PARTIE II

LE TRAVAIL RÉALISÉ

Chapitre 5 Des ontologies pour guider la construction d'une base de connaissances

5 Sommaire

5.1 Introduction

Une ontologie est un ensemble de *termes formels* permettant de représenter des connaissances et auxquels peuvent être associés des définitions. Ces définitions permettent :
1) de poser des contraintes sur l'utilisation des termes (notamment les définitions par conditions nécessaires et/ou suffisantes) et ainsi d'effectuer des vérifications syntaxiques ou sémantiques ;
2) de guider l'association de certains termes avec d'autres termes (notamment les schémas prototypiques) et ainsi de guider l'extraction et la modélisation des connaissances (par exemple, nous considérons les modèles de tâches génériques comme des schémas prototypiques associés à certains types de tâches génériques).

Les termes d'une ontologie peuvent également être utilisés comme des moyens de sélectionner les connaissances dans lesquelles ils sont utilisés. Ils peuvent donc servir de critères de sélection, i.e. de points de vue. Les termes pouvant être organisés, différents points de vue sur les connaissances peuvent être structurés.

Nous avons noté au chapitre 2, que dans les outils généraux d'AC actuels,
1) les langages permettant de construire les modèles de tâches ne sont pas encore suffisamment génériques : ils intègrent des primitives ontologiques qui devraient plutôt être offertes dans une ontologie afin de pouvoir être spécialisées et surtout complétées ;
2) seules des ontologies de tâches ou de méthodes sont généralement offertes, mais pas d'ontologie de haut niveau, ni d'ontologie de moyen niveau (e.g. grandes taxonomies 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 l'outil d'AC pour faciliter et vérifier ces tâches.

Le formalisme des GCs n'impose que l'usage de concepts et de relations, leurs types et les marqueurs conformes à ces types devant être définis dans un support (que nous assimilons à une ontologie) composé de trois parties :
1) un treillis de types de concepts,
2) un ensemble de types de relations signées et pouvant être organisés hiérarchiquement, et
3) un ensemble de marqueurs individuels ou génériques, représentant des individus et devant chacun être reliés à un type par une relation de conformité (notons également l'existence du marqueur universel et du marqueur absurde ; cf. 4.2.2.1).

Nous décrivons dans ce chapitre, l'ontologie proposée par défaut par CGKAT à ses utilisateurs, son intérêt, et des manières de l'exploiter pour l'extraction de connaissances. Cette ontologie intègre (i.e. regroupe et interclasse) et précise des types de haut niveau provenant de diverses autres ontologies destinées à guider la représentation ou l'acquisition des connaissances. Nous montrerons également comment une grande ontologie de moyen niveau telle que WordNet peut être exploitée.

Les techniques d'accès, de visualisation, et de manipulation d'une taxinomie sont également importantes pour faciliter sa construction et son exploitation. Aussi, nous allons dans un premier temps présenter celles actuellement offertes par CGKAT, et ainsi préciser certaines conventions que nous utilisons. Dans le prochain chapitre, nous montrerons comment Thot (Quint & Vatton, 1992) pourrait être exploité pour visualiser, éditer et documenter non seulement des graphes mais également une taxinomie.

5.2 Accès, visualisation et construction de taxinomies

5.2.1 Présentation des termes

Dans une ontologie, un terme formel (type ou marqueur) dénote un sens (ou une caractéristique) unique et doit avoir une étiquette unique. Le nom sous lequel est *vu* un terme, dans l'ontologie ou dans les connaissances qui l'utilisent, doit permettre à son lecteur de deviner le sens qu'il désigne. Un terme qui comprend un seul mot est ambigu puisque la plupart des mots ont plusieurs sens. Il vaut donc mieux utiliser une liste de mots synonymes pour construire un terme. Cependant pour des raisons d'ergonomie, le nom sous lequel est vu un terme, doit aussi être court tout en restant significatif pour chaque lecteur.

Une solution est de permettre la visualisation d'un même terme sous différents noms choisis par les utilisateurs. Dans ce cas, un terme a un nom "système" (i.e. interne au système) et peut avoir un nom différent pour chaque utilisateur[1]. Nous appelons cela une visualisation en mode "utilisateur" par opposition à la visualisation en mode "système" où le nom sous lequel est présenté un terme est son nom système. CGKAT ne permet pas encore la visualisation en mode "utilisateur".

Par ailleurs, pour avoir une vision synthétique d'une partie d'une taxinomie de types, ou pour mieux comprendre le sens d'un type et ses relations par rapport aux autres types, il est souvent intéressant de visualiser *en même temps*, pour un type sélectionné, plusieurs de ses sous-types et de ses supertypes, et pour chacun de ces types, des commentaires (e.g. une définition en langage naturel). En effet, plus le nombre de types présentés est grand (et leurs commentaires visibles), moins l'utilisateur est obligé de *naviguer* dans l'ontologie, c'est-à-dire de se focaliser sur un type donné, afin d'obtenir des informations supplémentaires sur ce type (e.g. ses supertypes et ses sous-types). Il est difficile de comparer des types qui ne sont pas présentés en même temps et la navigation ne facilite donc pas l'obtention d'une vision d'ensemble. De ce fait, elle conduit aux problèmes bien connus de surcharge mentale et de désorientation. La recherche et la comparaison de types et de commentaires étant une activité fréquente du cogniticien lorsqu'il construit une ontologie ou tente de comprendre celle d'un autre cogniticien, l'outil d'AC doit favoriser les recherches et comparaisons visuelles (et donc, au besoin, la présentation en même temps d'un grand nombre d'informations *structurées* entre elles, ce qui permet de visualiser rapidement, et si nécessaire, certaines d'entre elles).

Afin de visualiser une partie de taxinomie montrant des commentaires pour les types, une liste indentée telle celles des figures 5.1, 5.3, 5.4 et 5.5, semble préférable à une visualisation sous forme de graphe tel que celui de la figure 5.2. Si les commentaires ne doivent pas être montrés et si les noms des types sont courts[2], un graphe semble préférable. Il semble donc intéressant pour un système d'offrir ces deux moyens de visualisation. Dans CGKAT, le menu de gestion des types de concepts (voir figure 5.1) et le menu de gestion des types de relations (voir figure 5.5) permettent de rechercher par navigation ou requête une portion de hiérarchie de types et de la visualiser sous forme de liste indentée. CGKAT permet également de visualiser cette portion sous forme de graphe en réutilisant un générateur de graphe fonctionnant sous Netscape (cf. figure 5.2).
Nous montrerons dans le prochain chapitre comment Thot pourrait être exploité pour permettre à un utilisateur d'éditer des hiérarchies quelconques (pas seulement des arbres), de les visualiser sous diverses formes (liste indentée, hiérarchie horizontale ou verticale), et de les connecter par des liens hypertextes à divers éléments de documents. Les recherches par navigation ou requêtes dans l'ontologie d'une application s'effectueraient alors via la génération de document temporaires (actuellement c'est une liste indentée qui est régénérée dans un menu à chaque recherche).

---

1. Pour que la conversion soit possible, un utilisateur ne doit bien sûr pas donner le même nom à deux termes différents.

2. Dans l'ontologie par défaut proposée par CGKAT, les noms de types sont longs, car ils sont souvent construits par concaténation de mots synonymes (séparés par "__") afin d'être explicites et uniques.

A notre connaissance, dans les outils d'AC actuels ou bien dans les interfaces graphiques actuelles pour des bases de GCs, les ontologies sont visualisées sous forme de listes non indentées ou sous forme de graphes générés (c'est par exemple le cas dans GRIT (Leane, 1993) et CG-editor (Moeller & Detlev, 1996)).

Lorsqu'une partie de taxinomie contient des types ayant plusieurs supertypes directs, sa visualisation sous forme de graphe comme sous forme de liste indentée, pose des problèmes. Dans le premier cas, il faut limiter les croisements des relations entre les noeuds. Dans le second cas, le même type doit apparaître dans différentes branches de la taxinomie. Dans les listes indentées présentées par CGKAT, le fait qu'un type ait plusieurs supertypes directs, est signalé par un caractère '%' précédant le nom de ce type (voir figure 5.3 à la page 109). Si ce type est celui sélectionné, c'est-à-dire celui dont l'utilisateur a désiré voir les supertypes et les sous-types, tous les supertypes directs de ce type sont présentés et sont signalés en tant que tels par le caractère '^' (voir figure 5.4 à la page 109). Les autres supertypes ne sont pas montrés car cela ne serait pas lisible. Lorsque le type sélectionné n'a qu'un supertype direct, les supertypes sont donnés (e.g. figures 5.7 et 5.8 à la page 121) à moins que l'un d'eux n'ait plusieurs supertypes directs ; dans ce cas, les supertypes de ces supertypes directs ne sont pas montrés.

Une autre solution pour montrer les supertypes d'un type sélectionné aurait été de les afficher dans un arbre séparé au lieu d'utiliser une seule liste indentée. Cela aurait résolu les problèmes d'affichage de multiples supertypes directs mais une telle solution ne nous a pas semblé suffisamment ergonomique (un effort d'interprétation est nécessaire).

Afin d'éviter à l'utilisateur des efforts inutiles de navigation dans les listes indentées de CGKAT, les types qui n'ont pas de sous-type (ou pour les types de concepts, pas d'autre sous-type que le type Absurde) sont signalés par le caractère '.' (voir figures). Par ailleurs, grâce au sélecteur "Max depth" (voir figures), l'utilisateur peut contrôler le nombre de niveaux de sous-types affichés pour un type sélectionné. Cependant, si le commentaire associé à un sous-type X du type sélectionné contient l'expression "e.g.", les sous-types de ce type X ne seront pas affichés même si le nombre de niveaux le permet. Ainsi, le créateur de ce type X peut spécifier que les sous-types de ce type ne doivent être visualisés que lorsque ce type est sélectionné. Cela permet au créateur d'une taxinomie ou d'une partie de cette taxinomie de définir un affichage pertinent ou ergonomique pour une partie de cette ontologie, c'est-à-dire l'affichage des types les plus utiles ou les plus importants de cette partie. La figure 5.1 illustre ceci en montrant les sous-types du type Concept que nous jugeons les plus importants. Dans notre ontologie, lorsque "e.g." apparaît dans le commentaire d'un type, il est le plus souvent suivi de quelques sous-types de ce type qui sont importants ou représentatifs (voir figure 5.1).

Nous présentons en annexe 1 les menus de CGKAT dont ceux de gestion des taxinomies de types de concepts ou de relations. De nombreuses fonctionnalités offertes par ces menus, e.g. la spécialisation ou la destruction[1] d'un type, sont également accessibles via le langage de commandes de CGKAT. Notons que dans CGKAT, pour créer ou modifier un concept ou une relation dans un GC sous un format graphique, l'utilisateur doit avoir préalablement sélectionné son type dans un des menus de gestion des types. Ce type doit donc avoir été inséré dans une taxinomie, et la construction d'un GC sous un format graphique peut ainsi être validée étape par étape, i.e. à chaque ajout, destruction ou modification d'un concept ou d'une relation. Similairement, pour modifier ou ajouter un référent individuel à un concept d'un GC sous un format graphique, le menu de gestion des référents individuels doit être utilisé.

1. En cas de destruction de types, CGKAT vérifie qu'aucun GC de la base en cours n'utilise ces types ou leurs spécialisations. CGKAT vérifie également que l'ajout d'un type, directement ou via une définition de type, ne crée pas de cycle dans la hiérarchie, et qu'aucun des divers parents de ce type ne sont exclusifs.

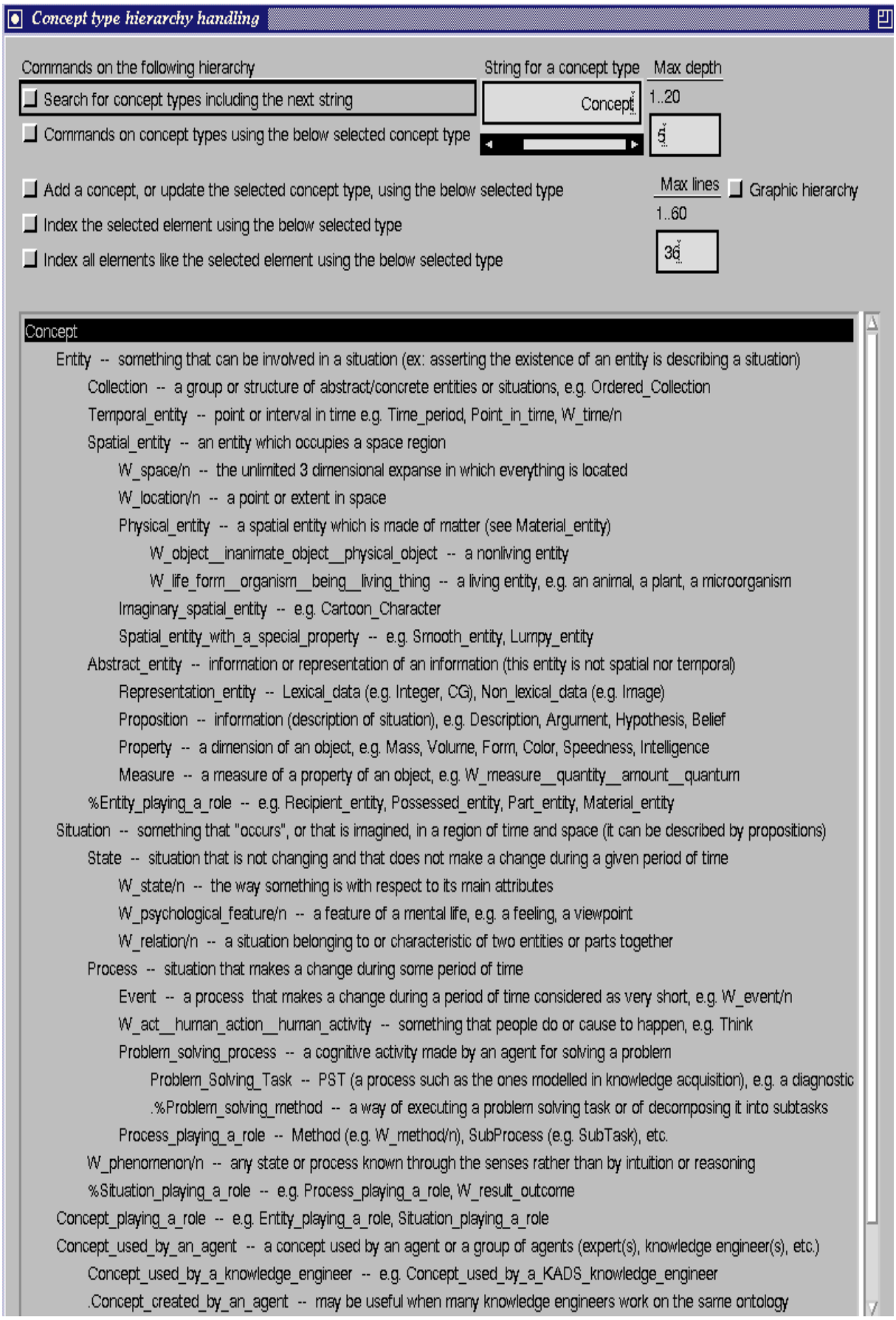


Figure 5.1: *Le menu de gestion des types de concepts montrant les premiers sous-types de Concept (le supertype de tous les types de concept) dans l'ontologie proposée par défaut par CGKAT.*

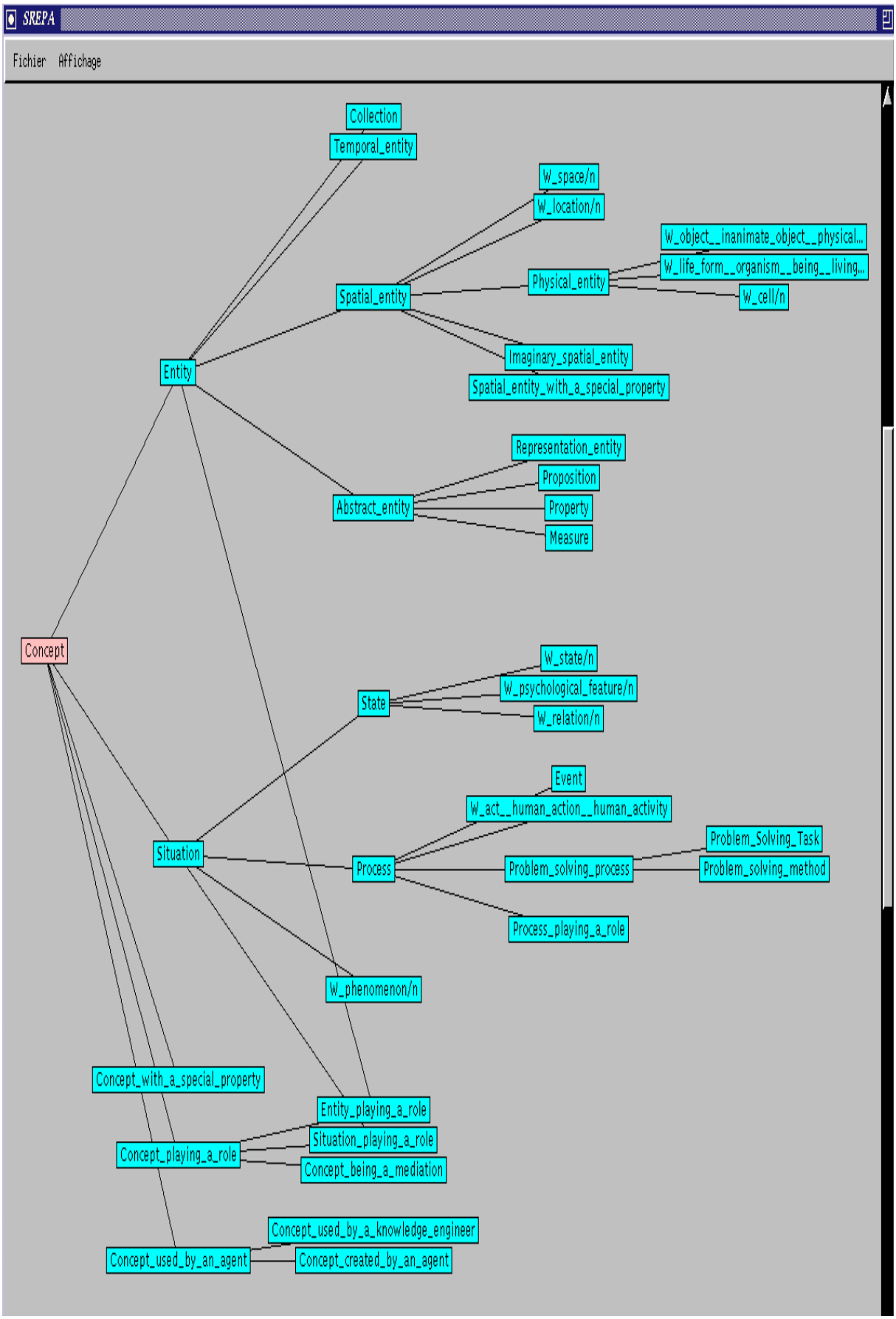


Figure 5.2: *Un générateur de graphes sous Netscape appelé par CGKAT pour permettre de visualiser sous forme de graphe la portion de hiérarchie donnée sous forme de liste indentée dans la figure 5.1.*

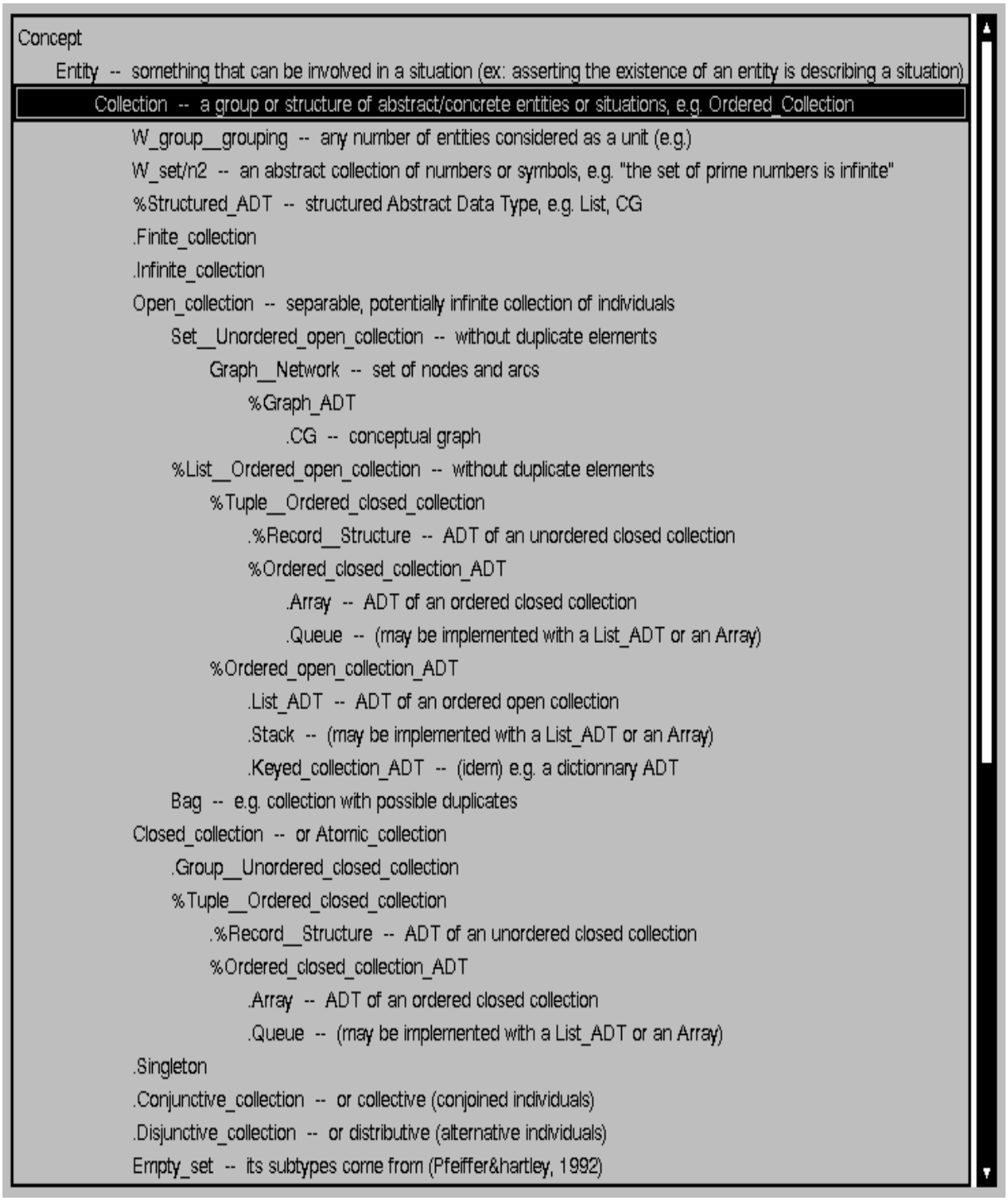


Figure 5.3: *Les supertypes du type "Collection" et quelques-uns de ses sous-types dans l'ontologie de CGKAT.*

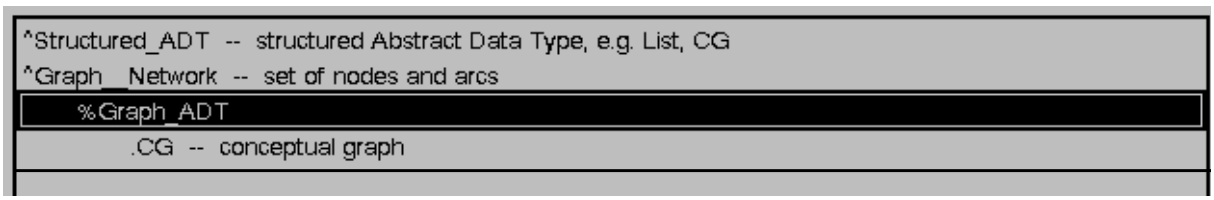


Figure 5.4: *Les supertypes directs du type "Graph_ADT" et son seul sous-type dans l'ontologie de CGKAT.*

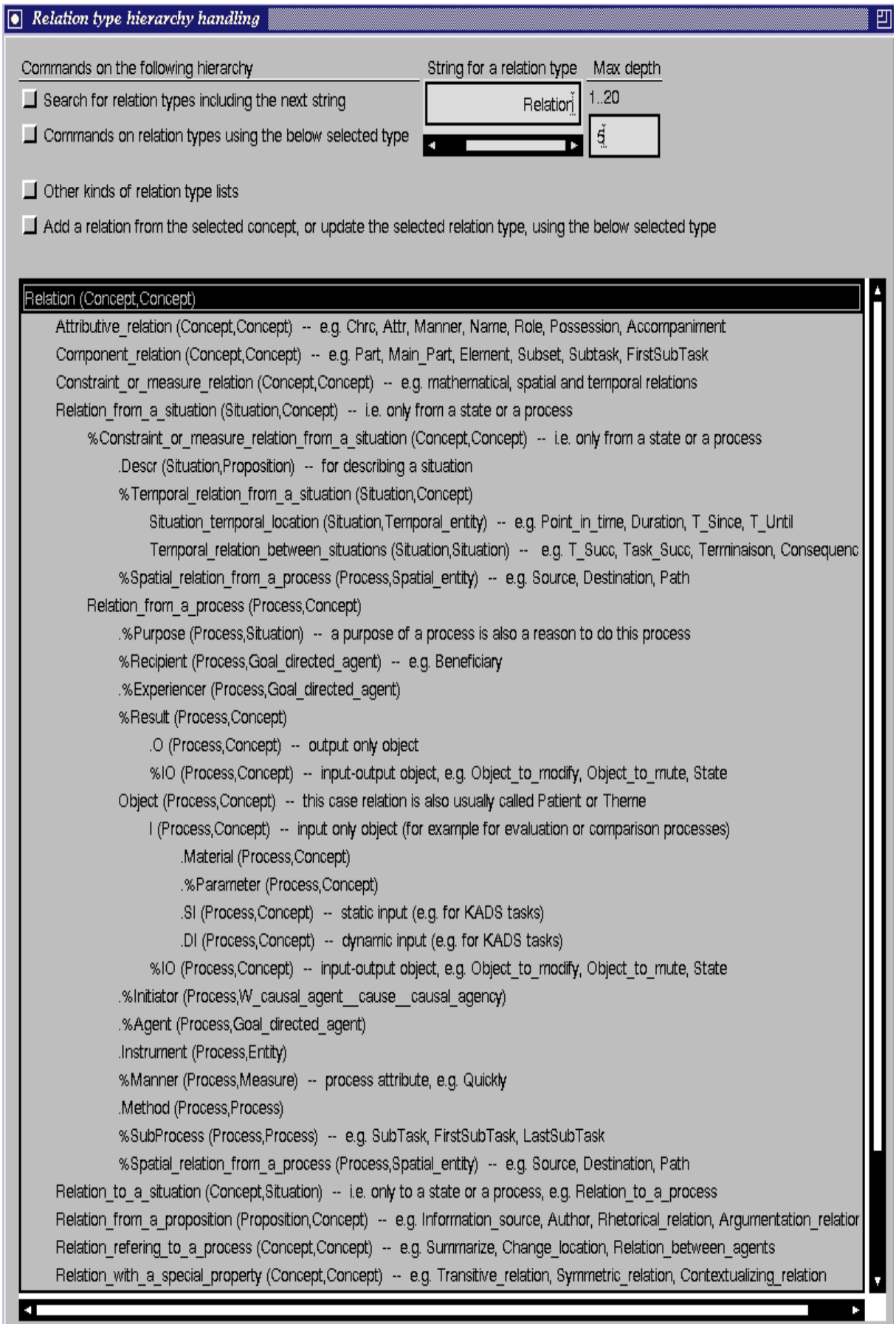


Figure 5.5: *Le menu de gestion des types de relations montrant les premiers sous-types de Relation dans l'ontologie proposée par défaut par CGKAT.*

5.2.2 Accès aux types et construction de vues dans une hiérarchie de types

Dans les menus de gestion de taxinomies de CGKAT, l'accès aux types se fait par ***navigation***, c'est-à-dire par sélections successives de type, ou bien par ***requête***, en donnant une chaîne de caractères. Les types qui contiennent cette chaîne sont alors présentés accompagnés de leurs supertypes (plusieurs listes indentées sont donc présentées).

5.2.2.1 Construction de vues en utilisant les relations de spécialisation entre les types

Nous avons noté qu'un type constitue un point de vue. L'ensemble des sous-types d'un type X constitue une vue sur les types spécialisant le type X. Ainsi dans CGKAT, une manière immédiate de retrouver les types d'une vue (resp. de construire une vue dans une ontologie) consiste à sélectionner le type correspondant au point de vue désiré et à naviguer vers ses sous-types (resp. à spécialiser ce type ou ses sous-types). Lorsqu'un sous-type est créé, d'autres parents n'appartenant pas forcément à la même vue, peuvent lui être donnés. Ainsi par exemple, le type "Tree" pourra être classé parmi les types d'entités physiques et les types de choses utiles à telle ou telle fin. De même, via leur types, des livres peuvent être classés (et donc retrouvés) suivant différentes classifications : genre (e.g. roman, article, article de recherche), thème (e.g. ouvrages sur l'informatique, le génie logiciel, l'approche objet, le génie logiciel par l'approche objet), etc. En résumé, une ontologie permet de structurer de manière fine des points de vue via des liens de spécialisation, des liens d'exclusion, et éventuellement d'autres sortes de liens (cf. section 5.2.3), et peut donc contenir autant de points de vue que désirés.

Une telle manière de procéder n'est cependant pas toujours adéquate, par exemple avec des points de vue indiquant une relation avec un domaine, une source d'expertise (expert ou document source), un cogniticien (le créateur ou les co-créateurs) ou des utilisateurs. En effet, dans ces cas, tous les types relatifs à de tels points de vue sont également sous-types d'autres types, ce qui est non seulement assez lourd à représenter avec des relations de spécialisation[1], mais peut aussi entraîner la création de nombreux types artificiels, par exemple Voiture_pour_un_accidentologue, Voiture_pour_un_expert_en_cinématique, Voiture_pour_un_expert_en_mécanique, Voiture_pour_l' expert_Jean et Voiture_pour_l'expert_Paul. Ces différents points de vue ont de nombreux sous-types en commun, mais doivent être différenciés puisque certains de leurs sous-types peuvent être différents. Pour la même raison, il n'est pas possible de déclarer le type Voiture comme sous-type de Concept_utile_en_accidentologie, ..., Concept_utile_à_l'expert_Paul. Comme on le voit sur cet exemple, de tels points de vue sont en quelque sorte "dupliqués" sous forme de sous-types artificiels pour tous les types qui sont relatifs à ces points de vue.

5.2.2.2 Construction de vues en utilisant les relations de spécialisation entre des méta-informations

Pour résoudre le problème que nous venons de souligner, nous avons introduit dans CGKAT une *nouvelle manière de définir des points de vue* sur un type : en permettant à l'utilisateur d'associer à ce type, ce que nous appelons des ***méta-informations***, c'est-à-dire une liste de couples attribut/valeur *de son choix*. Dans ces couples, l'***attribut*** est une *chaîne de caractères quelconque* et la ***valeur*** est une *chaîne de caractères* qui peut être *quelconque* ou être l'étiquette d'un *type* ou d'un *marqueur individuel*. Voici un exemple d'ensemble de couples attribut/valeur :
{domaine: Accidentologie; expert: Jean;} .
Nous définissons ci-après une relation d'ordre sur de tels ensembles de couples attribut/valeur.

Ainsi dans CGKAT, lors de l'affichage d'une (portion de) taxinomie (i.e. l'affichage de supertypes et de sous-types d'un type sélectionné), les types peuvent être ***filtrés*** (i.e. affichés ou non) suivant

1. Chaque type doit alors avoir plusieurs parents, ce qui est long à déclarer et difficile à présenter de manière ergonomique.

qu'ils sont associés ou non à certaines méta-informations (i.e. à certains points de vue)[1]. Pour cela, dans les menus de gestion de types, l'utilisateur peut définir un ***filtre*** en donnant des couples attributs/valeurs. Dés lors, lorsqu'une portion de taxinomie doit être visualisée (suite à une recherche de types par navigation ou requête), seuls les types auxquels ont été associés des méta-informations qui ***spécialisent*** celles du filtre sont affichés. Il existe cependant un moyen d'indiquer que des types doivent être toujours affichés même si leurs méta-informations ne spécialisent pas celles du filtre (ceci est par exemple utile pour conserver l'affichage des types de plus haut niveau).

Grâce à cette fonctionnalité, l'utilisateur n'est pas obligé d'introduire ou de visualiser des types artificiels ou de multiples supertypes pour chaque type. En contre-partie, ***il doit spécifier explicitement pour chaque type son appartenance à divers points de vue*** : l'appartenance d'un de ses supertypes à un point de vue n'entraîne pas son appartenance à ce point de vue (*sinon, les avantages et les inconvénients seraient les mêmes que lorsque les points de vue sont représentés avec des supertypes*). Par exemple, supposons qu'aucune méta-information n'a été associé au type Vehicle, que les méta-informations {domaine: Accidentologie; expert: Jean;} ont été associées au type Accidented_Vehicle sous-type de Vehicle, et que les méta-informations {domaine: Accidentologie; expert: Jacques;} ont été associées au type Accidented_Race_Vehicle sous-type de Accidented_vehicle. Ainsi, avec le filtre {domaine: Accidentologie;} un cogniticien pourra se concentrer sur les types relatifs à l'accidentologie (Vehicle n'apparaît pas), et avec le filtre {domaine: Accidentologie; expert: Jean;} il peut se concentrer sur les types de l'accidentologie créés pour modéliser l'expertise de Jean (Vehicle et Accidented_Race_Vehicle n'apparaissent pas).

Dans CGKAT, les méta-informations associées à un type sont stockées dans son champ commentaire qui est une chaîne de caractères. L'association de méta-informations à un type peut ainsi s'effectuer dans un menu de gestion de types en créant ou modifiant le commentaire qui lui est associé à ce type (i.e. en structurant la chaîne du commentaire en divers couples attribut/valeur séparés par un caractère ';' comme dans les exemples ci-dessus). Un commentaire ainsi structuré peut par exemple contenir une définition, une liste de mots dont un des sens est représenté par ce type, un identificateur pour l'expert source, un identificateur pour le cogniticien créateur du type, etc.

Nous montrerons au chapitre suivant qu'il est également possible dans CGKAT d'associer des méta-informations à des GCs (ces méta-informations sont stockées dans le champ commentaire de ce GC). CGKAT permet de rechercher dans une base de GCs, ceux qui spécialisent un GC requête ***et/ou*** ceux dont les méta-informations spécialisent un ensemble donné de couples attribut/valeur (cet ensemble peut être donné dans une requête à la suite d'un GC requête ; cf. figure 6.23). Il existe donc, pour les GCs comme pour les vues, deux manières de construire des vues, et ces deux manières peuvent être combinées.

Voici la définition de la relation d'ordre (une relation de "spécialisation") que nous avons implémentée sur les méta-informations, afin d'augmenter la puissance des recherches sur celles-ci.

5.2.2.2.1 Définition actuelle de la spécialisation entre méta-informations

Soient Ex et Ey, deux ensembles de couples attribut/valeur, Ey est plus "spécialisé" que Ex,
si tous les attributs présents dans Ex sont présents dans Ey,
et si pour *chaque* attribut (chaîne de caractères quelconque), la valeur dans Ey est
1) une chaîne de caractères *identique* à celle contenue dans Ex, ou
2) une étiquette d'un type *sous-type* du celui dont l'étiquette est contenue dans Ex, ou
3) une étiquette d'un marqueur individuel *conforme* au type dont l'étiquette est contenue dans Ex
(dans ces deux derniers cas, la valeur dans Ex doit être une étiquette de type).

1. Il s'agit donc bien de génération de vues dans le sens où nous avons défini ce terme.

Exemples, en notant '≤' cette relation de spécialisation :
{domaine: Accidentologie; expert: Jean;} ≤ {domaine: Accidentologie;}
et en supposant que Cinématique est sous-type de Accidentologie et que
Jean un individu du type Expert_en_accidentologie :
{domaine: Cinématique; expert: Jean;} ≤ {domaine: Accidentologie; expert: Expert_en_accidentologie;}

5.2.2.2.2 Extension de la définition actuelle de la spécialisation entre méta-informations.

La définition actuelle de la spécialisation entre méta-informations peut être étendue pour prendre en compte comme valeur d'attribut non seulement une chaîne de caractères mais de manière plus générale n'importe quel élément de document (ED) structuré (e.g. un graphe, une section, un chapitre) ou non structuré (e.g. une chaîne de caractères, une image). En effet, il est possible de considérer qu'un ED x qui contient un ED y, est plus spécialisé que cet ED y puisqu'il contient plus d'informations (il est plus précis).

Ainsi par exemple, un chapitre est plus spécialisé que chacune des sections qu'il contient, une chaîne de caractères est plus spécialisée que n'importe laquelle de ses sous-chaînes, et un ED qui affiche et stocke sous une forme structurée le GC "[Cat]→(On)→[Mat]" est plus spécialisé que son composant qui stocke et affiche le concept "[Cat]".

La définition précédente deviendrait donc :
Soient Ex et Ey, deux ensembles de couples attribut/valeur, Ey est plus "spécialisé" que Ex,
si tous les attributs présents dans Ex sont présents dans Ey,
et si pour chaque attribut (chaîne de caractères quelconque) la valeur dans Ey est
1) un ED qui est identique ou contient celui contenu dans Ex, ou
2) une étiquette d'un type sous-type de celui dont l'étiquette est contenue dans Ex, ou
3) une étiquette d'un marqueur individuel conforme au type dont l'étiquette est contenue dans Ex
(dans ces deux derniers cas, la valeur dans Ex doit être une étiquette de type).

Nous n'avons cependant pas encore implémenté cette dernière définition dans CGKAT.

5.2.2.3 Construction de vues en utilisant les définitions de types et la projection

Une définition de type est un moyen d'exprimer des relations entre des types et donc entre des points de vue. Une troisième façon de générer une vue dans une taxinomie serait donc de sélectionner les types définis en utilisant comme filtre un graphe requête qui serait projeté sur les définitions. L'utilisateur devrait aussi spécifier les types de définitions recherchées : celles par conditions nécessaires, celles par conditions nécessaires et suffisantes, celles par schéma prototypique, n'importe quel type de définition, etc.

CGKAT permet de rechercher des définitions de types (et donc des types définis) via un système de question/réponse (projection d'un graphe requête ; cf. figures 6.22 et 6.24), mais le filtrage sur les définitions n'est pas encore implémenté dans les menus de gestion des types.

Notons que le fait d'ajouter un concept ou une relation dans le graphe d'une définition de type, correspond à une spécialisation du type initial. Un tel ajout n'est donc pas toujours adéquat pour exprimer un point de vue supplémentaire sur le type défini, par exemple lorsque le point de vue ajouté indique une relation avec un domaine, une source d'expertise, un cogniticien ou des utilisateurs (pour les même raisons que celles données au 5.2.2.1, cela conduit à la "duplication" des points de vue sous forme de sous-types artificiels pour tous les types qui sont relatifs à ces points de vue).

5.2.2.4 Construction d'une même taxinomie de types par plusieurs utilisateurs

Une taxinomie de types ne comporte pas de règles ou de connaissances factuelles mais seulement des définitions de types. Même si cela est difficile, il semble donc toujours possible d'interclasser dans ***une même taxinomie***, des types provenant de ***plusieurs experts et/ou de plusieurs cogniticiens*** : en effet, soit des types sont strictement égaux (i.e. ils représentent un même ensemble d'objets) et doivent alors être représentés sous un même nom "système", soit ils sont différents et dans ce cas des noms "système" différents doivent être utilisés. Les choix dans la terminologie peuvent être facilités par l'utilisation d'une visualisation en mode "utilisateur". Par ailleurs, des systèmes de *vues* tels que ceux que nous avons présentés, permettent à un utilisateur de ne voir que les types relatifs aux points de vue qu'il désire, et donc de pouvoir travailler seulement dans les vues désirées. Ainsi, pour organiser, seul ou de manière coopérative, des types et des définitions de types (et donc différentes sortes de connaissances générales[1]), il ne semble pas nécessaire d'utiliser des *modules*, qui comme nous l'avons souligné au chapitre 2, ne favorisent pas la comparaison ou l'interclassement des connaissances.

Bien qu'à notre connaissance aucune comparaison expérimentale n'ait été faite sur ce point, il semble a priori plus rapide pour des cogniticiens travaillant à représenter une même expertise, de construire de manière coopérative et incrémentale une même taxinomie de types, plutôt que de travailler chacun de leur côté, "en aveugle", puis de fondre leurs taxinomies en une seule. En effet, dans le premier cas, chaque cogniticien peut réutiliser des parties de taxinomies définis par d'autres et positionner ses termes par rapport aux types déjà existants.
Nous ne connaissons pas non plus de méthode destinée à permettre à plusieurs utilisateurs de construire de manière coopérative et incrémentale ***une même taxinomie de types***. En effet, les recherches actuelles concernent :

1. des méthodes pour intéger de manière semi-automatique des ontologies développées séparément (Lehmann & Cohn, 1994) (Wiederhold, 1994) (Takeda & al., 1994) (Knight & Luk, 1994) (Dieng, 1996),
2. la construction de taxinomies de types atomiques dans des modules séparés mais où des types de différents modules peuvent être reliés par des liens de spécialisation (Garcia, 1995, 1996a, 1996b). Dans ce travail, un module peut être destiné à représenter les types utilisés par un expert dans un domaine (pour nommer les types du module, la terminologie de l'expert est utilisée). Un module peut également être commun à divers experts. A l'intérieur d'un tel module, un type ou un lien de spécialisation peut être "partagé par plusieurs experts" mais dans ce cas, ce type ne doit plus être modifié. Ce travail exploite la distinction entre experts, domaines et sous-domaines. Il n'adresse pas les problèmes de construction des modules par plusieurs cogniticiens, i.e. plusieurs utilisateurs.

Nous donnons en annexe 2, une liste de fonctions ou de protocoles destinées à permettre/faciliter la *construction coopérative et incrémentale d'une même hiérarchie de types par plusieurs* ***utilisateurs***. Pour gérer la provenance des types (experts, domaines, documents, etc.) nous suggérons d'utiliser des systèmes de génération de vue, c'est-à-dire de filtrage.

---

1. Des connaissances générales telles que "tous les chats sont mortels", "les chats aiment généralement le lait", "pour atteindre tel but, il faut toujours/généralement effectuer telle action" peuvent être représentées sous forme de définitions de types. Par exemple: TC for Cat (x) are [Cat]←(Experiencer)←[Like]→(Object)→[Milk].
De même, les modèles de tâches génériques peuvent être représentées dans le formalisme des GCs en associant aux types représentant ces tâches des définitions par conditions typiques (cf. section 5.5).

5.2.3 Gestion d'autres relations que la relation "sorte-de"

CGKAT propose pour permettre de visualiser, retrouver et gérer les relations de conformité entre les marqueurs individuels et les types, un menu similaire à ceux offerts pour les relations de spécialisation entre types de concepts et entre types de relations (cf. le menu de gestion des marqueurs individuels dans l'annexe 1). Cependant, dans le modèle des GCs que nous exploitons, il n'y a pas de type d'ordre supérieur : aucun type n'est "instance" d'un autre type (aucune étiquette de type n'est un marqueur individuel d'ordre supérieur "conforme" à un type d'ordre supérieur). Aussi, actuellement dans CGKAT, les hiérarchies gérées par le menu de gestion des marqueurs individuels n'ont qu'un seul niveau de profondeur.

La relation de subsomption entre les types (relation "sorte-de") et la relation d'instanciation entre des types d'ordre supérieur sont des relations permettant de hiérarchiser des types. D'autres sortes de relations entre types permettraient de les hiérarchiser, et pourraient donc être gérées via des menus similaires en apparence à ceux qu'offre CGKAT pour visualiser et gérer la relation de subsomption entre types. Toutefois, la représentation de telles relations entre types dans le formalisme des GCs n'est alors plus directe : ces relations ne peuvent être simplement stockées dans des structures hiérarchiques prévue à cet effet, mais des GCs doivent être construits. Si donc des menus permettent de rentrer, de visualiser et de gérer ces relations sous la forme de hiérarchies, une traduction automatique doit être gérée pour transformer ces hiérarchies en GCs, et pour cela différentes options doivent être spécifiées par l'utilisateur. Par exemple,
1) quelle est la signification d'une relation "partie-de" entre un type Voiture et un type Moteur : toutes les voitures ont-elle nécessairement un moteur ? Est-ce un schéma prototypique pour une voiture ? Ceci doit être spécifié par l'utilisateur ;
2) comment une telle relation doit être représentée dans le formalisme des GCs : avec une règle, avec une condition nécessaire et/ou suffisante associée au type Voiture, avec un schéma prototypique associé au type Voiture et/ou au type Moteur, etc. ?
C'est pourquoi la gestion de menus semblables à ceux actuellement offerts pour la relation de subsomption entre types, serait beaucoup plus difficile.

5.3 Exploitation de la base générale de connaissances terminologiques WordNet

Nous proposons dans cette section un moyen d'utiliser une base générale de connaissances terminologiques[1] telle que WordNet, pour guider et faciliter la création d'une ontologie, la rendre plus claire et plus réutilisable.

L'idée principale pour la réutilisation d'une ontologie, ou encore la comparaison d'ontologies, consiste à reposer, lors de la création de ces ontologies, sur une même ontologie de moyen niveau interclassant les concepts du langage naturel. Nous avons choisi WordNet, mais une autre ontologie similaire pourrait être exploitée pour jouer ce rôle de standardisation qui facilite la réutilisation.

Actuellement seules trois ontologies de moyen niveau nous semblent être suffisamment générales et organisées pour pouvoir jouer ce rôle et guider la construction de l'ontologie d'une application : celle de CYC (40.000 catégories conceptuelles), WordNet (90.000 catégories conceptuelles) et celle du système de traduction automatique Pangloss (Knight & Luk, 1994).

- Panglos, contrairement à WordNet, n'est pas du domaine public. Cette ontologie réunit WordNet et le dictionnaire Longman d'anglais contemporain (LDOCE). De plus, les catégories de haut niveau du résultat (i.e. WordNet-LDOCE) sont complétées et restructurées avec celles des ontologies de haut niveau Ontos (Carlson & Niremburg, 1990) et du PENMAN Upper Model (Bateman, 1990).
- L'ontologie de CYC (Lenat & Guha, 1990a) devrait bientôt être du domaine public mais, contrairement à WordNet, ses catégories ne semblent pas être reliées à des termes du langage naturel, ce qui rend plus difficile la recherche de types adéquats à réutiliser pour construire l'ontologie d'une application ou représenter des connaissances.

5.3.1 Présentation de WordNet

WordNet (Miller & al., 1990) est développé à l'université de Princeton. Sa conception est inspirée par les théories psycholinguistiques actuelles sur la mémoire lexicale humaine (c'est-à-dire sur l'organisation de termes par des relations linguistiques ou sémantiques). La dernière version de WordNet (1.5) est une base reliant environ 120.000 racines de noms, verbes, adjectifs et adverbes anglais à environ 91.600 ***ensembles de synonymes*** représentant leurs *sens* sous-jacents.

5.3.1.1 Principes de WordNet

WordNet associe des racines de mots (mots simples ou composés) à des ensembles de synonymes qui représentent leurs sens sous-jacents, et inversement, chaque ensemble de synonymes est lié aux racines de mots dont ils représentent le sens. Plus précisément, un ensemble de synonymes *est composé de* ces racines de mots synonymes[2]. Voici par exemple les ensembles de synonymes représentant selon WordNet les sens du mot anglais "table" (les définitions données proviennent également de WordNet) :

---

1. Le concept de "base de connaissances terminologiques" et quelques-uns de ses liens avec l'acquisition des connaissances est développé dans (Bourigault & Condamines, 1995) : "une base de connaissances terminologiques se distingue des bases de données terminologiques classique par la présence d'une description conceptuelle explicite et structurée". Il s'agit donc d'un thésaurus structuré par diverses relations sémantiques et où les termes ne désignent qu'un seul individu ou type de concept. C'est le cas de WordNet. Toutefois, pour Bourigault et Condamines, une base de connaissances terminologiques est dédiée à un domaine et extraite de manière automatique, au moins partiellement, à partir d'un corpus de référence. Ce n'est pas le cas de WordNet qui est général (il associe 120.000 termes de la langue anglaise à 91.600 catégories conceptuelles) et qui a été construit manuellement. C'est pourquoi nous appelons WordNet, une base "générale" de connaissances terminologiques.

2. Plus exactement, "synonymes dans certains contextes", car ces mots peuvent avoir plusieurs sens.

{ table, tabular array } -- a set of data arranged in rows and columns
{ table } -- a piece of furniture having a smooth flat top supported by one or more vertical legs
{ table } -- a piece of furniture with tableware for a meal laid out on it
{ mesa, table } -- flat tableland with steep edges
{ board, table } -- food or meals in general; «she sets a fine table»
{ postpone, hold over, put over, table, shelve, set back, defer, remit, put off } -- (pas de définition)

Chaque (racine de) mot peut avoir différents sens dans différents contextes, et peut donc appartenir à plusieurs ensembles de synonymes. Un ensemble de synonymes représente un seul sens et est donc unique. Comme le montre l'exemple ci-dessus, deux ensembles de synonymes différents peuvent n'être composés que d'une unique et même racine de mot (dans chacun de ces deux ensembles, une seule racine de mot est connu pour le sens représenté). Cependant, deux ensembles de synonymes ne partagent jamais le même "identificateur de sens" (un numéro d'identification interne à WordNet), ce qui en assure l'unicité.

Dans des prochaines versions de WordNet - International WordNet (Sutcliffe, 1995) ou EuroWordNet (Vossen, 1996) - un sens pourra être représenté par un ensemble de synonymes dans différentes langues (anglais, allemand, espagnol et peut-être français et italien). Ainsi, pour un mot d'une langue donnée, un ensemble de synonymes pourra être donné pour une ou plusieurs langues.

Les ensembles de synonymes sont liés par quelques ***relations lexicales***, e.g. l'antonymie, et diverses ***relations sémantiques***, e.g. l'hyperonymie, l'hyponymie, la méronymie (relations élément-de, substance-de[1] et partie-de), la relation d'attribut, une relation de causalité entre les actions (il y en a plusieurs sortes ; cf. (Miller & al., 1990)), et certaines relations casuelles entre des actions et leurs participants (e.g. Agent, Object).
Par exemple, les hyperonymes de {table, tabular array} sont, dans l'ordre, {array}, {arrangement} et {group, grouping}. Ses hyponymes directs sont {actuarial table, statistical table}, {calendar} et {periodic table}. Ses méréonymes directs selon la relation a-pour-élément (inverse de élément-de) sont {row} et {column}, et selon la relation a-pour-partie, {leg}, {tableware} et {tabletop}.

Dans WordNet, seuls les ensembles de synonymes formés avec des noms ou des verbes sont structurés par des relations d'hyponymie/hyperonymie. Cela n'est pas le cas, et ne semble pas possible, pour les ensembles de synonymes formés avec des adjectifs ou des adverbes (voir (Miller & al., 1990)). Pour les ensembles de synonymes formés avec des noms, voici la définition qui a été suivie : "un concept représenté par l'ensemble de synonymes {x, x', ...} est un hyponyme d'un concept représenté par l'ensemble de synonymes {y, y', ...} si des locuteurs de langue anglaise acceptent la construction de phrases suivant le schéma : *x is a y*". Pour les relations d'hyponymie/hyperonymie entre des ensembles de synonymes formés avec des verbes, qui sont alors appelées des relations de troponymie, le schéma est : "*to x is to y in some particular manner*". La relation de troponymie est ainsi distinguée des relations de causalité entre actions telles que celle existant entre acheter et payer. WordNet donne certaines relations de causalité entre des "catégories d'actions représentées par des ensembles de synonymes formés avec des verbes", mais pas entre les "catégories d'actions représentées par des ensembles de synonymes formés avec des noms" (exemples de tels noms : résistance, acceptation, observation).

Ces deux ensembles de catégories d'actions ne sont pas interclassés. Dans celui avec les noms, toutes les catégories sont bien interclassées par les relations d'hyponymie/hyperonymie et ont un seul ancêtre commun. Dans celui avec les verbes, la relation de troponymie conduit à de nombreuses catégories de haut niveau qui ne sont pas interclassées entre elles. Par ailleurs, ce dernier ensemble est plus petit que le premier (i.e. moins complet). Il semble donc préférable, pour un cogniticien qui via un outil tel que CGKAT peut exploiter les catégories de WordNet pour la construction d'une ontologie, d'exploiter l'ensemble avec les noms plutôt que celui avec les verbes.

1. Il existe une relation substance-de entre une entité physique et la matière qui le compose. Par exemple pour WordNet : {heartwood, duramen} a une relation substance-de avec {Tree}.

5.3.1.2 Exploitation de WordNet par des programmes

WordNet est accompagné d'une interface fonctionnelle qui permet son exploitation par des outils logiciels. Ainsi, par exemple, les fonctions de WordNet peuvent prendre un mot anglais, en extraire la racine (analyse morphologique) et renvoyer la liste des ensembles de synonymes qui représentent les sens de ce mot.

WordNet est actuellement exploité par divers analyseurs généraux de langage naturel, e.g. Pangloss, Snowy (Gomez, 1995) et Lolita (Long & Garigliano, 1994). WordNet a également été utilisé pour aider à la classification conceptuelle dans l'outil Tanka (Feng & al., 1994).

5.3.1.3 Discussion

Toutefois, comme le note Gomez, la version actuelle de WordNet (notamment ses catégories de haut niveau) nécessite des ajouts et des modifications afin d'être plus utile pour un interpréteur sémantique et donc pour la représentation de connaissances. Par exemple, dans la version actuelle, des catégories de haut niveau existent pour rassembler et permettre d'interclasser les concepts de location, de quantité, de relation et d'unité de mesure ; cependant toutes les catégories de WordNet relatives à ces notions ne sont pas toujours rattachées à ces catégories de haut niveau. Par ailleurs, nous avons noté certaines faiblesses dans la constitution de la base. Il existe par exemple quelques (rares) catégories qui ne pointent pas vers leurs sous-catégories, alors que celles-ci les réfèrent bien (i.e. le lien inverse n'est pas toujours stocké).

Dans les deux sections suivantes, nous allons décrire comment WordNet est exploité depuis CGKAT.

5.3.2 Inclusion dynamique de WordNet dans le treillis des types de concept

Les catégories conceptuelles de WordNet représentent très rarement des individus (entités ou situations individuelles). Une des exceptions est par exemple {Bach, Johann Sebastian Bach} qui est une sous-catégorie de {organist} et de {composer}. Il nous a donc semblé raisonnable pour *guider* la construction de l'ontologie d'une application, d'assimiler ces catégories à des types de concepts, et donc les relations d'hyponymie/hyperonymie à des relations de spécialisation/généralisation entre types de concepts. Lorsqu'il rencontre une catégorie représentant un individu, l'utilisateur ne doit pas l'inclure dans l'ontologie de son application. Notons que WordNet ne fournit aucun type de relation, même dans la catégorie {relation} dont les sous-catégories représentent des *états* où des objets entretiennent certaines relations.

5.3.2.1 Méthode

Nous avons trouvé un moyen pour générer des noms de types de concepts uniques et explicites à partir des catégories de WordNet : les synonymes d'une catégorie sont concaténés, séparés par "__", et si la catégorie ne contient qu'un seul synonyme, un "identificateur de sens" est ajouté. Ainsi, à partir d'un nom de type de concept provenant de WordNet, la catégorie correspondante peut être automatiquement retrouvée et donc ses relations avec d'autres catégories exploitées.
Nous avions dans un premier temps essayé de ne pas ajouter d'identificateur de sens lorsque cela était inutile, c'est-à-dire lorsque pour un ensemble de synonymes, la base ne contient pas d'autres ensembles de synonymes de même contenu, mais cette vérification dans la base entraînait un temps de génération (de nom de type de concept) beaucoup trop long.
L'identificateur de sens est composé 1) d'un identificateur de la catégorie grammaticale des synonymes de l'ensemble concerné ('n' pour nom, 'v' pour verbe, etc.), et 2) un numéro permettant de distinguer différents ensembles de même contenu et relatifs à une même catégorie grammaticale. Ainsi par exemple, pour les ensembles de synonymes exprimant les sens du mot "table", les noms de types de concepts générés par CGKAT sont : W_table__tabular_array, W_table/n1, W_table/n2, W_mesa_table, etc.

CGKAT permet d'inclure temporairement ou définitivement des types provenant de WordNet. De tels types peuvent être recherchés et visualisés dans le menu de gestion des types de concepts, de la même façon que les autres types de concepts : par navigation ou par requête, mais dans ce dernier cas, la chaîne de caractères donnée doit être un mot anglais entier.

Dans le cas d'une requête, CGKAT fournit en réponse la liste des types de WordNet représentant les sens de ce mot (l'analyse morphologique de ce mot et les recherches dans la base sont effectuées grâce aux fonctions de WordNet). Lorsque cette liste de types est présentée, chaque type est accompagné de ses supertypes (comme pour un type normal), afin de faciliter la compréhension de son sens (voir figure 5.4). Si aucun de ces supertypes n'était déjà inclus dans le treillis, le supertype de plus haut niveau est placé comme sous-type direct de "Concept". Si l'un des supertypes était déjà inclus, CGKAT recherche les supertypes de ce dernier non plus dans WordNet mais dans le treillis. *Ainsi, l'utilisateur peut modifier ou compléter l'organisation hiérarchique des types de WordNet et ces modifications sont prises en compte lors des recherches* ***par requête***. Tous les types de WordNet qui n'étaient pas déjà inclus dans le treillis sont affichés précédés du caractère '~' pour signaler qu'il sont temporairement inclus. Ils seront ôtés du treillis lors de la recherche suivante par requête ou navigation.

La navigation peut s'effectuer en sélectionnant un type de WordNet, temporairement inclus ou non. Les premiers sous-types du type sélectionné sont alors affichés (voir figure 5.6). Si le type n'était pas temporairement inclus, ces sous-types sont d'abord recherchés dans le treillis puis dans WordNet. *Ainsi, l'utilisateur peut compléter ou modifier l'organisation hiérarchique des types de WordNet et ces modifications sont prises en compte lors des recherches* ***par navigation***.

L'utilisateur peut ajouter (ou faire ajouter par CGKAT) un marqueur particulier dans le commentaire d'un type de concept afin que certains sous-types fournis par WordNet ne soient plus affichés. Ce masquage équivaut à une destruction de ce type dans WordNet mais pour une ontologie donnée. Similairement, pour inclure définitivement un type de WordNet dans le treillis, afin qu'il ne soit pas détruit lors des recherches ultérieures, l'utilisateur doit ajouter un autre marqueur dans le commentaire de ce type.
L'ajout de ces marqueurs peut s'effectuer de manière transparente via des commandes du menu de gestion des types de concepts. Dans ce menu, seuls certains champs du commentaire d'un type sont visibles, les marqueurs ne sont normalement pas affichés. Notons qu'une hiérarchie de types peut également être rentrée en construisant directement un fichier de sauvegarde au format BCGCT ou un fichier script, ce qui permet de modulariser l'ontologie en divers blocs, dans un ou plusieurs fichiers (cf. annexe 8). Lorsqu'il constitue ces fichiers, l'utilisateur voit et rentre lui-même tous les champs des commentaires des types.

5.3.2.2 Remarques

Le point le plus délicat que nous avons résolu pour permettre l'inclusion de parties de WordNet dans l'ontologie d'une application, a été de prendre en compte les modifications effectuées par l'utilisateur sur l'organisation des types de WordNet. Ce point était très important puisque l'utilisateur est amené à effectuer de telles modifications ou complétions, tant à cause de certains choix discutables dans l'organisation de certains types, que pour adapter cette ontologie générale à une application particulière. WordNet interclasse de manière fine de nombreux types de concepts du langage naturel ; c'est donc un support que le cogniticien peut spécialiser et dont il peut localement se différencier, mais ce support lui permet d'interclasser aisément et finement les types de concepts de son application qui sont proches de ceux du langage naturel, et ce support facilite la compréhension, l'extension et la réutilisation de son ontologie.

Pour l'inclusion dynamique de types de WordNet par navigation, CGKAT n'exploite actuellement que la relation de spécialisation/généralisation entre les types de WordNet. Nous avons noté que l'exploitation d'autres relations hiérarchiques comme les relations de méronymie (relations élément-de, substance-de et partie-de) serait plus difficile car leur traduction dans le formalisme des GCs n'est pas aussi directe.

Rappelons également que dans WordNet, seuls les ensembles de synonymes formés avec des noms ou des verbes sont structurés par des relations de spécialisation. Même pour les catégories d'actions, il est préférable d'exploiter les ensembles de synonymes formés avec des noms plutôt que ceux formés avec les verbes (les premiers sont plus nombreux et beaucoup mieux interclassés que les seconds).

Il faut noter que lorsque des types de WordNet sont inclus dans le treillis, celui-ci peut ne plus conserver sa structure de treillis. Mais c'est aussi le cas lorsque des types sont ajoutés manuellement. L'organisation des types de concepts selon une structure de treillis pourrait donc être vérifiée périodiquement par un outil, lorsqu'aucun type de WordNet n'est temporairement inclus dans ce treillis (par exemple un outil travaillant sur un fichier de sauvegarde). Un outil tel que C-CHIC (Leclère, 1995) (Leclère, 1996) permet d'effectuer une telle vérification et guide l'utilisateur pour l'obtention de cette structure (cet outil n'est pas inclus dans CGKAT).

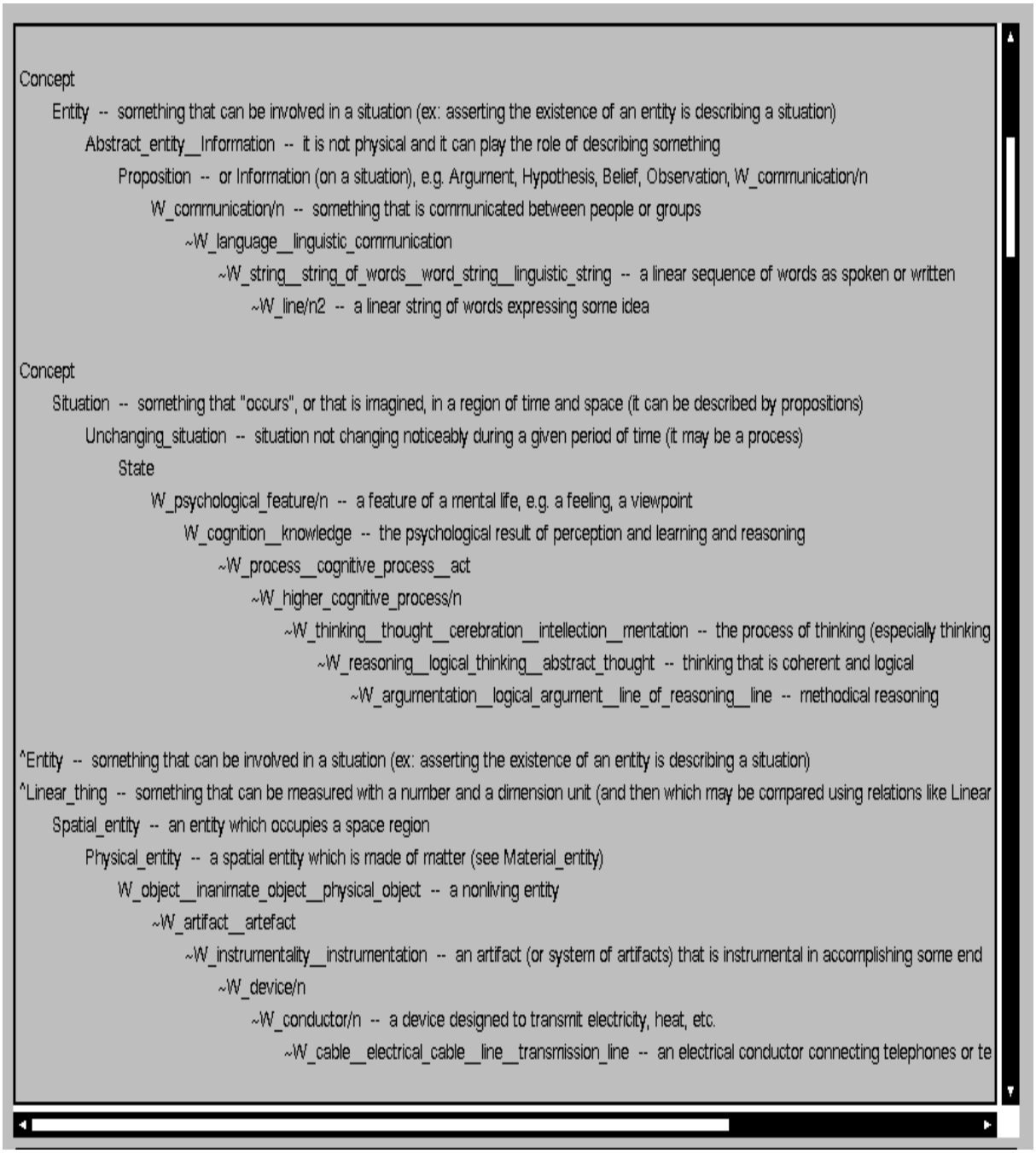


Figure 5.6: *Les types de concepts fournis par WordNet pour le mot "line" et leurs supertypes.*

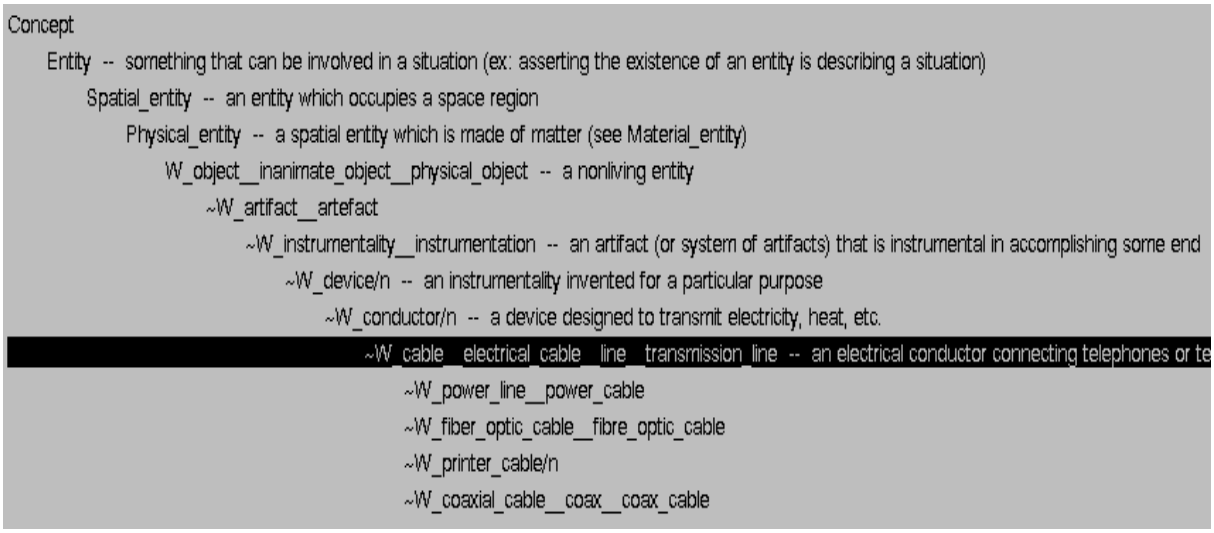


Figure 5.7: *L'un des types relatifs au mot "line" et ses sous-types directs.*

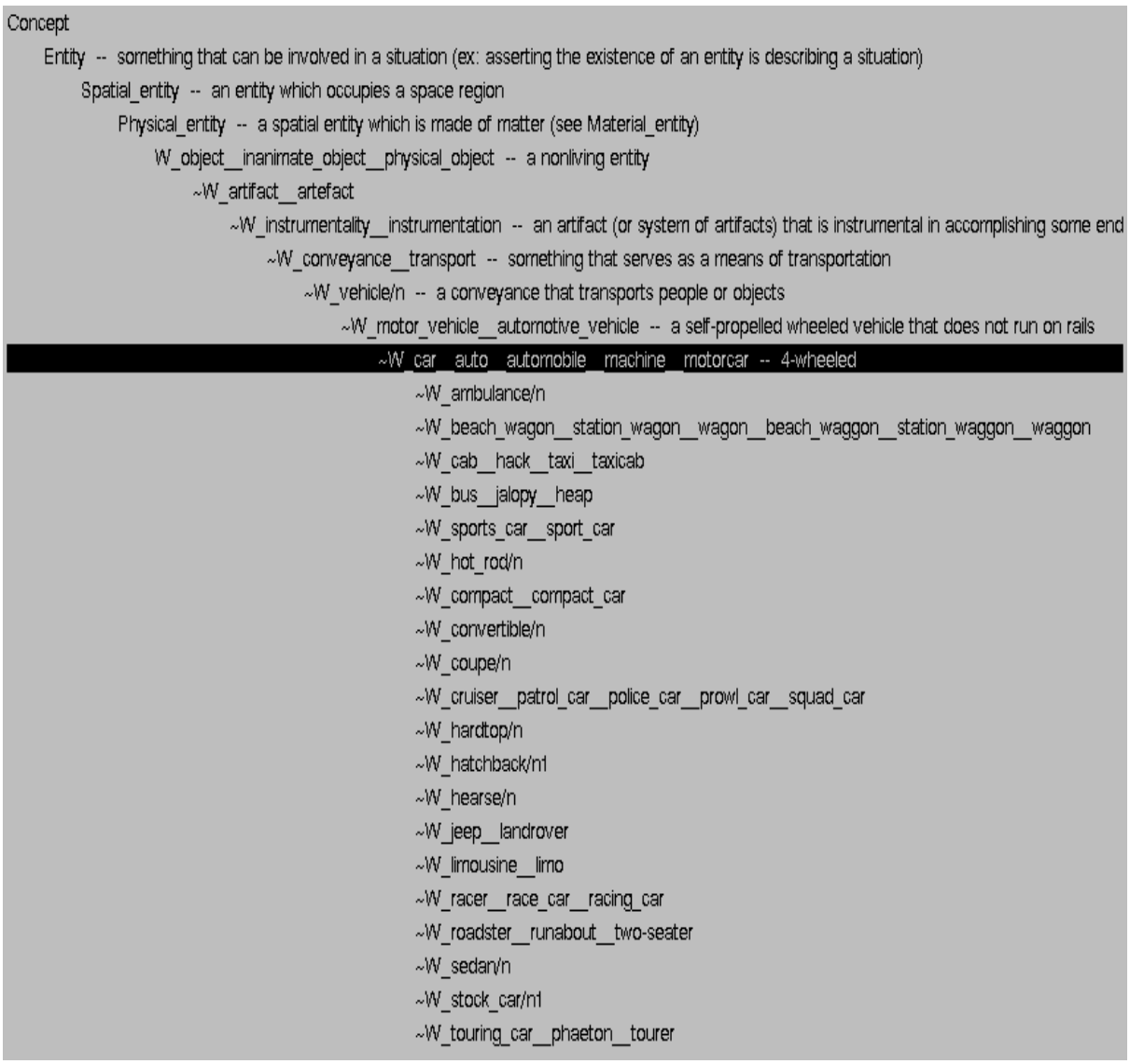


Figure 5.8: *L'un des types relatifs au mot "Car" et ses sous-types directs.*

5.3.3 Intérêt de WordNet pour l'acquisition des connaissances

Nous avons noté que WordNet était déjà utilisé pour certains outils d'extraction automatique de connaissances : analyseurs sémantiques comme Snowy (Gomez, 1995), outils de classification conceptuelle comme Tanka (Feng & al., 1994). De façon complémentaire, notre approche consiste à faciliter et à guider la création manuelle d'une ontologie compréhensible, extensible et réutilisable.

Lors de la création d'une ontologie, et particulièrement si le cogniticien réalise sur un document une *première phase de modélisation ascendante* de connaissances (donc peu guidée par des modèles), le cogniticien peut avoir besoin de créer de nombreux[1] types de concepts. Sans outil d'analyse terminologique (e.g. LEXTER (Bourigault, 1995)) ou de classification automatique (e.g. Tanka), la création de ces types et leur interclassement prend beaucoup de temps. Or de tels outils sont actuellement limités en domaine et en profondeur d'analyse. L'approche que nous proposons peut être utilisée en complément de ces outils ou bien directement, puisqu'elle guide et accélère la création/complétion manuelle d'une ontologie et la représentation des connaissances. En effet, un certain nombre d'étapes dans la création et l'organisation des types sont effectuées (ou proposées) par CGKAT et non par l'utilisateur.

Par exemple, pour représenter un élément de document (mot, phrase, image, section, etc.), l'utilisateur de CGKAT peut sélectionner cet élément puis 1) rechercher avec une requête (mot anglais) les types de concepts susceptibles d'être utilisés pour représenter l'élément avec un concept simple ou un GC plus complexe, 2) sélectionner le plus adéquat (et ainsi vérifier si un des sous-types du type sélectionné n'est pas plus adéquat), 3) demander l'inclusion définitive de ce type dans l'ontologie, et 4) demander la création d'un concept du type sélectionné (ou son ajout dans un GC) afin de représenter l'élément sélectionné. Ainsi, grâce à l'inclusion dynamique de WordNet, un élément de document quelconque peut dans de nombreux cas être représenté à l'aide de seulement quatre actions élémentaires, donc très rapidement. Rappelons également que les types ainsi créés, c'est-à-dire ceux fournis par WordNet, sont explicites, assortis d'une définition et fortement organisés (ce que ne peut réaliser automatiquement un outil d'analyse terminologique ou de classification conceptuelle, à moins qu'il n'exploite comme Tanka une base de connaissances terminologiques déjà constituée, structurée et documentée). Cette structuration est utile pour la compréhension et les extensions mais peut également être exploitée par les mécanismes de recherche et d'inférences (e.g. les recherches par projection)[2].

Notons que si le document source est en anglais, le cogniticien n'a pas à traduire un mot en anglais pour rechercher des types permettant de le représenter, il lui suffit de sélectionner ce mot et de lancer la recherche. Cependant, pour représenter d'autres types d'éléments que des mots, e.g. une image, une phrase, une section ou un document, même s'il peut réutiliser des mots contenus dans ces éléments, il doit souvent lui-même trouver des mots adéquats relatifs à ce qu'il veut représenter. Aussi, pour la représentation des connaissances d'un document ou plus généralement pour la création d'une ontologie, la limitation actuelle de WordNet à la langue anglaise ne semble pas devoir être un frein pour un cogniticien connaissant bien cette langue.

---

1. Dans une première phase de modélisation ascendante, un document technique ou une retranscription d'expertise peut contenir une centaine de phrases ayant un contenu intéressant à modéliser et impliquant la création de divers types et de leurs supertypes. Dans une seconde phase de modélisation ascendante, les connaissances sont regroupées et des sélections peuvent être effectuées parmi les types nécessaires.

2. Eventuellement, lors de la phase de conception et d'implémentation du SBC, pour des raisons d'efficacité dans l'exécution, tous les types de l'ontologie peuvent ne pas être inclus dans le SBC final. Les types provenant de WordNet pourraient alors être enlevés. Mais cela réduirait les possibilités de recherche et d'abstraction (et donc aussi d'explication) du SBC. En effet, 1) un type X et les connaissances qui utilisent ce type, ne pourraient plus être recherchés en utilisant les supertypes de WordNet pour ce type, 2) les sous-types de WordNet pour ce type X ne pourraient plus être exploités ou fournis comme exemples, et 3) ce type X (resp. les connaissances qui utilisent ce type) ne pourrai(en)t plus être comparé(es) aux types de WordNet (resp. les connaissances qui utilisent des types sous-types de ces types de WordNet).
Aussi, en phase de modélisation, il est important de conserver une taxinomie très structurée (et donc riche) pour faciliter les recherches, les comparaisons et la compréhension des types.

Nous avons cependant pensé à une autre exploitation de WordNet que celle présentée ci-dessus, et où la limitation à l'anglais est plus déterminante : la recherche dans un document de mots relatifs à des types de concepts fournis par l'utilisateur. Le principe est simple : 1) à partir des types de concepts fournis, la liste des racines de mots dont l'un des sens peut être représenté par un des types est créée, puis 2) chacun des mots du texte est comparé avec ceux recherchés. WordNet est intéressant pour cela puisqu'il fournit pour chacun de ses types, la liste des racines de mots dont l'un des sens est représenté par ces types. Par ailleurs, avant la comparaison d'un mot du texte avec l'un des mots recherchés, la fonction d'analyse morphologique de WordNet peut être appelée sur ce mot, ce qui augmente les possibilités de comparaison. Une telle fonction de recherche semble intéressante aussi bien en analyse descendante pour trouver des parties de documents relatifs à un concept dans un modèle (e.g. une inférence ou un rôle), qu'en analyse ascendante pour pouvoir regrouper des connaissances similaires. Nous avons partiellement implémenté cette fonction dans CGKAT (la fonction d'analyse morphologique de WordNet n'est pas encore appelée).

En ce qui concerne la création d'une ontologie, nous avons noté que compte-tenu de la structuration de WordNet, de sa généralité et du caractère explicite de ses types, son utilisation comme un support à spécialiser ou à compléter localement, favorise la réutilisation d'une ontologie : a) la réutilisation d'une ontologie pour un autre domaine ou une autre tâche, b) la construction coopérative et incrémentale d'une même ontologie par plusieurs cogniticiens, c) la fusion d'ontologies construites séparément mais reposant toutes sur WordNet. En effet, concernant ce dernier point, l'efficacité (en degré d'exactitude) de procédures automatiques de comparaison et d'unification de taxinomies de types[1], devrait alors être plus grande que lorsque les ontologies ne sont pas basées sur un même support, i.e. lorsqu'elles ne sont pas organisées autour des mêmes types (avec les mêmes noms).
A moyen ou long terme, des spécialisations ou corrections faites à partir de certaines ontologies à WordNet, pourraient être intégrées à cette dernière et ainsi augmenter encore son intérêt.

WordNet contient beaucoup de types et pour chaque type propose de nombreux supertypes et/ou sous-types. Il est légitime qu'un cogniticien ne désire pas être ennuyé par tous ces types lorsqu'il exploite WordNet. CGKAT répond à ce besoin de deux façons :
1) en offrant des mécanismes de recherche et de visualisation permettant de gérer de grandes taxinomies, et permettant une inclusion dynamique des types de WordNet dans le treillis des types de concepts, et
2) en permettant l'usage de filtres lors de la visualisation de ce treillis (cf. section 5.2.2.1) ; ainsi par exemple, l'utilisateur peut demander que les types provenant de WordNet ne soient pas affichés lorsqu'il navigue dans le treillis.
Nous montrerons également dans la section 7.2.2.3 comment CGKAT facilite l'utilisation des types de WordNet lors de la constitution d'une ontologie via la création manuelle d'un fichier de sauvegarde au format BCGCT (le format lu par CoGITo).

1. Il existe peu de procédures automatiques de comparaison et d'unification de taxinomies de types. Citons néanmoins le "Hierarchy Match" (Knight & Luk, 1994) pour les hiérarchies de types atomiques auxquels sont associés des définitions en langage naturel.

5.4 Une ontologie de haut niveau pour les types de concepts

Le mécanisme d'inclusion dynamique de WordNet dans le treillis des types de concepts est indépendant d'une quelconque ontologie de haut niveau. C'est pourquoi nous avons présenté l'exploitation de WordNet avant de discuter d'une ontologie de haut niveau qui pourrait compléter/ structurer WordNet et permettre encore de mieux l'utiliser.

Dans WordNet, seuls les ensembles de synonymes formés avec des noms ou des verbes sont structurés par des relations de spécialisation, et nous avons noté que même pour les actions, il valait mieux n'utiliser que les ensembles de synonymes formés avec des noms. Aussi nous n'inclurons dans notre ontologie de haut niveau pour les types de concepts, que les catégories de plus haut niveau pour les ensembles de synonymes formés avec des noms.

Les ontologies de haut niveau contiennent de nombreuses catégories similaires : celles par exemple représentant les notions d'objet physique, de processus, d'état, de collection, de propriété ou d'attribut, de valeur, et de mesure. Des catégories relatives à ces notions sont également présentes dans WordNet mais parfois de manière dispersée, de telle sorte qu'il est nécessaire de les regrouper. Lors de la constitution de notre ontologie de haut niveau, notre philosophie a été *d'organiser et de compléter* les catégories de haut niveau de WordNet avec d'autres catégories que nous avons jugées utiles pour guider la représentation et la recherche de connaissances.

Pour cela, nous nous sommes inspiré d'ontologies de haut niveau, notamment celle de Sowa (1992), celle de Sowa (1995b) et celle de Tepfenhart (1992). L'annexe 3 montre des figures pour ces trois ontologies ainsi que pour d'autres ontologies de haut niveau et pour l'organisation des catégories de haut niveau de WordNet.
Nous montrerons dans la section suivante comment nous avons également introduit dans notre ontologie, des types et des modèles génériques relatifs à la constitution de modèles de tâche.

La figure 5.1 montre les types de concepts de plus haut niveau de l'ontologie que CGKAT propose (mais n'impose pas) à ses utilisateurs (les types provenant de WordNet commencent par "W_"). Nous décrivons maintenant cette ontologie, ses principes, ses types et leur utilité.

5.4.1 Les distinctions élémentaires

S'il ne représente pas une caractéristique élémentaire, un type peut se définir en fonction d'un ou plusieurs autres types. Plus les caractéristiques utilisées pour définir les types sont élémentaires, plus les possibilités de recherches (avec une caractéristique ou une combinaison de caractéristiques) sont grandes. Les caractéristiques élémentaires permettent donc de définir les types et de les classer de manière à pouvoir les rechercher efficacement (manuellement ou automatiquement). De plus, des schémas, des règles ou des contraintes (e.g. des liens d'exclusion) peuvent être associés aux caractéristiques élémentaires et ainsi permettre de guider et vérifier la création d'une ontologie et la représentation des connaissances.

Les caractéristiques élémentaires peuvent se présenter sous forme de distinctions :
- Entité / Situation,
- Propriété / Chose qui n'est pas une propriété,
- Rôle (une situation particulière) / Chose ayant un rôle (chose dans une situation particulière),
- Changeant / Non changeant, Physique (matériel) / Non physique, Spatial / Non spatial, Temporel / Non temporel, Rationnel / Irrationnel, etc.

La distinction Entité / Situation provient de la "sémantique des situations" (Barwise & Perry, 1983) et est à la base de l'ontologie de Sowa (1992). Une situation est une configuration finie d'un aspect d'un monde réel ou imaginaire (cf. section 4.3.2.4 à la page 86), e.g. un état ou un processus. Une entité est une chose qui n'est pas *considérée* comme une situation, mais comme un *constituant de base* pour des situations. Ainsi, l'existence d'une entité ou de ses relations avec d'autres entités est une situation. Une proposition qui décrit une situation est une entité tandis que l'existence de

cette proposition est une situation. Nous avons noté en section 4.3.2.5, l'importance de ces notions pour permettre de représenter des relations explicites entre situations, entités et descriptions de situations. La distinction entre entité et situation permet de donner des signatures adéquates à de nombreux types de relations, et ainsi mieux définir et contraindre leur usage. Par exemple, nous pensons que des relations de type Point_in_Time ("position dans le temps") ou Duration ("durée dans le temps") ne peuvent s'appliquer qu'à des situations et non à des entités. Enfin, la distinction entre entités et situations semble être la plus élémentaire puisque une propriété est une entité et que les autres distinctions peuvent être exprimées en utilisant des propriétés particulières (Physique / Non physique, Changeant / Non changeant, etc.). Dans notre ontologie, nous avons déclaré un lien d'exclusion entre Entity et Situation, ce qui empêche la création de sous-types communs à ces deux types[1].

Nous considérons un rôle comme un type de situation particulière, c'est-à-dire un type d'ensemble particulier de relations entre entités et/ou situations. Différents types de choses (entités ou situations) peuvent jouer des rôles similaires, et donc avoir des relations similaires avec d'autres choses[2]. Par exemple, le rôle de "conducteur d'un engin roulant" peut être joué par un homme, un animal savant, un robot ou un personnage imaginaire dans un dessin animé. La notion de "chose jouant un rôle" peut donc être croisée avec n'importe quelle autre notion : tout type peut être spécialisé pour exprimer le fait qu'un individu de ce type peut jouer un rôle particulier. C'est pourquoi, dans notre ontologie nous proposons les types Concept_playing_a_role, Entity_playing_a_role et Situation_playing_a_role (nous avons donné le nom de "Concept" au supertype de tous les types de concepts, mais "Thing" aurait également été acceptable et aurait induit le nom de "Thing_playing_a_role" au lieu de "Concept_playing_a_role"). Nous présenterons plus loin les différents sous-types que nous avons trouvés pour ces types. Notons que dans les noms de ces types, le rôle n'est pas confondu avec celui qui le tient, ce qui n'est généralement pas le cas dans la plupart des ontologies de haut niveau.

Sowa (1995b) distingue les notions de rôle et de médiation :
1) Une chose *considérée* comme jouant un rôle particulier serait une chose entretenant des relations particulières avec d'autres choses (e.g. Epouse, Parent, Employé, Pilote).
2) Une chose *considérée* comme une médiation particulière serait une chose qui conduit à l'établissement de relations particulières (e.g. Mariage, Attendre un enfant, Entreprise, Aviation). Toute chose peut être *considérée* comme appartenant à une "catégorie naturelle", comme jouant un rôle particulier et comme étant une médiation particulière.
Cependant la distinction entre les notions de rôle et de médiation est controversée. Il est en fait très difficile de trouver des catégories de médiation qui ne puissent pas être aussi avantageusement considérées comme des catégories de rôle ou des "catégories naturelles". Par exemple, la catégorie Process peut être à la fois être considérée comme une "catégorie naturelle" et un type de médiation puisque tout processus est source de certaines relations entre objets. Aussi, l'ontologie de CGKAT propose le type "Concept_being_a_mediation" comme sous-type de "Concept_playing_a_role" et ne lui donne aucun sous-type : un point d'entrée est donné à l'utilisateur pour la notion de médiation et ce dernier peut préciser et spécialiser cette catégorie s'il le désire.

1. Ce lien d'exclusion empêche par exemple l'utilisateur de sous-typer un type de processus avec un type d'entité, ou inversement. Nous avons constaté que ceci arrive fréquemment lorsque les noms des types ne sont pas suffisamment explicites pour refléter leur nature (processus ou résultat de processus), e.g. Description, Observation, Objection et Construction. Les signatures des types de relations permettent d'effectuer des vérifications similaires lors de la construction des GCs.

2. Nous ne considérons pas la réciproque comme vraie. Nous détaillerons un peu plus dans la section 5.4.4 à la page 139 ce que nous entendons par la notion de "chose jouant un rôle".

L'ontologie de Sowa (1995b) a les catégories suivantes comme primitives (nous donnons des noms explicites à ces catégories ; les noms originaux figurent en annexe 3) :
- Chose naturelle / Chose jouant un rôle / Chose étant une médiation
- Chose concrète (physique) / Chose abstraite (non physique)
- Chose considérée comme changeante / Chose considérée comme non changeante.

Nous venons de voir certains des problèmes liés à la première partition. Dans les deux dernières, notons que les notions ne sont pas primitives puisqu'elles pourraient être décomposées en utilisant des propriétés : Physique / Non physique et Changeant / Non changeant. Dans l'ontologie de Sowa (1995b), la notion de propriété n'apparaît pas dans les premiers niveaux. Ces premiers niveaux sont construits par produit croisé des sept types primitifs en 12 sous-types (3 x 2 x 2). Comme aucune structuration arbitraire n'est choisie, chacun de ces 12 types a plusieurs parents. Le problème est qu'il est alors difficile de présenter de manière ergonomique ces types et leurs spécialisations, aussi bien avec une liste indentée qu'avec un graphe. Dans le premier cas, de nombreuses branches identiques apparaissent ce qui rend difficile l'obtention d'une vision globale. Dans le second cas, les croisements entre les relations rend le graphe peu lisible. Ainsi, aussi étonnant que cela puisse paraître a priori, certaines structurations *arbitraires* doivent parfois être effectuées pour faciliter la compréhension d'une ontologie.

Dans l'ontologie proposée par CGKAT, une structuration *naturelle* est imposée par la distinction entre entités et situations qui, nous l'avons noté, semble plus générale (ou élémentaire) que les autres (cette distinction oriente la construction de l'ontologie et réduit les problèmes de croisements). Au même niveau que les types Entity et Situation, nous n'avons trouvé nécessaire de rajouter que deux catégories : Concept_playing_a_role et Concept_used_by_an_agent. Nous détaillerons ces catégories plus loin.
Nous avons partitionné les entités et les situations suivant des critères pertinents compte-tenu de leur nature : nous n'avons donc pas retenu la distinction Changeant / Non changeant pour partitionner les entités, ni la distinction Physique / Non physique pour partitionner les situations (mais comme le montre la figure 5.1, des catégories d'états ou de processus pouvant selon certains points de vus être considérés comme abstraits, peuvent aisément être trouvées par l'utilisateur dans notre ontologie).

Nous détaillons maintenant les types de concepts de haut niveau de l'ontologie proposée par CGKAT.

5.4.2 Les entités

La distinction Physique (matériel) / Non physique semble naturelle et être un bon moyen de partionner les entités. Cependant, une entité physique est une entité spatiale particulière, i.e. constituée de matière. Une première distinction entre entités est donc entre entités spatiales et non spatiales. Les entités temporelles (temps et points ou intervalles dans le temps) ne sont pas des entités spatiales, de même que les entités représentant des informations (structures de données, propositions, propriétés et mesures) et que nous appelons "entités abstraites". Les collections peuvent être aussi bien rassembler des choses spatiales que non spatiales, et peuvent donc suivant leur contenu être considérées comme spatiales (e.g. les entités spatiales composites) ou bien non spatiales (e.g. les structures de données). Aussi, avons-nous donné à Entity les premiers sous-types suivants : Collection, Temporal_entity, Spatial_entity et Abstract_entity. La figure suivante montre ces types, quelques-uns de leurs sous-types et les liens d'exclusion qui les relient. Nous détaillons ci-après ces catégories.

Entity -- *something that can be involved in a situation*

Collection -- *a group or structure of entities or situations, e.g. Ordered_Collection*

%Temporal_entity -- *point or interval in time, e.g. Time_period, Point_in_time, W_time/n*

%Spatial_entity -- *an entity which occupies a space region*

W_space/n -- *the unlimited 3 dimensional expanse in which everything is located*

W_location/n -- *a point or extent in space*

Physical_entity -- *a spatial entity which is made of matter (see Material_entity)*

W_object__inanimate_object__physical_object -- *a nonliving entity*

W_life_form__organism__being__living_thing -- *a living entity, e.g. a plant*

Imaginary_spatial_entity -- *e.g. Cartoon_Character*

Spatial_entity_with_a_special_property -- *e.g. Smooth_entity, Lumpy_entity*

Abstract_entity -- *information or representation of an information*

Representation_entity -- *Lexical_data (e.g. Integer, CG), Non_lexical_data (e.g. Image)*

Proposition -- *information (situation description), e.g. Description, Argument, Hypothesis, Belief*

Property -- *a dimension of an object, e.g. Mass, Volume, Form, Color, Speed, Intelligence*

Measure -- *a measure of a property of an object, e.g. W_measure__quantity__amount__quantum*

%Entity_playing_a_role -- *e.g. Recipient_entity, Possessed_entity, Part_entity, Material_entity*

Liens d'exclusion entre types (types de WordNet exceptés) :

Abstract_entity Excl (Temporal_entity, Spatial_entity);

Physical_entity Excl Imaginary_spatial_entity;

Representation_entity Excl (Proposition, Property, Measure);

Proposition Excl (Property, Measure);

Property Excl Measure;

Figure 5.9: *Les premiers sous-types de Entity dans l'ontologie de CGKAT (les types précédés de '%" ont plusieurs supertypes).*

5.4.2.1 Les collections

Une collection peut regrouper des entités ou des situations. Cependant une collection de situations n'est pas une situation (c'est l'existence de ces situations ou de cette collection de situations qui forme une situation). Une collection est donc bien toujours une entité.

Une collection d'entiers n'est pas un entier mais peut former une structure de données, tandis qu'une collection de structures de données peut être considérée comme une nouvelle structure de donnée. De même, une collection d'entités physiques peut dans certaines conditions être considérée comme une nouvelle entité physique. Nous n'avons pas défini de lien d'exclusion entre Collection et les autres catégories d'entités. Ainsi, un type particulier peut être défini à la fois comme sous-type de Collection et comme sous-type d'une autre catégorie d'entités ; il peut ainsi hériter des propriétés et contraintes associées à ces deux catégories.

WordNet propose deux catégories de collections d'entités. Nous les avons regroupées sous Collection (voir figure 5.3). Il est ainsi possible de retrouver par un unique point d'entrée un nombre important de types relatifs aux collections et d'utiliser des concepts de ces types avec les relations signées sur le type Collection. Comme le montre la figure 5.3, nous avons complété les types de WordNet par un classement basé sur certaines propriétés mathématiques des collections : ouvertes, fermées, ordonnées, conjonctives, vides, etc. Nous n'avons cependant pas organisé les types de WordNet en fonction de ce classement. C'est donc à l'utilisateur de spécialiser avec les types de son application ceux de WordNet et/ou ceux que nous avons rajoutés.

Dans Pfeiffer & Hartley (1992), les auteurs montrent que le type Empty_set peut être considéré comme un supertype d'un certain nombre de types de collection. Nous avons repris leur ontologie pour sous-typer le type Empty_set.

```
Empty_set -- its subtypes come from (Pfeiffer&hartley, 1992)
     Empty_tuple
          Singleton_tuple
               %List__Ordered_open_collection -- without duplicate elements
                    %Tuple__Ordered_closed_collection
                    %Ordered_open_collection_ADT
     Disjunctive_set
          Disjunctive_group
               %Singleton_group
                    .Conjunctive_group
                    %Tuple__Ordered_closed_collection
          Singleton_set
               %Singleton_group
                    .Conjunctive_group
                    %Tuple__Ordered_closed_collection
               Conjunctive_set
                    %List__Ordered_open_collection -- without duplicate elements
```

Figure 5.10: *Les sous-types de Empty_set selon (Pfeiffer & Hartley, 1992)*

L'ontologie proposée par défaut par CGKAT est une ontologie de types atomiques mais des définitions pour certains de ces types peuvent également être aisément chargées si l'utilisateur le désire. Ces définitions, par conditions nécessaires et/ou suffisantes ou par schéma prototypique, peuvent guider l'utilisateur dans la création de ses GCs. Une définition pour le type Collection est :

*NSC for Collection (x) are [Entity:*x]→(Chrc)→[Cardinality].*

L'utilisateur peut bien sûr ajouter d'autres définitions au type Collection et à ses sous-types.

5.4.2.2 Les entités temporelles

Le temps est une entité temporelle. Aussi, nous avons classé comme sous-type de Temporal_entity, le type W_time/n qui dans WordNet représente cette notion de temps et qui est spécialisé par des types relatifs au temps géologique, biologique, cosmique, passé, présent, futur, etc.
Pour les signatures des types de relations temporelles proposés par CGKAT (cf. section 5.6), nous avions également besoin de deux types particuliers qui ne sont pas fournis par WordNet : le type Point_in_time et le type Time_period.

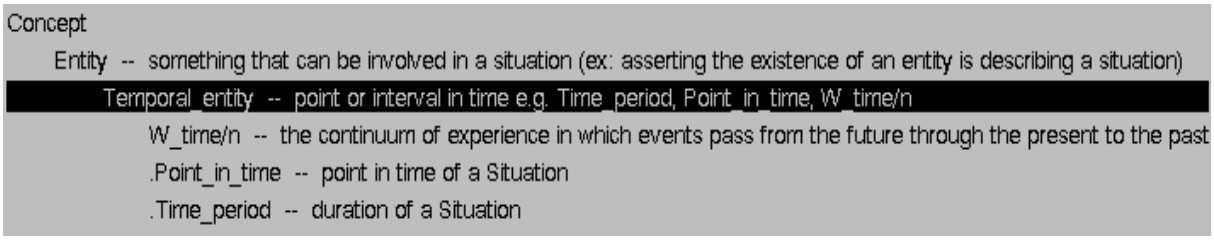


Figure 5.11: *Des supertypes et des sous-types pour Temporal_entity.*

*NSC for Temporal_entity (x) are [Entity:*x]→(Chrc)→[Temporal_dimension].*

Le type WordNet W_measure__quantity__amount__quantum, que nous avons classé sous Measure, regroupe entre autres des types permettant de représenter des mesures de temps (i.e. des mesures de dimension temporelle).

Le type Temporal_entity a des liens d'exclusion avec le type Abstract_entity. Nous n'avons pas mis de liens d'exclusion entre Temporal_entity et Spatial_entity de manière à autoriser la création de sous-types communs représentant des entités spatio-temporelles telles que celles que manipulent certains physiciens. Notons que nous n'avons pas retenu pour les situations, de distinction Temporel / Non temporel, ni Spatial / Non spatial, puisqu'une situation est une configuration finie d'un aspect d'un monde (réel ou imaginaire) *dans une région limitée de l'espace et du temps*.

5.4.2.3 Les entités spatiales

Le type Spatial_entity est important pour réunir toutes les entités qui ont des dimensions spatiales, comme les régions de l'espace, les objets physiques et beaucoup d'objets imaginaires (e.g. Mickey). Cela permet par exemple de signer des relations spatiales avec ce type et de pouvoir ainsi les appliquer aussi bien à des objets *occupant* des régions de l'espace qu'à ces régions elles-même. Spatial_entity a des liens d'exclusion avec les types Abstract_entity et Temporal_entity.

*NSC for Spatial_entity (x) are [Entity:*x]→(Chrc)→[Spatial_dimension].* //e.g. longueur, volume

Nous avons réuni cinq catégories de haut niveau de WordNet sous Spatial_entity (voir figure 5.1) : deux sont relatives à des *régions de l'espace*, une troisième aux *objets physiques non-vivants*, une quatrième aux *objets vivants* et une cinquième aux *cellules* ("constituants de base des organismes" ; dans WordNet, la catégorie des cellules n'est pas classée parmi les objets vivants). Nous avons réuni les trois dernières catégories sous le type Physical_entity (la catégorie des cellules a été classée parmi les catégories d'objets vivants).

*NSC for Physical_entity (x) are [Spatial_entity:*x]→(Chrc)→[Physical_dimension].* //e.g. la masse

Le type WordNet W_measure__quantity__amount__quantum regroupe entre autres des types permettant de représenter des mesures de dimension spatiale.

Par ailleurs, comme le note Sowa (1995b), les entités spatiales peuvent se classer suivant diverses propriétés, par exemple leur aspect discontinu (e.g. un livre) ou continu (e.g. la mer). Nous n'avons pas trouvé dans WordNet de catégorie de haut niveau pour exprimer de telles propriétés, mais afin d'offrir un point d'entrée à l'utilisateur pour classer des entités spatiales suivant de telles propriétés, nous avons rajouté la catégorie Spatial_entity_with_a_special_property et nous avons classé dans cette catégorie les types de Sowa (1995b) relatifs à ces propriétés).

5.4.2.4 Les entités abstraites (moyens de représentation, propositions, propriétés, mesures)

Nous considérons comme abstraites les entités qui ne sont ni spatiales, ni temporelles. Une entité abstraite est, ou bien représente, une information c'est-à-dire le sens (contenu sémantique) d'une description de situation. Ainsi, le type Abstract_entity regroupe 1) les entités servant de ***moyen de représentation*** (e.g. les éléments de documents et les structures de données), 2) les ***propositions*** (***descriptions de situations***, e.g. narrations, croyances, hypothèses), 3) les ***propriétés***, et 4) les ***mesures de propriétés***. Nous avons montré que les collections ne pouvaient être classées parmi les entités abstraites (comme le fait par exemple Sowa (1995b)) sinon il serait par exemple impossible d'exprimer le fait que des entités physiques composites sont des collections ou que des collections d'entités physiques ont les mêmes propriétés que les entités physiques (masse, volume, etc.).

5.4.2.4.1 Les entités de représentation

Les types W_representation/n1 et W_communication/n de WordNet regroupent de nombreux media de représentation ou de communication. Nous avons ajouté
1) la notion d'élément de document,
2) des types d'entités représentant des processus (extraits de Sowa (1995b)), et
3) les distinctions de Sowa (1992) pour les entités de représentation : lexicales / non lexicales, atomiques / structurées.

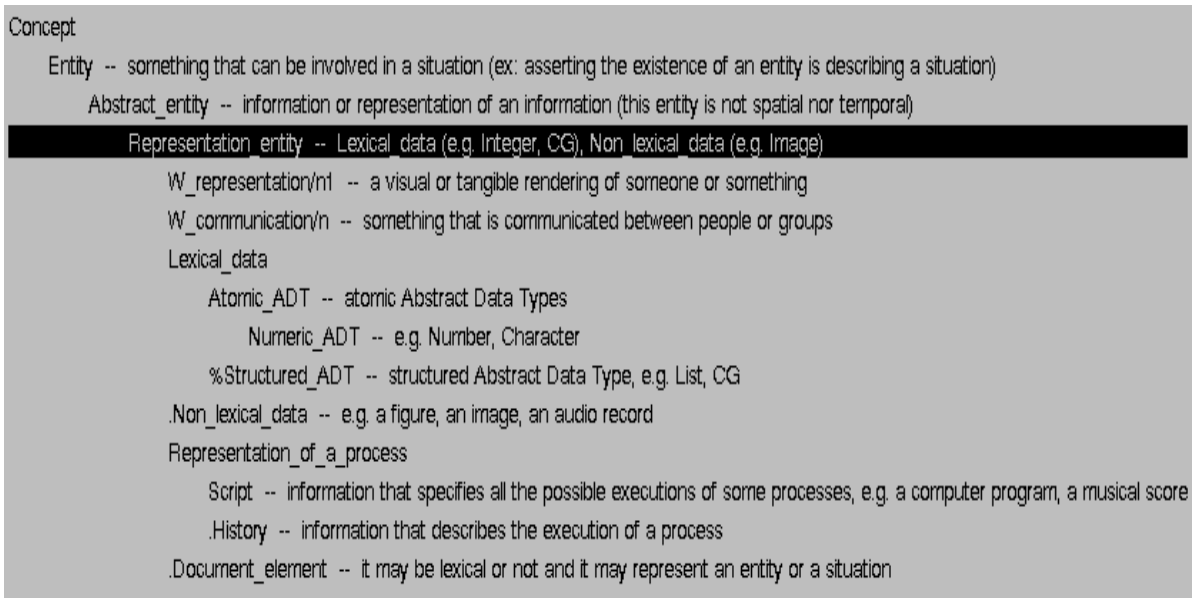


Figure 5.12: *Des supertypes et des sous-types pour Representation_entity.*

Nous avons classé les entités lexicales structurées, i.e. les ***structures abstraites de données***, en utilisant les grandes catégories que l'on utilise en informatique (e.g. liste, tableau, pile, dictionnaire) et en spécialisant des catégories que nous avions données pour les collections.
De plus, parmi les entités de représentation structurées, on peut classer les divers ***modèles*** que le cogniticien doit construire au cours de l'acquisition de connaissances afin de représenter et structurer une expertise. Nous offrons donc un point d'entrée pour classer ces modèles et nous le spécialisons avec les types de modèles proposés par la méthodologie KADS-I (voir figure 5.13).

```
^Lexical_data
^Collection -- a group or structure of abstract/concrete entities or situations, e.g. Ordered_Collection
    %Structured_ADT -- structured Abstract Data Type, e.g. List, CG
        .%Record__Structure -- ADT of an unordered closed collection
        %Ordered_open_collection_ADT
            .List_ADT -- ADT of an ordered open collection
            .Stack -- (may be implemented with a List_ADT or an Array)
            .Keyed_collection_ADT -- (idem) e.g. a dictionnary ADT
        %Ordered_closed_collection_ADT
            .Array -- ADT of an ordered closed collection
            .Queue -- (may be implemented with a List_ADT or an Array)
        %Graph_ADT
            .CG -- conceptual graph
        Model_ADT
            %KADS1_Model
                .KADS1_Organisational_Model
                .KADS1_Application_Model
                .KADS1_Task_Model
                .KADS1_Conceptual_Model
                .KADS1_Design_Model
                KADS1_Model_of_Expertise
                    .KADS1_Model_of_Problem_Solving_Expertise
                    .KADS1_Model_of_Cooperation_Expertise
                    .KADS1_Model_of_Communication_Expertise
                .KADS1_Domain_Layer
                .KADS1_Inference_Layer
                .KADS1_Task_Layer
                .KADS1_Strategy_Layer
```

Figure 5.13: *Des supertypes et des sous-types pour Structured_ADT.*

5.4.2.4.2 Quelques définitions pour les types de modèles préconisés par KADS_I

Voici une définition d'un modèle d'expertise dans KADS-I (Wielinga & al., 1992) (actuellement dans CommonKADS (Breuker & van de Velde, 1994), le niveau tâche est plus complet et le niveau "Stratégie" n'existe plus) :

NSC for KADS1_Model_of_Expertise (x) are
*[KADS1_Model :*x]- { (Part)→[KADS1_Strategy_layer];*
(Part)→[KADS1_Task_layer];
(Part)→[KADS1_Inference_layer];
(Part)→[KADS1_Domain_layer];
}.

Le modèle conceptuel représente l'expertise de résolution de problème, peut aussi représenter les expertises de communication et de coopération avec l'utilisateur. Aussi, comme nous le montrons dans (Martin, 1993a, 1993b, 1994), si ces expertises sont importantes, il peut être intéressant de les modéliser avec un modèle d'expertise tout comme l'expertise de résolution de problème. D'où la composition typique d'un modèle conceptuel dans KADS-1 (nous avons utilisé une définition par "conditions typiques") :

TC for KADS1_Conceptual_Model (x) are
*[KADS1_Conceptual_Model : *x]- { (Part)→[KADS1_Model_of_Problem_Solving_Expertise];*
(Part)→[KADS1_Model_of_Communication_Expertise];
(Part)→[KADS1_Model_of_Cooperation_Expertise];
}.

5.4.2.4.3 Les propositions

Une situation peut être décrite par diverses propositions, et celles-ci exprimées par diverses entités de représentation (phrases, images, graphes, formules logiques, etc.). Une entité de représentation peut être inscrite sur/dans plusieurs supports physiques (feuille de papier, disque, mur, etc.). Les liens entre situations, propositions et entités de représentation sont représentés dans le schéma suivant :

[Situation]→(Descr)→[Proposition]→(Stmt)→[Representation_entity].

Nous avons posé des liens d'exclusion entre Representation_entity et Proposition afin que l'utilisateur ne puisse pas confondre les deux notions. Ces deux types ont également des liens d'exclusion avec Situation puisqu'ils sont sous-types de Entity.

Lorsqu'une proposition a été assertée via une entité de représentation, un certain nombre d'inférences peuvent être effectuées : cette proposition a un auteur, elle a été pensée par son auteur à un instant donné (même si cet auteur pensait que cette proposition était fausse), elle a été représentée par cet auteur à un instant donné, etc. Des outils d'analyse de textes, comme celui de Charniak (1976) sont basés sur de telles inférences. Nous proposons dans notre ontologie les définitions suivantes pour représenter quelques relations importantes qu'entretiennent les propositions avec d'autres objets :

```
NSC for Proposition (x) are
   [Entity : *x]- { (Object)←[Think]- { (Experiencer)→[Cognitive_agent];
                                        (Situation_temporal_location)→[Temporal_entity];
                                      }
                    (Result)←[Make_statement]-
                                          { (Object)→[Statement];
                                            (Agent)→[Cognitive_agent];
                                            (Situation_temporal_location)→[Temporal_entity];
                                          }
                  }.

Descr (Situation,Proposition).   //Déclaration du type de relation Descr, avec une signature
Stmt(Proposition,Representation_entity).
SC for Proposition (x) are [Proposition : *x]←(Descr)←[Situation].
SC for Proposition (x) are [Proposition : *x]→(Stmt)→[Representation_entity].
                          // [Entity : *x] n'aurait pu être utilisé à cause des signatures des relations

NSC for Statement_author (x,y) are
    [Proposition : *x]←(Object)←[Make_statement]→(Agent)→[Cognitive_agent : *y].
NSC for Statement_time (x,y) are
   [Proposition : *x]←(Object)←[Think]→(Situation_temporal_location)→[Temporal_entity : *y].
NSC for Believer (x,y) are
   [Proposition : *x]←(Object)←[Believe]→(Agent)→[Cognitive_agent : *y].

Rhetorical_relation (Proposition,Proposition).
Modality (Proposition,Modality).     Modality < Contextualizing_relation_from_a_proposition.
   //La relation Modality est signée sur les types de concepts Proposition et Modality
   //Nous donnerons dans la section 5.6.5.4 d'autres relations "contextualisantes" portant sur un
   // concept de type Proposition.
```

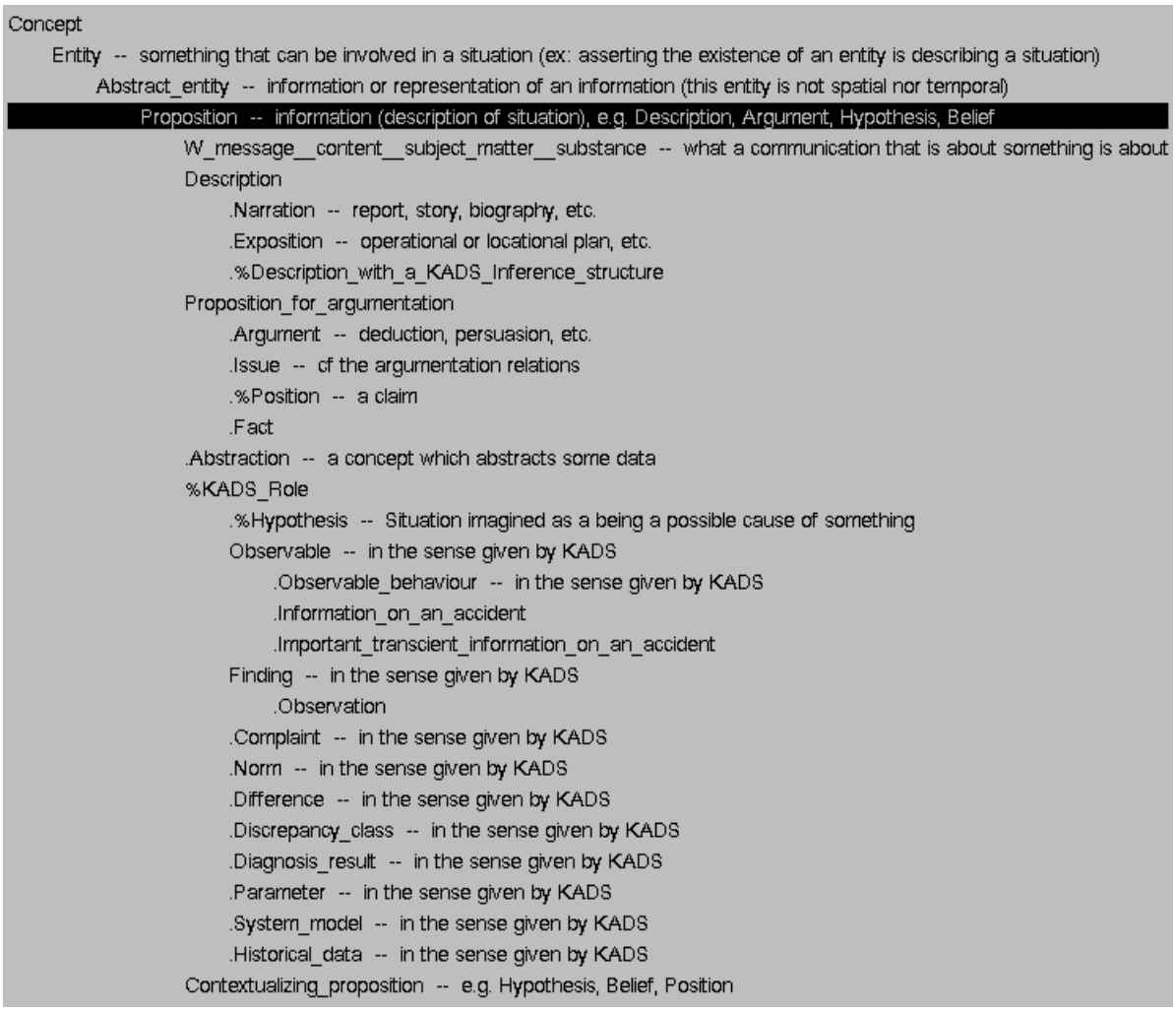


Figure 5.14: *Des supertypes et des sous-types pour Proposition.*

La catégorie de haut niveau de WordNet W_message__content__subject_matter__substance nous a semblé parfaitement adéquate[1] pour sous-typer Proposition. Dans WordNet, cette catégorie a pour parente W_communication/n qui en dehors de cette catégorie ne rassemble que des media de communication, i.e. des entités de représentation. Nous avons donc donné un nouveau parent à W_message__content__subject_matter__substance. W_communication/n ne contient donc maintenant que des entités de représentation.

Nous avons également rassemblé sous Proposition, des types de description et d'argumentation. Les types d'argumentation, Argument, Issue, Position et Fact, sont fournis par le système hypertexte PHIDIAS (McCall & al., 1990), avec des relations d'argumentation, pour permettre et guider la création par un ou plusieurs utilisateurs d'un document hypertexte complexe (la structure argumentative facilite la conception et les révisions d'une base d'informations)[2].

1. Elle contient par exemple les types W_ information__info, W_hypothesis/n et W_proposition, lequel a parmi ses sous-types : W_theorem/n, W_conclusion/n1, W_axiom, etc.

2. De nombreux autres systèmes hypertextes sont également focalisés sur ce sujet (e.g. Sepia, AAA, gIBIS, JANUS) et fournissent des types de relations d'argumentation pour guider les utilisateurs dans la création de structures argumentatives. Nous avons intégré ces types de relations dans notre ontologie, ainsi d'ailleurs que les relations rhétoriques de Mann & Thompson (1987) qui servent également à structurer des textes mais qui sont plus utilisées dans les travaux sur les explications. L'utilisateur de CGKAT, peut donc, comme dans ces systèmes hypertextes, exploiter ces types de concepts ou de relations pour typer et structurer des éléments de documents. Il peut également les spécialiser et les compléter, ce qui est plus rare dans les systèmes hypertextes.

Les hypothèses, les croyances, les normes sont des propositions. Ces noms se retrouvent dans les noms des types de rôles (entrées-sorties d'une inférence) fournis par KADS. Rappelons que dans une "instance de tâche", les rôles sont connectés par des pointeurs ou des relations à des termes ou des connaissances (faits, règles, etc.) du domaine, et représentent leur usage (leur rôle) lors de l'exécution de l'inférence. Si l'on représente une structure d'inférences dans le formalisme des GCs et en utilisant notre ontologie, les types des concepts utilisés pour représenter les rôles devraient logiquement être soit sous-types de Proposition, soit sous-types de Representation_entity (et plus précisément, d'un type tel que "Representation_entity_playing_a_role"). Comme nous désirions sous-typer Proposition avec des types aux noms semblables à ceux des types de rôles fournis par KADS, nous avons rassemblé ces types sous KADS_Role et introduit ce dernier comme sous-type de Proposition.

Finalement, nous avons introduit le type Contextualizing_proposition qui permet de regrouper des types de contextes "contextualisants" (cf. début de la section 4.3.2 et section 4.3.2.5).

5.4.2.4.4 Les propriétés et les mesures

Tout objet (entité ou situation) qui n'est pas une propriété élémentaire, possède des propriétés (ou caractéristiques ou encore dimensions[1]). Des exemples de propriété sont la masse, la couleur, la vitesse, l'âge, le coût, la fiabilité. Comme le montre l'exemple de la fiabilité, la valeur d'une propriété peut être le résultat d'un calcul sur les valeurs d'autres propriétés. Un objet se distingue d'autres objets par ses propriétés ou les valeurs de ses propriétés, et une valeur de propriété peut se mesurer par comparaison avec d'autres valeurs de propriété (e.g. la *longueur* d'une voiture peut se mesurer par comparaison avec la *longueur* du mètre étalon).

Nous considérons une "mesure" comme un objet composite qui contient :
1) une ou plusieurs valeurs numériques ou symboliques, et
2) une unité de mesure et une méthode de comparaison.
Des exemples de "mesures" sont : "égal à 35 kg", "entre 647 nm et 700 nm", "plus léger que l'air".
Nous avons le schéma : [Concept]→(Chrc)→[Property]→(Measure)→[Measure].

Notons la différence entre les deux notions que peut dénoter le mot "rouge" : 1) un certain type de couleurs (celles ayant une longueur d'onde entre 647 nm et 700 nm), et 2) une mesure d'une longueur d'onde ("entre 647 nm et 700 nm"). La première de ces notions peut être définie à l'aide de la seconde :

NSC for Couleur_rouge (x) are [Couleur : *x]→(Measure)→[Mesure_de_couleur_rouge].

La couleur d'une voiture particulière peut ainsi être représentée comme suit :

[Voiture : #234]→(Chrc)→[Couleur_rouge].

Il est également possible, dans le formalisme des GCs, de représenter une couleur particulière en utilisant un marqueur individuel conforme au type Couleur. D'où par exemple :

[Voiture : #234]→(Chrc)→[Couleur : Rouge].

Dans ce dernier cas cependant, la notion de mesure n'est pas représentée et, tandis que le type Couleur_rouge peut être sous-typé, le marqueur individuel rouge ne peut être spécialisé.

Sowa (1992) ne représente pas des adjectifs comme "rouge", "rapide" ou "froid" comme des mesures : il crée une catégorie à part, celle des "attributs", qui lui permet de représenter l'existence d'une "voiture rouge" de la manière suivante : [Voiture]→(Attr)→[Rouge] (Rouge étant un type d'attribut). Il relie les attributs aux propriétés en les considérant comme leurs instances. Ainsi, selon lui, [Voiture]→(Attr)→[Rouge] peut se transformer en [Voiture]→(Chrc)→[Couleur : Rouge]. Outre cette délicate transformation, on peut constater que Couleur et donc tous les autres types de

1. Toute propriété d'un objet peut être considérée comee une dimension par rapport à laquelle cet objet peut être comparé à d'autres objets : absence ou présence de cette dimension dans les objets et, lorsqu'elle est présente, comparaison des valeurs des objets pour cette dimension.

propriétés sont alors des types du second ordre, ce qui pose des problèmes techniques et ontologiques. De plus, les "attributs" ne sont pas reliés/comparés à des mesures de propriétés (e.g. "entre 647 nm et 700 nm"). Enfin, utiliser des "attributs" serait équivalent à utiliser des marqueurs individuels de propriétés, alors que des types de propriétés peuvent être plus adéquats.

En conséquence, nous n'offrons pas dans notre ontologie de catégorie d'attributs dans le sens donné par Sowa. Pour nous, attribut est synonyme de propriété (ou de dimension), comme c'est le cas dans WordNet, où l'un des deux types de plus haut niveau relatifs aux propriétés est W_property__attribute__dimension, tandis que l'autre est W_attribute/n et admet pour sous-type le type W_property/n. Nous avons regroupé ces deux types de haut niveau sous Property. Notre ontologie de types de relations offre le type Attribut ainsi défini :

NSC for Attr (x,y) are [Concept : *x]→(Chrc)→[Property]→(Measure)→[Measure : *y].

D'où par exemple la possibilité d'écrire : [Voiture]→(Attr)→[Mesure_de_couleur_rouge].

D'autres types devront être ajoutés par l'utilisateur sous Property afin de structurer les propriétés de son application. Pour notre ontologie, nous avons déjà ajouté le type Property_of_a_proposition dont un sous-type est Modality.

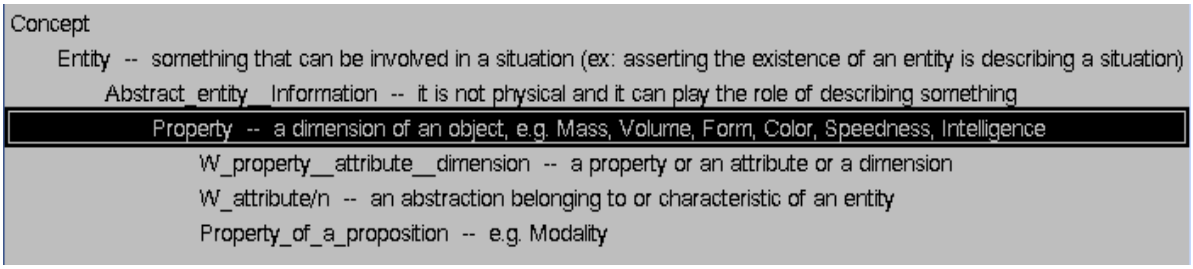


Figure 5.15: *Des supertypes et des sous-types pour Property.*

Une même mesure peut être représentée avec différentes entités de représentation (e.g. 1 kg et 1000 g). Aussi nous avons considéré le type Mesure comme un type d'information (entité abstraite) et non comme un type d'entité de représentation. Cela est néanmoins un choix que nous avons dû faire de manière assez arbitraire.

Nous avons sous-typé le type Mesure par le type W_measure__quantity__amount__quantum de WordNet. Ce type rassemble de nombreux systèmes de mesures mais aussi des unités de mesure. Cela n'est pas incorrect si l'on considère ces unités comme des mesures elles-mêmes. D'autres types de mesures peuvent être rajoutés, par exemple des types de mesures de modalités (Possible, Necessary, etc.).

Nous présenterons différents rôles que peuvent jouer les entités dans la section 5.4.4.

5.4.3 Les situations

Une situation est une configuration finie d'un aspect d'un monde (réel ou imaginaire) dans une région limitée de l'espace et du temps.

5.4.3.1 Définitions et remarques sur les processus, les états et les relations

La distinction entre processus et états semble naturelle et être un bon moyen de partionner les situations. Toutefois, les mots "processus" et "états" peuvent être interprétés de différentes façons. Par exemple, pour Sowa (1992), tout processus peut également être considéré comme un état lorsqu'il est vu comme une situation non changeante, et non comme une situation qui évolue, c'est-à-dire comme une séquence de sous-situations. Sa définition d'un état est "*une situation qui ne change pas de façon notable sur une période de temps considérée comme longue*", et celle d'un processus, "*une situation qui cause un changement durant une certaine période de temps*". Nous acceptons la définition pour un processus, mais pour nous, processus et états sont exclusifs. Nous définissons donc un *état* comme *une situation qui ne cause pas de changement durant une certaine période de temps*. Nous ne reprenons pas dans notre ontologie, la distinction entre processus considérés comme changeants et ceux considérés comme non changeants (chaque observateur d'un processus peut le considérer comme changeant ou pas suivant ses besoins ; nous ne pouvons donc classer des types en fonction de cette distinction).

Un *processus* crée des *relations* entre des objets (entités ou situations) et donc des *états*. Par exemple, l'action de résumer une description de situation en une autre description crée une relation entre ces deux situations. Notons la différence entre les deux types de relations suivants :

```
NSC for Résumer (x,y) are  [Résumer]-
                                  {  (Object)→[Proposition : *x];
                                     (Result)→[Proposition : *y];
                                  }.
NSC for Résumé (x,y) are  [Résumer]-
                                  {  (Past);     //unary relation
                                     (Object)→[Proposition : *x];
                                     (Result)→[Proposition : *y];
                                  }.
```

La définition du premier type de relation fait référence à un processus (que l'on suppose) en cours. Celle du second type de relation fait référence à un processus achevé et donc à l'état final qu'il a créé. La terminologie de ces types de relations (un verbe dans le premier cas, un nom commun dans le second) est conforme à cette distinction. Ainsi, "[Proposition : *x]→(Résumé)→[Proposition : *y]" se lit de manière classique (cf. section 4.2.1) : "Le Résumé de la Proposition x est la Proposition y", tandis que pour "[Proposition : *x]→(Résumer)→[Proposition : *y]" l'effort de lecture sera plus grand.

Des mots tels que "union", "agrégation" et "intersection" ont plusieurs sens : ils peuvent désigner des processus, des états créés par ces processus ou encore des entités nommées d'après ces états ou ces processus. Similairement nous avons noté que des mots comme "description", "observation" et "objection" peuvent désigner des processus ainsi que des entités que ces processus créent. Aussi, des types de concepts formés avec de tels mots sont ambigus et, avant que nous ne posions des liens d'exclusion entre Entity et Situation ainsi qu'entre Process et State, nous avons constaté des confusions de la part des utilisateurs lors des spécialisations de types ayant de tels noms ambigus. Les signatures des relations jouent le même rôle de vérification : ainsi par exemple, lorsque notre ontologie est utilisée, la pose de relations casuelles (Agent, Object, etc.) sur un concept de type autre que Process (ou un de ses sous-types) est refusé (citons le cas d'un utilisateur qui a tenté d'utiliser la relation Agent entre un état et un autre état au lieu d'utiliser la relation Consequence or Succ).

Situation -- *something that "occurs", or that is imagined, in a region of time and space (state or process)*

State -- *situation that is not changing and that does not make a change during a given period of time*
W_state/n -- *the way something is with respect to its main attributes*
W_psychological_feature/n -- *a feature of a mental life, e.g. a feeling, a viewpoint*
W_relation/n -- *an abstraction belonging to or characteristic of two entities or parts together*

Process -- *situation that can make a change during a given period of time*
Event -- *a process that makes a change during a period of time considered as very short, e.g. W_event/n*
W_act__human_action__human_activity -- *something that people do or cause to happen*
Problem_solving_process -- *e.g. the processes modelled in knowledge acquisition*
Problem_solving_task -- *a process such as the ones modelled in KA, e.g. a diagnostic*
%Problem_solving_method -- *a way of decomposing a problem solving task into subtasks*
Process_playing_a_role -- *Method (e.g. W_method/n), SubProcess (e.g. SubTask), etc.*

W_phenomenon/n -- *any state or process known through the senses rather than by intuition or reasoning*
%Situation_playing_a_role -- *e.g. W_result_outcome, W_method/n*

Liens d'exclusion (types de WordNet exceptés) : Process Excl State.

Figure 5.16: *Les premiers sous-types de Situation dans l'ontologie de CGKAT.*

5.4.3.2 Les états

Trois types de haut niveau de WordNet nous ont semblé relatifs à des états : W_state/n, W_psychological_feature/n et W_relation/n. Ces deux derniers types ont des noms et des commentaires un peu déroutants pour des types d'états, mais la liste de leurs sous-types est plus explicite.

W_psychological_feature/n ne rassemble pas des types de propriétés psychologiques mais bien des types d'états psychologiques. Voici les sous-types directs de W_psychological_feature/n :
- W_cognition__knowledge -- *the psychological result of perception and learning and reasoning*
- W_motivation__motive__need -- *the psychological feature that arouses an organism to action*
- W_feeling -- *the psychological feature of experiencing affective and emotional states*

Deux des sous-types (indirects) de W_cognition__knowledge sont particulièrement intéressants : W_knowledge_domain__knowledge_base et W_position__view__perspective. En effet, le premier fournit une classification de types de domaines ou de disciplines, tandis que le second fournit une classification de types de points de vue[1]. Afin de permettre à l'utilisateur de compléter ces classifications dès leurs racines dans WordNet, nous avons inséré entre chacun de ces deux types racines et leurs supertypes directs, deux types supplémentaires : AnyDomain dans le premier cas et AnyView dans le second. Nous avons préféré appeler le supertype des types de domaines, AnyDomain plutôt que Domain, pour marquer le fait qu'il est plus général que n'importe quel autre type de domaine et que donc il ne représente aucun domaine particulier. Idem pour AnyView. Ces deux types peuvent être utilisés dans les méta-informations de types ou de GCs, pour signaler que ces types ou ces GCs sont relatifs à n'importe quel domaine ou vue : (domain:AnyDomain; view:AnyView).

1. Les utilisateurs peuvent donc spécialiser ces types de domaines ou points de vue pour l'expertise qu'ils modélisent, et ces nouveaux types pourront alors dans une certaine mesure être comparés automatiquement, puisqu'ils seront organisés dans une structuration en (grande) partie commune.

Dans WordNet, W_relation/n est un sous-type de W_abstraction/n qui rassemble également les types W_time/n, W_space/n, W_attribute/n, W_measure__quantity__amount__quantum et W_set/n. Nous avons montré comment tous ces sous-types ont été répartis dans diverses catégories de l'ontologie par défaut de CGKAT. W_abstraction/n n'a maintenant plus de sous-type. De fait, cette catégorie était plutôt artificielle, et il était important de mieux organiser ses sous-types. Les sous-types de W_relation/n n'expriment pas des abstractions ni des relations en tant que telles mais des états fondés sur ces relations. Voici quelques sous-types directs de W_relation/n :
- W_interrelation__interrelationship__interrelatedness -- *mutual or reciprocal relation or relatedness*
- W_social_relation -- *a relation between living organisms, especially between people*
- W_kinship__family_relationship__relationship -- *state of relatedness or connection by blood or marriage*

5.4.3.3 Les processus

Deux types de haut niveau de WordNet sont relatifs à des processus : W_event/n et W_act_human_action__human_activity.

Sowa (1992) définit un événement comme un processus qui cause un changement dans un temps considéré comme très court. Nous avons repris cette notion avec le type Event que nous avons sous-typé par W_event, ce dernier et ses sous-types étant en accord avec la définition de Event. Un concept de type Event peut être utilisé pour représenter l'événement qui entraîne l'avènement ou la terminaison d'un état ou d'un processus. Par exemple :

[Aller_à_son_travail]→(Terminaison)→[Arrivée_à_son_travail]←(Causation)←[Etre_à_son_travail].

Notre ontologie offre les types de relations suivantes signées sur des concepts de type Situation : Condition, Causation, Task_Pred, Terminaison, Consequence, Task_Succ. Un automate d'états finis ou un réseau de Petri peuvent donc être représentés (bien sûr des règles ou des fonctions doivent être ajoutées pour une représentation plus complète de la sémantique des relations utilisées et ainsi permettre l'exécution de ces représentations ; notons à ce propos que des règles ou des fonctions peuvent[1] toujours être utilisées, en complément des définitions de types, pour compléter la représentation de la sémantique de types de concepts ou de relations, même des types de relations statique).

Nous avons ajouté le type Problem_solving_process qui rassemble les types de tâches de résolution de problème ainsi que les méthodes qui sont des manières de décomposer ces tâches en sous-tâches ou bien de les exécuter. Il nous a semblé plus naturel et plus intéressant de considérer une tâche de résolution de problème comme un processus plutôt que seulement comme une sorte de but à atteindre comme c'est souvent le cas dans les méthodologies d'acquisition de connaissances. En effet, tout comme un processus, une tâche de résolution de problèmes a des entrées-sorties et des méthodes pour la réaliser. Nous donnons dans la section 5.5 à la page 142 une classification de types de tâches de résolution de problèmes. Le type Problem_solving_method est également sous-type de Method, sous-type de Process_playing_a_role. Sous Process_playing_a_role, nous avons également introduit le type Process_contextualizing_its_object qui regroupe des types comme Think et Believe (cf. début de la section 4.3.2 et section 4.3.2.5).

Le type W_phenomenon/n rassemble aussi bien des états que des processus.

1. Nous parlons ici de manière générale et non par rapport à CGKAT qui ne contient encore aucun mécanisme de règles. Des procédures (ou fonctions) peuvent néanmoins être représentées avec le langage de commandes de CGKAT qui exploite le shell (l'interpréteur standard de commandes d'Unix).

5.4.4 Les choses jouant des rôles

Différents types de choses (entités ou situations) peuvent jouer des rôles similaires, et donc avoir des relations similaires avec d'autres choses (nous ne considérons pas la réciproque comme vraie : ce n'est pas parce que différentes choses ont des relations similaires avec d'autres choses, qu'elles jouent des rôles similaires). Pour préciser ce que nous signifions par "chose jouant un rôle" et donc éviter que toutes les catégories qui abstraient des relations entre un type de chose et d'autres types de choses, ne soient considérées comme des types de "choses jouant un rôle", voici quelques contraintes que nous rajoutons ; appelons R-type un type de choses jouant un rôle, et N-type un type considéré comme "naturel" (i.e. comme n'étant pas un R-type) :
- tous les sous-types d'un R-type doivent être des R-types ;
- un R-type a au plus un supertype direct qui est un N-type ;
- un R-type ne peut avoir de lien d'exclusion avec les N-types qui ont le même supertype direct que lui ;
- les R-types qui ne sont pas (entre autres) des sous-types directs de N-types doivent être des exceptions dans l'ontologie, de même que les R-types qui ont plusieurs R-types comme supertypes directs.

Ces règles ne permettent pas toujours de savoir lorsqu'un type doit être classé comme un N-type ou un R-type, mais elles donnent des indications. Ainsi, le type Etre_humain, sous-type de Animal, supertype de Homme et de Femme, et exclusif avec d'autres types de primates, semble clairement ne pas être un R-type tandis que les types Conducteur, Personne_mariée et Conducteur_marié peuvent être classés comme des R-types.

Voici dans notre ontologie de types de concepts, les catégories de plus haut niveau pour les R-types.

```
Concept_playing_a_role
    %Entity_playing_a_role -- e.g. Recipient_entity, Possessed_entity, Part_entity, Material_entity
    %Situation_playing_a_role -- e.g. Process_playing_a_role, W_result_outcome
    Concept_being_a_mediation -- a mediating object or circumstance for relating other things
```

Figure 5.17: *Les premiers sous-types de Concept_playing_a_role dans l'ontologie par défaut de CGKAT.*

Nous avons noté que les "médiations" pouvaient être considérées comme des choses jouant un rôle particulier (celui de médiation). Nous offrons avec le type Concept_being_a_mediation un point d'entrée à l'utilisateur au cas où il souhaiterait utiliser cette notion.

WordNet ne distingue pas les R-types des N-types. Dans notre classification des entités ou des situations qui jouent des rôles, nous avons seulement cherché à organiser les types de haut niveau de WordNet qui représentent des choses jouant un rôle simple, c'est-à-dire représentant des relations simples (e.g. Agent, Recipient, Object, Material, Part, Consequence, Method) entre ces choses et d'autres choses. Ces relations simples sont proposées par notre ontologie de types de relations et la plupart joignent un concept de type Process à un concept de type Entity ou plus rarement Situation. C'est à l'utilisateur de compléter s'il le désire cette classification avec les types de WordNet de plus bas niveau qu'il utilise ou avec les types liés à son application.

5.4.4.1 Les entités jouant un rôle

La figure suivante montre les premiers sous-types que nous avons trouvés pour Entity_playing_a_role. Nous discuterons seulement de W_causal_agent__cause__causal_agency. Ce type représente bien une entité qui joue un rôle (le fait d'être la cause d'un processus ou donc d'un état). Dans WordNet, ce type a de nombreux sous-types directs, e.g. W_agent/n et W_person__individual__someone__mortal__human__soul (les sous-types de ce type représentent des personnes jouant un rôle, e.g. W_consumer/n, W_contestant/n). Nous avons complété la classification des agents en introduisant les distinctions entre 1) les agents ayant un but et ceux qui n'en ont pas ou peuvent ne pas en avoir, 2) les agents cognitifs / non cognitif, et 3) les agents cognitifs conscients / non conscients. Une personne est un agent cognitif conscient et donc agissant avec des buts. Parmi les sous-types directs de W_person__individual__someone__mortal__human__soul, on peut trouver les types W_expert/n1 et le type W_engineer__applied_scientist__technologist qui est sous-typé par des types comme W_aerospace_engineer/n et W_electrical_engineer/n (nous avons rajouté le type Knowledge_engineer). Sous W_person__individual__someone__mortal__human__soul, nous avons également inséré un supertype pour tous les utilisateurs (ou groupes d'utilisateurs) : AnyUser. Ce type et ses sous-types peuvent par exemple être utilisés dans des méta-informations.

```
^Entity -- something that can be involved in a situation (ex: asserting the existence of an entity is describing a situation)
^Concept_playing_a_role
    %Entity_playing_a_role -- e.g. Recipient_entity, Possessed_entity, Part_entity, Material_entity
        Causal_entity
            W_causal_agent__cause__causal_agency -- any entity that causes events to happen
            Goal_directed_agent -- goal directed causal entity (ex:a problem solver or an interactional agent)
                Cognitive_agent -- for example an animal or an AI-agent
                    %Conscious_agent -- for example a person
                        W_person__individual__someone__mortal__human__soul -- a human being
                    .Non_conscious_cognitive_agent -- e.g. AI_Agent
            .Perhaps_goal_directed_causal_entity -- e.g. supernatural forces
            .Without_goal_causal_entity -- non conscious Entity and not an AI_Agent
        W_necessity__essential__requirement__requisite__need -- anything indispensible that is needed
        W_inessential/n -- anything that is not essential
        Recipient_entity
            W_recipient__receiver -- a person who gets something
        Possessed_entity
            W_possession/n -- anything owned or possessed
        Material_entity
            W_substance__matter -- that which has mass and occupies space")
        Part_entity
            W_part__portion -- something determined in relation to something that includes it
            W_part__piece -- a portion of a natural object
            W_unit__building_block -- a single undivided natural entity occurring in the composition of something else
        Whole_entity
            W_whole__whole_thing__unit -- an assemblage of parts that is regarded as a single entity
        W_subject__content__depicted_object -- something selected by an artist for graphic representation
        W_variable/n -- something that is likely to vary, something that is subject to variation
        W_anticipation/n -- some early entity whose type or style anticipates a later one
        W_thing/n1 -- an entity that is not named specifically
```

Figure 5.18: *Des supertypes et des sous-types pour Entity_playing_a_role.*

5.4.4.2 Les situations jouant un rôle

Nous n'avons trouvé dans WordNet que deux types de haut niveau relatifs à des états ou des processus jouant des rôles.

```
^Situation -- something that "occurs", or that is imagined, in a region of time and space (it can be described by propositions)
^Concept_playing_a_role
    %Situation_playing_a_role -- e.g. Process_playing_a_role, W_result_outcome
        W_consequence__effect__outcome__result__upshot -- a phenomenon that follows and is caused by some previous phenomenon
        W_result_outcome -- the situation that exists when something ends
```

Figure 5.19: *Des supertypes et des sous-types pour Situation_playing_a_role.*

5.4.5 Les concepts utilisés par des agents

Dans un but d'indexation des types de concepts et non pour leur conceptualisation, il nous a semblé intéressant d'introduire une dernière notion qui est orthogonale aux précédentes : Concept_used_by_an_agent. En effet, ce type n'est pas destiné à permettre la description de concepts (leur conceptualisation) mais à permettre de rassembler via des liens de spécialisation, des types de concepts qui sont manipulés (créés ou utilisés) par tel ou tel agent (cogniticien, agent artificiel, etc.).

Des méta-informations sur les types peuvent également être utilisées pour cela, mais nous voulions offrir un *point d'entrée* pour l'indexation via les liens de spécialisation. Le type Concept_used_by_an_agent (voir figure ci-dessous) ne pouvait être classé parmi les types de concepts jouant un rôle puisque ses sous-types peuvent être aussi bien des R_types que des N-types. Notons qu'au lieu de ce type, un type du second ordre tel que Concept_type_used_by_an_agent pourrait être exploité de manière similaire : il ne s'agirait alors plus de relation de spécialisation qui servirait à rassembler des types du premier ordre, mais de relation d'instanciation. Mais CGKAT ne permet pas l'exploitation de type du second ordre.

Concept_used_by_an_agent -- *a concept used by an agent or a group of agents (expert(s), etc.)*
  Concept_used_by_a_knowledge_engineer -- *e.g. Concept_used_by_a_KADS_knowledge_engineer*
  Concept_created_by_an_agent -- *may be useful if many users work on the same ontology*

Figure 5.20: *Les premiers sous-types de Concept_used_by_an_agent dans l'ontologie de CGKAT.*

Nous avons rassemblé sous le type Concept_used_by_a_KADS_knowledge_engineer, les types de concepts relatifs à KADS.

5.4.6 Bilan de la réorganisation des catégories de haut niveau de WordNet

Nous avons organisé et parfois réorganisé les catégories de haut niveau de WordNet dans une ontologie de haut niveau de manière à ce qu'elles soient plus utiles pour guider et vérifier la représentation des connaissances. Nous avons regroupé les catégories similaires, représenté des liens d'exclusion entre les catégories, ainsi que des relations entre les instances de différentes catégories (e.g. relations de description, de support de description, de mesure). Nous avons montré comment les liens d'exclusion et les signatures de relation évitent les confusions entre des catégories relatives à des entités, des états ou des processus. Il a été particulièrement délicat de trouver une organisation satisfaisante pour permettre d'utiliser et de comparer simplement différents types d'entités relatives au temps et à l'espace. De manière générale, la construction de notre ontologie de types de relations a posé de fortes (et bénéfiques) contraintes sur la construction de notre ontologie de types de concepts.

L'annexe 3 montre les dix catégories de plus haut niveau de WordNet et leurs sous-types directs. Nous avons réparti les six sous-types directs de W_abstraction/n dans diverses catégories plus adéquates. De même, nous avons réparti les sous-types de W_entity/n : trois sous Physical_entity et les dix autres sous Entity_playing_a_role. Nous avons également donné à d'autres types de haut niveau de WordNet une place nous semblant plus adéquate dans notre ontologie : W_representation/n1, W_communication/n, W_property__attribute__dimension, W_substance__matter, W_part__portion, W_method/n, W_result__outcome et W_consequence__effect__outcome__result__upshot. Au total nous avons donc organisé ou réorganisé 35 catégories de WordNet (10 - 2 + 6 + 13 + 8). Nous aurions pu ajouter à notre ontologie d'autres types afin de mieux structurer WordNet, mais les catégories de WordNet concernées (celles auxquelles il aurait fallu rajouter des supertypes) sont trop nombreuses et trop profondes dans la hiérarchie de WordNet pour que nous puissions intégrer ces catégories et leurs supertypes dans notre ontologie de haut niveau.

Nous avons complété les types de WordNet par d'autres types. Actuellement notre ontologie de types de concepts compte 259 types dont 68 types relatifs à des tâches ou des modèles de KADS.

5.5 Les tâches de résolution de problèmes et les modèles de tâches génériques

Comme nous l'avons précédemment souligné, les modèles que les méthodologies et outils d'AC proposent au cogniticien de construire sont des entités de représentation et nous avons ainsi classé les types de modèles que propose de construire la méthodologie KADS (Breuker & al., 1987) (Breuker & van de Velde, 1994).

Pour guider la construction de modèles d'expertise, KADS propose des modèles de tâches génériques (de résolution de problèmes). Ces modèles sont à adapter, combiner et "instancier" (connecter) avec des connaissances du domaine, pour constituer un modèle d'expertise. Ces modèles s'intègrent dans notre ontologie sous la forme de schémas prototypiques associés à des types de tâches. La hiérarchie des types de tâches permet l'accès aux modèles et peut être utilisée de manière similaire à un arbre de décision (tout comme les hiérarchies de types de problèmes de KADS). Un type de tâche de haut niveau (e.g. Diagnosis) représente une tâche générale dont le schéma est peu élaboré. Le choix de méthodes de résolution de problèmes permet de spécialiser une tâche générique. Ainsi, les types de tâches de plus bas niveau (e.g. KADS1_Systematic_Diagnosis) représentent des tâches plus spécialisées et ont des modèles plus élaborés.

KADS catégorise les tâches de résolution de problème en trois types :
1) les inférences qui sont des tâches primitives,
2) les tâches génériques de résolution de problèmes, et
3) les tâches réelles, celles que doit représenter un cogniticien et qui peuvent être modélisées, en théorie du moins, par adaptation et combinaison des tâches génériques.
Nous avons repris ces trois distinctions dans notre ontologie : nous avons sous-typer le type Problem_solving_task avec les types KADS_Inference_PST, KADS_Generic_PST et Real_life_PST. La figure suivante (5.21) montre ces types et leurs sous-types que nous détaillons maintenant.

- Les différentes tâches qu'un cogniticien réalise pour construire un SBC sont des exemples de tâches réelles. Nous avons classé dans notre ontologie les types de tâches que KADS conseille de réaliser, ainsi que le réseau de dépendance de données de ces tâches grâce à un schéma prototypique associé au type Knowledge_engineering_with_KADS.

- On peut trouver dans (Breuker & van de Velde, 1994) diverses classifications des inférences de KADS : suivant le type de leurs entrées-sorties, suivant les objets qu'elles créent ou détruisent, suivant leur fonction. C'est cette dernière classification qui guide le plus le cogniticien dans le choix de ses inférences. Aussi dans notre synthèse (voir figure 5.22), nous avons complété cette classification et nous avons intégré les types importants des autres classifications.

- Les types de tâches génériques de KADS-1 (Breuker & al., 1987) étaient organisés dans une taxinomie (cf. figure 2.7). Cette taxinomie a été réactualisée par Aamodt & al. (1992). C'est cette taxinomie que nous avons reprise pour hiérarchiser les types de tâches de CommonKADS, en intégrant les deux nouveaux types de tâches définies dans (Breuker & van de Velde, 1994) : KADS_Reconstruction_PST et KADS_Modelling_PST. Breuker & van de Velde (1994) ne donne pas de taxinomie pour les tâches génériques de CommonKADS, mais un réseau de dépendance de données (cf. figure 2.8). Nous avons représenté ce réseau de dépendance de données grâce à des schémas prototypiques associés aux types de tâches concernées. Par exemple :
  TC for KADS_Design_PST (x) are [KADS_Design_PST : *x]-
  { (Succ)←[KADS_Modelling_PST];
  (Succ)→[KADS_Assignment_PST];
  }.

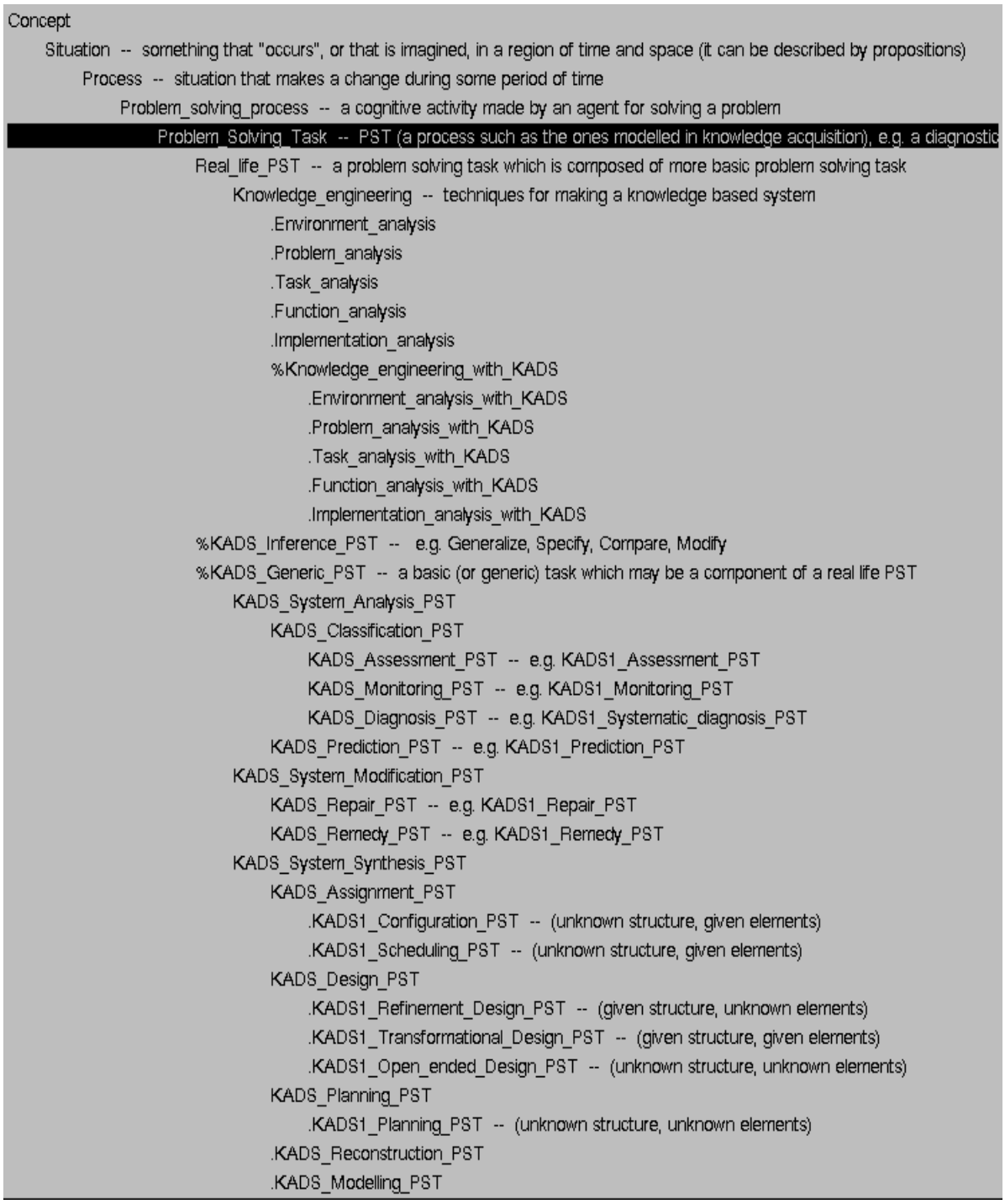


Figure 5.21: *Des supertypes et des sous-types pour Problem_Solving_Task.*

```
^Problem_Solving_Task -- PST (a process such as the ones modelled in knowledge acquisition), e.g. a diagnostic
^Concept_used_by_a_KADS_knowledge_engineer
    %KADS_Inference_PST -- e.g. Generalize, Specify, Compare, Modify
        .Collect
        Generalize -- find a superclass/superset for one or several instances/facts or classes
            Classify -- (the class hierarchy is given)
                .Abstract_class -- class->superclass
                .Identify -- instance->class
            .Cluster -- (here classes are learnt from examples)
        Specialize -- find a subclass/subset or an instance/fact
            .Refine -- class->subclass
            .Instanciate -- class->subclass/instance
            .%Select -- set of instance->subset of instance
        Compare -- in the sense given by KADS
            .Compare_Value
            .Match -- compare structures
        Modify -- in the sense given by KADS
            .Assign -- a value to an attribute of something
            .Evaluate -- produces a concept regarding the structure of something
            .Transform -- e.g. sort, restructure
            .Assemble
            .Decompose
        Determine_truth_or_relevance -- e.g. Establish (Cover, Verify)
        Explore_option -- e.g. Make_decision (Select), Propose_solution (Generate)
        Reduce_working_set -- e.g. Anticipate, Prefer, Rule_out
```

Figure 5.22: *Des supertypes et des sous-types pour KADS_Primitive_Task.*

Voici également à titre d'exemple une représentation du modèle générique de la tâche de diagnostic (celle de plus haut niveau) dans CommonKADS (il ne s'agit à ce niveau que de la description des entrées-sorties de la tâche) :

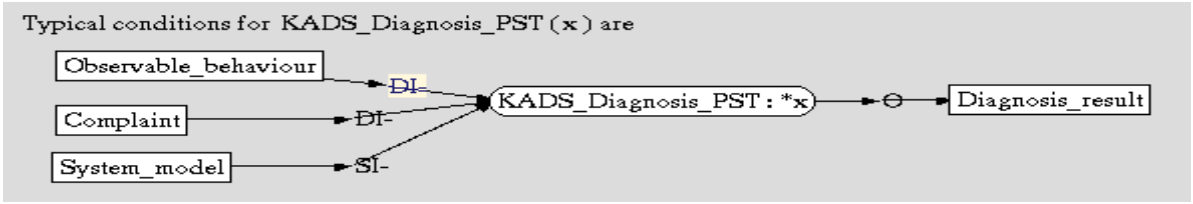


Figure 5.23: *Représentation des entrées-sorties d'un modèle de tâche générique dans CGKAT.*

Cette figure montre la représentation graphique avec laquelle nous avons représenté ce modèle générique dans CGKAT. *Nous avons essayé de nous rapprocher le plus possible de la représentation graphique donnée dans KADS pour représenter les tâches ou inférences et leurs entrées-sorties, afin qu'un cogniticien KADS les comprenne facilement*. C'est pourquoi 1) la boîte du concept [KADS_Diagnosis_PST : *x] a des bords arrondis (ce choix s'effectue en modifiant un attribut graphique et n'a aucune influence sur le contenu de la base de GCs), et 2) les relations connectées à ce concept sont ainsi orientées (de cette manière, la seule différence au niveau graphique avec les structures d'inférence de KADS est que celles-ci ne montrent pas les types des relations mais utilisent une double flèche au lieu d'une simple flèche pour les "entrées statiques"). Cette orientation des relations nous a obligé à donner des noms surprenants à deux types des relations, DI- et SI-, respectivement relations inverses de DI (dynamic input) et SI (static input)[1]. L'usage dans les GCs aurait été d'utiliser les relations DI et SI. Rappelons que le GC "[B : b]←(Entrée)←[A : a]" se lit "il existe des individus a et b respectivement de type A et B et l'Entrée de a est b".

1. Ces relations indiquent si le rôle en entrée d'une inférence est statique ou bien dynamique (i.e. en sortie d'une autre inférence). Cette caractéristique n'est pas constante pour chaque type de rôle, elle dépend des inférences. Pour plus de détails, voir CommonKADS (Breuker & van de Velde, 1994).

Nous avons donné dans la figure 2.5, un modèle plus élaboré, la structure d'inférences de la tâche de diagnostic systématique ; en voici une représentation dans le formalisme des GCs.

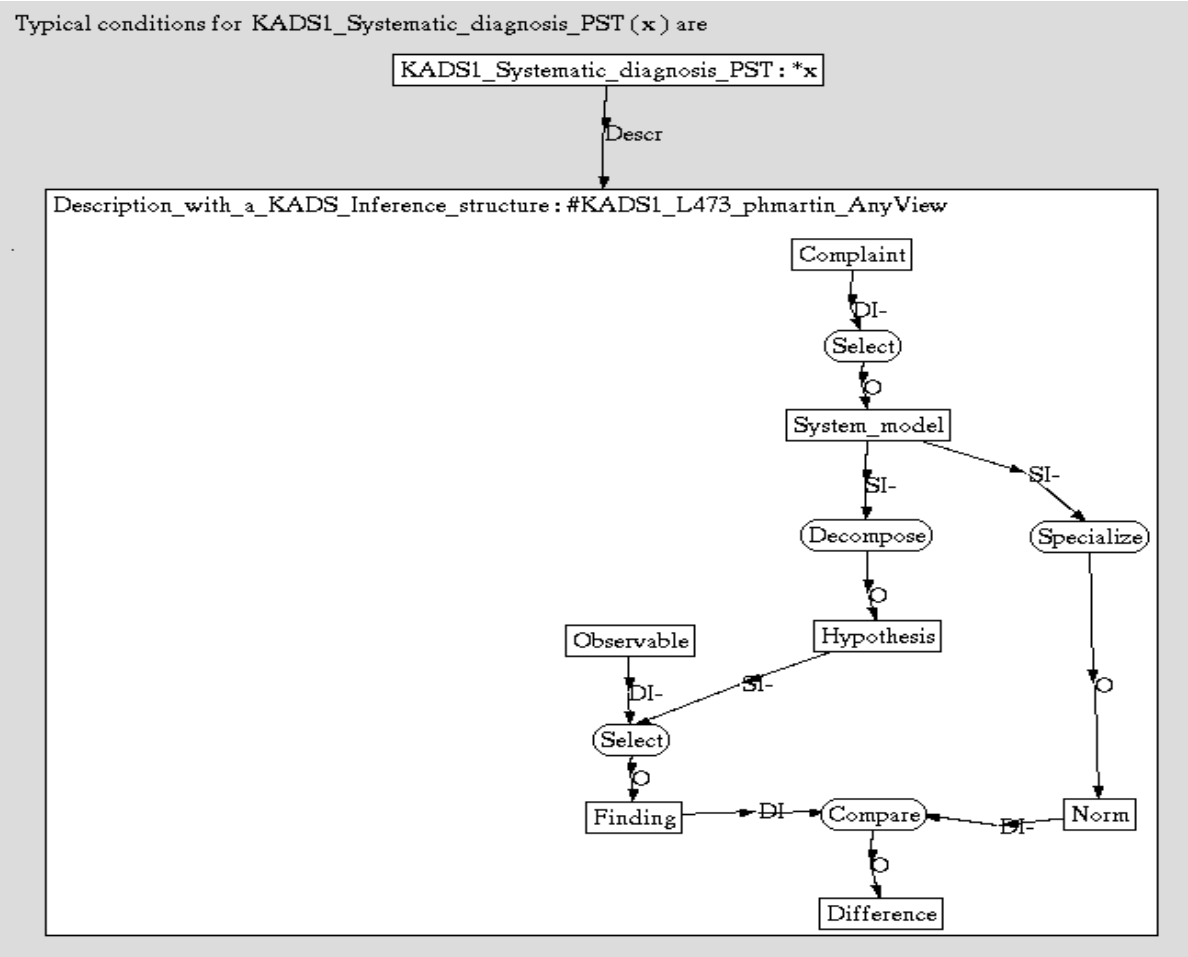


Figure 5.24: *Une représentation d'une structure d'inférences pour la tâche de diagnostic systématique.*

KADS1_Systematic_Diagnosis_PST est sous-type de Situation. Conformément à notre ontologie, nous utilisons une relation de type Descr pour relier un concept représentant une telle situation, à une de ses descriptions. Cette description est représentée avec un concept individuel d'un type sous-type de Proposition (Description_with_a_KADS_Inference_structure) et qui comporte un GC dans sa partie réfèrent (une représentation d'une structure d'inférence). Nous avons ainsi représenté dans le formalisme des GCs et de manière explicite, tous les liens entre un GC représentant une structure d'inférences et les tâches typiquement décrites par cette structure.

La construction de structures d'inférence dans le formalisme des GCs n'est pas une idée nouvelle mais ce sont les moyens de les rendre exécutables qui ont surtout été étudiés : nous avons noté en section 4.4.3 que Lukose (1995) et Moeller (1995) préconisent pour cela d'utiliser des acteurs. A notre connaissance, les structures d'inférences des tâches génériques de KADS n'avaient pas encore été représentées dans le formalisme des GCs. Pourtant, ainsi représentées dans CGKAT, des structures d'inférences peuvent être facilement comparées et recherchées (usage de la projection), combinées (usage de joints dirigés ou de joints maximaux) et adaptées (opérations de spécialisation ou de généralisation) dans le respect des contraintes liées aux types manipulés (e.g. les signatures des relations et les relations de conformité). Nous détaillerons au cours du prochain chapitre et dans l'annexe 5, le langage de commandes de CGKAT qui permet d'effectuer ces opérations de recherche, de comparaison et de combinaison. Les commandes de jointure dirigée et de jointure maximale se rapprochent des "opérateurs de combinaison de structures d'inférences" préconisés par CommonKADS (Breuker & van de Velde, 1994).
En phase de modélisation ascendante, des inférences et/ou des rôles peuvent être *recherchés* parmi

les GCs déjà représentés afin de pouvoir les *assembler* suivant les modèles donnés par des structures d'inférences. La modélisation descendante est plus basée sur les types d'éléments de domaine que peuvent exploiter les inférences et qui donc peuvent être connectés à leur rôles. Cela se trouve dans une description de chaque inférence d'une structure d'inférences. Lorsque des types d'éléments du domaine sont connus, les concepts qui utilisent ces types peuvent être *recherchés* dans une base de GCs en utilisant la projection.

Nous montrons dans le prochain chapitre comment CGKAT permet de stocker, documenter et organiser des GCs ou des définitions de types, à l'intérieur de documents structurés. La figure 6.5 montre le début d'un document représentant et documentant les modèles que la méthodologie KADS-I conseille de construire ainsi que les modèles génériques qu'elle propose pour aider à construire un modèle d'expertise. L'accès par l'utilisateur de CGKAT aux modèles contenus dans ce document peut se faire par requête (projection) ou par navigation hypertexte dans le document (e.g. navigation depuis la table des matières, navigation séquentielle de section en section, navigation sur les références croisées).

En plus des modèles de KADS-I, nous avons débuté la représentation et l'organisation dans des documents structurés
1) de modèles de CommonKADS (modèles à construire et modèles génériques),
2) de modèles que nous proposons pour faciliter le recueil et la modélisation des expertises liées aux explications (Martin, 1993a, 1993b, 1994), et
3) de définitions associées aux types de concepts de haut niveau de l'ontologie proposée par défaut par CGKAT.
Les figures 5.23, 5.24, 5.34 à 5.37 sont issues de ces documents.

5.6 Une ontologie de haut niveau pour les types de relations

5.6.1 Intérêt d'une ontologie structurée de types de relations

De même que pour les types de concepts, de nombreux travaux dans divers domaines proposent ou exploitent chacun un nombre assez limité de types de relations :

- pour permettre à un outil de générer un texte bien articulé : les outils de génération d'explications utilisent quelques relations rhétoriques et/ou d'argumentation (e.g. (Suthers, 1993)) ;
- pour permettre et guider les utilisateurs d'un système hypertexte dans la création et l'organisation des éléments de documents : de nombreux types de relations rhétorique et/ou d'argumentation devraient être proposés par les systèmes hypertextes (car de nombreux types de relations sont utiles pour organiser un document) mais actuellement seuls quelques types de relations d'argumentation sont parfois fournies (e.g. par gIBIS (Conklin & Begeman, 1988), SEPIA (Streitz & al, 1992) et AAA (Schuler & Smith, 1990)) ;
- pour permettre ou guider la représentation de connaissances par un outil ou un cogniticien : les outils ou méthodologies d'AC ne proposent que quelques relations sémantiques et/ou quelques relations d'ordonnancement de processus (e.g. KOD, MACAO, KADS, 3DKAT, KATEMES) ;
- pour guider un cogniticien dans la recherche ou l'extraction de connaissances à partir de sources d'expertise (documents ou experts) : tous les types de relations cités ci-dessus peuvent guider la recherche, l'extraction et la représentation de connaissances pour la résolution de problèmes et la production d'explications.

En résumé, divers travaux montrent que les types de relations qu'ils exploitent sont intéressants, mais chacun d'eux n'exploite qu'un nombre assez limité de types de relations. Il nous semble pourtant important qu'un outil d'AC,
1) offre une synthèse de tous ces types de relations puisque tous sont utiles en AC (i.e. en extraction, représentation et organisation de connaissances), et
2) permette à l'utilisateur de spécialiser et de compléter ces relations pour son application (cette possibilité est rarement offerte dans les systèmes d'AC ou hypertextes qui proposent des types de relations car ces types sont alors codés en dur dans ces systèmes et non offerts sous la forme d'ontologies éditables par les utilisateurs).

Afin d'en permettre un usage efficace par l'homme ou la machine, la synthèse de tous ces types doit être structurée. Si une ontologie *structurée* propose de *nombreux* types de relations,
1) l'utilisateur peut faire un choix adéquat en fonction de ses besoins (il n'y a pas de "bonnes" sélections de types utiles *a priori*),
2) l'utilisateur n'a pas à les inventer ; il réutilise la même ontologie que d'autres utilisateurs, ce qui facilite la comparaison ou la réutilisation des connaissances modélisées par les différents utilisateurs),
3) l'ensemble des types étant organisés, ils peuvent être retrouvés rapidement et être comparés entre eux (de même donc que les connaissances qui les utilisent) ; si l'ontologie proposée ne contient que peu de types de relations, le travail de complétion et de structuration est à la charge de chaque utilisateur.

Les types de relations peuvent être structurés de nombreuses façons : suivant les types d'états ou de processus qu'ils représentent, suivant les types des paramètres des relations, suivant certaines de leurs propriétés (e.g. symétrie, transitivité), etc. Un problème difficile est de trouver des catégories de haut niveau qui permettent à un utilisateur de trouver facilement les types de relations qu'ils recherchent.

Le modèle de base des GCs a maintenant été étendu pour prendre en compte un ordre partiel sur l'ensemble des types des relations (Mugnier & Chein, 1996). Vilnat (1992) note l'importance d'une

structuration sur les types de relations pour une analyse efficace de la langue naturelle, c'est-à-dire même lorsque les relations traitées ne sont généralement que des relations casuelles[1], et propose une structuration de ces relations casuelles (elle en donne 42 ; cf. annexe 3). Dans le cadre des travaux sur la gestion des relations dans le formalisme des GCs, citons Esch (1994a) et Ribière & al. (1996).

5.6.2 Classifications de types de relations primitives

Il existe peu de classifications de types de relations dans la littérature. Celles que nous avons trouvées concernent essentiellement des relations que nous appelons "primitives", car elles ne sont pas décomposables/définissables autrement qu'en utilisant une relation qui indique simplement qu'une connexion existent entre deux objets : nous notons "Relation" le type de cette relation de base. Par exemple, les relations de types Kind, Descr, Agent et Part sont des types de relations primitives.

NC for Agent (x, y) are [Process : *x]→(Relation)→[Goal_directed_causal_entity : *y].
//"NC" et non "NSC" car d'autres types de relations pourraient avoir la même défi nition

Par contre, un type de relation qui, comme dans la définition suivante, fait réfèrence à un processus n'est pas un type de relation primitive :

NSC for Résumer (x,y) are [Proposition : *x]←(Object)←[Résumer]→(Result)→[Proposition : *y].

Les relations casuelles[2] sont des exemples de relations primitives. Il est assez difficile de classer un type de relation comme le type Résumé qui permet d'exprimer un état (le résultat d'un processus de résumé) : il peut éventuellement être défini avec des relations plus primitives (e.g. Part) et dans ce cas ne pas être considéré comme primitif.

Sowa (1984) donne une liste non structurée de 37 relations primitives. Pour l'analyse du langage naturel et la recherche d'informations, Myaeng & Koo (1994) donne une liste assez proche de celle de Sowa (1984) un peu plus structurée de 74 relations primitives (voir annexe 3). Sowa (1992) donne quelques exemples de catégories de haut niveau pour les relations primitives : relations thématiques, relations spatiales, relations mathématiques (e.g. Measure, Less_than), relations d'attribut (e.g. Attr et Chrc), relations reliant un concept à un autre comportant un GC dans sa partie référent (e.g. Purpose, Method, Or) et "méta-relations" (e.g. Kind, Descr, Stmt). Quelques catégories de relations mais surtout des catégories de haut niveau ont été données dans les travaux sur les explications (Suthers, 1989), (Acker & al., 1994) : relations taxonomiques, attributives, comparatives, spatiales, temporelles, fonctionnelles, procédurales, causales et de perception. D'autres catégories de haut niveau pour les relations peuvent être trouvées dans CYC (Lenat & Guha, 1990a) et dans le Generalized Upper Model (Bateman & al, 1996) (voir annexe 3).

Enfin, nous avons trouvé des listes ou des hiérarchies de relations plus spécialisées, e.g. les relations rhétoriques de Mann & Thompson (1988), les relations d'argumentation de Schuler & Smith (1990), les relations temporelles d'Allen (1985) et les relations entre agents de Dieng (1991) et Labidi (1995).

---

1. Les relations casuelles de Vilnat (1992) proviennent des grammaires de cas de Fillmore (1968). Lorsqu'une phrase est représentée, les relations qui en sont extraites sont généralement des relations casuelles, mais les phrases elles-mêmes (ou parfois des portions de phrases) peuvent être reliées par des relations rhétoriques. Lorsque la représentation d'un domaine n'est pas effectuée à partir de phrases, des relations plus complexes et avec de nombreux arguments apparaissent naturellement (e.g. les relations en mathématiques ou dans les bases de données), cependant ces relations peuvent être définies (décomposées) avec des concepts et des relations plus primitives.

2. Notons que le terme de "relations casuelles" est parfois employé comme synonyme de "relation thématiques" alors que ce dernier terme ne désigne que les relations connectant un processus à ces participants (e.g. Agent, Instrument, Destinataire).

5.6.3 Les relations conceptuelles qui représentent des processus

Nous avons formalisé les types de processus comme des types de concepts et non comme des types de relations. Il est ainsi possible d'écrire les graphes suivants :
1a)[Eat]. //Il existe une instance du type du processus "Eat"
2a)[Person]←(Agent)←[Eat]. //Il existe une personne qui mange
3a)[Person: John]←(Agent)←[Eat]→(Object)→[Bread]. //John mange du pain.

Si des types de relations devaient être utilisés pour représenter dans le formalisme des GCs les trois faits ci-dessus, le premier fait ne pourrait être représenté et les deux autres nécessiteraient la déclaration de deux types de relations différents, par exemple :
2b)[Person]←(Agent_Eat). //Il existe une personne qui mange
3b)[Person: John]←(Agent_Eat_Object)→[Bread]. //John mange du pain.

Par ailleurs, pour que les représentations 2b) et 3b) aient une précision équivalente à 2a) et 3a), il faudrait arriver à définir les types Agent_Eat et Agent_Eat_Object par rapport à des types représentant respectivement (et uniquement) la notion d'agent, celle d'objet et celle de manger. Et ce travail serait à répéter à chaque introduction dans l'ontologie d'un nouveau type de relation relatif au processus de manger mais avec une "combinaison d'arguments" différente des autres types de relations relatifs au processus de manger, c'est-à-dire permettant de relier des concepts de types différents.

Ainsi, le fait que dans le formalisme des GCs une relation ne peut être connectée qu'à un concept et non à une relation, conduit souvent à représenter un grand nombre de type de processsus comme des types de concepts.

Cependant, des cogniticiens restent tentés d'utiliser des relations (et donc de définir des types de relation) pour représenter certains processus, comme dans les trois exemples suivants :

1) [Proposition : *x]→(Résumer)→[Proposition : *y].
2) [Proposition : *x]→(Résumé)→[Proposition : *y].
3) (Diviser)-
   { -1→[Number: *dividend];
   -2→[Number: *divisor];
   -3→[Number: *result];
   -4→[Number: *remainder];
   }

Si ces types de relations n'ont pas de définition, les relations de tels types ne peuvent être expansées et il est donc impossible de comparer les GCs qui utilisent ces types de relations avec des GCs qui utilisent des types de concepts représentant des processus. Les possibilités de recherche et d'inférences sont donc réduites.

Sinon, i.e. si ces types de relations ont des définitions utilisant des types de concepts représentant des processus (comme les types Résumé et Résumer), ces types de concepts sont en quelque sorte dupliqués dans l'ontologie des types de relations (et ce pour chaque "combinaison d'arguments"). De tels types de relations peuvent être utilisés pour raccourcir l'écriture de GCs mais, comme le montre la figure suivante (5.25), cela peut conduire à une représentation moins claire ou moins précise/ explicite que lorsque des types de concepts représentant des processus sont directement utilisés.

NCS for Conduire (x, y, z, t) are [Conduire]- //Accm : Accompaniment
{ (Agent)→[Personne : *x]→(Accm)→[Personne : *y];
(Source)→[Spatial_entity : *z];
(Destination)→[Spatial_entity : *t];
}.

1) L'usage d'un concept de type "Conduire" conduit à un GC plus clair que l'usage d'une relation de type "Conduire"

(Conduire)-
{ -1→[Personne : Jean];
-2→[Personne : Paul];
-3→[Ville : Antibes];
-4→[Ville : Nice];
}.

[Conduire]-
{ (Agent)→[Personne : Jean]→(Accm)→[Personne : Paul];
(Source)→[Ville : Antibes];
(Destination)→[Ville : Nice];
}.

2) Si le type de relation "Diviser" n'a pas de définition, l'usage d'un concept de type "Conduire" conduit à un GC plus précis (explicite) que l'usage d'une relation de type "Conduire" (car dans ce dernier cas, les rôles que jouent les arguments de la relation n'apparaissent que dans les noms de variables et ne sont donc pas explicitement représentés)

(Diviser)-
{ -1→[Number: *dividend];
-2→[Number: *divisor];
-3→[Number: *result];
-4→[Number: *remainder];
}.

[Diviser]-
{ (Dividend)→[Number];
(Divisor)→[Number];
(Result)→[Number];
(Remainder)→[Number];
}.

Figure 5.25: *Représentations de processus avec des relations ou bien avec des concepts.*

Dans notre ontologie de types de relations, le type Relation_refering_to_a_process permet de regrouper les relations représentant des processus. Nous avons classé sous ce type, les types de relations entre agents de Dieng (1991) et Labidi (1995) car ces types de relations représentent différentes formes de coordination, d'influence, de communication ou encore de coopération entre agents.

5.6.4 Les relations non binaires

La figure 5.25 illustre également le fait que l'usage de relations non binaires peut conduire à des représentations moins claires ou moins précises/explicites que lorsque des relations binaires sont utilisées. Voici un autre exemple avec des types de relations qui ne représentent pas des processus :

[Temporal_entity : *x]→(T_Before)→[Temporal_entity : *y]→(T_Before)→[Temporal_entity : *z]
semble plus explicite que : (T_Between)- { -1→[Temporal_entity];
-2→[Temporal_entity];
-3→[Temporal_entity];
}.

Aussi, les relations généralement utilisées sont binaires, surtout si la modélisation s'effectue de manière ascendante à partir du langage naturel, car toutes les relations primitives que nous avons rencontrées sont binaires (relations modales unaires exceptées ; T_Between n'est pas un type de relation primitive).

Toutefois, des relations unaires ayant une définition peuvent être utilisées pour raccourcir l'écriture de GCs. Ainsi par exemple,

(Possible)←[Proposition : #234].
peut être l'abréviation de
[Proposition : #234]→(Modality)→[Modality]→(Measure)→[Possible].
si le type de relation Possible est ainsi défi ni :
NSC for Possible (x) are [Proposition : *x]→(Modality)→[Modality]→(Measure)→[Possible].
// avec : Possible < Logical_modality_measure

Des types de relations de même arité peuvent être organisés par des liens de spécialisation à condition que les types de arguments de ces relations soient eux-mêmes respectivement liés par des liens de spécialisation (Leclère, 1995) (Chein, 1996). Ainsi par exemple, un sous-type de SubProcess<Process,Process> peut être SubTask<Problem_Solving_Task,Problem_Solving_Task> (les "<...>" représentent les signatures des relations).

Il peut être tentant de réunir des types de relations d'arité différentes sous un même supertype, et ainsi avoir par exemple un seul supertype pour tous les types de relations modales, pour tous les types de relations spatiales ou encore pour tous les types de relations. Mais de tels supertypes devraient-ils avoir plus ou moins d'arguments que leurs sous-types ? Dans le premier cas, par exemple le supertype des types de relations pourrait être Relation<Concept> (un type de relation unaire donc). Cependant, il faudrait alors également permettre d'utiliser des relations avec plus d'arguments que ne le prescrit leur signature, puisque l'utilisateur peut par exemple vouloir avec le graphe requête suivant, rechercher toutes les entités physiques reliées par une relation : [Physical_entity]→(Relation)→[Physical_entity] . Une telle extension pose donc certains problèmes. Dans (Leclère, 1995) et (Chein, 1996), les types de relations d'arités différentes ne peuvent être reliés par des liens de spécialisation et n'ont donc pas de supertype commun.

Comme nous venons de voir que
1) les relations généralement utilisées en modélisation à partir du langage naturel sont binaires, et qu'il est alors souvent plus clair ou explicite d'utiliser des relations binaires, et
2) que des relations d'arité différentes sont incomparables,
CGKAT n'encourage actuellement pas la création de relations non binaires :
a) notre ontologie ne propose pas de supertype pour les relations non binaires, et
b) des relations non binaires ne peuvent actuellement pas être incluses dans des GCs construits via l'éditeur Thot (nous avons néanmoins fait en sorte qu'une telle extension puisse rapidement être implémentée).

```
Relation (Concept,Concept)
    Attributive_relation (Concept,Concept)  --  e.g. Chrc, Attr, Manner, Name, Role, Possession, Accompaniment
    Component_relation (Concept,Concept)  --  e.g. Part, Main_Part, Element, Subset, Subtask, FirstSubTask
    Constraint_or_measure_relation (Concept,Concept)  --  e.g. mathematical, spatial and temporal relations
    Relation_from_a_situation (Situation,Concept)  --  i.e. only from a state or a process
        %Constraint_or_measure_relation_from_a_situation (Concept,Concept)  --  i.e. only from a state or a process
            .Descr (Situation,Proposition)  --  for describing a situation
            %Temporal_relation_from_a_situation (Situation,Concept)
                Situation_temporal_location (Situation,Temporal_entity)  --  e.g. Point_in_time, Duration, T_Since, T_Until
                Temporal_relation_between_situations (Situation,Situation)  --  e.g. T_Succ, Task_Succ, Terminaison, Consequence
            %Spatial_relation_from_a_process (Process,Spatial_entity)  --  e.g. Source, Destination, Path
        Relation_from_a_process (Process,Concept)
            .%Purpose (Process,Situation)  --  a purpose of a process is also a reason to do this process
            %Recipient (Process,Goal_directed_agent)  --  e.g. Beneficiary
            .%Experiencer (Process,Goal_directed_agent)
            %Result (Process,Concept)
                .O (Process,Concept)  --  output only object
                %IO (Process,Concept)  --  input-output object, e.g. Object_to_modify, Object_to_mute, State
            Object (Process,Concept)  --  this case relation is also usually called Patient or Theme
                I (Process,Concept)  --  input only object (for example for evaluation or comparison processes)
                    .Material (Process,Concept)
                    .%Parameter (Process,Concept)
                    .SI (Process,Concept)  --  static input (e.g. for KADS tasks)
                    .DI (Process,Concept)  --  dynamic input (e.g. for KADS tasks)
                %IO (Process,Concept)  --  input-output object, e.g. Object_to_modify, Object_to_mute, State
            .%Initiator (Process,W_causal_agent__cause__causal_agency)
            .%Agent (Process,Goal_directed_agent)
            .Instrument (Process,Entity)
            %Manner (Process,Measure)  --  process attribute, e.g. Quickly
            .Method (Process,Process)
            %SubProcess (Process,Process)  --  e.g. SubTask, FirstSubTask, LastSubTask
            %Spatial_relation_from_a_process (Process,Spatial_entity)  --  e.g. Source, Destination, Path
    Relation_to_a_situation (Concept,Situation)  --  i.e. only to a state or a process, e.g. Relation_to_a_process
    Relation_from_a_proposition (Proposition,Concept)  --  e.g. Information_source, Author, Rhetorical_relation, Argumentation_relation
    Relation_refering_to_a_process (Concept,Concept)  --  e.g. Summarize, Change_location, Relation_between_agents
    Relation_with_a_special_property (Concept,Concept)  --  e.g. Transitive_relation, Symmetric_relation, Contextualizing_relation
    Relation_used_by_an_agent (Concept,Concept)  --  for accessing the relation types accepted/used by a (group of) agent(s)
        .Relation_created_by_an_agent (Concept,Concept)  --  may be useful when many knowledge engineers work on the same ontology
```

Figure 5.26: *Les premiers sous-types de Relation dans l'ontologie proposée par défaut par CGKAT.*

5.6.5 Des catégories de haut niveau pour les types de relations

Nous avons noté que les types de relations peuvent être structurés de nombreuses façons mais que la structuration choisie doit permettre aux utilisateurs de retrouver rapidement les types de relations qu'ils recherchent. La figure 5.5 montre comment sont visualisées les catégories de plus haut niveau dans le menu de gestion des types de relations, lorsque l'option "avec signatures" est choisie. Les types de relations connectables à un concept de processus sont tous visualisés, car ce seront probablement les plus utilisés : ils sont ainsi rapidement accessibles. Nous avons regroupé les types de relations suivant différents critères, ce qui permet à des types de relations d'être retrouvés par différents chemins (ainsi l'ontologie comporte différents points d'entrée).

- Classement sur la nature des relations : nous offrons des catégories regroupant respectivement des types de relations attributives, compositionnelles, rhétoriques, argumentatives, logiques, spatiales, temporelles et "référant à un processus" (e.g. Résumer). Les types de relations mathématiques sont répartis sous les types Constraint_or_Measure_relation (e.g. Equal, GreaterThan) et Relation_with_special_property (e.g. Transitive_relation).
- Classement sur la fonction des relations : le type Constraint_or_Measure_relation regroupe les types de relations qui peuvent être utilisées pour exprimer des comparaisons (ou des mesures) et/ou des restrictions, c'est-à-dire principalement les types de relations logiques, spatiales, temporelles et certains types de relations mathématiques ; par ailleurs, le type Relation_used_by_an_agent a une fonction similaire pour les types de relations à celle du type Concept_used_by_an_agent pour les types de concepts.
- Classement sur les types de paramètres des relations : pour ceci, les types de haut niveau sont Relation_from_a_situation<Situation,Concept>, Relation_from_a_process<Process,Concept>, Relation_to_a_situation<Concept,Situation>, Relation_to_a_process<Concept,Process>, Relation_from_a_collection<Collection,Concept> et Relation_from_a_proposition<Proposition, Concept>[1]. Les deux premiers types regroupent des relations casuelles ainsi que certaines relations temporelles et spatiales ; le dernier regroupe des types de relations rhétoriques, logiques et modales (au sens large).

Nous n'avons pas inclus dans notre ontologie des types de relations comme Kind et Subtype car leurs signatures nécessitent des types de concepts du second ordre. Nous avons actuellement introduit et organisé dans notre ontologie 213 types de relations dont la plupart sont primitives et utiles en représentation des connaissances. Grâce à ces types et à leur organisation, l'utilisateur est guidé 1) dans sa recherche de types de relations, 2) dans la création et l'organisation de ses types de relations, et 3) dans l'extraction et la représentation de connaissances.

Ce troisième point (i.e. l'aide à l'extraction et à la représentation de connaissances) est lié aux signatures des relations. En effet, d'une part, ces signatures assurent une certaine cohérence sémantique dans la construction de GCs, et conduisent donc le cogniticien à organiser ses types de concepts. D'autre part, dans CGKAT, le menu de gestion des types de relations peut afficher de manière hiérarchique les *types des relations qui peuvent être connectées à un concept sélectionné (ou à un concept d'un type de concept sélectionné)*[2]. Grâce à ce filtrage,
1) la création d'un GC est rendue plus aisée puisque le cogniticien peut sélectionner un des types de relations et demander la création et la connexion d'une relation de ce type au concept sélectionné (le second concept de la relation est créé automatiquement) ;
2) il s'agit d'une méthode de recherche/extraction systématique de connaissances liées à un concept d'un type donné. Nous détaillerons ce deuxième point dans la section 5.7.1 à la page 161.

1. Il ne nous a pas semblé utile d'ajouter les types Relation_to_a_proposition<Concept,Proposition> et Relation_to_a_collection<Concept,Collection>.

2. Il s'agit de l'affichage sous forme de liste indentée des sous-types de "Relation" mais seulement de ceux qui incluent le type de concept sélectionné dans leur signature.

Les types de concepts actuellement utilisés dans les signatures des types de relations de notre ontologie sont les suivants : Concept, Collection, Situation, Process, Problem_Solving_Task, Temporal_entity, Spatial_entity, Goal_directed_agent, Cognitive_agent, Proposition, Issue, Position, Argument, Fact, Representation_entity, Number, Integer, Property, Measure, Modality et Truthness.

Nous n'allons pas détailler ici chacune des grandes catégories de types de relations de notre ontologie de manière aussi détaillée que nous l'avons fait pour les types de concepts. Le lecteur peut trouver dans les commentaires associés aux types de relations leurs sous-types les plus typiques (voir figure 5.5). Voici néanmoins quelques précisions sur l'organisation de certaines catégories.

5.6.5.1 Les relations à partir d'un concept de type Situation

La figure 5.5 montre les premiers sous-types de Relation_from_a_situation. Ses sous-types directs sont Constraint_or_measure_relation_from_a_situation et Relation_from_a_process (ce dernier type pourrait plus explicitement être nommé Relation_only_from_a_process) ; nous n'avons pas trouvé de relation spécifique aux états.

Les ***relations à partir de processus*** peuvent être regroupées de diverses façons : certaines relient le processus à une de ses causes (entités ou situations), d'autres à un de ses effets, d'autres encore à un moyen de réaliser le processus. Comme les sous-types directs de Relation_from_a_Process ne sont pas très nombreux, nous n'avons pas ajouté de catégories intermédiaires pour les regrouper, mais nous en avons classé certains comme sous-types de Causation_relation et de Effect_relation. Ces types (sous-types de Constraint_or_measure_relation) permettent de regrouper des types de relations exprimant une notion de causalité ou d'effet dans des sens très larges (sens communs) ; leur intérêt est donc seulement d'être des points d'entrées supplémentaires facilitant les recherches de l'utilisateur dans l'ontologie. Parmi les types de relations à partir de processus, Causation_relation regroupe Purpose, Parameter, Initiator, Agent, tandis que Effect_relation regroupe Recipient, Perceiver, Result.

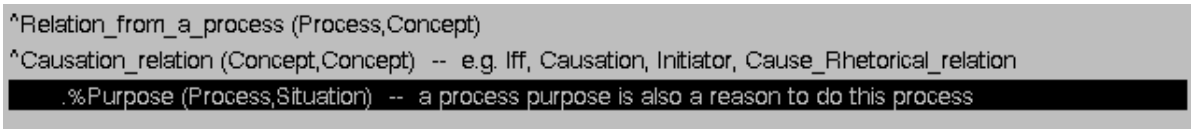


Figure 5.27: *Des supertypes pour le type de relation Purpose.*

Nous avons sous-typé les types assez généraux Result et Object avec des notions plus précises utilisées en informatique : I ("I" pour abrévier "input only object"), O ("O" pour "output only object") et IO ("IO" pour "input-output object"). Nous avons encore précisé cette dernière notion pour permettre de préciser si l'objet en entrée-sortie est modifié et si oui, s'il en est transformé.

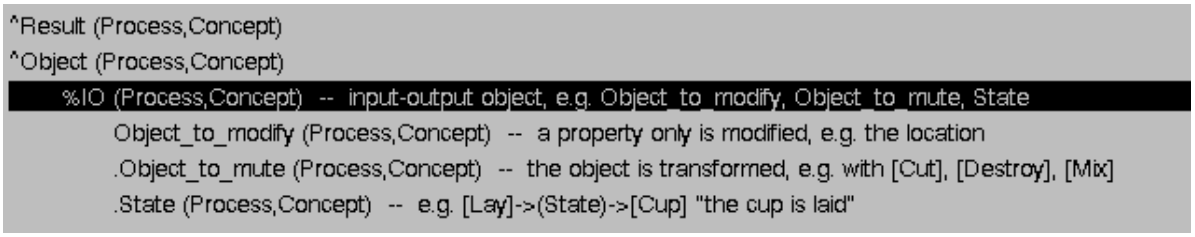


Figure 5.28: *Des supertypes pour le type de relation Purpose.*

Les ***relations temporelles et spatiales*** spécifiques aux situations et aux processus sont regroupées sous Temporal_relation_from_a_situation et Spatial_relation_from_a_process. Pour illustrer l'organisation des relations temporelles et spatiales, les deux figures suivantes montrent les super-types et sous-types directs de Temporal_relation et Spatial_relation.

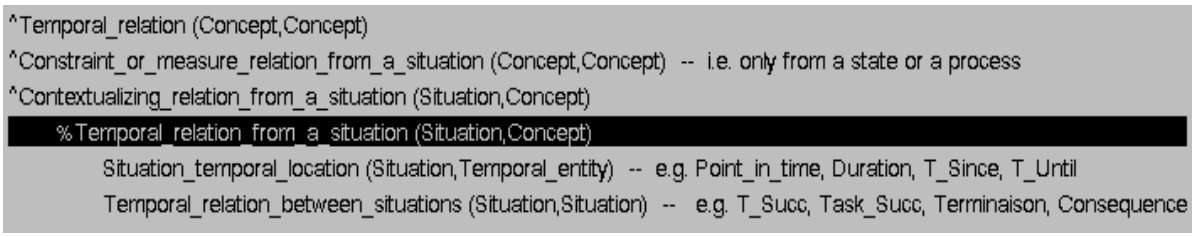


Figure 5.29: *Des supertypes et des sous-types pour Temporal_relation_from_a_Situation.*

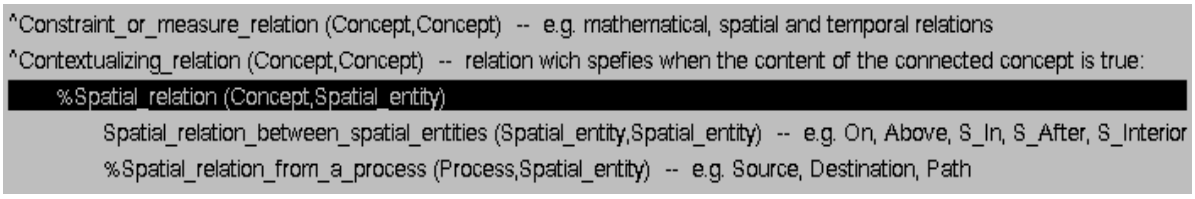


Figure 5.30: *Des supertypes et des sous-types pour Spatial_relation.*

Voici un exemple de l'usage de telles relations.
[Time : #yesterday]←(Point_in_time)←
[Situation]-
{ (Point_in_time)→[Temporal_entity : #18.05.96] ;
(Duration)→[Time_Period : #1hour] ;
(Descr)→[Proposition : #456 [Cat]→(On)→[Mat]] ;
}.

5.6.5.2 Les relations à partir d'un concept de type Proposition

Nous avons distingué trois catégories principales parmi les types de relations connectables à un concept de type Proposition : les relations contextualisantes à partir de processus, les relations rhétoriques et les relations d'argumentation. Nous avons également rajouté le type de relation Believer qui n'appartient à aucune des catégories précédentes.

```
Relation (Concept,Concept)
    Relation_from_a_proposition (Proposition,Concept) -- e.g. Information_source, Author, Rhetorical_relation, Argumentation_relation
        %Contextualizing_relation_from_a_proposition (Proposition,Concept) -- e.g. Author, Modality, Or, Iff, If_Then_relation
        .Believer (Proposition,Cognitive_agent)
        .Stmt (Proposition,Representation_entity) -- for stating a proposition
        Rhetorical_relation (Proposition,Proposition) -- from the Rhetorical Structure Theory (RST) and the PENMAN ontology
            Presentational_Rhetorical_relation (Proposition,Proposition) -- connect to details which should induce the reader to do or think something
                .RST_Enablement (Proposition,Proposition) -- details for enabling the reader to perform the described action
                .RST_Background (Proposition,Proposition) -- details for enabling the reader to understand the described situation
                .RST_Motivation (Proposition,Proposition) -- details for increasing the reader desire to perform the described action
                .RST_Evidence (Proposition,Proposition) -- details for increasing the reader belief
                .RST_Justify (Proposition,Proposition) -- details for increasing the reader acceptance
                .%RST_Antithesis (Proposition,Proposition) -- contrast for increasing the reader positive regard
                .RST_Consession (Proposition,Proposition) -- concession for increasing the reader positive regard
            Subject_matter_Rhetorical_relation (Proposition,Proposition) -- connect to details for making a better description
                .RST_Circumstance (Proposition,Proposition) -- circumstances about a situation
                .RST_Solution (Proposition,Proposition) -- solution to a problem
                %RST_Elaboration (Proposition,Proposition) -- e.g. RST_Subtype, RST_Property, RST_Part
                %Cause_Rhetorical_relation (Proposition,Proposition)
                    .RST_Volitional_Cause (Proposition,Proposition) -- cause of the described intended situation
                    .RST_NonVolitional_Cause (Proposition,Proposition) -- cause of the described not intended situation
                    .RST_Purpose (Proposition,Proposition) -- situation that the described action is intended to reach
                %Effect_Rhetorical_relation (Proposition,Proposition)
                    .RST_Volitional_result (Proposition,Proposition)
                    .RST_NonVolitional_result (Proposition,Proposition)
                .RST_Definition (Proposition,Proposition) -- logical relations should be used instead of this relation
                .RST_Comparison (Proposition,Proposition) -- use Similar_to and Different_from instead of this relation
                .RST_Means (Proposition,Proposition)
                .RST_Condition (Proposition,Proposition) -- situation necessary for the realization of the described situation
                .RST_Otherwise (Proposition,Proposition) -- situation which prevents the realization of the described situation
                RST_Interpretation (Proposition,Proposition) -- interpretation/abstraction/comment on the described facts
                    .RST_Evaluation (Proposition,Proposition) -- positive interpretation/abstraction/comment on facts
                .%RST_Restatement (Proposition,Proposition)
                .RST_Summary (Proposition,Proposition)
                .RST_Theme (Proposition,Proposition)
                %RST_Contrast (Proposition,Proposition) -- comparability and differences of two situations
                    .%RST_Antithesis (Proposition,Proposition) -- contrast for increasing the reader positive regard
            %Symmetric_Rhetorical_relation (Proposition,Proposition) -- e.g. RST_Restatement
        Argumentation_relation (Proposition,Proposition) -- from the AAA system (Schuler&Smith, 1990)
            .Serves (Issue,Issue) -- source issue serves dest. issue
            .Replacement (Issue,Issue) -- dest. issue is a replacement of source issue
            .Suggestion (Proposition,Issue) -- dest. issue is a suggested by the source proposition
            .Answer (Issue,Position) -- dest. position is an answer to the source issue
            .Objection (Position,Argument) -- dest. argument is an objection to the source position
            .Confirmation (Position,Argument) -- dest. argument is a support of the source position
            .Contribution (Proposition,Proposition)
            .Reference (Proposition,Fact) -- dest. fact is a reference of the source proposition
```

Figure 5.31: *Les sous-types de Relation_from_a_proposition dans l'ontologie de CGKAT.*

5.6.5.3 Les relations ayant une propriété spéciale

Le type Relation_with_a_special_property permet de réunir des types de relations ayant certaines caractéristiques comme le fait d'être binaires, transitives ou encore contextualisantes. La plupart des relations couramment utilisées étant asymétriques, il semble intéressant de considérer toutes les relations comme asymétriques à moins que leur type ne soit sous-type de Symmetric_relation.

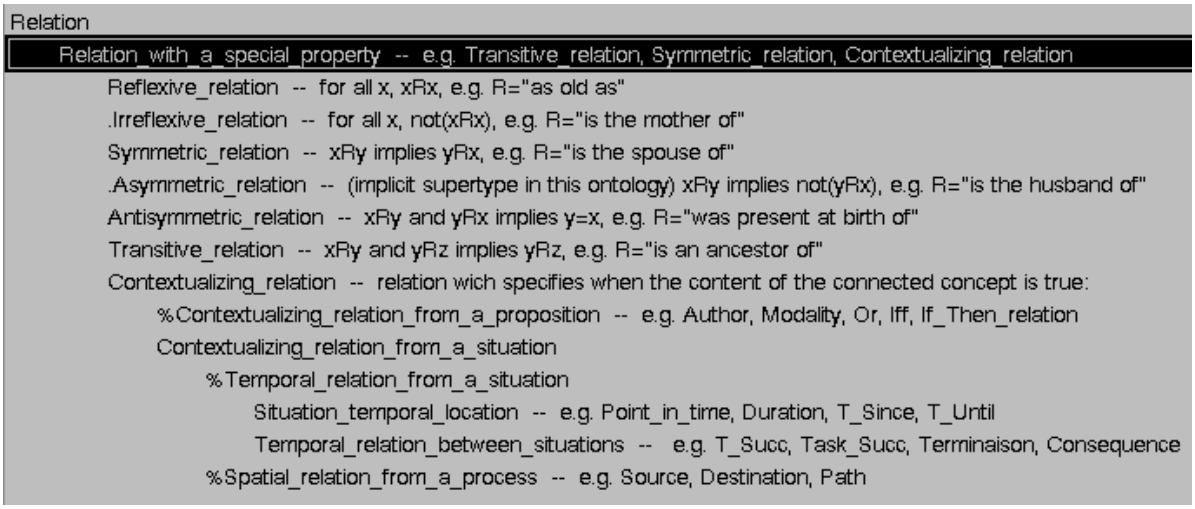


Figure 5.32: *Des sous-types pour le type de relation Relation_with_a_special_property.*

5.6.5.4 Les relations contextualisantes à partir d'un concept de type Proposition

Voici quelques relations que nous considérons comme contextualisantes. Rappelons qu'un Graphe Conceptuel Simple (GCS) qui utilise de telles relations ne correspond plus comme un GCS normal, à une formule de la logique du premier ordre, conjonctive, positive et fermée existentiellement. D'autres interprétations logiques sont à définir et donc aussi d'autres règles d'inférence que les opérations de généralisation sur ces GCSs. De même, les GCs emboîtés qui utilisent de telles relations n'ont pas la même interprétation logique que des GCs emboîtés n'utilisant pas de relations contextualisantes. Le type de relation Contextualizing_relation sert à regrouper et signaler explicitement les types de relations contextualisantes.

A chaque niveau (d'emboîtement) d'un GC emboîté, un GCS peut être associé (il suffit pour cela, dans chaque graphe correspondant à un niveau donné, d'ôter les référents graphes aux concepts qui en contiennent). Un GC emboîté peut donc être recherché via des recherches sur des GCSs, e.g. par projection. Comme nous l'avons souligné dans la section 4.3.2.4, si un GCS est résultat d'une requête et que ce GCS est emboîté dans un GC qui le contextualise, le GCS ne peut être affiché seul il doit être présenté à l'intérieur des niveaux qui le contextualisent dans le GC (ainsi ce GC peut ne pas être affiché en entier, mais les niveaux contextualisants le GCS doivent l'être)[1].
CGKAT considère qu'un GCS est contextualisé si
1) un des GCSs des niveaux emboîtants contient au moins une relation dont le type est sous-type de Contextualizing_relation, ou
2) si le type de l'un des concepts emboîtants est sous-type de Contextualizing_proposition, ou
3) si l'un des concepts emboîtants est lié par une relation de type Object (ou d'un de ses sous-types) à un concept de type Process_contextualizing_its_object (ou d'un de ses sous-types).

1. Notons que CGKAT n'interdit pas la création de différents GCs emboîtant un même GC (i.e. partageant une même sous-partie). Si cette sous-partie est contextualisée par plusieurs de ces GCs emboîtants, et doit être affichée parce qu'elle est le résultat d'une requête, chacun de ces GCs emboîtants est présenté par CGKAT (plus exactement, ces GCs emboîtants ne sont pas présentés en entier, mais pour chacun, les niveaux qui contextualisent la sous-partie résultat de la requête sont présentés avec (autour de) cette sous-partie.

Par exemple, Truthness étant un sous-type de Contextualizing_relation, si la base contient le GC suivant, ce GC sera affiché si les spécialisations du graphe "[Personne : Joe]→(Possession)→[Entity]" sont recherchées (bien que seule une partie de ce GC soit une "spécialisation" du graphe requête) :
[Truthness : #False]←(Thruthness)←[Proposition : #p24 [Personne : Joe]→(Possession)→[Voiture]].

Les relations spatiales et temporelles sont des relations contextualisantes qui sont liées à des situations. La figure suivante montre des types de relations contextualisantes liées à des propositions. L'utilisateur peut bien sûr modifier cette liste et ainsi par exemple, supprimer les types Author et Information_source de cette liste.

Comme notre ontologie ne contient pas de type de relation unaire, nous avons introduit le type Thruthness pour remplacer Not (cf. exemple précédent). Les types de relations logiques, And excepté, sont également des types de relations contextualisantes. Le type de relation If_Then_relation permet d'exprimer une règle. Par exemple :
[Proposition : #p1 [Cat]→(On)→[Mat]]→(If_Then_relation)→[Proposition : #p2 [Cat]→(Attr)→[Happy]].

Bien sûr, le mécanisme de règle exploitant de telles représentations reste à définir. CGKAT n'en comporte actuellement pas. Cependant un mécanisme de chaînage avant et arrière sur des règles de type "Si Alors" dont les prémisses et les conclusions sont des GCSs, est actuellement étudié dans l'équipe du LIRMM et sera à terme implémenté au dessus de CoGITo. Notons que l'utilisateur peut donner, s'il le désire, des sous-types communs à If_Then_relation et NecessaryCondition.

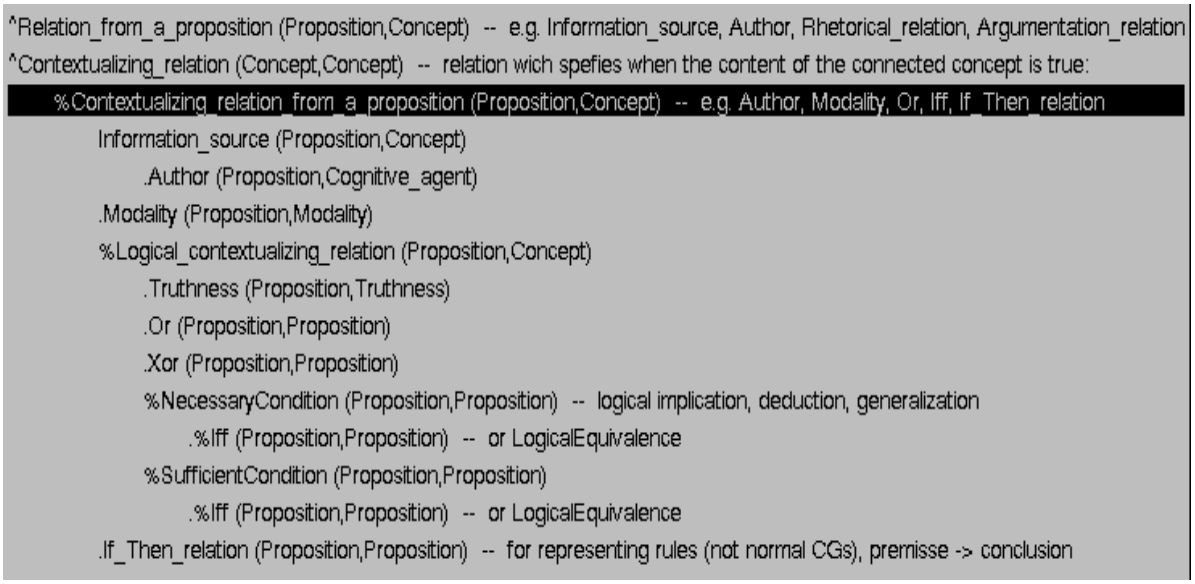


Figure 5.33: *Des sous-types pour le type de relation Contextualizing_relation_from_a_proposition.*

5.6.6 Nécessité de choix supplémentaires

L'ontologie proposée par CGKAT fournit un vocabulaire conceptuel qui guide l'extraction et la représentation de connaissances. Nous donnons ci-après l'exemple de quelques représentations d'une phrase qui peuvent être créées en utilisant notre ontologie. On peut remarquer avec cet exemple l'intérêt de disposer d'une ontologie de types de relations primitives pour savoir comment exprimer une connaissance. Le choix ou la création de types de concepts est plus intuitif (mais la difficulté se reporte sur l'organisation de ces types). Cet exemple montre que notre ontologie de types de relations permet de représenter des connaissances très diverses, tout en limitant pour chacune d'elles les alternatives de représentation, mais sans toutefois conduire à une seule alternative : différents ensembles de types de relations peuvent toujours être utilisés pour représenter une même connaissance. Cela est génant pour la modélisation de connaissances et la recherche d'informations, car des GCs peuvent alors représenter des connaissances voisines et pourtant ne pas pouvoir être comparés. Une solution est de donner des définitions aux types des relations de telles sortes que différentes alternatives de représentations se ramènent, après expansion des définitions dans les GCs, à des représentations comparables. Mais ceci n'est pas toujours possible. Une solution complémentaire pour un ou plusieurs cogniticiens travaillant sur une même expertise est de choisir quels types de relations devront être privilégiées lors de la représentation de connaissances.

Nous donnons ci-après quelques exemples de représentation du contenu de la phrase : "Pour guider un cogniticien dans la construction d'une ontologie, CGKAT fournit une ontologie par défaut". Comme cette phrase peut se décomposer en trois sous-propositions, nous donnons tout d'abord une représentation de ces propositions :

```
[Proposition : #CGKAT_fournit_ontologie_par_défaut
                    [W_provision__providing__supply__supplying]-
                        {  (Agent)→[W_software__software_system : #CGKAT];
                           (Object)→[Default_ontology];
                        }
].
[Proposition : #construction_ontologie   [W_design_designing]-
                                    {  (Agent)→[Knowledge_engineer : *x];
                                       (Result)→[Ontology];
                                    }
].
[Proposition : #CGKAT_guide_cogniticien_dans_construction_ontologie
                    [W_guidance__guiding__steering]-
                        {  (Agent)→[W_software__software_system : #CGKAT];
                           (Recipient)→[Knowledge_engineer : *x];
                           (Object)→[Proposition : #construction_ontologie];
                        }
].
```

Dans cette dernière représentation, nous n'avons pas répété le GC en partie référent du concept [Proposition : #construction_ontologie]. Nous faisons ci-après de même pour les concepts [Proposition : #CGKAT_fournit_ontologie_par_défaut] et [Proposition : #construction_ontologie].

```
(Causation)-          //"Consequence", inverse de "Causation" peut également être utilisé
   {  ←[Situation]→(Descr)→[Proposition : #CGKAT_fournit_ontologie_par_défaut];
      →[Situation]→(Descr)→[Proposition : #CGKAT_guide_cogniticien_dans_construction_ontologie];
   }.

(RST_Volitional_Cause)-      //Cette représentation peut être une forme contractée de la précédente
   {  →[Proposition : #CGKAT_fournit_ontologie_par_défaut];
      ←[Proposition : #CGKAT_guide_cogniticien_dans_construction_ontologie];
   }.
```

```
(RST_Volitional_result)-        // "RST_Volitional_result" est l'inverse de "RST_Volitional_Cause"
    { ←[Proposition : #CGKAT_fournit_ontologie_par_défaut];
      →[Proposition : #CGKAT_guide_cogniticien_dans_construction_ontologie];
    }.

[W_provision__providing__supply__supplying]-
            { (Agent)→[W_software__software_system : #CGKAT];
              (Object)→[Ontology];
              (Purpose)→[Proposition : #CGKAT_guide_cogniticien_dans_construction_ontologie];
            }.
```

5.7 Exploitation de définitions de types pour guider l'extraction et la représentation de connaissances

5.7.1 Des listes de relations connectables aux concepts de certains types

Il existe deux moyens dans CGKAT de spécifier que des relations de certains types sont connectables à un concept d'un certain type :

1. ***via des définitions de ce type de concept***. Une définition de type de concept par conditions typiques permet de représenter les types des *principales* relations qui peuvent être connectées à un tel concept (i.e. les types des relations qui lui sont typiquement connectées)[1]. Les figures 5.34 à 5.36 de la page suivante montrent ainsi comment nous avons représenté les types de relations généralement associées aux types de concepts de haut niveau de notre ontologie.
2. ***via les signatures des relations***. Rappelons que lorsqu'un concept d'un GC est sélectionné ou bien lorsqu'un type de concept est sélectionné dans le menu de gestion des types de concepts, CGKAT peut exploiter les signatures des relations pour afficher dans le menu de gestion des types de relations, une liste indentée des types des relations connectables au concept sélectionné ou à un concept d'un type correspondant à celui sélectionné.

Ainsi, partant d'un ***concept*** ou sachant qu'il va utiliser *un concept d'un certain* ***type***, l'utilisateur peut aisément construire un GC de deux façons :

1. en ***copiant*** (par une opération de copier/coller) des *parties d'une définition associée à ce type ou à un de ses supertypes*. L'utilisateur peut utiliser le corps d'une définition comme un ***modèle*** ou encore un ***masque de saisie*** : il copie les concepts et les relations qui l'intéresse puis spécialise ces copies pour exprimer l'information qu'il veut représenter.
   Certains types de relations utilisés dans les définitions des figures 5.34 à 5.36 de la page suivante sont assez généraux de façon à recouvrir de nombreux autres types. Par exemple, le type Situation_temporal_location est supertype entre autres de Point_in_time, Duration, T_Since et T_Until. L'utilisateur est guidé par l'ontologie pour spécialiser les concepts contenant des types généraux lorsqu'il copie des parties de définitions de types.
2. en ***sélectionnant*** *des types de relations dans la liste indentée des types de relations connectables à ce concept*. Lorsqu'un type est sélectionné, l'utilisateur peut demander la connexion au concept d'une relation du type sélectionné (les autres concepts nécessaires à cette relation sont également créés : ce sont des concepts génériques ayant pour types ceux précisés dans la signature de la relation).

Ces deux sortes de telles listes de relations typiquement connectables à un concept peuvent donc aider l'utilisateur :
1) à savoir comment représenter une information,
2) à la représenter rapidement, et
3) à ne pas oublier de représenter des informations importantes, par exemple celles qui sont contextualisantes : localisation dans le temps et l'espace d'une situation, modalités logiques, déontiques et épistémiques sur une proposition, etc.

La prochaine section montre comment des listes de relations connectables à un concept de ce type sont utilisées par des méthodes d'extraction ascendante de connaissances.

---

1. Rappelons les contrôles effectués par CGKAT lors de la création d'une définition de type par conditions nécessaires et/ou suffisantes ou typiques : CGKAT
a) vérifie que le graphe utilisé pour le corps de la définition respecte les contraintes posées sur les types de relations (signatures) et sur les référents individuels (relation de conformité),
b) définit ce type comme un sous-type du type du concept porteur du paramètre formel (cf. 6.2.2.5) à condition que cela n'entraîne pas de cycle dans la hiérarchie, ni ne crée un sous-type commun à des types exclusifs.

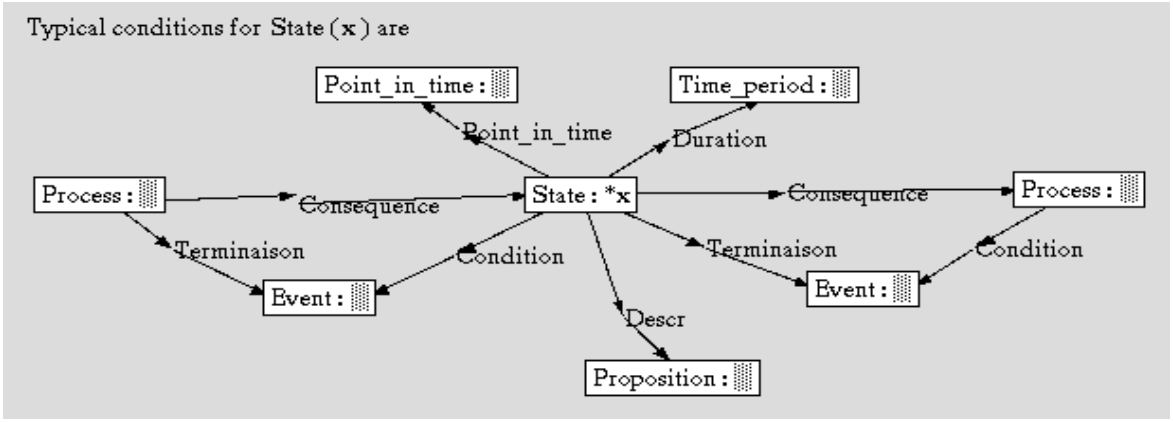


Figure 5.34: *Types de relations typiquement connectées à un concept de type State.*

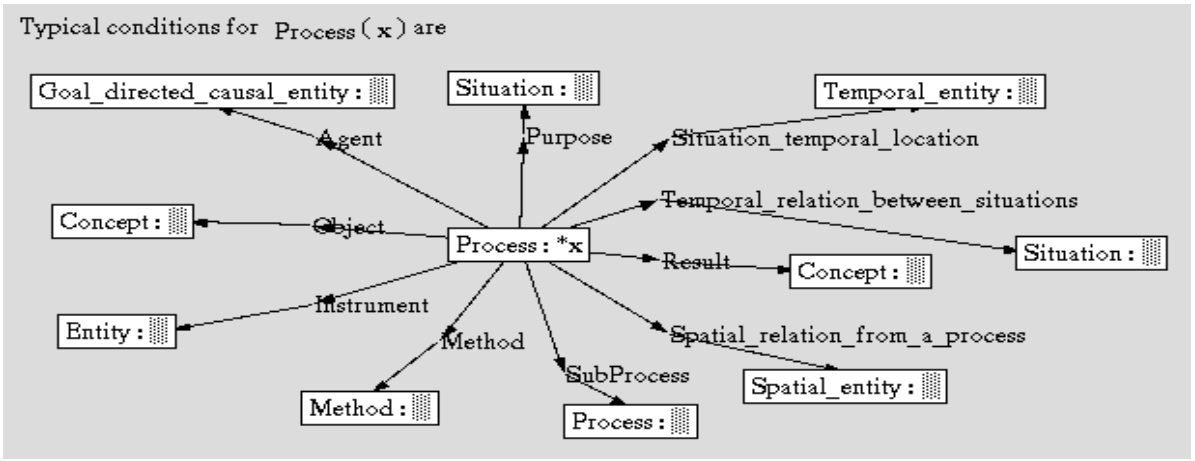


Figure 5.35: *Types de relations typiquement connectées à un concept de type Process.*

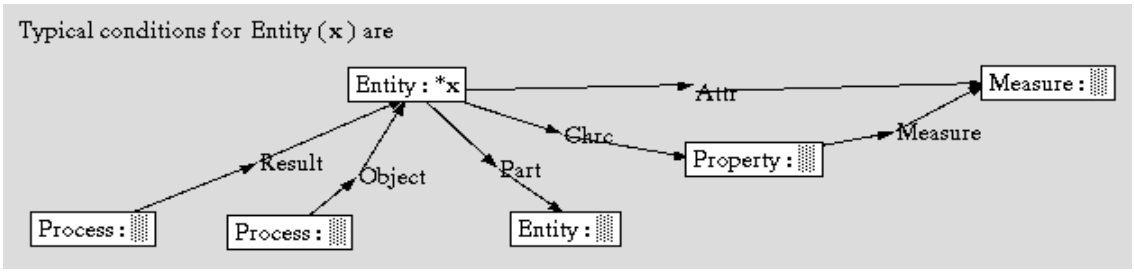


Figure 5.36: *Types de relations typiquement connectées à un concept de type Entity.*

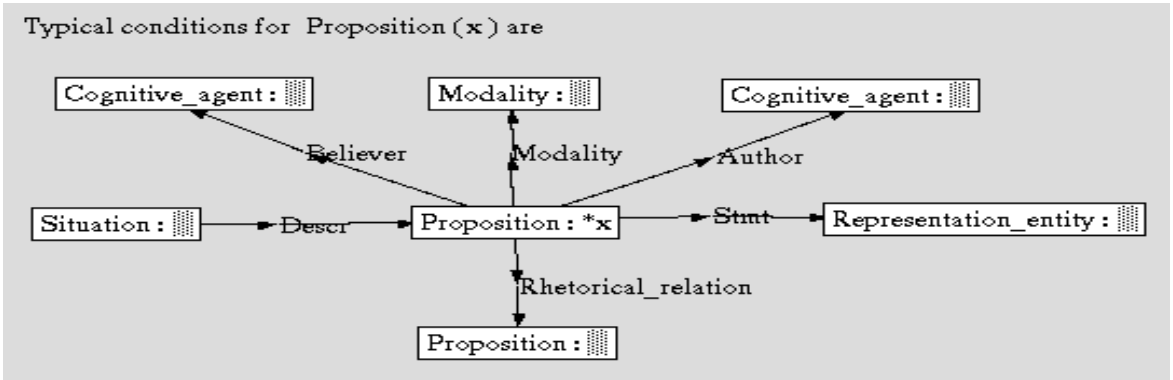


Figure 5.37: *Types de relations typiquement connectées à un concept de type Proposition.*

5.7.1.1 Extraction ascendante de connaissances

Dans l'***approche ascendante***, les données de l'expertise sont élicitées et analysées graduellement, classées et structurées. Pour guider le cogniticien dans cette tâche, certaines méthodes d'extraction ou de modélisation ascendante des connaissances préconisent l'utilisation de certaines catégories conceptuelles et de certaines relations pour organiser ces catégories. Nous détaillons ci-après deux méthodes pour utiliser de telles catégories et relations dans l'extraction de connaissances.

Gordon & Gill (1992), qui ont défini six types de concepts de haut niveau[1] et une vingtaine de types de relations[2] permettant de relier des concepts de ces types, ont proposé et expérimenté la méthode suivante pour "éliciter" les connaissances d'un expert :

1. Après une brève première interview, un graphe initial contenant quelques concept généraux est créé par le cogniticien.
2. Au cours de chacune des interviews suivantes, une portion de ce graphe est choisie et développée de manière systématique. Pour chaque concept (entité, état, processus), compte-tenu du type de ce concept et donc des relations qui peuvent le relier avec d'autres concepts de certains types, le cogniticien demande à l'expert quels sont, dans son domaine, ces autres concepts (si ces concepts existent, ils sont bien sûr ajoutés au graphe et le processus est itéré). Par exemple si l'expert parle d'un état, le cogniticien va lui demander entre autres quel est le processus qui a conduit à cet état, comment cet état va se terminer et quels processus vont alors être déclenchés.
   L'élicitation des connaissances, c'est-à-dire la construction progressive du graphe, se fait plus ou moins en largeur d'abord, par séquences d'interviews assez courtes (quelques minutes chacune), de façon à rester dans un même contexte ou à un même niveau de généralité au cours de chaque interview. L'élicitation s'arrête lorsque le domaine semble également couvert en profondeur (au moins pour les tâches que le système final doit exécuter).

Cette méthode est applicable seulement parce que le nombre des types de relations potentiellement connectables à chaque concept est assez limité. Chez Gordon & Gill (1992), cela est assuré par une ontologie de types de relations limitée à une vingtaine de relations (chaque concept peut être connecté à d'autres concepts par en moyenne 6 ou 7 relations de types différents).
L'ontologie proposée par CGKAT contient beaucoup plus de relations, et l'utilisateur peut compléter cette ontologie. Aussi, il ne semble pas réaliste que le cogniticien utilise la méthode de Gordon & Gill en s'appuyant sur des générations de "listes indentées des relations connectables à un concept d'un type donné" : même si de telles listes sont hiérarchiquement organisées (ce qui peut être utilisé pour poser à l'expert des questions générales et dans le cas de réponses positives, de raffiner les questions), le nombre de questions possibles reste grand. Par contre, il semble plus raisonnable qu'il utilise cette méthode en s'appuyant sur des "définitions de types de concepts par conditions typiques qui rassemblent les principales relations connectables à des concepts de ces types (ou de leurs sous-types)".

Tout comme la méthode de Gordon & Gill, la méthode KOD (Vogel, 1988) est une méthode uniquement ascendante.
Un cognititien qui suit la méthode KOD travaille tout d'abord sur des documents techniques ou des retranscriptions d'interviews non dirigés. Dans cette phase, il répertorie tout d'abord les différents termes et expressions relatifs à des "taxons" (que l'on peut rapprocher de nos entités), à des "actons" (que l'on peut rapprocher de nos processus) et à des "schémas" (que l'on peut considérer comme des propositions exprimant des relations entre sous-propositions, notamment des relations causales ou modales).
Après cette étude terminologique, il représente les diverses notions trouvées et les organise les

---

1. Ces types semblent correspondre à nos types Physical_entity, Property, State, Process, W_event/n, W_goal__end et Method.

2. Voici ces types de relations dans la terminologie de Gordon & Gill (1992) : Is-A, Instance-of, Has_Part, Property, Equivalent-to, Spatial_relation, Temporal_relation (Before, After, During), Logical_relation (And, Or), Reason, Initiate, Implies, Consequence, Manner, Means et Refers-to. Ces types sont inclus, parfois sous des noms plus précis, dans notre ontologie.

actons par des relations de spécialisation. Puis, pour chaque notion, il va rechercher dans les documents sources d'expertise, ses relations avec les autres notions, i.e. notamment,
a) pour un taxon, ses sous-parties, ses attributs et leurs domaines de valeurs, et
b) pour un acton, les contraintes relatives à ce que doit être son "émetteur" (Agent), son "récepteur" (Object), ses "ressources" (i.e. Instrument, Material, Parameter dans notre ontologie, mais KOD ne fait pas la différence), ses "versions" (Method), ses "constituants" (SubProcess), son "déclenchement" (Condition), et ses contraintes spatio-temporelles.
Les diverses notions sont ainsi liées entre elles. Des interviews dirigées sont alors effectuées afin de pouvoir représenter certaines relations et notions qui semblent manquer.

L'ontologie proposée par CGKAT permet la représentation des connaissances en utilisant les catégories conceptuelles ou les relations préconisées par la méthode de Gordon & Gill (1992) ou la méthode KOD. L'utilisateur de CGKAT peut donc suivre ces méthodes et être guidé en cela par les définitions associées à ces catégories ou par les listes indentée de types de relations connectables à un concept.

Dans notre ontologie, nous avons associé des définitions par conditions typiques uniquement à des types de concepts très généraux : Entity, Situation, Process, Proposition. Un cogniticien peut bien sûr associer d'autres définitions plus adéquats aux types de concepts qu'il utilise et ainsi faciliter sa tâche d'extraction ou de représentation ou celle des autres cogniticiens qui travaillent sur la même ontologie. Pour créer ces définitions, il peut s'inspirer de définitions de types plus généraux ou du résultat d'une génération de liste indentée des types des relations connectables à un concept d'un certain type[1].

Rappelons enfin que des types de concepts, y compris ceux créés par l'utilisateur peuvent spécialiser ou être spécialisés par des types de concepts de WordNet. Nous avons montré dans la section 5.3.3 à la page 122 que cela pouvait faciliter la modélisation ascendante des connaissances.

5.7.1.2 Recherche des définitions de types adéquates

La recherche dans CGKAT des définitions associées à un type peut s'effectuer par navigation, requête et filtrage (cf. section 5.2). Les requête peuvent être lexicales (recherche de sous-chaînes de caractères dans les noms de types) ou conceptuelles (projection d'un GCS).

Par exemple, la requête conceptuelle suivante effectuée sur l'ensemble des définitions chargées dans CoGITo[2], affichera sous forme linéaire ou graphique[3], les définitions par conditions typiques qui incluent un concept de type Proposition ou d'un de ses sous-types :
> ? [Proposition : *] with {defKind:TC;}; //→ spécialisations de ce GC avec cette méta-information

Dans CGKAT, les généralisations d'un GC peuvent être recherchées de la manière similaire :
> gene [Proposition : *] with {defKind:TC;}; //→ généralisations de ce GC avec cette méta-info.

Il serait peut-être intéressant dans une telle requête, de pouvoir ajouter comme contrainte que le concept doit comporter un référent générique nommé, par exemple de la manière suivante :
> gene [Proposition : *x] with {defKind:TC;}; //→ généralisations de ce GC avec cette méta-info.

---

1. Notons qu'une définition par conditions typiques associée à un type peut ne pas seulement être une spécialisation de définitions par conditions typiques associés aux supertypes de ce type. Par exemple, un schéma pour le type Oiseau peut représenter le fait que la plupart des oiseaux peuvent voler, mais un schéma pour un certain type d'oiseau peut représenter le fait que la plupart des individus de ce type ne peuvent pas voler. Cela n'est pas possible pour des définitions par conditions nécessaires (CNs) ou par conditions nécessaires et suffisantes (CNSs).

2. Dans CGKAT, les requêtes conceptuelles peuvent se faire sur l'ensemble des GCs qui forment les faits de la base de CoGITo, et/ou sur l'ensemble des GCs qui sont utilisés pour définir les types de concepts ou de relations utilisables dans la base.

3. Ceci dépend d'une option préalablement spécifiée ; cependant, si une définition n'a pas été rentrée sous une forme graphique, c'est toujours sa forme linéaire qui sera affichée.

En effet, dans ce cas, seules les définitions par conditions typiques qui ont été associées au type Proposition ou à un de ses supertypes, seraient retrouvées. Ainsi, sans qu'un mécanisme d'héritage soit ajouté, il serait possible de rechercher des définitions dont hérite[1] (CNs ou CNSs) ou peut hériter (schémas) un type donné.

5.7.2 Des définitions de types pour guider la modélisation de connaissances

Outre des relations typiquement connectées à des concepts, une définition de type peut être utilisée pour représenter différentes sortes de "modèles", e.g. des modèles de tâches génériques (cf figures 5.23 et 5.24 à la page 145). L'exploitation des définitions de types n'est donc pas limitée à l'approche ascendante. La recherche des modèles adéquats est facilité par l'organisation des types et la spécialisation entre les graphes (cf. section précédente).

Il est maintenant reconnu que les approches d'extraction/modélisation ascendantes et descendantes doivent être combinées (Rademakers & Vanwelkenhusen, 1992) (cf. section 2.2.3). Cela est possible dans CGKAT puisque dans l'ontologie qu'il exploite, des types et des modèles provenant de diverses méthodologies peuvent être représentés et organisés, comme c'est déjà le cas dans l'ontologie proposée par défaut. Il apparaît ainsi qu'il y a peu de différences dans la façon d'exploiter ces modèles pour l'extraction/modélisation ascendante ou descendante : pour chaque concept contenu dans le modèle suivi, le cogniticien va essayer de savoir, compte-tenu des contraintes qui ont été posées sur ce concept (i.e. des diverses relations qui doivent exister entre lui et les autres concepts), si un tel concept existe dans l'expertise considérée.
Nous avons décrit dans (Martin, 1993b, 1994) une méthode permettant de savoir quelles sortes d'informations rechercher ou éliciter pour "instancier" un modèle de tâche dans KADS et ainsi collecter les informations nécessaires à la résolution de problèmes et à la génération explications. Cette méthode qui prend la forme d'une typologie de questions est assez proche de celle de Gordon & Gill (1992) mais elle est basée sur les éléments d'un modèle de tâche : les différents types connaisssances qui peuvent être recherchées à partir de ces éléments sont listées. On retrouve, exprimés sous forme de questions, les types de relations que nous avons associées aux types de concepts généraux (cf. figures 5.34 à 5.37). D'autres sortes de relations/questions sont plus relatives à la nature de chaque élément d'un modèle de tâche ; par exemple pour une méthode : domaine d'application, méthodes alternatives, critères d'évaluation des résultats, rôles de ces résultats par rapport à diverses inférences dans des tâches, etc. Bien que nous ne l'ayons pas encore fait, ces différentes sortes de relations pourraient être représentées dans des définitions par conditions typiques associées aux types relatifs aux éléments d'un modèle de tâche.

Nous avons noté dans la section 2.3.1 à la page 29 que les outils ou les langages d'acquisition de connaissances incluent souvent "en dur" des distinctions conceptuelles.

1. Par exemple, les menus de l'outil K-Station (Albert & Vogel, 1989) (ILOG, 1993a) sont dédiés à la représentation des catégories conceptuelles préconisées par la méthodologie KOD.
2. Bien que de nombreux outils soient maintenant génériques vis-à-vis des modèles de tâches (une bibliothèque de modèles est proposée), les langages de représentation qu'ils permettent d'utiliser pour représenter ces modèles incluent en dur des notions comme tâche, sous-tâche, méthode, inférence et rôle (c'est par exemple le cas des modèles construits avec CommonKADS CML (Schreiber & al., 1994))[2]. Dans ces outils, les langages de représentation des connaissances du domaine sont généralement peu contraignants, mais ces outils ne fournissent pas d'ontologie par défaut pour guider la construction de ces représentations.

---

1. Des conditions nécessaires (et éventuellement suffisantes) pour un type sont des conditions nécessaires pour ses sous-types. Des conditions suffisantes pour un type ne sont pas des conditions suffisantes pour ses sous-types.

2. Rappelons cependant que certains outils, e.g. Protégé-II (Puerta & al, 1992) et Cue (Heijst & Schreiber, 1994), 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.

L'utilisateur de ces outils ou de ces langages est alors guidé par leurs menus ou leurs structures mais les distinctions conceptuelles qu'ils intègrent ne sont pas représentées par des types dans une ontologie où elles pourraient être complétées, spécialisées ou organisées avec d'autres types de cette ontologie. L'outil ou le formalisme peut ainsi être limité à la description de certaines sortes de connaissances. La comparaison des connaissances peut également être limitée du fait que les distinctions conceptuelles intégrées à l'outil ou au formalisme ne sont pas organisées entre elles ou par rapport aux autres types de l'ontologie.

Au contraire, CGKAT exploite le formalisme des GCs qui n'inclut que très peu de primitives épistémologiques (principalement : concept, relation, type de concept ou de relation, et référent) mais permet l'exploitation d'une ontologie : treillis de types de concepts, hiérarchie de type de relation, liste de référents conformes aux types de concepts.
Ainsi dans CGKAT, des ***contrôles de cohérence*** sont effectués en exploitant les contraintes définies dans l'ontologie (notamment via les signatures des relations, la relation de conformité, et les liens d'exclusion entre types).
Par ailleurs, l'utilisateur est ***guidé*** dans la représentation de connaissances par les définitions de types. Il peut effectuer des ***contrôles d'achèvement***, c'est-à-dire vérifier si des connaissances ont été modélisées pour tous les types de connaissances contenues dans les modèles suivies, et si pour chaque type, toutes les connaissances nécessaires ont été modélisées. Pour l'aider dans cette tâche, il peut utiliser l'opérateur de comparaison de graphes[1] de CGKAT pour comparer les graphes "modèles" au graphes qui "instancient ces modèles". En utilisant le langage de commandes de CGKAT, des scripts peuvent être écrits pour effectuer de telles vérifications (les possibilités offertes par ce langage de commandes seront détaillées dans le prochain chapitre).

La possibilité de représenter des connaissances simplement en réutilisant et combinant des parties de définitions de types d'une ontologie est intéressante puisque cela permet d'utiliser relativement facilement un langage de représentation *à la fois formel et général*. En effet, avec un formalisme comme celui des GCs où tous les types de concepts ou de relations utilisés doivent être déclarés et organisés dans des hiérarchies, la représentation de connaissances est une opération longue et difficile si l'utilisateur ne peut réutiliser rapidement des types adéquats. Cet obstacle a conduit nombre de chercheurs à utiliser des langages de représentation de connaissances moins formels, notamment des langages de frames où certains attributs sont prédéfinis et où les autres peuvent être des chaînes de caractères quelconques. Dans de tels langages[2], les connaissances peuvent être facilement rentrées mais les relations entre les objets ne peuvent être explicitement définies, ce qui limite les possibilités de comparaisons de connaissances et donc de recherches et d'inférences.

---

1. Cet opérateur utilise la projection pour vérifier si un GCS est une spécialisation d'un autre GCS.

2. Nous ne parlons pas ici de tous les langages de frames, mais de ceux "où certains attributs sont prédéfinis et où les autres peuvent être des chaînes de caractères quelconques".
Un exemple typique de tels langages est le langage ClearTalk (Skuce, 1996) des systèmes de gestion de connaissances CODE4 (Skuce & Lethbridge, 1995) et IKARUS (Kavanagh, 1996). Les expressions dans ClearTalk peuvent être assertées suivant le schéma "Objet Attribut Valeur". Seul l'objet (un type de concept ou un individu) peut et doit avoir été déclaré et organisé dans une taxinomie. Les autres champs n'ont pas à contenir des termes déclarés, mais le champ Valeur peut être librement structuré en divers sous-champs comme le montre les exemples suivants : 1) Car part 4_wheels, 2) Person goes_to work time_on:2_years, 3) Car gets attr:rusty time_after:2_years location:canada freq:usually, 4) Car will_get attr:rusty if:(#is not cared_for). Les attributs/valeurs ainsi associés à des types peuvent être hérités par leurs sous-types. De telles expressions sont aisées à écrire, mais les limites d'un tel langage sont claires : seules les relations de spécialisation entre les types d'objets étant explicitement définies, les recherches ou les comparaisons de connaissances ne peuvent reposer que sur ces relations et sur des comparaisons de chaînes de caractères.
Une certaine ressemblance apparaît entre les expressions de ClearTalk et les méta-informations que CGKAT permet d'associer aux types et aux GCs ; cependant, les valeurs dans les champs de ces méta-informations peuvent être des types ou des marqueurs individuels, et nous avons défini une relation de spécialisation/généralisation entre les méta-informations qui prend en compte ces types et ces marqueurs. Par ailleurs, les méta-informations ne sont à utiliser que comme des compléments particuliers à des GCs ou des définitions de types, i.e. typiquement pour représenter leurs auteurs et leur domaines d'application.

Chapitre 6 Des documents structurés pour rechercher et organiser des informations *et* des connaissances

6 Sommaire

6.1 Introduction

L'AC implique 1) des recherches d'informations[1] dans des documents (en particulier pour modéliser les connaissances sous-jacentes), 2) des recherches de connaissances dans la BC[2] (en particulier pour les comparer, les regrouper, les abstraire et les valider), et 3) des recherches, à partir des informations des documents, des connaissances les représentant dans la BC, et inversement. Afin de faciliter ces recherches par la navigation et des systèmes de requête, les éléments de ces documents sources ou générés doivent être structurés et/ou indexés, et reliés par des liens de type hypertexte.

Par ailleurs, nous avons montré que les systèmes de gestion et de recherche d'informations s'orientent de plus en plus vers une *indexation* fine des informations i.e. vers une représentation sémantique et formelle ou semi-formelle de leur contenu. Quelques systèmes hypertextes permettent à leurs utilisateurs de créer des réseaux sémantiques et de les exploiter pour la RI dans les documents par navigation ou requête (e.g. MORE (Lucarella & al., 1993), Macweb (Nanard & al., 1993a, 1993b) ; cf. section 3.1.2.3 pour les systèmes de requête sur la structure d'un réseau hypertexte). D'autres systèmes de RI effectuent une analyse du contenu sémantique des documents pour créer automatiquement des représentations de ces documents ou de leurs éléments, et/ou de certains liens sémantiques les reliant.

Ce double constat nous a conduit à suivre cette orientation : nous permettons l'*indexation* d'informations dans des documents par les éléments d'une BC, et surtout nous ***combinons*** des techniques avancées de gestion d'information (celles offertes par l'éditeur de documents structurés et hypertextes Thot) avec des techniques avancées de représentation et de structuration de connaissances (le formalisme des GCs), et ce pour gérer aussi bien des informations que des connaissances.

Pour cela, l'idée principale sur laquelle nous nous sommes appuyé a été de permettre la ***construction*** et le ***stockage*** de "connaissances" *(plus exactement de représentations de connaissances)* dans des éléments de documents structurés hypertextes.

1. Ces connaissances étant *stockées* comme des informations, elles peuvent être ***recherchées et gérées*** via Thot comme des informations. Elles peuvent par exemple être connectées à d'autres éléments de document (contenant des informations ou des connaissances) par des liens structurels ou hypertextes ou encore avec des inclusions.
2. Lors de la *construction* de ces connaissances par l'utilisateur, et lors de l'ouverture d'un document contenant ces connaissances, notre outil CGKAT construit automatiquement une base de connaissances dans un format permettant d'exploiter le formalisme dans lequel elles sont représentées. Cette base de connaissances est celle de CoGITo (Haemmerlé, 1995a, 1995b). Toute modification sur les GCs dans les documents est aussitôt répercutée par CGKAT dans la base de CoGITo. Inversement, dans CGKAT, les recherches ou manipulations possibles avec CoGITo sur les connaissances peuvent être exploitées pour rechercher et manipuler les informations auxquelles sont liées ces connaissances.

Pour indexer des informations par des connaissances, puis pour rechercher et manipuler des connaissances et des informations, CGKAT offre à ses utilisateurs :
1) ***différents types de liens hypertextes*** (i.e. différents types d'indexation d'information) ; l'un d'eux exprime le fait que la connaissance *représente* l'information à laquelle elle est liée par le lien hypertexte ;
2) ***un langage de requêtes*** qui exploite le formalisme de représentation de connaissances et certaines relations hypertextes entre connaissances et informations (notre langage ne permet pas encore d'exploiter tous les types de liens hypertextes entre connaissances et informations).

---

1. Comme au chapitre 3, par "informations" nous référons aux éléments de documents, et par "connaissances" aux éléments d'une BC.

2. Par "BC", nous référons aussi bien aux modèles formels ou semi-formels créés lors de la modélisation d'expertise qu'à la base de connaissances du système final.

Grâce à ce langage, des ***documents virtuels*** peuvent être générés par sélection d'informations dans un ou plusieurs documents. Ces documents virtuels sont des ***vues***[1] sur ces documents, et comme ils peuvent inclure des connaissances, ils peuvent également être des vues sur la BC. Pour la création de ces documents virtuels, nous utilisons des inclusions des éléments de document sélectionnés. Ceux-ci sont donc accessibles par navigation hypertexte. Les documents virtuels permettent donc la combinaison des recherches par requête et par navigation (l'exécution d'une requête permet de restreindre l'espace de recherche pour la navigation : l'exploration peut s'effectuer depuis les informations relatives à un point de vue). La génération de documents virtuels est également un des points forts de MacWeb.

Ainsi, grâce à l'intégration de l'éditeur de documents structurés Thot et du gestionnaire de GCs CoGITo, CGKAT est à la fois un outil d'acquisition des connaissances et un système de structurations d'informations "basé sur les connaissances". La figure 6.1 rappelle l'architecture de CGKAT.

Tout d'abord dans ce chapitre, nous présentons comment nous utilisons un éditeur de documents structurés pour stocker, construire et organiser une base de connaissances. Puis nous précisons les principes que nous avons choisis pour permettre, contrôler et exploiter l'indexation des informations des documents par des connaissances.

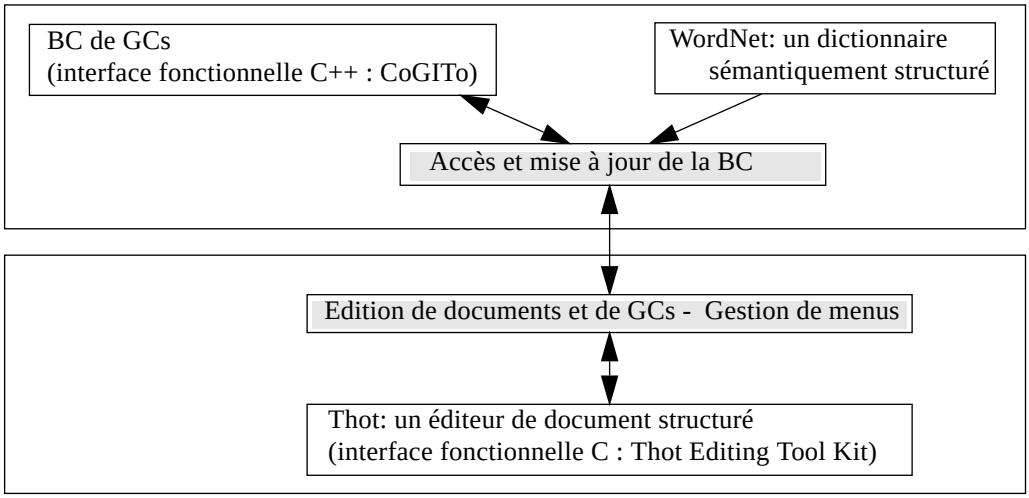


Figure 6.1: *Architecture de CGKAT* (les parties grisées représentent notre code).

1. Dans le sens où nous avons défini ce terme.

6.2 Construction d'une base de connaissances via un éditeur de documents structurés

6.2.1 Motivations

Rappelons tout d'abord l'intérêt d'utiliser un éditeur de documents structurés tels que Thot pour structurer et manipuler des informations.
1) Thot permet l'affichage et l'édition des éléments d'un document suivant le modèle de présentation choisi par l'utilisateur, et éventuellement dans plusieurs fenêtres montrant une "vue" différente sur le document ; le zoom est également géré.
2) Thot permet à un créateur de document d'appliquer des règles de présentation spécifiques (position, couleur, fonte, etc.) à un ED ou un ensemble d'EDs à condition que cette présentation soit permise par le modèle de présentation choisi.
3) Thot guide l'édition du document, tout d'abord en présentant les EDs à remplir obligatoirement, et en proposant pour chaque ED sélectionné, un menu contextuel qui ne contient que les commandes d'insertion (création) autorisées par le modèle structurel à cet endroit du document (e.g. l'insertion d'un composant de l'ED).
4) Thot permet d'organiser des EDs dans un ou plusieurs documents via des liens structurels ou hypertextes, et avec des inclusions (cela permet la gestion de plusieurs vues et le partage d'information entre plusieurs EDs).
5) Thot permet à un lecteur de document de rechercher des EDs a) par des requêtes sur leurs types ou leurs attributs, et b) par navigation hypertexte sur les liens structurels et les liens hypertextes (un lien d'inclusion étant également un lien hypertexte, la source d'une inclusion peut être retrouvée immédiatement, et depuis une source d'inclusion, ses inclusions peuvent être retrouvées une par une).
6) Thot dispose d'une interface fonctionnelle permettant de programmer certaines procédures à exécuter (e.g. pour modifier un document ouvert) lors de certaines actions de l'utilisateur sur des EDs de certains types (e.g. ouverture d'un document, création d'un ED, changement de sa présentation ou d'un de ses attributs). Ainsi un document Thot peut être un document actif et servir d'interface pour des applications.

Les applications exploitant et étendant Thot, telle Alliance (Decouchant, 1995) qui permet à plusieurs éditeurs Thot d'effectuer de l'édition coopérative, sont également intéressantes pour la construction et l'organisation de représentations de connaissances.

6.2.2 Construction d'un graphe conceptuel via un élément de document

Afin d'interfacer une base de GCs avec des documents Thot, nous permettons à l'utilisateur de construire des EDs qui affichent des GCs. Pour cela, de même que la librairie Thot propose des modèles structurels et des modèles de présentation pour les EDs de type Article, Section, Paragraphe, Tableau, Arbre, ..., nous avons créé un modèle structurel et un modèle de présentation pour les EDs de type CG. Pour simplifier, nous parlerons d'EDs GC ou tout simplement de GCs. Nous avons de plus créé un "modèle d'interface" pour l'ED GC de façon que toutes les créations, modifications et destructions d'EDs GC soient immédiatement répercutées par CGKAT dans CoGITo. Ces trois modèles sont décrits en annexe 4. Le modèle de présentation d'un ED GC ne permet qu'un affichage "littéral" des GCs au sens de Cyre & al (1994) : il s'agit d'un affichage des noeuds concepts et relations indépendant des catégories sémantiques auxquelles ces noeuds font référence (cf. section 4.5.1.1 à la page 99). Nous décrivons ci-dessous les principales parties de notre modèle structurel pour un GC. Plus précisément, ce modèle décrit 1) un GC composé d'une liste de concepts et de relations (nous appelons un tel GC, un CGgraph), 2) un GC composé d'un unique concept (nous présenterons plus loin un exemple d'utilisation d'un tel élément), et 3) une définition de type via une lambda-abstraction. Pour des raisons pratiques, il nous a semblé préférable de réunir ces trois types d'éléments dans un seul modèle. *Un "GC" désigne donc dorénavant un élément d'un de ces trois types*.

```
STRUCTURE CG;  DEFPRES CGP;  { Structural model for a (DE of type) CG;  default presentation: CGP }
STRUCT
   CG = CASE OF  { here a CG is either a CGgraph, a SingleConcept or a TypeDefinition }
            CGgraph (ATTR  CGname = TEXT;  Comment = TEXT;  To_Element = REFERENCE(ANY);
                           ChangeIntoDefinition = Yes, No;
                           ConceptFrame = Without, Rectangular, RectangularShaded,
                                          Rounded, RoundedShaded;
                           RelationFrame = WithoutFrame, Rounded, RoundedShaded;
                           OrigRelLink1_Pos = Center1_, Right, Left1_, Top, Bottom;
                           DestRelLink1_Pos = Center_1, Right, Left_1, Top, Bottom;
                           OrigRelLink2_Pos = Center2_, Right, Left2_, Top, Bottom;
                           DestRelLink2_Pos = Center_2, Right, Left_2, Top, Bottom;
                           ArrowOnLink1 = ok, no;  ArrowOnLink2 = yes, not )
                  = ConceptOrRel_List with ConceptFrame?=Rectangular, RelationFrame?=WithoutFrame,
                                           OrigRelLink1_Pos?=Center1_, DestRelLink1_Pos?=Left_1,
                                           OrigRelLink2_Pos?=Left2_,   DestRelLink2_Pos?=Left_2,
                                           ArrowOnLink1 ?= no,  ArrowOnLink2 ?= yes;
            SingleConcept  = CASE OF Concept; ConceptInclusion;  END;
            TypeDefinition (ATTR  !DefKind = NSC_for_ConceptType, NC_for_ConceptType,
                                             SC_for_ConceptType, TC_for_ConceptType,
                                             NSC_for_RelationType, Unspecified_DefKind,
                                             No_more_a_definition;
                                  !Completed = not_yet, yes;  { '!' means that the attribute is mandatory }
                                   ... { declaration of the same attributes as for CGgraph }  )
                  = BEGIN  DefinedType = TEXT;  LambdaVar = TEXT;
                           DefBody = CGgraph;
                    END  with  DefKind ?=TC_for_ConceptType,  Completed ?=not_yet;
        END;                { with → default values for attributes }
   ConceptOrRel_List = LIST OF (ConceptOrRelation);

   ConceptOrRelation = CASE OF
      Concept (ATTR  IDinCG=INTEGER;  ... { declaration of the same attributes as for CGgraph }  )
            = BEGIN  ConceptType = TEXT;
                     Referent = CASE OF  Variable=TEXT;  Individual=TEXT;  END;
                   ? CGReferent (ATTR  ... { same attributes as for CGgraph } )     { ? → optional }
                           = CASE OF  CGgraph; TypeDefinition;  END;
                    ? DescriptionReferent = UNIT;  { → texts, images and other EDs may be included
inside
                                                     a Concept; this information is not copied in CoGITo }
              END;
      ConceptInclusion (ATTR  IDinCG; Comment; To_Element; ChangeIntoDefinition; ConceptFrame )
            = BEGIN  TheIncludedConcept = Concept;  END;
      Relation (ATTR  IDinCG;  Comment; To_Element;  !OrigNode=REFERENCE(SingleConcept);
                                                     !DestNode=REFERENCE(SingleConcept);  ... )
            = BEGIN
                  LinkToOrig (ATTR OrigRelLink1_Pos; DestRelLink1_Pos; ArrowOnLink1; ... )
                        = BEGIN   GRAPHICS;   ? LinkToOrig_Mark = TEXT;   END;
                ? OtherLinkToOrig = LIST OF (LinkToOrig);
                  RelationType (ATTR RelationFrame) = TEXT;
                  LinkToDest (ATTR OrigRelLink2_Pos; DestRelLink2_Pos; ArrowOnLink2; ...)
                        = BEGIN   GRAPHICS;   ? LinkToDest_Mark = TEXT;   END;
                ? OtherLinkToDest = LIST OF (LinkToDest);
              END;
      RelationAndConcept = BEGIN  Concept; Relation;  END;  { gadget for building CGs faster }
                         END;
...
```

Figure 6.2: *Les parties principales de notre modèle structurel pour un GC (les "..." indiquent les parties omises).*

Nous détaillons maintenant ce modèle et la façon dont Thot l'utilise pour permettre la création d'EDs GC.

6.2.2.1 Principe général d'exploitation d'un modèle structurel par Thot

Quand un attribut est déclaré pour un type d'ED dans un modèle structurel, cela signifie qu'un ED de ce type *peut* porter un tel attribut, mais la présence de cet attribut est facultative à moins que la déclaration ne soit précédée de '!'. Dans ce cas, lors de la création de l'ED, l'attribut est créé automatiquement et si aucune valeur par défaut n'indiquée dans le modèle structurel, Thot demande cette valeur à l'utilisateur ; pour donner une valeur à un attribut référence[1], i.e. à un lien hypertexte, l'utilisateur doit sélectionner l'ED qui va être référé par cet attribut.
Par contre, lorsque des sous-parties sont déclarées pour un ED (avec "= Begin ... End"), cela signifie que ces sous-parties *doivent* être remplies, à moins que la déclaration d'une sous-partie ne soit précédée de '?', auquel cas l'utilisateur doit explicitement demander leur création. Si la déclaration n'est pas précédée de '?', lors de la création de l'ED, ses sous-parties sont créées automatiquement par l'éditeur et sont présentées vides.
La figure 6.3 montre un exemple d'ED CGgraph.

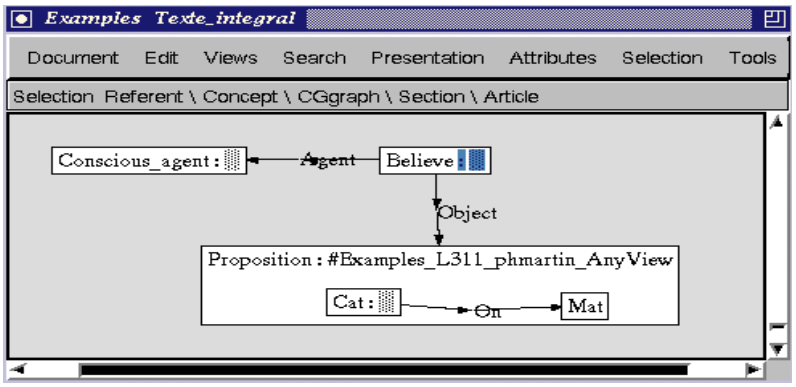
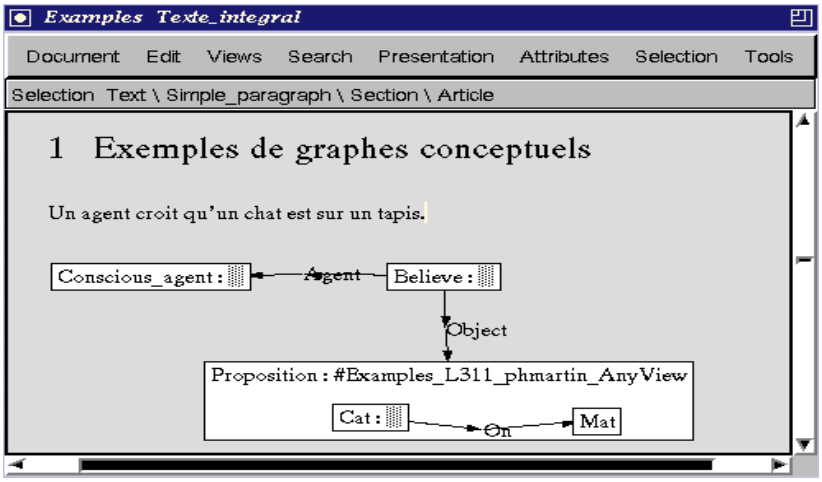


Figure 6.3: *Une fenêtre de l'éditeur Thot montrant un ED CGgraph appartenant au document Examples.*
Un ED Referent est sélectionné. Le témoin de sélection montre ses parents dans la structure : Concept, CGgraph, Section, Article. Les parties grisées représentent des référents non encore remplis. Un référent vide, ou détruit par l'utilisateur (e.g. ici celui de [Mat]), est interprété par CGKAT comme un référent générique. Lorsqu'un ED CGReferent est créé dans un concept, CGKAT génère pour ce concept un référent individuel, e.g. #Examples_L311_phmartin_Anyview.



Note: le curseur est situé à l'intérieur d'un ED Text. Le témoin de sélection montre les parents structurels de cet ED.

Figure 6.4: *Une fenêtre de l'éditeur Thot montrant un ED CGgraph parmi d'autres EDs (ici un paragraphe simple et une section).*

1. Un attribut de type Reference peut permettre la navigation hypertexte. Thot permet de créer un lien hypertexte de deux manières : avec un attribut de type Reference (ou attribut référence) et avec un ED de type Reference.

6.2.2.2 Détails sur la création des EDs GC et de leurs composants

Lorsqu'un ED est sélectionné (un ED déjà existant donc), le menu "Attributes" de l'éditeur Thot liste les attributs qui peuvent être créés et associés à cet ED compte-tenu de son type. Lorsqu'un attribut est choisi, Thot permet de lui affecter une valeur ou de le détruire.

Si des attributs référence doivent obligatoirement être associés à un ED, lorsque de la création de celui-ci, Thot demande à l'utilisateur de pointer les EDs qui vont être référés par ces attributs. Par exemple, les ***EDs Relation*** ont pour attributs imposés OrigNode et DestNode. Après avoir demandé la création d'une relation, l'utilisateur doit donc pointer les deux premiers concepts qu'elle relie. La présentation de la relation peut alors être déterminée par Thot selon les règles du modèle de présentation. Pour les relations non binaires, nous avons défini d'autres attributs référence facultatifs (non présentés dans la figure 6.2). Cependant, comme nous n'utilisons pas les relations non binaires, nous n'avons pas implémenté la répercussion dans CoGITo de la création de ces relations via l'éditeur Thot. Cependant, CGKAT permet également d'écrire des GCs sous une forme linéaire, laquelle permet la création dans CoGITo de GCs contenant des relations non binaires.

Toujours compte-tenu du type de l'ED sélectionné, Thot propose un ***menu d'insertion*** contenant les types d'EDs fils ou frères pouvant être créés autour de lui. Il suffit alors à l'utilisateur de sélectionner un des types. Si le type d'ED choisi est défini comme un choix entre plusieurs autres types d'EDs (comme le sont par exemple les types CG et ConceptOrRelation), l'utilisateur doit également effectuer ce choix. Une ***inclusion d'ED*** peut se créer de manière similaire : un menu d'inclusion présente les types d'EDs fils ou frères qui peuvent être créés autour de l'ED sélectionné, et lorsqu'un type est choisi, l'utilisateur doit pointer avec la souris l'ED source de l'inclusion (dans le même document ou pas). Une inclusion d'un ED peut être créée partout où la création d'un ED de ce type est autorisée par le modèle structurel.

L'attribut énuméré DefKind sur une ***définition de type*** doit être valué pour spécifier de quelle sorte de définition il s'agit. Par défaut, c'est une définition de type de concept par conditions typiques (schéma prototypique). Si la valeur No_more_a_definition est choisi, CGKAT transforme la définition en un CGgraph, et inversement lorsque l'attribut ChangeIntoDefinition d'un CGgraph est positionné à Yes.

6.2.2.3 Mise à jour de la base de COGITo

Toutes les créations, modifications et destructions des EDs constituant un CGgraph ou une définition de type sont immédiatement répercutées par CGKAT dans CoGITo. Si une opération est refusée par CoGITo, elle est alors également refusée dans l'éditeur Thot : l'action d'édition correspondante est refusée. Il s'agit là d'un contrôle sémantique sur chaque opération (e.g. contrôle du respect des signatures des relations et de la relation de conformité) qui s'ajoute au contrôle structurel effectué par Thot.

Par contre, ***les "concepts uniques", i.e. les EDs SingleConcept, ne sont pas répercutés dans la base de CoGITo lors de leur création*** (*mais si un concept unique contient un GC dans sa partie référent*[1]*, ce GC est bien créé dans CoGITo*). En effet, les concepts uniques n'ont pas un rôle de description (hormis via les GCs qu'ils peuvent contenir dans leur partie référent), mais sont

1. Si tel est le cas, rappelons que dans CGKAT, ce concept unique doit être de type Proposition.

notamment destinés à être "*réutilisés" via le mécanisme d'inclusion* dans divers GCs de façon à permettre les recherches par navigation hypertexte entre ces GCs[1]. Il serait donc inutile d'encombrer par des concepts uniques la base de graphes de CoGITo et par voie de conséquence les résultats des requêtes conceptuelles. Un contrôle sémantique est effectué par CGKAT sur les concepts uniques lorsqu'ils sont créés ou modifiés, sans qu'ils soient ajoutés à la base de CoGITo. Mais bien sûr, lorsqu'un concept unique est "réutilisé" dans un GC, il est asserté dans la base de CoGITo avec ce GC (si ce concept unique contient un graphe dans sa partie référent, ce graphe n'est pas asserté puisqu'il l'a déjà été ; cependant, le système de requête de CGKAT prend en compte le nouvel emboîtement de ce graphe).

6.2.2.4 Génération des noms de graphes

Nous avons expliqué en section 4.3.2.5 pourquoi dans CGKAT, seul un concept de type Proposition (ou d'un de ses sous-types) peut admettre un graphe dans sa partie référent, et pourquoi, si ce concept a un référent individuel, l'étiquette de ce référent doit correspondre au nom du graphe emboîté. Dans CGKAT, quand l'utilisateur crée un graphe via un ED CGgraph (graphe isolé) ou un ED CGReferent (graphe emboîté dans un concept), le nom du graphe est automatiquement généré. Ce nom est unique car il est construit à partir du nom du document et de l'identificateur de l'ED CGgraph ou CGReferent dans ce document. De plus, afin d'inclure plus d'informations dans ce nom, l'identificateur du créateur du graphe et le nom du point de vue pris en compte pour créer le graphe (par défaut "AnyView") figurent également dans le nom du graphe. Le graphe est créé sous ce nom dans la base de CoGITo lorsqu'il est rempli avec un premier concept. Si le graphe est créé en partie référent d'un concept, le référent de ce concept devient un référent individuel dont l'étiquette correspond au nom du graphe. De plus, dans ce cas, le nom du graphe est stocké dans un attribut CGName porté par le concept. Si le graphe est isolé, le nom du graphe est stocké dans un attribut CGName porté par l'ED CGgraph. L'utilisateur peut modifier le nom du graphe en modifiant le contenu de l'attribut CGName (si le graphe est emboîté dans un concept, le référent individuel de ce concept est modifié). C'est par le contenu d'un tel attribut que CGKAT peut retrouver depuis un ED CGgraph ou CGReferent, le graphe qui lui correspond dans la base de CoGITo.

6.2.2.5 Création des définitions de types

Lorsqu'un utilisateur de CGKAT crée ou modifie le corps d'une définition de type, c'est une lambda-abstraction qui est créée ou modifiée dans CoGITo (un nom unique est également généré par CGKAT pour chaque lambda-abstraction, et ce nom peut être changé via un attribut CGName). Lorsque l'utilisateur a terminé sa création ou sa modification, il doit le signaler via l'attribut Completed. CGKAT utilise alors la lambda-abstraction créée ou modifiée pour créer ou modifier le type qu'elle définit.

Lorsqu'une lambda-abstraction définit un type de concept, CGKAT définit ce type comme un sous-type du type du concept porteur de la variable paramètre de la définition. Ainsi par exemple, avec la définition suivante, Lazzy_cat est créé comme sous-type de Cat :
NC for Lazzy_cat (x) are [Cat : *x].

---

1. Dans CGKAT, nous encourageons les utilisateurs à construire des GCs en "réutilisant" des concepts uniques (EDs SingleConcept), i.e. en les "incluant", plutôt qu'en créant de nouvelles copies de ces concepts. Cela a en effet deux avantages :
a) D'une part, la modification d'un concept unique inclus par différents graphes est automatiquement répercutée dans ces graphes (dans les documents, la répercussion est gérée par Thot ; dans la base de CoGITo, la répercussion est gérée par CGKAT).
b) D'autre part, les liens d'inclusion étant également des liens hypertextes, à partir d'une inclusion de concept unique dans un GC, le concept source peut être directement retrouvé, et à partir de ce concept, tous les GCs qui l'incluent peuvent être retrouvés un par un. Si un graphe ainsi trouvé contient d'autres inclusions de concepts, ces inclusions sont le point de départ vers d'autres GCs, et l'utilisateur peut ainsi parcourir la base de GC en GC, au vu des diverses relations conceptuelles qui relient les concepts de ces GCs.
Nous présenterons plus loin que la réutilisation d'un concept unique, i.e. la création d'une inclusion de ce concept s'effectue par la création d'un *ED ConceptInclusion.*

Jusqu'à présent, nous avons préféré ne pas imposer à l'utilisateur que dans les définitions de types de concepts *par conditions suffisantes ou typiques*, le type du concept porteur de la variable paramètre soit le même que le type du concept défini. L'utilisateur peut donc actuellement écrire :
TC for Lazzy_cat (x) are [LazzyCat:*x]→(On)→[Mat]. mais également :
TC for Lazzy_cat (x) are [Cat:*x]→(On)→[Mat]. Dans ce dernier cas, comme indiqué dans le paragraphe précédent, si Lazzy_cat n'était pas déjà un sous-type de Cat, il le devient.

CGKAT vérifie lorsqu'un type devient sous-type d'un autre type (lors de la déclaration d'un type ou lorsqu'une définition lui est ajoutée) que cela n'entraîne pas de cycle dans la hiérarchie (i.e. qu'un ordre partiel est conservé), ni ne crée un sous-type commun à des types exclusifs.

La figure suivante (6.5) illustre la représentation de connaissances sous forme de définitions de types dans un document Thot. Cette figure montre le début du document "KADS1" qui représente, documente et organise les modèles que la méthodologie KADS-I conseille de construire ainsi que les modèles génériques qu'elle propose pour aider à construire un modèle d'expertise. La table des matières de ce document permet de retrouver rapidement (par navigation hypertexte) les modèles désirés. L'utilisateur peut également naviguer de section en section ou de paragraphe en paragraphe. Des requêtes lexicales (recherche de chaînes de caractères) ou conceptuelles (par projection) peuvent également être utilisées pour trouver les informations recherchées.

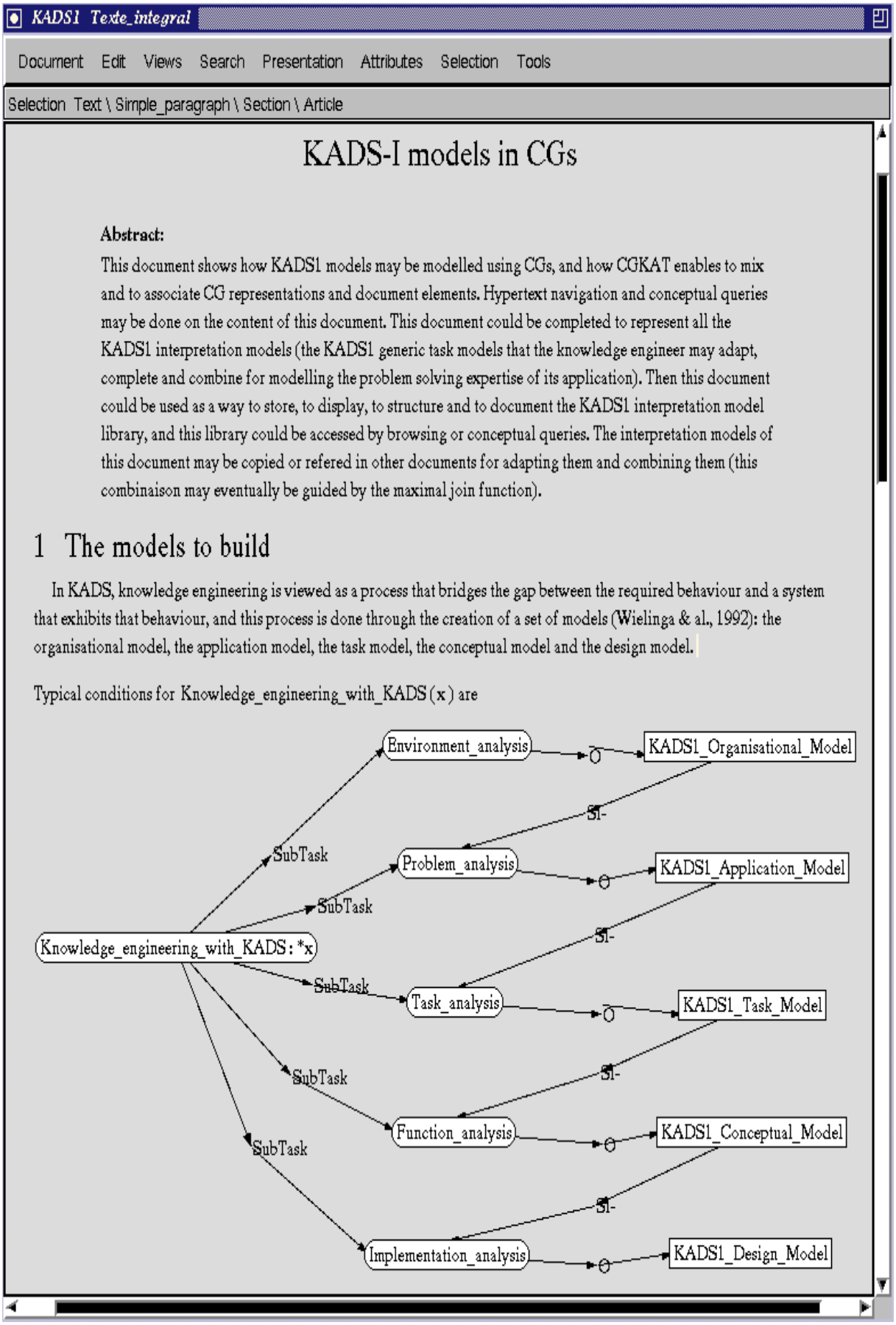


Figure 6.5: *Le début d'un document représentant, documentant et organisant des modèles de KADS-I.*

6.2.2.6 Gestion des inclusions

Dans Thot, une inclusion ne peut être modifiée directement et ne peut être pointée par un attribut référence. Aussi, une relation ne peut être connectée à une inclusion de concept. Nous avons donc introduit le type d'ED ConceptInclusion qui encapsule une inclusion de concept : un tel ED a les mêmes attributs qu'un ED Concept (identificateur dans le CGgraph dont il peut faire partie, commentaire, pointeur vers un ED qu'il représente, etc.) mais il n'a qu'une sous-partie, une inclusion de concept. Dans CoGITo, ce type d'élément est répercuté comme un concept normal, mais pour effectuer cette répercussion, le code est différent de celui pour les EDs Concept puisque les structures de ces deux types d'EDs sont différentes. Thot assure la mise à jour automatique, dans les documents, des inclusions lorsque leurs sources sont modifiées. CGKAT assure cette mise à jour dans la base de GCs : lorsqu'un ED Concept est modifié, CGKAT doit modifier en conséquence dans la base tous les graphes et lambda-abstractions qui dans les documents incluent les concepts modifiés.

6.2.2.7 Chargement des graphes lors de l'ouverture des documents

Lorsqu'un document est ouvert par l'utilisateur, s'il contient des EDs CGgraph ou des définitions de types, les représentations correspondantes sont créées dans CoGITo. Quand le document est fermé, ces représentations sont détruites dans CoGITo. Ainsi la base de GCs dans CoGITo ne contient à un instant donné que les GCs contenus dans les documents ouverts à cet instant, plus éventuellement des GCs directement chargés à partir de fichiers de sauvegarde. Les documents ouverts constituent donc bien une interface sur le contenu de la base. L'espace de recherche de connaissances ou d'informations via des requêtes conceptuelles peut donc être contrôlé en ouvrant plus ou moins de documents. L'ouverture peut se faire manuellement ou bien via un langage de commandes (voir section 6.2.5).

6.2.3 Association de méta-informations à un graphe conceptuel

Comme pour les types, il nous a semblé important de permettre à l'utilisateur d'associer des méta-informations à des GCs, i.e. des couples attribut/valeur de son choix (l'attribut peut être une chaîne quelconque mais comme nous allons le voir, certains attributs sont prédéfinis dans CGKAT et une valeur leur est automatiquement associée). Par exemple, un couple attribut/valeur peut être utilisé pour représenter l'utilisateur créateur du GC, la date de création du GC, le point de vue qui a été pris sur les EDs sources pour représenter leur contenu dans ce GC (i.e. quels aspects particuliers ou encore quels types de connaissances ont été choisis), les auteurs des EDs sources considérés (i.e. l'auteur source de l'information représentée), le contexte qui permet de mieux comprendre l'information représentée, ou encore le contexte d'utilisation de cette information. Tout ceci peut bien sûr être représenté en utilisant des GCs emboîtés, mais nous avons voulu offrir un moyen de représentation alternatif que l'utilisateur peut considérer comme plus ergonomique pour de tels cas (cela est précisé plus loin ; notons que le gain se fait au dépend de la précision de la représentation).

Comme pour les types, dans CoGITo les méta-informations des GCs sont stockées dans les champs commentaires associés aux graphes et lambda-abstractions. Ces champs correspondent aux attributs Comment des EDs CGgraph et définitions de types et sont donc aisément accessibles et éditables via l'éditeur Thot. Nous avons donné en section 5.2.2.2 à la page 111 une définition de la spécialisation entre méta-informations. Celle entre "GCSs avec méta-informations" en découle :
soient (G1,M1) et (G2,M2), deux GCSs avec des méta-informations, (G1,M1) spécialise (G2,M2) si G1 spécialise G2 et si M1 spécialise M2 (c'est toujours le cas si M2 est vide).
Ainsi dans CGKAT, une requête conceptuelle peut être posée avec un GCS et en indiquant au besoin des contraintes supplémentaires sous forme de méta-informations. Voici un exemple de recherche de spécialisations d'un graphe requête avec des méta-informations, suivi d'une réponse possible :

? [Cat] with {user:Acacia_member; document:Examples;}
[Cat]→(On)→[Mat] with user:phmartin; document:Examples;

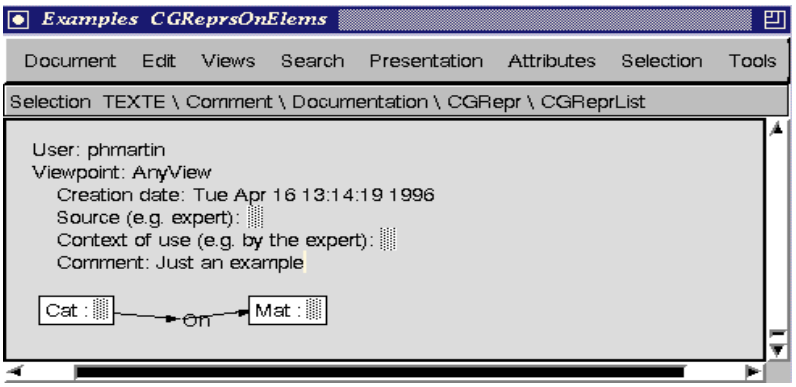
L'usage de méta-informations peut être considéré dans certains cas comme plus ergonomique que l'usage de GCs emboîtés, car
1) des opérations sont épargnées à l'utilisateur dans la représentation des connaissances puis leur recherche par requête (si l'utilisateur ne peut associer des méta-informations à un GC, il doit créer un graphe emboîtant ce GC), et
2) la présentation des GCs est plus concise (pas de graphes emboîtants), ce qui permet à l'utilisateur de visualiser simultanément plus de GCs et ainsi de les comparer rapidement (e.g. visualisation des résultats de requêtes ou visualisation de graphes lors des recherches par navigation).

Par ailleurs, nous avons créé un type d'ED destiné à rendre plus ergonomique la saisie et la présentation de certaines méta-informations généralement utiles pour une représentation. Il s'agit du type CGRepr dont le modèle structurel est donné dans la figure suivante. Notre modèle de présentation et notre modèle d'interface pour ce type d'ED se trouvent en annexe 4.

```
STRUCTURE CGRepr;   DEFPRES CGReprP;    { Structural model for a (DE of type) CGRepr }
STRUCT
   CGRepr = BEGIN
                  User = TEXT;  { Author of the CG }              Viewpoint = TEXT;
                  Documentation = BEGIN
                                      CreationDate  = TEXT;     SourceAuthor = TEXT;
                                      ContextOfUse = TEXT;      Comment = TEXT;
                                  END;
               ? GeneratedCG = CG;  { representation proposed by CGKAT to the user }
                  TheRepr = CG;  { the representation chosen by the user }
              END;
EXCEPT   { exceptions to standard edition rules for DEs }
   TheRepr : NoSelect,NoMove,NoResize;   { this DE won't be selected, moved and resized → instead }
END;                                     {    its structural parent will be selected, moved or resized }
```

Figure 6.6: *Modèle structurel pour un CGRepr (représentation avec un GC et certaines méta-informations).*

La figure suivante illustre un ED CGRepr. Un tel ED peut avoir été créé pour représenter une ou plusieurs phrases comme "Un chat est sur un tapis", ou encore une photo d'un chat sur un tapis.



Note: le curseur est situé à l'intérieur d'un ED Text. Le témoin de sélection montre les parents structurels de cet ED.

Figure 6.7: *Une fenêtre de l'éditeur Thot montrant un ED CGRepr*

Lorsque l'utilisateur crée un ED CGRepr, les EDs User, CreationDate et Viewpoint sont automatiquement remplis par CGKAT. L'ED Viewpoint est rempli par défaut avec la valeur AnyView qui est le supertype de tous les types de points de vue et qui donc ne représente aucune vue particulière. L'utilisateur peut préciser en changeant le contenu de cet ED s'il utilise un point de vue particulier. Lorsque les EDs User, ViewPoint et ceux de Documentation sont créés ou modifiés par l'utilisateur, leur contenu est répercuté sous forme de méta-informations dans l'attribut Comment associé à l'ED fils de TheRepr (cet ED doit d'après le modèle structurel être un CGgraph, un concept unique ou une définition de type). La date de création n'est actuellement pas répercutée car son utilisation pour sélectionner des GCs dans la base nécessiterait des opérations plus élaborées que des comparaisons de chaînes de caractères.

6.2.4 Construction d'une hiérarchie via un élément de document

La bibliothèque de Thot fournit des modèles structurels et de présentation permettant de construire un arbre dont les noeuds peuvent être des éléments quelconques. Un ED Arbre peut être construit et présenté de diverses façons : sous forme de liste indentée, sous forme d'un arbre vertical ou d'un arbre horizontal. Les règles de présentation qui permettent cette disposition automatique des noeuds suivant le format choisi, s'appuient sur la structure abstraite de l'ED Arbre qui est isomorphe à la structure de l'arbre qu'il représente : aux liens hiérarchiques entre les noeuds correspondent des liens structurels (i.e. d'emboîtement) entre les EDs qui représentent ces noeuds.

Un ED Arbre pourrait donc être exploité pour permettre l'affichage de hiérarchies de types ou de GCs. Si une hiérarchie contient trop de noeuds pour être affichée intégralement, son exploration pourrait être permise via l'affichage successif de sous-parties, selon des techniques similaires à celles actuellement utilisées dans CGKAT pour les hiérarchies de types de concepts ou de relations. Cependant il s'agirait là de générations de (parties de) documents alors qu'actuellement CGKAT effectue des générations de (parties de) menus Motif. La programmation en serait plus difficile (et c'est pourquoi nous avons dans un premier temps utilisé des menus), mais la génération d'EDs Arbre pour présenter des hiérarchies aurait les avantages liés aux documents structurés :
1) la présentation des EDs Arbre peut être choisie par l'utilisateur (et un modèle de présentation peut éventuellement être adapté par l'utilisateur),
2) les noeuds de ces arbres se sont pas limités à du texte et pourraient par exemple être des EDs CGgraph,
3) les ED Arbre peuvent être accompagnés d'autres EDs qui les documentent, et les noeuds peuvent être connectés par des liens hypertextes à d'autres EDs (e.g. un type peut être connecté à une définition de type ou bien à un ensemble de mots dont il représente le sens),
4) les commandes de l'éditeur peuvent être utilisées pour visualiser, copier, modifier, compléter et sauvegarder les informations générées.

Cependant, si un élément d'une hiérarchie a plusieurs parents directs, cet élément apparaîtra dans plusieurs branches de l'ED Arbre utilisé pour afficher cette hiérarchie (comme dans l'actuel système de liste indentée qu'utilise CGKAT). Nous avons donc tenté de définir un modèle structurel et un modèle de présentation pour un ED "Hierarchy" qui permette un affichage automatique d'une hiérarchie quelconque plus adéquat qu'un ED Arbre. Mais compte-tenu des caractéristiques du langage P (le langage de définition de modèles de présentation offert par Thot), afin que des règles de présentation puissent placer automatiquement des noeuds les uns par rapport aux autres, *il nous fallait conserver une structure similaire à celle utilisée pour un ED Arbre (nous ne pouvions donc définir un ED Hierarchy de manière similaire à un ED GC c'est-à-dire comme un graphe)*[1].
Nous avons alors voulu faire en sorte que les noeuds "clones", c'est-à-dire les noeuds représentant le même élément mais apparaissant dans diverses branches de l'ED Hierarchy, soient toujours affichés au même endroit, même lorsque l'un d'eux est déplacé par l'utilisateur. Pour cela,
1) dans le modèle structurel d'un ED hierarchy, nous avons introduit (par rapport à celui d'un ED Arbre), un attribut RefClone qui permet à un noeud de pointer sur un noeud "clone" (voir figure 6.8) ;
2) dans le modèle de présentation d'un ED hierarchy, nous avons ajouté une règle de présentation spécifiant qu'un noeud porteur d'un attribut RefClone devait avoir la même position que le noeud pointé. Malheureusement, nous n'avons pu éviter les conflits entre cette nouvelle règle et les précédentes ; aussi, lorsque des noeuds clones existent dans un ED Hierarchy, la présentation de cet ED est rarement celle escomptée. Nous n'avons pas obtenu de meilleurs résultats avec d'autres modèles structurels et de présentation.

1. Le langage P est un langage de boîtes : il permet d'écrire des règles simples pour positionner chaque boîte affichant un ED par rapport à certaines autres boîtes (boîte de l'ED parent, boîte d'un ED frère ou boîte d'un ED référé par un attribut) mais ne permet pas d'effectuer des calculs complexes sur la position des boîtes les unes par rapport aux autres.

Toutefois, comme le montre la figure 6.9 (qui contient très peu d'éléments), le calcul d'une présentation harmonieuse pour une hiérarchie est une opération difficile. Même si notre tentative avait abouti, la présentation d'une hiérarchie quelconque via l'ED Hierarchy n'aurait été satisfaisante que dans de rares cas. Le langage P est un langage de boîtes ne permet pas d'effectuer des calculs complexes sur la position des boîtes d'EDs les unes par rapport aux autres. Aussi, pour présenter une hiérarchie quelconque autrement que sous une forme d'arbre, il faudrait définir un ED Hierarchy de manière similaire à un ED GC c'est-à-dire comme un graphe, et calculer la position des noeuds par programme (et également, lorsque des lignes brisées doivent être utilisées pour joindre des noeuds, les positions des points principaux par lesquels doivent passer ces lignes). Le langage P ne permet pas non plus de prendre en compte le contenu des noeuds (seule la structure peut être exploitée). Pour cela, la présentation doit également être calculée par programme.

```
STRUCTURE Hierarchy;   DEFPRES HierarchyP;    { Structural model for a (DE of type) Hierarchy }
STRUCT
   Hierarchy (ATTR !Presentation_form = Vertical, Horizontal, Indented_List;
                        NodeFrame = Without, Rectangular, RectangularShaded, Rounded, RoundedShaded;
                        AboutNodeWidth = Fixed,Variable )
       = Sub_Hierarchy with Presentation_form?=Horizontal, AboutNodeWidth?=Variable, NodeFrame?=Without;

   Sub_Hierarchy (ATTR  NodeFrame)
       = BEGIN   Node (ATTR RefClone=REFERENCE(Node)) = LIST OF (UNIT);   {strings and images are
units}
                ? Descendants = LIST OF (Sub_Hierarchy);                {any ED type may be declared as a unit}
         END;

EXCEPT   { exceptions to standard edition rules for DEs }
   Node : NoMove;          Descendants: NoMove,NoResize;
END;
```

Figure 6.8: *Modèle structurel pour un ED Hierarchy et, en ôtant l'attribut RefClone, pour un ED Arbre.*

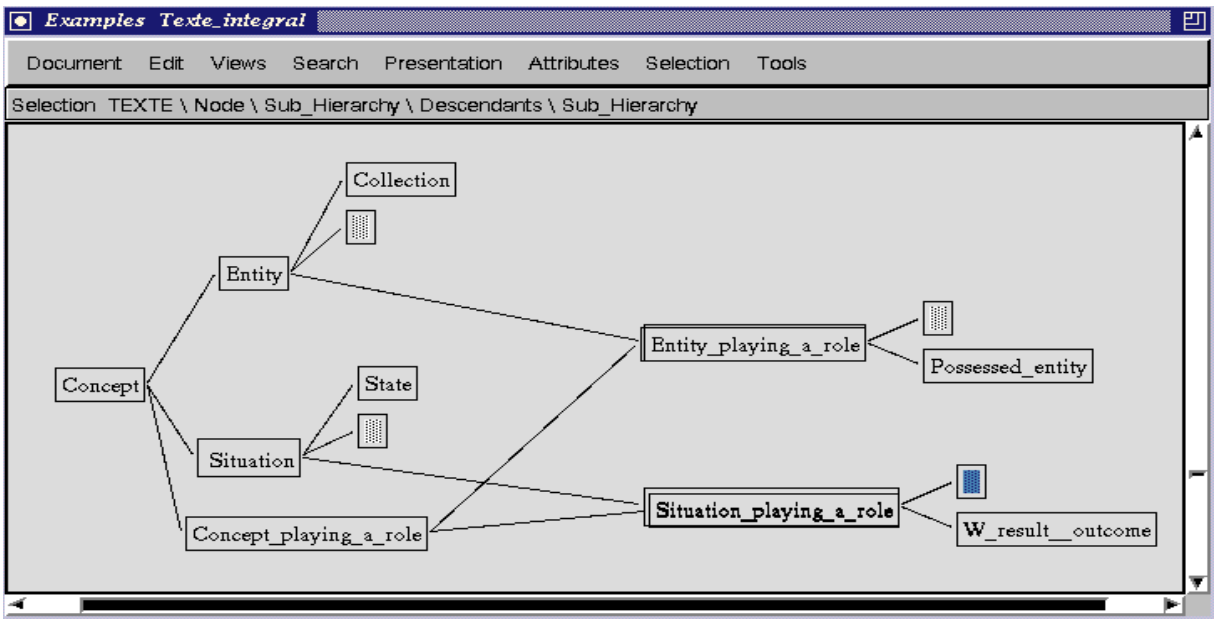


Figure 6.9: *Une fenêtre de l'éditeur Thot montrant une hiérarchie de types de concepts.*
Le placement des noeuds "Entity_playing_a_role" et "Situation_playing_a_role" a été fait à la main.

6.2.5 Utilisation d'un langage de commandes via un élément de document

Nous avons créé le type d'ED CGRequests pour permettre à un utilisateur de CGKAT d'écrire des commandes à l'intérieur d'un document structuré et d'avoir leurs réponses à l'intérieur de ce document. Ainsi,
1) les commandes et surtout leurs résultats peuvent être stockés à l'intérieur de documents et utiliser, outre du texte, des EDs structurés (e.g. des EDs CGgraph) ;
2) il s'agit d'un moyen simple pour l'utilisateur de faire exécuter des commandes complexes, soit directement, soit automatiquement lors de chaque ouverture du document qui les contient. Cela permet de générer des documents ou des parties de documents ;
3) ces parties de document peuvent lors de leur génération être liées à d'autres EDs par des liens structurels ou hypertextes ou par le mécanisme d'inclusion ;
4) l'utilisateur peut compléter, modifier ou organiser manuellement ces parties de documents en utilisant les moyens offerts par l'éditeur Thot.

La figure 6.10 montre le modèle structurel d'un ED CGRequests (les modèles de présentation et d'interface correspondants sont décrits en annexe 4). Cet ED peut inclure des EDs RequestAnswer et peut fonctionner comme un système de question/réponse. Lorsque l'utilisateur crée un ED de ce type, un seul ED RequestAnswer est proposé ; l'utilisateur peut remplir l'ED Request avec une commande puis le sélectionner ; CGKAT exécute alors la commande et en réponse 1) remplit l'ED Answers et 2) crée un nouvel ED RequestAnswer permettant à l'utilisateur de poser une nouvelle requête (voir figure 6.11).

Par ailleurs, au moment où un document contenant des EDs CGRequests est ouvert, les commandes contenues dans ces EDs sont exécutées ; pour chaque ED RequestAnswer, si l'ED Answers est présent et vide, il est rempli avec les réponses à la commande concernée, sinon (i.e. s'il a été détruit ou s'il est déjà rempli) les réponses ne sont pas intégrées dans le document mais affichées dans une fenêtre "shell".

```
STRUCTURE CGRequests;   DEFPRES CGRequestsP;    { Structural model for a (DE of type) CGRequests }
STRUCT
   CGRequests = LIST OF (RequestAnswer);

   RequestAnswer  = BEGIN  Request = UnitList;
                           Answers = LIST OF (Answer=UnitList);
                    END;

   UnitList = LIST OF (AnyElement=UNIT);

UNITS    { Declaration of the kinds of EDs which may be units
           (in addition to strings, images, graphics elements and symbols) }
   StructuredElement = NATURE;      { Any kind of structured DE declared as reusable in documents }
END;
```

Figure 6.10: *Modèle structurel pour un ED CGRequests.*

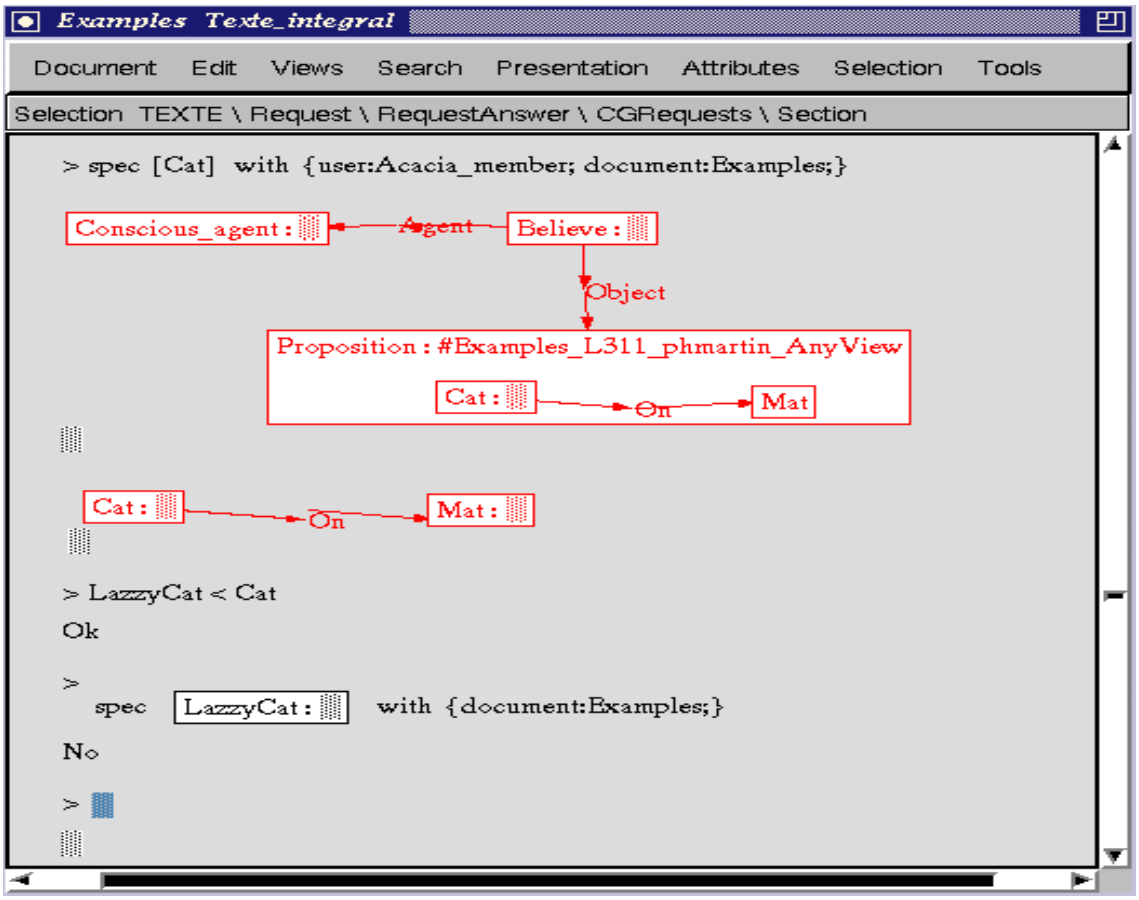


Figure 6.11: *Une fenêtre de l'éditeur Thot montrant un ED CGRequests.*
Cet ED CGRequest contient deux recherches des spécialisations d'un GC et une déclaration d'un type de concept. Pour afficher les GCs résultats de la première recherche, CGKAT a "inclus" les EDs avec lesquels ces GCs ont été construits.

Cependant, lorsqu'un document contenant des EDs CGRequests est fermé, CGKAT n'effectue actuellement rien de spécial dans CoGITo. L'utilisateur doit donc savoir que si certaines des commandes des EDs CGRequests ont affecté la base de CoGITo (e.g. création de GCs ou définitions de types), cette base conservera ces modifications lorsque le document sera fermé. Par contre, les graphes et les définitions de types créés via des EDs GC et non inclus dans des ED CGRequests, sont ôtés de la base de CoGITo lorsque le document qui contient ces EDs est fermé. Ces deux types d'EDs ont donc des comportements complémentaires. Notons également que CGKAT met à la disposition de l'utilisateur des commandes de destruction de types et de GCs dans la base de CoGITo, et permet de coupler des commandes de recherche à des commandes de destruction.

6.2.5.1 Le langage de commandes de CGKAT

L'annexe 5 détaille le langage de commandes qu'offre CGKAT pour charger, rechercher, créer, modifier, détruire et ordonner ou bien assembler les types et les graphes de la base de CoGITo.

Nous avons permis l'appel de ces commandes depuis les EDs CGRequests, mais également depuis le "shell" (l'interpréteur de commandes standard d'Unix) à condition qu'un processus CGKAT ait été préalablement lancé. Dans ce dernier cas, les résultats sont affichés sur la sortie standard du processus dans lequel la commande a été appelée. Inversement on peut appeler dans CGKAT, depuis un ED CGRequest, toutes les commandes qu'on peut appeler depuis le shell (les réponses sont alors affichées dans un ED Answers de l'ED CGRequest). Ces fonctionnalités et leur combinaison ont plusieurs avantages :
1) les fonctions de CGKAT peuvent être utilisées depuis n'importe quelle application ;
2) n'importe quelle application peut être utilisée depuis CGKAT ;
3) les instructions de contrôle du shell (tests, boucles, "pipes", etc.) peuvent être utilisées depuis CGKAT, ou depuis le shell afin de séquencer et de combiner les fonctions de CGKAT ou d'autres applications ;
4) des fichiers scripts shell peuvent être utilisés pour stocker certaines suites de commandes ; depuis CGKAT des fichiers scripts CGKAT peuvent également être utilisés. Les scripts sont intéressants pour simplifier le travail du cogniticien (tests, simulations, recherches) ou pour stocker des procédures permettant de répondre à des requêtes habituelles ("FAQs").

Nous détaillerons dans le chapitre 7 (Implémentation) comment nous avons permis ces fonctionnalités. Décrivons en le principe avec quelques exemples. Lorsqu'une commande du shell est appelée depuis CGKAT, elle doit être précédée de "sh". Lorsqu'une commande de CGKAT est appelée depuis le shell, elle doit être précédée de "ck".

```
> loadDoc Examples.PIV IntervJL.PIV        //Appel depuis CGKAT d'une commande
                                           // d'ouverture de documents Thot
$ ck loadDoc *.PIV            //Appel depuis le shell de cette commande en utilisant un filtrage sur
                              //les documents dans le répertoire courant
> sh ck loadDoc *.PIV         //La même commande appelée depuis CGKAT (le passage par le shell
                              //est nécessaire pour bénéficier du filtrage)
> sh if test -f Examples.PIV;          //Test de l'existence d'un document avant de l'ouvrir
     then loadDoc *.PIV;               //(il s'agit d'un exemple; en pratique, cela est inutile)
     fi
> sh ck spec [Cat] | ck maxjoin | ck linearForm      //combinaison via le "pipe" de trois
     // commandes de CGKAT : 1) "spec" ou "?", qui recherche des spécialisations d'un GC,
     //                      2) "maxjoin" qui effectue une jointure maximale sur les GCs qui
     // lui sont passés en paramètre ou dans son entrée standard comme c'est le cas ici
     //(et dans ce cas, cette commande affiche sur sa sortie standard non pas une forme linéaire
     // du GC résultant mais le nom de ce GC, ce qui permet à d'autres commandes de réutiliser
     // ce résultat),          et 3) "linearForm" qui à partir d'un nom de GC donné en paramètre
     // ou dans son entrée standard, retourne une forme linéaire de ce GC (et ici, la réponse est
     // affichée par CGKAT dans un ED Answer).
```

6.2.6 Comparaison avec d'autres travaux

Thot a été utilisé comme un éditeur syntaxique pour les langages graphiques Peplom et Argos (cf. (Francou, 1993) et (Schaar, 1994)). Argos est un langage synchrone dédié à la spécification et à la programmation de systèmes réactifs, et dont les composants élémentaires sont des automates. Schaar a développé un environnement de développement pour ce langage, et a préféré pour cela utiliser Thot plutôt qu'un générateur d'interface ou que Unidraw (Vilissides, 1990), une boîte à outils orientée objet de développement d'éditeurs graphiques, car les structures abstraites que lui fournissaient Thot correspondaient aux représentations internes dont il avait besoin pour gérer les objets de son langage[1]. Il n'a donc pas eu à gérer comme nous l'avons fait dans CGKAT, des traductions (et ainsi un maintien constant de la cohérence) entre les représentations dans Thot et d'autres sortes de représentations plus adéquates pour exploiter le formalisme des objets représentés (dans notre cas, celles de CoGITo). Par ailleurs, Schaar n'étudie pas les liens que l'utilisateur peut vouloir créer entre des représentations dans le langage Argos et d'autres EDs. Nous présenterons dans la prochaine section quels types de liens nous avons défini pour permettre l'indexation d'EDs par des représentations dans le formalisme des GCs, et comment ces liens peuvent être exploités pour la recherche d'informations.

Hurwitz & Rich (1993) ont écrit en SGML un modèle structurel pour un GC, mais ils ne semblent pas l'avoir exploité pour créer un éditeur syntaxique de GCs.

Si l'on ne retient de CGKAT que sa fonction d'éditeur de GCs, comparé aux autres éditeurs graphiques de GCs, e.g. GRIT (Leane, 1993), CGKAT hérite en plus des fonctionnalités d'un éditeur dédié à la gestion (création, présentation) d'éléments de documents, et que nous avons rappelées en section 6.2.1. Dans GRIT, comme dans la plupart des autres éditeurs graphiques de GCs actuels, aucune documentation (aucun commentaire) ne peut être associée aux GCs ni aux types de concepts ou de relations. Toutefois, GRIT permet un affichage automatique des noeuds d'une hiérarchie quelconque, qui selon l'auteur semble satisfaisant pour de petites hiérarchies (Leane, 1993).

Les plates-formes de développement Peirce et CGKEE intègrent un langage de commandes pour construire et manipuler des GCs. Dans CGKAT, l'utilisateur dispose en plus des facilités de contrôle offertes par le shell (tests, boucles, "pipes", scripts, variables, filtrage, etc.) et il peut combiner, depuis le shell ou depuis CGKAT, les commandes de CGKAT avec d'autres commandes accessibles depuis le shell. Dans le projet OPERA, auteur de Thot, des travaux s'effectuent sur un langage de commandes d'édition de documents structurés (Bois & Richy, 1996). Ces commandes pourraient être intégrées à notre langage de commandes afin par exemple de permettre à l'utilisateur de modifier des EDs en fonction du contenu de certains GCs.

Notre introduction de "méta-informations" pour les types et les GCs est également originale dans le cadre des GCs. La relation de spécialisation sur les méta-informations se rapproche de celle proposée par Esch & Levinson (1995) pour les référents multiples dans un concept (cf. section 4.3.1) : la différence est que les méta-informations sont des couples attribut/valeur et non des listes ordonnées de valeurs. Les méta-informations dans CGKAT devrait faciliter la gestion des points de vue multiples dans une base de GCs : multi-experts, multi-utilisateurs, multi-domaines, multi-fonctions, etc.

Nous avons noté qu'une application telle que Alliance (Decouchant, 1995) permet une édition coopérative sur des documents via plusieurs éditeurs Thot. On peut supposer que lorsqu'un document partagé par plusieurs éditeurs (i.e. par plusieurs utilisateurs) est modifié dans l'un d'eux, chaque éditeur a sensiblement le même comportement que si la modification avait été faite par son utilisateur. Dans ce cas, il serait intéressant de vérifier que seules des modifications minimes seraient à apporter au code de CGKAT pour l'adapter à une telle édition coopérative. En effet, comme chaque processus CGKAT est constitué d'un éditeur Thot et d'un gestionnaire de base de GCs (il y a donc une base de GCs par processus), les modifications effectuées sur un ED CGgraph, TypeDefinition ou CGRepr dans un document partagé, seront répercutées dans toutes les bases des processus CGKAT partageant le document.

1. De plus, une fois les modèles structurels et de présentation définis pour ces objets, Thot le déchargeait du contrôle des aspects structurels du langage et lui fournissait un système de gestion de vues intéressant pour son application.

6.3 Indexation d'éléments de documents par des connaissances

Il y a deux façons d'organiser des EDs : 1) par des liens structurels ou hypertextes reliant directement les EDs, 2) via l'organisation de descripteurs indexant les EDs. Même typés, les liens hypertextes véhiculent peu de sémantique (à moins que les types de relations ne soient pas prédéfinis dans le système mais puissent être définis par l'utilisateur par rapport à d'autres types). Les descripteurs offrent beaucoup plus de possibilités puisque 1) ils permettent de *représenter* certaines caractéristiques des EDs et 2) ils peuvent être organisés de façon complexe et incrémentale. Plus les descripteurs sont précis et organisés, plus les utilisateurs auront des chances de retrouver les informations qu'ils recherchent. L'organisation doit permettre aux utilisateurs de poser des requêtes à un niveau d'abstraction suffisant et d'être guidé dans sa navigation par la sémantique des descripteurs et des liens qui les relient. Comme l'a montré Bernstein (1990), la sémantique des liens réduit le problème de désorientation lié à la navigation.

Aussi, comme nous l'avons montré dans les sections 3.1.2.3, 3.1.2.4 et 3.2.2.2, les systèmes de RI s'orientent de plus en plus vers une indexation fine des informations, i.e. vers une représentation sémantique et formelle ou semi-formelle de leur contenu. On peut s'étonner que cette tendance ait été aussi tardive et ne soit pas plus prononcée. Le problème est que la création de représentations fines est une opération assez longue et rébarbative pour l'utilisateur et par ailleurs difficilement automatisable. Ce sont donc les insuffisances des représentations simples qui poussent les concepteurs des systèmes de RI à introduire progressivement des représentations plus formelles et plus complètes. Nous avons choisi d'accentuer cette tendance en adoptant pour les descripteurs, un formalisme dédié à la représentation et à l'organisation de connaissances, mais général et d'un abord relativement intuitif. Cette solution est naturelle dans un contexte d'acquisition des connaissances où les descripteurs des EDs vont souvent correspondre aux connaissances qui sont extraites de ces EDs ; les liens entre EDs et connaissances sont alors utiles aussi bien pour indexer les EDs que pour documenter les connaissances. Voici quelques exemples de descripteurs, des plus courants et moins précis, aux moins courants et plus précis : marqueur SGML ou mot(s)-clé(s), expression logique sur des valeurs d'attributs, type de concept ou classe d'objets ou encore identificateur de topique (CApH, 1995)[1], concept individuel ou générique[2], GC.

Avant de décrire comment peut se faire l'indexation d'EDs dans CGKAT, notons que dans un contexte d'acquisition de connaissances, le cogniticien peut trouver moins contraignant
1) de rassembler les connaissances des EDs dans divers graphes (chacun de ces graphes étant destiné à synthétiser les connaissances relatives à un point de vue) et d'*indexer des EDs avec des parties de ces graphes*, plutôt que
2) de *construire explicitement des descripteurs pour les EDs*, puis à partir de ces descripteurs, de demander la génération de graphes rassemblant les connaissances relatives à un point de vue.
La première solution n'est pas implémentée dans CGKAT pour les cas généraux (i.e. indexation avec des sous-graphes différents d'un concept ou d'une relation et des concepts qu'elle relie) car, pour être aussi générale que la seconde solution, elle implique la génération automatique de descripteurs et la gestion automatique de ces descripteurs lors de l'évolution des parties de graphes dont ils sont issus. Nous détaillons cela dans la section 6.3.1.4.3.2.

1. Le topique d'un ED est ce dont il parle. Un identificateur de topique doit être unique, représenter un topique unique et être défini par diverses relations par rapport aux autres identificateurs de topique. Dans cette optique, il y a peu de différences entre identificateurs de topique, types de concepts et classes d'objets (nous ne considérons pas ici le fait qu'une classe d'objets peut aussi permettre de représenter des méthodes).

2. Un type de concept représente une caractéristique (et à un ensemble d'individus porteurs de cette caractéristique). Un concept individuel ou générique est donc un descripteur plus précis qu'un type de concept puisqu'il représente *un* individu, déterminé ou non, porteur d'une caractéristique donnée.

6.3.1 Indexation des éléments de documents dans le formalisme des GCs

6.3.1.1 Différents types d'indexation

Si le formalisme des GCs est utilisé pour indexer des EDs, un utilisateur peut vouloir pour cela associer un type, un concept, un GC ou une définition de type à un ED.
Nous avons permis dans CGKAT l'indexation d'EDs par des GCs au sens large : concepts uniques, CGgraphs, définitions de types. Pour des raisons pratiques, nous n'avons pas permis l'indexation d'un ED par un type.

Les types des liens posés entre ces descripteurs et les EDs doivent être prédéfinis et en nombre limité. En effet, si l'utilisateur pouvait créer et utiliser librement des liens de différents types entre EDs et connaissances, la sémantique de ces liens ne serait pas connue de CGKAT, ce qui réduirait les possibilités de recherche : la représentation de connaissances doit s'effectuer via les descripteurs, i.e. avec le formalisme des GCs qui permet des recherches à différents niveaux d'abstraction, plutôt que via les liens hypertextes. Par ailleurs, dans Thot, seuls des liens hypertextes définis dans un modèle structurel peuvent être utilisés et il n'est pas possible de créer de lien hypertexte nommé c'est-à-dire auquel peut être associé un nom choisi par l'utilisateur[1].

Dans CGKAT, le principal type d'indexation entre un ED et une connaissance est le lien de **représentation**. Cependant, pour permettre une meilleure exploitation des descripteurs, nous avons posé un certain nombre de contraintes sur ce que doit être une représentation d'un ED, au moins dans le formalisme des GCs. En contrepartie, nous avons introduit le lien d'**annotation** sans poser de contraintes sur ce que doit être une annotation (si ce n'est le fait qu'elle doit être exprimée via un ED GC).

Pour permettre à différents utilisateurs de CGKAT d'indexer un même document et selon plusieurs vues, nous avons permis que chaque ED puisse être représenté ou annoté par diverses représentations d'EDs et annotations d'EDs. Notons également que des liens structurels peuvent relier des connaissances à d'autres EDs (e.g. une section), mais ce ne sont pas à proprement parler des liens d'indexation.

6.3.1.1.1 Premières restrictions sur les représentations d'EDs

Ce que nous appelons "*une représentation d'un ED*" ne doit représenter que l'ED "lui-même" ou "son contenu"[2], c'est-à-dire qu'il ne doit pas représenter des choses qui, bien que sémantiquement ou structurellement associées à l'ED, ne sont pas incluses ou référées à l'intérieur de l'ED.

Ainsi, typiquement, un GC représentant un ED ne peut inclure une relation conceptuelle entre deux concepts représentant des EDs que si ces EDs sont des sous-éléments du premier ED. Un GC qui inclut plus d'un concept peut représenter le "contenu d'un ED" mais ne peut représenter "l'ED lui-même" : un ED vu comme une entité unique ne peut être représenté que par un concept unique (ou graphe contenant un seul concept). A titre d'exemple, une relation entre EDs ne peut être représentée qu'en utilisant une relation conceptuelle entre des concepts : il est impossible de poser une relation conceptuelle entre des GCs.

De même, une définition de type peut être utilisée pour représenter un ED lorsque celui-ci exprime une définition ou des associations typiques, nécessaires ou suffisantes entre des objets, mais

1. Les liens hypertextes dont nous parlons dans ce chapitre sont des attributs référence. Il n'est pas possible dans Thot d'associer un nom (i.e d'associer un attribut textuel) à un attribut référence.

2. Par l'expression "*contenu de l'ED*", nous faisons référence aux sous-éléments de l'ED ainsi qu'aux entités ou situations réelles ou imaginaires décrites ou référés par ces sous-éléments. Par l'expression "*l'ED lui-même*", nous référons à l'ED vu comme un tout, i.e. à une entité unique qui inclut, réfère ou décrit d'autres entités.

afin de pouvoir représenter "l'ED lui-même", cette définition de type doit être encapsulée dans un concept unique, c'est-à-dire apparaître dans sa partie référent[1].

Enfin, un même utilisateur n'a le droit de construire qu'une seule représentation d'un ED pour un même point de vue (CGKAT connaît le point de vue choisi par l'utilisateur lorsqu'un ED CGRepr est utilisé pour la représentation).

Nous préciserons d'autres restrictions dans la section 6.3.1.3.

6.3.1.2 Un modèle structurel d'extension pour permettre l'indexation par des GCs

Dans Thot, les modèles structurels définis comme des "extensions", peuvent être associés par programme à un document et ainsi compléter les définitions des types d'EDs utilisables dans ce document. Ils peuvent également déclarer des "attributs globaux" qui peuvent alors être portés par n'importe quel ED du document.

6.3.1.2.1 Utilité

Un tel modèle nous était utile pour :

1. déclarer des attributs référence pouvant être portés par n'importe quel ED afin de les connecter à ses représentations et/ou ses annotations ;
2. déclarer des "marques de représentation" pour permettre de délimiter, et donc d'indexer, une suite d'EDs consécutifs, ou encore une sous-partie d'un ED textuel (e.g. un ou plusieurs mots consécutifs dans un paragraphe). Il aurait été plus juste mais peut-être plus déroutant pour l'utilisateur de CGKAT, d'appeler ces "marques de représentation", des "marques d'indexation". Les marques de représentation ont été déclarées comme étant des "paires de marques". Dans Thot, les paires de marques sont des éléments qui vont toujours par deux et qui permettent de délimiter des parties de document. Elles apparaissent par défaut sous la forme de guillemets ouvrants et fermants (le modèle de présentation régit cette apparence : visibilité, couleur, fontes, etc.). Ce sont des éléments de base (i.e. non structurés) comme le sont les chaînes de caractères, les éléments graphiques, les images et les symboles mathématiques. Nous avons autorisé dans notre modèle structurel d'extension, le positionnement des "marques de représentation" autour de n'importe quels EDs sauf des EDs GC.

Par ailleurs, plutôt que de créer des documents séparés pour stocker les indexations d'EDs d'un document, il est plus pratique dans Thot de stocker ces indexations dans le même document mais dans des arbres différents de l'arbre principal du document. C'est pourquoi dans CGKAT, lorsqu'un utilisateur désire représenter ou annoter les EDs d'un document, les arbres relatifs à chaque type d'indexation désiré sont créés dans le document afin de stocker les indexations. Un modèle structurel d'extension permet de définir des types d'arbres qui peuvent être ajoutés aux arbres déjà existants d'un document. Un arbre d'un document qui n'est pas l'arbre principal est appelé un arbre associé. Il est constitué d'une liste d'éléments dits "éléments associés" et son édition se fait dans une fenêtre séparée de celle éditant l'arbre principal. Des liens hypertextes peuvent exister entre les éléments de l'arbre principal et les éléments associés, par exemple dans CGKAT, les liens d'indexation entre EDs et leurs représentations ou annotations.

6.3.1.2.2 Contenu

La figure 6.12 montre le modèle structurel d'extension que nous avons défini pour permettre l'indexation d'EDs par des GCs. Les modèles de présentation et d'interface correspondants sont en

1. Le concept doit être unique puisque seul l'ED est représenté. Rappelons que la création d'un concept unique n'est pas répercutée dans CoGITo, mais s'il comporte un GC (CGgraph ou définition de type) dans sa partie référent, ce GC est répercuté dans CoGITo. Un concept unique est asserté dans CoGITo lorsqu'il est inclus dans d'autres GCs. La section 4.3.2.5.3 à la page 91 donne quelques exemples de concepts comportant des définitions de types dans leur partie référent.

annexe 4. Les marques de représentation sont spécifiées dans les sections "STRUCT" et "EXTENS", et les éléments associés sont définis dans la section "ASSOC". Nous avons utilisé trois "arbres associés" pour stocker les indexations d'EDs : un pour les annotations, un pour les représentations utilisant un concept unique (non répercuté dans CoGITo), et un pour les représentations utilisant un CGgraph ou une définition de type. Cette répartition des indexations dans différents arbres facilite l'accès et l'exploration des indexations.

Pour permettre à un ED de pointer vers ses représentations ou annotations, nous avons dû définir plusieurs attributs référence pour la représentation ainsi que pour l'annotation. Il y a ainsi plusieurs sortes de liens de représentation (resp. d'annotation) :

1. un attribut To_CGReprOnElem ou To_CGNoteOnElem réfère un ED CGRepr (cet ED permet la représentation ou annotation d'un ED selon un utilisateur et un point de vue) ;
2. un attribut To_CGReprsOnElem, To_CGcReprsOnElem ou To_CGNotesOnElem réfère une liste d'EDs CGRepr. Ces types d'attributs sont préférables aux précédents puisqu'une liste d'EDs CGRepr permet à un ED d'avoir plusieurs représentations ou annotations par un ou plusieurs utilisateurs et selon un ou plusieurs points de vue[1] (mais un même utilisateur n'a le droit de construire qu'*une seule représentation d'un ED pour une même vue* ; il n'y a pas de contraintes pour les annotations).
   Un attribut To_CGcReprsOnElem réfère une liste d'ED CGRepr utilisant des concepts uniques tandis qu'un attribut To_CGReprsOnElem réfère une liste d'ED CGRepr utilisant des CGgraph ou des définitions de types. Lorsqu'un ED portant plusieurs attributs référence est activé par l'utilisateur, Thot affiche un menu listant les types de ces attributs, et permettant ainsi à l'utilisateur de choisir le lien hypertexte qu'il désire suivre.

Lorsqu'un utilisateur a sélectionné un ED, il peut via un menu de CGKAT signaler qu'il désire créer une représentation ou bien une annotation sur cet ED. Prenons le cas d'une représentation (le principe est le même pour une annotation) :

1. si l'ED n'a pas déjà été représenté, une liste de CGRepr destinée à contenir les représentations de cet ED, est créée par CGKAT dans l'arbre associé adéquat et un attribut To_CGReprsOnElem ou bien To_CGcReprsOnElem est ajouté à l'ED pour référer cette liste. Lors de sa création, cette liste contient un seul ED CGRepr dont certains sous-éléments sont déjà remplis avec des valeurs par défaut[2] ;
2. si l'ED a déjà été représenté, une telle liste existe déjà et l'utilisateur peut y ajouter un ou plusieurs autres EDs CGRepr si les représentations déjà existantes ne lui semblent pas suffisantes. La figure 6.13 illustre ce point.

Une telle façon de faire est adéquate lorsque les EDs et leurs indexations n'ont pas à être mélangés dans un même "arbre d'EDs"[3]. C'est le cas normal pour une "indexation" de document au sens propre du terme (voir figures 6.13 à 6.19). Cependant, un utilisateur peut vouloir créer un document mélangeant des informations et des connaissances (e.g. figure 6.5). Des liens de représentation ou d'annotation peuvent alors être posés entre certaines informations et des connaissances

1. Notons que pour permettre ceci (i.e. différentes représentations ou annotations pour un même ED), l'usage d'une liste d'EDs pointée par un attribut référence était nécessaire car il n'existe pas dans Thot de type d'attributs permettant de pointer vers différents EDs (un tel attribut serait une liste d'attributs référence). Bien qu'un menu de Thot permette, lorsqu'un ED est référé par divers attributs référence portés par d'autres EDs, l'accés à ces EDs, cet accès est séquentiel et les types des attributs référence ne sont pas affichés. Aussi, même si ce menu aurait pu être utilisé si les ED CGRepr avaient pointé, via des attributs référence, sur les EDs qu'ils représentent ou annotent, cela n'aurait pas été pratique pour l'utilisateur.

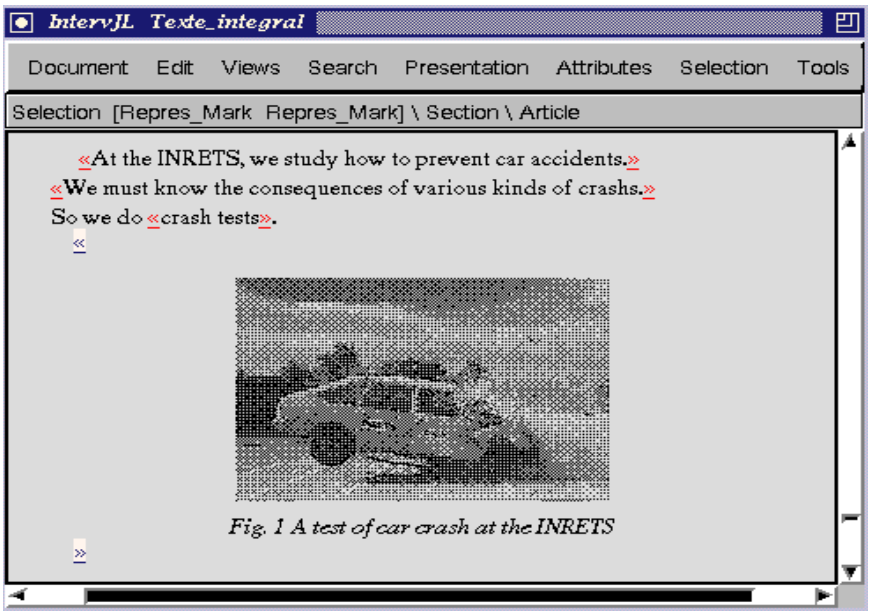
2. L'ED User est automatiquement rempli avec l'identificateur de l'utilisateur, l'ED ViewPoint avec AnyView, l'ED CreationDate avec la date et l'heure du jour, et l'ED TheRepr peut éventuellement être rempli avec un concept d'un type pré-sélectionné.

3. Rappelons qu'un document Thot peut contenir plusieurs arbres d'EDs : un arbre principal et divers arbres associés.

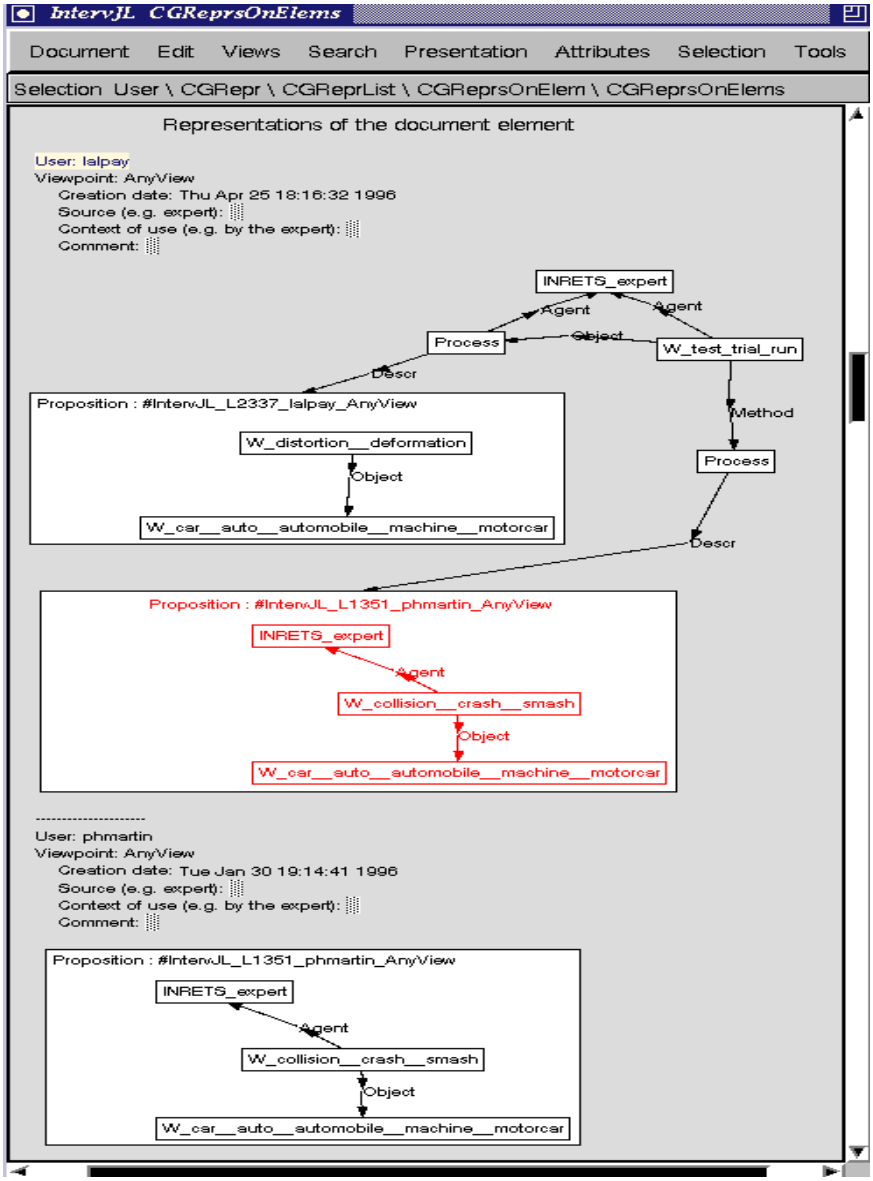
pour expliciter le fait que les premières documentent les secondes et que les secondes indexent les premières. Si les attributs To_CGReprsOnElem, To_CGcReprsOnElem ou To_CGNotesOnElem devaient être utilisés pour cela, une liste de CGRepr devrait être créée pour contenir chaque connaissance indexant une information. Nous avons constaté que cela n'était pas naturel pour les utilisateurs qui ont plutôt tendance, dans de tels cas, à créer non pas une liste de CGRepr mais un seul CGRepr. C'est pourquoi nous avons défini les types d'attributs To_CGReprOnElem et To_CGNoteOnElem qui permettent d'indexer un ED par un seul CGRepr. Des attributs de ces types sont destinés à être posés par l'utilisateur contrairement à ceux des trois types précédents qui sont posés par CGKAT.

```
STRUCTURE  EXTENSION CGExtRepr;    DEFPRES CGExtReprP;    { Structural extension model }

ATTR  { These attributes may be created and associated to any DE }
   To_CGReprOnElem = REFERENCE(CGRepr);  { any DE →(Representation)→ CGRepr }
   To_CGNoteOnElem = REFERENCE(CGRepr);  { any DE →(Annotation)→ CGRepr }

   To_CGReprsOnElem = REFERENCE(CGReprsOnElem);
      { any DE →(Representations)→ CGReprsOnElem=LIST OF (CGReprOnElem=CGRepr) }
   To_CGcReprsOnElem = REFERENCE(CGcReprsOnElem);
      { any DE →(Representations)→ CGReprsOnElem=LIST OF (CGcReprOnElem=CGRepr) }
   To_CGNotesOnElem = REFERENCE(CGNotesOnElem);

   To_Descriptor= REFERENCE(ANY); {any DE → Descriptor including main information of a CGRepr}
   To_Anything = REFERENCE(ANY); {any DE → any DE}

   SessionColored = REFERENCE(ANY);  { A way for CGKAT to handle various colorations for indexed
                                       DEs when a user access (read → know) their indexations }
   ForegroundColor  = black,grey,white,yellow,green,cyan,blue,violet,purple,magenta,red,orange;
   BackgroundColor = black,grey,white,yellow,green,cyan,blue,violet,purple,magenta,red,orange;
                        { → colors of DEs may be specifed by users or programs using attributes }

STRUCT   Repres_Mark = PAIR;  { A pair of representation marks is useful for isolating consecutive DEs
         in a document, or a substring in a textual element; then, since the above attributes may be
         associated to such a pair of marks, consecutive DEs or words may be indexed }
EXTENS   ROOT + (Repres_Mark);          { → representation marks may be inserted around any DE }
         CG - (Repres_Mark);   CGRepr - (Repres_Mark);     { → except around a CG and a CGRepr }

ASSOC  { → kinds of DEs which are not in the main tree of a document but which may be associated to
          the DEs of the main tree in a document → for each kind, a new tree root DE may be created in
          the document for collecting the DEs of this kind (the tree root DE is a list) }
   CGElem = CG; { → a tree root DE named CGElems (= LIST OF (CGElem)) may be created }
   CGReprsOnElem  = CGReprList;  { list of CGRepr where the CG is a CGgraph or a type definition }
   CGcReprsOnElem = CGReprList; { list of CGRepr where the CG is a single concept }
   CGNotesOnElem = CGReprList;  { list of CGRepr for noting any DE, not for representing it }

UNITS   aCG = CG;  { → where units are allowed, a CG may be built }

EXCEPT   { active reference attributes allow users to do hypertext navigation: when a DE has such an
            attribute, a double clic on this DE moves the focus to the refered DE }
   To_CGReprOnElem : ActiveRef;     To_CGNoteOnElem  : ActiveRef;
   To_CGReprsOnElem : ActiveRef;   To_CGNotesOnElem : ActiveRef;
   To_CGcReprsOnElem : ActiveRef;  To_Descriptor : ActiveRef;   To_Anything : ActiveRef;
END
```

Figure 6.12: *Modèle structurel "d'extension" permettant d'indexer des EDs par des GCs et listes de GCs.*



Note: cette portion de document contient quatre paires de marques de représentation. Celle qui entoure l'image a été sélectionnée (voir témoin de sélection). La liste de représentations qui a été associée à cette paire de marques, et donc à cette figure, est montrée dans la fenêtre ci-dessous.



Note: cette fenêtre montre deux représentations, par deux utilisateurs différents, de l'image montrée dans la fenêtre ci-dessus. La première représentation est plus précise que la seconde et la réutilise : elle comprend un ED Concept-Inclusion qui inclut la seconde représentation.

Les inclusions sont colorées en rouge dans l'éditeur Thot. Dans nos copies d'écrans, elles apparaissent donc plus claires que les autres EDs.
Les EDs ConceptInclusion se reconnaissent au large espace blanc qu'il contiennent dans la partie gauche de leur boîtes (cette espace est destinée à faciliter la sélection de l'ED ConceptInclusion avec la souris, en évitant la sélection de sa sous-partie, l'ED Concept qui est inclus).

Figure 6.13: *Une liste d'EDs CGRepr (2ème fenêtre) représentant la figure entre marques de représentation*

Par ailleurs, chaque GC (ED SingleConcept, CGgraph ou TypeDefinition) peut porter un attribut "référence" (To_Element) vers l'élément qu'il indexe (cf. le modèle structurel d'un ED GC). C'est cet attribut qui est exploité par CGKAT pour permettre la recherche d'EDs via les GCs qui les indexent. Cet attribut est automatiquement créé lors de la première représentation ou annotation d'un ED par un concept (ED SingleConcept). Mais pour des raisons techniques, dans la version actuelle de CGKAT, l'utilisateur doit créer manuellement cet attribut pour les EDs CGgraph ou TypeDefinition, ou pour les indexations suivantes du même ED, s'il veut pouvoir retrouver l'ED indexé, via des requêtes sur la base de GCs[1].

Nous donnons dans les figures 6.14 à 6.19 d'autres exemples d'indexations d'EDs d'un document. Nous avons souligné dans la section 6.3.1.1.1 certaines contraintes que devait à notre sens respecter une représentation d'un ED dans le formalisme des GCs. Dans la section prochaine, nous préciserons d'autres contraintes, toujours afin de permettre une exploitation plus sûre des "représentations d'EDs". Nous respectons ces contraintes dans les exemples qui suivent.

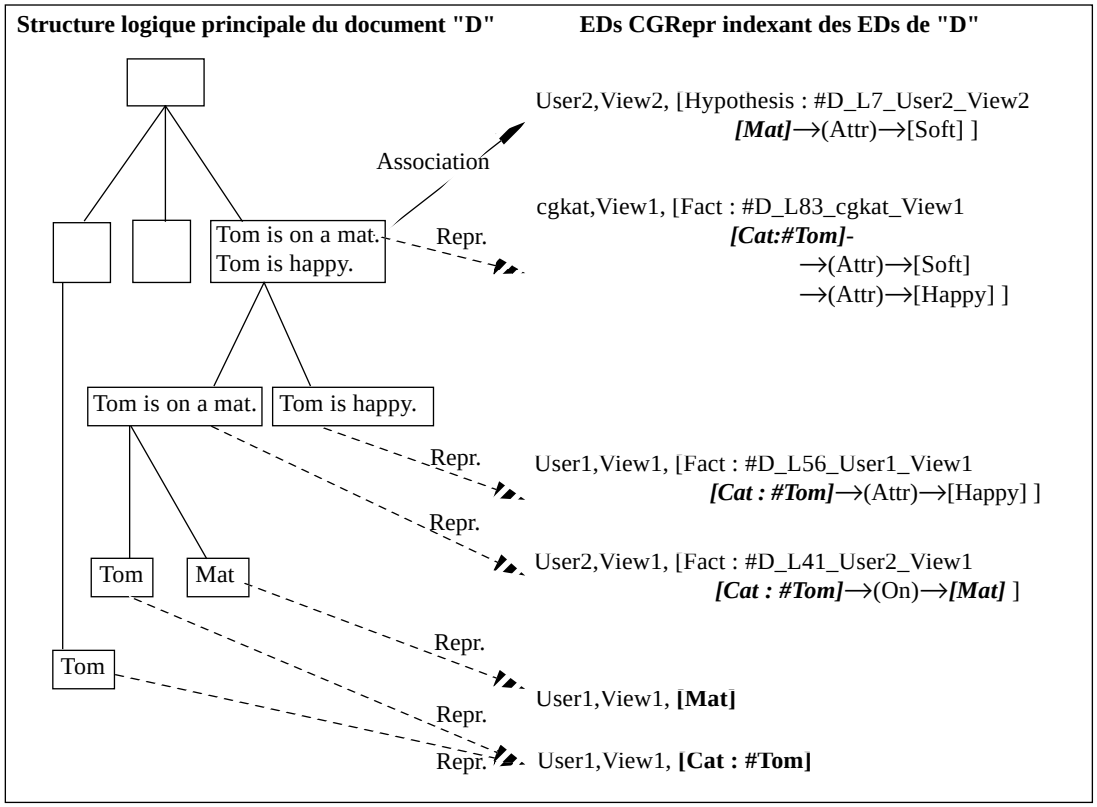


Figure 6.14: *Liens hypertextes entre des éléments de documents et leurs indexations.*

[Mat] et ***[Cat : #Tom]*** représentent des EDs ConceptInclusions incluant **[Mat]** et **[Cat : #Tom]**.
La navigation hypertexte est possible sur les liens d'indexation, sur les liens d'inclusion et sur les liens structurels.
Bien que cela ne soit pas encore implémenté dans CGKAT, nous donnons avec l'ED CGRepr dont l'auteur est "cgkat", un exemple de génération d'une représentation pour un ED, à partir des représentations de sous-éléments de cet ED :
l'ED CGRepr dont l'auteur est "cgkat" a été créé en effectuant 1) une jointure maximale sur les GCs des représentations de "Tom is on a mat" et de "Tom is happy", et 2) une généralisation sur les méta-informations associées à ces GCs. Ici, cela marche bien parce que la jointure s'est faite uniquement sur un concept individuel. Si l'on avait eu "A cat is on a mat" et "A cat is happy", la jointure sur les représentations de ces phrases aurait signifié "A cat is on a mat and is happy", ce qui aurait pu être incorrect. Dans le cas général, définir une "bonne" jointure maximale, i.e. utile pour une application, est complexe. Par exemple, comment définir une jointure sur des définitions de types ?

1. Ce problème ne nous est apparu que récemment car pour chaque ED, nous ne créions qu'une seule indexation et en utilisant un concept unique (avec éventuellement un GC dans sa partie référent). La création automatique de l'attribut To_Element est dans les autres cas plus difficile à automatiser car alors des recherches sont à effectuer dans la structure du document. Cela sera implémenté dans la prochaine version de CGKAT.

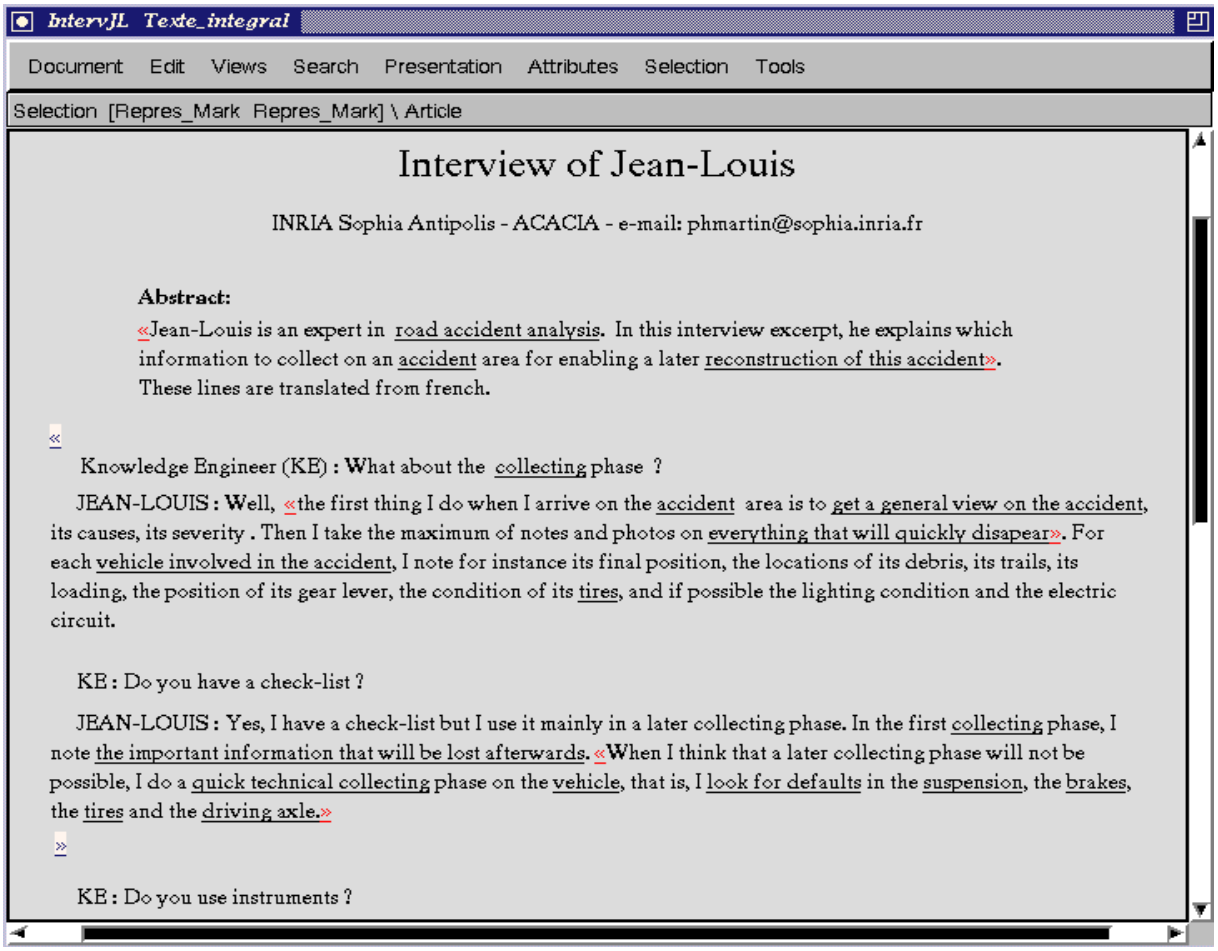


Figure 6.15: *Le début d'un document contenant la retranscription de l'interview d'un expert.*

Les éléments soulignés ou entourés de marques de représentation ont été indexés. Chacun des éléments soulignés pointe avec un attribut To_CGcReprsOnElem sur une liste de représentations de type CGcReprsOnElem (représentation avec un concept unique d'un "ED symbole" ; cf. section 6.3.1.3.2). Chacun des éléments entourés de marques est un "ED description" et pointe avec un attribut To_CGReprsOnElem sur une liste de représentations de type CGReprsOnElem.

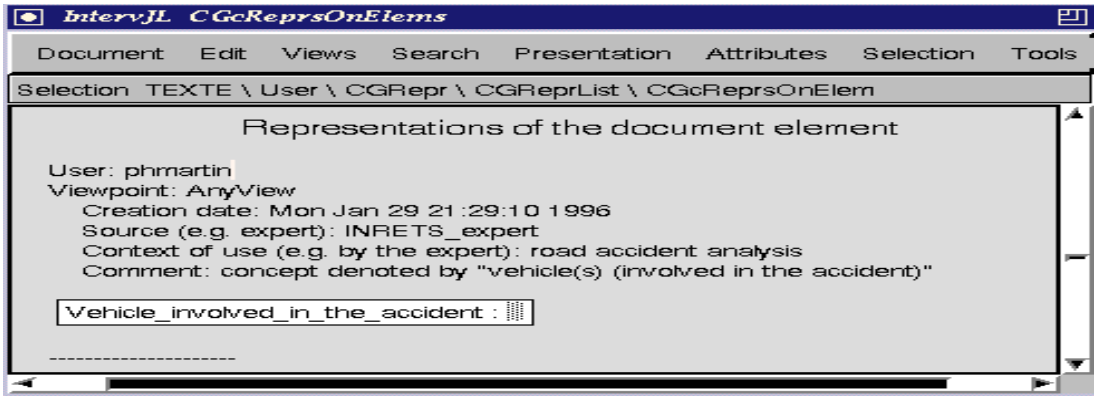


Figure 6.16: *Représentation d'une occurence du mot "vehicle".*

A chaque occurrence de "vehicle" ou de "vehicle involved in the accident" dans le document de la figure précédente, a été associée une liste de représentations, telle que celle illustrée ci-dessus. Ces listes "partagent" un même ED CGRepr dans le sens où elles contiennent une inclusion de cet ED (plus exactement, l'une d'elles contient l'ED original et les autres incluent cet ED). Ainsi, ces listes contiennent une même représentation et peuvent chacune contenir si nécessaire d'autres représentations créées par d'autres utilisateurs ou selon d'autres points de vue.
Autres exemples de représentations de mots du document ci-dessus : "tires" → [W_tire_tyre : *]
([W_tire_tyre : {*}] semblerait préférable mais les référents ensemblistes ne sont pas encore acceptés dans CoGITo), "collecting" → [Gathering_of_information_on_an_accident : *y].

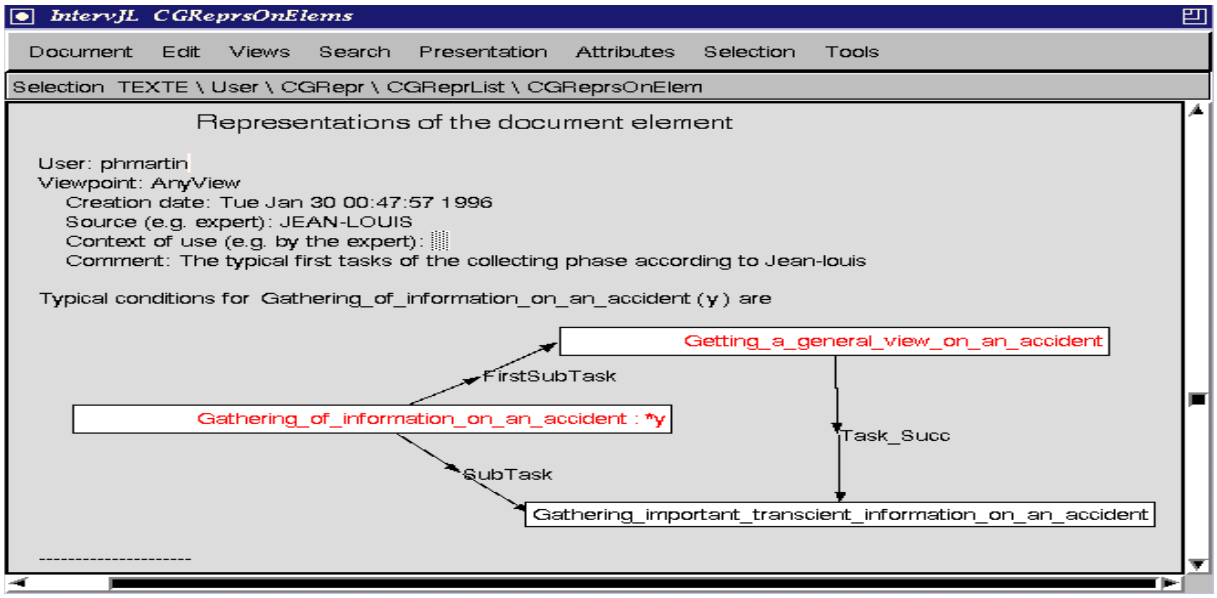


Figure 6.17: *Une représentation de deux sous-tâches typiques du recueil d'information sur un accident.*

Phrases de Jean-Louis ici modélisées : "the first thing I do when I arrive on the accident area is to get a general view on the accident, its causes, its severity . Then I take the maximum of notes and photos on everything that will quickly disapear".
Rappel : les concepts affichés avec une large espace à gauche sont des EDs ConceptInclusion.

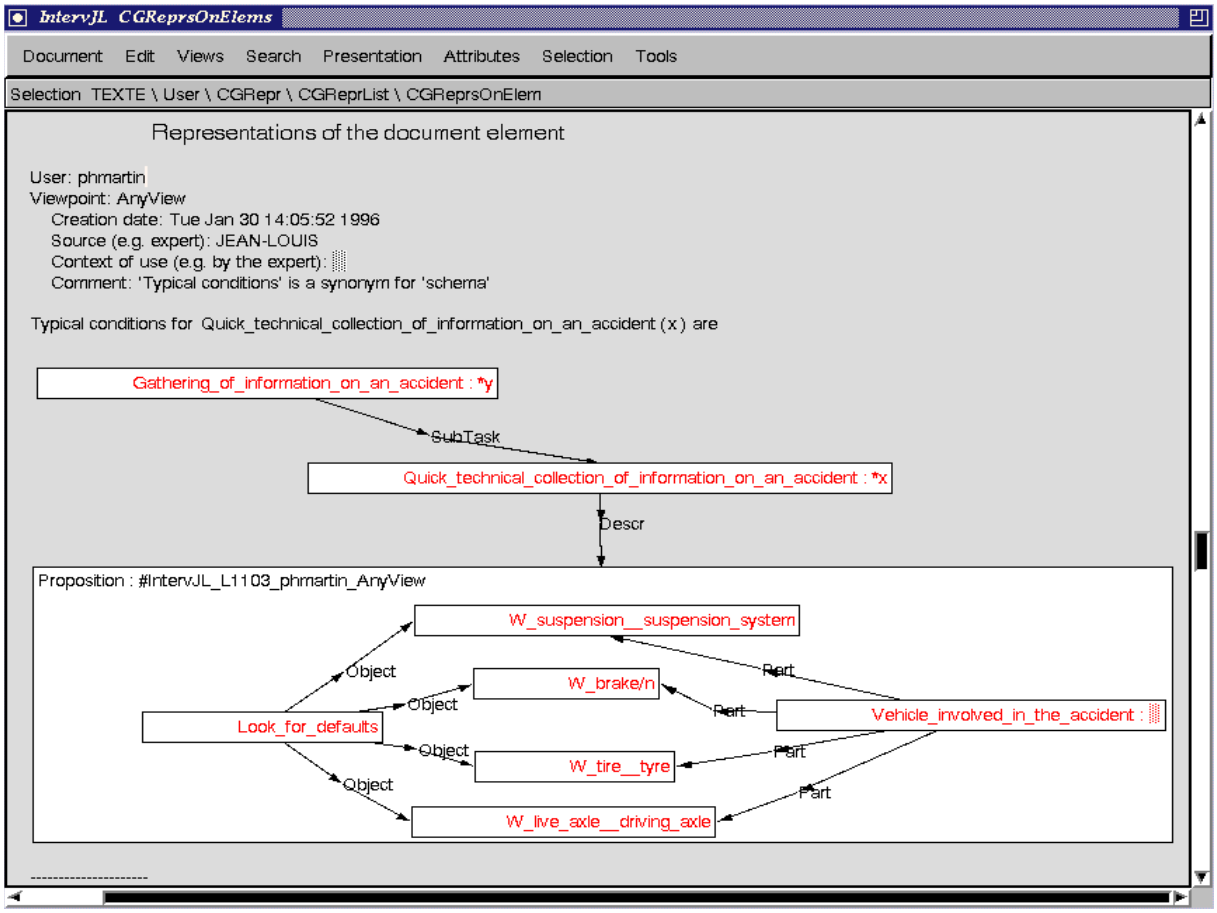


Figure 6.18: *Une représentation (partielle) d'informations liées à la "première phase de collection rapide".*

Phrase de Jean-Louis ici modélisée : "When I think that a later collecting phase will not be possible, I do a quick technical collecting phase on the vehicle, that is, I look for defaults in the suspension, the brakes, the tires and the driving axle". Le test "When I think ... possible" n'a pas été représenté. Rappelons également que CGKAT ne permet pas encore d'utiliser des référents ensemblistes.

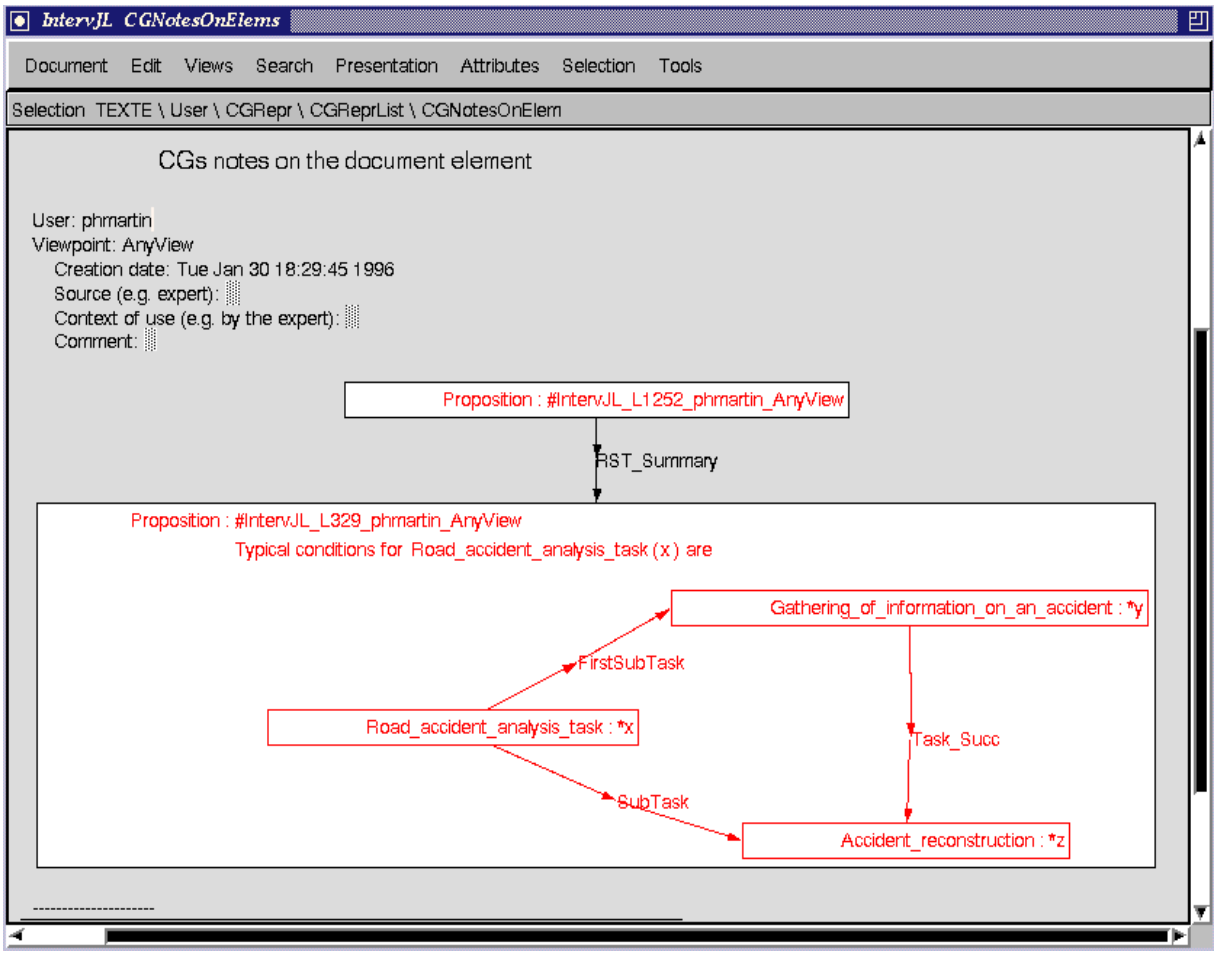


Figure 6.19: *Une liste d'annotations (liste d'ED CGRepr annotant un ED).*

Cette liste d'annotation est associée à la paire de marques de représentation sélectionnée dans le document de la figure 6.15, i.e. celle englobant la plus grande partie visible du document. Cette partie a été représentée par le concept [Proposition : #IntervJL_L1252_phmartin_AnyView]. Le résumé a été représentée par le concept [Proposition : #IntervJL_L329_phmartin_AnyView]. L'annotation présentée ci-dessus relie ces deux concepts par une relation de type RST_Summary afin de représenter la relation rhétorique qui relie les EDs représentés par ces deux concepts. Conformément à nos restrictions sur ce que doit être une représentation d'un ED, une telle relation pouvait être représentée soit dans une représentation d'un ED englobant les EDs liés par la relation, soit dans une annotation. C'est cette dernière solution que nous avons choisie.

6.3.1.3 Ontologie pour la représentation d'éléments de documents

6.3.1.3.1 Les différentes notions qui peuvent être représentées

Un ED peut soit *réfèrer (être une référence à)* une entité ou à une situation réelle ou imaginaire (e.g. un mot réfère une entité ou une situation), soit *décrire* une situation (c'est le cas de tout ED structuré et des images). Il est important lors de la représentation d'un ED de faire la distinction entre a) l'entité ou la situation *référée* ou *décrite* par un ED, et b) la *référence* ou la *description* elles-mêmes. En effet, le symbole n'est pas l'objet symbolisé et la description d'une situation n'est pas une situation. Des relations sémantiques différentes s'appliquent à ces objets[1].
Par ailleurs, pour effectuer cette *référence* ou cette *description*, l'ED utilise un format et un medium : image, graphique, texte, langage naturel, expression logique, paragraphe, etc.

Ainsi, au sens propre, "représenter un ED", c'est représenter un objet qui est une *référence* ou bien une *description*, utilisant un format et un medium (support d'information). Nous avons proposé le type Representation_entity pour catégoriser de tels objets (cf. section 5.4.2.4.1 à la page 130) et nous avons proposé le type Document_element, sous-type de Representation_entity, pour catégoriser les types d'EDs. Les objets de type Representation_entity peuvent être considérés comme atomiques (e.g. un caractère, un nombre) ou structurés (e.g. un tableau, un graphe, une phrase).

Cependant, pour simplifier la tâche du cogniticien, il convient d'accepter une définition plus large et plus naturelle de la représentation d'un ED. Dans CGKAT, "représenter un ED", c'est représenter :
1) l'entité ou la situation *référée* ou *décrite* par l'ED, *et/ou*
2) la *description* (d'une situation) faite par l'ED, *et/ou*
3) un objet qui est une référence ou une description utilisant un format et un medium.

Quatre types de haut niveau sont donc nécessaires pour la représentation d'EDs. Dans notre ontologie, nous avons adopté les noms suivants pour ces types : Entity, Situation, Proposition (pour les descriptions) et Representation_entity. Des relations peuvent être posées :
1) entre des concepts de type Entity ou Situation pour décrire un certain monde,
2) entre des concepts de type Proposition pour représenter des relations rhétoriques[2] ou d'argumentation[3] entre des descriptions,
3) entre des concepts de type Document_element (sous-type de Representation_entity) pour décrire des relations structurelles[4] entre des EDs.
Un concept de type Proposition peut également être connecté par des relations conceptuelles à des concepts autres que Proposition afin de représenter a) ses différentes origines (e.g. ED source, auteur source, auteur du document, auteur de la représentation), b) sa valeur de vérité (e.g. vrai, faux, inconnu) ou sa modalité (e.g. toujours vrai, possible), ou c) la situation qu'il décrit.

L'ontologie par défaut de CGKAT
1) propose de nombreux sous-types pour les types Entity, Situation, Proposition et Representation_entity (au moins 90.000 types puisque les types de haut niveau de WordNet sont classés sous ces types),
2) pose certains liens d'exclusion entre ces types, et
3) propose plus de 200 types de relations signées sur ces types de concepts ou certains de leurs sous-

---

1. Comme nous l'avons noté en section 4.3.2.4 à la page 86, on retrouve ces distinctions élémentaires dans de nombreux travaux en linguistique et en représentation des connaissances, sous différents noms : 1) objet référé, extension du concept, 2) idée, intention du concept, 3) signe ou symbole. Nous reprenons la terminologie de Sowa (1992) qui s'inspire des travaux sur la "Sémantique des situations" (Barwise & Perry, 1983).

2. Exemples de types de relations rhétoriques : Summary, Concession, Antithesis, Circumstance, Purpose.

3. Exemples de types de relations d'argumentation : Suggestion, Objection, Confirmation, Contribution, Answer.

4. Dans CGKAT, décrire avec des GCs les relations structurelles d'un document est a priori inutile puisque Thot est un outil permettant la gestion et la recherche de ce genre de relations. L'utilisateur peut néammoins représenter avec des GCs les relations structurelles d'un document s'il désire intégrer dans sa recherche d'informations "via les connaissances" cet aspect des documents.

types.
L'utilisateur est ainsi guidé dans sa modélisation des EDs.

Notons que si un concept de type Proposition représente un ED particulier ou encore un ensemble d'ED particuliers, ce doit être un concept individuel puisqu'il représente alors *l'objet description* que contient cet ED ou chacun de ces EDs. Ce concept peut inclure dans sa partie référent un GC afin de représenter plus en détail le contenu de la description. On peut trouver dans la figure 6.13 un exemple d'un tel concept individuel de type Proposition : celui utilisé dans la seconde représentation de l'image. Rappelons que lorsqu'un GC est créé dans la partie référent d'un concept (de type Proposition donc), CGKAT génère automatiquement un nom pour le graphe et donc un référent individuel pour le concept qui l'emboîte, mais ce nom et donc ce référent peuvent être modifiés par l'utilisateur.

6.3.1.3.2 Représentation des "EDs symboles" et des "EDs descriptions"

Nous appelons "EDs symboles" les EDs qui *réfèrent* des entités ou des situations, et "EDs descriptions" les EDs qui véhiculent des descriptions (propositions) *décrivant* des situations. Les icônes sont des EDs symboles. Les mots et les groupes nominaux sont également des EDs symboles lorsqu'ils ont été isolés dans des EDs (cela est fait automatiquent par CGKAT lorsqu'ils sont représentés). Des exemples d'EDs descriptions sont les phrases, les graphiques, les images, les paragraphes, les sections et tout autre ED structuré.

- Un ED description est composé d'EDs symboles ou descriptions, et peut donc être représenté par un GC composé de concepts simples ou comportant un graphe dans leur partie référent. CGKAT génère un message d'avertissement lorsqu'un concept unique de type autre que Proposition est utilisé pour représenter un ED description[1]. L'utilisateur doit utiliser un concept unique de type Proposition s'il veut pouvoir représenter des relations entre l'ED description et d'autres EDs. Au besoin, ce concept peut contenir un CGgraph ou une définition de type dans sa partie référent, afin de représenter le contenu de l'ED description.
- Comme les mots sont des EDs symboles et non des EDs descriptions, CGKAT génère un message d'avertissement lorsqu'un CGgraph, une définition de type ou un concept de type Proposition est utilisé pour représenter un mot (idem pour les paires de mots). Un ED symbole est typiquement représenté par un concept unique de type autre que Proposition.

Dans CGKAT, si une liste d'EDs CGRepr est utilisée pour contenir les représentations d'un ED description, cette liste doit être de type CGReprsOnElem. Pour un ED symbole, une telle liste doit être de type CGcReprsOnElem. C'est bien sûr l'utilisateur qui décide de considérer un ED comme un ED symbole ou un ED description : il effectue ce choix en utilisant pour représenter l'ED, un concept unique de type autre que Proposition, ou bien une autre représentation (e.g. en utilisant un ED CGgraph). Cependant, CGKAT possède des heuristiques afin de mettre en garde l'utilisateur en cas de choix étranges :
1) un mot, un groupe de deux mots, un symbole (notamment, l'un des 36 symboles graphiques proposés par Thot) et une icône devraient plutôt être considérés comme des EDs symboles ;
2) un ED structuré et un groupe de plus de trois mots[2] devraient plutôt être considérés comme des EDs descriptions.

Lorsque qu'un ED est représenté par une liste d'EDs CGRepr, il réfère par un attribut To_CGReprsOnElem à cette liste si elle est de type CGReprsOnElem, et par un attribut To_CGcReprsOnElem si elle est de type CGcReprsOnElem.

---

1. Nous supposons que l'utilisateur de CGKAT n'a pas envie de représenter sous la forme d'un concept unique, le format ou le medium utilisé par l'ED puisque cette information correspond au type de l'ED et est donc déjà stockée dans la structure logique du document (et Thot est un outil dédié à la gestion de ce genre d'informations). Si nous voulions prendre en compte ce cas, il nous faudrait introduire en dur dans le code de CGKAT le type Representation_entity (cf. section 5.4.2.4.1) et demander à l'utilisateur d'utiliser des spécialisations de ce type lorsqu'il représente le format ou le medium utilisé par un ED.

2. Pour les groupes de trois mots, CGKAT ne génère jamais de message d'avertissement.

6.3.1.4 Remarques supplémentaires sur la représentation d'EDs

6.3.1.4.1 Marques de représentation

Comme les utilisateurs de CGKAT peuvent représenter de nombreux EDs dans un document, nous avons choisi pour des raisons d'ergonomie, de limiter le nombre de marques de représentations : lorsqu'un utilisateur sélectionne un ED et indique à CGKAT qu'il veut le représenter, des marques de représentation ne sont créées par CGKAT pour isoler cet ED que lorsqu'il s'agit d'un ED description.

En effet, les marques ont des avantages et des inconvénients.

- Elles seules permettent d'isoler une suite d'EDs structurés consécutifs, et elles peuvent être partiellement ou totalement enchevêtrées (alors que les autres EDs sont liés par des liens structurels, c'est-à-dire de composition). Ainsi des bouts de documents (EDs) ayant des sous-parties communes peuvent être représentés, ce qui est intéressant même lorsqu'un seul utilisateur travaille sur un document.
- Cependant cet enchevêtrement est assez difficile à lire même dans des cas simples. Voici un exemple, en supposant que les EDs symboles soient eux aussi entourés de marques lorsqu'il sont représentés : «The «cat» on the «mat» is a «lazzy«cat»»».

Or, les EDs symboles sont des parties de document suffisamment courtes pour ne pas avoir de sous-parties en commun. D'autre part, dans Thot, il est possible à l'intérieur d'un ED textuel, de séparer le texte en plusieurs sous-EDs disjoints de type Texte. C'est cette solution qui a été choisie dans CGKAT pour isoler les mots ou groupes de mots lorsqu'ils sont considérés et représentés par l'utilisateur comme des EDs symboles. Pour les autres cas d'EDs symboles (e.g. les symboles graphiques de Thot), l'élément est déjà un ED isolé.

Dans notre modèle de présentation actuel, les EDs symboles qui sont représentés, sont par défaut colorés en rouge (pour que les EDs symboles apparaissent soulignés dans la figure 6.15, nous avons temporairement modifié notre modèle de présentation). Les marques de représentation sont également colorées en rouge.

6.3.1.4.2 Représentation selon un point de vue

Grâce à l'ED Viewpoint, un ED CGRepr permet à l'utilisateur d'indiquer qu'une représentation d'un ED est relative à un point de vue (celui-ci étant alors automatiquement représenté sous forme d'une méta-information). Mais, quelle est la signification de "représenter un ED selon un point de vue" ? Une interprétation semble être de représenter seulement les aspects de l'ED relatifs au point de vue choisi, c'est-à-dire 1) si l'ED est un ED symbole, ne représenter l'entité ou la situation référée que selon certains de ses aspects, et 2) si l'ED est un ED description, ne représenter de cette description que les informations relatives à certains aspects, les informations retenues pouvant être issues ou non de sous-parties bien définies de la description totale. Une telle interprétation est donc celle d'une représentation partielle. Nous n'avons pas voulu imposer d'interprétation aux utilisateurs de CGKAT. Notons par ailleurs que toute représentation est implicitement partielle même si le point de vue n'est pas explcitement défini, puisqu'elle correspond à une abstraction d'une certaine réalité.

Remarquons que l'interprétation d'une représentation partielle permet d'introduire de nouvelles dimensions dans la représentation de relations rhétoriques ou argumentatives entre EDs descriptions. En effet, supposons que deux EDs descriptions E1 et E2 aient été respectivement représentés selon les points de vue V1 et V2, par les concepts C1 et C2, et qu'un utilisateur crée un GC reliant C1 et C2 par une relation R. Cela peut être interprété comme la représentation d'une relation R entre les informations contenues dans E1 et relatives à V1 avec les informations contenues dans E2 relatives à V2. Il est ainsi possible de représenter le fait qu'il existe dans un ED des informations relatives à tel sujet, telle tâche ou encore telle catégorie de connaissances, et que ces informations sont le résumé d'autres informations contenues dans un autre ED. La figure 6.19 montre la représentation d'une relation de résumé entre deux EDs mais sans que des points de vue particuliers aient été adoptés pour

leurs représentations. Cela est utile lorsqu'un utilisateur ne peut ou n'a pas envie de représenter explicitement dans un ED toutes les sous-parties relatives à un point de vue particulier.

6.3.1.4.3 Perspectives : génération de représentations d'EDs

6.3.1.4.3.1 Génération de représentations d'EDs à partir de représentations de sous-EDs

Bien que cela ne soit pas encore implémenté dans CGKAT, notons que si des EDs E1,...,En sont représentés par des concepts, une représentation par défaut pourrait être automatiquement générée pour un ED incluant E1,...,En, par exemple en effectuant 1) une jointure maximale sur les GCs des représentations, et 2) une généralisation sur les méta-informations qui leur sont associées. Nous avons montré dans la figure 6.14, un exemple d'une telle génération. Mais comme nous l'avons souligné, il est complexe de définir une jointure maximale intéressante dans le cas général, et le cogniticien devra toujours vérifier et probablement corriger les résultats de telles générations. Néanmoins, de telles générations pourraient faciliter et accélèrer le travail du cogniticien. Les résultats de générations similaires sur les annotations seraient beaucoup plus douteux compte-tenu de l'absence de contraintes sémantiques sur celles-ci.
Dans RIME (Kheirbek & Chiaramella, 1995), un système de recherche d'informations fondé sur la représentation de documents ou de parties de documents par des GCs, une jointure maximale dirigé (Nogier, 1991) est utilisé pour générer, à partir des représentations d'EDs, celles des EDs qui les contiennent. Cependant, le but de ces générations n'est pas de générer des représentations correctes du contenu des EDs mais de faire en sorte que leurs composants (ceux dont les représentations ont été jointes) soient mieux retrouvés. En acquisition des connaissances, l'utilisation de jointures de représentations est différente.

6.3.1.4.3.2 Génération de représentations d'EDs à partir de parties de graphes

Compte-tenu de leur sémantique, les représentations d'EDs sont particulièrement intéressantes pour permettre la génération de documents (cf. section suivante) et pour permettre de représenter des relations sémantiques, rhétoriques et argumentatives entre EDs. Cependant, comme nous l'avons noté au début de la section 6.3, dans un contexte d'acquisition de connaissances, l'utilisateur de CGKAT peut trouver moins contraignant de représenter les connaissances de documents
1) en rassemblant les connaissances des EDs dans divers graphes (chacun de ces graphes étant destiné à synthétiser les connaissances relatives à un point de vue) et en *indexant des EDs avec des parties de ces graphes*, plutôt que
2) en *construisant explicitement des descripteurs pour les EDs*, puis à partir de ces descripteurs, en demandant la génération de graphes rassemblant les connaissances relatives à un point de vue.

Il serait donc intéressant de permettre à l'utilisateur de faire générer à CGKAT des descripteurs (EDs CGRepr) à posteriori, i.e. à partir de GCs déjà construits, simplement en désignant des sous-parties de ces GCs et leurs sources dans les documents. L'utilisateur pourrait ainsi bénéficier de représentations d'EDs aux endroits où il le désire, sans avoir eu à les construire systématiquement. L'implémentation de cette fonction dans CGKAT exploiterait le mécanisme d'inclusion. Ainsi, une modification des sources des concepts et relations inclus dans les représentations d'EDs générées entraînerait la modification de ces représentations.

Notons une autre possibilité qui, dans certains cas, ne nécessiterait pas une génération explicite de représentations d'EDs à partir de sous-parties de GCs. Ces cas sont :
1) lorsque la sous-partie d'un GC devant être utilisée pour représenter un ED se réduit à un seul concept,
2) lorsque cette sous-partie se réduit à une relation et aux concepts qu'elle relie (ses voisins immédiats).
En effet, dans ces deux cas, *l'attribut To_Element peut être porté par un concept mais aussi par une relation* (cf. le modèle structurel d'un ED GC).

1. Dans le premier cas, cela signifie que le *concept représente l'ED qu'il pointe via cet attribut, tout comme s'il était un concept unique* (ceci est implémenté dans la version actuelle de CGKAT). Il

n'est donc pas indispensable de créer un ED CGRepr contenant ce concept et représentant l'ED, puisque l'ED est déjà représenté par le concept. En d'autres termes, la sous-partie de graphe que constitue le concept, n'a pas besoin d'être copiée dans un ED CGRepr pour être isolée du reste du graphe : le fait de porter l'attribut To_Element suffit à isoler le concept dans le graphe.

2. Dans le second cas, cela signifierait par convention que *le sous-graphe formé de la relation et de ses voisins immédiats représente l'ED pointé par l'attribut To_Element porté par la relation*. Cela n'est pas encore véritablement implémenté dans CGKAT : le sous-graphe *ainsi isolé* par l'attribut To_Element, n'est pas créé en tant que tel dans CoGITo et ne peut donc être recherché par projection : l'ED représenté ne peut donc être recherché par ce moyen. Toutefois, des attributs To_Element peuvent actuellement être posés sur des relations et des recherches par navigation hypertexte peuvent donc exploiter de tels liens hypertextes. Notons que l'ED pointé par un attribut To_Element porté par une relation doit être un ED description puisqu'il est représenté par un graphe et non par un concept unique.

6.3.1.4.4 Génération d'index synthétisant des représentations d'EDs

Thot inclut un générateur d'index (Richy, 1994) travaillant sur des "marques d'index". Ces marques sont définies dans le modèle structurel d'extension ExtIndex.S de la librairie de Thot. Ce schéma spécifie également la composition d'éléments de type "descripteur de marque" et "table d'index". Tout comme les marques de représentation, les "marques d'index" sont des paires de marques. De même, les "descripteurs de marques" sont des "éléments associés" tout comme le sont les listes de représentation ou d'annotation d'un ED. Normalement, ce schéma d'extension est chargé depuis l'éditeur Thot et ajouté au modèle structurel d'un document en cours d'édition, de manière à pouvoir isoler des EDs de ce document par des marques d'index et leur associer des descripteurs de marques.

Un descripteur de marques contient une liste de clés d'indexation, chacune devant être composée avec des caractères alpha-numériques afin de permettre des tris alpha-numériques. Seule la clé primaire est obligatoire. Lorsque l'utilisateur insère des marques d'index autour d'une portion de texte, Thot crée automatiquement un descripteur de marque, remplit sa clé primaire avec le contenu de la portion de texte, et pose un lien hypertexte entre les marques d'index et ce descripteur de marque. Si l'utilisateur insère des marques d'index autour d'un ED quelconque, un descripteur de marque est automatiquement créé mais sa clé primaire n'est pas remplie. Un descripteur de marque a également trois sous-éléments optionnels : un sujet (texte), une sémantique (texte) et une glose (i.e. une définition ou une annotation en langage naturel ; un ED quelconque peut être utilisé pour cela).

Lorsque des descripteurs de marques ont été remplis et des éléments d'un document ainsi indexés, l'utilisateur peut demander la génération d'une table d'index triant et donc synthétisant ces descripteurs. Plusieurs sortes de tables peuvent être construites (e.g. index des auteurs, index des sujets et index général) et des options de tri peuvent être définies via des attributs. Chaque entrée d'une table d'index est reliée par des liens hypertextes aux éléments qu'ils indexent dans le document (et comme dans les index classiques, le numéro de page contenant l'élément est indiqué). Elle est également reliée aux descripteurs de marques qu'elle synthétise. Des références croisées, sous forme de liens hypertextes et de renvois, peuvent également être générées à l'intérieur d'une table d'index. Lorsqu'un document inclut d'autres documents, ses index synthétisent ceux de ces documents.

Nous avons permis l'exploitation de ce générateur d'index pour synthétiser des représentations d'EDs. Le principe est simple : pour chaque représentation d'ED créée par un utilisateur, CGKAT génère un descripteur de marque contenant dans sa clé primaire le contenu textuel de l'ED représenté, et dans ses autres sous-éléments, certaines des informations de la représentation. L'utilisateur peut alors faire générer ou régénérer des tables d'index synthétisant ces informations. Préalablement, il peut éventuellement compléter les descripteurs générés.

Deux cas assez différents de représentations d'EDs se présentent : les représentations des EDs symboles et celles des EDs descriptions.

1. Lorsque ce n'est pas un symbole graphique (e.g. le symbole proposé par Thot pour une flèche vers le haut), un ED symbole est généralement une portion de texte assez courte (typiquement un mot ou un groupe nominal) qu'on peut inclure en tant que clé primaire d'un descripteur de marque. De plus, un tel ED étant représenté par un seul concept, le type de ce concept peut être également inclus comme clé de tri.
2. Par contre un ED description peut être n'importe quel ED (e.g. une phrase, une section, un graphique) et ne peut donc être utilisé comme clé primaire d'un descripteur de marque. De plus, un tel ED est représenté par un CGgraph, par une définition de type ou par un concept de type Proposition pouvant contenir un GC dans sa partie référent : CGgraphs et définition de types ne peuvent être utilisés comme des clés de tri, et dans le troisième cas, utiliser le type du concept serait peu significatif.

Nous avons donc décidé que, au moins dans un premier temps, les descripteurs de marques générés par CGKAT ne le seraient qu'à partir de représentations d'EDs symboles. De plus, pour des raisons pratiques, nous n'avons également considéré que les représentations situées dans des listes de CGRepr de type CGcReprsOnElem[1]. Ainsi, lorsqu'un de ces EDs CGRepr est créé, CGKAT génère un descripteur de marques comportant les informations suivantes : le contenu de l'ED indexé, le type du concept utilisé pour la représentation, le nom de l'utilisateur ayant créé la représentation, la vue selon laquelle la représentation a été créée, ainsi que la source de la représentation et le commentaire qui y a été associée, si ces deux dernières informations existent. Le commentaire remplit le champ "Glose" du descripteur, la vue remplit le champ "Sujet" et le type de concept remplit le champ "Sémantique". La vue, le type de concept sont également utilisés dans une clé de tri, de même que le nom de l'utilisateur, la source de la représentation et le contenu de l'ED indexé.

Pour des raisons de présentation, nous n'avons pas utilisé plusieurs clés de tri, mais une seule (la clé principale donc), dans laquelle diverses informations sont concaténées. Une autre solution aurait été d'utiliser plusieurs clés et de modifier le modèle de présentation associé au modèle structurel d'extension ExtIndex.S, mais ceci pouvait éventuellement poser des problèmes de tri et/ou de présentation si, en plus des descripteurs générés par CGKAT, l'utilisateur créait à la main des descripteurs avec pour les clés des sémantiques différentes de celle employée par CGKAT.

Plus exactement, la clé principale d'un descripteur généré par CGKAT à partir d'un ED CGRepr, est constituée en utilisant des inclusions : celles de l'ED indexé et celles des EDs ConceptType, User, View et Source contenus dans l'ED CGRepr. Ainsi, lorsque les sources de ces inclusions sont modifiés, les descripteurs sont automatiquement mis à jour. De plus, la navigation est possible depuis ces inclusions vers leurs sources. C'est également le cas depuis les entrées de la table d'index puisque ces entrées sont formées en réutilisant le contenu des descripteurs. Bien sûr, la navigation est également possible depuis les EDs inclus vers leurs inclusions. Ces liens d'inclusion complètent les liens hypertextes posés par le générateur d'index.

La figure 6.20 montre une table d'index synthétisant les représentations associées aux EDs symboles montrés dans la figure 6.15 (i.e. les EDs soulignés dans la partie de retranscription d'interview de cette figure 6.15).

1. Ceci n'est pas une grande limitation puisque de telles listes sont en principe la manière la plus naturelle pour l'utilisateur de représenter des EDs symboles. Il n'y a en effet guère d'intérêt à créer des concepts uniques en plein milieu d'un document et à les relier à d'autres éléments de ce document par des liens To_Element.

Figure 6.20: *Une table d'index synthétisant certaines caractérisques de représentations de termes.*

Cette table a été générée par le générateur d'index de Thot à partir des marques d'index qui ont été créées par CGKAT à partir des représentations associées aux EDs symboles montrés dans la figure 6.15. Dans cette figure, chaque entrée de l'index ne contient que la clé primaire sur laquelle le tri est réalisé. Le sujet et la glose n'apparaissent pas.

Lorsqu'il a généré un descripteur de marque, CGKAT le connecte par un lien hypertexte à l'ED indexé sans rajouter des marques d'index autour de cet ED[1]. Inversement, il connecte par un lien hypertexte de type To_Descriptor (cf. figure 6.12) cet ED au descripteur. Dans le modèle ExtIndex.S, il n'est pas déclaré d'élément permettant de rassembler les descripteurs relatifs à un ED. Nous n'avons pas jugé nécessaire de rajouter et de gérer un tel élément (i.e. une liste de descripteur). Donc, si plusieurs représentations ont été créées pour un ED, cet ED ne pointe pas vers une liste de descripteurs mais vers le descripteur correspondant à la première représentation que pointe l'ED via l'attribut référence To_Descriptor. Cependant,
1) depuis ce descripteur, cette première représentation peut être retrouvée par navigation, et comme cette première représentation est située dans la liste des représentations de l'ED, les autres représentations sont visibles ou facilement accessibles ;
2) si les représentations de l'ED sont différentes sur l'utilisateur, le point de vue, la source d'information ou encore le type de concept utilisé, cela sera visible dans la table d'index générée. En effet, les différents descripteurs correspondants aux différentes représentations de l'ED sont bien triés et synthétisés (élimination des doublons).

Si dans un document, plusieurs occurrences d'un mot ou d'une expression incluent une même représentation dans leur listes de représentations respectives, CGKAT génère néanmoins différents descripteurs (aux contenus identiques) pour cette représentation partagée. Ainsi, ces différents descripteurs sont synthétisés dans une entrée de la table d'index mais les origines dans le document des différents EDs indexés sont listées avec des numéros de pages et des liens hypertextes sont posés pour permettre l'accès à ces EDs.

Les index qui peuvent être générés à partir des descripteurs créés par CGKAT sont en quelque sorte des glossaires de représentations de termes. Ils offrent un moyen complémentaire aux liens structurels et aux liens hypertextes pour 1) naviguer entre informations et connaissances, 2) retrouver des connaissances, et 3) comparer des connaissances. Quant aux index thématiques, l'utilisateur de CGKAT peut en créer suivant les critères l'intéressant : il lui suffit de poser une requête conceptuelle et un document virtuel (une vue) est alors généré par CGKAT pour collecter les informations et/ou connaissances répondant aux critères spécifiés. Nous détaillons ce point dans la prochaine section.

1. Normalement dans Thot, un ED indexé par un descripteur de marques est entouré de marques d'index et un lien hypertexte de type Vers_marque1 relie le descripteur à la première marque. Cependant, nous avons noté en section 6.3.1.4.1 que nous ne désirions pas entourer de marques les ED symboles. Afin que CGKAT puissent connecter les descripteurs qu'il génère aux EDs indexés, nous avons modifié le modèle structurel ExtIndex.S de telle sorte que l'attribut Vers_marque1 puisse pointer sur n'importe quel ED et non uniquement sur une marque.

6.3.2 Recherche d'informations ou de connaissances dans CGKAT

Nous avons montré que dans CGKAT, des EDs peuvent contenir certaines représentations de connaissances (EDs GC, CGRepr et CGRequests) et qu'ils peuvent, comme tous les EDs, être reliés à d'autres EDs par des liens structurels, hypertextes et d'inclusion. Afin de permettre "l'indexation" d'EDs par des connaissances, nous avons de plus introduit des liens hypertextes de représentation et d'annotation qui peuvent connecter n'importe quel ED, liste d'EDs consécutifs ou portion de texte, à des EDs GC et CGRepr ou listes d'EDs GC ou CGRepr. Enfin, nous avons indiqué des contraintes sémantiques sur l'utilisation des liens de représentation, et nous avons montré comment des synthèses des représentations des EDs symboles pouvaient être générées. Nous allons maintenant rappeler ou détailler certains points sur la façon dont les divers liens entre ces éléments peuvent être exploités pour la recherche d'informations ou de connaissances.

6.3.2.1 Navigation

Thot permet de naviguer sur les liens structurels, hypertextes et d'inclusion, vers leur destination ou vers leur source. Les liens hypertextes et d'inclusion peuvent joindre des EDs du même document ou de documents différents. Un lien structurel ou hypertexte peut joindre un ED "classique" (e.g. paragraphe, section, graphique) à un ED contenant une connaissance, et vice-versa. La recherche d'informations peut donc se faire *via l'accès à des connaissances, puis des recherches parmi celles-ci (par navigation ou requêtes), et enfin l'accès depuis l'une d'elles aux informations auxquelles elle est liée*.

Dans de nombreux systèmes hypertextes actuels, chaque noeud du réseau contient un ED et est relié à d'autres noeuds par des liens structurels ou des liens hypertextes nommés, les noms de ces liens suggérant au lecteur la sémantique du lien. Chaque noeud est donc un représentant de l'ED qu'il contient, i.e. une poignée ou encore un descripteur de cet ED. Dans CGKAT, les connaissances sont elles-mêmes construites et stockées avec des EDs, et elles ne contiennent pas les EDs qu'elles indexent. Ainsi, chaque ED peut être représenté ou annoté par différents auteurs et selon différents points de vue. Un ED peut également contenir une connaissance indexant d'autres informations (e.g. figure 6.5), alors que dans la plupart des systèmes hypertextes, il ne semble pas possible qu'un ED contienne (et donc permette de visualiser) avec d'autres informations, une portion du réseau hypertexte (dans les systèmes hypertextes auxquels nous faisons référence, le réseau hypertexte est également le réseau sémantique avec lequel les connaissances sont représentées).

6.3.2.1.1 Navigation entre représentations de connaissances

Dans CGKAT, le réseau sémantique (la base de connaissances) est représenté avec des GCs.

Le réseau peut être représenté avec quelques GCs relatifs chacun à un certain point de vue, domaine, auteur, etc. Ces GCs peuvent ainsi atteindre une taille importante. Des portions de ces GCs peuvent être utilisées pour indexer des EDs (cf. section 6.3.1.4.3 : la réutilisation est actuellement possible mais non automatisée). La fonction de zoom de Thot peut être utile pour avoir une vision d'ensemble de ces "gros" GCs.

Le réseau peut également être modularisé en divers GCs de petite taille, e.g. des représentations de petits EDs. Dans ce cas, il est important de pouvoir naviguer de GC en GC, ne serait-ce que pour suivre un chemin de relations conceptuelles entre des concepts. Il faut donc depuis un concept pouvoir retrouver tous les GCs qui lient ce concept à d'autres concepts. C'est pour cela que nous encourageons les utilisateurs de CGKAT à construire des GCs avec des EDs ConceptInclusion, c'est-à-dire en "réutilisant" des concepts. Le partage des concepts entre GCs est ainsi explicitement représenté via la structure logique des documents permettant de stocker ces GCs, et la navigation est possible entre ces GCs. En effet, depuis un ED ConceptInclusion, le concept source peut être retrouvé, et depuis ce concept, tous les GCs qui l'incluent peuvent être retrouvés également. De plus, une modification de ce concept source est automatiquement répercutée dans tous les GCs qui l'incluent : cela est effectué par Thot dans les documents et par CGKAT dans la base de CoGITo. Un concept inclus peut bien sûr contenir un graphe dans sa partie référent.

Lorsqu'un utilisateur de CGKAT accède à un GC afin de connaître certaines des relations liant un concept à d'autres concepts, il peut visualiser, outre ces relations, le contexte dans lequel elles ont été représentées, c'est-à-dire :
1) le GC en entier et donc les contextualisations éventuelles représentées par un emboîtement de GCs, ou encore le fait qu'il s'agisse d'une définition de type,
2) si l'ED fait partie d'un CGRepr, son auteur, le point de vue considéré, et les informations de documentation,
3) les EDs pères ou frères du GC ou du CGRepr, e.g. sections, paragraphes ou d'autres EDs CGRepr indexant la même information.
L'utilisateur peut donc prendre en compte ce contexte lors de sa navigation. Ces notions de contexte ne sont pas développées dans les systèmes hypertextes actuels où, à notre connaissance, seuls les attributs d'un noeud permettent de lui associer certaines informations : les réseaux sémantiques sont plats (pas d'emboîtement de graphes) et des portions de réseaux sémantiques ne peuvent être contenues à l'intérieur d'EDs.

Nous discuterons plus loin de l'emploi dans CGKAT de liens hypertextes *virtuels*. Rappelons que l'activation d'un lien hypertexte virtuel entraîne l'exécution d'une requête prédéfinie ("requête" au sens large car il peut s'agir d'un script ou d'un programme complexe). L'exécution de cette requête crée et affiche un document ou une portion d'un document existant (la destination du lien hypertexte est ainsi créée et affichée).

6.3.2.2 Requêtes lexicales, structurelles et conceptuelles

La navigation permet de rechercher des informations ou des connaissances tout en découvrant les principaux liens qui les relient. Les requêtes permettent des accès directs à certains types d'informations ou de connaissances.

Thot permet des ***requêtes lexicales*** (recherche de chaîne de caractère dans des EDs) et ***structurelles*** (recherche d'EDs compte-tenu de leurs types, des types de leurs attributs et des valeurs de ces attributs). CGKAT permet de plus des ***requêtes conceptuelles***, i.e. sur le contenu conceptuel des EDs : dans CGKAT, ces requêtes sont basées sur les connaissances représentées sous forme de GCs. Nous ne décrivons ci-dessous que de ces requêtes conceptuelles.

Ces requêtes sont basées sur le contenu de la base de GCs. Or ces GCs, s'ils indexent des EDs, sont stockés dans des documents et sont chargés dans la base lors de l'ouverture de ces documents. Ainsi, lorsque l'utilisateur recherche des connaissances, ou des informations via leur indexation par des connaissances, il peut contrôler de manière fine l'***espace de recherche*** en ouvrant ou fermant les documents contenant les GCs. Il peut également charger des GCs dans la base à partir de fichiers scripts ou de fichiers de sauvegarde au format BCGCT (le format de sauvegarde utilisé par CoGITo ; cf. annexe 8), mais de tels GCs n'indexent pas des EDs. Notons enfin que dans CGKAT, l'utilisateur peut choisir d'effectuer ses recherches soit dans l'espace des GCs proprement dit, soit dans celui des graphes servant aux définitions de types (i.e. dans l'espace des lambda-abstractions), soit dans les deux à la fois.

Dans CGKAT, une requête conceptuelle doit être écrite dans un ED Request (un sous-élément d'un ED CGRequests) et les résultats de la requête sont affichés dans l'ED Answers qui le suit (si cet ED n'a pas été préalablement détruit). L'utilisateur est ainsi libre de stocker ses requêtes et leurs résultats à l'intérieur du ou des documents de son choix. Cette solution nous a semblé plus pratique pour l'utilisateur que la génération d'un document temporaire différent à chaque exécution d'une requête afin d'afficher ses résultats.

Nous avons montré que grâce au langage de commandes de CGKAT et à son exploitation du shell, les commandes de recherche peuvent être combinées de manière complexe avec d'autres commandes, e.g. jointure maximale, chargement de fichiers, destructions de graphes, et programmes accessibles depuis le shell. Les requêtes peuvent donc être complexes. Nous ne présentons ci-après que les commandes de recherche.

Actuellement dans CGKAT, seules les relations de spécialisation entre GCSs et entre méta-informations sont exploitées dans les recherches par requête[1] (voir figure 6.11 à la page 183 ou les figures 6.21 à 6.24). Dans les systèmes hypertextes actuels, ce sont plutôt des recherches de sous-graphes, et notamment des chemins de relations entre concepts (noeuds), qui sont effectuées par requête. Nous montrerons en section 6.3.2.2.6 comment cette approche pourrait être intégrée dans CGKAT. Par ailleurs, dans CGKAT, la recherche des spécialisations d'un GC s'effectue en tentant de projeter ce GC requête sur chaque graphe de la base. Pour effectuer une telle recherche dans une base de graphes conséquente, CGKAT devrait plutôt exploiter une indexation des graphes, par exemple avec une indexation par hash-coding (Ellis & Lehmann, 1994) (Cogis & Guinaldo, 1995) ou par l'implémentation de la relation de spécialisation entre graphes dans une structure de donnée (e.g. UDS (Levinson, 1994)).

La relation de spécialisation qui peut être calculée entre les GCs permet un accès "par le contenu" à des connaissances, c'est-à-dire sans avoir à connaître les noms exacts des connaissances ou de leurs composants, contrairement à la plupart des bases de données où l'utilisateur doit connaître les noms exacts des tables et de leurs champs. Cependant, pour créer un GC requête, l'utilisateur doit connaître certains noms de types et éventuellement de référents. Pour cela, dans CGKAT, l'utilisateur peut être aidé par les menus de gestion de référents et de types de concepts et de relation. Nous avons également permis une approche plus rapide : l'utilisation dans les GCs requêtes de noms incomplets pour les types de concepts, ces noms incomplets devant correspondre à des sous-chaînes de noms de types.

6.3.2.2.1 Utilisation de sous-chaînes de noms de types dans les GCs requêtes

Lors d'une requête, à la place d'un nom de type de concept, l'utilisateur peut donner une chaîne de caractères quelconque (à condition de la précéder de '%') : CGKAT va rechercher dans la hiérarchie des types de concepts, tous ceux dont le nom comporte cette chaîne de caractères. Plusieurs types pouvant être trouvés pour chaque chaîne, CGKAT va générer autant de GCs requêtes que de combinaisons possibles. Les combinaisons non valides, e.g. ne satisfaisant pas les signatures des relations utilisées dans la requête initiale, sont éliminées. Les requêtes correspondant aux combinaisons valides sont ensuite exécutées les unes après les autres (voir figure 6.21 à la page 209).

Cette facilité est particulièrement intéressante lorsque les noms des types de l'ontologie utilisée sont construits par concaténation de divers mots synonymes, comme c'est par exemple le cas pour les types de concepts provenant de WordNet. En effet, dans ce cas, les chances sont plus grandes de retrouver à partir d'un mot, le type représentant le sens auquel pense l'utilisateur. Notons qu'il est inutile que la recherche de types de concepts comportant une chaîne de caractère donnée, se fasse en dehors de l'ontologie utilisée au moment de la requête, par exemple dans WordNet tout entier. En effet, les types des GCs recherchés, i.e. ceux de la base utilisée au moment de la requête, font tous partie de l'ontologie utilisée au moment de la requête.

Si comme c'est le cas dans notre ontologie, les types de relations restent assez élémentaires, bien moins nombreux que les types de concepts, et ne sont pas constitués par concaténation de mots synonymes, l'extension de cette facilité aux types de relations ne semble pas vraiment nécessaire car elle serait probablement assez peu utilisée ou sans succès (pas de type retrouvé). C'est pourquoi nous ne l'avons pas implémentée.

6.3.2.2.1.1 Perspective : exploitation de liens hypertextes entre des mots et des types

Si dans une version ultérieure de CGKAT, les hiérarchies de types sont, de la même façon que les

1. Une requête ne peut donc contenir qu'un GCS et non un graphe emboîté ou une définition de type, mais des graphes emboîtés ou des définitions de types peuvent néammoins être ainsi retrouvés. La recherche peut s'effectuer sur les graphes servant aux définitions de types, et la méta-information "defKind" permet de poser des contraintes sur les sortes de définitions recherchées (e.g. "defKind:TC" $\rightarrow$ "définitions par conditions typiques").

GCs, construites et stockées dans des documents structurés, il sera possible de connecter par des liens hypertextes, chaque type à un ED rassemblant la liste des mots dont l'un des sens correspond à celui représenté par le type. Une telle liste pourra être structurée de manière adéquate, e.g. par langue et/ou par utilisateur. Inversement, les entrées d'un glossaire de mots pourront être connectées aux types correspondant aux sens de ces mots. Alors, il sera intéressant de permettre l'exploitation de ces relations entre mots et types. Par exemple, quand l'utilisateur écrira un GC en utilisant à la place de certains types de concepts, un mot précédé de '@', le mot pourra être recherché dans le glossaire de mots, puis divers GCs requêtes pourront être générés en utilisant chacun des types correspondant au sens du mot. Une alternative serait de créer un système de menus guidant l'utilisateur dans la recherche des types adéquats pour sa requête, mais les menus de gestions des types de concepts ou de relations peuvent déjà jouer ce rôle.

6.3.2.2.2 Présentation des résultats : GCs et/ou informations indexées par ces GCs

L'utilisateur peut poser une requête pour rechercher des connaissances ou pour rechercher des informations via leur indexation par des connaissances. Il doit donc spécifier avant de poser sa requête, quelles sortes de résultats il désire.

Lorsqu'un graphe satisfaisant aux critères d'une requête :

1. Ce graphe peut être affiché au format linéaire.
2. Si le graphe a été construit via un ED GC, cet ED GC peut être affiché.
3. Si le graphe représente un ED (i.e. s'il existe un lien de reprsentation entre l'ED et l'ED GC contenant le graphe), cet ED représenté peut être affiché.
4. Si le graphe annote un ED (i.e. s'il existe un lien de reprsentation entre l'ED et l'ED GC contenant le graphe), cet ED annoté peut être affiché.
5. Enfin, pour permettre à l'utilisateur d'exploiter d'autres systèmes d'indexation que celui que nous avons mis en place (i.e représentation/annotation), CGKAT propose une cinquième option : présenter tous les EDs qui pointent par un attribut référence (quel qu'il soit) vers l'ED GC ayant permis de construire le graphe. Un raffinement de cette dernière option serait de permettre à l'utilisateur de spécifier le ou les types d'attributs référence dont il souhaite la prise en compte pour la présentation des résultats de requêtes. Cela suppose la déclaration dans un modèle structurel d'autres types d'attributs référence que ceux de représentation et d'annotation. Actuellement, seul le type d'attribut référence To_Anything peut être utilisé en complément des types de liens hypertextes définis et gérés dans CGKAT pour la représentation et l'annotation.

Les figures 6.21 à 6.24 illustrent différentes manières dont les résultats de recherches peuvent être présentés. L'utilisateur spécifie son choix avec la commande "use" et un paramètre qui peut être :
- "linear" (forme linéaire),
- "CGs" (les EDs GC ayant servi à construire les graphes résultats),
- "Repr" (les EDs représentés),
- "Annot" (les ED annotés),
- "Assoc" (les EDs qui pointent vers les EDs GC avec un attribut référence),
- "Repr&CGs" (les EDs représentés plus leurs représentations),
- "Repr&Annot", "Annot&CGs", "Repr&Annot&CGs".

Si l'option "CGs" est choisie et que des graphes résultats n'ont pas été construits via des EDs GC, la forme linéaire est utilisée. Lorsqu'une option contenant "Repr", "Annot" ou "Assoc" est choisie mais qu'un graphe résultat n'indexe pas d'ED suivant la ou les manières spécifiées, cela est signalé par CGKAT et le graphe est affiché avec un ED GC ou au format linéaire s'il n'a pas déjà été présenté. Cela permet de voir que des GCs n'ont été associés à aucun ED par certains liens.

Dans un document, plusieurs EDs peuvent inclure une même ED CGRepr dans leurs listes de représentation ou d'annotation respectives. Mais cet ED CGRepr ne pointe par le lien To_Element

que vers un seul ED (le premier qui a été indexé, les indexations suivantes réutilisant la première indexation). Or, c'est ce lien qui est suivi pour retrouver un ED indexé et pouvoir ainsi l'afficher. Si l'ED CGRepr qui est à la source du lien fait partie d'une liste d'annotations, l'indexation est reconnue par CGKAT comme une annotation. Sinon, l'indexation est reconnue par CGKAT comme une représentation. Dans tous les cas, pour chaque graphe résultat, un seul ED indexé au plus est retrouvé et donc affiché. Nous avons fait ce choix car il semble redondant d'afficher différents EDs qui véhiculent une même information (celle correspondant à l'objet de la requête) puisqu'ils ont été indexés par une même représentation. Compte-tenu de ce choix, si l'utilisateur désire voir les EDs qui ont une représentation semblable, il ne faut pas que leurs listes de représentation ou d'annotation respectives "incluent" une même représentation mais qu'elles comportent des représentations ayant un contenu semblable. Dans ce cas, plusieurs graphes au contenu semblable existeront dans la base de GCs mais chacun indexera un ED différent.

CGKAT élimine les doublons d'EDs ou de GCs dans la liste des résultats d'une requête : si un ED ou un GC existent déjà dans la liste, il n'est pas rajouté. Cependant, CGKAT peut présenter différents EDs ou GCs au contenu identique : le test est fait sur les identificateurs des EDs et les noms des graphes, pas sur leur contenu.

Lorsque des EDs doivent être présentés (EDs GC ou EDs indexés), ils le sont en utilisant des inclusions. Ainsi, les résultats d'une requête forment une vue (générée selon une combinaison arbitrairement complexe de critères conceptuels choisis par l'utilisateur) sur des parties de documents, et dans le cas d'EDs GC, sur une partie de la base. La génération de ces vues permet de combiner recherche par requête et recherche par navigation puisque de telles vues sont le point de départ de nombreux liens hypertextes (vers les EDs inclus et leurs contextes dans divers documents). En d'autres termes, la génération de vues est une façon de focaliser l'espace de recherche par navigation.

/usr/tmp/aaaa22939 MainView

Document Edit Views Search Presentation Attributes Selection Tools

Selection AnyElement \ Request \ RequestAnswer \ CGRequests

> on CGs //the following searches must be done only on the CGs in the base and not on the lambda-abstractions
//(this is the default option in CGKAT)

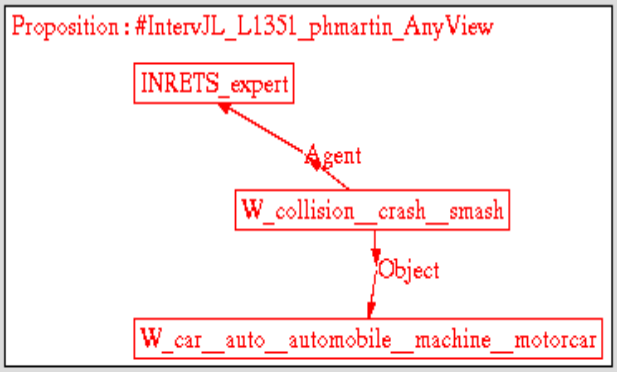
Ok

> use CGs //For the following searches of CGs, if a CG has been built via a DE CG, use it for displaying the CG

Ok

> spec [%car]<-(Object)<-[%crash] //search specialisations of this CG

--- Generated request: spec [W_car__auto__automobile__machine__motorcar]<-(Object)<-[W_collision__crash__smash]



> use Repr //For the following searches of CGs, if a CG represents a DE, display this DE instead of the CG

Ok

> spec [W_car__auto__automobile__machine__motorcar]<-(Object)<-[W_collision__crash__smash]

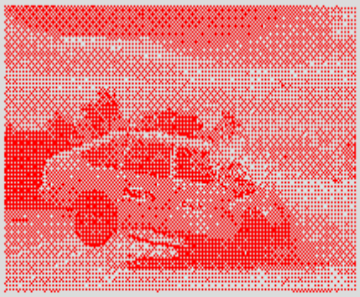


Fig. 0 A test of car crash at the INRETS

>

Figure 6.21: *Recherche de crashs d'automobiles dans la base de GCs (seul le document IntervJL ayant été ouvert) et affichage des résultats (ici il n'y en a qu'un) sous différentes formes.*

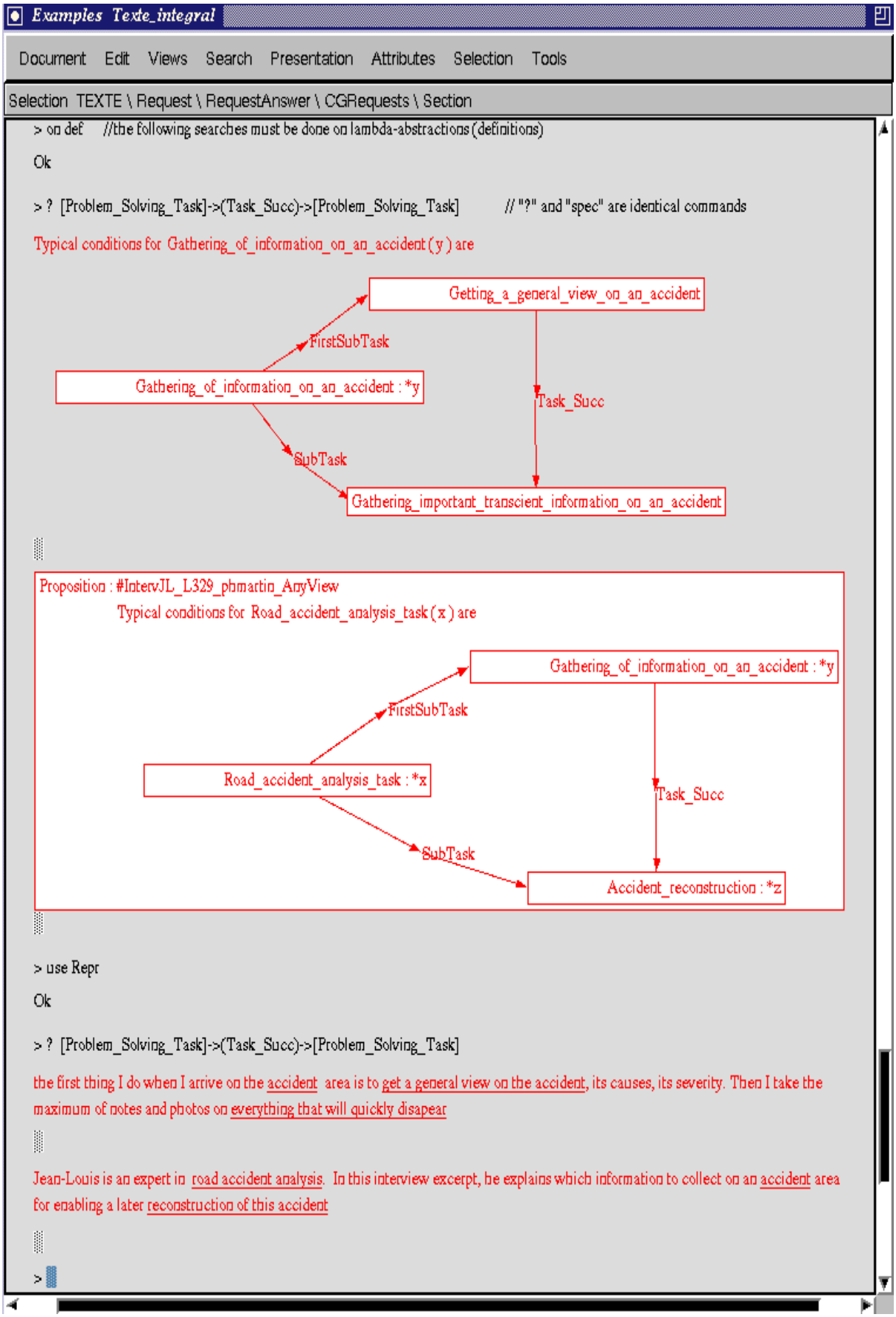


Figure 6.22: *Recherche de successions de tâches parmi les représentations sous forme de définitions de types (seul le document IntervJL ayant été ouvert) et affichage des résultats sous différentes formes.*

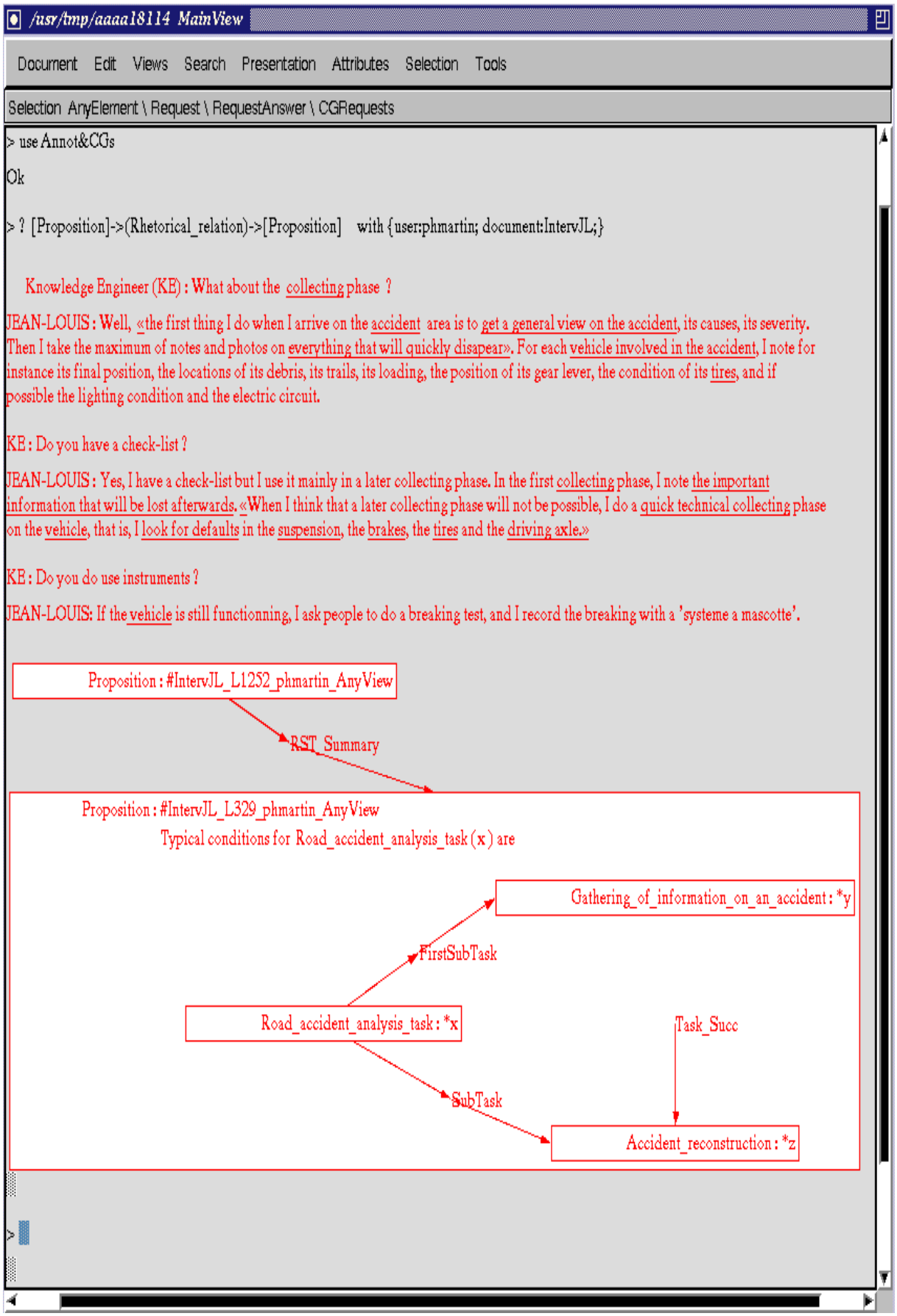


Figure 6.23: *Recherche de graphes créés par l'utilisateur "phmartin" dans le document "intervJL", et contenant au moins une relation rhétorique. Affichage du graphe trouvé et de l'ED qu'il annote.*

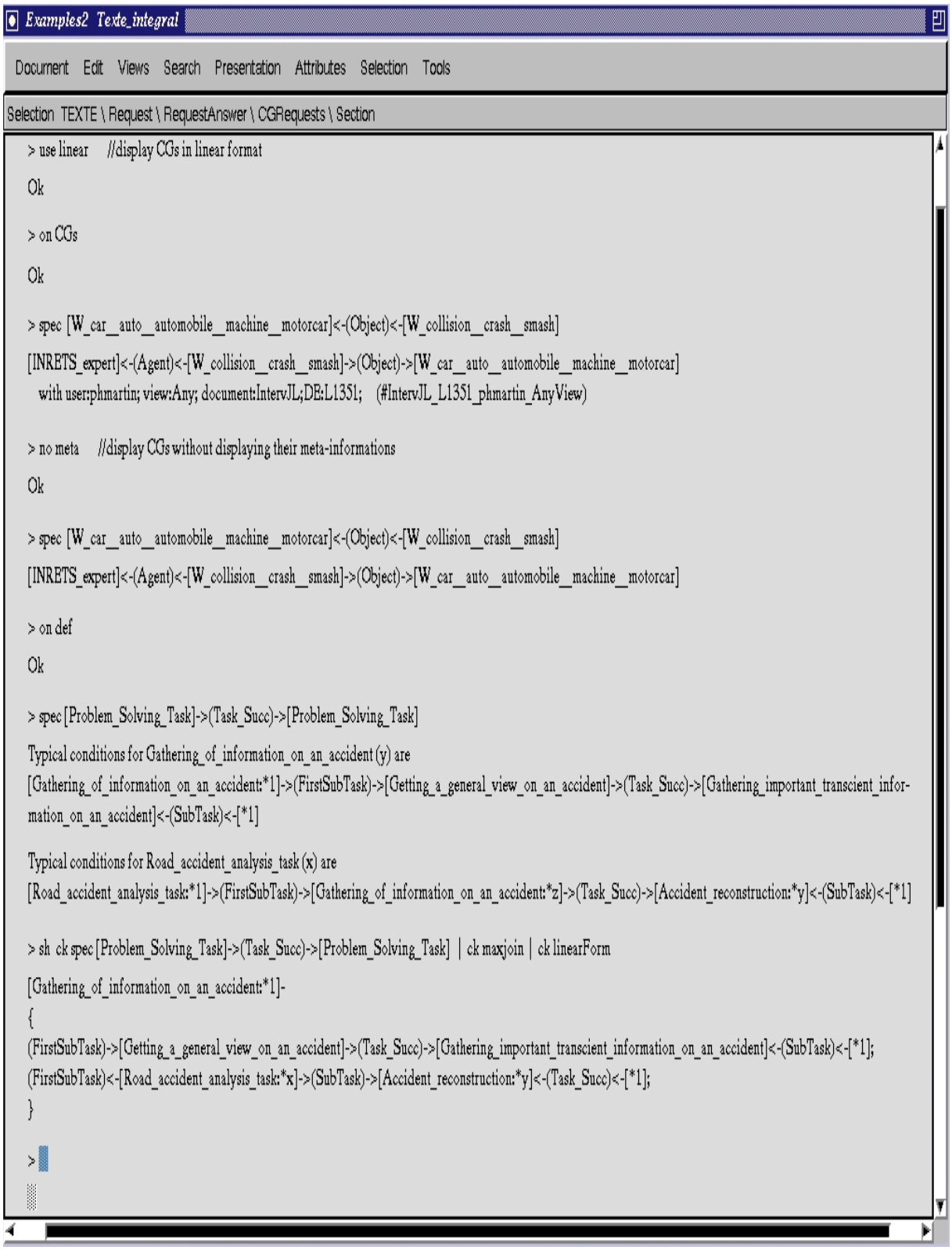


Figure 6.24: *Recherche de graphes et de définitions de types (seul le document IntervJL ayant été ouvert) et affichage au format linéaire.*

La dernière commande réalise également une jointure maximale sur les corps de définitions de types retrouvées. Dans de tels cas, la jointure maximale que nous avons implémentée (cf. section 7.2.3.4) retourne un graphe et non une définition de type (quel devrait être ce type sinon et de quelle sorte serait la définition : par conditions nécessaires et suffisantes, par conditions typiques, ... ?).
Il appartient à l'utilisateur de transformer ce graphe, e.g. en l'instanciant ou en le transformant en une définition de type, de façon à représenter une connaissance. Sinon, ce graphe doit être détruit puisqu'il ne représente pas une connaissance.

6.3.2.2.2.1 Perspectives : présentation des résultats sous une forme hiérarchique

Les résultats d'une requête conceptuelle peuvent être nombreux. Cependant, dans ce cas, certains résultats peuvent en spécialiser d'autres, ou des généralisations peuvent être calculées afin de regrouper ces résultats dans une structure hiérarchique. Une perspective intéressante pour la présentation de ces résultats serait donc de les présenter sous une forme hiérarchique. Pour cela, un ou plusieurs EDs Arbre pourraient être utilisés ou bien la disposition pourrait être calculée par programme. Un système de navigation dans une hiérarchie de résultats pourrait également être implémenté.

6.3.2.2.2.2 Perspectives : génération d'EDs par intégration de commandes d'édition de documents à notre langage de commandes

Le langage de commandes actuel permet de générer du texte et des EDs GC

Le langage de commandes de CGKAT permet d'exécuter un script de commandes. Un tel script génère une partie d'un document, et peut par exemple être destiné à répondre à une question fréquemment posée. S'il s'agit d'un script shell, l'affichage et l'interaction avec l'utilisateur peuvent être contrôlés via les commandes du script (comme dans tout script shell)[1].
La commande "display" prend en paramètre des noms de graphes. Si le script a été lancé depuis CGKAT, cette commande permet d'afficher les graphes avec des inclusions des EDs GC ayant servi à créer ces graphes. Si les graphes n'ont pas été construits via des EDs GC, ou si le script a été lancé depuis une fenêtre shell, les graphes sont affichés au format linéaire (un exemple de script utilisant la commande "display" est donné dans la note 1 à la page 223). A part des EDs GC, une partie de document générée via un script, ne peut actuellement contenir que du texte.

L'intégration de commandes d'édition structurées permettrait de générer divers sortes d'EDs

Dans le projet OPÉRA, auteur de Thot, des travaux sont actuellement effectués sur un langage de commandes d'édition de documents structurés (Bois & Richy, 1996). Ces commandes pourraient être intégrées à notre langage de commandes et ainsi permettre la génération d'EDs structurés et de liens hypertextes entre ces EDs.

Cela serait particulièrement intéressant pour la génération d'explications ou de documents techniques. En effet, la génération d'explications ou de documents en général, sous la forme d'un texte linéaire, pose de nombreux problèmes car a) il faut éviter de donner à l'utilisateur des informations déjà connues de lui, et b) il faut calculer un ordre satisfaisant pour la présentations d'informations. L'utilisation d'un document structuré hypertexte permettrait d'*éviter en partie* ces problèmes (et non de les résoudre) :

1. De nombreuses informations "annexes", comme par exemple des définitions de termes ou encore des preuves pour des résultats de raisonnement, peuvent être stockées dans des EDs à part mais connectées par des liens hypertextes et/ou présentées sous forme de notes. Ces EDs peuvent également être liés à d'autres EDs.
2. Les informations peuvent être structurées en différentes catégories (via divers EDs) organisées hiérarchiquement, e.g. par niveau de détail. Dans ce cas, un éditeur tel que Thot peut permettre d'avoir une vision synthétique sur ces catégories : génération de tables des matières, systèmes de

1. Depuis CGKAT, une commande shell peut être lancée à condition qu'elle soit préfixée par 'sh". Un script shell peut ainsi être exécuté (et ses résultats apparaissent alors dans l'ED RequestAnswers où a été lancé le script). Dans un tel script, l'appel à des commandes de CGKAT se fait en les préfixant par "ck", les autres commandes ne sont pas préfixées.
Une commande de CGKAT permet également d'exécuter des "scripts CGKAT". Dans ce cas, chaque ligne de commande du script est affichée puis exécutée comme si elle avait été tapée depuis CGKAT. Il s'agit donc en quelque sorte d'une exécution en mode "trace" d'un ensemble de commandes de CGKAT (les commandes du shell sont préfixées par "ck" afin d'être appelées). Compte-tenu de ce mode "trace", les scripts CGKAT sont plus intéressants pour la mise au point que pour la génération d'explications ou de documents techniques.

vues, navigation rapide sur les divers éléments d'une liste (e.g. des sections de même niveau ou des paragraphes).

Il serait également possible, quoique plus difficile, de prévoir dans un script, la *modification* de certains EDs en fonction du contenu de certains GCs (par exemple la modification, compte-tenu du contenu des GCs résultats d'une requête, des EDs qu'ils représentent). Le cas le plus difficile serait probablement la construction ou la modification depuis un script d'un ED contenant un graphe, e.g. un ED CGgraph. En effet, le parcours et la création ou la mise à jour, via une interface fonctionnelle, de la structure logique d'un graphe et de ses composants, n'est pas une tâche simple (le code de CGKAT à ce sujet en est un bel exemple). Il serait alors utile de pouvoir utiliser une commande prenant en paramètre un GC au format linéaire et *générant ou modifiant* à partir de celui-ci, un ED CGgraph à un emplacement spécifié dans un document (la disposition des concepts devrait également être déterminée automatiquement).

Notons qu'un document généré peut toujours être édité, structuré et complété à la main pour créer par exemple une documentation technique ou un document intermédiaire lors de l'acquisition des connaissances.

6.3.2.2.3 Présentation de graphes emboîtés

Rappelons que CGKAT permet de construire des GCs emboîtés via des EDs GC mais les stocke dans CoGITo sous la forme de GCSs (le lien entre un concept et le graphe qu'il emboîte est effectué via le référent individuel de ce concept dont l'étiquette doit être identique au nom du graphe). Aussi, par abus de langage et pour simplifier la description ci-dessous, nous dirons qu'un GCSs peut emboîter un autre GCS (il s'agit en fait d'un emboîtement de niveaux dans un GC). Notre méthode de représentation de GCs emboîtés permet leur construction via la construction de GCSs, avec des EDs GC ou sous forme linéaire (cf. section 4.3.2.4.1).

Lorsqu'un *GCS emboîtant d'autres graphes est présenté* avec un ED GC, ces graphes sont visibles dans les parties référents des concepts qui les emboîtent. Ce n'est pas le cas lorsque la forme linéaire est utilisée, mais grâce aux étiquettes des référents individuels, l'utilisateur connaît les noms des premiers GCSs emboîtés et peut donc les retrouver.

Lorsqu'un GCS est emboîté dans un autre GCS qui le contextualise[1], et doit être présenté en réponse à une requête conceptuelle, CGKAT affiche ce GCS emboîtant (le GCS emboîté est donc également affiché si un ED GC est utilisé). Plus exactement, comme il peut y avoir plusieurs emboîtements contextualisants successifs, CGKAT affiche le GCS emboîtant qui n'est pas lui-même contextualisé. Par exemple, "Possible" et "Believer" étant des types de relations définis comme contextualisants, si le graphe suivant est dans la base, il sera présenté en réponse à la requête "spec [Cat]" :

(Possible)←[Proposition: #p2 [Person:#John]←(Believer)←[Proposition: #p1 [Cat]→(On)→[Mat]]]

CGKAT permet à différents GCSs de réutiliser (ou partager) un même concept. Ainsi, en partageant un concept contenant un graphe, différents GCs peuvent emboîter le même graphe. Notons que le concept partagé est identique (même type, même référent individuel) dans tous les graphes qui le partagent. Jusqu'à présent, CGKAT n'interdit pas qu'un même graphe soit contextualisé par différents GCSs. Si un tel graphe doit être affiché parce qu'il est le résultat d'une requête, chacun de ses GCSs emboîtants est présenté par CGKAT. Plus exactement, si ces GCSs sont eux-mêmes contextua-

1. Rappelons que CGKAT considère qu'un ***GC est contextualisé si***
1) un des GCSs emboîtants contient au moins une relation dont le type est sous-type de Contextualizing_relation,
ou 2) si le type de l'un des concepts emboîtants est sous-type de Contextualizing_proposition,
ou 3) si l'un des concepts emboîtants est lié par une relation de type Object (ou d'un de ses sous-types) à un concept de type Process_contextualizing_its_object (ou d'un de ses sous-types).
L'utilisateur peut à tout moment ajouter ou ôter des sous-types à Contextualizing_relation, Contextualizing_proposition et Process_contextualizing_its_object, et ainsi changer les résultats d'une requête.

lisés, ce sont leurs GCSs les plus emboîtants (et donc non contextualisés) qui sont affichés.
Par exemple, si seuls les cinq graphes suivants sont présents dans la base, seuls les trois derniers seront présentés en réponse à la requête "spec [Cat]" :

[Proposition: #p1 [Cat]→(On)→[Mat]]. //ici, le graphe p1 n'est pas contextualisé.
[Proposition: #p2 [Person:#John]←(Agent)←[Believe]→(Object)→[Proposition: #p1]].
(Possible)←[Proposition: #p2].
[Person:#Jacques]←(Believer)←[Proposition: #p1].
[Temporal_entity: #14.06.95]← (Point_in_time)←[Situation]→(Descr)→[Proposition: #p3].

La commande "context" permet à l'utilisateur de voir afficher les graphes résultats d'une requête à l'intérieur des GCSs les emboîtant même si ces derniers ne sont pas contextualisants. Ainsi, si plusieurs GCSs réutilisent un graphe défini en partie référent d'un concept unique, tous ces GCs peuvent être présentés. La figure 6.22 donne un exemple d'utilisation de la commande "context". La commande "no context" permet de revenir à l'option par défaut (présentation des graphes résultats à l'intérieur des GCs les emboîtant, seulement si ces derniers les contextualisent).

Notons que les cinq options de présentation de résultats que nous avons décrites à la page 207 sont relatives au graphe qui doit être présenté : ce graphe peut être un graphe emboîtant celui qui répond au critères de la requête soit parce qu'il le contextualise, soit parce que la commande "context" a été préalablement lancée.

De même que les graphes spécialisant un graphe requête sont recherchés dans la base en cours (i.e. dans les documents ouverts, si la base est construite à partir de ces documents), les graphes emboîtant un graphe résultat sont recherchés dans la base en cours. Une autre possibilité aurait été de stocker automatiquement dans le champ commentaire d'un GCS emboîté, les *références* de tous graphes qui l'emboîtent (quels que soient les fichiers ou documents à partir desquels ils ont été chargés), *c'est-à-dire les informations nécessaires pour charger ces graphes* dans la base à partir d'un fichier ou d'un document, s'ils ne sont pas présents dans la base au moment de la requête. Cela aurait impliqué une gestion des graphes bien plus complexe que celle actuelle. La section 7.2.3.3 présente la manière dont est actuellement implémenté la recherche et la présentation des résultats de requêtes conceptuelles.

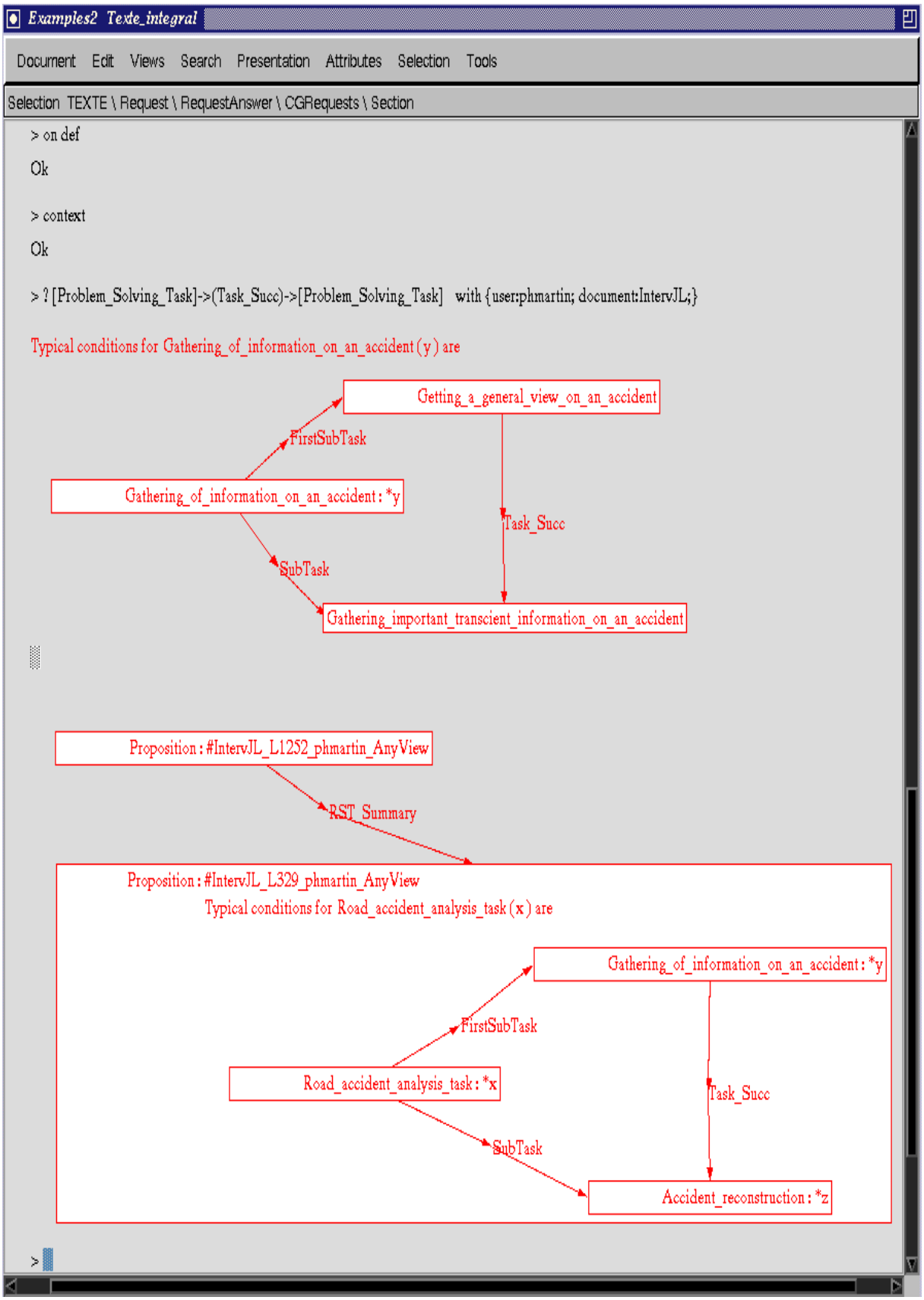


Figure 6.25: *Recherche de successions de tâches parmi les représentations sous forme de définitions de types créées par l'utilisateur "phmartin" dans le document "IntervJL", et affichage systématique des graphes résultats à l'intérieur des graphes les emboîtant (commande "context").*

Cette figure est à comparer avec la figure 6.22.

6.3.2.2.4 Recherche de généralisations et tests de généralisation

La commande "spec" permet de rechercher les spécialisations d'un graphe requête auquel peuvent être associées des méta-informations. La commande "gene" permet de rechercher ses généralisations. Les différentes options que nous avons indiquées pour la présentation des résultats d'une requête conceptuelle s'appliquent bien sûr à ce type de requête.

Le langage de commandes permet également de tester si un GCS en généralise un autre, ou si un type en généralise un autre. Les généralisations et spécialisations d'un type peuvent également être affichées en réponse à une requête, tout comme dans les menus de gestions des types de concepts ou de relations (la différence est que les résultats des requêtes peuvent être sauvegardées).

6.3.2.2.5 Perspective : recherche de concepts individuels via la recherche de spécialisations

L'utilisateur peut demander les spécialisations d'un graphe dans le seul but de savoir s'il existe un individu répondant aux critères spécifiés (ou plus exactement, s'il a été représenté). Dans ce cas, l'utilisateur ne souhaite pas l'affichage de graphes entiers, mais seulement l'affichage de certains concepts individuels.

Pour permettre cela, Sowa propose que l'utilisateur spécifie dans un graphe requête, le concept sur lequel porte la requête, en mettant la marque "?" à la place de son référent existentiel. Ainsi par exemple, si la base contient le graphe "[Cat: #Tom]→(On)→[Mat]→(Color)→[White]", une réponse à la requête "[Cat: ?]→(On)→[Mat]" pourra être "[Cat:#Tom]". Cela n'est pas encore implémenté dans CGKAT : actuellement, notre langage de commandes permet seulement de tester si un marqueur individuel est conforme à un type de concept.

On peut imaginer que la marque "?" puisse être posée dans plusieurs concepts afin de spécifier les sous-parties désirées des graphes résultats ; ainsi en reprenant l'exemple précédent, avec la requête "[Cat: ?]→(On)→[Mat:?]", c'est "[Cat:#Tom]→(On)→[Mat]" qui serait présenté et non le graphe "[Cat: #Tom]→(On)→[Mat]→(Color)→[White]".

6.3.2.2.6 Perspective : recherche de sous-graphes et de séquences

Dans les systèmes hypertextes actuels, le réseau hypertexte n'est pas séparé en divers graphes (ou du moins il n'est pas considéré comme tel pour les recherches par requête). Les requêtes sur les noeuds et/ou liens de ce réseau s'effectuent avec des expressions logiques sur des valeurs d'attributs et/ou des descriptions de sous-graphes (cf. sections 3.1.2.2 et 3.1.2.3).

Le réseau hypertexte étant considéré comme composé d'un seul graphe, si une recherche de spécialisation d'un graphe requête était effectuée de manière similaire à une recherche de spécialisation d'un graphe requête sur une base de GCs, le réseau entier devrait être présenté en cas de solution. Pour éviter cela, seules les sous-parties du réseau homéomorphes (Fortune & al., 1980) au graphe requête et spécialisant ses étiquettes, doivent être présentées. Certains langages de requêtes dans ces systèmes hypertextes prennent en compte une relation de spécialisation sur les étiquettes des noeuds et/ou des relations, e.g. WebTalk (Nanard & Nanard, 1991) (cf. section 3.1.2.3.6 à la page 53). Ils permettent également de rechercher des chemins contenant des séquences de certains noeuds et relations. Des expressions régulières (Hopcroft & Ullman, 1979) sont souvent utilisées dans les langages de requête pour la description de séquences de noeuds et de relations dans des graphes, par exemple dans GraphLog (Consens & Mendelzon, 1990) (cf. section 3.1.2.3.3 à la page 51).

Dans la prochaine section (6.3.2.2.6.1), nous proposons une manière d'étendre la syntaxe des graphes requêtes dans le langage de commandes de CGKAT, afin de permettre la recherche de sous-graphes dans un graphe. Nous comparons alors les possibilités de ce langage avec ceux d'autres systèmes hypertextes. Dans la section 6.3.2.2.6.2 nous étendons cette étude à la recherche de sous-graphes dans divers graphes, et nous spécifions la syntaxe d'une nouvelle commande destinée à faciliter cette recherche.

6.3.2.2.6.1 Recherches de sous-graphes dans un seul graphe

Nous supposons dans cette section que la base de GCs n'est composée que d'un seul "gros" GC, et que la commande "?" suivie d'un graphe requête sert à rechercher dans ce "gros" GC des sous-graphes isomorphes au graphe requête, mais dont les étiquettes des sommets concepts et relations peuvent spécialiser celles des sommets du graphe requête.

Actuellement dans notre langage de commandes, la syntaxe à utiliser pour les graphes requêtes est la même que pour construire des GCs. Afin de rechercher des sous-graphes dans le GC de la base, la syntaxe des graphes requêtes pourrait être étendue afin de permettre la description de séquences. Prenons l'exemple de la requête suivante :

> ? [Cat]- {→(Spatial_relation)→; }* [Mat] //un résultat peut être "[Cat]→(On)→[Mat]"

On peut imaginer que cette requête signifie : "rechercher les sous-graphes contenant un chemin de noeuds concepts et d'arcs relations, dont le premier noeud est un concept de type Cat (ou d'un de ses sous-types), le dernier est un concept de type Mat, et les arcs sont des relations binaires de type Spatial_relation (ou d'un de ses sous-types) ; l'extrémité initiale de la première relation est un concept de type Cat et l'extrémité finale de la dernière relation est un concept de type Mat".

Voici trois autres exemples.

> ? [Accident]→(First_phase)→[Accident_phase]-
{→(Next_phase)→;}*[Accident_phase]→(Location)→[Spatial_entity]
//recherche des sous-graphes reliant un accident à l'emplacement d'une des phases de d'accident

> ? [A]- {→(R1)→[B1]→(R2)→; //recherche des chemins reliant un concept [A] à un concept [D] via
→(R3)→[B2]→(R4)→;}*[D] //des séquences "→(R1)→[B1]→(R2)→" ***et*** "→(R3)→[B2]→(R4)→"
{→(R5)→[B3]→(R6)→;}*[E] //***ou*** à un concept [E] via des séquences "→(R5)→[B3]→(R6)→"
//(ex: [A]→(R5)→[B3]→(R6)→[X]→(R5)→[B3]→(R6)→[E])

> ? [A]- {→(R)→}*2+ [B] //recherche des chemins reliant un concept de type A à un concept de type B
//via au moins 2 relations de type R

Diverses autres conventions peuvent être adoptées. Dans tous les cas, de telles extensions sur la manière d'écrire un graphe requête ne sont pas des extensions au formalisme des GCs, c'est-à-dire à la façon de représenter des connaissances dans ce formalisme.

Comparaison avec d'autres langages de requêtes

Amann (1994) note que l'usage d'expressions régulières ne permet pas par exemple de connecter des chemins de deux ensembles de chemins différents, ou de garder seulement des sous-chemins d'un ensemble de chemins. Aussi, en plus de descriptions de chemins via des expressions régulières, il définit pour un langage d'interrogation les opérations suivantes : sélection, projection, renommage, jointure, concaténation, union, intersection et différence. Il illustre ces opérations en utilisant une syntaxe à la SQL. Nous avons donné quelques exemples dans la section 3.1.2.3.5. Voici l'un d'eux :

SELECT (Phase_de_accident phase_suivante)* Phase_de_accident //"projection" selon Amann
FROM Accident première_phase (Phase_de_accident1 phase_suivante)* Phase_de_accident2
WHERE EXISTS emplacement (Phase_de_accident2) = "Le rond-point des Lilas"

Dans CGKAT, les commandes de jointure simple et de jointure maximale sont respectivement proches des opérateurs de concaténation et de "jointure" chez Amann. La recherche de spécialisation correspond à la sélection chez Amann. La prise en compte des référents variables correspondrait au renommage chez Amann, et la prise en compte de la marque "?" dans les graphes requête correspondrait à la "projection" chez Amann. Les opérations d'union, d'intersection et de différence chez Amann peuvent être effectuées dans CGKAT en réutilisant des commandes du shell. Voici comment pourrait être effectuée dans CGKAT la requête précédente si la description de séquences et la marque "?" étaient prises en compte :

> ? [Accident]→(First_phase)→[Accident_phase:?]-
{→(Next_phase)→;}*[Accident_phase:?]→(Location)→[Spatial_entity:#Le_rond_point_des_Lilas]

Avec les requêtes précédentes (celles que nous venons d'illustrer et celles de Amann), chaque chemin retrouvé constituait un résultat. Le langage de requêtes de (Beeri & Kornatzky, 1990) est une extension de la logique modale (Emerson & Halpern, 1985) qui permet d'écrire des formules logiques sur l'existence de chemins dans un graphe (cf. section 3.1.2.3.2). Outre les connecteurs logiques et des opérateurs ensemblistes, différents quantificateurs peuvent être utilisés : ∀, ∃, 1 (i.e. au moins 1 chemin), 2, 3, ..., !1 (exactement 1 chemin), !2, !3, ..., most, many, few, etc. Avec le langage de requêtes de CGKAT, il n'est pas possible d'écrire de telles formules, mais des fonctionnalités assez proches peuvent être obtenues en combinant des commandes. Par exemple, il est possible de calculer le nombre de graphes résultats d'une requête puis, en fonction de ce nombre, lancer une nouvelle requête :

```
> sh ck spec [Concept] | wc -l | ck set var    // → var contient le nombre de GCs de la base
Ok
> sh if test $var = 2; then echo "Execution d'une nouvelle commande"; fi    //Exécution conditionnelle
Execution d'une nouvelle commande
```

Ainsi la combinaison de commandes de l'exemple ci-dessus correspondrait à l'usage du quantificateur "!2" dans le langage de Beeri & Kornatzky. Pour avoir l'équivalent des quantificateurs "most" (qui, si l'utilisateur ne le reprogramme pas, signifie par défaut "plus de la moitié des chemins") et "∀" (tous les chemins), il faut tester la différence entre les nombres des résultats de deux requêtes. Exemple :

```
> sh ck spec [Concept] | wc -l | ck set v1;  ck spec [Cat] | wc -l | ck set v2
Ok
> sh if test $v2 = $v1 ; then echo "All CGs of the base are about cats"; fi
All CGs of the base are about cats
```

Le langage de Beeri & Kornatzky comporte trois opérateurs de recherche (what, max, min) et un opérateur de construction (make). Un opérateur est déclenché si la formule logique auquel il est associé est vérifiée. L'opérateur "what" retourne les chemins décrits par la formule logique. Les arcs entre les noeuds du réseau hypertexte peuvent avoir des attributs valués ; l'opérateur "max" (resp. "min") prend en paramètre un type d'attribut et retourne, parmi les chemins décrits par la formule logique auquel il est associé, celui qui maximise (resp. minimise) la somme des valeurs des attributs du type spécifié. L'opérateur "make" permet de créer un lien entre deux noeuds. Actuellement, dans CGKAT, aucune commande ne correspond à ces opérateurs max, min et make. Il est possible de détruire et de créer un nouveau graphe mais aucune commande du langage de requête ne permet de modifier des sous-parties d'un graphe existant. Par ailleurs, si un graphe a été construit via un ED GC dans un document, sa destruction de la base de CoGITo via le langage de requête n'affecte pas le document. Nous avons souligné qu'il serait utile de pouvoir utiliser une commande prenant en paramètre un GC au format linéaire et générant ou modifiant à partir de celui-ci, un ED CGgraph à un emplacement spécifié dans un document.

Cette option de modification par destruction puis re-génération d'un graphe entier, est envisageable lorsque les graphes à modifier ne sont pas trop gros. Sinon, une commande de mise à jour doit pouvoir directement modifier des sous-parties d'un graphe et pour cela elle doit être connectée de manière fine à des opérateurs de recherche de sous-parties (dans le langage de requête de CGKAT, la connexion entre commande de recherche et commande de manipulation de graphes se fait par passage, en paramètre ou via l'usage de "pipes", de noms de graphes).

6.3.2.2.6.2 Recherche de sous-graphes dans plusieurs graphes

Nous avons proposé dans la section précédente que la syntaxe des graphes requêtes dans CGKAT soit étendue pour permettre la recherche de sous-graphes dans un seul graphe (comme c'est le cas dans tous les systèmes hypertextes que nous connaissons). Mais une base de GCs permet de manipuler différents graphes et il est intéressant d'exploiter cette possibilité. Au moins deux options sont possibles pour la recherche de sous-graphes dans une liste donnée de graphes :
1) recherches séparées dans chaque graphe,
2) recherche dans le graphe correspondant à la jointure des graphes de la liste donnée, sur les concepts qui sont coréférents[1], i.e. qui représentent le même individu (lors de la jointure, nous supposons que tous les concepts coréférents sont joints en un seul concept et qu'il n'y a alors qu'une seule jointure ; toutes les relations qui dans les divers graphes étaient connectées aux divers concepts coréférents sont dans le graphe de la jointure connectées au même concept).
De plus, dans ce dernier cas, si une réponse est un sous-graphe dont les noeuds concepts appartiennent à différents graphes de la liste donnée, ce sous-graphe pourrait être présenté en entier ou bien ses sous-parties provenant des différents graphes de la liste pourraient être présentées séparément.

Les requêtes relatives à ces recherches pourraient être posées de manière similaire à celle vue dans la section précédente. Par exemple, en appelant "subSpecInEach" la commande de recherches séparées dans chaque graphe d'une liste donnée en paramètre :

```
> subSpecInEach [Cat]- {→(Spatial_relation)→;}* [Mat]   in (#GraphName1, #GraphName2)
    //la liste des graphes donnée pourrait également être composée avec les résultats d'autres requêtes
```

Une autre commande à la syntaxe bien différente (voir ci-dessous) nous semble complémentaire, au moins pour le deuxième type de recherches (recherches dans le graphe correspondant à la jointure des graphes de la liste donnée, sur les concepts qui sont coréférents).

```
paths [from  (  (Concept_list  |  any)   [in Graph_list]  )+ ]
      [to      (  (Concept_list  |  any)   [in Graph_list]  )+ ]
      [via     (  (Relation_list  |  any)   [in Graph_list]  )+ ]   [depth Max_depth]
      [with    (  (Graph_list     |  any)    in Graph_list  )+ ]
      [noJoin]

Concept_list  ← Concept  [Concept_list]
Relation_list ← ( ( ["→" | "←"] RelationType ) | Sequence )  [Relation_list]
Graph_list    ← ( GraphName | request_for_graphs )  [Graph_list]
```

La commande "paths" permettrait de rechercher des chemins dans un ou plusieurs graphes. Dans le cas de plusieurs graphes, la recherche s'effectuerait sur le graphe correspondant à leur jointure sur

1. Des concepts individuels appartenant à un ou divers graphes de la base sont coréférents s'ils possèdent des marqueurs individuels identiques. A l'intérieur d'un graphe, des concepts génériques sont coréférents s'ils portent une même variable (i.e. un marqueur générique nommé). Spécifier que des concepts génériques appartenant à différents graphes sont coréférents est moins simple. Le système des variables peut être utilisé si les graphes sont regroupés dans des blocs qui définissent ainsi la portée des variables ; c'est le cas prévu dans le format de sauvegarde des graphes utilisé par CoGITo. Si des graphes sont construits sous une forme graphique, des "lignes d'identité" peuvent aussi être utilisées pour joindre et ainsi signaler des concepts génériques coréférents (Sowa, 1984). Dans CGKAT, si des EDs GCs sont utilisés pour construire des graphes, les concepts génériques coréférents sont ceux qui incluent un même concept unique. Mais cette information n'est actuellement pas répercutée dans la base de CoGITo car celle-ci n'offre pas de moyen approprié (e.g. un attribut spécial dans une classe d'objet) pour stocker des liens de coréférence (et nous n'avons pas modifié CoGITo dans ce sens). CGKAT ne permet pas de signaler que des concepts génériques appartenant à différents graphes sont coréférents si ces graphes sont construits dans un format linéaire.
Seul l'utilisateur peut signaler les concepts existentiels qui sont coréférents : si cette information n'est pas stockée, elle ne peut être générée.

les concepts qui sont coréférents. La commande "paths" permettrait de préciser les contraintes suivantes :
1) un ensemble de concepts initiaux devant éventuellement appartenir à des graphes précis[1] (l'extrémité initiale d'un chemin résultat peut être l'un quelconque de ces concepts),
2) un ou plusieurs concepts finaux[2] devant éventuellement appartenir à des graphes précis,
3) une ou plusieurs sortes de relations[3] ou de séquences[4] à traverser et devant éventuellement appartenir à des graphes précis,
4) un nombre maximal de relations dans un chemin,
5) des graphes à joindre au chemin lorsque certains graphes sont traversés,
6) une indication sur la manière de présenter chaque chemin retrouvé : en entier ou bien avec les sous-parties provenant des différents graphes traversés par le chemin (option "noJoin").

Ainsi, la commande paths permettrait de rechercher des sous-graphes parmi des graphes, en contrôlant précisément l'appartenance des composants des chemins retrouvés à ces différents graphes. Cela n'était pas possible avec les autres formes de requête (cf. exemple avec la commande subSpecInEach). Voici un exemple simple pour la commande paths (ici les même résultats auraient pu être obtenus avec les autres formes de requêtes) :

paths from [Accident] in spec [Accident]→(First_phase)→[Accident_phase]
to [Accident_phase] in spec [Accident_phase]→(Location)→[Spatial_entity]
via →Next_phase

6.3.2.2.7 Génération d'explications

Les différentes techniques de requête présentées ci-dessus fournissent de bons outils pour rechercher des informations/connaissances à intégrer dans une explication. Un générateur d'explications doit réaliser les tâches suivantes, chacune d'entre elles pouvant être complexe :
1) déterminer le besoin explicatif de l'utilisateur à partir d'une requête exprimée dans un langage et/ou compte-tenu du contexte (tâche en cours, modèle utilisateur, historique des échanges, etc.) ;
2) générer des requêtes telles que celles présentées ci-dessus, afin de rechercher des informations permettant de satisfaire le besoin explicatif déterminé ;
3) sélectionner et structurer les informations recueillies, et éventuellement les compléter en recherchant de nouvelles informations ;
4) présenter les informations à l'utilisateur grâce à la génération de phrases, de documents structurés, etc.

Il est donc essentiel qu'un générateur d'explications puisse exploiter, via des outils de recherche puissants, une base de connaissances bien organisée. De nombreux travaux relatifs aux explications portent sur la "génération de vues" sur une base de connaissances. Nombre de ces travaux concernent l'interprétation d'une demande d'explications afin de déterminer des points de vue pertinents permettant de rechercher et de présenter à l'utilisateur des informations répondant à ses besoins (McKeown, 1985a) (Acker & al., 1991) (Suthers, 1993). Quelques autres travaux comme celui de (Acker & Porter, 1994) concernent l'organisation de la base et des techniques à employer pour créer des vues à partir des points de vue retenus. On note surtout dans ces derniers travaux, la nécessité de permettre une recherche par le contenu (i.e. une recherche exploitant une relation de spécialisation entre des connaissances) et quelques techniques de présentation des connaissances retrouvées.

---

1. Les graphes peuvent être référés par leurs noms ou être le résultat de requêtes.

2. Si aucun concept final (resp. initial) n'est spécifié, n'importe quel concept de n'importe quel graphe peut être l'extrémité finale (resp. initial) d'un chemin résultat. Au moins un concept final ou initial doit être spécifié.

3. La description des relations peut se faire avec un type de relation et éventuellement un sens dans lequel un chemin doit les traverser (options "→" et "←").

4. La description des séquences peut se faire de manière similaire à celle illustrée dans la section précédente (e.g. comme ceci : "→(R1)→[B1]→(R2)→") ou bien avec d'autres conventions.

Une demande d'explications peut porter sur des connaissances de la base ou sur le résultat d'un raisonnement. Dans le deuxième cas, la trace du raisonnement doit permettre de retrouver les connaissances exploitées dans le raisonnement et la stratégie qui a été suivie. Deux types d'approches complémentaires sont classiquement utilisées pour déterminer le contenu et parfois également la structure d'une explication :

1. les *mécanismes de "traversées" de graphes* , i.e. la recherche de sous-graphes et de chemins dans des graphes tels que des réseaux causaux, des traces de raisonnements représentées sous forme de graphes, ou encore les graphes que constituent des hiérarchies de types ;
2. l'exploitation de *schémas explicatifs* via des *techniques de planification de discours* telles que le *remplissage de schémas rhétoriques (*McKeown & al., 1985), le *raisonnement sur les croyances des interlocuteurs* (Appelt, 1985), la *composition de relations rhétoriques* (Hovy, 1988), la *planification ascendante* (opportunistes) (Lester & Porter, 1991) (Lemaire, 1992) et la *planification descendante* (Moore & Paris, 1991), (Maybury, 1991) (Cawsey, 1991) (cette dernière approche peut également être utilisée pour gérer un dialogue).

Des architectures ont également été créées afin de compléter ces diverses approches les unes par les autres (e.g. (Suthers 1993)).

Afin de faciliter l'interprétation des requêtes de l'utilisateur, il existe des classifications de types de questions telles que "Qu'est-ce que X ?", "Pourquoi X ?", "Pourquoi pas X ?", "Comment X ?", "Comparer X et Y ?" (Moore, 1989) (Karsenty, 1992) mais les frontières entre les catégories correspondant à des questions réelles sont peu claires (Gilbert & al., 1990). En effet, la forme syntaxique de ces types de questions les rendant ambiguës[1], des questions très différentes peuvent être posées de manière similaire, et inversement une même question peut être posée de différentes façons. Aussi, les classifications de type de questions prennent maintenant plutôt la forme de types de réponses que le système peut fournir, c'est-à-dire des types de connaissances qui peuvent être fournies pour répondre à une question (Gilbert, 1987) (Sarantinos & Johnson, 1991). Ces classifications ressemblent en partie à notre ontologie de types de relations, quoiqu'elles soient beaucoup moins développées et organisées (elles contiennent une vingtaine ou une trentaine de types). Voici quelques exemples d'éléments de ces classifications : existence, instanciation, exemplification, définition, membres, similarité, fonction, raison, condition, antécédance, fréquence, implication, fréquence, lieu.

Nous avions pensé à représenter sous forme de types de relations, des types de questions générales telles que "Qu'est-ce que X ?", "Pourquoi X ?", "Comment X ?" ou "Comparer X ?", de manière à pouvoir les introduire dans notre ontologie de types de relations et ainsi les spécialiser par des types de relations plus précis (cf. note 1 pour quelques correspondances). Cependant, cela aurait surchargé un peu inutilement notre ontologie. En effet, cette ontologie et le menu de gestion des types de relations permettent assez facilement de retrouver un type de relation précis ; l'utilisateur

1. De nombreuses sortes de relations pourraient être recherchées dans le ou les graphes d'une base pour fournir le matériel nécessaire pour répondre à de telles questions. Voici quelques exemples des types de ces relations (nous utilisons ici des types de notre ontologie de types de relations).
a) "Qu'est-ce que X ?" : certains sous-types de Attributive_relation, les sous-types de Component_relation, les sous-types de Constraint_or_measure_relation.
b) "Pourquoi X ?": Result, Purpose, Recipient, Agent, les sous-types de Temporal_relation_between_situations (e.g. Condition, Causation, Task_Pred).
c) "Comment X ?" : Agent, Initiator, Instrument, SubProcess, Method, Result, et de nombreux sous-types de Rhetorical_relation (e.g. RST_Justify, RST_Background, RST_Purpose, RST_means, etc.).

Pour les questions "Qu'est-ce que ?", les relations de sous-typage et d'instanciation dans l'ontologie d'une base pourraient également être exploités.
Un schéma explicatif simple pour une question de type "Qu'est-ce que X ?" peut être : "si X est un type, donner ses supertypes immédiats, sinon donner son type et lister les concepts qui sont liés à lui par des relations sous-type de Attributive_relation ou de Component_relation. En termes de points de vue, ce schéma fournit une explication sur les points de vue "Classificatoire", puis "Attributif, puis "Structurel" (McKeown, 1985a) (Acker & al., 1991) (Suthers, 1993).

peut ainsi construire assez rapidement des requêtes en utilisant des types précis, et donc avoir des réponses précises. De plus, pour répondre à des questions générales, des *schémas explicatifs* sont plus adéquats que des relations de spécialisation pour stocker des correspondances entre chaque type de question générale et les sortes de relations à rechercher dans le ou les graphes d'une base afin de répondre à une telle question. Les schémas explicatifs étant des sortes de règles, ils peuvent spécifier en fonction du contexte 1) quelles informations rechercher étant donné un but explicatif donné, et 2) comment combiner ces informations. Rappelons que le contrôle de l'enchaînement des schémas explicatifs est déterminé par la technique de planification utilisée et que ce contrôle peut être descendant ou opportuniste. Notons également que pour répondre à certaines questions, e.g. des questions du type "Pourquoi pas X ?", des recherches dans la base (traces de raisonnement comprises) peuvent ne pas suffir et dans ce cas des raisonnements (calculs, inférences, etc.) doivent être effectués.

Dans le cadre de la recherche dans une base de connaissances avec des requêtes précises, signalons ROCK (Carbonneill & Haemmerlé, 1994), un système de Question/Réponse fondé sur le formalisme des Graphes Conceptuels. Dans ce système, les spécialisations d'un GCS peuvent être recherchées dans une base de GCSs mais lorsqu'aucune réponse n'est trouvée, des contraintes sur la requête sont progressivement relâchées, c'est-à-dire que le graphe requête est progressivement généralisé (généralisation des types et/ou sélection d'une sous-partie du graphe requête) de manière à obtenir des réponses approchées à défaut de réponses exactes. Ce système utilise CoGITo et pourrait donc à terme, et si nécessaire, être réutilisé dans CGKAT.

De plus, comme nous l'avons souligné dans la section 6.3.2.2.2.2, l'utilisation des documents structurés hypertextes pour présenter des explications, *évite en partie* certains problèmes de la génération d'explications (choix d'informations parmi celles pouvant être présentées, choix de l'ordre dans lequel les informations vont être présentées).

Un script peut contenir le moyen de répondre à une requête habituelle, e.g. une question générale du type "Qu'est ce que X ?" (le corps de la question étant alors passé en paramètre au script)[1]. Nous avons indiqué les perspectives qu'offrirait pour la génération de documents (explicatifs ou non) l'intégration à notre langage de commandes, de commandes d'édition de documents structurés. Un script qui est appelé simplement en effectuant un choix dans un menu ou en cliquant sur un ED (e.g. un mot, une phrase) peut être considéré comme l'implémentation d'un *lien hypertexte virtuel* (cf. section 3.1.1.2). Nous décrivons dans la prochaine section les fonctionnalités qu'offre CGKAT pour la création de liens virtuels.

1. Voici à titre d'exemple un script simple, nommé "whatIs", implémentant un moyen de répondre à une question du type "Qu'est-ce que X ?". Dans ce script shell, $1 représente le paramètre X qui doit être le nom d'un type ou d'un individu. Ce script ne prend pas en compte le contexte (tâche en cours, modèle utilisateur, etc.).

```
ck \? "$1 < Concept" | ck set v
if test v = "Yes"
then echo "Supertypes of $1";  ck gene $1 | ck display
     echo "Definition of $1";   ck def of $1 | ck display    //all kinds of def: NSC, SC, NC, TC
     echo "Subtypes and instances of $1";  ck spec $1 | ck display
else
     echo "Type of $1";   ck type of $1 | ck display
     echo "Attributs of $1"; ck spec "[Concept:$1]→(Attributive_relation)→[Concept]" | ck display
     echo "Components of $1";  ck spec "[Concept:$1]→(Component_relation)→[Concept]" | ck display
     echo "$1 with processes";  ck spec "[Concept:$1]←(Relation_from_a_process)←[Process]" | ck display
                                  ck spec "[Concept:$1]→(Relation_to_a_process)→[Process]" | ck display
fi
//→ If  "[Cat:Tom]→(Attr)→[Lazzy]" is in the base, "whatIs Tom" will display this graph.
```

6.3.2.2.8 Liens virtuels

Il existe un moyen simple dans CGKAT pour appeler une requête en cliquant sur un ED quelconque : il suffit que cet ED soit connecté à un ED Request par un lien hypertexte de type quelconque (e.g. un attribut référence de type To_Anything). En effet,
1) lorsqu'un lien hypertexte est activé (en double-cliquant sur son ED source et, si celui-ci porte plusieurs liens hypertextes, en choisissant le lien désiré), la destination du lien est *sélectionnée*, et
2) lorsqu'un ED Request est *sélectionné*, la requête qu'il contient est exécutée (à condition que l'ED Answers destiné à stocker les résultats soit vide).

Avec cette méthode, l'ED Request destination peut appartenir à n'importe quel document déjà créé : il peut par exemple être situé au milieu d'autres informations. L'exécution de la requête sélectionne des informations selon différents critères (points de vue) et dans tous les documents ouverts au moment de la requête. La vue ainsi générée est exhaustive pour les points de vue de la requête et pour les documents ouverts à cet instant. Une fois qu'il l'a visualisée, l'utilisateur peut choisir de conserver ou non cette vue. S'il ne la conserve pas (i.e. s'il détruit l'ED généré ou s'il ferme le document ainsi modifié sans le sauver), la prochaine sélection de l'ED Request va générer une nouvelle vue. Sinon, la vue sera conservée et pourra donc ne pas être exhaustive pour les points de vue de la requête et pour les documents ouverts au moment de cette nouvelle sélection. Cependant, l'auteur de cette sélection peut détruire la vue et demander sa re-génération.

Au lieu de relier un ED à un seul ED Request, il peut être préférable de le relier à une liste d'EDs RequestAnswer. Ainsi, lorsque cet ED source est activé, aucune requête particulière n'est automatiquement exécutée, mais une liste de requêtes est visualisée et l'utilisateur peut sélectionner celle qu'il veut exécuter.

La méthode que venons de proposer pour créer des liens virtuels a toutefois deux limitations :
1) un lien hypertexte doit être posé sur un ED afin que son activation déclenche une requête, et
2) la requête ne peut prendre en compte l'ED sélectionné (e.g. son contenu ou son type).

Pour résoudre ces limitations, nous avons implémenté la fonctionnalité suivante : CGKAT peut générer un document avec un script de l'utilisateur, lorsqu'un ED ne portant pas de lien hypertexte est activé[1]. Décrivons le principe de cette implémentation. Lorsqu'un ED ne portant pas d'attributs référence est activé, s'il existe dans le répertoire courant un script shell dont le nom est la concaténation de "script_for_", du nom du type de l'ED activé et du nom de l'utilisateur, CGKAT crée et affiche un document composé d'un seul ED CGRequests et dont le premier ED Request contient un appel au script shell avec comme paramètres, le nom du document où a eu lieu l'activation, l'identificateur de l'ED activé, son type et enfin, s'il s'agit d'un ED textuel, son contenu. Cet appel est alors déclenché comme si l'ED Request avait été sélectionné par l'utilisateur. Si le script shell recherché n'existe pas, CGKAT recherche un script dont le nom est la concaténation de "script_for_AnyElem." et du nom de l'utilisateur, et s'il existe, CGKAT l'exécute de la même façon que précédemment. Sinon, l'activation de l'ED n'a aucun effet.

Ainsi par exemple, si le mot "test" est activé dans un document de nom "Example" par l'utilisateur "phmartin", le premier script shell recherché est "script_for_Text.phmartin", et si ce script existe, la requête exécutée est par exemple "script_for_Text.phmartin Example L34 Text test". Si au lieu d'un mot, un paragraphe simple est activé mais que le script "script_for_Simple_paragraph.phmartin" n'existe pas, le script "script_for_AnyElem.phmartin", s'il existe, est appelé avec les paramètres "Example L23 Simple_paragraph". Les scripts appelés peuvent exploiter ou non ces paramètres. Lorsque des commandes d'édition de documents structurés seront intégrées à notre langage de commandes, elles pourront au besoin être utilisées pour récupérer plus d'informations sur l'ED activé, dans la structure logique du document contenant cet ED. Il pourra

1. Normalement, dans Thot, un double clic dans une partie de texte à l'intérieur des EDs textuels n'a aucun effet. Nous avons fait en sorte qu'une double sélection à l'intérieur d'un mot dans une portion de texte entraîne la *sélection et l'activation* de ce mot.

alors être utile d'appeler des scripts selon un mécanisme identique à celui que nous venons de décrire, mais pour tous les types d'actions de l'utilisateur sur un ED (e.g. destruction, création, modification, chargement, sauvegarde) et non seulement lors de son activation ; cette extension est aisée à réaliser. Ainsi, par exemple, des scripts pourront être écrits pour épier l'activité de l'utilisateur, gérer un historique de ses actions, et donc prendre en compte le contexte pour générer de manière spontanée des documents guidant l'utilisateur.

6.3.3 Comparaison avec d'autres systèmes de recherche d'informations ou d'acquisition de connaissances

Une caractéristique originale de CGKAT est d'être à la fois un outil d'acquisition des connaissances et un système de recherche d'informations "basé sur les connaissances".

6.3.3.1 Comparaison avec les systèmes hypertextes

Certains systèmes hypertextes, tel MacWeb, permettent de stocker et dans une certaine mesure de gérer des connaissances : MacWeb repose sur un modèle orienté objet et peut ainsi effectuer certains contrôles et prendre en compte la relation de spécialisation entre types de noeuds lors des recherches de chemins. Ainsi, comme le montrent (Hofmann, 1990) et (Nanard & al., 1993a, 1993b), ces systèmes peuvent être utiles pour modéliser des connaissances, notamment pour gérer différentes sources de connaissances et permettre la coopération entre plusieurs cogniticiens, grâce aux facilités de documentation, d'organisation des connaissances, d'accès aux informations, de génération de vues, etc. Toutefois, pour l'acquisition des connaissances, des langages de représentation de connaissances plus élaborés que ceux de ces systèmes peuvent être utiles, ainsi que des ontologies : modèles de tâches, ontologies du domaine, etc.

Malgré les aspects intéressants des systèmes hypertextes pour l'acquisition des connaissances, et la nécessité de manipuler des documents sources d'expertise, les principaux outils d'acquisition des connaissances n'offrent le plus souvent que les fonctions élémentaires des systèmes hypertextes, c'est-à-dire essentiellement la possibilité de poser des liens hypertextes directs entre des objets de certains types prédéfinis : c'est par exemple le cas dans la K-Station (Albert & Vogel, 1989) (ILOG, 1993a) et dans Kads-Tool (Albert & Jacques, 1993) (ILOG 1993b))

Par ailleurs, les systèmes de gestion et de recherche d'informations s'orientent de plus en plus vers une *indexation* fine des informations, i.e. vers une représentation sémantique et formelle ou semi-formelle de leur contenu (cf. section 3.1). CGKAT pousse cette tendance encore plus loin en permettant d'utiliser pour l'indexation des informations, un langage formel, *dédié à la représentation et l'organisation des connaissances*, et néanmoins d'une utilisation assez intuitive. Par ailleurs, CGKAT intègre un éditeur de documents structurés de telle sorte que 1) n'importe quel ED, structuré ou non, puisse être indexé, et 2) les connaissances, qu'elles indexent ou non des EDs, puissent être représentées, manipulées, présentées, associées à d'autres EDs et recherchées comme tous les autres EDs. Nous avons ainsi permis l'utilisation conjointe ***et*** combinée, sur des informations comme sur des connaissances, de techniques d'organisation/recherche/gestion d'informations de différents types : des techniques basées sur la structuration des documents (typage de EDs et de leurs attributs, séparation structure logique/structure physique), des techniques hypertextes, et des techniques basées sur une représentation formelle et organisée de connaissances sous-jacentes à des informations.

Thot a des fonctionnalités très intéressantes pour un système de gestion de documents et un système hypertexte : utilisation de modèles structurels et de modèles de présentation, gestions de vues, zoom, outils de recherches, etc. Normark & Osterbye (1996) ont récemment proposé un modèle hypertexte dont les fonctionnalités sont assez proches de celles de Thot. Mais Thot ne permet pas de représenter et de manipuler des connaissances. Les seules "représentations" utilisables sont les types des EDs et de leurs attributs, ces types doivent être définis dans un modèles structurel, et aucune relation de spécialisation/généralisation ne permet de les comparer. CGKAT complète Thot

en permettant de manipuler un langage général dédié à la représentation des connaissances. Ainsi, par rapport aux autres systèmes hypertextes "basés sur les connaissances", CGKAT hérite des avantages de Thot et des GCs.

Nous avons effectué en section 6.2.6 une comparaison de notre approche par rapport à d'autres travaux, au sujet de l'utilisation d'un éditeur de document structuré pour construire une base de connaissances. Nous nous focalisons maintenant sur les aspects indexation/recherche de notre approche.

Dans un système hypertexte typé, les connaissances sont représentées dans le réseau hypertexte. Dans CGKAT, la base de connaissances est séparée du système hypertexte. Il est peu important que les descripteurs utilisés dans une indexation appartiennent à une base de connaissances séparée ou non du système hypertexte. Les points importants dans un système de recherche d'informations sont :

1. *les sortes d'EDs pouvant être indexés* : seulement des mots ou des objets prédéfinis, ou n'importe quel ED structuré ou non ;
2. *la manière dont est connectée un ED à son ou ses descripteurs* : comme l'indique le modèle DEXTER (Halasz & Schartz, 1990), il est intéressant d'utiliser un système d'ancrage pour permettre l'indépendance entre EDs et le système de représentation. Cela permet à un ED d'être indexé de différentes manières. Il est également intéressant que les ancres soient typées (Nanard & Nanard, 1993), de même que les liens entre les ancres et leurs descripteurs : c'est le cas dans CGKAT où les EDs jouent le rôle d'ancres. La prise en compte comme dans CGKAT de différents types d'indexation (à savoir la représentation et l'annotation) et pour chacun, de plusieurs points de vue et de plusieurs utilisateurs, semble originale. Le fait que les connaissances soient stockées dans des EDs semble également original. Nous avons noté en section 6.3.2.1 qu'un des avantages de cette approche était la possibilité de mélanger des connaissances avec d'autres EDs et donc d'associer des informations complexes à des descripteurs et ce, de multiples façons (attributs, liens de composition, liens hypertextes, inclusions). Notons également que même dans les systèmes avancés, il n'est pas toujours possible de visualiser les descripteurs et leurs interconnections. Dans CGKAT, ces descripteurs et leurs interconnections peuvent être visualisées et manipulées grâce aux nombreuses fonctions de Thot (modèles de présentation, gestions de vues, zoom, outils de recherches, etc.) ;
3. *la précision du ou des descripteurs, et les inférences pouvant être effectuées* à partir de ceux-ci : un concept ou un GC est ainsi une meilleure description qu'un terme informel, même si ce dernier appartient à un thésaurus ;
4. *la manière dont chaque descripteur peut être connecté (et donc défini par rapport) à d'autres descripteurs ou connaissances* : pour cela, un réseau *sémantique*, permettant de contextualiser des connaissances et permettant de nombreuses inférences, est intéressant tant pour la navigation que pour les requêtes et les raisonnements. Dans les systèmes hypertextes actuels, les moyens d'indexation prennent peu en compte les notions de contexte et sont rarement formels, ce qui peut limiter la précision ou l'exploitation des descripteurs et de leur connexions et donc la recherche et l'exploitation des informations.
   Si les descripteurs sont formalisés avec des GCs, la relation de spécialisation/généralisation entre ceux-ci peut être exploitée via des requêtes, mais aussi par navigation si elle est implémentée dans une structure. La navigation dans cette structure permet à un utilisateur d'élargir ou de spécialiser sa recherche graduellement suivant les descripteurs présents dans la structure. Godin & al. (1995), ainsi que Cole & Eklund (1996), ont implémenté ce système de navigation pour faciliter les recherches dans des bases de documents. Dans CGKAT, ce système de navigation n'est pas implémenté car aucune structure n'est générée pour implémenter la relation de spécialisation/généralisation entre GCs ;
5. *la manière dont le système permet d'exploiter ces descripteurs et les liens qui les relient* : navigation, langage de requête et de génération de documents, génération de vues ou documents virtuels (CGKAT permet également d'insérer des vues à l'intérieur de documents déjà existants),

génération d'index (e.g. dans CGKAT, la génération de tables des matières ou de synthèses de représentations d'EDs symboles), liens virtuels ;

6. *la manière dont ces descripteurs peuvent être créés et organisés : manuellement ou automatiquement*. La recherche d'informations dans une grande masse d'informations exige qu'une bonne part de l'indexation puisse être faite automatiquement. Mais les méthodes actuelles d'analyse du langage naturel, et à fortiori les méthodes statistiques, ne fournissent pas des descripteurs précis et ne les organisent pas par des liens ayant un fort contenu sémantique. Les recherches qui peuvent alors être faites sont des "recherches approchées" et non des "recherches exactes" (cf. section 3.1.1.3).

Grâce aux fonctionnalités héritées de Thot et de CoGITo (i.e. grâce à l'utilisation combinée d'un éditeur de documents structurés et d'un gestionnaire d'une base de GCS), et grâce à notre indexation d'EDs par des éléments du formalisme des GCs, CGKAT offre une large palette de possibilités relatives aux cinq premiers points ci-dessus. Sur ces cinq points, CGKAT est donc bien placé par rapport aux autres systèmes de recherche d'informations orientés vers la précision des réponses.

En revanche, et contrairement à un certain nombre de systèmes de recherche d'informations (notamment les systèmes de "recherches approchées" comme les systèmes documentaires), nous n'avons intégré aucune méthode d'indexation automatique. Cependant, grâce à son langage de commandes, CGKAT est "ouvert" à d'autres applications, et différentes méthodes d'indexation pourraient être intégrées via le langage de commandes lorsque celui-ci intégrera des commandes d'édition de documents structurés.

CGKAT est un système hypertexte "basé sur les connaissances" où l'orientation "connaissances" est particulièrement accentuée. MacWeb (Nanard & Nanard, 1991) semble être le système hypertexte "basé sur les connaissances" avec lequel CGKAT partage le plus de fonctionnalités (cf. section 3.2..2.2.1)[1], notamment la génération de documents virtuels.

Le niveau ancrage de MacWeb est néanmoins particulier. Il permet d'isoler un ED avec une ancre et de l'associer à *un noeud concept*, ainsi qu'à un "contexte" (le plus petit ED qui englobe l'ED indexé et qui a une signification autonome pour un lecteur) et à un "type de contexte" (un marqueur pour le contexte). On peut noter que seuls les EDs symboles sont différents de leur "contexte". Avec un tel système d'ancrage, un ED symbole peut être connecté à l'ED description minimal qui le contient, et un type est donné à cet ED (e.g. "définition", "exemple"). Dans CGKAT, un ED symbole peut être représenté par *au moins un concept unique*, et un ED description, quel qu'il soit, peut être représenté par *au moins un graphe* (CGgraph, définition de type ou concept unique emboîtant un graphe), lequel peut inclure des représentations de sous-parties de l'ED description (il y a ainsi des connexions entre la représentation d'un ED symbole et celles de divers EDs description). L'utilisation de "contextes", au sens de MacWeb, n'est donc pas nécessaire dans CGKAT. Rappelons également, à titre de comparaison, que CGKAT permet à divers utilisateurs de représenter *ou* d'annoter un même ED, selon un ou plusieurs points de vue.

Nous avons également vérifié que la méthode de représentation d'EDs dans CGKAT permettait de couvrir les différents cas évoqués par Biezunski (1995)[2] sur l'indexation d'EDs par des noeuds concepts. Par ailleurs, nous avons analysé la "représentation d'EDs" de façon à en dégager un certain nombre de contraintes sémantiques (section 6.3.1.3) et à la gérer de manière appropriée (par

1. Notons que MacWeb n'utilise pas de modèle structurel ni de modèle de présentation, mais son langage de requête (cf. section 3.1.2.3.6) permet de définir le contenu et la présentation de documents. Pour la représentation des connaissances, un modèle objet est utilisé, mais les relations entre les classes d'objets peuvent être décrites de la même façon que les relations entre objets, c'est-à-dire par le réseau hypertexte. Ce langage se rapproche donc plus de KL-ONE (Brachman & Schmolze, 1985) que du formalisme des GCs.

2. Pour indexer des EDs, la "Convention pour l'application de HyTime" (Biezunski, 1995) note l'insuffisance des index classiques (e.g. bibliographies et autres références croisées entre documents, tables des matières, glossaires, mots-clés) pour la recherche d'informations, et préconise l'usage de "cartes de topiques" (sortes de réseaux sémantiques dont les types des concepts peuvent avoir des définitions) pour indexer des éléments des documents. Différents cas d'indexation d'EDs par des noeuds du réseau sont listés.

exemple, les représentations des EDs symboles sont normalement des concepts uniques de type autre que Proposition, ces concepts ne sont pas répercutés dans CoGITo mais "inclus" dans divers graphes pour permettre la navigation entre ceux-ci, et il est possible de générer des index synthétisant ces représentations et les méta-informations qui les accompagnent).

CGKAT a également un certain nombre de points communs avec RIME (Kheirbek & Chiaramella, 1995), un système hypertexte (mais orienté vers la recherche documentaire) exploitant le formalisme des GCs. Dans RIME, chaque ED est représenté sous la forme d'un concept dans différents GC, et son contenu est lui-même décrit par un GC. Les relations entre cet ED et d'autres EDs (relations sémantiques mais également les relations de composition entre EDs) sont représentées sous la forme de relations conceptuelles. La navigation est permise dans les GCs sur ces relations, ainsi que dans la hiérarchie des types de concepts, entre les concepts et leurs types dans la hiérarchie. Les requêtes exploitent la relation de spécialisation entre GCs. Pour augmenter les chances de retrouver un ED satisfaisant, des représentations d'EDs sont générées par jointure maximale dirigée sur les représentations des sous-parties de ces EDs. RIME est orienté vers la recherche documentaire ; il ne semble pas qu'il possède d'autres fonctionnalités de CGKAT, par exemple celles héritées de Thot, l'indexation par de multiples utilisateurs, la gestion de GCs emboîtés et un langage de combinaison de commandes.

Comme d'autres systèmes hypertextes, MacWeb et RIME intègrent un module d'analyse du langage naturel permettant une certaine indexation automatique. Dans CGKAT, qui est plus orienté vers l'acquisition des connaissances, l'indexation est manuelle, mais une grande ontologie de types de concepts et de types de relations, ainsi que des menus de gestion adaptés, sont fournis afin de guider la représentation des connaissances. Certains systèmes hypertextes fournissent quelques types de concepts ou de relations, notamment les systèmes d'aide à la création coopérative de documents (e.g. gIBIS, SEPIA et PHIDIAS), mais ces systèmes ne permettent alors généralement pas de spécialiser ou de compléter ces types.

Notons que pour permettre la représentation et la recherche de connaissances dans CGKAT, il aurait été possible d'exploiter un outil similaire à CoGITo mais basé sur un langage de représentation terminologique du type KL-ONE (Brachman & Schmolze, 1985) (cf. section 4.4.2.1) au lieu du formalisme des GCs. En termes de fonctionnallités, la différence essentielle serait venue du fait que dans de tels langages de représentation terminologiques, les connaissances rentrées par l'utilisateur sont généralement automatiquement classées dans un unique réseau sémantique. Aussi, une recherche des spécialisations d'un graphe requête aurait permis de retrouver une portion de ce réseau sémantique et non les graphes tels qu'ils sont rentrés par l'utilisateur. Ces deux sortes de solution pouvant être intéressantes, nous avons proposé dans la section 6.3.2.2.6 une manière d'étendre la syntaxe des graphes requêtes dans le langage de commandes de CGKAT afin de permettre la recherche de sous-graphes (dont des séquences) dans un graphe.

6.3.3.2 Comparaison avec les outils d'acquisition des connaissances

Par rapport aux autres outils d'acquisition de connaissances, et en dehors des aspects recherche d'information, CGKAT se distingue également par sa généricité, c'est-à-dire :
1) par la possibilité d'utiliser un langage général de représentation de connaissances qui n'intègre "en dur" qu'un nombre minimum de primitives épistémologiques mais qui repose sur la définition d'ontologies où les significations et les contraintes d'utilisation de types de concepts ou de relations peuvent être définies ;
2) des fonctionnalités aidant ses utilisateurs à créer et à exploiter une ontologie pour l'extraction et la modélisation d'une base de connaissances (cf. chapitre précédent) ;
3) par la fourniture d'une ontologie synthétisant et interclassant ou complétant des types de concepts ou de relations provenant de divers travaux en modélisation/représentation des connaissances (cf. chapitre précédent).
Outre le chapitre précédent, la table 2.1 à la page 23 et la section 2.5 à la page 42 contiennent des comparaisons sur ces points, entre CGKAT et les autres outils d'acquisition de connaissances.

Cependant, comme un certain nombre de systèmes d'acquisition de connaissances, par exemple Kads-Tool, CGKAT est centré sur la modélisation de connaissances et n'intègre pas de système permettant d'exécuter des connaissances dynamiques, e.g. des règles ou des acteurs.
Des travaux s'effectuent actuellement dans l'équipe du LIRMM sur un mécanisme de règles de type "Si Alors" dont les prémisses et les conclusions sont des GCSs. Il est prévu que ce mécanisme (chaînage avant, chaînage arrière) soit implémenté au dessus de CoGITo ; il pourra alors éventuellement être intégré à CGKAT. Nous avons proposé dans la section 5.6.5.4 un format pour représenter des règles dans CGKAT. Nous n'avons pas proposé de format pour créer des acteurs, mais nous avons proposé un langage de script permettant d'écrire des procédures et donc de remplacer ou de simuler, dans une certaine mesure, le fonctionnement d'acteurs. Pour une représentation explicite (i.e. au "knowledge level") des connaissances, il vaut mieux lorsque cela est possible représenter des connaissances sous forme déclarative plutôt que procédurale, et donc écrire des règles sous forme de GCs plutôt que des scripts ou des acteurs.

6.4 Eléments méthodologiques d'utilisation de CGKAT

Nous n'avons pas voulu que CGKAT soit dédié à une méthodologie particulière, mais au contraire qu'il offre les outils nécessaires permettre de suivre différentes méthodes de travail dans la modélisation de connaissances ou la gestion d'informations : par exemple, les outils de génération de vues sur des critères conceptuels choisis par l'utilisateur, et les outils de gestion de grandes ontologies. Nous ne proposons donc pas ici de méthodologie particulière, mais nous récapitulons quelques règles d'utilisation de CGKAT. La plupart sont assez évidentes mais elles peuvent guider l'utilisateur débutant dans la succession des tâches à accomplir [1].

CGKAT est basé sur l'exploitation d'ontologies. Si dans une ontologie exploitée par CGKAT, les primitives épistémologiques d'une méthodologie particulière peuvent être définies, ainsi que les modèles, les contraintes d'utilisation et les règles d'inférence[2] associés à ces primitives épistémologiques, CGKAT devient un outil permettant de suivre cette méthodologie et d'effectuer les contrôles et les inférences nécessaires. L'ontologie proposée par défaut par CGKAT contient déjà des primitives épistémologiques de plusieurs travaux en modélisation/représentation des connaissances (cf. sections 5.4 et 5.6 et annexe 3), ainsi que des modèles permettant d'extraire et de modéliser des connaissances relatives à ces primitives (cf. sections 5.5 et 5.7.1). De plus, l'exploitation de WordNet permet à l'utilisateur de rechercher des spécialisations de certaines primitives, ou inversement de classer des types de concepts (des significations de mots) entre eux et par rapport à ces primitives.

Un utilisateur de CGKAT peut représenter des connaissances pour 1) créer et documenter une base de connaissances, et/ou 2) indexer des documents et ainsi les organiser sémantiquement. Dans les deux cas, si l'utilisateur veut suivre une ou plusieurs méthodologies, il doit disposer d'une ontologie contenant des types de concepts ou de relations conseillés par ces méthodologies. Pour cela, il peut compléter l'ontologie proposée par défaut par CGKAT et éventuellement supprimer parmi les types de cette ontologie ceux qu'il juge superflus, ou encore créer une ontologie en ne reprenant que les types conseillés par la ou les méthodologies suivies.

L'indexation de documents dans CGKAT, dans un but d'AC et/ou de RI, implique que les documents sources (e.g. des retranscriptions d'interview ou des documents techniques) soient des

1. Certains des utilisateurs de CGKAT apprécieraient des règles d'utilisation du type "Avant de créer une connaissance, par exemple pour indexer une information, il faut insérer dans l'ontologie les types de concepts ou de relations nécessaires pour décrire cette connaissance". Cependant, pour alléger la lecture, nous ne listerons pas de telles règles ici.

2. La définition d'un type par rapport à d'autres types permet certaines inférences mais un mécanisme de règles serait un complément fort utile.

documents Thot. Thot peut convertir des fichiers de différents formats (ASCII simple, LaTeX, CorTeX, etc.) en des documents Thot (voir le manuel utilisateur de Thot (Quint & al., 1995) pour de telles conversions ainsi que pour toute autre fonction de Thot concernant la création, la gestion ou encore la présentation de documents).

La RI dans des documents avec CGKAT, que ce soit dans un but de modélisation de connaissances ou simplement pour retrouver certaines informations, peut se faire
1) par recherche de chaînes de caractères,
2) par recherche sur les types et les attributs d'EDs, si les documents sont composés de divers EDs structurés,
3) par navigation sur des liens hypertextes (statiques ou virtuels), s'il y en a, et
4) par recherche sur la base de connaissances (requêtes libres ou via des scripts existants).

Aussi, pour permettre la RI, les documents doivent être structurés et indexés. Les types d'EDs devraient être utilisés seulement pour représenter des formats ou des media de représentation, e.g. Article, Section, Sous-section, Paragraphe, Image et Texte. Il est préférable d'utiliser des véritables représentations (e.g. avec un concept unique d'un type sous-type de Proposition), dès qu'il s'agit d'exprimer le contenu sémantique des EDs, e.g. le fait qu'un ED contient une définition, un exemple ou une hypothèse. En effet, une représentation avec un type d'ED est extrêmement pauvre (pas de relation entre types d'EDs, ni avec d'autres représentations) et un ED ainsi représenté ne peut donc être recherché et manipulé que si le nom exact du type de l'ED est connu. Par ailleurs, tous les types d'EDs doivent être définis dans un modèle structurel ; il faut donc prévoir dans ces modèles tous les types d'EDs pouvant être utilisés car la mise à jour de documents dont le modèle structurel est modifié n'est pas une tâche simple (des règles de conversion de documents doivent être écrites puis appliquées dans chaque document). Au contraire, une ontologie permet d'organiser de nombreux types et peut être modifiée à tout moment [1].

Si le contenu d'un document déjà structuré doit être modélisé et si les types de ses EDs (y compris donc le type du document entier) ne représentent pas seulement des formats ou des media de représentation, ces types peuvent guider la modélisation. Les "concepts" exprimés par ces types peuvent alors être représentés de manière plus explicite et manipulable, par exemple sous forme de concepts uniques.

Il semble intéressant lors de l'indexation de documents, dans un but d'AC et/ou de RI, de commencer par indexer les EDs symboles dans les documents sources. Une commande de CGKAT permet de représenter toutes les occurrences d'un mot dans un document par un même concept (plus précisément, les listes de représentations associées à ces occurrences incluront un même concept unique). Il est ensuite possible de construire des GCs en utilisant des inclusions de ces concepts. De plus, cette façon de procéder permet de travailler sur l'ontologie de l'application avant de l'utiliser pour construire des GCs. Les synthèses des représentations des EDs symboles semblent intéressantes, notamment si plusieurs cogniticiens travaillent sur une même expertise, pour comparer diverses représentations et ainsi adopter des conventions avant de construire des GCs. Plus les GCs partageront de concepts (par inclusion des mêmes concepts), 1) plus la navigation sera possible entre ces GCs, et 2) plus les GCs seront comparables (et plus de généralisations ou de jointures pourront être effectuées).

La construction de GCs, c'est-à-dire l'extraction (auprès d'experts ou dans des documents) et la modélisation de connaissances peut s'effectuer de manière ascendante ou descendante. Dans la pratique, ces deux approches doivent être combinées (cf. section 2.2.3 à la page 26). Il est difficile de donner des méthodes générales pour effectuer cette combinaison, et peu de méthodologies se prononcent à ce sujet, mais elle peut être assez intuitive dans la pratique, compte-tenu des données de l'expertise, de la tâche en cours, et de la masse de connaissances déjà représentées. Nous listons ci-après les facilités qu'offre CGKAT pour des tâches relatives à chacune de ces approches.

---

1. En cas de destruction de types, CGKAT vérifie qu'aucun GC de la base en cours n'utilise ces types ou leurs spécialisations. CGKAT vérifie également que l'ajout d'un type ne crée pas de cycle dans la hiérarchie, et qu'aucun des divers parents de ce type ne sont exclusifs.

6.4.1 Extraction et modélisation ascendante avec CGKAT

Voici des tâches typiques d'extraction ou de modélisation ascendantes pour l'AC ou la RI :

1. La représentation systématique de termes ou de phrases et éventuellement d'autres types d'EDs, dans des documents (par exemple avec un outil d'analyse terminologique ou un outil d'analyse du langage naturel, utilisant des techniques de classification conceptuelle ou exploitant une ontologie (e.g. Tanka (Feng & al., 1994)).
2. La recherche de termes dans des documents, puis la représentation de ces termes compte-tenu de certaines distinctions épistémologiques données (e.g. Physical_entity, Property, Representation_entity, Process, Event). Cette tâche est la première étape du processus de modélisation dans KOD.
3. L'insertion et la (ré-)organisation de types dans une ontologie, par exemple pour pouvoir représenter des termes.
4. L'élicitation ou bien la recherche dans un document, puis la représentation, de relations entre des concepts, et donc, si la recherche se fait dans un document, entre des termes et/ou entre des phrases et éventuellement entre d'autres types d'EDs. La recherche peut être guidée, par exemple par une ontologie de types de relations, par des types de relations typiquement associées à des types de concepts, ou par des modèles de méthodes de résolution générales (e.g. Cover & Differentiate).
5. La recherche et la comparaison de représentations de connaissances afin de les généraliser, les compléter, les structurer ou encore les assembler ou les fusionner.

Les tâches du premier point peuvent être effectuées automatiquement par des outils dédiés. CGKAT facilite le travail du cogniticien dans la réalisation des tâches des quatre derniers points. Comme nous l'avons détaillé dans la section 5.3.3, notre exploitation de WordNet permet à un cogniticien :

1. de trouver rapidement un type "standard" afin d'exprimer le sens d'un mot, et en même temps de l'insérer (avec ses supertypes selon WordNet) dans l'ontologie de l'application : recherche automatique du lieu d'insertion, organisation automatique de l'ontologie à chaque ajout d'un type, création simple d'une ontologie détaillée, extensible et réutilisable ;
2. de classer aisément des types (et donc de représenter des mots d'un document) par rapport à des type de concepts de haut niveau donné (e.g. Physical_entity, Property, Event, etc.). Il suffit pour cela que ces types soient spécialisés par des types de haut niveau de WordNet. Nous illustrons en annexe 8, une fonctionnalité de CGKAT facilitant l'utilisation des types de WordNet lors de la construction de l'ontologie d'une application via des fichiers scripts ou BCGCT. Nous avons également partiellement implémenté une procédure permettant la recherche dans un document en anglais de mots dont l'un des sens correspond ou est une spécialisation de types de concepts donnés. Par ailleurs, rappelons que CGKAT permet de représenter toutes les occurrences d'un mot dans un document par un même concept.

A propos des tâches du quatrième point, CGKAT ne permet pas de rechercher des relations (non encore représentées) entre des EDs d'un document : un analyseur de langage naturel serait nécessaire pour cela. Mais CGKAT fournit une ontologie qui :
1) organise des types de relations signées provenant de différents travaux en représentation des connaissances,
2) organise de nombreux types de concepts sous des types de haut niveau auxquels sont associés, sous forme de définitions par conditions typiques, les sortes de relations typiquement connectables à ces concepts (la section 5.7.1.1 détaille et compare quelques moyens pour utiliser ces schémas en recherche et extraction de connaissances).
Rappelons également que des vérifications sont effectuées lors des modifications des ontologies et des GCs (e.g. respect des signatures de relations). Enfin, des domaines de valeurs pour certains concepts peuvent être spécifiés dans une ontologie sous forme de schémas prototypiques ou sous

forme de marqueurs individuels, et des cogniticiens peuvent être guidés par ces domaines de valeurs et éventuellement s'engager à les respecter. Par exemple, les différentes couleurs possibles des voitures d'un certain modèle peuvent être spécifiées sous forme de marqueurs individuels ; ainsi, lorsqu'un utilisateur doit représenter la couleur d'une voiture de ce modèle, il peut consulter dans le menu de gestion des marqueurs individuels, ceux relatifs à la couleur des voitures de ce modèle et sélectionner le marqueur adéquat.

Pour faciliter les tâches du cinquième point, CGKAT permet des recherches et/ou les comparaisons de GCSs par projection[1], la recherche d'informations ou de connaissances par requête lexicale ou structurelle ou par navigation, la génération de vues sur des critères conceptuels (méta-informations comme "l'auteur de la représentation" comprises), la génération de synthèses de représentation d'EDs symboles, des jointures maximales, et un langage de combinaison de commandes. Rappelons à titre d'exemple comment tous les GCSs d'une base qui spécialisent un graphe donné peuvent être joints ensemble :

```
> sh ck spec #a_GCS_name | ck maxjoin   //les GCs peuvent être donnés sous forme linéaire ou
                                        //avec leurs noms
```

6.4.2 Extraction et modélisation descendante avec CGKAT

Voici des tâches typiques d'extraction ou de modélisation descendantes (pour l'AC) :

1. le choix de modèles de tâches génériques à suivre ;
2. "l'instanciation" d'un modèle de tâche générique, i.e. l'élicitation ou bien la recherche dans des documents, puis la représentation, d'informations relatives à des inférences spécialisant celles spécifiées dans le modèle de tâche générique choisi ; des exemples de telles informations sont les types d'objets du domaine que les inférences manipulent (e.g. des historiques d'accidents), les connaissances exploitées par ces inférences (e.g. des modèles cinématiques, des chaînes causales typiques), les méthodes utilisées pour réaliser ces inférences (e.g. un raisonnement abductif), et le séquencement de telles inférences ;
3. l'adaptation et la combinaison de modèles de tâches génériques.

Le choix d'un modèle de tâche générique dans une bibliothèque de tâches génériques, est facilité
1) par une extraction et modélisation ascendante préalable, et
2) par le nombre de modèles disponibles, leur organisation dans la bibliothèque et des détails sur les contraintes d'application de chacun.
Dans CGKAT, des modèles de tâches (et leurs contraintes d'application) peuvent être représentés et organisés dans une ou plusieurs ontologies, en utilisant le formalisme des GCs. De plus, ils peuvent être construits sous forme d'EDs GCs dans des documents, et ainsi être associés à d'autres informations (i.e. documentés), indexés, inclus dans divers documents (e.g. figures 5.23 et 5.24 à la page 145 et figure 6.5 à la page 177). Ils peuvent alors être recherchés 1) par navigation ou requêtes dans le menu de gestion des types de concepts, et 2) par navigation et requêtes conceptuelles dans des documents. Nous avons détaillé dans la section 2.3.2.2 à la page 33, l'intérêt d'utiliser des vues et une unique ontologie plutôt que des modules, pour représenter et indexer des connaissances, e.g. une bibliothèque de modèles génériques.

Concernant les tâches génériques, l'ontologie par défaut de CGKAT n'organise actuellement que les tâches et les inférences[2] de KADS_I et de CommonKADS (e.g. figures 5.21 et 5.22 à la

1. Notons que la projection ne permet de rechercher dans une base que les GCs qui spécialisent (ou généralisent) un GC donné. Il serait également intéressant de pouvoir rechercher, par "projection partielle", les GCs qui spécialisent (ou généralisent) une "grande" partie d'un GC donné (e.g. 75% des noeuds de ce GC). ROCK (Carbonneill & Haemmerlé, 1994) répond en partie à ce besoin puisqu'il effectue des généralisations d'une requête lorsque celle-ci n'a pas de solution.

2. Rappelons qu'une inférence d'un modèle de tâche est une tâche primitive (cf. section 2.2.1 à la page 21).

page 144), et seules quelques structures d'inférences de modèles de tâches génériques sont représentées (e.g. figures 5.23 et 5.24 à la page 145), mais ce travail peut être complété. Par ailleurs, nous avons spécifié quelques modèles pour la modélisation des connaissances relatives aux explications, dont un modèle générique de la tâche d'explication (cf. (Martin, 1993a, 1993b, 1994)). Enfin, des définitions de types ou des GCs créés par un cogniticien au cours d'une modélisation, peuvent être considérés par ce cogniticien ou d'autres cogniticiens, comme des modèles et être suivis de la même manière que des modèles de tâches génériques.

L'"instanciation" d'un modèle de tâche générique pose deux problèmes : 1) savoir quelles informations relatives aux inférences du modèle éliciter ou bien rechercher, et dans ce dernier cas, 2) comment rechercher ces informations.

- Pour répondre au premier problème, nous avons dressé une typologie d'informations à éliciter ou à rechercher relativement à chaque type de composant d'un modèle de tâche générique (informations concernant la résolution de problèmes et la génération d'explications) (Martin, 1993b, 1994). Cette typologie réutilise et complète les listes de relations typiquement connectables à des concepts de certains types (cf. section 5.7.1).
- La recherche dans des documents d'informations relatives à des inférences d'un modèle de tâche générique nécessite une compréhension profonde à la fois des sortes d'inférences à rechercher et du contenu sémantique des EDs. Aussi, cette recherche semble difficilement automatisable. Cependant, si à la suite d'une phase de modélisation ascendante, des connaissances sont déjà représentées, il est possible avec des requêtes conceptuelles de rechercher dans ces connaissances, des spécialisations d'inférences génériques. Il vaut alors mieux que la phase de modélisation ascendante ait été guidée par une liste d'inférences génériques à rechercher et à instancier (et si tel est le cas, les requêtes peuvent contenir plus d'une inférence de façon à rechercher des "blocs d'inférences"). Les GCs contenant les inférences ou les blocs d'inférences recherchés peuvent alors être joints (par jointure simple ou maximale), et le cogniticien peut être guidé dans ces jointures par des structures d'inférences contenant et *organisant* les inférences ou blocs d'inférences recherchés (ces structures d'inférences peuvent avoir été recherchées dans les définitions associées aux types de tâches génériques, avec les même requêtes que celles utilisées pour trouver les GCs à joindre). La jointure de GCs guidée par une structure d'inférence, ou par tout autre élément d'un modèle de tâche générique représenté dans le formalisme des GCs, est une manière d'adapter et d'"instancier" ce modèle de tâche générique.

La construction d'un modèle de tâche peut s'effectuer par adaptation et combinaison de modèles de tâches génériques. Une combinaison de structures d'inférences peut par exemple être faite sur toutes les structures d'inférences contenant un certain bloc d'inférences :

```
> sh ck on def; ck spec [Select]- { (DI-)←[Observable];        //cette commande est
                                    (SI-)←[Hypothesis];        //similaire à la dernière
                                    (O)→[Finding];             //de la figure 6.24
                                  } | ck maxjoin
```

Un réseau de dépendance de données entre tâches génériques, tel celui de Breuker (1994) (cf. figure 2.8 à la page 27), peut également entraîner le cogniticien à combiner certaines structures d'inférences ; même dans ce cas, la commande de jointure maximale peut être utile :

```
> maxjoin #SI_for_Systematic_Diagnosis #SI_for_Repair
```

6.4.3 Utilisation de CGKAT en combinaison avec d'autres outils

Les commandes de CGKAT peuvent être appelées depuis le shell, et donc depuis n'importe quel script ou programme. Dans ces cas, comme pour toutes les commandes appelées depuis le shell, les résultats d'une commande sont affichées sur la sortie standard, laquelle peut être redirigée dans un fichier ou dans l'entrée d'une autre commande. L'appel et la combinaison de commandes de CGKAT avec d'autres commandes sont ainsi aisés depuis le shell ou depuis un script shell. Si l'appel s'effectue depuis un programme et non depuis un script, le programme doit gérer des lectures et éventuellement des écritures dans des fichiers pour pouvoir exploiter des commandes de CGKAT (comme lorsque des "pipes" ou des "sockets" sont utilisés par divers processus pour communiquer entre eux).

Nous fournissons également une interface fonctionnelle permettant d'appeler directement les fonctions de gestion de la base de CGKAT plutôt que de passer par des commandes. Ces fonctions peuvent être appelées depuis un programme par RPC [1].

6.4.3.1 Une expérience : échanges de données entre CoKaCe et CGKAT

Basé sur Centaur (Borras & al., 1987), CoKaCe (Corby & Dieng, 1996)[2] est un environnement de programmation dédié au langage CommonKADS-CML (Schreiber & al., 1992). Il comprend un éditeur syntaxique, un vérificateur de types et un interpréteur de ce langage. Cet interpréteur peut lancer un raisonnement en invoquant une tâche avec des paramètres d'entrée. Une tâche invoque des sous-tâches qui elles-même déclenchent des inférences. Les inférences sont réalisées par des déductions sur les connaissances du domaine, e.g. en chaînage avant ou en chaînage arrière.

Il nous a semblé intéressant de compléter CoKaCe par CGKAT, et vice-versa. Par exemple, il peut être intéressant de traduire un modèle d'expertise CommonKADS-CML dans le formalisme des GCs de façon à exploiter les recherches de connaissances permises dans ce formalisme.

Comme dans la plupart des outils d'AC, les commandes de CoKaCe sont uniquement accessibles par menu et non également depuis le shell. Cependant,
1) Centaur fournit une interface fonctionnelle permettant de manipuler la structure logique dans laquelle sont stockées les représentations de connaissances, et
2) des représentations de connaissances peuvent être lues ou stockées dans des fichiers de sauvegarde.
Dans le but de compléter CoKaCe par CGKAT et vice-versa, pour des raisons de simplicité, nous nous sommes limités à l'échange de représentation de connaissances entre ces outils via des fichiers.

Un traducteur du langage CommonKADS-CML vers le formalisme des GCs a été écrit par Olivier Corby en utilisant Centaur. Ce traducteur génère un fichier script contenant des commandes de CGKAT permettant de définir des types et d'introduire des GCs. Ainsi, des connaissances construites dans CoKaCe peuvent être chargées dans CGKAT puis manipulées à la main ou via des scripts ou des programmes. Les fonctions de recherche par projection et de jointure maximale sont par exemple d'intéressants compléments à CoKaCe.

Le traducteur inverse, i.e. du formalisme des GCs vers CommonKADS-CML, n'a pas encore été écrit. Il imposera à l'utilisateur de CGKAT d'utiliser une ontologie particulière afin qu'un modèle de tâche construit sous CGKAT puisse être converti en un modèle de tâche sous CommonKADS-CML. Aucune ontologie particulière ne sera à respecter pour les connaissances du domaine, car CommonKADS-CML permet de les modéliser sous la forme d'un réseau de concepts et de relations. Ainsi, les GCSs devraient pouvoir être convertis dans CommonKADS-CML.

---

1. Une option de compilation du code de CGKAT permet cela. Nous utilisions cette option au début du développement de CGKAT. Actuellement, pour des raisons pratiques, nous ne l'utilisons pas.

2. Cet outil a été développé par Olivier Corby.

Chapitre 7 Implémentation

7 Sommaire

7.1 Architecture de CGKAT : une approche ouverte

Le code de CGKAT est divisé en deux parties sur un modèle client-serveur (cf. figure suivante). Le serveur est un ensemble de fonctions permettant de construire et d'effectuer des recherches dans l'ontologie et la base de GCs. Le client est l'ensemble des fonctions qui s'occupent de l'interface utilisateur, via des menus ou des documents. Ces deux parties peuvent être compilées ensemble, ou bien séparément et alors communiquer par RPC.

Il peut ainsi être envisagé d'étendre CGKAT de telle sorte que plusieurs processus clients puissent avoir accés au même serveur et donc travailler sur la même base. L'extension serait assez simple si Unix et RPC sont exploités : tous les accès et calculs sur la base seraient effectués par un unique processus. Notons également qu'une version distribuée de CoGITo a été spécifiée et en grande partie implémentée (Haemmerlé, 1995a, 1995c). Contrairement à l'extension de CGKAT que nous suggérons, la version distribuée de CoGITo permet à plusieurs processus d'effectuer en parallèle des recherches ou des calculs sur la base : un processus central gère l'accès aux données et joue donc le rôle de serveur pour plusieurs processus clients utilisant l'interface fonctionnelle similaire à celle de la version de CoGITo non distribuée.

Nous avons également noté qu'une application telle que Alliance (Decouchant, 1995) permet une édition coopérative sur des documents via plusieurs éditeurs Thot. Chaque processus CGKAT étant formé d'un éditeur Thot et d'un gestionnaire de base de GCs (il y a donc une base par processus), il serait intéressant de vérifier qu'une extension minime de CGKAT permette que les modifications effectuées sur un ED CGgraph, TypeDefinition ou CGRepr dans un document partagé, soient répercutées dans toutes les bases des processus CGKAT partageant le document.

Enfin, grâce à son langage de commandes, CGKAT peut être utilisé par d'autres applications. L'une d'elles peut gérer, si nécessaire, les accès à CGKAT.

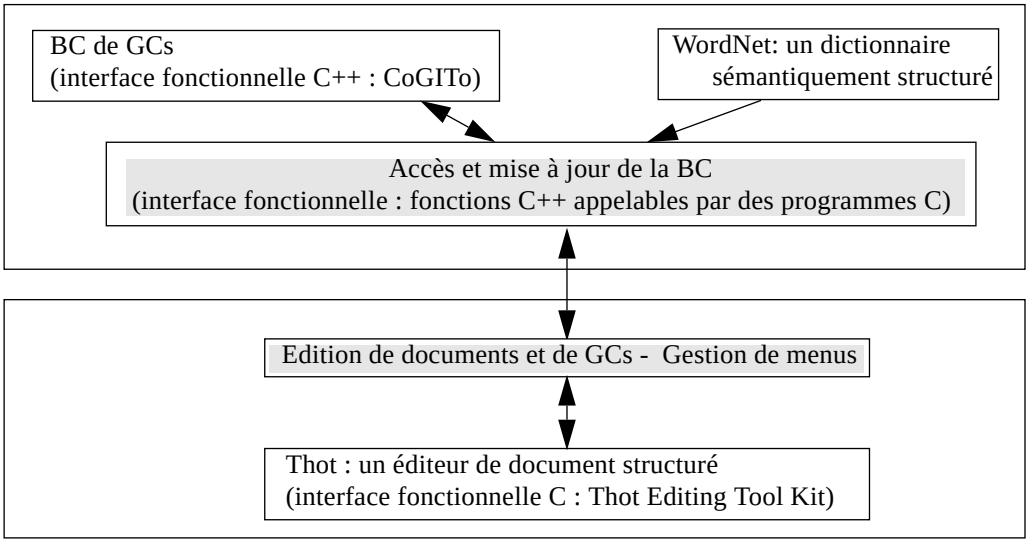


Figure 7.1: *Architecture de CGKAT* (les parties grisées représentent notre code).

Pour donner un ordre d'idée, le code source de CGKAT comprend
1) côté serveur (CoGITo[1] non inclus), environ 4000 lignes de code (125.000 caractères),
2) côté client, environ 10.000 lignes de code (410.000 caractères) dont 1500 lignes (60.000 caractères) pour les modèles structurels et de présentation (le reste étant du code C).

Nous détaillons ci-après la partie serveur puis la partie client de CGKAT.

1. Le code source de la version de CoGITo que nous utilisons fait environ 12.800 lignes (242.000 caractères).

7.2 Exploitation de CoGITo et de WordNet

7.2.1 L'interface fonctionnelle de CGKAT pour la manipulation d'une base de Graphes Conceptuels

CoGITo est un ensemble de classes d'objets écrites en C++, implémentant et permettant de manipuler des objets du formalisme des GCs : type de concept, treillis de types de concepts, type de relation, hiérarchie de types de relations, marqueur individuel, liste des marqueurs individuels, concept, relation, GCS, lambda-abstraction, ensemble de GCSs et de lambda-abstractions. Les méthodes d'une classe d'objet (e.g. celle d'un concept) permettent de créer un objet de cette classe, de le modifier, de le comparer à d'autres objets de la même classe, ou encore de le sauvegarder dans un fichier. Pour utiliser une méthode d'un objet, il faut disposer d'un pointeur sur cet objet ainsi que sur les objets pris en paramètre par la méthode. Un pointeur sur un objet est obtenu à la suite de la création ou bien de la recherche de cet objet (e.g. une méthode du treillis des types de concepts fournit, étant donné un nom de type de concept, un pointeur sur ce type).

En résumé, la manipulation d'un objet dans CoGITo s'effectue via des pointeurs et l'obtention de ces pointeurs nécessite plusieurs instructions préalables. Dans la partie serveur de CGKAT, nous avons donc dû créer des fonctions pour permettre la manipulation des objets de CoGITo sans avoir à obtenir des pointeurs sur ces objets mais simplement en fournissant des noms et des valeurs d'initialisation pour ces objets. Ainsi, les programmes clients de ce serveur n'ont pas à connaître (déclarer et utiliser) les classes d'objets de CoGITo puisque les échanges s'effectuent uniquement avec des chaînes de caractères ou des valeurs numériques. L'annexe 6 détaille l'interface fonctionnelle que nous avons créé pour ce serveur afin de satisfaire les besoins de la partie client de CGKAT. Voici à titre d'exemple, le contenu d'une fonction très simple, celle permettant de créer et d'ajouter un concept à un graphe.

```
char* SgAddConceptToCG (char *cgName, char *cgSet,                    //détails sur le GC
                        char *cTypeName, char *refName, char *comm,   //détails sur le concept
                        initP int *conceptIdent)   //la fonction initialise ce paramètre
{  ptrGC ← recherche du GC de nom cgName dans l'ensemble cgSet;
      si (ptrGC==NULL) retour d'une chaîne contenant un message d'erreur;
   ptrCType ← recherche du type de nom cTypeName dans le treillis de type de concept;
      si (ptrCType==NULL) retour d'une chaîne contenant un message d'erreur;
   si (refName=="*") ptrRef ← NULL;  //référent générique et ne contenant pas de variable
   sinon  //nom d'une variable (e.g. "*x") ou d'un marqueur individuel (e.g. "#phmartin")
   { ptrRef ← recherche de la variable ou du marqueur individuel de nom refName;
     si (ptrRef==NULL) ptrRef ← création d'une variable ou d'un marqueur individuel de nom refName;
   }
   ptrCo ← création d'un concept de type ptrCType, de référent ptrRef, et de commentaire comm;
   ajouter le concept ptrCo à la liste des concepts du graphe ptrGC;
   conceptIdent ← identificateur du concept ptrCo dans la liste des concepts du graphe ptrGC;
   retour d'une chaîne vide;  //pas de message d'erreur
}
```

CoGITo permet de manipuler plusieurs "environnements" en même temps, chaque environnement contenant un support (i.e. un treillis de types de concepts, une hiérarchie de types de relations et une liste des marqueurs individuels) et des ensembles (zones) de GCSs ou de lambda-abstractions. Pour des raisons de simplicité, CGKAT n'exploite qu'un seul environnement. Ainsi, les fonctions de la partie serveur de CGKAT n'ont pas d'identificateur d'environnement en paramètre, et les divers types nécessaires aux graphes de la base doivent appartenir au même support. Si l'utilisateur veut travailler avec différents supports en même temps, il doit donc lancer différents processus CGKAT.

Nous rappelons ou détaillons dans les deux sections suivantes, les extensions de CGKAT par rapport à CoGITo, concernant la gestion de la base.

7.2.2 Gestion de types

7.2.2.1 Définitions de types de concepts ou de types de relations

Nous avons permis la construction de définitions de types sous format graphique (ED TypeDefinition), sous format linéaire (cf. langage de commandes) et via la fonction SgDefineType(). Lorsqu'une lambda-abstraction définit un type de concept, CGKAT définit ce type comme un sous-type du type du concept porteur de la variable paramètre de la définition (cf. section 6.2.2). CGKAT vérifie que l'insertion d'un type dans une hiérarchie, directement ou via une définition de type, ne créé pas de cycle dans cette hiérarchie (c'est-à-dire que le sous-type inséré ne soit pas déjà un supertype de ses nouveaux supertypes).

7.2.2.2 Liens entre types de concepts ou entre types de relations

Nous avons modifié CoGITo et étendu la syntaxe du format BCGCT (le format des fichiers de sauvegarde générés par CoGITo) pour permettre la prise en compte et le stockage de liens entre types (quels que soient les noms donnés à ces liens). Cependant, seuls les liens d'exclusion (liens de nom "Excl") ont une sémantique codée dans CGKAT. D'autres modules doivent être ajoutés à CGKAT pour *gérer* d'autres sortes de liens entre types, e.g. des liens de nom "Part" (cf. section 5.2.3).

Pour faire respecter la sémantique des liens d'exclusion, CGKAT vérifie lors du sous-typage d'un type A par un type B ayant déjà des supertypes, qu'aucun de ces supertypes (directs et indirects) de B n'a de lien d'exclusion avec le type A ou ses supertypes.

Tous les liens entre types sont orientés, mais lorsque l'utilisateur déclare un lien entre un type X et un type Y comme un lien "symétrique", ce qui doit être le cas pour un lien d'exclusion, CGKAT pose automatiquement deux liens : un entre X et Y, et un entre Y et X.

Voici un exemple commenté de fichier BCGCT pour illustrer ces fonctionnalités.

```
Begin
  Support: Test1(100,100,100);
      Lattice:
          ConceptTypes:          //Déclaration de types de concept
  Color{Comment:exemple de commentaire pour un type de concept;};
  Red; Orange; DarkOrange; Blue;
          EndConceptTypes;

          SymRelOnTypes:         //Nouveau mot-clé → permet la déclaration
                                 //de liens "symétriques" entre types de concepts
  Red Excl Blue;       //"Blue Excl Red" est rajouté automatiquement par CGKAT.
  Red Excl Orange;     //Le lien "Excl" est actuellement le seul lien
  Orange Excl Blue;    // qui ait des consequences (cf ci-dessous)
  Orange Opp Blue;     //Les autres liens sont seulement stockés et sont à gérer par l' utilisateur
          AsymRelOnTypes:        //Nouveau mot-clé → liens assymétriques
  Orange Part Red;     // "red Part Orange" n' est pas rajouté
          EndRelOnTypes;

          Order:                 //Ordre entre les types de concept
  Red < Color;
  Orange < Color;
  DarkOrange < Orange;
  Orange < Red;        //Non pris en compte par CGKAT à cause de "Red Excl Orange";
  DarkOrange < Red;    //Non pris en compte par CGKAT à cause de "Red Excl Orange";
          EndOrder;
      EndLattice;
```

```
        TRelSet:
            RelationTypes:
Binary_relation{Signature:2,Concept,Concept} {exemple de commentaire d' un type de relation;};
            EndRelationTypes;
        EndTRelSet;

    EndSupport;
End
```

7.2.2.3 Recherche de types ou de marqueurs individuels et inclusion dynamique de types de WordNet

CGKAT permet de rechercher les types ou les marqueurs individuels contenant une certaine chaîne de caractères. L'interface de WordNet est également exploitée pour permettre la recherche des "ensembles de synonymes" contenant un certain terme. Dans ce cas, des noms de types uniques sont générés pour les "ensembles de synonymes" retrouvés, et ces types sont temporairement inclus dans le treillis des types de concepts avec leurs supertypes non déjà inclus. La recherche peut également s'effectuer sur les liens de spécialisation entre types du treillis ou entre ensembles de synonymes.

L'affichage des résultats d'une recherche peut s'effectuer sous forme de liste indentée, dans un menu de gestion de types (de concepts ou de relations), et sous forme de graphes via l'appel d'un générateur de graphes fonctionnant sous Netscape[1]. La recherche par navigation dans un menu de gestion de types montre des supertypes et des sous-types d'un type sélectionné (figure 5.3). Lorsque des types contenant une certaine chaîne de caractères sont retrouvés, des supertypes et des sous-types de chacun des types retrouvés sont affichés (figure 5.6). Lors de chaque recherche, les types de WordNet inclus lors de la recherche précédente et non sélectionnés pour inclusion définitive, sont préalablement ôtés du treillis des types de concepts.

Actuellement CGKAT ne permet pas de stocker des hiérarchies de types dans des documents. Aussi, des fichiers scripts ou des fichiers BCGCT doivent être utilisés pour cela. Une hiérarchie de types peut être construite via les menus de gestion de types (de concepts ou de relations) mais la sauvegarde s'effectue alors dans un seul fichier, alors que nous avons facilité la modularisation manuelle d'une hiérarchie dans plusieurs fichiers (cf. annexe 8)[2]. Les menus de gestion de types dans CGKAT sont intéressants pour visualiser des portions de hiérarchies, mais encore peu adaptés pour la construction de grandes hiérarchies où les types ont de nombreux supertypes directs. Dans ce genre de cas, nous conseillons plutôt d'utiliser en parallèle 1) un éditeur de texte pour éditer et modulariser une hiérarchie dans un fichier script ou BCGCT (un tel fichier peut être commenté et contenir différentes versions dont les chargements dépendent du contexte), et 2) les menus de gestion de types pour visualiser des portions de la hiérarchie.
Afin d'alléger l'utilisation des types de WordNet lors de la construction de tels fichiers, CGKAT offre une commande pour charger les supertypes de types de WordNet inclus dans le treillis ; ainsi, un cogniticien peut, lorsqu'il construit de tels fichiers, positionner les types de son application par rapport aux types de WordNet, mais il n'a pas à déclarer leurs supertypes ; ceux-ci sont chargés automatiquement lorsque la commande citée ci-dessus est activée. La deuxième section de l'annexe 10 montre un fichier BCGCT exploitant cette fonctionnalité.

---

1. Ce générateur de graphes a été développé avec le langage Java par Chritophe Cointe, un membre de l'équipe Acacia. Pour l'utiliser, CGKAT génère un fichier html contenant la liste des types à visualiser et les liens de spécialisation qui les organise, puis appelle NetScape avec l'adresse http du générateur de graphes.

2. Dans un fichier script, une commande peut entraîner l'exécution d'un autre fichier script. Pour faciliter la modularisation d'une hiérarchie dans des fichiers BCGCT, nous avons modifié CoGITo sur deux points : 1) le préprocesseur cpp est désormais appelé sur un fichier BCGCT avant que son contenu soit interprété (ce fichier peut ainsi contenir des macros de test et d'inclusion de fichiers tout comme dans un programme C), et 2) les fichiers BCGCT peuvent maintenant être découpés en blocs "anonymes" et non uniquement "nommés" (un bloc nommé spécifie le nom du support dans lequel il doit être chargé, tandis qu'un bloc anonyme peut être chargé dans n'importe quel support). L'annexe 8 illustre l'intérêt de cette modularisation.

7.2.2.4 Destruction de types

Via un menu de gestion de types, son langage de commandes ou son interface fonctionnelle, CGKAT peut détruire un type, avec ou sans tous ses sous-types. Dans le premier cas, il peut vérifier si le type n'est pas utilisé dans un des graphes de la base (une recherche par projection est actuellement utilisée pour cela).

7.2.3 Gestion de graphes

7.2.3.1 Indexation

Pour indexer les graphes en *mémoire centrale*, CoGITo offre un tableau de zones de graphes et des fonctions permettant de construire, détruire et rechercher des graphes dans des zones de numéros donnés. Nous avons tenté d'utiliser ce tableau de zones pour implémenter la gestion d'une indexation des graphes en fonction de leurs méta-informations, certaines zones étant alors destinées à recevoir des graphes possédant telle méta-information ou tel groupe de méta-informations. Cependant avec ce type d'indexation, compte-tenu qu'un graphe ne peut appartenir qu'à une seule zone, l'utilisation d'une nouvelle méta-information entraînait la réorganisation de l'indexation et donc des graphes dans les zones, ce qui obligeait à d'importantes modifications dans le code de CoGITo. Nous avons donc abandonné ce projet qui n'était pas central à la thèse. Pour faciliter l'indexation de graphes, l'intégration à CoGITo d'une autre structure de données qu'un tableau de zones, e.g. une hiérarchie ou une table de hash-code, semble nécessaire.

CoGITo n'offre pas de moyen d'indexer des graphes stockés en *mémoire secondaire*. Des travaux sont en cours sur ce point dans l'équipe du LIRMM qui a développé CoGITo. Pour notre part, nous avons utilisé à ce sujet un moyen d'indexation limité mais simple : chaque graphe construit via un ED GC est également stocké seul dans un fichier au format BCGCT dont le nom est formé à partir de celui du graphe ; comme le nom d'un graphe construit via un ED GC est généré par CGKAT à partir du nom du document, de l'identificateur de l'ED GC, de l'identificateur du créateur du graphe et du nom du point de vue pris en compte pour créer le graphe (par défaut "AnyView"), les fichiers de graphes (et donc les graphes à charger en mémoire centrale) peuvent être sélectionnés sur ces informations, par exemple via le langage de commandes de CGKAT :

```
> sh ck load .G_Examples_*_phmartin_*.PIV    //chargement depuis CGKAT des graphes contenus
                                             //créés dans le document Example par l'utilisateur phmartin
$ load .G_*_*_*_Task.PIV                     //chargement dans CGKAT depuis le shell de tous
                                             //les graphes relatifs à la vue Task quel que soit le document
                                             //où ils ont été créés et l'utilisateur qui les a créés.
```

7.2.3.2 Graphes emboîtés

CGKAT permet de construire des GCs emboîtés sous forme linéaire ou graphique (cf. sections 4.3.2.4.1, 4.3.2.4.2 et 6.2.2) et d'effectuer des vérifications sur la construction de ces graphes, moyennant quelques restrictions ontologiques : un concept qui emboîte un graphe représente ce graphe en tant que description, et non le medium (support d'information) utilisé par le graphe, ni la situation qu'il décrit, ni autre chose encore (module de connaissances, etc.). Ainsi, le référent individuel d'un concept qui emboîte un graphe est un pointeur explicite sur ce graphe. CGKAT stocke également dans le champ commentaire associé au graphe, le pointeur inverse (ce pointeur là est invisible pour l'utilisateur). Lorsqu'un graphe est construit via un ED GC, CGKAT génère le nom du graphe et donc le référent individuel d'un concept qui l'emboîte. Mais l'utilisateur peut changer ce nom. Dans le format linéaire, l'utilisateur affecte lui-même un nom à un graphe qu'il construit.

Les contextualisations de graphes effectuées via leurs emboîtements sont prises en compte dans les requêtes conceptuelles : un graphe contextualisé est toujours présenté avec son contexte. Pour cela, trois noms de types ont été introduits en dur dans notre code : Contextualizing_relation, Contextualizing_proposition et Process_contextualizing_its_object.

7.2.3.3 Requêtes conceptuelles

CGKAT utilise la fonction CoGITo de projection d'un graphe sur un autre pour rechercher les spécialisation ou les généralisations d'un graphe donné. Si le graphe est accompagné de méta-informations, la relation de spécialisation/généralisation que nous avons définie est également vérifiée (cf. section 5.2.2.2). Un graphe requête peut également contenir des chaînes de caractères quelconques à la place des noms de types de concepts, CGKAT remplace ces chaînes par les types qui contiennent ces chaînes et génère ainsi autant de véritables graphes requête qu'il y a de combinaison valides. L'utilisateur peut contrôler la présentation des résultats d'une requête (cf. section 6.3.2.2.2) : format linéaire, inclusion de l'ED GC ayant servi à construire le GC résultat et/ou inclusion d'un ED indexé par ce GC (représentation/annotation/indexation libre).

Lorsque la partie client de CGKAT demande à la partie serveur les spécialisations ou généralisations d'un graphe requête, cette dernière ne retourne pas les résultats dans des paramètres mais les stocke dans un fichier (car les données retournées peuvent être nombreuses). Pour chaque GC résultat, la partie serveur fournit
1) un GC au format linéaire si une telle présentation a été demandée, et/ou
2) l'identificateur de l'ED GC ayant servi à construire le GC, accompagné du nom du document contenant cet ED, si cet ED existe et si une présentation avec un ED a été demandé.
Dans ce dernier cas, c'est la partie client qui recherche l'ED et si besoin les EDs qu'il indexe, afin de les présenter sous forme d'inclusions. La présentation des résultats s'effectue par construction d'EDs à l'intérieur d'un ED Answers (cf. section 6.2.5).

Nous présentons ci-dessous les fonctions principales relatives à la recherche de spécialisations (un paramètre supplémentaire permet de réutiliser ces fonctions pour rechercher des généralisations). Les "graphes" ou "GCs" auxquels nous faisons référence dans ces fonctions sont des GCSs. Cependant, les référents individuels des concepts de ces GCSs peuvent référer d'autres GCSs (dans ce cas, par abus de langage, nous disons que les premiers "emboîtent" les seconds et donc qu'il sont "emboîtants").

RechercheDeSpécialisations (graphe requête initial, type de résultats choisi) → fichier des résultats
{ Pour chacun des GCs requête issus du graphe requête //remplacement des chaînes par des types
Pour chacune des *SpécialisationsDuGC* (gc requête courant)
Suivant le format choisi, ajouter dans le fichier des résultats, la forme linéaire
du gc résultat courant ou bien l'identificateur de l'ED GC ayant servi à le construire
}

SpécialisationsDuGC (gc requête) → liste de gc spécialisations
{ Pour chacun des GCs de la base //parcours séquentiel (pas de gestion d'indexation)
Si *EstUneSpécialisationDe* (gc courant, gc requête)
S'il y a des GCs contextualisant le gc courant //recherche séquentielle[1] des GCs emboîtants
Ajouter les GCs les plus emboîtants à la liste de gc spécialisations
Sinon Ajouter le gc courant à la liste de gc spécialisations
}

EstUneSpécialisationDe (gc1, gc2) → Vrai/Faux
{ Si les méta-informations associées à gc1 spécialisent celles de gc2
et s'il y a au moins une façon de projeter gc2 dans gc1, Retourner Vrai
Sinon Retourner Faux
}

1. Comme indiqué en fin de section 6.3.2.2.3, la recherche s'effectue dans la base courante. Actuellement, cette recherche s'effectue par un parcours séquentiel des graphes de la base : sur chacun d'eux CGKAT teste s'il contextualise le gc courant, et dans ce cas, s'il n'est pas lui-même contextualisé par d'autres graphes.

7.2.3.4 Jointure maximale

CGKAT permet d'exploiter, via son interface fonctionnelle ou son langage de commandes, les méthodes de jointure simple et d'isojointure (cf. section 4.2.2.3.2 à la page 76) de CoGITo. Ces méthodes créent un nouveau graphe par jointure de deux graphes sur deux concepts donnés et fusionnables (la méthode d'isojoint essaie d'étendre cette jointure au maximum). Nous avons implémenté une jointure maximale sur deux graphes en testant tous les isojoints possibles entre les deux graphes donnés et en ne retenant que celui dont le nombre de sommets joints est maximum. Cette méthode est polynomiale et ses résultats ont jusqu'à présent été conformes à ce que nous en attendions (au niveau de la fonctionnalité). Voici l'algorithme général de cette jointure maximale :

JointureMaximaleDe (gc1, gc2) → gcRésultat
{ nbSommetsJointureMax ← 0
  Pour chacun des concepts (c1) de gc1
    Pour chacun des concepts (c2) de gc2
      gcJointure ← isojoint de gc1 et gc2 sur c1 et c2
      Si le nombre de concepts et de relations dans gcJointure est supérieur à nbSommetsJointureMax
        Enregistrer ce nouveau maximum et sauvegarder c1 et c2 dans c1Max et c2Max
      Détruire gcJointure
  gcRésultat ← isojoint de gc1 et gc2 sur c1Max et c2Max
  Donner un nom unique à gcRésultat en concaténant "MaxJoin_of_", le nom de gc1 et celui de gc2
}

Afin de pouvoir joindre les corps de lambda-abstractions (dans ces graphes, le concept "genus" a pour référent une variable), nous avons légèrement étendu la fonction de CoGITo testant si deux concepts sont fusionnables : les référents variables sont maintenant traités mais de la même façon que des référents génériques. Ces graphes peuvent donc être joints à d'autres graphes mais si différentes variables sont utilisées dans ces graphes, le résultat peut ne pas être correct. Par ailleurs, bien que dans CGKAT les arguments d'une jointure maximale puissent être des définitions de types (la fonction travaille alors sur les corps de ces définitions), le résultat n'est pas une définition de type. L'utilisateur devra donc probablement transformer le GC résultat en une définition de type.

7.3 Exploitation de Thot

Pour permettre la construction dans Thot d'EDs CG, CGRepr, Hierarchy et CGRequest, nous avons construit des modèles structurels et de présentation pour de tels EDs. Nous avons également écrit un modèle structurel d'extension, et un modèle de présentation associé, pour permettre l'indexation d'EDs par des EDs CG ou CGRepr. Enfin et surtout, pour permettre la répercussion dans CoGITo des actions effectuées sur ces EDs et leurs composants (e.g. sélection, construction, destruction, chargement et sauvegarde), nous avons écrit des modèles d'interface et les fonctions C spécifiées dans ces modèles.

Ces modèles structurels, de présentation et d'interface sont décrits en annexe 4. Ces modèles constituent un peu moins du sixième du code que nous avons écrit pour CGKAT, côté client, lequel est environ trois fois plus important que celui côté serveur. La moitié de notre code pour CGKAT est ainsi constitué de fonctions C utilisant
1) l'interface fonctionnelle de Thot permettant de rechercher et de modifier des informations dans la structure logique des documents,
2) notre interface fonctionnelle pour le serveur de CGKAT (cf. annexe 7), et
3) l'interface fonctionnelle de Thot permettant de générer et gérer des menus.
Nous présentons les menus de CGKAT dans l'annexe 1.

7.3.1 Usage de l'interface fonctionnelle de Thot pour la gestion de structures logiques

Les fonctionnalités offertes par l'éditeur Thot sont également accessibles via son interface fonctionnelle, ce qui permet à un programme de gérer un document structuré. Cependant, comme le but de cette interface est de permettre la gestion d'une structure logique, i.e. d'une structure d'arbre dont les éléments ont des types différents, son usage est assez lourd et délicat car chaque accès ou création d'informations (e.g. EDs, attributs, valeurs d'EDs) impose généralement de parcourir des portions de la structure logique (c'est-à-dire d'effectuer des parcours d'arbre et/ou de listes) et pour utiliser les fonctions permettant cela, il faut construire des pointeurs sur les types des EDs ou attributs à traverser. Lorsqu'un type d'ED est défini dans un modèle structurel comme un choix entre divers autres types d'EDs, des tests de types doivent être effectués lors du parcours de tels EDs dans la structure logique.

Certaines gestions de la structure logique sont assez fréquentes et indépendantes de la structure des EDs. Nous avons donc créé des fonctions pour de telles gestions afin d'abstraire et simplifier l'écriture du code de CGKAT. Nous précisons en annexe 7 l'interface fonctionnelle de ces fonctions. Voici à titre d'exemple, le contenu de deux fonctions très simples mais omniprésentes dans le code de la partie client de CGKAT : l'ajout d'un sous-élément à un ED et l'accès à un attribut d'ED et à sa valeur.

```
        /* Cette fonction prend en paramètre 1) des pointeurs sur un ED, un document qui le contient,
           et un modèle structurel et 2) un identificateur de type d'ED. Elle crée un nouvel ED de type
           celui spécifié, l'ajoute comme sous-élément de l'ED donné en paramètre, et retourne le
           pointeur de ce nouvel ED */
Element GtMkSonTo (Element elem, Document doc, SSchema sschema, int elemIdent)
{ ElementType elemType; /* → va pointer sur le type du nouvel ED */
  Element son;   /* → va pointer sur le nouvel ED */
   elemType.ElSSchema = sschema;    elemType.ElTypeNum = elemIdent;
   son = GtNewElement (doc, elemType); /* création du nouvel ED */
   GtInsertFirstChild (&son, elem, doc);   /* insertion dans doc en tant que composant de elem */
   return son;
}

        /* Cette fonction prend en paramètre 1) des pointeurs sur un ED, un modèle structurel,
           et 2) un identificateur de type d'attribut. Elle retourne un pointeur sur l'attribut ainsi
           spécifié (→ NULL si aucun attribut de ce type n'est porté par l'ED) et la valeur de cet
           attribut (→ -1 si aucun attribut du type spécifié n'est porté par l'ED) */
Attribute GtGetAttrValue (Element elem, SSchema sschema, int attrIdent, initP int *pValue)
{ AttributeType attrType; Attribute attr;
   attrType.ElSSchema = sschema;    attrType.ElTypeNum = ident;
   attr = GtGetAttribute (elem, attrType);
   if (attr)   *pValue = GtGetAttributeValue (attr);
   else  *pValue = -1;
   return attr;
}
```

L'abstraction atteinte avec une telle interface reste néanmoins faible. Le code est très dépendant de la structure des EDs manipulés, si des "accesseurs" ne sont pas utilisées ; par le terme "accesseurs", nous désignons des fonctions encapsulant des méthodes d'accès à certains EDs ou attributs depuis certains autres EDs ou attributs. Ainsi par exemple, sans accesseurs, si un modèle structurel est modifié de telle sorte que le premier composant d'un ED devienne le deuxième sous-composant ou encore un sous-sous-composant, le code devra vraisemblablement être modifié en de multiples endroits.

Par ailleurs, les vérifications de types que peut effectuer le compilateur sont limitées : elles ne concernent pas les types d'EDs décrits dans les modèles structurels, mais seulement les types Element, Attribute, ElementType, Attribute, SSchema, etc. Le contrôle sur les types de la structure manipulée est effectué par Thot lors de l'exécution. Comme le contenu exact de la structure logique est parfois difficile à connaître, surtout pour le débutant dans la programmation avec Thot[1], les erreurs sur ces types sont assez nombreuses[2].

Pour minimiser ces problèmes, l'utilisation d'accesseurs est souhaitable. Le problème est qu'il peut être difficile de savoir par avance lorsque l'on écrit un programme (et ce fut le cas dans CGKAT), quels EDs ou attributs vont être atteints depuis d'autres EDs ou attributs. Nous avons pensé à utiliser des classes d'objets pour encapsuler de manière systématique dans des accesseurs, l'accès depuis différentes sortes d'objets représentés dans la structure logique, aux composants de ces objets. Des contrôles sur la manipulation des types d'EDs décrits dans les modèles structurels peuvent alors être effectués par le compilateur. Cependant avec une telle méthode, de nombreux accesseurs sont à écrire (et à modifier en cas de modification du modèle structurel concerné) alors qu'il est probable que peu de ces accesseurs seront utilisés (par exemple dans CGKAT, il est assez fréquent d'atteindre à un ED depuis un de ses attributs alors que dans un modèle objet, c'est l'accès inverse qui serait permis par un accesseur). Aussi dans CGKAT, nous n'avons encapsulé dans des accesseurs que des méthodes fréquemment utilisées pour retrouver et mettre à jour des composants de GCs : ajout d'un ED Concept dans un ED GC, accès et gestion de certains attributs associés aux EDs Concept ou GC tel que les attributs Comment, CGName et IDinCG, etc.

Voici à titre d'exemple une des fonctions les plus simples en ce qui concerne la répercussion dans CoGITo d'événements dans Thot sur des EDs GC ou leurs composants. Cette fonction est appelée par Thot quand l'utilisateur a ajouté dans un graphe un concept qu'il a préalablement copié dans un autre graphe ou dans le même graphe (cf. le modèle d'interface pour les EDs GC dans l'annexe 6).

```
void ConceptPastePost (NotifyElement *event)
{  SSchema cgSch = GtGetSSchema("CG", LDoc = event→document);
   Element concept = event→element;   Element dad = GtGetParent (concept);
   Element conceptType, referent, refVarOrIndiv;
    char typeName[100]="", refName[100];     int lTypeName=99, lRef=99;

     conceptType = GtGetFirstChild(concept);
     GtGiveTextContent(GtGetFirstChild(conceptType),typeName, &lTypeName,&Lang);
     referent = GtGetSibling(conceptType);
     if (referent  &&  (refVarOrIndiv=GtGetFirstChild(referent)))
        GtGiveTextContent(GtGetFirstChild(refVarOrIndiv),refName, &lRef,&Lang);
     else strcpy(refName,"*");
     AddConceptToCG (cgSch, concept, typeName, refName);
}
```

1. La manipulation des EDs déclarés comme des choix dans un modèle structurel (e.g. un "bloc" est un choix entre un paragraphe, une image, un graphique, une formule et n'importe quel autre ED structuré) est assez délicate et surprenante pour le débutant : un tel ED doit être construit comme n'importe quel autre ED (i.e. inséré dans la structure logique), mais lorsqu'un sous-élément lui est donné (i.e. lorsqu'un choix est fait), Thot le fait disparaître de la structure logique. Ainsi, c'est depuis chaque élément qui peut intégrer un ED déclaré comme un choix, que l'existence de cet ED ou bien de l'un de ses fils doit être testé.

2. Une erreur fréquente est l'accès à un sous-élément prévu par la structure logique mais qui n'a pas encore été construit ou qui a été détruit par l'utilisateur. Une solution est de tester l'existence d'un sous-élément avant d'effectuer un accés à son contenu. Cette solution est assez lourde car ce genre d'accès est omni-présent. Une autre solution pour les cas où la structure logique n'est pas conforme à ce qu'elle est habituellement, est de laisser l'erreur se produire : les fonctions de Thot appelées détectent alors l'erreur et n'effectuent rien, ce qui est un comportement généralement adéquat.

Notons que la création d'un ED ConceptInclusion s'effectue en plusieurs temps :
1) un ED ConceptInclusion est d'abord créé par l'utilisateur (la fonction ConceptInclNewPost() est appelée),
2) l'utilisateur va rechercher l'ED Concept à inclure (si des sélections sont alors effectuées sur d'autres EDs que des ED Concept, la fonction ElemSelectPost() est appelée et rappelle à l'utilisateur qu'il doit sélectionner un ED Concept), et enfin
3) l'utilisateur sélectionne un ED Concept (la fonction ConceptSelectPost() est alors appelée pour modifier l'ED ConceptInclusion initialement créé, de manière à ce qu'il contienne une inclusion de l'ED Concept sélectionné).
Si un ED Concept est modifié, CGKAT recherche les EDs CGgraph qui incluent cet ED via un ED ConceptIclusion, et s'il en existe, modifie ces graphes dans CoGITo (la modification graphique des EDs est effectuée par Thot).

7.3.2 Quelques choix effectués

Dans CGKAT, nous avons essayé le plus possible d'exploiter l'éditeur de documents structurés Thot. Pour cela, nous avons préféré concevoir et gérer différents types d'EDs via les langages et l'interface fonctionnelle de Thot, plutôt que d'utiliser des menus ou des logiciels séparés. Notre approche est ainsi uniforme et ouverte puisque l'utilisateur peut alors utiliser les outils de l'éditeur Thot pour manipuler ces différentes informations : outils de visualisation (présentations et vues), de structuration (copies et inclusions, liens hypertextes et structurels) et de recherche (navigation et requête sur la structure et/ou sur des chaînes de caractères). La conception et l'implémentation de CGKAT a ainsi été contrainte, et donc guidée, par les possibilités et les limites de Thot.

Par exemple, dans Thot, aucune chaîne de caractères ne peut être associée (de manière pratique ou ergonomique) à un attribut référence (lien hypertexte). La sémantique du lien ne peut donc être exprimée que par son type, et seuls quelques types peuvent être prédéfinis dans des modèles structurels. Aussi nous ne pouvions envisager la construction d'un réseau hypertexte classique où les liens hypertextes sont également les liens d'un réseau sémantique. Cela nous obligeait à séparer de manière explicite les EDs indexés de leur indexations et à représenter sous forme d'EDs ces indexations, ce qui a de nombreux avantages (voir section 6.3.2) : possibilité d'effectuer des représentations précises des EDs et de leurs inter-relations et de contextualiser ou d'associer diverses informations à ces représentations, visualisation et manipulation de ces représentations, etc.

De façon similaire, pour chaque ED indexé, nous avons utilisé un ED pour contenir la liste de ses indexations car 1) grâce à cet ED, ces diverses indexations sont facilement accessibles, comparables et manipulables, et 2) dans Thot, un ED ne peut porter différents attributs de même type, ni de liste d'attributs.

Nous avons également dû effectuer d'autres choix parfois plus arbitraires et nous avons parfois dû corriger de tels choix. Voici quelques exemples.

7.3.2.1 Construction de modèles structurels

Il y a souvent plusieurs alternatives pour définir un objet dans un modèle structurel, mais certaines semblent parfois meilleures que d'autres.

7.3.2.1.1 Premier exemple : structure d'un ED TypeDefinition

Voici comment nous avons défini un ED TypeDefinition.

```
TypeDefinition (ATTR  !DefKind = NSC_for_ConceptType, NC_for_ConceptType,
                                 SC_for_ConceptType, TC_for_ConceptType,
                                 NSC_for_RelationType, Unspecified_DefKind, No_more_a_definition;
                      !Completed = not_yet, yes;   ... )
    = BEGIN  DefinedType = TEXT;     LambdaVar = TEXT;    DefBody = CGgraph;
       END  with  DefKind ?=TC_for_ConceptType,  Completed ?=not_yet;
```

L'attribut énuméré DefKind permet de spécifier la sorte de définition dont il s'agit, l'attribut Completed spécifie si la définition est complète et "..." représente tous les attributs de CGgraph. Un élément choix aurait également pu être employé pour spécifier les sortes de définition comme ci-dessous.

```
TypeDefinition = CASE OF
                      NSC_for_ConceptType (ATTR completed=no,yes; ...) = ConceptTypeDef;
                      NC_for_ConceptType (ATTR completed; ...) = ConceptTypeDef;
                      SC_for_ConceptType (ATTR completed; ...)  = ConceptTypeDef;
                      TC_for_ConceptType (ATTR completed; ...)  = ConceptTypeDef;
                      NSC_for_RelationType (ATTR completed; ...) = RelationTypeDef;
                 END;
ConceptTypeDef = BEGIN  ConceptType;    LambdaVar = TEXT;    DefBody = CGgraph;  END;
RelationTypeDef = BEGIN   RelationType;    LambdaVar;              DefBody;              END;
```

Un élément choix disparaît de la structure logique lorsque l'utilisateur a effectué son choix. Ainsi, comme nous l'avons appris à nos dépens après un certain nombre d'essais, si un élément choix porte des attributs (ce que Thot permet), non seulement ses attributs ne sont pas accessibles (puisque l'élément n'est pas accessible) mais ses attributs ne peuvent être hérités par les sous-éléments de cet élément choix, ce qui aurait pu être intéressant avec des attributs contenant des valeurs par défaut. En effet, normalement, un ED qui peut porter le même attribut qu'un de ses ancêtres mais qui dans une structure logique ne le porte pas, "hérite" de cet attribut porté par son ancêtre le plus proche (avec sa valeur), c'est-à-dire qu'il est traité par les règles de présentation comme s'il portait cet attribut. Le modèle structurel des GCs exploite ceci pour simplifier la présentation des GCs[1]. Nous avons donc renoncé à définir des attributs sur des éléments choix.

Une conséquence pour la définition de l'ED TypeDefinition ci-dessus est que l'attribut Completed et les autres attributs de CGgraph doivent être définis dans tous les sous-éléments de l'élément choix. Cela est plus un inconvénient pour le programmeur que pour l'utilisateur. Un désavantage bien plus important est que l'utilisateur doit détruire un ED TypeDefinition et le reconstruire s'il veut changer son type afin d'exprimer une autre sorte de définition. Cela pourrait être géré par programme mais nécessiterait l'usage de menus ou d'attributs supplémentaires (comme les attributs No_more_a_definition et ChangeIntoDefinition qui permettent de spécifier à CGKAT qu'un ED TypeDefinition doit être muté en un ED CGgraph et réciproquement).

1. Par exemple, si un utilisateur donne la valeur "Rounded" à l'attribut ConceptFrame d'un ED CGgraph, tous les EDs Concept contenus dans cet ED CGgraph, s'ils ne portent pas eux-même un attribut "Rounded" seront représentés entourés d'un ovale, y compris les concepts des graphes emboîtés.

Voici ci-dessous une troisième manière de définir un ED TypeDefinition. Elle emprunte un peu aux deux premières et a non seulement les même défauts que la seconde mais disperse les attributs nécessaires à la gestion d'une définition sur deux EDs différents, ce qui est assez désagréable pour l'utilisateur.

```
TypeDefinition = CASE OF   ConceptTypeDef (ATTR  completed=no,yes; ...);
                           RelationTypeDef (ATTR completed; ...)
                 END;

ConceptTypeDef (ATTR ! DefKind = NSC_for_ConceptType, NC_for_ConceptType,
                                 SC_for_ConceptType, TC_for_ConceptType,
                                 NSC_for_RelationType, Unspecified_DefKind, No_more_a_definition; ...)
               = BEGIN  ConceptType;   LambdaVar = TEXT;        DefBody;
                 END with  DefKind ?=TC_for_ConceptType;
RelationTypeDef (ATTR  ! DefKind = NSC_for_ConceptType, NC_for_ConceptType,
                                 SC_for_ConceptType, TC_for_ConceptType,
                                 NSC_for_RelationType, Unspecified_DefKind, No_more_a_definition; ...)
               = BEGIN   RelationType;   LambdaVar;              DefBody;
                 END with  DefKind ?=NSC_for_RelationType;
```

Un petit avantage des deux dernières méthodes serait de permettre de définir l'ED lambdaVar de manière différente suivant qu'il est contenu par un ED ConceptTypeDef ou un ED RelationTypeDef, cest-à-dire comme un ED textuel dans le premier cas et une liste d'EDs textuels dans le second (LambdaVar = LIST OF (Variable=Text);). Cependant, cette contrainte syntaxique peut être aisément vérifiée par programme.

7.3.2.1.2 Second exemple : structure d'un ED Relation

Voici comment avions dans un premier temps définit l'ED Relation.

```
Relation (ATTR  ...)
      = BEGIN
            RelationType (ATTR RelationFrame) = TEXT;
        END;
```

Et voici comment nous l'avons finalement défini.

```
Relation (ATTR  ...)
      = BEGIN
            LinkToOrig (ATTR ...)  = BEGIN   GRAPHICS;   ? LinkToOrig_Mark = TEXT;   END;
          ? OtherLinkToOrig = LIST OF (LinkToOrig);
            RelationType (ATTR RelationFrame) = TEXT;
            LinkToDest (ATTR ...)  = BEGIN   GRAPHICS;   ? LinkToDest_Mark = TEXT;   END;
          ? OtherLinkToDest = LIST OF (LinkToDest);
        END;
```

Dans le premier cas, les relations ne pouvaient que être binaires et le modèle de présentation dessinait une ligne droite orientée joignant les deux EDs Concept joints et positionnait l'ED RelationType au milieu de cette ligne (l'orientation de la ligne devait être gérée par programme). Cette solution s'est avérée très difficile à utiliser pour les graphes dépassant six concepts à cause de la rigidité imposée par le modèle de présentation et surtout à cause des problèmes de sélection des EDs Relation ou RelationType : lorsque des boîtes de présentation d'EDs Relation se chevauchaient, ce qui était très fréquent à partir de plus de cinq concepts, il était difficile d'arriver à sélectionner la bonne boîte d'ED Relation ou RelationType.

Pour résoudre ces problèmes, nous avons introduit une représentation explicite des liens et surtout utilisé pour cela les EDs Graphics (dans le premier cas, nous nous étions inspirés de certaines parties du modèle de présentation donné par la bibliothèque de Thot pour un arbre, tandis que dans le second cas, nous avons imité certaines parties du modèle de présentation donné pour un graphique). Les EDs

Graphics peuvent représenter différentes sortes d'éléments graphiques : rectangle, ellipse, ligne courbe orientée, ligne droite orienté, etc. Lorsqu'un ED Relation est créé, CGKAT initialise chacun des deux EDs Graphics à une ligne droite orientée. Avec le nouveau modèle de présentation, ces lignes se modifient de manière adéquate lorsque l'emplacement des boîtes de l'un des EDs Concept ou de l'ED RelationType est modifié ; l'orientation n'a plus à être gérée par programme. Ces lignes restent transformables par l'utilisateur en d'autres objets graphiques, e.g. des lignes brisées orientées dont on peut déplacer les points de rupture. Toutefois des problèmes de sélection de liens ou d'ED RelationType apparaissent lorsque les boîtes des liens se chevauchent, ce qui est très fréquent dès qu'un graphe contient une dizaine de concepts (moins d'une dizaine si les concepts sont rapprochés).

Dans cette nouvelle définition de l'ED TypeDefinition, les relations peuvent être non binaires (i.e. d'autres liens peuvent être ajoutés pour joindre l'ED RelationType à d'autres EDs Concept) et chaque lien peut recevoir une marque, mais ceci n'est pas répercuté dans CoGITo.

Notons que lorsque l'utilisateur détruit un lien ou un concept, Thot ne détruit pas la relation contenant ce lien ou les relations connectées à ce concept. Ce sont les fonctions de CGKAT appelées lors de ces deux sortes de destructions qui détruisent l'ED Relation en entier en même temps qu'elles détruisent la relation dans CoGITo.

7.3.2.2 Application des modèles d'interface

Thot offre deux modes d'application des modèles d'interface. Dans les deux cas, lorsqu'un ED d'un type X est créé, copié ou détruit, cet ED est "notifié" : si un modèle d'interface pour les EDs du type X spécifie des fonctions à appeler lors d'une création, copie ou destruction de tels EDs, ces fonctions sont appelées. Dans l'un des deux modes, lorsqu'un ED est créé, copié ou détruit, ses sous-éléments sont également "notifiés".

Il nous a semblé dans un premier temps que le mode où seul l'ED créé, copié ou détruit est notifié serait plus aisé à gérer (ce mode est d'ailleurs le mode par défaut dans Thot). Nous avons donc écrit notre code en fonction de ce mode. Ce choix a probablement simplifié la programmation de la gestion des EDs GC et de leurs composants, mais a un inconvénient que nous n'avions pas vu initialement : lorsqu'un ED contenant un ou plusieurs EDs GC est détruit ou bien copié puis collé, les fonctions de CGKAT relatives à la création ou à la destruction d'EDs GC ne sont pas appelées. Pour qu'elles le soient, toutes les fonctions de CGKAT pouvant être appelées lorsqu'un ED est détruit ou bien copié puis collé, devraient tester si un des sous-composants de l'ED est un ED GC et dans ce cas, elles devraient appeller les fonctions de CGKAT relatives à la création ou à la destruction d'EDs GC. Cela n'est que partiellement le cas actuellement. Il semble préférable à l'avenir de choisir l'autre mode et d'adapter le code de CGKAT en conséquence.

7.3.2.3 Sauvegarde des documents et des EDs GC dans les documents

Chaque GC contenu dans un document est également sauvé séparément, dans un fichier au format BCGCT dont le nom est formé par la concaténation de ".G_" et du nom du graphe. Cela permet de charger dans CoGITo certains GCs contenus dans des documents sans ouvrir ces documents (cf. section 7.2.3.1). Actuellement, ces fichiers sont également exploités par CGKAT lorsqu'il ouvre un document pour charger dans CoGITo les GCs contenus dans ce document. Il semble préférable à l'avenir que les GCs contenus dans un document, soient construits dans CoGITo directement à partir des EDs GC. Ainsi, le chargement des GCs d'un document ne sera plus dépendant de divers fichiers, ce qui permettra d'effectuer des copies ou des changements de répertoire sur ce document sans avoir à les effectuer également sur les fichiers de sauvegarde des GCs.

Notons que la sauvegarde d'un document sous un autre nom ou sous un autre répertoire que celui initial ne doit pas s'effectuer par une commande classique de copie de fichier, e.g. la commande "cp" du shell, si le fichier contient des liens hypertextes inter-documents ou des EDs GC. Elle doit alors

s'effectuer via la commande "save as" accessible depuis un éditeur Thot. En effet, s'il existe des liens hypertextes inter-documents, la commande "save as" permet à Thot de mettre à jour ces références dans divers documents. Par ailleurs, si le document contient des EDs GC, les noms de ces GCs doivent être modifiés puisqu'ils doivent normalement contenir le nom du document qui les contient. Cette mise à jour n'est pas encore implémentée.

7.4 Le langage de commandes de CGKAT

7.4.1 Construction ou référence à des GCs dans les requêtes

Depuis CGKAT, une requête peut comporter un nom de GC, un ED GC ou encore un GC au format linéaire. En effet, une première phase d'analyse de la requête permet 1) de créer de manière temporaire des GCs dans CoGITo lorsque la requête contient des EDs GC ou des GCs au format linéaire (si des méta-informations leurs sont associées, elles sont enregistrées dans CoGITo), et 2) de remplacer dans la requête les EDs GC ou les GCs au format linéaire par les noms des GCs créés. Dans CoGITo, le fait que ces GCs ne représentent pas des connaissances mais sont créés pour une requête, est stocké dans le champ Comment associé à chaque GC. A chaque exécution d'une requête, les GCs temporairement créés pour la requête précédente sont détruits.

Lorsque la requête est lancée depuis CGKAT, les GCs résultats sont affichés sous forme de GCs au format linéaire ou avec des EDs. Nous avons expliqué dans la section 7.2.3.3. comment ceci est implémenté. Lorsque la requête est lancée depuis le shell, les GCs résultats sont retournés sous forme de noms de GCs de manière à pouvoir être réutilisés par d'autres requêtes.

7.4.2 Exploitation du shell dans le langage de commandes de CGKAT

Les commandes accessibles depuis le shell peuvent être appelées depuis CGKAT en les préfixant par "sh". Les commandes du langage de commandes de CGKAT peuvent être appelées depuis le shell en les préfixant par "ck", c'est-à-dire en les passant en paramètre au programme ck. Plus précisément, nous avons écrit ce programme, séparé de CGKAT et destiné à appeler une commande de CGKAT, de manière à ce qu'il puisse utiliser les informations qui lui sont passées en paramètre et/ou[1] via son fichier standard d'entrée. ck communique la requête à CGKAT, récupère les résultats, et les affiche sur son fichier de sortie standard. On peut donc combiner ck (et donc chaque commande de CGKAT) avec d'autres programmes via les instructions de combinaison du shell dont le pipe.

Rappelons que la commande "display" prend un nom de GC en paramètre et affiche un GC au format linéaire ou avec un ED GC suivant que la commande a été appelée depuis le shell ou bien depuis CGKAT. Dans ce dernier cas cependant, si le GC n'a pas été construit avec un ED GC, la forme linéaire est utilisée (il serait intéressant qu'un ED GC puisse être généré). Voici un exemple d'une même combinaison de commandes, appelée depuis le shell et depuis CGKAT.

```
$ ck  spec [Process] | ck maxjoin | ck display
> sh ck  spec [Process] | ck maxjoin | ck display
```

Notons enfin que pour faciliter la combinaison de commandes et l'écriture de scripts, nous avons dû introduire dans CGKAT la commande "setenv" (ou "set") qui a la même fonctionnalité que l'instruction du shell portant le même nom, c'est-à-dire de permettre de créer et d'affecter une variable d'environnement (i.e. globale au processus appelés).

1. Au moins un paramètre est nécessaire pour spécifier le nom de la commande. Les paramètres de la commande peuvent être donnés sous forme de paramètres supplémentaires pour "ck" et/ou dans le fichier standard d'entrée de ck. Les informations données dans ce fichier sont considérées par ck comme des paramètres supplémentaires.

7.5 Conclusion

CGKAT combine les fonctionnalités de Thot, de CoGITo, de WordNet et du shell. De plus, l'approche client/serveur et l'exploitation du shell par CGKAT permet à d'autres applications d'exploiter les fonctions de cet outil ou de les compléter.

L'exploitation du shell nous a également évité de programmer un langage de contrôle ou de combinaison de commandes.

Par contre, si l'on ne retient de CGKAT que son aspect d'éditeur graphique pour une base de graphes conceptuels, il est probable qu'il aurait été plus aisé de construire une interface dédiée en exploitant une boîte à outils de développement d'interfaces graphiques qui inclut un grapheur (e.g. Unidraw (Vilissides, 1990) ou ILOG views (ILOG, 1993c)). En effet, outre le fait que la mise au point de notre modèle de présentation pour les EDs GC ne fut pas aisée, nous avons eu à gérer la répercussion dans CoGITo des modifications effectuées sur ces EDs GC dans Thot. Nous n'étions donc pas dans le cas de Schaar (1994) qui a préféré utiliser Thot plutôt qu'un générateur d'interface ou que Unidraw car les structures abstraites que lui fournissaient Thot correspondaient aux représentations internes dont il avait besoin pour gérer les objets de son langage.

Nous avons également souligné les difficultés de la programmation de la gestion d'informations dans une structure logique. En contre-partie, notre approche permet à l'utilisateur 1) de stocker des GCs dans des documents, 2) de les structurer ou de les associer à d'autres informations en utilisant des liens structurels ou hypertextes, 3) d'utiliser les fonctions de Thot pour les retrouver et les visualiser, et 4) d'utiliser des requêtes conceptuelles pour rechercher ou rassembler des graphes ou les informations qu'ils indexent.

Notre travail côté CoGITo a consisté à :
1) créer une interface fonctionnelle pour permettre la gestion des objets de CoGITo via des identificateurs de ces objets plutôt que par des pointeurs (cette interface fonctionnelle est exploitée par le langage de commandes de CGKAT, les menus de gestion de types ou de marqueurs individuels, et les fonctions permettant de créer des graphes ou des définitions de types via des EDs GC) ;
2) ajouter des fonctions de gestion de connaissances, e.g. la recherche de types ou de graphes, la gestion des liens d'exclusion et celles des graphes emboîtés ;
3) créer les fonctions de génération de listes indentées permettant de visualiser des portions de hiérarchies de types sélectionnées par requête, navigation ou filtrage (filtrage sur les méta-informations ou sur les signatures des types de relations)
4) gérer l'inclusion dynamique ou sur commande de types de WordNet dans le treillis des types de concepts.

Chapitre 8 Conclusion

8 Sommaire

8.1 Fonctionnalités et guides

CGKAT combine les fonctionnalités de Thot, de CoGITo, de WordNet et du "shell". Thot, CoGITo et le shell sont conçus pour offrir de nombreuses possibilités à leurs utilisateurs. CGKAT permet l'exploitation de ces possibilités (ainsi que d'autres), et ce aussi bien sur des informations que sur des connaissances. CGKAT est ainsi un ***outil générique pour faciliter l'AC et la RI***. La généricité de CGKAT, qui est entre autre due au fait de pouvoir exploiter une ontologie et représenter diverses sortes d'EDs (e.g. texte, image, ED structuré). Cette généricité n'implique pas l'absence de guide pour l'utilisateur lors de la structuration et la représentation d'informations dans des documents, lors de la structuration de connaissances, et lors de la recherche d'informations ou de connaissances. En effet :

1. CGKAT propose une *ontologie générale* de types de concepts et de types de relations, synthétisant des distinctions conceptuelles préconisées par différentes recherches sur la modélisation des connaissances et la structuration d'informations. Ces distinctions sont accompagnées de *contraintes d'utilisation*, e.g. des signatures pour les relations et des liens d'exclusion entre types. Les types de concepts de plus haut niveau sont également accompagnés de définitions de types représentant des *descriptions* (e.g. les modèles de tâches génériques associés à certains types de tâches) ou de *modèles d'utilisation* (e.g. les relations qui sont typiquement connectées à des concepts de certains types).
   L'*exploitation de WordNet* permet de fournir à l'utilisateur, lorsqu'il le souhaite, de nombreux sous-types ou super-types à des types de concepts, qu'ils appartiennent à notre ontologie générale ou qu'ils aient été rentrés par l'utilisateur. Nous avons montré que cela pouvait faciliter a) la construction de l'ontologie de l'application, b) son extension ou sa réutilisation, c) la recherche de types adéquats pour représenter des connaissances, d) la comparaison de connaissances, e) la recherche d'informations ou de connaissances (notamment, de nombreux niveaux de généralité sont permis dans les requêtes conceptuelles si les types utilisés dans les connaissances ont de nombreux supertypes).
   Les *menus et le langage de commandes* de CGKAT facilitent la recherche des distinctions conceptuelles : recherches par navigation, requêtes lexicales, requêtes conceptuelles sur les graphes utilisés dans les définitions de type, filtrage sur les méta-informations associées aux types ou aux graphes. Il existe ainsi différentes manières de générer des vues sur l'ontologie de l'application.

2. Nous avons fixé quelques *contraintes* sur ce que devait être une connaissance liée par un lien de *représentation*. Par exemple, un même utilisateur n'a le droit de construire qu'une seule représentation d'un ED pour un même point de vue. Nous avons distingué trois aspects d'un ED qui peuvent être représentés : a) l'entité ou la situation référée ou décrite par l'ED, b) la description faite par l'ED d'une situation, c) le format et le medium utilisé pour effectuer la référence ou la description. Nous avons montré comment ces différents aspects pouvaient être représentés dans le formalisme de GCs et en utilisant notre ontologie. Nous avons distingué les "EDs symboles" des

"EDs descriptions" et proposé des représentations différentes adaptées à la nature de chacun de ces EDs.
Ces contraintes sont destinées à permettre une meilleure exploitation des représentations des EDs pour la recherche de ces EDs ou la génération de documents. Pour permettre à l'utilisateur de s'affranchir de ces contraintes, nous avons introduit le lien d'**annotation**. Nous avons également permis l'exploitation d'indexation utilisant d'autres types de liens hypertextes.
Par ailleurs, pour les EDs symboles qui sont représentés par des utilisateurs, nous avons permis la génération d'index triant ces EDs symboles et les caractéristiques importantes de leurs représentations : auteur de la représentation, point de vue choisi, type de concept utilisé, source d'information, commentaire. Des liens hypertextes permettent de naviguer (dans les deux sens) entre les informations dans les index et leurs origines dans le(s) document(s) : EDs symboles ou représentations de ces EDs. De tels index offrent donc un moyen d'accès supplémentaire aux informations ou aux connaissances.

3. Nous avons montré comment l'ontologie générale de CGKAT pouvait être exploitée et complétée pour guider l'extraction ou la recherche de connaissances à modéliser (cf. section 5.7), et nous avons proposé des éléments méthodologiques d'utilisation des fonctionnalités de CGKAT pour réaliser l'extraction et la modélisation de connaissances, d'une manière ascendante ou descendante (cf. section 6.4).

8.1.1 Fonctionnalités de CGKAT

Les fonctionnalités de CGKAT peuvent être classées en sept catégories :

1. ***Gestion d'une ontologie***. CGKAT permet de créer et de modifier sous un format graphique ou textuel, une ontologie composée d'un treillis[1] de types de concepts, d'une hiérarchie de types de relations, et d'une liste de marqueurs individuels. Des commentaires et des définitions par conditions nécessaires et/ou suffisantes ou typiques peuvent être associés aux types et, pour les types de relations, des signatures de relations. Nous avons également permis la pose de liens d'exclusion entre types, ainsi que l'association de méta-informations à des types de manière à permettre une gestion des points de vue complémentaire à celle qui est effectuée via les relations de spécialisation entre types (cf. section 5.2.2).
   CGKAT permet de visualiser des parties de hiérarchies de types sous forme de listes indentées ou sous forme graphique. Dans les menus de gestion des types de concepts ou de relations, la recherche de types et de leurs supertypes et sous-types peut s'effectuer par navigation ou requête lexicale. Parallèlement, les types à afficher peuvent être filtrés suivant les méta-informations qui leur sont associées. Cela devrait faciliter la construction d'une ontologie suivant différents points de vue, e.g. différents experts sources ou domaines sources. En annexe 2, nous spécifions des protocoles destinés à faciliter la construction d'une même hiérarchie de types par plusieurs utilisateurs.

2. ***Proposition d'une ontologie générale.*** Nous avons organisé les types de haut niveau de WordNet dans une ontologie de celle proposée par Sowa (1992) et nous les avons complété par d'autres distinctions conceptuelles provenant notamment de Sowa (1995b), Tepfenhart (1992), Skuce (1995), Bateman (1990). Nous avons introduit d'autres distinctions utiles pour faciliter la modélisation, e.g. les notions de "type de vue", "type de domaine", et "type de concept utilisé par un agent". Nous avons repris les types d'inférences, de rôles et de tâches génériques de KADS-I et CommonKADS et représenté les modèles de ces tâches génériques sous forme de définitions par conditions typiques. Nous avons également associé de telles définitions aux types de concepts de plus haut niveau afin de représenter les relations généralement connectées aux concepts ayant pour type l'un de ces types de haut niveau ou l'un de leur sous-types.
   Il existe environ 200 types de concepts de haut niveau dans l'ontologie générale proposée par

1. CGKAT ne vérifie pas que le "treillis" des types de concepts a bien une structure de treillis. CGKAT vérifie simplement que la structure est un ordre partiel.

CGKAT, dont 35 types représentant des catégories conceptuelles de haut niveau dans WordNet. Ces 35 types sont "virtuellement" spécialisées par les 90.000 types de concepts qui peuvent être extraits de WordNet. En effet, la base WordNet est exploitée de telle sorte que les types de WordNet puissent être inclus temporairement ou définitivement dans le treillis des types de concepts et classés par rapport aux types qui y existent déjà. Dans le treillis, l'organisation de ces types peut alors être complété ou modifié par l'utilisateur, ces modifications étant prises en compte par le mécanisme d'inclusion des types de WordNet.
Nous avons enfin rassemblé et organisé des types de relations élémentaires, notamment en nous inspirant des ontologies de types de relations proposés par Sowa (1984), Sowa(1992), Myaeng & Koo (1994), Mann & Thompson (1988), Schuler & Smith (1990), Allen (1985), KOD (Vogel, 1988), CommonKADS (Breuker & van de Velde, 1994), Vilnat (1992), CYC (Lenat & Guha, 1990a), Bateman & al (1996), Suthers (1989), Acker & al. (1994), Dieng (1991) et Labidi (1995). Cette ontologie contient environ 200 types de relations : relations casuelles, rhétoriques, spatiales, temporelles, mathématiques, attributives, méréologiques, procédurales, etc. Les signatures de ces relations utilisent les types de concepts de haut niveau cités plus haut.

3. ***Construction de documents et d'une base de GCs via l'éditeur Thot.*** L'utilisateur de CGKAT peut utiliser l'éditeur Thot pour créer des documents structurés, i.e. des EDs reliés par des liens de composition, des liens d'inclusion ou des liens hypertextes. Il bénéficie pour cela des fonctions de Thot : gestion de diverses sortes de vues, zoom, génération d'index, etc. Il peut naviguer sur les liens de composition, les liens d'inclusion et les liens hypertextes, ou rechercher des EDs sur leurs types ou ceux de leurs attributs.
   Nous avons écrit un modèle structurel et un modèle de présentation pour permettre la création d'EDs GC dans Thot et nous avons utilisé les interfaces fonctionnelles de Thot et de CoGITo pour que la création d'EDs GC, ou l'ouverture d'un document contenant de tel EDs, entraîne automatiquement la création des GCs correspondants dans la base de CoGITo. L'utilisateur peut ainsi créer des GCs (dont des définitions de type) via Thot dans des documents et les organiser avec des informations ou d'autres GCs par des liens de composition, des lien d'inclusion ou des liens hypertextes.
   Outre la création de GCs conformes au modèle de base du formalisme des GCs, nous avons permis la création d'une certaine forme de GCs emboîtés qui conduit l'utilisateur à expliciter les relations entre ses concepts. Des méta-informations peuvent également être associées aux GCs.

4. ***Indexation d'informations par des connaissances.*** Un même ED peut être indexé avec un GC par plusieurs utilisateurs et selon plusieurs points de vue. L'indexation s'effectue via un lien hypertexte. Deux liens principaux sont proposés : le lien de ***représentation*** auquel des contraintes ont été associées, et le lien d'***annotation***. L'utilisateur peut naviguer entre les EDs GC via les liens qui les relient (liens de composition, liens d'inclusion ou liens hypertextes) et naviguer entre ces EDs GC et les EDs qu'ils indexent. La recherche d'informations "via les connaissances" peut donc s'effectuer par navigation.
   Par ailleurs, un index peut être généré pour synthétiser les caractéristiques importantes des représentations des EDs symboles et fournir un moyen d'accès supplémentaire aux informations ou aux connaissances.

5. ***Des requêtes conceptuelles pour rechercher des connaissances et des informations.*** Une requête conceptuelle dans CGKAT correspond actuellement à la recherche des spécialisations ou des généralisations d'un GCS requête. Cependant, si un GCS emboîté dans un graphe qui le "contextualise" constitue une réponse à une requête, CGKAT retourne le graphe qui emboîte et contextualise le GCS (un GCS contextualisé n'est ainsi pas présenté sans son contexte).
   La requête s'effectue dans la base de CoGITo mais pour chaque GC réponse à la requête, CGKAT peut présenter 1) ce GC au format linéaire, ou 2) l'ED GC avec lequel le GC a été construit, ou 3) le ou les EDs indexés par le GC (les liens de représentation ou les liens d'annotation ou d'autres types liens hypertextes peuvent être exploités).
   Lorsque des EDs doivent être présentés en réponse à une requête, des inclusions des EDs sources

sont utilisés. Ainsi, le résultat d'une requête est un document virtuel ou encore une vue sur des parties de documents sélectionnés avec des critères conceptuels. La vue est automatiquement modifiée si les EDs sources sont modifiés, et des liens hypertextes permettent de retrouver ces EDs sources. Une requête peut ainsi être utilisée pour restreindre l'espace de recherche par navigation : recherches par requête et par navigation peuvent être combinées.
Un graphe requête peut être accompagné de contraintes supplémentaires sous la forme de méta-informations. Une relation de spécialisation a été définie sur les méta-informations. La recherche peut s'effectuer parmi les GCs de la base de faits ou ceux utilisés dans les définitions de types. Nous avons également permis l'usage de noms approchés pour les noms de types dans les graphes requête. Si de tels noms approchés sont utilisés, CGKAT peut générer puis exécuter plusieurs graphes requêtes. Nous avons enfin étudié la manière dont la syntaxe des requêtes conceptuelles pourraient être étendue pour permettre la recherche dans des GCs de sous-graphes contenant des séquences (répétition de chaînes de concept et de relation).

6. ***Des commandes pour manipuler des connaissances.*** Nous avons permis la construction d'une ontologie et d'une base de GCs via un langage de commandes. Avec ce langage, des GCs peuvent également être comparés ou bien recherchés (ce sont les requêtes conceptuelles décrites ci-dessus). Des GCs peuvent également être créés avec une jointure simple dirigée et une jointure maximale non dirigée.

7. ***Un langage de combinaison de commandes.*** Nous avons permis l'appel depuis le shell des commandes de CGKAT et l'appel depuis CGKAT des commandes accessibles depuis le shell. Les instructions du shell (tests, boucles, "pipe") peuvent alors être utilisées pour combiner les commandes de CGKAT avec d'autres commandes accessibles depuis le shell. Des scripts shell peuvent alors être écrits pour effectuer des vérifications dans la base, répondre à des questions habituelles, générer des parties de documents, ou permettre l'exploitation de CGKAT par d'autres applications. Une expérience en ce sens a été effectuée : l'échange de données entre CoKaCe (Corby & Dieng, 1996) et CGKAT.
Nous avons implémenté les liens virtuels, i.e des liens hypertextes dont la destination est un ED généré par une requête, en permettant le déclenchement automatique d'une requête ou d'un script.

8.2 Comparaison avec d'autres outils d'AC ou de RI

CGKAT est un outil d'AC et de RI. Il permet l'utilisation combinée des fonctionnalités de Thot, un système de gestion d'information, et de celles de CoGITo un système de gestion de connaissances. Cependant, CGKAT n'inclut aucun outil permettant d'effectuer une indexation automatique des EDs. Par ailleurs, il ne prend pas en compte les critères de précision et de rappel. Il a donc peu de similitude avec les systèmes de recherche documentaire par requête, même ceux qui exploitent le formalisme des GCs, e.g. DR_LINK (Myaeng, 1992), ELEN (Chevallet, 1992).
CGKAT est plutôt à comparer avec les systèmes hypertextes, notamment ceux de la seconde génération comme MORE (Lucarella & al., 1993), Macweb (Nanard & Nanard, 1991) et HyperPATH/O2 (Amann, 1994) qui permettent de représenter des connaissances et d'effectuer des recherches sur ces connaissances par navigation et requête. Nous comparons donc ci-après CGKAT à d'autres systèmes hypertextes ou outils d'AC.

8.2.1 Comparaison par rapport aux systèmes hypertextes

Les ***points forts*** de CGKAT par rapport aux systèmes hypertextes sont les suivants.

1. *Des documents structurés peuvent être créés et manipulés, et des EDs structurés peuvent être indexés.* Certains systèmes hypertextes comme Guide (Brown, 1991) gère cependant les notions d'emboîtement des EDs et d'héritage entre EDs de manière similaire aux systèmes de gestion de

documents structurés. Thot (et donc CGKAT) a d'autres fonctions intéressantes comme le zoom ou la gestion de diverses sortes de vues.

2. *Les connaissances peuvent être gérées de la même manière que les informations.* Cela permet à l'utilisateur d'associer des connaissances à des informations de manière souple, en utilisant des liens de composition ou des liens hypertextes. A notre connaissance, dans les autres systèmes hypertextes, les connaissances ne peuvent être mélangées dans un document avec d'autres informations.

3. *Un même ED peut être indexé par différents utilisateurs et selon différent points de vue.* C'est également, à notre connaissance, une originalité de CGKAT. Par ailleurs, nous avons différencié le lien de représentation du lien d'annotation car nous avons associé des contraintes aux connaissances indexant un ED par un lien de représentation.

4. *Des connaissances peuvent être représentées et recherchées par requête conceptuelle, y compris des connaissances contextualisées.* Des systèmes hypertextes de la seconde génération, e.g. MORE, MacWeb et HyperPATH/O2, permettent également de représenter des connaissances dans un réseau sémantique et de rechercher des portions de ce réseau avec des requêtes conceptuelles. Ce réseau est généralement sur un seul niveau (pas de graphe emboîté dans les noeuds).

5. *Des "vues" ou "documents virtuels" sont créés en réponse à des requêtes conceptuelles.* MacWeb permet de créer de tels documents virtuels mais ceux-ci contiennent uniquement des informations et non des connaissances. La génération de tels documents constitue un mécanisme d'aide à la navigation par "restriction de l'espace de recherche".

6. *Une ontologie générale est fournie.* 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).

7. *Des commandes de recherche et de manipulation de connaissances peuvent être utilisées et être combinées avec des commandes externes à CGKAT.* Nous avons montré dans la section 6.3.2.2.6.1 comment des commandes du shell pouvaient être utilisées pour simuler des opérateurs modaux du langage de requête de Beeri & Kornatzky (1990).
   Dans CGKAT, les requêtes conceptuelles sont des recherches de graphes. Par contre, lors des requêtes conceptuelles dans les autres systèmes hypertextes, le réseau est plutôt considéré comme un seul graphe et la recherche est donc une recherche de sous-graphes. Ces deux approches nous semblent complémentaires.

Voici des ***points faibles*** de CGKAT par rapport à certains autres systèmes hypertextes :

1. *Pas d'extraction automatique de connaissances.* Par exemple, dans MacWeb certains concepts et certaines relations entre ces concepts peuvent être extraits à partir d'un texte en français. Pour pallier à cela, CGKAT devrait exploiter un analyseur de langage naturel générant des GCs.

2. *Pas d'aide à la navigation par un mécanisme de retour (exploitation d'un historique de la navigation) ou par la visualisation globale du réseau hypertexte* comme dans Intermédia (Utting & Yankelovich, 1989). Il est prévu qu'un mécanisme de retour soit implémenté dans une version utltérieure de Thot.
   Par ailleurs, la conception de "chemins de consultation" ou "visites guidées" (Zellweger, 1989) (Stott & Furuta, 1989) (Guinan & Smeaton, 1992) en utilisant des scripts ne semble pas aisée.

3. *Pas de recherche de chemins contenant des séquences dans les graphes de la base*, comme par exemple avec GraphLog (Consens & Mendelzon, 1990), MacWeb et HyperPATH/O2.

4. *Pas de commande d'édition de document structuré dans le langage de commandes.* MacWeb possède un langage de script permettant de générer des documents ayant une certaine structure. Les documents générés avec le langage de commandes de CGKAT ne peuvent contenir que du texte et des inclusions d'EDs existants dans d'autres documents. Cependant, des travaux s'effectuent actuellement dans l'équipe OPERA sur des commandes d'édition de document structuré (Bois & Richy, 1996). Il serait intéressant d'inclure ces commandes au langage de commandes de CGKAT afin de permettre la génération ou la modification de documents structurés.

Rappelons que pour permettre la représentation et la recherche de connaissances dans CGKAT, il aurait été possible d'exploiter un outil similaire à CoGITo mais basé sur un langage de représentation terminologique du type KL-ONE (Brachman & Schmolze, 1985) (cf. section 4.4.2.1) au lieu du formalisme des GCs. En termes de fonctionnallités, la différence essentielle serait venue du fait que dans de tels langages de représentation terminologiques, les connaissances rentrées par l'utilisateur sont généralement automatiquement classées dans un unique réseau sémantique. Aussi, une recherche des spécialisations d'un graphe requête aurait permis de retrouver une portion de ce réseau sémantique et non les graphes tels qu'ils sont rentrés par l'utilisateur. Ces deux sortes de solution pouvant être intéressantes, nous avons proposé dans la section 6.3.2.2.6 une manière d'étendre la syntaxe des graphes requêtes dans le langage de commandes de CGKAT afin de permettre la recherche de sous-graphes (dont des séquences) dans un graphe.

8.2.2 Comparaison par rapport aux outils d'AC

Nous avons cité sept points forts de CGKAT par rapport aux systèmes hypertextes. Ces points sont également des points forts par rapport aux outils d'AC car :

1. 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 recherche ou de gestion d'informations. Seuls des liens hypertextes entre certains types d'informations ou de connaissances sont généralement fournis ainsi que des index (c'est par exemple le cas dans la K-Station (Albert & Vogel, 1989) et dans Kads-Tool (Albert & Jacques, 1993)). Notre approche semble intéressante dans un contexte d'AC car 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.

2. Les outils d'AC actuels ne permettent généralement pas non plus de rechercher des connaissances avec des requêtes conceptuelles du type "recherche des spécialisations d'un graphe requête" ou "recherche de chemins dans des graphes".

3. Ils proposent souvent des bibliothèques de modèles de tâches génériques mais plus rarement des ontologies du domaine. Cependant Krest (Steel, 1993), Dids (Runkel & Birmingham, 1994) et Protégé-II (Puerta & al, 1992) offrent à la fois de telles bibliothèques et de telles ontologies.
   CGKAT fournit une ontologie générale qui interclasse des distinctions provenant de divers travaux en modélisation des connaissances et qui inclut des types relatifs aux éléments d'un modèle de tâches (e.g. tâche, inférence, méthode, but, rôle et les types des relations qui peuvent les relier). Quelques modèles sont associés à certains types. De plus, une originalité de CGKAT est d'exploiter une base générale de connaissances terminologiques (WordNet).

Par ailleurs, les outils ou les langages d'acquisition de connaissances incluent souvent "en dur" des distinctions conceptuelles, principalement pour la représentation des modèles de tâche. Par exemple, CommonKADS CML (Schreiber & al., 1994) incluent en dur les notions de tâche, de sous-

tâche, de méthode, d'inférence et de rôle.
L'utilisateur de ces outils (e.g. la K-Station et Kads-Tool) ou de ces langages est alors guidé par leurs menus ou leurs structures mais les distinctions conceptuelles qu'ils intègrent ne sont pas représentées par des types dans une ontologie où elles pourraient être complétées, spécialisées ou organisées avec d'autres types de cette ontologie. L'outil ou le formalisme peut ainsi être limité à la description de certaines sortes de connaissances. La comparaison des connaissances peut également être limitée du fait que les distinctions conceptuelles intégrées à l'outil ou au formalisme ne sont pas organisées entre elles ou par rapport aux autres types de l'ontologie.

Au contraire, CGKAT exploite le formalisme des GCs, lequel n'inclut que très peu de primitives épistémologiques (principalement : concept, relation, type de concept ou de relation, et référent) mais permet l'exploitation d'une ontologie. Dans CGKAT, l'ontologie de l'application peut être construite par extension ou modification de notre ontologie générale. L'utilisateur peut ajouter d'autres distinctions ou modèles qu'il souhaite utiliser. Il peut ainsi suivre différentes méthodologies de modélisation des connaissances. CGKAT exploite les distinctions de l'ontologie et les contraintes qui leur sont associées pour effectuer des *contrôles de cohérence* sur les connaissances construites. Par ailleurs, l'utilisateur peut rechercher par navigation et requête lexicale ou conceptuelle les modèles qu'il désire spécialiser pour représenter des connaissances. Enfin, le langage de commandes de CGKAT lui permet de combiner des modèles et d'écrire des requêtes conceptuelles ou des scripts pour vérifier l'*achèvement* de la modélisation c'est-à-dire a) si des connaissances ont été modélisées pour tous les types de connaissances contenues dans les modèles suivis, et b) si pour chacun de ces types, toutes les connaissances nécessaires ont été modélisées.

CGKAT est ainsi à rapprocher de Protégé-II et de Cue (Heijst & Schreiber, 1994) qui sont suffisamment génériques pour pouvoir exploiter des modèles de tâches (même adaptés par l'utilisateur) afin de *guider* l'extraction et la modélisation des connaissances du domaine.

Cependant, comme un certain nombre de systèmes d'acquisition de connaissances (e.g. Kads-Tool), CGKAT est centré sur la modélisation de connaissances et n'intègre pas de système permettant d'exécuter des connaissances dynamiques, e.g. via un système d'inférence exploitant des règles. Un tel système d'inférence pourrait être apporté à CGKAT via les futures extensions de CoGITo. Il permettrait également de définir sur les types de l'ontologie, des contraintes d'utilisation complémentaires à celles qui peuvent actuellement être définies dans l'ontologie, et donc d'effectuer des contrôles de cohérence plus importants.

La table 2.1 à la page 23 permet de comparer CGKAT à divers autres outils d'AC selon différents types de supports qui peuvent être offerts par ces outils pour aider aux activités de modélisation de connaissances.

La version actuelle de CGKAT a été achevée il y a peu de temps et n'a donc pu déjà être utilisée par un cogniticien. Cependant, une version précédente l'a été sans toutefois que les fonctionnalités d'indexation de documents ou de recherche d'information aient pu être exploitées (cf. (Alpay, 1996) pour plus de détails sur cette utilisation). Cette utilisation a néammoins mis en évidence l'intérêt d'introduire dans les fonctionnalités de gestion d'ontologies, une gestion de points de vues basée sur des "méta-informations" associées aux types de ces ontologies (cf. section 5.2.2.2).

8.3 Perspectives

De nombreuses extensions pourraient être apportées à CGKAT.

Certaines extensions pourraient être réalisées en permettant l'exploitation dans CGKAT des futures extensions de Thot, CoGITo et WordNet. Citons à titre d'exemple les thèmes de recherche actuels sur ou à partir de ces outils :

1. à partir de Thot (cf. OPERA (1996)) : transformation de structures, outils d'édition, édition de documents sur le World Wide Web, présentation des documents structurés, édition coopérative, structure temporelle des documents multimedia, langage de commandes d'édition structurée, édition de feuille de style, analyse génétique ;
2. à partir de CoGITo (cf. LIRMM (1996)) : étude d'un SGDB de Graphes Conceptuels, construction d'une ontologie, interrogation de la base de connaissances, bases de données relationnelles et graphes conceptuels ;
3. WordNet sera étendu pour prendre en compte plusieurs langues : dans des prochaines versions de WordNet - International WordNet (Sutcliffe, 1995) ou EuroWordNet (Vossen, 1996) - chaque distinction conceptuelle sera connectée aux racines de mots dont l'un des sens correspond à cette distinction, ces racines de mots n'appartenant plus seulement à la langue anglaise mais également à d'autres langues comme l'allemand, l'espagnol et éventuellement le français et l'italien.

Des extensions plus directement reliées à notre travail pourraient être :
- la construction et la recherche de portions de hiérarchies de types via des documents Thot ;
- l'indexation d'EDs par des portions de graphes au lieu de graphes entiers ;
- la recherche par requête conceptuelle de sous-graphes (dont des séquences) dans des graphes.