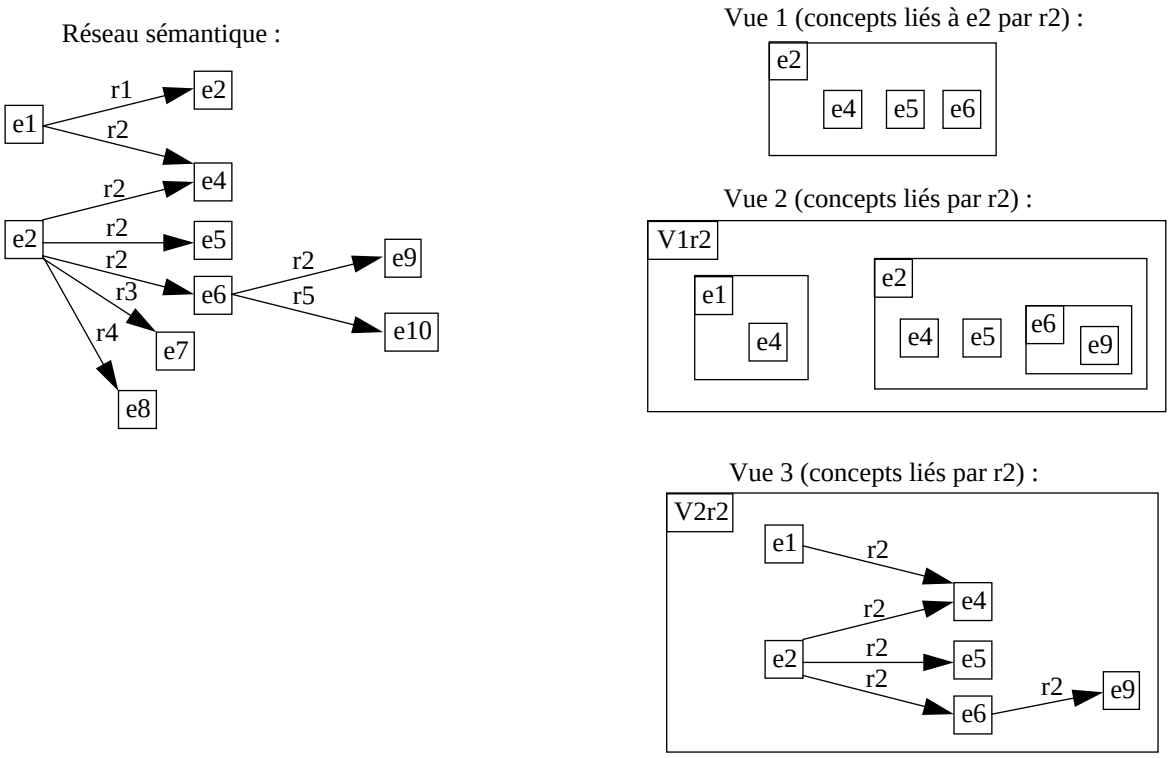


Figure 2.10: *A gauche un réseau sémantique, à droite des vues sur ce réseau.*

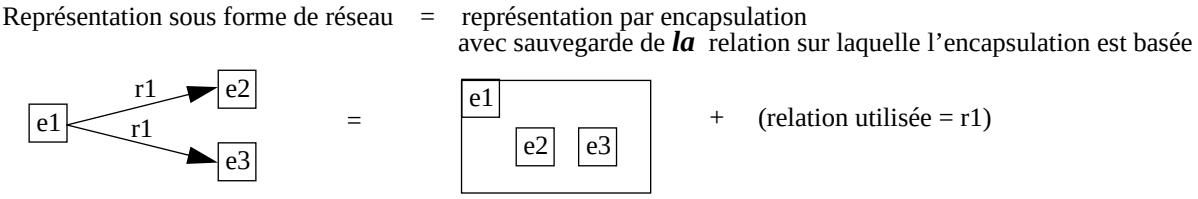


Figure 2.11: *Une représentation par encapsulation ou bien sous forme de réseau sémantique peuvent être équivalentes moyennant certaines précautions.*

Similairement, nous présenterons au chapitre 4 qu'il est possible et intéressant de contextualiser des connaissances de telle sorte qu'une seule relation lie les objets contextualisés avec l'objet "contexte" qui les inclut.

Définition d'un "module"[1]

Par "***module***", nous désignons un objet d'une base de connaissances permettant de regrouper un ensemble cohérent de connaissances (faits, règles, ontologies, etc.) mais tel que des connaissances appartenant à différents modules peuvent être contradictoires. Par exemple, les "théories" dans Ontolingua sont des modules.

A l'intérieur d'un module, un terme ne doit donc représenter qu'un seul concept : il ne doit avoir qu'une seule définition par conditions nécessaires et suffisantes, ou bien des définitions équivalentes. Mais des modules différents peuvent redéfinir le même terme de différentes façons, comme c'est par exemple le cas dans les théories.

Notons que cette notion de module est très générale : avec notre définition, un fichier ou une classe d'objet peuvent jouer le rôle de modules. Il ne s'agit donc pas de la notion de module classiquement utilisée en génie logiciel, où un module comprend 1) une représentation externe, publique (l'interface du module) qui permet d'utiliser le module, et 2) une représentation interne qui constitue la réalisation effective du module (Pierra, 1991) (Serres, 1995).

2.3.2.2.1 Comparaison des vues et des modules

L'intérêt des modules comme des vues est de permettre de construire ou de visualiser des connaissances de façon modulaire. Cependant dans une vue, les connaissances ne sont pas contextualisées et doivent donc être cohérentes avec l'ensemble des autres connaissances de la base (ou du module englobant si la vue est créée dans un module). Une vue n'est que le résultat d'une sélection manuelle ou automatique de connaissances sur un certain critère (le point de vue retenu) et ne permet pas de redéfinir des termes (i.e. il n'y a pas de termes locaux à la vue). Ainsi, un cogniticien, qu'il utilise un "système de gestion de vues" ou seulement un "système de génération de vues" est conduit à *distinguer et à situer* les connaissances qu'il ajoute par rapport aux autres connaissances de la base.

Au contraire, le cogniticien qui crée un module peut ignorer les autres connaissances de la base (cela peut être un avantage !). Notons à ce propos que plusieurs cogniticiens peuvent modéliser une même expertise, et que plusieurs experts peuvent intervenir dans l'expertise, en concurrence ou en coopération. Il y a donc de nombreuses façons de diviser la connaissance en modules. Contrairement à une vue, un module peut définir et utiliser des termes locaux au module, et il est construit à la main (une vue peut être générée ou regénérée à partir des relations qui existent entre les objets de la base). Le problème est alors soit, après la construction des modules, de comparer et de fusionner leurs connaissances (cela doit généralement se faire manuellement), soit de les organiser et indexer de telle sorte que cette fusion ne soit pas nécessaire pour une modélisation adéquate des connaissances de l'expertise, c'est-à-dire pour les résolutions de problèmes ultérieures, la recherche d'informations, la génération d'explications, la compréhension de la base, etc.

Dans cette dernière optique, comme il est difficile de gérer une même connaissance dans différents modules, le créateur d'un module cherche plutôt à rassembler des connaissances qui ont le maximum de relations entre elles et le minimum de relations avec les connaissances des autres modules (les "principes de modularité" selon (Heijst & al. (1996)). Mais comme chaque connaissance peut être liée à de nombreuses autres connaissances (dont des concepts représentant l'expert source de la connaissance, le cogniticien qui l'a représentée, et le domaine dont elle est issue), le problème de la division des connaissances en plusieurs modules est complexe et parfois arbitraire (ce problème est illustré plus loin). Il est cependant possible de structurer les modules entre eux par des relations d'inclusion, et de compléter la structuration par modules 1) en définissant des références ou des passerelles[2] entre des connaissances de différents modules, et 2) en structurant des termes de différents modules dans une même hiérarchie. Ces diverses relations facilitent la recherche d'infor-

1. Cette définition est indépendante de CGKAT et du formalisme des Graphes Conceptuels. Elle ne nous sert qu'à comparer deux types d'encapsulation : ce que nous appelons des vues et ce que nous appelons des modules.

2. Une passerelle peut être une relation de subsomption, une équivalence, une règle d'inférence permettant d'effectuer la traduction entre des termes, etc.

mations par navigation entre les modules et facilitent leur réutilisation. Toutefois, si un module est créé simplement pour modulariser des connaissances selon des points de vue et non pour les contextualiser, il nous semble préférable, pour les raisons évoquées dans les paragraphes précédents, d'utiliser un réseau sémantique ou des vues afin d'exprimer ces points de vue, puis de générer des vues pour visualiser les connaissances selon les points de vue désirés (les connaissances de chaque vue peuvent alors être modifiées et complétées).

Illustrons les problèmes de création de modules avec un exemple. Heijst & al. (1996) ont utilisé Ontolingua pour créer sous forme de théories, une ontologie sur le diagnostic de maladies du coeur. Leurs critères pour partitionner l'ensemble des définitions dans des théories et respecter les "principes de la modularité" furent 1) de les regrouper dans des "catégories naturelles" du domaine (e.g. patient) ou de la résolution de problème (e.g. diagnostic), et 2) de minimiser le nombre d'inclusions entre les théories afin de permettre un usage plus souple de ces théories et réduire le coût de l'engagement à une théorie (lorsqu'une théorie inclut une autre théorie, elle "s'engage" à respecter ses définitions, c'est-à-dire à être cohérente avec elle). Pour satisfaire cette minimisation, les théories ne sont pas organisées par niveau d'abstraction mais par "catégories naturelles". Une théorie n'incluant donc pas de théorie plus générale qu'elle, c'est-à-dire contenant des définitions moins détaillées, ils n'ont pu créer de hiérarchie de termes (hiérarchie de définitions) appartenant à différentes théories. Pour pallier à cela et permettre l'accès et la réutilisation des définitions entre les domaines ainsi qu'entre les tâches et méthodes de résolution de problèmes, ils ont créé deux hiérarchies, une pour les domaines de la médecine, une autre pour les tâches et méthodes de diagnostic en médecine, et ils ont indexé (i.e relié) les définitions dans les théories avec les concepts de ces hiérarchies. La recherche d'informations peut alors s'effectuer par navigation dans ces hiérarchies et des concepts de ces hiérarchies vers les théories dont les définitions sont indexées par ces concepts.

L'organisation proposée par Heijst & al. (1996) apparaît complexe et ne favorise pas la représentation de certaines relations entre les connaissances, e.g. les relations d'abstraction. Comme dans cette ontologie, les théories (et donc leurs définitions) sont cohérentes entre elles, une solution plus simple, uniforme et exploitable pour la gestion de l'expertise et la résolution de problèmes, nous semble être 1) de représenter explicitement pour chaque définition les relations (intéressantes pour l'expertise) qui la lient avec d'autres connaissances (catégorie naturelle, domaine, tâche, méthode, autre définition, etc.), 2) de structurer de la même façon ces autres connaissances, et 3) de générer des vues pour regrouper les connaissances selon différents critères ou combinaisons de critères. Notons que la représentation des relations peut s'effectuer en utilisant des vues.

Notons également que dans une taxinomie, donner différents parents à un élément revient à représenter différents points de vue sur cet élément. Si aucun des parents n'encapsule d'attributs locaux (e.g. les attributs d'une classe d'objet d'un formalisme à objets), c'est-à-dire si aucun des parents ne se comporte comme un module, l'héritage multiple ne peut entraîner de conflits de noms[1]. Une taxinomie peut ainsi être considérée comme une collection structurée de points de vue.

Heijst & al. (1996) notent que les caractéristiques d'un langage de création d'ontologies ne sont pas encore claires. KIF/Ontolingua, les GCs, LOOM (MacGregor, 1988) et CYCL (Lenat & Guha,

1. Sinon, des techniques doivent être utilisées pour traiter ou bien éviter les problèmes de l'héritage multiple. Citons à ce propos le système TROPES (Marino, 1993) qui utilise la notion de point de vue. TROPESest un système de représentation utilisant un formalisme à objets et orienté vers la classification d'instances de classes d'objets. TROPES permet à un objet d'être instance de plusieurs classes. Plusieurs points de vue sur cet objet peuvent ainsi être représentés. De plus, les classes d'objets sont organisées dans diverses taxonomies, chaque taxonomie étant relative à un point de vue et étant un arbre, ce qui évite les problèmes de conflits de noms lors de l'héritage d'attributs. Une classe d'objet peut appartenir à plusieurs de ces arbres et différents attributs peuvent être déclarés pour cette classe dans chacun des arbres (un arbre est donc un module puisque différents arbres peuvent définir différemment une même classe (ou classes de même nom)). Des passerelles peuvent exister entre des classes de différents arbres pour permettre d'exprimer l'égalité ou l'inclusion des extensions de ces classes. Des liens d'égalité existent entre les classes de même nom. Enfin, un objet peut être instance de plusieurs classes mais appartenant à des arbres différents afin d'éviter les conflits.

1990a) sont de tels langages, et pour des ontologies limitées à des modèles de tâches, citons Model (Tu & al. 1995) et CommonKADS CML (Schreiber & al., 1994). Nous pensons qu'un langage pour créer des ontologies doit avoir des caractéristiques similaires à celles d'un langage général de modélisation des connaissances, c'est-à-dire être 1) compréhensible et d'une grande puissance descriptive, 2) formel et si possible efficace. Ajoutons maintenant qu'un mécanisme de génération de vues, interne au langage ou bien externe comme c'est le cas dans CGKAT, semble intéressant pour regrouper des connaissances selon des points de vue et ainsi éviter, lorsque cela est possible, le regroupement manuel et statique de connaissances selon un nombre fixe et limité de points des vues. Nous synthétisons dans la section 2.4 différents guides méthodologiques pour la construction d'ontologies.

2.3.2.2.2 Usages habituels des vues et points de vue

Nous avons précisé notre définition des "vues" et "points de vue". Sous différents termes, des notions similaires ont déjà été définies et exploitées en AC, en génération d'explications, dans les bases de données et dans les systèmes hypertextes. Voici donc un bref panorama de l'usage de ces notions.

Dans les ***bases de données***, une "vue" est une sélection et agrégation de données de la base. Elle est décrite à l'aide de requêtes qui font intervenir des opérateurs de sélection, de jointure, des fonctions de calcul. L'écriture d'une requête peut être assimilée à la description d'un point de vue.

De manière similaire, dans les ***systèmes hypertextes***, une "vue" est une sous-partie du réseau hypertexte qui peut être générée avec une requête (un réseau hypertexte peut représenter un réseau sémantique). Nous détaillerons dans le prochain chapitre les divers types de requêtes qui peuvent être effectuées dans un réseau hypertexte.

L'usage de systèmes hypertextes et de leurs techniques hypertextes de génération de vues est intéressant en ***acquisition des connaissances*** pour rechercher, visualiser et structurer des informations ou des connaissances (Nanard & al, 1993b) (Neubert & al., 1994) (ceci sera également développé dans le chapitre suivant). Le cogniticien peut ainsi durant l'élicitation et la modélisation des connaissances se focaliser sur des connaissances relatives à un certain point de vue, tout en pouvant les connecter à d'autres connaissances.
Sous le terme de "vue", Schmidt (1995) désigne aussi bien des vues que des modules et note que ces objets sont utiles aussi bien en modélisation ascendante que descendante. En modélisation ascendante, ces "vues" permettent de structurer selon différents points de vue les connaissances du domaine puis de la résolution de problèmes. En modélisation descendante, ces "vues" sont des modèles de tâche, qui par les entrées-sorties de ces inférences (les rôles), fournissent des points de vue pour abstraire et organiser les connaissances du domaine.

Dans la littérature sur les ***explications***, de nombreuses définitions existent pour les termes de vue, de point de vue, de perspective et de type de vue, mais l'on retrouve souvent des notions proches de celles que nous avons définies. Par exemple, pour Acker & Porter (1994), un "point de vue est une collection cohérente de faits qui décrivent un concept à partir d'une perspective particulière" (notre définition d'une vue ressemble à cette définition mais celle-ci pourrait aussi s'appliquer à un module).

1. ***Pour la génération d'explication***, l'accent est mis sur les types de points de vue à prendre pour décrire un objet (entité, fait, règle, raisonnement, résultat de raisonnement, etc.), et sur l'ordre dans lequel les connaissances relatives à ces points de vue doivent être présentées, ceci en fonction du type de question, du type d'objet à présenter, du type d'utilisateur et de ce qui lui a déjà été présenté (McKeown & al., 1985a), (Acker & al., 1991), (Suthers, 1993). Les points de vue sont utilisés pour sélectionner les objets ou les connaissances à présenter dans l'explication, compte-tenu des caractéristiques de ces objets ou de ces connaissances, i.e. compte-tenu de leurs types ou des relations qui y sont attachées. Comme le type d'un objet ou d'une connaissance peut s'exprimer lui aussi à l'aide d'une relation, les types de points de vue correspondent souvent à des types de relations. De nombreuses catégorisations de ces types ont été proposées (Suthers, 1989),

(Acker & al., 1991). Voici quelques catégories de haut niveau pour les points de vue ou les relations et donc pour les vues qui en découlent : taxonomique, attributif, comparatif, spatial, temporel, fonctionnel, procédural, causal et de perception.

2. Par ailleurs, lors de la modélisation des connaissances, des relations de ces différents types doivent être représentées explicitement afin de permettre ultérieurement les générations d'explication par le SBC. La littérature concernant ***la préparation des explications au cours de l'AC*** met donc l'accent sur les types de connaissances et de relations à éliciter. Les types de relations sont les mêmes que ceux utilisés pour la génération d'explications. Parmi les types de connaissances nécessaires, outre les connaissances nécessaires à la résolution de problèmes, on peut noter (Martin, 1993a, 1993b, 1994) :
   1) des connaissances "de documentation" sur ces connaissances nécessaires à la résolution de problèmes ;
   2) des connaissances sur l'opportunité d'expliquer les connaissances de résolution de problèmes ou de documentation, e.g. à quels types d'utilisateurs et dans quelles conditions ;
   3) des connaissances sur le raisonnement explicatif ou sur la coopération avec l'utilisateur ;
   4) des modèles sur les types d'utilisateurs (leurs buts, leurs connaissances, etc.).

2.4 La création et la réutilisation d'ontologies

Rappelons qu'une ontologie est un ensemble de termes définis ou bien atomiques qui peut être structuré sous forme de taxonomie et/ou de modules. Les modules pouvant être structurés par des relations d'inclusion et des passerelles peuvent exister entre les termes de différents modules. Une ontologie permet de définir les termes utilisables pour représenter des connaissances, les relations entre ces termes et leur contraintes d'utilisation. Chaque terme peut être vu comme une catégorie de connaissances à instancier et utiliser.

Nous avons considéré qu'une ontologie incluant une taxinomie peut être considérée comme une collection structurée de ***points de vue***. Ces points de vue permettent de guider l'élicitation de connaissances et leur organisation, et sont des critères permettant de retrouver ces connaissances. Une ontologie peut ne décrire que certains types de connaissances, e.g. tâches, méthodes, instances de tâche, inférences, connaissances de domaine ou encore connaissances de représentation. Elle peut aussi réunir ces divers types et les structurer, comme le fait par exemple l'ontologie proposée par défaut par CGKAT, que l'utilisateur peut spécialiser ou compléter.

Une ontologie peut aussi permettre à plusieurs agents (systèmes, processus, etc.) de ***communiquer des connaissances*** et non uniquement des données dont l'organisation est prédéterminée. Pour cela, les ontologies des agents doivent avoir des parties communes. Des traductions de données dans différentes représentations peuvent alors être effectuées. C'est l'un des buts du "Knowledge Sharing Effort" (Gruber, 1992) (Gruber, 1994) pour la construction de ses théories (notamment la "frame ontology"). On peut trouver dans (Takeda & al., 1994) et (Wiederhold, 1994) quelques techniques pour traiter les problèmes d'intégration des ontologies de différents agents lors de leurs communications. La figure suivante extraite de (Tate, 1994) résume quelques applications des ontologies.

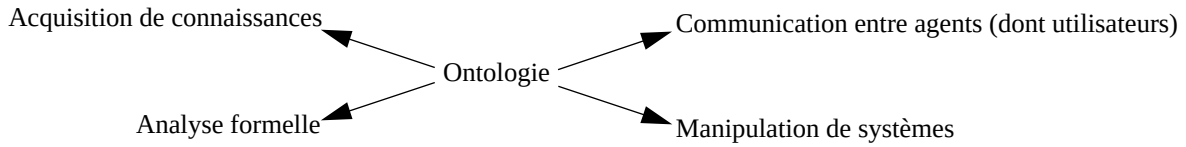


Figure 2.12: *Quelques applications des ontologies (Tate, 1994).*

Construire une ontologie est un travail difficile. Afin de simplifier sa tâche et être guidé, un cogniticien peut reprendre une ou plusieurs ontologies existantes. Il peut s'agir d'ontologies destinées à être réutilisées ou d'ontologies d'applications ou de domaines similaires. S'il y a plusieurs ontologies, le cogniticien doit d'abord effectuer un travail d'***intégration*** ou, dans le cas de modules destinés à être réutilisés, d'*assemblage*. Puis il lui faut compléter le résultat, et surtout dans le cas d'ontologies qui n'ont pas été conçues pour être réutilisables, l'***adapter*** ou le corriger.

Un cogniticien peut ***intégrer des ontologies*** provenant d'applications proches de la sienne ou travaillant sur le même domaine, ou des ontologies relatives à une même expertise mais modélisant les visions de différents experts ou encore modélisées par différents cogniticiens. L'intégration d'ontologies est bien sûr un problème très complexe, et difficilement automatisable : par delà, les problèmes de terminologie et de langages de représentation, les connaissances de différentes ontologies peuvent être réellement contradictoires. Il faut alors généralement consulter les experts pour pouvoir contextualiser ou raffiner ces connaissances.
Pour l'intégration de taxonomies possédant des définitions en langage naturel, Knight & Luk (1994) proposent des méthodes pour apparier les concepts de ces différentes taxonomies compte-tenu de 1) leurs noms, 2) leurs définitions, 3) leurs supertypes et sous-types.
Notons également les méthodes de classification conceptuelle (Michaski & al., 1981), telle MSG (Mineau & Allouche, 1995), qui permettent d'interclasser des concepts provenant de différentes sources et d'exhiber certains "concepts naturels" sous-jacents.
Lehmann & Cohn (1994) établissent une hiérarchie (ordre partiel) de différents cas possibles dans l'appariement de concepts (de l'appariement le plus exact au plus faux). La sûreté (ou inversement le risque) d'appariements peut ainsi être mesurée.
Garcia (1995, 1996a, 1996b) étudie différents cas de partage et d'intégration de taxonomies pour modéliser les connaissances d'experts différents et pouvant avoir des spécialités différentes.
Le problème de l'intégration de plusieurs taxinomies d'experts différents est abordé dans Dieng (1996).

Il est intéressant pour un cogniticien de pouvoir réutiliser une ontologie conçue de façon à être facilement ***réutilisable***. Il est également intéressant qu'il crée à son tour une ontologie réutilisable et donc compréhensible, explicite, non ambiguë, cohérente et extensible. Nous présenterons ci-après quelques principes de construction favorisant la réutilisabilité.

Par ailleurs, il est important que le cogniticien dispose d'***outils*** lui permettant de gérer son ontologie, c'est-à-dire de la visualiser, de l'éditer, de la documenter et d'effectuer divers types de recherche et de vérifications, comme c'est par exemple le cas dans Ontolingua, CODE4 (Skuce & Lethbridge, 1995) et CGKAT. Enfin, la construction d'une taxinomie peut être grandement facilitée par des outils de classification travaillant sur les définitions associées aux termes, e.g. C-CHIC (Leclère, 1995) (Leclère, 1996) pour le formalisme des GCs.

Parmi les ontologies non destinées à une application mais uniquement conçues pour être réutilisées, on trouve :

- des ontologies de haut niveau définissant et généralement spécialisant des types de concepts comme Entity, Collection, Situation, Process, Event, Property, Value, Measure, e.g. le Situation Data Model (Tepfenhart, 1992), Ontos (Carlson & Niremburg, 1990), le Generalized Upper Model[1] (Bateman & al., 1994) (Bateman & al, 1996) et le standard ISO/IEC 11179 ;
- des listes ou parfois des taxonomies de types de relations, e.g. dans (Sowa, 1984), (Myaeng & al., 1992), CYC (Lenat & Guha, 1990a), (Vilnat, 1992) ;
- des modules de définitions et de règles concernant divers aspects fondamentaux en représentation des connaissances (collection, liste, frame, topologie, méréologie, quantités et dimensions physiques, etc.), e.g les théories fournies par le "Knowledge Sharing Effort" (Gruber, 1992) (Gruber, 1994) ;

1. Le "Generalized Upper Model" est une évolution du "PENMAN upper Model" (Bateman, 1990).

- de grandes taxonomies de catégories conceptuelles, comme celles de WordNet (Miller & al., 1990) (90.000 catégories reliées 120.000 mots du langage naturel) et celle de CYC (40.000 catégories) ;
- CYC : à terme, cette base de connaissances doit ou devait rassembler une très grande quantité de connaissances (de l'ordre de 10^8 axiomes) sur les croyances ou connaissances de sens commun d'un américain moyen, le but étant de fournir des connaissances indispensables à des systèmes d'analyse de langage naturel, de traduction ou de résolution de problèmes, pour affronter une grande palette de situations et ainsi ne pas être limités à des domaines très réduits.

2.4.1 Quelques principes pour construire une ontologie partageable ou réutilisable

Gruber (1992) donne quelques critères que doit suivre une ontologie (ici une théorie) destinée à être partagée ou réutilisée par des agents (cogniticiens, systèmes, processus, etc.) :
1) Clarté : une ontologie doit contenir des définitions formelles, déclaratives, objectives et si possible par conditions nécessaires et suffisantes. Ces définitions doivent être documentées en langage naturel.
2) Cohérence : les définitions doivent être consistantes entre elles et les inférences doivent être consistantes avec les définitions qu'elle utilisent.
3) Extensibilité : les définitions et la représentation doivent permettre une extension monotone de l'ontologie, i.e. sans révision des définitions existantes.
4) Minimisation des biais dûs à la représentation ou à l'implémentation.
5) Minimisation des engagements ontologiques : le nombre des hypothèses faites sur le monde modélisé et donc de termes introduits pour supporter un processus, doit être minimum. Ainsi l'agent qui s'engage à une ontologie a plus de liberté pour la spécialiser ou l'instancier, et moins de termes à utiliser pour communiquer avec d'autres agents.

Gruber (1992) illustre l'application de ces critères sur la construction d'une ontologie pour "les quantités physiques, les unités de mesure et une algèbre pour les modèles d'ingénierie" et d'une ontologie pour "partager des données bibliographiques".

Ces critères sont donnés pour des théories mais semblent également valables pour des ontologies représentées sous forme de taxonomies. En effet, une taxonomie peut être vue comme un ensemble de définitions, qu'elle soit composée de termes définis ou atomiques (une condition nécessaire et suffisante d'appartenance à un type X est une condition nécessaire d'appartenance à un sous-type de X, qu'il soit atomique ou non). Les principes présentés ci-après s'appliquent plus aux taxonomies.

2.4.1.1 Expliciter les engagements ontologiques

Pour qu'une ontologie soit utilisée, réutilisée ou construite coopérativement par plusieurs cogniticiens, les hypothèses ou engagements sur lesquels est fondée cette ontologie doivent avoir été compris par ses utilisateurs ou constructeurs. Or, Skuce (1995) note que dans les ontologies actuelles, même celles conçues pour être réutilisées, ces hypothèses sont peu et rarement explicitées. Comme d'autres chercheurs, Guarino & al (1994) pensent que la construction d'une ontologie, et tout particulièrement si elle est coopérative, doit consister tout d'abord à fixer les *engagements* ontologiques généraux et les *catégories* de haut niveau (appellations et significations) de l'ontologie, puis à raffiner ou formaliser ces engagements et les définitions de ces catégories, et enfin spécialiser ces catégories.

Skuce propose, pour expliciter ces hypothèses ou engagements ontologiques, de procéder par niveau. Il présente un format pour les quatre premiers niveaux, ces niveaux étant alors à décrire en langage naturel et les niveaux suivants pouvant être décrits formellement.
- Le niveau 0 contient les engagements généraux pour l'ontologie. Les niveaux suivants concerne chaque catégorie.
- Le niveau 1 contient pour la catégorie a) le concept qu'elle représente et la justification de son existence ou de son utilité, b) les noms qui désignent ce concept et comment il va être appelé dans l'ontologie, c) une définition, et d) des exemples.
- Le niveau 2 contient a) une ou plusieurs propriétés nécessaires et/ou suffisantes de la catégorie (et parmi celles-ci, celles qui peuvent servir de définitions), et b) les dimensions selon lesquelles la catégorie est partitionnée en sous-catégories (une dimension est définie avec une propriété ou un prédicat).
- Le niveau 3 contient des propriétés plus "faibles" de la catégorie : propriétés typiques, propriétés optionnelles, propriétés fausses pour la catégorie, etc.

Skuce donne quelques exemples d'engagements généraux qui peuvent être pris pour construire une ontologie, ou par rapport auxquels les choix de construction doivent être explicités. Certains de ces engagement sont vérifiés par CODE4 (Skuce & Lethbridge, 1995), l'outil de construction d'ontologie de Skuce.
- *Une et une seule catégorie qui inclut toutes les autres*. C'est généralement le cas dans la plupart des ontologie car cela en simplifie la définition et l'utilisation. Skuce appelle cette catégorie "thing".
- *Pas d'entités primitives mais seulement des propriétés primitives*. Il n'y a ainsi qu'un seul type de primitive ce qui simplifie les choix ontologiques[1] (e.g. "propriété" et "rouge" sont des propriétés primitives, tandis que "propriété physique" et "entité rouge" doivent pouvoir se décomposer en propriétés primitives[2]). Skuce note que le problème le plus complexe et le plus crucial dans la création d'une ontologie est probablement la création d'un ensemble de propriétés primitives (Skuce en définit une trentaine dans son ontologie de haut niveau). Les autres catégories peuvent être définies à partir de ces propriétés, et dans le cas d'une construction coopérative, il est plus simple d'arriver à un consensus sur les primitives que sur leurs combinaisons. De nombreux choix sont à faire dans la structuration des primitives. Par exemple, faut-il permettre de définir une primitive comme sous-type d'un type non primitif (e.g. "rouge" comme sous-type de "couleur" qui est sous-type de "propriété physique") ?
- *Une liste des types de définitions utilisables (par ordre de préférence)*. Une définition peut être effectuée de plusieurs manières : par intension (conditions nécessaires et/ou suffisantes), par extension (liste exhaustive des instances du type à définir), par prototypes (propriétés typiques), par équivalence avec un autre type, par analogie, etc. Les types de définition utilisables pour construire une ontologie doivent être fixés.
- *A part la catégorie de plus haut niveau, toutes les catégories doivent avoir une ou plusieurs catégories parentes dont elles héritent des propriétés et dont elles diffèrent par l'introduction ou le raffinement d'au moins une propriété ou en posant une exception sur l'héritage d'au moins une propriété.*
- *Un terme d'une ontologie est soit un type, soit une instance de type.*
- *Une instance ne peut être spécialisée.*

Notons que l'utilisation d'un formalisme de représentation fixe certains de ces engagements. Par exemple, dans le formalisme de base des GCs, il y a une seule catégorie de plus haut niveau et pas de type de second ordre (types ayant pour instance d'autres types). De plus, la relation de spécialisation entre les GCs, et ainsi entre les types définis par condition nécessaires et suffisantes exprimées avec des GCs, est bien définie. Enfin, les définitions de types n'introduisent pas d'exceptions.

---

1. Notons également que cela uniformise les choix ontologiques, ce qui permet ultérieurement de mieux pouvoir décomposer les connaissances, donc les comparer, et ainsi les rechercher ou d'effectuer des inférences.

2. Skuce décompose "entité rouge" en "entité" et "rouge", et "entité" en "thing" et "existante".

2.4.1.2 Créer une ontologie de termes primitifs puis une ontologie formelle

Nous résumons ici la vision de Bachimont (1996) et de Bouaud & al. (1994) sur la création d'ontologies. Ces auteurs insistent sur la difficulté de réutiliser l'ontologie d'une application pour une application proche ou d'un domaine proche, et ce pour deux raisons :
1) le contenu et l'organisation de l'ontologie d'une application sont pertinents pour une tâche précise, aussi l'adaptation et la complétion des ces connaissances pour d'autres tâches n'est pas aisée ;
2) pour comprendre et réutiliser une ontologie, celle-ci doit avoir été construite selon certains principes explicites et l'ontologie doit être documentée, ce qui est rarement le cas. Nous présentons ci-après les principes de construction préconisés par Bachimont (1996) et Bouaud & al. (1994).

2.4.1.2.1 Ontologie de termes primitifs versus ontologie "formelle"

Représenter formellement des connaissances, c'est créer des constructions symboliques dont la signification réside dans la dénotation d'objets du monde. Pour ce faire, les formalismes de représentation reposent sur des règles syntaxiques (qui indiquent comment construire les formules du langage à partir de termes (prédicats) primitifs) auxquelles sont associées des règles de composition sémantique (qui indiquent la signification globale des formules compte-tenu de celles des termes primitifs).

Cela n'est possible que si 1) les termes primitifs sont définis indépendamment les uns des autres, et 2) un terme primitif doit avoir toujours le même sens quel que soit le contexte de son utilisation. Ainsi, les combinaisons de ces termes auront des sens uniques et non contextuels.

Modéliser un domaine consiste à décider quelles formules sont vraies et utiles dans le domaine. Pour cela, il faut déterminer quels sont les termes primitifs de modélisation du domaine et leur signification. Bachimont nomme "*ontologie régionale*" cet ensemble de termes primitifs et leur signification associée, et nomme "*ontologie formelle*", l'ensemble des termes primitifs ou non avec lesquels la modélisation formelle du domaine est effectuée (les termes non primitifs résultent de la composition des termes primitifs, e.g. le terme "entité_colorée" peut se définir à partir des termes "entité" et "couleur").

Pour Bachimont, construire l'ontologie régionale consiste à prendre un "engagement sémantique" sur le sens des termes primitifs, c'est-à-dire à fixer leur "sens linguistique", en construisant un système de différences entre ces termes (Rastier & al, 1994). Construire l'ontologie formelle, consiste à prendre un "engagement ontologique" (Guarino & al, 1994)[1] sur les termes primitifs ou non de l'ontologie, en décrivant la signification de ces termes avec une sémantique formelle, i.e une sémantique logique de type référentielle (un terme se comprend comme désignant des individus existants dans l'univers de référence retenu pour réaliser le modèle formel).

Les principes de construction d'une ontologie régionale, détaillés dans la section suivante, conduisent à un arbre taxinomique. Dans l'ontologie formelle, cette structure d'arbre peut ne pas être conservée puisque des termes peuvent être combinés pour définir d'autres termes.

2.4.1.2.2 La construction d'une ontologie de termes primitifs ("ontologie régionale")

Si le domaine est formel, les termes du domaine sont non contextuels (ils sont définis de manière analytique), les termes primitifs existent et il suffit de les trouver. Par contre, si le domaine est non formel, des termes sont non contextuels, les termes primitifs n'existent pas et il faut les créer par une modélisation particulière. Par exemple, en médecine, le terme "fonction diurétique" est contextuel : il

1. A l'aide d'une logique modale et d'une théorie méréo-topologique, Guarino & al. (1994) définissent 1) un "engagement ontologique" pour un langage, 2) des distinctions élémentaires parmi des prédicats dans ce langage, et à l'aide de ces distinctions, 3) la notion "d'engagement ontologique bien fondé". Les distinctions sont "categorial" (e.g. pour un objet physique, un événement), "sub-categorial" ("sortal" et "non-sortal"), "sortal" ("substantial" (e.g. pour une pomme ou une couleur) et "role" (e.g. pour un étudiant)) et "non-sortal" ("mass-like" e.g. pour du bois et "characterizing" e.g. pour des études ou le rouge).

dénote une indication thérapeutique lorsqu'il est appliqué à un médicament, et une fonction biochimique lorsqu'il est appliqué à une molécule biochimique.

Pour dégager à partir de l'expression linguistique des connaissances du domaine, des termes primitifs possédant une signification indépendante du contexte, il faut fixer un contexte de référence. Pour créer un SBC, le contexte privilégié est le point de vue de la tâche.

Le sens linguistique se caractérise comme un système de différences entre des termes (Rastier & al, 1994). Dégager des termes primitifs, c'est fixer un système de différences particulier dans lequel les termes se voient attribuer un sens unique et non contextuel. Deux termes se distinguent dans la mesure où ils sont identiques à quelques détails près. Un système de différences se construit donc en utilisant une relation taxinomique de type "est-un", et en fixant dans la hiérarchie de subsomption qui en résulte, les différences entre tout noeud et tous les autres noeuds. Cela peut se faire simplement en fixant la signification de la position d'un noeud par rapport à ses pères et frères dans la hiérarchie. Pour cela, quatre principes différentiels sont à appliquer :
1) expliciter en quoi un terme est semblable à son terme père dans la hiérarchie (son "genus"), i.e. en quoi être classé sous ce "genus" est une condition nécessaire pour le terme ;
2) expliciter (par une condition suffisante) en quoi un terme est différent de son "genus" (i.e. expliciter le "differentia") ;
3) expliciter en quoi un terme est différent de ses frères (pour cela, deux frères doivent s'exclure sur les valeurs d'au moins une propriété, ce qui fait que la hiérarchie a une structure d'arbre) ;
4) expliciter sur quelle(s) propriété(s) un terme va pouvoir être comparé à chacun de ses frères.

Bouaud & al (1994) notent à propos de ce dernier principe, qu'il est souhaitable mais non indispensable qu'il n'y ait qu'une seule propriété pour comparer tous les frères, c'est-à-dire que la spécialisation du terme père ne se fasse que sur un seul axe sémantique. Cela est souhaitable pour des raisons d'efficacité dans l'exploitation de la hiérarchie (recherche, classification, etc.) et pour éviter de réutiliser le même critère de différentiation plus bas dans la hiérarchie, et donc d'éviter de confondre des types ou d'avoir des définitions redondantes. Ils regrettent de n'avoir trouvé aucune méthode pour caractériser des critères de différenciation pertinents ni pour déterminer l'ordre dans lequel ils doivent être appliqués.

Les explicitations recommandées par les quatre principes ci-dessus, principalement le troisième et le quatrième, ne peuvent pas toujours être effectuées de manière formelle. Il s'agit alors de documenter les termes de la hiérarchies afin de préciser leur sens, et ainsi faciliter l'exploitation manuelle de la hiérarchie, sa construction ou son exploitation par plusieurs personnes, et sa réutilisation.

2.5 Conclusion

Les premiers systèmes experts étaient construits en codant très rapidement les connaissances des experts sous forme de règles ou de programmes. Le faible niveau d'abstraction et d'organisation des connaissances rendait difficile la construction, la validation et la maintenance du système. De plus, de tels systèmes ne pouvaient fournir que des explications limitées : les étapes du raisonnement pouvaient être indiquées mais ni la stratégie employée, ni les fondements des connaissances utilisées, ne pouvaient être fournis puisqu'ils n'avaient pas été explicités.

Depuis, l'évolution des méthodologies et outils d'AC a principalement consisté à permettre l'extraction et la représentation de connaissances explicites et à guider les cogniticiens dans cette voie en leur fournissant des primitives de représentation à utiliser et des modèles à suivre. Ces primitives et ces modèles furent tout d'abord intégrés dans les outils et exploités pour guider l'extraction et la validation des connaissances. Cependant, celles-ci sont alors limitées puisque aucune adaptation au domaine n'est possible. Actuellement, les outils généraux d'AC fournissent des langages de représentation plus génériques ainsi qu'une bibliothèque de modèles de tâches génériques qui peut être étendue par les utilisateurs. Certains de ces outils, e.g. CUE et Protégé-II, peuvent exploiter des modèles de tâches génériques, ou des modèles de tâches créés par l'utilisateur, pour guider l'extraction et la modélisation de connaissances du domaine. Toutefois, dans les outils d'AC,
1) les langages permettant de construire les modèles de tâches ne sont pas encore suffisamment génériques : ils intègrent des distinctions conceptuelles qui pourraient être offertes dans une ontologie afin de pouvoir être spécialisées ou complétées ;
2) seules des ontologies de tâches ou de méthodes sont généralement offertes, mais pas d'ontologie de haut niveau[1], ni d'ontologie de moyen niveau (e.g. une grande taxonomie de types de concepts du langage naturel). Or ces ontologies pourraient guider le cogniticien dans l'extraction et l'organisation de ses connaissances, et être exploitées par le système pour faciliter et vérifier ces tâches ;
3) les techniques avancées de recherche et de structuration d'information (documents structurés, générations de vues dans les systèmes hypertextes, etc.) ne sont pas ou peu exploitées, par exemple pour la documentation, l'organisation, la recherche et la présentation des connaissances du système.

C'est sur ces trois points que se situe l'originalité de CGKAT par rapport aux outils d'AC généraux actuels. CGKAT exploite :
1) un formalisme de représentation de connaissances générique et adapté pour comparer et rechercher de connaissances,
2) la base générale de connaissances terminologique WordNet et une ontologie de haut niveau synthétisant et organisant des distinctions conceptuelles (parfois accompagnées de définitions ou de modèles) provenant de différents travaux en acquisition des connaissances, en représentation des connaisances,
3) un éditeur de documents structurés et hypertextes.
Par contre, CGKAT n'inclut pas de moteur d'inférence permettant de vérifier ou d'exécuter certaines représentations de connaissances. Des vérifications sont cependant effectuées grâce à certaines contraintes associées aux types de concepts et aux types de relations, e.g. les signatures des relations ou les liens d'exclusion entre les types.

Nous faisons dans le prochain chapitre un état de l'art des techniques de structuration, recherche et présentation d'informations (documents ou éléments de documents). Certaines de ces techniques rejoignent celles portant sur la représentation et l'organisation de connaissances, d'autres sont complémentaires.

1. Cue (Heijst & Schreiber, 1994) permet l'utilisation des théories fournies par le "Knowledge Sharing Effort" mais ce sont plutôt des ontologies de représentation ou des ontologies d'application que des ontologies définissant et interclassant des types de concepts et de relations primitifs.

Chapitre 3 Structuration, recherche et présentation d'informations

3 Sommaire

Rappelons que dans ce rapport, la *recherche d'informations* (RI) ne désigne pas seulement la recherche de documents, mais la recherche de n'importe quel *élément de document (ED)*, qu'il soit structuré (e.g. une section, un graphique ou encore un document entier) ou non structuré (e.g. une portion de texte, un symbole, une image). Les documents et donc les EDs peuvent être multi-media. Pour alléger l'écriture, le terme "information" sera généralement utilisé comme un synonyme de "ED" et le terme "connaissances" pourra être employé seul pour désigner "une représentation de connaissances".

La RI peut se faire par requête ou bien par navigation (butinage ou "browsing" en anglais), c'est-à-dire l'exploration libre d'une structure telle qu'une hiérarchie ou un graphe. Si la navigation est possible sur des liens entre informations ou connaissances, nous appelons ces liens des liens "hypertextes". Les liens "hypertextes" peuvent être exploités aussi bien pour la navigation que par les requêtes. Ces liens peuvent être qualifiés d'hypermedia s'ils relient des EDs utilisant des media différents (e.g. texte, graphique, image, son). Dans la littérature sur les systèmes hypertextes, les termes de "noeud" ou de "document" sont utilisés au lieu de "ED", mais dans le même sens : un document est un noeud, et un noeud peut être un document entier ou une partie de document.

3.1 Les différents types de structuration et de recherche

3.1.1 Les différents types de recherche

Les techniques de structuration et de recherche d'informations sont nombreuses et diversifiées, et donc difficiles à comparer et à classer. Nous donnons ici quelques critères de comparaison des types de recherche et donc aussi, de manière indirecte, des types de structuration qu'elles exploitent. Nous détaillons en 3.1.2. différentes méthodes de recherche et de structuration.

3.1.1.1 Recherches séquentielles ou indexées

Pour retrouver une information parmi un ensemble d'informations, la RI peut s'effectuer :
1) par une recherche *séquentielle* dans l'ensemble des informations, comme c'est le cas par exemple pour les recherches de mots ou d'expressions régulières dans un texte ;
2) en exploitant une *indexation* de l'ensemble des informations, c'est-à-dire leur découpage en plusieurs éléments et l'indexation de chacun de ces éléments *et/ou* de leurs inter-relations ; un index (ou "descripteur") d'un élément ou d'une relation est une *représentation* de certaines de ses caractéristiques sous une forme exploitable par le système de recherche d'informations (SRI). Les noms, les types et les attributs sont des descripteurs simples. Les descripteurs peuvent être à leur tour structurés (i.e. indexés, organisés par diverses relations) afin de permettre des recherches plus aisées par navigation ou requêtes.

Les recherches séquentielles, à moins qu'elles n'effectuent une analyse lexicale, syntaxique ou sémantique sur chaque information, imposent à l'utilisateur de savoir exactement ce qu'il recherche et n'offrent aucun niveau d'abstraction. Notons que si l'utilisateur effectue une recherche en naviguant sur des liens hypertextes non typés, i.e. non indexés, c'est lui qui effectue une recherche séquentielle. Il est alors obligé de procéder par essais/erreurs et si l'ensemble d'informations est grand, il est probable que l'utilisateur sera soumis aux deux problèmes bien connus dans les systèmes hypertextes : la ***désorientation*** (ne pas savoir se diriger dans un réseau hypertexte) et la ***surcharge cognitive*** (l'effort effectué pour maintenir plusieurs tâches en parallèle) (Conklin, 1987).

Un ED peut avoir plusieurs descripteurs et un descripteur peut être formé en organisant des descripteurs de sous-éléments de l'ED. Par exemple, un document peut avoir une table des matières, une liste des figures et une liste des auteurs ; les entrées de ces index sont des représentations de certaines caractéristiques de certains EDs : leur type (e.g. les types Figure et Auteur), leur emplacement (e.g. via un numéro de page ou un lien hypertexte), etc.

Notons que tout descripteur forme lui-même un ED. La navigation peut donc être rendue possible entre des EDs et leurs descripteurs, ainsi que sur les relations entre ces descripteurs. Plus ces relations seront sémantiques et précises, plus les recherches par navigation seront aisées et satisferont le besoin en information de l'utilisateur.

Le stockage, la structuration et la recherche d'EDs peuvent exploiter des techniques classiques, par exemple celles des bases de données, ou des techniques d'intelligence artificielle comme celles des bases de connaissances. Une représentation du contenu des EDs avec un langage de représentation des connaissance offre de nombreuse possibilités de recherche mais est beaucoup plus difficile à acquérir de manière automatique.

3.1.1.2 Liens statiques ou virtuels, navigation ou requêtes

Un lien (ou "référence") peut relier un ED à un ou plusieurs EDs. Un lien *hypertexte* peut être *activé*, généralement en cliquant sur une zone sensible (ou *bouton*) d'un ED source du lien. Les EDs destination sont alors affichés, accompagnés des EDs dans lesquels ils sont imbriqués ou qu'ils contiennent, i.e. ceux avec qui ils entretiennent des *liens de composition*. Il est ainsi possible de naviguer dans le réseau hypertexte et de consulter le contenu de différents EDs. Lorsqu'un ED est source de plusieurs liens hypertextes, un menu de sélection doit être offert. Certains systèmes hypertextes permettent de retrouver la source d'un lien hypertexte, e.g. Thot (Quint & Vatton, 1992). Le créateur d'un lien est un "*auteur*", celui qui l'active est un "*lecteur*".

Un lien est "*statique*" (ou extensionnel) si les EDs référencés (EDs destination du lien) sont fixés. Il est "*virtuel*" (ou intentionnel ou inférentiel) si la liste ou la composition des EDs référencés est calculée au moment de l'activation de la source du lien (DeRose, 1989).

Un lien virtuel correspond donc à l'exécution d'une *requête prédéfinie*. Nous utilisons ici le terme "requête" au sens large car il peut ou pourrait s'agir d'un programme complexe prenant en compte un modèle des connaissances de l'utilisateur, de la tâche en cours, des chemins de liens visités, etc. Le déclenchement de la requête peut être transparent pour l'utilisateur qui peut alors naviguer sur un lien virtuel de la même manière que sur un lien statique, i.e. par activation de la source du lien. La requête prédéfinie peut également être explicitement montrée à l'utilisateur qui peut alors la lancer ou pas.

La liste des EDs retrouvés ou générés par une requête forme un nouvel ED. Cet ED est dit "*virtuel*" si, lorsqu'il est édité, les EDs sources qui ont servi à le générer sont également modifiés, ou inversement, si la modification des EDs sources entraîne la modification de l'ED virtuel. Cet ED peut parfois être conservé, mais un autre ED sera généré lors d'une nouvelle activation du lien virtuel. Grâce à de tels liens, une modification des informations (e.g. l'ajout d'un nouvel ED) est automatiquement traduite dans la structure d'un réseau hypertexte.

Les "liens mot-clé" de Conklin (1987) sont un exemple de liens virtuels : l'activation d'un "lien mot-clé" mène vers tous les documents contenant ce mot-clé.

La recherche par requête est adaptée si l'utilisateur a une idée précise de ce qu'il cherche et connaît bien le langage d'indexation du système. Sinon, la recherche par navigation est plus avantageuse car il est plus facile de reconnaître quelque chose d'intéressant que de le décrire. Toutefois, et particulièrement si les liens et les EDs ne sont pas sémantiquement typés, la navigation hypertexte conduit souvent aux phénomènes de désorientation et de surcharge cognitive. Il est donc intéressant

de pouvoir composer ces deux types de recherche pour trouver des informations et compenser leurs inconvénients respectifs. Par exemple, une requête peut permettre de réduire l'espace de navigation ou de donner accès à de nombreux liens hypertextes intéressants (Nanard & al., 1993a, 1993b). Cependant, très peu de systèmes permettent de composer réellement les résultats des requêtes et de la navigation. MacWeb (Nanard & al., 1993a, 1993b) est en cela une exception et utilise pour atteindre ce but des documents virtuels. C'est également ainsi que nous procédons dans CGKAT.

3.1.1.3 Recherches exactes ou approchées

L'ensemble des éléments indexés et que l'on peut rechercher s'appelle le *corpus*. Suivant le nombre et la taille des éléments du corpus, et donc la précision relative des indexations, la RI peut avoir deux buts sensiblement différents :
1) retrouver des éléments de document *qui contiennent (entre autres) des informations les plus proches possible* de celles recherchées ;
2) retrouver *s'ils existent* les éléments de document *qui correspondent exactement* à une connaissance recherchée.

Par exemple, une recherche exacte de la valeur du dollar australien en francs français fournira une réponse courte contenant cette valeur, tandis qu'avec une recherche approchée, les (éléments de) documents ou les noms des documents pouvant contenir la réponse à cette question seront fournis et l'utilisateur devra effectuer des recherches à l'intérieur de ces (éléments de) documents. Plus les documents ou les éléments de document indexés sont petits, plus la recherche est "exacte".

La recherche approchée est typique des systèmes de recherche documentaire. Dans ces systèmes, des documents plus ou moins gros sont recherchés par requête (e.g. dans les "Systèmes de Recherche d'Information" au sens classique du terme) ou par navigation (e.g. dans les systèmes hypertextes contenant de nombreux documents : "large-scaled hypertext systems").

La recherche exacte est plutôt typique de l'acquisition de connaissances où de petits éléments de documents sont indexés par une connaissance, ou inversement, la documentent[1]. Dans ce genre de cas, la RI peut être effectuée simplement via une recherche de connaissances dans l'ensemble des indexations, e.g. par navigation et/ou avec un système de question/réponse : CGKAT procède ainsi. Une telle recherche ne suffit pas en recherche documentaire car le descripteur d'un élément ne représente qu'une partie souvent faible de son contenu, c'est-à-dire seulement ses informations les plus *représentatives*. L'utilisateur doit donc consulter les documents fournis pour vérifier qu'il sont bien "en rapport" avec son besoin d'information et obtenir des réponses à ses questions.

3.1.1.3.1 Les critères de précision et de rappel pour la recherche approchée

Dans les systèmes de recherche documentaire par requête, l'adéquation entre le *besoin en information* de l'utilisateur et les *documents à présenter pour répondre à ce besoin*, est mesurée selon des critères de ***précision*** et de ***rappel***. Le besoin en information est partiellement connu via la requête et éventuellement aussi en utilisant un modèle de l'utilisateur (Defude, 1984).

Dans certains systèmes hypertextes, le besoin de l'utilisateur est évalué grâce à un modèle utilisateur ou compte-tenu de la tâche en cours et des éléments déjà visités. Des relations contextuelles peuvent alors être créées et les critères de précision et de rappel peuvent alors éventuellement être pris en compte.

1. Un certain "contexte de compréhension" peut éventuellement être associé à chacun de ces petits éléments ; ce contexte sera également présenté lorsque l'élément sera retrouvé.

Voici un extrait de (Chevallet, 1994) qui définit ces critères pour un "Système de Recherche d'Information" au sens classique du terme :

> Le *rappel* mesure le taux de documents *retrouvés et pertinents* par rapports aux documents *pertinents*. Plus ce taux est proche de la valeur 1, moins nombreux sont les documents pertinents que le système passe sous *silence*. A l'opposé, la *précision* mesure le taux de documents *retrouvés et pertinents* par rapport à l'ensemble des documents *effectivement retrouvés*. Lorsque ce taux est proche de la valeur 1, l'ensemble des documents non pertinents retrouvés (le bruit) est réduit.
>
> Pour un système idéal, ces deux caractéristiques ont la valeur 1. Mais concrètement, l'optimisation *simultanée* de ces deux critères est très difficile. Dans le but d'améliorer la qualité des réponses obtenues, ces systèmes utilisent des connaissances exprimées le plus souvent sous la forme d'un graphe de termes reliés par des relations de synonymie, spécialisation et généralisation. Ce graphe porte le nom de *thésaurus* de termes.

Toujours pour les "Système de Recherche d'Information" au sens classique du terme, Myaeng (1992) note que le principal critère pour juger de la qualité des textes retrouvés est leur "pertinence" vis-à-vis des besoins d'informations de l'utilisateur, et donc 1) le processus de RI est de nature différente des techniques de recherche par requête dans une BC, 2) les limitations des techniques actuelles d'analyse du langage naturel dues à leur dépendance vis-à-vis de domaines particuliers, ne sont pas très préjudiciables. Ce n'est bien sûr pas le cas pour la recherche exacte d'informations, e.g. dans un cadre d'acquisition de connaissances.

3.1.2 Différentes méthodes de recherche

Dans cette section, nous reprenons en les complétant et les restructurant des informations données dans (Amann, 1994).

3.1.2.1 L'aide à la navigation

Une multitude de mécanismes ont été proposés pour aider la navigation et réduire les problèmes de désorientation et de surcharge cognitive. Outre les requêtes, les liens virtuels, la mémorisation et la structuration des EDs visités, voici les idées les plus souvent utilisées.

3.1.2.1.1 Visualisation partielle de réseaux et/ou d'éléments de documents

Différentes solutions pour une *visualisation globale de parties de réseaux ou d'EDs* (comme sur une carte routière) ont été proposées (Utting & Yankelovich, 1989) (Feiner, 1988). Un principe souvent exploité est que plus un ED est éloigné de l'ED sélectionné (éloigné par le nombre de liens de composition ou de liens hypertextes qui relient ces EDs), moins il doit prendre de place sur l'écran. Le signalement lors d'une visualisation globale des EDs visités et chemins parcourus est également intéressant.

3.1.2.1.2 Les visites guidées

Des chemins de consultations peuvent être définis dans le réseau (Marshall & Irish, 1989) (Zellweger, 1989). Le lecteur doit pouvoir suivre un chemin et s'en échapper lorsqu'il le désire. Un parcours ou l'affichage d'informations peuvent également être programmés, e.g. avec des réseaux de Petri (Stotts & Furuta, 1989). Guinan & Smeaton (1992) proposent un mécanisme qui traduit le résultat d'une requête en une visite guidée (séquence de noeuds), où les noeuds sont triés par rapport à leur importance et aux liens qui les relient.

3.1.2.1.3 Restriction de l'espace de recherche

Souvent un utilisateur n'est intéressé que par les informations liées à certains EDs ou à certaines portions du réseau hypertexte. L'espace de recherche peut être par exemple être restreint a) à certains EDs, b) aux documents ouverts, c) aux EDs déjà visités (i.e. le contexte connu) (Brown, 1988), d) aux EDs accessibles à partir des documents ouverts (i.e. le contexte accessible) (Frei & Steiger, 1992). Cela est intéressant autant pour la navigation que pour les requêtes.

3.1.2.2 Les requêtes sur les attributs

Dans la plupart des systèmes hypertextes, il est possible d'associer des attributs aux noeuds et aux liens. Certains attributs sont gérés par le système : date de création, nom de l'auteur, etc. L'interface d'interrogation est souvent simple et permet de chercher des noeuds par leur nom ou un prédicat sur les valeurs d'attributs (Wiil & Legget, 1992). Lorsqu'un SGBD est utilisé pour stocker les noeuds et les liens, les recherches peuvent être effectuées via le langage de requête du SGBD, e.g. SQL (Schutt & Streitz, 1990). Utiliser un SGBD permet une recherche et une gestion efficace des informations (gestion des accès concurrents, des transactions, des versions, de la distribution des informations, ...).

3.1.2.3 Les requêtes sur la structure d'un réseau (noeuds et liens)

De nombreux systèmes hypertextes permettent de rechercher directement des parties de réseau compte-tenu de leur structure. La plupart des travaux s'appuient sur un SGBD relationnel ou orienté-objet pour le stockage du réseau hypertexte et fournissent des langages similaires à SQL pour l'interrogation des données, e.g. (Gallagher & al., 1990), (Chen & al., 1990), (Fuller & al., 1991), (Marman & Schlageter, 1992). Nous présentons maintenant plus en détail différents langages pour l'interrogation des structures hypertextes.

3.1.2.3.1 L'interrogation par Datalog

Afrati & Koutras (1990) utilisent une représentation des réseaux hypertextes sous forme de tables, ou *relations*, et le langage de requête Datalog (Ullman, 1988) pour la recherche de noeuds et de liens. Datalog est un langage déductif travaillant sur des formules logiques du premier ordre : les noms des relations sont considérés comme des prédicats à partir desquels il est possible de déduire de nouvelles relations. Six tables/relations sont utilisées : Lien, Noeud, Bouton, Region, Lié_à et Attribut (un bouton et une région sont des sous-parties particulières d'un noeud).

Prenons l'exemple d'une base de cas d'accidents de la route, où sont stockés pour chaque accident, sa cause principale et les principales phases de l'accident. La table Noeud stocke tous ces éléments (quel que soit le cas le cas dans lequel ils interviennent), la table Lien stocke les liens entre ces éléments, la table Lié_à enregistre pour chaque lien les éléments qu'il relie, et la table Attribut stocke pour chaque noeud ou lien, son type ou son nom.

De nouvelles tables/relations peuvent être définies avec le langage Datalog. Par exemple :

```
Cause_de_accident (A,C) :- Noeud (A), Noeud (C), Lien (L), Lié_à (A,C,L),
                           Attribut (A,type,accident), Attribut (L,type,cause).
```

Ces relations peuvent être utilisées dans des requêtes. Par exemple, pour rechercher les accidents ayant un même cause donnée : `:- Cause_de_l'accident (A,vitesse).`

Datalog permet des définitions récursives et donc de calculer des fermetures transitive de liens :

```
Phase_de_accident (A,P) :- Noeud (A), Noeud (P), Lien (L), Lié_à (A,P,L),
                           Attribut (L,type,première_phase).
Phase_de_accident (A,P) :- Phase_de_accident (A,P2), Noeud (P), Noeud (P2), Lien (L), Lié_à (P,P2,L),
                           Attribut (L,type,phase_suivante).
```

Exemple d'utilisation : `:- Phase_de_accident(A,P), Cause_de_l'accident (A,vitesse).`

Datalog peut ainsi apparaître comme un langage d'interrogation puissant mais assez lourd à utiliser, et nécessitant donc une interface pour des utilisateurs non spécialisés (Amann, 1994).

3.1.2.3.2 Utilisation d'une logique modale

Le langage de (Beeri & Kornatzky, 1990) utilise des formules d'une extension de la logique modale (Emerson & Halpern, 1985) pour la description de chemins. Un réseau hypertexte est représenté par un graphe dont les noeuds et les liens sont étiquetés par des propositions logiques et des attributs. Ainsi, en reprenant l'exemple de la base de cas d'accidents de la route, la requête suivante trouve toutes les portions de graphes contenant des noeuds de type "Accident" liés à des noeuds de type "cause_de_accident" par un lien de type "cause_de" :

Accident $\wedge$ < $\exists$ forward [cause] link > Cause_de_accident

(trouve chaque noeud de type "Accident" auquel est associé un lien sortant de type "cause" et menant à un noeud de type "Cause_de_accident").

Une requête peut spécifier plusieurs liens à suivre et également créer des liens. Par exemple, supposons que dans la base de cas d'accidents de la route, l'emplacement de chaque phase d'un accident soit enregistré ; la requête suivante crée des liens de type "sur_trajet_de_accident" entre chaque accident et les divers emplacements parcourus jusqu'à cet accident :

Accident $\cup$ < make (sur_trajet_de_accident) forward [première_phase, phase_suivante, emplacement] path > Emplacement_phase_accident

(trouve chaque noeud de type "accident" et les noeuds de type "Emplacement_phase_accident" qui peuvent être atteints depuis ce noeud par un chemin de liens sortants de type "première_phase", "phase_suivante" ou encore "emplacement" ; crée alors des liens de type "sur_trajet_de_accident" entre ces noeuds).

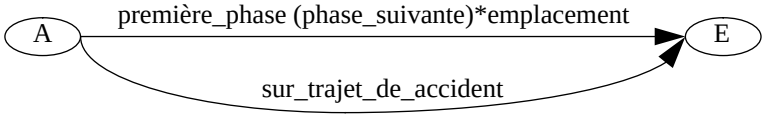
Des quantificateurs peuvent être utilisés dans les requêtes. Par exemple, la requête suivante trouve les noeuds qui contiennent le mot "médicament", et qui sont reliés directement ou indirectement à de nombreux noeuds contenant le mot "vertige" :

"médicament" $\wedge$ < many forward [path] > "vertige".

L'utilisateur peut définir ou redéfinir des quantificateurs adaptés à ses besoins. Ainsi, "many" peut exprimer les deux tiers des chemins retrouvés (c'est la signification par défaut) ou être le résultat d'un calcul plus complexe.

3.1.2.3.3 Le langage graphique GraphLog

Dans le langage graphique GraphLog (Consens & Mendelzon, 1990) une requête est un graphe dont les noeuds sont étiquetés par des variables ou des constantes, et les arcs par des *expressions régulières* (Hopcroft & Ullman, 1979) sur des étiquettes de liens. L'évaluation d'une requête consiste à trouver des graphes de données homéomorphes (Fortune & al., 1980) au graphe de la requête. Par exemple, dans ce langage, la requête pour créer des liens de type "sur_trajet_de_accident" entre chaque accident et les divers emplacements parcourus jusqu'à cet accident, s'exprime de la manière suivante.

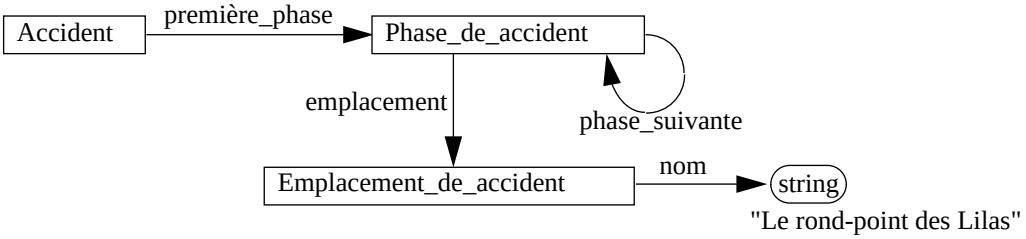


Dans cet exemple, les variables A et E seront remplacées par les extrèmités des chemins retrouvés, et des liens virtuels de type "sur_trajet_de_accident" sont créés entre ces extrèmités.

Consens & Mendelzon (1990) montrent que GraphLog a le même pouvoir d'expression que des programmes Datalog linéaires et stratifiés ou que la logique du premier ordre avec fermeture transitive.

3.1.2.3.4 Le système MORE

Le système MORE (Multimedia Object Retrieval Environment) (Lucarella & al., 1993) fournit un langage graphique pour la construction et l'interrogation de graphes orientés objet (Gyssens & al, 1990) : les arcs des graphes représentent des liens d'héritage ou des liens de composition (mono et multi-valués) entre des classes d'objets. Une requête est un sous-graphe connexe d'un des graphes de description de classe. Au cours de la recherche, ce graphe requête est considéré comme une formule logique et les variables de cette formule sont instanciées ; ces variables contiennent alors les résultats de la requête, i.e. un ensemble d'objets (noeuds) qui contiennent des documents. Des fonctions booléennes permettent de sélectionner des objets sur les valeurs de leurs composants de type atomique (string, integer, float, etc.). Voici un exemple de requête.



3.1.2.3.5 Le modèle GRAM

Amann (1994) note que les expressions régulières constituent un langage assez puissant mais ne permettent pas par exemple 1) de sélectionner des sous-graphes par rapport aux valeurs des noeuds ou des liens, ou 2) de connecter des chemins de deux ensembles de chemins différents, ou de garder seulement des sous-chemins d'un ensemble de chemins.
Aussi présente-t-il GRAM, un modèle de données avec un langage d'interrogation basé sur une algèbre de chemins. Ce langage inclut les opérations suivantes : sélection, projection, renommage, jointure, concaténation, union, intersection, différence. Amann définit formellement ces opérations et les illustre en utilisant une syntaxe à la SQL. Voici quelques exemples pour une base d'accidents de la route.

```
SELECT  *                                   //sélection de tous les noeuds et les liens
FROM    Accident cause Cause_de_accident    //parmi les chemins référés dans la clause FROM
WHERE   vitesse (Cause_de_accident)         //où la vitesse est (instance de) la cause de l'accident

SELECT  *                   //sélection de tous les noeuds et les liens (i.e. pas de projection)
FROM    Accident première_phase (Phase_de_accident phase _suivante)* Phase_de_accident
WHERE   EXISTS emplacement (Phase_de_accident) = "Le rond-point des Lilas"
```

Exemple de "projection" (selon Amann):

```
SELECT  Accident            //sélection des noeuds de type Accident : c'est l'opération de "projection"
FROM    Accident cause Cause_de_accident
WHERE   vitesse (Cause_de_accident)
```

Autre exemple de projection et exemple de renommage (avec sélection sur les noeuds nommés) :

```
SELECT  (Phase_de_accident phase _suivante)* Phase_de_accident          //autre "projection"
FROM    Accident première_phase (Phase_de_accident1 phase _suivante)* Phase_de_accident2
WHERE   EXISTS emplacement (Phase_de_accident2 ) = "Le rond-point des Lilas"
```

Exemple de "jointure" de chemins :

```
SELECT *
FROM    Accident cause Cause_de_accident +
        Accident première_phase (Phase_de_accident phase _suivante)* Phase_de_accident
WHERE   vitesse (Cause_de_accident) AND
        EXISTS emplacement (Phase_de_accident) = "Le rond-point des Lilas"
```

Amann (1994) montre comment ce langage de requête peut être implémenté en utilisant une intégration du système hypertexte Multicard (Rizk & Sauter, 1992) et du SGBD orienté-objet O2. Il montre également comment ce langage pourrait être utilisé pour implémenter différents mécanismes classiques de navigation : liens virtuels, liens inverses, restriction de l'espace de navigation, zoom, etc.

3.1.2.3.6 WebTalk, le langage de script de MacWeb

MacWeb (Nanard & Nanard, 1991) est un système hypertexte où les noeuds et les liens sont typés, et où, comme dans le système Multicard, la manipulation du contenu des noeuds est séparée de la gestion du réseau hypertexte (couche de stockage) par un mécanisme d'ancrage[1]. Les types de noeuds et de liens peuvent être créés explicitement et une approche orientée objet permet de spécifier des liens d'héritage entre les types (classes) et d'associer un comportement aux différents noeuds (objets) sous forme de scripts écrit dans le langage WebTalk. Une originalité de MacWeb est que les relations entre les classes d'objets (i.e. les schémas conceptuels) peuvent être décrites de la même façon que les relations entre objets, c'est-à-dire par le réseau hypertexte.

Le langage WebTalk permet de créer des noeuds (i.e. des éléments de documents) en spécifiant leur structure logique, leur présentation, leur contenu et leur comportement. Plus précisément, un script WebTalk peut comporter :
a) des règles de sélection qui permettent l'extraction d'EDs dans le réseau hypertexte par des expressions de chemins le long des relations d'un schéma conceptuel ;
b) des règles de présentation pour les EDs trouvés ;
c) des règles d'interaction pour associer aux EDs des droits d'accès et des méthodes qui vont propager dans le réseau hypertexte des modifications sur leur contenu (ceci est une façon de gérer des EDs virtuels) ;
d) des instructions de contrôle (test, boucle, etc.) sur ces règles.

Les règles de sélection permettent de sélectionner des chemins à partir :
1) d'un noeud sélectionné (mot-clé "self"),
2) de tous les noeuds d'un réseau (mot-clé "all"),
3) d'un noeud avec un nom donné (mot-clé "named"),
4) des noeuds représentant une classe ou de ses sous-classes (mot-clé "the" suivi du nom de la classe),
5) des noeuds représentant une instance d'une classe ou ses sous-classes (mot-clé "class" suivi du nom de la classe).

Les liens peuvent être suivis dans les deux sens et la fermeture transitive peut être calculée. Voici un exemple très simple de script qui peut être déclenché lorsqu'un noeud de type "Cause_de_accident" est sélectionné :

```
'L'emplacement final des accidents dont la cause est celle sélectionnée est : ';
NAME (self . cause . emplacement_final)   // les noeuds accessibles depuis le noeud sélectionné
                                          //par les liens cause et emplacement_final sont recherchés
```

Voici un exemple de résultat (i.e. le contenu du noeud créé) :

L'emplacement final des accidents dont la cause est celle sélectionnée est :
Le rond-point des Lilas
L'intersection de la N7 avec la RD 28

1. Nous détaillerons ce point et son intérêt pour la RI dans la section 3.2.2.2.1.

3.1.2.4 La recherche approchée d'éléments via l'indexation de leur contenu

Dans une recherche "approchée", le système cherche à augmenter les taux de rappel et de précision (cf 3.1.1.3.1), en privilégiant toutefois l'un des deux, ce qui conduit à des systèmes "orientés vers la précision" ou bien "orientés vers le rappel".

Il existe un grand nombre de méthodes d'indexation de contenu. Citons par exemple la mesure du pouvoir discriminant d'un terme (Yu & Salton, 1977), la mesure basée sur la densité de l'espace des documents (Salton & al., 1973), l'analyse de la sémantique latente (Landauer & Littman, 1991), et les modèles de distribution statistique (Bookstein & Swanson, 1975). Salton (1985) distingue les approches suivantes :

1. Dans le *modèle booléen*, le descripteur d'un ED est un ensemble de mots-clés et la recherche peut s'effectuer avec des prédicats booléens. Par exemple, la requête "accident et (vitesse ou médicament)" donne tous les EDs contenant le mot "accident" et le mot "vitesse" ou le mot "médicament".
2. Dans le *modèle d'espace vectoriel*, le descripteur d'un ED est également un ensemble de mots-clés mais la recherche s'effectue avec un vecteur de mots-clés, e.g. "<accident, vitesse, médicament>". La réponse est une liste d'EDs triée par rapport au nombre de mots-clés partagés entre chaque descripteur de document et le vecteur de la requête.
3. Dans le *modèle probabiliste*, le descripteur d'un ED est également un ensemble de mots-clés mais la recherche prend en compte la probabilité qu'un mot-clé "trouve" des EDs pertinents. Ainsi les documents retrouvés peuvent être triés par rapport à leur pertinence vis-à-vis de la requête.
4. Le *modèle linguistique* prend en considération des relations sémantiques entre les mots-clés, et est basé sur une analyse linguistique du contenu des EDs. Les avantages sont multiples (interrogation en langage semi-naturel, remplacement des mots-clés par des unités linguistiques plus complexes, etc.) mais de nombreuses restrictions sont imposées par les limitations actuelles des outils d'analyse du langage naturel.

Des techniques de ***représentation de connaissances*** sont de plus en plus utilisées en recherche documentaire pour pallier aux insuffisances des techniques classiques. Citons par exemple, DR_LINK (Myaeng, 1992), ELEN (Chevallet, 1992) et RIME (Kheirbek & Chiaramella, 1995) qui sont des systèmes de recherche documentaire exploitant le formalisme des Graphes Conceptuels. Dans ces systèmes, le descripteur d'un document est un graphe conceptuel, le plus souvent construit automatiquement (cf. par exemple (Myaeng & Khoo, 1994)). Des relations entre documents ou parties de documents sont également représentées avec des graphes conceptuels. RIME permet la navigation sur ces relations. La recherche de documents à partir d'une requête (traduite en un graphe conceptuel) exploite principalement la relation de spécialisation calculée entre graphes conceptuels. Afin de satisfaire les critères de rappel et de précision, d'autres calculs doivent également être effectués. Cependant, selon Myaeng (1992), pour une recherche approchée, les limitations actuelles des analyseurs de langage naturel ne sont pas trop préjudiciables car la représentation d'information n'a pas besoin d'être aussi fine que pour une recherche "exacte".

Boy (1991) note qu'une ***base de connaissances*** constitue le meilleur support pour indexer puis retrouver des documents (pour une recherche exacte ou approchée) car
1) une base de connaissances permet d'organiser les descripteurs par diverses relations sémantiques et permet d'effectuer des inférences[1] lors des recherches d'informations ;
2) elle permet de représenter dans les descripteurs diverses informations permettant de focaliser les recherches et de prendre en compte leurs contextes : des informations temporelles ou d'autres dimensions, les types d'utilisateur qui pourraient être concernés par l'information indexée, les types de tâches dans laquelle cette information pourrait être utile, etc. (ainsi par exemple, des liens virtuels peuvent être créés).

1. De manière plus générale, la recherche d'information peut être vue comme de la résolution de problème, et il est alors intéressant d'exploiter pour cela des techniques d'intelligence artificielle.

Notons que la ***base de connaissances peut être plus ou moins séparée des informations indexées*** : dans un système hypertexte typé, la base de connaissance peut être représentée avec les noeuds et les liens du système hypertexte, et les informations indexées se trouvent à l'intérieur des noeuds : un noeud est alors un descripteur de l'ED qu'il contient. Si les descripteurs sont liés entre eux ou avec d'autres connaissances par des liens explicitement représentés et sur lesquels la navigation est possible, l'utilisateur peut rechercher des EDs de cette manière. ***Les descripteurs peuvent être organisés dans un réseau sémantique ou dans d'autres types de structures***.

- Par exemple, Godin & al. (1995) proposent pour indexer des documents d'utiliser un "treillis de Galois élagué" généré à partir de la relation d'indexation entre des documents et les termes utilisés pour les indexer. Le treillis implémente une relation de généralisation/spécialisation sur un ensemble de descripteurs, chacun d'eux étant composé d'un ensemble de termes et de références aux documents indexés par ces termes. La navigation sur ce treillis permet donc à un utilisateur d'élargir ou de spécialiser sa recherche graduellement suivant les sous-ensembles de documents et de termes présents dans le treillis. Une requête avec un ensemble de termes permet de retrouver directement un élément du treillis. Le parcours d'un lien correspond alors à un élargissement (généralisation) ou un raffinement (spécialisation) minimal par rapport à la requête précédente. La construction automatique d'un treillis de Galois peut être vue comme une méthode de classification conceptuelle (Michaski & al., 1981).

- Un descripteur peut :
  a) être un élément d'une base de connaissances, e.g. un type de concept, un concept ou un GC comme dans RIME ou CGKAT, ou
  b) être relié à un élément d'une base de connaissances, comme c'est par exemple le cas dans (Lemaire, 1995) : les mots-clés à l'intérieur d'un texte ou représentant ce texte sont reliés à des termes d'index d'un thésaurus qui eux-mêmes sont reliés à des classes d'objets d'une base d'objets, et les recherches peuvent se faire par navigation sur ces liens et sur ceux dans la base d'objets.
  Dans RIME ou CGKAT, le thésaurus est intégré dans la base de connaissances et constitue l'ontologie de cette base. Les termes d'index sont alors des termes formels : a) ils représentent des concepts ou des types de concepts, b) ils peuvent être formellement définis, c) chaque terme n'a qu'un seul sens (ce qui n'est hélas pas toujours le cas dans tous les thésaurus, où un même terme peut parfois représenter plusieurs concepts ou types de concept), et d) ils peuvent être reliés à des mots dont ils représentent l'un des sens.
  En résumé, ce qui compte pour faciliter la recherche d'informations, ce n'est pas la façon dont les divers liens nécessaires à l'indexation sont implémentés (e.g. un seul réseau hypertexte vs des liens avec une base de connaissances où la navigation est possible), mais 1) la précision de l'indexation (les EDs représentés peuvent être des mots, des phrases ou de grands documents, et les représentations peuvent être plus ou moins précises et "sémantiques"), et 2) la possibilité de compléter les recherches par navigation par des recherches par requête. Mais comme nous l'avons déjà souligné, une représentation précise des connaissances est difficile à automatiser. Par exemple, compte-tenu de l'ambiguïté de la plupart des mots du langage naturel même dans des documents techniques, l'indexation automatique de termes d'un document par des termes formels doit être vérifiée et corrigée par un être humain (si l'indexation doit être précise).

De nombreux systèmes hypertextes permettent de rechercher des informations sur le contenu de noeuds du réseau, indépendamment des liens qui les relient. Frisse (1988) propose une méthode d'indexation qui prend également en compte la structure du réseau. Plus précisément, dans le Medical Handbook (Frisse, 1988), cette méthode suit les liens de composition pour le calcul de la pertinence d'un noeud du réseau. D'autres approches d'indexation, e.g. (Frei & Stieger, 1992), Lucarella (1990), prennent en compte les liens dans des réseaux Bayésiens, i.e. des graphes acycliques dirigés dans lesquels les noeuds représentent des constantes ou des variables propositionnelles, et les liens des relations de dépendance entre les propositions.

3.2 Les systèmes de structuration d'informations

Afin d'être accessible et manipulable via un outil informatique, l'information doit être stockée dans des supports électroniques comme des bases de données ou des documents électroniques. Afin de faciliter ou permettre les ***recherches*** et ***manipulations*** par un logiciel, cette information doit avoir été *découpée* en divers blocs qui sont ***reliés*** entre eux et/ou ***indexés***.

Un ***document structuré*** permet de stocker des informations découpées en divers blocs typés (i.e. des EDs) et connectés par des liens structurels (liens de composition entre EDs). Pour cela, les différents types d'EDs et leurs liens structurels doivent avoir été définis dans un modèle structurel (ou "Définition de Type de Données" (DTD)). Un langage comme SGML (Goldfarb, 1990) permet
1) de définir dans un modèle structurel l'ordre et la structure des types d'EDs à l'aide d'opérateurs d'agrégation, de choix et de séquencement (l'opérateur de séquencement permet de définir un ED formé d'une liste d'éléments identiques) ; des attributs peuvent également être associés aux EDs ;
2) d'isoler et de typer des portions de documents en les délimitant par des marques typées ("tags" en anglais).

Un document structuré peut également permettre de stocker à des références croisées entre éléments, par exemple en permettant l'utilisation d'EDs référence ou d'attributs référence. Par exemple, SGML, ODA (ISO 8613, 1987) et Thot (Quint & Vatton, 1992) permettent la création de documents structurés contenant de tels liens entre EDs.

Un document structuré permet de dissocier trois aspects d'un document :
1) son contenu, i.e. celui de ses EDs,
2) sa structure logique, i.e. l'organisation des EDs, et
3) sa structure physique, i.e. la présentation de ses EDs.
Un même contenu peut ainsi être organisé dans différentes structures logiques pour constituer différents documents.
De plus, différentes présentations peuvent être "appliquées" à un même document : l'application d'une présentation à un document consiste à associer à chacun de ses EDs, un ensemble de caractéristiques visibles ou audibles (e.g. couleur, position, police de caractère). Le langage P de Thot (Quint, 1995) permet de définir dans un modèle de présentation, une présentation par défaut pour les EDs d'un certain type. Plusieurs modèles de présentation peuvent être associés à un modèle structurel : différentes présentations par défaut sont ainsi applicables à un document. Par exemple, une présentation pour l'édition peut permettre de visualiser des notes dans des fenêtres séparées du texte principal tandis que dans une présentation pour l'impression, ces notes peuvent être insérées dans ce texte ou bien apparaître à la fin de celui-ci.

Un ***système hypertexte*** permet à un utilisateur de naviguer sur des liens entre EDs (ces EDs sont alors des EDs de ***documents hypertextes***), et permet généralement d'aider à cette navigation par différentes fonctionnalités (cf. section 3.1.2.1). Souvent, il permet également à l'utilisateur de créer ou modifier le réseau hypertexte, c'est-à-dire les EDs et les liens qui les relient. Ces liens peuvent être des liens structurels. Par exemple, Guide (Brown, 1991) gère les notions d'emboîtement des EDs et d'héritage entre EDs de manière similaire aux systèmes de gestion de documents structurés.
L'éditeur de document structuré Thot combine les principales caractéristiques des documents structurés et des documents hypertextes (Quint & Vatton, 1992).
Les documents structurés contenant non seulement des liens structurels mais également des liens hypertextes ou hypermédia sont parfois appelés des hyperdocuments.

Nous présentons ci-après de manière plus approfondie les deux grandes techniques de structuration et de gestion de documents que sont les documents structurées et les systèmes hypertextes. Puis nous présenterons l'éditeur Thot.

3.2.1 Les documents structurés

3.2.1.1 Avantages

L'exploitation d'un modèle structurel pour créer un document permet :
- de séparer son contenu, sa structure logique et sa sa structure physique ;
- de guider le rédacteur lors de l'élaboration de documents ;
- d'automatiser l'aide au contrôle de complétude d'un document par rapport à son modèle ;
- d'associer des règles syntaxiques à certains EDs, et de contrôler leurs applications ;
- de rechercher automatiquement des EDs suivant leur type et de leur appliquer différents traitements (e.g. mise à jour, archivage, impression) ;
- d'échanger ces éléments entre divers éditeurs ou applications en utilisant l'un des standards internationaux.

3.2.1.2 Inclusion d'un élément de document

Dans les documents structurés ou hypertextes (e.g. dans les documents SGML et ceux de Thot), un ED peut souvent être *inclus* par d'autres éléments du même document ou de documents différents. A la différence d'une copie simple, une ***inclusion*** est une copie "*vivante*" d'un élément : quand l'élément source est modifié, les inclusions de cet élément sont également modifiées (et inversement : si un éditeur de documents structurés ou un système hypertexte permet la modification directe d'une inclusion, l'élément source doit être modifié).

L'utilisation d'une inclusion permet le partage d'une même information par différents documents (et utilisateurs) et garantit la cohérence entre ses différentes occurrences. La réutilisation est de plus facilitée par le fait que dans un document structuré différentes présentations peuvent être appliquées à un élément. Notons qu'un ED qui contient des inclusions d'éléments appartenant à un ou plusieurs documents, correspond à une vue (dans le sens où nous avons défini ce terme) sur ces documents puisque l'ED présente une *sélection* des éléments de ces documents et que la modification de l'ED source (vs d'une inclusion) est répercutée sur les inclusions (vs l'ED source).

3.2.1.3 Les normes documentaires

Nous présentons maintenant de manière plus précise les deux normes internationales de construction et stockage de documents structurés.

ODA (Office Document Architecture) est une norme ISO (ISO 8613, 1987) conçue pour transférer des documents entre des applications sans perdre leur fonctionnalités. ODA distingue et permet de représenter :
1) la "structure logique générique" (modèle structurel) d'un document et sa structure logique,
2) des "styles de présentation" pour les composants élémentaires de sa structure logique ou de sa structure physique,
3) des "styles de formatage" (mise en page) pour les composants de sa structure logique, et
4) le "profil du document", i.e. un ensemble de données de gestion du document telles que son nom, son auteur, sa date de création ou ses mots-clés, et un ensemble de données techniques telles que le codage utilisé pour les différents types de contenu et de données de configuration.

SGML (Standard Generalized Markup Language) est une norme ISO (ISO 8879, 1986) qui permet de définir un langage de balisage pour des documents. Des balises ou "marques" peuvent alors être insérées dans un document afin de délimiter des EDs (et donc de les imbriquer), de leur donner un type et de spécifier le format dans lequel ils seront édités. Dans la norme SGML, le balisage doit décrire la structure d'un document plutôt que sa présentation, et doit être non ambiguë pour un programme ou un être humain.
Compte-tenu d'un langage de balisage (ou "syntaxe abstraite" d'une structure logique générique), un éditeur SGML contrôle et guide la construction d'un document correctement balisé.

Compte-tenu d'un langage de balisage et d'un document balisé, un analyseur syntaxique SGML produit un document SGML complet (avec les spécifications du langage de balisage déclarées au début du document) et logiquement parfait.
Un logiciel de formatage peut alors effectuer des traitements d'édition comme la création automatique de tables des matières, la césure des mots, la justification.
L'ISO a entrepris de formaliser les spécifications de composition de pages dans la norme DSSL (Document Style Semantics and Specification Language) (ISO 10179, 1991).
Avec SGML, un ou plusieurs attributs typés peuvent être associés à un élément. Par exemple, un attribut de type "ID" permet d'identifier un élément de façon unique et un attribut de type "IDREF" permet de référencer un autre élément (la norme HyTime que nous présenterons plus loin complète les concepts hypertextes de SGML). Un élément d'un document SGML peut référencer un élément d'un autre document SGML ; il peut également inclure un document quelconque (graphique, image, etc.). Enfin, SGML donne la possibilité de définir plusieurs structures logiques génériques concurrentes pour un même contenu.

3.2.2 Les systèmes hypertextes

L'approche hypertexte permet d'organiser des informations de façon non linéaire, avec des liens entre des éléments d'un ou de plusieurs documents. Si les informations reliées sont de types hétérogènes (texte, graphique, son, vidéo, etc.), on parle de documents hypermedia.

Les éléments reliés sont appelés des noeuds. Un noeud peut être relié à plusieurs autres noeuds et avoir plusieurs zones activables (points d'ancrage) matérialisant les points de départ de liens hypertextes. Il existe deux types de noeuds, les noeuds concrets et les noeuds abstraits. Les premiers sont des éléments atomiques de document, les seconds sont des agrégats de noeuds concrets et forment une structure hiérarchique. Si les éléments de documents sont structurés, on peut considérer les éléments structurés comme des noeuds abstraits. Le contenu d'un noeud abstrait est constitué de zones activables permettant de retrouver un sous-élément du noeud, et ces zones activables peuvent être des inclusions de sous-noeuds, i.e. des copies vivantes de ces sous-noeuds. Pour permettre à un utilisateur de référencer une partie d'un noeud concret sans avoir à le structurer, des systèmes hypertextes permettent de définir et de référencer des "régions" sur un noeud concret.

3.2.2.1 Les normes hypertextes

De nombreux travaux portent sur la standardisation des documents hypertextes/hypermedia. Certains d'entre eux comme les normes HyTime (ISO 10744, 1991) et MHEG (MHEG, 1990), concernent plus particulièrement les documents contenant des informations audio et vidéo. Les modèles DEXTER (Halasz & Schartz, 1990) et HDA (Mabrouk, 1992), traitent eux plus spécifiquement les documents à contenu textuel ou graphique.

HyTime (Hypermedia/Time-based document structuring language) est un standard ISO basé sur le standard SGML et proposé pour représenter la structure de documents multimedia basés sur le temps (tels que des partitions musicales) ou sur l'espace. Un élément peut être référencé par son nom sa position dans un ensemble de coordonnées, ou par un constructeur sémantique. Certaines relations spatiales ou temporelles sont spécifiées. Une application traitant un document HyTime, appelle et contrôle un "moteur HyTime" qui appelle lui-même l'analyseur syntaxique SGML.

MHEG (Multimedia and Hypermedia information coding Expert Group) est une norme en cours de spécification visant à standardiser la représentation et donc l'échange d'objets élémentaires multimedia et hypermedia. Une approche objet a été choisie pour cela, et une relation "Synchro" permet de synchroniser divers objets multimedia. MHEG s'appuie entre autres sur la norme SGML.

Le modèle DEXTER divise un système hypertexte en trois couches distinctes :
1) une "couche d'exécution" responsable de la présentation du réseau hypertexte et de tous les aspects de l'interface utilisateur (dont les outils de navigation) ;
2) une "couche de stockage" responsable de la gestion du réseau, et
3) une "couche de contenu" responsable de la gestion du contenu des noeuds. Cette couche n'est pas encore spécifiée par DEXTER. Elle peut s'implémenter avec des éditeurs adaptés à la gestion de différents types d'information (texte, graphique, image, etc.).
Le principal intérêt de DEXTER est l'indépendance entre la couche de stockage et la couche de contenu grâce à un mécanisme d'*ancrage* des éléments de la couche de stockage dans ceux de la couche de contenu (i.e. un mécanisme d'adressage des seconds par les premiers).

Le modèle HDA complète le modèle DEXTER en structurant la couche pour le stockage avec ODA et en définissant précisément un mécanisme d'adressage permettant de référencer de manière abstraite et transparente des éléments de documents dans la couche de contenu. HDA traite le partage (la référenciation) d'un même contenu par plusieurs noeuds.

La plupart des systèmes hypertextes actuels implémentent "en dur" les trois couches du modèle DEXTER. Ces systèmes sont fermés car :
1) ils ne permettent pas l'accés à la couche de stockage,
2) ils ne permettent d'utiliser qu'un ensemble fixe d'éditeurs intégrés au système.

Il existe cependant des systèmes qui gardent l'indépendance entre la couche de stockage et la couche de contenu grâce à une interface fonctionnelle permettant l'intégration d'applications externes pour la gestion du contenu des noeuds, e.g. Intermedia (Yankelovich & al, 1988), Multicard (Rizk & Sauter, 1992), Microcosm (Davis & al, 1992).
Les avantages d'une telle séparation sont multiples (Davis & al, 1992) :
1) l'implémentation et la maintenance du système sont simplifiées,
2) les utilisateurs ont la possibilité d'utiliser leurs logiciels préférés (éditeurs de texte, tableurs, ...) pour gérer les documents,
3) il est ainsi possible d'intégrer différentes applications de manière transparente pour l'utilisateur (la modification de données avec une application est prise en compte par d'autres applications).

3.2.2.2 Les modèles typés

Afin de permettre une conception plus structurée des applications hypertextes, et de rendre plus puissante la recherche d'informations dans les réseaux hypertextes, de nombreux systèmes hypertextes permettent maintenant de typer les noeuds et les liens. Voici quelques avantages de ce typage selon Nanard & Nanard (1991) :

1. Les aspects généraux de domaines d'applications peuvent être représentés sous forme de schémas conceptuels permettant de distinguer les différents types d'information et leurs inter-relations.
2. Un document peut être structuré par des liens structurels ou sémantiques (i.e. représentant une relation sémantique) sur lesquels la recherche par navigation ou requête peut être rendue possible. Cette structuration guide et facilite la création de documents et permet de représenter explicitement des connaissances et des intentions lors de la création des EDs. Lorsqu'un texte est construit pour être lu séquentiellement, il n'est plus possible d'en extraire automatiquement les connaissances et les intentions de l'auteur, et donc de le restructurer de façon satisfaisante.
3. Pendant la création et la modification d'un réseau hypertexte, les auteurs sont guidés par le schéma conceptuel, ce qui entraîne une meilleure structuration et facilite le travail coopératif entre différents auteurs.
4. Les lecteurs ont une vision plus globale et plus structurée du réseau, ce qui leur permet un accès plus efficace aux informations.
5. La structure du réseau hypertexte représente une base de connaissances qui peut être exploitée par différents mécanismes de recherche et d'inférence. Par exemple, l'auteur d'un ED peut lui associer un script afin de créer un lien virtuel.

Parmi les systèmes basés sur des descriptions d'applications sous forme de schémas conceptuels, certains ne permettent d'utiliser que certains types de noeuds et de liens prédéfinis, tandis que les autres permettent de créer de nouveaux types afin d'adapter les schémas à différentes applications.

1. Pour les premiers, citons gIBIS (Conklin & Begeman, 1988), SEPIA (Streitz & al, 1992) et PHIDIAS (McCall & al, 1990). Ils sont généralement conçus pour structurer des connaissances afin de permettre le travail coopératif entre plusieurs auteurs. Par exemple, dans gIBIS, plusieurs auteurs peuvent collaborer en créant des noeuds exprimant des "arguments" (i.e. des documents de type "Argument"), des "positions" et des "résultats", reliés par des liens différents types : "supports", "objects-to", etc.
2. Parmi les seconds, on peut citer Aquanet (Marshall & al., 1991) également conçu pour permettre le travail coopératif, MORE (Lucarella & al., 1993) ainsi que Macweb (Nanard & Nanard, 1991) que nous avons déjà partiellement décrit (section 3.1.2.3).

Nous décrivons maintenant plus en détail le système MacWeb car son approche "représentation des connaissances" 1) le rend plus flexible et apte à être utilisé en acquisition des connaissances, et 2) permet la recherche d'informations sur des critères conceptuels. De par ses fonctionnalités (représentation d'informations via un réseau sémantique permettant certaines inférences, génération de vues, combinaison de différents types de recherche, langage de script), MacWeb est le système hypertexte dont se rapproche le plus CGKAT.

3.2.2.2.1 MacWeb : une intégration de différentes méthodes d'indexation et de recherche

Nous avons souligné l'intérêt pour la recherche d'informations et la création de documents, de représenter de manière explicite leurs inter-relations structurelles et sémantiques. Le modèle de MacWeb permet l'intégration de différentes techniques d'indexation et de recherche. Il contient trois niveaux :

1. Un niveau "information" semblable à la "couche de contenu" de DEXTER Les documents peuvent y être organisés par des liens structurels.
2. Un niveau "ancrage" qui permet de relier des EDs du niveau information avec des noeuds ("concepts") du niveau suivant. Nous détaillerons plus loin ce niveau.
3. Un niveau "connaissance" qui est un réseau sémantique hypertexte. Ce réseau contient des concepts liés par des relations et auxquels peuvent être associées des méthodes (ou "liens virtuels"). Ces méthodes sont des scripts composés de requêtes. Ces scripts génèrent des documents virtuels qui collectent différentes parties du niveau information sur différents critères conceptuels et assemblent ces parties[1]. Les auteurs appellent ces document virtuels des "vues orientées par la tâche" sur la base de documents.
4. Les deux derniers niveaux correspondent à la "couche de stockage" du modèle DEXTER. Un "contrôleur de dialogue" coordonne les trois niveaux précédents et correspond à la couche d'exécution du modèle DEXTER.

L'utilisateur peut poser des requêtes et naviguer :
1) au niveau information comme dans un éditeur de texte,
2) au niveau ancrage comme dans une base de données gérant les types ou les attributs sur des EDs,
3) au niveau connaissance comme dans une base de connaissances,
4) entre ces niveaux.
Les requêtes et la navigation portent sur les mêmes objets et peuvent être alternées. La génération d'un document virtuel correspond à une restriction de l'espace de recherche par la navigation puisqu'un tel document est le point de départ de plusieurs liens hypertextes. Ainsi divers types de recherches peuvent être *combinés*.

1. Un script dans MacWeb ressemble à un document balisé avec un langage de marques mais où les portions de texte balisées sont remplacées par des requêtes permettant de générer ces portions.

Le niveau ancrage permet l'indexation d'EDs au niveau information. Actuellement dans MacWeb, l'utilisateur ne peut indexer que des parties de texte, mais l'extension à d'autres types d'EDs ne modifierait pas le système d'indexation (Nanard & al, 1993a). Pour chaque ED indexé, le niveau ancrage contient un descripteur composé de :
1) une "ancre" permettant d'isoler l'ED ;
2) un "concept" représentant l'abstraction dénotée par l'ED ;
3) un "contexte" qui est le plus petit ED qui englobe l'ED indexé et qui a une signification autonome pour un lecteur, e.g. une partie de phrase lorsque l'ED est un mot ;
4) un "type de contexte" qui est un marqueur pour le contexte (un concept peut également être associé à l'ED correspondant au contexte).

De plus, dans MacWeb, un certain contenu pour les niveaux "ancrage" et "connaissance" peut être généré automatiquement à partir d'un texte en français grâce à :
1) une analyse grammaticale qui isole les groupes nominaux pouvant servir d'ancrage, et détermine les locutions incluant ces groupes nominaux afin d'identifier les types de discours associés ;
2) une analyse sémantique des groupes nominaux pour déterminer s'ils repèrent un concept.
Pour Nanard & al. (1993b), cette indexation automatique n'est pas une représentation précise des connaissances du texte mais permet de l'organiser suffisamment pour, compte-tenu de la souplesse de la recherche dans MacWeb, permettre à un utilisateur d'avoir une vision globale des informations contenues dans le texte, et de les retrouver efficacement. Cela, et notamment la génération de vues (documents virtuels), peut par exemple être utilisé par un cogniticien cherchant à modéliser un texte, ou par un expert pour valider une modélisation.

Nous présentons dans la prochaine section l'éditeur de document structuré Thot. Dans le chapitre 6, nous montrerons comment nous avons exploité cet éditeur pour créer un outil d'acquisition des connaissances qui, bien qu'ayant une structure assez différente de MacWeb, partage la plupart de ses fonctionnalités.

3.2.3 L'éditeur de document structuré Thot

3.2.3.1 Un éditeur syntaxique et des langages

Thot est un éditeur dirigé par la structure, i.e. une sorte d'éditeur syntaxique. La structure d'un document Thot doit être défini par un modèle structurel écrit dans le langage S[1] (Quint, 1995). Similairement, pour afficher un document totalement ou partiellement, Thot utilise l'un des modèles de présentation associés au modèle structurel du document (un modèle de présentation est écrit en langage P). Les commandes d'édition de Thot ne permettent de construire que des EDs dont la structure et la présentation sont conformes aux règles définies dans les modèles. Les EDs prévus par la structure mais non encore construits sont affichés sous forme de petits rectangles grisés. L'utilisateur peut ainsi connaître les EDs qu'il lui reste à remplir (l'utilisateur connaît les types de ces EDs car lorsqu'un ED est sélectionné, son type et celui de ses EDs englobants sont affichés).

Dans le langage S de Thot, les types d'EDs atomiques sont : les chaînes de caractères, les éléments graphiques, les images et les symboles mathématiques. La structuration dans Thot est proche de celle permise par SGML. Des EDs structurés peuvent être définis avec des opérateurs d'agrégation, de liste, de choix et de référence. Dans le modèle structurel, il est également possible de définir la liste des attributs qui peuvent ou doivent être associés à un ED d'un certain type. Un attribut peut être textuel, numérique, énuméré ou référence. Les EDs références et les attributs références sont des liens hypertextes orientés. Notons que le contenu physique d'un ED référence peut être constant ou calculé via des règles de présentation mais n'est pas modifiable par l'utilisateur.

1. Une version SGML de Thot permet d'utiliser SGML. Nous parlons dans cette section de la structuration permise par le langage S.

Les principales différences par rapport à SGML sont les suivantes (Quint & Vatton, 1994a) :

1. Thot permet de définir des structures abstraites de manière modulaire. Tous les types d'EDs pouvant apparaître dans un document n'ont pas à être définis dans un seul modèle structurel mais peuvent être définis dans des modèles structurels différents. Ainsi un type d'ED peut être utilisé dans divers modèles structurels. Par ailleurs, certains modèles structurels, définis comme étant des "extensions", peuvent être associés par programme à un document et ainsi compléter les définitions des types d'EDs utilisables dans ce document.
2. Thot ne permet pas des définitions concurrentes pour un type d'ED, mais la modularisation des modèles structurels et le "partage dynamique" de documents peuvent être utilisés pour remplacer de telles définitions.
3. Thot offre des attributs locaux mais également des attributs globaux. Un attribut global peut être associé à n'importe quel ED dans un document. Ainsi par exemple, des attributs globaux dans un modèle structurel d'extension permet de rajouter à des EDs déjà construits des informations spécifiques (e.g. des droits d'accès pour permettre à une application de gérer l'édition coopérative sur un document (Decouchant, 1995)).

Il existe au moins trois manières dans Thot de créer des vues, dans le sens que nous avons donné à ce terme.

1. Créer des inclusions (cf. section 3.2.1.2). Thot permet de créer une inclusion d'un ED lorsque la création d'un ED de même type est autorisée par le modèle structurel. Ainsi une même information peut être partagée par différents EDs. L'éditeur de Thot assure la mise à jour dynamique d'une inclusion lorsque sa source est modifiée. Il ne permet pas la modification directe de l'inclusion mais il gère un lien hypertexte de l'inclusion vers sa source, ce qui permet d'aller rapidement faire des modifications sur la source.
2. L'éditeur offre une deuxième méthode : il permet d'ouvrir plusieurs fenêtres pour éditer le même document et chaque fenêtre peut offrir une présentation (partielle ou totale) différente de ce document. Lorsqu'une modification est effectuée sur un ED dans une fenêtre, les autres fenêtres affichant cet ED sont mises à jour par l'éditeur.
3. Le langage P permet de définir un ou plusieurs modèles de présentation pour un type de document, et pour chaque modèle, une ou plusieurs vues[1] (c'est aussi le terme utilisé par les auteurs de Thot). Dans chaque modèle et pour chaque vue, il est possible de définir les caractéristiques des boîtes de présentation à utiliser pour le document et ses sous-éléments : positions relatives des boîtes, dimensions, couleurs, fontes et type de mise en lignes du texte, pagination, etc. Notons que la conception du modèle structurel n'est pas entièrement indépendante de celle du modèle de présentation : il est souvent nécessaire de définir dans le modèle structurel des attributs, ou même parfois des types d'EDs, destinés uniquement à permettre l'écriture de règles de présentation dans le modèle de présentation. Cela est cependant parfois naturel, par exemple pour des attributs permettant à l'utilisateur de modifier des options de présentation.

Afin de permettre la mise à jour de documents lors de l'évolution des modèles structurels qu'ils utilisent, ou encore la sauvegarde de ces documents sous divers formats (e.g. SGML, Latex), Thot permet de définir des règles de conversion de structure à l'aide d'un langage de traduction, le langage T.

1. Un modèle de présentation définit une présentation par défaut pour un type de document, mais d'autres présentations peuvent également être définies à l'intérieur de ce modèle. Ces diverses définitions de présentation permettent d'éditer diverses vues sur le document. Les diverses vues peuvent ne pas présenter les mêmes EDs. Une vue dans Thot est bien une vue dans le sens où nous avons défini ce terme puisqu'elle permet de visualiser certains types d'éléments précis suivant des règles de présentation précises, et si plusieurs vues partagent le même ED, une modification de cet ED dans une vue est répercutée dans les autres vues.

3.2.3.2 La structuration et la recherche d'informations via l'éditeur Thot

Les EDs peuvent être structurés par des liens structurels, des liens hypertextes et des inclusions. Grâce aux inclusions, une même information être utilisée ou contextualisée de différentes manières.

De plus, l'utilisateur peut isoler une *portion d'un ED textuel*, ou un *ensemble d'EDs* structurés consécutifs et de même niveau dans la structure logique, en l'entourant avec une "paire de marques" (les paires de marques sont des EDs élémentaires). Les types d'éléments pouvant être entourés par une paire de marques et les types d'attributs pouvant ou devant être portés par celle-ci doivent être définis dans un modèle structurel.

Grâce aux liens structurels, les EDs peuvent être organisés selon une structure d'arbre. Un document peut contenir ***un ou plusieurs arbres*** d'EDs, et donc un ou plusieurs EDs racines. Des EDs de différents arbres peuvent être reliés par des liens hypertextes. De cette manière, des EDs de l'arbre principal peuvent être par exemple annotés par des EDs d'autres arbres. Une présentation particulière doit être définie pour l'impression d'un tel document, afin de spécifier si les annotations doivent être imprimées avec les EDs de l'arbre principal (e.g. avec des notes de bas de pages) ou séparément (affichage arbre par arbre).

Chaque arbre d'EDs d'un document est édité par Thot dans une fenêtre différente, c'est-à-dire, pour reprendre la terminologie utilisée par les auteurs de Thot, dans une "vue" différente. Il s'agit là simplement d'une visualisation partielle d'un document mais cette notion de vue s'accorde avec notre définition. Il y a donc dans Thot quatre manières de définir des vues : 1) avec des inclusions, 2) en ouvrant différentes fenêtres sur une même partie de document (un arbre d'EDs), 3) en appliquant différentes présentations à une même partie de document (e.g. n'importe quel ED), et 4) en éditant un arbre d'EDs d'un document, c'est-à-dire un des EDs racines du document.

L'éditeur Thot permet la navigation dans les deux sens sur les liens structurels et sur les liens hypertextes : depuis un ED qui est la destination de liens hypertextes, les EDs sources de ces liens peuvent être recherchés séquentiellement (le document qui les contient peut alors être ouvert afin de les afficher).

Notons que la valeur d'un tel attribut référence étant un pointeur sur l'ED référé, aucune autre valeur (e.g. un nom particulier) ne peut leur être associée. Ainsi lors de la navigation sur les liens hypertextes, l'utilisateur n'est guidé que par le type de l'attribut référence, i.e le nom sous lequel il a été défini dans le modèle structurel. Le nombre et le type des attributs référence utilisables dans un document est fixé par le modèle structurel. Un modèle structurel ne peut être modifié dynamiquement, c'est-à-dire au cours de l'édition.

L'éditeur de Thot permet de rechercher des EDs 1) sur leur contenu textuel par exemple par une expression régulière, et/ou 2) sur leurs types ou sur ceux de leurs attributs, mais il ne permet pas de prendre également en compte les valeurs des attributs.

Thot intègre un générateur d'index (Richy, 1994) qui :
1) permet à l'utilisateur d'***associer*** des descripteurs à des parties de documents (isolées avec des paires de marques) et
2) peut *synthétiser* ces descripteurs dans une table d'index en effectuant un tri alphabétique sur les clés de tri contenues dans les descripteurs.
Diverses options permettent de paramétrer le tri. Les champs d'un descripteurs sont des EDs atomiques permettant d'indiquer des clés de tri, un "sujet", une "sémantique" et une "glose". Lorsque la table d'index est créée, des liens hypertexte relient ses éléments
1) aux EDs indexés (plus exactement aux paires de marques les entourant),
2) aux descripteurs qui ont permis de construire ces éléments, et
3) entre eux si des références croisées sont créées.
L'utilisateur peut donc rechercher des informations dans le document, par navigation sur ces divers liens.

3.2.3.3 Une interface fonctionnelle permettant la création de documents actifs

Thot dispose d'une interface fonctionnelle (Quint & Vatton, 1995a) permettant à une application de créer, modifier et rechercher des EDs. Comme dans l'éditeur, les opérations éffectuées doivent respecter les modèles de structure et de présentation choisis.

Par ailleurs, le mécanisme d'appel externe de Thot (Quint & Vatton, 1995b) permet de définir des procédures à appeler lorsque l'utilisateur de Thot effectue certains *types d'actions* sur un document, par exemple la création d'un ED d'un certain type, la mise à jour de la valeur d'un attribut d'un certain type, et la sauvegarde d'un ED d'un certain type. De manière plus synthétique, un type d'action peut être :
1) la création, destruction, modification, sauvegarde ou lecture d'un ED ou d'un attribut d'un certain type,
2) l'ajout ou la modification de certaines règles de présentation pour un ED d'un certain type.

Un document Thot peut ainsi devenir un *document actif* et être utilisé comme une interface pour une application. Une procédure de l'application peut être appelée au commencement d'une certaine action ou lorsque celle-ci est achevée. Dans le premier cas, la procédure de l'application peut effectuer des vérifications préalables, et soit autoriser le traitement normal que Thot associe à cette action (e.g. la destruction d'un ED), soit indiquer à Thot de ne pas effectuer ce traitement normal (la procédure peut alors éventuellement effectuer un traitement de remplacement).
La procédure peut utiliser l'interface fonctionnelle de Thot, par exemple pour modifier le document en cours d'édition ou générer d'autres documents. Des liens virtuels peuvent ainsi être implémentés.

Quint & Vatton (1994a, 1994b) citent diverses applications utilisant les documents actifs dans Thot :
1) les applications intégrées à l'éditeur, i.e. le générateur d'index, un mécanisme d'annotation et un gestionnaire de documents ;
2) Griffon (Decouchant & al., 1993), une application permettant à plusieurs éditeurs Thot se trouvant sur diverses machines d'effectuer de l'édition coopérative ; l'application d'édition coopérative maintenant développée autour de Thot s'appelle Alliance (Decouchant, 1995) ;
3) les éditeurs syntaxiques pour les langages Peplom et Argos (cf. (Francou, 1993) et (Schaar, 1994)).

3.3 Conclusion

3.3.1 Bilan sur les techniques de recherche et de gestion d'information

Une recherche ou une gestion efficace d'informations nécessite la structuration ou l'indexation de ces informations. Les critères suivants nous semblent intéressants pour distinguer les différents systèmes de gestion ou de recherche d'informations :

1. *La méthode d'analyse utilisée pour structurer ou indexer les informations* : méthodes statistiques, méthodes de classification, méthodes d'analyse du langage naturel, etc.
2. *La nature des informations qui peuvent être structurées ou indexées* : texte, image, son, ED structuré, etc.
3. *La distinction effectuée entre le contenu des informations, leur organisation (structure logique ou index), et leur présentation (structure physique)*. Les documents structurés permettent cette séparation et la norme DEXTER pour les systèmes hypertextes préconise la séparation entre contenu et organisation.
4. *La précision et l'organisation des descripteurs utilisés pour la structuration ou l'indexation* : types d'EDs ou de liens prédéfinis, mots-clés, attributs, types de concepts ou de relations, concepts, graphes conceptuels, etc. Plus les descripteurs sont précis, à un niveau conceptuel, et organisés par des relations précises et à un niveau conceptuel, plus ils pourront être exploités pour permettre une recherche précise, à un niveau conceptuel, et souple ou performante. Pour cela, les descripteurs peuvent être représentés et organisés dans un langage de représentation de connaissances et ainsi former une base de connaissances.
5. *L'aide à la recherche* : recherche par navigation ou par requête, liens virtuels, génération et gestion de vues ou documents virtuels, langage de scripts, etc.
6. *L'aide à la structuration ou à l'indexation* : éditeurs de documents structurés ou hypertextes, ontologie de types de concepts ou de types de relations, langage graphique de représentation de connaissances, contrôles de cohérence, etc.

Par contre, nous n'avons pas retenu comme critère le type de support utilisé pour les descripteurs : des connaissances peut être représentées avec les noeuds et les liens d'un système hypertexte typé (e.g. avec MacWeb) ou bien dans une base de connaissances séparée (e.g. avec CGKAT) ou encore dans une base de données orientée objets séparée (e.g. avec HyperPATH/O2 (Amann & al., 1993)). L'important n'est pas le support en soi mais les recherches, les contrôles et les inférences qui sont effectuées dans le système qui exploite ce support.

3.3.2 Combinaison de techniques de recherche et de gestion d'information

CGKAT n'intègre pas d'outil d'analyse automatique ou semi-automatique pour la structuration ou l'indexation des informations de documents, mais combine l'éditeur de documents structurés hypertextes Thot avec la plate-forme de gestion de graphes conceptuels CoGITo de manière à offrir une grande palette de possibilités pour les points 2 à 6 ci-dessus :

1. L'utilisateur de CGKAT peut utiliser Thot pour créer et structurer diverses sortes d'EDs (texte, image, graphique, ED structuré) et il peut les indexer de différentes manières par des graphes conceptuels en utilisant différentes types de liens hypertextes ;
2. Il peut utiliser un langage de commandes pour rechercher ou combiner des connaissances et il peut utiliser ce langage pour rechercher par requête conceptuelle des informations, i.e. via les connaissances qui les indexent.
   Le résultat d'une requête est un document qui rassemble les informations qui répondent à la requête. Ce document est un document virtuel ou une vue : il permet de visualiser des parties de documents sélectionnées sur des critères conceptuelles, et la vue est automatiquement modifiée si les informations sources sont modifiées. De plus, ces informations sources sont accessibles par navigation depuis la vue. Pour permettre cela, CGKAT utilise des inclusions de ces informations sources pour construire le document affichant les réponses d'une requête.

L'utilisateur peut écrire des scripts avec le langage de commandes et créer des liens virtuels.
Il peut également rechercher des informations "via les connaissances" en combinant la navigation entre des informations et leurs descripteurs (des graphes conceptuels) et la navigation entre des graphes conceptuels.

3. L'utilisateur peut construire des graphes conceptuels sous format graphique et en utilisant l'éditeur Thot. Il peut donc rechercher, présenter et gérer ces graphes en utilisant les fonctionnalités de Thot. Des contrôles sont effectuées sur la construction des graphes : ils doivent respecter les contraintes définies dans l'ontologie de l'application. Pour guider et faciliter la construction de cette ontologie, CGKAT propose une ontologie générale contenant a) des types de concepts de haut niveau spécialisés par des types de concepts du langage naturel, et b) des types de relations de base.

Pour permettre la construction de graphes conceptuels dans des documents Thot, nous avons défini un modèle structurel et un modèle de présentation pour de tels éléments. Dans un autre modèle, nous avons défini les liens hypertextes qui permettent d'indexer par de tels éléments n'importe quel ED construit avec Thot.
De plus, afin de pouvoir exploiter le formalisme des graphes conceptuels, lorsqu'un graphe conceptuel est créé ou modifié dans Thot, il est automatiquement créé ou modifié dans la base de graphes de CoGITo. De même, lorsqu'un document contenant des graphes conceptuels est ouvert, ces graphes sont créés dans la base de CoGITo. Pour cela, nous avons exploité le mécanisme d'appel externe de Thot et son interface fonctionnelle : nous avons fait des documents Thot, des documents actifs interfaçant une base de graphes conceptuels. Les documents générés en réponse à une requête exploite également l'interface fonctionnelle de Thot.

Enfin, CGKAT exploite le générateur d'index de Thot afin d'indexer et de synthétiser pour des EDs certaines de leurs caractéristiques : auteur, point de vue, type de concept utilisé, source d'information (e.g. expert ou document), commentaire associé.

Nous concluons ce chapitre en soulignant les liens entre la recherche d'informations et la production d'explications.

3.3.3 Recherche d'informations et explications

Présenter des explications sur un objet (connaissance du domaine, tâche, résultat d'un raisonnement, etc.), c'est fournir à l'utilisateur des informations répondant à son besoin, celui-ci pouvant avoir été (partiellement) exprimé avec une requête ou non. Un système de recherche d'information est une composante essentielle d'un système explicatif :

1. Une solution pour fournir des explications sur un objet consiste à montrer à l'utilisateur les divers types d'informations pouvant concerner cet objet et à laisser faire à l'utilisateur le choix le plus pertinent pour ses besoins. L'explication repose ici sur la présentation de liens hypertextes statiques ou virtuels, i.e sur la recherche par navigation. Cette présentation de liens peut se faire de multiples façons, par exemple avec un graphe ou bien un menu déroulant. Insistons sur le fait que ces liens peuvent être des liens virtuels : par exemple, un menu déroulant peut montrer la liste des questions qui peuvent être posées sur un objet sélectionné. De plus, la liste des liens peut être contextuelle, et par exemple prendre en compte la tâche en cours, les informations déjà présentées, le modèle de l'utilisateur, etc.
2. Une autre solution pour fournir des explications est de répondre à une question (ou requête) de l'utilisateur. Ici aussi, le contexte peut aider à interpréter la requête et à mieux cerner le besoin en information de l'utilisateur. Afin de mieux cerner ce besoin, le système peut éventuellement gérer un dialogue avec l'utilisateur.
3. Une troisième solution pour le système est de fournir spontanément des informations à l'utilisateurs compte-tenu de la tâche en cours et des autres éléments du contexte.

Ces diverses méthodes peuvent être combinées. Par exemple, le résultat d'une requête peut être

une vue sur la base ou comporter de nombreux liens hypertextes vers différents objets de la base. Notons l'importance de la manière dont les informations sont présentées : ordre, structuration, usage de graphiques, etc.

La différence entre un système explicatif et un système de recherche d'information plus classique, réside dans :
1) l'exigence de la qualité des réponses (informations pertinentes et à un niveau d'abstraction adéquat, ordre pertinent, etc.), et
2) la complexité des questions qui peuvent être posées (e.g. questions du type pourquoi, pourquoi pas, comment, que se passerait-il si, etc.). Les questions peuvent nécessiter une résolution de problème, ou porter sur une résolution de problème et non plus seulement sur des connaissances "statiques".

Chapitre 4 Les Graphes Conceptuels

4 Sommaire

4.1 Introduction

Le formalisme des Graphes Conceptuels (Sowa, 1984) est dérivé à la fois des modèles de représentation de connaissances généraux de type "réseau sémantique" (Sowa Eds, 1991) et des Graphes Existentiels de C.S. Peirce (Roberts, 1973). Les Graphes Conceptuels (GCs) ont pour but d'être "un système de logique hautement expressif, permettant une correspondance directe avec la langue naturelle" (Sowa, 1992).

Comme le souligne Chein (1994), le formalisme des GCs partage les avantages des modèles de type réseau sémantique (i.e. les modèles utilisant des graphes étiquetés), qui permettent
1) la représentation de connaissances diverses et des aspects sémantiques de la langue naturelle,
2) des traitements du type algorithmique et programmation de graphes,
3) la visualisation sous une forme graphique de ces connaissances.

De plus, les GCs ont des particularités intéressantes comme par exemple (extrait de Chein (1994)) :
a) utilisation d'un seul modèle pour décrire tous les objets quel que soit leur niveau,
b) structuration des types de concepts et possibilité de structurer les types de relations,
c) mécanisme simple de définition de type et opérations associées (expansion/contraction),
d) séparation explicite de représentations de nature différente (treillis des types de concepts, lambda-expressions, graphes canoniques, schémas, etc.),
e) relations d'arité quelconque,
f) définition d'opérations simples sur les GCs simples (cf. section 4.2.2),
g) définition à partir de ces opérations, d'une relation de spécialisation entre ces GCs simples qui correspond à la notion de morphisme de graphes étiquetés et au problème de satisfaction de contraintes, ce qui permet d'importer des algorithmes développés dans ce domaine,
h) interprétation logique "consistante et complète" des GCs simples,
i) possibilité de représenter des notions complexes : référents ensemblistes, graphes emboîtés, etc.

Le livre initial de Sowa de 1984 contenait essentiellement un modèle simple et de nombreuses idées pour enrichir le modèle (Chein, 1992). Depuis, ce modèle de base a été adapté et étendu dans diverses directions par de nombreux chercheurs. Il a fait l'objet d'études formelles et a été appliqué dans divers domaines comme les bases de données, l'analyse du langage naturel et l'acquisition de connaissances. De nombreuses extensions proposées au modèle de base n'ont pas encore d'interprétation logique bien définie. Elles servent alors comme notations semi-formelles et non comme un langage formel. Néanmoins, nous les présenterons car elles peuvent être utiles en acquisition des connaissances de deux façons : 1) en offrant au cogniticien un langage semi-formel pour préciser des connaissances, et 2) en permettant des traitements ad-hoc.

Nous détaillerons tout d'abord ce modèle de base des GCs, décrit dans le chapitre 3 de Sowa (1984) et défini formellement dans (Chein & Mugnier, 1992a) (Haemmerlé, 1995) (Mugnier & Chein, 1996). Les GCs de ce modèle de base, ou GCs "simples" (GCSs), ont une interprétation en logique du premier ordre.
Seul ces GCSs sont actuellement gérés par CoGITo (Haemmerlé, 1995), la plate-forme de développement de base de GCs qu'utilise CGKAT.

Nous présenterons ensuite rapidement des extensions à ce modèle proposées par différents chercheurs, et nous montrerons comment nous avons permis dans CGKAT la construction d'une certaine forme de graphes de graphes emboîtés.

Enfin, nous soulignerons l'intérêt des GCs et évoquerons leurs applications dans divers domaines.

4.2 Le modèle de base des GCs

4.2.1 Présentation informelle

Les GCSs sont constitués de deux types de sommets: les sommets concepts (aussi appelés *concepts*) et les sommets relations (aussi appelés *relations conceptuelles)*. Chaque sommet (ou noeud) est muni d'une étiquette (voir figure 4.1).
L'étiquette d'une relation conceptuelle est un type de relation, tandis que celle d'un concept est constitué d'un type de concept[1] et d'un référent.
Les types et les référents utilisés dans les GCSs doivent avoir été préalablement déclarés ou définis dans un "*support*" (la définition formelle de ce support est donnée plus loin) afin d'*ordonner* et/ou de poser des contraintes sur ces types et ces référents. Nous assimilons un support à une ontologie.

Les types des concepts sont ordonnés en une structure de treillis fini par une relation d'*ordre partielle* représentant la relation sémantique "sorte-de". Cette relation est une relation de *spécialisation/généralisation* ou encore une relation de *subsomption* entre types de concepts.
Les types de relations conceptuelles peuvent également être ordonnés par une relation de subsomption.
Les sous-types d'un type, i.e. ses spécialisations, peuvent être "directs" (ou immédiats) ou bien "indirects" (existence de types intermédiaires).

Comme expliqué dans la section suivante, les relations de subsomption sur les étiquettes des sommets de GCSs induisent une relation de subsomption sur ces GCSs, par "*projection*".
Actuellement dans CoGITo, l'ordre sur les relations est pris en compte pour calculer la relation de spécialisation entre les GCSs. Dans le modèle de base des GCs (Sowa, 1984), cet ordre n'était pas pris en compte.

Dans un GCS, le référent d'un concept peut être :
1) un référent individuel, ou bien
2) un référent générique[2] qui peut être anonyme (il est alors noté "*") ou bien suivi d'un identificateur de variable (e.g. "*x", "*y", "*z").

1. Un référent individuel doit être "conforme" au type du concept qui le contient (cf. section 4.2.2.1 pour une définition de cette relation de conformité). Ce concept, alors appelé *concept individuel*, représente un *individu* particulier. Cet individu peut aussi bien être une entité physique particulière, e.g. une certaine personne, qu'un processus particulier, e.g. celui qui a consisté à écrire cette phrase.
2. Dans le deuxième cas, le concept est un *concept générique* qui représente et subsume l'ensemble des concepts individuels ayant le même type de concept. Un concept générique de type X signifie "il existe au moins un individu de type X".

---

1. Dans la littérature sur l'acquisition ou la représentation des connaissances, le terme de "concept" est souvent utilisé pour désigner ce que dans la terminologie des GCs on appelle un "type de concept". C'est également souvent le cas dans la littérature sur les GCs, lorsqu'il n'y a pas ambiguïté (e.g. dans l'expression "treillis de concept"). De même pour les relations (e.g. "hiérarchie de relations"). **Dans ce rapport, lorsque nous parlons des GCs, le terme de "concept" désigne toujours un "sommet concept", et le terme de relation, "un sommet relation"**. Par ailleurs, nous utilisons les expressions de "treillis de types de concept" et de "hiérarchie de types de relation".

2. Ce référent est également qualifié d'existentiel car il dénote le quantificateur existentiel "∃".

Un GC peut représenter la signification littérale d'une phrase. Par exemple, "Jean va à Nice avec une voiture" peut être représentée par le GC suivant.

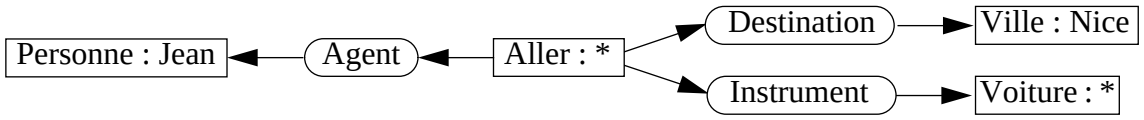


Figure 4.1: *Un GC représentant la signification littérale de "Jean va à Nice avec une voiture".*

Un GCS peut être traduit par une fonction notée ϕ (Sowa, 1984) en une formule bien formée de la logique des prédicats du 1er ordre. Par exemple pour celui ci-dessus:

$(\exists x)\,(\exists y)\,(\,\text{Personne(Jean)} \wedge \text{Aller(x)} \wedge \text{Ville(Nice)} \wedge \text{Voiture(y)} \wedge$
$\text{Agent(x,Jean)} \wedge \text{Destination(x,Nice)} \wedge \text{Instrument(x,y)}\,).$

Cette formule peut être ainsi lue : il existe un 'x' et un 'y' où Jean est une personne, 'x' une instance de Aller (un aller précis), Nice une ville, 'y' une voiture, et l'agent de 'x' est Jean, la destination de 'x' est Nice et l'instrument de 'y' est une voiture. Un GCS est une représentation graphique d'une formule logique conjonctive, positive, fermée existentiellement et ayant pour seuls termes des variables ou des constantes.
La hiérarchie des types peut se traduire avec des implications logiques : pour tous les couples (t, t') de types de concepts, si $t < t'$, alors $\forall x\ t'(x) \Rightarrow t(x)$.
De même, si G' est une spécialisation de G, on a : $\phi(G') \Rightarrow \phi(G)$.

Les GCs bénéficient également d'une notation linéaire. Par exemple, la phrase "Jean va à Nice" peut être représentée par le GC :

[Personne : Jean]←(Agent)←[Aller : *]→(Destination)→[Ville : Nice]. (GC1)

Et "Jean va à Nice en voiture" par :

[Aller : *]- { (Agent)→[Personne : Jean];
(Destination)→[Ville : Nice];
(Instrument)→[Voiture : *];
}. (GC2)

Cependant, hormis pour les cas très simples tel GC1, les chercheurs utilisent différentes notations linéaires. Des travaux de standardisation pour la notation linéaire sont en cours, e.g. (Wermelinger, 1995). Dans CGKAT et dans ce rapport, nous utilisons pour les GCSs la notation linéaire qui peut être comprise et générée par CoGITo (Haemmerlé, 1995).

Dans la notation linéaire comme dans la notation graphique, les référents génériques peuvent être omis. Pour alléger le dessin des GCs, nous omettrons désormais également le cercle autour des relations conceptuelles. D'où une variante pour l'affichage du GC de la figure précédente :

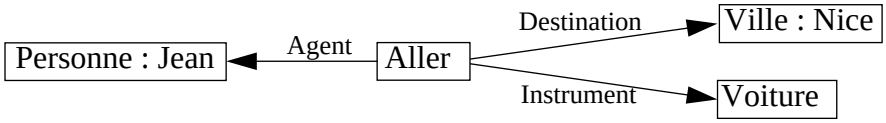


Figure 4.2: *Une variante graphique pour l'affichage du GC de la précédente figure.*

De manière à uniformiser les notations, Sowa (1984) préconise de ***nommer les types de relations binaires conceptuelles de telle sorte qu'une relation de type 'r' partant d'un concept de type 't1' pour arriver à un concept de type 't2' puisse être lue, le 'r' de 'c1' est 'c2', ou encore, le 'c1' a pour 'r' 'c2'***. Ainsi l'exemple ci-dessus peut être lu :
la 'Destination' de 'Aller' est une 'Ville', ou encore, 'Aller' a pour 'Destination' une 'Ville'.

Nous avons suivi cette règle de nommage dans nos exemples et pour la construction de notre ontologie de types de relations.

Un référent générique peut être nommé : la notation '*' est alors suivie d'un identificateur de variable. Si une même variable apparaît dans différents concepts d'un même GC, ces concepts sont dits "coréférents", c'est-à-dire qu'ils désignent les mêmes individus. Par exemple, "Quelqu'un se regarde" peut être ainsi représenté :

[Personne : *x]←(Agent)←[Regarder]→(Object)→[Personne : *x].

Pour représenter la coréférence dans la notation graphique, au lieu d'utiliser des variables identiques, une ligne d'identité peut être utilisée : cette ligne d'identité joint les concepts coréférents et est dessinée en pointillés.

Notons enfin qu'il est d'usage que l'identificateur d'un référent individuel commence par '#', particulièrement s'il contient des chiffres.

4.2.2 Présentation formelle

Nous présentons maintenant le modèle de base des GCs formellement défini par (Chein & Mugnier, 1992a). Le texte des trois sections suivantes (4.2.2.1, 4.2.2.2 et 4.2.2.3) est en majeure partie extrait de (Chein, 1992). Nous noterons plus loin quelques extensions apportées à ce modèle par Mugnier & Chein (1996). Nous considérons qu'un "support" (cf. définition ci-dessous) est une ontologie.

4.2.2.1 Support et graphes conceptuels simples

Un support S est constitué par :
- un treillis Tc de types de concepts,
- un ensemble Tr de types de relations conceptuelles, signé sur Tc, c'est-à-dire qu'à chaque élément de Tr on associe un tuple sur Tc (arité quelconque pour une relation conceptuelle),
- un ensemble M de marqueurs (ou référents) composé:
  - d'un ensemble de marqueurs individuels,
  - d'un marqueur générique anonyme *,
  - d'un ensemble de marqueurs génériques nommés,
  - d'un marqueur absurde 0 et d'un marqueur universel 1 (l'ensemble des étiquettes des concepts peut alors être muni d'une structure de treillis ; le marqueur 1 couvrant tous les marqueurs génériques, eux-mêmes couvrant tous les marqueurs individuels, eux-mêmes couvrant le marqueur absurde),
- un prédicat conf (relation de conformité) sur l'ensemble des couples Tc x M vérifiant les 4 axiomes suivants : $\forall\, m \in M$ et $\forall\, t, t' \in Tc$

(1) conf(1,m) et non conf(0,m) et non conf(t,0)
(2) $t' \leq t$ et conf(t',m) implique conf(t,m)
(3) conf(t',m) et conf(t,m) implique conf($t' \wedge t$,m), donc $t' \wedge t > 0$
(4) $\forall\, t \in Tc-\{0\}$ on a conf(t,*) et conf(t,*x) et conf(t,1) et non conf(0,*) et non conf(0,*x).

Un GCS est un multigraphe[1] biparti connexe étiqueté G = (R,C,U,eti) avec :
(1) R est l'ensemble des sommets relations (r-sommets),
(2) C est l'ensemble non vide des sommets concepts (c-sommets),
(3) U est l'ensemble des arêtes entre ces deux classes disjointes de sommets,
(4) eti étiquette les sommets et les arêtes de la manière suivante :
  (4.1) un c-sommet est étiqueté par un couple de TcxM vérifiant conf,
  (4.2) un r-sommet est étiqueté par un élément de Tr,
  (4.3) les arêtes incidentes à un r-sommet étiqueté par r sont étiquetées $1,2...a_r$, si a_r est l'arité de r, on note par Gi(r) le i-ème voisin de r,
  (4.4) le c-sommet extrémité d'une arête étiquetée i et ayant pour autre extrémité un r-sommet d'étiquette r a pour type de son étiquette un type inférieur ou égal (dans l'ordre du treillis) au i-ème type de la signature de r.

1. Un multigraphe est un graphe où il peut exister plusieurs arêtes entre deux sommets.

4.2.2.2 Morphismes et projections

Un morphisme de GCS de G = (R,C,U,eti) dans G' = (R',C',U',eti) est un couple (f,g) d'application, f de R dans R', et g de C dans C', tel que :
(1) $\forall$ r $\in$ R et $\forall$ i $\in$ {1...,degré(r)}, $G_i(r) = c$ implique $g(c) = G'_i(f(r))$,
(2) $\forall$ r $\in$ R eti'(f(r)) = eti(r),
(3) $\forall$ c $\in$ C eti'(g(c)) = eti(c).
Ce morphisme est donc un morphisme de graphe biparti étiqueté qui conserve l'ordre sur les voisins d'un r-sommet.

La définition d'une projection de GCS est obtenue à partir de celle d'un morphisme de GCS en remplaçant dans (3) l'égalité par une inégalité :
(3) $\forall$ c $\in$ C eti'(g(c)) $\leq$ eti(c). (pour une inf-projection)
(3) $\forall$ c $\in$ C eti'(g(c)) $\geq$ eti(c). (pour une sup-projection)

Pour conserver la terminologie de Sowa, le terme de *projection* sera désormais utilisé pour désigner une *inf-projection*, i.e. un morphisme de graphes qui conserve les étiquettes des r-sommets mais peut diminuer celles des c-sommets. Diminuer l'étiquette d'un c-sommet revient à remplacer son type par un type plus petit, ou, si le c-sommet est générique, à mettre un marqueur individuel à la place du marqueur générique, ceci tout en continuant à satisfaire la relation de conformité.

Un isomorphisme de GCS (resp. isoprojection) est un morphisme de GCS (resp. projection), noté (f,g), tel que f et g soient bijectives. Une composition de morphismes (resp. projections, isomorphismes, isoprojections) est un morphisme (resp. projection, isomorphisme, isoprojection). Le problème de la recherche d'une projection ou d'une isoprojection d'un graphes G dans un graphe G' est NP-complet si G et G' sont des GCSs quelconques. Cependant si G est un S-arbre, i.e. un GCS sans cycle, hormis les cycles dûs à des multi-arêtes entre un c-sommet et l'un de ses c-sommets voisins, le problème est polynomial. La recherche de graphes dans une base de GCSs, par projection d'un S-arbre "requête" est donc polynomiale.

On peut noter à propos des graphes donnés en exemples dans la section précédente, que GC1 se projette dans GC2.

4.2.2.3 Les opérations de spécialisation et de généralisation

4.2.2.3.1 Les opérations élémentaires

Ce sont des opérations internes, unaires ou binaires, sur l'ensemble des GCS définis sur un même support S.

Trois opérations élémentaires de spécialisation sont définies. Sowa (1984) les appelle des "règles de formation canoniques".

1. Suppression d'un r-sommet jumeau : deux r-sommets d'un même GCS sont dits jumeaux s'ils ont même étiquette et même voisinage (pour tout i, les i-èmes voisins de ces deux sommets sont identiques). Cette opération est appelée "simplification" par Sowa.
2. Diminution de l'étiquette d'un c-sommet : l'étiquette est remplacée par une étiquette plus petite suivant l'ordre sut Tc x M, la relation de conformité devant être satisfaite par la nouvelle étiquette. Cette opération est appelée "restriction" par Sowa. Un cas particulier de cette restriction est l'opération unaire de copie d'un graphe proposée par Sowa (il y a quatre "règles de formation canoniques" dans (Sowa, 1984)).
3. La jointure sur des c-sommets de même étiquette : soit G1 et G2 étant deux GCSs possédant respectivement un c-sommet c1 et c2 de même étiquette, ces deux graphes sont fusionnés par identification de c1 et c2. Si G1 et G2 sont distincts, il s'agit d'une jointure externe, sinon il s'agit d'une jointure interne.

Un graphe G1 est dit ***spécialisation*** d'un graphe G2 si l'on peut passer de G2 à G1 par une séquence d'opérations élémentaires de spécialisation.
Les GCSs relatifs à un support S sont ainsi liés par une relation de spécialisation. Cette relation est transitive mais n'est pas antisymétrique dans le cas général (certains graphes pouvant être équivalents) (Chein & Mugnier, 1992a). Elle n'est donc pas un ordre mais un pré-ordre. *Elle est toutefois notée* '≤'. Il est également prouvé dans (Chein & Mugnier, 1992a) que G1 est une spécialisation de G2 si et seulement si il existe une projection de G2 dans G1.

Trois opérations élémentaires de *généralisation*, symétriques des opérations élémentaires de spécialisation, peuvent également être définies. Un graphe G1 est dit généralisation d'un graphe G2 si l'on peut passer de G2 à G1 par une séquence d'opérations élémentaires de généralisation. G1 est une spécialisation de G2 si et seulement si G2 est une généralisation de G1.

Sowa a montré que si G1 ≤ G2, alors ϕ(G1) ⇒ ϕ(G2). Puisque l'on peut passer de G1 à G2 par une séquence d'opérations de généralisation, les opérations de généralisation de GCSs constituent un ensemble consistant de règles d'inférence sur l'ensemble des formules logiques associées aux GCSs.

La réciproque de ce théorème est prouvée dans (Chein & Mugnier, 1992a) : si ϕ(G1) ⇒ ϕ(G2), alors G1 ≤ G2. Les opérations de généralisation de GCS constituent donc un ensemble complet de règles d'inférence sur l'ensemble des formules logiques associées aux GCSs.

Le modèle de base des GCs est ainsi un modèle ***complet et consistant*** par rapport à la logique du premier ordre. Les *opérations élémentaires de généralisation sur les GCSs préservent la vérité*, tandis que les *opérations élémentaires de spécialisation préservent la fausseté*.

Sowa (1984) note également qu'à partir d'un ensemble de graphes représentant des situations *réelles ou possibles* dans un monde donné (Sowa (1984) les appelle des "graphes canoniques"), les diverses combinaisons de ces graphes par les opérations élémentaires de spécialisation représentent elles aussi des situations réelles ou *possibles* dans ce monde donné.
Ainsi, le graphe suivant "[Dormir]→(Agent)→[Idée]→(Attr)→[Vert]" qui est une représentation de la phrase "une idée verte dort", ne peut être dérivé de graphes canoniques via des opérations élémentaires de spécialisation, si l'on suppose qu'aucun graphe canonique ne relie un concept de type "Idée" (ou d'un de ses supertypes) à un concept de type "Dormir" (ou d'un de ses supertypes) ou à un concept de type "Vert" (ou d'un de ses supertypes).

Grâce à la relation de spécialisation sur les GCSs, ces derniers peuvent être recherchés dans une base de GCSs avec un GC "requête" : les réponses à cette requête sont tous les GCSs de la base qui spécialisent cette requête. La recherche de ces spécialisations peut s'effectuer par comparaison systématique avec chaque GCS de la base en utilisant l'opération de projection (Haemmerlé, 1995). Le nombre de GCSs à comparer peut être réduit en utilisant 1) une structure de données implémentant la relation de spécialisation entre les GCSs, e.g. UDS (Levinson, 1994), ou 2) une fonction de hash-code sur les GCs (de telles fonctions sont proposées par Ellis & Lehmann (1994) et Cogis & Guinaldo (1995)).

4.2.2.3.2 Des opérations de spécialisation plus complexes

La structure de graphe des GCSs permet de définir de nombreuses notions d'appariement plus ou moins partiel : étant donné deux GCSs G1 et G2, on peut chercher des spécialisations communes "maximales", des généralisations communes "minimales", un GCS à "distance minimale" de G1 et G2, etc.

Pour effectuer ces appariements, les opérations qui sont utilisées sont souvent des opérations particulières de spécialisation que l'on peut regrouper sous le terme générique de "jointures étendues". Deux types de jointure étendue sont particulièrement intéressants.

1. La jointure implique de trouver une partie "commune" et "maximale" à G1 et G2. Elle peut alors être appelée "isojoint" car les sous-graphes G1' et G2' mis en correspondance sont isomorphes. Le résultat de l'isojoint de G1 et G2, i.e. une spécialisation commune à G1 et G2, est obtenu en fusionnant (cf. définition ci-dessous) chaque sommet de G1' et son image dans G2', puis, pour deux c-sommets, l'étiquette du sommet obtenu est la borne inférieure des étiquettes des sommets à fusionner.

- La jointure a pour but de "plaquer" G1 sur G2, par exemple pour compléter ou expliquer des connaissances factuelles (G2) par des connaissances générales (G1). Les sous-graphes mis en correspondance ne sont alors pas isomorphes, mais il existe une projection de G1' dans G2' (i.e. G2' ≤ G1'). Le résultat de la jointure est alors construit de la façon suivante : on fusionne dans G1 les sommets ayant la même image, ce qui donne G1''. On fusionne ensuite tout sommet de G1'' et son image dans G2'. Notons que si G1' = G1, il existe une projection de G1 dans G2, et le résultat de la jointure est alors G2.

Etant donné n c-sommets c_1, c_2...c_n, n≥1 appartenant aux GCSs G_1, G_2...G_k, k≤n, ces c-sommets sont dits fusionnables si conf(e) est vrai, e étant la borne inférieure des étiquettes des c_i. La fusion de ces c-sommets consiste à les identifier en un unique c-sommet dont l'étiquette e' est telle que e'≤e, conf(e') étant vraie.
Similairement, on peut définir la fusion de n r-sommets ou même la fusion de deux GCSs.
G et G' sont fusionnables si et seulement si il existe un graphe connexe F et des projections surjectives tels que f:G→F et f':G'→F.

Soient G = (R,C,U,eti) et G' = (R',C',U',eti') deux graphes connexes, α une bijection de A ⊆ R sur A' ⊆ R', et β une bijection de B ⊆ C sur B' partie de C'. Le couple de bijections $\gamma=(\alpha,\beta)$ est dit compatible si l'on a :
(1) ∀ c ∈ A, c et β(c) sont fusionnables,
(2) ∀ r ∈ B, G(r) ⊆ B, et si α(r)=r' alors eti(r)=eti'(r') et ∀i, 1≤i≤degré(r), $\beta(G_i(r))=G'_i(r')$,
(3) le sous-graphe de G restreint aux sommets de A∪B, et le sous-graphe de G' restreint aux sommets de A'∪B' sont connexes.

Le résultat d'un isojoint sur G et G' est le résultat de la fusion de G et G' sur les couples de sommets mis en bijection par un couple de bijections γ compatibles.

Le problème de décision associé à la recherche d'un isojoint entre G et G' avec cardinal maximal est NP-complet, puisqu'il admet comme cas particulier le problème "isomorphisme de sous-graphes". Il reste NP-complet même si on le restreint à la recherche d'un sous-graphe g maximal au sens de l'inclusion. Par contre le problème de la recherche d'un isojoint avec γ *maximal au sens de l'inclusion* (i.e. tel qu'il n'existe pas d'extension de γ conservant la connexité du graphe fusionné) est polynomial.

Seul ce dernier type de jointure maximale est actuellement implémenté dans CoGITo. Comme pour une jointure externe simple, la fonction qui réalise la jointure maximale dans CoGITo prend en paramètres deux graphes G et G' ainsi que les concepts c et c', appartenant respectivement à G et G', sur lesquels la jointure doit être faite. Cependant, une fois la jointure simple effectuée, la fonction cherche à agrandir cette jointure au maximum (c'est la signification de "γ maximal au sens de l'inclusion").

CoGITo ne fournit pas de fonction permettant d'effectuer un isojoint entre deux graphes G et G' avec "A∪B ayant un cardinal proche du maximal". Nous avons implémenté une telle fonction dans CGKAT (cf. algorithme en section 7.2.3.4) selon la procédure suivante qui nous a été communiquée par l'auteur de CoGITo :
1) utiliser la fonction de jointure maximale précédente pour générer tous les isojoints "γ maximal au sens de l'inclusion",
2) ne retenir que le graphe dont le cardinal est proche du maximum.
Cette méthode est polynomiale. Nous l'avons implémentée car une telle jointure n'impose pas à l'utilisateur de choisir les concepts sur lesquels la jointure doit débuter : elle correspond ainsi assez bien à ce que l'on peut intuitivement attendre d'une "*jointure maximale*" sur deux GCSs.

4.2.2.4 Lambda-abstractions et définitions de types

Une lambda-abstraction n-aire est une expression de la forme *lambda(x1,x2...xn) G*, où G est un GCS appelé "corps" et ayant, pour tout i=1,2...,n un et un seul c-sommet, avec $*x_i$ pour référent générique (les x_i sont appelés "paramètres formels").

4.2.2.4.1 Définition de types de concepts

Un type de concept peut être défini par "genre et différence" ("genus&differentia") grâce à une lambda-abstraction unaire *lambda(x) G*. G est le "corps" de la définition. Le concept de G ayant pour type *x* est le "genre", tandis que le graphe G tout entier est la "difference" et exprime ce qu'un individu de ce type a de plus par rapport au genre. Cette définition peut s'écrire sous la forme "type t1(x) is G". Par exemple :

type She-Monkey (x) is [Monkey : *x]→(Characteristic)→[Sex : F].

Une telle définition exprime une ***condition nécessaire et suffisante*** d'appartenance à un type :
1) un individu de type She-Monkey est aussi nécessairement de type Monkey (et a la propriété d'être de sexe femelle), i.e. She-Monkey est défini comme un sous-type de Monkey, et
2) un individu de type Monkey ayant la propriété d'être de sexe femelle doit pouvoir être automatiquement classé comme étant de type She-Monkey.

Une telle définition est dite ***complète*** par opposition aux définitions ***partielles*** qui ne spécifient que des *conditions nécessaires ou bien suffisantes* d'appartenance à un type. Sowa ne propose pas de notation pour les définitions partielles. Comme de telles définitions sont intéressantes pour la représentation de connaissances, nous nous sommes inspiré de la notation que propose Sowa (1984) pour associer un schéma prototypique à un type, afin de trouver une notation linéaire pour les définitions de types par conditions nécessaires ou bien suffisantes ; voici comment nous les représentons dans ce rapport et dans CGKAT :

NC for She-Monkey(x) are [Monkey : *x].
SC for She-Monkey(x) are [She-Monkey : *x]→(Characteristic)→[Sex : F].

Da façon similaire, pour les définitions par conditions nécessaires et suffisantes, nous utiliserons désormais la forme suivante :

NSC for She-Monkey(x) are [Monkey : *x]→(Characteristic)→[Sex : F].

Un type qui a une définition complète est un "type défini", sinon c'est un "type atomique".
La place d'un type atomique dans le treillis des types de concepts doit être précisée. On peut utiliser pour cela la notation linéaire; par exemple :

t1 < t2, t3.

signifie que t2 et t3 doivent être sous-typés par t1. Voici un exemple plus concret :

She-Monkey < Monkey, Female.

Notons que "NC for She-Monkey(x) are [Monkey : *x]" et "She-Monkey < Monkey" sont équivalents, au moins dans CGKAT : ces deux notations lui indiquent que She-Monkey doit être un sous-type de Monkey.

Une définition complète exprimant une condition nécessaire et suffisante d'appartenance à un type, il y a équivalence logique entre les formes contractées et expansées d'un type défini. Ainsi les deux graphes suivants doivent avoir une même interprétation logique via la fonction ϕ :

[She-Monkey : Cheeta].

[Monkey : Cheeta]→(Characteristic)→[Sex : F].

Sowa (1984) propose des définitions et des méthodes pour résoudre les problèmes de ***contraction*** et d'***expansion*** de type. La contraction d'un GCS implique la recherche dans ce GCS de sous-GCS isomorphes à la "différence" (differentia) d'un type défini t, tandis que l'expansion fait appel à une jointure simple ("expansion minimale") ou bien maximale ("expansion maximale"). Leclère (1995) aborde les problèmes posés par les opérations de contraction et d'expansion de types.

4.2.2.4.2 Définition de types de relations

Les lambda-abstractions permettent également de définir de nouveaux ***types de relations conceptuelles***. Par exemple, le type de relation "Entre" peut être défini comme suit :

NSC for Between (x,y,z) are [Entity : *y]←(Left)←[Entity : *x]→(Right)→[Entity : *z].

Sowa (1984) ne propose pas de donner des définitions partielles aux types de relations. Ce n'est pas non plus possible dans CoGITo et dans CGKAT.

Lorsqu'une relation conceptuelle n'a pas de définition, les types maximaux des concepts qu'elle peut relier doivent pouvoir être précisés avec une signature de relation. Les types de relations peuvent également être ordonnés par une relation de subsomption. Dans CGKAT, cet ordre doit être précisé explicitement comme dans l'exemple suivant :

Left < Spatial_relation, Binary_relation.

4.2.2.4.3 Association de schémas prototypiques à un type de concept

Sowa (1984) introduit la notion de "schéma pour un type" qui est une lambda-abstraction unaire définissant des conditions typiques d'appartenance à ce type (i.e. la description de relation typiques entre un individu de ce type et d'autres individus).

Voici par exemple un schéma simple pour l'action de "Manger" :

```
schema for Eat (x) is [Eat : *x]- {  (Agent)→[Animal];
                                     (Object)→[Food];
                                  }.
```

Pour exprimer de telles conditions typiques, nous utiliserons désormais la notation suivante :

```
TC for Eat (x) are  [Eat : *x]- {  (Agent)→[Animal];
                                   (Object)→[Food];
                                }.
```

Pour simplifier l'écriture, nous utiliserons l'expression de "définition par conditions typiques", et une "définition" ne désignera pas implicitement une définition complète mais une définition par conditions nécessaires et/ou suffisantes ou typiques.

4.2.2.4.4 Individus et prototypes

Sowa (1984) permet également de définir des individus ainsi que des "prototypes pour un type" i.e. des descriptions d'individus typiques de ce type. Par exemple[1]:

individual CIrcus_Elephant (Jumbo) is
[Elephant : Jumbo]←(Agent)←[Perform]→(Location)→[Circus : GrussCircus].

prototype for Elephant(x) is
[Elephant : *x]- { (Characteristic)→[Hight : @3.3m];
(Characteristic)→[Weight : @5400kg];
}.

Notons que de telles définitions n'ont pas été retenues dans le modèle de base tel qu'il est défini dans (Chein & Mugnier 1992a).

1. Les "référents numériques", tels que ceux de cet exemple, sont introduits dans la section suivante.

4.3 Extensions au modèle de base

Seul le noyau défini ci-dessus est actuellement implémenté dans CoGITo (Haemmerlé, 1995). Mugnier & Chein (1996) définissent les extensions suivantes au modèle de base défini dans (Chein & Mugnier, 1992a) et étendent en conséquence les opérations élémentaires de spécialisation/généralisation et la projection :
1) les GCSs ne sont pas nécessairement connexes ;
2) l'ensemble des types de concepts forme un ordre partiel, muni d'un plus grand et d'un plus petit élément, mais n'est pas nécessairement un treillis ;
3) l'ensemble des types de relations est muni d'une structure qui est un ordre partiel.
La correspondance entre projection et déduction logique sur les formules associée est conservée, et le modèle est complété avec une sémantique ensembliste.

Un GCS correspond à une formule logique conjonctive, positive, fermée existentiellement et ayant pour seuls termes des variables ou des constantes. Cela est insuffisant pour pouvoir répondre aux besoins d'applications réelles. Sowa (1984) propose d'augmenter la capacité descriptive et déductive des GCs en introduisant d'autres sortes de référents, ainsi que des types (de concepts ou de relations) d'ordre supérieur. Il propose également d'ajouter des propriétés dynamiques aux GCs en introduisant des sommets appelés "acteurs" représentant des fonctions.

4.3.1 D'autres types de référents

Pour Sowa, un concept contient une *partie "type"* et une *partie "référent"*.

La partie type peut recevoir le *nom d'un type* ou bien une *lambda-abstraction* qui définit un type.

La partie référent peut contenir :
1) *un ou plusieurs référents* de natures différentes, et
2) des *notations simplifiées* permettant de raccourcir l'écriture du graphe : une telle notation symbolise un ensemble de concepts et des relations reliés au concept qui contient cette notation. Par une opération d'expansion, le graphe complet peut être reconstitué.

Outre un référent générique ou individuel, d'autres types de référents peuvent être utilisés : ensembliste, graphe, numérique, chaîne de caractère ou encore image. Par exemple :
[Graph : [Personne : Jean]←(Agent)←[Aller]→(Destination)→[Ville : Nice]].
[String : "ceci est une chaîne"].
[Number : 2] .

Sowa n'a pas donné d'interprétation formelle et ontologique bien définie à ces nouveaux types de référent. Par ailleurs, ils peuvent être vus comme des référents complétant le référent générique ou individuel. Par exemple :
[Graph : *x [Personne : Jean]←(Agent)←[Aller]→(Destination)→[Ville : Nice]].
[String : #246 "ceci est une chaîne"].
[Number : #568 2] .

Sowa rend le plus souvent implicite le référent individuel ou générique lorsqu'il utilise d'autres types de référents. Nous n'utiliserons pas cette facilité dans ce rapport.

Esch & Levinson (1995) proposent le principe suivant pour étendre la relation de spécialisation sur les GCSs aux CGs qui ont, outre le référent individuel ou générique, une combinaison de référents quelconques (GC, chaîne de caractères, icône, etc.) : étant donné deux combinaisons C1 et C2 de référents, C1 est plus spécifique (ou spécialisé) que C2 si C1 a au moins autant de référents que C2 et si chaque référent de C1 est égal ou plus spécifique que celui de même rang dans C2. Par exemple :
[Personne : Jean "Jean"] ≤ [Personne : * "Jean"] ≤ [Personne : *]
[Graph : * [Personne : Jean] ←(Agent)←[Aller]] ≤ [Graph : * [Personne : Jean]] ≤ [Graph : *]

Les référents graphes et les référents ensemblistes sont importants pour la représentation des langues naturelles ou des connaissances en général. Nous présentons tout d'abord les premiers puis plus rapidement les seconds car ils ne sont pas implémentés dans CoGITo ni dans CGKAT.

4.3.2 Les référents graphes

Afin de pouvoir représenter des relations portant sur des graphes, c'est-à-dire pour représenter des connaissances sur des connaissances, il est nécessaire de représenter chacun de ces graphes ou ce qu'il signifie par un concept. Un tel concept peut "comporter un graphe dans sa partie référent"[1]. Ce concept, qui est alors appelé un ***contexte***[2], constitue pour le graphe qu'il inclut, un cadre syntaxique qui permet de définir la portée des variables, des quantificateurs ou des modalités.

De manière générale, pour Sowa (1995a), un contexte a trois fonctions :
1) une fonction syntaxique en groupant ou délimitant une portion de texte ou de graphe ;
2) une fonction sémantique en référant à une entité ou une situation réelle ou imaginaire, ou à une autre expression dans un langage naturel ou artificiel ;
3) une fonction pragmatique en permettant à certaines opérations de s'appliquer sur la partie de texte ou de graphe distinguée.
Selon Moulin (1995), les contextes peuvent être utilisés de trois façons :
1) pour partitionner de la connaissance et ainsi pouvoir la traiter,
2) pour quantifier ou nier un ensemble de connaissances ou exprimer des modalités sur cet ensemble,
3) pour exprimer des relations entre un agent et les connaissances qu'il communique, et pour positionner ces connaissances communiquées par rapport à d'autres (pour traiter ou communiquer des connaissances, un agent construit un "espace mental").

4.3.2.1 Notion de contextualisation

Pour notre part, nous distinguons deux sortes de contextes :
1) ceux qui "*contextualisent*" le(s) graphe(s) qu'ils contiennent, c'est-à-dire ceux dans lesquels ces graphes inclus ne peuvent être considérés comme vrais que dans l'environnement défini par le graphe qui les inclut (i.e. le graphe qui comprend le concept contexte, ce graphe pouvant lui-même être contextualisé), et
2) ceux qui ne contextualisent pas le(s) graphe(s) inclus.

Par exemple, pour Sowa, un GC situé dans la partie référent d'un concept de type Proposition, et auquel aucune relation n'est accrochée, a la même interprétation logique que s'il était représenté directement : [Proposition : * G1] <=> G1. Nous considérons alors que G1 n'est pas contextualisé. Nous dirons que le type Proposition n'est pas un "type de contexte contextualisant".

Par contre, dans les exemples suivants, nous considérons que G1 est contextualisé :

[Hypothèse : * G1].	(Past)→[Proposition : * G1].
(Neg)→[Proposition : * G1].	[NegatedProposition : * G1].
[Personne]←(Agent)←[Croire]→(Object)→[Proposition : * G1].	[BelievedProposition : * G1].

En effet, nous considérons :
- Hypothèse, NegatedProposition et BelievedProposition comme des "types de contextes contextualisants",
- Past et Neg comme des "types de relations contextualisantes", et
- Croire comme un type de "processus contextualisant son objet".

---

1. Nous utilisons cette expression plutôt que le terme de "référent graphe" à cause des notations simplifiées définies plus loin qui permettent qu'un GC qui se trouve dans la "partie référent" d'un concept ne soit pas en réalité un référent de ce concept.

2. Le mot "**contexte**" a de nombreux sens dans la littérature ; **dans cette section, il désigne uniquement un concept qui a un GC dans sa partie référent**. Ainsi, par "type de contexte", nous désignerons le type d'un concept qui peut admettre un GC dans sa partie référent.

Une relation connectée à un contexte n'a pas toujours pour rôle de le contextualiser. Par exemple : [Situation: *x]→(Description)→[Proposition: *y G1]. (G1 a une interprétation logique normale)

Pour représenter ces notions de contextualisation, l'ontologie proposée par CGKAT offre les types Contextualizing_proposition, Contextualizing_relation et Process_contextualizing_its_object. L'utilisateur peut spécialiser ces types avec les types de son application de manière à définir ceux qu'il considère comme des "types de contextes contextualisants", des "types de relations contextualisantes" ou encore des "types de processus contextualisant leurs objets".
Dans CGKAT, afin de gérer la contextualisation des graphes, une gestion spéciale est associée aux types Contextualizing_proposition, Contextualizing_relation, Process_contextualizing_its_object et à leurs sous-types : lorsque CGKAT doit présenter un graphe en réponse à une requête, il vérifie tout d'abord si ce graphe est emboîté *et contextualisé* par d'autres graphes, et si tel est le cas, il présente le graphe résultat avec (i.e. emboîté dans) les graphes qui le contextualisent. Cela sera détaillé plus loin.

Notons par ailleurs que les graphes :
[Personne]←(Agent)←[Croire]→(Object)→[Proposition : * G1].
et : [BelievedProposition : * G1].
peuvent se dériver l'un de l'autre par une opération de contraction ou d'expansion de type, si le type BelievedProposition est ainsi défini :
NCS for BelievedProposition (x) are [Personne]←(Agent)←[Croire]→(Object)→[Proposition : *x].

Une interprétation logique spéciale peut être définie pour différentes sortes de contextualisation. Par exemple, pour la négation : ϕ((Neg)→[Proposition : *x G1]) = $\neg$(ϕ(G1)).

Différents sortes de modalité peuvent être exprimées sur un GC via les graphes qui le contextualisent (Sowa, 1984) :
- des modalités "aléthiques" (modalités de la vérité) comme la possibilité ou la nécessité ;
- des modalités déontiques comme la permission ou l'obligation ;
- des modalités épistémiques comme la croyance ou le fait d'être connu par un agent ;
- des modalités temporelles comme celles dénotées par les termes "jamais", "parfois" et "autrefois".
D'autres systèmes logiques que ceux de la logique du premier ordre peuvent donc être nécessaires pour donner une interprétation logique à certaines sortes de contextualisation et les gérer (e.g. une logique modale ou une logique temporelle).

Mugnier & Chein (1996) ont étendu les opérations de spécialisation/généralisation à une certaine forme de GCs emboîtés (cf. section 4.3.2.3.4). Mais si une interprétation logique spéciale est définie pour une relation ou un type, ces opérations sont également à redéfinir. Par exemple, dans le cadre de l'interprétation logique précédente, on ne peut généraliser (Neg)→[Proposition : *x G1] par [Proposition : *x G1] qui est équivalent à G1.
Des systèmes d'inférence particuliers peuvent également être définis pour permettre de gérer certaines sortes de contextualisation, par exemple pour gérer la négation ou les aspects temporels.

Sans interprétation logique ou gestion particulière, un GC contextualisé peut néanmoins être recherché indépendamment de son niveau d'emboîtement (e.g. par projection) puis être présenté ou manipulé en prenant en compte cet emboîtement. C'est l'option que nous offrons dans CGKAT.

Dans l'état de l'art actuel, deux approches d'interprétation et de gestion des contextes peuvent être distinguées (nous détaillons ces différentes approches les deux sections suivantes) :

1. Celle dans lequel le graphe inclus dans un contexte est considéré comme une définition du type de ce concept contexte.
2. Celle où le contexte est considéré comme *pouvant* contextualiser le(s) graphe(s) qu'il inclut, et donc comme étant un module, dans le sens où nous avons défini ce terme. Par défaut, c'est-à-dire sans analyse du type du contexte ou des types des relations qui y sont accrochées, ces graphes sont considérés comme vrais uniquement dans le contexte qui les inclut.

Dans les deux sections suivantes (4.3.2.2 et 4.3.2.3) nous présentons un aperçu de ces différentes gestions, puis dans la section 4.3.2.4, nous détaillons une ontologie possible des types de contextes modules (Sowa, 1992).

4.3.2.2 Interprétation et gestion des contextes comme des définitions de type

Pour Esch (1994a), un contexte peut s'obtenir à partir d'un concept simple (i.e. sans GC dans sa partie référent), et vice-versa, par une opération de contraction ou d'expansion. Pour cela, le graphe d'un contexte doit correspondre au corps de la définition (par conditions nécessaires et suffisantes) du type du concept. Ainsi, pour un type défini *t* dont le corps de la définition est *gc_differentia* :
ϕ([t : *x gc_differentia]) = $\exists$x (t(x) $\wedge$ ϕ(gc_differentia) $\wedge$ x=gc_differentia)

Esch définit un opérateur d'expansion χ (et inversement, un opérateur de contraction) :
ϕ(χ ([t : *x])) = $\exists$x (t(x) $\wedge$ ϕ(gc_differentia) $\wedge$ x=referent_graphe(genus(gc_differentia)))
Grâce à ceci : 1) χ([t : *x]) = [t : *x gc_differentia]
et 2) [t : *x] <=> [t : *x gc_differentia] (ces deux formes représentent le même individu et peuvent se dériver l'une de l'autre par contraction ou expansion).

Exemple : [She-Monkey : *x] <=> [She-Monkey : *x [Monkey]→(Characteristic)→[Sex : F]].

Ces définitions et cet opérateur s'appliquent de manière similaire pour un individu x défini par la construction : individual t (x) is gc_differentia.

4.3.2.3 Interprétation et gestion des contextes comme des modules

Esch (1994b) précise les notions de dominance, portée et coréférence dans les contextes modules.

4.3.2.3.1 Un contexte "module" permettant de définir des types locaux à ce module

Esch (1994b) spécifie un contexte particulier ayant la structure et les fonctionnalités d'un "canon" (Sowa, 1984), i.e. il spécifie un module ayant quatre composants :
1) une hiérarchie de types de concepts,
2) un ensemble de marqueurs individuels,
3) une relation de conformité reliant les marqueurs aux types, et
4) une base de graphes canoniques (i.e. des graphes représentant des situations réelles ou possibles dans le monde représenté).

Ainsi, un contexte "canon" peut contenir des définitions de types, d'individus et de graphes canoniques. Esch note qu'un canon forme un monde clos consistant et complet pour les "règles d'inférence propositionnelles" (Sowa, 1984) et qu'il est donc comparable aux micro-théories de Guha (1991) dans le fait qu'il constitue un système formel sur lesquel s'appliquent des règles consistantes et complètes par rapport à la logique du 1er ordre. Esch (1994b) suggère une façon d'étendre la fonction ϕ pour les canons.

4.3.2.3.2 Gestion de la négation de contextes avec les "règles d'inférence propositionnelles"

Dans Sowa (1984), seul un concept de type Proposition peut admettre un graphe comme référent et être nié avec une relation particulière de type Neg. Par la suite, Sowa utilisera de la même façon d'autres types de concepts que Proposition.
Voici l'interprétation logique donnée par Sowa (1984)[1], sg_1, ..., sg_n étant des GCs quelconques :

ϕ([Proposition : {sg_1, ..., sg_n}]) = $\phi(sg_1) \wedge ... \wedge \phi(sg_n)$
ϕ((Neg)→[Proposition : {sg_1, ..., sg_n}]) = $\neg$ ($\phi(sg_1) \wedge ... \wedge \phi(sg_n)$)

Ainsi, selon Sowa (1993a), la phrase "Tous les chats sont sur un tapis" peut être représentée par :
(Neg)→[Proposition : [Cat : *x] (Neg)→[Proposition: [Cat : *x]→(On)→[Mat]]].

De même, on peut représenter la phrase "Si un chat est sur un tapis, il est heureux" par :
(Neg)→[Proposition : [Cat : *x]→(On)→[Mat]
(Neg)→[Proposition : [Cat : *x]→(Attr)→[Happy]]].
Ou avec une notation simplifi ée, sous forme de règle : IF: [Cat : *x]→(On)→[Mat]
THEN: [Cat : *x]→(Attr)→[Happy].

Les concepts de type Proposition pouvant ainsi être emboîtés ***et*** niés à la manière de "boîtes de Peirce", Sowa reprend le système "Beta" de Peirce pour proposer une gestion de tels emboîtements. Ce système Beta est un ensemble consistant et complet de règles d'inférence par rapport à la logique du premier ordre (Roberts, 1973). Les adaptations de ces règles pour les GCs sont nommées par Sowa les "*règles d'inférence propositionnelles*". Wermelinger (1995a) a apporté des corrections à ces règles afin que ce système soit complet, et il a corrigé la définition de la fonction ϕ donnée par Sowa pour les GCSs et les emboîtements afin de la rendre consistante et complète y compris lorsque des types d'ordre supérieur sont utilisés. Par ailleurs, les règles d'inférence propositionnelles ont été raffinées par Esch & Levinson (1995) afin qu'elles puissent gérer des graphes n'appartenant pas à un même contexte, et elles ont été raffinées par Heaton & Kocura (1995) pour améliorer l'expressivité des GCs et résoudre certains problèmes liés notamment à l'utilisation de types d'ordre supérieur.

Notons que Esch & Levinson (1995) proposent une structure de données qui implémente la relation de spécialisation sur les GCs et permet d'exprimer qu'un GC a été introduit positivement ou bien négativement, ou encore les deux à la fois. L'ordre des GCs dans cette structure est impliquée :
1) par des "règles de formation canoniques" (i.e. les règles élémentaires de spécialisation/généralisation" présentées en section 4.2.2.3) ;
2) par les "règles d'inférence propositionnelles",
3) par les inférences logiques pouvant être faites mais non impliquées par les "règles d'inférence propositionnelles".

Divers autres travaux étudient des systèmes d'inférences logiques sur les GCs, ou les liens entre les GCs et d'autres systèmes de logique. Citons par exemple (Ghosh & Wuwongse, 1995), (Wuwongse & Manzano, 1993), (Kabbaj & al, 1994) et (Wermelinger, 1995a).

4.3.2.3.3 Gestion "orientée objet" des contextes

Sowa (1993b) propose que tout concept puisse admettre des GCs dans sa partie référent afin d'encapsuler a) des détails sur la composition de l'objet représenté, et b) des règles pour représenter des méthodes pouvant s'appliquer sur l'objet. Pour gérer ces règles et les états produits par leur applications (ces états étant représentés sous forme de contextes), Sowa utilise
1) les "règles d'inférence propositionnelles" (quel que soit le type du concept contexte), et
2) des "règles d'inférence orientées objet" (il en définit six).

1. Notons que cette interprétation ne prend pas en compte de possible liens de coréférence entre des concepts de différents graphes.

Ellis (1995) propose une autre version de ce modèle dans laquelle les méthodes sont définies avec des relations sur les concepts, et il utilise d'autre règles que celles de Sowa pour gérer ces méthodes. Selon Ellis, sa version serait "plus claire et plus efficace pour la programmation et les preuves de programme, car basée sur une simple transition d'états".

4.3.2.3.4 Gestion des contextes par la projection

Mugnier & Chein (1996) proposent une extension du modèle de base des GCSs en définissant une certaine forme de GCs emboîtés : les "graphes de description". Dans ces graphes, chacun des concepts a trois champs : un type de concept, un référent individuel ou générique, et un troisième champ qui peut recevoir a) un graphe de description d'un individu, ou b) l'étiquette "**" qui indique une description générique. Par exemple :

[Accident : #268 [Voiture : *x [Pneu]→(Attr)→[Usé]]].
[Accident : * **].

Un graphe de description a pour objectif de représenter des notions structurées par niveau, et de raisonner par niveau. Une projection et des opérations élémentaires de spécialisation ont été définies en utilisant cette notion de niveau d'un graphe de description (à chaque niveau, on peut associer un GCS). L'idée générale est qu'un graphe de description peut se spécialiser en spécialisant le graphe d'un de ses niveaux, et étant donné deux graphes de description, l'un se projette dans l'autre si pour les mêmes niveaux, la projection peut se faire sur les GCSs associés. Ainsi par exemple :

[Accident : #268 [Voiture : *x [Pneu]→(Attr)→[Usé]]] ≤ [Accident : * [Voiture : * **]].

Les graphes de description ont actuellement une sémantique logique consistante et complète pour la projection.

4.3.2.3.5 Gestion de contexte ad-hoc pour prendre en compte certains types de contextualisation

Sowa (1992) propose une ontologie relative aux types de contextes et à leurs inter-relations, cela afin de guider la représentation de connaissances et notamment celle des modalités. Néanmoins, cette ontologie, présentée dans la section suivante, se situe à un niveau trop général pour permettre de spécifier des mécanismes d'inférence adaptés à différents types de contextualisation. De tels mécanismes ne peuvent souvent être spécifiés que dans le cadre d'une application.

Moulin (1995a) pense que le formalisme des GCs et l'ontologie de Sowa (1992) permettent de représenter de façon adéquate des situations s'inscrivant dans le temps et décrites par des phrases isolées, mais que ce formalisme et cette ontologie doivent être étendus pour pouvoir représenter et traiter de manière adéquate des phrases ou des portions de phrases faisant référence les unes aux autres : en effet, dans de tels cas, des problèmes de perspective ou de référence dans le temps peuvent intervenir, e.g. avec "Hier, Jean pensait que demain il ferait beau" et "hier, Jean le pensait".

Moulin (1995a) a donc étendu le formalisme des GCs et l'ontologie de Sowa pour permettre une meilleure représentation des connaissances temporelles, en y ***intégrant*** plusieurs éléments de la "théorie des actes de langage" (Searle, 1985). Il introduit en effet des *concepts spéciaux* (situation temporelle, localisation temporelle et perspective) qui comportent *trois champs* (type, contenu propositionnel et intervalle temporel). Sa modification du formalisme des GCs lui permet de définir une notion de contexte intéressante pour représenter et traiter les connaissances temporelles et leurs inter-références dans des perspectives de divers agents (Moulin, 1995a) (Moulin, 1995b). En contre-partie, ce nouveau formalisme est plus contraignant et il faut redéfinir pour de tels GCs les règles élémentaires de spécialisation/généralisation et les règles d'inférence propositionnelles.

4.3.2.4 Une ontologie pour les types de contextes modules

Pour créer cette ontologie, Sowa (1992) s'est inspiré de la "sémantique des situations" (Barwise & Perry, 1983), qui selon lui a été largement adoptée comme l'un des moyens les plus flexibles de définir la sémantique du langage naturel. De nombreux travaux ont été faits dans le cadre de cette théorie (Barwise & al., 1991). Contrairement à l'approche de Montague (1974) qui relie la sémantique du langage naturel à des modèles potentiellement infinis du monde réel ou de mondes possibles, la "sémantique des situations" est basée sur des situations finies. Voici un extrait de Sowa (1992) où il définit son ontologie[1] :

> Chaque *situation* est une configuration finie d'un aspect du monde dans une région limitée de l'espace et du temps. Une situation peut être une configuration statique qui demeure inchangée pendant une période donnée, ou elle peut inclure des processus et des événements qui causent des changements. Elle peut inclure des gens et de choses avec leurs actions et leurs attributs ; elle peut être réelle ou imaginaire, et peut être présente, passée ou future. Une situation peut être aussi grande que le système solaire ou aussi petite qu'un atome, et elle peut contenir des sous-situations qui décrivent des aspects plus petits et plus détaillés du monde.
>
> Dans les graphes conceptuels, une situation est représentée par un *contexte*, qui est un concept qui contient une ou plusieurs propositions qui décrivent la situation. Les propositions dans un contexte peuvent être exprimées par un paragraphe de phrases en anglais ou par une collection de graphes conceptuels. ... La relation entre une situation et chacune des propositions qui la décrit est *Description* (Dscr) ; la relation entre une proposition et le graphe qui l'introduit est *Statement* (Stmt).

Ainsi, trois types d'objets sont exhibés :
1) les situations d'un monde réel ou imaginaire,
2) les descriptions et donc les interprétations qu'un être humain, ou plus généralement un agent, peut faire d'une situation, et
3) les media dont il se sert pour réaliser une description (supports, langages, etc.).

Voici pour illustrer les liens entre situations, descriptions et media, deux graphes extraits de Sowa (1992) et représentant le fait qu' (i.e. la situation d') un chat est sur un tapis :

[Situation]→(Descr)→[Proposition]→(Stmt)→[Graph : [Cat]→(On)→[Mat]].
[Situation]→(Descr)→[Proposition]→(Stmt)→[Sentence : "A cat is on a mat"].
("Descr" signifi e "Description", "Stmt" signifi e "Statement")

On retrouve là des distinctions déjà faites par Aristote et reprises par toutes les disciplines s'intéressant aux signes : la sémiotique, la linguistique, la psychologie, la philosophie, la représentation des connaissances, etc. Ogden & Ridchards (1923) ont codifié ces distinctions dans le "triangle sémiotique". Le voici avec ses différentes terminologies associées (les termes originaux d'Ogden &

1. Each *situation* is a finite configuration of some aspect of the world in a limited region of space and time. It may be a static configuration that remains unchanged for a period of time, or it may include processes and events that are causing changes. It may include people and things with actions and attributes; it may be real or imaginary; and its time may be present, past, or future. A situation may be as large as the solar system or as small as an atom, and it may contains nested situations that describes smaller and more detailed aspects of the world.

In conceptual graphs, a situation is represented by a *context*, which is a concept that contains one or more propositions that describe the situation. The propositions in a context could be expressed by a paragraph of English sentences or by a collection of conceptual graphs. ... The relation between a situation and a propositions that describe it is *Description* (Dscr); the relation between a proposition and the graph that states it is *Statement* (Stmt).

Ridchards sont en italique, ceux de Sowa sont en italique gras) :

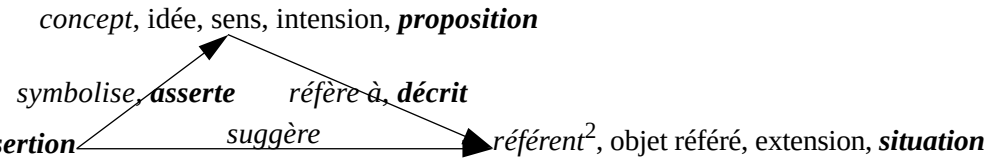


Figure 4.3: *Le triangle sémiotique.*

Notons cependant à propos de cette figure qu'une situation n'est pas n'importe quel "objet référé" : c'est un état ou un processus, c'est-à-dire des choses qui peut arriver dans le temps. Barwise & Perry (1983) proposent des règles d'inférence gérant des attitudes propositionnelles (voir, croire, savoir, douter, déclarer, etc.) sur les situations[1].

Reprenons les deux graphes représentant le fait qu' (i.e. la situation d') un chat est sur un tapis :
[Situation]→(Descr)→[Proposition]→(Stmt)→[Graph : [Cat]→(On)→[Mat]].
[Situation]→(Descr)→[Proposition]→(Stmt)→[Sentence : "A cat is on a mat"].

Le graphe, comme la phrase, est un support permettant de représenter une idée, une assertion, une interprétation ou encore une description d'une partie d'un monde.
De manière plus générale, il nous semble que tout élément de document peut jouer ce rôle (phrase, graphe, image, paragraphe, section, chapitre, etc.). Il est important lors de la représentation d'un élément de document, de distinguer trois notions : 1) l'élément lui-même, 2) sa signification pour son auteur, et 3) la situation que cet auteur décrit. Nous reviendrons sur ces distinctions dans la section 6.3.1.3.1.
Une situation peut être décrite par diverses propositions plus ou moins détaillées et non forcément équivalentes, et une proposition peut être formulée avec divers media, e.g. un GC, une formule logique du premier ordre, une phrase en langage naturel, une image ou un document entier.

Nazarenko (1992) note que les distinctions conceptuelles de l'ontologie de Sowa sont intuitives et que
a) il est possible de traduire une "proposition" (connaissance propositionnelle et donc non temporelle) en logique et de lui affecter une valeur de vérité à partir d'une base de connaissances,
b) une situation peut s'interpréter par rapport à un modèle du monde.
Elle regrette toutefois que ces distinctions et les relations qui les lient ne soient pas mieux définies formellement. Notamment elle s'interroge sur les cas où il faut utiliser un concept de type Proposition et ceux où un concept de type Situation est préférable. Le problème concerne ici davantage la machine que l'homme. Nous montrerons que l'ontologie de CGKAT facilite l'utilisation de l'ontologie de Sowa pour l'homme comme pour la machine, car
1) elle précise un peu cette ontologie de haut niveau,
2) elle spécialise cette ontologie avec les 90.000 catégories conceptuelles de WordNet,
3) elle propose une hiérarchie de types de relations signées sur les types de concepts généraux de cette ontologie.

Pour Sowa (1992), le graphe : [Proposition: *y [Cat]→(On)→[Mat]].
est une notation simplifiée de : [Proposition: *y]→(Stmt)→[Graph : [Cat]→(On)→[Mat]].
et le graphe :
[Situation : *x [Cat]→(On)→[Mat]].
est une notation simplifiée de :
[Situation : *x]→(Descr)→[Proposition: *y]→(Stmt)→[Graph : [Cat]→(On)→[Mat]].
Ainsi dans Sowa (1992), seul un concept de type Graph peut réellement admettre un "référent graphe".

1. Par exemple, à partir de "X doute que Y ou bien Z soit dans telle situation" on peut déduire que "X doute que Y soit dans telle situation" et que "X doute que Z soit dans telle situation", tandis que à partir de "X voit que Y ou bien Z est dans telle situation" on déduit que "il est faux que X voit que Y et Z sont dans telle situation".

2. "Référent" au sens d'Ogden & Ridchard, pas au sens de Sowa !

En utilisant de telles notations simplifiées, un concept de type Situation (ou d'un de ses sous-types) peut être utilisé pour contextualiser des GCs. Ainsi, et compte-tenu de l'ontologie de Sowa, voici comment la phrase "Hier, Jean croyait que Marie voulait se marier avec lui" peut être représentée :

[Time : #hier]←(Point_in_time)←[Situation :
[Person: Jean]←(Agent)←[Believe]→(Object)→[Proposition:
[Situation : [Person : Marie]←(Agent)←[Want]→(Object)→[Proposition :
[Situation : [Person : Marie]←(Agent)←[Marry]→(Object)→[Person: Jean]
]]]]].

Sowa (1992) considère les concepts de type Proposition contenant un graphe en partie référent comme des notations simplifiées, mais ne modifie pas l'interprétation logique de ces concepts :

ϕ([Proposition : {sg_1, ..., sg_n}]) = $\phi(sg_1) \wedge ... \wedge \phi(sg_n)$

ϕ((Neg)→[Proposition : {sg_1, ..., sg_n}]) = $\neg$ ($\phi(sg_1) \wedge ... \wedge \phi(sg_n)$)

Les "règles d'inférence propositionnelles" peuvent donc toujours s'appliquer.
Cependant Sowa ne donne aucune interprétation logique pour les concepts de type Situation ou pour les relations autres que la négation.

Sowa (1993b) adopte une approche plus "orientée objet" : comme tout objet peut être *vu plus précisément* comme une situation (l'objet n'est alors plus considéré comme une entité atomique, mais son contenu est examiné), Sowa (1993b) propose que tout concept quel que soit son type, puisse être décrit plus précisément par une ou plusieurs propositions, c'est-à-dire, *en utilisant les notations simplifiées*, que tout concept puisse admettre des GCs dans sa partie référent afin d'encapsuler des détails sur sa composition. Les "règles d'inférence propositionnelles" peuvent s'appliquer sur de tels contextes. Sowa reprend ainsi implicitement pour tout contexte l'interprétation logique donnée pour les contextes de type Proposition

4.3.2.5 L'approche retenue dans CGKAT

Les GCs emboîtés ne sont pas implémentés dans la version actuelle de CoGITo. Or, ils nous ont semblé importants pour permettre l'acquisition de connaissances et la représentation des éléments de documents et de leurs emboîtements. Nous référons ici aux contextes (i.e. aux concepts comportant un graphe dans leur partie référent) considérés comme des modules et non aux contextes considérés comme des définitions de types puisque dans ce dernier cas, de véritables définitions de types peuvent être utilisées. Aussi, avons-nous implémenté une gestion des contextes modules.

4.3.2.5.1 Principes

Nous n'avons voulu implémenter que les types de modules nécessaires, et non ceux qui peuvent conduire le cogniticien à ne pas représenter explicitement des relations entre objets, comme c'est par exemple le cas dans la représentation suivante :

[Accident : #268
[Cause: * [VitesseExcessive]] [Voiture : *x [Pneu]→(Attr)→[Usé] [Conducteur : Jean]]
]→(Point_in_time)→[Time : #270296].

Il est difficile de comparer un tel graphe à d'autres graphes qui ne suivent pas exactement le même modèle : même ordre et mêmes relations implicites. L'absence de relations réduit également les possibilités de vérification, de recherche et d'exploitation de ce genre de graphe. Une représentation "à plat", lorsqu'elle est possible, est préférable. En reprenant l'exemple ci-dessus :

[Accident : #268]- { (Point_in_time)→[Time : #270296] ;
(Cause)→[VitesseExcessive] ;
(ImpliquéeDans)← [Voiture]- { (Part)→[Pneu]→(Attr)→[Usé] ;
(Conducteur)→[Personne: Jean] ;
}
}.

Nous n'avons pas non plus voulu implémenter des contextes modules pouvant contenir des types ou des individus locaux au contexte. La notion d'environnement dans CoGITo peut répondre, au moins en partie, à ce besoin.

Notre implémentation a été guidée par l'ontologie de Sowa (1992), dont nous avons retenu les points décrits ci-dessous, en appelant "entités" les objets qui ne sont pas des situations (i.e. des états ou des processus).
Représenter le fait qu'une entité existe ou donner sa composition, c'est décrire une situation, c'est-à-dire représenter une proposition (*une description de situation*) dans une base de connaissances. Une entité ne peut donc être décrite qu'en décrivant une situation. Pour décrire une situation, une proposition utilise un medium ou support : graphe, phrase, image, etc. Ainsi, si l'on représente explicitement les diverses relations entre situations, propositions et media, un seul type de contexte semble nécessaire pour la représentation des connaissances : Graph ou bien Proposition (selon la terminologie de Sowa). Nous considérons le type Proposition comme un candidat pour être le seul type de contexte nécessaire car aucune "contextualisation" ne s'effectue directement sur un concept de type Graph : c'est ce qu'exprime un GC qui peut être contextualisé, i.e. une proposition, et non le GC lui-même. Par exemple, des relations "contextualisantes", comme Or, Neg, ou les relations qui expriment des modalités (e.g. déontiques, épistémiques, temporelles), portent sur des propositions et non des supports de propositions (graphe, phrase, image, etc.).

Dans le cadre de CGKAT, nous avons choisi le type Proposition pour ne pas obliger l'utilisateur à emboîter ses GCs dans un concept de type Graph lorsqu'il représente des propositions.
Dans CGKAT seul un concept de type Proposition (ou d'un de ses sous-types) peut admettre un GC dans sa partie référent. Les notations simplifiées ne sont pas permises. Cela a deux avantages :
1) le sens d'un contexte dans CGKAT est unique : un contexte est un concept qui représente la signification du graphe qu'il emboîte ;
2) cela évite l'encapsulation de graphes dans des concepts de type autre que Proposition et donc cela conduit l'utilisateur à expliciter les relations entre ses concepts.

Pour aider l'utilisateur dans la création de relations entre concepts, CGKAT fournit une ontologie de types de relations signées sur des types de concepts de haut niveau (e.g Entity, State, Process, Proposition) et, étant donné un concept sélectionné par l'utilisateur, il peut exploiter les signatures des types de relations pour lister toutes celles qui peuvent être connectées à ce concept (la liste est hiérarchisée par la relation de spécialisation entre les types). L'utilisateur peut alors sélectionner le type répondant à ses besoins et demander la création d'une relation de ce type. Les signatures des relations permettent également un certain contrôle de la sémantique des GCs construits par l'utilisateur.

Dans CGKAT, tous les GCSs, emboîtés ou non, sont représentés de la même façon. Cependant, CGKAT stocke le lien entre un contexte et le graphe qu'il emboîte. Il peut donc l'exploiter pour les recherches et les inférences. L'utilisateur peut spécialiser les types Contextualizing_proposition, Contextualizing_relation et Process_contextualizing_its_object pour indiquer à CGKAT des types de concepts ou de relations qu'il considère comme contextualisants. Ainsi, lorsqu'un graphe emboîte un GCS, CGKAT peut déterminer si ce graphe emboîtant est contextualisant. Lorsqu'un GCS contextualisé doit être affiché en réponse à une requête, il est présenté à l'intérieur du ou des graphes qui le contextualisent. Par exemple, si un utilisateur cherche à connaître les accidents causés par Jean, en recherchant les spécialisations du graphe requête "[Accident]→(Causé-par)→[Personne : Jean]", et si la base de graphe contient le graphe suivant, ce graphe sera présenté en entier :
(Neg)→[Proposition : #prop259 [Accident]→(Causé-par)→[Personne : Jean]].

Nous décrirons ces aspects de manière plus détaillée dans les sections 5.6.5.4, 6.3.2.2.3 et 7.2.3.3. Nous n'avons implémenté aucune autre gestion spéciale pour gérer la contextualisation ; par exemple, CGKAT ne gère pas la négation avec des règles d'inférence propositionnelles. Seuls quatre noms de types sont inclus en dur dans le code de CGKAT : Proposition, Contextualizing_proposition, Contextualizing_relation et Process_contextualizing_its_object. Ces types sont donc en quelque sorte prédéfinis, mais l'utilisateur peut leur donner les supertypes ou les sous-types qu'il souhaite dans son ontologie.

4.3.2.5.2 Implémentation des GCs emboîtés dans CGKAT

Résumons notre approche :
1) un contexte est un concept qui représente la signification du graphe qu'il emboîte,
2) tous les GCSs, emboîtés ou non, sont représentés de la même façon dans la base de CGKAT,
3) chaque GCS a un nom unique qui est généré automatiquement ou donné par l'utilisateur.

Aussi, le lien entre un GCS et le concept l'emboîtant peut être implémenté et exprimé de façon simple et explicite : en donnant au concept un référent individuel dont l'étiquette est le nom du GCS emboîté.

Ainsi, introduire le graphe suivant dans la base de CGKAT
(Neg)→[Proposition : #CatOnMat [Cat]→(On)→[Mat]].
est équivalent à
1) introduire [Cat]→(On)→[Mat], puis
2) donner le nom de CatOnMat à ce graphe, puis
3) déclarer CatOnMat comme un individu de type Proposition, et enfin
4) introduire (Neg)→[Proposition : #CatOnMat]. // (note: '#' ne fait pas partie du nom de l'individu)

Dans CGKAT, le format linéaire des graphes est celui des GCSs. Aussi pour construire :
(Neg)→[Proposition : #CatOnMat [Cat]→(On)→[Mat]].
il faut écrire, si l'on utilise le format linéaire :
name #CatOnMat [Cat]→(On)→[Mat].
(Neg)→[Proposition : #CatOnMat].

Cependant, en utilisant les langages S et P de Thot, nous avons défini un modèle structurel et un modèle de présentation qui permet de construire et d'afficher dans un format graphique un GC contenant plusieurs niveaux. Pour permettre cela, de manière similaire aux graphes de description de (Mugnier & Chein, 1996), un concept est défini comme ayant trois sous-éléments : un type, un référent individuel ou générique, et un graphe (les concepts de ce graphe peuvent à leur tour emboîter des graphes). Lorsqu'un utilisateur demande la création d'un graphe dans la partie référent d'un tel concept (ce concept est de type Proposition ou d'un de ses sous-types), deux cas se présentent :

1. le concept est un concept générique : CGKAT génère un nom unique pour le graphe et transforme le concept générique en un concept individuel en utilisant le nom du graphe pour créer l'étiquette du référent individuel (ainsi le concept ne représente que la signification du graphe qu'il emboîte) ;
2. le concept est un concept individuel : le graphe est nommé en utilisant l'étiquette du référent individuel du concept.

4.3.2.5.3 Extension aux définitions de types

Nous avons également permis qu'un concept de type Proposition (ou un de ses sous-types) emboîte une définition de type par conditions nécessaires et/ou suffisantes ou typiques. Cela est important pour permettre la représentation de phrases comportant des définitions ou des associations typiques, nécessaires ou suffisantes entre des objets. Voici deux exemples :

1) Représentation de "John pense que rédiger est une tâche complexe" :
[Person : John]←(Believer)←[Proposition : #NC1_Rediger
NC for Rediger (x) are [TacheComplexe : *x].
].

2) Représentation de "Les champignons sont généralement bons mais généralement dangereux" :
[Proposition : #TC1_for_Champignon
TC for Champignon_Bon (x) are [Champignon : *x]→(Attr)→[Bon_au_gout]
]→(RST_Contrast)→[Proposition : #TC2_for_Champignon
TC for Champignon_Dangereux (x) are [Champignon : *x]→(Attr)→[Dangereux]
].

4.3.3 Les référents ensemblistes

Sowa ne donne pas de définition formelle de la partie référent d'un concept, en utilisant une grammaire BNF par exemple. Il utilise des exemples. Voici les différentes sortes de référent ensembliste introduites par Sowa (1984) et Sowa (1992).

- Collectif : tous les éléments de l'ensemble participent ensemble aux relations attachées au concept contenant ce référent. Exemples :
  "Des personnes dansent ensemble" : [Person: Col{*}]←(Agent)←[Dance].
  "Dix personnes dansent ensemble" : [Person: Col{*}@10]←(Agent)←[Dance].
  "Bill et Mary dansent ensemble" : [Person: Col{Bill,Mary}]←(Agent)←[Dance].
  "Un groupe de 5 personnes dont Bill et Mary dansent ensemble" : [Person: Col{Bill,Mary}@5].

- Distributif : chaque élément de l'ensemble participe séparément aux relations attachées au concept contenant ce référent. Exemples :
  "Des personnes dansent séparément" : [Person: Dist{*}]←(Agent)←[Dance].
  "Deux étudiants lisent chacun 3 livres" :
  [Student: Dist {*}@2]←(Agent)←[Read]→(Object)→[Book: Dist {*}@3].

- Défaut : ce référent ne précise pas si les éléments de l'ensemble participent ensemble ou non aux relations attachées au concept contenant ce référent. Exemple :
  "Des personnes dansent" : [Person: {*}]←(Agent)←[Dance].

- Disjunctif : l'un des éléments de l'ensemble et lui seul participe aux relations attachées au concept contenant ce référent. Exemple (noter le séparateur ' | ') :
  "Bill ou bien Mary dansent" : [Person: Dist {Bill | Mary}]←(Agent)←[Dance].

- Respectif : chaque élément de l'ensemble participe respectivement avec les éléments d'un autre ensemble aux relations attachées au concept contenant ce référent. Exemple :
  "Bill et Mary dansent respectivement avec Pierre et Jacques" :
  [Person: Resp {Bill,Mary}]←(Agent)←[Dance]→(Object)→[Person: Resp{Bill,Mary}].

- Cumulatif : tout l'ensemble est considéré comme une seule entité. Exemple :
  "Le groupe formé par Bill et Mary comprend 2 éléments" :
  [Person: Cum {Bill,Mary}]→(Characteristic)→[Cardinality : 2].

Sowa (1992) donne la traduction en logique de ses exemples. Fargues (1992) note que l'interprétation du référent ensembliste chez Sowa est plutôt méréologique, c'est-à-dire que "la notion de référent ensembliste chez Sowa est plutôt une notion de collection, fondée sur une primitive unique du type Partie-de (collection)".

Dans (Gardiner & al.,1989) et (Tjan & al., 1992), les auteurs proposent une notation plus précise et plus unifiée du champ référent et des référents ensemblistes en particulier. Leur interprétation du référent ensembliste est formellement définie et "fondée sur une interprétation en théorie des ensembles" (Fargues, 1992).

Selon Fargues (1992), "la notation de Sowa est proche de l'interprétation naturelle des langues et l'on peut espérer assez facilement construire des graphes comprenant des types 'ensemble distributif ou collectif' par programme à partir de phrases. En revanche, passer à la notation de Gardiner directement à partir des langues semble beaucoup plus difficile. Cette notation convient mieux aux applications concernant l'ingénierie des connaissances ou les bases de données. Il serait donc judicieux de garder les deux variantes notationnelles, quitte à définir formellement des règles de réécriture de la notation de Sowa dans celle de Gardiner".

4.3.4 Les types d'ordre supérieur

Afin de pouvoir représenter des connaissances sur des types et non seulement sur des individus, Sowa (1993a) propose d'écrire des GCs utilisant des types d'ordre supérieur, i.e. des types dont les instances sont des types. Il propose pour cela les types de relations conceptuelles "Kind" et "Subt", et les opérateurs τ et ρ.

La relation de type Kind relie deux concepts, le type du premier étant instance du type du second.
Exemples : [Rectangle : *x]→(Kind)→[Shape : rectangle].
[T : *x]→(Kind)→[Type : t]. //Sowa nomme T le supertype de tous les types du 1er ordre
//et Type le supertype de tous les types du second ordre.

La relation de type Subt relie deux concepts dont les référents sont deux types dont le second est un sous-type du premier. Par exemple : [Type : entity]→(Subt)-→[Type : plant].

Les opérateurs τ et ρ permettent de traduire le nom d'une instance d'un type supérieur dans un nom du type correspondant (type de concept pour τ, type de relation pour ρ). Par exemple, τ permet de traduire "rectangle", une instance du type du second ordre "Type", en "Rectangle", un type du premier ordre.

Voici comment Sowa (1993a) modélise la transitivité d'une relation :
(Neg)→[Proposition : [Relation : *r]→(Attr)→[Transitive]
(Neg)→[Proposition :
(Neg)→[Proposition : [T : *x]→(ρ r)→[T : *y]→(ρ r)→[T : *z]]
(Neg)→[Proposition : [T : *x]→(ρ r)→[T : *z]]
]]].

La relation "Greater-than" peut donc être représentée comme transitive de la façon suivante, à condition qu'elle soit associée à "greater-than" individu instance du type "Type" :
[Relation : greater-than]→(Attr)−−>[Transitive].

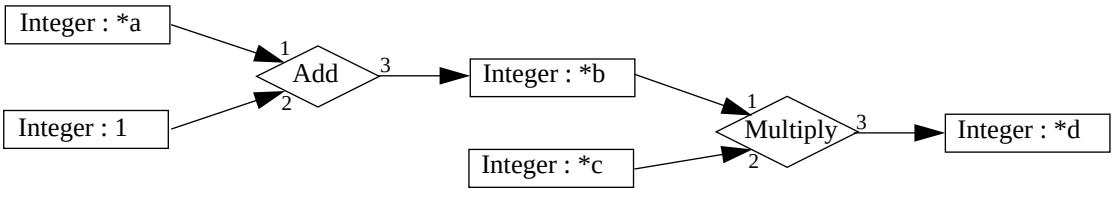
Sowa (1993a) utilise ces types d'ordre supérieur pour représenter avec des GCs les contraintes sur les relations utilisées dans les modèles Entité-Relation.

Sowa ne fournit pas d'interprétation logique particulière pour ces types d'ordre supérieur, ce que corrige Wermelinger (1995a) en conservant une interprétation en logique du premier ordre. Par ailleurs, Heaton & Kocura (1995) proposent une notation plus unifiée dans laquelle il n'est pas utile d'utiliser les opérateurs τ et ρ. Ils raffinent les "règles d'inférence propositionnelles" pour améliorer l'expressivité des GCs et résoudre certains problèmes liés notamment à l'utilisation de types d'ordre supérieur.

4.3.5 Les acteurs

Les acteurs sont des représentations de fonctions permettant d'effectuer des calculs sur les valeurs que peuvent prendre des référents de concepts. Ainsi, les GCs ne forment pas un système uniquement déclaratif : ce système peut être procédural.

Un acteur relie des concepts par des arcs entrants et sortants, et peut exécuter une fonction de calcul sur les valeurs des référents numériques de ses concepts entrants pour instancier le référent numérique de son concept sortant. Voici une représentation graphique d'un GC contenant deux acteurs, "Add" et "Multiply" : .



L'acteur peut être déclenché de deux façons : soit en instanciant le référent numérique d'un de ses concepts génériques entrants avec une valeur et une marque de requête particulière, soit en posant une marque de requête dans le référent numérique de son concept sortant. Dans les deux cas, si tous les référents numériques de ses concepts entrants sont instanciés, le calcul est fait, sinon ces référents sont instanciés à leur tour par une marque de requête et le calcul est fait lorsque ces référents sont tous instanciés par les résultats des sous-requêtes. Sowa (1984) propose une méthode pour propager des sous-requêtes, ou inversement, des résultats.

Pfeiffer & Hartley (1992) ont implémenté et raffiné ces acteurs pour CP, leur environnement de représentation de connaissances basé sur le formalisme des GCs.
Lukose (1993) a fait de même pour l'environnement UNE-CG-KEE (Munday & al., 1994). Les acteurs de Lukose sont similaires à des sources de connaissances dans un "blackboard", de par leur structure et leur possibilité de s'envoyer des messages entre eux. Ils ont des pré-conditions et des post-conditions sous forme de GCs, des méthodes, une mémoire à court terme et une mémoire à long terme. D'autres GCs particuliers peuvent être exécutés : par leurs emboîtements et les types des relations qui relient les concepts de ces GCs, des plans ou des successions d'actions peuvent être représentés et exécutés.

4.4 Applications des GCs

4.4.1 Analyse de la langue naturelle

Le formalisme des GCs présente de nombreux avantages pour l'analyse de la langue naturelle, et donc pour l'extraction de connaissances à partir de documents.
Citons par exemple la possibilité de représenter de manière explicite et lisible certaines finesses de la langue naturelle, l'existence de contextes, et la gestion de la coréférence tant dans les GCSs que dans les contextes emboîtés. Selon Sowa (1995a), les "règles d'inférence propositionnelles" de Peirce permettent de gérer les contextes et les coréférences (résolutions d'anaphores) de la même façon que dans la "Théorie de la Représentation du Discours" (Kamp (1991). Par ailleurs, des contraintes et connaissances prototypiques (e.g. les structures casuelles associées aux actions) peuvent être exprimées grâce aux schémas et aux signatures des relations. Cela permet l'analyse sémantique d'une phrase, et comme le montre Amghar & al. (1995), cela permet de reconnaître et traiter des problèmes de métonymie (i.e. des figures de langage dans lesquelles un mot est employé à la place d'un autre, les concepts liés à ces mots étant nécessairement reliés par une relation, e.g. cause/effet, contenant/contenu, "signe référant à un objet"/"objet référé", producteur/produit, matière/objet).

De nombreux travaux ont exploité ou étendu le formalisme des GCs pour l'analyse de la langue naturelle (e.g. Moulin, 1995). Nous présentons brièvement ci-dessous quelques systèmes utilisant les GCs, notamment les ontologies exploitées par ces systèmes. Une description plus détaillée de CARAMEL, KALIPSOS et MENELAS se trouve dans (Haemmerlé, 1995).

- Le système CARAMEL (Sabah & Briffault, 1993) est destiné à la compréhension automatique de récits en langue française et est conçu pour pouvoir s'intégrer dans divers types d'application manipulant la langue naturelle, comme des interfaces en langues naturelles, de l'indexation automatique de document, du résumé automatique de texte, de génération de textes, etc.
- Le système KALIPSOS (Berard-Dugourd & al., 1989) a également pour objet la compréhension de textes écrits en français, ainsi que la recherche de phrases via un module question/réponse. Ce système utilise un lexique sémantique composé d'un lexique général et d'un lexique spécifique au domaine utilisé. Le lexique général comprend un treillis de types de concepts, un ensemble de 70 types de relations (essentiellement de lieu, de temps, de comparaison et de causalité), un ensemble de graphes définissant 1) les mots courants en fonction des différents sens qu'ils peuvent prendre, 2) les schémas prototypique des verbes, 3) les ordinaux, mesures, dates, etc.

- Le projet MENELAS (Consortium Ménélas, 1994) vise à faciliter l'accès au informations contenues dans les compte-rendus d'hospitalisation en texte libre (en français, anglais ou néerlandais). En Octobre 1994, MENELAS utilisait un arbre de 1430 types atomiques de concept, un arbre de 314 types de relations, 484 types de concepts linguistiques définis, 84 types de relations linguistiques et 315 prototypes.
- Toujours dans le domaine médical, le projet GALEN vise à intégrer différents systèmes terminologiques pour supporter le développement et l'intégration de systèmes cliniques. Ce projet utilise les GCs pour analyser des textes médicaux écrits dans différentes langues (Rassinoux & al, 1994), et générer des textes ou des interfaces dans différentes langues également (Wagner & al., 1995). Pour cela, les types de l'ontologie sont reliés à différents dictionnaires (un par langue).
- DR-LINK (Myaeng 1992) est un système de recherche de documents en cours de développement, où les documents sont représentés par des GCs. L'extraction automatique des concepts et relations des textes s'effectue grâce à une base de 12.000 structures casuelles pour les actions et un grand nombre d'autres schémas linguistiques pour révéler les relations conceptuelles (Myaeng & Khoo, 1994).

Soulignons également que l'analyse de ressources lexicales peut se faire en prenant en compte la structure des documents sources. Un cas extrême est celui de l'extraction de connaissances à partir de dictionnaires. Barrière (1995) décrit une application exploitant les GCs dans ce domaine.

Plus une phrase doit être précisément (sémantiquement) représentée (ce qui n'est pas toujours nécessaire, par exemple en recherche documentaire (Myaeng, 1992)), plus l'analyseur doit disposer d'un grand nombre de connaissances dans le domaine traité. Il n'y a pas, à ce jour, de systèmes de compréhension de textes suffisamment général et puissant pour pouvoir être utilisé comme module d'acquisition de connaissances (PRC-GDR_IA, 1995).

4.4.2 Structuration de connaissances

Ellis (1995) note que les GCs permettent une représentation intuitive d'objets complexes, et qu'ils peuvent être recherchés par leur contenu (e.g. par projection) plutôt que par leur nom ou leur adresse.

La structure de donnée implémentant la relation de spécialisation sur les GCs présents dans une base s'appelle la hiérarchie de spécialisation (ou de généralisation). Cette hiérarchie peut être exploitée pour accélérer la recherche des graphes qui spécialisent un GC donné, ou plus généralement la recherche des graphes qui peuvent comparés, unifiés ou déduits. Elle est également un bon support pour les méthodes d'apprentissage qui exploitent la manière dont des parties d'un objet se combinent pour créer l'objet entier (e.g. descente de gradient et régression linéaire).

Cette structure de donnée peut être améliorée pour augmenter encore la vitesse et les possibilités de comparaison de GCs. Ainsi, Levinson (1994) décrit UDS (Universal Data Structure) qui comprend une hiérarchie de concepts (génériques et individuels), une hiérarchie des relations individuelles et une hiérarchie des GCs, ainsi que des liens bidirectionnels entre ces hiérarchies. Suivant l'exploitation de UDS (e.g. suivant les graphes dont on cherche les spécialisations), l'une ou l'autre de ces hiérarchies est exploitée. De plus, les CGs sont compactés et représentés avec des tables de relations. Ainsi, Levinson (1994) propose des méthodes d'accès qui compilent les méthodes utilisées dans les bases de données relationnelles, les réseaux sémantiques, les GCs, RETE, et les bases de cas. Il montre également l'intérêt d'UDS pour la recherche dans des espaces d'états, et pour des méthodes d'apprentissage.

Pour certaines classes de GCs, il est également possible d'accélérer leur recherche en leur affectant à chacun un code (la comparaison des codes est bien plus rapide que la comparaison de graphes). Pour un GCS, Ellis & Lehmann (1994) calculent son code suivant sa place dans la hiérarchie de spécialisation, tandis que Cogis & Guinaldo (1995) proposent une fonction de hash-code sur ses étiquettes.

D'autres travaux proposent une normalisation des GCs afin de pouvoir mieux et plus rapidement les comparer, e.g. Fall (1995). Par ailleurs, d'autres métriques pour comparer ou structurer des graphes sont étudiées, e.g. Poole & Campbell (1995), Willems (1995) et Champesme (1995). Des stratégies d'intégration de graphes sont présentées dans (Dieng, 1996).

La structuration d'une base de GCs peut également être effectuées avec des méthodes d'apprentissage, notamment avec les méthodes de "regroupement conceptuel" (conceptual clustering) qui visent à reconnaître des régularités parmi un ensemble d'objets qui n'ont pas été pré-classé. La méthode MSG de Mineau (1990) fait partie de ces méthodes. Elle construit un ordre partiel sur les triplets (Concept,Relation,Concept) contenus dans des GCSs décrivant des objets. Chaque élément de cette hiérarchie relie donc un ensemble de caractéristiques (sous forme de triplets) aux objets qui le possèdent, et peut se révéler être un type de concept naturel. La méthode MSG peut donc être utilisée pour construire une ontologie. Mineau & Allouche (1995) montrent que cette méthode peut aider à la désambiguïsation sémantique d'un type de concept, à l'enrichissement de l'ensemble de ces types, et à l'identification de primitives ontologiques (tout ceci étant intéressant pour l'intégration de différentes ontologies ou la construction d'une base de connaissances). Bournaud & Ganascia (1995) présentent une méthode pour extraire de la hiérarchie générée par MSG, un arbre de classification, et appliquent cette méthode à la classification de caractères chinois.

4.4.2.1 Comparaison avec KL-ONE

Tout comme le formalisme des GCs, KL-ONE (Brachman & Schmolze, 1985) est un système de représentation de connaissances qui a une audience internationale, qui appartient à la lignée des réseaux sémantiques à héritage, et qui permet de déduire si un type en subsume un autre. Ces deux systèmes possèdent des primitives de structuration semblables : concepts génériques et individuels, définitions de types par conditions nécessaires et suffisantes, relations de subsomption et d'instantiation.

Biebow (1992) effectue une comparaison entre le modèle de base des GCs et le modèle de base de KL-ONE. Dans ce dernier, contrairement au modèle de base des GCs,
1) la hiérarchie des types, les descriptions de types et les descriptions d'individus sont connectés dans un même réseau ;
2) un concept est classé automatiquement dans le réseau, de façon optimale en fonction des propriétés le définissant (notons que des classifications similaires pourraient également être effectuées avec le formalisme des GCs ; Leclère (1995) détaille des possibilités et des problèmes de la classification des types dans le formalisme des GCs) ;
3) les concepts génériques sont quantifiés universellement tandis que les concepts individuels sont quantifiés existentiellement, et seul les concepts génériques peuvent représenter des propriétés quantifiés universellement.
Par ailleurs, Biebow note que les GCs permettent de représenter la négation, l'implication et les finesses des langues, en gardant une relative clarté syntaxique.

4.4.3 Acquisition de connaissances

Les sections précédentes ont montré que le formalisme des GCs est a) un formalisme de représentation de connaissances assez général et qui est notamment intéressant pour représenter le langage naturel, et b) doté de mécanismes de structuration et de recherche qui semblent intéressants pour l'acquisition de connaissances. Ce formalisme semble particulièrement adapté pour formaliser les connaissances du domaine, de par son aspect déclaratif et son aptitude à traiter les connaissances terminologiques, ces deux caractéristiques étant importantes pour que le système puisse fournir des explications sur ses connaissances ou son raisonnement (Moeller, 1995). Des extensions du formalisme des CGs permettent à la fois de représenter **et** d'exécuter des connaissances opérationnelles, c'est-à-dire dans le vocabulaire de KADS, des inférences, des méthodes ou des tâches.

Ainsi, Lukose (1995) montre comment grâce aux GCs et à son implémentation des acteurs, il est possible de représenter et tester un modèle d'expertise KADS. Les connaissances du domaine sont

modélisées et représentées avec la hiérarchie des types de concepts, les définitions de ces types, et des "règles conceptuelles" (GCs formés de deux concepts exprimant chacun un état et décrit par un GC, reliés par une relation conceptuelle exprimant une causalité). Une inférence est représentée par un acteur (les acteurs de Lukose ont la structure et les fonctionnalités des sources de connaissances dans les "blackboards"). Les structures d'inférence et les structures de tâche sont représentées avec ses GCs exprimant des plans d'actions.

Moeller (1995) note l'importance d'une conception modulaire en acquisition de connaissance pour faciliter la gestion et la représentation explicite et formelle des connaissances et de leurs inter-relations. Il définit alors une représentation exécutable d'un modèle KADS en utilisant des canons pour chaque groupe de connaissances (du domaine ou inférence ou tâche) et des acteurs pour importer et exporter entre ces modules (e.g. entre une inférence et les connaissances du domaine qu'elle utilise). Les tâches sont représentées par des règles exprimées sous forme de GCs. Il regrette cependant de n'avoir pu implémenter un modèle KADS de cette manière, faute d'outils implémentant les canons et un moteur d'inférence exploitant les connaissances de ces canons et gérant l'héritage entre ceux-ci.

Suivant des principes similaires, Moeller & Willems (1995) redéfinissent dans le formalisme des GCs, le langage de représentation de spécifications fonctionnelles DESIRE. Ce langage est fondé sur les notions de "déclarativité, compositionalité, réflection et exécutabilité" (Langevelde & al., 1992) et représente les données comme le contrôle de manière déclarative en utilisant une logique de premier ordre typée.

Lukose & al. (1995) font le point sur les forces et faiblesses des GCs pour l'AC. Quatre idées peuvent être extraites de cet article.

1. Par rapport aux autres formalismes classiquement utilisés en AC, les GCs sont intéressants car ils permettent de représenter dans un seul formalisme les différents modèles des différentes phases de l'AC (modèles de tâche, de coopération, d'expertise, de conception) et les différentes connaissances de ces modèles (e.g. l'abstrait et le spécifique des méthodes de résolution de problèmes)[1]. Les GCs peuvent être utilisés comme des représentations graphiques et peu formelles durant la phase d'élicitation (i.e. comme un version plus contrainte des "Concept maps" (Kremer 1995)), puis formalisés durant la phase de modélisation des connaissances (ceci permettant par exemple la détection de connaissances incohérences ou manquantes), et enfin rendus exécutables durant la phase de conception. La simulation du modèle d'expertise ou la création de prototype au cours de la construction du modèle d'expertise est ainsi facilitée. Par ailleurs, la conservation de la structure du modèle d'expertise pour l'implémentation du système final, est importante pour la qualité des explications que ce système peut générer.
2. Il faut intégrer aux GCs des extensions qui soient acceptées par l'ensemble de la communauté et implémentées dans des outils disponibles : représentation et gestion des quantificateurs, des ensembles, des contraintes, des GCs emboîtés et de la classification et de l'héritage sur de tels GCs, des structures de contrôle (e.g. structures conditionnelles et boucles), etc. Notons cependant que beaucoup d'extensions nécessaires aux GCs sont des problèmes généraux en représentation des connaissances que l'on peut traiter différemment suivant les besoins.
3. Le formalisme des GCs (simples et avec extensions) devrait être comparés à d'autres formalismes proches (KL-One, autre formalisme de graphes, etc.) et aux langages de modélisation de connaissances classiquement utilisés en AC : ses forces, faiblesses et conditions d'applications devraient être explicités, par exemple avec des tests reconnus par la communauté en acquisition, comme ceux du projet Sisyphus_II (KAW, 1994). Une telle évaluation est en cours pour le système UNE-CG-KEE (Munday & al., 1994).
4. Un outil permettant de gérer des GCs (simples et avec extensions) devrait être implémenté et mis à la disposition de ceux qui veulent utiliser ou tester ce formalisme. C'était l'ambition du projet

1. Notons cependant que certaines représentations à base d'objets permettent également cela.

Peirce (PEIRCE, 1994) mais la diversité des applications actuelles ou possibles des GCs (et donc des extensions apportées ou apportables à ce formalisme) rend difficile la réalisation d'un langage et d'un outil accepté par tous. Pour être réellement utilisé en acquisition des connaissances, un tel outil devrait inclure un éditeur graphique de GCs, des outils pour manipuler des ontologies, et plusieurs BCs, une interface en langage naturel, et des outils pour supporter les diverses phases de l'AC. Il devrait également être fourni avec une bibliothèque de connaissances partageables et réutilisables : ontologies générales (i.e. de haut niveau) et catalogue de GCs basiques ou "canoniques".

Les méthodes de regroupement conceptuel, e.g. la méthode MSG de Mineau (1990), permettent de générer une hiérarchie de classes d'objets (cf. section 4.4.2).
De façon complémentaire, compte-tenu d'une taxinomie de types de concepts donnée, Aïmeur & Ganascia (1993) proposent une méthode et un outil d'élicitation auprès d'un expert pour
1) construire les descriptions de ces types de concepts et raffiner la taxinomie,
2) construire parallèlement un arbre de discrimination sur ces types de concepts (i.e. quelles caractéristiques ou valeurs de caractéristique permettent de différencier les instances des sous-types direct d'un type).
L'élicitation est incrémentale et descendante (de la racine vers les feuilles) et se fait par un jeu de questions réponses : pour un type à décrire, l'outil fait des propositions compte-tenu de la description de son supertype direct et de ses type frères.
Leclère (1995) a également défini un outil aidant par un jeu de questions réponses à placer un type atomique ou défini dans une hiérarchie de types de concepts ou de relations, et vérifiant si la hiérarchie reste, avec l'insertion du type, un treillis (pour les types de concept) ou un ordre partiel (pour les types de relation).

4.5 Environnements de travail pour manipuler des GCs

Les principaux outils permettant de manipuler des GCs et actuellement disponibles sont : CP (Pfeiffer & Hartley, 1992), Deakin ToolSet (Garner & al., 1992), Loughborough ToolSet (Heaton & Kocura, 1993), UNE-CG-KEE (Munday & al, 1994), Peirce (Ellis, 1993a, 1993b) (PEIRCE, 1994) et CoGITo (Haemmerlé, 1995a).

Le projet Peirce (PEIRCE, 1994) était la collaboration des chercheurs de toute origine géographique pour la mise en oeuvre d'un environnement de travail sur les graphes conceptuels qui soit robuste, portable et librement disponible. Ce projet semble cependant à présent abandonné.
L'outil actuel (Ellis, 1993a, 1993b) est du domaine public.
Il dispose d'un langage de commandes permettant de construire dans une base, une hiérarchie de types de concepts, une hiérarchie de types de relations et des GCs dont les concepts contiennent soit un type et un référent individuel ou générique, soit un graphe emboîté (le concept ne contient alors ni type ni référent individuel). Des définitions de types peuvent être écrites en utilisant de tels graphes. Les acteurs et les types d'ordre supérieur ne peuvent être utilisés. Le langage de commandes permet également de comparer deux graphes par projection injective, d'obtenir une jointure maximale ou une généralisation commune de deux graphes, et de rechercher les spécialisations ou les généralisations d'un graphe. Pour accélérer la réalisation de ces opérations, la relation de spécialisation sur les graphes d'une base est stockée dans une structure de données (Levinson & Ellis, 1992). Cette relation de spécialisation est limitée au cas de la projection injective. Les opérations s'effectuent en mémoire vive (une base de donnée en mémoire centrale n'est pas gérée).
Les sources de cet outil sont disponibles mais rien n'est actuellement prévu pour faciliter le développement d'applications fonctionnant sur les données manipulées par l'outil : il n'existe pas d'interface fonctionnelle pour cela et les commandes ne peuvent être appelées depuis un programme.

Au contraire, CoGITo (Haemmerlé, 1995a, 1995b) est une "plate-forme de développement de logiciels sur les graphes conceptuels" : elle n'offre pas de langage de commandes mais une interface

fonctionnelle pour construire et manipuler une base de GCSs. Elle se présente comme un ensemble de classes d'objets C++ dont les méthodes permettent de construire, de modifier, de sauvegarder ou encore de comparer les instances de ces classes. Voici les classes principales : type de concept, treillis de types de concepts, types de relations, hiérarchie de types de relations, concept, relation, graphe conceptuel. Cette dernière classe offre deux méthodes pour une jointure simple et dirigée (jointure externe et jointure interne), une méthode pour un isojoint "avec γ *maximal au sens de l'inclusion*" (cf. section 4.2.2.3.2) et deux méthodes pour la projection : l'une permet de projeter n'importe quel GCS dans un autre GCS, et a été implémenté selon un principe de "retour en arrière" (Salvat, 1993), l'autre permet de projeter un GCS ayant une structure d'arbre dans un autre GCS selon un algorithme polynomial (Mugnier & Chein, 1992).
Les opérations s'effectuent en mémoire vive et la sauvegarde des données peut s'effectuer dans des fichiers au format BCGCT (Haemmerlé, 1995a).
CoGITo permet la création des définitions de types par conditions nécessaires et suffisantes pour les types de concepts et les types de relations mais ne prend pas en compte ces définitions pour ordonner ces types et ne vérifie pas que les types de concepts sont ordonnés selon une structure de treillis. Ces deux points font partie des fonctionnalités de C-CHIC (Leclère, 1995), un outil basé sur CoGITo et destiné à guider la construction de hiérarchies de types atomiques ou définis.
CoGITo permet également d'associer des définitions par conditions nécessaires ou bien typiques à des types de concepts. Nous avons ajouté la possibilité d'associer des définitions par conditions suffisantes à des types de concepts.

4.5.1 Des interfaces graphiques

Sowa (1984) a défini des notations graphiques pour les concepts, les relations, les acteurs, les contextes et la coréférence. Des notations graphiques sont proposées par Esch (1992) pour les types, les quantificateurs, les modules, les ensembles et l'auto-référence, et par Esch (1991) pour les relations temporelles et intervalles temporels.

Plusieurs interfaces graphiques ont été développées pour permettre d'éditer graphiquement des GCs et d'effectuer des opérations sur ces derniers, e.g. création, copie, jointure sur un concept, chargement, sauvegarde. Citons GRIT (Leane, 1993), qui communique avec PEIRCE par échange de GCs au format linéaire, et l'interface de Guichard (1994) qui communique avec CoGITo par échange de fichier au format BCGCT ou bien en mode client/serveur. GRIT permet de générer une représentation graphique d'un treillis de types (Leane discute des problèmes liés à cette génération et des limites de la méthode qu'il utilise), mais aucune de ces deux interfaces ne génère de représentation graphique pour un GC.

4.5.1.1 Affichage automatique de graphes

Cyre & al (1994) distinguent trois types d'affichage pour les GCs : littéral, schématique et imagé. Dans l'affichage littéral, les noeuds concept ou relation sont représentés par deux types d'icônes (rectangle et ovale) et sont reliées par des flèches. Dans l'affichage schématique, différents types prédéfinis d'icônes et de connecteurs sont utilisés suivant leur sémantique, e.g. action ou valeur pour les icônes, et contrôle ou dépendance de donnée pour les connecteurs. Dans l'affichage imagé, des icônes variés et complexes peuvent être utilisés, les connecteurs peuvent être invisibles, et la position des icônes peut dépendre de la sémantique des concepts qu'ils représentent et de leurs inter-relations (e.g. si deux concepts sont reliés par une relation de type "Above", ils vont être graphiquement représentés l'un au dessus de l'autre). Par ailleurs, l'affichage peut être statique ou bien dynamique, i.e. animé (les changements et les durées des séquences étant décrits par des GCs). Cyre & al (1994) ont développé un prototype pour l'affichage statique littéral ou schématique de GCSs (dans un certain domaine), et travaillent sur les autres possibilités, ainsi que sur la mise à jour des GCSs lorsque leurs représentations graphiques sont éditées.

Une règle de base pour les algorithmes d'affichage automatique de graphes est de minimiser les croisements d'arêtes. Burrow & Eklund (1995) présentent un langage de description d'affichage de

GCs permettant d'exprimer plus de contraintes dans l'affichage de GCs, et présentent un algorithme utilisant ces contraintes pour générer la représentation graphique d'une jointure de deux GCs.

4.6 Conclusion

Nous avons présenté le modèle de base des GCs qui a l'avantage d'avoir une interprétation consistante et complète en logique du premier ordre et qui permet de structurer les GCSs par une relation de spécialisation.

Cependant, un GCS correspond à une formule logique conjonctive, positive, fermée existentiellement et ayant pour seuls termes des variables ou des constantes. Cela est insuffisant pour pouvoir répondre aux besoins d'applications réelles. Nous avons décrit un certain nombre d'extensions proposées par Sowa (1984) pour augmenter la capacité descriptive et déductive des GCs : 1) extension de la partie référent des concepts avec des référents graphes et des référents ensemblistes, 2) types d'ordre supérieur, et 3) "acteurs".

La possibilité de créer des graphes emboîtés est particulièrement importante car elle permet de représenter des notions de contexte. Nous avons distingué les emboîtements destinés à "contextualiser" des connaissances, de ceux simplement destinés à les isoler pour les manipuler mais sans en changer l'interprétation logique. Dans le premier cas, une interprétation logique et une gestion spéciale peuvent être associées à différents types de contextualisation. Par exemple, Sowa (1984) assigne une interprétation logique spéciale pour la relation unaire "Neg" sur un concept "contexte" et propose des "règles d'inférence propositionnelles" pour gérer cette négation dans des graphes emboîtés.
Dans CGKAT, nous avons été guidé par l'ontologie de Sowa (1992) pour implémenter une certaine forme de graphes emboîtés destinée à conduire les utilisateurs à expliciter les relations entre les concepts. Le chapitre suivant montre comment l'ontologie générale proposée par défaut par CGKAT guide l'utilisateur dans la construction des graphes. Les contextualisations de graphes sont prises en compte lors de la présentation de ces graphes en réponse à une requête : des graphes contextualisés sont toujours présentés à l'intérieur des graphes qui les emboîtent. L'utilisateur peut spécialiser les types Contextualizing_proposition et Contextualizing_relation et Process_contextualizing_its_object pour indiquer à CGKAT des types de concepts ou de relations qu'il considère comme "contextualisants".

Les extensions au modèle de base permettent de l'utiliser comme une notation semi-formelle, graphique ou linéaire. Cette notation permet de représenter dans un seul modèle, différentes sortes de connaissances et peut permettre des traitements ad-hoc. C'est pourquoi les GCs ont été jusqu'à présent surtout utilisés pour l'analyse du langage naturel.
Ils ont été également utilisés pour interfacer l'accès à des bases de données de manière à faciliter 1) la modélisation ou la recherche d'informations dans des systèmes d'informations (Creasy & Moulin, 1992) (Carbonneill & Haemmerlé, 1994b), et 2) l'intégration de bases de données via la traduction de leur schémas conceptuels dans le formalisme des Graphes Conceptuels (Creasy, 1994).
Pour l'acquisition des connaissances, Lukose & al. (1995) soulignent entre autres :
1) l'intérêt de formaliser et d'implémenter dans un outil disponible les diverses extensions actuelles, et de permettre l'exécution de connaissances par exemple des règles, des structures de contrôle ou des acteurs ;
2) l'intérêt pour un tel outil d'offrir une interface graphique et/ou en langage naturel, des ontologies et des moyens de manipuler ces ontologies.

Dans CGKAT, les GCs peuvent être construits avec un langage de commandes ou dans un format graphique. L'éditeur de document structurés est combiné avec la plate-forme de gestion de GCSs CoGITo et est utilisé comme une interface graphique au dessus de la base de GCs. Des menus facilitent la visualisation et la gestion de l'ontologie de l'application. Enfin, une ontotologie générale de types de concepts et de types de relations est fournie.